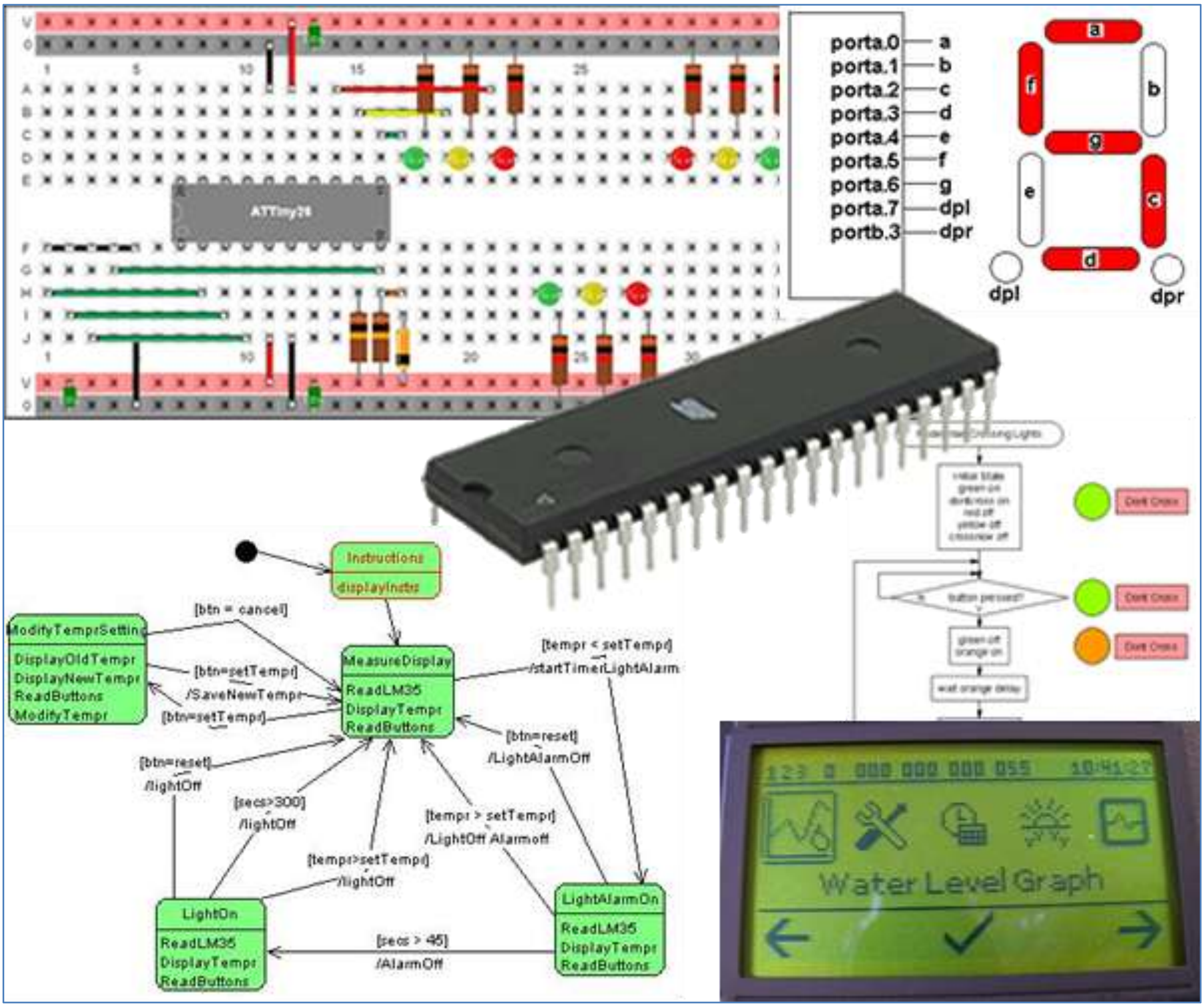
An Introduction to Practical Electronics, Microcontrollers and Software Design



2nd edition *B.Collis*

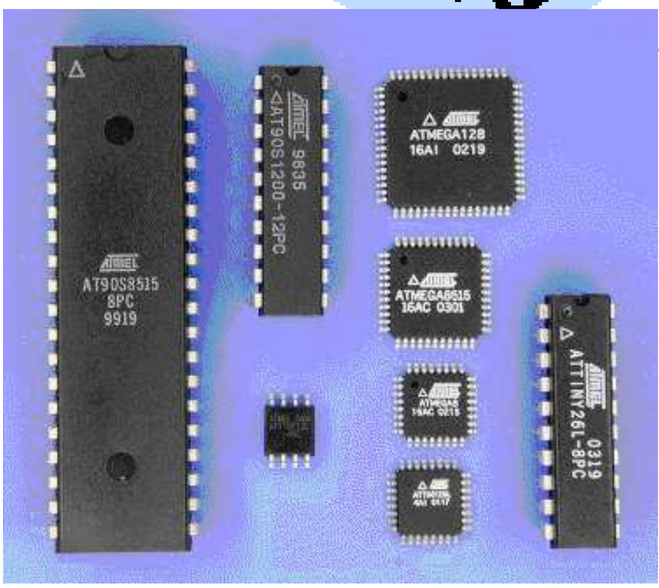
www.techideas.co.nz

Table of Contents

1 Introduction to Practical Electronics

This book has a number of focus areas.

- Electronic component recognition and correct handling
- Developing a solid set of conceptual understandings in basic electronics.
- Electronic breadboard use
- Hand soldering skills
- Use of Ohm's law for current limiting resistors
- The voltage divider
- CAD PCB design and manufacture
- Microcontroller programming and interfacing
- The transistor as a switch
- Power supply theory
- Motor driving principles and circuits
- Modelling solutions through testing and trialing
- Following codes of practice
- Safe workshop practices



1.1 Your learning in Technology

1.1.1 Technology Achievement Objectives from the NZ Curriculum

Technological Practice

Brief –develop clear specifications for your technology projects.

Planning – thinking about things before you start making them and using drawings such as flowcharts, circuit diagrams, pcb layouts, statecharts and sketchup plans while working.

Outcome Development – trialling, testing and building electronic circuits, designing and making PCBs, writing programs for microcontrollers.

Technological Knowledge

Technological Modelling – before building an electronic device, it is important to find out how well it works first by modelling and/or trialling its hardware and software.

Technological Products – getting to know about components and their characteristics.

Technological Systems - an electronic device is more than a collection of components it is a functioning system with inputs, outputs and a controlling process.

Nature of Technology

Characteristics of Technological Outcomes – knowing about electronic components especially microcontrollers as the basis for modern technologies.

Characteristics of Technology – electronic devices now play a central role in the infrastructure of our modern society; are we their masters, how have they changed our lives?

1.2 Key Competencies from The NZ Curriculum

Thinking – to me the subject of technology is all about thinking. My goal is to have students understand the technologies embedded within electronic devices. To achieve this students must actively enage with their work at the earliest stage so that they can construct their own understandings and go on to become good problem solvers. In the beginning of their learning in electronics this requires students to make sense of the instructions they have been given and search for clarity when they do not understand them. After that there are many new and different pieces of knowledge introduced in class and students are given problem solving exercises to help them think logically. The copying of someone elses answer is flawed but working together is encouraged. At the core of learning isbuilding correct conceptual models and to have things in the context of the 'big picture'.

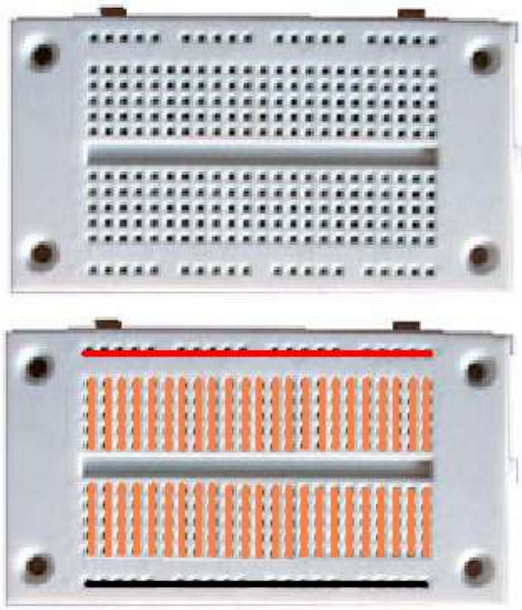
Relating to others – working together in pairs and groups is as essential in the classroom as it is in any other situation in life; we all have to share and negotiate resources and equipment with others; it is essential therefore to actively communicate with each other and assist one other.

Using language symbols and texts – At the heart of our subject is the language we use for communicating electronic circuits, concepts, algorithms and computer programming syntax; so the ability to recognise and using symbols and diagrams correctly for the work we do is vital.

Managing self – This is about students taking personal responsibility for their own learning; it is about challenging students who expect to read answers in a book or have a teacher tell them what to do. It means that students need to engage with the material in front of them. Sometimes the answers will come easily, sometimes they will not; often our subject involves a lot of trial and error (mostly error). Students should know that it is in the tough times that the most is learnt. And not to give up keep searching for understanding.

Participating and contributing – We live in a world that is incredibly dependent upon technology especially electronics, students need to develop an awareness of the importance of this area of human creativity to our daily lives and to recognise that our projects have a social function as well as a technical one.

2 An introductory electronic circuit

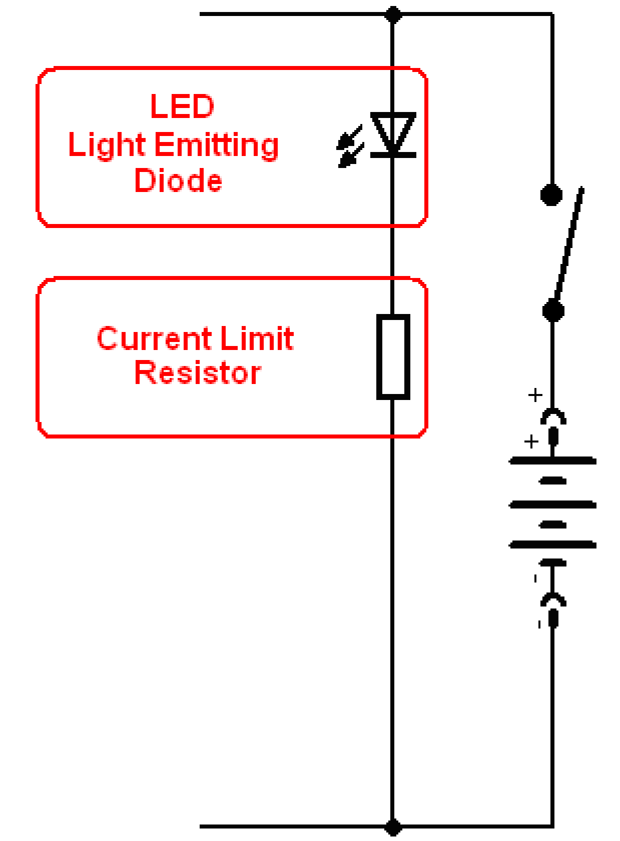
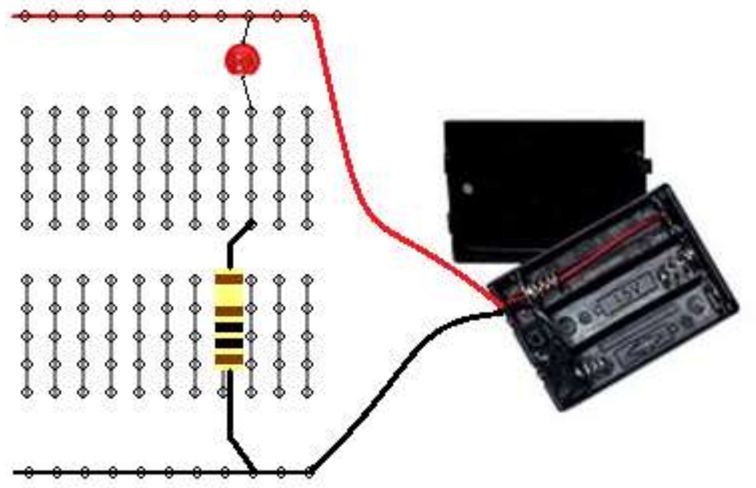


2.1 Where to buy stuff?

In New Zealand there are a number of reasonably priced and excellent suppliers for components including www.surplustronics.co.nz and www.activecomponents.com Overseas suppliers I use include www.digikey.co.nz, www.sparkfun.com ebay.com & aliexpress.com

A breadboard is a plastic block with holes and metal connection strips inside it to make circuits. The holes are arranged so that components can be connected together to form circuits. The top and bottom rows are usually used for power, top for positive which is red and the bottom for negative which is black.

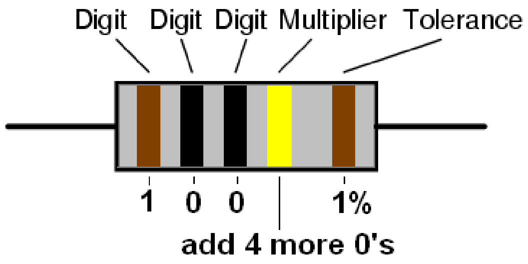
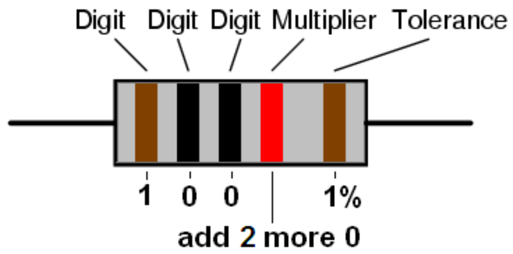
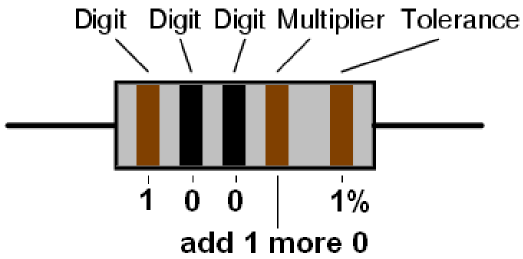
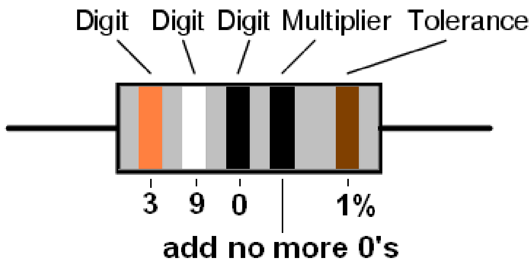
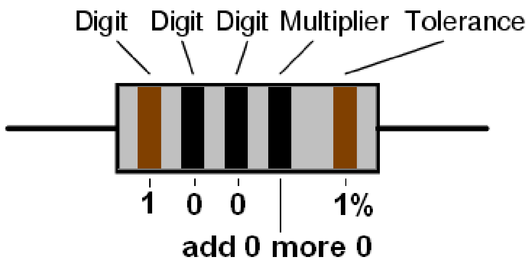
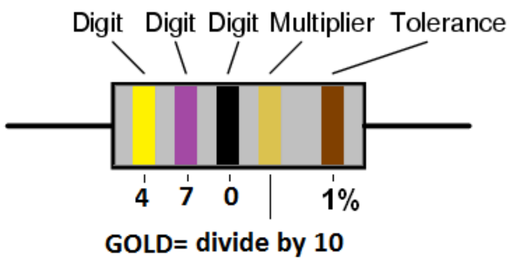
This circuit could be built like this, note that the LED must go around the correct way. If you have the LED and resistor connected in a closed circuit the LED should light up.

Schematic or circuit diagram	Layout
	

The LED requires 2V the battery is 9V, if you put the LED across the battery it would stop working! So a 1k (1000ohm) resistor is used to reduce the voltage to the LED and the current through it, get a multimeter and measure the voltage across the resistor, is it close to 7V?
If you disconnect any wire within the circuit it stops working, a circuit needs to be complete before electrons can flow.

2.2 Identifying resistors by their colour codes

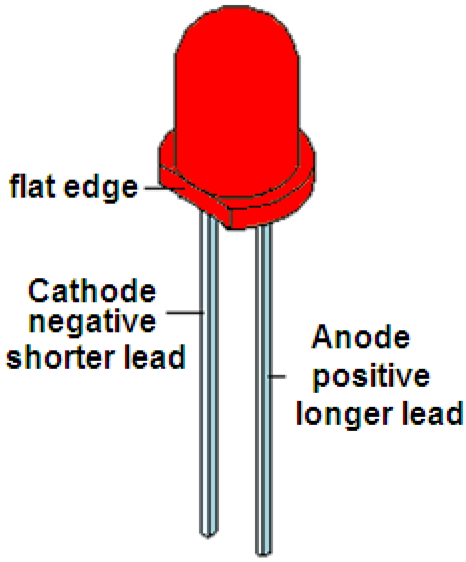
When getting a resistor check its value! In our circuits each resistor has a special pupose, and the

	1M '1 Meg' 1 Million Ohms 1M Ω 1,000,000 ohms
	10k 10 thousand ohms 10,000 ohms 10k Ω
	1k 1 thousand ohms 1,000 ohms 1k Ω
	390R 390 ohms 390Ω
	100R 1000 ohms 100Ω
	47R 47 ohms 47Ω

value is chosen depending on whether we want more or less current in that part of the circuit,The higher the value of the resistor the lower the currentThe lower the value of the resistor the higher the current.

2.3 LED's

Light Emitting Diodes are currently used in indicators and displays on equipment, however they are becoming used more and more as replacements for halogen and incandescent bulbs in many different applications. These include vehicle lights, traffic signals, outdoor large TV screens.



Compared to incandescent bulbs (wires inside glass bulbs that glow), LEDs give almost no heat and are therefore highly efficient. They also have much longer lives e.g. 10 years compared to 10 months. So in some situations e.g. traffic signals, once LEDs are installed there can be significant cost savings made on both power and maintenance. There is a small problem with LED traffic lights though – they don't melt snow that collects on them!!!

2.4 Some LED Specifications

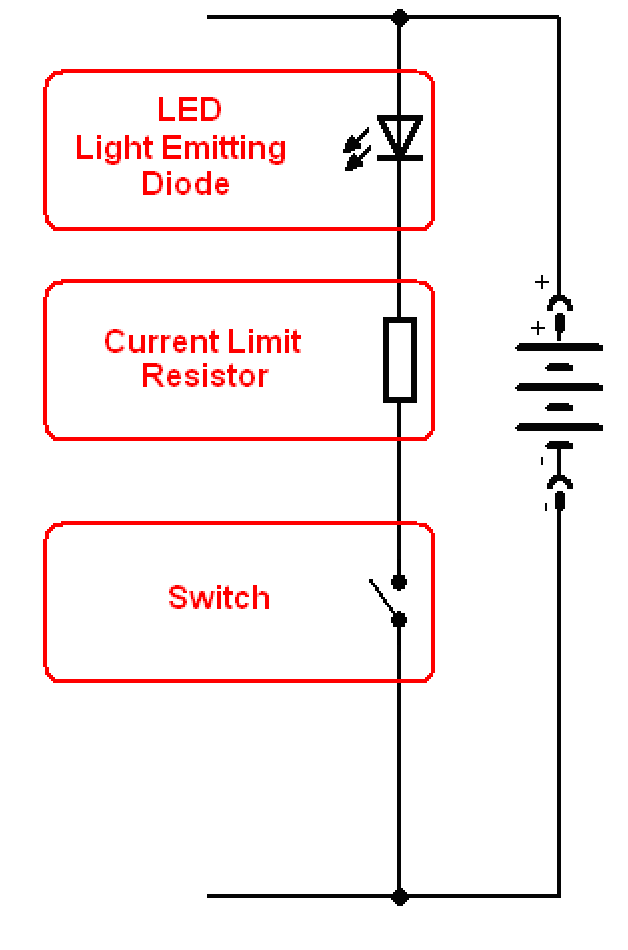
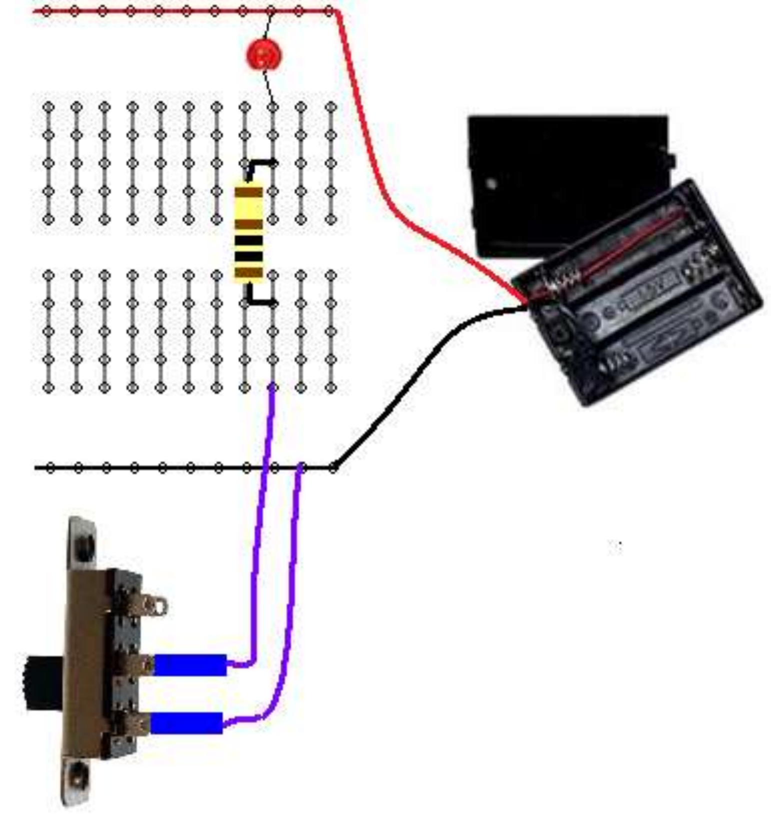
- Intensity: measured in mcd (millicandela)
- Viewing Angle: The angle from centre where intensity drops to 50%
- Forward Voltage: Voltage needed to get full brightness from the the LED
- Forward Current: Current that will give maximum brightness,
- Peak Wavelength: the brightest colour of light emitted

2.5 LED research task

From a supplier in New Zealand (e.g. Surplustronics, DSE, Jaycar, SICOM) find the information and the specifications / attributes for two LEDs, a normal RED 5mm LED and a 5mm high intensity LED.

LED	RED 5mm	High intensity 5mm
Supplier		
Part number		
Cost ($)		
Brightness (mcd)		
Forward voltage (Vf)		
Wavelength (nm)		
Forward current (If)		

2.6 Adding a switch to your circuit

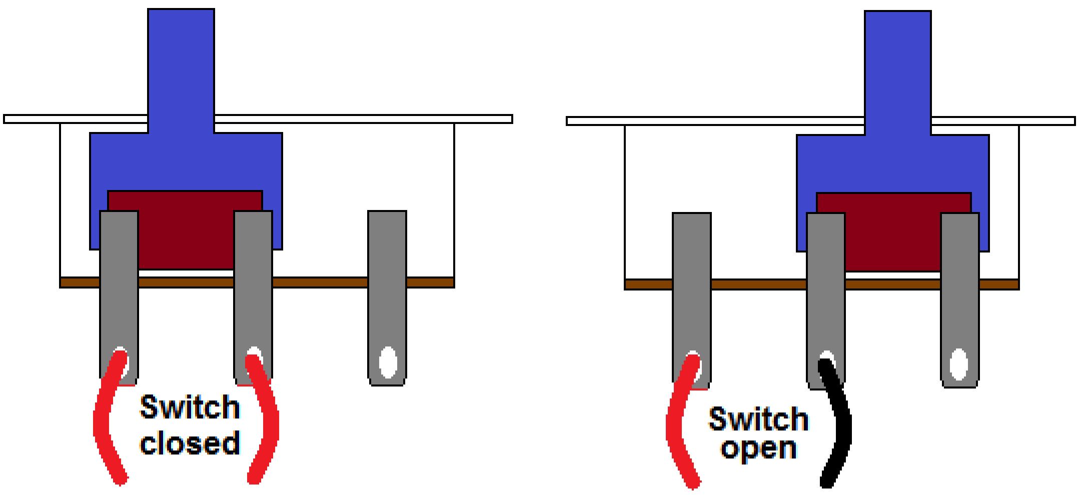
Schematic or circuit diagram	Layout
	

A switch is the way a user can manually control a circuit

2.7 Switch assignment

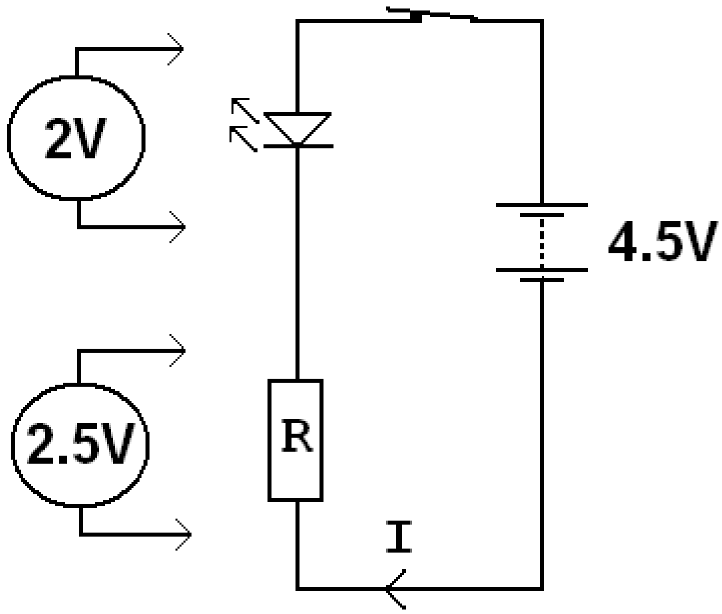
Find a small switch and carefully disassemble it (take it apart) draw how it works and explain its operation. Make sure you explain the purpose of the spring(s).

Here are simplified drawings of a small slide switch when it is in both positions.
When the switch is on electricity can flow, when it is open the circuit is broken.



2.8 Important circuit concepts

A circuit consists of a number of components and a power supply linked by wires.



<u>Electrons</u> (often called <u>charges</u>) flow in a circuit; however unless there is a <u>complete circuit</u> (a closed loop) no electrons can flow.

<u>Voltage</u> is the measure of energy in a circuit, it is used as a measure of the energy supplied from a battery **or** the energy (voltage) across a part of a circuit.

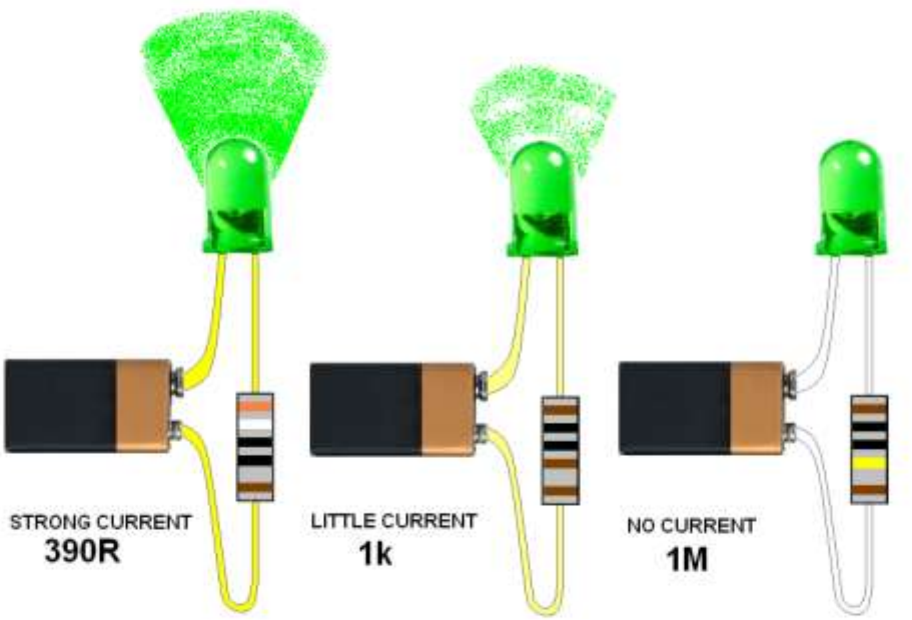
<u>Current (I) is the flow of electrons</u> from the battery around the circuit and back to the battery again. Current is measured in Amps (usually we will use milliamps or mA).
Note that <u>current doesn't flow</u> **<u>electrons or charges flow</u>**. Just like in a river the current doesn't flow the water flows.

<u>Resistance</u> works to <u>reduce current</u> , the resistors in the circuit offer resistance to the current.

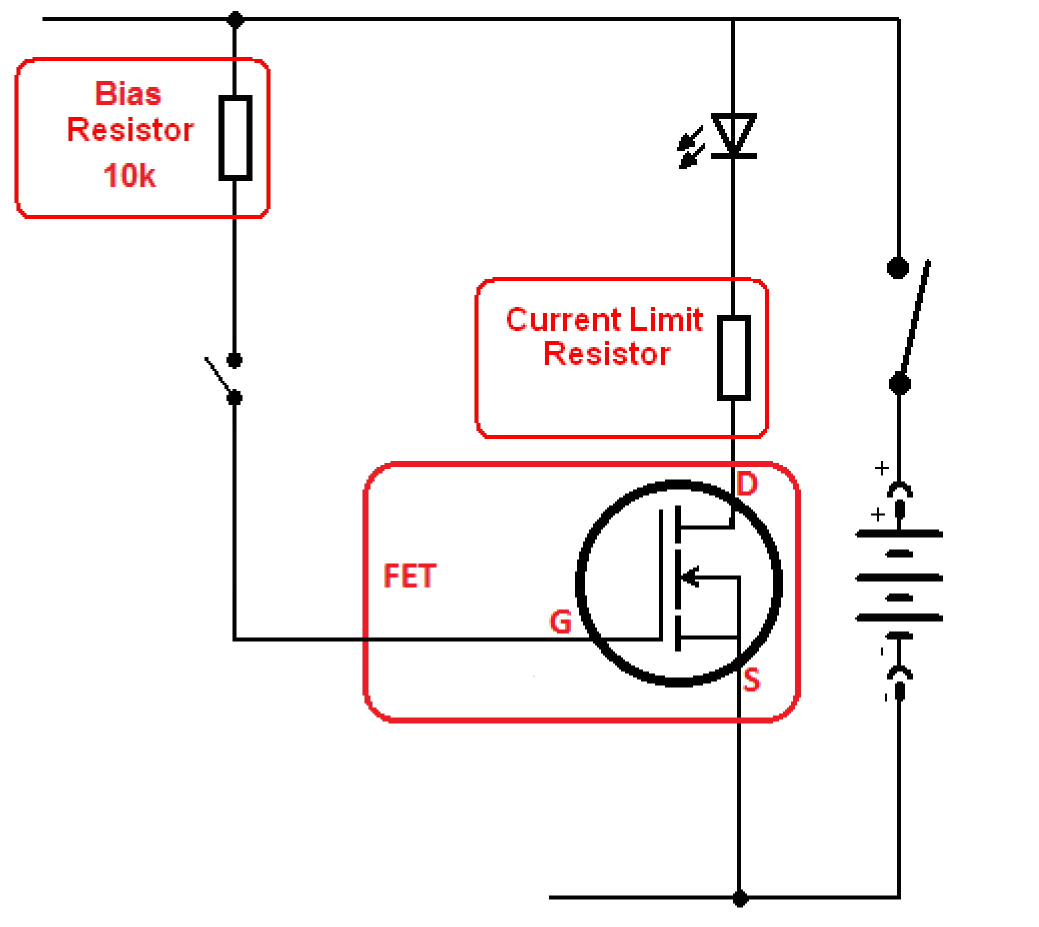
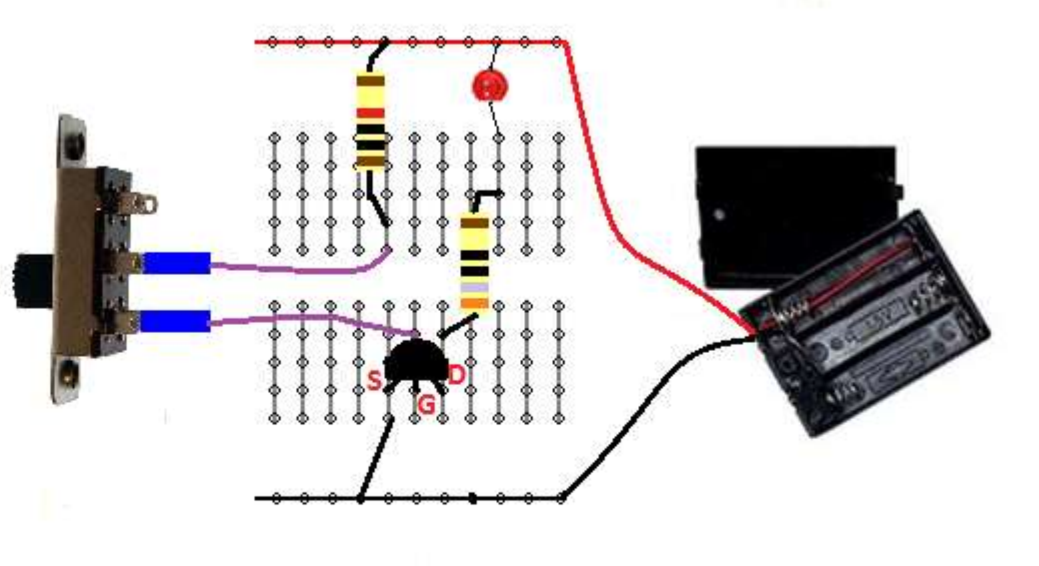
Conductors such as the wires connecting components together have (theoretically) no resistance to current.
A really important concept to get clear in your mind is that:
<u>Voltage is across components and current is through components</u>.

2.9 Changing the value of resistance

What is the effect of different resistor values on our circuit?
The resistor controls the current flow, the higher the resistor value the lower the current. (what would a 10K resistor look like?

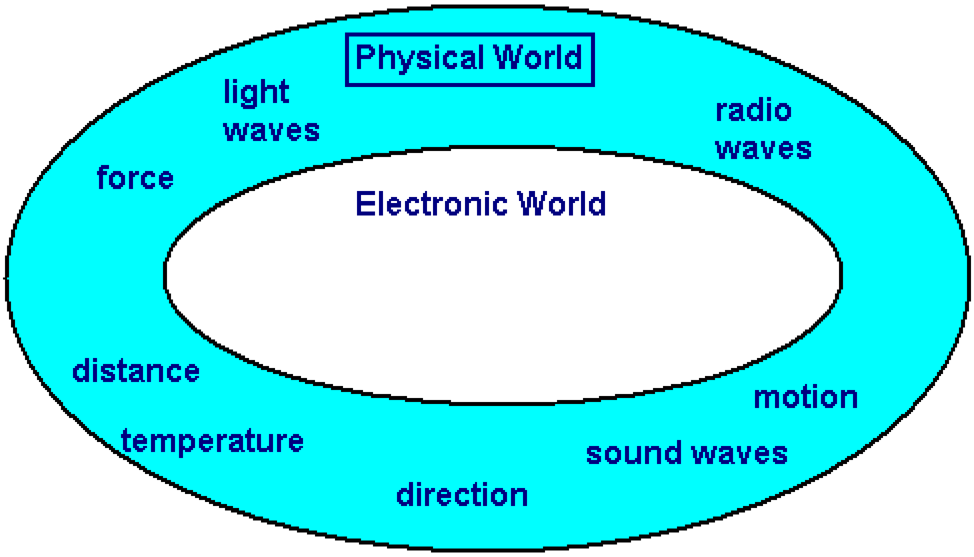


2.10 Adding a transistor to your circuit

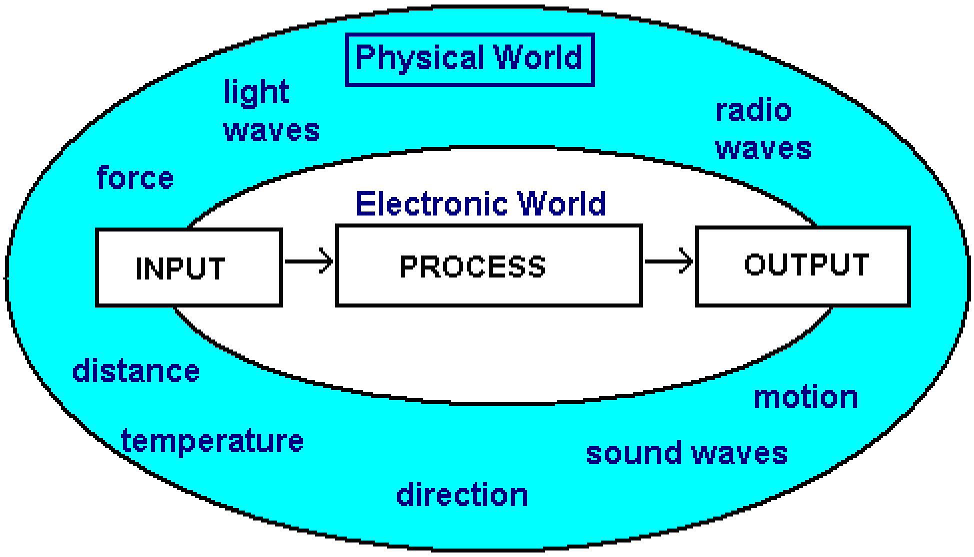
Schematic (circuit diagram)	2N7000 FET (Field Effect Transistor)
	A FET is a control component that amplifies small signals. It has three legs or leads: D – Drain G – Gate S – Source Only a small signal is required on the gate to control a larger current through the source to the drain. The collector current is the same current which lights the LED The 390 limits this current to an acceptable value for the LED.
Breadboard layout diagram	
	

2.11 Understanding circuits

Electronics is all about controlling the physical world. Physical objects have properties such as temperature, force, motion, sound/radio/light waves associated with them



Electronic devices have **input** circuits to convert the physical world (light sound etc) to different voltage levels.
They have **process** circuits that transform, manipulate and modify information (the information is coded as different voltages).
They have **output** circuits to convert differen coltgae levels back to the physical world where we can sense the outcome of the process (light, sound etc)



Take an example such as a television, the physical world radio signal on the input is converted to an voltage level, this is processed by the electronic circuit and converted to light which we see and sound which we can hear.

2.12 The input circuit – an LDR

The LDR or Light Dependant Resistor is a common component used in circuits to sense light level. An LDR varies resistance with the level of light falling on it.

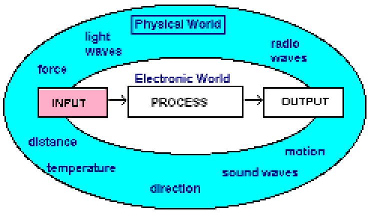
LDRs are made from semiconductors such as Selenium, Thalliumoxid and Cadmiumsulfide.

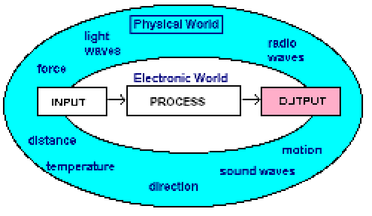
As photons of light hit the atoms within the LDR, electrons can flow through the circuit. This means that as light level increases, resistance decreases.

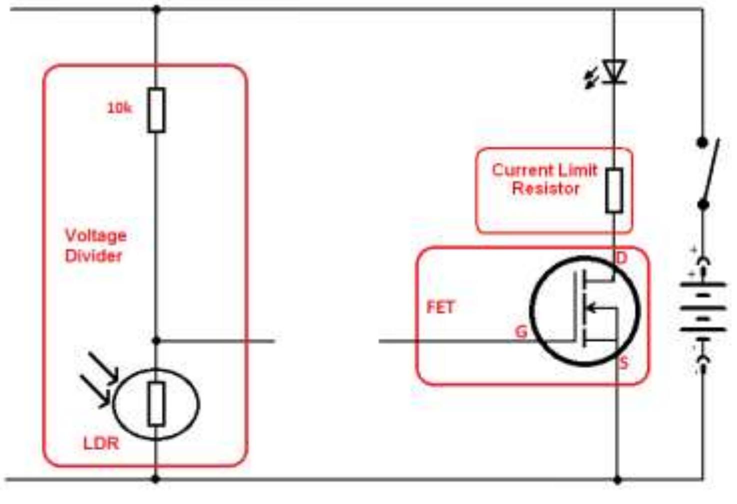
Find an LDR and measure its resistance:

in full daylight the LDRs resistance is approximately ____________

in darkness the LDRs resistance is approximately ______________

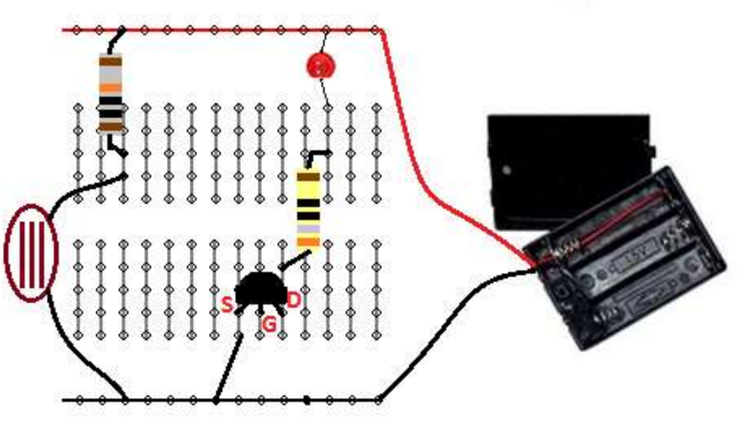






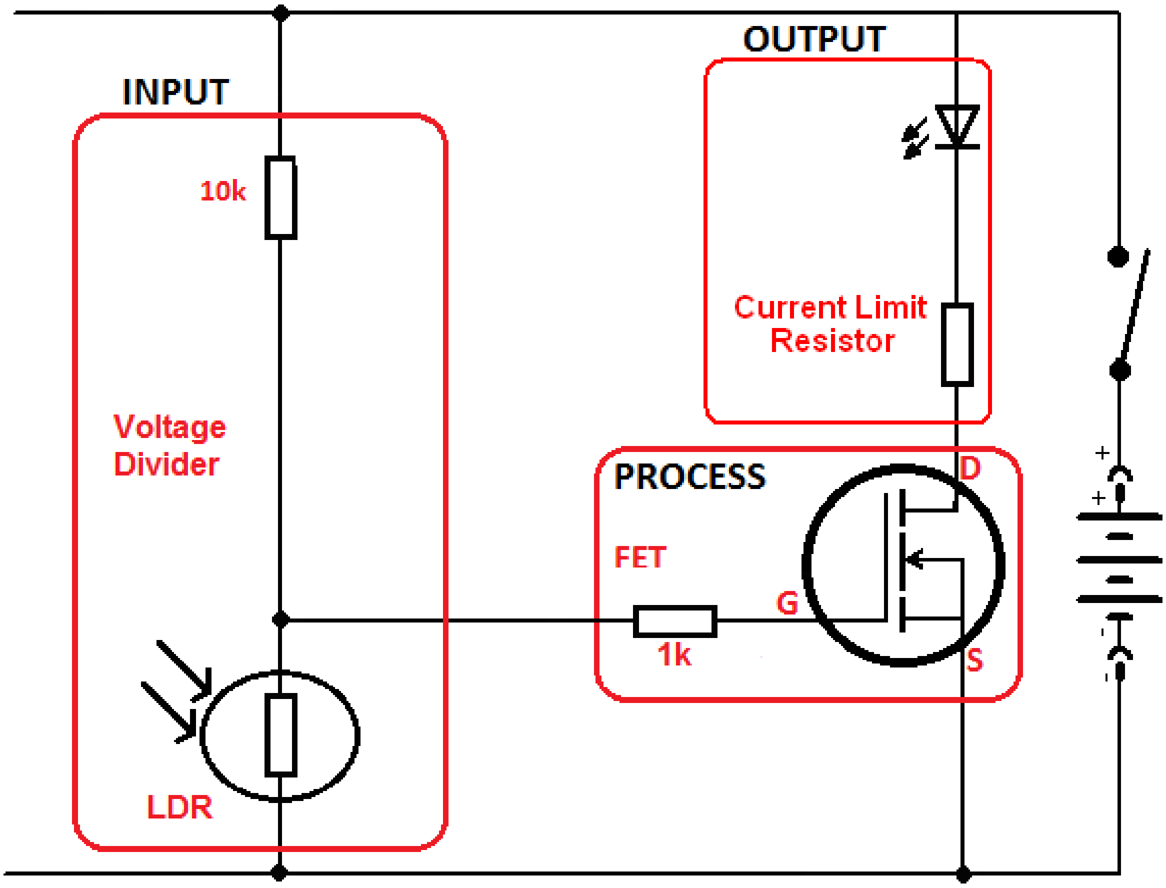
LDRs can only with stand a small current flow e.g. 5mA, if too much current flows they may overheat and burn out. They are used in voltage divider circuits with a series resistor. The components are a resistor from 10K (10,000) ohms to 1M (1,000,000) ohms, an LDR, a battery and the circuit is a series one.

When it is dark the LDR has a high resistance and the output voltage is high.

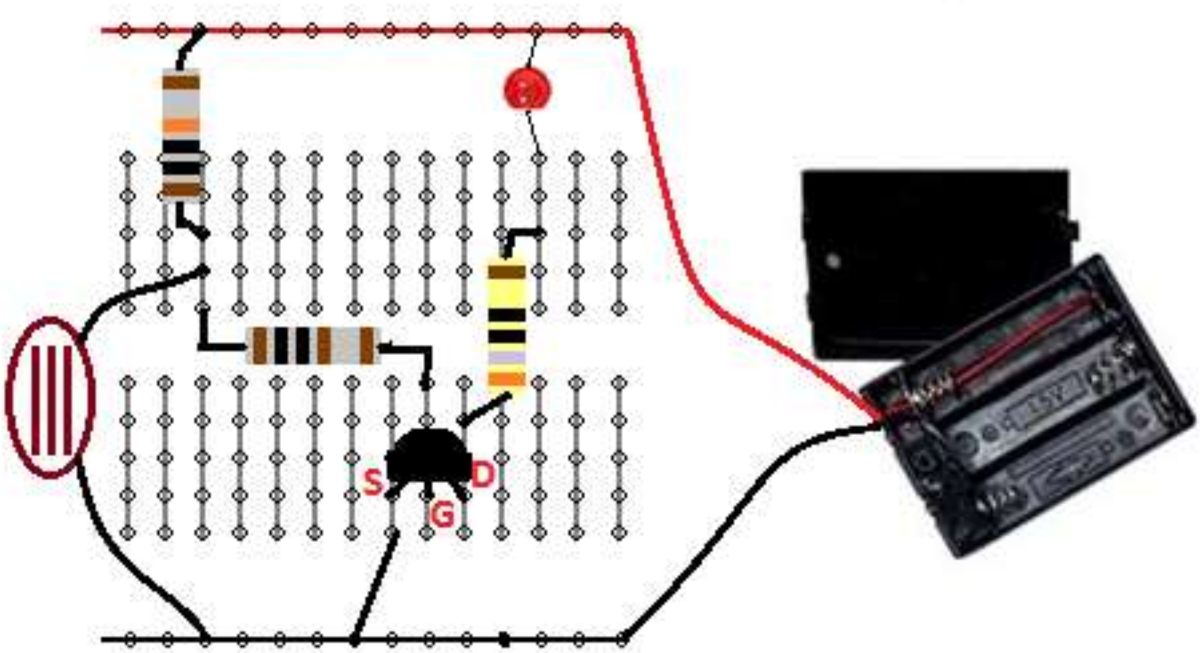


When it is bright the LDR has a low resistance and the voltage is low.

2.13 Working darkness detector circuit

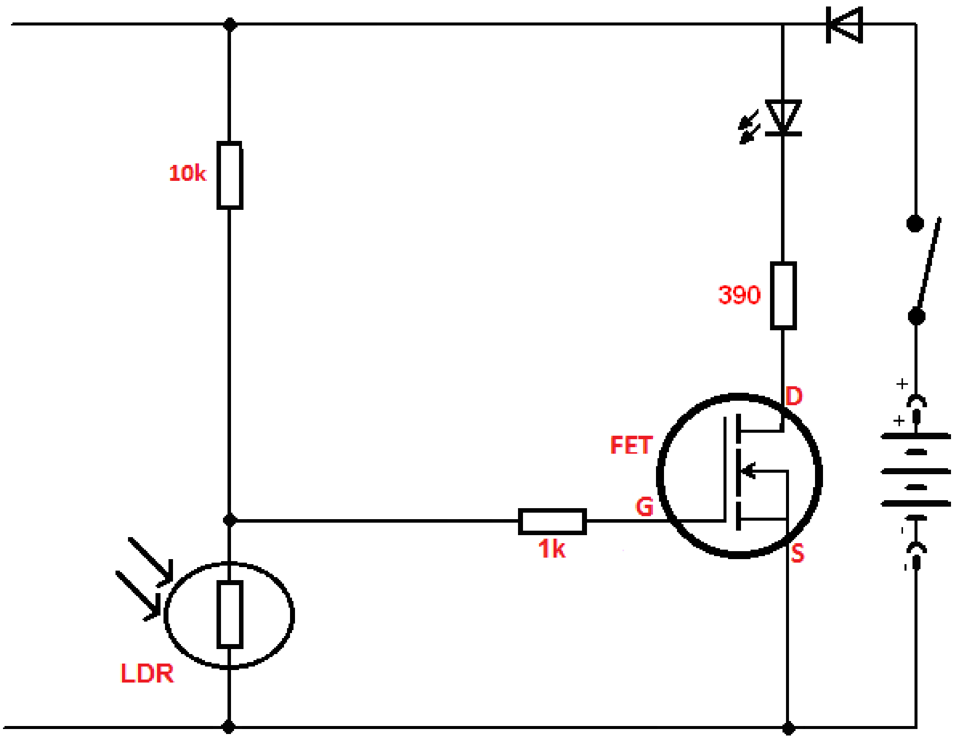


The resistor in series with the LDR can be experiments with to change the sensitivity of the circuit to different light levels.



2.14 Protecting circuits – using a diode

Diodes are very common components, they come in all shapes and sizes.



The key characteristic of a diode is that there is current in only one direction so you cannot reverse it in the circuit and expect it to work.

In this modified circuit the power is supplied from the battery. The circuit is protected by a diode, this means if the battery is connected in reverse then there is no current because the diode blocks it (this is commonly used in the workshop to protect our circuits from a reverse polarity situation).

Of course no diode is perfect and should the voltage of the power supply exceed the voltage rating of the diode then the diode would breakdown, this means the current would increase rapidly and it would burn up. The 1N4004 has a 400V rating.

Diodes can only take a certain current in the forward direction before they overheat and burn up. The 1N4004 has a maximum forward current of 1Amp.

2.15 Diode Research Task

Research the specifications for these two common diodes (ones we use often in class) and find out what each specification / attribute means.

	Description	1N4007	1N4148
Peak reverse voltage			
Maximum forward current			

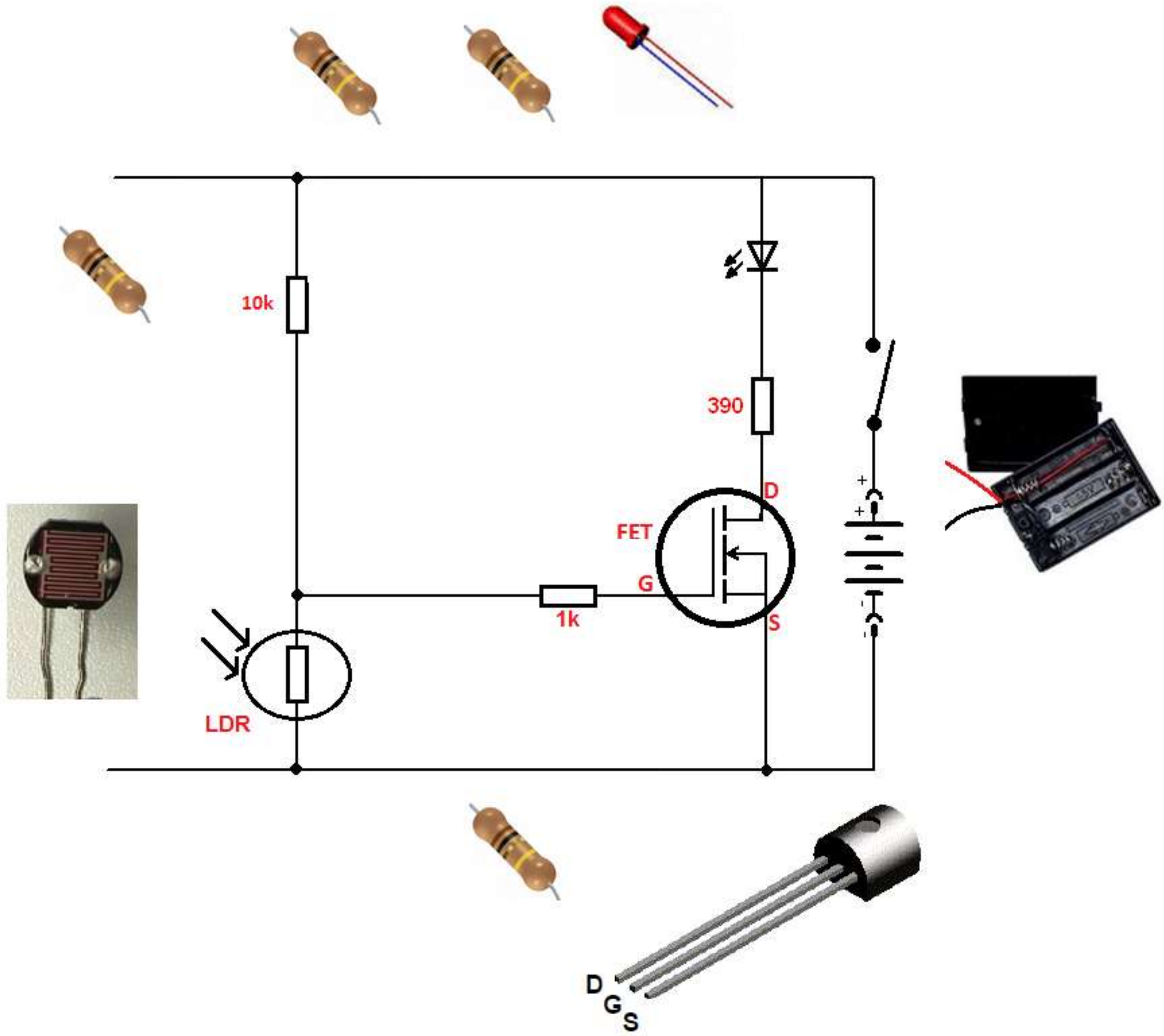
2.16 Final darkness detector circuit

The function of the **input** part of the circuit is to detect light level.
The function of the **process** part of the circuit (the transistor) is to amplify the small change in voltage due to light changes.
The function of the **output** part of the circuit is to indicate something to the end user.
The function of the power supply is to safely provide the energy for the circuit to work

When it is dark the LED is switched on, when there is light present the LED is switched off. This circuit could be used to help a younger child orientate themselves at night and to find the door in a darkened room.
The DIODE, LED and TRANSISTOR are polarised, have positive and negative ends and therefore require wiring into the circuit the right way round or it will not work

You can identify the LED polarity by the flat on the LED body(negative-cathide) or by the longer lead (positive or anode)
You can identify the TRANSISTOR polarity by the shape of the bidy and the layout of the three leads

Draw lines from the components to the symbols to help you remember them. Remember the resistor in the output circuit was made a lower value (changed from 1k to 390ohms) to make the LED brighter in the final circuit.

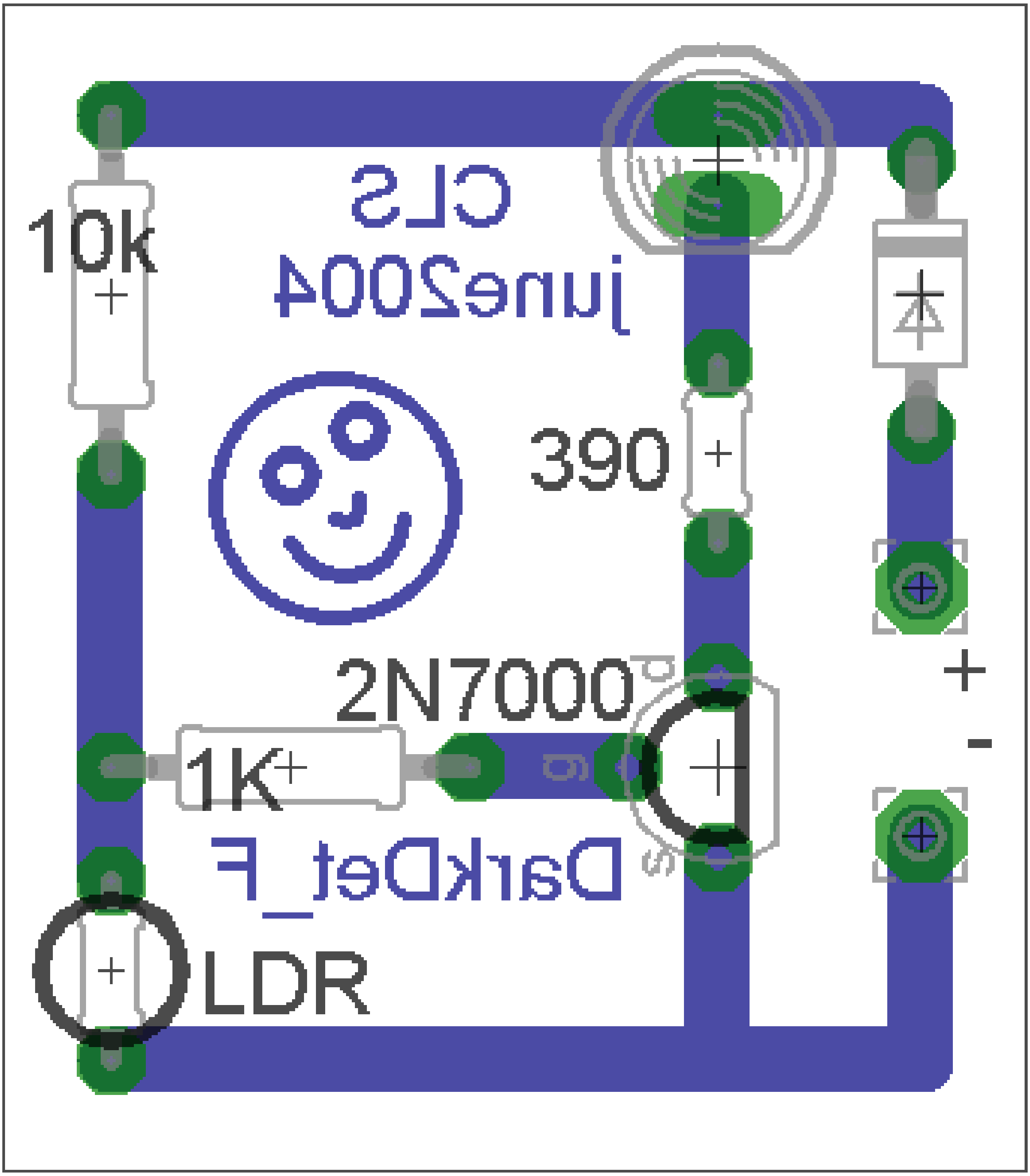


3 Introductory PCB contruction

We take a short break from programming to introduce a further topic of construction - PCB making

3.1 Eagle Schematic and Layout Editor Tutorial

A circuit such as the darkeness detector is no good to us on a breadboard it needs a permanent solution and so we will build it onto a PCB (printed circuit board).

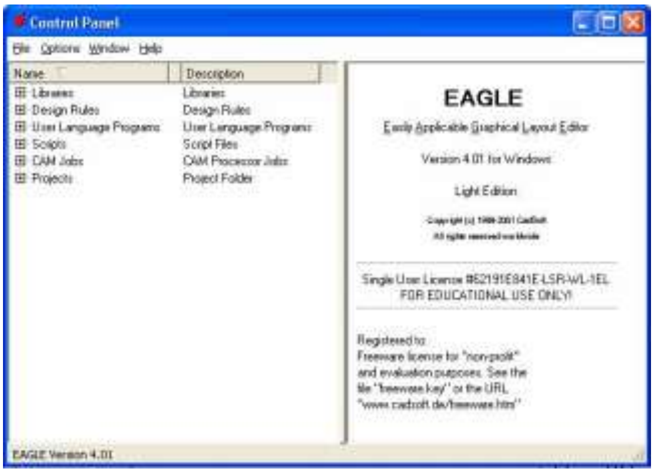


3.2 An Introduction to Eagle

Eagle is a program from www.cadsoft.de that enables users to draw the circuit diagram for an electronic circuit and then layout the printed circuit board. This is a very quick start tutorial, where you will be led step by step through creating a PCB for a TDA2822 circuit.

The version used is the freeware version which has the following limitations; the PCB size is limited to 100mm x 80mm and the board must be not for profit

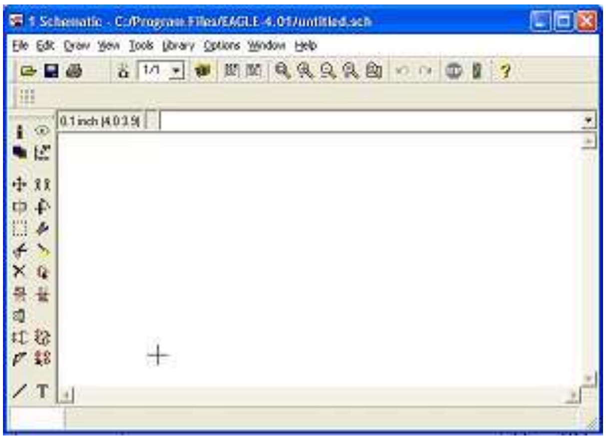
3.2.1 Open Eagle Control Panel



Start - Programs - Eagle - Eagle 4.13

3.2.2 Create a new schematic

On the menu go to FILE then NEW then SCHEMATIC

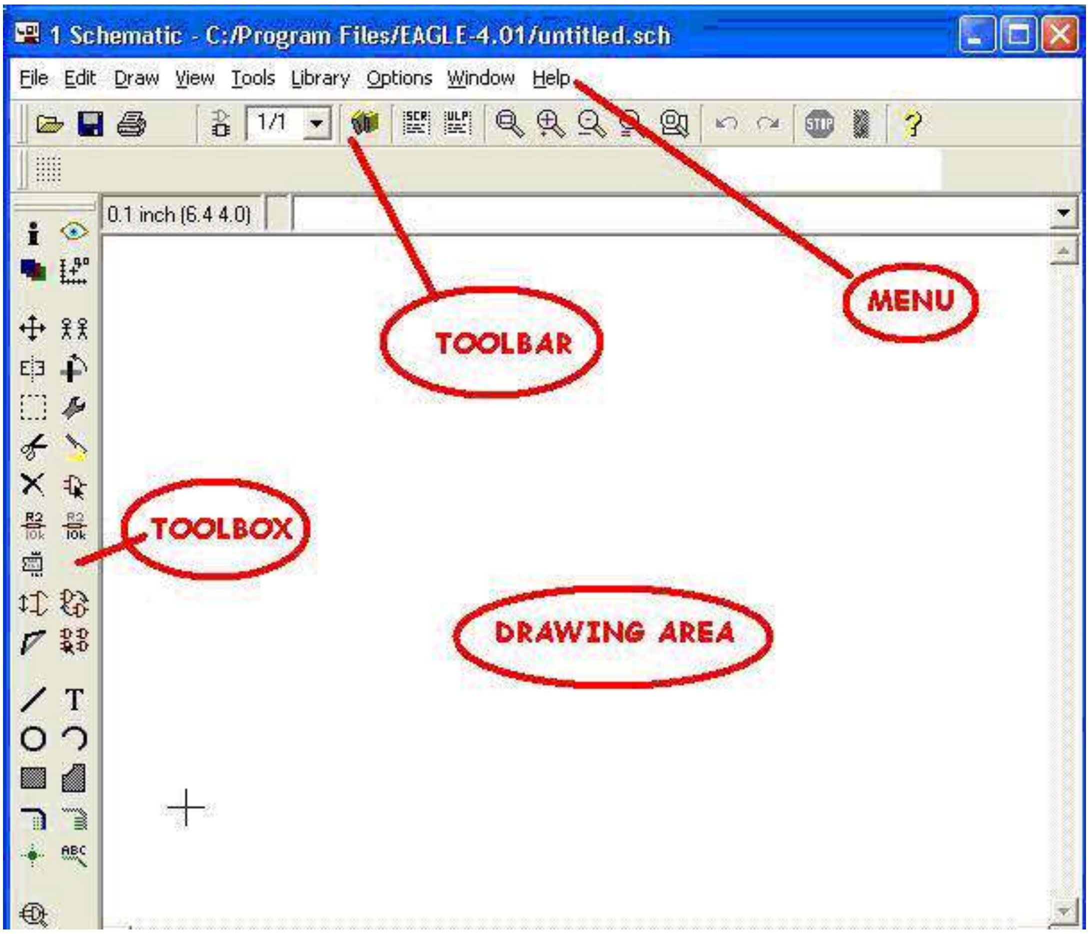


You will see the schematic editor

3.2.3 Saving your schematic

- It is always best to save your data before you start work
- Eagle creates many temporary files so you need to keep your folders tidy.
- If this is the first time you have used Eagle create an Eagle folder within your folder on the server.
- Within the Eagle folder create a folder for the name of this project e.g. DarkDetector
- Save the schematic as DarkDetector verA.sch within the DarkDetector folder.

3.3 The Schematic Editor



The first part of the process in creating a PCB is drawing the schematic.

1. **Parts** will be added from libraries
2. and joined together using '**nets'** to make the circuit

3.3.1 The Toolbox

As you point to the tools in the TOOLBOX their names will appear in a popup and also their description will appear in the status bar at the bottom of the window

Find the following tools

- ADD A PART
- MOVE AN OBJECT
- DELETE AN OBJECT
- DEFINE THE NAME OF AN OBJECT
- DEFINE THE VALUE OF AN OBJECT
- DRAW NETS (connections)
- ERC (electrical rule check)

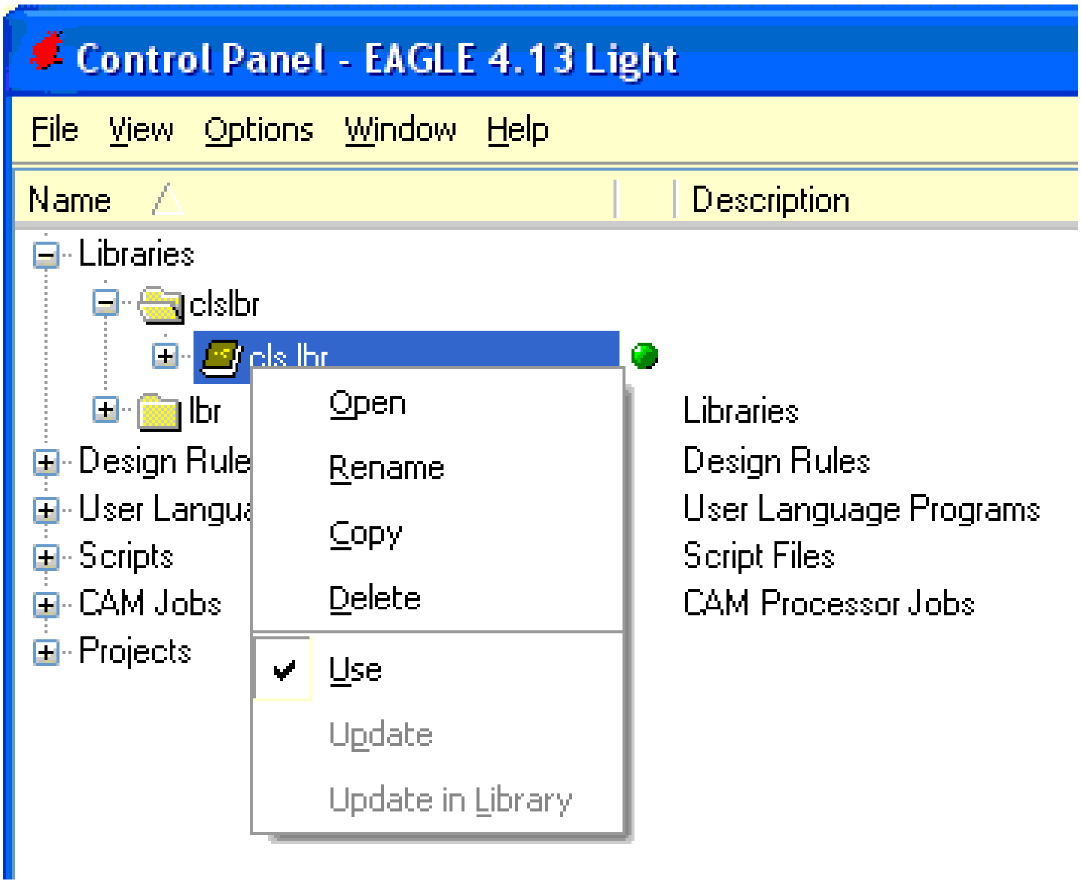
3.3.2 Using parts libraries

Selecting parts libraries to use.

Parts are stored within libraries and there are a large number of libraries in Eagle.

It is not hard to create your own library and modify the parts within it. The cls.lbr has many already modified components within it. If Eagle is not setup to use the cls library you will need to do it now.

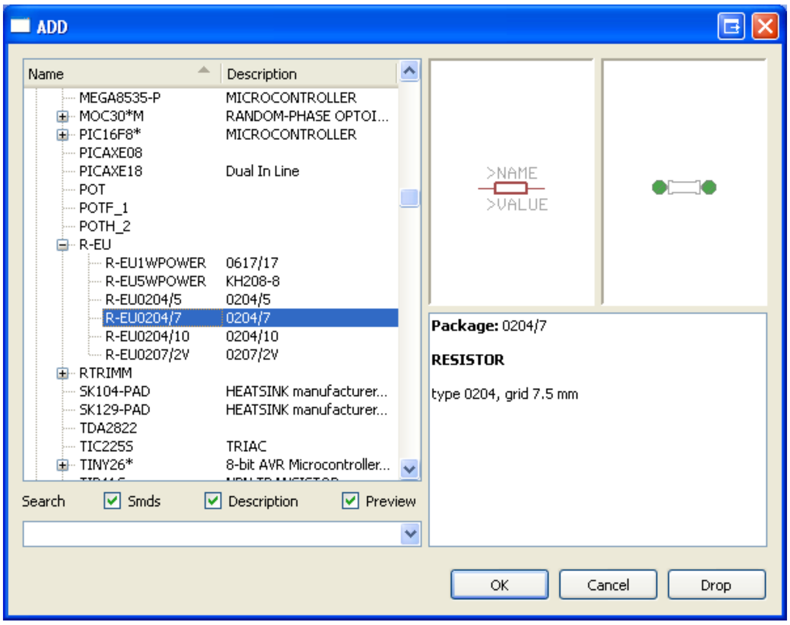
1. From your internet browser save the file cls.lbr into your Eagle folder.
2. In Eagle's control panel from the menu select **options** then **directories**
3. In the new window that appears make sure the directories for the libraries are highlighted
4. Click on **browse** and find your Eagle.directory
5. Next highlight the directories for Projects
6. Click on browse and find your Eagle directory again.
7. Choose **OK**.
8. You might need to close EAGLE and restart it to make sure it reads the libraries ok.
9. To use a library right click on it from within the Control Panel
10. Make sure **Use** is highlighted. It will have a green dot next to it if it is selected
11. At this time right click on the other lbr folder and select **Use none**.



NOTE THE IMPORTANCE OF THE GREEN DOT NEXT TO THE LIBRARY,
if its not there you will not see the library in the schematic editor!

3.3.3 Using Components from within libraries.

From your schematic Click the ADD button in the toolbox
A new window will open (it may take a while)



- Find the CLS library
- Open it by double clicking on it or by clicking the + sign
- Open the R-EU_ section (Resistor-European)
- Here you will find the 0204/10 resistor.
- Select it and then click OK

Add **2** more resistors of the same type.

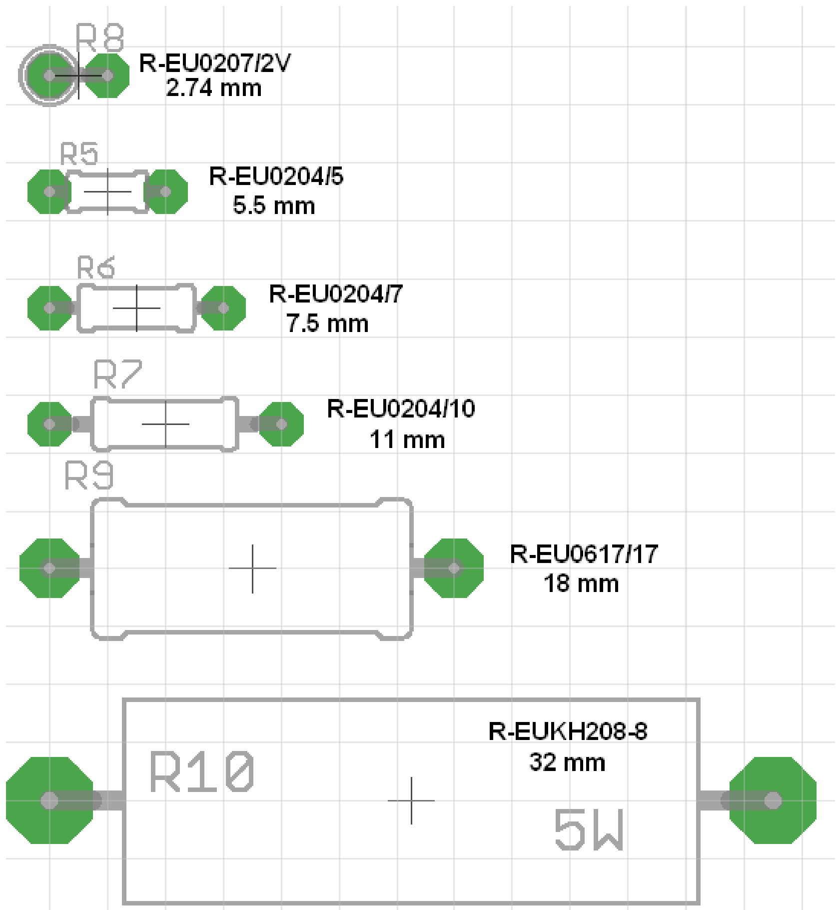
Add all of the following parts

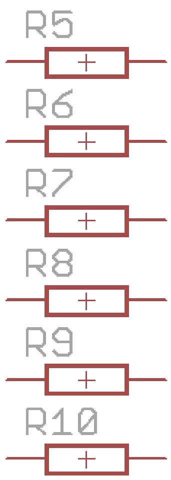
LIBRARY	PART	Qty
cls	REU-0204/10	3
cls	LDR	1
cls	2,54/0,8 (wirepads)	2
cls	led 5MM	1
cls	1N4148 D41-10	1
cls	2N7000	1
cls	GND	3

A wirepad allows us to connect wires to the PCB (such as wires to switches and batteries)

3.3.4 Different component packages

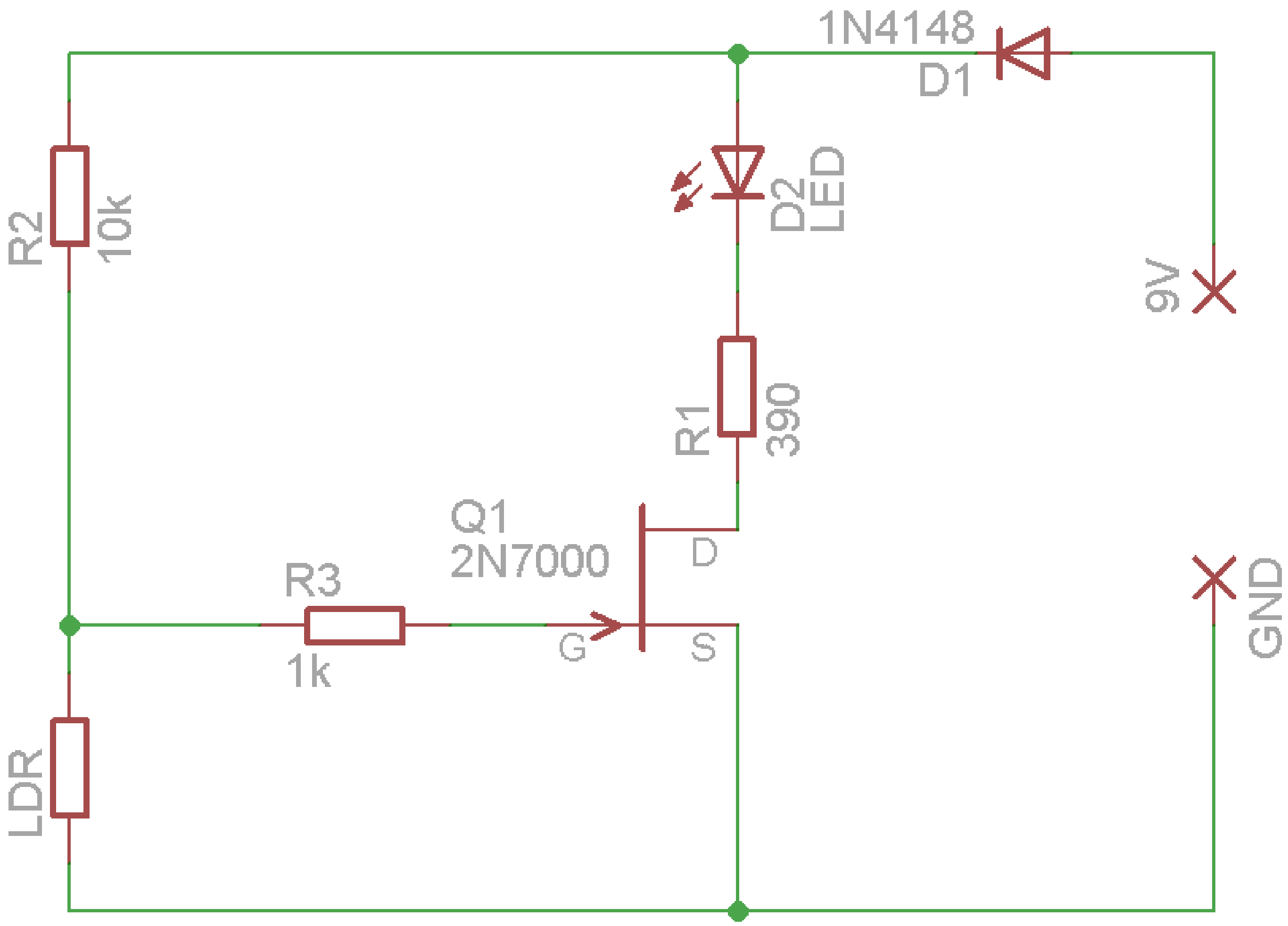
There are several different types of resistors; they all have the same symbol however resistors come in different physicalpackages so we must choose an appropriate one.The 0204/7 is suitable for us but any of the 4 smallest ones would be OK.



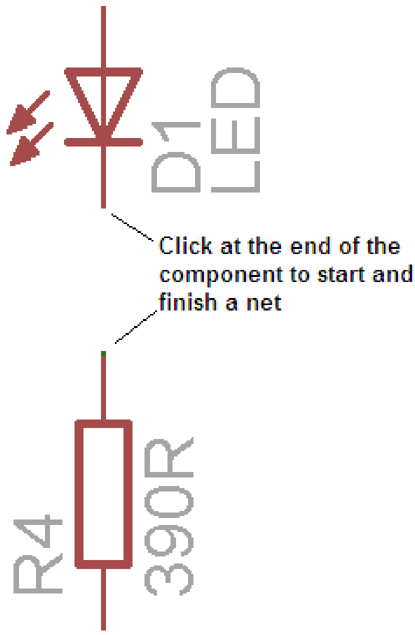


Moving parts

Move the parts around within the schematic editor so that they are arranged as per the schematic below. Keep the component identifiers (numbers like R1, R2, R3) in the same places as those below.



3.3.5 Wiring parts together



These form the electrical connections that makeup the circuit. Select the **net** button from the toolbox.

Left click on the very end of a component and draw in a straight line either up, down, left or right.

Left click again to stop at a point and draw before drawing in another direction.

Double left click at another component to finish the wire.

3.3.6 Zoom Controls

There are a number of zoom controls that can be used to help you work in your circuit.

Find these on the toolbar and identify what each does.

Nets

Nets are the wire connections between the components, each has a unique name.

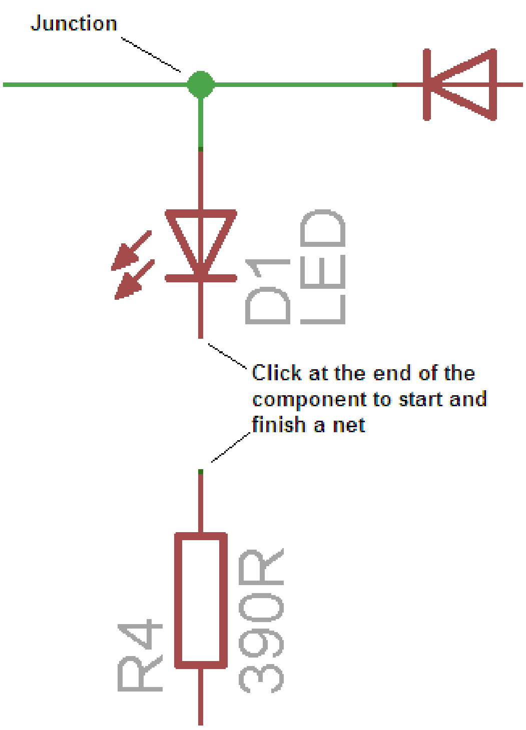
Find the info button in the toolbox and check the names and details of the components and nets/wires.

When you want to connect a new net to an existing net, Eagle will prompt you as to which name to give the combined net.

If one of the nets has a proper name i.e. VCC, V+,V-, ground... use that name, otherwise choose the net with the smallest number

3.3.7 Junctions

Junctions are the dots at joins in the circuit, they are there to make sure that the wires are electrically connected. Generally you will NOT need to add these to your circuit as the net tool puts them in place automatically



3.3.8 ERC

The ERC button causes Eagle to test the schematic for electrical errors.

Errors such as pins overlapping, and components unconnected are very common.

The ERC gives a position on the circuit as to where the error is; often zooming in on that point and moving components around will help identify the error.

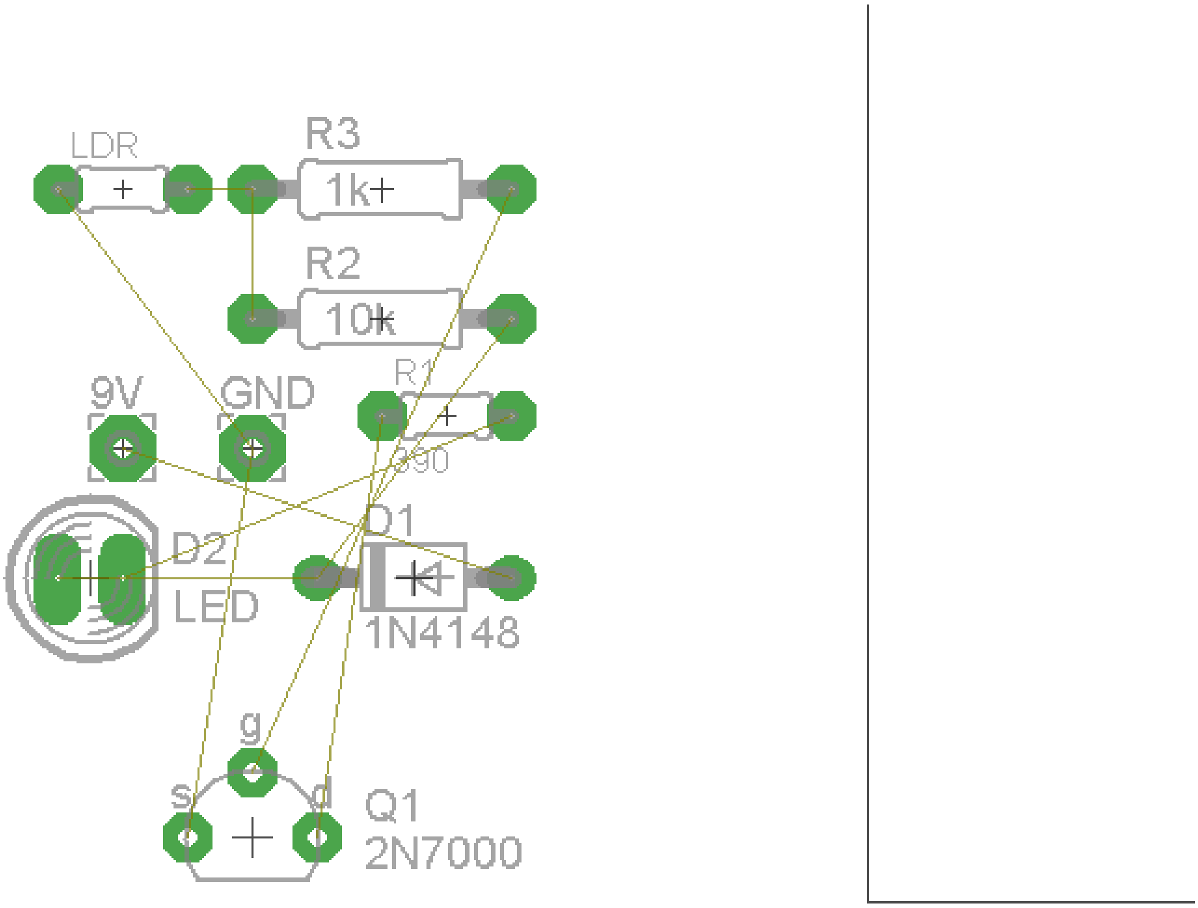
<u>You must correct all errors before going on</u>.

3.4 The Board Editor

The board editor is opened using a button in the toolbar, find this button and answer yes to the question about creating the board.

The new window has a pile of parts and an area upon which to place them.

<u>**WARNING: once you have started to create a board always have both the board and schematic open at the same time, never work on one without the other open or you will get horrible errors which will require you to delete the .brd file and restart the board from scratch.**</u>

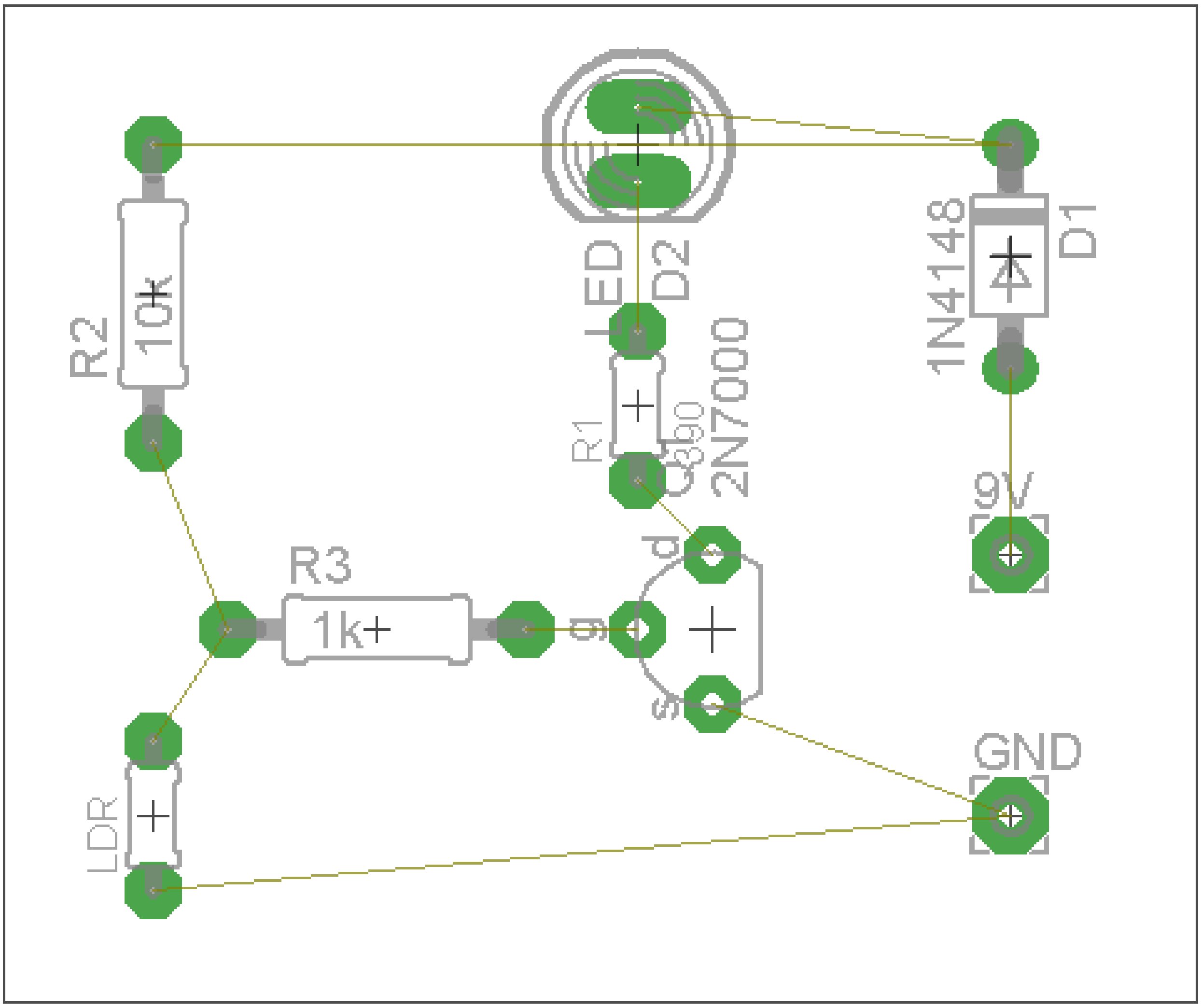


3.4.1 Airwires

The wires from the schematic have become connections called airwires, these wires will shortly become tracks on the PCB.

These connections can look very messy at times and at this stage it is called a **RATSNEST**.

3.4.2 Moving Components



Move the components into the highlighted area. In the demo version you cannot place parts outside this area. Keep the components in the lower left corner near the origin (cross).

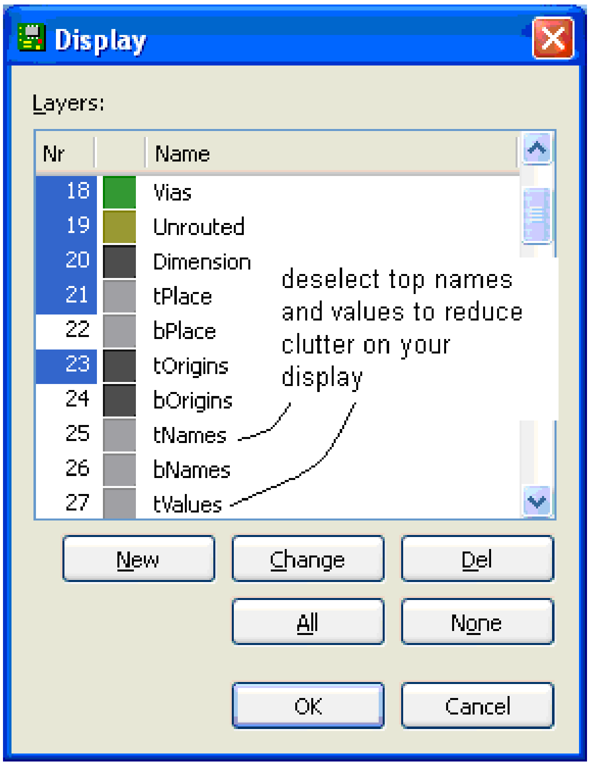
Reduce the size of the highlighted area you are using for the components. Then zoom to fit. <u>Progressively arrange the components so that there is the minimum number of crossovers.</u>

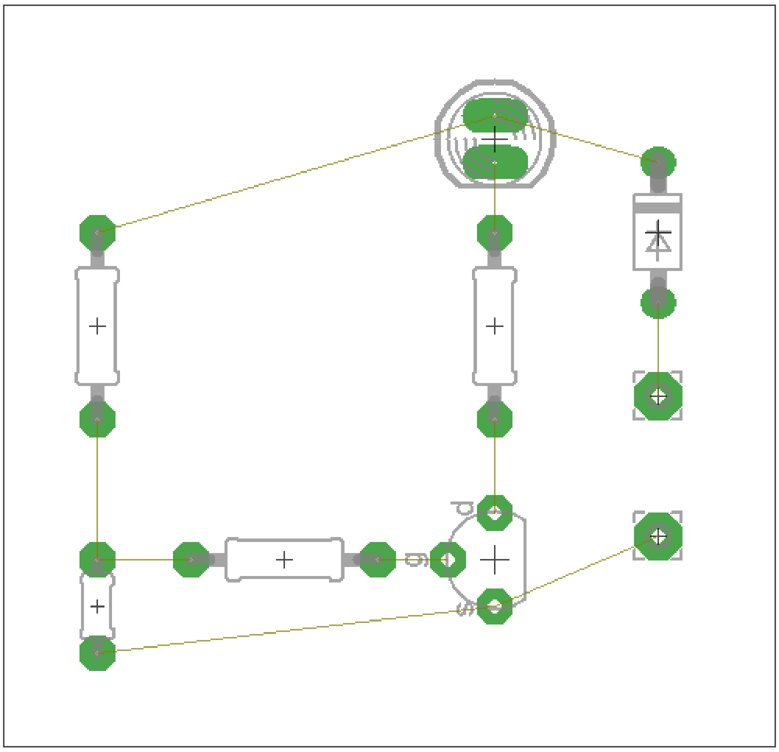
As you place components press the Ratsnest button often to reorganize the Airwires. Eventually your picture will look like the one on the right.

Good PCB design is more about placement of components than routing, so spending most of your time (80%) doing this step is crucial to success.
You want to make track lengths as short as possible

3.4.3 Hiding/Showing Layers

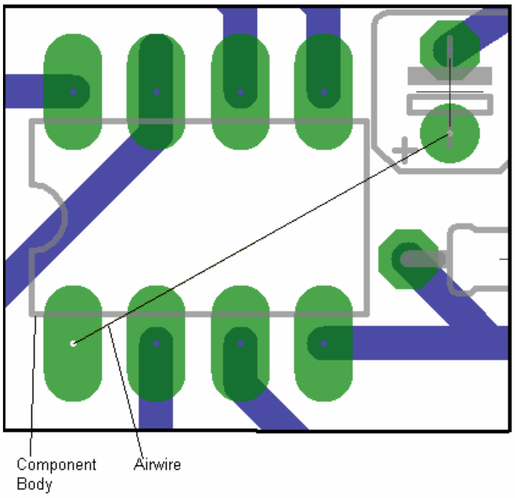
The DISPLAY button in the TOOLBOX is used to turn on and off different sets of screen information. Turn off the names, and values while you are placing components. This will keep the screen easier to read. Turn off the layer by selecting the display button and in the popup window pressing the number of the layer you no longer want to see.





3.4.4 Routing Tracks

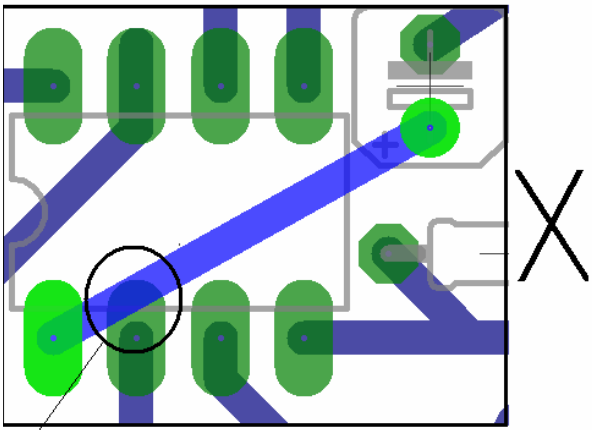
Now is the time to replace the airwires with actual PCB tracks. Tracks need to connect all the correct parts of the circuit together without connecting together other parts. This means that tracks cannot go over the top of one another, nor can they go through the middle of components!



Go to the Toolbar, Select the ROUTE button

On the Toolbar make sure the Bottom layer is selected (blue) and that the track width is **0.04**. Left click on a component.

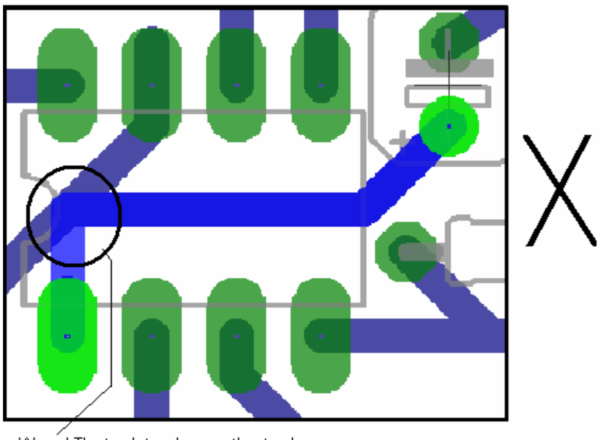
Note that around your circuit all of the pads on the same net will be highlighted. Route the track by moving the mouse and left clicking on corner points for your track as you go. YOU ONLY WANT TO CONNECT THE PADS ON THE SAME NET, DON'T CONNECT ANY OTHERS OR YOUR CIRCUIT WILL NOT WORK. Double click on a pad to finish laying down the track.



Wrong! The track is touching another pin

Track layout Rules

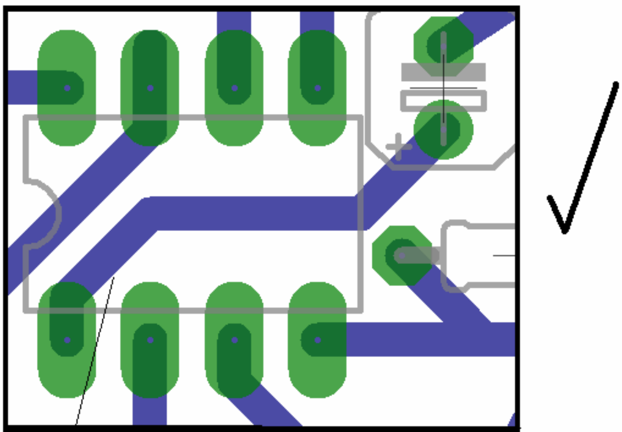
1. **Place tracks so that no track touches the leg of a component that it is not connected to on the schematic**
2. **No track may touch another track that it is not connected to on the schematic**
3. **Tracks may go underneath the body of a component as long as they meet the above rules**



Wrong! The track touches another track

3.4.5 Ripping up Tracks

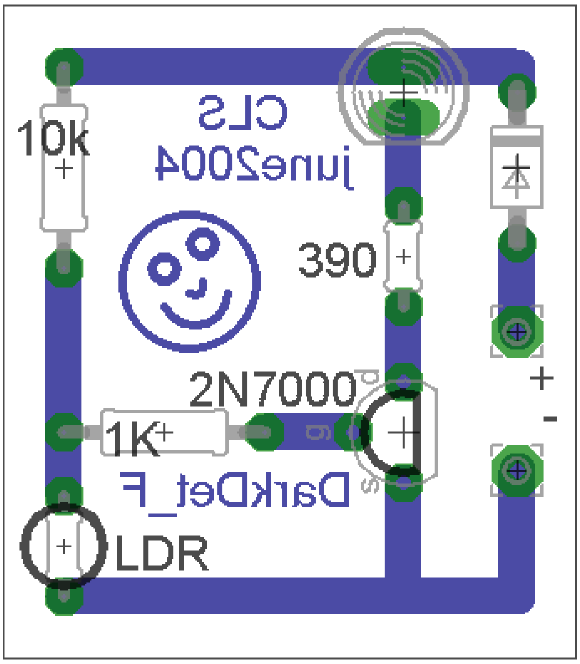
Ripping up a track is removing the track you have laid down and putting the airwire back in place. This will be necessary as you go to solve problems where it is not possible to route the tracks. You may even want to rip up all the tracks and move components around as you go.



Correct! The track doesn't touch any other track or any other pin

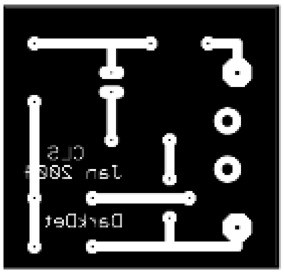
3.5 Making Negative Printouts

Eagle is straight forward at producing printouts for a positive photographic pcb making process.



(NOTE THE TEXT ON THE PCB APPEARS REVERSED THIS IS CORRECT)

If your photosensitive board requires a negative image such as this, another stage on the process is required..



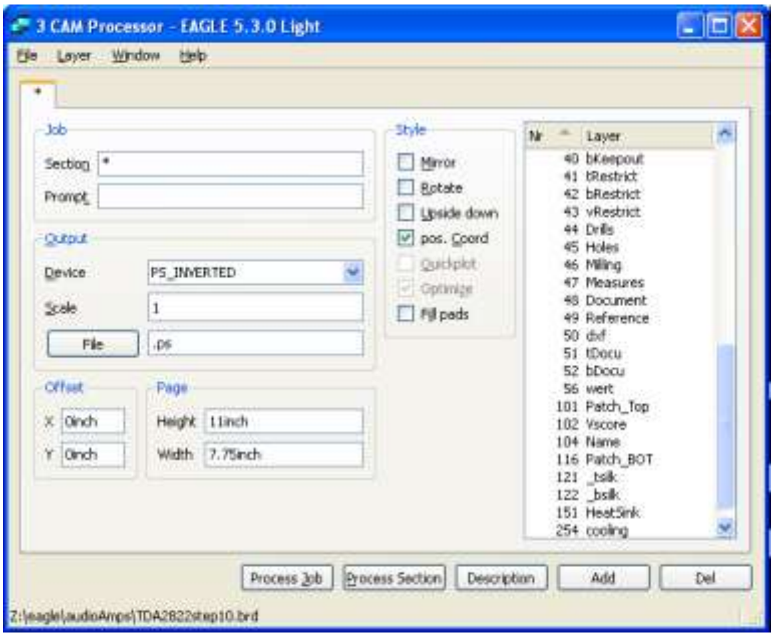
3.5.1 Other software required

The following software is required to manipulate the special CAM (computer aided manufacturing) files created by Eagle (and other pcb CAD software) into the printed image you require. All this software is shareware with no fees attached for its use by students.

* Install GhostScript - from http://www.ghostscript.com
* Install GSView - from http://www.ghostgum.com.au/

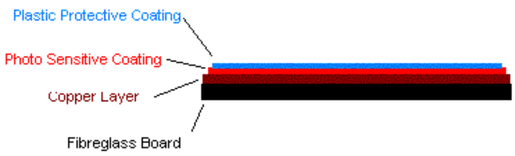
Conversion process
This process creates a '.ps' (postscript file), it is the best output from Eagle to use. It will keep the board exactly the same and correct size for printing.

* Open TDA2822verA.brd in Eagle
* From within the Eagle Board Editor start the CAM Processor
* select device as PS_INVERTED
* Scale = 1
* file = **.ps**
* make sure fill pads is **NOT** selected this makes small drill holes in the acetate which we use to line up the drill with when drilling
* for layers select only **16,17,18 and 20**,
* make sure **ALL** other layers are **NOT** selected.
* Select process job
* if you will use this process a lot save this cam setup as so that you can reuse it again



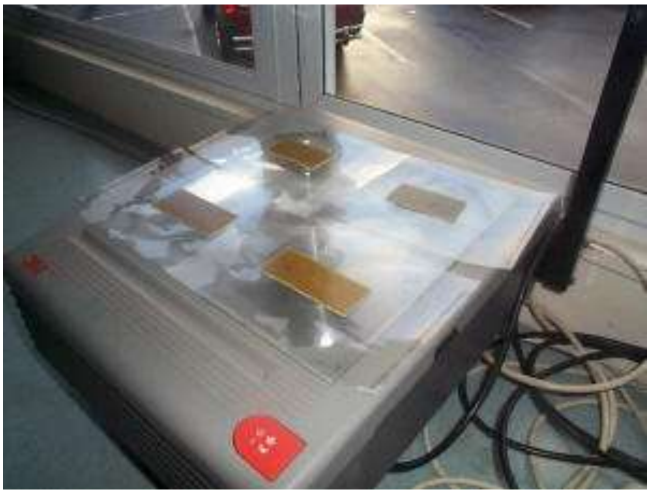
Open the TDA2822verA.ps file with Ghostview for printing and print it onto an over head transparency. Make sure you can see the drill holes!

3.6 PCB Making



PCB Board Layers

Measure, Cut:
Photosensitive board is expensive, so it is important not to waste it and cut it to the right size.
It is also sensitive to ordinary light so when cutting it don't leave it lying around.



Expose:
This over head projector is a great source of UV – ultra violet light, it takes three minutes on the OHP in my classroom.
The overhead transparency produced earlier must have some text on it. The text acts as a cue or indication of which way around the acetate and board should be. We want the text on the board to be around the right way.

Develop:
The developer chemical we use is sodium metasillicate which is a clear base or alkali. It will ruin your clothes so do not splash it around, it is a strong cleaning agent!
It should be heated to speed up the process.
The development process takes anywhere from 20 seconds to 2 minutes. The reason being that the chemical dilutes over time making the reaction slower.
The board should be removed twice during the process and washed gently in water to check the progress.

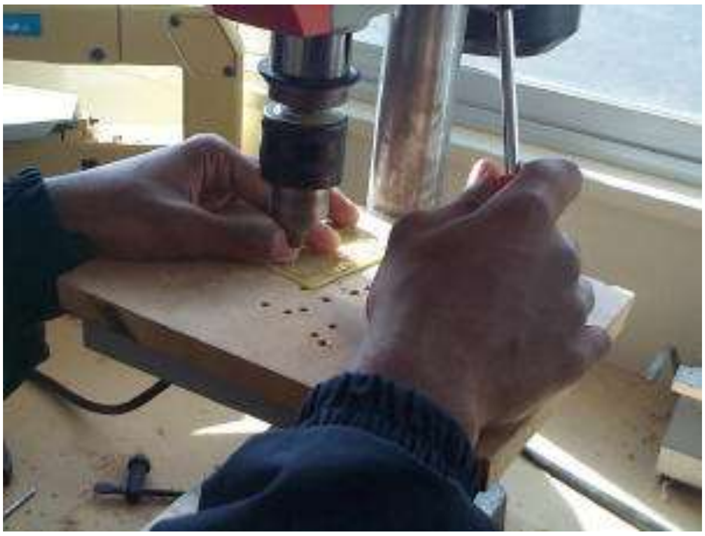
Rinse:
The developer must becompletely removed from the board.
At this stage if there is not time to etch the board, dry it and store it in a dark place.

Etch:
The etching chemical we use if ferric chloride, it is an acid and will stain your clothes.
The tank heats the etching solution and there is a pump to blow bubbles through the liquid, this speeds the process up radically so always use the pump.
Etching may take from 10 to 30 minutes depending upon the strength of the solution.

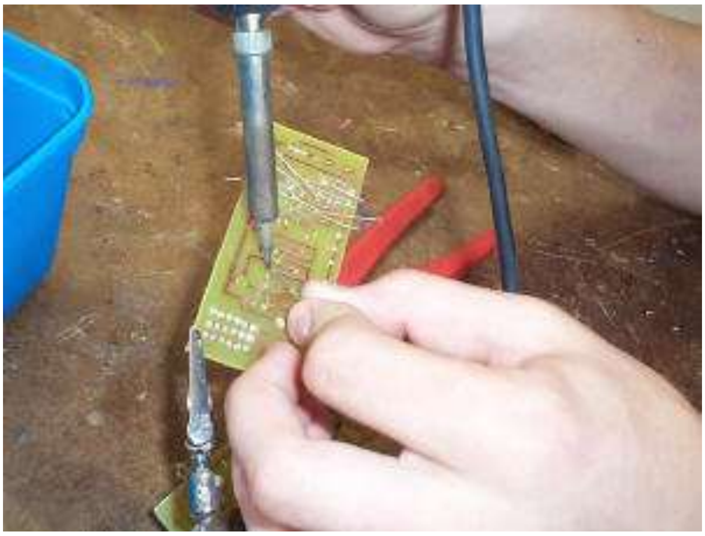
Rinse:
Thorughly clean the board.

Remove Photosensitve Resist:
The photosensitive layer left on the tracks after etching is complete must be removed. Thee asiest way to do this is to put the board back into the developer again. This may take about 15 minutes.

Laquer:
The copper tracks on the board will oxidise very quickly (within minutes the board may be ruined), so the tracks must be protected straight away, they can be sprayed with a special solder through laquer (or tinned).



Drilling & Safety:
Generally we use a 0.9mm drill in class. This suits almost all the components we use.
Take you time with drilling as the drill bit is very small and breaks easily.
As always wear safety glasses!



Use a third hand:
When soldering use something to support the board. Also bend the wires just a little to hold the component in place (do not bend them flat onto the track as this makes them very hard to remove if you make a mistake).

4 Soldering, solder and soldering irons

Soldering is a process of forming an electrical connection between two metals.
The most important point is **GOOD THINGS TAKE TIME, SO TAKE YOUR TIME!**
Quick soldering jobs can become really big headaches in the future, and people learning to solder tend to be quick because either they believe the temperatures will damage the components or they think of the solder as glue.

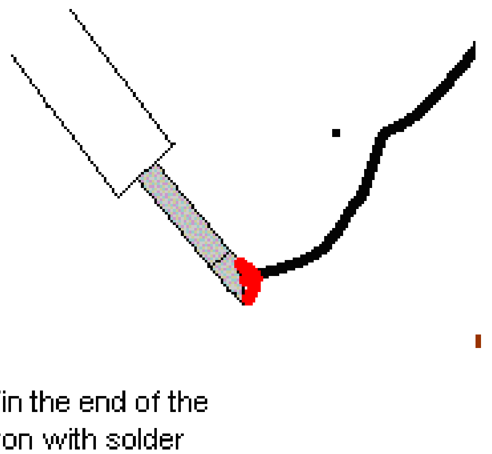
Soldering is best described therefore as a *graceful* process.

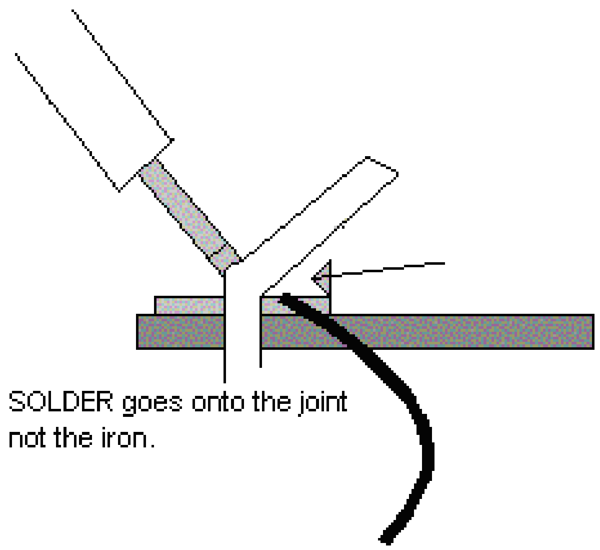
So approach it from that way, always **slowwwwing** down to get a good soldering joint.

Follow these simple steps to get the best results.

1. The materials must be clean.
2. Wipe clean the iron on a moist sponge (the splnge must not be dripping wet!)
3. The iron must be tinned with a small amount of solder.
4. Put the tinned iron onto the joint to heat the joint first.
5. The joint must be heated (be aware that excessive heat can ruin boards and components)
6. Apply the solder to the joint near the soldering iron but not onto the iron itself.
7. Use enough solder so the solder flows thoroughly around the joint- it takes time for the solder to siphon or capillary around all the gaps.
8. Remove the solder.
9. Keep the iron on the joint after the solder for an instant.
10. Remove the soldering iron last – do not clean the iron, the solder left on it will protect it from oxidising
11. Support the joint while it cools (do not cool it by blowing on it)

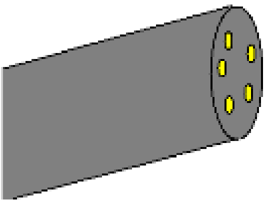
DO NOT - DO NOT - DO NOT - DO NOT repeatedly touch and remove the soldering iron on a joint this will never heat the joint properly, HOLD the iron onto the joint until both parts of it COMPLETELY heat through .s





When you are soldering properly you are following a code of practice

4.1 Soldering facts



- Currently the solder we use is a mix of tin and lead with as many as 5 cores of flux. Don’t use solder whichis too thick.
- When the solder flows smoothly onto surfaces it is know as "WETTING".
- Flux is a crucial element in soldering it cleans removing oxidisation and prevents reoxidisation of components by sealing the area of the joint as solder begins to flow. It also reduces surface tension so improves viscosity and wettability.
- Our use of lead solder may change in the future with the trend to move to non lead based materials in electronics.
- If a solder joint is not heated properly before applying more solder or the solder is applied to the iron not the joint then the flux will all burn away or evaporate before it can do its proper job of cleaning and sealing the materials.
- A new alloy of tin and copper must be formed for soldering to have taken place, it is not gluing!
- The new alloy must have time to form, it will only be around 4-6 um thick
- As solder goes from a solid to a liquid it goes through a plastic state. This is the state of risk for your joint, if something moves during that time the solder will crack.
- It is for this reason that we don't dab at a joint with a hot iron, the joint never really becomes hot enough to melt the solder hence no wetting takes place and the joint is going to be unreliable. If you apply the solder to the joint not the iron you will know the joint is hot enough because the solder will melt.
- Flux is useful for only about 5 seconds. Reheating joints without fresh solder often doesn't do much good, in fact it could even damage them.
- Too much heat on components during soldering can destroy the component or lift the tracks from the PCB.
- If components get very hot while your circuit is on, then they can deteriorate your solder joint and cause it to fracture.
- Soldering provides a certain amount of mechanical support to a joint, however be careful as to how much support you expect it to give. Very small components through the holes in a PCB are fine, some larger components may need other support, often just bending the legs slightly before soldering is enough.

4.2 Soldering Safety

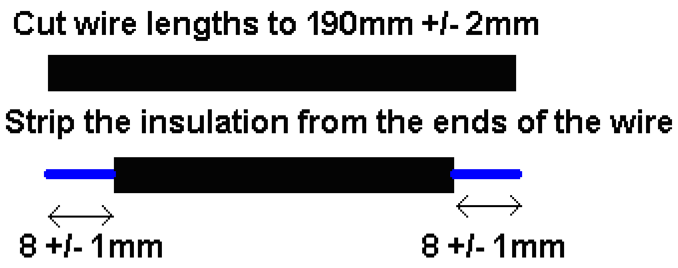
- Lead is a poison so don’t eat solder!
- Solder in a well ventilated area as the fumes coming form the solder are the burning flux and are a nuisance in that they can lead to asthma.
- The soldering iron needs to be hot to be useful around 360 degrees Celsius - it will burn you!

Good solder joints

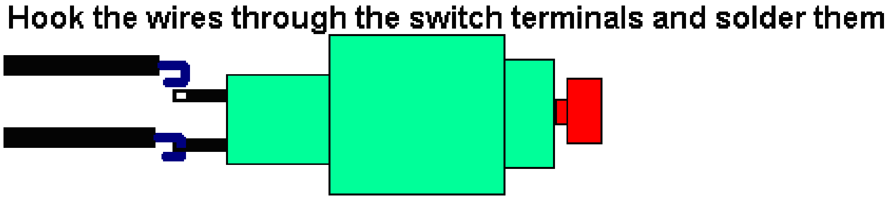
4.3 Soldering wires to switches

LED's and Switches are most often attached to the circuit board with wires. These must be correctly measured, cut, stripped and soldered.

Step 1:



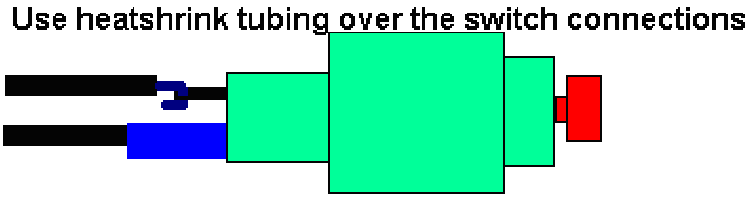
Step 2:

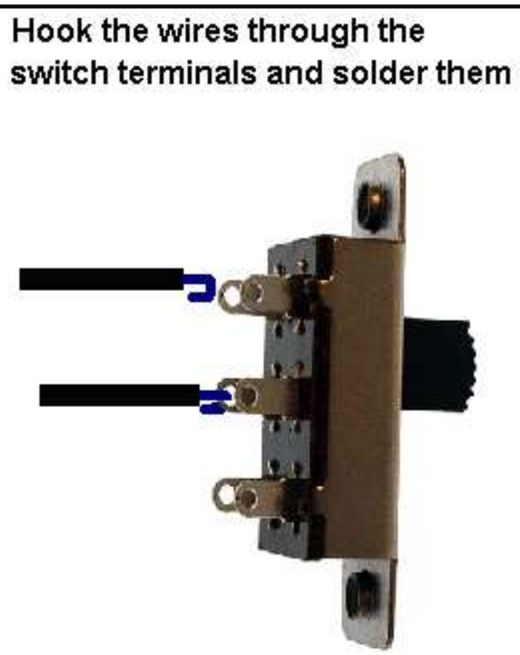


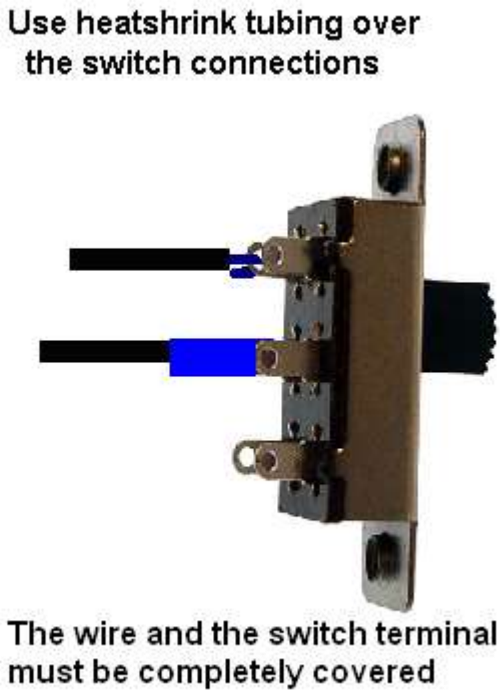
Step 3:

GET YOUR SOLDERING CHECKED
The solder should cover the joint fully, and after the joint has cooled the wire should not be able to move in the switch contact.

Ztep 4:







Follow these recommended codes of practice with your work

4.4 Codes of practice

Codes of practice are industry recognized ways of carrying out work on your project, so that it is safe for users and provides reliable operation. But how important are they?

This metal strip is a “wear strip”, it should have been made from stainless steel but was however made from titanium which is much stronger. A “wear strip” is a sacrificial metal strip that protects an edge on an aircraft; it is designed to be worn away with friction.

This titanium strip was a replacement part on a Continental Airways DC-10 aircraft. It was also not properly installed. The strip fell off the DC10 onto the runway at Charles de Gaulle airport, north of Paris on July 25, 2000.

The next aircraft to take off was an Air France Concorde. Before a Concord takes off the runway was supposed to be inspected and cleared of all foreign objects, this was also not done. The aircraft picked up the strip with one of its tires. The titanium strip caused the tire to burst, sending rubber fragments up into the wing of aircraft.

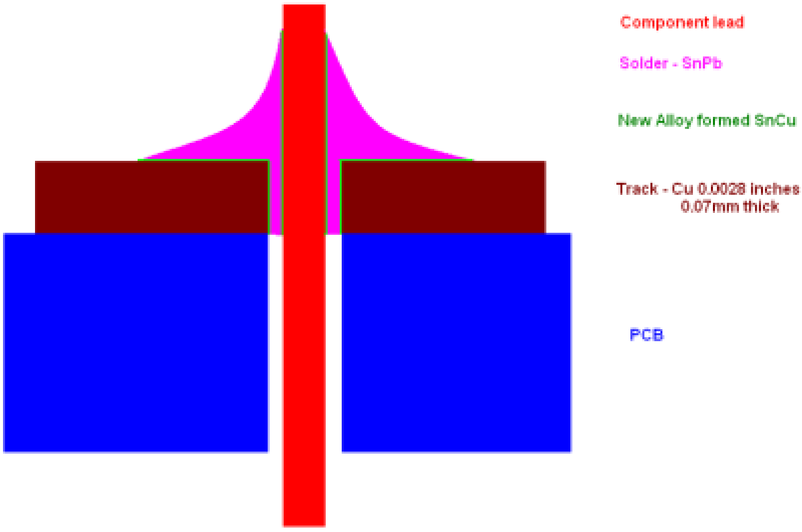
The aircraft stores its fuel in tanks in the wing. The wing is not very thick material and the tank burst open, the aircraft leaked fuel which ignited, sparking a bigger fuel leak and fire that brought the plane down.

The Air France Concorde crashed in a ball of flames 10km passed the runway, killing all 109 people aboard and four people at a hotel in an outer suburb of Paris.

Since the incident all Concorde aircraft have been retired from service, and in July 2008 it was determined that 5 people would stand trial for the crash.

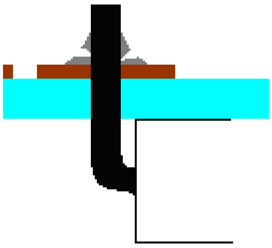
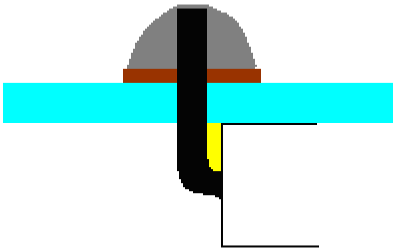
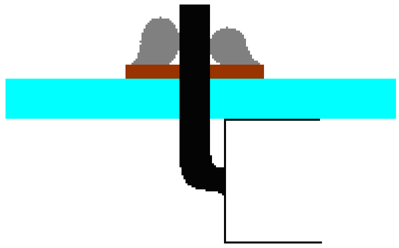
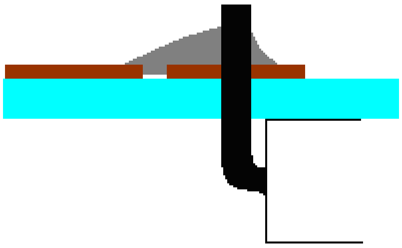
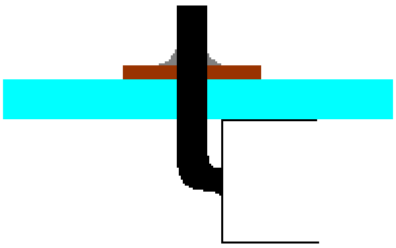
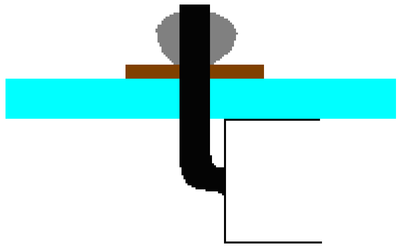
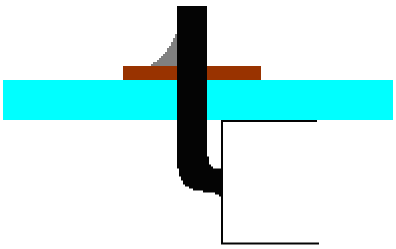
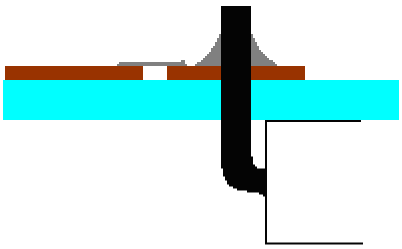
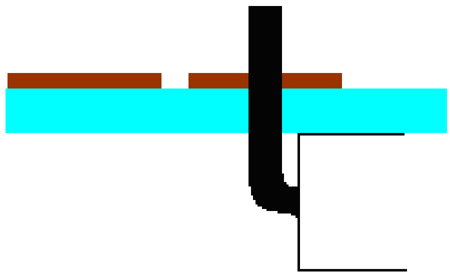
So how important are codes of practice?
So how important is your soldering?

4.5 Good and bad solder joints



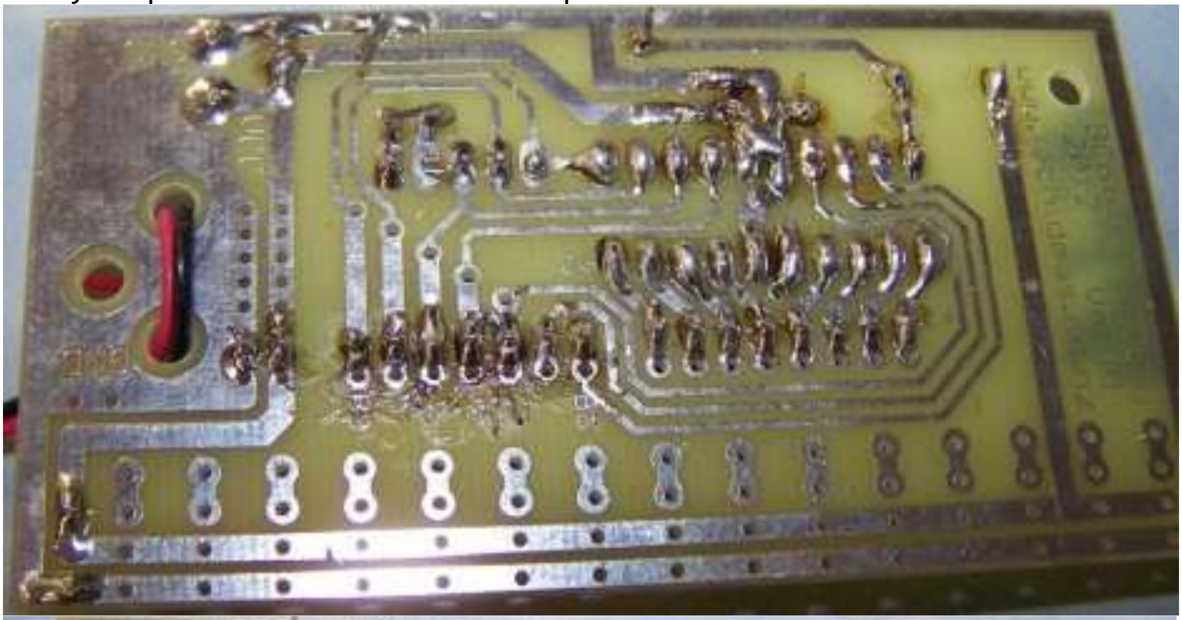
The finished solder joint should be cone shaped and bright in colour

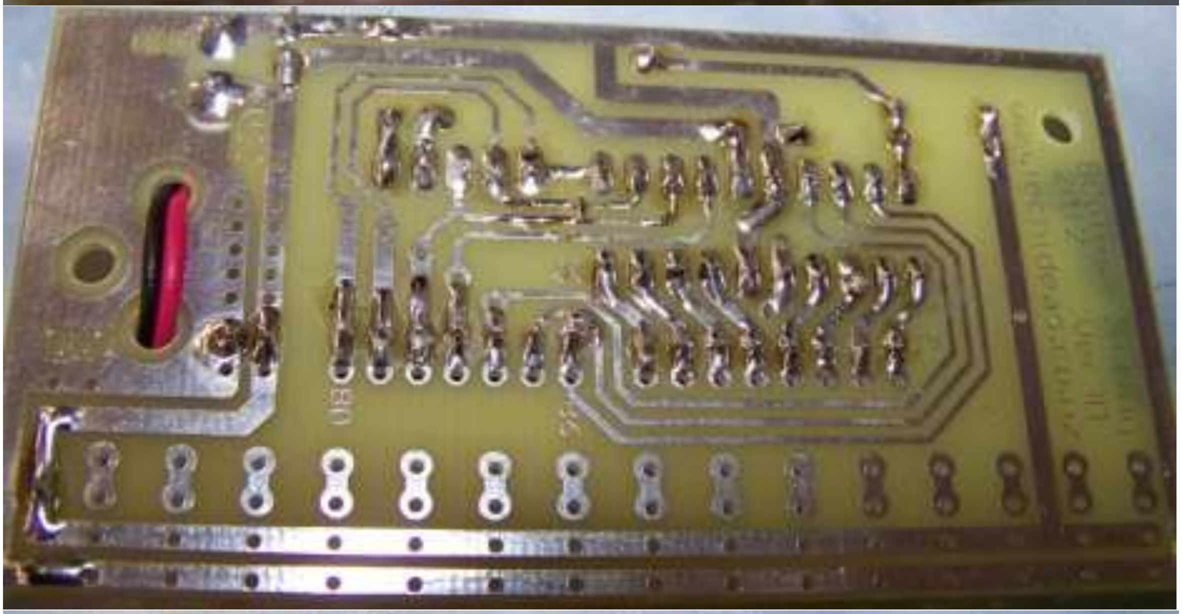
When a solder joint is correct there will be a new alloy od Sn-Cu formed between the solder and the track or component lead.

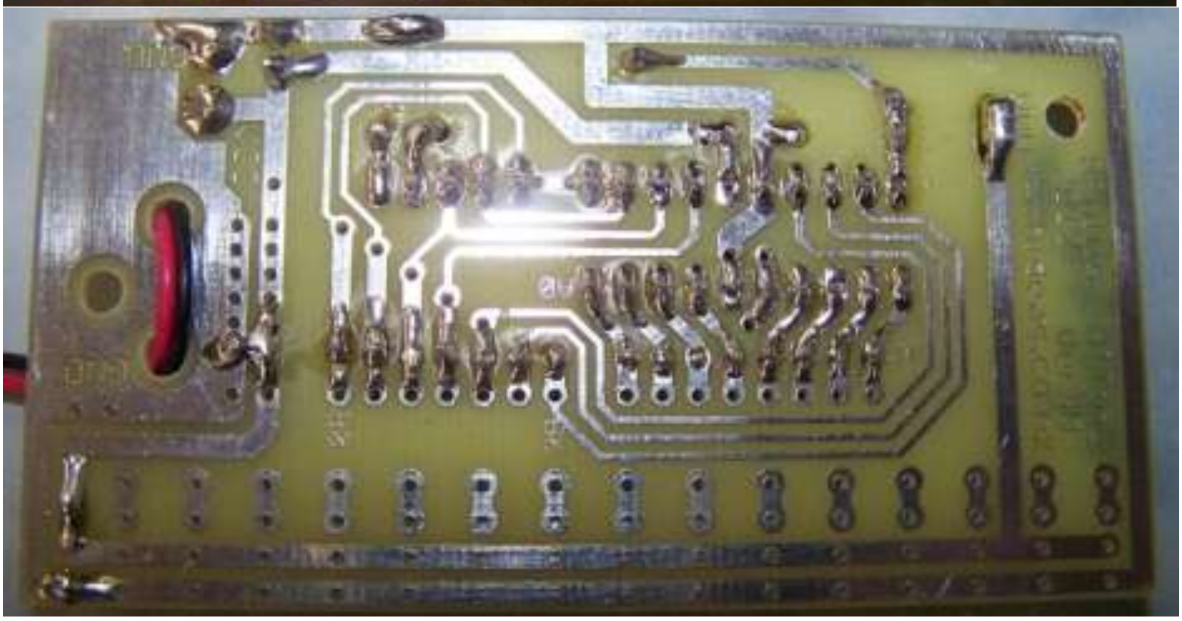
 Too little solder, not enough heat	 Too much solder	 Heated only the pcb track
 Too much solder, it has flowed onto another trac k	 Too little solder	 Heated only the leg of the component
 Only soldered on one side of the leg	 A whisker of solder is touching another track	 Forgot to solder it!!

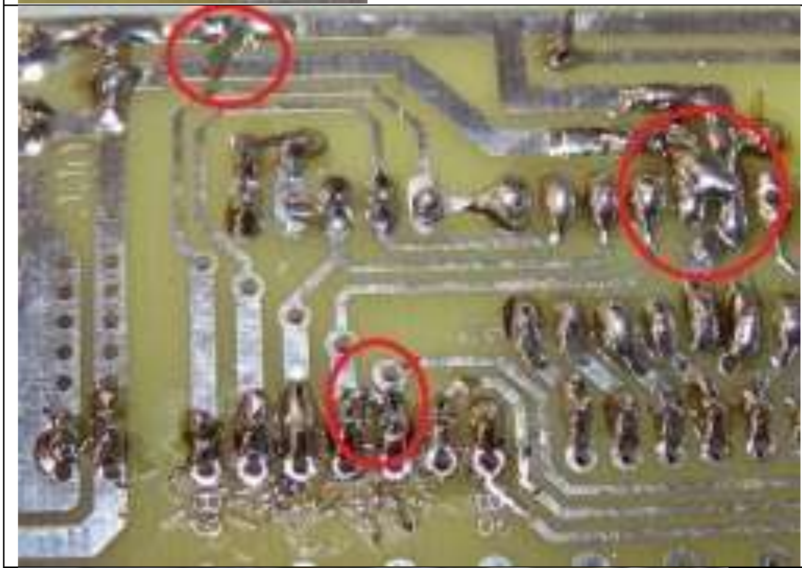
4.6 Short circuits

Can you spot the short circuits in these pictures?

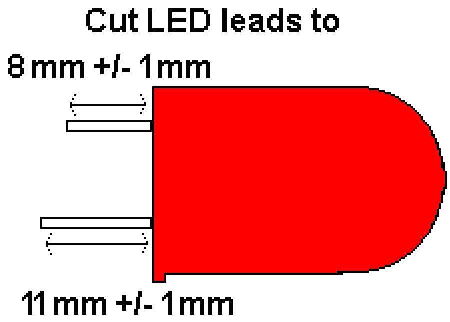
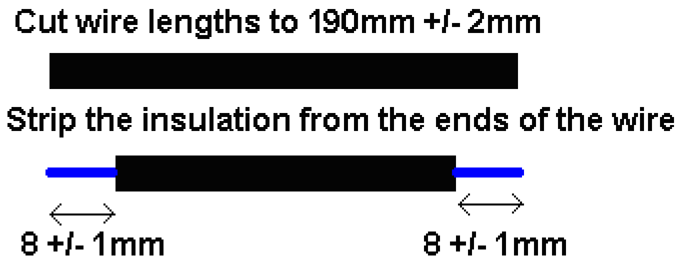
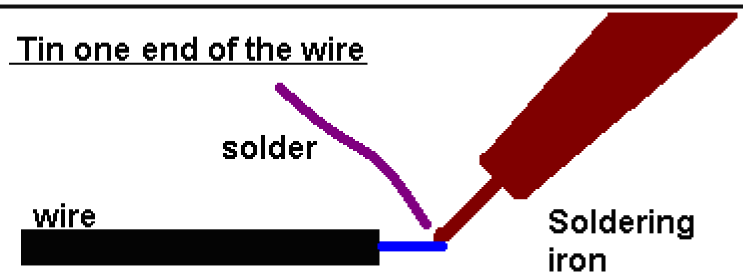


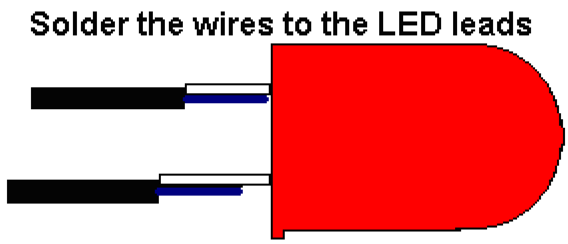
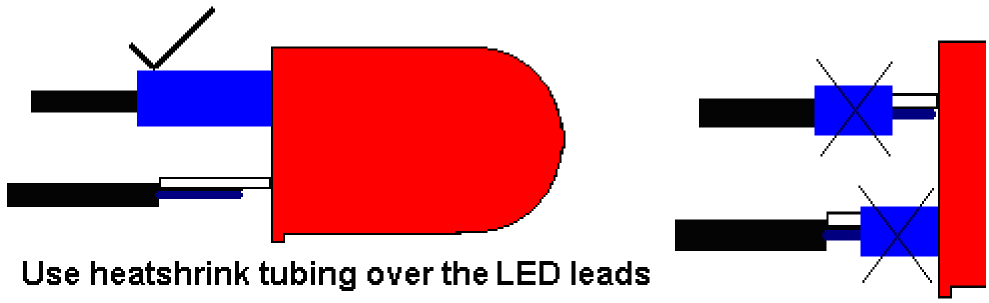




	Here the upper short circuit is between two of the tracks that connect to the programming pins, so the board wouldn't program. The lower short was noticed at the same time, but wouldn't havebecome a problem until either B5 or B6 were used
	Here there is a possible short at the top left as the wire hasn't been trimmed and bent over onto ro nearly onto the other track, the right hand short is between positive and negative, so the batteries were getting really hot!! Watch out, shorted batteries might actually burst into flame.
	Can you see the short between the battery connections here?

4.7 Soldering wires to LED's

<table>
<tr><td colspan="2">LED's and Switches are most often attached to the circuit board with wires. These must be correctly measured, cut, stripped and soldered.
To begin improving your accuracy practically keep to these measurements</td></tr>
<tr><td>Step 1:</td><td>
</td></tr>
<tr><td>Step 2:</td><td>
</td></tr>
<tr><td>Step 3:</td><td>

Only a small amount of solder, NO BLOBS!</td></tr>
<tr><td>Step 4:</td><td>
</td></tr>
<tr><td>Step 5:</td><td>GET YOUR SOLDERING CHECKED</td></tr>
<tr><td>Step 6:</td><td>CHECK THAT YOU GOT YOUR SOLDERING CHECKED!!!</td></tr>
<tr><td>Step 7:</td><td>

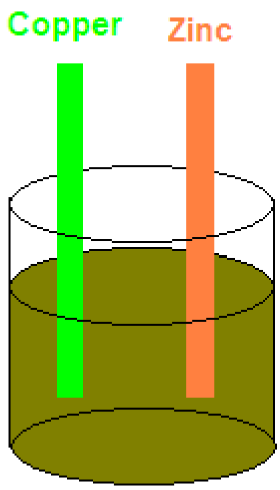
Heatshrink needs to provide BOTH mechanical and electrical cover!</td></tr>
</table>

5 Introductory Electronics Theory

5.1 Making electricity

<u>Electronic circuits need energy, this energy is in the form of moving charges(electrons)</u>

There are a number of ways that we can get charges moving around circuits.



- from chemical reactions (cells, batteries and the newer fuel cells),
- from magnets, wires and motion (generators and alternators),
- from light (photovoltaic cells),
- from friction (electrostatics e.g. the Van de Graaff generator),
- from heat (a thermocouple),
- from pressure (piezoelectric).

5.1.1 Cells

A cell is a single chemical container, and can produce a voltage of 1.1 volts to 2 volts depending on its type.

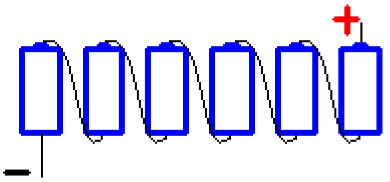
In the diagram on the copper side there are plenty of electrons(-), on the zinc side (+) there is an absence of electrons.

Here is a tomato cell powering an LCD clock.

Lemons make good cells too!

5.1.2 Batteries

<u>A battery is a collection of cells in series</u> e.g. a 12 volt car battery is six 2 volt lead-acid cells in series.

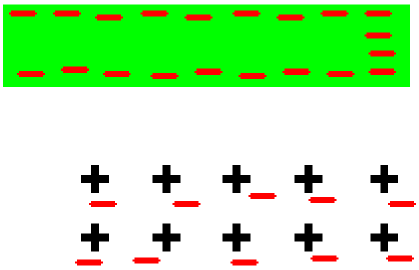


5.1.3 Different types of cells

- Primary cells (not rechargeable)
  - Zinc-carbon - inexpensive AAA, AA, C and D dry-cells and batteries. The electrodes are zinc and carbon, with an acidic paste between them that serves as the electrolyte.
  - Alkaline - Used in common Duracell and Energizer batteries, the electrodes are zinc and manganese-oxide, with an alkaline electrolyte.
  - Lithium photo - Lithium, lithium-iodide and lead-iodide are used in cameras because of their ability to supply high currents for short periods of time.
  - Zinc-mercury oxide - This is often used in hearing-aids.
  - Silver-zinc - This is used in aeronautical applications because the power-to-weight ratio is good.
- Secondary Cells (Rechargeable)
  - Lead-acid - Used in automobiles, the electrodes are made of lead and lead-oxide with a strong acidic electrolyte.
  - Zinc-air - lightweight.
  - Nickel-cadmium - The electrodes are nickel-hydroxide and cadmium, with potassium-hydroxide as the electrolyte.
  - Nickel-metal hydride (NiMh).
  - Lithium-ion - Excellent power-to-weight ratio.
  - Metal-chloride

5.1.4 Electrostatics

When certain materials such as wool and a plastic ruler are rubbed against each other an electric charge is generated. This is the principle of electrostatics.



The rubbing process causes electrons to be pulled from the surface of one material and relocated on the surface of the other material.

As the charged plastic moves over a piece of paper the electrons within the paper will be repelled (The paper is an insulator so the electrons cannot move far). This causes a slight positive charge on the paper.

This will mean that the negatively charged plastic will attract and pick up the positively charged paper (because opposite charges attract).

The positive side effects of Static Electricity

Smoke stack pollution control, Air fresheners, Photocopiers, Laser Printers, Car Painting,

The negative side effects of static electricity

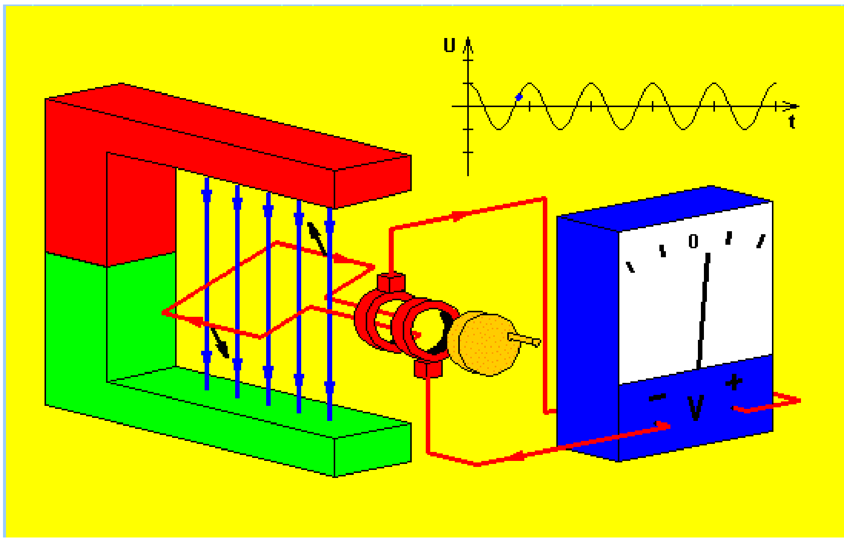
Lightning
Sparks from car – they hurt,
Damage/reduce life of electronic components
Danger around any flammable material (like at petrol stations)

5.2 ESD electrostatic discharge

A	Static dissipative mats/sheets	G	ST-Poly APET	M	Wet tissue with antistatic agent
B	Antistatic flooring	H	ESD shoes and slippers	N	Conductive adhesive mat
C	PCB Holders	I	Wrist straps and alarms	O	Wafer Transport System
D	Parts Boxes	J	Ion Blower	P	Static eliminators
E	Antistatic bags	K	Electrostatic tester		
F	Static dissipative seat cover	L	Grounding Wire & Conductive Steel Plate		

Ever got a shock getting out of a car? That is caused by a build up of static electronicty. Electronic components can be damaged by the high voltage of static electricity that we produce by waking around (we can easily generate several thousand volts). A large industry exists to provide anti-static devices to prevent static electricity from damaging electronic components.

5.3 Magnets, wires and motion



When a wire moves in a magnetic field electricity is produced. This picture shows the process of generating electricity from motion.

This mechanical torch has no batteries, this means that it will only generate electricity while the lever is being worked.

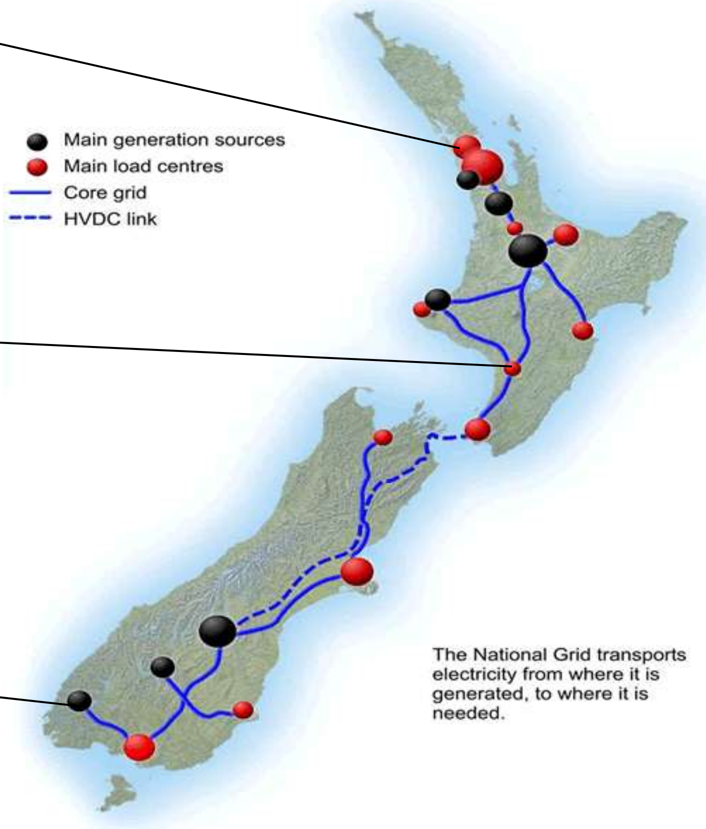
Turning the hand crack on the front of this radio will charge the internal rechargeable batteries. A one minute crank will give 30 minutes of listening; 30minutes of cranking will fully charge the batteries for 15 hours of listening

5.4 Group Power Assignment

In groups of six, choose one of the following each:
A. Power stations: Geothermal, Gas Fired, Hydro, Wind, Solar, Wave
Describe in detail its operation, typical uses, hazards, advantages and disadvantages, where it is used (if used) In New Zealand
B. Cells and Batteries
Zinc Carbon, Alkaline, Lithium, Lead Acid, NiCad, NiMh
Describe in detail its operation, typical uses, hazards, advantages, disadvantages

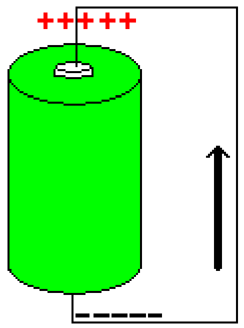
	Achieved	Merit	Excellence
Power Station technology	Diagram, location(s), some attempt at description of operation in own words	Pictures and Diagrams with clear descriptions of operation.	Thorough explanations and clear diagrams and pictures of working, sources are referenced.
Battery / Cell Technology	Diagram, location(s), some attempt at description of operation in own words	Pictures and Diagrams with clear descriptions of operation.	Thorough explanations and clear diagrams and pictures of working, sourc, explains mAH ratings, energy to weight ratio, sources are referenced
In your group you will need to agree on a common format for presentation: A2, A3 or Web, fonts, colours, layout. You will have 2 periods in class to work on this together. Please do not copy information straight from wikipedia or some other source, write the information in your own words.			

5.5 Electricity supply in New Zealand



In Auckland the mains power comes up from power stations in the south via over head lines that carry voltages of 220,000 Volts (220kV) at thousands of amps.

5.6 Conductors

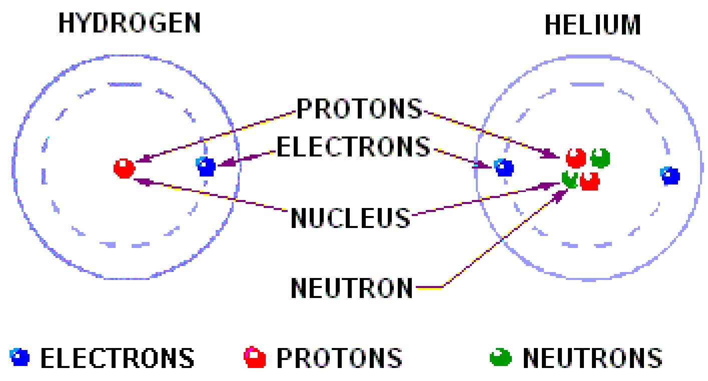


When a difference in energy exists in a circuit electrons (charges) want to flow from the negative to the positive.

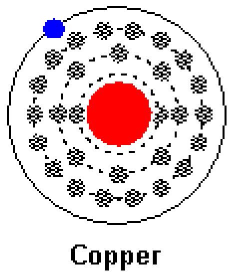
Materials that allow charges to flow freely are called conductors. Insulators are materials that do not allow charges to move freely.

Materials that have high conductivity are silver, gold, copper, aluminium, steel and iron.

To understand why these are good conductors some knowledge about atoms is required. Everything is made up of atoms or structures of atoms. Atoms themselves are made of a nucleus of protons and neutrons surrounded by numbers of electrons. The electrons spin around the nucleus. Electrons have a negative charge, protons a positive charge, neutrons no charge. The sum of all charges in a normal atom is zero making the atom electrically neutral.



The numbers of different neutrons, protons and electrons determine what type of material something is. With larger atoms the nucleus contains more protons and neutrons, and the electrons are arranged in layers or shells.



Less electrons in the outer shell means that a material is better at conducting.

A single electron in an outer shell on its own tends to be held weakly or loosely bound by the nucleus and is very free to move. This is shown in the copper atom. The atoms in the outer shell are known as Valence electrons

5.7 Insulators

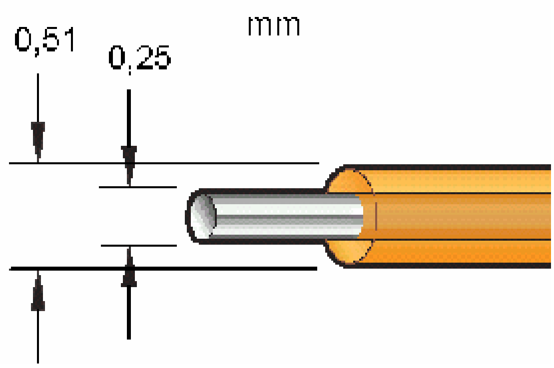
When the outer shell of an atom is full there are no free electrons, these tightly bound valence electrons make the material better at insulating, i.e. no current can flow.
Insulators are used in electronics just as much as conductors to control where current flows and where it doesn't.
An insulating material can break down however if enough voltage is applied.

5.8 Choosing the right wire

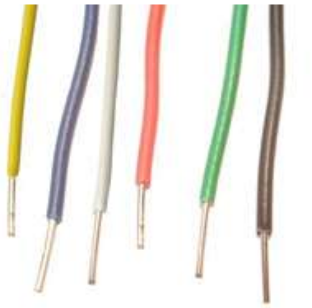
We use different types of wire for different jobs. Wires can be categorised by the number and diameter of the strands and whether they are tinned or not.

Collect samples of the different types of wire used in class, label each with the wires by its characteristics: e.g. single or multi-stranded, tinned or un-tinned and number and thickness of the strands.

Tinned single strand 0.25mm



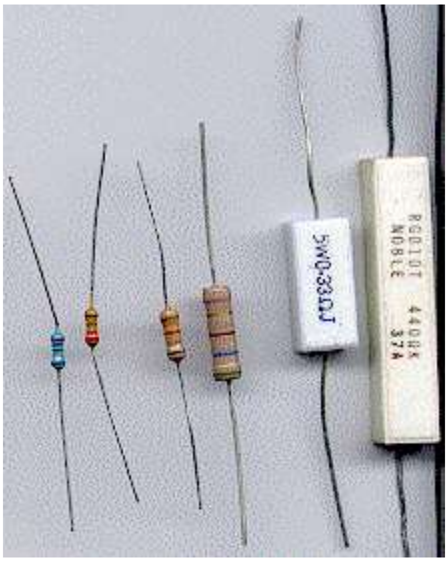
Solid core wire is really useful for breadboard use, but really bad for anytime the wire will be moved a lot as it breaks easily.



Multistranded wire is great for anytime the wire is moved, choose a thicker wire for high power.Tinned wire (looks like it has solder on it already) is great as it doesn't corrode/oxidise and so it is easier to solder.

7/0.2 wire we use a lot in classroom means 7 strands each 0.2mm in diameter, giving a total area of 0.22mm^2 . This can carry currents upto 1amp. We have thicker wire with more strands for higher current use.

NOTE: we use red for 5V, black for ground (0V/negative) and Yellow for voltages over 5V in the workshop.



5.9 Resistors

Resistors reduce the current (flow of electrons/charges) in a circuit.

The unit of resistance is ohms and the symbol is the Greek symbol omega. (Note that we often use the letter **R** on computers because an omega is harder to insert.)

Resistors can be variable in value (used in volume controls, light dimmers, etc) or fixed in value. Common fixed resistor types are Metal Film and Carbon Film.

5.10 Resistor Assignment

Write a description of how a metal film resistor is constructed.
Write description of how a carbon film resistor is constructed.
Include pictures with both.

5.11 Resistivity

Resisitivity is the measure of how a material opposes electrical current, it is measured in ohm-meters.

Silver	1.6×10^{-8} Ω/m	0.000000016 Ω-m	Silver cadmium oxide is used in high voltage contacts because it can withstand arcing, resists oxidation
Gold	2.44×10^{-8}		Used in sliding contacts on circuit boards, more corrosion resistant than silver, resists oxidation
Copper	$1.68\ 10^{-8}$		Electrical hookup wire, house wiring, printed circuit boards
Aluminium	2.82×10^{-8}		Used in high voltage power cables, it has 65% of the conductivity by volume of copper but 200% by weight
Tungsten	5.6×10^{-8}		High melting point so good for lightbulbs
Iron	1×10^{-7}		Used to make steel
Tin	1.09×10^{-7}		Used in Solder
Lead	2.2×10^{-7}		Used in solder
Mercury	9.8×10^{-7}		Used in tilt switches, because it is liquid at room temperature
Nichrome	1×10^{-6}		Used in heating elements
Carbon	3.5×10^{-5}		Used in resistors
Germanium	4.6×10^{-5}		Was used in making diodes and transistors
Seawater	2×10^{-1}		
Silicon	6.4×10^{2}	640 Ω-m	Used as the main material for semiconductors
pure water	2.5×15^{5}		Doesn't conduct!
Glass & porcelain	1×10^{10}		Used in power line insulators
Rubber	1×10^{13}		Insulating boots for electrical workers
Quartz (SiO4)	7.5×10^{17}		silicon–oxygen tetrahedral -used for its piexo electric properties
PTFE (Teflon)	1×10^{24}		Polytetrafluoroethylene, insulation for wires

5.12 Resistor prefixes

Some common resistor values are 1k (1,000) 10k (10,000) 1M (1,000,000) 2k2 (2,200) 47k (47,000).
Conversions between, ohms, kilo and Mega are very important in electronics.
So how do you remember that 1 kOhm = 1000Ohms or 22,000 Ohms = 22k?
First know that the prefixes are normally in groups of thousands and secondly writing them into a table helps.

Giga	Mega	kilo		milli	micro	nano	pica
G	M	k	R	m	u	n	p
	1	0 0 0	0 0 0				
		2 2	0 0 0				
			0 1 4				
			1 8	2 0 0			
2 0	0 0 0	0 0 0					

1Mohm = 1,000,000 ohms
22k ohms = 22,000 ohms

2.2 ohm = 2R2 ohms
4,700 = 4k7 ohms

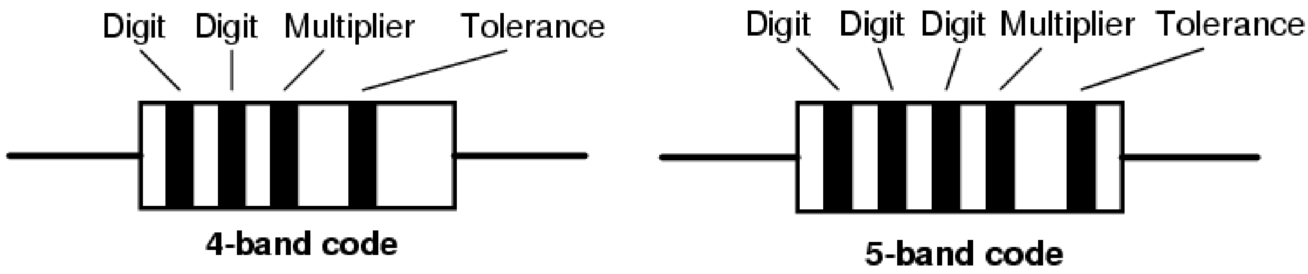
Every conversion in in groups of three or thousands so decimal points and commas can only go when lines are shown on the table Note the special case in electronics where we use 2k2 not 2.2K. The reason for this is that when a schematic or circuit diagram is photocopied a number of times then the decimal point may disappear leaving 2.2 as 22. This cannot happen when using 2k2 (2,200), 2R2 (2.2) or 2M2 (2,2000,000).

Convert the following:

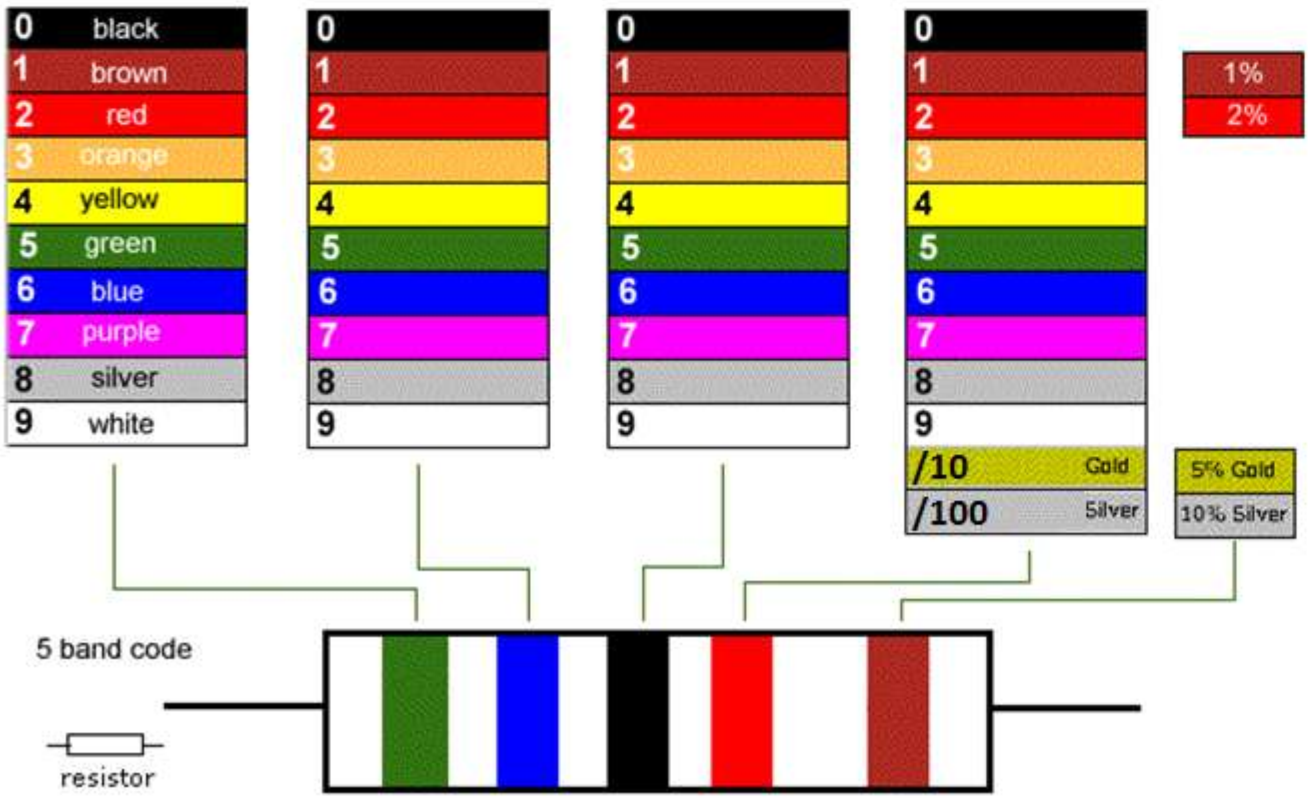
Ohms	Correctly formatted
1500	1K5
5,600,000	5M6
3,300	3k3
12.5	12R5
9,100,000	
22,000	
4,700	
5.6	
10,000	
9100	
1.8	
22,400	
10.31	
100,000	
1000k	
4,300,000	
0.22	
3,900K	
91,000	
3.1k	

5.13 Resistor Values Exercises

Resistor values are normally shown on the body of the resistor using colour codes
There are 2 schemes, one with 4 bands of colour and one with 5 bands of colour

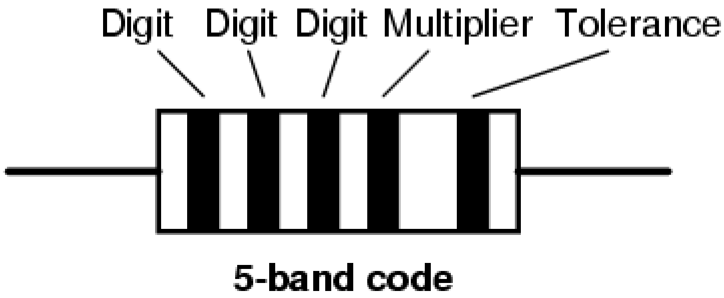


The colour code is



You will need some practice at using this table.

Here are some common values

 	1st band, Bn = 1 2nd band, R = 2 3rd band, BK = 0 4th band, Y = 0000 (4 zero's) 5th band, Bn = 1% Answer: 1,200,000 ohm, +/-1% = 1M2
1st band: Y_____ 2nd band: Pu_____ 3rd band: Bk_____ 4th band: Bk_____ 5th band: Bn_____ Answer:	1st band: BN_____ 2nd band: Bk_____ 3rd band: Bk_____ 4th band: Bk_____ 5th band: Bn_____ Answer
1st band: BN_____ 2nd band: Bk_____ 3rd band: Bk_____ 4th band: BN_____ 5th band: Bn_____ Answer:	1st band: Or_____ 2nd band: Or_____ 3rd band: Bk_____ 4th band: R_____ 5th band: Bn_____ Answer
1st band: BN_____ 2nd band: Bk_____ 3rd band: Bk_____ 4th band: R_____ 5th band: Bn Answer:	1st band: Or_____ 2nd band: Wh_____ 3rd band: Bk_____ 4th band: Bk_____ 5th band: Bn_____ Answer:
1st band: Gn_____ 2nd band: Bu_____ 3rd band: Bk_____ 4th band: Bk_____ 5th band: Bn_____ Answer:	1st band: Bn_____ 2nd band: Bk_____ 3rd band: Bk_____ 4th band: Gold_____ 5th band: Bn_____ Answer:
1st band: BN_____ 2nd band: Bk_____ 3rd band: Bk_____ 4th band: Silver_____ 5th band: Bn_____ Answer:	1st band: Y_____ 2nd band: Pu_____ 3rd band: BK_____ 4th band: Gold_____ 5th band: Bn_____ Answer:

Find the colour codes for the following resistors (5 band)
1K2 1% (1,200 ohms = Bk – Rd – Bk – Bn __ Bn)

18k 1%

4M7 1%

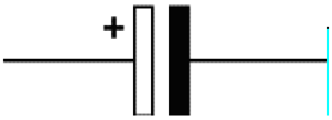
8K2 1%

5.14 Capacitors

There are two different symbols for the two main categories of capacitors

and many several types within each category

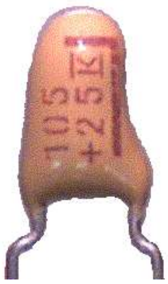
Polarised



such as an electrolytic

Note the <u>25V voltage rating</u> on the above electrolytic and the 16V rating on the one below, all capacitors are rated up to a particular voltage, exceeding this may cause the capacitor to <u>overheat leak and even **explode**</u>!

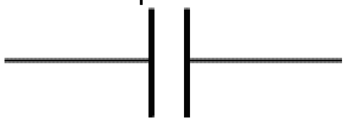
and tantalum



Values will be written on these capacitors, generally in microfarads (uF)

They are polarised (have a positive and a negative legs) – the positive leg is the longer one and there is a line on the body of the capacitor to show whcihc side is negative.

non polarised



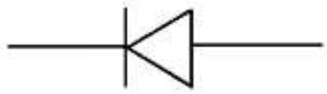
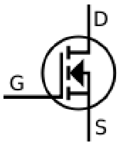
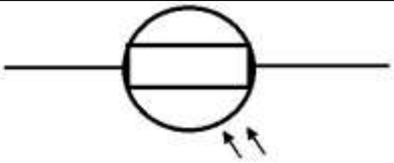
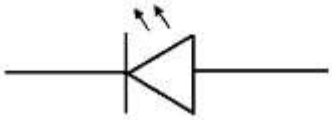
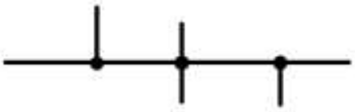
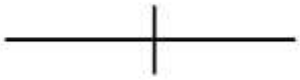
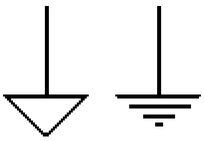
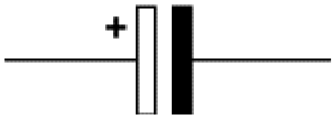
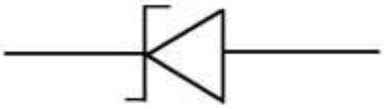
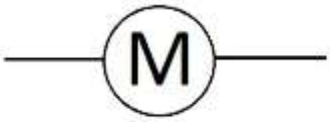
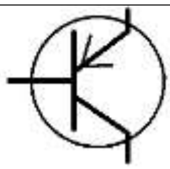
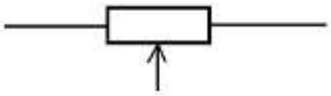
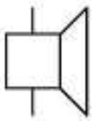
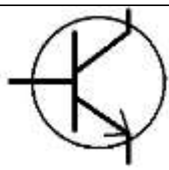
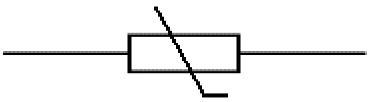
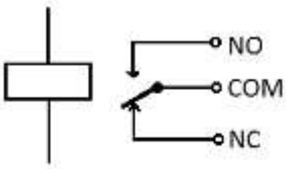
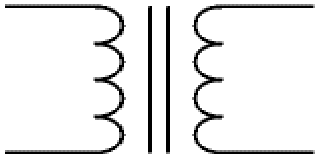
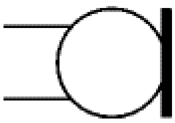
such as ceramic

Values will be written on these capacitors, generally in picofarads and in code
104 = 100,000 pF
(means 10 + 4 more zeros)
The main one of these we use in the workshop will be the 0.1uF = 100nF = 100,000pF

There are polyester types as well

5.15 Component symbols reference

Get to know the first 11 of these straight away

	**Resistor**		**Diode**
	FET		**Battery**
	**LDR - Light Dependent Resistor**		**LED - Light Emitting Diode**
	**Wires – joined (junctions used)**		**Wires – unjoined (no junction)**
	**Switch**		**Capacitor (non polarised type)**
	**Ground, Earth or 0V Capacitor**		**Capacitor (polarised type e.g.electrolytic)**
	Zener Diode		Motor
	PNP Transistor		Variable Resistor (or Potentiometer)
	Speaker		**NPN Transistor**
	Thermistor (senses temperature)		Piezo or crystal
	Relay		Transformer
	Microphone		

5.16 Year 10/11 - Typical test questions so far

Darkness Detector

1. What are the color codes for all the resistors used in the darkness detector?
2. Draw the circuit for the darkness detector
3. What is the diode for?
4. Draw a breadboard with a resistor, LED, switch and battery connected so that the LED lights up?
5. How can you tell the right way to put in an LED?
6. What is your electronics teachers favourite type of chocolate?
7. What does LED stand for?
8. What does LDR stand for?
9. When a switch is turning a circuit on and off what is it actually doing?
10. What is the LDR for?
11. What components make up the input part of the circuit?
12. What components make up the output part of the circuit?
13. What components make up the process part of the circuit?
14. What components make up the power supply part of the circuit?

Soldering

15. What is solder made of?
16. What is flux for?
17. What temperature is a soldering iron?
18. What is a code of practice?
19. Think of at least one terrible thing that could go wrong due to poor soldering
20. Why must the sponge be damp but not wet?
21. Describe three types of bad solder joints
22. Describe a good solder joint
23. Why do we put heatshrink over wires?

General electronic theory

24. What is current?
25. Where does electricity come from in NZ?
26. What is the voltage of a AA cell?
27. When is static electricity bad?
28. Does current flow in a circuit? (trick question!)
29. Why do some things conduct and others not?
30. Name three conductors used in electronics.
31. What are some different types of wire and where do we use each one?
32. Use a resistor colour code table to find the values of 3 different resistors used in the workshop.
33. Draw and name the first 11 symbols in the symbol table

    .

6 Introduction to microcontroller electronics

Microcontrollers are a fundamental electronic building block used for many solutions to needs throughout industry, commerce and everyday life.

They are found inside aircraft instruments.

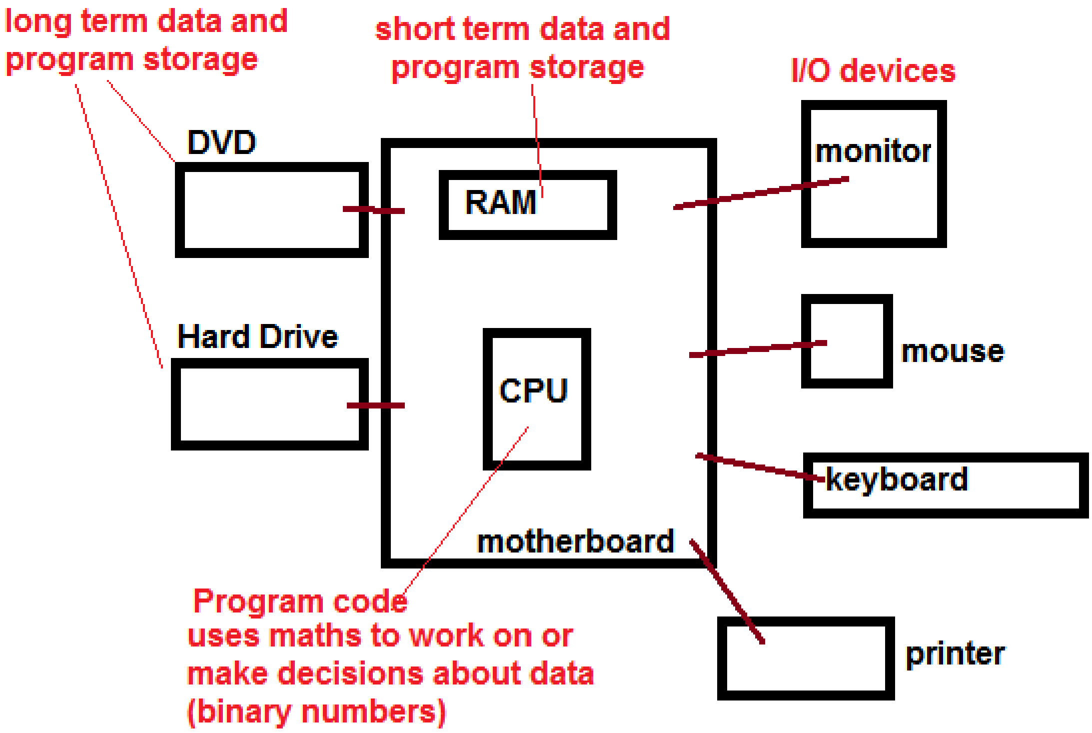
They are used extensively within cellular phones, modern cars,

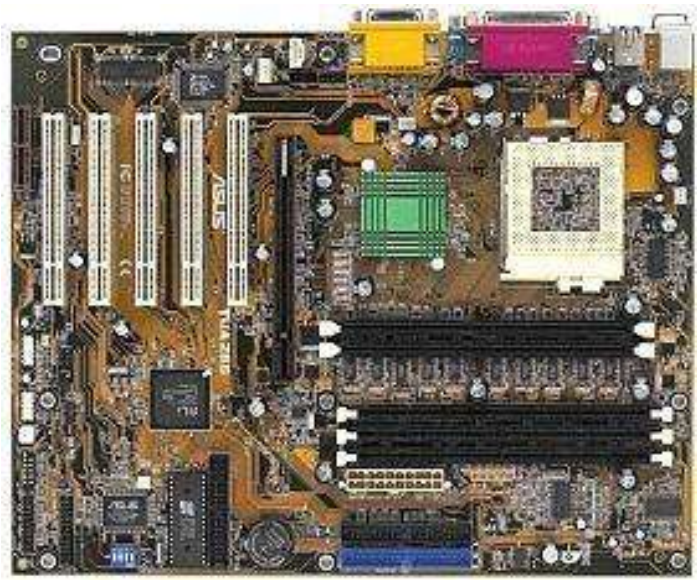
domestic appliances such as stereos and washing machines

and in automated processes through out industry

6.1 What is a computer?

A computer system that we are familiar with includes components such as DVD writers, hard drives, a motherboard which has a CPU, RAM and other things on it, and a bunch of I/O devices connected to it.





6.2 What does a computer system do?

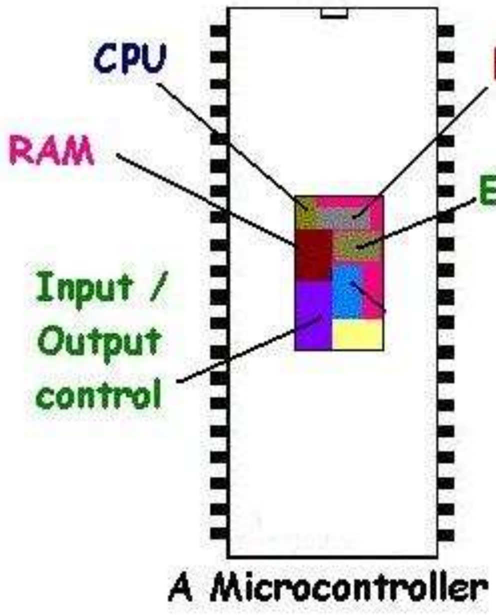
A computer carries out simple maths on data.

Data is information which is input from I/O devices and stored inside the computers memory devices in the form of binary numbers.

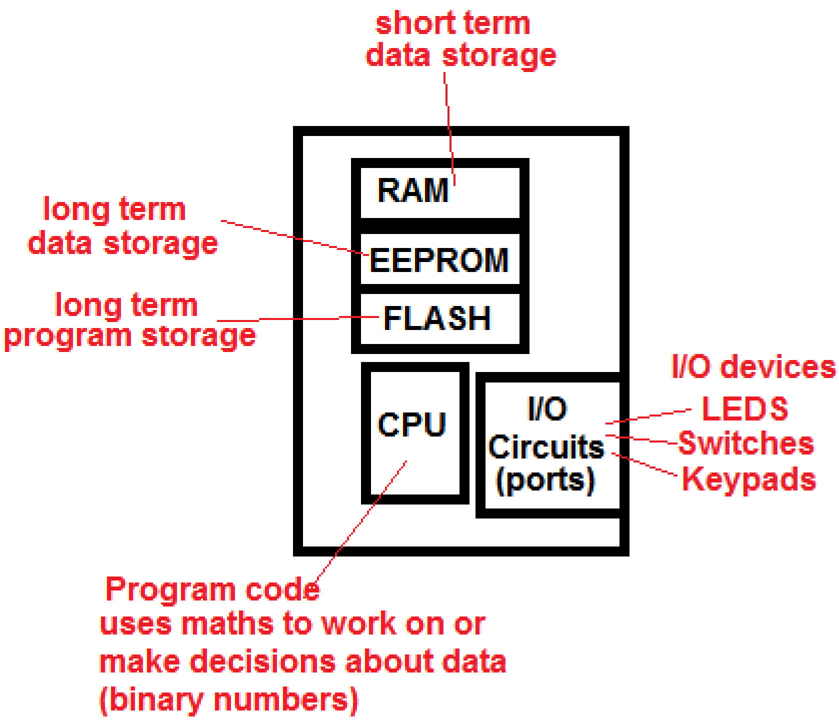
But don't computers do complex things? Yes, but as you will learn, the art of computer science is to break big complex tasks down into a lot of simple tasks.

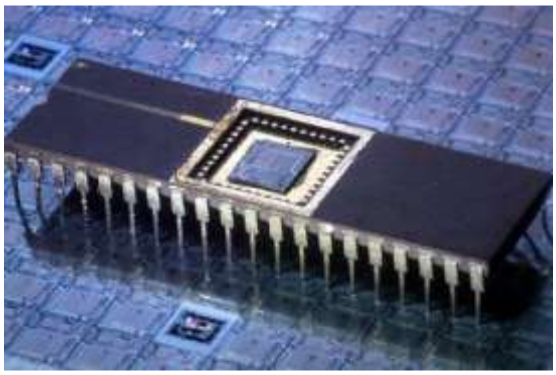
6.3 What exactly is a microcontroller?

A microcontroller has the same things in it that bigger computers have, data and program storage, I/O control circuits and a CPU (cental processing unit) however it is inside a single IC package.



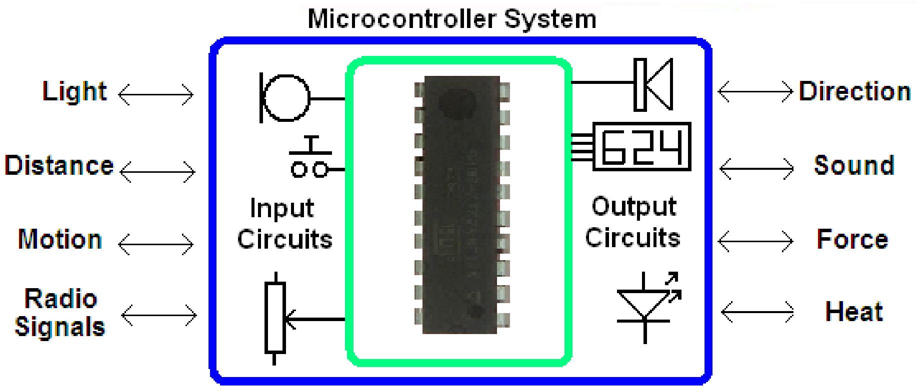
The purpose of the parts of a microcontroller are exactly the same as in a larger computer. Data and programs are stored in memory and a CPU carries out simple maths on the data.



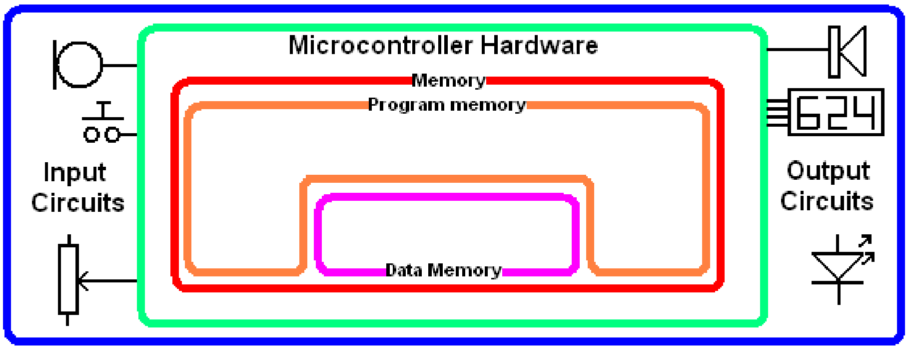


However don't think that because a microcontroller is smaller than a PC that it is the same comparison as between a real car and a toy car. The microcontroller is capable of carrying out millions of instructions every second. And there are billions of these controllers out there in the world doing just that. You will find them inside cars, stereos, calculators, remote controls, airplanes, radios, microwaves, washing machines, industrial equipment and so on.

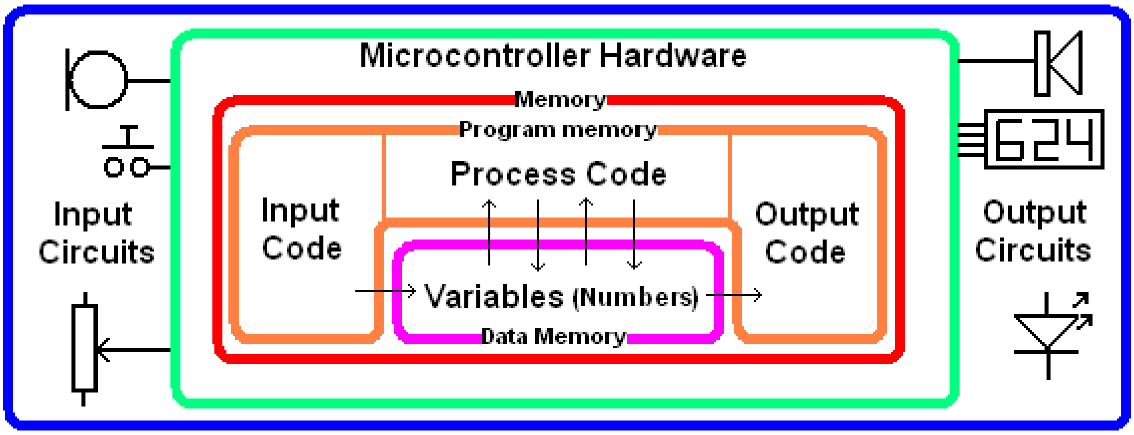
6.4 What does a microcontroller system do?



As with any electronic circuit the microcontroller circuit is a system with three parts,: INPUT, PROCESS (or CONTROL) and OUTPUT. Input circuits convert physical world properties to electrical signals (current/ voltage) which are processed and converted back to physical properties (heat, light etc)



In a microcontroller there is a second conversion, where the electrical properties of voltage and current are changed to data and stored in memory. The programmer writes programs (program code) which are made up of input instructions (convert electrical signals from input circuits to data), control instructions (which work on data) and output instructions (convert data to electrical signals)



1. Input circuits convert light, heat, sound etc to voltages and currents.
2. Input instructions convert the electronic signals to data (numbers) and store them in its data memory (RAM) – A variable is the name for a RAM location.
3. The processor runs a program which carries out mathematical operations on data or makes decisions about the data
4. The output code converts the data (numbers) to electronic signals (voltage and current).
5. Output circuits convert electronic signals to light, heat, sound etc

In a microcontroller circuit that creates light patterns based upon sounds the control process is

SOUND to ELECTRICITY to DATA

Processing of the DATA (numbers)

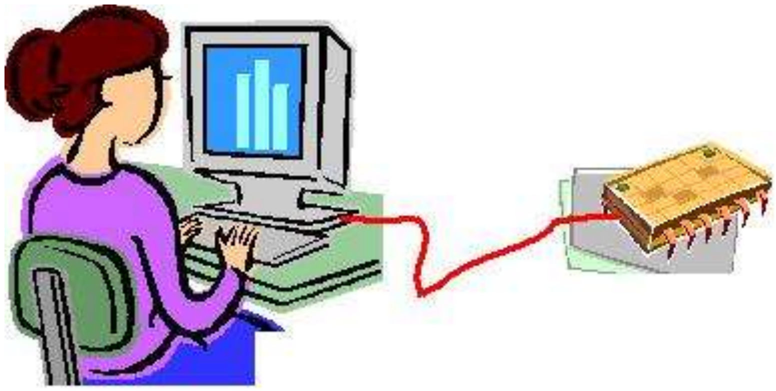
DATA to ELECTRICITY to LIGHT

6.5 Getting started with AVR Programming

Microcontrollers, such as the ATMEL AVR, are controlled by software and they can do nothing until they have a program inside them.

The AVR programs are written on a PC using BASCOM-AVR.
This software is a type of computer program called a compiler, it comes from www.mcselec.com. It comes in a freeware version so students may download it and use it at home.

The AVR is connected to the PC with a 5 wire cable.

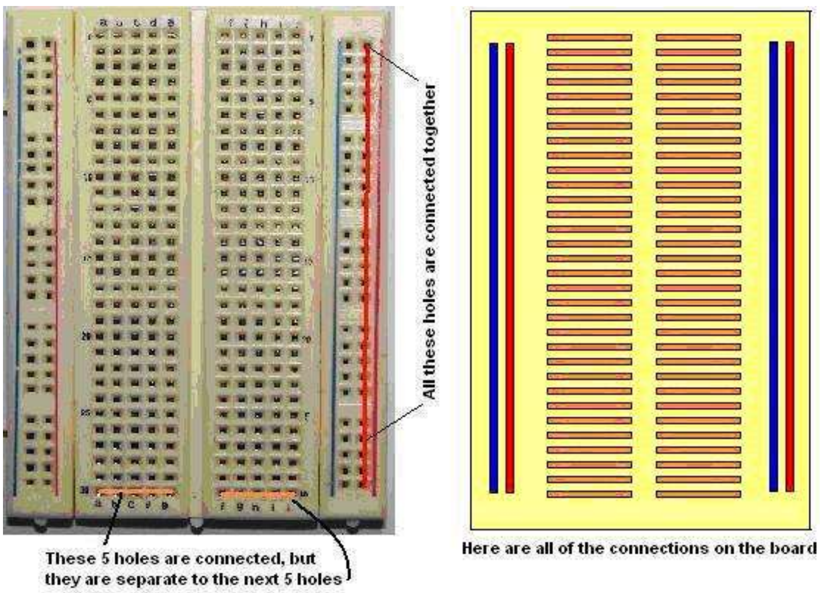


6.6 Breadboard

Often in electronics some experimentation is required to prototype (trial) specific circuits. A prototype circuit is needed before a PCB is designed for the final circuit.

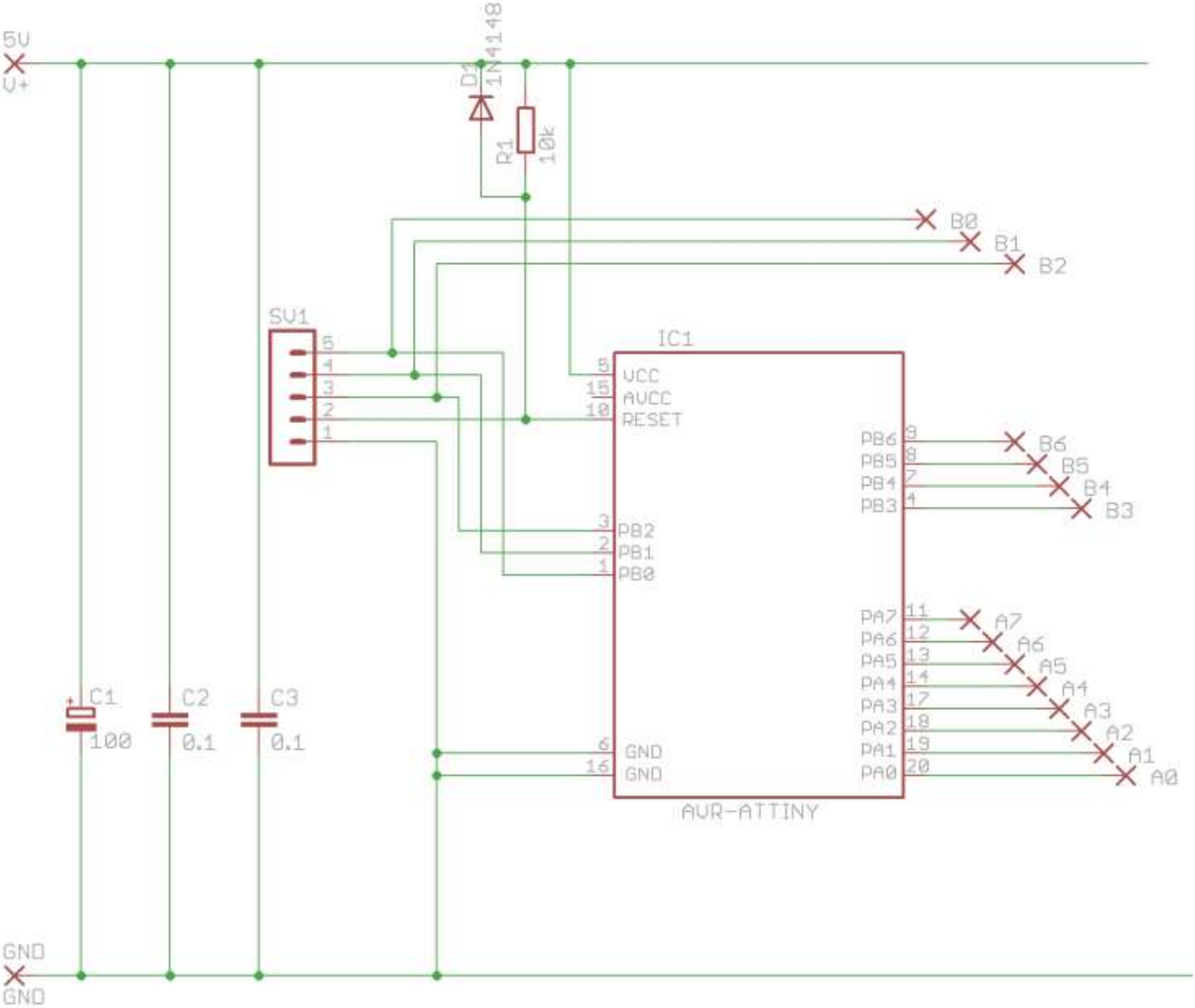
A breadboard can be used to prototype the circuit. It has holes into which components can be inserted and has electrical connections between the holes as per the diagram below.

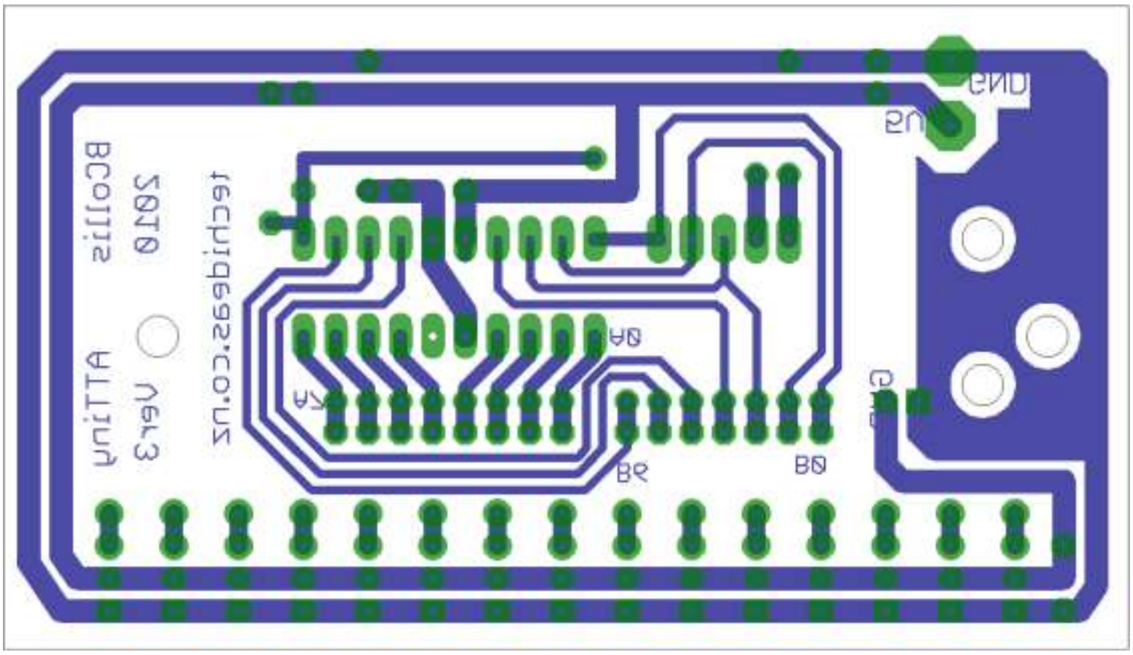
Using a breadboard means no soldering and a circuit can be constructed quickly and modified easily before a final solution is decided upon.



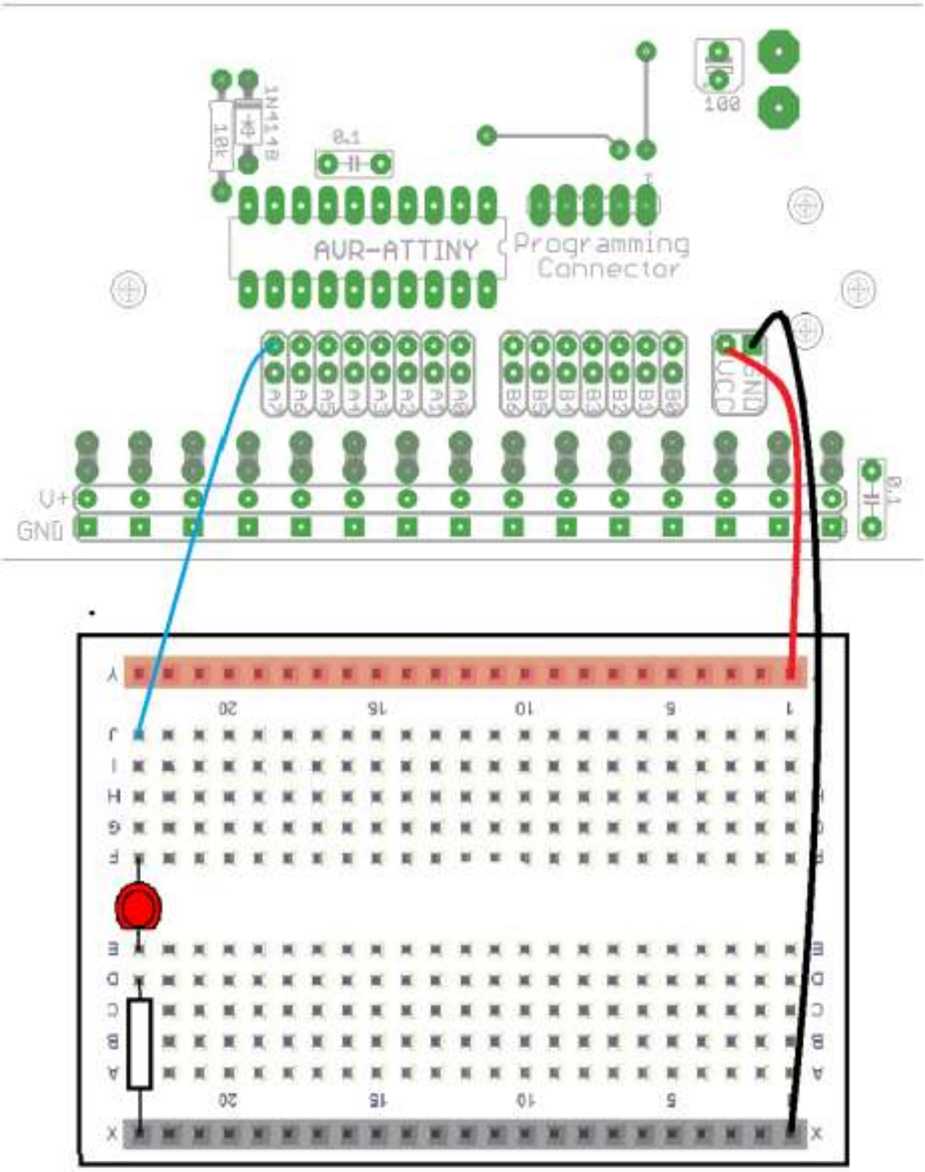
6.7 Breadboard+Prototyping board circuit

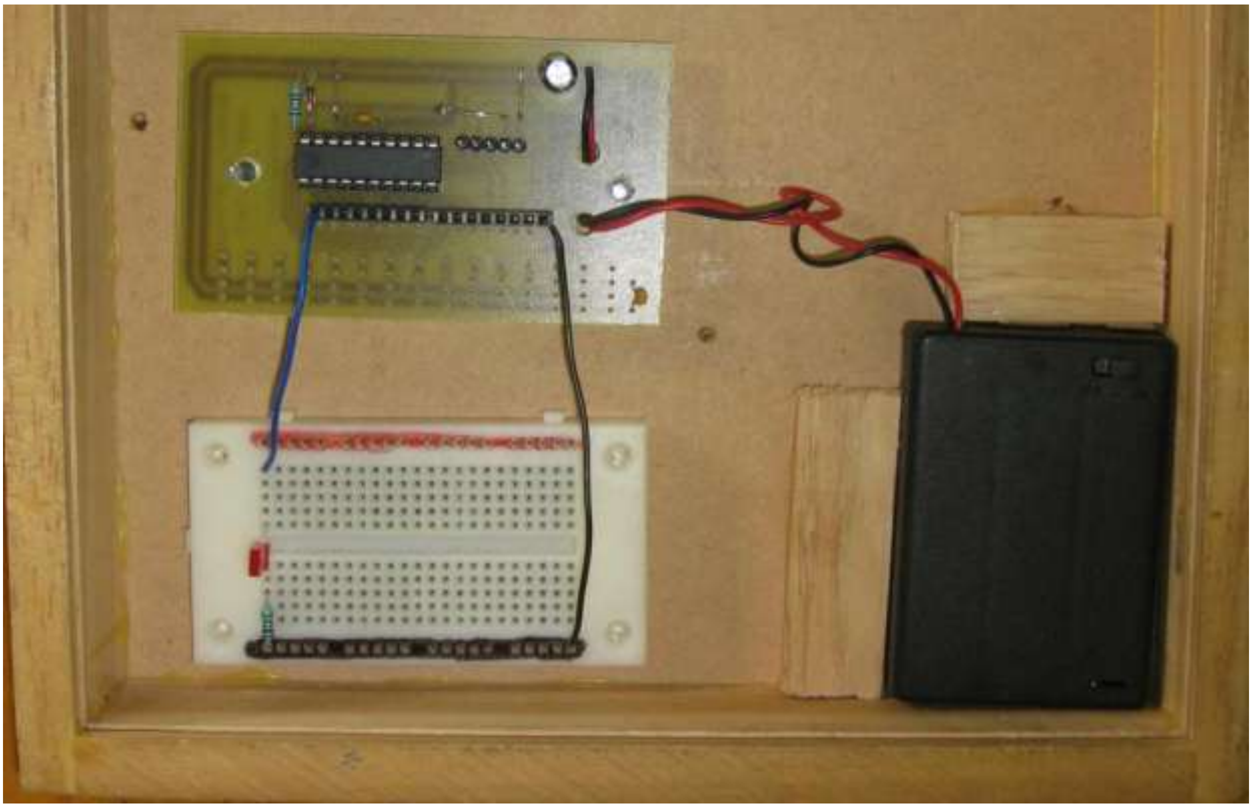
This prototyping board along with a breadboard works well for trialling circuits.





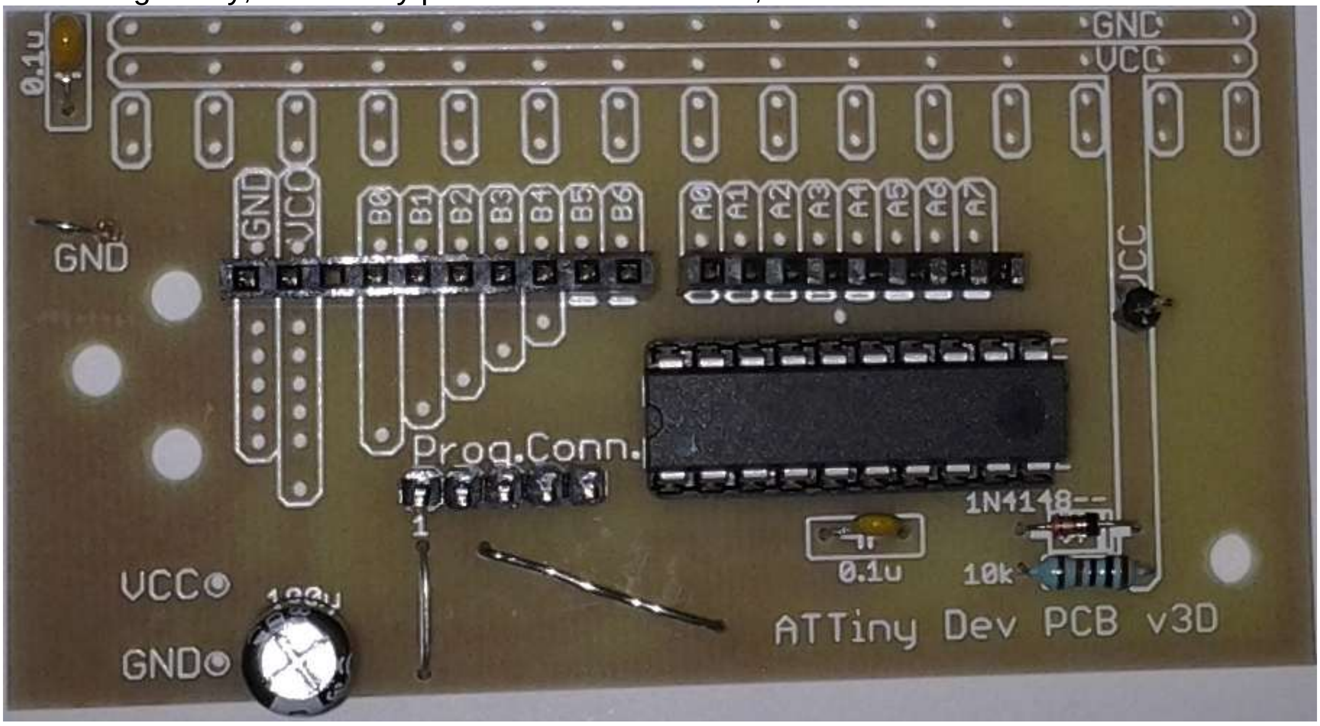
On this breadboard a single LED has been setup along with the ground wire to complete the circuit.



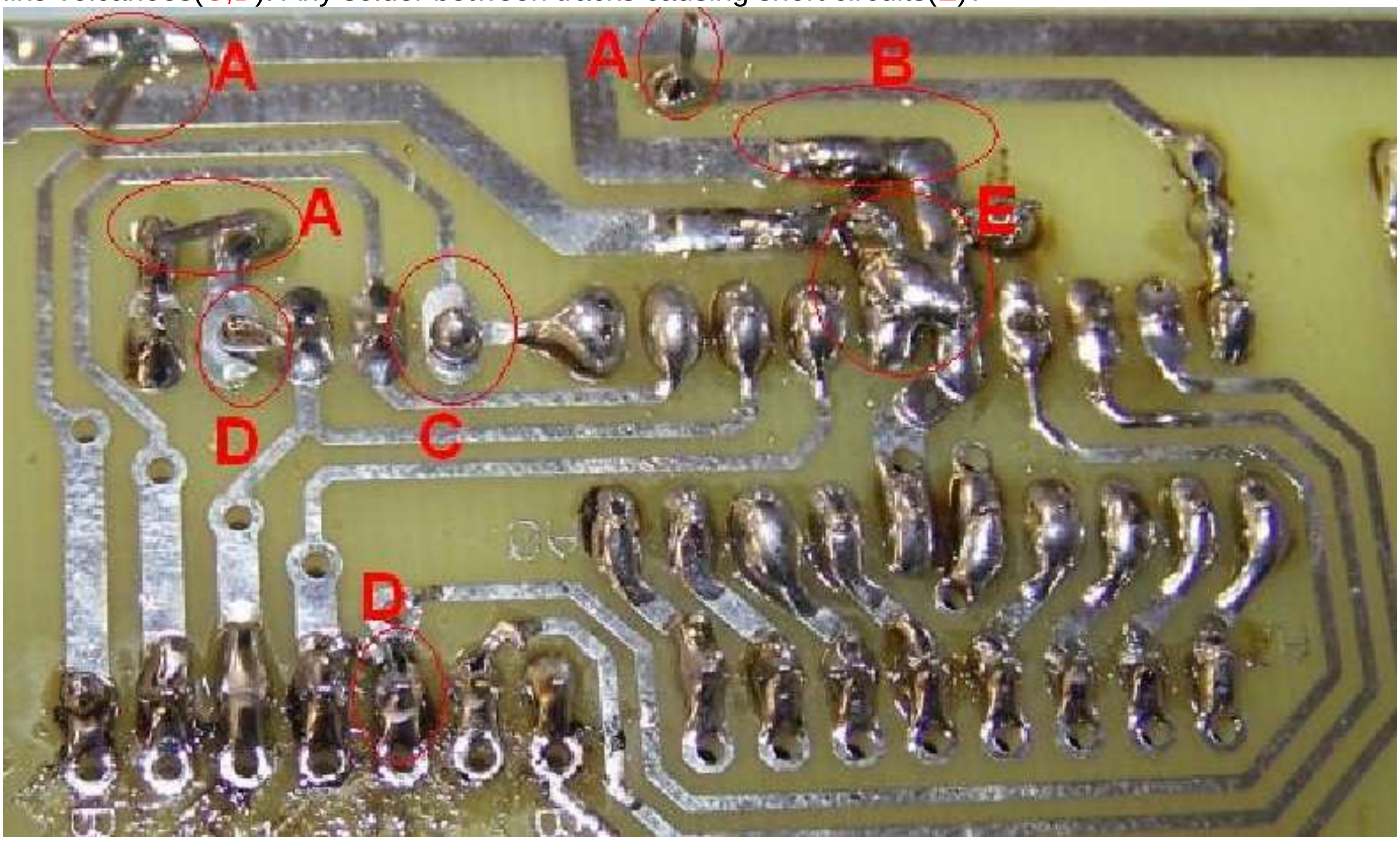


6.8 Checking your workmanship

Check your workmanship, if you fins any problems it is a good idea to ask the teacher what to do to fix it, you don’t want to damagee the board while trying to fixing it. Check all the following:
The value of the resistor is 10K, the diode is the right way around, the IC is in the right way, the two links are in, the 8 way and 10 way sockets are in the rows of holes closest to the IC, the Electrolytic capacitor is in the rght way, the battery pack red wire is in VCC, the black wire is in GND.

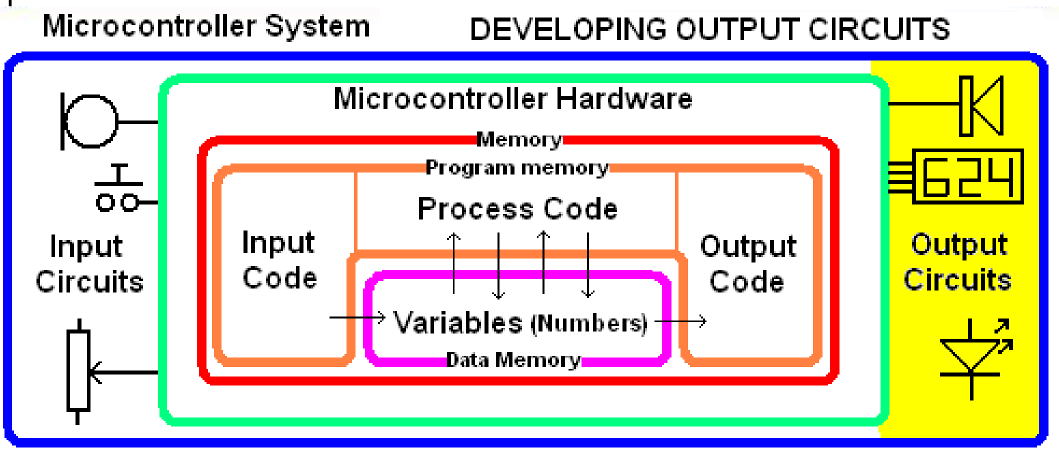


Us the soldering good enough? Are there long wires left uncut (A,B,C)? Any solder joints that don’t look like volcanoes(C,D). Any solder between tracks causing short circuits(E)?



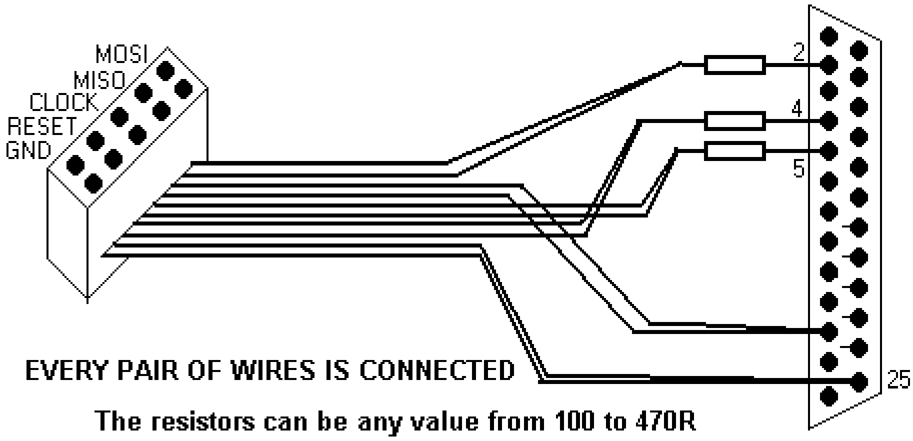
6.9 Output Circuit - LED

There is an LED with a 1k 'current limit' resistor. An LED needs only 2V to operate so if connected without a resistor in series too much current would flow and destroy the LED. With 2V across the LED, there will be 3V across the resistor, and the current will be limited to (V/R) 3/1000 = 3mA. This is enough current to make the LED clearly visible but not too much for the micro to provide.



6.10 AVR programming cable

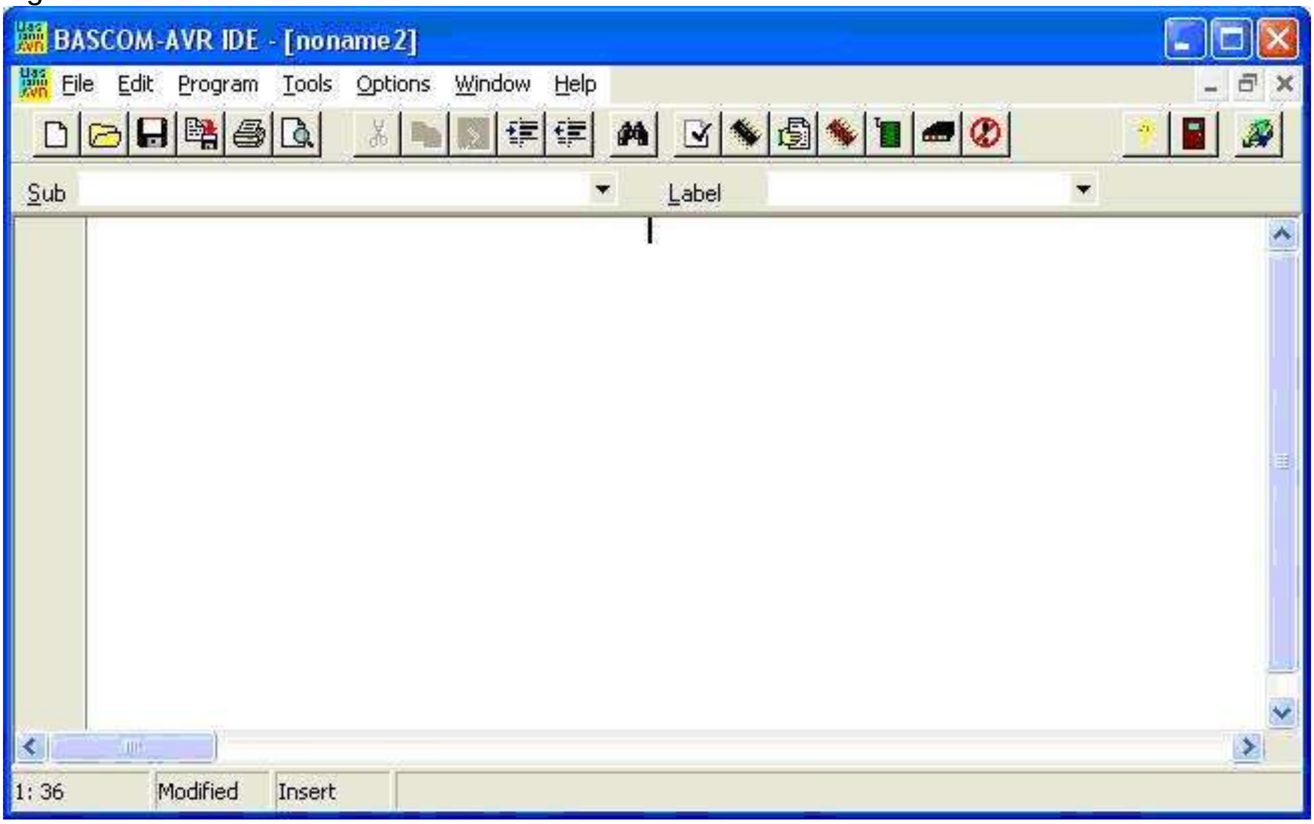
A five wire cable is needed to connect the AVR circuit to a PC.
It connects the PC's parallel port to the AVR circuit. One end has a DB25M connector on it (as in this picture)



The other end has a 10 way IDC socket attached to it (as in this picture). These were used because they are readily available even though only 5 conductors are required the 10 wires are connected to the DB25 in 5 pairs. Put heatshrink over the resistor connections to stop them shorting together.

6.11 Getting started with Bascom & AVR

BASCOM-AVR is **four programs in one package**, it is known as an IDE (integrated development environment); it includes the Program Editor, the Compiler, the Programmer and the Simulator all together.



After installing the program there are some set-up options that you might want to change.
If its not already setup from the menu select.
OPTIONS – **PROGRAMMER** and select **Sample Electronics programmer**. Choose the parallel tab and select LPT-address of 378 for LPT1 (if you only have 1 parallel port on the computer choose this), also select **autoflash**.
The following are not absolutely necessary but will help you get better printouts.
OPTIONS – PRINTER change the margins to 15.00 10.00 10.00 10.00
OPTIONS – ENVIRONMENT – EDITOR change the Comment Position to 040.

6.12 The compiler

The command to start the compiler is **F7** or the black IC picture in the toolbar.
This will change your high-level BASIC program into low-level machine code.
If your program is in error then a compilation will not complete and an error box will appear. Double click on the error to get to the line which has the problem.

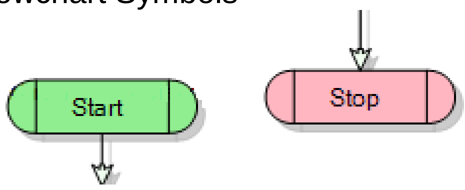
6.13 The programmer

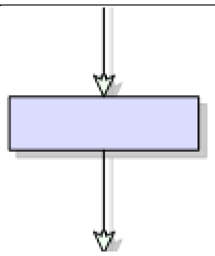
When you have successfully compiled a program pressing **F4** or the green IC picture in the toolbar starts the programmer. If no microcontroller is connected an error will pop up. If the IC s connected then the BASCOM completes the programming process and automatically resets your microcontroller to start execution of your program.

6.14 An introduction to flowcharts

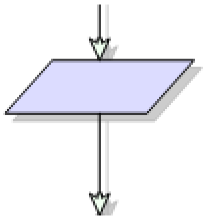
Flowcharts are an incredibly important planning tool in use not just by software designers but by many professionals who communicate sequences and actions for systems of all types.

Flowchart Symbols

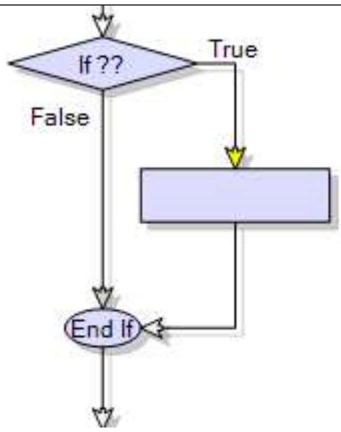




This is a process step, where procedures are carried out

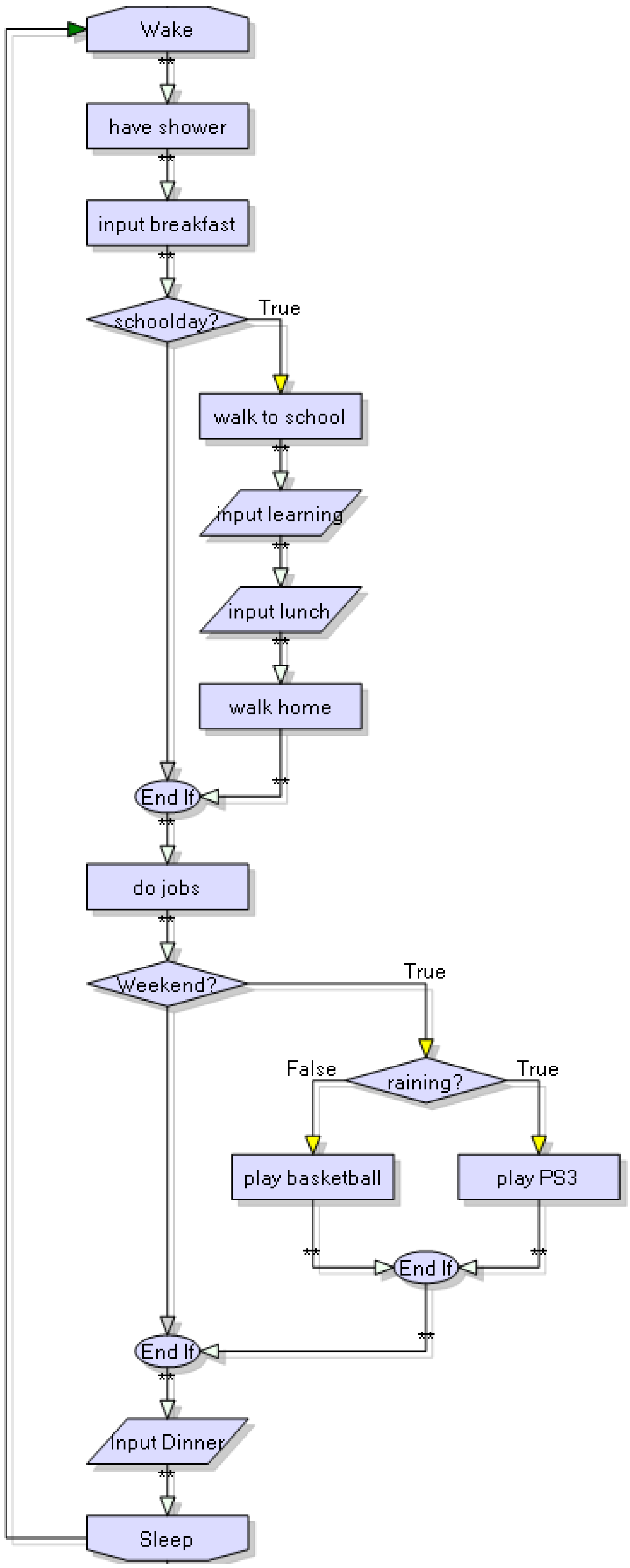


This is an Input or Output process step where

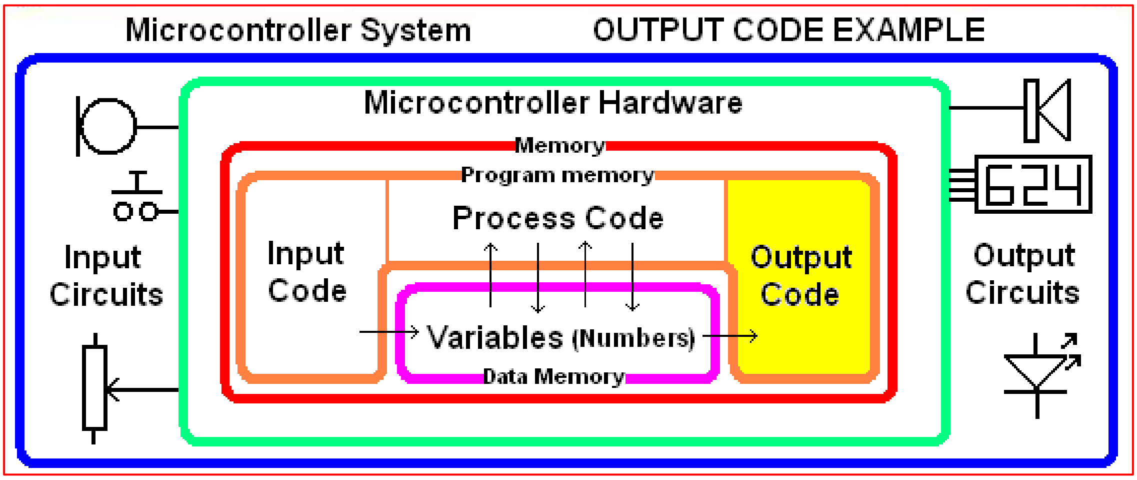


Here we test to see if something is true or false.

Daily Routine FlowChart

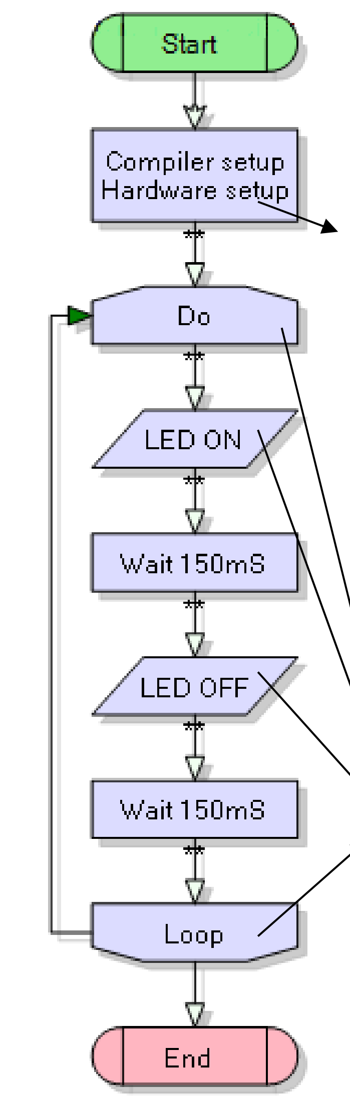


6.15 Bascom output commands



Flash1LEDv1.bas

Type the code below into BASCOM, save it, then F7 to compile and then F4 to program




```
‘ Flash1LEDv1.bas
'-----------------------------------------------------------------
' Compiler Setup (tell Bascom about our micro)
$regfile = "attiny461.dat"  'wgich micro
$crystal = 1000000         'its speed
'-----------------------------------------------------------------
' Hardware Setups
' (these tell bascom how to setup our micro)
Config Porta = Output          'LEDs on port
'-----------------------------------------------------------------
' Declare Constants
' (these tell bascom names we will use for numbers
' in our program, this makes it easy
'to change things quickly later)
Const Flashdelay = 250       ‘ preset how long a wait will be
'-----------------------------------------------------------------
Do                           ‘start of a loop
    PortA.7 = 1              ‘ LED 7 on
    Waitms Flashdelay        ‘wait a preset time
    PortA.7 = 0             ‘LED 7 off
    Waitms Flashdelay        ‘wait a preset time
Loop                         ‘return to do and start again
End
```

YOU NEED TO INDENT CODE BETWEEN ALL CONTROL STRUCTURES SUCH AS WITH THIS DO-LOOP, it really helps make your code more readable and easier to debug! Use the TAB key in Bascom to do it.

This is a typical first program to test your hardware
Every line of the code is important.

$regfile="attiny461.dat", Bascom needs to know which micro is being used as each micro has different features; this is the name of a file in the Bascom program folder with every detail about the ATTiny461.

$crystal=1000000, This line tells Bascom the speed at which our microcontroller is executing operations 1 million per second)so that Bascom can calculate delays such as waitms properly

Config porta=output, each I/O must be configured to be either an input or output; (it cannot be both at once)

Const Flashdelay=150, 'constants' are used in a program, it is easier to remember names and it is useful to keep them all together in one place in the program (this is a code of practice).

DO - LOOP statements enclose code which is to repeat forever; when programming it is important to indent (tab) code within loops; this makes your code easier to follow (this is a code of practice).

Waitms flashdelay wait a bit, a microcontroller carries out operations sequentially, so if there is no pause between turning an LED on and turning it off the led will not be seen flashing

Output Code

PortA.7 =1 make porta.7 <u>high</u> (which will turn <u>on</u> the LED connected to that port)
PortA.7 = 0 make porta.7 <u>low</u> (which will turn <u>off</u> the LED connected to that port)

6.16 Exercises

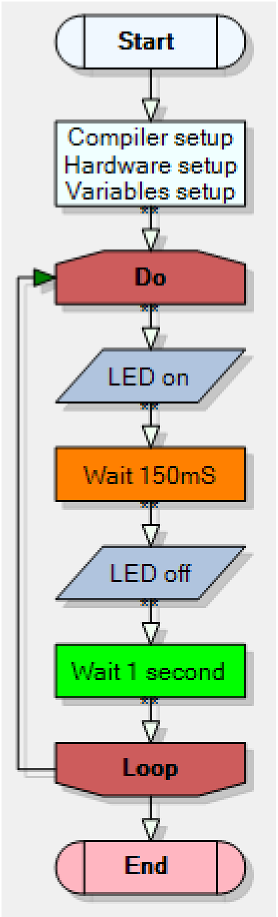
1. When a computer monitor is in standbay mode often an LED is going to alert the user that the power is left on but ther ei s no signal to the monitor. Sometimes thisis a permanently on LED sometimes it is a slow flashing one Find the value of Flashdelay so that the LED is on for 2 seconds and off for 2 seconds
2. Find the value of Flashdelay so that the LED is on for ½ a second and off for ½ a second
3. Find the value of Flashdelay so that the LED is on for 5 seconds and off for 5 seconds

6.17 Two delays

Often pieces of equipment have a flashing LED that is on very briefly then off for a long time. E.g. on for 0.15Seconds (150mSec) and off for a second (1000mSec)

Flash1LEDv2.bas

This program has TWO delays one for the on time and one for the off time

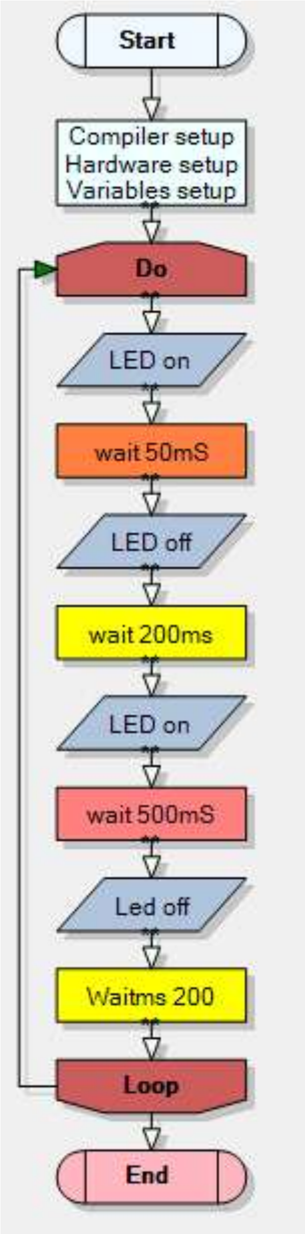



```
' Flash1LEDv1.bas
'-----------------------------------------------
' Compiler Setup (this tell Bascom things about our micro)
$regfile = "attiny461.dat"   'bascom must know the micro
$crystal = 1000000      'bascom must know its speed
'-----------------------------------------------
' Hardware Setups
' (these tell bascom how to setup our micro)
Config Porta = Output          'LEDs on port
'-----------------------------------------------
' Declare Constants
' (these tell bascom names we will use for numbers
' in our program, this makes it easy
'to change things quickly later)
Const Ondelay = 150      'how long an LED will be on for
Const Offdelay = 1000     'how long an LED will be off for
'-----------------------------------------------
Do                        'start of a loop
    PortA.7 = 1           ' LED 7 on
    Waitms Ondelay        'wait a preset time
    PortA.7 = 0           'LED 7 off
    Waitms Offdelay       'wait a preset time
Loop                      'return to do and start again
End
```

REMEMBER YOU NEED TO INDENT CODE BETWEEN ALL CONTROL STRUCTURES SUCH AS WITH THIS DO-LOOP, Use the TAB key

4. Change the on time to the smallest possible length you can see

5. A piece of equipment that has a flashing LED like this is sometimes referred to as having a 'heatbeat' indicator to show it is 'alive' or on. Change the on and off time to match your heart beat.



All sorts of 'heartbeat' indicators can be used in equipment to show it is on. Double flashes are common ands some equipment might have a short then a long flash like this program.
It needs three delays:

```
Const Ondelay1 = 50
Const Ondelay2 = 500
Const OFFdelay = 200
```

Write this program then modify it to make what you think is a good heartbeat.

6.18 Syntax errors -'bugs'

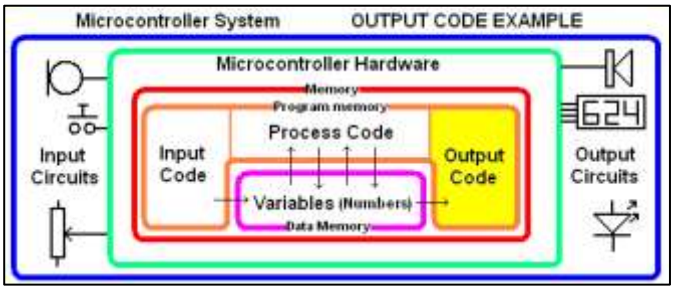
Playing around will develop your understanding, carry out AT LEAST these to see what happens

- What happens if you change Const Flashdelay to Const faslhdelay? (deliberate spelling error)
- What happens if $crystal = 10000000 or 100000 instead of 1000000?
- What happens if your change the $regfile to "attin26.dat"? (deliberate spelling eror)
- What happens if one of the waitms flashdelay statements is deleted (look closely at the LED)?
- What happens when the two waitms flashdelay statements are deleted (look closely at the LED)?

In programming we call these **syntax** errors. It's like having a conversation with a person whose first language is different to your own and they get the order of words in a sentence jumbled or use the worng word. We can generally get the meaning of the sentence but computers cannot understand the small mistakes that a programmer makes. The syntax has to be 100%.
E.g. Cup tea make me you or time when lunch is or stop bus where is we can make meaning fo these but a computer cannot make sense between flasshdelay and flashdelay.

6.19 Microcontroller ports: write a Knightrider program using LED's

Learn about controlling ports.
Ports are groups of 8 I/O pins.

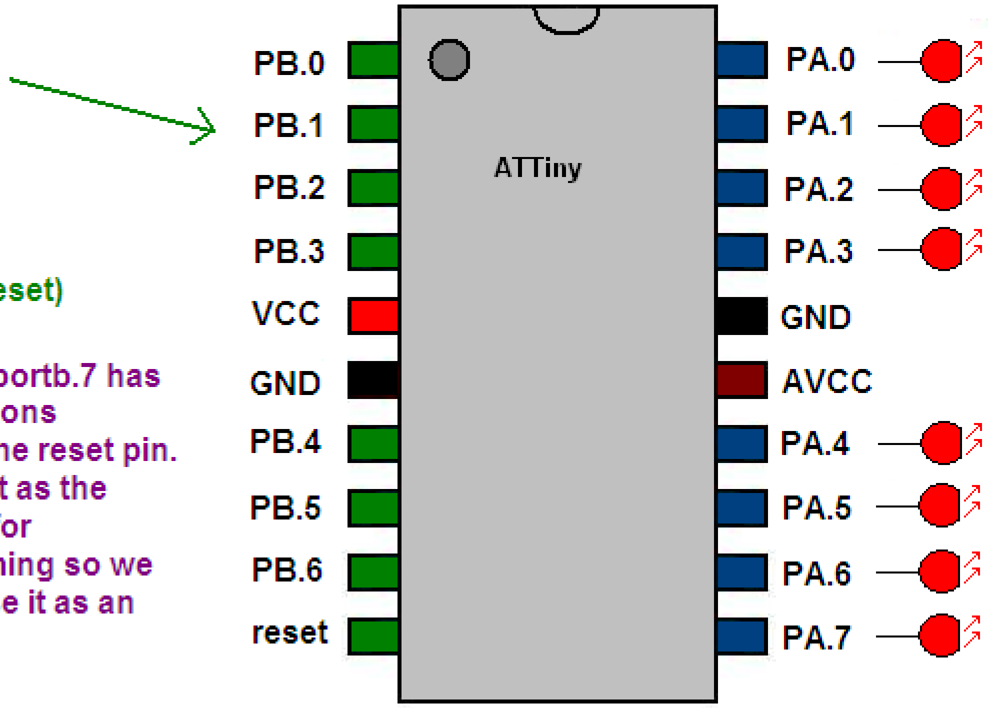


Microcontrollers have their pins arranged in groups of 8 pins called PORTS

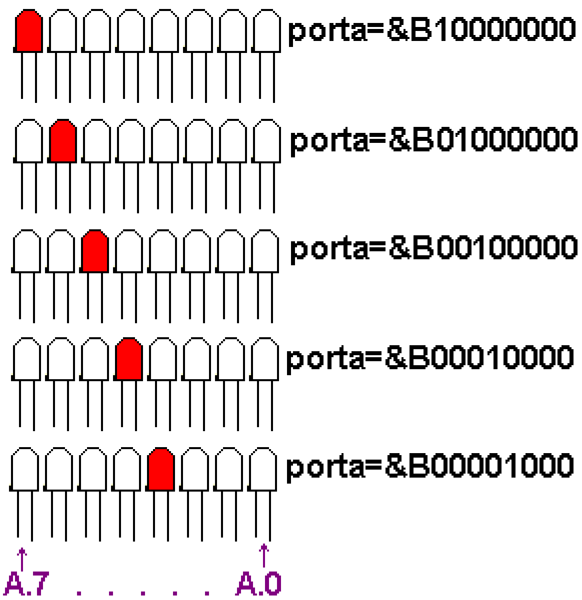
PortB has 8 pins
these are
portb.0
portb.1
portb.2
portb.3
portb.4
portb.5
portb.6
portb.7(reset)

however portb.7 has two functions
it is also the reset pin.
We need it as the reset pin for programming so we cannot use it as an I/O pin

PortA has all 8 pins available for us to use



If we have 8 LEDs connected to portA we could control them individually
HOWEVER…there is a better way..

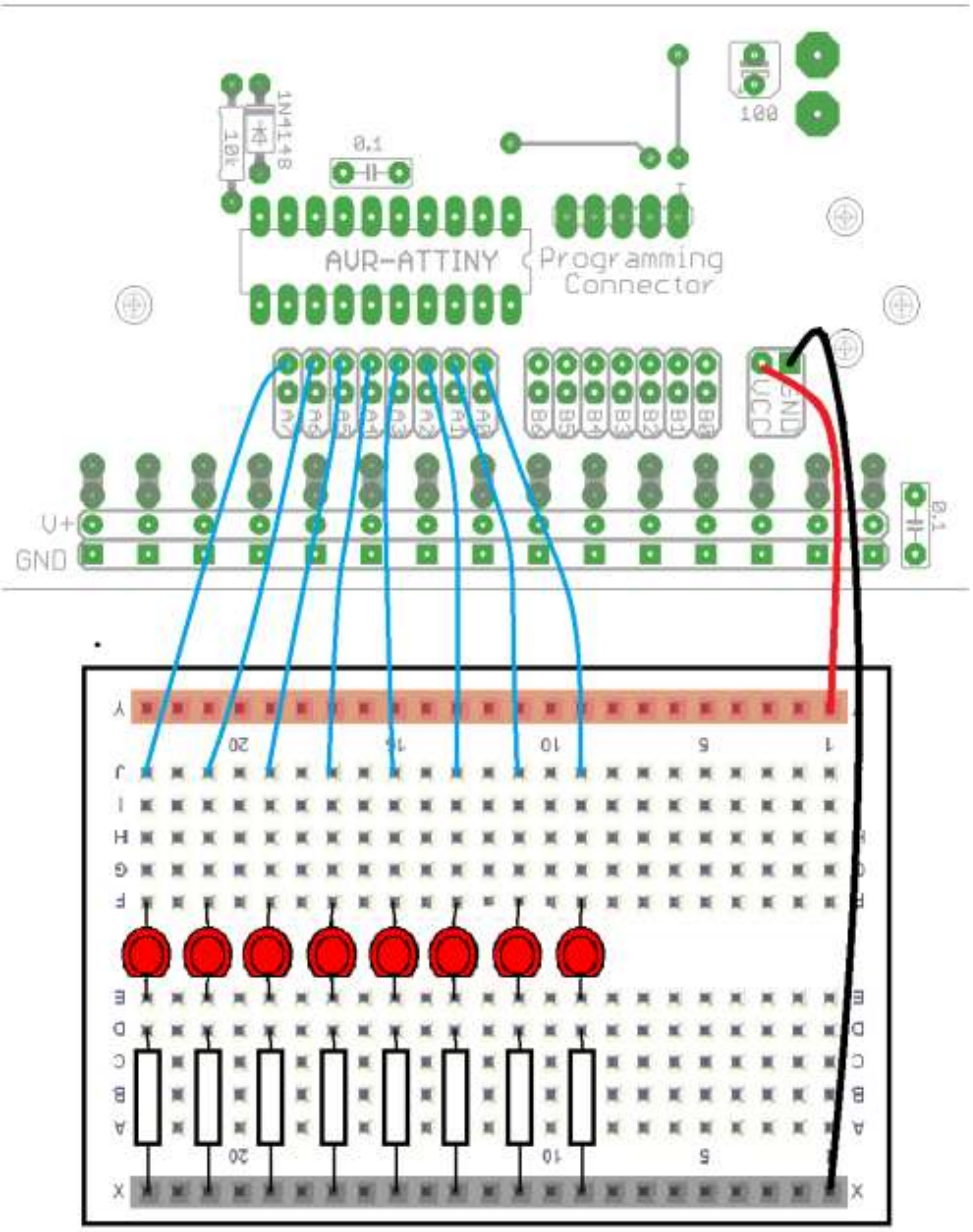


we should use the commands to control the whole port at once

You already have 1 LED connected to portA.7 now connect another 7 LEDs to your microcontroller from ports A.6 through to A.0 **(each needs an individual 1k current limit resistor, see the picture below)** . Write a program to flash all 8 LEDs in a repeating sequence e.g. 'led1, 2, 3, 4, 5, 6, 7, 8. 7, 6, 5, 4, 3, 2, 1, 2, 3... Use the following code to get started

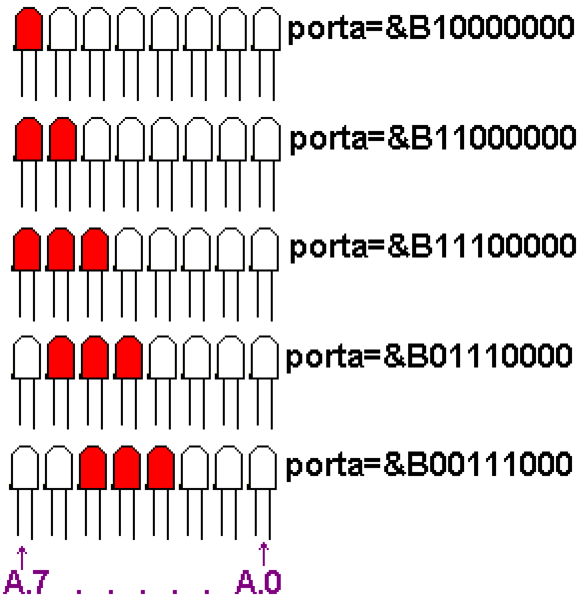
```
Porta=&B10000000
Waitms flashdelay
Porta=&B01000000
Waitms flashdelay
Porta=&B00100000
Waitms flashdelay
…
```

Using the above command to control the whole port at once is quicker and easier for some applications than individually controlling each pin. You need to choose the best way when thinking about readability and understandability.



6.20 Knightrider v2

As a second exercise rewrite the program so that three LEDs turn on at as in the Knightrider car. Sequence = LED0, LED01, LED012, LED123, LED234, LED345, LED456, LED567, LED67, LED7, LED67, LED567…



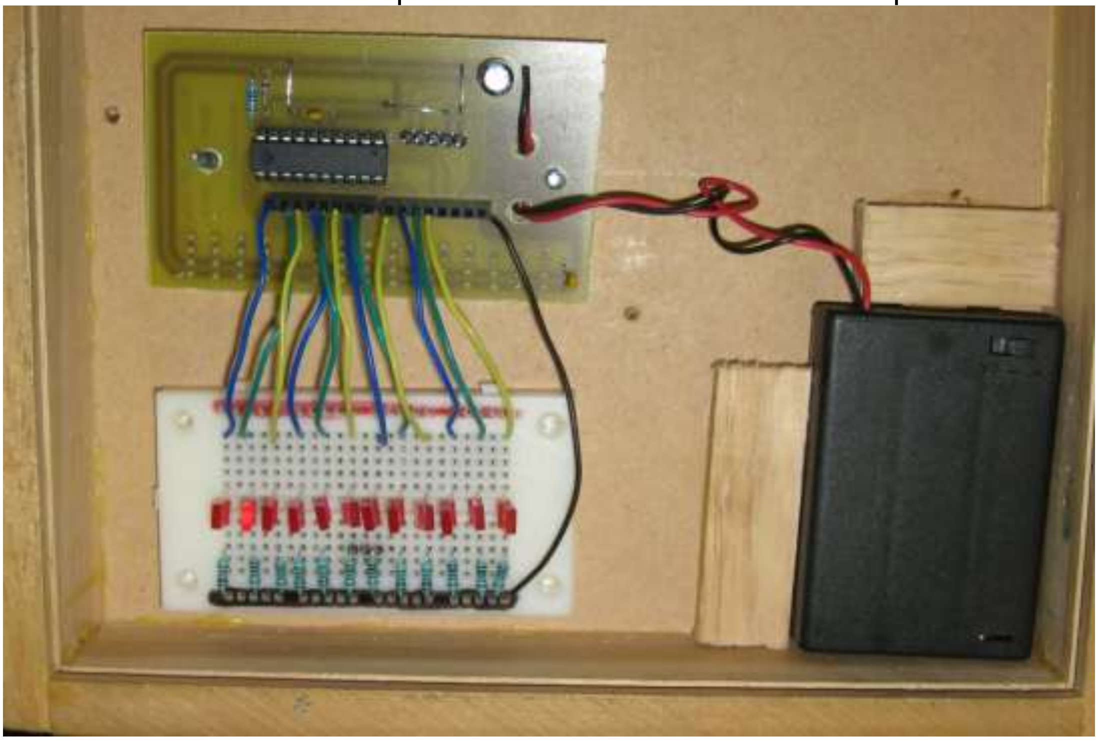
<u>Success criteria to work on in your program</u>

2. <u>Use spaces to help layout your program so it looks good</u>
3. <u>Comment your program with short clear descriptions</u>
4. <u>Use constants with good names e.g. waitms flashdelay not waitms 150</u>
5. <u>Keep a record of BOTH the schematic and layout changes in your notebook</u>

Remember that using a constant is meeting good programming **codes of practice**; it means that when you want to change the speed all you have to do is change it in one place in the program. If you didn't use Const then you would have to go through your whole program and change every waitms line individually.

6.21 Knightroder v3

Now we want to extend the pattern to cover 15 LEDs and both ports



```
'KnightRiderV3.bas
'Leds arranged
' 1    2    3    4    5    6    7    8    9   10   11   12'A.7  A.6  A.5
A.4  A.3  A.2  A.1  A.0  B.6  B.5  B.4  B.3(achieved level comments)

'this program shows how to write code which controls the whole port at
'once using the commands portA=&B00000001, rather than individual set
and 'reset commands which are very wasteful of code space when multiple
LEDs 'have to 'be controlled (excellence comment)
'-------------------------------------------------------------------
' Compiler Setup (these tell Bascom things about our micro)
$regfile = "attiny461.dat"                'bascom needs to know the micro
$crystal = 1000000                        'bascom needs to know its speed

'-------------------------------------------------------------------
' Hardware Setups (these tell bascom how to setup our micro)
' setup direction of all ports
Config Porta = Output                     'LEDs on portA
Config Portb = Input                      'switches on portB
' Hardware Aliases (these tell bascom names we will use for I/O devices
' attached to the Micro, names are easier to remember that ports)
Config Porta = Output
Config Portb = Output
'-------------------------------------------------------------------
' Declare Constants (these tell bascom names we will use for numbers in
' ' our program, this makes it easy to change things quickly later)
'   times have been made shorter for testing purposes
```

```
Const Delaytime = 25
Do
    Porta = &B10000000                       '1 =A.7
    Waitms Delaytime
    Porta = &B01000000                       '2 =A.6
    Waitms Delaytime
    Porta = &B00100000                       '3 =A.5
    Waitms Delaytime
    Porta = &B00010000                       '4 =A.4
    Waitms Delaytime
    Porta = &B00001000                       '5 =A.3
    Waitms Delaytime
    Porta = &B00000100                       '6 =A.2
    Waitms Delaytime
    Porta = &B00000010                       '7 =A.1
    Waitms Delaytime
    Porta = &B00000001                       '8 =A.0
    Waitms Delaytime
    'the hand over between ports requires 2 lines one to turn off the
    ' the LED one port and the other to turn on the LED on the other port
    ' (example of an merit level comment - it explains what you did)
    Porta = &B00000000                       '8 off
    Portb = &B01000000                       '9 =B.6
    Waitms Delaytime
    Portb = &B00100000                       '10 =B.5
    Waitms Delaytime
    Portb = &B00010000                       '11 =B.4
    Waitms Delaytime
    Portb = &B00001000                       '12 =B.3
    Waitms Delaytime
    Portb = &B00010000                       '11 =B.4
    Waitms Delaytime
    Portb = &B00100000                       '10 =B.5
    Waitms Delaytime
    Portb = &B01000000                       '9  =B.6
    Waitms Delaytime
    'the hand over between ports requires 2 lines one to turn off the
    ' the LED one port and the other to turn on the LED on the other port
    Portb = &B00000000                       '9 off
    Porta = &B00000001                       '8 =A.0
    Waitms Delaytime
    Porta = &B00000010                       '7 =A.1
    Waitms Delaytime
    Porta = &B00000100                       '6 =A.2
    Waitms Delaytime
    Porta = &B00001000                       '5 =A.3
    Waitms Delaytime
    Porta = &B00010000                       '4 =A.4
    Waitms Delaytime
    Porta = &B00100000                       '3 =A.5
    Waitms Delaytime
    Porta = &B01000000                       '2 =A.6
    Waitms Delaytime
Loop
End
```

6.22 Commenting your programs

Comments in your program code are used to explain (not just describe) to others what your program is doing or how your program is doing it.

Take note of the commenting in the code above.– it is showing the reader which LED is coming on and explains the special case of hand over of the LED control from one port to the other.

In your studies we often distinquish between describe=Achieved, explain=Merit and justify=Excellence. Discuss would be where you explain and justify why you did it one way rather than another. The code above is an excellence for commenting because it justifies why it works the way it does!

If you can write good comments that explain thoroughly and where necessary discuss your code you are an excellent programmer!

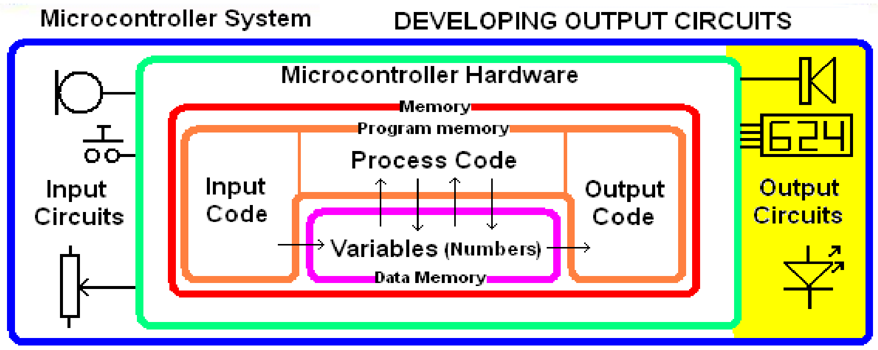
6.23 Learning review

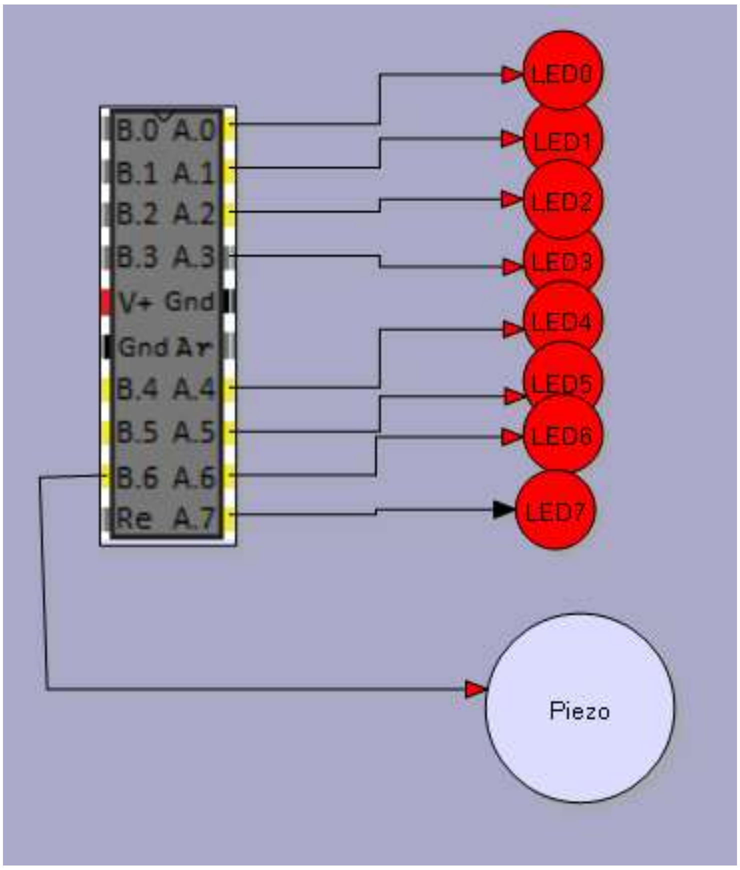
1. Microcontrollers input and output pins are grouped into 8 and called ports.
   - e.g. **PORTA**, or **PORTB**
2. Before we use a pin or port we must set it up as either an input or an output
   - **Config porta=output** OR
   - we can configure each pin separately **config pina.3=output**
3. The 8 pins in a port are numbered from 7 down to 0
   - porta.7, porta.6, … porta.2, porta.1, porta.0
4. We can make each pin individually high or low
   - e.g. **porta.7 = 1** or **porta.7 = 0**
5. We can control all 8 pins at once
   - **Porta= &B10100011**
     This is the same as
   - **porta.7 = 1**
   - **porta.6 = 0**
   - **porta.5 =1**
   - **porta.4 = 0**
   - **porta.3 = 0**
   - **port a.2 = 0**
   - **port a.1 = 1**
   - **porta.0 = 1**
6. We can delay a microcontroller using program code
   - **Waitms 50**
     Or better still use a constant
   - **Const timedelay=50**
   - **Waitms timedelay**
7. Comments make your program more readable
   - and especially explain how/why you did something
8. Programs are sequential and run forever within a
   - **Do-Loop**

6.24 What is a piezo and how does it make sound?

A piezo is made from a nonsymmetrical crystal; these are generally ceramic nowadays although the principle was originally discovered in naturally occuring quartz (and other) crystals. When a crystal has an electrical charge applied to it, it moves in one direction. We make use of this property to produce sound and also in ultrasonic cleaning and other things. The opposite occurs too, if a crystal is moved or stressed a voltage potential can be created. This property is put to work in piezo lighters (such as in a bbq) and in ceramic microphones. Modern ceramic type piezos are much more efficient than natural quartz ones.

The piezo can be attached directly between a microcontrollers output pin and ground.





Bascom's sound command can be used to directly make a tone.

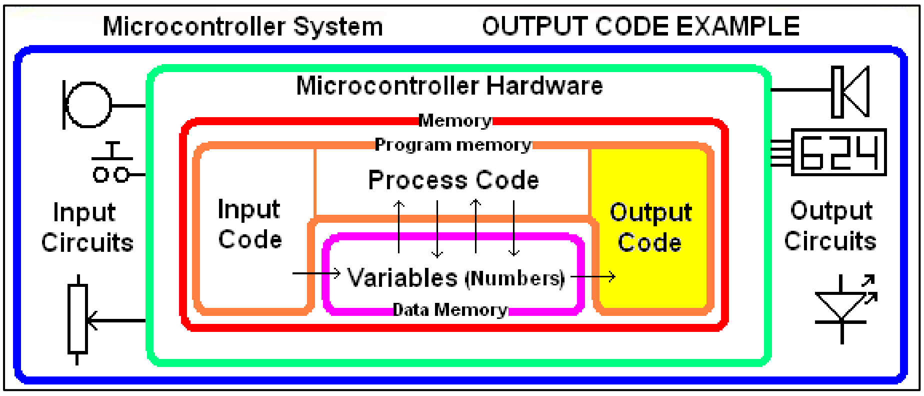
Piezo Alias Portb.6
Sound piezo, 500, 300 'that's all that's required

The Bascom sound command has three parameters (values) attached ot it.

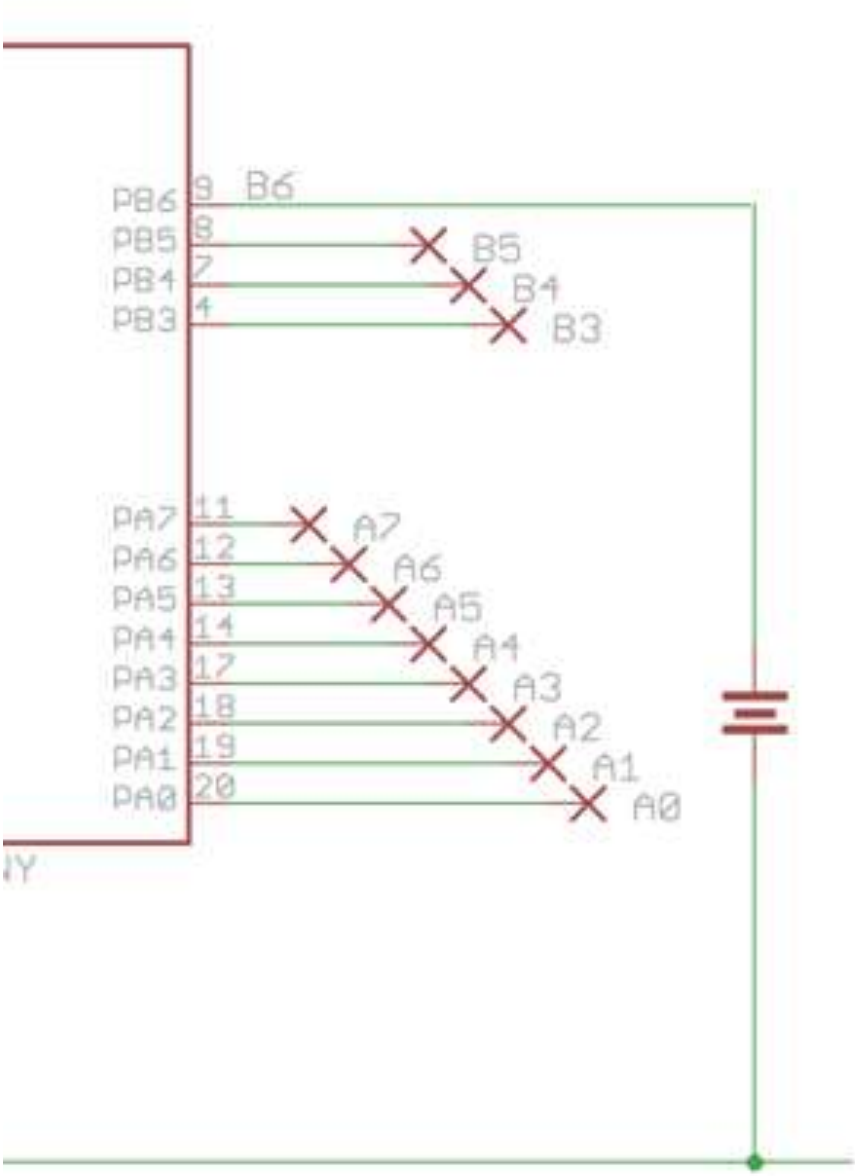
- The port or pin of the microcontroller used
- The duration of the sound (number of pulses)
- The time the pin is high and low for.

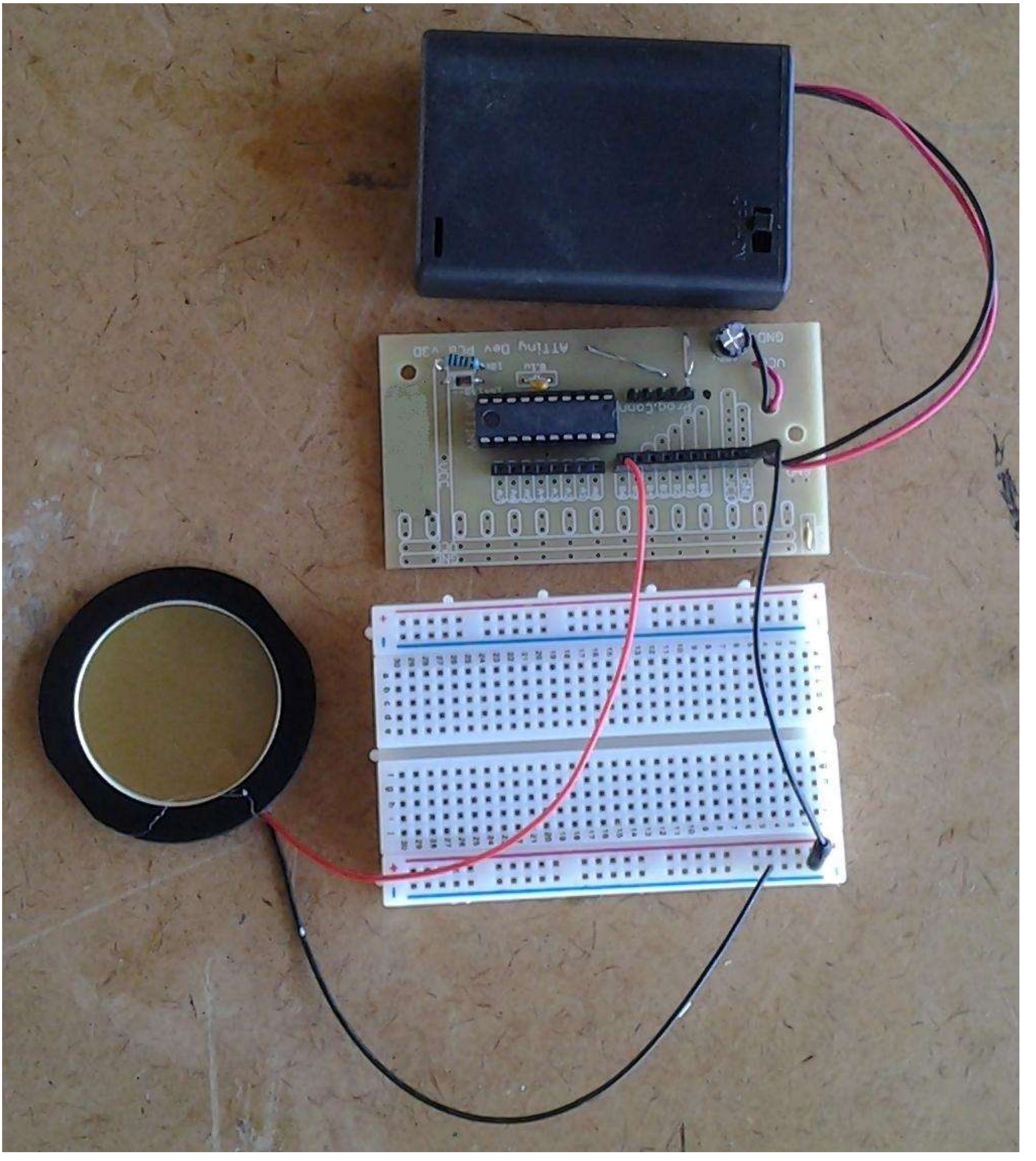
This command is not easy to use to get accurate tones from your AVR, but they do make useful sounds. Experiment with the sound command and make a series of tones suitable for an alarm

6.25 Sounding Off



.
Add a piezo to your project, the piezo is connected to PortB.6, then see the layout pic on the next page





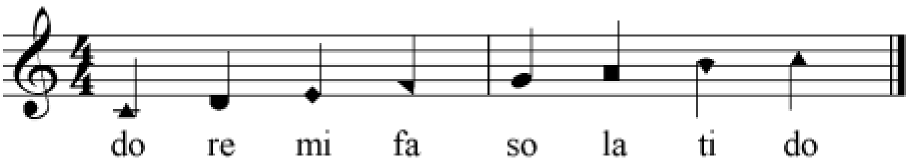
This picture only shows the piezo, DO NOT REMOVE ALL YOUR LEDS

```
'-------------------------------------------------------------------
' Compiler Setup (these tell Bascom things about our micro)
$regfile = "attiny461.dat"                'bascom needs to know the micro
$crystal = 1000000                        'bascom needs to know its speed
'-------------------------------------------------------------------
' Hardware Setups (these tell bascom how to setup our micro)
' setup direction of all ports
Config Porta = Output                     'LEDs on portA
Config Portb = Output                          'piezo on portB
' Hardware Aliases (these tell bascom names we will use for I/O devices
' attached to the Micro, names are easier to remember that ports)
Piezo Alias PortB.6
'-------------------------------------------------------------------
' Declare Constants (these tell bascom names we will use for numbers in
' ' our program, this makes it easy to change things quickly later)
'   times have been made shorter for testing purposes
Const Flashdelay = 150
'-------------------------------------------------------------------
'Program starts here
Do
   PortA = &B00000001
   Waitms Flashdelay
   PortA = &B00000010
   Waitms Flashdelay
   PortA = &B00000100
   Waitms Flashdelay
   PortA = &B00001000
   Waitms Flashdelay
   PortA = &B00010000
   Waitms Flashdelay
   PortA = &B00100000
   Waitms Flashdelay
   PortA = &B01000000
' insert the line below into your program where you want a beep to
happen
   Sound Piezo , 50 , 150
Loop
```

6.26 Sound exercises

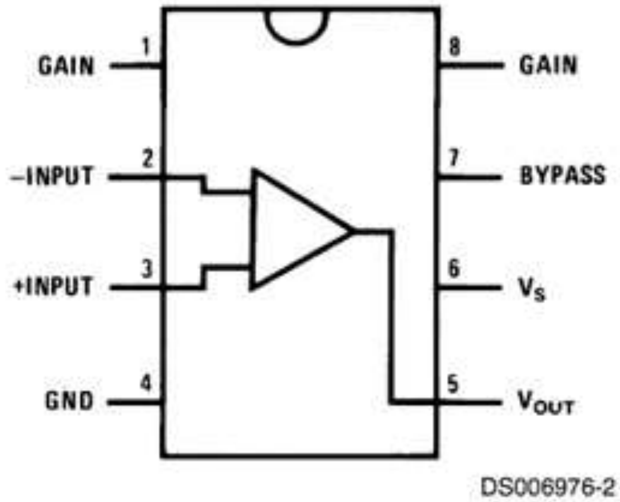
1. Make the knightrider program beep a each change of the LED

```
PortA = &B00000001
Waitms Flashdelay
Sound Piezo , 50 , 150
PortA + &B00000010
Waitms Flashdelay
Sound Piezo , 50 , 150
```

2. Develop a short sequence of tones that increase in pitch

3. Try and create a simple tune
   Don’t spend too much time on this (there is still more to learn)

6.27 Amp it up

If the piezo is not loud enough then you might like to add an amplifier to the output of your project.



The LM386 is an audio amplifier IC that is capable of upto 1.25Watts output.
The datasheet gives the following information

General Description

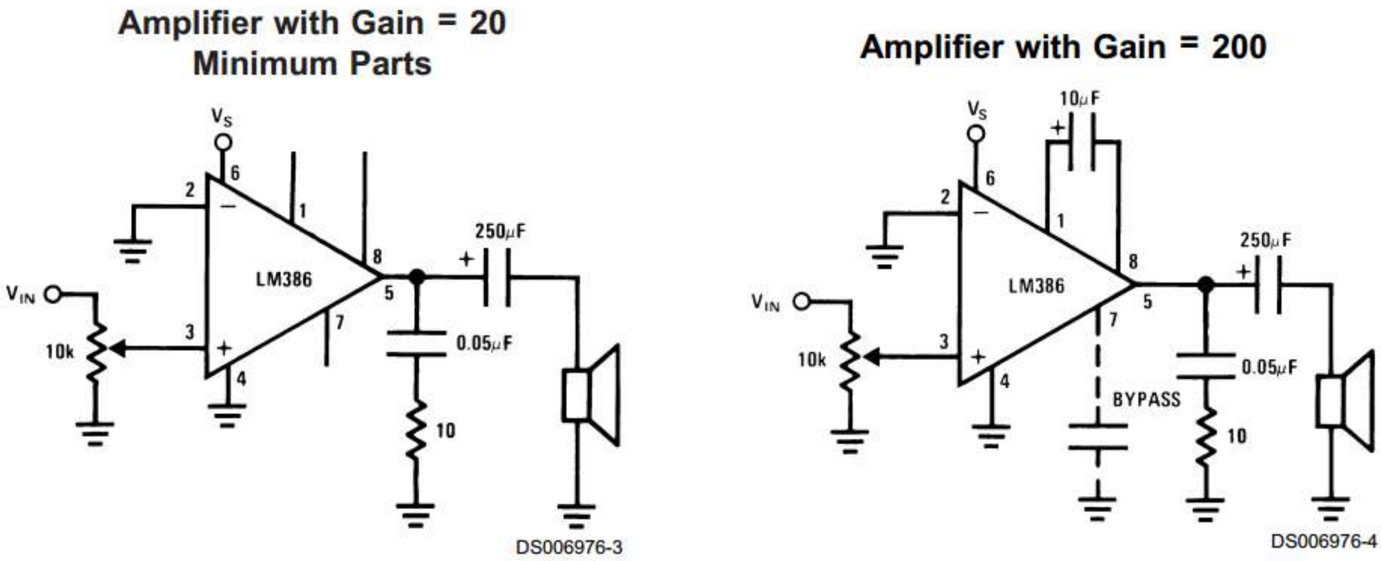
The LM386 is a power amplifier designed for use in low voltage consumer applications. The gain is internally set to 20 to keep external part count low, but the addition of an external resistor and capacitor between pins 1 and 8 will increase the gain to any value up to 200.

The inputs are ground referenced while the output is automatically biased to one half the supply voltage. The quiescent power drain is only 24 milliwatts when operating from a 6 volt supply, making the LM386 ideal for battery operation.

Features

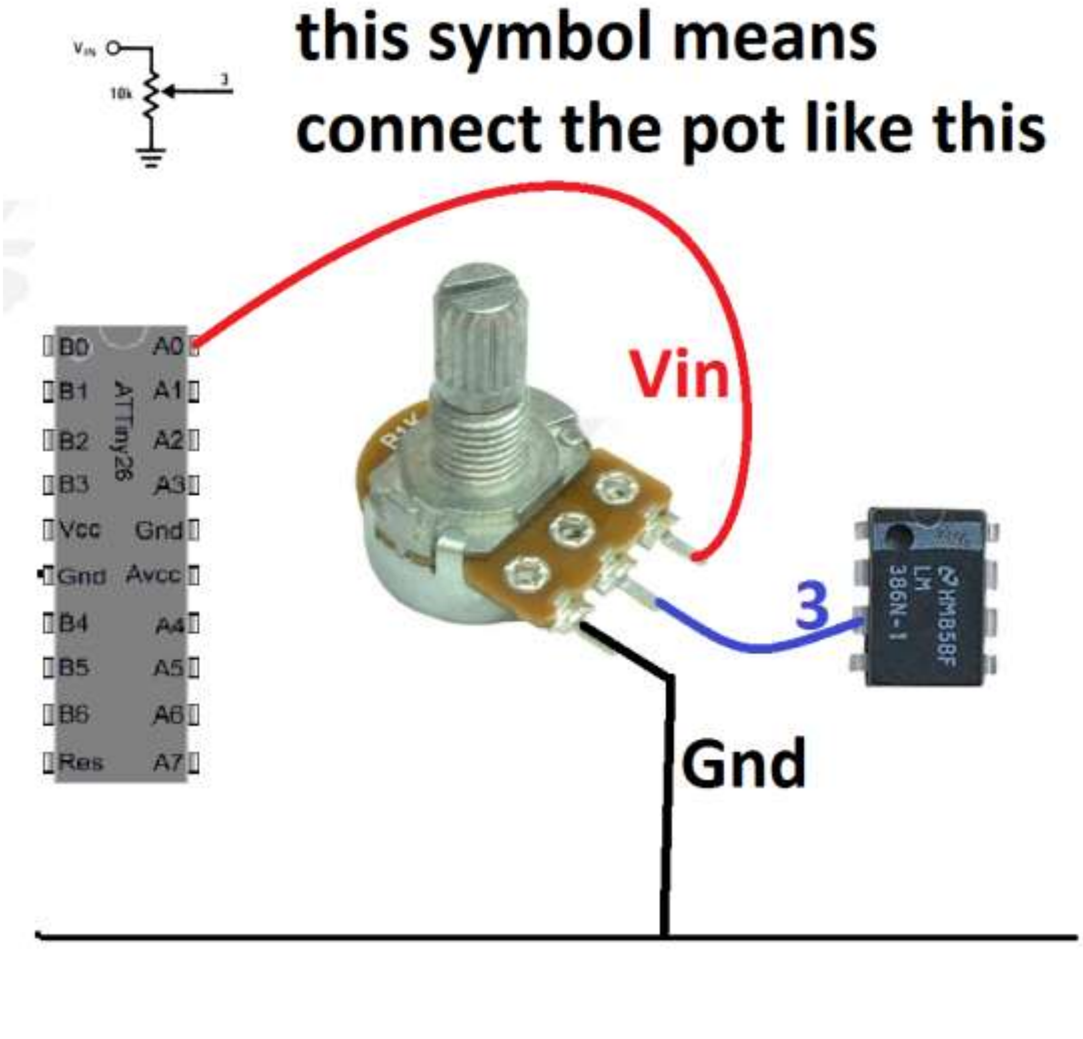
- Battery operation
- Minimum external parts
- Wide supply voltage range: 4V–12V
- Low quiescent current drain: 4 mA
- Voltage gains from 20 to 200

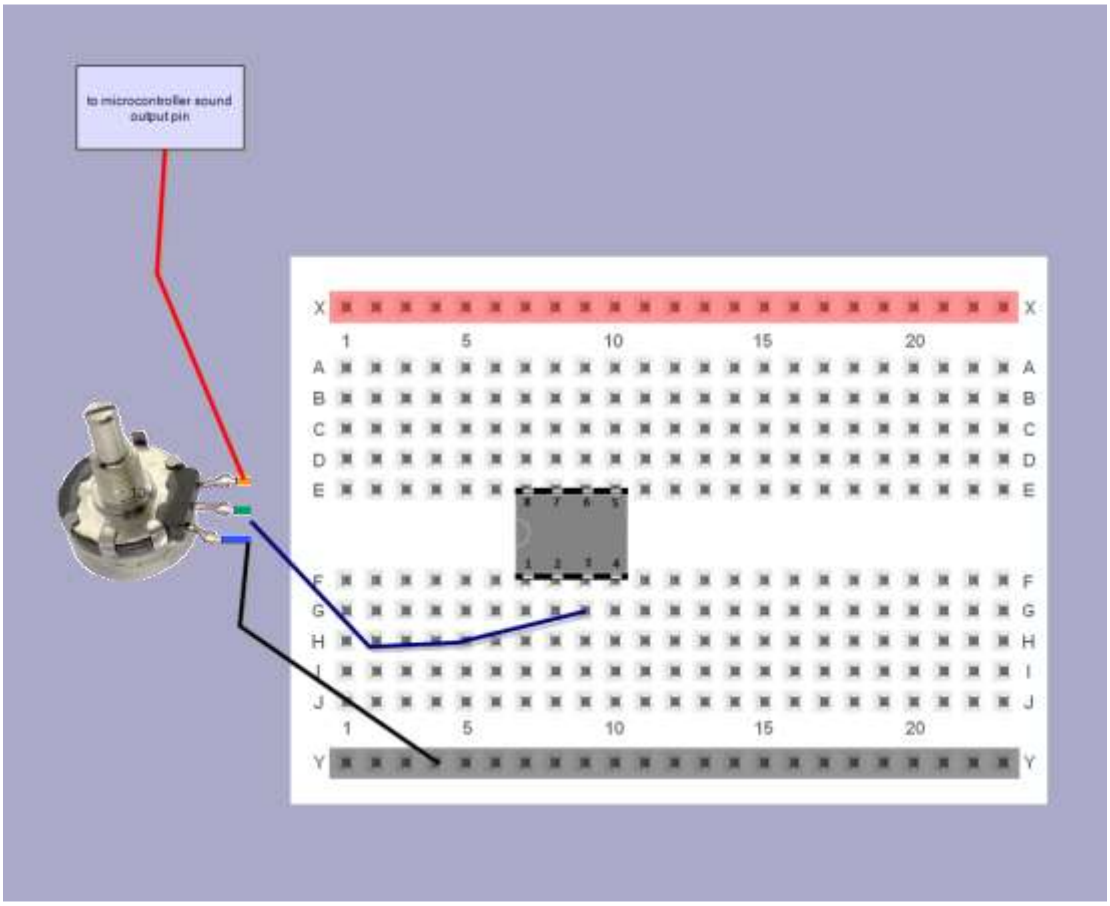
Typical Applications

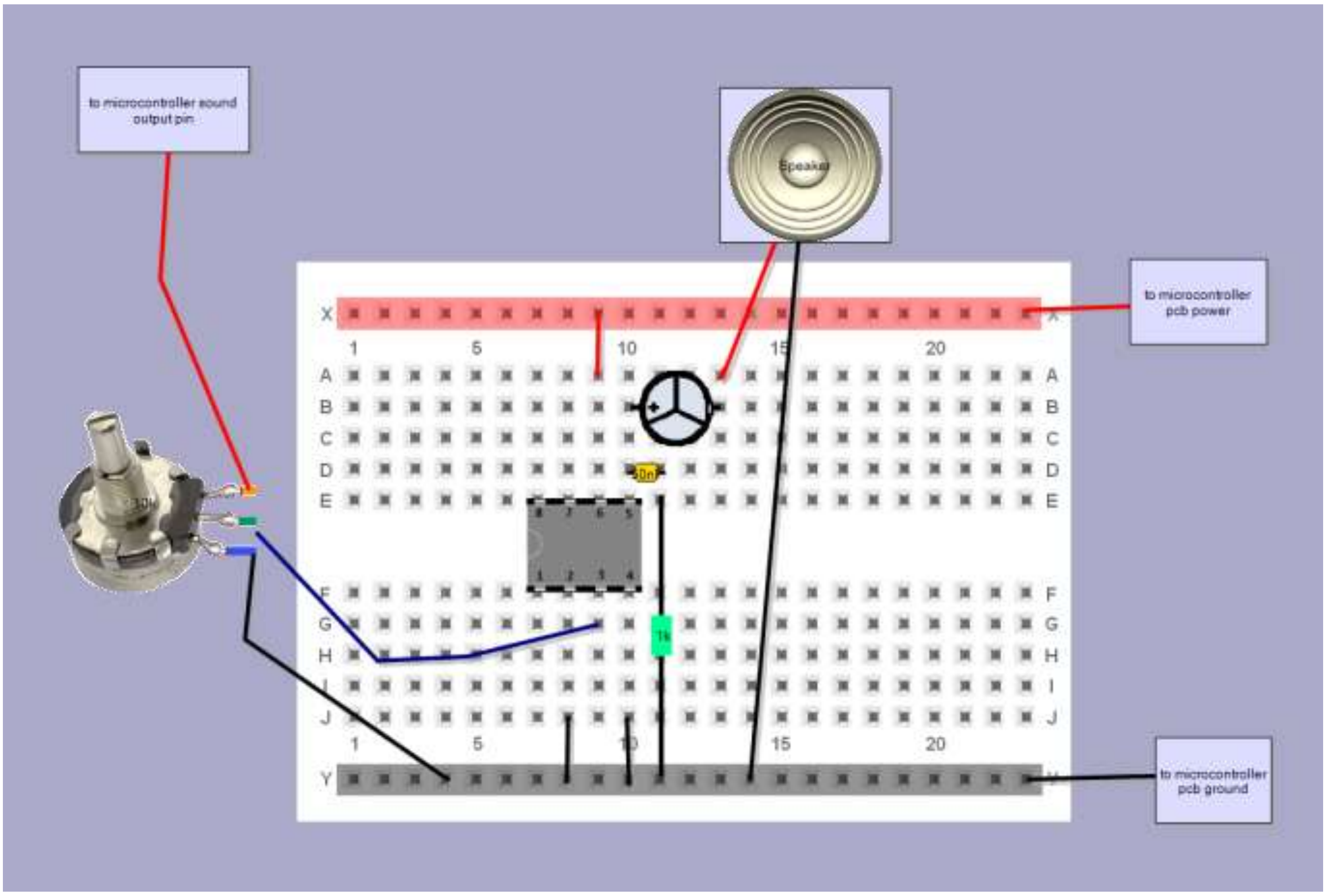


Can you see the difference between the two circuits, what has been added and where and which way around to increase the amplification from 20 to 200. We can build one of these circuits easily and quickly on breadboard to test it.

You will need a potentiometer, the diagram is not clear to beginners exactly what to do with the connection so this is how you connect it.





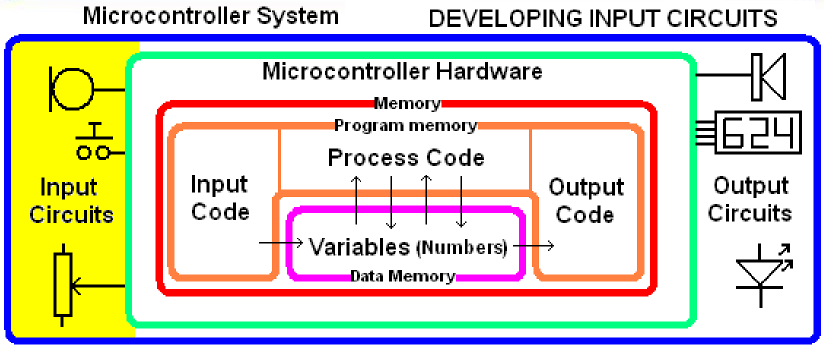


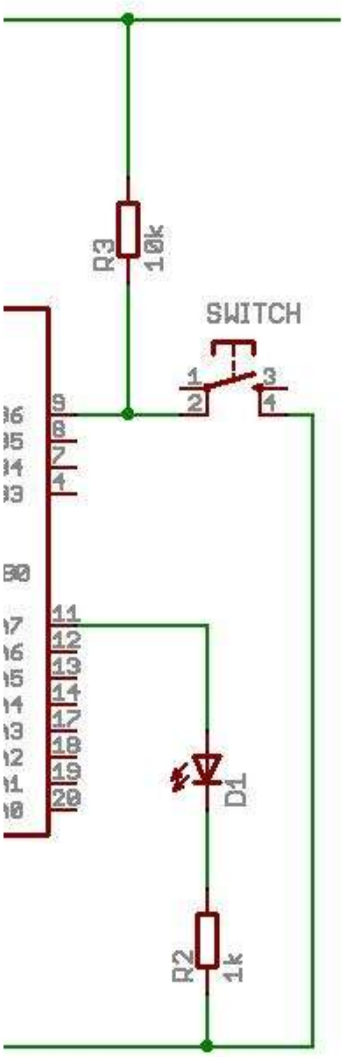
To boost the power of this circuit the schematic from the datasheet on the previous page shows an extra capacitor in the circuit. Can you add that to your circuit?

7 Microcontroller input circuits

A computer is not much use to us if it only has outputs we must have some inputs for the user or the world to tell the computer what to do.

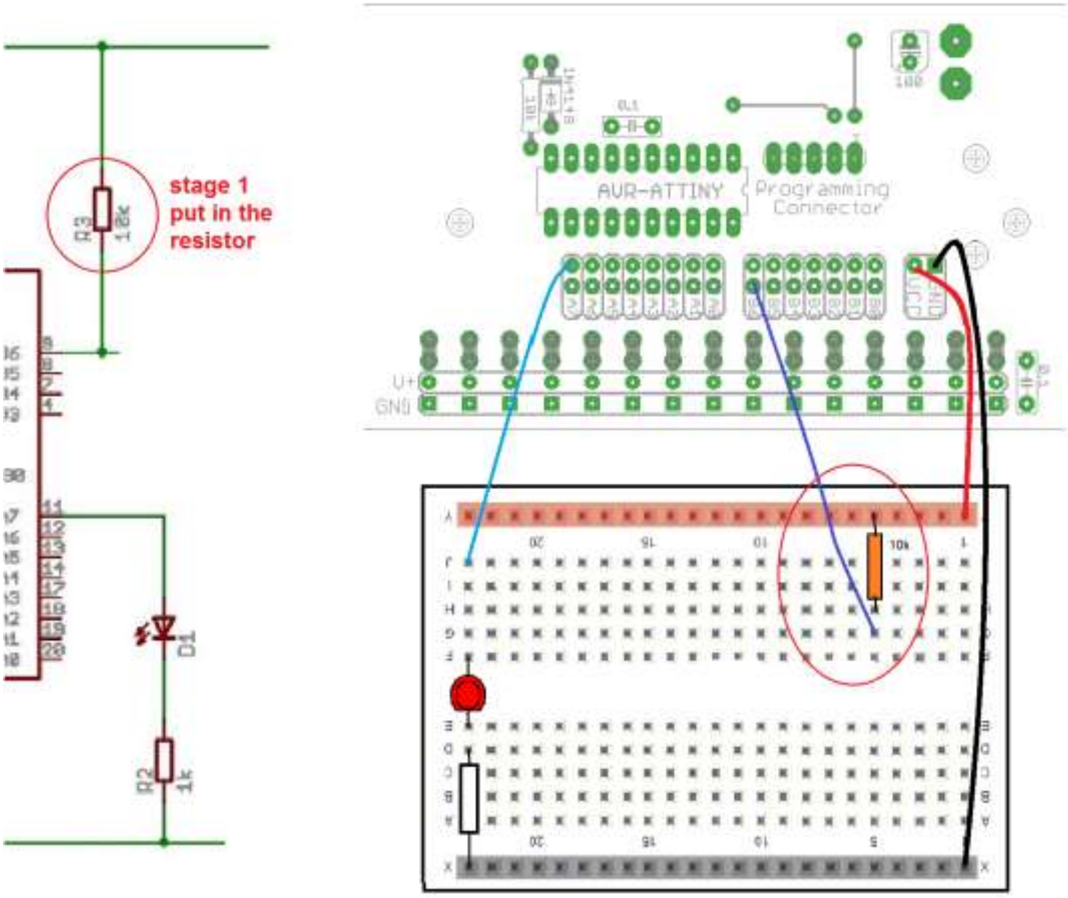
7.1 Single push button switch



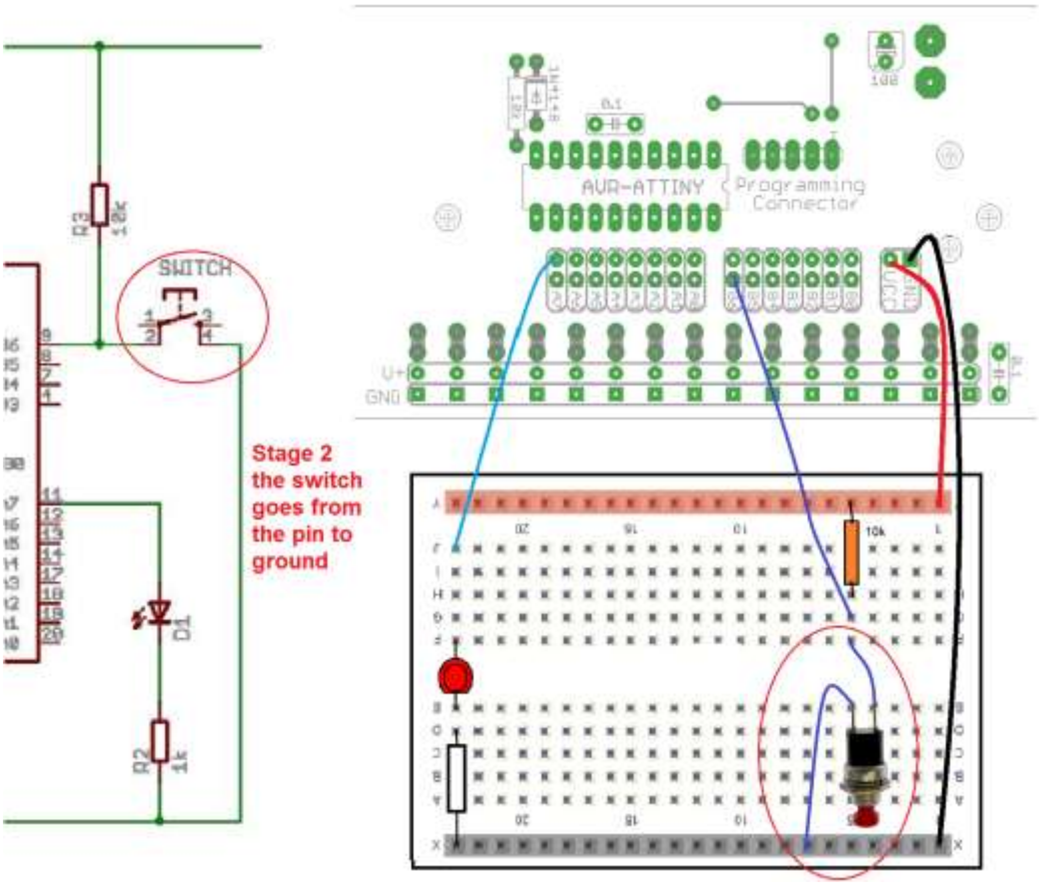


A ‘pullup’ resistor is essential in this circuit, as when the switch is not pressed it connects the input pin to a known voltage, if the resistor was not there then the input pin would be ‘floating’ and give unreliable readings. .

A lot of students get the switch wiring incorrect, here it has been broken down into two stages, first put in the 10k resistor from the pin to 5V.

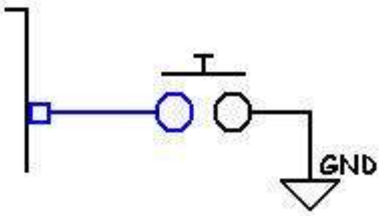


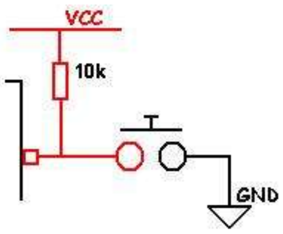
Next put in the Switch

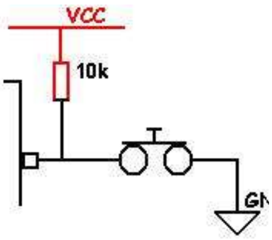


Get a mulitimeter and check the voltage goes up and down when the switch is pressed and released

7.2 Pullup resistor theory





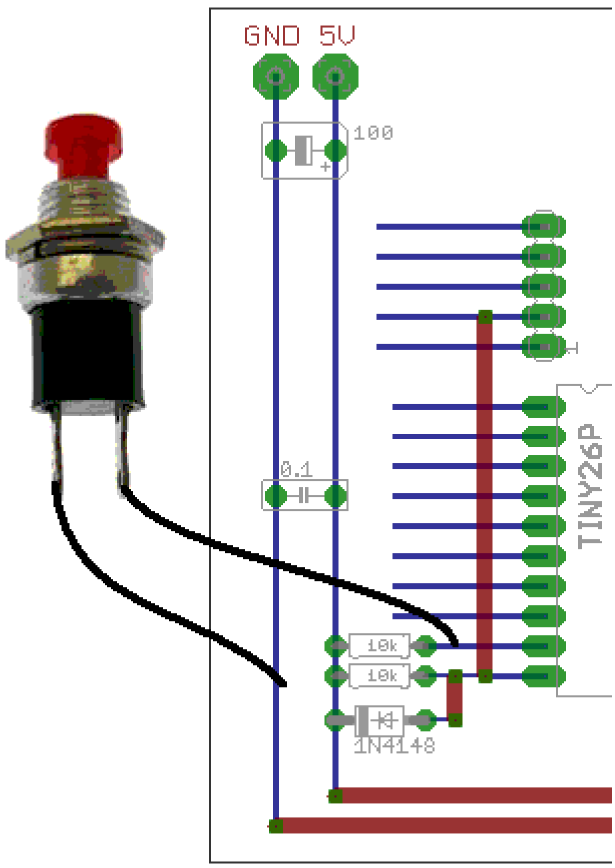


In this circuit the switch is connected without a pull-up resistor. The input pin of the microcontroller has no voltage source applied to it and is said to be 'floating'; the microcontroller input voltage will drift, sometimes be high (5V), sometimes low (0V) and is sensitive to touch and static leading to very unreliable results.

In this circuit the 10k resistor pulls the microcontroller input pin high (to 5V) making the input reliable when the switch is not pressed.

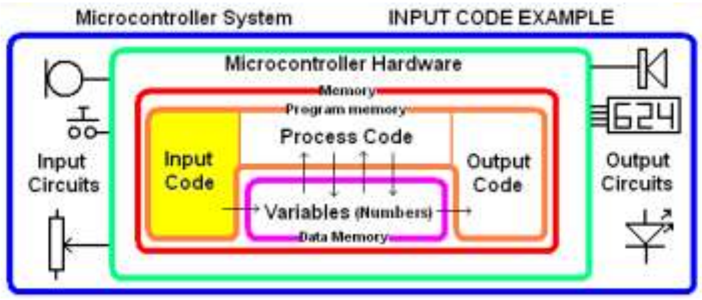
When the switch is pressed the voltage goes low (0V).

7.3 Switch in a breadboard circuit



In this circuit make sure the schematic is followed very closely.
The switch goes from the port to ground, the resistor from the port to 5V

7.4 Checking switches in your program



There are two main methods of checking for switch activity, we can wait until a switch is pressed before we continue or we can test the switch and if not pressed move on to do the rest of our program

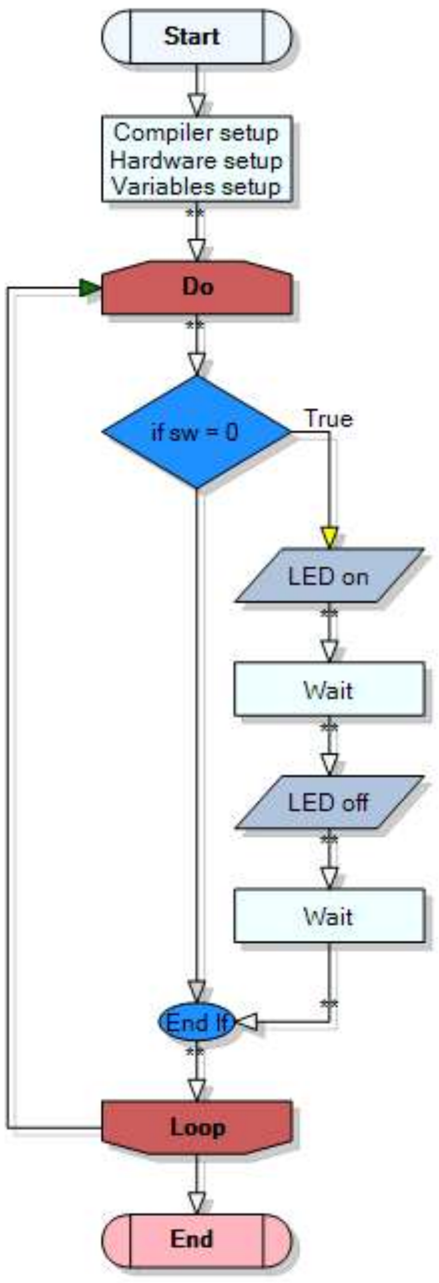
```
' check if switch pressed – main method
If Redsw = 0  then          'do this only if pressed
      do_something
end if
…
…
```

```
' check if switch pressed – method 2
Do
Loop Until Redsw = 0      ' wait here until pressed
…
…
```

7.5 Program Logic – the 'If-Then' Switch Test

In this first program we would like the LED to change from off to on every time the switch is pressed.

"When the switch is pressed toggle the LED"



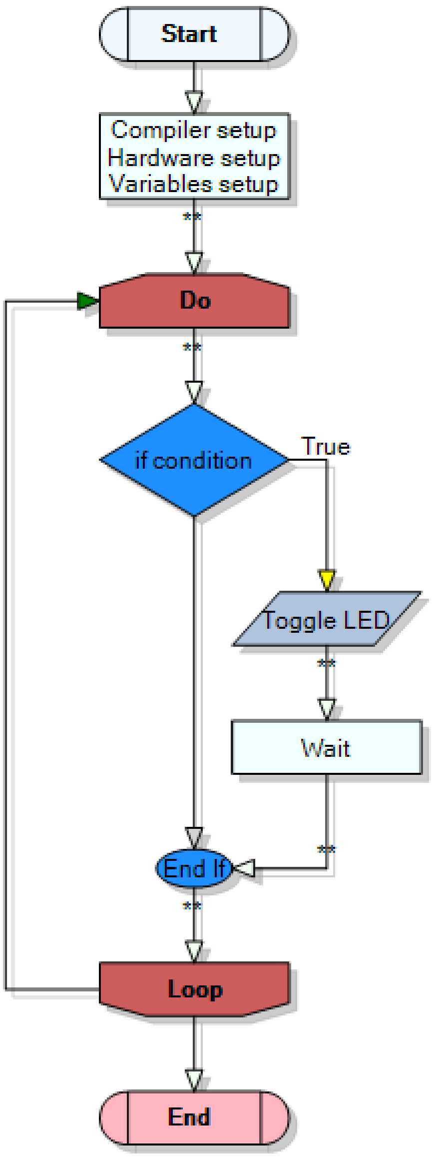

```
'-----------------------------------------------------------
' Compiler Setup (tell Bascom about our micro)
$regfile = "attiny461.dat"       ' the micro
$crystal = 1000000              ' its speed
'-----------------------------------------------------------
' Hardware Setups
' tell bascom how to setup our micro
' setup direction of all ports
Config Porta = Output          'LEDs on portA
Config Portb.5 = Input          'switch on portB
' Hardware Aliases
' tell bascom names for I/O devices
' attached to the Micro
' names are easier to remember than ports
' when writing big programs
RedSw Alias Pinb.5              ' hardware alias
Led Alias PortA.7               ' hardware alias
Const waitdelay = 500
'-----------------------------------------------------------
'  Program starts here
Do
    If Redsw = 0 Then ' wait for switch press
        Led = 1
        Waitms waitdelay
        Led = 0
        Waitms waitdelay
    End If
Loop
End
```

When the switch is pressed and held down, the LED will flash on and off at the rate determined by the waitdelay value.

Notes:

- **when the switch is released the LED will always turn off**
- **Without the delay we cannot see the LED toggle because the micro can toggle the LED really fast, too fast for our eyes to see.**

"When the switch is pressed toggle the LED"




```
'-----------------------------------------------
'Compiler Setup (tell Bascom about our micro)
$regfile = "attiny461.dat"       ' the micro
$crystal = 1000000              ' its speed
'----------------------------------------------
' Hardware Setups
' tell bascom how to setup our micro
' setup direction of all ports

' setup direction of all ports
Config Porta = Output          'LEDs on portA
Config Portb.5 = Input          'switches on portB
' Hardware Aliases
' tell bascom names for I/O devices
' attached to the Micro
' names are easier to remember than ports
' when writing big programs
RedSw Alias Pinb.5                ' hardware alias
Led Alias PortA.7                ' hardware alias
Const waitdelay = 500
'---------------------------------------------
'  Program starts here
Do
    If Redsw = 0 Then ' wait for switch press
        Toggle Led
        Waitms waitdelay
    End If
Loop
End
```

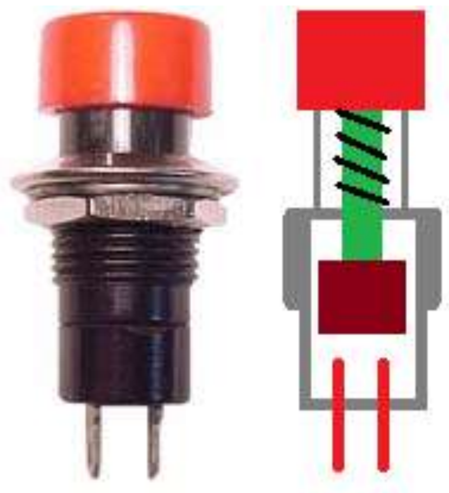
This program also toggle the LED when you hold the switch down**, HOWEVER** when you release the switch, sometimes it will be on and sometimes it will be off and the LED will stay that way.

7.6 If-then exercises

1. Modify the program so that inside the IF-THEN you have your tune played
2. Modify your progam so that inside the IF-THEN you have your knightrider
3. Extension excerise for quick students – get another 2 switches and use them to do different things like play different tunes.

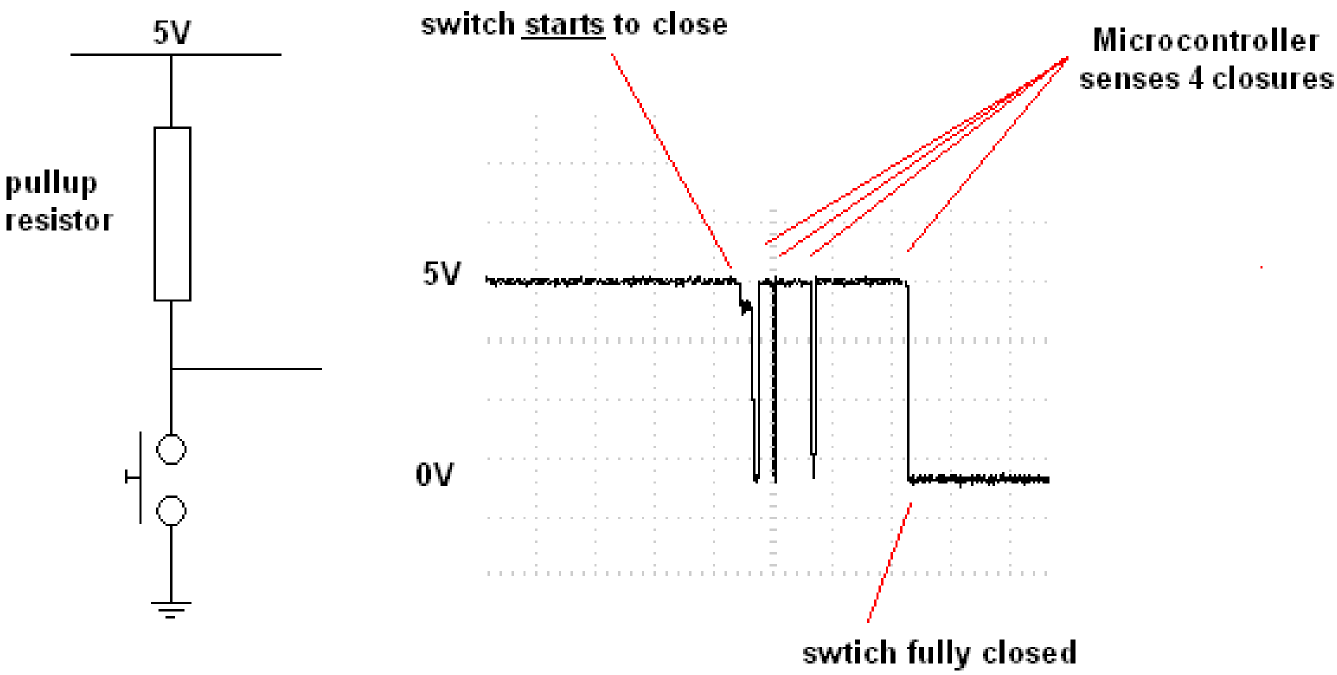
7.7 Switch contact bounce

We have another problem but this one is quite hidden from us; it is called contact bounce.



When someone presses a push button switch the contacts inside the switch move together very fast, and they actually bounce several times together before staying closed. This would be OK if the micro was as slow as we are, however a switch bounce might last 2 or more millseconds, and our microcontroller can detect things as fast as 1microsecond so it might actually think the switch has been opened and closed many times when we pressed it only once! Similarly it might think the switch has been pressed several times when we release it too!

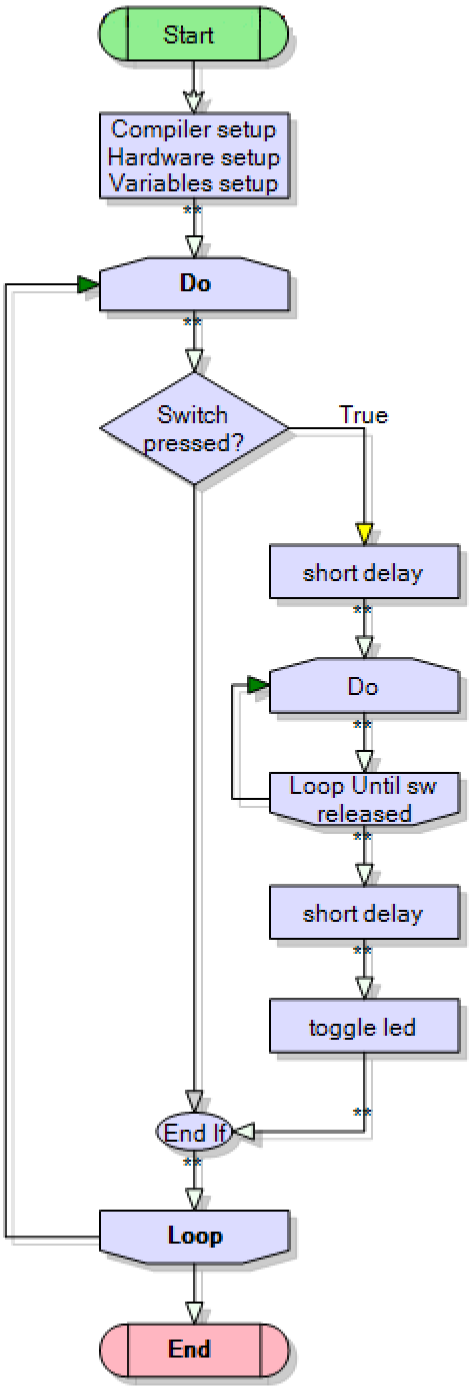
(sometimes you might see this at home with an old lightswitch, sometimes when you turn the light on or off there is a little glow tihin the switch that is sparking caused by the high voltage as the switch contacts bounce.



In this circuit the voltage is being measured and you can see that the switch contacts have bounced 4 times, our micro could easily sense all these bounces as you opening and closing the switch really fast. In the next program we will add some delays to fix this issue.

"If the switch is pressed, only toggle the LED once

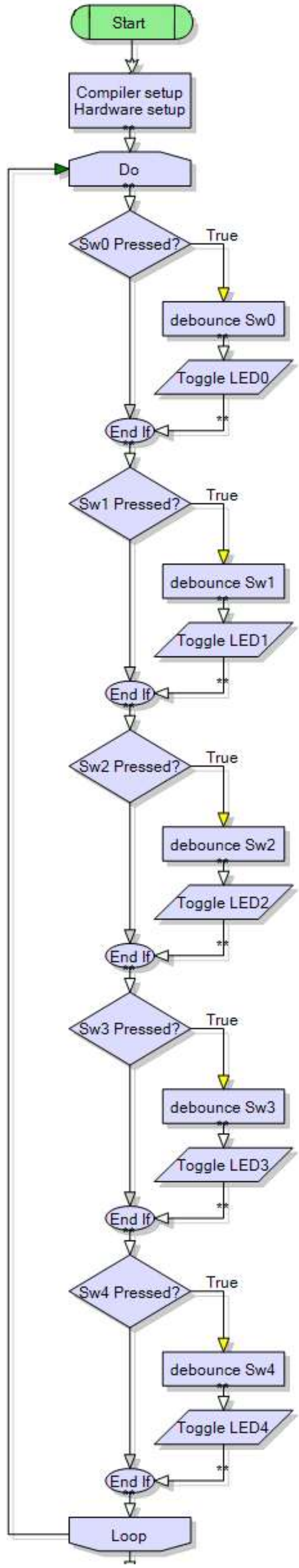
- To do this we check to see if the switch is pressed,
- then we wait a short bit (for the switch to stop any contact bouncing)
- then we wait for the switch to be released
- then we wait for a short bit (for the switch to stop any contact bouncing)
- then we toggle the LED




```
'-----------------------------------------
' Compiler Setup (these tell Bascom things about our micro)
$regfile = "attiny461.dat"      'bascom needs to know the micro
$crystal = 1000000            'bascom needs to know its speed
'-----------------------------------------
' Hardware Setups (these tell bascom how to setup our micro)
' setup direction of all ports
Config Porta = Output        'LEDs on portA
Config Portb.5 = Input        'switches on portB
' Hardware Aliases (these tell bascom names we will use for I/O devices
' attached to the Micro, names are easier to remember that ports)
RedSw Alias Pinb.5            ' hardware alias
Led Alias PortA.7
Const debouncetime = 30
'-----------------------------------------
'  Program starts here

Do
   If Redsw = 0 Then
      Waitms debouncetime
      Do
      Loop until Redsw = 1
      Waitms debouncetime
      Toggle Led
   End If
Loop
End
```

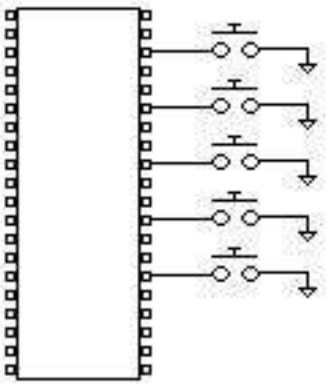
We now have a debounce switch program.

7.8 Reading multiple switches



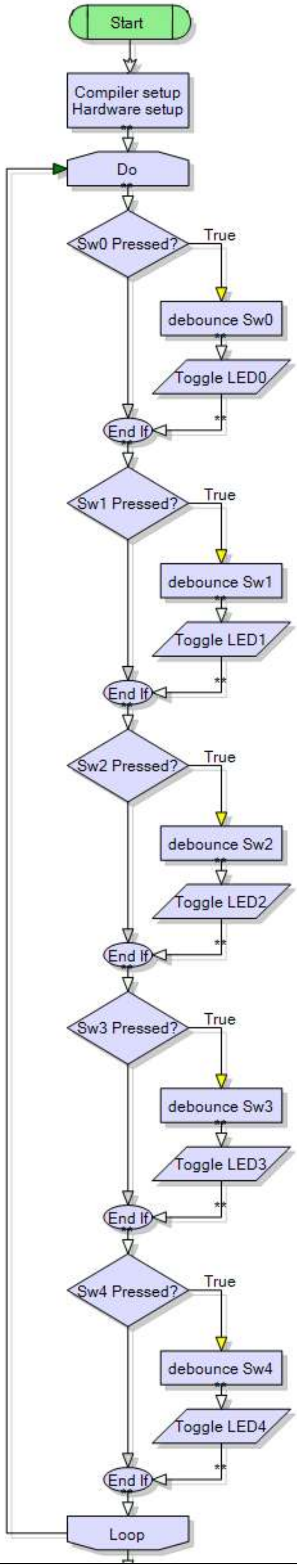
Often the microcontroller is required to read multiple input switches and then control something based upon the switch inputs. These switches might be connected to an assembly line to indicate the presence of an item, to indicate if a window is open or to the landing gear of a jet aircraft to indicate its position.

A common method of using switches within a program is to **poll** the switch (check it regularly to see if it has been pressed).



```
Do
 If Sw0 = 0 Then
     Waitms debouncetime
     Do
     Loop until Sw0 = 1
     Waitms debouncetime
     Toggle Led0
   End If

   If Sw1 = 0 Then
     Waitms debouncetime
     Do
     Loop until Sw1 = 1
     Waitms debouncetime
     Toggle Led1
   End If
     ...
     ...
   If Sw4 = 0 Then
     Waitms debouncetime
     Do
     Loop until Sw4 = 1
     Waitms debouncetime
    Toggle Led4
     ...
   End If
Loop
End
```

7.9 Bascom debounce command



A simpler method of programming when there is a sequence of checking multiple switches is to use the Bascom DEBOUNCE command, when a switch is pressed it is debounced by bascom software code and the subroutine is called (more on subroutines later)

```
'debounceBascom5SwV1

'compiler setup
$crystal = 1000000
$regfile = "attiny26.dat"

'microcontroller setup
Config Porta = Input
Config Portb = Output


Led1 Alias Portb.3
Led2 Alias Portb.4
Led3 Alias Portb.5
Led4 Alias Portb.6
Led5 Alias Portb.7

Sw1 Alias Pina.1
Sw2 Alias Pina.2
Sw3 Alias Pina.3
Sw4 Alias Pina.4
Sw5 Alias Pina.5

'program starts here
Do
   Debounce Sw1 , 0 , Sw1_pressed , Sub
   Debounce Sw2 , 0 , Sw2_pressed , Sub
   Debounce Sw3 , 0 , Sw3_pressed , Sub
   Debounce Sw4 , 0 , Sw4_pressed , Sub
   Debounce Sw5 , 0 , Sw5_pressed , Sub
Loop
End


'Subroutines
Sw1_pressed:
    Toggle Led1
Return

Sw2_pressed:
    Toggle Led2
Return

Sw3_pressed:
    Toggle Led3
Return

Sw4_pressed:
    Toggle Led4
Return

Sw5_pressed:
    Toggle Led5
Return
```

7.10 Different types of switches you can use

Various types of switches can be connected to microcontrollers for various purposes:
Find another type of switch and use it in your program, write it up in your notebook. Use it for the next programs.

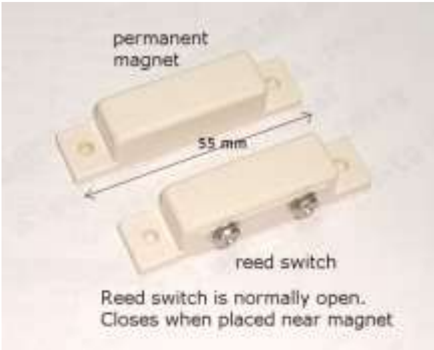
Key switches

So that only authorised people can operate a device

Micro switches

Used inside moving machinery ,on doors and cupboards

Magnetic or Reed switch



Useful for parts or doors that open and close

Tilt or Mercury Switch

Useful to sense movement or something falling over

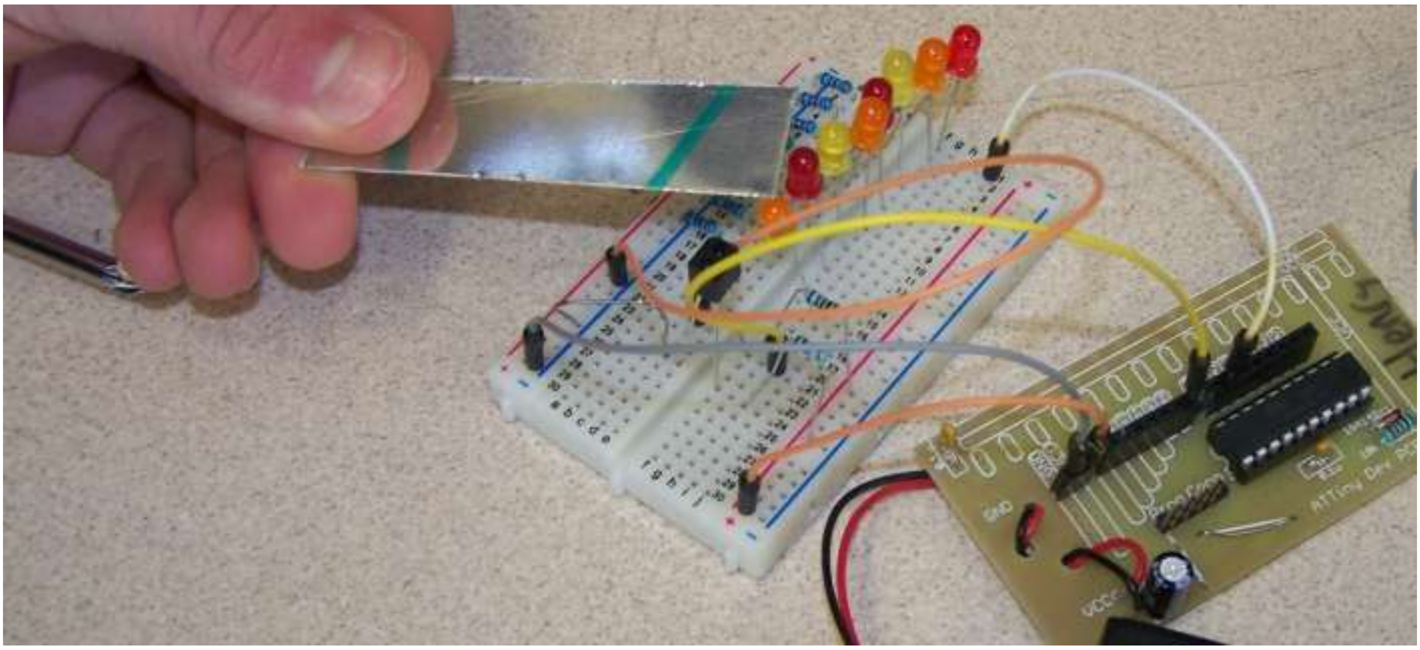
Rotary Switch

Can be used to select one of several different values

Tact switch

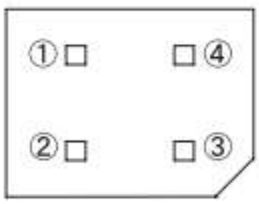
Directly soldered to a circuit board, better quality that the cheap push button switch

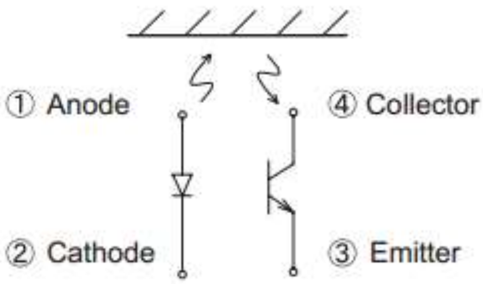
7.11 Reflective opto switch



The RPR220 is a reflective photosensor, it has an LED and a phototransistor built into it. Have a close look at the shape, note that one corner is cut on an angle. This is to help you identify which connection is which.

Looking at the device **from underneath** (from the pins end **NOT** the top) this is the layout





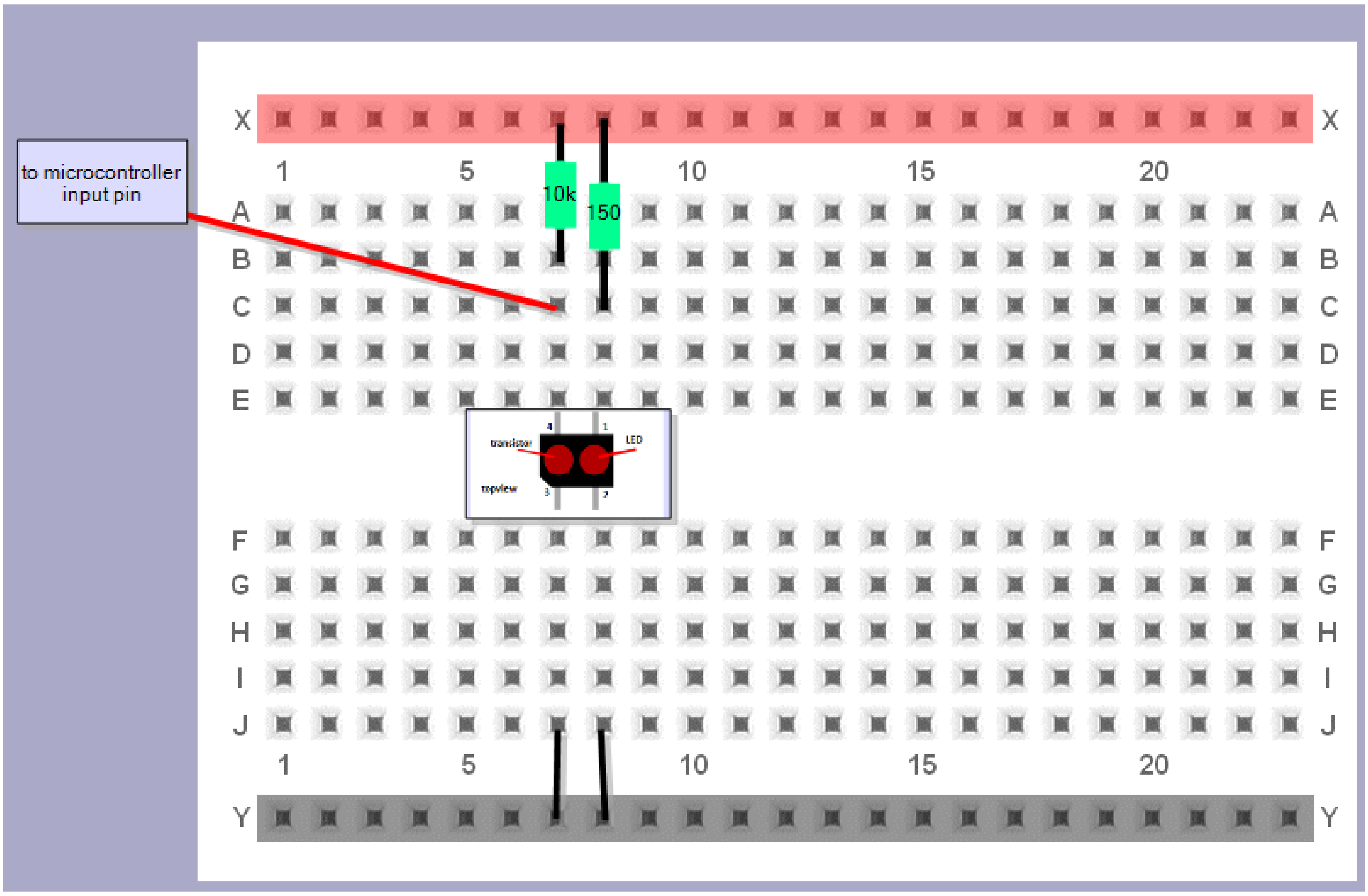
To connect it into a circuit we need two resistors
A current limit resistor for the LED and a pullup resistor for the microcontroller input pin.
The current limit resistor can be calculated suing the data from the datasheet

	Parameter	Symbol	Limits	Unit
Input (LED)	Forward current	I_F	50	mA
	Reverse voltage	V_R	5	V
	Power dissipation	P_D	80	mW

	Parameter	Symbol	Min.	Typ.	Max.	Unit	
Input charac-teristics	Forward voltage	V_F	–	1.34	1.6	V	I_F=50mA
	Reverse current	I_R	–	–	10	µA	V_R=5V

The LED will drop 1.34V and is more powerful than a normal LED as it can handle 50mA.
So using ohms law
R=V/I = (5-1.34)/0.05 = 73 ohms minimum resistance.

For testing purposes a 150 was used, this could be changed to adjust the sensitivity of the unit, maybe a lower value would mean the reflective surface could be further away.



It can be used in a program just like a switch

Opto_sensor alias pin B.6

…

If Opto_sensor = 0 then….

‘do something here

End if

8 Programming Review

8.1 Three steps to help you write good programs

1. Name each program with a meaningful name and save it into its own directory
2. Use a template to setup your program from the start
3. Add lots and lots and lots of comments **as you go**

<u>You must layout programs properly and comment them well to gain achievement</u>

8.2 Saving Programs

When saving programs you need a good quality directory / folder structure, so use a different folder for each program:

- it keeps the files that BASCOM generates for your program in one place
- this helps you find programs quickly when you want to
- it is less confusing
- it is good practice
- Save your program at the beginning when you start it, this helps guard against teachers that like to turn the computers off unexpectedly.

8.3 Organisation is everything

As with structuring and organising your folders you also need to structure and organise your program code.

Messy code is hard to understand and it is surprising how fast you forget what you did; and then when you want to refer to it in the future you find that you cannot understand what you have written!

The use of a template or pattern to follow will help discipline your code writing. Break the code up into the following sections,

- title block
- program description
- compiler directives
- hardware setups
- hardware aliases
- initialise hardware
- declare variables
- initialise variables
- initialise constants
- main program code
- subroutines.
- Interrupt routines

You will really need to be organised with what is coming up.

8.4 Programming template

```
'-----------------------------------------------------------------
' Title Block
' Author:
' Date:
' File Name:
'-----------------------------------------------------------------
' Program Description:




'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"          'the micro we are using
$crystal = 1000000                 'the speed of the micro

'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output              'LEDs on portA
Config Portb =Input                'switches on portB

' Hardware Aliases
Led0 alias portb.0
' Initialise ports so hardware starts correctly
Porta = &B11111111                 'turns off LEDs

'-----------------------------------------------------------------
' Declare Variables

'  Initialise Variables

'  Declare Constants

'-----------------------------------------------------------------
'  Program starts here
Do

Loop
End                                'end program

'-----------------------------------------------------------------
'  Subroutines
```

8.5 What you do when learning to program

1. Develop an understanding of what a computer is and build a correct mental model for one
    a. Input and output conversion at the voltage level
    b. Conversion of input and output voltages into data
    c. Processing and manipulating data which is stored in variables
2. Get to know about the hardware you are using
    a. Get a copy of the datasheet
    b. Learn about the power supply required
    c. Learn how to configure pins as either input or output
    d. Learn how to interface common I/O circuits: LED's, Switches, Piezo, LDR…
    e. Find out about the different types of memory and amount of each
    f. Find out about the speed of processing
3. Get to know the language and the IDE you are using
    a. Learn to access the helpfile (e.g. highlight a word and press F1)
    b. The language has syntax (specific grammar/word rules) you must use correctly
    c. The IDE (Integrated Development Environment) has special commands and built in functions you must know and use: F7, F4, $crystal, $regfile, config, alias, const, port, pin
    d. Learn common I/O functions: set, reset, locate, LCD, GetADC
    e. Understand the limitations of and use variables: byte, word, long, single, double
    f. Use constants instead of numbers in the code (e.g. waitms timedelay)
    g. Get to know the control functions: Do-Loop (Until), For-Next, While-Wend, If-Then (Else)
    h. Get to know about text and math functions (read help file, write a few simple programs using the simulator)
4. Develop Algorithms (written plans for the process the program must carry out)
    a. Have a goal in mind for the program – use specifications and write a simple brief
    b. Plan your I/O by drawing a system block diagram
    c. Determine variables and constants required in the program
    d. Determine the state of all the I/O when the program begins
    e. Write the algorithm – Identify, order and describe the major processes the micro must do.
5. Draw Flowcharts or Statecharts (visual diagram for the process the program must carry out)
    a. Identify the blocks/states that will be used
    b. Use arrows to link the blocks and visualise control processes and program flow
6. Develop code from the flowcharts
    a. The outer looping line is replaced with a do-loop
    b. Backwards loops are replaced with do-loop do-loop-until, for-next, while-wend
    c. Forward loops are generally replaced with If-Then-EndIf
    d. Replace the blocks with actual commands
    e. Layout the code with correct indentations(tabs) to improve readability
    f. Learn to comment code so that it explains what is happening (not just describes)
    g. Use subroutines to organise complex code so that logic code is separate from I/O code
    h. Trial different ways of solving the problem and keep records of you experiments

This is not a step by step process; as when you get to know about one area you get to know about others at the same time. The key to gaining depth in your knowledge and understanding comes from **LOTS OF EXPERIMENTATION!** That means making mistakes and above all having fun, you need to know that **good decisions come from experience and experience comes from bad decisions!!!** So experimenting is ok.

In your electronics courses at school the aim is not to make you an expert in all the above (expertise comes after about 10 years working in an area), the aim is to introduce you to microcontroller electronics and programming, and to understand some of what is happening in the world around you and to feel able to see that you can control it and not have it control you.

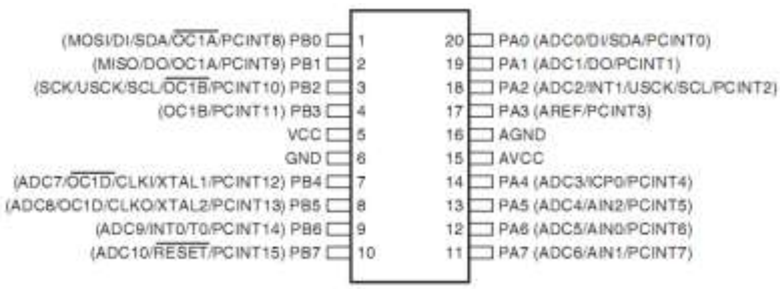
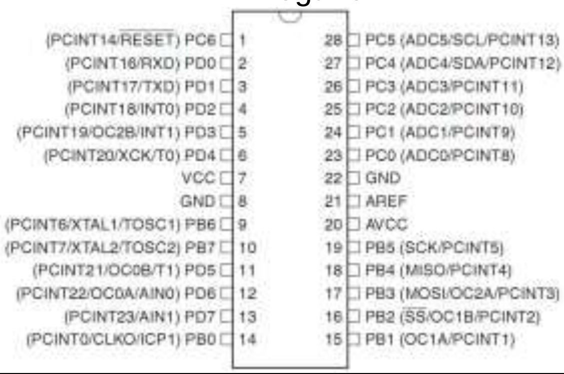
8.6 AVR microcontroller hardware

A microcontroller is a general purpose electronic circuit; it is a full computer inside a single integrated circuit (IC or chip). Often ICs have fixed functions e.g. the TDA2822M amplifier or LM358 opamp, they only do one job and their input and output pins have fixed roles, so you have limited control over what they do, and therefore limited control over how to connect them.
With a microcontroller however you are in control, you decide:

- what the function of the IC is
- what most of the pins are used for (inputs or outputs)
- and what external input/output devices these pins are connected to.

If you want an egg timer, a car alarm, an infrared remote control or whatever, it can all be done with a microcontroller.

A commercial range of microcontrollers called ‘AVR’ is available from ATMEL (www.atmel.com). You could start with the ATTiny461, it has 4kbytes of Flash for program storage, 128 bytes of Ram and 128 bytes of EEPROM for long term data storage. Or you could start with the ATMega48, it has 4kbytes of Flash, 512 bytes of RAM and 256 bytes of EEPROM.

ATTiny461	ATMega48
	

Important pins:

- VCC & GND are dedicated for power, VCC is positive voltage and GND is negative
- AVCC and AREF are special pins for measuring analog voltages (connect to VCC).
- I/O ports are a group of 8 I/O pins which can be controlled together
- MOSI, MISO, SCK and RESET are pins used to upload the programs.
  (You cannot use RESET as an I/O pin, but MOSI, MISO, SCK can be used with care)

8.7 Power supplies

Most microcontrollers work off low voltages from 4.5V to 5.5V, so yours can be run off batteries or a dc power pack, voltages in excess of these will destroy the micro. Check the datasheet to see what the range is for your micro, the ATTINY26-16PI will work from 4.5 to 5.5V, the ATMEGA48-10PU will work from 2.7V to 5.5V.

8.8 BASCOM and AVR assignment

Learning goal:
Students should become independent learners able to find support to help their own learning

The AVR is a microcontroller from which manufacturer________________

The URL for their website is: ________________________

Download the specific datasheet for our microcontroller (the summary version not the full version) and print the first 2 pages and put them in your journal.

The Programmable Memory size is ______ The SRAM size is _______The EEPROM size is _______

The number of I/O lines is __________ and they are arranged in _______ports

BASCOM-AVR is a compiler from _____________________

The URL for their website is: ________________________

Download the latest version of the BASCOM AVR demo and install it on your PC.

There are a number of application notes on the website for the AVR
Describe what AN128 is about

____

____

____

There are a number of other great resource websites for the AVR and BASCOM
Find 3 websites on the internet that have useful resource information on BASCOM
List the websites URL and what you found there

____

____

____

____

____

____

____

The ATTiny26 datasheet is full of useful information here is what some of it means

Program memory note we use the demo version of Bascom which has a 4k limit, this chip has only 2K

EEPROM is non volatile data storage (it is not lost when the power goes off)

Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- RISC Architecture
  - 118 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
- Data and Non-volatile Program Memory
  - 2K Bytes of In-System Programmable Program Memory Flash
    Endurance: 10,000 Write/Erase Cycles
  - 128 Bytes of In-System Programmable EEPROM
    Endurance: 100,000 Write/Erase Cycles
  - 128 Bytes Internal SRAM
  - Programming Lock for Flash Program and EEPROM Data Security
- Peripheral Features
  - 8-bit Timer/Counter with Separate Prescaler
  - 8-bit High-speed Timer with Separate Prescaler
    2 High Frequency PWM Outputs with Separate Output Compare Registers
    Non-overlapping Inverted PWM Output Pins
  - Universal Serial Interface with Start Condition Detector
  - 10-bit ADC
    11 Single Ended Channels
    8 Differential ADC Channels
    7 Differential ADC Channel Pairs with Programmable Gain (1x, 20x)
  - On-chip Analog Comparator
  - External Interrupt
  - Pin Change Interrupt on 11 Pins
  - Programmable Watchdog Timer with Separate On-chip Oscillator
- Special Microcontroller Features
  - Low Power Idle, Noise Reduction, and Power-down Modes
  - Power-on Reset and Programmable Brown-out Detection
  - External and Internal Interrupt Sources
  - In-System Programmable via SPI Port
  - Internal Calibrated RC Oscillator
- I/O and Packages
  - 20-lead PDIP/SOIC: 16 Programmable I/O Lines
  - 32-lead QFN/MLF: 16 programmable I/O Lines
- Operating Voltages
  - 2.7V - 5.5V for ATtiny26L
  - 4.5V - 5.5V for ATtiny26
- Speed Grades
  - 0 - 8 MHz for ATtiny26L
  - 0 - 16 MHz for ATtiny26
- Power Consumption at 1 MHz, 3V and 25°C for ATtiny26L
  - Active 16 MHz, 5V and 25°C: Typ 15 mA
  - Active 1 MHz, 3V and 25°C: 0.70 mA
  - Idle Mode 1 MHz, 3V and 25°C: 0.18 mA
  - Power-down Mode: < 1 µA

The speed of the micro - or number of operations per second is set by a clock signal - we will use the internal clock which has been factory set at 1MHz, = 1MIPS, 1 million opeartions per second.

RAM is temporary or volatile data storage (it is lost when the power goes off)

We are using the ATtiny26-16PU, the 16 is the speed grade not the number of I/O pins

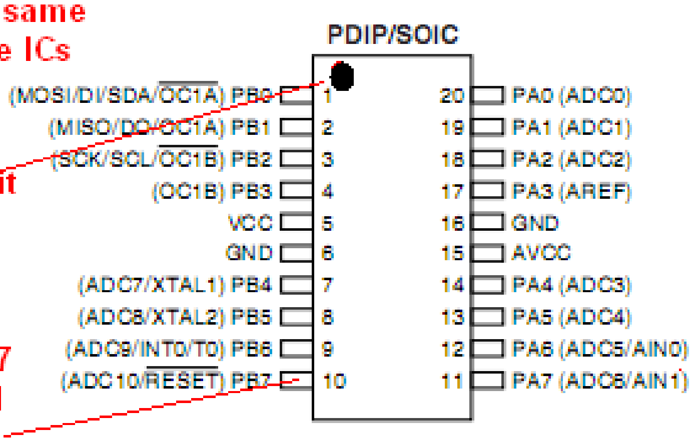
P means dip package (dual in line plastic)

we use the 26 not the 26L note the voltage range

get to know the pin numbering its the same for all dip package ICs

pin 1 generally has a dot next to it

there are 16 I/O lines but we can only use 15 as PB7 or reset is needed for programming



although there are 16 I/O lines we cannot use PB7 beacuse we need the RESET pin for programming the IC

8.9 Programming words you need to be able to use correctly

Find definitions for them

computer	
microcontroller	
hardware	
software	
memory	
RAM	
variable	
data	
byte	
word	
program	
algorithm	
flowchart	
BASIC	
port	
code	
upload	
compile	
command	
repetition	
do-loop	
for-next	
subroutine	
gosub	
return	

8.10 Year10/11 typical test questions so far

What have you learned about connecting power to a microcontroller?
What is a typical power supply voltage?
What range of voltages is acceptable?
Which pin(s) are positive and which are negative?
What are the names for these pins?
What batteries would you use?
What have you learned about programming a microcontroller?
What is the software we are using called? Where does it come from?
What does IDE stand for?
What are the names for the 4 different parts of the IDE software?
How many wires are there in the programming cable?
What happens if $regfile is wrong?
What happens if $crystal is wrong?
What does compiling mean?
What have you learned about interfacing LEDs to a microcontroller?
Draw the connection for an LED and resistor to a microcontroller. Draw this on a bread board diagram as well. Are these series or parallel?
What is a typical value of resistor?
What would be a minimum value?
What would be a maximum value?
What does the toggle command do?
What have you learned about programming the pins of a microcontroller?
How many I/O pins does an ATTiny461 have?
With an LED on A.5 and a switch on A.7 write the config statements for both
What are the different commands for driving a single output pin?
What command can you use to drive multiple output pins all at once?
What have you learned about program style?
We 'tab' or indent code for what reason?
Why do we comment programs? Write comments for a simple flashing LED program.
What is const used for? Write a few lines of program that uses const.
What is alias used for? Write a few lines of program that uses alias.
What have you learned about making sound?
How is a piezo connected?
What is the command used to make sound?
Write a line of code to show how it the command used?
What have you learned about interfacing switches?
What is the resistor in the circuit called?
Why is it necessary?
What value is typically used?
Draw the circuit for a switch connected to a microcontroller?
Explain the code used to test a switch to see if it pressed?
What is the problem with switch contact bounce for software?
.

9 Introduction to program flow

9.1 Pedestrian crossing lights controller

Client, customer or end-user: …

Description of the problem, issue, need or opportunity (diagrams may be required):

Vehicles travel at high speeds on this road and although there is a pedestrian crossing, pedestrians are at considerable risk

Conceptual Statement:

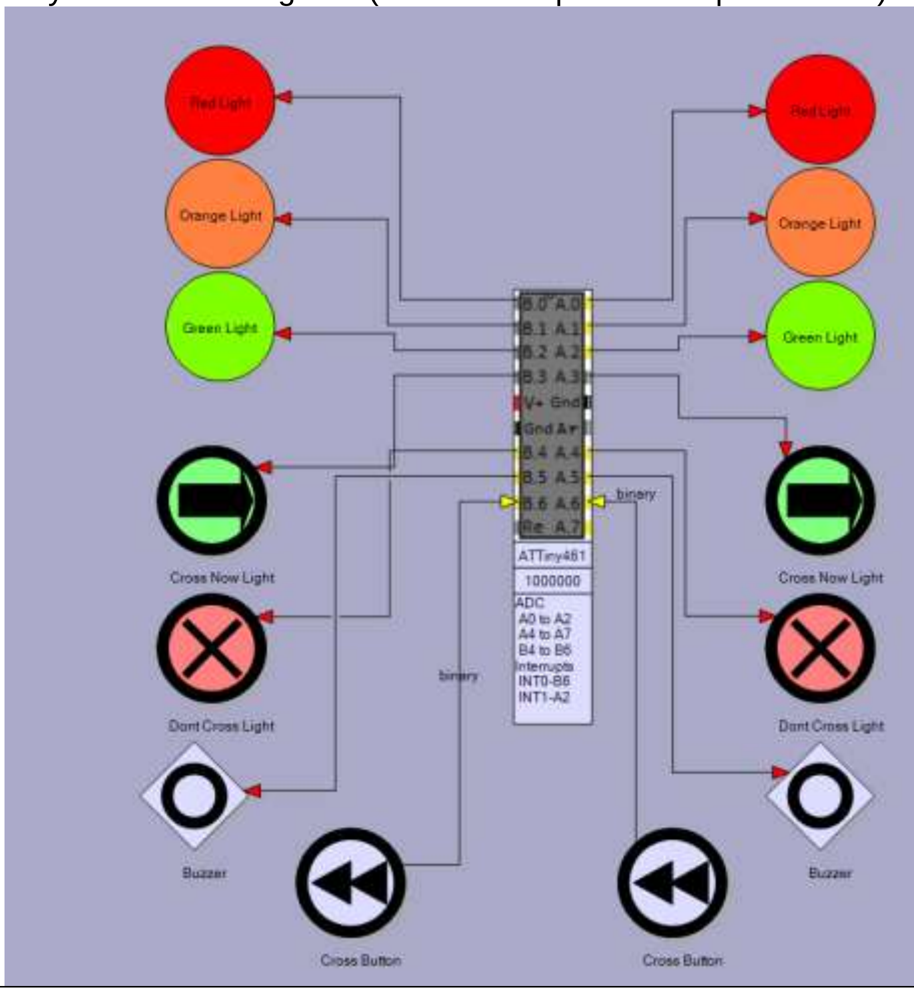
Design and construct a set of traffic lights for a pedestrian crossing

Functional attributes:

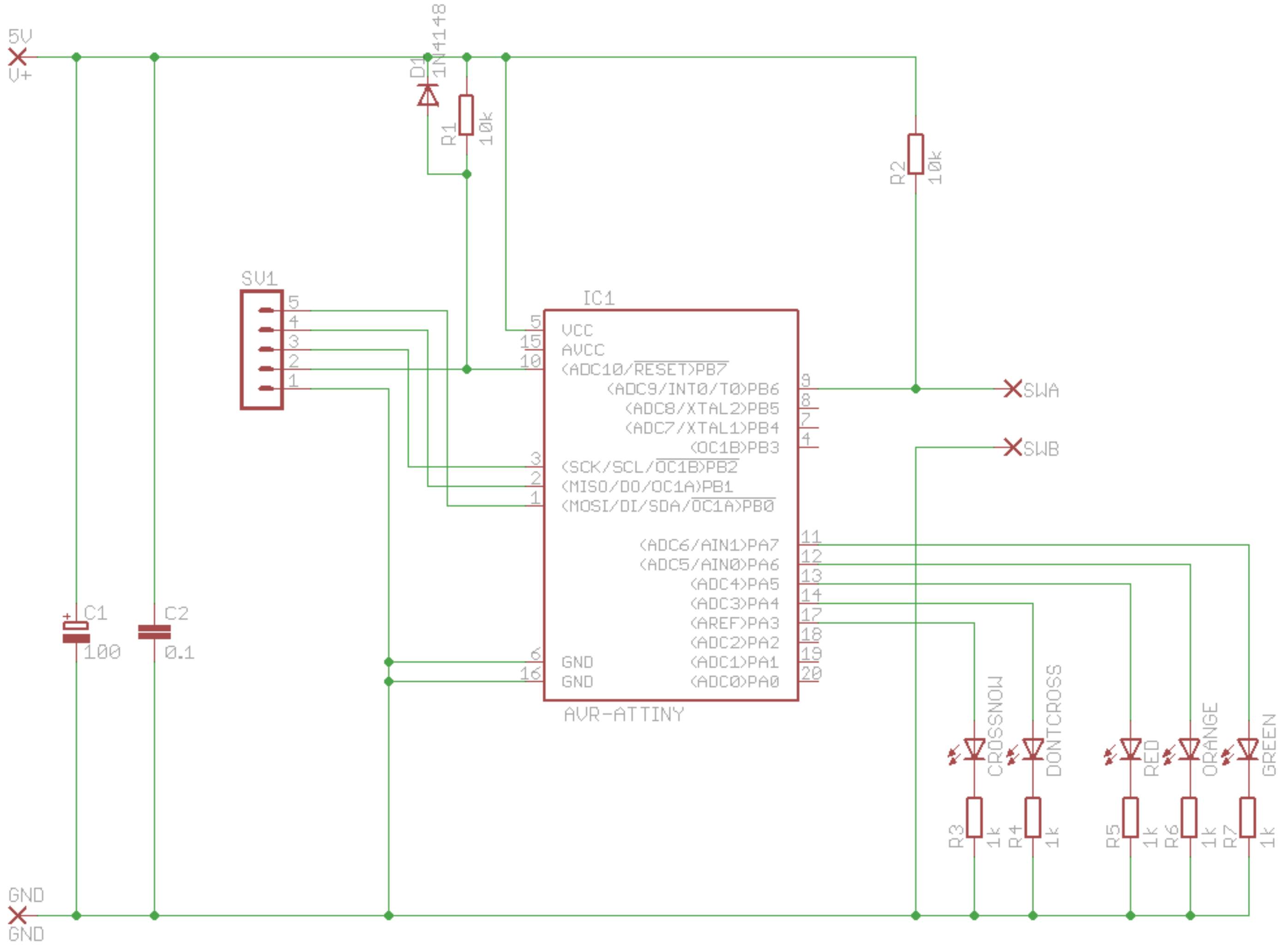
When the button is pressed the lights change from green to orange,
there is a delay of 25 seconds
Then the lights go red
There is a delay for 1 minute
Then the lights go back to green,

Cross and DontCross lights work as expected.

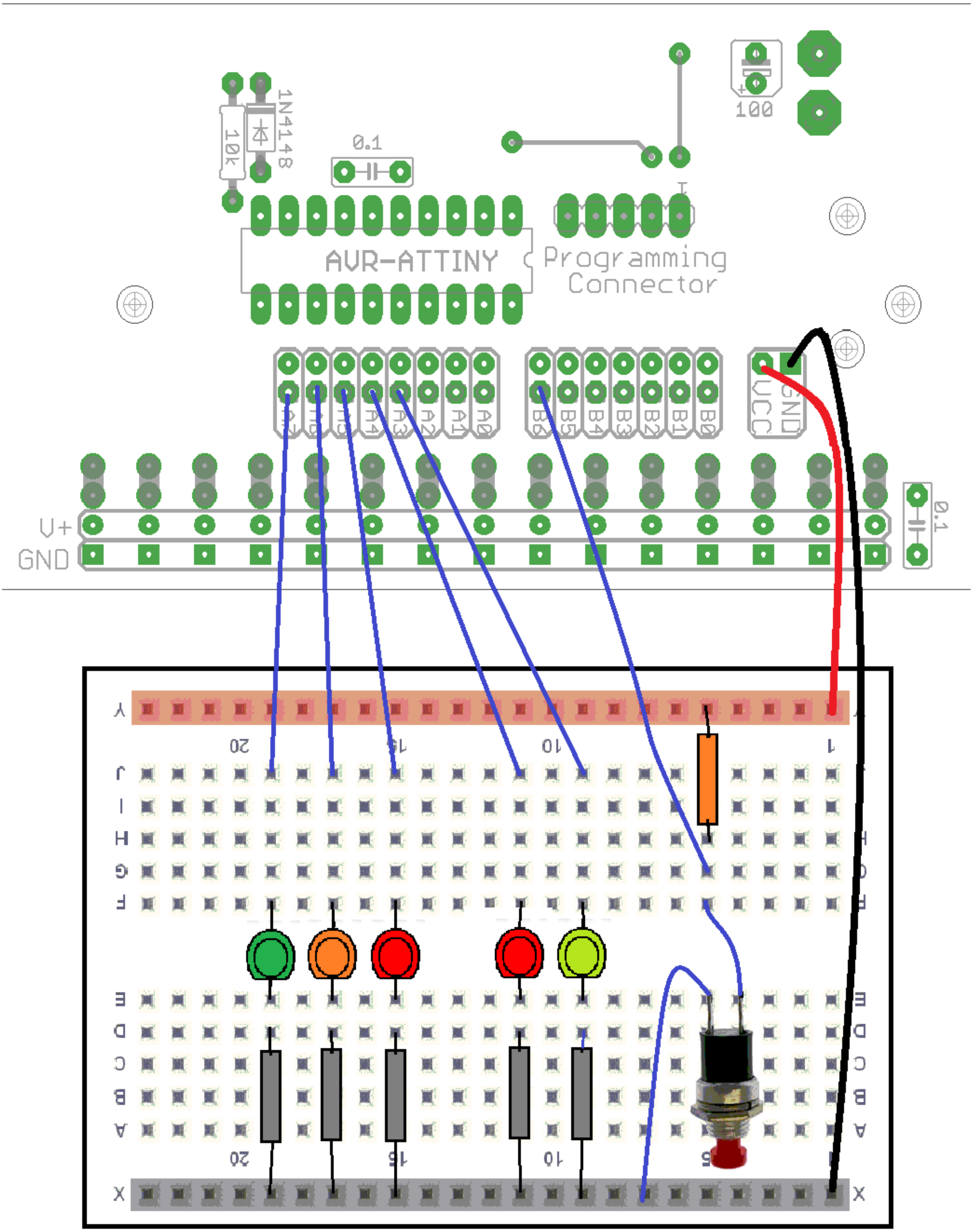
System Block Diagram: (include all input and output devices)



9.2 Pedestrian Crossing Lights schematic



9.3 Pedestrian Crossing Lights PCB Layout



9.4 Algorithm planning example – pedestrian crossing lights

(define the operation of the system)

Name: _______________ Project: _______________ Date: _____

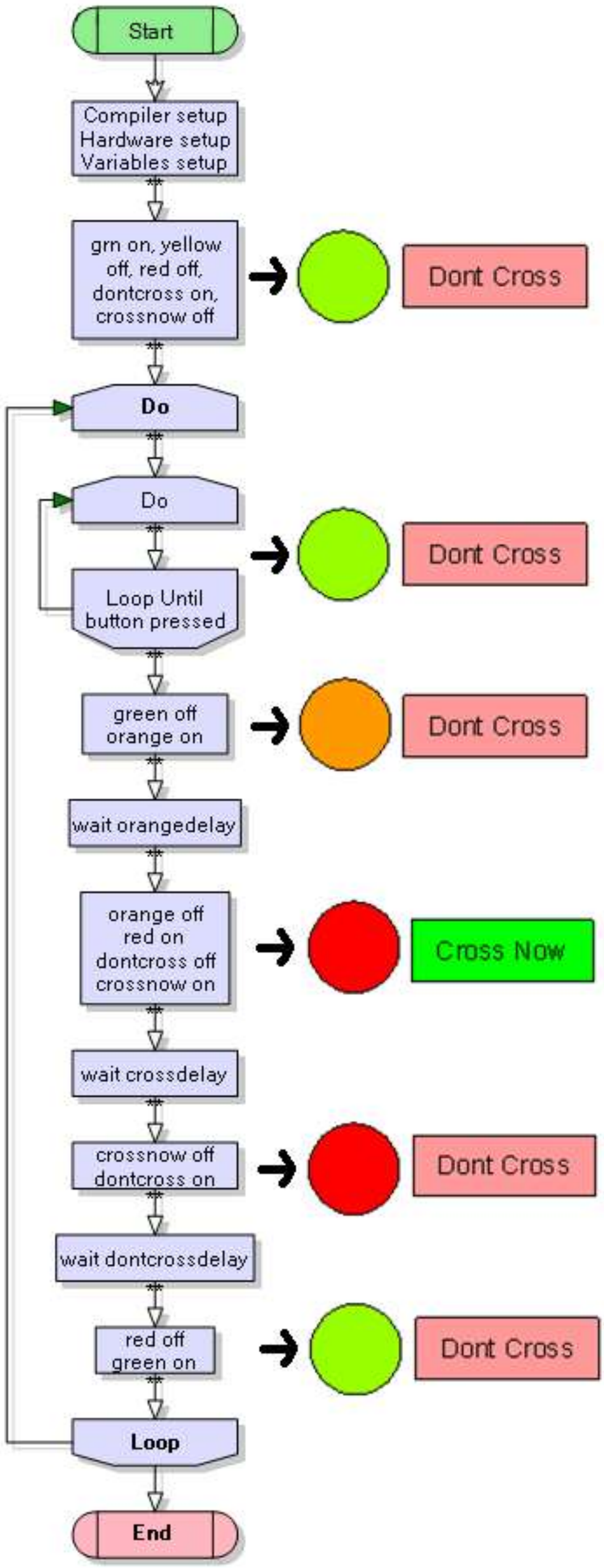
Define all the input and output devices				
Inputs		**Outputs**		
Device Description	**Name**	**Device Description**	**Name**	**Starting State**
Large buttons on each pole for pedestrians to press to cross	CROSSBUTTON	RED traffic lights for cars on pole	REDLIGHT	OFF
		Orange traffic lights for cars	ORANGELIGHT	OFF
		Green traffic lights for cars	GREENLIGHT	ON
		Buzzer to indicate to pedestrians to cross now	BUZZER	OFF
		CROSS NOW light on each pole	CROSSNOW	OFF
		DON'T CROSS light on each pole	DONTCROSS	On

The algorithm
Initially the Redlight , orangelight, buzzer and cross are off, Greenlight, dontcross are on
For each input describe what happens to any output devices Use **"if __________ then _________"** or **" do___________ until ___________"** statements
If the pedestrian presses the crossbutton then The greenlight goes off, the orange light goes on Then after 25 seconds the orangelight goes off the redlight goes on the don't cross goes off the cross now goes on Then after 1 minute the red light goes off the cross now goes off the don't cross comes on the green light comes on

9.5 Flowchart planning example – pedestrian crossing lights

Programs flow in sequence and can be represented well with flowcharts

Note how the planning for this program includes a graphic detailing the colour of the lights, this helps visualise the program and is an excellent example of choosing a planning tool that will help your thinking.



9.6 Getting started code

```
' PedestrianCrossingsVer1.bas
' B.Collis 1 Aug 2008
' reads a switch to check if pedestrian wants to cross
$crystal = 1000000
$regfile = "attiny461.dat"
Config Porta = Output
Config Portb = Output
Config Portb.6 = Input

'here we use aliases to make the code easy to write and easy to read
'lights for cars
Greenlight Alias Porta.7
Orangelight Alias Porta.6
Redlight Alias Porta.5
'lights for pedestrians
Dontcrosslight Alias Porta.4
Crossnowlight Alias Porta.3

Crossbutton Alias Pinb.6

'we need different delays for different purposes
Const Orangedelay = 10
Const Crossdelay = 20
Const Dontcrossdelay = 5

'initial state of lights for cars
Greenlight = 1                                                 'on
Orangelight = 0                                               'off
Redlight = 0                                                  'off
'initial state of lights for pedestrians
Dontcrosslight = 1                                            'on
Crossnowlight = 0                                             'off

Do
      'wait for pedestrian to press button
      Do
      Loop Until Crossbutton = 0

      Greenlight = 0
      Orangelight = 1
      Wait Orangedelay

      'you finish the rest of this code

Loop
End
```

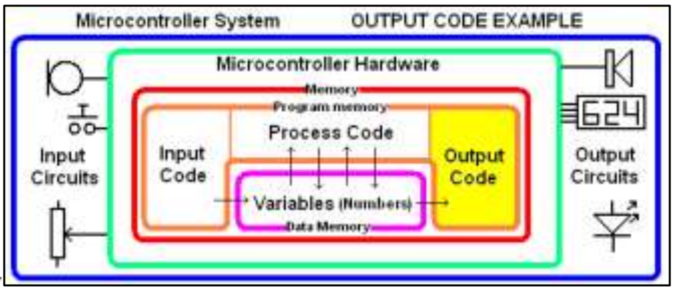
9.7 Modification exercise for the pedestrian crossing

1. Generally the dontcross light is off until the pedestrian presses the button
2. After the redlight comes on there be a short delay before the crossnow
3. Put a 5 second delay into the system after the pedestrian pushes the button and before the light goes red.
4. Implement a short beep into the system when the cross now light comes on

Achieved	Merit	Excellence
Implements 1 above into the algorithm AND the program AND adds useful describing comments in the program	Also impliments 2 above in both the algorithm AND the program AND uses comments to explain the program	Implements 3 above in both the algorithm AND program AND with good explanatory comments in the program.

Can you see that achievement criteria are actually algorithms?
SO MAKE SURE YOU UNDERSTAND THEM!

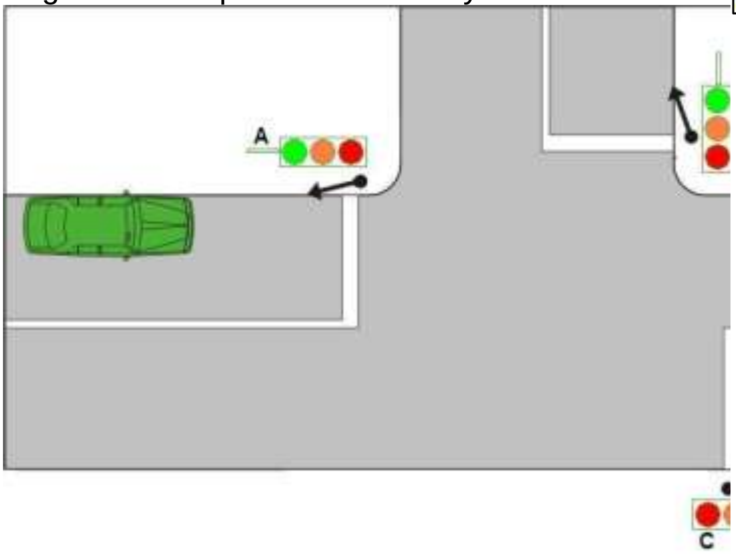
9.8 Traffic lights program flow



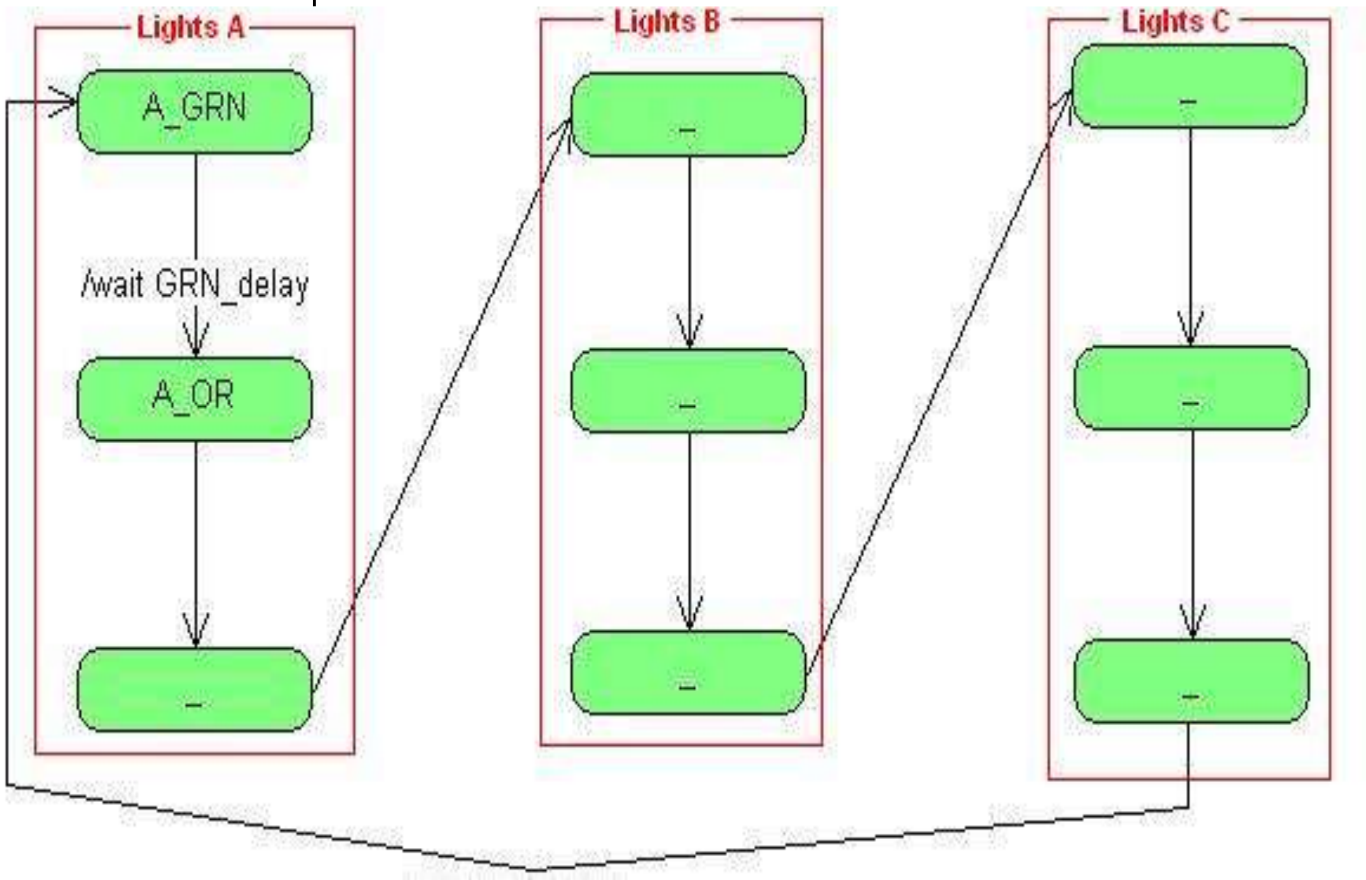
Learning to develop useful planning tools to help solve problems such as drawings, block diagrams, tables & flowcharts.

Learning about the Bascom commands ALIAS

1. Understand the situation by drawing a planning diagram that explains the road layout



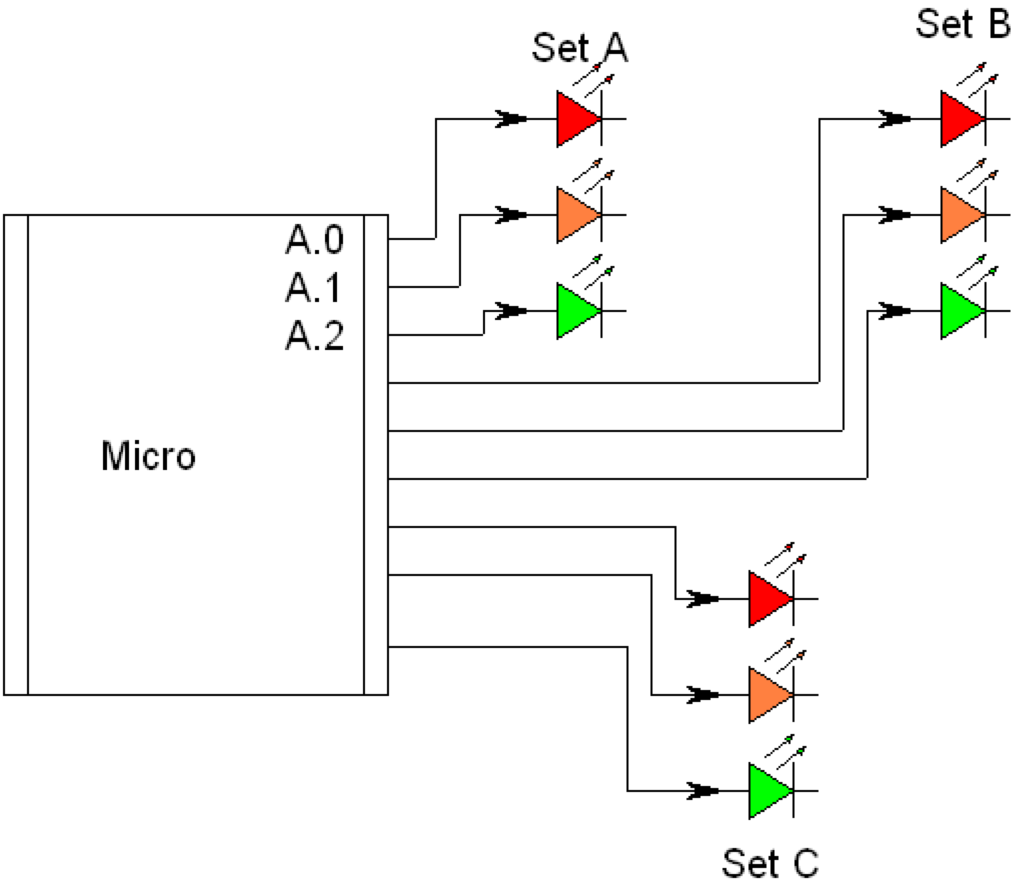
2. The traffic light sequence process is actually very confusing and a planning tool such as a sequence diagram will help you plan the program. Complete this sequence which shows which lights come on in the sequence



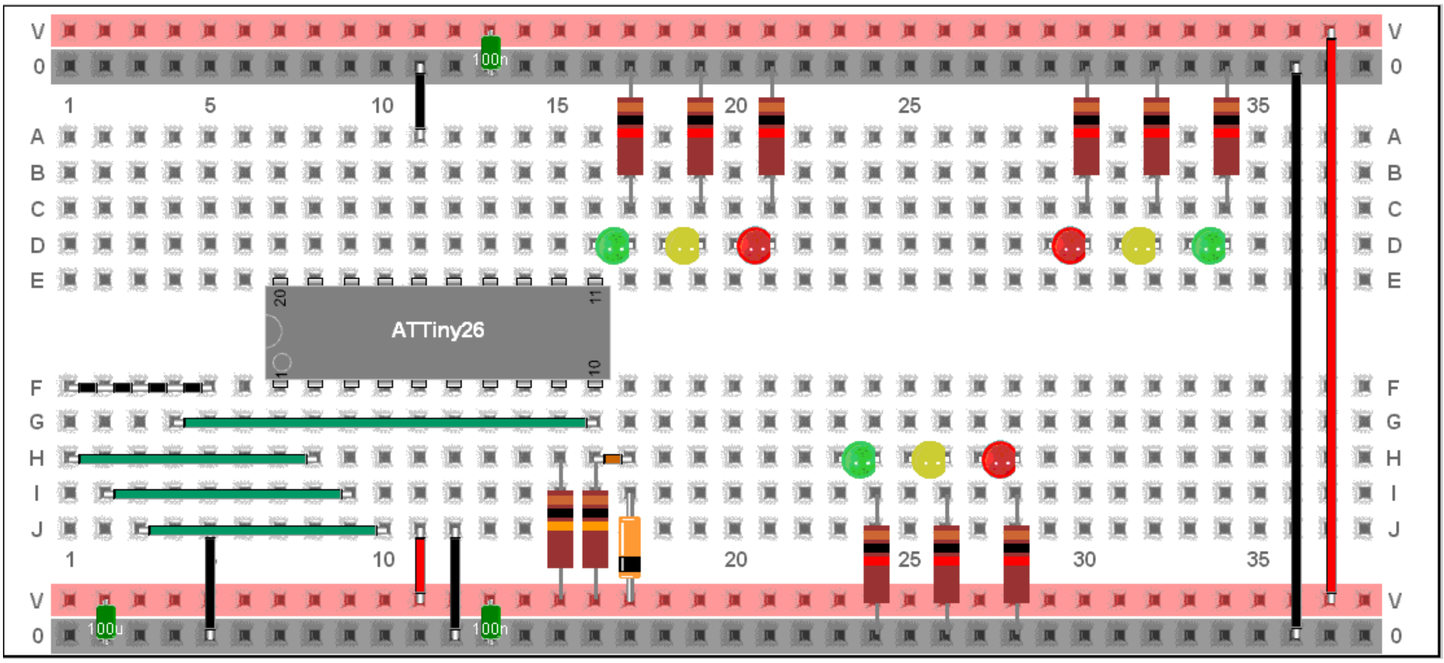
How long should the delays between LED changes be for real traffic lights?
In our model we only need to test that the sequence is correct so we will choose shorted delays

Real lights	Our Model for testing purposes will be
Green is on for 1 minute	Grn_delay = 8
Orange 30 seconds?	Or_delay = 3
Delay after one road goes red before the green for the next road goes on	Red_delay = 1

3. Draw a system block diagram – which shows important connections within the system, but is not a full circuit diagram (complete the schematic below with the pin connections for Set B and Set C.
Label the rst of this diagram with the pins on the micro you will use for the other 2 sets of lights. Take special note that you will have to use at least one of the output pins on portb. I chose portB.4.

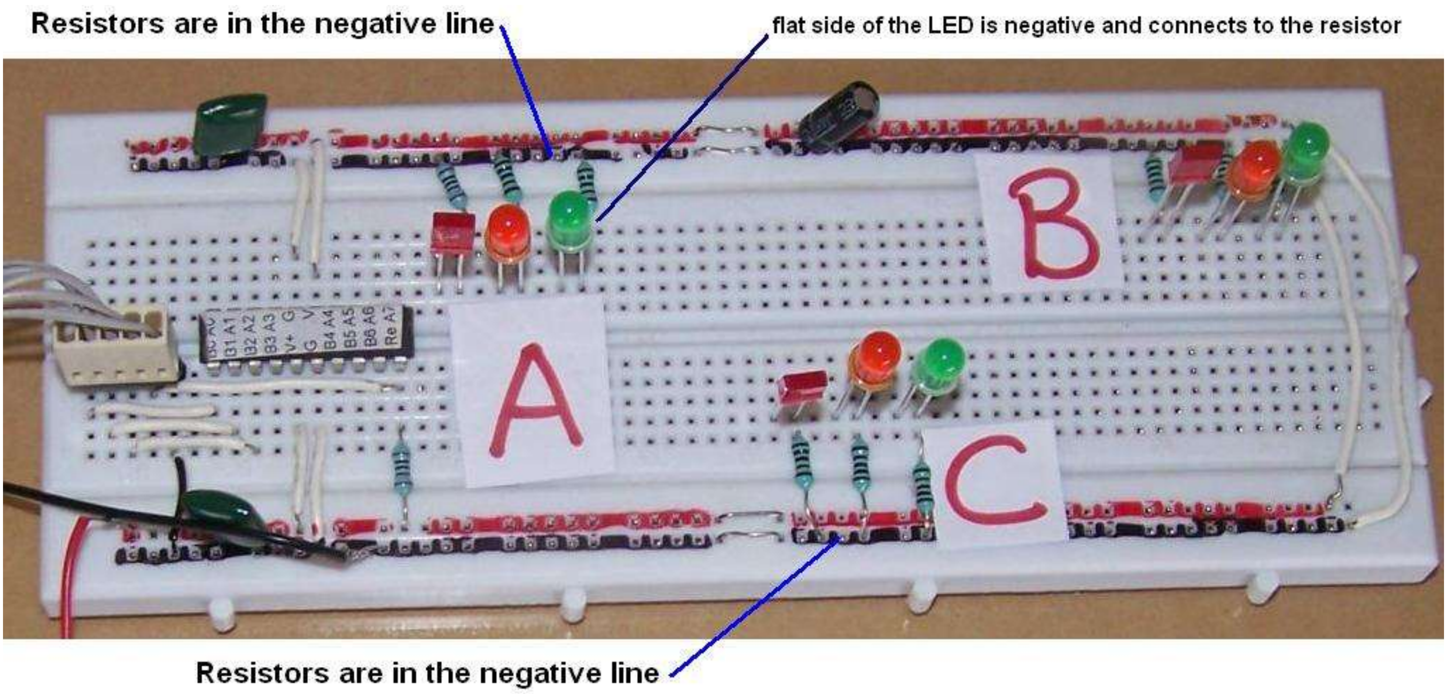


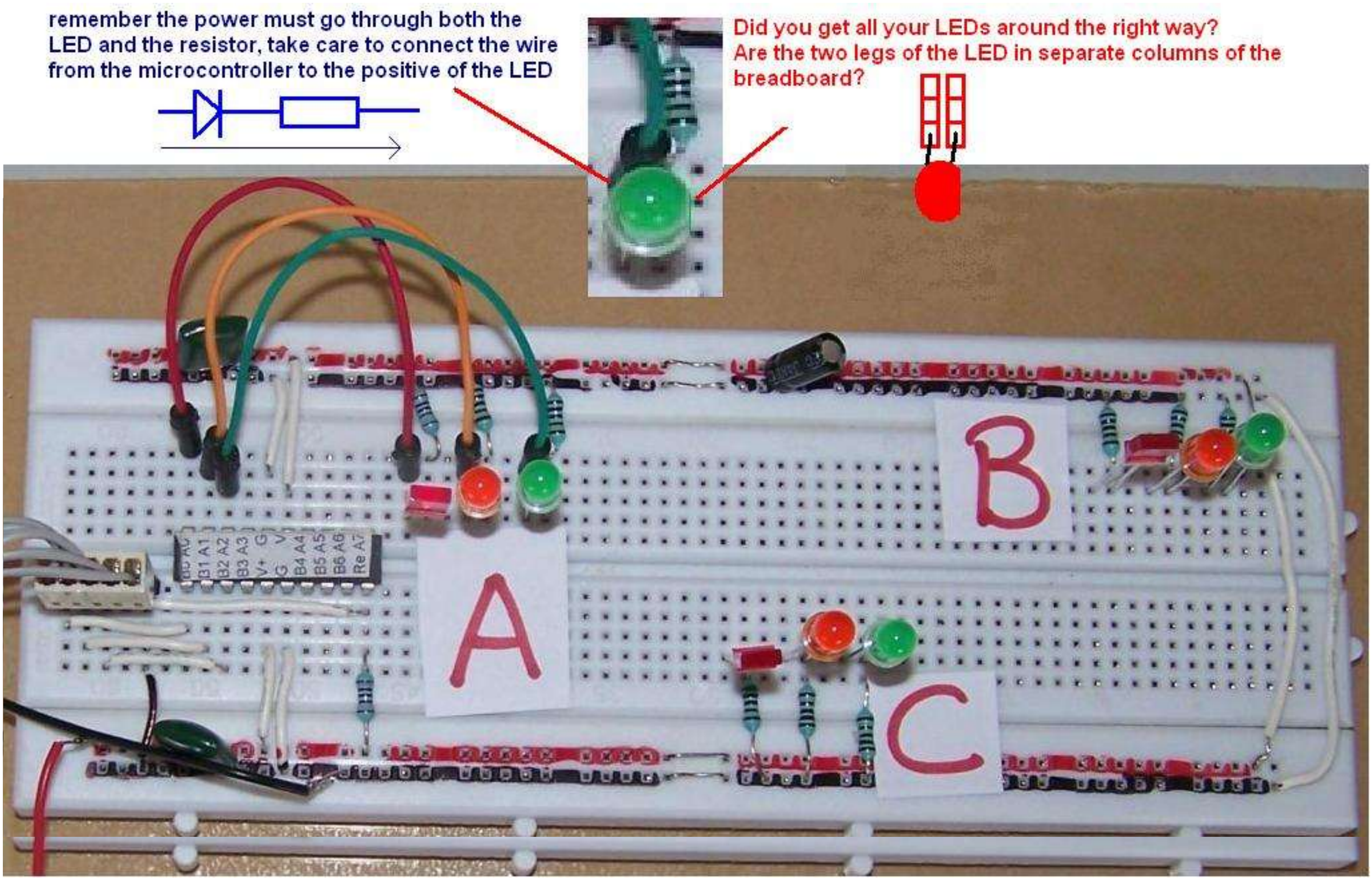
4. Do the physical wiring of the 3 sets of LEDs to the microcontroller.
- Layout the physical LEDs to follow the real physical layout
- Use appropriate coloured LEDs
- Keep it tidy, use short wires.



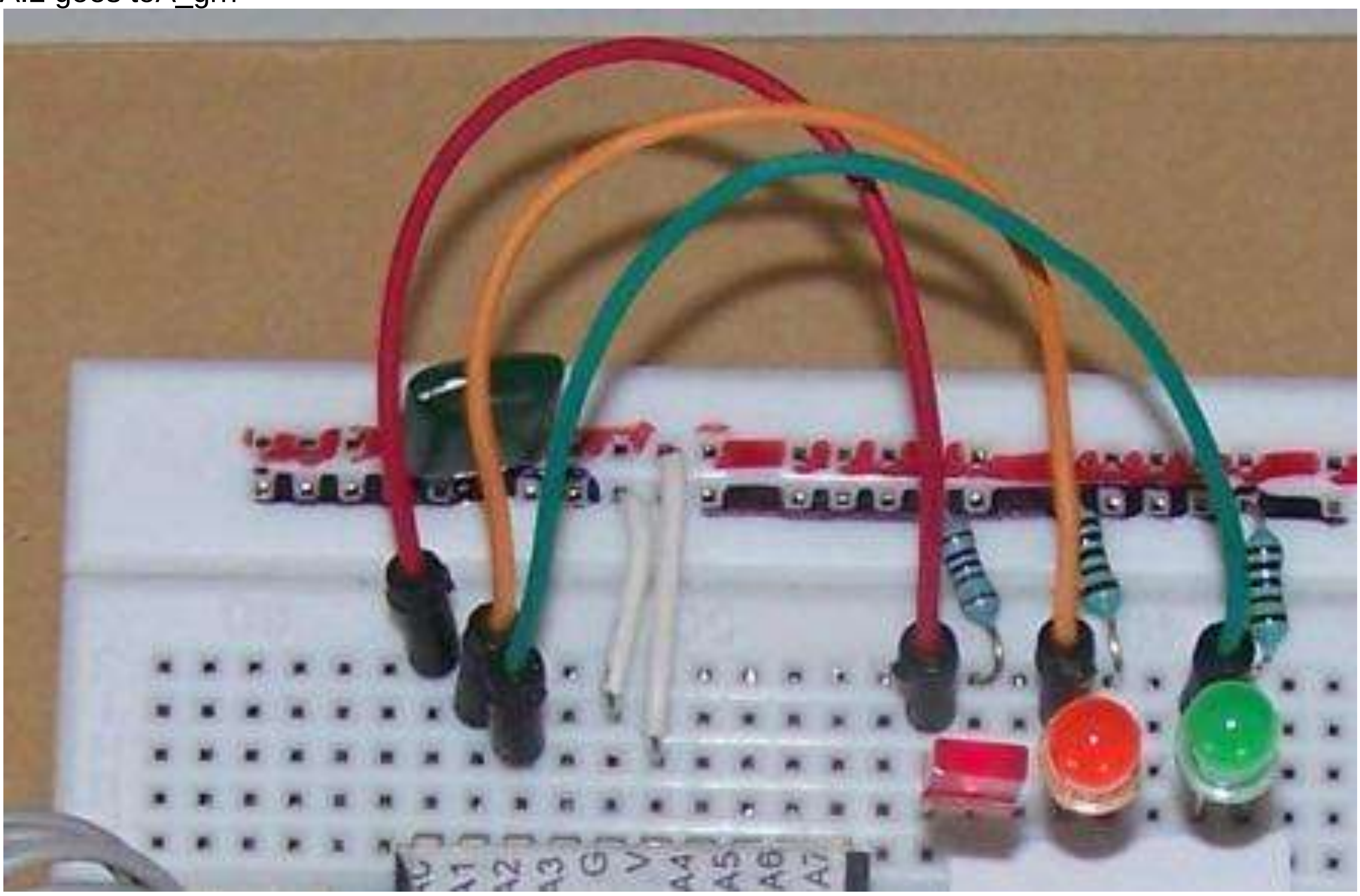
Here are some photos of the process

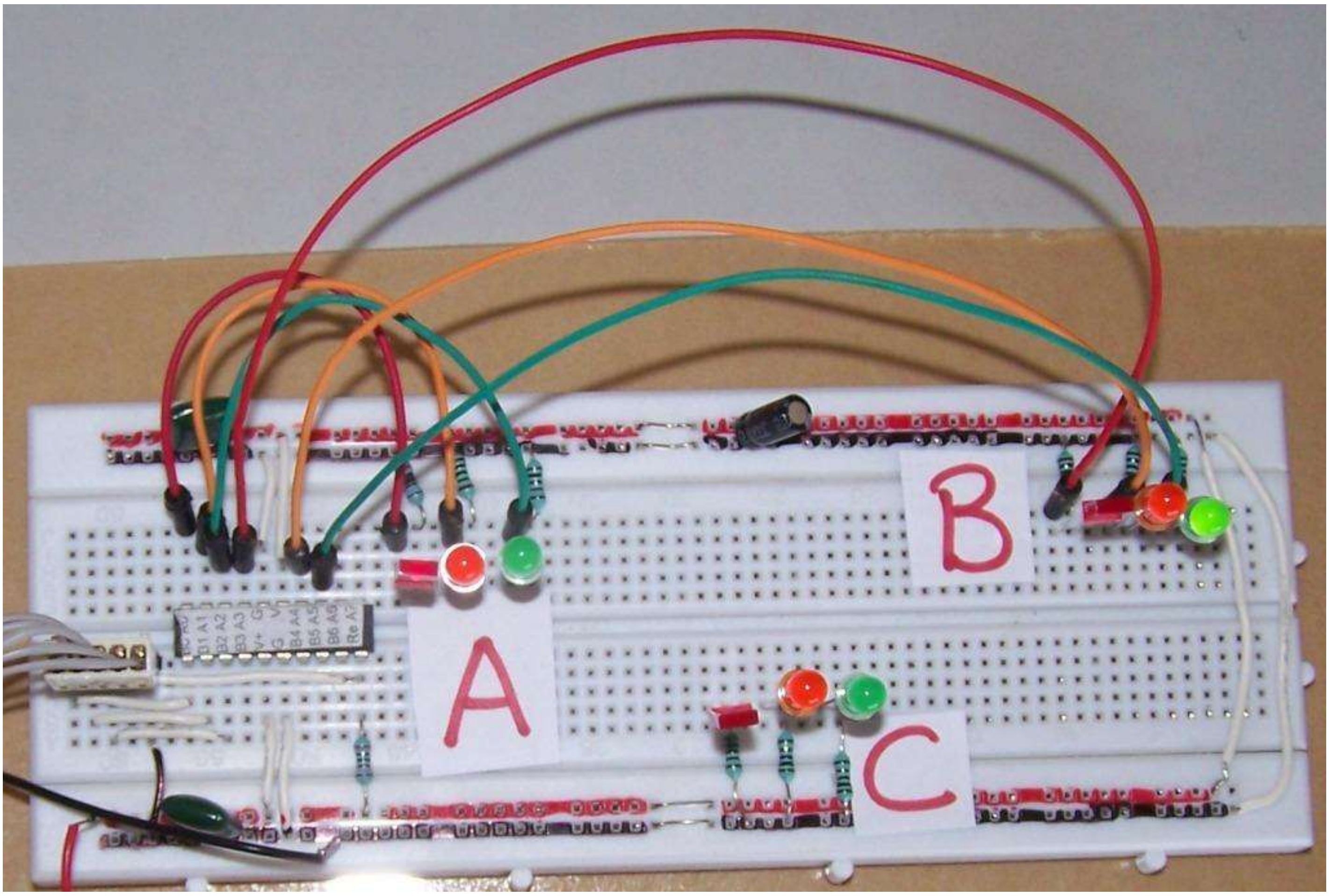
Wiring stage one: all the LEDs and resistors are mounted



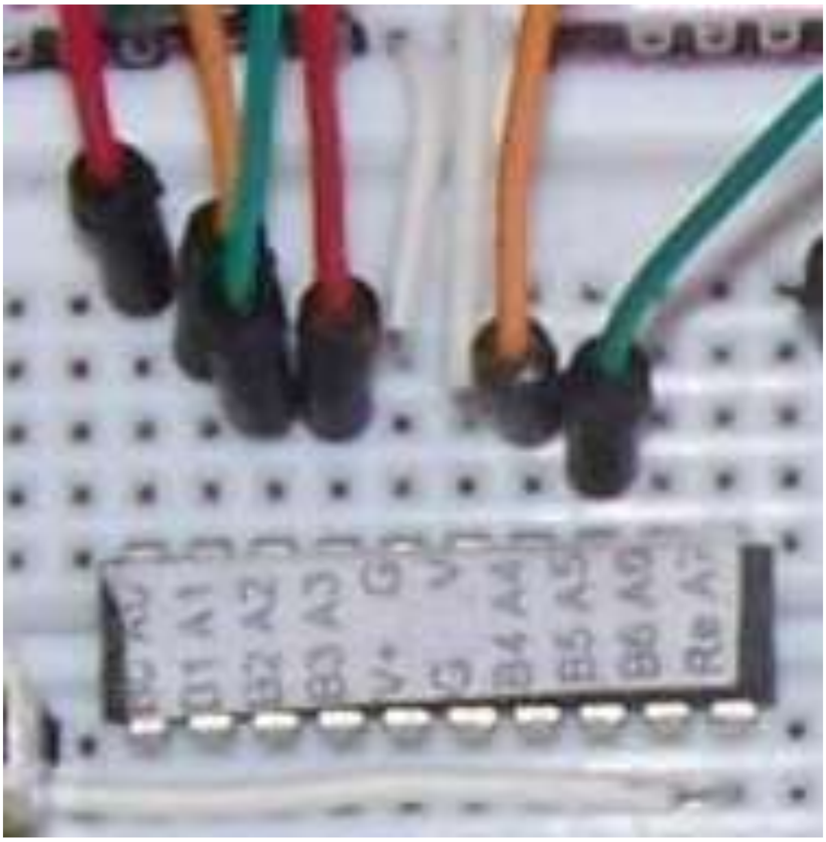


Wiring stage two: the ‘A’ set of lights are wired up
A.0 goes to A_red
A.1 goes to A_or
A.2 goes toA_grn





Wiring stage 3: B set of LEDs are wired to three ports of the microcontroller, here I have chosen portA.3, portA.4,portA.5.



Note thatportsA.3, A.4 and A.5 are used
Also note that the G(ground) and V(positive voltage) pins are not connec ted to I/O devices but to the power supply!

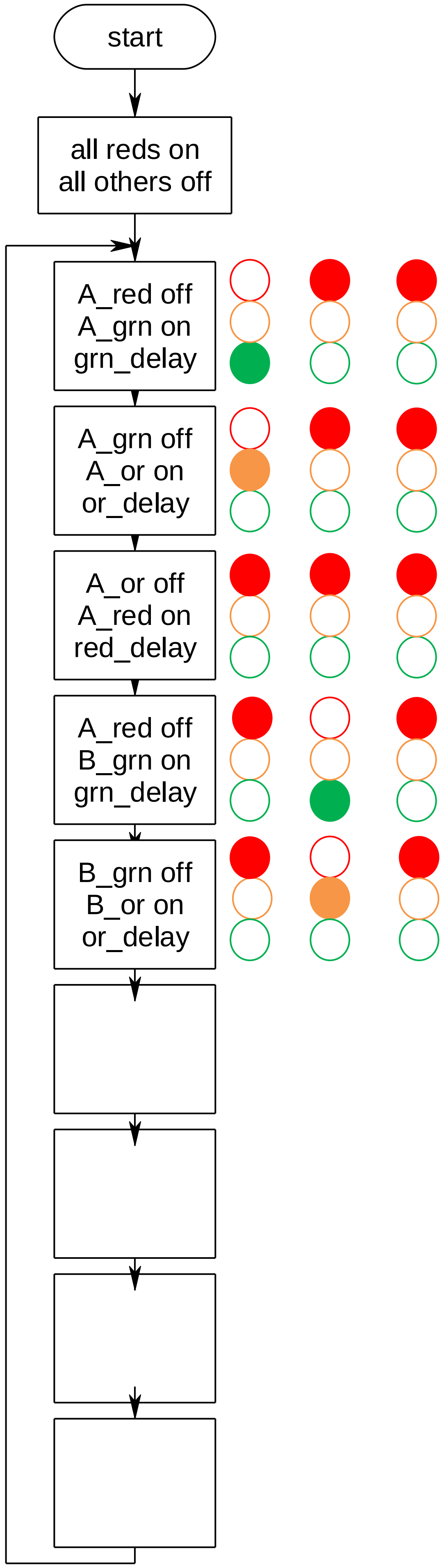
Can you complete the last stage of the LED wiring? You will have to put one of the LEDs on portB. I chose portB.4

If you need more help search the rest of the book for the last picture.

5. Complete this flowchart in your workbook with the rest of the sequence. There are 9 LEDs so there will need to be 9 stages inside the loop.

CAN you see thepattern emerging

Having the flowchart will help you debug (correct errors in your program) later on



6. Write your

```
'TrafficLightsVer1.bas
'B.Collis
```

program, things to work on in your program: • Describe the hardware at the top of the file and use aliases for the port pins that describe what is connected to each one • Use spaces to help layout your program so it looks good • Comment your program with short clear descriptions • Use constants with good names e.g. waitms red_delay	see code below

```
'**************************
$crystal = 1000000
$regfile = "attiny26.dat"

Config Porta = Output
Config Portb = Output

'**************************
'LED connections
'use aliases so that the program is easier to write and
understand
A_red Alias Porta.0
A_or Alias Porta.1
A_grn Alias Porta.2

B_red Alias Porta.3
B_or Alias Porta.4
B_grn Alias Porta.5

C_red Alias Porta.6
C_or Alias Porta.7
C_grn Alias Portb.4
'use constants to make the program easier to read and to
modify
Const Grn_delay = 8                     'green on time
Const Or_delay = 3                      'orange on time
Const Red_delay = 1                     'safety delay

'initially set the red lights on and all others off
'introducing the new commands SET and RESET to
individually control port pins
A_red = 1                               'on
B_red = 1                               'on
C_red = 1                               'on
A_or = 0                              'off
A_grn = 0                             'off
B_or = 0                              'off
B_grn = 0                             'off
C_or = 0                              'off
C_grn = 0                                 'off
Do
   'A lights
   A_red = 0                         'off
   A_grn = 1                            'on
   Wait Grn_delay

   A_grn = 0                        'off
   A_or = 1                            'on
   Wait Or_delay

   A_or = 0
   A_red = 1
   Wait Red_delay          'delay for red light runners!

   'B lights
   B_red = 0
```

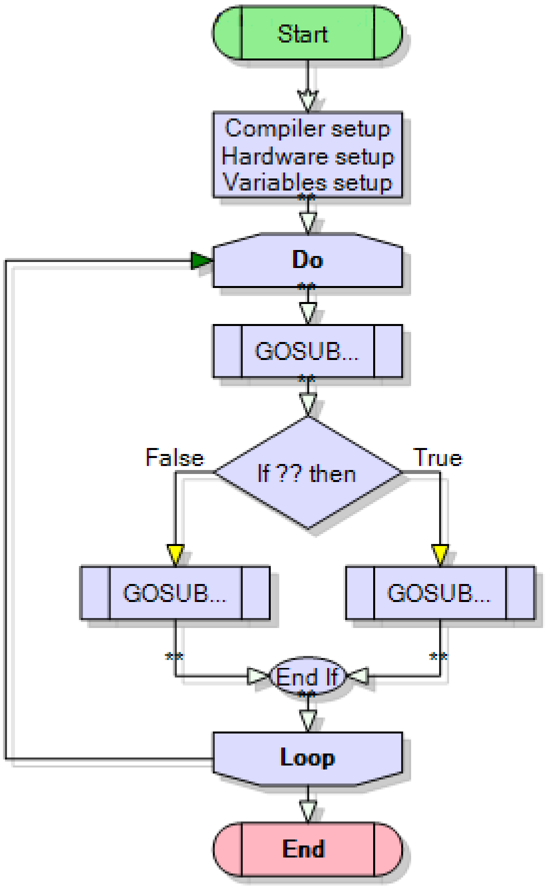
	``
	   B_grn = 1 'grn on
	   Wait Grn_delay
	
	   B_grn = 0 'grn off
	   B_or = 1
	   Wait Or_delay
	
	   B_or = 0
	   B_red = 1
	   Wait Red_delay 'delay for red light
	runners!
	
	   'C lights you write the rest of the code
	
	Loop
	End
	``

10 Introductory programming - using subroutines

Once a program gets large we need to learn how to manage it properly.
Subroutines have been seen already when we have used the debounce command but here is a list of what they can do for you:

Refine you code by Reducing, Reusing & Recycling

- **Reduce the complexity of your programs, by hiding detail**
- **Reuse - reuse the program code multiple times within the same program**
- **Recycle – you can use the same program code easily in other programs**



Here is an example of calling some subroutines

```
Do
        Gosub test_sensor
        If sensor_output = 0 then
                gosub got_it
        Else
                gosub tell_the_user_again
        End if
Loop
End
```

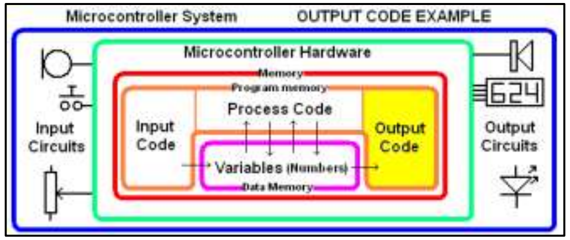
And another example

```
DO
        Gosub test_sensor
        If sensor_output =10 then gosub do_a
        If sensor_output =11 then gosub do_b
        If sensor_output =12 then gosub do_c
        If sensor_output =13 then gosub do_d
        If sensor_output =14 then gosub do_e
        …
Loop
End
```

You can see that they really can de-complicate code (make it easy to read and understand) by removing a lot of I/O code

Subroutines are used to make code easier to read, understand and maintain, however they can be used well or used poorly. The clue to using subroutines well is to keep the logic for the program in the main loop and the input and output detail in the subroutines. As above and in the next example.

10.1 Sending Morse code



Morse code is a form of communication used in the early days of telegraph and radio communication when voice could not be sent just short messages using codes. It was also used between ships using lights.
Draw a flowchart and write a program to send your name using Morse code.

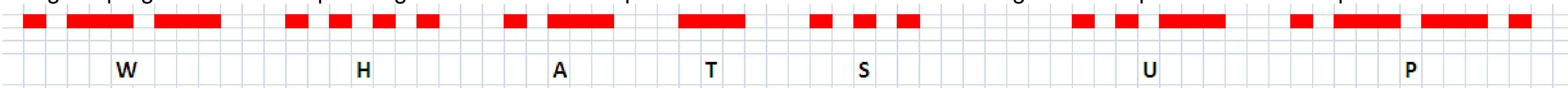
A	•—	H	••••	O	———	U	••—	1	•————	6	—••••
B	—•••	I	••	P	•——•	V	•••—	2	••———	7	——•••
C	—•—•	J	•———	Q	——•—	W	•——	3	•••——	8	———••
D	—••	K	—•—	R	•—•	X	—••—	4	••••—	9	————•
E	•	L	•—••	S	•••	Y	—•——	5	•••••	0	—————
F	••—•	M	——	T	—	Z	——••				
G	——•	N	—•								

To make sense timing is important so we will follow these rules

- A dash is equal to three dots
- The space between the parts of the same letter is equal to one dot
- The space between letters is equal to three dots
- The space between two words is equal to seven dots

If you wanted to send a short sentence like “ whats up.” It is crucial that you get the gaps between letters , parts of letters and parts of words correct or the message willl not be understandable by the person receiving it.

Using the program Excel as a planning tool we can draw up a chart that shows the correct timing for the sequence for ‘whats up’.



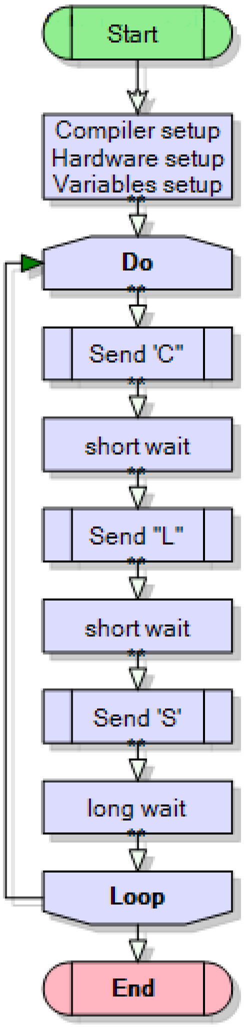
Check that:

- the width of 1 dot it is 1 cell in excel
- the width of 1 dash is 3 cells,
- the gap between parts of a letter is 1 cell,
- the gap between letters is 3 cells
- the gap between words is 7 cells.

A program like this though could be very very long so we will break it up into sections called **subroutines** by putting the I/O code into subroutines

The use of subroutines as well as comments, aliases and constants will make your code easier to understand and maintain .

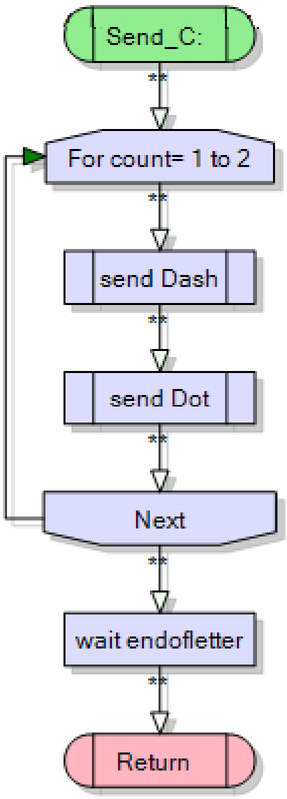
Uncommentedandpoorlysetoutcodeislikereadingasentencewithoutpunctuationorspacesthemeaningisstilltherebutitisalittlehardtofollowandunderstand.

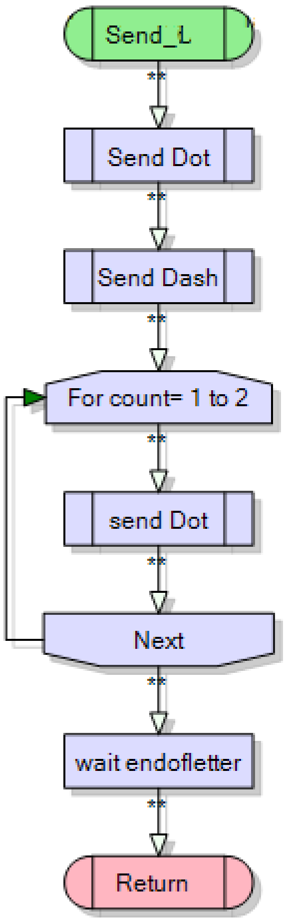


NOTE: the different shape of a gosub

```
'-----------------------------------------------------------------
' Title Block
' Author:        B. Collis
' Date:          May 2008
' File Name:     MorseMeV2.bas
'-----------------------------------------------------------------
' Program Description:
' send morse code using an LED
' show off good use of subroutines and repitition
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat        ' bascom needs to know our micro
$crystal = 1000000      ' bascom needs to know how fast it is going

'-----------------------------------------------------------------
' Hardware Setups
Config Porta = Output                ' make these micro pins outputs
' Hardware Aliases
Morseled Alias Porta.7     ' the name morseled is easy to remember

'-----------------------------------------------------------------
' Declare Variables
Dim Count As Byte       'temporary variable to count repetitions
' Initialise Variables
' Declare Constants
Const Dotdelay = 250                    ' length of a dot
Const Dashdelay = 750                   ' length of a dash
Const Endofworddelay = 1750             ' gap between words

'-----------------------------------------------------------------
'  Program starts here
Do                                      'start of a loop
   Gosub Send_c
   Gosub Send_l
   Gosub Send_s
   Waitms Dashdelay
   'send more letters here
   Wait 5
Loop                           'return to do and start again
End

'-----------------------------------------------------------------
'  Subroutines
Send_c:
   'letter c  - the sequence is dash dot dash dot
   For Count = 1 To 2          'send these twice
      Gosub Dash
      Gosub Dot
   Next
   Waitms Dashdelay            'longer delay between letters
Return

Send_l:
   'letter l  - the sequence is dot dash dot dot
   Gosub Dot                  'a dot
   Gosub Dash                 'a dash
   For Count = 1 To 2         'send 2 dots
      Gosub Dot
   Next
   Waitms Dashdelay            'longer delay between letters
Return
```

```
Send_s:
  'letter s  - the sequence is 3 dots
   For Count = 1 To 3             'send it 3 times
      Gosub Dot
   Next
   Waitms Dashdelay              'longer delay between letters
Return

'--------------------------------------------------------------
Dot:
   Morseled = 1                       ' on
   Waitms Dotdelay                     ' wait 1 dot time
   Morseled = 0                      'off
   Waitms Dotdelay                'short delay between dots & dashes
Return

Dash:
   Morseled = 1                          ' on
   Waitms Dashdelay                    ' wait 1 dash time
   Morseled = 0                     'off
   Waitms Dotdelay               'short delay between dots & dashes
Return
```



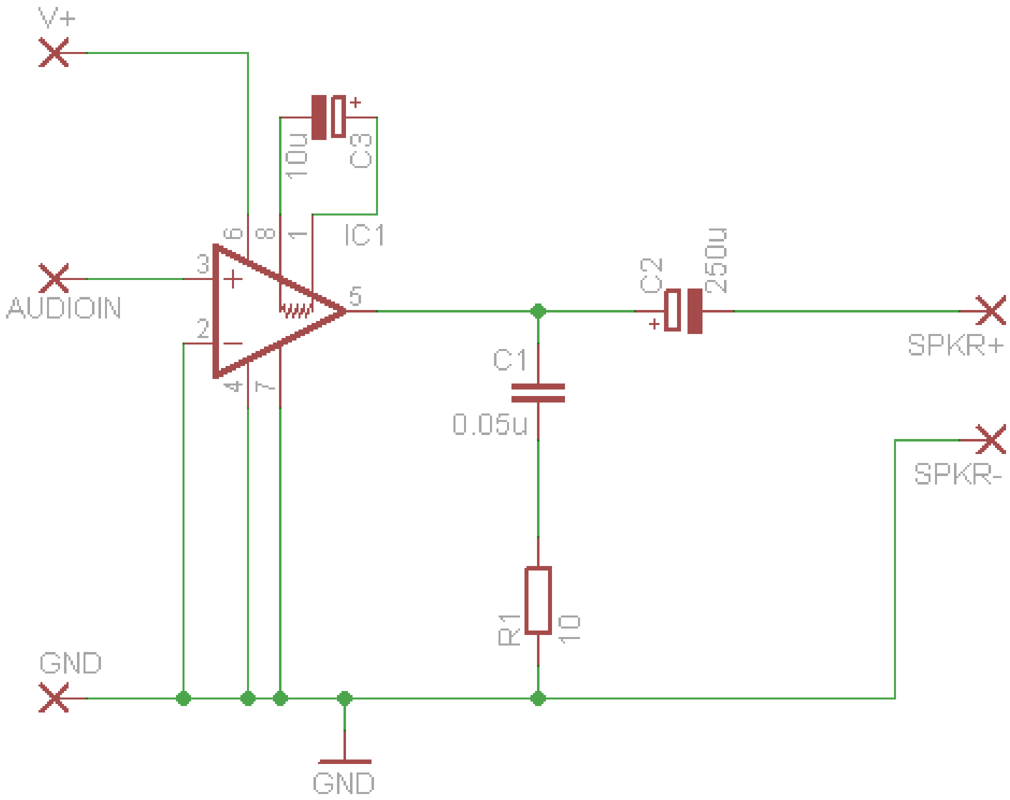


Not only do things like subroutines, comments, indenting code, the use of alias and const make your code easier for you to read and debug, imagine going to a job interview and being asked to bring in some code you had written to show your prospective boss – which would you show him?

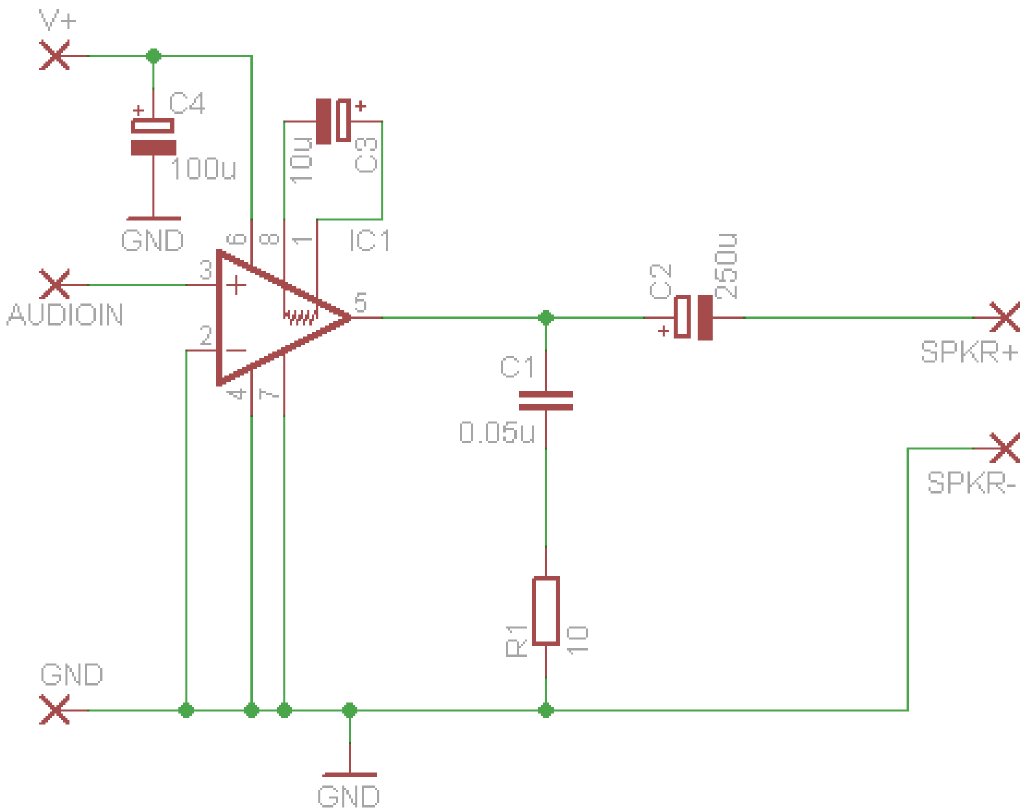
Using const, alias, subroutines and comments properly in programs is an essential code of practice and worth credits to

10.2 LM386 audio amplifier PCB

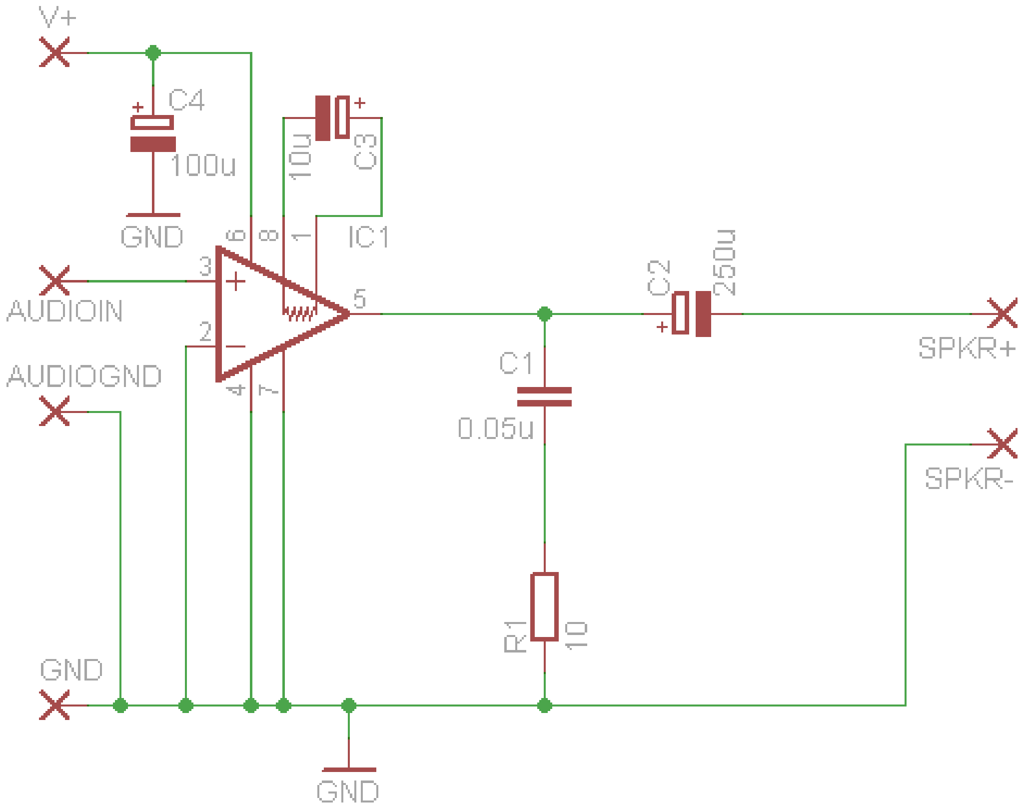
Stage 1 of the LM386 audio amplifier schematic



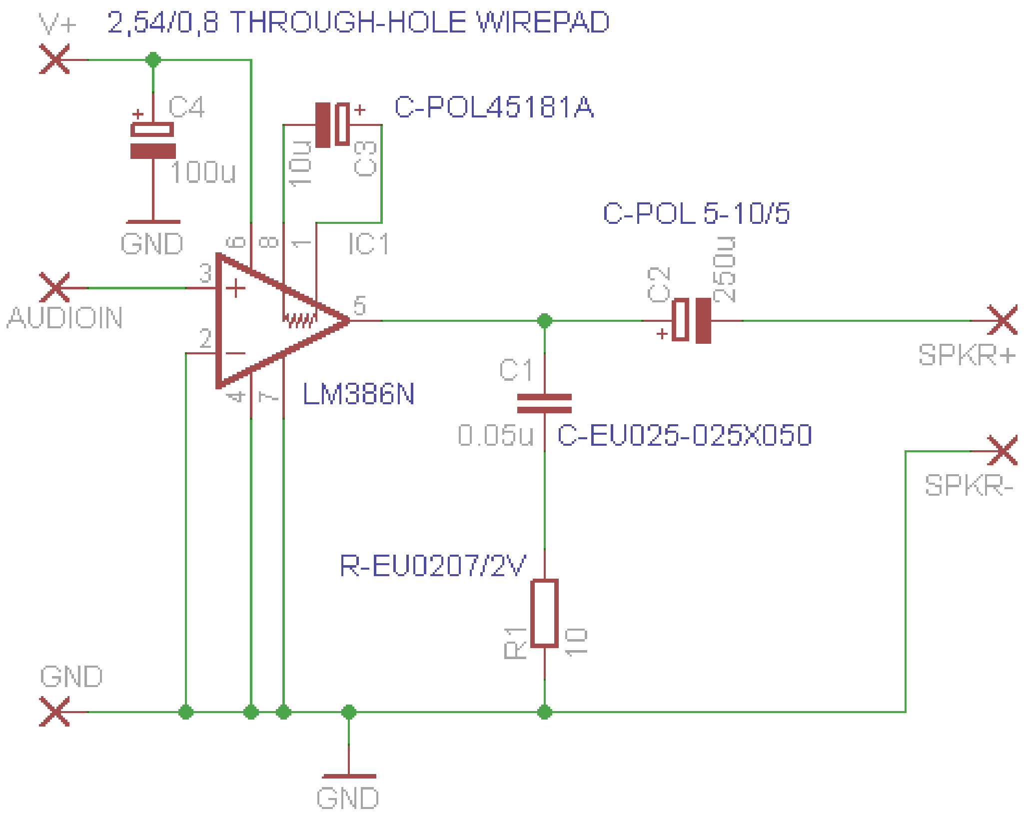
This is more or less from the datasheet, the 10uF capacitor C3 has been added to increase the gain, however a couple of practical changes need to be made ot the schematic before it can be used. First it needs a filter capacitor on the DC supply, otherwise the speaker output will be extremely noisy.



Next the audio input has only a signal conection niot a ground connection and cables that come from the device you want to amplify generally have a signal plus ground wire, so that has been added in the next diagram.



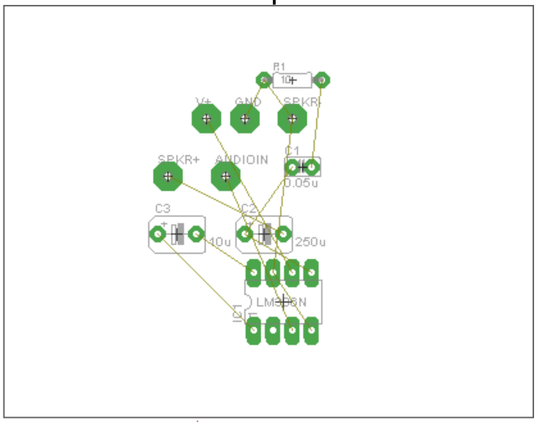
The labels in blue have been added to the schematic are to help you choose which components to use from the CLS library inEagle.It is important to choose the right size components otherwise they may not fit on the PCB when you make it.



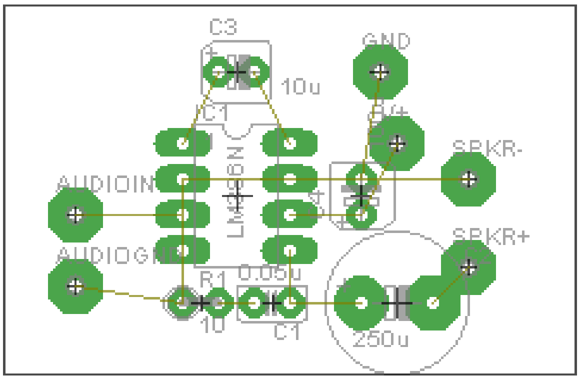
10.3 LM386 PCB Layout

The layout progresses through several stages

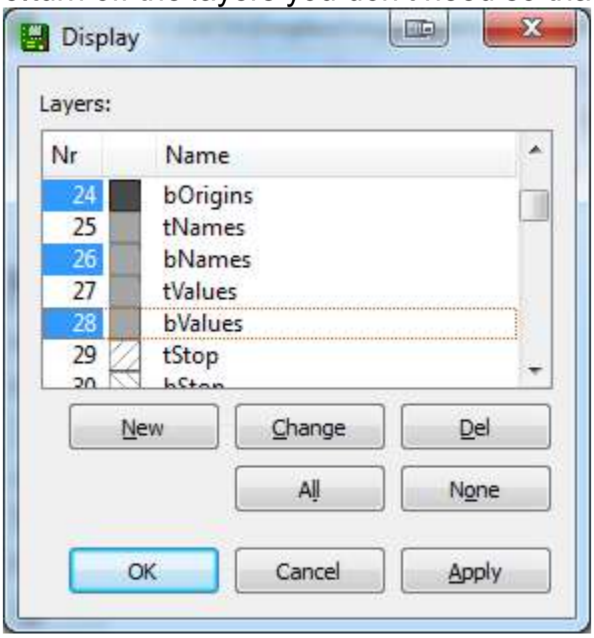
1.move all the components onto the working area of the layout

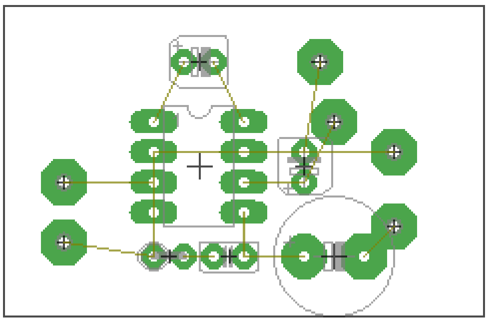


4. try to layout the components with the minimum number of crossing airwires..

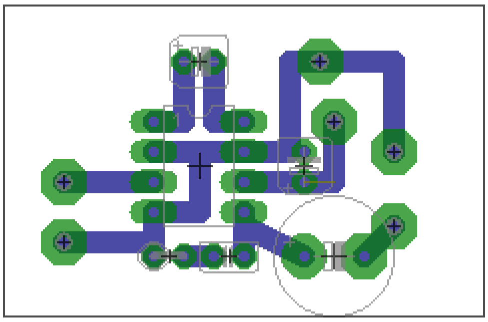


3.turn off the layers you don't need so that you can focus clarly on the layout

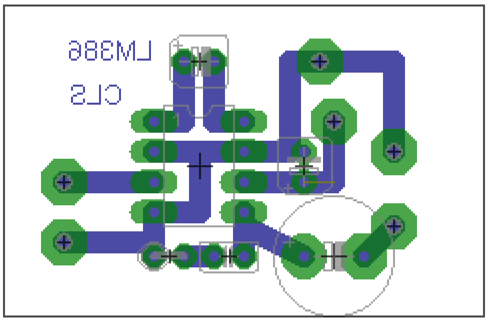




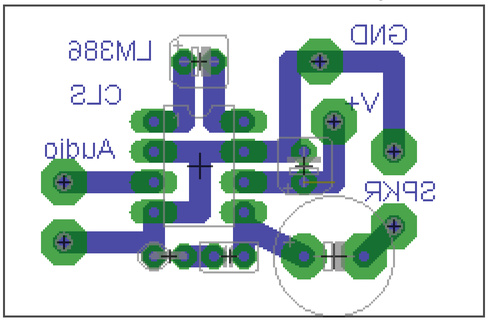
4.layout the tracks



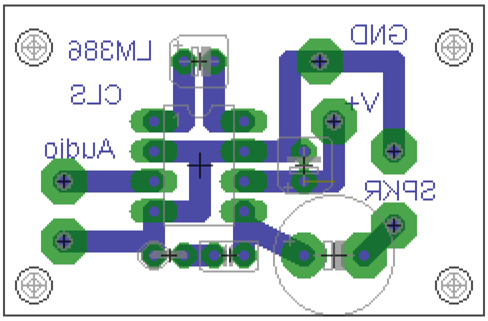
5.Add your name and the board name



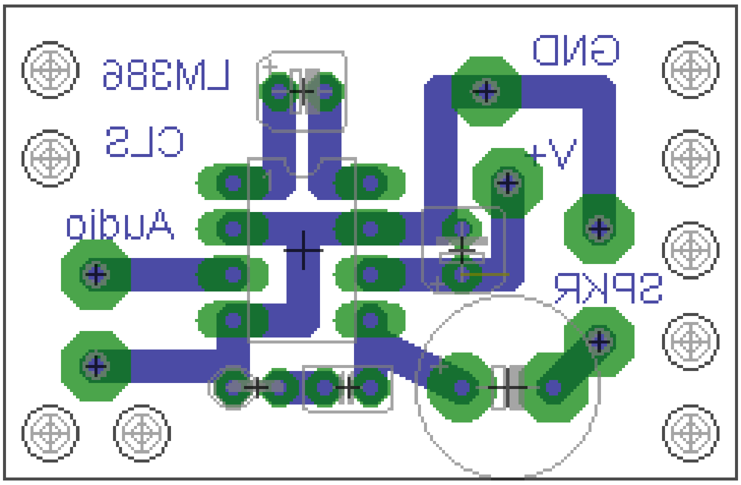
6. Add labels for the wires you will have to connect to the board



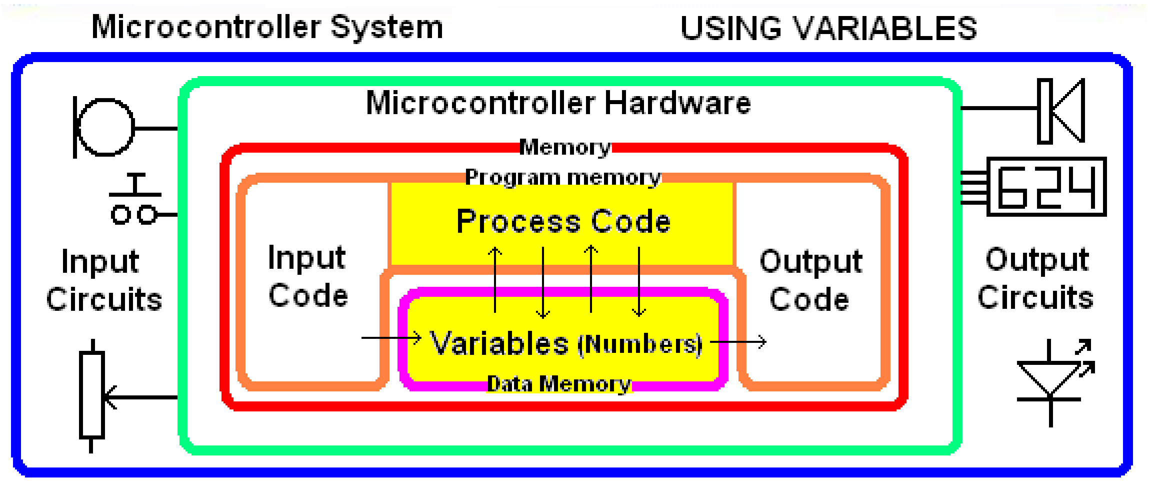
7. Add some mounting holes fr the board, so that it can be attached inside a case.



8. Add some stress relief holes for wires that come on and off the board

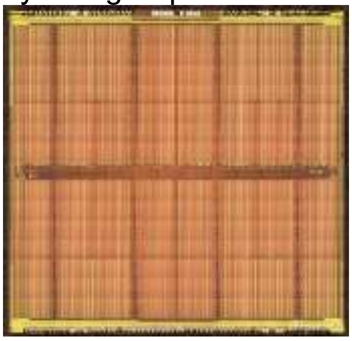


11 Introductory programming – using variables



Inside our brain is our memory, it is where we store and work on information, it is the same in a computer. We often use the different terms **information**, **RAM**, **data, address** or **variable** without really understanding their separate meanings; it is useful to clarfify a meaning for each one.

- **RAM** is the physical place (like our brain cells/synapses). In a computer it is arranged in 'bytes' -groups of 8 individual bits (8 bits = 1 byte)




We can think of it like a series of numbered storage containers or pigeon holes

- **Data** is what is stored in the RAM,
    - data is numbers e.g. 5 or &B00000101 in binary.
- **Address:** this is the physical location of a byte of RAM in the microcontroller (e.g. 0 to 1023).
    - Addresses are sequential.
- **Variable** is the name we give to the place in RAM , it is a useful way to keep track of what we stored there. E.g. height is a variable, it contains the number 6, width is a variable it contains the number 3. These numbers may change a lot while a programis running.
- **Information:** data such as '13' has little meaning to us, it has more meaning if we store it in a variable called weight but it has information when we know that it is the weight of a particular pen in grams.

RAM Address	Variable (name for address)	Data (actual number in the RAM)
1	Orangedelay	10
2	Crossdelay	20
3	Num_flashes	0
4	Flashedelay	500

Programs use, alter and even create data while they are running. This data varies as the program executes so we name it **variables**.

A variable is the unique name we give to a location in the microcontroller's RAM to store data. When data is stored in ram we say we are **storing it in a variable.**

11.1 Stepping or counting using variables

Have you noticed that at a pedestrian crossing that after the Crossnow light goes off the Dontcross light actually flashes before staying on.

In this program we want the dontcross light to flash 10 times while the pedestrian is crossing.

```
Dim Num_flashes As Byte
Dim Orangedelay As Byte
Dim Crossdelay As Byte
Dim Flashdelay As Byte

Orangedelay = 10
Crossdelay = 20
Flashdelay = 500

 'Here is the wrong way to do it
Do
   Do                                      'wait for ped cross button
   Loop Until Crossbutton = 0

   Reset Greenlight                        'stop the traffic
   Set Orangelight
   Wait Orangedelay

   Reset Orangelight
   Set Redlight
   Reset Dontcrosslight                    'allow pedestrian to cross
   Set Crossnowlight
   Wait Crossdelay
   Reset Crossnowlight

   'flash the don't cross light 10 times to tell pedestrians to stop crossing
   Set Dontcrosslight                   'flash1
   Waitms Flashdelay
   Reset Dontcrosslight
   Waitms Flashdelay

   Set Dontcrosslight                   'flash2
   Waitms Flashdelay
   Reset Dontcrosslight
   Waitms Flashdelay

   Set Dontcrosslight                   'flash3
   Waitms Flashdelay
   Reset Dontcrosslight
   Waitms Flashdelay

   Set Dontcrosslight                   'flash4
   Waitms Flashdelay
   Reset Dontcrosslight
   Waitms Flashdelay

'...

   Reset Redlight                          'let traffic continue
   Set Greenlight
Loop
End
```

The above code wastes a lot of our program memory.

'Here is the right way to do it

```
Set Greenlight                          'on
Reset Orangelight                       'off
Reset Redlight                          'off
Set Dontcrosslight                      'on
Reset Crossnowlight                     'off
Do
   Do                                   'wait for ped cross button
   Loop Until Crossbutton = 0

   Reset Greenlight                     'stop the traffic
   Set Orangelight
   Wait Orangedelay

   Reset Orangelight
   Set Redlight
   Reset Dontcrosslight                 'allow pedestrian to cross
   Set Crossnowlight
   Wait Crossdelay
   Reset Crossnowlight

    'flash the don't cross light 10 times -
    For Num_flashes = 1 To 10
        Set Dontcrosslight
        Waitms Flashdelay
        Reset Dontcrosslight
        Waitms Flashdelay
    Next

    Reset Redlight                      'let traffic continue
    Set Greenlight
Loop
```

This is the **for-next loop** in programming – every programming language has it (in some form or another) and we use it **when we want something to repeat or step a fixed number of times**. The variable num_flashes starts at 1 and each time through the loop it increases by 1 until it has completed the loop 10 times.

11.2 For-Next

Repetition is what computers do best here is another example of repetition using a for-next. Example: when you join a gym they give you a workout card which has the exercises and the number of repetitions on it to do.

Bench Max	Max 100 3 sets of 8	Bar Wt 40
Incline Max	Max 80 3 sets of 8	Bar Wt 30
Lat Pull Max	Max 90 3 sets of 8	Bar Wt 40
Leg Exten Max	Max 130 3 sets of 8	Bar Wt 45
Leg Flex Max	Max 120 3 sets of 8	Bar Wt 40
Squat Max	Max 150 3 sets of 8	Bar Wt 40

They don't give you a list:

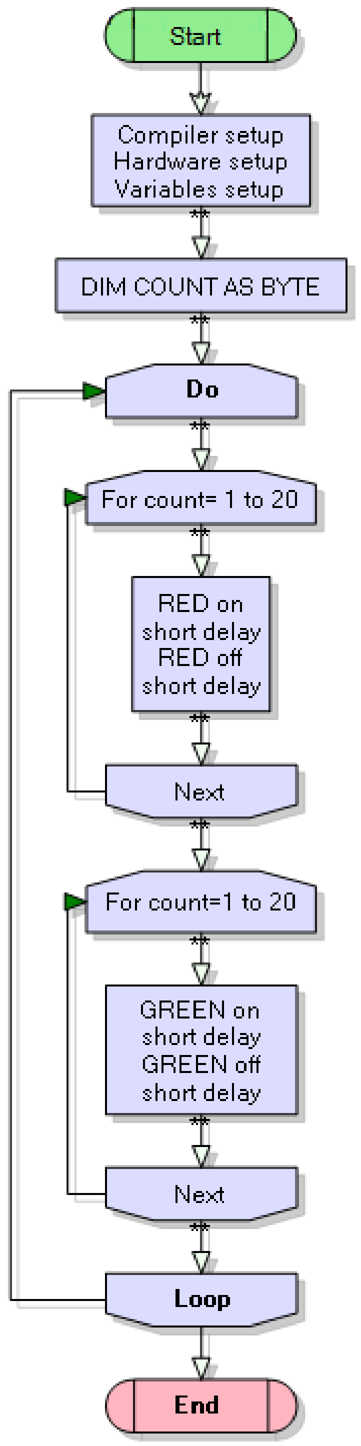
Bench Max
Bench Max
Bench Max
Bench Max
Bench Max
Bench Max
Inline Max
Inline Max
Inline Max
Inline Max
Inline Max
Inline Max
...
...
...

The same with computer programming, when you see something that looks like it is repeating you replace it with a loop of some form (there are several choices).

E.g. at a very busy gym everyone has to be split into one of two groups, those that exercise on the machines and those that work on the mats. Every 60 seconds everyone changes from the mat to the machines. There are two big lights, one red and one green. When the red light is flashing the red group is on the machines, when the green light is flashing the green group is on the machines.

Each light flashes 20 times per minute(on for ½ second off for 2½ seconds). We could write a program the goes:
Red on
Wait ½ sec
Red off
Wait 2½ sec
Red on
Wait ½ sec
Red off
Wait 2½ sec
Red on
Wait ½ sec
Red off
Wait 2½ sec
Red on
Wait ½ sec
Red off
Wait 2½ sec
...
but this is not really computer programming

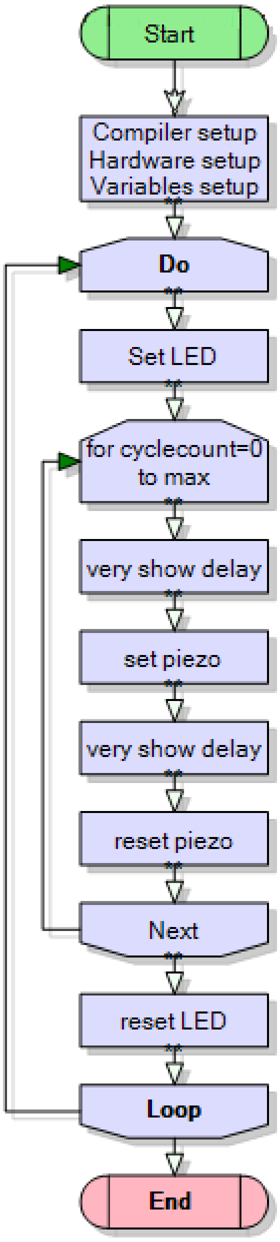
We need a simple way of controlling how many times the lights flash and we can use a variable to count the flashes and a loop that repeats depending upon what number is stored in the variable.

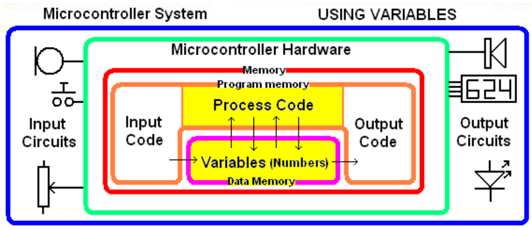



```
'--------------------------------------------------
' Compiler Setup
$regfile = "attiny461.dat"
$crystal = 1000000 'bascom needs to know its speed
'--------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output           'LEDs on portA
Config Portb = Input         'switches on portB
' Hardware Aliases
Green Alias Porta.1
Red Alias Porta.0
'--------------------------------------------------
'Declare Constants
Const Lightontime = 500
Const Lightofftime = 2500
'--------------------------------------------------
'Declare Variables
 Dim Count As Byte
'--------------------------------------------------
Program starts here
Do
      For Count = 1 to 20
            Set Red
            Waitms Lightontime
            Reset Red
            Waitms Lightofftime
            Incr Count
      Next
      For Count = 1 to 20
            Set Green
            Waitms Lightontime
            Reset Green
            Waitms Lightofftime
            Incr Count
      Next
Loop
End
```

11.3 Siren sound - programming using variables

In this program we will use a variable to control the duration (length) of a tone.





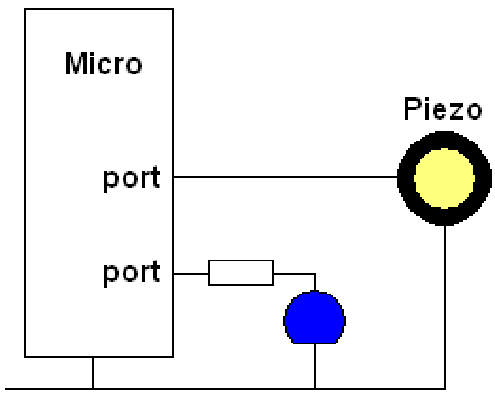
First lets review what a tone is. It is a repeated turning on and off of our piezo. The frequency of the tone is 1/period. The duration of the tone is the number of complete cycles.

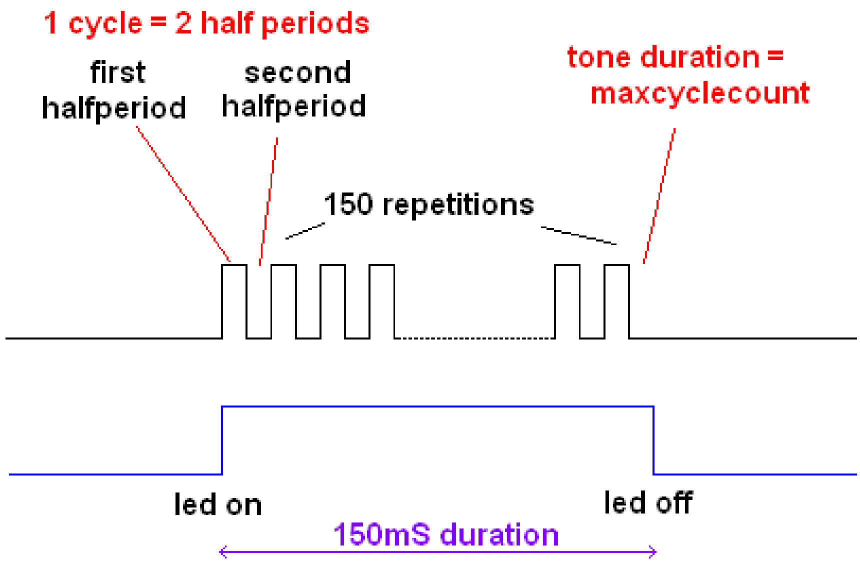
A piezo will not make a sound when you turn it on; it only makes a sound when you turn it on and off rapidly. So to make a tone we must turn the piezo on then wait a bit, then turn it off and we repeat this for the duration of the tone. In this program the tone period will be 1mS so the piezo must be on for 500uS (1/2 mS) and off for the same. Bascom has a waitus command (it is not particularly accurate but its good enough for this exercise). We want the tone to last long enough to hear it so we need to repeat it 150 times. 150 times 1mS will give us a tone duration of 150mS (0.15S).

To count the number of cycles we will dimension a variable called cyclecount, and we will increase it inside a do-loop. It will count upto the max number of cycles and then we will have a 2 second break. Then it will repeat.

Remember to reset cycle count to 0 or it will overflow.

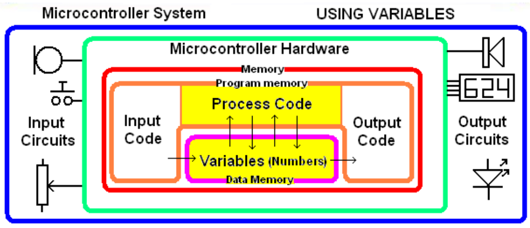
This program works similalrly to the Bascom SOUND command.



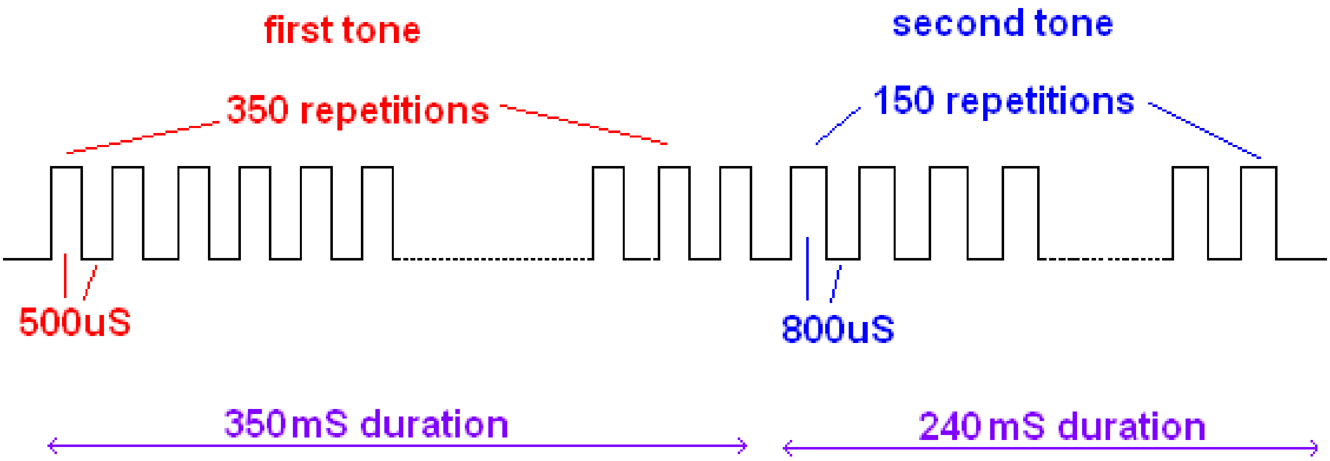


```
'------------------------------------------------------------------
' Title Block
' Author: B.Collis
' Date: 22 Feb 08
' File Name: SirenV1.bas
'------------------------------------------------------------------
' Program Description:
' This program makes a simple tone using a piezo

' Program Features:
'  makes use of Bascom waitus (microseconds) command
' introduces first use of a variable
' the variable cyclecount increases from 0 until it reaches the preset
(constant)
'  value maxcyclecount at which point there is a quiet time
'  the led is on when the the tone is occuring

' Hardware Features
' a pezo can be directly connected to the micro port
' the led has a 1k resistor in series to limit the current
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"                'the micro we are using
$crystal = 1000000                       'rate of executing code
'------------------------------------------------------------------
' Hardware Setups
Config Portb = Output
' Hardware Aliases
Piezo Alias Portb.3             'use useful name PIEZO not PORTb.3
Blueled Alias Portb.4           'use useful name BLUELED not PORTB.4
'------------------------------------------------------------------
'Declare Constants
Const Halfperioddelay = 500              ' delay for 1/2 period
Const Maxcyclecount = 150                'number of cycles to do
'------------------------------------------------------------------
' Declare Variables
Dim Cyclecount As Byte
'------------------------------------------------------------------
' Program starts here
Do
    Set Blueled                          'turn led on
    For Cyclecount = 0 to Maxcyclecount1
        Waitus Halfperioddelay1
        Set Piezo
        Waitus Halfperioddelay1
        Reset Piezo
    Next
    Reset Blueled                        'turn led off
    Waitms 2000                          'quiet time
Loop
End                                      'end program
```

11.4 Make a simple siren



A simple siren sound can be made using two repeating tones A tone of 1 frequency and 1 duration followed by a tone of a different frequency and a different duration. We will use our knowledge of variables to make our own tones.




```
----------------------------------------
' Title Block
' Author: B.Collis
' Date: 22 Feb 08
' File Name: SirenV2.bas
'-------------------------------------------------------------------
' Program Description:
' This program makes a simple siren on a piezo
'
' Program Features:
'  makes use of bascom waitus (microseconds) command
' Hardware Features:
'-------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 1000000                        'the crystal we are using
$regfile = "attiny26.dat"                 'the micro we are using
'-------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Portb = Output
' Hardware Aliases
Piezo Alias Portb.3                   'use useful name PIEZO not PORTb.3
Blueled Alias Portb.4                 'use useful name BLUELED not PORTB.4
'-------------------------------------------------------------------
'Declare Constants
Const Halfperioddelay1 = 500              ' first tone 1/2 period
Const Halfperioddelay2 = 800              ' second tone 1/2 period
Const Maxcyclecount1 = 350                ' longer than 255!!!
Const Maxcyclecount2 = 150
'-------------------------------------------------------------------
' Declare Variables
Dim Cyclecount As Word                'keep count of number of cycls(periods)
Dim Sirens As Byte
'-------------------------------------------------------------------
```

```
' Program starts here
Do
   Set Blueled
   For Sirens = 1 To 3       'just make 3 for testing purposes
      For Cyclecount = 0 to Maxcyclecount1
         Waitus Halfperioddelay1
         Set Piezo
         Waitus Halfperioddelay1
         Reset Piezo
      Next
      For Cyclecount = 0 to Maxcyclecount2
         Waitus Halfperioddelay2
         Set Piezo
         Waitus Halfperioddelay2
         Reset Piezo
      Next
   Next
   Reset Blueled
   Wait 10                             'have a bit of quiet!!!
Loop
End                                    'end program

'------------------------------------------------------------------
```

Point to take note of:

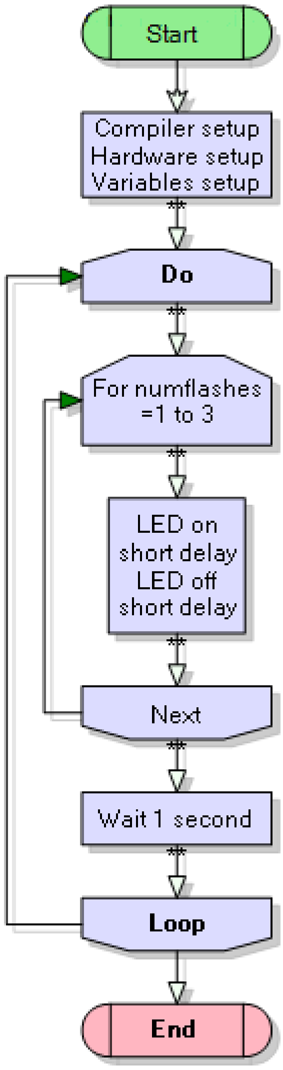
- A single sirensound has been put into a subroutine, this subroutine will last approx 350mS + 240mS = 590mS. Subroutines are a great way of decomplicating your programs.
- How code is indented/tabbed over to aid readabilaity
- If you are using a do-loop – remember to reset your counter
- Use constants rather than putting numbers into your code (waitus halpperioddelay2). It makes it so much easier to read
- Use decent names for variables, constants aand aliases; '*waitus a'* isn't much use when trying to debug a program
- Use pictures/diagrams to help you plan things

11.5 Siren exercise

Modify the delays and count values in this to find a siren you like the most.

11.6 A note about layout of program code

We could create a program that flashes an LED 3 times waits a bit then flashes it again.




```
' ForNextTricks.bas

$crystal = 1000000
$regfile = "attiny461.dat"

Config Porta = Output
Config Portb = Output

Led7 Alias Porta.7

Dim Num_flashes As Byte

Const Ondelay = 50
Const Offdelay = 200
Const Longdelay = 1000

Do
   For Num_flashes = 1 To 3
      Set Led7
      Waitms Ondelay
      Reset Led7
      Waitms Offdelay
   Next
   Waitms Longdelay
Loop
```

Indenting (tabbing code, is an extremely important aspect of writing programs, it adds to their readability and your ease of debugging.

I often fix students code simply by setting up the indenting and find things like this

<table>
<tr><th>HARD TO SPOT THE ERROR</th><th>EASY TO SPOT THE ERROR</th></tr>
<tr><td><pre>Do
For Num_flashes = 1 To 10
Set Dontcrosslight
Waitms Flashdelay
Reset Dontcrosslight
Waitms Flashdelay
Loop
Next</pre></td><td><pre>Do
   For Num_flashes = 1 To 10
      Set Dontcrosslight
      Waitms Flashdelay
      Reset Dontcrosslight
      Waitms Flashdelay
Loop
Next</pre></td></tr>
</table>

When a block of code is inside a control structure of some kind the inside code is indented and the end of control structure lines up with the beginning of it. In this case it can now be seen that the For has no closing Next as the Next is outside the Do-Loop. OOPS – need to move the Next above the Loop and then indent it so that it lines up with the For

11.7 Using variables for data

We have seen ho w a variable can be used to create a stepping pattern in program code now we see how numbers can store information. In a calculator with several memory locations each is given a name such as A,B,C,D,E,F,X,Y.M. etc. The name of the memory location has nothing to do with what you are using it for and it is up to you to remember what was stored in each location. In a microcontroller each memory location is given a name by the programmer. This means it is much easier for you to remember what is in the memory location and easier to use within your program.

Here are some examples of using variables

Dim Width as Byte DIM is short for dimension and means set aside a part of RAM for our program to use. From now on in the program it will be called Width. It is easier for us to have names for memory locations such as ‘width’ than using the physical address of the RAM, address 1.

Dim Height as Byte
Dim V_Position as Byte
Dim Speed as Byte
Dim X_position as Byte
Dim Color as Byte
Dim Mass as Byte

Here are some common things you will see in programs
Height = 10 (put 10 into the memory location we dimensioned called height)

Incr X_position (increase the value in X_position by 1)

Color = Width / Height (divide the number in Width with Height and put the answer into Color - the values of Width and Height do not change)

Speed = Speed + 12 (get the number from memory location called Speed and increase it by 12 and put it back into the same memory location)

A variable of type Byte can store numbers from 0 to 255 (&B11111111) so it has limited use so often we group bytes together to store bigger numbers.

11.8 Different types of variables

Bigger numbers require more RAM than smaller numbers, also different kinds of numbers require different amounts of RAM (e.g. negative, decimals). Microcontrollers have limited RAM, so to make the best use of RAM we use the best variable type we can. If we dimension a variable as a type that can store huge numbers and only every use numbers up to 10 then we are wasting a precious resource.
Using the Bascom-AVR help file research the following information on the different types of variable you can use.

Variable type	**Minimum Value (before underflow)**	**Maximum Value (before overflow)**	**Number of bytes used to store it**
Bit	0	1	1byte for 1 bit however if you dimension 8 bits they will all be stored in the same byte
Byte	0	255	1
Word			
Integer			
Long			
Single			
Double			

Every microcontroller has different amount of RAM available for storing variables
Carry out research on these different AVR microcontrollers

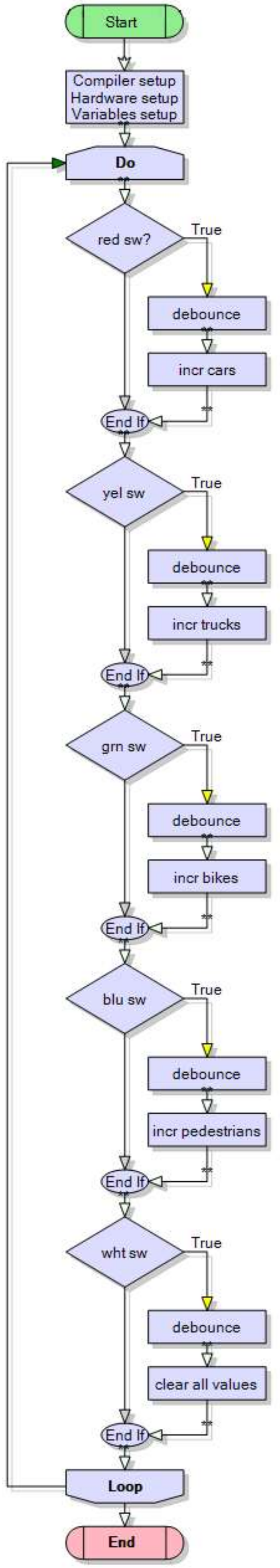
	RAM size (bytes)	FLASH - program size(bytes)	EEPROM size (bytes)
ATTiny13			
ATTIny45			
ATTIny461			
ATMega48			
ATMega16			
ATMega32			
ATMega644			
ATMega1284			

There is no point in memorizing this data; its just a matter of knowing about so that you can find it when you need it.

11.9 Variables and their uses

' ShowComandsV1.bas $sim $crystal = 1000000 $regfile = "attiny26.dat" Config Porta = Output Config Portb = Output Config Pinb.6 = Input	.
'dimension variables Dim Byte1 As Byte Dim Byte2 As Byte Dim Word1 As Word Dim Int1 As Integer Dim Single1 As Single Dim Single2 As Single	Allocating some parts of the RAM, and giving those parts names so that we can refer to it more easily (dimensioning).
Byte1 = 12 Byte1 = Byte1 + 3 Incr Byte1 Byte2 = byte1	What is the value of the variable byte1 after this?
Byte2 = Byte1 / 10	Division - a byte can only represent whole numbers from 0 to 255 so division truncates (IT DOES NOT ROUND) 16/10 = 1 (whole numbers only!)
Byte2 = Byte1 Mod 10	MOD gives you the remainder of a division (16 mod 10 = 6)
Byte2 = Byte1 * 150	This gives the wrong answer because a byte can only hold a number as big as 255
Word1 = Byte1 * 150	This gives the right answer!
Int1 = 200 Int1 = Int1 – 100 Int1 = Int1 – 100 Int1 = Int1 – 100 Int1 = Int1 – 100	need negative numbers then use integer or long
For Single1 = 0 To 90 Step 5 Single2 = SQR(single1) Next	need DECIMALS use single or double
End	Make sure you put an END to your program or it will continue on and potentially cause crashes (if you micro was controlling a car then it might be a car crash- ouch!!)

11.10 Vehicle counter



This program counts different vehicle types everytime a different switch is pressed, if we run it inthe simulator we can see the numbers in decimal, binary and hexadecimal

```
'VehicleCounterV2.bas

'test our ability to count vehicles,
'as used by someone standing at an intersection monitoring traffic
flows

$crystal = 1000000
$regfile = "attiny461.dat"

Config Porta = Output
Config Portb = Input

Cars_sw Alias Pinb.0                       'RED switch
Trucks_sw Alias Pinb.1                     'YELLOW switch
Bikes_sw Alias Pinb.2                      'GREEN Sswitch
Peds_sw Alias Pinb.3                       'BLUE switch
Clear_sw Alias Pinb.4                      'WHITE switch


'dimension variables before they are used or you will get a
compiler error!
Dim Cars As Byte
Dim Trucks As Byte
Dim Bikes As Byte
Dim Peds As Byte

Const Debouncedelay = 25

Do
    Debounce Cars_sw , 0 , Cars_sw_pressed , Sub    'red switch
    Debounce Trucks_sw , 0 , Trucks_sw_pressed , Sub 'yellow switch
    Debounce Bikes_sw , 0 , Bikes_sw_pressed , Sub   'green switch
    Debounce Peds_sw , 0 , Peds_sw_pressed , Sub     'blue switch
    Debounce Clear_sw , 0 , Clear_sw_pressed , Sub   'white switch
Loop
End


'Subroutines
Cars_sw_pressed:
   Incr Cars
Return

Trucks_sw_pressed:
   Incr Trucks
Return

Bikes_sw_pressed:
   Incr Bikes
Return

Peds_sw_pressed:
   Incr Peds
Return

Clear_sw_pressed:
   Cars = 0
   Trucks = 0
   Bikes = 0
   Peds = 0
Return
```

Note how in this program the names of the switches relate to their function, and the comments tell you which switch is which colour. These are very good programming practice

11.11 Rules about variables

Variabes must start with a letter not a digit

e.g. **Dim** Red_cars **As Byte** not **Dim** 1cars **As Byte**

Variabes must not be Bascom reserved(special) words

e.g. **Dim** band **As Byte** not **Dim And As Byte**

Variables must contain no spaces

e.g. **Dim** Red_cars **As Byte** not **Dim** Red cars **As Byte**

Variable names should relate to what the variable is used for

e.g. **Dim** Red_cars **As Byte**, not **Dim** hgashg **As Byte**

Variable names cannot be used for other things such as constants or subroutines

e.g. **Dim** Red_cars **As Byte**, means yu cannot have **Const** Red_cars = 12 as well

11.12 Examples of variables in use

A points table for a competition

```
Dim Blues As Byte
Dim Hurricanes As Byte
Dim Waratahs As Byte
```

as the season progresses the points are added.

```
Incr Hurricanes   (adds one to their score)
Blues = Blues + 1   (adds one to their score)
Waratahs = Waratahs + 3
```

Conversions between units

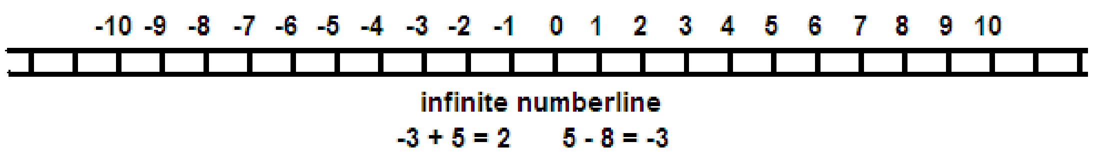
```
Dim Celcius As Integer
Dim Fahrenheit As Integer

Fahrenheit = 100

Celcius = 32 - Fahrenheit
Celcius = Celcius * 5
Celcius = Celcius / 9
```

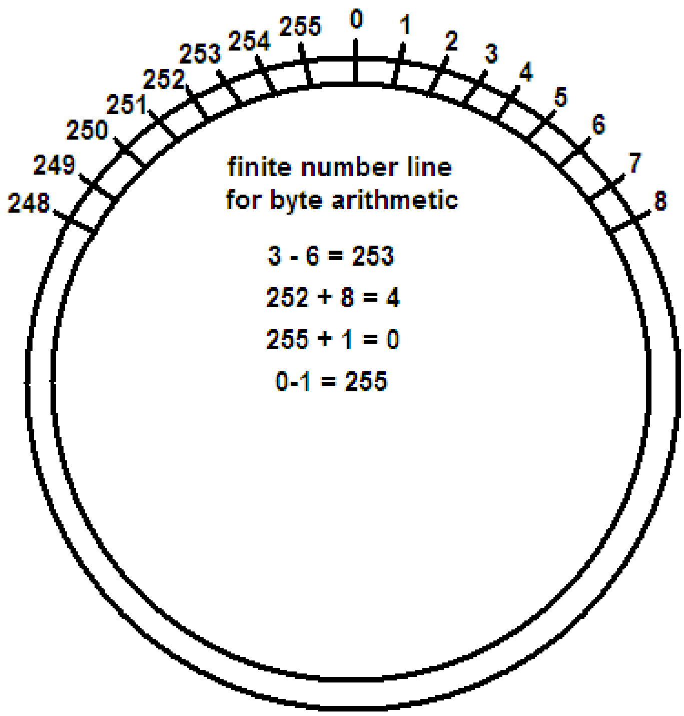
11.13 Byte variable limitations

RAM (the memory inside a computer) is capable of storing 1 byte (or 8 bits) of binary data. This is a **finite range of positive, whole numbers from 0 to 255**. No negative numbers can be stored, no decimal fractions, and no number greater than 255.

Binary Number	Decimal equivalent
00000000	0
00000001	1
00000010	2
--- --- ---	
11111101	243
11111110	254
11111111	255

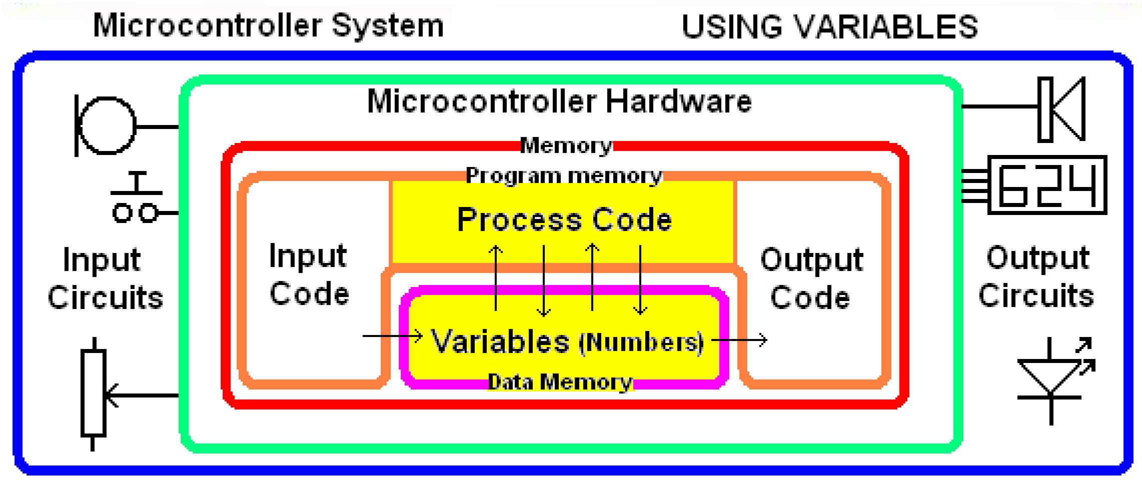
We can see the diference by comparing counting in byte math to counting in the decimal system. In the decimal system the numbers we are used go from –infinity to +infinity, so the numberline goes on forever.
Byte arithmetic because it has a finite set of numbers is like having a number line that goes around on itself.





The difficulty arises when we do arithmetic that exceeds the limits of our range.
e.g. what does 250 + 9 = ? What does 4-7 = ?
When we add 9 to 250 we get 3. It has overflowed 255.
The opposite to **OVERFLOW** is **UNDERFLOW** and is seen by using the circular number line above.

11.14 Random Numbers



This program generates a random number from 1 to 6 and stores it into a variable in memory

```
‘ DiceV1.bas
$sim
$crystal = 1000000
$regfile = "attiny26.dat"
Config Porta = Output
Config Portb = Input

Dim dicethrow As Byte

Do
      'generate a random number from 0 to 5
      dicethrow = Rnd(6)
       'change the range to 1 to 6
       dicethrow = dicethrow + 1
Loop
End
```

The line Dim dicethrow As Byte means allocate to the program 1 byte of ram to use and refer to it as dicethrow.

Every variable must be dimensioned before it can be used.

With variables you can do maths
E.g. add 1 to throw. dicethrow=dicethrow+1 literally means get the contents of dicethrow add 1 to it, and then put the answer back into dicethrow.

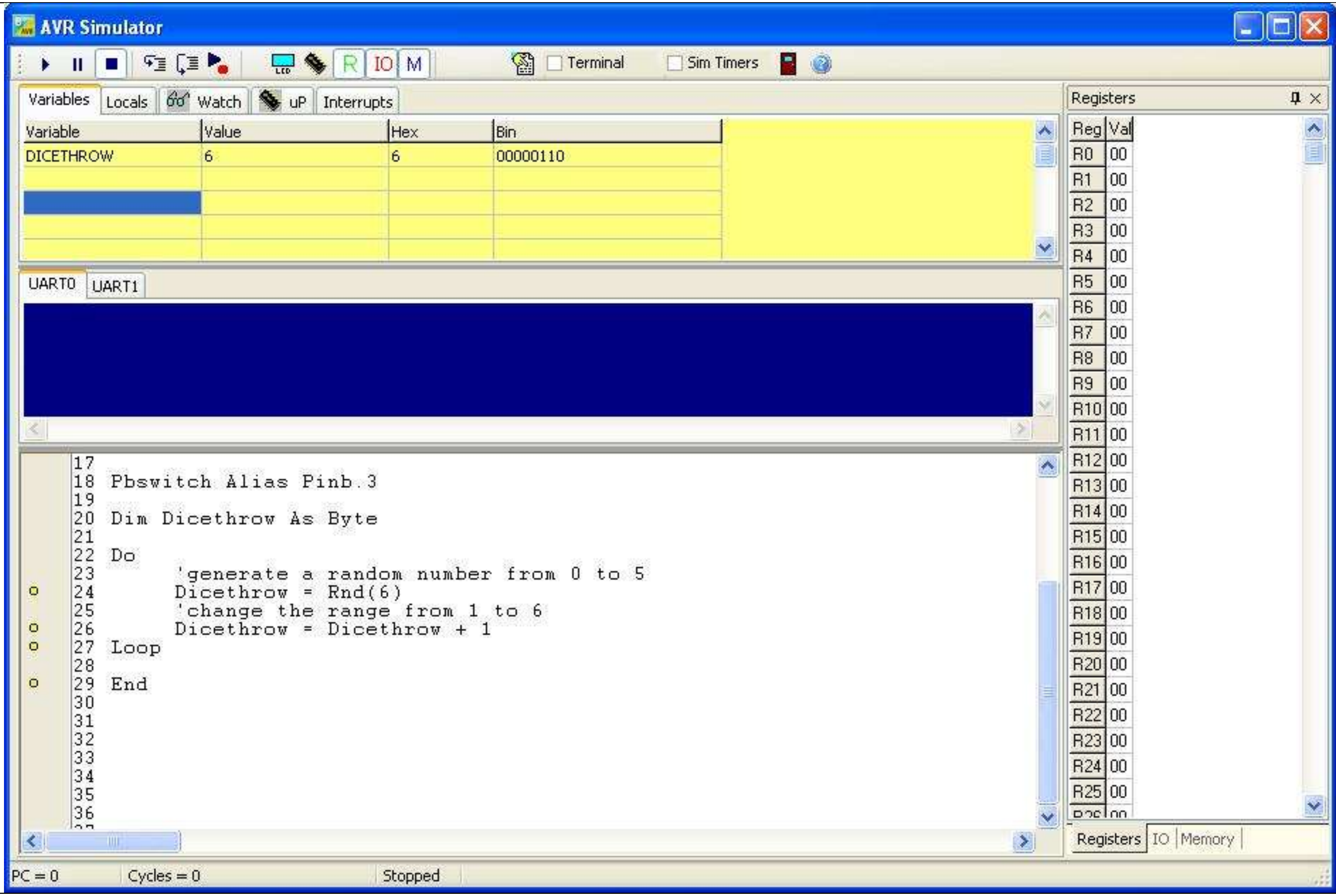
Compile the program and then open the simulator (F2), select the variable dicethrow from the variables list and use F8 (don’t press run) to step through the program to see the numbers generated by the program

11.15 The Bascom-AVR simulator

Press F2 to pen the simulator

Double click in the yellow area under the word VARIABLE to select the variables you want to watch.

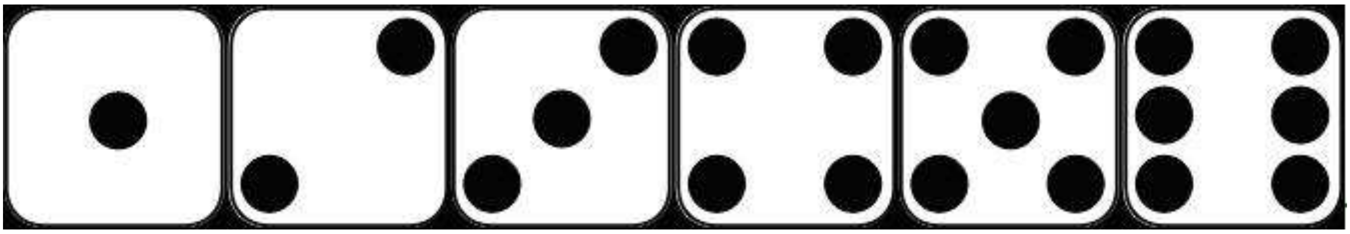
Press F8 to step through the program and see what happens to the value of the variable at each step.

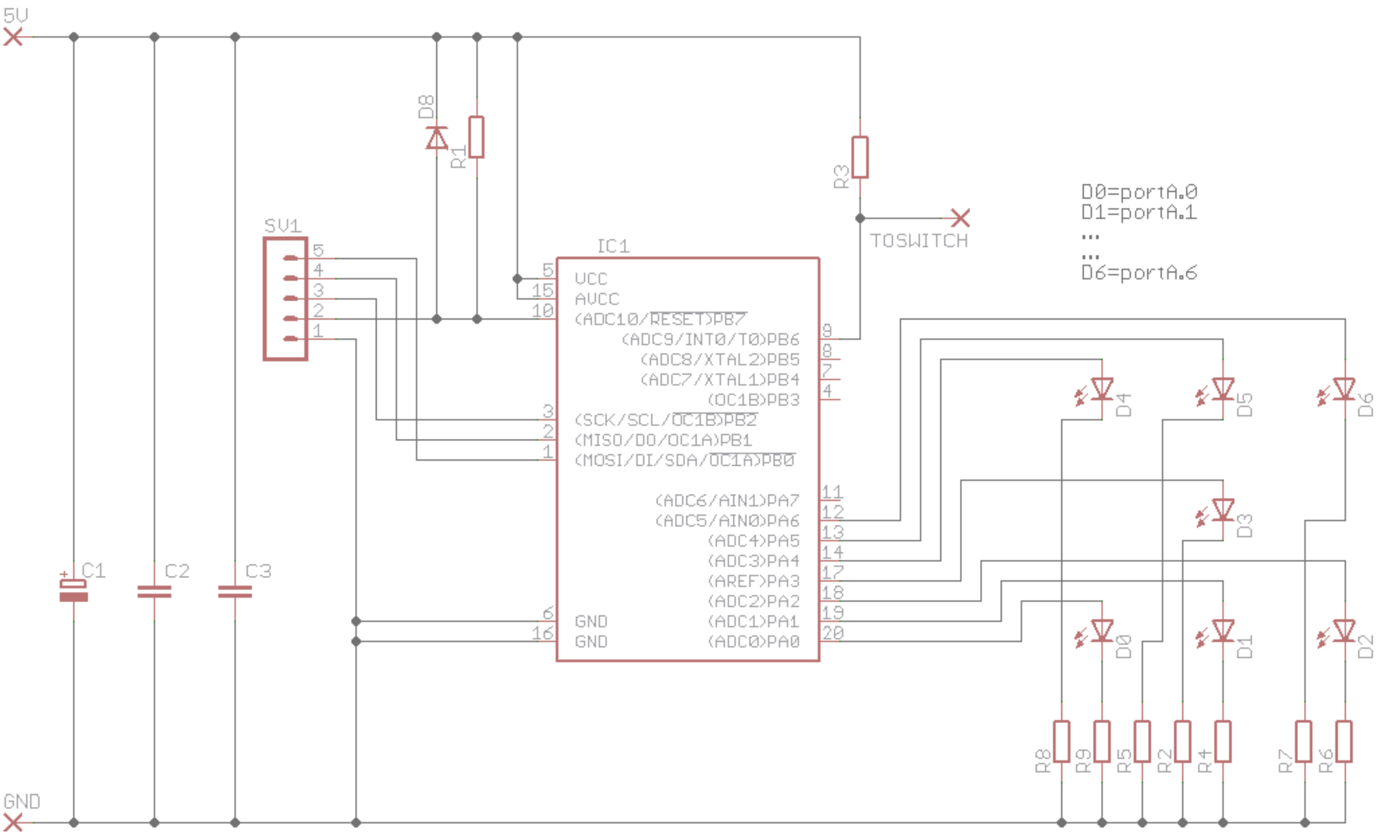


11.16 Electronic dice project

11.17 Programming using variables – dice

A dice can be made using 7 LEDs (why do we need 7? – look closely at the patterns here)





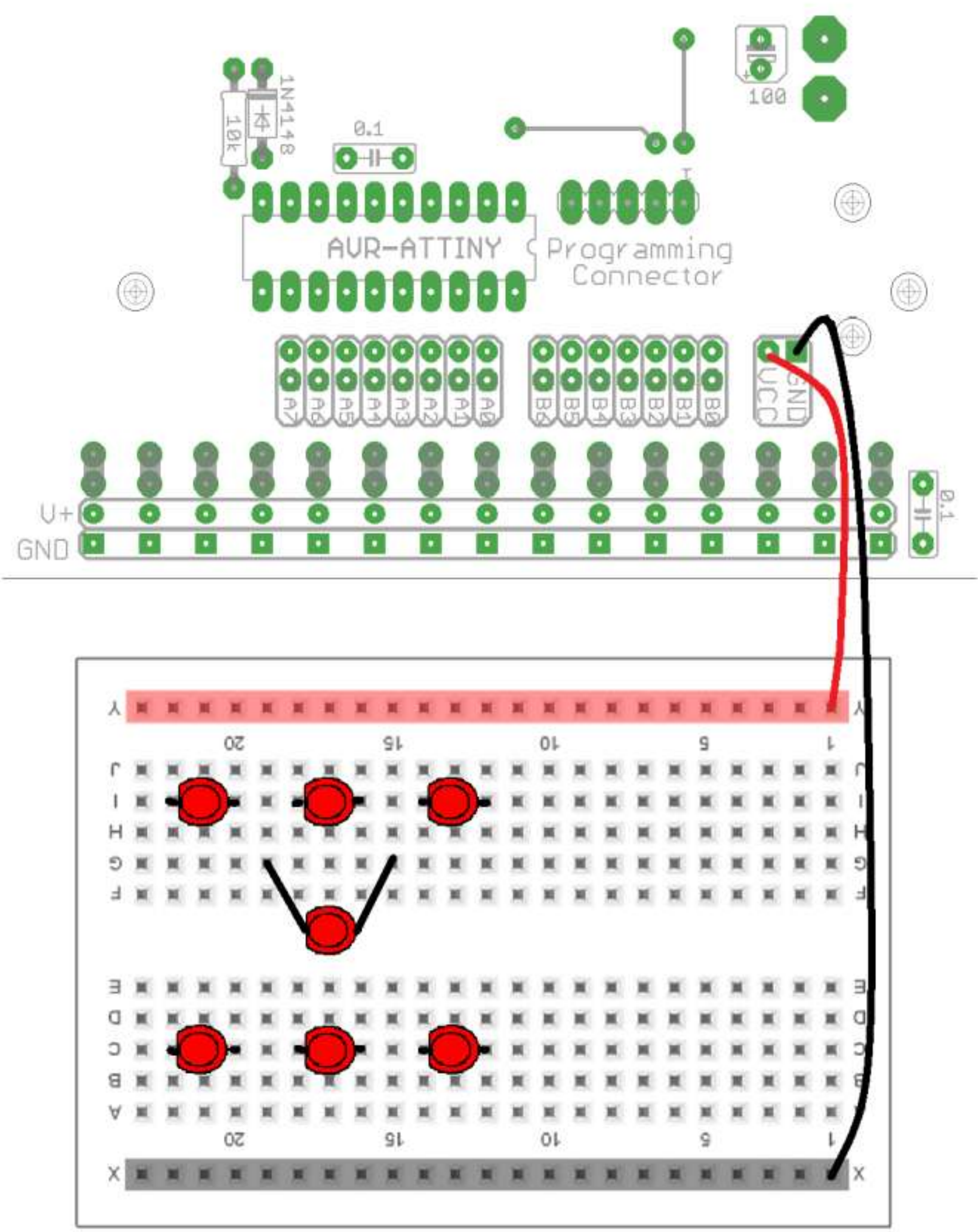
In the above circuit the LEDs have been labelled to match the pin of porta they are connected to. Note there is a switch connected to Pinb.6

Fill in the table below which shows which LED are on and whichare off to make a particular pattern, remember that even though only 7 LEDs are used we need to control the whole port so need to specify all 8 bits.

	A.7	A.6	A.5	A.4	A.3	A.2	A.1	A.0	
	NO LED	LED 6	LED 5	LED4	LED3	LED2	LED1	LED0	
1									
2									
3									
4	off	**on**	off	**on**	off	**on**	off	**on**	portA=&B01010101
5									
6									

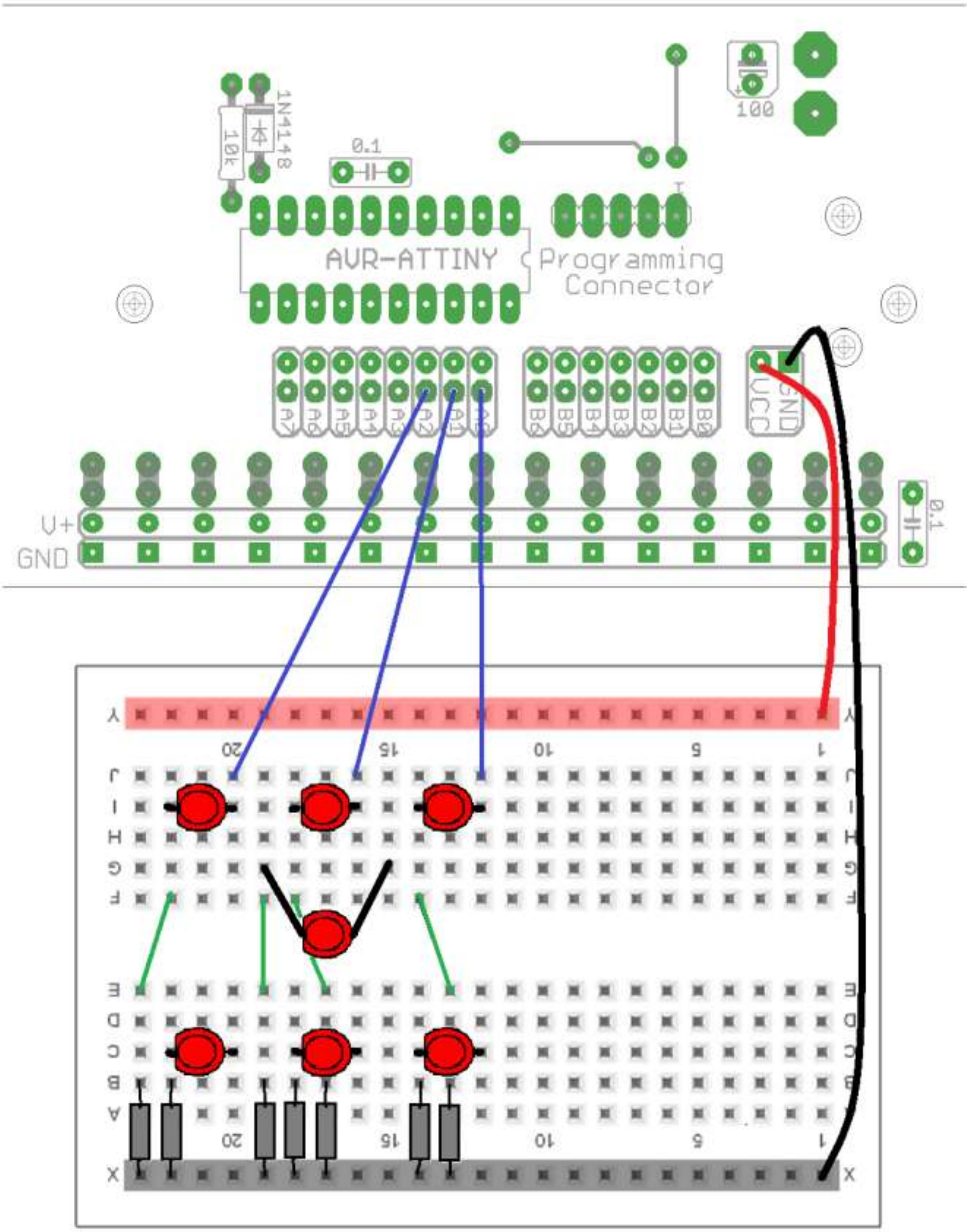
11.18 Dice layout stage 1

In the diagram the 7 LEDs have been physically arranged to match the dots on the face of a dice, but to do that the middle LED has had its legs bent so that it lines up with the middle LED but does not share any breadboard connections with it



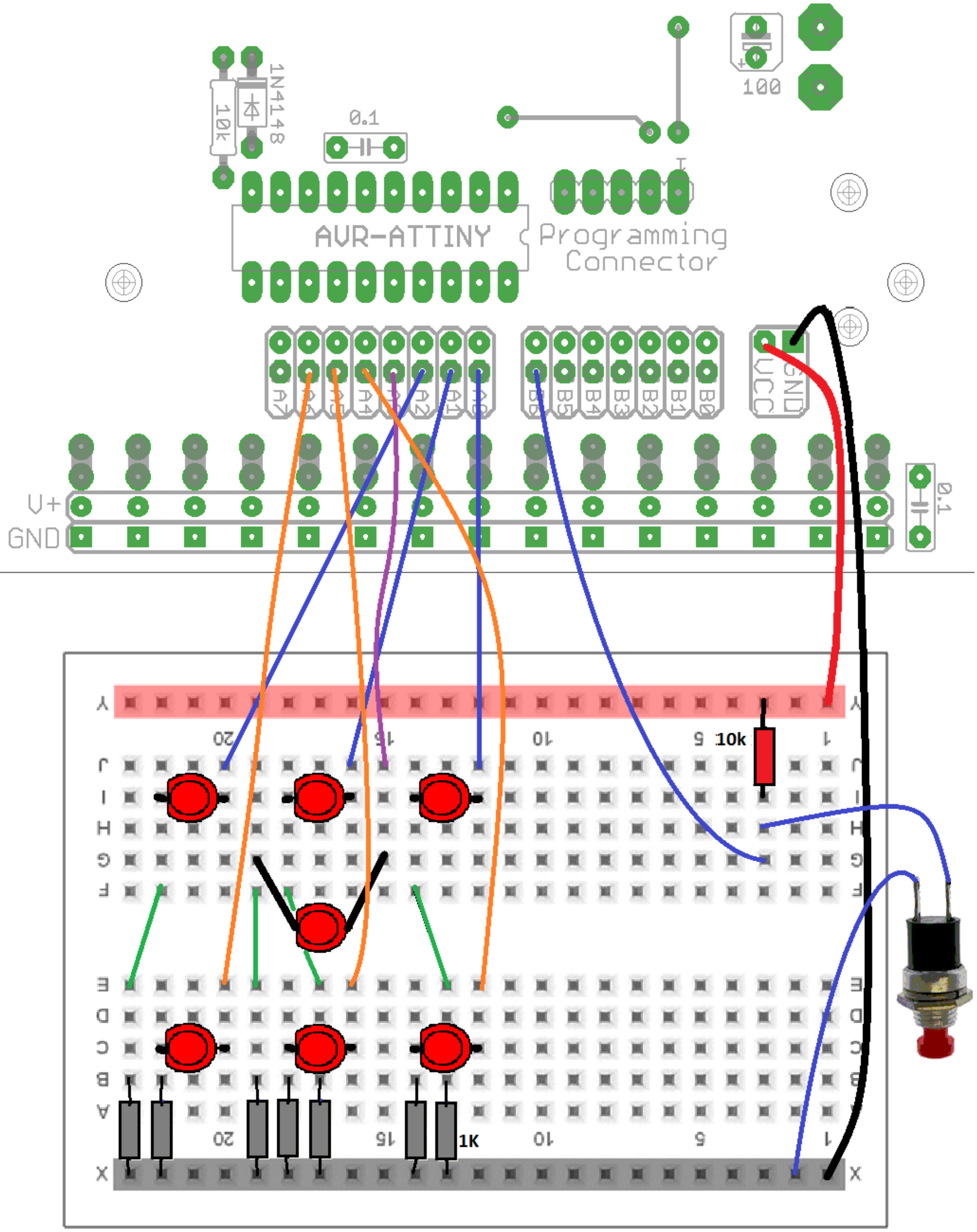
11.19 Dice layout stage 2

In this second stage the resistors have been added and the wiring has been started for the LEDs,

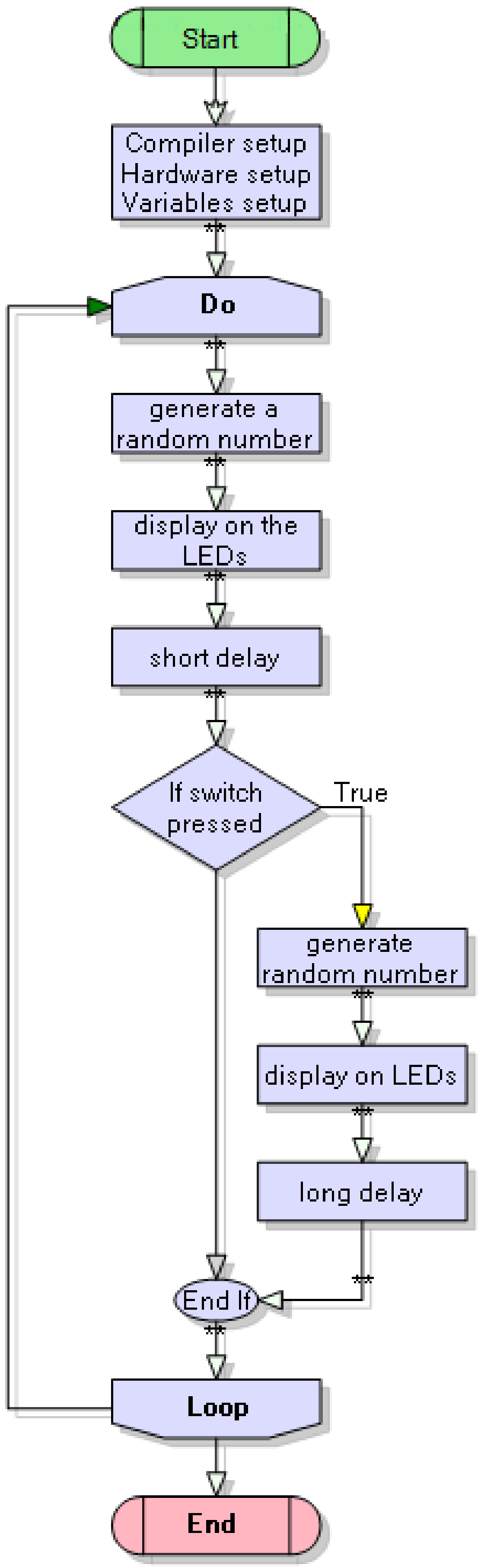


11.20 Dice Layout final

Before the rest of the wiring for the LEDs has been added the switch has been connected, agin note that it is switch wiring that confuses students the most.



11.21 First Dice Program flowchart



```
' DiceV1-random.bas
' 7 leds arranged in a pattern on a breadboard
$crystal = 1000000
$regfile = "attiny461.dat"

Config Porta = Output
Config Pinb.6 = Input
Blu_sw Alias Pinb.6

Dim Dicethrow As Byte                    'a variable to hold the value

Const Dicedisplay = 80
Const Displaytime = 3                    'waiting time in seconds

Do
    Dicethrow = Rnd(6)                   'get a random num from 0 to 5
    Incr Dicethrow                       'make it from 1 to 6
    If Dicethrow = 1 Then Porta = &B0.......      'turns on 1 led
    If Dicethrow = 2 Then Porta = &B0.......      'turns on 2 leds
    If Dicethrow = 3 Then Porta = &B0.......      'turns on 3 leds
    If Dicethrow = 4 Then Porta = &B01010101      'turns on 4 leds
    If Dicethrow = 5 Then Porta = &B0.......      'turns on 5 leds
    If Dicethrow = 6 Then Porta = &B0.......      'turns on 6 leds
    Waitms Dicedisplay                   'wait a little
    If Blu_sw = 0 Then                   'if switch is pressed
        Dicethrow = Rnd(6)               'get a random num from 0 to 5
        Incr Dicethrow                   'make it from 1 to 6
        If Dicethrow = 1 Then Porta = &B0.......      'turns on 1 led
        If Dicethrow = 2 Then Porta = &B0.......      'turns on 2 leds
        If Dicethrow = 3 Then Porta = &B0.......      'turns on 3 leds
        If Dicethrow = 4 Then Porta = &B01010101      'turns on 4 leds
        If Dicethrow = 5 Then Porta = &B0.......      'turns on 5 leds
        If Dicethrow = 6 Then Porta = &B0.......      'turns on 6 leds
        Wait Displaytime
    End If
Loop

End
```

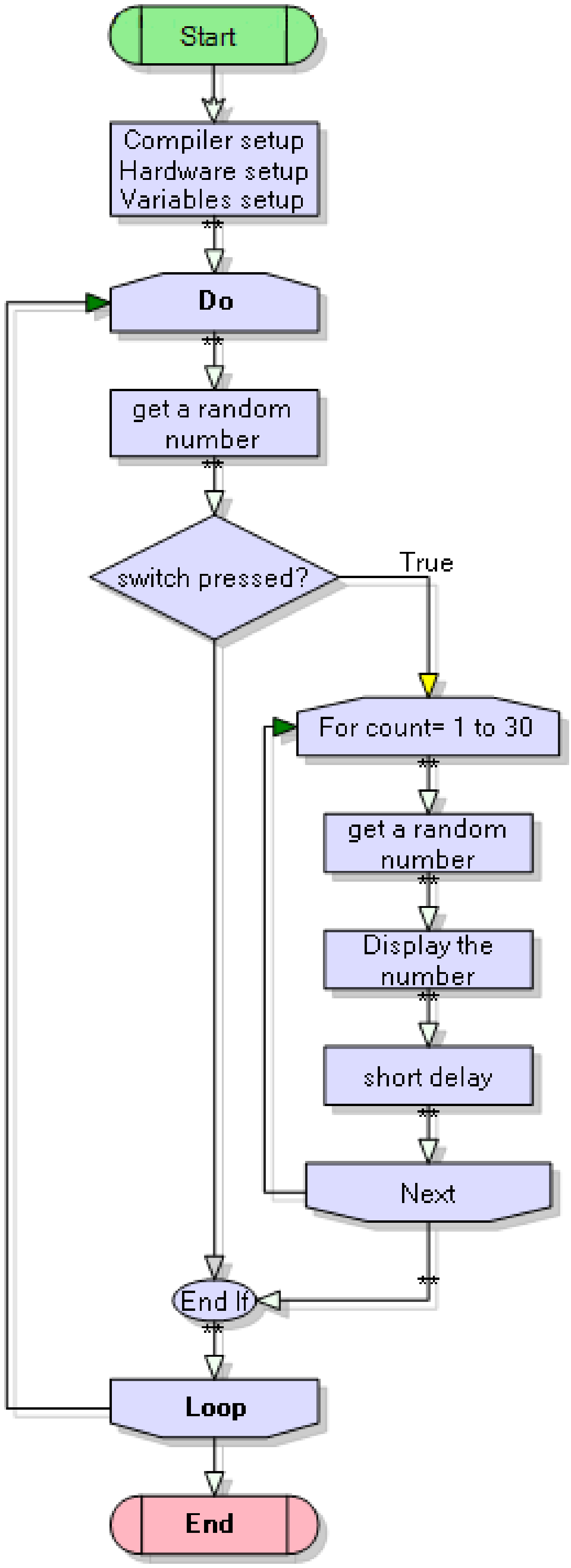
In this case we don't need any debounce timing because there is a long delay after the switch is pressed.

11.22 A note about the Bascom Rnd command

It is actually quite difficult to generate random numbers; microcontrollers use a maths equation to do it. The problem with this is that the sequence is always the same, you can check this out using the simulator or by modifying your dice program later to see that the sequence is always the same. To get around this problems we use a little trick; we always have the program generating random numbers even when the button isn't pressed, that way the position in the sequence when we press the button cannot be guessed.

11.23 Modified dice

In this dice the number stays on the screen and when the switch is pressed it displays 30 random numbers before stopping on the 30th



```
' DiceV2-random.bas
' 7 leds arranged in a pattern on a breadboard
$crystal = 1000000
$regfile = "attiny461.dat"

Config Porta = Output
Config Pinb.6 = Input

Set Portb.6
Blu_sw Alias Pinb.6

Dim Dicethrow As Byte                    'a variable to hold the value
Dim I As Byte
Const Dicedisplay = 100

Dicethrow = 1                            'initial display is 1
If Dicethrow = 1 Then Porta = &B0               'turns on 1 led
If Dicethrow = 2 Then Porta = &B0               'turns on 2 leds
If Dicethrow = 3 Then Porta = &B0               'turns on 3 leds
If Dicethrow = 4 Then Porta = &B01010101        'turns on 4 leds
If Dicethrow = 5 Then Porta = &B0               'turns on 5 leds
If Dicethrow = 6 Then Porta = &B0               'turns on 6 leds

Do
    Dicethrow = Rnd(6)                   'get a random num from 0 to 5
    If Blu_sw = 0 Then                   'if switch is pressed
        For I = 1 To 30                  'do 30 random numbers
            Dicethrow = Rnd(6)           'get a random num from 0 to 5
            Incr Dicethrow               'make it from 1 to 6
            If Dicethrow = 1 Then Porta = &B0           'turns on 1 led
            If Dicethrow = 2 Then Porta = &B0           'turns on 2 leds
            If Dicethrow = 3 Then Porta = &B0           'turns on 3 leds
            If Dicethrow = 4 Then Porta = &B01010101    'turns on 4 leds
            If Dicethrow = 5 Then Porta = &B0           'turns on 5 leds
            If Dicethrow = 6 Then Porta = &B0           'turns on 6 leds
            Waitms Dicedisplay           'wait here a while
        Next
    End If
Loop
End
```

Exercises for the dice program

1. Do a trial of at least 200 presses and draw a tally of the results, how ‘fair’ is our dice?
2. Merge the two progams above so that random numbers are displayed until the button is pressed, then 10 random numbers are generated and it stops for 5 seconds
3. Make the electronic dice display 2 random numbers to simulate 2 dice
4. Make your own dice that is different to this described so far with some interesting sound feature

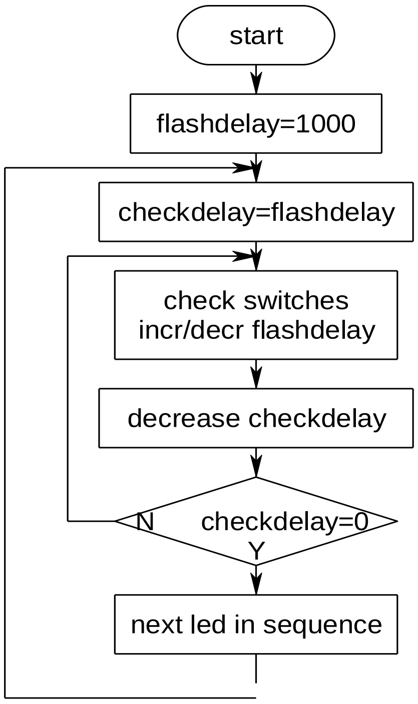
Achieved	Merit	Excellence
Do number 1 and 2 above with comments in the program	Also implements 3 above and uses lots of comments to explain the program	Implements 4 above with good explanatory comments in the program.

11.24 Modified Knightrider

A neat feature for the Knightrider program would be if the speed of the sequence could be varied.

So for the same reasons as before the switches need checking often; so after each led in the sequence of LEDs, read the switches, wait a preset amount of time, if one button is pressed increase the delay time, if the other button is pressed decrease the delay time.
The switches should be checked often so that they can detect user input and I have chosen 1mS because its easy to do the maths with 1mS.

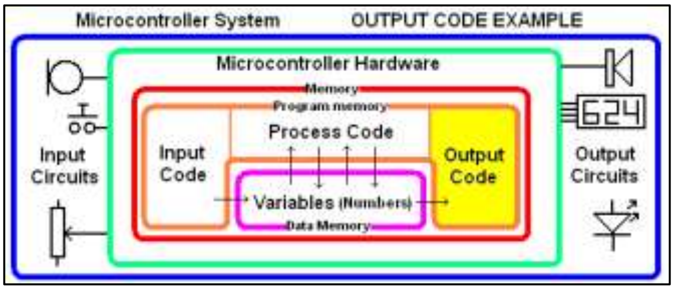
To do this we implement a loop within the program that initially begins at the value of flashdelay and counts down to 0, a second variable checkdelay is needed as a copy of flashdelay




```
Dim Flashdelay As Word
Dim Led As Byte
Dim Checkdelay As word
dim direction as bit
Flashdelay = 1000
Do

    Checkdelay = Flashdelay
     Do
        Debounce Sw1 , 0  , Decrflashdelay, Sub
        Debounce Sw2 , 0  , Incrflashdelay, Sub
        Decr Checkdelay
     Loop Until Checkdelay = 0
     If Direction = 0 Then
        Gosub Nextright
     Else
        Gosub Nextleft
     End If
Loop
End
'

Subroutines
Decrflashdelay:
  Decr Flashdelay
Return

Incrflashdelay:
  Incr Flashdelay
Return

Nextright:
…
Return

Nextleft:
…
return
```

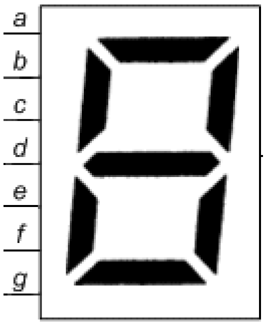
12 Basic displays

12.1 7 segment displays



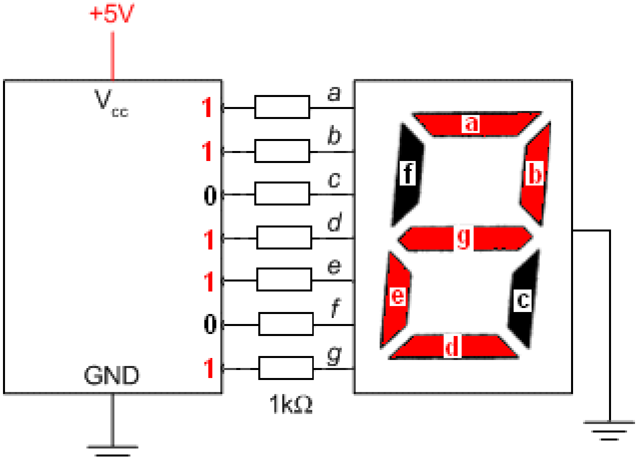
It is important to understand a new device so that it can be used with confidence. The 7 segment display is simply a number of LEDs put together inside a package with pins sticking down so that they can be soldered into a PCB. They are still very common today in many electronic products.

They are available in many different styles and sizes.

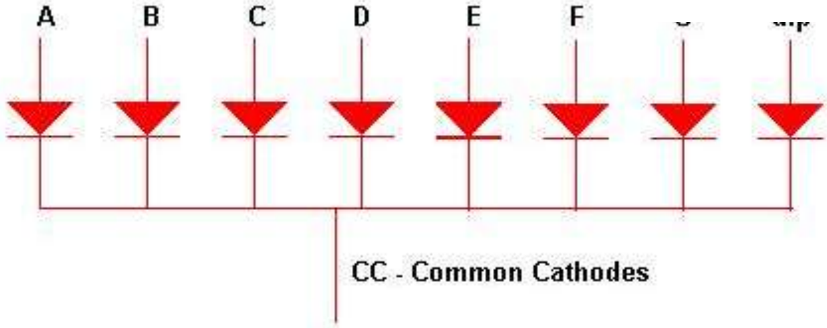


The first thing to know is how the LEDs are connected within the package.

Each LED is a segment of the display and they are labelled a, b, c, d,e,f,g

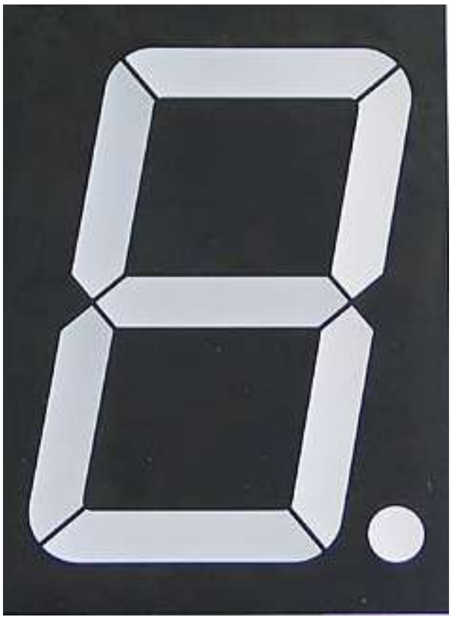


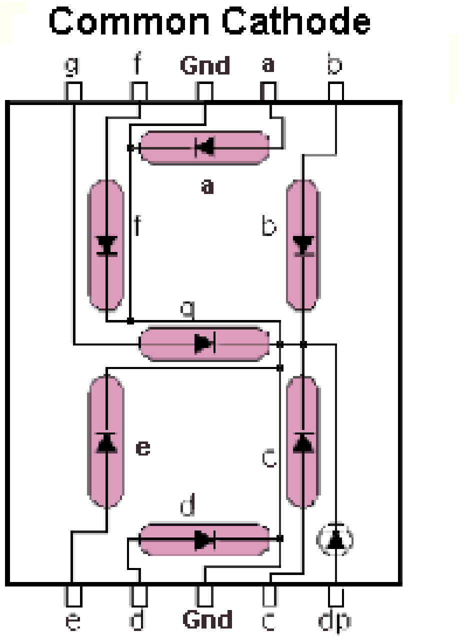
To create the number 2 you turn on segments a, b, d, e, g



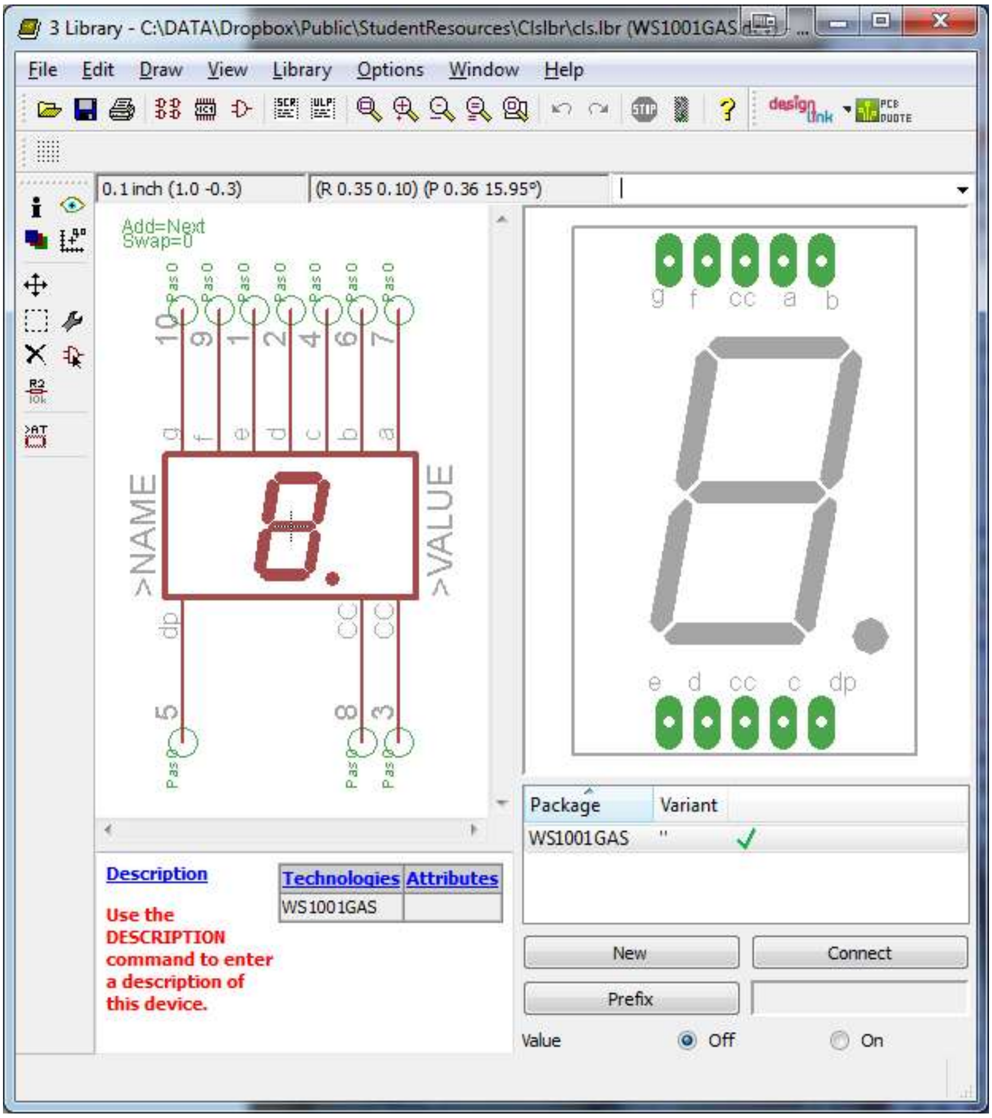
The LEDs have separate Anodes but COMMON Cathodes so our display is called ‘common cathode’. All the cathodes (negative ends) are connected togther and will be connected to the negative (0V or ground) of the circuit.

This display is the WS1001GAS, we had no datasheet for it, so had to figure out the wiring for ourselves.

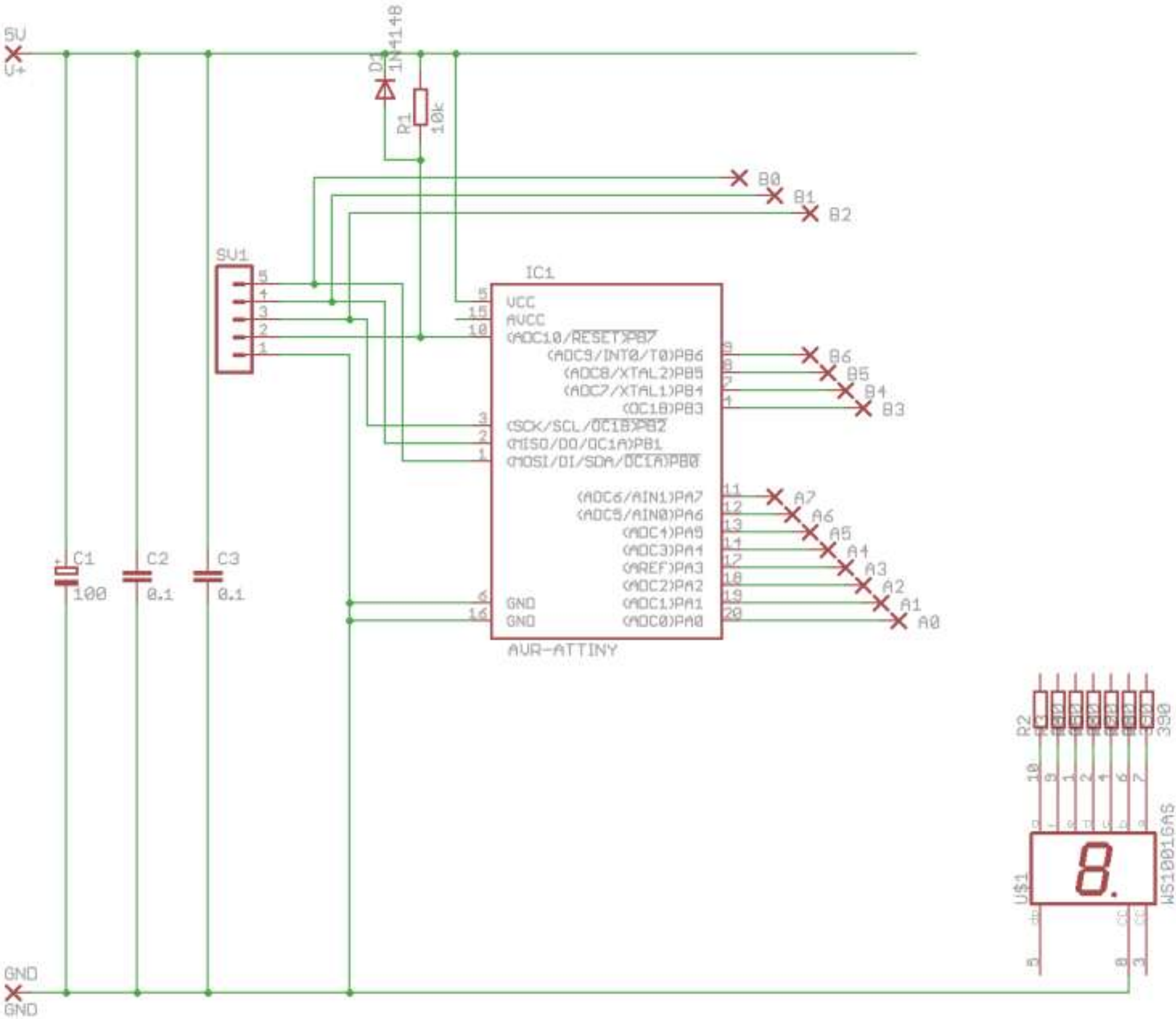




It had 10 pins and using a powersupply on 5V and a resistor we figured out what each pin does. Then realised that tgis seems to be a reasonably standard setup for the displays.
A component for the Eagle library was created so it could be used in making our own schematics.

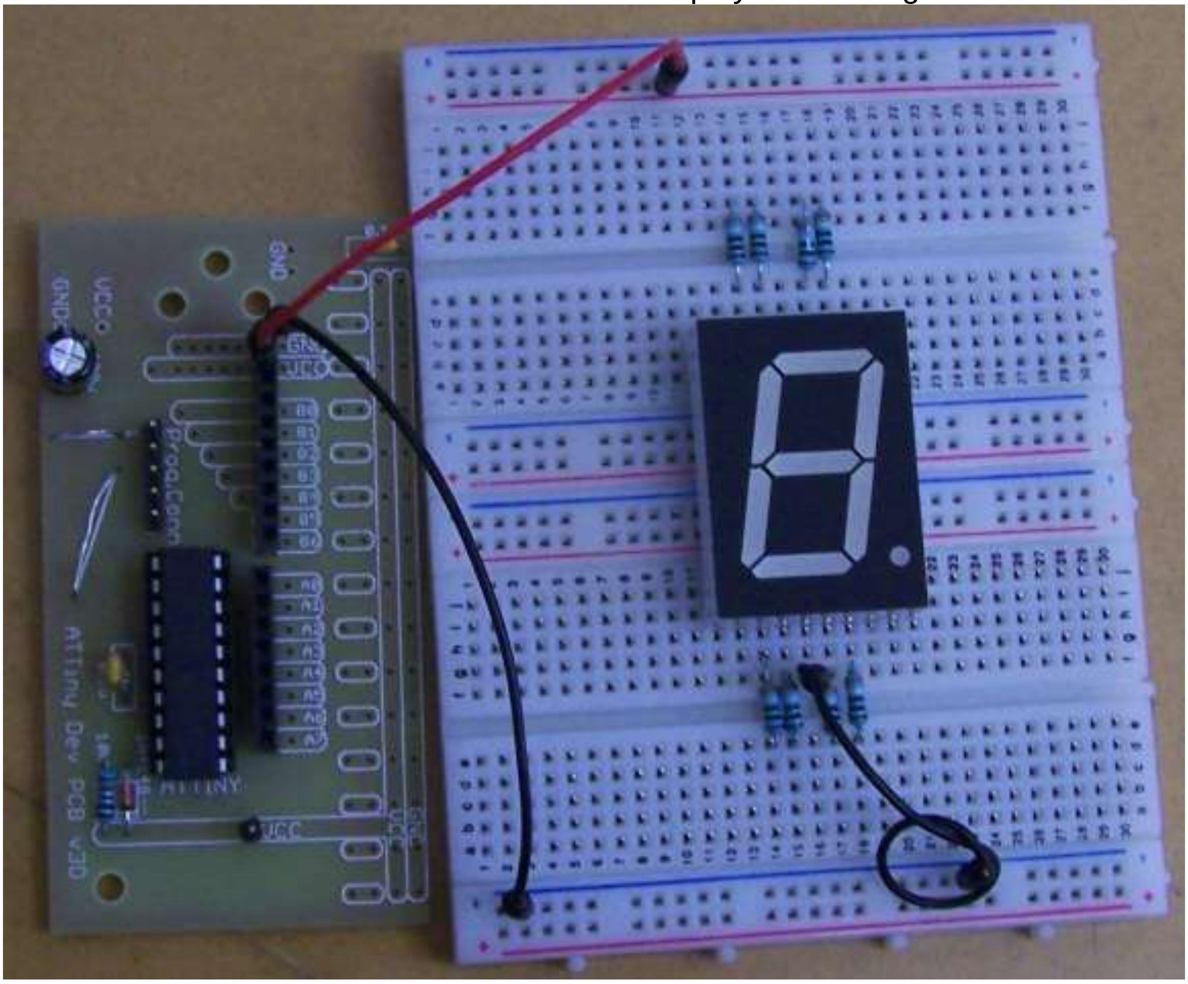

3 Library - C:\DATA\Dropbox\Public\StudentResources\Clslbr\cls.lbr (WS1001GAS
File Edit Draw View Library Options Window Help
0.1 inch (1.0 -0.3)
(R 0.35 0.10) (P 0.36 15.95°)
Add=Next
Swap=0
>NAME
>VALUE
g f cc a b
e d cc c dp
Package
Variant
WS1001GAS
Description
Technologies
Attributes
WS1001GAS
Use the DESCRIPTION command to enter a description of this device.
New
Connect
Prefix
Value
Off
On

The schematic was started.

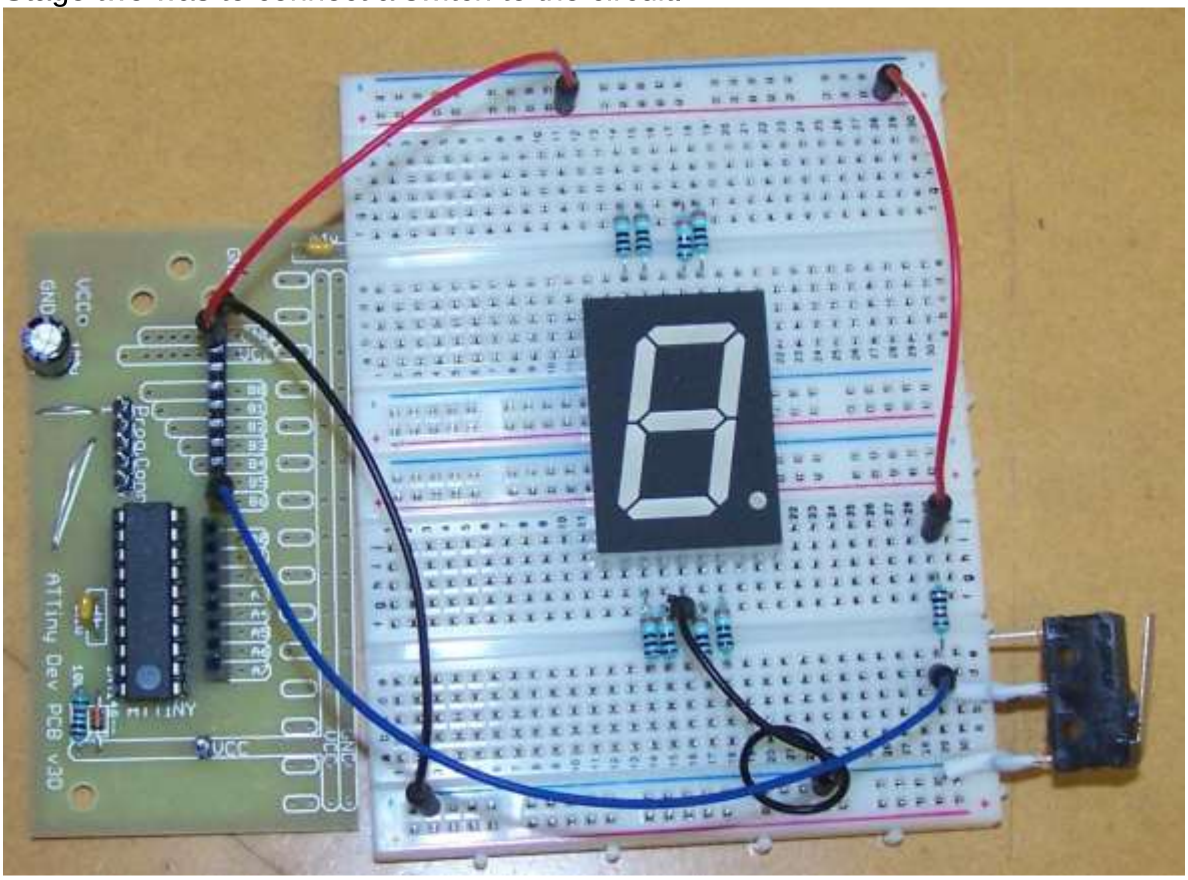


You only have to connect one of the two pins 3 or 8 to gound not both.
We didn’t connect the pins on the schematic as it is easier to connect the pins on the layout and figure out which is best and then draw the schematic afterwards.

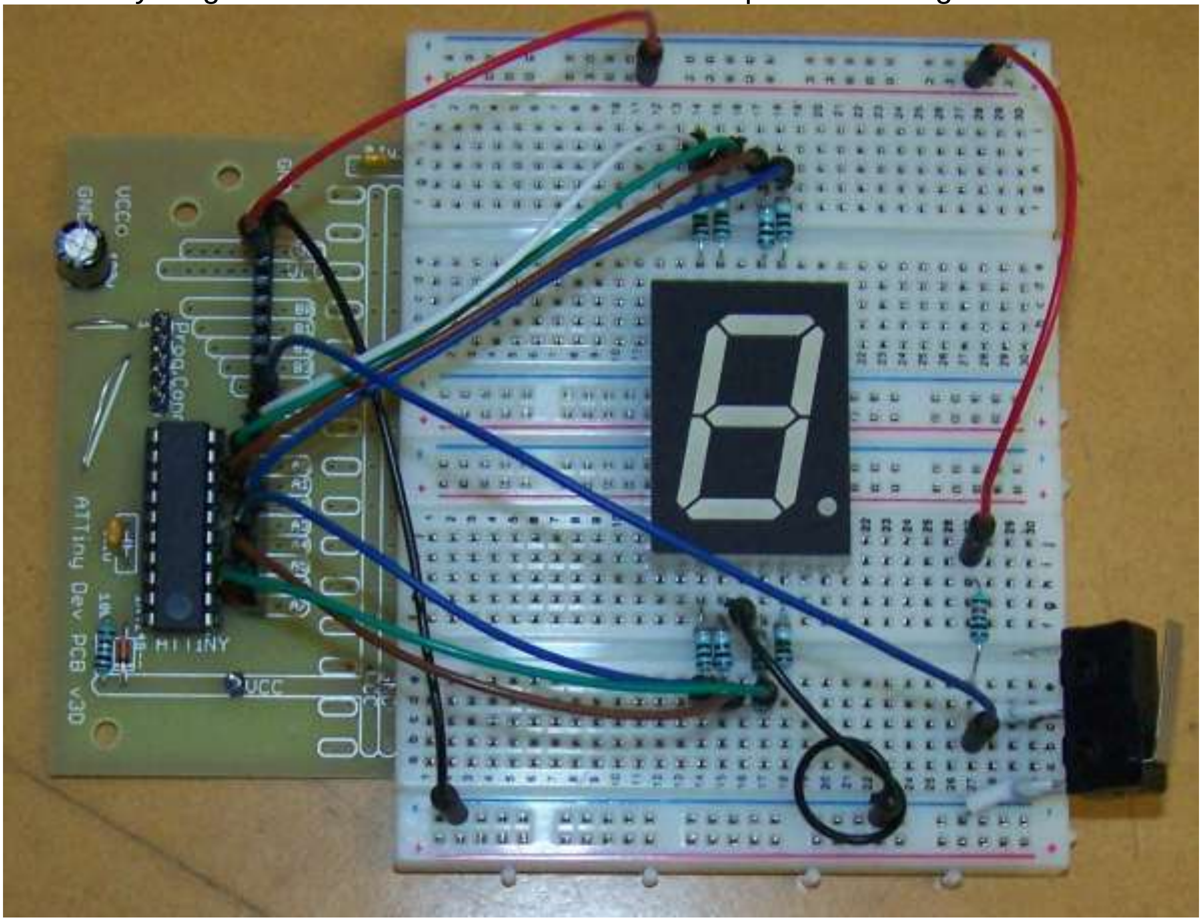
Start with the display, the 8 resistors and the power conections to the breadboard
We had to use two breadboards because the display was too big to fit onto one.



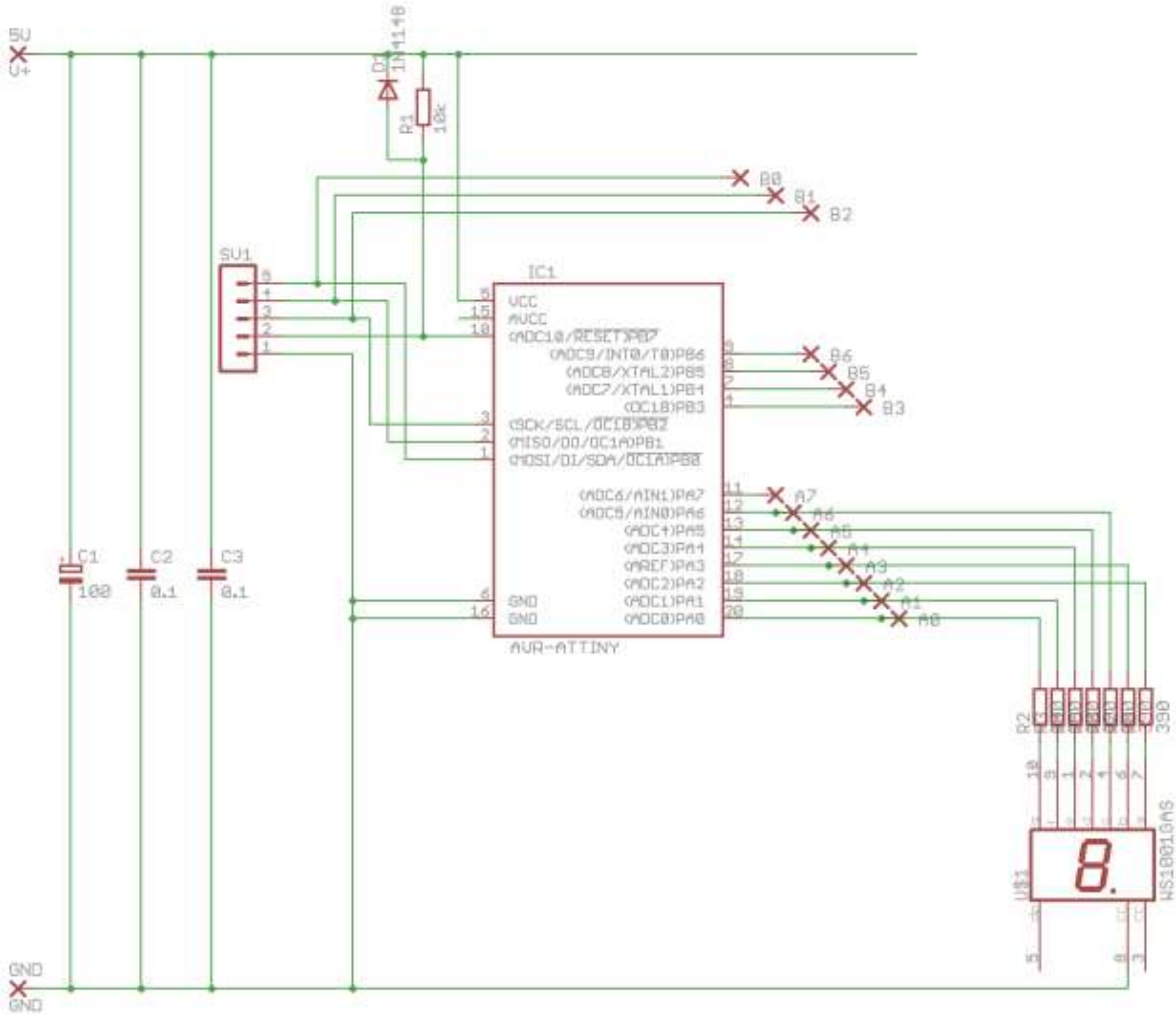
Stage two was to connect a switch to the circuit.



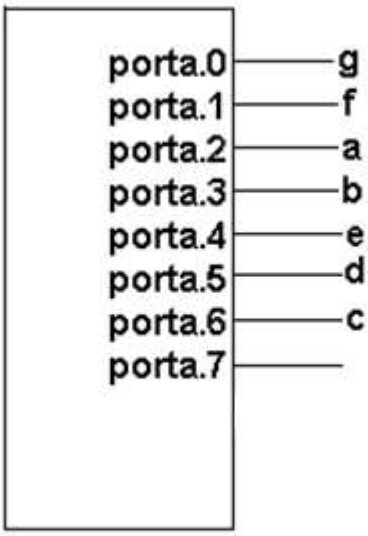
And finally stage 3 to connect the microcontroller IO pins to the segments

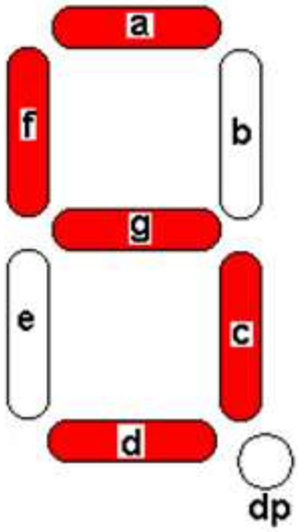


After wiring the schematic was completed in Eagle

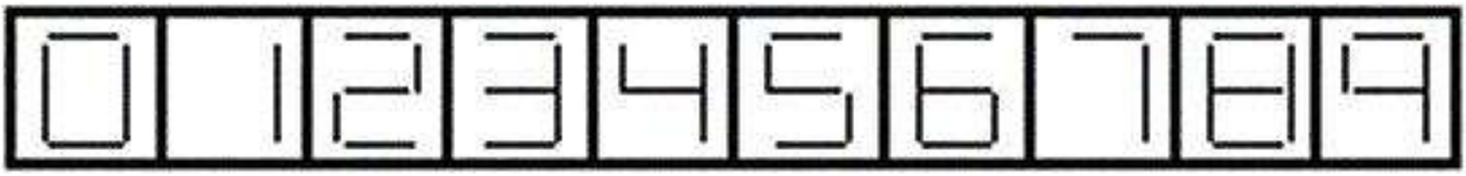


Outcome development in Technology education includes not just making the product (outcome) but includes the development of it. Things such as tables to help manage your programming will help you achieve really good results in Technology. Complete the table below and use things like tables yourself to logically lay out things .





To display the number five, only the segments a,c,d,f &g must be on. and the code &B01100111 must be written out the port. Work out the other values required to show all the digits on the display and determine their corresponding values in Hex and Decimal and put them in the table below
NOTE portA.7 isnt used so it will always be 0, the order is **0cdebafg** for this wiring

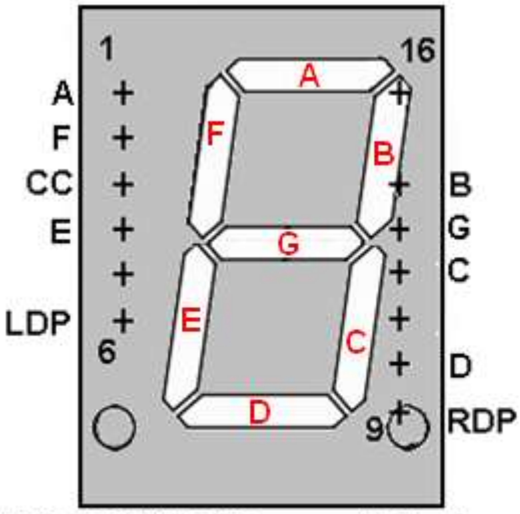


Display	Segments ON	Segments OFF	PORT Binary command Segment order is 0cdebafg
0			
1			
2			
3			
4			
5	a,c,d,f,g	b,e	&B01100111
6			
7			
8			
9			
A			
b			
C			
d			
E			
F			
g			
H			

Complete this table for yourself

Another different 7 segment display

This particular display was made by OasisTek.
Note that it has 2 decimal points (LDP and RDP)

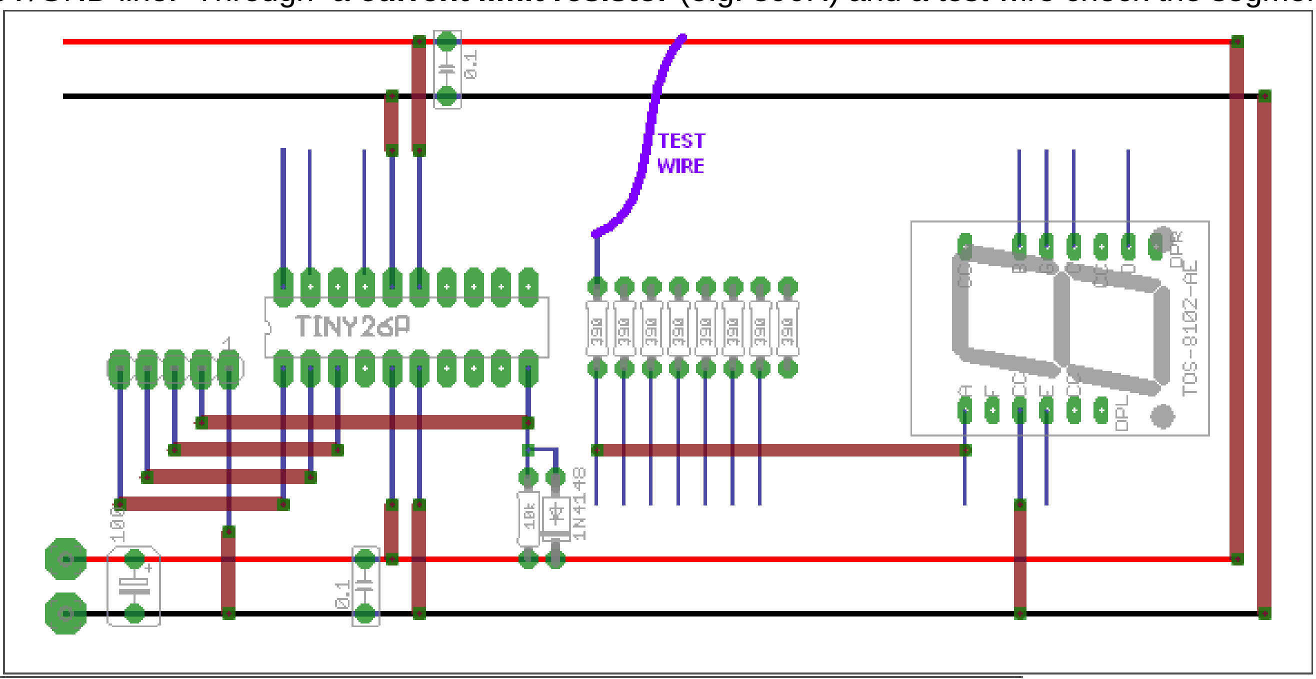


TOS-8102AE 7Segment Display pin connections

This view is from the front of the display; the **+** indicates the pins in the two rows underneath.
Pin 1 is segment A
Pin 2 is segment F
Pin 3 is the common cathode
Pin 4…

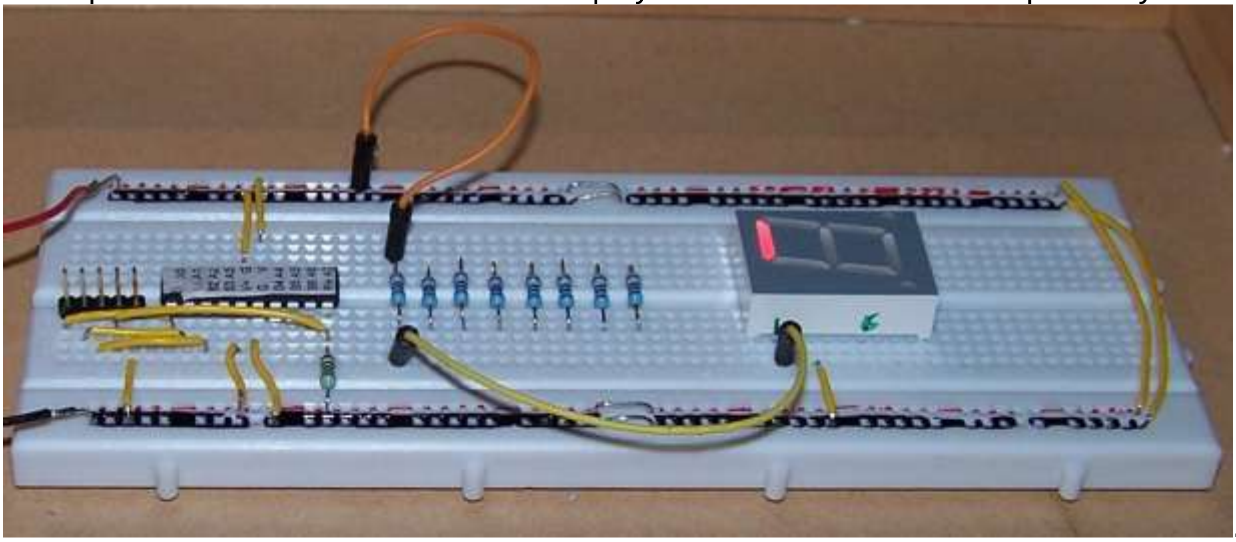
Note that although pins 7,8 &15 don't exist they are still counted!!

Connect the 7segment display to the breadboard, so that the common cathode is connected to the 0V/GND line. Through a **current limit resistor** (e.g. 390R) and a test wire check the segment works .



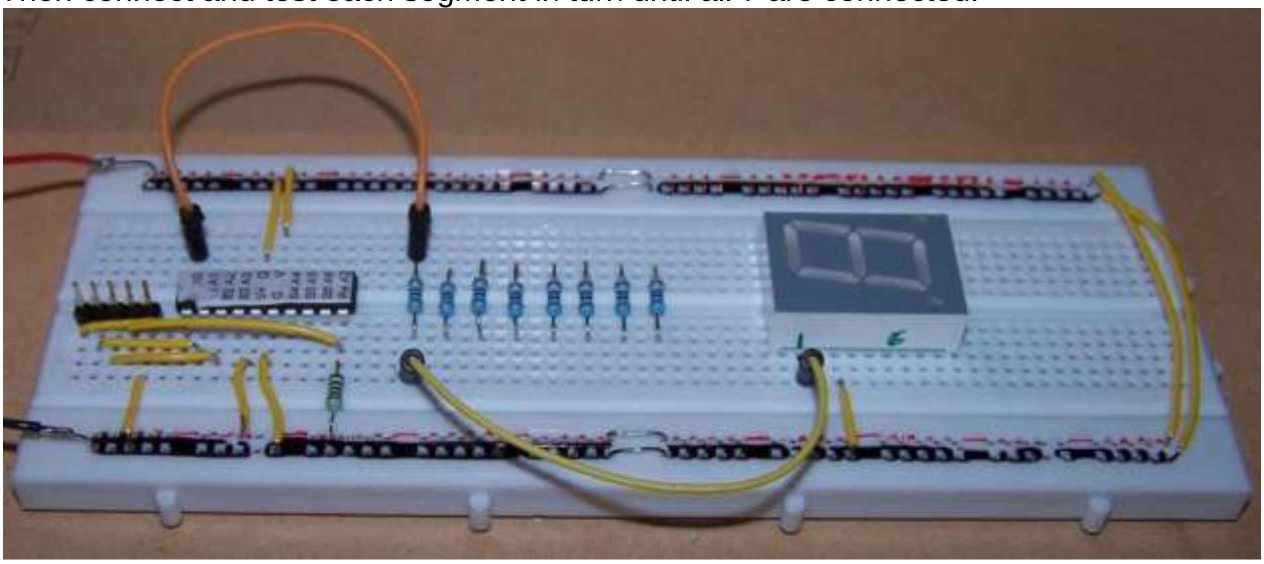
Each segment should glow like segment a does in the next picture.

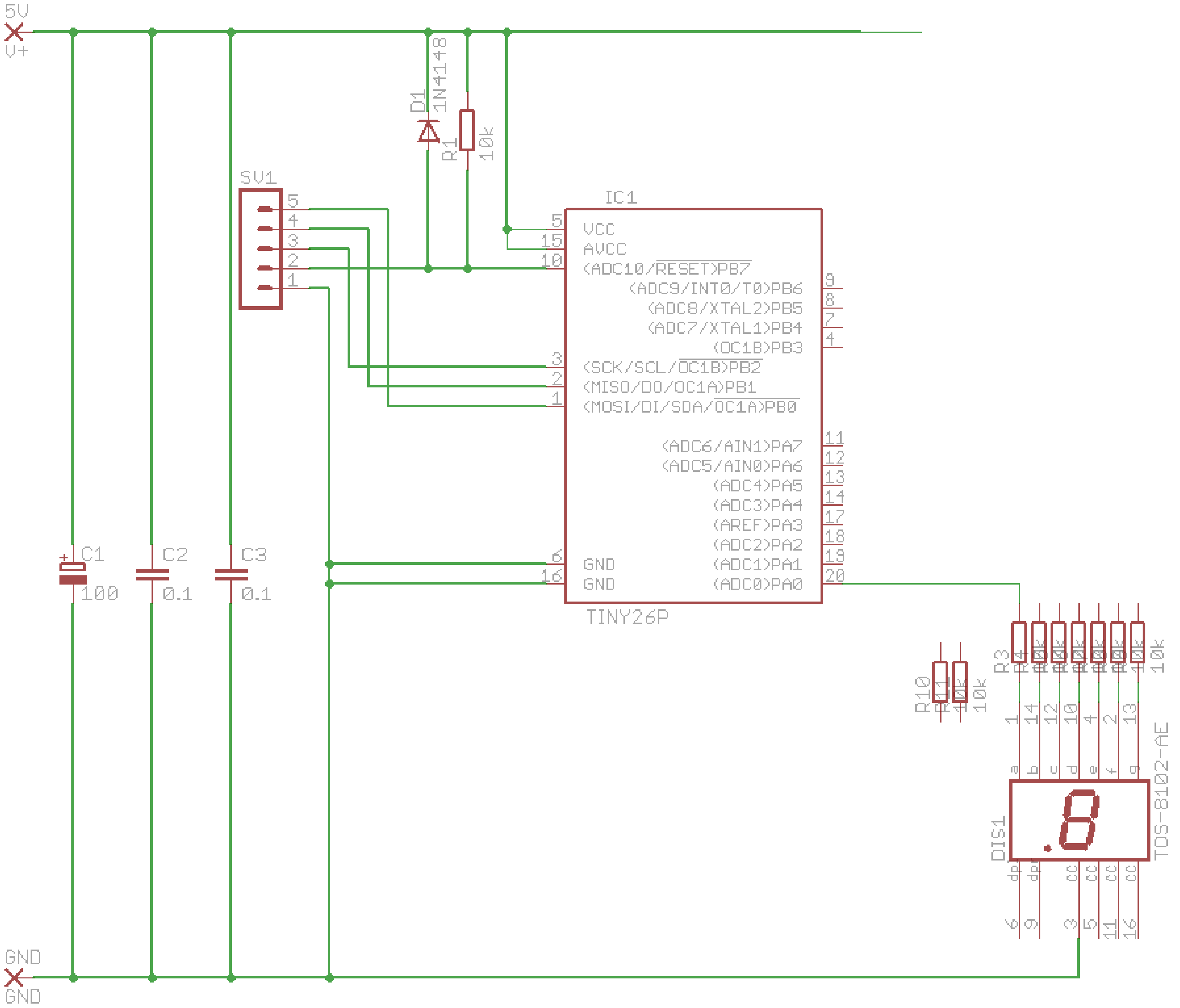
In the photo below note the side of the display has been written on to help identify where the pins are



.

After testing the segment connect it to the correct pin of the microcontroller.
Then connect and test each segment in turn until all 7 are connected.



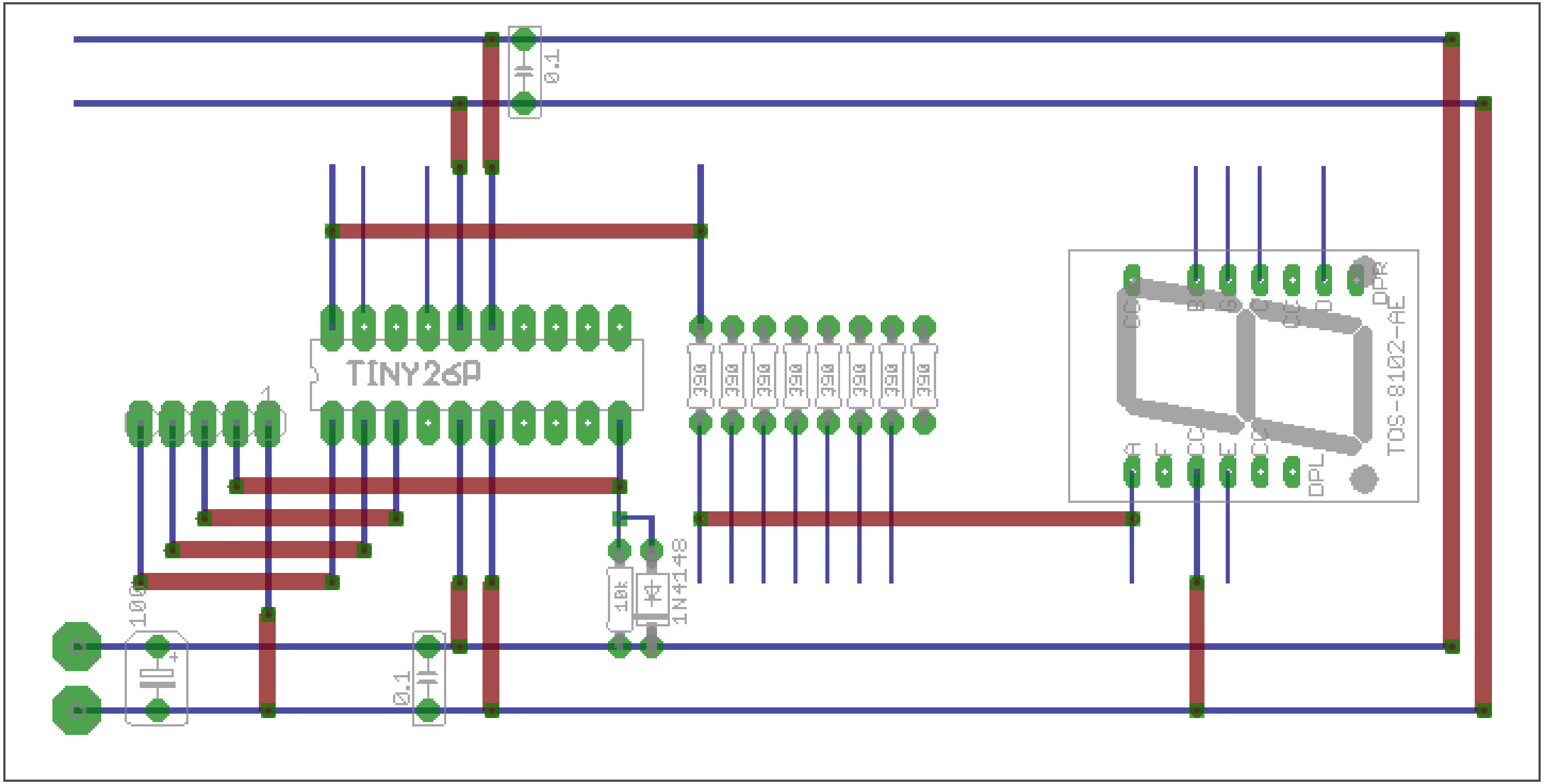


Complete this diagram with the rest of the connections to the 7 Segment display, then make the circuit on breadboard. On the next page you will find a layout partially started.

Draw a flowchart and write a program to display the numbers 0 to 9.

Design as many letters of the alphabet as you can and write your name. **Print your name program for your workbook**.

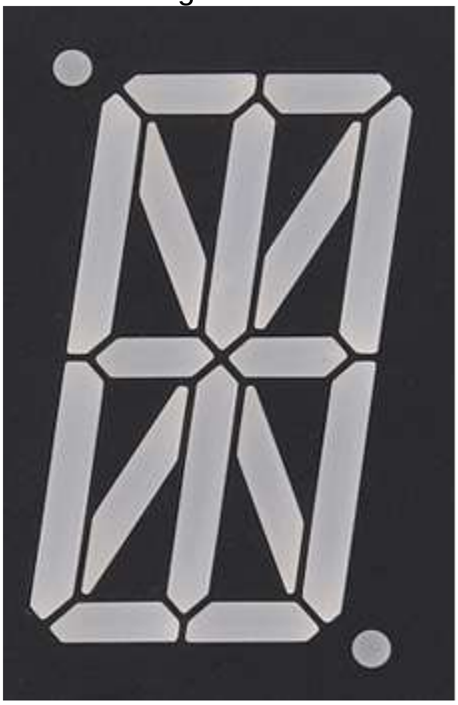
Complete this diagram with the connections for all of the seven segments

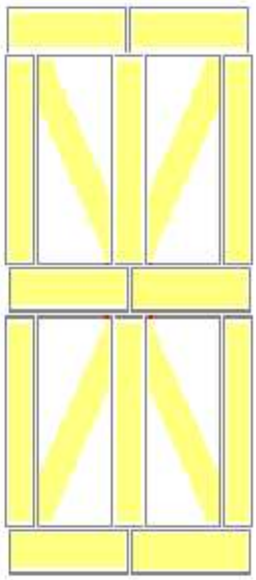


CAN YOU MAKE THIS 7 SEGMENT DISPLAY INTO A DICE?

12.2 Alphanumeric LED displays

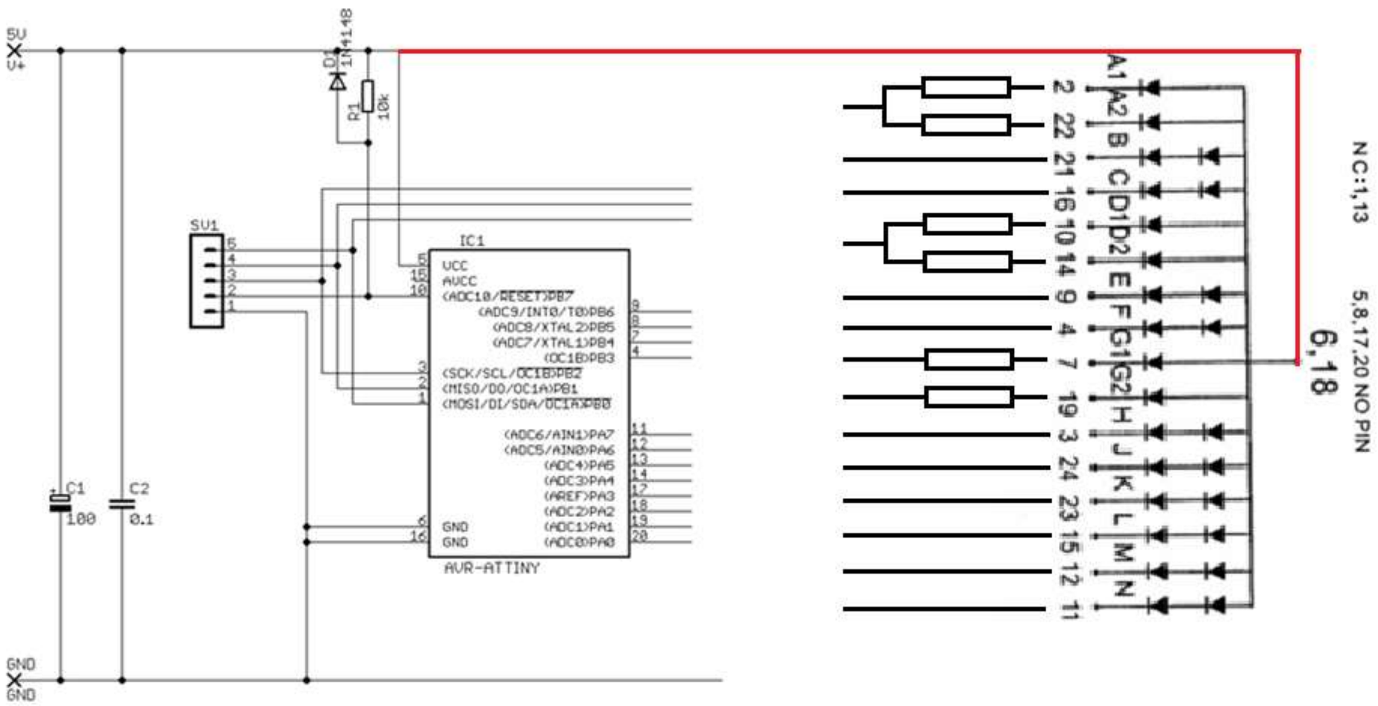
These are very similar to the 7 Segment, but have multiple segments so that you can easily make letters as well as digits.



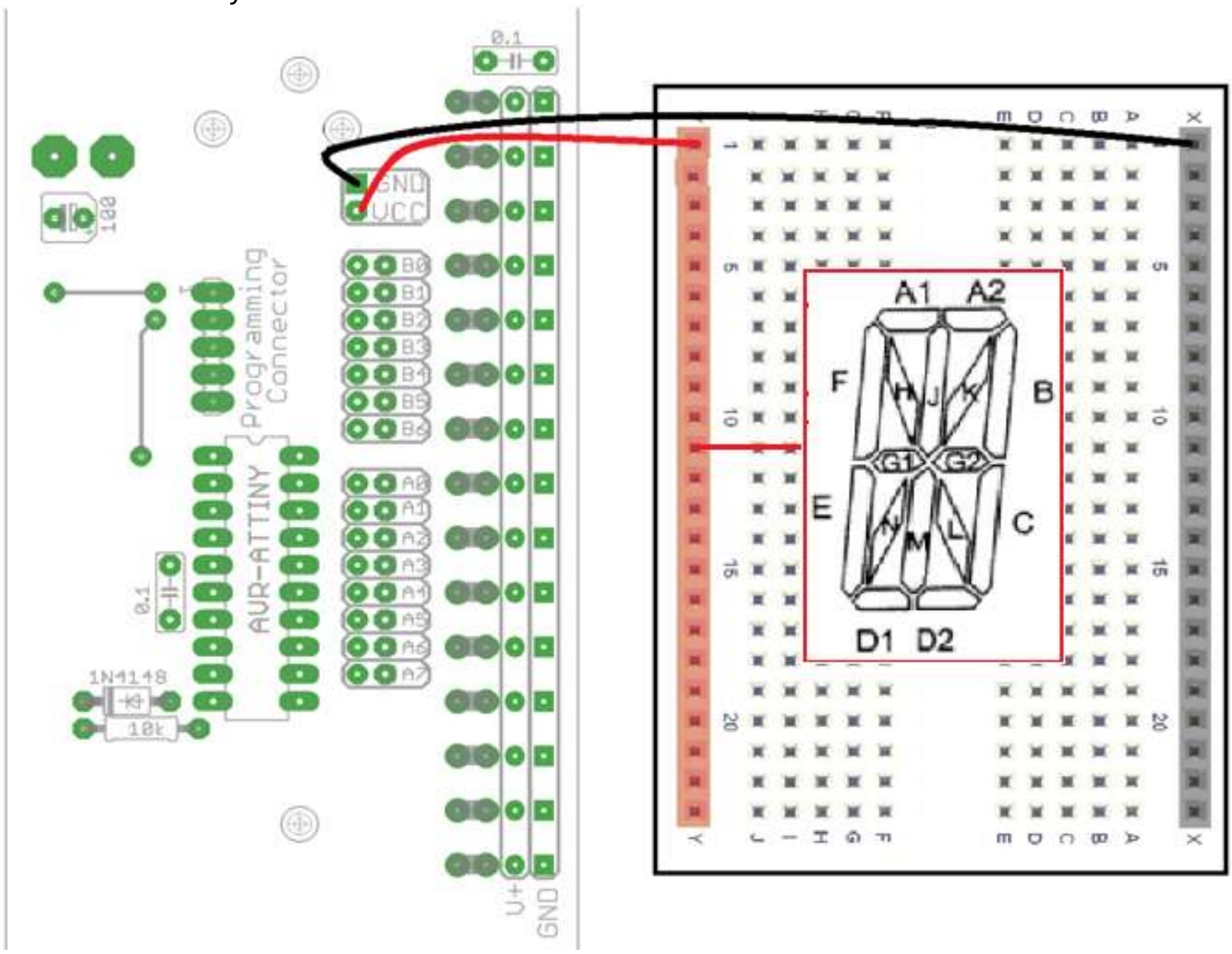


Here there are 16 segments plus 2 decimal points.
Now the ATtiny461 doesn't have 16 I/O pins so we combined a couple of the segments and used only 14 I/O pins of the micro.

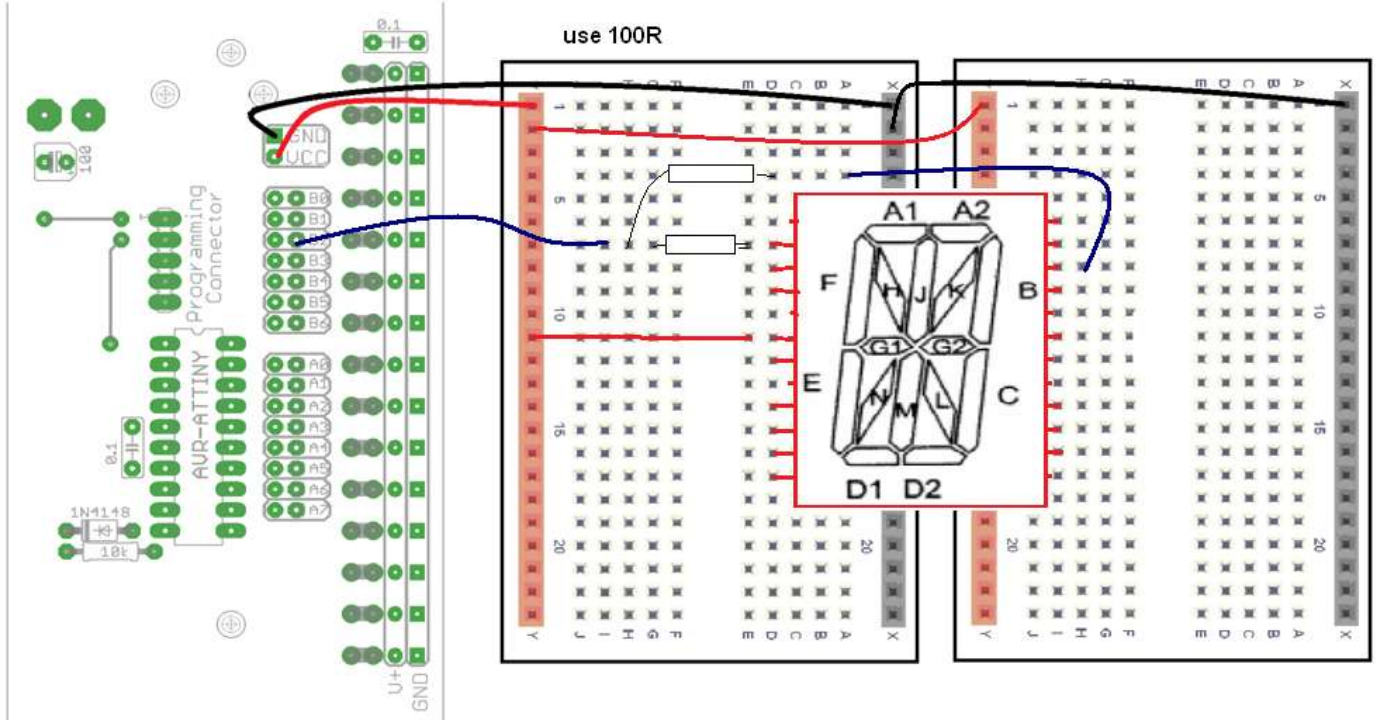
In this schematic the top two segments A1 and A2 were combined and used as 1 also the bottom two segments D1 and D2.



Here is how the layout looked

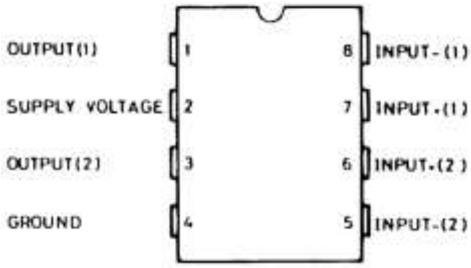


See below how A1 and A2 were linked together to one I/O pin (but 2 100ohm resistors were used)

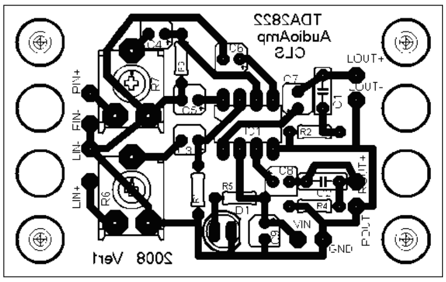


13 TDA2822M Portable Audio Amplifier Project

This project is based around the TDA2822M IC (integrated circuit) from a company called SGS Thompson Microelectronics.



The project involves making a portable (battery powered) audio amplifier that can be used with an MP3 player and keeping a portfolio of the processes used. You may design and make or modify something else fro your case



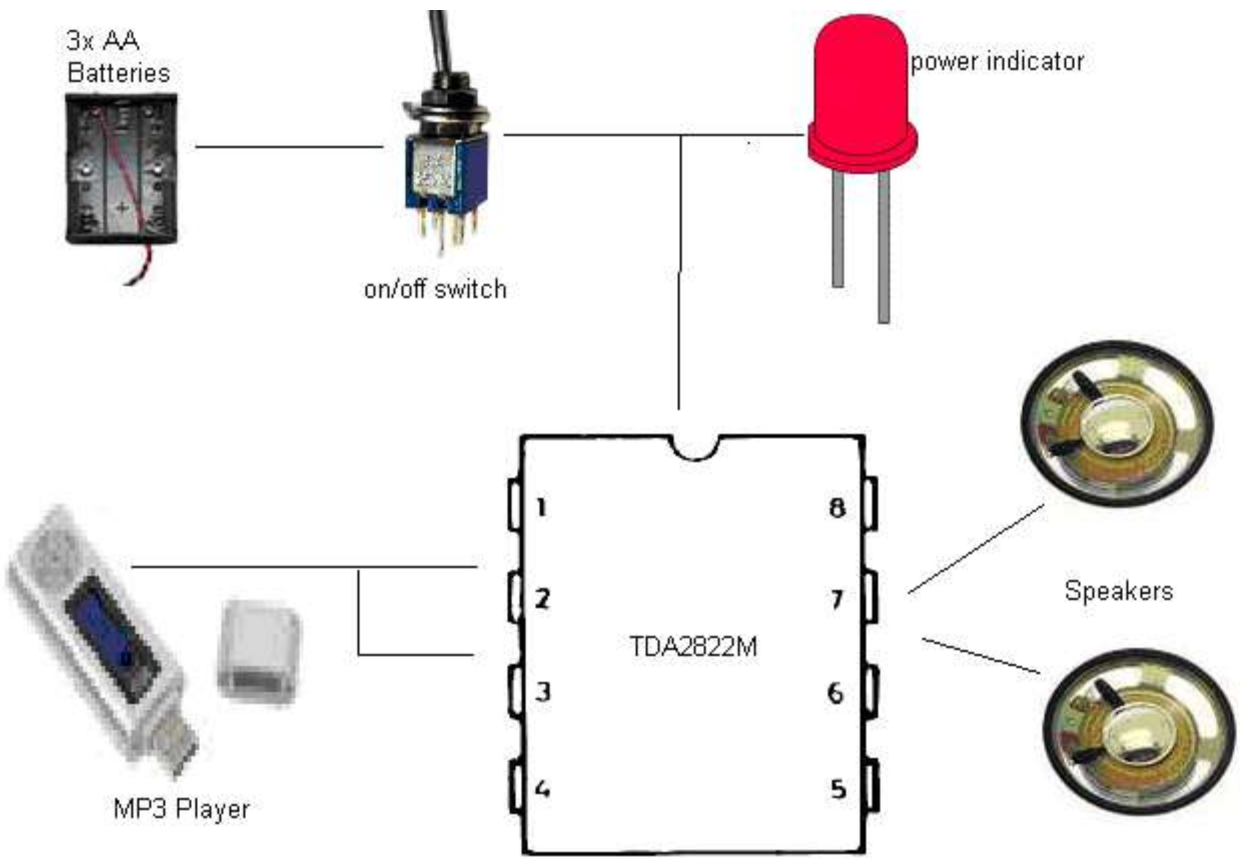
You will design and make the printed circuit board and case for the amplifier. You may use the provided speakers (50mm, 8 ohm 0.5W) or find your own.

13.1 Portfolio Assessment Schedule

Achieved	Merit	Excellence
Workbook content		
Printed Datasheet	Printed Datasheet	Printed Datasheet
Component Price List	Component Price List	Component Price List
Schematic Diagram from Eagle	Schematic Diagram from Eagle	Schematic Diagram from Eagle
Layout Diagram from Eagle	Layout Diagram from Eagle	Layout Diagram from Eagle
OHT of PCB	OHT of PCB	OHT of PCB
Board works	All solder joins reliable, heat shrink used correctly to strengthen joints, stress relief on all wires	All solder joins reliable, heat shrink used correctly to strengthen joints, stress relief on all wires
CAD Design drawing for case	At least two design drawings for case With changes made	AT least two design drawings for case With detailed explanation for changes
Photo of case	Photos of case + some description of process of making	Photos of case With detailed explanations of process of making
Final Outcome	Quality outcome, (refer to codes of practice)	Final product shows some flair, elegance, innovation or creativity, and explanation is given of these elements
Workbook Presentation		
Material is readable	All materials are clear, labelled, named and follow a logical sequence	Overall presentation is easy to follow and all materials are very well presented, a table of contents is given and page numbers are used.
Key Competencies		
Interacts with others occasionally or when asked to work in groups	Works cooperatively, relates easily and shares workshop resources freely with others.	Helps others and seeks others help in the workshop often
Cleans up after self	Works cooperatively with others to clean up the workshop	Takes initiative in keeping the workshop clean and tidy, puts tools and materials away for others regularly
Generally uses workshop time well	Efficient use of workshop time	Disciplined, optimised and efficient use of workshop time

13.2 Initial One Page Brief

Project: **TDA2822 Portable Audio Amp** Date: _____

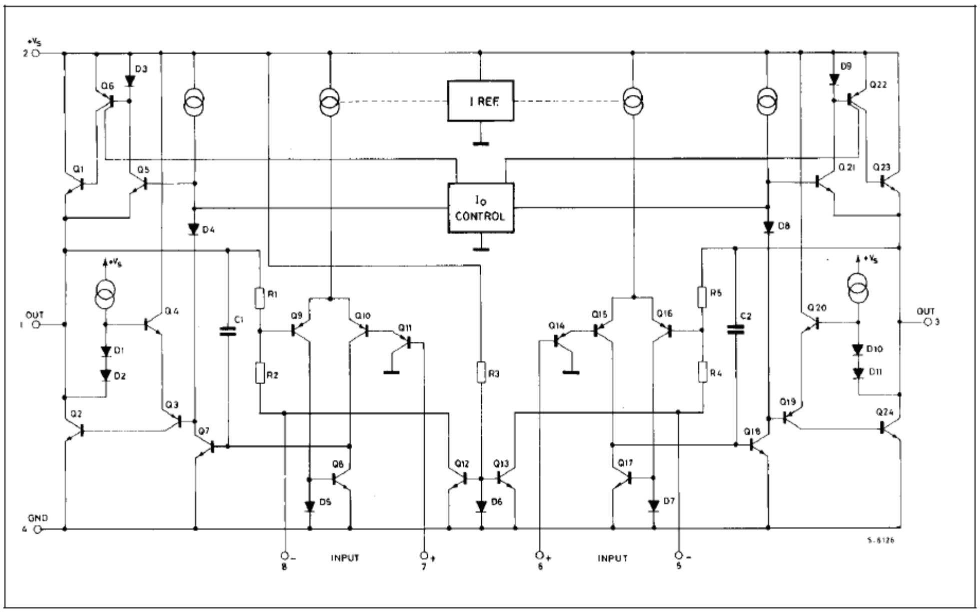
Client, customer or end-user: **ME**!
Description of the problem, issue, need or opportunity(diagrams may be required): **MP3 players are useful personal items however the music cannot be shared with others**
Conceptual Statement: **Design and construct a portable audio amplifier to allow music to be played when with a group of friends**
System Block Diagram: (include all input and output devices) 
Further written specifications: **Need to make or find a case for it all**

13.3 TDA2822M specifications

Electronic components are complex (especially IC's) and manufacturers provide detailed specifications called **datasheets** for their products.

Find and print the datasheet for your portfolio of the TDA2822M, it is easily available on the WEB. It contains things such as the pin connections, a simplified internal schematic diagram, recommended circuits and voltage, current and power specifications.

SCHEMATIC DIAGRAM



ELECTRICAL CHARACTERISTICS (V_S = 6V, T_{amb} = 25°C, unless otherwise specified)

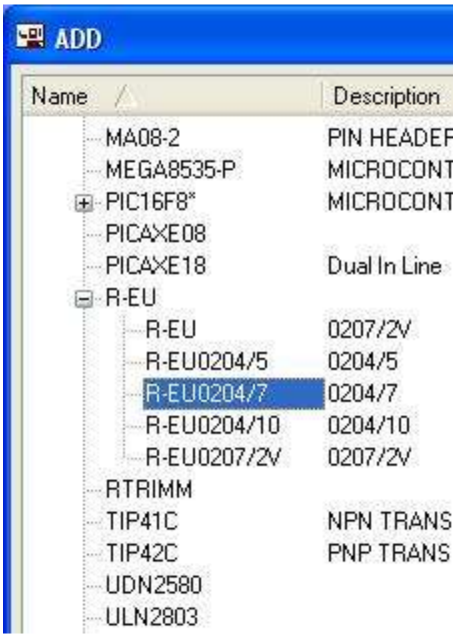
Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
STEREO (test circuit of Figure 1)						
V_s	Supply Voltage		1.8		15	V
V_o	Quiescent Output Voltage	 V_s = 3V		2.7 1.2		V V
I_d	Quiescent Drain Current			6	9	mA
I_b	Input Bias Current			100		nA
P_o	Output Power (each channel) (f = 1kHz, d = 10%)	 R_L = 32Ω V_S = 9V V_S = 6V V_S = 4.5V V_S = 3V V_S = 2V R_L = 16Ω V_S = 6V R_L = 8Ω V_S = 9V V_S = 6V R_L = 4Ω V_S = 6V V_S = 4.5V V_S = 3V	 90 15 170 300 450	 300 120 60 20 5 220 1000 380 650 320 110		mW
d	Distortion (f = 1kHz)	R_L = 32Ω P_o = 40mW R_L = 16Ω P_o = 75mW R_L = 8Ω P_o = 150mW		0.2 0.2 0.2		% % %
G_v	Closed Loop Voltage Gain	f = 1kHz	36	39	41	dB
ΔG_v	Channel Balance				± 1	dB
R_i	Input Resistance	f = 1kHz	100			kΩ
e_N	Total Input Noise	R_s = 10kΩ B = Curve A B = 22Hz to 22kHz		2 2.5		μV μV
SVR	Supply Voltage Rejection	f = 100Hz, C1 = C2 = 100μF	24	30		dB
C_s	Channel Separation	f = 1kHz		50		dB

13.4 Making a PCB for the TDA2822 Amp Project

Open eagle and create a new schematic.
From your schematic Click the ADD button in the toolbox and the ADD dialog box will open (it may take a while)

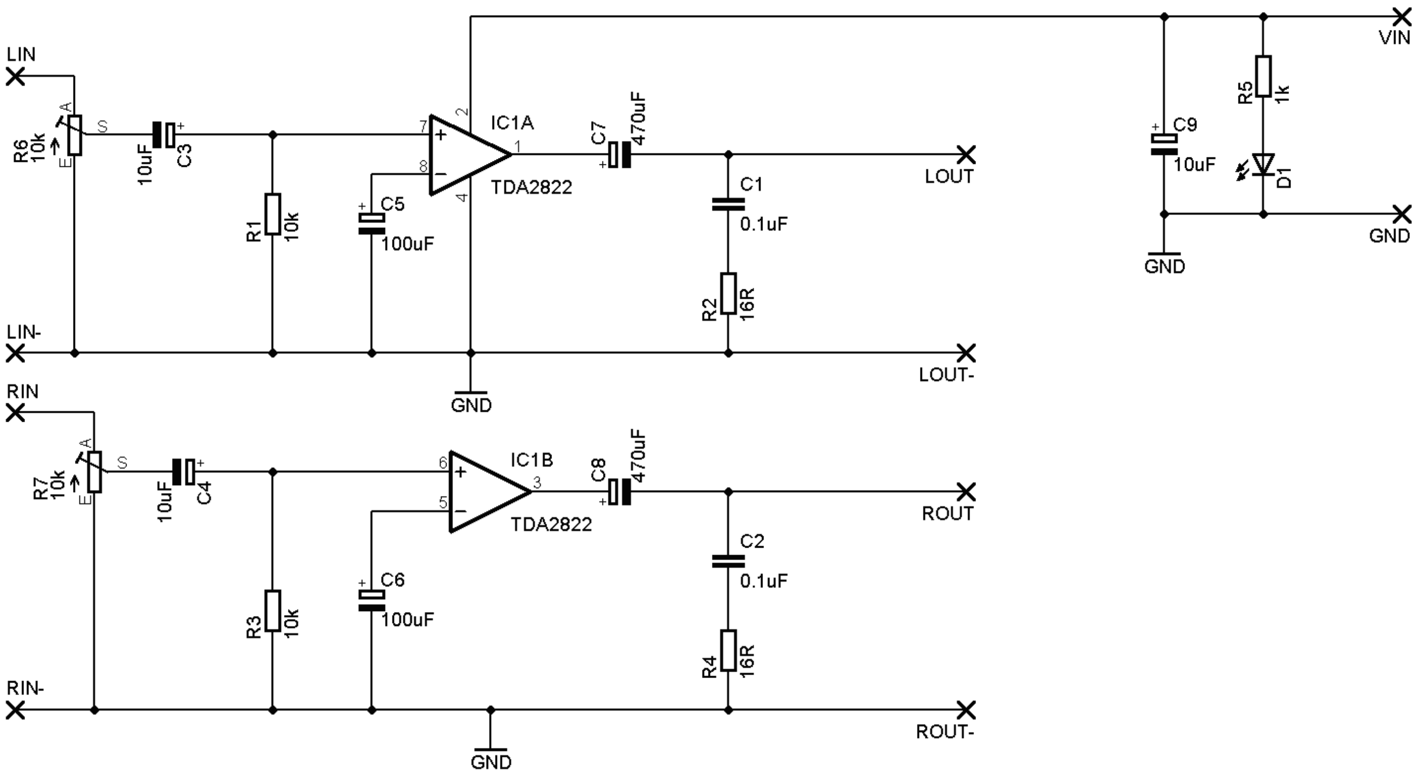
Open the CLS library
Add all of the following parts

LIBRARY	PART	Qty
cls	REU-0204/7	6
cls	2,54/0,8	10
cls	C-EU050-025x075	2
cls	C-POLB45181A	5
cls	C-POLE5-10,5	2
cls	led 5MM	1
cls	TDA2822	1
cls	RTRIMMECP10S	2
cls	GND	3



13.4.1 Moving parts

Move the parts around within the schematic editor so that they are arranged as per the schematic below.

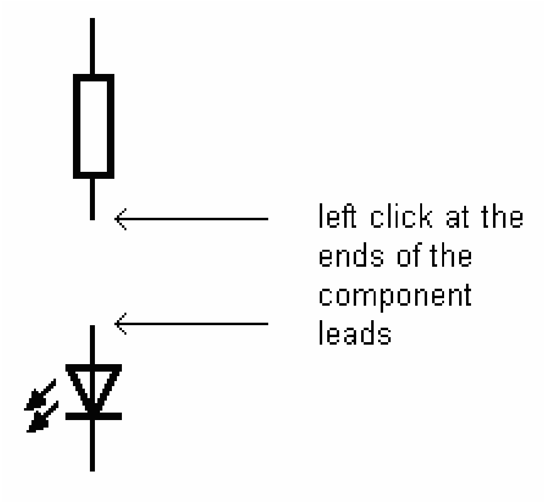


13.4.2 Wiring parts together

Select the net button from the toolbox.
Remember to left click on the very end of a component and draw in a straight line either up, down, left or right.

Left click again to stop at a point and draw before drawing in another direction.

Click at another component or net to finish the connection.



13.4.3 ERC

The ERC tests the schematic for electrical errors.

Errors such as pins overlapping, and components unconnected are very common.

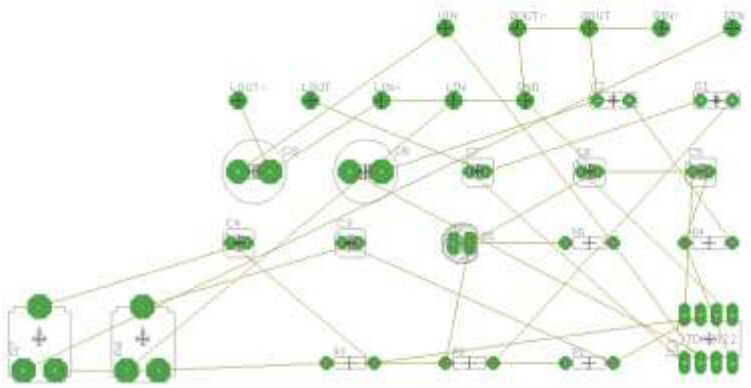
The ERC gives a position on the circuit as to where the error is; often zooming in on that point and moving components around will help identify the error.

<u>You must correct all errors before going on</u>.

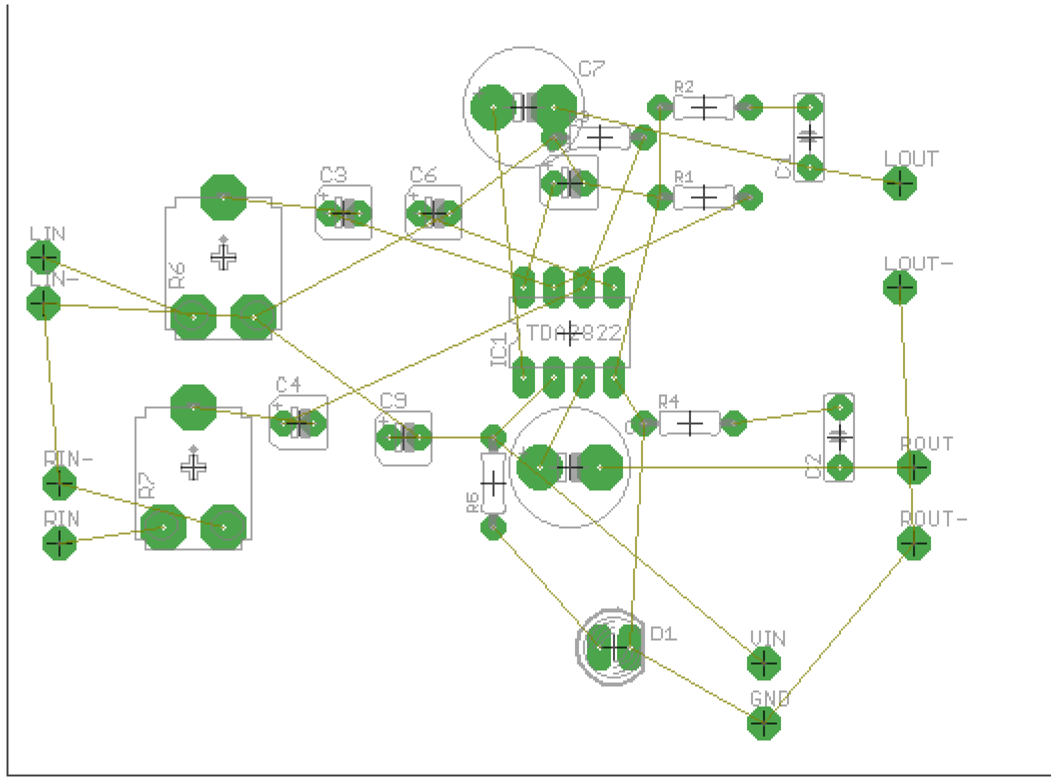
13.4.4 Laying out the board

Open the board editor

<u>Remember: once you have started to create a board always have both the board and schematic open at the same time, never work on one without the other open or you will get horrible errors which will require you to delete the .brd file and restart the board from scratch.</u>



13.4.5 Minimise airwire length



Move the components into the highlighted area. Keep the components in the lower left corner near the origin (cross).

Reduce the size of the highlighted area you are using for the components. Then zoom to fit.

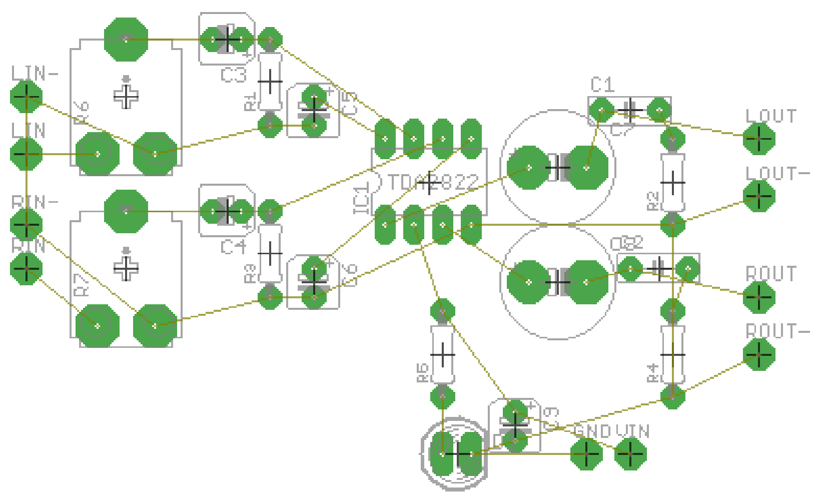
Progressively arrange the components so that there is the minimum number of crossovers.

As you place components press the Ratsnest button often to reorganize the Airwires. Eventually your picture will look like the one here.

Good PCB design is more about placement of components than routing, so spending most of your time (90%) doing this step is crucial to success.
You want to make track lengths as short as possible

13.4.6 Hiding layersto help you see the airwire paths clearly

The DISPLAY button in the TOOLBOX is used to turn on and off different sets of screen information. Turn off the names, and values while you are placing components. This will keep the screen easier to read. Turn off the layer by selecting the display button and in the popup window pressing the number of the layer you no longer want to see.
Turn off tnames and tvalues now



13.4.7 Routing Tracks

Now is the time to replace the airwires with actual PCB tracks. Tracks need to connect all the correct pads of the components together without connecting together other pads or tracks. This means that tracks cannot go over the top of one another!
Select the ROUTE button and on the Toolbar make sure the Bottom layer is selected (blue) and that the track width is **0.04**. Left click on a component. Note that around your circuit all of the pads on the same net will be highlighted.

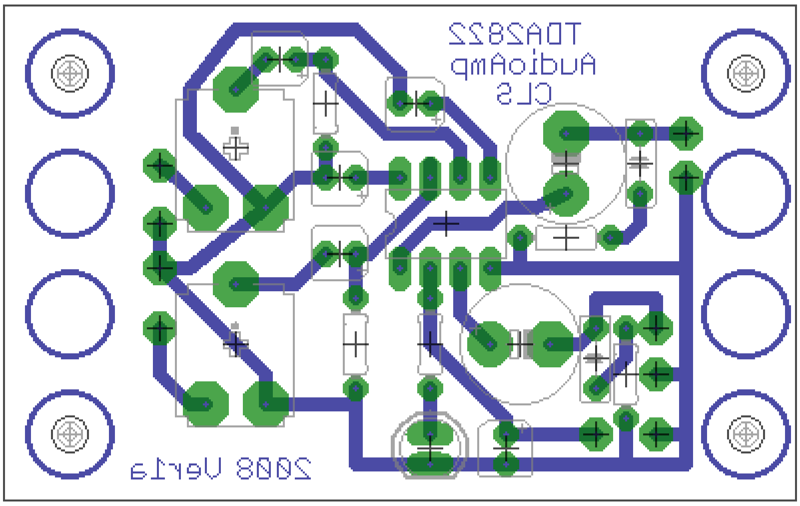
Route the track by moving the mouse and left clicking on corner points for your track as you go. YOU ONLY WANT TO CONNECT THE PADS ON THE SAME NET, DON'T CONNECT ANY OTHERS OR YOUR CIRCUIT WILL NOT WORK.

Track layout Rules

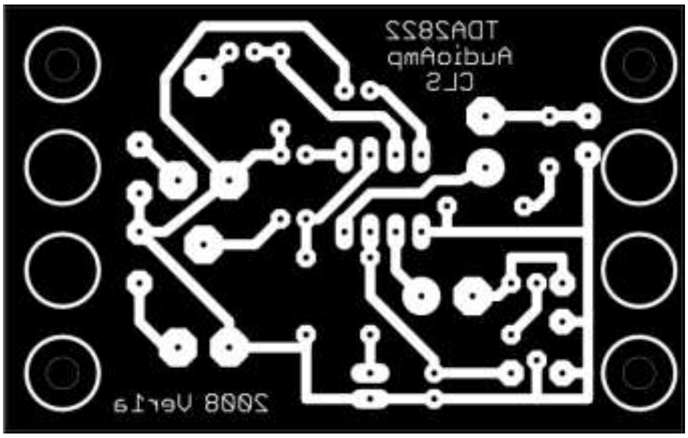
1. **Route tracks so that no track touches the leg of a component that it is not connected to by an airwire**
2. **No track may touch another track that it is not connected to by an airwire**
3. **Tracks may go underneath the body of a component as long as they meet the above rules**

After track routing add holes for mounting the board and any for looping wires through to act as stress relief DO NOT ROUTE TRACKS BETWEEN THE PINS OF IC'S

13.4.8 Make the Negative Printout

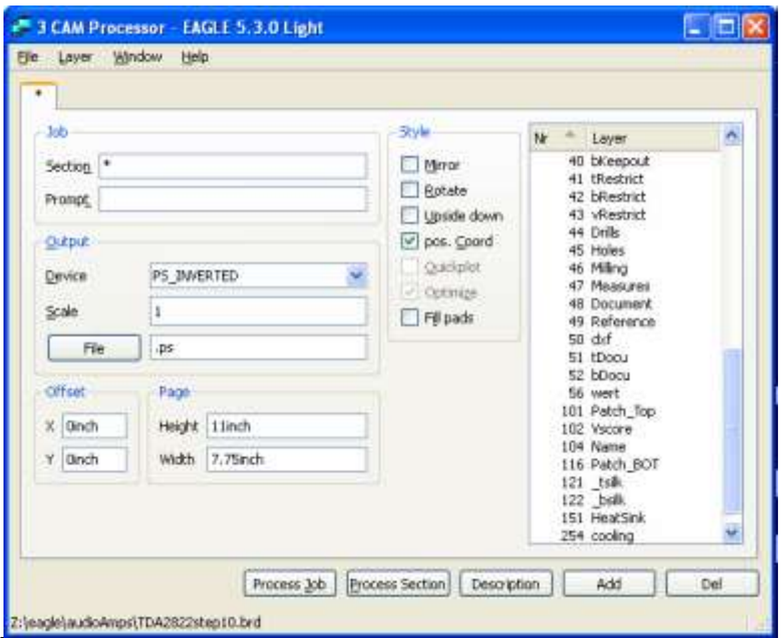


(Remember the text on the PCB appears reversed)



* Open TDA2822verA.brd in Eagle
* From within the Eagle Board Editor start the CAM Processor
* select device as PS_INVERTED
* Scale = 1
* file = **.ps**
* make sure fill pads is **NOT** selected this makes small drill holes in the acetate which we use to line up the drill with when drilling
* for layers select only **16,17,18 and 20**,
* make sure **ALL** other layers are **NOT** selected.
* Select process job

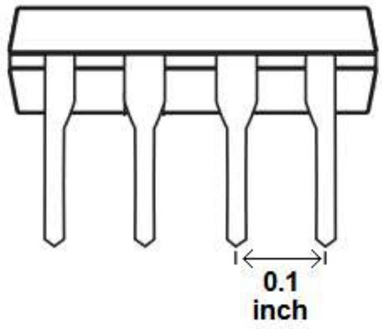
Open the TDA2822verA.ps file with Ghostview. Double check that you can see the drill holes and then print it on to an OHT (transparency)



13.5 Extra PCB making information

Grids

An important point to note is that the rulers and grids in Eagle are generally in inches, this is because IC's (such as the TDA2822) and other components have legs that are 0.1 inch between centres.



The current grid spacing is shown in the layout window most likely as 0,05 inch, if you want to see the actual grid, type **grid on**. For all layouts we will use inches because that is the spacing of component legs. Although when we specify a drill size wew ill use mm. Also never change the grid size, we will use 0.05 inch (50 thou). If you want ot start squeezing things together – well don't especially in your first few boards. it just makes the boards hard to etch and to solder.

Track width, copper thickness and current ratings

The board we buy is 2oz (ounces), that means the amount of copper in one square foot of pcb is 2oz, That equals 0.0028 inches thick (2.8 thou – or just to confuse you PCB people often say 2.8mils). We generally use 0.032 or 0.04 inch tracks on our boards in the classroom as they print and etch easily.

Even though tracks are made of copper and are a conductor, they are not perfect conductors and have some resistance. This means that as charges move through the circuit the tracks get warm! The thinner they are the higher the resistance and the warmer they get. If they get too hot they will burn up (and smoke and possibly flames will appear).

A track of 0.04 inches width on the boards we use is about 0.006 ohms per inch will when carrying a current of 4 amps will rise in temperature by around 10 degress which is ok. Our circuits don't in general need to carry 4 amps but its good to know this sort of thing. If you want to carry 10amps then go to about 0.15 inch to be on the safe side!

Grounding

The ground connection is a circuit is the path for current back to the power supply, and the bigger and the more of it we can make the better. We almost always make single sided pcbs so its a good idea to put a ground right around the whole circuit board. There is an example of using polygon fill later on.

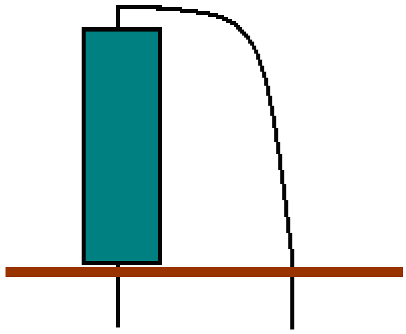
Forwards and Backwards

You must always have your schematic and layout open at the same time, if you have only one open then any changes you make to one will not appear on the other. Then when you open them both Eagle will complain and say that no forward-backward annotation will happen, now you are stuffed, it can actually take longer to fix annotation problems tha starting all over again!

13.6 Component Forming Codes of Practice

Component leads are bent at least 2 mm away from the component body, not bent close to the body as this would stress the component and reduce its life expectancy.

The component is placed firmly against the PCB. This helps mechanical rigidity. (Components would only be put up off the board if during normal operation they would become warm enough to damage the PCB itself)

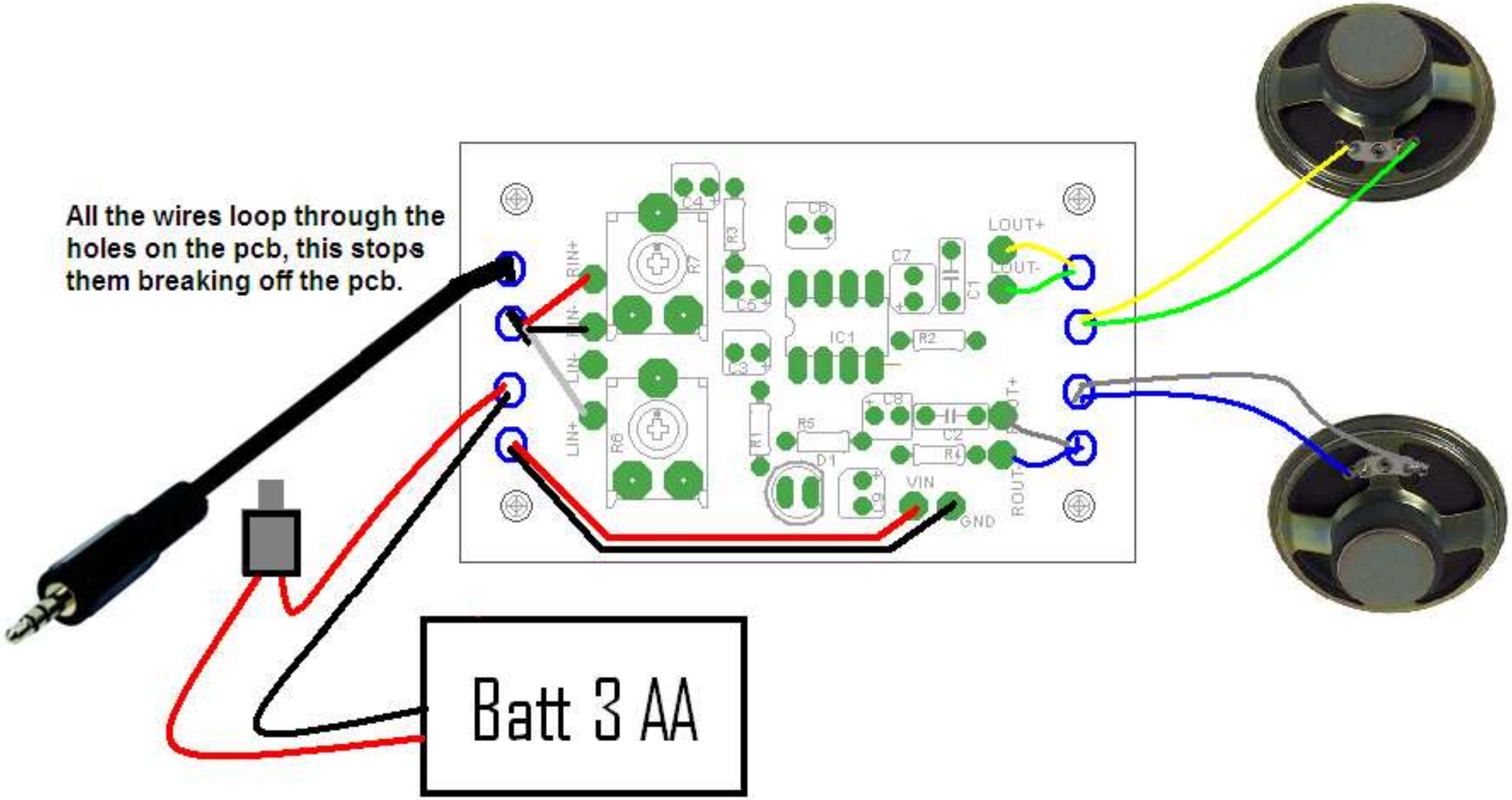


If there is not enough room to lay the component flat on the PCB then one leg may be bent over.

Under the pcb the component leads are bent over slightly to hold the component in place during soldering, they are not bent flat as then it would be difficult to remove the component later on.

Component leads are cut off after soldering; during soldering they act as a heat sink and keep excess heat away from the component.

13.7 TDA2811 wiring diagram

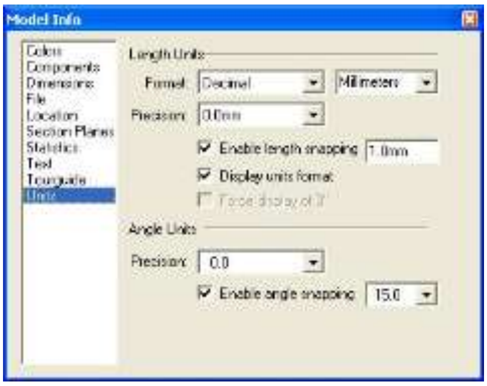


Make the wiring as neat as possible on the speaker connector

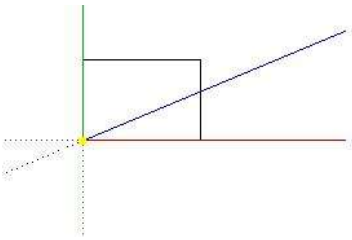
Solder it and get your soldering checked by the teacher

Cover it with heatshrink, make sure the heatshrink covers the wire and connector to protect the wire from being pulled off

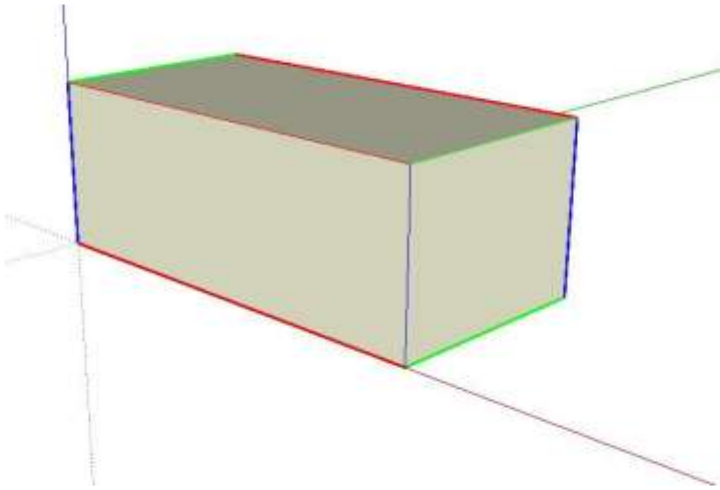
13.8 SKETCHUP Quick Start Tutorial



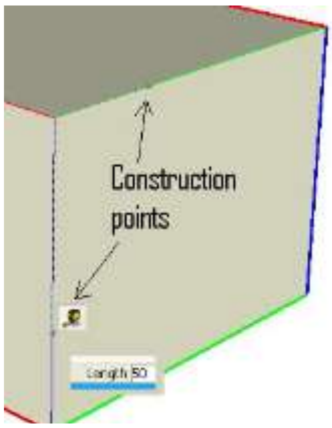
1. From the menu select <u>Window</u> then <u>Model Info</u> and then select units, set up units as shown in this picture.
2. Close this dialog box
3. Select the <u>Rectangle</u> tool in the toolbox (the set of tools on the left hand side of the SketchUp window).
4. Click the nouse mouse pointer once on the origin and move it right and upwards to start drawing a rectangle (do not click again to stop drawing).



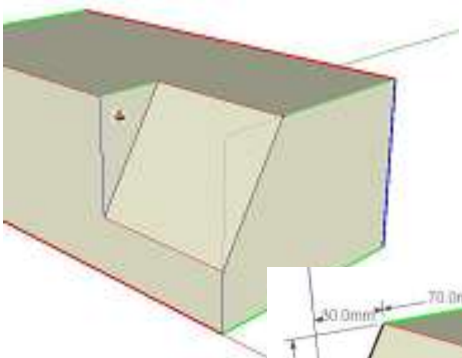
5. In the bottom right hand corner the dimensions of the rectangle are shown; without clicking there just type on the keyboard <u>200,100</u> and <u>press Enter</u>. The rectangle will take on the dimensions you have typed in.
6. Your rectangle may well have disappeared because you are zoomed out too much. From the tool box, identify the <u>zoom extents</u> tool by hovering the mouse pointer over the buttons. Get use to the other zoom controls now and zoom out a little.
7. From the menu select <u>Window</u> then <u>Display Settings</u> and change the <u>Edge Color</u> to <u>By Axis (</u>now you can see whether what you are drawing lines up with the axis you want it in).



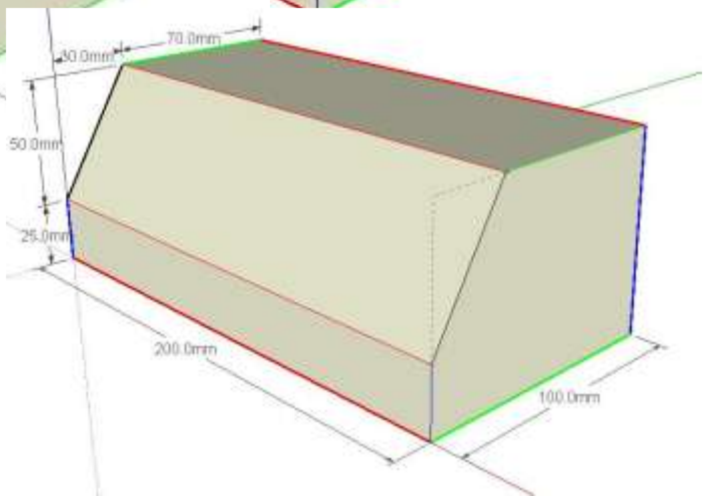
8. Under the menu is the tool bar identify the <u>Iso</u> view button (isometric) and click it.
9. In the toolbox identify the <u>Push/Pull</u> tool and then move the mouse pointer over the rectangle, the rectangles surface will change in appearance. Click once on the surface and drag the rectangle upwards along the blue axis into a 3D box; type <u>75</u> as a dimension and press enter. Your box should be aligned to the three axes and the edge colours should match the axes colours.
10. Select the Tape Measure from the toolbox and click on the upper front right corner and then move along the green axis, type 30 and press enter, a grey construction point will appear. From the same corner place another construction point 50mm down the blue axis.



11. From the toolbox choose the line tool and draw a line between the two construction points, notice how the cursor snaps to the construction points as it nears them (it also snaps to edges, ends and centre points and each has a different colour).



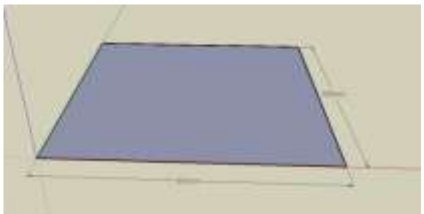
12. Using the push pull tool push the new surface completely away to change your box to one with a sloping front panel.
13. From the toobox select the <u>Dimension</u> tool, add dimension lines by hovering the mouse over an edge line (it will change to yellow), then click on the line and drag the new dimension away from the edge to place it.

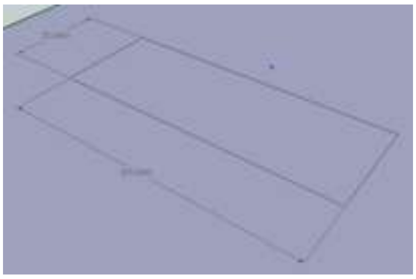


13.9 Creating reusable components in SketchUp

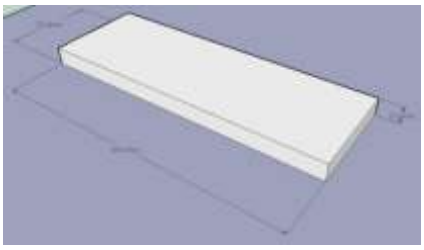
Creating a component that you can reuse in other SketchUp drawings is simple if you follow a few simple steps

1. You need a large surface on which to create the component. For example, if we are to make a breadboard, create a flat horizontal surface larger than the breadboard to start with (e.g. 300 x 300mm).

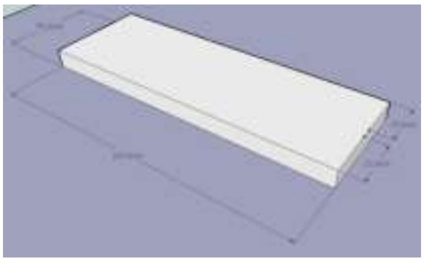




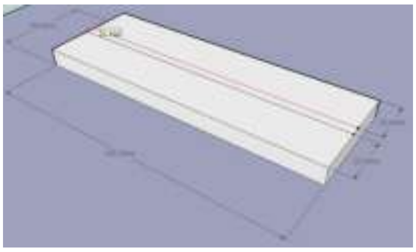
2. Create the base for the breadboard component (e.g. a rectangle 165 x 55 mm).
3. Extrude the breadboard 10mm.



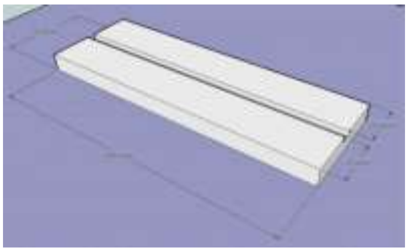
4. Use the TapeMeasure button to mark out the two points for the groove in the centre of the breadboard



5. Then draw two parallel lines.

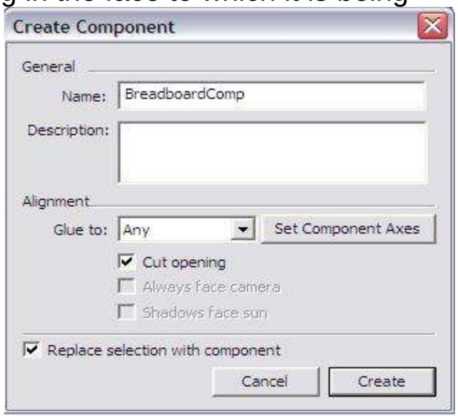


6. Extrude downwards 3 mm to make the slot

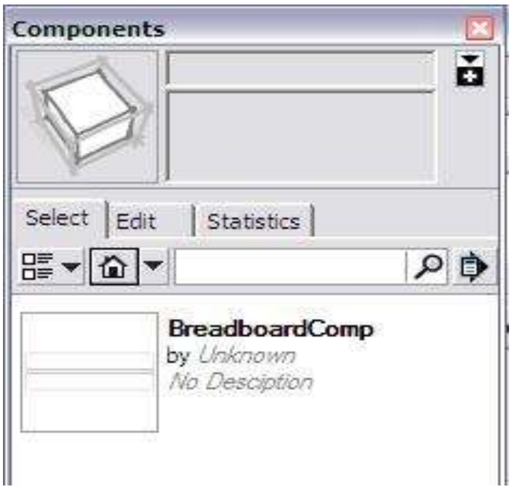


7. Select all of the entities you want to include in the component. Then right click and in the drop down menu select **Make Component**.

8. The Create Component dialog box opens:
   - **Name**. Type a name for the component.
   - **Description**. Optionally enter a description of the component.
   - **Glue to**. Select a glue-to alignment. The most flexible choice for components you want to glue is "Any."
   - **Cut opening**. Select this if you want the component to cut an opening in the face to which it is being glued. For example, you would typically use this option for a window.



9. You need to view the components in your model. From the menu select Window then click Components. In the Components window click the "In Model" button (little house),



10. In the components window right click the component and save it somewhere you can find it again.

Adding a component to another drawing:

1. In the new SketchUp drawing
2. From the menu choose File then Import
3. Select the component you want to import
4. It should 'glue' onto faces of your model.

14 Basic programming logic

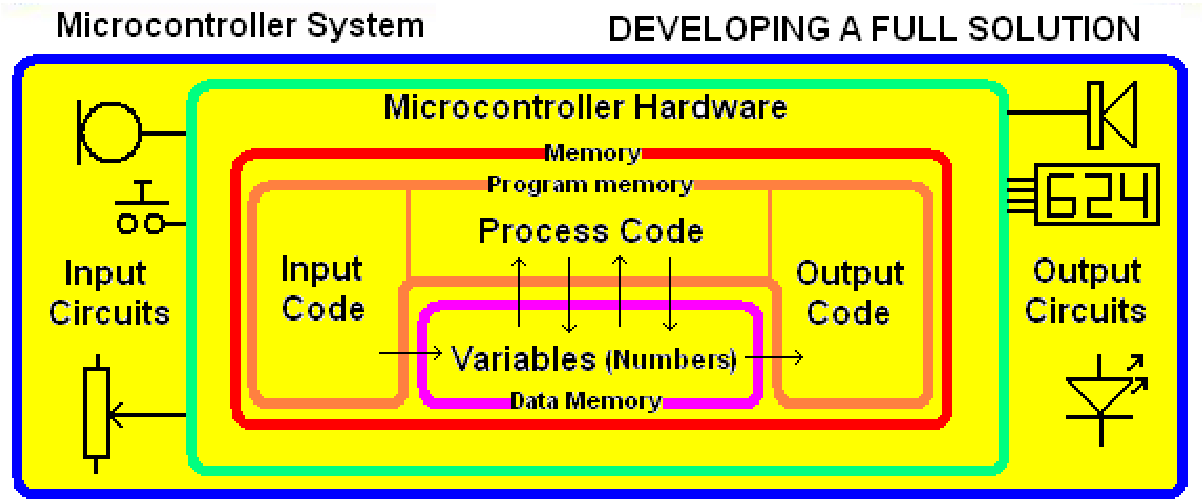
Using our knowledge of programming so far we can create a quiz game controller.
We have some important specifications we need to meet with this program.

Specifications
When the user presses their button, there is a short beep and all the other users are locked out until the reset button is pressed

14.1 Quiz Game Controller

In this program we will cover everything that has been learnt so far, make sure you understand throroughly everything that is going on.
We will use

- Input circuits
- Output circuits
- Input code
- Output code
- Variables
- Process code

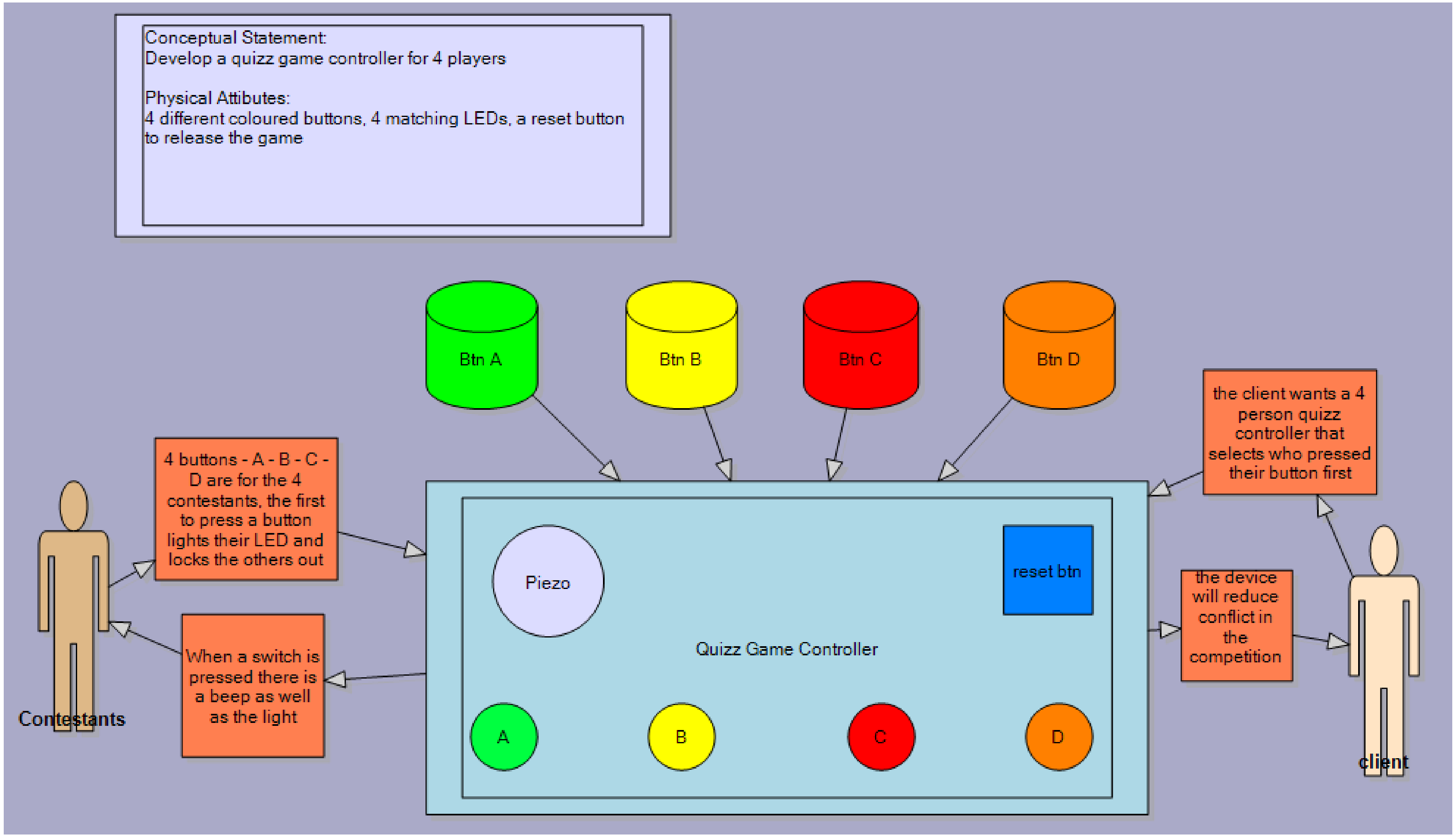


In this program we will use the concept of

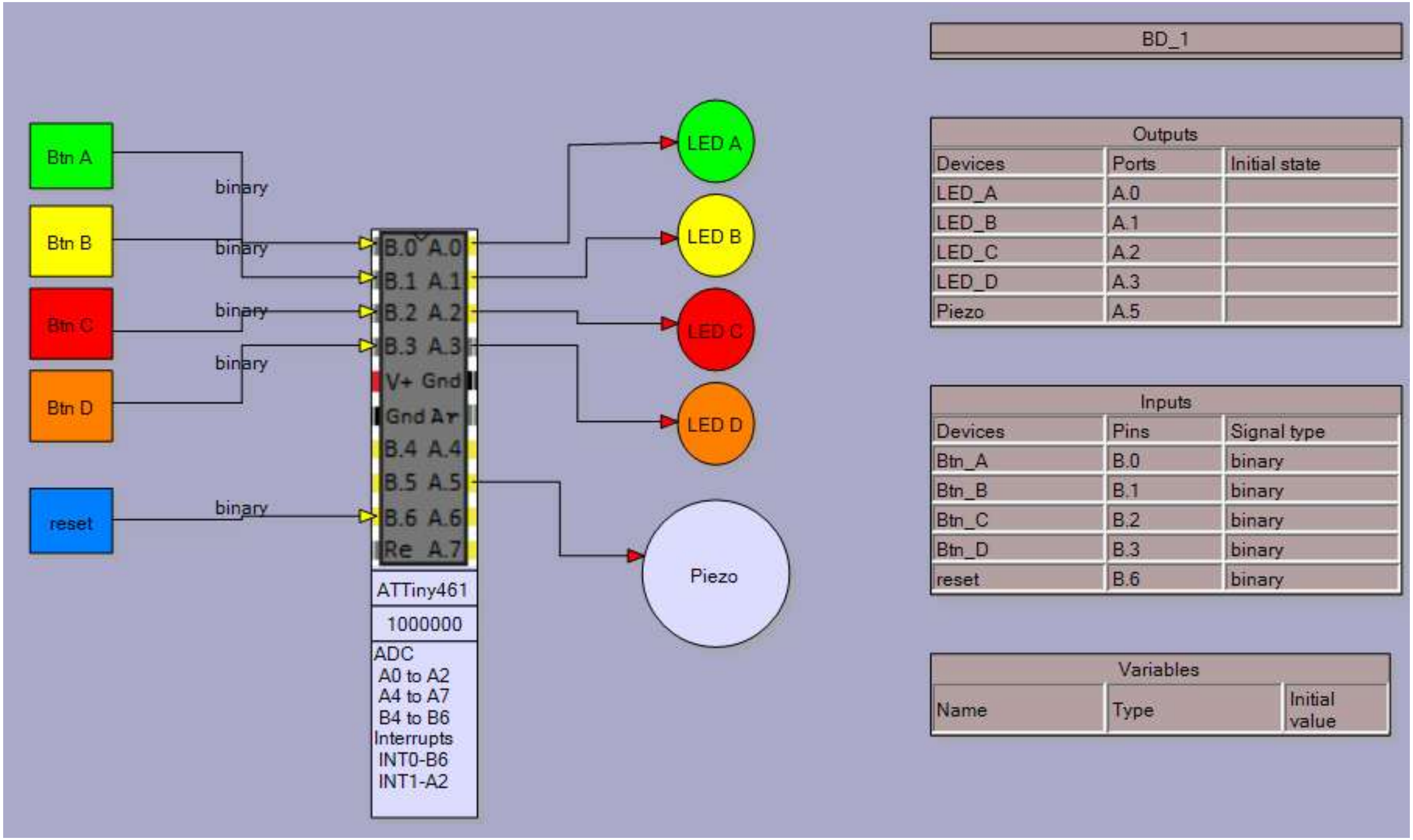
Do
Loop unitl ….

to make the program lock out other users and wait for the quiz master to press the reset button.
Here is the full project including using veroboard as a prototyping tool.

14.2 Quiz game controller system context diagram



14.3 Quiz game controller block diagram



Outputs		
Devices	Ports	Initial state
LED_A	A.0	
LED_B	A.1	
LED_C	A.2	
LED_D	A.3	
Piezo	A.5	

Inputs		
Devices	Pins	Signal type
Btn_A	B.0	binary
Btn_B	B.1	binary
Btn_C	B.2	binary
Btn_D	B.3	binary
reset	B.6	binary

Variables		
Name	Type	Initial value

14.4 Quiz game controller Algorithm

Note the addition to the variables table, we will need to store data in the program, the winner of the round.

An algorithm describes the operation of the system in terms of its
interactions with the world and
its internal functions.

Describe what happens to output devices or variables when an input subsytem or variable changes

A lot of detail is not required here, this is a 'big picture' understanding of how your device functions and is operated by the user

If a player presses their switch their LED goes on
And a beep sounds
And all the other players are locked out
Unitl the reset button is pressed

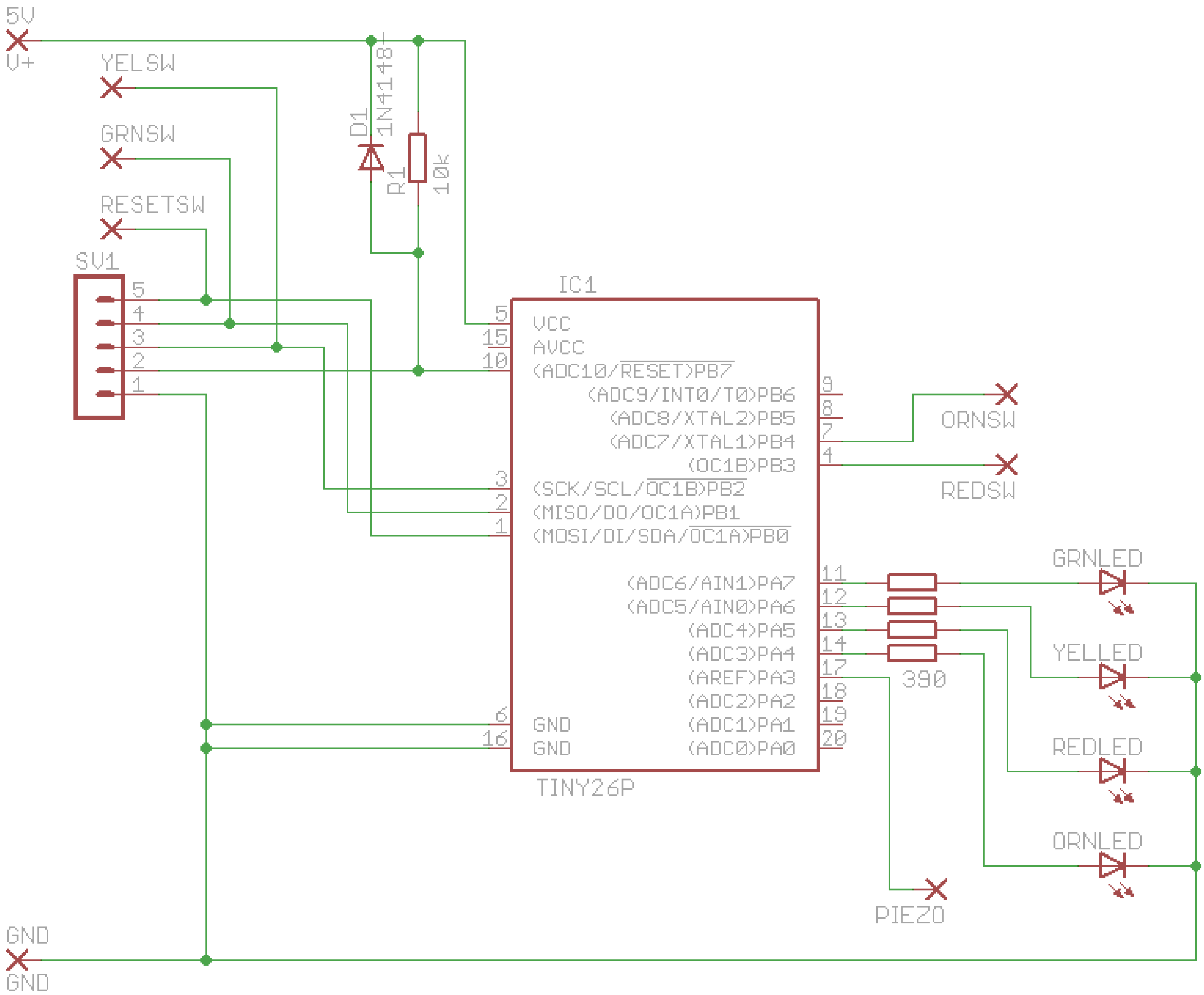
BD_1

Outputs		
Devices	Ports	Initial state
LED_A	A.0	
LED_B	A.1	
LED_C	A.2	
LED_D	A.3	
Piezo	A.5	

Inputs		
Devices	Pins	Signal type
Btn_A	B.0	binary
Btn_B	B.1	binary
Btn_C	B.2	binary
Btn_D	B.3	binary
reset	B.6	binary

Variables		
Name	Type	Initial Value
Winner	BYTE	0

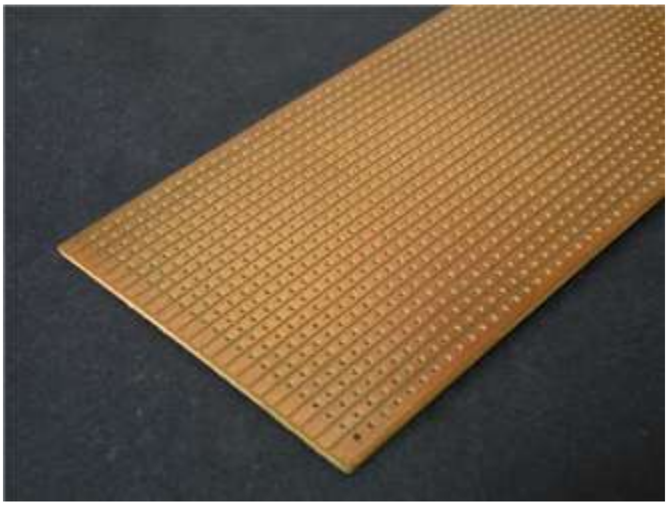
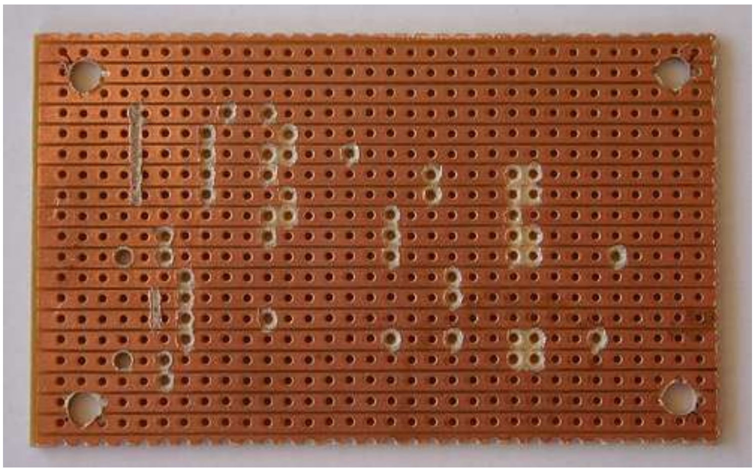
14.5 Quiz game schematic



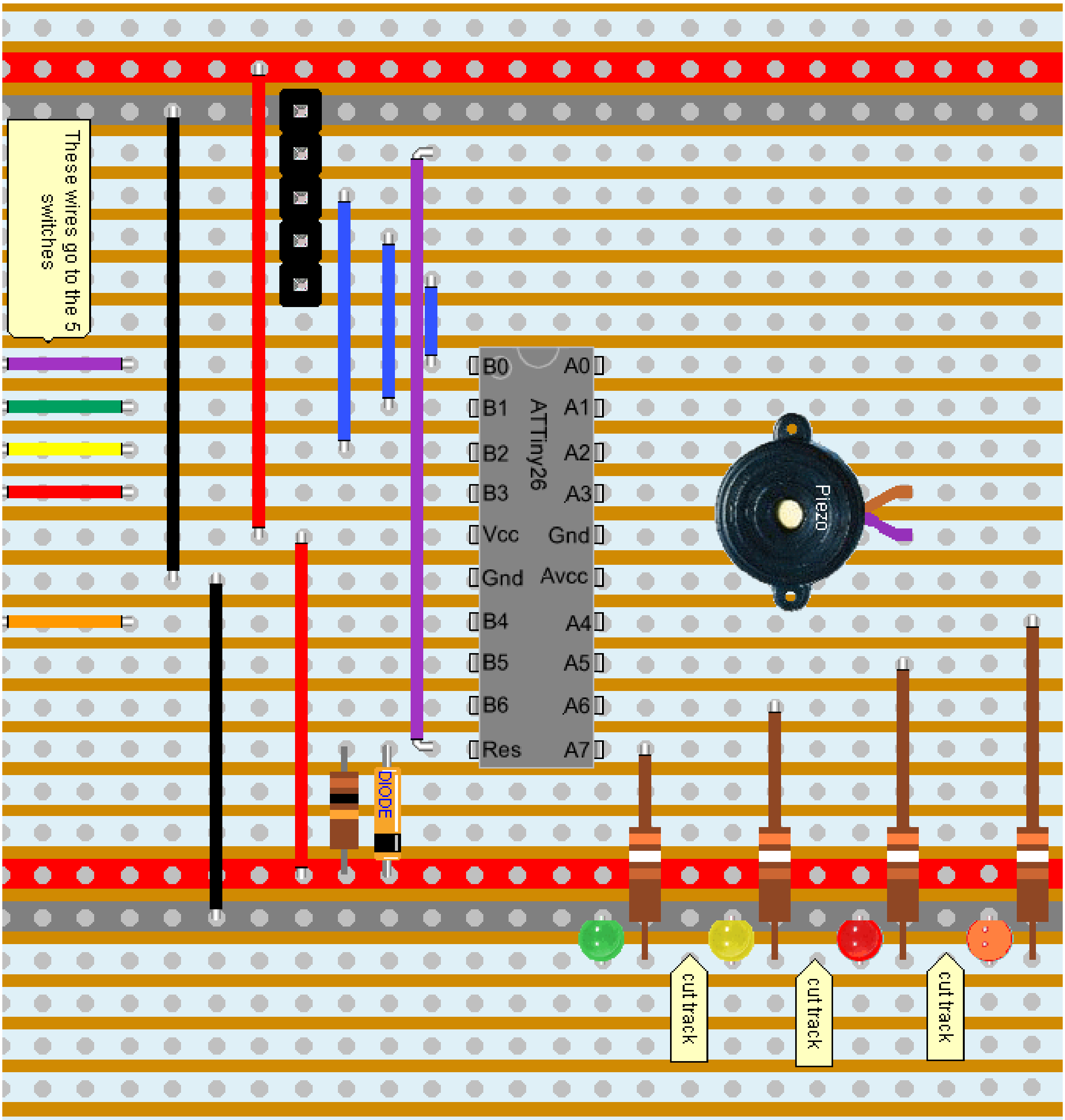
The circuit for the device has been drawn in eagle. The decisions about where to connect the LEDs and switches are not really important, but do take note that three of the switches are connected to the pins used for programming. This means that while the programming cable is connected it may interfere with the correct operation of the program.

14.6 Quiz game board veroboard layout

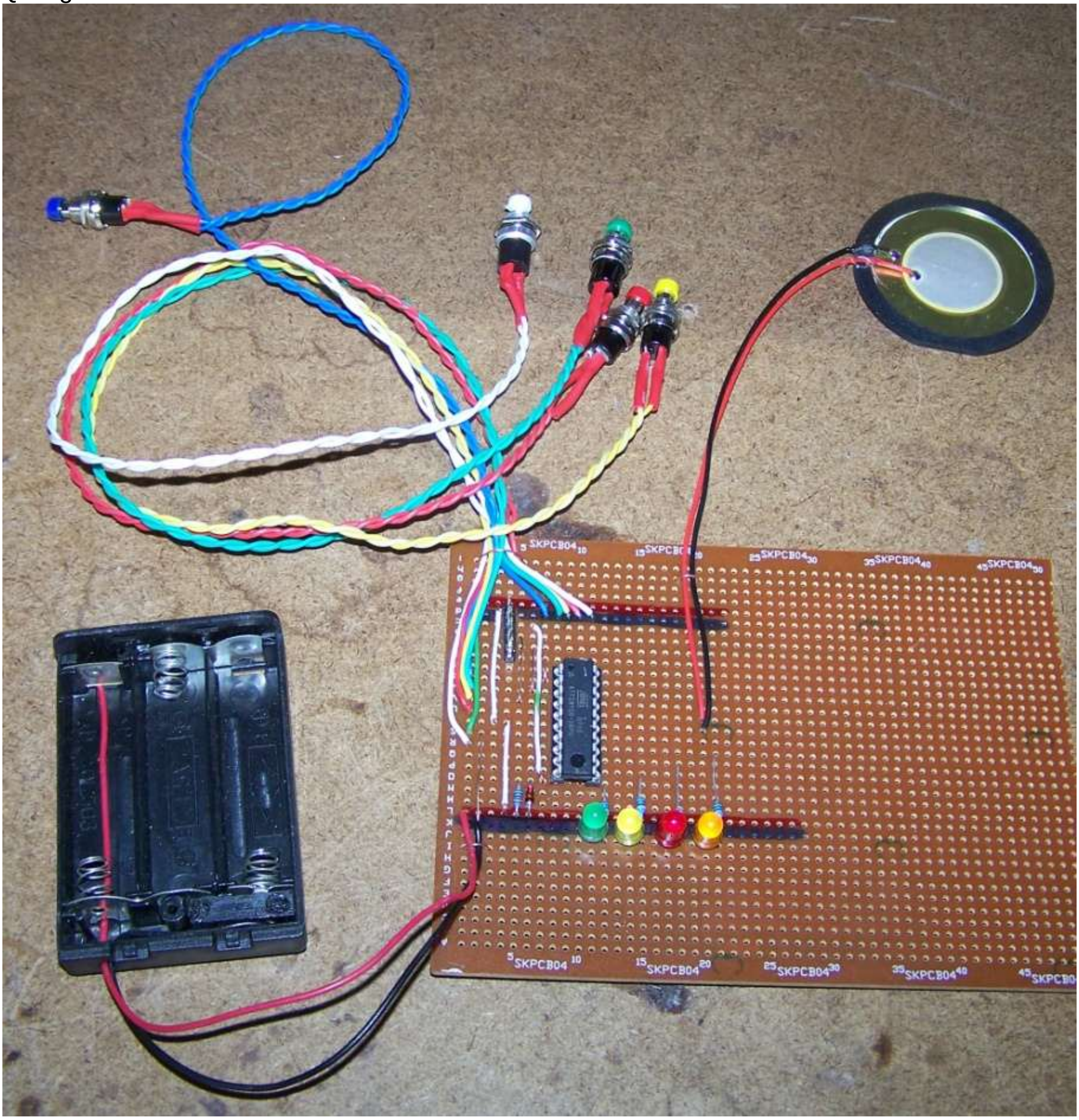
It was decided to use veroboard for the circuit rather than design a PCB. Veroboard or strip board is a highly useful pcb for prototyping one off circuits. As per the picture(below left) it is a predrilled board with tracks at 0.1 inch spacing so DIP IC packages and sockets fit exactly. The copper tracks will occasionally need to be cut in certain places. The board (below right) shows where cuts have been made using a drill bit. Don't use an electric drill just turn the bit by hand so that you cut through the copper track and not the board. I have a 4.5mm drill bit with some tape around it so that I don't cut my fingers while using it.

Plan the layout of vero board first

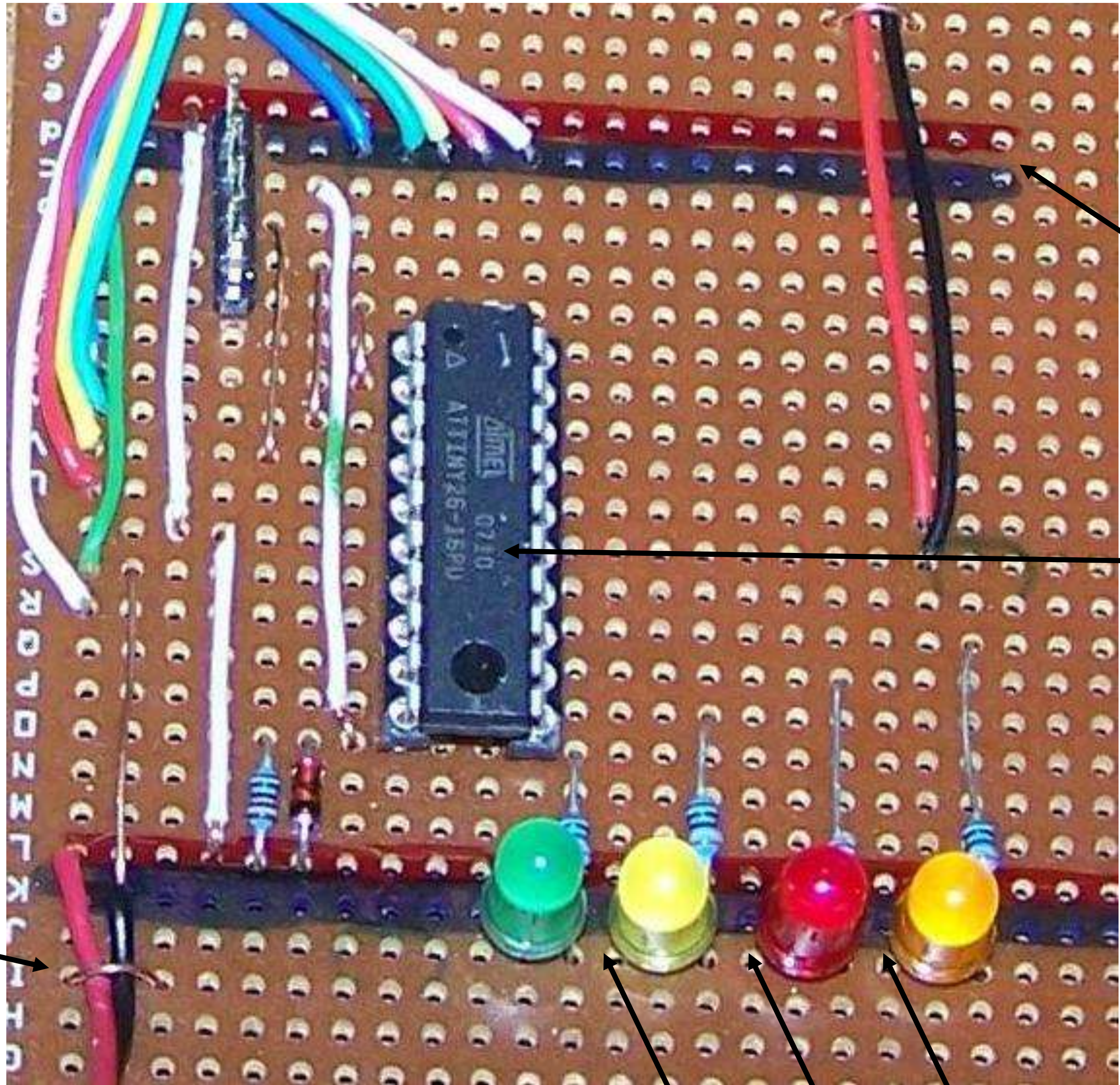


Quiz game Veroboard



Points of note when using veroboard

When I start laying out veroboard for a project I first plan it using either software or I place as many of the components as possible onto the board first before I start cutting any tracks so I can move them around before commiting to my design.



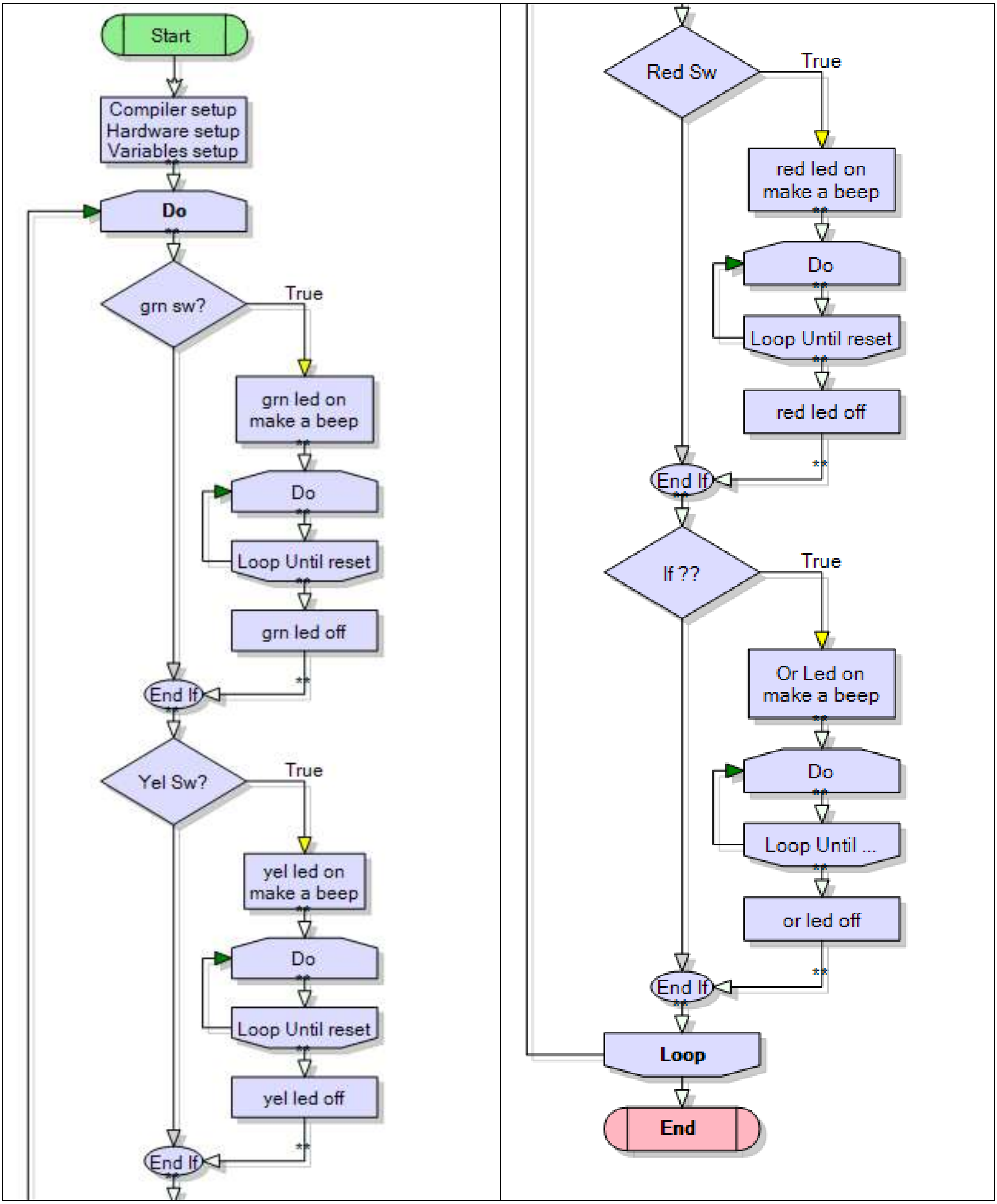
The board power supply lines have been coloured in red and black to make design easier

Remember to cut all of the 10 tracks under the IC so that the pins don't short out!

A loop of wire soldered onto the board acts as stress relief for the wires going off board to components such as the battery, switches and piezo

Remember to cut the tracks between the LEDs so they don't short cirucuit

14.7 Quiz Controller flowchart



14.8 'Quiz Controller program code

```
'compiler setup
$crystal = 1000000
$regfile = "attiny26.dat"

'microcontroller setup
Config Porta = Output
Config Portb = Input

'hardware aliases
Grnled Alias Porta.7                        'use port for output
Yelled Alias Porta.6
Redled Alias Porta.5
Ornled Alias Porta.4
Piezo Alias Porta.3
Resetsw Alias Pinb.0
Grnsw Alias Pinb.1                          'use pin for input
Yelsw Alias Pinb.2
Redsw Alias Pinb.3
Ornsw Alias Pinb.4
Set Portb.0                                 'activate pullup resistors
Set Portb.1                                 'for the 5 switches
Set Portb.2
Set Portb.3
Set Portb.4

'a simple test pattern on powerup on the leds to show they work
Set Grnled
Waitms 100
Set Yelled
Waitms 100
Set Redled
Waitms 100
Set Ornled
Waitms 100
Sound Piezo , 90 , 200
Waitms 100
Sound Piezo , 90 , 200
Waitms 1000


'-------------------------------------------------------------------
' Declare Variables
Dim Winner As Byte
'-------------------------------------------------------------------
```

```
'program starts here
Do
   Winner = 0                              'reset the winner flag
   Do
      If Grnsw = 0 Then
         Set Grnled
         Sound Piezo , 90 , 200            'make a beep
         Do                            'stay here until reset pressed
         Loop Until Resetsw = 0
         Reset Grnled
      Elseif Yelsw = 0 Then                'its important to use an elseif
         Set Yelled             'rather than separate if statements
         Sound Piezo , 90 , 200            'make a beep
         Do                            'stay here until reset pressed
         Loop Until Resetsw = 0
         Reset Yelled
      Elseif Redsw = 0 Then
         Set Redled
         Sound Piezo , 90 , 200            'make a beep
         Do                            'stay here until reset pressed
         Loop Until Resetsw = 0
         Reset Redled
      Elseif Ornsw = 0 Then
         Set Ornled
         Sound Piezo , 90 , 200            'make a beep
         Do                            'stay here until reset pressed
         Loop Until Resetsw = 0
         Reset Ornled
      End If
Loop
End


'note you could add other features to the device such as:
' having a different number of beeps for each player
' have some indication that the device is on as normally there are no
LEDs lit,
' add a timing fucntion that gives players a fixed number of seconds to
answer
' a counter that tracks how often each person has won
' ...
```

14.9 Don't delay - use logic

Delays such as wait and waitms can become real headaches in longer or complex programs, it is vital to start to learn how **not** to use them! Here is the do-loop.

Although they are both looping structures the do-loop is significantly different to the for-next; as they can be used very differently when programming. With a for-next we repeat something a fixed number of times, and we know the number of times before the loop starts. With a do-loop we are repeating something a number of times that is unknown at the time we start the loop.

Take the example of hammering a nail

E.g. in real life we don't say hammer the nail 5 times, we say hammer the nail UNTIL IT IS IN

```
Do
    Gosub hammer_nail
Loop until nail_height = flat_in_wall
```

The do-loop is similar to the for-next however in the do-loop we have to remember to write the code to clear the variable <u>everytime</u> we start the loop (count=0) and increment the variable (incr count).

Here is the siren code rewritten using do-loops so you can see how to structure it.

<table>
<tr>
<td><pre>
Siren:
    For count = 0 to Maxcount1
        Waitus Halfperioddelay1
        Set Piezo
        Waitus Halfperioddelay1
        Reset Piezo
    Next

    For count = 0 to Maxcount2
        Waitus Halfperioddelay2
        Set Piezo
        Waitus Halfperioddelay2
        Reset Piezo
    Next
Return
</pre></td>
<td><pre>
Siren:
    count=0
    Do
        Waitus Halfperioddelay1
        Set Piezo
        Waitus Halfperioddelay1
        Reset Piezo
        Incr count
    Loop until count = Maxcount1
    count=0
    Do
        Waitus Halfperioddelay2
        Set Piezo
        Waitus Halfperioddelay2
        Reset Piezo
        Incr count
    Loop until count = Maxcount2
Return
</pre></td>
</tr>
</table>

Here we aren't using the do-loop any differently to the for-next I am only showing you how to write the code properly.

Sometimes in a program we want to repeat something, but we don't know how many times it has to be repeated, we just wait or do something until we are told to move on.

e.g. Do
    Loop Until clear_sw=0

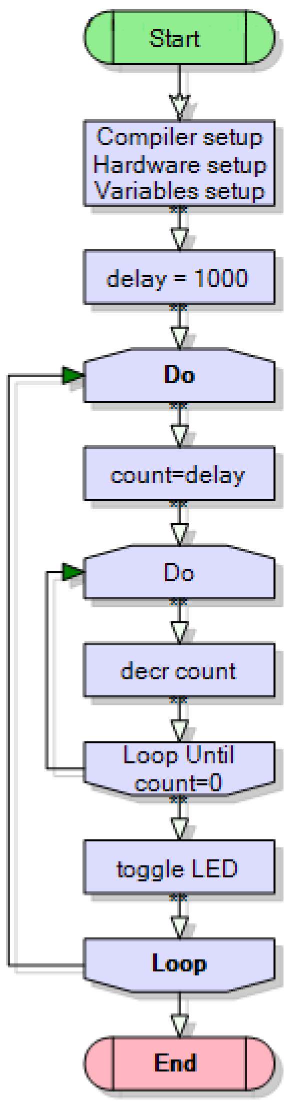
In this case the length of time we are waiting is unknow as we are waiting for a user.

But in a program we may have to wait for some calculation to complete

e.g. Do
        gosub wash_clothes
        gosub rinse_clothes
        gosub measure_water_mirkiness
    Loop Until water_mirkiness < 10

What is the point of washing clothes 100 times, when they might only need 50 or they might actually need 200 so we wash the number of times it takes for the clothes to be clean.
We will use do-loop like this in the next solutions.

Now back to the delay issue. To begin to solve the issue you should understand that a delay routine in a program is simply a loop that repeats a large number of times e.g.in this loop we are using our own counter to keep track of the time. We start it at 1000 and then decrement its value until it gets to 0 then we toggle the LED.



If this loop takes approximately 2 uSec (microseconds) to complete and does it 1000 times then it will give a delay of 2 mSec

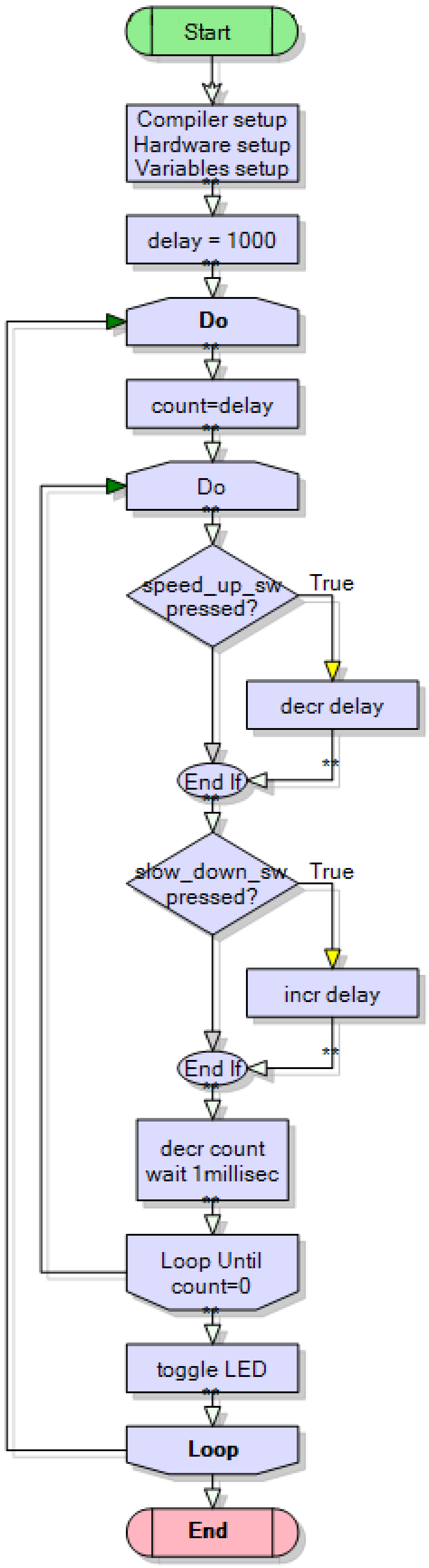
How many times would the loop have to repeat to delay:

1mS ?
10mS ?
1 Second ?
1 Minute ?

In some programs it is acceptable to put in a very small delay, in other programs it is not. You must start to think through the consequences of putting a delay within your specific program.

At this stage we are working on simple programs so we can see the consequences of a small delay. In big programs the consequences of delays can be very hard to fix!

Here is a way of speeding up or slowing down the rate at which an LED is flashing.
A variable is used to count 1mS delays. We can use 1mS delays because when a user presses a switch they will always press it for longer than 1mS. Now we add to the program the ability for thew user to prss a switch to change the value of the delay, therefore making the flashing rate shorter or longer.




```
Dim count as word
Dim delay as word

Delay = 1000

do
    count=delay
    do
        if swa=0 then decr delay
        if swb=0 then incr delay
        waitms 1
        decr count
    loop until count = 0
    toggle led
loop
```

Note that we need to keep 2 variables, one is DELAY which we increase and decrease using the switches. The other is a temporary copy of it COUNT which is decremented within the loops.

Although the main problem is fixed there are some other problems to fix:

1. There are debounce issues
2. When you keep incrementing delay eventually it will get to 65535, and another increment will cause it to roll over or **overflow** back to 0 (an If-then would help you)/
3. Also when delaycount gets down to 0, another decrement will cause it to **underflow** to 65535!(another if-then would fix this)
4. The resolution (degree of change) of our delay program is not very good if we increase or decrease each time by one. Perhaps a bigger increment/decrement value might be more useful (instead of incr delay we could use delay=delay +50).

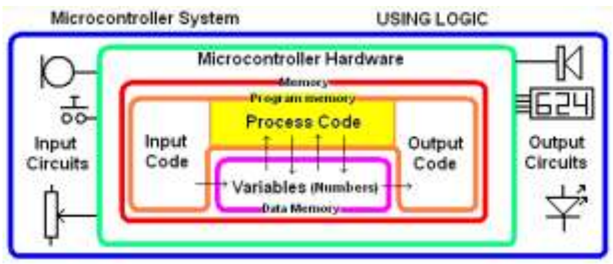
15 Algorithm development – an alarm system

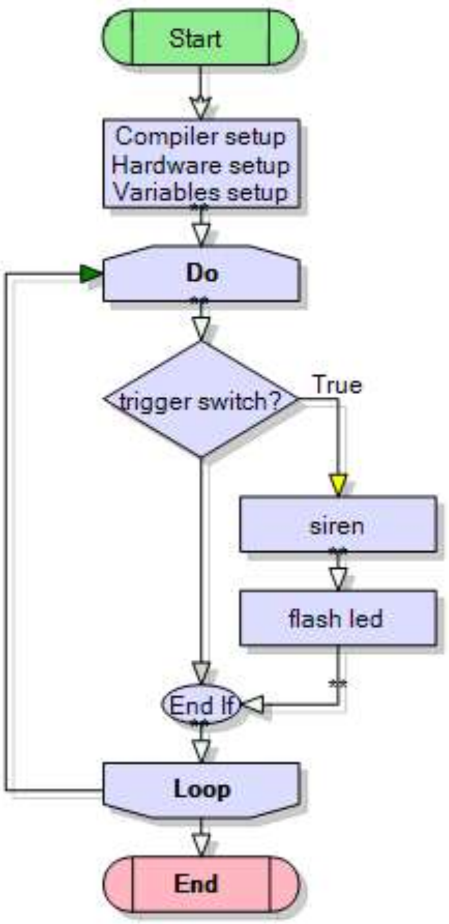
When learning to program students find it straight forward to write programs which contain one simple process and which require a few lines of code; however you must move on to the next level and this requires learning about another way of thinking called algorithmic thinking. This is seeing a problem as an ordered and organised process of steps. Because of their growing knowledge of computer syntax students generally begin programming at the keyboard rather than with thinking through a problem and using a pen and paper to organise their program. Programs become confused very quickly in this situation.

Note that with technological practice (at all levels) students are required to plan, trial and test ideas. So when writing software students must not write software without spending time planning it first AND keep a record of their work.

In these next examples instead of presenting a final prototype the process of development is produced from the very simple to the more complex (as complex as we will go with flowcharts). The process of development of a program should be incremental – don't try and do everything in one program all at once. All that does is produce loads of errors and even if you fix the errors the software probably wont work!

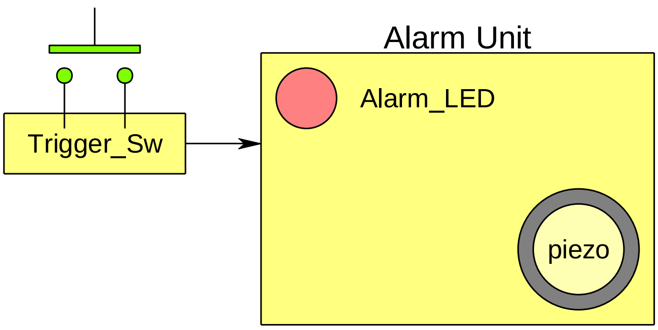
15.1   Simple alarm system – stage 1



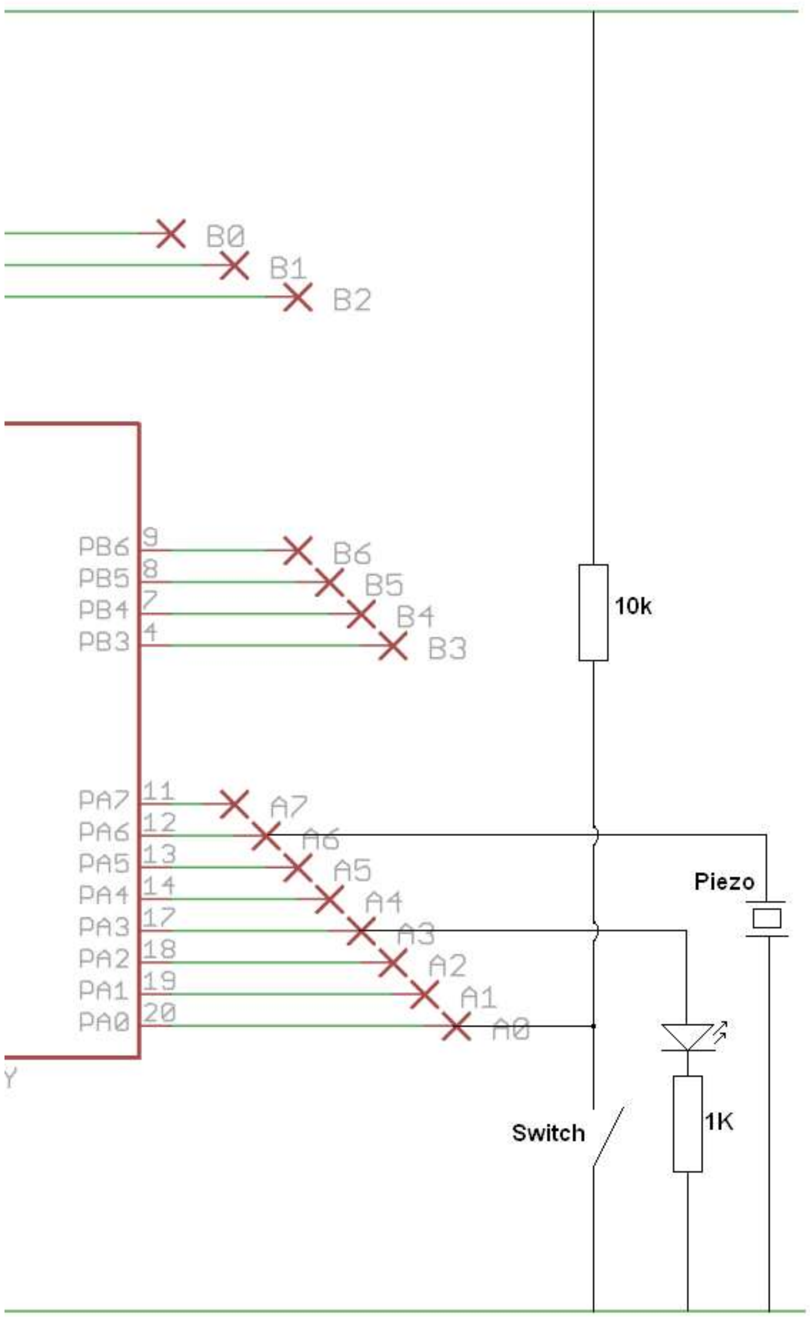


Here is a very simple alarm. When the trigger switch is pressed the LED flashes and it makes a siren (using our siren subroutine from the previous programs)

In this first alarm the alarm only sounds while the switch is pressed



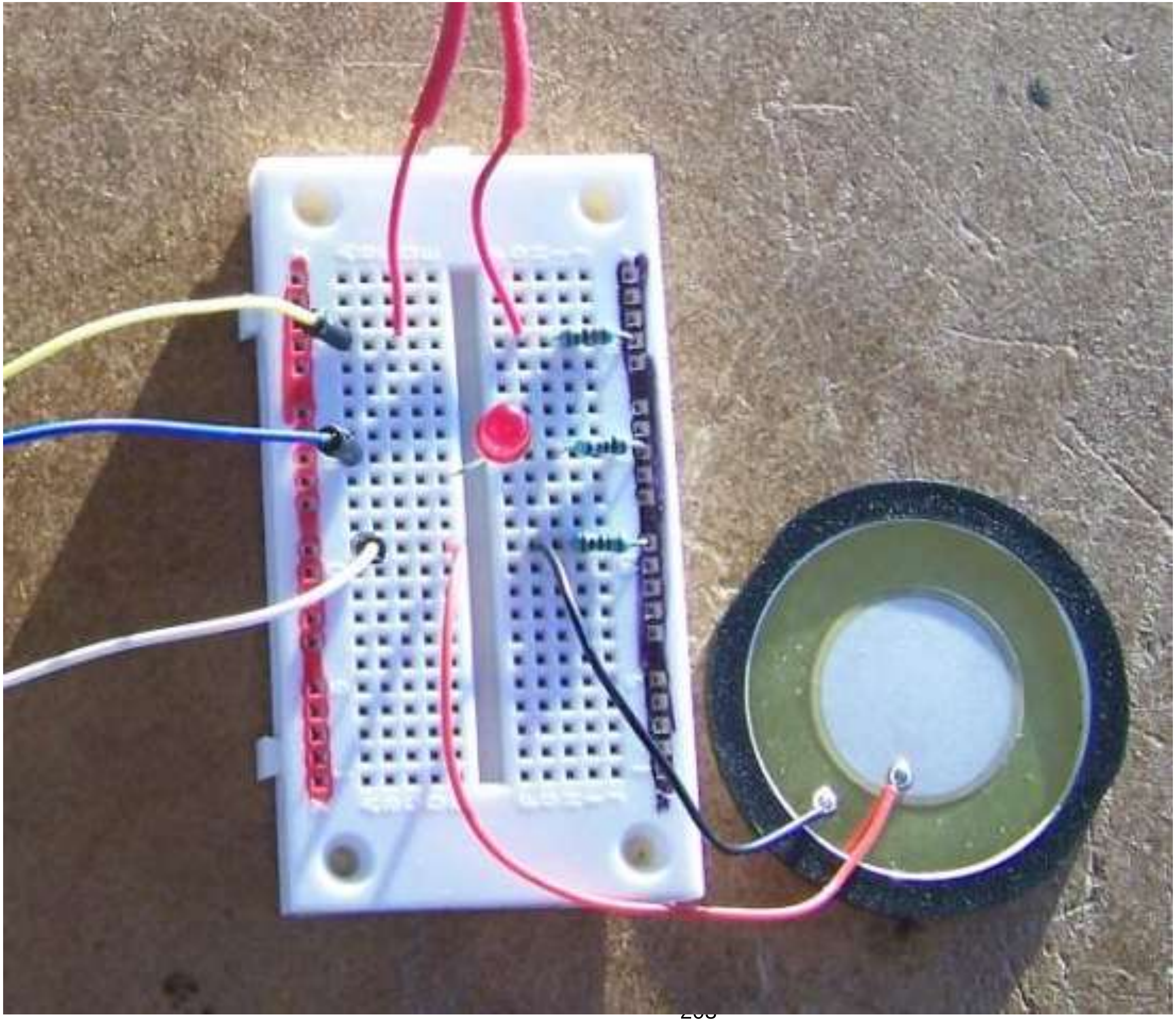
15.2 Alarm System Schematic

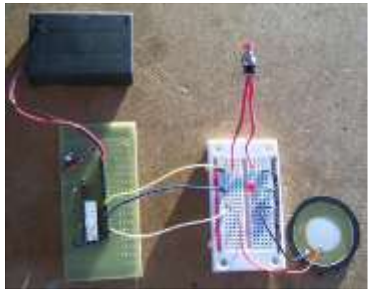


Note that the connections.
Piezo on portA.5
LED on portA.3
Switch on pina.0
NOTE THE NAMES
PORT for outputs
PIN for inputs

The next thing to do is to record the configurations for the I/O devices.

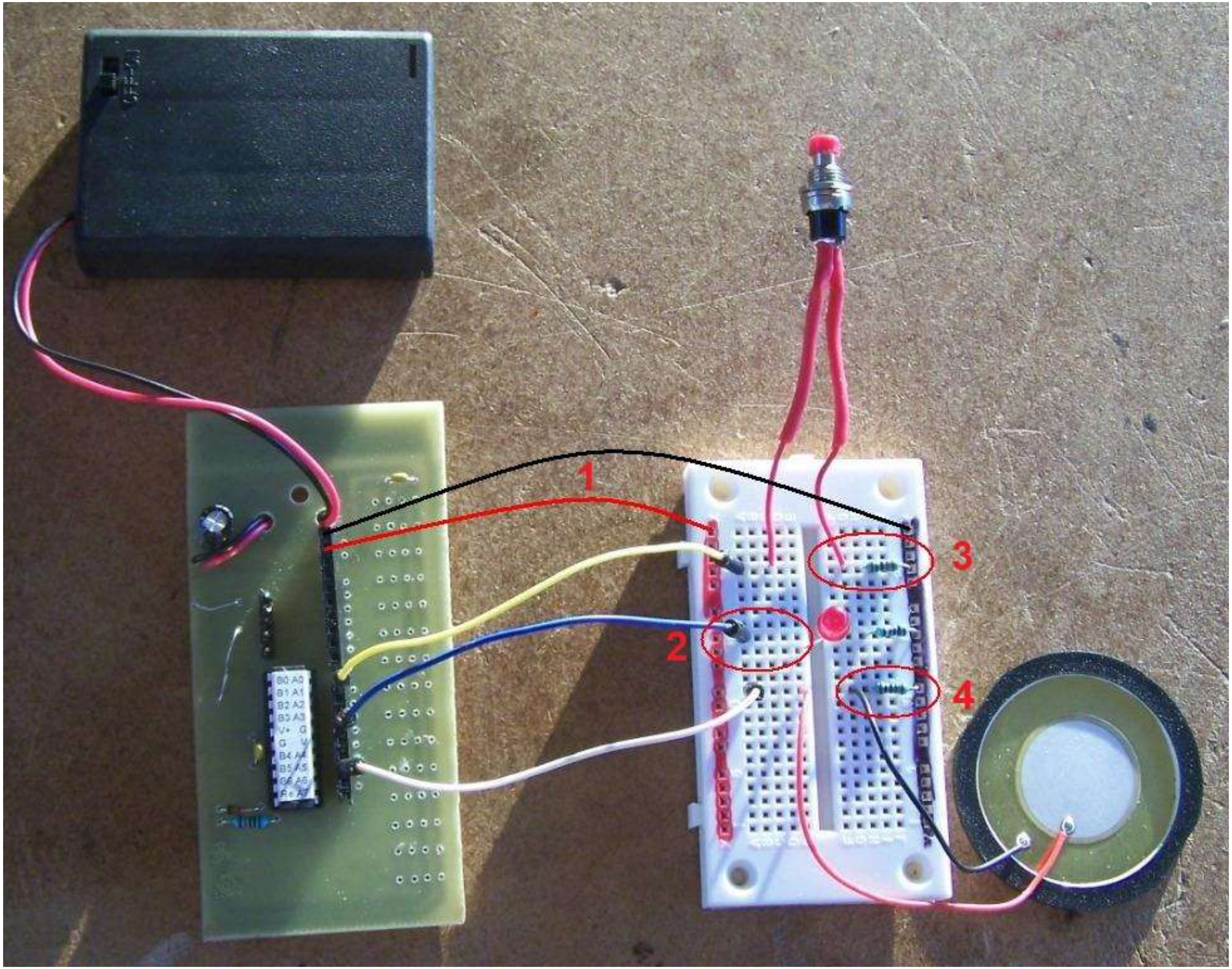
```
Config Porta = Output
Config Pina.0 = Input

Trigger_sw Alias Pina.0
Alarm_led Alias Porta.3
Piezo Alias Porta.6
```





Here is one INCORRECT attempt at wiring up the circuit

There are several problems with the wiring; how many can you spot?



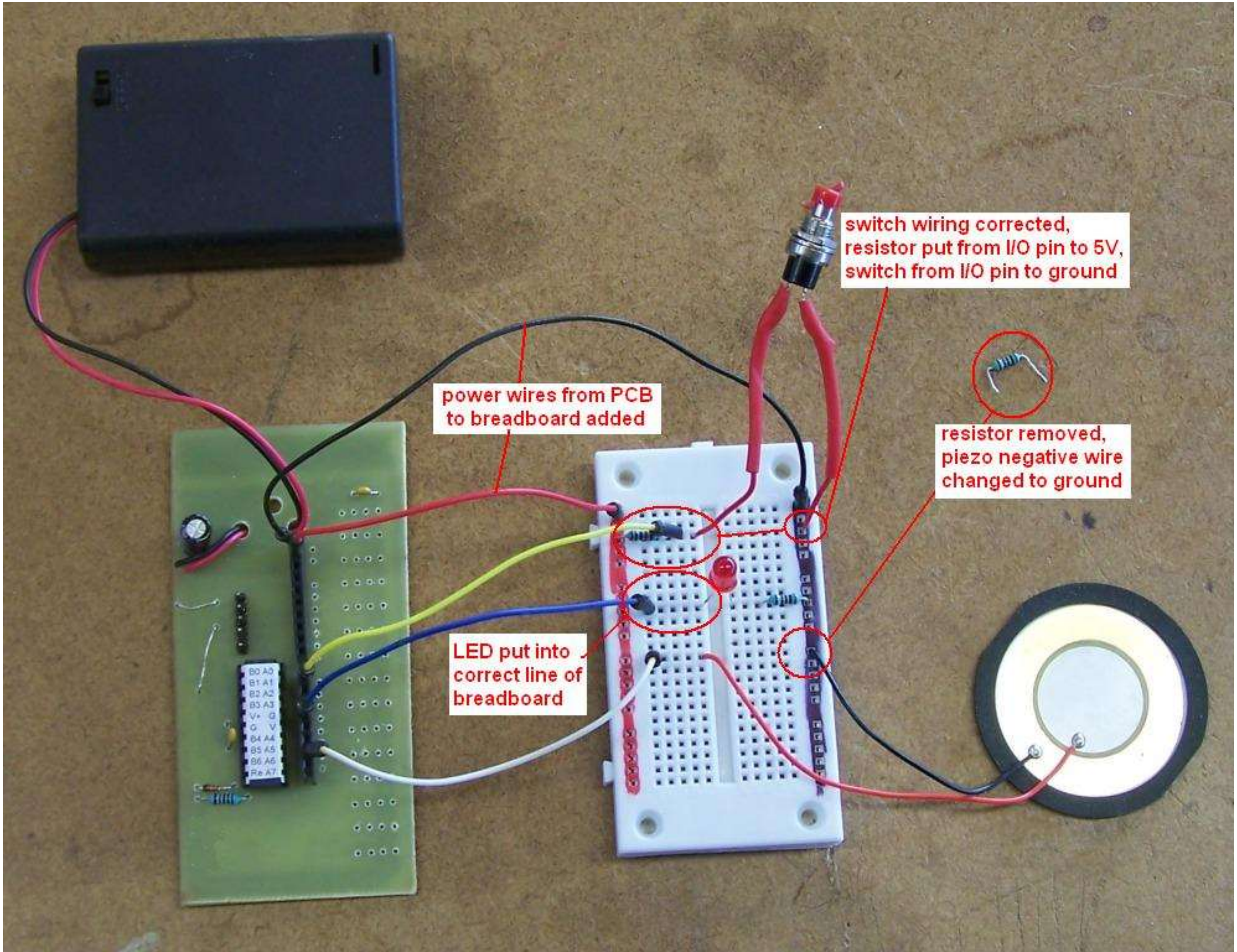
Problems:

1. forgotten the red and black power wires to the breadboard.

2. the LED and resistor dont link on the breadboard.

3. the switch wring is quite incorrect.

4. there is a resistor in series with the piezo.


switch wiring corrected,
resistor put from I/O pin to 5V,
switch from I/O pin to ground
power wires from PCB
to breadboard added
resistor removed,
piezo negative wire
changed to ground
LED put into
correct line of
breadboard

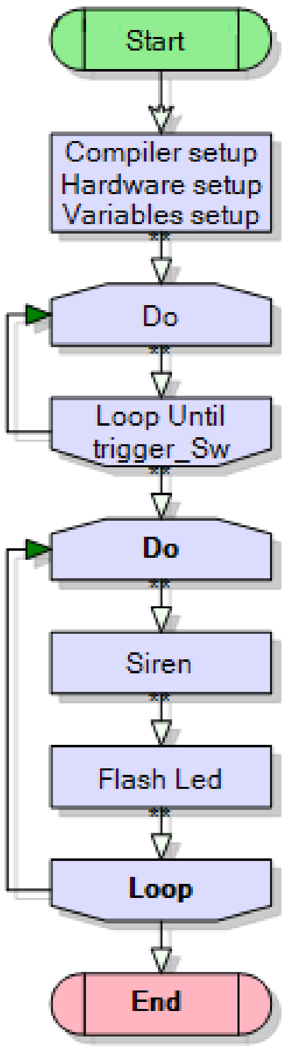
```
'B Collis    2009
'file: ALARM_1.BAS
$regfile = "attiny26.dat"
$crystal = 1000000                          'rate of executing code
Config Porta = Output
Config Pina.0 = Input

Trigger_sw Alias Pina.0                     'white switch
Alarm_led Alias Porta.3
Piezo Alias Porta.6               'use useful name PIEZO not PORTb.3

Const Flashdelay = 50

Const Halfperioddelay1 = 200                ' first tone 1/2 period
Const Halfperioddelay2 = 500                ' second tone 1/2 period
Const Maxcyclecount1 = 350                  'length of first tone
Const Maxcyclecount2 = 150                  'length of second tone

Dim Cyclecount As Word          'keep count of number of cycls(periods)
Dim Sirens As Byte

Do
     If Trigger_sw = 0 Then
         Gosub Siren_sound
         'flash the led rapidly
         Set Alarm_led
         Waitms 20
         Reset Alarm_led
         Waitms 200
     End if
Loop
End

Siren_sound:
   For Cyclecount = 0 to Maxcyclecount1
      Waitus Halfperioddelay1
      Set Piezo
      Waitus Halfperioddelay1
      Reset Piezo
   Next
   For Cyclecount = 0 to Maxcyclecount2
      Waitus Halfperioddelay2
      Set Piezo
      Waitus Halfperioddelay2
      Reset Piezo
   Next
Return
```

Note how we have reused the software for the siren created earlier.

15.3 A simple alarm system – stage 2

In this second alarm the IF-THEN has been replaced by a DO-LOOP-UNTIL

It is a much tidier piece of code, replacing the If trigger_sw=0 with a do loop until separates the two concepts of waiting for the switch and what happens after it is pressed. This reduces the complexity of the main loop by a layer,

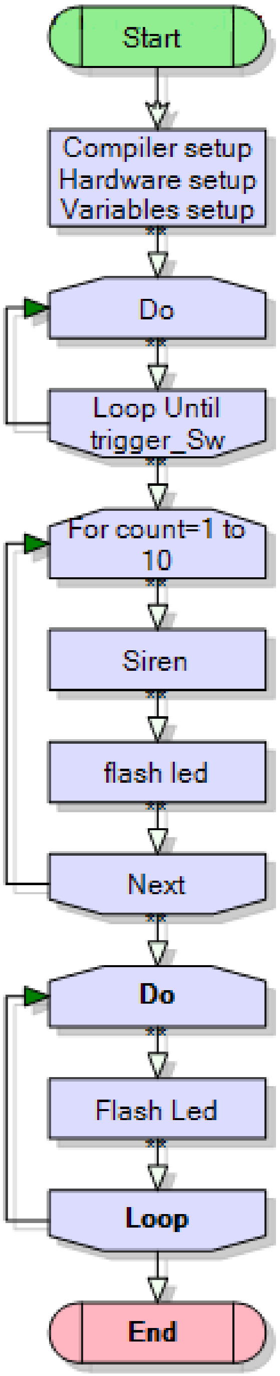



```
'B Collis    2009
'file: ALARM_2.BAS
$regfile = "attiny26.dat"
$crystal = 1000000                        'rate of
executing code
Config Porta = Output
Config Pina.0 = Input

Trigger_sw Alias Pina.0                   'white switch
Alarm_led Alias Porta.3
Piezo Alias Porta.6

Const Flashdelay = 50
Const Halfperioddelay1 = 200   ' first tone 1/2 period
Const Halfperioddelay2 = 500   second tone 1/2 period
Const Maxcyclecount1 = 350     'length of first tone
Const Maxcyclecount2 = 150     'length of second tone

Dim Cyclecount As Word 'keep count of number of cycles
Dim Sirens As Byte
'wait for trigger switch to be pressed
Do
Loop Until Trigger_sw = 0
Do
     Gosub Siren_sound

     'flash the led rapidly
     Set Alarm_led
     Waitms 20
     Reset Alarm_led
     Waitms 200
Loop
End
```

The problem with this bit of code is that the siren keeps going until the power to the circuit is turned off. This is not very satisfactory.

15.4 A simple alarm system – stage 3

In this version we the siren only goes 10 times and then the LED stays flashing.

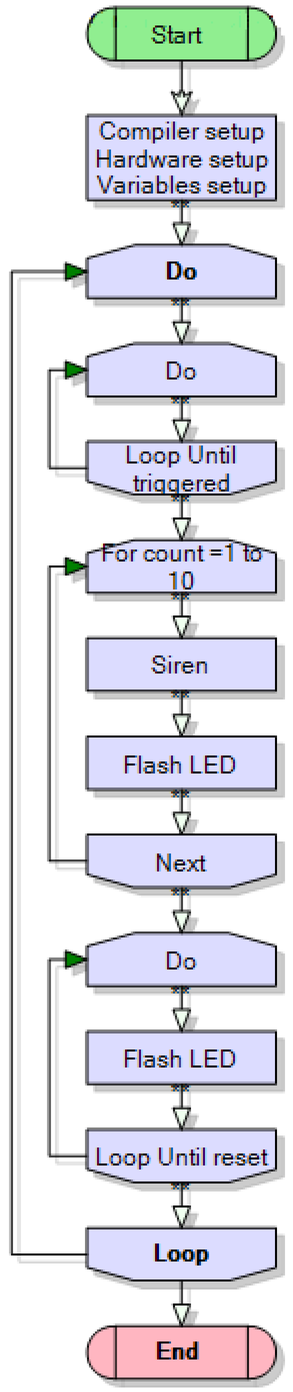


The problem with this is that there is no way to reset the system without removing the power from it. All the code realy needs to be inside the main do-loop.

```
'B Collis    2009
'file: ALARM_3.BAS
$regfile = "attiny26.dat"
$crystal = 1000000          'rate of executing code
Config Porta = Output
Config Pina.0 = Input

Trigger_sw Alias Pina.0                'white switch
Alarm_led Alias Porta.3
Piezo Alias Porta.6

Const Flashdelay = 50
Const Halfperioddelay1 = 200  ' first tone 1/2 period
Const Halfperioddelay2 = 500  ' second tone 1/2 period
Const Maxcyclecount1 = 350   'length of first tone
Const Maxcyclecount2 = 150   'length of second tone

Dim Cyclecount As Word 'keep count of number of cycles
Dim Count As Byte

'wait for trigger switch to be pressed
Do
Loop Until Trigger_sw = 0

For Count = 1 to 10
     Gosub Siren_sound

     'flash the led rapidly
     Set Alarm_led
     Waitms 20
     Reset Alarm_led
     Waitms 200
Next

Do
     'flash the led continuously
     Set Alarm_led
     Waitms 20
     Reset Alarm_led
     Waitms 200
Loop
End
```

15.5 A simple alarm system – stage 4

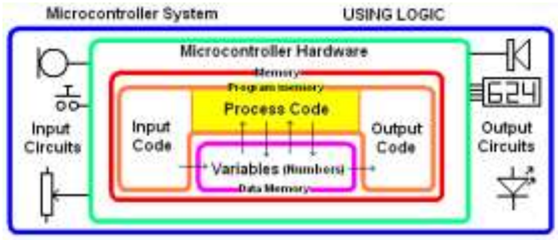
In the 4th version we add a second switch to reset the alarm.



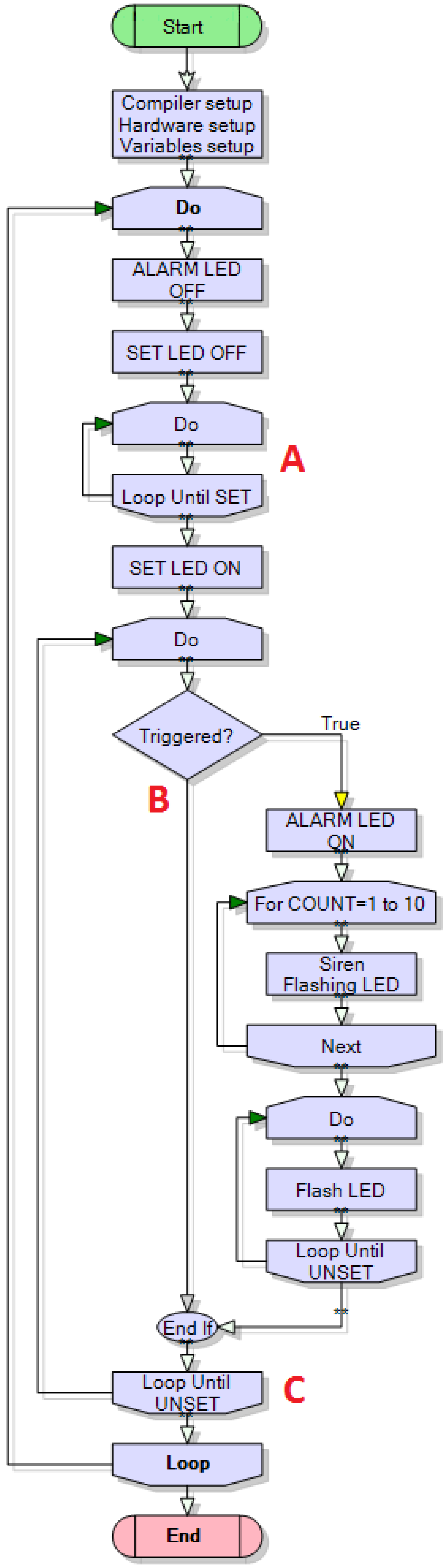
The problem with this stage of the alarm project is that the alarm is always on, there is no way to turn it on or off, apart from the power supply.

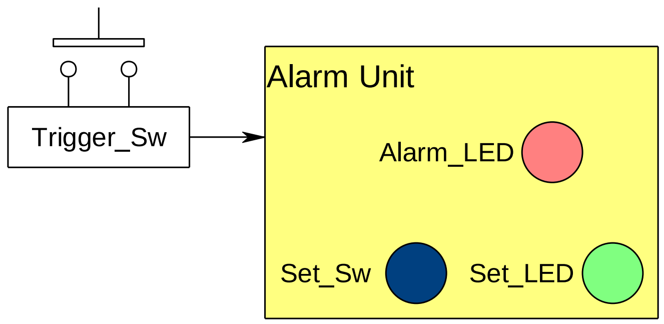
```
'B Collis     2009
'file: ALARM_4.BAS
$regfile = "attiny26.dat"
$crystal = 1000000
Config Porta = Output
Config Pina.0 = Input
Config Pina.1 = Input

Trigger_sw Alias Pina.0               'my white switch
Reset_sw Alias Pina.1                 'my green switch
Alarm_led Alias Porta.3
Piezo Alias Porta.6

Const Flashdelay = 50
Const Halfperioddelay1 = 200 ' first tone 1/2 period
Const Halfperioddelay2 = 500 ' second tone 1/2 period
Const Maxcyclecount1 = 350     'length of first tone
Const Maxcyclecount2 = 150   'length of second tone

Dim Cyclecount As Word 'keep count of cycles
Dim Count As Byte
Do
    'wait for trigger switch to be pressed
    Do
    Loop Until Trigger_sw = 0

    For Count = 1 to 10
       Gosub Siren_sound

       'flash the led rapidly
       Set Alarm_led
       Waitms 20
       Reset Alarm_led
       Waitms 200
    Next
    Do
    'flash the led until the reset button is pressed
       Set Alarm_led
       Waitms 20
       Reset Alarm_led
       Waitms 200
    Loop Until Reset_sw = 0
Loop   'return to the start
```

15.6 More complex alarm system



Program for a more sophisticated alarm unit, with 2 switches and 2 LEDs. In this alarm the reset switch has been replaced by a set switch which is used to activate and deactivate the alarm.





Alarm 5 system block diagram:

15.7 Alarm unit algorithm 5:

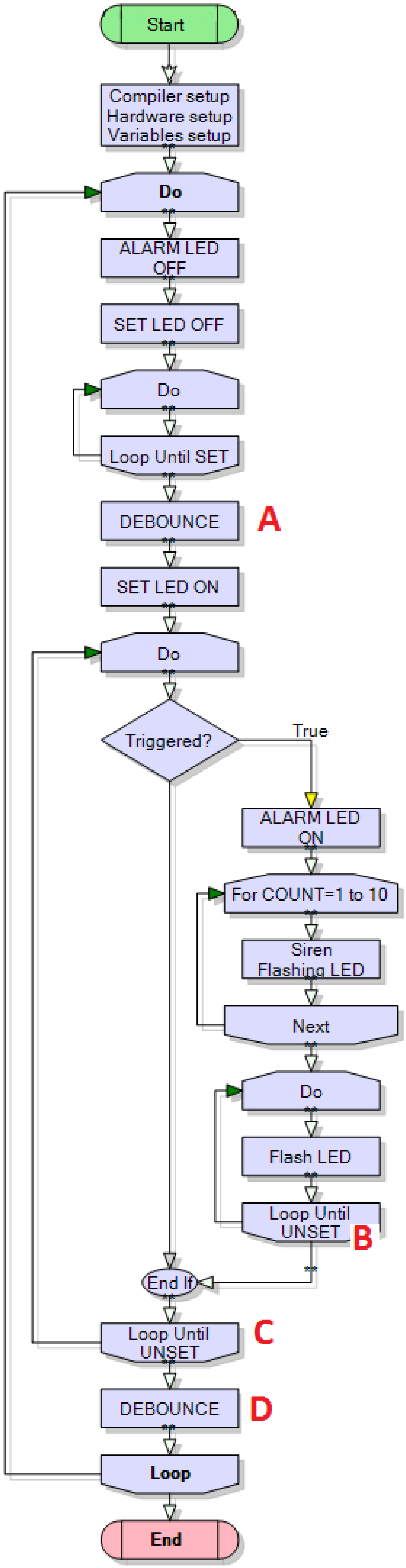
Initially the two LEDs are off

When SetSw is pressed
the program begins to monitor Trigger_Sw
and Set_LED comes on

If TriggerSw is detected Alarm_LED flashes
If SetSw is pressed Alarm_LED stops

PROBLEMS WITH THIS VERSION
When thinking through this after planning it it a problem was identified.

When the alarm is turned on it waits at point **A** for the SET switch to be pressed. When it is pressed the program continues on to point **B** where it checks the trigger switch, it is not triggered so it takes the path to the **loop until unset** at point C where it immediately exits the loop. This is caused by the program being carried out so fast. We need to add a debounce to the reset switch to fix this.
So this program is not developed any further but it is kept on file for an important reason. In technology education a record of trialling is essential to developing clear problem solving and leads to good grades.



15.8 Alarm 6 algorithm:

- Initially the two LEDs are off
- When Set_Sw is pressed and released A
- the program begins to monitor Trigger_Sw
- and the Set_LED comes on
- If Trigger_Sw is detected Alarm_LED flashes
- If Trigger_Sw is reset Alarm_LED keeps flashing
- If Set_Sw is pressed and released (D) the

Alarm_LED stops

NOTE: at point B there is no debounce, this is because we want the program to continue to sense the switch is pressed at point C and then wait for it to be released.

Now this is a complex piece of code and really we have gone justabout as far as we should with flowcharts. Later in the book there is another concept called state macines which is much easier for laregr programs!

```
'file: ALARM_6.BAS
'compiler setups
$regfile = "attiny26.dat"
$crystal = 1000000
'-------------------------------------------
'Hardware setups
Config Porta = Output
Config Pina.0 = Input
Config Pina.1 = Input
'-------------------------------------------
'Hardware Aliases
Trigger_sw Alias Pina.0      'my white switch
Set_sw Alias Pina.1          'my green switch
Alarm_led Alias Porta.3
Set_led Alias Porta.4
Piezo Alias Porta.6
'use useful name PIEZO not PORTb.3
'-------------------------------------------
'Variables
Dim Count As Byte
Dim Cyclecount As Word
'keep count of number of cycles
'-------------------------------------------
'Constants
Const Flashdelay = 50
Const Debouncedelay = 30
Const Halfperioddelay1 = 200            '
first tone 1/2 period
Const Halfperioddelay2 = 500            '
second tone 1/2 period
Const Maxcyclecount1 = 350
'length of first tone
Const Maxcyclecount2 = 150
'length of second tone
'-------------------------------------------
'program starts here
Do
```

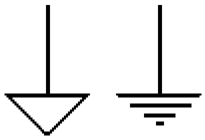
```
    'turn off both LEDs
    Reset Alarm_led
    Reset Set_led
    'wait for set switch to be pressed and released
    Do
    Loop Until Set_sw = 0
    Waitms Debouncedelay
    Do
    Loop Until Set_sw = 1
    Waitms Debouncedelay
    Set Set_led
    'wait for set switch to be unset and check for alarm at same time
    Do
       If Trigger_sw = 0 Then            'sound alarm
          For Count = 1 To 10
             Gosub Siren_sound
             'flash the led rapidly
             Set Alarm_led
             Waitms 20
             Reset Alarm_led
             Waitms 200
             Incr Count
          Next
         'flash the led until alarm is unset
          Do
             Set Alarm_led
             Waitms 20
             Reset Alarm_led
             Waitms 200
          Loop Until Set_sw = 0
       End If
    Loop Until Set_sw = 0                'debounce set switch
    Waitms Debouncedelay
    Do
    Loop Until Set_sw = 1
    Waitms Debouncedelay
Loop
End
```

16 Basic electronic theory

16.1 Conventional Current

Before the electron was discovered it was thought that the movement of charge was from positive to negative. It is common when current is being discussed for conventional current to be meant, that is current will be from positive to negative. If we want to make the difference clear we will say conventional current (positive to negative) or electron current flow (negative to positive)

16.2 Ground



In a circuit we need a reference point for all the voltage measurements, we often refer to this point as ground. At the ground point in the circuit the voltage potential is zero. In a battery powered circuit the negative side of the battery is often referred to as ground. These are the symbols you will see for a ground connection.

16.3 Preferred resistor values

Not every resistor value is made, there are ranges called the E series (Exponent?) This is useful because then not all values have to be held in stock by a company for manufacturing purposes.

E6 series	E12 series	E24 series
1.0	1.0	1.0
		1.1
	1.2	1.2
		1.3
1.5	1.5	1.5
		1.6
	1.8	1.8
		2.0
2.2	2.2	2.2
		2.4
	2.7	2.7
		3.0
3.3	3.3	3.3
		3.6
	3.9	3.9
		4.3
4.7	4.7	4.7
		5.1
	5.6	5.6
		6.2
6.8	6.8	6.8
		7.5
	8.2	8.2
		9.1

In the E6 series there are 6 values per decade, so the following values are made:
0.1, 0.15, 0.22, 0.33, 0.47, 0.68
1,1.5, 2.2, 3.3, 4.7, 6.8.
10, 15, 22, 33, 47, 68,
100, 150, 220, 330, 470, 680,
1000, 1500, 2200, 3300, 4700, 6800, and so on

16.4 Resistor Tolerances

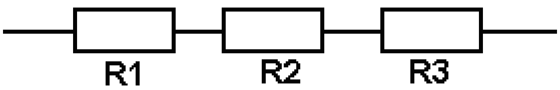
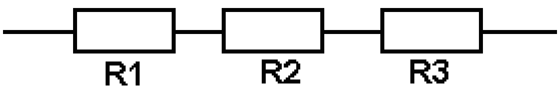
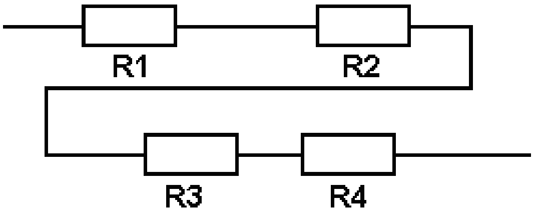
Resistors are not perfect values they are made by machine and therefore have a NOMINAL value which is correct to a reasonable accuracy. Usually we buy 1% resistors for the workshop so they are guaranteed to be close in value.

Calculate these tolerances:

Nominal Value	Tolerance	Min value	Max value
390R	1%	390 - 1% = 390 - 3.9 = 386.1R	390 + 1% = 390 + 3.9 = 393.9 R
1K			
4k7			
10K			
33K			

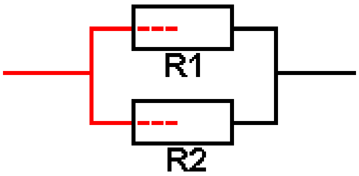
16.5 Combining resistors in series

Sometimes it is necessary to put resistors in series to get the value we need.
In circuit diagrams we use names for components such as R1, R2, R3, R4 and Rt means the total resistance. (Wherever you see **ohms** you can replace it with the symbol Ω in your work)

1. 	R1= 100R R2=300R	Rt =
2. 	R1= 10k R2=30k	Rt =
3. 	R1= 1k8 R2=10k	Rt =
4. 	R1 = 4k7 R2 = 1K8 R3 = 9K2	Rt = Rt = Rt =
5. 	R1 = 2M6 R2 = 110K R3 = 330K	Rt = Rt = Rt =
 6.	R1 = 1M8 R2 = 720K R3 = 390K R4 = 180K	Rt =

16.6 Combining resistors in parallel

When two resistors are put in parallel the current has 2 paths it can take.



The current will split between the two resistors, the current in each split will be related to the values of each resistor. The overall effect is the same as if a **smaller** value of resistance was used.

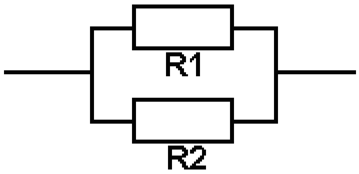
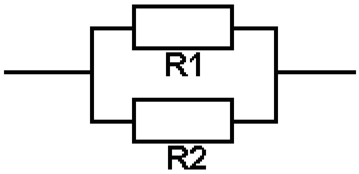
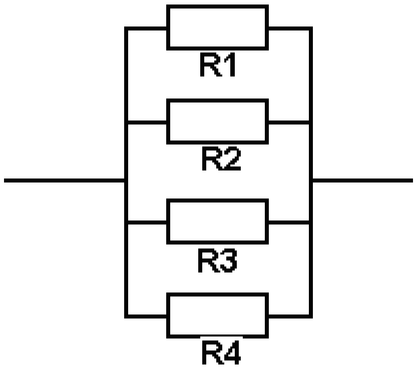
The formula for calculating the total resistance is:

$1/Rt = 1/R1 + 1/R2$ or

$Rt = 1 / (1/R1 + 1/R2)$

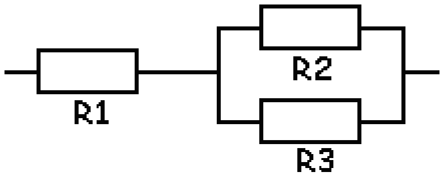
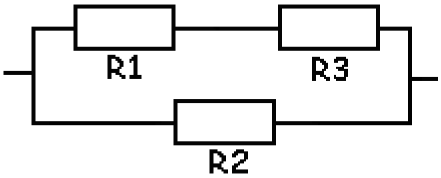
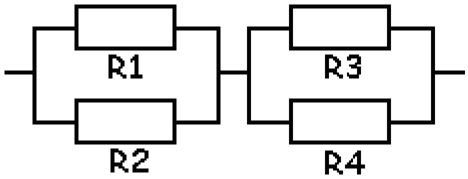
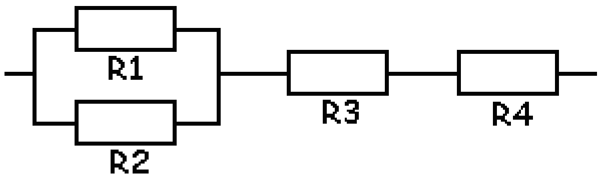
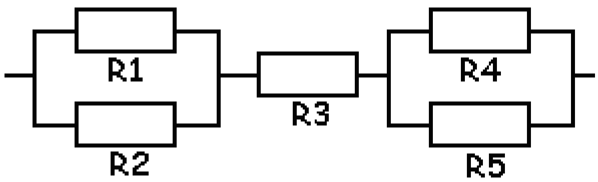
On a calculator this can be entered directly using the inverse function the 1/x button.

Enter value of R1
press 1/x
press +
enter value of R2
press 1/x
press =
press 1/x

1.		R1 = 100 R2 = 400	Rt =
2.		R1 = 1K R2 = 2K2	Rt = Rt = Rt =
3.		R1 = 2K2 R2 = 3K3 R3 = 2K R4 = 4K7	Rt =
4.	You need 180R, you have the following resistors choose 2 in parallel that would give the value closest to the desired value: 360R, 4k7, 680R 2k2		

16.7 Resistor Combination Circuits

When solving these circuits you have to look for the least complicated thing to solve first.
This can be thought of as which resistors are in a very simple combination, one that I could replace with a single resistor and not affect the current flow and voltage in another part of the circuit (its not easy and takes a lot of understanding to be able to do this, the yellow colours are hints to help with the first few)

1. R2 and R3 can be replaced by a single resistor that would not affect the current through or voltage across R1  	R1 = 10k R2 = 2k R3 = 3k	Rt=
2. R1 and R3 can be replaced.  	R1 = 4k7 R2 = 8k2 R3 = 1k5	Rt=
3.  	R1 = 16k R2 = 12k R3 = 18k R4 = 15k	Rt=
4.  	R1 = 1k1 R2 = 1k5 R3 = 470R R4 = 680R	Rt=
5.  	R1 = 4k7 R2 = 1k8 R3 = 33R R4 = 560R R5 = 330R	Rt=

16.8 Multimeters

To understand how circuits function and to find faults with them when they are not working it is necessary to know how to use a multimeter.

There is a rotary switch to select the correct measurement scale.

If you are measuring voltage in a circuit with a 9V battery you would put the meter scale onto 20V

. As the range gets closer to the actual value the accuracy gets better.

16.9 Multimeter controls

This multimeter is a common type.

The display has ______ digits. It can display numbers from 0.00 to 1999.

There are ________ different positions on the rotary switch.

V is for ______ and the ranges are

____

A is for ______ and the ranges are

____

The ohms scale has an ______ symbol.

Its ranges are ____________________

____

There are 3 different sockets for the probes to plug into these are labelled

____

The hFE selection is for testing ___________________________

COM stands for ___________ and the black/red probe goes into it.

The black/red probe goes into one of the other sockets.

What is the power source for the meter itself? __________________________

16.10 Choosing correct meter settings

Selecting the switch position is very important to making accurate measurements.
Know what you want to measure voltage, current or resistance.

The second step is selecting the range of the measurement. If an approximate value is known then choose the next higher setting on the range switch. Generally we use 9 volt batteries in our circuits, if you want to measure voltages around a 9 volt circuit then what range would you choose for the meter? ________

If you did not know the voltage in the circuit which range would you choose? __________

Many of the resistors we use are 5 band, very small size and hard to read. What range would be best to choose first on the meter? ___________

What range would you choose to measure a resistor you thought was 91Kohms.________

What range would you choose to measure a resistor with colours red, red, orange, brown? ____________

What is the highest resistance value that can be read on the meter? ________

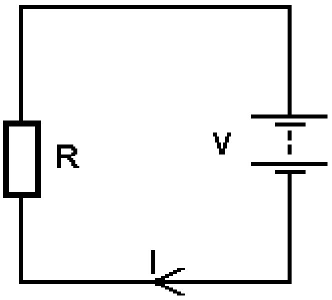
What is the lowest resistance that could be measured on the meter? ________

When measuring current where would you put the probes and what range would you choose to start with?
____

If no current readings are being shown on the meter it is possible that the _______________________________.

When making a measurement and its value is greater than the scale used the display shows ______________

16.11 Ohms law



This a very important formula in electronics. You must be able to use it correctly and develop a comprehensive understanding of its meaning.

In a circuit one volt will drive one ampere of current through a one ohm resistor (or when one amp is flowing in a one ohm resistor one volt will be developed across the resistor)

The formula is Voltage = Current times Resistance or **V = I x R**

If 0.5A is flowing through a 10 ohm resistor then what is the voltage across the resistor?

Answer: V=I*R, V=0.5*10, V=5Volts.

If the voltage is 10volts and the resistance is 2ohms then what current through the circuit?

Answer:I=V/R, I=10/2, I=5A.

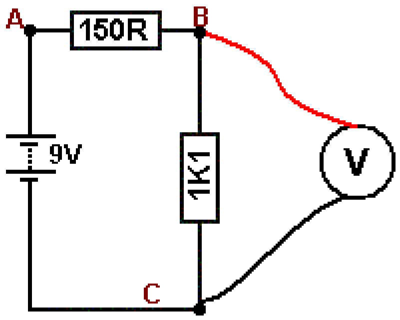
At 9V, if 0.0019A is flowing through the circuit what is the value of R?

Answer: R=V/I, R=9/0.0019, R=4,700 ohms

1.	I= 0.002A, V= 16V	R =V/I, R=16/0.002, R=8000 ohms
2.	V= 12V, I= 0.015A	R =
3.	V= 9V, I= 2A	R =
4.	I= 0.0001A, V= 5V	R =
5.	R= 2000, V= 6V	I =
6.	V= 50V, R= 10,000	I =
7.	V= 3V, R= 100,000	I =
8.	R= 47,000, V= 20V	I =
9.	I= 0.00183A, R= 12000 ohms	V =
10.	I= 0.0015, R= 1000 ohms	V =
11.	R= 20R, I= 0.2	V =
12.	I= 0.4, R= 120R	V =

16.12 Voltage & Current Measurements

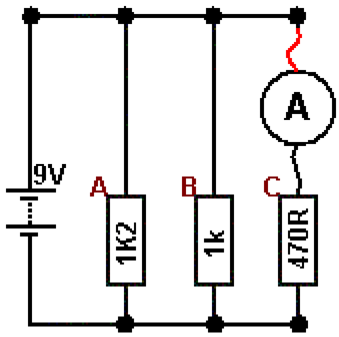
16.12.1 Measuring Voltage



- Calculate the voltage across each resistor
- Circuit current first.
- I = V/R =____________
- V(1k1) = IxR = ___________________
- V(150R) = IxR = _________________
- Setup the multimeter correctly and measure the voltages in this circuit.
- What was the voltage measured across the 1k1 ________across the 150R _______________

16.12.2 Measuring Current

To measure current in a circuit the circuit must be broken and the meter inserted into it.



- Calculate the current through each resistor.
- $I_A = V/R_A$; I_A = _______________________
- $I_B = V/R_B$; I_B = _______________________
- $I_C = V/R_C$; I_C = _______________________
- Measured Values
- I_A = ______________,
- I_B = ______________,
- I_C = ______________

There are at least two reasons for differences between calculated and measured values in this circuit what could they be?

____

____

16.12.3 Meter Safety

- The meter is a delicate instrument handle it with care.
- Estimate what your measuring first and set the meter range to a larger value(or even to the maximum value),
- Do not measure resistance in a circuit when the circuit is on.
- Check the internal fuse is correct before measuring current.
- Turn the meter off after use.

16.12.4 Circuit Safety

- Using the meter on a current setting when wanting to measure voltage can easily damage components and even the circuit board.
- Take care not to short parts of the circuit with the probes.

16.12.5 Battery Life

- Switch the meter off when finished using it.

16.14 Continuity

One range on the meter will beep when the probes are shorted together, or a very low value of resistor is connected. It is very useful for

- checking cables are not broken
- checking that tracks between parts of a PCB are not broken
- checking that tracks are not shorted together on a PCB

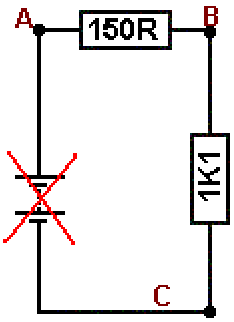
Find 6 items that are good conductors

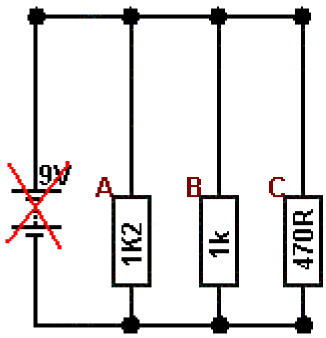
and 6 items that are poor conductors

16.14.1 In-circuit measurements

When a resistor is unknown or suspected faulty its resistance can be measured using the multimeter on ohms range. <u>When measuring resistors "in circuit" you must disconnect the power</u>. To measure resistance the meter puts current through the resistor and measures the voltage across it so current from within the circuit will confuse the readings and the meter or the circuit could be damaged.

Measure the resistors in the following circuits.





Can you explain your readings for the second circuit.

all three resistors are measured at once so the meter reads
only the parallel combined resistance

16.15 Variable Resistors

Variable resistors or potentiometers, are used to change the input to an electronic circuit.

They come in different shapes, sizes and values as well ‘dual-gang’ (what use is a dual one?)

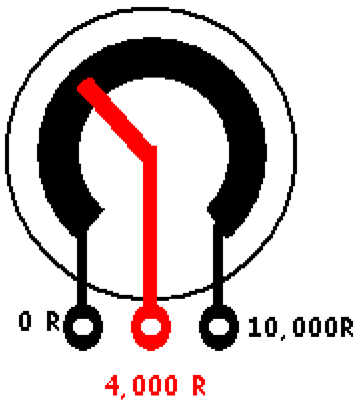
Some are designed to be varied by the user of the circuit, and are fitted with knobs to turn them, such as those used as volume controls.

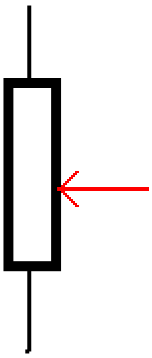
Others are called trimpots and are meant to be varied only by service people when working on the inside of equipment, these are turned with a screwdriver.

Most pots vary over 270 degrees not the full 360 degrees.

The resistance between the two outer terminals does not change, only the resistance between the centre terminal and both the outer terminals.

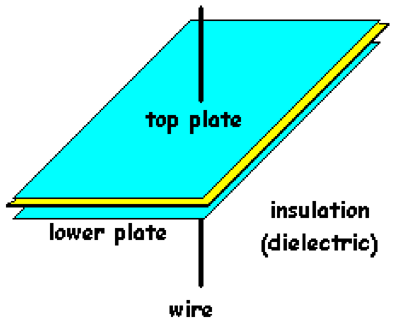
For this 10k pot, fill in the missing values from the table





angle	0 to centre	centre to 10k
0	0 R	10,000 R
30	1,000 R	
108	4,000 R	
	5,000 R	5,000 R
190	7,000 R	
		1,000
270	10,000 R	0 R

If a lever was attached to the control of a pot what sort of things could be sensed by the circuit?



16.16 Capacitors

A capacitor is made from 2 conductors separated by an insulator. Electrons do not flow through a capacitor, they flow onto one plate causing electrons to flow away from the other plate. Once the capacitor is full no more electrons can flow. A capacitors action is to **<u>store charges.</u>**

16.17 Capacitor Codes and Values

Capacitors not only come in a variety of packages and types but there are also a number of different ways that their values can be printed onto them. Some values are in uF, some in nF and some in pF, and it can be confusing until you learn the few simple rules.

1. learn the prefixes first, <u>micro uF, nano nF, and pico pF</u> micro is the biggest, nano in the middle and pico the smallest and learn how to convert between them.

2. Look at the capacitor to see what is written on it. If it has 10uF or 22n the it is obvious what value it is.

However when it is written with 3 digits such as 333, then it will be in pF even though it it not stated, and the last digit will be the number of zeros (a bit like resistor colour codes) so 333 means 33,000 pF.

Convert the following

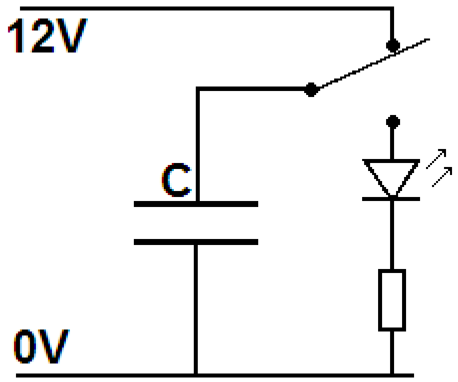
333 =	33,000pF
330 =	330pF
221 =	
470 =	
474 =	

33 =	33pF
685 =	
220 =	
68 =	
276 =	

16.18 Converting Capacitor Values uF, nF , pF

			farads units		micro u	nano n	pico p
					1		
					1	0 0 0	
					1	0 0 0	0 0 0
					1uF = 1,000nF = 1,000,000pF		
					0	1	
						1 0 0	
						1 0 0	0 0 0
					0.1uf = 100nF = 100,000pF		
10nf to pf	A						
82nF to uF	B						
2200pf to nF	C						
100,000nF to uF	D						
370pF to nF	E						

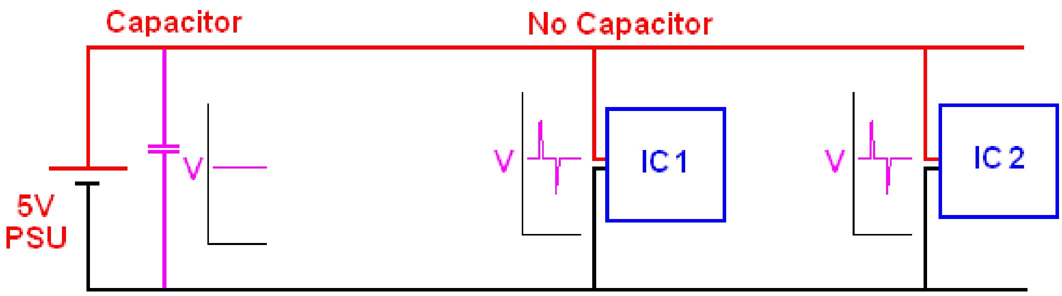
16.19 Capacitor action in DC circuits



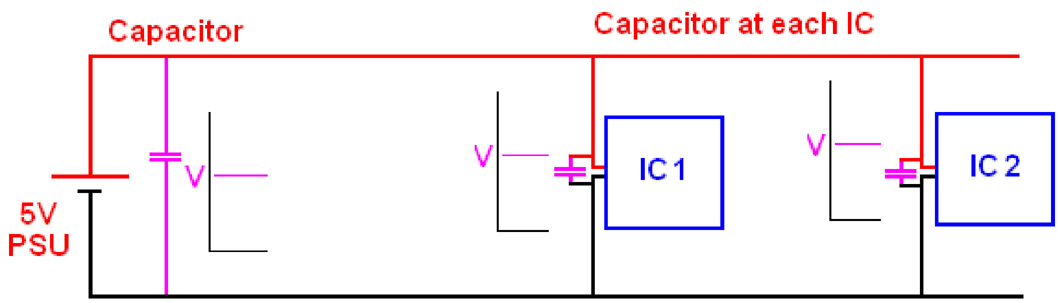
In this circuit when the switch is in the upper position the capacitor will store the charges on its plates; when moved to the lower position the stored charges will be released back to ground through the LED and resistor. The higher the value of the capacitor and the lower the value of the resistor the longer the capacitor will take to discharge and the longer the LED will glow. The value of capacitance is the amount of charge that can be stored; it is related to the size of the plates and the thinness of the insulator. A Capacitor is fully charged when the voltage across it equals the supply voltage.

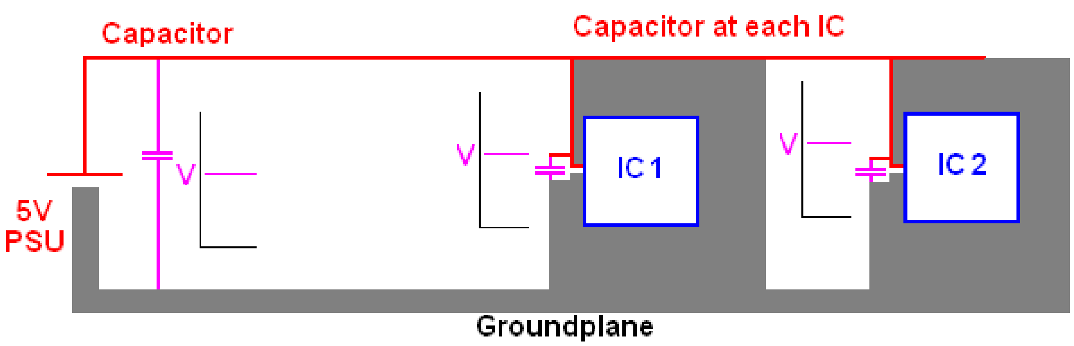
This ability to store charge is absolutely crucial in circuits that need quality power. In a computer circuit that switches signals at megahertz or gigahertz a lot of power can be required for tiny periods of time e.g. 1 nanoseconds (0.000000001 second).

If there is no capacitor close to the IC, it pulls the extra charges it needs from the power supply wires close to the chip, this appears as rapid changes in voltage level or 'spikes' in the voltage, these spikes transfer along the power lines on a pcb and upset nearby ICs as well.



A common practice in electronics is to have a 0.1uF cap next to the power pins of every IC to minimise this effect.





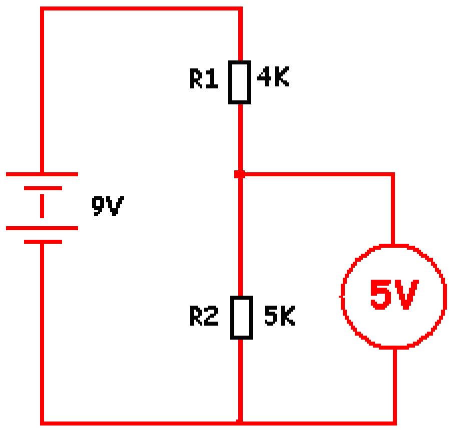
Another common practice nowadays is to have large areas of copper on the circuit board connected to ground (0V). This acts as a large store of charges.

Many circuit boards have multiple layers of copper tracks inside the board, one of which is ground and another of which may be the power (e.g. 5V).

16.20 The Voltage Divider

The voltage divider is is one of the most important circuits in electronics. It is used extensively in input circuits. To understand its operation you must know about ohms law.

Below is a 2 resistor voltage divider circuit. The output voltage is the voltage across R_2,
Step 1. Voltage and total resistance are known, so $I = V_{in}/R_t$
Step 2: R_2 and Current through R_2 are known, so $V_{out} = I*R_2$

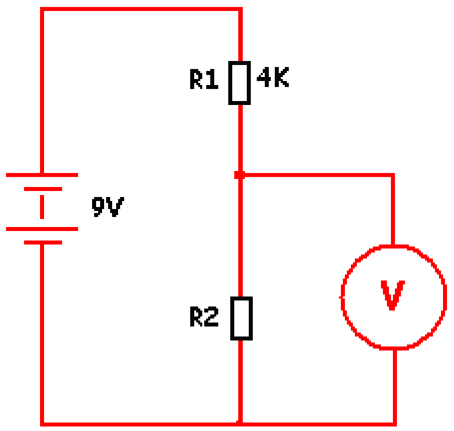


With 9 volts across both resistors then:

$I = V_{in}/R_t$
I = 9/(4000 + 5000)
I = 9/9000
I = 0.001A

across the 5k resistor
$V_{out} = I*R_2$
V_{out} = 0.001*5000
V_{out} = 5V

Work out the solution to the following.



R_2 = 1K

$I= V_{in}/R_t$

I = 9/ (4000+ ________)

I = ___________

$V_{out} = I * R_2$

V_{out} = ________ * 1000

V_{out} =

R_2 = 8K

$I= V_{in}/R_t$

I = ______________________

I = ___________

$V_{out} = I * R_2$

V_{out} = ________ * 1000

V_{out} =

R_2 = 4K

I=

I =

I =

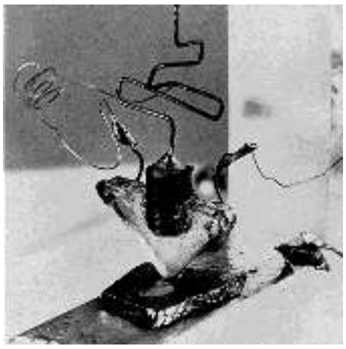
V_{out}=

V_{out} =

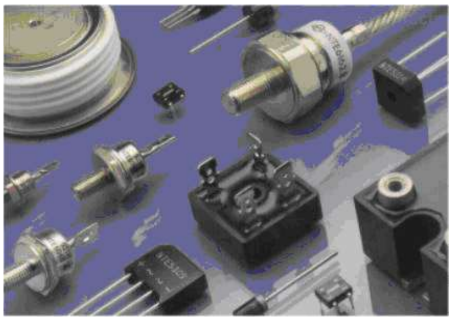
V_{out} =

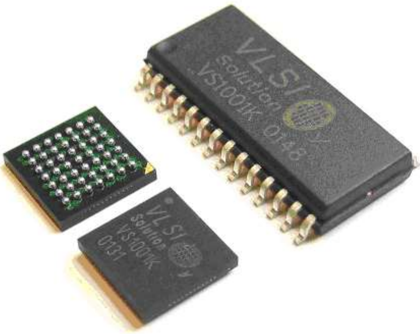
16.21 Using semiconductors

Semiconductors are the group of electronic components responsible for everything smart that electronic circuits do. Made mostly from the semiconductor silicon, which is itself a very poor conductor, they take on fantastic features when mixed with other material.

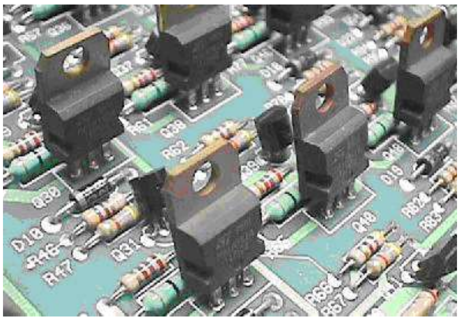


Since the first transistor was developed in 1947 they have come a long way.

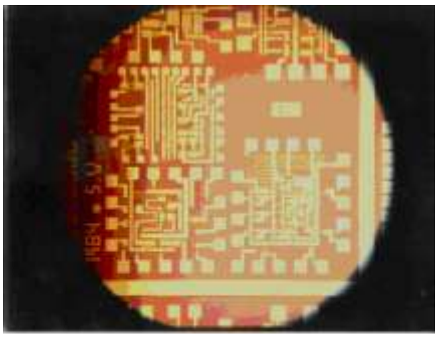




They now come in all shapes and sizes. from miniature surface mount packages to large high power packages.

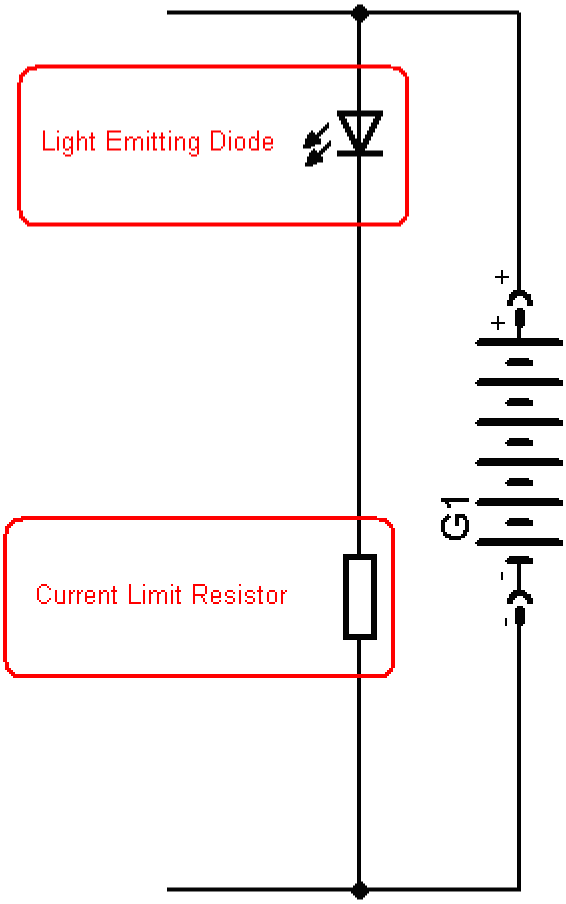


They amplify, switch, and control every conceivable process



all over the world

16.22 Calculating current limit resistors for an LED



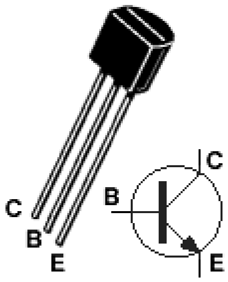
In the amplifier circuit there is an LED to indicate that power is on.
The resistor in series with the LED functions to **limit the current** through the LED.

- AN LED requires a small forward voltage e.g. _____V across it to operate, however the circuit is powered by a 9V battery. The rest of the battery voltage must be dropped across the resistor.
- Ohms law will assist with this calculation.
- The resistor will have 9V - ____V = ____V across it.
- An led draws about _____mA of current, this current goes through the resistor so
- the resistor will need to be R = V/ I = ______ / ______ = ______ ohms.
- Choose the closest value from the available values of resistors.

If two LEDS were placed in series what value of resistor would be required?

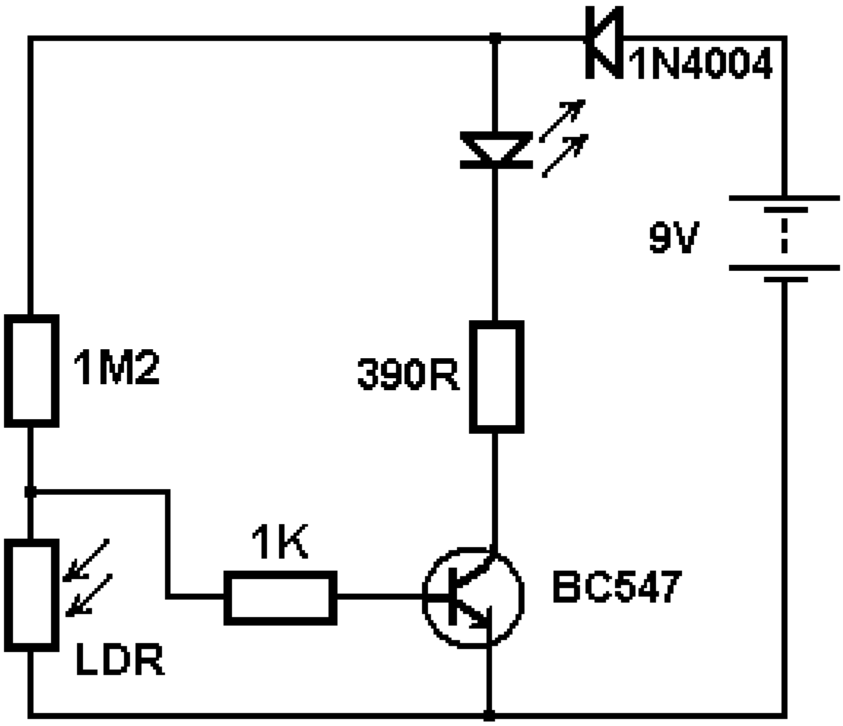
16.23 The Bipolar Junction Transistor

There are thousands (millions?) of different types of transistors made by different manufacturers all over the world, and they come in all shapes and sizes. The correct name for the usual transistor is the BJT or Bipolar Junction Transistor. We could have used a BC547 instead of the 2N7000 FET for the darkness detector.

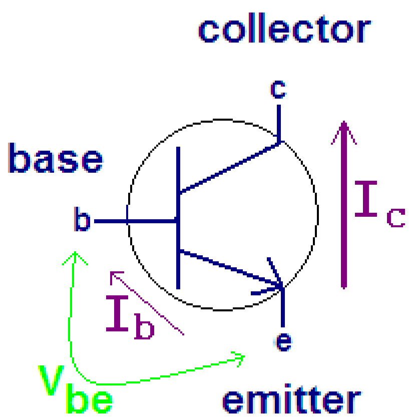


Transistors are semiconductor devices with three leads: an emitter, a base and a collector.

The BC547 transistor is just one of the many different types of BJT transistor. The BC547 is an NPN transistor, there are also PNP transistors the BC557 is an equivalent PNP transistor .



Transistors are amplifiers, a small voltage across the base-emitter junction (the small arrow in the transistor symbol) will control the current (the large arrow) from the emitter through to the collector.



The small voltage across the base is called V_{be} , the current through the base caused by this voltage is called I_b. And the current through the collector is called I_c.

Small variations in the base voltage V_{be} can create large changes in the collector current I_c.

The voltage required across the base of the transistor (V_{be}) is normally around 0.6V to 0.7V when it is fully conducting.

16.24 Transistor Specifications Assignment

Transistors have <u>current gain</u> (h_{FE}), this is the ratio of base current ($I_{b)}$ to collector current ($I_{c)}$. If I_b is 2mA and I_{ce} is 100mA then the gain is said to be 100/2 = 50.

Transistors have limits to the voltages and currents applied to them in circuits. They should not be exceeded. If the voltages across the base or collector are too high then the transistor will most likely blow up internally; if you try to draw too much current from the collector then it will most likely overheat and burn up

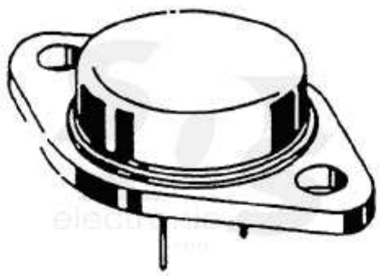
Look up the specifications for the following transistors in a catalogue

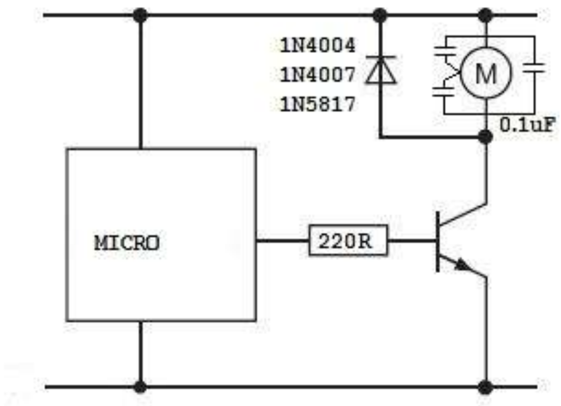
	BC547	BC557	BC337	BC327	BD139	BD140	TIP41C	TIP42C	2N3055
Type	NPN								
Case	T092								
I_C (mA)	100 mA								
$V_{ce\ MAX}$	45 V								
h_{FE} (gain)	110-800								
P_{TOT} (power)	500 mW								

16.25 Transistor Case styles

T0_________ T0_________ T0_________ T0_________



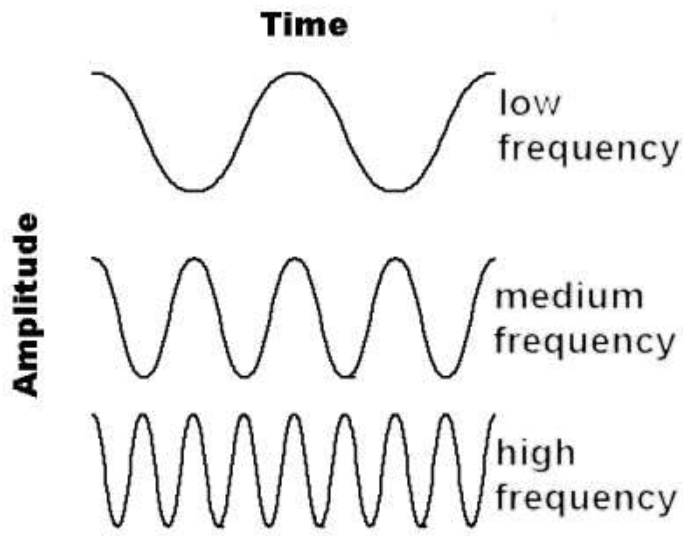


16.26 Transistor amplifier in a microcontroller circuit

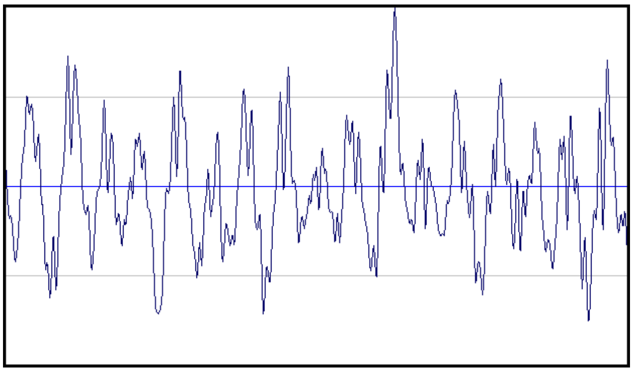
We often use a NPN transistor in our circuits so that the microcontroller can control low to medium power devices such as small motors or lots of LEDs

16.27 Transistor Audio Amplifier

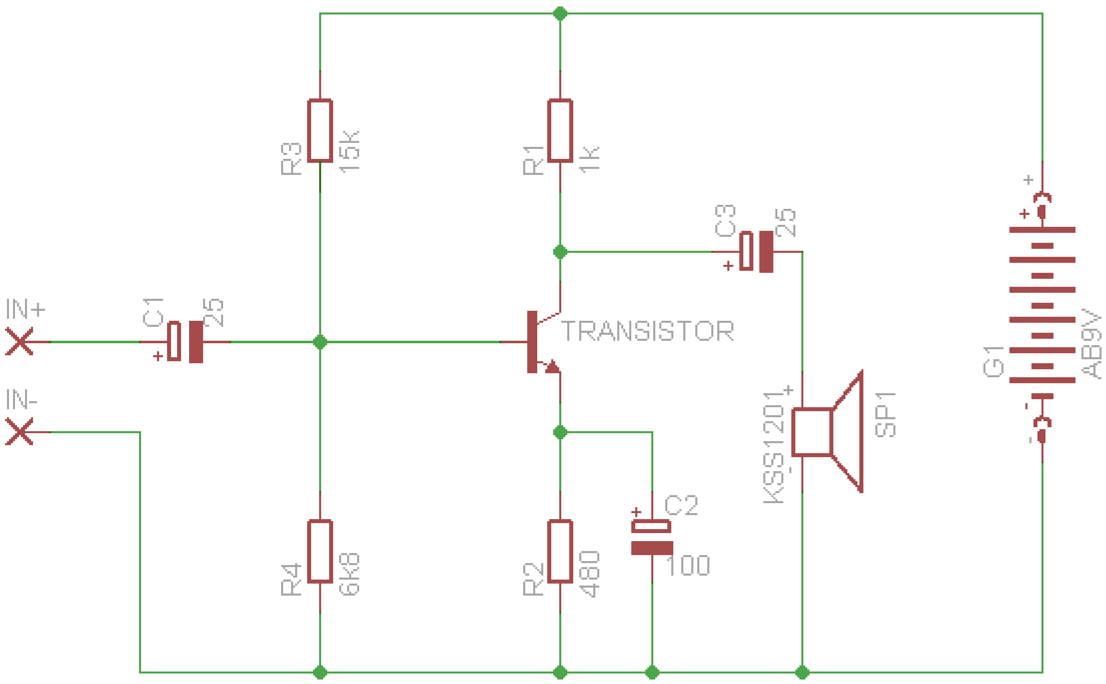
Audio signals are not DC like that in a microcontroller circuit they are alternating current (AC) signals. AC is measured in frequency (number of cycles per second) and amplitude (size).



Audio signals such as voices are not single waves but complex waves of many frequencies each of differing amplitude as in the picture below.



When amplifying audio through a transistor amplifier the frequency should not change but the amplitude will. (In a single transitor circuit the signal is inverted, but that doesn't really make any difference to what we hear)

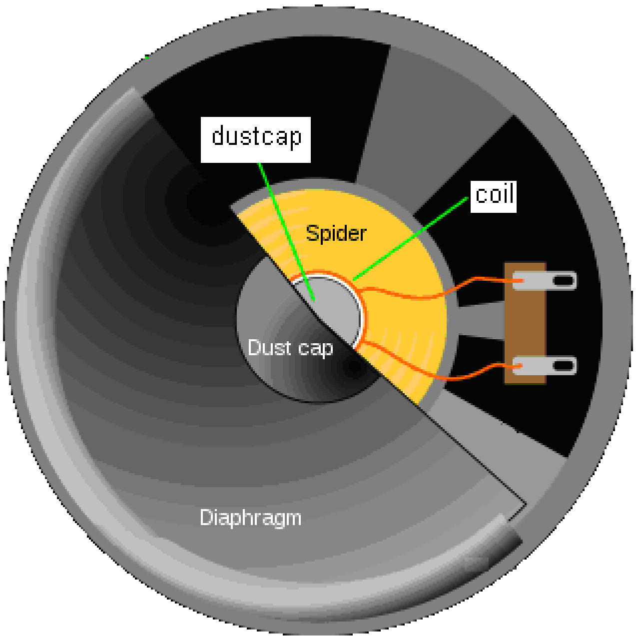


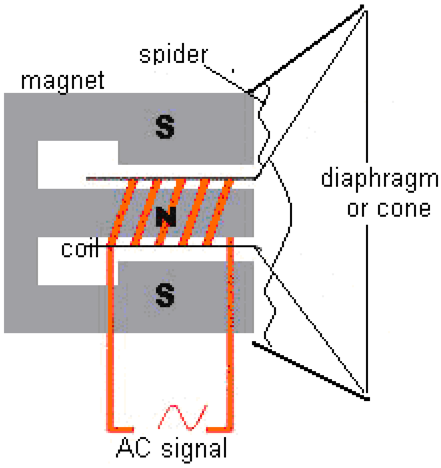
This transistor circuit is setup to amplify small audio signals (it is not a very high gain/amplification circuit)

A lot of components are required to control the transistor circuit so that it doesn't distort the audio signal.

16.28 Speakers

Sound is vibrations of air particles; a speaker will change the audio signal from an amplifier by moving the cone of the speaker rapidly back and forth vibrating the surrounding air.



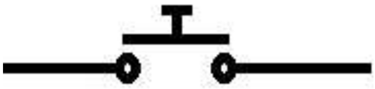
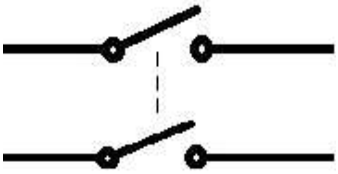
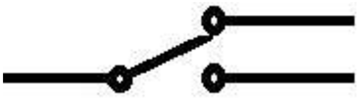
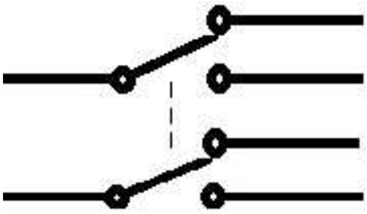
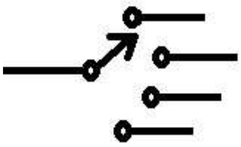


Speakers come in various types each with specific frequency ranges they can reproduce: subwoofers (very low frequencies), woofers (low frequencies), mid-range speakers (middle frequencies), and tweeters (high frequencies).

Speakers have a resistance and typical values are 4 or 8 ohms. They also have a power rating e.g. 100W, 20W or o.25W.

If you connect a speaker directly to a battery you will destroy it (no smoke or explosion just a dead speaker).

16.29 Switch types and symbols

Symbol	Switch description	Example	Example name
	SPST Switch Single pole single throw		Toggle switch Mecury Switch Rocker switch
	push to make		Push Button Switch
	push to break		Push Button Switch
	DPST switch Double pole single throw		Rocker switch
	SPDT switch Single pole double throw		Toggle switch Or Microswitch
	DPDT Switch Double pole double throw		Toggle Switch or slide switch
	4 way (or more)		Rotary Switch

17 Basic project planning

The development of a technology project requires much more than the making of a working prototype, it requires students to undertake a full development process of planning, design, client and stakeholder liaison along with much modification to develop the prototype that meets a clients' needs.

A great number of tools are available for use when planning and executing the development of a project, such as:

- action plans
- Gantt or PERT charts
- timelines
- goal/target setting
- keeping a journal
- publishing a website
- stakeholder surveys and questionnaires
- emails
- spreadsheets
- mind maps
- presentation software
- drawing software
- surveymonkey
- CAD and PCB design software
- Block Diagrams
- Schematics and Layout

Many planning tools can be found at
www.mind-tools.com or www.visual-literacy.org

As you go thorugh the various stages of developing a project, your **effective selection**, **review** and **use** of these tools will count towards your grades.

17.1 System Designer

System Designer software was developed to help students both design and manage their project; it contains various different types of drawings that will be used during development of a prototype

17.1.1 Creating a new project.

It is essential that each project is saved into its own folder, as a unique file for each diagram within System Designer is created.

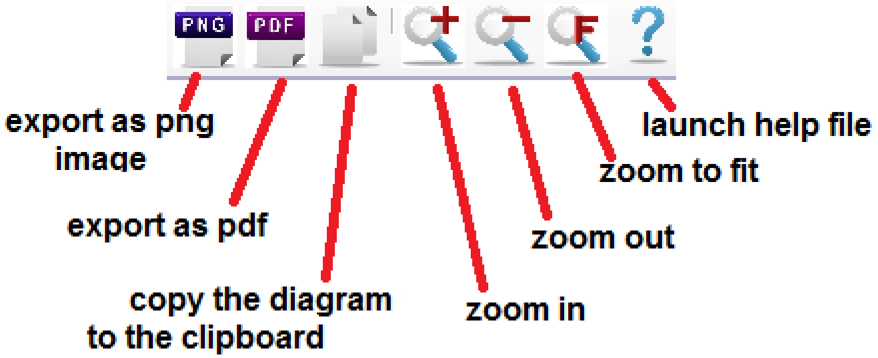
Use the toolbar along the top to create various diagrams.

The process you go through may vary but here is a guide to follow initially:

1. First create a Mind map for the project
    a. This diagram will help you to think about the different stages required when developing your project.
    b. Initially there may not be much in the diagram as the planning cannot really be undertaken fully until after the system is designed
2. Then develop a System Context Diagram
    a. This diagram shows your system from the outside, all of the internal workings of it are hidden. This will take several iterations (cycles of development)
    b. Keep different diagrams for the different stages and changes you go through
3. Next create a Timeline – go back and modify the mind map diagram (and use the auto create timeline function)
    a. In this diagram you can begin to plan the processes and resources required to develop the prototype.
4. Next create a System Block Diagram
    a. In this diagram you can visualize the internal subsystems within the device.-This will also be an iterative process so keep different drawings for different options
5. A Board Layout can be created next
    a. A board layout can be used to plan the layout of components onto breadboard, Veroboard and selected development boards.
    b. Note that a board layout will not be required if a PCB was designed specifically for the project
6. Add an Algorithm
    a. An Algorithm is a written explanation or set of instructions that describe the functions the microcontroller program will carry out.
7. Flowcharts/Subroutine diagrams
    a. Smaller systems can be designed using a Flowchart and as many subroutines as required.
8. State machines
    a. Larger systems will need a State Machine Diagram and possibly some subroutines
    b. A state machine is a very common diagram used in designing software for embedded systems

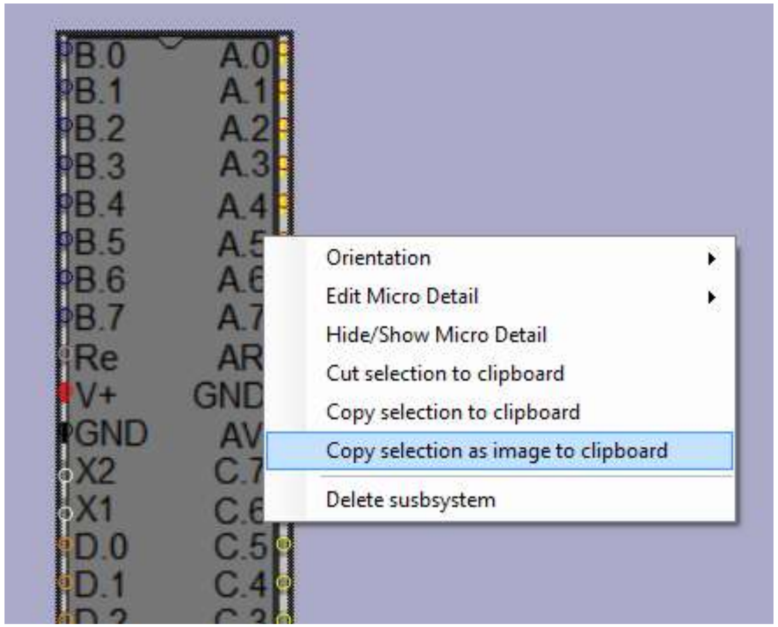
17.1.2 Toolbars

The toolbars in each diagram contain tools to add specific components to each diagram.
Some components are the same in each diagram though



17.1.3 Context Menus

Many features of diagrams are accessed through right clicking on the components, links and backgrounds of each diagram



17.1.4 Selecting items to copy them

Press the ctrl key and click and drag over portions of the diagram to select it. Then right click on the selection to decide whether to copy them to the clipboard, so they can be pasted into another diagram, or copy as an image to the clipboard so they can be copied into another program.

17.1.5 Pan diagrams

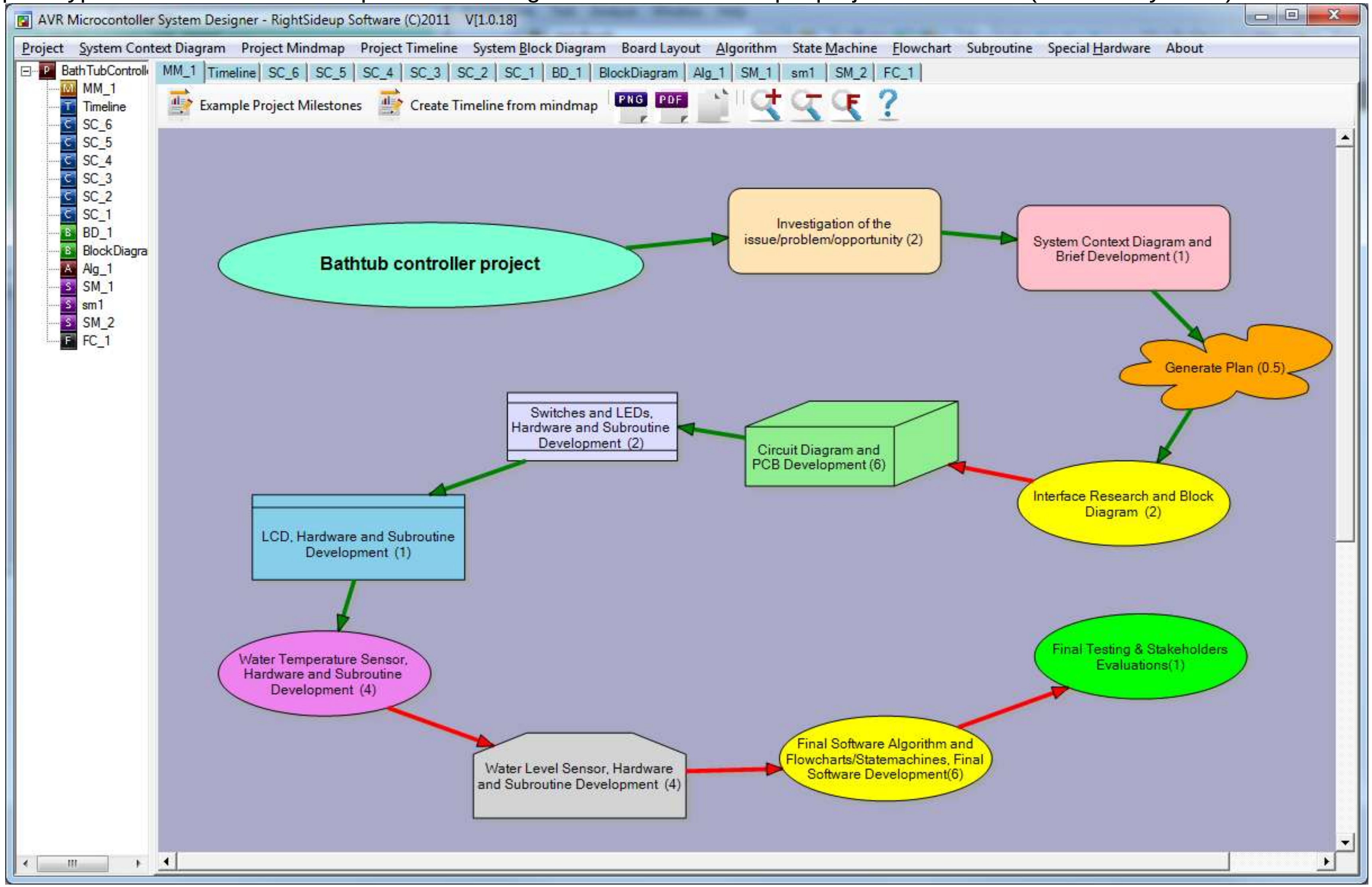
Press the mouse wheel button to select the diagram to move (pan) it around.

17.1.6 .Zoom diagrams

Use the mouse wheel or the buttons on the toolbar

17.2 Project mind map

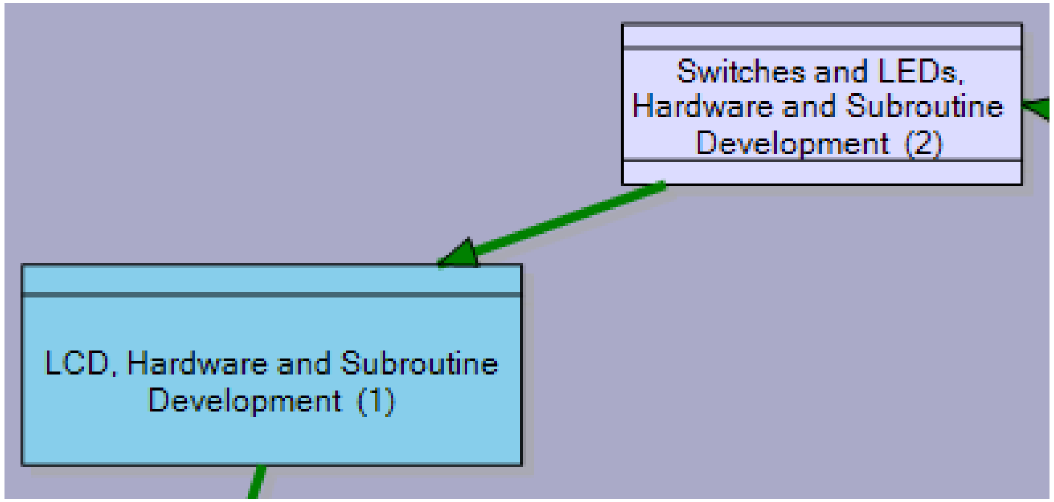
This diagram is a simple brainstorm of the milestones (major stages) required to develop a project from an issue right through to a working prototype. Students can develop their own diagram or use the example project milestones (and modify them)



Colours and other details can be changed by right clicking on the milestone or background.

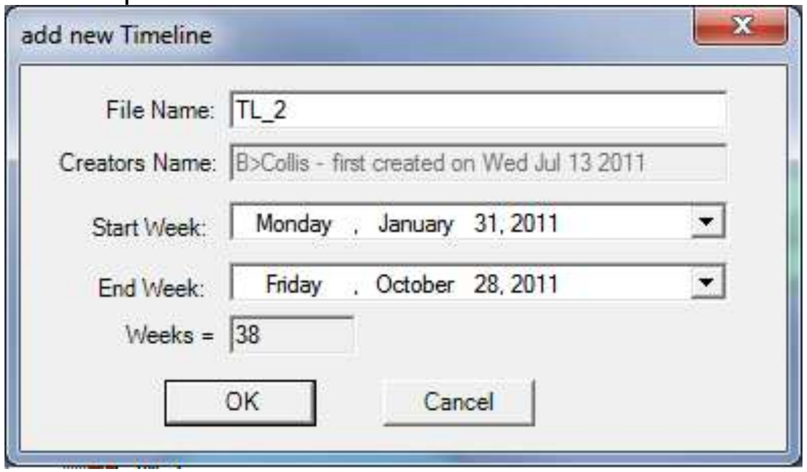
17.2.1 Milestone duration

At each milestone if the number of weeks is added in brackets it can be copied thru to the timeline
Values include part weeks e.g. (0.3).



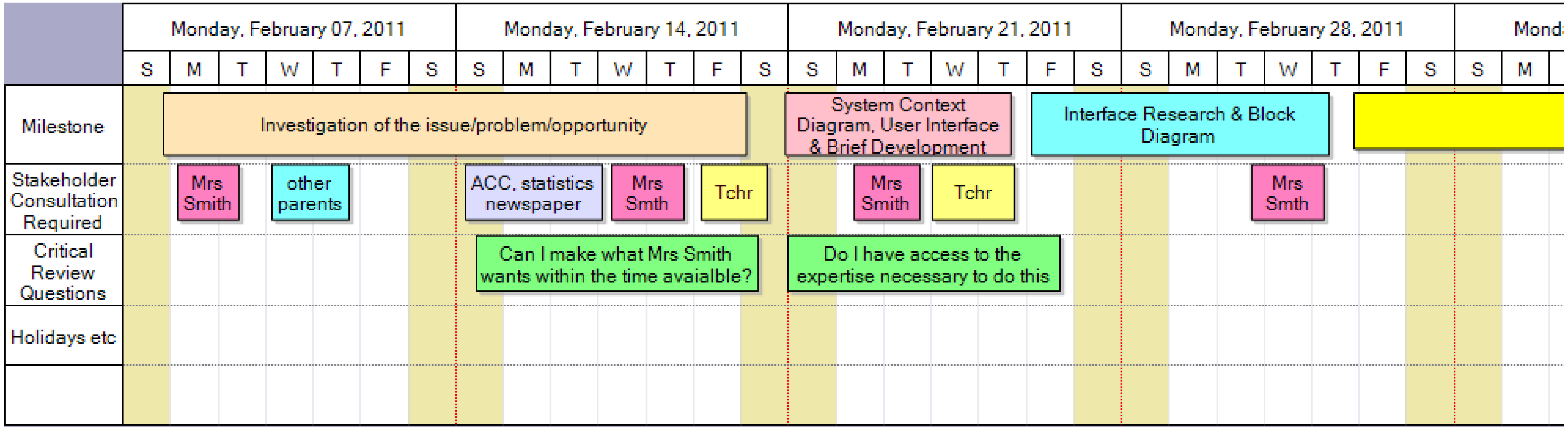
17.2.2 Automatic timeline creation

Once the milestone stages have been decided upon a timeline can be automatically created using the milestone colours and weeks values from the mind map.



The form that opens will automatically start from the beginning of the current year.

17.3 Project timeline



Investigation of the issue/problem/opportunity	
Actions	research issue with client to find her key concerns, identify other stakeholders to find their opoinions. research accident stats on burns
Resources	approx 2 weeks to carry out interviews and write up findings
Expertise	Discuss with teacher to see if the project will be achieveable
Equipment	Internet stakeholder questions e-journal to record progress and results
Research	Accident stats find 3-4 suitable other stakeholders
Budget	Nil
CRITICAL REVIEW POINT	These things went right... These things I had to change so that I would be able to continue... Before I started work on this stage I should have... Adjustments made to future plans so I will be able to complete...

System Context Diagram, User Interface & Brief Development	
Actions	identify the essential product features and determine clear specifications for them
Resources	catalogues Internet - suppliers Internet - Forum to see if someone has already done it
Expertise	Teacher friends
Equipment	
Research	find water level sensor, water temperatre sensor, large light
Budget	may need extra mony to buy expensive sensors
CRITICAL REVIEW POINT	These things went right... These things I had to change so that I would be able to continue... Before I started work on this stage I should have... Adjustments made to future plans so I will be able to complete...

In the timeline diagram milestones can be drawn (if not already created automatically from the mind map). Double clicking on a milestone allows it to be edited.

17.3.1 Milestone Planning

A milestone is made up of several planning steps as well a review of progress ad reflection at the end of it.
The following information is required by the planning standard: actions, resources, expertise, equipment, research, and budget.
Take time to complete these as thoroughly as possible.
The tables can be resized and moved around the diagram to create a better layout for exporting.

17.3.2 Stakeholder Consultations

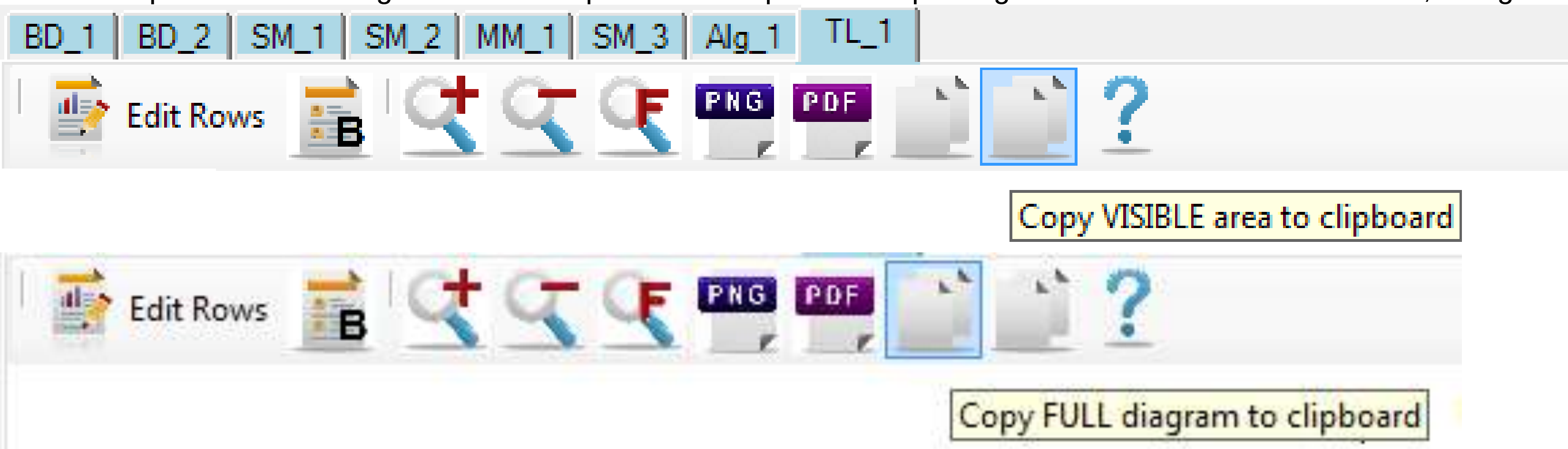
It is important to identify the points in your project where different stakeholders will have to be consulted.
As well as the information required from them.

17.3.3 Critical review points

Each milestone in the project will have critical points associated with it that will need to be overcome so that they don't stop you from reaching the next stage and subsequently the final goal of finishing your project. You need to identify these and comment on them.

17.3.4 Copying Timelines to put them into your journal

To export a timeline to another document such as Word etc, first resize and move the tables around the diagram and also change the zoom level to obtain the view wanted.
The visible portion of the diagram can be copied to the clipboard for pasting into a word or other document, using the button on the toolbar.



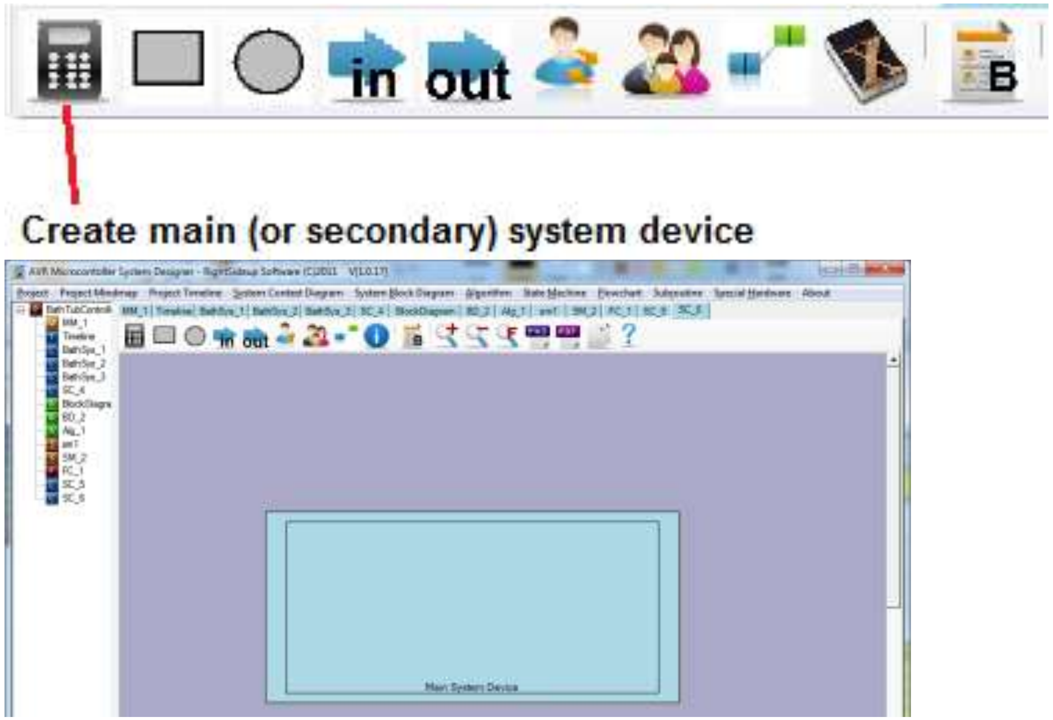
17.4 System context diagram

Although you are developing a prototype (product/outcome), you need to see it as both a system and a subsystem (smaller component of a larger system) with all the associated inputs and outputs.

The system context diagram is to recognize that your prototype is a subsystem within its larger context/environment.
A context diagram shows how your prototype interacts with users (called 'actors' in the programming industry) and its immediate environment.
No detail about the inner workings of the prototype is required. Think of the prototype as a 'black box'; all we know about it are its inputs, outputs and attributes (physical characteristics, functions, qualities and features)
A system context diagram is also an essential tool in writing an initial brief as it helps to document stakeholder requirements

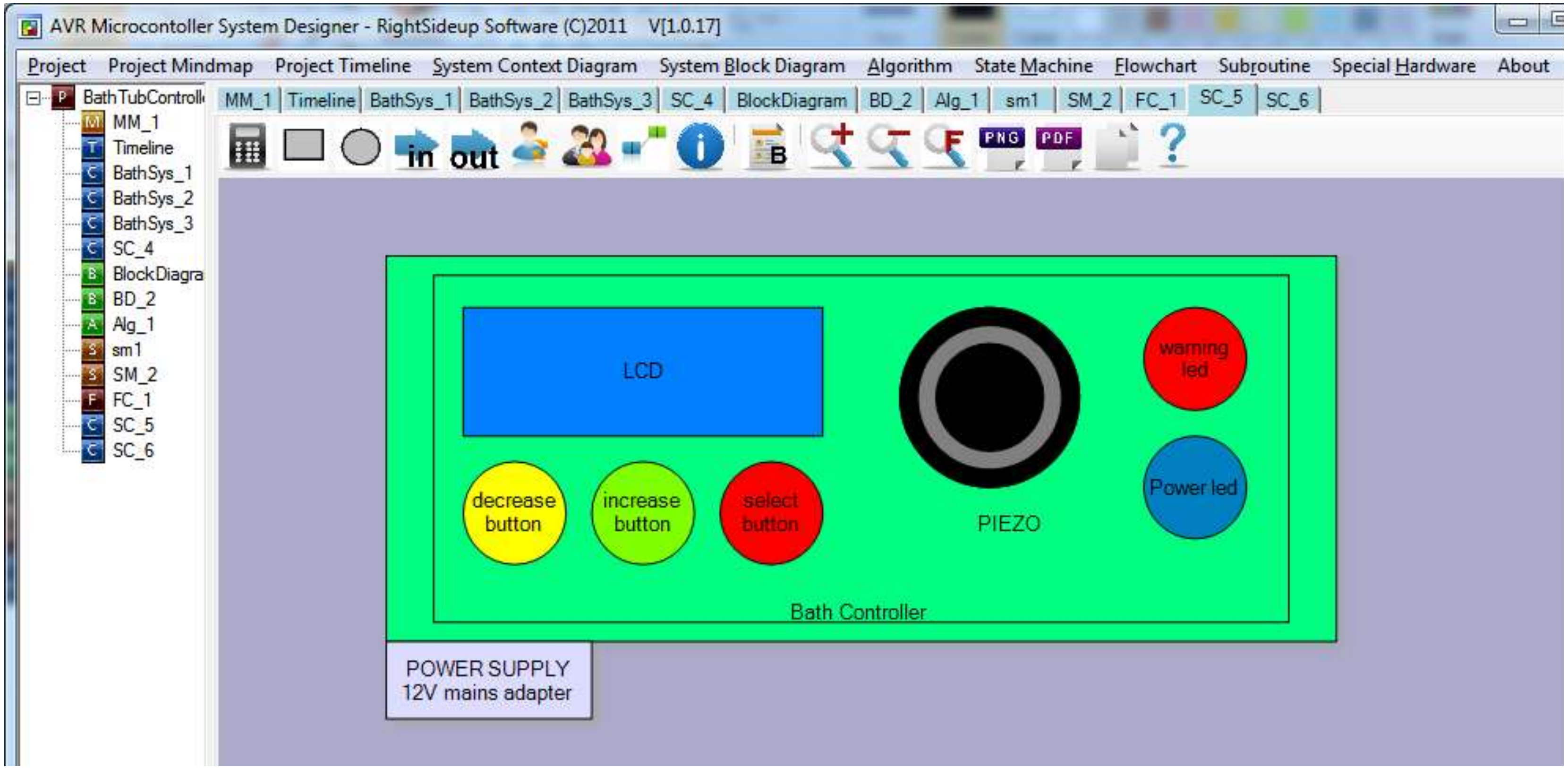
As well as this, the system context diagram will provide evidence for the following standards: modeling, systems, brief writing, planning, and prototyping.

17.4.1 First step is to create a main system device



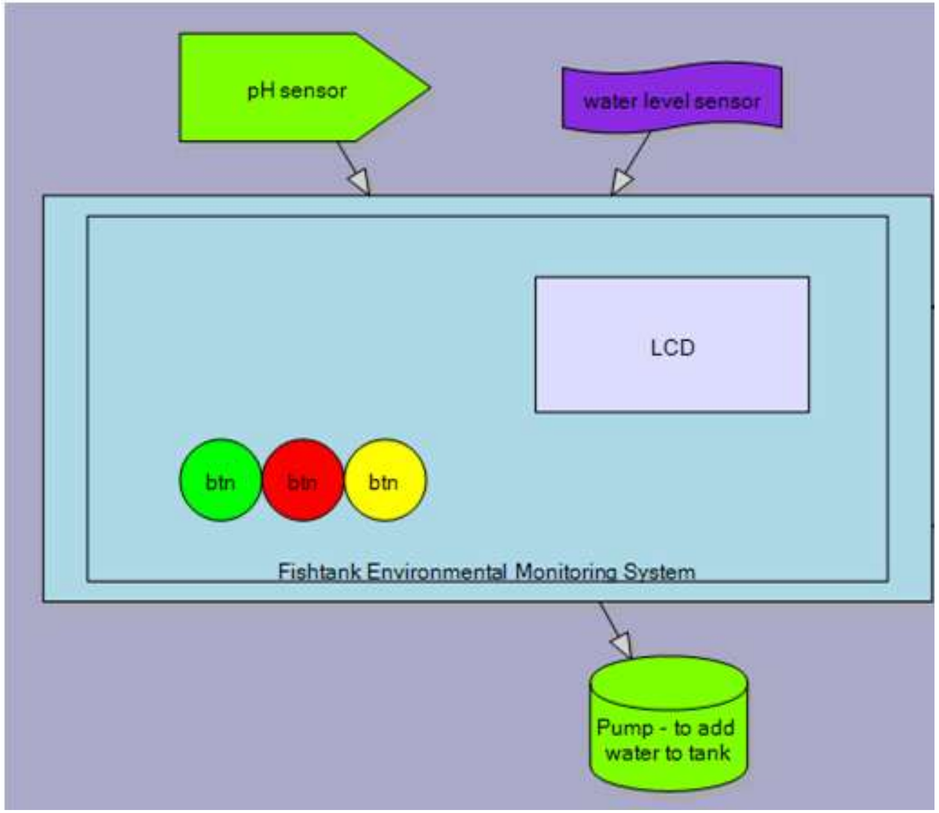
17.4.2 Add attributes to the device

Use the rectangle and circular buttons on the toolbar to add physical attributes to the device (right click on an attribute to change its shape)
Give the device and all its attributes useful names.



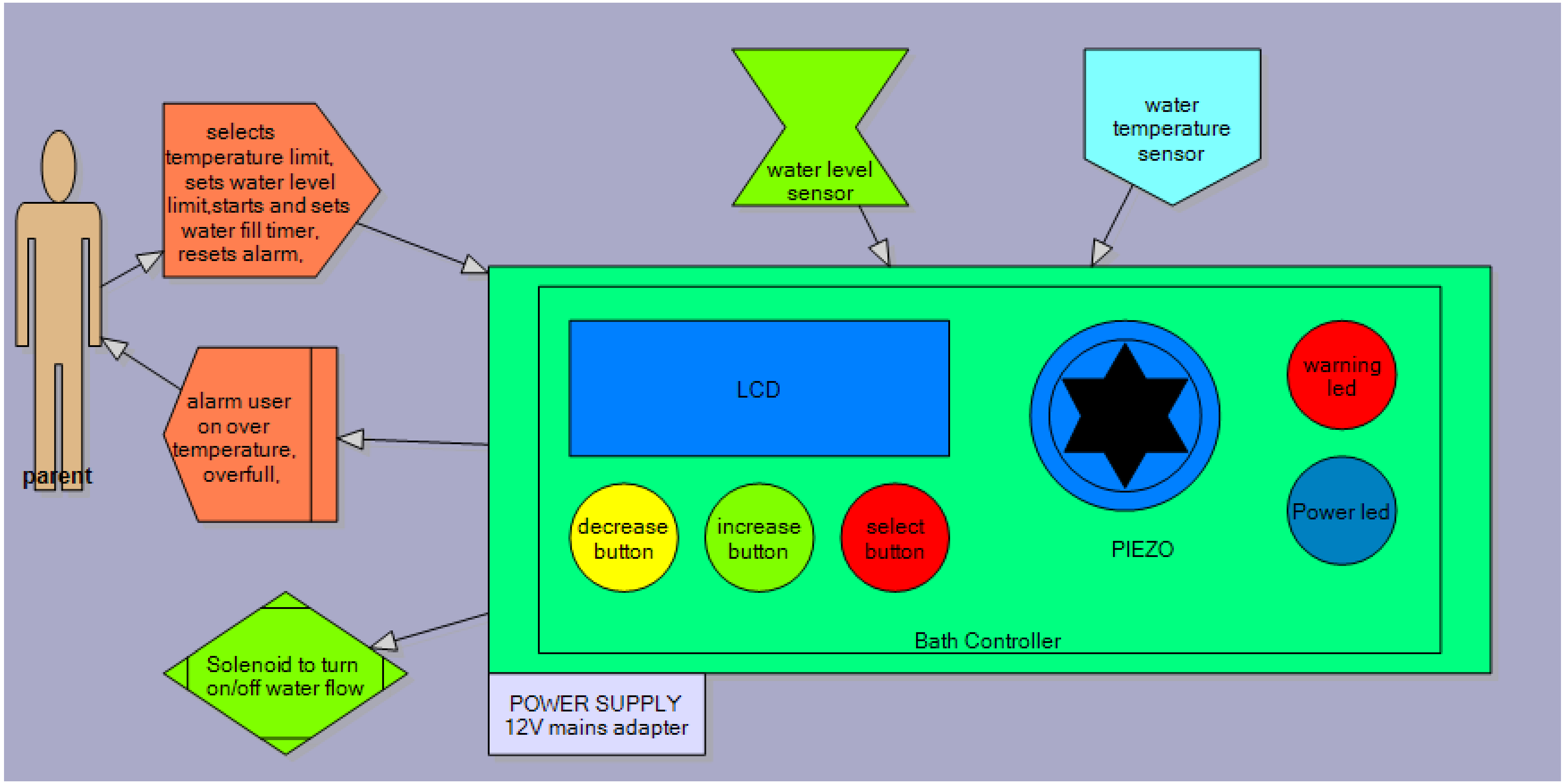
17.4.3 External sensors and actuators

Add any external environmental sensors or actuator outputs, (these are things not contained within the device itself, note that the devices are not hardware specific names like ‘LM35’ but ‘water temperature sensor’. These are useful for stakeholder consultations and identify the information the sensor gives.



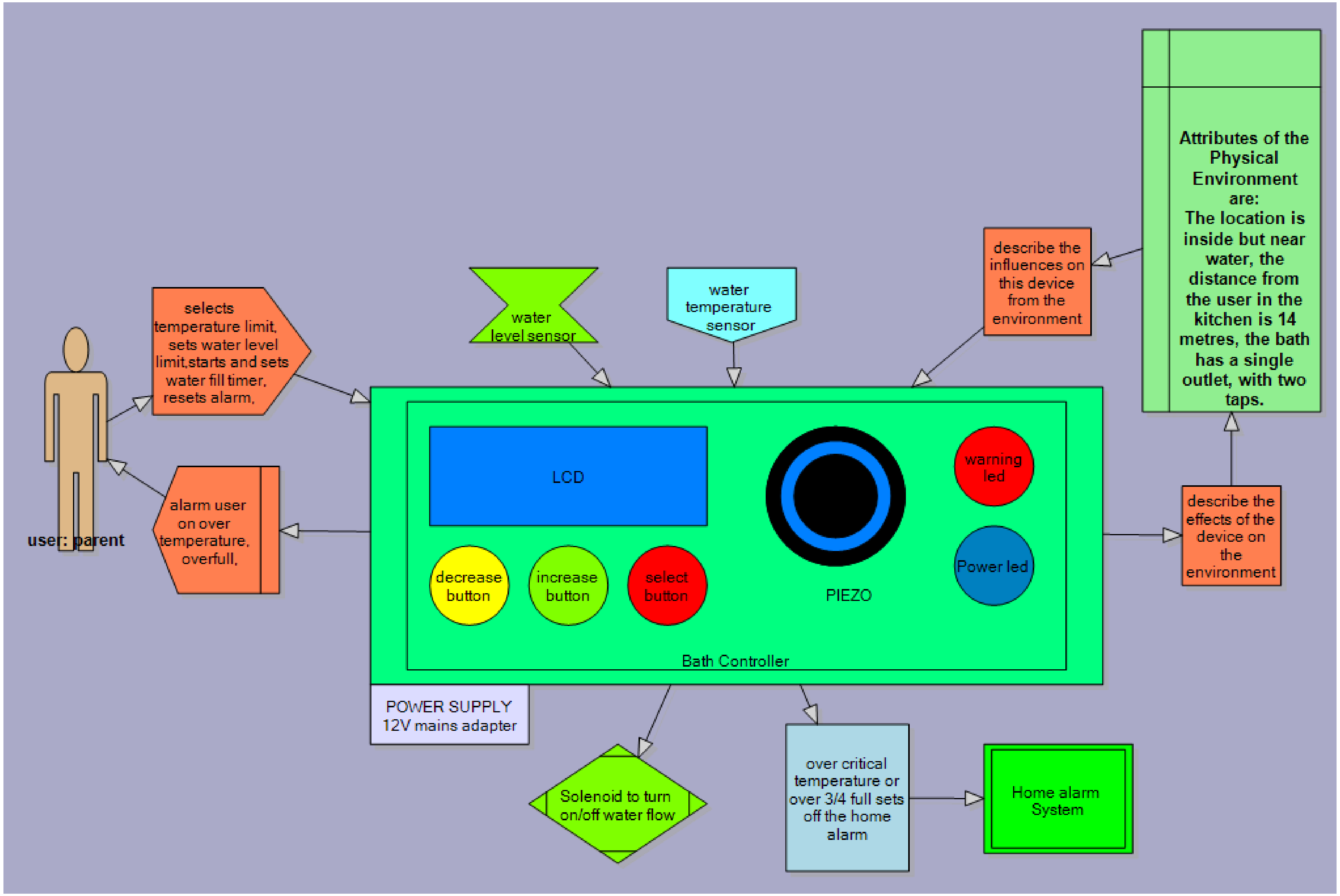
17.4.4 User interactions with the system (social environment)

Add a normal user - how will this user interact with the prototype (input things into it and be alerted by it).
Some systems have different categories or levels of users (normal and special e.g. cellphone have normal users and technicians which have access to extra features).



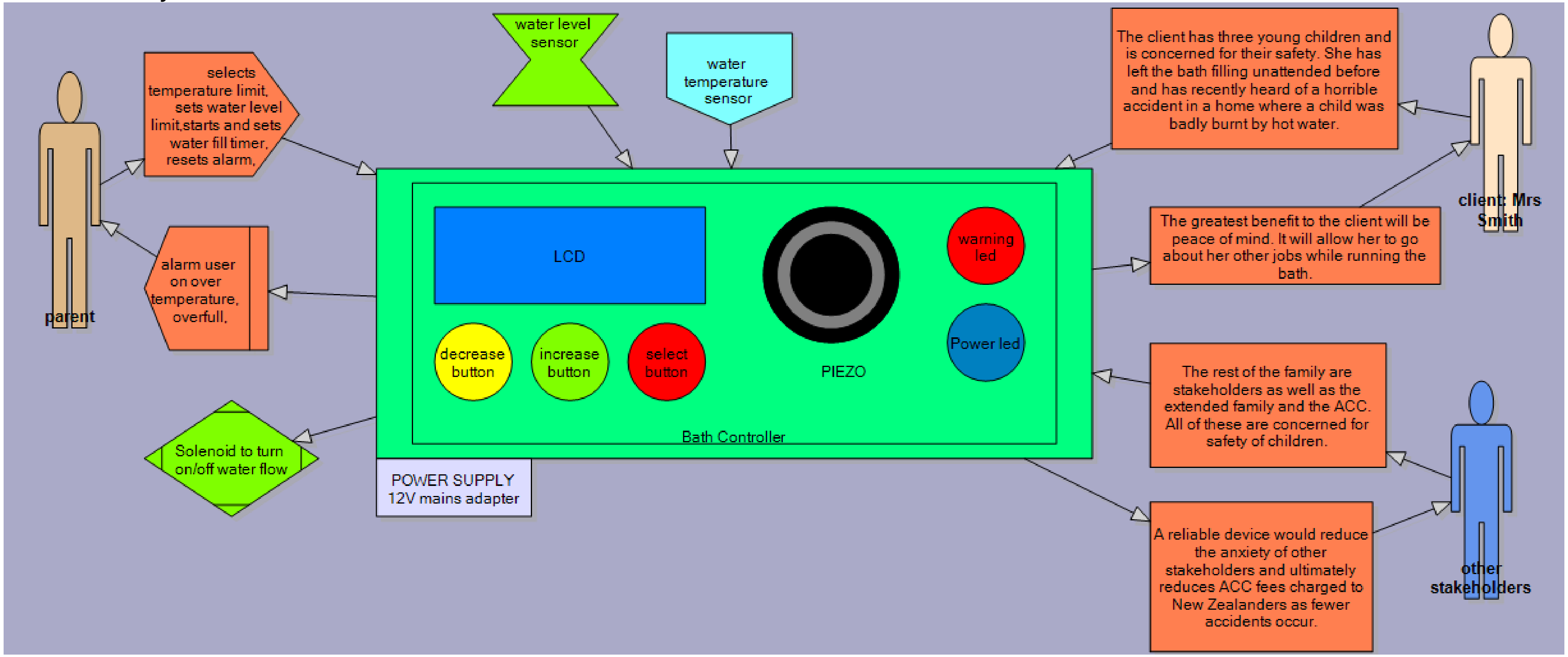
17.4.5 Physical Environment

Each product exists within with a physical world that forces certain things upon it, e.g. cellphones are kept in the pockets of clothes, what influence does this have on their design; also the cellphone must not have a negative effect on the clothing it is kept in. In the bathtub controller the device will be inside but near water.



17.4.6 Clients and stakeholders

Add stakeholders to the diagram, at this stage you can discuss the diagram with the client and other stakeholders to make sure that their needs have been fully documented.



If you change the design after speaking with the stakeholders keep a record of the old design or even start a new system context diagram within your project.

The reason for keeping ongoing changes will be to show you iterative (ongoing) planning and proof of stakeholder consultation.

17.4.7 Conceptual statement and physical attributes

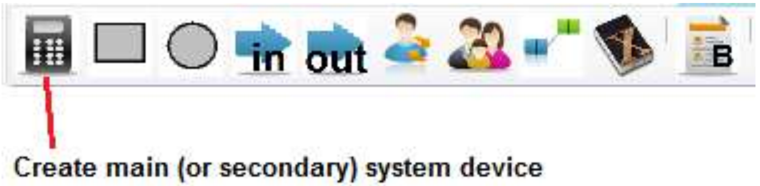
1. Write a conceptual statement, 3 sentences is usually enough
   a. Why is the device to be created?
   b. What is it?
   c. Why do it?
2. Describe the physical attributes (characteristics and features) of the system, the function of the system (functional attributes) need not be described here as they will be thoroughly covered in later drawings.

Conceptual statement:
The client has described her concerns about running a bath and being distracted away from it leaving a hazard for her small children.
This project will be a bath water temperature and water level controller that will allow the automatic monitoring and control of the bath.
This will provide the user with a safer environment for her family.

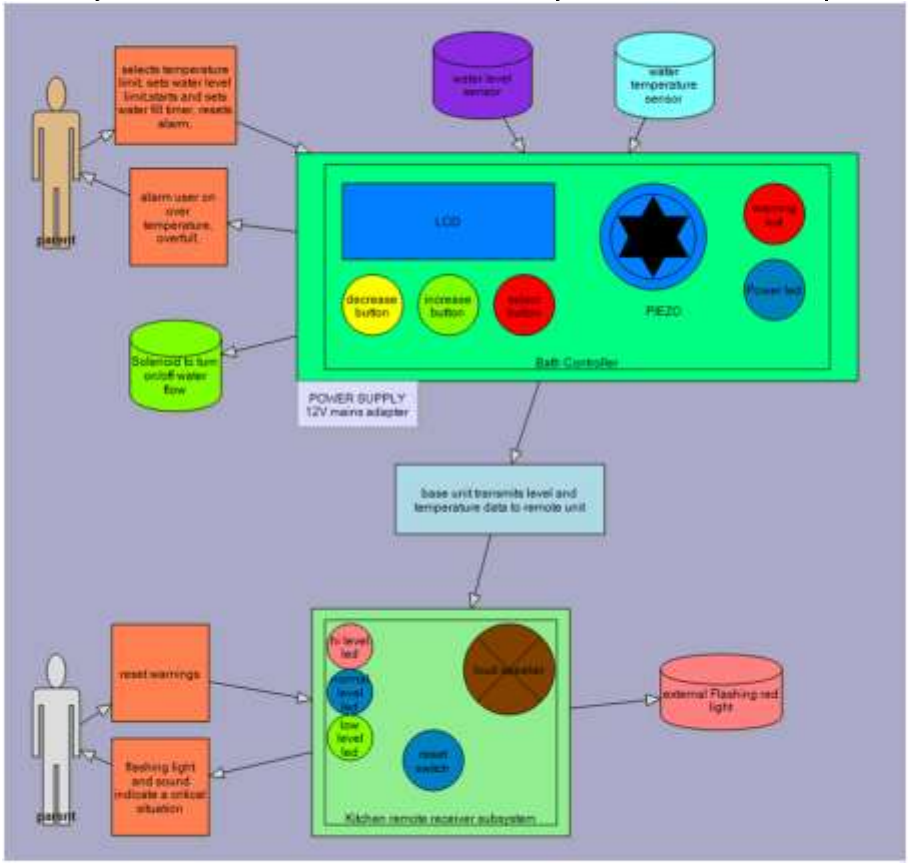
Physical attirbutes:
There is a power led to show the device is on
The user sets the amount of time the bath fills for, the water level to automatically shut off at and the water temperature alarm
The LCD displays: water termperature, water level, time left to fill
A red led and piezo alert the user of either overfill or over temperature
The warning LED flahes quickly at over temperature,
The piezo emits two different ypes of tones one for over temperature the other for over full
A solenoid controlled water valve turns off the water on over full

The base unit sends the temperature and fill status to a remote unit in the kitchen,

17.4.8 Secondary system devices



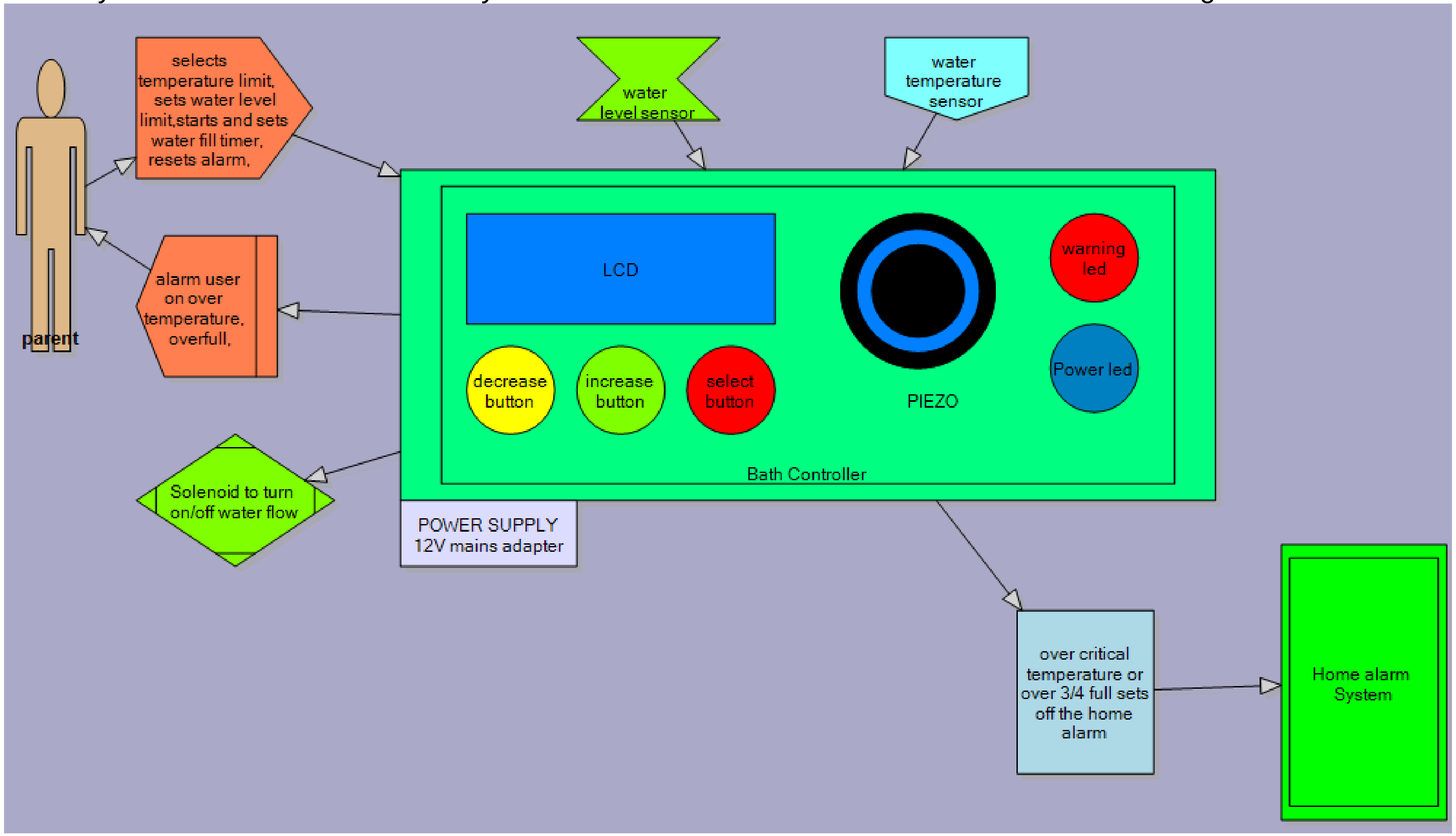
If the system includes external devices you have to develop as well then add another system device.



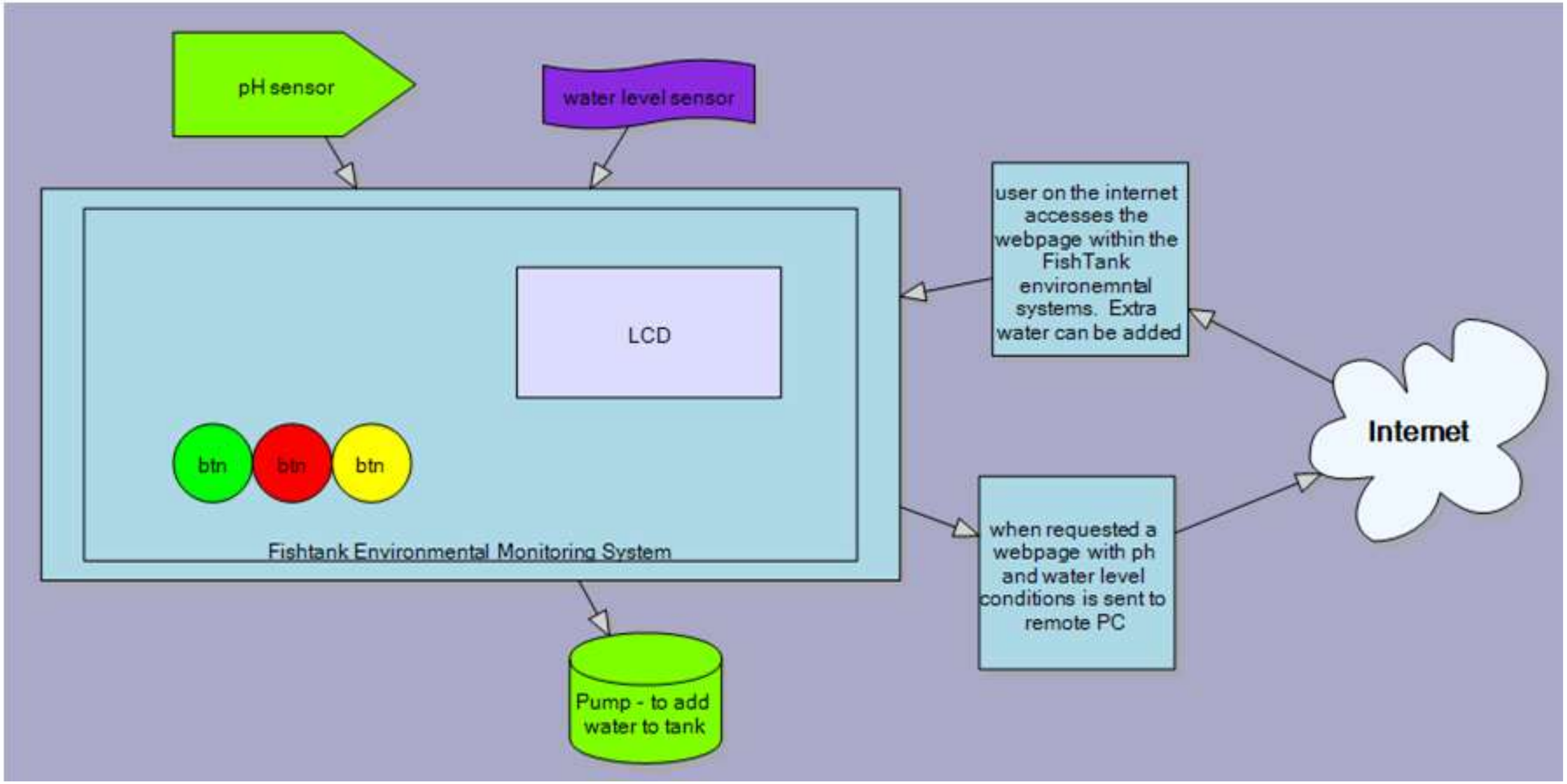
Take note that the communication between these two devices in this system is in one direction only. In some systems it will be bidirectional.

17.4.9 External system connections

Some systems interact with external systems such as if the bath tub controller was to send a signal to the home alarm system.



In this system context diagram a fish tank controller is linked to the internet.



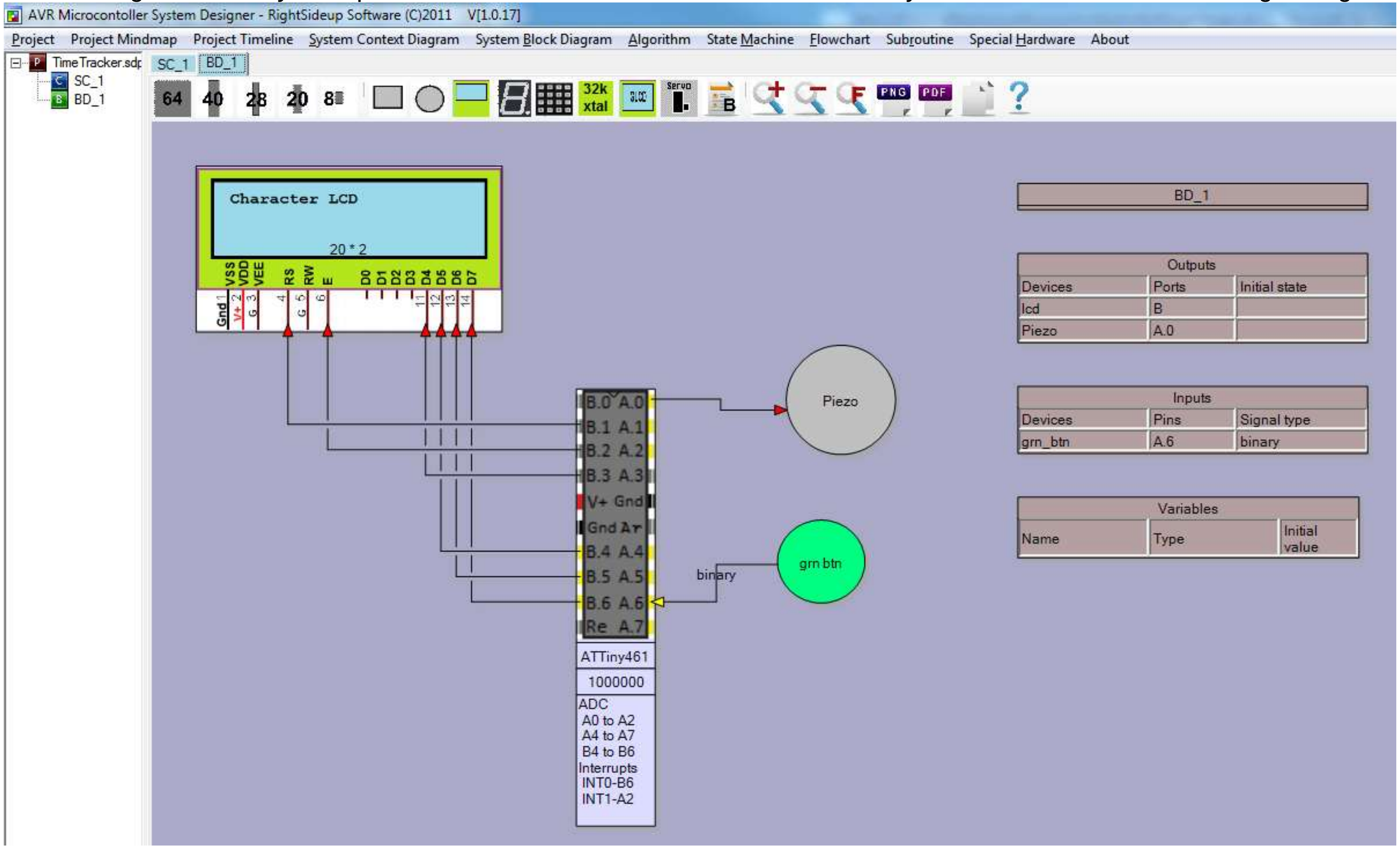
17.4.10 Export diagram to written documentation

Once the diagram is completed it can be checked with stakeholders for its accuracy, and then a written version of it can be produced by clicking on the ‘Written brief’ button in the toolbar.
This text document can then be expanded to include more detail

17.5 Block Diagram

In this diagram you need to develop the design of your product as a system itself.

A block diagram allows you to plan where interfaces will be connected before you do the connection, allowing changes to be made.



A system block diagram reveals the inner secrets of your prototype, using blocks to represent subsystems within the device.

Note that some specific detail is hidden and will be found in a schematic (circuit diagram).

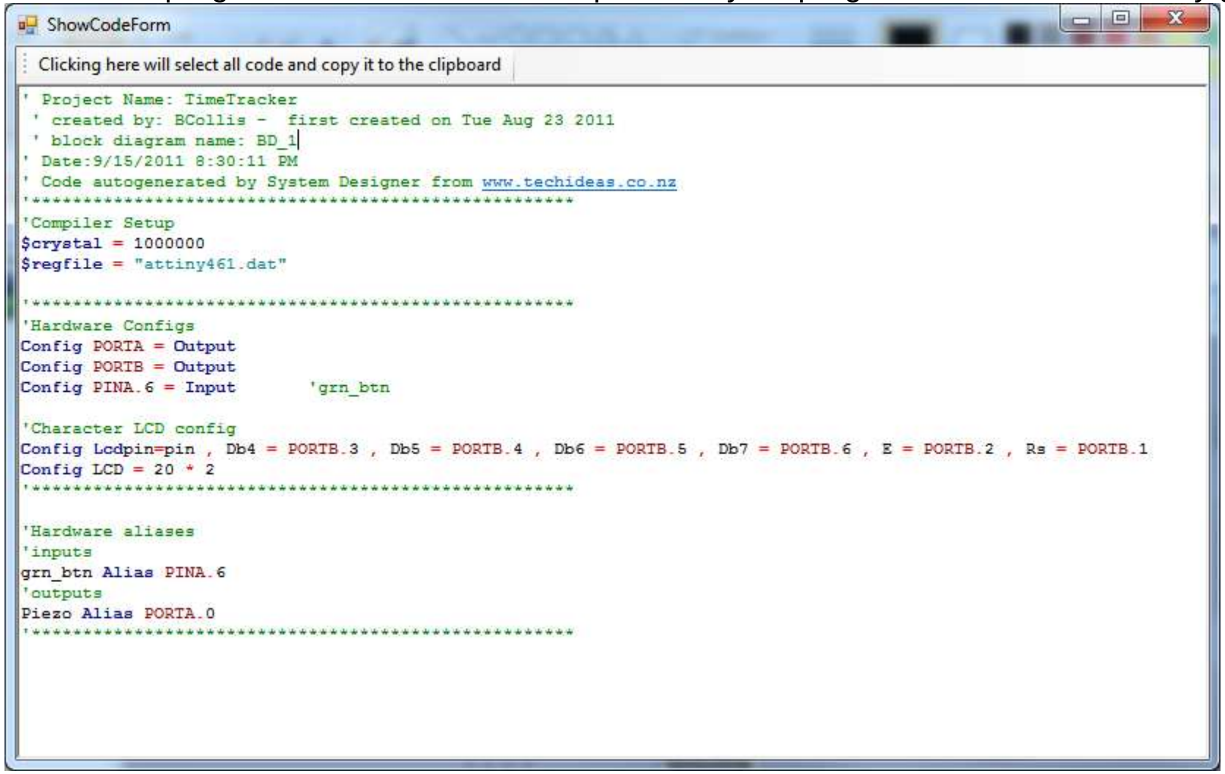
Start by adding the microcontroller you are using and right click on it to edit part numbers etc.

Then add things that it might have, an LCD, buttons, piezo, LEDs.
Use the rectangle and circular buttons and other shapes to add to the device.
Make sure that links between the micro and inputs/outputs are made in the right direction either coming in to the micro or out of it.

Blocks are used to represent parts of the circuit, so an LED subsystem is created by just adding a circle and calling it red led.
You do not show the current limit resistor, detail for that will be in the schematic.

Sometimes it may be a good idea to have two separate block diagrams, one for I/O (input and output) devices and a second for the power supply (it just makes it easier to separate the two parts of your design).

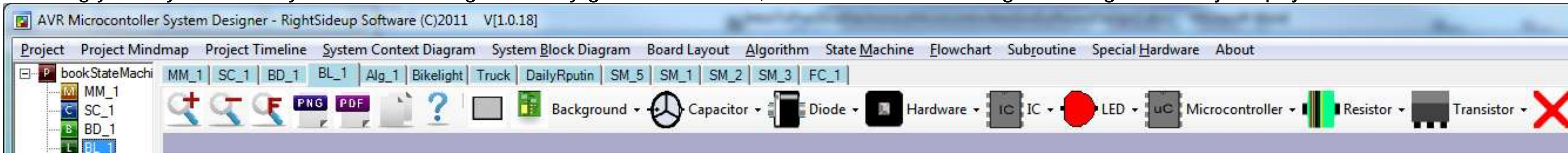
On the right hand side of the diagram are tables that list the outputs, inputs and variables that are created. These will be modified in later diagrams. The detail about port connections is useful in developing the setup program code for your program. By clicking on the Basic Code button in the toolbar the program code to form the setup area in your program will be automatically generated.




```
' Project Name: TimeTracker
 ' created by: BCollis -  first created on Tue Aug 23 2011
 ' block diagram name: BD_1
' Date:9/15/2011 8:30:11 PM
' Code autogenerated by System Designer from www.techideas.co.nz
'*****************************************************
'Compiler Setup
$crystal = 1000000
$regfile = "attiny461.dat"

'*****************************************************
'Hardware Configs
Config PORTA = Output
Config PORTB = Output
Config PINA.6 = Input       'grn_btn

'Character LCD config
Config Lcdpin=pin , Db4 = PORTB.3 , Db5 = PORTB.4 , Db6 = PORTB.5 , Db7 = PORTB.6 , E = PORTB.2 , Rs = PORTB.1
Config LCD = 20 * 2
'*****************************************************

'Hardware aliases
'inputs
grn_btn Alias PINA.6
'outputs
Piezo Alias PORTA.0
'*****************************************************
```

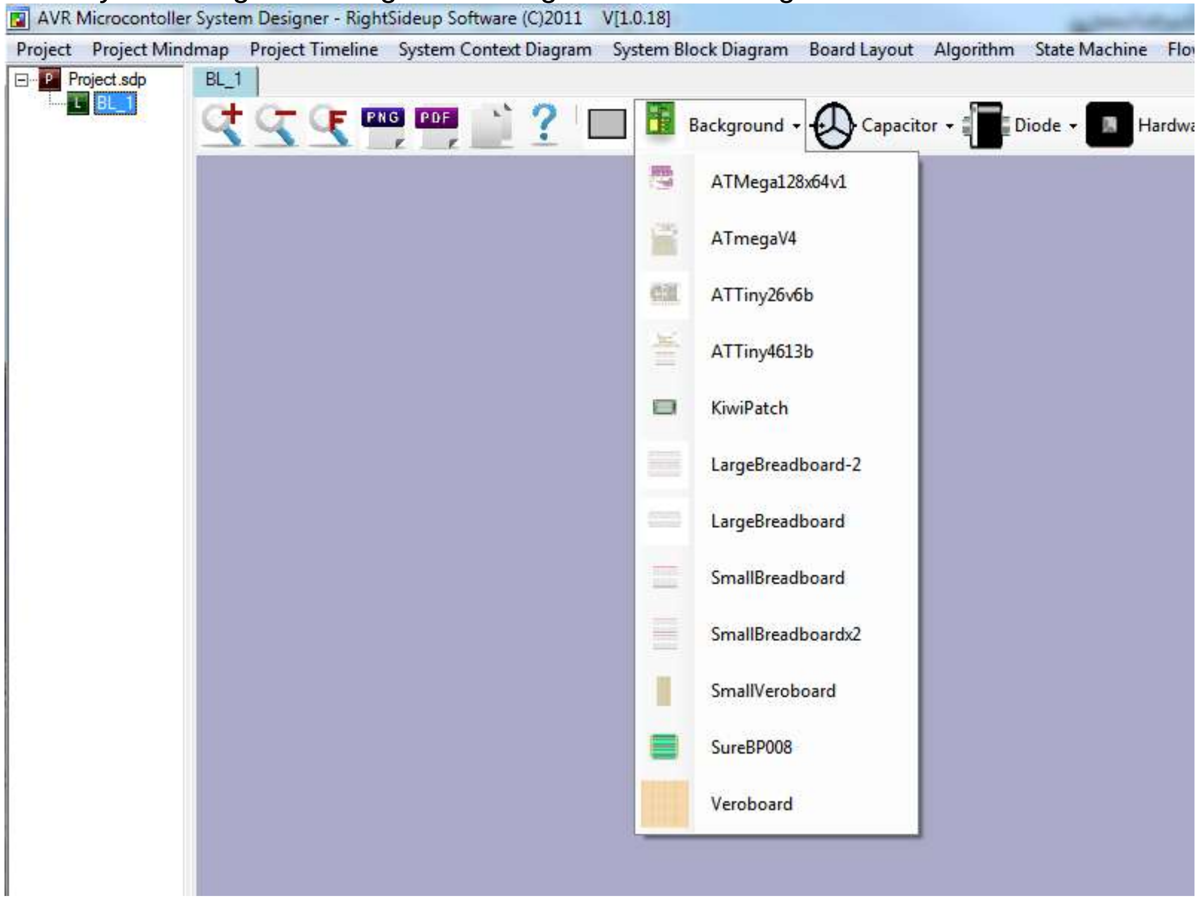
17.6 Board Layouts

If you will be using breadboard or an existing development board then completing a board layout drawing will be a useful planning tool.
(also If a schematic and PCB have been developed using a program such as Eagle then a board layout may be useful as you can create your own background using your layout from eagle and add I/O devices to it yourself)
Planning your layout before you start soldering is a really good use of time; it's a lot easier to change the diagram than your physical board!!



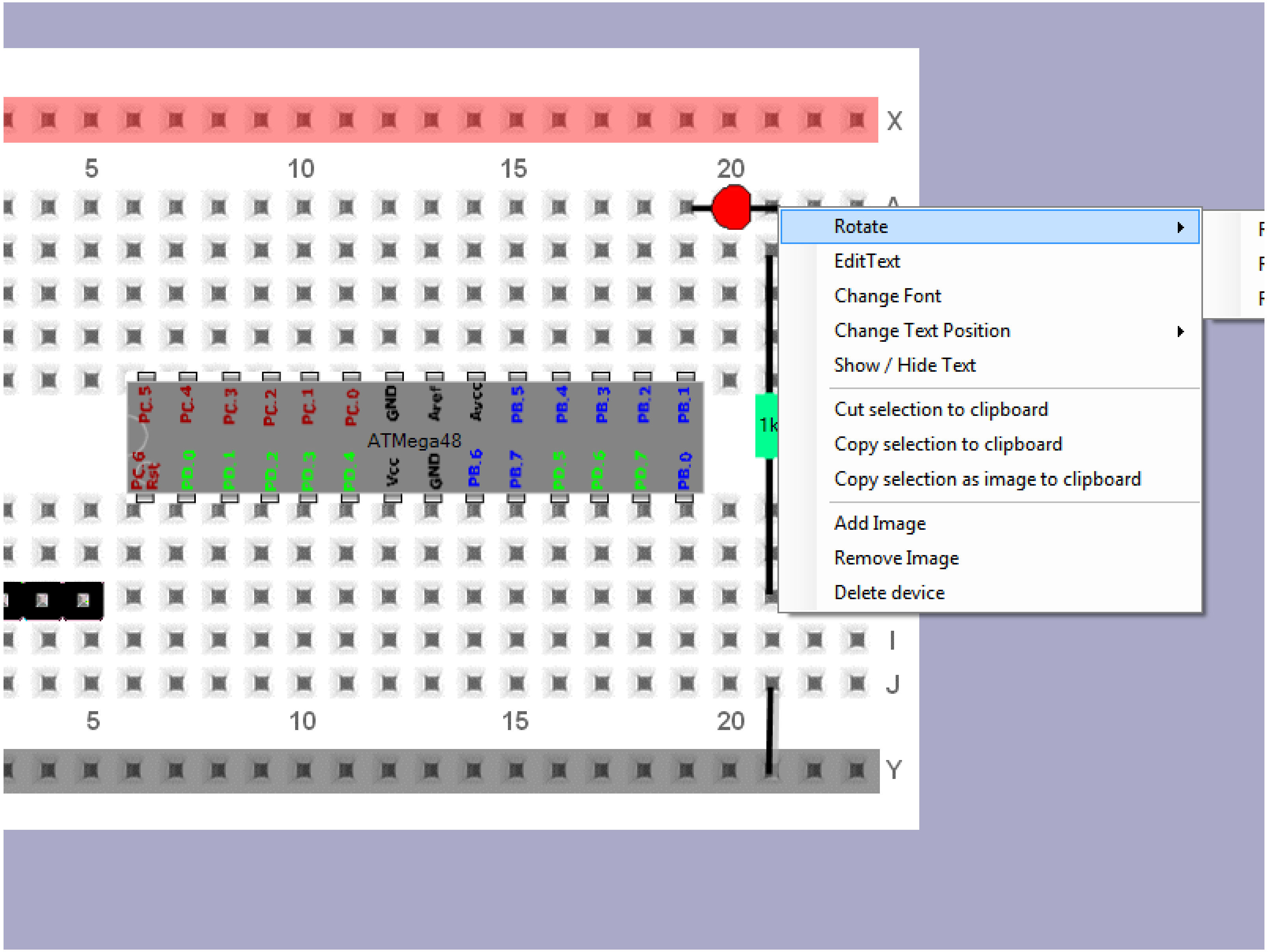
17.6.1 Backgrounds

Start by selecting the background image for the drawing.



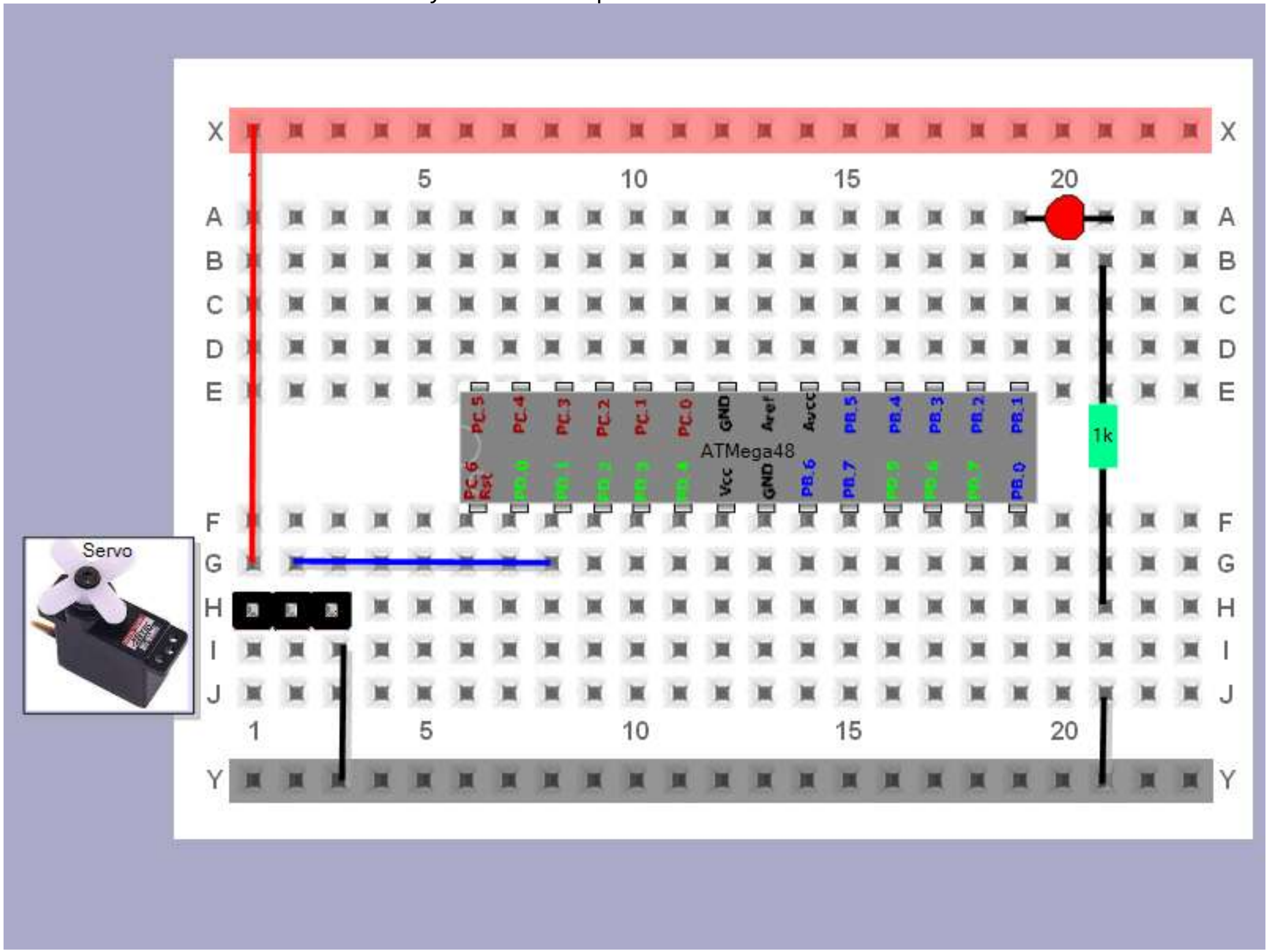
17.6.2 Add Components

Components can be added by clicking on them in the toolbar, then right clicking on them will allow you to change features.



17.6.3 Add your own pictures to the layout

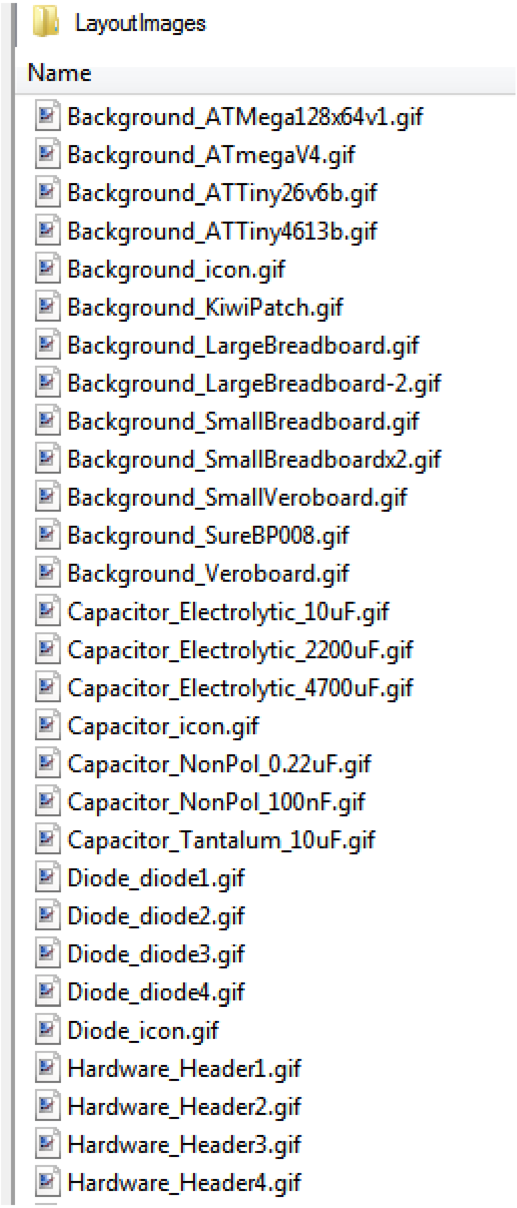
Here a servo has been added to the layout and the 3 pin header for it to connect to



17.6.4 Create your own backgrounds and components

The software is flexible enough for you to add your own backgrounds automatically.

Open the installation folder and find the folder named layout images



1. The images can only be of type .gif
2. There can be no spaces in the file names
3. Each category must have its own icon e.g. Background_icon.gif
    a. The naming must be with an underscore between the category name and the word icon
4. Each image must start with the same category name e.g. Background_SmallVeroboard.gif
    a. The name must be capitalized the same background is NOT the same as Background
    b. Again no spaces and the underscore separates the category from the image name
5. If a component is to have a text value it can be added to the component name with another underscore
    a. Capacitor_Electrolytic_10uF.gif
6. If you create a component type but forget to create the icon then it will not appear
7. If a component doesn't appear then check your spelling!
8. Have fun

17.7 Algorithm design

Algorithms are well defined instructions for getting the microcontroller to do something.
Pseudo-code is when an algorithm is written down using 'sort-of' program code commands.
Algorithms can also be designed using diagrams such as flowcharts or state machines as well as several others.

Why write an algorithm (either using pseudo-code or flowcharts)?
Because it helps you solve the problem and you need to do this before you start programming;
If you can solve the problem on pen and paper with an algorithm then you can write a program that will solve the problem.

Stage 1: determine the initial states of each output device.(right click on the row you want to moidify in the outputs table)

- e.g. will LEDS be on or off when the power is turned on
- what will a display show
- will a pump, motor or relay be on or off

Stage2: Data storage (cariables) – you need to specify these at this stage, before you start programming

- As well as reading inputs and controlling outputs your programs use, create and change data.
- What data will your program be processing?
- The data is stored inside the microcontrollers RAM (memory).
- A variable is the name given to a location in RAM.
- e.g. dim X_position as byte.
- This means dimension (allocate or set aside) 1 byte of ram and in the program and from now on the location can be called X_position

To make the use of ram as efficient as possible different variable types exist.
BIT (uses 1 bit of memory - values are either 1 or 0)
BYTE (uses 1 byte of memory - values can be any whole number from 0 to 255)
WORD (uses 2 bytes of memory - values can be any whole number from 0 to 65535)
INTEGER (uses 2 bytes of memory - values can be any whole number from -32,768 to +32,767)
LONG (uses 4 bytes of memory - values can be any whole number from -2,147,483,648 to +2,147,483,647)
SINGLE (uses 4 bytes of memory - values can be positive and negative fractions as small as 1.5x10^-48 up to 3.4x10^38
DOUBLE (uses 8 bytes of memory - values can be positive and negative fractions as small as 5.0x10^-324 up to 1.7x10^308)

STRING (uses ascii code to represent letters and digits, 1 character takes up one byte of ram)
e.g. dim my_name as string * 10 can store up to 10 characters only!
the largest string you can have is 254 characters

When choosing a variable to store data think about the right type to use (so as not to waste memory). But make sure you choose one that gives you what you need. Does your variable need to store both positive and negative numbers? Whole or fractional numbers? Big or small?
Variable names cannot have spaces, must start with a letter, can contain digits but not symbols.

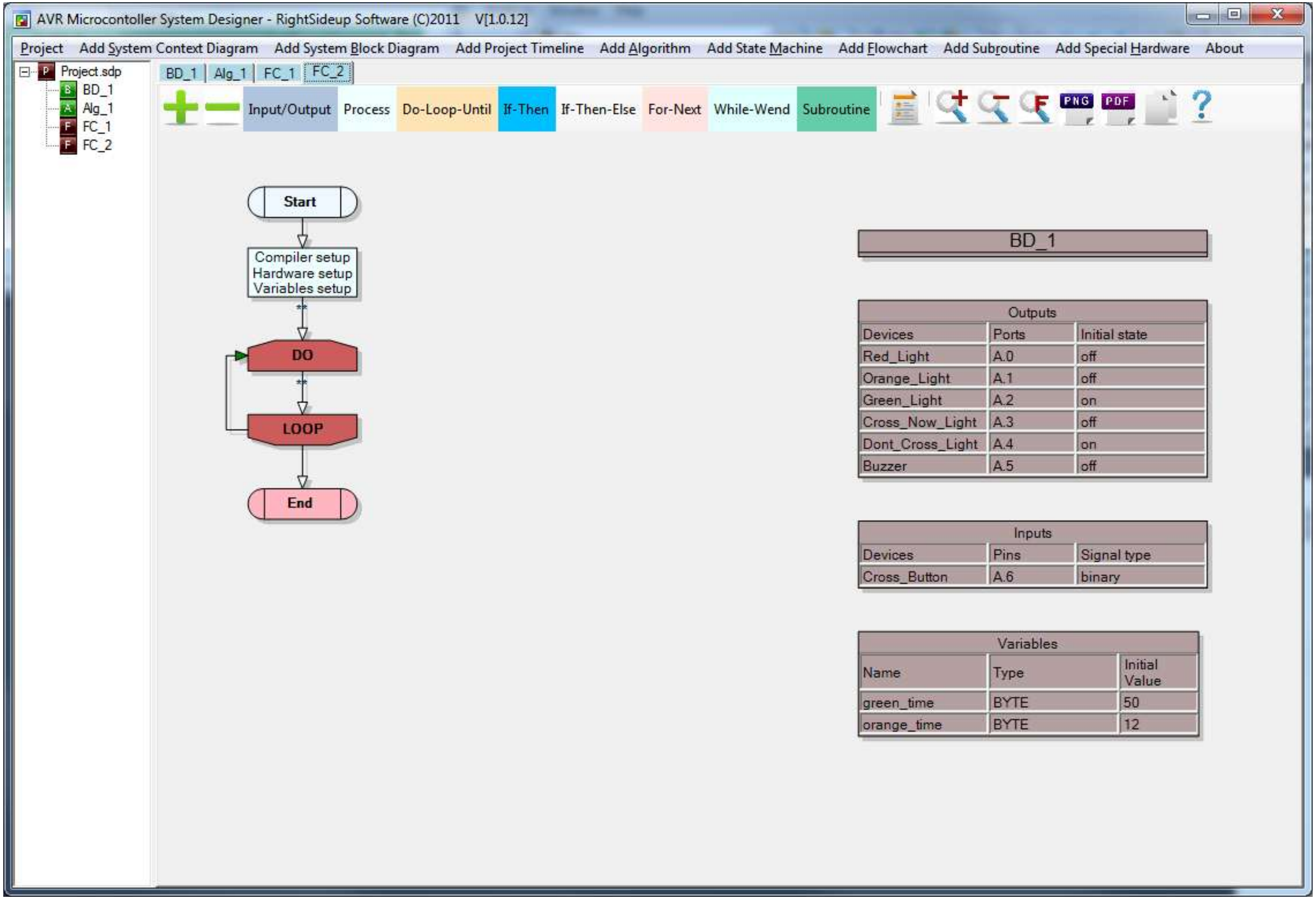
- Examples

  Temperature range is from 3 to 40 degrees - Dim outside_temperature as byte (is within the range 0 to 255)
  Temperature range is from -30 to 12 – Dim freezer_tempr as integer (needs to store negative numbers)
  Angle to move is from 0 to 360 – Dim move_angle as word (positive whole number from 0 t 65,535)
  Calculate the difference in milliseconds between 2 dates – Dim millsecs_diff as long
  Dividing numbers requires decimals, Dim percent_of _day as single

Stage3: Decomposition

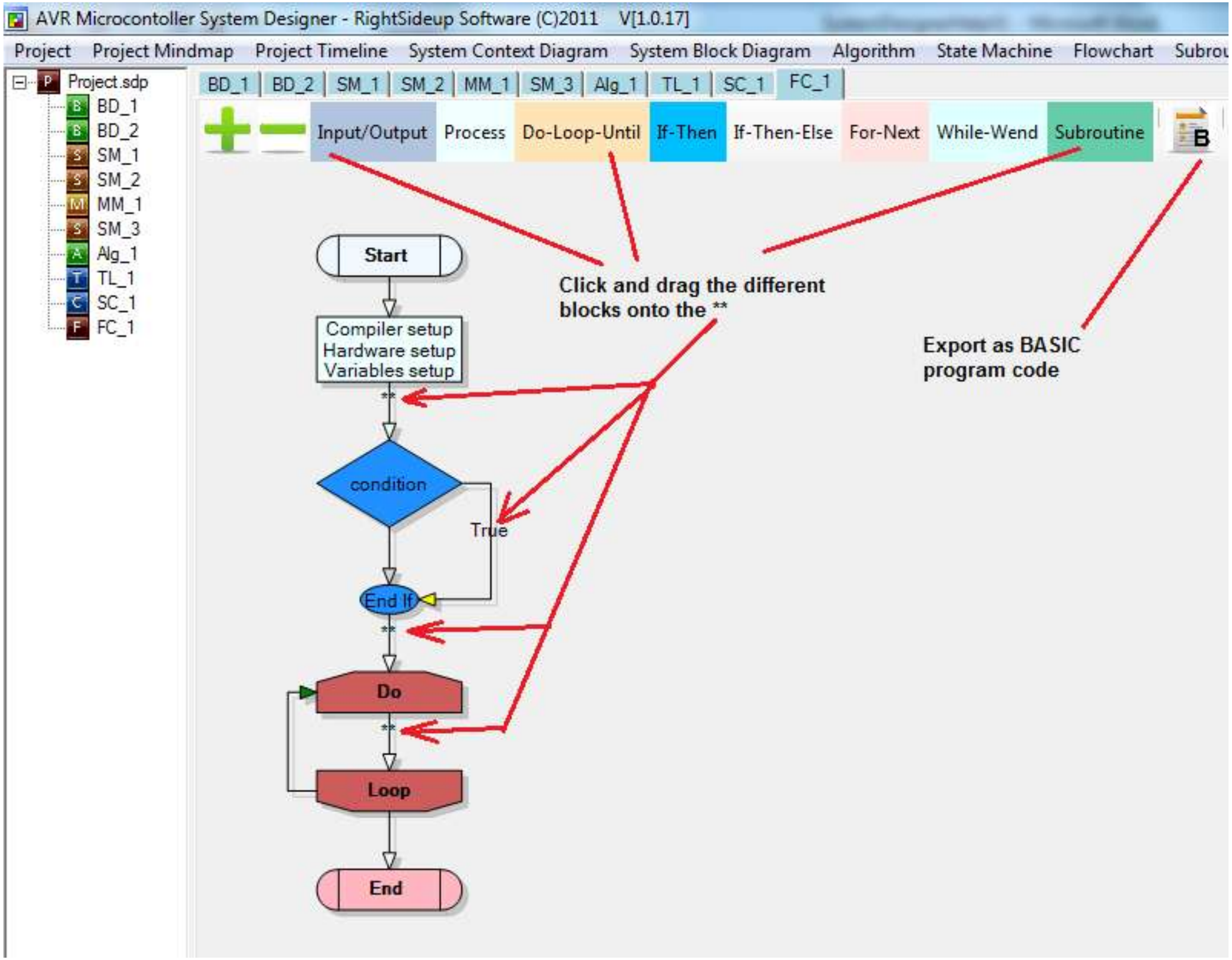
- Break up your problem into small solvable chunks
- The conceptual chunks should separate between: reading sensors, storing data, retrieving data, doing calculations, repeating actions and driving outputs, such as:
  - Read the temperature (input)
  - Close the door (output)
  - Keep the last 2 temperature readings (data storage)
  - Read the humidity (input)
  - Move the arm up (output)
  - Keep the last 2 humidity readings (data storage)
  - Read the distance from the infrared sensor (input)
  - Find out if we need to open or close the vent
  - If the second temperature readings minus than the first is > 2 then open the vent (calculation)
  - Find out how long to turn the fan on for (calculation)
  - Open the window (output)
  - Display the time (output)
  - Tilt the deck (output)
- In each calculation add some maths or logic about what your program will do using the IF, DO, WHILE, AND, OR, NOT
  - IF the blue switch is pressed AND NOT the red switch THEN make the led flash (logic)
  - IF the blue switch is pressed AND the end is NOT reached THEN X_position = X_position + 4 (calculation and logic)
- Repetition
  - DO increase X_position UNTIL end is reached (uses calculation)
  - WHILE the temperature > 5 flash the led (uses calculation)

17.8 Flowcharts



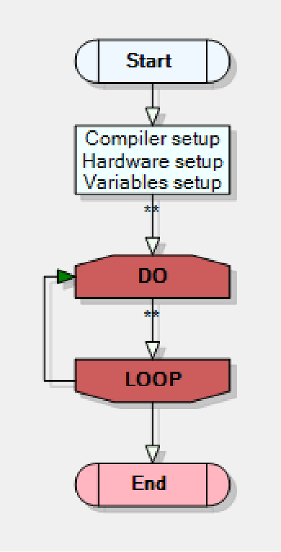
System Designer software includes a flowcharting feature which can be used to graphically explore programming concepts.

17.8.1 Drag and drop flowchart blocks



17.8.2 Beginning template

A new flowchart file starts with a template that is the minimum needed for a microcontroller program to function.



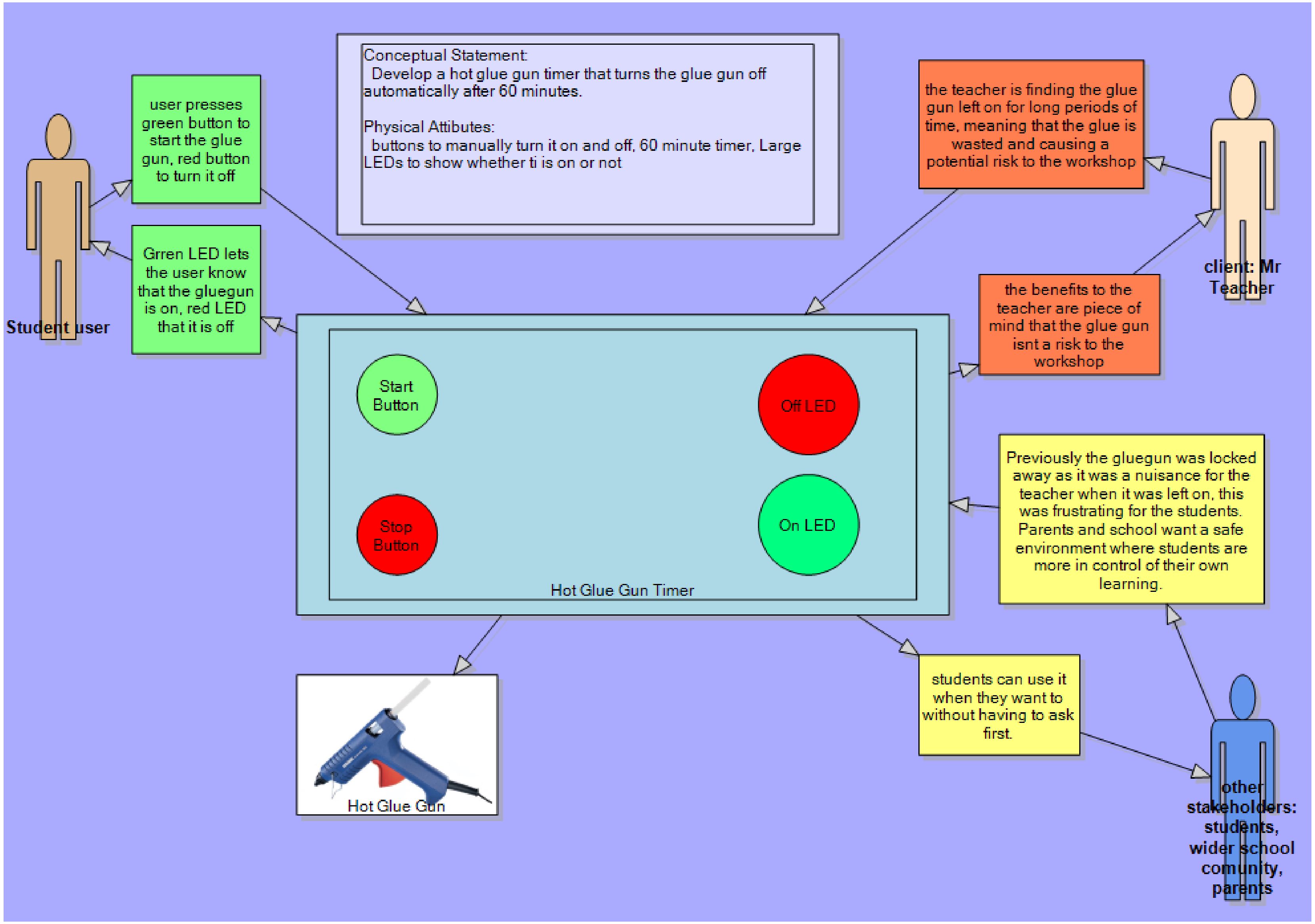
The first block sets up the Bascom Compiler to recognise your specific micro e.g.
$crystal =
$regfile =
As well as your specific hardware e.g.
Config Port...
And then variables your program will use to store data e.g.
Dim car_count as byte

A blank program can be generated by system designer, it looks like this:

```
' Project Name: Dice
 ' created by:
 ' block diagram name: BD_1
' Date:9/16/2011 10:51:15 AM
' Code autogenerated by System Designer from www.techideas.co.nz
'**************************************************************
'Compiler Setup
$crystal = 1000000
$regfile = "attiny461.dat"

'**************************************************************
'Hardware Configs
Config PORTA = Output
Config PORTB = Output
Config PINB.6 = Input       'red_sw
'**************************************************************

'Hardware aliases
'inputs
red_sw Alias PINB.6
'outputs
LED1 Alias PORTA.0
LED2 Alias PORTA.1
LED3 Alias PORTA.2
LED4 Alias PORTA.3
LED7 Alias PORTA.6
LED5 Alias PORTA.4
LED6 Alias PORTA.5
'**************************************************************
''------Program starts here --------
Do
Loop
'END
```

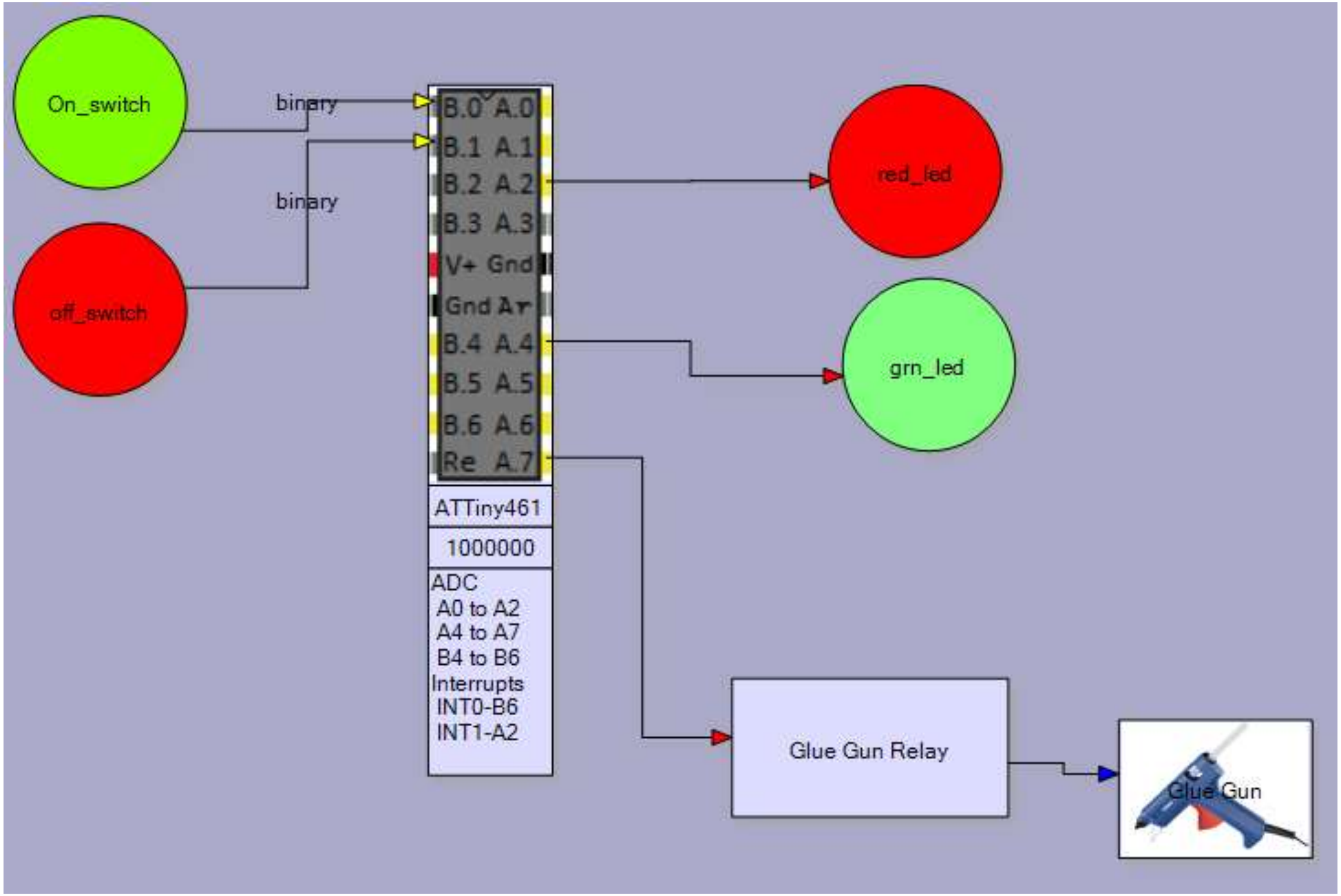
18 Example system design - hot glue gun timer

18.1 System context diagram



18.2 Hot glue gun timer block diagram

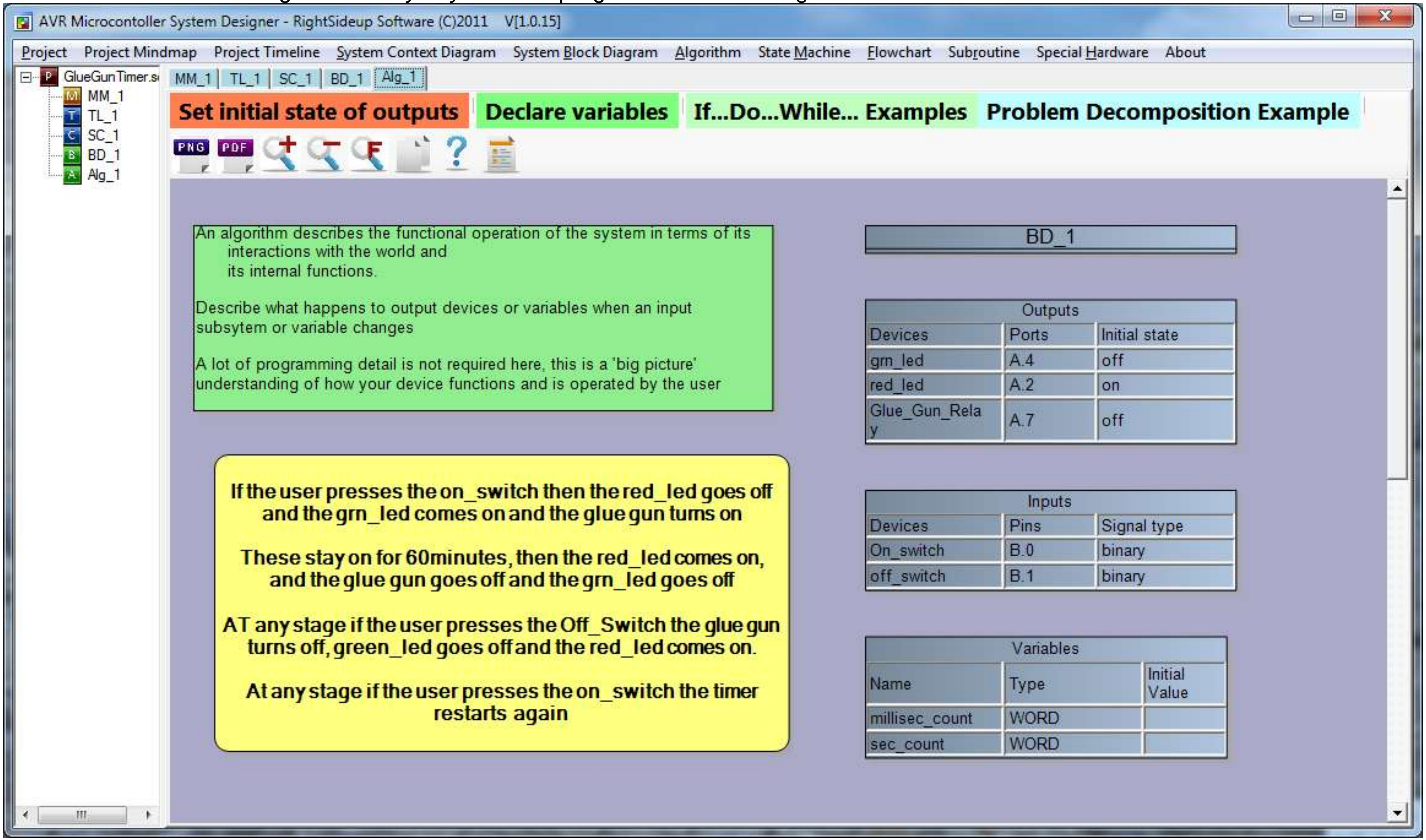
This reveals detail about the inner physical attributes or characteristics of your product, note it is not a full circuit or schematic diagram, but is still in some conceptual form. Make sure links between I/O devices and the microcontroller go inthe right direction.



18.3 Hot glue gun timer algorithm

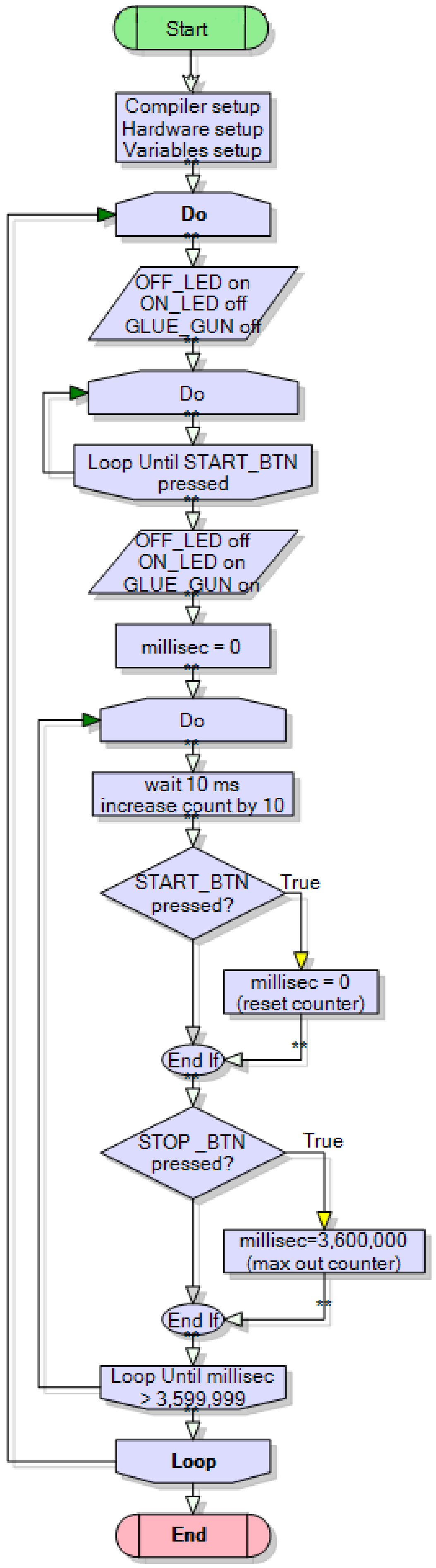
Here the functional attributes (characteristics and features) of the product are revealed.

1. Start by identifying the initial states of any outputs – on or off in this situation
2. Describe the algorithm – how the device responds to user input and computations it must carry out.
3. At the same time begin to identify any data the program will need and give these variables useful names.



18.4 Hot glue gun timer flowchart

A flowchart is a visual algorithm for a simple system



Intepreting the algorithm:

Initially:
OFF_LED = on
ON_LED = off
GLUE_GUN = off

Wait until START_BTN is pressed

OFF_LED = off
ON_LED = on
GLUE_GUN = on

Zero the counter

Wait 10 ms
Increase counter by 10

Check the switches

-If start pressed reset count to 0 to restart the timing for another hour

-If stop pressed set count to max so looping stops

Repeat until the time has reached 1 hour

18.5 Hot glue gun timer program

```
' GlueGunTimerVer1.bas
' B.Collis 1 Aug 2008
' 1 hour glue gun timer program
' the timer restarts if the start button is pressed again
' the timer can be stopped before timing out with the stop button
'compiler setup
$crystal = 1000000
$regfile = "attiny26.dat"
'hardware setup
Config Porta = Output
Config Portb = Output
Config Pina.2 = Input
Config Pina.3 = Input
'Alias names for the hardware
Gluegun Alias Porta.5                    'names easy to read and follow
Offled Alias Porta.6
Onled Alias Porta.7
Startbutton Alias Pina.2
Stopbutton Alias Pina.3
'Dimension variables
Dim Mscount As LONG                  'need a variable that can hold a really big number
Const Max_mscount = 3600000
'program starts here
Do
    Set Offled
    Reset Onled
    Reset Gluegun                     'initially off

    Do                                'wait for start button press
    Loop Until Startbutton = 0

    Reset Offled
    Set Onled
    Set Gluegun                       'glue gun on
    Mscount = 0                       'start counting from zero

    'note the use of a do-loop rather than a for-next to count the repititions
    'we do this because it is unknown when the user will push a button and reset/restart the count
    Do
             Mscount = Mscount + 10           'add 10 to milliseconds
             Waitms 10
             If Startbutton = 0 Then          'Check Switch
                Mscount = 0                   'reset time to zero, so restart timer
             End If
             If Stopbutton = 0 Then           'Check Switch
                Mscount = Max_mscount         'set time to max, so cancel timing
             End If
     Loop Until Mscount > Max_mscount     'loop 3,600,000 times unless user changes mscount

Loop
```

Notes:

1. We wait 10mS – we could wait 1MS however 10mS is not so long that we would miss the switch press
2. Tthere is no debouncing of the switches, this is not really needed in this program because repeat switch presses don't cause any problems for us.

19 Basic interfaces and their programming

Having completed some introductory learning about interfacing and programming microcontrollers it is time to learn more detail about interfacing.

Switches

Analogue to digital conversion using

LDRS

and Thermistors

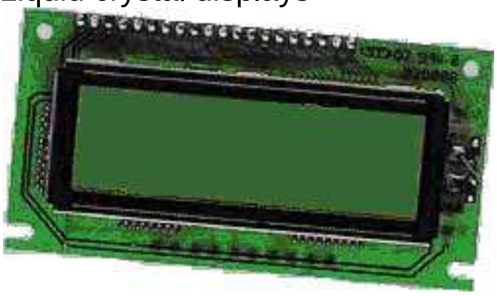
Boosting the power output

to make sound

and drive small inductive loads

Parallel interfaces to

Liquid crystal displays



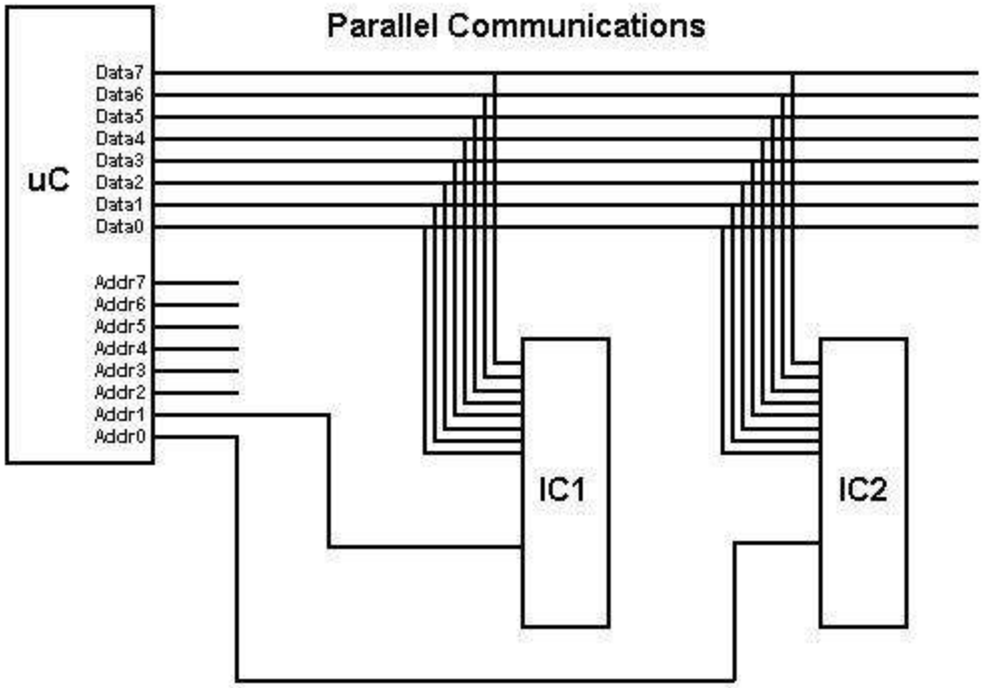
and multiple seven segment displays

Serial interfaces to
Real Time Clocks

and computer RS232 ports

19.1 Parallel data communications

Both internal and external communications with microcontrollers are carried out via **buses**, these are groups of wires. A bus is often 8 bits/wires (byte sized) however in systems with larger and more complex microcontrollers and microprocessors these buses are often 16, 32 or 64 bits wide.



Communication is carried out using 8 or more bits at a time. This is efficient as an 8 bit bus can carry numbers/codes form 0 to 255, a 16 bit bus can carry numbers/codes from 0 to 65,535 and 32 bits can carry numbers/codes from 0 to 4,294,967,295. So data can move fairly fast on a parallel bus.

Parallel communication is often used by computers to communicate with printers, because of this speed. Only one printer can be connected to the parallel port on a computer, however within the computer itself all the devices on the bus are connected all the time to the data bus. They all share access to the data, however only the device that is activated by the address bus wakes up to receive/send data.

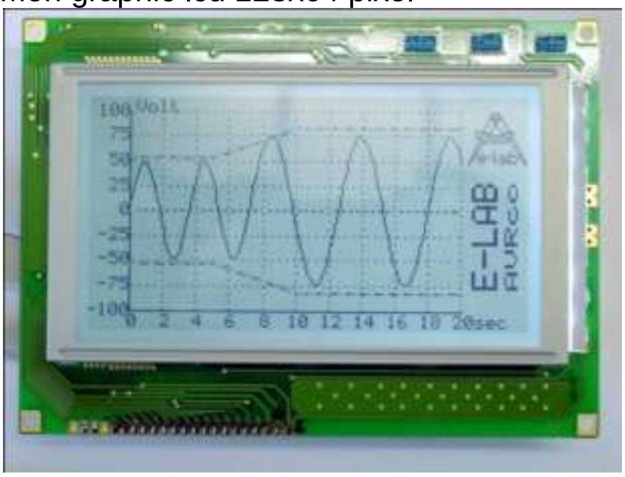
19.2 LCDs (liquid crystal displays)

There are a great many different types of LCD available, we describe them by there various attributes. Colour/Monochrome, alphanumeric/graphic. Some LCDs which are made for specific purposes with fixed Characters such as these two.

Monochrome alphanumeric LCD wirh no backlight

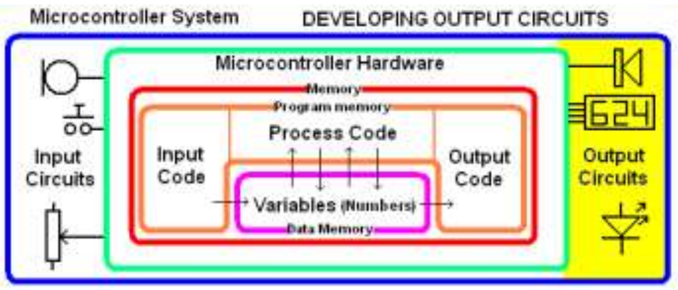
4 line alpahanumeric mono LCD with backlight

Mon graphic lcd 128x64 pixel



Colour graphic LCD 320x128pixel.

19.3 Alphanumeric LCDs



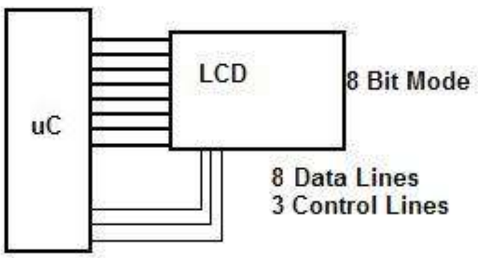
One of the best things about electronic equipment nowadays are the alphanumeric LCD displays these are simple single, double or 4 line displays for text and numbers. These displays are becoming cheaper and cheaper in cost, we buy them in bulk from China using www.alibaba.com. The LCD is a great output device and with Bascom so very easy to use. They fit the need for student learning in technology education very nicely.

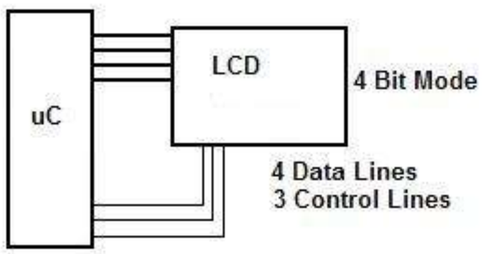
Some common commands are

- **cls** - clear the screen
- **LCD "Hello"** - will display hello on the display
- **locate y,x** - line and position on the line of the cursor (where text will appear)
- **Cursor OFF** – hide the cursor (still there but invisible)
- **LCD temperature** – will display the value in the variable temperature on the display

Connecting an LCD to the microcontroller is not difficult.
There are 14 or 16 pins on the LCD

1. 0V
2. +5V
3. Contrast
4. RS - register select
5. R/W - read/not write
6. E - Enable
7. D0
8. D1
9. D2
10. D3
11. D4
12. D5
13. D6
14. D7
15. Backlight + (optional)
16. Backlight 0V (optional)





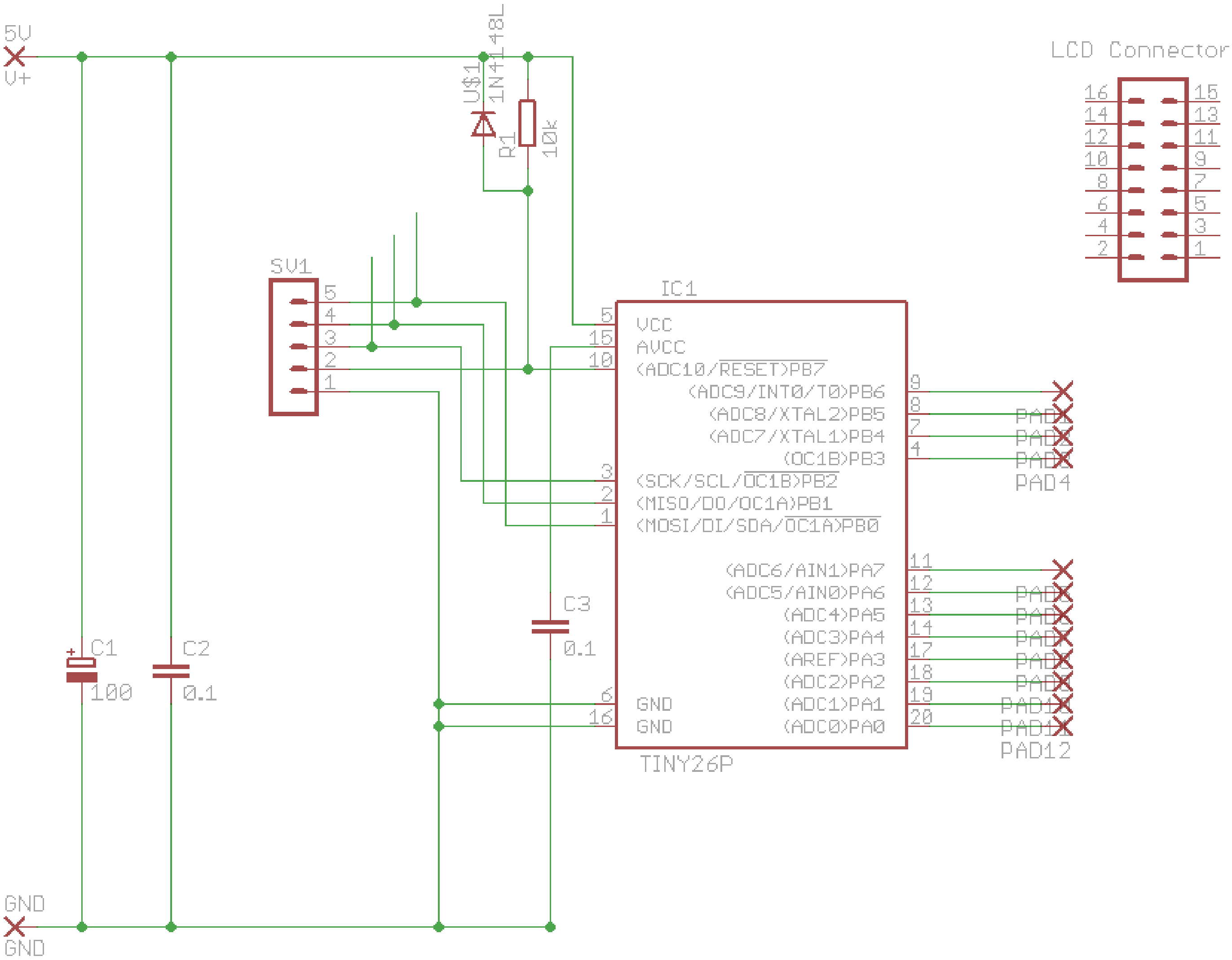
Most LCDs are set up so that they can communicate in parallel with either 4 bits or 8 bits at a time. The faster system is 8 bits as all the data or commands sent to the LCD happen at the same time, with 4 bit operation the data/command is split into 2 parts and each is sent separately. Hence it takes twice as long.

Apart from the 4 data lines another couple of lines are necessary, these are control lines, RS , R/W, E. When using Bascom the **R/W line is connected permanently to griund**, and the other two lines need to be connected to the micro. The advantage of 4 bit operation is that the LCD uses only 6 I/O lines in total on the micro. At the current time the **contrast line can be connected to ground** as well.

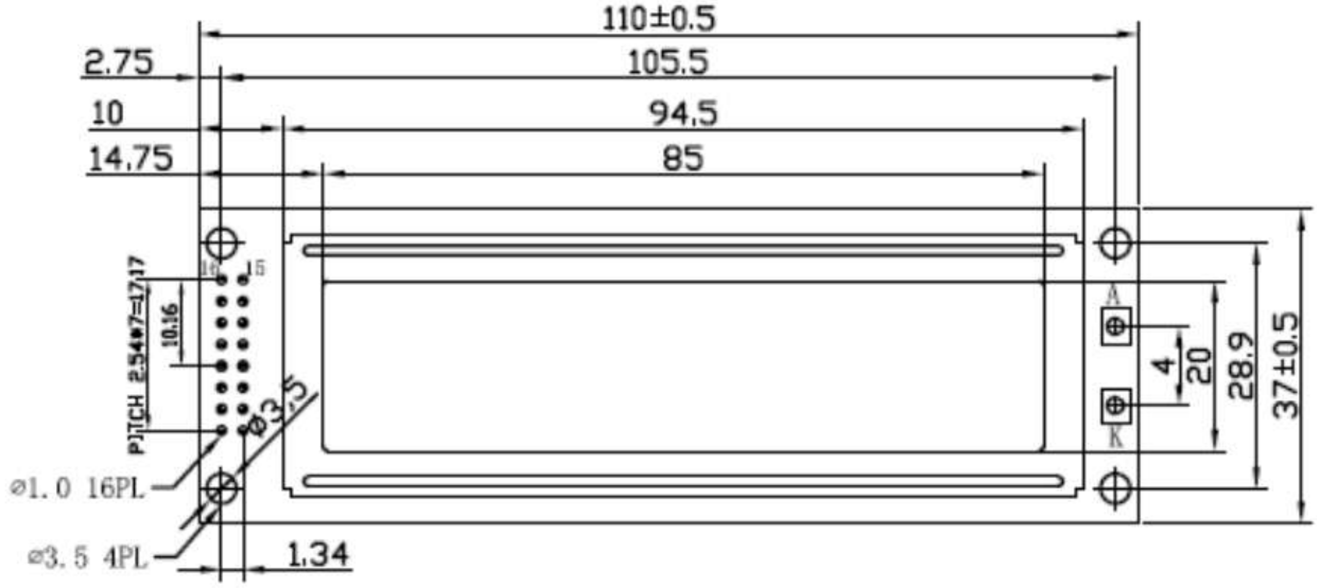
19.4 ATTINY26 Development PCB with LCD

Although a breadboard was useful earlier for some introductory learning about connecting a microcontroller and interfacing simple components such as LEDs and switche;, trying to use a breadboard to connect an LCD is not easy, you just end up with too many wires that fall out of the breadboard if the LCD gets moved. It is more useful to have a circuit board of some description. Here is a development PCB that was designed to be useful for students when building their circuits. It makes use of a standard 2 line 20 character alphanumeric LCD. It has a 16 way connector (although the LCD used has no backlight so only 14 connections are used)

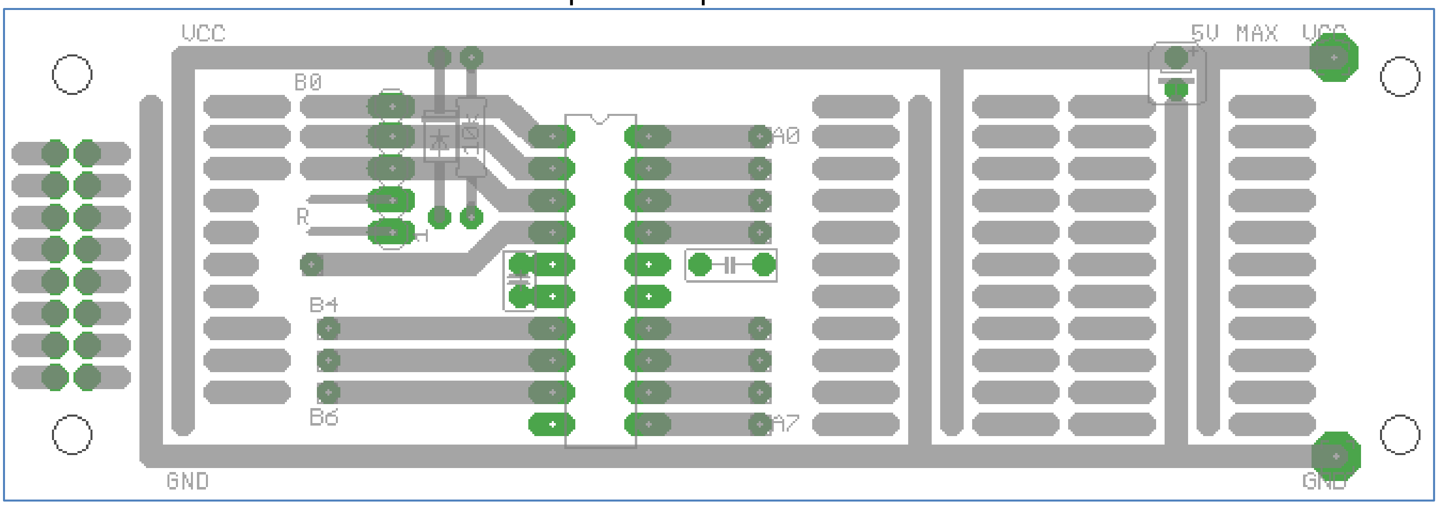
In the schematic we have connected the power to the LCD but not actually connected the control lines. These are left unconnected so that students become familiar with the connections, it also made the PCB much easier for students to solder not having so many thin tracks.



The physical pcb is designed around the physical dimensions of the LCD, so that the LCD and board can be bolted together.

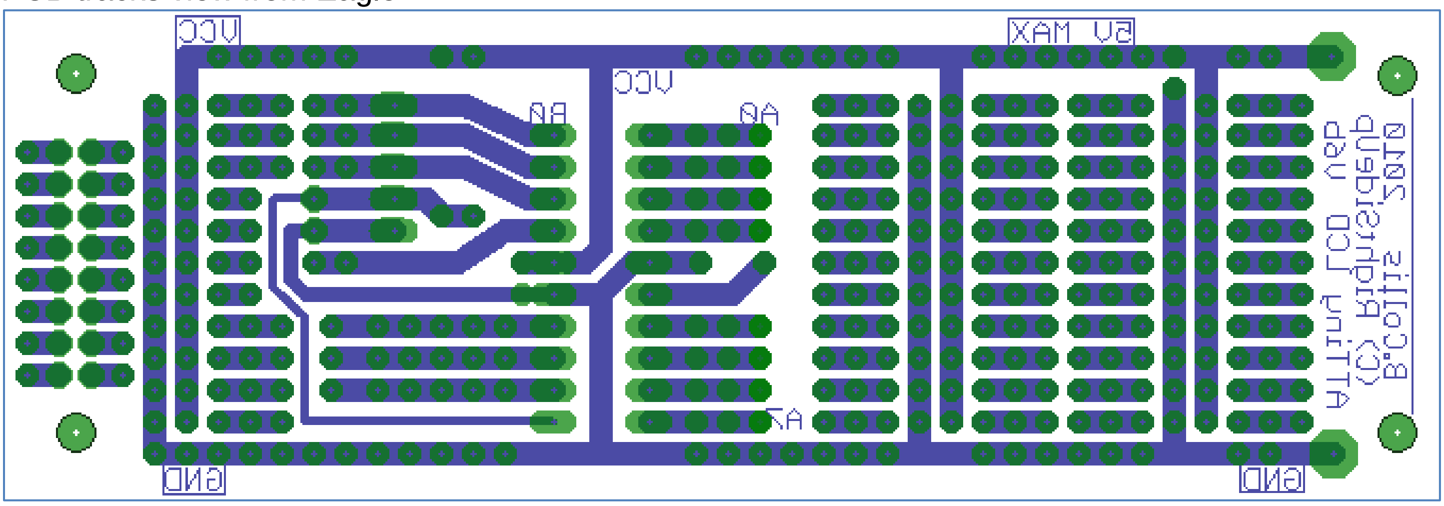


Top or Component view



Take care when wiring the header pins (connector) for the LCD as he polarity for the power must be correct, there is a an area for prototyping other circuits on the board

PCB tracks view from Eagle

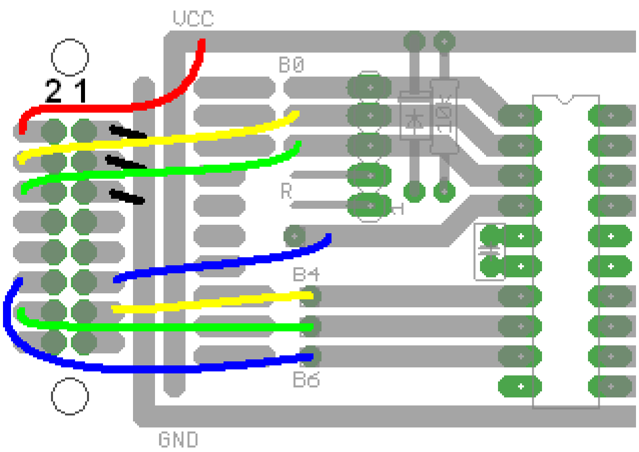


19.5 Completing the wiring for the LCD

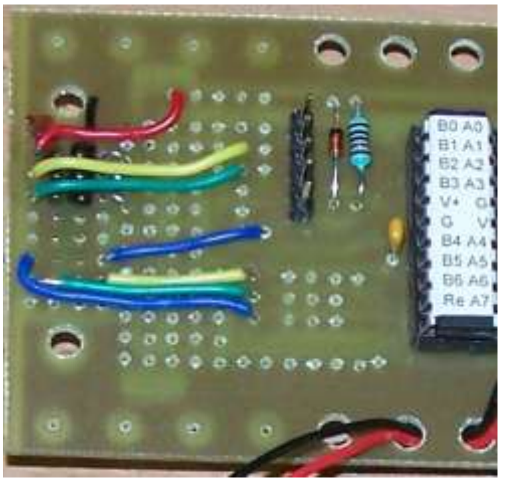
Here are the details for the specific Sure Electronics LCD we are using. Highlighted are the 6 data and control connections we need to make (note that pins 1,2,3,5 are already connected via PCB tracks). The two control lines are RS(register select) and Enable. The 4 data lines are DB4 to DB7.

Pin NO.	Symbol	Level	Description
1	VSS	0V	Ground
2	VDD	5.0V	Supply voltage for logic
3	VO	--	Input voltage for LCD
4	RS	H/L	H : Data signal, L : Instruction signal
5	R/W	H/L	H : Read mode, L : Write mode
6	E	H, H → L	Enable signal for KS0076
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB7	H/L	Data bit 7
15	NC	--	No connection
16	NC	--	No connection

Looking at the development board it can be seen that there are already pads for the LCD,
The 6 connections have been added on the diagram below.



The order the 6 lines are connected from the LCD to the micro does not matter as long as 1. They are on the same port and 2. the order used matches the configuration command in Bascom.



To program the LCD using Bascom we need to add two lines of configuration program code, and then use specific commands to make the display show something

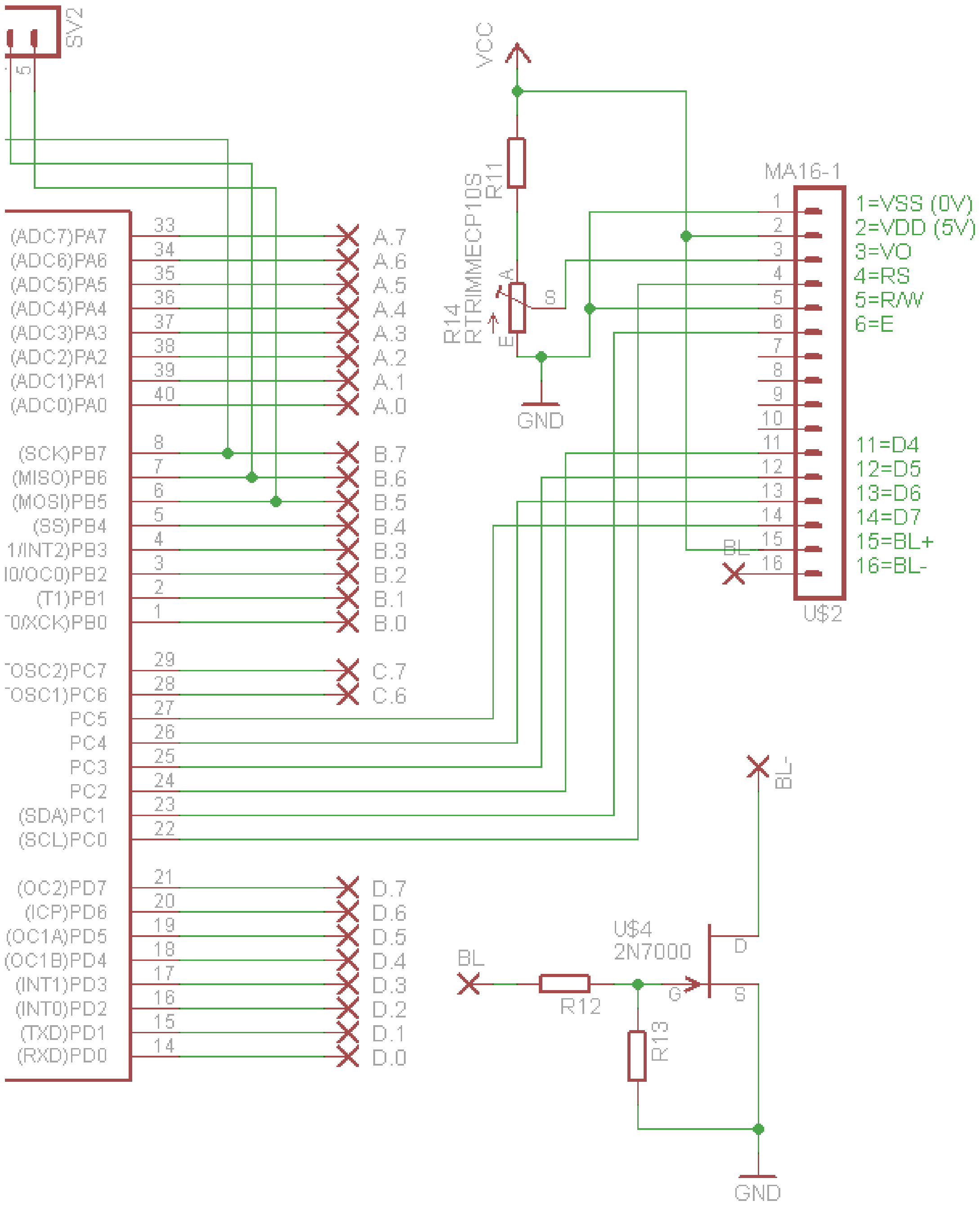
```
Config Lcdpin =Pin , Db4 =Portb.3 , Db5 =Portb.6 ,
Db6 =Portb.4, Db7 =Portb.3 , E  =Portb.1 , Rs =Portb.0
Config Lcd = 20 * 2     'configure lcd screen
```

19.6 LCD Contrast Control

In addition to the 4 data lines and the 3 control lines, there are two more pins on the LCD for power (5V-VDD and 0V-VSS) and one for adjusting the contrast or viewing angle (VO or VEE). Check the displays’ datasheet to find out what is required for VO however for almost all modern alphanumeric type LCDs the voltage is often very close to 0V so can be connected to 0V directly. You can connect via a potentiometer or trimpot so that it is adjustable as in this circuit.

HThe voltage divider here is made up of both fixed and a variable resistance.

If the trim pot was 10k and the resistor was 47 k then the voltage for the contast would be



19.7 Learning to use the LCD

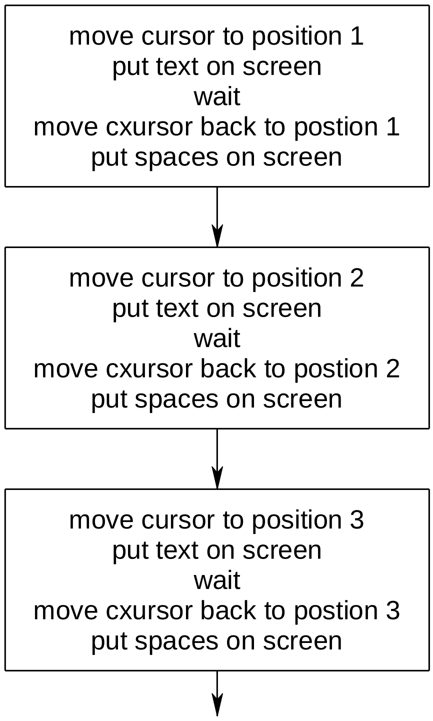
The first thing to learn about is how to put simple text on the LCD. In this program a number of different variable types are used including strings.

```
'------------------------------------------------
'Title Block
'Author: B.Collis
'Date: Aug 2009
'File Name: LCD_Ver2.bas
'-----------------------------------------------
'Program Description:
'use an LCD to display strings
'Hardware Features:
'LCD on portb - note the use of 4 bit mode and only 2 control lines
'Program Features:
'-----------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                      'the speed of operations inside the micro
$regfile = "attiny461.dat"              'the micro we are using
'-----------------------------------------------
'Hardware Setups
Config Porta = Output
Config Portb = Output
Config Lcdpin = Pin , Db4 = Portb.3 , Db5 = Portb.4 , Db6 = Portb.5 , Db7 = Portb.6 , E = Portb.2
, Rs = Portb.1
Config Lcd = 20 * 2                     'conifgure lcd screen
'Hardware Aliases
'Initialise hardware
Cls                                     'clears LCD display
Cursor Off                              'cursor not displayed
'Declare Constants
Const Waitabout = 6
Const Flashdelay = 250
'Declare Variables
Dim Message1 As String * 20
Dim Message2 As String * 20
Dim Xposition As Byte
Dim Count As Byte
'Initialise Variable
Message1 = "hello"
Message2 = "there"
Xposition = 5
'---------------------------------------------
'Program starts here
Do
   For Count = 1 To 3
      Locate 1 , Xposition
      Lcd Message1                      'display message stored in the variable
      Waitms Flashdelay
      Locate 1 , 1
      Lcd "                   "         'delete anything on this line of the lcd
      Waitms Flashdelay
      Locate 2 , Xposition
      Lcd Message2                      'display message stored in the variable
      Waitms Flashdelay
      Locate 2 , 1
      Lcd "                   "         'delete anything on this line of the lcd
      Waitms Flashdelay
   Next
   Wait Waitabout                      'seconds
Loop
```

19.8 Repetition again - the ‘For-Next’ and the LCD

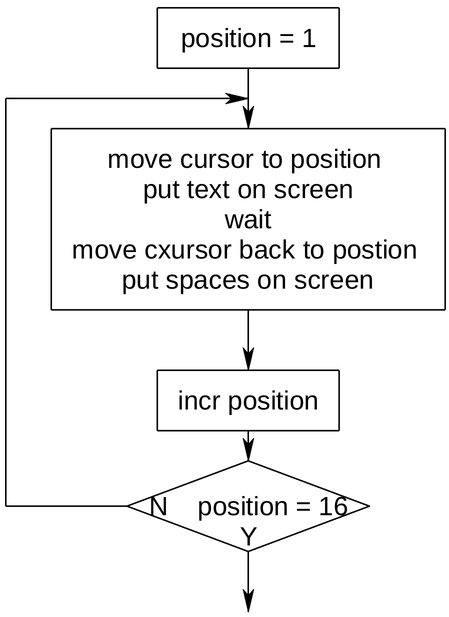
This command makes programmers life easier by allowing easy control of the number of times something happens. This is perhaps the essence of computer programming, getting the computer to do repetitive work for you. If you want some text to move across an LCD then **you could do it the long way**

```
Do
        Locate 2,1              'first position
        Lcd "Hello"
        Waitms timedelay
        Locate 2,1
        Lcd "     "

        Locate 2,2              'second position
         Lcd "Hello"
        Waitms timedelay
         Locate 2,2
        Lcd "     "

        Locate 2,3              'third posistion
         Lcd "Hello"
         Waitms timedelay
         Locate 23
        Lcd "     "
…
...
Loop
```



OR the smart way

```
Do
    For Position = 1 To 16          '
        Locate 2, position          'move cursor w
        Lcd "Hello"                 'display text
        Waitms Timedelay            'wait a bit
        Locate 2, position          'move cursor
        Lcd "     "                 'blank lcd
    Next
    For Position = 16 To 1, step -1
        Locate 2, position          'move cursor
        Lcd "world"                 'display text
        Waitms Timedelay            'wait a bit
        Locate 2, position          'move cursor
        Lcd " "                     'blank text
    Next
Loop
End          'end program
```



Identifying where and how to use loops in your programs is an essential skill to practice lots when learning to program. This is only one of several looping commands which all do similar (but not exactly the same) things.

19.9 LCD Exerises

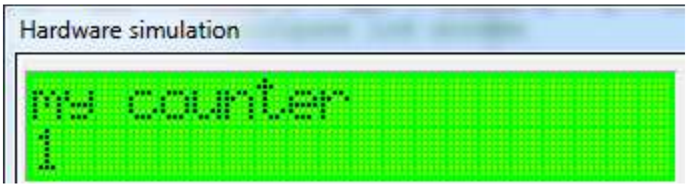
Here is a program that counts on the LCD

```
'------------------------------------------------
'Title Block
'Author: B.Collis
'Date: Aug 2009
'File Name: LCD_Count1.bas
$sim
'-----------------------------------------------
'Program Description:
'use an LCD to display strings and numbers
'Hardware Features:
'LCD on portb - note the use of 4 bit mode and only 2 control lines
'Program Features:
'-----------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$crystal = 1000000        'the speed of operations inside the micro
$regfile = "attiny461.dat"  'the micro we are using
'-----------------------------------------------
'Hardware Setups
Config Porta = Output
Config Portb = Output
Config Lcdpin = Pin , Db4 = Portb.3 , Db5 = Portb.4 , Db6 = Portb.5
, Db7 = Portb.6 , E = Portb.2 , Rs = Portb.1
Config Lcd = 20 * 2
'conifgure lcd screen
'Hardware Aliases
'Initialise hardware
Cls                              'clears LCD display
Cursor Off                   'cursor not displayed
'----------------------------------------------
'Declare Constants
Const Waitabit = 2
Const Flashdelay = 250
'----------------------------------------------
'Declare Variables
Dim Message1 As String * 20 'a variable to store some text
Dim Message2 As String * 20 'a variable to store some text
Dim Xposition As Byte  'position of the text on the LCD
Dim Count As Byte    'a variable to count
'Initialise Variable
Message1 = "my counter"
'----------------------------------------------
'Program starts here
Do
   Locate 1 , 1
   Lcd Message1
   For Count = 1 To 20
      Locate 2 , 1
      Lcd Count
      Waitms 500
   Next
Loop
End
```

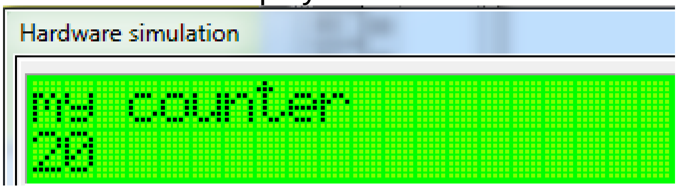
When you run this program you will see there is a problem with the displaying of the numbers
The zero stays on the LCD whenthe counter goes from 20 back to 1 again.

Now here is a really important conept you need to understand. You need to separate the two things going on here in your system
The first is the process of counting: 1,2,3,4,5,6…18,19,20,1,2,3,…
And the second is the output code LCD count.

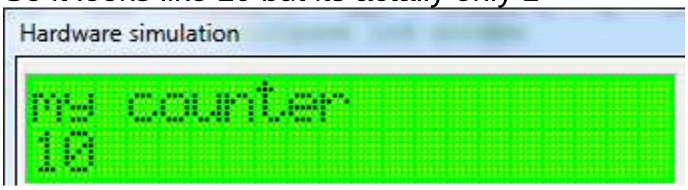
These are two very separate things.
When we say LCD count, it puts the variable count onto the LCD if countis 1 digit it writes 1 digit, if count is 2 digits it writes 2 digits.
The program doesn’t care what is on the LCD already it just overwrites it.
So the first time through the loop is does this



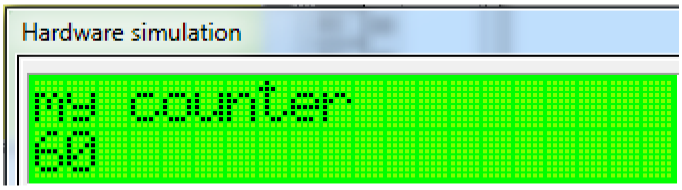
But after it has displayed 20



It goes back to 1 again and the 0 is stuck on the LCD.
So it looks like 10 but its actally only 1



Of course 6 looks like 60



So you need to know how to clear some digits on the lcd and you also need to know how to apply it logically to each problem you encounter like this.
Fix 1: In this case we are displaying 2 digits so we could do this in our program

```
Do
   Locate 1 , 1
   Lcd Message1
   For Count = 1 To 20
      Locate 2 , 1 'blank the digits we are going to use before using
      Lcd "  "
      Locate 2 , 1
      Lcd Count
      Waitms 1500
   Next
Loop
```

Try this out on your LCD, does the counting on the look nice or not.

Fix 2: add an extra space to the end o fthe count like this Lcd Count ; " "

```
Do
   Locate 1 , 1
   Lcd Message1
   For Count = 1 To 20
      Locate 2 , 1
      Lcd Count ; " "
      Waitms 1500
   Next
Loop
```

This code has a hidden problem, when the count is over 9 it takes up not 2 but three digits on the LCD, and if you are displaying anything else on the LCD then it might overwrite it.

Fix3: only fix exactly what we want to fix
In this case when there is 1 digit blank the unused digit on the LCD

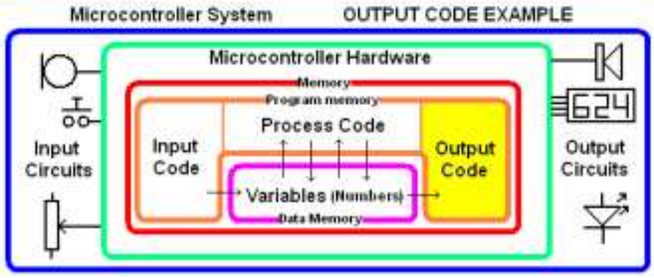
```
Do
   Locate 1 , 1
   Lcd Message1
   For Count = 1 To 20
      Locate 2 , 1
      Lcd Count
      If Count < 9 Then Lcd " "
      Waitms 1500
   Next
Loop
```

Note in my fixing of this probelm that I didn't even consider using CLS in my loop, this is because these LCDs are so slow that using a CLS in a loop causes the whole display to flicker a lot and it looks aweful – to prove this I suggest you try this solution!

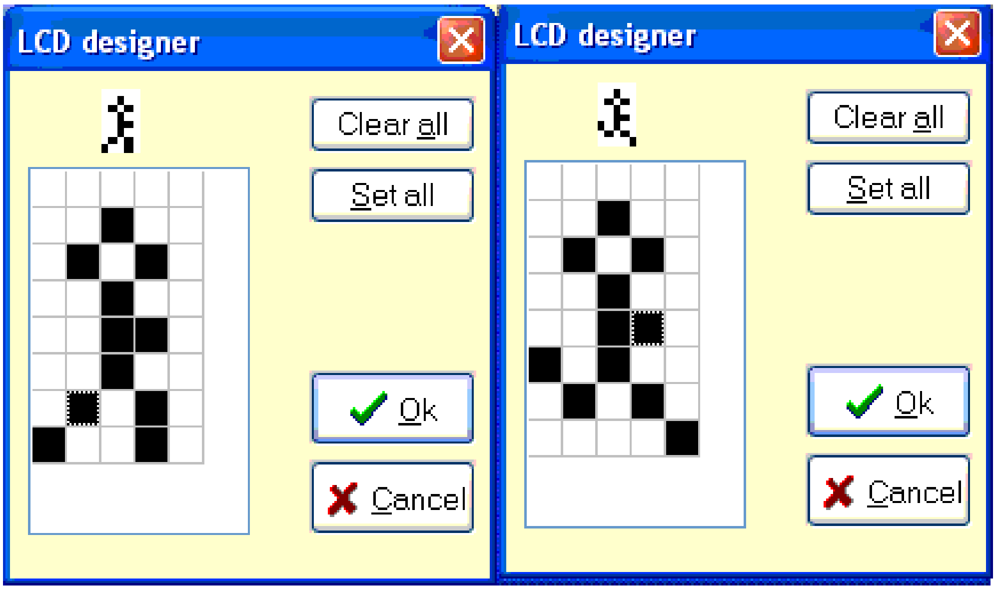
```
Do
   For Count = 1 To 20
      Cls
      Locate 1 , 1
      Lcd Message1
      Locate 2 , 1
      Lcd Count
      Waitms 1500
   Next
Loop
```

These ideas are repeated in different conexts in the next few sections to help you get used to them.

19.10 Defining your own LCD characters



The displays have 8 locations (0 to 7) where you can define your own characters
If you want to define a simple animation you can draw these using the LCD DESIGNER in Bascom and have the program write these to the screen one at a time using a loop.



19.11 LCD custom character program

```
'-----------------------------------------------------------------
'  Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                 'the speed of the micro
$regfile = "m32def.dat"             'our micro, the ATMEGA8535-16PI
'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output               'LEDs on portA
Config Portb = Output               'LEDs on portB
Config Portc = Output               'LEDs on portC
Config Portd = Output               'LEDs on portD
'config inputs

'LCD
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.5 , Db6 = Portb.6 , Db7 = Portb.7 , E = Portb.0 ,
Rs = Portb.1
Config Lcd = 20 * 4                'configure lcd screen
'  Hardware Aliases
                        'clear lcd screen
'-----------------------------------------------------------------
'  Declare Constants
Const Rundelay = 300
'-----------------------------------------------------------------
```

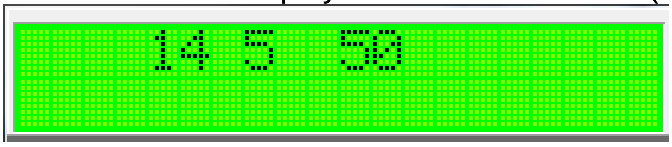
```
'  Declare Variables
Dim X_pos As Byte
Dim location As Byte
' Initialise Variables
'-----------------------------------------------------------------
'  Program starts here
Cls
Cursor Off
Deflcdchar 0 , 32 , 4 , 10 , 4 , 6 , 20 , 10 , 1
Deflcdchar 1 , 32 , 4 , 10 , 4 , 6 , 4 , 10 , 18
Do
   For X_pos = 1 To 20                'for the wodth of the screen
      Locate 1 , X_pos                'position the cursor
      'find if odd(0) or even(1) location
      '-mod returns the remainder of the division I/2 (0 or 1)
      Location = X_pos Mod 2
      If Location = 0 Then            'no remainder so second location and all even ones
         Lcd Chr(0)
      Else                            'rem =1 so first location and all odd ones
         Lcd Chr(1)
      End If
      Waitms Rundelay
      Locate 1 , X_pos        'reposition cursor
      Lcd " "
   Next

   'for a 3 stage animation
   '- define your third character here
   For X_pos = 1 To 20                'for the width of the screen
      Locate 1 , X_pos                'position the cursor
      'find if odd(0) or even(1) location
      '-mod returns the remainder of the division I/3 (0,1 or 2)
      Location = X_pos Mod 3
      If Location = 0 Then            'no remainder so third location
         Lcd Chr(0)
      Elseif Location = 1 Then        'first location
         Lcd Chr(1)
      Else                            'second location
         Lcd Chr(2)
      End If
      Waitms Rundelay                 'wait a bit
      Locate 1 , X_pos                'reposition cursor
      Lcd " "                         'blank the old character
   Next
Loop
End
```

19.12 A simple digital clock

Here is a simple clock using the LCD as a display. It is a great way to know more about if-then and making an LCD do what you want it to do.

```
$sim

'--------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                      'the crystal we are using
$regfile = "attiny26.dat"               'the micro we are using
'--------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                   'LEDs on portA
Config Portb = Output                   'LEDs on portB
Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 = Portb.4 , Db7 = Portb.5 , E
= Portb.1 , Rs = Portb.0
Config Lcd = 20 * 2                     'configure lcd screen

' Harware Aliases
' initialise hardware
Cls                                     'clears LCD display
Cursor Off                              'no cursor
'--------------------------------------------------------------------
' Declare Constants
Const Timedelay = 350
'--------------------------------------------------------------------
' Declare Variables
Dim Seconds As Byte
Dim Minutes As Byte
Dim Hours As Byte
Dim Day As Byte
Dim Month As Byte
Dim Year As Byte

'  Initialise Variables
Seconds = 50
Minutes = 5
Hours = 14                               '2pm
Day = 21
Month = 4                               'april
Year = 10                               '2010
'--------------------------------------------------------------------
' Program starts here
Do
   Locate 1 , 5
   Lcd Hours
   Locate 1 , 8
   Lcd Minutes
   Locate 1 , 11
   Lcd Seconds

   Wait 1
   Incr Seconds

Loop
End                                     'end program
'--------------------------------------------------------------------
```

Here is what the display looks like at the start (using the simulator)



There are two big problems to solve with this program:

1. The clock goes up by 1 second, however it doesn’t go from 59 back to 0
2. There is no ‘leading 0’ before any of the numbers i.e. 5 is shown not 05

Firstly lets solve the 59 going back to 0

```
Do
   Locate 1 , 5
   Lcd Hours
   Locate 1 , 8
   Lcd Minutes
   Locate 1 , 11
   Lcd Seconds

   Wait 1
   Incr Seconds

   If Seconds > 59 Then
      Seconds = 0
      Incr Minutes
   End If
Loop
End                                         'end program
```

Now you can write the rest of the code to sort out minutes and hours.

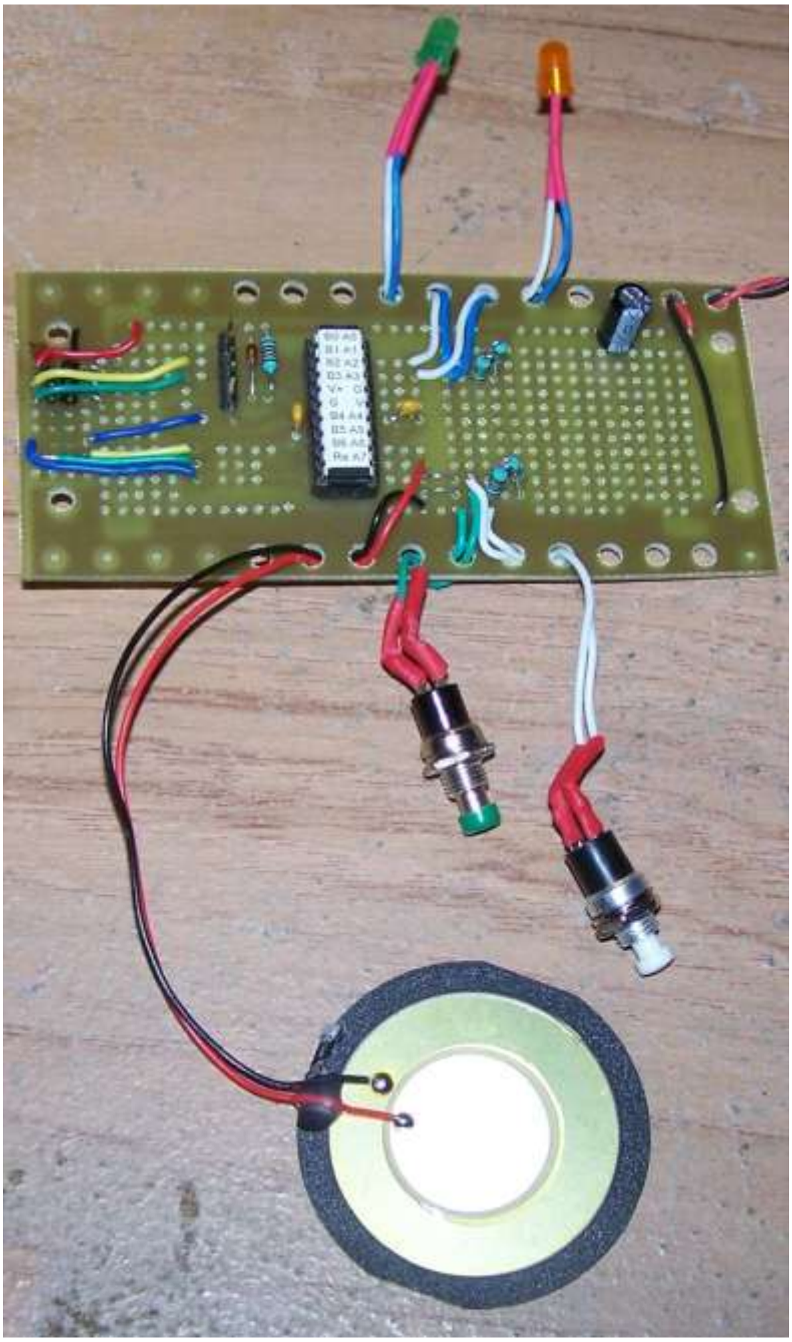
Second I will solve the leading zeros.
Think about when we want a leading zero, it is if the minutes are less than 10.

```
Do
   'display the time
   Locate 1 , 5
   Lcd Hours
   Locate 1 , 8
   If Minutes < 10 Then Lcd "0"
   Lcd Minutes
   Locate 1 , 11
   Lcd Seconds
   'increase the time
   Wait 1
   Incr Seconds
   'add code to read switches and set time
   '...
   'fix the time
   If Seconds > 59 Then
      Seconds = 0
      Incr Minutes
   End If
Loop
End                                         'end program
```

Note that when an if-then has only one command it can go on the same line and we don’t need the end-if.
There is a third issue, the clock will also need some more code so that you can set the time. Also the clock is quite inaccurate, you can check this by monitoring the time over a few minutes. Some of this can be fixed by checking how accurate the clock is over a day and changing wait 1 to waitms something. This wont really fix the issue but it will improve it. A better solution is later in the book.

19.13 Adding more interfaces to the ATTiny26 Development board

Using this board we can add other components such as LEDs, switches and a PIEZO.



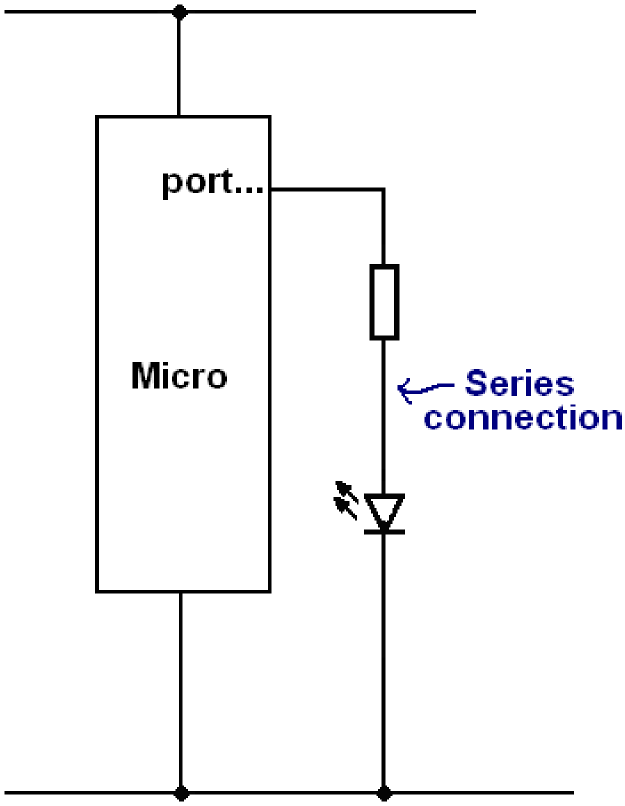
Here is a board with two switches, two LEDs and a piezo added to it. Now we will look at how to add these components one at a time.
Note that when this board was made an area around the outside of the board was left with holes for stress reliefing wires that go off the board.

The process for adding these components is:

2. Decide what you want to add and find out the correct wiring connections for it

3. Find the most convenient place for them to connect to on the board,

4. Wire them up and add your changes to the schematic.



First stage: add an LED

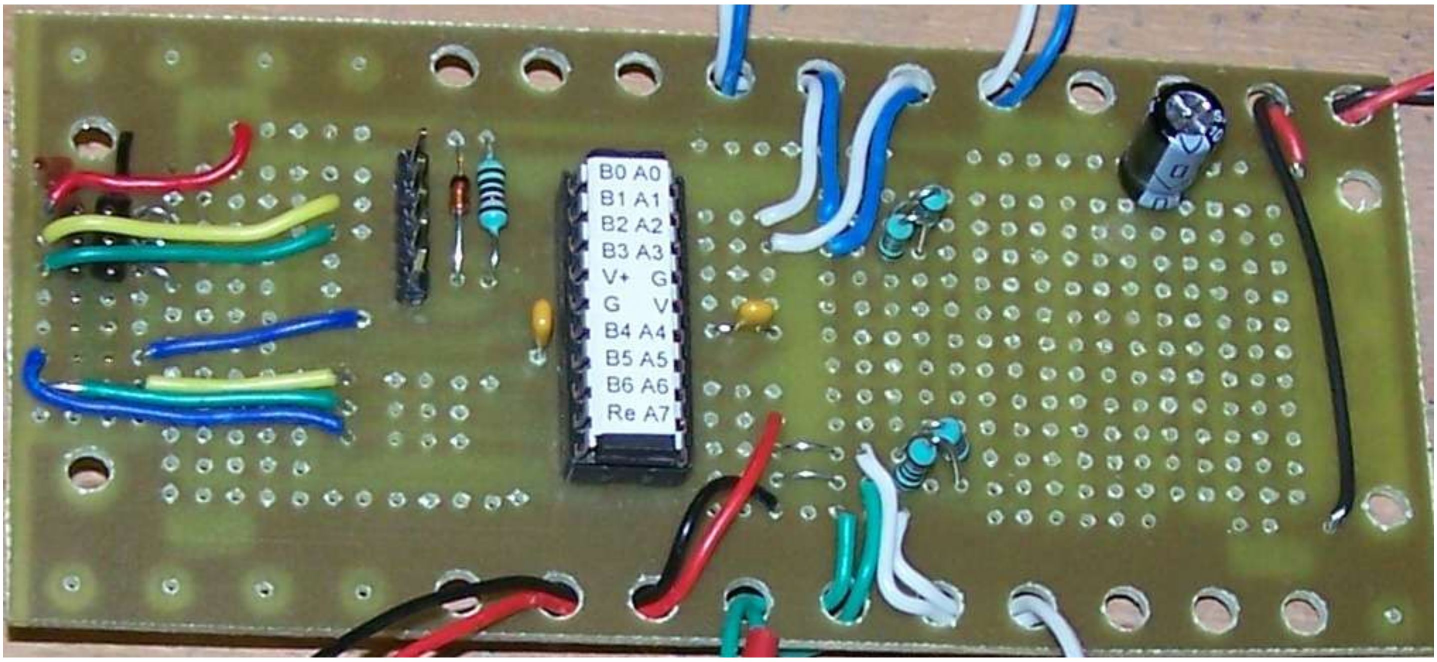
An LED requires a current limit resistor of about 1k in series with the LED (it could also be another common value such as 390, 470,560, 820 – changing the value will make the brightness change).

The schematic above shows the series connections of the LED, note that the LED and resistor canbe reversed in order but that the polarity of the LED must be the same.

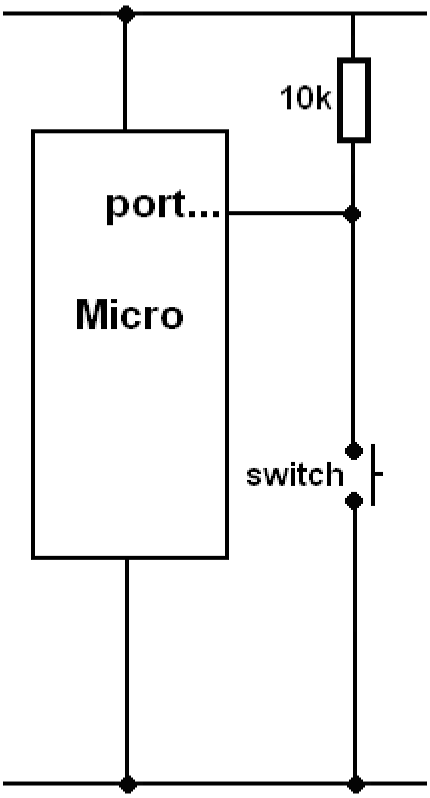
We have not chosen a specific I/O pin at this stage.

Stage two find the best I/O pin to use
An LED can be connected to any available I/O pin so in this case it was easier to choose the pin based upon where the LED was to be mounted and then select a close I/O pin.

Here PortA.0 and PortA.1 were chosen.



The negative(cathode) of the LED (blue wire) is connected to a resistor, the other side of witch connects to ground, the positive anode) of the LED (white wire) is connected to the pin of the microcontroller.



Adding 2 switches – each switch requires its own pullup resistor

This is a circuit that students initially get wrong very often, they connect the switch and resistor in series from the pin to gound, when they are in series between VCC and ground.

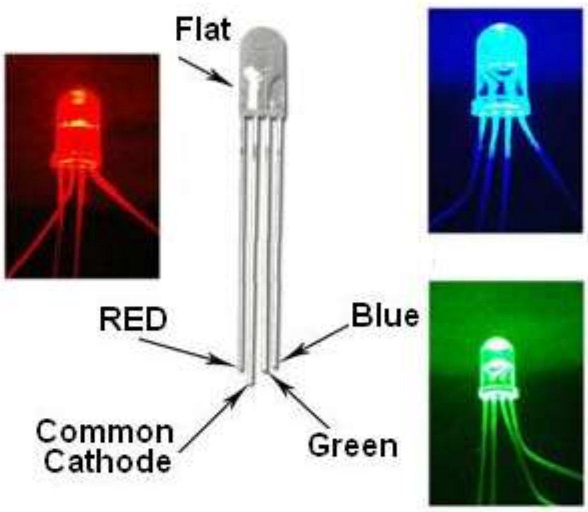
So be very careful and make sure that the resistor goes from the pin to VCC and the switch goes from the pin to ground

In the diagram only one switch and pullup resistor are shown however 2 switches and their 2 pullup resistors are shown in the picture.

What is the voltage on the micro when the switch is open?
What is the voltage on the micro when the switch is closed?

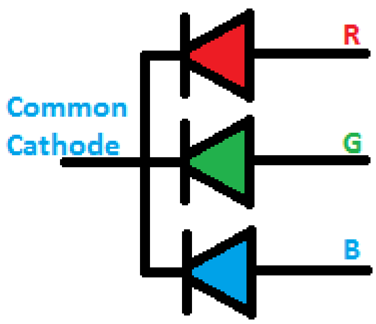
19.14 Ohms law in action – a multicoloured LED

Here is the datasheet for a multicoloured LED.Look carefully at the physica llayout, there are 4 legs

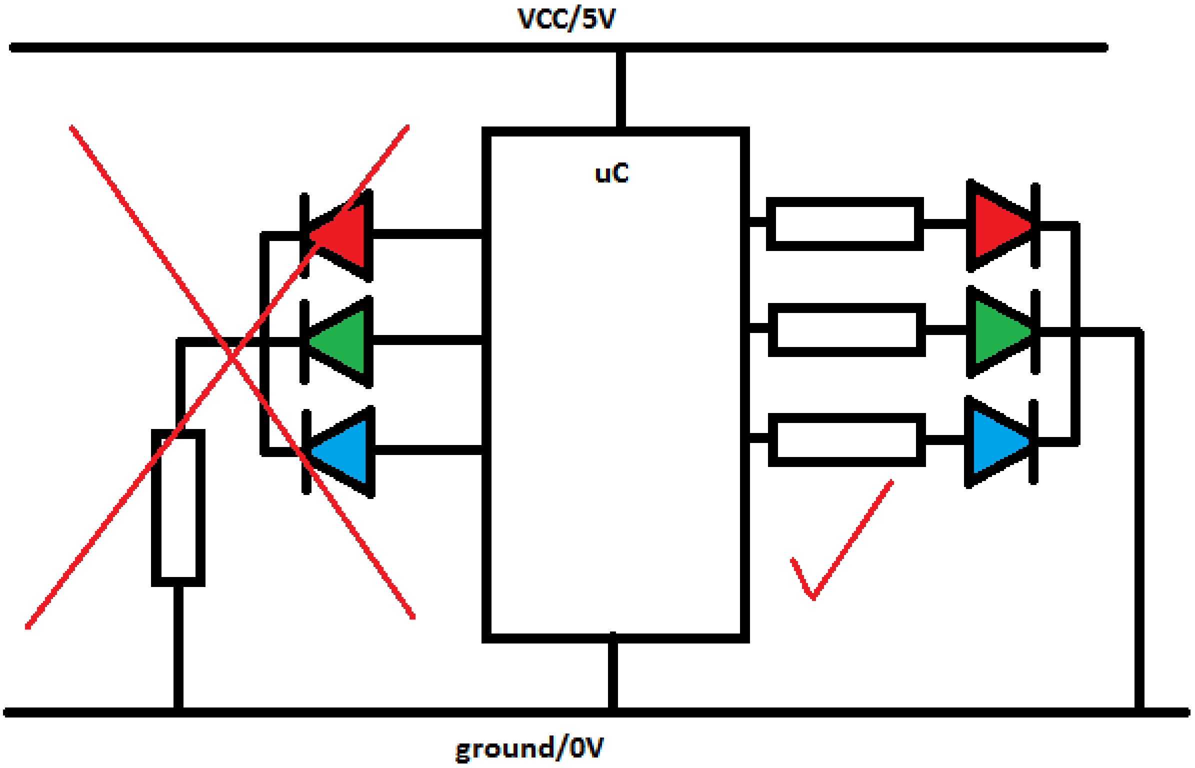


Product Number	LE-DS161
Product Name	5mm RGB LED 4000mcd
Emitted Color	Red / Green / Blue
Size (mm)	5mm
Lens Color	Water Clear
Forward Current	20mA
Life Rating	100,000 Hours
Forward Voltage (V)	RED: Typical: 2 V Max: 2.4V GREEN: Typical: 3.4 V Max: 3.8V BLUE: Typical: 3.4 V Max: 3.8V
Viewing Angle	25°~35°
Luminous Intensity (mcd)	4000(Typical)~5000(Max)
Net Weight	100g / 3.6oz

Note the wiring inside the LED how all the cathodes are connected together.

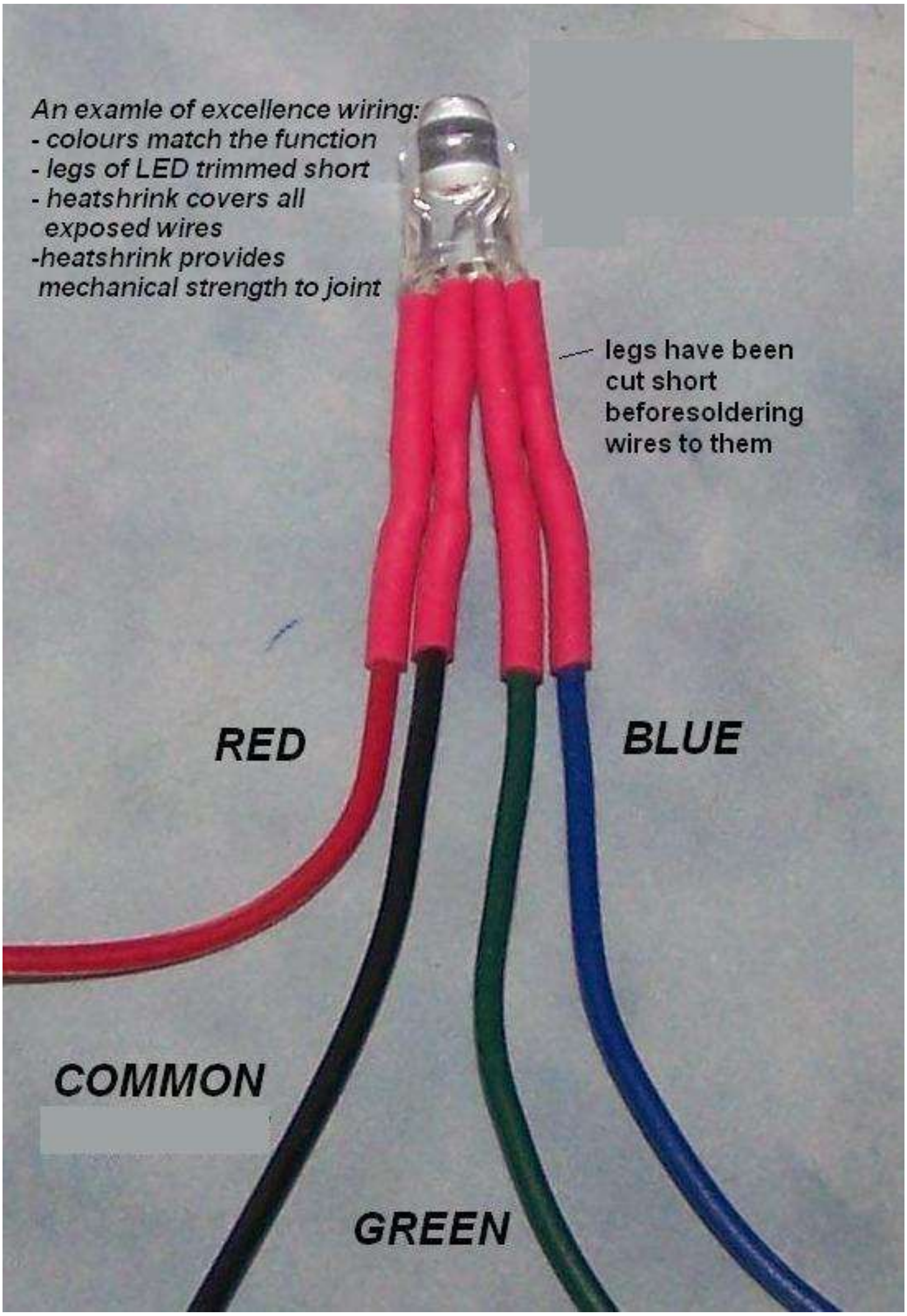


To wire this to a microscontroller we will need to use three I/O pins of the microand three resistors. We do not use a single resistor on the cathode to ground.
Why?Imagine we turned on the red LED and i ws going, then we turned on the green LED the current in the resistor would change changing the current in the red LED as well.



To work out the values of the 3 resistors we need toloo at the datasheet, there we find that the LEDs have different volage requirements, (yet another good reason for not using 1 resistor)

RED	Green	Blue
Needs 2V	Needs 3.4V	Needs 3.4V (same as green)
20mA max current = max brightness	20mA	
V = 5V - 2V V = 3V	V = 5V - 3.4V V = 1.6V	
R = V/I R = 3V/0.020A R = 150 ohm	R = V/I R =1.6V/0.020A R = 80 Ohm	same as green
If the LEDs don’t need to be so bright we could test them with a power supply and try different values of resistors. If we found that 5mA was enough we would need to calculate the values again.		
R = V/I R = 3V/0.005A R = 600 ohm	R = V/I R =1.6V/0.005A R = 320 Ohm	same as green


An examle of excellence wiring:
- colours match the function
- legs of LED trimmed short
- heatshrink covers all
exposed wires
-heatshrink provides
mechanical strength to joint
legs have been
cut short
beforesoldering
wires to them
RED
BLUE
COMMON
GREEN

20 Basic analog to digital interfaces

In the real world we measure things in continuously varying amounts.

The golf ball is some distance from the hole. It might be 11 metres from the hole, it might be 213.46236464865465437326542 metres from the hole.

The airplane might have an altitude of 11,983 metres or perhaps 1,380.38765983 metres.

A computer works in binary (or digital) which means it has the ability to sense only two states. For example the golf ball is either in the hole or not. The plane is either in the air or not.

When we want to measure the actual distance in binary we must use a number made up of many digits e.g. 101011010 (=346 decimal) metres.

20.1 ADC - Analog to Digital conversion

We need to be able to determine measurements of more than on and off, 1 and 0, or in and out. To do this we convert a continuously varying analogue input such as distance, height, weight, lighltlevel etc to a voltage.

Using the AVR this analogue value can then be converted to a binary number within the range 0 to 1111111111 (decimal 1023) within the microcontroller. We can then make decisions within our program based upon this information to control some output.

20.2 Light level sensing

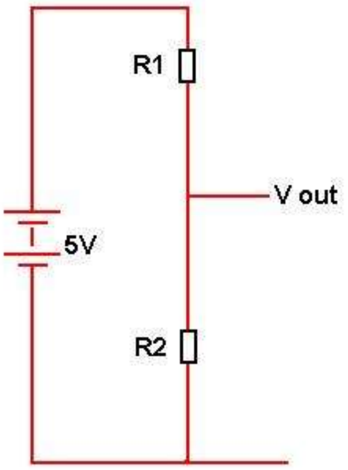
We will measure the amount of light falling on a sensor and use the LED's on the microcontroller board to display its level.

The LDR

The LDR (Light Dependant Resistor) is a semiconductor device that can be used in circuits to sense the amount of light. Get an LDR and measure the resistance when it is in the dark and measure the resistance when it is in bright sunlight. Record the two values.

20.3 Voltage dividers review

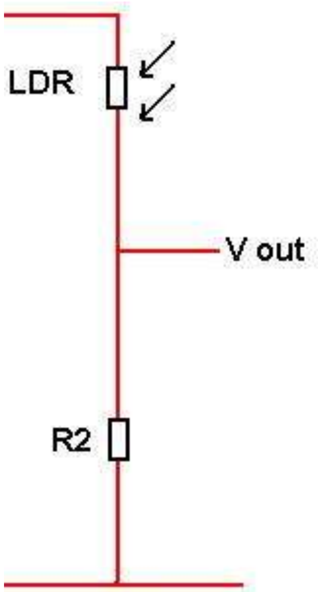
When you studied ohms law you also studied the use of voltage dividers. A voltage divider is typically two resistors across a battery or power supply.



A voltage divider is shown here. With the 5volts applied to the circuit the output voltage will be some proportion of the input voltage.

If the two resistors are the same value then the output voltage will be one_____ (quarter/half/third) of the input voltage; i.e. it has been divided by ______ (2/3/4). If we change the ratio of the two values then the output voltage will vary.

With R1 larger than R2 the output voltage will be low and with R2 larger than R1 the output voltage will be high.



Replace one of the resistors with an LDR, we know that the resistance of an LDR changes with the amount of light falling on it.

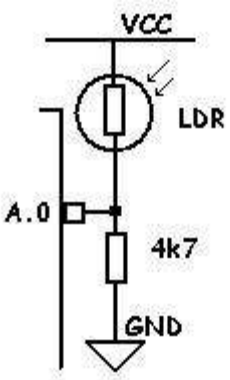
If the light level is low, and then the resistance is _____ (high/low), therefore the output voltage is _____ (low/high).

If the light level is high then the resistance is _____(high/low), therefore the output voltage is _____ (low/high).

Now this is what we call an analogue voltage. Analogue means that the voltage varies continuously between 0 and 5 volts.

But computers only know about digital voltages 0 volts or 5 Volts. We need to convert the analog voltage to a digital number that the computer can work with. We do that with the built in ADC (Analogue to Digital Converter) inside the Microcontroller.

20.4 AVR ADC connections



On a micro such as the ATMega8535/16, Port A has dual functions inside the microcontroller. Its second function is that of input to the internal ADC. In fact there are 8 separate inputs to the ADC one for each pin of portA.

In the diagram a 4k7 resistor is shown, this can be changed for a higher or lower value to achive the effect you want with the LDR (also the LDR and resistor can be swapped in the circuit to alter the effect as well)

20.5 Select-Case

In this example you will learn about how to use select case which is a very tidy way of writing a whole lot of if-then statements.

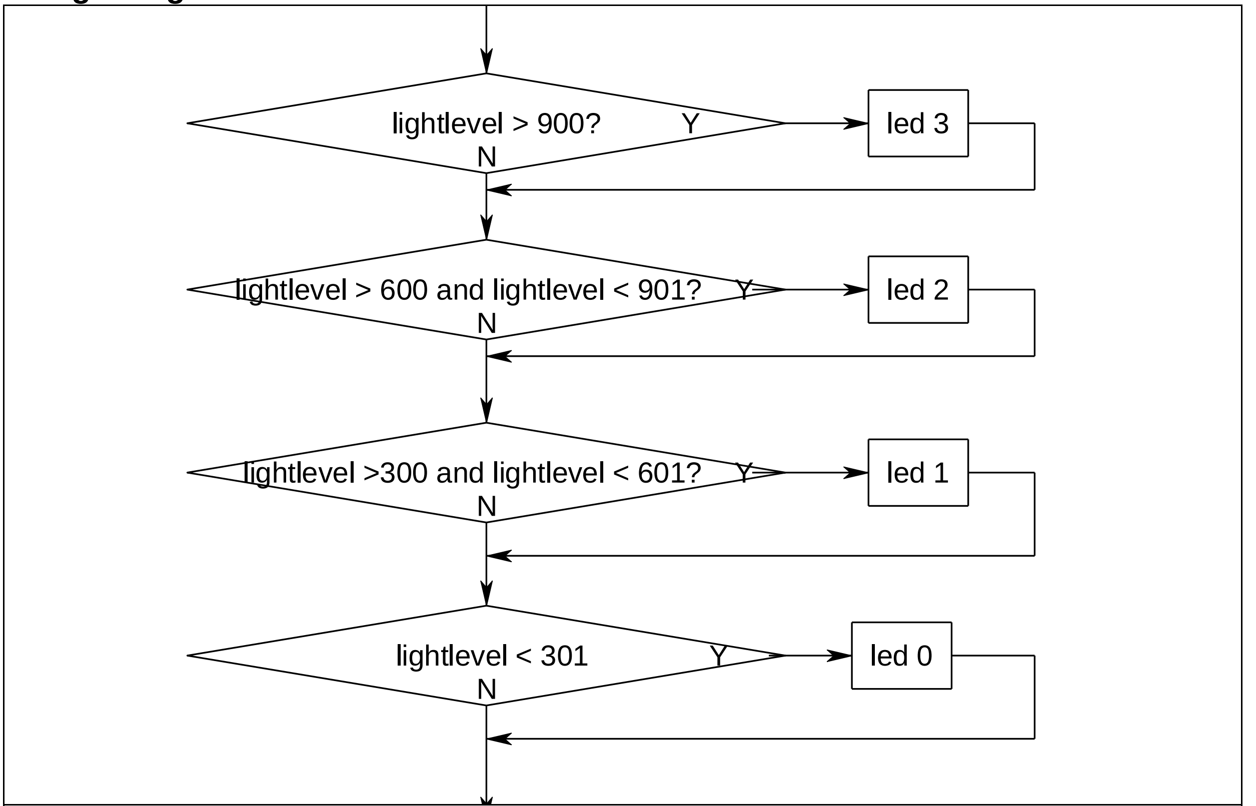
Specification from the brief:
Turn on one of 4 leds which represents one of 4 levels of light.

Algortithm
When the lightlevel is brightest turn on the 4th led
When the lightlevel is medium high turn on 3rd led
When the lightlevel is low turn on 2nd LED
When the lightlevel is very low/dark turn on 1st LED

Planning Tool Selection
(A table is selected to help us clarify the algorithm and plan the program)

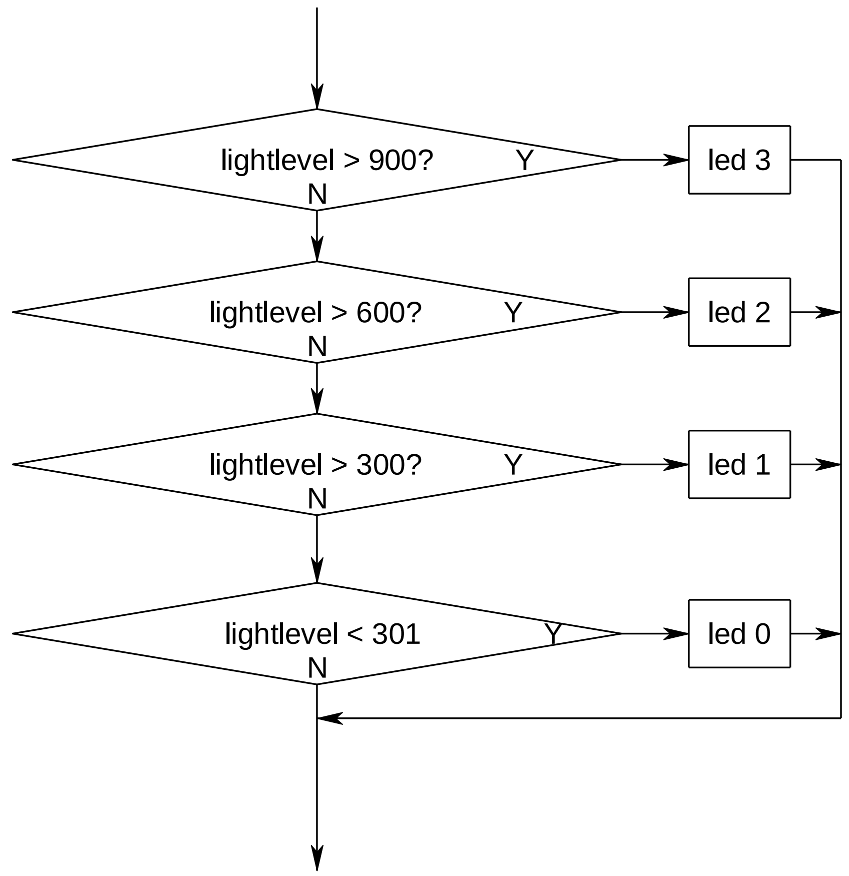
Lightlevel range	**testing values using simple math**	**output**
From 901 to 1023	Lightlevel > 900 (ignore over 1023)	LED 3
From 601 to 900	Lightlevel > 600 AND Lightlevel < 901	LED 2
From 401 to 600	Lightlevel > 400 AND Lightlevel < 601	LED 1
From 0 to 400	Lightlevel < 401	LED 0

Planning using a flowchart




```
If Lightlevel > 900 Then Portc = &B11110111
If Lightlevel > 600 And Lightlevel < 901 Then Portc = &B11111011
If Lightlevel > 400 And Lightlevel < 601 Then Portc = &B11111101
If Lightlevel < 401 Then Portc = &B11111110
```

There is a much better way to **plan** this code, so that it is more **efficient** (the micro has less to do and the program runs faster). It does this by once having found a solution it stops checking for any other solutions. This can save a lot of processing in large programs. You do however have to watch the order in which you check values and how you use the < and > tests.



This is handled for us by the select case statement

```
Select Case Lightlevel
        Case Is > 900 : Portc = &B11110111
        Case Is > 600 : Portc = &B11111011
        Case Is > 400 : Portc = &B11111101
        Case Is < 401 : Portc = &B11111110
End  Select
```

Once the select case has found a solution it does no more checking and exits the at the END SELECT

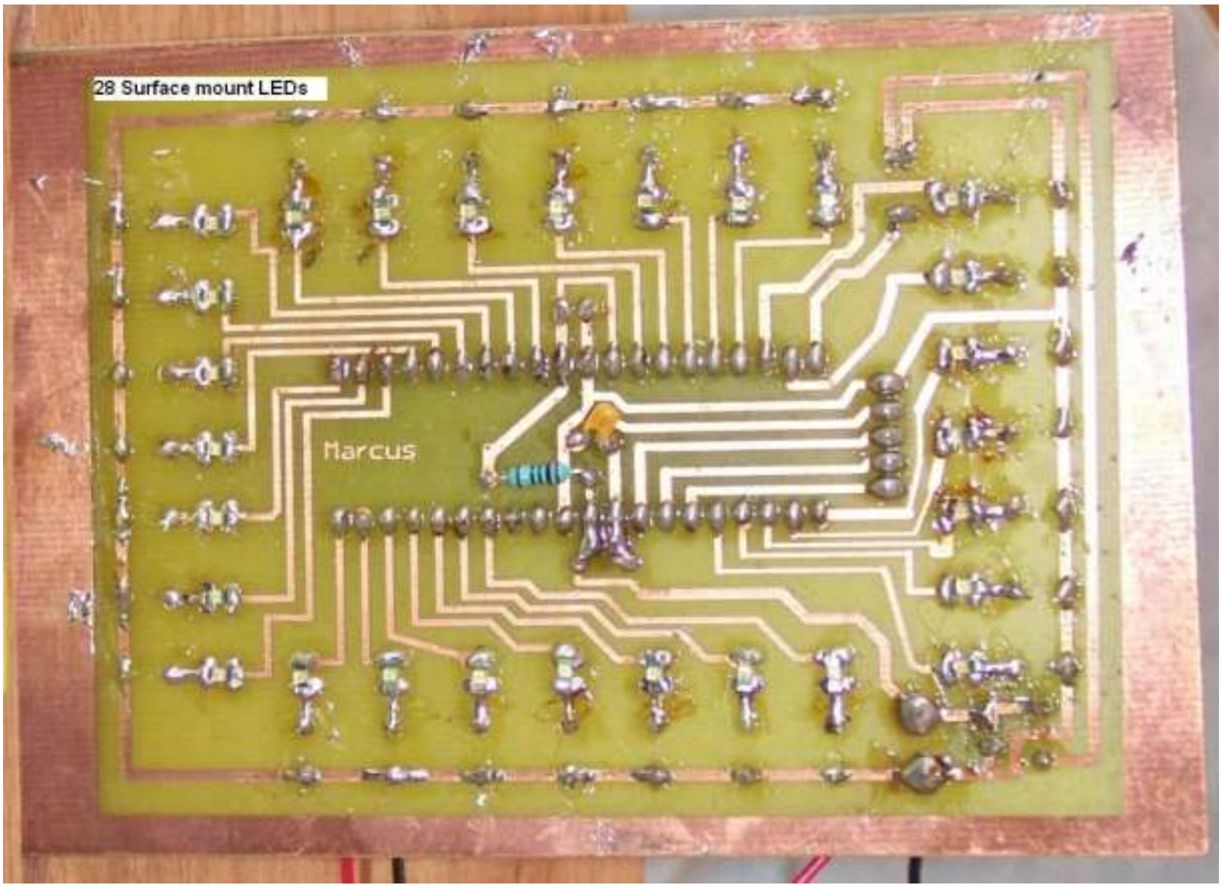
20.6 Reading an LDR's values

Now we will write some code to make use of the LDR.
Note that the variable used in this program is of size WORD i.e. 2bytes (16 bits)
This is because the values given from the analogue to digital converter are bigger than 255.
Note also a new programming structure **select-case-end select** has been used.Select-case is equivalent to a whole lot of IF-THEN statements

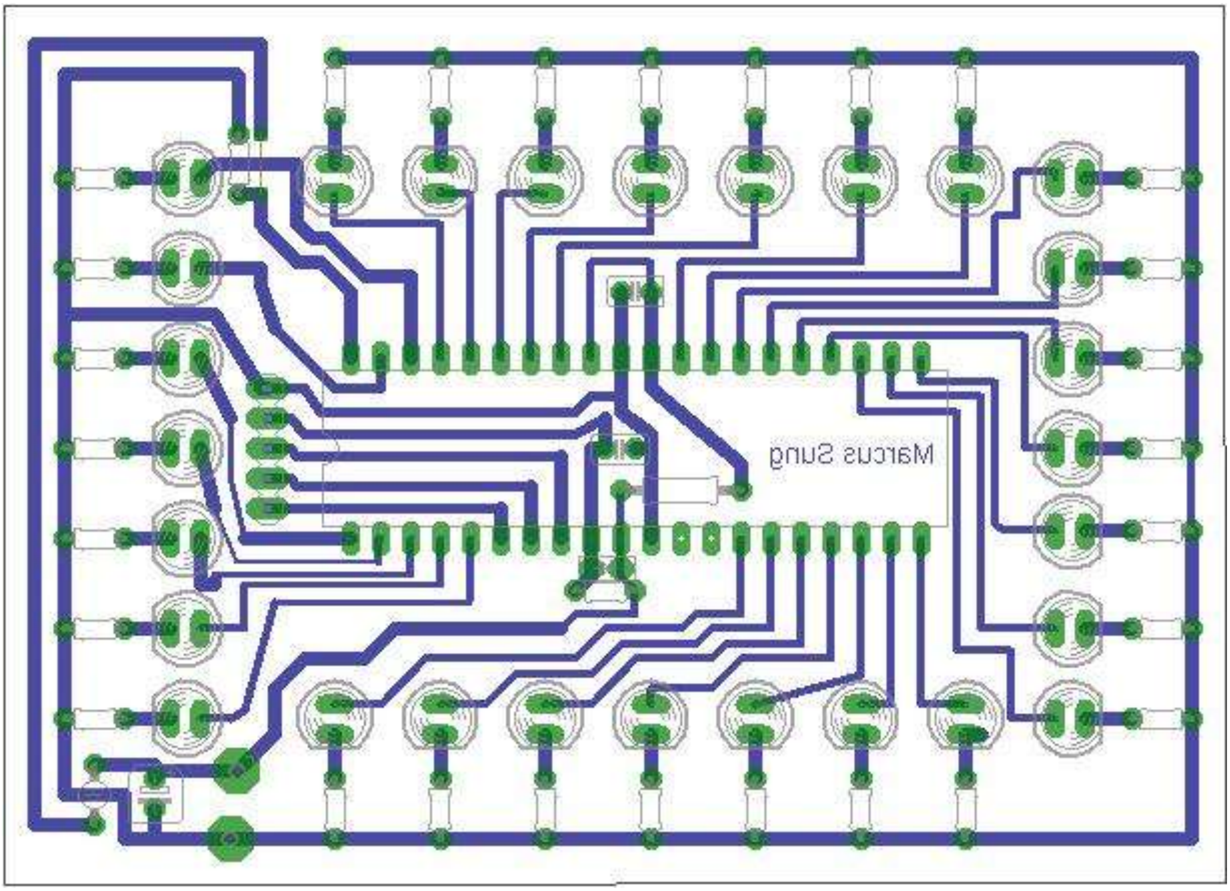
```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 7 Aug 2003
' Version: 1.0
' File Name: LDR_Ver1.bas
'-----------------------------------------------------------------
' 2. Program Description:
' This program displays light level on the LEDs of portc
' 3. Hardware Features:
' LEDs as outputs
' An LDR is connected in a voltage divider circuit to portA.0
' in the dark the voltage is close to 0 volts, the ADC will read a low number
' in bright sunlight the voltage is close to 5V, the ADC will be a high value

' 4. Software Features:
' ADC converts input voltage level to a number in range from 0 to 1023
' Select Case to choose one of 8 values to turn on the corresponding LED
' 1023, 895, 767, 639, 511, 383, 255, 127,

' -----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000            'the speed of operations inside the micro
$regfile = "m8535.dat"        ' the micro we are using

'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Pina.0 = input ' LDR
Config Portd = Output 'LEDs on portD
Config Adc = Single , Prescaler = Auto, Reference = Avcc
Start Adc
' 7. Hardware Aliases

' 8. initialise ports so hardware starts correctly
'    must not put a high on the 2 adc lines as this will turn on the micros
'    internal pull up resistor and the results will be erratic
Portc = &B11111100 'turns off LEDs



'-----------------------------------------------------------------
```

```
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Lightlevel As Word
' 11. Initialise Variables
'-----------------------------------------------------------------
' 12. Program starts here
‘ note the use of select case instead of many if statements(see next section)
Do
   Lightlevel = Getadc(0) ' number from 0 to 1023 represents the light level
   Select Case Lightlevel
      Case Is > 895 : Portc = &B01111111 'turn on top LED in bright light
      Case Is > 767 : Portc = &B10111111
      Case Is > 639 : Portc = &B11011111
      Case Is > 511 : Portc = &B11101111
      Case Is > 383 : Portc = &B11110111
      Case Is > 255 : Portc = &B11111011
      Case Is > 127 : Portc = &B11111101
      Case Is < 128 : Portc = &B11111110 'turn on bottom LED in dark
   End Select
Loop ' go back to "do"

End 'end program
'-----------------------------------------------------------------
' 13. Subroutines
'-----------------------------------------------------------------
' 14. Interrupts
```

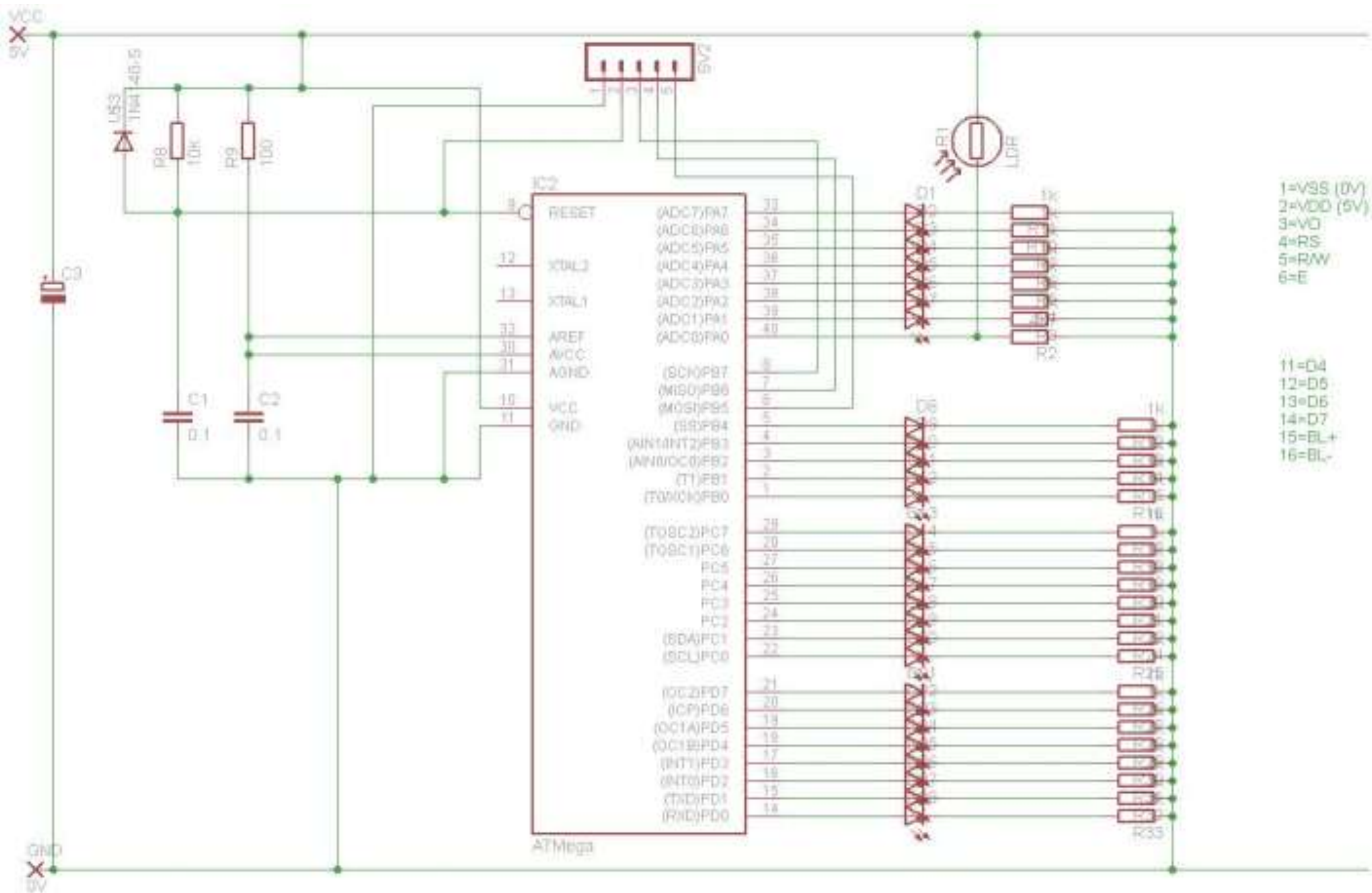
20.7 Marcus' year10 night light project



In this project Marcus used 28 high intensity surface mount LEDs soldered to the copper side of the PCB.



The schematic is quite straight forward with an LDR on PinA.0.



Initial programs were desgined to
Test the LEDs and the LDR, then the next program combined them together.

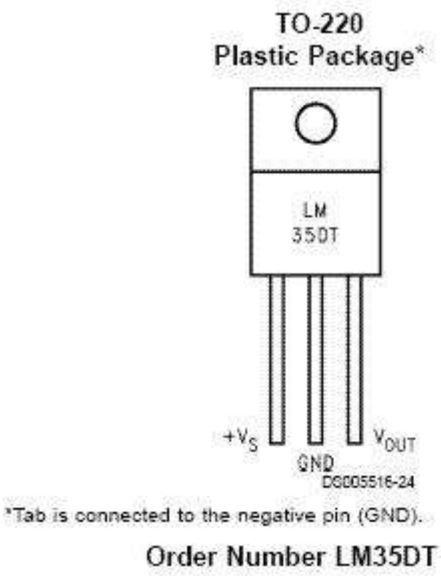
```
'-------------------------------------------------------------
'Program Description
'This program shows LED working when LDR detects different light
level
'-------------------------------------------------------------
'Hardware features:
'LEDs as outputs
'An LDR is connected in a voltage divider circuit to portA.0
'In the dark the voltage is close to 0 volts, the ADC will read a
high number
'In bright sunlight the voltage is close to 5v, the AVC will be a
high value
'------------------------------------------------------------
'Software features:
'ADC converts input voltage level to a number in range from 0 to 1023
'-----------------------------------------------------------
'Computer directives
$crystal = 8000000
$regfile = "m8535.dat"
'------------------------------------------------------------
'Hardware setups
Config Porta = Output
Config Pina.0 = Input
Config Portb = Output
Config Portc = Output                    'make these micro pins
outputs
```

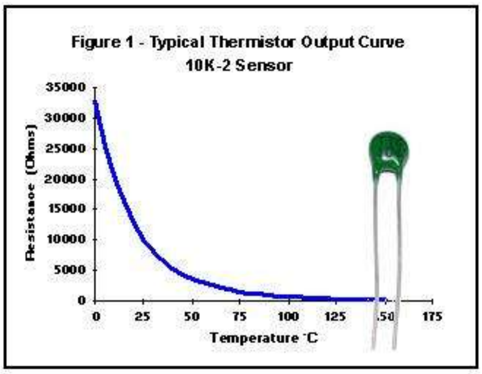
```
Config Portd = Output
Config Adc = Single , Prescaler = Auto
Start Adc
'-----------------------------------------------------------------

'Declare variables
Dim Lightlevel As Word
Dim I As Byte
'-----------------------------------------------------------------
'Program starts here

Do
  Lightlevel = Getadc(0)
  Select Case Lightlevel
         Case Is > 700 : Porta = &B00001000
                          Portb = &B00010000
                          Portc = &B00100000
                          Portd = &B01000000
                          Wait 10
         Case Is > 600 : Porta = &B00011000
                          Portb = &B00011000
                          Portc = &B00110000
                          Portd = &B01100000
                          Wait 10
         Case Is > 500 : Porta = &B00111000
                          Portb = &B00011100
                          Portc = &B00111000
                          Portd = &B01110000
                          Wait 10
         Case Is > 400 : Porta = &B01111000
                          Portb = &B00011110
                          Portc = &B00111100
                          Portd = &B01111000
                          Wait 10
         Case Is > 300 : Porta = &B11111000
                          Portb = &B00011111
                          Portc = &B10111100
                          Portd = &B11111100
                          Wait 10
         Case Is > 200 : Porta = &B11111010
                          Portb = &B00011111
                          Portc = &B10111110
                          Portd = &B01111111
                          Wait 10
         Case Is < 201 : Porta = &B01111111
                          Portb = &B00011111
                          Portc = &B11111111
                          Portd = &B11111111
                          Wait 10
  End Select
Loop
End
```

The next stage in a project like this might be to implement a timer so that the night light turns off automatically after a set period of time.

20.8 Temperature measurement using the LM35

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to degrees Celsius temperature.





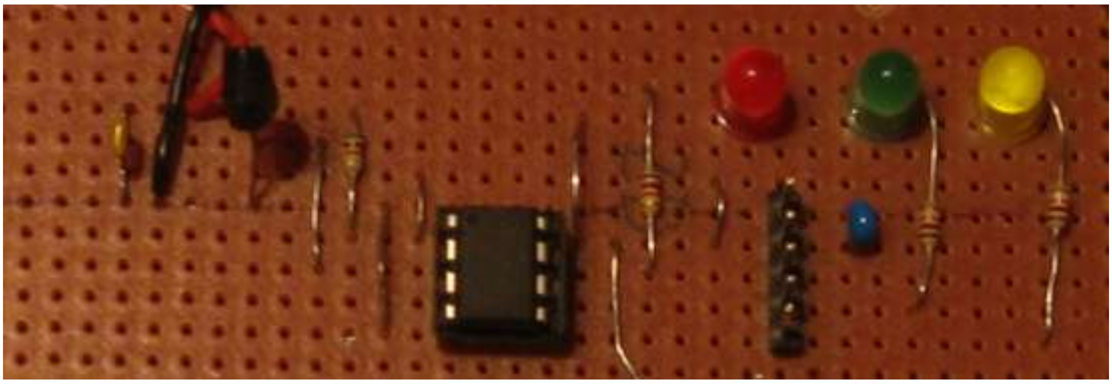
The usual temperature sensor that comes to mind is the Thermistor however thermistors are not linear but logarithmic devices as shown in this graph. If you do want to use a thermistor then try putting a resistor in parallel with it to make it more linear, however it will not be linear over its whole range.

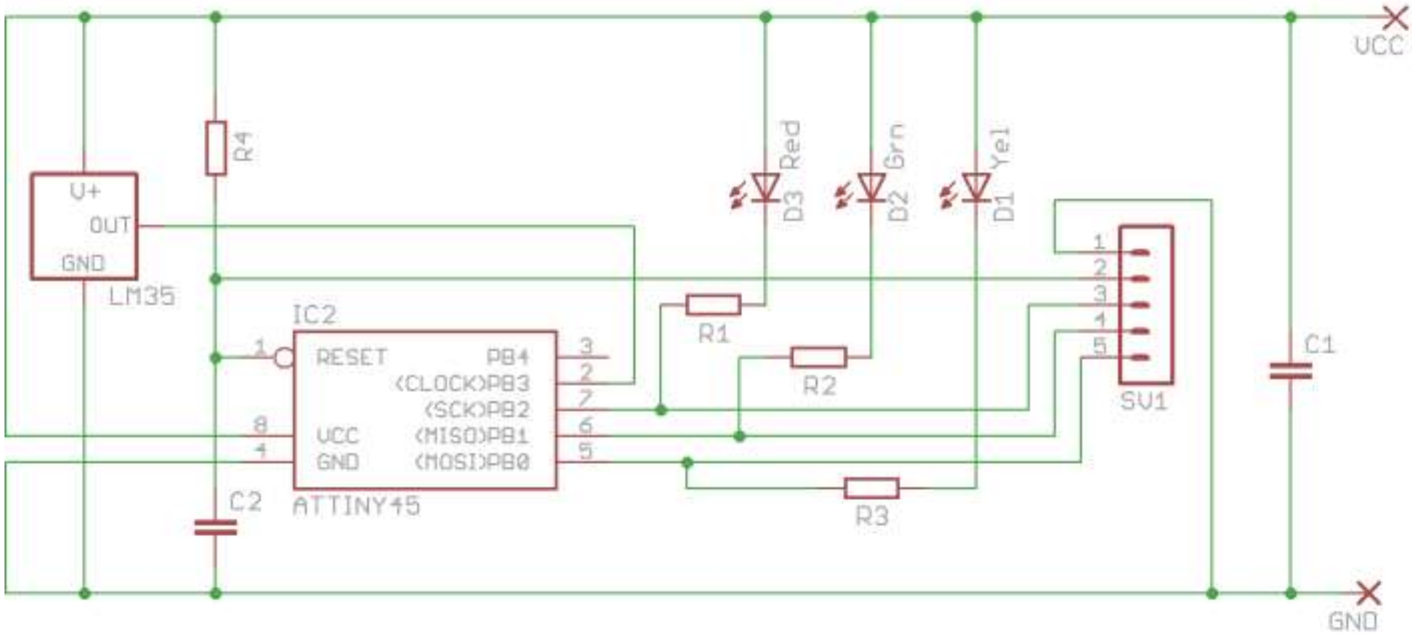
The LM35 varies linearly over its range with typically less than a ¼ degree of error. The voltage output changes at 10mV per degree. Connect the LM35 to 5V, ground and one analog input pin. The code is very straight forward

```
Config  ADC= Single, prescaler = auto
Dim Lm35 as word
Read_LM35:
        Lm35 = getadc(2)
        Locate 2,1
        Lm35 = lm35 / 2 (rough conversion to degrees)
        Lcd "temperature= " ; LM35 '
return
```

The value increases by 2 for every increase of 1 degree. When connected to 5V a temperature of 25 degrees will give an output of 250mV and an ADC reading of approximately 50 (the ADC range is 0 to 1024 for 0 to 5v).

20.9 A simple temperature display





Algorithm:
In this project there is no display apart from the LEDs so the temperature is displayed by flashing the LEDs, Red is 10s of degrees, Green is units of degrees. So a temperature of 23 degrees celcius will be displayed as the red LED flashing twice followed by the green LED flashing 3 times, followed by a 2 second wait.

Here is the code for this

```
'--------------------------------------------------------------------
' Author:      B.Collis
' Date:        19 June 2010
' File Name:   Tiny45_temprV3.bas
'--------------------------------------------------------------------
' Program Description:
' reads LM35 connected to ADC and displays temp by flashing leds
' Hardware Features:
'
'                        _____
'          RESET -|     |- VCC
' LM35 - ADC3/PB3 -|     |- PB2/ADC1 - RED LED
'       ADC2/PB4 -|     |- PB1      - GRN LED
'            GND -|_____|- PB0      - YEL LED
'
'--------------------------------------------------------------------
```

It is good practice to include a title block and full description of your hardware and program at the beginning of your code.

With a small micro a simple text diagram was created to show the connections.

```
'Compiler Directives (these tell Bascom things about the hardware)
$regfile = "attiny45.dat"                    'the micro we are using
```

<pre>$crystal = 1200000   'the speed of the micro
'attiny45 is 9.6 MHz / 8 = 1.2MHz
'---</pre>	

<pre>' Hardware Setups
' setup direction of all ports as outputs (by default they power up
as inputs)
Config Portb = Output
Config Pinb.3 = Input   'LM35 on B.3
Config Adc = Single , Prescaler = Auto , Reference = Internal_1.1
Start Adc
' the attiny45 has 2 internal references 1.1 and 2.56
' We want to measure voltages in the range of 0 to 0.5 or so,
' the 1.1V reference is better because it will give us a more
precise reading
' A voltage of 0.25V will be converted by 0.25/1.1 * 1023
' and become the number 232,
' so the ratio of ADC voltage to temperature is 1023/1.1*100 = 9.3
' if you can live with the error divide it by 9
' 0.25V (25deg) becomes 232/9 = 25.77 on the display
' if you want more accuracy then use single sized variables for the
division
 	
'Hardware Aliases
Redled Alias Portb.2   '10s of degrees
Grnled Alias Portb.1   'units of degrees
Yelled Alias Portb.0   'not used inthis verison of
the code
'---</pre>
	Here the hardware attached to the micro is setup, there is a decription of why the 1.1v reference was chosen

<pre>'Declare Constants
Const Flashdelay = 250
Const Tempr_conv_factor = 9   'rough conversion factor
Const Tempr_conv_offset = 1   'rough offset for rough conversion
'---</pre>
	Although a conversion of 9 looked like it would work, the temperature was out by a degree at room temperature. This was found by measuring the LM35 voltage out put as 0.228 and seeing the LED flash the number 24. This probably enough accuracy for room temperature measurements as the LM35 has at best an accuracy of ¼ of adegree anyway.

<pre>' Declare Variables
Dim Tempr As Word   '
Dim Tempr_10s As Byte   'temperature tens
Dim Tempr_1s As Byte   'temperature units</pre>
	Variables used in the program These are not

`'-------------------------------------------------------------------`	given initial values because they are measured

`' Program starts here`
`Set Redled                              'led off`
`Set Grnled                              'led off`
`Set Yelled                              'led off`
	Turn off the LEDs at the beginning

<table>
<tr><td>

```
Do
   Tempr = Getadc(3)                        'read the ADC value
   Tempr = Tempr / 9                        'rough conversion due
to 1.1V internal reference being used
   Tempr = Tempr - 1       'rough compensation for rough
conversion
```

</td><td>The first part of the program reads the temperatire and converts it to a useable value</td></tr>
<tr><td>

```
   'split the tempr into 2 digits
   Tempr_10s = Tempr / 10
   Tempr_1s = Tempr Mod 10
```

</td><td>This is a vital piece of learning here about division and the use of modulus. We are dealing with whole numbers when we use words , bytes and intergers in Bascom. So if we divide 27 by 10 we get 2 (note that there is no rounding) so a command exists <MOD> that allows us to get the remainder of the division 27 MOD 10 will return 7.</td></tr>
<tr><td>

```
      'flash the tempr on the LEDs
   While Tempr_10s > 0          'flash the red led the number
of 10s
      Reset Redled
      Waitms Flashdelay
      Set Redled
      Waitms Flashdelay
      Decr Tempr_10s
   Wend
   Waitms 200
   While Tempr_1s > 0          'flash the grn led the number of
units
      Reset Grnled
      Waitms Flashdelay
      Set Grnled
      Waitms Flashdelay
      Decr Tempr_1s
   Wend

   Wait 2
Loop
End                                         'end program
```

</td><td>Flashing the leds requires us to set a loop in motion, we control the number of times the loop repeats by starting it with the number and progressively subtracting 1 each time.

This is actually an efficient piece of code as microcontrollers programs are generally more efficient if they loop down to zero rather than some number other than zero, this is because of the way the hardware in a micro works</td></tr>
</table>

20.10 LM35 temperature display

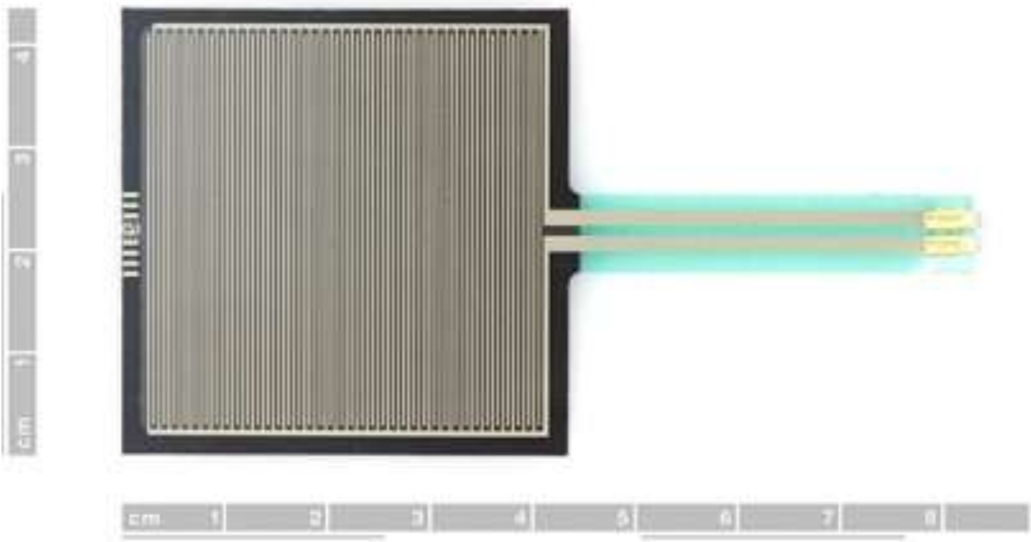
```
'-------------------------------------------------------------------
' Title Block
' Author: B.Collis
' Date: Nov 2011
' File Name: LM35_Ver2.bas
' -------------------------------------------------------------------
' Program Description:
' This program displays temperature on the LCD
' An LM35 temperature sensor is connected to portA.0
' LCD to PortB
' -------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000           'the speed the micro porcesses
instructions
$regfile = "m16def.dat"        'the particular micro we are using
' -------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports, initially as outputs
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output

Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.5 , Db6 = Portb.6 ,
Db7 = Portb.7 , E = Portb.1 , Rs = Portb.0
Config Lcd = 20 * 4                        'configure lcd screen

Config Pina.0 = Input                      'LM35 temperature sensor

'setup the ADC to do a conversion whenever we use the command
getadc()
Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc
' Hardware Aliases
Backlight Alias Portd.4
' -------------------------------------------------------------------
' Declare Constants
Const Reading_delay = 2000
' -------------------------------------------------------------------
' Declare Variables
Dim Adc_value As Word          '10bit adc value needs word variable
Dim Rough_temperature As Byte
Dim Accurate_temperature As Single
Dim Temperature As String * 5
' Initialise Variables
```

```
' -------------------------------------------------------------------
' Program starts here
Cursor Off
Cls
Set Backlight
Do
    Gosub Read_lm35_voltage                     Gosub Disp_adc_reading
    Gosub Calc_rough_tempr
    Gosub Disp_rough_tempr
    Gosub Calc_accurate_temp
    Gosub Disp_accurate_temp
    Waitms Reading_delay
   'these subroutines do not need comments as they have useful names
Loop

End                                         'end program



' -------------------------------------------------------------------
'Subroutines  -these are actions so often start with words like read,
calc, displ, squeeze, move...
' a subroutine is best if it only contains one action (even if it
consists of only a few lines of code
' this makes them easier to follow, modify and reuse.



Read_lm35_voltage:
   Adc_value = Getadc(0)   'number from 0to1023 represents the voltage
in
Return



Disp_adc_reading:
   Locate 1 , 1
   Lcd "adc reading= " ; Adc_value ; "   "
Return



Calc_rough_tempr:
   'this is a rough conversion as words can only be whole numbers
   Rough_temperature = Adc_value / 2
Return



Disp_rough_tempr:
   Locate 2 , 1
   Lcd "rough tempr= " ; Rough_temperature ; "   "
Return
```

```
Calc_accurate_temp:
   'using singles we can have decimal places in our calculations
   Accurate_temperature = Adc_value     'convert word to single
   'adc_value of 51 = 0.259V
   'conversion factor is 51/25.9= 1.96911197
   Accurate_temperature = Accurate_temperature / 1.96911197
   'turn the single into a string and round it to 1 decimalplace
   Temperature = Fusing(accurate_temperature , "#.#")
   'note we can do maths with numbers stored in singles
   ' we cannot do maths with numbers strored in string form
   ' as they are no longer numbers just codes representing digits
Return

Disp_accurate_temp:
   'this subroutine displays the two accurate readings one is a
number
   'the other is a number in string form
   Locate 3 , 1
   Lcd "tempr= " ; Accurate_temperature ; " "
   'display 1 decimal place plus deg symbol and capital C
   Locate 4 , 1
   Lcd "tempr (1dp)= " ; Temperature ; Chr(223) ; Chr(67)
Return
```

20.11 Force Sensitive Resistors

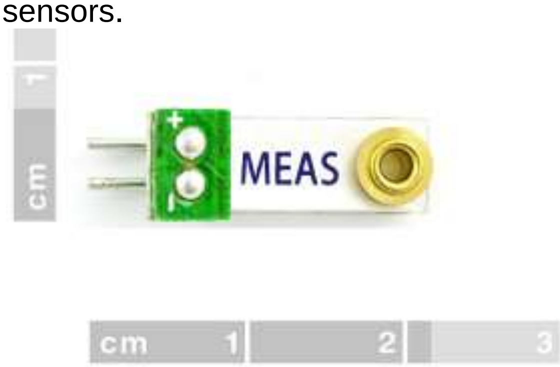
The FSR is a neat device for sensing pressure, its not accurate enough to measure weight but useful to detect the presence of someone standing on a mat or tapping on a surface.



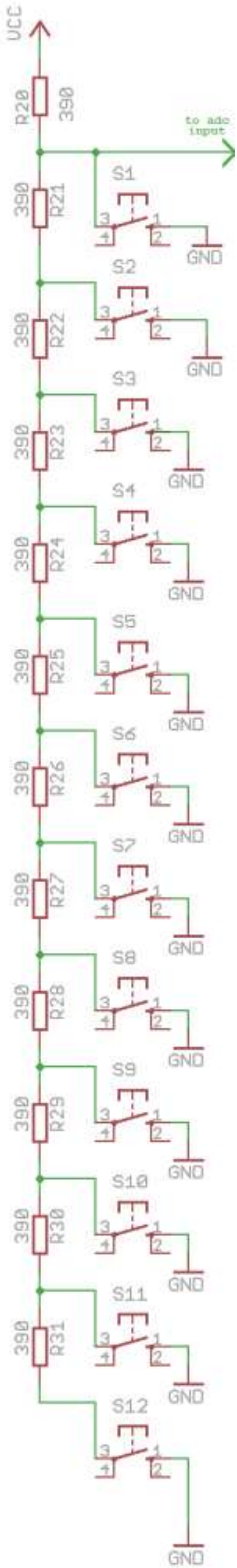
These are used in exactly the same type of circuit as the LDR (voltage divider with a 10K). You must be extremely careful trying to solder to these as the plastic melts so easily. You may find that the use of some type of connector may make your project cheaper!

20.12 Piezo sensor

A piezo make s aperfect vibration sensor in exactly the same voltage divider circuit, especially if you fixed one side of it mechanically to something and the other side is left to float inthe air. You can even buy more sensitive version of this type of sensor – they make great impact sensors.



20.13 Multiple switches and ADC



There is a very convenient way of reading multiple switches with your microcontroller and only use 1 input port.

By making up a long voltage divider as in this diagram and connecting its output to a microcontroller ADC pin, the voltage will change to a different voltage output for every different switch press. This happens because the voltage divider changes the number of resistors in the voltage divider for every different switch press

If no switch is pressed then there is no voltage divider as all the resistors R21 to R31 are unconnected. The input voltage to the ADC will be Vcc (5V) and the ADC reading will be max (1023).

If S1 is pressed then othere is also no voltage divider, however the adc input is now connect to ground (0V) and the adc reading will be 0.

If s2 is pressed there will only be two resistors in the voltage divider and the output will be

Vout = $5v * \frac{390*1}{390*1+390}$ = 0.5V (ADC reading of 0.5/5*1023 = 102)

If S3 is pressed then only 3 resistors will be in the voltage divider and the output will be

Vout = $5v * \frac{390*2}{390*2+390}$ = 0.667V (ADC reading of 0.667/5*1023 = 136

If S4 is pressed then only 4 resistors will be in the voltage divider and the output will be

Vout = $5v * \frac{390*3}{390*4+390}$ = 0.75V (ADC reading of 0.75/5*1023 = 153

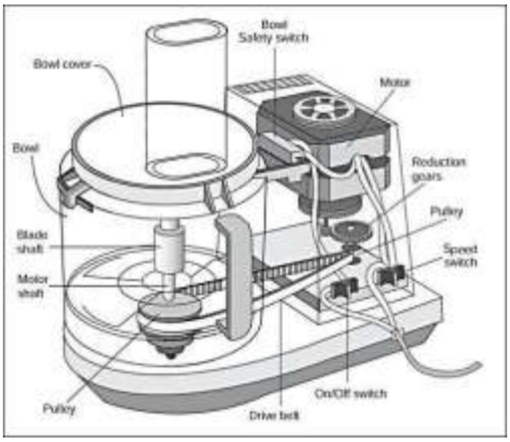
The emerging patterns here are that the output is becoming larger and larger, and the differences between the steps are becoming closer and closer. Note the pattern in the voltages 1/2Vcc , 2/3Vcc, 3/4Vcc, 4/5Vcc, 5/6Vcc, 6/7Vcc....

This means that there is a limit to the number of switches that can be put in this type of circuit.

21 Basic System Design

21.1 Understanding how systems are put together

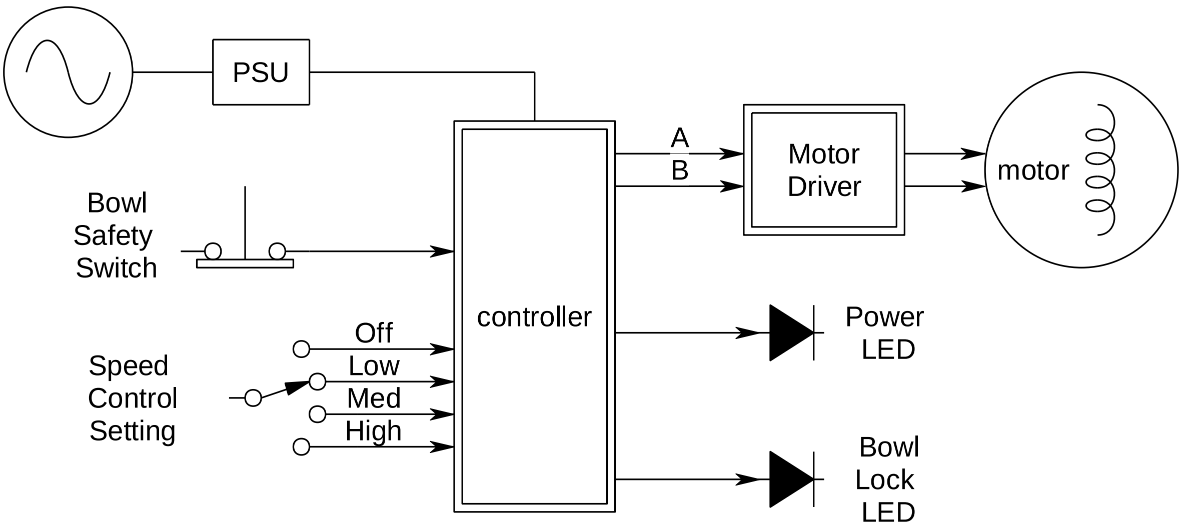
A product or device is not just a collection of components, it is much more, the inventor of the device didn't just combine some parts together, they created a system. They envisaged it as a whole system where all the parts have a unique purpose and together they function to make the product complete.AND they developed it as part of a bigger process.



An example is a food processor.
To analyse the system

1. Draw a system block diagram identifying and describing all the inputs and outputs of the system
   a. Motor – 3 speed
   b. Motor driver electronics
   c. speed control – 4 position switch
   d. bowl safety switch
   e. Power LED, Bowl Lock LED (not shown in picture)
2. Describe in words how these interact with each other, use logic descriptors such as AND, OR and NOT.
3. Design the flowchart to represent the operational logic

21.2 Food Processor system block diagram



21.3 Subsystems

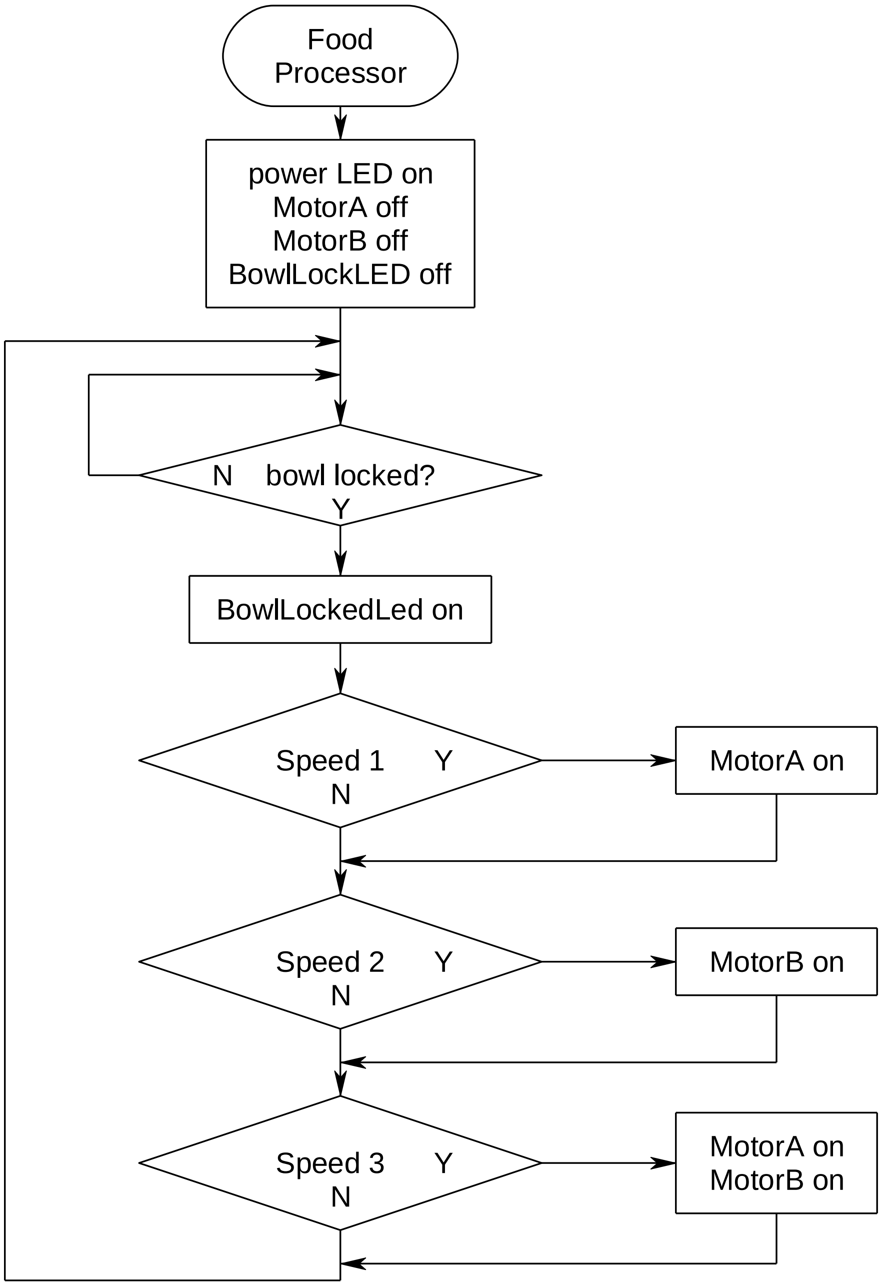
Note that some of the items in the above system are systems themselves. The motor driver, the PSU, the motor and the controller are all systems (the LEDs and switch are components). When we use a system within another system we call it a **subsystem**.

21.4 Food Processor system functional attributes- algorithm

- When power is applied the power LED goes
- When power is applied AND the bowl is securely fitted the Bowl lock LED is on.
- When power is applied AND the bowl is securely fitted AND the speed control is set above zero the motor will run.
- The motor has 2 inputs:
  - When no power is applied to either the motor is off.
  - When power is applied to A it goes slow.

- When power is applied to B it goes medium speed.
- When power is applied to both it goes fast. When the speed control is varied the motor

21.5 Food Processor system flowchart



Here is a first pass at a flowchart for the system. It does however need work as there are a number of problems with it.

Can you identify any?

1. It can be turned on but when the speed switch is turned off, the motor does not turn off.
2. If the bowl is removed while turned on then the motor does not turn off.
3. The BowlLockedLed can never be turned off.

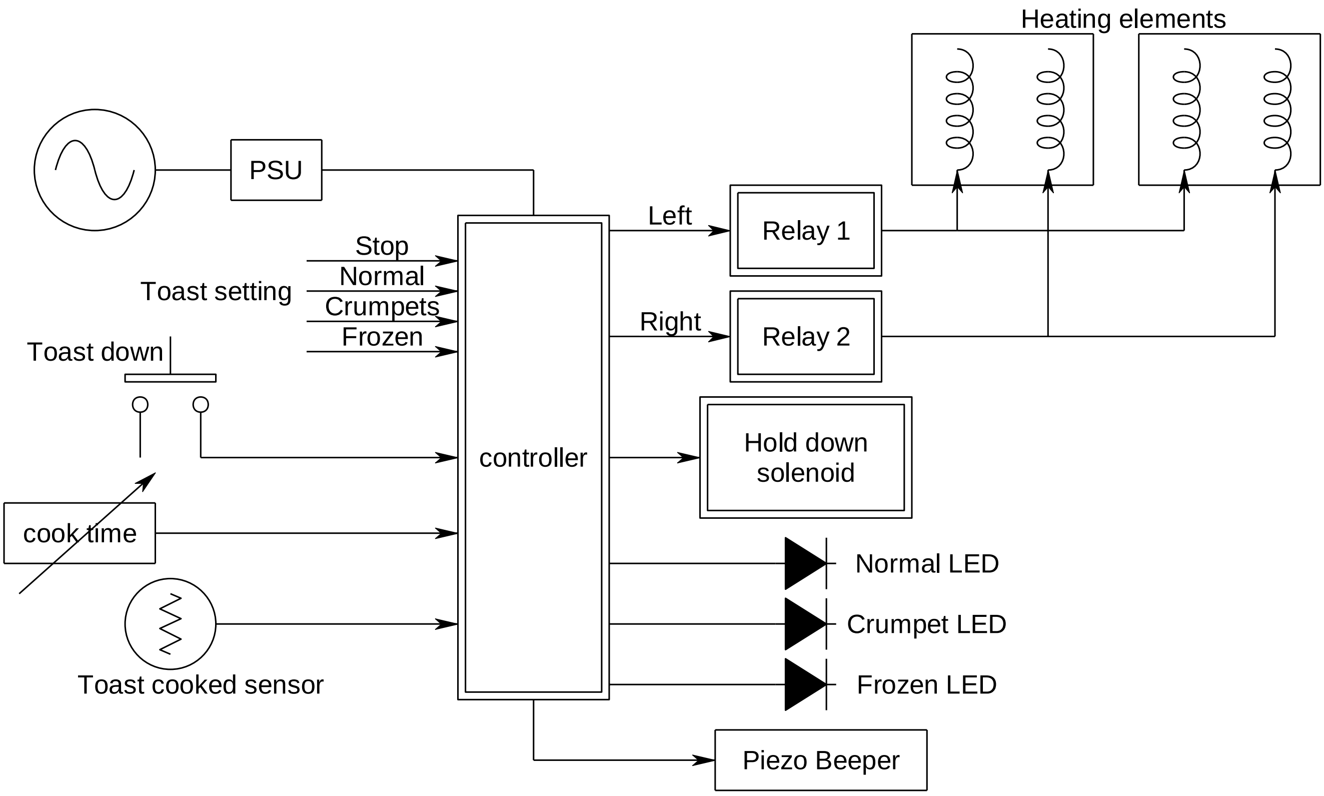
Develop a better flowchart for a program for the food processor.

21.6 Toaster Design

A toaster is another good example of a system.

1. Identify all the parts of the toaster and draw a system block diagram
2. Describe the system operation – how the parts of the system interact with each other
3. Design the flowchart

21.7 Toaster - system block diagram

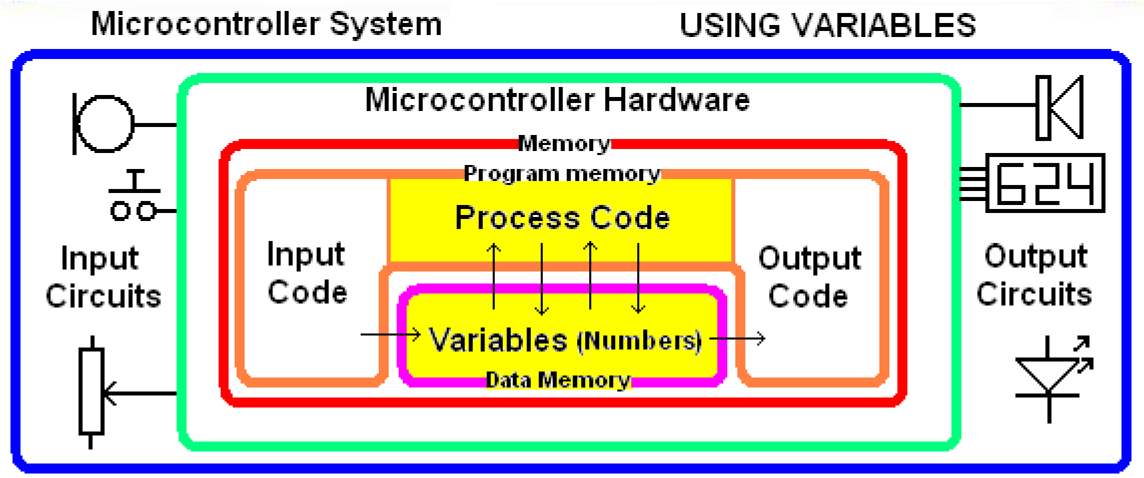


21.8 Toaster Algortihm

Initially: the solenoid is off, the LEDs are off,the piezo is quiet and the elements are off
When the toast lever is pressed down the solenoid is activated to hold the toast down

If the setting is normal both the elements turn on
  and the normal LED comes on
  for the time set by the cook control
If the setting is crumpets, the left comes on max and the right comes on at half power
  and the crumpet LED comes on
  for the time set by the cook control
If the setting is frozen the time is extended by 1 min (either crumpet or normal)
  and the frozen LED comes on
If the sensor detects smoke the solenoid is released and the piezo beeps quickly 4 times
If the time is up the the solenoid is released and the piezo beeps twice

22 Basic System development - Time Tracker.



It is often useful for students to see worked examples; this small project is a worked example not just of a timer project but of the process of development for an electronics project at school.

The process requires several iterations (cycles) of development. For some students the process described here will be trivial (extremely simply), however it is important that students understand the process and can carry it out.

Stage 1:

- Stakeholder consultation
- Initial brief
- Block diagram
- Algorithm
- Flowchart - a model of the internal process that the microcontroller must carry out
- Schematic
- Prototype development
- Program development
- Feedback from stakeholder

Stage 2:

- Refinement of brief – modify/ add/delete specifications
- Modification of schematic/algorithm/flowchart/prototype/program as required
- Feedback from stakeholder

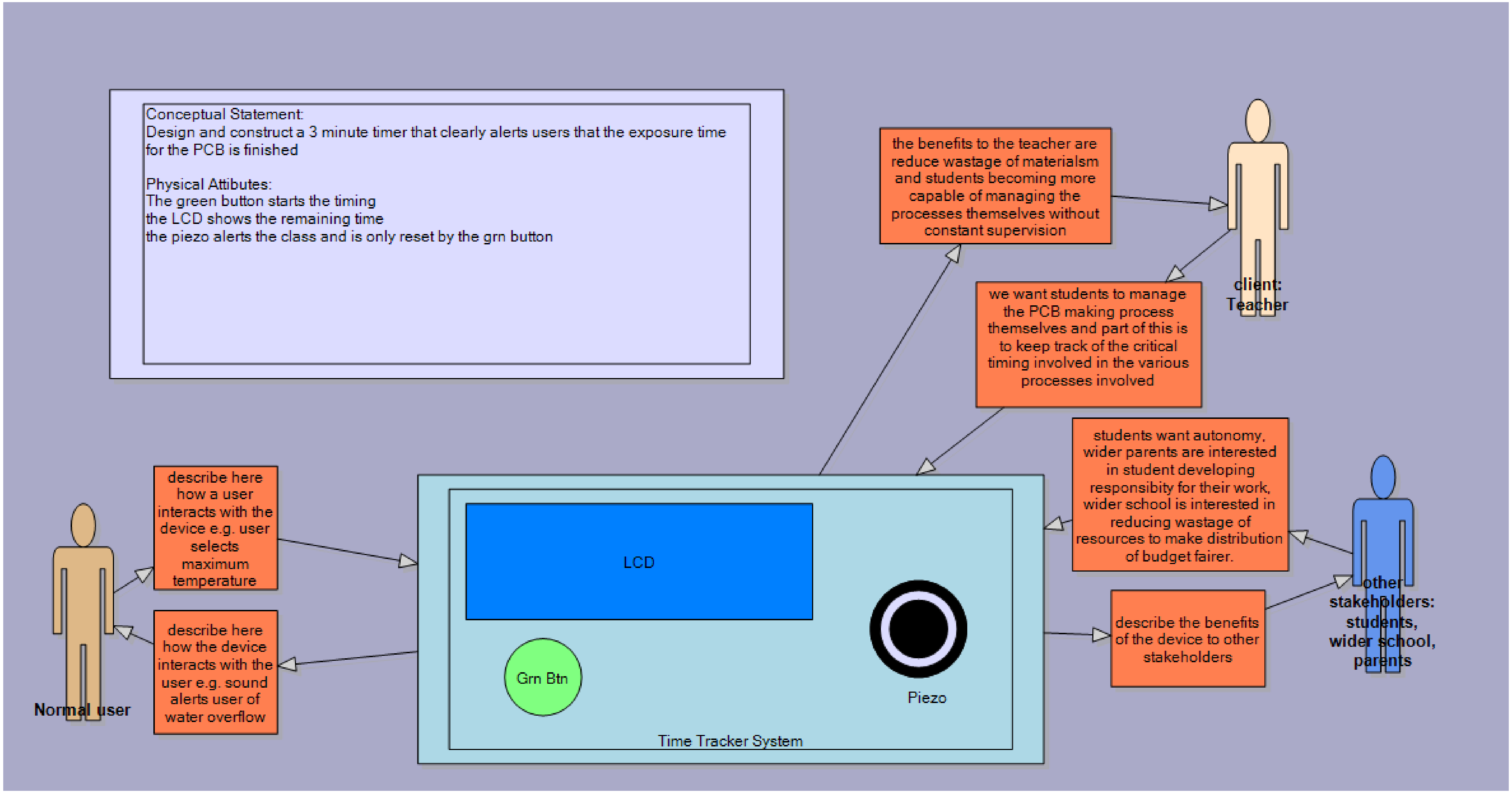
Stage 3:

- Refinement of brief – modify/ add/delete specifications
- Modification of schematic/algorithm/flowchart/prototype/program as required
- Feedback from stakeholder

Stage 4:

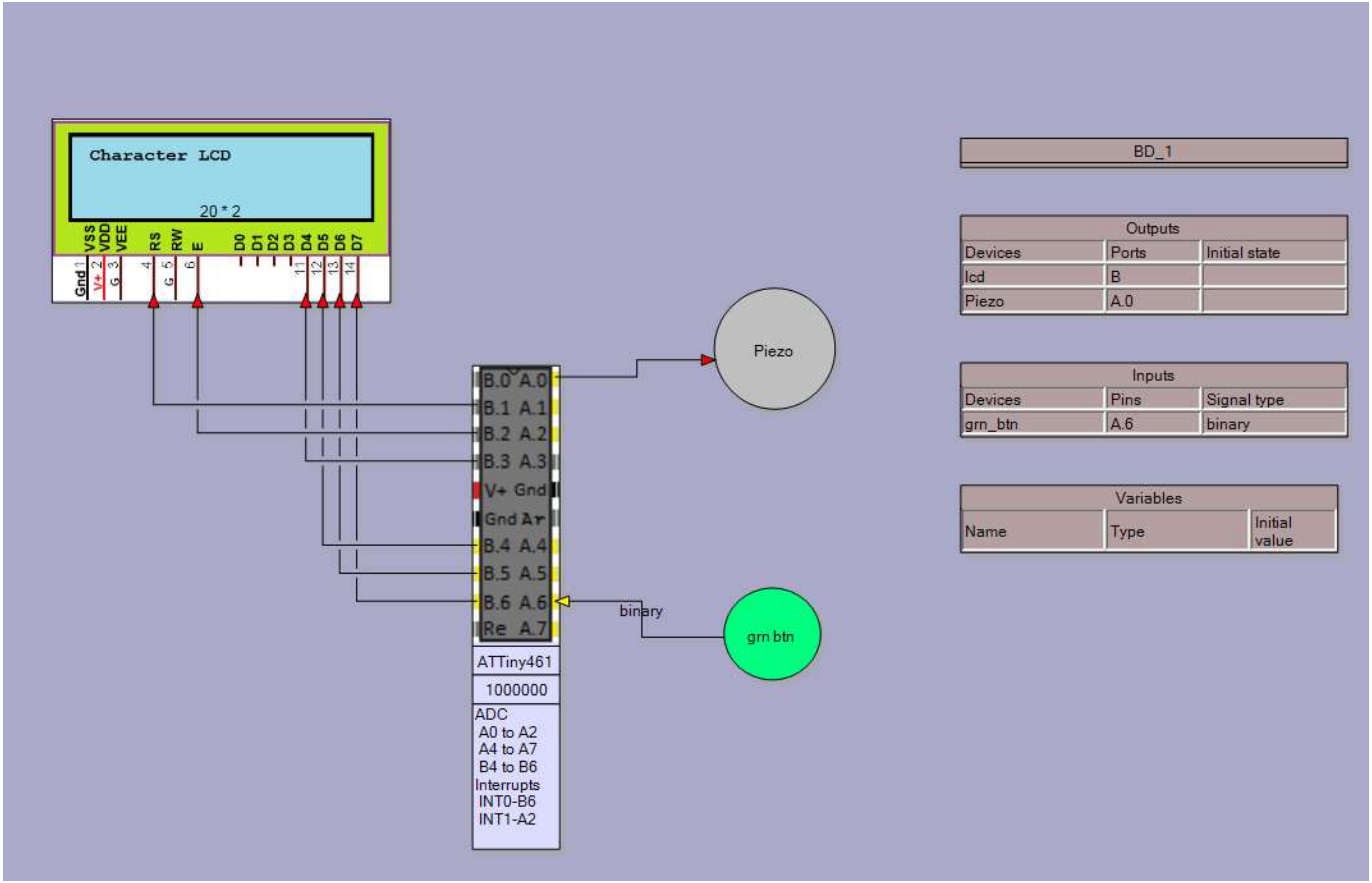
- Refinement of brief – modify/ add/delete specifications
- Modification of schematic/algorithm/flowchart/prototype/program as required
- Evaluation by stakeholder

22.1 System context diagram and brief



The system context diagram is a visual representation of a brief.

22.2 Time tracker block diagram



22.3 Algorithm development

first algorithm:

Initially the display should show 180 seconds count down time.

When the switch is pressed the seconds decrease until 0
then the piezo should play a short tune.

BD_1

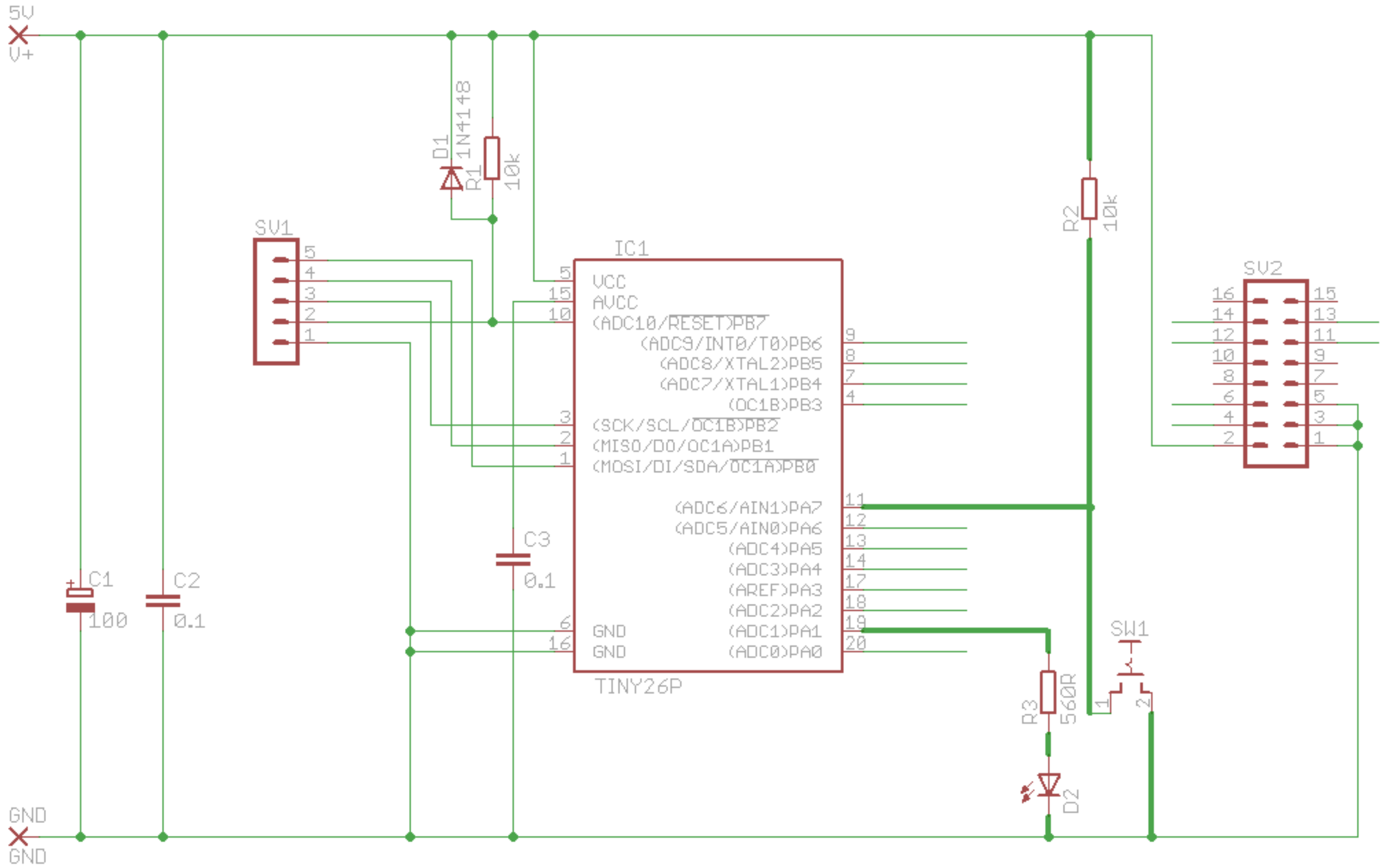
Outputs		
Devices	Ports	Initial state
lcd	B	shows 180 seconds
Piezo	A.0	

Inputs		
Devices	Pins	Signal type
grn_btn	A.6	binary

Variables		
Name	Type	Initial Value
seconds	BYTE	180

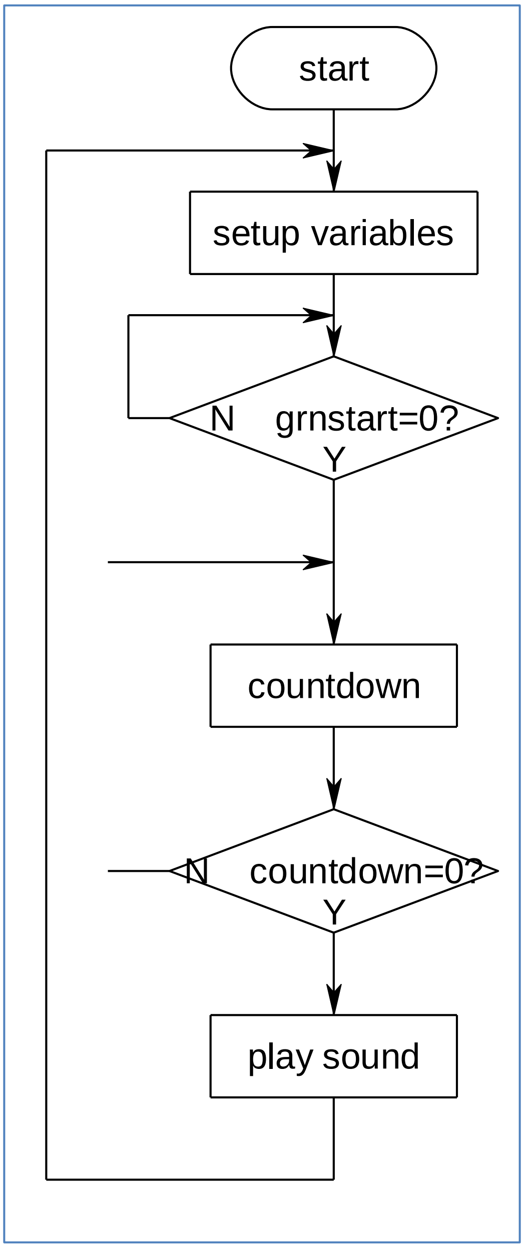
22.4 Schematic

The schematic for the ATTiny26 prototype PCB has been modified to include the components for the switch and LED. Note the LED connection via a current limit resistor, and the switch connection has a pullup resistor.



22.5 Time tracker flowchart and program version 1

```
$crystal = 1000000                      'the speed of the micro
$regfile = "attiny26.dat"               'our micro

'Hardware Setups
' setup direction of all ports
Config Porta = Output                   '
Config Pina.7 = Input
'LCD setup
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.3 , Db6 = Portb.5 ,
Db7 = Portb.6 , E = Portb.1 , Rs = Portb.0
Config Lcd = 20 * 2                     'configure lcd screen

'Hardware Aliases
Grn_sw Alias Pina.7
Piezo Alias Porta.3
```



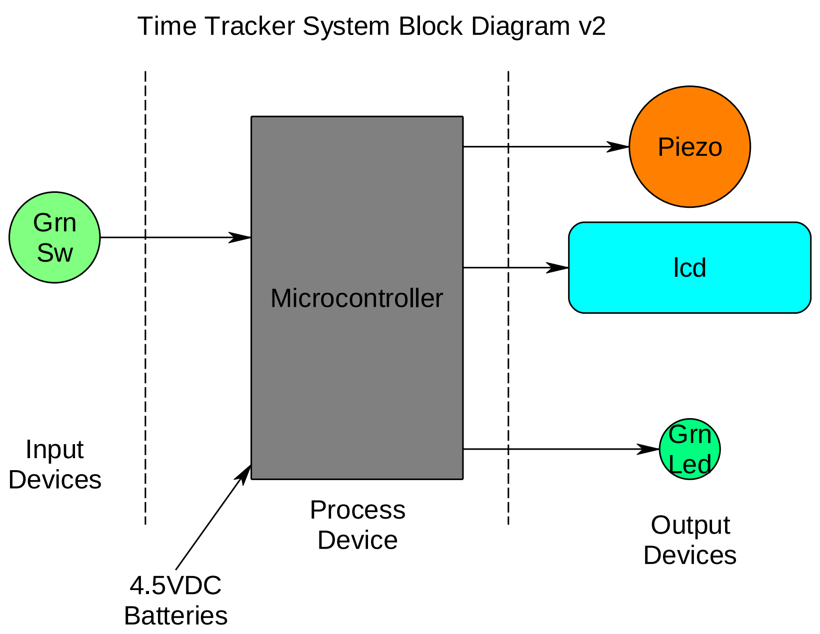

```
'Declare Variable to store timing
Dim Seconds As Byte


'program starts here
Cls
Cursor Off
Lcd "Time Tracker V1"


Do
   'setup countdown value
   Seconds = 5      '5secs for testing
purposes

   'wait for start switch
   Do
   Loop Until Grn_sw = 0
   'need no debounce as next line has a
delay

   'start countdown
   Do
      Waitms 1000
      Decr Seconds
      Locate 2 , 1
      Lcd Seconds
   Loop Until Seconds = 0

   'countdown finished so play sound
   Sound Piezo , 150 , 100
Loop     'return to start
End
```

22.6 Time Tracker stage 2

At this point the student should make contact with their stakeholder or client and show them what has been done. After the client in this case wanted an LED added to the product to show when the timer was not timing and to change to a double beep when the timer times out.

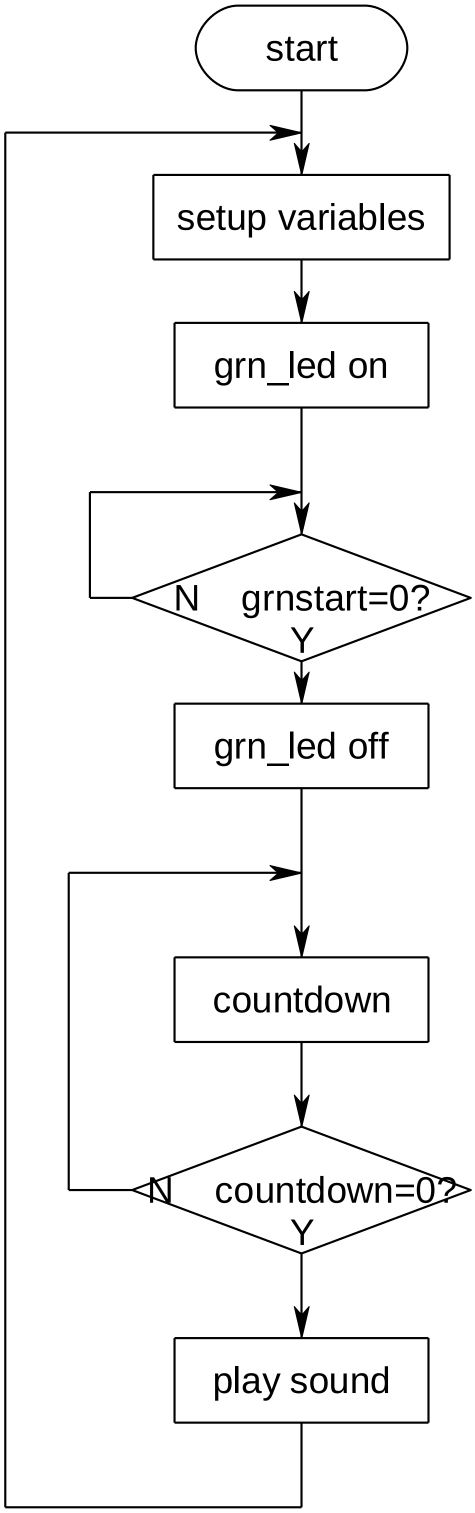
<u>The student makes the following additions to their journal for their project:</u>
Stakeholder consultation carried out and:

1. Brief: new or changed specifications recorded.
2. Algorithm changes described (no need for a new form - just write it into the journal)
3. Block diagram – saves as new version, makes changes and print for journal
4. Schematic: save as new version, make changes and print for journal
5. Layout: make changes to layout in journal or print new version with changes
6. Flowchart – saves as new verison, makes changes and prints for journal
7. Program – saves as new version, makes changes and prints for journal




```
'-------------------------------------------------------------------
'Program Description:
'30 second countdown timer
'lcd displays seconds counting  after switch pressed
'green led is on when not counting
'double beep at end
'-------------------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"                  'our micro
$crystal = 1000000                         'the speed of our micro

'-------------------------------------------------------------------
'Hardware Setups
' setup direction of all ports
Config Porta = Output                      '
Config Pina.7 = Input
'LCD setup
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.3 , Db6 = Portb.5 ,
Db7 = Portb.6 , E = Portb.1 , Rs = Portb.0
Config Lcd = 20 * 2                        'configure lcd screen
```

```
'Hardware Aliases
Grn_led Alias Porta.1
Piezo Alias Porta.3
Grn_sw Alias Pina.7

'-----------------------------------------------
'Declare Variables
Dim Seconds As word 'changed to word as need to count more than 255
```



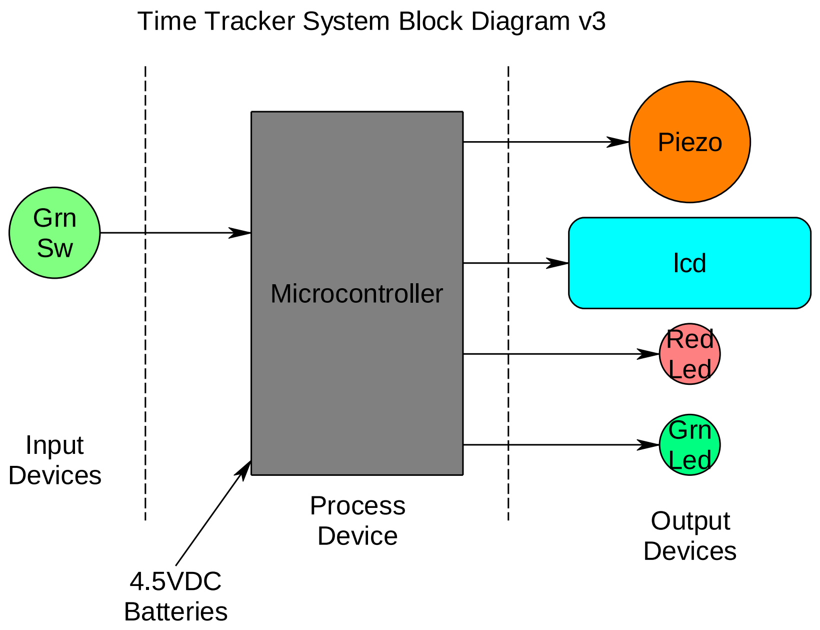

```
'----------------------------
'program starts here


Cls
Cursor Off
Lcd "Time Tracker V2"
Do
   Seconds = 5 initial value to count down
from
   Set Grn_led
   Do
   Loop Until Grn_sw = 0'wait for start
switch
   Reset Grn_led
   Do
      'start countdown
      Waitms 1000
      Decr Seconds
      'display time
      Locate 2 , 1
      Lcd Seconds
   Loop Until Seconds = 0
   'countdown finished so play sound
   Sound Piezo , 150 , 100
   Waitms 50
   Sound Piezo , 150 , 100

Loop
End                 'end program
```

22.7 Time Tracker stage 3

At this point the student should make another contact with their stakeholder or client and show them what has been done. After this the client wanted a second (red) LED added to the product to flash while the timer was timing.

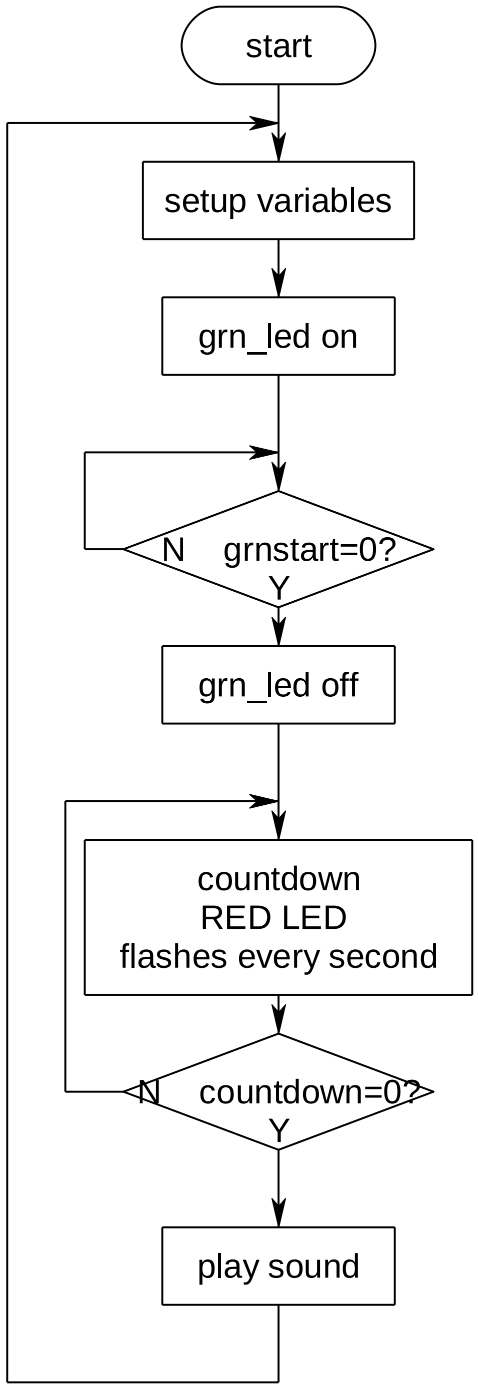
<u>The students makes the following additions to their journal for their project:</u>
Stakeholder consultation carried out and:

1. Brief: new or changed specifications recorded.
2. Algorithm: changes described (no need for a new form - just write it into the journal)
3. Block diagram: saves as new version, makes changes and prints for journal
4. Schematic: save as new version, make changes and print for journal
5. Layout: make changes to layout in journal or print new version with changes
6. Flowchart: saves as new verison, makes changes and prints for journal
7. Program: saves as new version, makes changes and prints for journal




```
'-------------------------------------------------------------------
'Program Description:
'30 second countdown timer
'lcd displays seconds counting  after switch pressed
'green led is on when not counting
'double beep at end
'red led flashes once per second
'-------------------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"                'our micro
$crystal = 1000000                       'the speed of our micro

'-------------------------------------------------------------------
'Hardware Setups
' setup direction of all ports
Config Porta = Output                    '
Config Pina.7 = Input
'LCD setup
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.3 , Db6 = Portb.5 ,
Db7 = Portb.6 , E = Portb.1 , Rs = Portb.0
```

```
Config Lcd = 20 * 2                       'configure lcd screen

'Hardware Aliases
Grn_led Alias Porta.1
Red_led Alias Porta.0
Piezo Alias Porta.3
Grn_sw Alias Pina.7

'----------------------------------------
'Declare Variables
Dim Seconds As word
```




```
'----------------------
'program starts here
Cls
Cursor Off
Locate 1 , 3                          '
Lcd "Time Tracker V3"
Do
   Set Grn_led
   Seconds = 5   'initial value to count down
from
   Do
   Loop Until Grn_sw = 0    'wait for start
switch
   Reset Grn_led
   Do
      'countdown
      Set Red_led    'added flashing red LED
      Waitms 50
      Reset Red_led
      Waitms 950
      Decr Seconds
      'display time
      Locate 2 , 10
      If Seconds < 10 Then Lcd "0"
      Lcd Seconds
   Loop Until Seconds = 0
   'beeps
   Sound Piezo , 150 , 100
   Waitms 50
   Sound Piezo , 150 , 100

Loop
End              'end program
```

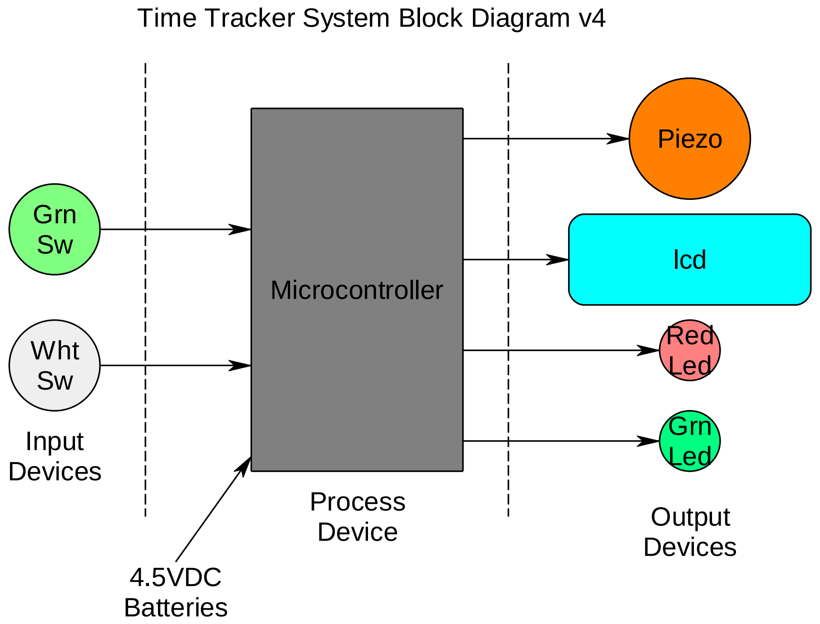
22.8 Time Tracker stage 4

At this point the student made yet another contact with their stakeholder or client and showed them what has been done. After this the client wanted a significant change to the project; they thought the timer would be really useful if the time delay could be changed. Specifically they want to be able to push a second switch to increase the count time from 30 to 100 seconds in amounts of 30 seconds; e.g. 30-60-90-120-150-180-210-240-270-300 seconds.

<u>The students makes the following additions to their journal for their project:</u>
Stakeholder consultation carried out and:

1. Brief: new or changed specifications recorded.
2. Algorithm: changes described (no need for a new form - just write it into the journal)
3. Block diagram: saves as new version, makes changes and prints for journal
4. Schematic: save as new version, make changes and print for journal
5. Layout: make changes to layout in journal or print new version with changes
6. Flowchart: saves as new verison, makes changes and prints for journal
7. Program: saves as new version, makes changes and prints for journal

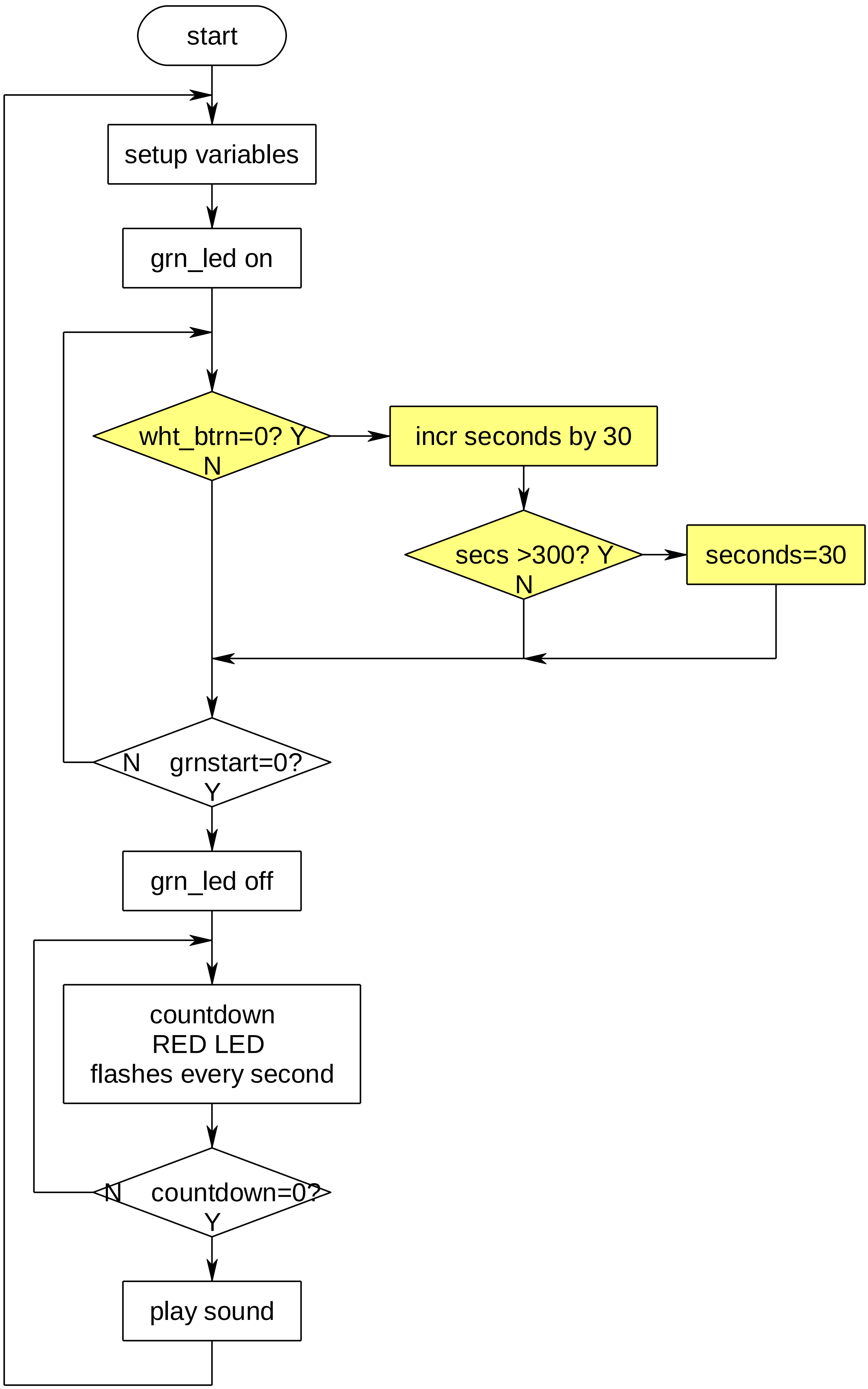
Of course some students may be able to go straight to this final version of the product straight away; however in doing this they are missing out on critical marks, as the highest grades come from stakeholder consultations and subsequent modification to their project.



This final version of the block fiagram has all of the components to date.

The algorithm now has been modified to include:

While waiting for the user to press the green start button, f they press the white button the time will increase in amount sof 30 seconds to a maximum of 300 seconds.


start
setup variables
grn_led on
wht_btrn=0? Y
N
incr seconds by 30
secs >300? Y
N
seconds=30
N grnstart=0?
Y
grn_led off
countdown
RED LED
flashes every second
N countdown=0?
Y
play sound

```
'------------------------------------------------------------------
'Title Block
' Author:     A. Student
' Date:       Jul 09
' File Name:  TimeTrackerV4
'------------------------------------------------------------------
'Program Description:
'30 second countdown timer
'lcd displays seconds counting  after switch pressed
'green led is on when not counting
'double beep at end
'red led flashes once per second
'added ability to increase seconds count with white switch
'added switch labels to LCD screen
'------------------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "attiny26.dat"                   'our micro
$crystal = 1000000                          'the speed of our micro

'------------------------------------------------------------------
'Hardware Setups
' setup direction of all ports
Config Porta = Output                       '
Config Pina.6 = Input
Config Pina.7 = Input
'LCD setup
Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.3 , Db6 = Portb.5 ,
Db7 = Portb.6 , E = Portb.1 , Rs = Portb.0
Config Lcd = 20 * 2                         'configure lcd screen


'Hardware Aliases
Grn_led Alias Porta.1
Red_led Alias Porta.0
Piezo Alias Porta.3
Grn_sw Alias Pina.7
Wht_sw Alias Pina.6

'------------------------------------------------------------------
'Declare Constants
Const Debouncetime = 10
Deflcdchar 0 , 32 , 4 , 2 , 31 , 2 , 4 , 32 , 32
'------------------------------------------------------------------
'Declare Variables
Dim Seconds As Word
'Initialise Variables
Seconds = 30
'------------------------------------------------------------------
'program starts here
Cls
Cursor Off
```

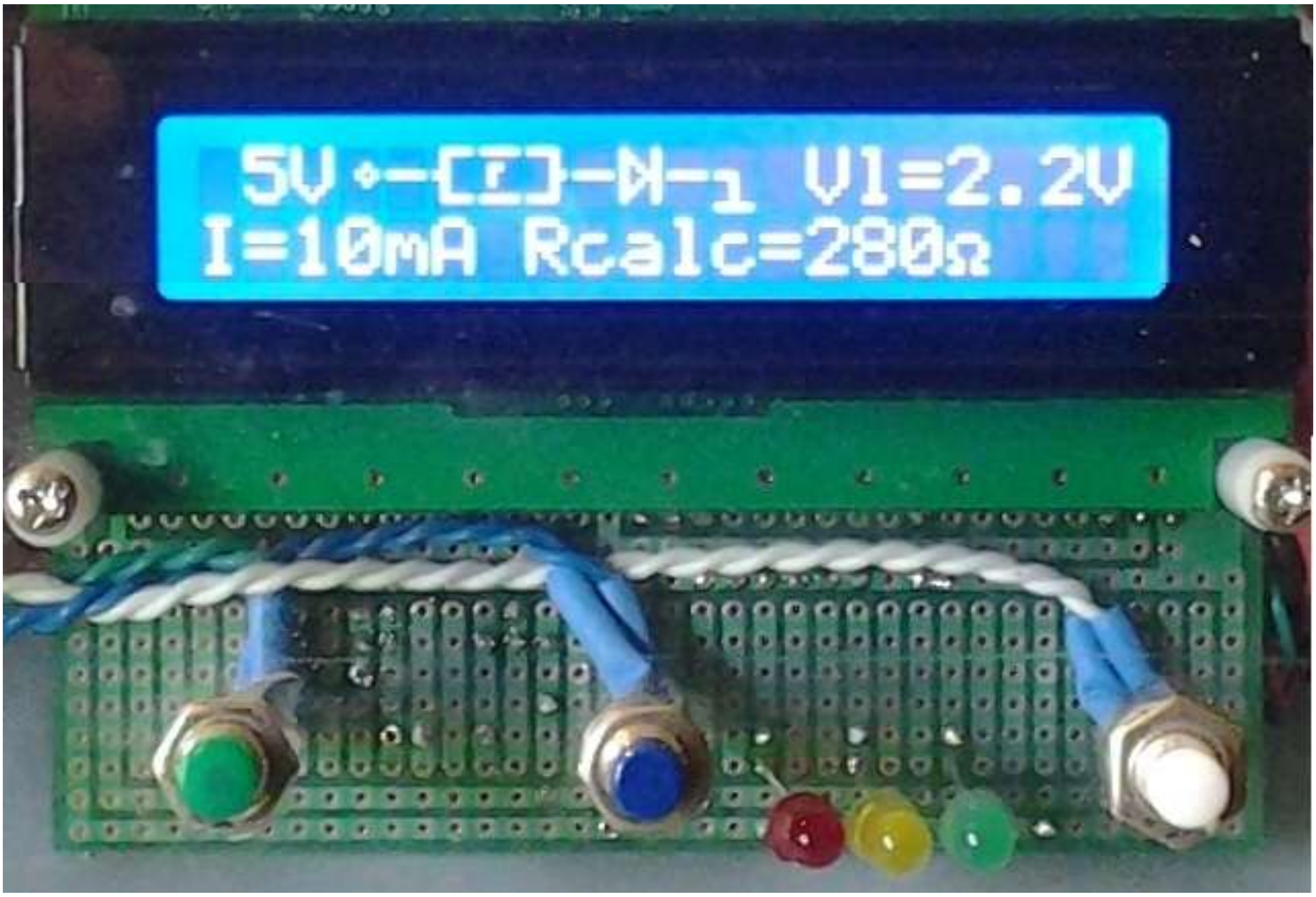
```
Do
  'setup initial lcd display
   Cls
   Set Grn_led
   Lcd "Time Tracker  start" ; Chr(0)
   Seconds = 30                     'initial value to count down from
   Locate 2 , 1                    'display labels for switches on the LCD
   Lcd "count=         incr" ; Chr(0)
   Locate 2 , 7
   Lcd Seconds

   'wait for start switch, allow user to change time while waiting
   Do
      'allow user to increase count in amounts of 30 seconds
      If Wht_sw = 0 Then
         Seconds = Seconds + 30
         If Seconds > 300 Then Seconds = 30       'set max
         Locate 2 , 7
         Lcd Seconds ; "  "                 'add clear feature " "
         Waitms Debouncetime                'must debounce switch
         Do                                 'wait for switch up
         Loop Until Wht_sw = 1
         Waitms Debouncetime                'waita little longer
      End If
   Loop Until Grn_sw = 0                    'wait for start switch
   Reset Grn_led

   'countdown
   Do
      Set Red_led
      Waitms 50
      Reset Red_led
      Waitms 950
      Decr Seconds
      'display time
      Locate 2 , 7
      If Seconds < 10 Then Lcd "0"
      Lcd Seconds ; " "          'space blanks unwanted digits on lcd
   Loop Until Seconds = 0
   'beeps
   Sound Piezo , 150 , 100
   Waitms 50
   Sound Piezo , 150 , 100

Loop
End                                         'end program
```

23 Basic maths time

Microcontrollers only store numbers, soit follows logically that they can do maths with these numbers.

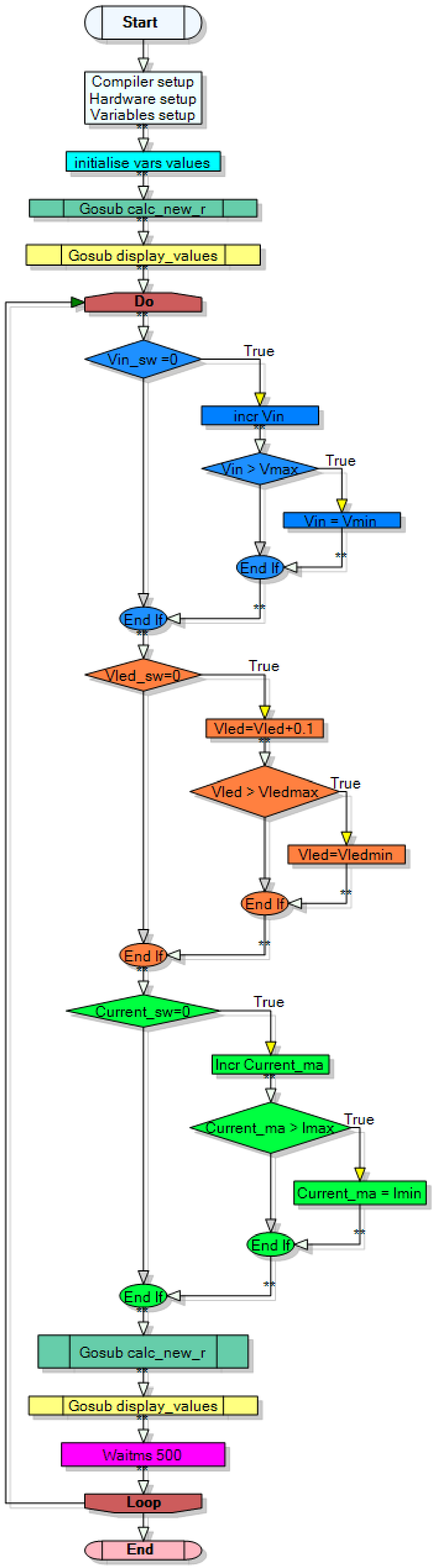
Here is a program that makes use of some maths and some different types of variables.

23.1 Ohms law calculator



Specifications

- Pressing the first button increases the voltage in the circuit by 1 Volt
    - A maximum of 24V
    - A minumium of 3V
    - After 24V it goes back to 3V
- Pressing the second button increases Vl by0.1V. Vl (short for Vled) is the manufacturers voltage specification for the LED
    - A maximum of 4.0V
    - A minimum of 1.5V
    - After 4.0V it wraps back to 1.5V
- Pressing the third button increases the current that you want by 1mA
    - A maximum of 50mA
    - A minimum of 1mA
    - After 50mA it wraps back to 1mA
- After any button press the new resistor value is calculated and displayed
  Take note here that I separate the processing from the output code by writing the two different and separate things the micro must do; first calculate (process code) and then display (output code).



BD_1

Outputs		
Devices	Ports	Initial state
lcd	C	
Grn_Led	B.5	
Yel_Led	B.6	
Red_Led	B.7	

Inputs		
Devices	Pins	Signal type
Vin_sw	B.2	binary
Vled_sw	B.3	binary
Current_sw	B.4	binary

Variables		
Name	Type	Initial Value
Vin	BYTE	5
Vled	BYTE	2.2
Current_ma	BYTE	10
Temp	SINGLE	
string_val	STRING * 3	

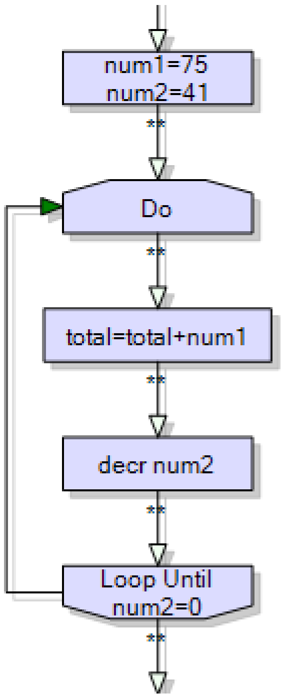
The flowchart for the program reveals my thinking while designing the program flow.

- Again I have separated out the process code from the output code. This is an important concept in programming; the separation of different functions.
    - Even though the only time we use these two subroutines is together it is better to separate them because they do different things. This makes the code easier to understand and easier to recycle into other programs.
- I put the subroutines calc_new_r and display_values at the beginning of theprogram so that they are used once. If you don't do this then the program sits there with a blank LCD until the user presses a button.
    - Alternatively you could put in an instructions screen at the beginning that told the user what to do and then either automatically times out or waited until the user pressed a button to continue.
- I could use the Bascom debounce command rather than reading the switches with IF-THEN. However I chose not to because the debounce command doesn't allow auto repeat (you have to let the button go before you can press it again to increase the value). I felt that the program would be easier to use if you could just hold the button down and the values would increase at a regular rate.
    - This means however that my program needs a delay in the loop somewhere otherwise the values count up much too quickly when a button is pressed.
    - See where the delay (waitms 500) is in the program, it is in the main loop. Now the problem with putting a 500mS delay in the main loop has been covered before. It is that a button can be pressed and the micro can easily miss it because it might be in the waitms 500 doing nothing.
    - This delay should be moved into each switch press loop, because then it only occurs when a switch is pressed and not at any other time. So why didn't I do this already, because I wanted you to learn about it again!
    - If you wanted to you could write a really neat debounce command that checked the switch and if it was held down increased at a slow rate.
- I have put the calculation and lcd drawing into the main loop rather than into each loop.
    - This is redundant. What I mean by redundant is that it is put there but not doing anything useful. This code only does something when a value changes otherwise it just recalculates using the same values and redraws the same values on the LCD.
    - There is no real reason for this, and there is little side effect from it either in this program. If these two routines took a long time then we would have the same effect as putting the 500mS delay into the main loop; we could miss switch presses while the micro was busy doing nothing. However they don't take long enough to cause a problem. So putting them in the main loop or putting them into each subroutine isn't important because of speed.
    - In general these would be better in the subroutines because it is better programming practice. There is one reason why these might be better in the main loop and not each switch press. This actually makes our program take up less room in program memory. This is hardly noticeable (its just 4 Gosub lines that we save) in this program but I say it because there is an important concept in programming here.
    - We can reduce size of program code through strategic thinking and understanding of how a program runs.

```
'*****************************************************
'Compiler Setup
$crystal = 8000000
$regfile = "m8535.dat"

'*****************************************************
'Hardware Configs
Config PORTA = Output
Config PORTB = Output
Config PORTC = Output
Config PORTD = Output
Config PINB.2 = Input       'Vin_sw
Config PINB.3 = Input       'Vled_sw
Config PINB.4 = Input       'Current_sw

'Character LCD config
Config Lcdpin=pin , Db4 = PORTC.2 , Db5 = PORTC.3 , Db6 = PORTC.4 , Db7 = PORTC.5 , E = PORTC.1
, Rs = PORTC.0
Config Lcd = 20 * 4
'*****************************************************

'Hardware aliases
'inputs
Vin_sw Alias PINB.2
Vled_sw Alias PINB.3
Current_sw Alias Pinb.4
'activate internal pullups for switches
Set Portb.2                             'Grn_sw
Set Portb.3                             'Blu_sw
Set Portb.4                             'Wht_sw

'outputs
Grn_Led Alias PORTB.5
Yel_Led Alias PORTB.6
Red_Led Alias PORTB.7
'*****************************************************
'Dimension Variables
Dim Vin As Byte                         '3 to 24V
Dim Current_ma As Byte                  '1 to 50mA
Dim Vled As Single                      '2.0 to 4.0V
Dim R As Word
Dim String_val As String * 3
Dim Temp As Single

'Initiliase Variables
Vin = 5
Current_ma = 10
Vled = 2.2

'constants, if you use constants then it is easier to make changes to the program'
Const Vmin = 3
Const Vmax = 24
Const Vledmax = 4
Const Vledmin = 1.5
Const Imin = 1
Const Imax = 50

'define the LCD chars
Deflcdchar 0 , 15 , 8 , 8 , 24 , 8 , 8 , 15 , 32        ' res symbol lhs
Deflcdchar 1 , 31 , 32 , 12 , 8 , 8 , 32 , 31 , 32        ' res symbol with r
Deflcdchar 2 , 30 , 2 , 2 , 3 , 2 , 2 , 30 , 32        ' rs symbol rhs
Deflcdchar 3 , 17 , 25 , 21 , 19 , 21 , 25 , 17 , 32        'diode
Deflcdchar 4 , 32 , 32 , 32 , 28 , 4 , 4 , 4 , 31        'ground
Deflcdchar 5 , 32 , 32 , 2 , 5 , 2 , 32 , 32 , 32        ' circle
```

```
'------Program starts here --------
Cls
Cursor Off
Gosub calc_new_r
Gosub Display_values
Do
   If Vin_sw =0 Then
      Incr Vin
      If Vin > Vmax Then
         VIn = Vmin
      End If
   End If
   If Vled_sw=0 Then
      Vled = Vled + 0.1
      If Vled > Vledmax Then
         Vled=Vledmin
      End If
   End If
   If Current_sw=0 Then
      Incr Current_ma
      If Current_ma > Imax Then
         Current_ma = Imin
      End If
   End If
   Gosub calc_new_r
   Gosub Display_values
   Waitms 500
Loop
'Subroutines
Vin_sw_press:
   Incr Vin
   If Vin > Vmax Then Vin = Vmin
Return

Vled_sw_press:
   Vled = Vled + 0.1
   If Vled > Vledmax Then Vled = Vledmin
Return

Current_sw_press:
   Incr Current_ma
   If Current_ma > Imax Then Current_ma = Imin
Return

'calcultae the resistor value
Calc_new_r:
   'voltage v across R = vin-vled
   Temp = Vin - Vled
   'r=v/i
   Temp = Temp / Current_ma
   Temp = Temp * 1000
   R = Temp                            'convert single to word
Return

Display_values:
   'top line
   Locate 1 , 1
   If Vin < 10 Then Lcd " "            'put in a leading zero if less than 10
   Lcd Vin ; "V"                       'display Vin
   'display graphic
   Lcd Chr(5) ; "-" ; Chr(0) ; Chr(1) ; Chr(2) ; "-" ; Chr(3) ; "-" ; Chr(4)
   'display voltage
   Lcd " Vl="
   String_val = Fusing(vled , "#.#")   'trick to get 1 decimal digit
   Lcd String_val ; "V"
   'second_line:
   Locate 2 , 1
   Lcd "I="
   If Current_ma < 10 Then Lcd " "
   Lcd Current_ma ; "mA"
   Lcd " Rcalc=        "
   Locate 2 , 14
   Lcd R ; Chr(244)
Return
```

23.2 more maths - multiplication

Process	Notes
Issue: Multiply two numbers together using only addition e.g. AxB=Answer	*Pretty much all microcontrollers do multiplication inside their hardware nowadays but its useful as a learning exercise.*
Algorithm: Add A to the answer B times e.g. 5 x 4 = 5+5+5+5	*Finding the right words to describe the algorithm can be difficult at times, you need to concise, accurate and clear. This can be a step students struggle with.*
Variables: (memory locations to store data in) numA – byte size numB – byte size Answer – word size	*Choose useful names and think about the size of the variable (a byte stores 0-255, a word 0-65535, an integer stores -32768 to 32767, a long stores -2147483648 to 2147483647)*
Flowchart:  	*Note the shapes of the elements:* *Start and end* *Inputs and outputs* *Processes* *Decisions* *Learn the process of keeping track of how many times something is done. A variable is used to count the number of times a loop is carried out. In this case the variable is decreased each time through the loop until it is 0. An alternative is to increase a variable until it reaches a specific value.* *Within a microcontroller though it is often faster to test a variable against 0 than some other number.*
Test the flowchart with an example (see table below)	*Does it work?* *Note how the columns in the test follow the same order as the processes in the loop.* *This stage can be a little confusing and often we can be out by 1 either way (if it is then our answer might not be 54 but 48 or 60)* *If you get wrong answers after a loop check that you are decreasing or increasing them the right number of times.*
Identify the control statements to be used.	*In BASCOM there are several control mechanisms to manage loops.*

Answer	Num2
6	8
12	7
18	6
24	5
30	4
36	3
42	2
48	1
54	0

```
' SimpleMultiplicationV1.bas
$crystal = 1000000
$regfile = "attiny461.dat"
Config Porta = Output
Config Portb = Output
Config Pina.3 = Input

Dim I As Byte
Dim Num1 As Byte
Dim Num2 As Byte
Dim Answer As Word

'************************************
Num1 = 6
Num2 = 9
Answer = 0
Do
   Answer = Answer + Num1
   Decr Num2
Loop Until Num2 = 0

'************************************
Num1 = 6
Num2 = 9
Answer = 0
For I = 0 To Num2
   Answer = Answer + Num1
Next

'************************************
Num1 = 6
Num2 = 9
Answer = 0
For I = Num2 To 0 Step -1
   Answer = Answer + Num1
Next

'************************************
Num1 = 6
Num2 = 9
Answer = 0
While Num2 > 0
   Answer = Answer + Num1
   Decr Num2
Wend
End
```

If you copy this code into BASCOM-AVR, then save it and compile it you can try it out using the simulator (F2).

Do-Loop Until…

For-Next…
this requires another variable to act as the loop counter, and can either count up or count down.

While – Wend

When you run this program you will find that two of them work correctly and two do not! You need to understand which and fix them; so watch carefully the values of the variables in the simulator and fix the two that need fixing.

23.3 Algorithms for multiplication of very large numbers

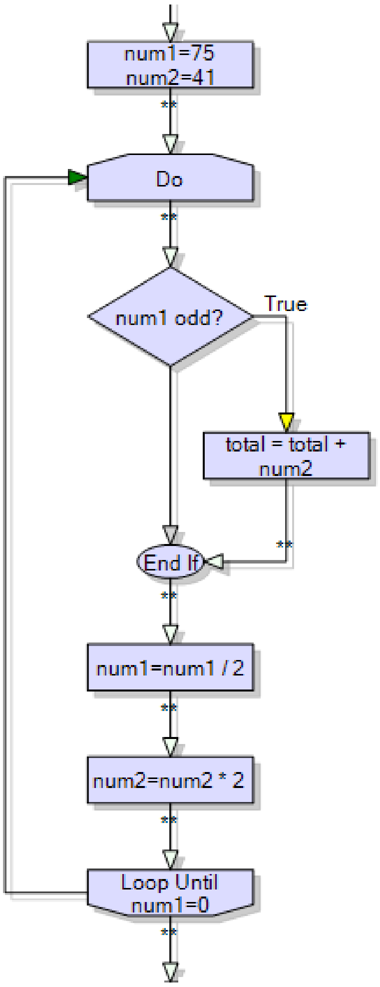
The previous code is OK for small to medium size problems however there are much more efficient algorithms; here are 2 alternatives.

'Peasant' Multiplication 75 x 41

75 41
37 ~~82~~
18 ~~164~~
9 ~~328~~
4 ~~656~~
2 ~~1312~~
1 2625
3075

Write down the Algorithm:
Divide the first number by 2 (ignore remainder) and multiply the second number by 2.
If the second number is odd add it to the total.
Keep doing this process until after the first number is 1.

What variables will be needed:
Num1, Num2, Total



Program:

```
' PeasantMultiplicationV1.bas

$crystal = 1000000
$regfile = "attiny26.dat"

Config Porta = Output
Config Portb = Output

Dim Temp As Word
Dim Num1 As Word
Dim Num2 As Word
Dim Answer As Word

Num1 = 16
Num2 = 39
Answer = 0

'note again the use of do-loop as we don't know
how many times the loop needs to be repeated
Do
 (Mod is used to find if a number is odd or even)
  Temp = Num1 Mod 2
  If Temp = 1 Then Answer = Answer + Num2
  Num1 = Num1 / 2
  Num2 = Num2 * 2
Loop Until Num1 = 0

End
```

Long Multiplication 41,231 x 3,1231

```
      41,321
     x 3,131
      41,321
   1,239,630
   4,132,100
 123,963,000
 129,376,051
```

Write down the Algorithm:

What variables will be needed:

Flowchart:

23.4 Program ideas - algorithm and flowchart exercises

1. In this game the first person picks a number between 1 and 10 and the other person must guess this number in 4 or less guesses. If you play this game a few times with someone you will get a feel for the algorithm (the process for solving the problem) . Can you write the process down?

2. This is a game played with any number of players who take turns saying a number. The first player says "1" and each player in turn increases the number by 1, 2, or 3, but may not exceed 21; the player forced to say "21" loses. There is a winning strategy for this game you will need to research it or figure it out to be able to write a program that can beat a human opponent.

3. A factory fills drink bottles; it has a machine that puts the drink bottles into cartons and full cartons onto pallets.
3A. Design an algorithm and flowchart that counts 24 bottles into each carton and keeps track of the number of cartons.

3B. Extend this in a second algorithm and flowchart that tracks the number of bottles and the number of cartons, when number of cartons is over 48 then increase the number of pallets.

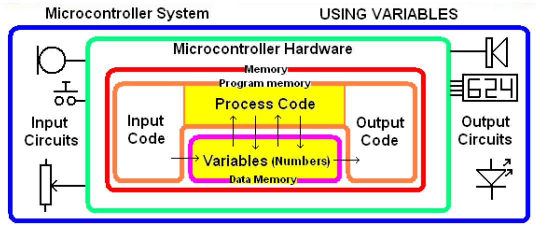
4. A program marks test scores and gives grades of N, A, M, or E based upon the following scores
0% to 33% = N, 34% to 55% = A, 56% to 83% = M 83% to 100% = E
Write the algorithm and draw the flowchart for this process.

5. Design an algorithm and flowchart for a program that gets a player to guess a random number from 1 to 1000.
If correct, then display the number of guesses and start again
If incorrect then give as too high' or 'too low'
When the number of guesses goes over 8 the player loses

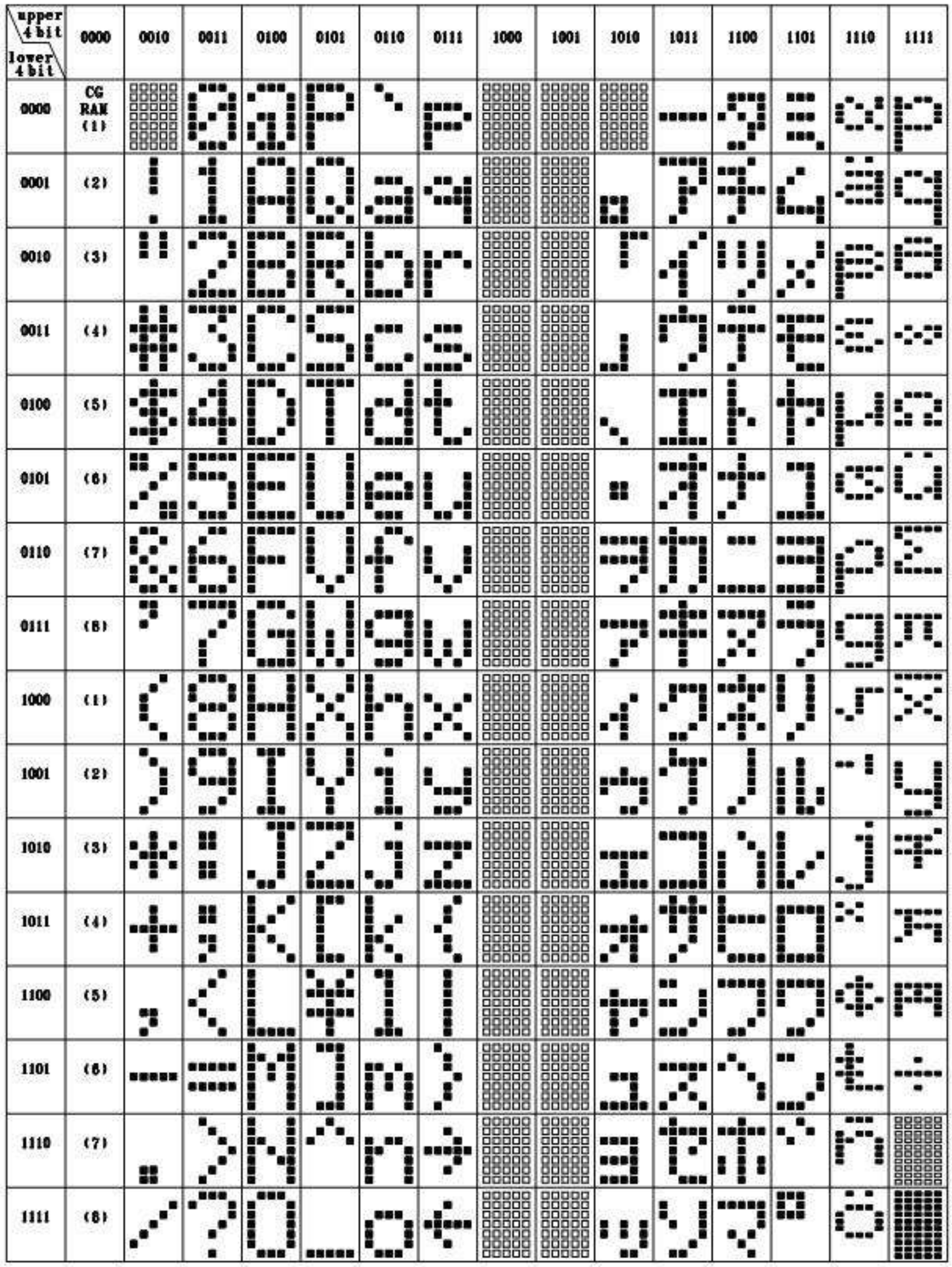
6A. a golf course watering system monitors the time and moisture level of the ground and waters the grass in the early evening if it is needed.
6B. the watering system comes on for 30 minutes then waits 60 minutes to measure the moisture level and comes on for a second watering if it is below a fixed level.

7.Design an algorithm and flowchart for a program that calculates powers eg. $2^5 = 32$ (use only addition and loops)

24 Basic string variables



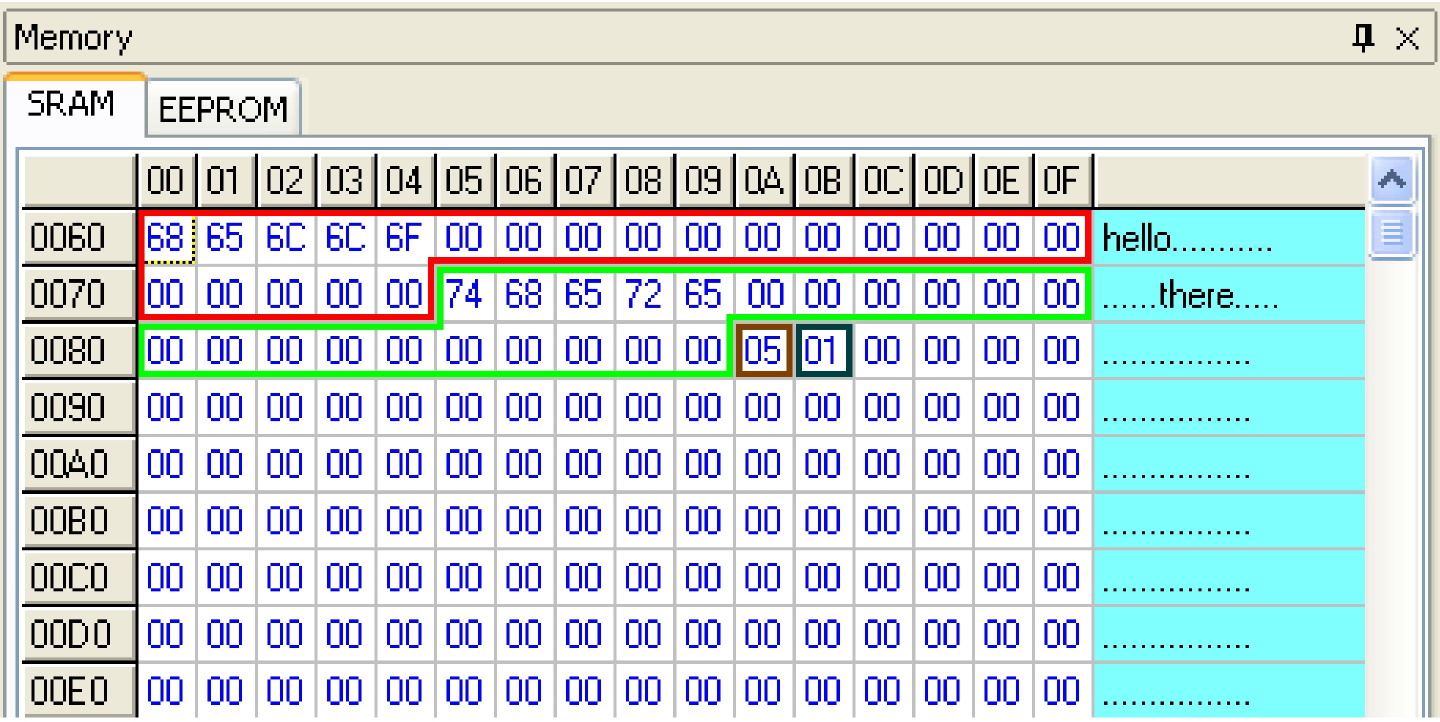
So far we have used constants on the display such as lcd”Hello”.
But what if we want our text to vary e.g. different names and addresses or different colours or different days of the week.
All computer languages allow you to store this text in a variable called a STRING. Computers all store text in the same way too. Ram stores only numbers so to store text in RAM we store a code for each letter of the text string.
This table gives us the binary code for each character e.g. ‘A’ is 01000001 or 65 in decimal.
In a program text can be displayed using the command LCD CHR(...), so to diaplay an A LCD CHR(65).



A string is a variable that is a collection of letters (and digits) such as “*My name is Fred*” or “*37 Frost Road, Mount Roskill*”
When you dimension a string you must think about how big it might become during the time your program will use it, and then allocate enough memory for it. e.g. dim address as string * 20

Below is a snapshot of the RAM from the simulator in Bascom this program.
Variables are stored in ram in the order in which they are declared in Bascom.

```
Dim Message1 As String * 20 (first 21 bytes in red below)
Dim Message2 As String * 20 (second 21 bytes in green below)
Dim Xposition As Byte (a single byte in dark red)
Dim Count As Byte (a single byte in dark green)
```



The data stored in the variable changes during the program , so after the first loop of the program the memory looks like this above.

Message1 has ‘hello’ stored in it. You can see that Bascom has actually allocated 21 bytes not 20 as we asked for when we configured the string; this is because Bascom puts a 0 on the end of each string in memory. The simulator conveniently displays any viewable ascii characters stored in ram on the right hand side of its window.

Message2 has ‘there’ stored in it, again 21 characters are used.

The next byte of ram has the number 5 stored in it, this is the position on the lcd that we want the text to appear at.
The next byte is the variable count it goes up from 1 to 3 to control the number of times the text flashes on the LCD.

You can look up the values in the ASCII table for the above RAM, these are hexadecimal numbers

hexadecimal	binary	Decimal	ASCII
68	&B 0100 1000	104	H
65	&B 0100 0101	101	E
6C	&B 0100 1100	108	L
6C	&B 0100 1100	108	L
6F	&B 0100 1111	111	O

24.1 Strings assignment

```
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output             'LEDs on portA
Config Portb = Output             'LEDs on portB
Config Portc = Output             'LEDs on portC
Config Portd = Output             'LEDs on portD
'config inputs
Config Pina.0 = Input             ' ldr
Config Pind.2 = Input             'switch A
Config Pind.3 = Input             'switch B
Config Pind.6 = Input             'switch C
Config Pinb.1 = Input             'switch D
Config Pinb.0 = Input             'switch E
'LCD

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

' 7. Hardware Aliases
Sw_a Alias Pinb.0
Sw_b Alias Pinb.1
Sw_c Alias Pind.2
Sw_d Alias Pind.3
Sw_e Alias Pind.6

' 8. initialise ports so hardware starts correctly
Porta = &B11111100                'turns off LEDs ignores ADC inputs
Portb = &B11111100                'turns off LEDs ignores switches
Portc = &B11111111                'turns off LEDs
Portd = &B10110011                'turns off LEDs ignores switches
Cls                        'clear lcd screen
Cursor On Noblink
'-----------------------------------------------------------------
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Mix As Byte
Dim Firstname As String * 12
Dim Middlename As String * 12
Dim Lastname As String * 12
Dim Fullname As String * 40
' 11. Initialise Variables
Mix = 0
Firstname = "Edgar"
Middlename = "Alan"
Lastname = "Poe"
Fullname = ""
```

```
'--------------------------------------------------------------------
' 12. Program starts here
Cls
Gosub Welcome
Do
  Debounce Sw_a , 0 , Welcome , Sub
  Debounce Sw_b , 0 , Mixup , Sub
Loop
End                              'end program

'--------------------------------------------------------------------
' 13. Subroutines
Welcome:
  Cls
  Lcd "Welcome"
  Lowerline
  Lcd Chr(126) : Lcd "to strings" : Lcd Chr(127)
Return

Mixup:
  Incr Mix
  If Mix =
  If Mix = 1 Then Fullname = Firstname + " " + Middlename + " " + Lastname
  If Mix = 2 Then Fullname = Middlename + " " + Lastname + " " + Firstname
  If Mix = 3 Then Fullname = Lastname + " " + Firstname + " " + Middlename
  If Mix = 4 Then Fullname = Mid(fullname , 10 , 5)
  If Mix = 5 Then Fullname = Lastname + "," + Left(firstname , 2)
  If Mix = 6 Then Fullname = Version(1)
  If Mix = 7 Then
  If Mix = 8 Then
  If Mix = 9 Then
  If Mix > 10 Then Mix = 0
  Cls
  Lcd Fullname
Return
```

From the help file find out how to use and then add to this program 3 of the following at 7,8,9

Instr Lcase Len Lookupstr Ltrim Left Right Rtrim Space Spc String Trim Ucase Mid

Use these to convert numbers to and from strings and display them

Format Fusing Hex Bin Hexval Str Val Split

24.2 ASCII Assignment

1. Copy the following code into BASCOM
2. Compare the datasheet for the LCD with the characters that actually appear on your LCD.
3. Write the code for the **decrementcode** subroutine

```
'------------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 1 June 2005
' File Name: LCDcharactersV1.bas
'------------------------------------------------------------------
' 2. Program Description:
' everytime btn is pressed the character on the lcd changes
' highlights the use of the ASCII code
' 3. Hardware Features:
' LEDS
' 5 switches
' LCD
' 4. Program Features
' do-loop to keep program going forever
' debounce to test switches
' if-then-endif to test variables
'------------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000 'the speed of the micro
$regfile = "m8535.dat" 'our micro, the ATMEGA8535-16PI
'------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
'config inputs
Config Pind.2 = Input 'switch A
Config Pind.3 = Input 'switch B
Config Pind.6 = Input 'switch C
Config Pinb.1 = Input 'switch D
Config Pinb.0 = Input 'switch E
'LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
' 7. Hardware Aliases
Sw_a Alias Pinb.0
Sw_b Alias Pinb.1
Sw_c Alias Pind.2
Sw_d Alias Pind.3
Sw_e Alias Pind.6

' 8. initialise ports so hardware starts correctly
Porta = &B11111100 'turns off LEDs ignores ADC inputs
Portb = &B11111100 'turns off LEDs ignores switches
```

```
Portc = &B11111111 'turns off LEDs
Portd = &B10110011 'turns off LEDs ignores switches
Cls 'clear lcd screen
'-----------------------------------------------------------------
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Code As Byte
Dim State As Byte
' 11. Initialise Variables
Code = 0
State = 0
'-----------------------------------------------------------------
' 12. Program starts here
Do
        Debounce Sw_a , 0 , Swa_press , Sub
        Debounce Sw_b , 0 , Swb_press , Sub
        If State = 0 Then Gosub Intro
        If State = 1 Then Gosub Increasecode
        If State = 2 Then Gosub Decreasecode
        If State = 4 Then Gosub Waiting
Loop
End 'end program

'-----------------------------------------------------------------
' 13. Subroutines
Intro:
        Lcd "ASCII codes"
        Lowerline
        Lcd "btn A incrs code"
Return

Waiting:
'       do nothing
Return

Increasecode:
        If Code < 255 Then 'max value is 255
                Incr Code
        Else
                Code = 0 'if > 255 reset to 0
        End If
        Cls
        Lcd Code : Lcd " " : Lcd Chr(code)
        State = 4
Return
```

```
Decreasecode:
'write your code here

Return

Swa_press:
   State = 1
Return

Swb_press:
   State = 2
Return
```

24.3 Time in a string

Previously we wrote a small program that created a very simple clock. To display the time we put the time on the screen as hours, minutes and seconds e.g. 10:07:01
We could create a string to hold the time and display it using **Lcd** Timestr

```
$sim

'------------------------------------------------------------------
' Title Block
' Author:   B.Collis
' Date:     14 Aug 2003
' File Name: simple clock v1.bas
'------------------------------------------------------------------
' Program Description:
' use an LCD to display
' Program Features:
' outer do-loop
' Hardware Features:
' LCD on portc - note the use of 4 bit mode and only 2 control
lines
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our
hardware)
$crystal = 8000000                       'the crystal we are using
$regfile = "attiny26.dat"                'the micro we are using
'------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                    'LEDs on portA
Config Portb = Output                    'LEDs on portB
Config Lcdpin = Pin , Db4 = Portb.2 , Db5 = Portb.3 , Db6 =
Portb.4 , Db7 = Portb.5 , E = Portb.1 , Rs = Portb.0
Config Lcd = 20 * 2                      'configure lcd screen

' Harware Aliases
' initialise hardware
Cls                                      'clears LCD display
Cursor Off                               'no cursor
'------------------------------------------------------------------
' Declare Constants
Const Timedelay = 350
'------------------------------------------------------------------
' Declare Variables
Dim Seconds As Byte
Dim Minutes As Byte
Dim Hours As Byte
Dim Day As Byte
Dim Month As Byte
Dim Year As Byte

Dim Timestr As String * 8

'  Initialise Variables
```

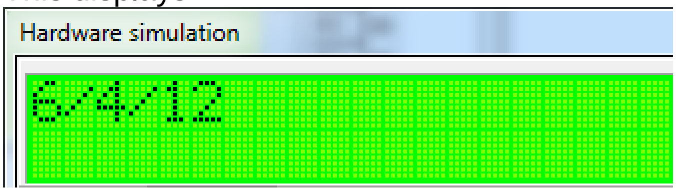
```
Seconds = 50
Minutes = 5
Hours = 14                                  '2pm
Day = 21
Month = 4                                   'april
Year = 10                                   '2010
'-----------------------------------------------------------------
' Program starts here
Do
   Wait 1
   Incr Seconds

   If Seconds > 59 Then
      Seconds = 0
      Incr Minutes
   End If

   Gosub Maketime                           'make a string of the time
   Locate 1 , 5
   Lcd Timestr                              'display the string

Loop
End                                         'end program
'-----------------------------------------------------------------
Maketime:
      Timestr = ""                          'delete the string
       'rebuild the string
      If Hours < 10 Then Timestr = Timestr + "0"
      Timestr = Timestr + Str(hours)
      Timestr = Timestr + ":"
      If Minutes < 10 Then Timestr = Timestr + "0"
      Timestr = Timestr + Str(minutes)
      Timestr = Timestr + ":"
      If Seconds < 10 Then Timestr = Timestr + "0"
      Timestr = Timestr + Str(seconds)
Return
```

24.4 Date in a string

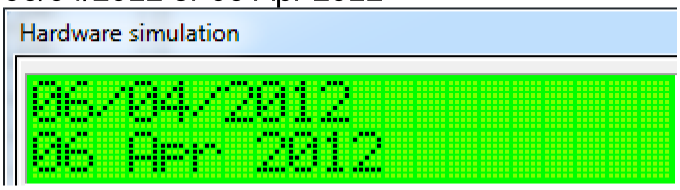
Here is a program segment to display the date in a string

```
'-----------------------------------------------
'Declare Variables
Dim Day As Byte                        'e.g. 6
Dim Month As Byte                      'e.g. 4
Dim Month_str As String * 3            'e.g. apr
Dim Year As Byte                       'e.g. 12 means 2012
Dim Year_str As String * 4             'e.g. "2012"
Dim Today As String * 20               'a variable to store some
text

 'Initialise Variable
Day = 6
Month = 4
Year = 12
'----------------------------------------------
'Program starts here
Do
   Gosub Makedate
   Locate 1 , 1
   Lcd Today
Loop
End

Makedate1:
   'str is a function to convert a number to a string
   Today = Str(day) + "/" + Str(month) + "/" + Str(year)
Return
```

This displays



Which is not what we want we want to be able to display it in either of these formats 06/04/2012 or 06 Apr 2012



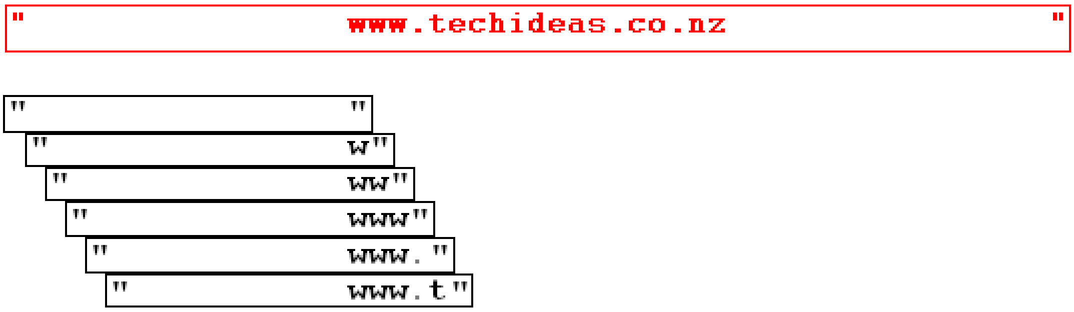
On the next page you will see the code for this it needs completing

```
'-----------------------------------------------
'Declare Variables
Dim Day As Byte                          'e.g. 20
Dim Month As Byte                        'e.g. 4
Dim Month_str As String * 3              'e.g. apr
Dim Year As Byte                         'e.g. 12 means 2012
Dim Year_str As String * 4               'e.g. "2012"
Dim Today As String * 20                 'a variable to store some
text

 'Initialise Variable
Day = 6
Month = 4
Year = 12
'-----------------------------------------------
'Program starts here
Do
   Gosub Makedate1
   Locate 1 , 1
   Lcd Today

   Gosub Makedate2
   Locate 2 , 1
   Lcd Today
Loop
End

Makedate1:
   Today = ""
   If Day < 10 Then Today = "0"
   Today = Today + Str(day) + "/"
   'you complete the rest of this routine
Return

Makedate2:
   'str is a function to convert a number to a string
    Today = ""
    If Day < 10 Then Today = "0"
    Today = Today + Str(day) + " "
    Select Case Month
      Case 1 : Month_str = "Jan"
      Case 2 : Month_str = "Feb"

    'you complete the rest of this routine


Return
```

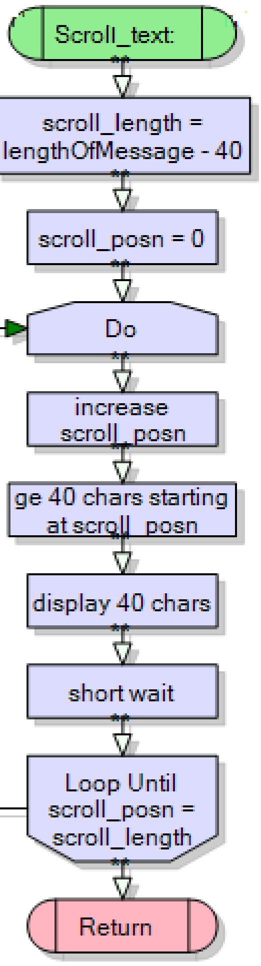
24.5 Scrolling message assignment

An alphanumeric (text) LCD is a very common output device used with microcontrollers however they have limited screen size so a longer message must be either split up and shown page by page or scrolled across the screen.

If the string was 50 charcters long as with the one below and the LCD was 16 characters wide then using the mid command we could take the first 16 characters and put them on the display then wait a bit, then get the next 16 characters and put them on the display, and so on continuously.



In this assignment you will scroll a message across the screen. The message will be an information message regarding a news item or weather forecast up to 200 characters in length.




```
'Declare Variables
Dim message as string * 200
Dim scroll_length as byte
Dim scroll_posn as byte
Dim forty_chars as string * 40

'Initialise Variables
Message = " the weather today will be ….."

Scroll_text:
   Scroll_length = len(message)
   If Scroll_length > 40 then
      Scroll_length = scroll_length – 40
   End if
   Scroll_posn = 0
   While scroll_posn < scroll_length
      Incr scroll_posn
      Forty_chars =mid(message,scroll_posn,40)
      Locate 1,1
      Lcd forty_chars
      Waitms 150
   Wend
Return
```

This routine scrolls the complete message once then returns to the main loop. This makes it a very long routine to execute (150mS times the number of characters in the string)

Change the code so that it uses: a Do-Loop-Until structure and then a For-Next

24.6 Some LCD programming exercises.

These exercises will require you to manipulate the display, manipulate text, manipulate numbers. And become familiar with the use of loops to get things done.
You need to save each version of the program separately e.g wassup_b.bas, wassup_p.bas, wassup_a.bas.

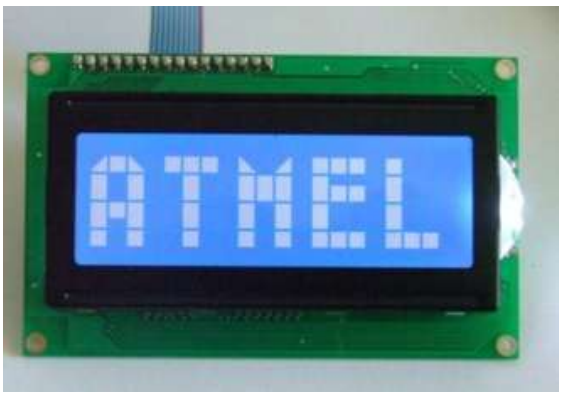
Basic: put ‘wassup’ on the display
Proficient: Have ‘wassup’ scroll around the screen continuously
Advanced: Have the 6 letters of ‘wassup’ appear spread out over the display and then after a brief delay move in towards the centre and in order.

Basic: calculate 2^8 and display it
Proficient: for n from 1 to 25, display 2^n on the screen, wait for 1 sec and then do the next number
Advanced: Write you own code to calculate the square root of the answer for each of the above answers

Basic: Display a static weather report for Auckland on the LCD screen
Proficient: Do graphics for sunny, cloudy, wet, and snowy for your weather report, that flash on the screen, these graphics should be larger than a single lcd square, perhaps 2/3 lines x 4squares
Advanced: Scroll the message on and off the display and have the graphics flash for a while, then the weather report scrolls back on again.

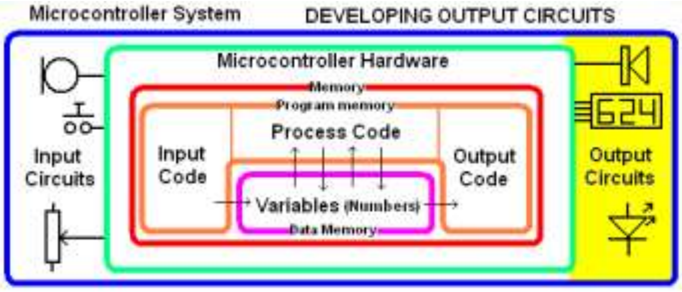
Basic: Display 2 random numbers between 2,000 and 99,000
Proficient: repeat this process continuously, and also subtract the smaller from the larger number and display the answer, have a 3 second delay between each new calculation
Advanced: Scroll the results off the display 0.5 seconds after the calculation

Basic: Create 4 different pacman graphics: one pacman mouth open, one pacman mouth closed, one a target and the last the target exploding
Proficient: Have the pacman move around the screen these, staying on each square for only 0.5 seconds.
Advanced: Generate a random location on the LCD and place the target there, have the pacman move around the screen and when it lands on the target the target explodes and the pacman moves on around the rest of the screen



Proficient: create ‘12TCE’ in one large font that covers all four lines of the lcd like the wording of atmel in this picture
Proficient: flash the message on the screen three times, 1 second on then 1 second off after that have it stay on for 12 seconds then repeat the 3 flashes.

25 Advanced power interfaces



So far we have looked at lower power output interfaces for the microcontroller such as LEDs and LCDs the problem though is that we will want to add high power things to our designs so we must know what to use and how to use it. The learning for this best takes place in some order, here is what I have chosen:

1. know what we can do and what we cannot do with a microcontroller output port.
2. know about power
3. know some more detail about how certain semiconductors are used and work
4. know about the output devices and their power requirements
5. know about the extra features the AVR has to help us drive those devices

25.1 Microcontroller power limitations

The microcontroller specifications we are interested in are found in the electrical characteristics section of the datasheet for the microcontroller, here are the specs for an ATTiny461.

23. Electrical Characteristics

23.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{RESET}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{RESET}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA
Injection Current at VCC=0V	±5.0 mA[1]
Injection Current at VCC=5V	±1.0 mA

Notes: 1. Maximum current per port = ±30mA

We are initially interested in the DC current specification 40mA per I/O pin –that sounds great 40mA is heaps for a pin we could do lots with that.

BUT wait – the next line says 200mA for the power pins so we cannot draw 40mA from all 15 pins because that would exceed the 200mA for the power pins by 400mA (15 x 40 = 600mA)

There is more data we need to know about.

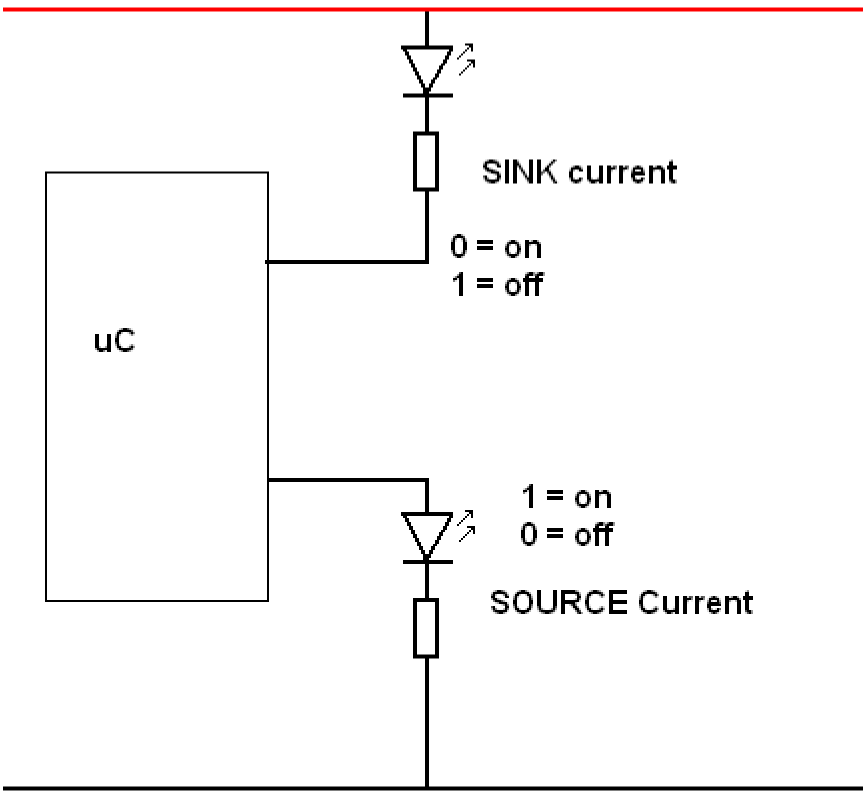
23.2 DC Characteristics

T_A = -40°C to 125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)[1]

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{OL}	Output Low Voltage[4] (Except Reset pin)	I_{OL} = 10 mA, V_{CC} = 5V I_{OL} = 5 mA, V_{CC} = 3V			0.6 0.5	V V
V_{OH}	Output High-voltage[5] (Except Reset pin)	I_{OH} = -10 mA, V_{CC} = 5V I_{OH} = -5 mA, V_{CC} = 3V	4.3 2.5			V V

4. Although each I/O port can sink more than the test conditions (10 mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOL, for all ports, should not exceed 60 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
5. Although each I/O port can source more than the test conditions (10 mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOH, for all ports, should not exceed 60 mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

Two terms sink and source are used here we first need to understand these specifications.



The names sink and source describe which way the current is going in a circuit, either to positive or ground. They are with repect to conventional current (not electron current).

It was common for microcontrollers to have different sink and soure characteristics but nowadays it seems more common to see the sink and source ratings for a microcontroller are the same (but not always).

The really important characteristic from the datasheet is in notes 3 and 4 where it states that the sum of all currents for all ports should not exceed 60mA sink and 60mA source. So if we wanted to use all 15 pins of the ATtiny as outputs and switch them all on at the same time then we cannot sink more than 4mA current from each pin (60mA/15pins)! So be warned!!

In the first example we will use the microcontroller to switch a backlight for an LCD on and off.

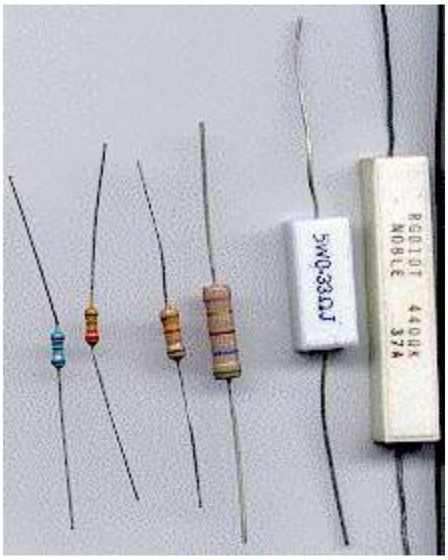
25.2 Power

So far the concepts of voltage and current have been introduced however when these are present a third important aspect of circuits is present as well, that is power.

Any device that has a voltage across it and current is flowing uses energy, and therefore dissipates this energy in the form of heat.

Components don't like to get too hot and are rated to work only below a certain temperature. The more energy the hotter a component gets and the more likely it is to overheat and be destroyed

25.3 Power dissipation in resistors



Power = voltage times current, P=V*I , Power is measured in Watts. 2V across a 10ohm resistor. I=V/R, I= 2/10, I=0.2A, so P=V*I, P=2*0.2, P= 0.4W.

Resistors come in different power ratings so it is important in a circuit to understand that the power ratings should not be exceeded or the component may overheat, become burnt and have its life shortened or be destroyed.

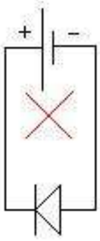
Resistors can be bought in various ratings, on the left are 1/8, 1/4, 1/2, 1, 5 & 10 Watts.
On the right 5 and 10 watt metal cased ones
Note that the physical size grows proportionally with the rating

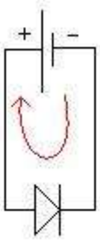
V = 10V I = 2A	P = V * I, P = 10*2, P=20W
V = 5V I = 0.3A	
I = 200mA V = 12V	
V = 100V I = 3mA	
V = 100V P = 50W	
V = 48V I = 20mA	
A 5 Watt bulb draws 1.6A what is the voltage?	
A 12 battery supplies 20mA to a resistor, what is the power?	
What wattage resistor would you use for 15V and 0.2A	
What wattage resistor would you use for 36V and 100mA	

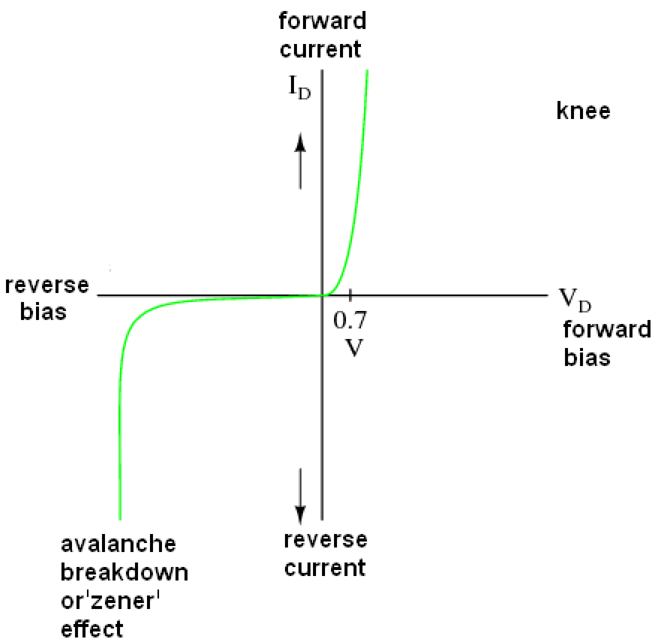
25.4 Diode characteristics

When a voltage is applied to the diode in a forward direction it is called forward bias; as this increases there is little current until the voltage reaches 0.65 to 0.7V and the diode will conduct fully.

When voltage is applied in a reverse direction it is called reverse biased and as the voltage is increased a point will be reached where the voltage is greater than the diode can handle the diode will suddently conduct. In a normal diode exceeding the reverse voltage specification will generally destroy the diode.





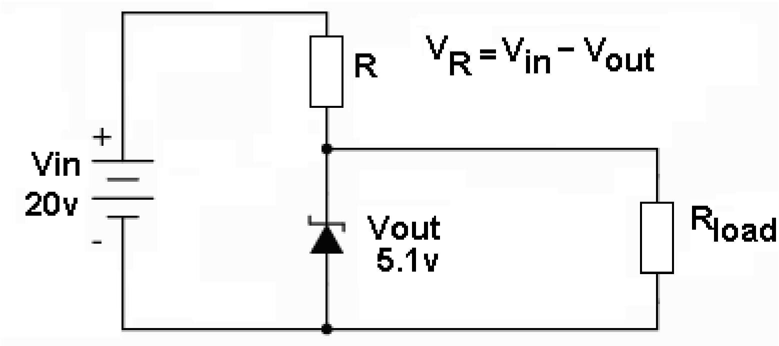


This graph decribes the characteristic of diode conduction in a visual form. When the diode is forward biased above 0.65 the diode conducts, when it is reversed biased it will not conduct until it safe operating voltage is exceeded. At reverse voltages higher than that it will probably be detroyed.

25.5 Using Zener diodes

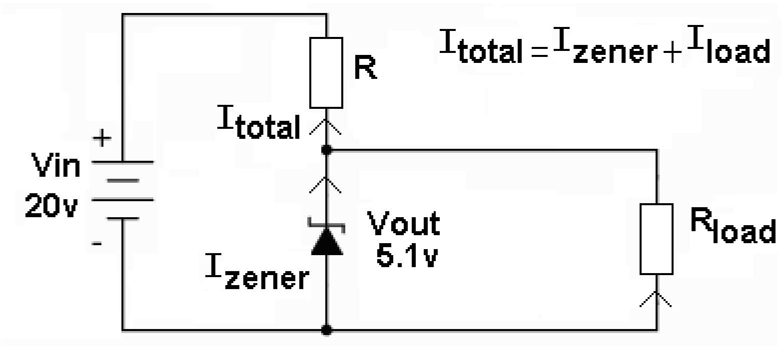
The reverse conduction effect can be put to use in controlled circumsnaces and in Zener diodes this effect is used to make small regulated power supplies.

Note the symbol for a zener is different to a normal diode and shows the knee and avalanche effects in the symbol with the angled line at the cathode end.



If we want to make a small power supply for a common circuit (5V) and we find a 20V dc power pack we can use a zener diode.

The first calculation is simply the voltage across the Resistor $V_R = V_{in} - V_{out}$



We must know what load the rest of the circuit presents to the power supply. We don't need to draw the rest of the circuit to help us we can represent it as a resistor R_{Load} e.g. a small microcontroller circuit might draw 150mA (0.15A).

The current though the load will be 150mA, a zener requires some small current to work e.g. 5mA, so the total current will be 150mA + 5mA – 155mA (0.155A).
Using Ohms law the value of R will be V/I = (20-5.1)/0.155 = 96ohms.

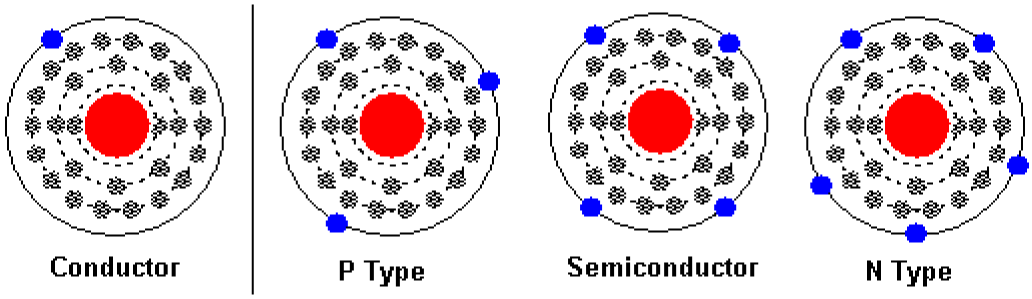
The issue however with zener circuits is not so much the voltage and resistance calculations it's the power calculations.

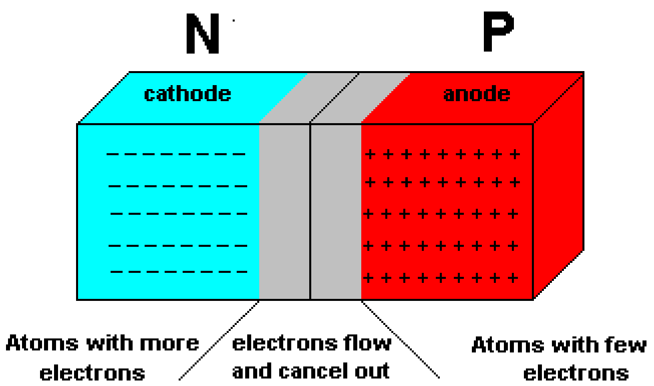
We assume worst case so the power the resistor has to dissipate is V x I = (20-5.1) x 0.155 = 2.3W so we would use a 5W resistor, not a usual 400mW one we would find in the workshop!
For a zener diode, power is also factor and worst case will be when the load draws no current.
Power = V x I = 5.1 x 0.155 = 0.79W so a 1W zener would be used (not a usual 400mW one).

V_{in}	V_{out}	$V_R = V_{in} - V_{out}$	I_{Load}	I_{Zener}	I_{total}	$R = V_R/I_{total}$	$P_R = V_R \times I_{total}$	$P_{Zener} = V_{out} \times I_{zener}$
20	5.1	14.9	0.15	0.005	0.155	96	2.3W	0.79W
12	5.1		0.08	0.005				
24	5.1							

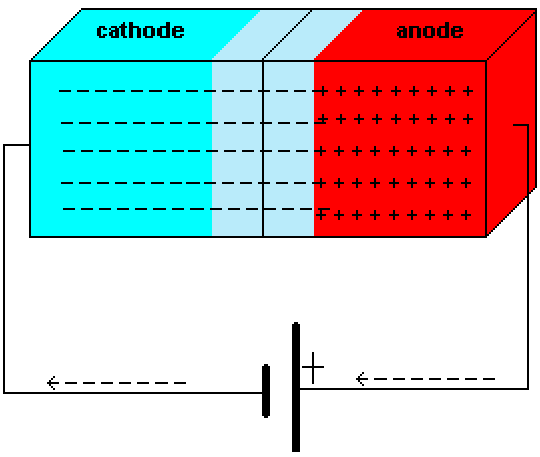
25.6 How diodes work

A diode is made from <u>silicon (a semiconductor).</u> Semiconductors have more electrons in their outer shells than conductors. To the silicon other materials (<u>impurities</u>) are added, these other materials have either more or less electrons in the outer shell. A diode is made from a piece of silicon which is doped with both N-type and P-type impurities. Knowing how a normal diode works will help you understand the basics of how an LED gives off light.





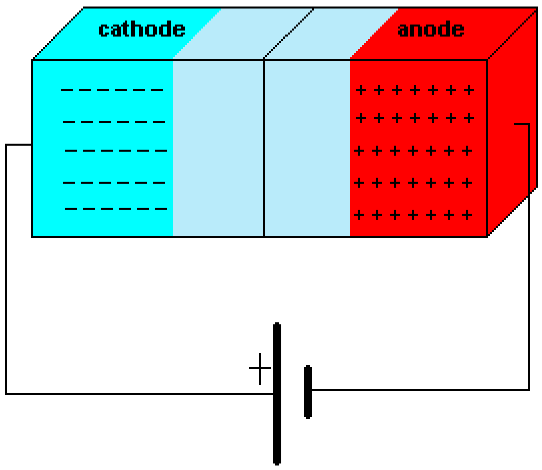
One part of the silicon has N-type impurities added (slightly more conductive), in the other part P-type impurities are added (slightly less conductive).



When no voltage is connecte some (the

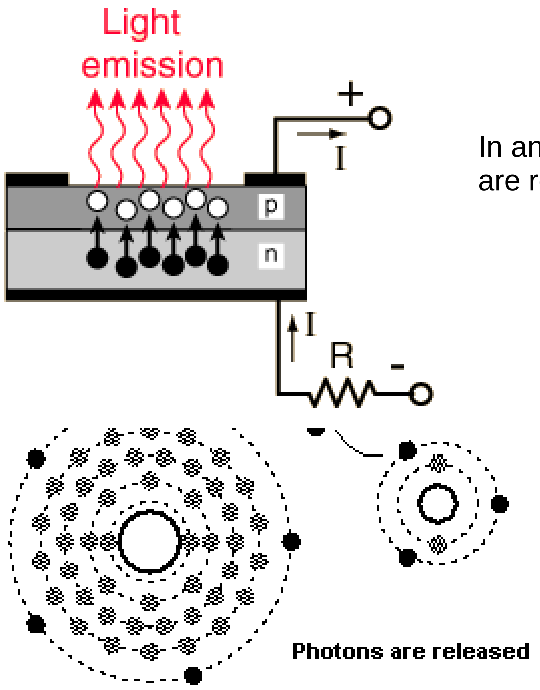
d to the diode there is a region in the middle where electrons flow over and the effect is cancelled out depletion region).

<u>When a large enough voltage is applied to a diode</u> <u>(about</u> <u>0.4v to 0.6V) electrons will flow from the negative to</u> <u>the</u> <u>positive</u>. This is called forward bias. In the process the depletion region disappears.



When the battery is connected back to front the diode is "reverse biased" and the depletion region in the middle gets larger, so electrons cannot flow. This explains why **<u>diodes conduct only when connected into a circuit the right way around</u>.**

25.7 How does a LED give off light?



In an LED when electrons move from the N side to the P side photons are released.

Photons are released whenever electrons move from one shell level in an atom to another. In an LED the electrons move from the N to the P and also change levels within the atomic structure at the same time, therefore releasing photons.

Note that the voltage required for an LED to conduct is much greater than a normal diode. Typical values range from 1.8V to 3.6V, and like an ordinary diode they only work in one direction

LED Colours

In an LED different colours are achieved by using different types of impurities.

Light Emitting Diode Colour Variations

Color Name	Wavelength (Nanometers)	Semiconductor Composition
Infrared	880	GaAlAs/GaAs
Ultra Red	660	GaAlAs/GaAlAs
Super Red	633	AlGaInP
Super Orange	612	AlGaInP
Orange	605	GaAsP/GaP
Yellow	585	GaAsP/GaP
Incandescent White	4500K (CT)	InGaN/SiC
Pale White	6500K (CT)	InGaN/SiC
Cool White	8000K (CT)	InGaN/SiC
Pure Green	555	GaP/GaP
Super Blue	470	GaN/SiC
Blue Violet	430	GaN/SiC
Ultraviolet	395	InGaN/SiC

SURE electronics

DE-LM005

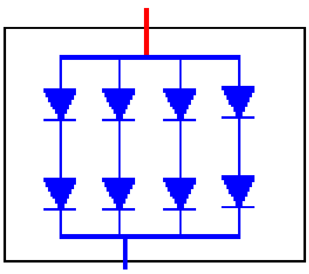
■ Electrical Characteristics

ITEM	Symbol	Condition	Standard value			UNIT
			Min.	Typ.	Max.	
Supply Voltage For Logic	V_{DD}-V_{SS}	--	4.5	5.0	5.5	V
Supply Voltage For LCD	V_{DD}-V_0	--	--	4.7	--	V
Input High Voltage	V_{IH}	--	2.2	--	V_{DD}	V
Input Low Voltage	V_{IL}	--	-0.3	--	0.6	V
output High Voltage	V_{OH}	$-I_{OH}$=0.2mA	2.4	--	--	V
output Low Voltage	V_{OL}	I_{OL}=1.2mA	--	--	0.4	V
Power Supply Current	V_{DD}	V_{DD}=5.0V	--	2.0	5	mA
With B/L	I_{LED}	V_{LED}=5.0V	--	72	80	mA

In the datsheet for a 4 line LCD, the LCD typically draws 2mA with the backlight off and 72mA with it on, so the backlight requires 70mA, it also requires 4.7V.

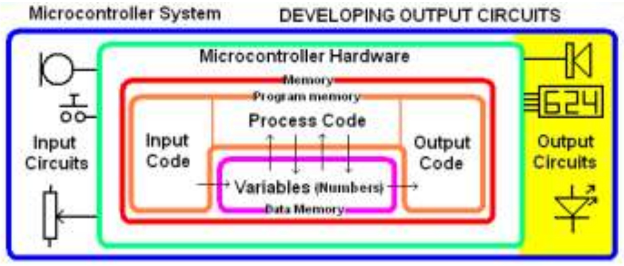
Although we don't have a schematic for the backlight we can make a good guess at what the circuit for it might look like. A typical LED requires 2V to 2.5V to drive it, so if the backlight LEDs require 4.7V we can safely assume that there are 2 LEDs in series. As the backlight LEDs draw 70mA in total and a typical LED is up to 20mA we could guess at either 3 or 4 sets of LEDs in parallel.

LED Backlight



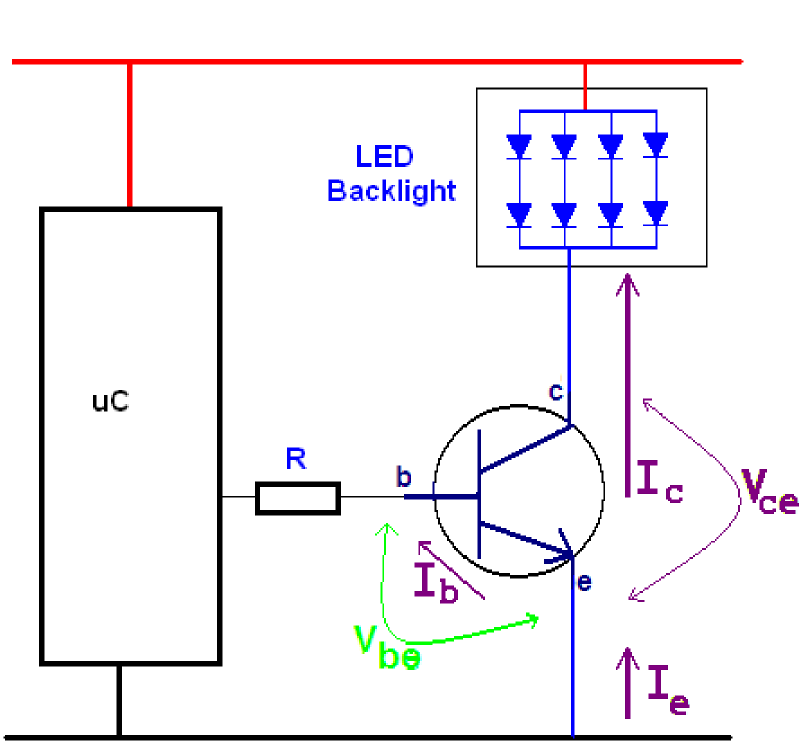
As the backlight LEDs draw 70mA it is not possible to drive them directly from a microcontroller I/O pin, we need another control component in between.

25.9 Transistors as power switches



There are many different types of transistor and the BJT has already been introduced so we will investigate it as an intermediate stage of switching between the microcontroller and the backlight.

	BC547	
Type	NPN	BJT type
Case	T092	
I_C (mA)	100 mA	The maximum current that we can control
$V_{ce\ MAX}$	45 V	The maximum voltage we can apply to the circuit
h_{FE} (gain)	110-800	The amplification factor I_c/I_b
P_{TOT} (power)	500 mW	The maximum power that can be dissipated by the device



What we know:
The backlight is a bunch of LEDs requiring 4.7V and 70mA.

You need to know:
A transistor when it is completely switched on will have a Vbe of 0.7V and a Vce of 0.3V

The current to the LED backlight comes from the transistor and is the same as I_c, (collector current) We want this to be 70mA.

To get and I_c of 70mA we need some current through the base I_b.
The relationship between Colector and base current is called gain or h_{FE}. Gain or $h_{FE} = I_c / I_b$

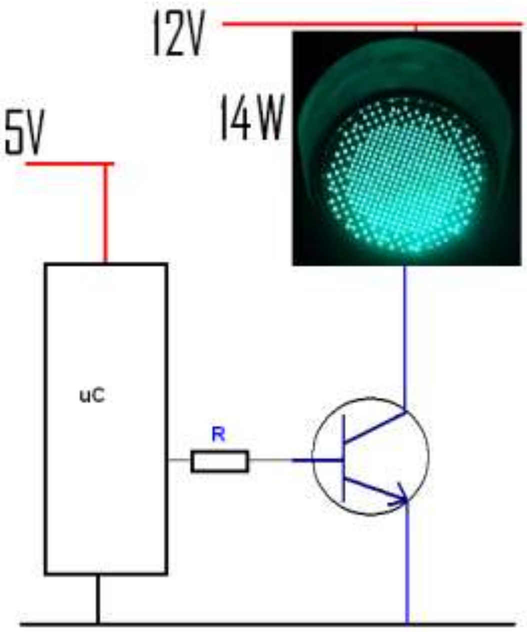
$I_b = I_C / h_{FE} = 70/110 = 0.6$mA

The current in the base is the same as the current in the the resistor R from the microcontroller.
Using ohms law R = V/I = (5-0.7) / 0.0006 = 7k166 ohms
A suitable value of R would be lower than 7K to make sure that at least 0.0006A flows. So we would choose a convenient 4k7. In fact it would be fine to go lower or a bit higher.

Now the hidden calculation is power, the transistor has a voltage of V_{CE} across the emitter and collector. This will always be about 0.3V for a BJT transistor when it is fully switched on.
Power = V x I = 0.3 x 70mA = 0.3 x 0.07 = 0.021W = 21mW.
Looking at the specifications in the above table the BC547 can dissipate 500mW and we want it to dissipate 21mW, so it should work fine.

This fine for a 70mA, 4.7V backlight but more powerful devices will require bigger transistors. The problem with bigger transistors however is that you have to drive them with a lot of current from the microcontroller which cannot provide a lot of current!! So…

25.10 High power loads



When we have a load that requires higher power we may need a higher voltage supply and more current.

Here is an LED based traffic light, it has 168 LEDs and requires a 12V supply voltage.

1	2	3	4	5	6	7	8
Load	I_c	h_{FE}	I_b	$\mathbf{V_{be}}$	R	$\mathbf{V_{ce}}$	P_{tot}
Green 300mm traffic light 12V 14W (168 LEDs)	I = P / V = 14/12 = 1.16A	BC547 = 110	$I_b = I_c / h_{FE}$ =1.16/110 =0.011A **= 11mA**				

Now 11mA from a microcontroller sounds ok but lets review the datasheet for the AVR.

25.11 AVR Power matters

23. Electrical Characteristics

23.1 Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on any Pin except $\overline{\text{RESET}}$ with respect to Ground	-0.5V to V_{CC}+0.5V
Voltage on $\overline{\text{RESET}}$ with respect to Ground	-0.5V to +13.0V
Maximum Operating Voltage	6.0V
DC Current per I/O Pin	40.0 mA
DC Current V_{CC} and GND Pins	200.0 mA
Injection Current at VCC=0V	±5.0 mA[1]
Injection Current at VCC=5V	±1.0 mA

Notes: 1. Maximum current per port = ±30mA

The datasheet might initially lead you to believe that we can draw 40mA from an I/O pin. However there is an absolute maximum rating of 200mA from the power supply pins, so if we were to draw 40mA from 5 I/O pins then we would have reached the maimum for our device.

But theres more…

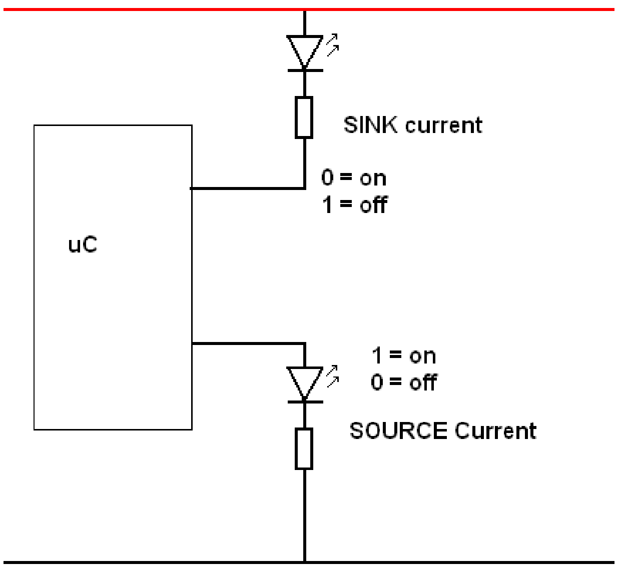
23.2 DC Characteristics

T_A = -40°C to 125°C, V_{CC} = 2.7V to 5.5V (unless otherwise noted)[1]

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
V_{OL}	Output Low Voltage[4] (Except Reset pin)	I_{OL} = 10 mA, V_{CC} = 5V I_{OL} = 5 mA, V_{CC} = 3V			0.6 0.5	V V
V_{OH}	Output High-voltage[5] (Except Reset pin)	I_{OH} = -10 mA, V_{CC} = 5V I_{OH} = -5 mA, V_{CC} = 3V	4.3 2.5			V V

4. Although each I/O port can sink more than the test conditions (10 mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOL, for all ports, should not exceed 60 mA.
   If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test condition.
5. Although each I/O port can source more than the test conditions (10 mA at V_{CC} = 5V, 5 mA at V_{CC} = 3V) under steady state conditions (non-transient), the following must be observed:
   1] The sum of all IOH, for all ports, should not exceed 60 mA.
   If IOH exceeds the test condition, VOH may exceed the related specification. Pins are not guaranteed to source current greater than the listed test condition.

In note 4 and 5 above from the datasheet there is a maximum rating of sinking 60mA and sourcing 60mA total from all I/O ports. This is in effect 120mA in total.



Sink current is when the current is from ther AVR I/O pin to Vcc (a low or 0 turns on the load)

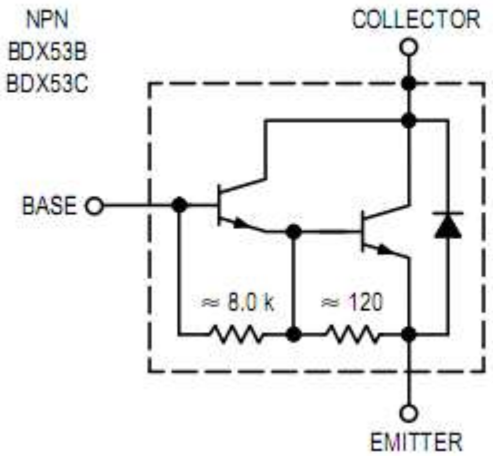
Source current us when the current is from ground to the AVR I/O pin (a high or 1 turns on the load)

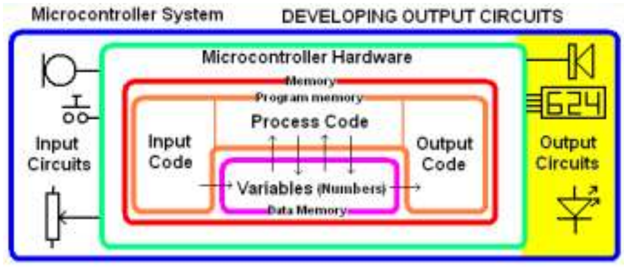
So there are significant limits to what we can drive from our AVR. This is why the current has been limited to a few mA by a 1K resistor with the all the multiple LED circuits so far, so that we do cannot stress the AVR.

So back to our LED traffic light, we could drive a few of them from our AVR but not many. It would be better to use an alternative.

25.12 Darlington transistors - high power

A darlington transistor is two transistors inside one package like this BDX53C





This device has a gain of at least 750 so to get the maximum current of 6A out of it will require only 6/750 = 0.008A = 8mA into the base.

MAXIMUM RATINGS

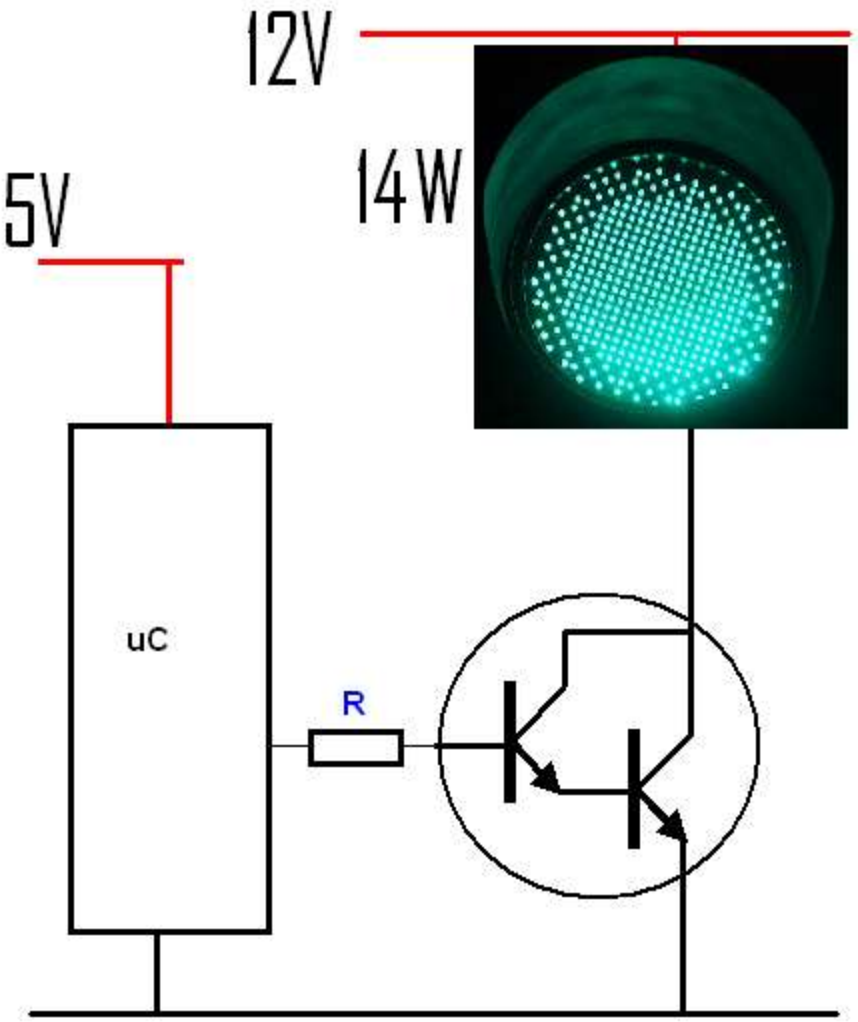
Rating	Symbol	BDX53B BDX54B	BDX53C BDX54C	Unit
Collector–Emitter Voltage	V_{CEO}	80	100	Vdc
Collector–Base Voltage	V_{CB}	80	100	Vdc
Emitter–Base Voltage	V_{EB}	5.0		Vdc
Collector Current — Continuous Peak	I_C	8.0 12		Adc
Base Current	I_B	0.2		Adc
Total Device Dissipation @ T_C = 25°C Derate above 25°C	P_D	60 0.48		Watts W/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}	–65 to +150		°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction to Ambient	$R_{\theta JA}$	70	°C/W
Thermal Resistance, Junction to Case	$R_{\theta JC}$	70	°C/W

ON CHARACTERISTICS (1)

DC Current Gain (I_C = 3.0 Adc, V_{CE} = 3.0 Vdc)	h_{FE}	750	—	—
Collector–Emitter Saturation Voltage (I_C = 3.0 Adc, I_B = 12 mAdc)	$V_{CE(sat)}$	— —	2.0 4.0	Vdc
Base–Emitter Saturation Voltage (I_C = 3.0 Adc, I_C = 12 mA)	$V_{BE(sat)}$	—	2.5	Vdc



The BJT NPN transistor has been replaced by an NPN Darlington transistor.

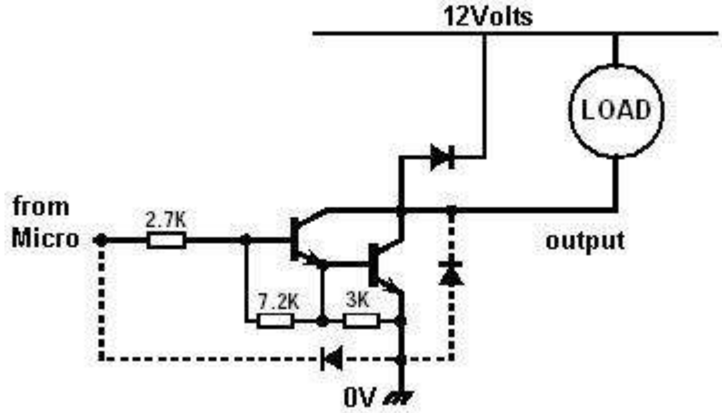
1	2	3	4	5	6	7	8
Load	I_c	h_{FE}	I_b	**V_{be}**	R	**V_{ce}**	P_{tot}
Green 300mm traffic light 12V 14W (168 LEDs)	I = P / V = 14/12 = 1.16A	BDX53C h_{FE}=750	$I_b = I_c / h_{FE}$ =1.16/750 =0.0015A **= 1.5mA**	**2.5V**	$=V_R / I_b$ = (5-2.5) / 0.0015 = 1,667 ohms Use 1k5	**2**	$= V_{ce} \times I_c$ = 2 x 1.16 = 2.32W

The BDX53C can dissipate 60W power, however it will heat up at the rate of 70 degrees per watt that it dissipates.

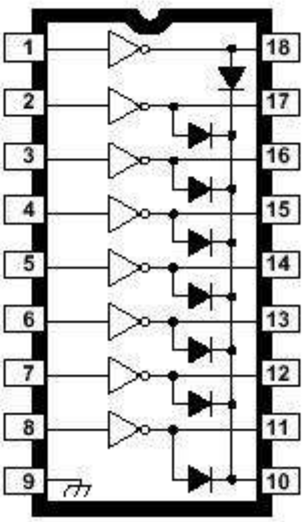
The BDX53 will then heat up by 2.32 x 70 = 162.4 degrees over and above ambient temperature. Ambient temperature is the temperature of the piece of equipment and is influenced by the air temperature other components that generate heat. This exceeds the temperature range of the device which is 150 degrees. So we should use a heat sink.

25.13 ULN2803 Octal Darlington Driver

This really useful IC has 8 darlington transistors built into it. Which makes it really useful for connecting to the 8 pins of one port on a microcontroller.



In this circuit for one I/O of the ULN2803 you can see the protection diodes on both the input and the outputs.



The protection diodes go to pin 18 which must be connected to the power for loads you are driving.

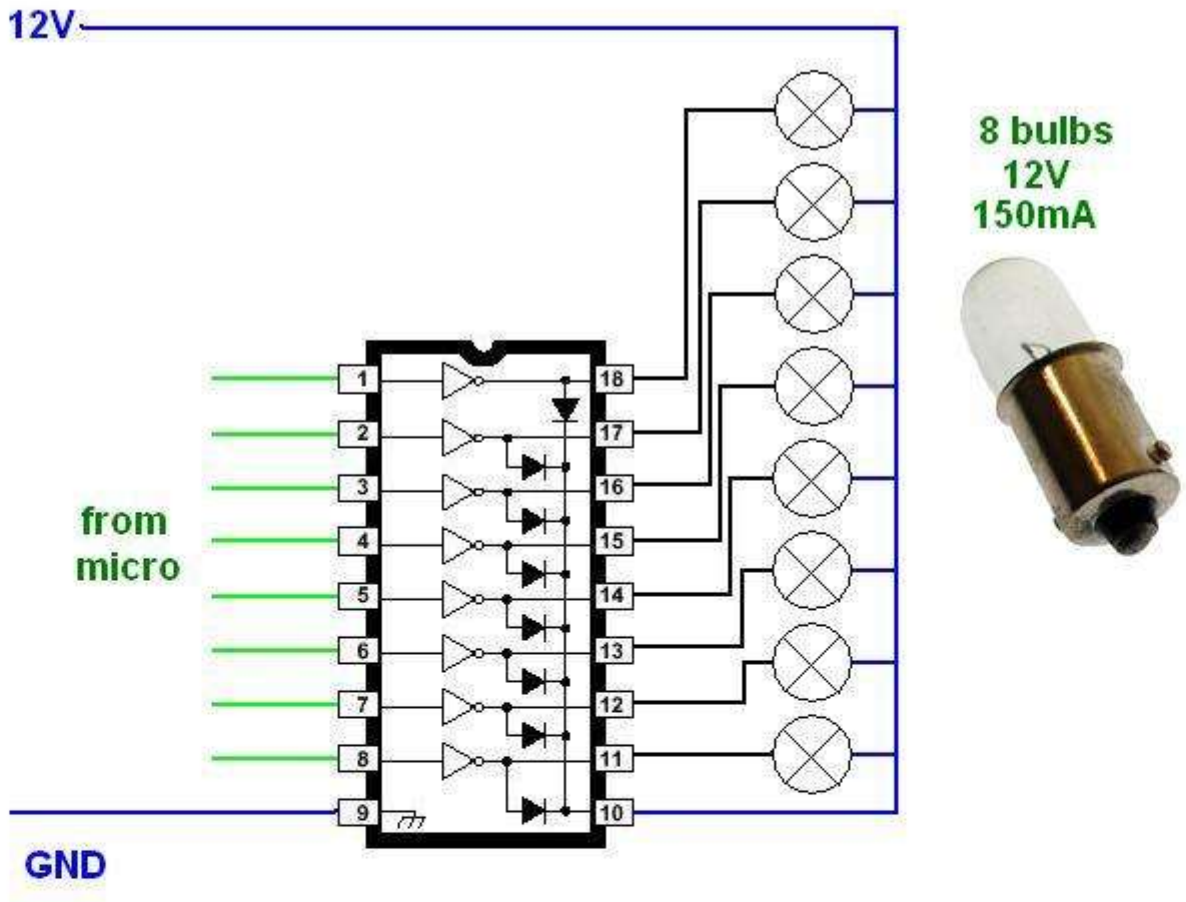
This device is great for connecting high power loads such as relays, solenoids, light bulbs
Each transistor can switch 500mA each however you cannot have more than 1W per output and a total of 2.25W per IC (all 8 outputs) at once.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_o	Output Voltage	50	V
V_i	Input Voltage for ULN2802A, UL2803A, ULN2804A for ULN2805A	 30 15	V
I_C	Continuous Collector Current	500	mA
I_B	Continuous Base Current	25	mA
P_{tot}	Power Dissipation (one Darlington pair) (total package)	 1.0 2.25	W
T_{amb}	Operating Ambient Temperature Range	– 20 to 85	°C
T_{stg}	Storage Temperature Range	– 55 to 150	°C
T_j	Junction Temperature Range	– 20 to 150	°C

ELECTRICAL CHARACTERISTICS (T_{amb} = 25°C unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit	Fig.
I_{CEX}	Output Leakage Current	V_{CE} = 50V T_{amb} = 70°C, V_{CE} = 50V T_{amb} = 70°C for ULN2802A V_{CE} = 50V, V_i = 6V for ULN2804A V_{CE} = 50V, V_i = 1V			50 100 500 500	µA µA µA µA	1a 1a 1b 1b
$V_{CE(sat)}$	Collector-emitter Saturation Voltage	I_C = 100mA, I_B = 250µA I_C = 200mA, I_B = 350µA I_C = 350mA, I_B = 500µA		0.9 1.1 1.3	1.1 1.3 1.6	V V V	2



In this example we want to drive 8 bulbs, bulbs are not socommon but theywillserve as an example of power calculations.

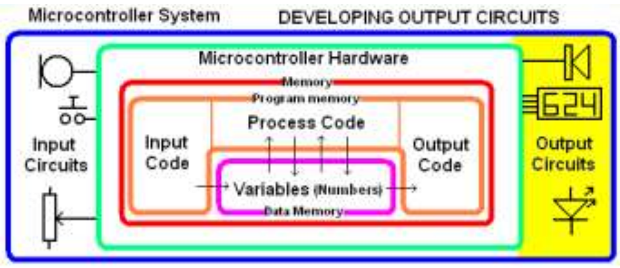
1. Power for each transistor
The transistor will have to supply 0.15A, when it is turned on (saturated)
The voltage across the Collector to Emitter will be 1.1V (worst case)
So the power for each transistor will be P = V x I = 1.1 x 0.15 = 0.165W

2. this measn if we want all 8 bulbs on at once we will have
P = 8 x 0.165 = 1.32W

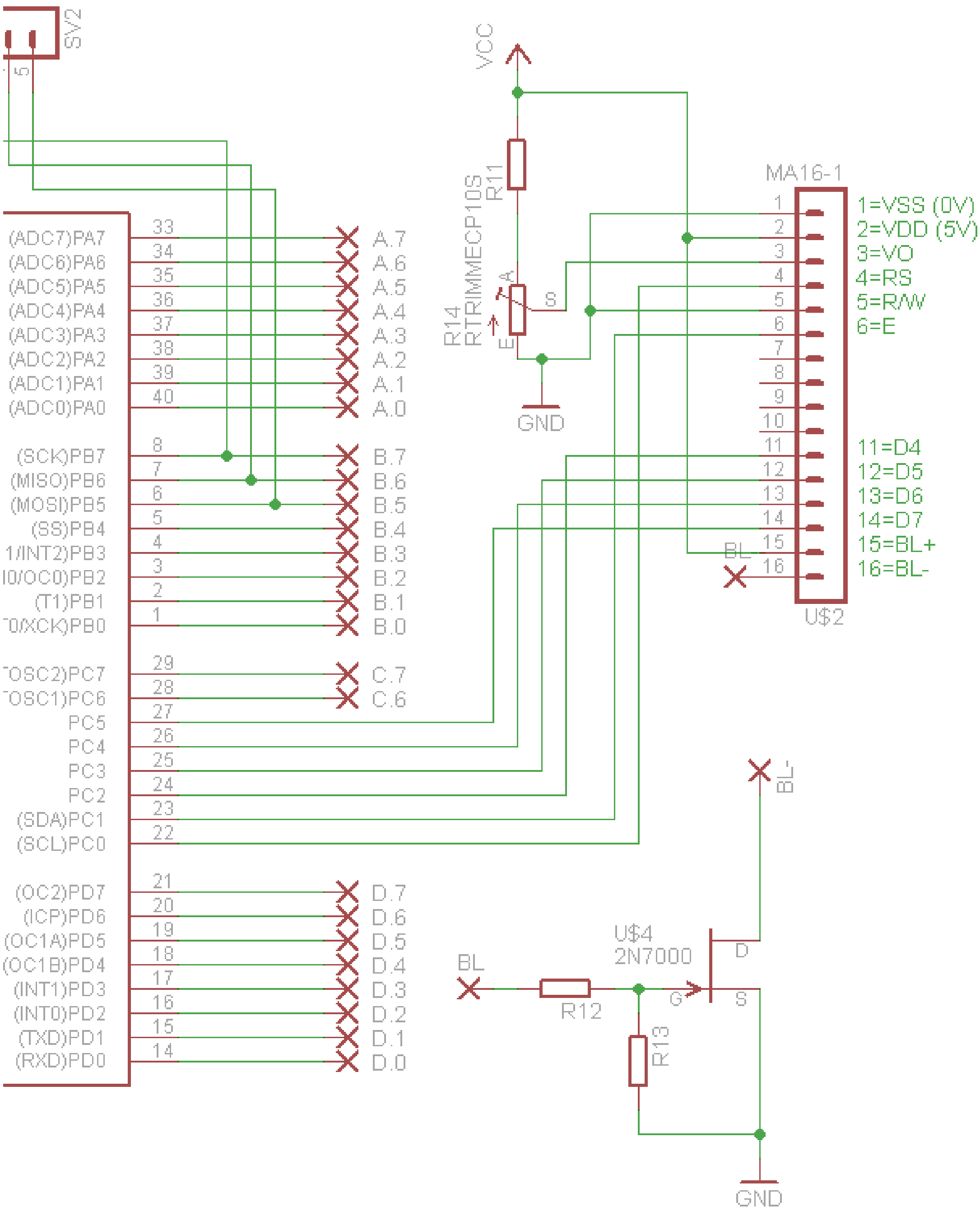
3. We can do this as the specification for each transitor and for the the whole package have not been exceeded.

4. We will need a power supply capable of delivering 12V and 1.2A (8 x 0.15A) plus other power requirements of the circuit.

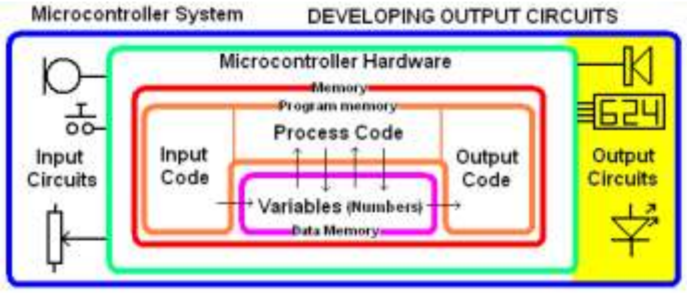
25.14 Connecting a FET backlight control to your microcontroller



The LCD requires six I/O lines to be used on the micro to control the data to it plus 1 more to switch the backlight

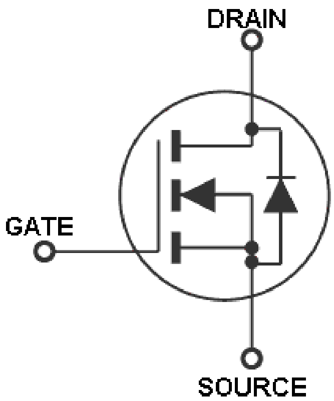


25.15 FET backlight control



The FET (field effect transistor) is different to the more familiar BC547 which is a BJT (bipolar junction transistor).

- A FET's output current is controlled by the voltage in and there is almost no current in the gate of the FET from the microcontroller meaning a microcontroller can control large FETs directly.
- Generally FETs require about 10V to drive the gate but low voltage versions called 'logic' FETs are available. The 2N7000 is a logic MOSFET, capable of driving 200mA loads and dissipating 1W of power at 25 degrees Celsius AND can be controlled directly by 5V (a logic 1) from a microcontroller
- The power dissipated by a FET is much lower than a BJT. It is measured by multiplying the current flowing by the R_{ds} value (5ohms for a 2N7000, but typically milliohms for high current FETs)



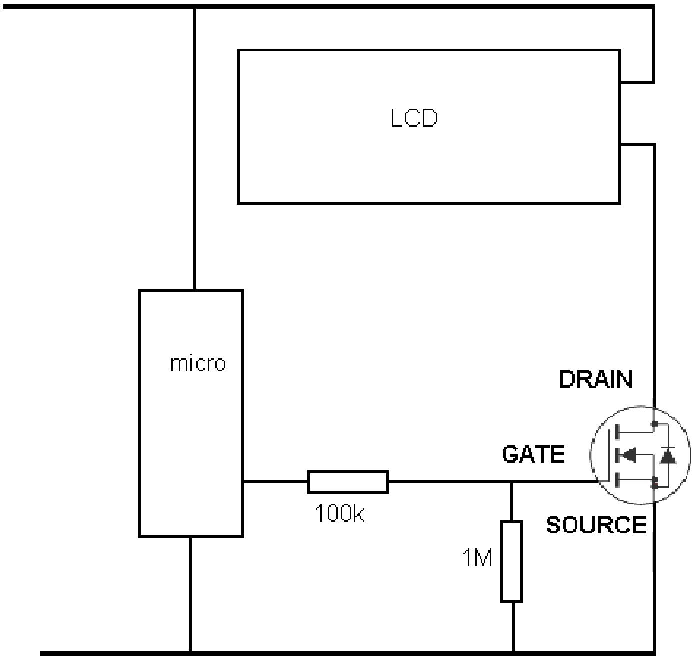
2N7000 – 'N channel enhanancement-mode MOSFET'

BV_{DSS} / BV_{DGS}	$R_{DS(ON)}$ (max)	$I_{D(ON)}$ (min)
60V	5.0Ω	75mA

I_D (continuous)*	I_D (pulsed)	Power Dissipation @ T_C = 25°C
200mA	500mA	1W

The FET can be connected directly to the microcontroller output pin without the 100k resistor; however we prefer to connect it with a high value resistor.

It is good practice to connect the gate to ground with a high value resistor. The reason being that the gate is so highly sensitive that if the mircrocontroller pin is configured as an input it will easily drift in voltage and the FET might turn on due to noise nearby in the circuit (and so will the device you have connected to it). This is the case when an AVR is turned on and before any config statements have been run in your program.

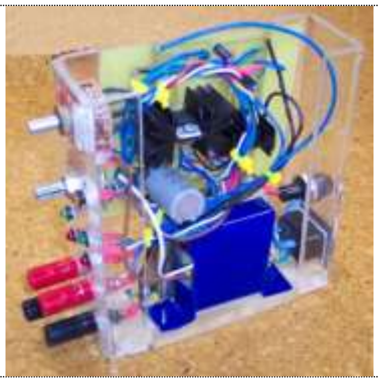


In worst case the power dissipation will be
P=V x I and V= I x R so P = I x R x I
P= I^2R = 0.07 x 0.07 x 5 ohms
P = 0.0245W = 24.5mW
So a lot less power is wasted by using a FET rather than a BJT.
Note the 5 ohms in the datsheet is a maximum value for RDS, looking through the datsheet shows that it is typically going to be around 2 ohms, but we used 5 as a worst case scenario.

26 Advanced Power Supply Theory

Every circuit needs high quality power

Over time a significant number of students have developed power supplies and breadboard prototyping centres for their own use



Some have been built into existing items like this toolbox

Some included microcontroller based control of the voltages

A range of various materials are used

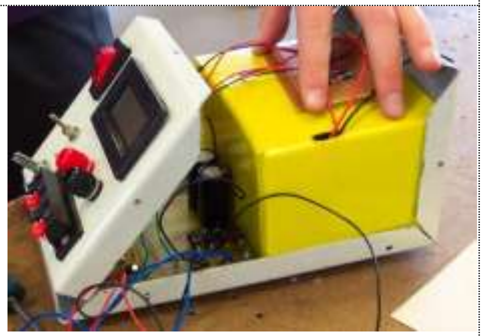
Sheet metal

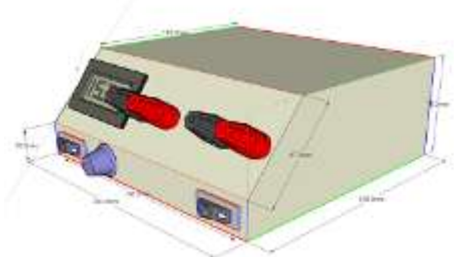
Acrylic

MDF

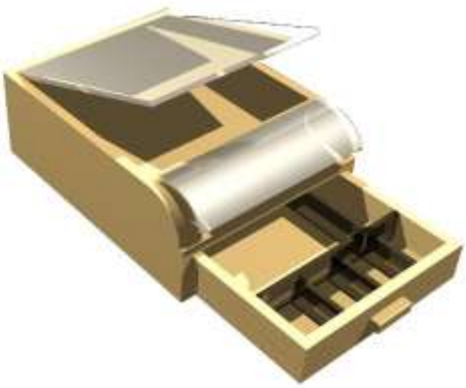
Pine

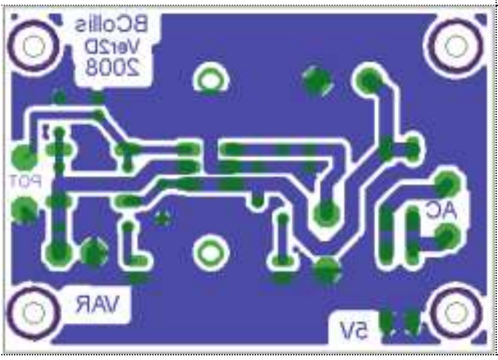
Plywood





Sketchup plays a significant role in helping students visualise and plan the final product





26.1 Typical PSUs

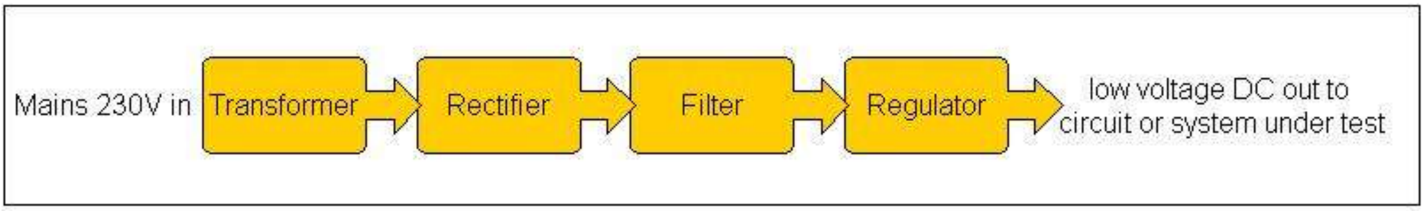
Typical power supply units and their characteristics/features

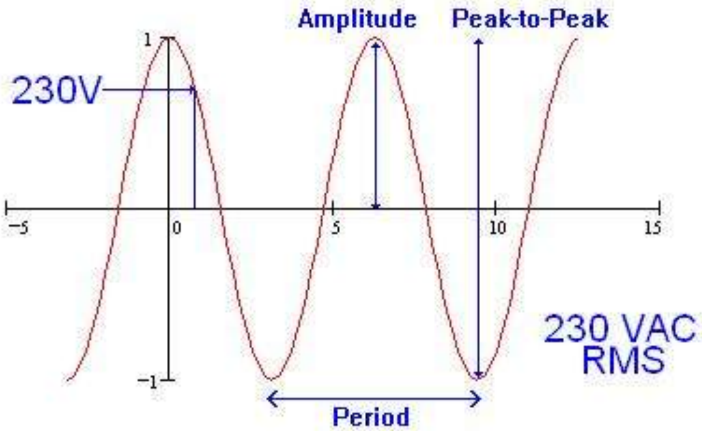
	Input voltage range: 230V AC 50Hz DC Output range: 5 - 15 Volt Output Current:10 amp DC (no variable limit) Analog Amp & Volt Meter 270mm x 200 mm x 120mm
	Input Voltage: 230-240V AC. 50Hz Output Voltage: Variable 0 to 30V DC Variable Output Current Limit: 0 to 2.5A Load regulation (0-100% load): 10mV Line regulation (240V +/-5%): 10mV Digital Volt and Ammeter Accuracy: 0.7% Mains overcurrent protection: 1A resettable circuit breaker
	Output Voltage: 3-15V DC or fixed at 13.8V Output Current: 40A regulated (no variable limit) Ripple & noise: 10mV rms Load regulation: 230mV @ 0 - 100% load. Measures 220(W) x 110(H) x 300(L)mm. Weight 3.5Kgs. – digital volt and amp meters
	Input voltage range: 230V AC DC Output range: 5 - 15 Volt Output Current:10 amp DC (no variable limit) Analoge Amp & Volt Meter 270mm x 200 mm x 120mm
	Input Voltage:240VAC10%/50Hz Output Voltage: 0 - 30 Volts DC Output Current: 0 - 5 Amps Line Regulation: ≤0.01%+3mV ≤0.2%+3mA Load Regulation:≤0.01%+2mV ≤0.2%+3mA Ripple & Noise: ≤0.5mVrms ≤3mArms Display Accuracy:Voltmeter(0.2%Rdg+2digits), 2.5% Full Scale
	Silicon Chip Magazine power supply kitset Two independently switched outputs: 5V,12V &15V
	Voltage outputs: +1.25V to 15VDC @ 0.25A. –1.25V to –15VDC @ 0.25A. +5VDC @ 0.25A. +30VAC center-tapped to 15VAC @0.25A. Various switches, LEDs, potentiometes, breadboard for testing circuits
	WOW: a power supply and breadboard prototyping super kit Extra features include: tools storage, multimeter

The specifications we need to know more about are highlighted above: Input Voltage range and frequency, variable output voltage range, output current limits, ripple, line regulation and load regulation.

26.2 The four stages of a PSU (power supply unit)

Most modern electronic devices require fixed and stable power supply voltages, to achive this we follow a recommended design.



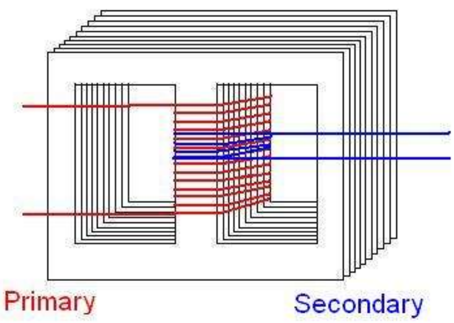


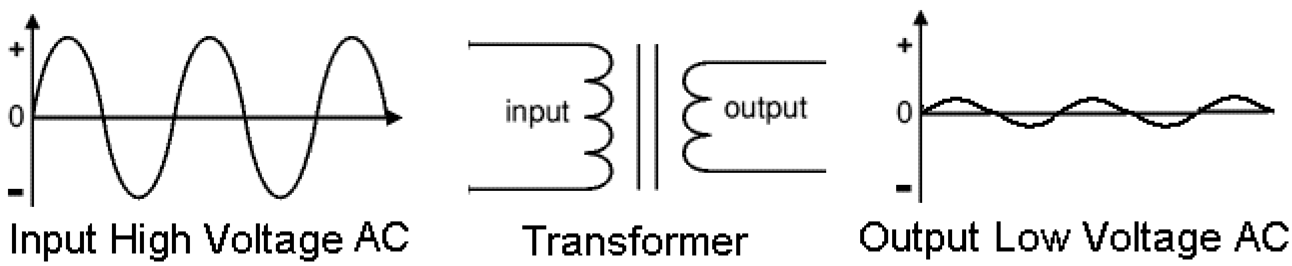
In NZ we use an AC (alternating current) mains power supply system which delivers 230V to our homes.
The 230 is an RMS (root means square) value. Although it is 230VRMS it peaks at about 230 x 1.414 (+325V and – 325V).

Of course we cannot use 230V directly in our projects as it is unsafe to so so. We use a transformer to convert the voltage to a lower value. A transformer is 2 (or more) insulated coils of wire wound on a laminated metal core.

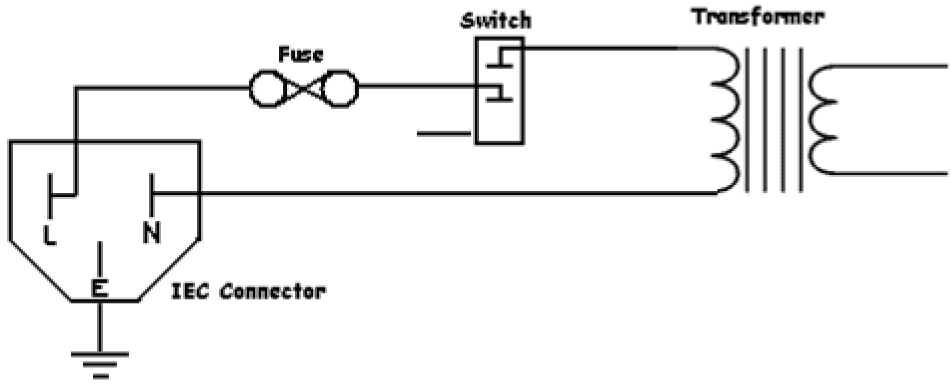
The ratio of the number of turns between the primary and secondary windings determines the voltage out put. If we want 23Volts out of our transformer we would have 1/10th the number of windings on the secondary as we have on the primary.

26.3 Stage 1: step down transformer



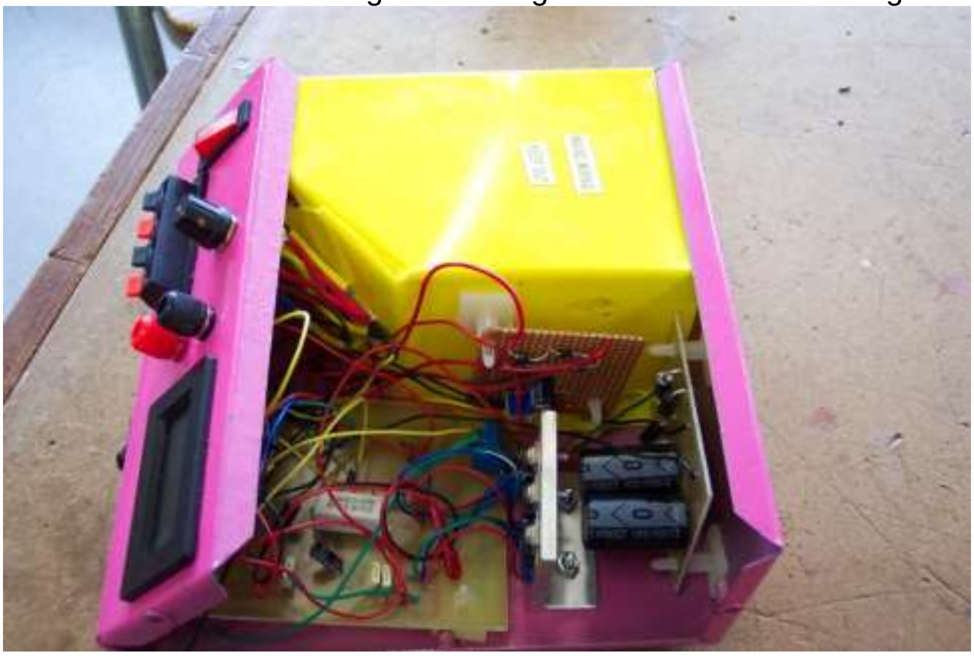


Wiring up our own mains transformer within a project is complex and requires a specific process to be followed thoroughly. This circuit looks simple enough it shows the switch, fuse, mains connector and primary of the transformer all in series.



However the actual product requires very specific wiring and earthing as well as testing by a registered person before it is used.

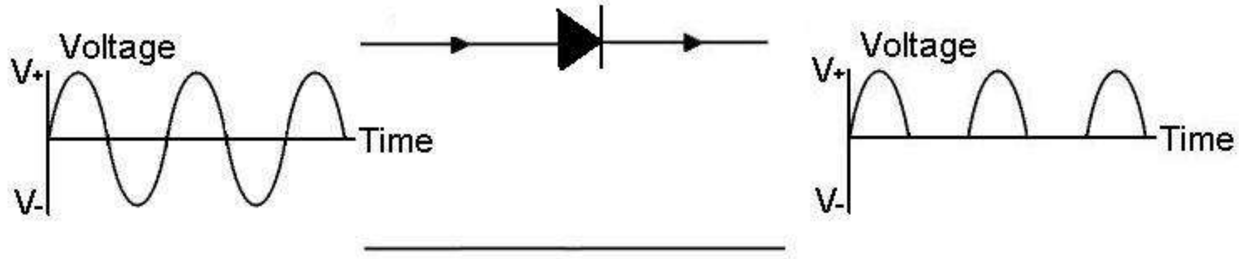
In this power supply DH covered the mains area with a plastic cover, then we had it certified by an electrician before mounting and testing the rest of the low voltage circuits.



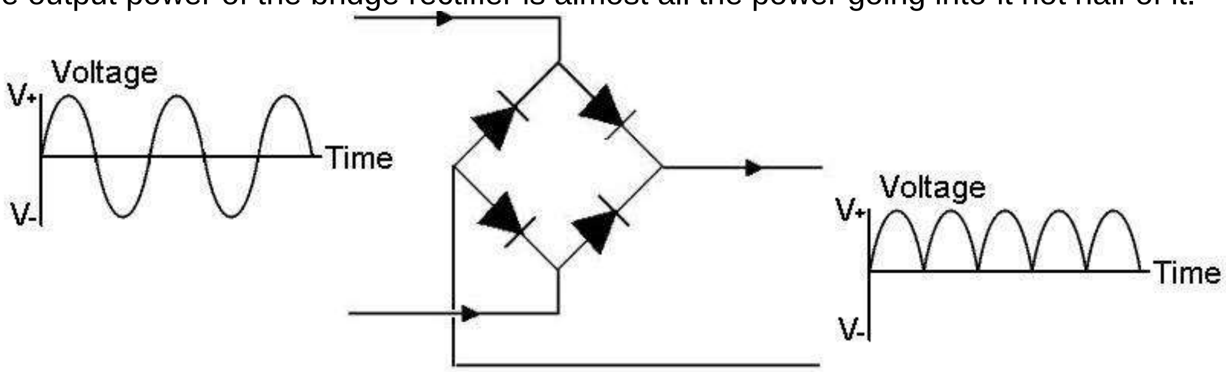
26.4 Stage 2: AC to DC Conversion

The second stage of the power suppy requires the conversion of AC to DC because all the circuits we use require DC voltage. A diode **rectifies** the AC .

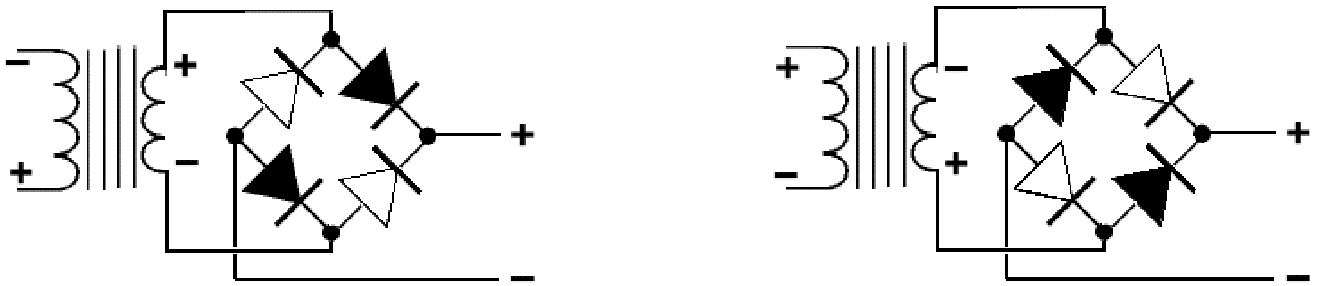
A half-wave rectifier (a single diode) blocks the negative voltage. This is however very inefficient use of a resource as half the power is never available for use – this means we might buy a 100VA transformer but only be able to use 50VA – translrmers are expensive so this is a waste of money.



A more efficient use of the power is to use a full wave rectifier, where there are 4 diodes. The output power of the bridge rectifier is almost all the power going into it not half of it.

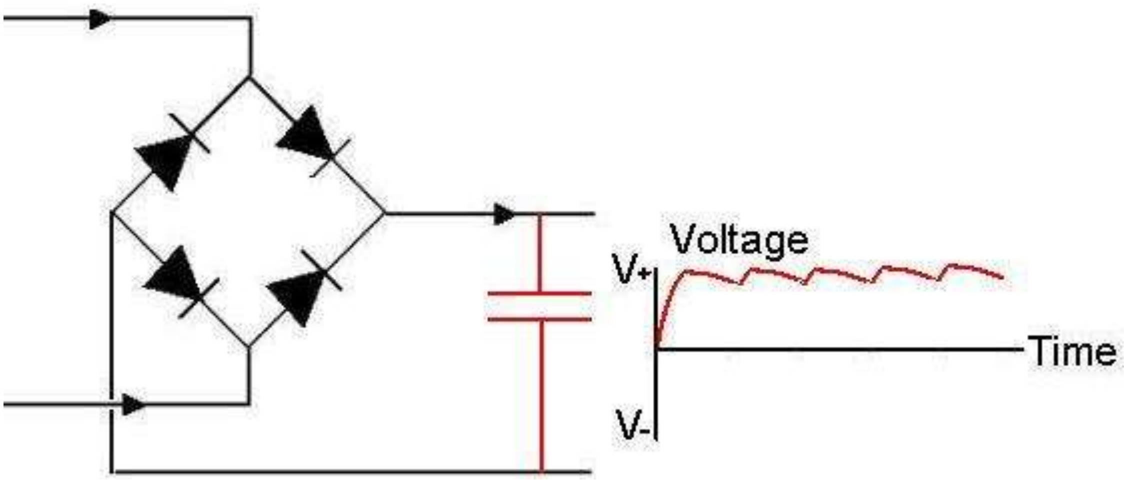


When the mains voltage is one polarity only two diodes conduct.



When it is the opposite polarity the other two diodes conduct. The output however is always the same polarity

26.5 Stage 3: Filtering AC component

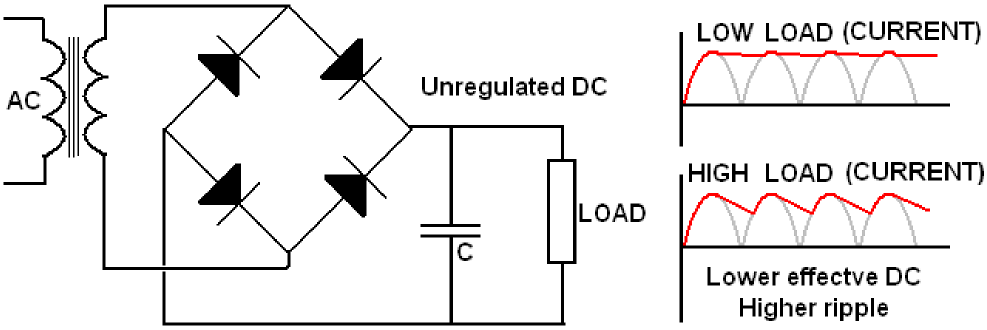


We need a steady DC voltage from our power supply, to assits we will use a capacitor. A capacitor is made of two metal plates separated by an insulator (called a dielectric). The characteristic of a capacitor is to store charges (electrons). If there is no voltage on a capacitor and a voltage is applied a large flow of charges (current) will occur, when the applied voltage is removed the capacitor will release these to the circuit. In our Power supply circuit the voltage rises and falls 100 times per second, while the voltage is low the charges stored in the capacitor will be used by the circuit, while it is high the charges used by the circuit will be supplied by the rectified AC which will also charge the capacitor. In a power supply we typically use very large capacitors e.g. 2200uF or 4700uF.
These capacitors are polarised, so must go around the right way – they can explode so get it right!
We also need to make sure that the voltage rating is more than the peak volate of the transformer. So a 13VAC transformer will have a peak output of 13x1.414 = 18V. Capacitors come in standard values 16V is a common value as is 25V. A 16V capacitor will not do, here a 25VDC one was used.

26.6 Stage 4: Voltage Regulation

The 'DC' coming out of the filter section of the PSU is not completely smooth and it has a slight ripple component due to capacitor discharging and recharging. As the load changes the ripple increases (the load is the circuit we connect to the PSU and we show it as a resistor in the circuit below). This means that the voltage can go up and down as the load changes, something that happens a lot in digital circuits as things switch on and off.

Also we want 5V for our microcontroller, so an unstable 16-18V DC supply is too high.



From the portions of the datasheets below for the ATMega16 and the ATTiny26 we can see that they need around 5V for the standard higher speed devices and 3V would be fine for the type L devices Voltages that exceed 5.5V will very like damage the microcontroller. Every now and again there is a loud POP in the classroom and the smoke inside a microcontroller is released as another student forgets to check the voltage on the bench power supplies we are using and tries to run their micro at 30VDC!!

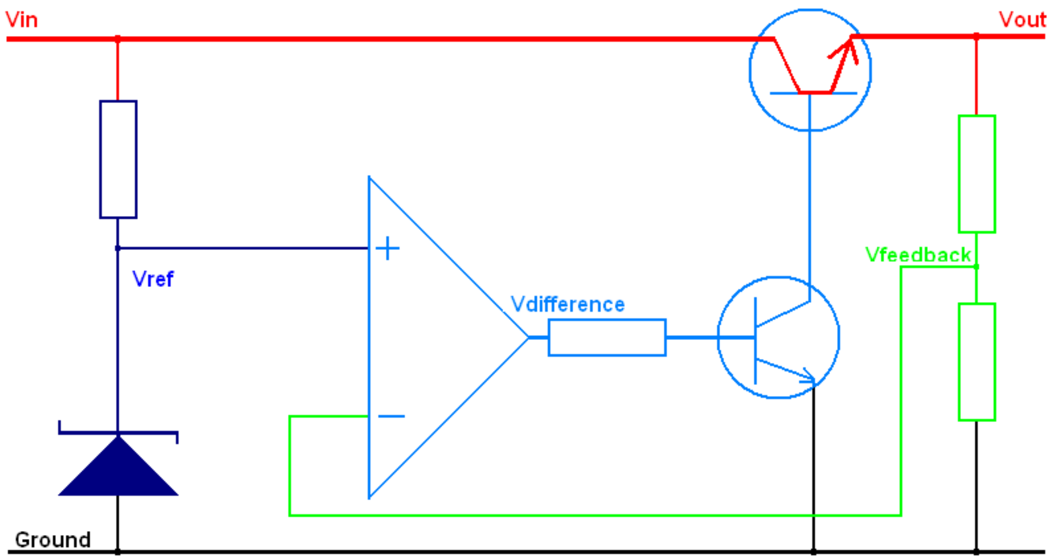
ATmega16

- **Operating Voltages**
  - **2.7 - 5.5V for ATmega16L**
  - **4.5 - 5.5V for ATmega16**
- **Speed Grades**
  - **0 - 8 MHz for ATmega16L**
  - **0 - 16 MHz for ATmega16**
- **Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L**
  - **Active: 1.1 mA**
  - **Idle Mode: 0.35 mA**
  - **Power-down Mode: < 1 µA**

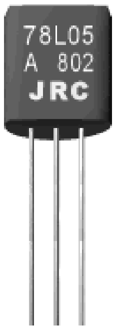
ATtiny26

- **Operating Voltages**
  - **2.7V - 5.5V for ATtiny26L**
  - **4.5V - 5.5V for ATtiny26**
- **Speed Grades**
  - **0 - 8 MHz for ATtiny26L**
  - **0 - 16 MHz for ATtiny26**
- **Power Consumption at 1 MHz, 3V and 25°C for ATtiny26L**
  - **Active 16 MHz, 5V and 25°C: Typ 15 mA**
  - **Active 1 MHz, 3V and 25°C: 0.70 mA**
  - **Idle Mode 1 MHz, 3V and 25°C: 0.18 mA**
  - **Power-down Mode: < 1 µA**

The output voltage must be controlled by some form of voltage regulator circuit. Here the regulator is a series pass transistor controlled by an opamp and transistor. The opamp compares the difference between the output voltage (Vfeedback from the voltage divider) and the reference voltage (Vref from the zener diode). It increases or decreases the drive voltage to the series pass transistor to keep the two input voltages equal.



Here is a common commercial device to do just that for us. It is the 7805 (or LM340T-5).
It comes in various package styles depending upon its use or its current limiting characteristics.

T092	TO220	TO5	SMD (surface mount device)
			

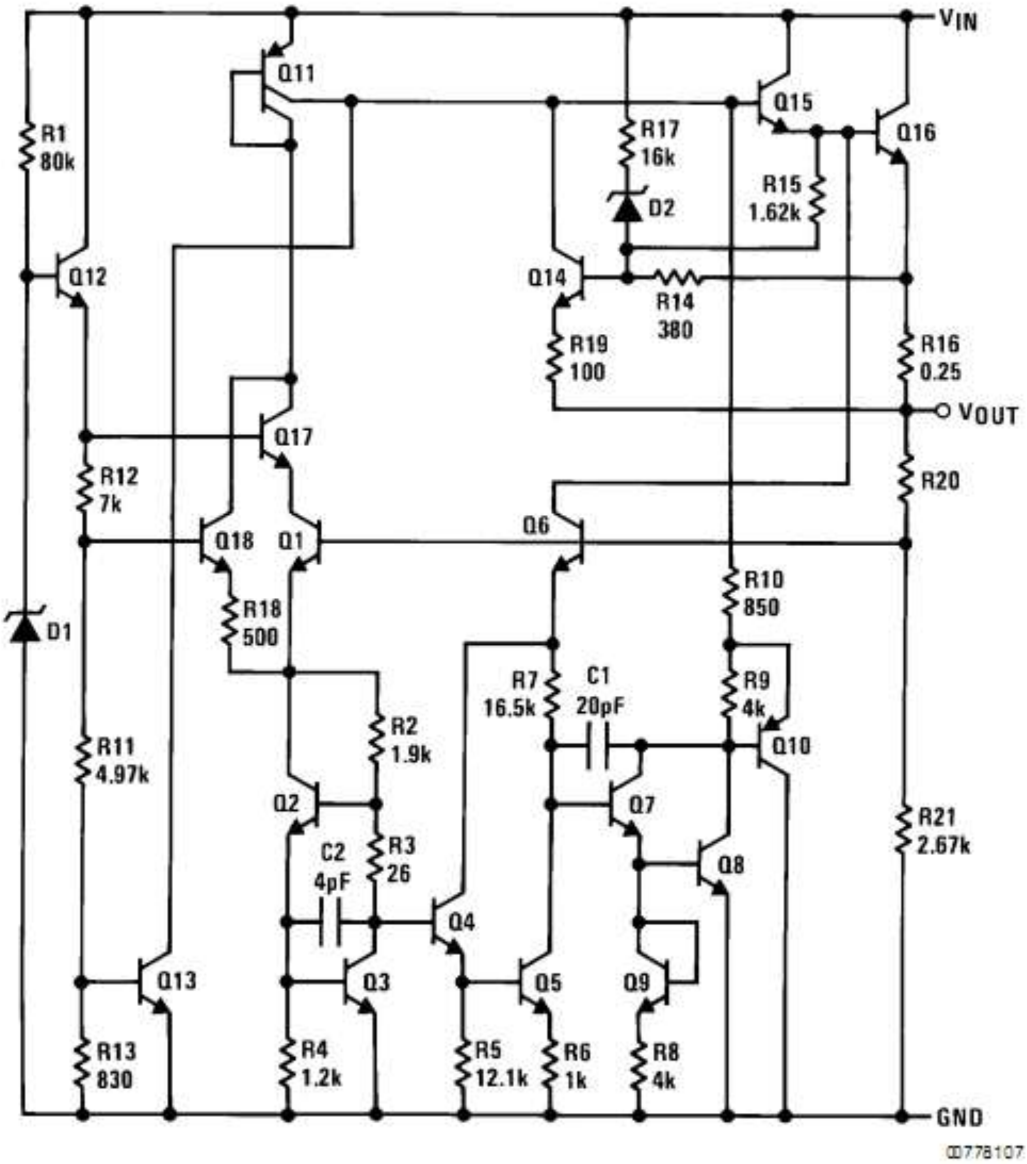
There are also different voltage ratings available e.g. 7808 (8V), 7812 (12V), 7815 (15V).

Inside the 7805 IC there is a reasonably complex circuit.

The components of interest however can be identified easily they are R1 and D1 (Vref), Q16 (series pass transistor), R20 and R21 (Vfeedback).

Transitors Q1 and Q18 form the main part of the comparator circuit.

This circuit has a current limit built into it, R16 is a 0.25ohm resistor and is used to detect the amount of current flowing, more about that later.



A 7805 can be added easily to our circuit. But we must know about it so that we use it correctly. The datsheet for a 7805 can be downloaded from the internet, here are some sections from it.

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

(Note 5)

DC Input Voltage	
All Devices except LM7824/LM7824C	35V
LM7824/LM7824C	40V
Internal Power Dissipation (Note 2)	Internally Limited
Maximum Junction Temperature	150°C
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 10 sec.)	
TO-3 Package (K)	300°C
TO-220 Package (T), TO-263 Package (S)	230°C
ESD Susceptibility (Note 3)	2 kV

Operating Conditions (Note 1)

Temperature Range (T_A) (Note 2)	
LM140A, LM140	−55°C to +125°C
LM340A, LM340, LM7805C, LM7812C, LM7815C, LM7808C	0°C to +125°C

LM340A Electrical Characteristics

I_{OUT} = 1A, $-55°C \le T_J \le +150°C$ (LM140A), or $0°C \le T_J \le +125°C$ (LM340A) unless otherwise specified (Note 4)

Symbol	Output Voltage		5V			12V			15V			Units
	Input Voltage (unless otherwise noted)		10V			19V			23V			
	Parameter	Conditions	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
V_O	Output Voltage	T_J = 25°C	4.9	5	5.1	11.75	12	12.25	14.7	15	15.3	V
		$P_D \le 15W$, 5 mA $\le I_O \le$ 1A	4.8		5.2	11.5		12.5	14.4		15.6	V
		$V_{MIN} \le V_{IN} \le V_{MAX}$	$(7.5 \le V_{IN} \le 20)$			$(14.8 \le V_{IN} \le 27)$			$(17.9 \le V_{IN} \le 30)$			V

What is the maximum input voltage? _________________

What do you think storing the device below -65 degC might do to it?

If it got hotter than its maximum operating temperature of _______ degC what might happen?

What is the typical output ?________________,

the maximum output voltage?_____________

the minimum output voltage? _____________

					28.5)	
R_O	Dropout Voltage	T_J = 25°C, I_O = 1A	2.0	2.0	2.0	V
	Output Resistance	f = 1 kHz	8	18	19	mΩ

From the small section above we can determine what the minimum input voltage is that we can use to get 5V out. This spec is called dropout voltage and it is the voltage difference between the input and output that is required to make sure the 7805 operates correctly.

To get 5V out we need at least ____________ input voltage

26.7 Ripple (decibel & dB)

$\frac{\Delta V_{IN}}{\Delta V_{OUT}}$	Ripple Rejection	T_J = 25°C, f = 120 Hz, I_O = 1A	68 80	61 72	60 70	dB
		or f = 120 Hz, I_O = 500 mA,	68	61	60	dB
		Over Temperature,				
		$V_{MIN} \le V_{IN} \le V_{MAX}$	$(8 \le V_{IN} \le 18)$	$(15 \le V_{IN} \le 25)$	$(18.5 \le V_{IN} \le 28.5)$	V

Although the filter capacitor reduces the ripple voltage we do not want any of it getting onto the power pins of our microcontroller. That sort of thing really upsets fast switching digital and microcontroller circuits and also can create hum in audio circuits. The 7805 rejects ripple, the datasheet gives its specification as 80dB (decibels).

A Decibel is a measure that is not linear but logarithmic in scale .
+3dB means 2times the power (or if a voltage is specificed ,1.4 times the voltage)
-3dB means half the power (0.71 x the voltage)

+6bB means 4x the power (2x the voltage)
-6dB means ¼ of the power (1/2 the voltage)

+80dB means 100,000,000 x the power (10,000 x the voltage)
-80dB means 1/100000000 of the power (1/10000 the voltage)

80db from the datasheet means it reduces ripple output to 1/10000 of the ripple voltage coming in.

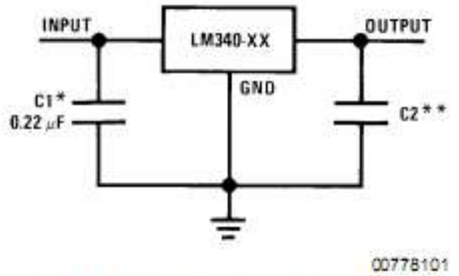
If the ripple voltage was 100mV (0.1V) coming in it would be ______________ coming out of the 7805 (not much!)

The power supply units looked at earlier had ripple specifications of 10mV that means that if we set our PSU to 5V then the voltage will fluctuate from 4.990V to 5.010V at the rate of 100 Hertz (100 because we full wave rectify the 50Hz AC voltage)

Often a datasheet will give typical applications for a device

Typical Applications

Fixed Output Regulator



*Required if the regulator is located far from the power supply filter.

**Although no output capacitor is needed for stability, it does help transient response. (If needed, use 0.1 µF, ceramic disc).

Note: Bypass capacitors are recommended for optimum stability and transient response, and should be located as close as possible to the regulator.

The note about the two small capactiors is very important when designing a 7805 circuit put them real close to the IC (within a few millimetres)

As an aside I always use at least a 10uF electrolytic capacitor on the output of the 7805 if I will be using the ADC circuit of the ATMEL AVR, as this makes the ADC readings more stable!

26.8 Line Regulation

Line regulation refers to the line input voltage varying. In our case we have a nominal (typical) mains voltage of 230V AC. This voltage however fluctuates as people turn applicances on and off, expecially large power users. So these changes in line input voltage should not effect the output voltage.
One of the power supplies above quoted Line regulation (240V +/-5%): 10mV this means that if the mains voltage varies by up to 5% either side of 240V then the output voltage will change by no more than 10mV. Another one quoted Line Regulation: ≤0.01%+3mV so when the input AC voltage varies 0.01% of that variation + 3mV may be passed through to the output.

The 7805 Line regulation from the datasheet is 10mV, which means that if the input DC voltage changes then the output voltage will change no more than 10mV.

26.9 Load Regulation

Load regulation is perhaps the most important specification for our power supplies as we want the output voltage to be constant while our circuits current load changes (i.e. we trun LEDs, motors etc on and off). Three of the power supplies had specifications for load regulation.

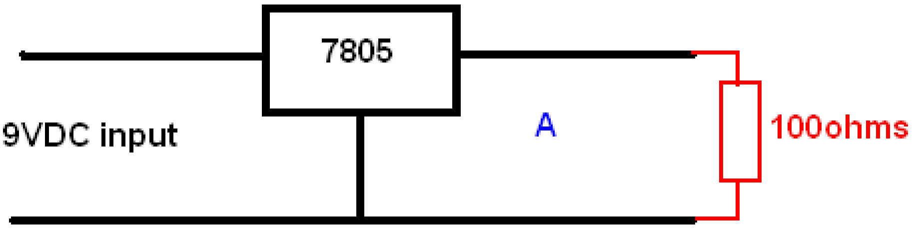
Load regulation: 230mV @ 0 - 100%
Load regulation (0-100% load): 10mV
Load Regulation:≤0.01%+2mV

The first one is the worst upto 230mV variation, so a 5V setting might drop down to 4.770V, the second at 10mV means that the 5V would drop down to 4.990V and the last one by a little more than 2mV.

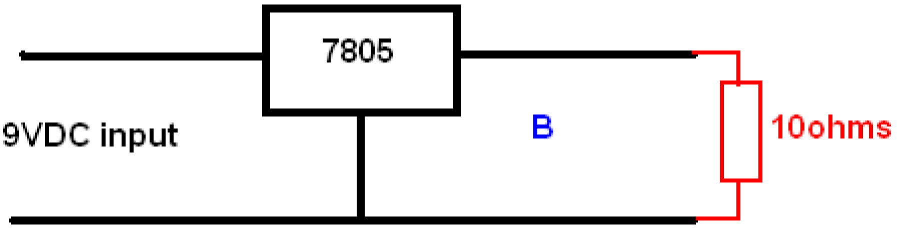
The 7805 has a load regulation specification of 10mV typical and a maximum of 25mV. So it is really good!

26.10 Current Limit

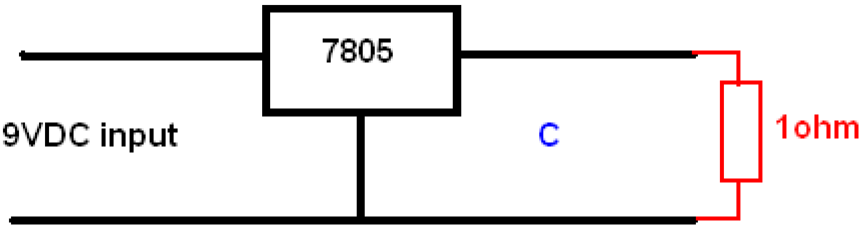
Although we regulate voltage we seldom regulate the current that a circuit can draw. Using ohms law we can work out what the different currents are for circuits below



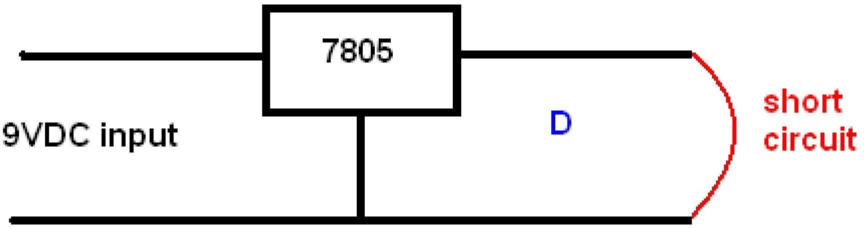
In this circuit 5V into a load of 100 Ohms
I=V/R = 5/100 = 0.05A or 50mA



In this circuit 5V into a load of 10 Ohms
I=V/R = 5/10 = 0.5A or 500mA



In this circuit 5V into a load of 1 Ohms
I=V/R = 5/1 = 5A (!!!!!!!!!)
NO it wont work, sorry



I=V/R =5/0, don't try this on your calculator because it generally gives E, which students sometimes think is infinity so infinite Amps!

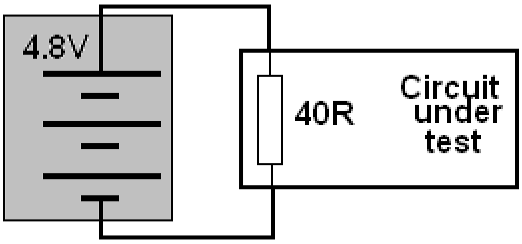
The 7805 has a built in current limit circuitry to protect itself

Symbol	Output Voltage		5V			12V			15V			Units
	Input Voltage (unless otherwise noted)		10V			19V			23V			
	Parameter	Conditions	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
	Short-Circuit Current	T_J = 25°C		2.1			1.5			1.2		A
	Peak Output Current	T_J = 25°C		2.4			2.4			2.4		A
	Average TC of V_O	Min, T_J = 0°C, I_O = 5 mA		−0.6			−1.5			−1.8		mV/°C

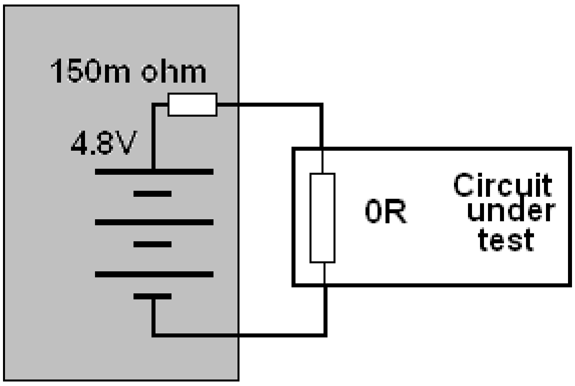
It can deliver no more than 2.1A maximum. HOWEVER, current limit is a function of the whole circuit, if your 9VDC coming in is provided by a plugpack that has a 500mA output rating then you will only ever get 500mA max (trying to draw more may kill the plugpack) If it is coming from a 10A power supply then it will allow you to draw an absolute max of 2.1A if you put a short circuit on the 7805.

What exactly is current limiting and why is it important?

Often batteries are used to test circuits. This is fine if the circuit is working well. The circuit under test may be drawing 120mA so it can be thought of as a (R=4.8/0.12 =) 40 ohm equivalent resistance.



If however you make a mistake with your breadboard or pcb and the circuit becomes **<u>0 ohms</u>** then a problem can occur!
In fact explosions can occur!!!

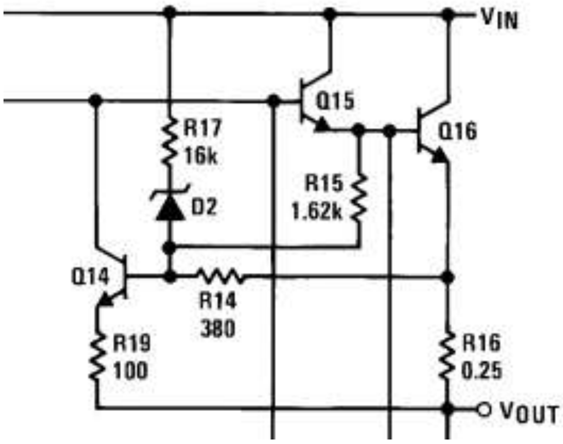


Batteries are not perfect but they are very good; they have a small internal resistance, which will limit the current. I = V/R = 4.8/0.15 = 32A!! This internal resistance depends on things like temperature and the chemical reactions going on and could even be lower.

In the class we have had batteries explode into fire. When testing circuits if it doesn't work check the temperature of your batteries, if they are very hot disconnect them and if they are really hot put them outside immediately; as they may explode even after having been disconnected as they can continue to heat up.

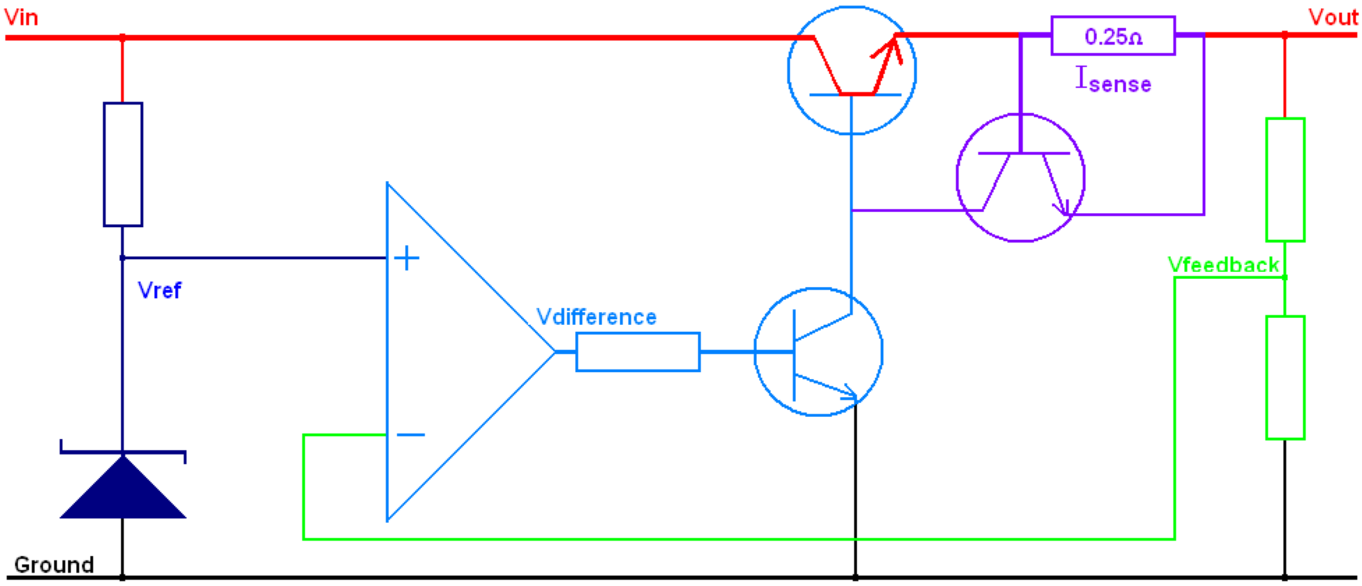
Check out the internet for videos and pictures of exploding batteries if you don't believe!

How does current limiting work?



The 7805 current limiting circuitry from the datasheet (above), this has been reduced down to a basic block type diagram in the circuit below.

Between the input and output of the 7805 is transistor Q16 and resistor R16 (0.25ohm). The current that the 7805 supplies to the circuit goes through the 0.25ohm Current Sense (I_{sense}) resistor.
This resistor will develop a voltage (potential difference) across it which is directly proportional to the current (V = IxR - ohms law - as current increase so does voltage).



V = I x R

At 50mA
V = 0.05 x 0.25 = 0.0125V

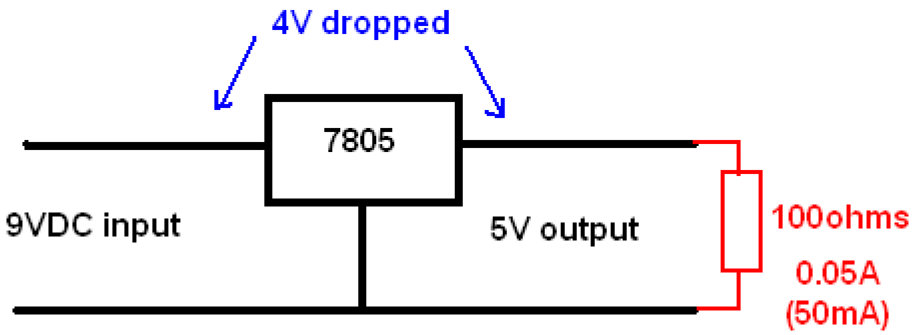
At 100mA
V = 0.1 x 0.25 = 0.025V

At 1A
V = 1 x 0.25 = 0.25V

At some point the current sensing transistor Q14 will turn on and shut off the main transistor Q16.

26.11 Power, temperature and heatsinking

Using this diagram we can work out some power calculations for our 7805.



If the 7805 needs to drop 4V at 0.05A, then it will have to **dissapate** 0.05A x 4V watts of power

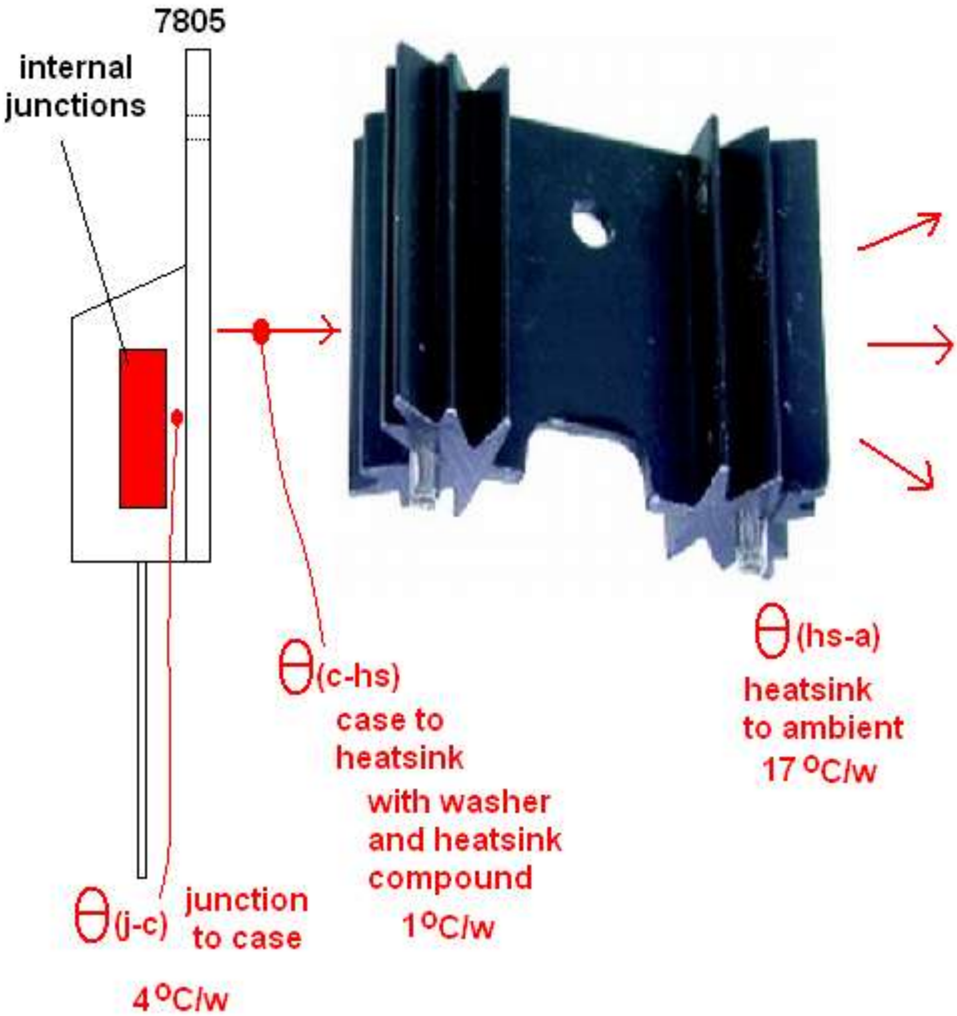
P=VI = 0.05 x 4 = 2Watts. In doing this the 7805 will act as a heater and get warm.
This is where the specifications for a device become very important, as we do not want to exceed the power ratings or damage may occur. Damage is not really a problem with the 7805 as it is "essentially indestructible" as the datsheet says. However it will shut itself down if it gets too hot.

> **Note 2:** The maximum allowable power dissipation at any ambient temperature is a function of the maximum junction temperature for operation (T_{JMAX} = 125˚C or 150˚C), the junction-to-ambient thermal resistance (θ_{JA}), and the ambient temperature (T_A). $P_{DMAX} = (T_{JMAX} - T_A)/\theta_{JA}$.
>
> If this dissipation is exceeded, the die temperature will rise above T_{JMAX} and the electrical specifications do not apply.
>
> If the die temperature rises above 150˚C, the device will go into thermal shutdown. For the TO-220 package (T), θ_{JA} is 54˚C/W and θ_{JC} is 4˚C/W.

The 'die' is the internal silicon wafer (slice) that the circuit is built on; if this goes over 150 °C the device will shut itself down. The 7805 is able to radiate heat however it has only a small surface area and so it is not very efficient at getting rid of heat. Its warms up at the rate of 54 °C/W . The specification of interest is Θj-a (theta junction to ambient).

If in the example we want to dissapate 4W then the junction temperatire will rise to 4 x 65 or 260 °C, clearly the device will shut itself down if this were to happen as it would get too hot.

So we bolt a heatsink to the 7805. The specification of interest becomes Θj-c (theta junction to case) which is 4°C /W



Each part of the chain of dissipating heat has a negative impact, the lower the overall number the better heat can be dissipated. A small heatsink might be 20°C/W, in this case the one shown is 17°C/W. A large heatsink might be 4°C/W.

If we use a mica insultor between the 7805 and the heatsink and thermal paste (to exclude any air from the join) it adds 1°C/W.

Our total is now 4+1+17 = 22°C/W. Much better than 54°C/W.
At 4W our junction temperatire will be 4*22 = 88°C, which is below the max of 150°C .

If we raise the input voltage to 16VDC, and we want to draw 1A from the 7805.
We will have 16-5 = 11V across the 7805 and it will have to dissipate 11V x 1A = 11Watts.

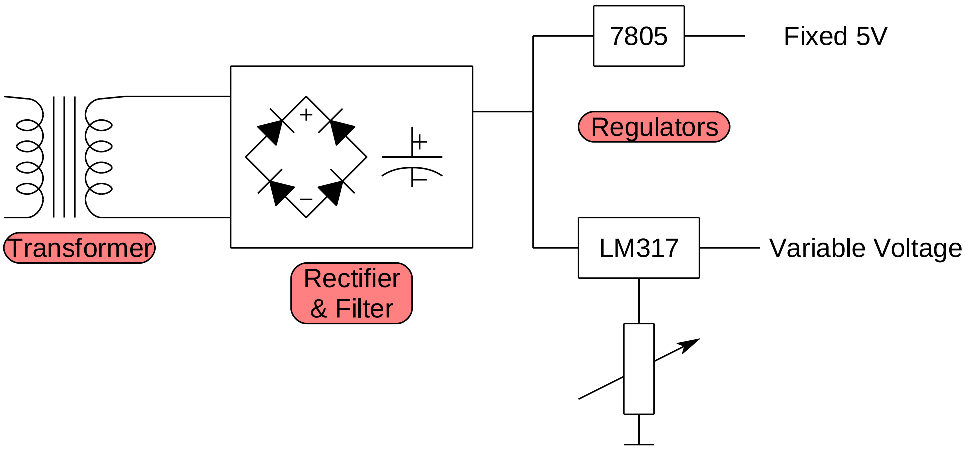
At 22 °C/W, that means 11W x 22 = 242°C.

To be within range of our 150°C we will have to have reduce the rating to 150/11 = 13°C/W.

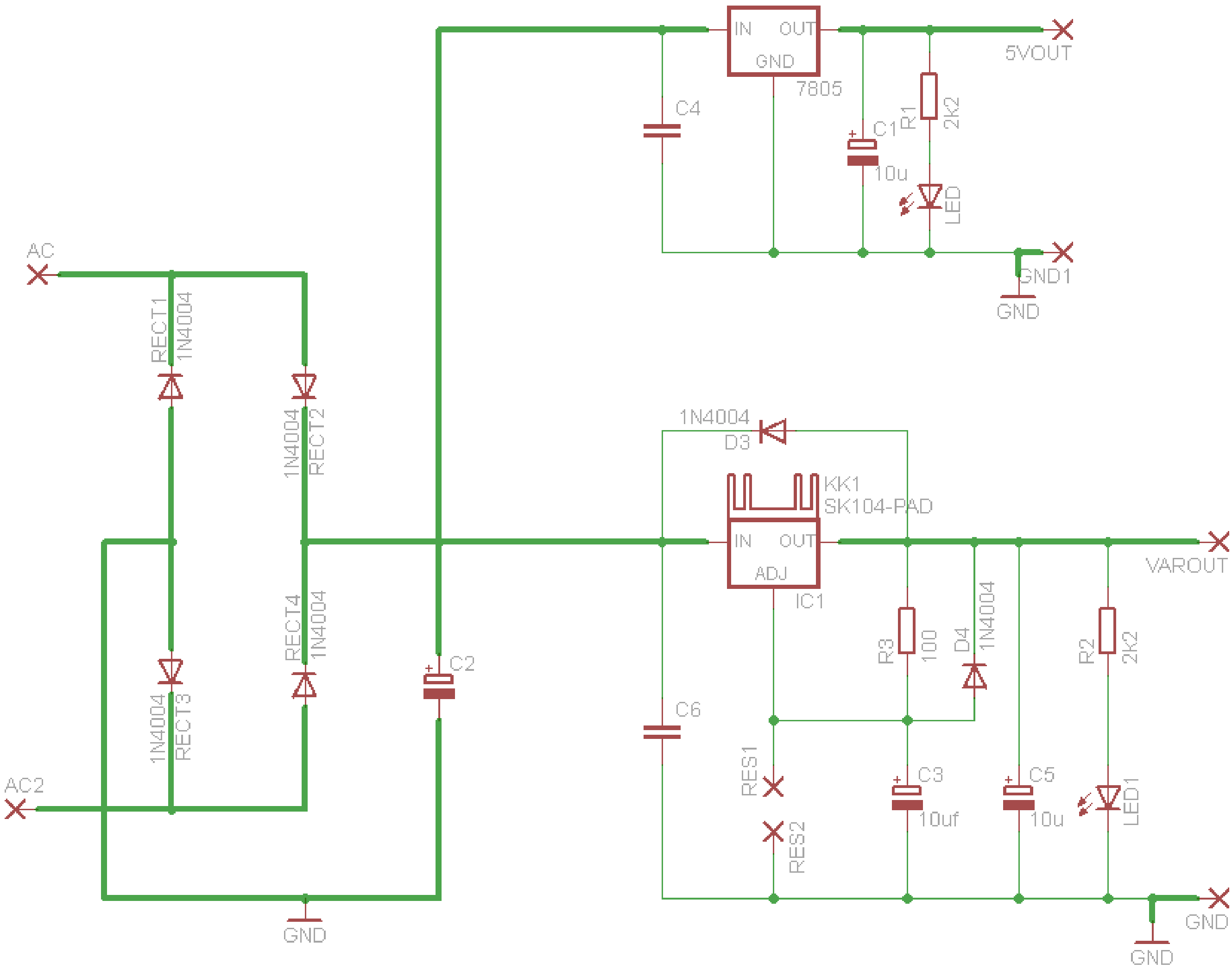
If we have a heatsink of 8°C/W it will be OK, but that is a reasonable size heatsink.

26.12 Typical PSU circuit designs

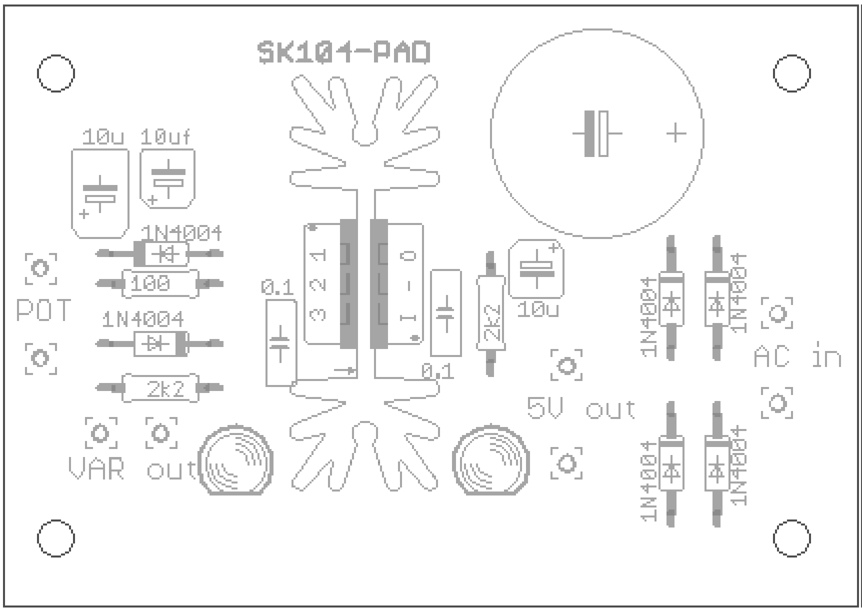
26.13 PSU block diagram



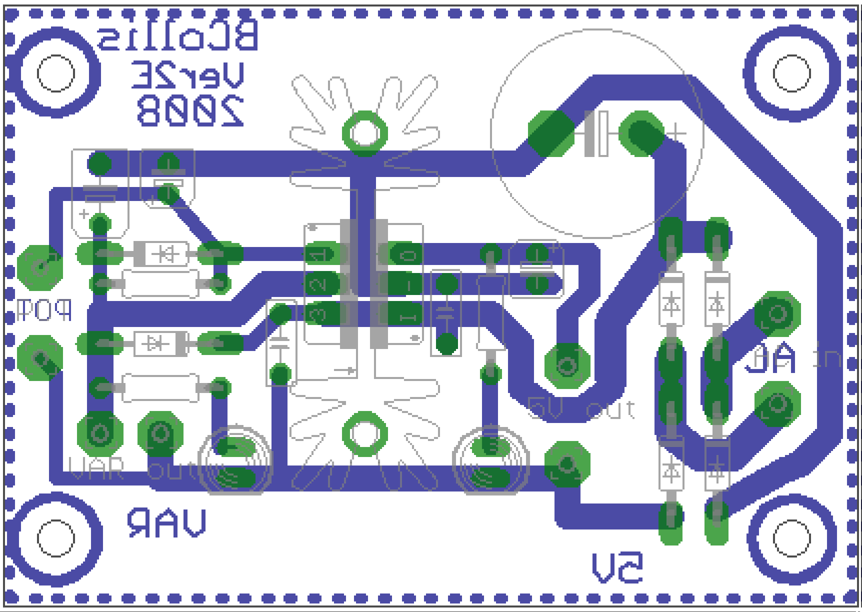
26.14 PSU Schematic



In this circuit the thick lines indicate higher current paths, which will require thicker tracks on the PCB Note that there is no current limit apart from the 7805 and LM317 internal current limits (at least that's better than 30+amps direct from batteries). Note the three GND connections, these points are connected as they are all called GND, there is no need to add wires to connect the points.

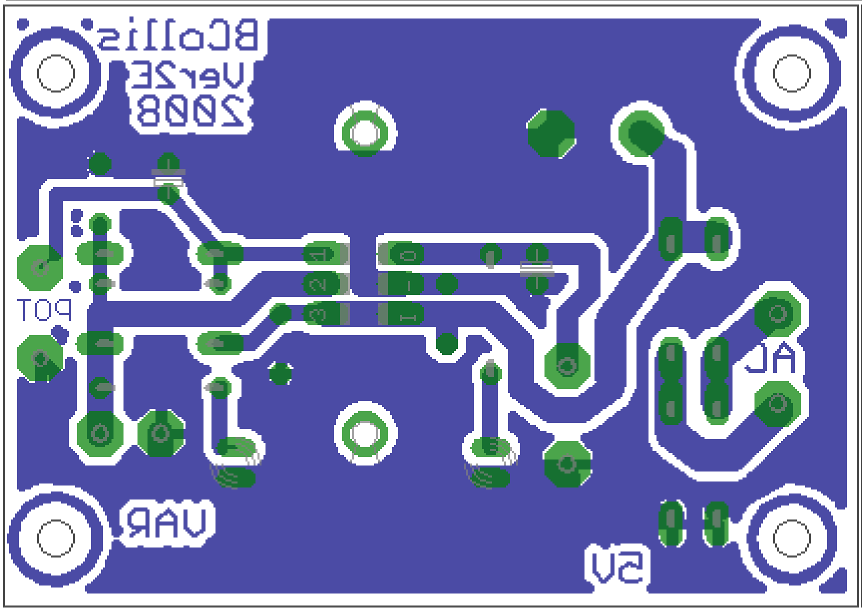


Initially layout your components in a logical way
Here a small heatsink was used in the centre of the PCB and the two regulators were mounted on either side of it. The components that belonged with each part of the circuit were put on each side of the heatsink. The capacitor and voltage regulator were added to one end f the board. The wires to connect to other components were all placed around as few sides of the board as possible, and as close to the edges as possible. 3.5mm mounting holes were placed in the corners.



Next the tracks were started. The ground was laid first around the outside of the board and using 0.086in thickness, this is the thickest track possible to connect to the voltage regulators as their leads are 0.1inch apart.

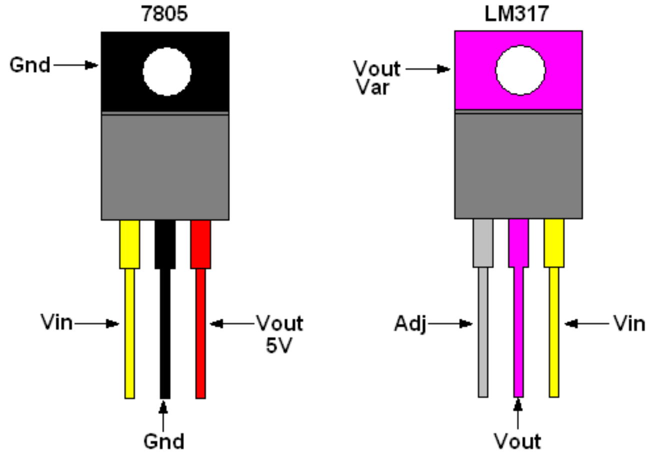
There is another consideration here, this is a powersupply designed to deliver current to other ciruits, we must know about the current limits of the PCB tracks. This is all to do with resistance and heat. A copper track although a conductor still has a finite resistance and will burn up if too hot (too much current flows through it). We use PCB which has 2oz (ounce) of copper per square foot. This equates to 0.0028 inch thick tracks. A 0.086 inch wde track can carry about 3.5A and will increase in temperature by about 10 degrees C. (which is ok).



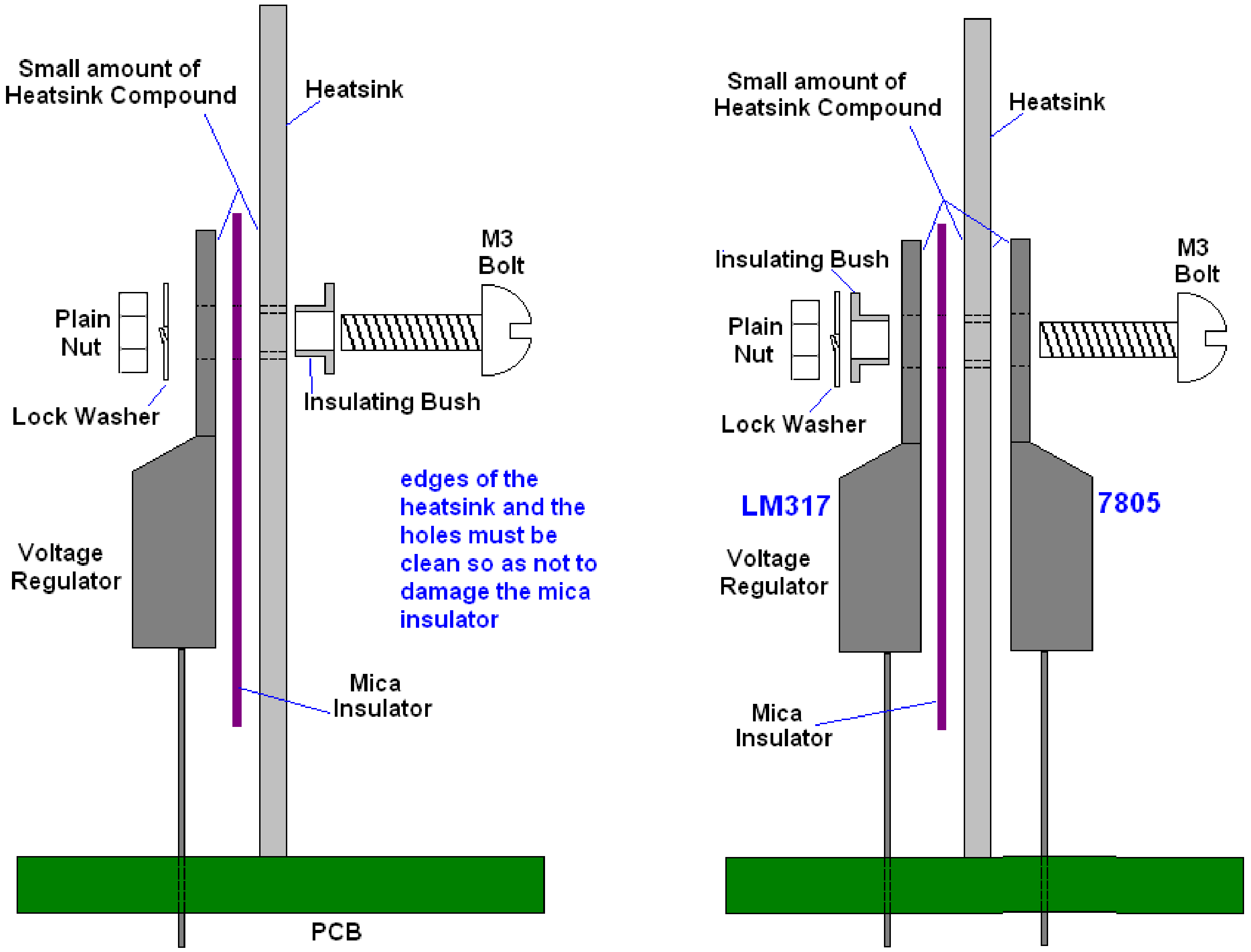
In an effort to reduce electrical noise and any voltage fluctuation a large ground plane is added to the board. Type ‘polygon gnd’ into eagle and set the values for width, isolate and spacing for 0.032 inch. Then draw the polygon around the edge of the board and redraw the ratsnest to fill in the polygon. A ground plane also reduces the amount of copper that will need to be etched, saving on chemicals.

Insulating of heatsinks and voltage regulators

Most devices need insulating from heatsinks, because the metal tab of the IC package os electrically conneted to one of the legs.

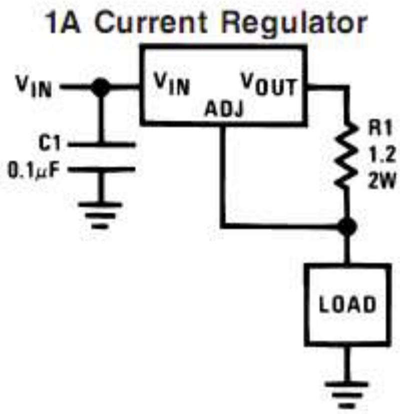


In the 7805 the metal tab is electrically ground (or 0V),
In the LM317 variable voltage regulator the metal tab is connected to Vout, the variable voltage.
If we were to bolt them to the heatsink without insultating them the variable voltage would short out to ground. When we have a 7805, its case is already ground sowe don’t need to insulate it, but the LM317 still needs insulation.

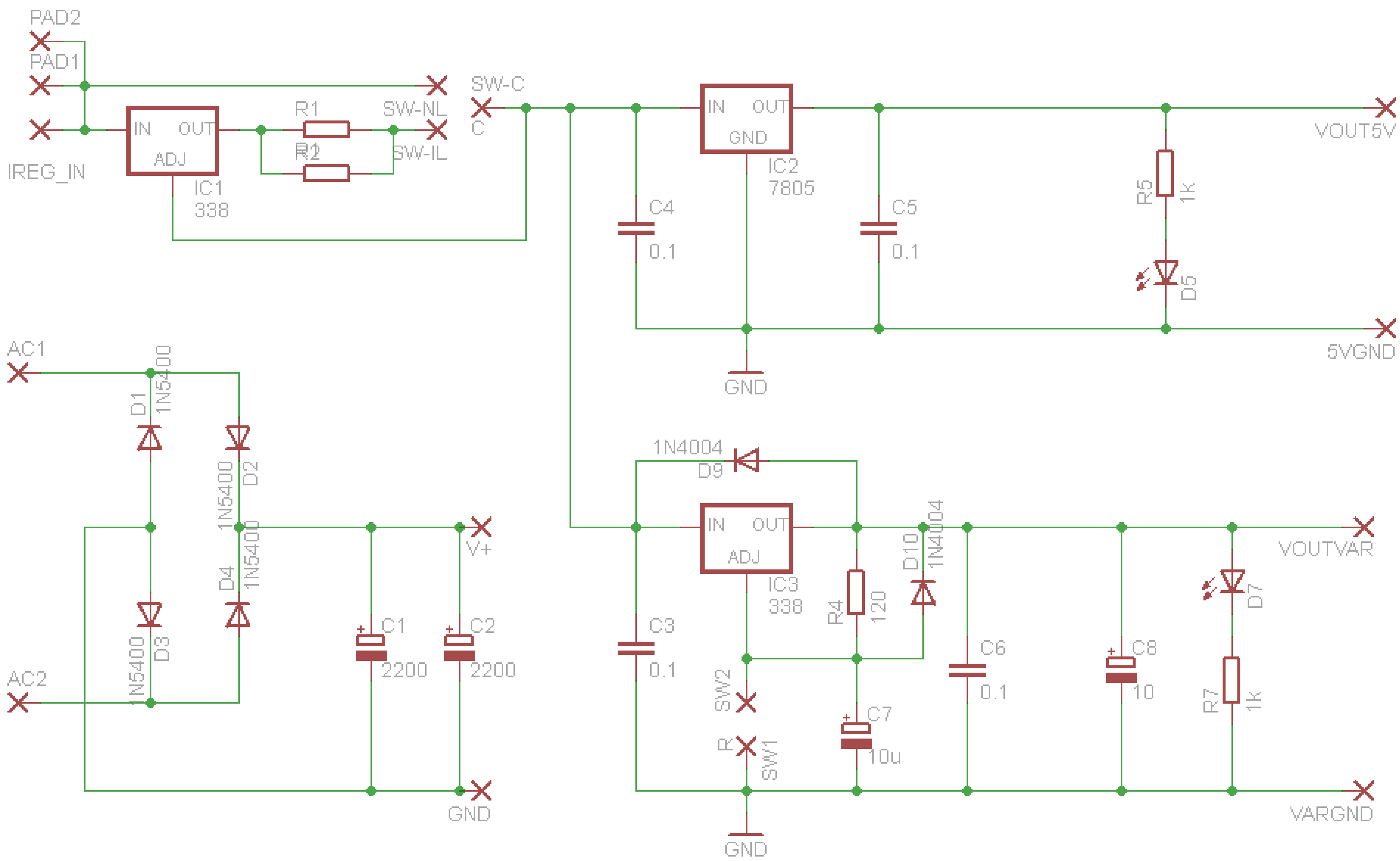


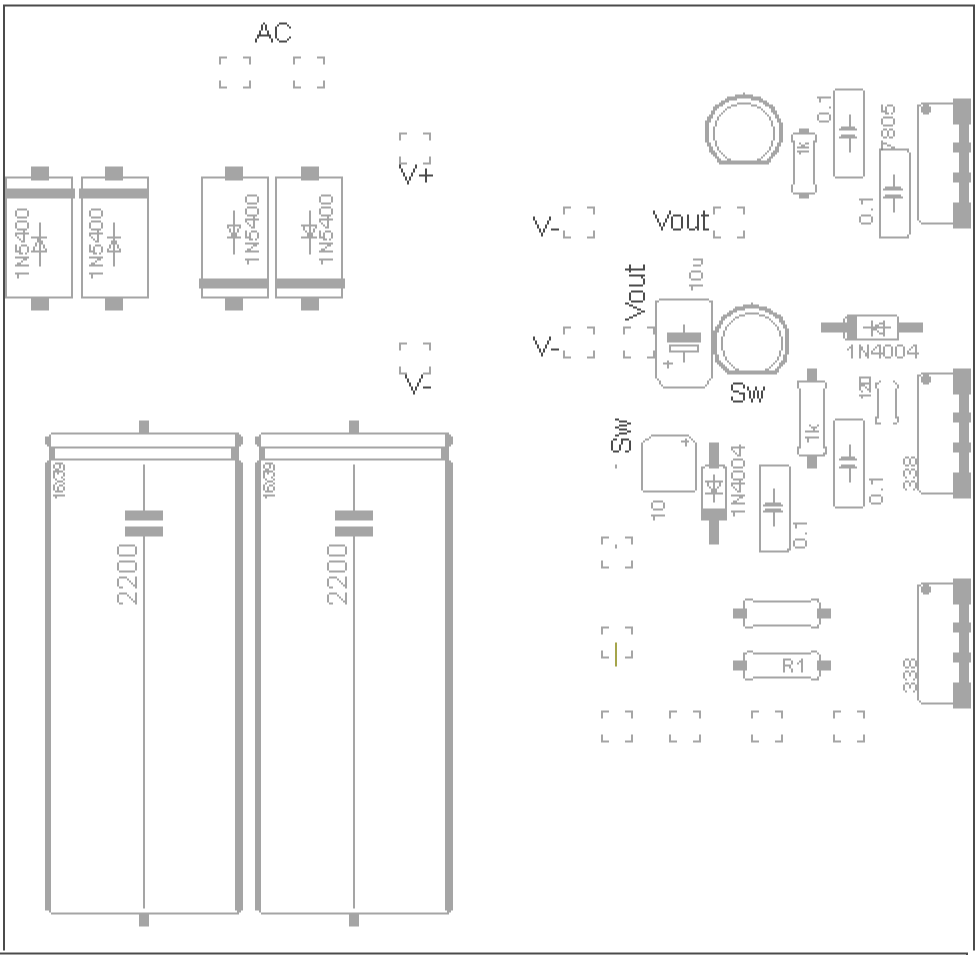
26.15 Practical current limit circuit.

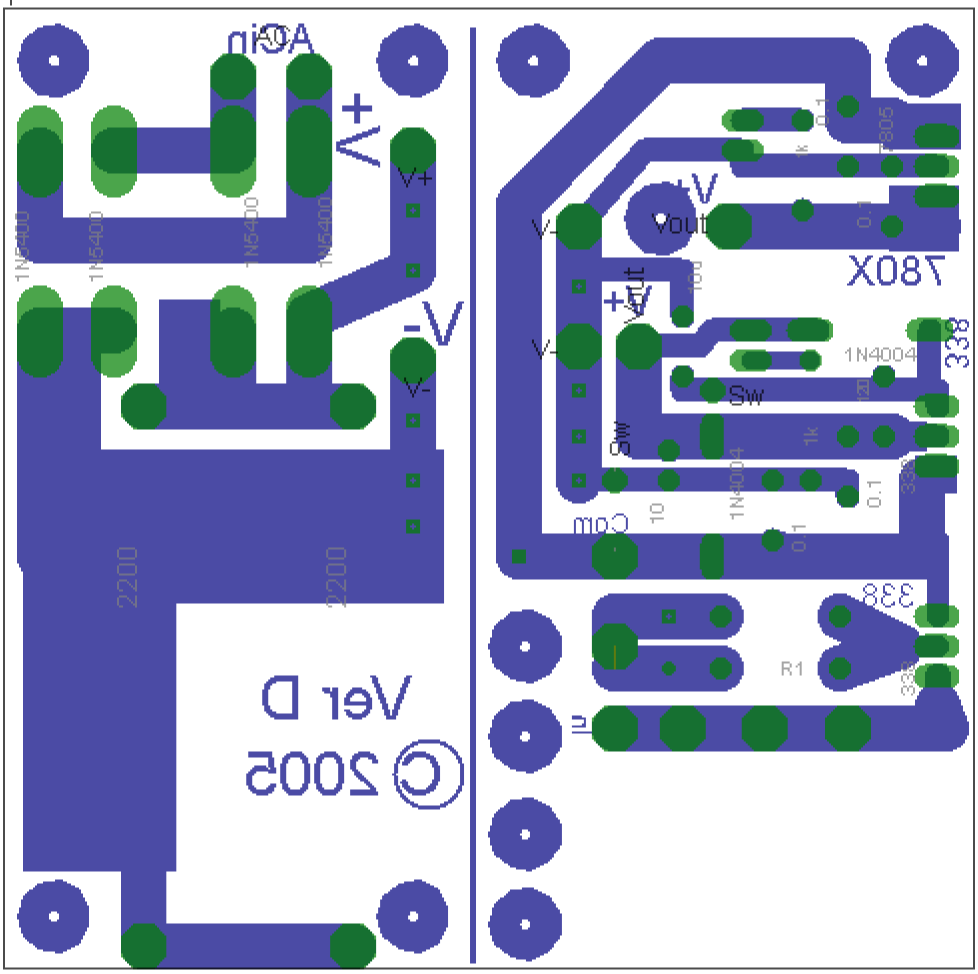
From the LM317 datsheet there is an application to build a current limit. The current can be controlled by using different values of resistor (a potentiometer could be fitted if it was a special high power one). Check out the datasheet for other applications for the LM317.



In this circuit below the current can be set using two values for R1 and R2 and a switch to select either or both (giving three different preset values)
If 1R2 ohms gives 1 amp limit
What value of R would give a current limit of 200mA?





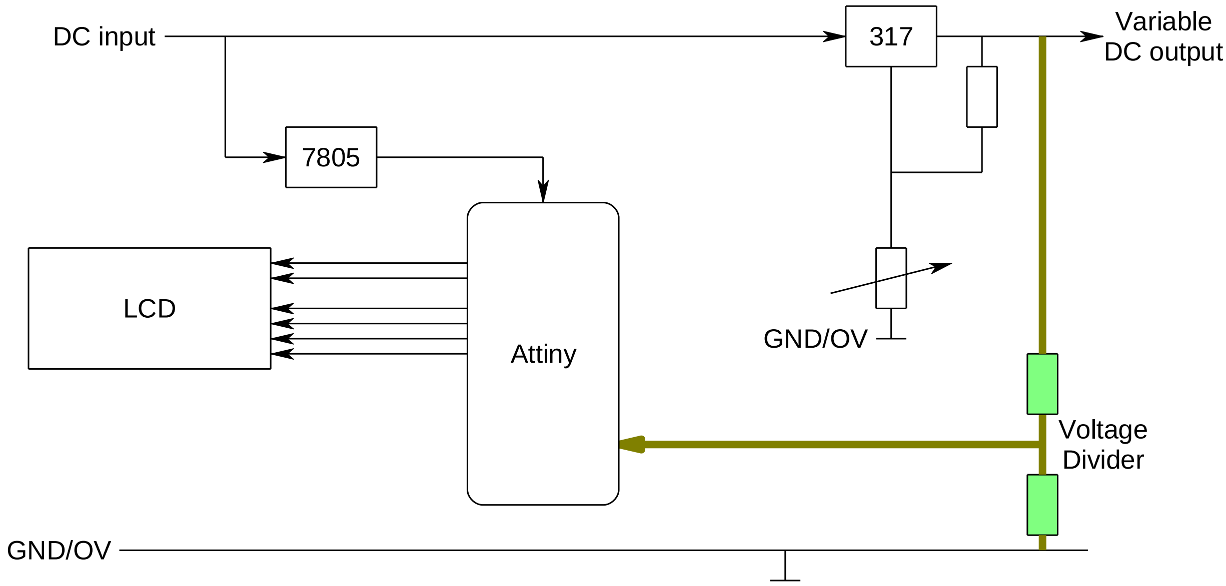


In this layout the 3 voltage regulators are mounted on the very edge of the PCB.

This means that we can solder them onto the PCB and then heatsink them easily against a large heatsink or a metal case.

26.16 Voltage measurement using a voltage divider

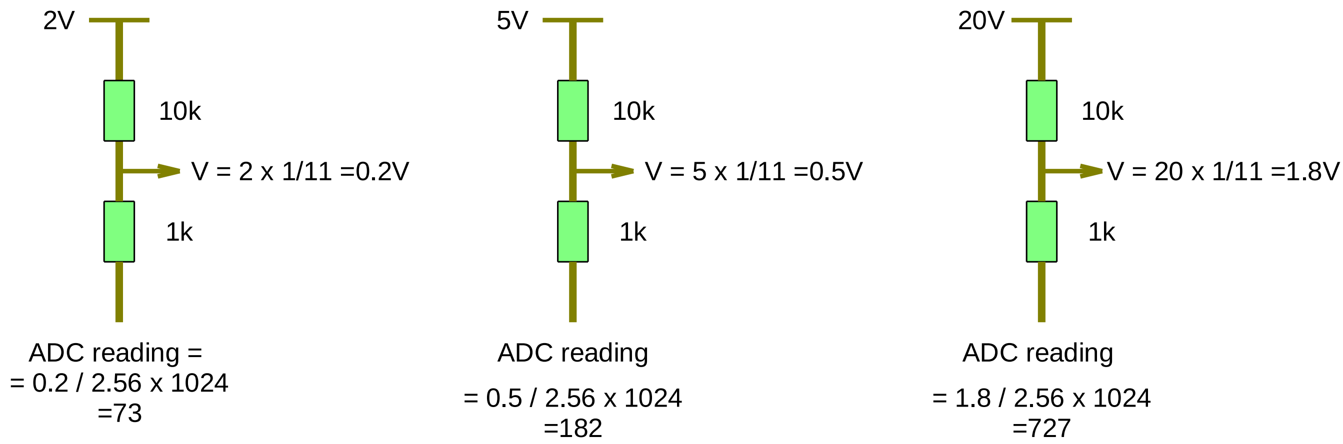
Having developed a variable power supply it is important to be able to measure the voltage it is set to. We can monitor the output of a power supply by reading the voltage with an ADC pin on the microcontroller and converting this to voltage display on the LCD.



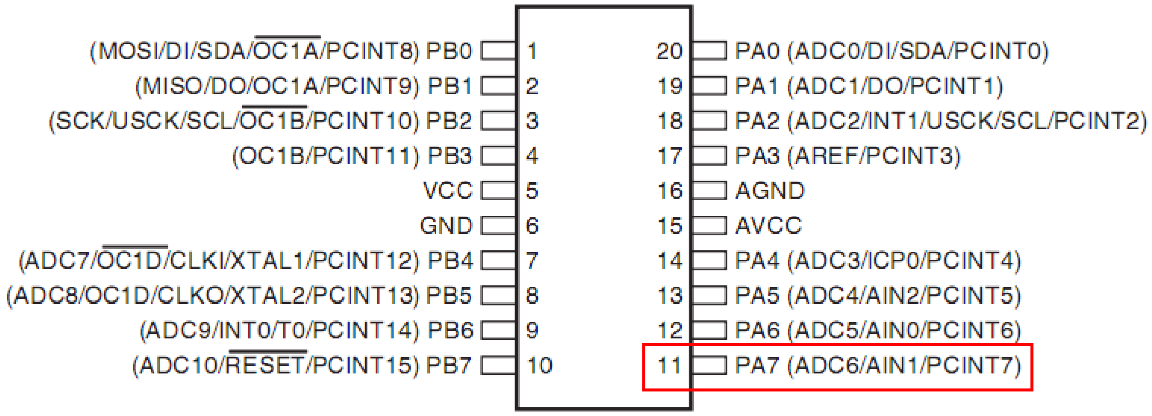
In the block diagram above the voltage divider divides the output voltage of the PSU down to a value within the range of the ATTiny26 ADC port and uses that to measure the voltage.

The AVR has an internal reference voltage we can use. It is 2.56 volts so you must make sure that the voltage into the ADC cannot exceed 2.56V so some ohms law and resistance calculations are necessary.

If the maximum voltage out of the PSU is 20V then a ratio of 10:1 for the resistors would be satisfactory
The following shows what the voltage (to 1d.p.) would be for 2V, 5V and 20V in along with the reading for the ADC.



We used the Attiny461-20PU for this project. ATMEL like to change models of its microcontrollers all the time, we don't mind this as each time they do they tend to get a little better for the same cost! However it does mean keeping up to date with the micros specifications. The ATTiny461 has 11 ADC inputs (although we cannot use ADC10 because it's the reset pin and we need ot for programming).

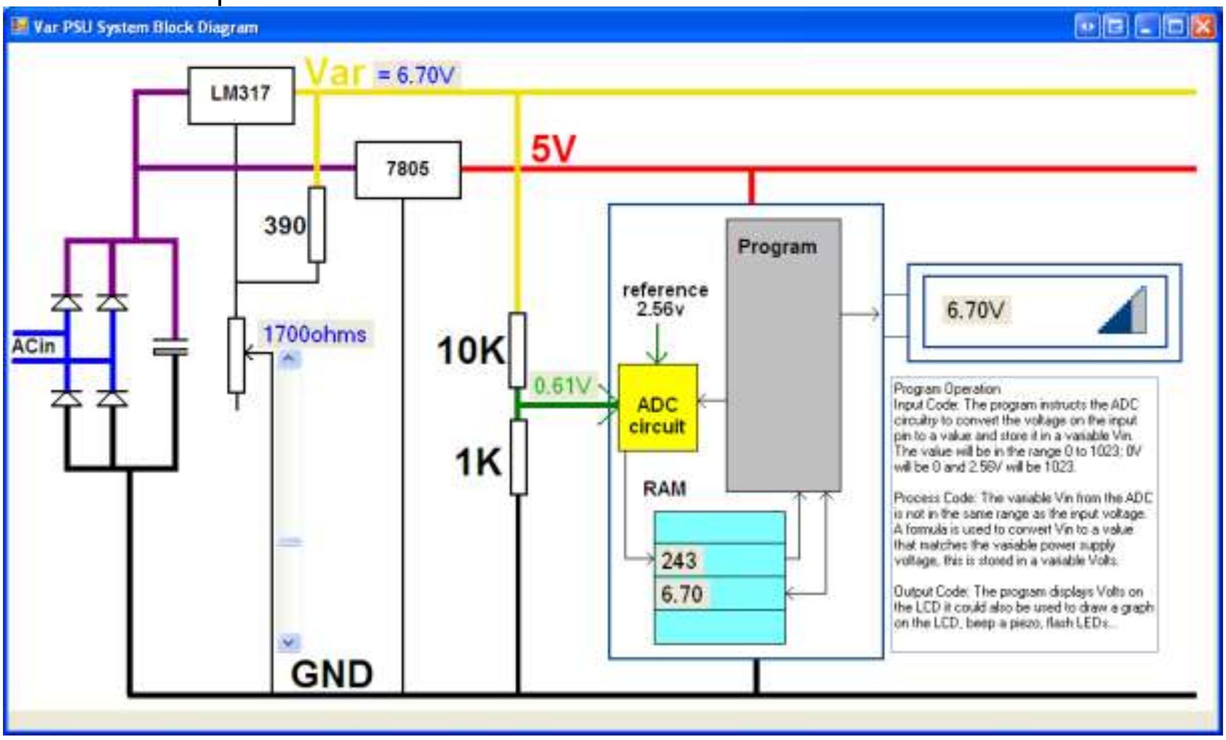


6.2 ATtiny461

Speed (MHz)[3]	Power Supply	Ordering Code[2]	Package[1]	Operational Range
10	1.8 - 5.5V	ATtiny461V-10MU ATtiny461V-10PU ATtiny461V-10SU	32M1-A 20P3 20S2	Industrial (-40 C to 85 C)
20	2.7 - 5.5V	ATtiny461-20MU ATtiny461-20PU ATtiny461-20SU	32M1-A 20P3 20S2	Industrial (-40 C to 85 C)

- **Non-volatile Program and Data Memories**
  - **2/4/8K Byte of In-System Programmable Program Memory Flash**
    - **Endurance: 10,000 Write/Erase Cycles**
  - **128/256/512 Bytes In-System Programmable EEPROM**
    - **Endurance: 100,000 Write/Erase Cycles**
  - **128/256/512 Bytes Internal SRAM**
  - **Data retention: 20 years at 85°C / 100 years at 25°C**

This computer program simulates the variable power supply, the action of the voltage divider and the conversion process within the microcontroller



26.17 Variable power supply voltmeter program

<table>
<tr><td><pre>'Title Block
'Name: B.Collis and Anka
'Date: May 2010
'File Name: Voltmeter.bas
'--
'Program Description:
'use ADC to read voltage from output of a voltage divider
'convert adc value to one that matches the voltage into
the voltage divider
'use an LCD to display value of the voltage
'--
'Compiler Directives
$crystal = 1000000   'speed of operations
inside the micro
$regfile = "attiny461.dat"   'the micro we are using</pre></td><td>This program was developed to display the voltage of the variable powersupply, Anka (year11) and I worked on it together, since then he has taken his program to a further stage to incorporate more features such as audible warnings and other visual warnings.</td></tr>
<tr><td><pre>'--
'Hardware Setups
Config Porta = Output
Config Pina.7 = Input</pre></td><td>Initially we configure all the pins on port A as outputs, however the voltage divider is connected to A.7 so it must be configured as an input.</td></tr>
<tr><td><pre>Config Lcdpin = Pin , Db4 = Portb.3 , Db5 = Portb.6 , Db6
= Portb.4 , Db7 = Portb.5 , E = Portb.2 , Rs = Portb.1
Config Lcd = 20 * 2   'configure lcd
connections</pre></td><td></td></tr>
<tr><td><pre>Config Adc = Single , Prescaler = Auto , Reference =
Internal_2.56_extcap
Start Adc</pre></td><td>The first line sets up the analogue to digital conversion circuits within the AVR. In terms of systems knowledge this is is an example of sub systems where students must be familiar with the I/O characteristics and function of a device but not the detail of its internal operation. The Attiny26 has 11(though we can only use 10) ADC inputs. AN ADC requires an input voltage and a reference voltage against which to compare the input voltage. It has different voltage references we can use, external, 1.11V or 2.56 internal. In this case we are using the internal 2.56 volt reference with a 0.1uF capacitor on AVCC (pin 15).   The ADC reading will be in the range of 0 to 1023, where a 0 means 0Volts and 1023 means the same as the reference voltage.</td></tr>
</table>

'initialise hardware **Cls** 'clears LCD display **Cursor Off** 'cursor not displayed	No need to display the cursor on the LCD
'--- 'Declare Constants 'Declare Variables **Dim** Adc_in **As Word** **Dim** Voltage **As Single** **Dim** Dividor **As Single** **Dim** Volts **As String** * 5 'Initialise Variable Dividor = 32.6255	Variables store data, here we need a variable to store the value we read from the ADC input. This must be a word sized variable as it may store up to 1023 (remember a byte can only store upto 255). We want to display decimals so we must use a single or a double, we do not need the precision of a double so we use the single. We want to display the number on the LCD as well. We could use the same variable voltage however it will give us loads of decimal places so we will convert it to a string and then format the string so we need a varibel that can hold a string.

'--
'Program starts here
Do
    Adc_in = **Getadc**(6)
    Voltage = Adc_in
    Voltage = Voltage / Dividor
    Volts = **Fusing**(voltage , "#.##")
    Locate 1 , 1
    Lcd Volts ; "V" ; "   "
Loop
 	
End
	1. Read the voltage into the word variable adc_in.
	2. Put this number into the single variable
	3.This number will not be the voltage but a number that changes in relation to the voltage so we must convert it into a number that is the same as the voltage.
	4. This will be a number with loads of decimal places so we conver it to a string
	5. the string is formatted to have only 2 decimal places.
	6. position the cursor
	7. display the string version of the voltage, the letter V and then a couple of blank spaces on the LCD.
	8. repeat the process all over again

27 Year11/12/13 typical test questions so far

Capacitors

What is the value of the small yellow Capacitor in the microcontroller circuit- in pF? nF? uF?
What is the number written on it and what does it mean?
Why is it used?
What does polarised mean?
What are the two ways of knowing how to put an electrolytic capacitor into the circuit correctly?

Resistors

Calculate the value for a current limit resistor with a 12V battery and an LED drawing 2mA
Select the closest value we have in class that you could use.
If you could use 2 values of resistor found in class combining them together which 2 would you use?
Explain what a voltage divider does
What do we use potentiometers in circuits for? Expalin how a potentiometer is a voltage divider

Multimeter use

You want to measure the current drawn by your LED in a microcontroller circuit, draw a diagram of how you would do it and what settings you would use on the multimeter.

Algorithms/Modelling

Why do we write algorithms before we program? (Do 2 of the following algorithms)
Write pseudo-code then draw a flowchart for a program to read 2 switches to control the position of an LCD character, one to move it left, one to move it right and press both to change line.
Write an algorithm to play as many different tones as possible if you have 4 switches and press them in different combinations
Write an algorithm to change the speed of a flashing led using 2 switches
Write an algorithm that uses 1 switch to enter the number of times an led will flash and a second switch to start the LED flashing
Write an algorithm to allow a user to enter their name into a variable, using 3 switches, the first to increase the litter, the second to move to the next letter, the third to finish.

Variables

If you were to record the position of a character on an LCD what type of variable would you use?
Describe overflow
If you were have a user enter their age what type of variable would you use?
If you were counting seconds in a minute what type of variable would you use? In an hour? In a day? In a year? In a century? Give good names for these variables.
Dimension variables that would hold each of your first, last and any middle names.

Programming

Write a short piece of code that counts 15 switch presses and then flashes an LED
Write a short piece of code that checks 4 switches to see if they are all pressed.
Write a subroutine to check if a value is a multiple of 10 and if it is to flash an led once
Write a subroutine to add three strings together with a space beweeen each string
Write a subroutine that gets the first character from each of three strings and displays it on the lcd
Write asubroutine to get the middle letter of a string and display it on the lcd
Write a subroutine to get a random letter from a string and display it on the lcd

Microcontollers

What are the different uses of the three microcontroller memory types:RAM, FLASH & EEPROM

Subsystems

Draw a system context diagram for your project

Draw a block diagram for your project
What does ‘black box mean’
What are at least 3 things about a 7805 that makes it so useful for a microcontroller circuit
Describe the inputs and outputs of an LCD,
Explain each of the main commands to use an LCD

28 Advanced programming -arrays

It is easy to dimension variables to store data, however what do you do when you want to store many similar variables e.g. 50 light level readings over a period of time.

Do you create 50 variables e.g. lightlevel1, lightlevel2, lightlevel3 .... lightlevel50 ?
The answer is no because it is so difficult to read and write to 50 different variables.

Think of the data we want to collect as in a table, each row is labelled with a number to identify the row – we call this an INDEX.

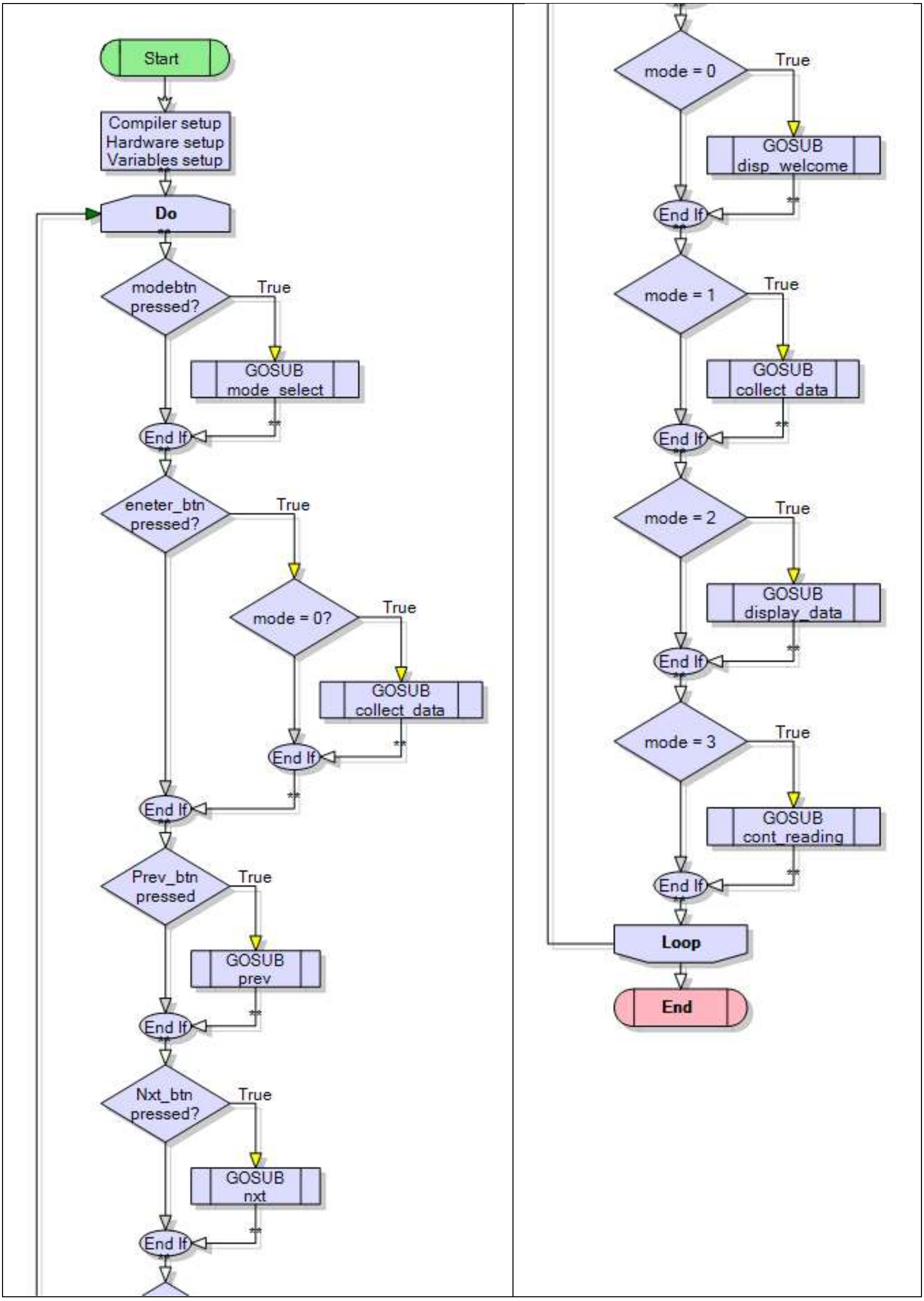
Index	lightlevel
1	345
2	267
3	378
4	120
5	203
.	.
.	.
49	432
50	198

An **ARRAY** type variable is dimensioned to store the data. Arrays are a highly important programming structure in computer science.

e.g **Dim lightlevel as byte(50)** this array becomes very easy to read and write using a loop. In Bascom the variable lightlevel(1) will be the first value and lightlevel(50) will be the last.

```
'get 50 values and store them in the array
For index=1 to 50
   lightlevel(index) = getadc(0)
   Waitms 50
Next

'read the 50 values from the array and display them
For index=1 to 50
   Locate 2,1
   Lcd lightlevel(index)
   Waitms 50
Next
```

In this next program a system has been developed that takes 50 lightlevel readings. The user can start the readings process and control the display of the readings on the LCD.
Note that the flowchart is split into 2 parts to allow for 1 page printing.
There are 8 if conditions, the first 4 read the 4 buttons, the second are carried out depending on the value of the variable MODE. All processing is within the subroutines.


Start
Compiler setup
Hardware setup
Variables setup
Do
modebtn pressed?
True
GOSUB mode_select
End If
eneter_btn pressed?
True
mode = 0?
True
GOSUB collect_data
End If
End If
Prev_btn pressed
True
GOSUB prev
End If
Nxt_btn pressed?
True
GOSUB nxt
End If
mode = 0
True
GOSUB disp_welcome
End If
mode = 1
True
GOSUB collect_data
End If
mode = 2
True
GOSUB display_data
End If
mode = 3
True
GOSUB cont_reading
End If
Loop
End

In this exercise you will need to make a small modification to the given program.

```
' File Name: arrayV1.bas
'  Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                  'the speed of the micro
$regfile = "m8535.dat"               'our micro, the ATMEGA8535-16PI
'------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                'LEDs on portA
Config Portb = Output                'LEDs on portB
Config Portc = Output                'LEDs on portC
Config Portd = Output                'LEDs on portD
'config inputs
Config Pina.0 = Input               ' ldr
Config Pind.2 = Input               'switch A
Config Pind.3 = Input               'switch B
Config Pind.6 = Input               'switch C
Config Pinb.1 = Input               'switch D
Config Pinb.0 = Input               'switch E
'LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E =
Portc.3 , Rs = Portc.2
Config Lcd = 40 * 2                 'configure lcd screen
'ADC
Config Adc = Single , Prescaler = Auto , Reference = Internal
Start Adc

' Hardware Aliases
Sw_a Alias Pind.6
Sw_b Alias Pind.3
Sw_c Alias Pind.2
Sw_d Alias Pinb.1
Sw_e Alias Pinb.0

' initialise ports so hardware starts correctly
Porta = &B11111100                  'turns off LEDs ignores ADC inputs
Portb = &B11111100                  'turns off LEDs ignores switches
Portc = &B11111111                  'turns off LEDs
Portd = &B10110011                  'turns off LEDs ignores switches
'------------------------------------------------------------------
'  Declare Variables
Dim Opmode As Byte
Dim Reading As Word
Dim Lightlevel(50) As Word
Dim index As Byte
Dim Reading_delay As Byte
Dim num_eadings  As Byte
'  Initialise Variables
Opmode = 0
num_eadings=50
'------------------------------------------------------------------
```

```
'  Program starts here
Cls                             'clear lcd screen
Do
  'read the switches
  Debounce Sw_a , 0 , Mode_select , Sub
  Debounce Sw_b , 0 , Enter_button , Sub
  Debounce Sw_c , 0 , Prev , Sub
  Debounce Sw_d , 0 , Nxt , Sub
  'choose what to do
  Select Case Opmode
    Case 0 : Gosub Display_welcome
    Case 1 : Gosub Collect_data
    Case 2 : Gosub Display_data
    Case 3 : Gosub Cont_reading
  End Select
Loop
End                             'end program
'--------------------------------------------------------------
' 13. Subroutines
Mode_select:
  Cls                           'when mode changes clear the lcd
   Incr Opmode
   If Opmode > 3 Then Opmode = 0
Return

Display_welcome:
  Locate 1 , 1
  Lcd " Data Collector "
  Lowerline
  Lcd "  version 1.0   "
Return

Enter_button:
  If Opmode = 1 Then Gosub Collect_data
Return

Collect_data:
    Locate 1 , 1
    Lcd " press enter to "
    Lowerline
    Lcd "start collection"
   Cls
   For index = 1 To num_eadings
      Reading = Getadc(0)             'read lightlevel
      Lightlevel(index) = Reading      ' store reading in array
      Locate 3 , 1
      Lcd index                   'display the index
      Locate 4 , 1
      Lcd Reading ; "    "          'diplay the reading
      Waitms Reading_delay
   Next
   Opmode = 0
Return
Display_data:
```

```
   Locate 1 , 1
   Lcd index ; " "
   Locate 2 , 1
   Lcd Lightlevel(index) ; "     "
Return

Cont_reading:
   Locate 1 , 1
   Lcd "continous readings"
   Locate 2 , 1
   Reading = Getadc(0)
   Lcd Reading ; "   "
Return

Prev:
   Decr index
   'fix this routine so that it doesn't underflow
Return

Nxt:
   Incr index
   'fix this routine so that it doesn't overflow
Return
```

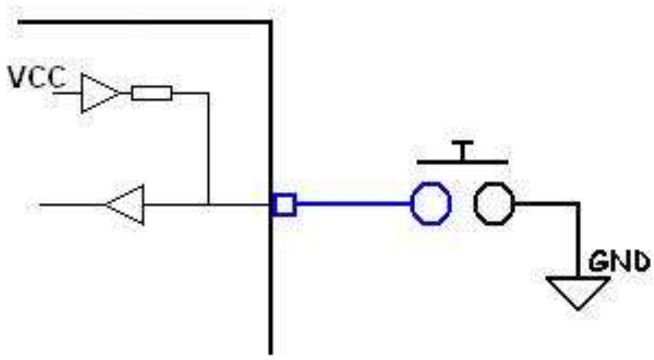
1. Fix the bugs with the prev and nxt routines so that they don't go below 0 or above 50.
2. can you modify the proram so that prev and nxt buttons change the timing of the reading, which mode would it be best to place the new code in?
3. can you modify the program so that the prev and nxt buttons change the number of readings to be stored.

29 AVR pull-up resistors

A useful thing to know about is that the AVRs have internal pullup resistors for use when you connect a switch to an input pin.

These can be activated from within software; this means you don't have to connect a separate resistor; however you still have to activate it.

Note that by default it is not activated.



Config Pind.2 = Input
Set portd.2 'activate internal pull-up

If pinb.2 = 0 then
…
…
…
end if

Why didn't you learn about this straight away, well its important to understand the concept of pullup resistors and by physically using them you gain a better understanding of them.

30 Advanced keypad interfacing

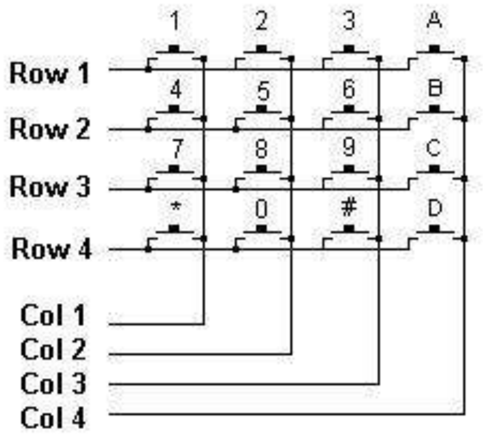
It is quite straightforward using Bascom to read a keypad, it handles all the hard work for us with the built in function Getkbd().

```
Config Kbd = Portb
Dim kbd_data As Byte
Kbd_data = Getkbd()  'keybdb returns a digit from 0 to 15
LCD kybd_data
```

The connection to the microcontroller is straightforward as well, just 8 pins. Solder headers into the 8 pins of the keypad and 8 pins as shown on the PCB

How do the 16 key keypad and the software work together?

The Keypad is
and column are
Software:
The micro sets the
ports. The columns are
any key is pressed
there is a 0 then it
inputs and columns as
rows it has a valid
to determine exactly



arranged in a matrix of 4x4 and each row
connected to the microcontroller.

rows as outputs and puts a low on those
set as inputs, it reads the columns and if
there will be a 0 on one of the columns. If
reverses the situation with the rows as
outputs and if there is a low on one of the
keypress. The combination of 0's is used
which key is pressed.

The code which is returned from getkbd() will not match the number on the keypad so a translation process is required. It is also better to have a subroutine handle this process and keep it away from your main code. Then this routine can be called from anywhere in the program.
In this code not only is the key translated but it is not returned untilt he user releases the button, this stops the key frombeing sensed multiple times.

30.1 Keypad program 1

```
'-----------------------------------------------------------------
' 1. Title Block
' Author:   B.Collis
' Date:     14 Aug 2003
' File Name: keypad_Ver1.bas
'develop a simple subroutine that translates key press codes into more recognisable key values.
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the crystal we are using
$regfile = "m8535.dat"             'the micro we are using
'-----------------------------------------------------------------
' 6. Hardware Setups
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs =
Portc.0
Config Lcd = 20 * 4               'configure lcd screen
Config Kbd = Portd
'8. initialise hardware
```

```
'-----------------------------------------------------------------
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Kbd_data As Byte
Dim Key As Byte
' 11. Initialise Variables
Key = 16
Cls                            'clears LCD display
Cursor On Noblink
'-----------------------------------------------------------------
' 12. Program starts here
Do
   Gosub Readkeypad
   Lcd Key                     '; " "
Loop
End                             'end program


'-----------------------------------------------------------------
Readkeypad:
'gets a key press and returns a key value 0 to 16
'16 is no key pressed
   Kbd_data = Getkbd()
   If Kbd_data < 16 Then
      Select Case Kbd_data
         Case 0 : Key = 1
         Case 1 : Key = 2
         Case 2 : Key = 3
         Case 3 : Key = 10        'A
         Case 4 : Key = 4
         Case 5 : Key = 5
         Case 6 : Key = 6
         Case 7 : Key = 11        'B
         Case 8 : Key = 7
         Case 9 : Key = 8
         Case 10 : Key = 9
         Case 11 : Key = 12       'C
         Case 12 : Key = 14       '*
         Case 13 : Key = 0
         Case 14 : Key = 15       '#
         Case 15 : Key = 13       'D
      End Select
   End If
Return
```

30.2 Keypad program 2

Generate text / ASCII rather than a numeric value

```
' Declare Variables
Dim Kbd_data As Byte
Dim Key As String * 2
' Initialise Variables
Key = " "
Cls                             'clears LCD display
Cursor On Noblink
'-----------------------------------------------------------------
' Program starts here
Do
   Gosub Readkeypad
   Lcd Key                        '; " "
Loop
End                             'end program
```

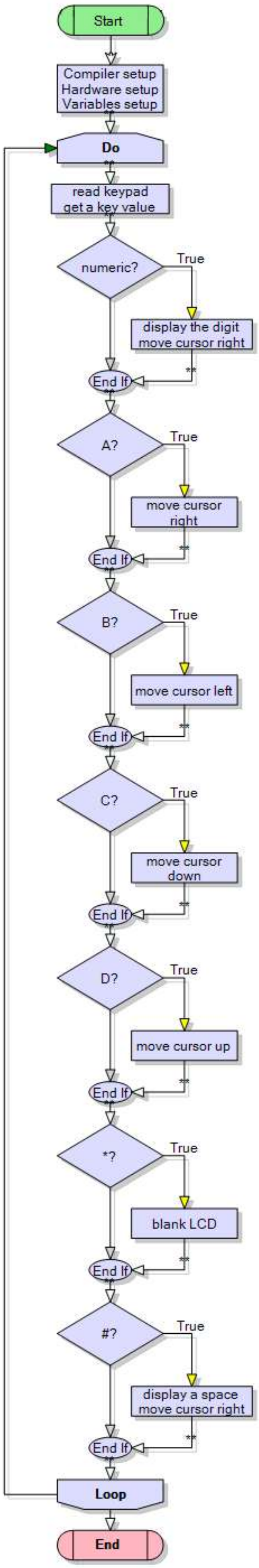
Changes to use a string

```
'-----------------------------------------------------------------
Readkeypad:
'gets a key press and returns a key value 0 to 16
'16 is no key pressed
   Kbd_data = Getkbd()
   If Kbd_data < 16 Then
      Select Case Kbd_data
         Case 0 : Key = "1"
         Case 1 : Key = "2"
         Case 2 : Key = "3"
         Case 3 : Key = "A"
         Case 4 : Key = "4"
         Case 5 : Key = "5"
         Case 6 : Key = "6"
         Case 7 : Key = "B"
         Case 8 : Key = "7"
         Case 9 : Key = "8"
         Case 10 : Key = "9"
         Case 11 : Key = "C"
         Case 12 : Key = "*"
         Case 13 : Key = "0"
         Case 14 : Key = "#"
         Case 15 : Key = "D"
      End Select
   End If
Return
```

This program however don't do anything much for us, they need a little more control to be useful

- Debounce the keys a little
- Only return the value once if a key is held down
- Use the other keys to do something different like move the cursor around the lcd

30.3 Keypad program 3 – cursor control



The really big concepts to understand here are 1. cursor control and 2. that numbers on an LCD are not data.

1. A cursor is a flashing or steady line on a screen to show you where the next text will be entered. If you want text to appear in certain places on an LCD (or any screen) you must control it within your program, the LCD itself has very limited cursor control.

Often with LCDs there appears to be no cursor, as it is not turned on. The cursor however is still there; just invisible. When text is sent to the display it will appear at the cursor location and the LCD will move its cursor one space to the right. In simple programs as with the above two the microcontroller has no idea where the cursor is, it just gives the LCD data to display.

If you want text to appear in a certain location on the screen then you have to move the cursor with Bascom's LOCATE function.

In a complex program you may want to move the text around the screen at will, so you do this by moving the cursor first and then sending data to the display. In this case you need to keep track of the cursor location yourself by using some variables, as in this next program.

2. Data is in your program. In this program data is collected from a keypad and stored in a variable. Then this data is put onto the LCD, these are two separate and different control processes. Don't mix them up, when programming keep them within separate sub routines.

```
' Declare Variables
Dim I As Byte
Dim Cursor_x As Byte
Dim Cursor_y As Byte
Dim Kbd_data As Byte
Dim Key As Byte
' Initialise Variables
I = 0
Cursor_x = 1
Cursor_y = 1
Key = 16                         'nothing to process to start with
Cls                              'clears LCD display
Cursor Noblink                   'steady cursor
'-----------------------------------------------------------------
' Program starts here
Locate Cursor_y , Cursor_x       'move lcd cursor to top left corner of LCD
Do
   Gosub Read_1_keypress         'get a single key press
   Gosub Disp_char               'display char and move cursor
Loop
End                              'end program

Disp_char:
'displays numbers on lcd
'uses A,B,C,D to move the cursor , * to clear the screen, # to insert space
'the use of key=16 is so that the key is sensed only once per press
```

Cursor control variables

```
'cursor control is one of the big concepts here.
   Select Case Key
   Case Is < 10:                                         'number
      Lcd Key
      Incr Cursor_x
      If Cursor_x > 20 Then Cursor_x = 1                 'on overflow wrap to left
      Locate Cursor_y , Cursor_x                        'position the cursor
      Key = 16                                          'key processed
   Case 10:                                              'A = go right
      Incr Cursor_x
      If Cursor_x > 20 Then Cursor_x = 1                 'on overflow wrap to left
      Locate Cursor_y , Cursor_x
      Key = 16                                           'key processed
   Case 11:                                              'B = go left
      Decr Cursor_x
      If Cursor_x = 0 Then Cursor_x = 20                 'on underflow wrap to right
      Locate Cursor_y , Cursor_x
      Key = 16                                          'key processed
   Case 12 :                                            'C = go down
      Incr Cursor_y
      If Cursor_y > 4 Then Cursor_y = 1                 'on overflow wrap to top
      Locate Cursor_y , Cursor_x
      Key = 16                                           'key processed
   Case 13 :                                             'D = go up
      Decr Cursor_y
      If Cursor_y = 0 Then Cursor_y = 4                 'on underflow wrap to bottom
      Locate Cursor_y , Cursor_x
      Key = 16                                          'key processed
   Case 14 :                                             '* = clear screen
      Cls
      Cursor_x = 1
      Cursor_y = 1
      Key = 16                                          'key processed
   Case 15 :                                             '# = clear screen
      Lcd " "
      Incr Cursor_x
      If Cursor_x > 20 Then Cursor_x = 1                 'on overflow wrap to left
      Locate Cursor_y , Cursor_x
      Key = 16                                          'key processed
   End Select
Return
```

```
'-------------------------------------------------------------
Read_1_keypress:
'gets a key press and returns a key value 0 to 16
'16 is no key pressed
   Kbd_data = Getkbd()
   If Kbd_data < 16 Then
      Select Case Kbd_data
         Case 0 : Key = 1
         Case 1 : Key = 2
         Case 2 : Key = 3
         Case 3 : Key = 10           'A
         Case 4 : Key = 4
         Case 5 : Key = 5
         Case 6 : Key = 6
         Case 7 : Key = 11           'B
         Case 8 : Key = 7
         Case 9 : Key = 8
         Case 10 : Key = 9
         Case 11 : Key = 12          'C
         Case 12 : Key = 14          '*
         Case 13 : Key = 0
         Case 14 : Key = 15          '#
         Case 15 : Key = 13          'D
         'Case 16 : Key = 16          'nothing pressed
      End Select
   End If
   'wait until the user releases the key
   Do
      Kbd_data = Getkbd()
   Loop Until Kbd_data = 16
   'by experimentation, it was realised that a small debounce
   'delay made this routine stable
   Waitms 5
Return
```

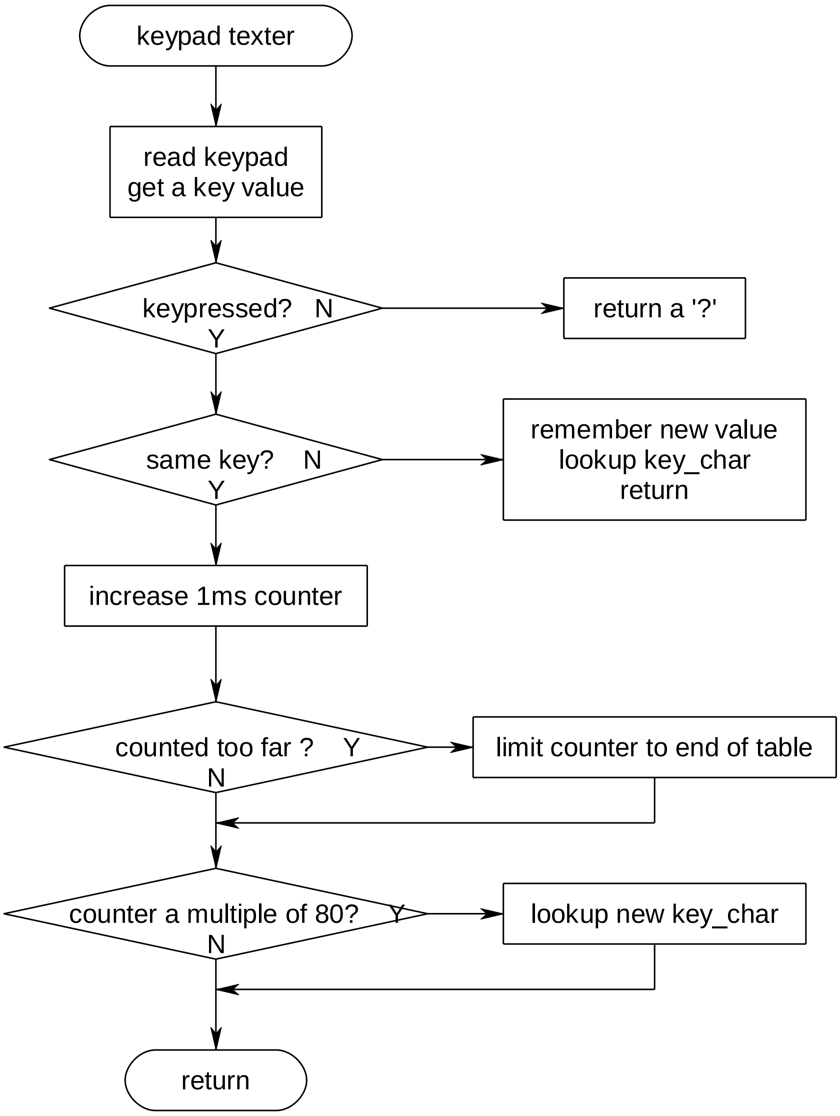
Routines like this are useful where the user has to enter data into the program and you want it on the display as well.

Remember the two concepts

1. Cursor control
2. Reading data and displaying data are two separate things

30.4 Keypad texter program V1

In this program we want to get text froma keypad. It will operate so that when the button is held down it will scroll through the text on the key pad as well. e.g. holding down 6, will initially return ‘6’ then after 80ms ‘M’, then after 80ms ‘N’, then after 80ms ‘O’, then after 80ms ‘m’, then after 80ms ‘’n then after 80ms ‘o’.



So we start a counter (and every 1ms increase it)
The routine exits but everytime it returns it increase count

From 0 to 79 the routine returns ‘6’.

From 81 to 160 it returns ‘M and so on

```
'-----------------------------------------------------------------
' Title Block
' Author:B.Collis
' Date: Aug09
' Version: 1.0
' File Name: keypad_texterV1.bas
'-----------------------------------------------------------------
' Program Description:
' This program reads a keypad for digits and letters (both small & caps)
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                        ' internal clock
$regfile = "m8535.dat"
'-----------------------------------------------------------------
' Hardware Setups
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs =
Portc.0
Config Lcd = 20 * 4                       'configure lcd screen
```

```
Config Kbd = Portd
'-----------------------------------------------------------------
' Declare Constants
Const Key_repeatdelay = 50
Const Key_debouncedelay = 20
Const Key_repeat1 = 80
Const Key_repeat2 = 160
Const Key_repeat3 = 240
Const Key_repeat4 = 320
Const Key_repeat5 = 400
Const Key_repeat6 = 480
Const Key_repeat7 = 560
Const Key_repeat8 = 640

' Declare Variables
Dim Kbd_data As Byte
Dim Key As Byte
Dim Oldkey As Byte
Dim Lookupval As Byte
Dim Key_counter As Word
Dim Key_char As String * 2
' Initialise Variables
Key_counter = 0



'-----------------------------------------------------------------
' Program starts here
Cls
Cursor Off
Do
   Gosub Read_keychar
   If Key_char <> "?" Then
      Locate 1 , 5
      Lcd Key_char ; "  "
   End If
Loop
End                                         'end program
```

```
'----------------------------------------------------------------
' Subroutines
Read_keychar:
   Kbd_data = Getkbd()                                   'read a key
   Key = Kbd_data                                        'store the keypress
   If Kbd_data = 16 Then                                'no key pressed
      Oldkey = 16                                        'remember no key pressed
      Lookupval = 144                                    'return '?'
      Key_char = Lookupstr(lookupval , Chrcodes)
      Return                                             'exit the subroutine
   End If
  If Key = Oldkey Then                                  'key still pressed
      Waitms 1
      Incr Key_counter
      Select Case Key_counter
         Case Key_repeat1 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat2 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat3 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat4 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat5 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat6 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat7 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
         Case Key_repeat8 :
            Lookupval = Lookupval + 16
            Key_char = Lookupstr(lookupval , Chrcodes)
      End Select
      If Key_counter > Key_repeat8 Then Key_counter = Key_repeat8
   Else                                                 'new keypress
      Oldkey = Key
      Lookupval = Key
      Key_counter = 0
      Key_char = Lookupstr(lookupval , Chrcodes)
   End If
Return
```

```
'----------------------------------------------------------------
Chrcodes:
Data "1" , "2" , "3" , "A" , "4" , "5" , "6" , "B" ,
Data "7" , "8" , "9" , "C" , "*" , "0" , "#" , "D" ,
'2nd press
Data "1" , "A" , "D" , "A" , "G" , "J" , "M" , "B" ,
Data "P" , "T" , "W" , "C" , "*" , "C" , "#" , "D" ,
'3rd press
Data "1" , "B" , "E" , "A" , "H" , "K" , "N" , "B" ,
Data "Q" , "U" , "X" , "C" , "*" , "L" , "#" , "D" ,
'4th press
Data "1" , "C" , "F" , "A" , "I" , "L" , "O" , "B" ,
Data "R" , "V" , "Y" , "C" , "*" , "S" , "#" , "D" ,
'5th press
Data "1" , "a" , "d" , "A" , "g" , "j" , "m" , "B" ,
Data "S" , "t" , "Z" , "C" , "*" , "d" , "#" , "D" ,
'6th press
Data "1" , "b" , "e" , "A" , "h" , "k" , "n" , "B" ,
Data "p" , "u" , "w" , "C" , "*" , "a" , "#" , "D" ,
'7th press
Data "1" , "c" , "f" , "A" , "i" , "l" , "o" , "B" ,
Data "q" , "v" , "x" , "C" , "*" , "M" , "#" , "D" ,
'8th press
Data "1" , "c" , "f" , "A" , "i" , "l" , "o" , "B" ,
Data "r" , "v" , "y" , "C" , "*" , "A" , "#" , "D" ,
'9th press
Data "1" , "c" , "f" , "A" , "i" , "l" , "o" , "B" ,
Data "s" , "v" , "z" , "C" , "*" , "N" , "#" , "D" , "?"
'keypad layout and codes
'1    2    3    A
'4    5    6    B
'7    8    9    C
'*    0    #    D
```

This program works however there is some repetition in it with the lookups so that there is the opportunity for it to be rewritten as per the next page

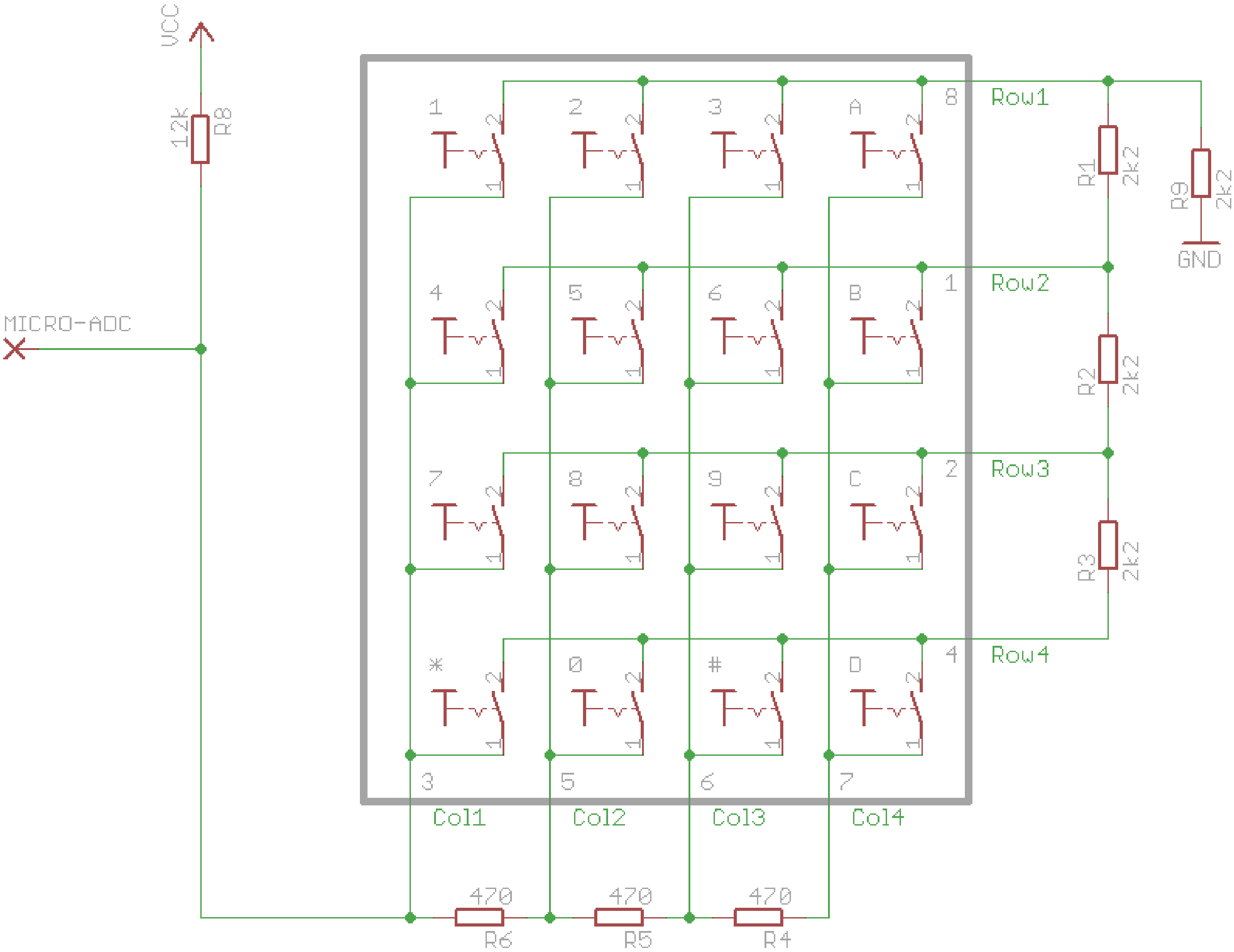
30.5 Keypad texter program 1a

This version of the program instead of having a lot of repeating code does some maths to work out the multiple of 80 and uses that to lookup the key character.

```
' new constants to replace all the old ones
Const Key_repeatdelay = 80
' ADD ONE NEW VARIABLE TO THE OTHERS ABOVE
Dim I As Word



' Subroutine
Read_keychar:
   Kbd_data = Getkbd()                              'read a key
   Key = Kbd_data                                    'store the keypress
   If Kbd_data = 16 Then                            'no key pressed
      Oldkey = 16                                   'remember no key pressed
      Lookupval = 144                                'return '?'
      Key_char = Lookupstr(lookupval , Chrcodes)
      Return                                         'exit the subroutine
   End If
   If Key = Oldkey Then                              'same key still pressed
      Waitms 1
      Incr Key_counter                               'count in 1ms increments
      I = Key_repeatdelay * 8                        'check we havent gone too far
      If Key_counter > I Then Key_counter = I        'so we dont overflow end of table
      I = Key_counter Mod Key_repeatdelay            'MOD means get remainder
      If I = 0 Then                                  '0 means it is a multiple of 80
         I = Key_counter / Key_repeatdelay           'how many multiples of 80
         Lookupval = I * 16                          'get char from table
         Lookupval = Lookupval + Kbd_data
         Key_char = Lookupstr(lookupval , Chrcodes)
      End If
   Else                                             'new keypress
      Oldkey = Key                                   'remember key press
      Lookupval = Key
      Key_counter = 0                               'start counting again
      Key_char = Lookupstr(lookupval , Chrcodes)    'get char from table
   End If
Return
```

30.6 ADC keypad interface

A 16 button keypad is a really nice feature for our projects but generally it requires 8 lines to connect it to a microcontroller; and sometimes we just don’t have these available as we have used them all up.
In this voltage divider circuit whenever a key is pressed the voltage to the microcontroller changes and can be sensed using a single ADC input.



This program reads the ADC value and displays both it and a value representing which key is pressed on the LCD. The values of resistor chosen in the above schematic allow a range of values from 0-2V, so we will use the internal reference voltage rather than the VCC voltage as comparison value for our ADC converter. NOTE YOU MUST NOT HAVE AREF PIN CONNECTED ON THE MICRO WHEN USING THE INTERNAL VOLRAGE REFERENCE!!

```
'------------------------------------------------------------------
'Title Block
' Author:    B.Collis
' Date:      July 2010
' File Name: keypad1ioLine.bas
'------------------------------------------------------------------
' Program Description:
' Hardware Features:
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' keypad connected as per R4R circuit on 1 ADC line
' lm35 on adc
```

```
' AREF PIN32 disconnected - uses internal 2.56V reference
'-------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                        'the crystal we are using
$regfile = "m32def.dat"                   'the micro we are using
'-------------------------------------------------------------------
'Hardware Setups
Config Porta = Input                      '
Config Adc = Single , Prescaler = Auto , Reference = Internal
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 20 * 4                       'configure lcd screen
'Harware Aliases
Kp Alias 1
Lm35 Alias 0
Led0 Alias Portc.0
Led1 Alias Portc.1

'------------------------------------------------------------------
'Declare Constants
Const Timedelay = 150

'-------------------------------------------------------------------
'Declare Variables
Dim Keypress As Word
Dim Key As Byte
Dim Tempr As Word
'Initialise Variables
Key = 16                                  'no press

'-------------------------------------------------------------------
'Program starts here
Cls                                       'clears LCD display
Cursor Off                                'no cursor
Lcd "ADC Keypad tester"
Do
    Keypress = Getadc(kp)
    Locate 2 , 1
    Lcd Keypress ; "    "
    If Keypress < 955 Then
      Gosub Lookupkey
      Lcd Key ; " "
    End If
    Tempr = Getadc(lm35)
    Tempr = Tempr / 2
    Locate 3 , 2
    Lcd Tempr ; "  "
    Waitms 100
Loop
End                                       'end program
```

```
'------------------------------------------------------------------
'Subroutines
Lookupkey:
   Select Case Keypress
      Case 290 To 340 : Key = 1
      Case 341 To 394 : Key = 2
      Case 395 To 443 : Key = 3
      Case 444 To 505 : Key = 10
      Case 506 To 563 : Key = 4
      Case 564 To 603 : Key = 5
      Case 604 To 640 : Key = 6
      Case 641 To 688 : Key = 11
      Case 689 To 734 : Key = 7
      Case 735 To 765 : Key = 8
      Case 766 To 795 : Key = 9
      Case 796 To 832 : Key = 12
      Case 833 To 868 : Key = 14
      Case 869 To 894 : Key = 0
      Case 895 To 917 : Key = 15
      Case 918 To 940 : Key = 13
      Case Else : Key = 16
   End Select

Return
'------------------------------------------------------------------
'Interrupts
```

31 Do-Loop & While-Wend subtleties

Learning to keep things under control by understanding what happens with loops

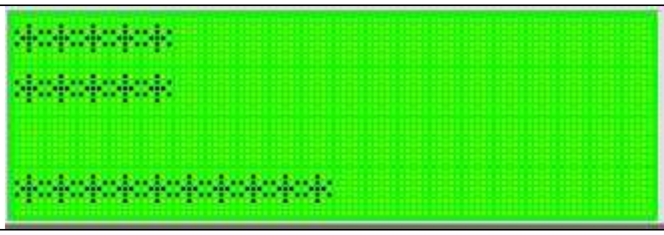
```
$sim                          'copy this code into Bascom and run it in the simulator
$crystal = 8000000
$regfile = "m8535.dat"
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
Cls
Cursor Off
Const Timedelay = 150
Dim Count As Byte
```

Locate 1 , 1
Count = 0
While Count < 5
   Incr Count
   Lcd "*"
Wend
	Prints 5 *****
	even though the count never gets to 5

Locate 2 , 1
Count = 0
Do
  Incr Count
  Lcd "*"
Loop Until Count =5
	Prints 5 *****
	Count must get to 5 for the output to be 5 asterisks

Locate 3 , 1
Count = 5
While Count < 5
   Incr Count
   Lcd "*"
Wend
	Does not print anything
	A while wend might not execute

Locate 4 , 1
Count = 5
Do
  Incr Count
  Lcd "*"
Loop Until Count = 5
	Gets stuck and continues to print ******
	A do loop will always execute at least once
	So in this case it executes the first time and increases count to 6 and then just keeps going

Output of the above code	

It is essential when programming to test your code and when you have loops getting out of control look for tests that might be wrong

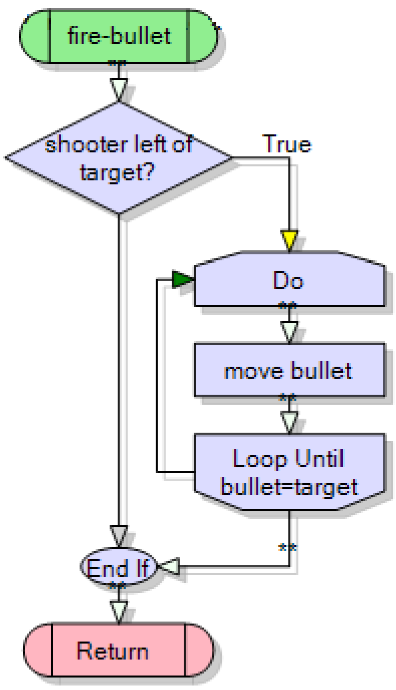
31.1 While-Wend or Do-Loop-Until or For-Next?

When you want something to repeat there are different ways to do it Here are a number of different ways to do the same thing. The program puts a shooter and a target on an LCD and fires bullets if the shooter is to the left of the target. The differences however are subtle and require careful testing of the routines to expose the clearest and best functioning

The first 2 use the do-loop-until, then the next 3 use while-wend and the last uses a for-next

```
'------------------------------------------------------------------
' 1. Title Block
' Author:     B.Collis
' Date:       21 April 2005
' File Name:  shoot_v1.bas
'------------------------------------------------------------------
' 2. Program Description:
'    Program moves a bullet across the lcd display
'  Hardware Features:
'    LCD
'  Program Features
'------------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m8535.dat"              'our micro, the ATMEGA8535-16PI
$crystal = 8000000                  'the speed of the micro
'------------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Portc = Output               'LCD on portC
Config Portd = Output
'LCD    redefine these for your LCD connection
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
'LCD special characters
Deflcdchar 0 , 8 , 20 , 11 , 30 , 8 , 8 , 20 , 20            ' shooter
Deflcdchar 1 , 32 , 32 , 16 , 32 , 32 , 32 , 32 , 32         ' bullet
Deflcdchar 2 , 2 , 7 , 18 , 15 , 2 , 2 , 5 , 5               ' target
Deflcdchar 3 , 32 , 4 , 16 , 32 , 2 , 8 , 14 , 31            'dyingman
Deflcdchar 4 , 32 , 32 , 32 , 32 , 32 , 6 , 14 , 31          ' deadman
' 5. Hardware Aliases

' 6. initialise ports so hardware starts correctly
Cls
Cursor Off
'------------------------------------------------------------------
' 7. Declare Variables
Dim Bullet_pos As Byte
Dim Shooter_pos As Byte
Dim Target_pos As Byte

' 8. Initialise Variables
Shooter_pos = 1
Target_pos = 20
```

```
'-------------------------------------------------------------
' 9. Declare Constants and program aliases
Const Bullet_speed = 600
Const Deathroll = 300
Const Bullet = 1
Const shooter = 0
Const Target = 2
Const Dyingman = 3
Const Deadman2 = 4

'-------------------------------------------------------------
' 10. Program starts here

Do
   Lcd "shooter"
   'test program for bullet routine
   Shooter_pos = Rnd(20)              'get a random position (0 to 19)
   Incr Shooter_pos                   'get a random position (1 to 20)
   Target_pos = Rnd(20)               'get a random position (0 to 19)
   Incr Target_pos                    'get a random position (1 to 20)
   Locate 3 , 1
   Lcd "S=" ; Shooter_pos ; " "
   Locate 4 , 1
   Lcd "T=" ; Target_pos ; " "
   Locate 2 , Shooter_pos
   Lcd Chr(shooter)                   'man with gun
   Locate 2 , Target_pos
   Lcd Chr(target)                    'target man
   Gosub Fire_bullet_do_v1            ‘replace with alternative routines
   Wait 3
   Cls                                'use cls carefully in programs
                                      ' or the LCDs can flicker
Loop
End                                   'end program
```



Here is a flowchart for the fire_bullet routine, on the next pages are different implementations of it and explanations of their problems

<table>
<tr>
<td>

```
Fire_bullet_do_v1:                      '1336 bytes
'this routine moves a bullet across the display
  If Target_pos > Shooter_pos Then     'shooter is left of target
    Bullet_pos = Shooter_pos           'start at the shooter position
    Do                                 'not hit yet
      Incr Bullet_pos                  'increase first
      Locate 2 , Bullet_pos            'draw bullet
      Lcd Chr(bullet)
      Waitms Bullet_speed
      Locate 2 , Bullet_pos            'blank the bullet
      Lcd " "
    Loop Until Bullet_pos = Target_pos
    Locate 2 , Target_pos
    Lcd Chr(dyingman)
    Waitms Deathroll
    Locate 2 , Target_pos
    Lcd Chr(deadman2)
  End If
Return
```

</td>
<td>Using the do-loop this way resulted in the programming taking up 1336 bytes in flash making it the shortest version.

However it has a subtle problem. When the bullet reaches the target it first replaces the target then there is a delay and then the dying man image appears. Using a high value for bulletspeed allows you to see the problem happen.</td>
</tr>
<tr>
<td>

```
Fire_bullet_do_v2:                    '1343 bytes
'this routine moves a bullet across the display
  If Target_pos > Shooter_pos Then     'shooter is left of target
    Bullet_pos = Shooter_pos + 1       'start in next lcd segment
    Do        'not hit yet
      Locate 2 , Bullet_pos
      Lcd Chr(bullet)                   'draw bullet
      Waitms Bullet_speed
      Locate 2 , Bullet_pos             'blank the bullet
      Lcd " "
      Incr Bullet_pos                   'increase after
    Loop Until Bullet_pos >= Target_pos       'check if gone past
    Locate 2 , Target_pos
    Lcd Chr(dyingman)
    Waitms Deathroll
    Locate 2 , Target_pos
    Lcd Chr(deadman2)
  End If
Return
```

</td>
<td>This code implements the bullet hitting the target properly as the last bullet appears in the space before the target and then after the bulletspeed delay the target becomes the dying man. To do this the code had to be changed. Note the changes in the lines in bold that are different or in different locations to the previous routine.</td>
</tr>
</table>

<pre>Fire_bullet_while_v1:   '1344 bytes
'this routine moves a bullet across the display
  If Target_pos > Shooter_pos Then   ' shooter is left of target
    Bullet_pos = Shooter_pos + 1   'start in next lcd segment
    While Bullet_pos < Target_pos   'not hit yet
      Locate 2 , Bullet_pos
      Lcd Chr(bullet)   'draw bullet
      Waitms Bullet_speed
      Locate 2 , Bullet_pos   'blank the bullet
      Lcd " "
      Incr Bullet_pos
    Wend
    Locate 2 , Target_pos
    Lcd Chr(dyingman)
    Waitms Deathroll
    Locate 2 , Target_pos
    Lcd Chr(deadman2)
  End If
Return</pre>
	This code segment uses the while-wend. Even though it is longer than the above code when compiled it correctly implements the final bullet not hitting the target.

<pre>Fire_bullet_while_v2:   '1342 bytes
'this routine moves a bullet across the display
  Bullet_pos = Shooter_pos + 1   'start in next lcd segment
  While Bullet_pos <= Target_pos   'not hit yet
    Locate 2 , Bullet_pos
    Lcd Chr(bullet)   'bullet
    Waitms Bullet_speed
    Locate 2 , Bullet_pos   'blank the bullet
    Lcd " "
    If Bullet_pos = Target_pos Then
      Locate 2 , Target_pos
      Lcd Chr(dyingman)
      Waitms Deathroll
      Locate 2 , Target_pos
      Lcd Chr(deadman2)
    End If
    Incr Bullet_pos
  Wend
Return</pre>
	In this subroutine the initial if-then statement that checks the relative positions of the shooter and targets is removed in an attempt to streamline the code. However it is not quite as efficient code as the first. When the target is left of the shooter 2 lines of code are executed, first the bullet pos is calculated and then the position is checked.
	It also reintroduces the same problem as first do-loop with the bullet replacing the target.

<pre>Fire_bullet_while_v3:   '1340 bytes
'this routine moves a bullet across the display
  Bullet_pos = Shooter_pos + 1   'start in next segment
  While Bullet_pos < Target_pos   'not hit yet
    Locate 2 , Bullet_pos
    Lcd Chr(bullet)   'draw bullet
    Waitms Bullet_speed
    Locate 2 , Bullet_pos   'blank the bullet
    Lcd " "
    Incr Bullet_pos
  Wend
  If Bullet_pos = Target_pos Then   'hit
    Locate 2 , Target_pos
    Lcd Chr(dyingman)
    Waitms Deathroll
    Locate 2 , Target_pos
    Lcd Chr(deadman2)
  End If
Return</pre>
	This code executes correctly however it is also inefficient. If the target is left of the shooter three lines of code are executed. Bullet_pos is calculated, the while is checked and the if is checked. It is really untidy code as it tries to separate the 2 ideas which are integrated together in the flowchart by separating the while and if parts, These 2 ideas are importantly linked together. This can lead to real big problems as changing one of them has consequences on the other.

<pre>Fire_bullet_for: '1352 bytes
'this routine moves a bullet across the display
  If Target_pos > Shooter_pos Then   ' shooter is left of
target
  Incr Shooter_pos   'start in next segment
of lcd
    For Bullet_pos = Shooter_pos To Target_pos
      Locate 2 , Bullet_pos
      Lcd Chr(bullet)   'draw bullet
      Waitms Bullet_speed
      Locate 2 , Bullet_pos   'blank the bullet
      Lcd " "
    Next
    Locate 2 , Target_pos
    Lcd Chr(dyingman)
    Waitms Deathroll
    Locate 2 , Target_pos
    Lcd Chr(deadman2)
  End If
Return</pre>
	This also has the problem of the bullet replacing the target. It is really bad programming practice though as the variable shooter_pos had to be increased for the code to work. It is poor programming practice to alter a variable you don't need to. If you use the variable shooter_pos elsewhere in your program then it could have disastrous effects. This also compiled into the longest code

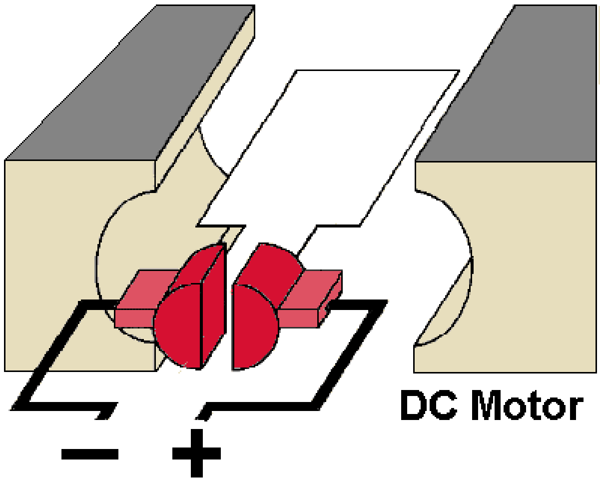
The best of these is the first while loop, it is the easiest to follow and works correctly.
Lessons:

- Get to know the three looping methids
- TEST TEST TEST your code carefully and methodically to identify correct operation
- When changing code retest it thoroughly for introduced errors
- Avoid changing variables you shouldn't change
- Keep records of your experiments to get the best possible grades

32 DC Motor interfacing

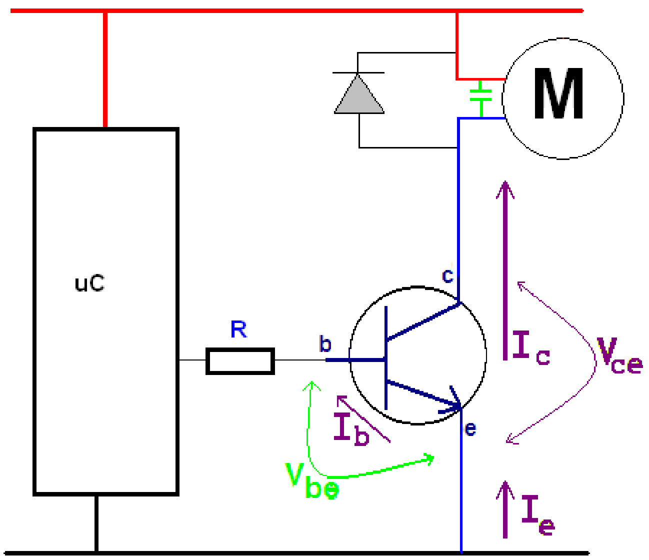
Nowadays who doesn’t want to see motor attached to a microcontroller moving something around! But to do this a bit of knowledge and understanding is required first, some of which is important physics knowledge.

A dc motor is made from a coil of wire, a magnet, a battery, brushes and a commutator (rotary switch). There is a neat video on youtube http://www.youtube.com/watch?v=zOdboRYf1hM of a simple motor and another one that demonstrates the importance of the commutator (only one side of the wire has its insulation removed) http://www.youtube.com/watch?v=it_Z7NdKgmY



While a diagram such as this on the left shows a simple description of the construction of a DC motor a typical dc motor has:

- several separate coils and multiple connections to the commutator,
- many turns on each coil of wire
- a shaft through the coil to which we can connect things like wheels or gearboxes.

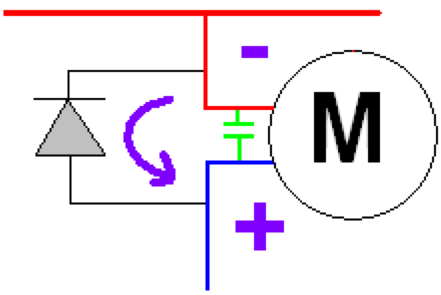


We can control a small DC motor with a simple transistor switch ciruit, similar to the LCD backlight control.In this case the backlight has been replaced by a motor, a capacitor and a diode.

When a motor is running it produces a lot of electrical noise, this is due to the current being switched on and off by the commutator several times per second. The actual sparking can be seen between the brushes ane the comutator on some motors. This noise appears as spikes in the voltage on the power lines to the microcontroller and can cause your micro to reset all the time.

The diode is another important safety device to protect your transistor and microcontroller from sure desctruction.

A motor is a coil of wire i.e. an inductor; when there is current a magnetic field forms around the coil and when you turn it off this field collapses back into the coil turning your coil into a generator for a very short period of time, the field collapse causes charges to flow in the opposite direction and these can flow back into tyour transistor killing it instantaneously. The diode conducts these charges away safely.



DC Motors come in all shapes and sizes

Tamiya RE-260 Motor
RPM: 5040 (max efficiency) to 6300
supply voltage: 1.5V (4.5V max)
operating current: 640mA
torque: 15gcm
gear ratios: 41.7:1 to 64.8:1

Wheel is turned acrylic with rubber rim

Car electric window motor
12V approxiamately 4 Amps
with fitted worm gear

Car windscreen wiper motor
supply voltage: 12V
operating current: 6-8A
these have a built in gearbox and a reversing switch.
We dont use the reversing switch so we wire directly to the wires on the motor itself not the connector.

reclaimed printer DC Motor
supply voltage: 12V
operating current: 300mA

Knowledge about driving these devices relies on understanding the specifications for your motor.

A DC motor is rated at the voltage it is most efficient at. It is always tempting to run it at a higher voltage but if you apply too much it will overheat, when it gets too hot the insulation on the wires of the coil will melt shorting the whole lot out and cause a small (hopefully not big) fire. If you run it at a lower voltage, it just wont work or it wont work anywhere as well. The reason being that voltage is directly related to motor torque. Less voltage less torque, more voltage more torque.

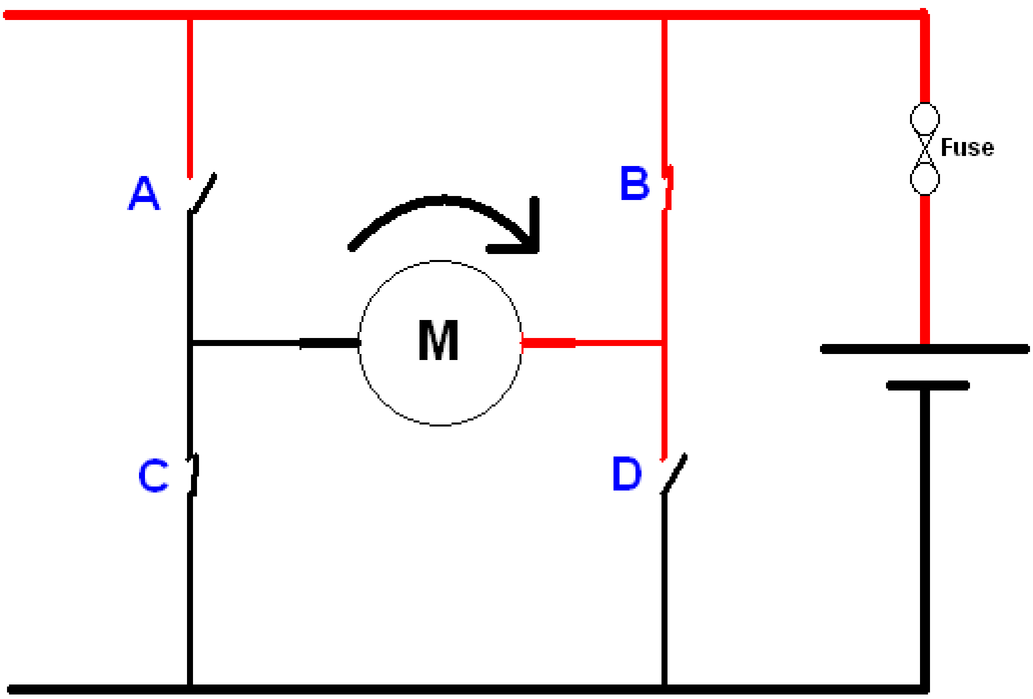
DC motors are generally made as non-polarized do if you reverse the voltage it goes in the opposite direction.

They have an operating current which is the typical current the motor will use under normal load/torque. The power used wll be the operating current times the rated voltage.
Your power supply must be able to meet this power requirement. If you have a 12V 2A (24W) motor and your power supply is only capable of 12V 500mA you will never drive the motor properly.

Another current rating is of significance it is the stall current. If you run you motor, but you hold the shaft so that it stops rotating a lot of current will flow (stall current) and a lot of power will be required. You must understand this when designing the power control circuits. Your power supply should be fused as well in case problems with the motor draw too much current over heating it.

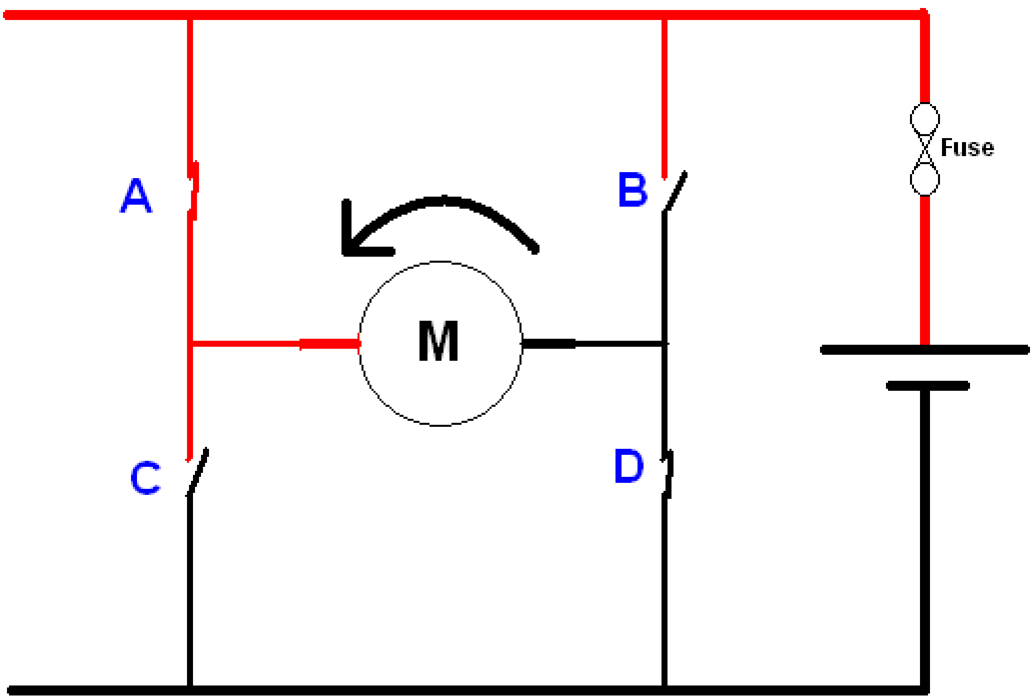
32.1 H-Bridge

A single transistor may be useful for turning a motor on or off however if a motor needs to be reversed in direction then an H-Bridge circuit is called for.
The principal is simple to reverse direction reverse the connection to the battery



B and C switches closed

A and D switches MUST BE OPEN or the battery will be shorted out!

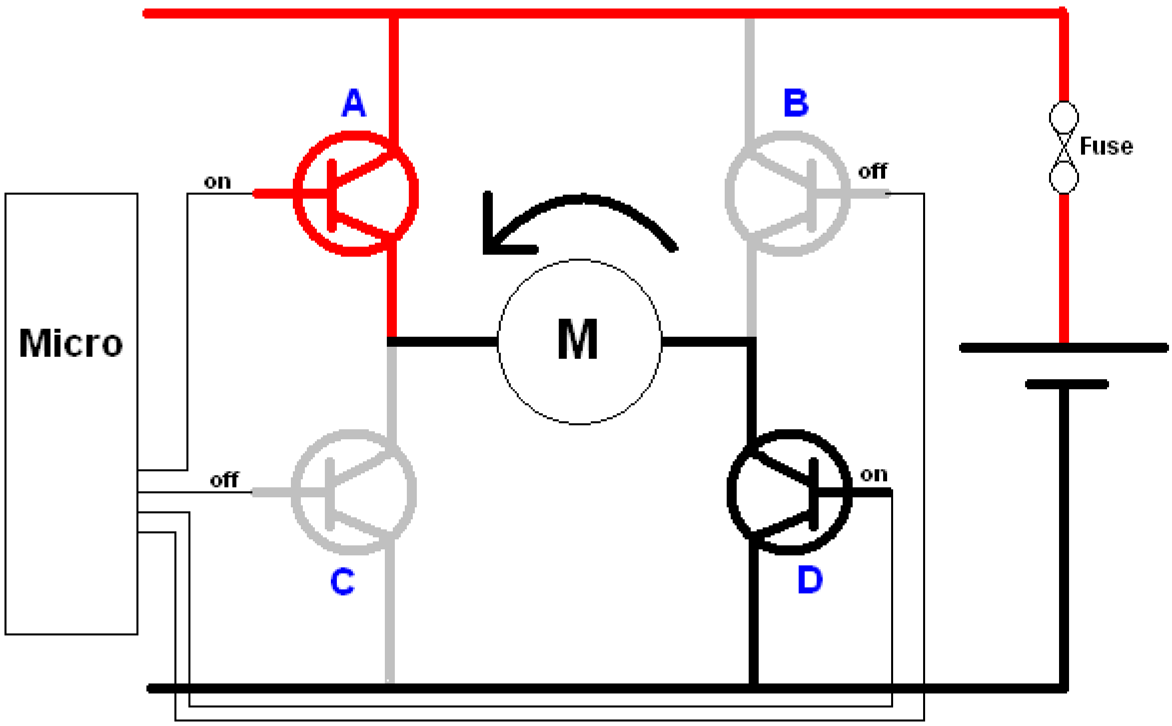


A and D switches closed

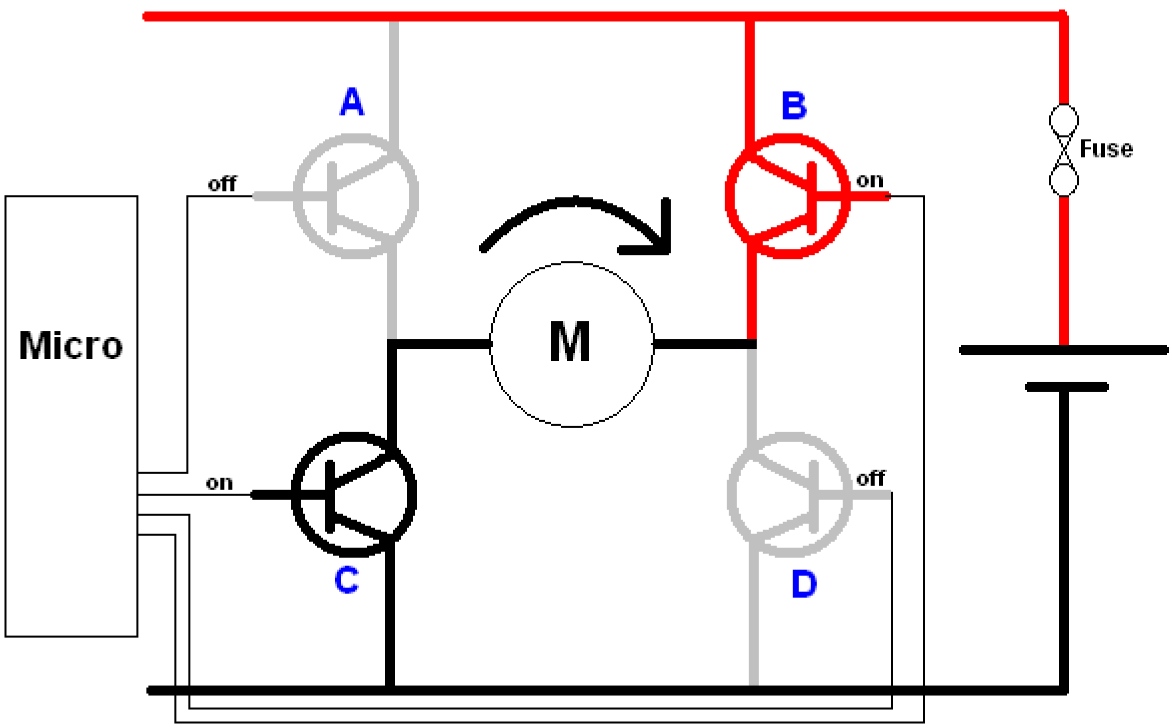
B and C switches muts be open or the battery will be shorted out!

NOTE : the circuit has fuses in it – these are a really really really good idea!!

A microcontroller can be used successfully to achieve this by switching 2 out of 4 transistors on and off in sequence.

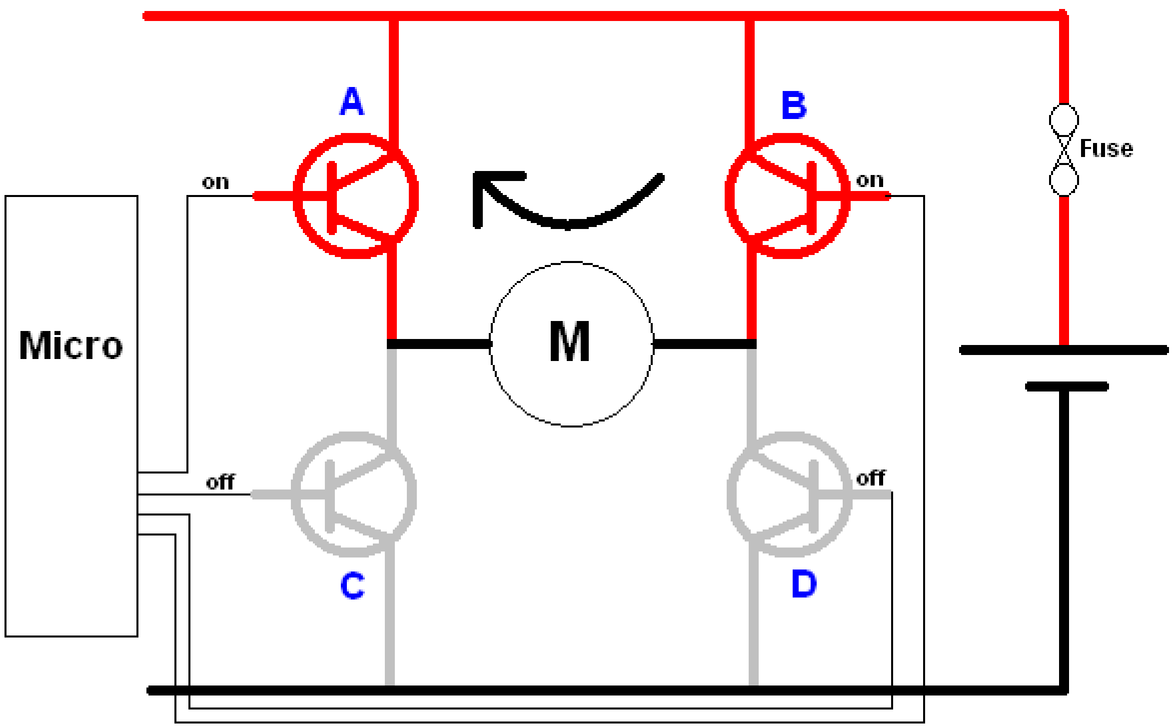


In the above diagrams the thick lines represent the fact that large currents are drawn through the motor and transistors, so heavy wiring is also required as well as fuses!



32.2 H-Bridge Braking

If we turn off all the transistors in an H-Bridge then the motor is free to turn. If we want it to stop in a hurry though we can force the motor to brake by shorting it out. To do do this we turn on two transistors such as A and B OR C and D .

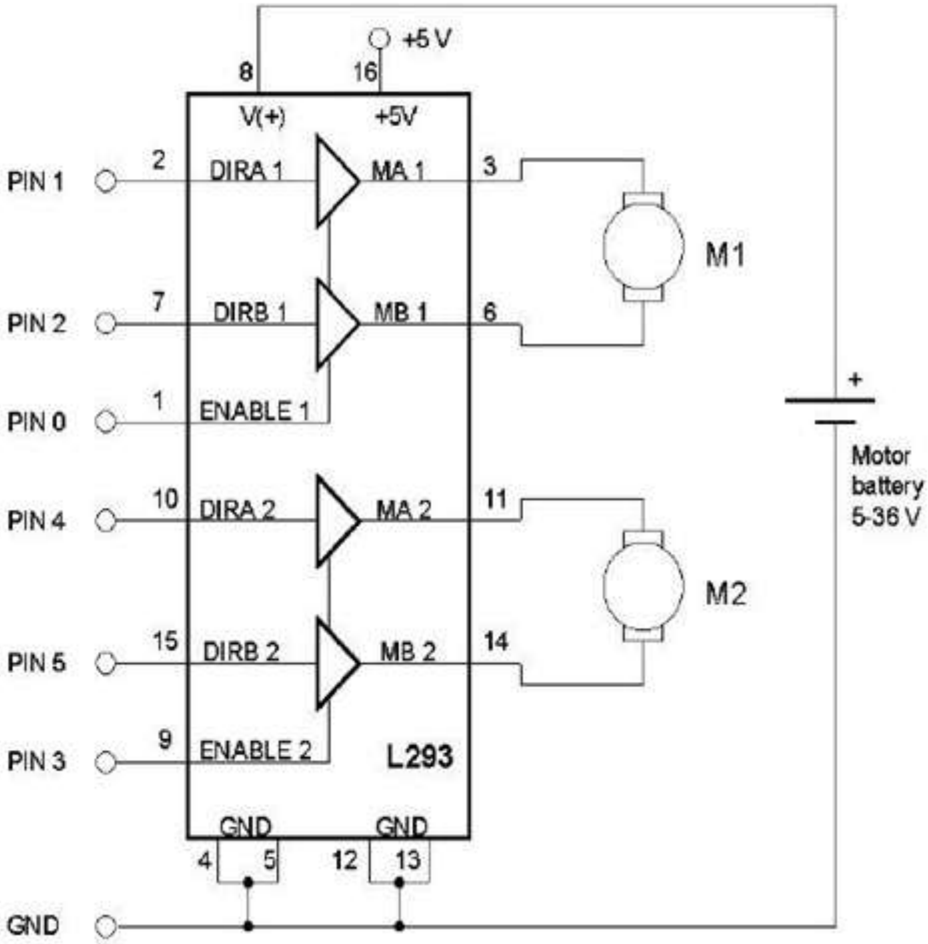


Truth table
This is a common thing to see in electronics a table that describes what happens on the output for each different combination of inputs. With 4 inputs there are 16 possible inputs. All combinations of inputs have been covered in this table.

A	B	C	D	Motor
H	L	L	H	Rotate Left
L	H	H	L	Rotate Right
H	H	L	L	Brake
L	L	H	H	Brake
L	L	L	L	Free
H	L	L	L	Free
L	H	L	L	Free
L	L	H	L	Free
L	L	L	L	Free
L	L	L	H	Free
H	X	H	X	Shorted Battery!!
X	H	X	H	Shorted Battery!!

H = high = 1
L = low = 0
X = don't care (this means that the otherinputs selected as high or low already have priority over these and it doesn't matter what you choose here)

32.3 L293D H-Bridge IC



Making an H-bridge circuit is not necessary for small and medium sized motors as plenty of ICs exist to help you, one of these is the L293D.

There are a couple of different versions of this IC the D model has internal protection diodes.

There are 4 ground pins which all must be connected to the pcb, they act as a heatsink for power to dissipoate through.

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Supply voltage, V_{CC1} (see Note 1)		36 V
Output supply voltage, V_{CC2}		36 V
Input voltage, V_I		7 V
Output voltage range, V_O		−3 V to V_{CC2} + 3 V
Peak output current, I_O (nonrepetitive, t ≤ 5 ms): L293		±2 A
Peak output current, I_O (nonrepetitive, t ≤ 100 µs): L293D		±1.2 A
Continuous output current, I_O: L293		±1 A
Continuous output current, I_O: L293D		±600 mA
Package thermal impedance, θ_{JA} (see Notes 2 and 3):	DWP package	TBD°C/W
	N package	67°C/W
	NE package	TBD°C/W
Maximum junction temperature, T_J		150°C
Storage temperature range, T_{stg}		−65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltage values are with respect to the network ground terminal.

2. Maximum power dissipation is a function of T_J(max), θ_{JA}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JA}$. Operating at the absolute maximum T_J of 150°C can affect reliability.

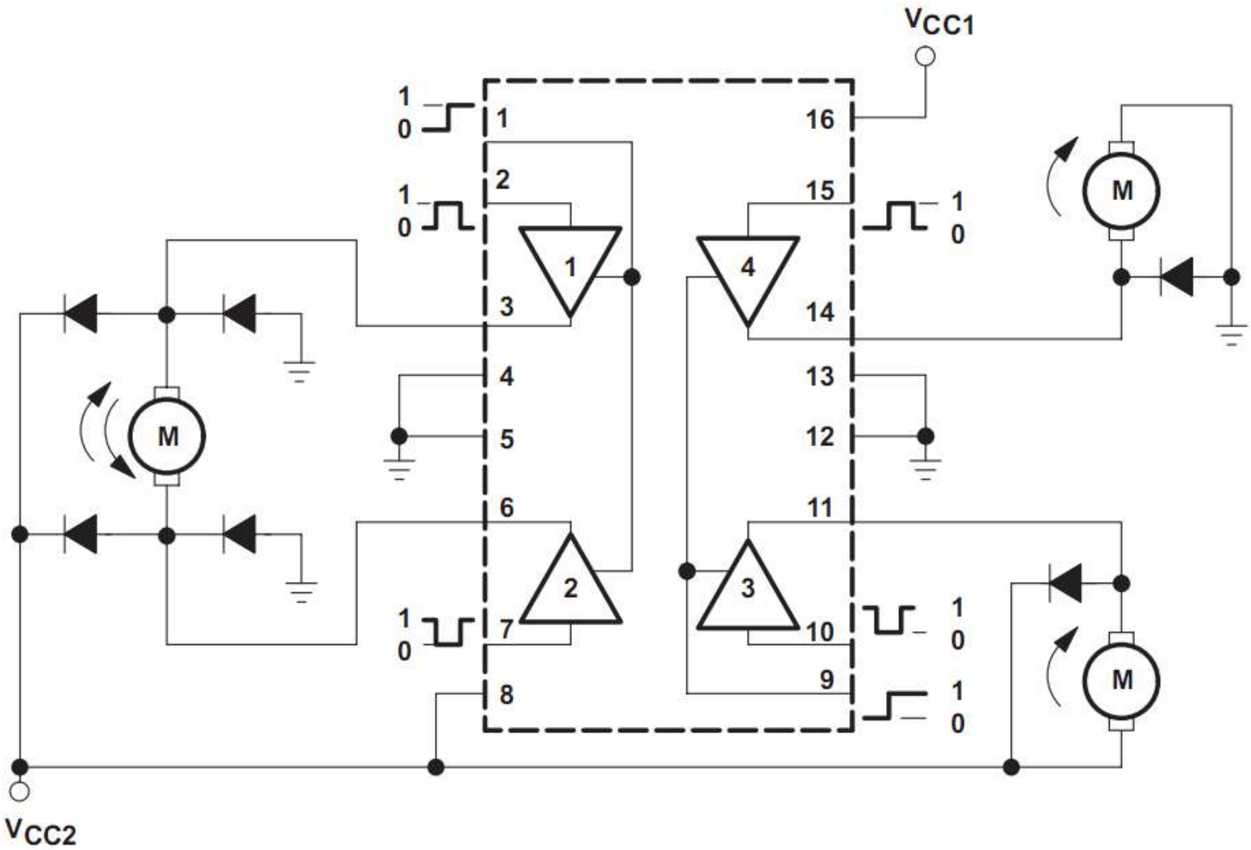
3. The package thermal impedance is calculated in accordance with JESD 51-7.

FUNCTION TABLE
(each driver)

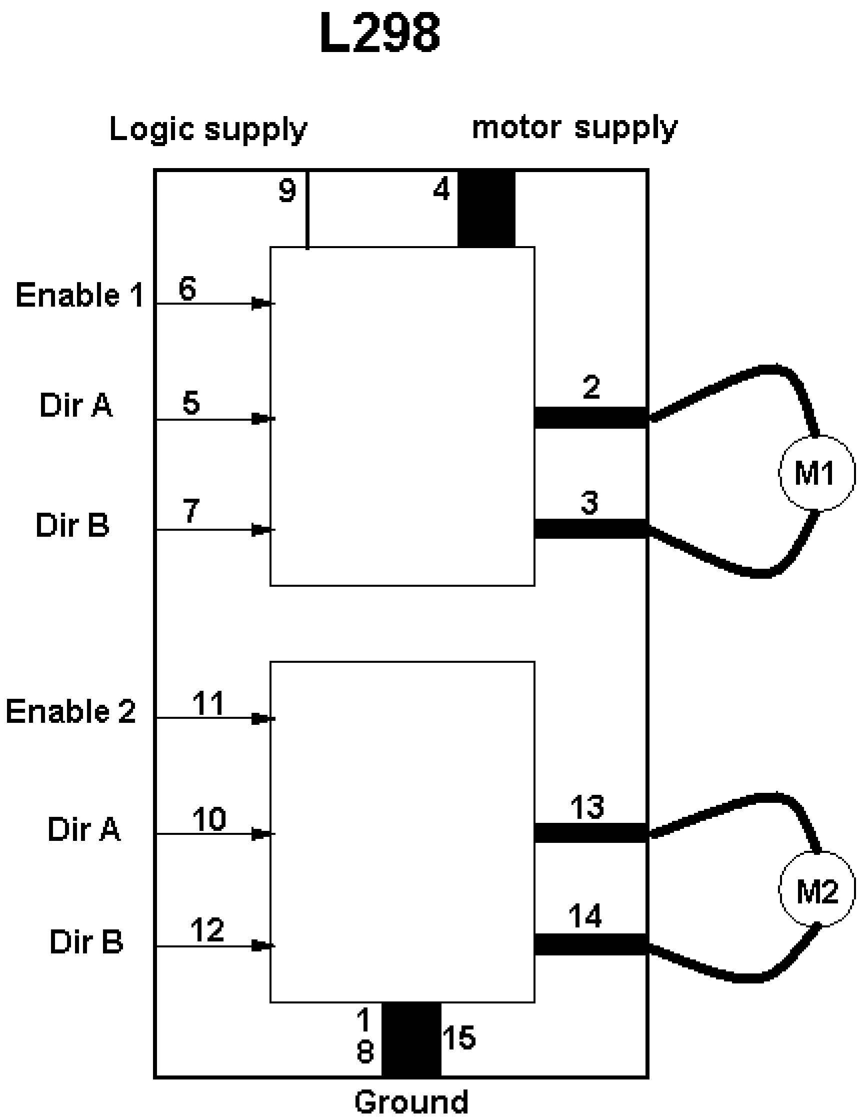
INPUTS†		OUTPUT
A	EN	Y
H	H	H
L	H	L
X	L	Z

H = high level, L = low level, X = irrelevant,
Z = high impedance (off)
† In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

The Enable pin must be high (1) for the chip to do its job, if it is low (0) then the output is off, what we call high impedance, that means floating, something we normally want to avoid on input pins to a microcontroller but whichis great on outputs.

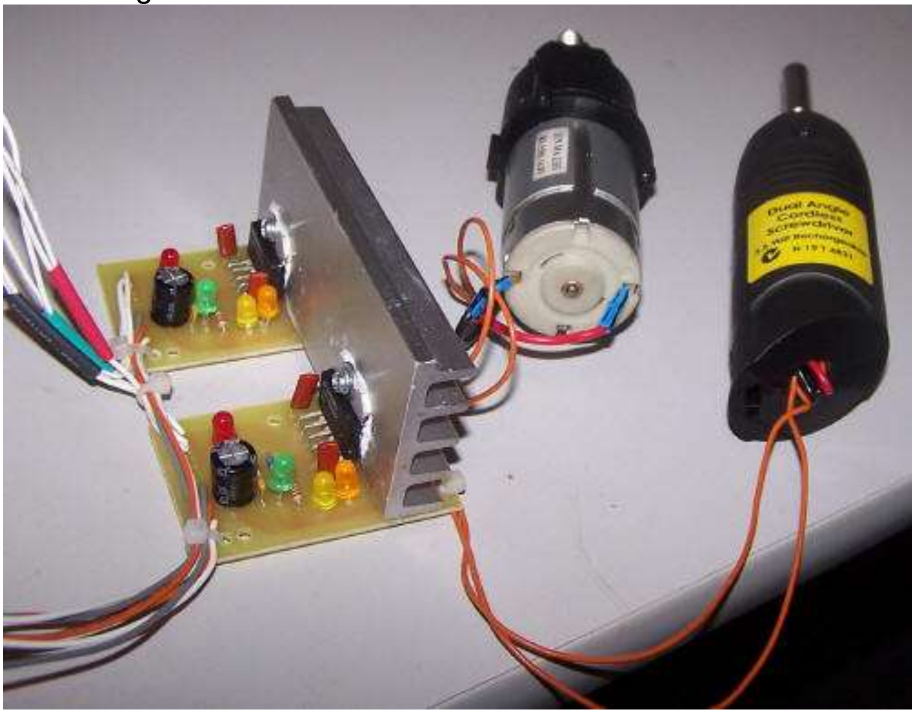


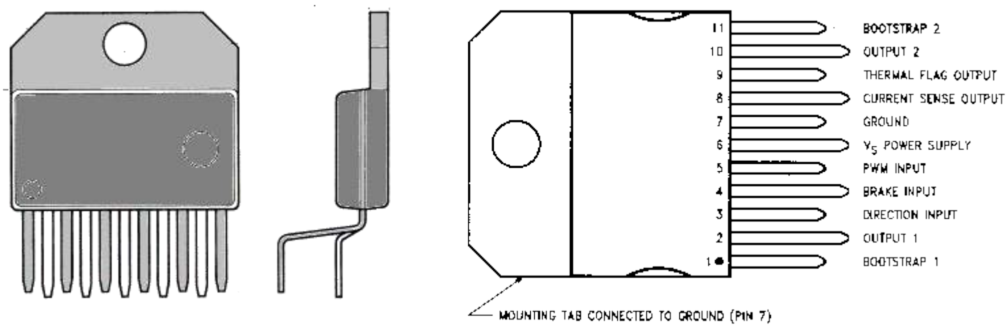
32.4 L298 H-Bridge IC

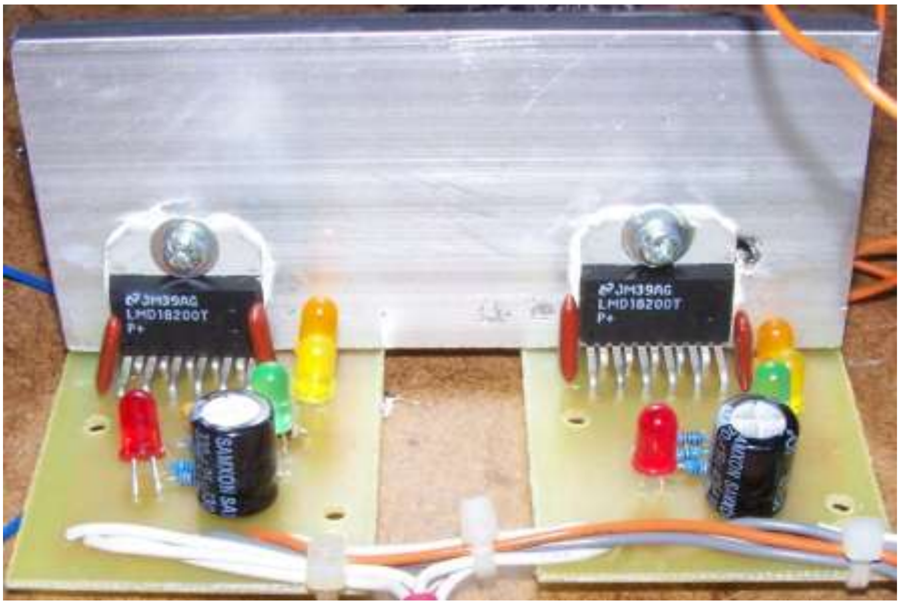


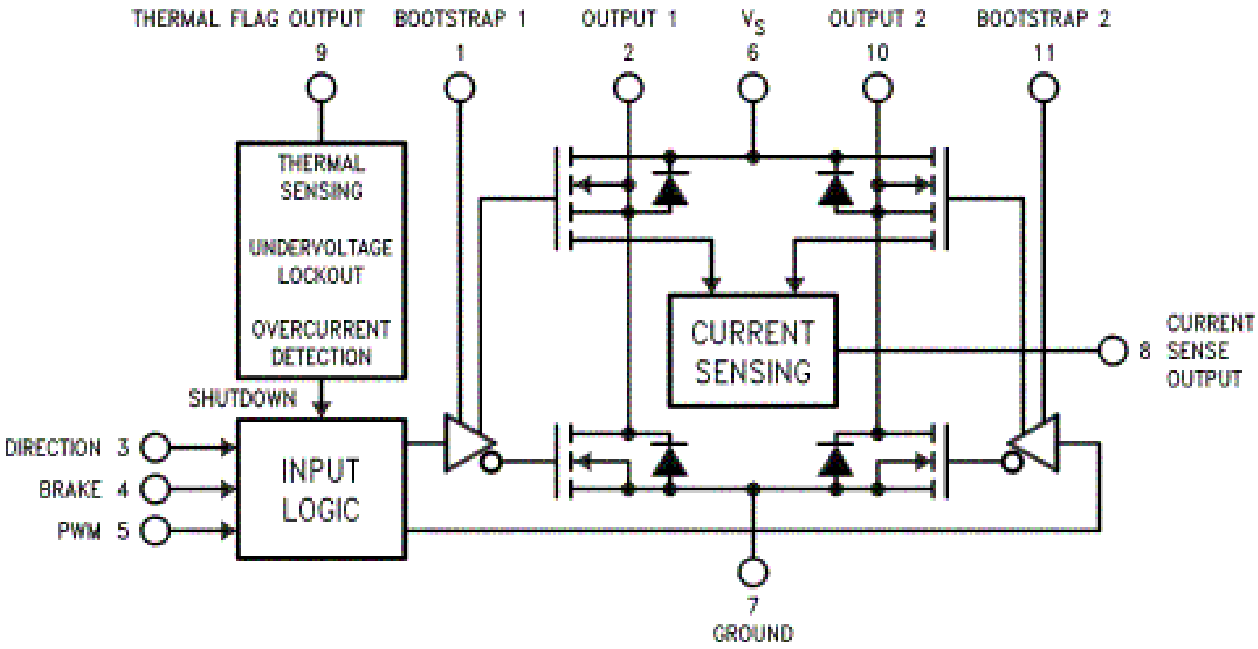
32.5 LMD18200 H-Bridge IC

In this diagram two LMD18200 circuits are connected to two DC motors from handheld drills.

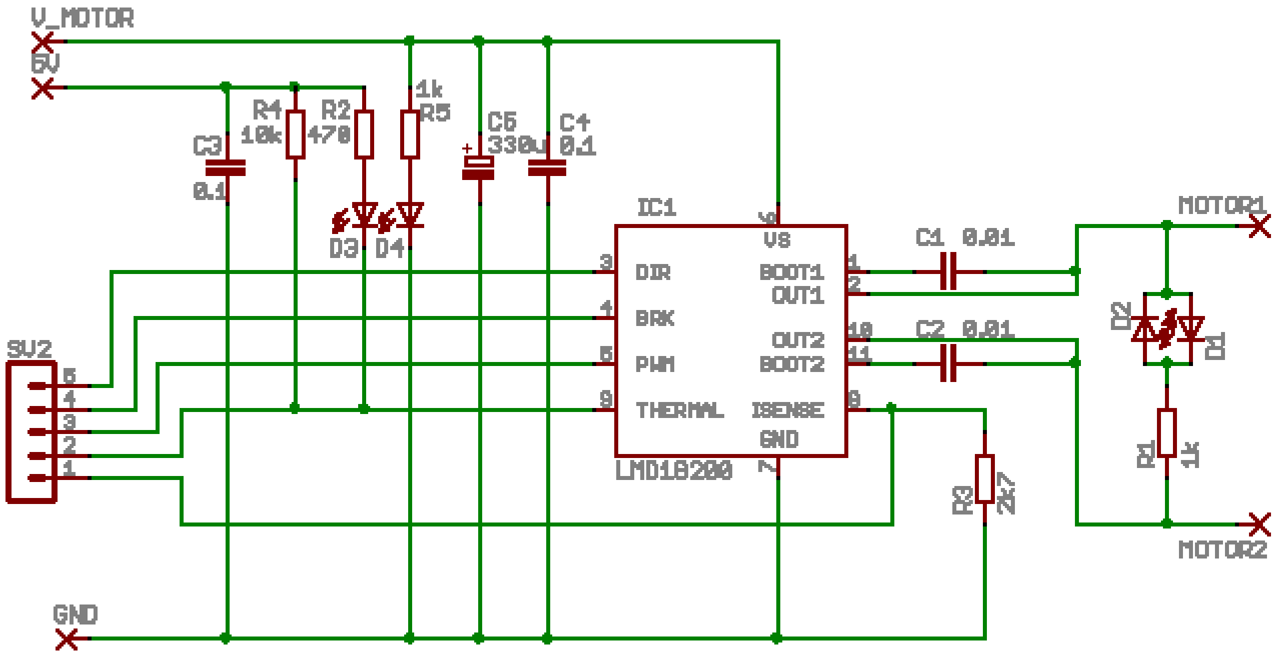








The circuit is straight forward, but some LEDs have been added so that the operation of the circuit can be observed while under the control of the microcontroller.
There is on this chip a great current sense feature that we can use to feedback information to the micro.



To control this IC we need to know how to turn it on and off

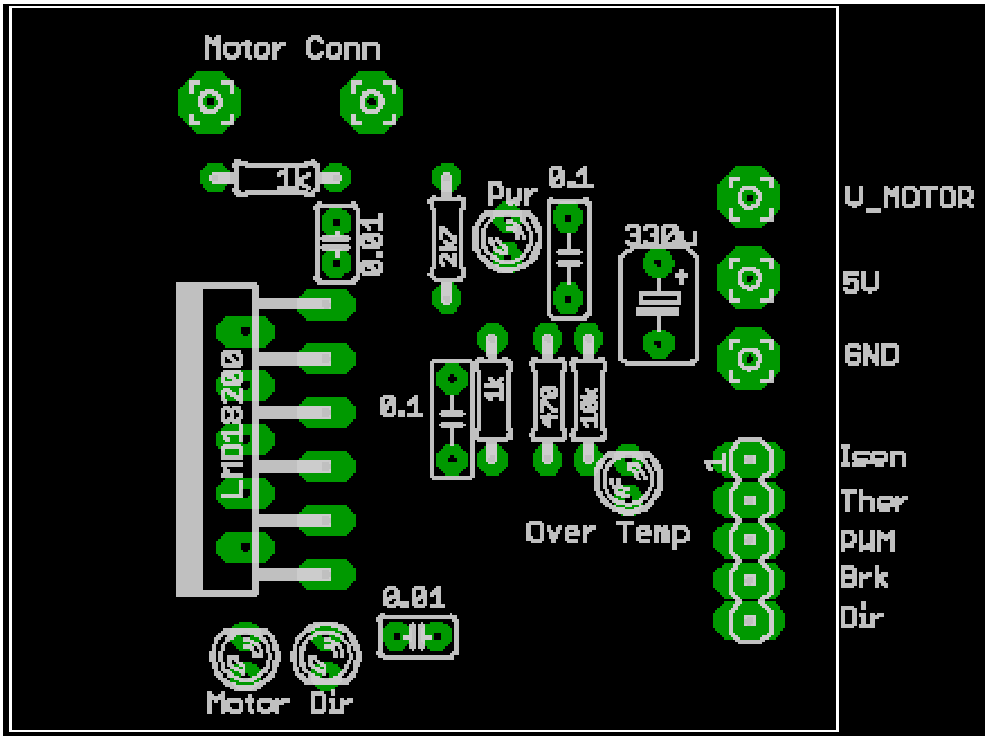
TABLE 1. Logic Truth Table

PWM	Dir	Brake	Active Output Drivers
H	H	L	Source 1, Sink 2
H	L	L	Sink 1, Source 2
L	X	L	Source 1, Source 2
H	H	H	Source 1, Source 2
H	L	H	Sink 1, Sink 2
L	X	H	NONE

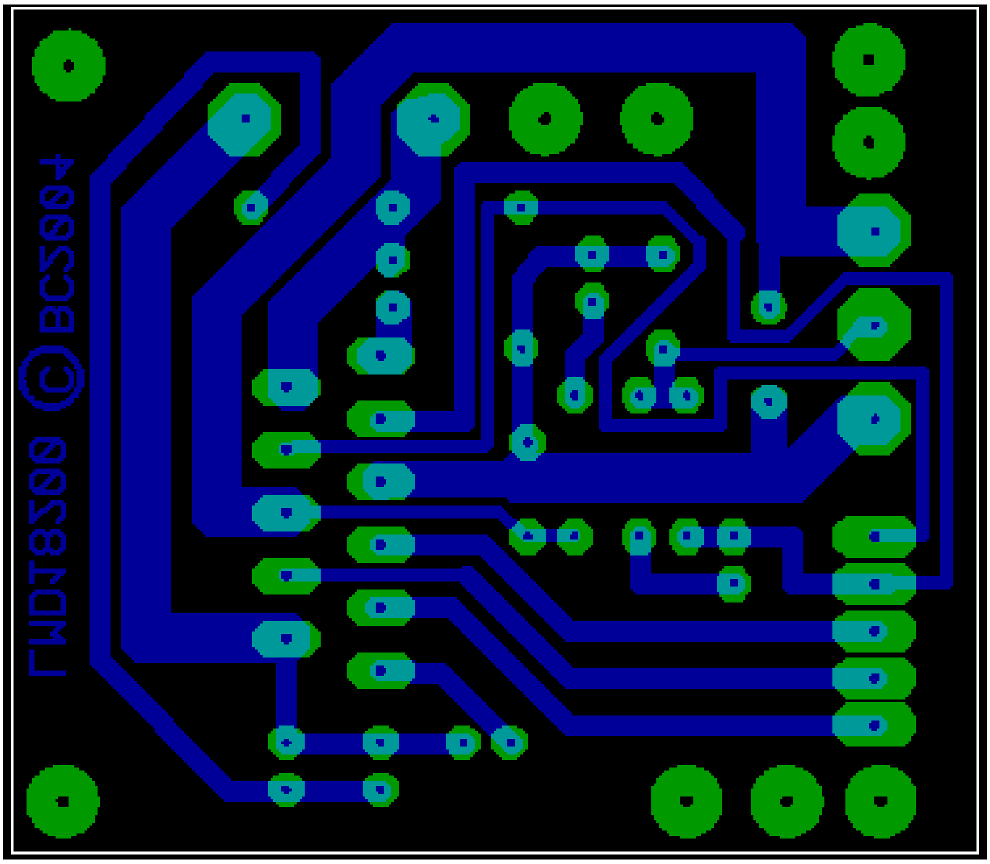
From this truth table we read:

To run the motor brake should be low, direction will be high or low and PWM should be high

To stop the motor, the brake should be high, PWM should be high and DIR can be either high or low.



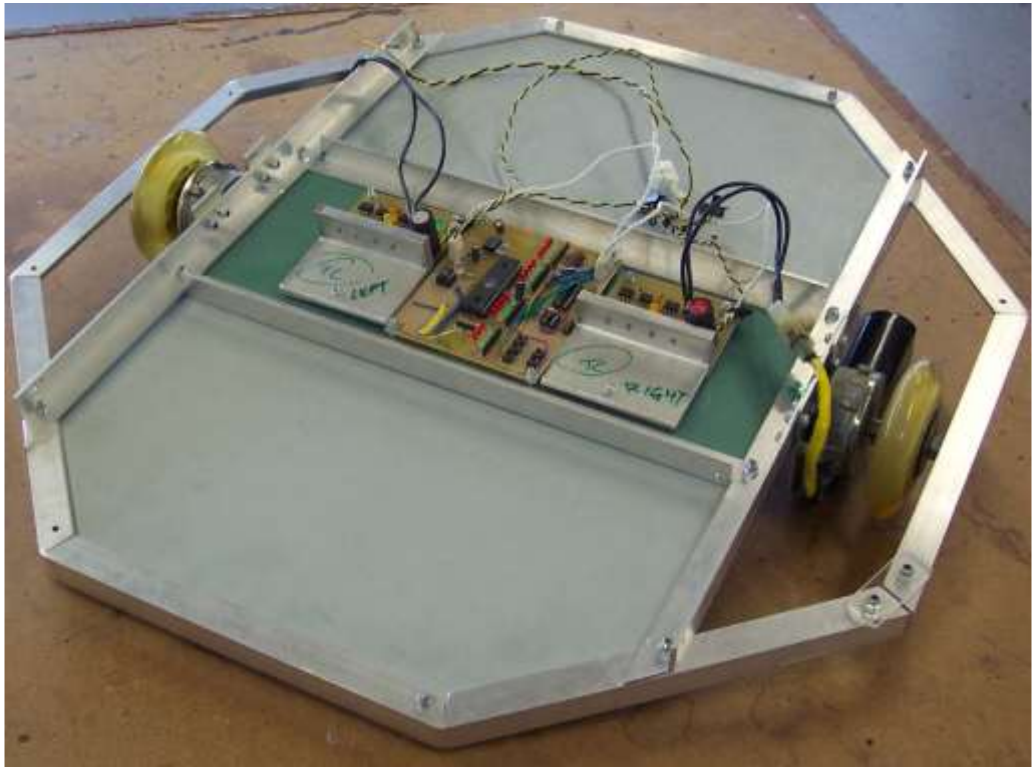
Layouts for the board, note the very large tracks becase a lot of power can be used in this circuit.



32.6
LMD18200 program

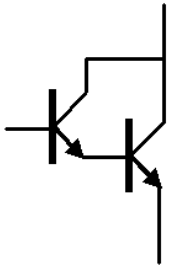
```
$regfile = "8535def.dat"                ' the micro we are using
'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Portd = Output
' 7. Hardware Aliases
M1dir Alias Portd.0
M1brk Alias Portd.1
M1pwm Alias Portd.4
M2brk Alias Portd.3
M2dir Alias Portd.2
M2pwm Alias Portd.5
'-----------------------------------------------------------------
' Program starts here
Reset M2brk
Set M2dir
Reset M2pwm
Reset M1brk
Reset M1dir
Set M1pwm
Wait 3

Do
     Reset M1pwm                     'off
     Waitms 10
     Set M1pwm                       'on
     Waitms 1
Loop                         ' keep looping forever
End                          'end program
```

32.7 Darlington H-Bridge



In this project TC developed a tool trolley for a mechanic working under a car. Here is it shown upside down with two darlington H-bridge boards on it.

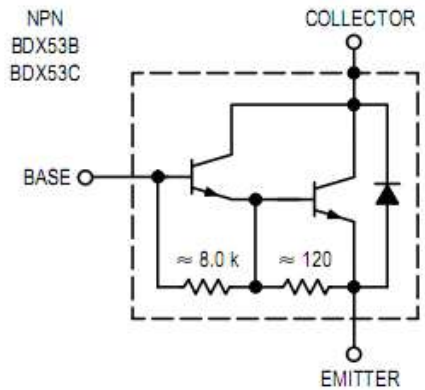
The motors are used electric window motors form a car and the wheels were from roller skates. Two castors were also needed for the final product.



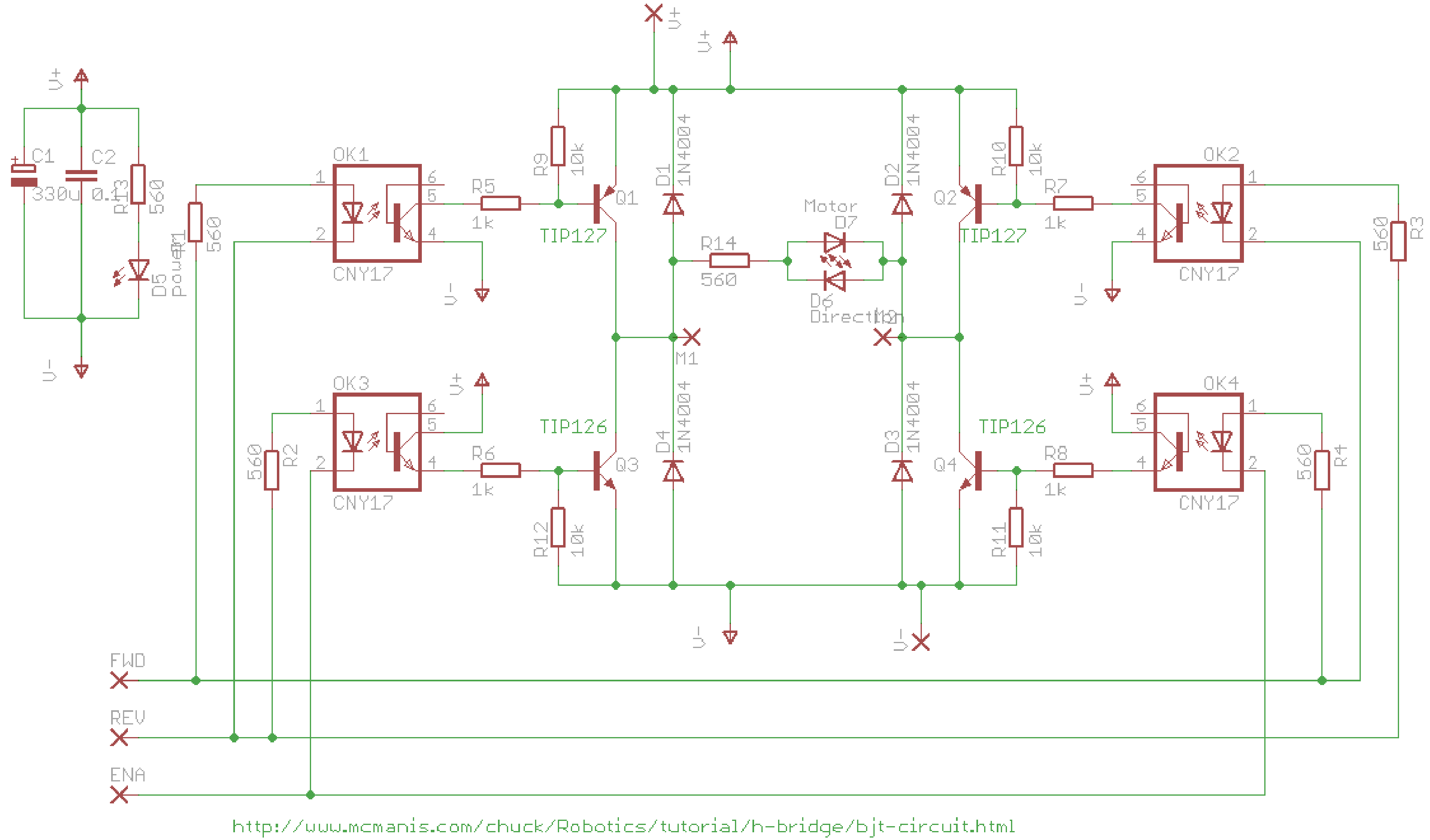
A high current circuit was needed so Darlington transistors were used.

Darlingtons such as BDX53C have much higher gain, because they effectively have 2 transistors one after the other in the circuit.
h_{FE} for the BDX53C is at least 750.

Note that it has a protection diode built into it already, but more were added in the circuit in case transistors without protection diodes were used to replace them in the future.



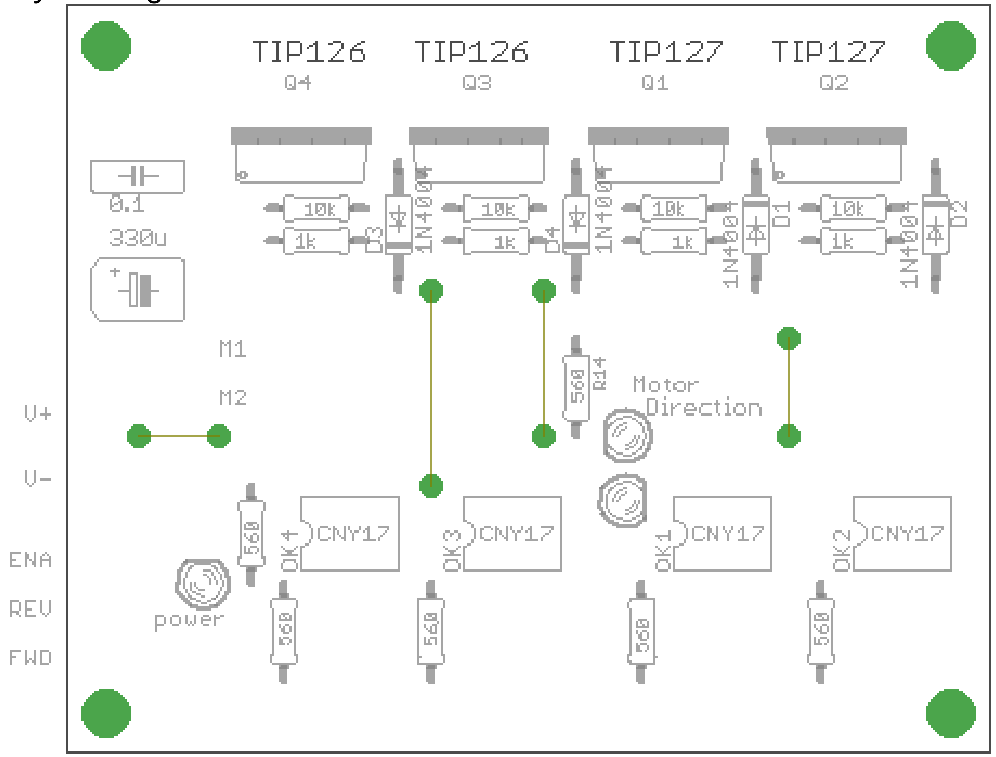
In other uses of this circuit TIP126 and TIP127 transistors were used. They have an h_{FE} of at last 1000,\

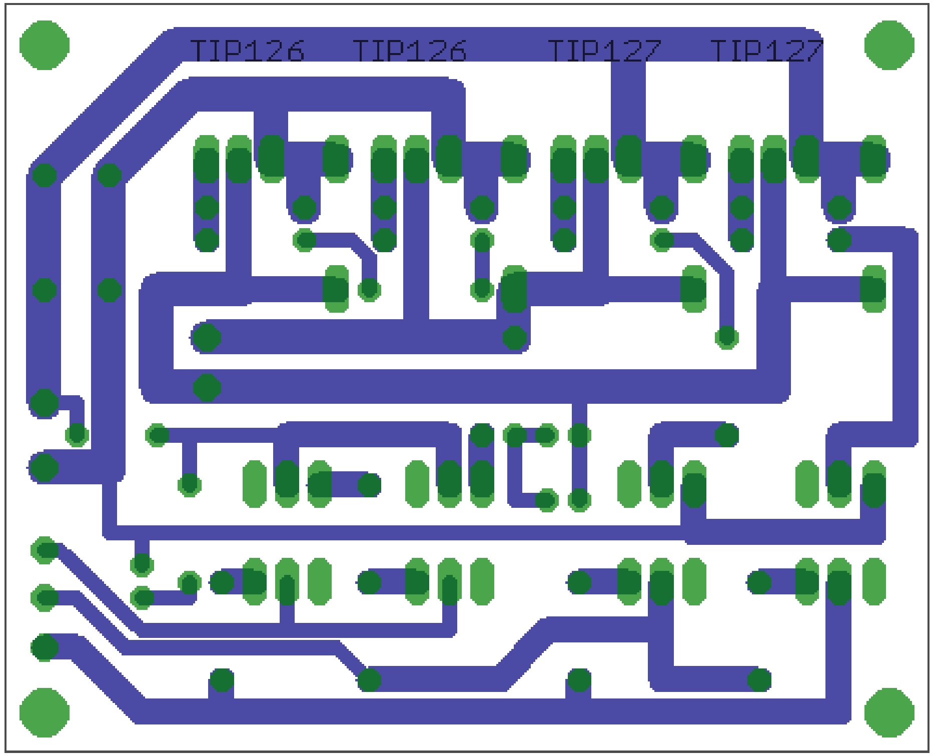


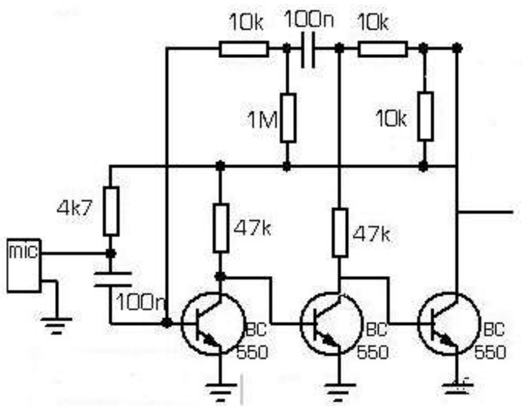
Fwd	Rev	Ena	
1	0	0	Q1 & Q4 On
0	1	0	Q2 & Q4 On

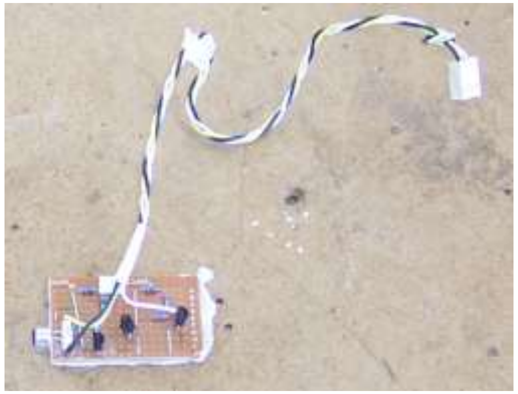
This circuit was based upon the circuit from www.mcmanis .com all we did differenyl was use parts easily available to us in NZ.
It has a really neat feature of protectingthe micr from transistor and motor noise using opto isolators and the smart way in which it is wired means we cannot turn on Q1 and Q3 (or Q2 and Q4) at the same time and blow them up!

Layout diagrams







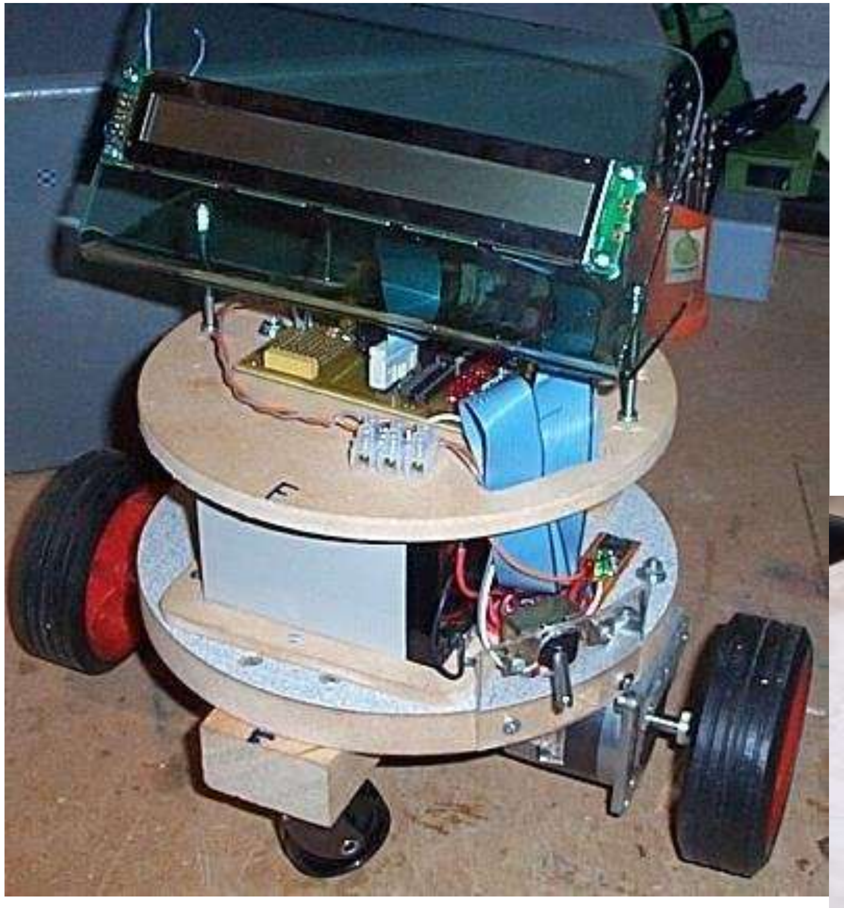


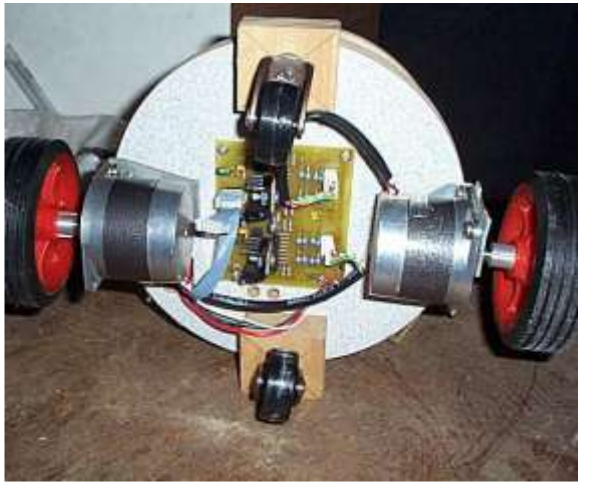
An important point to note are the heavy current tracks from the power supply to the power transistors.

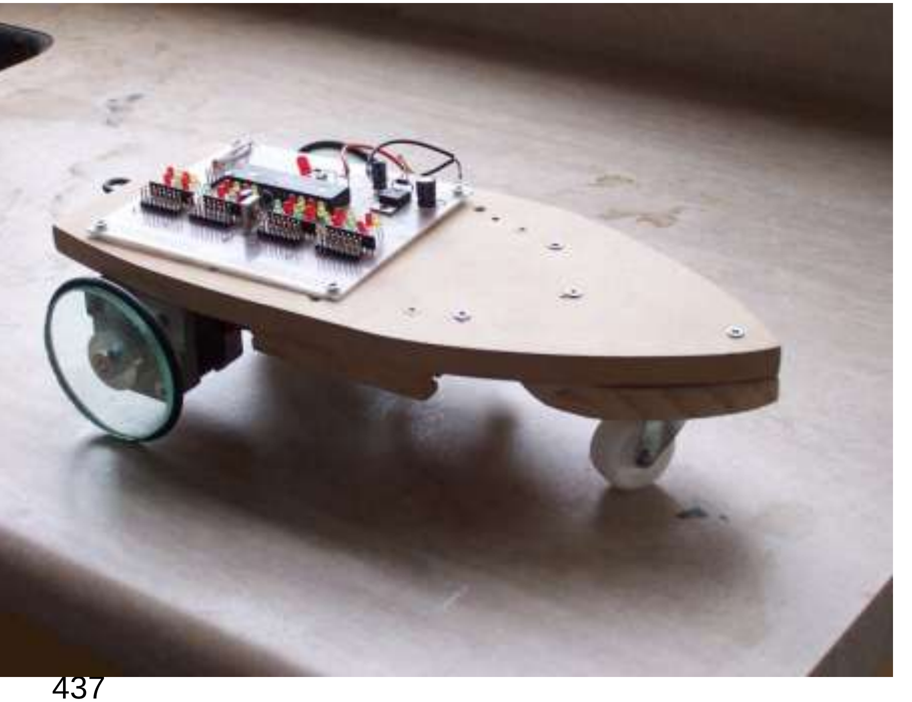
Here is the microphone sensor circuit.fo this sound tracking robot; 4 of these were needed with one mounted in each corner.

32.8 Stepper motors

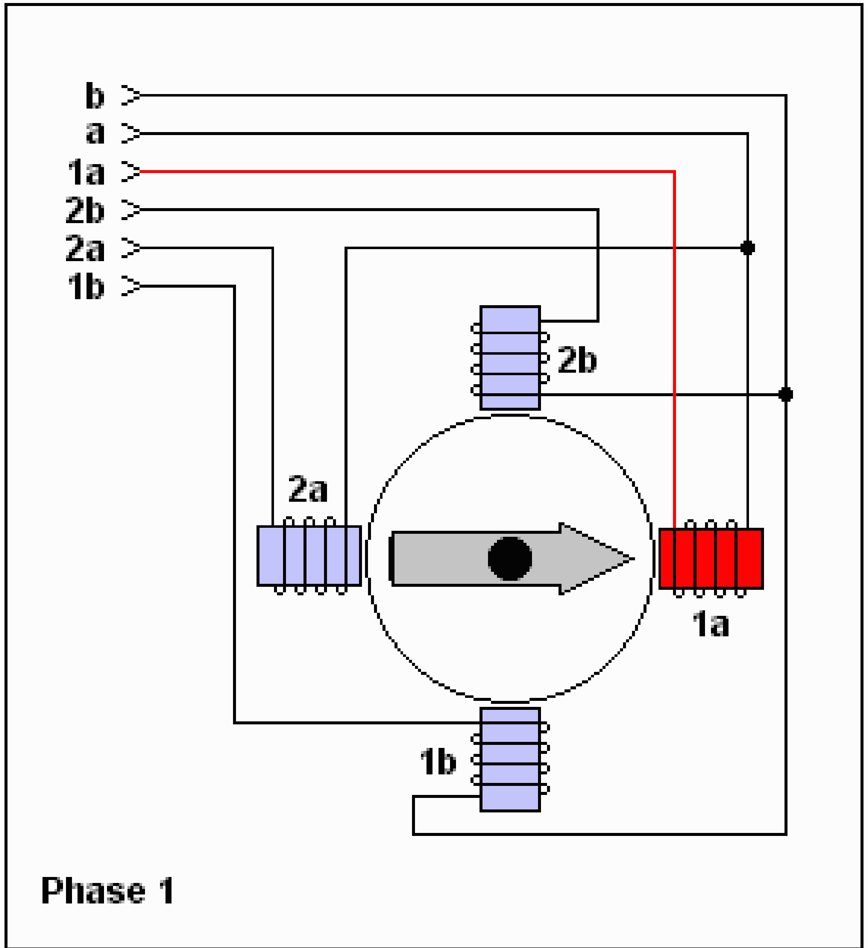
Stepper motors can be found in old printers and depending on the voltage and current can make small robots.



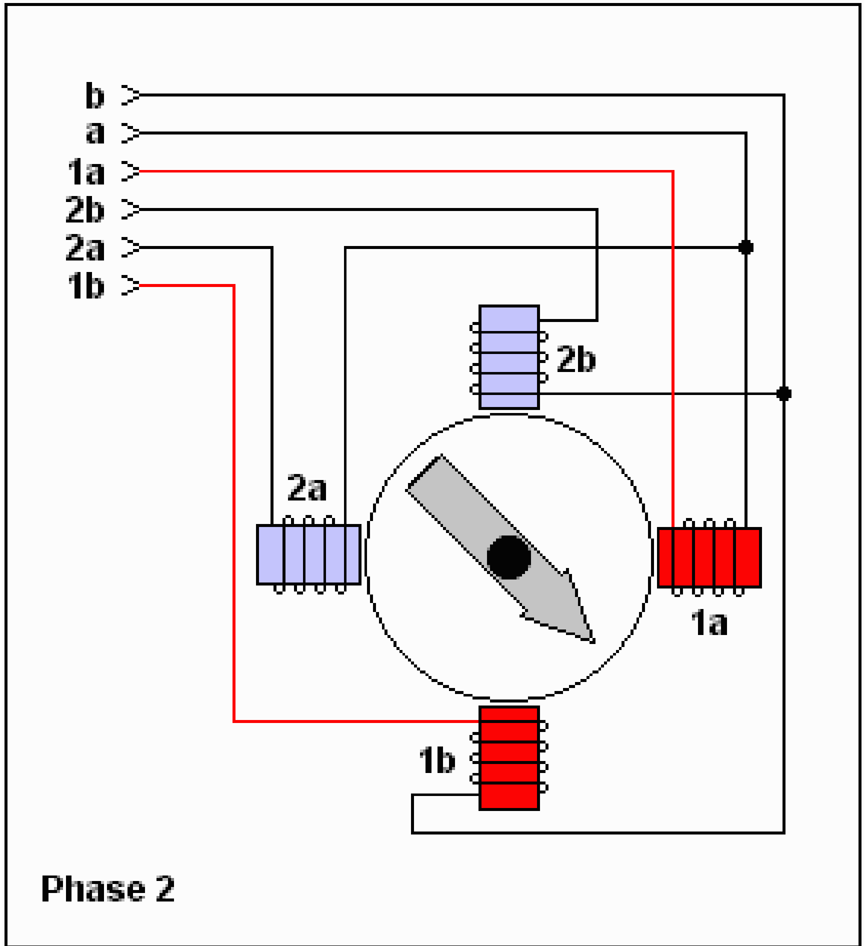




Think of a stepper motor as having 4 windings, they can be driven in full step mode where only one winding is on at a time, however they are better driven in half step mode where either one winding or two windings are on at a time.

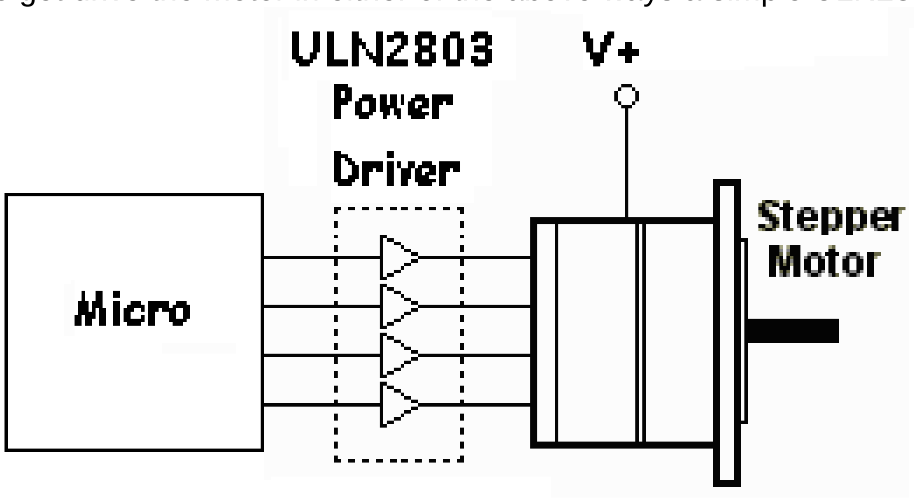


Stepper Motor Operation (Unipolar, Full step)

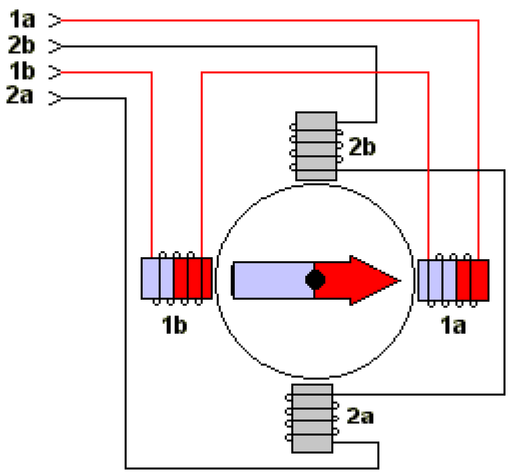


Stepper Motor Operation (Unipolar, Half step)

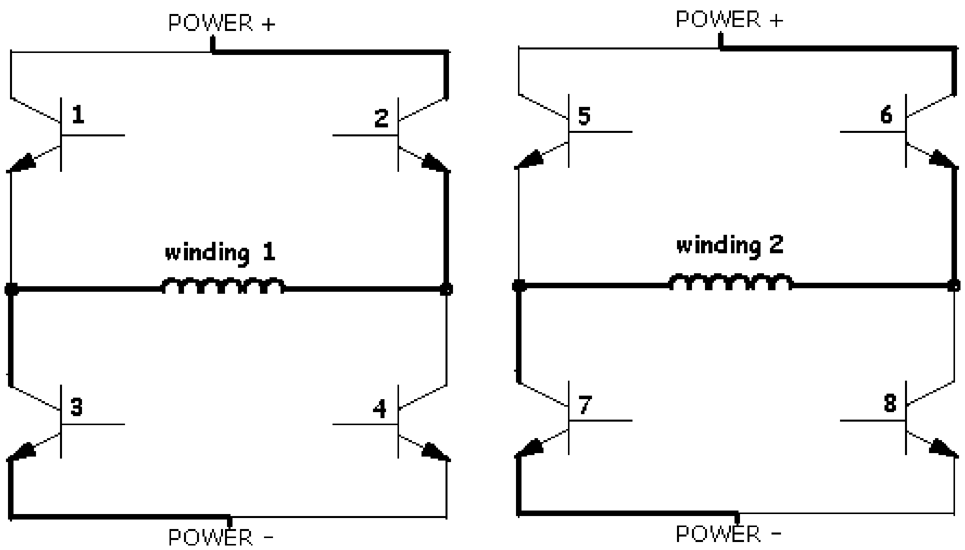
To get drive the motor in either of the above ways a simple ULN2803 darlington transistor array could be used



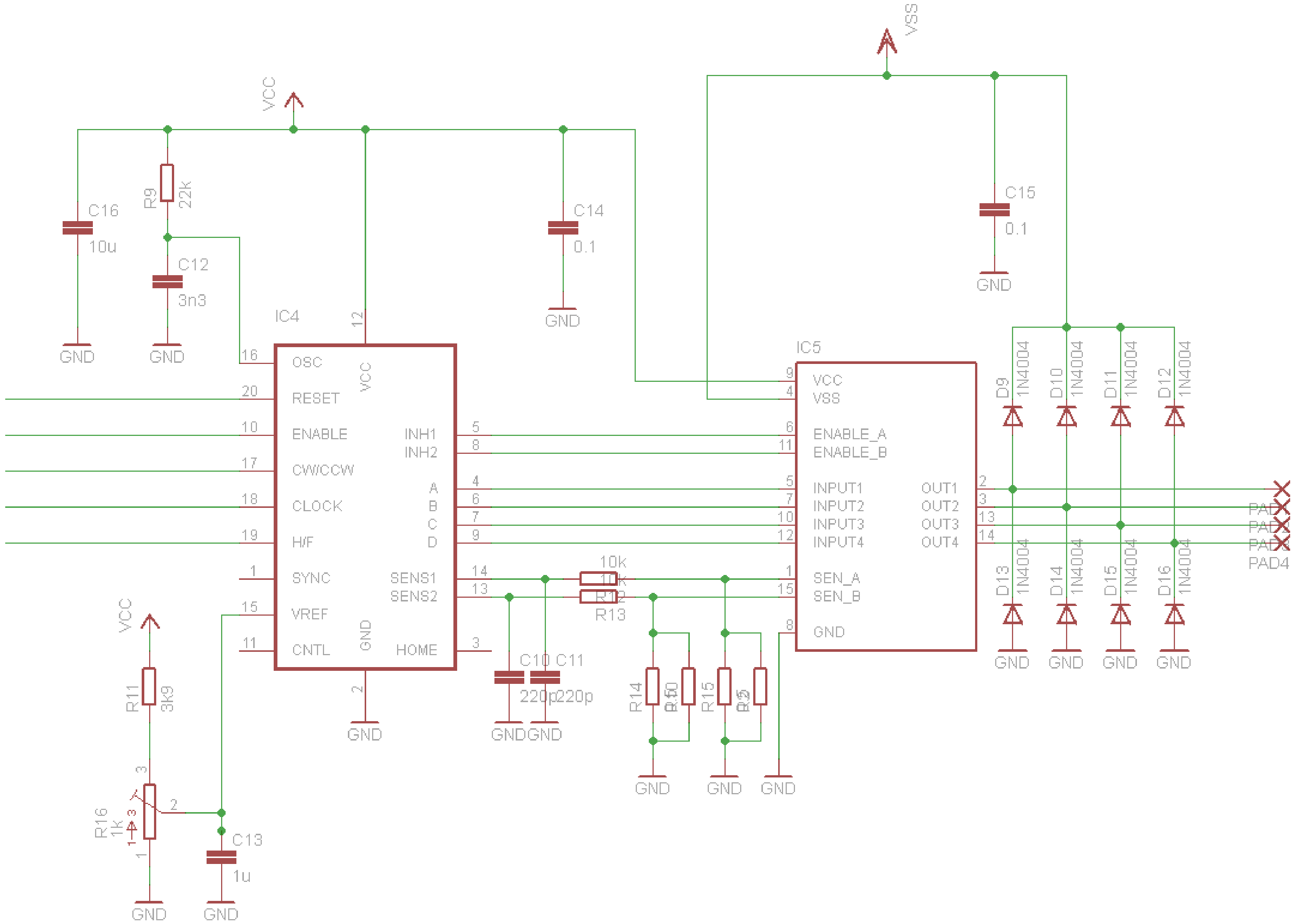
However there are a lot of inefficiencies in this sort of circuit and the motor power can be more fully made use of by driving more than one winding at a time, sometimes in differentdirections, which requires an H-Bridge type circuit.

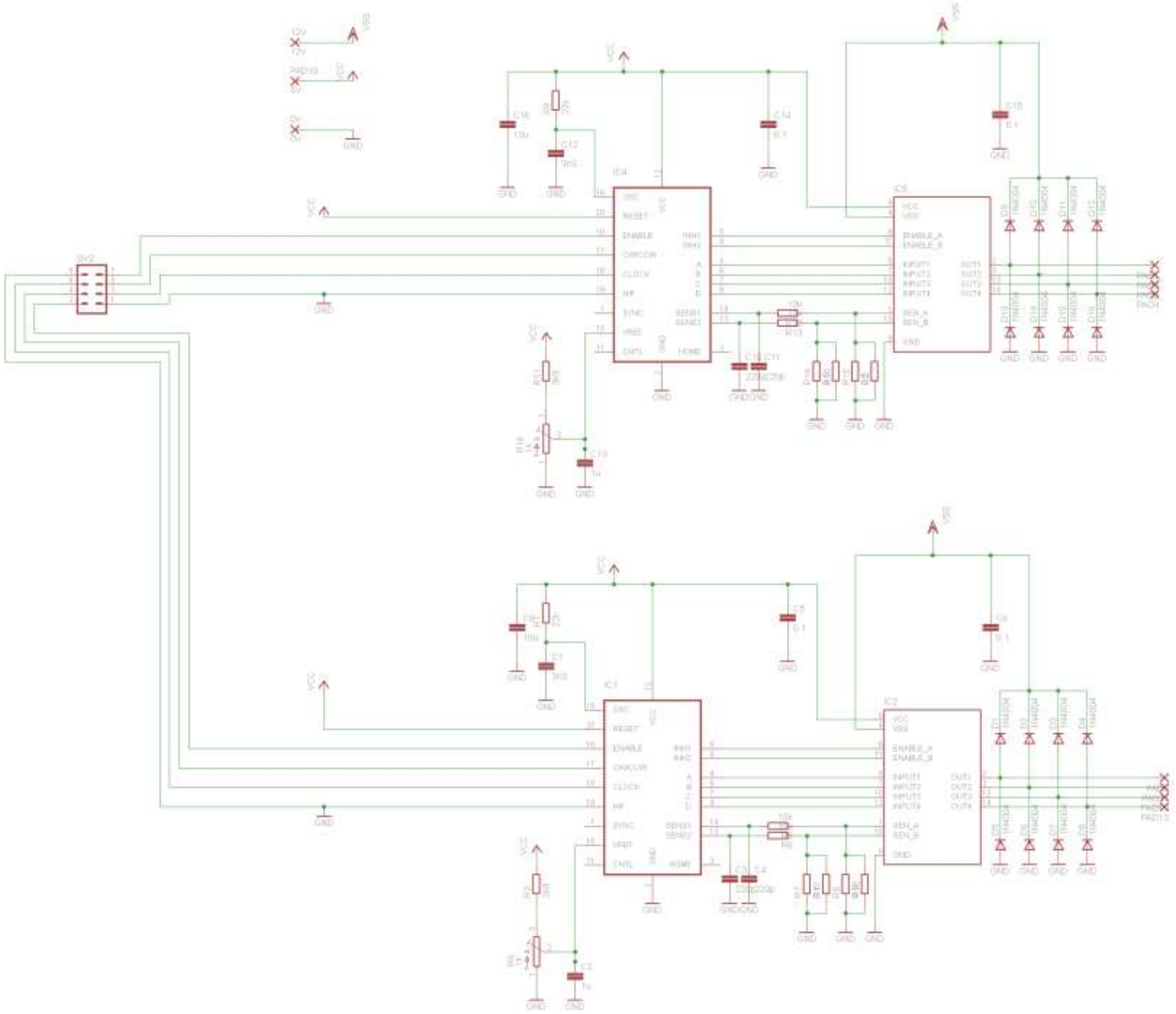


Conceptual Model of Bipolar Stepper Motor

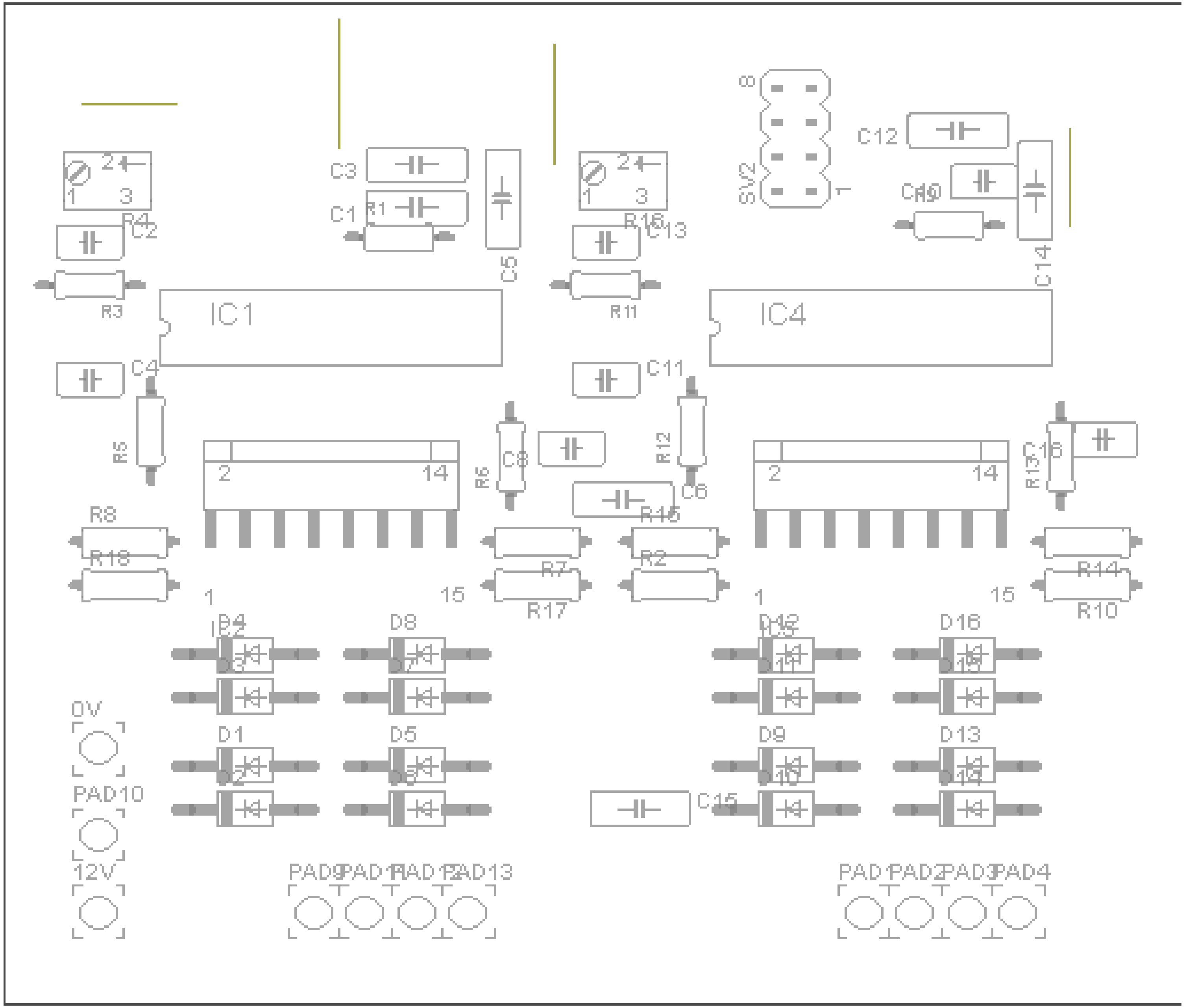


The L297 and L298 are some great driver chips for stepper motors, they do require careful use and are probably harder to find nowadays.

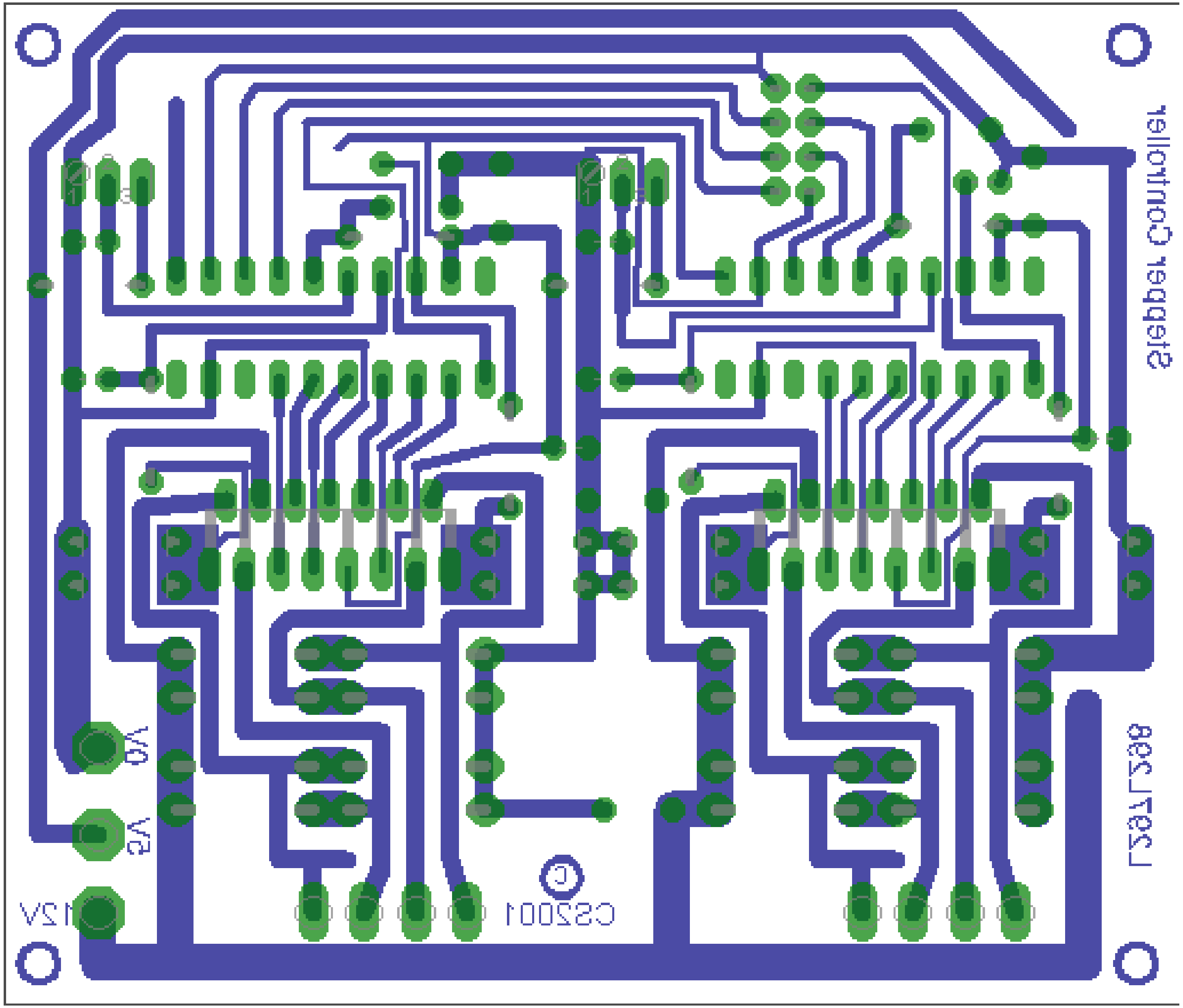




Full schemtic of the PCB with two complete driver circuits



Component layout for the PCB

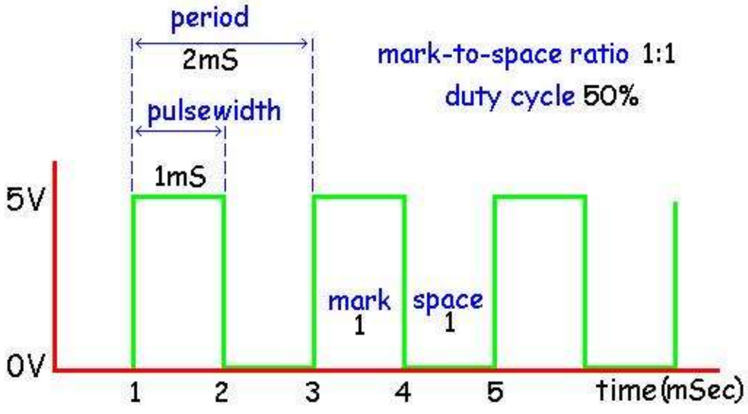


As with all motor circuits there is a need to keep tracks as short and direct as possible do note how this is achieved on the board

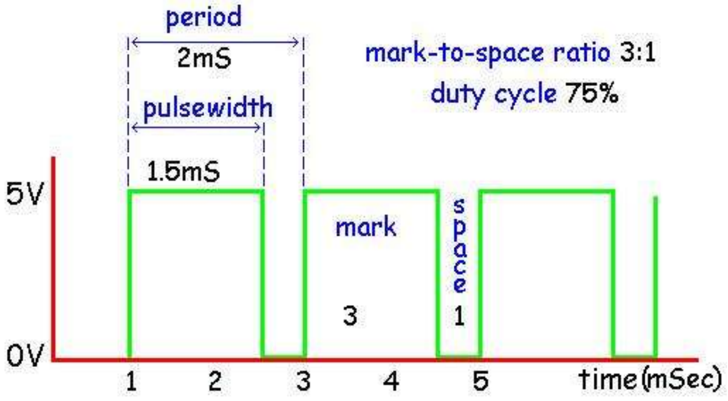
32.9 PWM - pulse width modulation

To control the brightness of an LED or speed of a dc motor we could reduce the voltage to it, however this has several disadvantages in terms of power reduction; a better solution is to turn it on and off rapidly. If the rate is fast enough then the flickering of the LED or the pulsing of the motor is not noticeable.

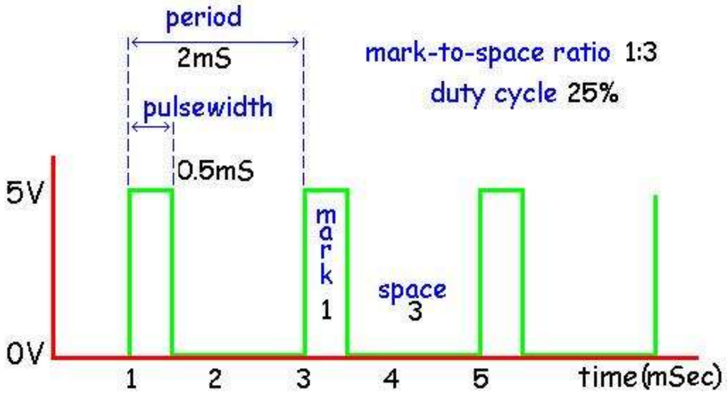
If this waveform was applied to a motor it would run at around half speed.



If this waveform were applied to an LED it would be at about ¾ brightness



If this waveform were applied to an motor it would be run at about ¼ speed



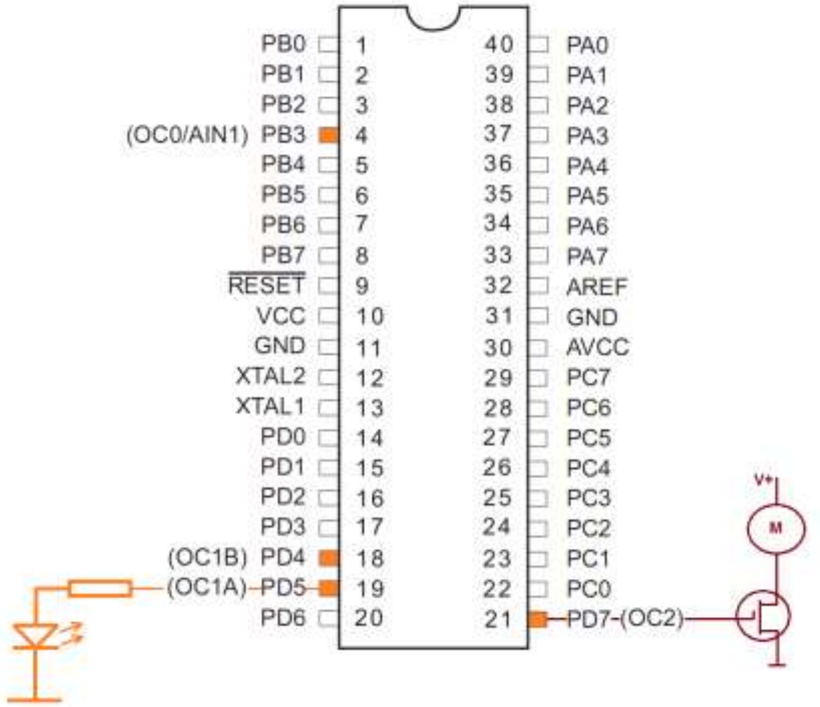
The AVR timer/counters can be used in PWM mode where the period of the wave or frequency is kept the same but the pulse width is varied. This is shown in the 3 diagrams, the period is 2mS for each of the three waveforms, yet the pulsewidth (on time) is different for each one (other modes do exist however these will not be described yet).

32.10 PWM outputs

In the Atmel microcontrollers there are one, two or sometimes more PWM output pins attached to each timer. On the ATMega16 Timer 0 has 1 PWM output, Timer 1 has two PWM outputs and Timer 2 has 1 PWM output :

These special pins mean that the PWM output once it is going is completely separate from your software.

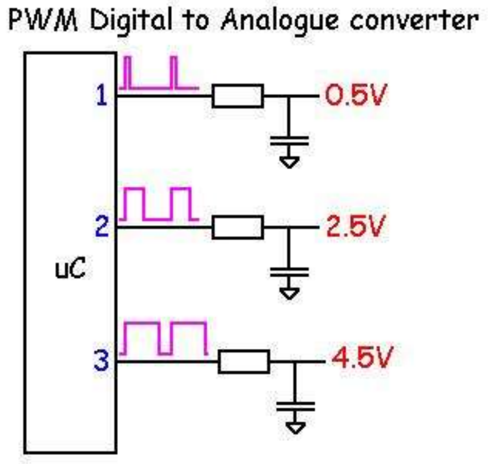
- For Timer0 the pin is OC0 (portB.3)
- For Timer1 the pins are OC1A (portD.5) and OC1B (portD.4)
- For Timer2 the pin is OC2 (portD.7)



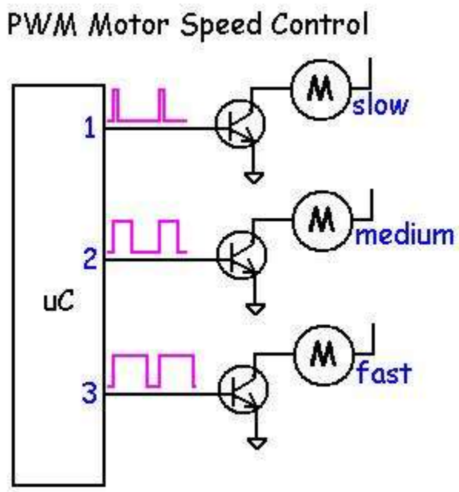
Here is example code to drive some different output devices connected to OC1A and OC1B

```
'O/P Period = 4ms /freq = 250Hz  (suitable for dimming an LED)
' range of brightness is controlled by the Compare1a and Compare1b registers
' as the Timer is set in 8 bit mode the values can be from 0 to 255
Config Timer1 = Pwm , Prescale = 64 , Pwm = 8 , Compare A Pwm = Clear Down ,
Compare B Pwm = Clear Down
Compare1a = 200 'high values = bright
Compare1b = 2 'low values = dim and high values = bright

'O/P freq = 16kHz  (suitable for speed control of a dc motor) , range is 0 to
255
Config Timer1 = Pwm , Prescale = 1 , Pwm = 8 , Compare A Pwm = Clear Down ,
Compare B Pwm = Clear Down
Compare1a = 200  'high speed
Compare1b = 20   'low speed

'O/P freq = 8kHz  (suitable for speed control of a dc motor)  , range = 0 to
511
Config Timer1 = Pwm , Prescale = 1 , Pwm = 9 , Compare A Pwm = Clear Down ,
Compare B Pwm = Clear Down
Compare1a = 511   'high speed
Compare1b = 20    'low speed
```

32.11 Uses for PWM



A pulse is used to charge a capacitor through a resistor, when the pulse is high the capacitor will charge, when it is low the capacitor will discharge, the wider the pulse the longer the capacitor charges and the higher the voltage will be.



The width of the pulse determines the average DC voltage getting to the motor which in turn slows or speeds up the motor. the advantage of using PWM rather than reducing the actual voltage is that torque (power) of the motor maintained at low speeds.

Period - the time from one point in the waveform to the same point in the next cycle of the waveform.
Frequency - the inverse of the period, if period = 2mS the frequency = 1/0.002 = 500 Hz (Hertz).
Pulse width - the length of time the pulse is high or on. The 'mark' time.
Duty cycle - the on time of the pulse as a proportion of the whole period of the waveform.

32.12 ATMEL AVRs PWM pins

As time goes by every new model of the AVR microcontroller that is introduced has more features; and it can be hard to keep up with all these features. For instance PWM each chip has different capabilities for hardware PWM.

AVR	PWM	Pins
ATTiny13	2 using Timer 0	OC0A OC0B
ATTiny45	2 using Timer 0 2 using Timer 1	OC0A OC0B OC1A OC1B (note OC0B and OC1A share the same pin so cannot be used at the same time)
ATTiny2313	2 using Timer 0 2 using Timer 1	OC0A OC0B OC1A OC1B
ATTiny26	2 using Timer 1	OC1A OC1B
ATTiny461	6 using Timer 1	OC1A OC1B OC1D (and their inverses)
ATMega8535 / 16 / 32	1 using Timer 0 2 using Timer 1 1 using Timer 2	OC0 OC1A OC1B OC2
ATMega48 / 644	2 using Timer 0 2 using Timer 1 2 using Timer 2	OC0A OC0B OC1A OC1B OC2A OC2B

32.13 PWM on any port

The issue with hardware PWM is that it is fixed to particular pins on the microcontroller.
What happens then when you want more PWM outputs or to use different pins.
Here is a PWM solution for PWM on portA.7 using the 8 bit timer0.

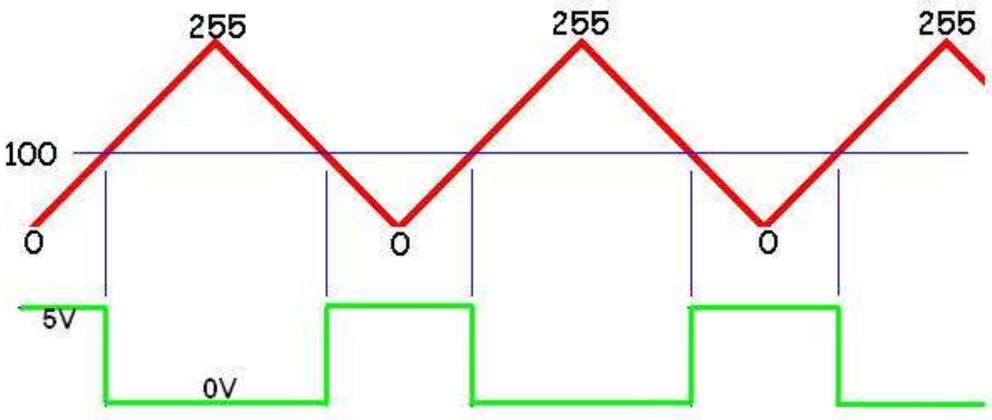
```
'PWM Timer2 pwm on any port
'Timer 2 PWM 8bit period = 15.8mS  =64Hz (suitable for driving a servo motor)
Config Timer2 = Pwm , Prescale = 256 , Compare Pwm = Disconnect
Compare2 = 50
Enable Timer2 : Enable Oc2
Enable Interrupts

'*********************************************
'Program starts here
Do

Loop
End

'*********************************************
'Interrupt Routines
'Timer2 pwm on any port, freq = 64Hz
   T2_ovf:
      Set PORTA.7
   Return
   T2_oc2:
      Reset PORTA.7
   Return
```

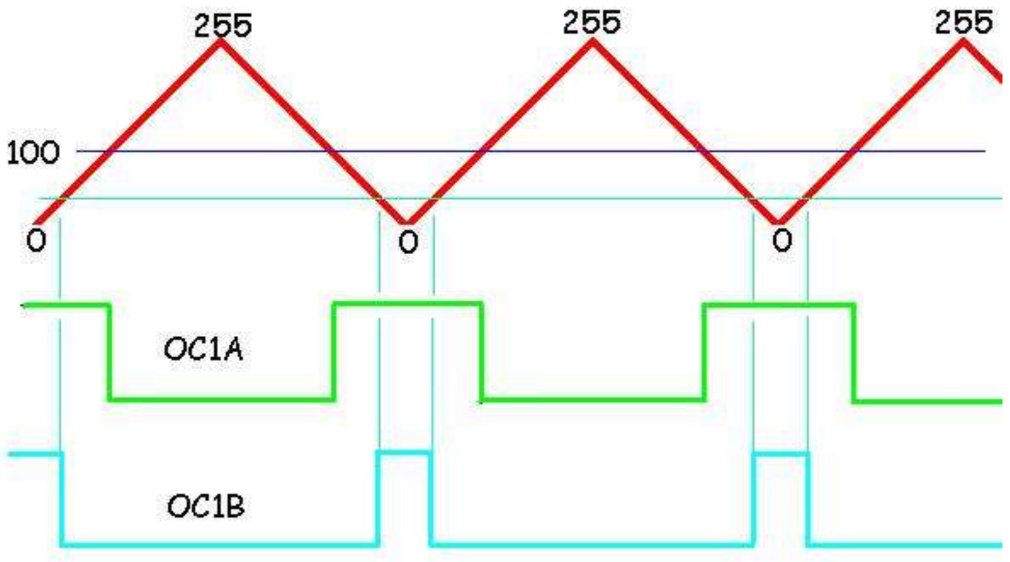
32.14 PWM internals

Each PWM output has independent settings for the pulse width however if they are controlled by the same timer they will will run at the same frequency.
The 3 PWM modes for timer1 discussed here are the 8, 9 & 10 bit mode.

- In 8 bit mode the counter counts from 0 to 255 then back down to 0.
- In 9 bit mode the counter counts from 0 to 511 then back down to 0.
- In 10 bit mode the counter counts from 0 to 1023 then back down to 0.

The programmer sets a point from 0 to 255 at which the output will change from high to low.
If the value were set to 100 then the output pulse on portd.5 (OC1A) would switch from 0Volts (0) to 5 Volts (1) as in the next picture.





To work out the frequency of the pulses
For 8 bit: Freq = 8000000/prescale/256/2
For 9 bit: Freq = 8000000/prescale/512/2
For 10 bit: Freq = 8000000/prescale/1024/2

The lines of code to get the above waveforms on OC1A and OC1B would be

- Config Timer1 = Pwm , Pwm = 8 , Compare A Pwm = Clear Up , Compare B Pwm = Clear up , Prescale = 1024
- Compare1a = 100
- Compare1b = 10

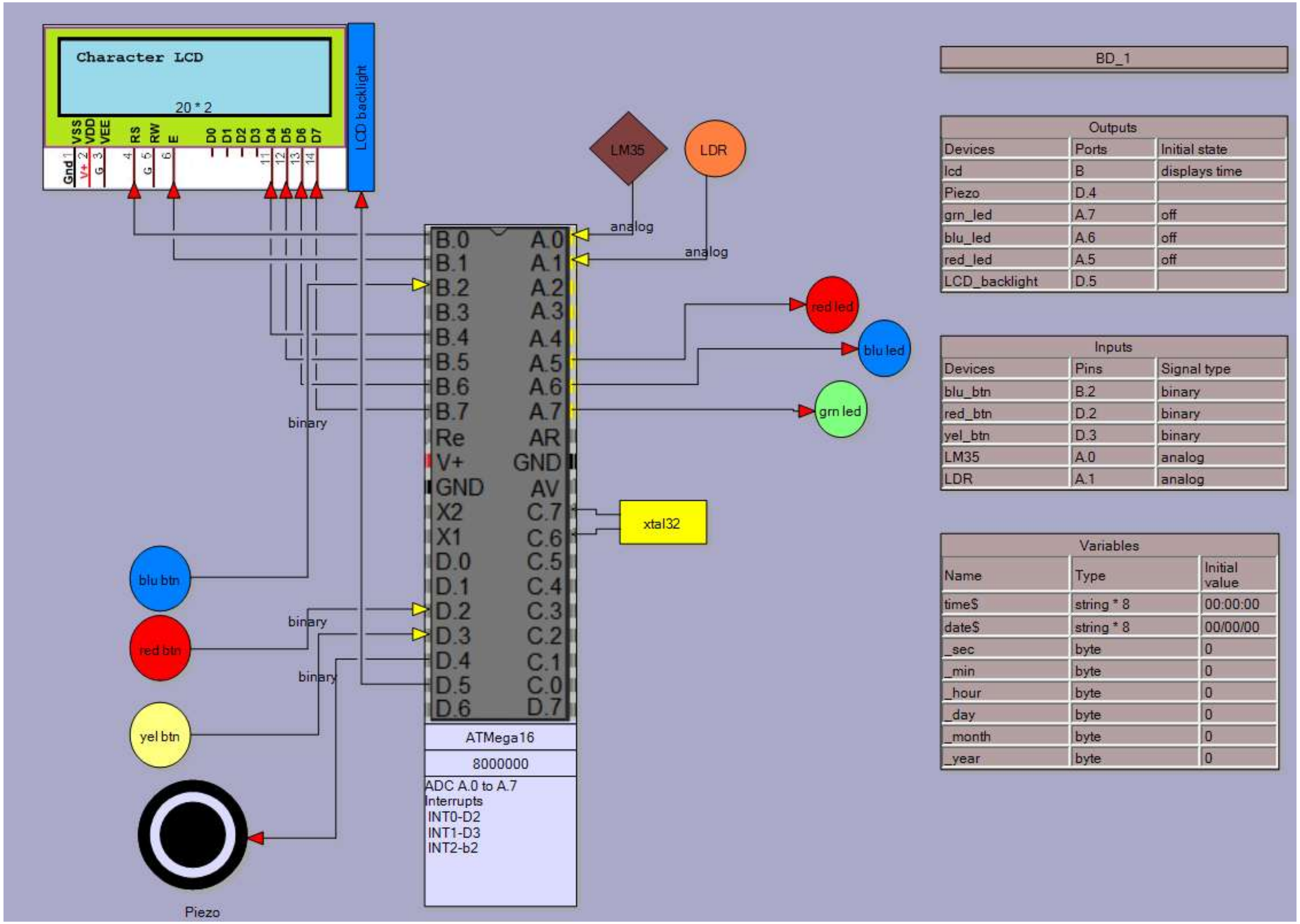
Frequency values for different input crystal and prescale value

OUTPUT FREQUENCY (Hz) for a crystal frequency of 7,372,800

		Prescale Value				
		1	8	64	256	1024
PWM	8 Bit	14,456	1,807	226	56	14
	9 Bit	7,214	902	113	28	7
	10 Bit	3604	450	56	14	4

33 Advanced System Example – Alarm Clock

Bascom has built in functions for managing the time and date. These require a 32.768Khz crystal to be connected to the micro.



In System Designer you can add the crystal to the diagram. Take note that this must go onto the pins shown and that Bascom software routines for the time use Timer2. So it canot be used for anything else.

In the variables table the variables that Bascom creates automatically are avaialbel for you to use within your program.

To use the cryatal and these features add the following 3 lines to your program

```
Config Clock = Soft
Config Date=Mdy, Separator=/
Enable Interrupts
```

In this first program the date and time are displayed on an LCD

```
'SoftClockDemoProgam1.bas
'32.768kHz crystal is soldered onto C.6 and C.7 of an ATMEGA

$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portd = Output
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

Enable Interrupts                 '1 activate internal timer

Config Date = Mdy , Separator = /       '2 you have some choices here
Config Clock = Soft                '3 – note uses internal timer

Date$ = "06/24/09"                 '4 set the date using the Bascom created variable
Time$ = "23:59:56"                 '5 Bascom created variable to store the time

Cls
Cursor Off

Do
  Locate 1 , 1
  Lcd Time$ ; " " ; Date$          '6 display the two strings on the LCD
Loop
End
```

This next program introduces the 1 second interrupt called sectic and the built in Bascom routine to find the day of the week

```
'SoftClockTrialDemoProgam2.bas
$crystal = 8000000
$regfile = "m8535.dat"
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
Grnled Alias Portd.7

Enable Interrupts
Config Date = Mdy , Separator = /
Config Clock = Soft , Gosub = Sectic    '1 - every second automatically interrupt the main dprogram and go and what is in the subrotuine sectic
Dim Strweekday As String * 10          '2 – a string holds texst so we can  display the day of the week
Dim Bweekday as byte
Dim strmonth as String * 10
Date$ = "06/24/09"
Time$ = "23:59:56"
Cls
Cursor Off
Do
  Locate 1 , 1
  Lcd Time$ ; " " ; Date$
  Locate 2 , 1
  Lcd _sec ;" "; _min;" " ; _hour ; _day ; _month ; _year      '3 – these are the other internal Bascom variables you can use

  Bweekday = Dayofweek()            '4 – this Bascom function gives us a number representing which day of the week a date is
  Strweekday = Lookupstr(bweekday , Weekdays)       '5 – WOW – a neat function to look up a table of values, so
  Strmonth – lookupstr(_month, Months)
  Locate 3 , 1
  Lcd Bweekday ; " = " ; Strweekday    '6 display the day of week, first the number of the day, then the string we looked up
  Lcd _month ; " = " ; Strmonth    '7 display the month using lookup as well!
Loop
End
```

```
Sectic:                          '8 – every second your program will stop its noral execution of commands and come here
  Toggle Grnled                        '9 Toggle means, change from 0 to 1 or 1 to 0
Return

Weekdays:                          '10 – this is not program code but fixed data put into the flash program memory for the program to use
Data "Monday" , "Tuesday" , "Wednesday" , "Thursday" , "Friday" , "Saturday" , "Sunday"
 Months:
Data “”, “January”, “February”, …
```

Other neat Bascom functions include:

' DayOfWeek, DayOfYear, SecOfDay, SecElapsed, SysDay, SysSec ,SysSecElapsed

Read a switch and change the time using our own simple debounce function

```
'SoftClockTrialDemoProgam4.bas
$crystal = 8000000
$regfile = "m8535.dat"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Input
Red_sw Alias Pind.2
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
Enable Interrupts
Config Date = Mdy , Separator = /
Config Clock = Soft
Date$ = "06/24/12"
Time$ = "23:59:56"

Cls
Cursor Off
Do
  If Red_sw = 0 Then Gosub Red_pressed   '1put the code into a subroutine not in the main loop this makes the main loop easier to read
  Locate 1 , 1
  Lcd Time$ ; " " ; Date$
Loop
End

Red_pressed:
  Waitms 25                     '2 wait for  any contact bounce to stop (these are cheap switches we use and can bounce a lot)
  Do                            '3 wait for switch release
  Loop Until Red_sw = 1
  Incr _min                     '4 note the position of this statement (the min increases after the switch is released)
  If _ min > 59 then _min=0   '5 if we increase the mins to 60 then it must go back to 0.
Return
```

33.2 Analogue seconds display on an LCD

In this case the analogue is a bar graph that changes with the seconds on the clock.

```
'--------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 25 June 2009
' File Name: softclock4.bas
'--------------------------------------------------------------
' 2. Program Description:
' declaration of subroutines and
' passing values to a subroutine
'--------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000
$regfile = "m8535.dat"
$hwstack = 32
$swstack = 16                        'needed to increase this from the default of 8
$framesize = 24

'4. Hardware Setups
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Input

Config Date = Mdy , Separator = /
Config Clock = Soft

'5. Hardware Aliases
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs =
Portc.0
Config Lcd = 20 * 4

'6. initialise hardware
Enable Interrupts
Cls
Cursor Off
```

```
'7. Declare variables
Dim Row As Byte

'8. Initialise variables
Date$ = "06/24/09"                  'start time and date
Time$ = "23:59:56"
Row = 2                             'row of lcd to display bar graph on


' subroutine that accepts 2 values, the x=number of lines to draw, y=row)
Declare Sub Displaybars (x As Byte , Y As Byte)

'10. Program starts here
Do
  Locate 1 , 1
  Lcd Time$ ; " " ; Date$
  Call Displaybars(_sec , Row)
Loop
End

'11. Subroutines
Sub Displaybars(x As Byte , Y As Byte)

  'this generic routine displays vertical bars along the lcd
  ' 1 bar per digit from 1 to 100
  ' every 5th and 10th bar is bigger
  '  Special LCD Characters
  Deflcdchar 1 , 32 , 32 , 16 , 16 , 16 , 16 , 32 , 32
  Deflcdchar 2 , 32 , 32 , 24 , 24 , 24 , 24 , 32 , 32
  Deflcdchar 3 , 32 , 32 , 28 , 28 , 28 , 28 , 32 , 32
  Deflcdchar 4 , 32 , 32 , 30 , 30 , 30 , 30 , 32 , 32
  Deflcdchar 5 , 32 , 1 , 31 , 31 , 31 , 31 , 1 , 32
  Deflcdchar 6 , 1 , 1 , 31 , 31 , 31 , 31 , 1 , 1

 'variables needed within this sub
  Local Lines As Byte
  Local Fullblocks As Byte
  Local Temp As Byte
  Local Flag As Byte
  Lines = 0
  Fullblocks = 0
  Temp = 0
  Flag = 0

  'start at beginning of the line
  Locate Y , 1

  'Check If Data is within limits (1-100)
  If X > 100 Then
    Lcd "  PROBLEM:DATA>100  "
    Flag = 1                 'problem so don't display
  End If
  If X = 0 Then
    Flag = 1                 'zero so don't bother to display
    Lcd Spc(20)                ' just put in 20 spaces
```

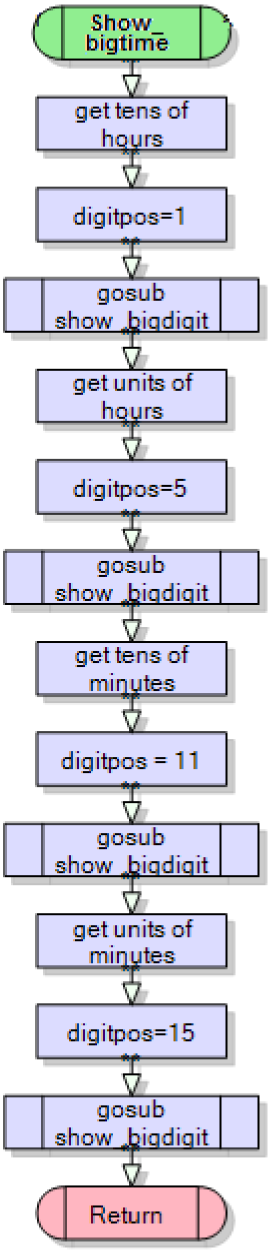
```
  End If

  If Y > 4 Then
    Flag = 1                          'problem so don't display
  End If

  If Flag = 0 Then                    'no problem so display
    'find out how many display blocks need complete filling
    Fullblocks = X - 1
    Fullblocks = Fullblocks / 10
    'fill up the full blocks
    For Temp = 1 To Fullblocks
      Lcd Chr(5)
      Lcd Chr(6)
    Next

    'find out how many more lines to display
    Temp = Fullblocks * 10
    Lines = X - Temp
    'draw the partial block bars
    If Lines < 6 Then                 '
      Select Case Lines
        Case 1 : Lcd Chr(1)          'draw 1 line
        Case 2 : Lcd Chr(2)          'draw 2 lines
        Case 3 : Lcd Chr(3)          'draw 3 lines
        Case 4 : Lcd Chr(4)          'draw 4 lines
        Case 5 : Lcd Chr(5)          'draw 5 lines
      End Select
      Lcd " "
    Else
      Lcd Chr(5)                     'draw 5 lines
      Select Case Lines
        Case 6 : Lcd Chr(1)          'draw 1 line
        Case 7 : Lcd Chr(2)          'draw 2 lines
        Case 8 : Lcd Chr(3)          'draw 3 lines
        Case 9 : Lcd Chr(4)          'draw 4 lines
        Case 10 : Lcd Chr(6)          'draw 5 lines
      End Select
    End If

    'fill to the end with spaces
    Incr Fullblocks
    Incr Fullblocks
    While Fullblocks < 11
      Lcd "  "
     Incr Fullblocks
    Wend
  End If
End Sub
```

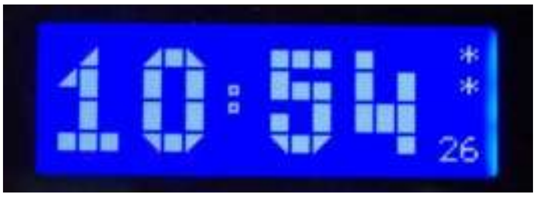
33.3 LCD big digits

In the exercise above large text was to be displayed on the LCD, however it was static, i.e. it wasn't changeable using the program. To display large text on the LCD that is changeable by the program we need to be able to create any character at any location on the display.

This does not mean that we have to setup the letter A at 1,1 in one subroutine and 1,2 in the next and 1,3 in the next. That would be very inefficient; we will ue a variable to determine where on the display the A will be. So in a program we might have the code

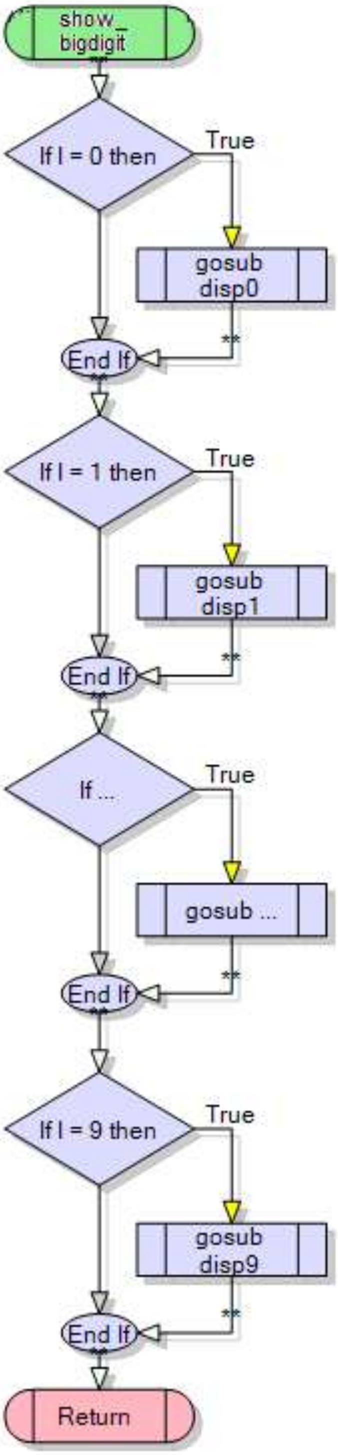
Digitpos=1
Gosub dispA
and
digitpos = 5
gosub dispT





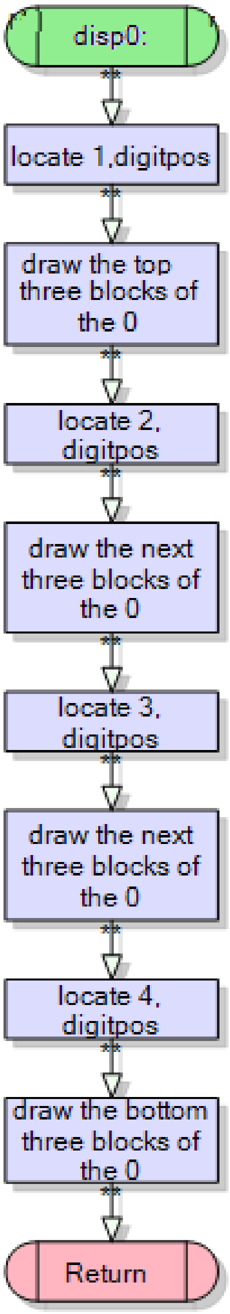
If we wanted to display the time on the LCD this subroutine might be used. First the program must extract the digits from each of hours and minutes. e.g. 23:57 is made up of 2x10 hours and 3 hours, and 5x10 minutes and 7 minutes.
Using knowledge of maths with byte type varibles (there are no fractions) we can divide the variable _hour by 10, to get the value we want.
Dim I as byte ' a temporary variable
I = _hour/10 ' e.g. if _hour = 23 then I will be 2
To get the units of hours we use the mod command, which gives us the remainder of a division in byte math.
I = _hour mod 10 'e.g. if _hour = 23 then I will be 3

```
Show_bigtime:
   'find the digit in the tens of hours position
   I = _hour / 10      'e.g. 19/10 = 1 (byte math!!)
   Digitpos = 1
   Gosub Show_bigdigit
   'find the digit in the units of hours position
   I = _hour Mod 10     'e,g. 19mod10 = 9 (finds remainder)
   Digitpos = 5
   Gosub Show_bigdigit
   'find the digit in the tens of minutes position
   I = _min / 10    'e.g. 21/10 = 2 (byte math!!)
   Digitpos = 11
   Gosub Show_bigdigit
   'find the digit in the units of minutes position
   I = _min Mod 10     'e.g 21mod10 = 1 (finds remainder)
   Digitpos = 15
   Gosub Show_bigdigit
   'display the seconds in the bottom corner of the display
   Locate 4 , 19
   If _sec < 10 Then Lcd "0"
   Lcd _sec
Return
```

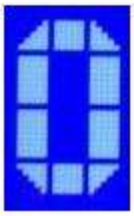


This routine doesn't have all 10 digits shown in the flowchart, however it would need all of them as in thelisting below

```
Show_bigdigit:
    If I = 0 Then Gosub Disp0
    If I = 1 Then Gosub Disp1
    If I = 2 Then Gosub Disp2
    If I = 3 Then Gosub Disp3
    If I = 4 Then Gosub Disp4
    If I = 5 Then Gosub Disp5
    If I = 6 Then Gosub Disp6
    If I = 7 Then Gosub Disp7
    If I = 8 Then Gosub Disp8
    If I = 9 Then Gosub Disp9
Return
```



we have 8 user defined symbols plus a full block, plus
a space we can use to create large digits



top row - 3 different symbols
next row - 2 different symbols (inc space)
next row - same as above
bottom row - 3 different symbols

```
Disp0:
    'line 1
    Locate 1 , Digitpos
    Lcd Chr(1)
    Lcd Chr(2)
    Lcd Chr(3)
    'line 2
    Locate 2 , Digitpos
    Lcd Chr(255)
    Lcd " "
    Lcd Chr(255)
    'line 3
    Locate 3 , Digitpos
    Lcd Chr(255)
    Lcd " "
    Lcd Chr(255)
    'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)
Return
```

Full Listing of the test program

```
'--------------------------------------------------------------------
'Title Block
'Author: BCollis
'Date :  May 2010
'File name: BigDigitTest.V3
'--------------------------------------------------------------------
$crystal = 8000000                        'speed of processing
$regfile = "m8535.dat"                    'our micro
'-------------------------------------------------------------------
'setup/configure hardware
Config Porta = Input
Config Portb = Input                      'switches connected here
Config Pina.4 = Output                    'backlight


'bascom internal features and functions to make a clock in software
```

```
'requires 32,768 Hz crystal on PortC.6 and PortC.7
Config Date = Dmy , Separator = /
Config Clock = Soft , Gosub = Sectic       'with 1 second interrupt configured
Enable Interrupts                          'starts the clock

'setup connection of LCD to micro
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 =
Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

'these characters are used to build the bigdigits
Deflcdchar 1 , 32 , 32 , 32 , 1 , 3 , 7 , 15 , 31
Deflcdchar 4 , 31 , 15 , 7 , 3 , 1 , 32 , 32 , 32
Deflcdchar 2 , 32 , 32 , 32 , 31 , 31 , 31 , 31 , 31
Deflcdchar 3 , 32 , 32 , 32 , 16 , 24 , 28 , 30 , 31
Deflcdchar 5 , 1 , 3 , 7 , 15 , 31 , 32 , 32 , 32
Deflcdchar 6 , 31 , 31 , 31 , 31 , 31 , 32 , 32 , 32
Deflcdchar 7 , 1 , 3 , 7 , 15 , 31 , 31 , 31 , 31
Deflcdchar 0 , 31 , 30 , 28 , 24 , 16 , 32 , 32 , 32

'  Harware Aliases
Lcdbacklight Alias Porta.4
Piezo Alias Portb.0

Yel_btn Alias Pinb.3
Red_btn Alias Pinb.4
Blu_btn Alias Pinb.5
Blk_btn Alias Pinb.6
White_btn Alias Pinb.7

'8. initialise hardware
Cls                                        'Clears screen
Cursor Off                                 'no cursor to be displayed on lcd
Set Lcdbacklight                           'turn on LCD backlight

'------------------------------------------------------------------
' Declare Constants
Const Delay_time = 100
'------------------------------------------------------------------
' Declare Variables

Dim Digitpos As Byte
Dim Seccount As Word
Dim I As Byte

' Initialise Variables
Date$ = "22/07/10"                         'preset time on powerup
Time$ = "03:10:00"
Digitpos = 1
```

```
'-------------------------------------------------------------------
' 12. Program starts here
Do
    Digitpos = 1
    For I = 0 To 9
        Gosub Show_bigdigit
        Waitms 100
    Next
    Gosub Show_smalltime
    Wait 1
    Gosub Show_bigtime
    Wait 1
Loop
'-------------------------------------------------------------------
' Subroutines
Show_smalltime:                    'Display time in small digits so that title
   Locate 2 , 4                          'and the time can fit in to the lcd.
   Lcd "Time: "
   Lcd Time$ ; ""
Return

Show_bigtime:
   'find the digit in the tens of hours position
   I = _hour / 10                        'e.g. 19/10 = 1 (byte arithmentic!!)
   Digitpos = 1
   Gosub Show_bigdigit
   'find the digit in the units of hours position
   I = _hour Mod 10                      'e,g. 19mod10 = 9 (finds remainder)
   Digitpos = 5
   Gosub Show_bigdigit

   Locate 2 , 9
   Lcd Chr(6)
   Locate 3 , 9
   Lcd Chr(2)
   'find the digit in the tens of minutes position
   I = _min / 10                         'e.g. 21/10 = 2 (byte arithmentic!!)
   Digitpos = 11
   Gosub Show_bigdigit
   'find the digit in the units of minutes position
   I = _min Mod 10                       'e.g 21mod10 = 1 (finds remainder)
   Digitpos = 15
   Gosub Show_bigdigit
   'display the seconds in the bottom corner of the display
   Locate 4 , 19
   If _sec < 10 Then Lcd "0"
   Lcd _sec
Return
```

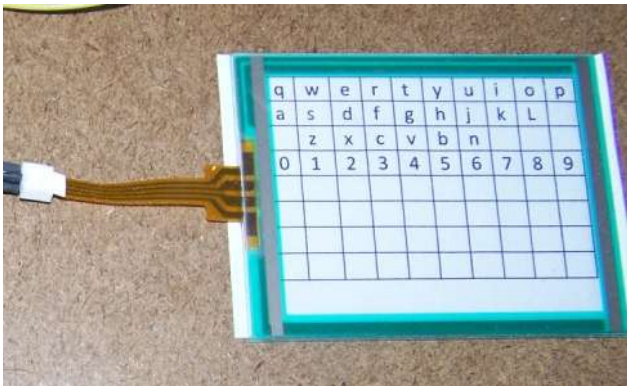
```
Show_bigdigit:
   If I = 0 Then Gosub Disp0
   If I = 1 Then Gosub Disp1
   If I = 2 Then Gosub Disp2
   If I = 3 Then Gosub Disp3
   If I = 4 Then Gosub Disp4
   If I = 5 Then Gosub Disp5
   If I = 6 Then Gosub Disp6
   If I = 7 Then Gosub Disp7
   If I = 8 Then Gosub Disp8
   If I = 9 Then Gosub Disp9
Return

Disp0:
    'line 1
    Locate 1 , Digitpos
    Lcd Chr(1)
    Lcd Chr(2)
    Lcd Chr(3)
    'line 2
    Locate 2 , Digitpos
    Lcd Chr(255)
    Lcd " "
    Lcd Chr(255)
    'line 3
    Locate 3 , Digitpos
    Lcd Chr(255)
    Lcd " "
    Lcd Chr(255)
    'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)

Return

Disp1:
   'line 1
   Locate 1 , Digitpos
   Lcd " "
   Lcd Chr(1)
   Lcd " "
   'line 2
   Locate 2 , Digitpos
   Lcd Chr(5)
   Lcd Chr(255)
   Lcd " "
   'line 3
   Locate 3 , Digitpos
   Lcd " "
   Lcd Chr(255)
   Lcd " "
   'line 4
   Locate 4 , Digitpos
   Lcd Chr(6)
   Lcd Chr(6)
   Lcd Chr(6)
Return


Disp2:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(1)
   Lcd Chr(2)
   Lcd Chr(3)
   'line 2
   Locate 2 , Digitpos
   Lcd Chr(6)
   Lcd " "
   Lcd Chr(255)
   'line 3
   Locate 3 , Digitpos
   Lcd Chr(7)
   Lcd Chr(6)
   Lcd Chr(0)
   'line 4
   Locate 4 , Digitpos
   Lcd Chr(6)
   Lcd Chr(6)
   Lcd Chr(6)
Return

Disp3:
   'line 1
    Locate 1 , Digitpos
    Lcd Chr(1)
    Lcd Chr(2)
    Lcd Chr(3)
   'line 2
    Locate 2 , Digitpos
    Lcd " "
    Lcd Chr(2)
    Lcd Chr(255)
   'line 3
    Locate 3 , Digitpos
    Lcd " "
    Lcd " "
    Lcd Chr(255)
   'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)
Return
```

```
Disp4:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(2)
   Lcd " "
   Lcd " "
   'Line 2
   Locate 2 , Digitpos
   Lcd Chr(255)
   Lcd " "
   Lcd Chr(255)
   'line 3
   Locate 3 , Digitpos
   Lcd Chr(255)
   Lcd Chr(255)
   Lcd Chr(255)
   Locate 4 , Digitpos
   Lcd "  "
   Lcd Chr(6)
Return

Disp5:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(2)
   Lcd Chr(2)
   Lcd Chr(2)
   'line 2
   Locate 2 , Digitpos
   Lcd Chr(255)
   Lcd Chr(2)
   Lcd Chr(2)
   'line 3
   Locate 3 , Digitpos
   Lcd Chr(2)
   Lcd " "
   Lcd Chr(255)
   'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)
Return

Disp6:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(1)
   Lcd Chr(2)
   Lcd Chr(3)
   'Line 2
   Locate 2 , Digitpos
   Lcd Chr(255)
   Lcd Chr(2)
   Lcd Chr(3)
   'line 3
   Locate 3 , Digitpos
   Lcd Chr(255)
   Lcd " "
   Lcd Chr(255)
   'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)
Return

Disp7:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(1)
   Lcd Chr(2)
   Lcd Chr(3)
   'line 2
   Locate 2 , Digitpos
   Lcd "  "
   Lcd Chr(255)
   'line 3
   Locate 3 , Digitpos
   Lcd "  "
   Lcd Chr(255)
   'line 4
   Locate 4 , Digitpos
   Lcd "  "
   Lcd Chr(4)
Return

Disp8:
    'line 1
    Locate 1 , Digitpos
    Lcd Chr(1)
    Lcd Chr(2)
    Lcd Chr(3)
    'line 2
    Locate 2 , Digitpos
    Lcd Chr(255)
    Lcd Chr(2)
    Lcd Chr(255)
    'line 3
    Locate 3 , Digitpos
    Lcd Chr(255)
    Lcd " "
    Lcd Chr(255)
    'line 4
    Locate 4 , Digitpos
    Lcd Chr(4)
    Lcd Chr(6)
    Lcd Chr(0)
Return
```

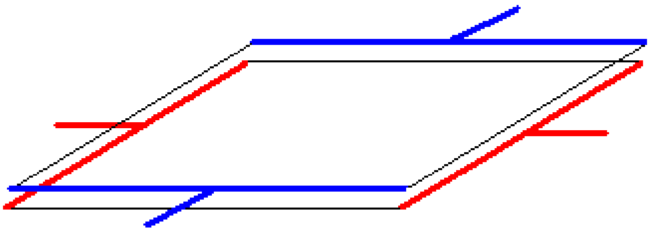
```
Disp9:
   'line 1
   Locate 1 , Digitpos
   Lcd Chr(1)
   Lcd Chr(2)
   Lcd Chr(3)
   'line 2
   Locate 2 , Digitpos
   Lcd Chr(255)
   Lcd " "
   Lcd Chr(255)
   'line 3
   Locate 3 , Digitpos
   Lcd Chr(4)
   Lcd Chr(6)
   Lcd Chr(255)
   'line 4
   Locate 4 , Digitpos
   Lcd Chr(4)
   Lcd Chr(6)
   Lcd Chr(0)
Return


Sectic:
   Incr Seccount
Return
```

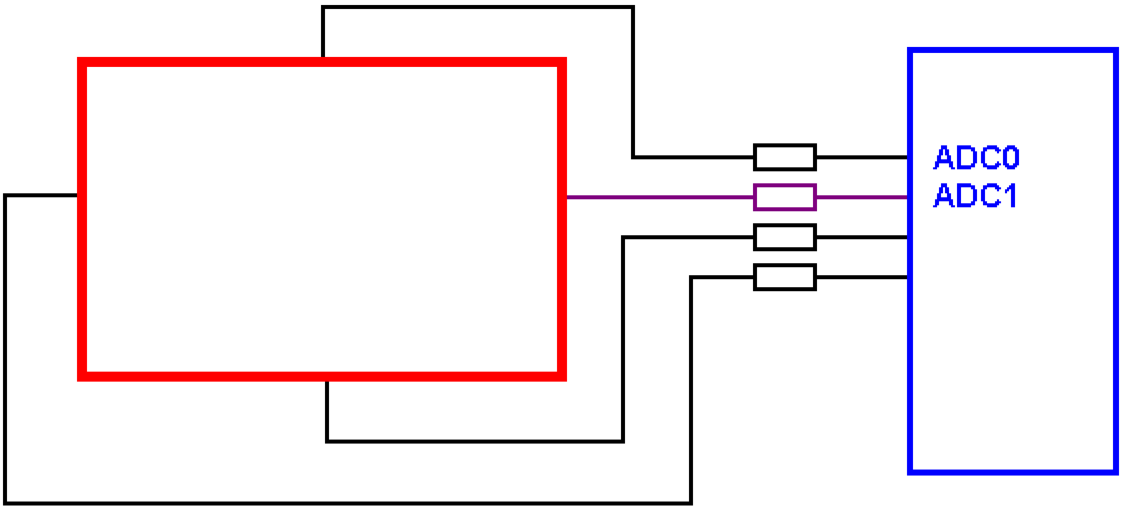
34 Resistive touch screen

The resistive touch screen is made of several layers all transparent.

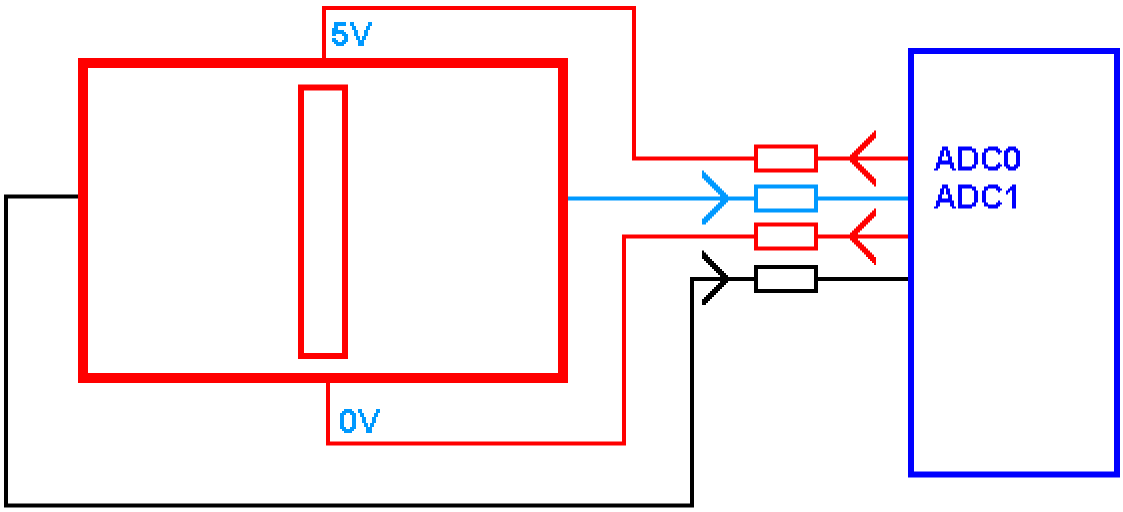


There are two resitive layers that when pressed together conduct. The resistance is measured by passing a current through one layer and measuring the voltage on the other layer.The first stage is to wire the 4 connections to the microcontroller, at least two adjacent pins must be connected to the ADC input pins.

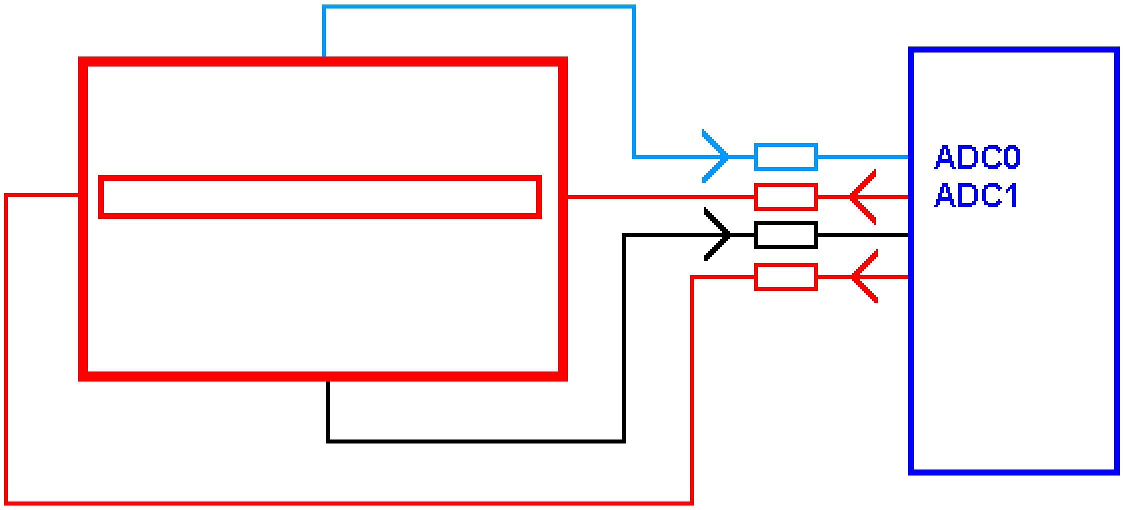




Connect the 4 wires of the touch pad to the micro
At least 2 of these must be ADC so they can be read
the others can be ordinary i/o pins
the 4 resistors limit the current and can be approx 100R

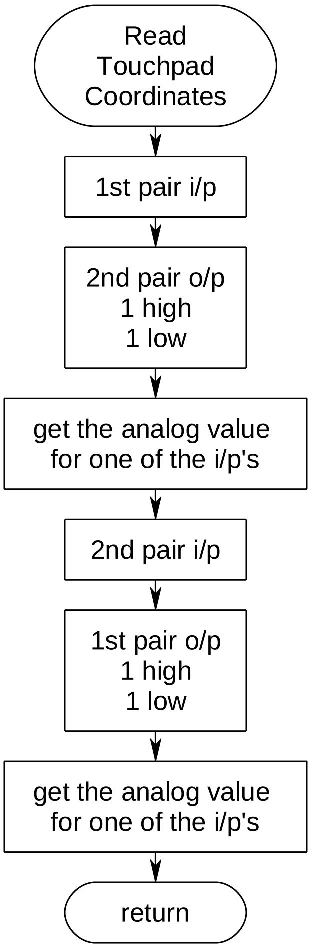


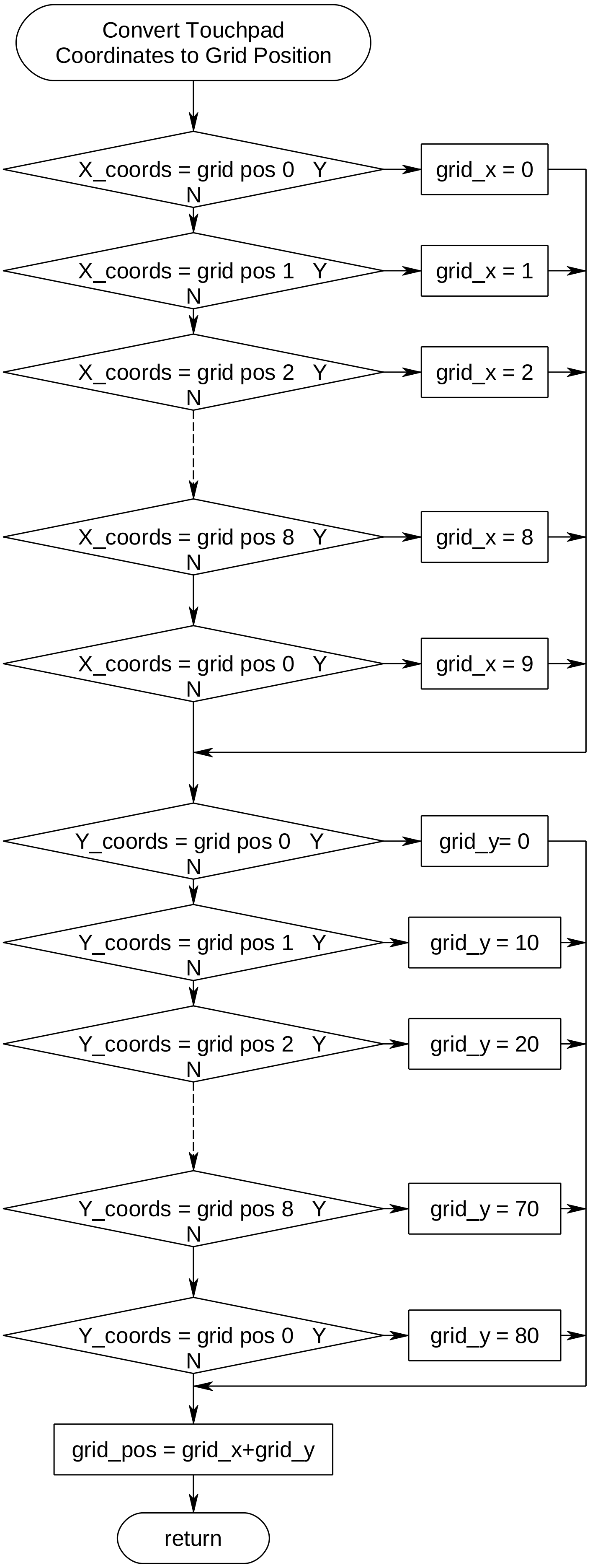
set 2 pins as i/p, 2 as o/p,
put a high on 1 o/p and a low on the other o/p (set and reset)
read one of the inputs e.g. getadc(1)
the input reading will represent the vertical position



reverse the 2 i/p and 2 o/p pin configurations
put a high on 1 o/p and a low on the other o/p (set and reset)
read one of the i/p e.g. getadc(0)
the value represents the horizontal position

Following are the flowcharts for the routines to read the touch screen coordinates and then convert these to a grid position.


Read Touchpad Coordinates
1st pair i/p
2nd pair o/p
1 high
1 low
get the analog value for one of the i/p's
2nd pair i/p
1st pair o/p
1 high
1 low
get the analog value for one of the i/p's
return


Convert Touchpad Coordinates to Grid Position
X_coords = grid pos 0
Y
N
grid_x = 0
X_coords = grid pos 1
grid_x = 1
X_coords = grid pos 2
grid_x = 2
X_coords = grid pos 8
grid_x = 8
X_coords = grid pos 0
grid_x = 9
Y_coords = grid pos 0
grid_y= 0
Y_coords = grid pos 1
grid_y = 10
Y_coords = grid pos 2
grid_y = 20
Y_coords = grid pos 8
grid_y = 70
Y_coords = grid pos 0
grid_y = 80
grid_pos = grid_x+grid_y
return

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: April 2008
' File Name: touchscreen_V2.bas
'-----------------------------------------------------------------
' 2. Program Description:
' Touch Screeen on PortA.5 to PortA.7
'-----------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$map
$crystal = 8000000                   'the speed of the micro
$regfile = "m8535.dat"               'our micro, the ATMEGA8535-16PI
'-----------------------------------------------------------------
' 4. Hardware Setups
' 5. Hardware Aliases
' 6. initialise ports so hardware starts correctly
' DDRA is the internal register that controls the ports
Ddra = &B00000000                    'all pins set as inputs
'LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4                  'configure lcd screen
'ADC
Config Adc = Single , Prescaler = Auto
Start Adc
'-----------------------------------------------------------------
' 7. Declare Constants
'-----------------------------------------------------------------
' 8. Declare Variables
Dim X_coord As Word
Dim Y_coord As Word
Dim I As Byte
Dim J As Byte
Dim Gridposition As Byte
Dim Character As String * 2
' 9. Initialise Variables

'-----------------------------------------------------------------
' 10. Program starts here
Cursor Off
Cls
Do
  Gosub Readtouchcoords              'get the values for the touch area
  Locate 1 , 1
  Lcd "x=" ; X_coord ; "   "         'display x-coordinate
  Locate 2 , 1
  Lcd "y=" ; Y_coord ; "   "         'display y-coordinate
  Gosub Getgridposition              'turn coordinates into grid
  Locate 3 , 1
  Lcd "                    "
  Locate 3 , 1
  If Gridposition < 90 Then          'only if valid press
    Lcd Gridposition ; "  "
    If Gridposition < 40 Then        'only lookup if valid character
      Character = Lookupstr(gridposition , Characters)
      Lcd Character ; " "
    End If
    Waitms 500                       'holds the value on the screen a bit
  End If
Loop
```

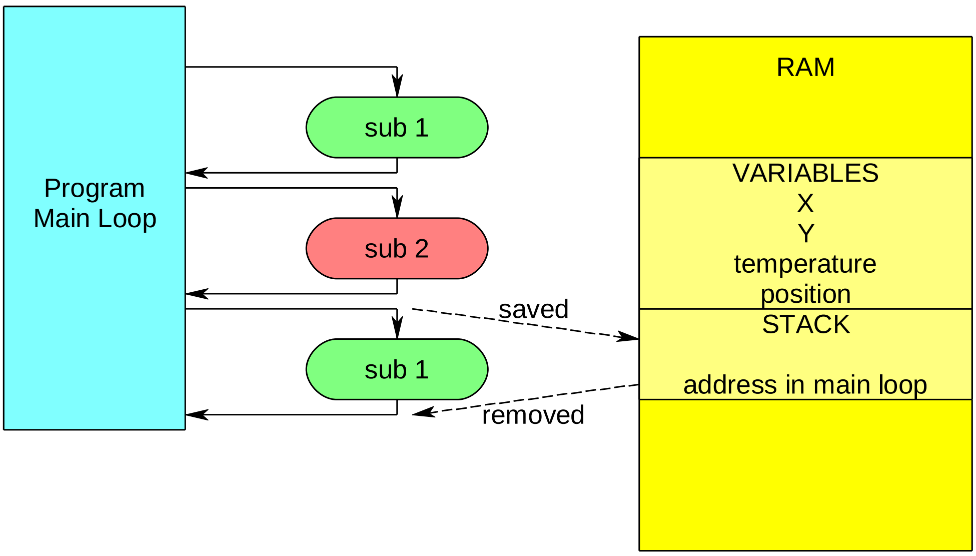
```
End

'----------------------------------------------------------------
' 11. Subroutines

Getgridposition:
'returns a grid number from 0 to 89
'depending on where touch is within the touch area
'otherwise returns 90
' _______________________________
'| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|
'|10|11|12|13|14|15|16|17|18|19|
'|20|21|22|23|24|25|26|27|28|29|
'...
'|70|71|72|73|74|75|76|77|78|79|
'|80|81|82|83|84|85|86|87|88|89|
' _______________________________
'the values below were worked out by trial and error!
  Select Case X_coord
    Case 100 To 170 : I = 0
    Case 171 To 270 : I = 1
    Case 271 To 360 : I = 2
    Case 361 To 450 : I = 3
    Case 451 To 530 : I = 4
    Case 531 To 610 : I = 5
    Case 611 To 700 : I = 6
    Case 701 To 790 : I = 7
    Case 791 To 870 : I = 8
    Case 871 To 999 : I = 9
    Case Else : I = 90
  End Select
  Select Case Y_coord
    Case 100 To 240 : J = 80
    Case 241 To 320 : J = 70
    Case 321 To 410 : J = 60
    Case 411 To 500 : J = 50
    Case 501 To 580 : J = 40
    Case 581 To 670 : J = 30
    Case 671 To 750 : J = 20
    Case 751 To 850 : J = 10
    Case 851 To 920 : J = 0
    Case Else : J = 90
  End Select
  Gridposition = I + J
  If Gridposition > 89 Then Gridposition = 90
Return
```

```
Readtouchcoords:
'finds the position of a touch on a 4 wire resistive touch pad
'first by making 1 pair of wires  outputs and measuring
'one of the others as an analogue to digital input
'then swaps the 2 i/p's for the 3 o/p's and repeats the process
   Ddra.4 = 1                          'output
   Ddra.5 = 0                          'input
   Ddra.6 = 1                          'output
   Ddra.7 = 0                          'input
   Set Porta.4                         '1=5V
   Reset Porta.6                         '0=0V
   Waitms 10                            'short delay to settle pins
   X_coord = Getadc(5)                       'somevalue from 0 & 1023

   Ddra.4 = 0                          'input
   Ddra.5 = 1                          'output
   Ddra.6 = 0                          'input
   Ddra.7 = 1                          'output
   Set Porta.5                         '1= 5V
   Reset Porta.7                         '0=0V
   Waitms 10                            'short delay to settle pins
   Y_coord = Getadc(4)                       'somevalue from 0 & 1023
Return

'each character below maps to one of the grid positions in the first 4 rows
Characters:
Data "q" , "w" , "e" , "r" , "t" , "y" , "u" , "i" , "o" , "p"
Data "a" , "s" , "d" , "f" , "g" , "h" , "j" , "k" , "l" , " "
Data " " , "z" , "x" , "c" , "v" , "b" , "n" , "m" , " " , " "
Data "0" , "1" , "2" , "3" , "4" , "5" , "6" , "7" , "8" , "9"
```

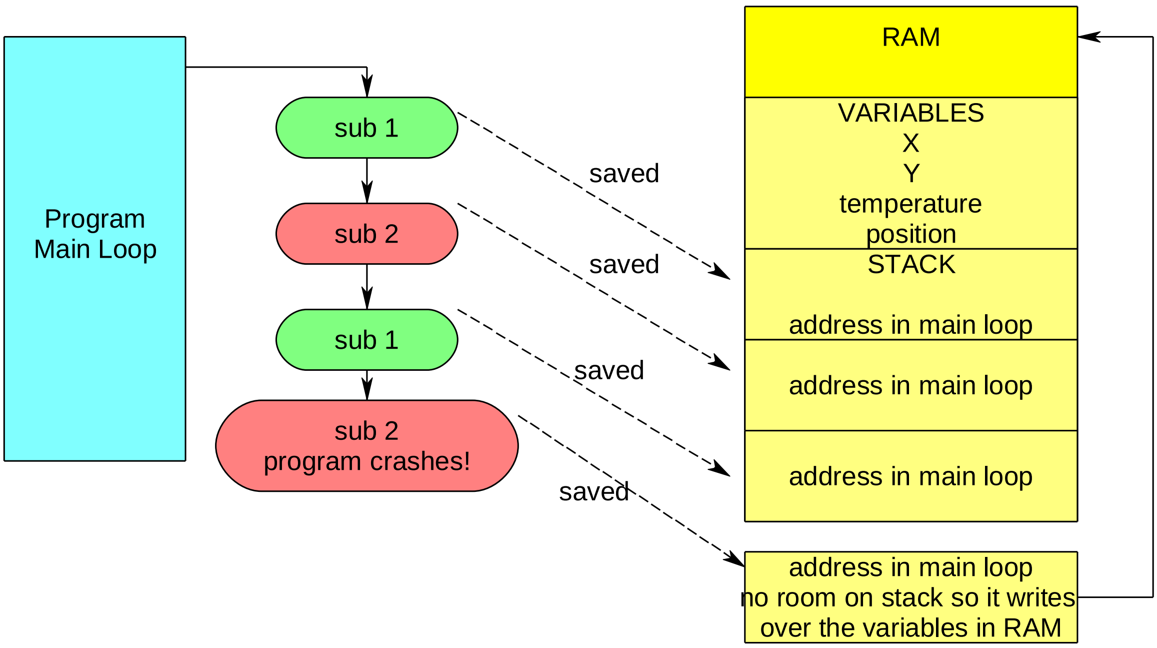
34.1 Keeping control so you dont lose your ‘stack’

As students begin to develop projects they seldom take a big picture approach to what is required; often a system’s components are seen as separate objects that will just fit together and the important relationships (interdependencies) between these objects are missed. In practice this is seen when a project is started with a simple or familiar I/O component such as an LCD and code is written for that device. Then another I/O device is added to the project such as a temperature sensor or a switch and more code is written; then another I/O device is added; at some stage though the programming begins to break down. Many of the I/O functions may be coded at this stage but there is little appreciation for the overriding control nature of the system as it has not been planned from the beginning.

Often around this stage the project will have a number of subroutines, and a problem arises where the program crashes after it has been running for a short time or after a certain number of things have happened such as switch presses. A common fault that causes this is treating subroutine calls (GOSUBs) in a similar way to GOTO statements (which are not allowed). In a microcontroller there is a portion of the RAM set aside by the compiler as the STACK, it is used by the compiler to manage program flow. It exists as a portion of RAM after the variables and may grow downwards towards the end of RAM. When a subroutine is entered, the stack is used to remember the address in main memory where code was running so that when the subroutine exits the program may restart at the correct address in the main code.



When a program leaves a subroutine for another subroutine the stack grows, ultimately however when too many subroutines are called the stack overflows around into the top of RAM overwriting variables.

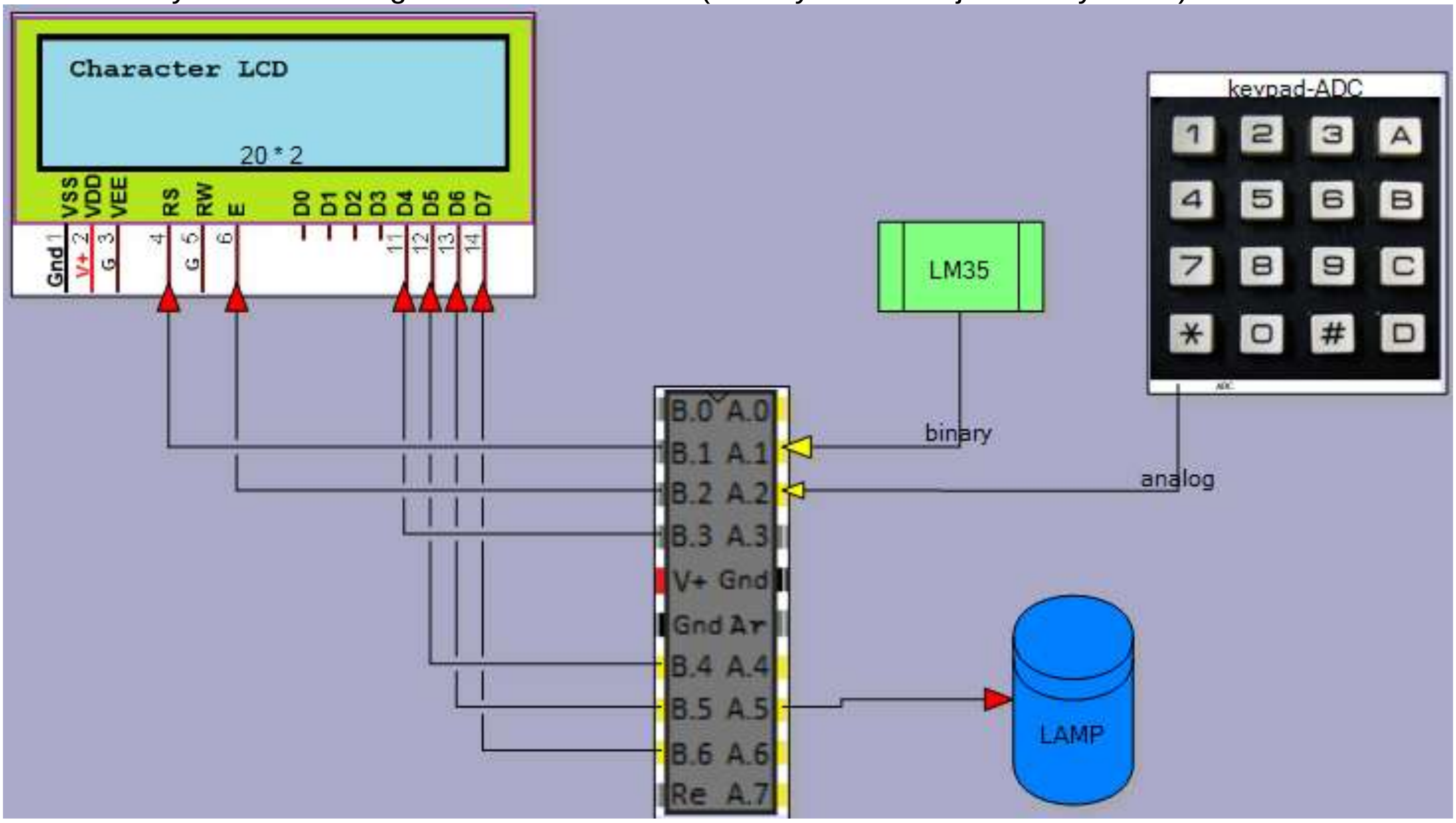


After some time helping students with their code I have recognised this as “my program crashes after I press the switch 6 times” or “after a while it just stops working”. It is before this stage that the designer needs to step back and redesign the control process for the project.

35 System Design Example – Temperature Controller

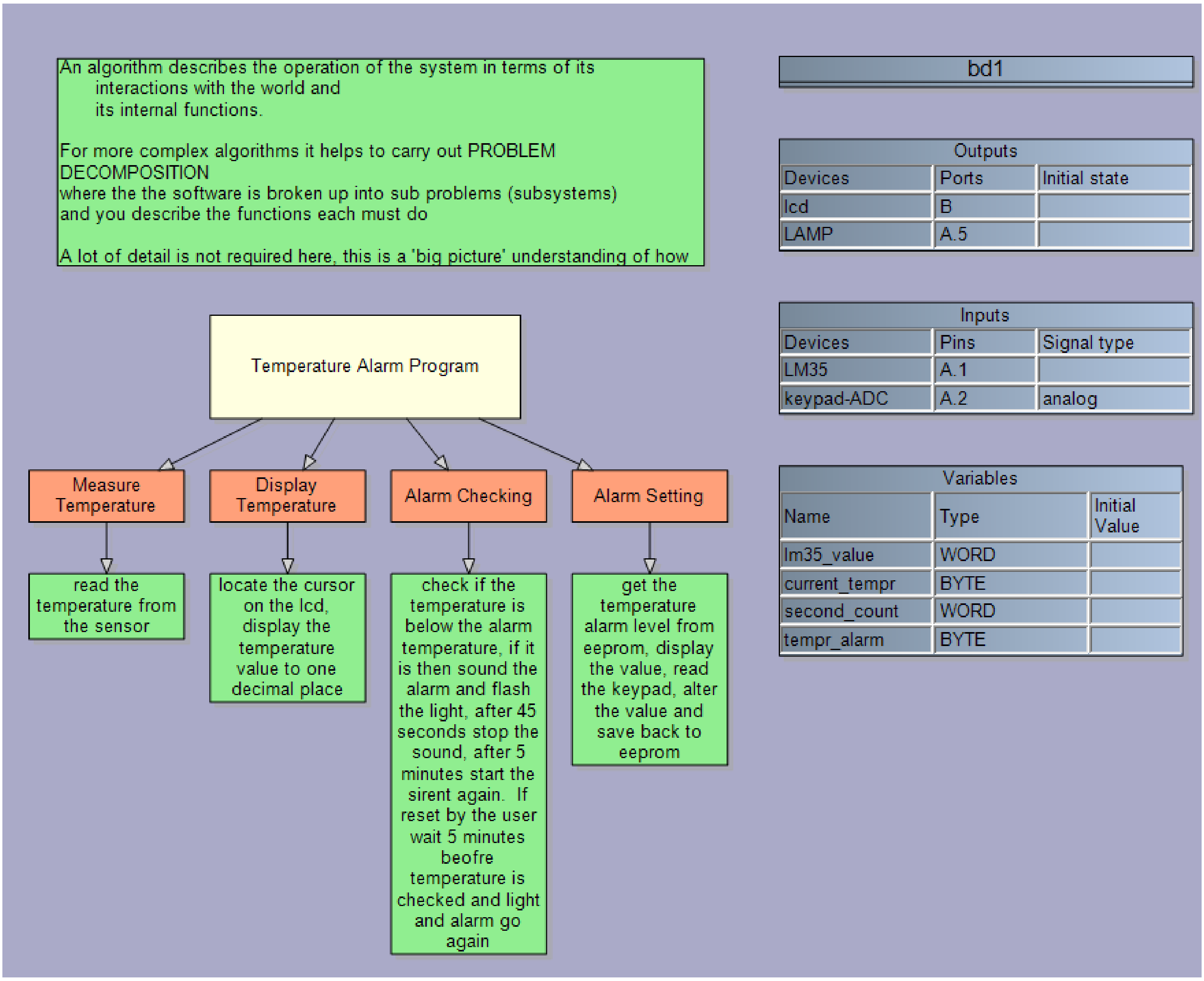
Here is a more complex system that we will develop the software for

1. Define a conceptual statement for the solution to the problem, e.g.
   The system will monitor temperature inside a room and display it on an LCD, an alarm will sound for 45 seconds if it goes below a user preset value. A light will stay flashing until reset. If not reset within 5 minutes the alarm will retrigger again. If the temperature rises at any time then the alarm will automatically reset.
2. Draw a system block diagram of the hardware (identify all the major sub-systems)



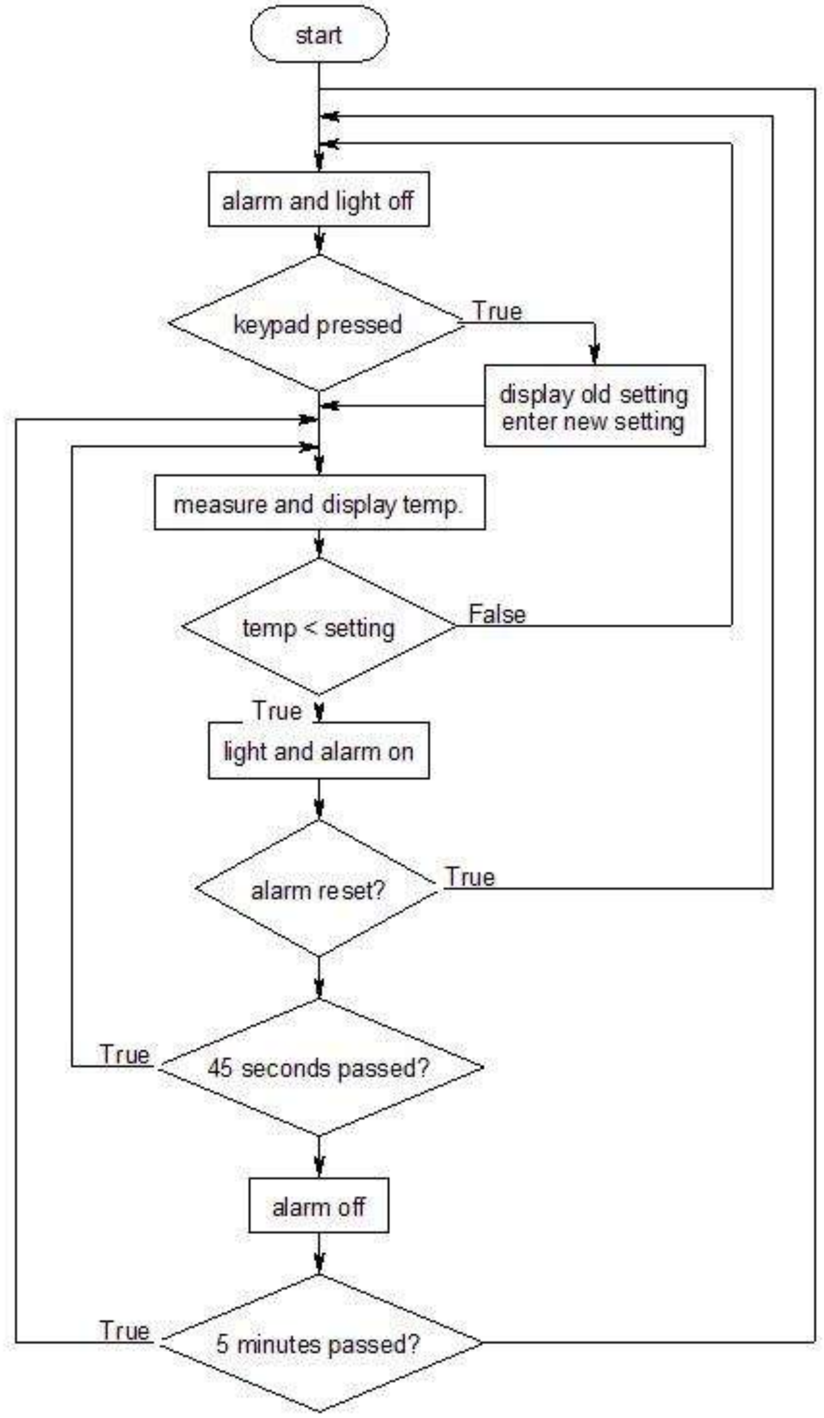
3. Research and identify the interfaces to the system e.g.
   a. An LM35 temperature sensor
   b. A 2 line x 16 character LCD
   c. A flashing light that can be seen from 6 meters away
   d. A speaker with sufficient volume to be heard in the next room
   e. A keypad for entering values
4. Draw interface circuits for each of the interfaces
5. Build the interfaces one at a time, design test subroutines for them and test them thoroughly

6. Problem decomposition stage: break the software for the system down into successive sub-systems, until the sub-systems are trivial (simple) in nature. In this diagram the systems function has been broken down into 4 parts of which one has been broken down further.



7. Design the logic flow for the solution using flow or state diagrams
Test your logic thoroughly! If you miss an error now you will take 219.2 times longer to fix it than if you do not fix it now!!!

Here is a possible flowchart for the temperature system.



This is a small but very complex flowchart and it is not a good solution for a number of reasons:

A. It is difficult to manage all the relationships to get the logic absolutely correct, it took a while to think it through and it may not be exactly right yet!
B. Because the loops in the flowchart overlap it is not possible to write a program without the use of goto statements which are poor (terrible, abysmal, horrible) programming practice and not a feature of the higher level languages you will meet in the future.
C. Once the code is written it is difficult to maintain this code as it lacks identifiable structure

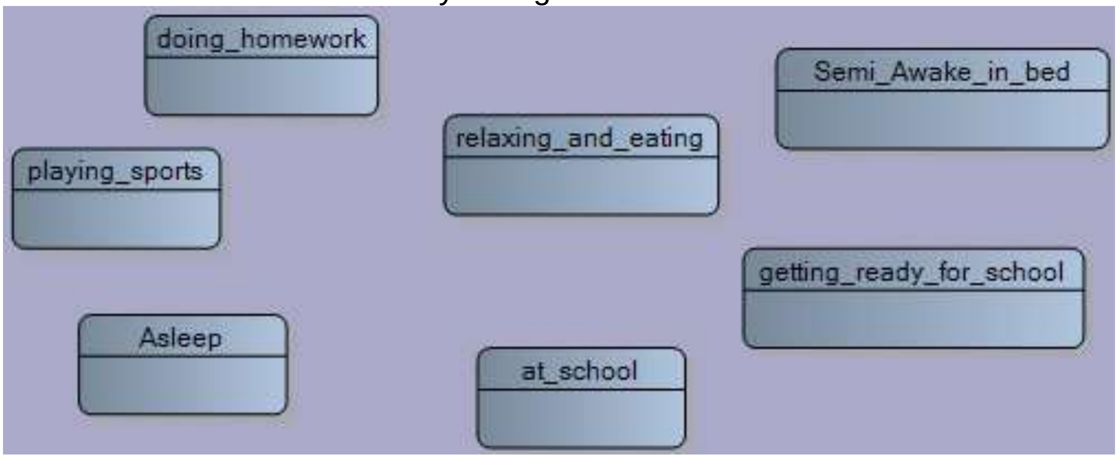
It is OK to use flowcharts for small problems with only a few variable tests but by attempting to put too much logic into a flowchart you astronomically increase the difficulty of turning it into program code; if a flowchart has more than 3 or 4 loops or the loops cross over each other as above use an alternative method!

36 Advanced programming - state machines

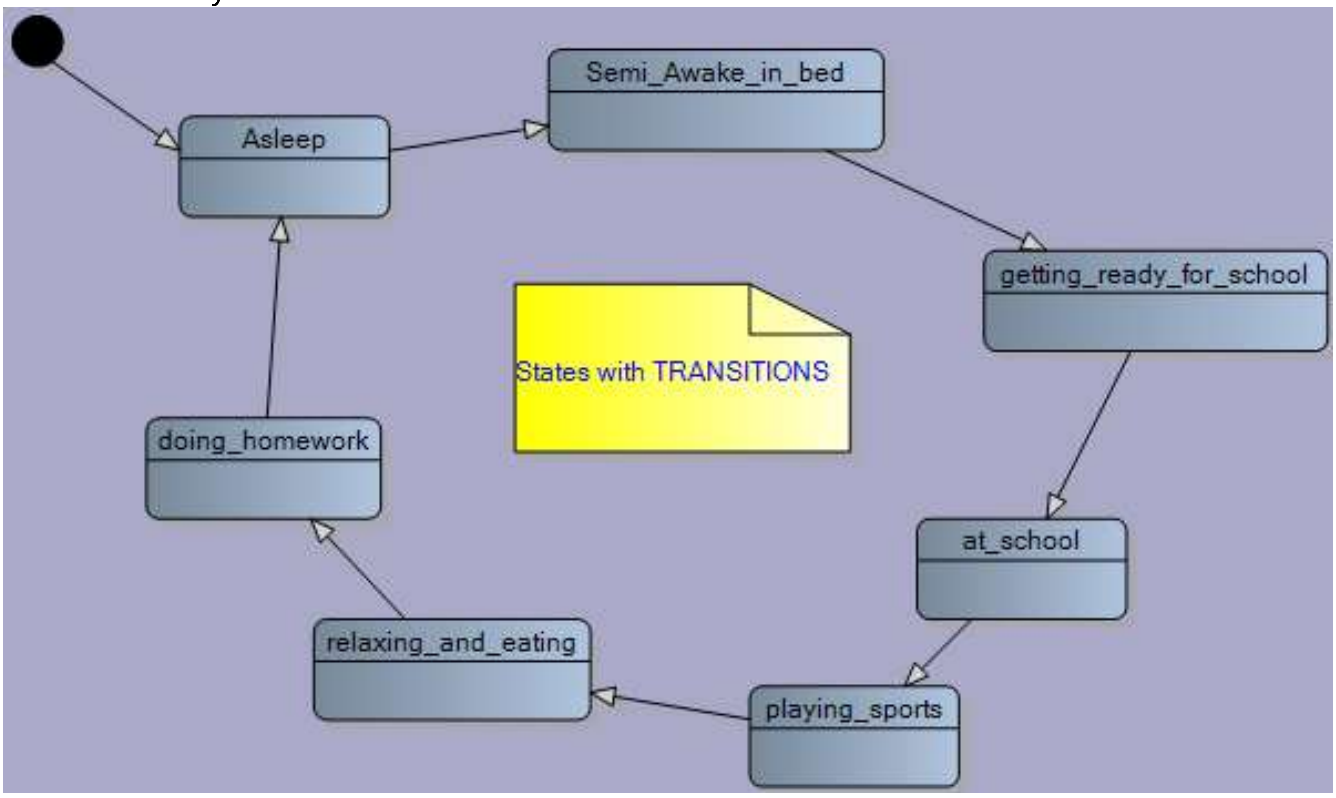
State machines are very different to flowcharts; a flowchart looks primarily at the process operating within a system a state machine looks primarily at the state the system is in and then the processes that support those states. These diagrams have been used extensively in industry for modelling systems and software behaviour for a long time. They are one of the 7 behaviour modelling diagrams in the UML (unified modelling language) specification from OMG (Object Management Group – a consortium of software organsiations). State machines are much better at modelling software than flowcharts because our systems react to inputs and events that can vary at anytime whereas a flowchart is not as responsive to this type of behaviour. Note in UML specification 2.2 OMG have changed the name from statechart back to state machine diagram so if you hear the term statechart it means the same thing.

36.1 Daily routine state machine

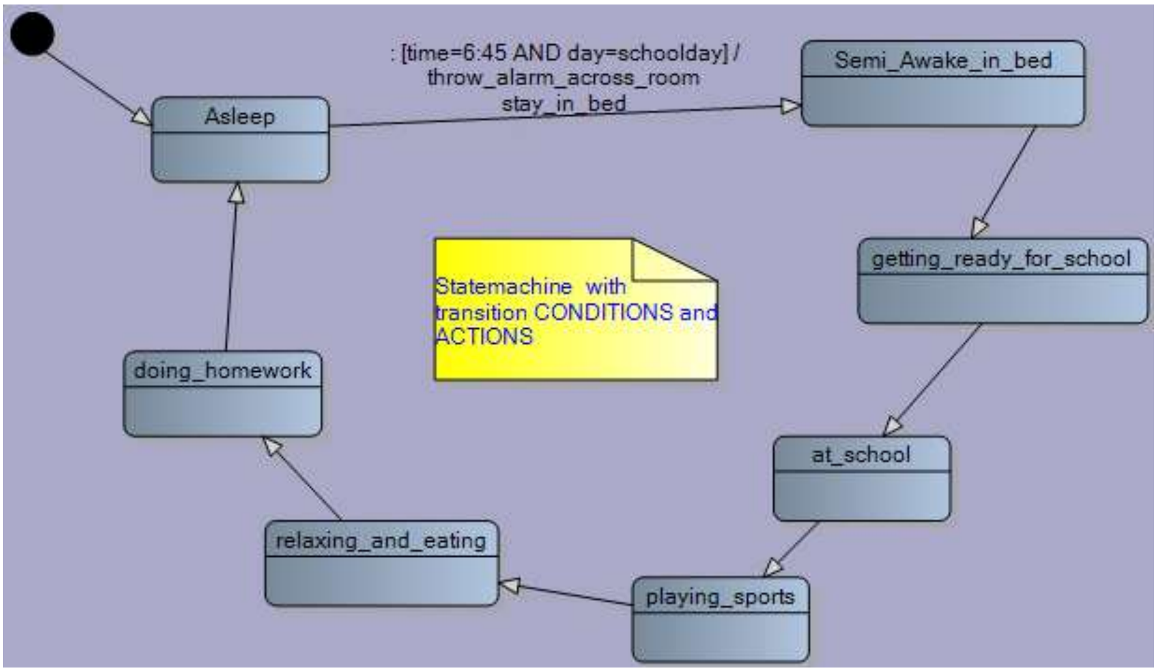
Earlier we looked at a flowchart for a daily routine. Lets develop a state machine for a school day. Here are some different states you might be in.



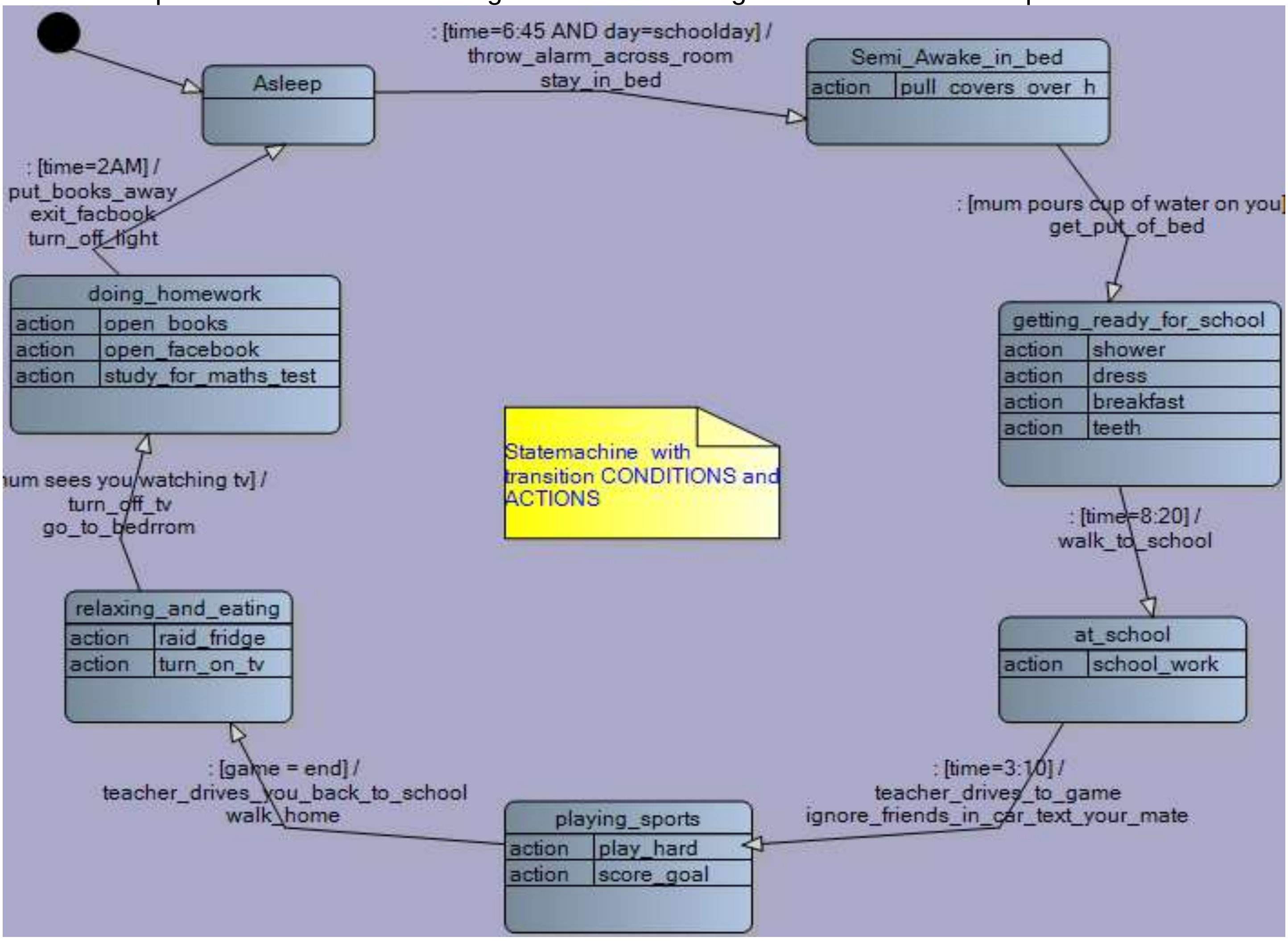
You **transition** from one state to another as the day progresses, The black circle represents which state you start the day in.



Transitions normally occurred when triggered by some event or condition. Here is one possible transition **condition** and an associated transition **action.**



The transition **condition** is time=6:45 AND day=school day.
The transition **actions** are throw alarm clock across room and stay in bed.
If we develop this a little further we might see the following state machine develop.

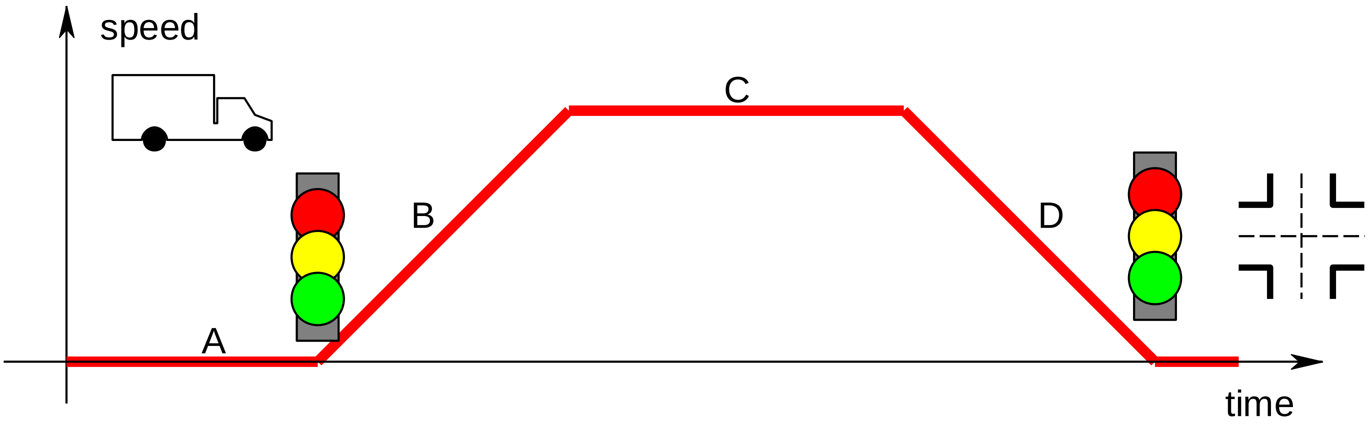


Now although this is a state machine it is not necessary to use a state machine to develop this system; you can see that there are no choices in it so a simple flowchart would be just as useful. It does however show how to start using state machines.

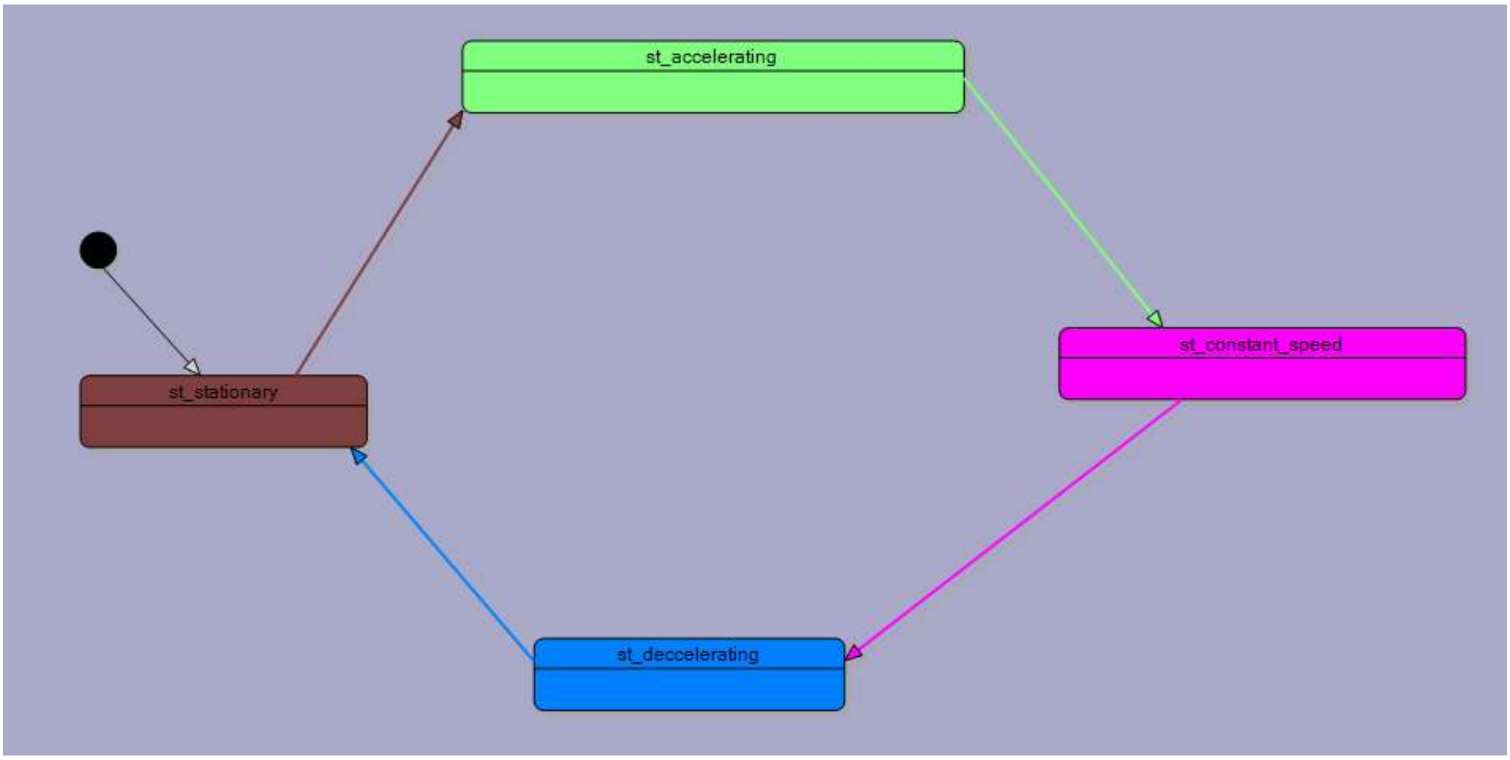
36.2 Truck driving state machine

Lets look at a second example for a state machine based system and introduce how a state machine is more suitable for reactive systems and so much easier than a flowchart.

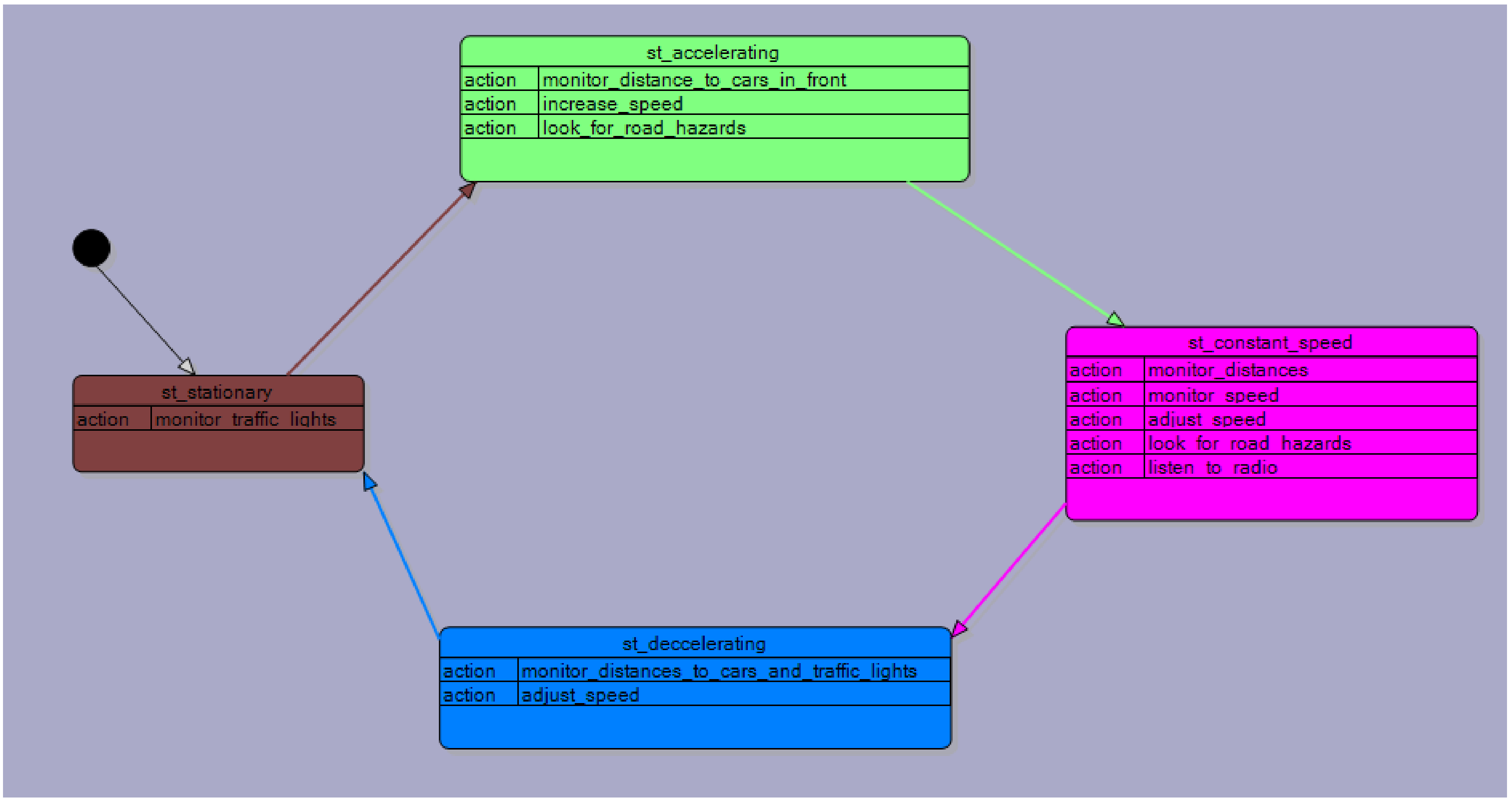
Think of a truck driving around town and its speed as it moves from one set of traffic lights to another. It could be represented by a graph of speed versus time. The truck has 4 states:
A: stationary
B: accelerating
C: constant speed of 50km/hr
D: decelerating



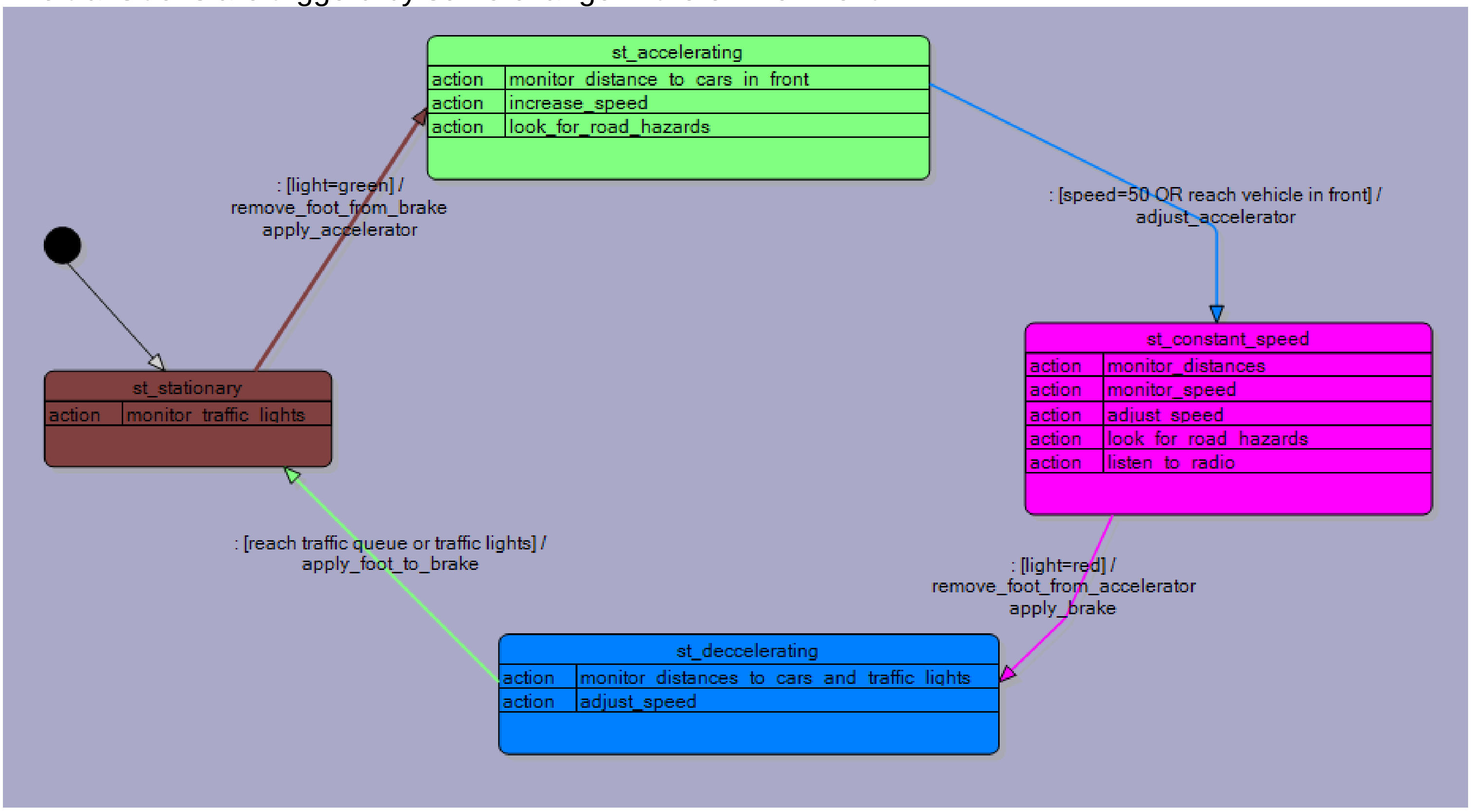
Here is the beginning state machine, note the flow of the diagram..



Here is the state machine with some **actions** within each state. These are things that have to be repeated while the machine is in that state.

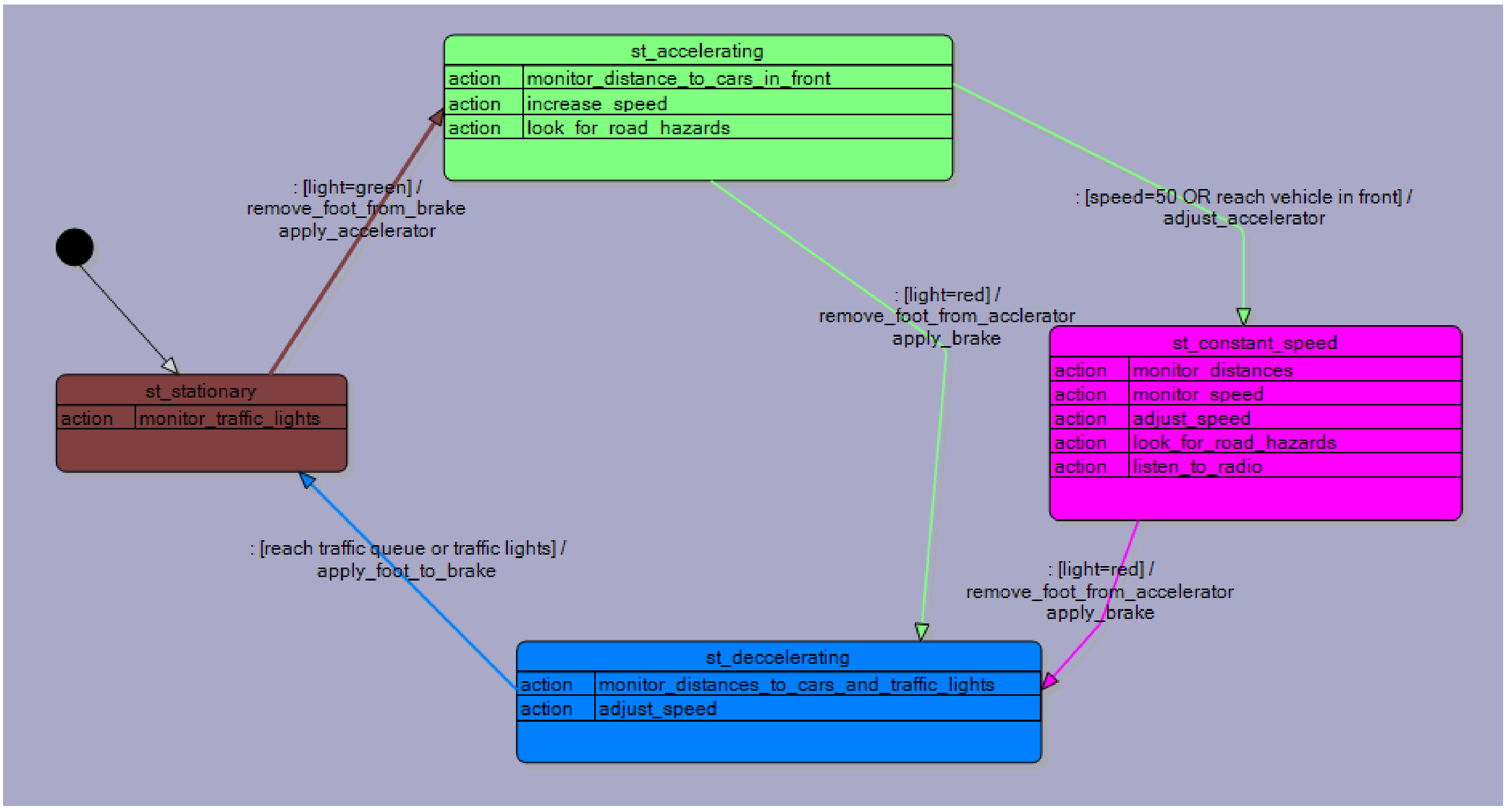


Here is the state machine with **transitions,** some **conditions** and their associated **actions**.
The transitions are triggerd by some change in the environment.

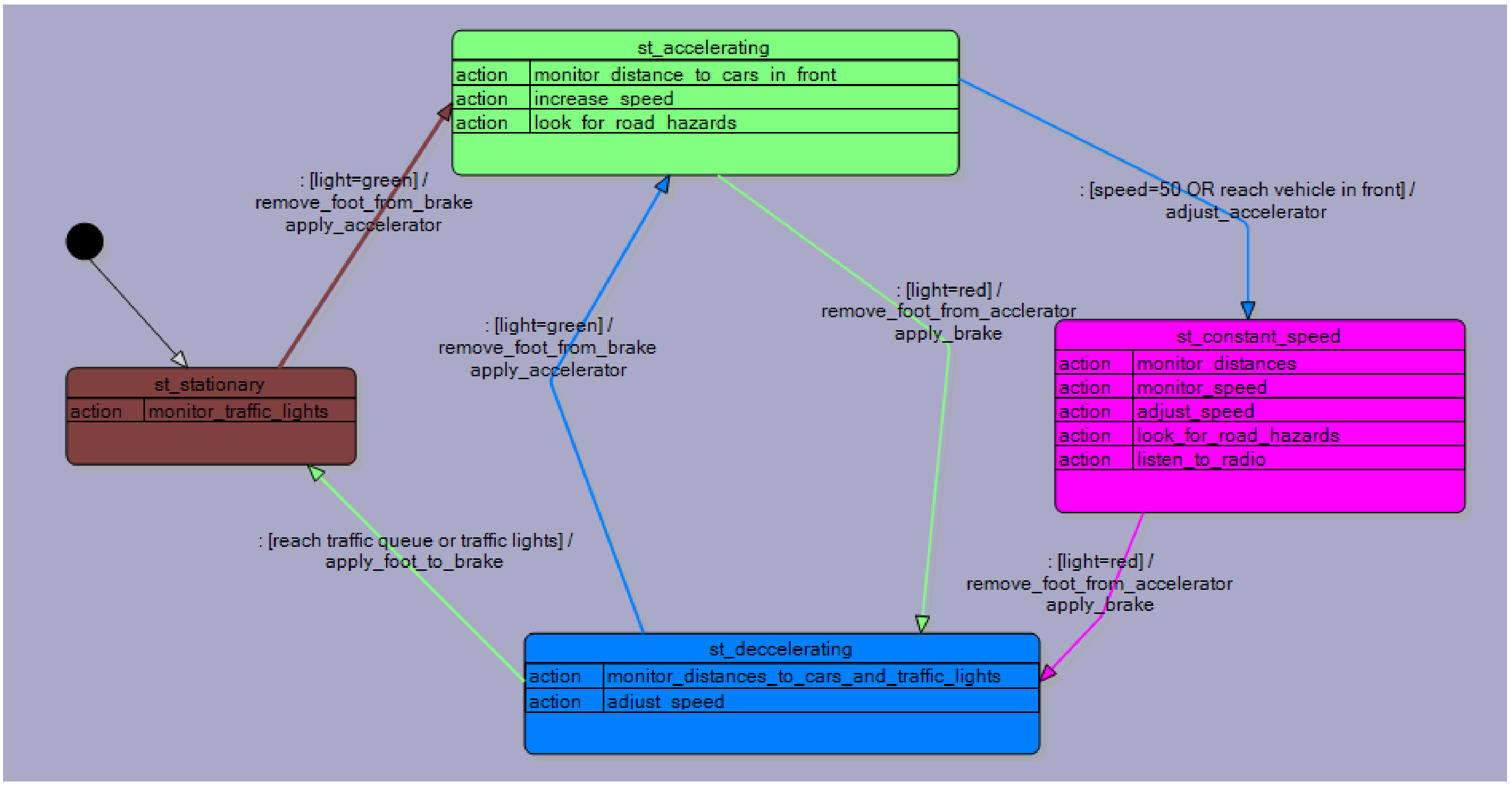


The flow at this stage is still very linear, however that doesn't really describe what happens in real life.

It is now that we will explore what a state machine can do for us that a flowchart cannot!
A flowchart is ok for routine systems which have fixed choices, however they are not useful for what embedded systems such as microcontrollers are used for: **REACTIVE** systems. Flowcharts cannot handle reactive systems very well. In our case what happens if while the truck is accelerating the driver sees another red traffic light ahead. According to our state machine he must continue unitl 50Km/hr and then he can react to another red light. We can easily modify our state machine with another transistion to add this detail.



The same exists if during the state of decelerating for a red light the light changes to green. According to our state machine he must stop first. Another transition will fix this easily.



These two example systems we have looked might be described as a macro view, what people and devices are doing. We are interested in a micro view, what is actually happening inside an electronic black box, for us that means modelling what software is doing within our microcontroller and a state machine id perfect for this.

36.3 Developing a state machine

Developing States (starts with defining outputs)

To identify the different states for your machine, identify the different states of the various output devices e.g. temperature alarm system outputs:

- LCD – displays temperature / displays setting of the temperature alarm value
- Light – on / off
- Alarm – on / off

If you have an LCD, you might plan each different screen of the LCD (which could include instructions)

- Displaying the temperature

```
Temperature now 22c
# to set alarm
* to reset alarm
Test A=light B=sound
```

- Modifying the temperature alarm

```
Alarm on below 18c
A to increase
B to decrease
D=save&exit C=cancel
```

(Note that if you hear the word 'mode' this also means the state of a device)

Developing Actions, what are the actions the device needs to carryout e.g.

- Control output devices
  - turn light on
  - turn light off
  - sound alarm
  - display temperature
  - show main instructions screen
  - show temperature setting screen
- Monitor input devices
  - Read a keypad
  - Read the temperature sensor
- Control functions
  - start the timer
  - stop the timer
  - zero the timer

When do these actions have to take place?

- Repeated all the time within a state
  - Read keypad
  - Read temperature
  - Display temperature
- Only once in the transition between states
  - Turn LED on
  - Turn LED off
  - Save a new setting

Some actions could be put into either category, but some couldn't e.g.

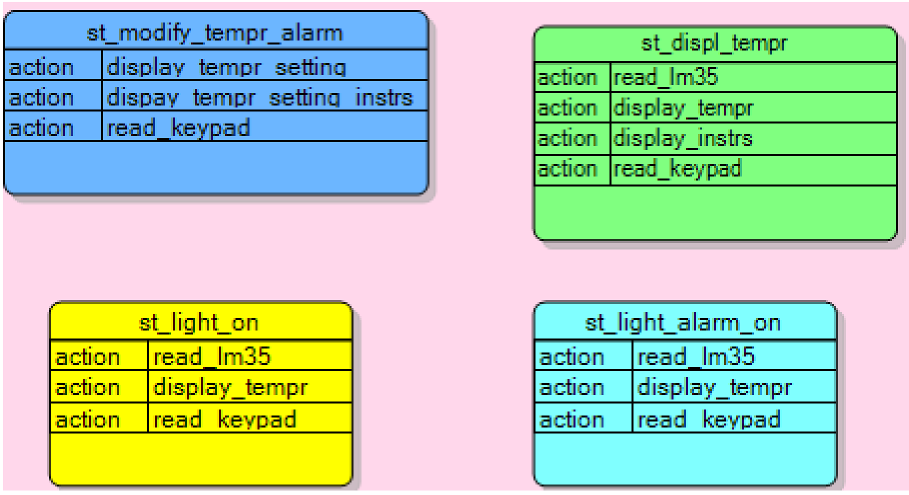
- What is the effect of putting the action clear_the_lcd inside a state compared to inside a transition?
- What is the effect of putting the action led_on inside a state compared to inside a transition?
- What is the effect of putting the action zero_timer inside a state compared to inside a transition?

Developing Transitions

- Testing inputs and variables to see if some condition is true or not
  - Was a particular key or button pressed
  - Has a variable reached a particular value

36.4 A state machine for the temperature alarm system

Here are the 4 states for the temperature controller and a diagram representation of it



<u>State 1: measure and display temperature</u>

<u>State 2: light and alarm are both on</u>

<u>State 3: light only is on</u>

<u>State 4: modify the preset temp alarm setting</u>

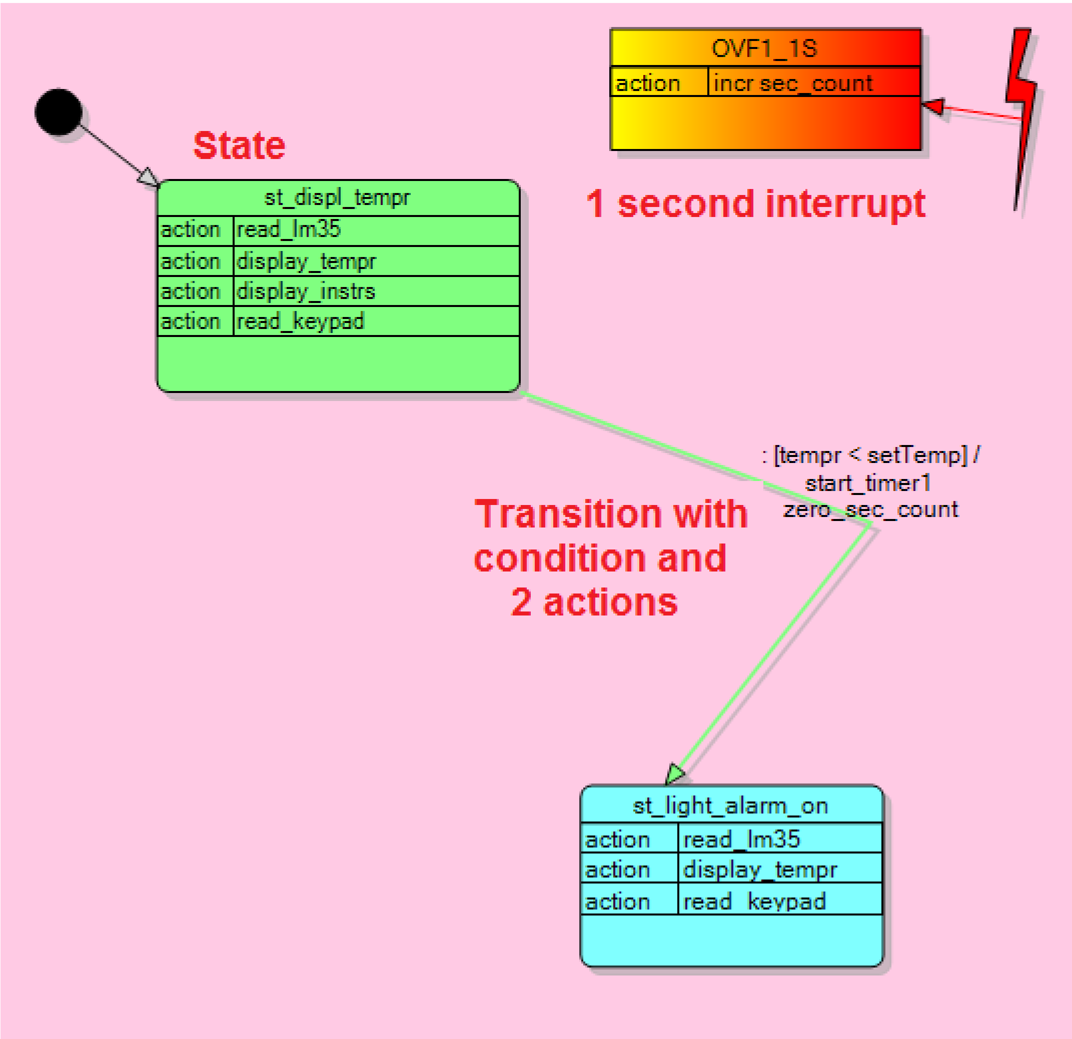
Each state includes the names of ations(subroutines) that will be called to do different things. It is good practice not to put code into the state, so that the control structure is not confused with control of I/O devices. Also if any subroutine is complex it may require a flowchart or even another state machine to plan it.

The second part of the process is to build the transitions between the states and what conditions cause them to occur. The black circle indicates the starting state for when power is applied.

Here one transition is shown for when the temperature reading has fallen below the set level.

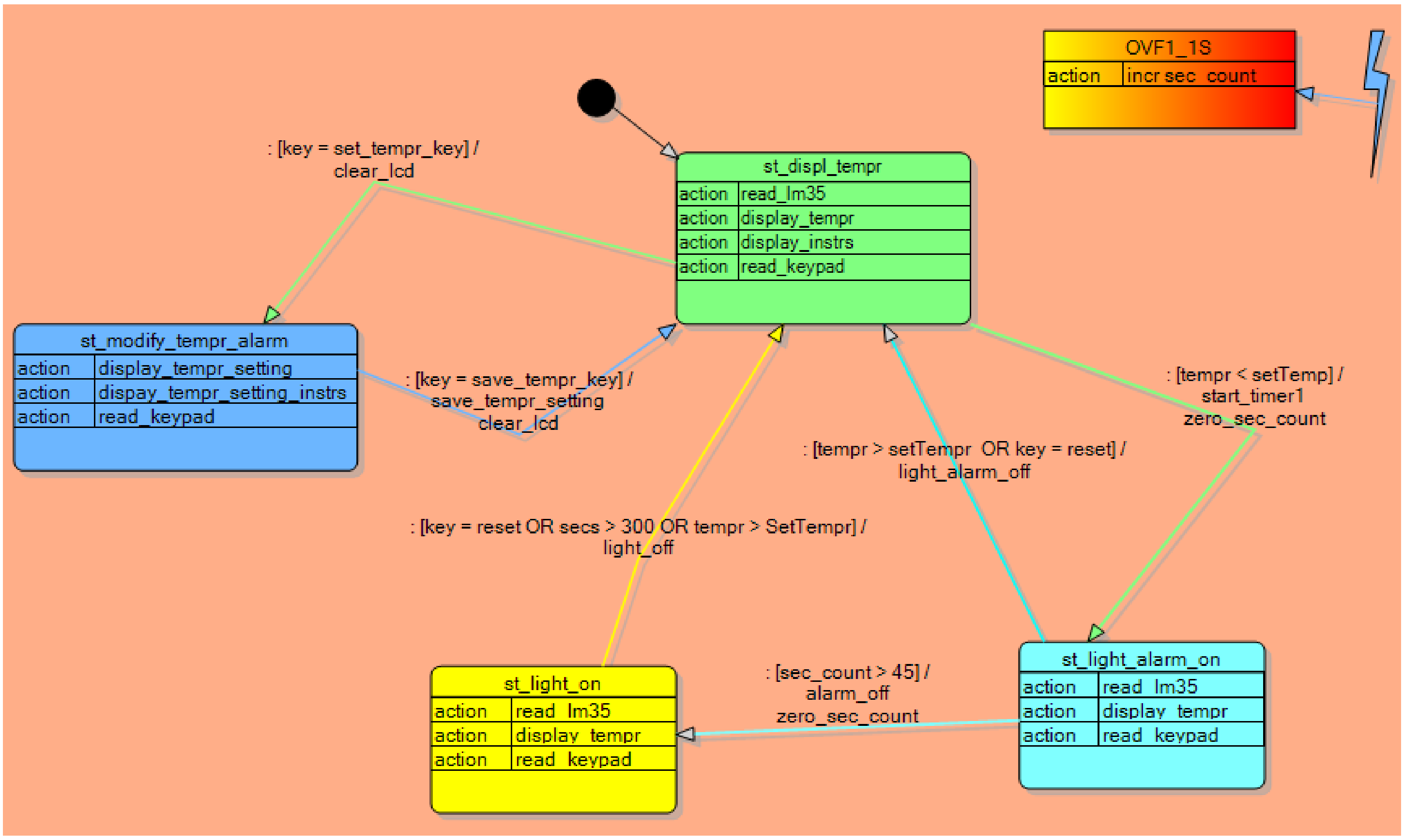
A condition is in square brackets [ ], it looks like any test that would be part of an if…then, while… wend or do loop until…

Along with the condition are the actions you want the program to carry out after one state has stopped execution and before the next state starts executing. An action could be a call to a subroutine or a very short one or two lines of code. Actions are optional, but almost all (though not all) transitions will have conditions



Here are all the states and transitions for our temperature system.

State 1: display temperature
Conditions: temp < setting, keypad to change setting
State 2: light and alarm are on
Conditions: reset pressed, temperature <= setting, 45 second time out
State 3: light on
Conditions: reset pressed, temperature <= setting, 5 minute time out
State 4: modify temp setting
Conditions: finished changing setting



Note that this state machine has a central state and it can be seen that there are a transitions into and out of this state. Not all systems will have a central state like this.

This style of problem solving overcomes the issues identified relating to flowcharts
- They are intuitive – in fact clients can easily understand them
- Errors are seen easily as the relationships between states are logically laid out.
- It is actually very easy to write the code to match this diagram using if-then and while-wend statements
- The code is easily maintained in the future and flows logically when it is written making it easier to remember what you did or for others to read and maintain.
- Students can very easily develop quite sophisticated software solutions using this process.
- If you closely follow the structure using subroutine names then you can use the software I have developed to create your code for you in BASCOM_AVR!!!

States

Each unique state of your device is represented by a block in a state machine diagram
To identify the different states for your machine, identify the different states of the various output devices e.g. temperature alarm system outputs:

- LCD – displays temperature / displays setting of the temperature alarm value
- Light – on / off
- Alarm – on / off

If you have an LCD, you might plan each different screen of the LCD (which could include instructions)

- Displaying the temperature

```
Temperature now 22c
# to set alarm
* to reset alarm
Test A=light B=sound
```

- Modifying the temperature alarm

```
Alarm on below 18c
A to increase
B to decrease
D=save&exit C=cancel
```

(Note that if you hear the word ‘mode’ this also means the state of a device)

Actions, what are the actions the device needs to carryout e.g.

- Control output devices
  - turn light on
  - turn light off
  - sound alarm
  - display temperature
  - show main instructions screen
  - show temperature setting screen
- Monitor input devices
  - Read a keypad
  - Read the temperature sensor
- Control functions
  - start the timer
  - stop the timer
  - zero the timer

When do these actions have to take place?

- Repeated all the time within a state
  - Read keypad
  - Read temperature
  - Display temperature
- Only once in the transition between states
  - Turn LED on
  - Turn LED off
  - Save a new setting

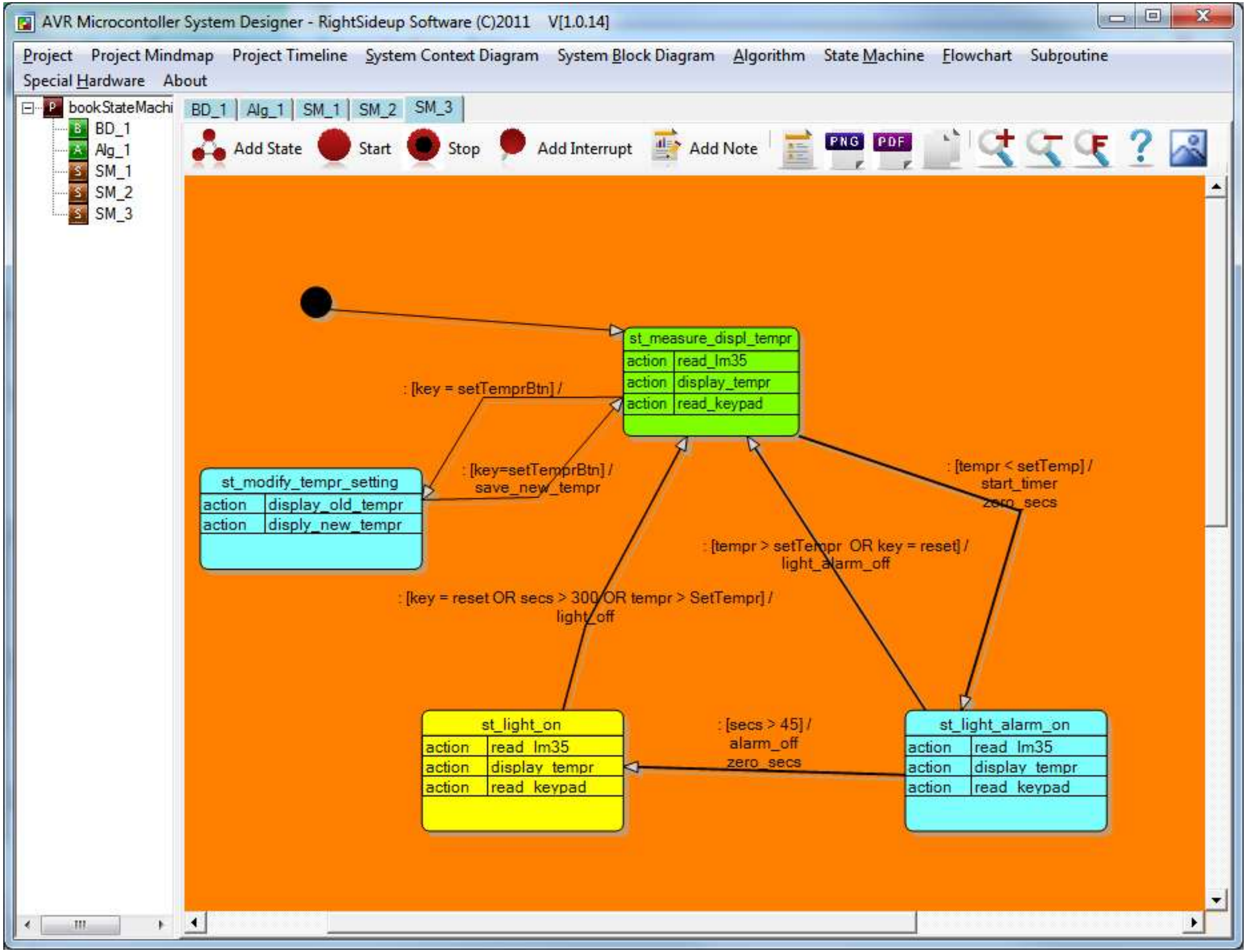
Some actions could be put into either category, but some couldn’t e.g.

- What is the effect of putting the action clear_the_lcd inside a state compared to inside a transition?
- What is the effect of putting the action led_on inside a state compared to inside a transition?
- What is the effect of putting the action zero_timer inside a state compared to inside a transition?

Transitions

- Testing inputs and variables to see if some condition is true or not
  - Was a particular key or button pressed
  - Has a variable reached a particular value

36.5 Using System Designer software to design state machines

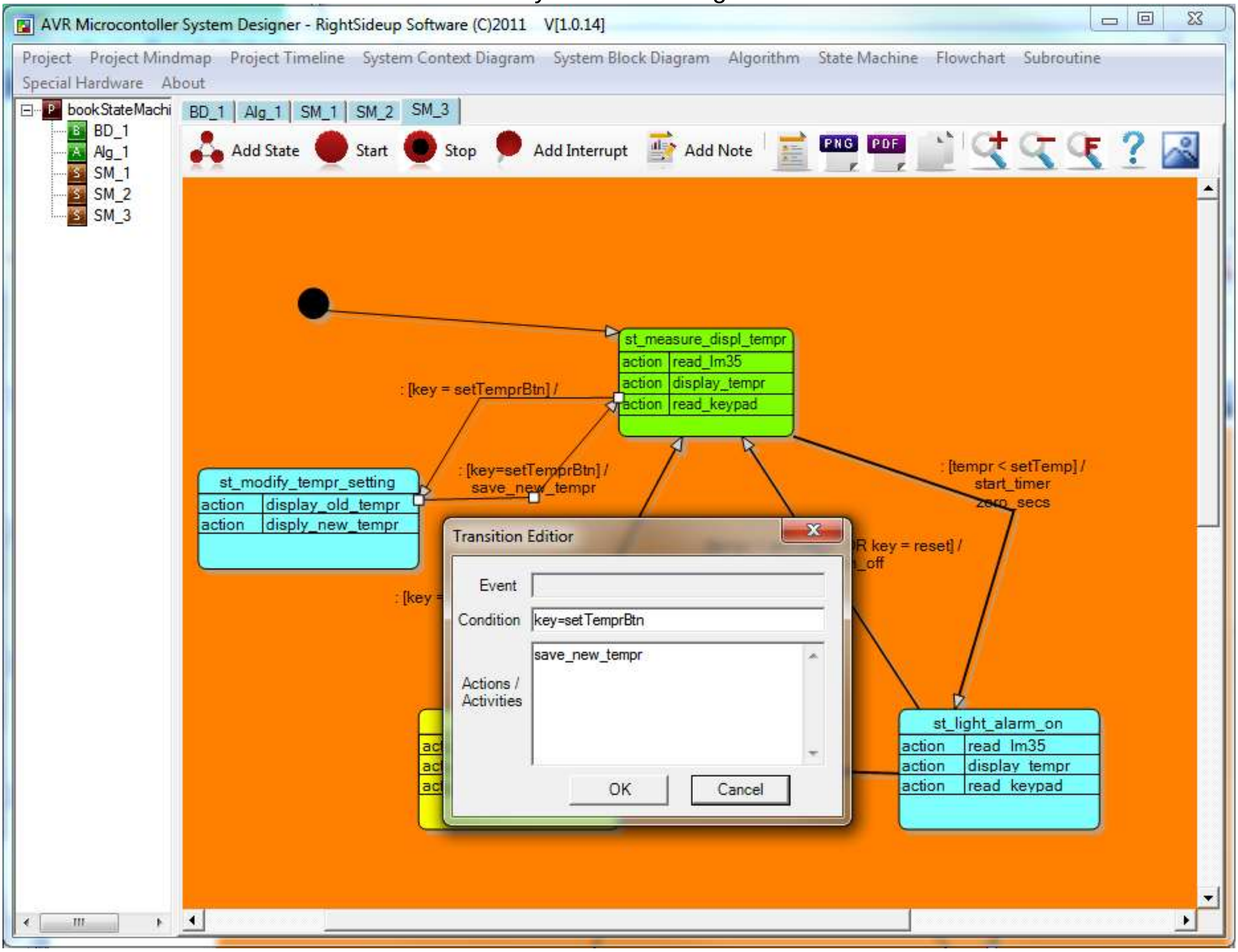


After opening System Designer add a state machine, then some states and then transitions.

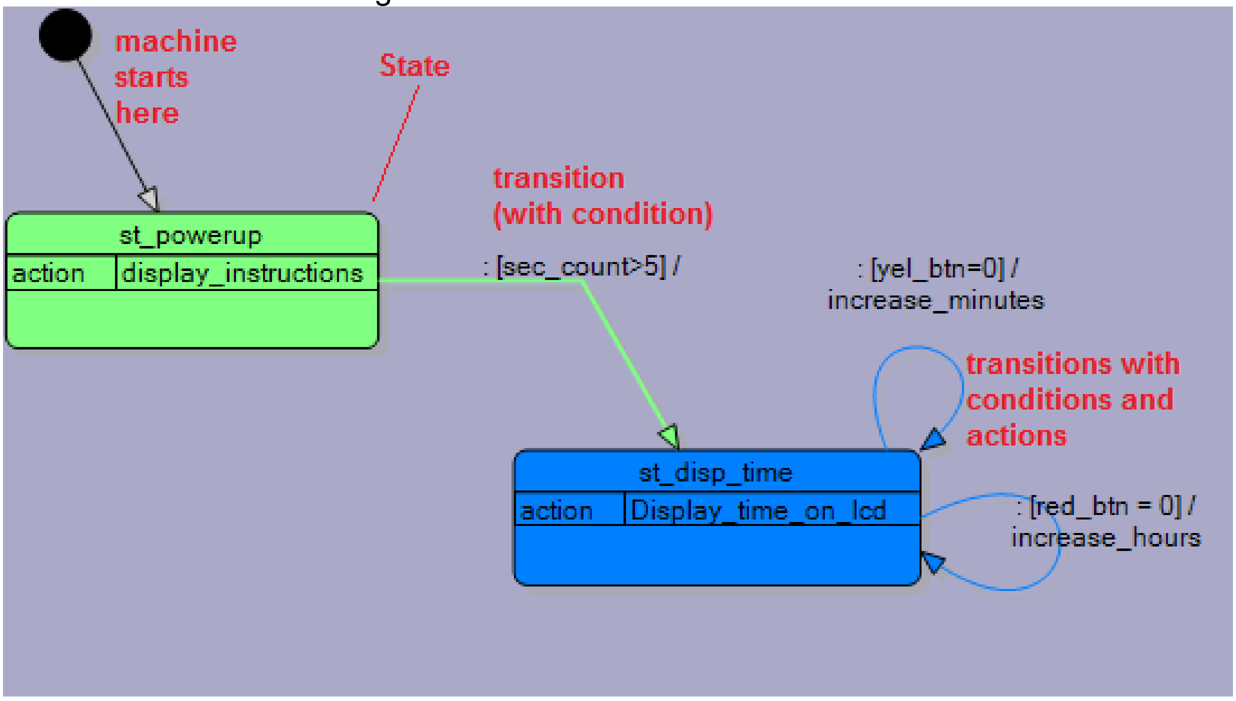
Adding transitions by clicking on a state and drawing with the mouse (make sure the state is not selected first)

Identify the transition arrow that indicates program flow outwards towards the state ModifyTemprSetting. Having drawn the transition line between the two states, double clicking on the line allows the user to add conditions that trigger the transition and any actions that might need to be performed between state changes. In this case the state change is triggered when a keypad is read and the value setTemprbtn is returned. Key will be a variable and setTemprbtn will be a constant in our program.
As seen in this diagram colours and even fonts can be changed (by right clicking on the diagram/state/transition)

Transition conditions and actions are edited by double clicking on a transition



Transitions that don't change state are common in state machines



36.6 State machine to program code

Once the initial logic of the state machine is planned the program code can be written. To write the code in BASCOM a state variable is dimensioned and each state is assigned a value as a constant.

```
dim state as byte

Const st_light_alarm_on = 1
Const st_Light_On = 2
Const st_displ_tempr = 3
Const st_modify_tempr_setting = 4
```

Using constants rather than values within program code makes the code so much easier to read.

The starting state is determined by initialising the state variable

```
state = st_displ_tempr
```

In the main body of the code a do-loop is used to enclose all the states, which are coded using while-wend statements.

```
Do
  while state = st_light_alarm_on
  wend

  while state = st_light_on
  wend

  while state = st_displ_tempr
  wend

  while state = st_modify_tempr_setting
  wend

Loop
```

Note: so far we have predominantly used do-loop-until as a looping control in our programs.
The while –wend is a little easier to follow in this instance but both do exactly the same thing.
So we could replace the the while-wend's above with

```
Do
Loop Until state <> st_Light_On
```

Program flow is controlled by the value of the variable **state**.
When the value of state is **4** (**St_measure_displ_tempr**) the code within that while wend will be executed.

If the value of state changes then a different section of code will be executed.

```
dim state as byte


Const st_light_alarm_on = 1
Const st_light_on = 2
Const st_displ_tempr = 3
Const st_modify_tempr_setting = 4

state = st_measure_displ_tempr

Do

   while state = st_light_alarm_on
   wend

   while state = st_light_on
   wend

   while state = st_displ_tempr
   wend

   while state = st_modify_tempr_setting
   wend

Loop
```

The next stage is to add calls to subroutines within each state, for example:

```
while state = st_Measure_displ_tempr
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
wend
```

Next the code for the transitions is written, these have conditions (if-then-end if) tests that trigger or cause one state to transition to the next:

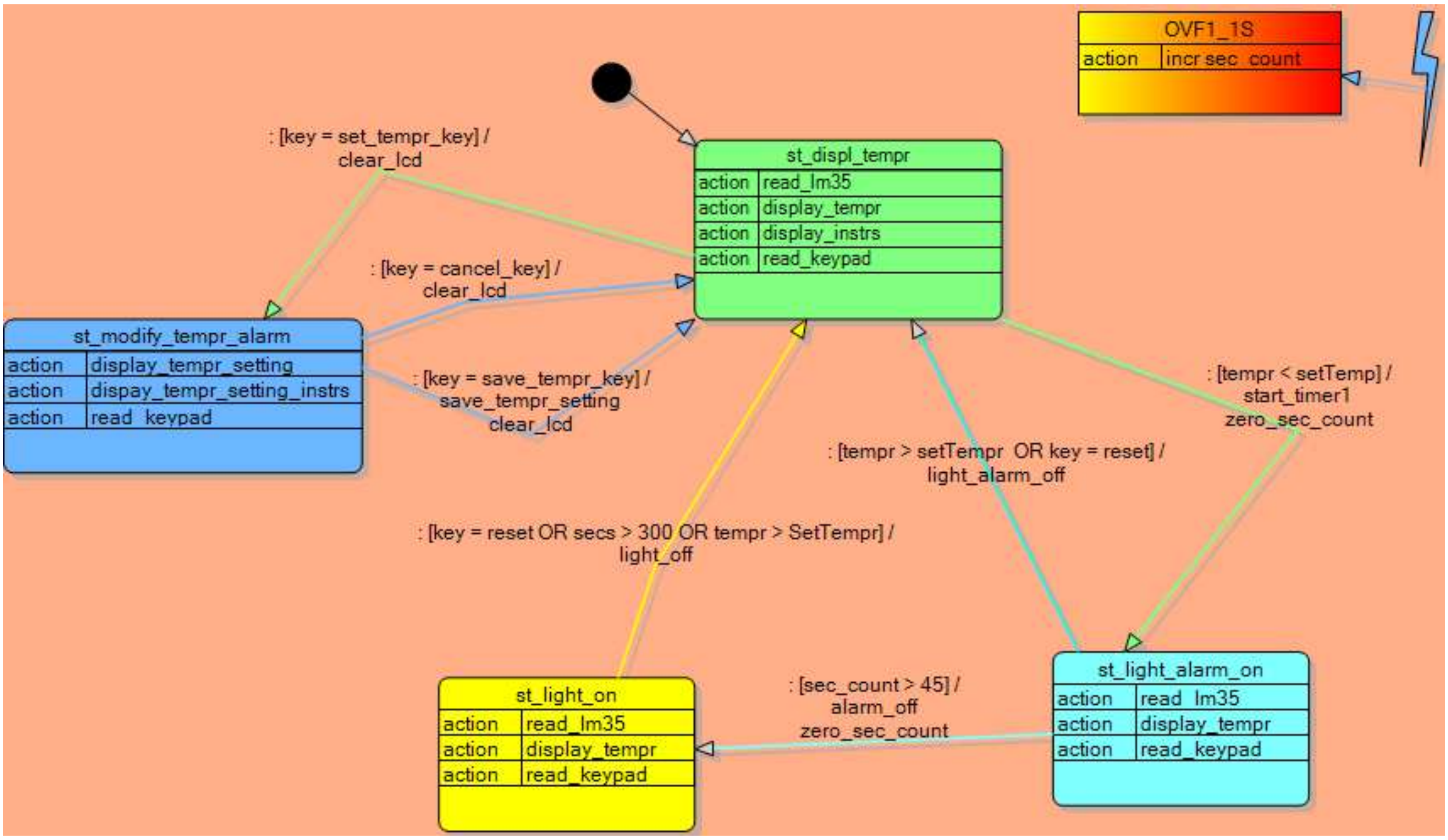
```
while state = st_displ_tempr
    gosub ReadLM35
    gosub DisplayTempr
    gosub ReadButtons
    if btn = setTempr then
      state = st_modify_tempr_setting
    end if
    if tempr < setTempr then
      state = st_Light_Alarm_On
      GOSUB startTimer
    end if
wend
```

When a condition or trigger for a state change has occurred, the state variable takes on a new value, the currently executing while-wend will continue on to completion, then from within the main do-loop the new state is identified and the appropriate while-wend is entered.

In this example there are many shortcuts that proficient and competent programmers could take; however using a very structured process means that novice student programmers begin good practices early on with strong naming conventions and logical practices. It makes my job as teacher less difficult as I can debug code more easily and will therefore grow gray less quickly.

36.7 The power of state machines over flowcharts

Having coded the system and got it working any changes or new features are easily implemented. In the current state machine a user can only exit **ModifyTemprSetting** state by saving the change. What if the client adds the specification that the user should be able to either save or exit without saving. A cancel or nosave button could be implemented very easily? This is shown via the change in this version .



A user could add this code to the state machine program very easily.

```
 while state = st_modify_tempr_setting
   gosub DisplayOldTempr
   gosub DisplayNewTempr
   gosub ReadButtons
   gosub ModifyTempr
   if btn=setTempr then
     state = st_measure_dspl_tempr
     GOSUB SaveNewTempr
   end if
     if btn = cancel then
        state = st_displ_tempr
     end if
Wend
```

The Bascom Program for our temperature alarm system

```
Const st_Light_Alarm_On = 1
Const st_Light_On = 2
Const st_measure_displ_tempr = 3
Const st_Modify_Tempr_Setting = 4

Do
   while state = st_Light_Alarm_On
      gosub ReadLM35
      gosub DisplayTempr
      gosub ReadButtons
      if secs > 45 then
        state = LightOn
        GOSUB AlarmOff
      end if
      if tempr > setTempr then
        state = St_displ_tempr
        GOSUB LightAlarmOff
      end if
      if btn=reset then
        state = St_measure_displ_tempr
        GOSUB LightAlarmOff
      end if
   wend

   while state = st_Light_On
      gosub ReadLM35
      gosub DisplayTempr
      gosub ReadButtons
      if btn=reset then
        state = St_measure_displ_tempr
        GOSUB lightOff
      end if
      if tempr>setTempr then
        state = St_displ_tempr
        GOSUB lightOff
      end if
      if secs>300 then
        state = St_measure_displ_tempr
        GOSUB lightOff
      end if
   wend

   while state = st_measure_display
      gosub ReadLM35
      gosub DisplayTempr
      gosub ReadButtons
      if tempr < setTempr then
        state = LightAlarmOn
        GOSUB startTimer
      end if
      if btn=setTempr then
        state = ModifyTemprSetting
      end if
   wend

   while state = st_modify_tempr_setting
      gosub DisplayOldTempr
      gosub DisplayNewTempr
      if btn=setTempr then
        state = St_measure_displ_tempr
        GOSUB SaveNewTempr
      end if
   wend
Loop
```

Labels are used for states rather than numbers to facilitate program readability

The state variable is used to manage which code segment is executed

Changing to another state only occurs when specific conditions happen.

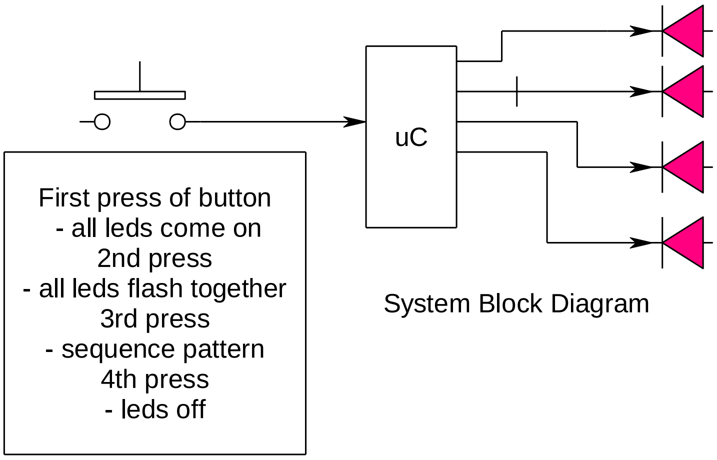
```
'*********************************
subroutines

ReadLM35:
Return

DisplayTempr:
Return

ReadButtons:
Return

DisplayOldTempr
Return

DisplayNewTempr
Return

startTimer:
Return

lightOff:
Return

AlarmOff:
Return

SaveNewTempr:
Return

LightAlarmOff:
Return
```

The rest of the program controls all the I/O and is in subroutines which are then easier to write and check individually

36.8 Bike light – state machine example

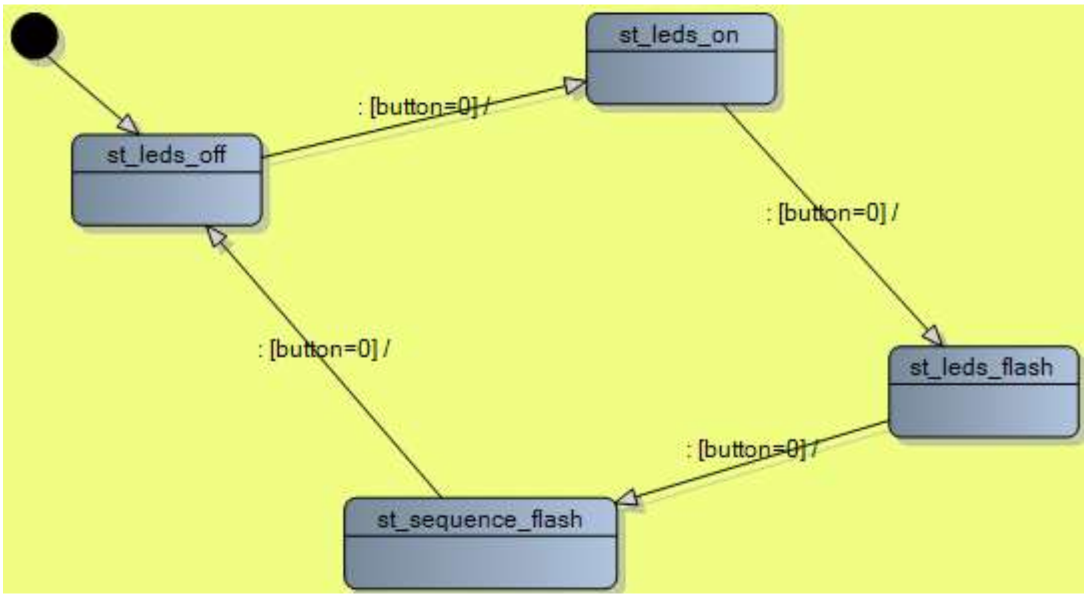
These rear lights for bicycles have different modes of operation. In this example they are called states:
State1: LEDs_OFF
State2: LEDs_ON
State3: ALL_FLASH
State4:SEQUENCE_FLASH (1-2-3-4-1-2-…)

The light '**transitions'** between the 4 states every time the '**condition'** occurs (button is pressed).

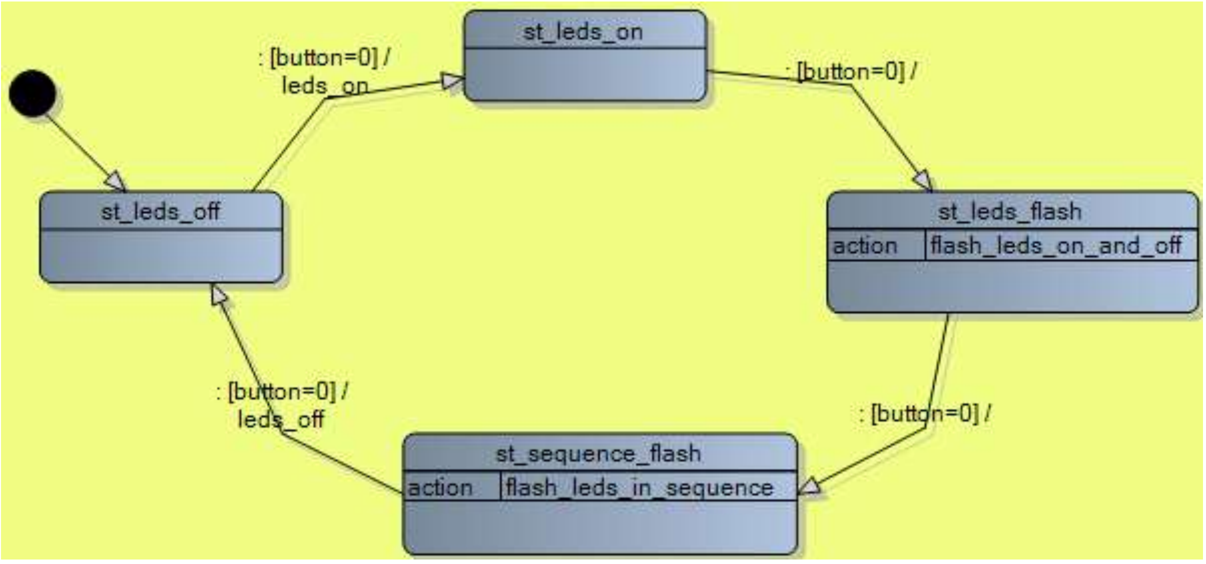


System Block Diagram

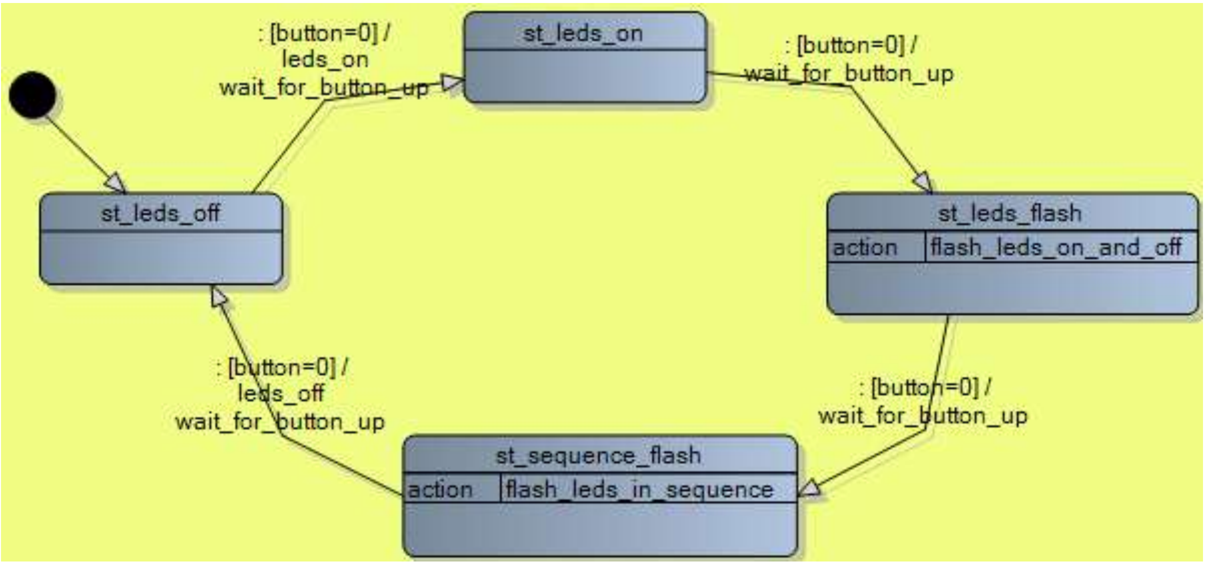
Here is a first state machine to describe the process



This needs some further development and subroutines have been added to each state to handle the various activities.



There is an issue with transitioning between states as microcontrollers are very quick and our button pressing skills by comparison are very slow! So we need to wait during the transition from one state to another so that the micro will not skip states. We setup an' **action'** to wait for the button to be released, and every state transition needs it.



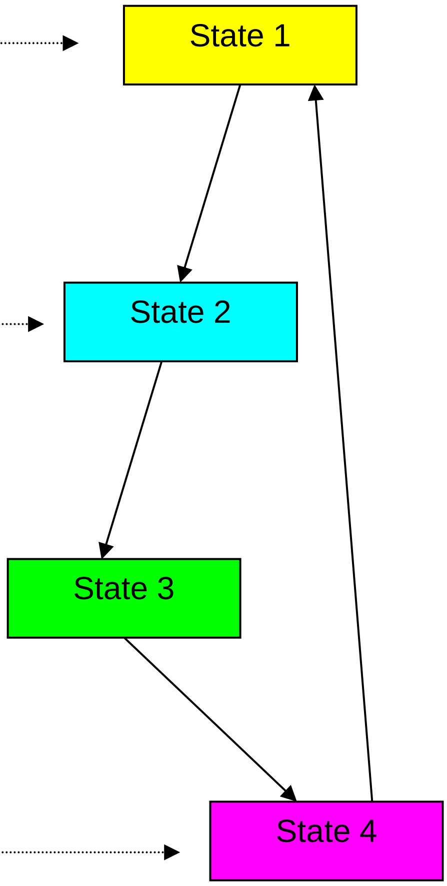
The actual code for the routine might look like

```
Waitforbuttonup:
   Do
    Waitms debouncedelay
   Loop until button=1
   Waitms Debouncedelay
Return
```

36.9 Bike light program version1b

Using system designer the following code was produced

```
Dim State As Byte
'REMEMBER TO DIMENSON ALL YOUR VARIABLES HERE
Const st_LEDs_off = 1
Const st_LEDs_Sequence_Flash = 2
Const st_LEDs_On = 3
Const st_LEDs_Flash = 4
'REMEMBER TO DEFINE ALL YOUR CONSTANTS HERE
state = st_LEDs_off
Do
   while state = st_LEDs_off
      gosub LEDs_Off
      if button=0 then
        state = LEDs_On
        GOSUB waitforbuttonup
      end if
   wend

   while state = st_LEDs_On
      gosub LEDs_On
      if button=0 then
        state = LEDs_Flash
        GOSUB waitforbuttonup
      end if
   wend

   while state = st_LEDs_Flash
      gosub LEDs_Flash
      if button=0 then
        state = LEDs_Sequence_Flash
        GOSUB waitforbuttonup
      end if
   wend

   while state = st_LEDs_Sequence_Flash
      gosub LEDs_sequence_Flash
      if button=0 then
        state = LEDs_off
        GOSUB waitforbuttonup
      end if
   wend

Loop
```



```
'**********************************
'subroutines
LEDs_Off:
Return

LEDs_On:
Return

LEDs_Flash:
Return

LEDs_sequence_Flash:
Return

waitforbuttonup:
Return
```

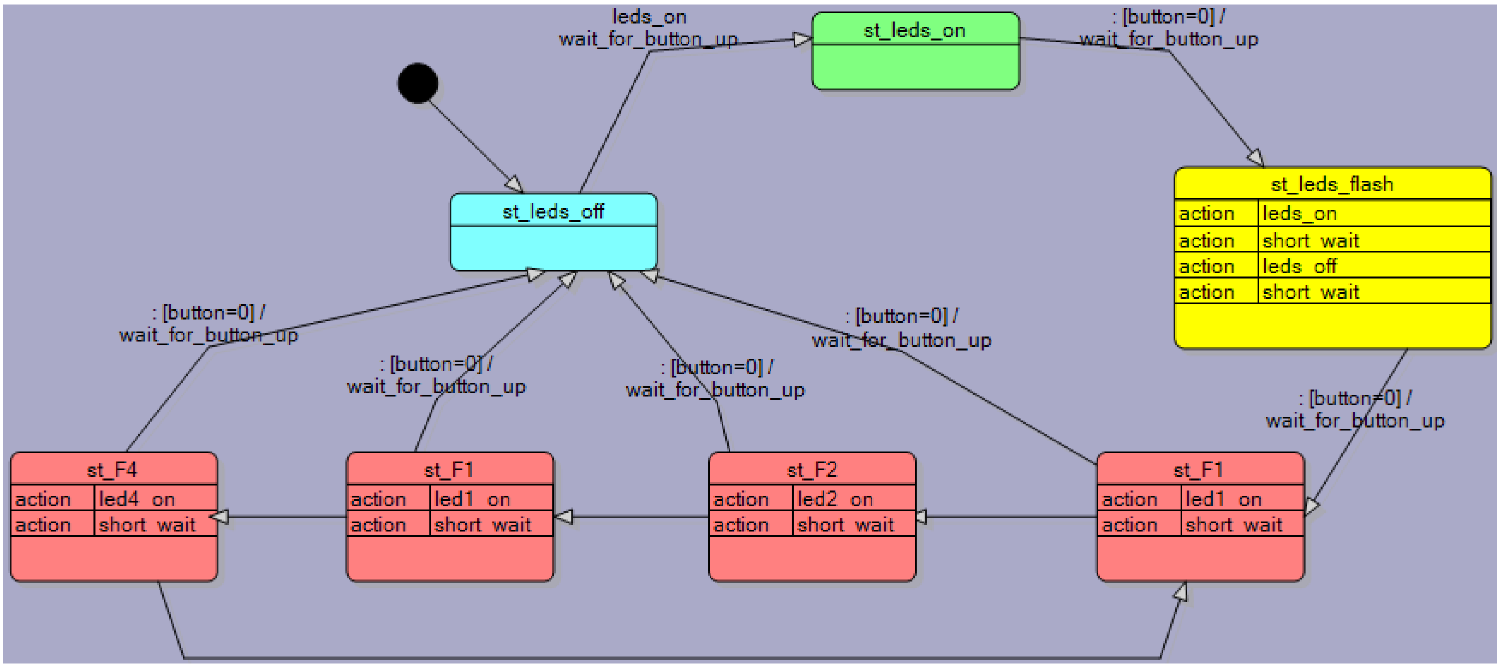
All these subroutines need code to be written for them

BUT WAIT A SECOND!!

Seeing the code led me to the realisation that during the subroutine sub_LEDs_sequence_Flash the micro needs to check for a button press from the user or it is possible that it might miss it while it is doing the full sequence of flashing each LED individually.

There are no delays in sub_LEDs_Off and sub_LEDs_On as they have no need for them. However sub_LEDs_sequence_Flash and sub_LEDs_Flash need some form of delay. During sub_LEDs_Flash if the delays are short enough then we can get away without checking the switch. However during sub_LEDs_sequence_Flash we will need to check the switch .

Bike light state machine V2 solves this by introducing some new states for the sequence flashing.



See how easy the state machine is to modify; and the code is not hard to modify either.

36.10 Bike light program version2

```
'State Variables
Dim state as byte
Const st_leds_on = 0
Const st_leds_off = 1
Const st_leds_flash = 2
Const st_F1 = 3
Const st_F2 = 4
Const st_F1 = 5
Const st_F4 = 6
State = st_leds_off                     'set the initial state

Do

   '*************** state st_leds_on ***************
   While state = st_leds_on
      If button=0 Then
         state = st_leds_flash
         Gosub wait_for_button_up
      End If
   Wend

   '*************** state st_leds_off ***************
   While state = st_leds_off
      If button=0 Then
         state = st_leds_on
         Gosub leds_on
         Gosub wait_for_button_up
      End If
   Wend

   '*************** state st_leds_flash ***************
   While state = st_leds_flash
      Gosub leds_on
      Gosub short_wait
      Gosub leds_off
      Gosub short_wait
      If button=0 Then
         state = st_F1
         Gosub wait_for_button_up
      End If
   Wend

   '*************** state st_F1 ***************
   While state = st_F1
      Gosub led1_on
      Gosub short_wait
      state = st_F2
      If button=0 Then
         state = st_leds_off
         Gosub wait_for_button_up
      End If
   Wend

   '*************** state st_F2 ***************
```

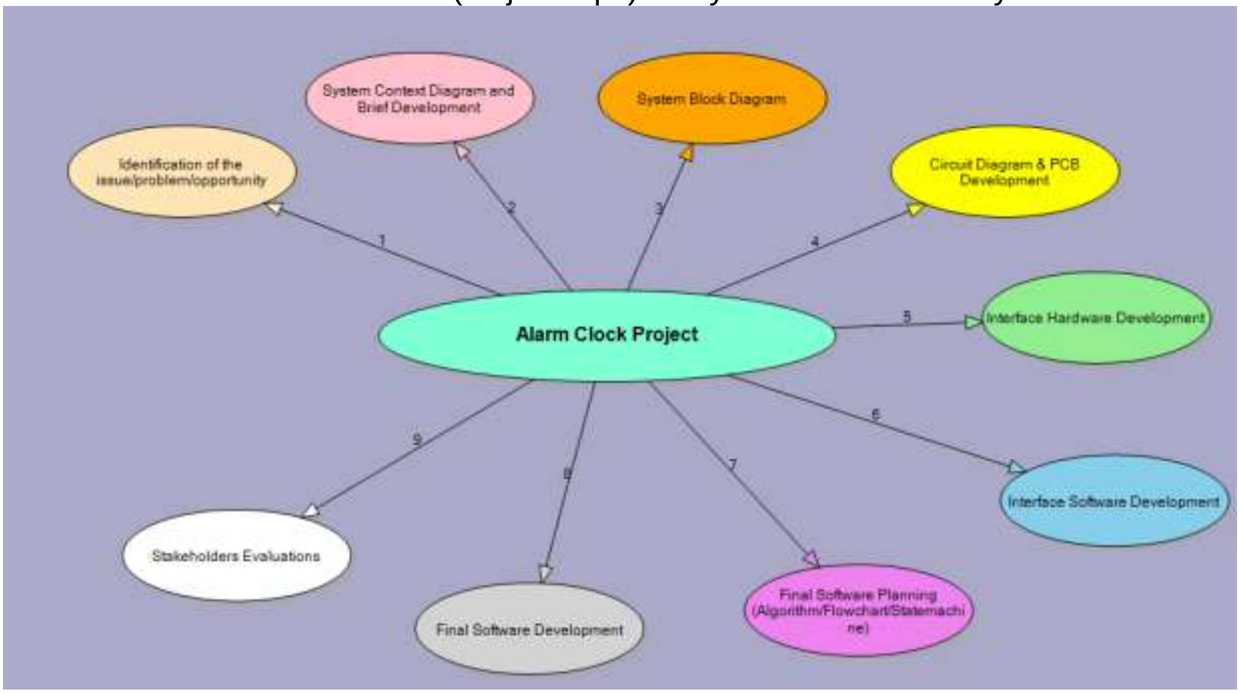
```
   While state = st_F2
      Gosub led2_on
      Gosub short_wait
      state = st_F1
      If button=0 Then
         state = st_leds_off
         Gosub wait_for_button_up
      End If
   Wend
   '*************** state st_F1 ***************
   While state = st_F1
      Gosub led1_on
      Gosub short_wait
      state = st_F4
      If button=0 Then
         state = st_leds_off
         Gosub wait_for_button_up
      End If
   Wend
   '*************** state st_F4 ***************
   While state = st_F4
      Gosub led4_on
      Gosub short_wait
      If button=0 Then
         state = st_leds_off
         Gosub wait_for_button_up
      End If
      state = st_F1
   Wend

Loop
End
'***********************************************
'Subroutines
wait_for_button_up:
Return
leds_on:
Return
short_wait:
Return
leds_off:
Return
led1_on:
Return

led2_on:
Return

led4_on:
Return
```

37 Alarm clock project re-developed

Let's try building a digital alarm clock.

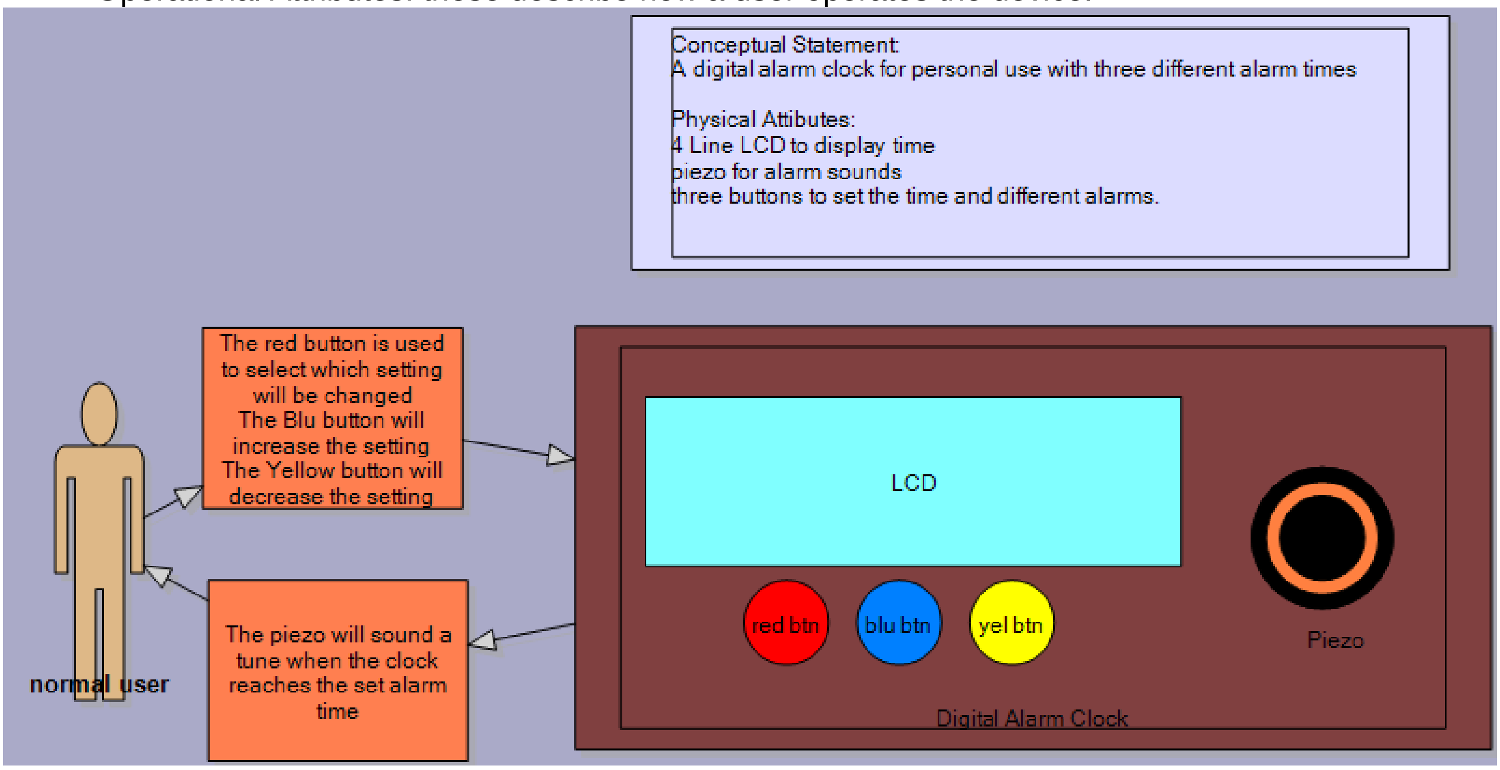
37.1 System Designer to develop a Product Brainstorm

Start with a brainstorm of the milestones (major steps) that you will need to carry out



There are some important attributes (characteristics) of the system to describe that will make designing the hardware and software easier later on.

- Build a simple picture of the device with all its inputs and outputs
- A conceptual statement gives a one line overview of what is to be designed
- Physical Attributes: these describe a bit more detail about what the device looks like
- Operational Attributes: these describe how a user operates the device.



A button on the toolbar in system designer will generate a written brief built from the information in the diagram.

System Description (Brief)

Conceptual Statement:
A digital alarm clock for personal use with three different alarm times

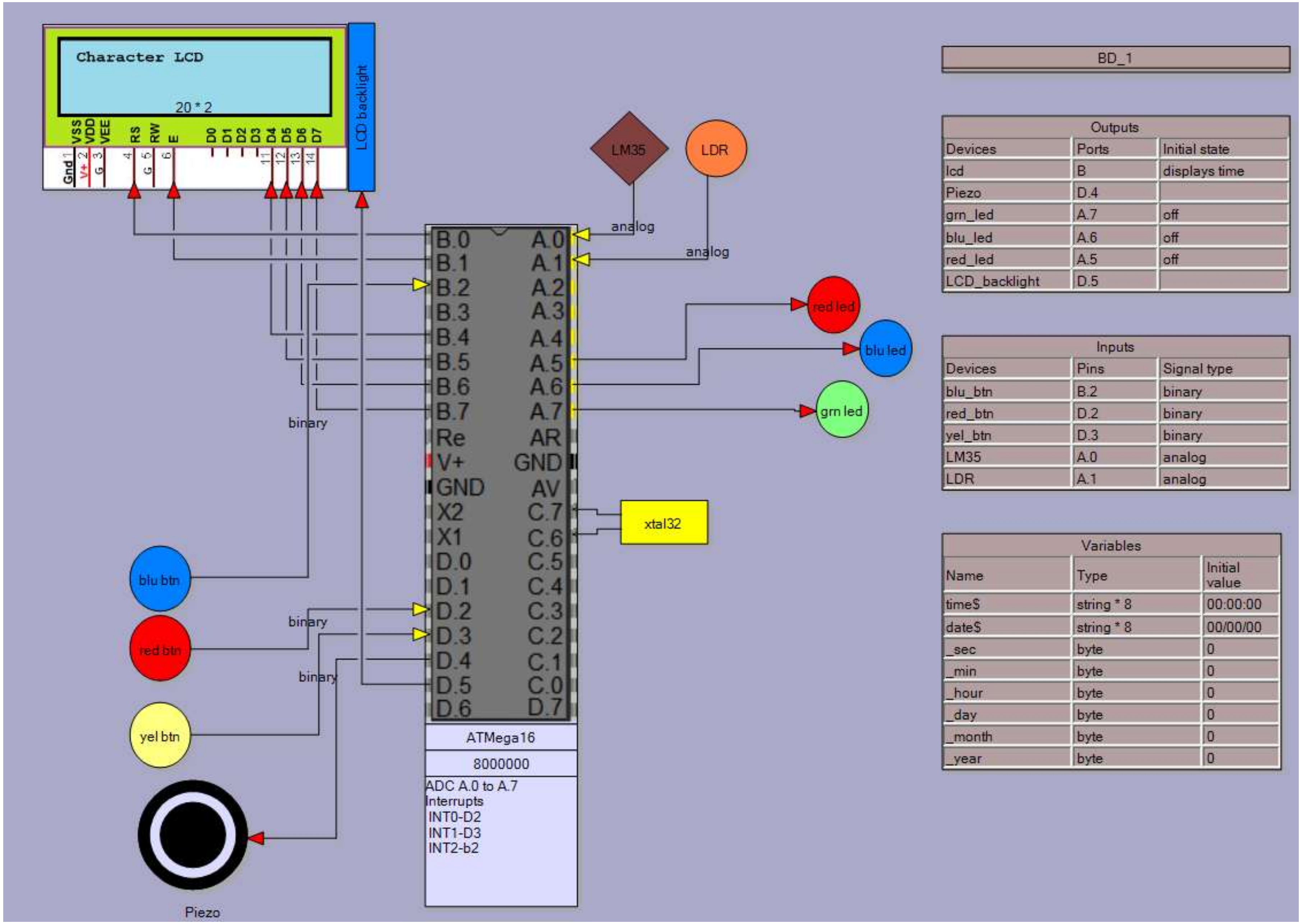
Physical Attibutes:
4 Line LCD to display time
piezo for alarm sounds
three buttons to set the time and different alarms.

Physical Attributes for Digital Alarm Clock
  It contains:
    -red btn
    -LCD
    -yel btn
    -blu btn
    -Piezo

Digital Alarm Clock interactions with Normal user are:
   -The piezo will sound a tune when the clock reaches the set alarm time
Normal user interactions with Digital Alarm Clock are:
   -The red button is used to select which setting will be changed
The Blu button will increase the setting
The Yellow button will decrease the setting

37.2 Initial block diagram for the alarm clock

Using System Designer the block diagram is created to express the electrical connections to the microcontroller but without full detail of the schematic diagram which includes things like current limit resistors and pullup resistors.



Note the following devices:
LM35 - a temperatiure sensor – produces an analog rather than binary signal and requires an ADC input.(ADC inputs to the microcontroller have yellow pins)

LDR – produces an analog rather than binary signal and requires an ADC input.

The xtal32 is a 32.768Khz crystal for making a clock, when it is added the variables associated with it are automatically created in Bascom and are also shown in the table.

The BasicCode button in System Designer will generate the following code setup for your program, which is taken directly from the various parts of the block diagram.

```
' Project Name: AlarmClock
' created by: B.Collis -  first created on Mon Aug 15 2011
' block diagram name: BD_1
' Date:8/22/2011 8:49:15 PM
' Code autogenerated by System Designer from www.techideas.co.nz
'*****************************************************
'Compiler Setup
$crystal = 8000000
$regfile = "m16def.dat"

'*****************************************************
'Hardware Configs
Config PORTA = Output
Config PORTB = Output
Config PORTC = Output
Config PORTD = Output
Config PINB.2 = Input       'blu_btn
Config PIND.2 = Input       'red_btn
Config PIND.3 = Input       'yel_btn
Config PINA.0 = Input       'LM35
Config PINA.1 = Input       'LDR

'ADC config
Config Adc = Single , Prescaler = Auto        ', Reference = AVCC/internal/...
Start Adc
'bascom internal features and functions to make a clock in software
'uses 32,768 Hz crystal on PortC.6 and PortC.7
Config Date = Dmy , Separator = /
Config Clock = Soft , Gosub = sectic    'with 1 second interrupt configured

'Character LCD config
Config Lcdpin=pin , Db4 = PORTB.4 , Db5 = PORTB.5 , Db6 = PORTB.6 , Db7 = PORTB.7 , E = PORTB.1 , Rs = PORTB.0
Config LCD = 20 * 2
'*****************************************************

'Hardware aliases
'inputs
blu_btn Alias PINB.2
red_btn Alias PIND.2
yel_btn Alias PIND.3
LM35 Alias PINA.0
LDR Alias PINA.1
'outputs
lcd Alias PORTB
Piezo Alias PORTD.4
grn_led Alias PORTA.7
blu_led Alias PORTA.6
red_led Alias PORTA.5
'*****************************************************
```

37.3 A first (simple) algorithm is developed

An algorithm describes the operation of the system in terms of its
interactions with the world and
its internal functions.

Describe what happens to output devices or variables when an input subsytem or variable changes

A lot of detail is not required here, this is a 'big picture' understanding of how your device functions/is operated by the user

Algorithm 1

Initially the backlight comes on and the lcd displays the time

If the red_btn is pressed the hour increases
If the yel_btn is pressed the minute increases

BD_1

Outputs		
Devices	Ports	Initial state
lcd	B	displays time
Piezo	D.4	
grn_led	A.7	off
blu_led	A.6	off
red_led	A.5	off
LCD_backlight	D.5	

Inputs		
Devices	Pins	Signal type
blu_btn	B.2	binary
red_btn	D.2	binary
yel_btn	D.3	binary
LM35	A.0	analog
LDR	A.1	analog

Variables		
Name	Type	Initial Value
time$	string * 8	00:00:00
date$	string * 8	00/00/00
_sec	byte	0
_min	byte	0
_hour	byte	0
_day	byte	0
_month	byte	0
_year	byte	0

It is important to understand some of the things the device will have to be doing 'inside'.

Note that this is an initial algorithm without a great deal of features, it is a good idea to build your ideas up as you go as they will be easier to develop.

The inputs and outputs you have created in the block digram will appear here making it easier to think about the functions you need to describe.

If you are aware of any Variables you will need to keep data then add them as well at this time.

37.4 A statemachine for the first clock

When starting out using state machines it is important that you take on a little piece of advice!

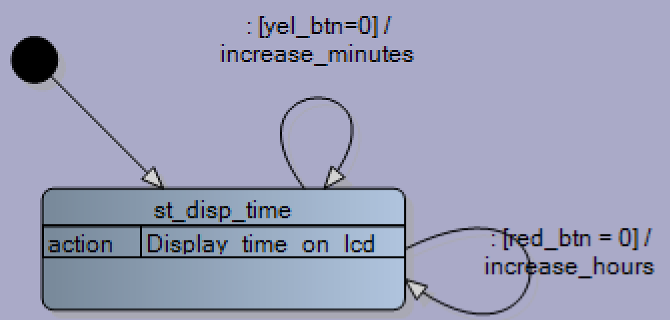
It doesn't take long to gain a lot of confidence and understanding in using statecharts and it wont be long before you are producing large ones.

THEN you want to turn them into program code and you end up in a heap on the floor cursing your teacher because your compiler just told you that your code has 1,967 errors in it!

I have seen it before where students look at this, throw their hands up in horror and go back to trying to rescue their old program because it only had one error in it (even though I told them it would never work)

SO START WITH LITTLE STEPS – your very first real program should have only 1or 2 states in it!!

YOU HAVE BEEN WARNED!



BD_1

Outputs		
Devices	Ports	Initial state
lcd	B	displays time
Piezo	D.4	
grn_led	A.7	off
blu_led	A.6	off
red_led	A.5	off
LCD_backlight	D.5	

Inputs		
Devices	Pins	Signal type
blu_btn	B.2	binary
red_btn	D.2	binary
yel_btn	D.3	binary
LM35	A.0	analog
LDR	A.1	analog

Variables		
Name	Type	Initial Value
time$	string * 8	00:00:00
date$	string * 8	00/00/00
_sec	byte	0
_min	byte	0
_hour	byte	0
_day	byte	0
_month	byte	0
_year	byte	0

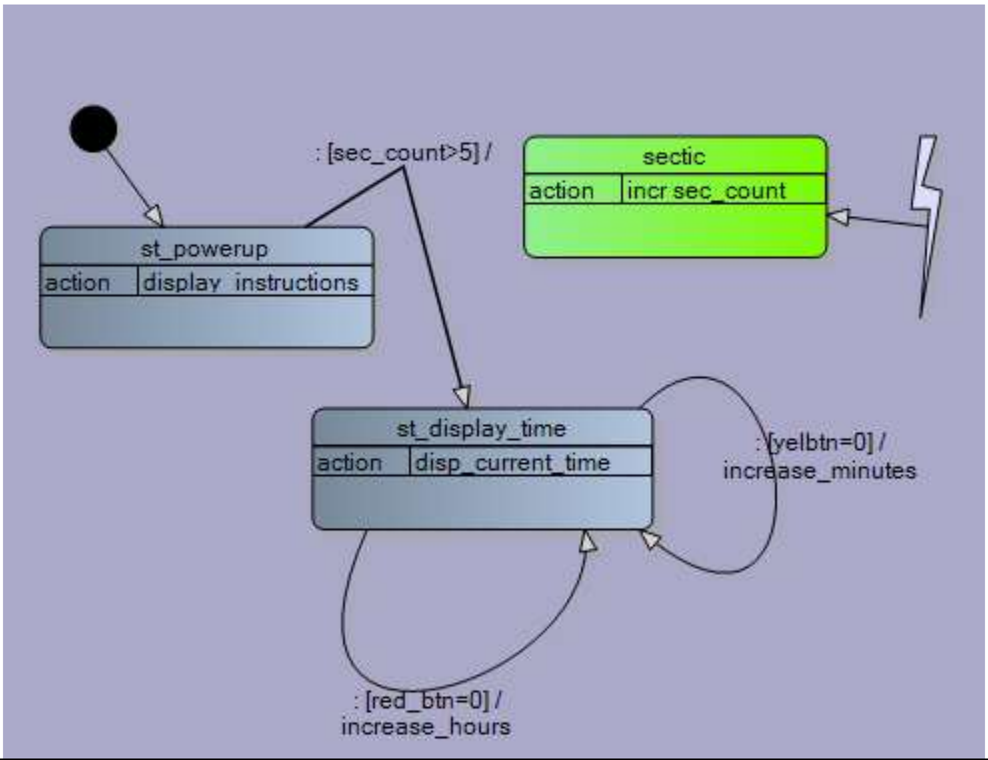
Here the statemachine consists of only one state.

The code for this state is very straightforward

Note:
There is an overall do-loop
A state consists of a While –Wend loop.
There is a variable named state to store the current state in.
To change state the process is simple, change the value of the state variable!
Code has been added to one of the subroutines to make it work as needed

```
'**********************************************
'State Variables
Dim state as byte
Const st_disp_time = 0
State = st_disp_time              'set the initial state

Do

   '*************** state st_disp_time ***************
   While state = st_disp_time
      Gosub Display_time_on_lcd
      If red_btn = 0 Then  Gosub increase_hours
      If yel_btn=0 Then  Gosub increase_minutes
   Wend

Loop
End

'**********************************************
'Subroutines


Display_time_on_lcd:
Return

increase_hours:
   incr _hour                         //increase by 1
   if _hour > 23 then _hour = 0       //fix rollover of hours
   waitms 150                         //delay between increments
Return

increase_minutes:
Return

'**********************************************
'Interrupt Routines
```

37.5 Alarm clock state machine and code version 2



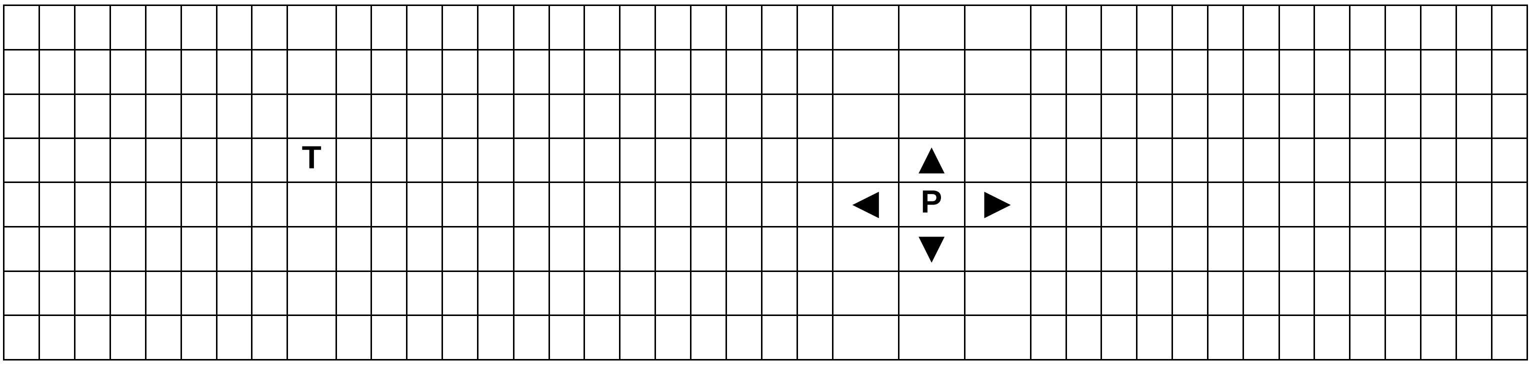

```
'***********************************************
'State Variables
Dim state as byte
Const st_powerup = 0
Const st_display_time = 1
State = st_powerup                'set the initial state

Do

   '*************** state st_powerup ***************
   While state = st_powerup
      Gosub display_instructions
      If sec_count>5 Then st_display_time
   Wend

   '*************** state st_display_time ***************
   While state = st_display_time
      Gosub disp_current_time
      If yelbtn=0 Then  Gosub increase_minutes
      If red_btn=0 Then  Gosub increase_hours
   Wend

Loop
End

'***********************************************
'Subroutines
display_instructions:
Return

disp_current_time:
Return

increase_minutes:
Return

increase_hours:
Return

'***********************************************
'Interrupt Routines
sectic:
   incr sec_count
Return
```

These are the first 2 stages of development of a state machine for an alarm clock, only 2 states and some transitions have been added.

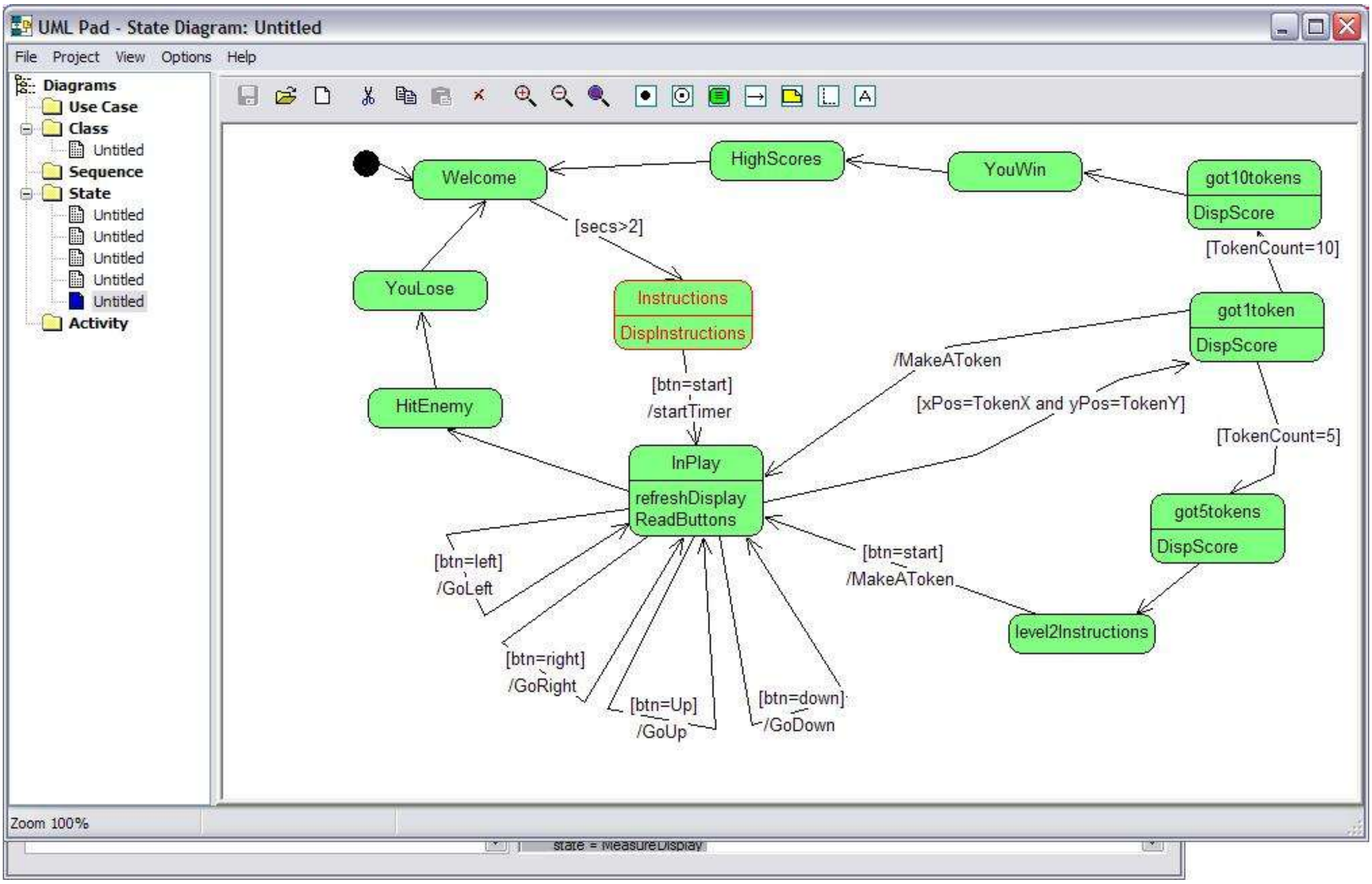
Students must keep progressive versions of plans such as state machines to show their ongoing development work.

37.6 Token game – state machine design example

BRIEF: The game starts with a welcome screen then after 2 seconds the instruction screen appears. The game waits until a button is pressed then a token **T** is randomly placed onto the LCD. 4 buttons are required to move the player **P** around the LCD: 8(up), 4(left), 6(right) and 2(down) to capture the token. Note that the player movements wrap around the screen.
When the player has captured a token, another is randomly generated. After capturing 5 tokens the time taken is displayed, after capturing 10 tokens display the time taken.



Here is the **state machine** for this game (note in this version after collecting 10 tokens nothing happens)**.**



(UMLPAD)

In the program there is a **state** variable that manages the current state and controls what the program is doing at any particular time. This state variable is altered by the program as various events occur (e.g. a token has been captured) or by user input (pressing a button to restart the game).

```
dim state as byte
'REMEMBER TO DIMENSON ALL YOUR VARIABLES HERE
Const got5tokens = 1
Const HitEnemy = 2
Const YouLose = 3
Const InPlay = 4
Const HighScores = 5
Const level2Instructions = 6
Const got10tokens = 7
Const got1token = 8
Const YouWin = 9
Const Welcome = 10
Const Instructions = 11
'REMEMBER TO DEFINE ALL YOUR CONSTANTS HERE
state = Welcome

Do
    while state = got5tokens
        gosub DispScore
        state = level2Instructions
    wend

    while state = HitEnemy
        state = YouLose
    wend

    while state = YouLose
        state = Welcome
    wend

    while state = InPlay
        gosub refreshDisplay
        gosub ReadButtons
        if xPos=TokenX and yPos=TokenY then
            state = got1token
        end if
         if btn=right then
            state = InPlay
            GOSUB GoRight
        end if
        if btn=left then
            state = InPlay
            GOSUB GoLeft
         end if
         if btn=down then
            state = InPlay
            GOSUB GoDown
         end if
        state = HitEnemy
        if btn=Up then
            state = InPlay
            GOSUB GoUp
        end if
    wend
    while state = HighScores
```

In the main do-loop Remember the subroutines to run are within the While-Wend statements

To change what a program is doing you don't Gosub to a new subroutine. You change the state variable to a new value, the current subroutine is then completed.

The While_Wend statements detect the state change and control which new subroutines are called.

The variable state is a 'flag', 'signal' or 'semaphore' in computer science. It is a very common technique. We set the flag in one part of the program to tell another part of the program what to do.

Notice how the reading of buttons and processing of actions relating to the buttons are different things

```
        state = Welcome
    wend

    while state = level2Instructions
      if btn=start then
        state = InPlay
        GOSUB MakeAToken
      end if
    wend

    while state = got10tokens
      gosub DispScore
        state = YouWin
    wend

    while state = got1token
      gosub DispScore
      if TokenCount=10 then
        state = got10tokens
      end if
        state = InPlay
        GOSUB MakeAToken
      if TokenCount=5 then
        state = got5tokens
      end if
    wend

    while state = YouWin
        state = HighScores
    wend

    while state = Welcome
      if secs>2 then
        state = Instructions
      end if
    wend

    while state = Instructions
      gosub DispInstructions
      if btn=start then
        state = InPlay
        GOSUB startTimer
      end if
    wend

Loop

'**********************************
subroutines
```

```
Disp_welcome:
  Locate 1 , 1
  LCD "     Welcome to the TOKEN GAME"
  Wait 2
  State = Instructions
  Cls
Return

Disp_instrustions:
      Cls
      State = Instructions
Return

Disp_instructions:
  Locate 1 , 1
  LCD "capture the tokens  "
  Locate 2 , 1
  LCD "4=left, 6=right"
  Locate 3 , 1
  LCD "2=up, 8=down     "
  Locate 4 , 1
  LCD "D to start"
Return

Got1:
  Cls
  Incr Tokencount
  Select Case Tokencount
  Case 1 To 4:
    Locate 1 , 10
    LCD "you got " ; Tokencount       'display
number of tokens
    Waitms 500                        'wait
    Cls
    State = Inplay                    'restart play
    Gosub Makeatoken
  Case 5:
    State = Got5tokens
  End Select
Return

Got5:
  Cls
  Locate 1 , 2
  LCD " YOU GOT 5 TOKENS"
  Locate 2 , 1
  Seconds = Hundredths / 100          'seconds
  LCD "  in " ; Seconds ; "."
  Seconds = Seconds * 100
  Hundredths = Hundredths - Seconds
  LCD Hundredths ; "seconds"
  State = Gameover
Return

Got10:  'nothing here yet!!
Return

Makeatoken:
  'puts a token on the lcd in a random position
  Tokenx = Rnd(rhs)                   'get a random
number from 0 to Xmax-1
  Tokeny = Rnd(bot_row)               'get a random
number from 0 to Ymax-1
  Incr Tokenx                         'to fit 1 to Xmax
display columns
  If Tokenx > Rhs Then Tokenx = Rhs    'dbl
check for errors
  Incr Tokeny                         'to fit 1 to Ymax
disp rows
  If Tokeny > Bot_row Then Tokeny = Bot_row
'dbl check for errors
  Locate Tokeny , Tokenx              'y.x
  LCD "T"                             'Chr(1)
Return
```

```
Go_left:
    Select Case Xpos
    Case Lhs :                     'at left hand side of lcd
      Oldx = Xpos                  'remember old x position
      Xpos = Rhs                   'wrap around display
      Oldy = Ypos                  'remember old y position
    Case Is > Lhs                  'not at left hand side of lcd
      Oldx = Xpos                  'remember old x position
      Xpos = Xpos - 1              'move left
      Oldy = Ypos                  'remember old y position
    End Select
Return

Go_right:
    Select Case Xpos
    Case Is < Rhs:
      Oldx = Xpos
      Xpos = Xpos + 1
      Oldy = Ypos
    Case Rhs:
      Oldx = Xpos
      Xpos = Lhs
      Oldy = Ypos
    End Select
Return
```

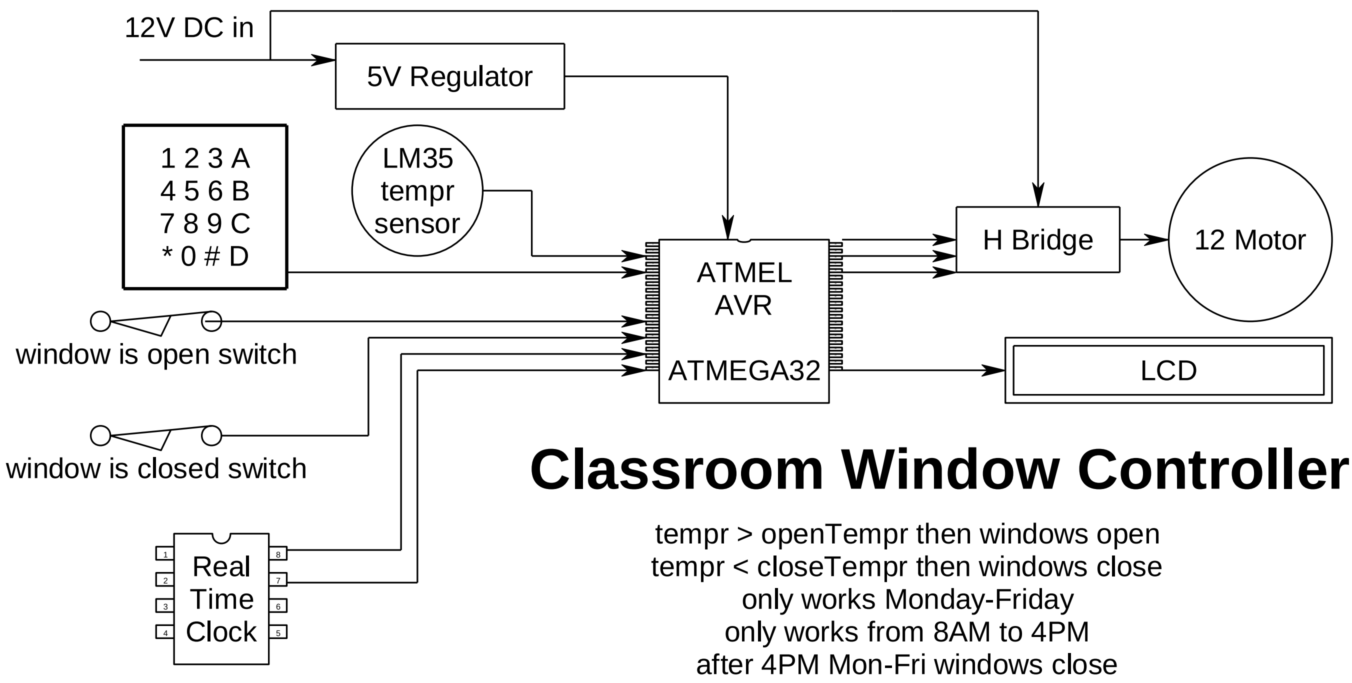
These routines keep track of player movements. We always know the current position and the old position for the refresh display routine.

This gets a little complicated when the player moves off the screen, e.g. when going from left to right it wraps around to the left hand side.

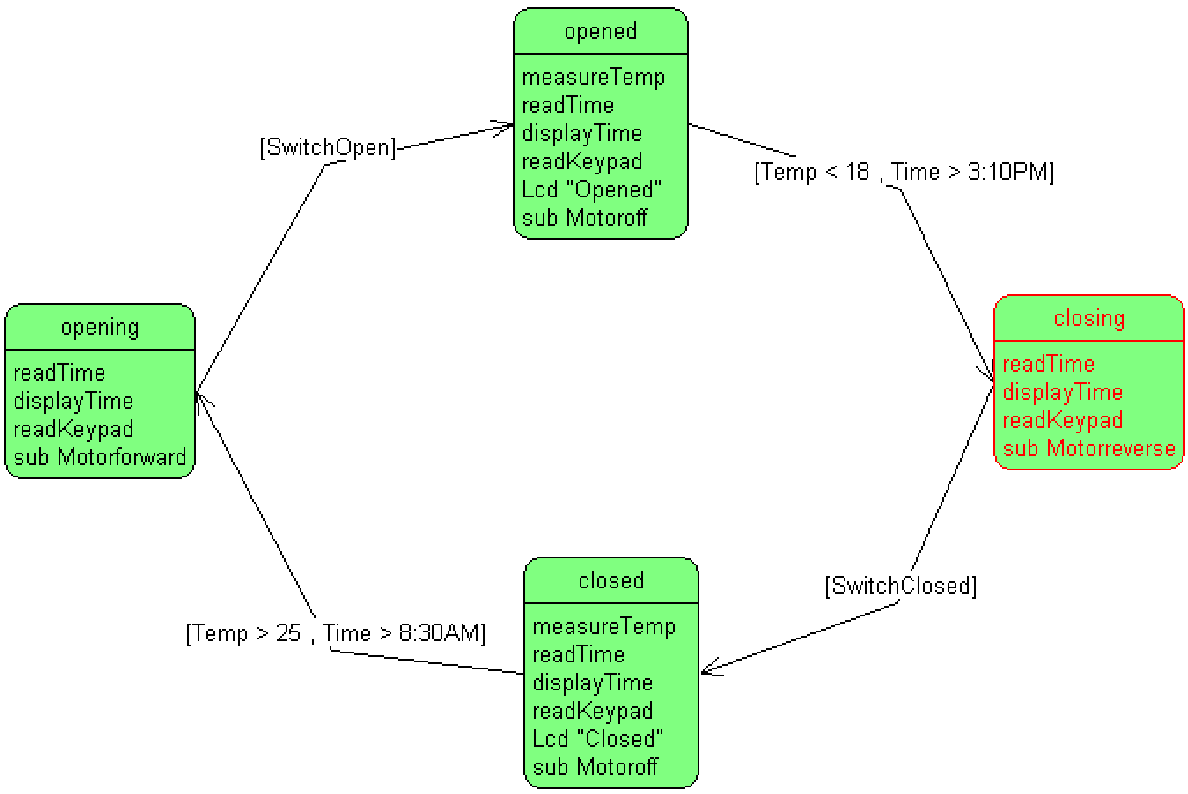
```
Go_up:
    Select Case Ypos
    Case Top_row :
      Oldy = Ypos
      Ypos = Bot_row
      Oldx = Xpos
    Case Is > Top_row
      Oldy = Ypos
      Ypos = Ypos - 1
      Oldx = Xpos
    End Select
Return

Go_down:
    Select Case Ypos
    Case Is < Bot_row :
      Oldy = Ypos
      Ypos = Ypos + 1
      Oldx = Xpos
    Case Bot_row :
      Oldy = Ypos
      Ypos = Top_row
      Oldx = Xpos
    End Select
Return
```

38 Advanced window controller student project

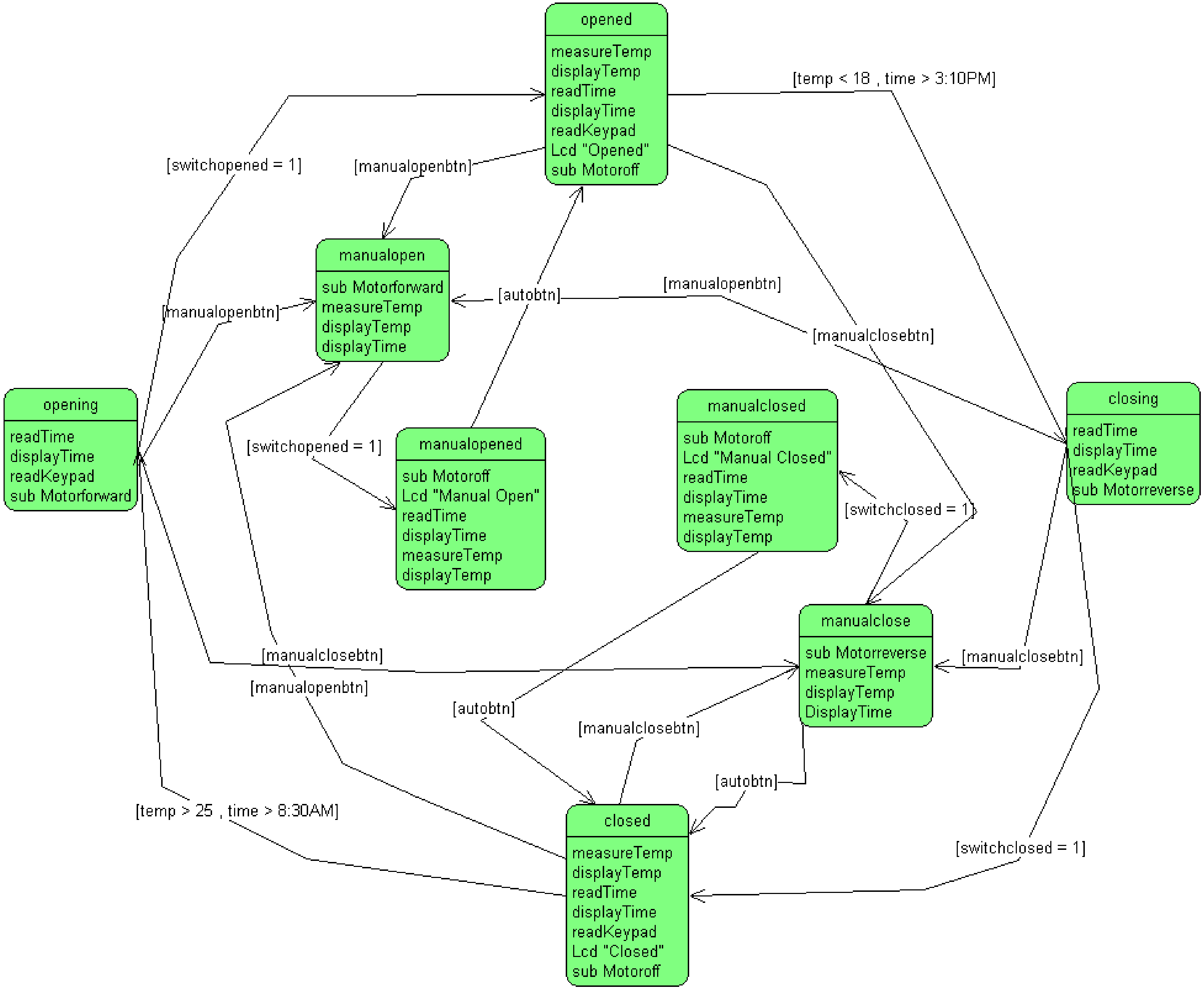
One of my year13 students found a client who wanted an automatic window controller for their classroom. Here is the system block diagram



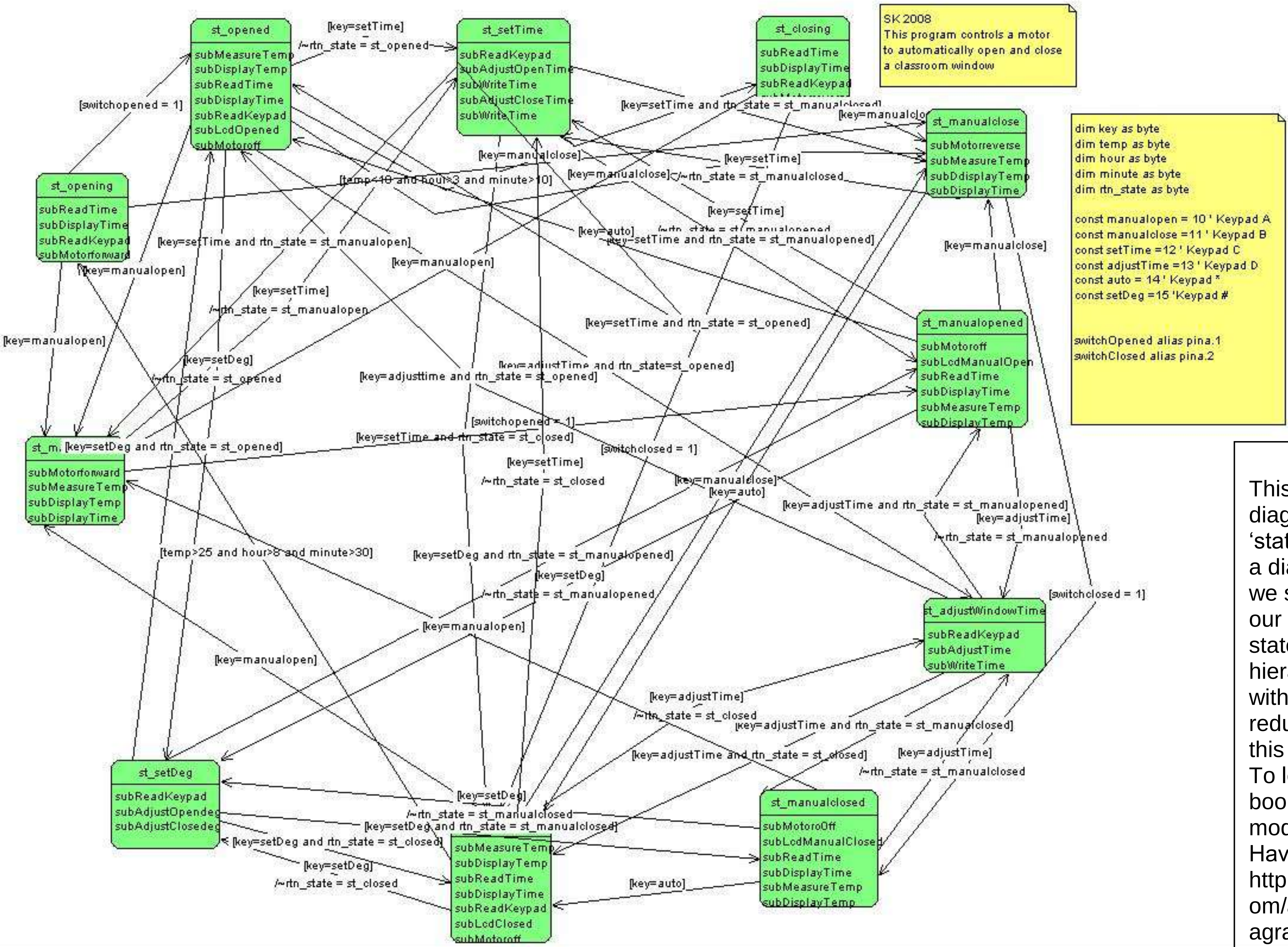
38.1 Window controller state machine #1



38.2 Window controller state machine #3.



It has grown in complexity as he realised that he needed to add more states for the motor while it was on and in the process of closing and opening. The window He also added controls at his clients request for manual open and close.



38.3 Window controller state machine #5

5th and final state machine for the project. Allowed control of the time and temperature settings AND IT WORKED!

This is a very messy diagram as it suffers from 'state explosion'. It is with a diagram such as this that we see the limitations of our process; a true UML statechart allows for hierarchies of states (states within states) and would reduce the complexity of this process immensely.
To learn about this read a book on UML, unified modelling language.
Have a look at…
http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm

38.4 Window controller program

```
'-------------------------------------------------------
'WindowControllerV5b.uss
'Created using StateCharter
'13/09/2009 8:47:12 p.m.
'SK 2008
'This program controls a motor
'to automatically open and close
'a classroom window
'-------------------------------------------------------
'COMPILER DIRECTIVES
$Crystal = 8000000
$regfile = "m8535.dat"
'-------------------------------------------------------
'HARWARE SETUPS
Config PortA=input
Config PortB=output
Config PortC=output
Config PortD=output
'HARWARE ALIASES
switchOpened alias pina.1
switchClosed alias pina.2

'-------------------------------------------------------
'VARIABLES
dim state as byte
dim key as byte
dim temp as byte
dim hour as byte
dim minute as byte
dim rtn_state as byte
'REMEMBER TO INITIALISE YOUR VARIABLES HERE

'-------------------------------------------------------
'STATE CONSTANTS
Const st_manualopened = 1
Const st_adjustWindowTime = 2
Const st_closed = 3
Const st_setDeg = 4
Const st_closing = 5
Const st_setTime = 6
Const st_opening = 7
Const st_manualopen = 8
Const st_opened = 9
Const st_manualclose = 10
Const st_manualclosed = 11
'OTHER CONSTANTS
const manualopen = 10 ' Keypad A
const manualclose =11 ' Keypad B
const setTime =12 ' Keypad C
const adjustTime =13 ' Keypad D
const auto = 14 ' Keypad *
const setDeg =15 'Keypad #
```

```
'--------------------------------------------------------
'PROGRAM STARTS HERE
Do
  while state = st_manualopened
    gosub subMotoroff
    gosub subLcdManualOpen
    gosub subReadTime
    gosub subDisplayTime
    gosub subMeasureTemp
    gosub subDisplayTemp
    if key=adjustTime then
      state = st_adjustWindowTime
      rtn_state = st_manualopened
    end if
    if key=setTime then
      state = st_setTime
      rtn_state = st_manualopened
    end if
    if key=setDeg then
      state = st_setDeg
      rtn_state = st_manualopened
    end if
    if key=manualclose then state = st_manualclose
    if key=auto then state = st_opened
  wend

  while state = st_adjustWindowTime
    gosub subReadKeypad
    gosub subAdjustTime
    gosub subWriteTime
    if key=adjustTime and rtn_state =  st_closed then state = st_closed
    if key=adjustTime and rtn_state = st_manualopened then state = st_manualopened
    if key=adjustTime and rtn_state = st_manualclosed then state = st_manualclosed
    if key=adjusttime and rtn_state = st_opened then state = st_opened
  wend
```

```
while state = st_closed
    gosub subMeasureTemp
    gosub subDisplayTemp
    gosub subReadTime
    gosub subDisplayTime
    gosub subReadKeypad
    gosub subLcdClosed
    gosub subMotoroff
    if key=setDeg then
      state = st_setDeg
      rtn_state = st_closed
    end if
    if key=manualclose then state = st_manualclose
    if key=manualopen then state = st_manualopen
    if temp>25 and hour>8 and minute>30 then state = st_opening
    if key=adjustTime then
      state = st_adjustWindowTime
      rtn_state =  st_closed
    end if
    if key=setTime then
      state = st_setTime
      rtn_state = st_closed
    end if
  wend

  while state = st_setDeg
    gosub subReadKeypad
    gosub subAdjustOpendeg
    gosub subAdjustClosedeg
    if key=setDeg and rtn_state = st_closed then state = st_closed
    if key=setDeg and rtn_state = st_manualopened then state = st_manualopened
    if key=setDeg and rtn_state = st_manualclosed then state = st_manualclosed
    if key=setDeg and rtn_state = st_opened then state = st_opened
  wend

  while state = st_closing
    gosub subReadTime
    gosub subDisplayTime
    gosub subReadKeypad
    gosub subMotorreverse
    if switchclosed = 1 then state = st_closed
    if key=manualopen then state = st_manualopen
    if key=manualclose then state = st_manualclose
  wend
```

```
while state = st_setTime
    gosub subReadKeypad
    gosub subAdjustOpenTime
    gosub subWriteTime
    gosub subAdjustCloseTime
    gosub subWriteTime
    if key=setTime and rtn_state = st_closed then state = st_closed
    if key=setTime and rtn_state = st_manualopened then state = st_manualopened
    if key=setTime and rtn_state = st_manualclosed then state = st_manualclose
    if key=setTime and rtn_state = st_manualopen then state = st_manualopen
    if key=setTime and rtn_state =  st_opened then state = st_opened
  wend

  while state = st_opening
    gosub subReadTime
    gosub subDisplayTime
    gosub subReadKeypad
    gosub subMotorforward
    if key=manualopen then state = st_manualopen
    if switchopened = 1 then state = st_opened
    if key=manualclose then state = st_manualclose
  wend

  while state = st_manualopen
    gosub subMotorforward
    gosub subMeasureTemp
    gosub subDisplayTemp
    gosub subDisplayTime
    if key=setTime then
      state = st_setTime
      rtn_state = st_manualopen
    end if
    if switchopened = 1 then state = st_manualopened
  wend
```

```
while state = st_opened
    gosub subMeasureTemp
    gosub subDisplayTemp
    gosub subReadTime
    gosub subDisplayTime
    gosub subReadKeypad
    gosub subLcdOpened
    gosub subMotoroff
    if key=setTime then
      state = st_setTime
      rtn_state =  st_opened
    end if
    if key=setDeg then
      state = st_setDeg
      rtn_state = st_opened
    end if
    if key=manualclose then state = st_manualclose
    if key=adjustTime and rtn_state=st_opened then state = st_adjustWindowTime
    if temp<18 and hour>3 and minute>10 then state = st_closing
    if key=manualopen then state = st_manualopen
  wend

  while state = st_manualclose
    gosub subMotorreverse
    gosub subMeasureTemp
    gosub subDisplayTemp
    gosub subDisplayTime
    if switchclosed = 1 then state = st_manualclosed
    if key=setTime then
      state = st_setTime
      rtn_state = st_manualclosed
    end if
    if key=auto then state = st_closed
  wend

  while state = st_manualclosed
    gosub subMotoro0ff
    gosub subLcdManualClosed
    gosub subReadTime
    gosub subDisplayTime
    gosub subMeasureTemp
    gosub subDisplayTemp
    if key=adjustTime then
      state = st_adjustWindowTime
      rtn_state = st_manualclosed
    end if
    if key=setDeg then
      state = st_setDeg
      rtn_state = st_manualclosed
    end if
    if key=manualopen then state = st_manualopen
    if key=auto then state = st_closed
  wend
```

```
Loop

'-------------------------------------------------------
'SUBROUTINES

subAdjustClosedeg:
Return

subAdjustCloseTime:
Return

subAdjustOpendeg:
Return

subAdjustOpenTime:
Return

subAdjustTime:
Return

subDisplayTemp:
Return

subDisplayTemp:
Return

subDisplayTime:
Return

subLcdClosed:
Return

subLcdManualClosed:
Return

subLcdManualOpen:
Return

subLcdOpened:
Return

subMeasureTemp:
Return

subMotorforward:
Return

subMotoro0ff:
Return

subMotoroff:
Return
```

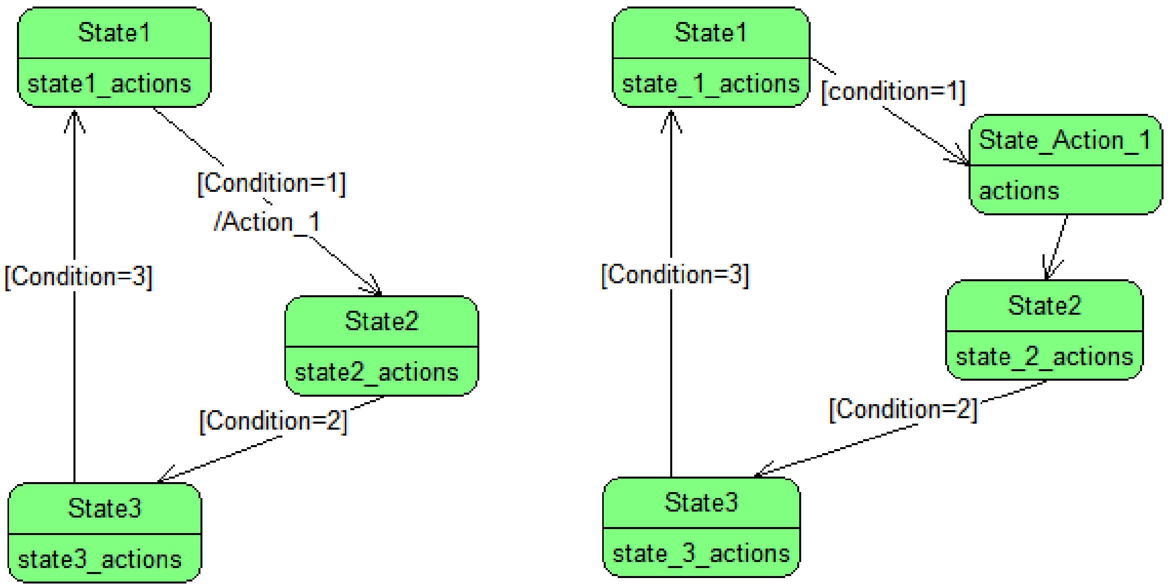
```
subMotorreverse:
Return

subReadKeypad:
Return

subReadTime:
Return

subWriteTime:
Return
```

39 Alternative state machine coding techniques

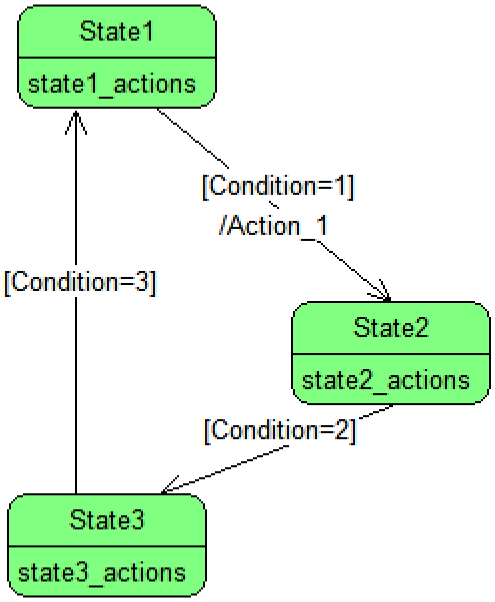
The While-Wend method of coding a state machine is not the only option available to you. Here is an alternative code segment for control of states using a **Select-Case-End-Select** methodology

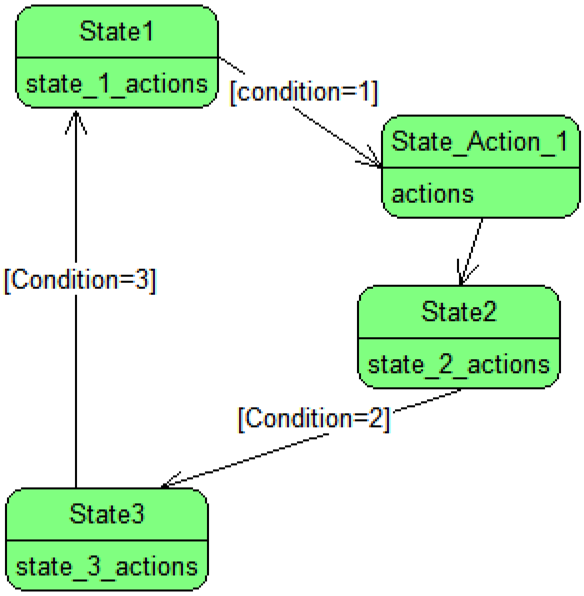
```
Do
        Select Case State
                Case State_1
                        Gosub Actions1a
                        Gosub Actions1b
                        Gosub Actions1c
                Case State_2: Gosub Actions2
                Case State_3 :
                        Gosub Actions3a
                        Gosub Actions3b
                Case State_4 : Gosub Actions4
                Case State_5 : Gosub Actions5
                Case State_6 : Gosub Actions5
        End Select
Loop
```

This code is similar to the previous examples using while wend in that you can still have multiple actions within states. The difference though is that there are no actions perfomed between states. In code like this if you want to perform an action between two states you need to implement another state inbetween the two states as in the example below.



In the state machine above there is an action ACTION_1, that must happen between states, (remember an action is code that will be run only once between states)	In this second state machine Action_1 has been replaced by a state state_action_1, and a second transition that has no condition attached to it. While State1 is executing once condition_1 is met the state will change to Action_1. This code will be executed only once and the state will change automatically to State2.






```
Do
    while state = State1
        If Condition = 1 Then
            state = State2
            Gosub Action_1
        end if
    wend

    while state = State2
        If Condition = 2 Then State = State3
    wend

    while state = State3
        If Condition = 3 Then State = State1
    wend
Loop

Action_1:
    'actions for this state
Return

State1_actions:
    'actions for this state
Return

State2_actions:
    'actions for this state
Return

State3_actions:
    'actions for this state
Return
```

Action_1 will run between state1 and state2, once condition =1 has happened

Condition testing is within the while wend

```
Do
  Select Case State
      Case State1: Gosub State1_actions
      Case State_action_1:  Gosub Actions
      Case State2: Gosub State2_actions
      Case State3: Gosub State_3_actions
  End Select
Loop

State_1_actions:
    'actions for this state
    If Condition = 1 Then State = State_action_1
Return

Actions:
    'actions for this state
    State = State2
Return

State_2_actions:
    'actions for this state
    If Condition = 2 Then State = State3
Return

State_3_actions:
    'actions for this state
    If Condition = 3 Then State = State1
Return
```

Action_1 is a state on its own

Condition testing has moved to the subroutines to keep the select case code tidy, note there is no condition testing in sub actions: for state_action_1

40 Complex - serial communications

Parallel communications is sending data all at once on many wires and serial communications is all about sending data sequentially using a single or a few wires. With serial communications the data is sent from one end of a link to the other end one bit at a time. There are 2 ways of classifying serial data communications.

1. as either Simplex, half duplex or full duplex

And 2. as either synchronous or asynchronous

40.1 Simplex and duplex

In serial communications **simplex** is where data is only ever travelling in one direction, there is one transmitter and one receiver.

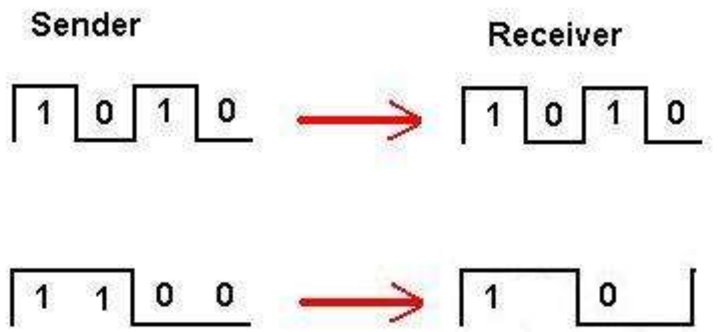
In **half duplex** communications both ends of a link will have a transmitter and receiver but they take turns sending and receiving. A combined transmitter and receiver in one unit is called a transceiver.

In **full duplex** both ends can send and receive data at the same time.

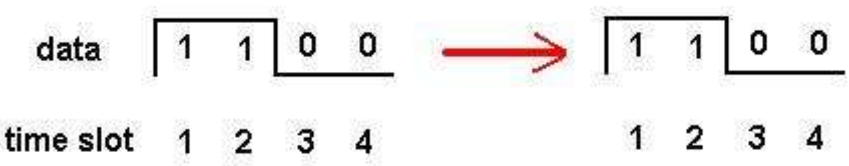
40.2 Synchronous and asynchronous

Imagine sending the data 1010 serially, this is quite straight forward, the sender sends a 1 ,then a 0, then a 1, then a 0. The receiver gets a 1, then a 0, then a 1, then a 0; No problems.

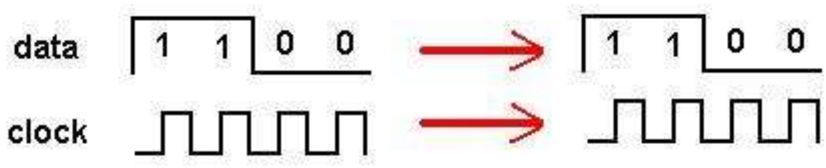
Now send 1100 the sender sends a 1 then a 1 then a 0 then a 0, the receiver gets a one then a zero, hey what happened!!



The receiver has no way of knowing how long a 1 or 0 is without some extra information. In an **asynchronous** system the sender and receiver are setup to expect data at a certain number of bits per second e.g. 19200, 2400. Knowing the bit rate means that the spacing is known and the data is allocated a time slot, therefore the receiver will know when to move on to receiving the next bit.



Synchronous communications is where a second wire in the system carries a clock signal, to tell the receiver when the data should be read.



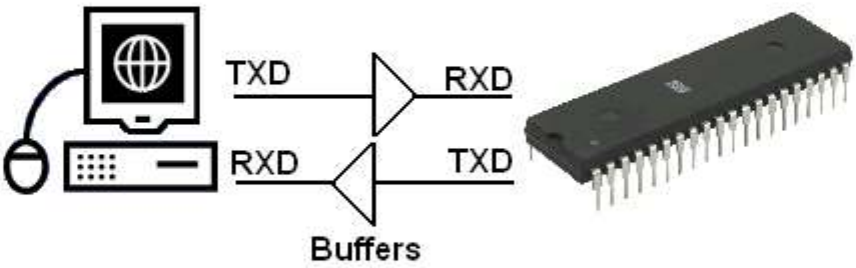
Every time the clock goes from 0 to 1 the data is available at the receiver. Now there is no confusion about when a 1 is present or a zero. The receiver checks the data line only at the right time.

40.3 Serial communications, Bascom and the AVR

The AVR has built in serial communications hardware and Bascom has software commands to use it.

- USART: (universal synchronous and asynchronous receiver transmitter), which when used with suitable circuitry is used for serial communications via RS232. It has separate txd (transmit data) and rxd (receive data) lines, it is capable of synchronous (using a clock line) and asynchronous (no clock line), it is capable of full duplex, both transmitting and receiving at the same time.

  Computers have RS232 (or comm) ports and the AVR can be connected to this (via suitable buffer circuitry)



- SPI: (serial peripheral interface) which has 2 data lines and 1 clock line, these are the three lines used for programming the microcontroller in circuit as well as for communications between the AVR and other devices. This is a synchronous communications interface, it has a separate clock line. It is also full duplex. The 2 data lines are MISO (master in slave out) and MOSI (master out slave in) these are full duplex, because data can travel on the 2 lines at the same time.

Bascom also has libraries of software commands built into it for two other communications protocols

- I2C: (pronounced I squared C) this stands for Inter IC bus, it has 1 data line and 1 clock line. Because it has only 1 data line it is half duplex, the sender and receiver take turns, and because it has a clock line it is synchronous.
- Dallas 1-Wire: this is literally 1 wire only, so the data is half duplex, and asynchronous.

40.4 RS232 serial communications

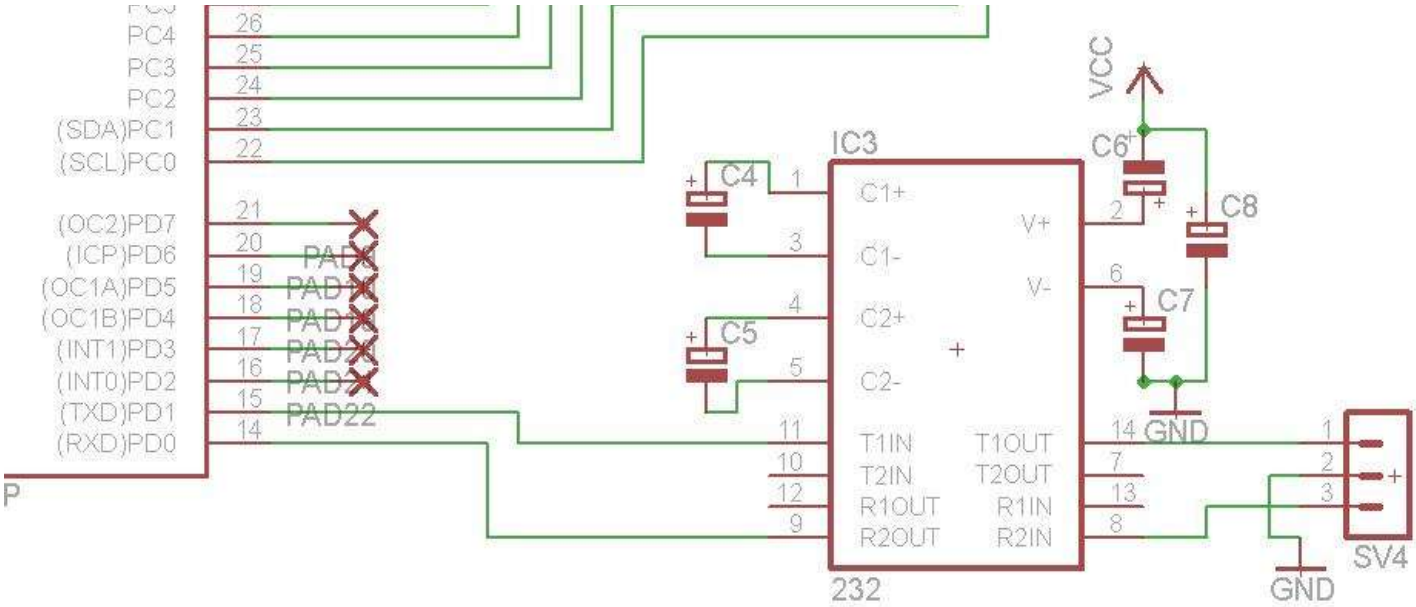
RS232/Serial communications is a very popular communications protocol between computers and peripheral devices such as modems. It is an ideal communication medium to use between a PC and the microcontroller.

The different parts of the RS232 system specification include the plugs, cables, their functions and the process for communications. The plugs have either 9 or 25 pins, more commonly today the PC has two 9 pin male connectors.

There are two data lines one is TXD (transmit data) the other RXD (receive data), as these are independent lines devices can send and receive at the same time, making the system full duplex. There is a common or ground wire and a number of signal wires.

There is no clock wire so the system of communications is asynchronous. There are a number of separate control lines to handle 'handshaking' commands, i.e. which device is ready to transmit, receive etc.

The AVR microcontroller has built in hardware to handle RS232 communications, the lines involved are portd.0 (RXD) and portd.1 (TXD). These two data lines however cannot be directly connected to a PCs RS232 port because the RS232 specification does not use 5V and 0V, but +15V as a zero and -15V as a one. Therefore a buffer circuit is required, the MAX232 is a common device used for this.



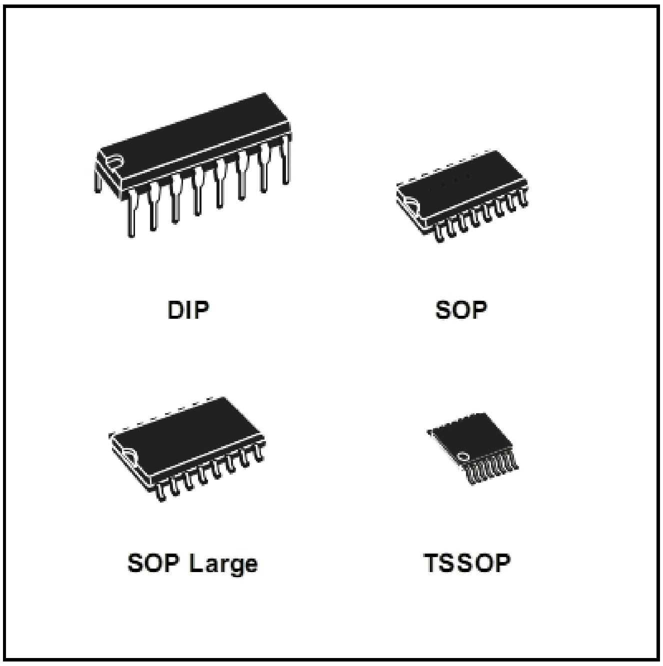
A connector (DB9-Female) is required for the PC end and a simple 3 way header can be used on the PCB (SV4 in the diagram)
TXD (PortD.1) will go through the buffer in the Max232 then the header to pin 2 of the DB9
RXD(PortD.0) comes from the buffer of the MAX232 which is connected to pin3 of the DB9

The ‘MAX232’ is a common chip used; in the classroom we have the ST232, the capacitors we use with the ST232 do not need to be polarised and 0.1uF values will do. It will give +/- 8V.

ST232

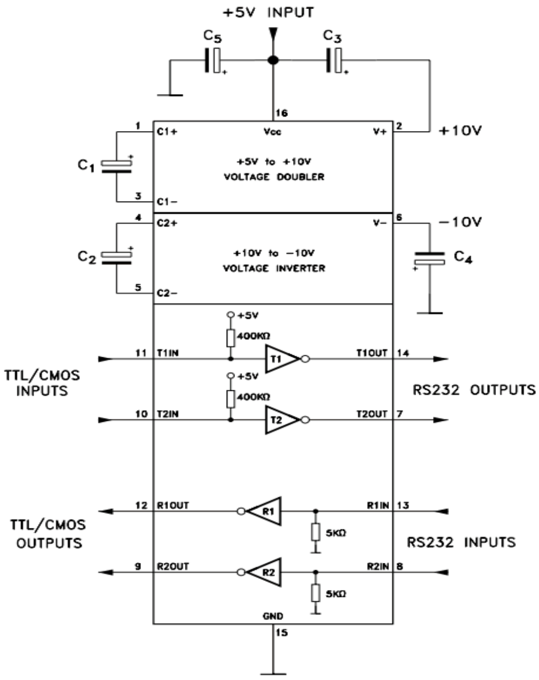
5V POWERED MULTI-CHANNEL RS-232 DRIVERS AND RECEIVERS

- SUPPLY VOLTAGE RANGE: 4.5 TO 5.5V
- SUPPLY CURRENT NO LOAD (TYP): 5mA
- TRANSMITTER OUTPUT VOLTAGE SWING (TYP): ±7.8V
- CONTROLLED OUTPUT SLEW RATE
- RECEIVER INPUT VOLTAGE RANGE: ±30V
- DATA RATE (TYP): 220Kbps
- OPERATING TEMPERATURE RANGE: -40 TO 85°C, 0 TO 70°C
- COMPATIBLE WITH MAX232 AND MAX202



DESCRIPTION

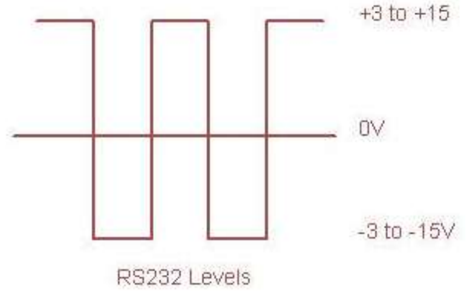
The ST232 is a 2 driver, 2 receiver device following EIA/TIA-232 and V.28 communication standard. It is particularly suitable for applications where ±12V is not available. The ST232 uses a single 5V power supply and only four external capacitors (0.1µF). Typical applications are in: Portable Computers, Low Power Modems, Interfaces Translation, Battery Powered RS-232 System, Multi-Drop RS-232 Networks.



The ST232 (and MAX232) have two sets of buffers so two separate devices can be connected to the AVR at the same time. Some ATMega chips have two UARTs and if your ATMega has only one that is ok as BASCOM has the software built into it to handle software UARTs.

40.5 Build your own RS232 buffer

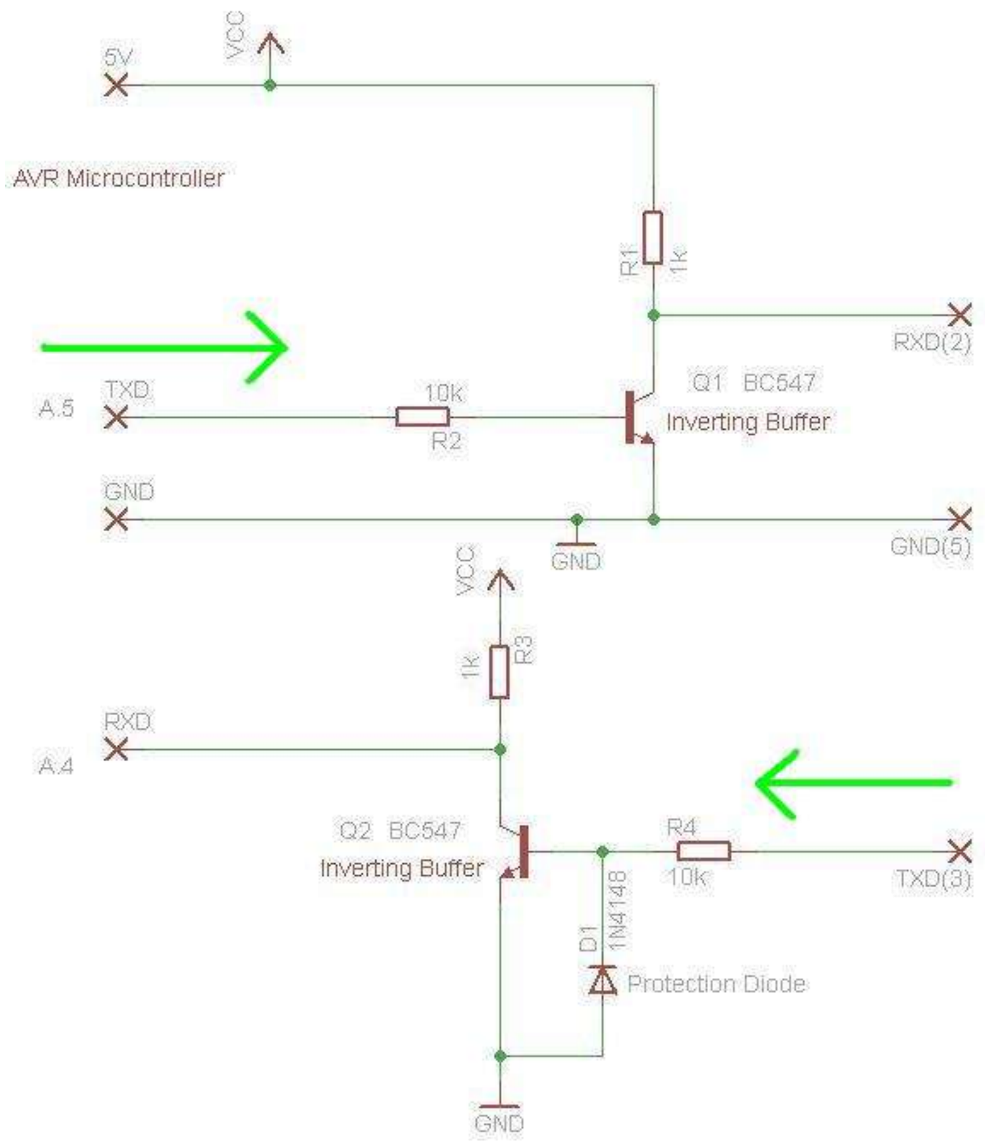
Why do we need a buffer again?



RS232 is designed to send data over reasonable distances between different devices that might run on different voltages.

To do this the designers of the specification decided that a transmitter could send up to +/- 15VDC and a receiver should be able to reliably detect signals if the voltages were as low as +/- 3VDC.

Note that a ‘1’ is 5V for a microcontroller and -3 to -15 for a RS232(it is inverted).



It is easy to build a simple transitor circuit to achieve this buffering for us (it is however not a perfect circuit).

<u>AVR to RS232</u>
When the AVR transmits it switches from 0V to 5V and the output to the RS232 actually only switches between 5V and 0V, this is outside the RS232 specification of -3V, but it seems to work OK most of the time.

<u>RS232 to AVR</u>
The input to the AVR is more accurate as it converts the +V input to 0V and the –V to 5V (note the diode protects the transistor by not allowing the base voltage to go below -0.6V).

40.6 Talking to an AVR from Windows XP

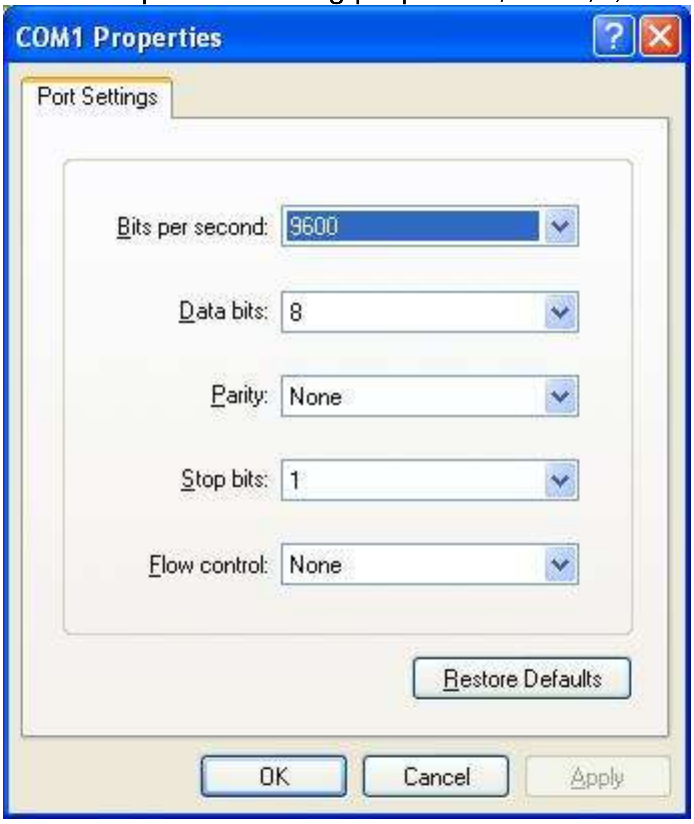
There are several different software options for communicating over rs232 from the AVR, the simplest is the print statement.

print "hello" will send the ASCII text string to the pc. At the pc end there must be some software listening to the comport, Windows has **HyperTerminal** already built in to do this.

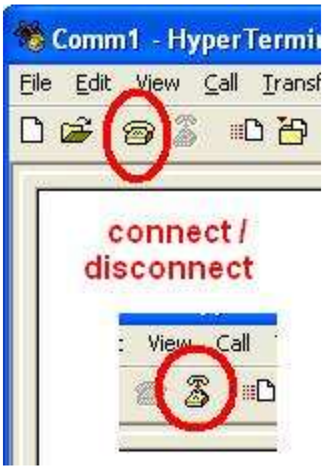
Open HyperTerminal (normally found in programs/accessories/communications).
Start a new connection and name it comm1

On the next screen make sure you select comm1 as the port.

Then setup the following properties, 9600,8, none, 1, none



When you click on OK HyperTerminal can now send and receive using comm1.

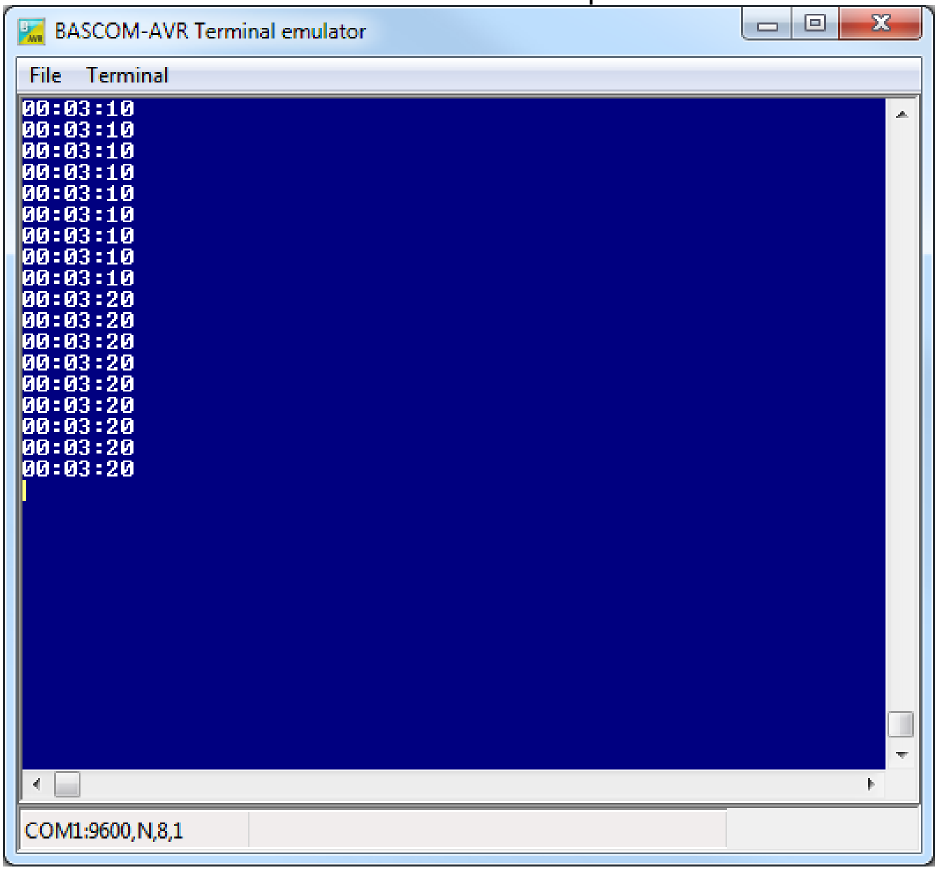


If nothing happens make sure the communications is connected.

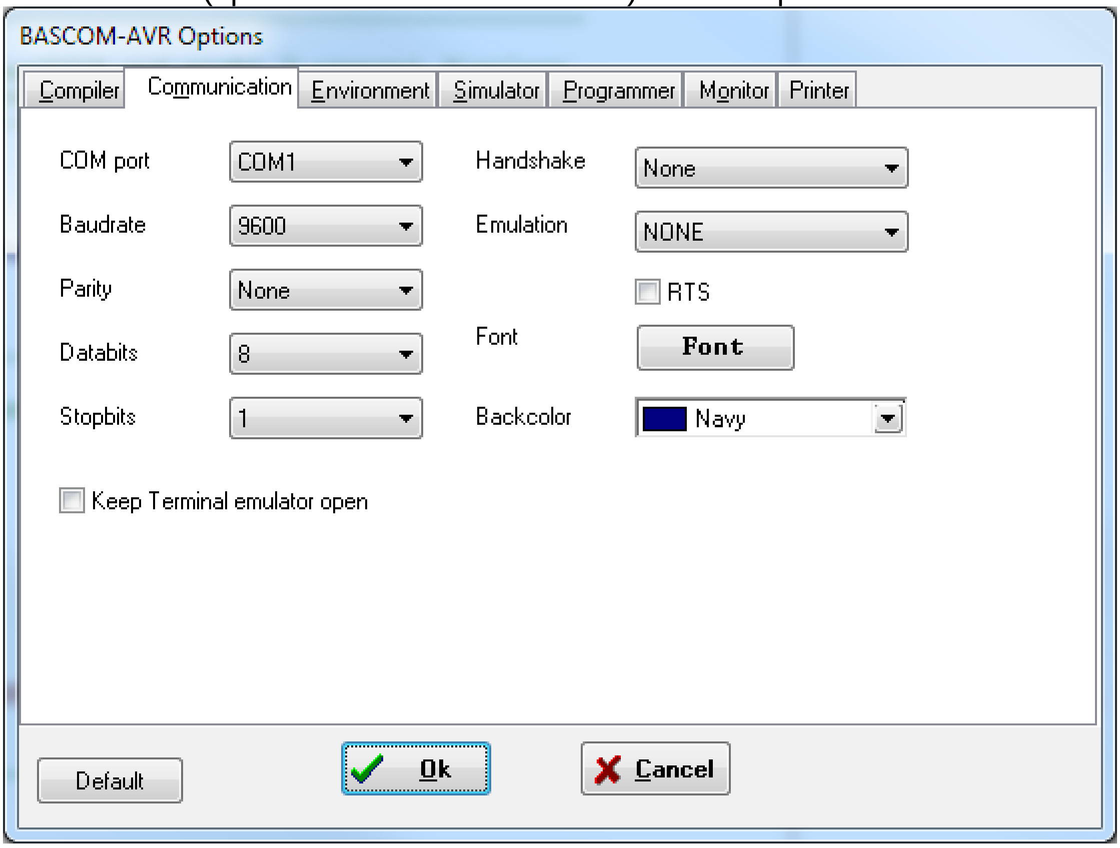
There are many many different communication programs on the internet to try, Termite is one that is useful.

40.7 Talking to an AVR from Win7

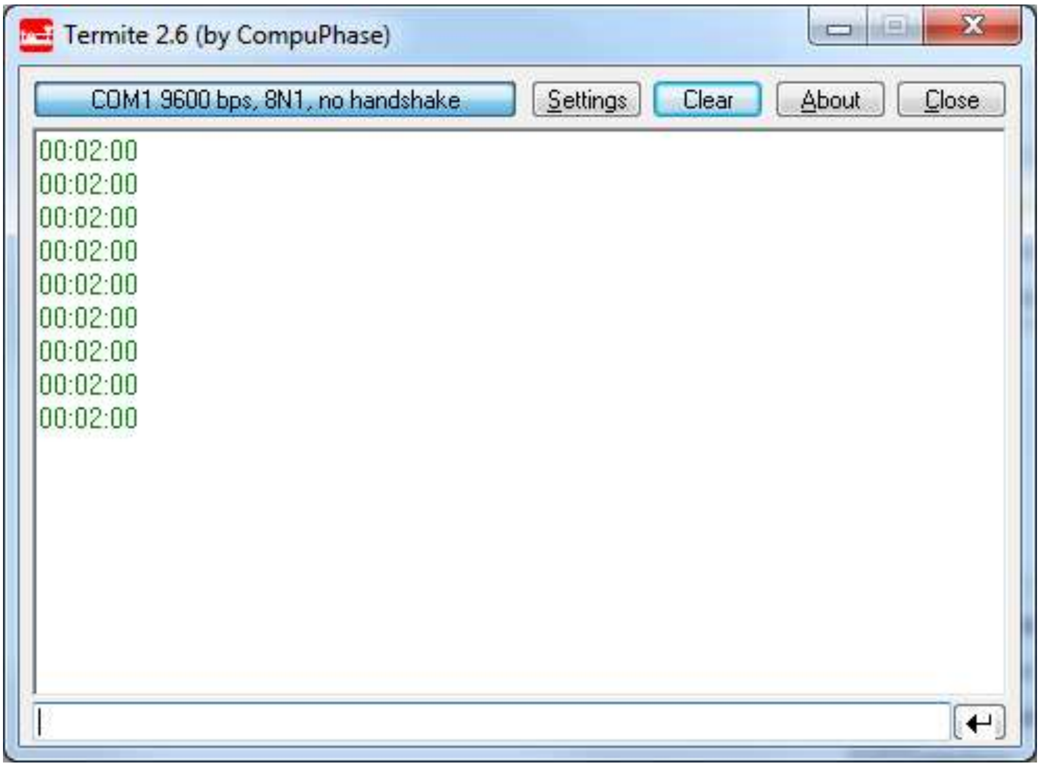
Hyper terminal no longer exists in Windows7, but there are many useful applications that we can use. Bascom has abuilt in terminal under the options menu.

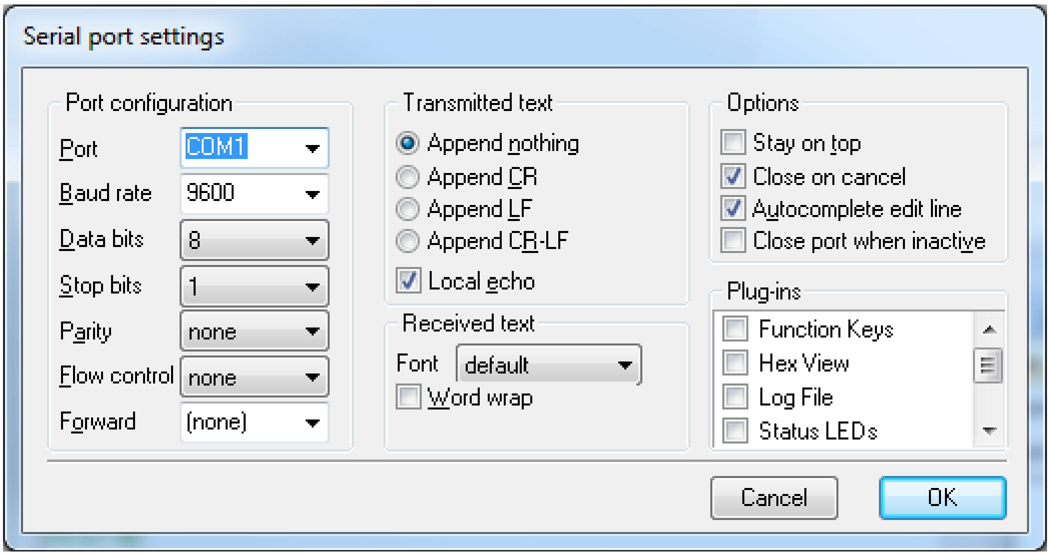


use the menu (options then communications) to set it up



Termite 2.6 is a comprehensive free program

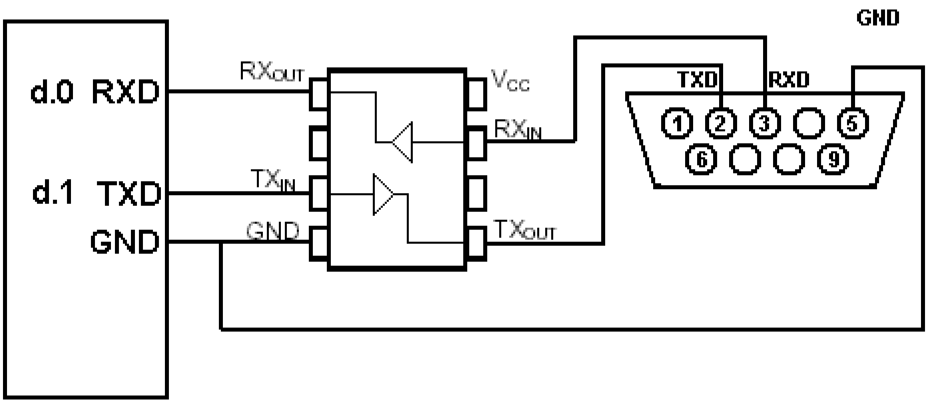


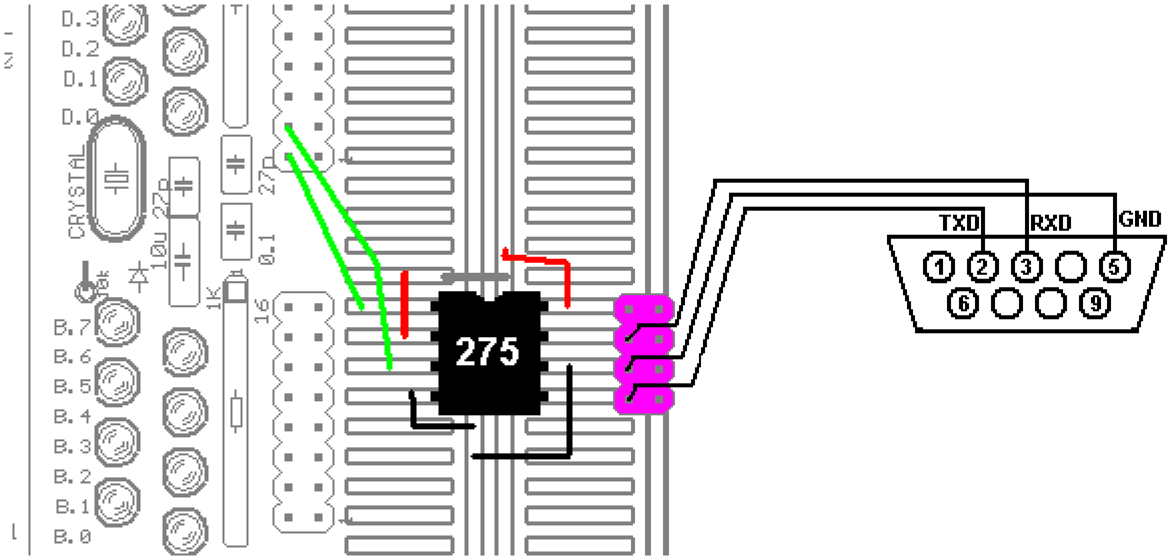


40.8 First Bascom RS-232 program

```
' Hardware Features:
' MAX232 connected to the micro TXD and RXD lines. then wired to a DB9F.
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' Program Features:
' print statement
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000            'the speed of operations inside the micro
$regfile = "m8535.dat"        ' the micro we are using
$baud = 9600                  'set data rate for serial comms
'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 40 * 2 'configure lcd screen
' Hardware Aliases

'-----------------------------------------------------------------
' Declare Constants
Const Timedelay = 500
'-----------------------------------------------------------------
' Declare Variables
Dim Count As Byte
' Initialise Variables
Count = 0
'-----------------------------------------------------------------
' Program starts here
Print "Can you see this"
Do
   Incr Count
   Cls
   Lcd Count
   Print " the value is " ; Count
   Waitms Timedelay
Loop
End 'end program
'-----------------------------------------------------------------
```

Another useful interface (if you have easy access to the IC) is the DS275. No capacitors just the IC and a three pin header. I always wire up the three pin headers with ground in the middle, it means that if you get the wiring wrong all you have to do is unplug it and try it in reverse!





40.9 Receiving text from a PC

```
' Hardware Features:
' DS275 connected to the micro TXD and RXD lines. then wired to a DB9F.
' Program Features:
' input statement
' string variables
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000 'the crystal we are using
$regfile = "m8535.dat" 'the micro we are using
$baud = 9600 'set data rate for serial comms
'-----------------------------------------------------------------
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 40 * 2 'configure lcd screen
' 7. Hardware Aliases
Cls
Cursor Noblink
'-----------------------------------------------------------------
```

```
' 9. Declare Constants
Const Timedelay = 2
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Text As String * 15
' 11. Initialise Variables
Text = ""
'-----------------------------------------------------------------
' 12. Program starts here
Print "Can you see this"
Do
   Input "type in something" , Text
   Lcd Text
   Wait Timedelay
   Cls
Loop
End 'end program
'-----------------------------------------------------------------
' 13. Subroutines
```

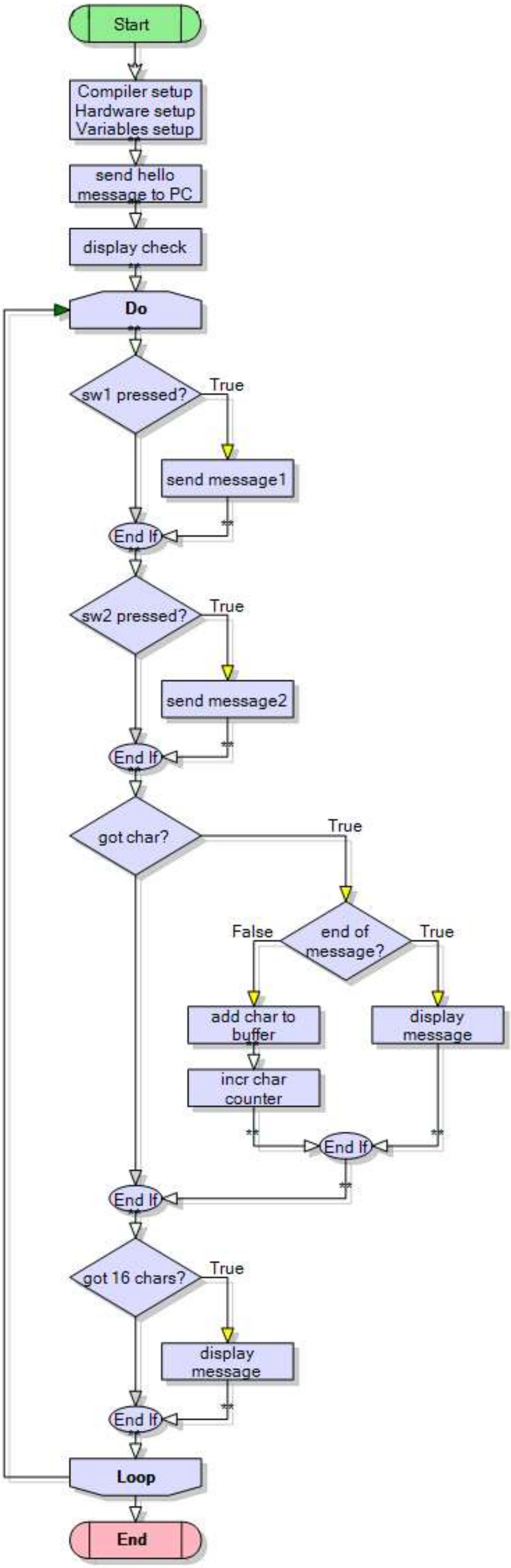
40.10 BASCOM serial commands

There are a number of different serial commands in Bascom to achieve different functions, find these in the help file and write in the description of each one.

Print
PrintBin
Config SerialIn
Config SerialOut
Input
InputBin
InputHex
Waitkey
Inkey
IsCharWaiting
$SerialInput2LCD
$SerialInput
$SerialOutput
Spc

Some AVRs have more than one UART (the internal serial device) and it is possible to have software only serial comms in Bascom and use
Serin, Serout,
Open
Close
Config Waitsuart

40.11 Serial IO using Inkey()




```
'-----------------------------------------------------------------
' Title Block
' Author:    B.Collis
' Date:      22 Aug 03
'-----------------------------------------------------------------
' Program Description:
' This program receives characters from the
RS232/comm/serial port of a PC
' and  displays them on the LCD
' Hardware Features:
' MAX232 connected to the micro TXD and RXD
lines. then wired to a DB9F.
' LCD on portc - note the use of 4 bit mode and
only 2 control lines

' Program Features:
' print statement
' message buffer
' inkey reads the serial buffer to see if a char has
arrived
' note that a max of 16 chars can arrive before the
program
' automatically prints the message on the LCD
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things
about our hardware)
$crystal = 8000000                'the crystal we
are using
$regfile = "m8535.dat"            'the micro we
are using
$baud = 9600                      'set data rate for
serial comms
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Pinb.0 = Input
Config Pinb.1 = Input
Config Portc = Output
Config Portd = Output
Config Pind.2 = Input
Config Pind.3 = Input
Config Pind.6 = Input
```

```
Config Lcd = 40 * 2                    'configure lcd screen
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 ,
Rs = Portc.0

Config Serialin = Buffered , Size = 20  'buffer the incoming data
' 7. Hardware Aliases
Sw_1 Alias Pinb.0
Sw_2 Alias Pinb.1
Sw_3 Alias Pind.2
Sw_4 Alias Pind.3
Sw_5 Alias Pind.6
' 8. initialise ports so hardware starts correctly
'-----------------------------------------------------------------
' 9. Declare Constants

'-----------------------------------------------------------------
' 10. Declare Variables
Dim Count As Byte
Dim Char As Byte
Dim Charctr As Byte
Dim Message As String * 16

' 11. Initialise Variables
Count = 0
'-----------------------------------------------------------------
' Program starts here
Enable Interrupts                   'used by the serial buffer
Print "Hello PC"
Cls
Lcd "LCD is ok"
Wait 3
Do
  Debounce Sw_1 , 0 , Sub_send1 , Sub  'when switch pressed
  Debounce Sw_2 , 0 , Sub_send2 , Sub  'when switch pressed
  Char = Inkey()                    'get a char from the serial buffer
  Select Case Char                  'choose what to do with it
    Case 0 :                        ' no char  so do nothing
    Case 13 : Gosub Dispmessage       'Ascii 13 is CR so show
    Case Else : Incr Charctr          'keep count of chars
       Message = Message + Chr(char)       'add new char
  End Select
  If Charctr > 15 Then                'if 16 chars received
    Gosub Dispmessage                   'display the message anyway
  End If
Loop
End                                 'end program
```

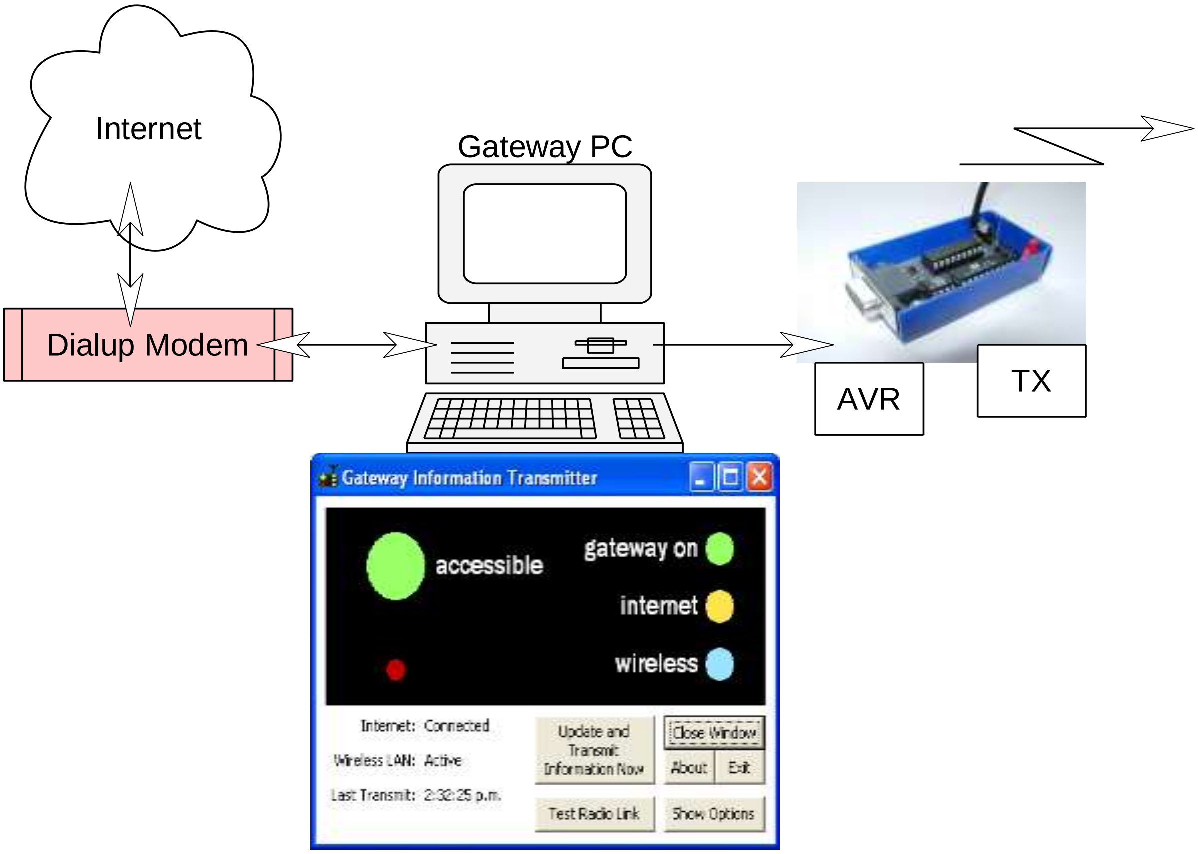
```
'-----------------------------------------------------------------
' 13. Subroutines

Sub_send1:
  Print "this is hard work"          'send it to comm port
Return

Sub_send2:
  Print "not really"                 'send it to comm port
Return

Dispmessage:
  Cls
  Lcd Message
  Message = ""
  Charctr = 0
  Incr Count                         'send some data to the comm port
  Print "you have sent = " ; Count ; " messages"
Return
'-----------------------------------------------------------------
' 14. Interrupts
```

Inkey allows you to manage the input of characters yourself, but you have to poll (check) regularly that a character is there and process it or it will disappear when a new one comes in (the AVR's have a USART with error detecting that can inform you if you have missed reading the buffer, you might want to get to understand that if you are going to do commercial programms). There are also interrupts built into the AVR for serial USART comms, but these are not implemented in BASCOM.

40.12 Creating your own software to communicate with the AVR

Several student projects have incorporated PC based software that communicates with an AVR.

In this project CZL built a unit that informed remote users in the building that a gateway was on, the internet was connected and that the wireless network was up.



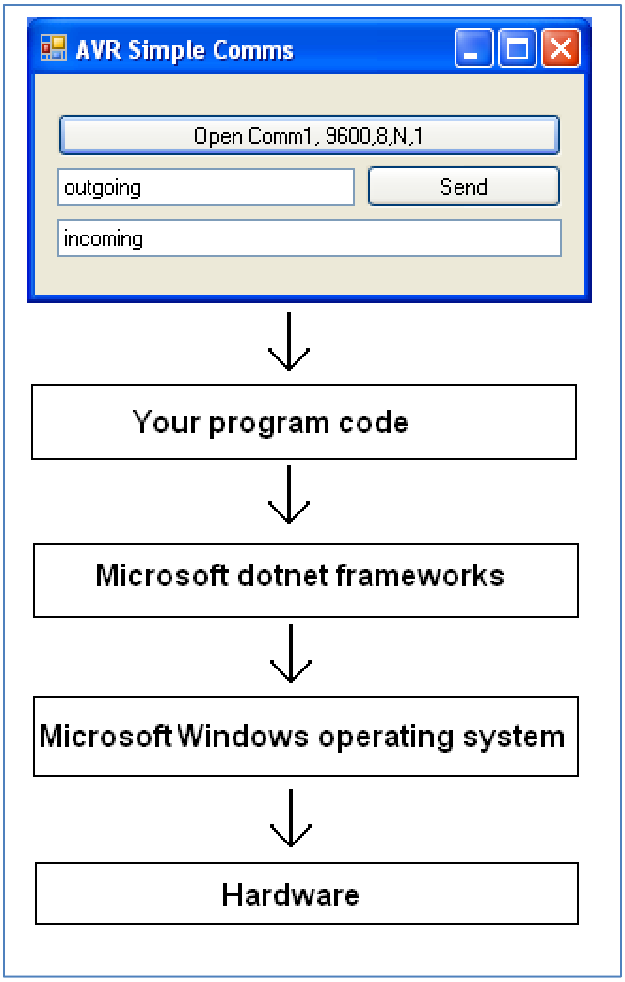
The receiver consisted of a single box of receiver, decoder and AVR.

At this point we are interested in the PC software. It is written in Visual Basic 6. There isn't much point in going into VB6 as it has been superceeded by Visual Studio (currently 2010) and the Expres edition is available freely from Microsoft.

40.13 Microsoft Visual Basic 2008 Express Edition

To begin you must understand just a little about how Windows based programs work, their different parts and what they are called.

Programs you write for a pc make use of the software already on the PC, this way you don't have to figure out how to draw lines on the screen and check where the mouse is and how to read and write to hard drives etc etc.



What we think of a program is a GUI (graphical user interface) to…

Your **functions**(subs or subroutines) in your program code which is written in Visual basic (or C#) which uses …

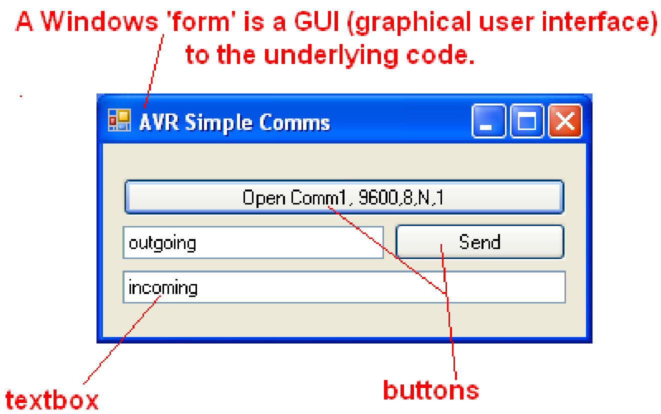
Microsoft dotnet functions within which use…

Windows operating system functions which requires…

Drivers and hardware such as a PC with an Intel or AMD microprocessor.

(THIS ISNT THE HARD BIT; THAT COMES IN A FEW PAGES)

The actual program is called a **form**, with **controls** on it.

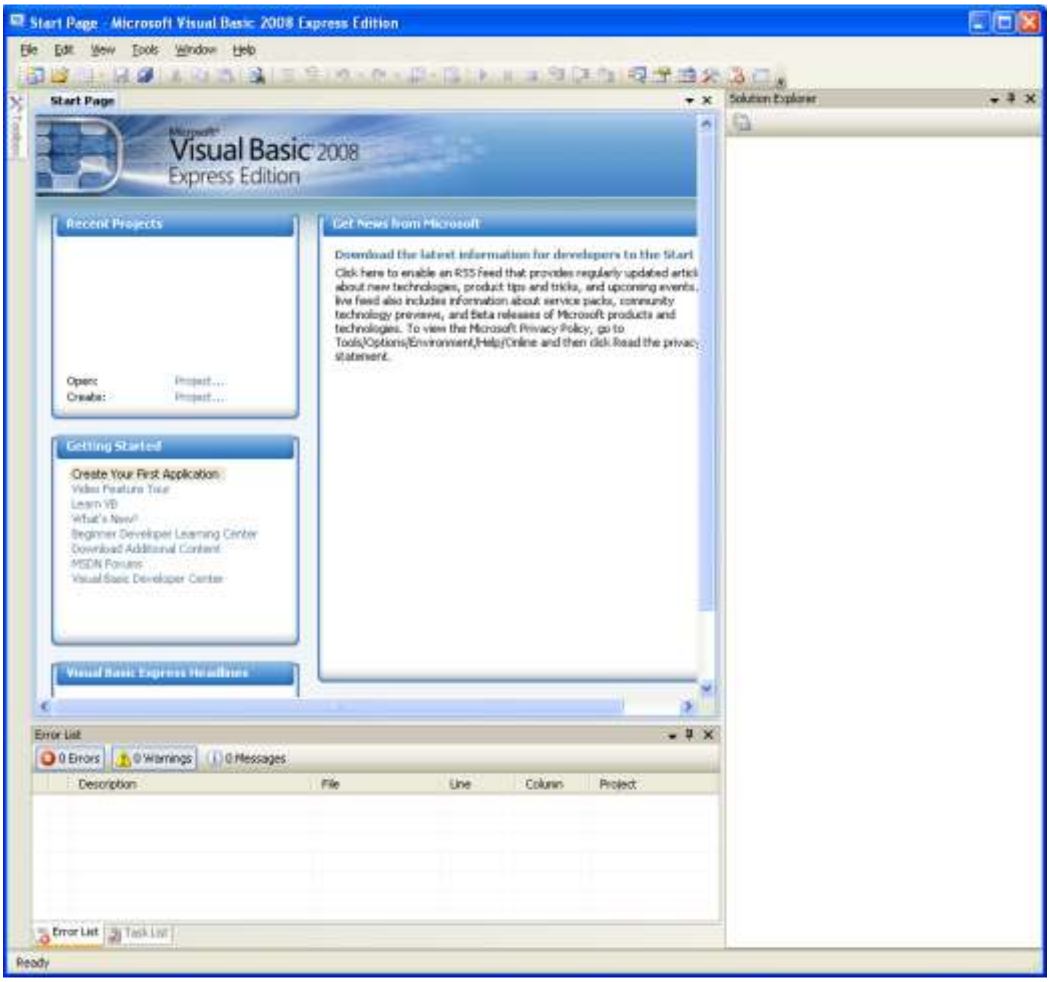


Textboxes and **buttons** are examples of controls on a form

Controls have **properties** such as a 'name' property and a 'text' property (things written on the control) as well as many other properties

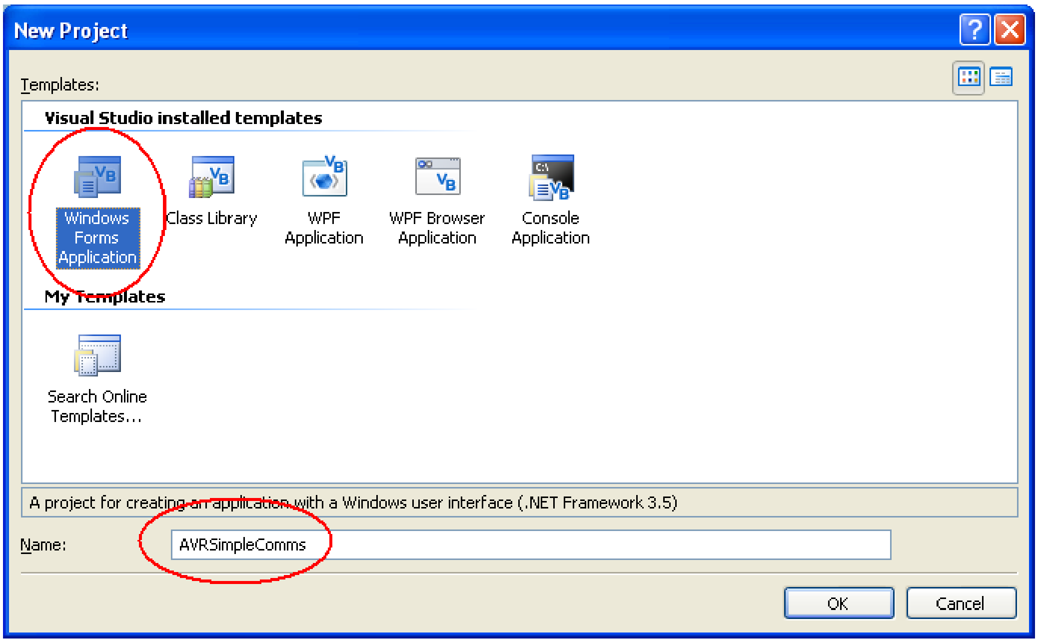
Take note of the words GUI, form, control, textbox, button, property and function.

First make sure you have installed the latest version of Microsoft Visual studio and dotnet (free from www.microsoft .com)

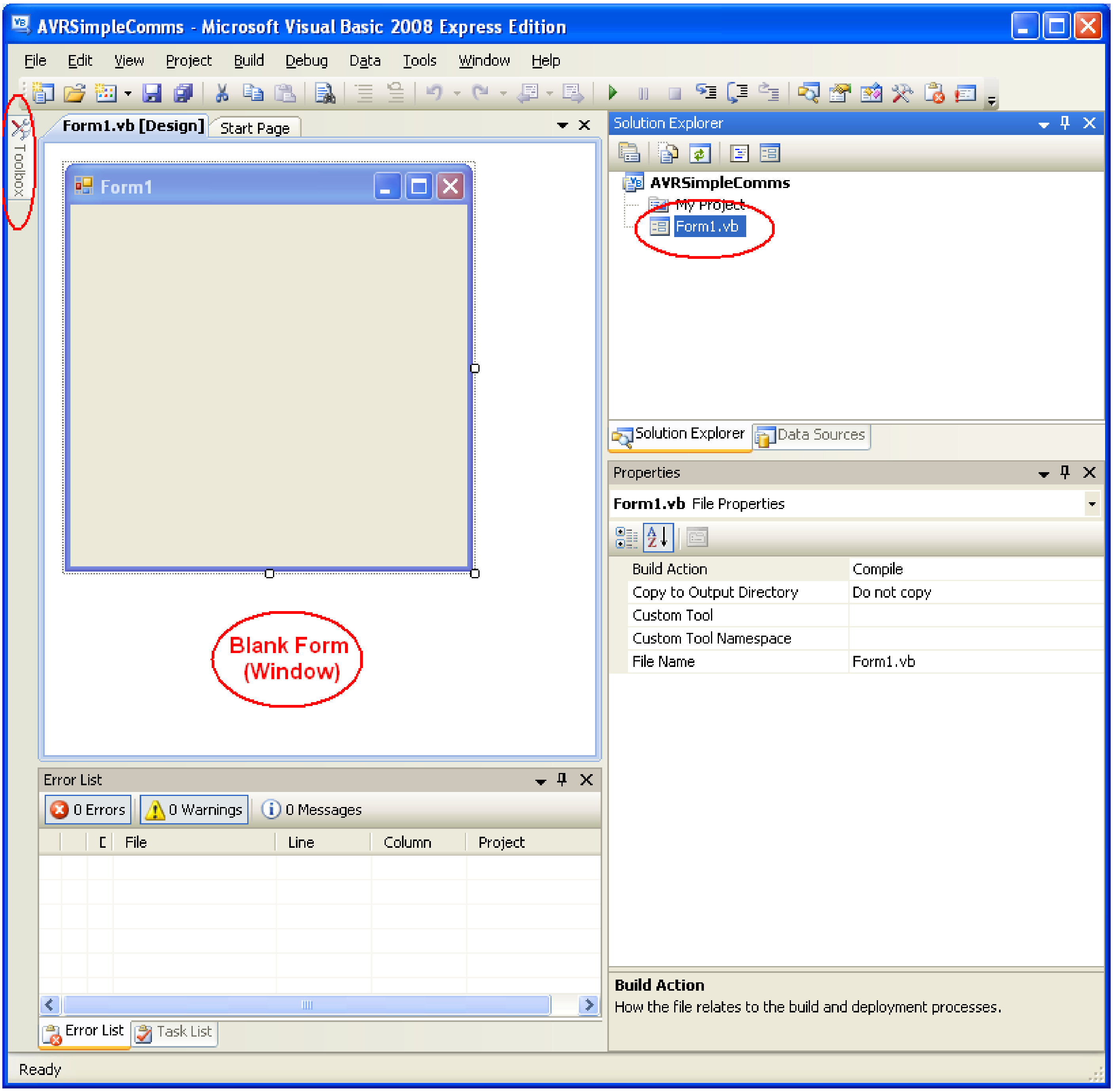


40.14 Stage 1 – GUI creation

From the menu select **file** then **new …**



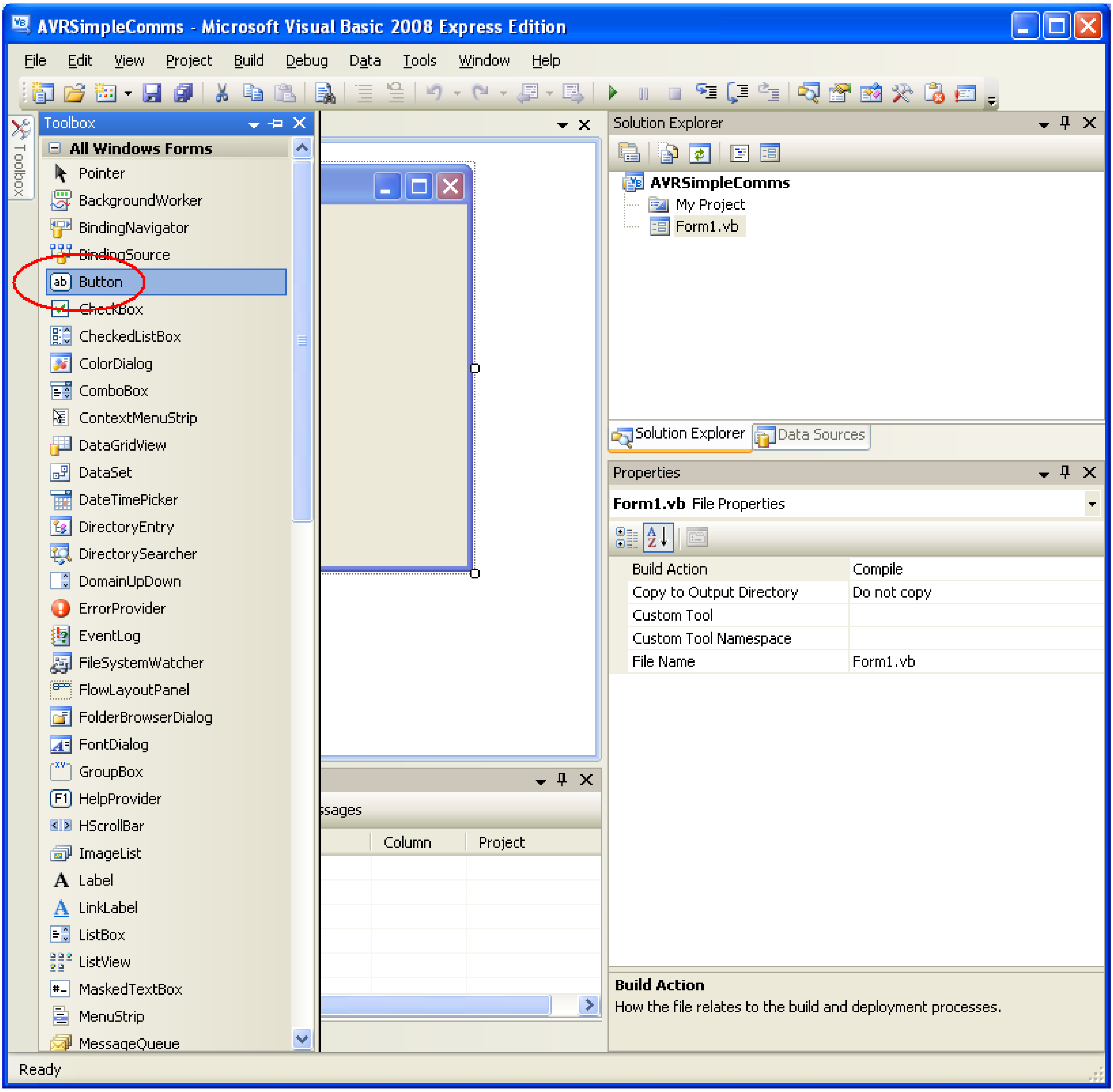
Select Windows form application and name it **AVRSimpleComms**



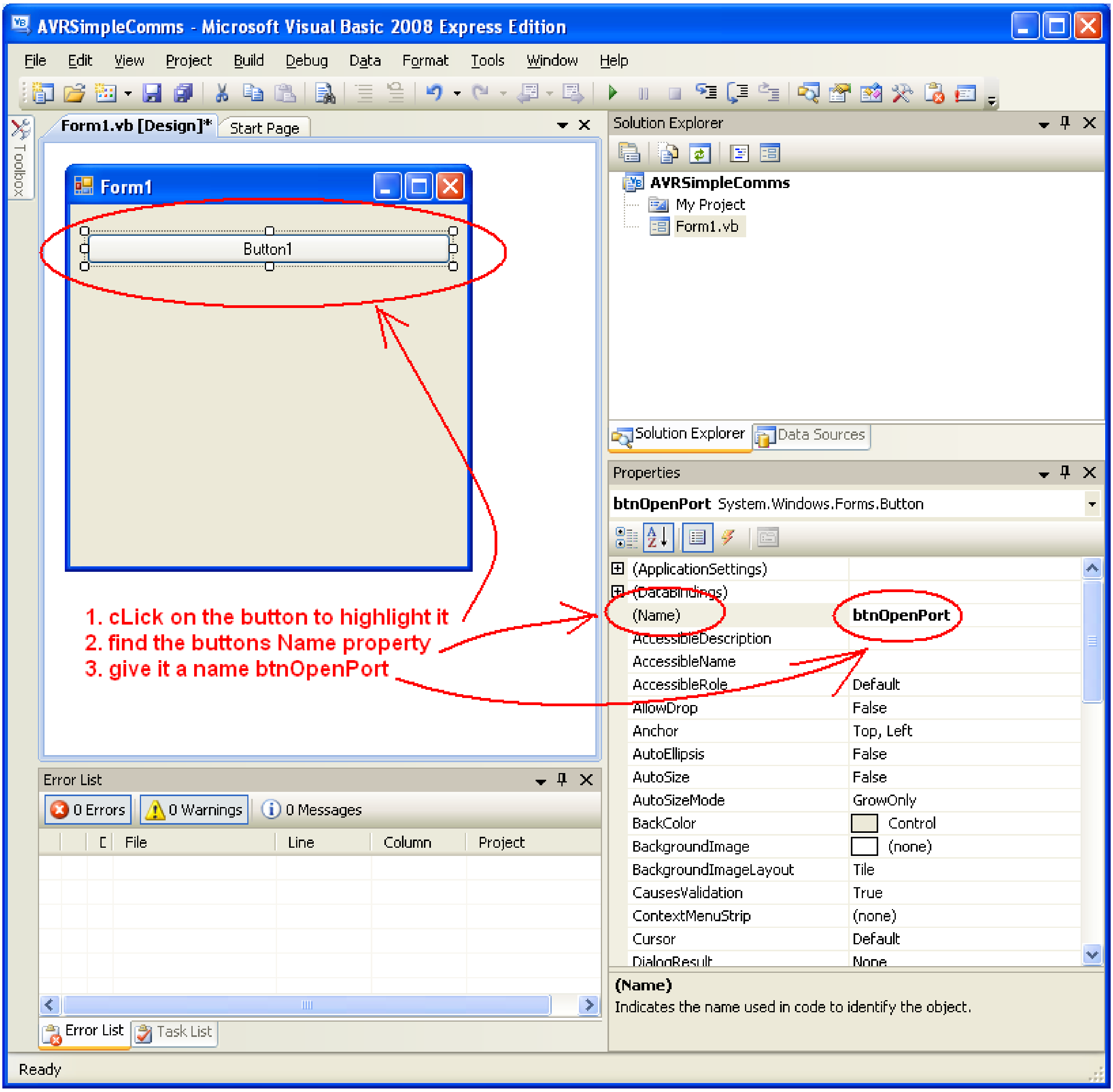
A blank **form** will appear where you can add **controls**.

If you cannot see the form or it disappears at any stage behind new strange looking windows with code in them, then click on **Form1.vb** in the solution explorer on the right hand side or select the **Form1.vb(Design)** Tab.

Adding a control is easy click on the Toolbox popup on the very left hand side of the screen…



Select the Button control and double click it or drag it onto your form.



Controls such as buttons have lots and lost of properties.

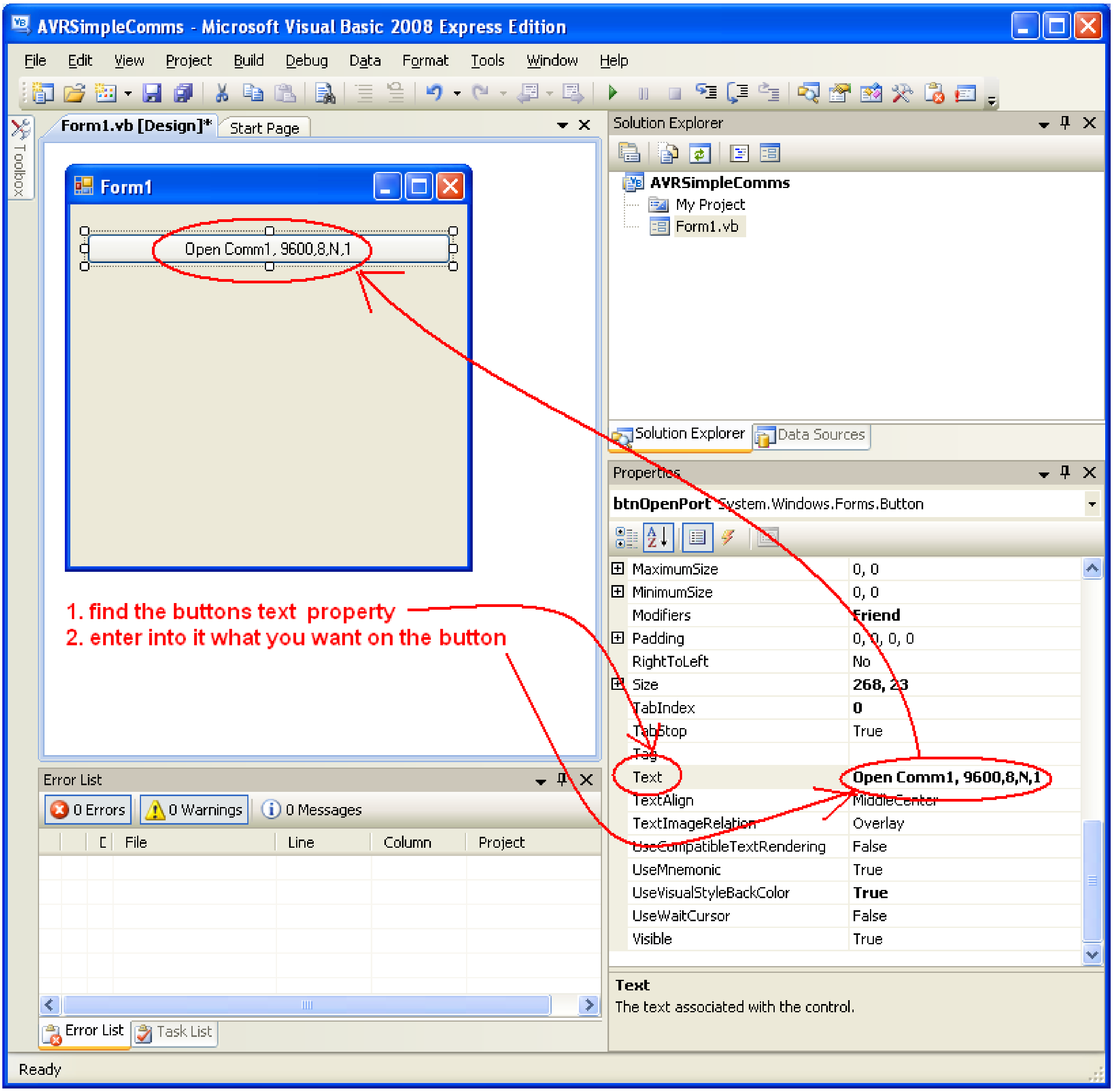
Click on the button to highlight it, change its size by dragging the corners and locate it in the upper area of the form.

On the left hand side you should see the properties, find the Name property the default name **Button1** is no use to us when programming so change its name to **btnOpenPort**

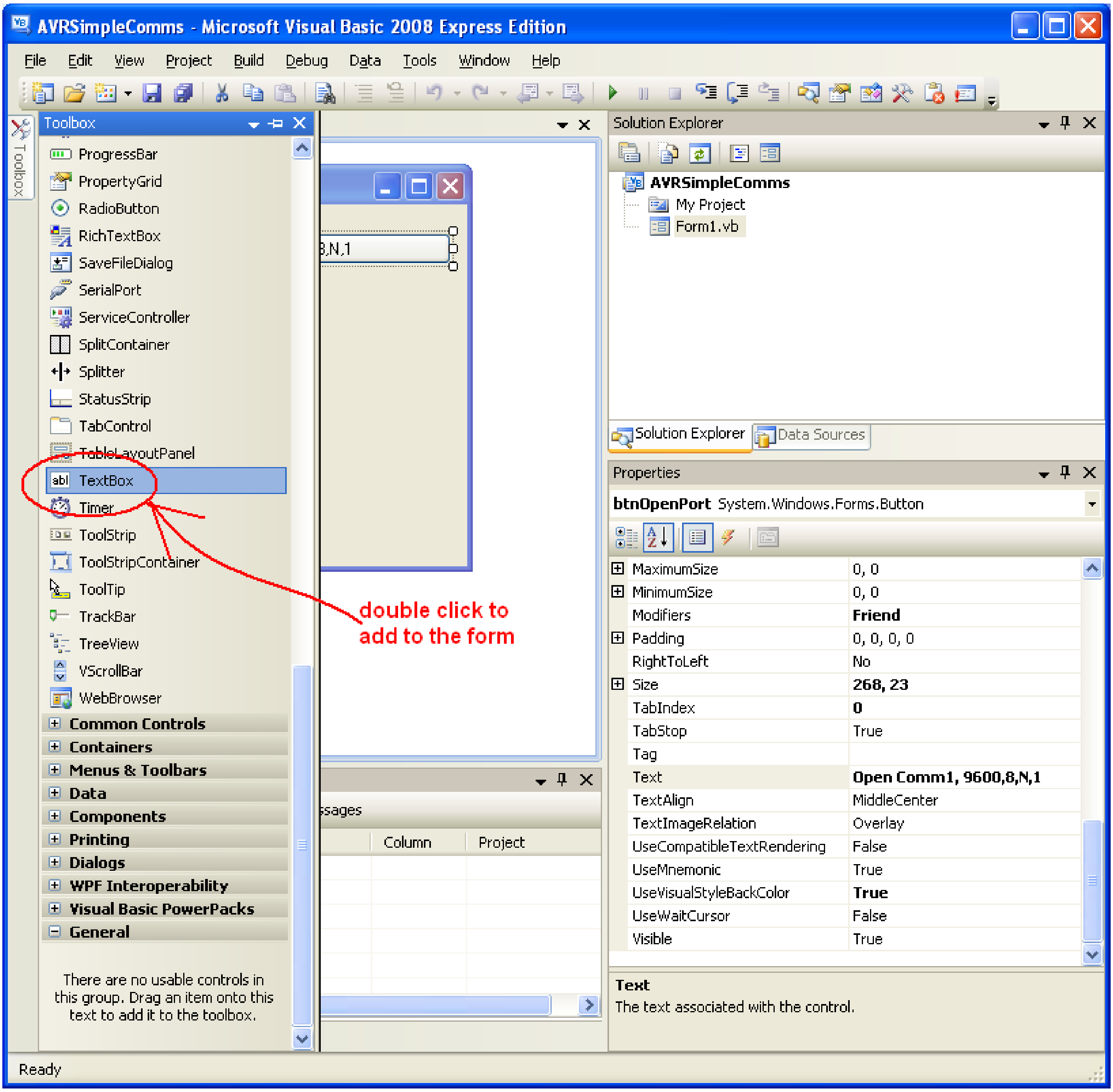
We will follow the same simple convention for naming every control, the first three letters tell us what type of control it is **btn** for button, this is always in lower case.

The next part of the name tell us a short description of what it is used for **OpenPort**, we use uppercase letters to separate the words not spaces.

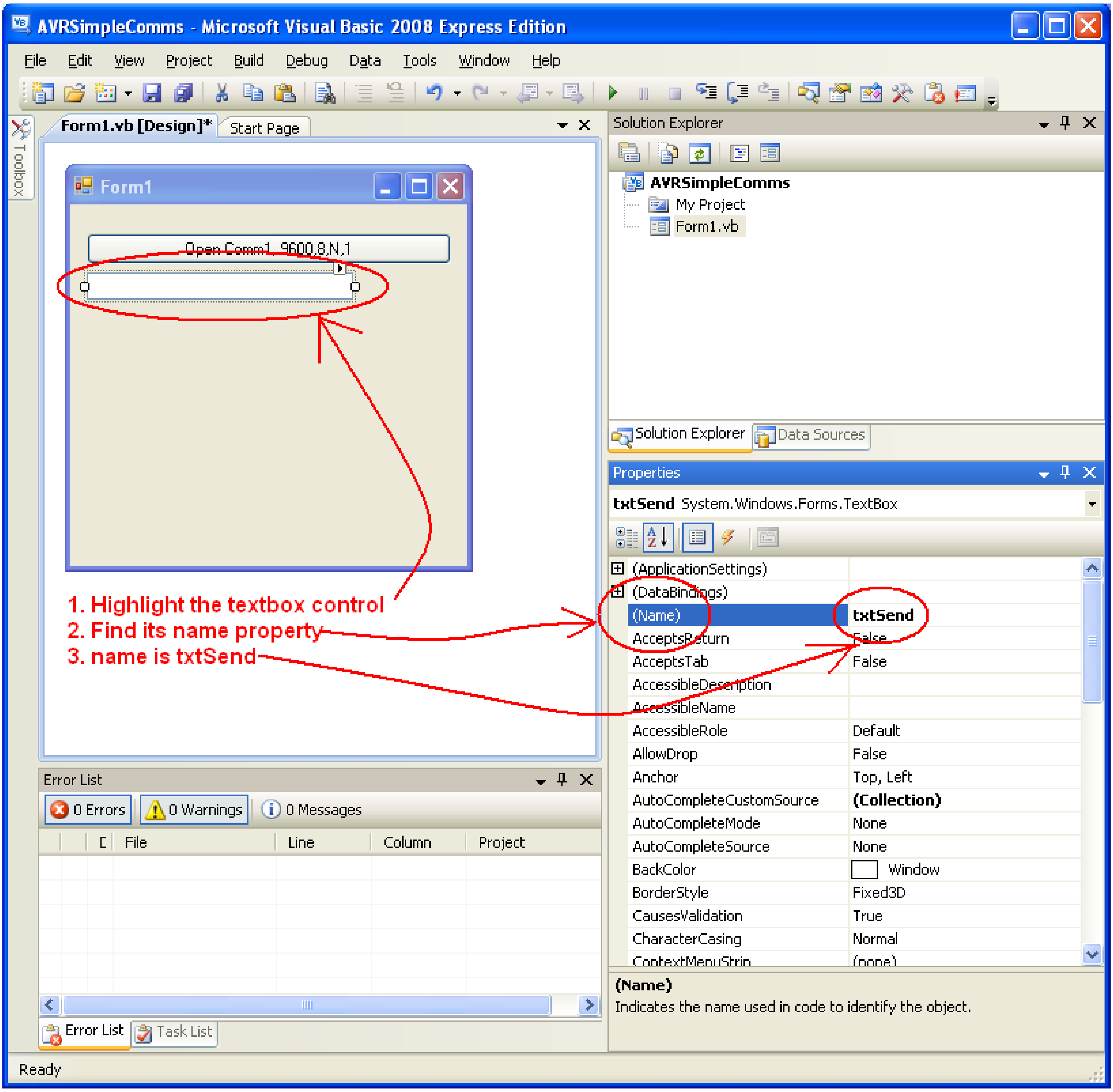
Remember the whole name **btnOpenPort** has no spaces in it, starts with lowercase 3 letters to tell us what sortof control it is.



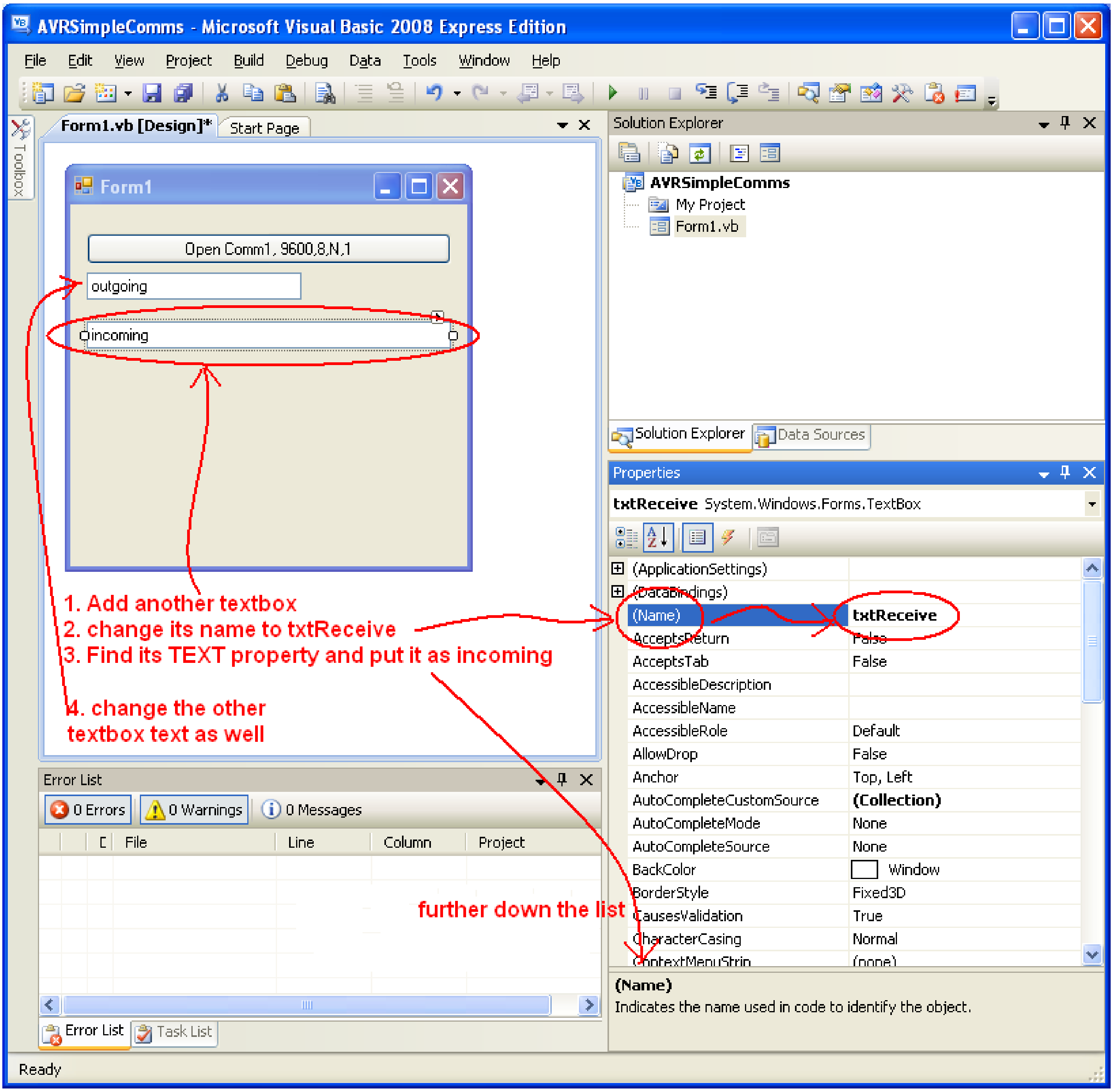
The button **btnOpenPort** has another property its **Text** property. Find this and type in **Open Comm1, 9600,8,N,1** – spaces are fine in this.
You can experiment with other properties like colors and fonts as well.



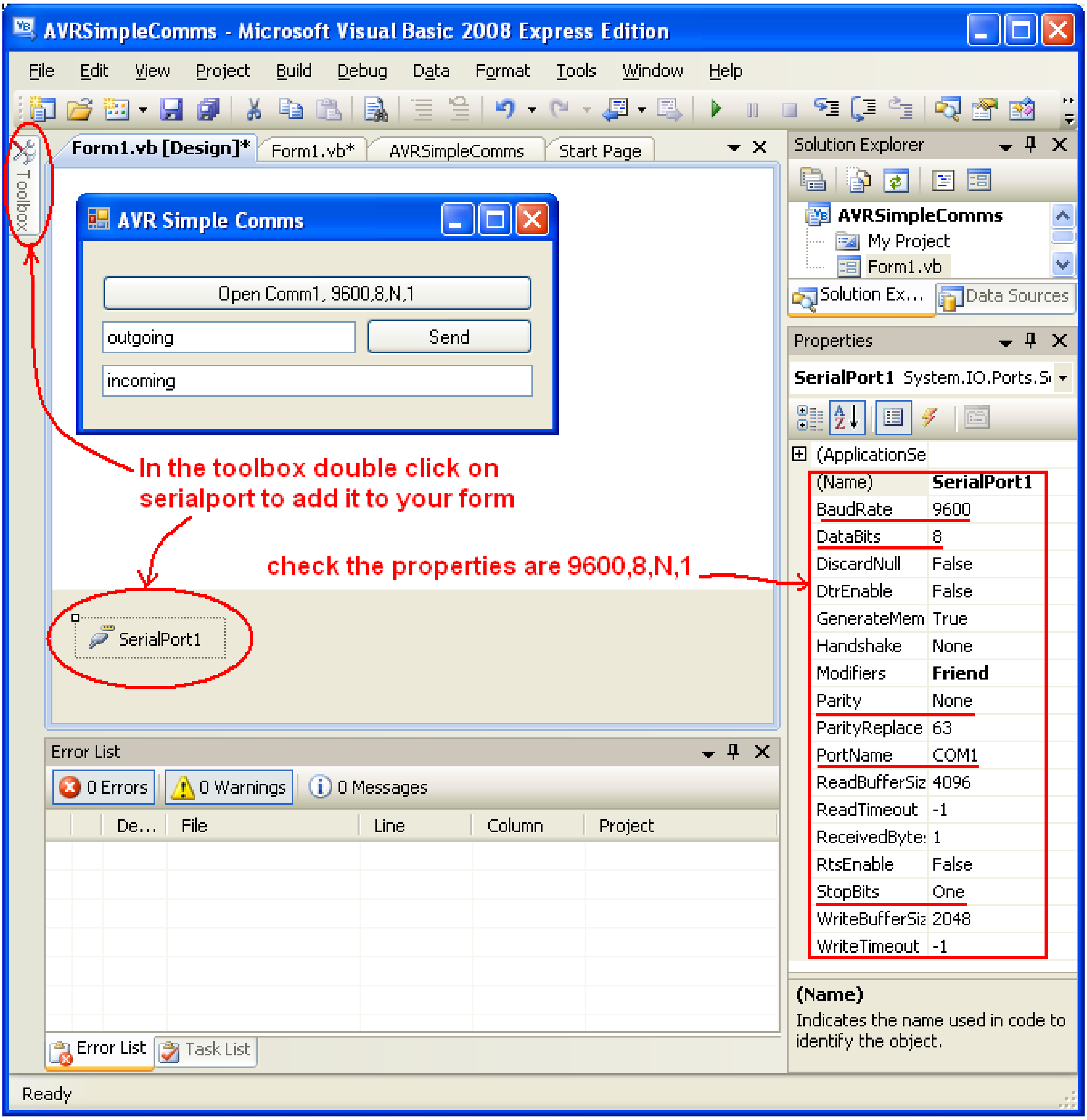
Add another control, a **TextBox** control.



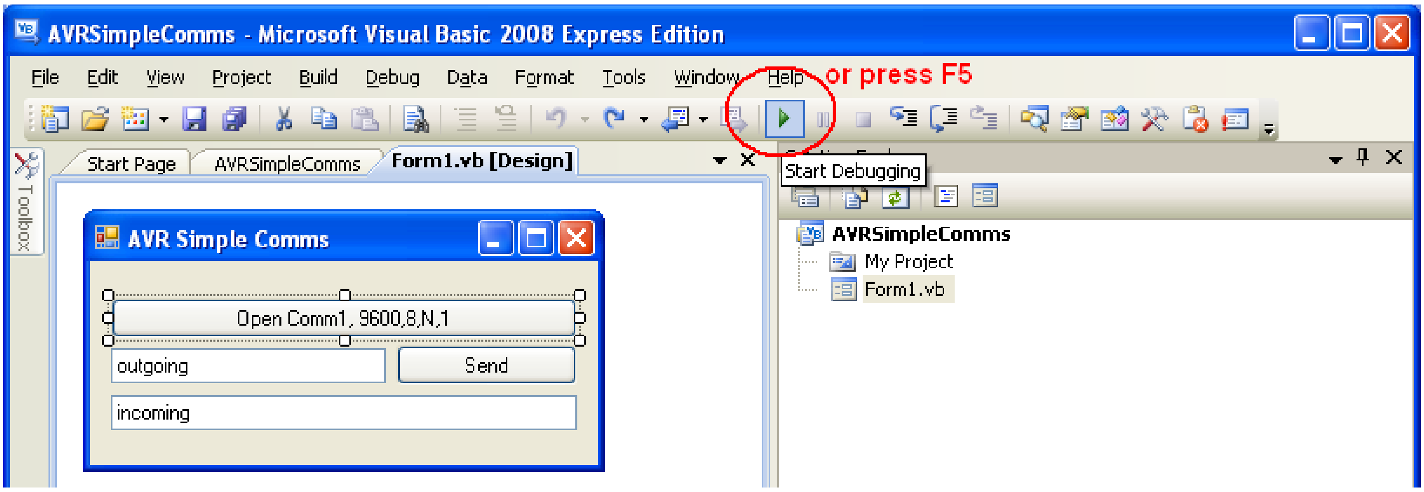
Change the Name property to **txtSend**, txt tells us it is a **TextBox control** and Send is its purpose, text to send! We follow the naming convention 3 lower case letters for the type, capital letters for the following words in the name and NO SPACES!



Add a second TextBox control and change its name size, position and text.



The last control to add is a hidden one (the user cannot see it). It is a SerialPort contol.
We wont bother to change its name from SerialPort1 as we only need one of these for the whole program. But do check its properties are correct.



The GUI is finished!!! But the program isn't.
You can run your program (in debug mode) by pressing F5 or the green play button.

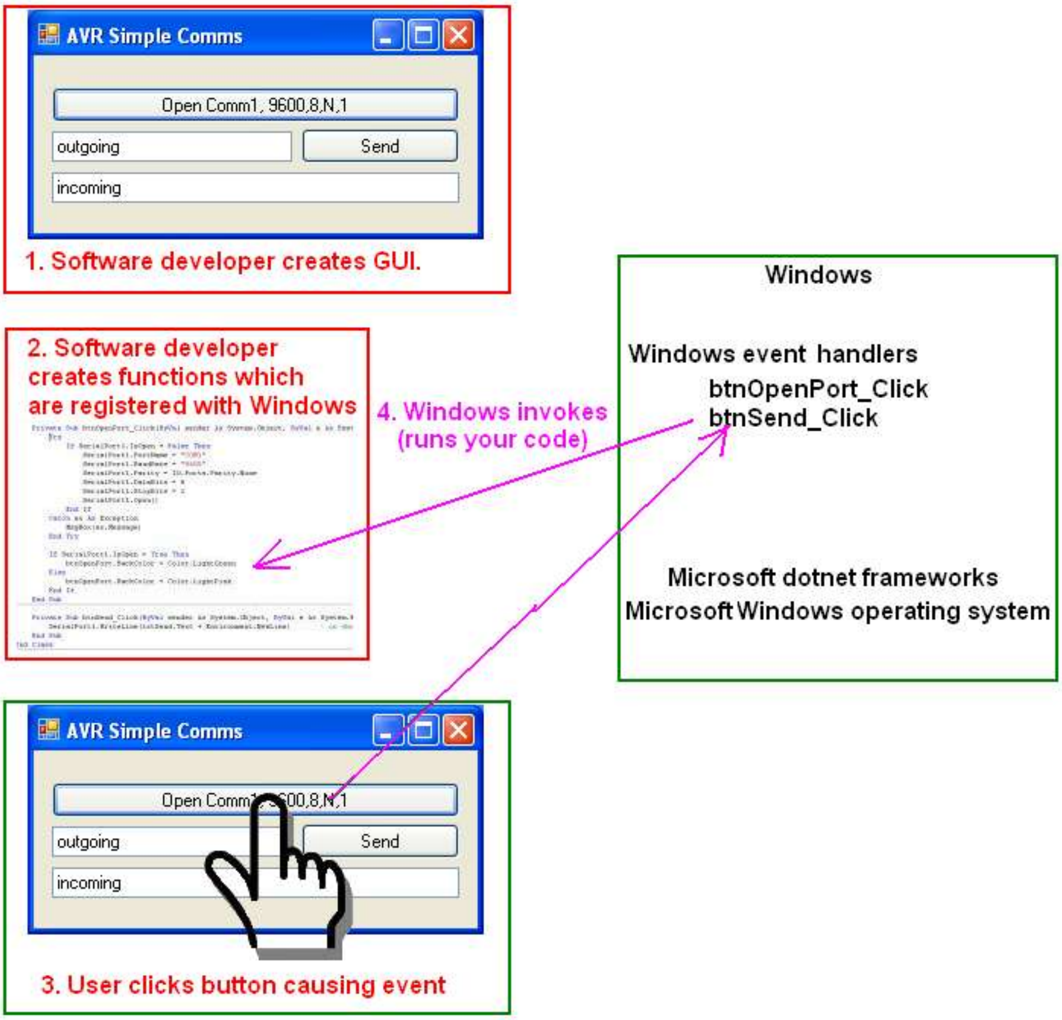
Your program will run, you can select buttons and type in text but nothing will happen yet as you have not written any code.

40.15 Stage 2 – Coding and understanding event programming

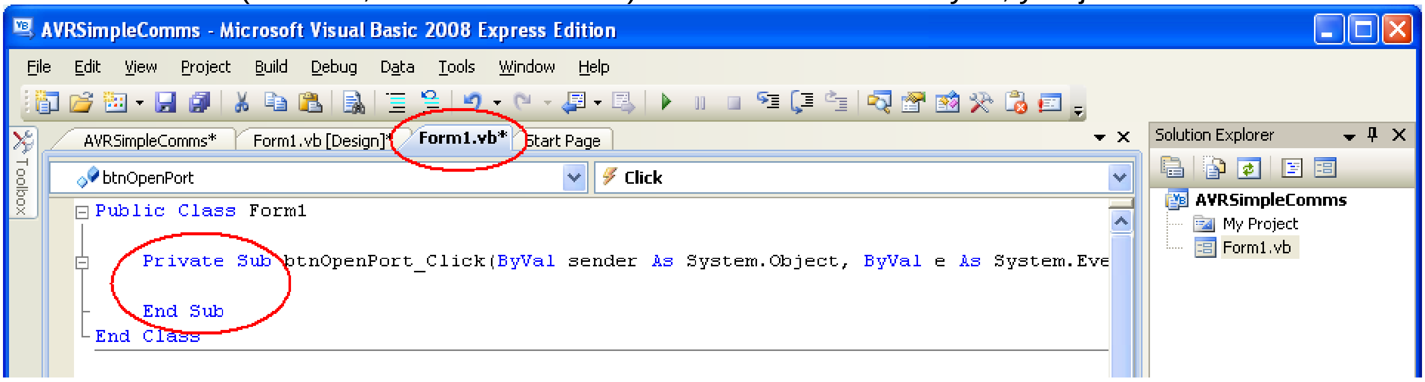
Programs in windows are not sequential as they are in BASCOM, they are **event driven**. This means that you write a whole bunch of what looks like disconnected functions (subroutines) without any overriding control structure.

Its just that windows handles all the calling of these routines.

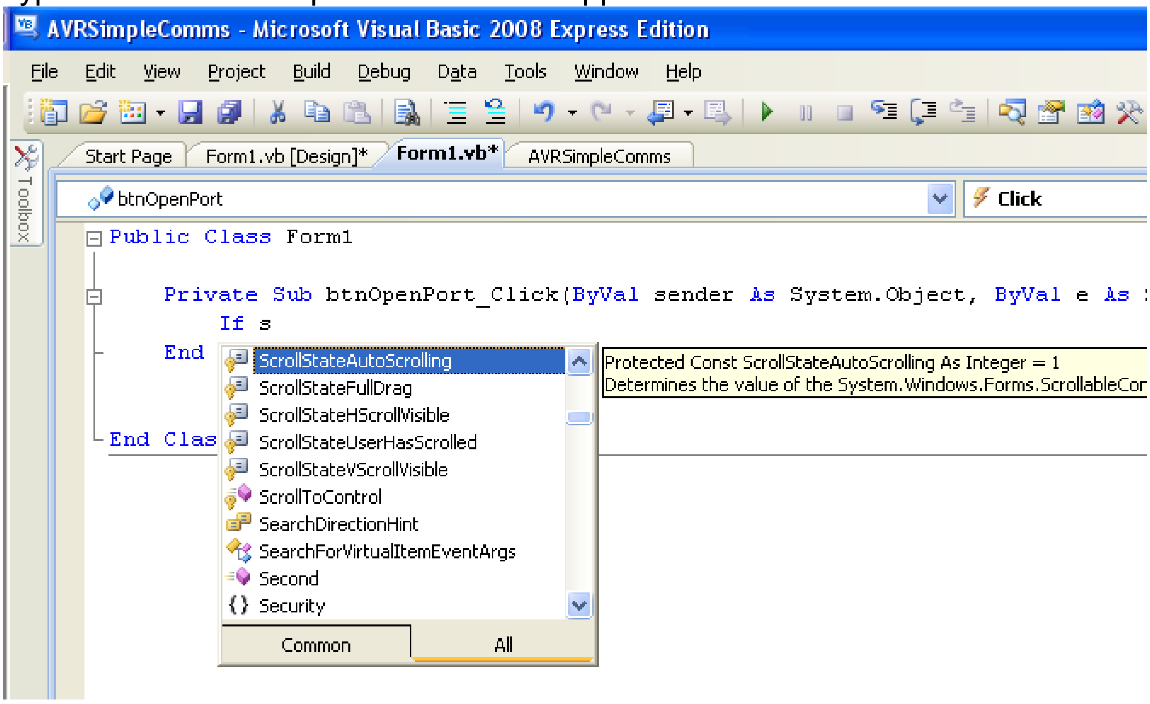
This means that nothing happens in your program until the user interacts with it. This is called an **event**. An event might be a mouse click on a Button or the user changing text in a TextBox.



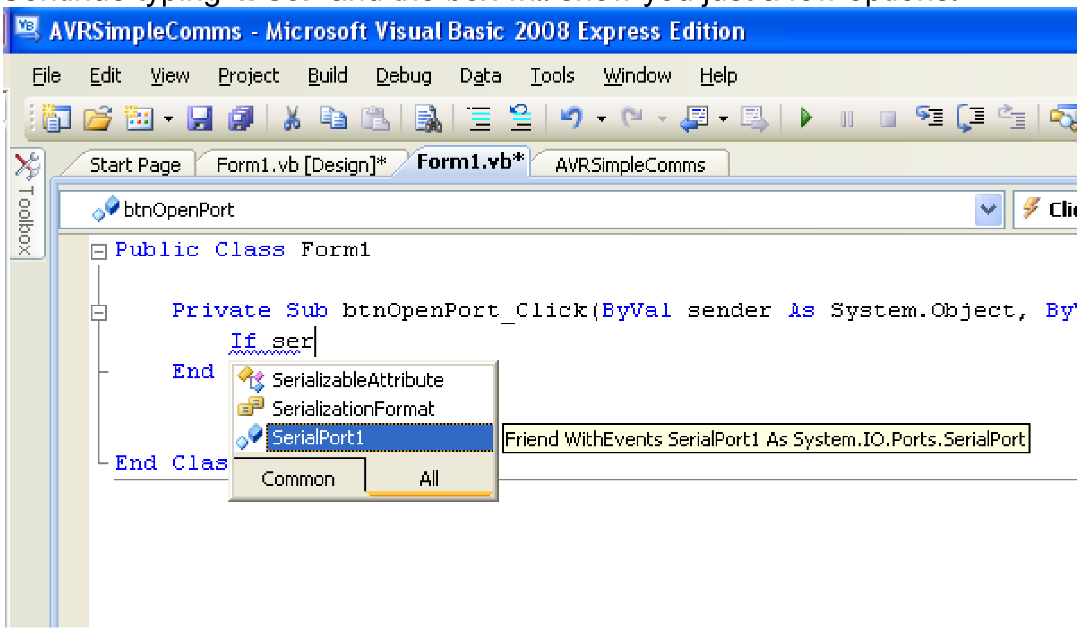
To add code to your program double click on the Open Port button in the designer and this new window will appear.
Note the title of of it. Form1.vb
Also the method (function, sub or subroutine) has been started for you, you just add code within it.



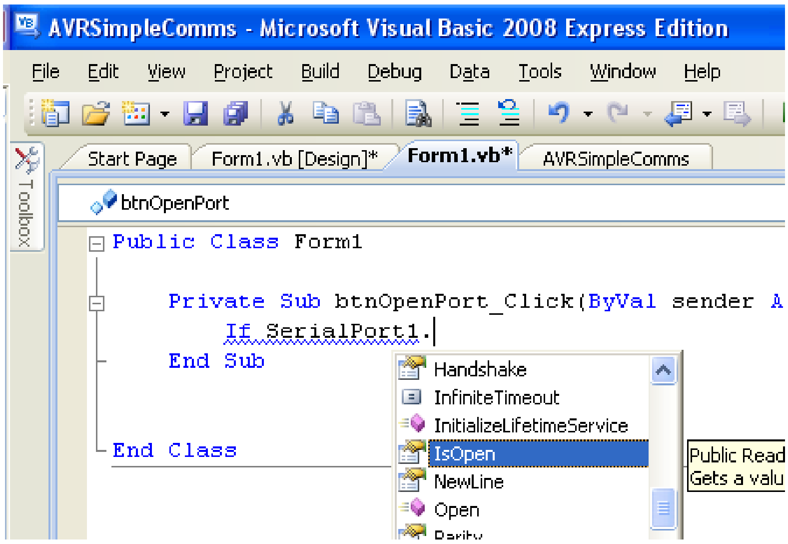
Visual studio is very helpful with the next steps as well, as it has a fantatstic autocomplete feature.
Type ‘**if s**’ and the drop down menu will appear.



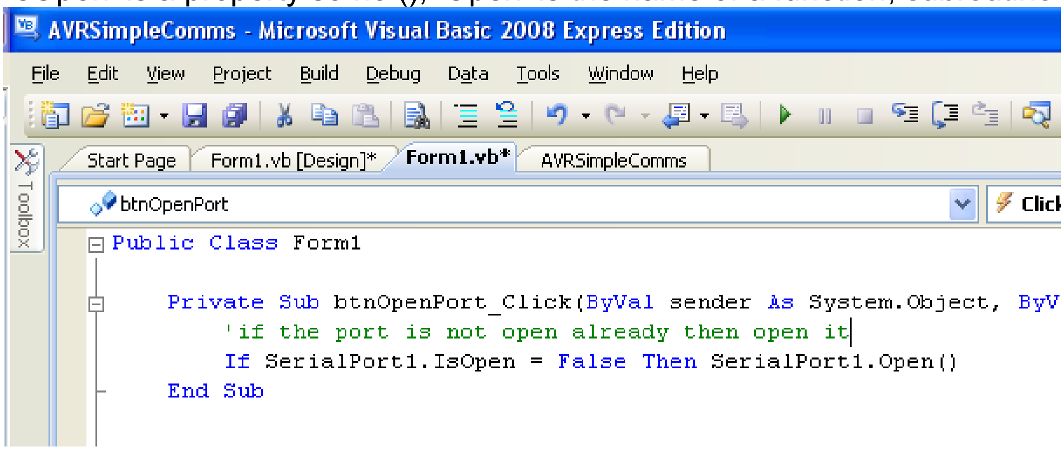
Continue typing ‘**if ser**’ and the box will show you just a few options.



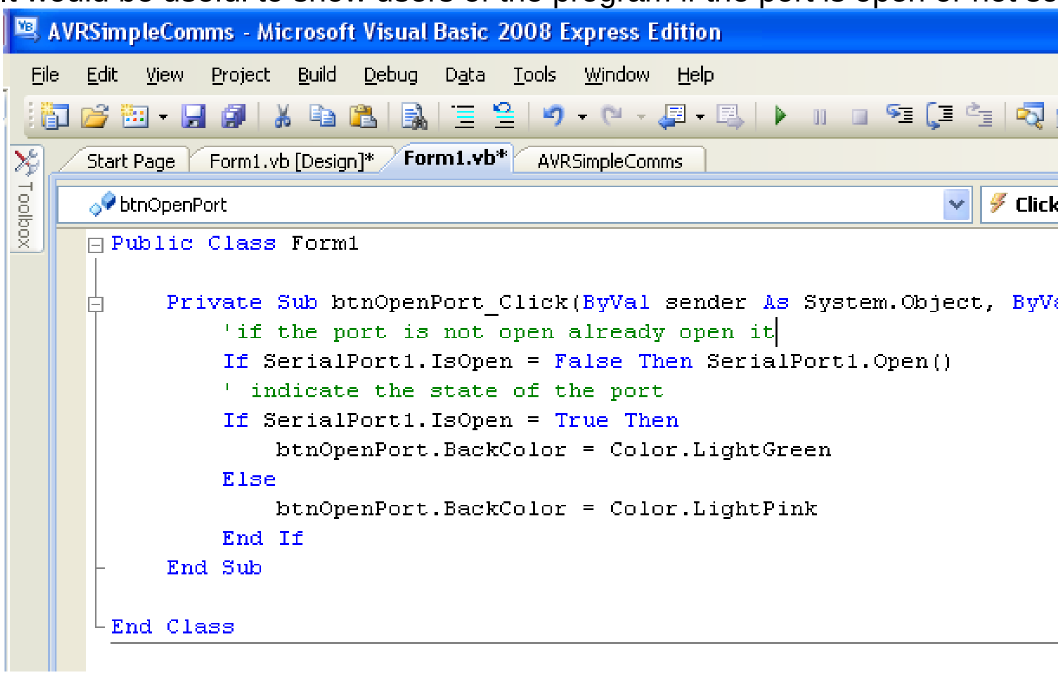
Click on ‘**SerialPort1**’ and then press the fullstop ‘.’ This will give you the different properties you can access for the serialport, choose ‘**IsOpen**’.



Finish typing the full line of text and the comment above it.
MAKE SURE YOU PUT () at the end of the line.
‘IsOpen’ is a property so no (), ‘Open’ is the name of a function, subroutine or method so it has ().

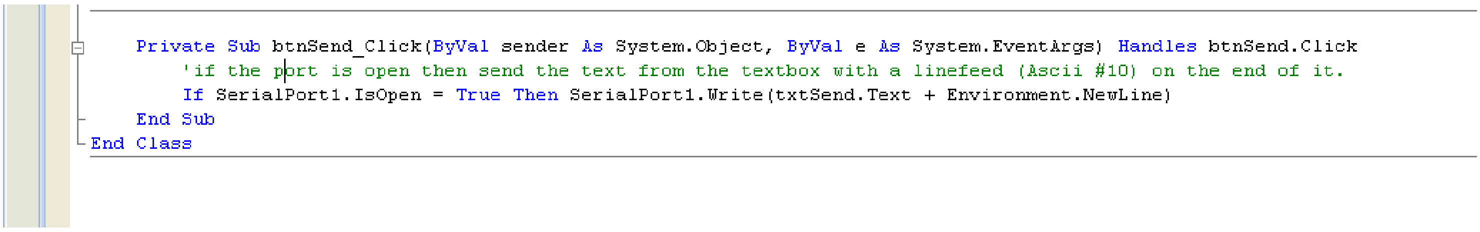


It would be useful to show users of the program if the port is open or not so add some more code.



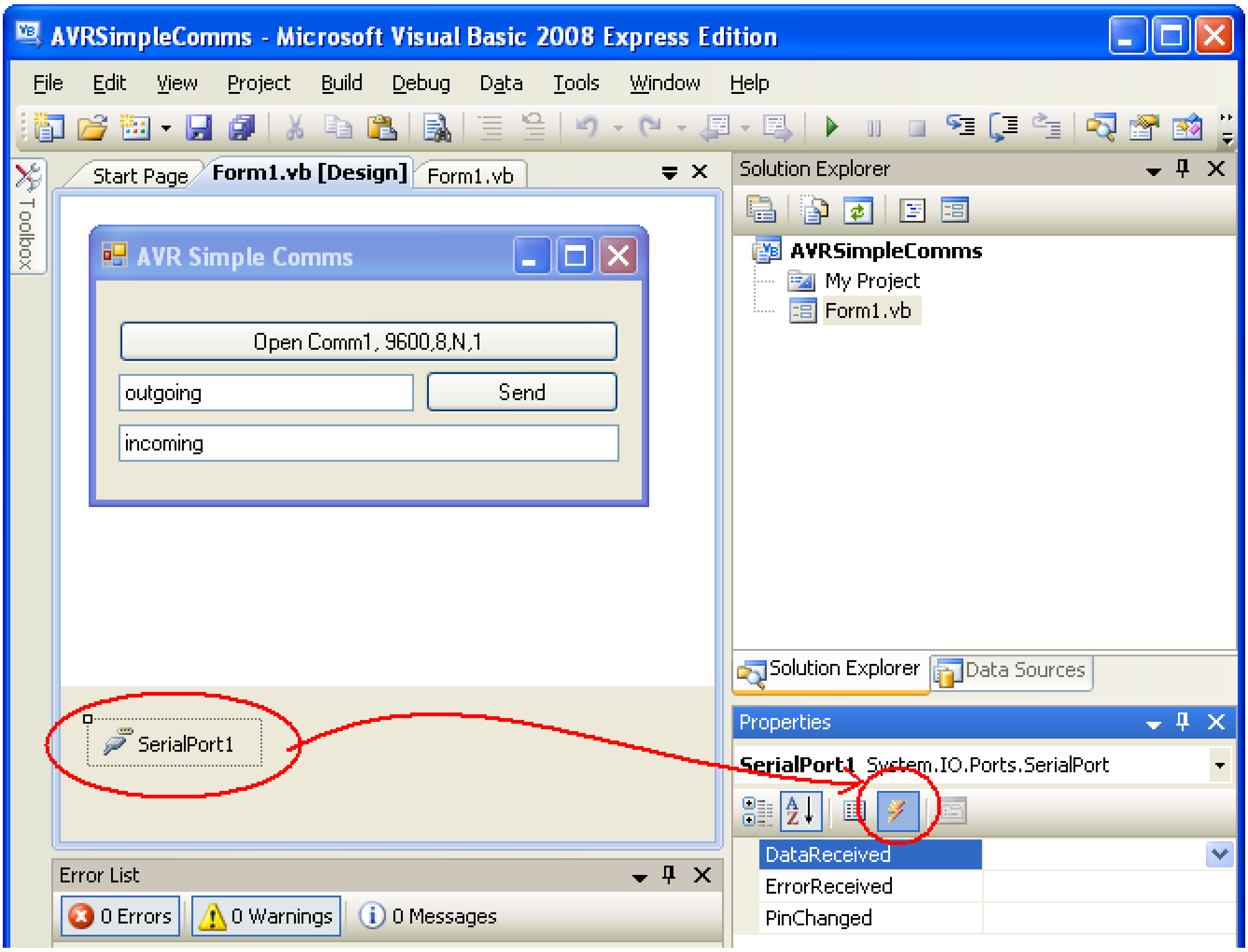
You can run this program now, and if your computer has a Com1 then it should work (if not, it will crash).

Double click on the other button in the GUI the SEND button, then add the following code. Use Visual Studio's autocomplete to help you enter the code.

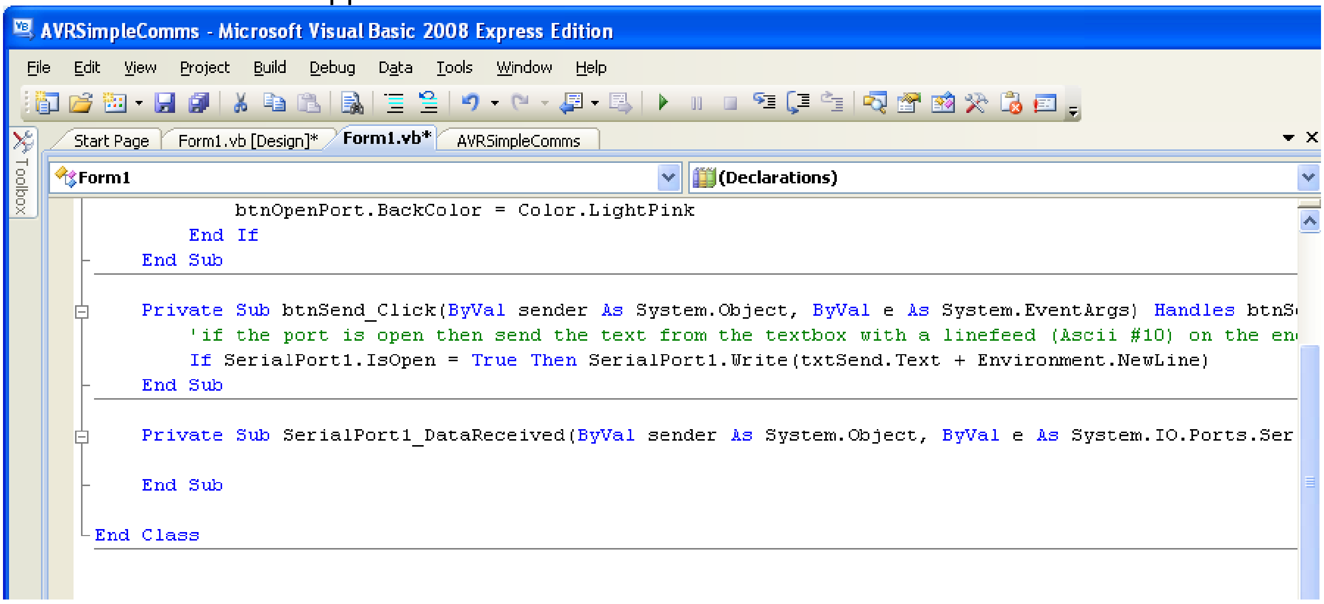



```
'if the port is open then send the text from the textbox with a linefeed (Ascii #10) on the end of it.
If SerialPort1.IsOpen = True Then SerialPort1.Write(txtSend.Text + Environment.NewLine)
```

The next step is to add code that will allow your program to display incoming text.
This is more tricky. Click on the SerialPort1 control and then on the lightning symbol, this will then list all the available events for the control. Double click on the **DataReceived** event.



The code window will appear



Then enter the code below. This code is very complex to understand, but it is necessary because of how Windows multitasks everything. We will not try to understand how it works, just why it is required and a bit about what it is doing.

Our program must monitor the serialport as well as our Form at the same time because data could come in while someone was typing text into a textbox. To do this windows creates two threads (parallel running tasks which are part of the same program) one to monitor the serialport and one for our form. When we want to pass something from the serial port to the form it must go from one thread to another, to do this the code below is required.

```
    Private Sub SerialPort1_DataReceived(ByVal sender As System.Object, ByVal e As System.I
        Me.Invoke(New MyDelegate(AddressOf processtextin), SerialPort1.ReadLine())
    End Sub

    Public Delegate Sub MyDelegate(ByVal textin As String)

    Private Sub processtextin(ByVal textin As String)
        txtReceive.Text = textin
    End Sub
End Class
```

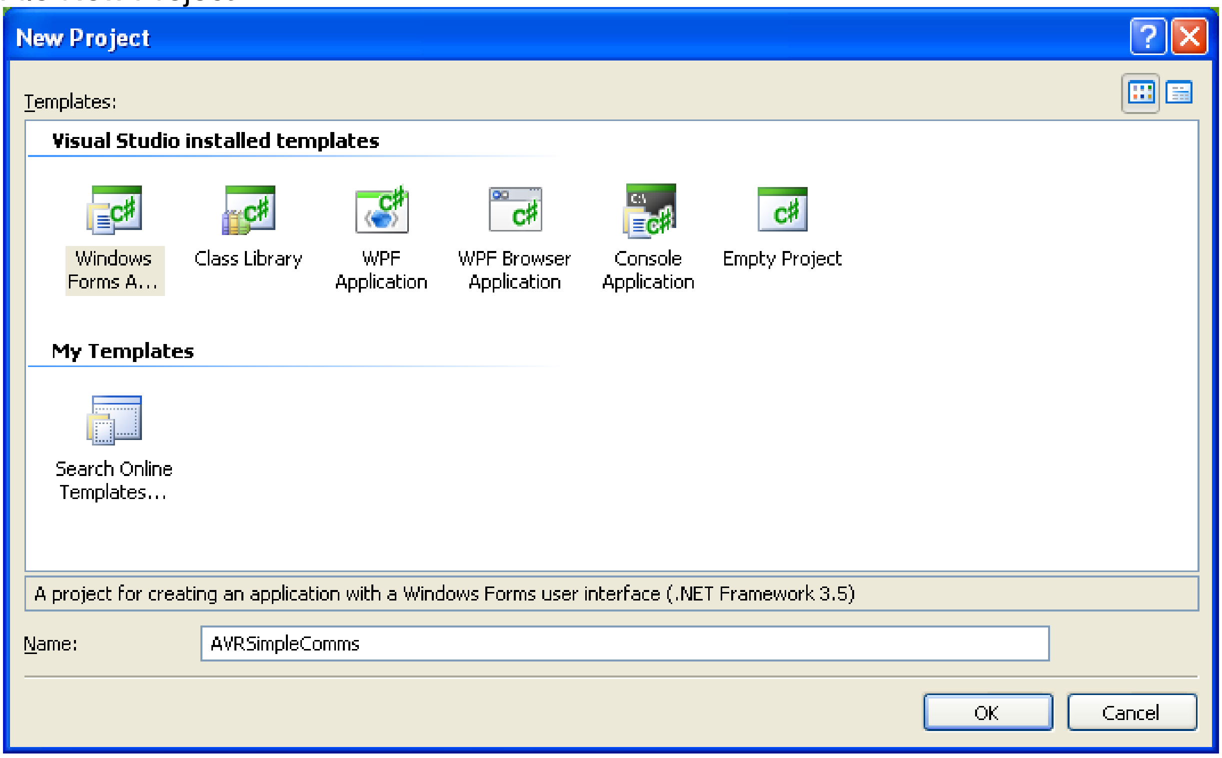
The program now works!
This is a very short introduction to Visual Basic, we will go on to develop a larger program as well, but if you are interested in learning more get a book out of the library or jump on the wen and learn more.

Having created this program in Visual Basic, we can also create it in …

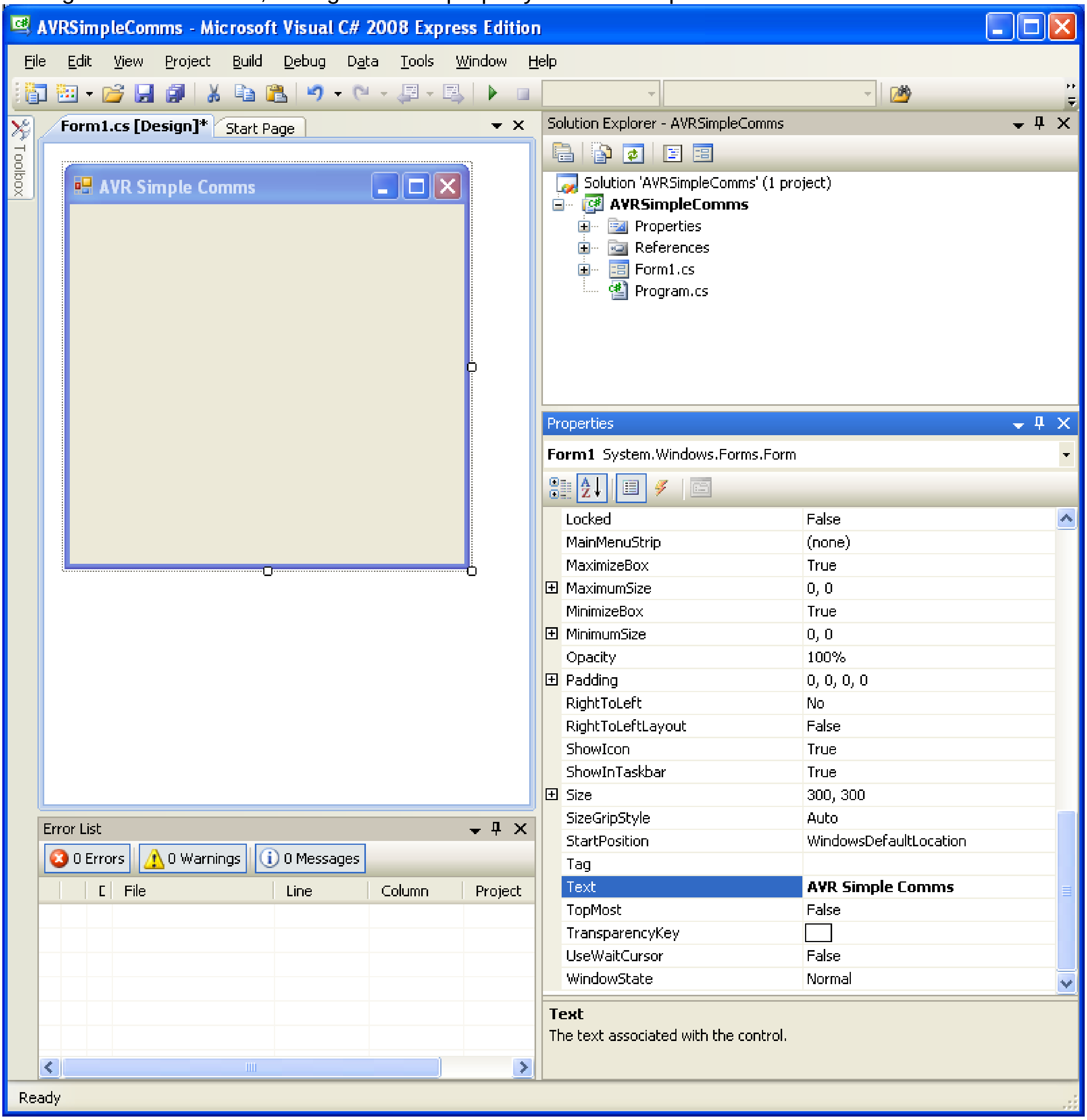
40.16 Microsoft Visual C# commport application

Here is the same application developed in Visual C# 2008 Express Edition

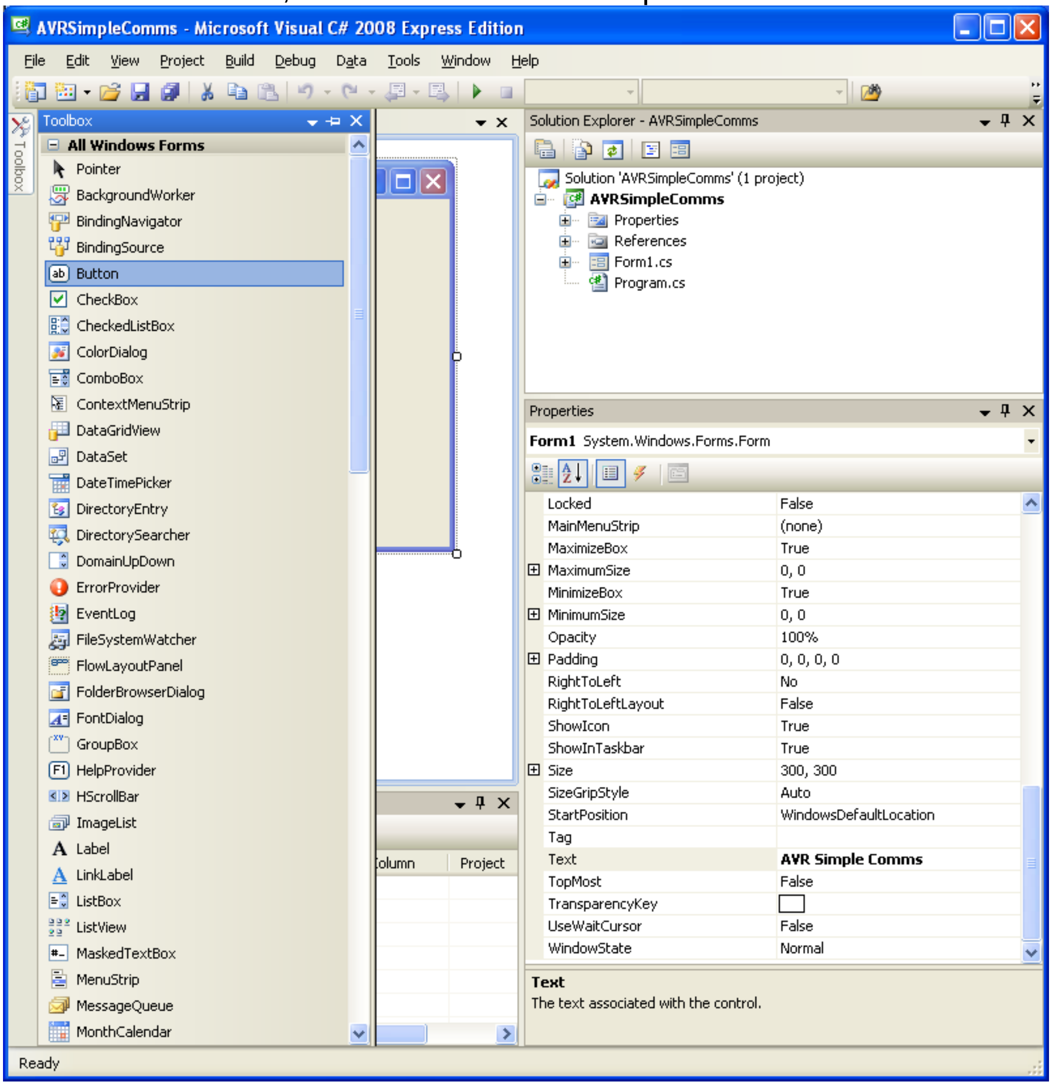
File-New Project



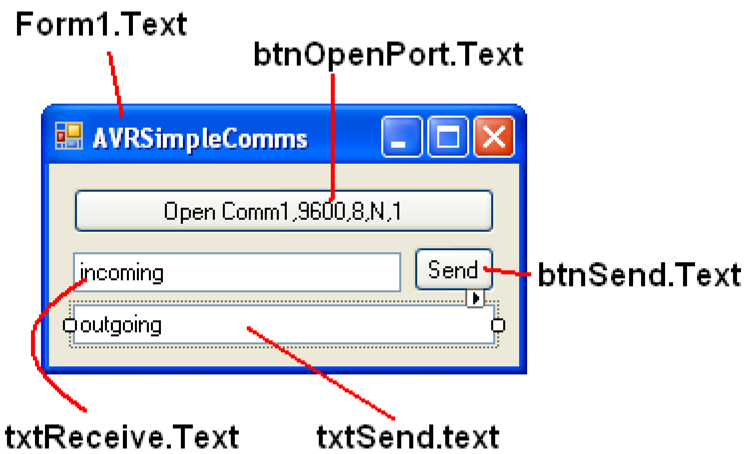
Having created the form, change its Text property to AVR Simple Comms



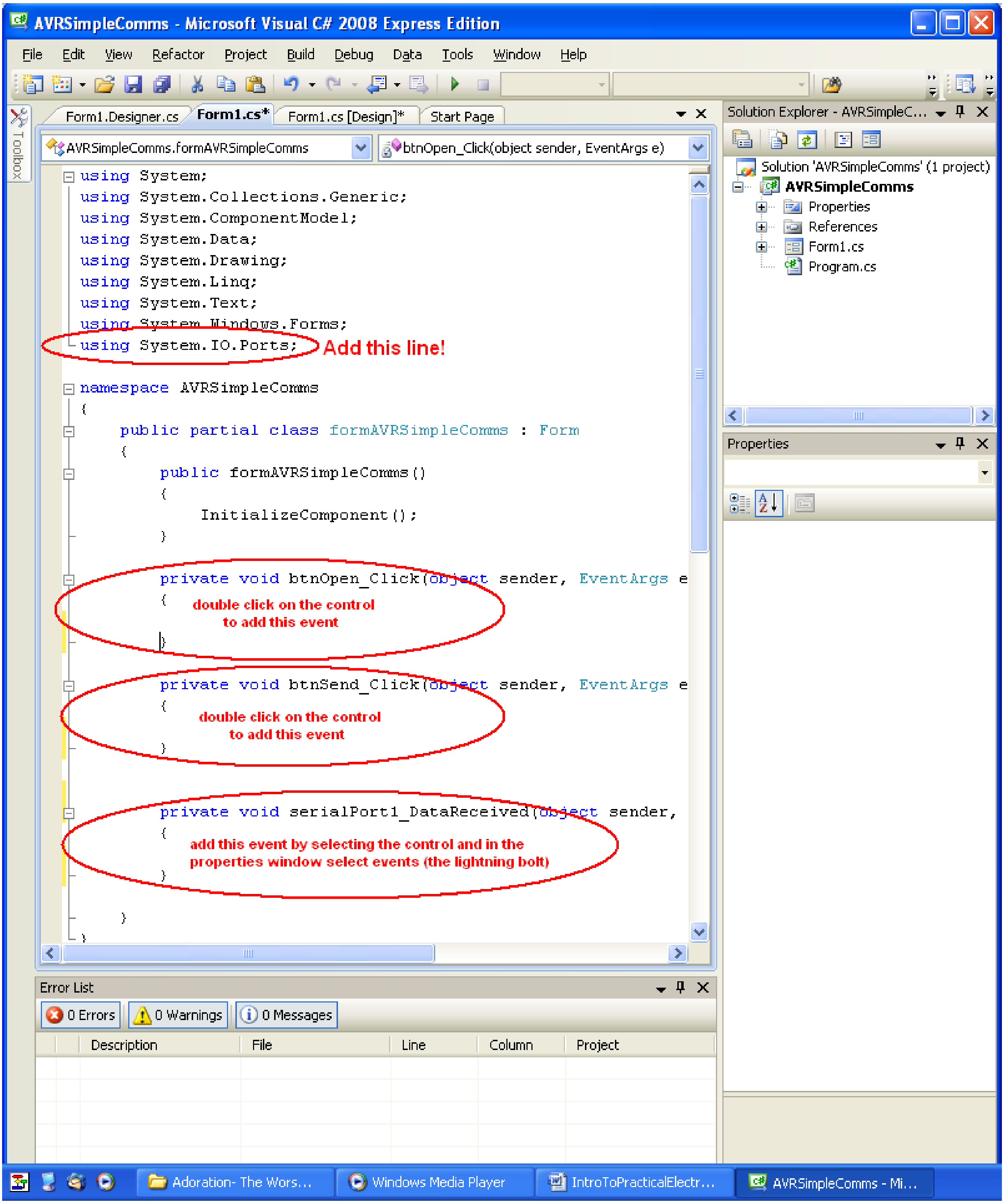
Add the two buttons, two textboxes and serialport



Call the buttons btnOpenPort and btnSend, change their text properties.
Call the textboxes txtSend and txtReceive and change their text properties as well.



C# Events
As with VB when you double click on a control, you will then go to the code window and can add code to the control. The serialport control is different, you must select events in the properties window and add the DataReceived event.

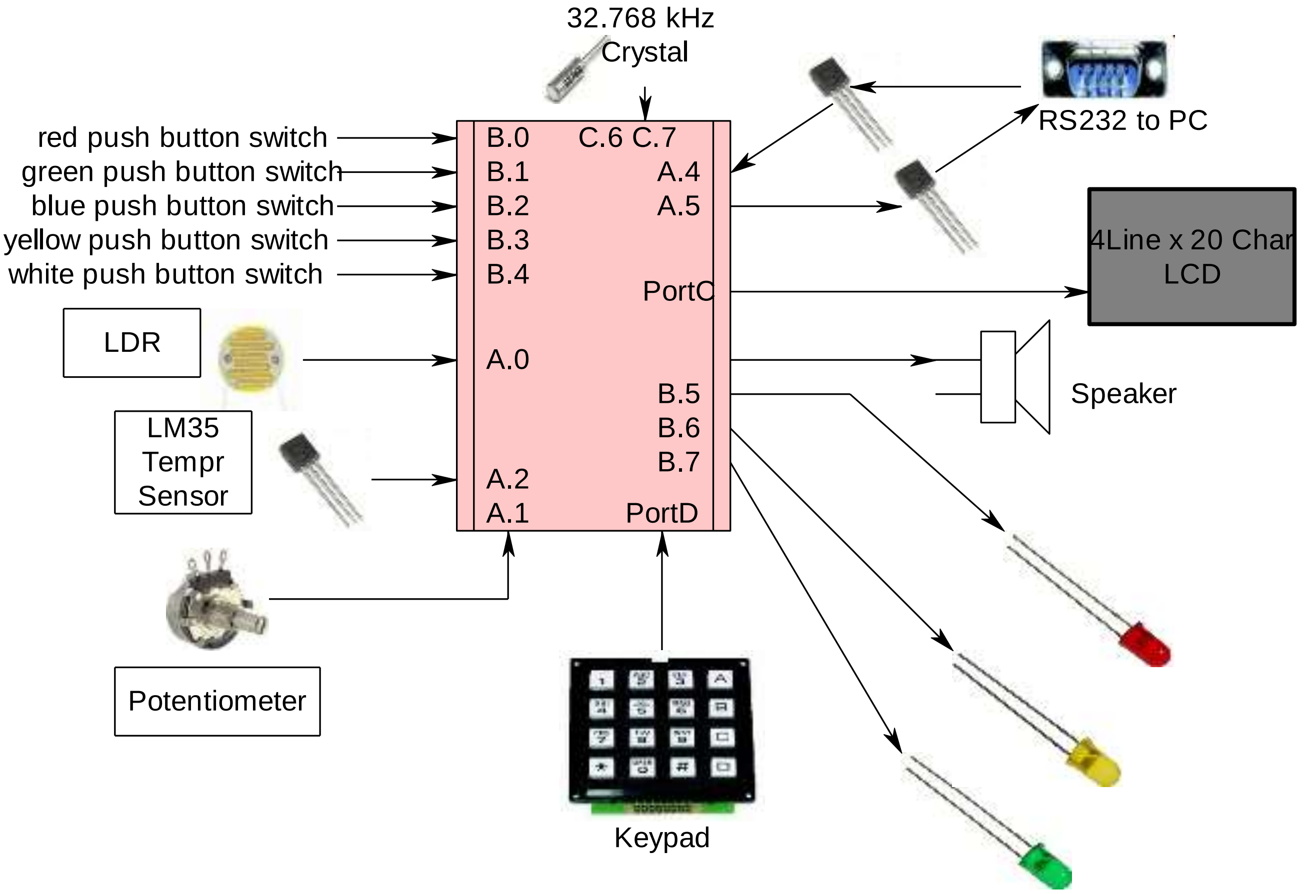


Here is all the code for this program. Add the parts in yellow <u>after</u> you have added the events (don't try and add the event code directly, it wont work if you do)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;

namespace AVRSimpleComms
{
    public partial class formAVRSimpleComms : Form
    {
        public formAVRSimpleComms()
        {
            InitializeComponent();
        }

        private void btnOpen_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen  == false)
            {
                serialPort1.Open();
            }
            if (serialPort1.IsOpen)
            {
                btnOpen.BackColor = Color.LightGreen;
            }
            else
            {
                btnOpen.BackColor = Color.LightPink;
            }
        }

        private void btnSend_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)//can only write out if the port is open
            {
                serialPort1.WriteLine(txtSend.Text + Environment.NewLine);
            }
        }

        private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            txtReceive.Invoke(new EventHandler(delegate
                {processtextin(serialPort1.ReadExisting()); }));
        }

        private void processtextin(string txtstring)
        {
            txtstring = txtstring.Replace('\n', ' ');       //remove newline
            txtstring = txtstring.Replace('\r', ' ');       //remove carriage return
            txtReceive.Text = txtstring;                    //display received data
        }
    }
}
```

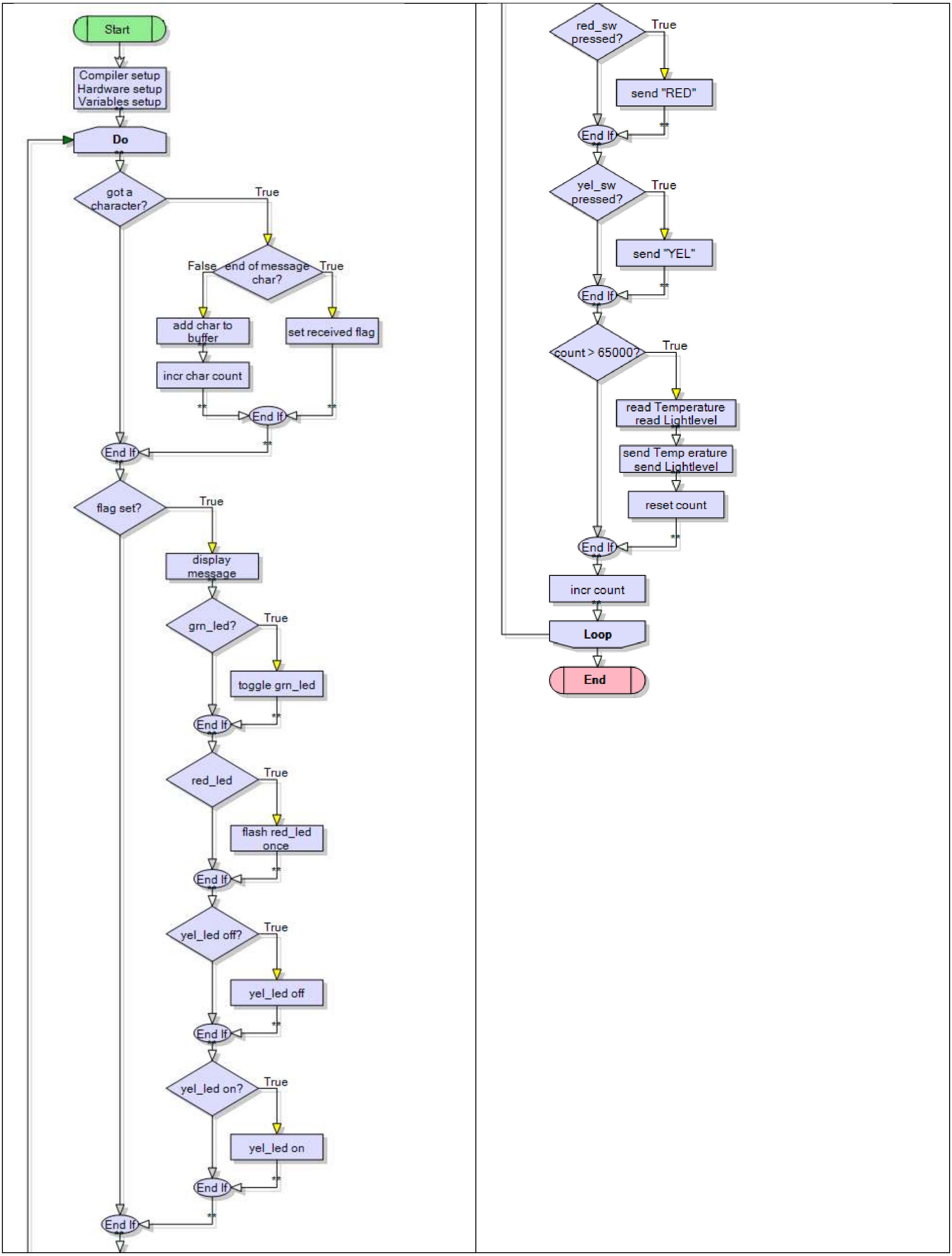
Things to note with C# (also C and C++) there is a semicolon at the end of each line

40.17 Microcontroller with serial IO.



This AVR based system monitors some input devices and outputs the data from them to the local LCD as well as via the RS232 port to a PC. It also monitors the serial input to see if there are ant messages to display on the LCD or to decode to do certain tasks.

Note that the analog inputs are not read and sent all the time just every ½ second. This is achieved through setting up a counter and counting up to 65000 before reading and sending.


Start
Compiler setup
Hardware setup
Variables setup
Do
got a character?
True
False
end of message char?
True
add char to buffer
incr char count
set received flag
End If
End If
flag set?
True
display message
grn_led?
True
toggle grn_led
End If
red_led
True
flash red_led once
End If
yel_led off?
True
yel_led off
End If
yel_led on?
True
yel_led on
End If
End If
red_sw pressed?
True
send "RED"
End If
yel_sw pressed?
True
send "YEL"
End If
count > 65000?
True
read Temperature
read Lightlevel
send Temp erature
send Lightlevel
reset count
End If
incr count
Loop
End

In this case the Bascom program monitos the LDR, LM35, and two switches.

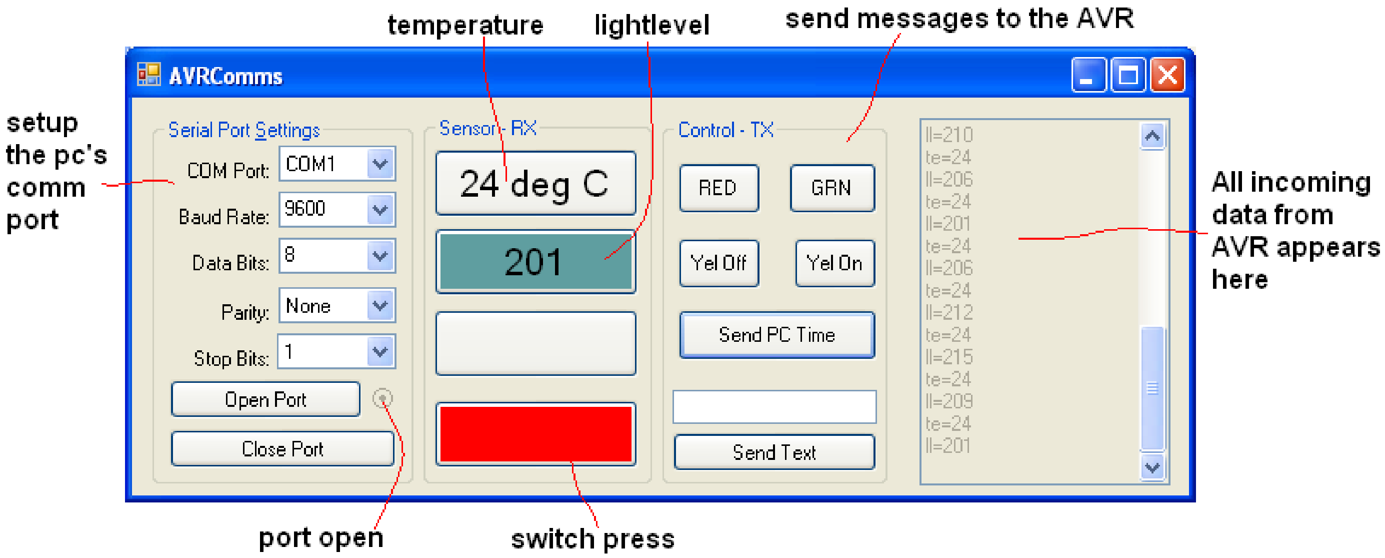
```
'------------------------------------------------------------------
' Title Block
' Author:    B.Collis
' Date:      Feb 08
' File Name: SerialioSoftUARTver2.bas
'------------------------------------------------------------------
' Program Description:
'
' Hardware Features:
'  LCD on portc
'  5 buttons on pinb.0,1,2,3,4   , red, yellow, green, blue, white
'  3 LEDs on B5,6,7 , green, yellow ,red
'  Buffer Transistors on for SW UART A.5(TXD), A.4(RXD)
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the crystal we are using
$regfile = "m8535.dat"             'the micro we are using
'------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Input               '
Config Porta.5 = Output              ' software UART TXD
Config Portb = Input
Config Portb.5 = Output              '
Config Portb.6 = Output              '
Config Portb.7 = Output              '

' LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs =
Portc.0
Config Lcd = 20 * 4                'configure lcd screen
'ADC
'know about the different references or possibly damage your chip!
Config Adc = Single , Prescaler = Auto , Reference = Avcc
Start Adc

' software UART
Open "comA.5:9600,8,n,1" For Output As #1
Open "comA.4:9600,8,n,1" For Input As #2

' hardware aliases
Red_sw Alias Pinb.0
Yel_sw Alias Pinb.1
Grn_sw Alias Pinb.2
Blu_sw Alias Pinb.3
Wht_sw Alias Pinb.4
Set Portb.0                    'enable pullup resistors
Set Portb.1
Set Portb.2
Set Portb.3
Set Portb.4
Grn_led Alias Portb.5
```

The internal UART is on D.0 and D.1, which are in use for the keypad.
A software UART using A.4 as input and A.5 for output is configured .

```
Yel_led Alias Portb.6
Red_led Alias Portb.7

' ADC Constants
Const Pot = 0                      'getadc(pot)
Const Lm35 = 1                       'getadc(lm35)
Const Ldr = 2                      'getadc(ldr)
Const False = 0
Const True = 1
'-----------------------------------------------------------------------
' Variables
Dim Tempr As Word
Dim Lightlevel As Word
Dim Potval As Word
Dim Buffer As String * 20
Dim Inputstring As String * 20
Dim Char As Byte
Dim Received As Bit                   'flag used to signal a complete receive
Received = False
Dim I As Word
I = 0
'---------------------------------------------------------------------
'constants
Const Timedelay = 1000
'-------------------------------------------------------------------
'program starts here
Cls
Cursor Off
Lcd "AVR Data Program"
Print "AVR Data Program"

Do
  'reads all the data coming in to the micro's software uart into a buffer
  ' a buffer is a portion of memory
  Char = Inkey(#2)                   'see if there is a character
  If Char > 0 Then                   'if there is
    If Char = 13 Then                'if its a Carriage return
      Nop                         'ignore it
    Elseif Char = 10 Then             'if Linefeed (signals end of message)
      Inputstring = Buffer          'copy to output
      Buffer = ""                 'release the buffer
      Received = True               'signal we have the complete string
    Else
      Buffer = Buffer + Chr(char)    'add new char to buffer
    End If
  End If

  If Received = True Then
    'display the incoming message on the LCD
    Locate 4 , 1
    Lcd Spc(20)
    Locate 4 , 1
    Lcd Inputstring
    'Print Inputstring  'echo the message back to the PC
```

```
    'process the incoming messages
    If Instr(inputstring , "grnled") > 0 Then Toggle Grn_led
    If Instr(inputstring , "redled") > 0 Then
      Set Red_led
      Waitms 50
      Reset Red_led
    End If
    If Instr(inputstring , "yelledon") > 0 Then Set Yel_led
    If Instr(inputstring , "yelledoff") > 0 Then Reset Yel_led
    Received = False                'signal we have processed the message
  End If

  'send switch press
  If Red_sw = 0 Then
    Waitms 30                       'debounce
    Do
    Loop Until Red_sw = 1
    Waitms 10                       'debounce
    Print #1 , "RED"                'send the message to the PC
  End If
  If Yel_sw = 0 Then
    Waitms 30                       'debounce
    Do
    Loop Until Red_sw = 1
    Waitms 10                       'debounce
    Print #1 , "YEL"
  End If

  'only read the analogue pins occasionally
  If I > 65000 Then
    Tempr = Getadc(lm35) / 2        'approx conversion
    Locate 2 , 1
    Lcd "temperature=" ; Tempr ; "  "
    Print #1 , "te=" ; Tempr        ' ; at end means send no linefeed
    Lightlevel = Getadc(ldr)
    Locate 3 , 1
    Lcd "lightlevel=" ; Lightlevel ; "  "
    Print #1 , "ll=" ; Lightlevel
    I = 0
  End If
  Incr I


Loop
End                                 'end program
'-----------------------------------------------------------------
' Subroutines

'-----------------------------------------------------------------
' Interrupts
```

40.18 PC software (C#) to communicate with the AVR



This program monitrs the Comm port and allows the user to send messages (including the PCs time) to the AVR.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;          //added this to use serialport

namespace AVRComms
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //here are the default values, 0 means the first in the collection
            cmbPortName.SelectedIndex = 0;   //com1
            cmbBaudRate.SelectedIndex = 5;   //9600
            cmbDataBits.SelectedIndex = 1;   //8
            cmbParity.SelectedIndex = 0;     //none
            cmbStopBits.SelectedIndex = 0;   //1
        }
```

```
private void btnOpenPort_Click(object sender, EventArgs e)
{
    if (serialPort1.IsOpen==false)
    {
        // Setup the port as per the combo box settings
        serialPort1.PortName = cmbPortName.Text;
        serialPort1.BaudRate = int.Parse(cmbBaudRate.Text);
        serialPort1.DataBits = int.Parse(cmbDataBits.Text);
        serialPort1.StopBits = (StopBits)Enum.Parse(typeof(StopBits),
                    cmbStopBits.Text);
        serialPort1.Parity = (Parity)Enum.Parse(typeof(Parity), cmbParity.Text);

        // try to open the port,
        try
        {
            serialPort1.Open();
        }
        catch (Exception ex)
                //if it cannot be opened then
        {
            MessageBox.Show(ex.ToString());
                //show us the exception that occurred
        }
            //if it is open then show the dot in the radio button
        if (serialPort1.IsOpen) radPortOpen.Checked = true;
    }
}


private void btnClosePort_Click(object sender, EventArgs e)
{
    //if the port is open close it
    if (serialPort1.IsOpen)
    {
        serialPort1.Close();
        //if it closed ok then remove dot from radiobutton
        if(serialPort1.IsOpen == false ) radPortOpen.Checked = false;
    }
}


private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    txtDataRx.Invoke(new EventHandler(delegate
    {
        processtextin(serialPort1.ReadExisting());
    }));
}
```

```
private void processtextin(string txtstring)
        {
            txtDataRx.AppendText(txtstring);                    //display received data
            txtstring = txtstring.Replace('\n', ' ');           //remove newline character
            txtstring = txtstring.Replace('\r', ' ');           //remove carriage return
            txtstring = txtstring.Trim();                       //remove spaces
            if (txtstring.Contains("te="))                      //temperature reading
            {
                txtstring = txtstring.Replace("te=", "");   //get rid of the code
                btnTempr.Text =txtstring + " deg C";        //add some text to the end

            }
            if (txtstring.Contains("ll="))                      //lightlevel
            {
                txtstring = txtstring.Replace("ll=", "");   //get rid of the code
                try
                {
                    int lightlevel = Convert.ToInt32(txtstring);
                    if (lightlevel < 20) {btnLDR .BackColor = Color.Sienna ;}
                    if (lightlevel > 100) {btnLDR .BackColor = Color.Blue; }
                    if (lightlevel > 200) { btnLDR.BackColor = Color.CadetBlue; }
                    if (lightlevel > 400) {btnLDR .BackColor = Color.DarkOrange ;}
                    if (lightlevel > 500) {btnLDR .BackColor = Color.DarkRed ;}
                    btnLDR.Text = lightlevel.ToString();
                }
                catch { }
            }
            if (txtstring.Contains("pv="))
            {
                //txtstring = txtstring.Replace("pv=", "");   //get rid of the code
                btnPot.Text = txtstring;
            }
            if (txtstring.Contains("RED")) { btn_Color.BackColor = Color.Red; }
            if (txtstring.Contains("YEL"))
            {
                btn_Color.BackColor = Color.Yellow;

            }
        }

        private void btnSendText_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write(txtDataTx.Text + "\r" + "\n");
                  //must send at least LF to remote so its know end of message
            }
            else {MessageBox.Show (" port not open");}
        }

        private void btnGrnLed_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write("grnled" + "\r" + "\n");
                  //must send at least LF to remote so its know end of message
            }
            else { MessageBox.Show(" port not open"); }
        }

        private void btnRedLed_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write("redled" + "\r" + "\n");
                  //must send at least LF to remote so its know end of message
```

```
            }
            else { MessageBox.Show(" port not open"); }
        }

        private void btnYelOff_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write("yelledoff" + "\r" + "\n");
                  //must send at least LF to remote so its know end of message
            }
            else { MessageBox.Show(" port not open"); }
        }

        private void btnYelOn_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write("yelledon" + "\r" + "\n");
                  //must send at least LF to remote so its know end of message
            }
            else { MessageBox.Show(" port not open"); }
        }
        private void btnSendTime_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Write(DateTime.Now.ToString("hh.mm.ss dd/MM/yyyy")  + "\n");
            }
            else { MessageBox.Show(" port not open"); }
        }
    }
}
```

40.19 Using excel to capture serial data

It is straightforward to use excel to look at data sent from the microcontroller. First download PLX-DAQ from the net and follow the setup instructions.

Next write a program that sends the right commands out the comm. Port to PLX-DAQ.

```
$crystal = 8000000
$regfile = "m8535.dat"
Open "comA.5:9600,8,n,1" For Output As #1

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 =
Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4
'      Declare Variables
Dim D As Single
Dim R As Single
Dim Sin_x As Single
Dim H As Byte
'-----------------------------------------------------------------------
'      Program starts here
Wait 1
'put some labels in row 1 of the speadsheet
Print #1 , "LABEL, Degrees, Radians, Sin"
'send a message to the program message area
Print #1 , "MSG, Starting data plotting "
'put a label in a specific cell on the spreadsheet (can only be in column A
thru F)
Print #1 , "CELL,SET, E2, data1_in"

Do
   Print #1 , "CLEARDATA"
   For D = 0 To 359
      'calculate the values to send
      R = Deg2rad(D)
      Sin_x = Sin(D)
      'send values that will appear in sequential columns in the spreadsheet
      Print #1 , "DATA," ; D ; "," ; R ; "," ; Sin_x
      'send data to a specific cell (can only be in columns A thru F)
      Print #1 , "CELL,SET," ; "F2," ; Sin_x
      'display the values on the lcd
      Locate 1 , 1
      Lcd D
      Locate 1 , 8
      Lcd R
      Locate 2 , 1
      Lcd Sin_x
      Waitms 10
   Next
Loop
End                                        'end program
```

```
Open "comA.5:9600,8,n,1" For Output As #1
```

This line sets up Bascom to know that you are going to send data out Porta.5, it will be at 9600 baud, 8 data bits, no paraity and 1 stop bit. It will be called #1 in the program.

```
Print #1 , "LABEL, Degrees, Radians, Sine"
```

This line sends the LABEL command out on #1 (portA.5). note that LABEL must be in capitals.
The words following LABEL will appear in excel cells, A1, A2, A3.... in that order.

```
Print #1 , "MSG, Starting data plotting "
```

send a message to be displayed in PLQ-DAX

```
Print #1 , "CELL,SET, E2, data1_in"
```

Write a label in a specific cell in excel. Note this can only be in columns A,B,C,D, or F.

```
Print #1 , "CLEARDATA"
```

Clear all data from the cells we are controlling in spreadsheet (other cells contents will not be deleted)

```
Print #1 , "DATA," ; W ; "," ; X ; "," ; Sin_x
```

Now send some data. Because there are three pieces of data they will automatically go into columns A, B & C. The first time PLX-DAQ receives this command it will put the data into A2, B2, C2. The next time it will put it into A3, B3, C3 and so on. This will create a data table.
Note that PLX-DAQ requires a comma between each piece of data.

In the code the data is sent 360 times (using the `For W = 0 To 359`)
This is the number of degrees in a circle.

The actual data are the sin values for each degree from 0 to 359, so we will get PLX-DAQ to plot the data on a graph. Note that Bascom works in radians to do sin,cos,tan not degrees so we convert it to radians with `R = Deg2rad(D)`

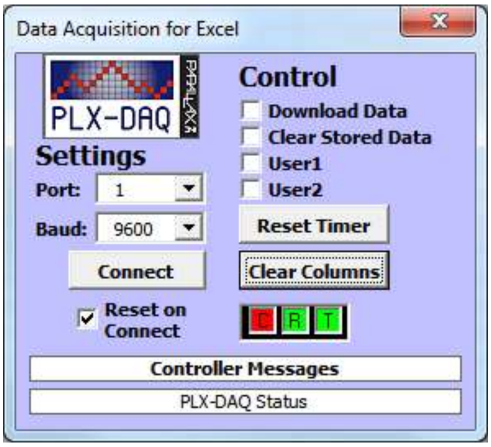
```
Print #1 , "CELL,SET," ; "F2," ; Sin_x
```

If we want just a single piece of data then we can put it into a specific cell on the table.
This can be plotted by a line/bar/dot graph that will follow the changing value.

40.20 PLX-DAQ

Download and install PLX-DQA and run it.
Excel may complain about macros and ActiveX controls, you must allow these or it will not work.

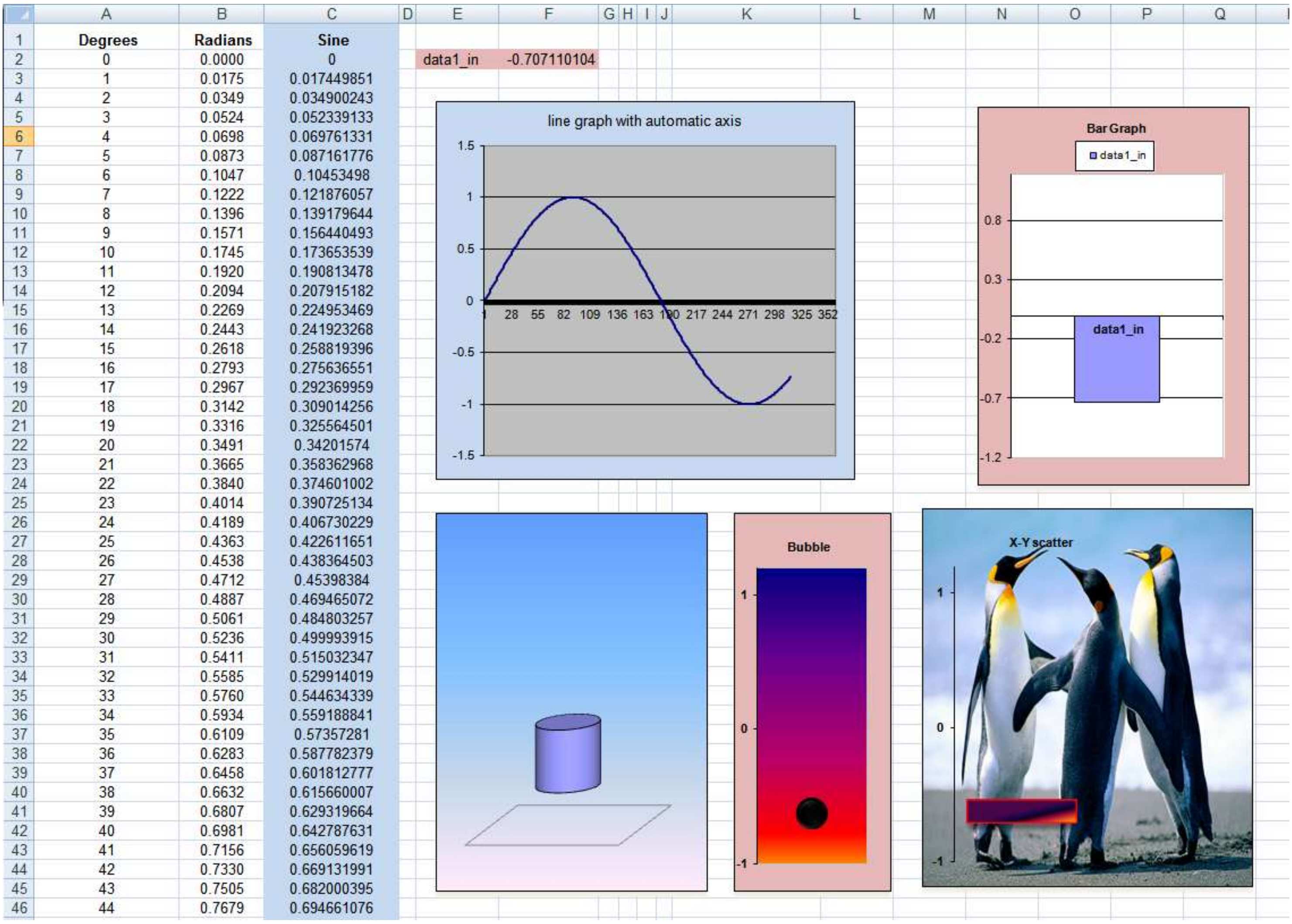
To connect to the incoming data from your microcontroller you must setup the comm port and the baud rate. You can try faster baud rates but 9600 is reliable in most instances for the AVR at 8MHZ.



The R will flash with incoming data so that you know it is all running ok.

The data coming intot excel is plotted according to the commands sent my the microcontroller.

Note that PLX-DAQ will ony respond to data in the first sheet in a multisheet spreadsheet!



	Degrees	Radians	Sine
2	0	0.0000	0
3	1	0.0175	0.017449851
4	2	0.0349	0.034900243
5	3	0.0524	0.052339133
6	4	0.0698	0.069761331
7	5	0.0873	0.087161776
8	6	0.1047	0.10453498
9	7	0.1222	0.121876057
10	8	0.1396	0.139179644
11	9	0.1571	0.156440493
12	10	0.1745	0.173653539
13	11	0.1920	0.190813478
14	12	0.2094	0.207915182
15	13	0.2269	0.224953469
16	14	0.2443	0.241923268
17	15	0.2618	0.258819396
18	16	0.2793	0.275636551
19	17	0.2967	0.292369959
20	18	0.3142	0.309014256
21	19	0.3316	0.325564501
22	20	0.3491	0.34201574
23	21	0.3665	0.358362968
24	22	0.3840	0.374601002
25	23	0.4014	0.390725134
26	24	0.4189	0.406730229
27	25	0.4363	0.422611651
28	26	0.4538	0.438364503
29	27	0.4712	0.45398384
30	28	0.4887	0.469465072
31	29	0.5061	0.484803257
32	30	0.5236	0.499993915
33	31	0.5411	0.515032347
34	32	0.5585	0.529914019
35	33	0.5760	0.544634339
36	34	0.5934	0.559188841
37	35	0.6109	0.57357281
38	36	0.6283	0.587782379
39	37	0.6458	0.601812777
40	38	0.6632	0.615660007
41	39	0.6807	0.629319664
42	40	0.6981	0.642787631
43	41	0.7156	0.656059619
44	42	0.7330	0.669131991
45	43	0.7505	0.682000395
46	44	0.7679	0.694661076

Several different types of graphs have been created to plot the values. The line graph plots the values in Column C and the other 4 graphs look only at the data in F2.

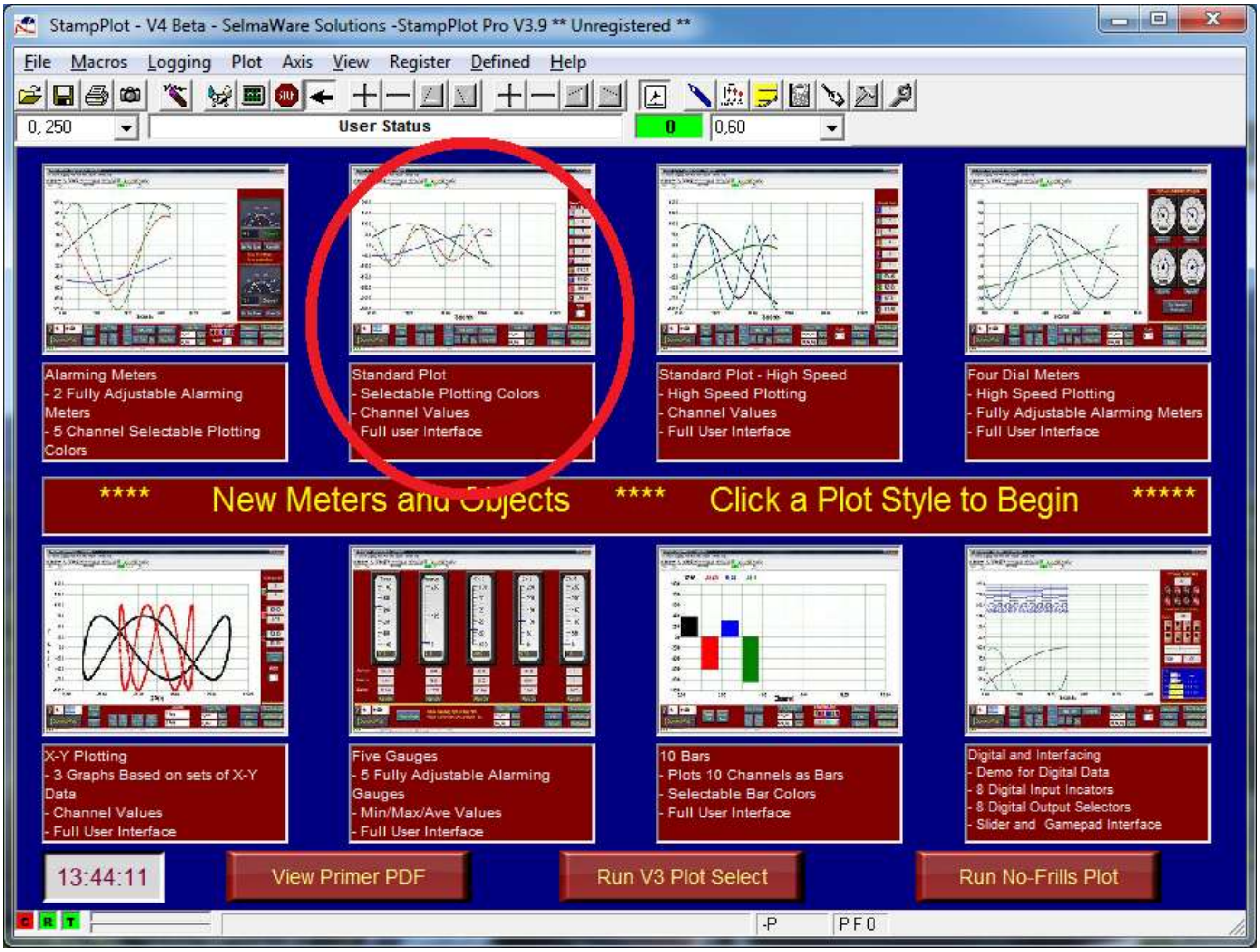
40.21 StampPlot

Another very useful (and exceedingly more comprehensive) data plotting program is StampPlot. Initially lets start with a simple program to send data and plot it over time.

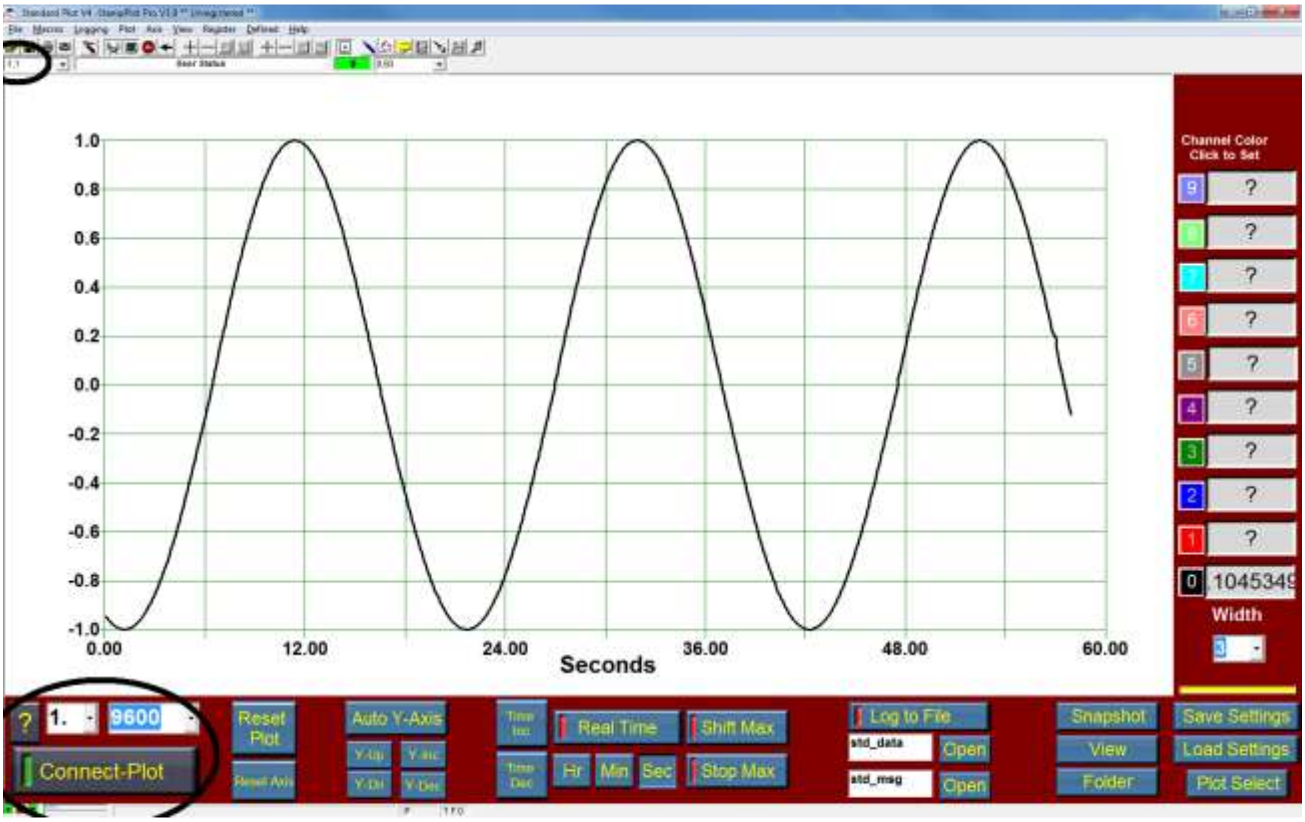
```
Do
   For D = 0 To 359
      'calculate the values to send
      R = Deg2rad(d)
      Sin_x = Sin(r)
      Print #1 , Sin_x
      'display the values on the lcd
      Locate 1 , 1
      Lcd D
      Locate 1 , 8
      Lcd R
      Locate 2 , 1
      Lcd Sin_x
      Waitms 5
   Next
Loop
```

the data is simple to send, just use the line `Print #1 , Sin_x`

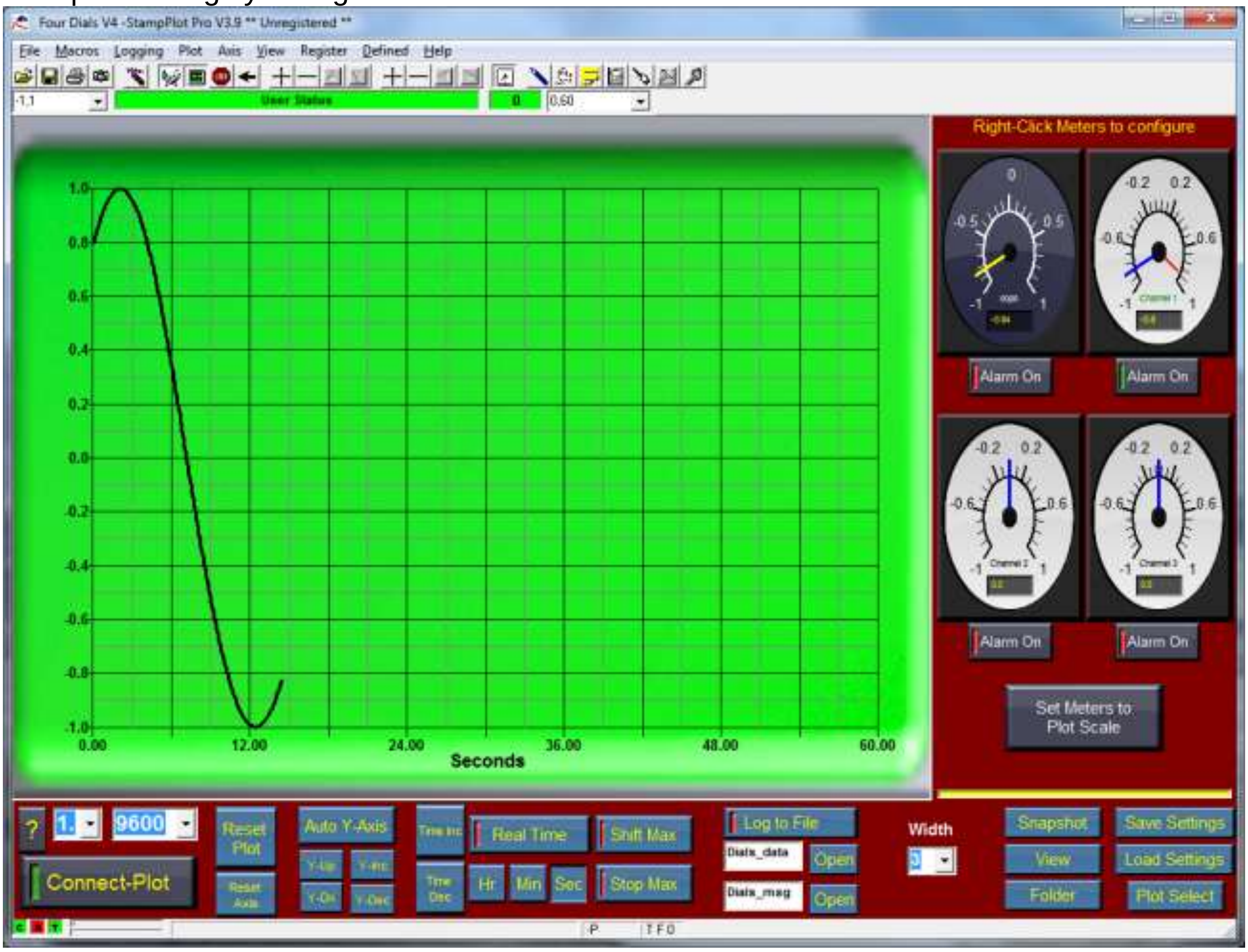
Start StampPlot and select Standard Plot.



In the next screen start the comms (comm. Port 1 and 9600) in the bottom left corner and change the scale in the top left corner to -1,1



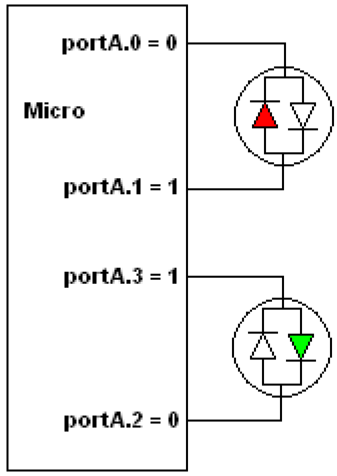
StampPlot is highly configurable with alarms and meters



.

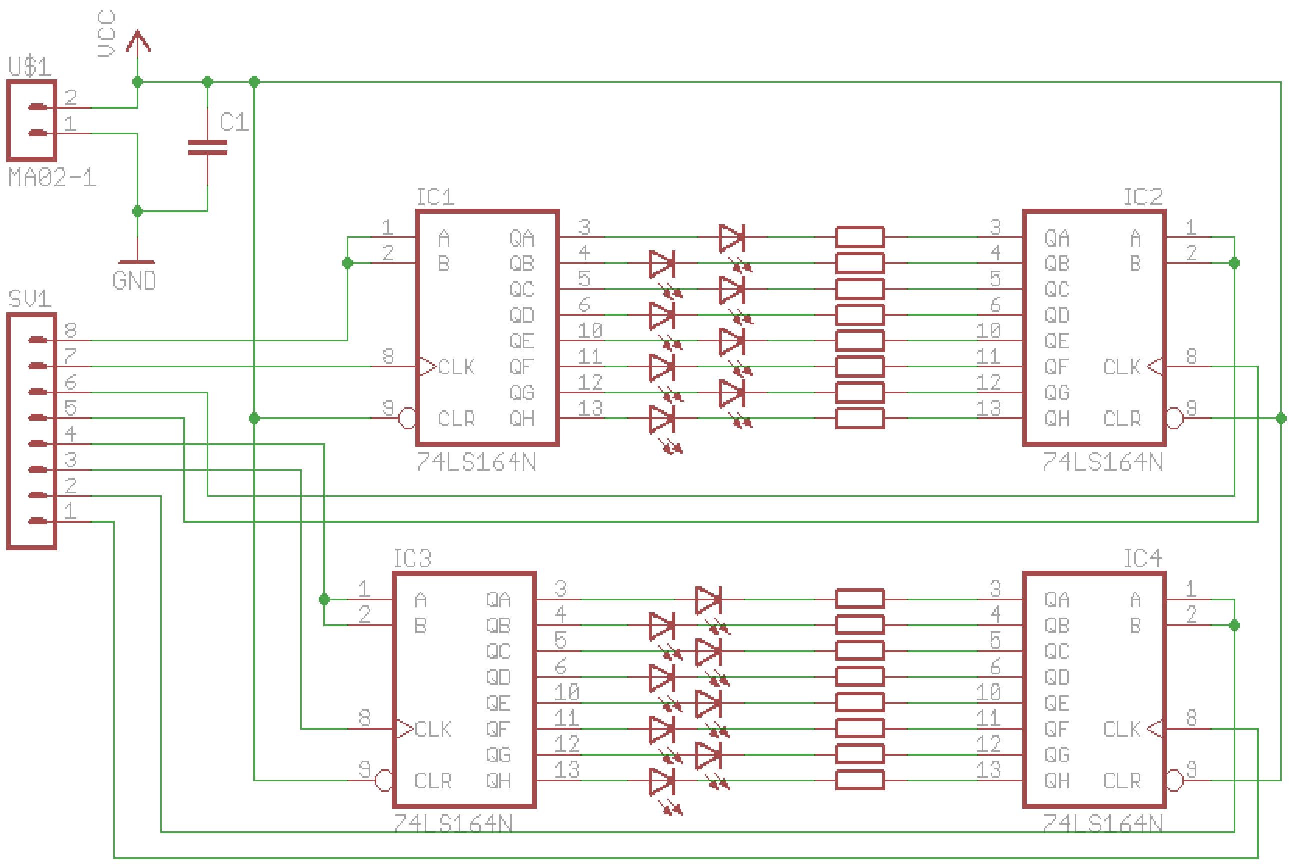
40.22 Serial to parallel

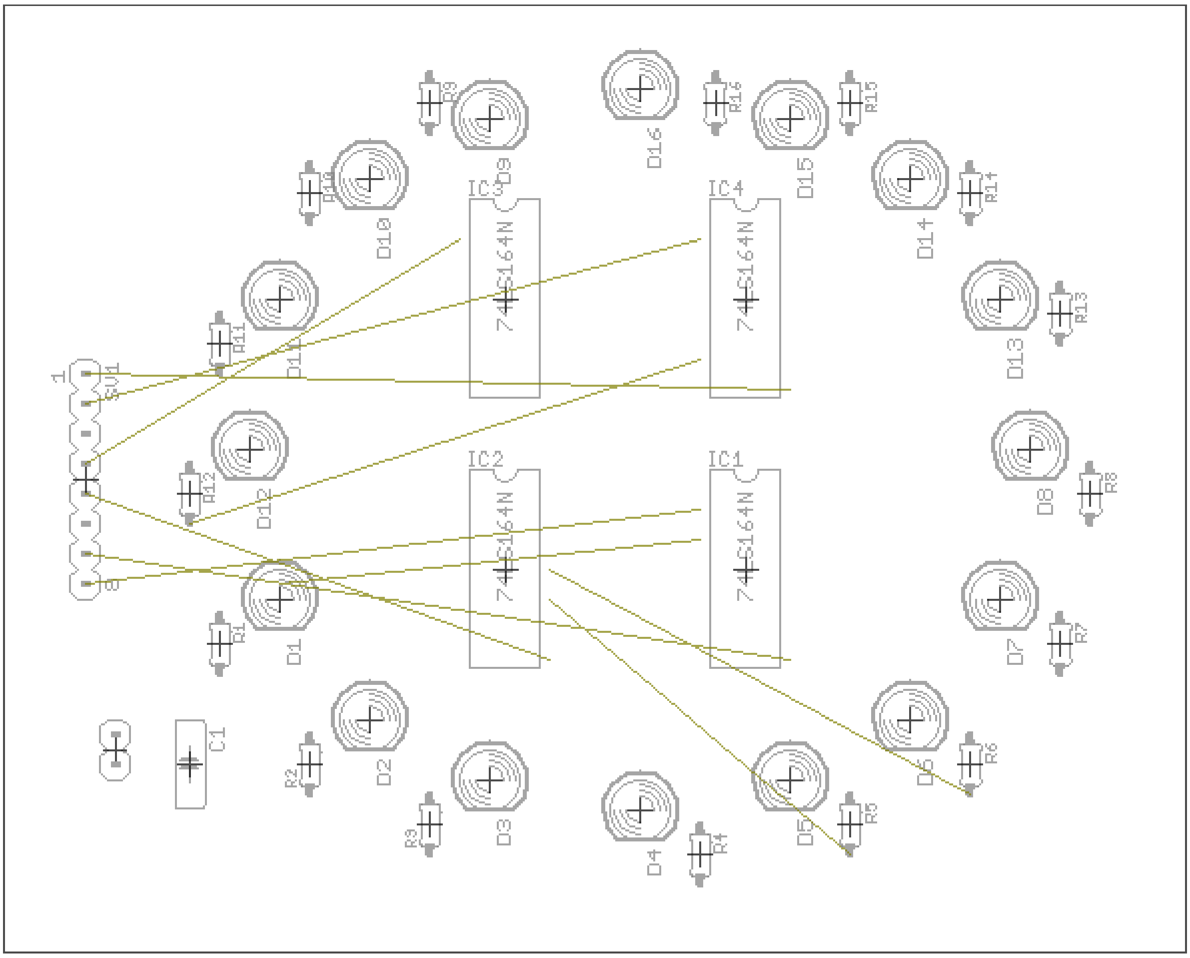
We came across some bi-colour LEDs and wanted to add them to a circuit in a circular pattern.

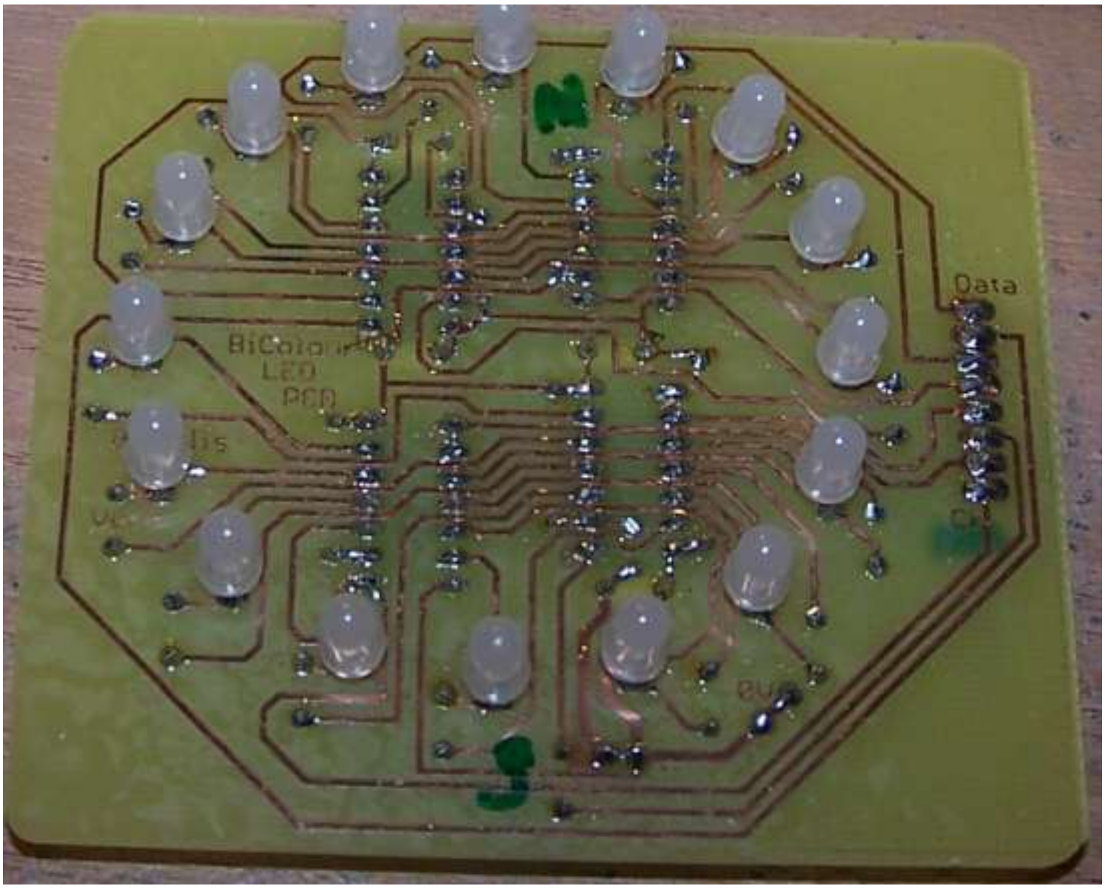


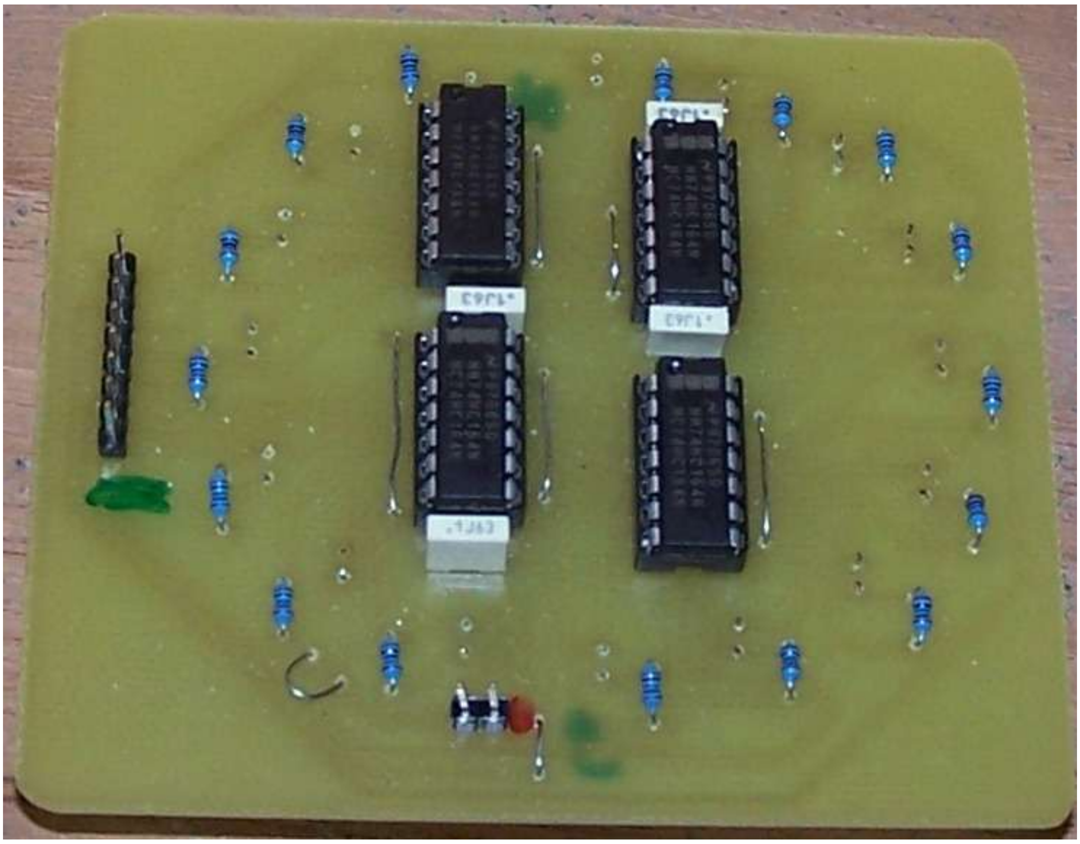
When driven in one direction the LEDs glow red, when reversed they glow green. They could be driven directly from a microcontroller, but would require two I/O pins each as in this diagram

This schematic shows the LS164 serial to parallel ICs used to implement control 16 LEDs and the 8 I/O connections required to drive them . The ICs require a data line and a clock line (so it is synchronous communication)

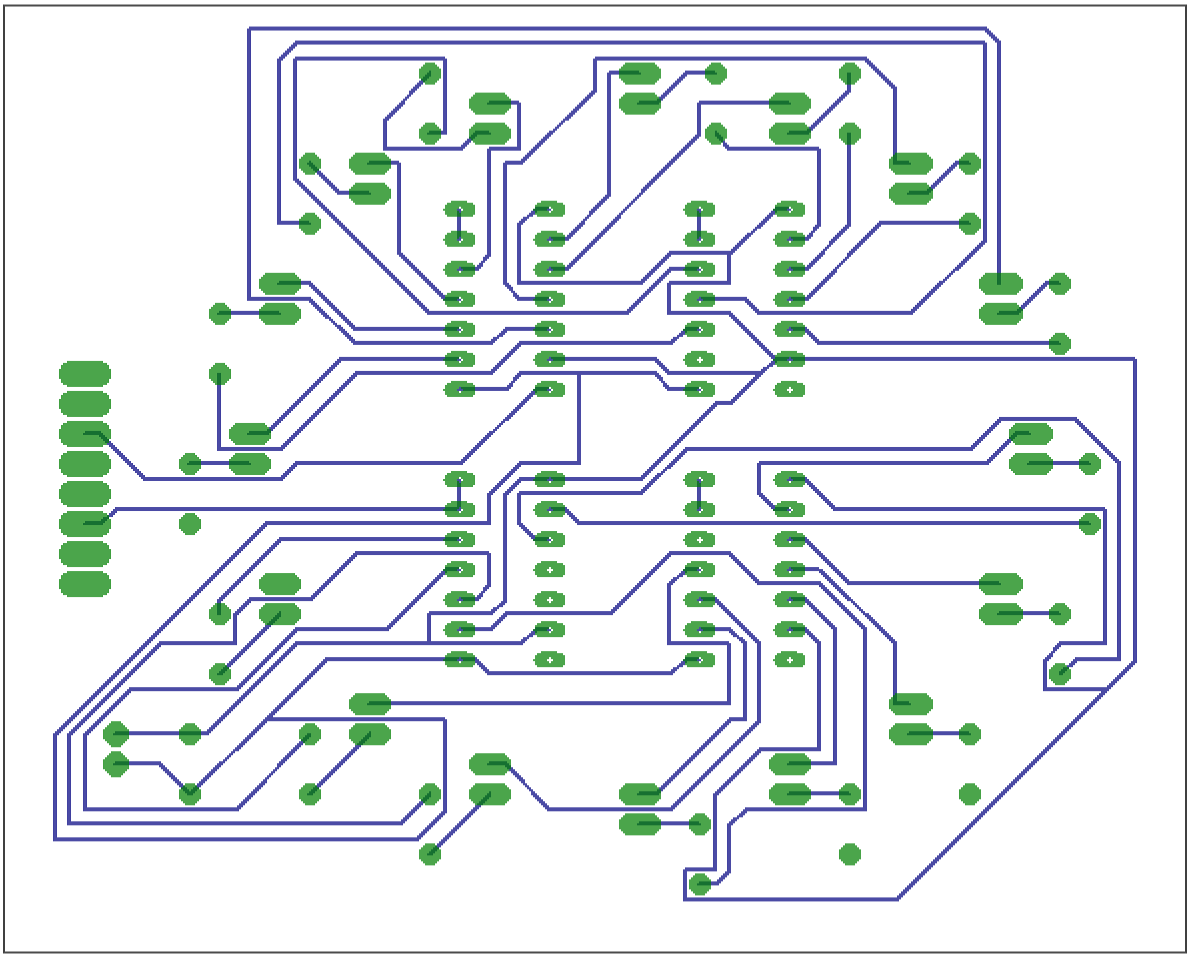



IC3
IC4
IC2
IC1
74HC164N
SV1
C1


Data
BiColour
LED
PCB



PCB track layout



Program to show off the bi color LEDs and serial to parallel conversion

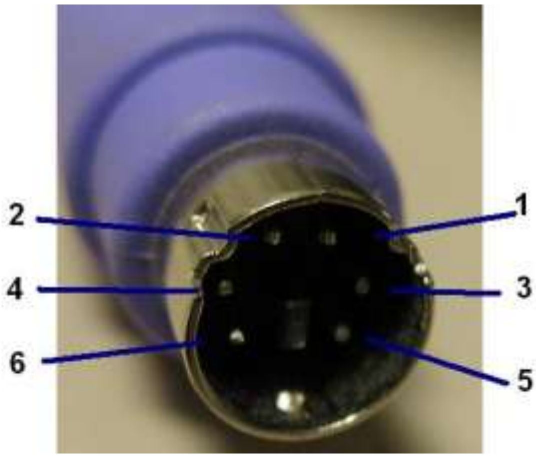
```
'-----------------------------------------------------------------
' Title Block
' Author:
' Date:
' Version: 1.0
' File Name: bicolourled_Ver1.bas
'-----------------------------------------------------------------
' Program Description:
' This program flashes a bicolour continuously  A.6 A.7
' Hardware features
'  leds on portd
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                        ' internal clock
$regfile = "m64.dat"                      ' ATMEGA64-16AI
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output                     '
Config Portb = Output                     '
Config Portc = Output                     '
Config Portd = Output                     '
' 7. Hardware Aliases
Clk4 Alias Portc.7
Data4 Alias Portc.6
Clk3 Alias Portc.5
Data3 Alias Portc.4
Clk2 Alias Portc.3
Data2 Alias Portc.2
Clk1 Alias Portc.1
Data1 Alias Portc.0
' 8. initialise ports so hardware starts correctly

'-----------------------------------------------------------------
' 9. Declare Constants
Const Timedelay = 500                     ' the timing for the flash
Const Pulse = 10000
'-----------------------------------------------------------------
' 10. Declare Variables
Dim I As Byte
Dim J As Byte
Dim Dat As Byte
I = 255
J = 255
'all leds off
Shiftout Data1 , Clk1 , I , 3 , 8 , 32000        'shiftout 8 bits
Shiftout Data2 , Clk2 , J , 3 , 8 , 32000        'shiftout 8 bits
Shiftout Data3 , Clk3 , I , 3 , 8 , 32000        'shiftout 8 bits
Shiftout Data4 , Clk4 , J , 3 , 8 , 32000        'shiftout 8 bits
Wait 3
Dat = &B00000001

'Initialise Variables
```

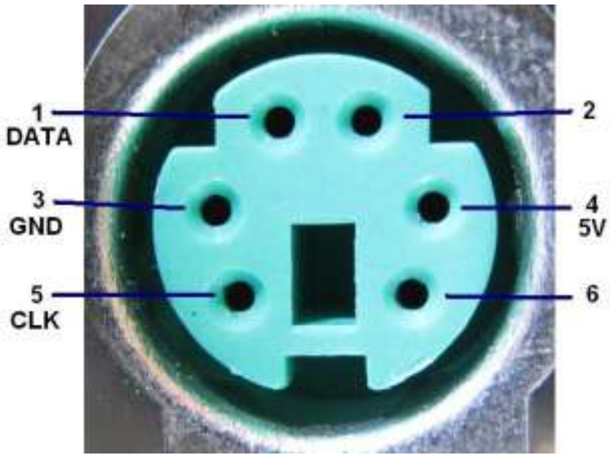
```
'---------------------------------------------------------------
'Program starts here
Do
  Rotate Dat , Left
  I = Dat                                  '0=Rd
  J = 0                                    '0=gn
  Shiftout Data1 , Clk1 , I , 3            ', 8 , 1       'shiftout 8 bits
  Shiftout Data2 , Clk2 , J , 3            ', 8 , 1       'shiftout 8 bits
  Set Porta.0
  Waitms Timedelay
  I = 0                                    '0=Rd
  J = Dat                                  '0=gn
  Shiftout Data1 , Clk1 , I , 3            ', 8 , 1       'shiftout 8 bits
  Shiftout Data2 , Clk2 , J , 3            ', 8 , 1       'shiftout 8 bits
  Reset Porta.0
  Waitms Timedelay
Loop

End                                        'end program
```

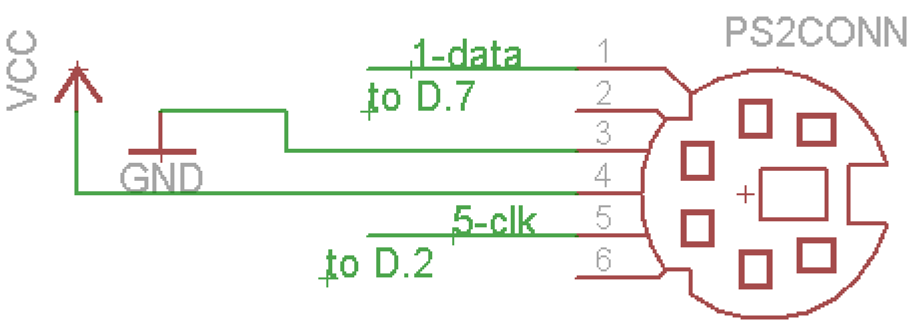
40.23 Keyboard interfacing – synchronous serial data

The computer PS/2 keyboard is an example of synchronous serial communication and can be connected directly to an AVR microcontroller (synchronous means that a clock signal is sent as well as the data signal to help the receiver know the timing for the data).

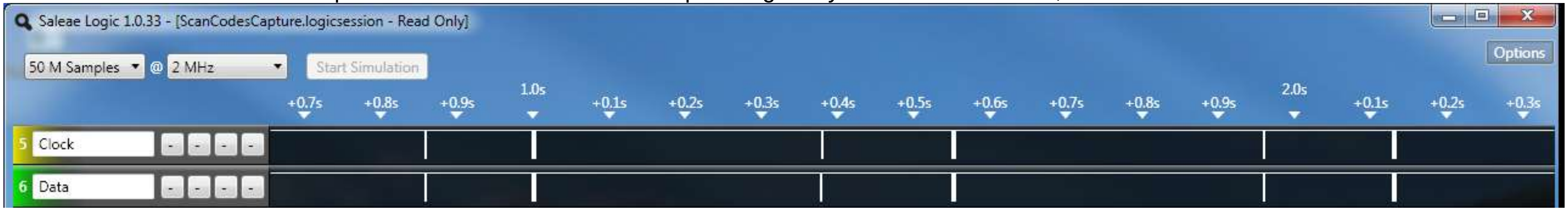
On the left is the PS/2 (or 6-pin mini DIN) plug on a cable, it is known as the male connector. The socket on the right is as seen on a computer motherboard and is called the female connector. Note the wiring on the socket is the mirror image of the plug, and that it is the socket we will be wiring to a microcontroller.

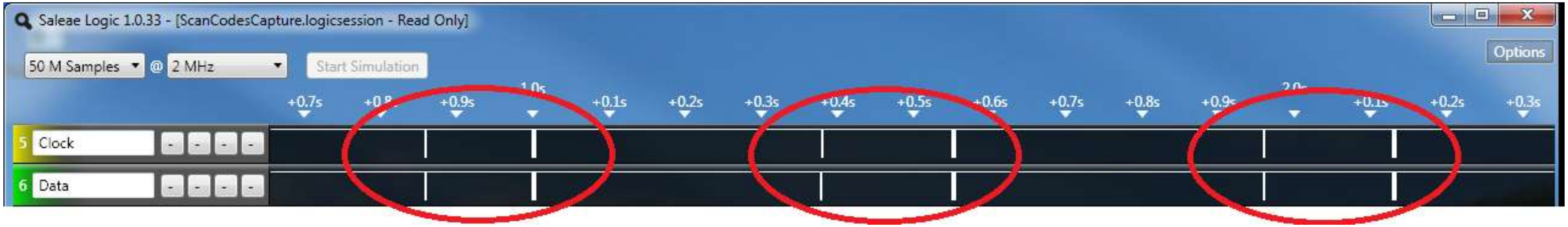




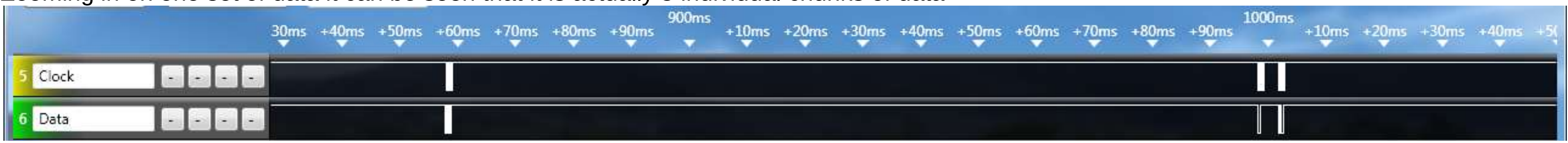


The data from the keyboard has been captured using a Saleae Logic Analyser. These are the 2 lines, data and clock, from the keyboard; and the horizontal scale is 0.1 seconds per division. Here is the result of pressing 3 keys one after the other, there are 3 sets of data

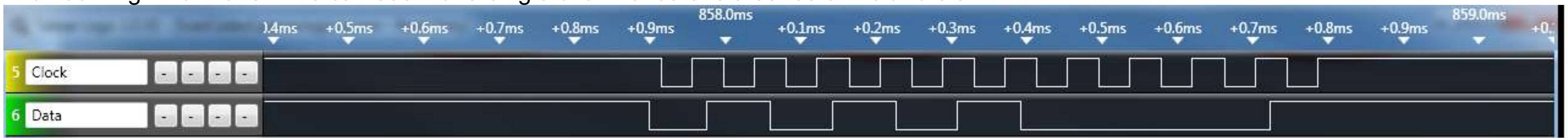




Zooming in on one set of data it can be seen that it is actually 3 individual chunks of data

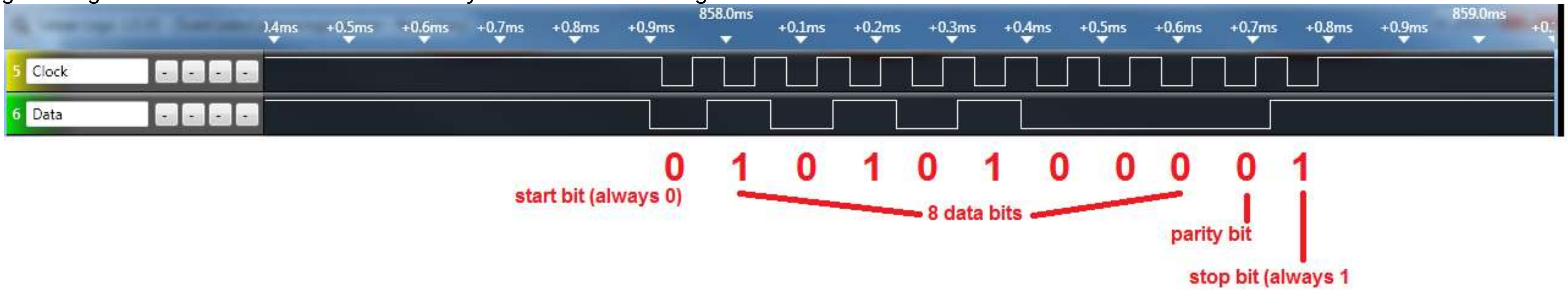


And zooming in further still we can see that a single chunk of data is a series of 1's and 0's



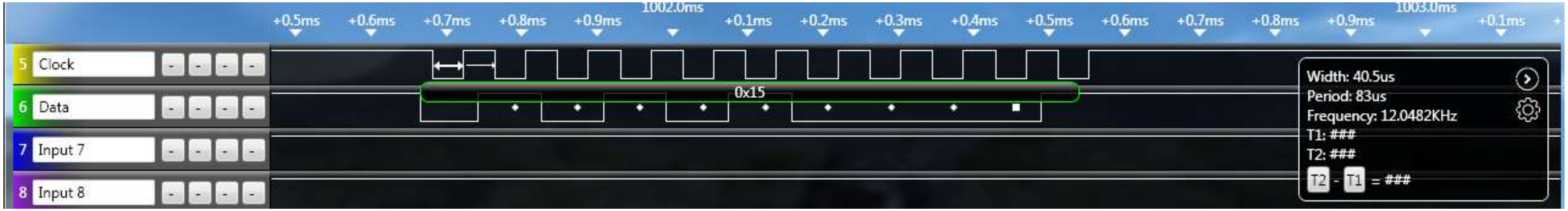
The clock is a regular alternating signal of eleven 1's and 0's, and indicates to us when the data is valid (can be read). The data must be read along wth the clock so there are eleven bits of data even though it appears tere are fewer.

The data sequence is reguar it always consists of a start bit, followed by 8 data bits, then a parity error checking bit and finally a stop bit)
The data is sent LSB (least significant bit) first so when it is used by your micro it is binary 00010101 (which in hex is15_H)
The specification for data from a keyboard can be found on the internet and states that the data bit must be valid at least 5uS before the clock goes negative. So we can read the data any time after the clock goes low.

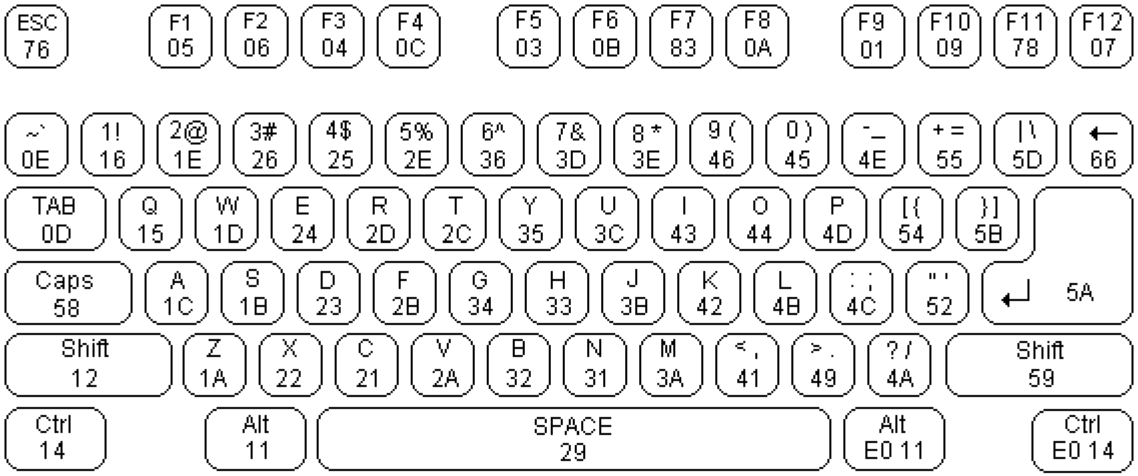


The logic analyser has the ability to interpret the data for us , its just a matter of working out its speed (bits per second) which is around 12,000 bits per second for the keyboard which we tested.

Once these settings are made the logic analyser software will show the hex code for the data.



Each key of the key board has a unique scan code (some have a sequence) e.g. Ctrl is E0(hex) then 14(hex)
The key that corresponds to the scan code of 15(hex) is the letter ‘Q’

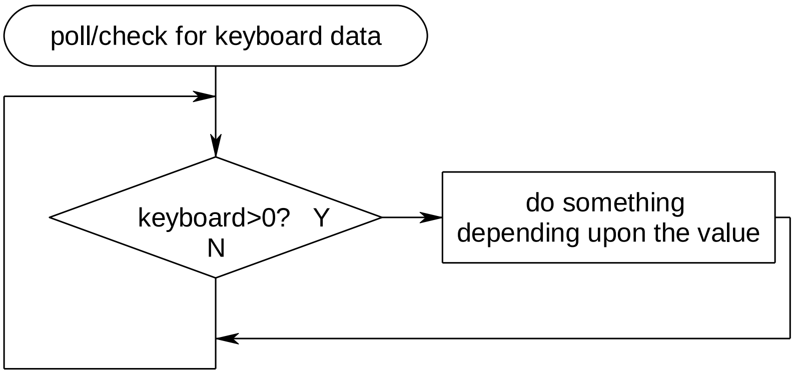


Party
Along with the data a single parity bit is sent; the parity bit is set (to 1) if there is an even number of 1's in the data bits or reset (to 0) if there is an odd number of 1's in the data bits. In our case the data has 3 bits set to 1 so a 0 is sent. The purpose of parity is to help the receiver know if the message was received correctly. At the receiving end the number of 1’s is added up and compared to the parity bit, if there is a match it was assumed that the data was received correctly. However if a single bit of data was corrupted then the receiver could identify a problem (wouldn’t this be useful when people are talking to each other!!)

The use of parity along with the use of a synchronous clock makes this communication protocol reasonable robust to interference.
Do note though that it is not completely immune to corruption as if 2 bits of thedata were corrupted then the parity bit might still be correct.

Lots more information about the data being sent (protocol) can be found at http://www.computer-engineering.org/ps2protocol/

There are a number of choices we have when we want to receive data fro the keyboard.
The first is to use the built in function in Bascom GETATKBD(). Along with this function we need to do a conversion process. Microcontrollers don't use scan codes for letters(and digits) they use the ascii code so the received scan code is translated to ascii code using a lookup table.




```
'------------------------------------------------------------------
'Title Block
' Author:   B.Collis
' Date:     July 2010
' File Name: ps2kbV1.bas
'------------------------------------------------------------------
' Program Description:
' Hardware Features:
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' keypad connected as per R4R circuit on 1 ADC line
' lm35 on adc
' AREF PIN32 disconnected - uses internal 2.56V reference
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                       'the crystal we are using
$regfile = "m32def.dat"                  'the micro we are using
'------------------------------------------------------------------
'Hardware Setups
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 20 * 4                      'configure lcd screen
Config Keyboard = Pind.6 , Data = Pind.7 , Keydata = Keydata
Config Portd = Input

'-----------------------------------------------------------------
'Declare Constants
'------------------------------------------------------------------
'Declare Variables
Dim Kb_textstring As String * 20
Dim Kb_character As String * 1
Dim Kb_bytevalue As Byte
Dim Length As Byte
'Initialise Variables

'------------------------------------------------------------------
```

```
'Program starts here
Cursor Off
Cls
Locate 1 , 1
Lcd "keyboard reader"
'here are 2 examples of what you can do with the keyboard
'--------------------------------------------------------
Do
   'read the keyboard
   Kb_bytevalue = Getatkbd()
   'if a recognised key is pressed then do something
   If Kb_bytevalue > 0 Then
      Locate 2 , 1
      Lcd "byte value=" ; Kb_bytevalue ; "  "
      Kb_character = Chr(kb_bytevalue)
      Locate 3 , 1
      Lcd "ascii char=" ; Kb_character ; "  "
   End If
Loop

'--------------------------------------------------------
Do
   'wait until a recognised key is pressed
   Do
       Kb_bytevalue = Getatkbd()
   Loop Until Kb_bytevalue <> 0
   Locate 2 , 1
   Lcd "byte value=" ; Kb_bytevalue ; "  "
   Kb_character = Chr(kb_bytevalue)
   Locate 3 , 1
   Lcd "ascii char=" ; Kb_character ; "  "
Loop

End



'convert the data from the keyboard to an ascii character
'only ascii characters are here if you want other data to be recognised
'     then change the table specific key below from a 0 to another number
Keydata:
'normal keys lower case
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 , 50 ,
0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 , 114
, 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117 , 55
, 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 , 112 ,
43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 , 0 ,
0
```

```
'shifted keys UPPER case
Data 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0
Data 0 , 0 , 0 , 0 , 0 , 81 , 33 , 0 , 0 , 0 , 90 , 83 , 65 , 87 , 34 , 0
Data 0 , 67 , 88 , 68 , 69 , 0 , 35 , 0 , 0 , 32 , 86 , 70 , 84 , 82 , 37 ,
0
Data 0 , 78 , 66 , 72 , 71 , 89 , 38 , 0 , 0 , 76 , 77 , 74 , 85 , 47 , 40 ,
0
Data 0 , 59 , 75 , 73 , 79 , 61 , 41 , 0 , 0 , 58 , 95 , 76 , 48 , 80 , 63 ,
0
Data 0 , 0 , 0 , 0 , 0 , 96 , 0 , 0 , 0 , 0 , 13 , 94 , 0 , 42 , 0 , 0
Data 0 , 62 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45
```

Now there is a slight problem with the Bascom Getatkbd() function and that is that once you enter it there is no easy way out of it unless a key is pressed.

It is possible to get out of the routine by starting a timer before caling getatkbd(),and when the timer timesout set the ERR flag; once that is set the getatkbd() routine will exit.

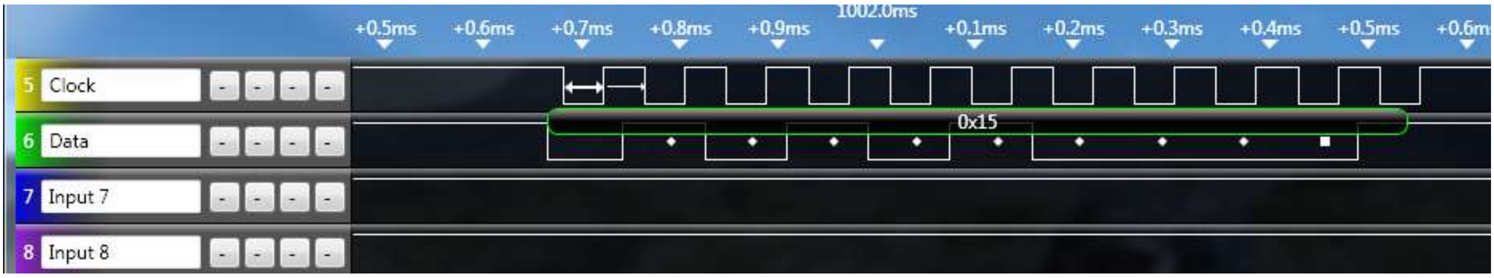
```
Do
   'read the keyboard
   Start timer
   Kb_bytevalue = Getatkbd()
   Stop timer
   'if a recognised key is pressed then do something
   If Kb_bytevalue > 0 Then
      Locate 2 , 1
      Lcd "byte value=" ; Kb_bytevalue ; "  "
      Kb_character = Chr(kb_bytevalue)
      Locate 3 , 1
      Lcd "ascii char=" ; Kb_character ; "  "
   End If
Loop

Timer_isr:
   Err=1
   Stop timer
return
```

Altough this is a for using a keyboard it is not really an elegant solution to crash out of a loop by creating an error. We an write our own software.

Before we can go on though we need to know about the scan codes sequence. The keyboard sends (at least) three characters everytime a key is pressed.
For an ‘a’ the codes 1C F0 1C will be sent,
For an ‘s’ the codes 1B F0 1B will be sent.
If we are to write our own handler for keycoodes then we must ignore F0 and the repeated scan code.

40.24 Keyboard as asynchronous data

For a one-off project a simple method of dealing with a keyboard is to treat it as an asynchronous serial data connection and to ignore the clock line.



The logic analyser was used to monitor the two input signals, clock and data, however it was also used to analyse the data signal and it did this independently of the clock signal (or asynchronously). It can do this because the data bits are all the same width.

Using the ‘soft’ UART features in Bascom we can open a channel for receiving serial data on any pin.

```
'-------------------------------------------------------------------
'Title Block
' Author:   B.Collis
' Date:     July 2010
' File Name: ps2kb-serialtrial-V1.bas
'-------------------------------------------------------------------
' Program Description:
' Hardware Features:
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' AREF PIN32 disconnected - uses internal 2.56V reference
' make kb clock
'-------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                      'the crystal we are using
$regfile = "m32def.dat"                 'the micro we are using
'-------------------------------------------------------------------
'Hardware Setups
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 20 * 4                     'configure lcd screen

Open "comd.3:12000,8,o,1" For Input As #1

'aliases
Kd_data Alias Pind.7
Kb_clock Alias Pind.6
Kb_control Alias Portd.6
'Config Kb_data = Input
Config Kb_clock = Input


'-------------------------------------------------------------------
'Declare Constants
'-------------------------------------------------------------------
'Declare Variables
Dim Kb_textstring As String * 20
Dim Kb_character As Byte
Dim Kb_bytevalue As Byte
```

```
Dim Kb_bytevalue_old As Byte
Dim Repeat As Bit
'Initialise Variables

'-----------------------------------------------------------------
'Program starts here
Cursor Off
Cls
Locate 1 , 1
Lcd " async keyboard reader "
'--------------------------------------------------------
Cls
Lcd "serial kb in"
Do
   'look for input
   'the data is not sent as a single keycode for each character pressed
there are 3 data bursts
   'e.g.'a' sends 1C F0 1C ,so we ignore F0 and respond to only the first 1C
   Kb_bytevalue = Inkey(#1)
   If Kb_bytevalue > 0 And Kb_bytevalue <> &HF0 Then        'ignore F0
      If Kb_bytevalue <> Kb_bytevalue_old Then        'only respond once
         'remember char for next time thru loop
         Kb_bytevalue_old = Kb_bytevalue
         'get the ascii value for the scan code value
         Kb_character = Lookup(kb_bytevalue , Keydata)
         'build a string of characters
         Kb_textstring = Kb_textstring + Chr(kb_character)
         'display some stuff on the LCD for test purposes
         Locate 2 , 1
         Lcd Hex(kb_bytevalue) ; " " ; Kb_bytevalue
         Locate 3 , 1
         Lcd Chr(kb_character)
         Locate 4 , 1
         Lcd "                     "
         Locate 4 , 1
         Lcd Kb_textstring
      Else
         Kb_bytevalue_old = 0            'we repeat key presses
      End If
      If Kb_bytevalue = &H5A Then        'we got a return key
         'do something with it?
      End If
   End If
Loop


'
End



'convert the data from the keyboard to an ascii character
'only ascii characters are here if you want other data to be recognised
'    then change the table specific key below from a 0 to another number
Keydata:
'normal keys lower case
```

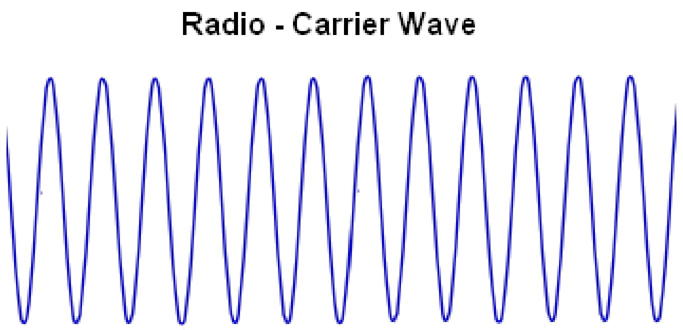
```
Data 0 , 0 , 0 , 0 , 0 , 200 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , &H5E , 0

'                          q     1                      z      s     a     w    2
Data 0 , 0 , 0 , 0 , 0 , 113 , 49 , 0 , 0 , 0 , 122 , 115 , 97 , 119 , 50 , 0
Data 0 , 99 , 120 , 100 , 101 , 52 , 51 , 0 , 0 , 32 , 118 , 102 , 116 , 114 , 53 , 0
Data 0 , 110 , 98 , 104 , 103 , 121 , 54 , 7 , 8 , 44 , 109 , 106 , 117 , 55 , 56 , 0
Data 0 , 44 , 107 , 105 , 111 , 48 , 57 , 0 , 0 , 46 , 45 , 108 , 48 , 112 , 43 , 0
Data 0 , 0 , 0 , 0 , 0 , 92 , 0 , 0 , 0 , 0 , 13 , 0 , 0 , 92 , 0 , 0
Data 0 , 60 , 0 , 0 , 0 , 0 , 8 , 0 , 0 , 49 , 0 , 52 , 55 , 0 , 0 , 0
Data 48 , 44 , 50 , 53 , 54 , 56 , 0 , 0 , 0 , 43 , 51 , 45 , 42 , 57 , 0 , 0
```

41 Radio Data Communication

41.1 An Introduction to data over radio

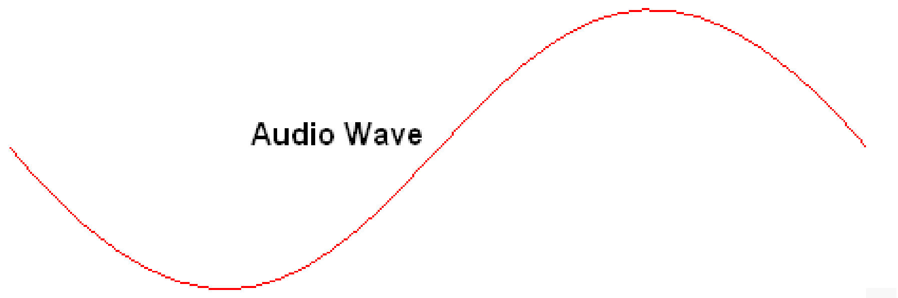
Radio (electromagnetic) waves are used to transfer information from one place to another through the atmosphere (that's without wires) . A radio wave consists of two signals, **a carrier wave** and the information to be sent called the **modulating wave** this wave could be audio or digital data.

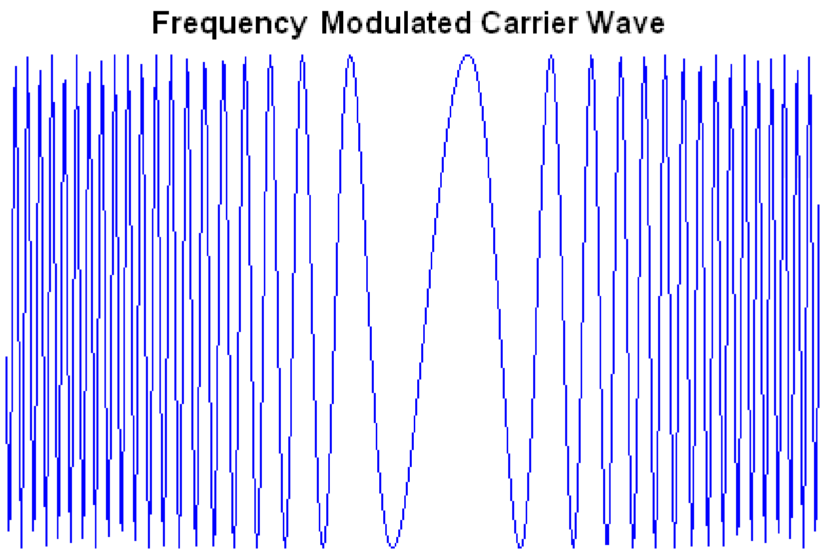
These two are combined together to produce the radio signal. There are many different ways that the carrier can be modulated. With audio signals this can be AM (amplitude modulation), FM (frequency modulation), PM phase modulation.



In FM the carrier signal is modulated by an audio signal.

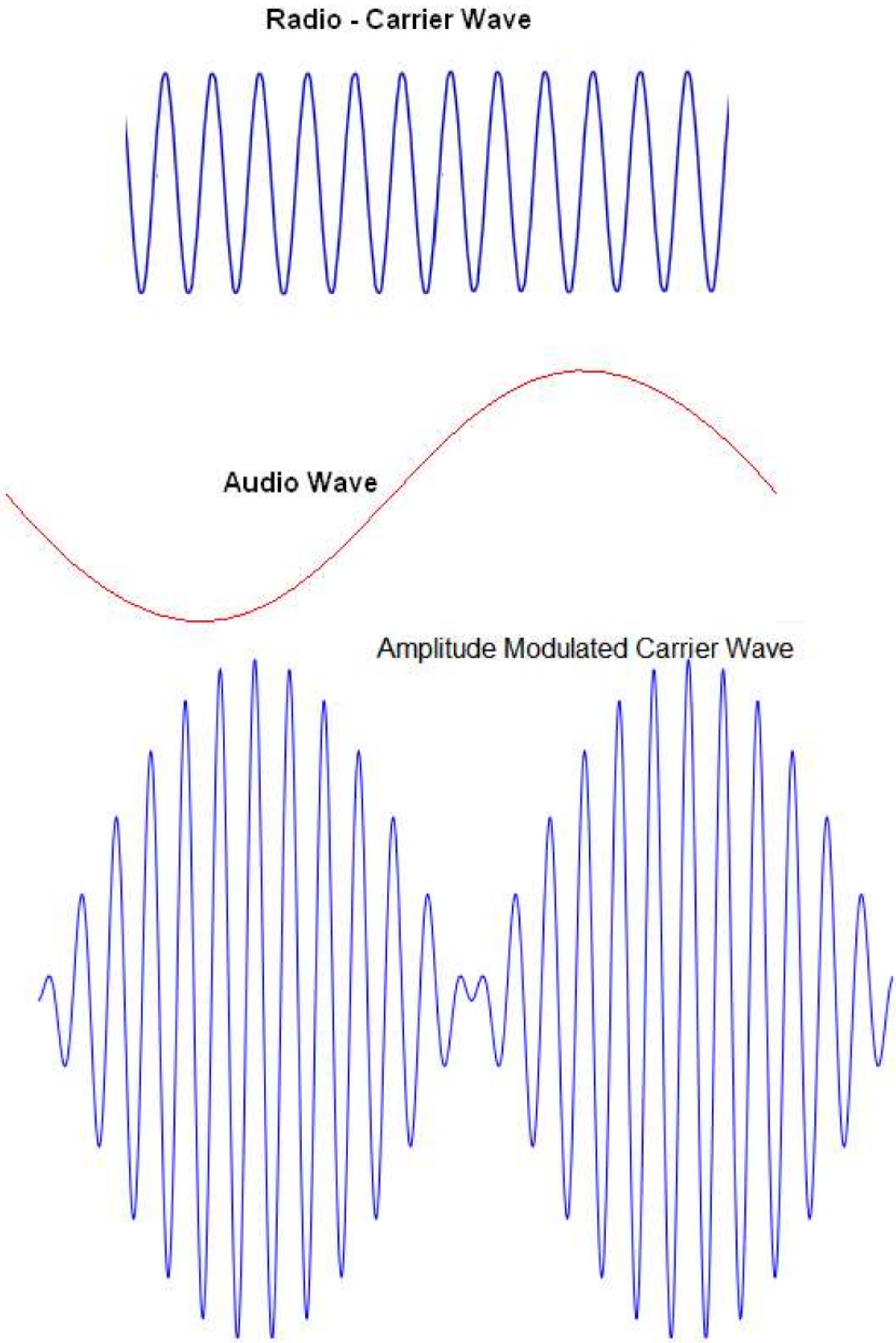
If the carrier is 89.8MHz (Life FM) and an audio tone is applied then the signal transmitted will vary in frequency depending upon the frequency and amplitude of the audio wave.





In Amplitude modulation the frequency of the carrier wave is fixed however its amplitude changes in time with the modulating signa,. e.g National Radio 756Khz.

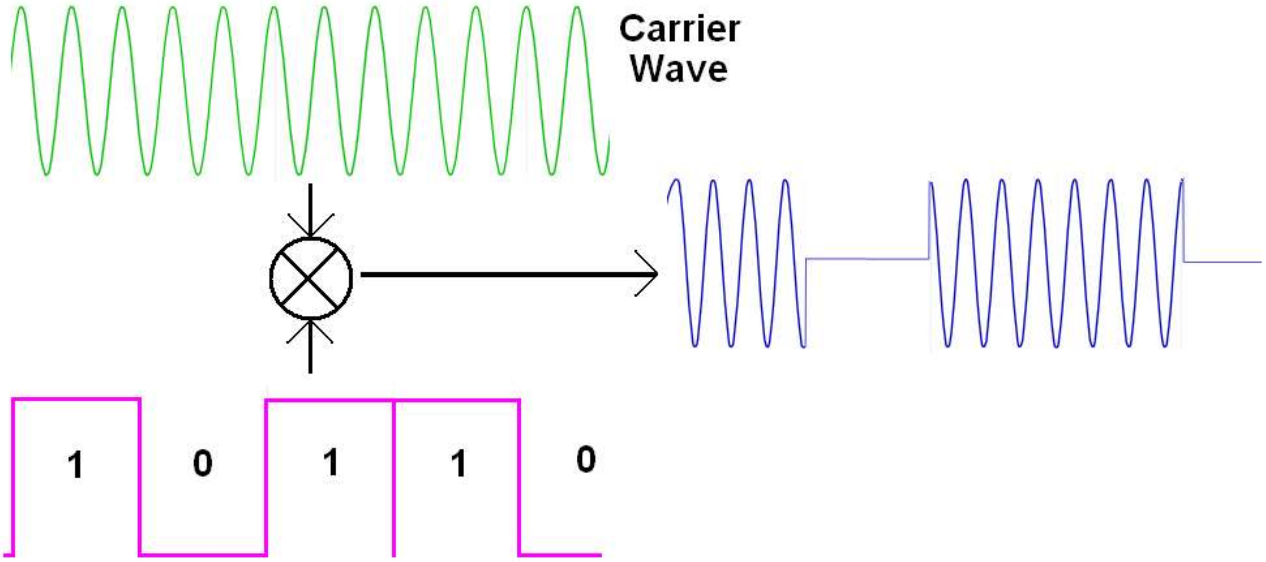
AM picks up interference from other electrical and electronic devices and is noiser than FM.



41.1.1 Pulse modulation

Data is often sent using some form of pulse modulation, pulses represent either a 1 or 0.

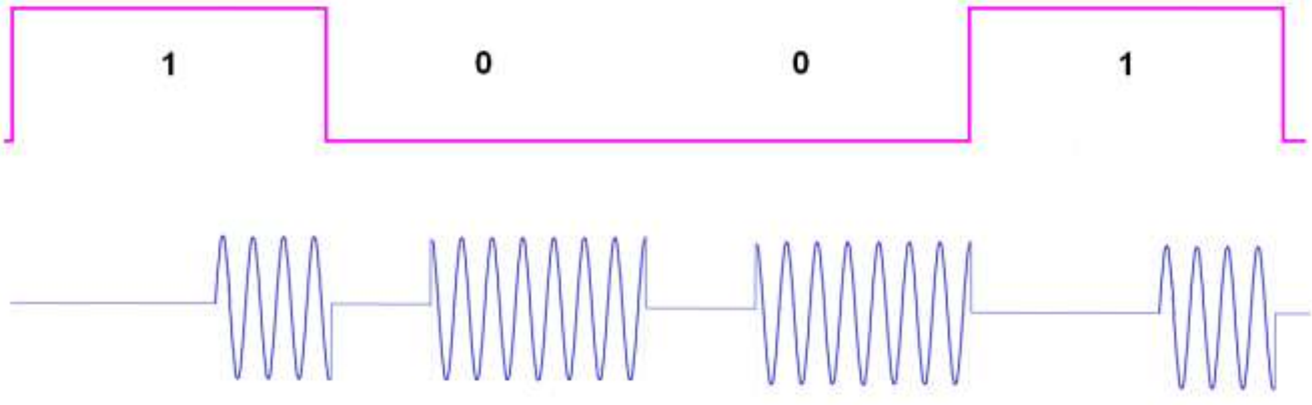
When sending data over any communication link it is important to realise that the system is asynchronous (no clock) so the receiver relies solely on the incoming signal to rebuild the data. If we want to send data then we need to send something for a '1' and we need to send something for a '0' We canot rely on the absence of data to be a '0' as in this diagram below.



A receiver just cannot reliably determine a zero; as how can it determine that an absence of signal is a zero or due to a lost or broken transmission? Also how long is a 1, if 111 is sent will the system get a 1, a 11 or 111?

Digital modulation systems range from very simple to highly highly complex.
OOK is 'on off keying' (keying is the term originally used to describe controlling a radio carrier wave with a Morse key),

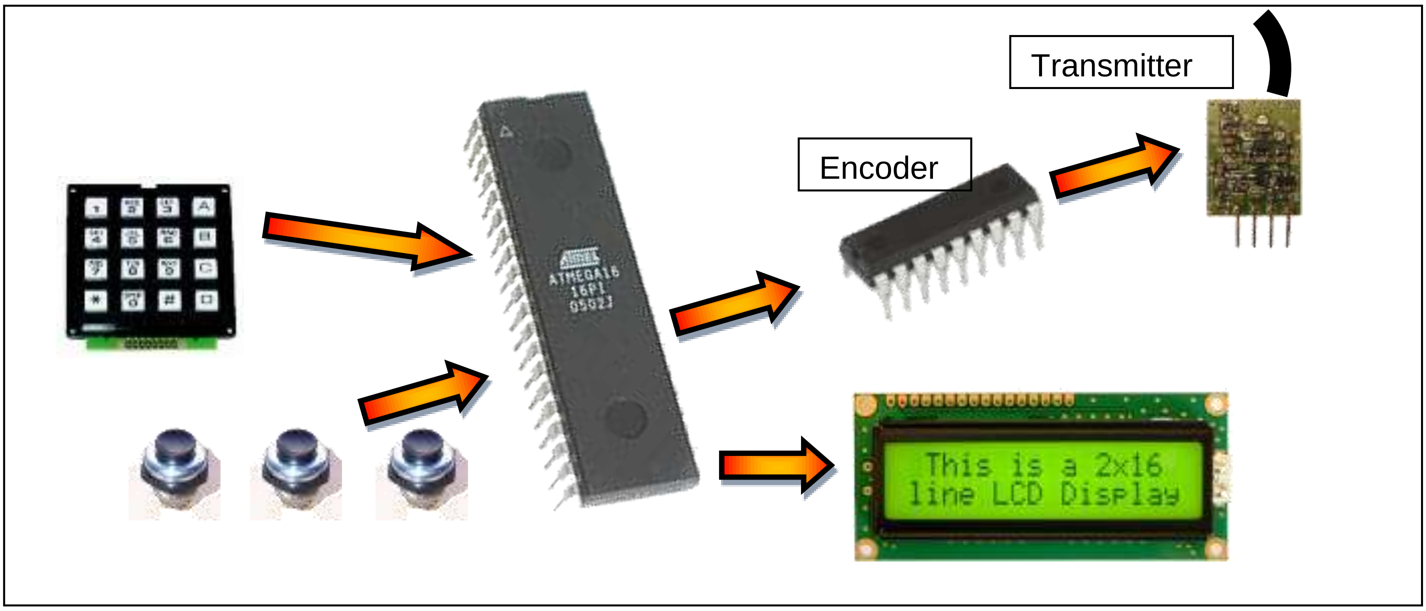
Using OOK the signal is turned on and off in patterns to send 1's and 0's. This is asynchronous, which means that the receiver has to figure out from the transmitted signal what is a 1 and what is a 0. The sequence is very easy to receive though as the overall length of a 1 and 0 is the same, the difference is the length of time the transmitted signal is present.

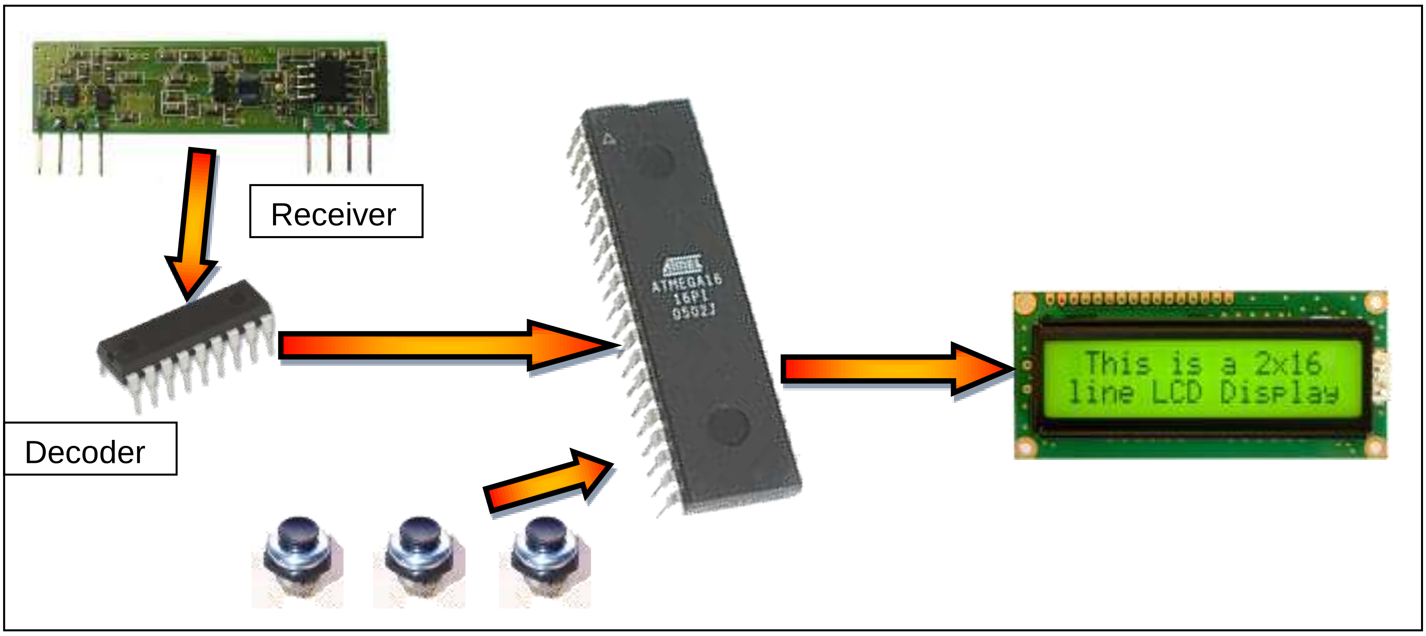


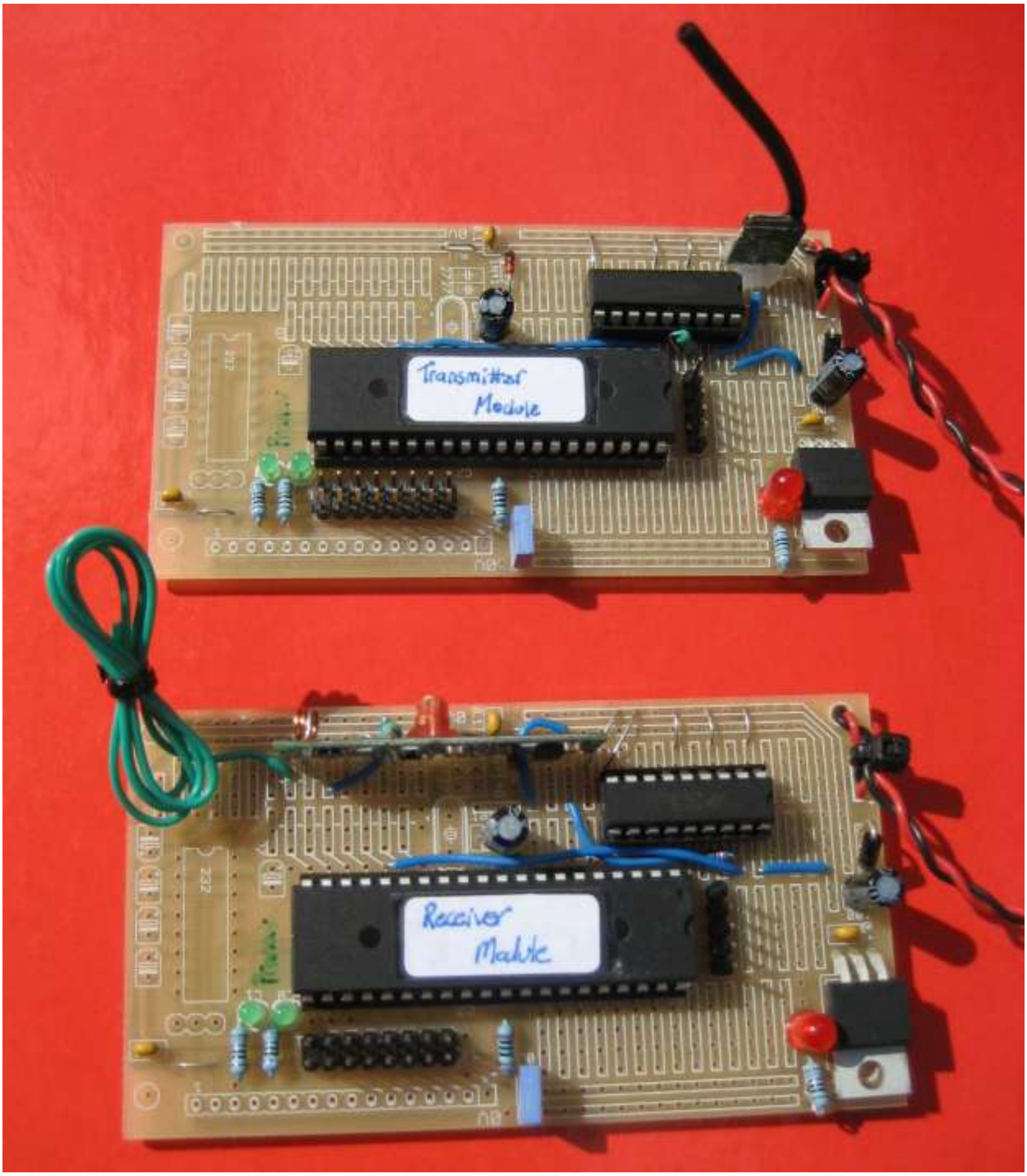
434MHz is a frequency that can be used in many countries for free (unlicensed) radio transmission and is commonly used in systems such as remote controlled garage doors.

There are a large range of transmitters, receivers and tranceivers (a device which both transmits and receives) available in 434Mhz.

There are also simple encoder and decoder ICs to help with the modualation of the signals. Here is a block diagram of a student (PB) radio system that was designed to send messages from loation to another.

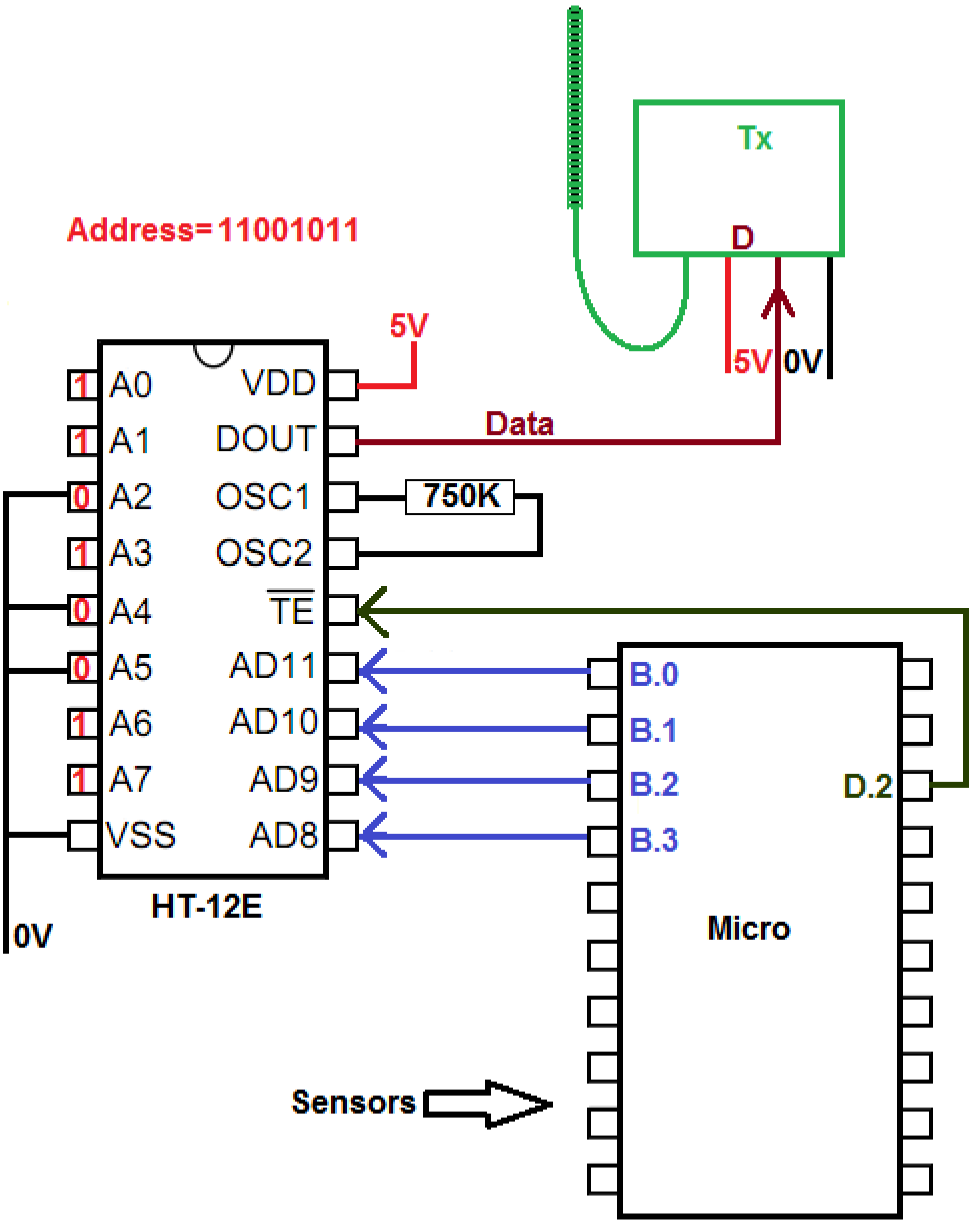




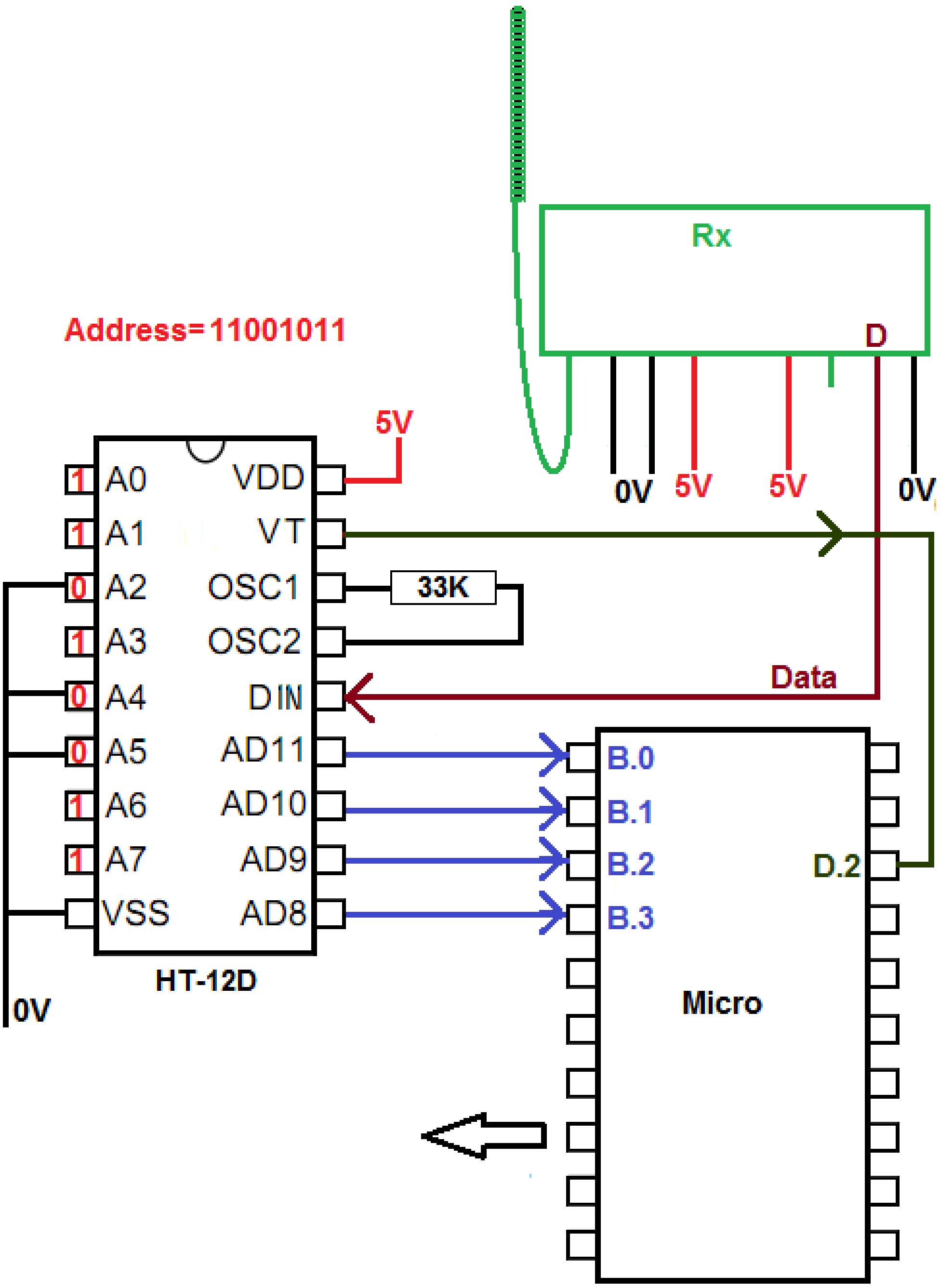


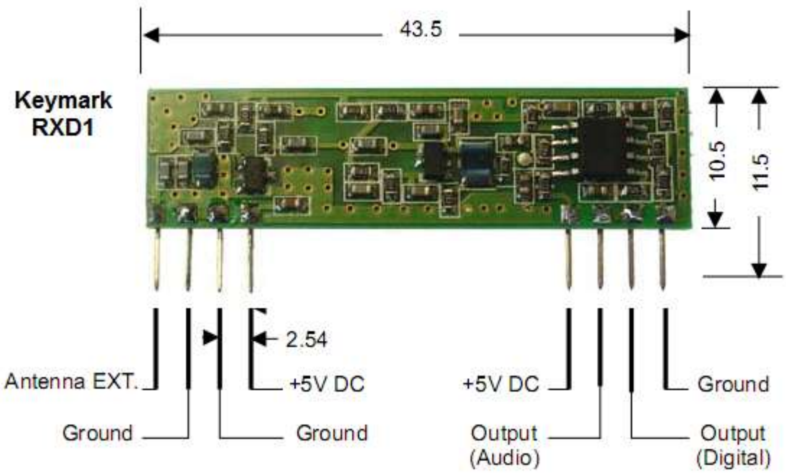
The transmitter has a built in antenna, the receiver has a wire soldered to it as an antenna (green wire currently cable tied in the picture). This needs to be 16.4cm long, if you were making your own PCB you make it a track, or you could also wind 24 turns of 0.5mm wire around something 3.2mm in diameter.

In this partial schematic the HT12Encode receives 4 bits of data from the microcontroller and sends it along with the 8 address bits serially to the transmitter. The speed of the data is set by the value of the resistor. Also any convenient pins can be used on the microcontroller.



The receiving system is very similar to the transmitting system, the receiver board has more power pins to connect and two output pins, audio out and data out. The audio out pin is not used. It is essential that the address on the HT12D is the same as that on the HT12E, otherwise the data will be ignored.

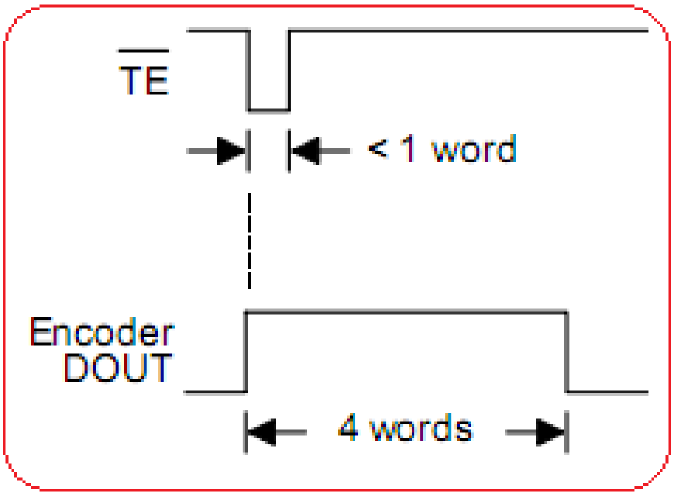




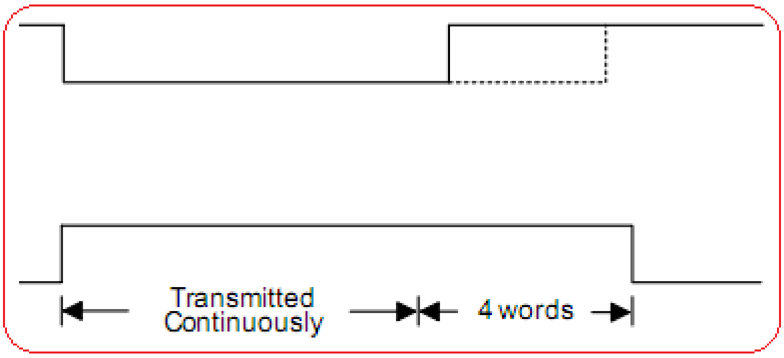
41.2 HT12E Datasheet, transmission and timing

It is quite important to gain experience reading manufacturers datasheets, it is worth reading this with the datasheet for the HT12 open as well. One confusing thing about datasheets is that they sometimes cover a number of different parts in one sheet. This datsheet covers the HT12A and HT12E, the HT12A is used for infrared remote controls the HT12E for RF (radio) . Datasheets also have various pinouts for the ICs such as DIP (dual inline package) and SOP (small outline package) in this case. Make sure you order the right one!

In the datsheet you will find timing diagrams, they occur a great deal in electronics; this diagram has been taken from the HT12E datasheet and modified a little to help explain its detail.



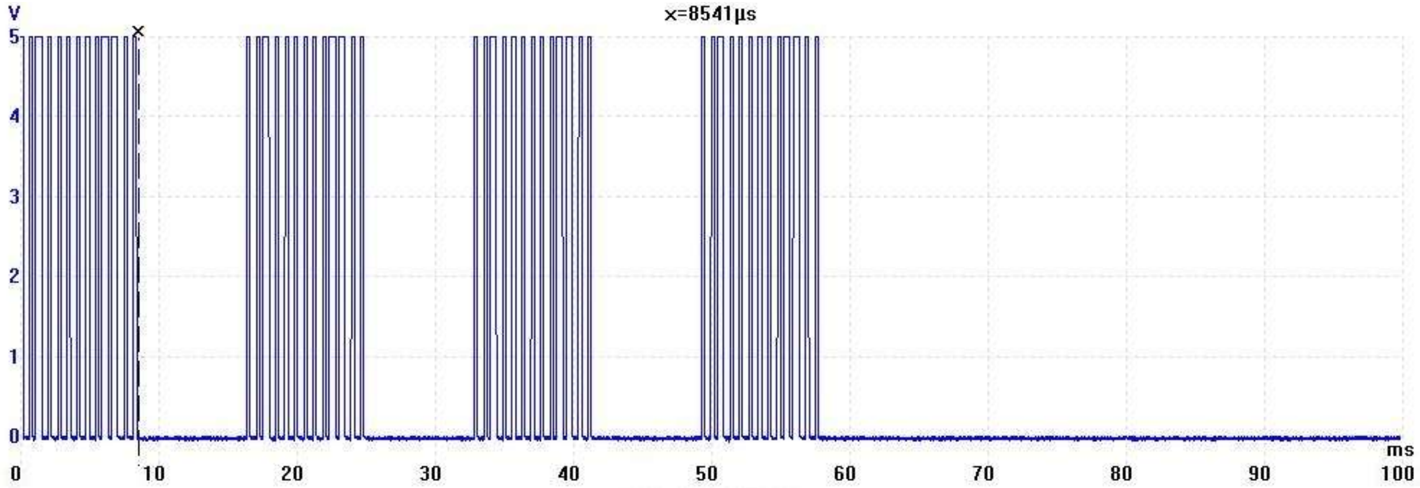
In this diagram two time-voltage graphs are drawn one above the other, the reason for this is that they line up in time. When TE (transmit enable) goes low Dout (data out) goes high and sends the data 4 times. The line or bar above the TE in the daatsheet means that it is an active low signal, i.e. the line should usually be high and when it goes low the IC will do something.



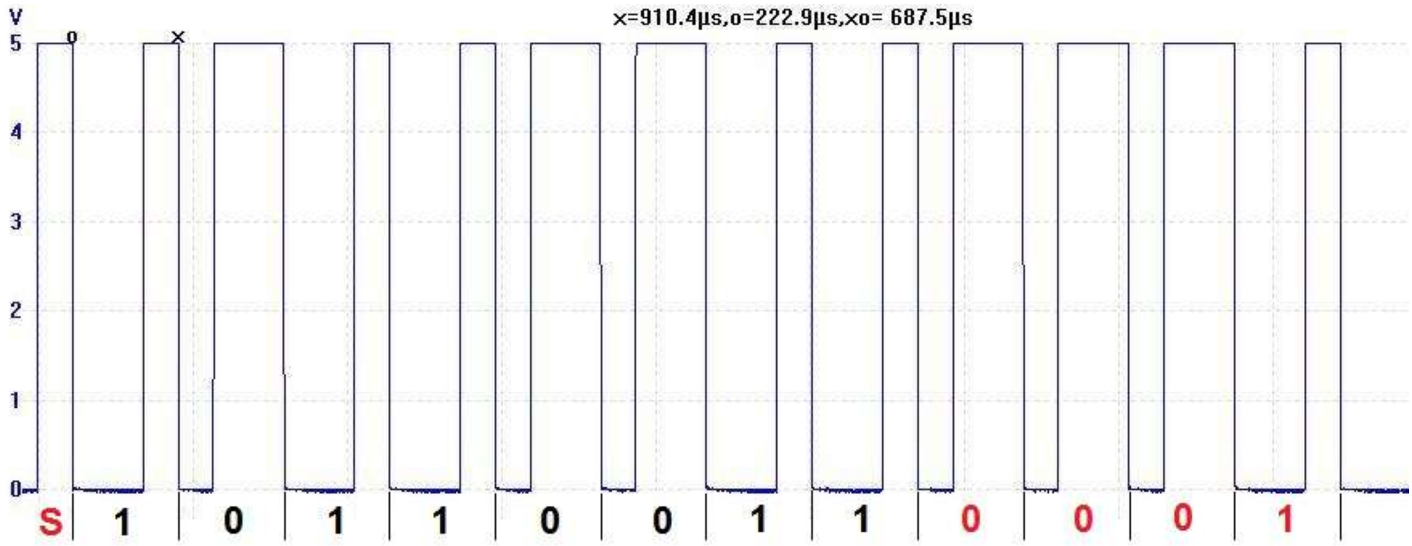
The second diagram is the same, however in this case it shows that if TE is held low then the HT-12E continues repeats sending the word until it goes high again(however it will always send at least 4 words)

These diagrams represent the flow of the process from the micro to the HT-12E and the HT-12E to the transmitter. We are not looking at what comes out of the transmitter.

The datasheet gets a little confusing and isn't clear about he data word structure for the two devices so an oscilloscope was use to capture the transmission sequence on Dout from the HT-12E. The time in millisecs is shown on the X axis, it can be seen that the whole sequence of 4 data words took almost 60mS to send. (Why does it send the data word 4 times?)



Here is one data word, a data 'word' is 13 bits of data from the oscilloscope.
A single start bit, then the 8 address bits (10110011) then the 4 data bits (0001).



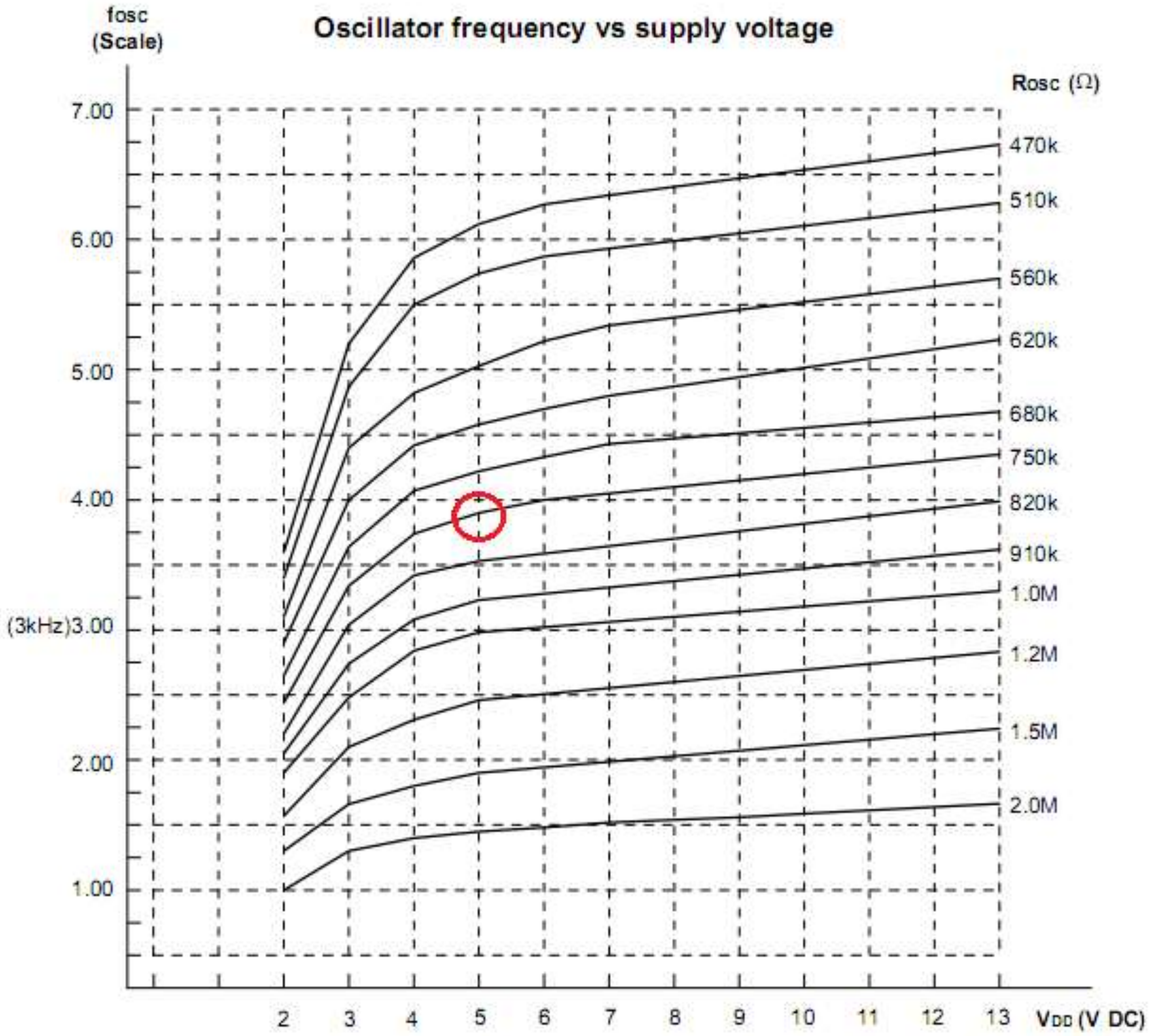
Each full bit includes a period of low and a period of high time and lasts for 687.5uSecs (the difference in time between the o and the x on the scope display)

Other measurements were taken and a single pulse was measured as 229 uSecs in duration and a double pulse was measured as 458 uSecs, with the whole word taking about 8.5mSecs to transmit.

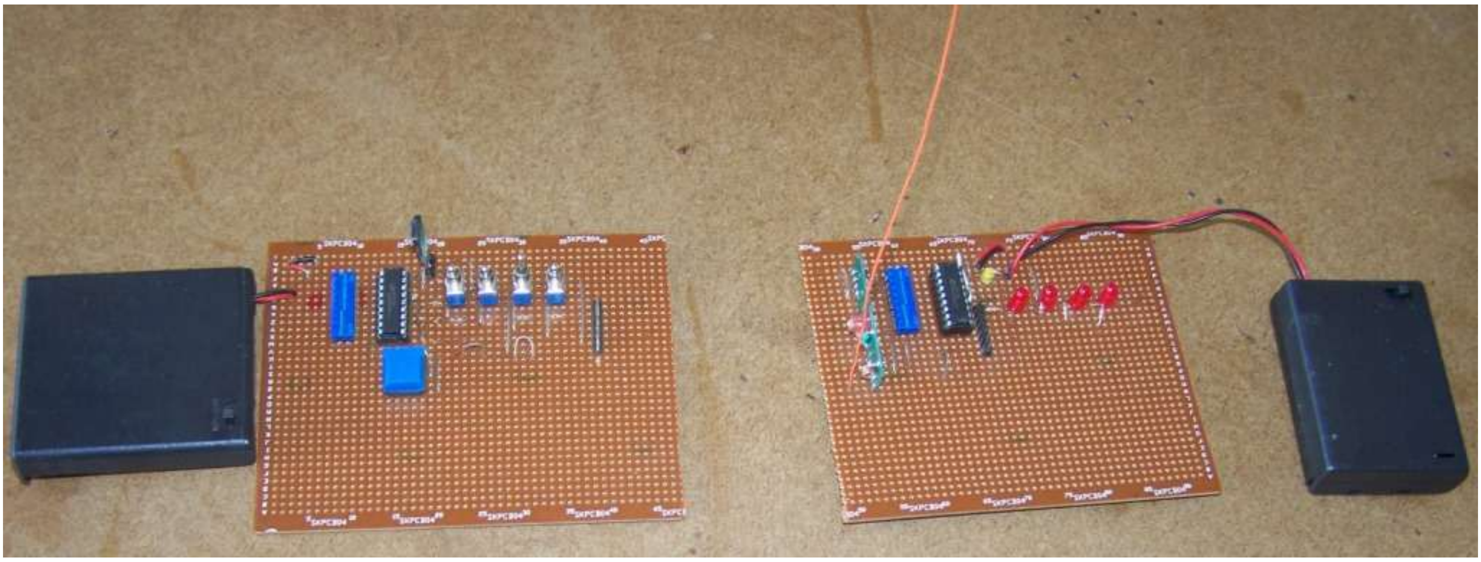
These rates are all determined by the value of R connected to the HT-12E, which in our case is 750K.

This graph from the datasheet shows how the frequency of the oscillator relates to the supply voltage and resistor value. The 750k resistor at 5V will make the oscillator run at about 3.9Khz.
A 3.9Khz wave form has a period of 0.256mSecs (256uSecs).

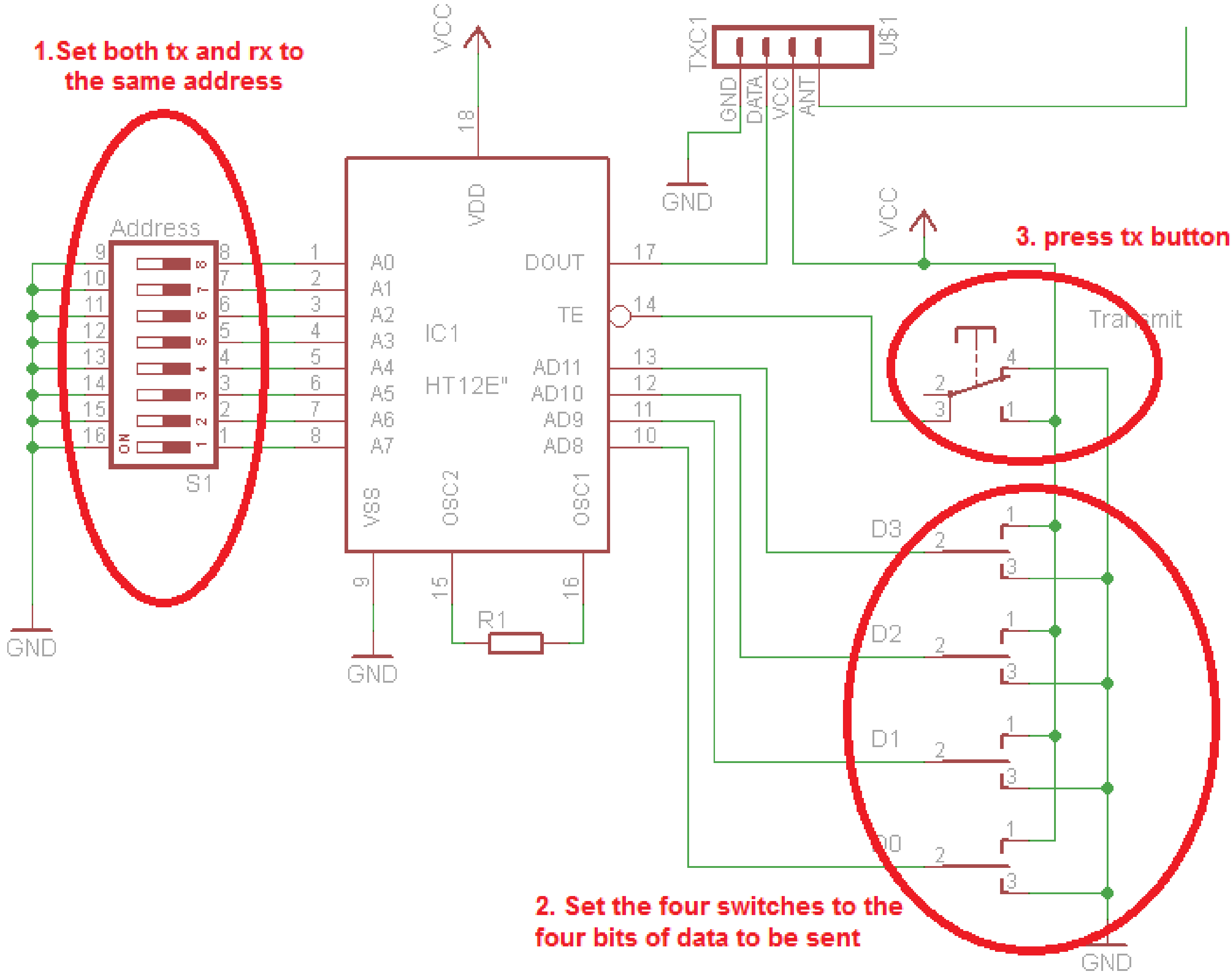
The measured value was 228uSecs which is a 4.4kHz It doesn't quite match, its about 10% off. This could be due to variation in temperature, voltage, resistance or even inside the IC.

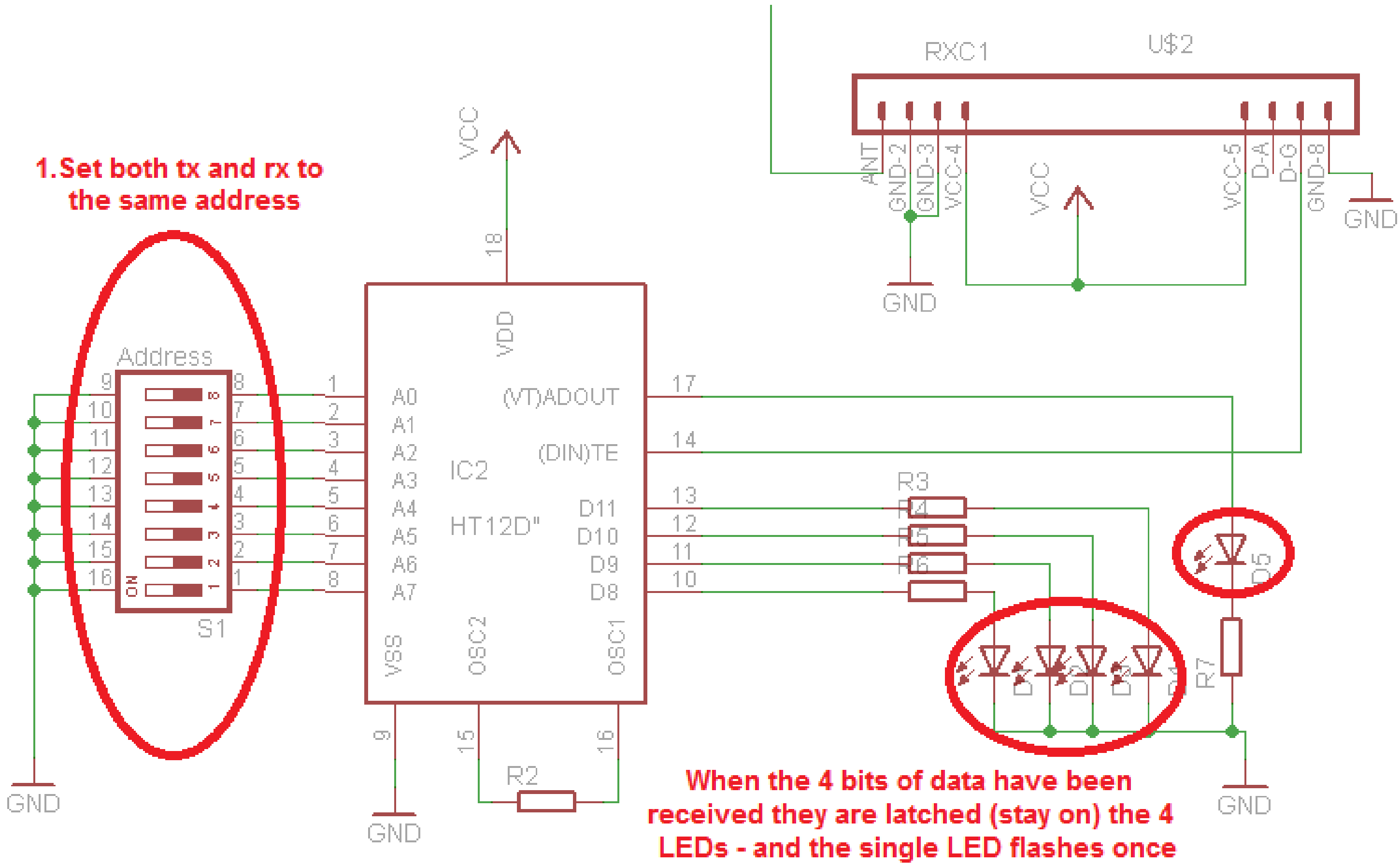


41.3 HT12 test setup



The above 2 boards have been setup in the classroom to test the system. The transmitter is on the left, the schematics for these are:




1.Set both tx and rx to the same address
Address
S1
VCC
IC2
HT12D"
A0
A1
A2
A3
A4
A5
A6
A7
(VT)ADOUT
(DIN)TE
D11
D10
D9
D8
VDD
VSS
OSC2
OSC1
R2
GND
RXC1
U$2
ANT
GND-2
GND-3
VCC-4
VCC-5
D-A
D-G
GND-8
R3
R4
R5
R6
R7
D5
When the 4 bits of data have been received they are latched (stay on) the 4 LEDs - and the single LED flashes once

41.4 HT12E Program

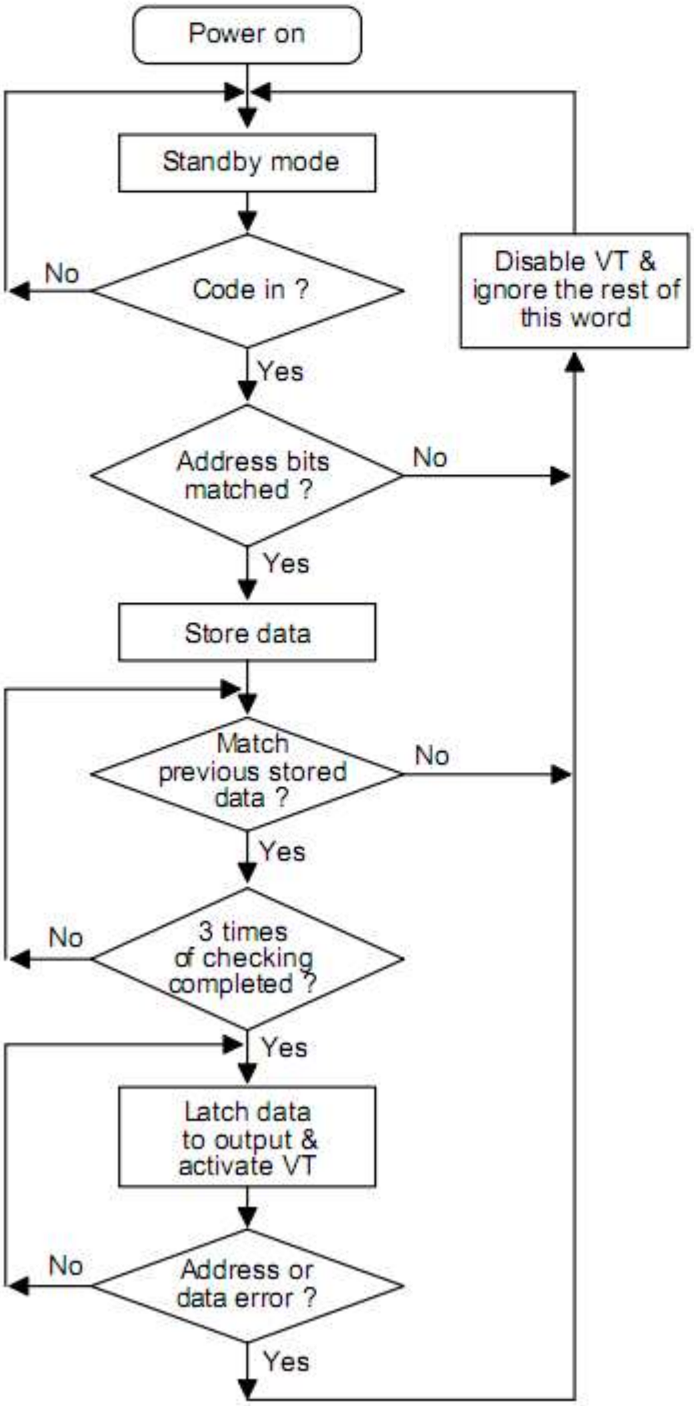
Writing a program to send data usingthe HT12E is straight forward because the IC hides all the complexity from us and we don't have to worry about what it is actually doing. Here is a program that sends the numbers 0 to15 continuously to the transmitter, at 2 second intervals.

```
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                    ' internal clock
$regfile = "m16def.dat"
'------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output

' Hardware Aliases
Ht12e_te Alias Portd.2
Te_led Alias Portc.0

' initialise ports so hardware starts correctly Porta = &H00
Portc = &HFF           ' Turn Off Led's on Portc.0 and  PORTC.0.1
Portd = &HFF           ' Ensure encoder is not transmitting
'------------------------------------------------------------------
' Declare Variables
Dim I As Byte

' Initialise Variables
'------------------------------------------------------------------
' Program starts here
' Transmit the values 0 to 15 then repeat
Do
   For I = 0 To 15
      Portb = I        ' Put the value into the encoder via PortB
      Gosub Transmit   ' Allow the data to be transmitted
      Waitms 2000      'some Delay is for necessary testing.
                       ' without effecting transmission reliabilty
   Next I
Loop

'------------------------------------------------------------------
' Subroutines
Transmit:
   Set Ht12e_te        ' Enable transmission of 4bits from PortB
   Set Te_led          ' Turn on Transmission indicator
   Waitms 5            ' Need a short delay for HT12E
   Reset Ht12e_te      ' Stop the encoding and transmission of data
   Waitms 60           ' Need to see LED and wait till transmission completed
   Reset Te_led
Return
```

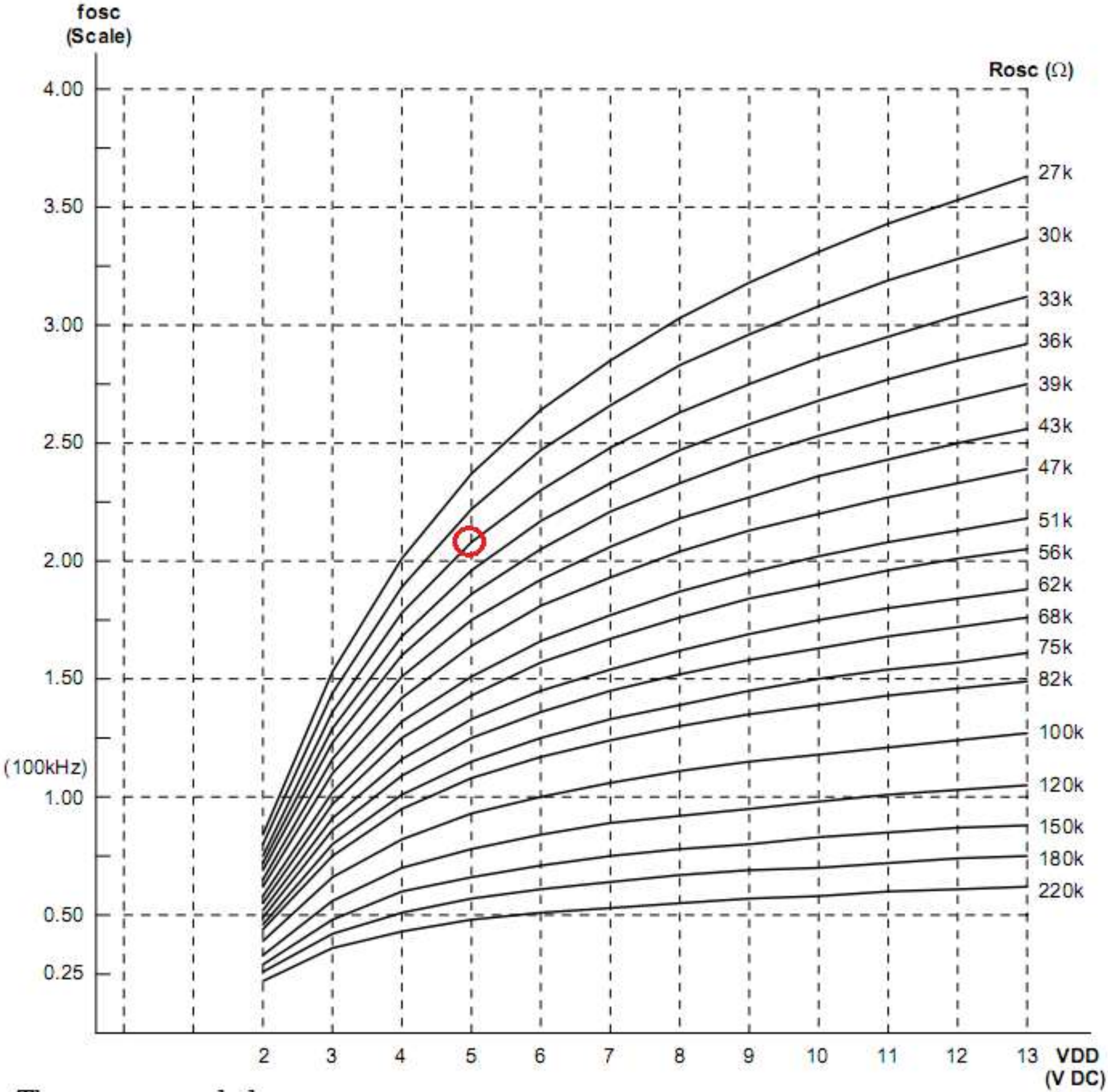
41.5 HT12D datasheet

The matching part for the HT12E is the HT12D. The HT12D decodes the data from the receiver, if it receives the same message 3 times in a row it will put the 4 bits of data onto the 4 data pins and then put the VT (valid transmission) pin high for a short period. Note that the encoder repeats the data 4 times, this allows for some error, this repeating or sending duplicate data is called redundancy.

The flowchart from the datasheet explains the process.



The graph from the datasheet shows that a 33k resistor at 5V will oscillate at 210kHz. The datasheet states that the decoder oscillator must be about 50 times that of the encoder oscillator.



The recommended oscillator frequency is f_{OSCD} (decoder) $\cong$ 50 f_{OSCE} (HT12E encoder)

41.6 HT12D Program

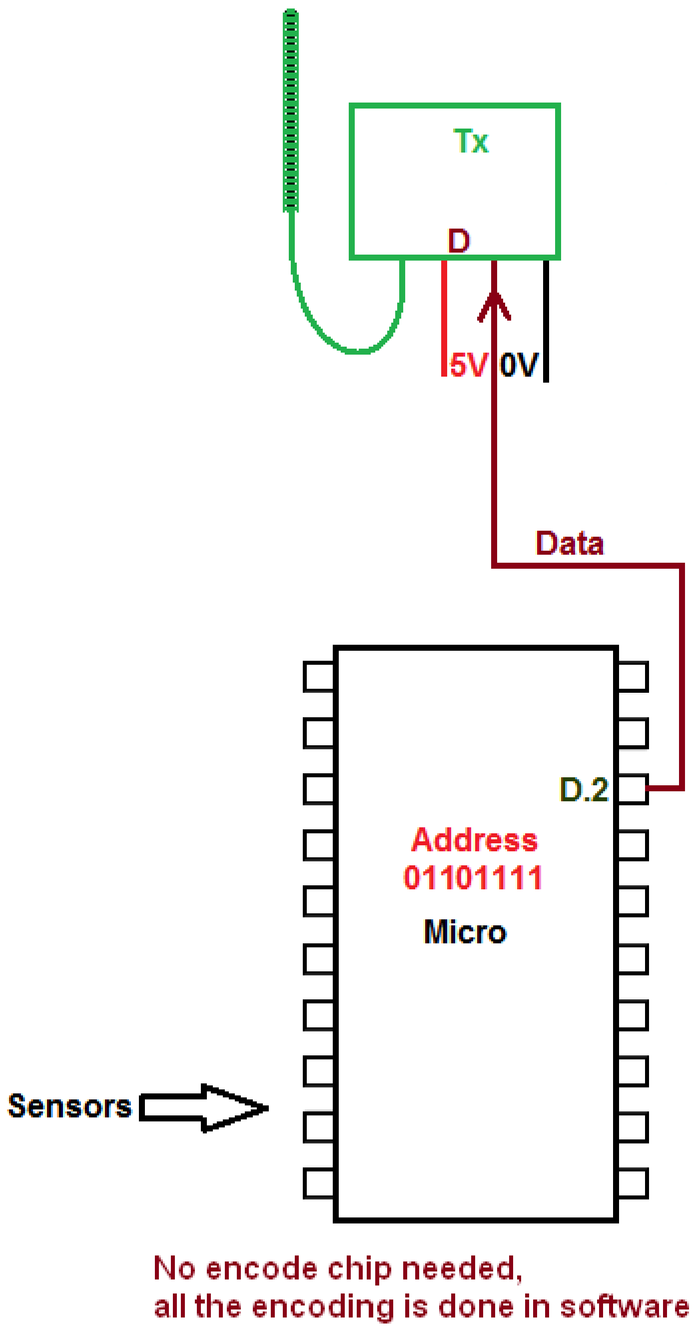
Writing a program to receive data is not hard as the HT12D takes care of the difficult details and signals us when valid data has arrived via the VT pin.

```
'-------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                       ' internal clock
$regfile = "m16def.dat"                  ' ATMEGA16
'-------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                    '4 leds on PortA.0to A.3
Config Portb = Input                     ' Valid data is input on this port
Config Portc = Output                    ' Used for LED's and LCD
Config Portd = Input                     ' PortD.2 is used for Data Valid
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 16 * 2
'  Hardware Aliases
Ht12d_dv Alias Pind.2
'-------------------------------------------------------------------
' Declare Constants
Const True = 1
Const False = 0
' Declare Variables
Dim Rcvd_value As Byte
'  Initialise Variables
'-------------------------------------------------------------------
'  Program starts here
Cls
Cursor Off
Locate 1 , 1
Lcd "HT12D test program"
Do

   If Ht12d_dv = True Then               ' If signal present
      Gosub Get_data                     ' Wait until a valid value
      Porta = Not Rcvd_value             'display on leds - inverse
      Locate 2 , 1
      Lcd "Rcvd Value = "
      Lcd Rcvd_value ; " "               ' display value
   End If
Loop                                     '
End                                      'end program
'-------------------------------------------------------------------
Get_data:
   Rcvd_value = Pinb And &H0F            ' get value from lower nibble PortB
   While Ht12d_dv = True                 ' wait until data no longer valid
   Wend
Return
```

The difficult part of the previous program is integrating it into a larger program where more things are happening, the trouble is that we often don’t want to check if something has happened (polling) we want to be told when it has happened (interrupted).

In a larger program it would make sense then to use one of the AVR’s hardware interrupt, this is covered further on after the topic ofinterrupts has been introduced.

41.7 Replacing the HT12E encoding with software



The HT12E is not that complex (the HT12D is), it can easily be replaced with a program as in this code below. The program continuously sends the numbers 0 to 15 as data to a fixed address &B01101111. The code is in the subroutine transmit:
It sends the start bit, then 8 bits if address then 4 bits of data.

The code is easily implemented using for-next loops , within the loop it checks each bit to see if it is a 1 or 0. To do this it uses the code **If Addr.i = 1 Then** ...

The loop goes from 7 down to 0, if the address is &B01101111 then as i changes the code addr.i selects each bit of the address. This is similar to addressing port pins e.g. portd.7 or portd.0

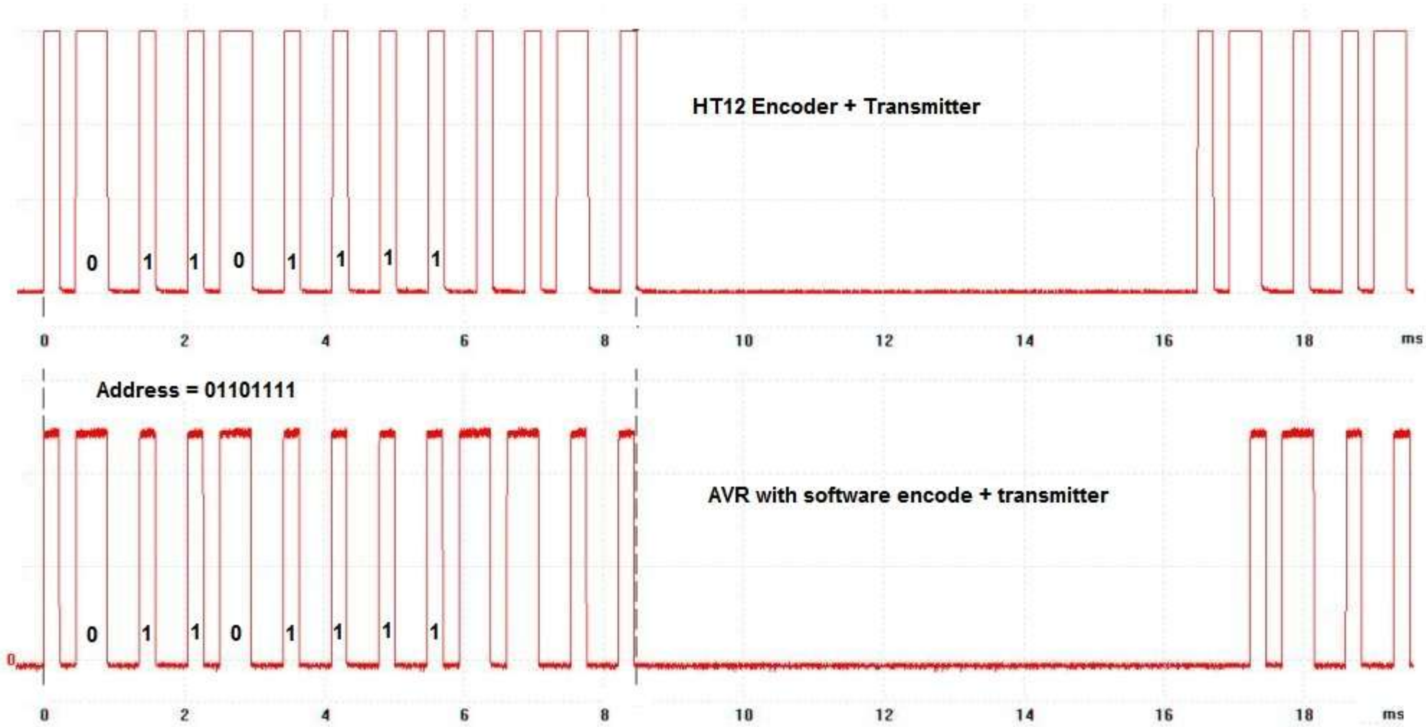
i	Addr.i
7	0
6	1
5	1
4	0
3	1
2	1
1	1
0	1

```
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our
hardware)
$crystal = 8000000                      ' internal clock
$regfile = "m32def.dat"
'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output
' Hardware Aliases
Tx_data Alias Portd.2
Tx_led Alias Portc.0
' initialise ports so hardware starts correctly Porta = &H00
Set Tx_data
Set Tx_led
Set Portc.1
'--------------------------------------------------------------------
' Declare Constants
Const Tx_del = 230                      'micro seconds
' Declare Variables
Dim I As Byte                           'temporary variable
Dim J As Byte                           'temporary variable
Dim Addr As Byte
Dim Dat As Byte                         'the 4 bits of data to send
' Initialise Variables
Addr = &B01101111                        'the address for this system
'--------------------------------------------------------------------
' Program starts here
' the main program is just a test routine to test the subroutine
'   that does the actual work
' Continuously transmit the values 0 to 15

For I = 1 To 4                          ' toggle the LED on and off a few
times
   Toggle Tx_led                        ' to show the PCB is working
Waitms 500
Next
Do
   For Dat = 0 To 15                    ' test code
      Gosub Transmit                    ' Allow the data to be transmitted
      Waitms 2000                       ' Delay is for visual testing.
   Next I
Loop
'--------------------------------------------------------------------
' Subroutines
Transmit:
```

```
  Reset Tx_led                            'light tx LED
  For J = 1 To 6                          'send full word 6 times
     'send the start bit first
     Set Tx_data                          'carrier on
     Waitus Tx_del                        'start bit time
     Reset Tx_data                        'carrier off
     'send the address
     For I = 7 To 0 Step -1               'send most significant bit(7) first
        Waitus Tx_del                     'start with 1 period of no carrier
        If Addr.i = 1 Then
           Waitus Tx_del                  'extra low time for 1
           Set Tx_data                    'carrier on
           Waitus Tx_del
           Reset Tx_data                  'carrier off
        Else
           Set Tx_data                    'carrier on
           Waitus Tx_del                  'extra carrier on time for 1
           Waitus Tx_del
           Reset Tx_data                  'carrier off
        End If
     Next
     'send the data
     For I = 3 To 0 Step -1               'send most significant bit(3) first
        Waitus Tx_del                     'start with 1 period of no carrier
        If Dat.i = 1 Then
           Waitus Tx_del                  'extra low time for 1
           Set Tx_data                    'carrier on
           Waitus Tx_del
           Reset Tx_data                  'off
        Else
           Set Tx_data                    'on
           Waitus Tx_del                  'extra carrier on time for 0
           Waitus Tx_del
           Reset Tx_data                  'off
        End If
        Reset Tx_led
     Next
     Waitus 9000                          'pause between words
  Next
  Set Tx_led                              'TX LED off
Return
```

Here are two screen shots from the oscilloscope the timing in each is almost identical apart from the delay between datawords. This time period could be reduced from 9000uS to 8000uS to match the HT12E.

It should be noted that although the HT12E sends the data word 4 times, we found it necessary to send the data word at least 6 times to get a reliable transmission.

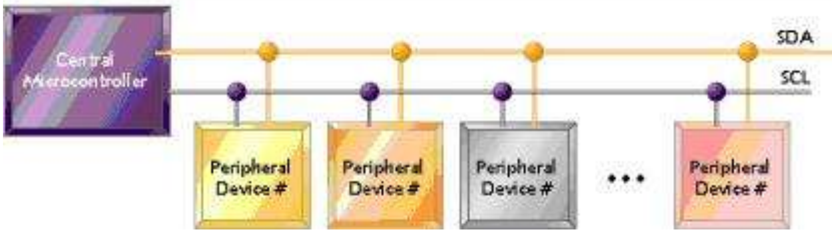

HT12 Encoder + Transmitter
0 1 1 0 1 1 1 1
0 2 4 6 8 10 12 14 16 18 ms
Address = 01101111
AVR with software encode + transmitter
0 1 1 0 1 1 1 1
0
0 2 4 6 8 10 12 14 16 18 ms

42 Introduction to I2C

The Inter-IC bus (I2C pronounced "eye-squared-see") was developed by Philips to communicate between devices in their TV sets. It is now popular and is often used when short distance communications is needed. It is normally used within equipment to communicate between pcb's, e.g. main boards and display boards rather than externally to other equipment.

It is a half duplex synchronous protocol, which means that only one end of the link can talk at once (half duplex) and that there are separate data and clock lines (synchronous). The real strength of this protocol is that many devices can share the bus which reduces the number of I/O lines needed on microcontrollers,it increases the number of devices 1 micro can interface to and several manufacturers now make I2C devices.

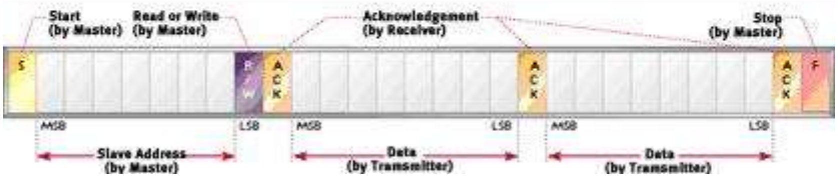
Figure 1: I^2C has two lines in total



The two lines are SDA - Serial data and SCL - Serial Clock
Communication

The system of communications is not too difficult to follow, the first event is when the master issues a start pulse causing all slaves to wake up and listen. the master then sends a 7 bit address which corresponds to one of the slaves on the bus. Then one more bit is sent that tells the slave whether it is going to be receiving or sending information. This is then followed by an ACK bit (acknowledge) issued by the receiver, saying it got the message. Data is then sent over the bus by the transmitter.

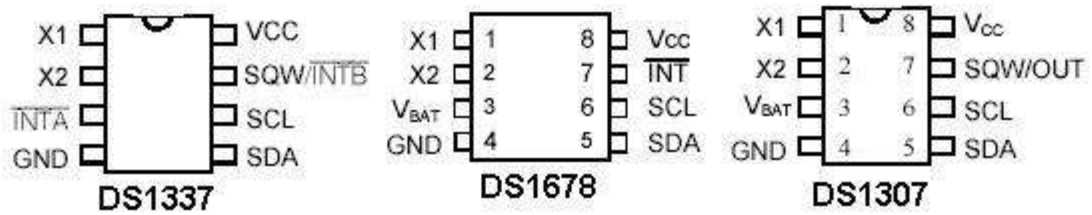
Figure 2: I^2C communication



The I2C protocol is not too hard to generate using software; Bascom comes with the software already built in making I2C very easy to use.

42.1 I2C Real Time Clocks

These are fantastic devices that connect to the microcontroller and keep the time for you. Some common devices are the DS1337, DS1678 and DS1307.

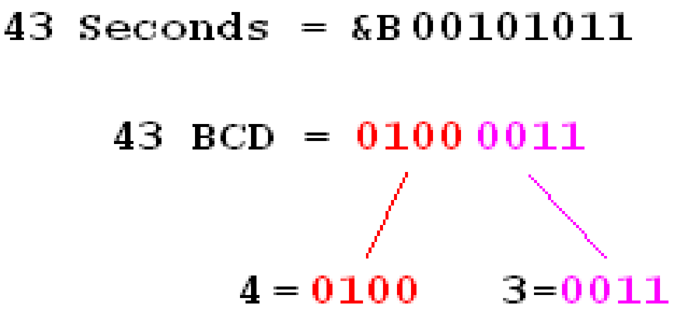


All three require an external 32.768KHz crystal connected to X1 and X2, 5Volts from your circuit connected to Vcc, a ground connection (OV) and connection of two interface pins to the microcontroller, SCL (serial clock) and SDA (serial data).

The DS1678 and DS1307 can have a 3V battery connected to them as backups to keep the RTC time going even though the circuit is powered down. This will last for a couple of years and note that it is not rechargeable. There are datasheets on www.maxim-ic.com website for each of these components as well as many other interesting datasheets on topics such as battery backup. Each of these devices has other unique features that can be explored once the basic time functions are operational.

In these RTCs the registers are split into BCD digits. What this means is that instead of storing seconds as one variable it splits the variable into two parts the units value and the tens value.

register 0	Tens of seconds	Units of seconds
register 1	Tens of minutes	Units of minutes
register 2	Tens of hours	Units of hours
register 3	Tens of hours	Units of hours
register ..	Tens of ...	Units of ...



When we want to put the variable onto an LCD we cannot write lcd seconds as the number would not be correct. We must first convert the BCD to true binary using

Seconds = Makedec(seconds).
LCD Seconds

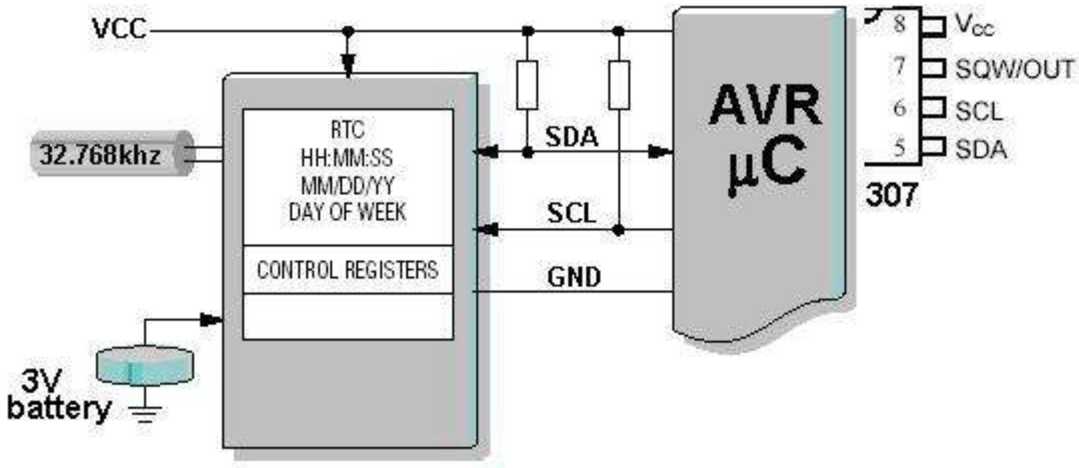
The oppositeneeds to happen when writing to the time registers, we must convert the binary to bcd.

Temp = Makebcd(seconds)
I2cwbyte Temp

42.2 Real time clocks

These devices are very common in microcontroller products such as microwave ovens, cellular phones, wrist watches, industrial process controllers etc.

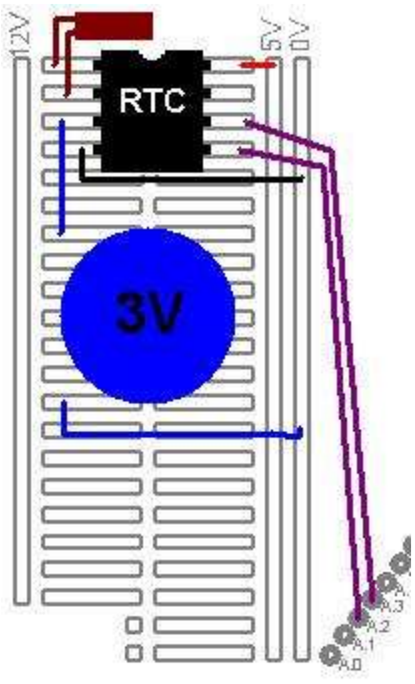
42.3 Connecting the RTC



The crystal for the RTC is a 32.768khz crystal. The reason for the strange number is that 32768 is a multiple of 2, so all that is needed to obtain 1 second pulses is to divide the frequency by two 15 times to get exactly 1 second pulses.

32768
/2 = 16384, /2 = 8192, /2 = 4096, /2 = 2048….2 = 8, /2 = 4, /2 = 2, /2 = 1

42.4 Connecting the RTC to the board



Take special note about bending the leads and soldering to avoid damage to the crystal. Also fix the crystal to the board somehow to reduce strain on the leads.

The I2C lines SDA and SCL require pull up resistors of 4k7 each to 5V.

The battery is a 3V lithium cell, connect it between 0V and the battery pin of the RTC. If a battery is not used then the battery backup pin probably needs connecting to 0V, but check the datasheet first.

42.5 Internal features

First open the datasheet for the DS1307 RTC

There is a memory within the RTC, firstly all the time and dates are stored individually. The units and the 10s of each number are stored separately.

Here is the layout of the memory within the RTC

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	10 Seconds			Seconds			
01	0	10 Minutes			Minutes			
02	0	12/24	AM/PM 10Hr	10Hr	Hour			
03	0	0	0	0		Day of week		
04	0	0	10 Date		Date			
05	0	0	0	10 Mo	Month			
06	10 Year				Year			
07	CONTROL							
08 3F	RAM							

The date and time Sunday, 24 September 2007 21:48:00 are stored as this

ADDRESS	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	
00	0				0				Seconds
01	4				8				Minutes
(02	2				1				Hours
03	0				7				(Sunday)
04	2				4				Day
05	0				9				month
06	0				7				Year

When we read the RTC we send a message to it, (SEND DATA FROM ADDRESS 0)
and it sends 0,48,21,07,24,08,7,..
until we tell it to stop sending

42.6 DS1307 RTC code

Here is the process for setting up communication with a DS1307 RTC followed by the code for one connected to an 8535.

Step1: configure the hardware and dimension a variable, temp, to hold the data we want to send to/receive from the 1678. Dimension the variables used to hold the year, month, day, hours, etc. Don't forget to configure all the compiler directives and hardware such as the LCD, thermistor, switches etc.

Step2: setup the control register in the RTC, to specify the unique functions we require the 1307 to carry out. This is only ever sent once to the 1307.

Step 3: write a number of subroutines that handle the actual communication with the control and status registers inside the 1307. These routines make use of the Bascom functions for I2C communication.

Step 4: write a subroutine that gets the time, hours, date, etc from the 1307.

step 5 : write a subroutine that sets the time, hours, date, etc from the 1307.

step 6: write a program that incorporates these features and puts the time on an LCD.

```
'------------------------------------------------------------------
' Title Block
' Author:   B.Collis
' Date:     26 Mar 2005
' File Name: 1307_Ver4.bas
'------------------------------------------------------------------
' Program Description:
' use an LCD to display the time
' has subroutines to start clock,write time/date to the rtc,
' read date/time from the rtc, setup the SQW pin at 1Hz
'added subroutines to read and write to ram locations
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' DS1307 SDA=porta.2 SDC=porta.3
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                          'the crystal we are using
$regfile = "m32def.dat"                     'the micro we are using
'------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                       '
Config Portb = Output                       '
Config Portc = Output                       '
Config Portd = Output                       '
' config 2 wire I2C interface
'Config I2cdelay = 5                         ' default slow mode
Config Sda = Porta.2
Config Scl = Porta.3
'Config lcd
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 16 * 2                         'configure lcd screen
```

```
'Hardware Aliases

'Initialise ports so harware starts correctly
Cls                                         'clears LCD display
Cursor Off                                  'no cursor

'-------------------------------------------------------------------
' Declare Constants

'-------------------------------------------------------------------
' Declare Variables
Dim Temp As Byte
Dim Year As Byte
Dim Month As Byte
Dim Day As Byte
Dim Weekday As Byte
Dim Hours As Byte
Dim Minutes As Byte
Dim Seconds As Byte
Dim Ramlocation As Byte
Dim Ramvalue As Byte
' Initialise Variables
Year = 5
Month = 3
Weekday = 6
Day = 26
Hours = 6
Minutes = 01
Seconds = 0

'-------------------------------------------------------------------
'  Program starts here
Waitms 300
Cls

'these 3 subroutines should be called once and then commented out
'Gosub Start1307clk
'Gosub Write1307ctrl
'Gosub Write1307time

'Gosub Clear1307ram           'need to use once as initial powerup is
undefined

'Gosub Writeram
'Gosub Readram

'Ramvalue = &HAA
'Call Write1307ram(ramlocation , Ramvalue)
```

```
Do
  Gosub Read1307time                        'read the rtc
  Locate 1 , 1
  Lcd Hours
  Lcd ":"
  Lcd Minutes
  Lcd ":"
  Lcd Seconds
  Lcd "            "
  Lowerline
  Lcd Weekday
  Lcd ":"
  Lcd Day
  Lcd ":"
  Lcd Month
  Lcd ":"
  Lcd Year
  Lcd "            "
  Waitms 200
Loop



End                                         'end program
'----------------------------------------------------------------
'  Subroutines
Read1307time:                               'RTC Real Time Clock
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte 0                               'address to start sending from
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                      'device code (reading)
   I2crbyte Seconds , Ack
   I2crbyte Minutes , Ack
   I2crbyte Hours , Ack
   I2crbyte Weekday , Ack
   I2crbyte Day , Ack
   I2crbyte Month , Ack
   I2crbyte Year , Nack
   Seconds = Makedec(seconds)
   Minutes = Makedec(minutes)
   Hours = Makedec(hours)
   Weekday = Makedec(weekday)
   Day = Makedec(day)
   Month = Makedec(month)
   Year = Makedec(year)
   I2cstop
Return
```

```
'write the time and date to the RTC
Write1307time:
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte &H00                          'send address of first byte to
access
   Temp = Makebcd(seconds)                'seconds
   I2cwbyte Temp
   Temp = Makebcd(minutes)                'minutes
   I2cwbyte Temp
   Temp = Makebcd(hours)                  'hours
   I2cwbyte Temp
   Temp = Makebcd(weekday)                'day of week
   I2cwbyte Temp
   Temp = Makebcd(day)                    'day
   I2cwbyte Temp
   Temp = Makebcd(month)                  'month
   I2cwbyte Temp
   Temp = Makebcd(year)                   'year
   I2cwbyte Temp
   I2cstop
Return


Write1307ctrl:
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte &H07                          'send address of first byte to
access
   I2cwbyte &B10010000                    'start squarewav output 1Hz
   I2cstop
Return


Start1307clk:
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte 0                             'send address of first byte to
access
   I2cwbyte 0                             'enable clock-also sets seconds to 0
   I2cstop
Return


Write1307ram:
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte Ramlocation                   'send address of byte to access
   I2cwbyte Ramvalue                      'send value to store
   I2cstop
Return
```

```
'routine to read the contents of one ram location
'setup ramlocation first and the data will be in ramvalue afterwards
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
Read1307ram:
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte Ramlocation                   'send address of first byte to
access
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                    'device code (reading)
   I2crbyte Ramvalue , Nack
   I2cstop
Return

Clear1307ram:
   Ramvalue = 00
   Ramlocation = &H08
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte Ramlocation                   'send address of byte to access
   For Ramlocation = &H08 To &H3F
       I2cwbyte Ramvalue                  'send value to store
   Next
   I2cstop
Return

Writeram:
   Ramlocation = &H08
   Ramvalue = 111
   Gosub Write1307ram
   Ramlocation = &H09
   Ramvalue = 222
   Gosub Write1307ram
Return

Readram:
   Cls
   Ramlocation = &H08
   Gosub Read1307ram
   Lcd Ramvalue
   Lcd ":"
   Ramlocation = &H09
   Gosub Read1307ram
   Lcd Ramvalue
   Ramlocation = &H0A
   Gosub Read1307ram
   Lcd ":"
   Lcd Ramvalue
   Wait 5
Return

'-------------------------------------------------------------------
' Interrupts
```

42.7 DS1678 RTC code

```
'---------------------------------------------------------------
' 1. Title Block
' Author:   B.Collis
' Date:     10 mar 03
' Version:  1
' File Name: 1678_Ver1.bas
'---------------------------------------------------------------
' 2. Program Description:
' read the time from the RTC
' display it on the LCD
' 3. Hardware Features:
' Dallas DS1678 connected with 32.768khz crystal and battery backup
' SDA on A.2   SCL on A.3
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 5 switches on portB.0, B.1, D.2, D.3, D.6
' 4. Program Features:
'---------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 7372800                'the crystal we are using
$regfile = "m8535.dat"            'the micro we are using
$noramclear                       'so the compiler saves on memory
$lib "mcsbyteint.lbx"
'---------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output             'LEDs on portA
Config Portb = Output             'LEDs on portB
Config Pinb.0 = Input
Config Pinb.1 = Input
Config Portc = Output             'LEDs on portC
Config Portd = Output             'LEDs on portD
Config Pind.2 = Input
Config Pind.3 = Input
Config Pind.6 = Input
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.2 , Rs =
Portc.0
Config Lcd = 40 * 2               'configure lcd screen
Config Sda = Porta.2
Config Scl = Porta.3
' 7. Hardware Aliases
Sw_1 Alias Pinb.0
Sw_2 Alias Pinb.1
Sw_3 Alias Pind.2
Sw_4 Alias Pind.3
Sw_5 Alias Pind.6
Spkr Alias Portd.7                'refer to spkr not PORTd.7
' 8. initialise ports so hardware starts correctly
Porta = &B11110000                'turns off LEDs
Portb = &B11111111                'turns off LEDs
Portc = &B11111111                'turns off LEDs
Portd = &B11111111                'turns off LEDs
Reset Spkr
```

```
Cls
Cursor Off
'-----------------------------------------------------------------
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Temp As Byte
Dim Century As Byte
Dim Year As Byte
Dim Month As Byte
Dim Day As Byte
Dim _Dayofweek As Byte
Dim Hours As Byte
Dim Minutes As Byte
Dim Seconds As Byte
Dim Control As Byte                    'the control byte for the DS1678
' 11. Initialise Variables
'-----------------------------------------------------------------
' 12. Program starts here
Locate 1 , 1
Lcd "IT'S TIME"
Do
 'Debounce Sw_1 , 0 , Startrtc , Sub
 Gosub Displaytimedate
Loop
End                                    'end program
'-----------------------------------------------------------------
' 13. Subroutines
Displaytimedate:
  Locate 2 , 1
  Gosub Read1678time                   'read the rtc ds1678
  Select Case _Dayofweek
    Case 1 : Lcd "Mon"
    Case 2 : Lcd "Tue"
    Case 3 : Lcd "Wed"
    Case 4 : Lcd "Thu"
    Case 5 : Lcd "Fri"
    Case 6 : Lcd "Sat"
    Case 7 : Lcd "Sun"
  End Select
  Lcd " "
  Select Case Month
    Case 1 : Lcd "Jan"
    Case 2 : Lcd "Feb"
    Case 3 : Lcd "Mar"
    Case 4 : Lcd "Apr"
    Case 5 : Lcd "May"
    Case 6 : Lcd "Jun"
    Case 7 : Lcd "Jul"
    Case 8 : Lcd "Aug"
    Case 9 : Lcd "Sep"
    Case 10 : Lcd "Oct"
    Case 11 : Lcd "Nov"
    Case 12 : Lcd "Dec"
```

```
  End Select
  Lcd " "
  Lcd Day
  Lcd " "
  Lcd Century
  If Year < 10 Then Lcd "0"
  Lcd Year
  Lcd "  "
  If Hours < 10 Then Lcd "0"
  Lcd Hours
  Lcd ":"
  If Minutes < 10 Then Lcd "0"
  Lcd Minutes
  Lcd ":"
  If Seconds < 10 Then Lcd "0"
  Lcd Seconds
  Lcd "          "
Return

'read time and date from 1678
Read1678time:                         'RTC Real Time Clock
  I2cstart
  I2cwbyte &B10010100                     'send device code (writing)
  I2cwbyte &H00                       'send address of first byte to access
  I2cstop

  I2cstart
  I2cwbyte &B10010101                     'send device code (reading data)
  I2crbyte Seconds , Ack
  I2crbyte Minutes , Ack
  I2crbyte Hours , Ack
  I2crbyte _Dayofweek , Ack
  I2crbyte Day , Ack
  I2crbyte Month , Ack
  I2crbyte Year , Ack
  I2crbyte Century , Nack
  I2cstop
  Seconds = Makedec(seconds)
  Minutes = Makedec(minutes)
  Hours = Makedec(hours)
  _Dayofweek = Makedec(_dayofweek)
  Day = Makedec(day)
  Month = Makedec(month)
  Year = Makedec(year)
  Century = Makedec(century)
Return

'-----------------------------------------------------------------------------
'write the time and date to the DS1678 RTC
Write1678time:                        'RTC Real Time Clock
  I2cstart
  I2cwbyte &B10010100                     'send device code (writing)
  I2cwbyte &H00                       'send address of first byte to access
  Temp = Makebcd(seconds)                'seconds
```

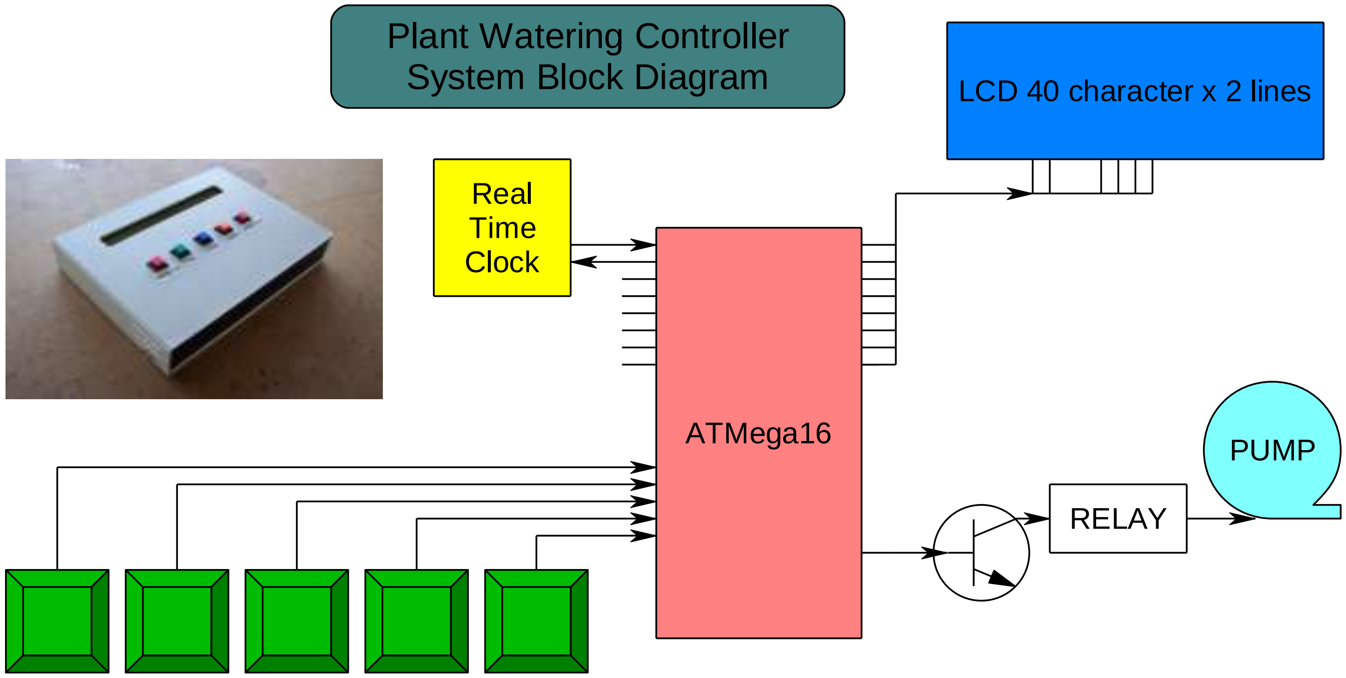
```
  I2cwbyte Temp
  Temp = Makebcd(minutes)           'minutes
  I2cwbyte Temp
  Temp = Makebcd(hours)             'hours
  I2cwbyte Temp
  Temp = Makebcd(_dayofweek)        'day of week
  I2cwbyte Temp
  Temp = Makebcd(day)               'day
  I2cwbyte Temp
  Temp = Makebcd(month)             'month
  I2cwbyte Temp
  Temp = Makebcd(year)              'year
  I2cwbyte Temp
  Temp = Makebcd(century)           'century
  I2cwbyte Temp
  I2cstop
Return

'------------------------------------------------------------------------
'write to the control register in the DS1678 RTC
'Write1678control:'comment out because its used only once at the start
'  I2cstart
'  I2cwbyte &B10010100          'send device code (writing)
'  I2cwbyte &H0E                'send address of first byte to access
'  I2cwbyte Control
'  I2cstop
'Return

'------------------------------------------------------------------------
'read Status Register in DS1678 RTC into Temp register
'Read1678status:
  'send address to read data from
'  I2cstart
'  I2cwbyte &B10010100          'send device code  (writing)
'  I2cwbyte &H0F                'send address of first byte to access
'  I2cstop
  'read data from that address
'  I2cstart
'  I2cwbyte &B10010101          'send device code  (reading)
'  I2crbyte Temp , Nack         'get just the one byte
'  I2cstop
'Return

'------------------------------------------------------------------------
'read Control Register in DS1678 RTC into Temp register
'Read1678control:
  'send address  to read data from
'  I2cstart
'  I2cwbyte &B10010100          'send device code  (writing)
'  I2cwbyte &H0E                'send address of first byte to access
'  I2cstop
  'read data from that address
'  I2cstart
'  I2cwbyte &B10010101          'send device code  (reading)
```

```
'   I2crbyte Temp , Nack            'get just the one byte
'   I2cstop
'   Cls
'   Lcd Temp
'   Wait 10
'Return

'-----------------------------------------------------------------------
'Startrtc:
'   Cls
'   Wait 1
'   Control = &B00000111
   'me=0
   'clr=0               clear the RTC memory
   'dis1=0 dis0=0
   'ro=0 '
   'tr1=1 tr0=1 '
   'ce=1                RTC clock on
'   Gosub Write1678control
'   Lcd "written control"
'   Wait 1
'   Century = 20
'   Year = 03
'   Month = 8
'   Day = 24
'   _Dayofweek = 7
'   Hours = 16
'   Minutes = 44
'   Seconds = 50
'   Gosub Write1678time
'   Cls
'   Lcd "written time"
'   Wait 1
'Return
```

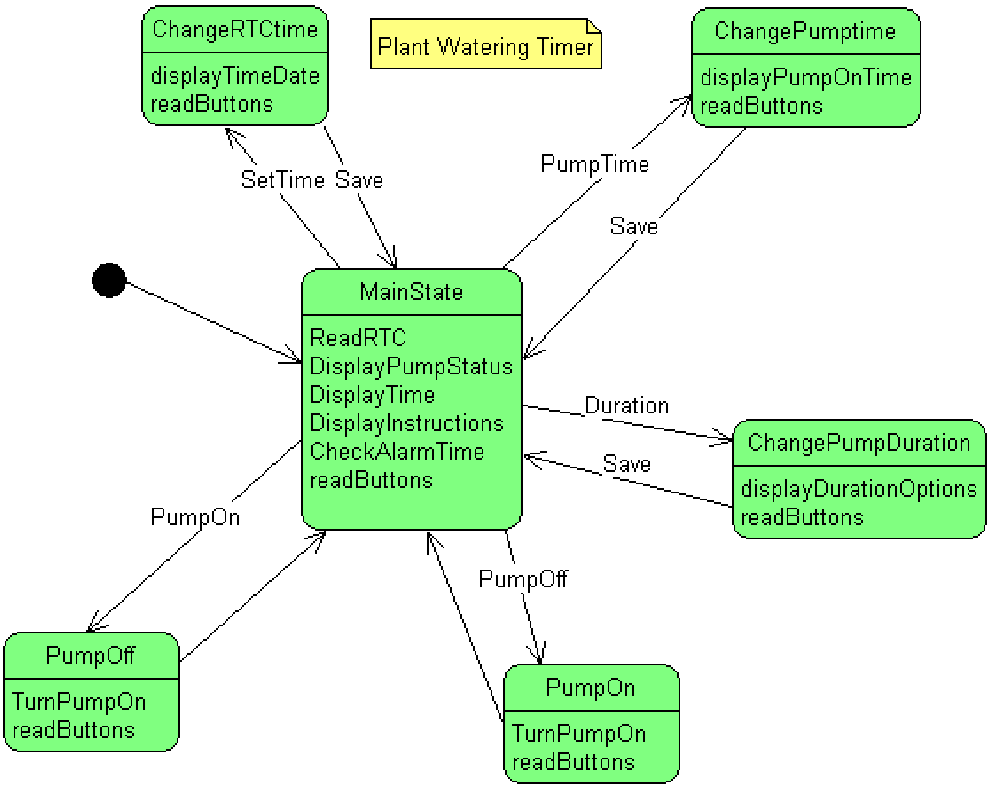
43 Plant watering timer student project

A client needed a system to control a small pump for an indoor garden, here is Ishan's project.

43.1 System block diagram



43.2 State machine



43.3 Program code

```
'-----------------------------------------------------------------
' 1. Title Block
' Plant WateringTimer v0.10
' Ishan 2006
'-----------------------------------------------------------------
' 2. Program Description:
' statemachine implementation for pump timer
' read the time from the RTC
' display it on the LCD
' 3. Hardware Features:
' Dallas DS1678 connected with 32.768khz crystal and battery backup
' SDA on A.2   SCL on A.3
' LCD on portc - note the use of 4 bit mode and only 2 control lines
' 5 switches on portB.0, B.1, D.2, D.3, D.6
' 4. Program Features:
'-----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                  'the crystal we are using
$regfile = "m32def.dat"             'the micro we are using
'$noramclear                        'so the compiler saves on memory
'$lib "mcsbyteint.lbx"
'-----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
'Config Porta = Output              'LEDs on portA
Config Portb = Output               'LEDs on portB
Config Pinb.0 = Input               'switch
Config Pinb.1 = Input               'switch
Config Portc = Output               'LEDs on portC
Config Portd = Output               'LEDs on portD
Config Pind.2 = Input               'switch
Config Pind.3 = Input               'switch
Config Pind.6 = Input               'switch
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.1 , Rs =
Portc.0
Config Lcd = 40 * 2                 'configure lcd screen
Config Sda = Porta.2
Config Scl = Porta.3
' 7. Hardware Aliases
Sw_5 Alias Pinb.0
Sw_4 Alias Pinb.1
Sw_3 Alias Pind.2
Sw_2 Alias Pind.3
Sw_1 Alias Pind.6
Pump Alias Portb.2
' 8. initialise ports so hardware starts correctly, activate pullups on sw's
Porta = &B11110000                  'turns off LEDs
Portb = &B11111111                  'turns off LEDs
Portc = &B11111111                  'turns off LEDs
Portd = &B01111111                  'turns off LEDs
Reset Pump                          ' turn the pump off'
'-----------------------------------------------------------------
```

```
' 9. Declare Constants
Const State_main = 0
Const State_pumpon = 1
Const State_pumpoff = 2
Const State_change_time = 3
Const State_change_pumptime = 4
Const State_change_pumpdur = 5
'-----------------------------------------------------------------
' 10. Declare Variables

Dim Curr_state As Byte              'the state machine variable
Dim Switch As Byte                  'which switch is pressed
Dim Pump_hours As Byte
Dim Pump_mins As Byte
Dim Pump_dur As Byte
Dim Cursor_posn As Byte
Dim Oldseconds As Byte
'RTC variables for a DS1678
Dim Control As Byte
Dim Temp As Byte
Dim Century As Byte
Dim Year As Byte
Dim Month As Byte
Dim Day As Byte
Dim _dayofweek As Byte
Dim Hours As Byte
Dim Minutes As Byte
Dim Seconds As Byte

' 11. Initialise Variables
Curr_state = State_main             'begin here
Cursor_posn = 1
Century = 20
'Control = &B00000111               'tell rtc to go on battery
'me=0
'clr=0 clear the RTC memory
'dis1=0 dis0=0
'ro=0 '
'tr1=1 tr0=1 '
'ce=1 RTC clock on



'-----------------------------------------------------------------
' 12. Program starts here
Cls
Cursor Off
Lcd "welcome to the pump controller"
Wait 1
Cls
```

```
'---------------------------------------------------------------
'state machine implementation
Do
  'read switches (common to all states so put here)
  Switch = 0
  Debounce Sw_1 , 0 , S1 , Sub
  Debounce Sw_2 , 0 , S2 , Sub
  Debounce Sw_3 , 0 , S3 , Sub
  Debounce Sw_4 , 0 , S4 , Sub
  Debounce Sw_5 , 0 , S5 , Sub
  'action the current state
  Select Case Curr_state
    Case State_main : Gosub Sub_main
    Case State_pumpon : Gosub Sub_pumpon
    Case State_pumpoff : Gosub Sub_pumpoff
    Case State_change_time : Gosub Sub_change_time
    Case State_change_pumptime : Gosub Sub_change_pumptime
    Case State_change_pumpdur : Gosub Sub_change_pumpdur
  End Select
Loop
End



'---------------------------------------------------------------

'switch routines
S1:
         Switch = 1
Return

S2:
         Switch = 2
Return

S3:
         Switch = 3
Return

S4:
         Switch = 4
Return

S5:
         Switch = 5
Return
```

```
'--------------------------------------------------------------
'individual states' routines

Sub_main:
        'display pump condition
        Locate 1 , 1
        Lcd "pump is "
        If Pump = 0 Then
          Lcd "OFF"
        Else
          Lcd "ON "
        End If

        'get and display the time
        Gosub Read1678time                'read the rtc (ds1678)
        Gosub Displaytime                 'put time on the display
        'display user instructions on second line
        Locate 2 , 1
        Lcd "TurnOn  TurnOff SetTime  PumpTime  Dur"

        'if user has pressed a switch action their choice
        'but prior to changing to the new state setup any parameters
        Select Case Switch
          Case 1 : Cls
                Curr_state = State_pumpon
          Case 2 : Cls
                Curr_state = State_pumpoff
          Case 3 : Cls
                Gosub Displaytimedate    'get current time
                Cursor_posn = 1          'start with known cursor position
                Locate 1 , Cursor_posn   'tell display to start there
                Cursor On Blink          'let the user see the cursor
                Curr_state = State_change_time
          Case 4 : Cls
                Curr_state = State_change_pumptime
          Case 5 : Cls
                Curr_state = State_change_pumpdur
        End Select

        'see if it is time to turn pump on/off
        Gosub Check_pumptime
Return

Sub_pumpon:
        Set Pump
        Curr_state = State_main
Return
```

```
Sub_pumpoff:
          Reset Pump
          Curr_state = State_main
Return

Sub_change_pumptime:
          'display the time and instructions
          Locate 1 , 1
          Lcd " pump will go on at "
          Locate 1 , 21
          If Pump_hours < 10 Then Lcd "0"
          Lcd Pump_hours
          Lcd ":"
          If Pump_mins < 10 Then Lcd "0"
          Lcd Pump_mins
          'display switch actions
          Locate 2 , 1
          Lcd "   -hr    +hr    -min    +min   save"
          'action any switch press
          If Switch = 1 Then Gosub Decr_hours
          If Switch = 2 Then Gosub Incr_hours
          If Switch = 3 Then Gosub Decr_mins
          If Switch = 4 Then Gosub Incr_mins
          If Switch = 5 Then Gosub Save_pumptime
          'if the max pump duration is 25 then
          'it makes sense not to have the time cross midnight
          'so make sure pump time is not greater than 11:30pm
          If Pump_hours = 23 Then
            If Pump_mins > 30 Then Pump_mins = 30
          End If
Return

Sub_change_time:
          Locate 2 , 1
          Lcd "   left   right   decr   incr   save"
          If Switch = 1 Then Gosub Cursor_left
          If Switch = 2 Then Gosub Cursor_right
          If Switch = 3 Then Gosub Decrement
          If Switch = 4 Then Gosub Increment
          If Switch = 5 Then Gosub Save_time
Return

Sub_change_pumpdur:
          Locate 2 , 1
          Lcd "   5min  10min  15min  20min  25min"
          Select Case Switch
                    Case 1 : Pump_dur = 5
                    Case 2 : Pump_dur = 10
                    Case 3 : Pump_dur = 15
                    Case 4 : Pump_dur = 20
                    Case 5 : Pump_dur = 25
          End Select
          If Switch > 0 Then Gosub Save_pumpdur
Return
```

```
'----------------------------------------------------------------
'auxuillary routines
Save_pumpdur:
        Curr_state = State_main
        'save pump_dur
Return

Check_pumptime:
Return


Displaytime:
        Locate 1 , 16
        If Hours < 10 Then Lcd "0"
        Lcd Hours
        Lcd ":"
        If Minutes < 10 Then Lcd "0"
        Lcd Minutes
        Lcd ":"
        If Seconds < 10 Then Lcd "0"
        Lcd Seconds
        Locate 1 , 28
        Lcd Pump_hours
        Lcd ":"
        Lcd Pump_mins
        Locate 1 , 36
        Lcd Pump_dur
        Lcd "min"
Return

Displaytimedate:
        Locate 1 , 1
        Select Case _dayofweek
          Case 1 : Lcd "Mon"
          Case 2 : Lcd "Tue"
          Case 3 : Lcd "Wed"
          Case 4 : Lcd "Thu"
          Case 5 : Lcd "Fri"
          Case 6 : Lcd "Sat"
          Case 7 : Lcd "Sun"
        End Select
        Lcd " "
        Select Case Month
          Case 1 : Lcd "Jan"
          Case 2 : Lcd "Feb"
          Case 3 : Lcd "Mar"
          Case 4 : Lcd "Apr"
          Case 5 : Lcd "May"
          Case 6 : Lcd "Jun"
          Case 7 : Lcd "Jul"
          Case 8 : Lcd "Aug"
          Case 9 : Lcd "Sep"
          Case 10 : Lcd "Oct"
```

```
            Case 11 : Lcd "Nov"
            Case 12 : Lcd "Dec"
        End Select
        Lcd " "
        If Day < 10 Then Lcd "0"             'insert a leading zero
        Lcd Day
        Lcd " "
        Lcd Century
        If Year < 10 Then Lcd "0"
        Lcd Year
        Lcd "  "
        Locate 1 , 17
        If Hours < 10 Then Lcd "0"
        Lcd Hours
        Lcd ":"
        If Minutes < 10 Then Lcd "0"
        Lcd Minutes
        Lcd ":"
        If Seconds < 10 Then Lcd "0"
        Lcd Seconds
Return

'-----------------------------------------------------------------
'the pump on time routines
Incr_hours:
        Incr Pump_hours
        If Pump_hours > 23 Then Pump_hours = 0
Return

Decr_hours:
        Decr Pump_hours
        If Pump_hours > 23 Then Pump_hours = 23
Return

Incr_mins:
        Incr Pump_mins
        If Pump_mins > 59 Then Pump_mins = 0
Return

Decr_mins:
        Decr Pump_mins
        If Pump_mins > 59 Then Pump_mins = 59
Return

Save_pumptime:
'save into eeprom
‘not implemented yet
Return
```

```
'----------------------------------------------------------------
'Time modification routines
Increment:
          Select Case Cursor_posn
              Case 1 : Incr _dayofweek
                    If _dayofweek > 7 Then _dayofweek = 1
              Case 5 : Incr Month
                    If Month > 12 Then Month = 1
              Case 10 : Incr Day
                    If Day > 31 Then Day = 1
              Case 15 : Incr Year
                      If Year > 12 Then Year = 0
              Case 18 : Incr Hours
                    If Hours > 23 Then Hours = 0
              Case 21 : Incr Minutes
                    If Minutes > 59 Then Minutes = 0
              Case 24 : Incr Seconds
                    If Seconds > 59 Then Seconds = 0
             Case Else:
          End Select
          Gosub Displaytimedate
Return

Decrement:
          Select Case Cursor_posn
              Case 1 : Decr _dayofweek
                    If _dayofweek < 1 Then _dayofweek = 7
              Case 5 : Decr Month
                    If Month < 1 Then Month = 12
              Case 10 : Decr Day
                    If Day < 1 Then Day = 31
              Case 15 : Decr Year
                    If Year = 255 Then Year = 0
              Case 18 : Decr Hours
                    If Hours = 255 Then Hours = 23
              Case 21 : Decr Minutes
                    If Minutes = 255 Then Minutes = 59
              Case 24 : Decr Seconds
                    If Seconds = 255 Then Seconds = 59
              Case Else:
          End Select
          Gosub Displaytimedate
Return
```

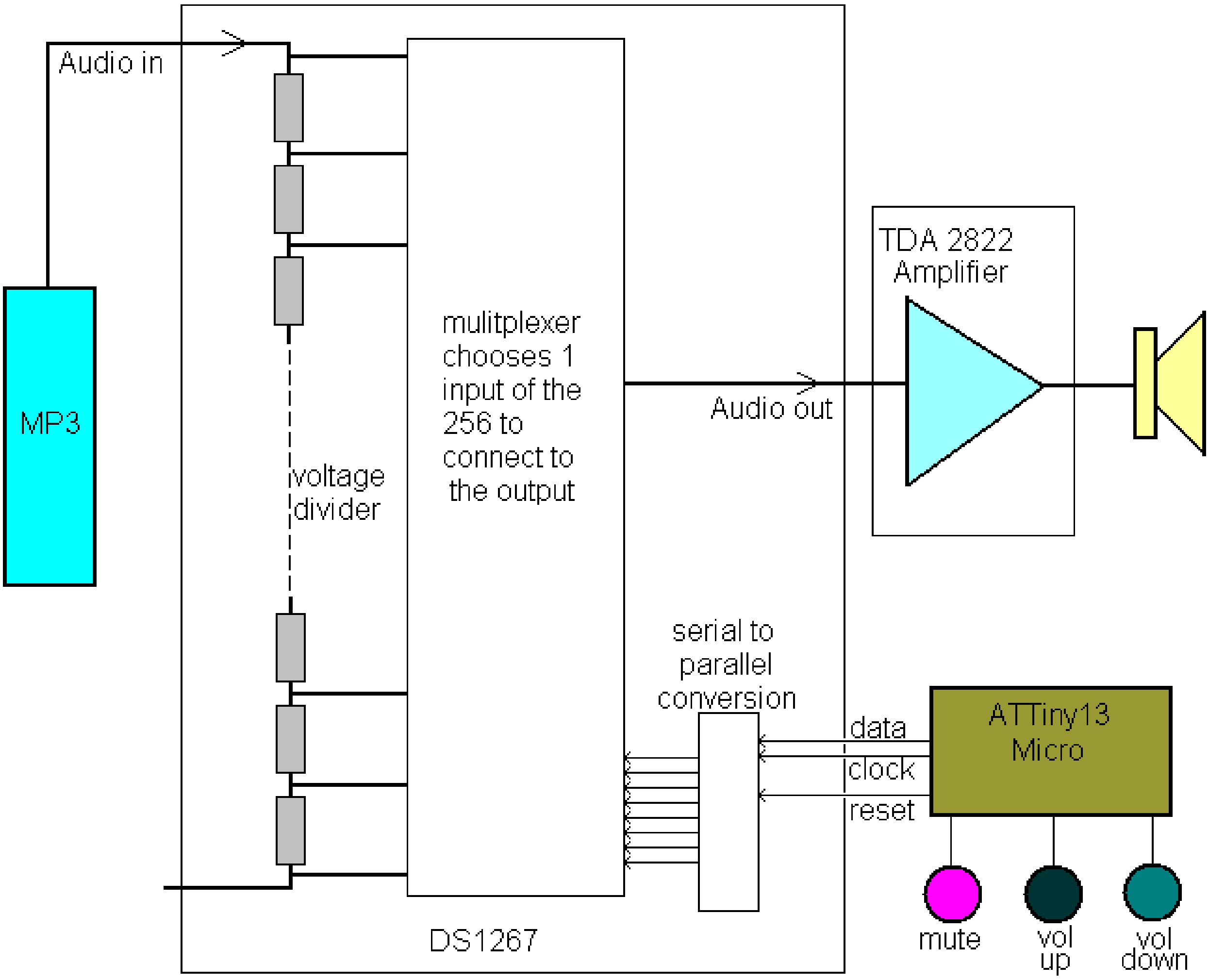
```
Cursor_left:
        Select Case Cursor_posn
                Case 1 : Cursor_posn = 24
                Case 24 : Cursor_posn = 21
                Case 21 : Cursor_posn = 18
                Case 18 : Cursor_posn = 15
                Case 15 : Cursor_posn = 10
                Case 10 : Cursor_posn = 5
                Case 5 : Cursor_posn = 1
        End Select
        Locate 1 , Cursor_posn
Return

Cursor_right:
        Select Case Cursor_posn
                Case 1 : Cursor_posn = 5
                Case 5 : Cursor_posn = 10
                Case 10 : Cursor_posn = 15
                Case 15 : Cursor_posn = 18
                Case 18 : Cursor_posn = 21
                Case 21 : Cursor_posn = 24
                Case 24 : Cursor_posn = 1
        End Select
        Locate 1 , Cursor_posn
Return

Save_time:
        Cursor_posn = 1
        Cls
        Cursor Off
        Gosub Write1678time
        Curr_state = State_main
Return
'-----------------------------------------------------------------
'RTC routines
'read time and date from 1678
Read1678time:                   'RTC Real Time Clock
        I2cstart
        I2cwbyte &B10010100             'send device code (writing)
        I2cwbyte &H00                   'send address of first byte to access
        I2cstop
        I2cstart
        I2cwbyte &B10010101             'send device code (reading data)
        I2crbyte Seconds , Ack
        I2crbyte Minutes , Ack
        I2crbyte Hours , Ack
        I2crbyte _dayofweek , Ack
        I2crbyte Day , Ack
        I2crbyte Month , Ack
        I2crbyte Year , Ack
        I2crbyte Century , Nack
        I2cstop
        Seconds = Makedec(seconds)
        Minutes = Makedec(minutes)
```

```
        Hours = Makedec(hours)
        _dayofweek = Makedec(_dayofweek)
        Day = Makedec(day)
        Month = Makedec(month)
        Year = Makedec(year)
        Century = Makedec(century)
Return
'write the time and date to the DS1678 RTC
Write1678time:                          'RTC Real Time Clock
        I2cstart
        I2cwbyte &B10010100             'send device code (writing)
        I2cwbyte &H00                   'send address of first byte to access
        Temp = Makebcd(seconds)         'seconds
        I2cwbyte Temp
        Temp = Makebcd(minutes)         'minutes
        I2cwbyte Temp
        Temp = Makebcd(hours)           'hours
        I2cwbyte Temp
        Temp = Makebcd(_dayofweek)      'day of week
        I2cwbyte Temp
        Temp = Makebcd(day)             'day
        I2cwbyte Temp
        Temp = Makebcd(month)           'month
        I2cwbyte Temp
        Temp = Makebcd(year)            'year
        I2cwbyte Temp
        Temp = Makebcd(century)         'century
        I2cwbyte Temp
        I2cstop
Return
 'write to the control register in the DS1678 RTC
Write1678control:
        I2cstart
        I2cwbyte &B10010100             'send device code (writing)
        I2cwbyte &H0E                   'send address of first byte to access
        I2cwbyte Control                'control must have COE set to 1 to enable osc
        I2cstop
Return
'read Control Register in DS1678 RTC into Temp register
Read1678control:
        Lcd Control
        Wait 5
        'send address  to read data from
        I2cstart
        I2cwbyte &B10010100             'send device code  (writing)
        I2cwbyte &H0E                   'send address of first byte to access
        I2cstop
        'read data from that address
        I2cstart
        I2cwbyte &B10010101             'send device code  (reading)
        I2crbyte Control , Nack         'get just the one byte
        'I2crbyte Status , Nack         'get just the one byte
        I2cstop
Return
```

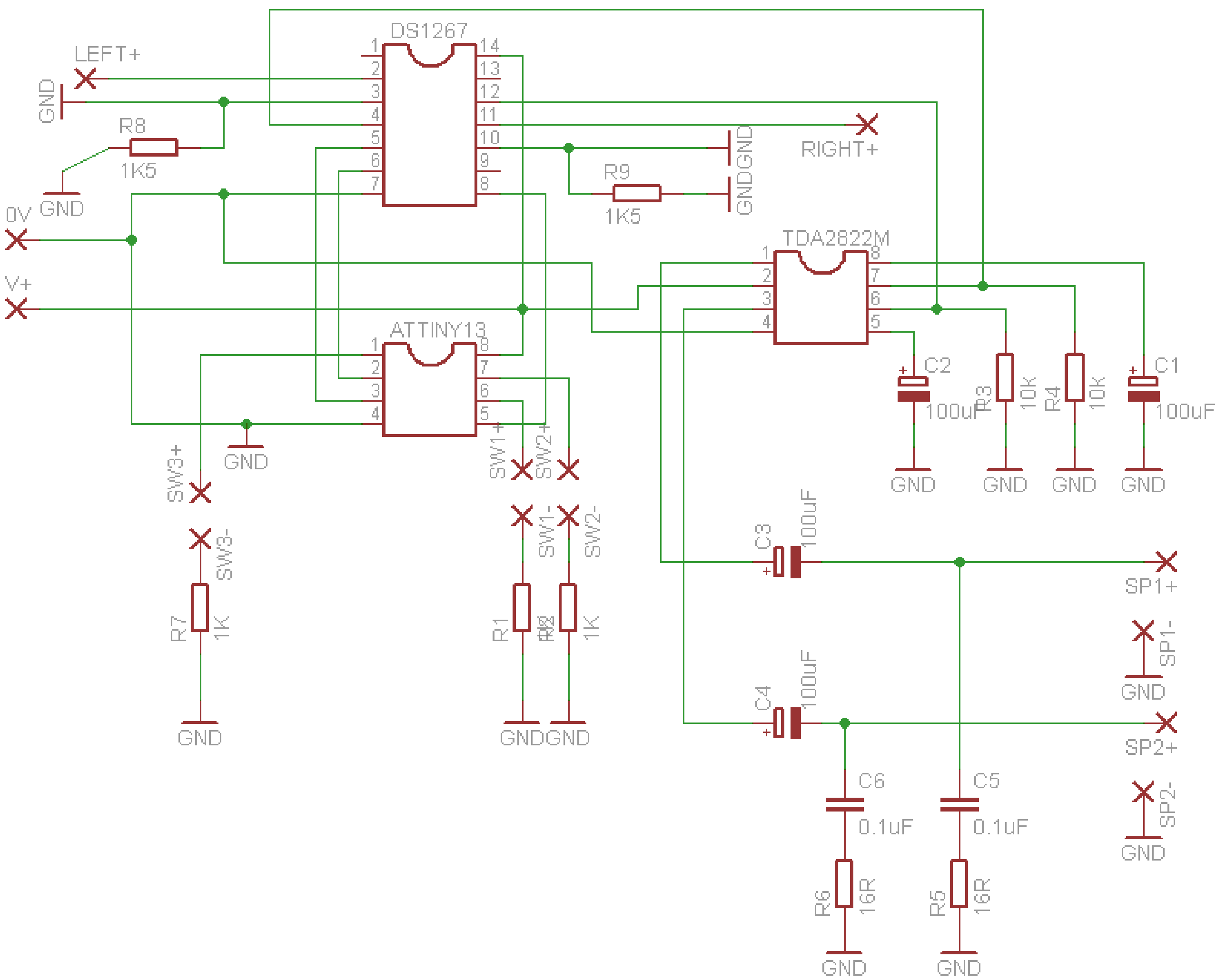
44 Bike audio amplifier project

In this case the client wanted an easy to use and safe audio system for mountain biking.
The solution was to have a small box containing the circuit and battery mounted to the rear of the helmet and speakers clipped onto the helmet near the ears but not blocking out surrounding sounds from other bikers.
There are 3 buttons on the device VOL UP, VOL DOWN and MUTE.
The amplifier is a common TDA2822 stereo audio amp and there is a digital potentiometer controlled by an ATTiny13 to manage the volume settings.

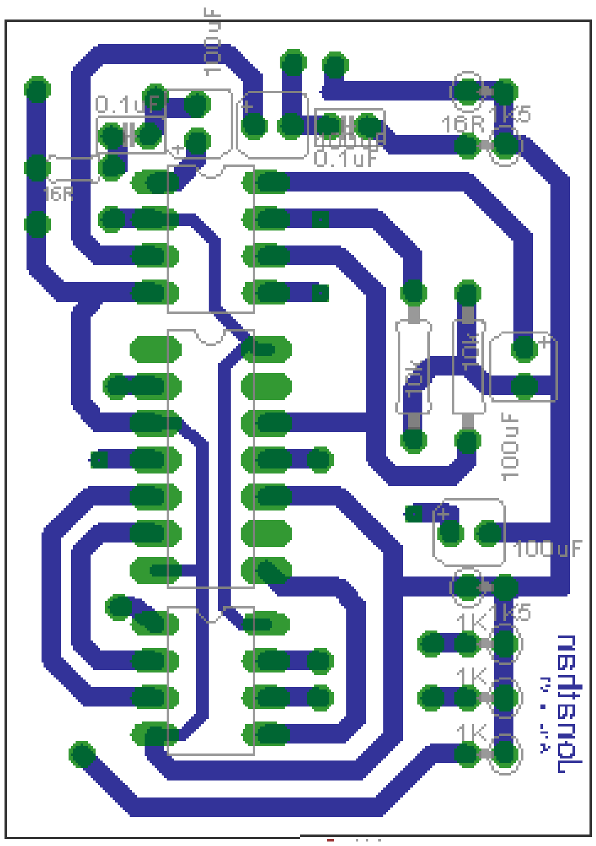
Bike Audio Amp Block Diagram (single channel)



The DS1267 digital pot has 256 settings and requires a serial signal of 17 bits in length sent to it to control it. Bascom has a serial out command however it sends 8 bits, Jonathan decided to ‘bit-bang’ it (send serial bit by bit via software rather than using any hardware device).



DS1267
LEFT+
GND
R8
1K5
0V
GND
V+
RIGHT+
GNDGND
R9
1K5
TDA2822M
ATTINY13
C2
100uF
R3
10K
R4
10K
C1
100uF
GND
SW3+
SW1+
SW2+
SW3-
SW1-
SW2-
R7
1K
R1
R2
1K
C3
100uF
SP1+
SP1-
C4
100uF
SP2+
SP2-
C6
0.1uF
C5
0.1uF
R6
16R
R5
16R
GNDGND



100uF
0.1uF
16R
1K5
0.1uF
10K
10K
100uF
100uF
1K
1K5
1K
1K

```
'-----------------------------------------------------------------
'1.title blcok
'author: jonathan
'date: 2 july 2008
' version 7.0
'file name:potentiometer control7.bas
'-----------------------------------------------------------------
'2.program descrption
'manually shifts out 17 bits to digital potentiometer
'uses buttons to select data to be sent out
'3.hardware features
'2 switches and 3 wire serial interface to digital pot on one port
'-----------------------------------------------------------------
'5. complier directives
$regfile = "atTiny13.dat"
$crystal = 1200000
$hwstack = 20
$swstack = 8
$framesize = 20
'-----------------------------------------------------------------
'6. define hardware
Config Portb = Output
Config Pinb.2 = Input
Config Pinb.1 = Input
Config Pinb.5 = Input
Set Pinb.5
Set Pinb.2
Set Pinb.1
'-----------------------------------------------------------------
'7. hardware aliases
Qb Alias Portb.0
Clk Alias Portb.3
Rst Alias Portb.4
Sw_up Alias Pinb.2
Sw_down Alias Pinb.1
Sw_mute Alias Pinb.5

'8. initialise hardware ports so program starts correctly
Rst = 0
Qb = 0
Clk = 0

'-----------------------------------------------------------------
'9.declare constants

'10. declare variables
Dim V As Byte
Dim B As Byte
Dim S As Byte
Dim State As Bit
'11. initialise variables
B = 8
State = 0
```

```
'12. main program code
Gosub Caseselect
Do
        Debounce Sw_up , 1 , Up , Sub
        Debounce Sw_down , 1 , Down , Sub
        Debounce Sw_mute , 1 , Mute , Sub
Loop
'------------------------------------------------------------------
'13. subroutines
Up:
        B = B + 1
        If B > 22 Then B = 22
        Gosub Caseselect
Return

Down:
        B = B - 1
        If B < 1 Then B = 1
        Gosub Caseselect
Return

Caseselect:
        Select Case B
                Case 1 : V = 0
                Case 2 : V = 4
                Case 3 : V = 10
                Case 4 : V = 16
                Case 5 : V = 25
                Case 6 : V = 35
                Case 7 : V = 50
                Case 8 : V = 65
                Case 9 : V = 80
                Case 10 : V = 100
                Case 11 : V = 120
                Case 12 : V = 145
                Case 13 : V = 170
                Case 14 : V = 200
                Case 15 : V = 230
                Case 16 : V = 255
        End Select
        Gosub Send
Return

Mute:
        If State = 0 Then
                State = 1
                S = V
                V = 0
        Else
                V = S
                State = 0
        End If
        Gosub Send
Return
```

```
Send:
 'bit bang 17 bits of serial data  to digital pot
  Rst = 1
  Qb = 1                              '1
  Clk = 1
  Qb = 0
  Clk = 0
    Qb = V.7                           '2
    Clk = 1
    Qb = 0
    Clk = 0
  Qb = V.6                             '3
  Clk = 1
  Qb = 0
  Clk = 0
    Qb = V.5                           '4
    Clk = 1
    Qb = 0
    Clk = 0
  Qb = V.4                             '5
  Clk = 1
  Qb = 0
  Clk = 0
    Qb = V.3                           '6
    Clk = 1
    Qb = 0
    Clk = 0
  Qb = V.2
  Clk = 1                             '7
  Qb = 0
  Clk = 0
    Qb = V.1
    Clk = 1                           '8
    Qb = 0
    Clk = 0
  Qb = V.0                             '9
  Clk = 1
  Qb = 0
  Clk = 0
    Qb = V.7                           '9
    Clk = 1
    Qb = 0
    Clk = 0
  Qb = V.6
  Clk = 1                             '11
  Qb = 0
  Clk = 0
    Qb = V.5
    Clk = 1                           '12
    Qb = 0
    Clk = 0
  Qb = V.4
  Clk = 1                             '13
```

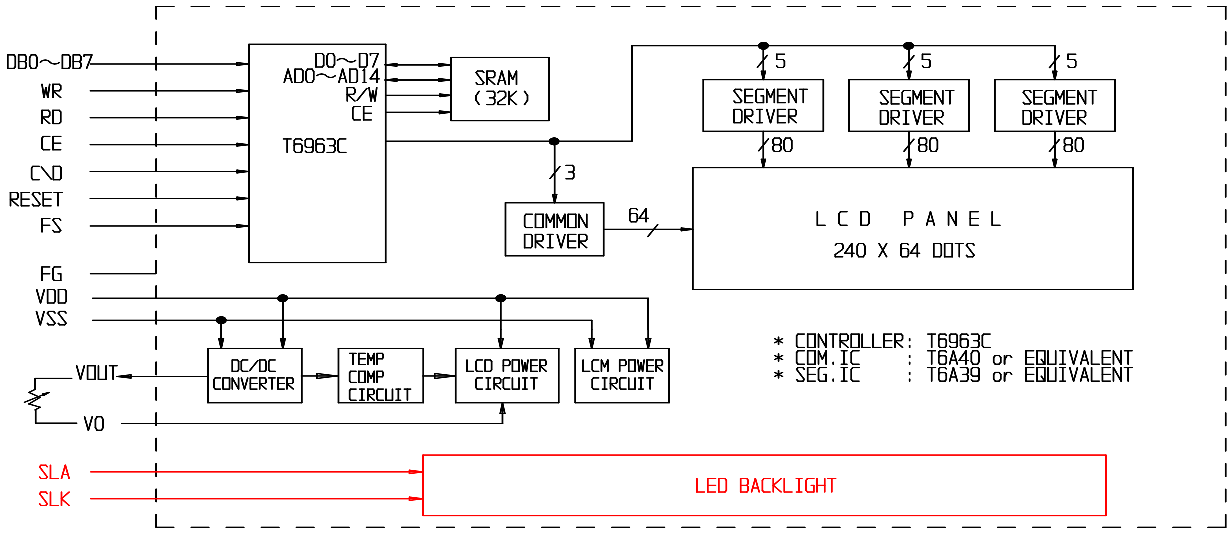
```
  Qb = 0
  Clk = 0
    Qb = V.3
    Clk = 1                          '14
    Qb = 0
    Clk = 0
  Qb = V.2
  Clk = 1
  Qb = 0                             '15
  Clk = 0
    Qb = V.1
    Clk = 1
    Qb = 0                           '16
    Clk = 0
  Qb = V.0                            '17
  Clk = 1
  Qb = 0
  Clk = 0
  Rst = 0
Return
```

45 Graphics LCDs

45.1 The T6963 controller

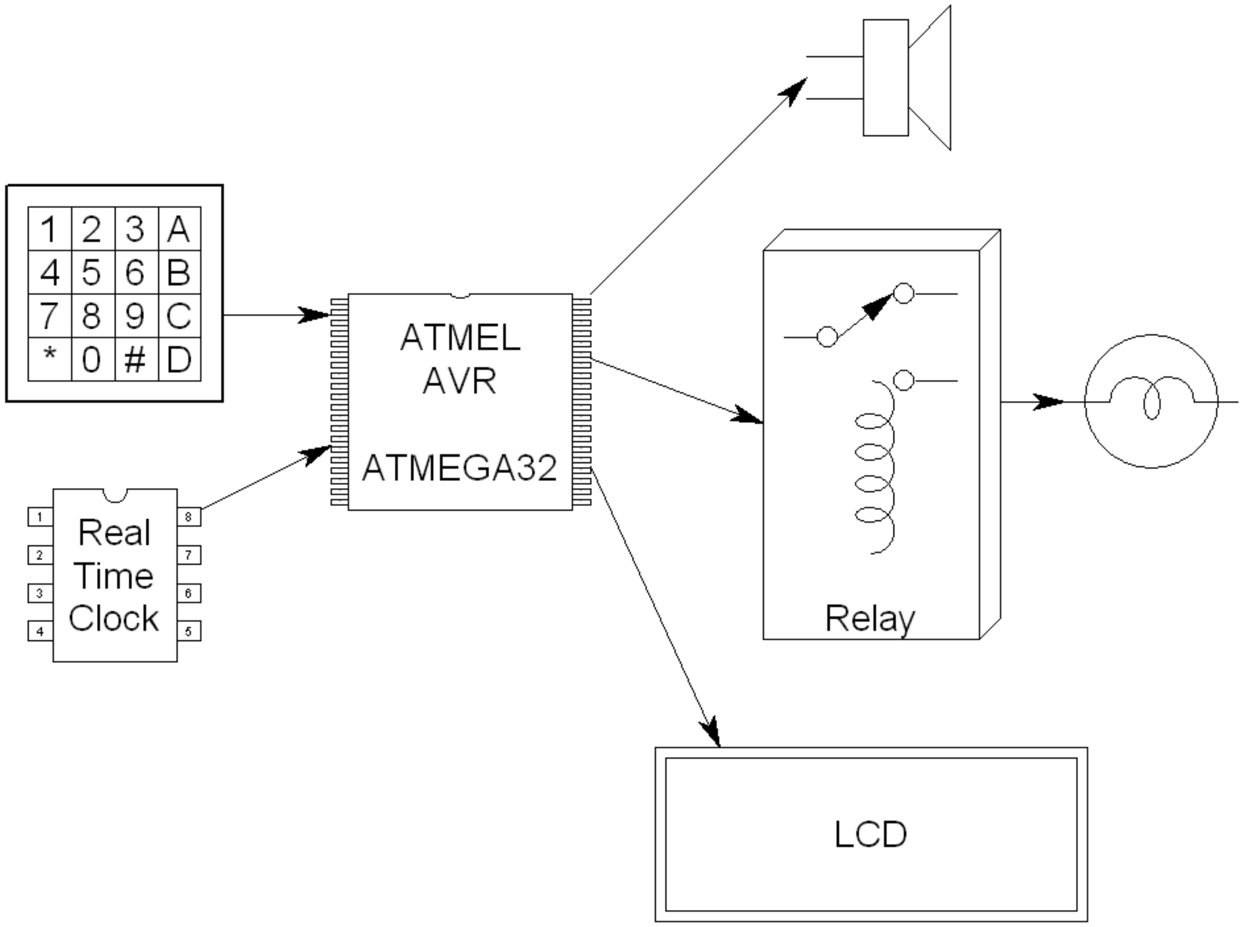
There are a number of different types of graphics LCDs; this display is based on the T6963 driver IC. The display is from TRULY and is 240 pixels wide x 64 pixels high.

The LCD is a complex circuit as shown in the block diagram below, however Bascom has built in routines to drive it which makes it very straight forward to use. There are also built in fonts so it can be used in a similar way to a character LCD (the FS pin is used to select either a 6x8 or 5x7 size font).



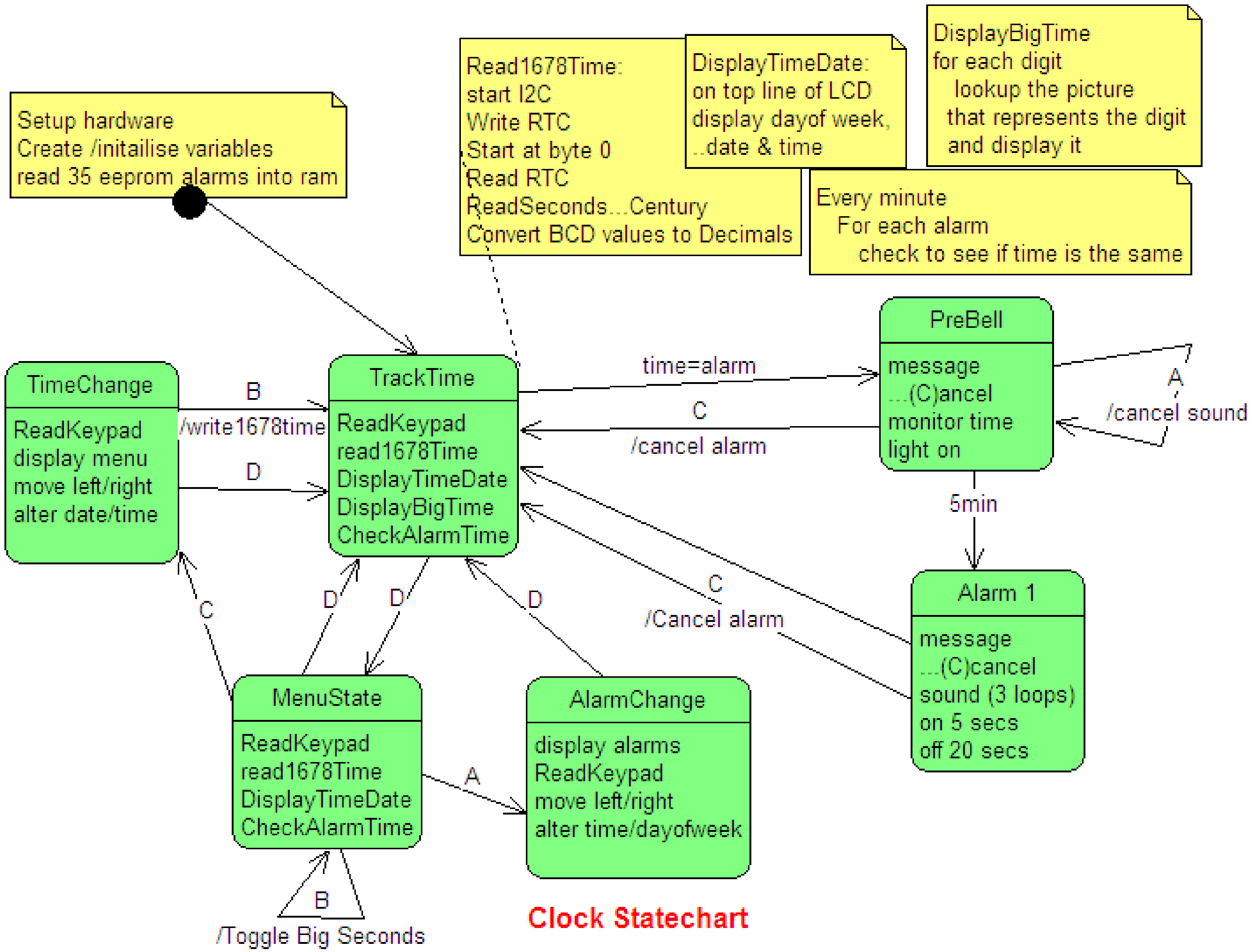
A Bascom program requires a config line for the display:

```
Config Graphlcd = 240 * 64 , Dataport = Portc , Controlport = Portd , Ce = 4 , Cd = 1 ,
          Wr = 6 , Rd = 5 , Reset = 0 , Fs = 7 , Mode = 6
```



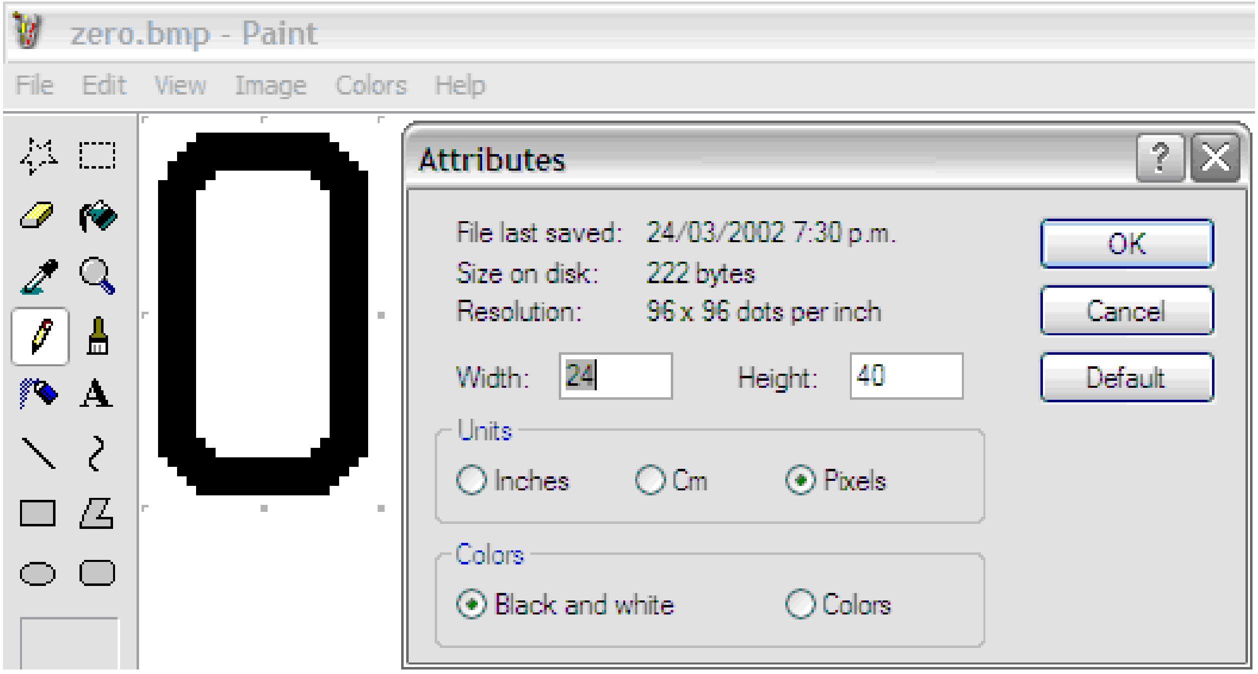
System Block Diagram for this student’s clock project.

The (almost) finished product.



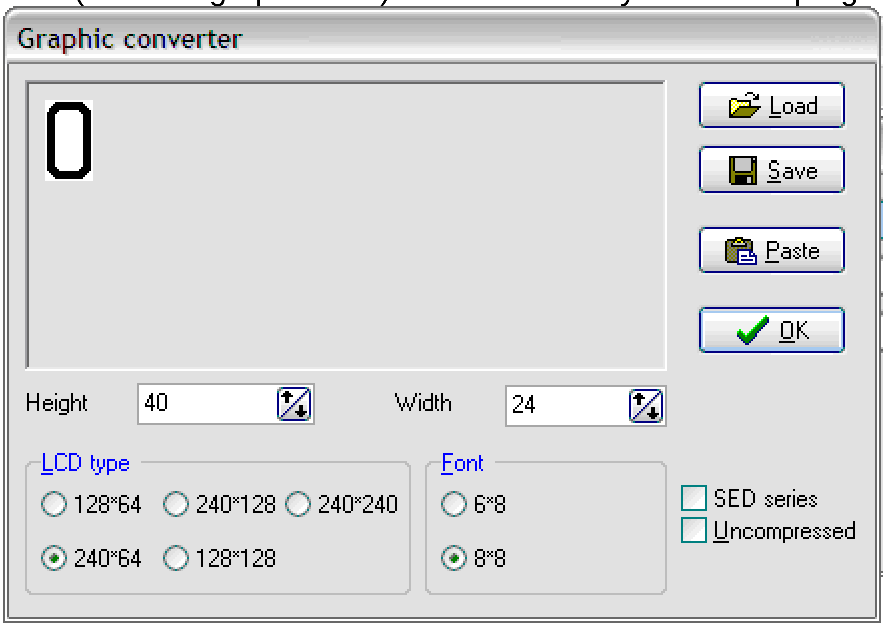
Clock Statechart

The big digits are actually 10 individual pictures which are selected to be displayed on the screen.



Each one is created in a simple drawing program like MS Paint. Use exactly the size BMP file you want the picture to be, in MSPaint the attributes can be set from the menu. Each digit was 24 pixels wide and 40 pixels high (they need to be in multiples of 8 pixels).

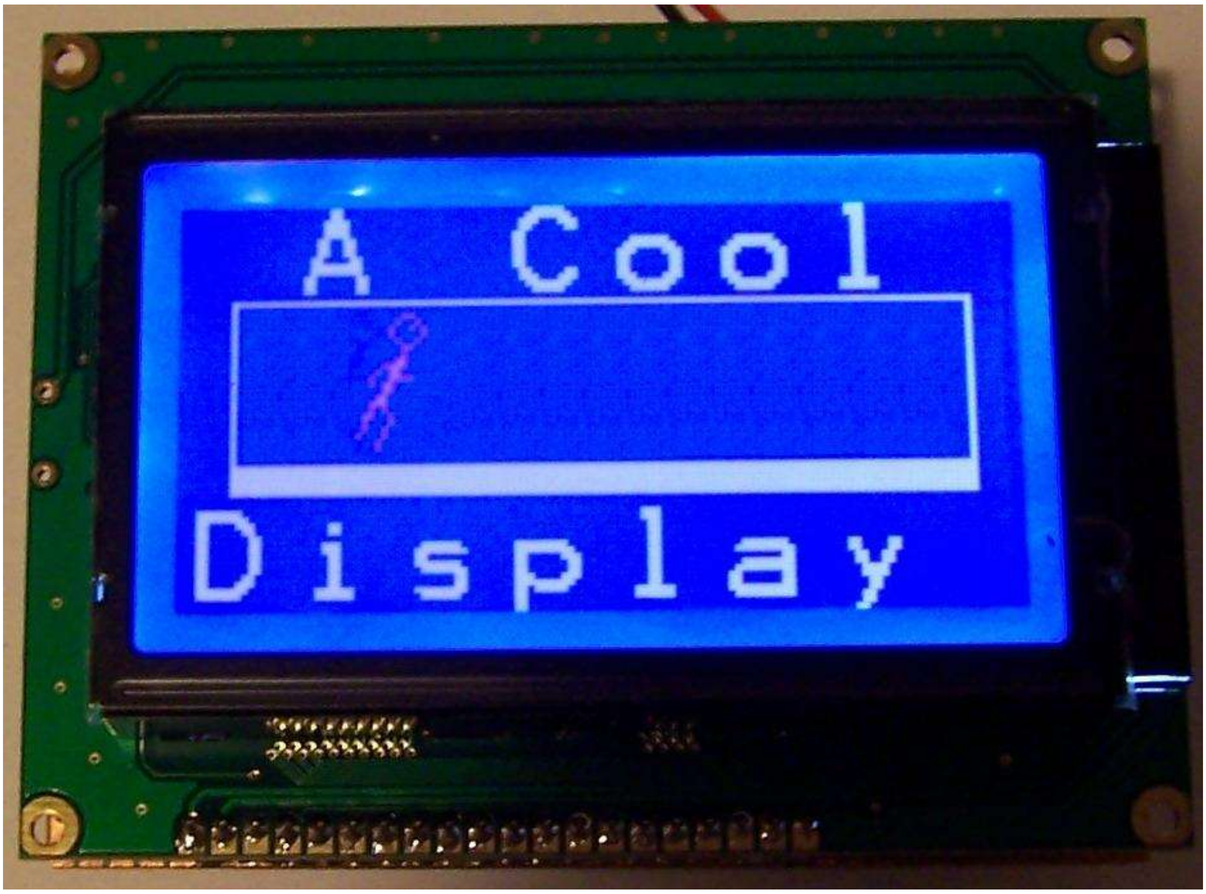
In Bascom open the Graphic Converter, load the bitmap image and then save the file as a BGF (Bascom graphics file) into the directory where the program will be.



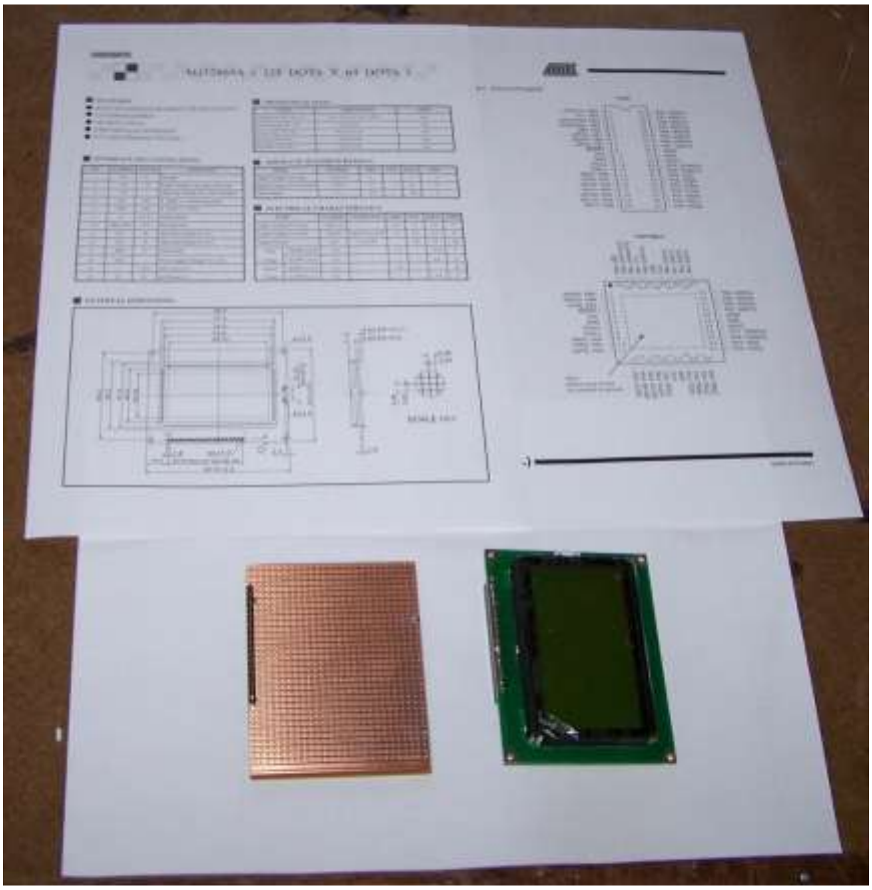
The full program is not listed here however the routine to display the time is.

```
Displaybigtime:
   'first digit
   Digit = 1                     'first digit
   Pic_y = 16                    'fixed location up the GLCD for each graphic
   For Digit = 1 To Numdig       'for each digit location on the GLCD
      Select Case Digit          'get the location of the digit on the display
               Case 1 : Pic_x = 16          at x=16
                        Dig = Hours / 10   ' display tens of hours
               Case 2 : Pic_x = 40          'units of hours go at x=40
                        Dig = Hours Mod 10
               Case 3 : Pic_x = 80           'tens of minutes
                        Dig = Minutes / 10
               Case 4 : Pic_x = 112          'unit minute
                        Dig = Minutes Mod 10
               Case 5 : Pic_x = 144         'tenth second
                        Dig = Seconds / 10
               Case 6 : Pic_x = 176        'unit second
                         Dig = Seconds Mod 10
       End Select
       Select Case Dig                      'actually display the picture at the location
               Case 0 : Showpic , Pic_x , Pic_y , Zero
               Case 1 : Showpic , Pic_x , Pic_y , One
               Case 2 : Showpic , Pic_x , Pic_y , Two
               Case 3 : Showpic , Pic_x , Pic_y , Three
               Case 4 : Showpic , Pic_x , Pic_y , Four
               Case 5 : Showpic , Pic_x , Pic_y , Five
               Case 6 : Showpic , Pic_x , Pic_y , Six
               Case 7 : Showpic , Pic_x , Pic_y , Seven
               Case 8 : Showpic , Pic_x , Pic_y , Eight
               Case 9 : Showpic , Pic_x , Pic_y , Nine
   End Select
   Next
Return
   Zero:                          'labels are required for each picture
      $bgf "zero_6.bgf"
   One:
       $bgf "one_6.bgf"
   Two:
      $bgf "two_6.bgf"
   Three:
      $bgf "three_6.bgf"
   Four:
      $bgf "four_6.bgf"
   Five:
      $bgf "five_6.bgf"
   Six:
      $bgf "six_6.bgf"
   Seven:
      $bgf "seven_6.bgf"
   Eight:
      $bgf "eight_6.bgf"
   Nine:
      $bgf "nine_6.bgf"
```

45.2 Graphics LCD (128x64) – KS0108



In this project the goal is to keep the final product the same size as the LCD. And as it was a one off veroboard was a good choice.

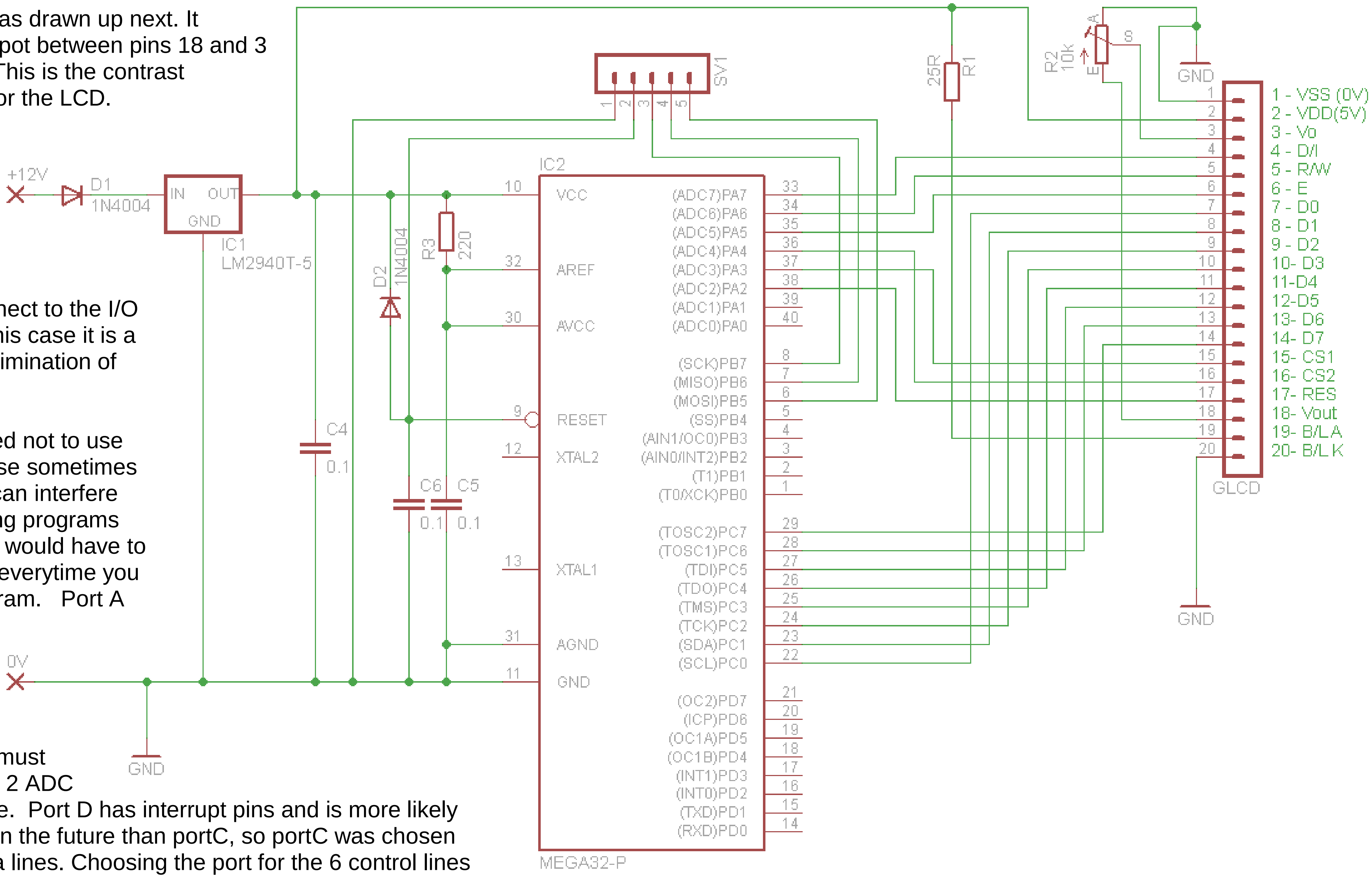


Veroboard is straight forward to use however to get a good product requires some careful planning.

Here the Veroboard, LCD, datasheet for the Microcontroller showing its pin connections and the datasheet for the LCD showing its pin connections are in use to help decide on the ciruit and layout.

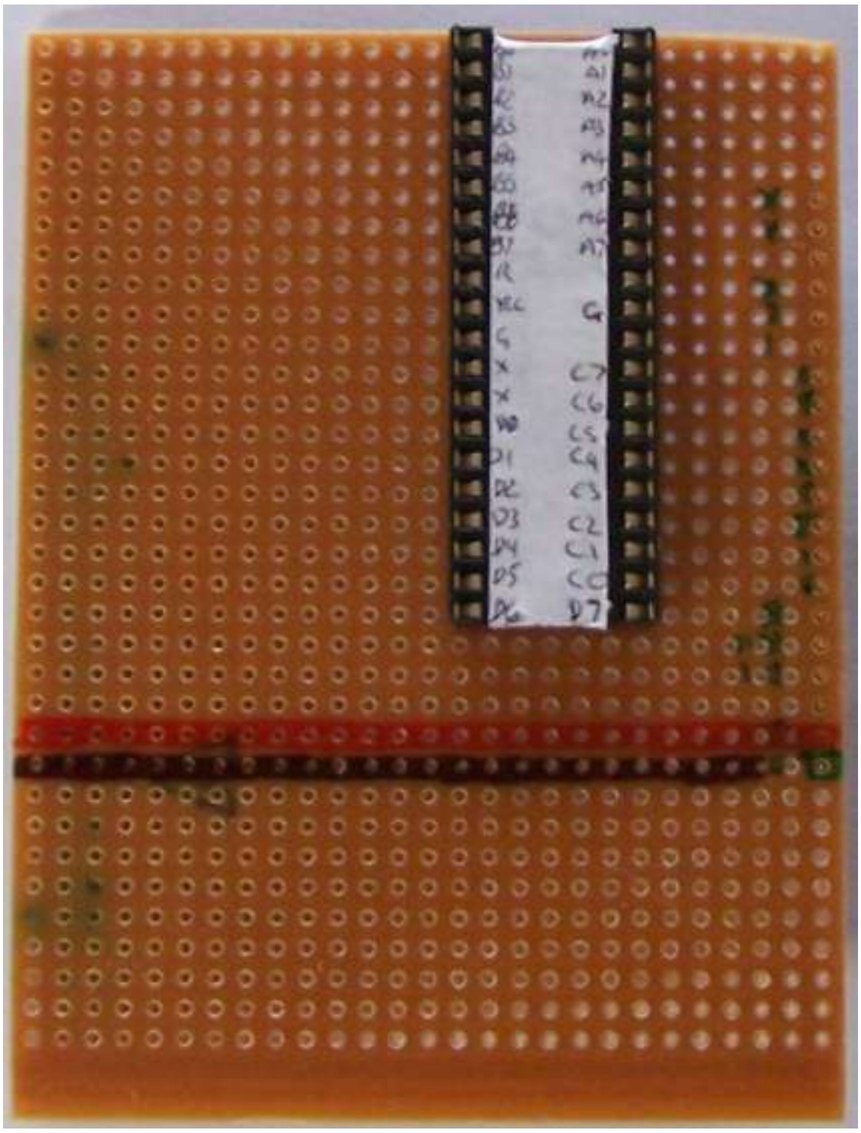
(The display was purchase from sure-electronics)

The circuit was drawn up next. It shows a trimpot between pins 18 and 3 of the LCD. This is the contrast adjustment for the LCD.



It is always a balance working out which pins on the micro to connect to the I/O devices. In this case it is a process of elimination of constraints.

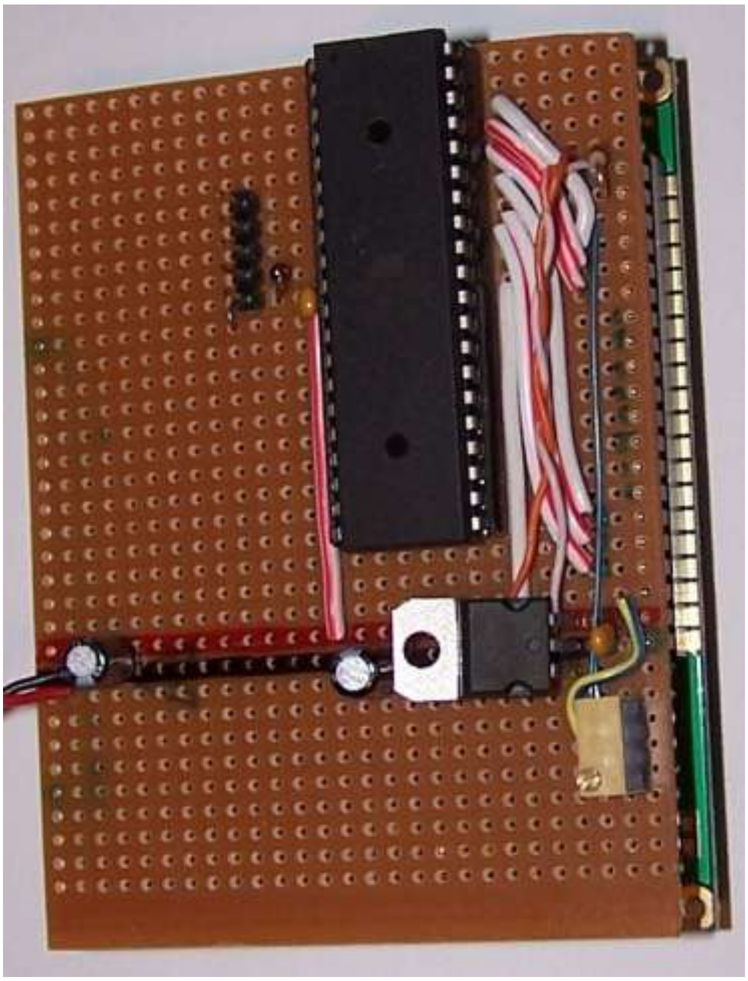
It was decided not to use PortB because sometimes I/O devices can interfere with uploading programs and the LCD would have to be removed everytime you want to program. Port A has the ADC on it and if a touch screen is required we must have at least 2 ADC pins available. Port D has interrupt pins and is more likely to be useful in the future than portC, so portC was chosen for the 8 data lines. Choosing the port for the 6 control lines was easy, portA, as we will have 2 spare. Note that it is a good idea not to write data to the LCD while doing an ADC conversion as this could mess up the ADC results. 0.1uf (100nF) bypass capacitors were added to the circuit, one on the power pins of the micro and one next to the power pins of the LCD, these stop voltage spikes on the power supply caused by fast switching devices like microcontrollers and LCDs upsetting the power supply to other devices like microcontrollers, LCDs and any other ICs that will be added. We need to bypass each device with a capacitor real close to the IC.



To make assembly easier to follow the IC was mounted right up to the edge of the board so that its portc pins physically lined up with the 8 datalinesof the LCD. This reduced the wiring.

Before attempting to do the wiring of the micro to the LCD a label was placed onto the IC socket with the names of the pins, and the names of the LCD pins were written using a permanent marker onto the board itself. This really helps avoid confusion when flowing the schematic.

The 5V and 0V/GND lines were coloured red and black on the board. The reason these are where they are on the veroboard is that they line up with the power pins of the LCD.



The 7805 was positioned so that it was directly onto the 5V and 0V lines.

There is plenty of space left on the board for other circuitry. Perhaps a real time clock and a touch screen connection.

The code for the display is straight forward

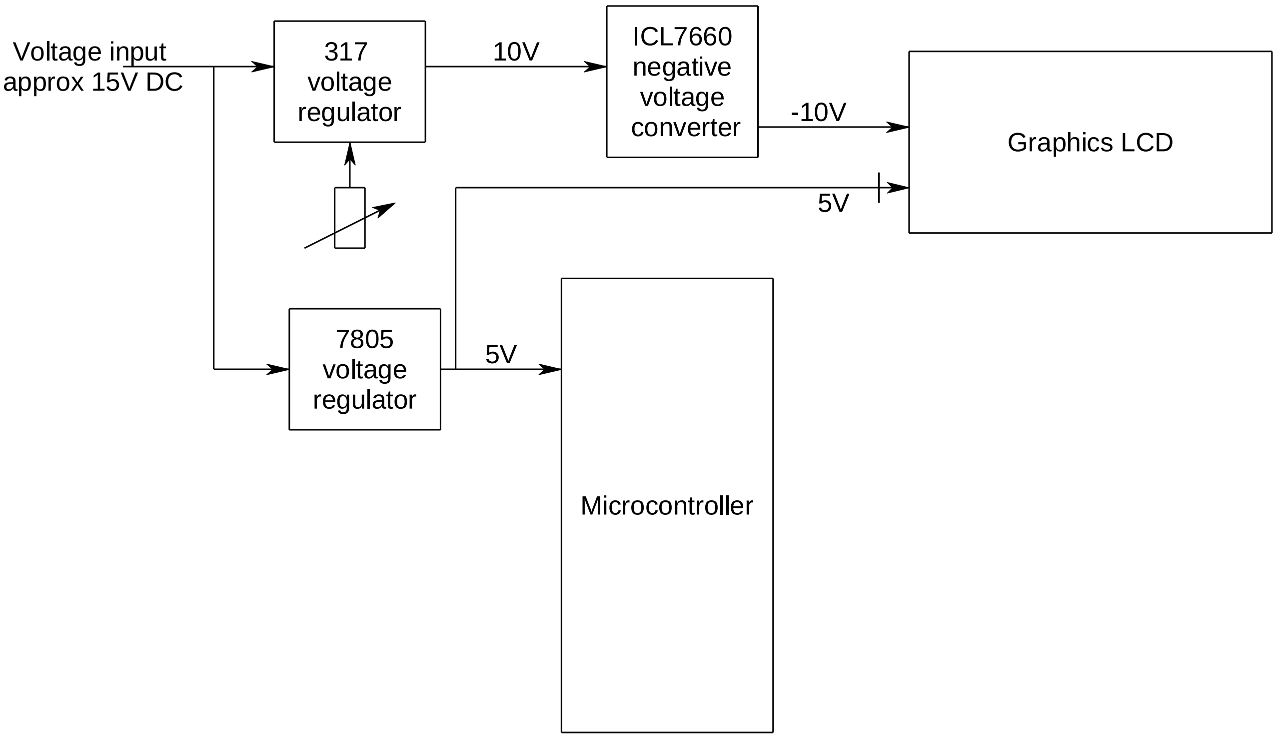
```
'-----------------------------------------------------------------
' Title Block
' Author: B.Collis
' Date: 1 June 2008
' File Name: GLCD_KS_ver1.bas
'-----------------------------------------------------------------
' Program Description:
' A simple clock
' Hardware Features:
' 128x64 GLCD
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m32DEF.dat"              ' specify the used micro
$crystal = 8000000                   ' used crystal frequency
$lib "glcdKS108.lib"                 ' library of display routines
'-----------------------------------------------------------------
' Hardware Setups
'Configure GLCD interface
'CE      CS1 select     pin15   CE      A.3
'CE2     CS2 select2    pin16   CE2     A.4
'CD      DI             pin4    CD      A.7
'RD      Read           pin5    RD      A.6
'RESET   reset          pin17   R       A.2
'ENABLE  Chip Enable    pin6    En      A.5
Config Graphlcd = 128 * 64sed , Dataport = Portc , Controlport = Porta , Ce = 3 , Ce2 = 4 ,
Cd = 7 , Rd = 6 , Reset = 2 , Enable = 5
'Hardware Aliases
'-----------------------------------------------------------------
' Declare Constants
Const Runningdelay = 170
'-----------------------------------------------------------------
' Declare Variables
Dim X As Byte
Dim Y As Long
' 11. Initialise Variables
'Date$ = "14/06/08"                  'default starting date
'Time$ = "19:12:00"                  'default starting time
'-----------------------------------------------------------------
' Program starts here
Cls
Setfont Font 16x16                   'specify the small font

Lcdat 1 , 1 , " A Cool"              'the rows are from 1 to 8
Lcdat 7 , 1 , "Display"              '

Line(8 , 15) -(120 , 15) , 1         'top line
Line(8 , 15) -(8 , 41) , 1           'left vertical line
Line(120 , 15) -(120 , 41) , 1       'right vertical line

For Y = 41 To 45                     'own simple filledbox
   Line(8 , Y) -(120 , Y) , 1
Next

'show the three pics in sequence to get simple animation
```

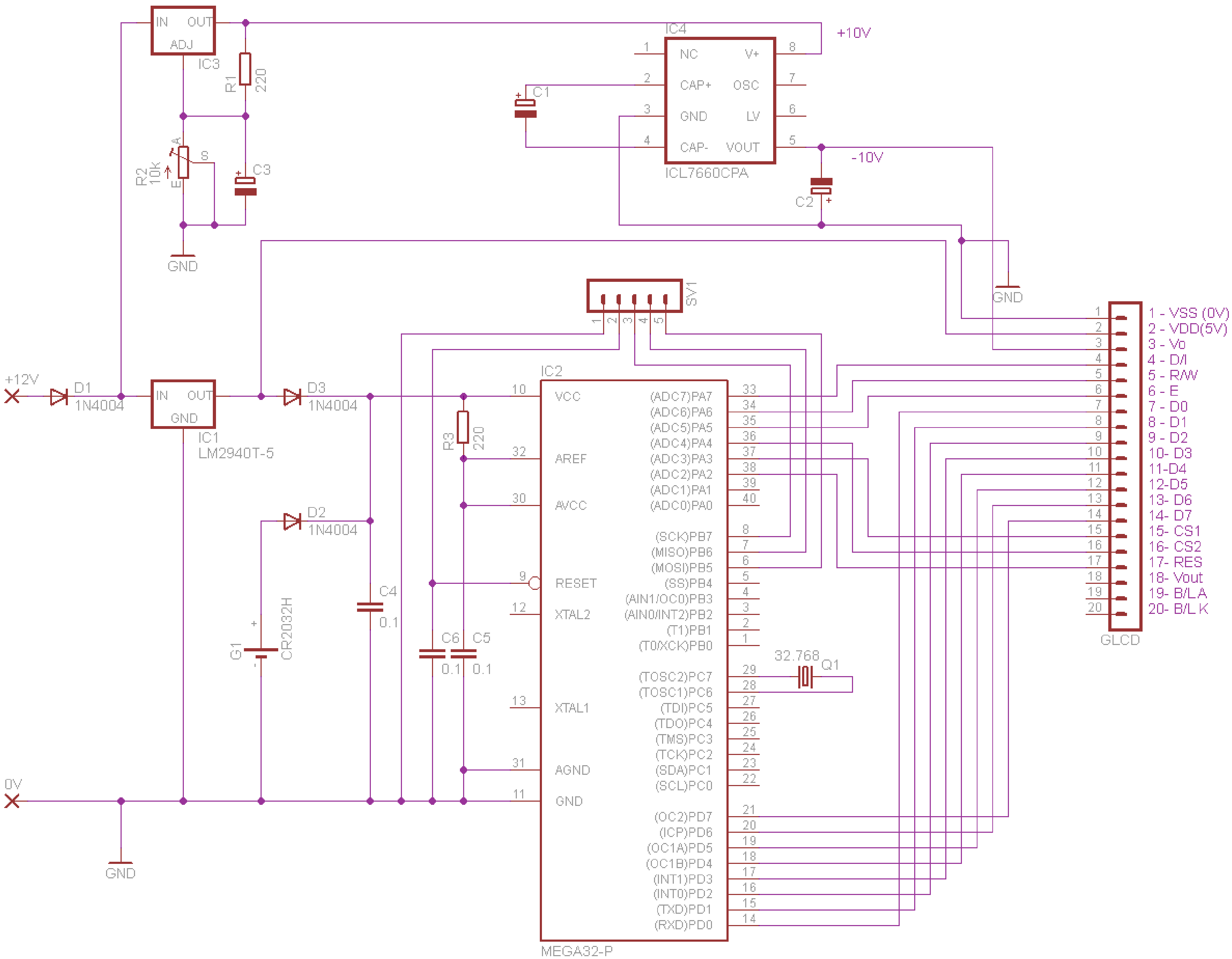
```
Do
  For X = 10 To 104 Step 8
    Showpic X , 20 , Run1
    Waitms Runningdelay
    Showpic X , 20 , Blank
    X = X + 8
    Showpic X , 20 , Run2
    Waitms Runningdelay
    Showpic X , 20 , Blank
    X = X + 8
    Showpic X , 20 , Run3
    Waitms Runningdelay
    Showpic X , 20 , Blank
  Next
  Waitms 500
Loop
End                              'end program

'-----------------------------------------------------------------
'the font and graphic files must be in the same directory as the .bas file
'these lines put the fonts into the program flash
$include "font16x16.font"

Run1:
$bgf "run1.bgf"
Run2:
$bgf "run2.bgf"
Run3:
$bgf "run3.bgf"
Blank:
$bgf "blank.bgf"
```

45.3 Generating a negative supply for a graphics LCD

These particular displays were available at a very good price; however they did not have the negative voltage circuit on the display for the contrast adjustment making them a little trickier to use.

This block diagram shows the power supply voltages required and how they were developed. The 317 is an adjustable regulator and a trimpot on it will be used to vary the voltage and consequently the LCD’s contrast.



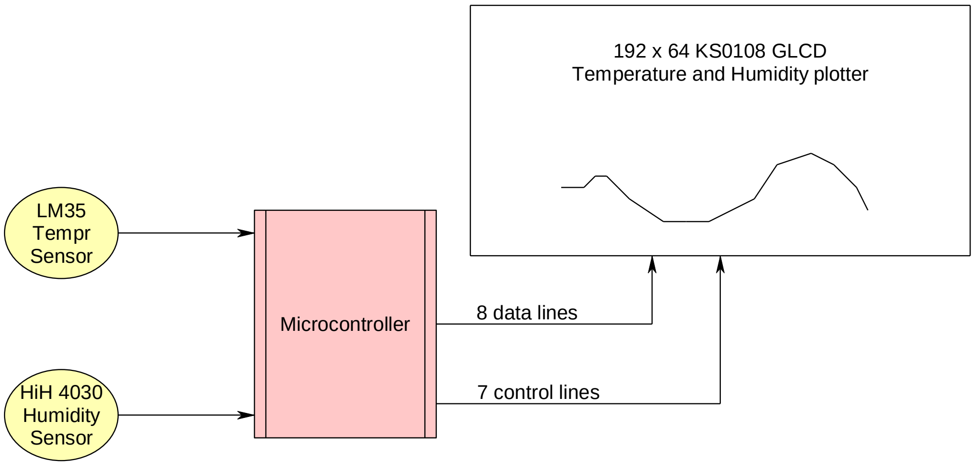

IC3
IN OUT ADJ
R1 220
R2 10K
C3
GND
IC4
NC V+ CAP+ OSC GND LV CAP- VOUT
ICL7660CPA
C1
C2
+10V
-10V
SV1
+12V
D1 1N4004
IC1 LM2940T-5
IN OUT GND
D3 1N4004
D2 1N4004
G1 CR2032H
C4 0.1
C6 0.1
C5 0.1
R3 220
0V
IC2
VCC
AREF
AVCC
RESET
XTAL2
XTAL1
AGND
GND
(ADC7)PA7
(ADC6)PA6
(ADC5)PA5
(ADC4)PA4
(ADC3)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
(SCK)PB7
(MISO)PB6
(MOSI)PB5
(SS)PB4
(AIN1/OC0)PB3
(AIN0/INT2)PB2
(T1)PB1
(T0/XCK)PB0
(TOSC2)PC7
(TOSC1)PC6
(TDI)PC5
(TDO)PC4
(TMS)PC3
(TCK)PC2
(SDA)PC1
(SCL)PC0
(OC2)PD7
(ICP)PD6
(OC1A)PD5
(OC1B)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
MEGA32-P
32.768 Q1
GLCD
1 - VSS (0V)
2 - VDD(5V)
3 - Vo
4 - D/I
5 - R/W
6 - E
7 - D0
8 - D1
9 - D2
10- D3
11-D4
12-D5
13- D6
14- D7
15- CS1
16- CS2
17- RES
18- Vout
19- B/L A
20- B/L K

46 GLCD Temperature Tracking Project

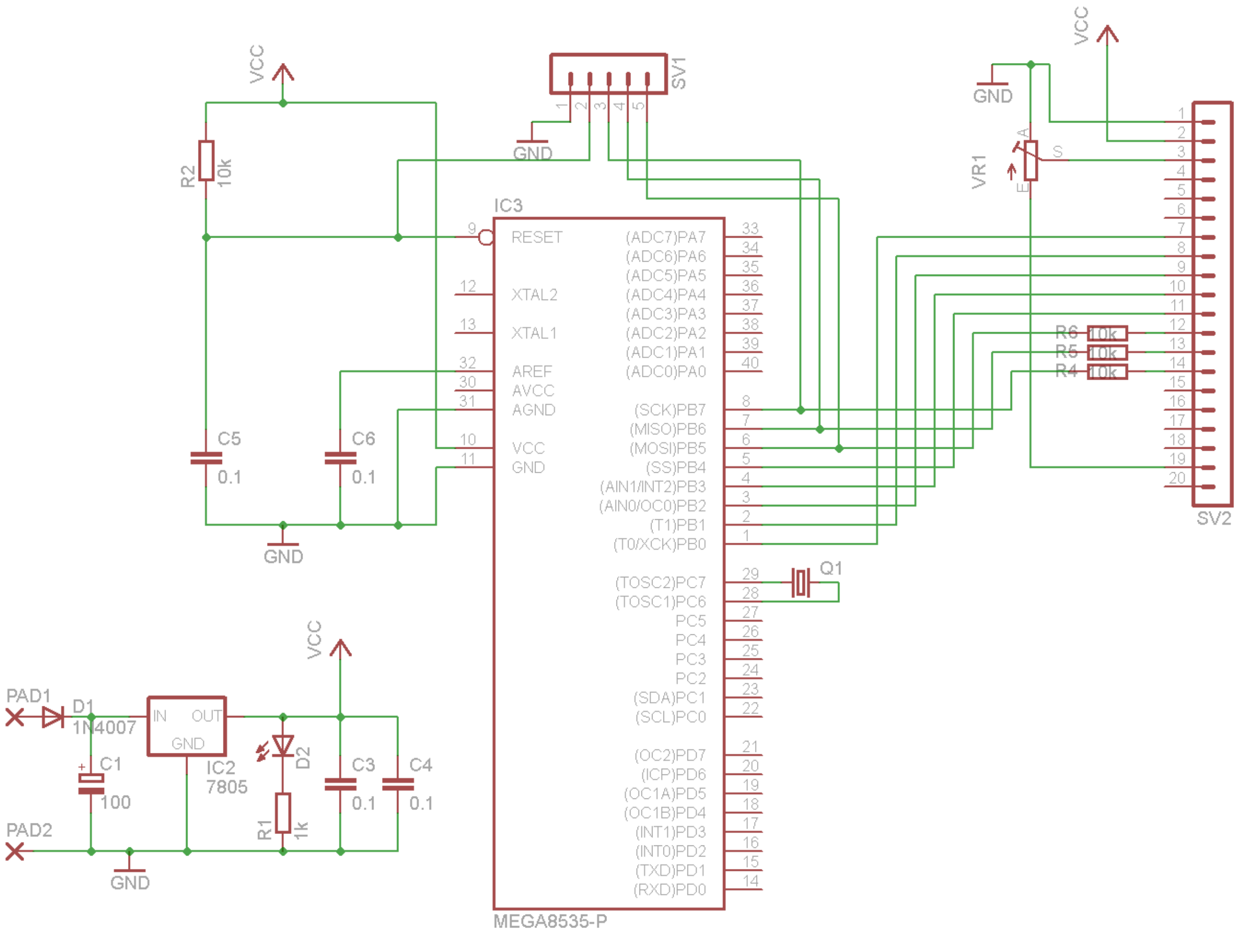
46.1 Project hardware

In thisproject I wanted to use a GLCD to display a graph of temperature and humidity over time. I had the following:

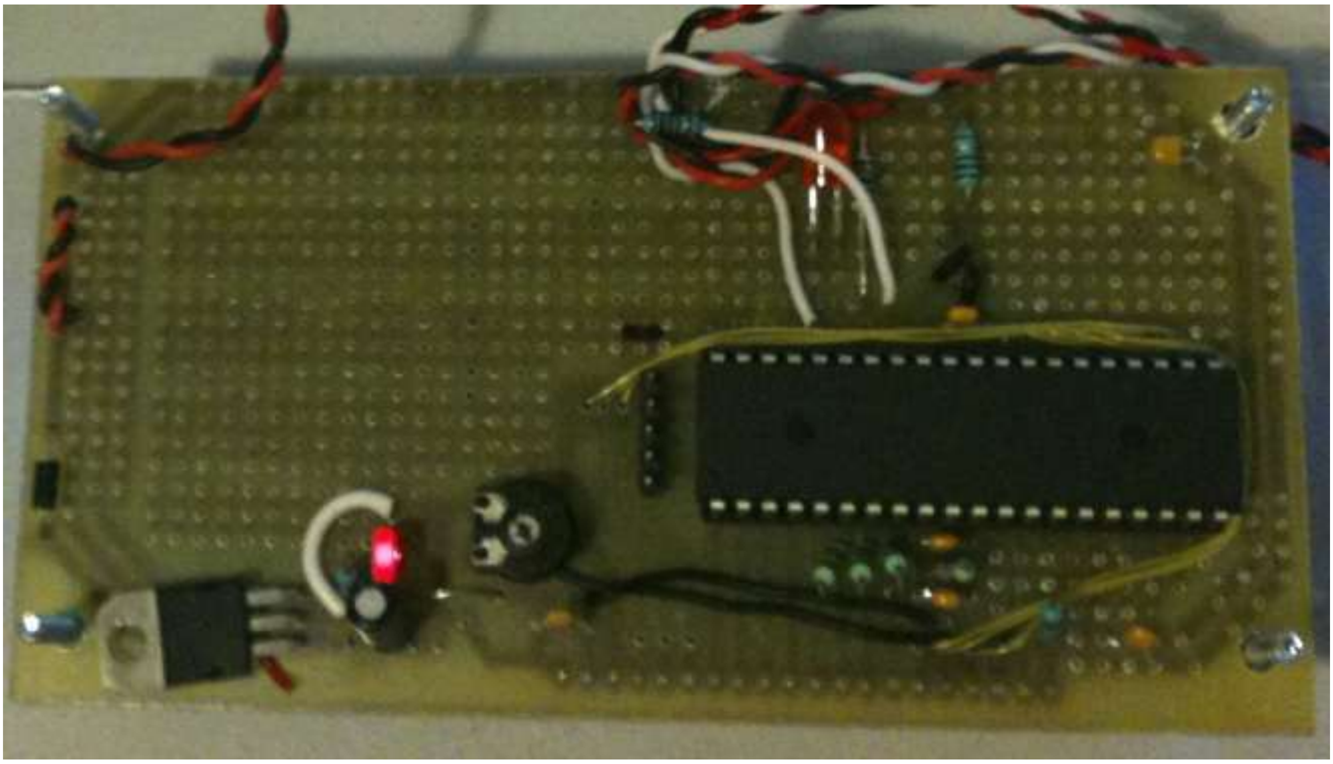
a 192x64 pixel GLCD (KS0108 type from Sure Electronics)
an LM35 temperature sensor
an HiH4030 humidity sensor



The 192x64 GLCD has 1 more interface pin than the 128x64 GLCD as it has a third controller for the display. This makes a total of 7 control lines between the microcontroller and the GLCD. When I designed this board for student use I decided that the data lines could be on PortB (shared with the programming port – which is ok if you add the 10k resistors as per the schematic) and that the control lines would have to be flexible so that depending on the use for the board the students could change them.



In this photo the fine yellow wires are the 7 control lines added later.

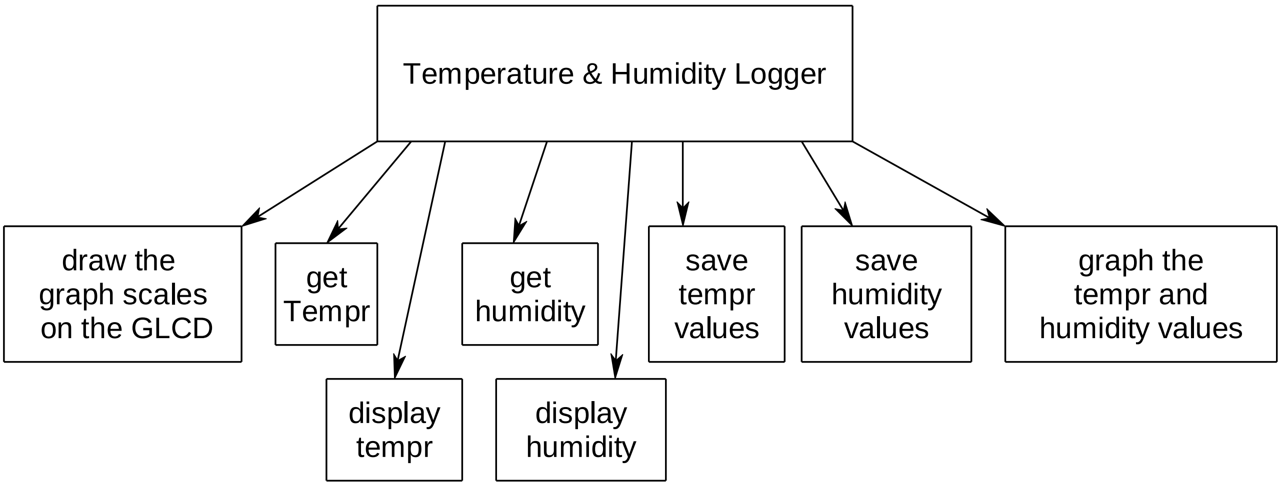


In the software Bascom has a different library for this GLCD so it must be added and your wiring above must be configured in the software as below.

```
'-------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$lib "glcdKS108-192x64.lib"                   ' library of display routines
'-------------------------------------------------------------------
' Hardware Setups
'Configure GLCD interface
'CE      CS1 select     GLCD-pin15   CE      portC.3
'CE2     CS2 select2    GLCD-pin17   CE2     portC.5
'CE3     CS3 select3    GLCD-pin18   CE6     portC.6
'CD      RS             GLCD-pin4    CD      portC.0
'RD      RW             GLCD-pin5    RD      portC.1
'RESET   reset          GLCD-pin16   R       portC.4
'ENABLE  Chip Enable    GLCD-pin6    En      portC.2
Config Graphlcd = 192 * 64sed , Dataport = Portb , Controlport = Portc
, Ce = 3 , Ce2 = 5 , Cd = 0 , Rd = 1 , Reset = 4 , Enable = 2 , Ce3 =
6
```

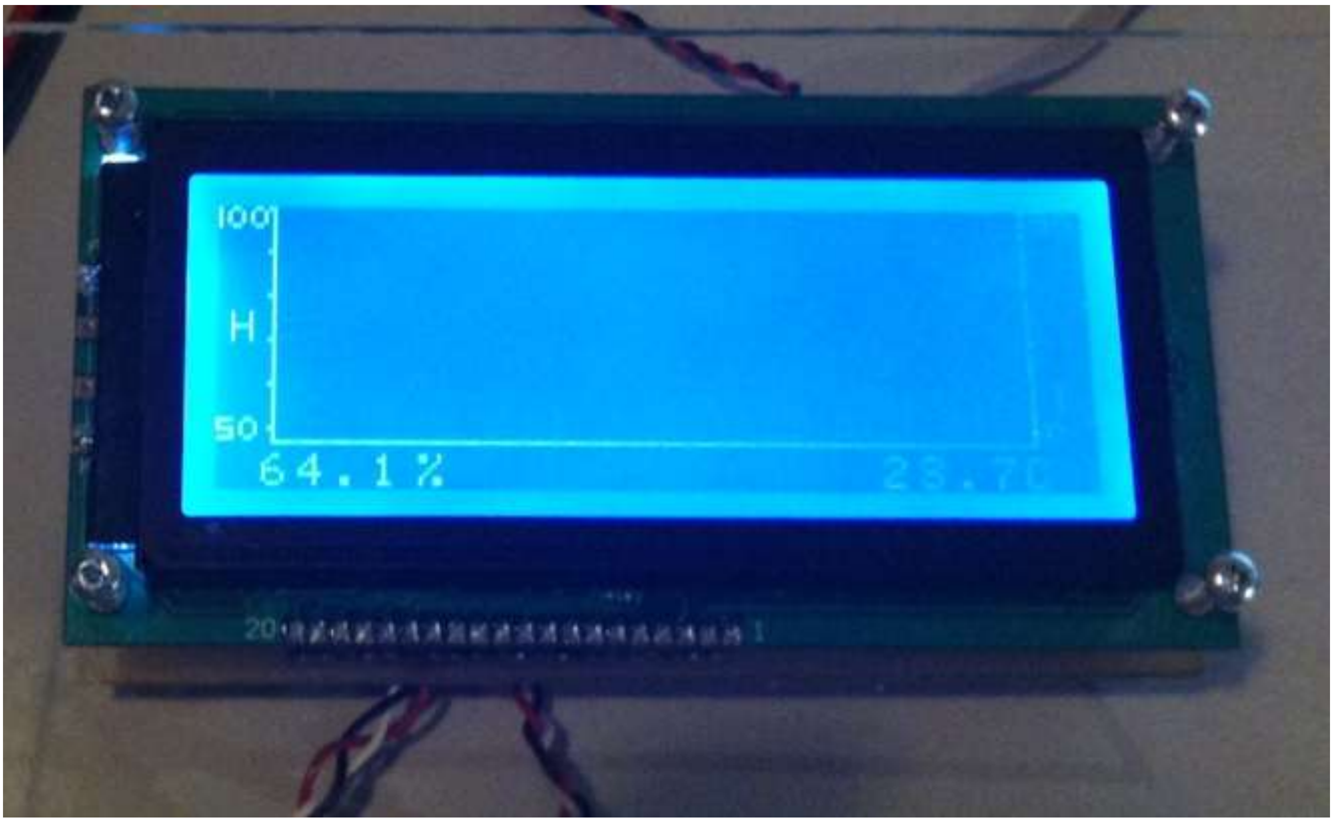
46.2 Project software planning

This is a realtively complex system which will require some interesting software to plot a graph of values so I will use decomposition to break the software down into subroutines each with its own job to do.



The least complex parts of the software for the project will be the displaying of the graph scales and the values, the next will be reading the values from the sensors and translating these to humidity and temperature, the most challenging will be the last part actually graphing the values.

Here is what the display looks like with the graph scales and the temperature and humidity values displayed.



46.3 Draw the graph scales

```
'-----------------------------------------------------------------
Draw_graph_scales:
   Line(12 , 0) -(12 , 52) , 1           'left vertical
   Line(178 , 0) -(178 , 52) , 1         'right vertical
   Line(12 , 53) -(178 , 53) , 1         'bottom horizontal
   'left hand side humidity scale
   Setfont Font 8x8
   Lcdat 4 , 3 , "H"
   Pset 11 , 0 , 1
   Pset 11 , 10 , 1
   Pset 11 , 20 , 1
   Pset 11 , 30 , 1
   Pset 11 , 40 , 1
   Pset 11 , 50 , 1
   Setfont Font 5x5
   Line(0 , 0) -(0 , 4) , 1              '1 in the 100 to save space
   Lcdat 1 , 2 , "00"
   Lcdat 7 , 0 , "50"
   'right hand side temperature scale
   Setfont Font 8x8
   Lcdat 4 , 3 , "T"
   Pset 179 , 0 , 1
   Pset 179 , 10 , 1
   Pset 179 , 20 , 1
   Pset 179 , 30 , 1
   Pset 179 , 40 , 1
   Pset 179 , 50 , 1
   Setfont Font 5x5
   Lcdat 1 , 181 , "50"
   Lcdat 7 , 181 , "0"
Return
```

This routine makes use of some of the Bascom functions for the display and use two different font sizes. I use comments to help me to remember what each part does.

There is one small point to make about the 100 on the left of the display. I wanted to maximise the display space for plotting values so when I went to display the number 100 it took up a lot of space as each character is 5 pixels wide. I reduced that by drawing a line in place of the character 1 and then putting in “00” after it, thus reducing my width for the 100 from 15 pixels to 12, leaving me room for 3 more data point in the display itself. When I went to draw the check marks for the scale I wrote the check mark over the top of the last 0 increasing my display by another data point. I now have 165 data points that I can use to display values out fo the full 192 pixels width.

46.4 Read the values

The LM35 temperature sensor has been covered already but note the conversion from volts to degrees. To do this I measured the voltage on the LM35 it was 0.282V (28.2 degrees) the ADC value was 56 (on a scale from 0 to 1023) and as I know there is a straight line relationship between the two that starts at 0. I got a simple conversion factor of 1.9858. In maths I might express that as a formula of the type Y=mX+C or in this case Tempr = conversion factor times ADC reading (plus zero for C as the graph crosses at 0 volts).

```
'-----------------------------------------------------------------
Get_tempr:
' lm35 temperature sensor on pinA.7
'  calibrated at adc=56 and temperature=28.2deg (0.282V)
'  56/28.2 = 1.9858
   Lm35 = Getadc(7)                    'get the raw analog reading
   Tempr_single = Lm35           'convert to single to use decimals
   Tempr_single = Tempr_single / 1.9858
   Tempr = Tempr_single                'convert to byte for storage
Return

Disp_tempr_val:
   Setfont Font 8x8
   Lcdat 8 , 145 , Tempr_single
   Lcdat 8 , 176 , "C"
Return                                 '
'-----------------------------------------------------------------
```

Note the need to convert the between different variable types.

The ADC readings are whole numbers in the range of 0 to 1023, so these are initially word types(e.g LM35 above). I want to do division with these though and word type variables truncate division so I convert the values to single variable types (tempr_single above). After I have finished doing the fomula I want to store the values in memory and I want to store a lot of them so I convert the values to byte type variables which take up much less space (e.g. tempr above)

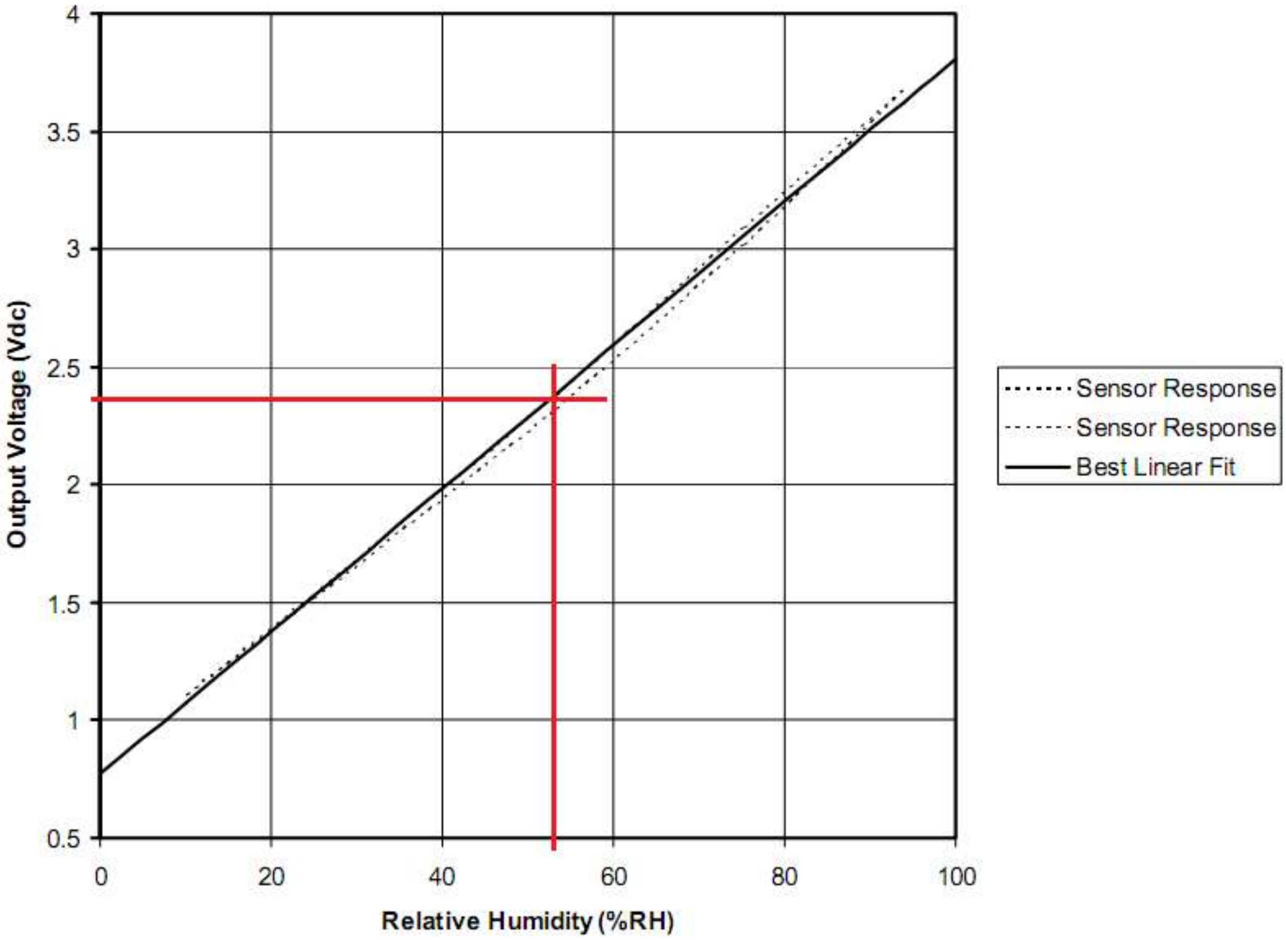
I used the HIH4030 humidity sensor, it can be bought from Sparkfun.com mounted on a small PCB. It is another easy to use analogue sensor and has a very linear scale so a straight forward formulae is required.

In this case the voltage corresponds to a humidity value which we look up on a graph from the datasheet.

I measured 2.37V which was an ADC value of 480.

An ADC value of 480 (2.37V) is a humidity of about 55% on the graph.

Note that 0% humidity is not 0V (as it is with the LM35 for temperature) so our formula is more in the form Y=mX+C.From the graphI estimated that the formula is Voltage=0.0306 x humidity + 0.78.
To get humidity I changed this around to be humidity = (Voltage-0.78)/0.0306.




```
'----------------------------------------------------------------
Get_humidity:
' humidity sensor HIH4030 on pin a.4
'  calibrated at adc=480 and voltage = 2.37
'  formula for hum=(V-0.78)/0.0306 - worked out from datasheet
   Hih4030 = Getadc(4)                    'get raw adc value
   Hum_single = Hih4030                   'convert to single
   Hum_single = Hum_single / 203.85       'convert raw adc to volts
   Hum_single = Hum_single - 0.78
   Hum_single = Hum_single / 0.0306
   Humidity = Hum_single                  'convert to byte for storage
Return

Disp_humidity_val:
   Setfont Font 8x8
   Lcdat 8 , 10 , Hum_single              'single to display decimal
value
   Lcdat 8 , 44 , "%"
Return
```

46.5 Store the values

First I need to store 165 readings for each so I dimension two arrays

```
Dim T(165) As Byte                  '165 readings stored
Dim H(165) As Byte
```

The first location is T(1) and then next T(2) all the way up to T(165).

In the main loop I wait for 5 minutes (wait 300) between readings and after each reading I increase a variable which is keeping track of the number of readings. I also do not want to go over 165 so I test this variable and reset it back to 1 if it goes over 165.

```
'------------------------------------------------------------------
' Program starts here
'setup inital screen
Cls
Gosub Draw_graph_scales

Do
   Gosub Get_humidity
   Gosub Save_humidity
   Gosub Get_tempr
   Gosub Save_tempr
   Gosub Disp_humidity_val
   Gosub Disp_tempr_val
   Gosub Draw_tempr_hum_graphs
   Wait 300                            'reading every 5 minutes
   Incr Curr_reading
   If Curr_reading > 165 Then Curr_reading = 1
Loop
End
```

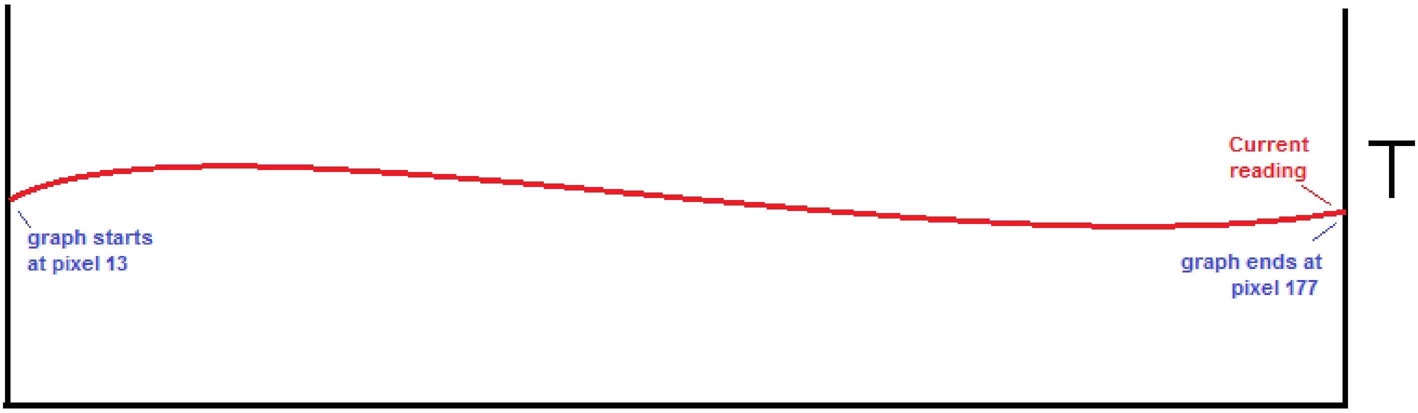
I have two routines for storing the values in ram even though I could do it in one subroutine and call it something like save_values. This is because each has a slightly different function to perform and if I extend the program in the future I might want to add features to one routine that aren't in the other such as keeping track of the maximum temperature or something else.

```
'----------------------------------------------------------------
Save_humidity:
   H(curr_reading) = Humidity
Return
'----------------------------------------------------------------
Save_tempr:
   T(curr_reading) = Tempr
Return
```

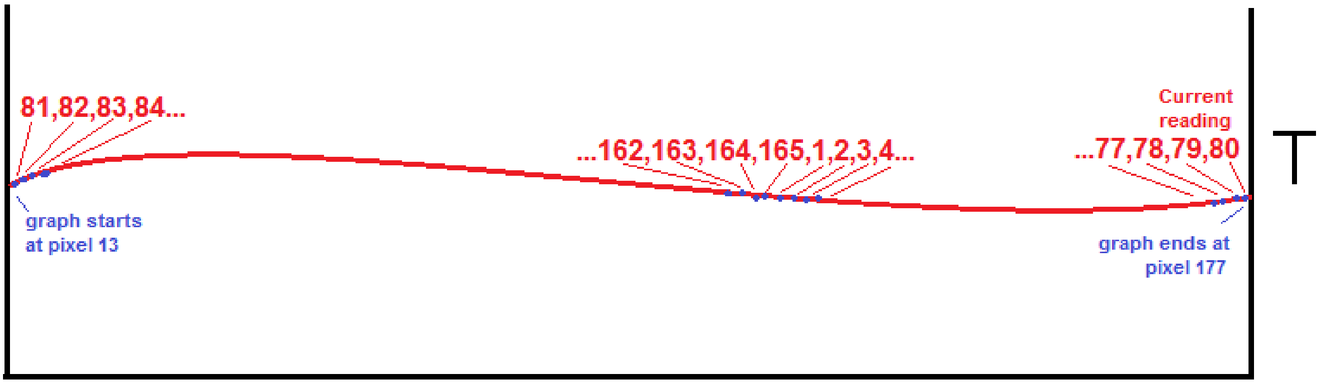
Storing the values is easy I copy the value from the variable Humidity into the array at the position determined by my increasing variable curr_reading

46.6 Plot the values as a graph

What I want the graph to do is to always draw the current value at the very right hand side of the display. This will achieve the effect of the data scrolling left with each new value.



To do this was not difficult in the end but to understand it may take a little explanation. Note that I solved it this way, another person might look at this problem and solve it in another (and even better) way. If my current reading is 80, then I want to draw the data points from 81 to 165 and 1 to 80 inthat order on my graph.



pixel	Data location in array
13	81
14	82
	163
	164
	165
	1
	2
	3
175	78
176	79
177	80

If you look at these two sequences you can see the pattern for my program is that it must lookup the data location which is the pixel location minus 13, plus current location(80) + 1.

This code does this

```
Tmp = Xpos - Graph_left
Tmp = Tmp + Curr_reading          'exceeds byte size
Incr Tmp
```

Of course we want to restart at 1 again after 65 so we add this as well

```
If Tmp > 165 Then Tmp = Tmp - 165
```

When I first wrote the program I declared Tmp as a byte, but that didn't work and I got a strange shifting of the display, I realised it was because tmp can actually get much larger than 255 before I subtract 165 from it.

The final part of the routine requires me to make sure that the display os blank before I draw data on it.
There are two ways(at least) that I could do this
I chose to draw a blank vertical line before I put the data at that point.

```
       Line(xpos , 0) -(xpos , 51) , 0  'remove anything on col already there
```

I also must plot the actual point.

```
       Ypos = 50 - T(tmp)                'turn the value into a position
       Pset Xpos , Ypos , T(tmp)         'set the pixel,if > 0
```

the display points 0,0 is the top left pixel on the display so I turn my temperature value into a location 50 degrees is at the top, (pixel 0) and 0 degrees is 50 pixels down the display (pixel 50)

Here is the complete loop

```
Draw_tempr_hum_graphs:
    'draw the two sets of data
    For Xpos = Graph_left To Graph_right
       Line(xpos , 0) -(xpos , 51) , 0  'remove anything on col already there
       Tmp = Xpos - Graph_left
       Tmp = Tmp + Curr_reading          'exceeds byte size
       Incr Tmp
       If Tmp > 165 Then Tmp = Tmp - 165
       Ypos = 50 - T(tmp)                'turn the value into a position
       Pset Xpos , Ypos , T(tmp)         'set the pixel,if > 0
       'Pset Xpos , H(xpos) , 1           'set the pixel
    Next
Return
```

46.7 Full software listing

```
'----------------------------------------------------------------------
' Title Block
' Author: Bill Collis
' Date:  June 2010
' File Name: HumidityTempLogV1a.bas
'----------------------------------------------------------------------
' Program Description:
'1  read temperature and humidity and display values
'1a setup graph scales
'   read multiple values and store in ram
'1b get storage and display working so that
'   data in array goes onto the display with the current reading last
'   e.g. if the curr_reading is stored at 125
'   then the display shows from 126 to 165 then 1 to 125
'
' Hardware Features:
' 128x64 GLCD on portB and 7 pins of portC
'
' lm35 temperature sensor on pinA.7
'  calibrated at adc=56 and temperature=28.2deg (0.282V)
'  56/28.2 = 1.9858
' humidity sensor HIH4030 on pin a.4
'  calibrated at adc=480 and voltage = 2.37
'  formula for hum=(V-0.78)/0.0306 - worked out from datasheet
'----------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m8535.dat"                  ' specify the used micro
$crystal = 8000000                      ' used crystal frequency
$lib "glcdKS108-192x64.lib"             ' library of display routines
'$noramclear
'----------------------------------------------------------------------
' Hardware Setups
Config Porta.4 = Input                  'ADC inputs
Config Porta.7 = Input                  'ADC inputs


'Configure GLCD interface
'CE      CS1 select      GLCD-pin15   CE      portC.3
'CE2     CS2 select2     GLCD-pin17   CE2     portC.5
'CE3     CS3 select3     GLCD-pin18   CE6     portC.6
'CD      RS              GLCD-pin4    CD      portC.0
'RD      RW              GLCD-pin5    RD      portC.1
'RESET   reset           GLCD-pin16   R       portC.4
'ENABLE  Chip Enable     GLCD-pin6    En      portC.2
Config Graphlcd = 192 * 64sed , Dataport = Portb , Controlport = Portc , Ce = 3 ,
Ce2 = 5 , Cd = 0 , Rd = 1 , Reset = 4 , Enable = 2 , Ce3 = 6
Config Adc = Single , Prescaler = Auto
Start Adc
'Hardware Aliases
'----------------------------------------------------------------------
' Declare Constants
Const Graph_left = 13
Const Graph_right = 177
'----------------------------------------------------------------------
' Declare Variables
Dim X As Byte
Dim Y As Long
Dim Hih4030 As Word
Dim Hum_single As Single                'single for fractional calculations
Dim Humidity As Byte
Dim Lm35 As Word
Dim Tempr_single As Single              'single for fractional calculations
Dim Tempr As Byte
Config Single = Scientific , Digits = 1
```

```
Dim T(165) As Byte                 '165 readings stored
Dim H(165) As Byte
Dim Arr_pos As Byte
Dim Curr_reading As Byte
Dim Xpos As Byte
Dim Ypos As Byte
Dim T_ypos As Byte
Dim H_ypos As Byte
Dim I As Byte
Dim Tmp As Word                             'temp variable
'initialise variables
Arr_pos = Graph_left                        'start here
Curr_reading = 1                            'start at 1st location in ram
'------------------------------------------------------------------
' Program starts here
'setup inital screen
Cls
Gosub Draw_graph_scales

Do
   Gosub Get_humidity
   Gosub Save_humidity
   Gosub Get_tempr
   Gosub Save_tempr
   Gosub Disp_humidity_val
   Gosub Disp_tempr_val
   Gosub Draw_tempr_hum_graphs
   Wait 300                                 'reading every 5 minutes
   Incr Curr_reading
   If Curr_reading > 165 Then Curr_reading = 1
Loop
End

'----------------------------------------------------------------
Save_humidity:
   H(curr_reading) = Humidity
Return
'----------------------------------------------------------------
Save_tempr:
   T(curr_reading) = Tempr
Return
'----------------------------------------------------------------
Draw_tempr_hum_graphs:
    'draw the two sets of data
    For Xpos = Graph_left To Graph_right
       Line(xpos , 0) -(xpos , 51) , 0  'remove anything on col already there
       Tmp = Xpos - Graph_left
       Tmp = Tmp + Curr_reading             'exceeds byte size
       Incr Tmp
       If Tmp > 165 Then Tmp = Tmp - 165
       Ypos = 50 - T(tmp)                   'turn the value into a position
       Pset Xpos , Ypos , T(tmp)            'set the pixel,if > 0
       'Pset Xpos , H(xpos) , 1              'set the pixel
    Next
Return

'----------------------------------------------------------------
Draw_graph_scales:
   Line(12 , 0) -(12 , 52) , 1              'left vertical
   Line(178 , 0) -(178 , 52) , 1            'right vertical
   Line(12 , 53) -(178 , 53) , 1            'bottom horizontal
   'left hand side humidity scale
   Setfont Font 8x8
   Lcdat 4 , 3 , "H"
   Pset 11 , 0 , 1
   Pset 11 , 10 , 1
```

```
   Pset 11 , 20 , 1
   Pset 11 , 30 , 1
   Pset 11 , 40 , 1
   Pset 11 , 50 , 1
   Setfont Font 5x5
   Line(0 , 0) -(0 , 4) , 1                  '1 in the 100 to save space
   Lcdat 1 , 2 , "00"
   Lcdat 7 , 0 , "50"

   'right hand side temperature scale
   Setfont Font 8x8
   Lcdat 4 , 182 , "T"
   Pset 179 , 0 , 1
   Pset 179 , 10 , 1
   Pset 179 , 20 , 1
   Pset 179 , 30 , 1
   Pset 179 , 40 , 1
   Pset 179 , 50 , 1
   Setfont Font 5x5
   Lcdat 1 , 181 , "50"
   Lcdat 7 , 181 , "0"
Return
'------------------------------------------------------------------
Get_humidity:
   Hih4030 = Getadc(4)                       'get raw adc value
   Hum_single = Hih4030                      'convert to single
   Hum_single = Hum_single / 203.85          'convert raw adc numbr to volts
   Hum_single = Hum_single - 0.78
   Hum_single = Hum_single / 0.0306
   Humidity = Hum_single                     'convert to byte for storage
Return

Disp_humidity_val:
   Setfont Font 8x8
   Lcdat 8 , 10 , Hum_single                 'single to display decimal value
   Lcdat 8 , 44 , "%"
Return
'------------------------------------------------------------------
Get_tempr:
   Lm35 = Getadc(7)                          'get the raw analog reading
   Tempr_single = Lm35                       'convert to single to use decimals
   Tempr_single = Tempr_single / 1.9858
   Tempr = Tempr_single                      'convert to byte for storage
Return

Disp_tempr_val:
   Setfont Font 8x8
   Lcdat 8 , 145 , Tempr_single
   Lcdat 8 , 176 , "C"
Return                                       '

'--------------------------------------------------------------------
'the font and graphic files must be in the same directory as the .bas file
'these lines put the fonts into the program flash
$include "font5x5.font"
'$include "font6x8.font"
$include "font8x8.font"
'$include "font16x16.font"
'$include "font32x32.font"
```

47 Interrupts

Microcontrollers are sequential devices, they step through the program code one step after another faithfully without any problem, and it is for this reason that they are used reliably in all sorts of environments. However what happens if we want to interrupt the usual program because some exception or irregular event has occurred and we want our micro to so something else briefly.

For example, a bottling machine is measuring the drink being poured into bottles on a conveyor. There could be a sensor connected to the conveyor which senses if the bottle is not there. When the bottle is expected but not there (an irregular event) the code can be interrupted so that drink is not poured out onto the conveyor.

All microcontrollers/microprocessors have hardware features called interrupts. There are two interrupt lines on the ATmega8535, these are pind.2 and pind.3 and are called Int0 and Int1. These are connected to switches on the development pcb. When using the interrupts the first step is to set up the hardware and go into a normal programming loop. Then at the end of the code add the interrupt subroutine (called a handler)

The code to use the interrupt is:

```
'----------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 9 Aug 2003
' Version: 1.0
' File Name: Interrupt_Ver1.bas
'----------------------------------------------------------------
' 2. Program Description:
' This program rotates one flashing led on portb
' when INT0 occurs the flashing led moves left
' when INT1 occurs the flashing led moves right
' 3. Hardware Features
' Eight LEDs on portb
' switches on INT0 and INT1
' 4. Software Features:
' do-loop to flash LED
' Interrupt INT0 and INT1
'----------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000              'the speed of operations inside the micro
$regfile = "m8535.dat"          ' the micro we are using
'----------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output
Config Pind.2 = Input 'Interrupt 0
Config Pind.3 = Input 'Interrupt 1

On Int0 Int0_handler 'if at anytime an interrupt occurs handle it
On Int1 Int1_handler 'if at anytime an interrupt occurs handle it
```

```
Enable Int0 Nosave 'enable this specific interrupt to occur
Enable Int1 Nosave 'enable this specific interrupt to occur
Enable Interrupts 'enable micro to process all interrupts
' 7. hardware Aliases
' 8. initialise ports so hardware starts correctly
'-----------------------------------------------------------------
' 9. Declare Constants
'-----------------------------------------------------------------
' 10. Declare Variables
Dim Pattern As Byte
Dim Direction As Bit
' 11. Initialise Variables
Pattern = 254
Direction = 0
'-----------------------------------------------------------------
' 12. Program starts here
Do
   If Direction = 1 Then
       Rotate Pattern , Left
       Rotate Pattern , Left
   Else
      Rotate Pattern , Right
      Rotate Pattern , Right
   End If
   Portb = Pattern 'only 1 led on
   Waitms 150
   Portb = 255 ' all leds off
   Waitms 50
Loop
'-----------------------------------------------------------------
' 13. Subroutines
'-----------------------------------------------------------------
' 14. Interrupt subroutines
Int0_handler:
   Direction = 1
Return

Int1_handler:
   Direction = 0
Return
```

Note that enabling interrupts is a 2 step process both the individual interrupt flag and the global interrupt flag must be enabled.

Exercise

Change the program so that only one interrupt is used to change the direction.

With the other interrupt change the speed of the pattern.

47.1 Switch bounce problem investigation

Most peole don't have an oscilloscope at home to investigate switch bounce but its effects can be seen in programs. Connecting a poor quality press button switch to portB.2 on an ATMega64 running at 8MHz and running this program reveals what happens with contact or switch bounce.

The interrupt is setup so that when PINB.2 goes low an interrupt occurs. This should happen when the switch is pressed but not released. When an INT2 occurs a counter value is increased. The main program loop just sits there displaying the value of count and if a switch on PINB.0 is pressed reset the count to 0.

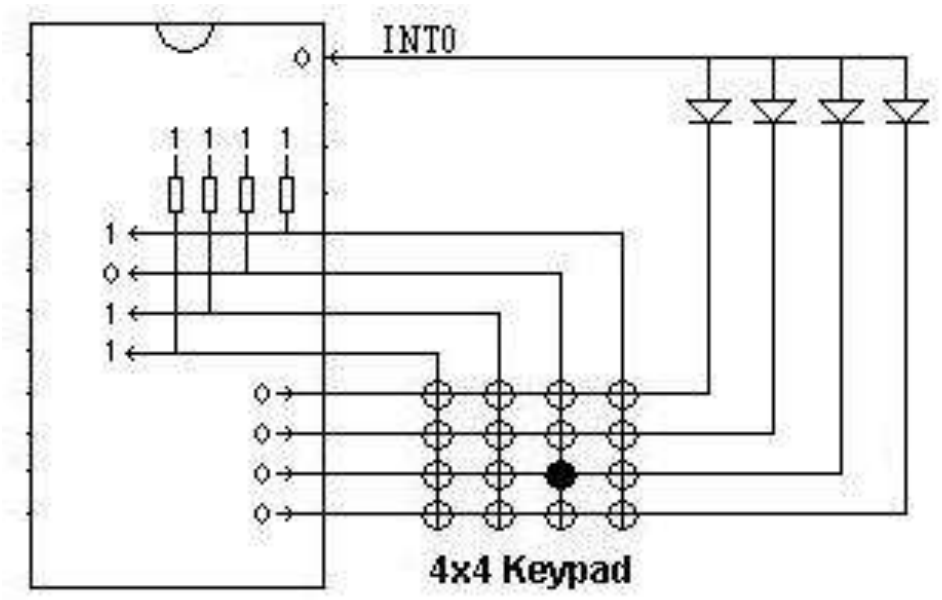
```
'debounce test program
$regfile = "m644def.dat"
$crystal = 8000000

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E =
Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4

Config Portb = Input
Set Portb.0                                 'pullup resistor on
Set Portb.1                                 'pullup resistor on
Set Portb.2                                 'pullup resistor on
Set Portb.3                                 'pullup resistor on
Set Portb.4                                 'pullup resistor on

'Interrupt INT2
'this code enables an interrupt on pin INT2
Config Int2 = Falling                       'reset count
On Int2 Int2_isr
Enable Int2
Enable Interrupts

Dim Count As Byte
Cls
Cursor Off

Lcd "debounce test"
Do
      If Pinb.0 = 0 Then Count = 0
      Locate 2 , 1
      Lcd "decimal=" ; Count ; "  "
      Locate 3 , 1
      Lcd "binary =" ; Bin(count)
      Locate 4 , 1
      Lcd "hex    =" ; Hex(count)
Loop
End

'Interrupt service routine - program comes here when int2 pin goes low
Int2_isr:
   Incr Count
Return
```

The results of this program show how poor quality the switch actually is. A single firm press of the switch will increase the count by as much as 16 or more, a soft press of the switch can increase the count by hundreds. In addition to this when the switch is released te variable count also increases as the countacts bounce when they come apart. Results from 10 trials of a single press and release were 11, 117, 29, 36, 59, 102, 29, 15, 9, 27.

47.2 Keypad- polling versus interrupt driven

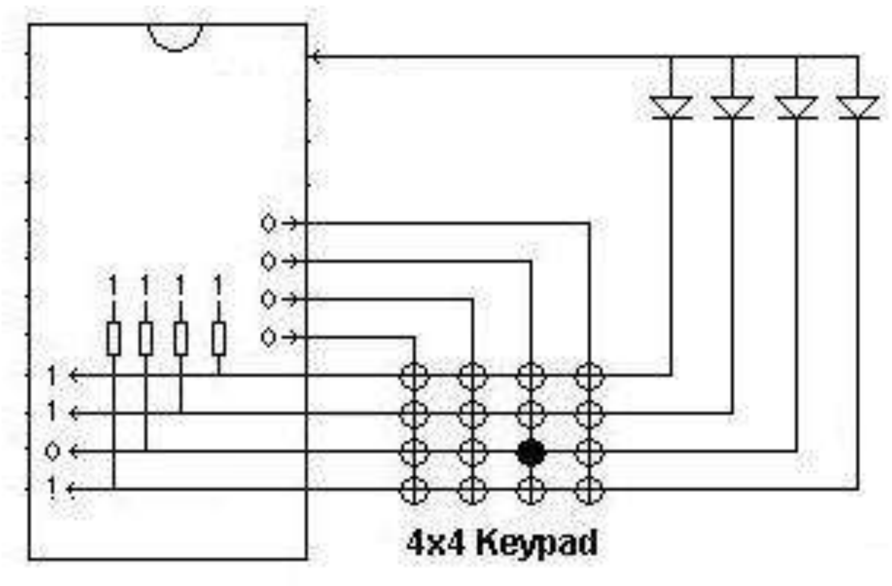
With the earlier keypad circuits we have had to poll (check them often) to see if a key has been pressed.

It is not always possible however to poll inputs all the time to see if they have changed it can be much easier using an interrupt.

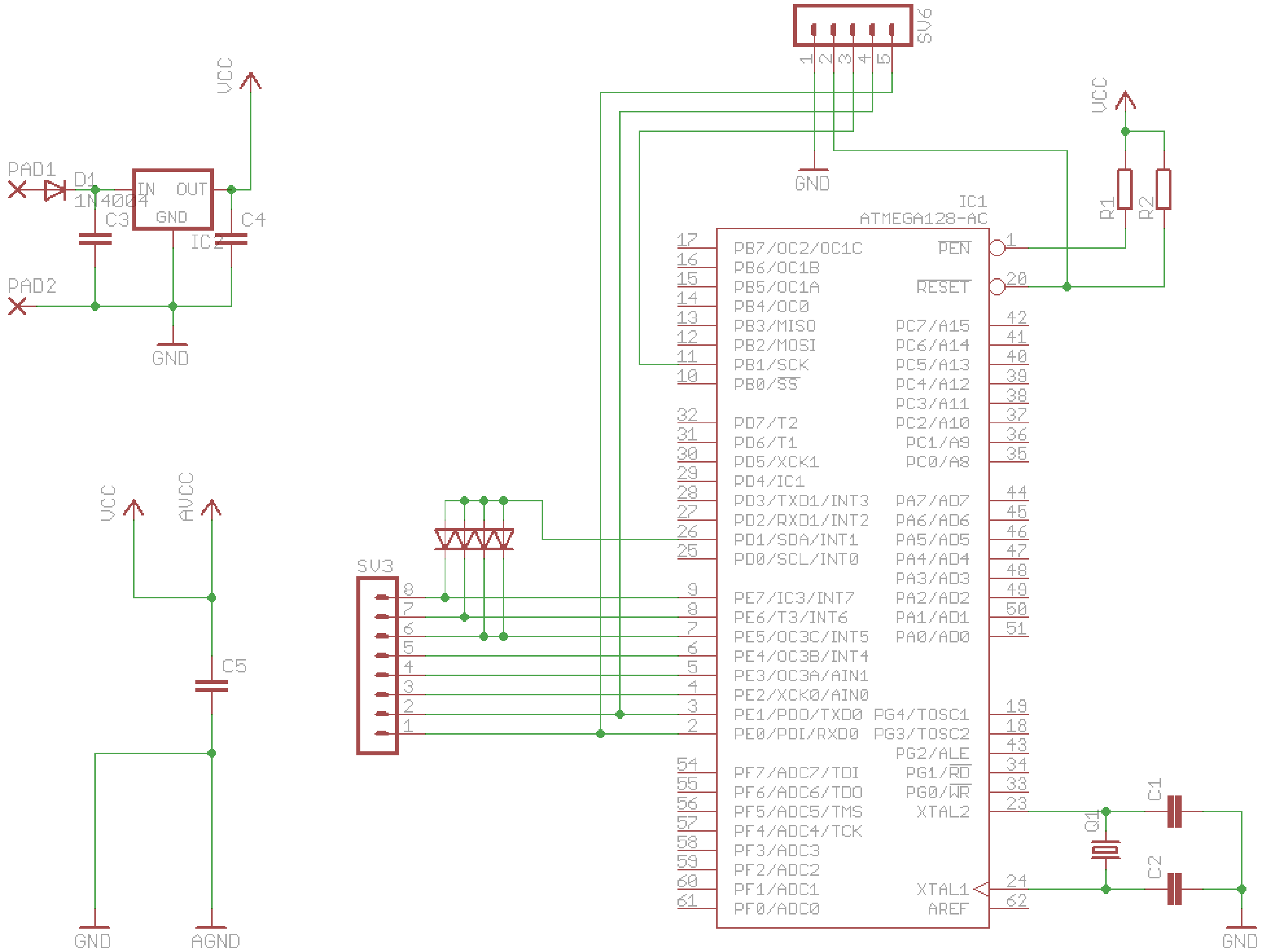


In this circuit 4 pins are configured as outputs and 4 as inputs, when a keypad button is pressed down the 0 on the output pulls the diode down triggering the interrupt.

In the interrupt routine the inputs are read to identify which pin is 0. Then the inputs become outputs and the outputs become inputs. The outputs are driven low and one of the inputs will become low. This combination is unique and identifies which key was pressed.



Here is the circuit diagram for an ATMega64 with the keypad circuit shown



Program code for this keypad

```
'-----------------------------------------------------------------
' Title Block
' Author:B.Collis

' File Name: kybd_v2.bas
'-----------------------------------------------------------------
' Program Description:
' This program reads a keypad using interrupts rather than polling
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                           ' internal clock
$regfile = "m64def.dat"                      ' ATMEGA64-16AI
'-----------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portc = &B11111111                    '1=output 0=input

Config Lcdpin = Pin , Db4 = Portb.4 , Db5 = Portb.5 , Db6 = Portb.6 , Db7 = Portb.7 , E = Portb.3 , Rs
= Portb.2
Config Lcd = 20 * 4                          'configure lcd screen
'the keypad interrupt
Config Pind.1 = Input                        'int INT0
Config Int1 = Falling                        'negative edge trigger
On Int1 Int1_int                             'go here on interrupt
Enable Interrupts                            'global interrupts on

'Hardware Aliases
Keypad_int Alias Pind.1                      'not used
Keypad_out Alias Porte
Keypad_dir Alias Ddre
Keypad_in Alias Pine
'Initialise hardware state
Keypad_dir = &B00001111                      ' upper half of port input=0, lower half output=1
Keypad_out = &B11110000                      ' enable pullups upper 4 bits, lower half port to 0
'-----------------------------------------------------------------
'Declare Constants
Const Timedelay = 450
Const Debouncetime = 20

'Declare Variables
Dim Keyrow As Byte
Dim Keycol As Byte
Dim Keycode As Byte
Dim Lastkey As Byte                          'the last key that was pressed
Dim Keyval As Byte                           'the extended value of the key that has just been pressed
Dim Keycount As Byte                         'records how may times the key has been pressed
Dim Keychar As String * 1                    'the character gotten from the keypad
Dim Intcount As Word
Dim Keypress As Bit
'Initialise Variables
Intcount = 0
Keychar = "r"
Keypress = 0                                 'no key down
```

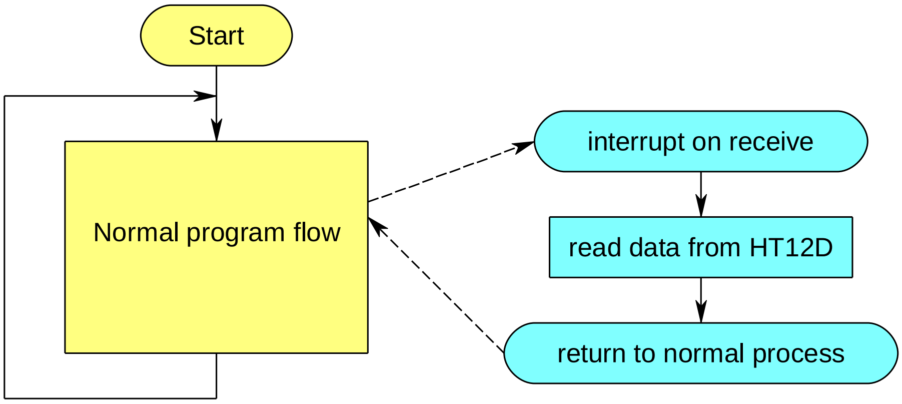
```
'-----------------------------------------------------------------
'Program starts here
Reset Porta.0                              'led on
Cls
Lcd "ATMEGA64-16Ai"
Lowerline
Lcd "keypad reader:"
Locate 3 , 1
Lcd "I_ctr="
Locate 3 , 10
Lcd "code="
Locate 4 , 1
Lcd "col="
Locate 4 , 10
Lcd "row="
Enable Int1
Do
  Locate 3 , 7
  Lcd Intcount : Lcd " "
  Locate 3 , 15
  Lcd Keycode : Lcd " "
  Locate 4 , 5
  Lcd Keycol : Lcd " "
  Locate 4 , 14
  Lcd Keyrow : Lcd " "
  Locate 2 , 16
  Lcd Keychar
  Toggle Porta.6
  Waitms Timedelay
  Toggle Porta.7
  Waitms Timedelay
Loop
End                                   'end program
```

```
'-----------------------------------------------------------------
' Interrupts
Int1_int:
  Toggle Porta.0                              'indicate a key press
  Incr Intcount                              'tally of key presses
  Keypress = 1                                'flag a key down
  Keycol = Keypad_in
  'swap port upper nibble to input, lower to output
  Keypad_dir = &B11110000
  Keypad_out = &B00001111
  Waitms 1                                   ' port needs a little time
  Keyrow = Keypad_in                               'read the col is zero
  'set port back to original state
  Keypad_dir = &B00001111
  Keypad_out = &B11110000
 'make keycode from port pins read
  Shift Keycol , Right , 4
  Select Case Keycol
    Case 7 : Keycode = 0
    Case 11 : Keycode = 4
    Case 13 : Keycode = 8
    Case 14 : Keycode = 12
    Case Else : Keycode = 99
  End Select
  'make final keycode from port pins read
  Select Case Keyrow
    Case 7 : Keycode = Keycode + 0
    Case 11 : Keycode = Keycode + 1
   Case 13 : Keycode = Keycode + 2
    Case 14 : Keycode = Keycode + 3
    Case Else : Keycode = Keycode + 99
  End Select
  'illegal keycode from bounce effects
  'If Keycode > 15 Then Keycode = 16
  Keychar = Lookupstr(keycode , Keycodes)
  'the changing of ports causes interrupts to be flagged a second time
  'however interrupts are not processed during an intr routine
  '    because the global flag is halted (CLI)
  'so we must clear the second interrupt so that we do not enter here again
  'this took a few hours to figure this one out!!!
  'this line clears any pending interrupts before the routine exits
  Eifr = 2
Return
'-----------------------------------------------------------------
Keycodes:
Data "1" , "4" , "7" , "s" , "2" , "5" , "8" , "0" ,
Data "3" , "6" , "9" , "h" , "A" , "B" , "C" , "D" , "?"
```

47.3 Improving the HT12 radio system by using interrupts

Earlier a radio system was described that used the HT12E and HT12D ICs. The receiver side of the system used a polling type design, where the program regularly checked the VT pin from the HT12D to see if data was present.

It would be useful in some situations to have an interrupt driven design, so that a program could be doing other functions and only respond when something actually happens.



In this program the data is stored when it arrives and the main program loop is free to check it when it wants.

```
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                          ' internal clock
$regfile = "m16def.dat"                     ' ATMEGA16
'-------------------------------------------------------------------
' Hardware Setups
' setup direction of all ports
Config Porta = Output                       '4 leds on PortA.0to A.3
Config Portb = Input                        ' Valid data is input on this port
Config Portc = Output                       ' Used for LED's and LCD
Config Portd = Input                        ' PortD.2 is used for Data  Valid input

'setup LCD
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 =
Portc.7 , E = Portc.3 , Rs = Portc.2
Config Lcd = 40 * 2

'setup Interrupts
On Int0 Get_data
Enable Int0
Config Int0 = Rising
Enable Interrupts

'  Hardware Aliases
Ht12d_dv Alias Pind.2

' Turn off LED's on PortC.0 & PortC.1
'-------------------------------------------------------------------
' Declare Constants
Const True = 1
Const False = 0
' Declare Variables
Dim Rcvd_value As Byte
Dim Data_rcvd_flag As Bit
Dim Data_rcvd_count As Byte
Dim Message As String * 81
```

```
'  Initialise Variables
'-----------------------------------------------------------------
'  Program starts here
Cls
Cursor Off
Locate 1 , 1
Lcd "HT12D interrupt test program"


Do
   'do other program stuff here
   If Data_rcvd_flag = True Then        'do something with the new data
      Porta = Not Rcvd_value            'display on leds
      Message = Lookupstr(rcvd_value , Messages)
      Cls
      Lcd Message
      Data_rcvd_flag = False            'remove flag
   End If
Loop                                    '
End

'-----------------------------------------------------------------
'interrupt routine
Get_data:
   Data_rcvd_flag = True
   Rcvd_value = Pinb And &H0F           ' get value from lower nibble PortB
   While Ht12d_dv = True                ' wait until data no longer valid
   Wend                                 'so that the program only actions data
once
Return

'---------------------------------------- ---------------------------------
Messages:
Data "The only time success comes before work        is in the dictionary!!"
Data "         Ma Te Mahi Ka Ora                Fulfillment comes through hard
work!"
Data "good decisions come from experience        experience comes from bad
decisions"
Data "         the trouble with normal                is it only gets worse!"
Data "       What you do speaks so loud        that I cannot hear what you are
saying"
Data "Never confuse motion with action"
Data "The only thing necessary for the triumph  of evil is for good men to do
nothing"
Data "Ability is what you're capable of doing  Attitude determines if you will do
it"
Data "      The first will be last                and the last will be first"
Data "If a blind person leads a blind person,     both will end up in a ditch"
Data "10"
Data "11"
Data "12"
Data "13"
Data "14"
Data "15"
```

The limitation of this program is that it only stores one piece of data; and if new data arrives before it has had an opportunity to process the first value, then the first value is lost. This program could do with a buffer to remember received data, in fact a queue would be useful, where data arrives it is stored at one end of the queue and it is processed from the other end . In computer programming terms its called a First In First Out (FIFO) queue or buffer.

47.4 Magnetic Card Reader

The JSR-1250 is only a few dollars and can make the basis for a neat project involving magnetic cards.

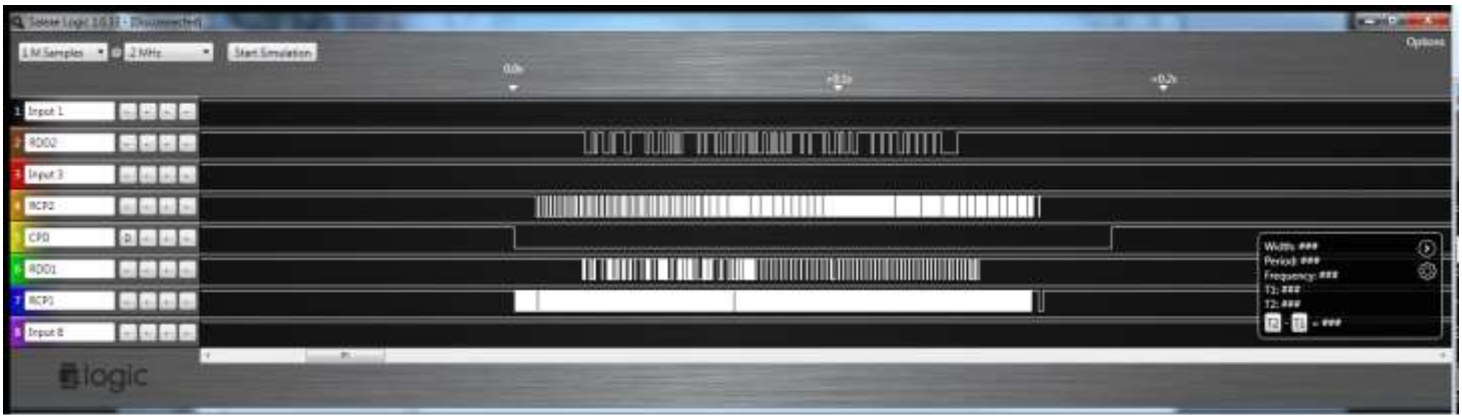
The card reader has 5V and ground/0V power supply pins as well as 5 interface pins. This is how the interface pins were connected (each pin also had a 4k7 pullup resistor connected to VCC).
RDD2 onto Pind.6 (data 2)
RCP2 onto Pind.2 - INT0 (clock pulse 2)
CPD onto Pind.3 - INT1 (card present detect)
RDD1 onto Pind.4 (data 1)
RCP1 onto Pind.5 (clock pulse 1)

47.5 Card reader data structure

Before program code can be written it must be planned, AND before it can be planned the hardware must be understood in fine detail.
A card was swiped upwards through the reader and using a logic analyzer the data was captured.
Note the following:

- CPD is high when there is no data.
- When a card is swiped CPD goes low and remains low during the complete data send process.
- There are two sets of data (RDD1 And RDD2) and their respective clock signals (RCP1 and RCP2).

We can use all this information when writing code to understand the incoming data.

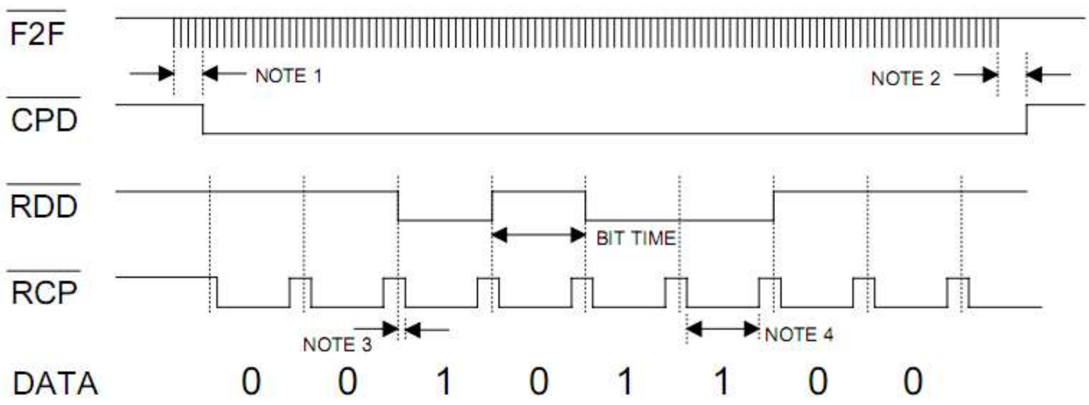


47.6 Card reader data timing

There is still much more to understand. When writing program code to read the data from a magnetic card reader it is important to understand exactly when the data is valid. This is a synchronous data transfer process, which means that two signals are sent both clock and data, and we must know when to read the data in relation to the level of the clock data.

The datasheet has this diagram in it and explains that the data should be read when the clock goes from high to low (its negative edge).

Timing Chart

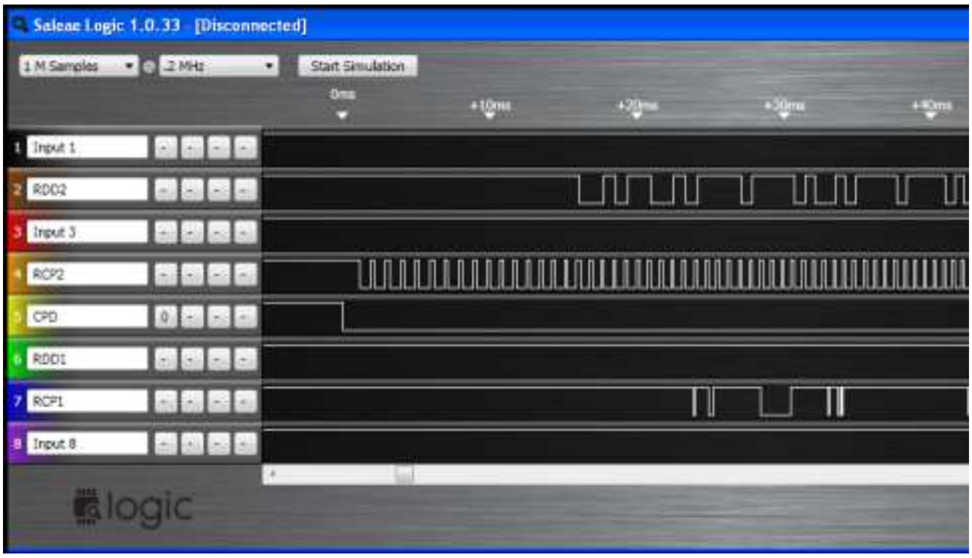


NOTES:
1. 8 or 9 head flux reversals for low density configuration.
2. TIMEOUT of the CPD signal approx. 50m Sec after last Head Signal transition.
3. The RDD is valid at 1.6μ Sec(min.) before negative edge of the RCP.
4. The low pulse width of RCP is approx. 70% of the bit time.

RDD The DATA signal is valid while the RCP is low. If the RDD signal is high, the bit is zero. And If low, the bit is one

RCP The RCP signal indicates that RDD is valid. The RDD should be loaded by the user when the RCP signal goes low.(Negative edge)

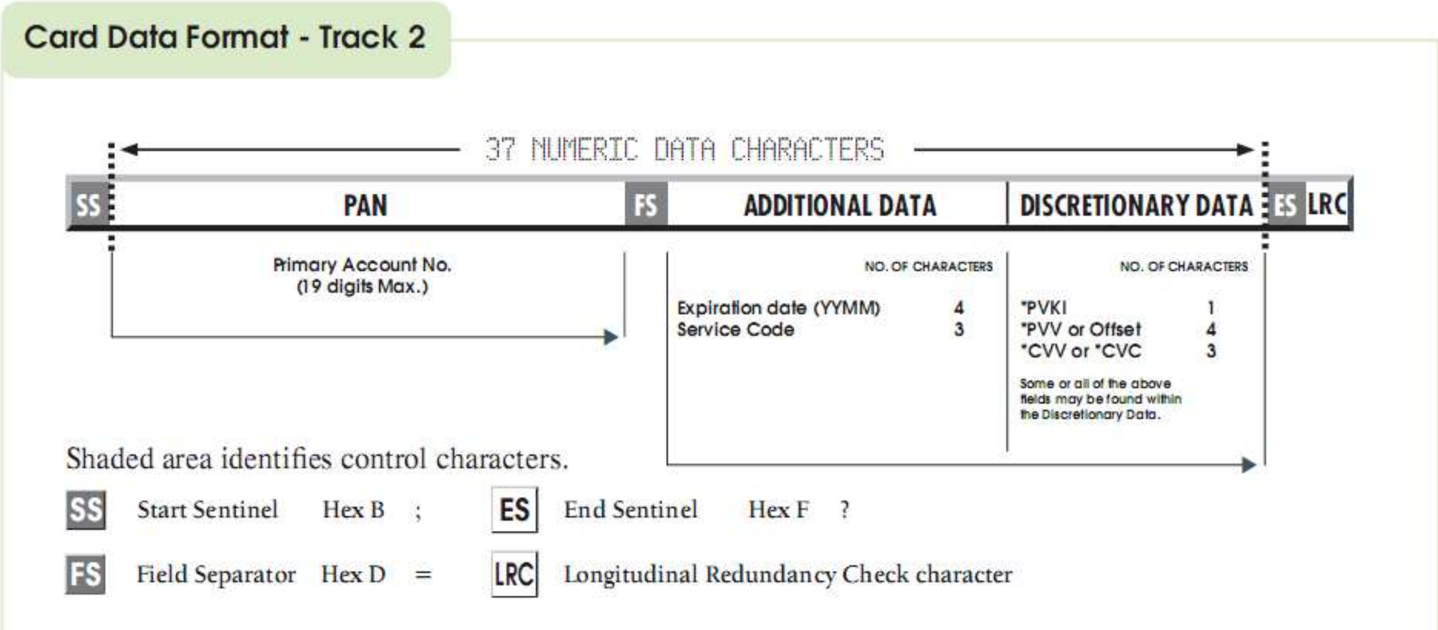
CPD CPD signal will go to low after the 8 or 9th flux reversal and will return to high when the 20m Sec approx. was elapsed.



In this screen capture from the logic analyser it can be seen that there is a gap of around 15mS between CPD going low and the data starting.

47.7 Card reader data formats

Next we must know how the binary data (1's and 0's) needs to be put back into information we can use (numbers such as credit card numbers!). There are many sources of information on the internet about magnetic card readers, perhaps one of the best is http://stripesnoop.sourceforge.net/devel/index.html. On this site are documents that explain in quite a lot of detail the number of tracks of data on a card and its format. There are two tracks available from our reader, 1 and 2.
Here is the track 2 data format.
It has a start sentinel (signal), then 19 digit code, then... as per the diagram



Further research on the web leads to the format that the data is in. The data is sent 5 bits at a time, 4 data bits and 1 parity bit (error checking). The data comes in LSB (least significant bit) first. The number 3 in binary is 0011; this means that a 1 is sent then another 1 then a 0 then another 0; and then the parity bit is sent.

47.8 Understanding interrupts in Bascom- trialling

The tricky thing with Bascom and interrupts is that Bascom does not give us complete control over how the interrupts are configured, and there are a number of features in the AVR that we can make use of. In the AVR we can actually configure the interrupts to be negative edge, positive edge, both edge or low level detect.

Bascom configures the interrupt to be level detected, so interrupts occur when the pin goes low and continue to occur while it is low. In this program an edge rather than a level detection is better. We only want one interrupt to occur on the edges.

Here is how the interrupts are configured by Bascom (level detection).

```
On Int1 Int1_cpd                        'card present detect
Enable Int1                             'enable card detect interrupt
Enable Interrupts                       'enable micro to process all interrupts
```

However this is not what we need; to figure out the settings the datasheet was downloaded and the sections on interrupts and external interrupts were read. The interrupts are controlled by registers (memory locations which directly control hardware) within the micro, so a program was then written to display all of the register values involved with interrupts.

```
Lcdat 1 , 1 , "8535 Interrupt Testing"
Lcdat 2 , 1 , Sreg
Lcdat 3 , 1 , Gicr
Lcdat 4 , 1 , Gifr
Lcdat 5 , 1 , Mcucsr
Lcdat 6 , 1 , Mcucr
```

Here are the results of displaying the values of the registers.

Register	Value	Meaning
SREG Status Reg	&B10000010	This register is the status register for the whole AVR, we are only interested in bit 7, which is the global interrupt flag. If we set this to 1 then any enabled interrupt will occur, if it is reset to 0 then any enabled interrupts will not occur (hence the name global interrupt flag). We can set it by using any one of the following commands in Bascom enable interrupts or SEI or set SREG.7
GICR General Interrupt Control Reg	&B10000000	This register is used to control the external interrupts. We can disable INT0 using the following commands disable INT0 or RESET GICR.INT0 We can enable INT1 using the following commands enable INT1 or SET GICR.INT1
GIFR General Interrupt Flag Reg	&B00100000	We don't set or reset any of the bits in this register
MCUCSR MCU Control Status Reg	&B00000011	We don't set or reset any of the bits in this register
MCUCR MCU Control Reg	&B00000000	The type of interrupt is set with this register, we are really interested in this. When we write in Bascom ''On INT1 int1_cpd' Bascom configures 2 bits of this register, ISC11 and ISC10, and it resets them to 0, meaning low level interrupt is configured. We really want an interrupt on both the negative edge and positive edge of this pin. So we write these 3 lines On INT1 int1_cpd 'Bascom sets up the interrupts for us Reset MCUCR.ISC11 'we modify the type of interrupt Set MCUCR.ISC10 When we write in Bascom ''On INT0 int0_rcp2' Bascom configures 2 bits of this register, ISC01 and ISC00, and it resets them to 0, meaning low level interrupt is configured. We really want a negative (falling) edge interrupt on the clock so we write these 3 lines On INT0 int0_rcp2 'Bascom sets up the interrupts for us Set MCUCR.ISC01 'we modify the type of interrupt Reset MCUCR.ISC00 After doing this MCUCR = &B00000110

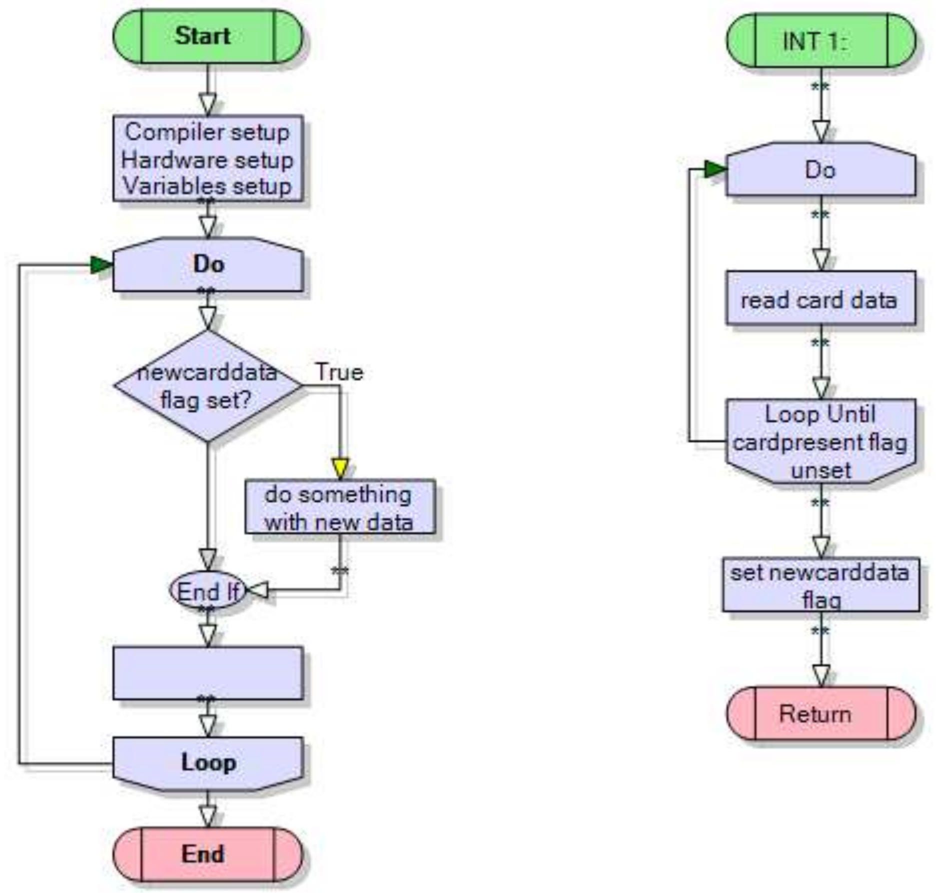
Initially the CPD (card present) interrupt is enabled and the clock (RCP) interrupt is disabled. When a card is present (CPD goes low) the interrupt routine is used to enable the clock interrupt. When the clock goes low we will read the data.

An initial program to test the ideas was created. This program detects the positive and negative edges on the CPD (card present detect) and counts them, it then counts the number of clock pulses.

```
'-----------------------------------------------------------------
' File Name: MagReaderV1a.bas
' Program Description:
' uses interrupts to read the data from a magnetic card
' Hardware Features:
' 128x64 GLCD
' JSR-1250 magnetic card reader
' waits for int0 (CPD) then enables int1(RCP)
' every swipe the number of clocks is counted
'-----------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m8535.dat"              ' specify the used micro
$crystal = 8000000                  ' used crystal frequency
$lib "glcdKS108-192x64.lib"         ' library of display routines
'-----------------------------------------------------------------
' Hardware Setups
Config Portd = Input                'Mag card
Config Portc = Input                'switches
'Configure KS0108 GLCD interface
Config Graphlcd = 192 * 64sed , Dataport = Portb , Controlport = Porta , Ce = 3 , Ce2 = 5 , Cd = 0 , Rd = 1 ,
Reset = 4 , Enable = 2 , Ce3 = 6
'interrupt setups - NOTE the special configs
On Int1 Int1_cpd                    'card present detect
Reset Mcucr.isc11                   'change to both edges interrupt detect
Set Mcucr.isc10

On Int0 Int0_rcp2                   'read clock pulse
Set Mcucr.isc01                     'change to negative edge detect
Reset Mcucr.isc00

Disable Int0                        'disable clock pulse interrupt
Enable Int1                         'enable card detect interrupt
Enable Interrupts                   'enable micro to process all interrupts
'Hardware Aliases
Rdd2 Alias Pind.6                   '&B00000000
Rcp2 Alias Pind.2                   'int0
Cpd Alias Pind.3                    'int1
Rdd1 Alias Pind.4
Rcp1 Alias Pind.5
' Declare Variables
Dim Positive_edge As Byte
Dim Negative_edge As Byte
Dim Clock_count As Word
' Program starts here
Cls
Setfont Font 8x8                    'specify the small font
Lcdat 1 , 1 , "Magnetic card reader"    'the rows are from 1 to 8
Do
   Lcdat 2 , 1 , Positive_edge
   Lcdat 3 , 1 , Negative_edge
   Lcdat 4 , 1 , Clock_count
Loop
End                                 'end program
'the font and graphic files must be in the same directory as the .bas file
$include "font8x8.font"
'-----------------------------------------------------------------
'interrupts
'card detect  - both edges generate an interrupt
Int1_cpd:
   If Cpd = 0 Then                  'while low we want to collect data
      Incr Positive_edge            'keep track of the negative edges
      Enable Int0                   'allow clock interrupts
      Clock_count = 0                'start clock counter from 0
   Else
      Incr Negative_edge            'keep track of positive edges
      Disable Int0                  'finidhed so stop data collection
   End If                           '
Return
'clock - negative edge interrupt
Int0_rcp2:
   Incr Clock_count                 'keep track of number of clocks per swipe
Return
```

47.9 Planning the program

In this first example it was decided that a single interrupt would be sufficient and it would be used to capture the CPD. In the interrupt routine program code has been written that reads data from the card reader.



It should be noted here that it is considered bad practice to put lengthy code inside an interrupt routine. It can cause the micro to crash if interrupts occur during the processing of an interrupt and further interrupts occur during that interrupt. The micro has to keep track of all interrupts and has only a finite amount of memory space to do this; too many interrupts inside others and your progam easily crashes.

If this is understood and the rest of the program is written with this in mind then it will be ok; but in a big project where multiple people are writing different parts of a program this would be bad to do.

```
'------------------------------------------------------------------
' Title Block
' Author: B.Collis
' Date: April 2011
' File Name: MagReaderV3a.bas
'------------------------------------------------------------------
' Program Description:
' uses interrupts to read the data from a magnetic card
' Hardware Features:
' 128x64 GLCD
' JSR-1250 magnetic card reader
' 3a - when card is swipped, int1 occurs
'      all data is read inside the int routine
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m8535.dat"                ' specify the used micro
$crystal = 8000000                    ' used crystal frequency
$lib "glcdKS108-192x64.lib"           ' library of display routines
'------------------------------------------------------------------
' Hardware Setups
Config Portd = Input                  'Mag card
Config Portc = Input                  'switches
Set Portc.1                           'activate internal pullup resistor
Set Portc.2                           'activate internal pullup resistor
```

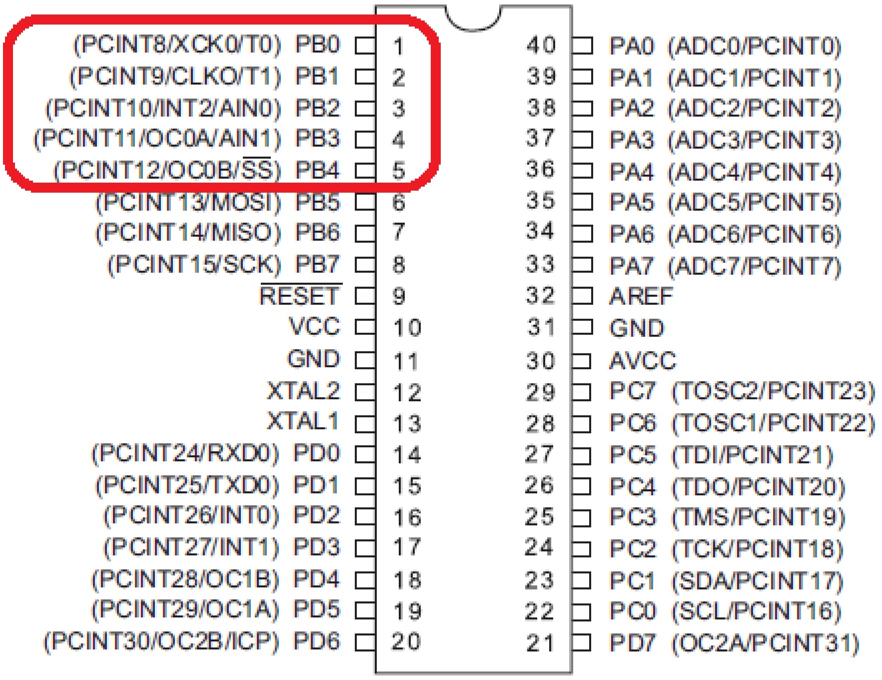
```
Set Portc.3                             'activate internal pullup resistor
Set Portc.4                             'activate internal pullup resistor
Config Portc.0 = Output                 'led
Config Portd.7 = Output                 'led
'Configure KS0108 GLCD interface
Config Graphlcd = 192 * 64sed , Dataport = Portb , Controlport = Porta , Ce = 3 , Ce2 = 5 , Cd = 0 , Rd = 1 ,
Reset = 4 , Enable = 2 , Ce3 = 6

'interrupt setups - NOTE the special configs
On Int1 Int1_cpd                        'card present detect
'bascom configures the int to level detect,
' so ints are continuously generated while int1 is low
' we want neg edge int only so set the appropriate bits
Set Mcucr.isc11                         'change to negative edge detect
Reset Mcucr.isc10
Enable Int1
Enable Interrupts

'Hardware Aliases
'switches
Yel_sw Alias Pinc.1
Blk_sw Alias Pinc.2
Blu_sw Alias Pinc.3
Red_sw Alias Pinc.4
Or_led Alias Portc.0
Yel_led Alias Portd.7
Reset Yel_led
Reset Or_led

'magnetic card reader
Rdd2 Alias Pind.6                       'data 2
Rcp2 Alias Pind.2                       'clock pulse2
Cpd Alias Pind.3                        'int1  -card present detect
Rdd1 Alias Pind.4
Rcp1 Alias Pind.5

'------------------------------------------------------------------
'constants
Const Fl_nocpd = 1                      'no card detected
Const Fl_cpd = 2                        'card detected
Const Fl_newcard = 3                    'new card info to process

' Declare Variables
Dim Flag As Byte
Dim Temp As Byte
Dim Tempstr As String * 30
Dim Carddata As String * 52
Dim Bit_counter As Byte
Dim Count As Byte
Dim Byte_counter As Byte
' Initialise Variables
Flag = Fl_nocpd

'------------------------------------------------------------------
' Program starts here
Cls
Setfont Font 5x5                        'specify the small font
Lcdat 1 , 1 , "Mag card reader Ver2a"   '
Lcdat 2 , 1 , "Swipe a card upwards"

Do
   If Flag = Fl_newcard Then            'do something with the new info
      Tempstr = Left(carddata , 30)
      Lcdat 4 , 1 , Tempstr
      Tempstr = Mid(carddata , 30 , 20)
      Lcdat 5 , 1 , Tempstr
      Flag = Fl_nocpd                   'no card detected
   End If
   'rest of program goes here
Loop
End                                     'end program



'------------------------------------------------------------------
'the font and graphic files must be in the same directory as the .bas file
$include "font5x5.font"
'------------------------------------------------------------------


'interrupts
'card detect  - negative edge generates an interrupt
'------------------------------------------------------------------
```

```
'this routine is called when there is a CPD interrupt(card present)
'  with no card swiped the flag is Fl_nocpd
' when CPD goes low INT1 happens
'  flag is set to Fl_cpd,   at this time RDD is high
'  wait for first neg edge of RDD
'  process edge
'  wait for both new neg edge and CPD
'  if CPD exit , if neg edge process new data bit
'  processing data:
'  after 5 data bits, a new byte is created with the data in it
'  data comes in the form of 4 inverted bits (LSB first) + parity

Int1_cpd:
   If Cpd = 0 Then                      'neg edge,card detected
      Flag = Fl_cpd                     'reading a card
      Set Or_led                        'show an indicator led
      Carddata = ""                        'delete any previously read card data
      Bit_counter = 0                      'reset bit counter
      Byte_counter = 1
      Do                                   'if cpd=1 then start reading data
         'wait for data to start
         Do
            If Cpd = 1 Then Exit Do     'card finished so dont get stuck
         Loop Until Rdd2 = 0
         'process all incoming data until CPD goes high at end of read
         Do
            Set Yel_led
            'wait for clock to go low
            Do
               If Cpd = 1 Then Exit Do  'card finished so dont get stuck
            Loop Until Rcp2 = 0

            'process a single bit
            If Bit_counter < 4 Then        'only store bits 0 to 3
               Temp.bit_counter = Not Rdd2 'get value of input, negate and store
            End If

            If Bit_counter = 4 Then        '5 bits completed
               Bit_counter = 255           '255 because we incr it after this to 0
               'add code to check parity??? - not really necessary
               Temp = Temp + 48            'convert to asci code
               Carddata = Carddata + Chr(temp)        'store the data
               Temp = 0                    'reset for next 5 bit read
               Incr Byte_counter           'next store location
            End If
            Incr Bit_counter

            'wait for RCP to return high
            Do
               If Cpd = 1 Then Exit Do  'card finished so dont get stuck
            Loop Until Rcp2 = 1            'clock has returned high
         Loop Until Cpd = 1
      Reset Yel_led
      Loop Until Cpd = 1                   'will be set by int routine
   Flag = Fl_newcard
      Reset Or_led
   End If
Return
```

47.10 Pin Change Interrupts PCINT0-31

Each modern AVR microcontroller has a number of other external interrupts known as Pin Change Interrupts (PCI). Here the interrupt is triggered when the pin changes, so that means either from 1 to 0 ot 0 to 1.

In the datasheet for each micro they are labelled.



There is not an interrupt for each pin, they are arranged into groups of 8 which share one interrupt. So there are only 4 pin change interrupts PCINT0, PCINT1, PCINT2, PCINT3 in the ATMEGA644.
Try not to confuse PCINT0 the interrupt pin PortA.0 with PCINT0 the interrupt!!!

In our program we will make use of 5 switches on pins B.0 thru B.4 (PCINT8 thru PCINT12) which uses PCINT1 (pin change interrupt 1)

So before we use any of the interrupts we need to tell the micro which of the 8 pins on PORTb we want to trigger PCINT1. We don't want any changes on pinb.5, 8.6 or b.7 to cause interruts, so we mask them out using PCMSK1.

```
Pcmsk1 = &B00011111                'only use pcint8-pcint12  (pinb.0-pinb.4)
On Pcint1 Isr_pcint1               'jump here when one of the pins is changed
Enable Pcint1                      'must enable pcint1
Enable Interrupts                  'global interupt flag
```

In the ISR (interrupt device routine) we need to figure out which of the 5 pins actually caused the interrupt and then take the right acton.

Also note that these are pinchange interrupts so if you press a switch you get an interrupt and when you release the swiutch you get another interrupt; and all the switch bounces inbetwen cause more interrupts.

```
'PCINT test program
$regfile = "m644def.dat"
$crystal = 8000000

Config Lcdpin=pin , Db4 = PORTC.2 , Db5 = PORTC.3 , Db6 = PORTC.4 , Db7 = PORTC.5 , E = PORTC.1 , Rs = PORTC.0
Config Lcd = 20 * 4
Config Portb = Input
Set Portb.0                             'pullup resistor on PCINT8
Set Portb.1                             'pullup resistor on PCINT9
Set Portb.2                             'pullup resistor on PCINT10
Set Portb.3                             'pullup resistor on PCINT11
Set Portb.4                             'pullup resistor on PCINT12

'With pcmsk you activate which pins will respond to a change on the pin
'When you write a 1, the change in logic level will be detected.
Pcmsk1 = &B00011111                     'only use pcint8-pcint12  (pinb.0-pinb.4)
On Pcint1 Isr_pcint1                    'jump here when one of the pins is changed
Enable Pcint1                           'must enable pcint1
Enable Interrupts                       'global interupt flag

Dim count As Byte

Cls
Cursor Off

Lcd "PCINT test"
Do
  Locate 2 , 1
  Lcd "decimal=" ; Count ; "  "
  Locate 3 , 1
  Lcd "binary =" ; Bin(count)
  Locate 4 , 1
  Lcd "hex    =" ; Hex(count)
Loop
End

Isr_pcint1:
   'to find out which pin changed we test each pin
   Waitms 20                            'debounce cheap switches
   If Pinb.0 = 0 Then
      Decr Count
      Do
      Loop Until Pinb.0 = 1
   End If
   If Pinb.1 = 0 Then
      Incr Count
      Do
      Loop Until Pinb.1 = 1
   End If
   If Pinb.2 = 0 Then
      Count = Count * 2
      Do
      Loop Until Pinb.2 = 1
   End If
   If Pinb.3 = 0 Then
      Count = Count / 2
      Do
      Loop Until Pinb.3 = 1
   End If
   If Pinb.4 = 0 Then
      Count = Count * 4
      Do
      Loop Until Pinb.4 = 1
   End If
  Waitms 20
Return
```

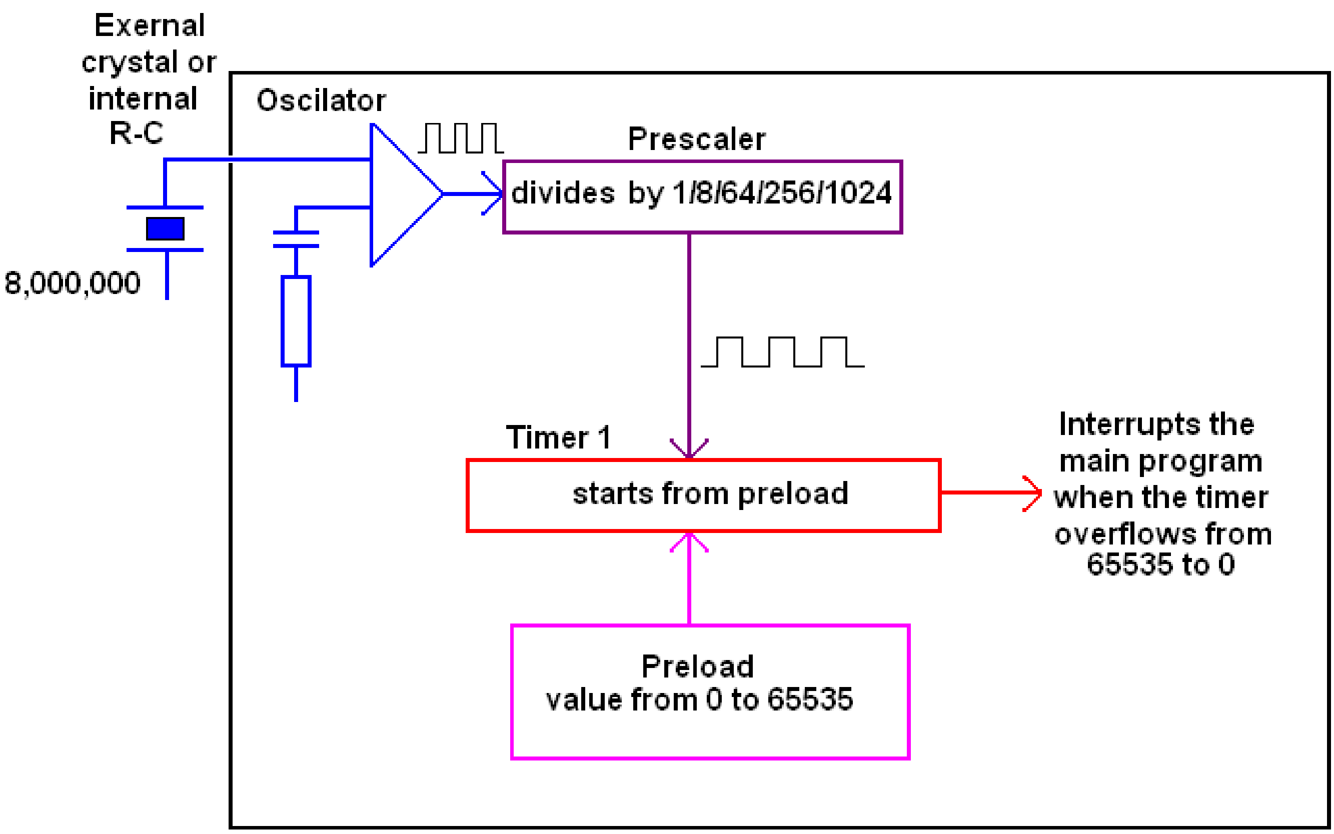
To overcome the fact we get an interrupt on switch press and another on switch release in this program there is a do-loop-until in each switch press that waits for the pin to be released before exiting the ISR. And to overcome switch bounce there is a short delay at the beginning and end.

48 Timer/Counters

The ATMega48/8535/16/32microcontroller shave a number of harware registers that have special functions. Three of these registers are Timer0, Timer1, and Timer2.

Timer0 is 8 bits so can count from 0 to 255
Timer1 is 16 bits so can count from 0 to 65535
Timer2 is 8 bits so can count from 0 to 255

Here is a block diagram of some of Timer1's features – it is possible to set very accurate output timings by varying the prescale and the preload values (of you use an external crystal oscillator rather then the internal RC, resistor capacitor, one)



The timer/counters can be written to and read from just like ordinary RAM but they also have so much more to offer a designer,

- Timers can count automatically; you just give the microcontroller the command to start, **enable timer1** and **enable interrupts** or to stop i.e. **disable timer1**.
- You don't even have to keep track of the count in your program; when a timer overflows it will call an interrupt subroutine for you via the command **on ovf1 tim1_isr** (on overflow of timer1 do the subroutine called tim1_isr), an overflow occurs when a variable goes from its maximum value (65535) back to 0.
- The rate of counting can be from the microcontrollers internal oscillator, i.e. **timer1 = timer,** or it can count pulses from an external pin i.e. **timer1 = counter** (which is pin B.1 for timer1).
- When counting from the internal oscillator it will count at the R-C/Crystal rate or at a slower rate. This can be the osciallator frequency, the oscillator/8 or /64 or /256 or /1024, in our program **prescale = 256** (which is 8,000,000/256 = 31,250 counts per second)
- The timer doesn't have to start counting from 0 it can be preloaded to start from any number less than 65535 i.e. **timer1 = 34286**, so that we can program accurate time periods.

There are over 60 pages in the datasheet describing all the neat things timers can do!

48.1 Timer2 (16 bit) Program

Timer1 is setup to give 1 second interrupts, every second the led will toggle.

```
'LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 20 * 4                         'configure lcd screen
'Timer 1
' preload = 65536 - 8000000 / ( prescale * intinterval) = 34286 (1sec interrupts)
' there are calculators on the web to help with this sort of thing
Config Timer1 = Counter , Prescale = 256
On Ovf1 Timer1_isr
Const Preload_value = 34286
Timer1 = Preload_value                       'reload timer1
Enable Timer1                               'enable timer 1 interrupt
Enable Interrupts                           'allow global interrupts to occur

Grn_led Alias Portb.5

Dim Count As Word
'--------------------------------------------------------------------
'progam starts here

Cls                                   'clears LCD display
Cursor Off                               'no cursor
Lcd "timer testing"
Do
  Locate 2 , 10
  Lcd Count
Loop
End                                   'end program

'--------------------------------------------------------------------
Timer1_isr:
  Timer1 = Preload_value                     'reload timer1
  Toggle Grn_led                    'if LED is off turn it on, if it is on turn it off
  Incr Count
Return
```

48.2 Timer0 (8bit) Program

This program toggles the led 100 times per second, too fast to see, but the count is usable. You could make a stop watch using this.

```
'LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 20 * 4                    'configure lcd screen
'Timer 1
' preload = 255 - 8000000 / ( prescale * intinterval) = 177 (1millisec interrupts)
' there are calculators on the web to help with this sort of thing
Config Timer0 = Counter , Prescale = 1024
On Ovf0 Timer0_isr
Const Preload_value = 177
Timer1 = Preload_value                   'reload timer1
Enable Timer0                          'enable timer 1 interrupt
Enable Interrupts                       'allow global interrupts to occur

' hardware aliases
Grn_led Alias Portb.5

Dim millsecs As byte

'------------------------------------------------------------------
'progam starts here

Cls                                'clears LCD display
Cursor Off                             'no cursor
Lcd "timer testing"
Do
  Locate 2 , 10
  Lcd millisecs
Loop
End                                'end program

'------------------------------------------------------------------
Timer0_isr:
  Timer0 = Preload_value                 'reload timer1
  Toggle Grn_led
  Incr millisecs
Return
```

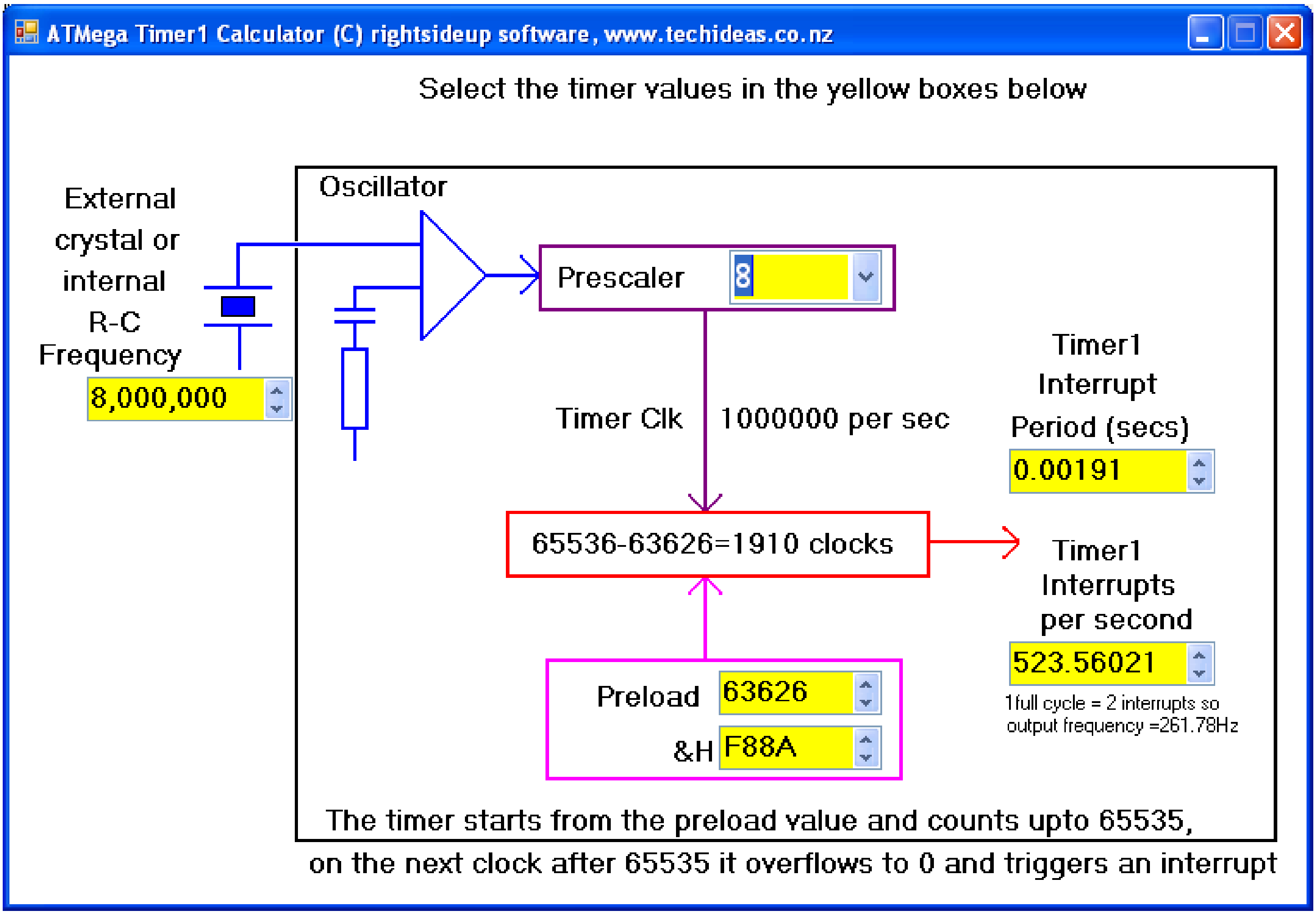
It is really important to undertstand that the timer will reoccur at the rate you set it at, in this program that is every 100mS. If the code in the timer routine takes more than 100mS to execute then you have too much code in it and your micro will crash.

In the program above the displaying of the value millisecs is not in the interrupt routine it is in the main code. The 3 lines of code in the interrupt routine can execute in less than 1 microsecond in total. The actual program code for commands like '**Locate** 2 , 10'and '**Lcd** millisecs' is long and very complex and they may take quite some time to execute, you would have to know a lot about assembly language to figure out exactly how long.

48.3 Accurate tones using a timer (Middle C)

```
' setup direction of all ports
Config Porta = Input                    '
Config Portb = Input
Config Portb.5 = Output                 '
Config Portb.6 = Output                 '
Config Portb.7 = Output                 '

'LCD
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 20 * 4                     'configure lcd screen
'Timer 1
Config Timer1 = Counter , Prescale = 8
On Ovf1 Timer1_isr

Enable Timer1                           'enable timer 1 interrupt
Enable Interrupts                        'allow global interrupts to occur

' hardware aliases
Red_sw Alias Pinb.0
Yel_sw Alias Pinb.1
Grn_sw Alias Pinb.2
Blu_sw Alias Pinb.3
Wht_sw Alias Pinb.4

Yel_led Alias Portb.6
Red_led Alias Portb.7
Piezo Alias Portb.5                     'could be a piezo or speaker+resitor or amplifier (LM386)


Dim Preload As Word
Preload = 63626                         '261.78Hz-middle C
Timer1 = Preload                        'reload timer1

Const Toneduration = 500

Do
   Enable Timer1  'restart the sound
  Waitms Toneduration
   Disable Timer1 ' stop the sound
   Reset Piezo   'make sure power to the output audio device is off
   Wait 5
Loop    ' keep going forever
End
'---------------------------------------------------------------
'. Subroutines
Timer1_isr:
   Timer1 = Preload            ' reload the counter (how long to wait for)
   Piezo = Not Piezo         ' toggle piezo     pin to make sound
Return
```

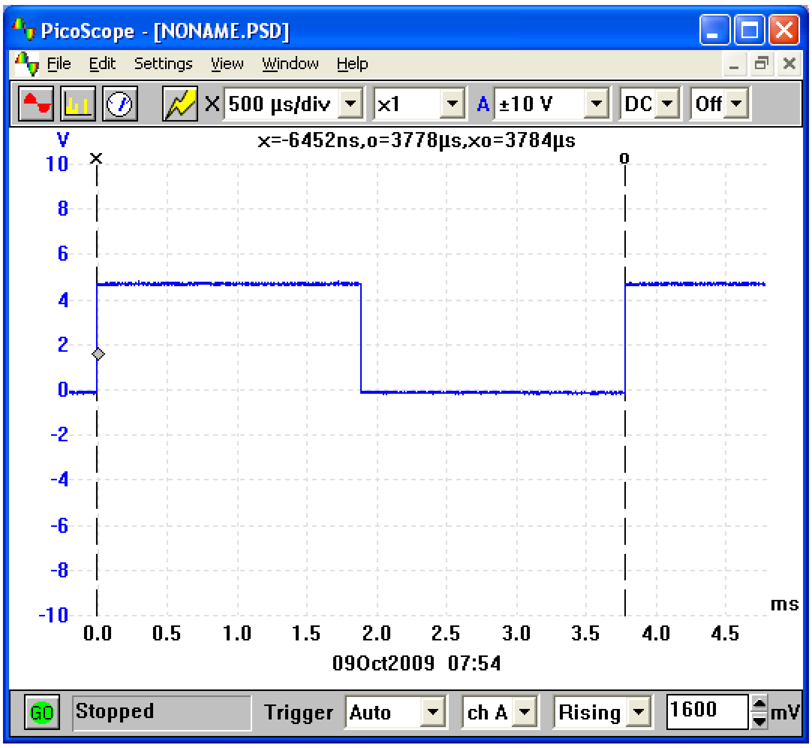
48.4 Timer1 Calculator Program



Using this program the calculations are easily done, simply enter a value into any of the yellow number boxes and the rest of the values will be calculated automatically.
The actual frequency wanted was Middle C (261.78Hz), this means we need 523.56 interrupts per second (2 interrupts per heterz of frequency)

Note that the microcontroller is working on an internal R-C (resistor-capacitor) oscillator and it is not very accurate.

In an experiment to get Middle C, the timer was preloaded with a value of 63626 and the following measurement was made on an oscilloscope.
The period actually is 3784uSecs (3.784mSecs), which is a frequency of 264.27Hz.

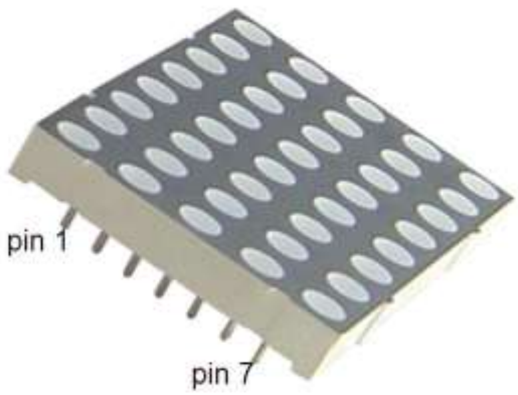


This error will vary from micro to micro and even as the temperature increases and decreases it can change. If you want more accuracy use an external crystal.
(There are versions of the above program for timer0 and timer2 as well).

48.5 Timer code to make a siren by varying the preload value

```
'-----------------------------------------------------------------
' Hardware Setups
Config Timer1 = Timer , Prescale = 1
On Ovf1 Timer1_isr   'at end of count do this subroutine
Enable Interrupts    'global interrupt enable
' Hardware Aliases
Spkr Alias Portb.2    'speaker is on this port
'-----------------------------------------------------------------
' Declare Constants
Const Countfrom = 55000  '
Const Countto = 64500
Const Countupstep = 100
Const Countdnstep = -100
Const Countdelay = 3
Const Delaybetween = 20
Const numbrSirens = 10
'-----------------------------------------------------------------
' Declare Variables
Dim Count As Word   'use useful names to help program understanding
Dim Sirencount As Byte
Dim Timer1_preload As Word
Timer1 = Timer1_preload
'-----------------------------------------------------------------
'  Program starts here
Do
    Gosub Makesiren
    Wait 5
Loop
End
'-----------------------------------------------------------------
' Subroutines
Makesiren:
   Enable Timer1                 'sound on
   For Sirencount = 1 To numbrSirens    'how many siren cycles to do
     For Count = Countfrom To Countto Step Countupstep  'rising pitch
        Timer1_preload = Count      'pitch value
        Waitms Countdelay      'length of each tone
     Next
     For Count = Countto To Countfrom Step Countdnstep  'falling pitch
        Timer1_preload = Count      'pitch value
        Waitms Countdelay      'length of each tone
     Next
     Waitms Delaybetween     'delay between each cycle
   Next
   Disable Timer1                'sound off
Return
' Interrupt service routines (isr)
Timer1_isr:
     Timer1 = Timer1_preload'if the timer isnt preloaded it will start from 0 after an interrupt
  Toggle Spkr
Return
```

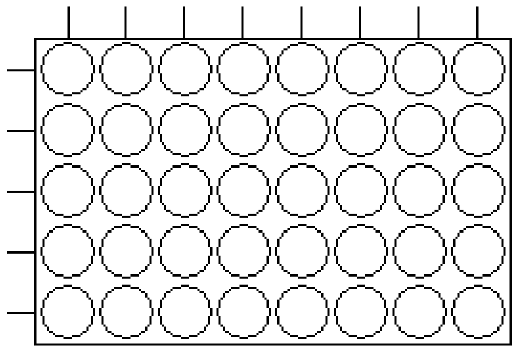
49 LED dot matrix scrolling display project – arrays and timers



The display is an excellent opportunity to learn more about arrays and timers

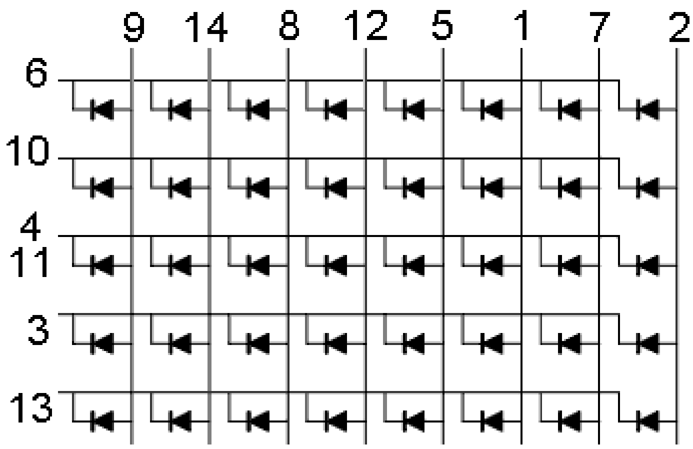
Before the display can be used though it must be understood. Sometimes it is enough to understand how to use a device without knowing everything about it (such as an LCD or LM35) however in this case the display is not really that complex and so must be thoroughly understood before it can be used. This means knowing what is indside it.

The LED dot matrix display is a grid of LEDs e.g. 35 LEDs arranged as 5x7,or 40 LEDs arranged as 5x8, or 64 LEDs arranged as 8x8.

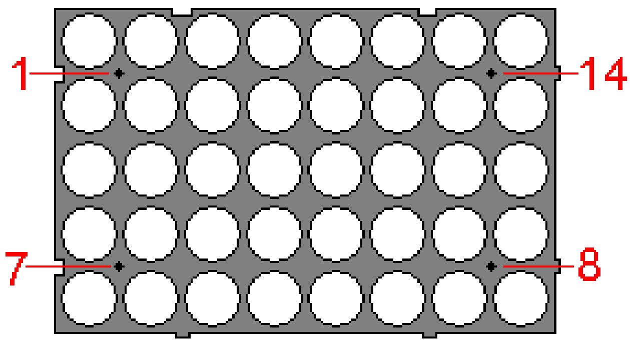


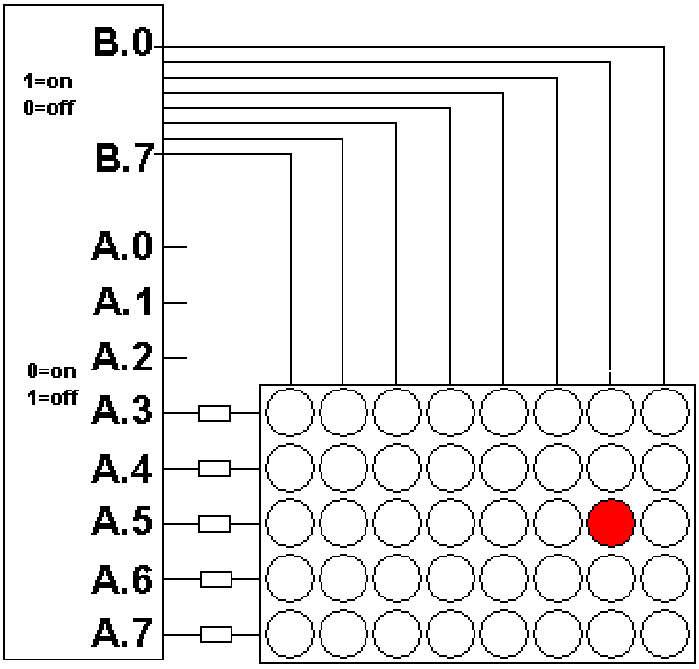
A dot matrix of 40 LEDs does not hace 80 pins (2 pins per LED) or even 41 pins (40 pins plus 1 common as with a 7 segment display) it needs onlt 13 pins (5+8) as the LEDs are arranged in a grid and share anodes and cathodes

Here is the actual schematic for the Sharlight CMD-3581300-W LED dotmatrix (13 is an odd number so they gave it 14 pins and joined 11 and 4 together)



The final step in understanding the device is the layout of the pins; these are numbered like an IC.
Make sure the display is the correct way around.
Check it with the pins and slots around the edges.



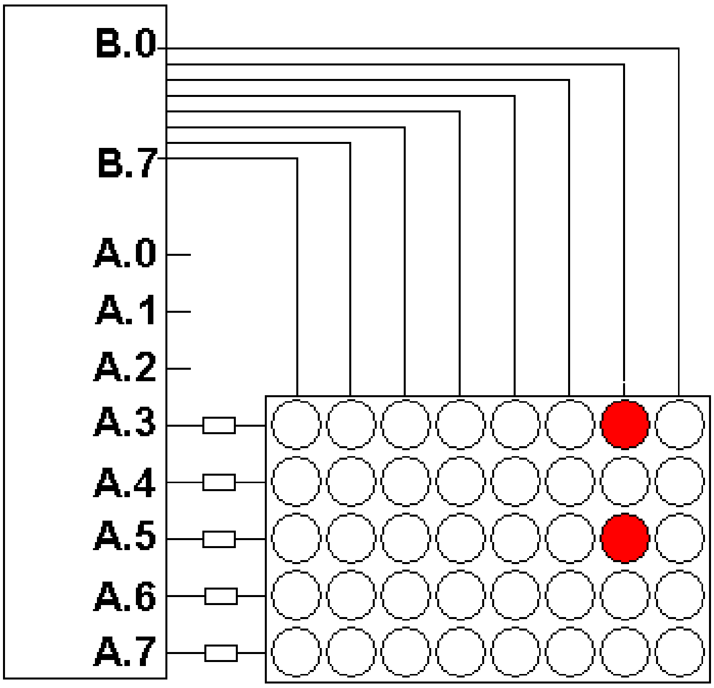


Here is an LED matrix connected to an ATMega8535. To get one particular LED to turn on the cathode needs to have a low (0V) applied and and the anode needs a high (5V). There are 5 resistors in series with the cathodes to reduce the 5V or too much current could flow and damage the LEDs, initially these could be set at 470R.

Both pins to the LEDs have to be the correct polarity for it to work/

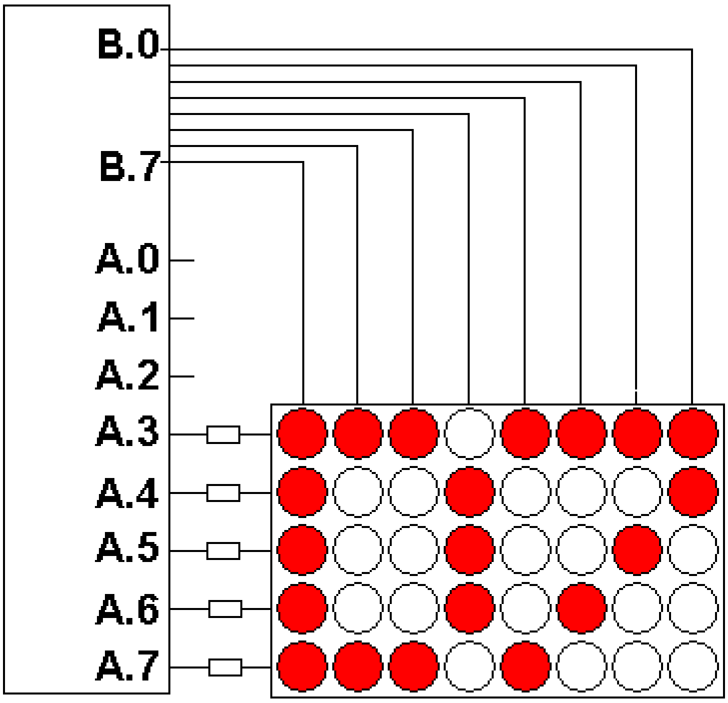
PORT A	PORTB
&B1101 1111 (only correct row low)	&B0000 0010 (only correct column high)

Any other combination will have different effects



To turn on both these LEDs the following sequence is required

PORT A	PORTB
&B 1101 0111 (2 rows low)	&B 0000 0010 (only correct column high)



To turn on a pattern all at once is not possible , the columns have to to be scanned one at a time. This is not difficult but requires some fast processing.

Note that from a hardware point of view this connection method does not really work the best. When a column has 1 or 2 LEDs going they are bright enough, however when there are 5 leds going they can be a bit dim, So 1 column might have 2 bright LEDs and the next 5 dim ones! This because a port on a micro can deliver about 20mA max- to 2 LEDS that means 10mA each, but to 5 LEDs it means 4mA each. Removing the resistors will help (as the leds are cycled rapidly they effectively don't get stressed. But the better solution is to use driver transitors on each column.

Here is a portion of a program to display the number of my classroom D7.

```
const w8=1
Do
  Porta = &B00000111              ' *****  (last 3 bits not used, so can be 0 or 1))
  Portb = &B10000000              'turn on column 1
  Waitms W8                         'small delay so it flashes quickly
  Porta = &B01110111              ' *   *
  Portb = &B01000000              'turn on column 2
  Waitms W8
  Porta = &B01110111              ' *   *
  Portb = &B00100000              'turn on column 3
  Waitms W8
  Porta = &B10001111              '  ***
  Portb = &B00010000              'turn on column 4
  Waitms W8
  Porta = &B01110111              ' *   *
  Portb = &B00001000              'turn on column 5
  Waitms W8
  Porta = &B10110111              '  *  *
  Portb = &B00000100              'turn on column 6
  Waitms W8
  Porta = &B11010111              '   * *
  Portb = &B00000010              'turn on column 7
  Waitms W8
  Porta = &B11100111              '    **
  Portb = &B00000001              'turn on column 8
  Waitms W8
Loop
```

49.1 Scrolling text code

A better solution is to use the built in timer of the microcontroller to do the scannig. The advantage of this is it de-compilcates your program code immensily by not having to worry about the timing for the scanning of the columns. In effect it is simple multitasking behaviour.

```
'-----------------------------------------------------------------
' 1. Title Block
' Author:     B. Collis
' Date:       12 Dec 07
' File Name:  dotmatrix_d7_timer_v1.bas
'-----------------------------------------------------------------
' 2. Program Description:
'     The text D7 is broken up into the following bytes
'       *** ****
'       *  *   *
'       *  *  *
'       *  * *
'       *** *
' as per the binary below (read it sideways)
'       00010000
'       01101110
'       01101101
'       01101011
'       00110111
' Hardware Features:
'    8 rows of dotmatrix LED connected to port B
'    5 cols of dotmatrix LED connected to port A
'Program Features
'    Flashes so fast that the message appears to be there all the time
'    works because of human persistence of vision
'-----------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the speed of the micro
$regfile = "m32def.dat"           'our micro, the ATMEGA32
'-----------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Output             'LEDs on portA
Config Portb = Output             'LEDs on portB
Config Portc = Output             'LEDs on portC
Config Portd = Output             'LEDs on portD
'timer
Config Timer1 = Counter , Prescale = 1
On Ovf1 Timer1_isr
Enable Timer1                     'enable timer 1 interrupt
Enable Interrupts                 'global flag allows interrupts to occur
' 5. Hardware Aliases
Row Alias Porta                   'digitdata on portA
Column Alias Portb                'B.0 to B.7
' 6. initialise ports so hardware starts correctly
Column = 2                        'second column on at first
' 7. Declare Constants
Const Preload_value = 56500
'-----------------------------------------------------------------
' 8 Declare Variables
Dim col_data(8) As Byte
```

```
Dim Col_count As Byte
' 9 Initialise Variables
Timer1 = Preload_value              'preload timer1
Col_count = 1
Col_data(1) = &B00000111             ' *****    (last 3 bits not used)
Col_data(2) = &B01110111             ' *   *
Col_data(3) = &B01110111             ' *   *
Col_data(4) = &B10001111             '  ***
Col_data(5) = &B01110111             ' *   *
Col_data(6) = &B10110111             '  *  *
Col_data(7) = &B11010111             '   * *
Col_data(8) = &B11100111             '    **
'-----------------------------------------------------------------
' 10. Program starts here
Do
  'nothing here yet
Loop
End
'-----------------------------------------------------------------
' 11. Subroutines
'subroutines
Timer1_isr:
  'puts the data in the array onto the rows, 1 column at a time
  'every time through turn on next column and get data for it
  Timer1 = Preload_value            'reload timer1
  Row = Col_data(col_count)         'put data onto row
  Rotate Column , Right             'turn on next column
  Incr Col_count                    'increase to next column
  If Col_count = 9 Then Col_count = 1  'only have 8 columns
Return
```

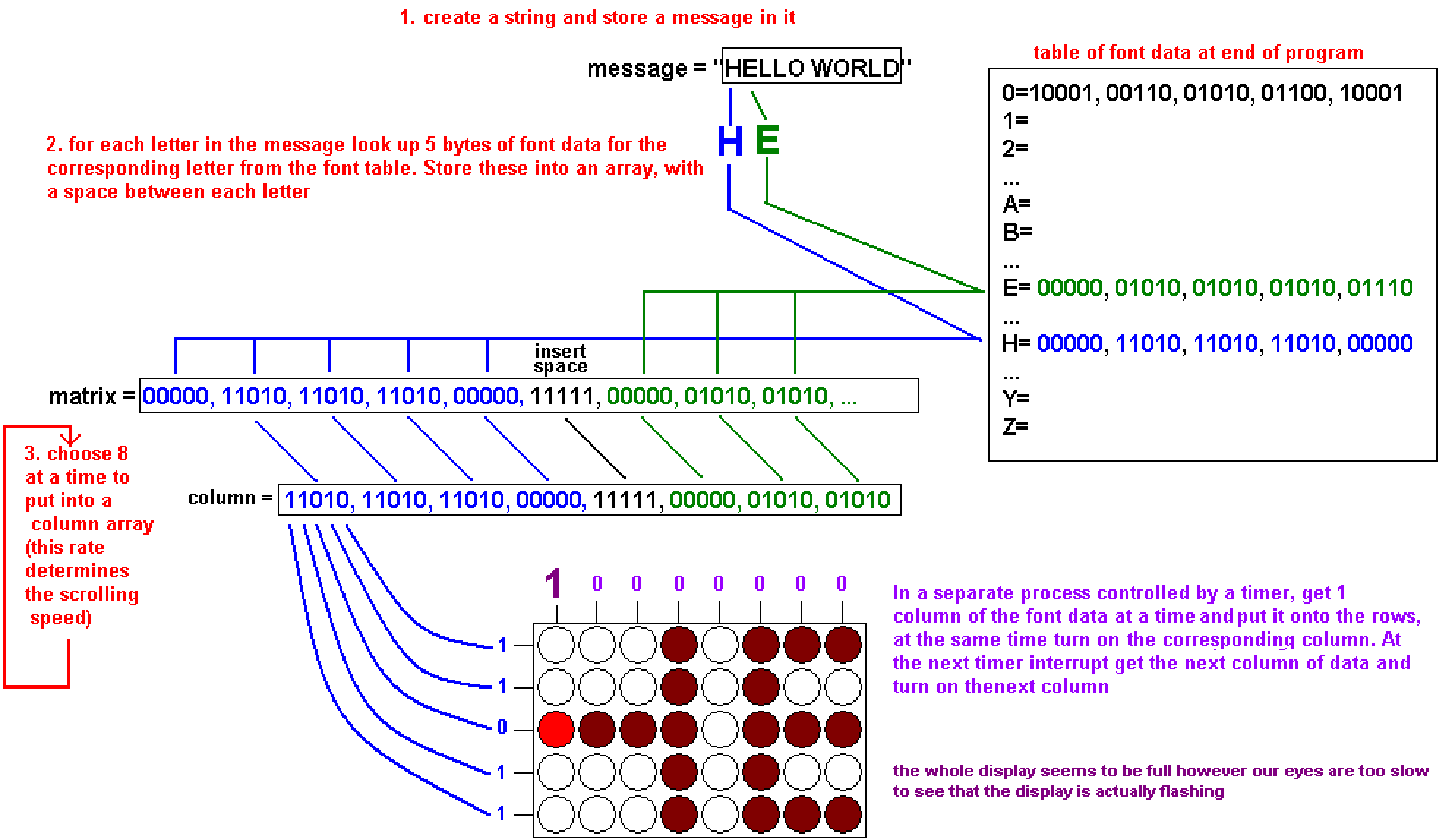
The next stage on the program is to have a scrolling message.

First algortihm:

1. The message is stored in a string
2. The string is converted to an array of data, 6 bytes per letter (1 for a space) – this is a large array
3. Get the first 8 pieces of data (1-8) and store them where the timer can access them
   - Wait a bit
   - Get the next 8 pieces of data (2-9)
   - And so on

The timers job is to scan the 8 columns with the data it is given
Note that there is no translation process for the ascii codes in the message string to LED dotmatrix data, this must be created manually .

49.2 Scrolling text – algorithm design



49.3 Scrolling test - code

```
'--------------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date:  June 08
' File Name: DotmatrixV3
'--------------------------------------------------------------------
' 2. Program Description:
' scrolls text across one 5x8 LED dot matrix
' uses timers, arrays and lookup tables
' 3. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000
$regfile = "m16def.dat"
$swstack = 40                  '
$hwstack = 32
$framesize = 32

'--------------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Output
Config Portc = Output
Config Portd = Output
Config Pind.3 = Input
'configure the timer for the LED scanning
Config Timer1 = Counter , Prescale = 1
On Ovf1 Timer1_isr
Const Preload_value = 56500
Timer1 = Preload_value             'reload timer1
Enable Timer1                    'enable timer 1 interrupt
Enable Interrupts                'allow global interrupts to occur

' 5. Hardware Aliases
Led Alias Portd.6
Col_diga Alias Porta               'A.0 to A.4
Col Alias Portc                  'c.0 to c.7
' 6. initialise ports so hardware starts correctly
Col = 2                        'second column on as first time around
                               ' rotate makes it 1
'--------------------------------------------------------------------
' 7. Declare Constants

'--------------------------------------------------------------------
' 8. Declare Variables
Dim Message As String * 50         'max 50 characters
Dim Matrix(308) As Byte            '6 times nmbr of chars + 8
Dim Count As Byte
Dim Singlechar As String * 1
Dim Char As Byte
Dim Mesg_char As Byte
Dim Temp As Byte
Dim M As Byte
```

```
Dim Col_count As Byte
Dim Matrix_ptr As Byte
Dim Table_ptr As Integer
Dim Speed As Byte
Dim Column(8) As Byte                ' the 8 cols on the display
Dim Messagelength As Word
Dim Matrixlength As Word
Message = "abcd"                    'USE @ FOR A SPACE
Messagelength = Len(message)
Matrixlength = Messagelength * 6
Matrixlength = Matrixlength + 8
Speed = 100

'-----------------------------------------------------------------
' 10. Program starts here
'fill array with 1's - all leds off
Count = 1
For Count = 1 To 8
  Column(count) = &B11111
Next

'get each character from the message
'and create a larger array of 5 bytes of font data for each character
Matrix_ptr = 1
For Count = 1 To 8
  Matrix(matrix_ptr) = &B11111        'insert 8 spaces at
  Incr Matrix_ptr                   ' beginning of message
Next
For Mesg_char = 1 To Messagelength
  'for each character in the message
  Singlechar = Mid(message , Mesg_char , 1)       ' get a char
  Table_ptr = Asc(singlechar)          ' get ascii value for character
  Table_ptr = Table_ptr - 48          ' not using ascii codes below "0-zero"
  Table_ptr = Table_ptr * 5           ' get pointer to font data in the table
  'copy 5 consecutive bytes from the table into the matrix array
    For Count = 0 To 4              'for 5 bytes of the font
    Temp = Lookup(table_ptr , Table)  'get the  font data
    Matrix(matrix_ptr) = Temp        'put it into the matrix table
    Incr Table_ptr
    Incr Matrix_ptr
  Next
  If Singlechar = ":" Then
    Matrix_ptr = Matrix_ptr - 4
    Matrixlength = Matrixlength - 4
  End If
  Matrix(matrix_ptr) = &B11111        'insert a space between
  Incr Matrix_ptr                   ' each character
Next
For Count = 1 To 8
  Matrix(matrix_ptr) = &B11111        'insert 8 spaces at
  Incr Matrix_ptr                   ' end of message
Next
```

```
'get 8 pieces of font at a time
Matrix_ptr = 1
Do
  'put the font into the display
  For Count = 1 To 10
    M = Matrix_ptr + Count
    Temp = Matrix(m)
    Column(count) = Temp
  Next
  Waitms Speed                    'scroll delay
  Matrix_ptr = Matrix_ptr + 1        'increase by 1 to scroll 1 column at a time
  If Matrix_ptr > Matrixlength Then Matrix_ptr = 0
Loop


End

'-----------------------------------------------------------------
' 11. Subroutines
'timer process
Timer1_isr:
  'puts the data in the column array out the port and onto the display
  '1 column at a time
  Timer1 = Preload_value              'reload timer1
  Col_diga = Column(col_count)        'put data onto column
  Rotate Col , Right                  'turn on next column
  Incr Col_count                      'increase to next column
  If Col_count = 9 Then Col_count = 1  'only have 8 columns
Return

End

'-----------------------------------------------------------------
' data fir font
Table:
' ***
'*  **
'* * *
'**  *
' ***
'Zero:
Data &B10001 , &B00110 , &B01010 , &B01100 , &B10001
'  *
' **
'  *
'  *
'  *
'one
Data &B11111 , &B11101 , &B00000 , &B11111 , &B11111
'two
Data &B01101 , &B00110 , &B01010 , &B01101 , &B11111
'Three:
Data &B10110 , &B01110 , &B01100 , &B10010 , &B11111
'Four:
```

```
Data &B10111 , &B10011 , &B10101 , &B00000 , &B10111
'Five:
Data &B01000 , &B01010 , &B01010 , &B10110 , &B11111
'Six:
Data &B10001 , &B01010 , &B01010 , &B10111 , &B11111
'Seven:
Data &B01110 , &B10110 , &B11010 , &B11100 , &B11110
'Eight:
Data &B10101 , &B01010 , &B01010 , &B10101 , &B11111
'Nine:
Data &B11001 , &B01010 , &B01010 , &B10001 , &B11111
'Colon:
Data &B10101 , &B11111 , &B10101 , &B11111 , &B11111
'Semicolon:
Data &B11111 , &B01111 , &B10101 , &B11111 , &B11111
'Lessthan:
Data &B11111 , &B11011 , &B10101 , &B01110 , &B11111
'Equals:
Data &B11111 , &B10011 , &B10011 , &B10011 , &B11111
'Greaterthan:
Data &B11111 , &B01110 , &B10101 , &B11011 , &B11111
'Question:
Data &B11101 , &B11110 , &B01010 , &B11101 , &B11111
'At:@ BUT ACTUALLY USE FOR SPACE
Data &B11111 , &B11111 , &B11111 , &B11111 , &B11111
' ***
'*   *
'*****
'*   *
'*   *
'A:
Data &B00001 , &B11010 , &B11010 , &B11010 , &B00001
'B:
Data &B00000 , &B01010 , &B01010 , &B01010 , &B10101
'C:
Data &B10001 , &B01110 , &B01110 , &B01110 , &B10101
'D:
Data &B00000 , &B01110 , &B01110 , &B01110 , &B10001
'E:
Data &B00000 , &B01010 , &B01010 , &B01010 , &B01110
'F:
Data &B00000 , &B11010 , &B11010 , &B11010 , &B11110
'G:
Data &B10001 , &B01110 , &B01110 , &B01010 , &B10011
'H:
Data &B00000 , &B11011 , &B11011 , &B11011 , &B00000
'I:
Data &B01110 , &B01110 , &B00000 , &B01110 , &B01110
'J:
Data &B10111 , &B01111 , &B01111 , &B01111 , &B10000
'K:
Data &B00000 , &B11011 , &B11011 , &B10101 , &B01110
'L:
Data &B00000 , &B01111 , &B01111 , &B01111 , &B01111
```

```
'M:
Data &B00000 , &B11101 , &B11011 , &B11101 , &B00000
'N:
Data &B00000 , &B11101 , &B11011 , &B10111 , &B00000
'O:
Data &B10001 , &B01110 , &B01110 , &B01110 , &B10001
'P:
Data &B00000 , &B11010 , &B11010 , &B11010 , &B111101
'Q:
Data &B10001 , &B01110 , &B01110 , &B00110 , &B00001
'R:
Data &B00000 , &B11010 , &B11010 , &B10010 , &B01101
'S:
Data &B01101 , &B01010 , &B01010 , &B01010 , &B10110
'T:
Data &B11110 , &B11110 , &B00000 , &B11110 , &B11110
'U:
Data &B10000 , &B01111 , &B01111 , &B01111 , &B10000
'V:
Data &B11000 , &B10111 , &B01111 , &B10111 , &B11000
'W:
Data &B00000 , &B10111 , &B11011 , &B10111 , &B00000
'X:
Data &B01110 , &B10101 , &B11011 , &B10101 , &B01110
'Y:
Data &B11110 , &B11101 , &B00011 , &B11101 , &B11110
'Z:
Data &B01110 , &B00110 , &B01010 , &B01100 , &B01110
'[:
Data &B11111 , &B00000 , &B01110 , &B01110 , &B11111
'\:
Data &B11110 , &B11101 , &B11011 , &B10111 , &B01111
'[:
Data &B11111 , &B01110 , &B01110 , &B00000 , &B11111
'^:
Data &B11111 , &B11101 , &B11110 , &B11101 , &B11111
'_:
Data &B01111 , &B01111 , &B01111 , &B01111 , &B01111
'\:
Data &B11110 , &B11101 , &B11111 , &B11111 , &B11111
' **
'   *
' ***
'*  *
' ***
'a:
Data &B10111 , &B01010 , &B01010 , &B00001 , &B11111
'b:
Data &B00000 , &B01011 , &B01011 , &B10111 , &B11111
'c:
Data &B10011 , &B01101 , &B01101 , &B11111 , &B11111
'd:
Data &B10111 , &B01011 , &B01011 , &B00000 , &B11111
' you can do the rest!!!
```

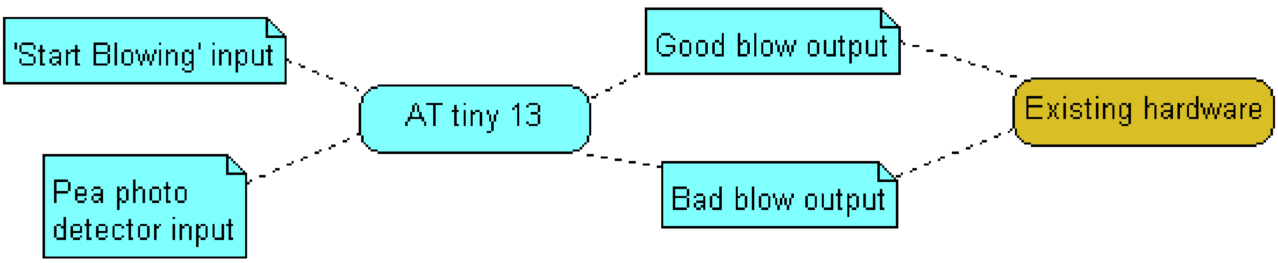
50 Medical machine project – timer implementation

Situation:
The client had built a machine that measured certain aspects of air in a persons lungs. It required the person to blow a minimum volume of air through a straw into the machine.

The product was highly satisfactory however it had a limitation in that if the person did not blow long or hard enough then deep air from the lungs might not come out. In that case the device might give a false reading. The client was an expert in analogue electronics and mechanical design but needed some assistance with solving this issue as they did not know enough about programming or microcontrollers.

Alex and Victor two year12 students designed this product in 2006.

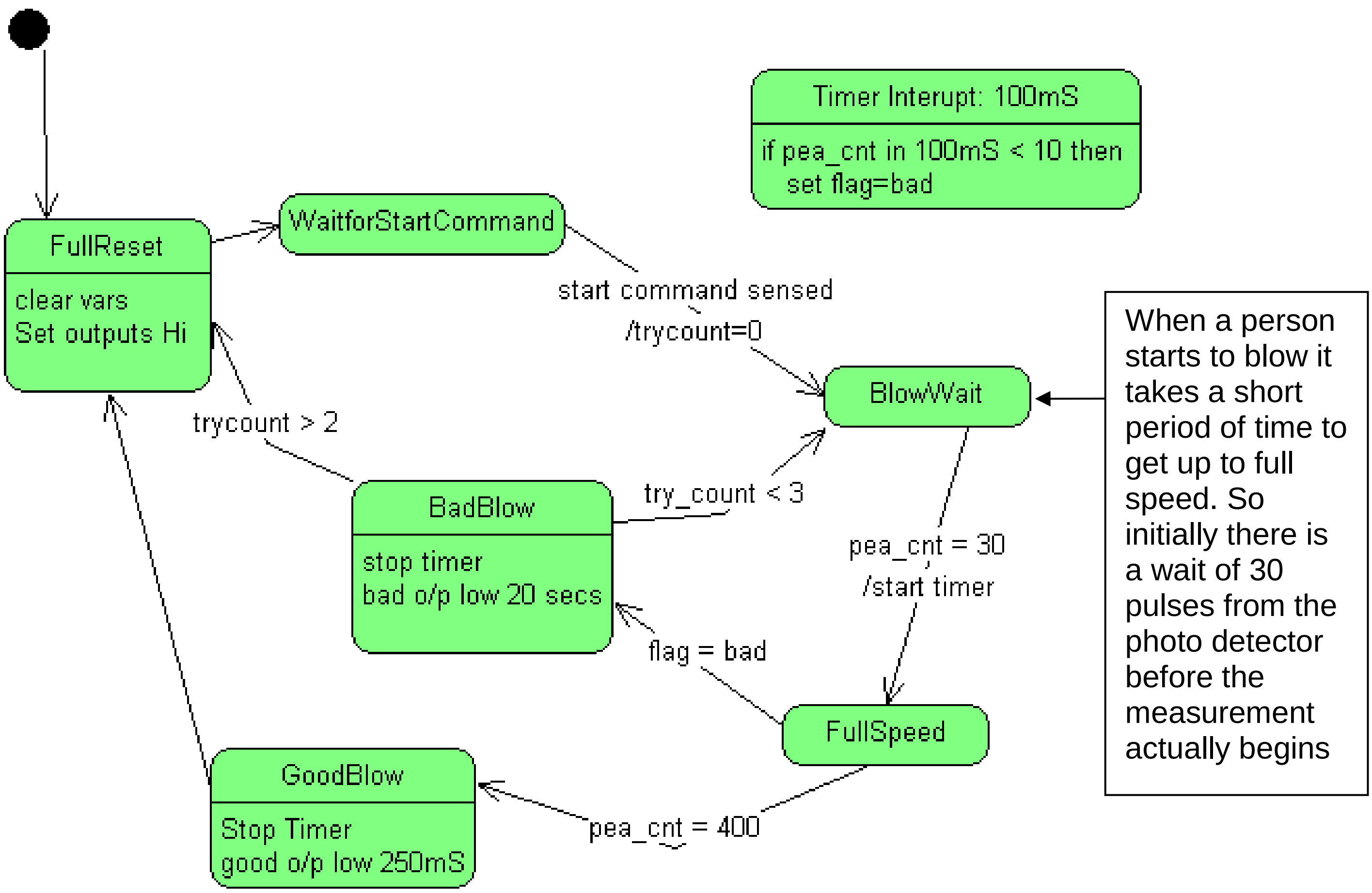
50.1 Block diagram



A small chamber with a pea sized ball in it (imagine a whistle) was inserted into the airline to measure the air flow. There is an infrared led on one side and a photodetector on the other to measure the air speed. As the user blows the ball rotatesin the chamber breaking the infrared path between the LED and photodetector.
A second input to the circuit is a start blowing command, 2 outputs were required: good blow and bad blow to interface to the existing circuitry.

50.2 Blower - state machine

This state machine was designed with the student to count the revolutions of the pea once the start command was sensed.



It is important to get enough air for an accurate reading so the user must blow **both** hard and long enough. The timer is used to count the number of pea rotations. Every 100mS the count must increase by at least 10 or the user is deemed not to be blowing hard enough. If the blow lasts for 400 pea counts then it is a good blow. This would mean at least 4 seconds of blow.

The client also wanted 3 tries so that if the user gave a short blow they could try again.

The client also wanted field adjustments so that when programming in the field they could alter things for different situations.

As the tacher I was a significant stakeholder in the project as well and I wanted significant input to the project as I knew that in the future if the client wanted anything changed I was the one who would get the call! I therefore made sure that the documentation was of a high standard.

50.3 Blower program code

```
'-----------------------------------------------------------------
' 1. Title Block
' Author: Alex  & Victor
' Date: 12 Sep 2006
' File Name: peactr_v3.bas
'-----------------------------------------------------------------
' 2. Program Description:
'    v3 changed pull up to pull down
'    v2 implemented fail retries
' 3. Hardware features
'    2 photo diodes, one senses startcommand, other senses rotating pea
'    2 outputs, one for a pass, one for a fail
' 4. Software Features:
'-----------------------------------------------------------------
' 5. Compiler Directives
$crystal = 1200000                    ' internal clock
$regfile = "attiny13.dat"             ' ATTINY13V
$hwstack = 20
$swstack = 8
$framesize = 16
'-----------------------------------------------------------------
' 6. Hardware Setups
Config Portb = Output
Config Pinb.3 = Input
Config Pinb.4 = Input

Config Timer0 = Timer , Prescale = 1024
On Ovf0 Tim0_isr
Enable Interrupts
Dim Preload_value As Byte
Preload_value = 138

'-----------------------------------------------------------------
'USER FIELD ADJUSTMENTS
Const Pass_time = 250                 'milliseconds
Const Fail_time = 20                  'seconds
Dim Tries As Byte
Tries = 3
Const P_limit = 10                    '10=5 revs reqd every 100mS for a pass
'if too high then the person cannot blow hard enough to register
'if too low then they can blow too softly and give inaccurate readings
Const P_trigger = 30                  'doesnot count the first 30 pulses (15 revs)
'allows the person to get blow to full speed
Const P_target = 400                  '400=200 revs , the length of the blow

'-----------------------------------------------------------------
'USER FIELD ADJUSTMENTS END
'-----------------------------------------------------------------
```

```
'-----------------------------------------------------------------
'flag values
Const Counting = 0
Const Good = 1
Const Bad = 2
'states
Const State_waitforstartcommand = 1
Const State_blowwait = 2
Const State_fullspeed = 3
Const State_badblow = 4
Const State_goodblow = 5
Const State_fullreset = 6

'alias
Bad_output Alias Portb.0
Good_output Alias Portb.1
Startblowing_input Alias Pinb.3
P_sensor Alias Pinb.4

Dim New_pcnt As Word
Dim Old_pcnt As Word
Dim Diff_pcnt As Byte
Dim Flag As Byte                'timer interrupt
Dim Pstate As Bit
Dim Try_count As Byte
Dim State As Byte

Tries = Tries - 1              'need to reduce for count to work

State = State_fullreset
Do
     Gosub Pcounting
     Select Case State
          Case State_waitforstartcommand : Gosub Startwait
          Case State_blowwait : Gosub Blowwait
          Case State_fullspeed : Gosub Fullspeed
          Case State_badblow : Gosub Badblow
          Case State_goodblow : Gosub Goodblow
          Case State_fullreset : Gosub Fullreset
          Case Else : Gosub Fullreset     'just in case
     End Select
Loop
End
```

This an alternative form of state chart control to that previoslt described. With this code there are are no actions that take place between states.

```
'-------------------------------------------------------------
Fullreset:
            Try_count = 0
            Gosub Resetvar
            State = State_waitforstartcommand
Return

Resetvar:
            Good_output = 1
            Bad_output = 1
            New_pcnt = 0
            Old_pcnt = 0
            Flag = 0
Return

Startwait:
            If Startblowing_input = 0 Then
               State = State_blowwait
               New_pcnt = 0
               Old_pcnt = 0
            End If
Return

Blowwait:
            If New_pcnt = P_trigger Then
                        State = State_fullspeed
                        Enable Timer0               'start timing
            End If
Return

'Count the pulses when a change occurs
Pcounting:
            If P_sensor = Pstate Then          'check if sensor has changed
                      Incr New_pcnt                'increase the count
                      Pstate = Not Pstate            'change to other input value
            End If
Return

'Stay In This State until either bad enough or good enough
Fullspeed:
            If Flag = Bad Then State = State_badblow
            If New_pcnt = P_target Then State = State_goodblow
Return
```

```
Badblow:
          Disable Timer0                    'stop timing
          Bad_output = 0                    'signal to rest of machine
          Wait Fail_time
          Bad_output = 1
          Incr Try_count
          If Try_count > Tries Then
                   State = 6                        'reset machine
          Else
                   State = State_blowwait           'have another go
                   Gosub Resetvar                   'reset counters etc
          End If
Return

Goodblow:
          Disable Timer0                    'stop timing
          Good_output = 0                   'signal to rest of machine
          Waitms Pass_time
          Good_output = 1
          State = State_fullreset           'reset machine
Return

'-----------------------------------------------------------------------
'timer
'Every 100ms
Tim0_isr:
        Timer0 = Preload_value
        Diff_pcnt = New_pcnt - Old_pcnt     'find out how many counts
        If Diff_pcnt < P_limit Then         'if not enough
                 Flag = Bad
        Else                                'or if enough
                 Old_pcnt = New_pcnt                'remember current count
        End If
Return
```

51 Multiple 7-segment clock project – dual timer action

Some surplus 7-segment display boards were found on trademe and it was decided that my classroom needed a fancy new clock.
The display digits are 70mm high x 48 mm wide and the whole board is 360mm in length

Not just any clock is required though; one of the problems in the classroom is that school periods can be a little short for students and once they get going with practical work it is hard for them to stop when the bell goes – well actually the truth is its my fault, I loose track of the time. So I needed a special clock one that not only displayed the time but that kept track of how long there was left in a period and could warn both the students and me that the period was rapidly coming to an end.

51.1 Understanding the complexities of the situation

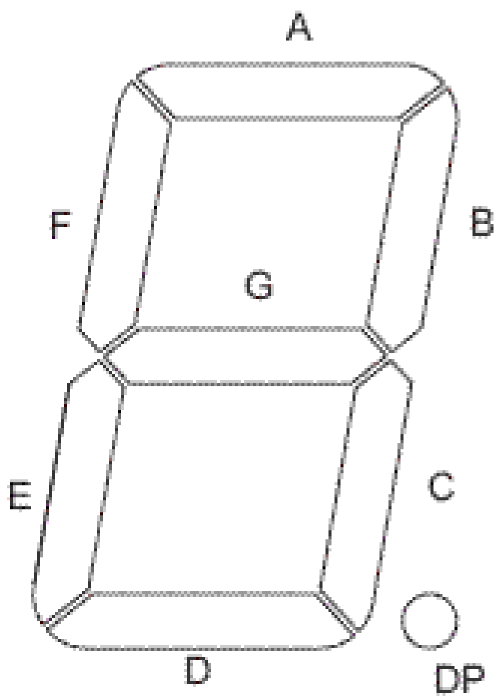
The situation is much more complicated than initially might be thought because the school timetable is actually a device of torture used by those in the know to torment humble teachers and students alike.

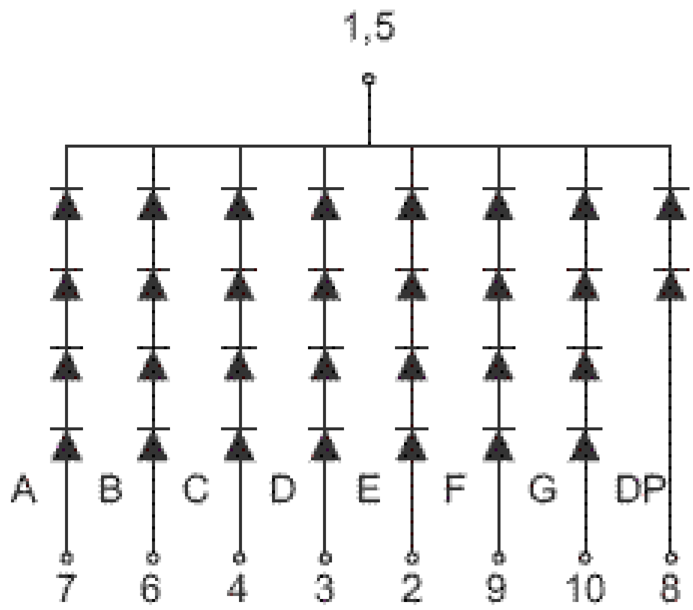
- Mondays and Tuesdays have the same bell times.
- Wednesday has its own because of a late start that day.
- Thursday and Friday have the same bell times but these are a different to Monday, Tuesday and Wednesday (got it so far?)
- We actually only teach 5 periods in a day but on a Tuesday and Friday there are 6 periods($%$%#)
- This rotates every week so we teach periods 1 to 5 one week periods 2 to 6 the next and 3 to 1 etc etc
- In the first version of this project it was made worse by the fact that we use to have assemblies on Friday which changed with the rotation as to who went and who didn't so the times changed for some Friday periods some weeks and not others. This has changed however but I keep a copy of that version safely stored which means that next year should those in command change again I can reimplement that trickery into the code
- I concluded early on that I needed to manage each day of the week individually!
- There is an emergency power stop in my room, so the clock must be battery backed up.
- The school periods should only be displayed during school weeks, of which there are about 36 each year. Weekends and school holidays only the time should display.
- Because the bell times are so different showing just the the time itself is meaningless, the clock needs to show how many minutes are left until the next bell.
- There should be an extra message that happens 5 minutes before the bell goes to remind people to cleanup
- My classroom is shared by another teacher once per day, the clock should mean something to that person too!

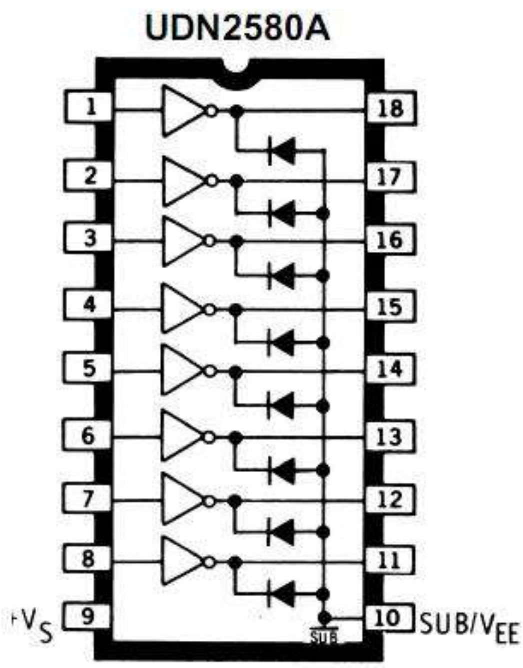
A messge will rotate around the 7 digits that will look like:
" **10-37 3-1 P1-Yr10 4T0G0 CLEANUP** "
10-37 is the time, then **3-1** the rotation (if a Tuesday or Friday), I always get asked this by students so it was good to see it. **P1-Yr10** who is in the class at the moment, this is really redundant information because both the students and I know who is there but it is important in that it clarifies to all who see the clock that it is correct in its operation. **4T0G0** how many minutes are left till the bell and finally the **CLEANUP** message if it is less than 6 minutes to go in the period.

51.2 Hardware understanding:

There are 7 seven segment diplays on the PCB with a nice connector, each segment has 4 LEDs and the decimal point has 2 LEDs. This makes them very bright and suitable for the classroom.







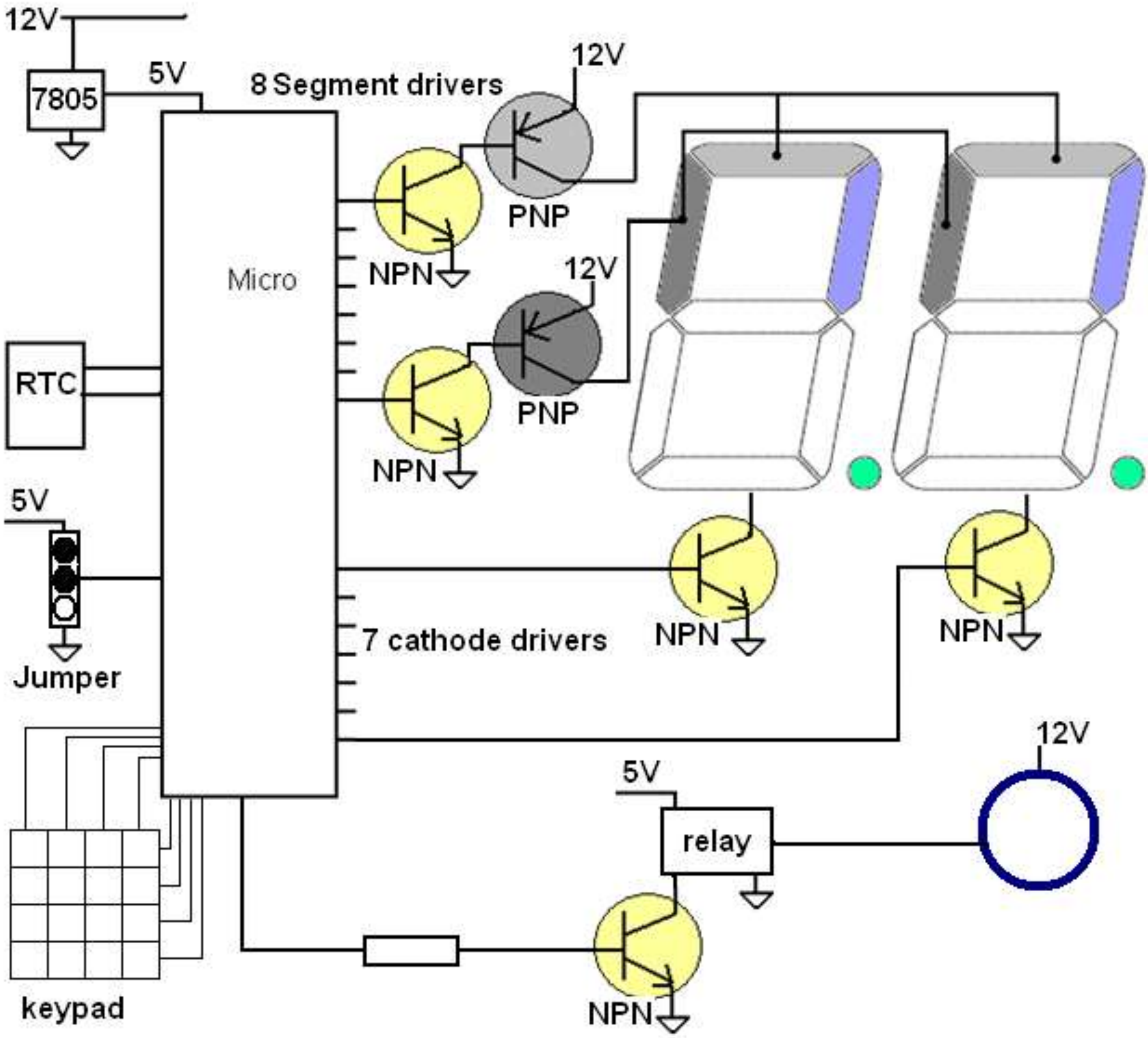
The problem with the dot matrix introductory scrolling text project was the issue of brightness of the LEDs, this was resolved by developing a circuit with driver transistors. Amongst my component stock pile I had some driver ICs both NPN (ULN2803) and PNP (UDN2580).Both have 8 transistors each, are Darlington types so are high gain and good for switching medium power.

51.3 Classroom clock – block diagram

This is the final system block diagram for the classroom clock it shows the connections for the seven 7-segment displays to the microcontroller (j ust 2 digits and 2 segments are shown in the diagram to reduce complexity).

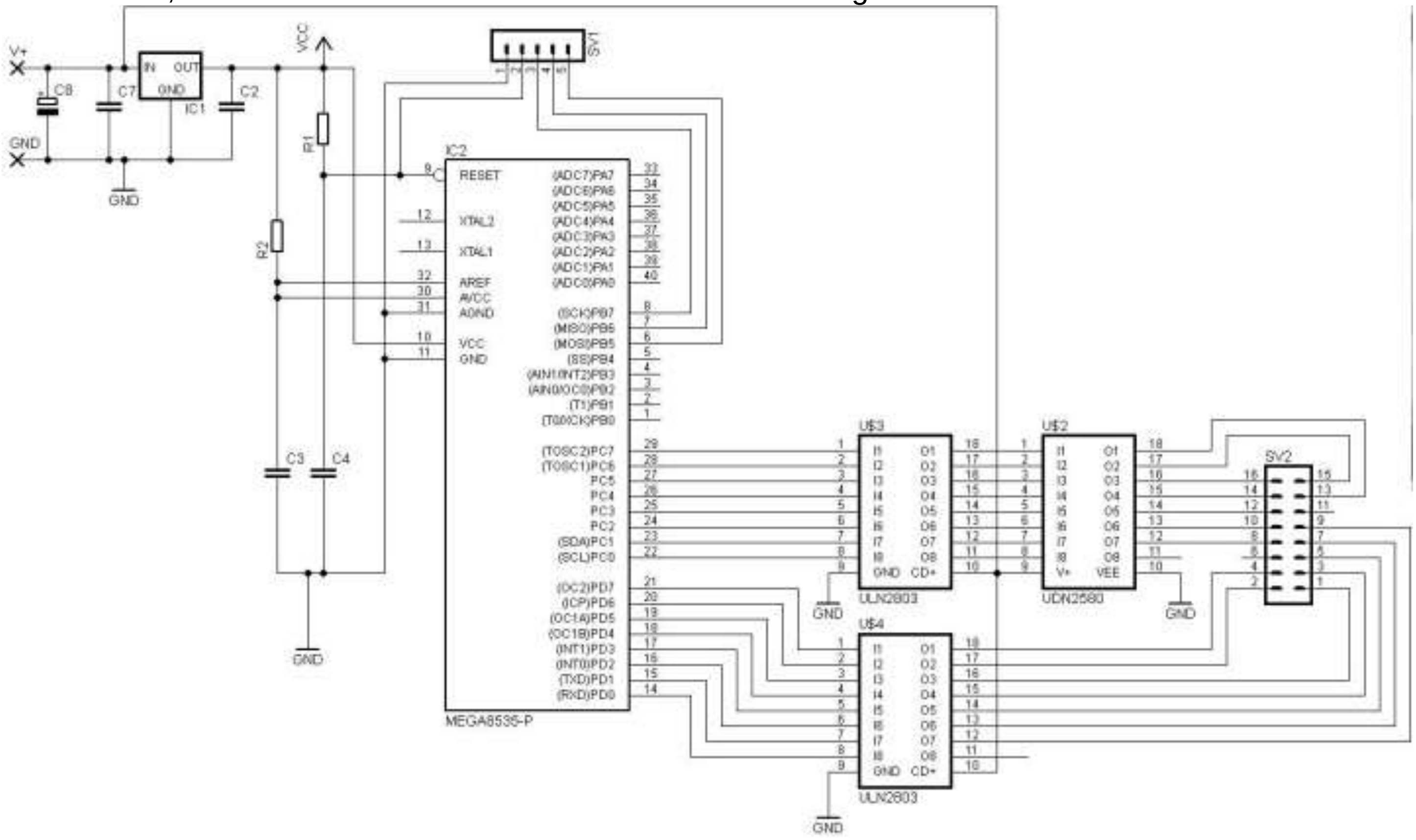
As well as the 7seg displays the other interfaces that were added as the project developed have been included:

- RTC (real time clock)
- Jumper (to select normal/settings modes)
- Blue flashing light (with transistor and relay to drive it)
- Keypad



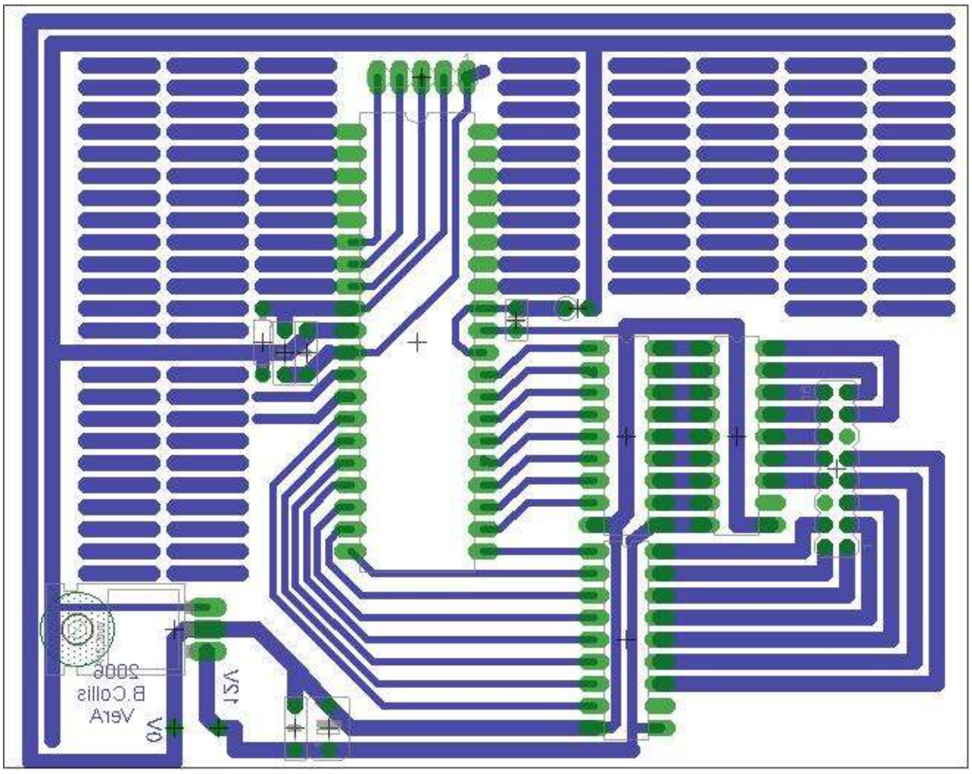
51.4 Classroom clock - schematic

When the schematic was initially developed it was not known exactly what interfaces would be needed for the clock, so a board that could be added to later was designed.



51.5 Classroom clock - PCB layout

This layout shows the extra breadboarding area available for other circuits (such as the RTC etc) which can be added later.

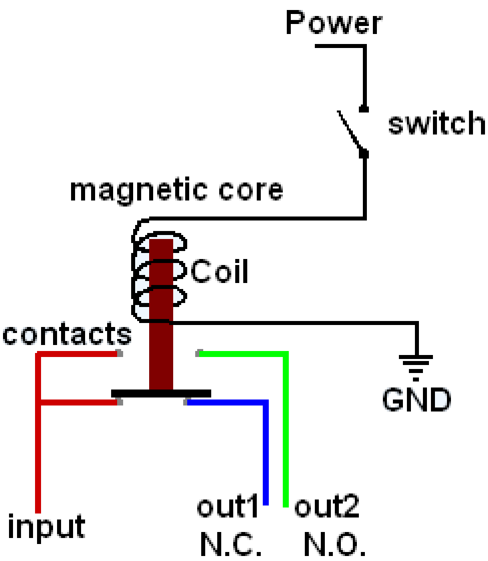


51.6 Relay Circuit Example

A flashing light was needed for the clock to act as a warning that the end of the period was approaching.
A Jaycar blue mini strobe was purchased. It uses a xenon tube, is real bright, runs off 12V, draws 180mA and flashes at a rate of 90 per minute.

As the light requires 180mA to work it cannot be run straight from a microcontroller port pin as they can only provide 20mA. So some amplifier device was needed.

The light could be run from a transistor or fet, however if I wanted to change it for some other light in the future then I might have to change the transistor as well. So I decided to make the device as general purpose as possible and add a relay circuit that would provide more flexibility. A relay is also an isolation device, the input and output circuits are not electrically connected, so a high voltage power supplyor the light cannot get backinto the Microcontroller.

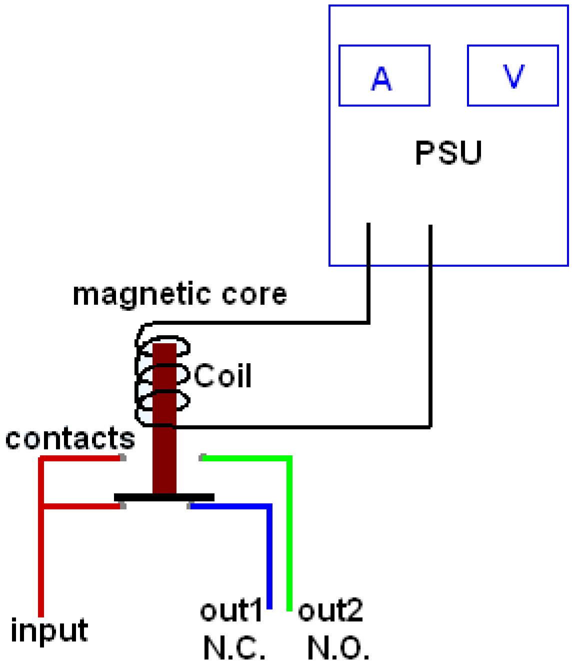


Relays come in all shapes and sizes and current and voltage ratings, they are however fairly standard in theory. There are two types today electro-mechanical and solid-state, this theory is about the electro mechanical type.

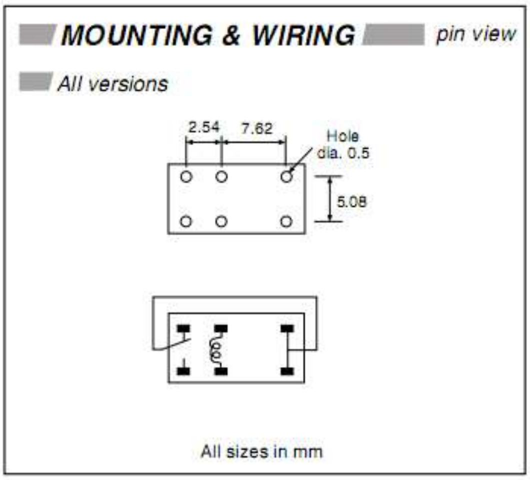
A relay consists of 2 parts a coil and a set of contacts. Through the centre of the coil is a metal bar that moves when power is applied to the coil. Attached to the metal bar are switch contacts that change connections when the bar moves. In the diagram when power is applied to the coil, the input will change from being connected to out1 to out2. These contacts are sometimes known as NC- normally closed and normally open.

For this project an OKO K51A05 was on hand so I found out the connection details for it. If you don't have a data sheet then use a multimeter to help you. Measure the resistance between all the different pins on the relay, the coil will have a fixed resistance such as 1000 ohms or less. The NC contacts will be 0 and the NO contact will have no connection to any other contacts. Once you have identified the coils apply voltage to the coil, start with a low voltage 5V, if you hear it click then you have the right voltage, if you don't increase it. Some relays work off 5V some off 12V some off 24V, and others all in between.

It's a good idea to know the current that it draws as well so a bench PSU is useful.



The K51A05 part number on the device led to the datsheet on the internet, The connection details are in the datsheet.



The part number or what ever else is written on the relay may give clues as to the ratings of the switching contacts . In this case the datasheet gives all the details

SPECIFICATION

Contact Form	1C
Contact Material	AgPd (Au clad)
Contact Rating (resistive load)	1A @ 24Vd.c. 0.5A @ 125Va.c.
Min. Switching	1mA @ 5V
Contact Resistance (initial)	≤100mΩ (measured @ 1A, 6Vd.c.)
Coil Consumption D.C.	150mW
Max. Coil Voltage at 70°C	160% nominal
Operate Time	5ms typ.
Release Time	5ms typ.
Insulation Resistance	≥1000MΩ @ 500Vd.c.
Dielectric Strength (coil-contact)	1000Va.c. 50/60Hz 1min.
Dielectric Strength (contact-contact)	400Va.c. 50/60Hz 1min.
Vibration Resistance	10 ~ 55Hz, 3.3mm D.A.
Shock Resistance (malfunction)	approx. 10G
Mechanical Life	5 x 10^6 ops. min. (36000 ops./hour)
Electrical Life	1 x 10^5 ops. min. (18000 ops./hour)
Weight	2.2g
Ambient Temperature	-30°C to +70°C
Approvals*	

(*applied for)

COIL DATA

Nominal Coil Voltage (V)	Pick-Up Volts (max.) (V)	Drop-Out Volts (min.) (V)	Coil Resistance (Ω)	Nominal Current (mA)
		D.C. COILS		
1.5	1.2	0.15	15 ±10%	100.0
3	2.4	0.3	60 ±10%	50.0
5	4.0	0.5	167 ±10%	29.9
6	4.8	0.6	240 ±10%	25.0
9	7.2	0.9	540 ±10%	16.7
12	9.6	1.2	960 ±10%	12.5
24	19.2	2.4	3840 ±15%	6.25

The interesting specifications are:

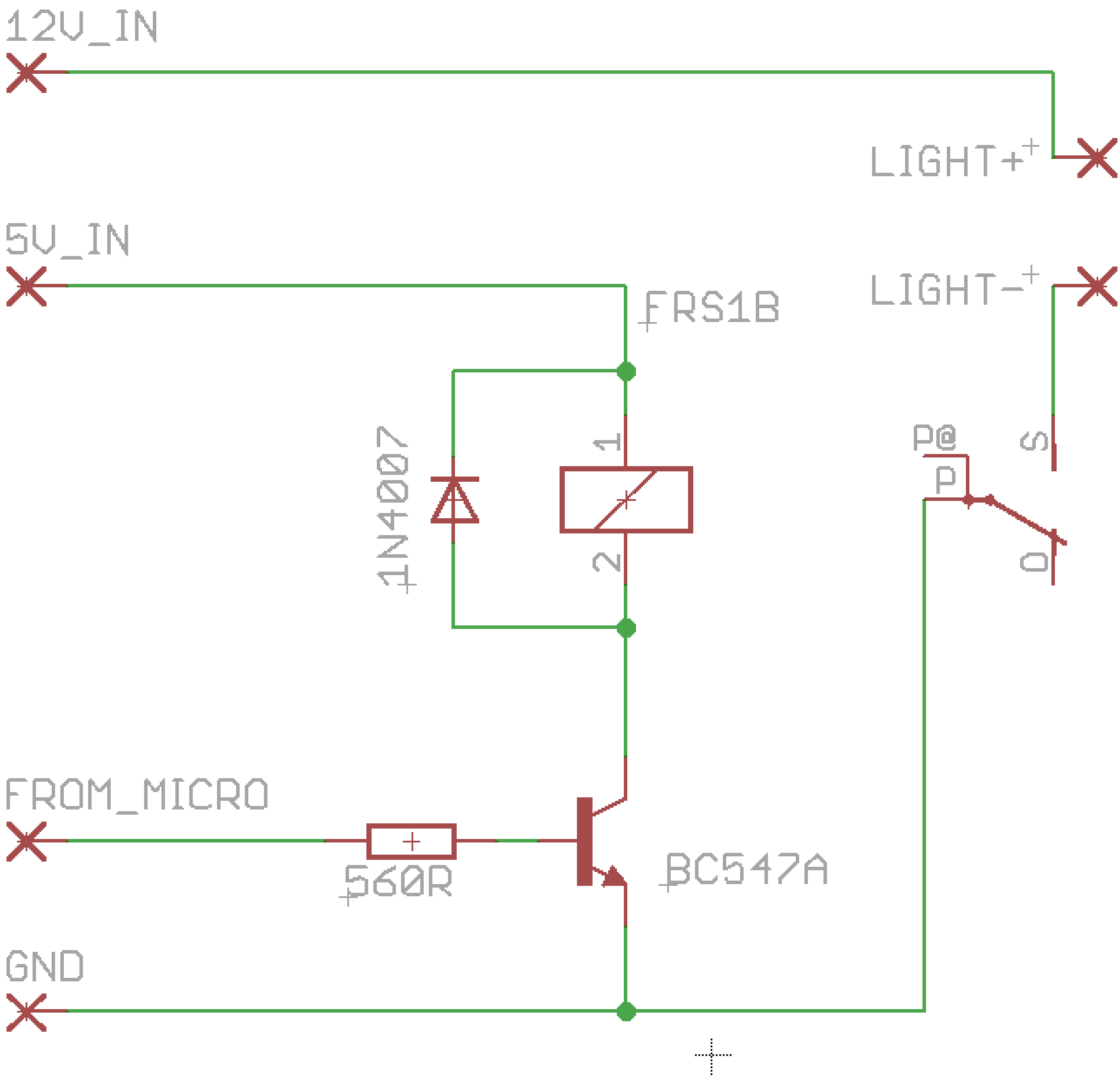
- Contact ratings: 1A at 24VDC (we are switch a 12V strobe light that requires 180mA)
- Coil draws 150mW power, so at 5V that's 0.03mA (P=V*I, so I = P/V)
- It needs at least 4V to pickup or close the contacts
- The contacts will stay closed (drop out) unitl the voltage goes below 0.5V.
- The current is 29.9ma (confirms our power calculation above)

In this case it seems that we cannot drive our relay from the microcontroller directly as it needs 30mA and a micro pin can only give 20mA, so we need to a firststage of amplification.
A transistor such as the BC547 could be useful.

SYMBOL	PARAMETER	CONDITIONS	MIN.	MAX.	UNIT
V_{CBO}	collector-base voltage	open emitter			
	BC546		–	80	V
	BC547		–	50	V
V_{CEO}	collector-emitter voltage	open base			
	BC546		–	65	V
	BC547		–	45	V
V_{EBO}	emitter-base voltage	open collector			
	BC546		–	6	V
	BC547		–	6	V
I_C	collector current (DC)		–	100	mA
I_{CM}	peak collector current		–	200	mA
I_{BM}	peak base current		–	200	mA
P_{tot}	total power dissipation	$T_{amb} \leq 25$ °C; note 1	–	500	mW

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
h_{FE}	DC current gain	$I_C = 10$ µA; $V_{CE} = 5$ V; see Figs 2, 3 and 4				
	BC546A		–	90	–	
	BC546B; BC547B		–	150	–	
	BC547C		–	270	–	
	DC current gain	$I_C = 2$ mA; $V_{CE} = 5$ V; see Figs 2, 3 and 4				
	BC546A		110	180	220	
	BC546B; BC547B		200	290	450	
	BC547C		420	520	800	
	BC547		110	–	800	
	BC546		110	–	450	

- We are switching 12BV the BC547 can switch 45VDC so that is fine
- We need 30mA, the BC547 can switch 100mA so it will be ok.
- A transistor when it is fully on still has 0.3V across it, so that means 4.7V avaialble for the relay (the relay requires 4V minimum so that is ok)
- The BC547 can dissipate (get rid of) no more than 500mW of power, we are drawing 30mA and the voltage across the BC547 is 0.3V so P=V*I = 0.3 * 0.03 = 0.009W = 9mW, so that is ok too.
- The BC547B we have has a gain (hFE) for at least 200, that is the ratio of output current to input current. We want 30mA out so input current = output current /gain = 0.03/200 = 0.00015A = 0.15mA from the microcontroller. Our micro can supply 20mA so that is no problem, we just need a resistor to limit the current from the micro to the transistor, a 560R was chosen as it was at hand, but we could calculate it. %v from the micro and 0.00015A , R = V/I = 33K.

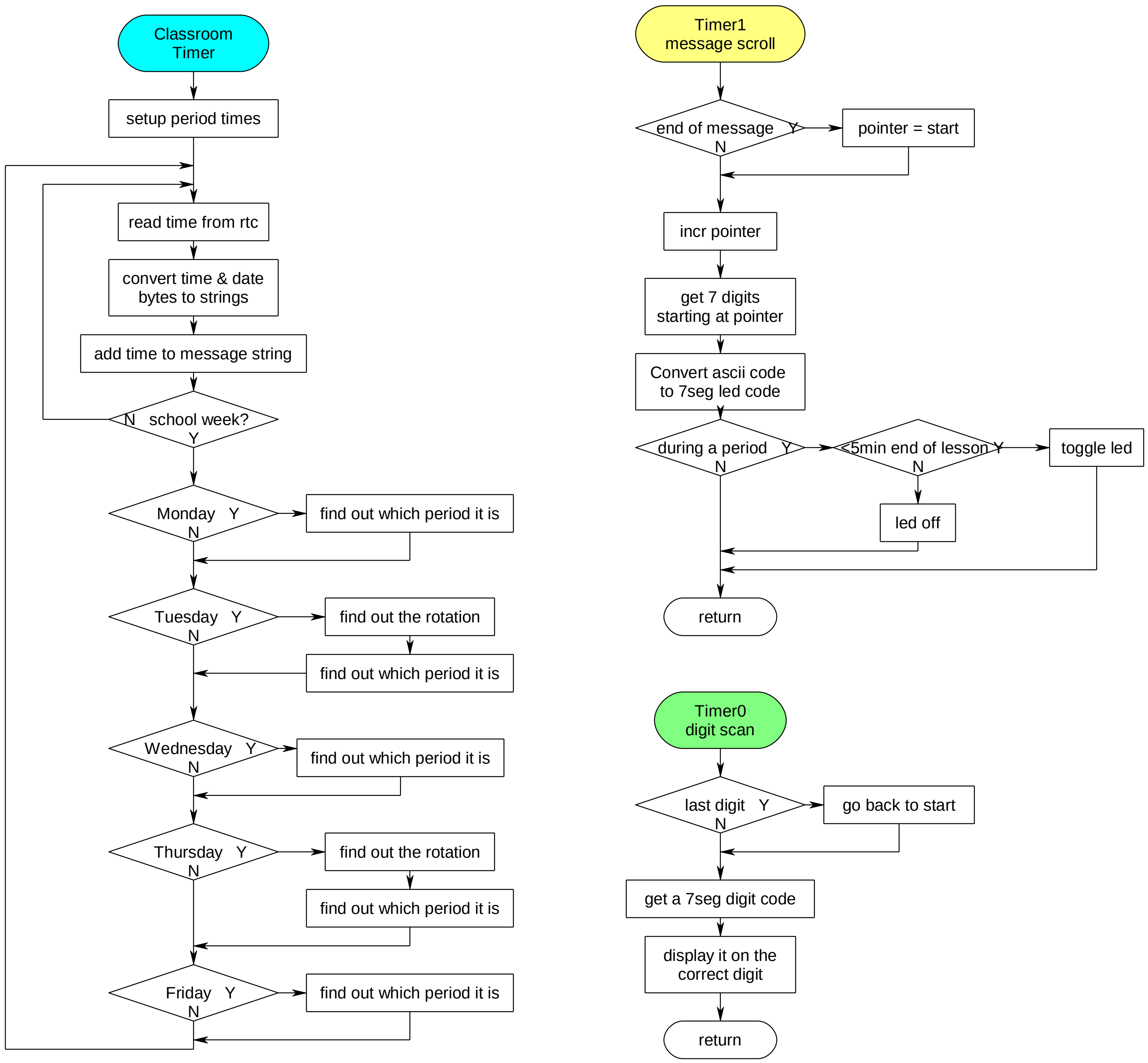


This is the circuit developed. Now there is a very important component, that has not been discussed so far, the diode across the relay coil. This diode is **VERY IMPORTANT**. I will explain why.

A coil of wire is known as an inductor and inductors have a very interesting electrical property, they don't like changes in current flow (just as a capacitor doesn't like change in voltage across it, an inductor doesn't like change of current through it). This is due to the magnetic field that is associated with current and wires.

So when the relay is powered up and we switch off the transistor, the magnetic field that is around the coil slowly collapses back into the coil (its called back EMF), this however can have devastating effects as the field causes electrons to flow in the coil which can have very high potential. In fact they could have hundreds of volts potential, enough to kill our little 45V BC547B and 5V microcontroller very very quickly. To protect the BC547 and the microcontroller we put a reverse polarised diode across the coil. This shorts out that back EMF and protects our circuit.

51.7 Classroom clock – flowcharts



The settigs mode is entered by moving the jumper on pinb.3; when in this mode the display is used to display various times/dates and set them using a keypad on portA.

' 0 - nothing pressed
' 1 - display time
' 2 - display date
' 3 - minute of day + secs
' 4 - weekday
' 5 - day of year
' 6 - week of year
' 7 - rotation
' 8 - increase day
' 9 - decreas day
' A - increase minutes
' B - decrease minutes
' C - increase hours
' D - decrease hours
' * - zero seconds
' # - increase month

51.8 Classroom clock – program

```
'------------------------------------------------------------------
' 1. Title Block
' Author:     B.Collis
' Date:       JUL 2009
' File Name:  ClassClock7SegVer4a.bas
'------------------------------------------------------------------
' 2. Program Description:
' routines to drive large seven segment display boards
'
' the display digits are on portd and the segments on portc
' the digits are interfaced via octal darlington drivers
' only 1 digit can actually be turned on at one time so the digits
' are scanned rapidly and the eye cannot detect the flashing
' the segments are in the order c d e b a f g 0
' so the letter b would turn on c,d,e,f,g its binary is &B1110011X
'
' two timers used, one for digit scanning, the other for scrolling the message
' because we want the period to be displayed there are lots of calcs
'
'v4 - changed to 2009 timetable
'v4A - changed to 2010 timetable  ,ADDED TIME SETTING FEATURES
'TIME SETTING FEATURES - PUT JUMPER INTO OTHER POSITION
' 0 - nothing pressed
' 1 - display time
' 2 - display date
' 3 - minute of day + secs
' 4 - weekday
' 5 - day of year
' 6 - week of year
' 7 - rotation
' 8 - increase day
' 9 - decreas day
' A - increase minutes
' B - decrease minutes
' C - increase hours
' D - decrease hours
' * - zero seconds
' # - increase month


'look for ************** in the code
'these are the things that will have to be rewritten each year
'------------------------------------------------------------------
' 3. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000                'the crystal we are using
$regfile = "m32def.dat"           ' the micro we are using
$hwstack = 126
$swstack = 40
$framesize = 120
'------------------------------------------------------------------
' 4. Hardware Setups
' setup direction of all ports
Config Porta = Input              'keypad
Config Portb = Output             'RTC, LED, JUMPER
Config Portc = Output             'segments
Config Portd = Output             'digits

'scan timer for 7seg digits
Config Timer0 = Timer , Prescale = 1024
Enable Timer0
Enable Interrupts
On Ovf0 Timer0_digitscan

'message scrolling timer
'timer1 is 16 Bit
Config Timer1 = Timer , Prescale = 1024
Enable Timer1
Enable Interrupts
On Ovf1 Timer1_messagescroll
```

```
'keypad on porta
Config Kbd = Porta

' config 2 wire I2C interface
'Config I2cdelay = 5                      ' default slow mode
Config Sda = Portb.1
Config Scl = Portb.0
Config Clock = User                      'dimensions time&date variables

Config Portb.2 = Output                  'LED
Config Pinb.3 = Input                    'jumper
Portb.3 = 1                              'turn on internal pullup
' 7. Hardware Aliases
Segmentbus Alias Portc
Digitbus Alias Portd
Led Alias Portb.2
Jumper Alias Pinb.3
Bluelight Alias Portb.4
' 8. initialise ports so hardware starts correctly
Porta = &B11111111                       '
'Portb = &B11111111                       ' kills 1307
Portc = 0                                'turns off segments
Portd = 0                                'turns off digits
'------------------------------------------------------------------
' 9. Declare Constants
Const Scrolltime = 64500                 'timer1 value to control scrolling speed
Const Scantimer = 235                    'timer0 value to control scanning of digits
Const True = 1
Const False = 0

Const Mondayrc = 520                     ' 8:40
Const Mondayp1 = 530                     '8:50
Const Mondayp2 = 590                     '9:50
Const Mondayint = 645                    '10:45
Const Mondayp3 = 670                     '11:10
Const Mondayp4 = 725                     '12:05
Const Mondaylunch = 785                  '13:05
Const Mondayssr = 825                    '13:45
Const Mondayp5 = 855                     '14:10
Const Mondayend = 910                    '15:10

Const Tuesdayrc = 520                    '8:40
Const Tuesdayp1 = 530                    '8:50
Const Tuesdayp2 = 590                    '9:50
Const Tuesdayint = 645                   '10:45
Const Tuesdayp3 = 670                    '11:10
Const Tuesdayp4 = 725                    '12:05
Const Tuesdaylunch = 785                 '13:05
Const Tuesdayssr = 825                   '13:45
Const Tuesdayp5 = 855                    '14:10
Const Tuesdayend = 910                   '15:10

Const Wednesdaypd = 500                  '8:20
Const Wednesdayp1 = 560                  '9:20
Const Wednesdayp2 = 615                  '10:15
Const Wednesdayint = 665                 '11:05
Const Wednesdayp3 = 685                  '11:25
Const Wednesdayp4 = 735                  '12:15
Const Wednesdaylunch = 785               '13:05
Const Wednesdayssr = 830                 '13:50
Const Wednesdayp5 = 855                  '14:10
Const Wednesdayend = 910                 '15:10

Const Thursdayrc = 520                   '8:40
Const Thursdayp1 = 530                   '8:50
Const Thursdayp2 = 590                   '9:50
Const Thursdayint = 650                  '10:50
Const Thursdayp3 = 675                   '11:15
Const Thursdayp4 = 730                   '12:10
Const Thursdaylunch = 785                '13:05
Const Thursdayssr = 830                  '13:50
Const Thursdayp5 = 855                   '14:10
Const Thursdayend = 910                  '15:10
```

```
Const Fridayrc = 520                    '8:40
Const Fridayp1 = 530                    '8:50
Const Fridayp2 = 590                    '9:50
Const Fridayint = 650                   '10:50
Const Fridayp3 = 675                    '11:15
Const Fridayp4 = 730                    '12:10
Const Fridaylunch = 785                 '13:05
Const Fridayssr = 830                   '13:50
Const Fridayp5 = 855                    '14:10
Const Fridayend = 910                   '15:10
'-------------------------------------------------------------------
'**************
Const Dayoffset = 4                     '**************
'**************
Const Keydelay = 300
'-------------------------------------------------------------------

' 10. Declare Variables
Dim Key As Byte
Key = 0
Dim K As Byte
Dim Digit(7) As Byte                    'data for each digit
Dim Dig As Byte                         'which digit is on
Dim Msgstr As String * 80               'the full text to be scrolled
Dim Msgptr As Byte                      'points to digits
Dim Msglen As Byte
Dim Dispstr As String * 7               'string on display
Dim Ascii As String * 1                 'single character in a string
Dim Timestr As String * 8               'my time string
Dim Secstr As String * 4                '-seconds
Dim Datestr As String * 7               'my date string
Dim Rotationstr As String * 8           'my date string
Dim Periodstr As String * 25            'school period name
Dim Period As Byte                      'the rotated period
Dim Asci As Byte
Dim I As Byte
Dim Temp As Byte
Dim Minuteofday As Word                 'stores minutes since midnight
Dim Minutesleft As Word                 'minutes to go this period
Dim Days As Word                        'days of year 1 to 365/6
Dim Weekday As Byte                     'day of week mon=1
Dim Weekofyear As Word                  'needsto be word!!
Dim Rotation As Byte
Dim Periodflag As Bit                   'true will mean it is a teaching period
Dim Ramlocation As Byte
Dim Ramvalue As Byte

' 11. Initialise some values for time/date
_year = 10
_month = 1
_day = 17
_hour = 11
_min = 01
_sec = 0
Dispstr = ""
Msgstr = ""
Msgptr = 1
Minutesleft = 10
Periodflag = False

Timer1 = Scrolltime                     'start timer correctly
```

```
'-----------------------------------------------------------------
' 12. Program starts here
Do
   'clock and period display mode
   Gosub Read1307time                'get the current time and date
   Gosub Converttime                 'put time into a string 14-12-46
   'need week of year to see if a school week and not holidays
   'first need to know the day of the year to calculate week of the year
   Days = Dayofyear(_day)            '1jan = 0
   Incr Days                         'so add one, 1jan = 1
   'however to get our weeks correct we need to adjust for the fact
   'that the first day of the year is not on a monday.
   'this is important otherwise rotations can be ok on a tue but not fri!!
   'the first week of the year is the week that has the first thursday
   Days = Days + Dayoffset
   Weekofyear = Days / 7             'must use word size
   Minuteofday = _hour * 60          'work out minutes since start of day
   Minuteofday = Minuteofday + _min
   Weekday = Dayofweek(_day)         'mon = 0
   Incr Weekday                      'add one so monday = 1
   Rotation = Lookup(weekofyear , Weekrotation)
   I = Rotation + 4                  'make a string to display rotation
   If I > 6 Then I = I - 6
   Rotationstr = Str(rotation) + "x" + Str(i)

   If Jumper = 1 Then                   'normal mode
      If Minuteofday > 910 Then         'dont disp rotation after sch
         Rotationstr = ""
      End If
      Msgstr = "        "               'leading spaces
      Msgstr = Msgstr + Timestr + "        "
      Periodstr = ""
      'week of year starts with first full week i.e. 4Jan10 = week1
      Select Case Weekofyear            'if a school week get current period
         Case 4 To 14 : Gosub Getperiodstring       'term1 **************
         Case 17 To 26 : Gosub Getperiodstring       'term2 **************
         Case 28 To 37 : Gosub Getperiodstring       'term3 **************
         Case 40 To 49 : Gosub Getperiodstring       ' term4 **************
      End Select
      If Periodflag = True Then Msgstr = Msgstr + Periodstr + "       "
      If Periodflag = True And Minutesleft < 6 Then
         Msgstr = Msgstr + "cleanup       "
         Led = 1
      Else
         Led = 0
      End If
      If Periodflag = True And Minutesleft < 6 And _sec < 15 Then
         Set Bluelight
      Else
          reset bluelight
      End If
      Msglen = Len(msgstr)
   End If                               ' keep looping forever
```

```
'time/date display/set mode
   If Jumper = 0 Then
      Led = 1                          'led on
      Gosub Convertdate
      Gosub Readkeypad
      Select Case Key
         Case 0 : Msgstr = "press"      'initial value
         Case 1 : Msgstr = Timestr + Secstr
         Case 2 : Msgstr = Datestr
         Case 3 : Msgstr = Str(minuteofday) + "+" + Secstr
         Case 4 : Msgstr = Str(weekday) + " of7"
         Case 5 : Msgstr = "d+" + Str(days)        'day of year
         Case 6 : Msgstr = Str(weekofyear) + " of52"
         Case 7 : Msgstr = Rotationstr
         Case 8 : Gosub Incrday
         Case 9 : Gosub Decrday
         Case 11 : Gosub Incrmonth      '#
         Case 10 : Gosub Zerosecs       '*
         Case 12 : Gosub Incrmin        'A
         Case 13 : Gosub Decrmin        'B
         Case 14 : Gosub Incrhour       'C
         Case 15 : Gosub Decrhour       'D
      End Select
      Msglen = 7                       'only ever display 7 characters
   End If
Loop
End                                    'end program
'-------------------------------------------------------------------
'-------------------------------------------------------------------
' 13. Subroutines

'read the keypad and convert to a recognisable digit
Readkeypad:
      K = Getkbd()
      Waitms 100
      Select Case K
         Case 0 : Key = 15              'D
         Case 1 : Key = 14              'C
         Case 2 : Key = 13              'B
         Case 3 : Key = 12              'A
         Case 4 : Key = 11              '#
         Case 5 : Key = 9
         Case 6 : Key = 6
         Case 7 : Key = 3
         Case 8 : Key = 0
         Case 9 : Key = 8
         Case 10 : Key = 5
         Case 11 : Key = 2
         Case 12 : Key = 10             '*
         Case 13 : Key = 7
         Case 14 : Key = 4
         Case 15 : Key = 1
         'Case 16 : Key = 16          'do not use this, rem last key press
      End Select
Return
'this routine zeros the seconds and writes the new time to the RTC
Zerosecs:
   _sec = 0
   Gosub Write1307time                 'use only to set time
   Waitms Keydelay
   Key = 1                             'display time
Return
'this routine increases the minute by one and writes the new time to the RTC
Incrmin:
   Incr _min
   If _min > 59 Then _min = 0
   Gosub Write1307time                 'use only to set time
   Waitms Keydelay
   Key = 1                             'display time
Return
```

```
'this routine decreases the minute by one and writes the new time to the RTC
Decrmin:
   Decr _min
   If _min > 59 Then _min = 59
   Gosub Write1307time                   'use only to set time
   Waitms Keydelay
   Key = 1                               'display time
Return
'this routine increasea the hours by one and writes the new time to the RTC
Incrhour:
   Incr _hour
   If _hour > 23 Then _hour = 0
   Gosub Write1307time                   'use only to set time
   Waitms Keydelay
   Key = 1                               'display time
Return
'this routine decreases the hours by one and writes the new time to the RTC
Decrhour:
   Decr _hour
   If _hour > 23 Then _hour = 23
   Gosub Write1307time                   'use only to set time
   Waitms Keydelay
   Key = 1                               'display time
Return
'this routine increasea the day by one and writes the new time to the RTC
Incrday:
   Incr _day
   If _day > 31 Then _day = 1            'no checking for month of year!!!!!
   Gosub Write1307time                   'use only to set time
   Waitms Keydelay
   Key = 2                               'display DATE
Return
'this routine decreases the hours by one and writes the new time to the RTC
Decrday:
   Decr _day
   If _day = 0 Then _day = 31
   Gosub Write1307time                   'use only to set time
   Waitms Keydelay
   Key = 2                               'display DATE
Return
'this routine increasea the day by one and writes the new time to the RTC
Incrmonth:
   Incr _month
   If _month > 12 Then _month = 1        'no checking for month of year!!!!!
   Gosub Write1307time                   'use only to set time
   Waitms keydelay
   Key = 2                               'display DATE
Return
```

```
to identify the current period
'basedupon day and time and rotation
Getperiodstring:
   If Weekday = 1 Then                  'Mon
      Select Case Minuteofday
         Case Is < Mondayrc :           'before roll check
                        Minutesleft = Mondayrc - Minuteofday
                        Periodstr = Str(minutesleft) + " to go"
                        Periodflag = False       'display period? true=yes
         Case Is < Mondayp1 :           'before P1
                        Minutesleft = Mondayp1 - Minuteofday
                        Periodstr = Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayp2 :           'before P2
                        Minutesleft = Mondayp2 - Minuteofday
                        Periodstr = "p1-yr10       " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayint :          'before interval
                        Minutesleft = Mondayint - Minuteofday
                        Periodstr = "p2 yr11       " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayp3 :           'before P3
                        Minutesleft = Mondayp3 - Minuteofday
                        Periodstr = "1nteval       " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayp4 :           'before P4 begins
                        Minutesleft = Mondayp4 - Minuteofday
                        Periodstr = "p3 yr12       " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondaylunch :        'before lunch begins
                        Minutesleft = Mondaylunch - Minuteofday
                        Periodstr = "p4 yr13       " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayssr :          'before SSR begins
                        Minutesleft = Mondayssr - Minuteofday
                        Periodstr = "lunch         " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < Mondayp5 :           'before P5 begins
                        Minutesleft = Mondayp5 - Minuteofday
                        Periodstr = "ssr       "        '+ Str(minutesleft) + " to go"
                        Periodflag = False
         Case Is < Mondayend :          'before school ends
                        Minutesleft = Mondayend - Minuteofday
                        Periodstr = "p5        " + Str(minutesleft) + " to go"
                        Periodflag = True
         Case Is < 915 :                'before 3:20
                        Minutesleft = 0
                        Periodstr = "bye bye"
                        Periodflag = False
         Case Is > 914 :                '3:20 and on
                        Minutesleft = 0
                        Periodstr = ""
                        Periodflag = False

    End Select
   End If
```

```
If Weekday = 2 Then                     'tuesday ROTATION
      Msgstr = Msgstr + Rotationstr        'display rotation
      Msgstr = Msgstr + "    "
      Select Case Minuteofday
         Case Is < Tuesdayrc :              'before roll check begins
                          Minutesleft = Tuesdayrc - Minuteofday
                          Periodstr = Str(minutesleft) + " to go"
                          Periodflag = False
         Case Is < Tuesdayp1 :              'before P1 begins
                          Minutesleft = Tuesdayp1 - Minuteofday
                          Periodstr = Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayp2 :              'before p2 begins
                          Period = Rotation
                          Minutesleft = Tuesdayp2 - Minuteofday
                          Periodstr = Lookupstr(period , Tuett)        'get text from tue tt table
                          Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayint :             'before interval begins
                          Minutesleft = Tuesdayint - Minuteofday
                          Period = Rotation + 1
                          If Period > 6 Then Period = Period - 6
                          Periodstr = Lookupstr(period , Tuett)        'get text from tue tt table
                          Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayp3 :              'interval till before p3 begins
                          Minutesleft = Tuesdayp3 - Minuteofday
                          Periodstr = "1nteval        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayp4 :              'P3 till before P4 begins
                          Minutesleft = Tuesdayp4 - Minuteofday
                          Period = Rotation + 2
                          If Period > 6 Then Period = Period - 6
                          Periodstr = Lookupstr(period , Tuett)        'get text from tue tt table
                          Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdaylunch :           'before lunch begins
                          Minutesleft = Tuesdaylunch - Minuteofday
                          Period = Rotation + 3
                          If Period > 6 Then Period = Period - 6
                          Periodstr = Lookupstr(period , Tuett)        'get text from tue tt table
                          Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayssr :             'before SSR begins
                          Minutesleft = Tuesdayssr - Minuteofday
                          Periodstr = "lunch          " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < Tuesdayp5 :              'before  p5 Begins
                          Minutesleft = Tuesdayp5 - Minuteofday
                          Periodstr = "ssr       "        '+ Str(minutesleft) + " to go"
                          Periodflag = False
         Case Is < Tuesdayend :             'before school ends
                          Minutesleft = Tuesdayend - Minuteofday
                          Period = Rotation + 4
                          If Period > 6 Then Period = Period - 6
                          Periodstr = Lookupstr(period , Tuett)        'get text from tue tt table
                          Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                          Periodflag = True
         Case Is < 915 :                    '3:20    school ended
                          Minutesleft = 0
                          Periodstr = "bye bye"
                          Periodflag = False
         Case Is > 914 :
                          Minutesleft = 0
                          Periodstr = ""
                          Periodflag = False

      End Select
   End If
```

```
If Weekday = 3 Then                   ' wed
   Select Case Minuteofday
      Case Is < Wednesdaypd :          'before roll check begins
                     Minutesleft = Wednesdaypd - Minuteofday
                     Periodstr = Str(minutesleft) + " to go"
                     Periodflag = False
      Case Is < Wednesdayp1 :          'before  P1 begins
                     Minutesleft = Wednesdayp1 - Minuteofday
                     Periodstr = Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayp2 :          'before p2 begins
                     Minutesleft = Wednesdayp2 - Minuteofday
                     Periodstr = "p1 yr12       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayint :         'before interval begins
                     Minutesleft = Wednesdayint - Minuteofday
                     Periodstr = "p2 yr11       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayp3 :          'before p3 begins
                     Minutesleft = Wednesdayp3 - Minuteofday
                     Periodstr = "1nteval       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayp4 :          'before P4 begins
                     Minutesleft = Wednesdayp4 - Minuteofday
                     Periodstr = "p3 yr10       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdaylunch :       'before  lunch begins
                     Minutesleft = Wednesdaylunch - Minuteofday
                     Periodstr = "p4-yr10       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayssr :         'before before SSR begins
                     Minutesleft = Wednesdayssr - Minuteofday
                     Periodstr = "lunch         " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < Wednesdayp5 :          'before p5 Begins
                     Minutesleft = Wednesdayp5 - Minuteofday
                    Periodstr = "ssr        " + Str(minutesleft) + " to go"
                     Periodflag = False
      Case Is < Wednesdayend :         'before school ends
                     Minutesleft = Wednesdayend - Minuteofday
                     Periodstr = "p5 yr13       " + Str(minutesleft) + " to go"
                     Periodflag = True
      Case Is < 915 :                  '3:20    school ended
                     Minutesleft = 0
                     Periodstr = "bye bye"
                     Periodflag = False
      Case Is > 914 :
                     Minutesleft = 0
                     Periodstr = ""
                     Periodflag = False

 End Select
End If
```

```
If Weekday = 4 Then                   ' thu
     Select Case Minuteofday
        Case Is < Thursdayrc :          'before roll check begins
                       Minutesleft = Thursdayrc - Minuteofday
                       Periodstr = Str(minutesleft) + " to go"
                       Periodflag = False
        Case Is < Thursdayp1 :          'before P1 begins
                       Minutesleft = Thursdayp1 - Minuteofday
                       Periodstr = Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayp2 :          'before p2 begins
                       Minutesleft = Thursdayp2 - Minuteofday
                       Periodstr = "p1 yr13       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayint :         'before interval begins
                       Minutesleft = Thursdayint - Minuteofday
                       Periodstr = "p2 yr10       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayp3 :          'before p3 begins
                       Minutesleft = Thursdayp3 - Minuteofday
                       Periodstr = "1nteval       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayp4 :          'before P4 begins
                       Minutesleft = Thursdayp4 - Minuteofday
                       Periodstr = "p3       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdaylunch :       'before lunch begins
                       Minutesleft = Thursdaylunch - Minuteofday
                       Periodstr = "p4 yr11       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayssr :         'before SSR begins
                       Minutesleft = Thursdayssr - Minuteofday
                       Periodstr = "lunch         " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < Thursdayp5 :          ' before p5 Begins
                       Minutesleft = Thursdayp5 - Minuteofday
                      Periodstr = "ssr       " + Str(minutesleft) + " to go"
                       Periodflag = False
        Case Is < Thursdayend :         'school ends
                       Minutesleft = Thursdayend - Minuteofday
                       Periodstr = "p5 yr12       " + Str(minutesleft) + " to go"
                       Periodflag = True
        Case Is < 915 :                 '3:20    school ended
                       Minutesleft = 0
                       Periodstr = "bye bye"
                       Periodflag = False
        Case Is > 914 :
                       Minutesleft = 0
                       Periodstr = ""
                       Periodflag = False

   End Select
  End If
```

```
 If Weekday = 5 Then                  'friday rotation
      Msgstr = Msgstr + Rotationstr     'display rotation
      Msgstr = Msgstr + "    "
      Select Case Minuteofday
         Case Is < Fridayrc :           'before roll check begin
                  Minutesleft = Fridayrc - Minuteofday
                  Periodstr = Str(minutesleft) + " to go"
                  Periodflag = False
         Case Is < Fridayp1 :           'before P1 begins
                  Minutesleft = Fridayp1 - Minuteofday
                  Periodstr = Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < Fridayp2 :           'before P2 begins
                  Minutesleft = Fridayp2 - Minuteofday
                  Period = Rotation
                  Periodstr = Lookupstr(period , Fritt)        'get text from fri tt table
                  Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < Fridayint :          'beforeinterval begins
                  Minutesleft = Fridayint - Minuteofday
                  Period = Rotation + 1
                  If Period > 6 Then Period = Period - 6
                  Periodstr = Lookupstr(period , Fritt)        'get text from fri tt table
                  Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < Fridayp3 :           'before p3
                  Minutesleft = Fridayp3 - Minuteofday
                  Periodstr = "1nteval        " + Str(minutesleft) + " to go"
                  Periodflag = True
        Case Is < Fridayp4 :            'before p4
                  Minutesleft = Fridayp4 - Minuteofday
                  Period = Rotation + 2
                  If Period > 6 Then Period = Period - 6
                  Periodstr = Lookupstr(period , Fritt)        'get text from fri tt table
                  Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < Fridaylunch :        'begins lunch begins
                  Minutesleft = Fridaylunch - Minuteofday
                  Period = Rotation + 3
                  If Period > 6 Then Period = Period - 6
                  Periodstr = Lookupstr(period , Fritt)        'get text from fri tt table
                  Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < Fridayssr :          'before ssr begins
                  Minutesleft = Fridayssr - Minuteofday
                  Periodstr = "lunch        "        '+ Str(minutesleft) + " to go"
                  Periodflag = False
         Case Is < Fridayp5 :           'before p5 begins
                  Minutesleft = Fridayp5 - Minuteofday
                  Periodstr = "ssr        "        '+ Str(minutesleft) + " to go"
                  Periodflag = False
         Case Is < Fridayend :          '3:10    school ends
                  Minutesleft = Fridayend - Minuteofday
                  Period = Rotation + 4
                  If Period > 6 Then Period = Period - 6
                  Periodstr = Lookupstr(period , Fritt)        'get text from fri tt table
                  Periodstr = Periodstr + "        " + Str(minutesleft) + " to go"
                  Periodflag = True
         Case Is < 915 :                '3:20    school ended
                  Minutesleft = 0
                  Periodstr = "bye bye"
                  Periodflag = False
         Case Is > 914 :
                  Minutesleft = 0
                  Periodstr = ""
                  Periodflag = False

      End Select
   End If
Return
```

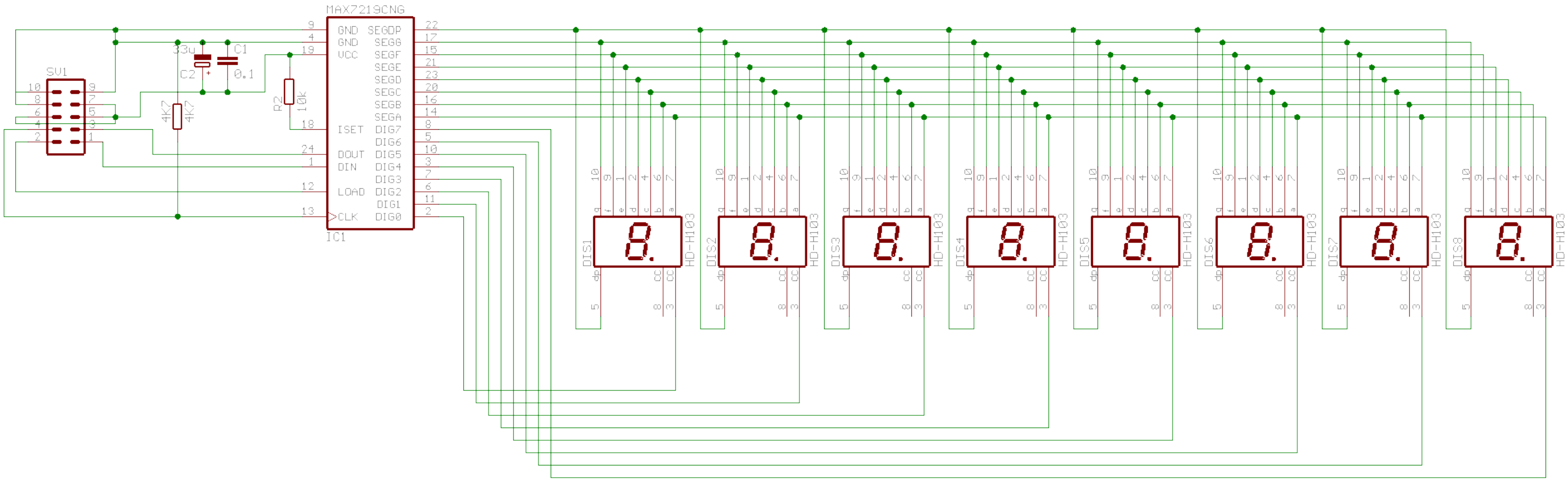
```
'-----------------------------------------------------------------

Converttime:
'Converts Time In Bytes To A String
   Timestr = ""
   If _hour < 10 Then Timestr = "0"
   Timestr = Timestr + Str(_hour)
   Timestr = Timestr + "x"                ' x = a dash
   If _min < 10 Then Timestr = Timestr + "0"
   Timestr = Timestr + Str(_min)
   'seconds
   Secstr = ""                            ' x = a dash
   If _sec < 10 Then Secstr = Secstr + "0"
   Secstr = Secstr + Str(_sec)
Return

Convertdate:
'converts date in bytes to a string
   Datestr = ""
   If _day < 10 Then Datestr = Datestr + "0"
   Datestr = Datestr + Str(_day)
   'Datestr = Datestr + "x"                ' x = a dash
   'Select Case _month
   '   Case 1 : Datestr = Datestr + "jan"
   '   Case 2 : Datestr = Datestr + "feb"
   '   Case 3 : Datestr = Datestr + "nar"
   '   Case 4 : Datestr = Datestr + "apr"
   '   Case 5 : Datestr = Datestr + "nay"
   '   Case 6 : Datestr = Datestr + "jun"
   '   Case 7 : Datestr = Datestr + "jul"
   '   Case 8 : Datestr = Datestr + "aug"
   '   Case 9 : Datestr = Datestr + "sep"
   '   Case 10 : Datestr = Datestr + "0ct"
   '   Case 11 : Datestr = Datestr + "n0v"
   '   Case 12 : Datestr = Datestr + "dec"
   '   Case Else : Datestr = "x x x x"
   'End Select
   If _month < 10 Then Datestr = Datestr + "0"
   Datestr = Datestr + Str(_month)
   Datestr = Datestr + "x" + Str(_year)
Return


'-----------------------------------------------------------------
Read1307time:                           'RTC Real Time Clock
   I2cstart
   I2cwbyte &B11010000                  'send device code (writing data)
   I2cwbyte 0                           'address to start sending from
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                  'device code (reading)
   I2crbyte _sec , Ack
   I2crbyte _min , Ack
   I2crbyte _hour , Ack
   I2crbyte Weekday , Ack
   I2crbyte _day , Ack
   I2crbyte _month , Ack
   I2crbyte _year , Nack
   _sec = Makedec(_sec)                 'convert 2xbcd in 1 byte to decimal byte
   _min = Makedec(_min)
   _hour = Makedec(_hour)
   Weekday = Makedec(weekday)
   _day = Makedec(_day)
   _month = Makedec(_month)
   _year = Makedec(_year)
   I2cstop
Return
```

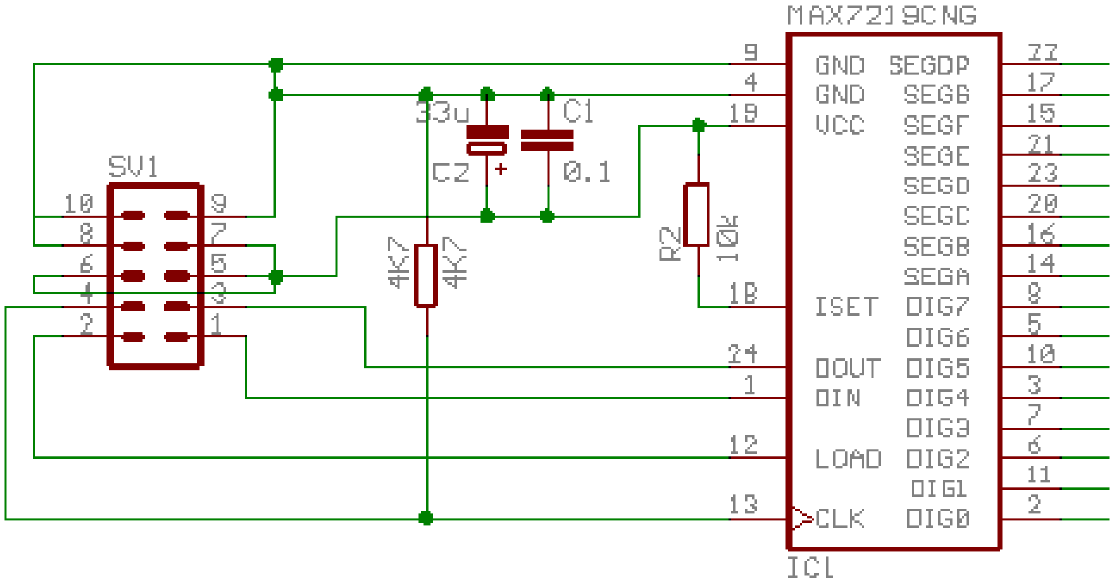
```
'write the time and date to the RTC
Write1307time:
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte &H00                            'send address of first byte to access
   Temp = Makebcd(_sec)                     'seconds
   I2cwbyte Temp
   Temp = Makebcd(_min)                     'minutes
   I2cwbyte Temp
   Temp = Makebcd(_hour)                    'hours
   I2cwbyte Temp
   Temp = Makebcd(weekday)                  'day of week
   I2cwbyte Temp
   Temp = Makebcd(_day)                     'day
   I2cwbyte Temp
   Temp = Makebcd(_month)                   'month
   I2cwbyte Temp
   Temp = Makebcd(_year)                    'year
   I2cwbyte Temp
   I2cstop
Return


Write1307ctrl:
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte &H07                            'send address of first byte to access
   I2cwbyte &B10010000                      'start squarewav output 1Hz
   I2cstop
Return


Start1307clk:
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte 0                               'send address of first byte to access
   I2cwbyte 0                               'enable clock-also sets seconds to 0
   I2cstop
Return


Write1307ram:
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte Ramlocation                     'send address of byte to access
   I2cwbyte Ramvalue                        'send value to store
   I2cstop
Return


'routine to read the contents of one ram location
'setup ramlocation first and the data will be in ramvalue afterwards
'no error checking ramlocation should be from &H08 to &H3F (56 bytes only)
Read1307ram:
   I2cstart
   I2cwbyte &B11010000                      'send device code (writing data)
   I2cwbyte Ramlocation                     'send address of first byte to access
   I2cstop
   Waitms 50
   I2cstart
   I2cwbyte &B11010001                      'device code (reading)
   I2crbyte Ramvalue , Nack
   I2cstop
Return
```

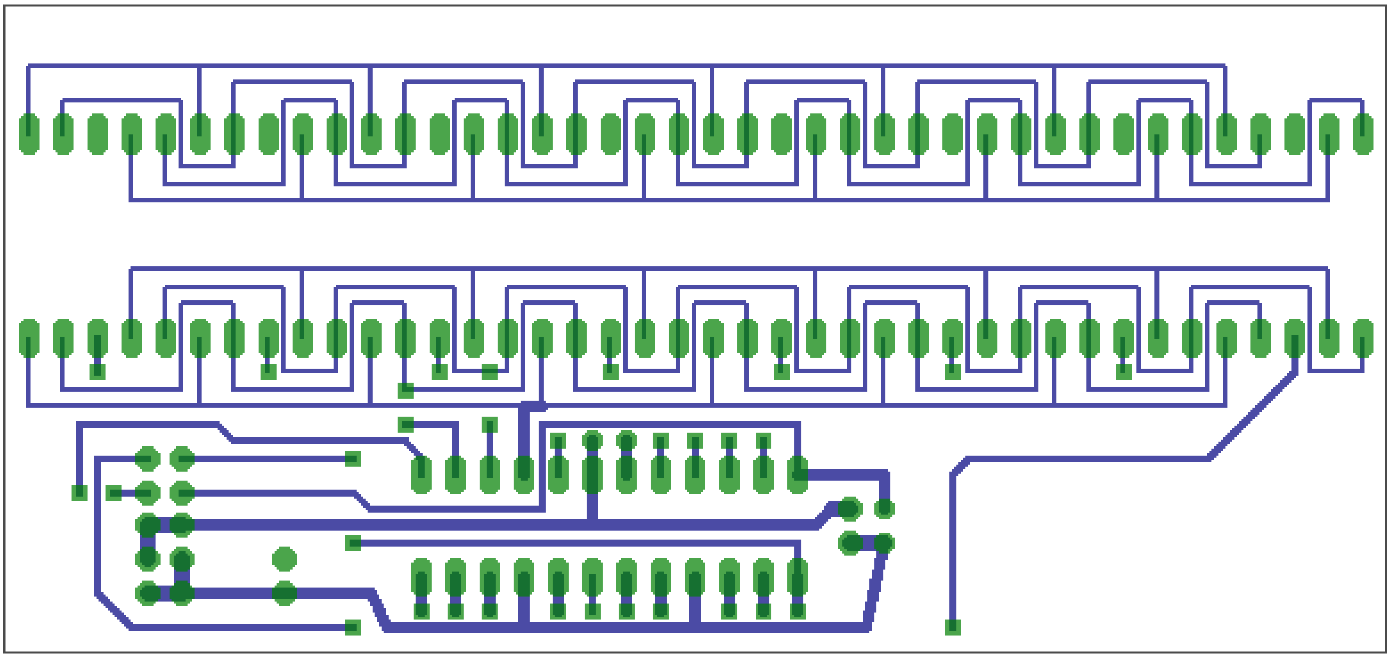
```
Clear1307ram:
   Ramvalue = 00
   Ramlocation = &H08
   I2cstart
   I2cwbyte &B11010000                    'send device code (writing data)
   I2cwbyte Ramlocation                   'send address of byte to access
   For Ramlocation = &H08 To &H3F
       I2cwbyte Ramvalue                  'send value to store
   Next
   I2cstop
Return

Writeram:
   Ramlocation = &H08
   Ramvalue = 111
   Gosub Write1307ram
   Ramlocation = &H09
   Ramvalue = 222
   Gosub Write1307ram
Return

Readram:
   Cls
   Ramlocation = &H08
   Gosub Read1307ram
   Lcd Ramvalue
   Lcd ":"
   Ramlocation = &H09
   Gosub Read1307ram
   Lcd Ramvalue
   Ramlocation = &H0A
   Gosub Read1307ram
   Lcd ":"
   Lcd Ramvalue
   Wait 5
Return

'------------------------------------------------------------------
'Interrupts
'message scrolling
Timer1_messagescroll:
   Timer1 = Scrolltime
   'copy 7 digits from message string into dispstring
   Dispstr = Mid(msgstr , Msgptr , 7)
   'only scroll if more than 7 digits
   If Msglen > 7 Then
       Incr Msgptr                        'Move Msgptr
       If Msgptr > Msglen Then Msgptr = 1
   Else                                   'added  080510 for test mode
       Msgptr = 1
   End If
   'Gets each character from the dispstr
   ' looks up the binary for that character
   ' and puts it into the digit array
   For I = 1 To 7
      Ascii = Mid(dispstr , I , 1)
      Asci = Asc(ascii)                   'convert asc to A NUMBER
      Select Case Asci                    'convert assci to index for table below
         Case 0 To 47 : Asci = 25         'ignore non alpha, use spaces
         Case 48 To 57 : Asci = Asci - 22        'digits 0 to 9
         Case 57 To 96 : Asci = 25        'uppercase plus others
         Case 97 To 122 : Asci = Asci - 97        'lowercase
         Case Else : Asci = 25
      End Select
      Digit(i) = Lookup(asci , Text)
   Next
Return
```

```
'digit scanning , gets 1 digit at a time to display it
Timer0_digitscan:
   Timer0 = Scantimer                       ' preload timer
   'only 1 digit can be displayed at a time
   'so put data for next digit onto the segments
   'then turn on the next digit
   Incr Dig
   If Dig = 8 Then Dig = 1                 'max is 7 digits
   Segmentbus = Digit(dig)                 'get segmentsdata for this digit
   Select Case Dig                         'turn on one digit
      Case 1 : Digitbus = &B10000000       'note there is no 0 digit
      Case 2 : Digitbus = &B01000000
      Case 3 : Digitbus = &B00100000
      Case 4 : Digitbus = &B00010000
      Case 5 : Digitbus = &B00001000
      Case 6 : Digitbus = &B00000100
      Case 7 : Digitbus = &B00000010
   End Select
 Return

'lookup tables FOR DIGIT DISPLAY
'code for each segment to identify it   'segments = &b C D E B A F G 0
'even though some characters appear as capitals only use small letters in the text
Text:
'A,b,C,d,E,F
Data &B10111110 , &B11100110 , &B01101100 , &B11110010 , &B01101110 , &B00101110
Text2:
'G,h,i,J, ,L
Data &B11101100 , &B10100110 , &B00100000 , &B11110000 , &B00000000 , &B01100100
Text3:
', ,n,o,P, ,r
Data &B00000000 , &B10111100 , &B11100010 , &B00111110 , &B00000000 , &B00100010
Text4:
'S,t,u, , , ,
Data &B11001110 , &B01100110 , &B11110100 , &B11110100 , &B00000000 , &B00000010
Text5:
'Y,-,
Data &B11010110 , &B00000000
Numbers:
'0,1,2,3,4
Data &B11111100 , &B10010000 , &B01111010 , &B11011010 , &B10010110
'5,6,7,8,9
Data &B11001110 , &B11101110 , &B10011000 , &B11111110 , &B10011110

'use bascom to get dayofyear calc
'divide dayofyear by 7 to get week
'note that the first week of the year will be 0 not  1
Weekrotation:
'2010
Data 0 , 0 , 0 , 0 , 0 , 0               'weekofyear 0,1,2,3 ,4
Data 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 0 , 0      'weekofyear 5-12 -  rotations started in 2nd week
Data 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 5 , 6 , 1 , 0 , 0       'weekofyear 17-26
Data 2 , 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 5 , 0 , 0       'weekofyesr 29-38
Data 6 , 1 , 2 , 3 , 4 , 5 , 6 , 1 , 0 , 0 , 0       'weekofyear 41-49
'2009
'Data 0 , 0 , 0 , 0                       'weekofyear 0,1,2,3
'Data 0 , 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 0 , 0       'weekofyear 4-14
'Data 5 , 6 , 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 0 , 0       'weekofyear 17-26
'Data 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 5 , 6 , 0 , 0       'weekofyesr 29-38
'Data 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 3 , 0 , 0 , 0       'weekofyear 41-49
'2008
'Data 0 , 0 , 0 , 0 , 0                   'weekofyear 0,1,2,3,4
'Data 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 5 , 0 , 0       'weekofyear 5-15...
'Data 6 , 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 0 , 0       'weekofyear 18-26
'Data 3 , 4 , 5 , 6 , 1 , 2 , 3 , 4 , 5 , 6 , 0 , 0       'weekofyesr 29-38
'Data 1 , 2 , 3 , 4 , 5 , 6 , 1 , 2 , 3 , 0 , 0 , 0       'weekofyear 41-48

Tuett:                                  'note blank data as lookup starts at 0
Data "" , "p1 yr11" , "p2 yr12" , "p3 yr13" , "p4 yr10" , "" , "p6-yr10"
Fritt:
Data "" , "p1 yr10" , "p2-yr10" , "p3 yr11" , "p4 yr12" , "p5 yr13" , "        "
```

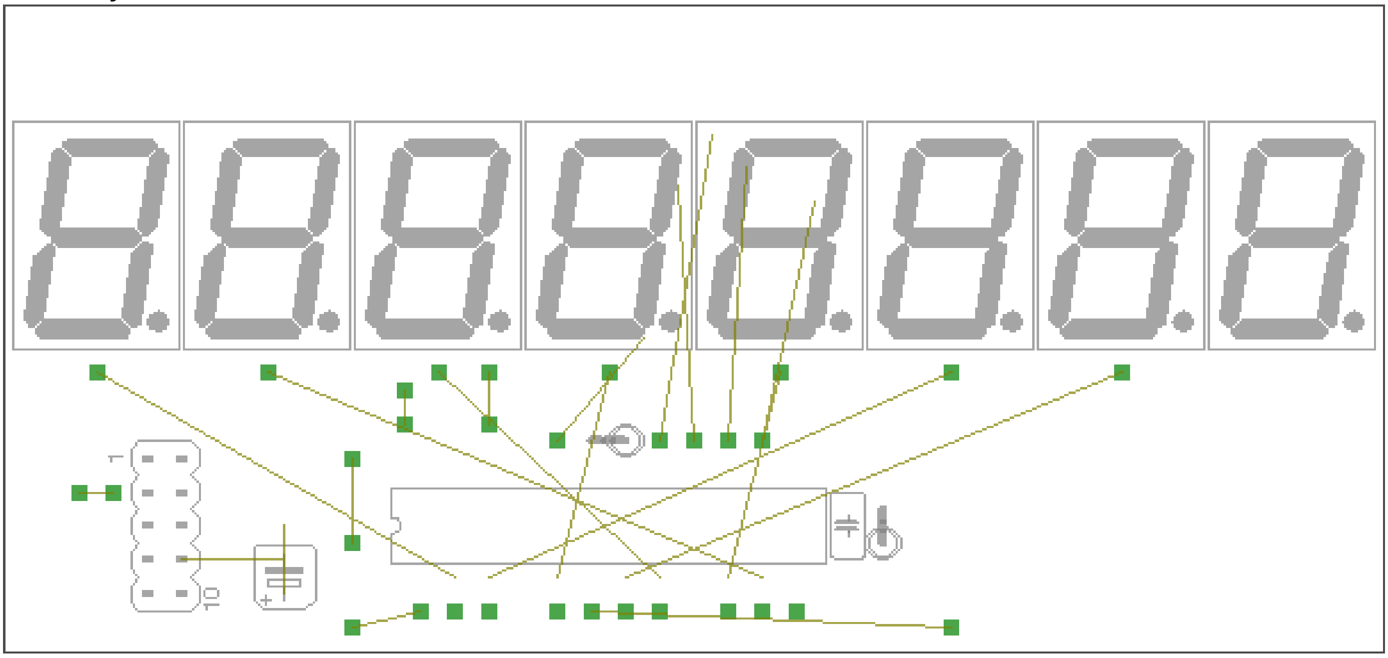
52 The MAX 7219/7221 display driver IC's







PCB Layout



```
'-----------------------------------------------------------------
'     Author:     B. Collis
'     Date:       14 April 2003
'     Version:    2.00
'     File Name:  7219_v2.00.bas
'-----------------------------------------------------------------
'     Description:
'     This program drives the max7219/7221 Display Driver IC and
'       eight 7-segment displays
'     Display initialisation is in a routine
'     A small subroutine handles the clocking of data to the display
'     So far this program only sets up the display, puts 1 to 8 on
'       the digits and then flashes them all on and off
'-----------------------------------------------------------------
'     Compiler Directives
$crystal = 8000000                    ' calculate delays accurately
$regfile = "m8535.dat"                ' so compiler can identify
                                      ' particular micro features
'-----------------------------------------------------------------
'     Hardware Setups
'     setup direction of all ports
Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output

Porta = 255                           'turn off LEDs on ports
Portb = 255
Portc = 255
Portd = 255

Disp_data Alias Portb.0               'Data into 7219
Disp_load Alias Portb.3               'Load
Disp_clk Alias Portb.2                'clock

'-----------------------------------------------------------------
'     Constants
Const Timedelay = 75

'-----------------------------------------------------------------
'     Declare Variables
Dim Command As Integer

'-----------------------------------------------------------------
'     Program starts here

Gosub Max_init
Gosub Max_1on1
Do
  Gosub Max_flash
Loop
End                                   'end program
```

```
'    Sub-routines here
Max_1on1:
  Command = &H0101                              '1 on display no. 1
  Gosub Max_disp
  Command = &H0202                              '2 on display no. 2
  Gosub Max_disp
  Command = &H0303                              '3 on display no. 3
  Gosub Max_disp
  Command = &H0404                              '4 on display no. 4
  Gosub Max_disp
  Command = &H0505                              '5 on display no. 5
  Gosub Max_disp
  Command = &H0606                              '6 on display no. 6
  Gosub Max_disp
  Command = &H0707                              '7 on display no. 7
  Gosub Max_disp
  Command = &H0808                              '8 on display no. 8
  Gosub Max_disp
Return
'subroutine to initialise the display
Max_init:
  Command = &H0F01                              'display test on
  Gosub Max_disp
  Waitms 1000
  Command = &H0F00                              'display test off
  Gosub Max_disp
  Waitms 1000
  Command = &H0C01                              'normal operation
  Gosub Max_disp
  Command = &H09FF                              'decode mode bcd all digits
  Gosub Max_disp
  Command = &H0A02                              'set intensity 0=min  F=max
  Gosub Max_disp
  Command = &H0B07                              'all 7 digits active
  Gosub Max_disp
Return
'subroutine to flash display on and off  does not flash individual digits but shutsdown the IC
' and puts it back into normal operation
Max_flash:
  Command = &H0C00                              'shutdown display
  Gosub Max_disp
  Waitms 1500
  Command = &H0C01                              'normal operation
  Gosub Max_disp
  Waitms 1500
Return

'simple routine to clock data out to the display
Max_disp:
  Reset Disp_load
  Shiftout Disp_data , Disp_clk , Command , 1    'msb first
  Set Disp_load
Return
```

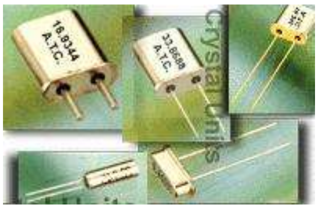
52.1 AVR clock/oscillator

The AVR executes instructions at the rate set by the system clock (oscillator). There are a number of different ways that this clock can be set up using either internal components of the micro or external components. These are:

- Internal Resistor-Capacitor (lesser accuracy)
- External RC
- External Ceramic Resonator
- External Crystal (more accuracy)

ceramic resonator

crystals

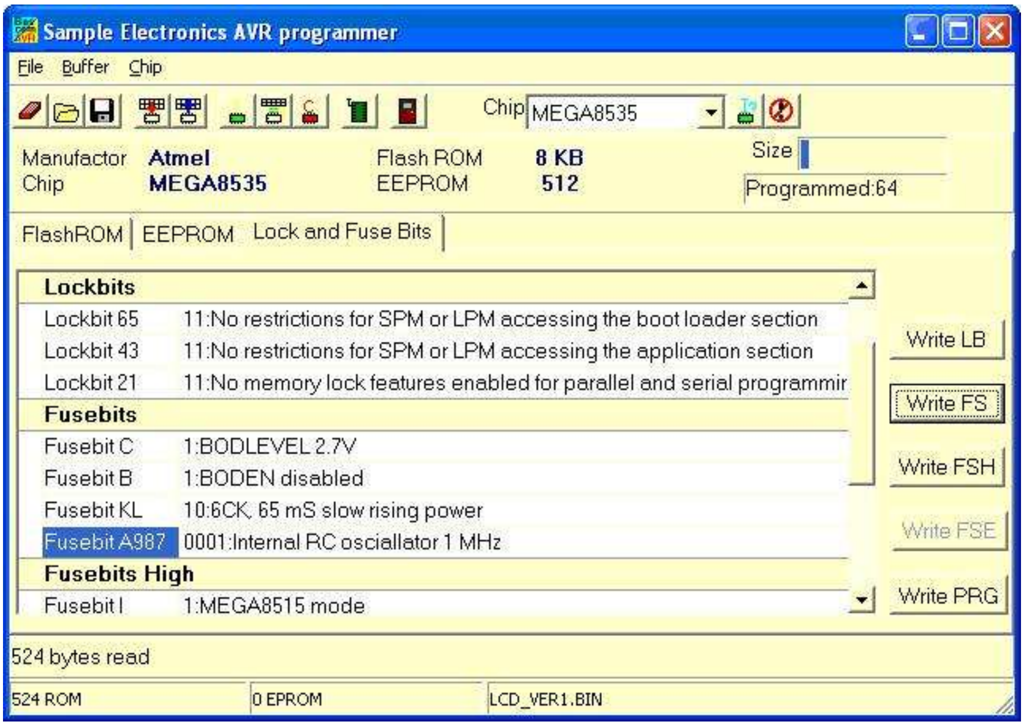


Within the micro reprogrammable fuse links (just like the links on a computer motherboard but set via software) are used to determine which method is used.

The ATMega8535-16PI clock can range up to 16MHz, however initially it is configured to run from the internal RC clock at a 8MHz rate.

In BASCOM when the micro is connected and powered up the settings can be changed by selecting MANUAL PROGRAM.

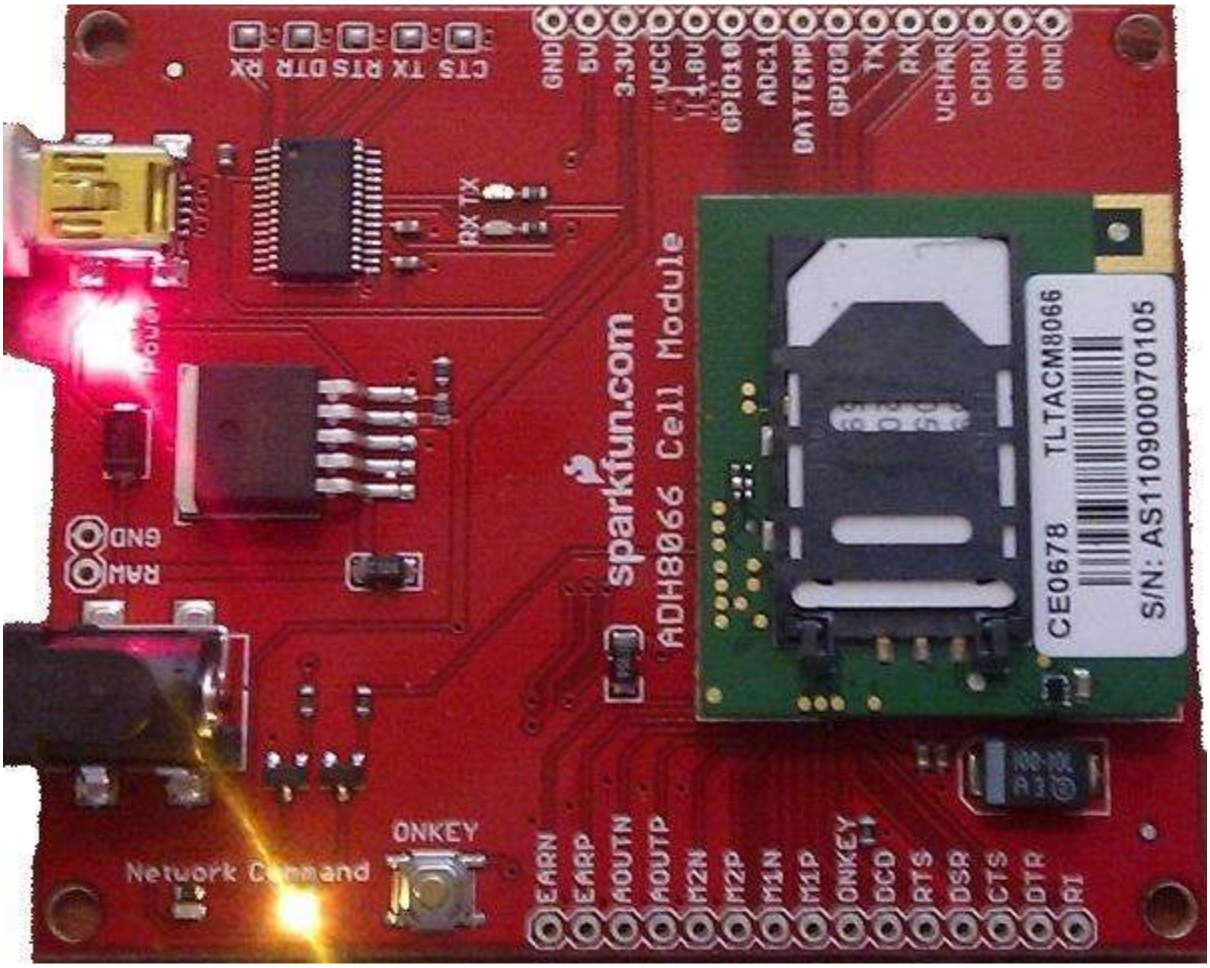
From the window that appears select the LOCK AND FUSE BITS tab. Bascom will then read the current settings.



The Internal RC oscillator may be changed to 1, 2 or 4MHz by selecting the line in the window and using the drop down that appears to.

After changing the Fusebit settings select the Write FS button. After it has programmed the fusebits, select the FlashRom tab before exiting (YOU MAY NEED TO DISABLE THE JTAG SETTING AS WELL) DO NOT CHANGE ANYTHING ELSE, YOU RISK STUFFING UP YOUR MICRO!

53 Cellular Connectivity-ADH8066



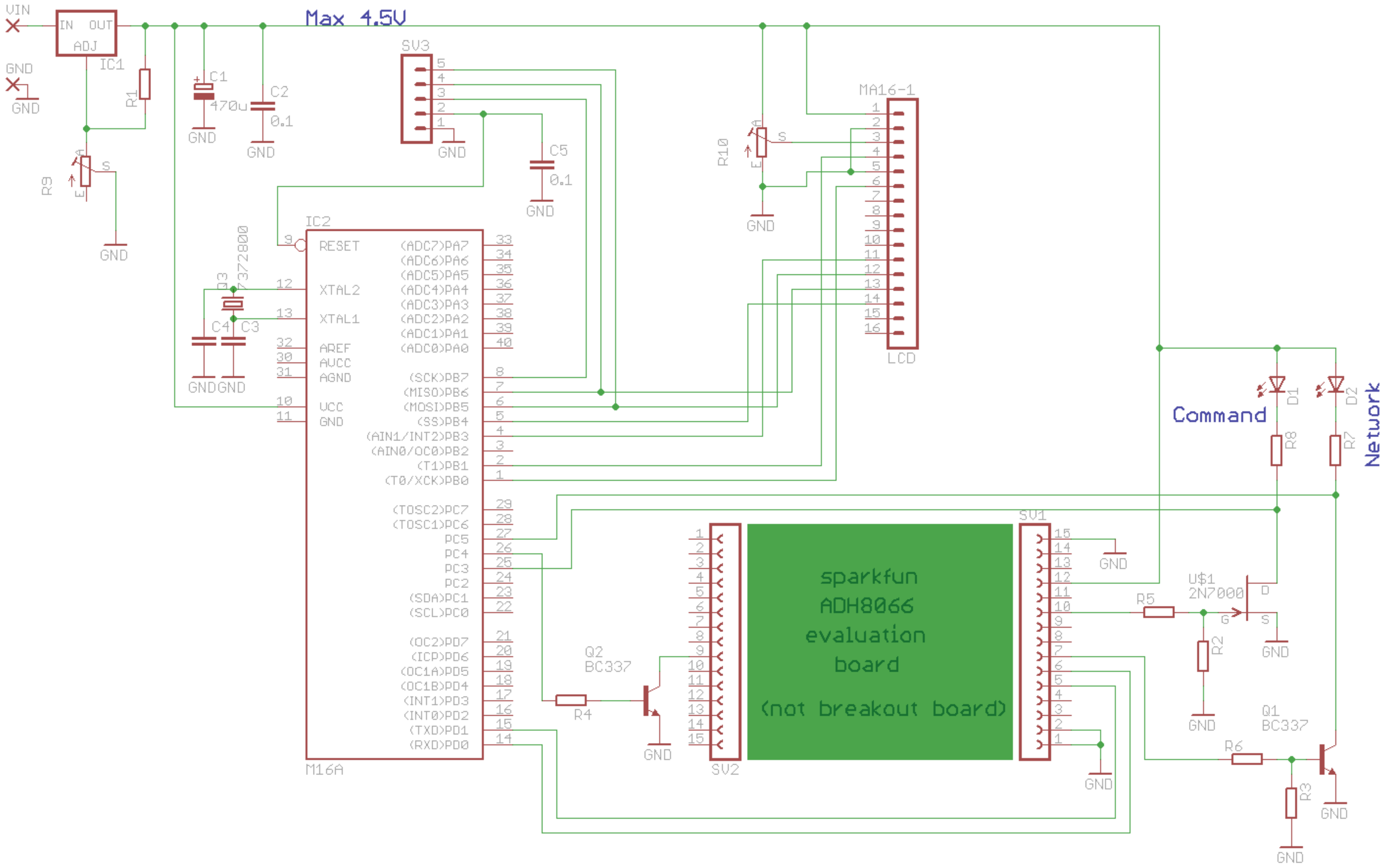
The ADH8066 is a cellular module from www.sparkfun.com. The module is the green PCB with the SIM card on it. The larger board is sparkfun's evaluation board.

The photos below show both sides of the ADHmodule.

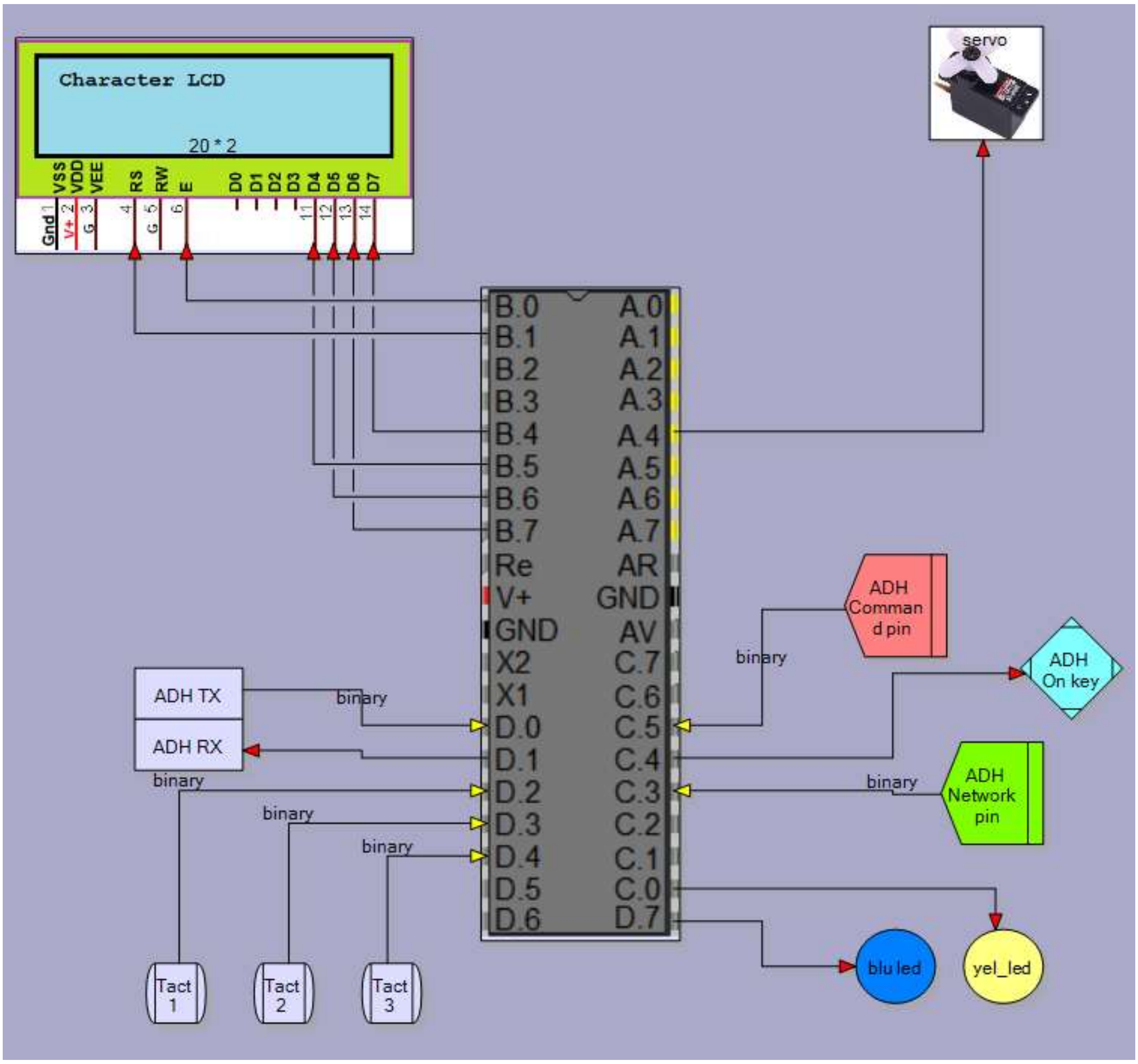
53.1 ADH prototype development

The evaluation board was built up into a circuit using an ATMega16 and a 20x4 charcter LCD on veroboard.

Even though the evaluation board was used, no features of it were used, the power supply was provided by an LM350 voltage regulator on the veroboard not through the voltage regulator on the ADH evaluation board. This prototype was made this way as the eval board was at hand and the circuit design was made to help students design boards for their own projects which would use the breakout board. With the breakout board you must connect DSR0 to DTR0 and both to ground via a 1K resistor)



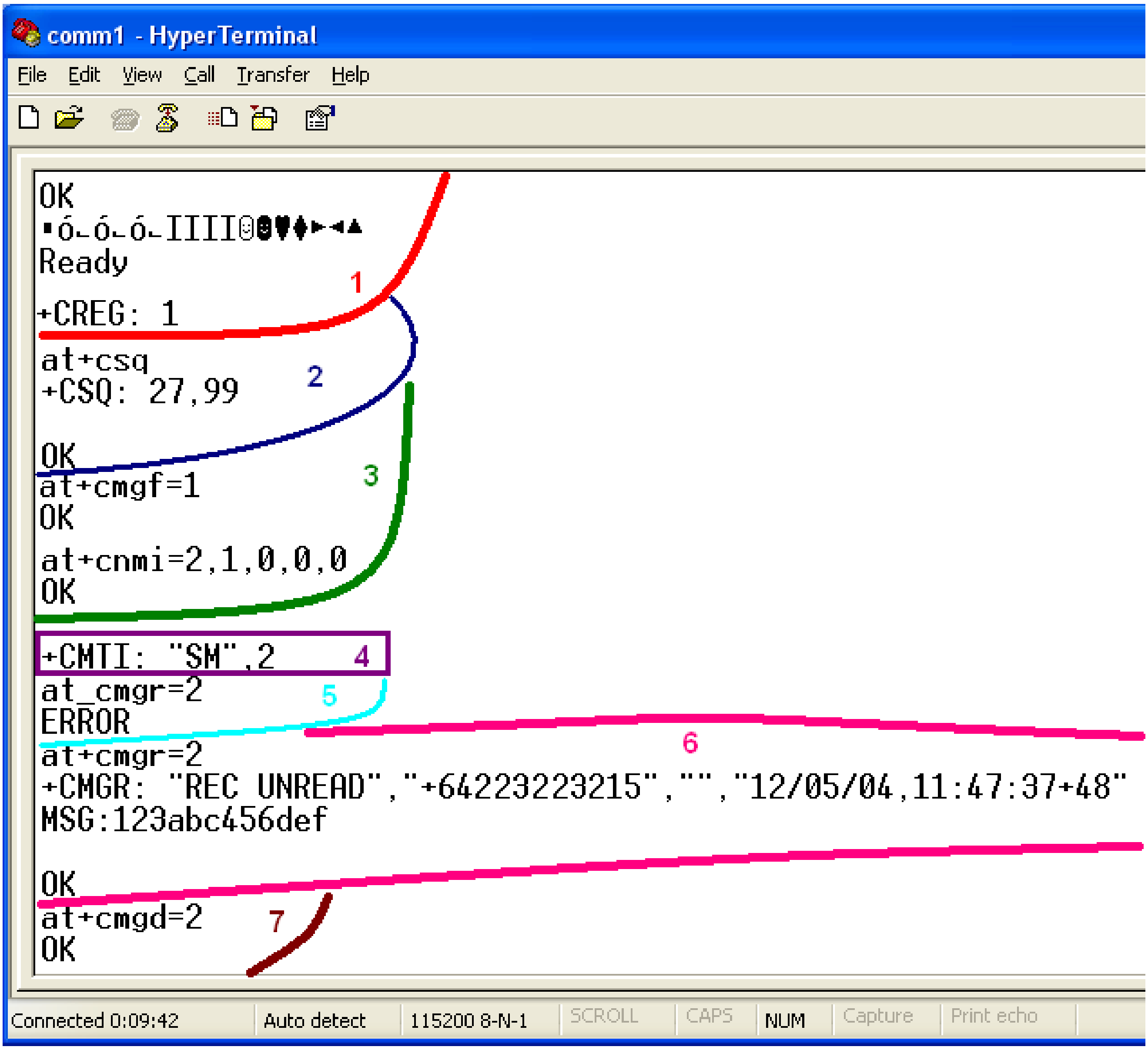
53.2 ADH initial test setup block diagram



Block diagram and schematic explanation:

1. ADH ON_KEY: a transistor circuit using a BC337 is used to pull the ON KEY input low when portC.4 is taken high.
2. The COMMAND and NETWORK outputs of the ADH Eval board are taken via transistors to two input pins of the AVR PinC.3 and PinC.5.
3. A 2N7000 FET was used as one of those just to show that a FET could be used just as effectively as a transistor as an interface.
4. Note how the input pins of the AVR are connected to the outputs of the transistor circuits not the output pins of the ADH module.
5. The RX pin of the ADH is connected tothe TX pin of the AVR
6. The TX pin of the ADH is connected to the RX pin of the AVR
7. The ADH communicates at 115200 baud 8N1 (8 bit, no parity, 1 stop bit) No flow control is required.
8. An external crystal is used for the AVR, 7.372800Mhz, at such a high baud rate of 115200 using the internal clock or a crystal

53.3 Process for using the ADH



Here the ADH evaluation board is connected to a PC via a USB cable and under the control of hyperterminal. In the above screen shot the text in lower case I typed the text in upper case was received from the ADH. Note in the descriptionhere I refer to both ‘message’ and ‘sms’, a message is the serial communication sent from the ADH to the microcontroller; an sms is the text message from another cellular phone to the ADH.

1. Power is applied to the evaluation board.
   - The ON KEY input is pulled low for over 2 seconds then released high.
   - The Command LED will come on
   - The ADH sends a bunch of characters inclusing the text IIII and the text READY to the AVR (IIII is a unique message so we can detect this to see that the adh is alive and ok and we are reading the serial comms properly).
   - Then the ADH module will try and register with a cellular network. I put in a prepay Vodafone NZ sim card and the network LED came on within 15 seconds.
   - This turns on the LED and giving a hardware input to the AVR that the network is on .
   - The module sends CREG+1 to the AVR for registration succesful, or +CREG: 3 for network denied, or +CREG: 0 for no network(is the antenna unplugged?).

- For testing purposes I put in an old sim card that had expired (not been topped up with credit for over 12months) and it responded initially with +CREG: 1 and the network pin went on, then a few seconds later sent +CREG: 3 – network denied and the network pin turned off. Testing for CREG:1 at this stage is not a good idea as it could mislead you)

2. The module is ready so it can now be controlled using AT commands.
    - We can send AT+CPIN? To check the sim card is ok and the ADH responds +CPIN: READYOK;
    - We can send AT+CREG=? And the ADH responds +CREG: 1 (we should test thisoften in our program to see if everything is ok).
    - However for the above test I sent AT+CSQ and the ADH responded with signal strength e.g. CSQ: 27/99 and OK (the number should range from 5/99 to 31/99). CSQ: 99/99 means no signal (did you plug in the antenna?)
    - We should test this often in our program tosee if all is ok
3. I then put the module into txt mode using at+cmgf=1 and set the module to notify us when a new sms comes in with at+2,1,0,0,0.
4. An sms was received and the module sent +CMTI: “SM”,2. This means that a message has arrived and it is in the sim memory in slot 2.
    - I could put the ADH into a mode where the message is delivered automatically, but chose to have an indication delivered instead.
    - Note that the default setting is to have no indication from the ADH that an sms has arrived.
5. I tried to retrieve the message but made an error
6. I retrieved the message with at+cmgr=2 as it is in memory slot 2.
    - It has “REC UNREAD” as it is the first time I have read the message. Every message is tagged as READ or UNREAD and there is a command to read all messages or all unread messages.
    - The number it cam from
    - The time and date it was received
    - The message I sent “MSG:123abc456def”
    - The OK response.
    - Note this message from the ADH is 60 characters long plus the actual sms contents, making it over 80 characters in length including any non printable charactersa such as the 4 CR’s and LF’s that are in the message.
7. I sent the command at+cmgd=2 to delete the message .

The final software will have to do an extensive start up routine to determine that the ADH is ok and on the network. A good point ot note is to use a prepaid cellular account for this sort of system rather than an account where you are billed. If the system locks up and sends a lot of messages then it could become very costly!$!

53.4 ADH communications

The ADH will send data to the microcontroller and can send a lot of data at once especially if you tell it to send a stored message to you. So the serial communications requires a buffer to hold all the incoming information otherwise we would only see some of it coming in.
Config Serialin = Buffered , Size = 200
The way the buffering works in Bascom is that when you compile your program Bascom sets aside RAM (200 bytes in this case) to hold the incoming data. This is a circular buffer so if too much data arrives then data past 200 characters will overwrite the beginning of the buffer and you will begin to lose data. If you read the data from the buffer before new data arrives then it wont be lost. Data is read using the INKEY() function in Bascom (or you can have the program wait for data to come in using WAITKEY).

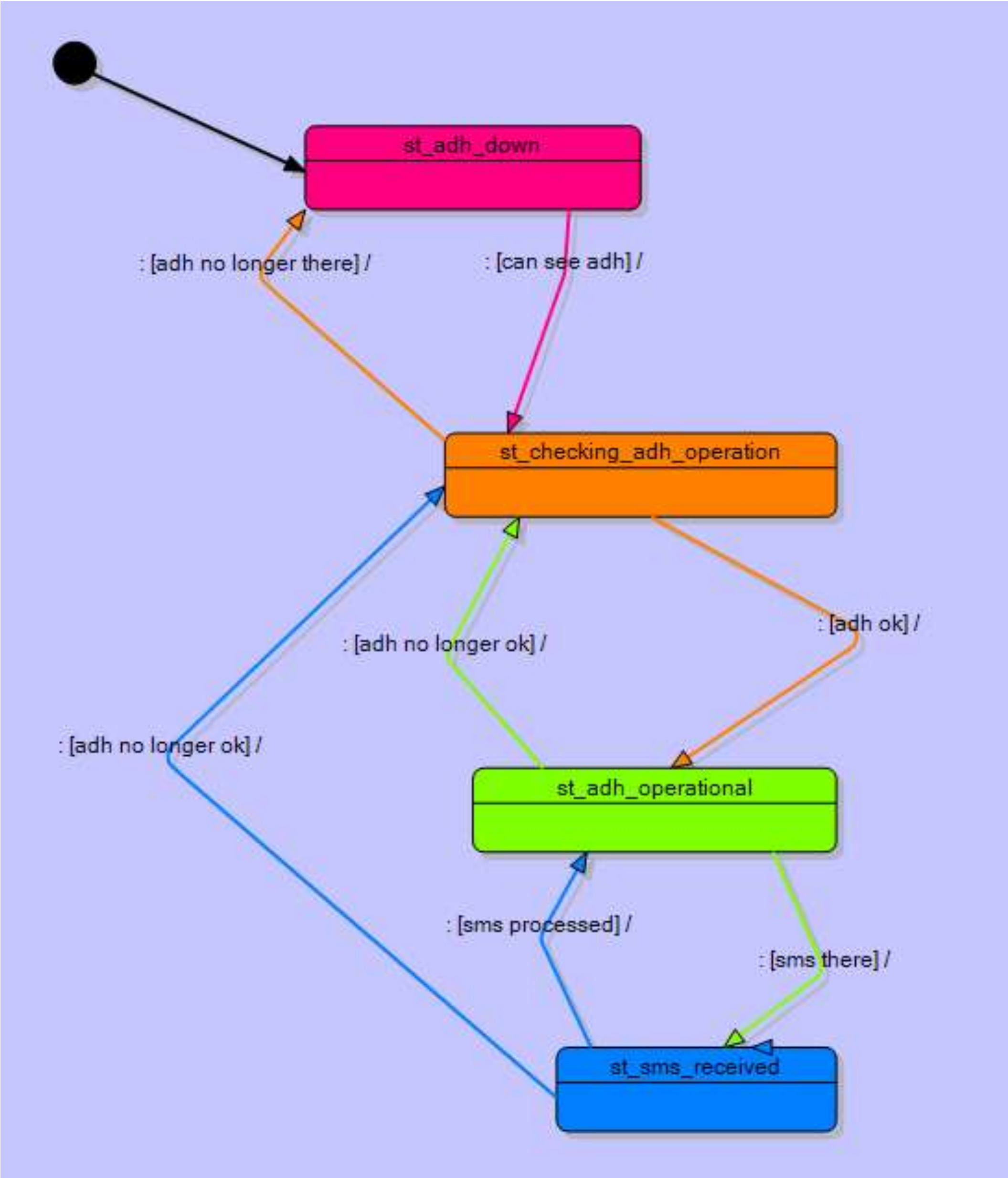
This routine checks to see if new data has come in and then copies it to a string we have dimensioned. Now it seems a bit redundant to have 2 buffers for the data coming in, one that Bascom dimensioned and one that we dimensioned, and you could just use the Bascom buffer if you really wanted to. However because it is a circular buffer then data can be spread from the end of it to the beginning making it hard for doing things with. So unless you need to really conserve ram space having you own buffer as well is much easier.

```
Check_for_adh_comms:
   I = Ischarwaiting()                       'see if buf has something (I=0 Is no, I=1 is yes)
   While I = 1                               'copy all chars to our string
      Bytein = Inkey()                       'get one char
      If Bytein > 31 Then                    'if printable char
         Adh_rcvd_message = Adh_rcvd_message + Chr(bytein)          'add
      End If
      I = Ischarwaiting()                    'see if any more charaters in meesage
      Set Adh_new_mesg_flag                  'flag that our string has something
   Wend                                      'if no data exit the loop
Return
```

Note how a while-wend is used here rather than a do-loop-until. The big difference here explains why we have both in programming. A while –wend may never execute at all, so if there is no data to read it will skip past and return from the subroutine. A do-loop-until will always be executed at least once, and we don't want this as it will try and process data that isn't there!

There is a second way of doing this and that is to use a serial interrupt, Bascom has this built into it, however I decided to not use the interrupt. I did this because I don't think speed of getting to the buffer is critical for my application.

53.5 Initial state machine



There are several different operational aspects of the device to keep track of:

- is the hardware ok?
- Can we talk to it?
- Can it register on the network?
- Is the sim ok?
- What is the signal strength?
- Is their credit?

The initial thoughts about the different states of the device are that it is:

- DOWN
- TRYING TO BECOME OPERATIONAL
- OPERATIONAL
- RECEIVED AN SMS

53.6 Status flags

A number of binary flags were created to keep track of all the different things happening within the system. There are (at least) two ways of doing this.

- I could dimension a whole lot of flags individually e.g. dim adh_sim_flag as bit, adh_creg_flag as bit and so on.

- Or do what I chose to do which was to dimension a status variable (dim adh_status as word) and then allocate my flags to individual bits within that variable using the alias command.

```
'status bits
Adh_com_pin_flag Alias Adh_status.15     'command pin hardware connection ok
Adh_nw_pin_flag Alias Adh_status.14      'network pin hardware connection ok
Adh_alive_flag Alias Adh_status.13       'serial comms is working between micro and ADH
Adh_creg_flag Alias Adh_status.12        'ADH is registerd on cell nw
Adh_sim_flag Alias Adh_status.11         'sim card is functioning ok
Adh_ss_flag Alias Adh_status.10          'signal strength ok (not 99)
Adh_echo_flag Alias Adh_status.9         'we turned echo off
Adh_sms_mode_flag Alias Adh_status.8     'we set sms mode
Adh_ok_flag Alias Adh_status.7           'get yourself a smiley here!
Adh_sms_rcvd_flag Alias Adh_status.6     'an sms has been received from other cellphone
Adh_$_flag Alias Adh_status.5            'we have credit to send
Adh_new_mesg_flag Alias Adh_status.4     'new serial message from adh to micro
Adh_ok_rcvd_flag Alias Adh_status.3      'ADH all functioning ok
Credit_bal_flag Alias Adh_status.2       'credit over $1.00
Adh_sms_sending_flag Alias Adh_status.1 'sfter send while waiting for sent response
Adh_error_flag Alias Adh_status.0        'had an error returned from the adh
Adh_status = 0                           'reset all flags for initial start
```

The reason I chose the second way is because I wanted to be able to display them all easily at once on the LCD, especially during the initial stages of programming.

```
Adh_com_pin_flag Alias Adh_status.15
Adh_nw_pin_flag Alias Adh_status.14
```

these two flags will be used to tell us that the ADH pin outputs command and network are functioning. At any stage if either of these drop out then there is a problem with our system.

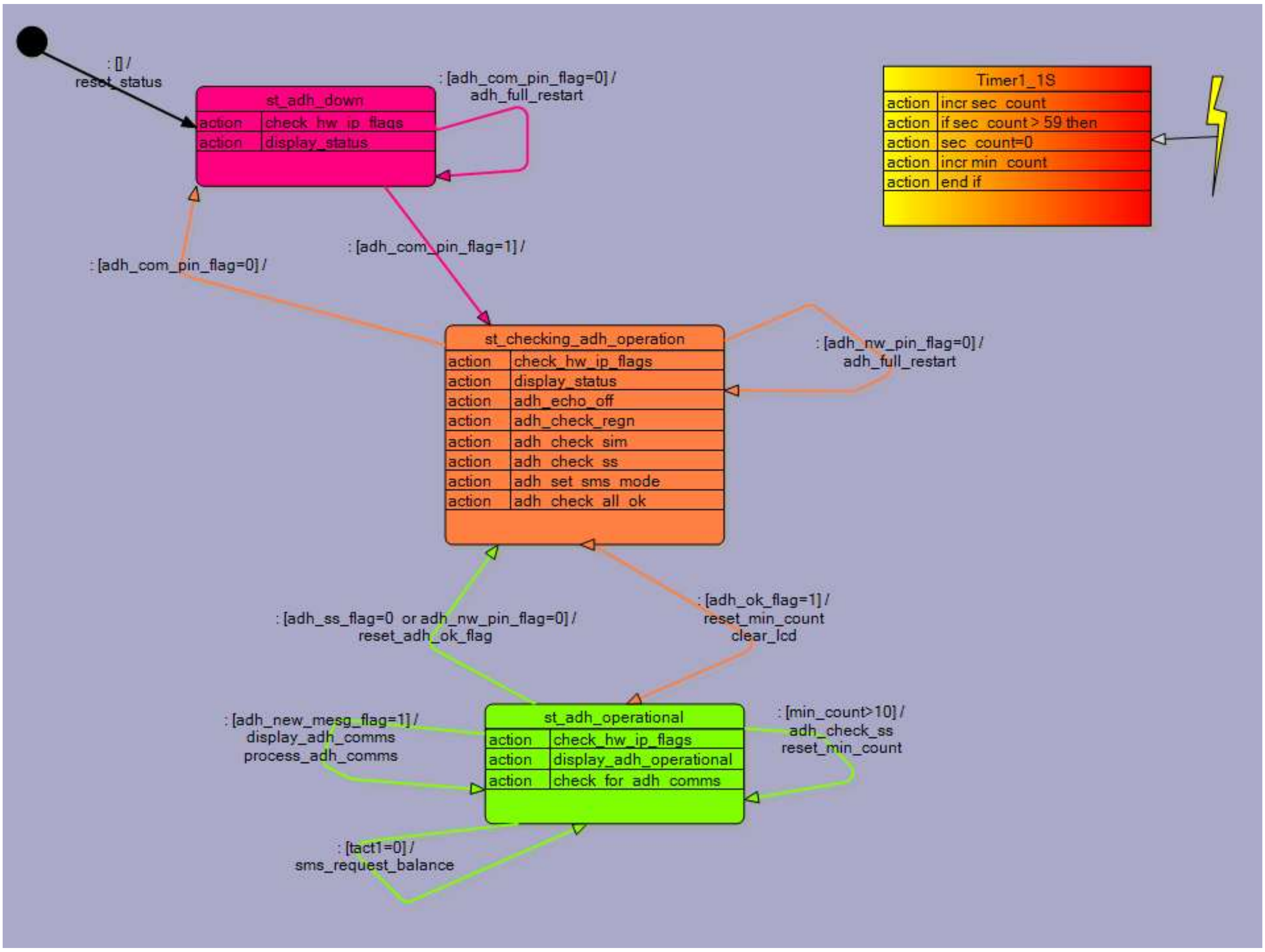
```
Adh_alive_flag Alias Adh_status.13       'serial comms is working between micro and ADH
Adh_creg_flag Alias Adh_status.12        'ADH is registerd on cell nw
Adh_sim_flag Alias Adh_status.11         'sim card is functioning ok
Adh_ss_flag Alias Adh_status.10          'signal strength ok (not 99)
```

these flags inidicate to us that the ADH is functioning ok.

```
Adh_echo_flag Alias Adh_status.9         'we turned echo off
Adh_sms_mode_flag Alias Adh_status.8     'we set sms mode
```

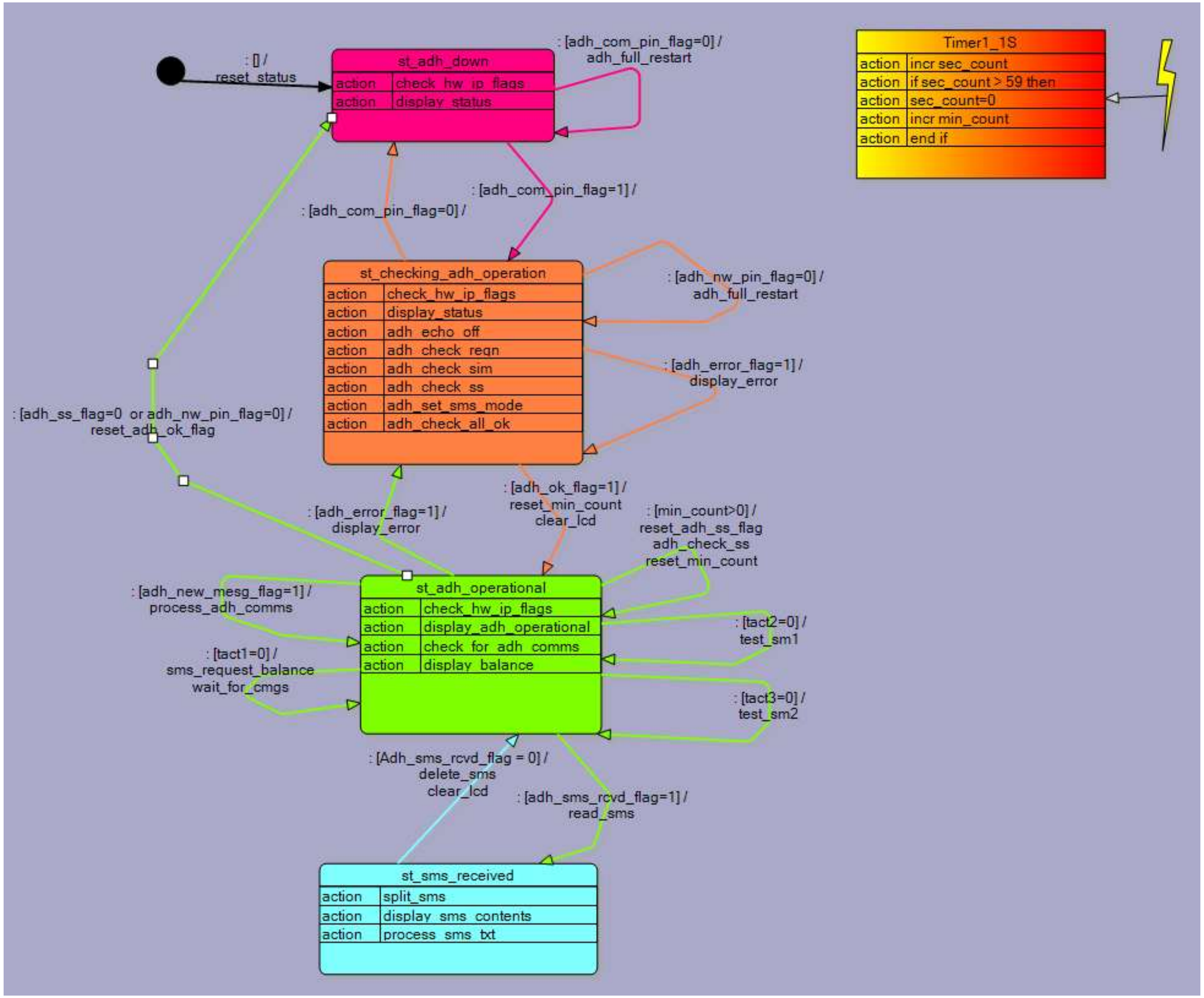
these two flags really aren't used except to keep track at the beginning that we have set the ADH up how we need it before going on to using it.

53.7 Second state machine



In the second state machine the detail of most of these flags and their settings are exposed (although at this time the credit amount hasn't been checked)

53.8 StateMachine 3



53.9 Sending an SMS text

The state st_adh_operational now has another event, when a switch is pressed the ADH requests a credit balance from the cellular provider.

```
'*************************
Sms_request_balance:
   Sms_txt = "bal"
   Sms_number = "777"
   Print "at+cmgs={034}" ; Sms_number ; "{034}"
   Waitms 50
   Print Sms_txt
   Waitms 50
   Print "{026}";                       'send Ctrl-Z suppress crlf
Return
```

The format for sending an SMS requires the number to be entered as AT+CMGS="02187654321"
And the ADH responds with a > (greater than symbol) awaiting the sms contents.

Of special note is how we embed the " within the string to be sent to the ADH. The print command uses the " as the start and finish of the string, but we also need it to be part of the string. This is where we use a special feature of Bascom with braces {} so in the line `Print "at+cmgs={034}" ; Sms_number ; "{034}"` the {034} means put the ascii character 34 within the string. So the string sent to the ADH will be <u>at+cmgs="777"</u>

After the sms text has been sent it needs a CRTL-Z or ascii character 26 to be sent to tell the ADH that all the text contents have been sent. You cannot send a CR-LF to the ADH as that just means a new line of text within the same message. To send a character 26 we again use the {} so the command is `Print "{026}";` Now take extra special note of the semicolon at the end of the line. When thi sis added at the end of a line it means do not send a CR-LF at the end of the string. If you foget this the ADH will not recognise the special character ascii 26.

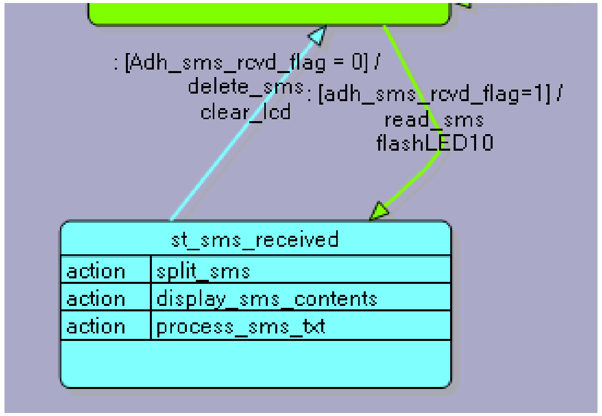
53.10 Receiving an SMS text

When an SMS text comes into the ADH it sends a message to the microcontroller Containg +CMTI: this is detected within the `Process_adh_comms:` subroutine.

To this subroutine is added another test and another flag is added to the system ‘`Adh_sms_rcvd_flag`’

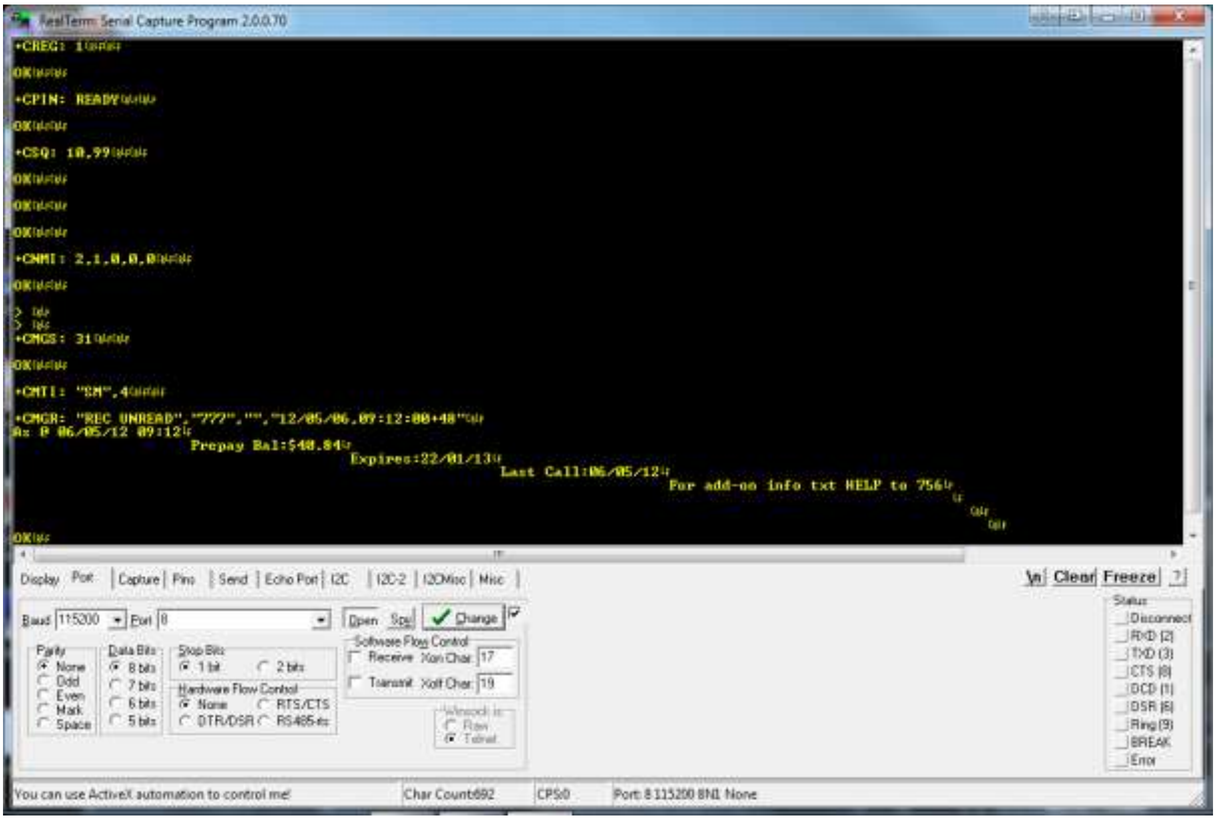
```
'received an sms
Temp_b = Instr(adh_message , "+CMTI:")          'see if within message
If Temp_b > 0 Then                              'yes it is
   Temp_b = Temp_b + 12                         'ignore first part
   Message_len = Len(adh_message)               'find end of message
   Incr Message_len                             'or we get wrong part
   Message_len = Message_len - Temp_b           'get the number of chars in sim mem address
   Temp_str = Right(adh_message , Message_len)
   Reset Adh_new_mesg_flag                      'got the new message
   Set Adh_sms_rcvd_flag                        'it is an sms so move to process it
End If
```

Once the flag is set the state will change to st_sms_received, the sms will be read from the ADH and the sms will be split into its different parts, displayed and any programmed actions are processed.

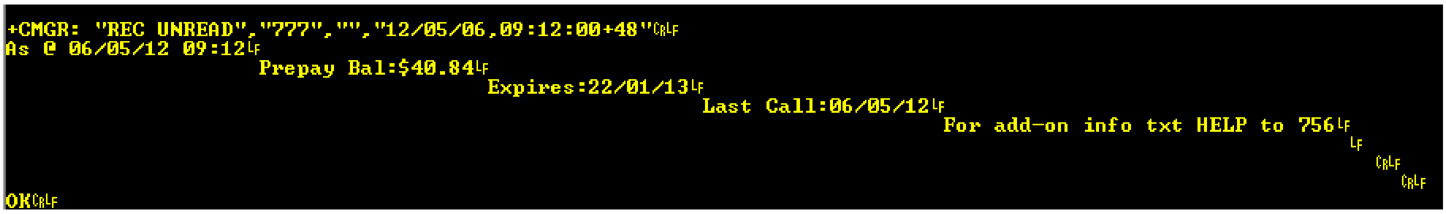


53.11 Splitting a large string (SMS message)

Using the program Realterm the adh to microcontroller communications were monitored. This program is useful in that it exposes all the characters sent e.g. CR LF and other non printing ascii codes.



Specifically this message is of interest



To be able to split this string into its component parts we must understand it in general terms; what that means is identify what is structure and what is contents.

+CMGR:	This is structure, it is the same for all sms messages
“REC UNREAD”	This is structure in that it starts with “ and finishes with “ But inside the “” the contents will differ. It might be “REC READ” if the message has been read once before, so we can use the “” possibly to help us.
,	The first comma will be before the senders number – very useful structure, note the structure has no spaces before or after the comma
“777”	Again some structure (the “”) and some contents (777) note no spaces
,	Another comma – definite structure indicator here
“”	This is some sort of sender identification, sometimes the text comes back and has “Customer Service” here
,	Another comma
“date,time”	Closely inspect the structure here “” to start and stop and comma between the date and time. Also no spaces. Not sure what the +48 means
Txt message	This has a LF between each line of the txt, a final LF then two CRLF then an OK +CRLF at the end

There is a SPLIT function in Bascom, however its not useful in this situation as the structure is not even between the different parts. We use the command INSTR to find the parts of the string.

In out string adh_message all the ascii characters less than 32 in the ascii table have been removed, this includes 13 and 10 (CR LF). So we cannot use those to help us with the messages structure. We will use the commas as they wil always be the in the same place.
The first part of the routine to fn the phone number is this:

- Find the first comma
  - INSTR will help here as it gives us the first position of a character in a string
  - `Temp_b = Instr(1 , Adh_message , "," )`
  - Looking at this message count all the characters from the + at the beginning until you get to the first comma. Include all spaces. This means the variable temp_b will have 19 in it
  - The first character of the phone number will always be another 2 places further on
  - `Temp_b = Temp_b + 2`
  - Temp_b now has 21 in it
- Find how many characters to retrieve from the string. If phone numbers were always the same length it would be easy just get X digits starting from the first digit. However they aren't so we need to get the position of the last digit then work out the difference between the two.
- Find the next comma, the last digit will be two back from this
  - `Temp_b2 = Instr(temp_b , Adh_message , ",")`
  - This means starting at the first digit of the phone number go forward until another comma is found
  - The variable Temp_b2 will have 25 in it
  - `Temp_b2 = Temp_b2 - 2`
  - The last character of the phone number will be 2 places before the comma (23)
- Find the difference between the two and add 1
  - `Temp_b2 = Temp_b2 - Temp_b`
  - Temp_bw will now be 23-21 = 2
  - `Incr Temp_b2`
  - Temp_b2 will now be 3, the right number of digits to get
- Get the telephone number
  - `Sms_number = Mid(adh_message , Temp_b , Temp_b2)`
  - The number will be located at temp_b and we need to get temp_b2 number of digits
  - So in this message we get the three characters starting from 21 to get 777
- Note that most phone numbers will have a + at the front as well (see a few pages back)
- Now we will ignore the next part of the message the "" or "Customer Service"
  - So starting from temp_b we find the position of the next comma
  - We reuse the variable temp_b because we wont need the old value anymore
  - Then we add 1 to get past the comma
- Now we will get the date
  - Find the position of the next comma
  - Then we add 2 to get to the first character of the date
  - The date willalways be 8 characters e.g. 12/05/23
  - And get the date into a temporary string temp_20
- Then we get change the date string around because its in year/month/day and in NZ we want day/month/year
  - This is done by copying the parts we want from temp_20 into sms_date.
  - RIGHT, MID and LEFT are useful commands for this.
- Time is then extracted – it is also 8 characters e.g. 23:12:45
- The sms contents are extracted using the LEN command to tell us the length of the whole message.

Here is the routine to split the SMS up into the parts we want

```
Split_sms:
   'identify diff parts of sms by using the commas between them

   'get first part of sms - the number
   Temp_b = Instr(1 , Adh_message , "," )         'find the first comma
   Temp_b = Temp_b + 2
   Temp_b2 = Instr(temp_b , Adh_message , ",")         'get nex comma
   Temp_b2 = Temp_b2 - 2
   Temp_b2 = Temp_b2 - Temp_b
   Incr Temp_b2
   Sms_number = Mid(adh_message , Temp_b , Temp_b2)

   'ignore second part of sms this will most likely be ""  , to do this
   'get the next comma after the beginning of the number
   'this will be the end of the number
   'note that when you txt bal to 777 this part contains "Customer Service"
   'increase this by 1 then get the next comma
   Temp_b = Instr(temp_b , Adh_message , ",")
   Incr Temp_b                               '

   'get 3rd part of sms this will be date (it is 8 characters long)
   Temp_b = Instr(temp_b , Adh_message , ",")         'get next comma
   Temp_b = Temp_b + 2                              'get the first char of date
   Temp_20 = Mid(adh_message , Temp_b , 8)
   'change to NZ date format
   Sms_date = Right(temp_20 , 2)
   Sms_date = Sms_date + "/"
   Sms_date = Sms_date + Mid(temp_20 , 4 , 2)
   Sms_date = Sms_date + "/"
   Sms_date = Sms_date + Left(temp_20 , 2)

   'get 4th part of sms this will be time (it is 8 characters long)
   Temp_b = Instr(temp_b , Adh_message , ",")
   Temp_b = Temp_b + 1                            'time is 1 on from the comma
   Sms_time = Mid(adh_message , Temp_b , 8)

   'get 5th part of sms this will be contents
   'it starts at 1 after the " after the end of the time
   ' and goes through to 2 characters before the end
   Temp_b = Instr(temp_b , Adh_message , "{034}")       'find the {
   Temp_b = Temp_b + 1
   Temp_b2 = Len(adh_message )            'get the full length
   Temp_b2 = Temp_b2 - Temp_b
   Decr Temp_b2                           'exclude OK on the end of the message
   Sms_txt = Mid(adh_message , Temp_b , Temp_b2)
Return
```

53.12 Converting strings to numbers

The credit balance is currently stored as a string, a bunch of ascii characters, it is not a number that we can add and subtract to.
So we need to convert it and place it in a numeric type variable. As it has a decimal pointit could be a single type, but that's a bit wasteful as a single is 4 bytes. A word willd o if we drop the decimal place and just store the creit as cents instead of dollars. $40.84 would then become 4084.

```
'what should happen when a specific sms is received
Process_sms_txt:
   'look for a balance
   Temp_b = Instr(sms_txt , "Bal:")
   If Temp_b > 0 Then                    'got an sms with Bal in it
      Temp_b = Temp_b + 5                'starting pt for amount
      Temp_b2 = Instr(temp_b , Sms_txt , ".")         'find decimal pt
      Temp_b2 = Temp_b2 + 2              'place of last digit of the amount
      Temp_b2 = Temp_b2 - Temp_b
      Incr Temp_b2                       'number of digits to get
      Credit_bal_str = Mid(sms_txt , Temp_b , Temp_b2)
      'convert string to number
      Temp_20 = Credit_bal_str
      Temp_b = Instr(Credit_bal_str, ".")
      Delchar Temp_20 , temp_b                'remove decimal pt
      Credit_bal_cents = Val(temp_20)    'convert to word var
      If Credit_bal_cents > 100 Then     'more than a dollar
         Set Credit_bal_flag
      Else
         Reset Credit_bal_flag
      End If
   End If

Return
```

Again the structure is identified.

- It will always be Bal:$ and then a number like 5.65 or 124.56 or 0.34
- So first we copy just the characters of the amunt e.g. 40.84 into the variable credit_bal_str
    - We use instr to find the location of the decimal point and know that the end of the credit value will be two characters after the decimal point.
    - We copy just the credit amount to another string
- Instr is used to finfd the decimal point in this new string
- Then we use the DELCHAR command we delete the character at that location
- The Bascom command VAL is used to convert the string (e.g. "4084") to a number and the variable credit_val_cents now has 4,084 in it.

53.13 Full Program listing for SM3

```
'------------------------------------------------------------------
' Title Block
' Author:   B.Collis
' Date:     2 may 2012
' File Name: ADH8066_SM3.bas
'------------------------------------------------------------------
' Program Description:
'
'------------------------------------------------------------------
' Compiler Directives (these tell Bascom things about our hardware)
$crystal = 7372800                  'the crystal we are using
$regfile = "m16def.dat"             'the micro we are using
$baud = 115200

'**********************************************
'Hardware Setups
' setup direction of all ports
Config Porta = Output               '
Config Portb = Output               '
Config Portc = Input                '
Config Portd = Input                '

'LCD
Config Lcdpin = Pin , Db4 = Portb.5 , Db5 = Portb.6 , Db6 = Portb.7 , Db7 = Portb.4 , E =
Portb.0 , Rs = Portb.1
Config Lcd = 20 * 4                 'configure lcd screen

'Serial
Config Serialin = Buffered , Size = 200 ', Bytematch = All        'int on rx of CR

'Configure internal interrupt hardware
'Interrupt Timer1_1S
'this code setup gets timer1 to interrupt the micro every 1 second
Config Timer1 = Timer , Prescale = 256
On Ovf1 Timer1_1s_isr
'Const T1_preload = 34287            '8MHz
Const T1_preload = 36736            '7.372800MHz
Enable Timer1
Enable Interrupts

'Hardware Aliases
Adh_command_pin Alias Pinc.5
Adh_network_pin Alias Pinc.3
Adh_on_key Alias Portc.4
Set Portc.5                         'pullup resistor
Set Portc.3                         'pullup resistor
Tact1 Alias Pind.2
Tact2 Alias Pind.3
Tact3 Alias Pind.4
Set Portd.2                         'pullup resistor
Set Portd.3                         'pullup resistor
Set Portd.4                         'pullup resistor

Config Portd.7 = Output
Blu_led Alias Portd.7
Config Portc.0 = Output
Yel_led Alias Portc.0
Config Porta.2 = Output
Piezo Alias Porta.2
Config Porta.4 = Output
Servo Alias Porta.4
```

```
'*********************************************
'Dimension Variables

'State Variables
Dim State As Byte
Const St_adh_down = 0
Const St_checking_adh_operation = 1
Const St_adh_operational = 2
Const St_sms_received = 3
State = St_adh_down                         'set the initial state


'Global variables
Dim Adh_status As Word
Dim Bytein As Byte                          'reading bytes from uart
Dim Temp_20 As String * 20
Dim Message_len As Byte
Dim I As Byte
Dim Temp_b As Byte
Dim Sig_str As String * 3
Dim Temp_b2 As Byte
Dim Temp_w As Word
Dim Temp_str As String * 3
Dim Adh_message As String * 200
Dim Sec_count As Byte
Dim Min_count As Byte
Dim Sim_sms_memory As String * 3
Dim Sms_number As String * 18
Dim Sms_date As String * 8
Dim Sms_time As String * 8
Dim Sms_txt As String * 100
Dim Credit_bal_str As String * 7
Dim Credit_bal_cents As Word                'in cents

'Initialise Variables
Credit_bal_str = "00.00"
```

```
'status bits
Adh_com_pin_flag Alias Adh_status.15    'command pin hardware connection ok
Adh_nw_pin_flag Alias Adh_status.14     'network pin hardware connection ok
Adh_alive_flag Alias Adh_status.13      'serial comms is working between micro and ADH
Adh_creg_flag Alias Adh_status.12       'ADH is registerd on cell nw
Adh_sim_flag Alias Adh_status.11        'sim card is functioning ok
Adh_ss_flag Alias Adh_status.10         'signal strength ok (not 99)
Adh_echo_flag Alias Adh_status.9        'we turned echo off
Adh_sms_mode_flag Alias Adh_status.8    'we set sms mode
Adh_ok_flag Alias Adh_status.7          'get yourself a smiley here!
Adh_sms_rcvd_flag Alias Adh_status.6    'an sms has been received from other cellphone
Adh_$_flag Alias Adh_status.5           'we have credit to send
Adh_new_mesg_flag Alias Adh_status.4    'new serial message from adh to micro
Adh_ok_rcvd_flag Alias Adh_status.3     'ADH all functioning ok
Credit_bal_flag Alias Adh_status.2      'credit over $1.00
Adh_sms_sending_flag Alias Adh_status.1 'sfter send while waiting for sent response
Adh_error_flag Alias Adh_status.0       'had an error returned from the adh
Adh_status = 0                          'reset all flags for initial start


'constants
Const Display_delay = 500
Const Adh_receive_delay = 20

Deflcdchar 7 , 28 , 16 , 28 , 7 , 5 , 7 , 4 , 4         'commpin
Deflcdchar 6 , 4 , 8 , 16 , 8 , 23 , 5 , 7 , 4         ' network pin
Deflcdchar 5 , 8 , 31 , 8 , 32 , 2 , 31 , 2 , 32         ' alive-comm ok
Deflcdchar 4 , 2 , 4 , 8 , 4 , 2 , 4 , 8 , 16         ' registered
Deflcdchar 3 , 14 , 21 , 21 , 21 , 17 , 27 , 21 , 14         ' sim
Deflcdchar 2 , 4 , 10 , 21 , 10 , 4 , 4 , 4 , 4         ' sig strength
Deflcdchar 1 , 31 , 32 , 10 , 10 , 32 , 4 , 10 , 17         ' not ok smiley
Deflcdchar 0 , 31 , 32 , 10 , 32 , 4 , 17 , 10 , 4         ' ok smiley



'**********************************************
'Program starts here
Cls                                     'clears LCD display
Cursor Off                              'no cursor
Lcd "ADH"

For I = 1 To 5
   Set Yel_led
   Set Blu_led
   Waitms 50
   Reset Yel_led
   Reset Blu_led
   Waitms 100
Next
```

```
'*********************************************
'*************    State Machine    *************
'*********************************************
Gosub Reset_status
Do

   '*************** state st_adh_down ***************
   While State = St_adh_down
      Locate 4 , 1
      Lcd "st=" ; State
      Gosub Check_hw_ip_flags
      Gosub Display_status
      If Adh_com_pin_flag = 0 Then Gosub Adh_full_restart
      If Adh_com_pin_flag = 1 Then State = St_checking_adh_operation
   Wend

   '*************** state st_checking_adh_operation ***************
   While State = St_checking_adh_operation
      Locate 4 , 1
      Lcd "st=" ; State
      Gosub Check_hw_ip_flags
      Gosub Display_status
      Gosub Adh_echo_off
      Gosub Adh_check_regn
      Gosub Adh_check_sim
      Gosub Adh_check_ss
      Gosub Adh_set_sms_mode
      Gosub Adh_check_all_ok
      If Adh_nw_pin_flag = 0 Then Gosub Adh_full_restart
      If Adh_ok_flag = 1 Then
         State = St_adh_operational
         Gosub Reset_min_count
         Gosub Clear_lcd
      End If
      If Adh_com_pin_flag = 0 Then State = St_adh_down
   Wend
```

```
'*************** state st_adh_operational ***************
   While State = St_adh_operational
      Locate 4 , 1
      Lcd "st=" ; State
      Gosub Check_hw_ip_flags
      Gosub Display_adh_operational
      Gosub Check_for_adh_comms
      Gosub Display_balance
      If Min_count > 0 Then
         Gosub Reset_adh_ss_flag
         Gosub Adh_check_ss
         Gosub Reset_min_count
      End If
      If Adh_ss_flag = 0 Or Adh_nw_pin_flag = 0 Then
         State = St_adh_down
         Gosub Reset_adh_ok_flag
      End If
      If Adh_new_mesg_flag = 1 Then Gosub Process_adh_comms
      If Tact1 = 0 Then
      Locate 2 , 1
      Lcd "sending..."
         Gosub Sms_request_balance
         Gosub Wait_for_cmgs
      End If
      If Tact2 = 0 Then Gosub Test_sm1
      If Tact3 = 0 Then Gosub Test_sm2
      If Adh_sms_rcvd_flag = 1 Then
         State = St_sms_received
         Gosub Read_sms
         Gosub Flashled10
      End If
   Wend

   '*************** state st_sms_received ***************
   While State = St_sms_received
      Locate 4 , 1
      Lcd "st=" ; State
      Gosub Split_sms
      Gosub Display_sms_contents
      Gosub Process_sms_txt
      If Adh_sms_rcvd_flag = 0 Then
         State = St_adh_operational
         Gosub Delete_sms
         Gosub Clear_lcd
      End If
   Wend
Loop
End                                        'end program
```

```
'**********************************************
'*************    Subroutines    *************
'**********************************************

'*************************
Check_for_adh_comms:
   I = Ischarwaiting()                      'see if buf has something (when I is 0 then no,
when I is 1 then yes)
   While I = 1                              'while chars are there copy to the string
      Bytein = Inkey()                      'get one char
      If Bytein > 31 Then                   'if printable char
         Adh_message = Adh_message + Chr(bytein)       'add
      End If
      I = Ischarwaiting()                   'see if any more
      Set Adh_new_mesg_flag                 'flag that our string has something
   Wend                                     'if no data exit the loop
Return


'*************************
Check_hw_ip_flags:
   Adh_com_pin_flag = Not Adh_command_pin
   Adh_nw_pin_flag = Not Adh_network_pin
Return

'*************************
Reset_status:
   Adh_status = 0
Return

'*************************
Reset_adh_ss_flag:
   Reset Adh_ss_flag
Return
'*************************
Clear_lcd:
   Cls
Return

'*************************
Set_adh_ok_flag:
   Set Adh_ok_flag
Return

'*************************
Reset_adh_ok_flag:
   Reset Adh_ok_flag
Return


'*************************
Display_balance:
      Locate 1 , 12
      Lcd "$" ; Credit_bal_str ; " "
Return
```

```
'*************************
Adh_full_restart:
   Adh_status = 0
   For I = 5 To 1 Step -1
      Locate 3 , 1
      Lcd Spc(20)
      Locate 3 , 1
      Lcd "Power up ADH in " ; I
      Gosub Display_status
      Wait 1
   Next

   Set Adh_on_key
   For I = 3 To 1 Step -1
      Locate 3 , 1
      Lcd Spc(20)
      Locate 3 , 1
      Lcd "ADH_ON_KEY low:" ; I
      Gosub Display_status
      Wait 1
   Next

   Reset Adh_on_key
   'wait for upto 20 secs
   ' if hw starts ok then exit counting
   I = 20
   Do
         Locate 3 , 1
         Lcd Spc(20)
         Locate 3 , 1
         Lcd "waiting for ADH:" ; I
         Decr I
         Gosub Check_hw_ip_flags
         Gosub Display_status
         If Adh_nw_pin_flag = 1 And Adh_com_pin_flag = 1 Then
            Locate 3 , 1
            Lcd Spc(20)
            Locate 3 , 1
            Lcd "ADH OK"
            Exit Do
         End If
         Wait 1
   Loop Until I = 0

   'this 5 secs was found to be useful if the sim couldnt register
   'as the ADH comes up then drops out after a few secs
   If Adh_nw_pin_flag = 1 And Adh_com_pin_flag = 1 Then
      I = 5
      Do
         Gosub Display_status
         Locate 3 , 1
         Lcd Spc(20)
         Locate 3 , 1
         Lcd "ADH OK:" ; I
         Decr I
         Wait 1
      Loop Until I = 0
   End If
Return
```

```
'***************************
'just get the bit we want
Split_sms:
   'identify parts of sms by using the commas between them
    Sms_number = "????????"
    Sms_date = "????????"
    Sms_time = "????????"
    Sms_txt = "no message!"
   'get first part of sms - the number
   Temp_b = Instr(1 , Adh_message , "," )        'find the first comma
   If Temp_b = 0 Then Return                  'no comma means no message so return

   Temp_b = Temp_b + 2
   Temp_b2 = Instr(temp_b , Adh_message , ",")        'get next comma
   Temp_b2 = Temp_b2 - 2
   Temp_b2 = Temp_b2 - Temp_b
   Incr Temp_b2
   Sms_number = Mid(adh_message , Temp_b , Temp_b2)

   'ignore second part of sms this will most likely be ""  , to do this
   'get the next comma after the beginning of the number
   'this will be the end of the number
   'note that when you txt bal to 777 this part contains "Customer Service"
   'increase this by 1 then get the next comma
   Temp_b = Instr(temp_b , Adh_message , ",")
   Incr Temp_b                                '

   'get 3rd part of sms this will be date (it is 8 characters long)
   Temp_b = Instr(temp_b , Adh_message , ",")        'get next comma
   Temp_b = Temp_b + 2                        'get the first char of date
   Temp_20 = Mid(adh_message , Temp_b , 8)
   'change to NZ date format
   Sms_date = Right(temp_20 , 2)
   Sms_date = Sms_date + "/"
   Sms_date = Sms_date + Mid(temp_20 , 4 , 2)
   Sms_date = Sms_date + "/"
   Sms_date = Sms_date + Left(temp_20 , 2)

   'get 4th part of sms this will be time (it is 8 characters long)
   Temp_b = Instr(temp_b , Adh_message , ",")
   Temp_b = Temp_b + 1                        'time is 1 on from the comma
   Sms_time = Mid(adh_message , Temp_b , 8)

   'get 5th part of sms this will be contents
   'it starts at 1 after the " after the end of the time
   ' and goes through to 2 characters before the end
   Temp_b = Instr(temp_b , Adh_message , "{034}")        'find the {
   Temp_b = Temp_b + 1
   Temp_b2 = Len(adh_message )                'get the full length
   Temp_b2 = Temp_b2 - Temp_b
   Decr Temp_b2                               'exclude OK on the end of the message
   Sms_txt = Mid(adh_message , Temp_b , Temp_b2)
Return
```

```
'**************************
'put the first 40 characters of any communications from the adh on line 2&3
Display_adh_comms:
   If Adh_new_mesg_flag = 1 Then         'new message
      Locate 2 , 1
      Lcd Spc(20)
      Locate 3 , 1
      Lcd Spc(20)
      Temp_20 = Left(adh_message , 20)
      Locate 2 , 1
      Lcd Temp_20
      Temp_20 = Mid(adh_message , 21 , 20)
      Locate 3 , 1
      Lcd Temp_20
      Waitms Display_delay
   End If
Return

'**************************
Display_sms_contents:
   Cls
   Lcd "from" ; Sms_number
   Locate 2 , 1
   Lcd Sms_time
   Lcd " - " ; Sms_date
   Temp_20 = Left(sms_txt , 20)
   Locate 3 , 1
   Lcd Temp_20
   Temp_20 = Mid(sms_txt , 21 , 20)
   Locate 4 , 1
   Lcd Temp_20
   For I = 1 To 10
      Set Blu_led
      Waitms 250
      Reset Blu_led
      Waitms 750
   Next

Return


'**************************
Adh_echo_off:
   If Adh_echo_flag = 1 Then Return      'already ok then dont do it again
   Adh_message = ""                      'clear any previously received data
   Adh_new_mesg_flag = 0                 'so prog will wait for new message
   Reset Adh_ok_rcvd_flag                'we will wait for an OK
   Print "ATE0"                          'should respond with OK
   I = 0
   Do
      Incr I
      Waitms 100                         'give ADH a while to respond
      Gosub Check_for_adh_comms
      If I > 10 Then Return              'took too long to respond
   Loop Until Adh_new_mesg_flag = 1      'wait for answer
   Gosub Process_adh_comms               'see what came from adh
   If Adh_ok_rcvd_flag = 1 Then
      Set Adh_echo_flag                  'got an OK
      Reset Adh_ok_rcvd_flag
   End If
   Gosub Display_status                  'show status so far
   Waitms Display_delay                  'for debug purposes wait a while
Return
```

```
'************************
'when the user presses tact2 it creates a false sms received message
'and sets the adh_new_mesg_flag so st_adh_operational will think an sms has arrived
Test_sm1:
   Adh_message = "+CMTI: {034}SM{034},2"
   Locate 2 , 1
   Lcd "test sms receive:"
   Locate 3 , 1
   Lcd Adh_message
   Wait 2
   Set Adh_new_mesg_flag
Return

Test_sm2:
   Adh_message = "+CMTI: {034}SM{034},3"
   Locate 2 , 1
   Lcd "test sms receive:"
   Locate 3 , 1
   Lcd Adh_message
   Wait 2
   Locate 2 , 1
   Lcd Spc(20)
   Locate 3 , 1
   Lcd Spc(20)
   Set Adh_new_mesg_flag
Return

'************************
Flashled10:
   For I = 1 To 10
      Set Yel_led : Waitms 100 : Reset Yel_led : Waitms 100
   Next
Return
```

```
'*************************
Adh_check_regn:
   If Adh_creg_flag = 1 Then Return       'already ok then dont do it again
   Adh_message = ""                        'clear any received data
   Adh_new_mesg_flag = 0
   Print "AT+CREG?"                        'send the request
   I = 0
   Do
      Incr I
      Waitms 100
      Gosub Check_for_adh_comms
      If I > 10 Then Return                'took too long to respond
   Loop Until Adh_new_mesg_flag = 1        'wait for answer
   Gosub Process_adh_comms
   Gosub Display_status
   Waitms Display_delay
Return


'*************************
Adh_check_sim:
   If Adh_sim_flag = 1 Then Return        'already ok then dont do it again
   Adh_message = ""                        'clear any received data
   Adh_new_mesg_flag = 0
   Print "AT+CPIN?"
   I = 0
   Do
      Incr I
      Waitms 100
      Gosub Check_for_adh_comms
      If I > 10 Then Return
   Loop Until Adh_new_mesg_flag = 1        'wait for answer
   Gosub Process_adh_comms
   Gosub Display_status
   Waitms Display_delay
Return

'*************************
Adh_check_ss:
   If Adh_ss_flag = 1 Then Return         'already ok then dont do it again
   Adh_message = ""                        'clear any received data
   Adh_new_mesg_flag = 0
   Print "AT+CSQ"
   I = 0
   Do
      Incr I
      Waitms 100
      Gosub Check_for_adh_comms
      If I > 10 Then Return
   Loop Until Adh_new_mesg_flag = 1        'wait for answer
   Gosub Process_adh_comms                 'check what we received
   'Gosub Display_status
   Waitms Display_delay
Return
```

```
'*************************
Adh_set_sms_mode:
   If Adh_sms_mode_flag = 1 Then Return 'already ok then dont do it again
   Adh_message = ""                     'clear any received data
   Adh_new_mesg_flag = 0
   Print "AT+CMGF=1"                    'text mode
   Waitms 100
   Print "AT+CNMI=2,1,0,0,0"            'mode tells us when sms arrives
   Waitms 100
   Adh_message = ""                     'ignore ok responses
   Adh_new_mesg_flag = 0
   Print "AT+CNMI?"                     'get mode check
   I = 0
   Do
      Incr I
      Waitms 100
      Gosub Check_for_adh_comms
      If I > 10 Then Return
   Loop Until Adh_new_mesg_flag = 1     'wait for answer
   Gosub Process_adh_comms              'check what we received
   Gosub Display_status
   Waitms Display_delay
   Adh_message = ""                     'ignore ok responses
   Adh_new_mesg_flag = 0
Return


'*************************
'the message from the adh contains time, date, number, and the message
Read_sms:
   Adh_message = ""
   Temp_20 = "at+cmgr=" + Sim_sms_memory
   Print Temp_20
   I = 0
   Do
      Incr I
      Waitms 100                        'give ADH a while to respond
      Gosub Check_for_adh_comms
      If I > 10 Then Return             'took too long to respond
   Loop Until Adh_new_mesg_flag = 1     'wait for answer
   Reset Adh_sms_rcvd_flag
   'check for error here ?
Return
```

```
'*************************
'what should happen when a specific sms is received
Process_sms_txt:
   'look for a balance
   Temp_b = Instr(sms_txt , "Bal:")
   If Temp_b > 0 Then                    'got an sms with Bal in it
      Temp_b = Temp_b + 5                'starting pt for amount
      Temp_b2 = Instr(temp_b , Sms_txt , ".")        'find decimal pt
      Temp_b2 = Temp_b2 + 2              'place of last digit of the amount
      Temp_b2 = Temp_b2 - Temp_b
      Incr Temp_b2                       'number of digits to get
      Credit_bal_str = Mid(sms_txt , Temp_b , Temp_b2)
      'convert string to number
      Temp_20 = Credit_bal_str
      Temp_b = Instr(credit_bal_str , ".")
      Delchar Temp_20 , Temp_b           'remove decimal pt
      Credit_bal_cents = Val(temp_20)    'convert to word var
      If Credit_bal_cents > 100 Then     'more than a dollar
         Set Credit_bal_flag
      Else
         Reset Credit_bal_flag
      End If
   End If

Return

'*************************
Sms_request_balance:
   Sms_txt = "bal"
   Sms_number = "777"
   Print "at+cmgs={034}" ; Sms_number ; "{034}"
   Waitms 50
   Print Sms_txt
   Waitms 50
   Print "{026}";                        'send Ctrl-Z suppress crlf
   Waitms 50
Return

'*************************
'check the sms was sent
Wait_for_cmgs:
   Reset Adh_error_flag                  'no error
   I = 0
   Do
      Incr I
      Gosub Check_for_adh_comms          'read anything from adh
      Temp_b = Instr(adh_message , "+CMGS:")        'see if we got cmgs
      Waitms 100
      If I > 100 Then                    'no response in time
         Set Adh_error_flag
         Return
      End If
   Loop Until Temp_b > 0                 'once we have got cmgs it will exit
Return


'*************************
Delete_sms:
   Print "at+cmgd=" ; Sim_sms_memory
   Cls
   Lcd "deleting message " ; Sim_sms_memory
   Wait 5
Return
```

```
'*************************
'check what was received from the adh
Process_adh_comms:
   If Instr(adh_message , "OK") > 0 Then
      Set Adh_ok_rcvd_flag
   End If

   If Instr(adh_message , "+CREG: 0") > 0 Then
      Reset Adh_creg_flag
   End If

   If Instr(adh_message , "+CREG: 1") > 0 Then
      Set Adh_creg_flag
      Set Adh_alive_flag                'make sure it is set
   End If

   If Instr(adh_message , "+CREG: 3") > 0 Then
      Reset Adh_creg_flag
   End If

   If Instr(adh_message , "+CMGS") > 0 Then
      Set Adh_sim_flag
   End If

   If Instr(adh_message , "+CPIN: READY") > 0 Then
      Set Adh_sim_flag
   End If

   'find the position of the signal strength in the string
   Temp_b = Instr(adh_message , "CSQ:")
   If Temp_b > 0 Then
      Temp_b = Temp_b + 5
      Sig_str = Mid(adh_message , Temp_b , 2)
      If Instr(sig_str , ",") > 0 Then   'SS is single digit
         Sig_str = Mid(adh_message , Temp_b , 1)
         Sig_str = "0" + Sig_str
      End If
      If Sig_str = "99" Or Sig_str = "00" Then
         Adh_ss_flag = 0                'no sig
      Else
         Adh_ss_flag = 1                'ok sig
      End If
   End If

   'check that sms mode is ok
   Temp_b = Instr(adh_message , "+CNMI:")
   If Temp_b > 0 Then
      Temp_b = Temp_b + 7               'get indx of part we want
      Temp_str = Mid(adh_message , Temp_b , 3)
      If Temp_str = "2,1" Then
         Set Adh_sms_mode_flag
      Else
         Reset Adh_sms_mode_flag
      End If
   End If

   'received an sms
   Temp_b = Instr(adh_message , "+CMTI:")        'see if within message
   If Temp_b > 0 Then                   'yes it is
      Temp_b = Temp_b + 12              'ignore first part
      Message_len = Len(adh_message)    'find end of message
      Incr Message_len                  'or we get wrong part
      Message_len = Message_len - Temp_b       'get the number of chars in sim mem
address
```

```
      'Sim_sms_memory = Mid(adh_message , Temp_b , Message_len)        'get the address in
sim memory
      Sim_sms_memory = Right(adh_message , Message_len)
      Reset Adh_new_mesg_flag           'got the new message
      Set Adh_sms_rcvd_flag             'it is an sms so move to process it
   End If
Return


'**************************
'in operational state
'puts simple single status on LCD
Display_adh_operational:
   Locate 1 , 1
   Lcd "ADH cell"
   Locate 1 , 20
   If Adh_ok_flag = 1 Then
      Lcd Chr(0)                        'good
   Else
      Lcd Chr(1)                        'bad
   End If
   'show timer
   Locate 4 , 8
   If Adh_ok_flag = 1 Then
      If Min_count < 10 Then Lcd "0"
      Lcd Min_count ; ":"
      If Sec_count < 10 Then Lcd "0"
      Lcd Sec_count
   Else
      Lcd "--:--"
   End If
   'show signal strength
      Locate 4 , 16
      Lcd "SS=" ; Sig_str
   'show state
   Locate 4 , 1
   Lcd "st=" ; State
Return


'**************************
Adh_check_all_ok:
   Temp_w = Adh_status And &B1111111100000000        'get just the first 8 bits
   If Temp_w = &B1111111100000000 Then
      Gosub Set_adh_ok_flag
   End If
Return
```

```
'**************************
'in down and checking states
'puts the full status on the LCD
Display_status:
   Locate 1 , 5
   If Adh_com_pin_flag = 1 Then
      Lcd "C"
   Else
      Lcd "-"
   End If
   If Adh_nw_pin_flag = 1 Then
      Lcd "N"
   Else
      Lcd "-"
   End If
   If Adh_alive_flag = 1 Then
      Lcd Chr(5)
   Else
      Lcd "-"
   End If
   If Adh_creg_flag = 1 Then
      Lcd Chr(4)
   Else
      Lcd "-"
   End If
   If Adh_sim_flag = 1 Then
      Lcd Chr(3)
   Else
      Lcd "-"
   End If
   If Adh_ss_flag = 1 Then
      Lcd "S"
   Else
      Lcd "-"
   End If
   If Adh_echo_flag = 1 Then
      Lcd "E"
   Else
      Lcd "-"
   End If
   If Adh_sms_mode_flag = 1 Then
      Lcd "M"
   Else
      Lcd "-"
   End If

   If Adh_$_flag = 1 Then
      Lcd "$"
   Else
      Lcd "-"
   End If
   'display overall status
   Locate 1 , 20
   If Adh_ok_flag = 1 Then
      Lcd Chr(0)
   Else
      Lcd Chr(1)
   End If
   'show state
   Locate 4 , 1
   Lcd "st=" ; State
Return
```

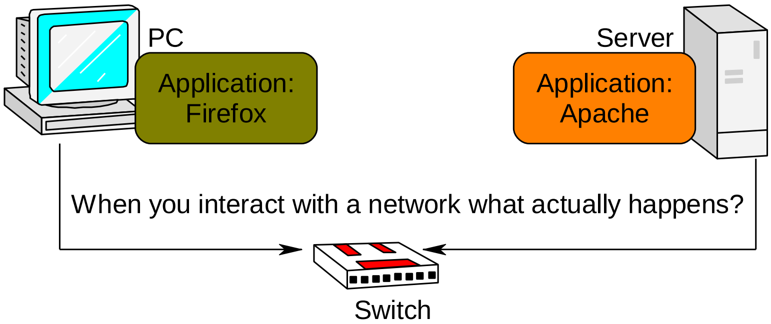
```
'**********************************************
'*********    Interrupt Routines     **********
'**********************************************
'Timer1 interrupt service routine - program comes here automatically every 1Second
Timer1_1s_isr:
   Timer1 = T1_preload
   Incr Sec_count
   If Sec_count > 59 Then
      Sec_count = 0
   Incr Min_count
   End If
Return

Reset_min_count:
   Min_count = 0
Return
```

54 Data transmission across the internet

(its all about understanding layers!!!)

Ogres are like onions.
-They stink?
Yes. No!
-Oh, they make you cry.
No!. . .Layers. Onions have layers. Ogres have layers. Onions have layers. You get it? We both have layers.
-Oh, you both have layers. Oh. You know, not everybody likes onions.
Shrek, 2001

Here is a very simple network, 2 computers communicating in our classroom, one is a client PC and the other is the local web server.



A switch is a device that connects together two computers that both talk Ethernet.

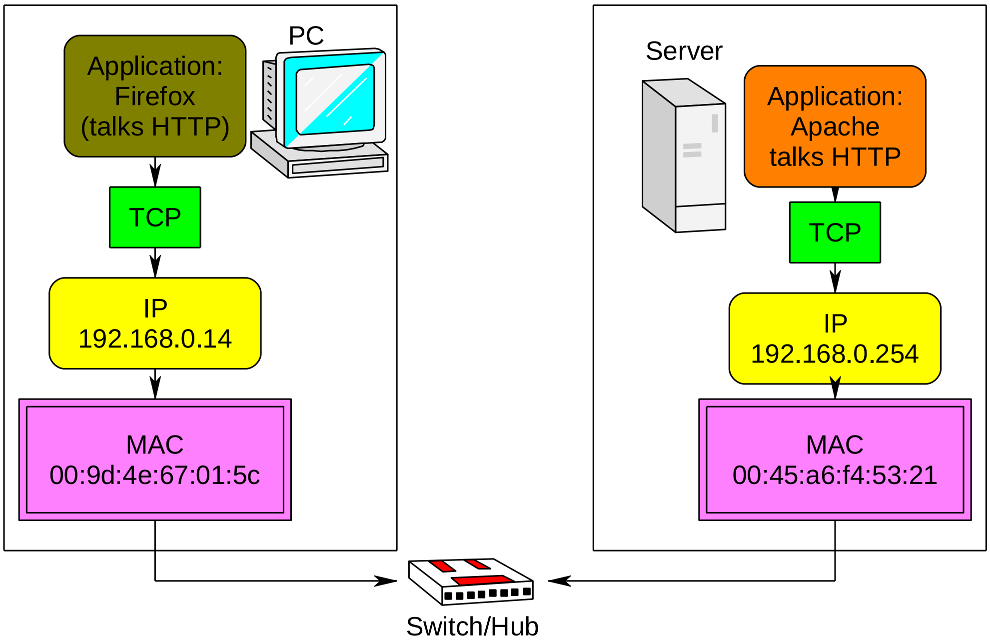
Important point: Data does not go directly between the applications on the two computers! Firefox doesn't talk directly to Apache, there are a number of layers that data is converted through that make the system flexible for all the applications that run on computers, e.g. accessing mail, file sharing, getting the time, streaming music videos, playing games etc.

Because some of these applications are so incredibly different a simple conversion to one common layer is not enough so there are a numbers of layers in use in the PC.

The whole set of internet protocols that allow communication is called the Internet Protocol Suite.

Important point: **When two applications do talk to each other they have to talk the same language**; e.g. Apache and Firefox (also Chrome and IE) talk in HTTP – hyper text transport protocol.

The HTTP is converted to the TCP (transmission control protocol) layer by programs in the operating system which are part of the TCP/IP stack. TCP is then converted to IP (internet protocol).

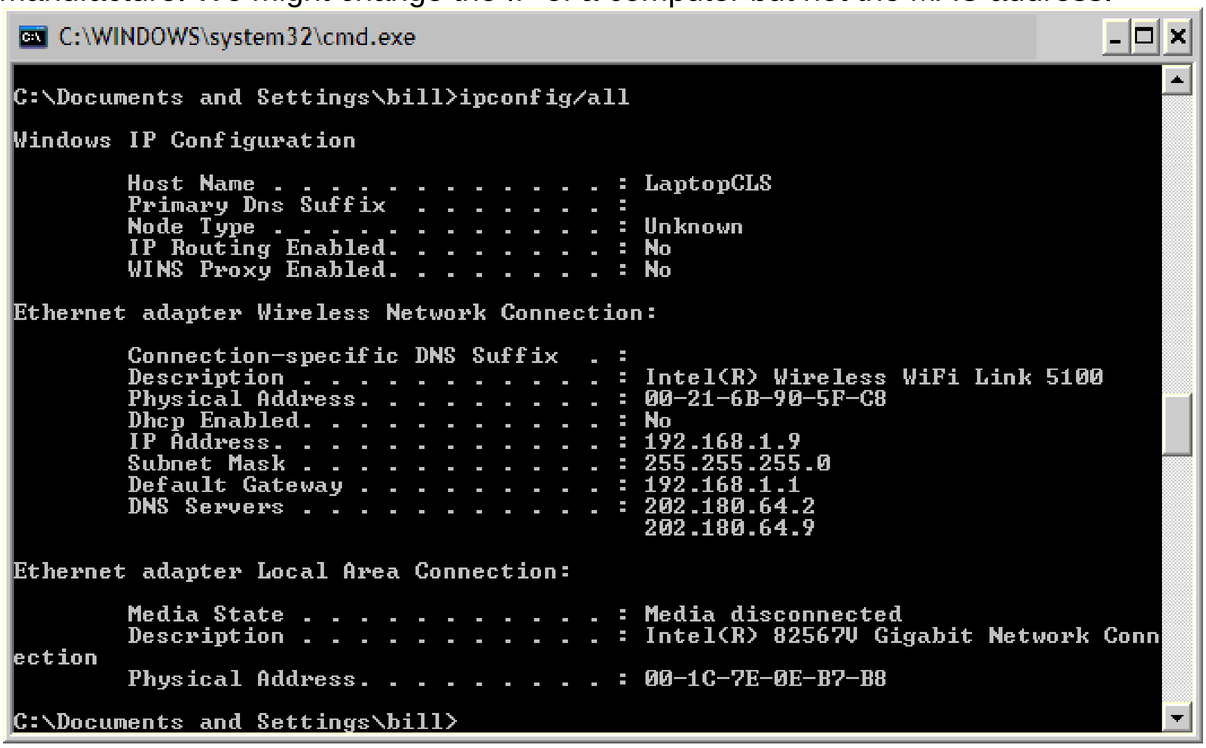


54.1 IP address

Our web client (Firefox, IE, Chrome) asks the OS (operating system) of its PC (with IP address 192.168.0.14) to get a web page from the server (with IP address 192.168.0.254). **IP addresses** must be unique for each computer on the same network. Humans often allocate and use IP addresses when they refer to computers on a network. Computers, however don't know each other by IP they use the...

54.2 MAC (physical) address

Computers know each other by attaching their IP address to the MAC (media access layer) of your network card. You can check out the ip and mac addresses of your network card on a PC by selecting START and RUN and typing in CMD and then typing into the command window - **ipconfig/all**. MAC addresses are unique for every network card ever made and are assigned by the manufacture. We might change the IP of a computer but not the MAC address.



```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\bill>ipconfig/all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : LaptopCLS
        Primary Dns Suffix  . . . . . . . :
        Node Type . . . . . . . . . . . . : Unknown
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Wireless Network Connection:

        Connection-specific DNS Suffix  . :
        Description . . . . . . . . . . . : Intel(R) Wireless WiFi Link 5100
        Physical Address. . . . . . . . . : 00-21-6B-90-5F-C8
        Dhcp Enabled. . . . . . . . . . . : No
        IP Address. . . . . . . . . . . . : 192.168.1.9
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 192.168.1.1
        DNS Servers . . . . . . . . . . . : 202.180.64.2
                                            202.180.64.9

Ethernet adapter Local Area Connection:

        Media State . . . . . . . . . . . : Media disconnected
        Description . . . . . . . . . . . : Intel(R) 82567V Gigabit Network Conn
ection
        Physical Address. . . . . . . . . : 00-1C-7E-0E-B7-B8

C:\Documents and Settings\bill>
```

The PC keeps track of the IP and MAC address of computers around it in a table using ARP (address resolution protocol). Type **ARP –a** into the command window to check out what other devices (PCs/routers....) that your PC can see.

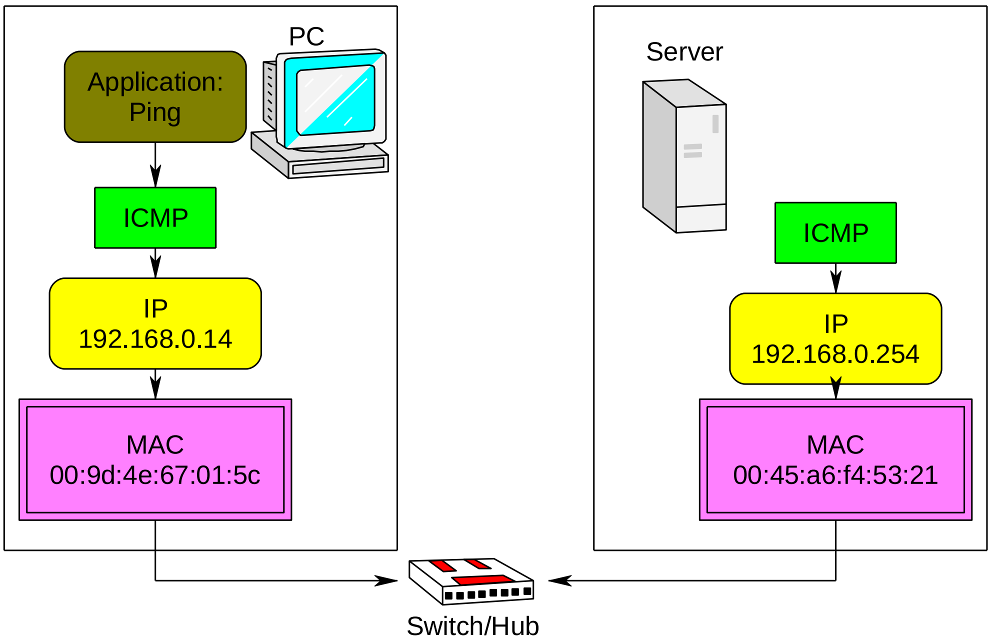
The PC shown in the previous window has two Ethernet adapters, one is wireless and one is wired. Only one is connected and has an IP, the other is not used at all so has no IP assigned to it. Both however have MAC addresses as the MAC address is a permanent ID for the hardware, whereas IPs can be changed.

Ethernet is the name of the protocol for moving data between PCs on the same hub or switch and specifies such things as what wires do what and what voltages are present. Having multiple layers to communications means that applications can be simpler because they don't have to know everything about the layers below such as about voltage and wires or IP just http to TCP.

54.3 Subnet mask

When you setup the IP of a network card, you set the subnet mask as well. If you want computers on the same switch to see each other the subnet mask must be the same, e.g. 255.255.255.0 as in the previous window.

54.4 Ping



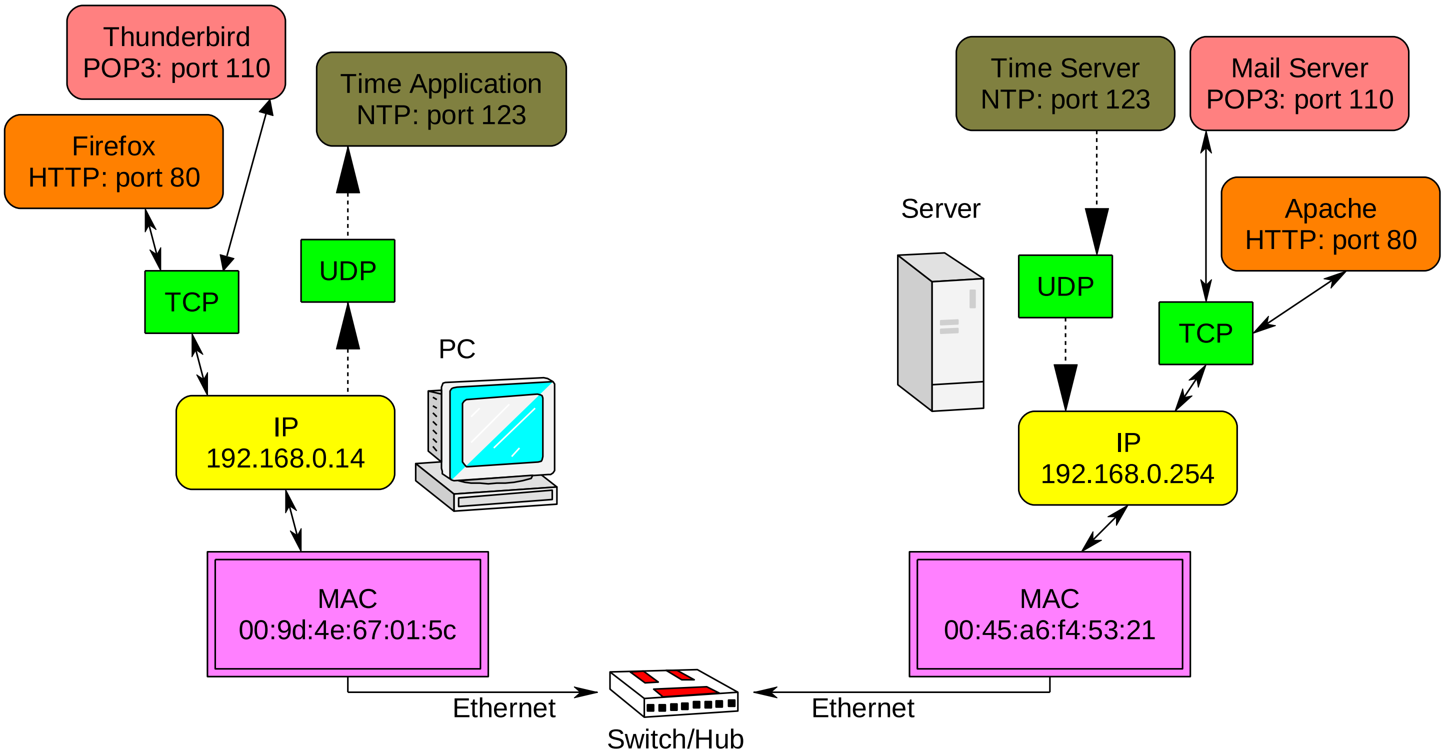
Type in ping and the ip of the computer that you want to check communications with.

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings\bill>ping 192.168.1.2

Pinging 192.168.1.2 with 32 bytes of data:

Reply from 192.168.1.2: bytes=32 time=1ms TTL=64
Reply from 192.168.1.2: bytes=32 time=1ms TTL=64
Reply from 192.168.1.2: bytes=32 time=1ms TTL=64
Reply from 192.168.1.2: bytes=32 time=1ms TTL=64

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 1ms, Average = 1ms

C:\Documents and Settings\bill>
```

Ping is an application on your computer, it communicates with another computer using ICMP (internet control message protocol) which is used by operating systems to manage messaging errors between computers. In the picture of the layers you can see that ping doesn't talk using TCP it uses ICMP. On the other computer there is no application above ICMP it answers pings itself.

54.5 Ports

There is always more than one application on a PC wanting network access; we have email, web browsers, time synchronisation and many others. Attached to each IP are 65,536 different ports that can be used, many of them are dedicated to certain applications, here are three.



Some of the applications require two way data transmission , like web browsing and email. Some like Time and Shoutcast and VoIP are really only one way. Two way applications run on TCP, one way applications can run using UDP.

54.6 Packets

Having two different protocols, TCP and UDP, on the same layer is useful and necessary.

Sometimes it doesn't matter if a bit of data gets lost and sometimes it does. When a web page is sent, TCP breaks it up into chunks called packets and attaches a sequence number to each packet, if a packet gets lost across the network then TCP on the receiving computer responds with a message to resend the packet that was lost.

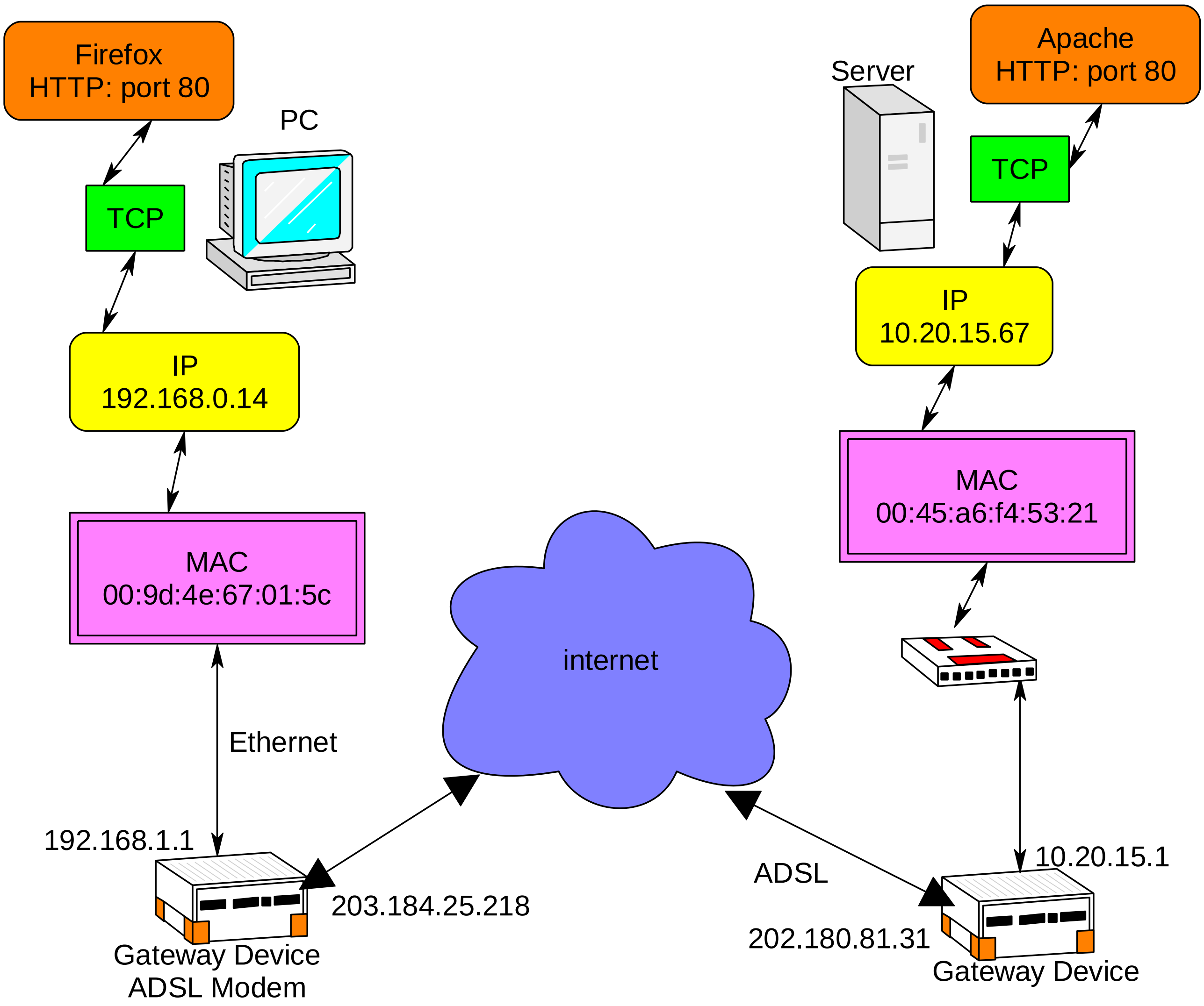
However if you are listening to the radio over the network and some data goes missing you don't want it sent again so it goes via UDP (user datagram protocol), which means that we don't resend lost packets, just ignore them.

54.7 Gateway

There are different ways of connecting to the internet a popular way is via broadband using ADSL.

A switch is ok for connecting computers via Ethernet however Ethernet only works for short distances, so other technologies are necessary to transport data long distances over the internet.

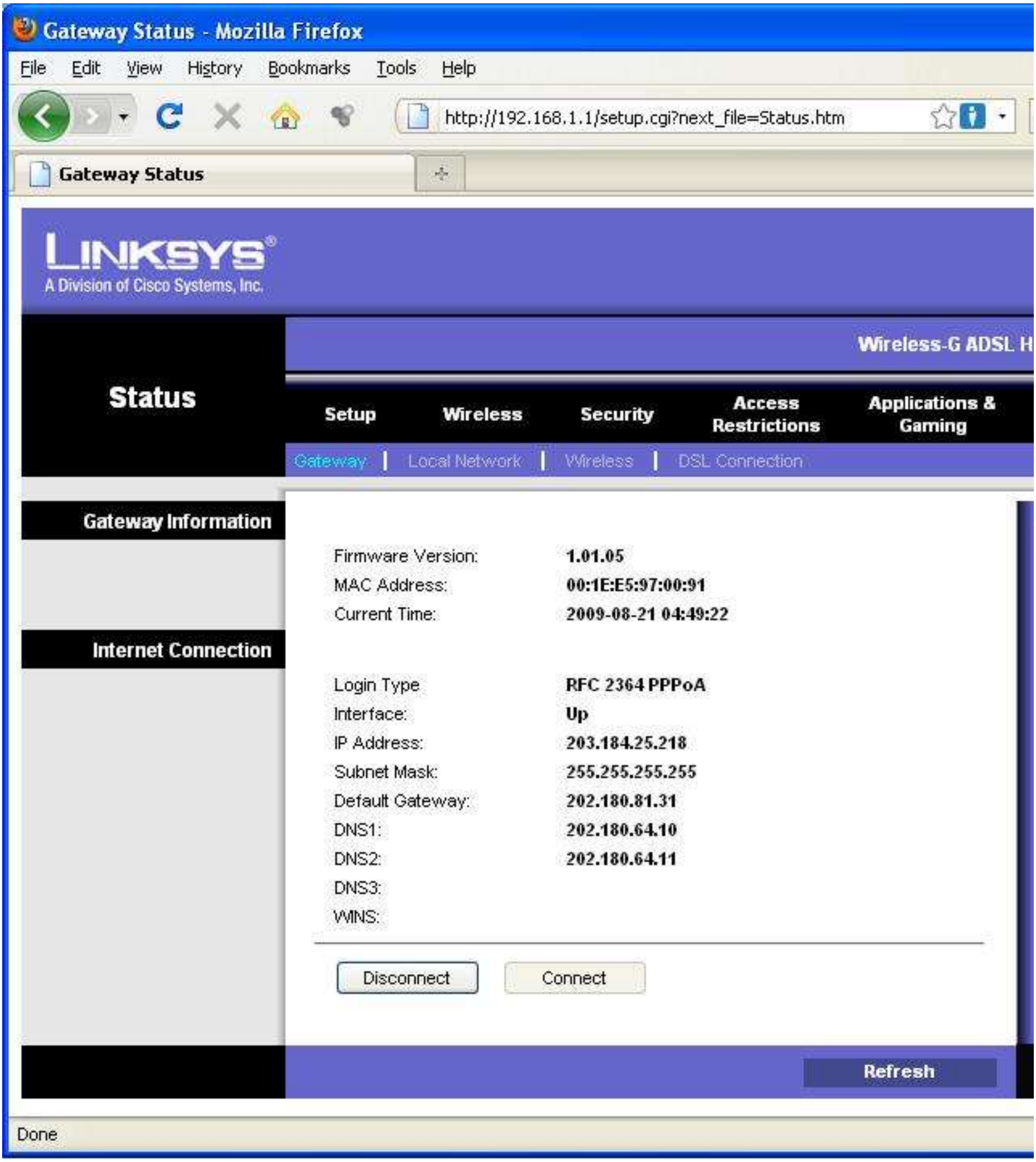
A gateway translates one type of data protocol to another e.g. Ethernet to ADSL.



A gateway has two IP addresses one for the LAN (local area network) and one for the WAN (wide area network). If you are setting up a small network at home then you don't worry about your gateway IP on the internet, your ISP (internet service provider) has hardware that gives you one automatically when your modem logs in. You just set up your gateway address on each computer on your LAN.

When you open a web browser you don't have to worry about any of this because TCP handles it all for you. Its only when you want to build servers and things that you really get into it.

You can see your actual ip on the internet by going to a site like www.whatismyipaddress.com. Or open the status page of your modem.

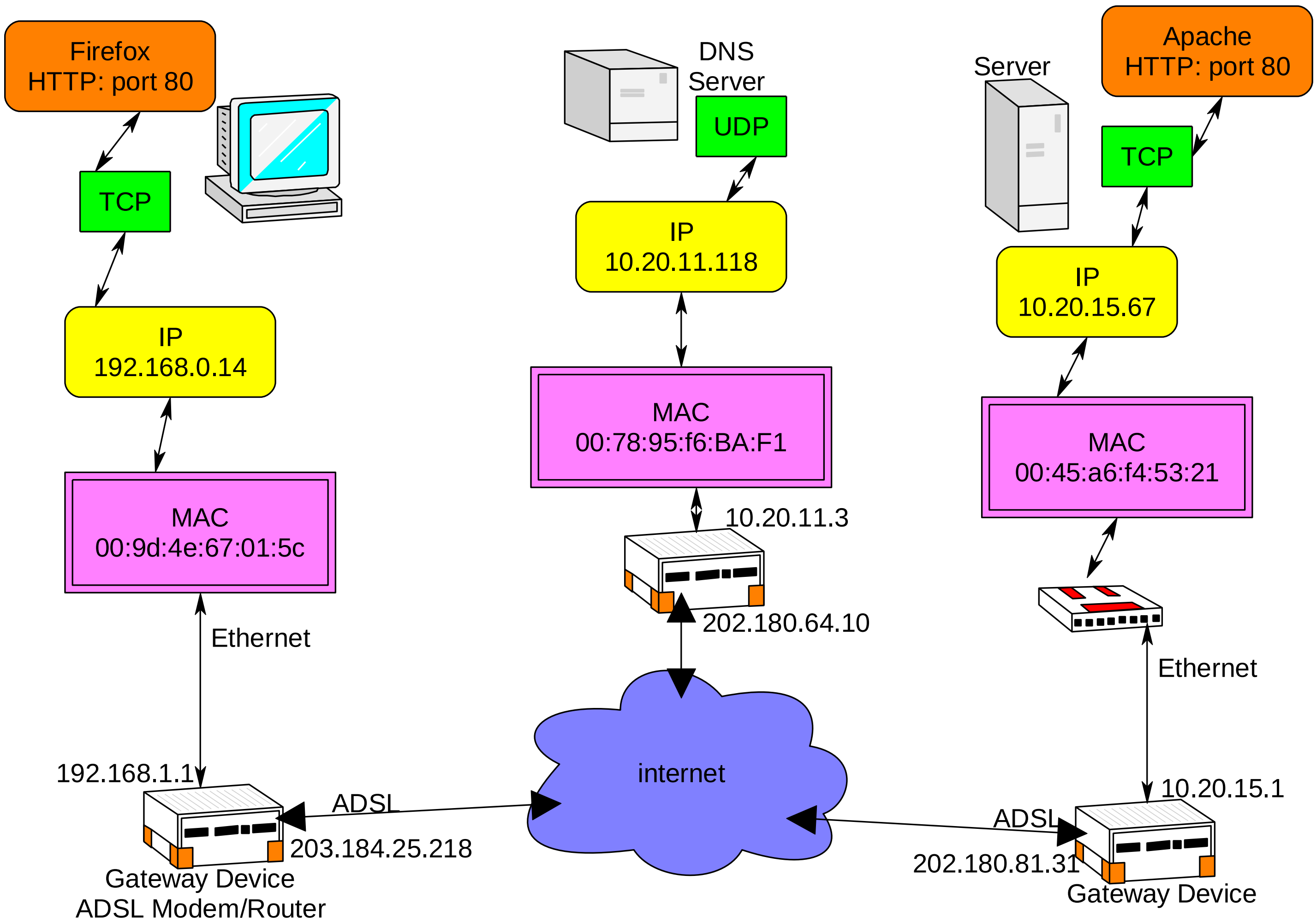


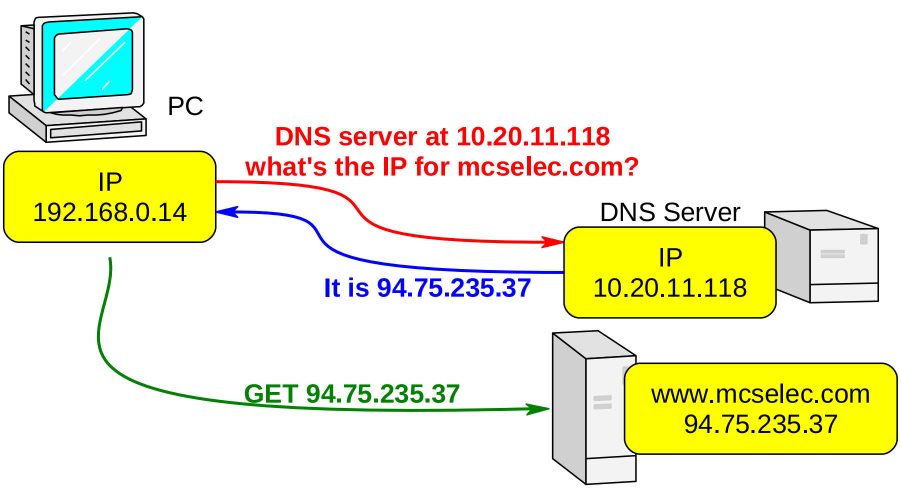
Better still go to www.grc.com and find Shieldsup and test the firewall of your modem, a firewall protects (opens/closes/hides) ports on your modem through which other devices can get into your network.

54.8 DNS

Even though computers may work on numbers humans do not, we like to use names for websites on the internet like www.techideas.co.nz or www.mcselec.com

When you type www.techideas.co.nz into a computer it has to find the ip address for it. On the status page for your modem is the IP address of the DNS (domain name system) server on the internet (usually at your ISP) that will help you. Normally your modem gets the IP for the DNS server automatically when it logs on to your ISP. It is such an important hing that you generally have access to at least 2 of them as they can get busy.



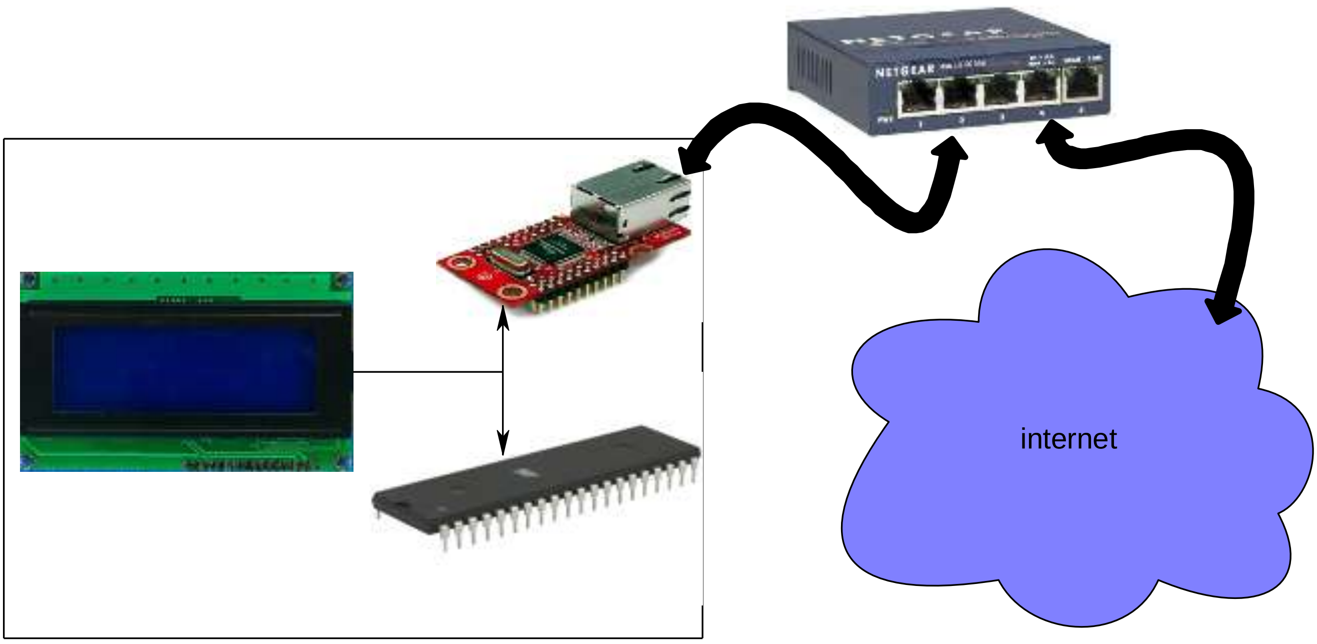


When you want a website, your computer asks the DNS server for the IP of the website, then your PC asks the server at that IP address for a web page.

54.9 WIZNET812

We are going to build a small webserver using an AVR and put it on the internet so that we can control things from anywhere around the world.

There are a number of ways of implementing a network device but using the Wiznet812 is definitely one of the easiest. It has all the TCP/IP stack (protocols) built into it. You just have to configure its IP and MAC addresses and then talk to it (sounds simple sorry it's not!!)

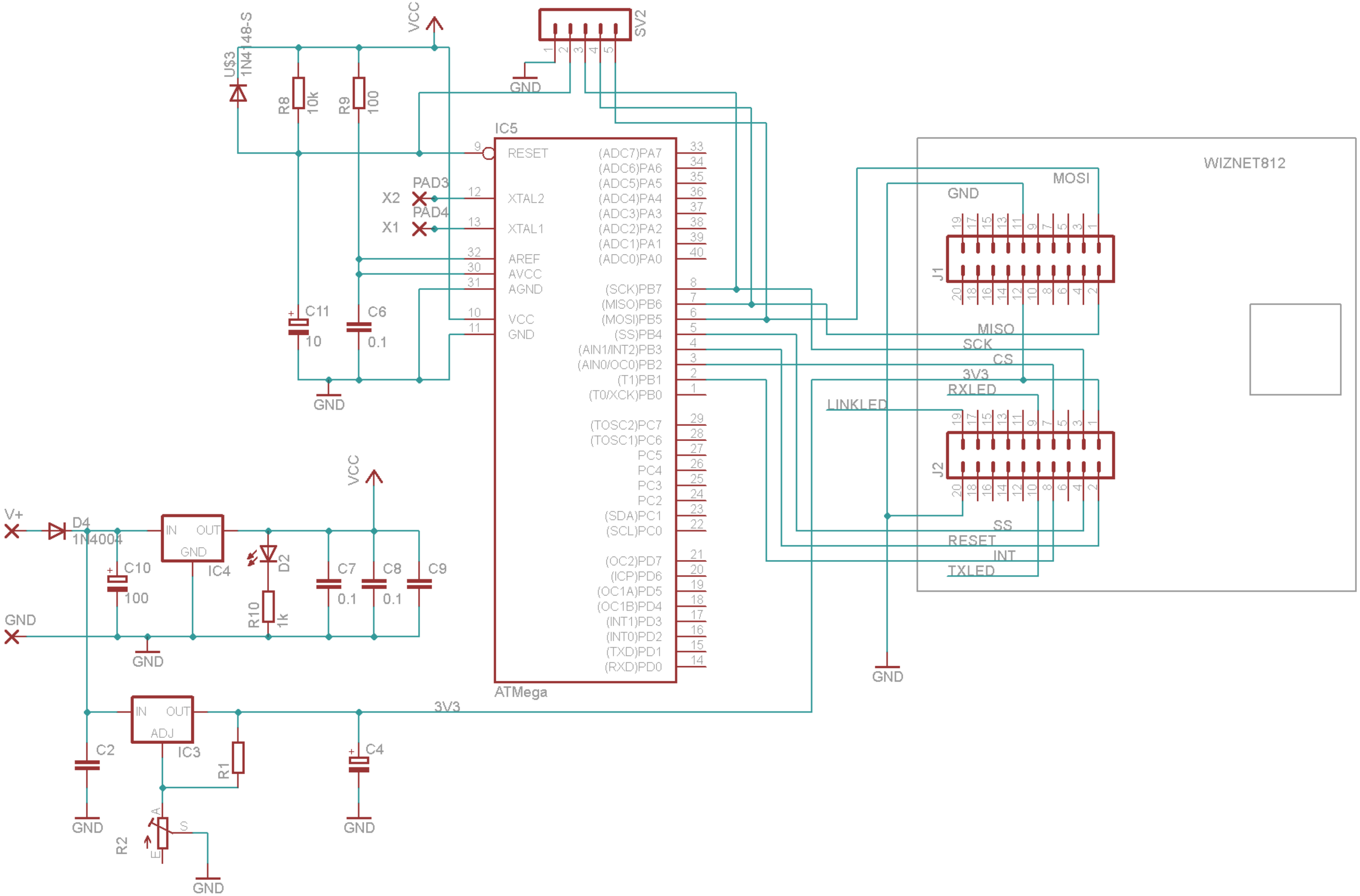


System Block Diagram

Our first Ethernet application will be to get a simple Ping working.
Note that all that is needed for a ping is to configure the Wiznet, the TCP/IP protocol stack is configured within the Wiznet, you don't have to write much of a program for this to happen.

Much of the code that follows was written based upon the most excellent work from http://members.home.nl/bzijlstra/

Circuit diagram



The wiznet requires a 3V3 power supply, its pins are however tolerant of 5V so the Wiznet will run off 3V3 and the micro and LCD off 5V

Wiznet ping program

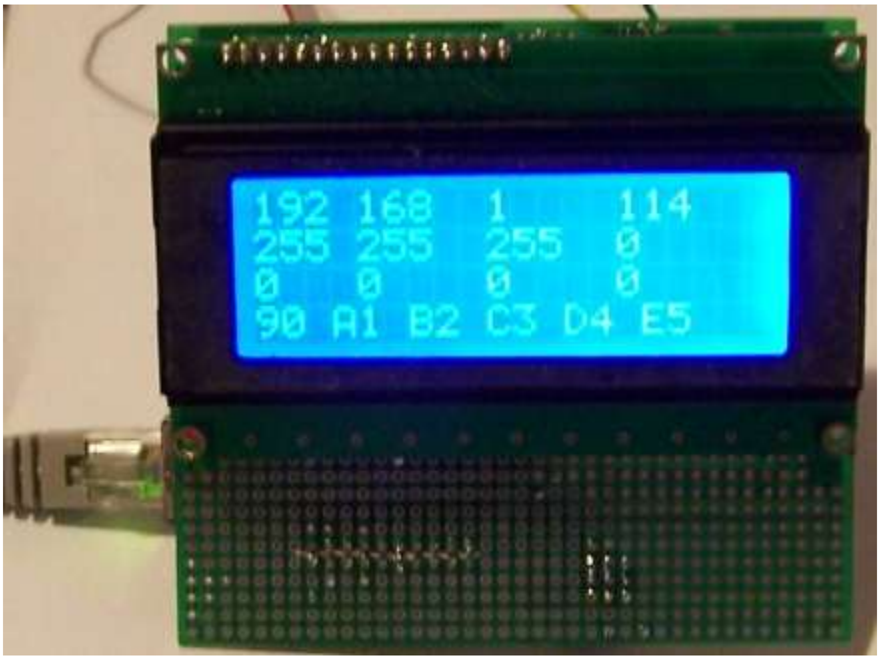
```
'-----------------------------------------------------------------
'Title Block
'
' Date:     July 09
' File Name: wiz812_Ping_v1
'-----------------------------------------------------------------
'Program Description:
' Atmega8535, char LCD and wiz812MJ PING program
'-----------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m8535.dat"
$crystal = 8000000
$hwstack = 64
$swstack = 64
$framesize = 64

'-----------------------------------------------------------------
'Hardware Setups
'Hardware Alias
Wiz812_cs Alias Portb.2                    'Chipselect wiz812
Wiz812_ss Alias Portb.4
Wiz812_int Alias Pinb.1                    'INT of wiz812
Wiz812_res Alias Portb.3                   'Reset of wiz812

'configure hardware
Config Wiz812_cs = Output
Config Wiz812_ss = Output
Config Wiz812_int = Input
Config Wiz812_res = Output

'Configuration of the SPI-bus
Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = Low , Phase = 0 ,
Clockrate = 4 , Noss = 0

'-----------------------------------------------------------------
'Declare Constants – registers within the wiznet that tell it what to do.
Const Wiz812_modereg = &H0000              'Mode register

Const Wiz812_gw0 = &H0001                  'Gateway address
Const Wiz812_gw1 = &H0002
Const Wiz812_gw2 = &H0003
Const Wiz812_gw3 = &H0004

Const Wiz812_subnet0 = &H0005              'Subnet mask
Const Wiz812_subnet1 = &H0006
Const Wiz812_subnet2 = &H0007
Const Wiz812_subnet3 = &H0008

Const Wiz812_mac0 = &H0009                 'Source Hardware Address
Const Wiz812_mac1 = &H000A
Const Wiz812_mac2 = &H000B
Const Wiz812_mac3 = &H000C
Const Wiz812_mac4 = &H000D
```

```
Const Wiz812_mac5 = &H000E

Const Wiz812_ip0 = &H000F                          'Source IP Address
Const Wiz812_ip1 = &H0010
Const Wiz812_ip2 = &H0011
Const Wiz812_ip3 = &H0012


'-----------------------------------------------------------------
'Declare Variables
Dim Value As Byte
Dim Address As Word
Dim Address_lo As Byte At Address Overlay
Dim Address_hi As Byte At Address + 1 Overlay
Dim Wiz812_rd_code As Byte
Dim Wiz812_wr_code As Byte

'Initialise Variables
 Wiz812_rd_code = 15
 Wiz812_wr_code = 240

'-----------------------------------------------------------------
'Declare subroutines
Declare Sub Wiz812_init
Declare Sub Wiz812_read(byval Register As Word)
Declare Sub Wiz812_write(byval Register As Word , Byval Value As Byte)
Declare Sub Wiz812_reset

Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 , Rs
= Portc.0
Config Lcd = 20 * 4                          'configure lcd screen

'-----------------------------------------------------------------
'Program starts here


Spiinit                                      'Initialise the spi bus

Call Wiz812_init                             'We initialize the wiz812
Cls

Do
   Gosub Display_setup                       'Just print the configuration on the LCD
Loop
End

'-----------------------------------------------------------------
Display_setup:
    Call Wiz812_read(wiz812_ip0)
    Locate 1 , 1
    Lcd Value
    Call Wiz812_read(wiz812_ip1)
    Locate 1 , 5
    Lcd Value
```

```
    Call Wiz812_read(wiz812_ip2)
    Locate 1 , 10
    Lcd Value
    Call Wiz812_read(wiz812_ip3)
    Locate 1 , 15
    Lcd Value

    Call Wiz812_read(wiz812_subnet0)
    Locate 2 , 1
    Lcd Value
    Call Wiz812_read(wiz812_subnet1)
    Locate 2 , 5
    Lcd Value
    Call Wiz812_read(wiz812_subnet2)
    Locate 2 , 10
    Lcd Value
    Call Wiz812_read(wiz812_subnet3)
    Locate 2 , 15
    Lcd Value

    Call Wiz812_read(wiz812_gw0)
    Locate 3 , 1
    Lcd Value
    Call Wiz812_read(wiz812_gw1)
    Locate 3 , 5
    Lcd Value
    Call Wiz812_read(wiz812_gw2)
    Locate 3 , 10
    Lcd Value
    Call Wiz812_read(wiz812_gw3)
    Locate 3 , 15
    Lcd Value

    Call Wiz812_read(wiz812_mac0)
    Locate 4 , 1
    Lcd Hex(value)
    Call Wiz812_read(wiz812_mac1)
    Locate 4 , 4
    Lcd Hex(value)
    Call Wiz812_read(wiz812_mac2)
    Locate 4 , 7
    Lcd Hex(value)
    Call Wiz812_read(wiz812_mac3)
    Locate 4 , 10
    Lcd Hex(value)
    Call Wiz812_read(wiz812_mac4)
    Locate 4 , 13
    Lcd Hex(value)
    Call Wiz812_read(wiz812_mac5)
    Locate 4 , 16
    Lcd Hex(value)
Return
```

```
'-----------------------------------------------------------------
Sub Wiz812_init
  Call Wiz812_reset                       'Hardware reset
  'Register reset
  Call Wiz812_write(wiz812_modereg , &H80)
  'Set static IP
  Call Wiz812_write(wiz812_ip0 , 192)
  Call Wiz812_write(wiz812_ip1 , 168)
  Call Wiz812_write(wiz812_ip2 , 1)
  Call Wiz812_write(wiz812_ip3 , 114)
  'Set Subnet mask
  Call Wiz812_write(wiz812_subnet0 , 255)
  Call Wiz812_write(wiz812_subnet1 , 255)
  Call Wiz812_write(wiz812_subnet2 , 255)
  Call Wiz812_write(wiz812_subnet3 , 0)
  'Set gateway IP address
  Call Wiz812_write(wiz812_gw0 , 0)
  Call Wiz812_write(wiz812_gw1 , 0)
  Call Wiz812_write(wiz812_gw2 , 0)
  Call Wiz812_write(wiz812_gw3 , 0)
  'Set MAC to any unique number
  Call Wiz812_write(wiz812_mac0 , &H90)
  Call Wiz812_write(wiz812_mac1 , &HA1)
  Call Wiz812_write(wiz812_mac2 , &HB2)
  Call Wiz812_write(wiz812_mac3 , &HC3)
  Call Wiz812_write(wiz812_mac4 , &HD4)
  Call Wiz812_write(wiz812_mac5 , &HE5)
End Sub

'-----------------------------------------------------------------
Sub Wiz812_read(register)
  Address = Register
  Reset Wiz812_cs
  Spiout Wiz812_rd_code , 1
  Spiout Address_hi , 1
  Spiout Address_lo , 1
  Spiin Value , 1
  Set Wiz812_cs
End Sub

'-----------------------------------------------------------------
Sub Wiz812_write(register , Value )
  Address = Register
  Reset Wiz812_cs
  Spiout Wiz812_wr_code , 1
  Spiout Address_hi , 1
  Spiout Address_lo , 1
  Spiout Value , 1
  Set Wiz812_cs
End Sub
```

```
'----------------------------------------------------------------
Sub Wiz812_reset
  Wiz812_res = 1
  Waitms 10
  Wiz812_res = 0
  Waitms 30
  Wiz812_res = 1
End Sub
'----------------------------------------------------------------
```



54.10 Wiznet 812 Webserver V1

To setup a webserver also involves understanding a bit about http communication that takes place between a browser and a server.

The browser (client) sends a GET to the server and then the server sends its webpage

A message from a browser is made up of two parts a header and a body.
The initial request is a GET message which has no body just a header and at least 2CRLF's (carriage return, line feeds) on the end.

CR & LF codes are stored in a document or sent in a message to signify to return to the beginning of the line (CR-carriage return) and go to the next line down (LF-line feed). The ASCII code for CR is 13 or &H0D, the code for LF is 10 or &H0A. A browser sends a CRLF at the end of each line and after the end of the last line a second CRLF to indicate the break between the header and any body. It also sends a CRLF at the end of the body.

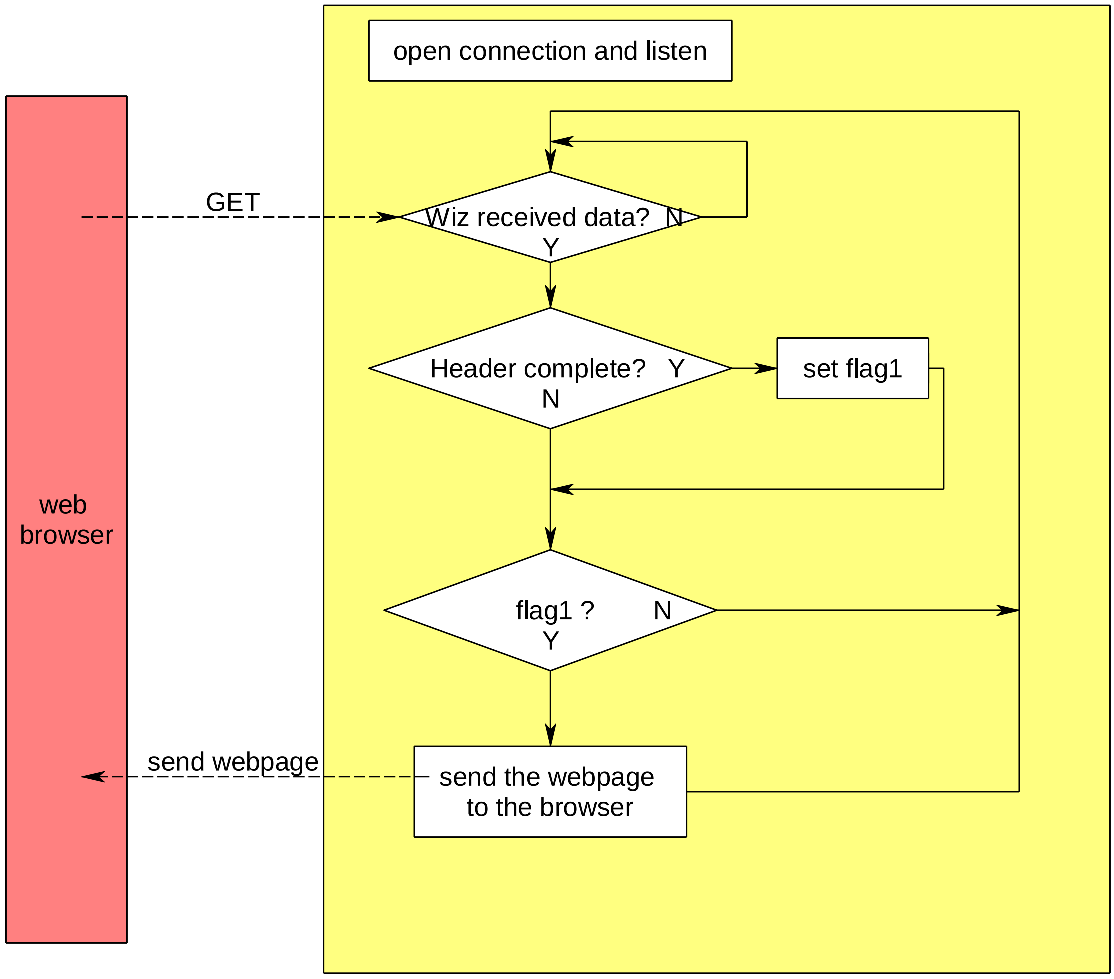
The actual GET message is a text message like this from Firefox

```
GET / HTTP/1.1
Host: 192.168.1.73
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR
3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: Close
CRLF
CRLF
```

And from intenet explorer it is:

```
GET / HTTP/1.1
Accept: image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-shockwave-flash, application/x-ms-application,
application/x-ms-xbap, application/vnd.ms-xpsdocument, application/xaml+xml, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, application/x-silverlight, */*
Accept-Language: en-nz
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CLR
3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2)
Accept-Encoding: gzip, deflate
Connection: Close
Host: 192.168.1.73
CRLF
CRLF
```

When understanding the program code for a webserver you start from when the web browser sends a GET and the server receives it.



The server software must wait for data and then check that the header is complete, it knows it is complete when it finds two CRLF in a row. If that happens it sends its webpage to the client browser.

Important point: program flags

In our program when the complete header is detected a flag is set (a single bit in a byte sized variable); afterwards in a later part of the program the flag can be checked to action something else. As our code becomes more complex, more flags will be necessary.

The Wiznet812 is based around the WIZ5100 IC which has a large memory to store data that it receives and data that you want it to send. Reasonable size memories are required because there are often significant size data transfers involved: e.g. the GET header was 386 bytes.

&H0000 &H002F	Common registers
&H0400 &H07FF	Socket registers
&H4000 &H5FFF	TX Memory (8K)
&H6000 &H7FFF	RX Memory(8K)

Here is a webpage which has been served by the wiznet

When text is entered into the textbo and enter is pressed the following HTTP header and body are sent (header in green and body in red).

POST / HTTP/1.1
Host: 192.168.1.73
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://192.168.1.73/
Content-Type: application/x-www-form-urlencoded
Content-Length: 43

TEXT2SEND=Hello world, is anyone there?

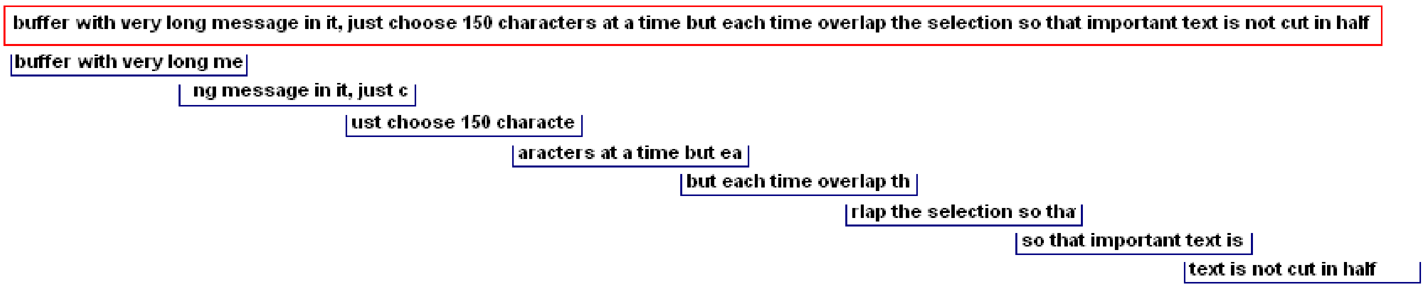
It is a POST so the webpage is not getting something from the server it is sending it to the server.

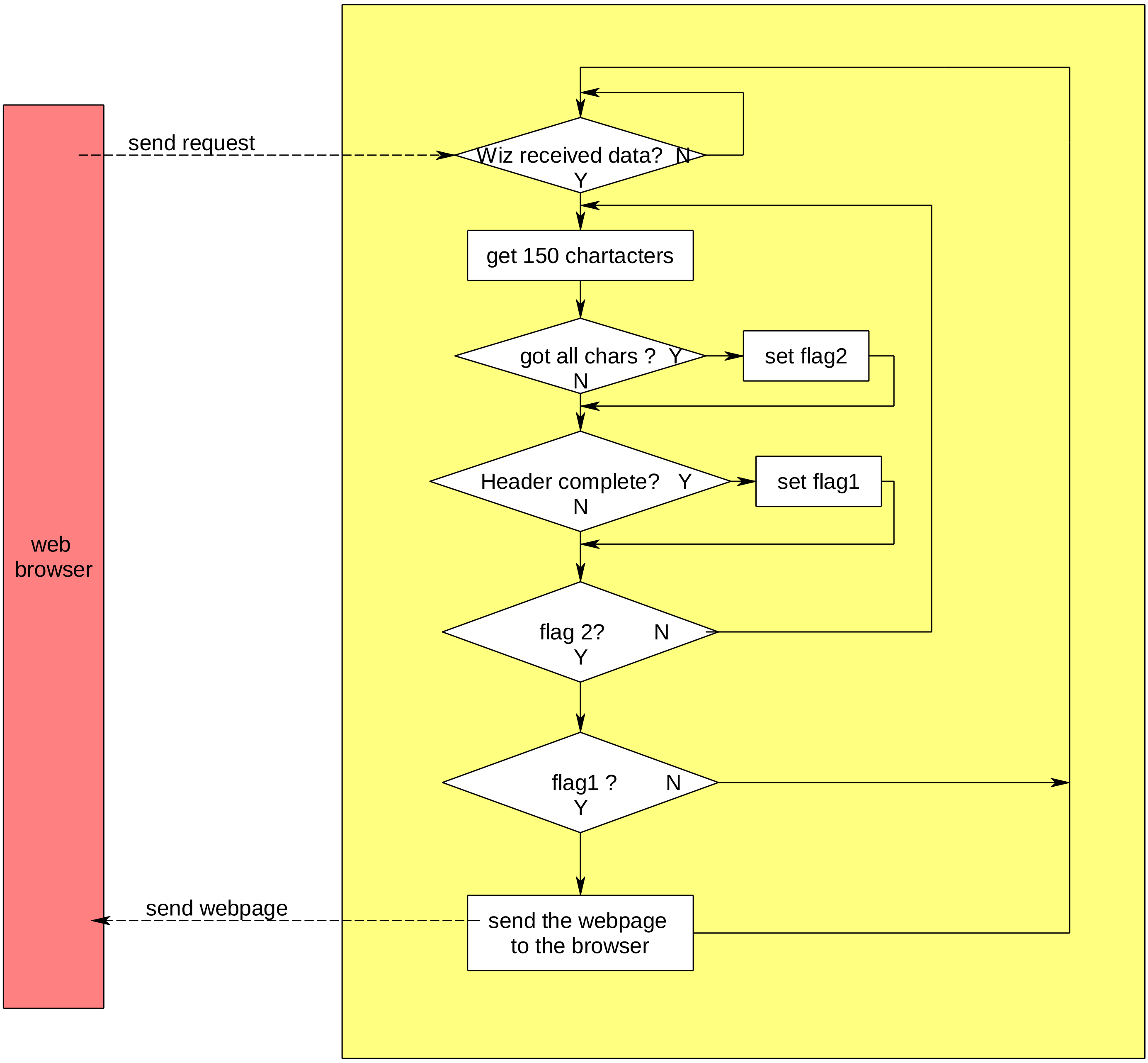
The server software must loop through the header to find the end of the header (marked by two CRLF), then it extracts the content length and using this value gets that number fo characters from the body.

There is a limitation with Bascom-AVR though which really complicates our program. Bascom is set up to handle strings of a maximum of 254 characters in length yet the simple GET header was almost 400 characters and the POST header is over 500 characters.

To read an HTTP header we only grab 200 characters at a time from the wiznet and check these for the CRLF CRLF end of header. When we have this we set a flag _flag.2)

A complication exists though as we do not want to get blocks of 200 characters at a time and find that an important piece of data was cut. We would loose important content doing that. So an overlap process is used with the buffer to avoid cutting important words or phrases up.
The first read is from 1 to 200, the second from 150 to 249, the third from 200 to 249 and so on



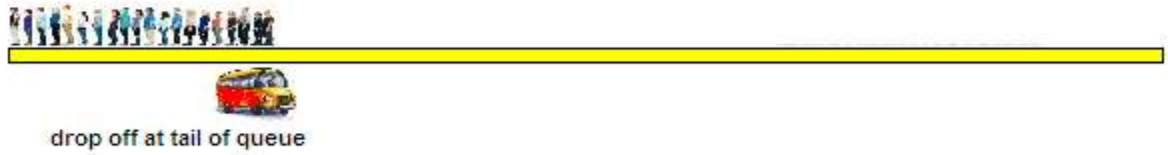

send request
Wiz received data? N
Y
get 150 chartacters
got all chars ? Y
N
set flag2
Header complete? Y
N
set flag1
flag 2? N
Y
flag1 ? N
Y
send the webpage to the browser
send webpage
web browser

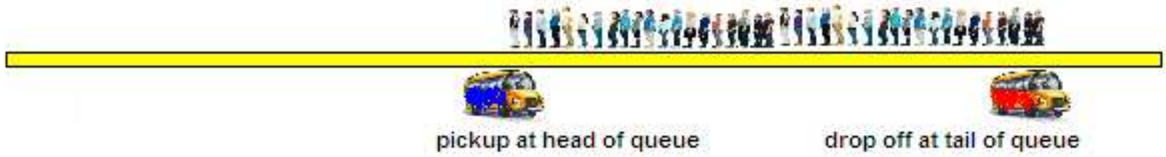
54.11 Transmitting data

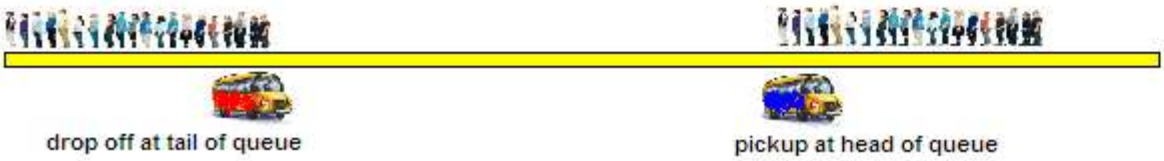
There can be a lot of data to send when it comes to web pages, and we have a limited resource of memory available in the wiznet to store and send this data. A data structure called a queue or buffer is required to manage the sending of the data and the holding of it until it can be sent. It is a FIFO (first in first out) queue.

Imagine a major bus station, it has a 300metre long platform where a lot of passengers have to transfer from one bus to another; except the busses run at slightly different schedules. At a normal bus stop the people join the end of the queue and as people get on the first available bus the whole queue moves forward, just like people waiting at a supermarket checkout or a bank ATM.

However at a busy bus platform that is 300metres long we don't want the people to shuffle all the way down the platform to catch the outgoing bus. Everyone would get really cross with having to pick up their parcels and move every few seconds and then wait, then move, then wait a bit more…
So we have the bus drivers drop people off at the end of the queue and the people wait in one spot, then the outgoing bus drivers pick up people at the front of the queue.



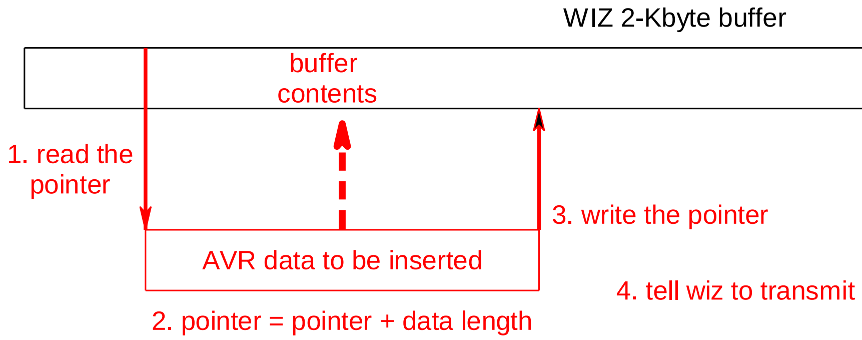




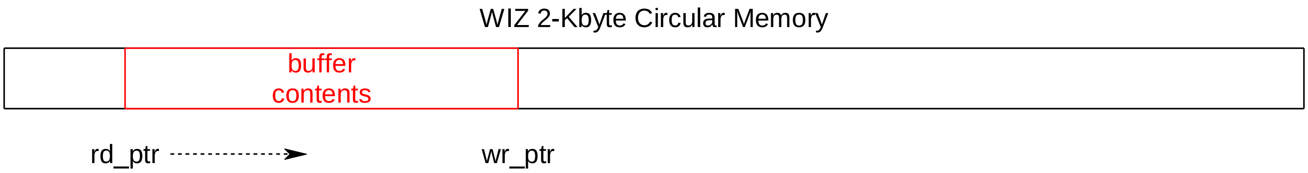
The first pictures above are clear, the queue grows down the platform. But as we get to the end of the platform it is clear that we cannot drop off the new passengers as there is no room, so the bus driver drops them off at the other end of the platform.

Memory in a computer is a bit like the bus platform, it is of limited length(size) so if we add data to the end of ram, eventually we must run out and then we need to start our queue from the beginning again.

In the wiznet there are two pointers used to manage the head and tail of the buffer or queue. We are going to add the contents of the AVR ram buffer into the wiznet buffer.



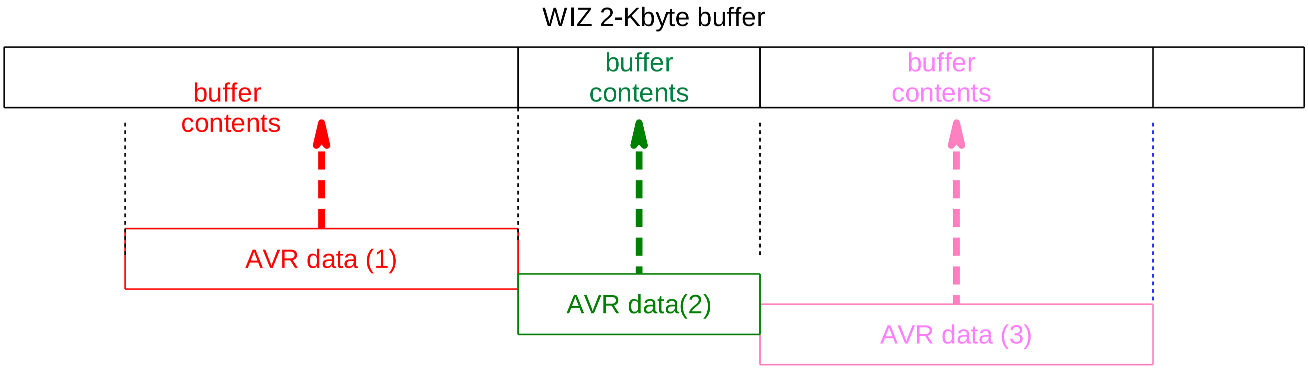
When inserting the new contents into the wiznet buffer, the program first reads the pointer soc0_tx_wr_ptr which tells it where to start copying into the buffer, it then copies the data from that point , then calculates the new value for soc0_tx_wr_ptr by adding the length of the new data to it and finally writes the new value into the pointer.
e.g.
soc0_tx_wr_prtr contains the address &H413E
AVR data = "*<html><head><meta http-equiv={034}PRAGMA{034} Content={034}NO-CACHE{034}/>*"
Data length is 74 characters = &H4A
new value for soc0_tx_wr_ptr = &H413E + &H41 = &H4188



The wiznet maintains a second memory pointer soc0_tx_rd_ptr which it uses to read the memory content from the head of the queue.

As data is transmitted across the network the rd_ptr moves towards the wr_ptr, the wiznet stops sending when the two pointers are the same as it has then sent all the data in its buffer.

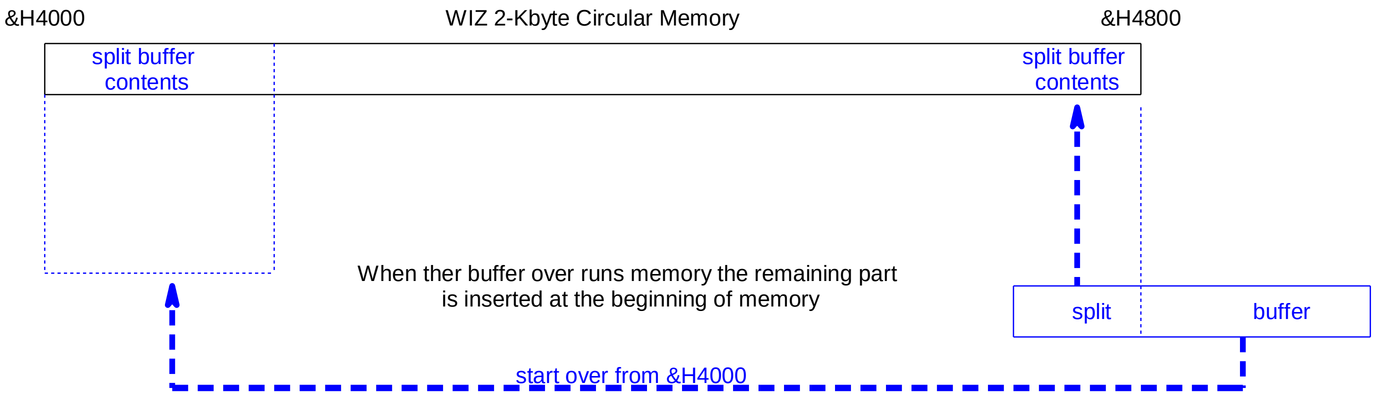
The wiznet has a freesize register as well which can be read at anytime to find out how much tx buffer memory is available.



The program reads data from the end of the program line by line; each new line replaces the old one in the buffer (the buffer does not get longer).

After this the entire buffer is copied into the memory at the next location after the last.

After each copy into memory the tx_wr_ptr is set to the new value at the end of the buffer contents and and the wiz sends the data onto the network.



At the end of WIZnet memory the buffer may be split up and wraps to the beginning again, just like at the bus station.

Wiznet812 server program Ver 1

```
'-----------------------------------------------------------------
'Title Block
' Date:     July 09
' File Name: WebServ_V1
'-----------------------------------------------------------------
'Program Description:
' AtMega16, char LCD and w812MJ Webserver
'-----------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m16def.dat"
$crystal = 8000000
$baud = 9600
$hwstack = 60
$swstack = 60
$framesize = 80

'-----------------------------------------------------------------
'Hardware Setups
'Hardware Alias
w812_cs Alias Portb.2                    'Chipselect w812
w812_ss Alias Portb.4
w812_int Alias Pinb.1                    'INT of w812
w812_res Alias Portb.3                   'Reset of w812

'configure hardware
Config w812_cs = Output
Config w812_ss = Output
Config w812_int = Input
Config w812_res = Output

'Configuration of the SPI-bus
Config Spi = Hard , Interrupt = Off , Data Order = Msb , Master = Yes , Polarity = Low , Phase = 0 ,
Clockrate = 4 , Noss = 0

'lcd
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7 = Portc.5 , E = Portc.1 ,
Rs = Portc.0
Config Lcd = 20 * 4                      'configure lcd screen
'-----------------------------------------------------------------
'Declare subroutines
Declare Function w812_receive_check() As Byte
Declare Sub w812_send_webpage
Declare Sub W812_send_buffer
Declare Sub w812_init
Declare Sub w812_reset
Declare Sub w812_cycleport
Declare Function W812_readb(byval Register As Word) As Byte
Declare Function W812_readw(byval Register_h As Word) As Word
Declare Sub W812_writeb(byval Register As Word , Byval Dat As Byte)
Declare Sub W812_writew(byval Register_h As Word , Byval Dat As Word)
```

```
'-----------------------------------------------------------------
'Declare Constants
Const w812_modereg = &H0000                    'Mode register

Const w812_gw0 = &H0001                     'Gateway address
Const w812_gw1 = &H0002
Const w812_gw2 = &H0003
Const w812_gw3 = &H0004

Const w812_subnet0 = &H0005                  'Subnet mask
Const w812_subnet1 = &H0006
Const w812_subnet2 = &H0007
Const w812_subnet3 = &H0008

Const w812_mac0 = &H0009                     'Source Hardware Address
Const w812_mac1 = &H000A
Const w812_mac2 = &H000B
Const w812_mac3 = &H000C
Const w812_mac4 = &H000D
Const w812_mac5 = &H000E

Const w812_ip0 = &H000F                      'Source IP Address
Const w812_ip1 = &H0010
Const w812_ip2 = &H0011
Const w812_ip3 = &H0012

Const W812_s0_modereg = &H0400
Const w812_s0_commandreg = &H0401
Const w812_s0_intr = &H0402
Const w812_s0_status = &H0403
Const w812_s0_porth = &H0404
Const w812_s0_portl = &H0405
Const w812_s0_destip_1 = &H040C
Const w812_s0_destip_2 = &H040D
Const w812_s0_destip_3 = &H040E
Const w812_s0_destip_4 = &H040F
Const w812_s0_txfreesizeh = &H0420
Const w812_s0_txfreesizel = &H0421
Const w812_s0_txrdptrh = &H0422
Const w812_s0_txrdptrl = &H0423
Const w812_s0_txwrptrh = &H0424
Const w812_s0_txwrptrl = &H0425
Const w812_s0_rxsizeh = &H0426
Const w812_s0_rxsizel = &H0427

Const Buffersize = 254

'-----------------------------------------------------------------
'Declare Variables
Dim Buffer As String * Buffersize
Dim Rx_flag As Byte
Dim W812_rd_code As Byte
Dim w812_wr_code As Byte
Dim Soc0_status As Byte
```

```
'Initialise Variables
W812_rd_code = 15
W812_wr_code = 240


'-----------------------------------------------------------------
 'program starts here
Cls
Lcd "  w812 Server  "
Spiinit                                                    'Initialise the spi pins
Call w812_init                                             'initialize  w812
Do
   Soc0_status = w812_readb(w812_s0_status)               ''find state of socket
   If Soc0_status = &H0 Or Soc0_status = &H1C Then
      Call w812_cycleport()                                'try to open socket
   End If
   Rx_flag = w812_receive_check()                          'see if anything received
   If Rx_flag.1 = 1 Then                                   'full header found
      Call w812_writeb(w812_s0_commandreg , &H40)
      Call w812_send_webpage()                             'send out the webpage
   End If
Loop
End

'-----------------------------------------------------------------
Sub w812_send_webpage
  Local Char As Byte
  'fill buffer with lines from the data at end of the program and send them
  Restore Served_webpage                     'start from data beginning
  Print "-------------starting to send webpage----------------------"
  Do
     Print "--------"
     Read Buffer                          'get data line by line from below
     If Buffer = "SEND_CLIENT_IP" Then        ' insert the client IP-address
        Char = W812_readb(w812_s0_destip_1)   'in the web page
        Buffer = Str(char) + "."             'empty buffer to start with
        Char = W812_readb(w812_s0_destip_2)
        Buffer = Buffer + Str(char) + "."
        Char = W812_readb(w812_s0_destip_3)
        Buffer = Buffer + Str(char) + "."
        Char = W812_readb(w812_s0_destip_4)
        Buffer = Buffer + Str(char)
     End If
     If Buffer = "END_OF_WEB_PAGE" Then       'Look for the end of a webpage
        Exit Do
     End If
     Call W812_send_buffer                  'send the buffer
   Loop
  'finished sending so disconnect
  Call w812_writeb(w812_s0_commandreg , &H8)
End Sub


'-----------------------------------------------------------------
```

```
'check to see if the wiz has received any data
' if a full header has been received set flag.1
Function w812_receive_check() As Byte
   Local Temp As Word , I As Word , J As Word , Flag As Byte , _status As Byte
   Local Contentpos As Word , Top As Word , Addr_ptr As Word , Rx_count As Word
   Local Complete_header As String * 4 , Char As Byte
   Buffer = ""
   Complete_header = Chr(13) + Chr(10) + Chr(13) + Chr(10)     'gap between header and body
   Contentpos = 0
   Addr_ptr = 0
   Flag = 0

   _status = W812_readb(w812_s0_status)
   If _status = &H17 Then                                    'check if connected first
      'Check for new data  received by wiz
      Rx_count = W812_readw(w812_s0_rxsizeh)

      If Rx_count > 0 Then                                   'received something
         I = &H6000
         J = &H6000 + 200
         While Flag.2 = 0                                    'for all received data
            'get 200 characters at a time from wiz
            Buffer = ""                                      'empty the buffer
            Rx_count = Rx_count - 1
            Top = &H6000 + Rx_count
            Top = Top + 3
            For Addr_ptr = I To J
               If Addr_ptr < Top Then      'not at end yet
                  Char = W812_readb(addr_ptr)      'get a byte from wiz
                  Buffer = Buffer + Chr(char)      'store ascii char in buffer
               Else
                  Flag.2 = 1            'reached the end
               End If
            Next
            Temp = Instr(buffer , Complete_header)
            If Temp > 0 Then Flag.1 = 1      'full header and body
            I = I + 150                 'slide up the buffer 150 chars
            J = J + 150                  'slide up the buffer 150 chars
         Wend
      End If
   End If
   w812_receive_check = Flag
End Function
```

```
'-----------------------------------------------------------------
'copies contents of the buffer into tx_mem and tells wiz to send it
Sub W812_send_buffer
   Local _tx_wr_ptr As Word , _bufferlength As Integer , _tx_freesize As Word
   Local _tx_mem_offset_low As Word , _tx_mem_offset_high As Word , _tx_mem_ptr As Word
   Local _lower_buffer As Word , _str As String * 1 , _char As Byte , _i As Byte
   _bufferlength = Len(buffer)                'length of data to send

   '1. wait until wiz has enough memory available to insert the full contents of the buffer
   Do
   _tx_freesize = W812_readw(w812_s0_txfreesizeh)
   Loop Until _tx_freesize > _bufferlength

   '2. find tx_wr_ptr - the position in memory for inserting buffer contents
   _tx_wr_ptr = W812_readw(w812_s0_txwrptrh)
   _tx_mem_offset_low = _tx_wr_ptr And &H7FF
   _tx_mem_offset_high = _tx_mem_offset_low + _bufferlength
   _tx_mem_ptr = &H4000 + _tx_mem_offset_low

   '3. copy the buffer into tx_memory
   If _tx_mem_offset_high < &H801 Then        'no need to split buffer
      For _i = 1 To _bufferlength
          _str = Mid(buffer , _i , 1)
         _char = Asc(_str)
         Call W812_writeb(_tx_mem_ptr , _char)
         Incr _tx_mem_ptr
      Next _i
   Else                                       'we need to split buffer
      _lower_buffer = &H800 - _tx_mem_offset_low   'through to the end of mem
      For _i = 1 To _lower_buffer
         _str = Mid(buffer , _i , 1)
         _char = Asc(_str)
         Call W812_writeb(_tx_mem_ptr , _char)
         Incr _tx_mem_ptr
      Next _i
      _tx_mem_ptr = &H4000
      Incr _lower_buffer
      For _i = _lower_buffer To _bufferlength
         _str = Mid(buffer , _i , 1)
         _char = Asc(_str)
         Call W812_writeb(_tx_mem_ptr , _char)
         Incr _tx_mem_ptr
      Next _i
   End If

   '4. tell wiz the end of the data to send by moving tx_ptr forward
   _tx_wr_ptr = _tx_wr_ptr + _bufferlength
   Call W812_writew(w812_s0_txwrptrh , _tx_wr_ptr)
   '5. tell wiz to send data from tx_rd_ptr to tx_wr_ptr
   Call w812_writeb(w812_s0_commandreg , &H20)      'send
End Sub
```

```
'--------------------------------------------------------------
Sub w812_init
  Call w812_reset
  Call w812_writeb(w812_modereg , &H80)
  'Set Subnet mask
  Call w812_writeb(w812_subnet0 , 255)
  Call w812_writeb(w812_subnet1 , 255)
  Call w812_writeb(w812_subnet2 , 255)
  Call w812_writeb(w812_subnet3 , 0)
  'Set gateway IP address
  Call w812_writeb(w812_gw0 , 0)
  Call w812_writeb(w812_gw1 , 0)
  Call w812_writeb(w812_gw2 , 0)
  Call w812_writeb(w812_gw3 , 0)
  'Set MAC to any unique number
  Call w812_writeb(w812_mac0 , &H90)
  Call w812_writeb(w812_mac1 , &HA1)
  Call w812_writeb(w812_mac2 , &HB2)
  Call w812_writeb(w812_mac3 , &HC3)
  Call w812_writeb(w812_mac4 , &HD4)
  Call w812_writeb(w812_mac5 , &HE5)
   'Set static IP
  Call w812_writeb(w812_ip0 , 192)
  Call w812_writeb(w812_ip1 , 168)
  Call w812_writeb(w812_ip2 , 1)
  Call w812_writeb(w812_ip3 , 73)
  'Initialize socket 0 as TCP
   Call w812_writeb(w812_s0_modereg , &H1)
  'Port 5000=&H1388     80=&H0050 HTTP
  Call w812_writeb(w812_s0_porth , &H0 )
  Call w812_writeb(w812_s0_portl , &H50 )

  Call w812_cycleport()
End Sub

'--------------------------------------------------------------
Sub w812_reset
   'hardware reset for wiz
  w812_res = 1
  Waitms 10
  w812_res = 0
  Waitms 30                            'Minimum 20 µs
  w812_res = 1
End Sub

'--------------------------------------------------------------
Sub w812_cycleport
  'close the socket, reopen it and wait
  Call W812_writeb(w812_s0_commandreg , &H0)    'close soc0
  Call W812_writeb(w812_s0_commandreg , &H1)    'open soc0
  Call W812_writeb(w812_s0_commandreg , &H2)    'listen on soc0
End Sub
```

```
'-------------------------------------------------------------------
Sub w812_writeb(register , Dat )
   Local _bh As Byte
   Local _bl As Byte
   _bh = High(register)                    'send address high byte
   _bl = Low(register)                     'send address low byte
   Reset w812_cs
   Spiout w812_wr_code , 1
   Spiout _bh , 1
   Spiout _bl , 1
   Spiout Dat , 1
   Set w812_cs
End Sub
'-------------------------------------------------------------------
Sub W812_writew(register_h , Dat)
   Local _d As Byte

   _d = High(dat)
   Call W812_writeb(register_h , _d)          'send high byte to high addr
   Incr Register_h
   _d = Low(dat)
   Call W812_writeb(register_h , _d)          'send low byte to low addr
End Sub
'-------------------------------------------------------------------
Function W812_readb(register)
    'get 1 byte from a wiznet register
   Local _bh As Byte
   Local _bl As Byte
   _bh = High(register)                    'send address high byte
   _bl = Low(register)                     'send address low byte
   Reset W812_cs
   Spiout W812_rd_code , 1                   'tell wiz we want to read
   Spiout _bh , 1
   Spiout _bl , 1
   Spiin _bl , 1                          'get 1 byte
   Set w812_cs
   W812_readb = _bl                        'return the byte
End Function
'-------------------------------------------------------------------
Function W812_readw(register_h)
    'get 1 word from a register pair
    'read high address then low address
   Local _b As Byte
   Local _w As Word
   'get high byte
    _b = W812_readb(register_h)               'get data from high addr
   _w = _b                              'put into low 8 buts of a word
   Shift _w , Left , 8                      'move to hi 8 bits
   Incr Register_h                          'set next address
   _b = W812_readb(register_h)               'get data from low addr
   _w = _w + _b                           'put together
   W812_readw = _w                         'return the word
End Function

'-------------------------------------------------------------------
```

```
Served_webpage:
'BE CAREFUL EDITING AS SOME SPACES ARE CRUCIAL
'{013}{010} replaces CR LF
'{034} replaces "
Data "<html><head><meta http-equiv={034}PRAGMA{034} Content={034}NO-CACHE{034}/>"
'tell browser not to cache page
Data "<title>WIZNET812 WebServ_V1</title></head><body><center><H1> Welcome "
Data "SEND_CLIENT_IP"                         'dynamically build ip addr in loop
Data " to the WIZNET812 webserver </H1></body></html>"
Data "END_OF_WEB_PAGE"                        'tell program webpage is finished
```

54.12 Wiznet Server2 (version1)

The above programs explain the operation of the wiznet server, however they are highly complex for students to work with so the program has been broken down into three major sections.

A. The main program
B. The wiznet setups
C. The routines to control the wiznet (that the user doesn't have to know about)

Here is the main loop.

```
'-------------------------------------------------------------------
'Title Block
' Author:    B.Collis
' Date:      Aug 09
' File Name: WebServ_V4

'-------------------------------------------------------------------
'Program Description:
' Atmega16, char LCD and w812MJ Webserver

'-------------------------------------------------------------
'Compiler Directives (these tell Bascom things about our hardware)
$regfile = "m16def.dat"
$crystal = 8000000
$baud = 9600
$hwstack = 60
$swstack = 60
$framesize = 80


'-------------------------------------------------------------------
'Hardware Setups
'the pins the wiz is connected to
W812_cs Alias Portb.2                       'Chipselect w812
W812_ss Alias Portb.4                       'not used
W812_int Alias Pinb.1                       'INT of w812
W812_res Alias Portb.3                      'Reset of w812
```

```
'all the other setups are in here
$include "WebServ2_setups.bas"

'the address etc for our wiz on the local network
W812_ip(1) = 192
W812_ip(2) = 168
W812_ip(3) = 1
W812_ip(4) = 73
W812_gw(1) = 192
W812_gw(2) = 168
W812_gw(3) = 1
W812_gw(4) = 1
W812_msk(1) = 255
W812_msk(2) = 255
W812_msk(3) = 255
W812_msk(4) = 0
W812_mac(1) = 10
W812_mac(2) = 11
W812_mac(3) = 12
W812_mac(4) = 13
W812_mac(5) = 14
W812_mac(6) = 15


'lcd
Config Lcdpin = Pin , Db4 = Portc.2 , Db5 = Portc.3 , Db6 = Portc.4 , Db7
= Portc.5 , E = Portc.1 , Rs = Portc.0
Config Lcd = 20 * 4                         'configure lcd screen

'ports to be controlled
Ctrl_0 Alias Porta.0
Ctrl_1 Alias Porta.2

'Config as outputs
Config Ctrl_0 = Output
Config Ctrl_1 = Output
'intiialise as on or off
Ctrl_0 = 0
Ctrl_1 = 1
```

```
'------------------------------------------------------------------
'wiznet program starts here
Ctrl_0 = 1                                'flash an LED
Waitms 500
Ctrl_0 = 0
Cls
Lcd " Wiznet812  CONTROL "

'Init the spi pins
Spiinit
Gosub W812_init                           'We initialize the wiz with its
settings

'we setup the watchdog timer for 2048mSecs
'if the program doesn't execute the reset watchdog command at least
'every 2 seconds, the microcontroller hardware will reset itself
'this is a really good safety mechanism
Config Watchdog = 2048                    'Watchdog configuration for
Start Watchdog                            'Start the watchdog
If Debug_word.3 = 1 Then Stop Watchdog    'in test mode
```

54.13 'Main do loop

```
'the main do-loop looks to see if something arrived,
'if it did then it looks to see if it contained a message
Do
    Reset Watchdog                        'Reset the watchdog
    'Get socket status
    Gosub Get_w812_status

    'do something if a connection has happened
    If W812_status = W812_connected Then        'if we are connected
        Rx_flag = W812_receive_check()   'see if anything received
        If Rx_flag.1 = 1 And Rx_flag.0 = 1 Then    'body and"Content-
Length:" both present
            If Debug_word.6 = 1 Then Print "rx_flag=" ; Bin(rx_flag)
            Gosub Process_received_data 'here to process received mesgs
        End If
        'if we got at least a request then send the web page back
        If Rx_flag.1 = 1 Then             'full header found
            Call W812_writeb(w812_s0_commandreg , &H40)
            Call W812_send_webpage()      'send out the webpage
            Rx_flag = 0                   'everything processed
            If Debug_word.6 = 1 Then Print "rx_flag=" ; Bin(rx_flag)
        End If

    End If
    'Connection was closed or is in the process of closing so we start
the socket new
    If W812_status = &H0 Or W812_status = &H1C Or W812_status = &H18 Then
        Call W812_cycleport()
    End If
Loop

End
```

54.14 process any messages received from browser

```
'-----------------------------------------------------------------------
'this sub will be entered when the user has interacted with the webpage
in some way
'e.g. pressed a button or pressed enter in a text box.
'it will not be entered when the user first looks at the page in their
browser.
'the codes in the buffer that it looks for are built into the webpage
below.

Process_received_data:
   'here we check to see if the user pressed the button ctrl_0_on
   If Instr(buffer , "CTRL_0=ON") > 0 Then
      Ctrl_0 = 1                          'turn that port on
      Locate 1 , 1                        'blank a line of the LCD
      Lcd Spc(20)
      Locate 1 , 1
      Lcd "CTRL_0=ON"                     'say what was received
   End If

   If Instr(buffer , "CTRL_0=OFF") > 0 Then
      Ctrl_0 = 0
      Locate 1 , 1
      Lcd Spc(20)
      Locate 1 , 1
      Lcd "CTRL_0=OFF"
   End If

   If Instr(buffer , "CTRL_1=ON") > 0 Then
      Locate 1 , 1
      Lcd Spc(20)
      Locate 1 , 1
      Lcd "CTRL_1=ON"
      Ctrl_1 = 1
   End If

   If Instr(buffer , "CTRL_1=OFF") > 0 Then
      Locate 1 , 1
      Lcd Spc(20)
      Locate 1 , 1
      Lcd "CTRL_1=OFF"
      Ctrl_1 = 0
   End If

   'here we process the text the browsersent us
   If Instr(buffer , "TEXT2SEND=") > 0 Then
   Cls
   Lcd "text arrived"
   'separate the text into three lines for the lcd
   Length = Len(buffer)
   Buffer = Mid(buffer , 11 , Length)    'strip 'TEXT2SEND='
   Locate 2 , 1
   Length = Len(buffer)
   Select Case Length
     Case 1 To 20 :
                    Lcd Buffer
```

```
                         Locate 3 , 1
                         Lcd Spc(20)
                         Locate 4 , 1
                         Lcd Spc(20)
         Case 21 To 40:
                         Lcd Left(buffer , 20)
                         Locate 3 , 1
                         I = Length - 20
                         Lcd Mid(buffer , 21 , I)
                         Locate 4 , 1
                         Lcd Spc(20)
         Case 41 To 60:
                         Lcd Left(buffer , 20)
                         Locate 3 , 1
                         Lcd Mid(buffer , 21 , 20)
                         Locate 4 , 1
                         I = Length - 40
                         Lcd Mid(buffer , 41 , I)
         Case Is > 60:
                         Lcd Left(buffer , 20)
                         Locate 3 , 1
                         Lcd Mid(buffer , 21 , 20)
                         Locate 4 , 1
                         I = Length - 40
                         Lcd Mid(buffer , 41 , 60)
   End Select
   End If
Return


'-------------------------------------------------------------------
'-------------------------------------------------------------------
'-------------------------------------------------------------------
'this external file has all the routines we need to control the wiznet
$include "WebServ2_functions.bas"
```

54.15 Served webpage

```
'-------------------------------------------------------------------
'here we build the webpage that the wiz will send to the  browser
```

Served_webpage:

```
'BE CAREFUL EDITING AS SOME SPACES ARE CRUCIAL
'Variables must be on their own lines !!!
'everytime an input is wanted a form is created for it
'  rather than 1 big form for for the whole webpage
'  this means that only the data changed is sent not the whole lot
'{013}{010} measb send a CR LF
'{034} means send a "
Data "HTTP/1.0 200 Document follows{013}{010}"
Data "Server: w812MJ AVR server{013}{010}"
Data "Content-Type: text/html{013}{010}{013}{010}"
Data "<html>"
Data "<head>"
Data "<meta http-equiv={034}PRAGMA{034} Content={034}NO-CACHE{034}/>"
'tell browser not to cache page
Data "<title>WIZNET812 WebServ_V2</title>"
Data "</head>"

Data "<body>"                               'body of the html document
Data "<center>"                             'center the web page
Data "<h1> Welcome "                        'in heading 1 format
Data "<font color={034}blue{034}>"
Data "SEND_CLIENT_IP"                       'this tells the sendng routine to
send your ip back to you
Data "<font color={034}black{034}>"         'in a different colour
Data " to my WIZnet812 web server</h1>"     'a title for the page
Data "<hr>"                                 'insert a blank line

Data "<table width={034}400{034} border = {034}9{034}>"       'create a
table with wide border
Data "<caption><h3>I/O Control</h3></caption>"       'with a caption

Data "<tr>"                                 'begin a row in the table
Data "<th>PORT</th>"                        'text in first cell <th> means
heading
Data "<td> </td>"                      'blank space so cell looks good
Data "<td> </td>"                      'blank space so cell looks good
Data "</tr>"                                'finish this row

Data "<tr>"                                 'begin a new row
Data "<th>A.0</th>"                         'text in first cell <th> means
heading
Data "<td><center>"                         'table data
Data "<form style={034}display:inline{034} action= {034}{034} method=
{034}POST{034} >"       'need a form to post data
Data "<input type= {034}submit{034} name= {034}CTRL_0{034} value=
{034}ON{034}>"       'button & data to send
Data "</form></td>"                         'end of form, end of table data
Data "<td><center> "                        'new cell
Data "<form style={034}display:inline{034} action= {034}{034} method=
{034}POST{034} >"       'need a form to post data
Data "<input type= {034}submit{034} name= {034}CTRL_0{034} value=
```

```
{034}OFF{034}>"        'button & data to send
Data "</form></td>"                      'end of form, end of table data
Data "</tr>"                             'finish row

Data "<tr>"                              'begin a row
Data "<th>A.1</th>"
Data "<td><center>"
Data "<form style={034}display:inline{034} action= {034}/{034} method=
{034}POST{034} >"        'need a form to post data
Data "<input type= {034}submit{034} name= {034}CTRL_1{034} value=
{034}ON{034}>"        'button & data to send
Data "</form></td>"                      'end of form
Data "<td><center>"
Data "<form style={034}display:inline{034} action= {034}{034} method=
{034}POST{034} >"        'need a form to post data
Data "<input type= {034}submit{034} name= {034}CTRL_1{034} value=
{034}OFF{034}>"        'button & data to send
Data "</form></td>"                      'end of form , end of table data
Data "</tr>"                             'finish row
Data "</table>"                          'finish table
Data "<br>"                              'line

'create another table
Data "<table width={034}250{034} border = {034}9{034}>"       'basic
table with wide border
Data "<caption><h3>To send me a message type it in here and press
enter</h3></caption>"       'with a caption
Data "<tr>"                              'begin a row
Data "<td><center>"
Data "<form action={034}{034} method={034}POST{034}>"       'need a form
to post data
Data "<input type={034}text{034} name={034}TEXT2SEND{034} value=
{034}type here{034} size=70 maxlength=60>"       'text & data to send
Data "</form>"                           '
Data "</td>"                             'end of form , end of table data
Data "</tr>"                             'finish row

Data "<tr>"                              'begin a row
Data "<td><center>max 60 characters can be sent</td>"       'button &
data to send
Data "</tr>"                             'finish row
Data "<hr>"                              'a line
Data "</body>"
Data "</html>"
Data "END_OF_WEB_PAGE"                   'tell program webpage is finished
```

55 Assignment – maths in the real world

5 numbers are to be entered into memory via the 5 buttons and then displayed on the LCD. Press btn A to move between the 5 numbers. Btn B to increment the number, btn C to decrement the number. The maximum number will be 255, the minimum number will be 1. The display looks like this.

1	3			6	2			1	6	5		3	4		
4	6														

The current code is listed below, load it into your microcontroller to see how it works. Then go onto the next exercise.

```
'------------------------------------------------------------------
' 1. Title Block
' Author: B.Collis
' Date: 1 June 2005
' File Name: numberentryV0.1.bas

'------------------------------------------------------------------
' 2. Program Description:
' enters 5 numbers into variables A,B,C,D,E and display them
' 3. Hardware Features:
' LEDS
' LDR, Thermistor on ADC
' 5 switches
' LCD
' 4. Program Features
' do-loop to keep program going forever
' debounce to test switches
' if-then-endif to test variables

'------------------------------------------------------------------
' 5. Compiler Directives (these tell Bascom things about our hardware)
$crystal = 8000000
$regfile = "m8535.dat"

'------------------------------------------------------------------
' 6. Hardware Setups
' setup direction of all ports
Config Porta = Output 'LEDs on portA
Config Portb = Output 'LEDs on portB
Config Portc = Output 'LEDs on portC
Config Portd = Output 'LEDs on portD
'config inputs
Config Pina.0 = Input ' ldr
Config Pind.2 = Input 'switch A
Config Pind.3 = Input 'switch B
Config Pind.6 = Input 'switch C
Config Pinb.1 = Input 'switch D
Config Pinb.0 = Input 'switch E


'LCD
```

```
Config Lcdpin = Pin , Db4 = Portc.4 , Db5 = Portc.5 , Db6 = Portc.6 , Db7 = Portc.7 , E = Portc.1 ,
Rs = Portc.0
Config Lcd = 40 * 2 'configure lcd screen

' 7. Hardware Aliases
Led3 Alias Portd.4
Sw_c Alias Pind.2
Sw_b Alias Pind.3
Sw_a Alias Pind.6

Spkr Alias Portd.7 'refer to spkr not PORTd.7
Cursor Off
' 8. initialise ports so hardware starts correctly
Porta = &B11111100 'turns off LEDs ignores ADC inputs
Portb = &B11111111 'turns off LEDs activate pullups switches
Portc = &B11111111 'turns off LEDs
Portd = &B11111111 'turns off LEDs activate pullups switches
Cls 'clear lcd screen

'-----------------------------------------------------------------
' 9. Declare Constants
Const Btndelay = 15

'-----------------------------------------------------------------
' 10. Declare Variables
Dim State As Byte
Dim A As Byte
Dim B As Byte
Dim C As Byte
Dim D As Byte
Dim E As Byte
Dim Sum As Byte
' 11. Initialise Variables
State = 0
'-----------------------------------------------------------------

' 12. Program starts here
Cls
Do
        Debounce Sw_a , 0 , Swa_press , Sub
        Debounce Sw_b , 0 , Swb_press , Sub
        Debounce Sw_c , 0 , Swc_press , Sub
Loop
End
```

```
'-----------------------------------------------------------------
' 13. Subroutines
Disp_numbrs:
        Locate 1 , 1
        Lcd A
        Locate 1 , 5
        Lcd B
        Locate 1 , 9
        Lcd C
        Locate 1 , 13
        Lcd D
        Locate 2 , 1
        Lcd E
Return

Swa_press:
        If State < 5 Then
            Incr State
        Else
            State = 1
        End If
        Gosub Disp_numbrs
Return

Swb_press:
    Select Case State
        Case 1 : Incr A
        Case 2 : Incr B
        Case 3 : Incr C
        Case 4 : Incr D
        Case 5 : Incr E
    End Select
    Gosub Disp_numbrs
Return

Swc_press:
    Select Case State
        Case 1 : Decr A
        Case 2 : Decr B
        Case 3 : Decr C
        Case 4 : Decr D
        Case 5 : Decr E
    End Select
    Gosub Disp_numbrs
Return
```

55.1 Math sssignment - part 1

The program as given to you has a few bugs for you to fix

1. After the power is applied the lcd is blank it should display the 5 numbers.
Write your code here that fixes this

2. The display does not blank any zeros when the numbers go from 100 to 99 and 10 to 9. Fix this and explain here how you did it.

3. The numbers start at 0, they need to start at 1, fix this and explain here how you did it

4. Make the maximum number that can be entered 200, Write the code here that fixes this.

55.2 Math assignment - part 2

At the moment the user must press the button to increment or decrement the numbers one at a time. There is no auto-repeat feature included in the debounce function. Add some form of repeat feature so that the user can hold a button and after a short delay the numbers will increase/decrease until the button is released.

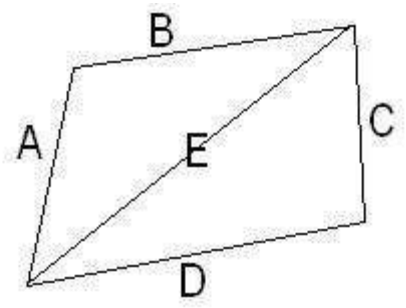
You may want to try and do this using if pin=0 then..... rather than debounce.

Make your routine as generic or portable as possible, so that it could be easily transferred to other programs.

Explain how your auto-repeat code works.

55.3 Math assignment - part 3

This program is going to be used by a groundsman to calculate the area of a piece of land so that he can work out the amount of grass seed to buy. He will use your program and pace out the 4 sides: a,b,c,d, and the diagonal e.



the formulae to work out the area of a triangle is:
s= (a+b+e)/2
Area of first triangle = sqroot(s(s-a)(s-b)(s-e))

t= (c+d+e)/2
Area of second triangle = sqroot(t(t-c)(t-d)(t-e))

1. All the calculations must be in one subroutine.
2. You will also need to dimension some temporary variables to help you, e.g.
   dim sngl1 as single, sngl2 as single, sngl3 as single
3. Bascom can only do one arithmetic equation per line so you will need to break up each equation into individual parts.

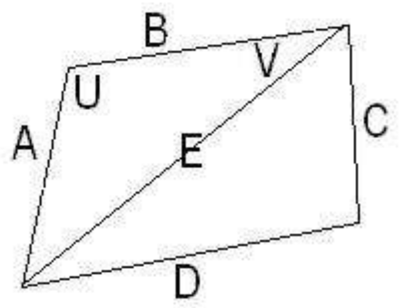
Here is half of the routine.

```
calcarea:
        s= a+b
        s=s+e
        s=s/2
        singl1=s-a
        s=s*singl1          's(s-a)
        singl2=s-b
        s=s*singl2          's(s-a)(s-b)
        singl3=s-e
        s=s*singl3          ' s(s-a)(s-b)(s-e)
        area=sqr(s)        'area of the first triangle
return
```

1. You complete the rest of the equation to work out the area of the second triangle and then work out the total area for the whole shape.
2. Modify your program to automatically update the lcd with the calculated area as the grounds man enters the data for each variable. Explain where in your code you put the changes to make this update happen all

55.4 Math assignment - part 4

When the groundsman gets back to the office, he needs to draw a plan of the area. To do this he needs the angles within the shape.



Using the cosine rule we can calculate these for him.

U is the angle opposite side E
$E^2 = A^2 + B^2 - 2AB\cos(U)$

V is the angle opposite side E
$A^2 = E^2 + B^2 – 2EB\cos(V)$

1. calculate each of the 6 angles
2. U will be in radians, convert each angle to degrees.
3. display them on the LCD

Write the code for calculating one of the angles below.

55.5 Math assignment - part 5

When the groundsman has calculated the area and angles, the data must be stored into eeprom so that it will be there when he goes back to his office.
To do this you must declare some new variables e.g. eep_a, eep_b, ... and dimension these **dim eep_A as eram byte.**
add a state and subroutine to your program which copies the variables A,B,C.etc into the corresponding eeprom variables eep_a, eep_b, eep_c etc. Write it below (you may want to change the fuselink in the AVR that causes the EEPROM to be cleared every time the AVR is reprogrammed)

add a state and subroutine to your program that reads the eeprom variables and copies them into the ram variables. Copy the subroutine here

55.6 Math assignment - part 6

Create a simple menu that allows the groundsman to select the operation to perform

- enter 5 lengths
- calculate and view the area
- calculate and view the angles
- store the values into eeprom
- read the values from eeprom

You must use a state variable to manage the program flow. Explain your code below.

55.7 Extension exercise

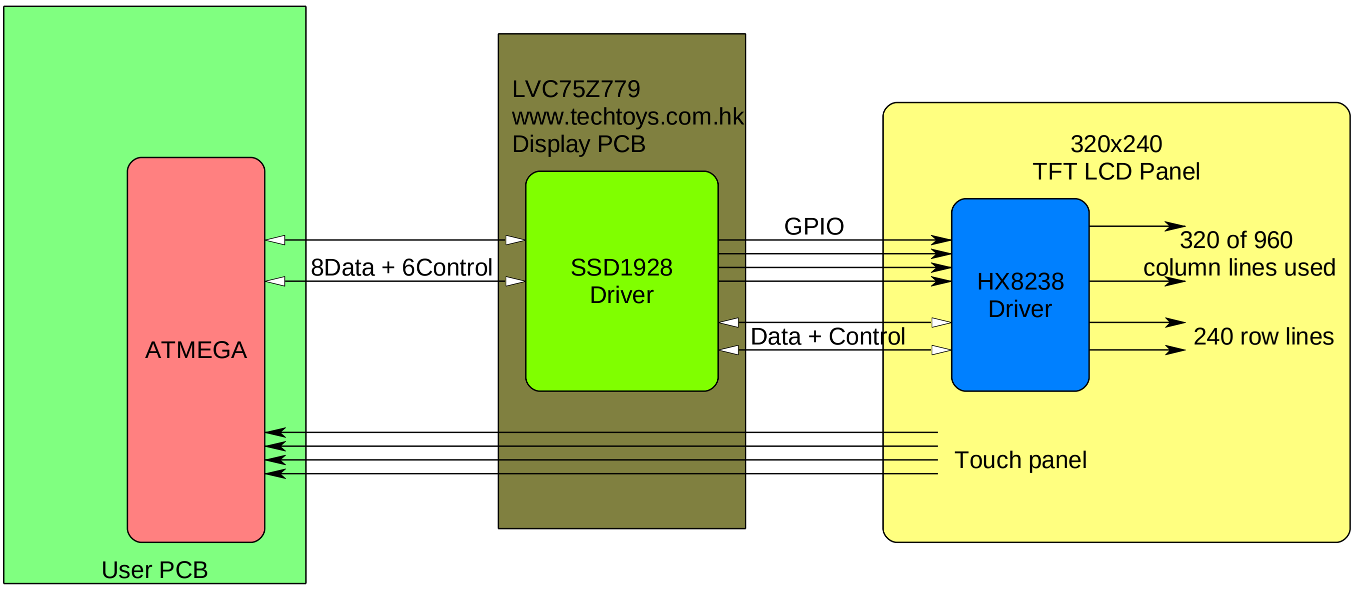
Give the groundsman the option to store multiple areas of land

56 SSD1928 based colour graphics LCD

The Display used is from techtoys.com.hk, it is not cheap but is the most suitable one I could find for student projects.

So far in this course the LCDs covered have all had driver code built into Bascom or within code libraries that hide the complexity of using the LCDs. This is not the case for the SSD, no libraries exist for driving the SSD from Bascom or even from an AVR. Research to date of these has found PIC microcontroller (not the Picaxe) libraries and faster more capable 32 bit microcontrollers being used. In this case the drivers were written for Bascom. Also note that at this time only a certain amount of SSD1928 is covered here but as students have further opportunity (and the funds) to explore it, more information will be added.

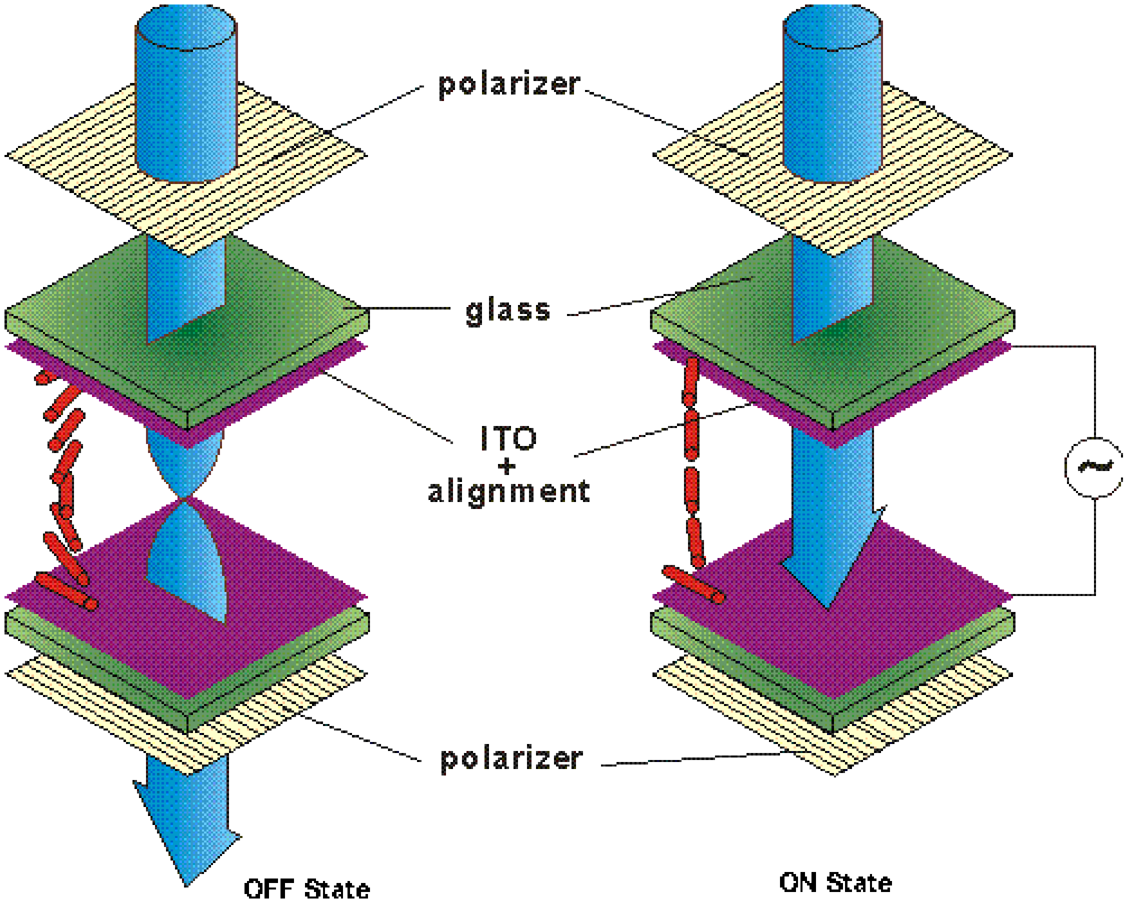
56.1 System block diagram



There are three ICs – the ATMEga, THE SSD1928 on the display PCB and the HX8238 hidden on the back of the LCD itself.

56.2 TFT LCDs

It is useful to know a little about LCDs so that you can understand the software for driving them.



An LCD requires quite specific driving signals; these are managed by ICs on the back of LCD.

To get an LCD pixel to appear requires an AC voltage to be applied to each pixel individually.

Light passes through one polarizer, then through a crystal structure which has been twisted and then it passes through the second polarizer which is at 90 degrees to the first one.

The applied voltage untwists the crystal and this blocks light from passing through the second polarizer. The darkness of the pixel can be controlled by the amount of signal applied to it. In a colour display each pixel is actually three separate sub pixels (R, G, B) which are controlled individually.

Each pixel also has its own transistor embedded in it on the glass and hence the name of the display type TFT, thin film transistor.

Having a transistor on each pixel helps switch the signals quickly reducing blurring and other issues.
These animations from 3M about how LCDs work are of interest

http://solutions.3m.com/wps/portal/3M/en_US/Vikuiti1/BrandProducts/secondary/optics101/?slideIndex=14 http://solutions.3m.com/wps/portal/3M/en_US/Vikuiti1/BrandProducts/secondary/optics101/?slideIndex=34

56.3 System memory requirements

Each pixel is driven individually one after another in each line and 1 row after another. In a 320 row by 240 line display there are (320 x 240) 76,800 pixels to be driven. Each pixel is however actually three sub pixels of red, green and blue. If we had 1 byte per colour then we would need 320 x 240 x 3 = 230,400 bytes of information for 1 screen. In our system however we only use 16 bit colour so 2 bytes per pixel (320 x 240 x 2 = 153,600 bytes). The SSD1928 has a 256kByte RAM for storing the LCD panel data which therefore leaves us spare ram.

56.4 System speed

Data must be sent from the SSD1928 to the HX8238 on the LCD many times per second otherwise the LCD image will fade (LCD screens are refreshed at rates of 50 or more times per second). The rate of this particular system setup is 52 screen refreshes per second. You might think then that we need to send 320 x 240 x 2 x 52 = 7,987,200 bytes every second , but it is actually more than this due to the timing requirements of the ICs – more about this later.

To achieve all this high rate of timing inside the SSD1928 is a special oscillator circuit called a PLL (phase locked loop) which generates the main internal clock signal of 72MHz from a 4MHz crystal on the PCB.

56.5 SSD and HX ICs

The SSD1928 has 128 pins in what is called a LQFP (low profile quad flat package) and can either be driven with data which is 16bits in parallel or with 8 bit parallel data or even serially. The HX8238 has 1,521 pins(!!); there are 320 columns each of which has a separate red, green and blue line, so 960 connections are needed and another 240 pins are needed for the 240 rows. It comes as a COG (chip on glass) not as a usual package with pins but as a ‘bump’ package which has tiny pads underneath; it is also very small; just 22.18mm long x 0.96mm wide and only 0.015mm high!

56.6 Colour capability

Although the SSD1928 is capable of 16M colours (8bits each of red, green and blue, 255x255x255=16,581,375), we won’t actually get 16M because the HX is only capable of 262k colours (6bits of red, green and blue, 64x64x64=262,144 this is 18bit colour). Note that all 24 bits of colour from the SSD are connected to the 24 colour input pins of the HX, but the HX only uses the lower 6 bits of each colour.

18bit colour is of little use as we send data in byte size chunks, so in our software we are only using 16 bits (2 bytes) to store colour information which gives us 65,536 colours. so our data will take up 320x240x2 =152,600 bytes of the ram. The16 bits are arranged as: RRRRRRGGGGGGBBBBB (5 bits of red, 6 bits of green and 5 bits of blue)

56.7 SSD1928 and HX8238 control requirements

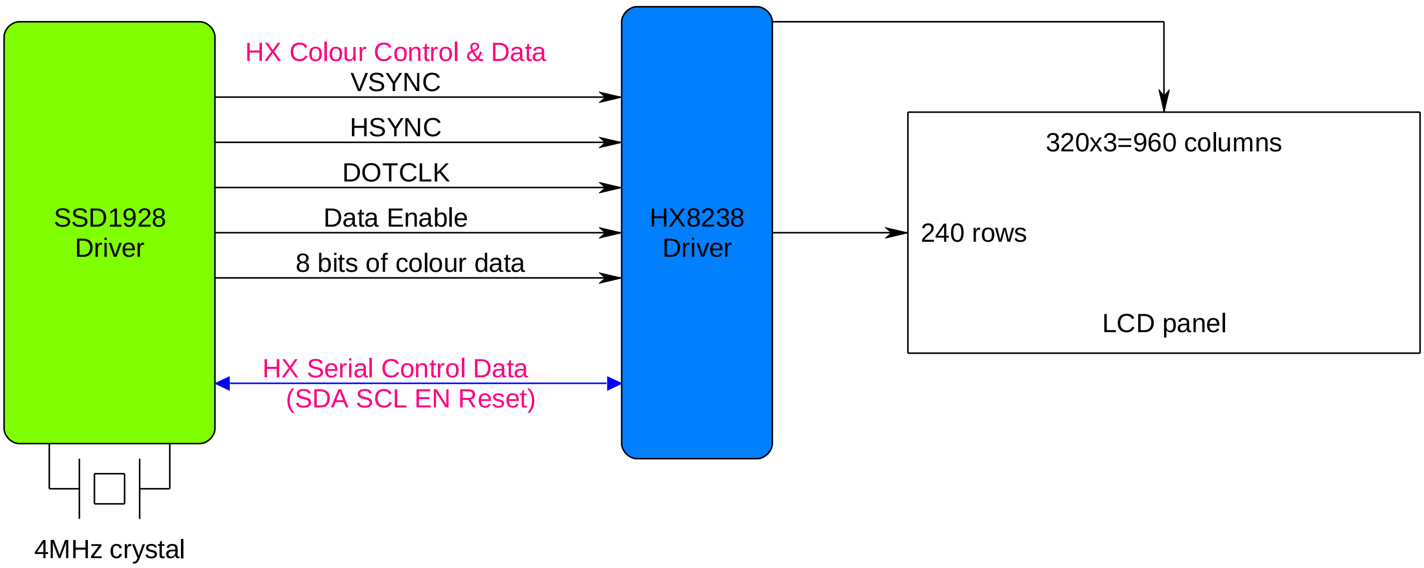
Referring to the previous block diagram the ATMega controls the SSD through 8 data and 6 control lines, over these lines travels both information to control the SSD and HX chips as well as the colour information for the LCD panel.

In the extended block diagram below the SSD to HX connection is shown, it has 2 separate sets of interface pins in the 54 pin flexi circuit; the first is for the colour control and data signals, the second for control information for the HX chip. These are kept separate because the HX chip requires precise timing for the colour data and timing control signals so these cannot be halted to send control information to HX chip.

The colour data and control lines include 8 parallel data lines for colour information (even though we send 16 bits of colour data), the two synchronization pulses (VSync and HSync), the clock signal and a data enable pulse that is high only when actual colour information is present.

The SSD1928 has 5 GPIO (general purpose IO) lines, 4 of which are connected to the 4 serial command lines of the HX8238 (CSB, SCK, SDI and SDO). These 4 serial lines are used to send commands to the HX8238 to tell it about the LCD panel connected to it and information about the timing of the Sync pulses.

The SSD1928 control signals to the HX8238 are fairly complex and normally you wouldn’t have to know much about them however we have to write software that sets up the SSD to generate the signals and more software to set up the HX8238 to be able to interpret the signals being sent to it by the SSD1928. Interestingly although an LCD is technically different to the old CRT screen the terms and signal timings used here are very similar.



56.8 SSD1928 Software

The software is broken up into a main program and a number of routines in other included files. This reduces the over all size of the main program and helps to logically structure code for others to understand. The only function of the main program then is to call the routinwes that set up the LCD and then draw some text onto the LCD.

Before subroutines in other files can be used BASCOM requires that they must be declared. An easy way to do this is to have two files setup one with the saubroutines in it (it ends with .bas) and one with the declarations in it (these end with the extension .h), so in the directory there are both .bas and .h files with the same name.

```
'*********************************************************
'SSD1928 and HX8238 software - most work done by Ethan O.(School
student)
'Text routines by Abhilash K.(school student)
'Debugging, tidying up, and commenting by Bill Collis(teacher of
above 2!)
'*********************************************************
$regfile = "m644def.dat"
$crystal = 20000000
$hwstack = 256
$swstack = 80
$framesize = 160

'****************************************
'declarations for routines are in the header files
$include "SSD1928_Register_Routines.h"
$include "SSD1928_GPIO_Routines.h"
$include "SSD1928_Hardware_Setup_Routines.h"
$include "SSD1928_Window_Control_Routines.h"
$include "SSD1928_Memory_Routines.h"
$include "SSD1928_Simple_Graphics_Routines.h"
$include "SSD1928_Text_Routines.h"
$include "SSD1928_Color_Defines.h"

Config Porta = Output
Config Portb = Output
Config Portc = Output
Config Portd = Output

'Hardware Aliases
'****************************************
'configure 8 bit dataport settings here
Datout Alias Portc
Datin Alias Pinc
Datdir Alias Ddrc
Ctrlout Alias Porta                     'rd, wr, cs, rs, rst, bl,
slp, xx
'configure control lines here
Rd Alias Porta.7                        'read active low
Wr Alias Porta.6                        'write active low
Cs Alias Porta.5                        'chip select - falling edge
latch
Rs Alias Porta.4                        'data/#command
Rst Alias Porta.3                       'active low  0=reset/halt
Bl Alias Porta.2                        'active high  1=on
Slp Alias Porta.1                       'pll 1=disable  0=enable



Dim Forecolor As Word
Dim Backcolor As Word
Dim _bit As Bit
Dim _byte As Byte
Dim _byte2 As Byte
Dim _word As Word
Dim _long As Long
```

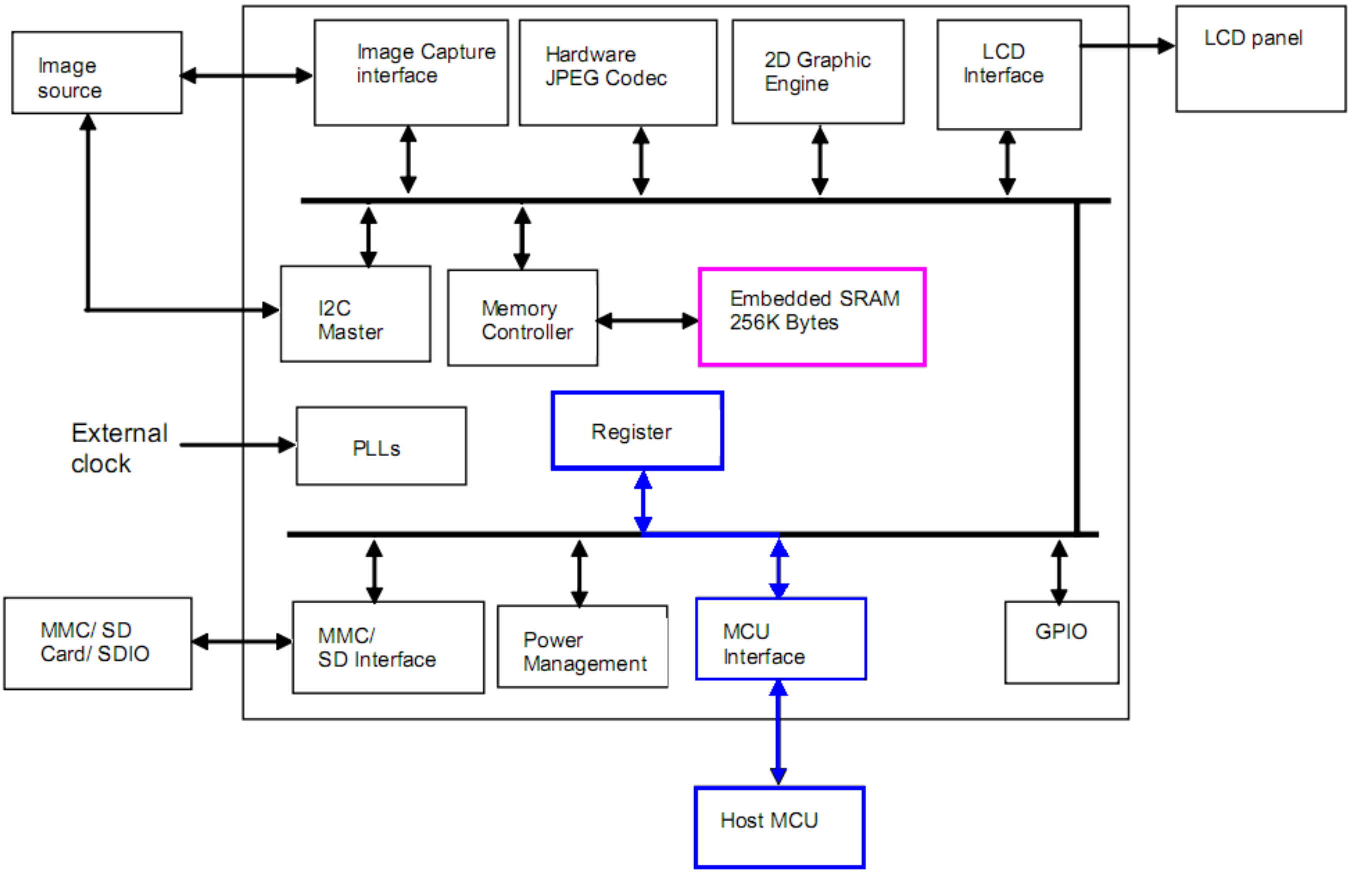
```
Dim _pageAs Byte                              '0=main window, 1=floating
window
Dim _line_mem_pitch As Long                    '320 for main wnd or width of
float wnd

'**************************************************************
Const Screen_width = 320
Const Screen_height = 240
Const Page_mem_size = 153600
'screen width * screen height * 2bytes per pixel(16bit)
Const Line_mem_pitch = 320

_page = 0                                      'mem for main window

Dim Mx As Word
Dim My As Word
Dim Mdy As Integer
Dim Mdx As Integer
Dim _count As Word
Dim Strval As String * 10
Dim Byteval As Byte
Dim I As Byte , J As Word , K As Word

'**************************************************************
'Program starts here
Call Resetdevice()                             'setup PLL, MX8238, memory
areas
Call Ssd1928_mainwndenable(0)                  ' turn lcd off before
configuring

'Ssd1928_mainwndinit(Startaddr, Linewidth, Bitsperpixel, Orientation,
Rgb/yuv)
Call Ssd1928_mainwndinit(0 , 320 , 16 , 0 , 1)
Call Ssd1928_focuswnd(0)                       'we are writing to main not
floating wind

Backcolor = &H0000                             'use hex colour
Call Cleardevice()                             'fill the screen with
backcolor
Call Ssd1928_mainwndenable(1)                  'turn on the lcd
Wait 1

Backcolor = &B1110011000000100                 'use 16 bit binary colour
Call Cleardevice()                             'fill the screen with new
backcolor
Wait 1

Backcolor = Lightblue                          'use predefined colours from
.h file
Call Cleardevice()

Forecolor = Blue
Textpos 0 , 0
Text8 "Some size 8 text, "
Forecolor = Brightcyan
Text8 "this is  a great display!!!"           'text wrapping
```

```
Textpos 0 , 20
Forecolor = Magenta
Backcolor = Lightgreen
Text16 "size 16 font"

Textpos 0 , 40
Forecolor = White
Backcolor = Lightblue
Verdana "size 16 true type font verdana!"

Backcolor = Black
Forecolor = Red
For I = 0 To 50 Step 5
   J = I + 250
   K = 300 - I
   Drawline J , 180 , K , 220 , Forecolor
Next

Drawbox 10 , 200 , 60 , 233 , Blue
Drawbox 11 , 201 , 59 , 231 , Blue
Fillbox 15 , 205 , 55 , 228 , Red

Forecolor = Black
Backcolor = Red
Do
    For I = 0 To 50
        Strval = Str(i)     'text routines only display text so
convert
        Strval = Format(strval , "  ")   '2 spaces means 2 digits
displayed
        Textpos 20 , 210
        Call Text16(strval )
        Waitms 500
    Next
Loop
End                                        'end program

'********************************************************************
$include "SSD1928_Text_Routines.bas"    'various text routines
$include "SSD1928_GPIO_Routines.bas"    'talk to MX8238
$include "SSD1928_Memory_Routines.bas"  'data memory
$include "SSD1928_Register_Routines.bas"       'control registers
$include "SSD1928_Window_Control_Routines.bas"       'window size and
enable
$include "SSD1928_Hardware_Setup_Routines.bas"       'SSD & LCD setup
$include "SSD1928_Simple_Graphics_Routines.bas"       'putpixel,
drawline, rgb
'********************************************************************
$include Verdana.font       'modify font routines & remove to save
space
$include Font8x8.font       'modify font routines & remove to save
space
$include Font16x16.font     'modify font routines & remove to save
space
```

56.9 SSD1928 microcontroller hardware interface

The SSD1928 is a very complex device with many interfaces and features. To use these features requires the developer to become familiar with many of the thousand plus registers within the SSD, these control everything that the SSD does.

Figure 4-1 : SSD1928 Block Diagram



Before we can access the SSD registers however we need to configure the SSD to micro interface and we have to setup some jumpers on the SSD interface board that tell the SSD the configuration of the data we are going to be sending to it. These are the 4 switches labeled CNF3, CNF2, CNF1 & CNF0 on the board and they should be set to 0011. This setting indicates to the SSD to expect 8 bits at a time in indirect mode. Indirect mode means sending 3 bytes of address and then the required bytes of data over a single 8 bit data bus.

You can also configure and use 16 bit indirect mode as well. Direct mode is also configurable where the address and data are on separate buses. However this development board does not give you access to the address bus so you cannot use direct modes.

56.10 Accessing SSD control registers

The first set of subroutines we will need will allow all of our other routines to write to and read from the control registers in the SSD1928. Addressing a register requires 3 bytes of address to be sent to the SSD. Note that this is more than the address range of the actual registers in the SSD which could be addressed using 2 bytes; the addressing however is the same as that used to access the 256kbyte SRAM in the SSD1928 for pixel colour data which requires 19 bits of address [A18:A0].

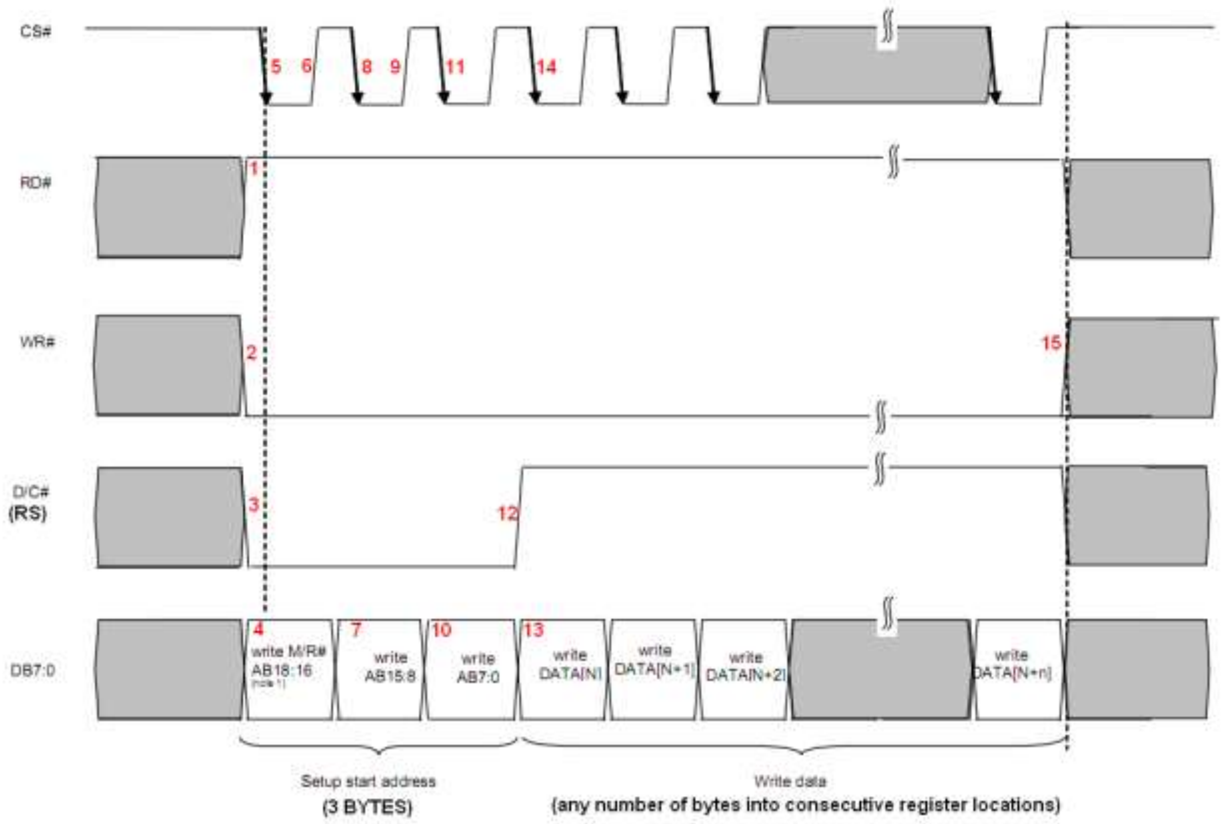
To tell the difference between an address of a register and an address in memory the SSD requires the first bit of our three bytes of address to be a 1 for memory and a 0 for a register. e.g. &B1000 0000 0000 0000 1111 1111 is memory and &B0000 0000 0000 0000 1111 1111 is a register address.

We will need routines that can read and write 8, 16 and 32 bit data registers.
The routines we will write are:

- Read a byte from a register **getreg(word_addr)**
- Write a byte into a register **setregb(word_addr, byte_data)**
- Read a word (2 bytes) from a register **getregw(word_addr)**
- Write a word (2 bytes) into a register **setregw(word_addr, word_data)**
- Read a long (4 bytes) from a register **getregl(word_addr)**
- Write a long (4 bytes) into a register **setregl(word_addr, long_data)**

As an example the sequence for writing one word (two bytes) of data into two consecutive registers is taken from the timing diagram in the SSD1928 datasheet. The three bytes of address are sent first and then two bytes of data are sent. The first byte of data will go into the register we set the address of; the second byte will go into the next register.

1. Read must be high – this is the default or usual state but we set it anyway
2. Write must be low as we are going to be writing into a register
3. RS(DC) – register select or DataCommand must be low (we are sending the address of the register or command
4. Setup first of three address bytes – bit 7 of the first byte must be 0 to tell the SSD that the address we want to access is a register and not a memory address (note that steps 1 to 4 can happen in any order)
5. Take CS low – this is the important action that the SSD is waiting for to trigger it to do something, the dotted line on the diagram tells us that the previous steps must all happen before the negative edge of CS.
6. Return CS high
7. Setup the second byte of the address
8. Take CS low – triggering the SSD to know that another byte of address is on the data bus
9. Return CS high
10. Setup third byte of the address
11. Take CS low then high again.
12. Take RS(DC) high, this means we have finished sending the address and will now send the data
13. Setup the first byte of the data
14. Take CS low then high again.
15. As we have finished sending the data we return the write line to its default state which is high



```
$nocompile
'***********************************************************************
****
```

56.11 SSD1928_Register_routines.bas

```
' allow reading and writing of the control registers in the SSD1928
'these routines have not been streamlined and are therefore
reasonably slow
'not a big issue though as we dont use them a lot.
'***********************************************************************
****'set 1 byte
Sub Setregb(byval Index As Word , Byval Value As Byte)
   Local L As Word
   L = Index

   Datdir = &HFF                         'set as output

   Rs = 0                                'first send address to SSD
   Wr = 0                                '0 means we are writing
   Rd = 1                                '1 means we are not reading

   Datout = 0                            'bit7 = 0 so writing to
register
   Cs = 0
   Cs = 1

   Rotate L , Right , 8                  'send upper byte
   Datout = L
   Cs = 0
   Cs = 1

   Rotate L , Left , 8                   'send lower byte
   Datout = L
   Cs = 0
   Cs = 1

   Rs = 1                                'next send data byte to SSD
   Datout = Value
   Cs = 0
   Cs = 1

   Wr = 1
End Sub
```

```
'***********************************************************************
****
'get a single byte from a register
Function Getregb(byval Index As Word) As Byte
   Local W As Word
   W = Index

   Datdir = &HFF

   Rs = 0                                       'first send address to SSD
   Wr = 0
   Rd = 1

   Datout = 0                                   'bit7 = 0 so writing to
register
   Cs = 0
   Cs = 1

   Rotate W , Right , 8
   Datout = W                                   'write AB15:8
   Cs = 0
   Cs = 1

   Rotate W , Left , 8
   Datout = W                                   'write AB7:0
   Cs = 0
   Cs = 1

   Datdir = 0                                   'set as input to recieve data

   Rd = 0                                       'setup for read command
   Rs = 1
   Wr = 1

   Cs = 0                                       'dummy read
   Cs = 1

   Cs = 0                                       'read real strobe
   Cs = 1

   Getregb = Datin                     'get the data which the LCD sends us

   Datdir = &HFF                       'reset port direction for write action
   Rd = 1
End Function
```

```
'*********************************************************************
'read 1 word from 2 consecutive registers
'Checked By Readin &H0000 which correctly returns 10000000 00101000
Function Getregw(byval Index As Word) As Word
   Local W As Word , B As Word
   W = Index

   Datdir = &HFF                                  'output
   Rs = 0                                         'first send address to SSD
   Wr = 0
   Rd = 1

   Datout = 0                                     'M/R = 0  => register
   Cs = 0                                         'write M/R
   Cs = 1

   Rotate W , Right , 8
   Datout = W
   Cs = 0                                         'write AB15:8
   Cs = 1

   Rotate W , Left , 8
   Datout = W
   Cs = 0                                         'write AB7:0
   Cs = 1

   Datdir = 0                                     'set as input to get data

   Rs = 1
   Wr = 1
   Rd = 0

   Cs = 0                                          'dummy read
   Cs = 1

   Cs = 0                                          'real read
   Cs = 1
   B = Datin                                      'get data
   'second read
   Cs = 0
   Cs = 1
   W = 0                                          'word going to be written to
L
   W = Datin

   Rotate W , Left , 8
   W = W Or B

   Getregw = W
   Datdir = &HFF                                  'return to output
   Rd = 1
   Rs = 0
End Function
```

```
'*********************************************************************
' write a word to 2 consecutive registers
'this is inefficient as setregb is called twice which sets up the
address
' each call, it cam be streamlined by removing the second call, this
wont
'increase progam speed though as it is hardly used.
Sub Setregw(byval Index As Word , Byval Value As Word)
   Local Byte2write As Byte
   Local W As Word

   Byte2write = Value And &HFF
   Call Setregb(index , Byte2write)      'write lower byte

   W = Value
   Rotate W , Right , 8                  'get most significant byte
   Byte2write = W And &HFF
   Index = Index + 1                     'next register
   Call Setregb(index , Byte2write)      'write upper byte
End Sub

'*********************************************************************
'write a long - 4 bytes into 4 consecutive registers
'very inefficient as it calls setregw twice which calls setregb 4
times!
Sub Setregl(byval Index As Word , Byval Value As Long)
   Local Word2write As Word
   Local L As Long
   L = Value

   Word2write = Value And &HFFFF
   Call Setregw(index , Word2write)      'write lower word for
register 'index'

   Rotate Value , Right , 16
   Word2write = Value And &HFFFF
   Index = Index + 2
   Call Setregw(index , Word2write)      'write upper word for
register 'index+2'
End Sub

'*********************************************************************
Function Getregl(byval Index As Word)
   Local W As Word
   Local L As Long

   W = Index + 2
   W = Getregw(w)
   L = W
   Shift L , Left , 16
   W = Getregw(index)
   L = L Or W

   Getregl = L
End Function
'*********************************************************************
```

56.12 Accessing the HX8238.

The only way to control the HX is to send data serially from the SSD. These routines give us access to register &HAC in the SSD which controls the 5 GPIO lines, 4 of which are connected to the HX.

```
'*********************************************************************
```

56.13 SSD1928_GPIO_routines.bas

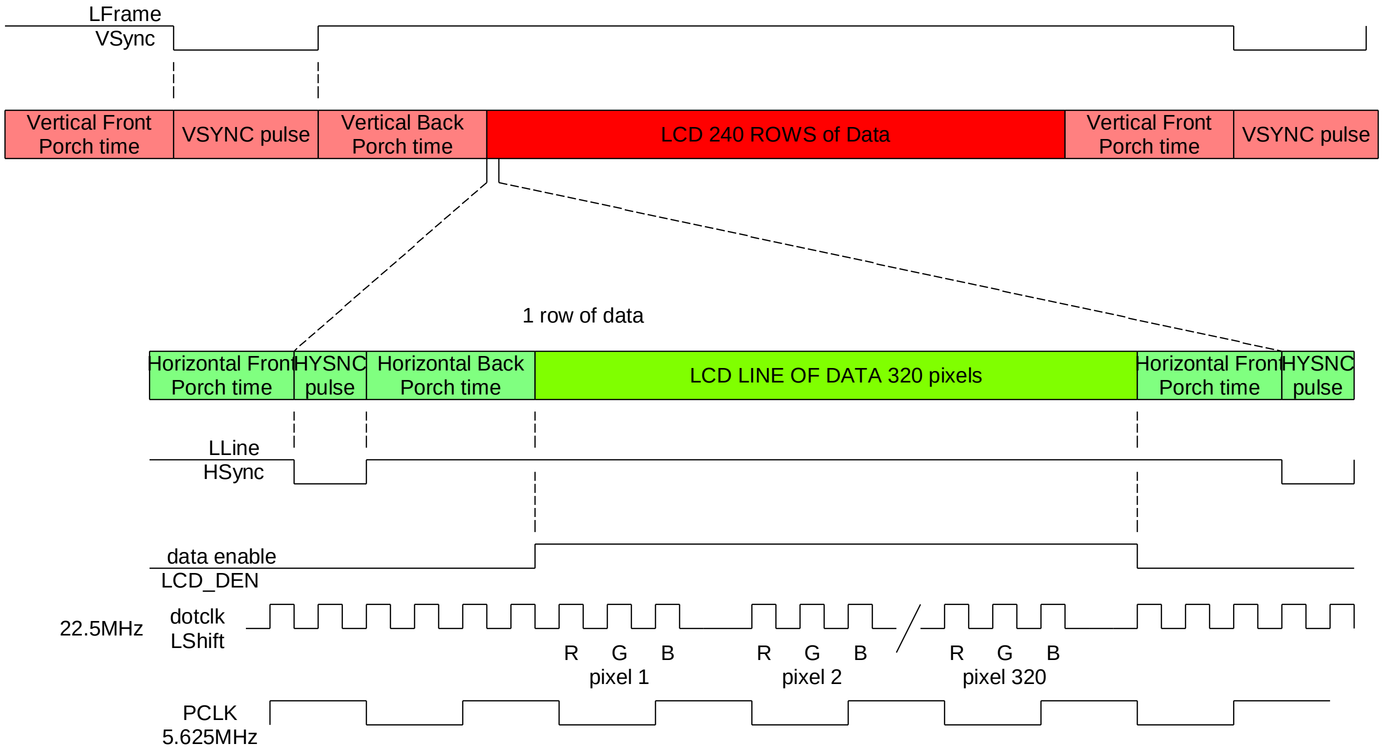
```
'*********************************************************************
'these routines manage the communication between the SSD1928 on the
PCB
' and the HX8238 on the LCD panel itself
$nocompile
'*********************************************************************
'4 lines are used to communicate to the HX from the SSD
'we must use these routines to configure specific registers in the HX
'the state of these 4 lines is controlled by the GPIO status/ctl
register &HAC
'_gpiostatus keeps track of which bits are set or reset in the
register
'so it must be a global variable

' SSD_Gpio3 = HX_Lcd_reset = bit 3
' SSD_Gpio2 = HX_Lcd_spena = bit 2
' SSD_Gpio1 = HX_Lcd_spclk = bit 1
' SSD_Gpio0 = HX_Lcd_spdat = bit 0

Dim _gpiostatus As Byte
_gpiostatus = 0                         'initially no bits are set
'*********************************************************************
Sub Gpio_spreset(byval State As Byte)
   If State = 1 Then
      _gpiostatus.3 = 1
   Else
      _gpiostatus.3 = 0
   End If
   Call Setregb(&Hac , _gpiostatus)
End Sub
'*********************************************************************
Sub Gpio_spena(byval State As Byte)
   If State = 1 Then
      _gpiostatus.2 = 1
   Else
      _gpiostatus.2 = 0
   End If
   Call Setregb(&Hac , _gpiostatus)
End Sub
'*********************************************************************
Sub Gpio_spclk(byval State As Byte)
   If State = 1 Then
      _gpiostatus.1 = 1
   Else
      _gpiostatus.1 = 0
   End If
   Call Setregb(&Hac , _gpiostatus)
End Sub
```

```
'***********************************************************************
Sub Gpio_spdat(byval State As Byte)
   If State = 1 Then
      _gpiostatus.0 = 1
   Else
      _gpiostatus.0 = 0
   End If
   Call Setregb(&Hac , _gpiostatus)
End Sub

'***********************************************************************
Sub Spi_write(byval B As Byte)
   Local Bit_cntr As Byte
   Local Temp As Byte
   'send the 8 bits of data out to spdat, toggling clk after each bit
   For Bit_cntr = 0 To 7
      Temp = B And &H80
      If Temp = 128 Then
         Call Gpio_spdat(1)
      Else
         Call Gpio_spdat(0)
      End If
      Call Gpio_spclk(0)
      Call Gpio_spclk(1)
      Shift B , Left , 1
   Next
End Sub

'***********************************************************************
Sub Spi_setreg(byval Reg As Byte , Byval Cmd As Word)
   Local B As Byte
   Local W As Word

   Call Gpio_spena(0)
   Call Spi_write(&H70)
   Call Spi_write(&H00)
   Call Spi_write(reg)
   Call Gpio_spena(1)

   Call Gpio_spena(0)
   Call Spi_write(&H72)
   W = Cmd
   Rotate W , Right , 8
   B = W
   Call Spi_write(b)
   Rotate W , Right , 8
   B = W
   Call Spi_write(b)

   Call Gpio_spena(1)
End Sub
'***********************************************************************
```

56.14 LCD timing signals

Now we know how to write to registers we need to figure out exactly what we should write into those registers, this requires quite a lot of understanding about how an LCD is setup.

- Data is sent to the HX one pixel at a time in rows to make up one full screen of colour information. The data cannot be sent asynchronously (without extra timing pulses) as the HX must know when each new line and when each new screen starts so the synchronizing signals HSync and VSync are sent as well.
- The data cannot be sent line by line continuously as the HX must have time between lines and between frames to set up its internal electronic circuits.
- In the timing diagram below the bright green area shows the time for one line of 320 pixels to be sent sequentially (one after the other) to the HX chip; note that one visible line is 320 pixels and they are all sent during the bright green, the rest of the green time is used as a gap between each line. In a CRT (cathode ray tube)monitor or CRT TV a delay is required after sending each line of information because the cathode ray (electron beam) had to be repositioned to the beginning of the next line (flyback time) or to the top of the screen. In an LCD everything is controlled by a single clock rate, so at the end of a line and before the beginning of the next line a number of clock pulses are needed to allow time for the internal electronics of the HX to reset the line counter inside the HX to the left edge of the panel (the front and back horizontal porch times).



- The HX must know when a new line begins, and this is signaled by the HSync (horizontal synchronization) pulse, which goes low for a short period of time. Its positive edge is the reference for the horizontal or line timing
- The HX also must know when the colour data is present and this is signaled by the data or LCD enable line being high.
- The SSD must send each row of data one after the other, and in the upper red areas of the timing diagram the darkest area is the 240 visible lines of data. The total red areas of the timing diagram show all the timing for one complete frame of the LCD. A frame is a full screen of data sent line by line to the LCD panel. Note that there is also time at the end of

a frame (whole screen of data) and before the next frame (the light red areas), again to setup the internal electronics to reset the row counter to the top line of the LCD panel (front and back horizontal porches).

- The DOTCLK comes from the SSD and is the clock signal for the HX to time all LCD events, it is 22.5MHz, however note that during the visible line time (when LCD_DEN is high) every 4th clock cycle is dropped.

56.15 HX setups

To use the device you don't have to understand how the LCD and all the code works however understanding the code and the datasheet is important so that students can explore other features of the device.

- Register &H01, sets up some basic parameters for the connection of the LCD, e.g. changing &H7300 to &H3300 changes bit RL and consequently mirrors the display. It also changes the order of RGB so colours change as well; it would seem we can fix this by changing the BGR bit which should reverse the colours but for some reason it doesn't work. You can rotate the display by changing both TB and RL bits (but again the colours are changed and BGR doesn't seem to affect the colours for some reason).

- Register &H0A of the HX alters the brightness and contrast settings. From the datasheet: the brightness default is &H40 which is a brightness level of 0, the range of brightness is from 7F (+126) to 00 (-128); the contrast default is &H08 which is a contrast level of 1, the range is from &H0(0) to &H1F(3.875).

- Register &H0F changes the starting line of the LCD thus allowing you to roll the display vertically.

- Of the HX setups the most important seem to be registers H16 and H17 in the HX.

H16 sets up the HX to know that there will be 320 pixels of horizontal data (see page 40 of the HX datasheet-although it calls this register the horizontal porch it is not).

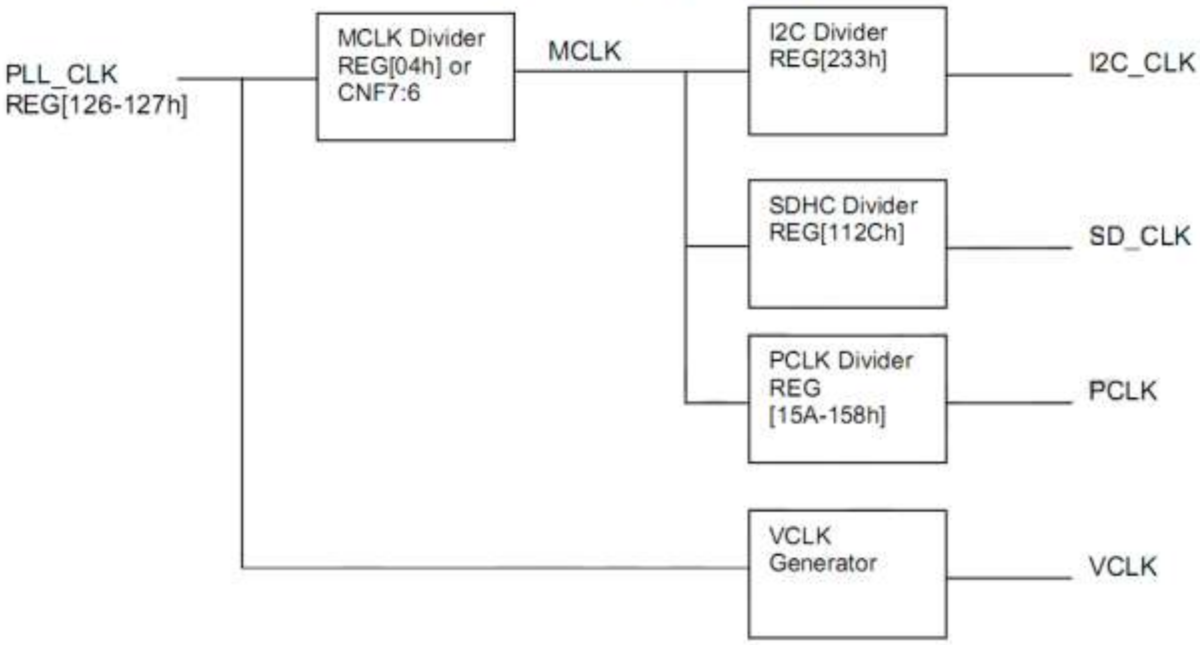
H17 sets up two vital aspects of the synchronization, the vertical and horizontal porch timings (pages 40-42 of the HX datasheet). We set it to the valus &H2122 = &B 0010 0001 0010 0010

The first 2 bits are ignored.
The next 7 bits 1000010 are the horizontal back porch time; i.e. the time after HSync goes high and before the next line starts. This is 66 in decimal and is measured in pixel clocks.
The next 7 bits 0100010 are the vertical back porch time; the time after VSync goes high and before the next frame starts. This is 34 in decimal and is measured in lines.

56.16 SSD setups

There are a number of clocks to setup in the SSD to generate all the timing signals; from page 8 of the SSD application note is the diagram below.

Figure 2-1: Clock configuration

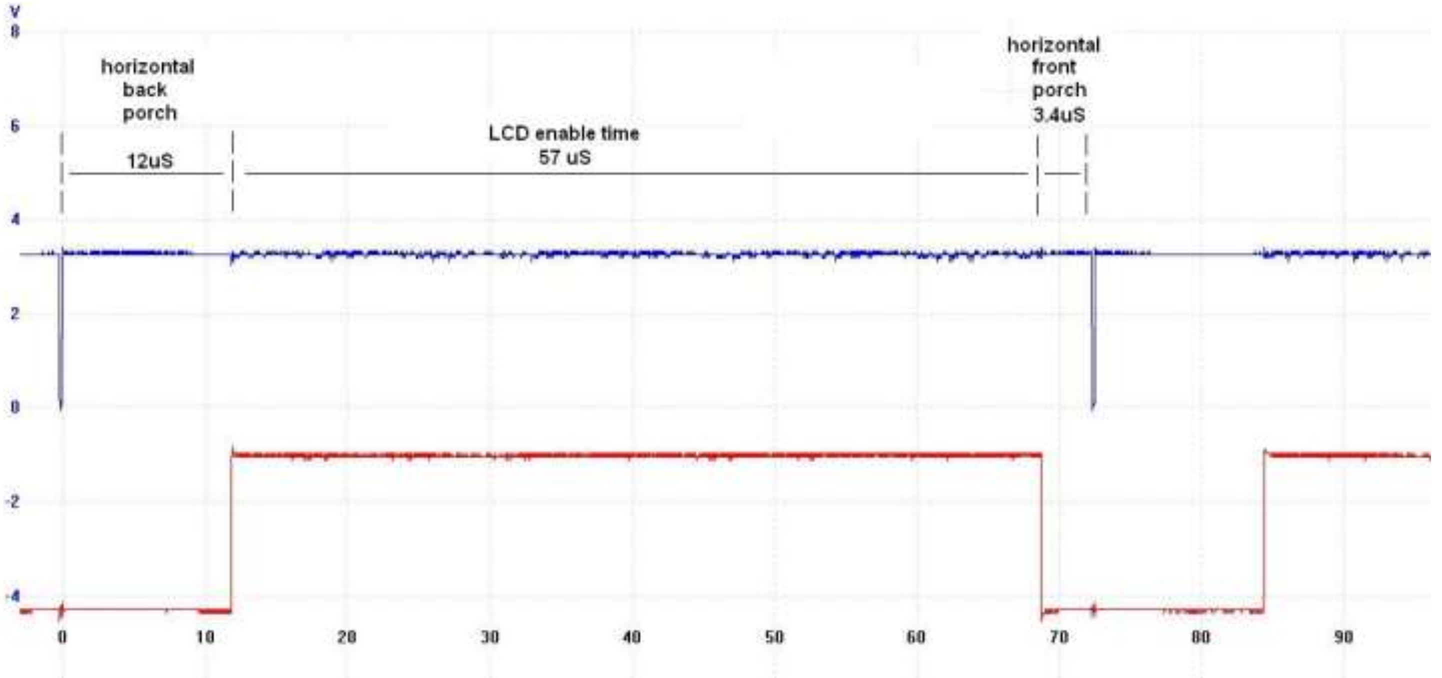


- The SSD generates the PLL clock of 72MHz from the 4MHz crystal using a phase locked loop. The M and N values for the PLL are set up in registers &H126 and &H127 in the SSD and described on page 8 of the SSD1928A application note on pages 8 and 9. A PLL is a fancy digital divider network that outputs a higher frequency than the one coming into it. The output is 4MHZ x M value / N value = 4 x 180/10 = 72MHz.
- The next stage in the clock sequence is MCLK, Register &H04 is 0 so MCLK = PLL = 72MHz
- PCLK is the pixel clock or frequency = MCLK x (registers &H15A, 159, 158 +1)/ 2^{20}= 72 x 81920 / 1048576 = 5.625MHz.
- The registers in the PCLK calculation are also used in the important LShift (dotclk) calculation. LShift (dotclk) = MCLK x (PCLK ratio+1) / 2^{18} = 72 x 81920 / 262144 = 22.5MHz.
- Note that dotclk and PCLK are not the same, dotclk is 4 times the freq of PCLK; this is necessary because each pixel is actually 3 sub pixels (R-G-B), so for 1 count of PCLK at least three cycles of the dotclk must occur. To achieve three cycles the SSD drops one cycle in every four during the actual visible time as shown in the HSYNC and VSYNC timing diagram earlier.

SSD Register &H10 is an important setup:
It sets the panel type (CSTN delta), colour, 8 bit data width and serial TFT.

56.17 SSD line / HSync timing

The actual timing for one line of data from the SSD to the HX as displayed on an oscilloscope is shown below. The HSync pulse is very narrow, just 180nSec (0.18uSec) and a full line of data takes about 72.5uS to send, of which 57uS is the time taken to send the 320 pixels (960 RGB sub pixels) of data, 12uS is the back porch (blank time after HSync pulse before data) and 3.4uS is the front porch (blank time after data before HSync pulse).



All timing for lines is taken from a reference point and HPS –horizontal pulse start position (registers &H22 and &H23 + 1) is the time in pixels from this point to the negative edge of the horizontal sync pulse. The diagram on page 24 of the SSD app note shows the timings relative to this point. In our case the registers are set to 0 so HPS=1 cycle of PCLK (0.178uS) so all line timing is relative to 0.178uS before the negative edge of HSync. Note that if HPS is set to more than 0 then it will impact on the other timings as well.

HT = Horizontal total and is set by registers &H12 and &H13, HT= 408 pixels (periods of PCLK). This is the complete length of time to send 1 full line of colour information to the display. PCLK period is 1/5.625MHZ=0.178uS so HT is set to 0.178 x 408 = 72.53uS, the scope shows a period of 72.5uS.

HDPW = horizontal display pulse width = LLine pulse width. Register &H20 also sets up HSync to be a negative pulse, the register is set to a value of 0 which sets up a negative pulse of 1 PCLK duration = 0.178uS which was the measured time on the oscilloscope.

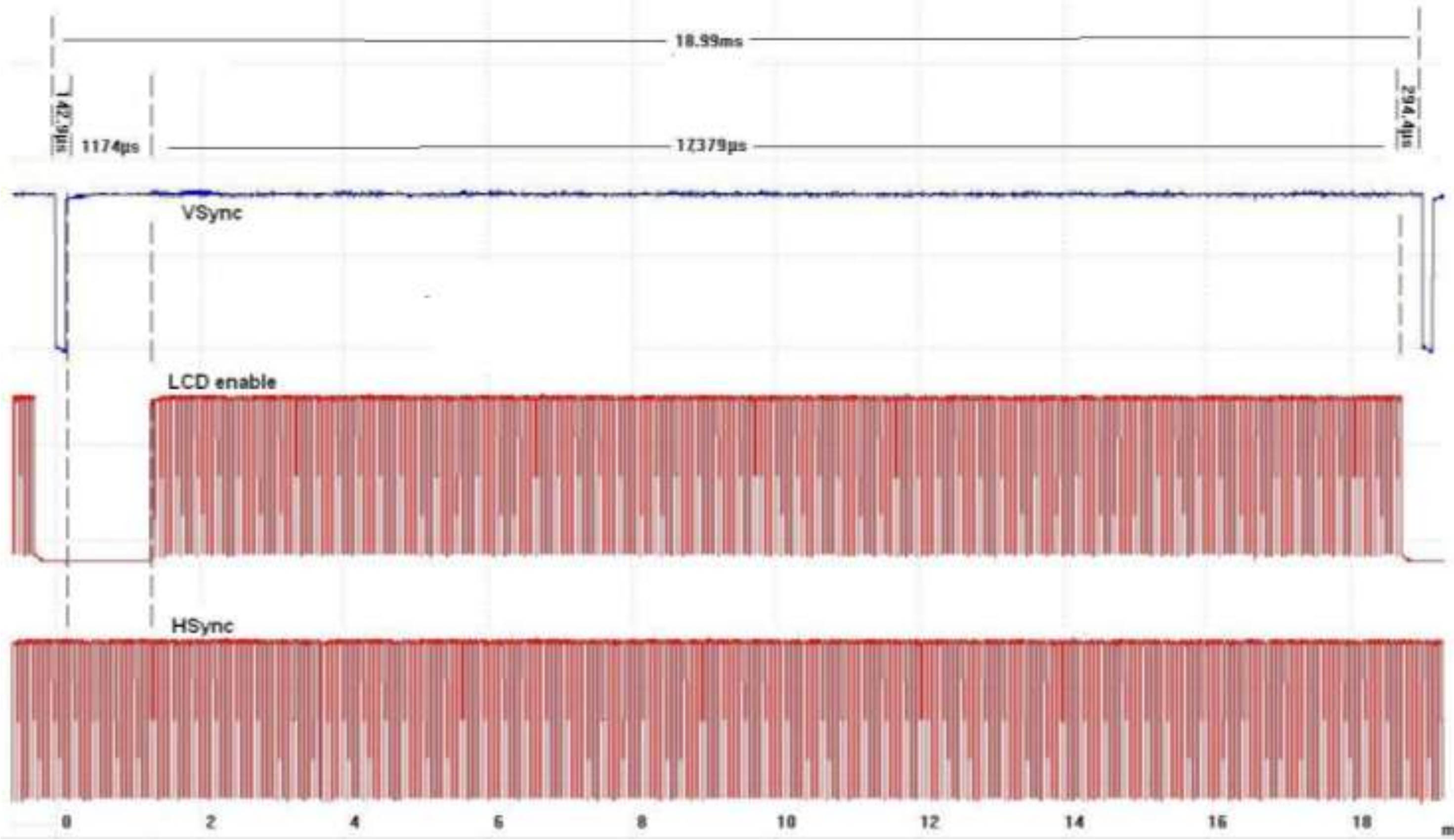
HDPS = horizontal display period start position and is the back porch timing + HPW + HPS. Reg&H17 and reg&H16 are set to &H44 = 68 pixels = 68 x 0.178uS = 12uS.

HDP = horizontal display period and is set by register &H14, HDP = (&H27+1) x 8 = 320 pixels = 320 x 0.178uS = 57uS

Having set HT, HPS, HDPW, HDPS and HDP, the remaining time is the front porch time.

56.18 SSD row / VSync/ frame timing

Having generated the HSync pulses the next step is to setup the VSync or row timings. The oscilloscope picture below shows the measured values. Note that a complete frame takes 18.99mS to send, so the LCD refresh rate is 1/0.01899 = 52 frames per second.
In the previous diagram we can see that it takes about 72.5uS to send 1 row of data, so for 240 rows it should take 240x72.5uS = 17,400uS, on the scope it was measured as 17379uS.
In the lower part of the diagram note the HSync pulse is continuously sent even during the times when there is no pixel data.



All timing for frames is taken from a reference point and VPS –vertical pulse start position is set in lines (registers &H31 and &H30) and pixels (registers &H31 and &H30) and is the time from this point to the negative edge of the vertical sync pulse. These registers are all set to 0, so all timing can be taken from the negative edge of VSync.

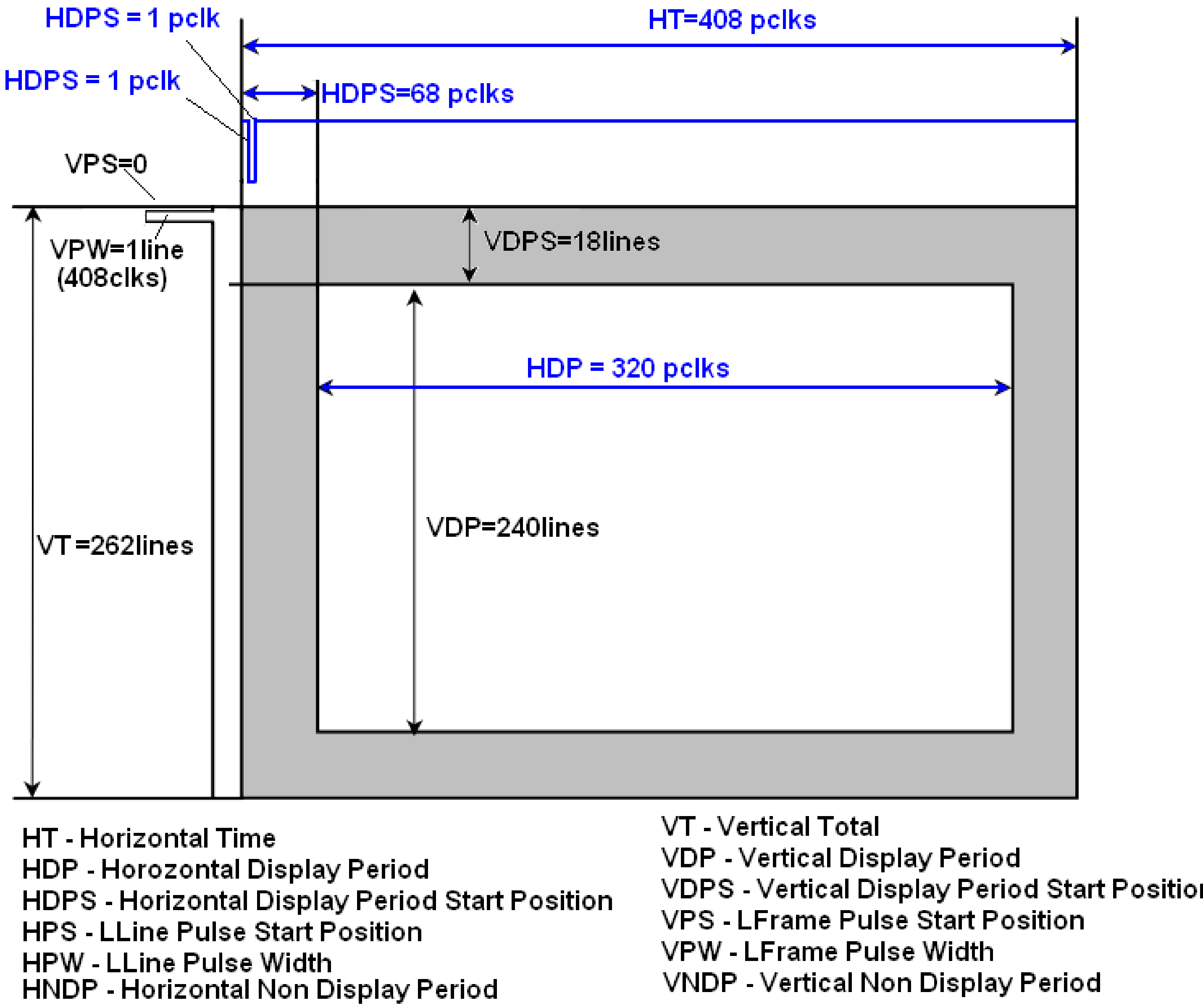
VT = vertical total is the values of registers &H19 and &H18 plus 1 and is measured in lines. These registers are setup with the values &H01, &H05. &H0105 = 261, so VT is 262lines. From the previous measurements we know 1 line is HT (horizontal total) and is 72.53uS therefore VT = 262 x 72.53 = 19mS.

VPW is vertical pulse width and is register &H24 value + 1 = 2 lines (145uS). Reg &H24 also sets VSync to be a negative pulse.

VDPS = vertical display period start position (VSync pulse width + vertical back porch). This is set by registers &H1F and &H1E. These are set to &H12 = 18 lines, 18 x 72.5uS = 1308uS, this was measured as 1174uS+143uS = 1317uS.

VDP = vertical display period and is set by registers &H1D, &H1C. These have the value &HEF, so VDP = &HEF+1 = 240 lines.
All these values are shown on Page 25 of the SSD app note.

This timing diagram has been taken from the SSD application note and modified to show the clocks



The following code is responsible for all these setups.

56.19 HX and SSD setup routine

'***

56.20 'SSD1928_HardwareSetup_Routines.bas

```
$nocompile

Sub Resetdevice()
   'setup default levels for micro to SSD control lines
   Rd = 1                                     'read high
   Wr = 1                                     'write high
   Cs = 1                                     'chip select high
   Rs = 0                                     'send/receive command low
   Bl = 1                                     'back light on
   Slp = 0                                    'pll enabled
   Datdir = &HFF

   'pulse reset line to make sure SSD is in known state
   Rst = 0                                    'ssd halt
   Waitms 10
   Rst = 1                                    'ssd run
   Waitms 10

   Call Setregb(&Ha0 , &H0)                   'reg power save off
   Waitms 200
   'setup SSD to HX serial lines
   Call Setregb(&Ha8 , &H0F)                  'reg gpio config0
   Call Setregb(&Ha9 , &H80)                  'reg gpio config1

   'set up HX,
   'see HX8238 datasheet for each word and indiv bit descriptions
   'set serial lines to known state
   Call Gpio_spena(1)
   Call Gpio_spclk(1)
   Call Gpio_spdat(1)
   Call Gpio_spreset(1)
   'reset HX
   Call Gpio_spreset(0)
   Waitms 1
   Call Gpio_spreset(1)
   'setup HX
   Call Spi_setreg(&H01 , &H7300)             'driver output control
   Call Spi_setreg(&H02 , &H0200)             'LCD driving waveform control
   Call Spi_setreg(&H03 , &H6364)             'power control 1

   'input data and color filter control
   'palm=1
   'blt1-0=00,
   'sel2-0=001 input interface mode=serial rgb, 19.5MHz operating
freq
   'swd2-0=111
   Call Spi_setreg(&H04 , &H040F)             'input data and colour filter

   Call Spi_setreg(&H05 , &HBCC4)             'function control
```

```
'brightness default=&H40(0) range is from 7F(+126) to 00(-128)
'contrast default=&H08 (1) range is from &H0(0) to &H1F(3.875)
Call Spi_setreg(&H0a , &H4008)       'contrast/brightness

Call Spi_setreg(&H0b , &HD400)       'frame cycle control
Call Spi_setreg(&H0d , &H3229)       'power control 2
Call Spi_setreg(&H0e , &H3200)       'power control 3 VOML

'Vertical rolling of the display
Call Spi_setreg(&H0f , &H0000)       'gate scan position

'320 pixels
'&H9F80 = 1001 1111 1000 0000 page 40 in HX datasheet
Call Spi_setreg(&H16 , &H9F80)

' vertical porch
'&H2122 = 0010 0001 0010 0010
'00 are ignored
'1000010 are the HBP bits (horizontal back porch) = 66 in decimal
'0100010 are the VBP bits (vertical back porch) = 34 in decimal
Call Spi_setreg(&H17 , &H2212)       'vertical/horizontal porch

Call Spi_setreg(&H1e , &H0052)       'power control 4 VCOMH
Call Spi_setreg(&H30 , &H0000)       'gamma control 1
Call Spi_setreg(&H31 , &H0407)       'gamma control 1
Call Spi_setreg(&H32 , &H0202)       'gamma control 1
Call Spi_setreg(&H33 , &H0000)       'gamma control 1
Call Spi_setreg(&H34 , &H0505)       'gamma control 1
Call Spi_setreg(&H35 , &H0003)       'gamma control 1
Call Spi_setreg(&H36 , &H0707)       'gamma control 1
Call Spi_setreg(&H37 , &H0000)       'gamma control 1
Call Spi_setreg(&H3a , &H0904)       'gamma control 2
Call Spi_setreg(&H3b , &H0904)       'gamma control 2

'****************************************************************
'setup SSD1928
'LLine = LCD line
'LFrame = LCD frame
'LShift = LCD shift = dotclock
'LCD_DEN = LCD enable

'SSD PLL - phasew locked loop setup
'internal clocks and the dotclock/LShift are generated from this
clock
'72MHz
Call Setregb(&H126 , &H0A)           'N Value
Call Setregb(&H127 , &HB4)           'M Value
Call Setregb(&H12b , &HAE)           '72mhz
Call Setregb(&H126 , &H8A)           'enable pll pll clock on
config 0

'MCLK divider register
'MCLK = PLL/(mclk div reg+1)
'so MCLK = PLL clk
Call Setregb(&H04 , &H0)             'mem reg        mem clock
config
```

```
   Waitms 20

   'PCLK freq ratio register
   '&H013FFF
   'PCLK = MCLK * (ratio+1) / 2^20
   '=72MHZ *(&H14000)/ 1048576
   '=72000000 * 81920 / 1048576
   '= 5625000 = 5.625MHz
   Call Setregb(&H158 , &HFF)            'pclk freq ratio register0
   Call Setregb(&H159 , &H3F)            'pclk freq ratio register1
   Call Setregb(&H15a , &H01)            'pclk freq ratio register2
   'LShift/DotClk varies in frequency, see page 25 of the SSD app
Note
   'LShift=3/4 * MCLK *(PCLK ratio+1)/2^18    - during visible data
   'LShift=MCLK *(PCLK ratio+1)/2^18          - during non visible
time
   'LShift=22.5MHz

   'set up display timing
   '&H52 = 0101 0010
   '0 = color CSTN delta type panel
   '1 = color
   '01 = 8 bit data width
   '0  must be programmed as 0
   '010 = serial TFT
   Call Setregb(&H10 , &H52)  'panel type = delta cstn, color, serial
tft
   Call Setregb(&H11 , &H0)              'reg mod rate


   '-----------------------------------------------------------------------
---
   'LLine pulse start position register
   'HPS or horizontal pulse start position= time to neg edge of hysnc
= 0
   'lline start position
   Call Setregb(&H22 , 0)                'reg hsync pulse start pos
   Call Setregb(&H23 , 0)                'reg hsync pulse start pos

   Call Setregb(&H21 , &H0)          'reg lline pulse start TIMER0
subpixel pos

   'HT
   'horizontal total
   '&H32, &H07 = 0011 0010 0000 0111
   'HT= 00110010111 + 1 = dec 408
   Call Setregb(&H12 , &H32)             'reg horiz total0
   Call Setregb(&H13 , &H07)             'reg horiz total1

   'HDP
   'horizontal display period (H27+1)*8 = H140 = 320dec
   'must be less than  step above
   Call Setregb(&H14 , &H27)
```

```
   'HDPS horizontal display period start position=&H0044=0100
0100=dec 68
   Call Setregb(&H16 , &H44)            'horizontal display start
position
   Call Setregb(&H17 , &H00)    'horizontal display start position

   'Lline or HPW = 0 = active low
   Call Setregb(&H20 , &H0)             'reg hsync pulse width


   '-----------------------------------------------------------
   'VPS = 'LFrame pulse start position
   Call Setregb(&H26 , &H00)            'lframe pulse start position
   Call Setregb(&H27 , &H00)            'lframe pulse start position

   'LFrame pulse start offset
   Call Setregb(&H31 , &H00)
   Call Setregb(&H30 , &H00)

   'VT
   'vertical total register = &H0105 = 261+1 = 262 lines
   'the sum of vertical display and vertical non display period
   'VDS + VDP must be less than VT
   Call Setregb(&H18 , &H05)            'vertical total
   Call Setregb(&H19 , &H01)            'vertical total

   'VPW
   'LFrame pulse width reg value + 1 = 2
   Call Setregb(&H24 , &H01)            'lframe pulse width


   Call Setregb(&H35 , &H00)            'lframe pulse stop offset

   'VDPS = &H12 = 18 lines
   Call Setregb(&H1e , &H12)         'Vertical display period start
pos
   Call Setregb(&H1f , &H00)         'Vertical display period start
pos

   'VDP = &HEF+1 = 240
   Call Setregb(&H1c , &HEF)            'vertical display period
   Call Setregb(&H1d , &H00)            'vertical display period

   Call Setregb(&Ha0 , &H00)            'display enable
End Sub
```

There are two window areas that can be used within the SSD memory, the first is the main window the second is a floating window that can be drawn over the top of the main window in this file are the routines for the windows and the routine to set which window is in focus. At this stage the floating window routines have not been used.

```
'*****************************************************************
```

56.21 SSD1928_Window_Control_Routines.bas

```
$nocompile
'*****************************************************************
Sub Ssd1928_focuswnd(byval Wnd As Byte)
   Local Linewidth As Word

   If Wnd = 0 Then                               'main window
       _page= 0
      _line_mem_pitch = Line_mem_pitch
   Else                                          'floating window
       page  = 1
      _byte = Getreg(&H81)
      _word = _byte
      Shift _word , Left , 8
      Linewidth = _word
      _byte = Getreg(&H80) And &HFF
      _word = _byte
      Linewidth = Linewidth Or _word
      Shift , Linewidth , Left , 1
      _line_mem_pitch = Linewidth
   End If
End Sub

'****************************************************************'Call
Ssd1928_mainwndinit(0 , 320 , 16 , 0 , 1)
'orientation rotates lcd but is not implemeted yet
'no point in setting rgb to anything but 1 yet

Sub Ssd1928_mainwndinit(byval Startaddr As Long , Byval Linewidth As
Word , Byval Bpp As Word , Byval Orient As Byte , Byval Rgb As Byte)
  'SSD memory is 256K so 17 bits are reqd to address it
  'main window start address register requires 3bytes to storeaddr
  'Reg &H74  main window display start address -least significant
byte
  'Reg &H75  main window display start address
  'Reg &H76  main window display start address - bit 17 is stored
here
   'although we pass it the address we pass it 0
   'so this doesnt actually do anything
   _long = Startaddr
   Shift _long , Right , 2       'pg 53 ssd1928 appnote reqs addr/4
   Call Setregl(&H74 , _long)

   'Reg &H78=main window line address offset reg-least signifcant
byte
   'Reg &H79 = main window line address offset register
   'tell the SSD how far in ram each line is stored from the last
   'each addr is a double word (32bits) divide 320 pix by 2 to get
offset
   _word = Linewidth
   Shift _word , Right , 1
   Call Setregw(&H78 , _word)

   'Reg &H70 = display mode register
```

```
    'even though we get bpp we fix it as 16 bits at this stage
    _byte = Getreg(&H70) Or &B100  'read reg, fix bit2 (assume
bits1,0=0)
    Call Setregb(&H70 , _byte)           'write back reg

    '&H71 special effects register   pages 93-99 in ssd1928a appnote
    'get all bits force byte swap to 1, leave word swap which is 0
    _byte = Getreg(&H71) Or &B01000000
    'set bits 1 nd 0 as per orientation
    Select Case Orient
          Case 0:                         '00
               _byte = _byte And &B11111100    'bits 1,0 low (no
rotate)
          Case 1:                         '01
               Reset _byte.1              'force bit 1 to 0
               Set _byte.0                'force bit 0 to 1
          Case 2:                         '10
               Set _byte.1                'force bit 1 to 1
               Reset _byte.0              'force bit 0 to 0
          Case 3:                         '11
               _byte = _byte Or &B00000011       'force bit 1 and 0
to 1
    End Select
    Call Setregb(&H71 , _byte)

    '&H1A4 RGB/YUV setting register  - main window = bit 6
    'YUV setting not used as yet
    _byte = Getreg(&H1a4)                 ' read register
    _byte.6 = Rgb.0                       'force bit 6 to be the same
as rgb.0
    Call Setregb(&H1a4 , _byte)           'save register
End Sub


'****************************************************************
'Reg &H70 = display mode register

Sub Ssd1928_mainwndenable(byval _enable As Byte)
    _bit = _enable                        '.0
    _byte = Getreg(&H70)
    _byte.7 = Not _bit                    '1 = on so invert
    Call Setregb(&H70 , _byte)
End Sub
```

```
'**************************************************************Sub
Ssd1928_floatwndinit(byval Startaddr As Long , Byval Linewidth As
Word , Byval X As Word , Byval Y As Word , Byval Width As Word ,
Byval Height As Word , Byval Rgb As Byte)
   'Local _word As Word
   'Local _long As Long

   _word = X
   Shift _word , Right , 1
   Call Setregw(&H84 , _word)
   _word = X + Width
   Shift _word , Right , 1
   _word = _word - 1
   Call Setregw(&H8c , _word)

   Call Setregw(&H88 , Y)
   _word = Y + Height
   _word = _word - 1
   Call Setregw(&H90 , _word)

   _long = Startaddr
   Shift _long , Right , 2
   Call Setregl(&H7c , _long)
   _word = Linewidth
   Shift _word , Right , 1
   Call Setregw(&H80 , _word)

   _byte = Getreg(&H70)
   _byte2 = _byte Or 4
   Call Setregb(&H70 , _byte2)

   '&H1A4 RGB/YUV setting register  - float window = bit 7
   _byte = Getreg(&H1a4)                  ' read reg, force bit 7 high
   _byte.7 = Rgb.0                        'force bit 6 to be the same
as rgb.0
   Call Setregb(&H1a4 , _byte)            'save register

   End Sub

'**************************************************************Sub
Ssd1928_floatwndenable(byval _enable As Byte)
   _byte = Getreg(&H71)                   'read register
   _byte.4 = _enable.0                    'bit4  0 = off 1 = visible
   Call Setregb(&H71 , _byte)             'write register
End Sub

'**************************************************************
```

56.22 Colour data in the SSD memory

The 256kbyte SRAM stores the colour data for each pixel as 2 bytes.

These are setup as RRRRRRGGGGGGBBBBB (5 bits of red, 6 bits of green and 5 bits of blue)

The following colors have been predefined.

```
'*********************************************************************
**
'SSD1928_Color_Defines.h
'*********************************************************************
**
$nocompile
'color definitions

Black Alias &B0000000000000000
Brightblue Alias &B0000000000011111
Brightgreen Alias &B0000011111100000
Brightcyan Alias &B0000011111111111
Brightred Alias &B1111100000000000
Brightmagenta Alias &B1111100000011111
Brightyellow Alias &B1111111111100000
Blue Alias &B0000000000010000
Green Alias &B0000010000000000
Cyan Alias &B0000010000010000
Red Alias &B1000000000000000
Magenta Alias &B1000000000010000
Brown Alias &B1111110000000000
Lightgray Alias &B1000010000010000
Darkgray Alias &B0100001000001000
Lightblue Alias &B1000010000011111
Lightgreen Alias &B1000011111110000
Lightcyan Alias &B1000011111111111
Lightred Alias &B1111110000010000
Lightmagenta Alias &B1111110000011111
Yellow Alias &B1111111111110000
White Alias &B1111111111111111
```

56.23 Accessing the SSD1928 colour memory

These routines are used to access the 256K byte colour data ram. When using these the address is sent first using one routine and then the pixel data is sent using a second routine.

We only have an 8bit databus however between the AVR and the SSD1928, so usual practice would be to get 8 bits at a time from a 16bit (word) or 32 bit(long) variable by rotating the var 8 times for each byte. This is the process used in the previous routines to access the control registers in the SSD, however this is too slow even using a 20MHZ AVR when we want to draw lines and fill boxes in the colour data memory so to improve the speed of these routines it is quicker to use the BASCOM overlay function where a byte or word can be accessed which is part of a larger variable in memory.
If the var is a long then the 2 words sized vars that make it up can be accessed as also can the 4 byte size vars.

long (4 bytes) 0000 0001 0000 0002 0000 0003 0000 0004			
word1 (2 bytes) 0000 0001 0000 0002		word2 (2 bytes) 0000 0003 0000 0004	
byte1 0000 0001	byte2 0000 0002	byte3 0000 0003	byte4 0000 0004
Address	Address +1	Address +2	Address +3

```
$nocompile
'***********************************************************************
***
```

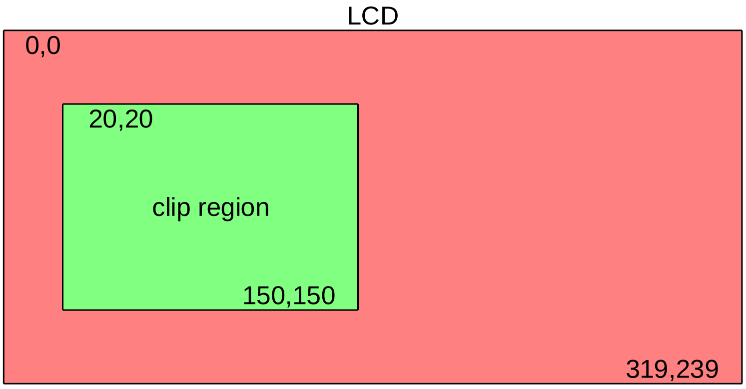
56.24 'SSD1928_Memory_Routines.bas

```
'routines that allow access to the 256K ram in the SSD1928
Dim _mem_lng As Long                      'e.g. at &H60
Dim _mem_wrd2 As Word At _mem_lng + 2 Overlay         'so at &H62
Dim _mem_wrd1 As Word At _mem_lng Overlay         'so at &H60
Dim _mem_b4 As Byte At _mem_lng + 3 Overlay         'so at &H63
Dim _mem_b3 As Byte At _mem_lng + 2 Overlay         'so at &H62
Dim _mem_b2 As Byte At _mem_lng + 1 Overlay         'so at &H61
Dim _mem_b1 As Byte At _mem_lng Overlay 'so at &h60
'***********************************************************************
****
'write an address in memory to the SSD before sending data
Sub Setaddress(byval Address As Long)
   'Datdir = &HFF
   Rd = 1
   Wr = 0
   Rs = 0
   _mem_lng = Address

   'send first byte, and make bit7 = 1 because we are accessing
memory
   Datout = _b3 Or &B10000000           'third byte
   Cs = 0
   Cs = 1
   Datout = _mem_b2                     'second byte
   Cs = 0
   Cs = 1
   Datout = _mem_b1                     'first byte
   Cs = 0
```

```
   Cs = 1
   Wr = 1
End Sub

'*********************************************************************
****'writes 16 bits of colour data to a previously setup address
Sub Writedata(byval Value As Word)

   'Datdir = &HFF
   Rd = 1
   Wr = 0
   Rs = 1                                     'Rs is high to write data

   _mem_wrd1 = Value
   Datout = _mem_b2                           'high 8 bits
   Cs = 0
   Cs = 1

   Datout = _mem_b1                           'low 8 bits
   Cs = 0
   Cs = 1

   Wr = 1
End Sub


'*********************************************************************
****'gets a byte of data from the ram in the ssd
Function Getdata(byval Void As Byte) As Byte
   Local Value As Byte

   Rs = 1
   Rd = 0
   Wr = 1

   Datdir = 0                                 'set portb to input
   Cs = 0

   Value = Datin                              'read data
   Cs = 1
   Rd = 1
   Datdir = &HFF                              'set portb back to input

   Getdata = Value
End Function
'*********************************************************************
****
```

56.25 Drawing simple graphics

Finally we have the SSD1928 and the HX8238 set up correctly, we have the ability to put colour data into the SSD1928 RAM now we need some routines to draw some simple graphics like place a pixel, draw lines and boxes.



The first routine allows us to set a pixel in the LCD at the coordinates X,Y of the LCD. Typically with an LCD 0,0 is the top left coordinate (note that this is different from a line or bar graph that we might draw which has the bottom left corner as 0,0). The bottom right corner is 319, 239.

We may already have previously defined a clip region, this is a smaller area of the screen that we might set aside as ok for drawing graphics, and we first test to see if it is defined and then if it is whether the pixel falls within it.

The next step is to locate where in the RAM the pixel data should actually be. Here are some sample calculations, note that we need to store 2 bytes at once so we multiply X by 2 and also need to offset Y by 640 bytes in RAM each time we come to a new line on the LCD.

LCD location (X,Y)	RAM address calculation Y x 640 + X x 2	RAM address
0, 0 (top left)	0 x 640 + 0 x 2	0
1, 0	0 x 640 + 1 x 2	2
2, 0	0 x 640 + 2 x 2	4
319, 0 (top right)	0 x 640 + 319 x 2	638
1,0	1 * 640 + 0 x 2	640
1,1	1 x 640 + 1 x 2	642
0, 239 (bottom left)	239 x 640 + 0 x 2	152,960
319 ,239 (bottom right)	239 x 640 + 239 * 2	153,598

```
'*********************************************************************
***
```

56.26 'SSD1928_Simple_Graphics_Routines.bas

```
$nocompile
Sub Putpixel(byval _x As Word , Byval _y As Word , Byval _color As
Word)
   Local _address As Long
   Local _draw As Byte
   Local _temp As Long
   'work out position of lcd pixel in SSD1928 ram
   _address = Page_mem_size * _page
   _temp = _line_mem_pitch
   Shift _temp , Left , 1
   _temp = _temp * _y
   Shift _x , Left , 1
   _temp = _temp + _x
   _address = _address + _temp
   Call Setaddress(_address)
   Call Writedata(_color)
End Sub
```

```
'**********************************************************************
Sub Drawline(byval X1 As Word , Byval Y1 As Word , Byval X2 As Word ,
Byval Y2 As Word , Byval _color As Word)
  Local _i As Integer
  Local Dx As Integer
  Local Dy As Integer
  Local Sdx As Integer
  Local Sdy As Integer
  Local Dxabs As Integer
  Local Dyabs As Integer
  Local X As Integer
  Local Y As Integer
  Local Px As Word
  Local Py As Word
  Local Itemp As Integer

  Dx = X2 - X1                    'the horizontal distance of the line
  Dy = Y2 - Y1                    'the vertical distance of the line
  Dxabs = Abs(dx)
  Dyabs = Abs(dy)
  Sdx = Dx / Abs(dx)
  Sdy = Dy / Abs(dy)
  X = Dyabs / 2
  Y = Dxabs / 2
  Px = X1
  Py = Y1

  Call Putpixel(px , Py , _color)

  If Dxabs >= Dyabs Then     'the line is more horizontal than
vertical
    Itemp = Dxabs - 1
    For _i = 0 To Itemp
      Y = Y + Dyabs
      If Y >= Dxabs Then
        Y = Y - Dxabs
        Py = Py + Sdy
      End If
      Px = Px + Sdx
      Call Putpixel(px , Py , _color)
    Next
  Else                       'the line is more vertical than
horizontal
    Itemp = Dyabs - 1
    For _i = 0 To Itemp
      X = X + Dxabs
      If X >= Dyabs Then
        X = X - Dyabs
        Px = Px + Sdx
      End If
      Py = Py + Sdy
      Call Putpixel(px , Py , _color)
    Next
  End If
End Sub
```

```
'**********************************************************************
Sub Fillbox(byval _x1 As Word , Byval _y1 As Word , Byval _x2 As Word
, Byval _y2 As Word , Byval _color As Word)
   Local _x As Word
   Local _y As Word

    For _y = _y1 To _y2
        For _x = _x1 To _x2
            Putpixel _x , _y , _color
        Next
    Next
End Sub
'**********************************************************************
Sub Drawbox(byval _x1 As Word , Byval _y1 As Word , Byval _x2 As Word
, Byval _y2 As Word , Byval _color As Word)
   Drawline _x1 , _y1 , _x1 , _y2 , _color
   Drawline _x2 , _y1 , _x2 , _y2 , _color
   Drawline _x1 , _y1 , _x2 , _y1 , _color
   Drawline _x1 , _y2 , _x2 , _y2 , _color
End Sub
'**********************************************************************
Function Rgb(byval _r As Byte , Byval _g As Byte , Byval _b As Byte)
   Local Wtemp As Word
   Local Return_val As Word
   Return_val = 0

   Wtemp = _r
   Shift Wtemp , Left , 8
   Wtemp = Wtemp And &B1111100000000000
   Return_val = Return_val Or Wtemp

   Wtemp = _g
   Shift Wtemp , Left , 3
   Wtemp = Wtemp And &B0000011111100000
   Return_val = Return_val Or Wtemp

   Wtemp = _b
   Shift Wtemp , Right , 3
   Wtemp = Wtemp And &B0000000000011111
   Return_val = Return_val Or Wtemp

   Rgb = Return_val
End Function

'**********************************************************************
Sub Cleardevice()
   Local Counterr As Long
   Local L As Long
   L = Getregl(&H74)
   Rotate L , Left , 2
   Call Setaddress(0)
   For Counterr = 0 To 76799
      Call Writedata(backcolor)
   Next
End Sub
'*********************************************************************
```

56.27 SSD1928_text_routines

```
$nocompile
'***********************************************************************
****'Verdana font + initial routines by Abhilash (student)
'2 globals used so that new text flows on from the previous location
'note: a line is 8 rows high so 240 rows = 30 lines of text8

Dim _xpos As Word
Dim _ypos As Word
'locates cursor position
Sub Textpos(byval _x As Byte , Byval _y As Byte)
   _xpos = _x
   _ypos = _y
End Sub

'This routine prints lines of 8 point text on the LCD,
' it automatically wraps from one line to the next
' also remembers its position so that new text will flow on from the
old
Sub Text8(byval _text As String )
   Local _char As String * 1 , _letter As Word
   Local _textlen As Word , _charcount As Byte
   Local _columns As Word , _lookuppos As Word , _columndat As Byte ,
      _pixel As Byte
   _textlen = Len(_text)
   For _charcount = 1 To _textlen       'for each char in string
      _char = Mid(_text , _charcount , 1)        'get one character
      _letter = Asc(_char)               'find its pos in the ascii
table
      _letter = _letter - 32         'printable chars start at ascii
32
      For _columns = 0 To 7              '8 cols of data per char
      _lookuppos = _letter * 8     'find look up position in font
table
      _lookuppos = _lookuppos + 3     'ignore first 3 bytes in the
table
      _lookuppos = _lookuppos + _columns
      _columndat = Lookup(_lookuppos , Font8x8)
         For _pixel = 0 To 7
            If _columndat._pixel = 1 Then
               Call Putpixel(_xpos , _ypos , Forecolor)
            Else            'pixel not set so clear pixel to backcolor
               Call Putpixel(_xpos , _ypos , Backcolor)
            End If
            Incr _ypos                 'next row
         Next
         _xpos = _xpos + 1
         _ypos = _ypos - 8             'done all 8 rows go back up
         If _xpos > 320 Then
            _ypos = _ypos + 8          'go down 1 line
           _xpos = 0                   'start at beginning of line
        End If
      Next
   Next
End Sub
```

```
Sub Text16(byval _text As String )
   Local _yval As Word , _xval As Word , _line As Byte ,
     _char As String * 1 , _letter As Word
   Local _textlen As Word , _charcount As Byte
   Local _columns As Word , _lookuppos As Word ,
     _columndat As Byte , _pixel As Byte         '

   _textlen = Len(_text)
   'here we look up the upper line of each char then display it ,
   'then go on to the next char,
   'when all upper lines are displayed we do the lower lines
   For _line = 1 To 2             'there are 2 lines (16 bits height)
      _yval = _ypos
      _xval = _xpos
      For _charcount = 1 To _textlen     'for each char in string
         _char = Mid(_text , _charcount , 1)         'get one character
         _letter = Asc(_char)         'find its pos in the ascii table
         _letter = _letter - 32     'printable chars start at ascii 32

         For _columns = 0 To 15 'characters are 16 pixels wide
            _lookuppos = _letter * 32'find look up pos in font table
            _lookuppos = _lookuppos + 3'ignore first 3 bytes in table
            _lookuppos = _lookuppos + _columns
            If _line = 2 Then _lookuppos = _lookuppos + 16
            'data for 2nd line
            _columndat = Lookup(_lookuppos , Font16x16)
            For _pixel = 0 To 7
               If _columndat._pixel = 1 Then  'display in forecolor
                  Call Putpixel(_xval , _yval , Forecolor)
               Else                           ' clear pixel to backcolor
                  Call Putpixel(_xval , _yval , Backcolor)
               End If
               Incr _yval                     ' next column
            Next
            _yval = _yval - 8                 'back to top of column
            _xval = _xval + 1                 'next column
         Next
         If _xval > 319 Then
             _xval = 0
             _yval = _yval + 16
         End If
      Next
      _ypos = _ypos + 8                       'next line down
   Next
   _xpos = _xval                              'reset value for next text
input
   _ypos = _yval - 8                          'reset value for next text
input
End Sub
```

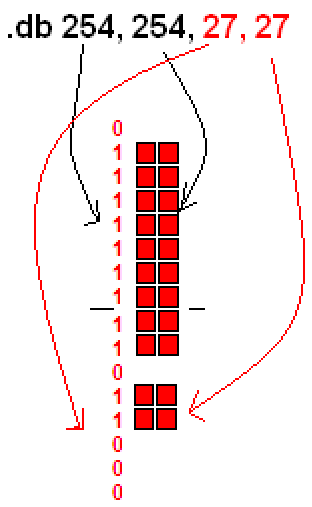
```
Sub Verdana(byval _text As String )
   Local _yval As Word , _xval As Word , _temp As Word
   Local _charwidth As Byte , _lookuppos_b As Byte , _lookuppos As
Word
   Local _line As Byte , _textlen As Word , _charcount As Byte ,
     _char As String * 1 , _letter As Word
   Local _columns As Word , _pixel As Word , _columndat As Byte
   _textlen = Len(_text)
   'this process is different to the 16x16 font
   'write top line of a charater then the bottom line of the
character
   'before going on to the next character
   For _charcount = 1 To _textlen       'for each char in string
      _char = Mid(_text , _charcount , 1)        'get one character
      _letter = Asc(_char)                'find its pos in the ascii
table
      _letter = _letter - 32            'printable chars start at ascii
32
      _letter = _letter * 3    'each letter in font table has 3 bytes
      'first byte is how many pixels wide the character is
      _charwidth = Lookup(_letter , Fontv)
      Incr _letter                     'move to addr of letter data
      'get hundreds of letter data lookup address
      _lookuppos_b = Lookup(_letter , Fontv)
      _lookuppos = _lookuppos_b * 100
      Incr _letter                'move to second part of lookup addr
      _lookuppos_b = Lookup(_letter , Fontv)
      _lookuppos = _lookuppos + _lookuppos_b
      _lookuppos = _lookuppos + 1         'lookup addr in the font
table
      _temp = _xpos + _charwidth
      'check there is room for the whole character to be displayed
      If _temp > 319 Then
         _xpos = 0
         _ypos = _ypos + 16
      End If
      'display the character
      For _line = 1 To 2    'there are 2xlines(8rows) for each
character
         _yval = _ypos
         _xval = _xpos
         For _columns = 1 To _charwidth 'get data for each character
            _columndat = Lookup(_lookuppos , Fontv)       'looks up
byte
               For _pixel = 0 To 7       'looks at each bit in byte
                  If _columndat._pixel = 1 Then        'turn on pixel
                     Call Putpixel(_xval , _yval , Forecolor)
                  Else          'pixel not set so clear pixel to
backcolor
                     Call Putpixel(_xval , _yval , Backcolor)
                  End If
                  Incr _yval            'next pixel
               Next
            Incr _lookuppos    'increase column position for next
loop
            _yval = _yval - 8
```

```
            Incr _xval                   'insert 1 column betwen characters
            For _pixel = 0 To 7               'fill space column with
backcolor
               Call Putpixel(_xval , _yval , Backcolor)
               Incr _yval
            Next
            _yval = _yval - 8
         Next
         _ypos = _ypos + 8                    'now writing the lower row
         If _line = 1 Then _xval = _xpos         'resetting the x
position
      Next
      Incr _xval                              'set x for next character
      _xpos = _xval
      _ypos = _ypos - 16                      'back to top
   Next
End Sub
```

Because each character in a true type font is not a fixed width as in the 8x8 and 16x16 font tables the lookup scheme for each character requires us to make 2 lookups. The first lookup finds the number of bytes for each character that need to be retrieved from the table and their starting position in the table, the second look up is the actual font data for displaying.

Here is some of the first line of the font table
.db 7 , 2 , 84 , 2 , 2 , 98 , 5 , 3 , 02 , 11 , 3 , 12 , 9 , 3 , 34 , 18 , 3 , 52 ....
The first character is 7 pixels wide and at location 284 in the table
The second character is 2 pixels wide and at position 298 in the table
The third character is 5 pixels wide and at position 302 in the table, etc
Each ine after the first line is an actual line of font data

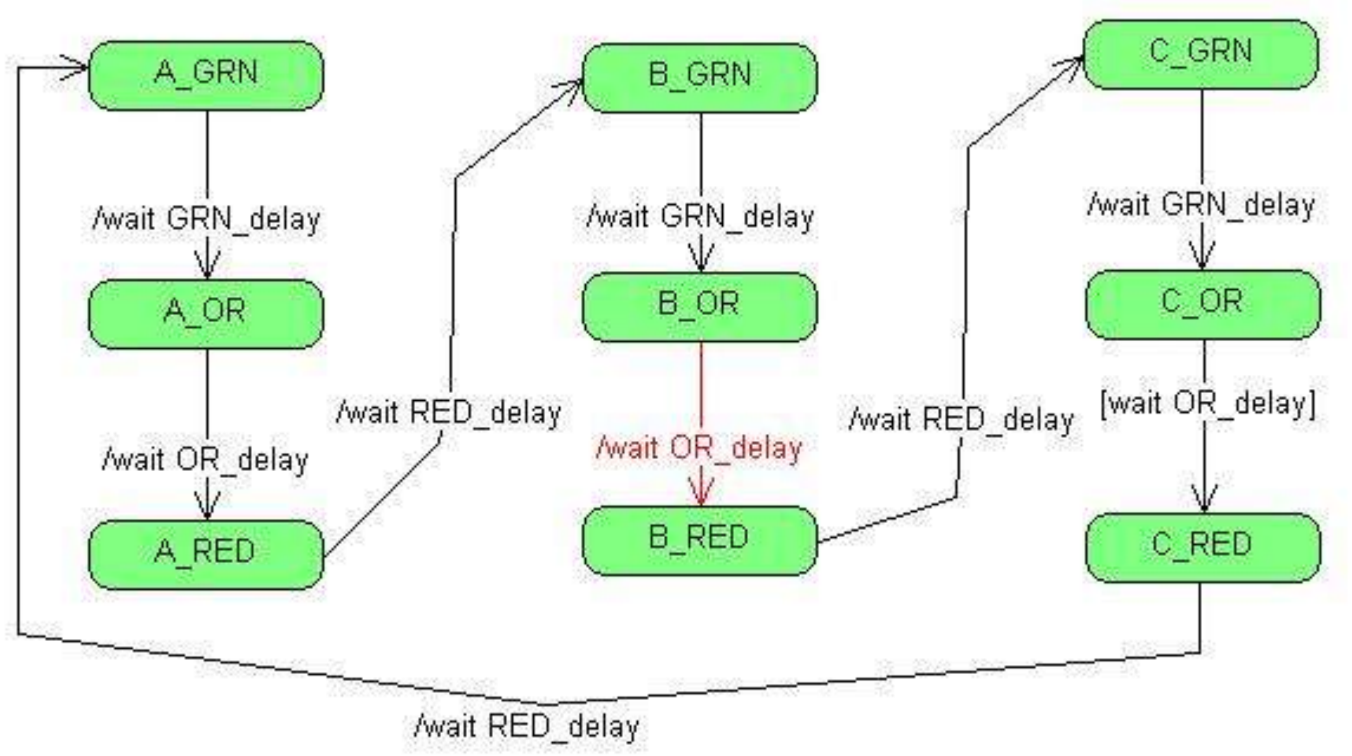
The third line in the font table is the exclamation mark it contains 16 vertical bits of data and takes up only two pixels width of the LCD. All the upper line (8 bits) of data are stored in the table first then the lower line
The exclamation mark is stored as .db 254 , 254 , 27 , 27 ; ! - 2pix



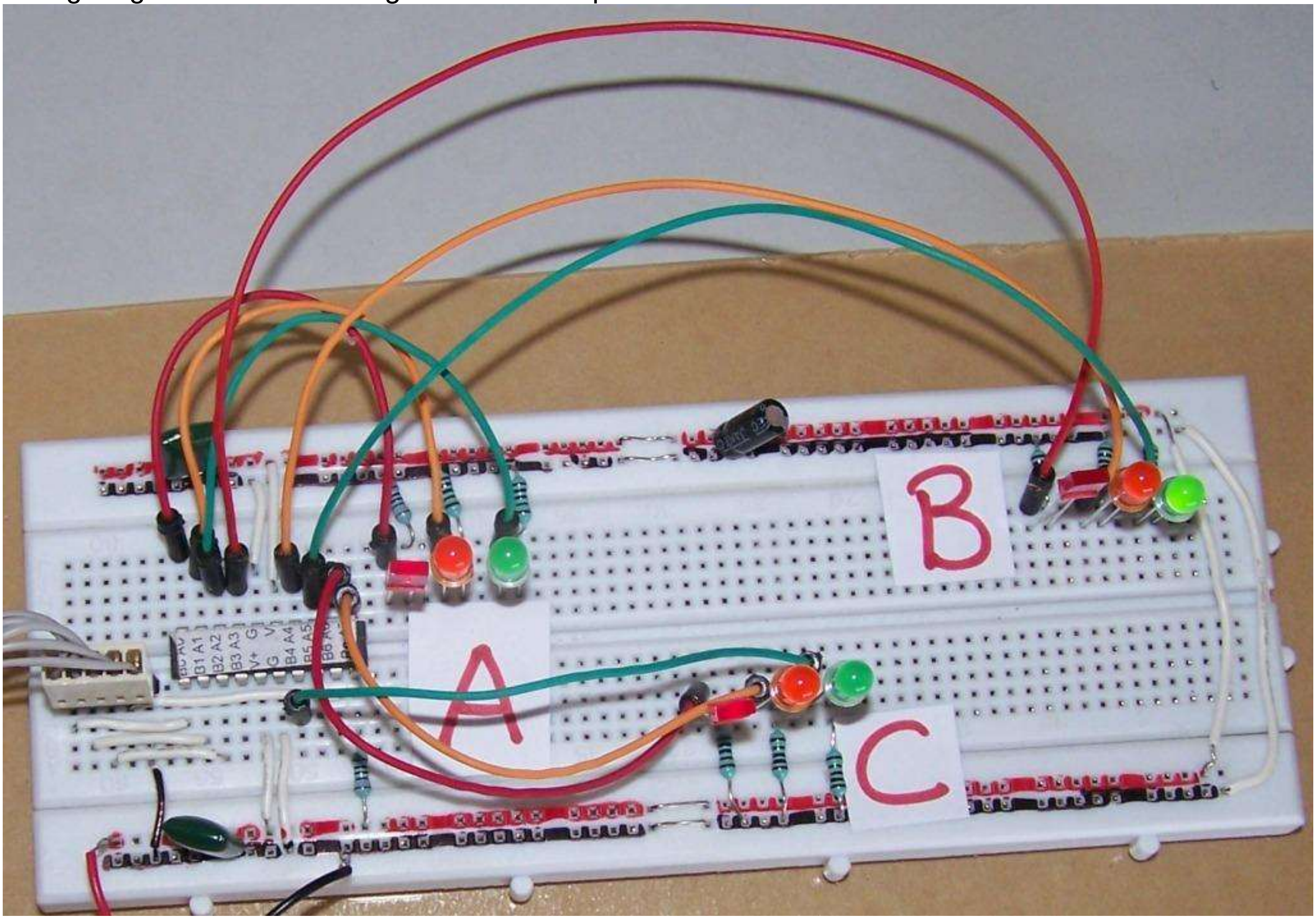
The first column of the first line is 254 = &B11111110
The second column repeats the first
The first column of the second line is 27 = &B00011011
The second column repeats the first

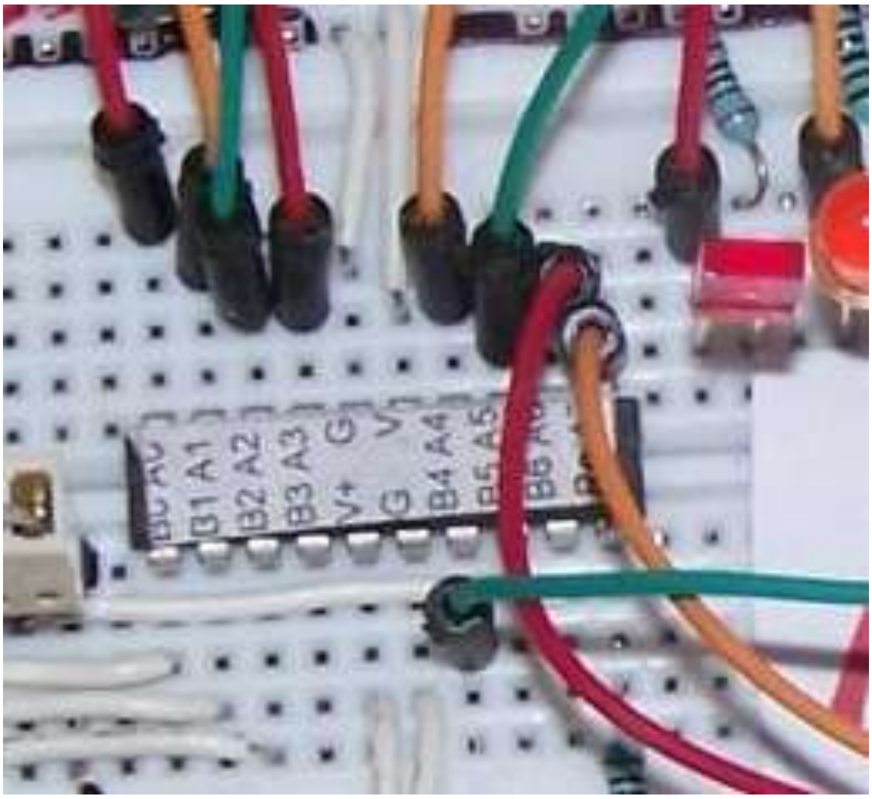
57 Traffic Light help and solution

Now here is some assistance for the traffic light exercise from early in the book



Wiring stage 4: the ‘C’ set of lights are wired up





For the last set of lights ports A.6 and A.7 are used as well as portB.4

Here is the final program for the traffic lights

```
'TrafficLightsVer1.bas
'B.Collis
'*************************

$crystal = 1000000
$regfile = "attiny26.dat"

Config Porta = Output
Config Portb = Output

'*************************
'LED connections
'use aliases so that the program is easier to write and understand
A_red Alias Porta.0
A_or Alias Porta.1
A_grn Alias Porta.2

B_red Alias Porta.3
B_or Alias Porta.4
B_grn Alias Porta.5

C_red Alias Porta.6
C_or Alias Porta.7
C_grn Alias Portb.4
'...
'...
'use constants to make the program easier to read and to change
Const Grn_delay = 8                     'green on time
Const Or_delay = 3                      'orange on time
Const Red_delay = 1                     'safety delay between red &
next green

'initially set the red lights on and all others off
'introducing the new commands SET and RESET to individually control
port pins
Set A_red                               'on
Reset A_or                              'off
Reset A_grn                             'off

Set B_red                               'on
Reset B_or                              'off
Reset B_grn                             'off

Set C_red                               'on
Reset C_or                              'off
Reset C_grn                             'off
```

```
Do
   'A lights
   Reset A_red                                 'off
   Set A_grn                                   'on
   Wait Grn_delay

   Reset A_grn                                 'off
   Set A_or                                    'on
   Wait Or_delay

   Reset A_or
   Set A_red
   Wait Red_delay              'small delay allows for red light runners!

   'B lights
   Reset B_red
   Set B_grn                                   'grn on
   Wait Grn_delay

   Reset B_grn                                 'grn off
   Set B_or
   Wait Or_delay

   Reset B_or
   Set B_red
   Wait Red_delay               'small delay allows for red light
runners!

   'C lights
   Reset C_red
   Set C_grn                                   'grn on
   Wait Grn_delay

   Reset C_grn                                 'grn off
   Set C_or
   Wait Or_delay

   Reset C_or
   Set C_red
   Wait Red_delay                'small delay allows for red light
runners!

Loop
End
```

58 Computer programming – low level detail

We refer to programming languages as either **HIGH LEVEL** languages or **LOW LEVEL**.

High Level Languages include Basic, C, Java, Haskell, Lisp, Prolog, C++, C# and many more.
High level languages are written using text editors such as Programmers Notepad or within an IDE such as Eclipse or Visual Studio or… These languages are typically easy for us to understand. However microcontrollers do not understand these words they only understand binary numbers which are called Machine Code. A computer program is ultimately a file with a .hex extension containing machine code. Commands written in high level languages must be compiled into these binary codes.

58.1 Low level languages:

Machine code for **all** microcontrollers and microprocessors (all computers) are groups of

binary digits (bits) arranged in bytes (8 bits) or words of 16, 32 or 64 bits. Understanding a program in machine code is not at all easy. The AVR machine code to add the numbers in 2 memory registers is 0001 1100 1010 0111.

To make machine code a little easier to understand we can abbreviate every 4 bits into hexadecimal numbers; HEX uses numbers 0 to 9 and the letters from a to f.

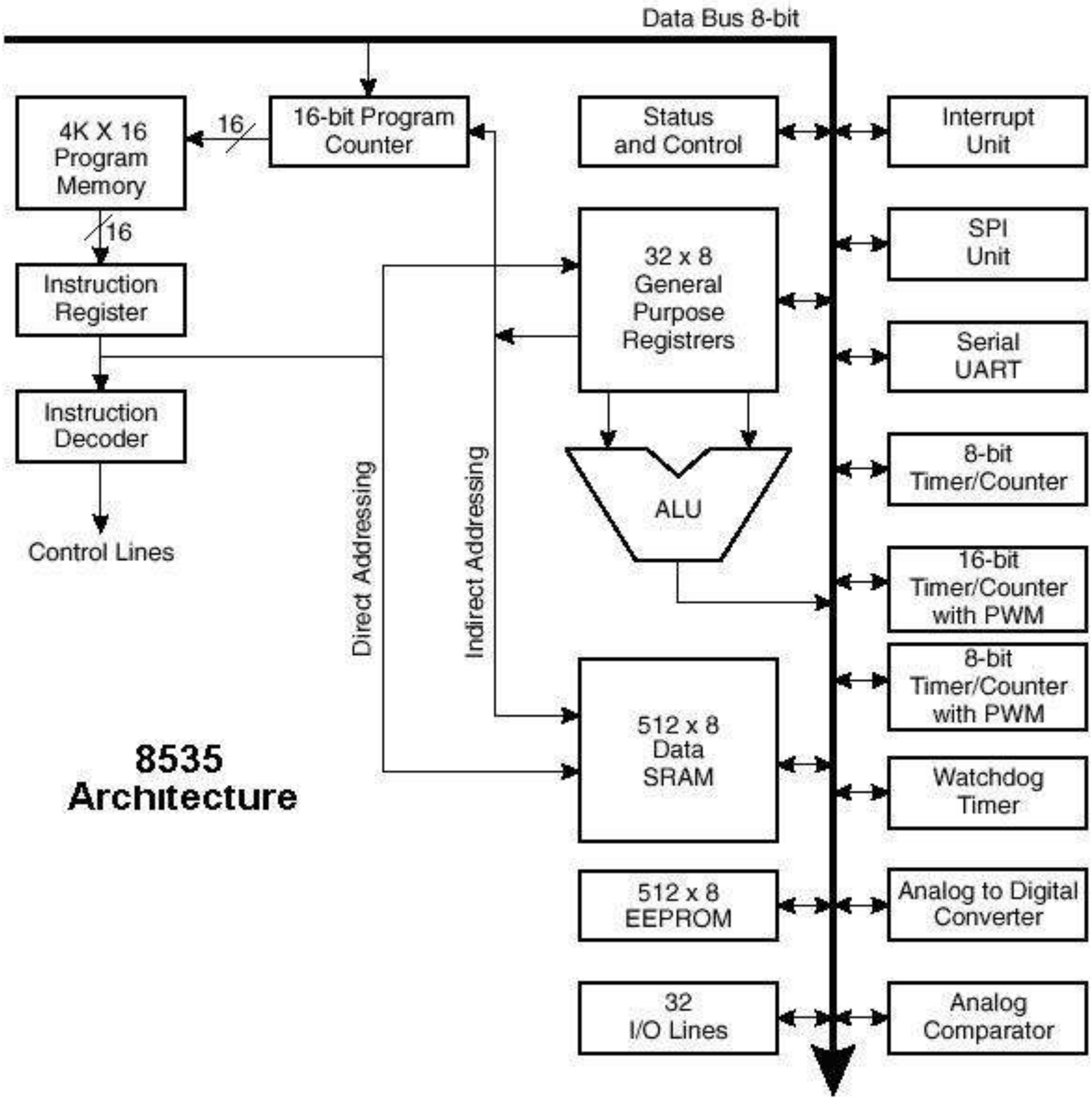
It is easier on the eyes than machine code but still very difficult to read. It looks like this **1CA7** which is easier to read than is 0001 1100 1010 0111**,** but no easier to understand! Program code for micros is never written today directly in machine code, abbreviations called mnemonics are used and we call it assembler or assembly language or, assembly code which is more readable, for example:
add r12 , r7 instead of **1C A7**

Assembler is much easier to understand than machine code and is in very common use for programming microcontrollers, however It does take more effort to understand the microcontroller internals when programming in assembler.

You can see the machine code in BASCOM by going to the directory where your programs are stored and opening the .hex file (ignore the colon and the first 8 digits in each line, the rest is the actual program). You can also see it when you go to manual programming mode its all the hexadecimal in the program window.

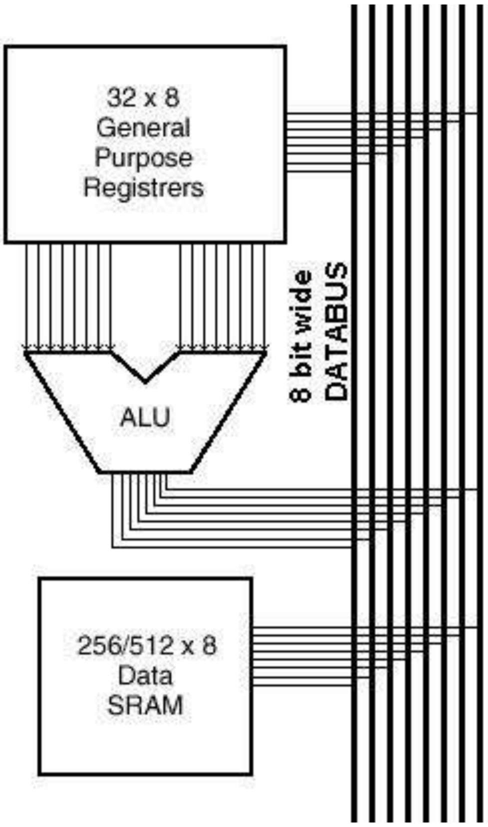
58.2 AVR Internals – how the microcontroller works

The AVR microcontroller is a complex integrated circuit, with many features as shown in this block diagram of the AVR's internal architecture.



There are memory, calculation, control and I/O components.

58.3 1. The 8bit data bus



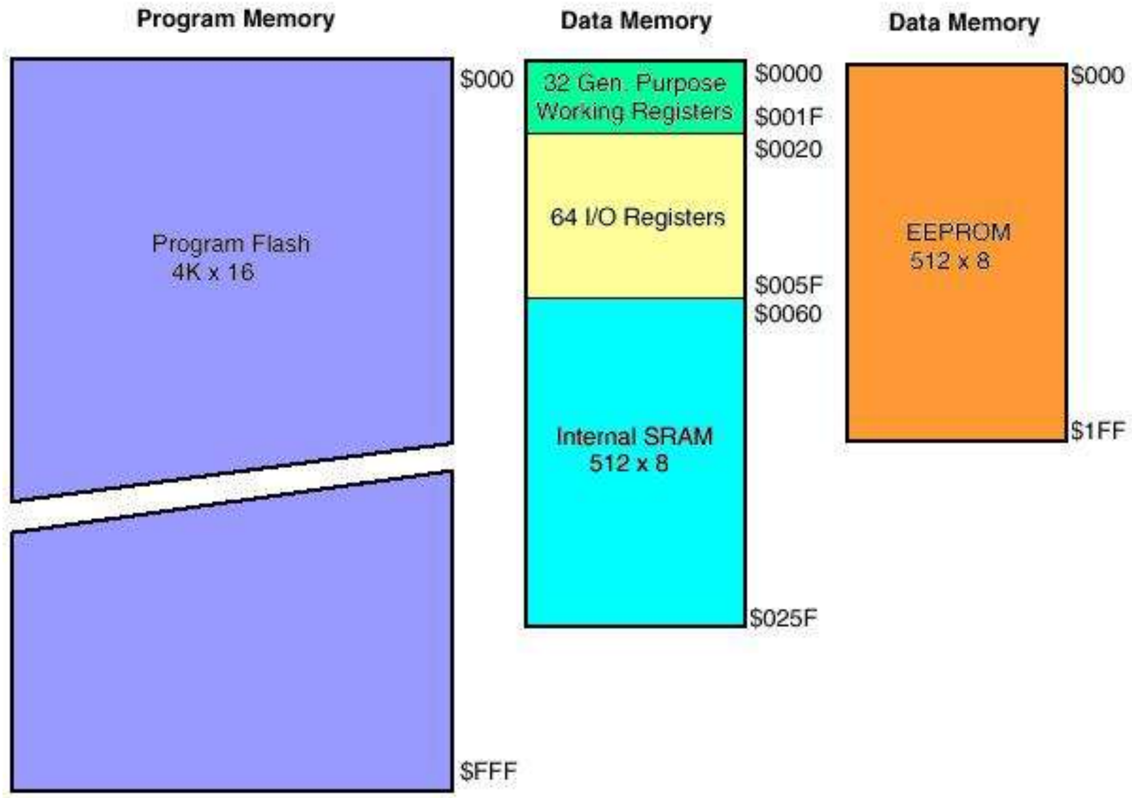
This is actually 8 parallel wires that interconnect the different parts within the IC. At any one time only one section of the 8535 is able to transmit on the bus.

Each device has its own address on the bus and is told when it can receive and when it can transmit data.

Note that with 8 bits (1 byte) only numbers up to 255 may be transmitted at once, larger numbers need to be transferred in several sequential moves.

58.4 2. Memory

There are three separate memory areas within the AVR, these are the Flash, the Data Memory and the EEPROM.



In the 8535 the Flash or program memory is 4k of words (8k bytes) of program. The AVR stores program instructions as 16 bit words. Flash Memory is like a row of lockers or pigeon holes. When the micro starts it goes to the first one to fetch an instruction, it carries out that instruction then gets the next one.

The Static RAM is a volatile store for variables within the program.

The EEPROM is a non-volatile store for variables within the program.

The 32 general purpose registers are used by your programs as temporary storage for data while the microcontroller is working on it (in some micros these are called accumulators).

If you had a line on your code to add 2 numbers e.g. z=x+y. The micro will get the contents of ram location X and store it in a register, it will get the contents of ram location Y and puts it into a second register, it will then add the 2 numbers and the result will go into one of the registers, it then writes the answer from that register into memory location Z.

The 64 I/O registers are memeory locations with special hardware abilities, when you change something in a register the hardware attached to it changes; it is here that you access the ports, ADC etc and their control them.

58.5 3. Special Function registers

There are several special high speed memory registers within the microcontroller.

* Program counter: 16 bits wide, this keeps track of which instruction in flash the microcontroller is carrying out. After completing an instruction it will be incremented to point at the next location.
* Instruction register: As a program instruction is called from program memory it is held here and decoded.
* Status Register: holds information relating to the outcome of processing within the microcontroller, e.g. did the addition overflow?

4. ALU
The arithmetic logic unit carries out mathematical operations on the binary data in the registers and memory, it can add, subtract, multiply, compare, shift, test, AND, OR, NOR the data.

58.6 A simple program to demonstrate the AVR in operation

Lets take a simple program in Bascom then analyse the equivalent machine code program and then what happens within the microcontroller itself.
This program below configures all of portc pins as outputs, then counts binary in a never ending loop on the LEDs on portc.

```
Config Portc = Output     'all of portc pins as outputs
Dim Temp As Byte          'set memory aside
Temp = 0                  'set its initial value to 0
Do
   Incr Temp              'increment memory
   Portc = Temp           'write the memory to port c
Loop                      'loop forever
End
```

This is compiled into machine code, which is a long line of binary numbers. However we don't normally view the numbers as binary, it is shorter to use hexadecimal notation.

Equivalent machine code to the Bascom code above is:
EF0F (1110 1111 0000 1111)

BB04
E000
BB05
9503
CFFD

These program commands are programmed into the microcontroller starting from the first address of the FLASH (program memory). When the micro is powered up (or reset) it starts executing instructions from that first memory location.

The equivalent assembly language to the above machine code

EF 0F	**SER R16**	set all bits in register 16	
BB 04	**OUT 0x14,R16**	store register 16 at address 14	(portc = output)
E0 00	**LDI R16,0x00**	load immediate register 16 with 0	(temp=0)
BB 05	**OUT 0x15,R16**	store register 16 at address 15	(port C = temp)
95 03	**INC R16**	increment register 16	(incr temp)
CF FD	**RJMP -0x0003**	jump back 3 steps in the program	(back to BB05)

1. The microcontroller powers up and the program counter is loaded with address &H000, the first location in the flash (program memory). The first instruction is EF 0F and it is transferred into the instruction register. The program counter is then incremented by one to 0x01. The instruction is decoded and register 16 is set to all ones.
2. The next cycle of the clock occurs and BB 04 is moved from the flash into the instruction register. The program counter is incremented by one to 0x02. The instruction is decoded and R16 contents are copied to address 0x14 (0x means hex), this is the i/o register that controls the direction of port c, so now all pins of portc are outputs.
3. The next cycle of the clock occurs and E0 00 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x03). The instruction is decoded and Register 16 is loaded with all 0's.
4. The next cycle of the clock occurs and BB 05 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x04). The instruction is decoded and the contents of register 16 (0) are copied to address 0x15 this is the i/o register address for portc itself – so all portc goes low.
5. The next cycle of the clock occurs and 95 03 is moved into the instruction register from the flash. The program counter is incremented by one (to 0x05). The instruction is decoded and the contents of register 16 are incremented by 1 (to 01). This operation requires the use of the ALU as a mathematical calculation is involved.
6. The next cycle of the clock occurs and CF FD is moved into the instruction register from the flash. The program counter is incremented by one (to 0x06). CF FD is decoded and the program counter has 3 subtracted from it (It is 0x06 at the moment so it becomes 0x03). The sequence jumps back to number three causing a never ending loop.

58.7 Bascom keyword reference

1WIRE

1Wire routines allow you to communicate with Dallas 1wire chips.
1WRESET , 1WREAD , 1WWRITE , 1WSEARCHFIRST , 1WSEARCHNEXT ,1WVERIFY , 1WIRECOUNT

Conditions

Conditions execute a part of the program depending on the condition
IF-THEN-ELSE-END IF , WHILE-WEND , ELSE , DO-LOOP , SELECT CASE - END SELECT , FOR-NEXT

Configuration

Configuration command initialize the hardware to the desired state.
CONFIG , CONFIG ACI , CONFIG ADC , CONFIG BCCARD , CONFIG CLOCK , CONFIG COM1 , CONFIG COM2 , CONFIG DATE , CONFIG PS2EMU , CONFIG ATEMU , CONFIG I2CSLAVE , CONFIG GRAPHLCD , CONFIG KEYBOARD , CONFIG TIMER0 , CONFIG TIMER1 , CONFIG LCDBUS , CONFIG LCDMODE , CONFIG 1WIRE , CONFIG LCD , CONFIG SERIALOUT , CONFIG SERIALOUT1 , CONFIG SERIALIN , CONFIG SERIALIN1 , CONFIG SPI , CONFIG LCDPIN , CONFIG SDA , CONFIG SCL , CONFIG DEBOUNCE , CONFIG WATCHDOG , CONFIG PORT , COUNTER0 AND COUNTER1 , CONFIG TCPIP

Conversion

A conversion routine is a function that converts a number or string.
BCD , GRAY2BIN , BIN2GRAY , BIN , MAKEBCD , MAKEDEC , MAKEINT , FORMAT , FUSING , BINVAL , CRC8 , CRC16 , CRC32 , HIGH , HIGHW , LOW

DateTime

Date Time routines can be used to calculate with date and/or times.
DATE , TIME , DATE$, TIME$, DAYOFWEEK , DAYOFYEAR , SECOFDAY , SECELAPSED , SYSDAY , SYSSEC , SYSSECELAPSED

Delay

Delay routines delay the program for the specified time.
WAIT , WAITMS , WAITUS , DELAY

Directives

Directives are special instructions for the compiler. They can override a setting from the IDE.
$ASM , $BAUD , $BAUD1 , $BGF , $BOOT , $CRYSTAL , $DATA , $DBG , $DEFAULT , $EEPLEAVE , $EEPROM , $EEPROMHEX , $EXTERNAL , $HWSTACK , $INC , $INCLUDE , $INITMICRO , $LCD , $LCDRS , $LCDPUTCTRL , $LCDPUTDATA , $LCDVFO , $LIB , $LOADER , $LOADERSIZE , $MAP , $NOINIT , $NORAMCLEAR , $PROG , $PROGRAMMER , $REGFILE , $ROMSTART $SERIALINPUT, $SERIALINPUT1 , $SERIALINPUT2LCD , $SERIALOUTPUT , $SERIALOUTPUT1 , $SIM , $SWSTACK , $TIMEOUT , $TINY , $WAITSTATE , $XRAMSIZE , $XRAMSTART , $XA

File

File commands can be used with AVR-DOS, the Disk Operating System for AVR.
BSAVE , BLOAD , GET , VER , , DISKFREE , DIR , DriveReset , DriveInit , , LINE INPUT , INITFILESYSTEM , EOF , WRITE , FLUSH , FREEFILE , FILEATTR , FILEDATE , FILETIME , FILEDATETIME , FILELEN , SEEK , KILL , DriveGetIdentity , DriveWriteSector , DriveReadSector , LOC , LOF , PUT , OPEN , CLOSE

Graphical LCD

Graphical LCD commands extend the normal text LCD commands.
GLCDCMD , GLCDDATA , SETFONT , LINE , PSET , SHOWPIC , SHOWPICE , CIRCLE

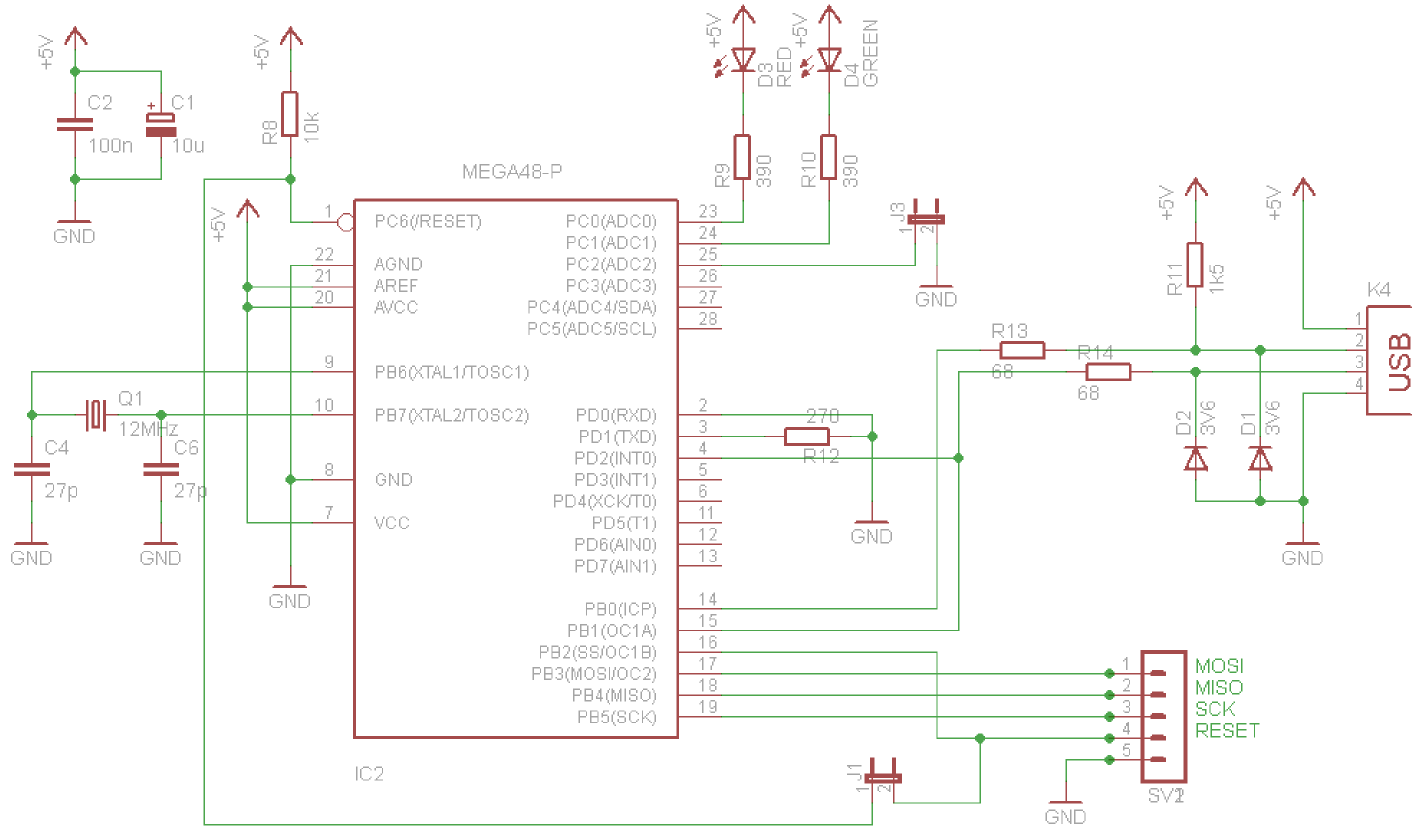
I2C

I2C commands allow you to communicate with I2C chips with the TWI hardware or with emulated I2C hardware.
I2CINIT , I2CRECEIVE , I2CSEND , I2CSTART,I2CSTOP,I2CRBYTE,I2CWBYTE

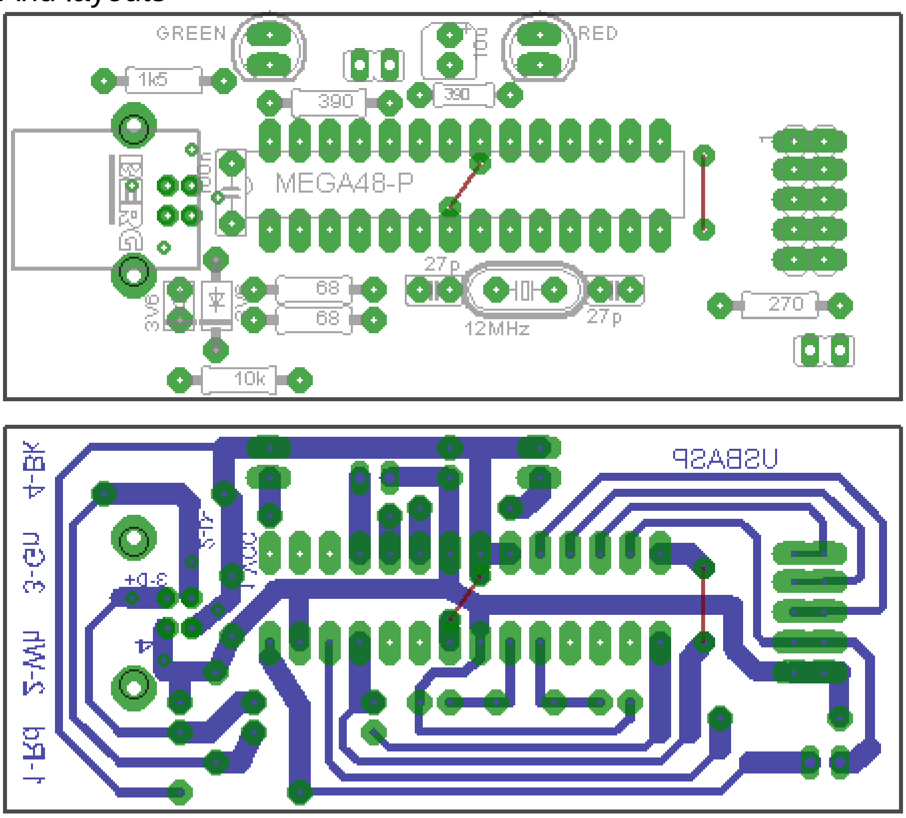
IO
I/O commands are related to the I/O pins of the processor.
ALIAS , BITWAIT , TOGGLE , RESET , SET , SHIFTIN , SHIFTOUT , DEBOUNCE , PULSEIN , PULSEOUT
Micro
Micro statements are highly related to the micro processor.
IDLE , POWERDOWN , POWERSAVE , ON INTERRUPT , ENABLE , DISABLE , START , END , VERSION , CLOCKDIVISION , CRYSTAL , STOP
Memory
Memory functions set or read RAM , EEPROM or flash memory.
WRITEEEPROM , CPEEK , CPEEKH , PEEK , POKE , OUT , READEEPROM , DATA , INP , READ , RESTORE , LOOKDOWN , LOOKUP , LOOKUPSTR , CPEEKH , LOAD , LOADADR , LOADLABEL , LOADWORDADR , MEMCOPY
Remote Control
Remote control statements send or receive IR commands for remote control.
RC5SEND , RC6SEND , GETRC5 , SONYSEND
RS-232
RS-232 are serial routines that use the UART or emulate a UART.
BAUD , BAUD1, BUFSPACE , ECHO , WAITKEY , ISCHARWAITING , INKEY , INPUTBIN , INPUTHEX , INPUT , PRINT , PRINTBIN , SERIN , SEROUT , SPC
SPI
SPI routines communicate according to the SPI protocol with either hardware SPI or software emulated SPI.
SPIIN , SPIINIT , SPIMOVE , SPIOUT
String
String routines are used to manipulate strings.
ASC , UCASE , LCASE , TRIM , SPLIT , LTRIM , INSTR , SPACE , STRING , RTRIM , LEFT , LEN , MID , RIGHT , VAL , STR , CHR , CHECKSUM , HEX , HEXVAL
TCP/IP
TCP/IP routines can be used with the W3100/IIM7000/IIM7010 modules.
BASE64DEC , BASE64ENC , IP2STR , UDPREAD , UDPWRITE , UDPWRITESTR , TCPWRITE , TCPWRITESTR , TCPREAD , GETDSTIP , GETDSTPORT , SOCKETSTAT , SOCKETCONNECT , SOCKETLISTEN , GETSOCKET , CLOSESOCKET , SETTCP , GETTCPREGS , SETTCPREGS
Text LCD
Text LCD routines work with the normal text based LCD displays.
HOME , CURSOR , UPPERLINE , THIRDLINE , INITLCD , LOWERLINE , LCD , LCDAT , FOURTHLINE , DISPLAY , LCDCONTRAST , LOCATE , SHIFTCURSOR , DEFLCDCHAR , SHIFTLCD , CLS
Trig & Math
Trig and Math routines worj with numeric variables.
ACOS , ASIN , ATN , ATN2 , EXP , RAD2DEG , FRAC , TAN , TANH , COS , COSH , LOG , LOG10 , ROUND , ABS , INT , MAX , MIN , SQR , SGN , POWER , SIN , SINH , FIX , INCR , DECR , DEG2RAD
Various
This section contains all statements that were hard to put into another group
CONST , DBG , DECLARE FUNCTION , DECLARE SUB , DEFXXX , DIM , DTMFOUT , EXIT , ENCODER , GETADC , GETKBD , GETATKBD , GETRC , GOSUB , GOTO , LOCAL ,ON VALUE , POPALL , PS2MOUSEXY , PUSHALL , RETURN , RND , ROTATE , SENDSCAN , SENDSCANKBD , SHIFT , SOUND , STCHECK , SUB , SWAP , VARPTR , X10DETECT , X10SEND , READMAGCARD , REM , BITS , BYVAL , CALL , #IF , #ELSE , #EN

59 USB programmer - USBASP

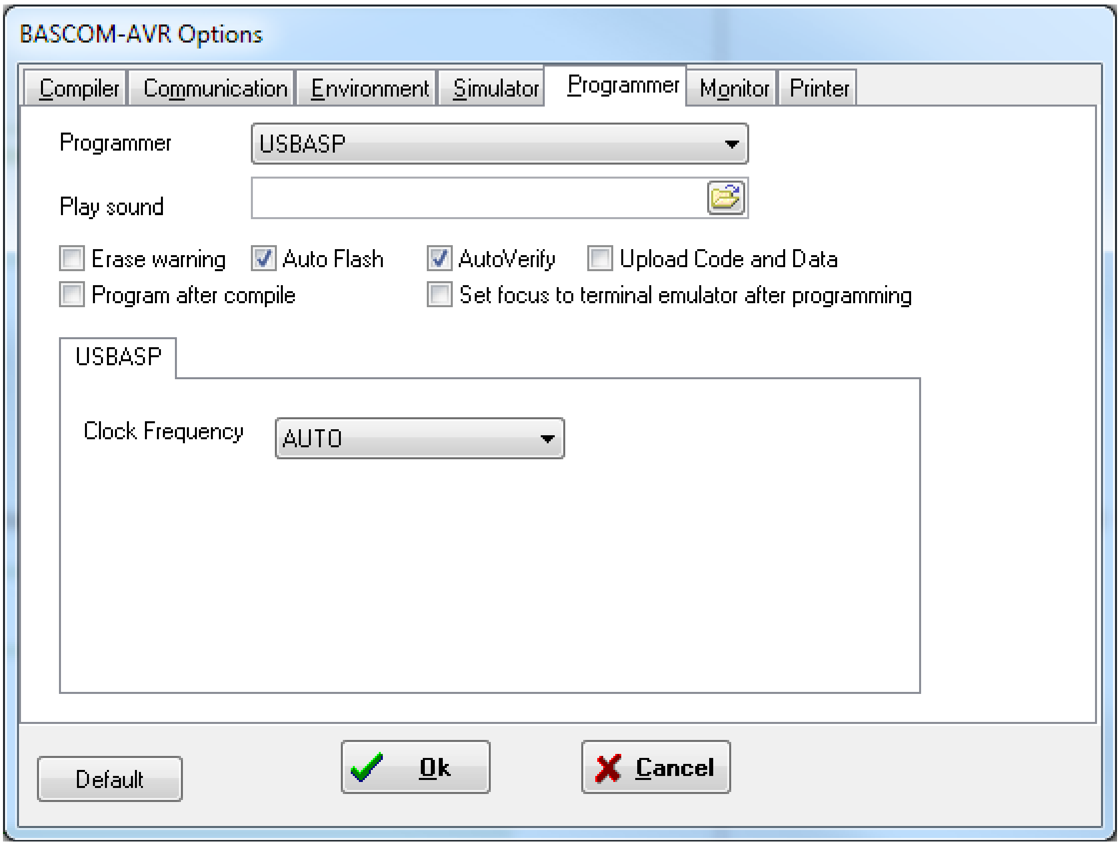
More recently we have been building the USBASP programmer from http://www.fischl.de/usbasp/
Using this PCB design



And layouts



I am currently using version 2.0.4.0 of Bascom allows you to select USBASP as a programmer, this has only worked in the later versions, so make sure you are using the latest version of Bascom.



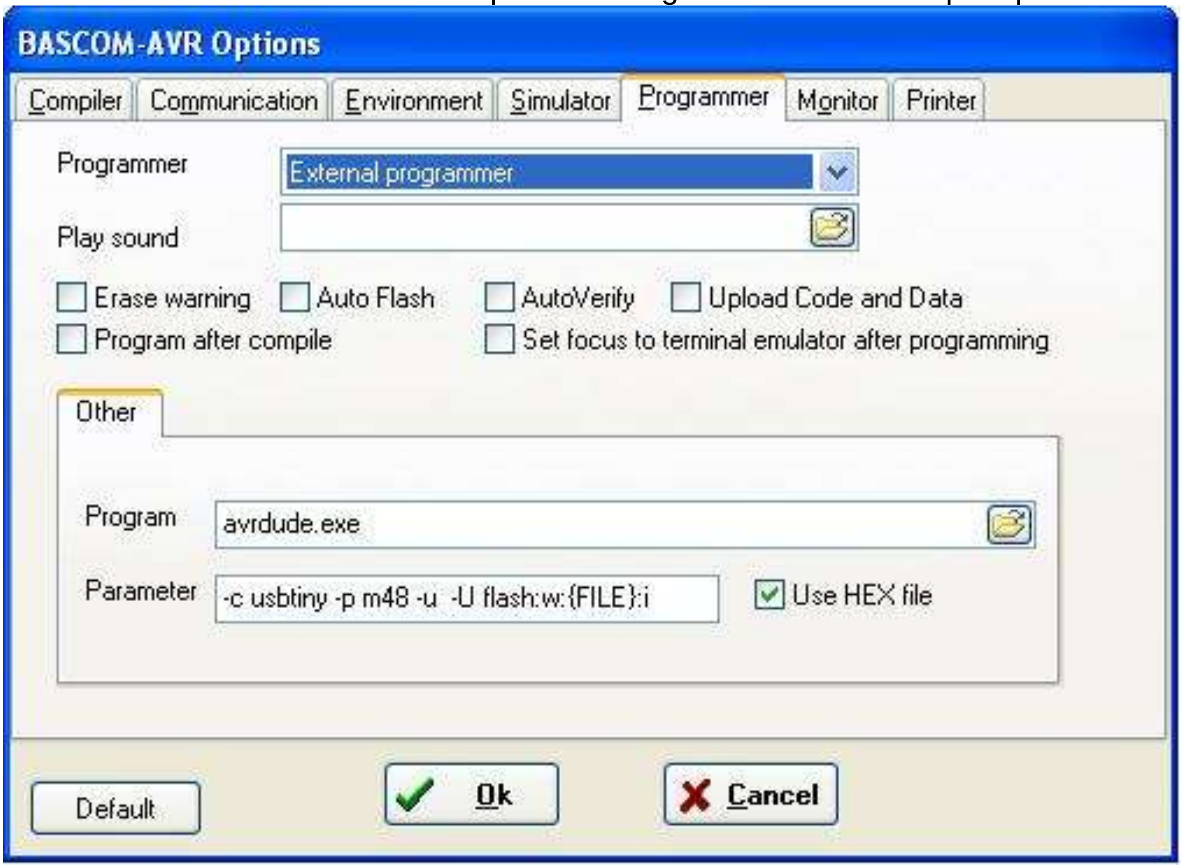
Installing drivers on Windows 7 use to be a real pain, however the latest version libusb is great.

Its actually not worth making one of these as there are some real cheap and good USBASP programmers on EBAY!!.

60 USBTinyISP programmer

A full kitset for the hardware can be purchased from http://www.adafruit.com/ or it can be built from scratch from circuits at http://www.ladyada.net/make/usbtinyisp/, or within the workshop we have eagle files for the programmer and we can program the chip.

It is easy to use and setup;
1. Install the latest version of winavr and the programming software avrdude will be installed with it.
2. You will need a USB driver get it from http://www.ladyada.net/make/usbtinyisp/download.html.
When you plug in the programmer it will ask for drivers, install them from wherever you downloaded them to.
3. Setup Bascom to start the program automatically.
From Bascom-AVR menu select Options – Compiler – Output tab and make sure hex file is selected
From Bascom-AVR menu select Options – Programmer and setit up as per the following



When you have compiled your program press F4 and it should work fine.

Notes: the –u option has been specified, this tells AVRDUDE not to read the fusebits and not to set them. The default option (not using –u) reads the fusebits and rewrites them again. This means that if there is a glitch in the programming the fuse bits could be overwritten with something that doesn't work well and your micro becomes unuseable!! This has been experienced first hand so always use the –u option!!

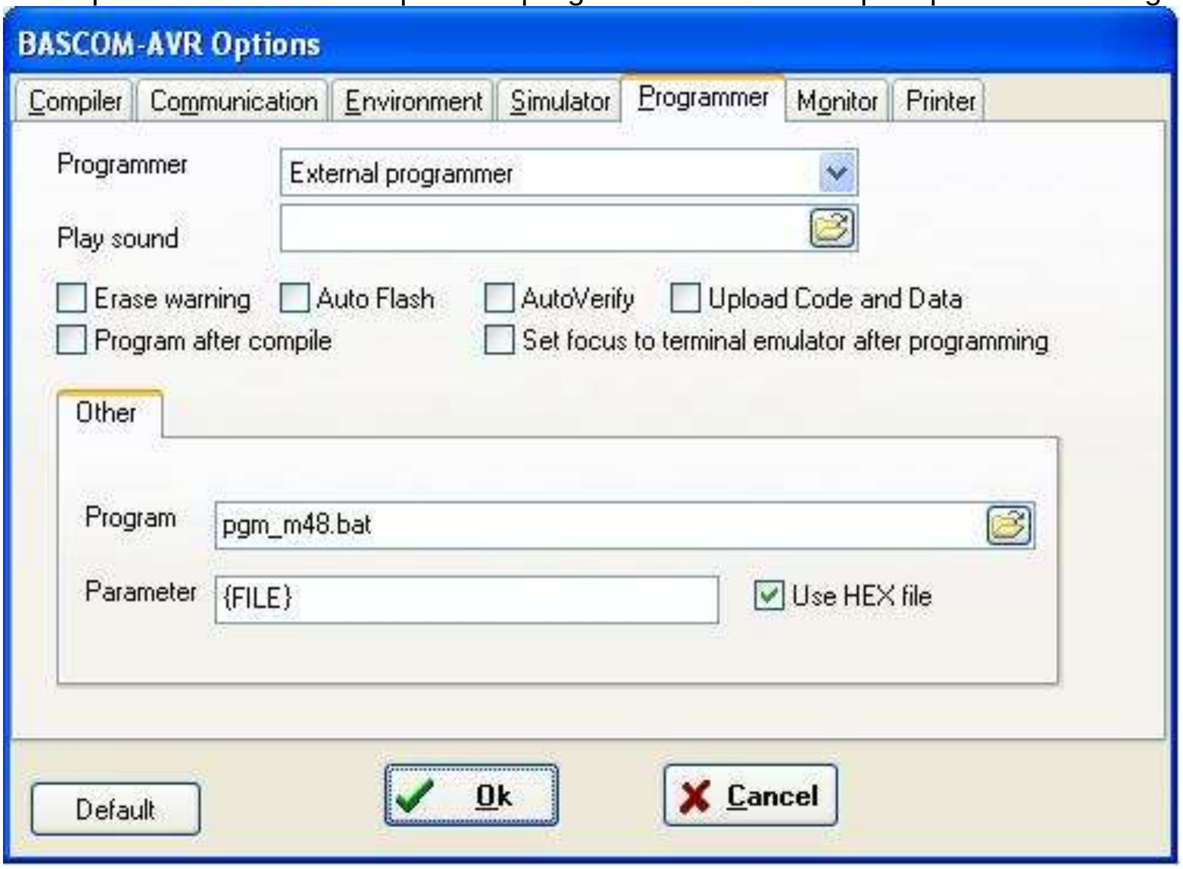
An issue with this process is that you get no feedback from BASCOM that the programming has worked (or not) as the command window appears and then rapidly disappears.

An alternative to the above setup is to create a small batch (text file).
Here is one called **pgm_m48.bat** for programming the ATMEGA48.

Open Windows Notepad and copy these two lines into it and save it in the c:\winavr\bin directory.
avrdude -c usbtiny -p m48 -u -U flash:w:%1:i
pause

(%1 is used to refer to the first parameter that is passed to the batch file in this case the name of the hex file created by Bascom)

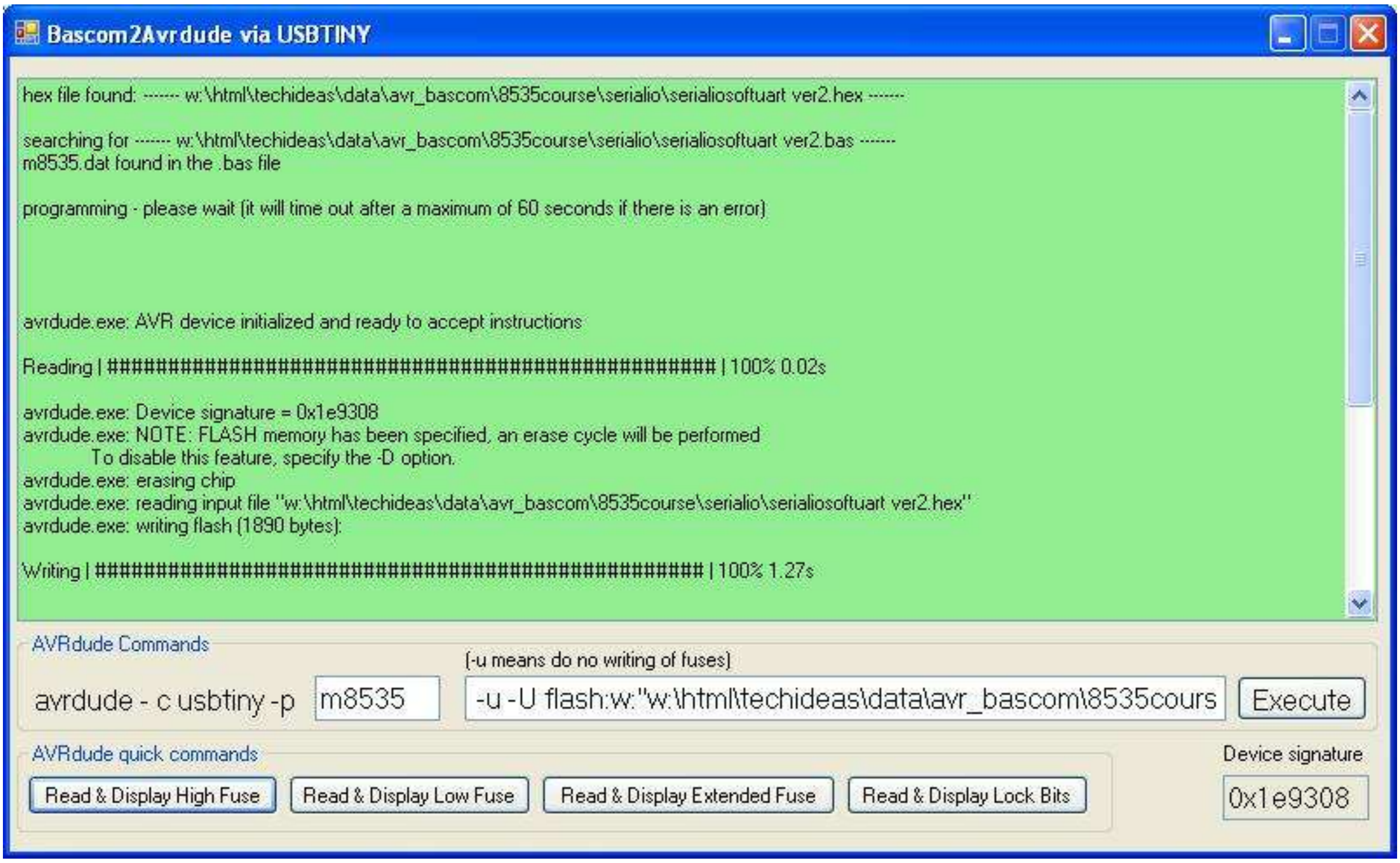
Next open Bascom-AVR – options – programmer and set it up as per the following



After you press F4 to program the micro the command window appears and will stay open after programming so that you can see the program output. It is closed by pressing any key. If you are using different AVRs just great different batch files. You will need to change the batchfile selected in the programming options when you change AVR type (that's only a small inconvenience though).

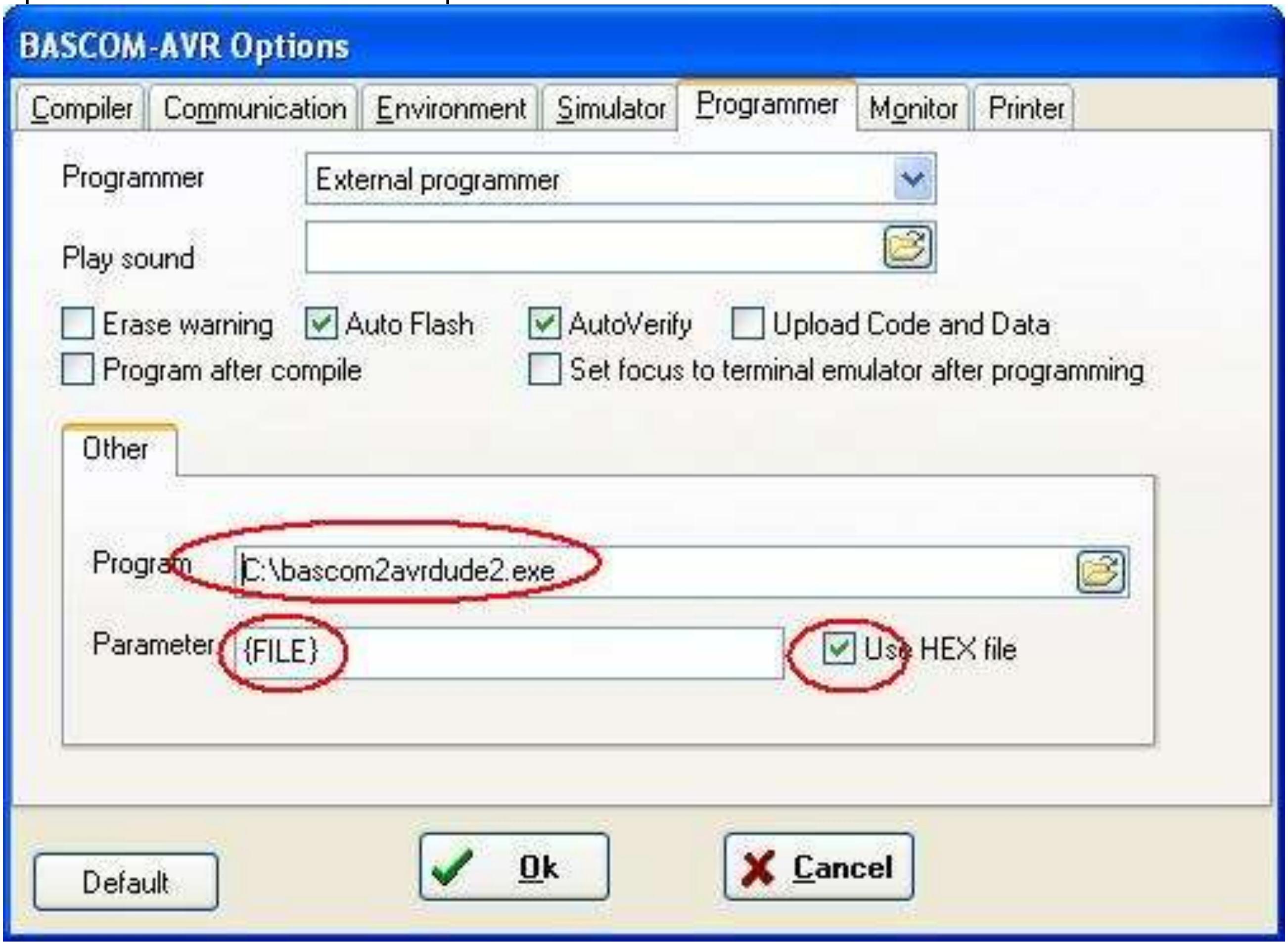
You will need to create different batch files for each different chip.
OR>>>

There is a third option, it is to use a program I have written bascom2avrdude2.exe that handles programming nicely for you.



Download the executable file (you must have dotnet3.5 installed to use it)
There is no install, just save it somewhere like the root of C:\.

Open Bascom and add it to the path like this

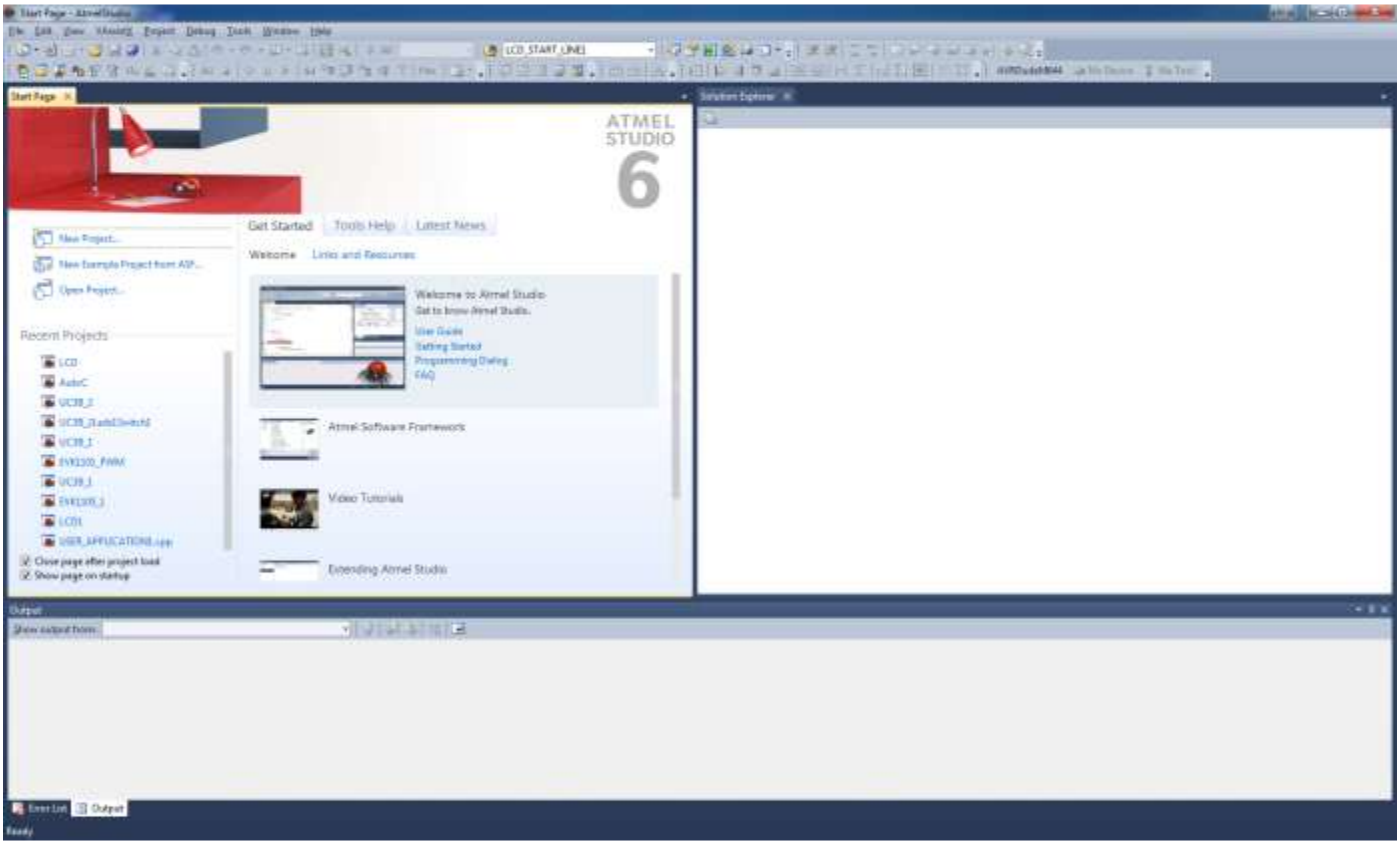


Whenever you press F4 to program the microcontroller, the bascom2avrdude2 window will open and try to program your chip using avrdude. If it gives you a green textbox, it programmed successfully, if it gives you another colour then there was an error and a messagebox will give an idea as to what went wrong.

Happy programming…

61 C-Programming and the AVR

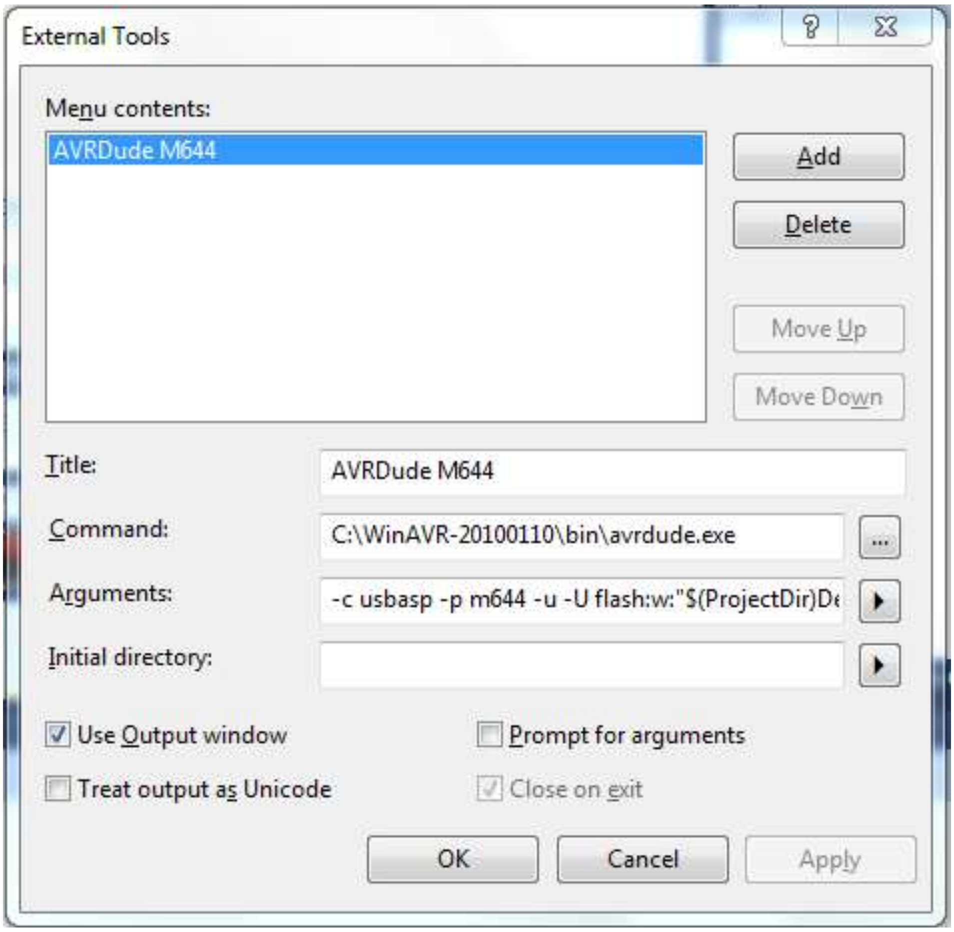
It is no problem at all to jump into C programming for the AVR.
First download AtmelStudio; you can use WinAVR as well but here it will be AtmelStudio we will focus on.
Download and install it from www.atmel.com, this tutorial will use Version 6.



There are many useful tutorials on the internet about this so briefly.

61.1 Configuring a programmer

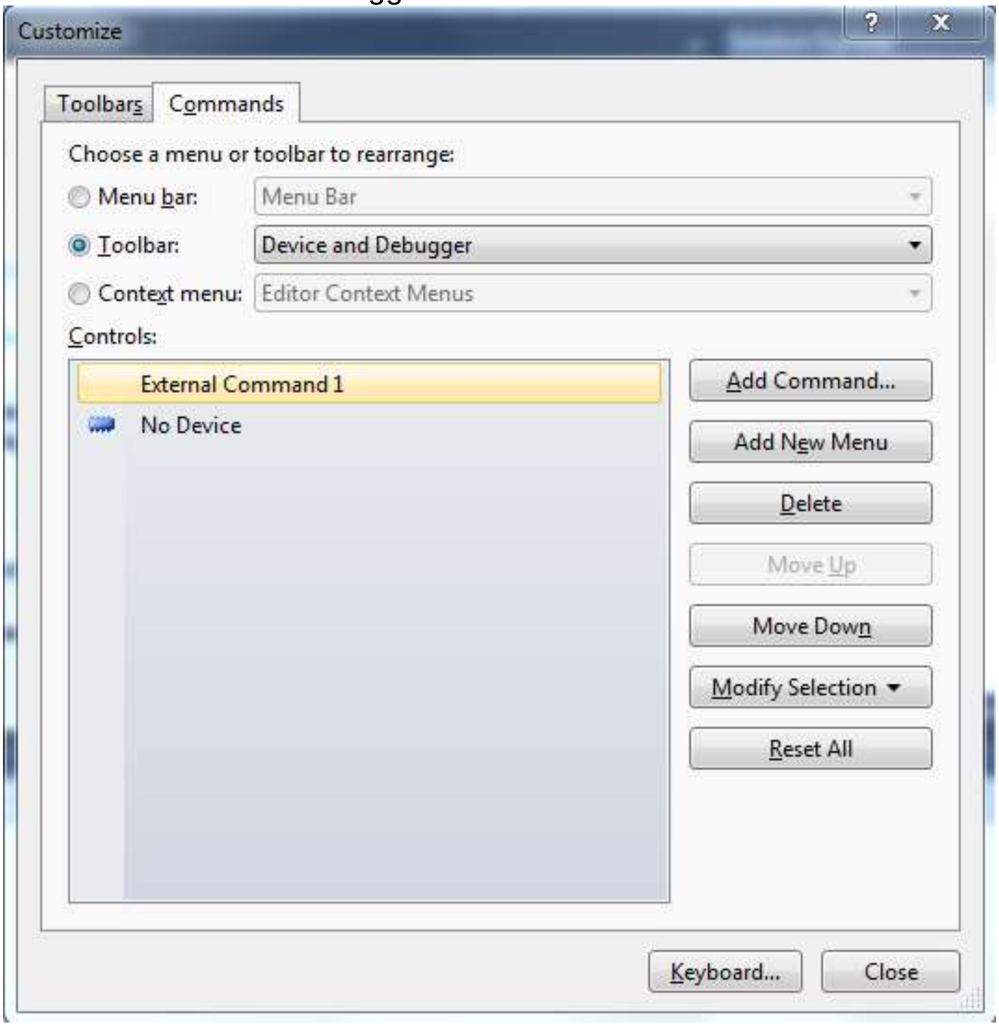
Atmel Studio does not include the USBasp programmer but it can be added as a button on the toolbar. Atmel Studio will call AVRDude to do the actual programming.
If you have installed WinAVR as well you wont need to down load AVRDude programing software.
On the Tools menu select External Tools and add a new tool, the first micro is the Mega644 so it will be labelled AVRDudeM644



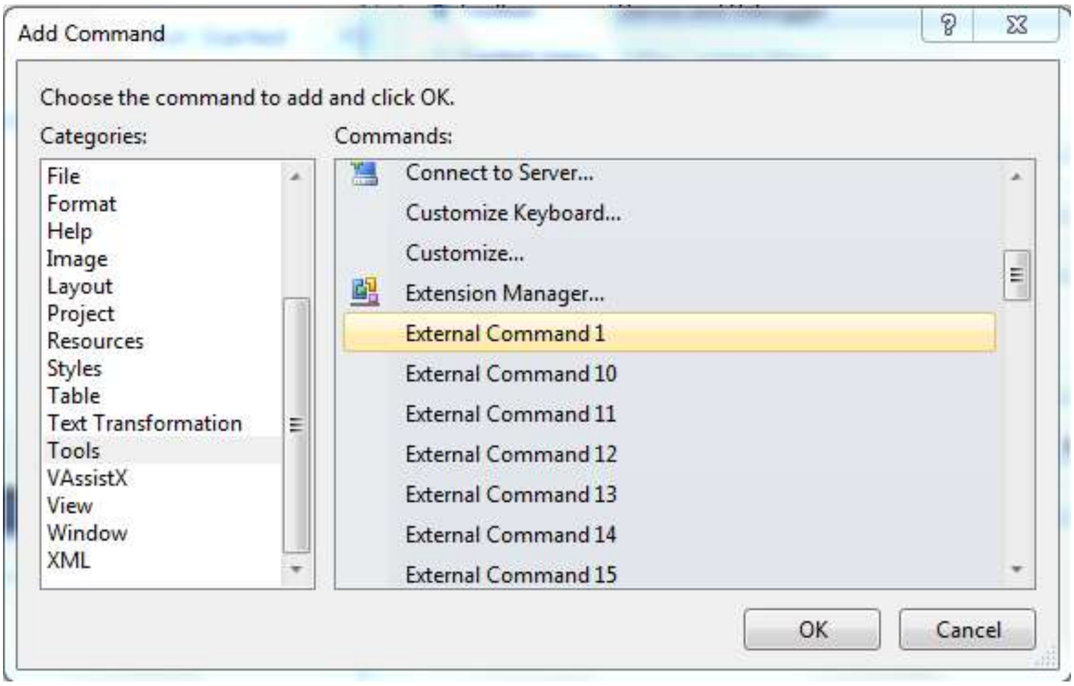
The arguments line is:
-c usbasp -p m644 -u -U flash:w:"$(ProjectDir)Debug\$(ItemFileName).hex":i
Remember to select Use Output window so that the results from AVRDude can be seen.
The board can now be programmed from the tools menu.

Because Atmel Studio cannot pass the part number to AVRDude it must be in the arguments line. So a new tool will be created for each device used.

Right click on the toolbar and select the specific toolbar where the button should appear. In this case the device and debugger toolbar is chosen.



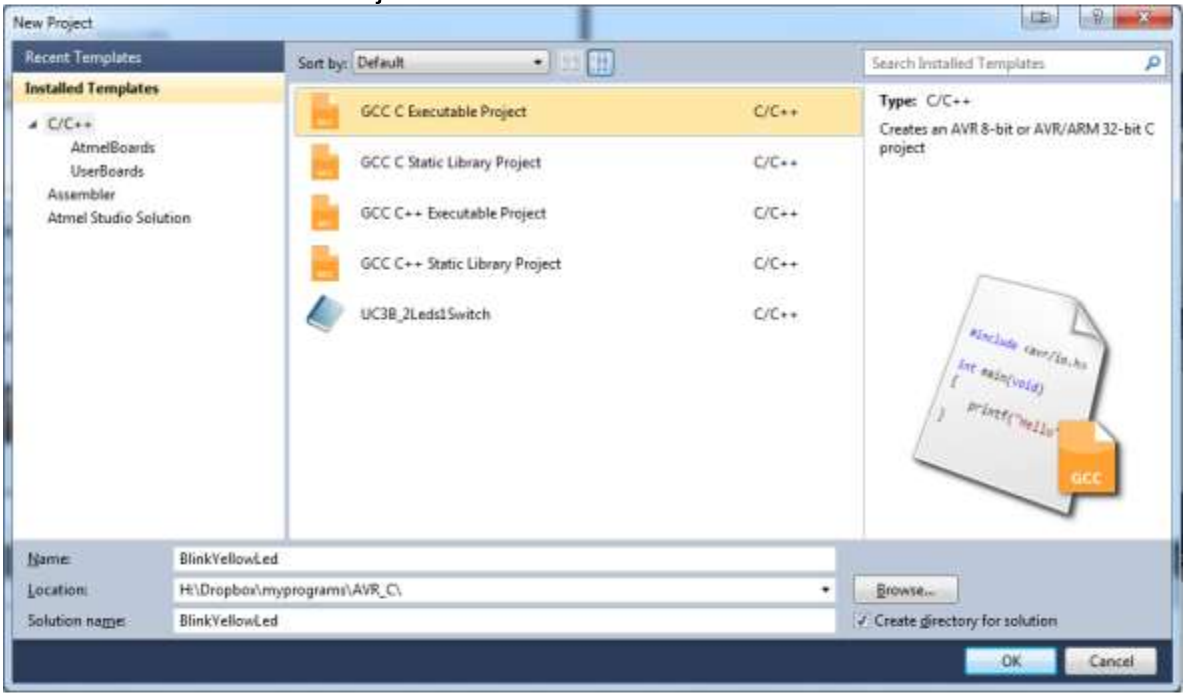
Choose Add Command



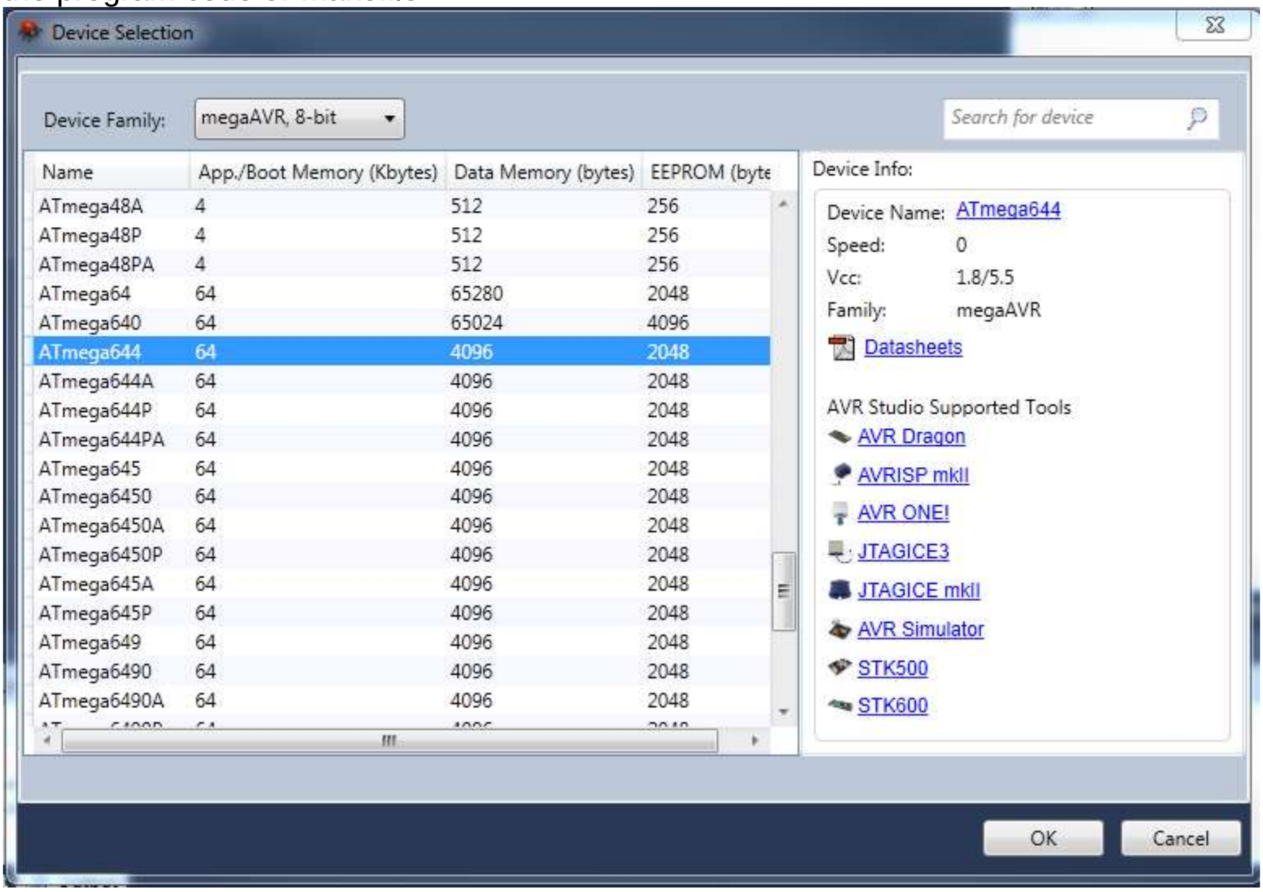
The AVRDudeM644 tool will not appear as a named item but will be External Command1 under the Tools category. After selecting OK it can be renamed; then a button for it will appear in the toolbar

61.2 First program

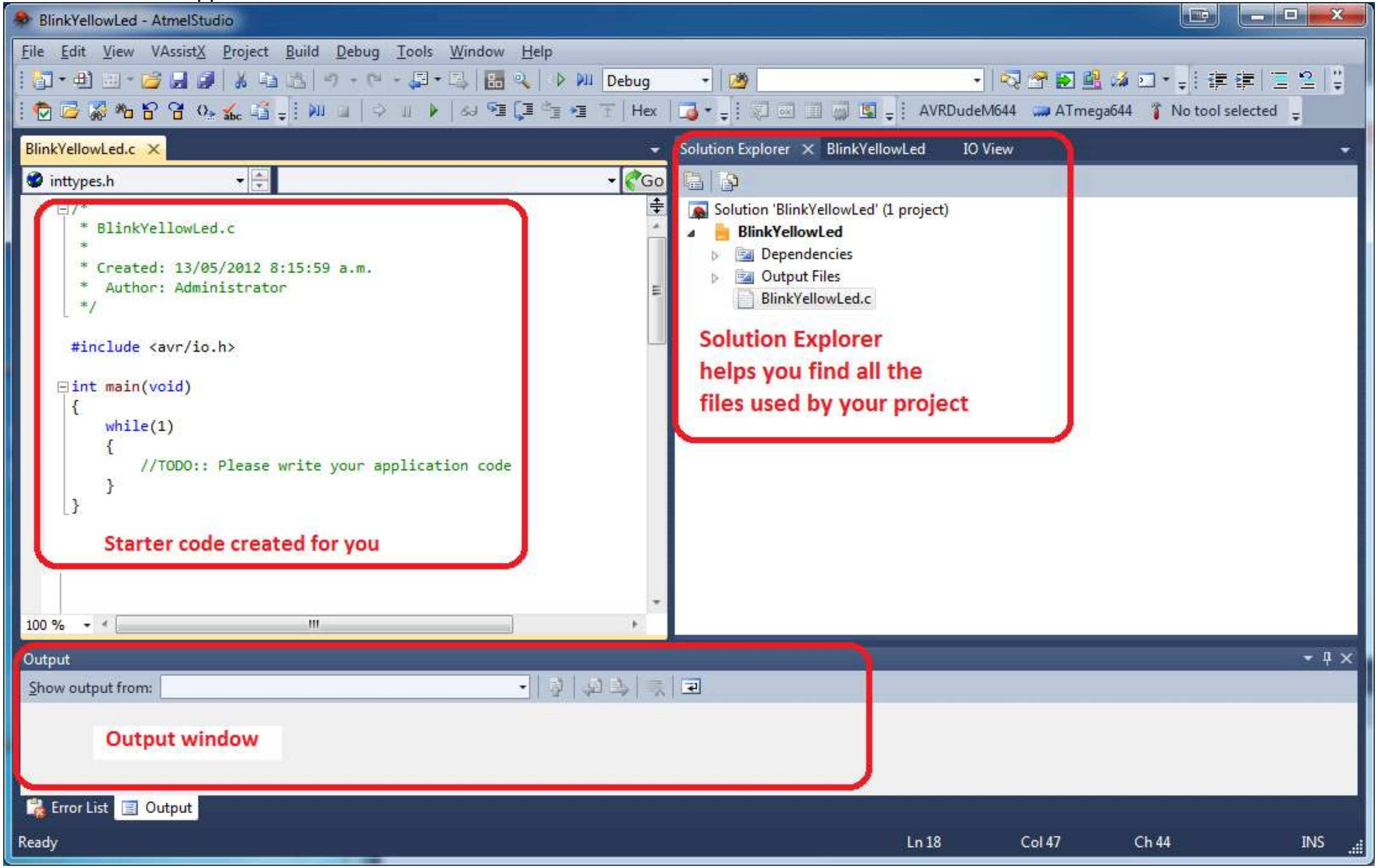
Choose File then New Project



Choose C/C+ and AVRGCC C Executable Project
I store all my programs in a Dropbox folder; c programs go into the C folder under C:\DATA\Dropbox\myprograms\C\
This project will be called BlinkYelLed
After clicking Ok choose your device, in this case the ATMEGA644
Atmel Studio will now look after device selection for you, you don't need to add it anywhere else in the program code or makefile.

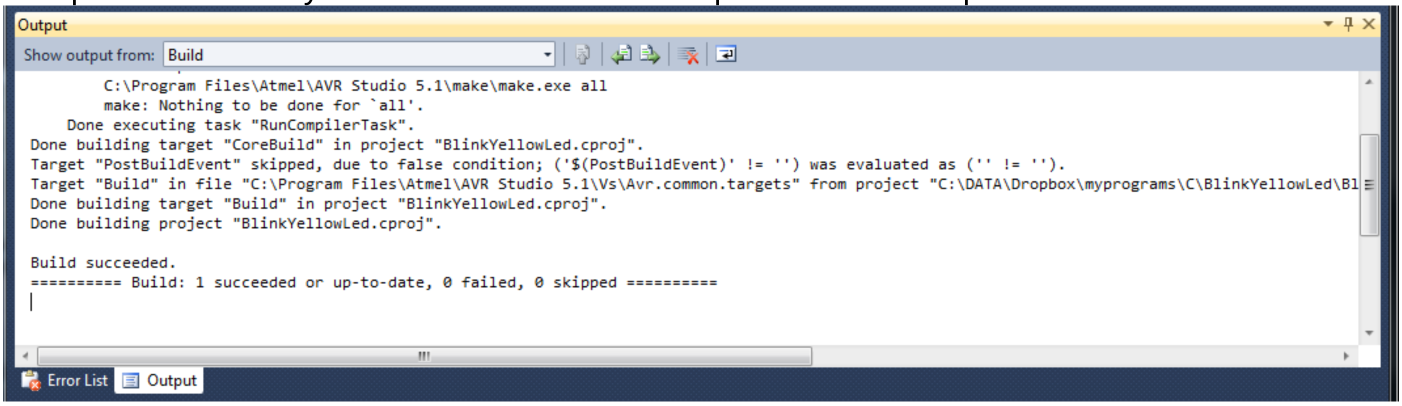


The main window appears



61.3 Output window

This program can be compiled and programmed into the AVR (but it wont do anything yet)
Compile with F7 and you will see the result of compilation in the Output window.

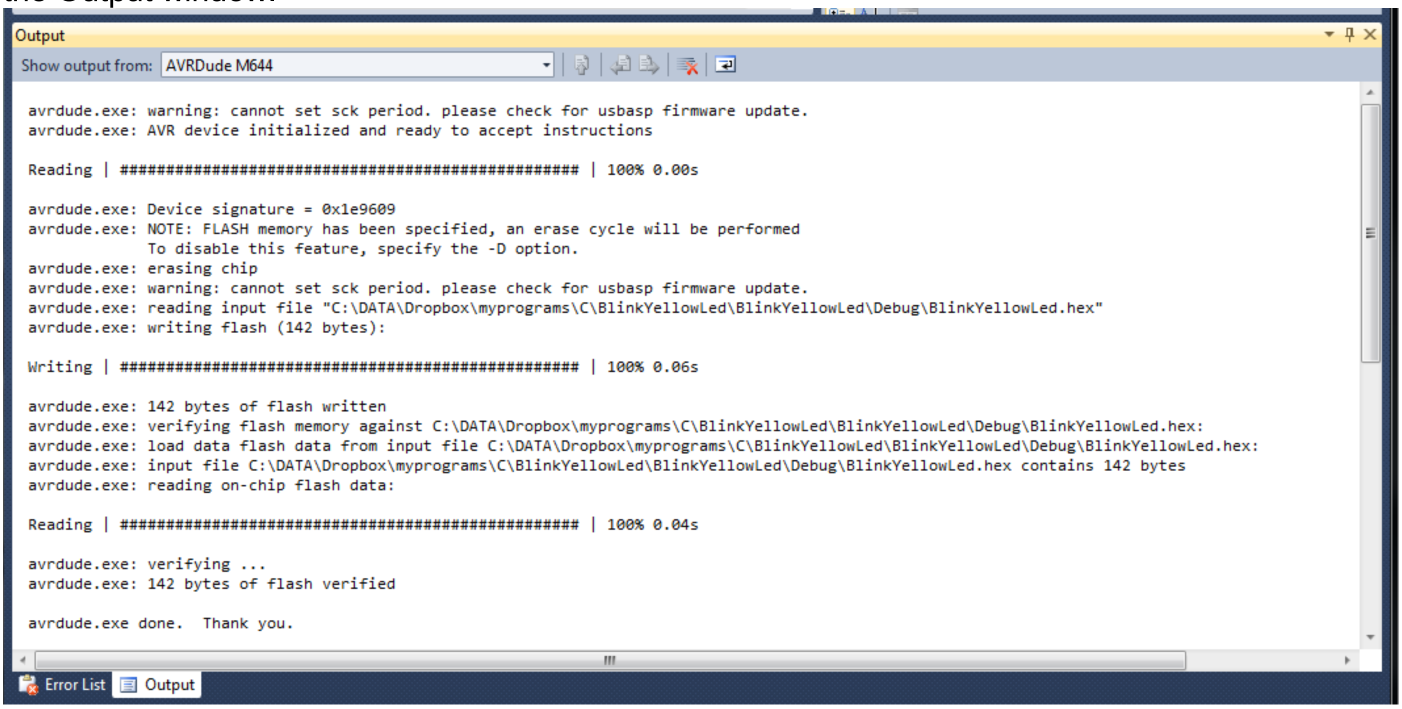


```
Output
Show output from: Build
        C:\Program Files\Atmel\AVR Studio 5.1\make\make.exe all
        make: Nothing to be done for `all'.
    Done executing task "RunCompilerTask".
Done building target "CoreBuild" in project "BlinkYellowLed.cproj".
Target "PostBuildEvent" skipped, due to false condition; ('$(PostBuildEvent)' != '') was evaluated as ('' != '').
Target "Build" in file "C:\Program Files\Atmel\AVR Studio 5.1\Vs\Avr.common.targets" from project "C:\DATA\Dropbox\myprograms\C\BlinkYellowLed\Bl
Done building target "Build" in project "BlinkYellowLed.cproj".
Done building project "BlinkYellowLed.cproj".

Build succeeded.
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
Error List   Output
```

An important thing to know here is that Atmel Studio creates a makefile for you. In other programming environments (e.g. WinAVR) you will need to create your own (System Designer software can create one for you or you can download one and modify it).

The think to look out for when compiling are any errors, and always check the syntax (correct spelling and use of symbols) of your program code.

Select the tool button you created to program your chip and you will see the results of AVRDude in the Output window.



```
Output
Show output from: AVRDude M644

avrdude.exe: warning: cannot set sck period. please check for usbasp firmware update.
avrdude.exe: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.00s

avrdude.exe: Device signature = 0x1e9609
avrdude.exe: NOTE: FLASH memory has been specified, an erase cycle will be performed
             To disable this feature, specify the -D option.
avrdude.exe: erasing chip
avrdude.exe: warning: cannot set sck period. please check for usbasp firmware update.
avrdude.exe: reading input file "C:\DATA\Dropbox\myprograms\C\BlinkYellowLed\BlinkYellowLed\Debug\BlinkYellowLed.hex"
avrdude.exe: writing flash (142 bytes):

Writing | ################################################## | 100% 0.06s

avrdude.exe: 142 bytes of flash written
avrdude.exe: verifying flash memory against C:\DATA\Dropbox\myprograms\C\BlinkYellowLed\BlinkYellowLed\Debug\BlinkYellowLed.hex:
avrdude.exe: load data flash data from input file C:\DATA\Dropbox\myprograms\C\BlinkYellowLed\BlinkYellowLed\Debug\BlinkYellowLed.hex:
avrdude.exe: input file C:\DATA\Dropbox\myprograms\C\BlinkYellowLed\BlinkYellowLed\Debug\BlinkYellowLed.hex contains 142 bytes
avrdude.exe: reading on-chip flash data:

Reading | ################################################## | 100% 0.04s

avrdude.exe: verifying ...
avrdude.exe: 142 bytes of flash verified

avrdude.exe done.  Thank you.
Error List   Output
```

61.4 Configuring inputs & outputs

In any AVR program you will have to add configuration for your I/O ports. This code was auto generated from System Designer but is not hard to write your own once you get used to it.

```
int main(void)
{
    //hardware setups
    DDRA = 0xff;        //make port all outputs
    DDRB = 0xff;        //make port all outputs
    DDRC = 0xff;        //make port all outputs
    DDRD = 0xff;        //make port all outputs
    DDRB &=~_BV(0);      //set pin B.0 to input  - Red_sw
    DDRB &=~_BV(1);      //set pin B.1 to input  - Yel_sw
    DDRB &=~_BV(2);      //set pin B.2 to input  - Grn_sw
    DDRB &=~_BV(3);      //set pin B.3 to input  - Blu_sw
    DDRB &=~_BV(4);      //set pin B.4 to input  - Wht_sw
    DDRA &=~_BV(0);      //set pin A.0 to input  - POT
    DDRA &=~_BV(1);      //set pin A.1 to input  - LM35
    DDRA &=~_BV(2);      //set pin A.2 to input  - LDR
    DDRA &=~_BV(4);      //set pin A.4 to input  - Ser_Rx

    while(1)
    {
        //TODO:: Please write your application code
    }
}
```

DDRA – data direction register for PORTA; every AVR port (group of 8 pins) has 3 separate registers to control and access it (a register is an address inside the microcontroller that has direct control over the internal hardware).
DDRA register is used to control whether a pin is input or output.
PORTA register is used to change the devices attached to the pins when the pin is an output.
PINA register is used to read the pins when the pins are configured as inputs.
A really good tutorial on this is at http://iamsuhasm.wordpress.com/tutsproj/avr-gcc-tutorial/

Note that in C upper and lower case is very important DDRA is not the same as ddra.

To make pins into outputs we put a 1 into each bit of the DDR. So **DDRA=0xff;** means put hexadecimal FF into DDRA it could be written in binary instead of hexadecimal as **DDRA=0b11111111;**

61.5 Making a single pin an input

To set an individual pin (e.g. PortB.5) to an input that bit in the DDR must be set to '0'. To do that use any of these lines of code

```
DDRB &= ~(1<<5);          //uses right shift
DDRB &= _BV(5);           //macro use
DDRB &= 0b00100000;       // binary
DDRB &= 0x20              // hexadecimal
DDRB &= 32                // decimal
```

Here there are two bitwise operations the '**~**' (not) and the '**&**' (and)

Bitwise & means individually 'and' each bit of the byte.
The rules for an 'and' are written into a table like this.
A is one input, B is the other input and X is the output

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

This can be generalised to 2 rules: 'and'ing anything with a '0' makes the output a '0', 'and'ing with a 1 keeps the output the same as the input.

'Not' or **~** means inverse something

A	X
0	1
1	0

1<<5	0	0	**1**	0	0	0	0	0
~(1<<5)	1	1	**0**	1	1	1	1	1
DDRB before	1	1	**1**	1	1	1	1	1
DDRB after	1	1	**0**	1	1	1	1	1

The effect of this is force pin 5 to low (to be an input) and to keep the others unchanged.

61.6 Making a single pin an output

To set an individual pin (e.g. PortB.5) to an output we must make that bit in the DDR a '1' to do that we can use any of these lines of code

DDRB |= (1<<5); **//uses right shift**
DDRB |= _BV(5)); **//macro use**
DDRB |= 0b00100000; **// binary**
DDRB |= 0x20 **// hexadecimal**
DDRB |= 32 **// decimal**

_BV(5) is a macro in C and when _BV(5) is found in a program it is replaced with (1<<5)
So those 2 lines of code are actually exactly the same.

Here are some crucial understandings in C that you will use lots and lots in programs.

DDRB |= (1<<5);

First **(1<<5)** means take a byte with 1 in it (0b00000001) and shift the '1' 5 places to the left so it becomes 0b00100000. The reason it's in brackets is that we want it to happen as the first step in the execution of this line of code.

So we could rewrite it now to **DDRB |= 0b00100000;**
This code is actually a C short cut way of writing **DDRB = DDRB | 0b00100000;**
C uses this concept a lot **X += 1;** means **X = X + 1**; **hour -= 2;** means **hour = hour - 2;**

The | is the symbol in C for '**bitwise or**' we are going to do a **bitwise 'or'** between the current contents of the register DDRB and the number 0b00100000 and put the answer back into DDRB.

'Bitwise Or' means individually '**or**' each bit of the byte.
The rules for an 'or' written into a table are this.
A is one input, B is the other input and X is the output

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

This can be generalised to 2 rules: 'or'ing anything with a '1' makes the output a '1', 'or'ing with a 0 keeps the output the same as the input.

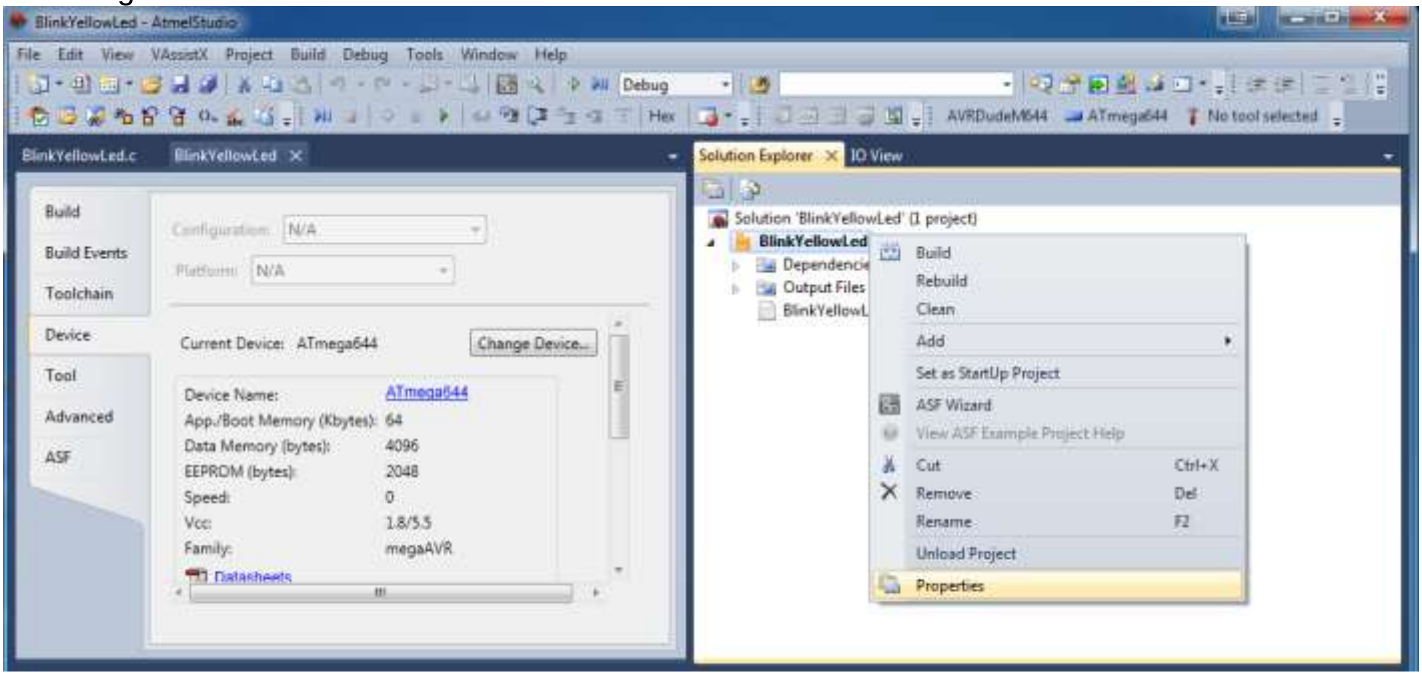
If DDRB was 0b11001000 and we 'or' it with 0b00100000 then we get

DDRB before	1	1	**0**	0	1	0	0	0
	0	0	**1**	0	0	0	0	0
DDRB after	1	1	**1**	0	1	0	0	0

The effect of this is to force bit 5 in DDRB to be a 1 (force it to be an output), and keep the other bits of DDRB unchanged.

61.7 Microcontroller type

The compiler needs to know some things about our hardware; the first is the microcontroller type. That was setup at the beginning when the project was created however it can be changed by right clicking on the project in the Solution Explorer and opening properties. The tabs on the side of the project properties allow different aspects ot be seen of the project. Under Device the micro type can be changed.



61.8 Includes

Understanding what happens with includes is a crucial part of C programming.

```
#include <avr/io.h>
```

If the compiler cannot find a function that you used in your program code it will go looking in the file io.h (h stands for header file) which is in the avr directory. Try and find the AVR directory on your system, but don't scare yourself too much by looking inside the file.

On my system this file is in:
C:\ProgramFiles\Atmel\Atmel Studio 6.0\extensions\Atmel\AVRGCC\3.4.0.65\AVRToolchain\avr\include\avr

You will find lots of other files with many functions that will have future use to you.

61.9 Main function

```c
int main(void)
{


	while(1)
	{
		//TODO:: Please write your application code
	}
}
```

Functions are the core structure in programming; the ‘main’ program in C is where program execution starts.
A function can be passed arguments or parameters (in this case none so the word void is used) and it returns a value when finished executing (a value of type ‘int’)

The braces enclose everything within a function in C.

The **While(1)** means while everything inside the brackets () is true repeat everything inside the braces{}. Sometimes you will see this written in programs as **for(;;)** which effectively means the same thing.

61.10 The blinkyelled program

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>

int main(void)
{
    //hardware setups
    DDRA = 0xff;        //make port all outputs
    DDRB = 0xff;        //make port all outputs
    DDRC = 0xff;        //make port all outputs
    DDRD = 0xff;        //make port all outputs
    DDRB &=~_BV(0);      //set pin B.0 to input  - Red_sw
    DDRB &=~_BV(1);      //set pin B.1 to input  - Yel_sw
    DDRB &=~_BV(2);      //set pin B.2 to input  - Grn_sw
    DDRB &=~_BV(3);      //set pin B.3 to input  - Blu_sw
    DDRB &=~_BV(4);      //set pin B.4 to input  - Wht_sw
    DDRA &=~_BV(0);      //set pin A.0 to input  - POT
    DDRA &=~_BV(1);      //set pin A.1 to input  - LM35
    DDRA &=~_BV(2);      //set pin A.2 to input  - LDR
    DDRA &=~_BV(4);      //set pin A.4 to input  - Ser_Rx

    while(1)
    {
        PORTB &= ~( 1 << PORTB6 );      // drive PB6 low
        _delay_ms( 900 );               // delay 900 ms
        PORTB |= 1 << PORTB6;           // drive PB6 high
        _delay_ms( 100 );               // delay 100 ms

    }
}
```

`#define F_CPU 8000000UL`

This is a macro that will be used by the compiler to calculate delay loops, and states it to be 8MHz, without this line the program defaults to some other value (1000000) and all the timing would be wrong.

`#include <util/delay.h>`

This says to the program to include any functions from this file that we use in the main program.

`PORTB &= ~(1 << PORTB6);      // drive PB6 low`

This line is the same as earlier for driving a DDR pin low, but this time we use PORTB6; PORTB6 is another macro and just means 6.

So these lines of code are all the same

PORTB &= ~(1<<PORTB6);
PORTB &= ~(1<<6);
PORTB &= ~_BV(6);
PORTB &= 0b01000000;
PORTB &= 0x40;
PORTB &= 64;

Why did I try and confuse you with all these at once, well that’s because when you look on the internet you will see most of them and one of them is no more correct that another (just some are easier to read).

61.11 Counting your bytes

```
_delay_ms( 900 );                    // delay 900 ms
```

This is a function call and the compiler will look for the function in the included files.
It is in the util/delay.h
Also in util/delay.h is another function _delay_us which you can use (microseconds)

If you look inside the delay.h file the start of the function is

void _delay_ms(double __ms)

This means you can pass a big number to the function, a double is an 8byte number in C and can include decimals. The void means that when the function is finished it doesn't return any value to the function that called it.

It is a bit silly to use the _delay_ms routine with an AVR as all we want is a simple delay, using doubles where we don't need them can create a larger program than we want.

Inside the delay.h file is another include to delay_basic.h
There are two routines in there that delay can use

void _delay_loop_1(uint8_t __count)
void _delay_loop_2(uint16_t __count)

uint8_t is an unsigned 8 bit number (a byte – stores numbers from 0 to 255
uint16_t is an unsigned 16bit number (2 bytes – stores numbers from 0 to 65535)

Now we could happily go on using _delay_ms as a routine and to be honest it doesn't use a lot more memory than the alternative at this stage but it is important when programming microcontrollers to really understand what is going on and make informed decisions about what you program code is doing. So why would you use a routine that takes a double when a uint8_t and uint16_t are available.

Changing the program to use _delay_loop_2 saves us 4 bytes of program code.

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay_basic.h>
#include <inttypes.h>

int main(void)
{
	//hardware setups
	DDRB = 0xff;	//make port all outputs

	uint16_t count;

	while(1)
    {
	PORTB &= ~( 1 << PORTB6 );              // drive PB6 low
	for (count=900; count >0; count --)
	{
		_delay_loop_2(1000);
	}
	PORTB |= 1 << PORTB6;                          // drive PB6 high
	for (count=100; count >0; count --)
	{
		_delay_loop_2(1000);
	}
    }
```

```
}
```

Note that we have included the new file inttypes.h, otherwise our compiler will not know what an unit16_t means.

We have now declared our first variable as well

```
uint16_t count;
```

and we have created our own delay loop

```
    for (count=900; count >0; count --)
    {
        _delay_loop_2(1000);
    }
```

Begin to get use to the way C for loops are written.
This loop means start the variable count at 900 (count=900) and while it is greater than 0 (count>0) decrease it by 1 (count --)

We could have written it for (count =0; count <900, count++) but generally it's is better to count down to 0 rather than count up as microcontrollers have simpler comparisons to do when they compare to 0 rather than other numbers. Using this up counting loop is actually more costly in terms of flash (program memory).

```
_delay_loop_2(1000);
```

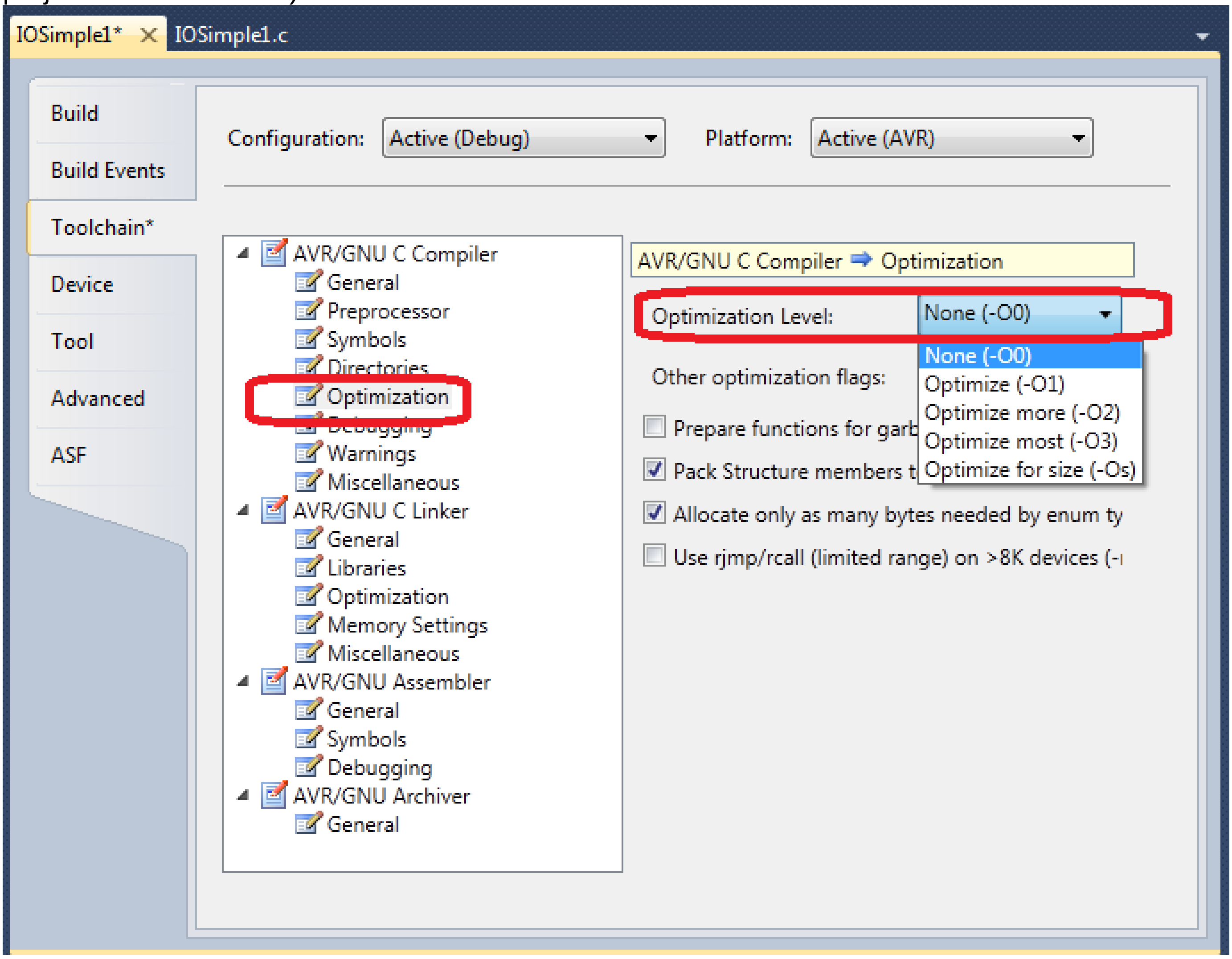
Now this _delay_loop_2(1000) is only approximately 1mS when the crystal is 8MHz, in fact it is ok for flashing an LED but not really very accurate. Also if you change your crystal then this will be way off.

That is the reason the function _delay_ms is often used because it hides all the calculations from us when trying to create an accurate delay based upon the crystal frequency. And the reason it needs to be a double is so that it can do more accurate divisions when trying to work out more exact values.

61.12 Optimising your code

GCC can create programs which are highly optimised (have all the unneeded bits reduced down or even taken out.

Open the project properties by right clicking on the project in the solution window (right click on the project not the solution).



Slect optimization and then choose between the different levels and recompile for each one.

This program when compiled gave these different sizes based upon the optimization setting:

- None or -O0 as 408 bytes,
- -O1, -O2, -O3 at 208 bytes
- -Os at 214 bytes.

Note that if you are simulating always change to –O0 no optimization.

61.13 Reading input switches

```
while(1)
{
    if(~PINB & (1<<1))
          {
                PORTB &= ~( 1 << PORTB6 );              // drive PB6 low
                for (count=0; count <900; count ++)
                {
                        _delay_loop_2(1000);
                }
                PORTB |= (1 << PORTB6);                        // drive PB6 high
                for (count=100; count >0; count --)
                {
                        _delay_loop_2(1000);
                }
          }
}
```

Here we want the led to flash only when the switch is pressed.

```
if(~PINB & (1<<1))
```

Means read the state of PINB invert this PINB then 'and' it with 0b00000001

Here is the result when the switch is not pressed

PINB	1	0	1	0	1	1	**1**	1	**Read the port (1 is not pressed)**
~PINB	0	1	0	1	0	0	**0**	0	**Invert the port**
(1<<1)	0	0	0	0	0	0	**1**	0	**Shift 1 to the left 1 time**
result	0	0	0	0	0	0	**0**	0	**AND the two numbers, Answer is 0 so if test is not true**

Here is the result when the switch is pressed

PINB	1	0	1	0	1	1	**0**	1	**Read the port (0 is swtch pressed)**
~PINB	0	1	0	1	0	0	**1**	0	
(1<<1)	0	0	0	0	0	0	**1**	0	
result	0	0	0	0	0	0	**1**	0	**AND the two numbers, Answer is 1 so the IF test is true**

Note that way single bits are set in C programming

1 << 0 == 1
1 << 1 == 2
1 << 2 == 4
1 << 3 == 8
1 << 4 == 16
1 << 7 == 128

61.14 Macros

C programs can be a little difficult for new programmers to follow so it helps to add some short cuts. Macros allow just that; we can replace hard to read lines of code like

```
PORTB &= ~(1<<6);       // drive PB6 low
PORTB |= (1<<6);        // drive PB6 high
```

With code such as

```
Clr_yel_led
Set_yel_led
```

We do that with #define statements at the beginning of the program code

```
//Hardware Macros for output ports
#define set_Yel_Led PORTB |= (1<<6)     //force portb.6 high
#define clr_Yel_Led PORTB &= ~(1<<6)    //force portb.6 low
```

Macros can be used for input testing as well

```
if(~PINB & (1<<1))
if(PINB & (1<<1))
```

can be replaced with

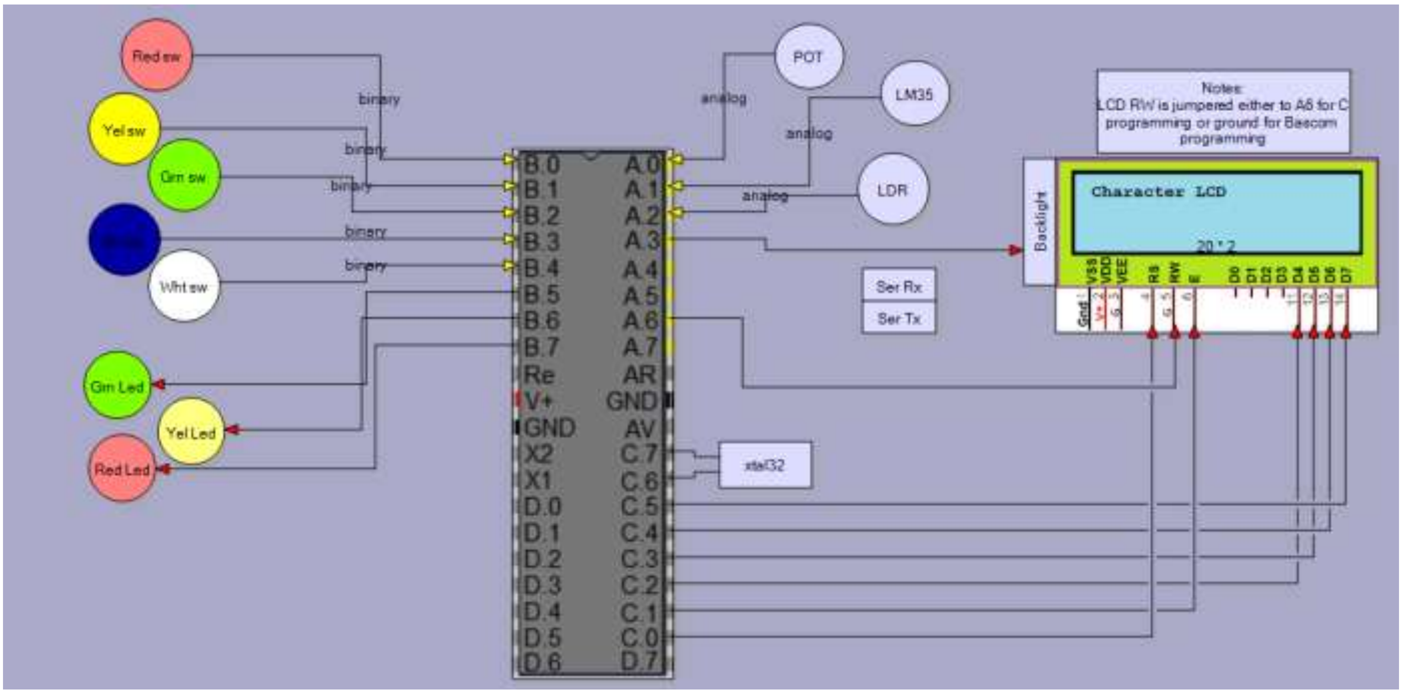
```
if (yel_sw_is_clr)
if (yel_sw_is_set)
```

The #define statements for these are

```
//Hardware Macros for input pins
#define yel_sw_is_clr ~PINB & (1<<1)     //pinb.1 input low
#define yel_sw_is_set PINB & (1<<1)      //pinb.1 input high
PORTB |= (1<<1);      //activate pinb.1 internal pull-up resistor
```

When you use System Designer software to develop your block diagram this code can be auto generated for you.

61.15 Auto-generated config from System Designer



The above block diagram generated the following code

```
//***********************************************
// Project Name: 12TCEDemoBoard
// created by:
// using block diagram: BD_1
// Date:17/05/2012 6:10:44 a.m.
// Code autogenerated by System Designer from www.techideas.co.nz
//***********************************************
// Comment next line if using WINAVR and CPU Speed is in the Makefile
#define F_CPU 8000000UL
//***********************************************
#include <avr/io.h.>
#include <inttypes.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <stdio.h>
#include <string.h>
#include <avr/pgmspace.h>
//#include <lcd.h>

int main (void)
{
    //***********************************************
    //Hardware definitions
    DDRA = 0xff;       //make port all outputs
    DDRB = 0xff;       //make port all outputs
    DDRC = 0xff;       //make port all outputs
    DDRD = 0xff;       //make port all outputs
    DDRB &= ~_BV(0);     //set pin B.0 to input  - Red_sw
    DDRB &= ~_BV(1);     //set pin B.1 to input  - Yel_sw
    DDRB &= ~_BV(2);     //set pin B.2 to input  - Grn_sw
    DDRB &= ~_BV(3);     //set pin B.3 to input  - Blu_sw
    DDRB &= ~_BV(4);     //set pin B.4 to input  - Wht_sw
    DDRA &= ~_BV(0);     //set pin A.0 to input  - POT
    DDRA &= ~_BV(1);     //set pin A.1 to input  - LM35
    DDRA &= ~_BV(2);     //set pin A.2 to input  - LDR
    DDRA &= ~_BV(4);     //set pin A.4 to input  - Ser_Rx
```

```
    //************************************************
    //Hardware macros for output ports
    #define set_Grn_Led PORTB |= (1<<5)        //force portB.5 output high
    #define clr_Grn_Led PORTB &= ~(1<<5)        //force portB.5 output low
    #define set_Yel_Led PORTB |= (1<<6)        //force portB.6 output high
    #define clr_Yel_Led PORTB &= ~(1<<6)        //force portB.6 output low
    #define set_Red_Led PORTB |= (1<<7)        //force portB.7 output high
    #define clr_Red_Led PORTB &= ~(1<<7)        //force portB.7 output low
    #define set_Ser_Tx PORTA |= (1<<5)        //force portA.5 output high
    #define clr_Ser_Tx PORTA &= ~(1<<5)        //force portA.5 output low
    #define set_Backlight PORTA |= (1<<3)        //force portA.3 output high
    #define clr_Backlight PORTA &= ~(1<<3)        //force portA.3 output low

    //************************************************
    //Hardware macros for input pins
    #define Red_sw_is_clr ~PINB & (1<<0)      //pinB.0 input==low
    #define Red_sw_is_set PINB & (1<<0)      //pinB.0 input==high
    //PORTB |= (1<<0);        //activate internal pull-up resistor for pinB.0
    #define Yel_sw_is_clr ~PINB & (1<<1)      //pinB.1 input==low
    #define Yel_sw_is_set PINB & (1<<1)      //pinB.1 input==high
    //PORTB |= (1<<1);        //activate internal pull-up resistor for pinB.1
    #define Grn_sw_is_clr ~PINB & (1<<2)      //pinB.2 input==low
    #define Grn_sw_is_set PINB & (1<<2)      //pinB.2 input==high
    //PORTB |= (1<<2);        //activate internal pull-up resistor for pinB.2
    #define Blu_sw_is_clr ~PINB & (1<<3)      //pinB.3 input==low
    #define Blu_sw_is_set PINB & (1<<3)      //pinB.3 input==high
    //PORTB |= (1<<3);        //activate internal pull-up resistor for pinB.3
    #define Wht_sw_is_clr ~PINB & (1<<4)      //pinB.4 input==low
    #define Wht_sw_is_set PINB & (1<<4)      //pinB.4 input==high
    //PORTB |= (1<<4);        //activate internal pull-up resistor for pinB.4
    #define Ser_Rx_is_clr ~PINA & (1<<4)      //pinA.4 input==low
    #define Ser_Rx_is_set PINA & (1<<4)      //pinA.4 input==high
    //PORTA |= (1<<4);        //activate internal pull-up resistor for pinA.4

    while(1)
    {
    }
}
```

61.16 Writing your own functions

Currently we have the following program, we have tidied up the code a great deal with the use of macros but the two ‘for’ loops are very messy and clutter the structure of the code, these will be replaced with a function

```
    while(1)
    {
        if(Yel_sw_is_set)
         {
              clr_Yel_Led;            // drive PB6 low
              for (count=0; count <900; count ++)
              {
                   _delay_loop_2(1000);
              }
              set_Yel_Led;
              for (count=100; count >0; count --)
              {
                   _delay_loop_2(1000);
              }
         }
    }
```

The function takes a uint16_t (2 byte) number and returns void (no value)

```
void my_inaccurate_ms_delay(uint16_t count)
{
     uint16_t i;
     for (i=count; i>0; i--)
          {
               _delay_loop_2(1000);
          }
}
```

For readability the function is placed at the end of our program, however when the compiler tries to compile the code <u>if it comes across a call to the function before it knows what it is then it gives an error</u>.

So in C a copy of the function definition is placed before the main function as in the full listing here.This is called a function prototype in C.

Replacing the for loop with our own function will make it

```
//**********************************************
// Project Name: 12TCEDemoBoard
// created by:
// using block diagram: BD_1
// Date:17/05/2012 6:10:44 a.m.
// Code autogenerated by System Designer from www.techideas.co.nz
//**********************************************
// Comment next line if using WINAVR and CPU Speed is in the Makefile
#define F_CPU 8000000UL
//**********************************************
#include <avr/io.h.>
#include <inttypes.h>
#include <util/delay.h>

int main (void)
{
    //**********************************************
    //Hardware definitions
    DDRA = 0xff;        //make port all outputs
    DDRB = 0xff;        //make port all outputs
    DDRC = 0xff;        //make port all outputs
    DDRD = 0xff;        //make port all outputs
    DDRB &= ~_BV(1);      //set pin B.1 to input  - Yel_sw

    //**********************************************
    //Hardware macros for output ports
    #define set_Yel_Led PORTB |= (1<<6)        //force portB.6 output high
    #define clr_Yel_Led PORTB &= ~(1<<6)        //force portB.6 output low

    //**********************************************
    //Hardware macros for input pins
    #define Yel_sw_is_clr ~PINB & (1<<1)      //pinB.1 input==low
    #define Yel_sw_is_set PINB & (1<<1)      //pinB.1 input==high
    PORTB |= (1<<1);        //activate internal pull-up resistor for pinB.1

     //**********************************************
      //function prototypes
      void my_ms_delay(uint16_t count);

    //**********************************************
      while(1)
    {
     if(Yel_sw_is_clr)
            {
                   set_Yel_Led;
                   my_inaccurate_ms_delay(100);
                   clr_Yel_Led;
                   my_inaccurate_ms_delay(900);
            }
    }
}
//**********************************************
// functions
//**********************************************
void my_inaccurate_ms_delay(uint16_t count)
{
       uint16_t i;
       for (i=count; i>0; i--)
            {
                   _delay_loop_2(1000);
            }
}
```

61.17 AVR Studio editor features

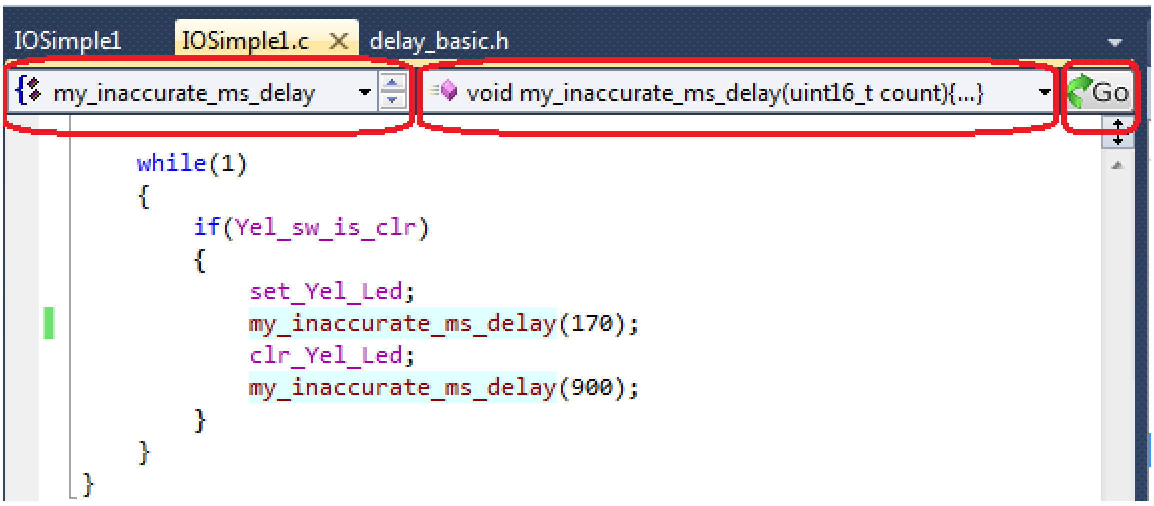
```
//************************************

while(1)
{
    if(Yel_sw_is_clr)
    {
        set_Yel_Led;
        my_inaccurate_ms_delay(170);
        clr_Yel_Led;
        my_inaccurate_ms_delay(900);
    }
}
}
```

Notice the faint yellow line on the left hand of the editor window; this tells you that the last change you made was to this line and you have not saved it yet.

```
//************************************

while(1)
{
    if(Yel_sw_is_clr)
    {
        set_Yel_Led;
        my_inaccurate_ms_delay(170);
        clr_Yel_Led;
        my_inaccurate_ms_delay(900);
    }
}
}
```

Once saved it goes green.

Click on a function or define and take note of the top of the editor window as circled in red here



When my_inaccurate_ms_day is clicked these show usthe context and definition of what we click on And allow us to jump to that definition by clicking on the GO button. Clickin on this and the editor jumps to the function. Go into the function and click on _delay_loop_2 and the editor opens the delay_basic.h file.

61.18 AVR hardware registers

We will use the understanding of this in the nextsection about writing to an LCD.
In this program we have used these macros

```
#define set_grn_led PORTA |= (1<<7)        //force portA.7 output high
#define clr_grn_led PORTA &= ~(1<<7)        //force portA.7 output low
```

Now the word PORTA is a macro itself and if you select it and push the GO button it takes you to a file that declares exactly what PORTA means here it is for the Mega644

```
#define PINA   _SFR_IO8(0X00)
#define PINA7  7
#define PINA6  6
#define PINA5  5
#define PINA4  4
#define PINA3  3
#define PINA2  2
#define PINA1  1
#define PINA0  0

#define DDRA   _SFR_IO8(0X01)
#define DDA7   7
#define DDA6   6
#define DDA5   5
#define DDA4   4
#define DDA3   3
#define DDA2   2
#define DDA1   1
#define DDA0   0

#define PORTA  _SFR_IO8(0X02)
#define PA7 7
#define PA6 6
#define PA5 5
#define PA4 4
#define PA3 3
#define PA2 2
#define PA1 1
#define PA0 0
```

Note thate there are three registers for this port, PORTA, DDRA and PINA

In a different micro this may be a different address
e.g in the ATMega16 and ATMega32 PORT A is in a different location to the ATMega644
here from the datasheet is a section of some of the registers

0x0E (0x2E)	Reserved	-	-	-	-	-	-	-	-
0x0D (0x2D)	Reserved	-	-	-	-	-	-	-	-
0x0C (0x2C)	Reserved	-	-	-	-	-	-	-	-
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0

61.19 Character LCD programming in C

C programmingis very different to Bascom as Bascom contains a great many library of functions that do not exist in C as it comes delivered to you in Atmel Studio.

On the internet you will find a great many functions to drive LCDs but here we will develop one ourselves. There are a number of reasons why you should learn this.

1. You will learn about making your own header files and including them
2. You will learn about writing functions and returning values
3. The LCD libraries on the internet often use the R/W line of the LCD to read data from the LCD, if you disconnect the LCD hardware they your prorgam can hang in a loop trying to read the LCD.
4. You will learn how to write software to control devices yourself
5. This software will enable you to put the lcd on any pins of the microcontroller, across different ports if you want to.

61.20 CharLCD.h Header file

This is the file that declares the information and functions for our LCD
All our functions will make use of actual hardware pins of the micro, speed of the microcontroller,and the number of lines and the length of the line.
This part of the file would be modified by users when theyhabe different displays or connections.

```
/*  CharLCD.h
 *  Declarations for char LCD functions
 */

#ifndef CharLCD_H
#define CharLCD_H

//make sure this is the same as the crystal/R-C frequency
#define XTAL 8000000

//change these to reflect the display
#define LCD_DISP_LENGTH   20
#define LCD_DISP_LINES     4

//change these to reflect the display connections, any pin on any port
#define LCD_PORT_DAT4       PORTC
#define LCD_PIN_DAT4        2
#define LCD_PORT_DAT5       PORTC
#define LCD_PIN_DAT5        3
#define LCD_PORT_DAT6       PORTC
#define LCD_PIN_DAT6        4
#define LCD_PORT_DAT7       PORTC
#define LCD_PIN_DAT7        5
#define LCD_PORT_EN         PORTC
#define LCD_PIN_EN          1
#define LCD_PORT_RS         PORTC
#define LCD_PIN_RS          0
```

Next we need to know about the commands that control the LCD these are declared

```
//LCD commands
#define LCD_CLR         0x01        // clear LCD
#define LCD_HOME        0x02        // clear LCD
// see animations http://www.geocities.com/dinceraydin/lcd/commands.htm for the following commands
#define LCD_INC         0x04        // decrement address counter, display shift off
#define LCD_INC         0x05        // decrement address counter, display shift on
#define LCD_INC         0x06        // Increment address counter, display shift off - default
#define LCD_INC         0x07        // Increment address counter, display shift on

#define LCD_ALL         0x0F        // LCD On, LCD display on, cursor on and blink on
#define LCD_ON          0x0C        // turn lcd on/no cursor
#define LCD_OFF         0x08        // turn lcd off
#define LCD_ON_DISPLAY  0x04        // turn display on
#define LCD_ON_CURSOR   0x0E        // turn cursor on
#define LCD_ON_BLINK    0x0F        // cursor blink
#define LCD_X0Y0        0x80        // cursor Pos on line 1 (or with column)
#define LCD_X0Y1        0xC0        // cursor Pos on line 2 (or with column)
#define LCD_X0Y2        0x94        // cursor Pos on line 3 (or with column)
#define LCD_X0Y3        0xD4        // cursor Pos on line 4 (or with column)
#define LCD_CURSOR_LEFT 0x10        //move cursor one place to left
#define LCD_CURSOR_RIGHT 0x14       //move cursor one place to right

#define LCD_DELAY       100
```

And lastly out CharLCD.h contains prototypes of all the functions we will write

```
void lcd_write(unsigned char dat, char rs);
void lcd_pulse_en();
void lcd_init();
void lcd_off();
void lcd_on();
void lcd_cursorOn();
void lcd_cursorOff();
void lcd_cursorBlink();
void lcd_cls();
void lcd_home();
void lcd_cursorXY(char x, char y);
void lcd_line0();
void lcd_line1();
void lcd_line2();
void lcd_line3();
void lcd_disp_str(const char *str);
void lcd_disp_dec_uchar(unsigned char n);
void lcd_disp_bin_uchar(unsigned char n);
void lcd_disp_bin_schar(signed char n);
void lcd_disp_dec_uint(unsigned int n);
void lcd_dispdec_sint(signed int n);
void lcd_disp_bin_unit(unsigned int n);
void lcd_disp_bin_sint(signed int n);
```

CharLCD.c – the macros

These macros are shortcuts that will make our function writing a little easier, they simply set and reset the various output pins of the micro.

```
/*  CharLCD.c
 *  Implementation of functions that handle output to char lcd.
 */

#include <avr/io.h>
#include "CharLCD.h"

#define lcd_en_high()    LCD_PORT_EN  |=  _BV(LCD_PIN_EN);
#define lcd_en_low()     LCD_PORT_EN  &= ~_BV(LCD_PIN_EN);

#define lcd_rs_high()   LCD_PORT_RS |= _BV(LCD_PIN_RS)
#define lcd_rs_low()    LCD_PORT_RS &= ~_BV(LCD_PIN_RS)


#define lcd_dat4_high()   LCD_PORT_DAT4 |=  _BV(LCD_PIN_DAT4)
#define lcd_dat4_low()    LCD_PORT_DAT4 &= ~_BV(LCD_PIN_DAT4)

#define lcd_dat5_high()   LCD_PORT_DAT5 |=  _BV(LCD_PIN_DAT5)
#define lcd_dat5_low()    LCD_PORT_DAT5 &= ~_BV(LCD_PIN_DAT5)

#define lcd_dat6_high()   LCD_PORT_DAT6 |=  _BV(LCD_PIN_DAT6)
#define lcd_dat6_low()    LCD_PORT_DAT6 &= ~_BV(LCD_PIN_DAT6)

#define lcd_dat7_high()   LCD_PORT_DAT7 |=  _BV(LCD_PIN_DAT7)
#define lcd_dat7_low()    LCD_PORT_DAT7 &= ~_BV(LCD_PIN_DAT7)
```

61.21 Manipulating AVR register addresses

You will need to understand the previous short section on AVR hardware registers before understanding this one!

There is another very important define, we have the port and pin of each connection from the microcontroller to each LCD pin but we also need to know the data direction register of each of these pins as well. i.e. if we have RS on PORTC.0 we need to make DDRC.0 an output.

So to figure this out we find out what the address of the PORT is (e.g. 0x08 for PORTC in our M644) and subtract one from it to get 0x07 which is the DDR for portC.
We need to be able to do this for any port, so DDRA is 1 less than PORTA and so on.

We need to be able to find the address of a port (or any variable in RAM as well) then we do this with the ''& in c. So if I use &PORTC I will get 0x08, the address of PORTC register. If I go &DDRC I will get 0x07.

So to get DDR of any port I subtract 1 from the address of a port
e.g. &port-1

now I have the address of DDRC, next I want to be able to change its contents, so I de-reference it with the * so when I want to change the contents of the DDR register and I only know the PORT register
I go (*(&port-1)), this finds the address of the PORT, subtracts one to get the DDR and then I can change the contents of the DDR

```
// address of data direction register of port, this is 1 less than the PORT address
#define DDR(port) (*(&port - 1))
```

This sort of thing is used in lots of ways within C we could store 20 numbers in RAM and then get to the numbers by knowing their addresses. We will do more of it later.

61.22 Writing to the LCD

We will next get an understanding of how to write to an alphanumeric LCD in 4 bit mode (our LCD routines will not be complicated with 8 bit mode) – start by reviewing what has been written about 4 bit versus 8 bit mode elsewhere in the book under Alphanumeric LCDs.
There are many useful website that explain the use of Alphanumeric or Character LCDs.

We will need some different functions that control the LCD (we are going to totally ignore reading from the LCD)

First there are two different different types of information we need to send to the display, the first is data for displaying and the second is information to go into the displays control register to tell the hardware of the LCD what to do.
The pin RS stands for register select and we tell the display whether we are sending a command to the register by making this high or data to the memory by making this low.
Then because we are in 4 bit mode we need to send the uppr 4 bits of our 8 bit byte first the the lower 4 bits.

Command	0	0	0	0	1	0	0	0	**Command 0x08 = turn display on with the cursor hidden**
Upper 4	0	0	0	0					**Send these 4 bits first**
Lower 4					1	0	0	0	**Send these 4 bits next**

We could write two different functions like some people do the first to write commands the second to write data, I chose to write 3 functions:
The first sends either data or commands to the display
The second sends data only using the above function
The third sends commands only using the above function

Here are the two for sending only one type, commnd or data

```
void lcd_command(unsigned char dat) {
    delay_us(LCD_DELAY);
    lcd_write(dat,0);
}
void lcd_data(unsigned char dat)    {
    delay_us(LCD_DELAY);
    lcd_write(dat,1);
}
```

Here is the function to write either data or commands to the LCD

```
void lcd_write(unsigned char dat, char rs)
{
    if (rs ) {lcd_rs_high();}else{lcd_rs_low();};   //command=1 or data=0
    //get upper4 bits of dat and put onto the 4 pins
    if (dat & 0x80) {lcd_dat7_high();}else{lcd_dat7_low();}
    if (dat & 0x40) {lcd_dat6_high();}else{lcd_dat6_low();}
    if (dat & 0x20) {lcd_dat5_high();}else{lcd_dat5_low();}
    if (dat & 0x10) {lcd_dat4_high();}else{lcd_dat4_low();}
    lcd_pulse_en();
    //get lower4 bits of dat and put onto the 4 pins
    if (dat & 0x08) {lcd_dat7_high();}else{lcd_dat7_low();}
    if (dat & 0x04) {lcd_dat6_high();}else{lcd_dat6_low();}
    if (dat & 0x02) {lcd_dat5_high();}else{lcd_dat5_low();}
    if (dat & 0x01) {lcd_dat4_high();}else{lcd_dat4_low();}
    lcd_pulse_en();
}
```

`void lcd_write(unsigned char dat, char rs)`

this line is the function name and tells us that it expects two parameters an 8 bit (unsigned char) variable that will be called dat in this function and a char RS which will either be a 1 or a 0. It also returns nothing after it has completed.

Next set the RS bit of the display

```
	if (rs) {lcd_rs_high();}else{lcd_rs_low();}	//command=1 or data=0
```

to make my overall code clearer to read I have compressed all this onto one line ususaly we would write it

```
	if (rs)
	{
		lcd_rs_high();
	}
	else
	{
		lcd_rs_low();
	}	//command=1 or data=0
```

Note that `if(rs)` is a c programming convention that is an abbreviation of `if (rs==1)`

Now send the upper 4 bits, here is the first line

```
	//get upper4 bits of dat and put onto the 4 pins
	if (dat & 0x80) {lcd_dat7_high();}else{lcd_dat7_low();}
```

`if (dat & 0x80)` means if f bit7 is 1 then make the pin high else make it low.

`lcd_pulse_en();`

is a call to another function that makes our enable line high then waits a little bit then makes it low.This must happen after the 4 bits have been setup on the 4 pins to the LCD.

```
void lcd_pulse_en()
{
    lcd_en_high();
    delay_us(100);
    lcd_en_low();
}
```

Delay_us is an important delay to the LCD, the LCD requires a bit of time to process wat we are telling it to do so every time we write to it we wait 100uS. Delay_us makes use of some more complex assembly language code we will not go into.

61.23 Initialise the LCD

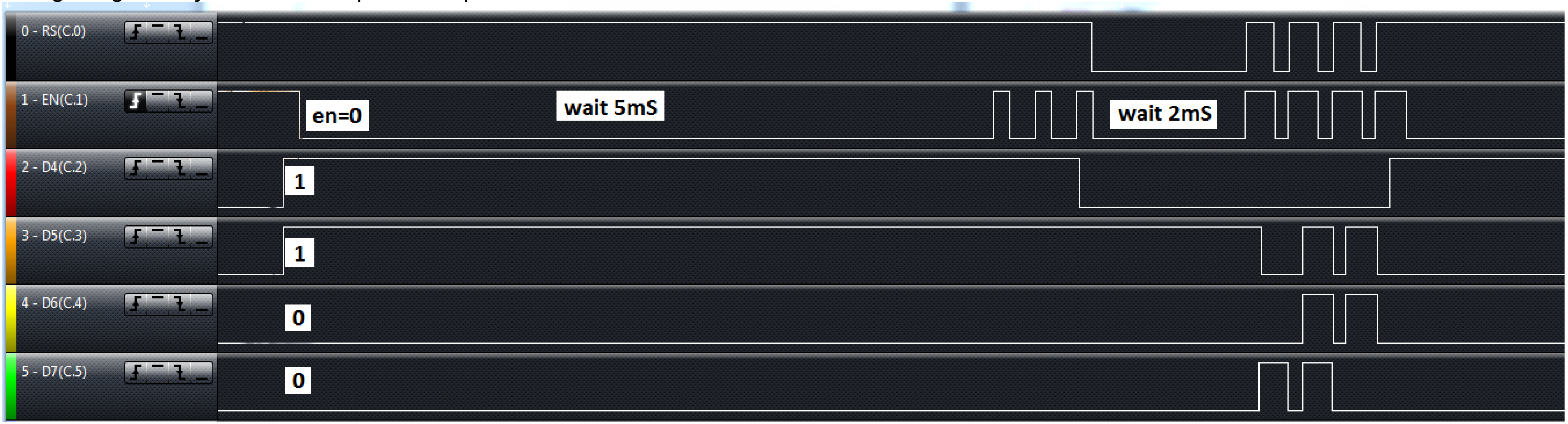
When power is applied t the LCD there is a very specific process to go through before it can be used.

```
void lcd_init()
{
    //setup 6 pins as outputs
    DDR(LCD_PORT_RS) |= _BV(LCD_PIN_RS);
    DDR(LCD_PORT_EN) |= _BV(LCD_PIN_EN);
    DDR(LCD_PORT_DAT7) |= _BV(LCD_PIN_DAT7);
    DDR(LCD_PORT_DAT6) |= _BV(LCD_PIN_DAT6);
    DDR(LCD_PORT_DAT5) |= _BV(LCD_PIN_DAT5);
    DDR(LCD_PORT_DAT4) |= _BV(LCD_PIN_DAT4);

    lcd_en_high();
    delay_us(20000);            //power up delay
    lcd_en_low();
    lcd_rs_low();
    //Write D7-4 = 0011
    lcd_dat7_low();
    lcd_dat6_low();
    lcd_dat5_high();
    lcd_dat4_high();
    lcd_pulse_en();
    delay_us(5000);
    //repeat again
    lcd_pulse_en();
    delay_us(200);
    //repeat again
    lcd_pulse_en();
    delay_us(200);
    //Write D7-4 = 0010
    lcd_dat7_low();
    lcd_dat6_low();
    lcd_dat5_high();
    lcd_dat4_low();
    lcd_pulse_en();
    delay_us(1000);

    lcd_command(0x28);  //0b0010100 is interface=4bits, 2 lines, 5*7 pixels
    lcd_command(0x06);  //move cursor right after each write to the display
    lcd_command(0x01);  //clear and home lcd
    delay_us(4000);     //give display a chance to do the above
    lcd_command(0x0C);  //display on ,cursor off, no blink
}
```

We configure the 6 interface pins as outputs.
Then we:

- Wait 20ms for LCD to power up
- Write D7-4 = 3 hex, and pulse high and low
- Wait 5ms
- Write D7-4 = 3 hex, and pulse high and low again
- Wait 200us
- Write D7-4 = 3 hex, and pulse high and low again
- Wait 200us
- Write D7-4 = 2 hex, to enable four-bit mode
- Wait 5ms
- Write Commands as required to set up the display how we want it

Using a logic analyser we can capture this process



Here you can see the sequence of signals after the first 20mS delay
On the left we put 0011 onto the LCD then take EN low
wait for 5mS
then en is pulsed 3 times
there is a 2mS wait
and then we send the commands to set up the display

61.24 lcd commands

we need a bunch of functions that send commands to the LCD

```
void lcd_off()          {lcd_command(LCD_OFF);}
void lcd_on()           {lcd_command(LCD_ON);}
void lcd_cursorOn()     {lcd_command(LCD_ON_CURSOR);}//no blink
void lcd_cursorOff()    {lcd_command(LCD_ON);}
void lcd_cursorBlink()  {lcd_command(LCD_ON_BLINK);}
void lcd_cls()          {lcd_command(LCD_CLR);}
void lcd_home()         {lcd_command(LCD_HOME);}
void lcd_line0()        {lcd_command(LCD_X0Y0);}
void lcd_line1()        {lcd_command(LCD_X0Y1);}
void lcd_line2()        {lcd_command(LCD_X0Y2);}
void lcd_line3()        {lcd_command(LCD_X0Y3);}
```

```
void lcd_cursorXY(char x, char y){              //0,0 is top left
    if (y>=LCD_DISP_LINES || x>=LCD_DISP_LENGTH ) {return;}     //ignore nonsense values
    if (y==0){lcd_command(LCD_X0Y0+x);}          //0x80+ x value
    if (y==1){lcd_command(LCD_X0Y1+x);}
    if (y==2){lcd_command(LCD_X0Y2+x);}
    if (y==3){lcd_command(LCD_X0Y3+x);}
}
```

61.25 Writing text to the LCD

These displays require ascii characters to be sent to them such as "cat". In C these strings are really just arrays of chars with 0 on the end.
So cat is 4 bytes of RAM with the numbers 0x63, 0x61, 0x74, 0x00 in it.
In the AVR RAM starts at address 0x60 so our ram might look like this

Address (hexadecimal)	Contents (hexadecimal)	Contents (ascii)
63	00	null
62	74	t
61	61	a
60	63	c

In our program we declare our variable animal2 by

```
char animal2[12]="cat";
```

this means allocate 12 bytes of RAM and put cat into the the first three bytes.

In our program we write code to locate the cursor and then put the string onto the LCD

```
lcd_line2();
lcd_disp_str(animal2);
```

and cat will appear on the display

our function to write the string to the lcd is

```
void lcd_disp_str(const char *str)  {        //text string
    register unsigned char i;
    for (i=0; str[i];i++) //loop till null termination
    {
        delay_us(LCD_DELAY);
        lcd_data(str[i]);
    }
}
```

First when passing strings around (or any array) we don't pass the data we pass its address.
So `lcd_disp_str(animal2);` actually passes the address of the first character in memory to the function and not the characters c,a,t (the address will be a number such as 0x60).

The function must be written so that it knows it is getting an address of a variable not the contents of the variable.The de-referencing * operator is used to get the variable from an address.

passed an address,
we actually want the variable not its address
so we use the * (de-reference it)

```
void lcd_disp_str(const char *str)  {        //text string
    register unsigned char i;
    for (i=0; str[i];i++) //loop till null termination
    {
        delay_us(LCD_DELAY);
        lcd_data(str[i]);
    }
}
```

all strings end in 0, so loop till we get 0
the for loop sees a 0 as false and exits

send each character one at a time to the display

There are other ways of doing this, here is another

```
void lcd_disp_str2(const char *str) {
    register unsigned char c;
    while ((c=*str++)){
        delay_us(LCD_DELAY);
        lcd_data(c);
    }
}
```

passed an address,
we actually want the variable not its address
so we use the * (de-reference it)

c=*str get the contents of the address
then using ++ increment the address (not the contents)
do this while the contents are true (not 0)

send each character one at a time to the display

Now these functions are very useful if we are manipulating strings in our programs however often we dotn want to manipulate the string we just want something displayed on the LCD and that will never change.
We could do either of these

`char* animal3="dolphin";`
and
`lcd_line3();`
`lcd_disp_str(animal3);`
	OR	`lcd_disp_str("dolphin");`
		(called a string literal or constant)

Here we declare a variable animal3 - actually we declare a pointer to a variable (a pointer stores an address of a variable)		Here just write the characters straight into the program
The effect of these is actually both excatly the same The word dolphin is stored in our program and copied into RAM when the program starts. AND THIS IS A HUGE WASTE OF OUR PRECIOUS RAM		

61.26 Program Flash and Strings

So we use another function that forces our string to be put into our program flash and used from the program flash.
First we must include the functions that allow us to read and write into the flash

```
#include <avr/pgmspace.h>
```

Then we declare our variable of special type

```
const char animal1[] PROGMEM= "Giraffe"; //forces storage in flash
```

Then use the new display function

```
lcd_line1();
lcd_disp_str_P(animal1);
```

And here is our function for writing from program flash to the lcd.

```
void lcd_disp_str_P(const char *str)    {       //text string
    register unsigned char i;
    for (i=0; (char)pgm_read_byte(&str[i]);i++) //loop till null termination
    {
        delay_us(LCD_DELAY);
        lcd_data((char)pgm_read_byte(&str[i]));
    }
}
```

61.27 LCD test program1

```c
/*
 * LCD.c
 *
 * Created: 6/10/2012 8:54:41 PM
 *  Author: B.Collis
 */

#define F_CPU 8000000

#include <avr/io.h>
#include <avr/pgmspace.h>
#include "CharLCD.h"
#include "util/delay.h"

const char animal1[] PROGMEM= "Giraffe"; //forces storage in flash
char animal2[12]="cat"; //stored in RAM - takes 12 bytes
char* animal3="dolphin"; //stored in RAM allocates 7 bytes
char *animal4="kangaroo";
char *animal5="monkey";
//see this website for a great tutorial on using progmem
//http://www.fourwalledcubicle.com/AVRArticles.php


int main(void)
{
    lcd_init();
       lcd_cursorOn();

       lcd_disp_str("caterpillar");//string literal or constant - copied to ram at startup

       lcd_line1();
       lcd_disp_str_P(animal1);    //string in ram

       lcd_line2();
       lcd_disp_str(animal2);      // string in flash

       animal2[3]='y';
       lcd_cursorXY(10,2);
       lcd_disp_str2(animal2);     //string in ram

       lcd_line3();
       lcd_disp_str(animal3);      // string in flash

       animal3[0]='D';
       lcd_cursorXY(10,3);
       lcd_disp_str(animal3);      //string in ram

       while(1)
    {
              _delay_ms(800);
              lcd_command(LCD_CURSOR_RIGHT);
    }
}
```

61.28 CharLCD.h

Note that the first part of CharLCD.h can be automatically generated for you from System Designer

```
/*      CharLCD.h
*       Declarations for char LCD functions
*/

#ifndef CharLCD_H
#define CharLCD_H

//make sure this is the same as the crystal/R-C frequency
#define XTAL 8000000

//change these to reflect the display
#define LCD_DISP_LENGTH   20
#define LCD_DISP_LINES     4

//change these to reflect the display connections, any pin on any port
#define LCD_PORT_DAT4       PORTC
#define LCD_PIN_DAT4        2
#define LCD_PORT_DAT5       PORTC
#define LCD_PIN_DAT5        3
#define LCD_PORT_DAT6       PORTC
#define LCD_PIN_DAT6        4
#define LCD_PORT_DAT7       PORTC
#define LCD_PIN_DAT7        5
#define LCD_PORT_EN         PORTC
#define LCD_PIN_EN          1
#define LCD_PORT_RS         PORTC
#define LCD_PIN_RS          0


//LCD commands
#define LCD_CLR             0x01          // clear LCD
#define LCD_HOME            0x02          // clear LCD
// see animations http://www.geocities.com/dinceraydin/lcd/commands.htm for the following commands
#define LCD_INC             0x04          // decrement address counter, display shift off
#define LCD_INC             0x05          // decrement address counter, display shift on
#define LCD_INC             0x06          // Increment address counter, display shift off - default
#define LCD_INC             0x07          // Increment address counter, display shift on

#define LCD_ALL             0x0F          // LCD On, LCD display on, cursor on and blink on
#define LCD_ON              0x0C          // turn lcd on/no cursor
#define LCD_OFF             0x08          // turn lcd off
#define LCD_ON_DISPLAY      0x04          // turn display on
#define LCD_ON_CURSOR       0x0E          // turn cursor on
#define LCD_ON_BLINK        0x0F          // cursor blink
#define LCD_X0Y0            0x80         // cursor Pos on line 1 (or with column)
#define LCD_X0Y1            0xC0         // cursor Pos on line 2 (or with column)
#define LCD_X0Y2            0x94         // cursor Pos on line 3 (or with column)
#define LCD_X0Y3            0xD4         // cursor Pos on line 4 (or with column)
#define LCD_CURSOR_LEFT     0x10          //move cursor one place to left
#define LCD_CURSOR_RIGHT    0x14          //move cursor one place to right

#define LCD_DELAY           100
```

```c
void lcd_write(unsigned char dat, char rs);
void lcd_pulse_en();
void lcd_init();
void lcd_off();
void lcd_on();
void lcd_cursorOn();
void lcd_cursorOff();
void lcd_cursorBlink();
void lcd_cls();
void lcd_home();
void lcd_cursorXY(char x, char y);
void lcd_line0();
void lcd_line1();
void lcd_line2();
void lcd_line3();
void lcd_disp_str(const char *str);
void lcd_disp_str2(const char *str); //temp
void lcd_disp_str_P(const char *str);
void lcd_disp_dec_uchar(unsigned char n);
void lcd_disp_bin_uchar(unsigned char n);
void lcd_disp_bin_schar(signed char n);
void lcd_disp_dec_uint(unsigned int n);
void lcd_dispdec_sint(signed int n);
void lcd_disp_bin_unit(unsigned int n);
void lcd_disp_bin_sint(signed int n);


#endif //CharLCD_h
```

61.29 CharLCD.c

```c
/*     CharLCD.c
*      Implementation of functions that handle output to char lcd.
*/

#include <avr/io.h>
#include "CharLCD.h"
#include <avr/pgmspace.h>

#define lcd_en_high()    LCD_PORT_EN  |=  _BV(LCD_PIN_EN);
#define lcd_en_low()     LCD_PORT_EN  &= ~_BV(LCD_PIN_EN);

#define lcd_rs_high()   LCD_PORT_RS |= _BV(LCD_PIN_RS)
#define lcd_rs_low()    LCD_PORT_RS &= ~_BV(LCD_PIN_RS)


#define lcd_dat4_high()   LCD_PORT_DAT4 |=  _BV(LCD_PIN_DAT4)
#define lcd_dat4_low()    LCD_PORT_DAT4 &= ~_BV(LCD_PIN_DAT4)

#define lcd_dat5_high()   LCD_PORT_DAT5 |=  _BV(LCD_PIN_DAT5)
#define lcd_dat5_low()    LCD_PORT_DAT5 &= ~_BV(LCD_PIN_DAT5)

#define lcd_dat6_high()   LCD_PORT_DAT6 |=  _BV(LCD_PIN_DAT6)
#define lcd_dat6_low()    LCD_PORT_DAT6 &= ~_BV(LCD_PIN_DAT6)

#define lcd_dat7_high()   LCD_PORT_DAT7 |=  _BV(LCD_PIN_DAT7)
#define lcd_dat7_low()    LCD_PORT_DAT7 &= ~_BV(LCD_PIN_DAT7)



static inline void _delayFourCycles(unsigned int __count)
{
       if ( __count == 0 )
       __asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
       else
       __asm__ __volatile__ (
       "1: sbiw %0,1" "\n\t"
       "brne 1b"                              // 4 cycles/loop
       : "=w" (__count)
       : "0" (__count)
       );
}

#define delay_us(us)  _delayFourCycles( ( ( 1*(XTAL/4000) )*us)/1000 )

// address of data direction register of port, this is 1 less than the PORT address
#define DDR(port) (*(&port - 1))
```

```c
void lcd_init()
{
	//setup 6 pins as outputs
	DDR(LCD_PORT_RS) |= _BV(LCD_PIN_RS);
	DDR(LCD_PORT_EN) |= _BV(LCD_PIN_EN);
	DDR(LCD_PORT_DAT7) |= _BV(LCD_PIN_DAT7);
	DDR(LCD_PORT_DAT6) |= _BV(LCD_PIN_DAT6);
	DDR(LCD_PORT_DAT5) |= _BV(LCD_PIN_DAT5);
	DDR(LCD_PORT_DAT4) |= _BV(LCD_PIN_DAT4);

	lcd_en_high();
	delay_us(20000);			//power up delay
	lcd_en_low();
	lcd_rs_low();
	//Write D7-4 = 0011
	lcd_dat7_low();
	lcd_dat6_low();
	lcd_dat5_high();
	lcd_dat4_high();
	lcd_pulse_en();
	delay_us(5000);
	//repeat again
	lcd_pulse_en();
	delay_us(200);
	//repeat again
	lcd_pulse_en();
	delay_us(200);
	//Write D7-4 = 0010
	lcd_dat7_low();
	lcd_dat6_low();
	lcd_dat5_high();
	lcd_dat4_low();
	lcd_pulse_en();
	delay_us(1000);

	lcd_command(0x28);	//0b0010100 is interface=4bits, 2 lines, 5*7 pixels
	lcd_command(0x06);	//move cursor right after each write to the display
	lcd_command(0x01);	//clear and home lcd
	delay_us(4000);		//give display a chance to do the above
	lcd_command(0x0C);	//display on ,cursor off, no blink
}

void lcd_write(unsigned char dat, char rs)
{
	if (rs ) {lcd_rs_high();} else {lcd_rs_low();}	//command=1 or data=0
	//get upper4 bits of dat and put onto the 4 pins
	if (dat & 0x80) {lcd_dat7_high();}else{lcd_dat7_low();}
	if (dat & 0x40) {lcd_dat6_high();}else{lcd_dat6_low();}
	if (dat & 0x20) {lcd_dat5_high();}else{lcd_dat5_low();}
	if (dat & 0x10) {lcd_dat4_high();}else{lcd_dat4_low();}
	lcd_pulse_en();
	//get lower4 bits of dat and put onto the 4 pins
	if (dat & 0x08) {lcd_dat7_high();}else{lcd_dat7_low();}
	if (dat & 0x04) {lcd_dat6_high();}else{lcd_dat6_low();}
	if (dat & 0x02) {lcd_dat5_high();}else{lcd_dat5_low();}
	if (dat & 0x01) {lcd_dat4_high();}else{lcd_dat4_low();}
	lcd_pulse_en();
}

void lcd_pulse_en()
{
	lcd_en_high();
	delay_us(100);
	lcd_en_low();
}
```

```
void lcd_off()                      {lcd_command(LCD_OFF);}
void lcd_on()               {lcd_command(LCD_ON);}
void lcd_cursorOn()         {lcd_command(LCD_ON_CURSOR);}//no blink
void lcd_cursorOff() {lcd_command(LCD_ON);}
void lcd_cursorBlink()      {lcd_command(LCD_ON_BLINK);}
void lcd_cls()                      {lcd_command(LCD_CLR);}
void lcd_home()                     {lcd_command(LCD_HOME);}
void lcd_line0()            {lcd_command(LCD_X0Y0);}
void lcd_line1()            {lcd_command(LCD_X0Y1);}
void lcd_line2()            {lcd_command(LCD_X0Y2);}
void lcd_line3()            {lcd_command(LCD_X0Y3);}

void lcd_cursorXY(char x, char y){                      //0,0 is top left
       if (y>=LCD_DISP_LINES || x>=LCD_DISP_LENGTH ) {return;}       //ignore nonsense values
       if (y==0){lcd_command(LCD_X0Y0+x);}                      //0x80+ x value
       if (y==1){lcd_command(LCD_X0Y1+x);}
       if (y==2){lcd_command(LCD_X0Y2+x);}
       if (y==3){lcd_command(LCD_X0Y3+x);}
}


void lcd_disp_str(const char *str) {             //text string
       register unsigned char i;
       for (i=0; str[i];i++) //loop till null termination
       {
              delay_us(LCD_DELAY);
              lcd_data(str[i]);
       }
}

void lcd_disp_str2(const char *str)       {             //text string
       register unsigned char c;
       while ((c=*str++)){                              //get contents of str then incr str
              delay_us(LCD_DELAY);
              lcd_data(c);
       }
}
void lcd_disp_str_P(const char *str)      {             //text string
       register unsigned char i;
       for (i=0; (char)pgm_read_byte(&str[i]);i++) //loop till null termination
       {
              delay_us(LCD_DELAY);
              lcd_data((char)pgm_read_byte(&str[i]));
       }
}
```

```
void lcd_disp_dec_uchar(unsigned char n) {                    //0 to 255
       char buffer[3];
       itoa (n, buffer,10);                            //decimal display of
       lcd_disp_str(buffer);
}
void lcd_disp_bin_uchar(unsigned char n) {             //0 to 255
       char buffer[8];
       itoa (n, buffer,2);                             //binary display of
       lcd_disp_str(buffer);
}
void lcd_disp_bin_schar(signed char n)   {             //0 to 255
       char buffer[8];
       itoa (n, buffer,2);                                    //binary display of
       lcd_disp_str(buffer);
}

void lcd_disp_dec_uint(unsigned int n)   {                    //- to
       char buffer[7];
       itoa (n, buffer,10);                     //decimal display
       lcd_disp_str(buffer);
}

void lcd_dispdec_sint(signed int n)      {                           //- to
       char buffer[7];
       itoa (n, buffer,10);                     //decimal display
       lcd_disp_str(buffer);
}
void lcd_disp_bin_unit(unsigned int n)   {             //- to
       char buffer[16];
       itoa (n, buffer,2);                             //binary display
       lcd_disp_str(buffer);
}
void lcd_disp_bin_sint(signed int n)     {                    //- to
       char buffer[16];
       itoa (n, buffer,2);                             //binary display
       lcd_disp_str(buffer);
}

void lcd_command(unsigned char dat)      {
       delay_us(LCD_DELAY);
       lcd_write(dat,0);
}
void lcd_data(unsigned char dat)   {
       delay_us(LCD_DELAY);
       lcd_write(dat,1);
}
```

62 Object Oriented Programming (OOP) in CPP and the AVR

The basis for OOP is that we need to effectively manage the development of large programs and those written by multiple programmers. If we don't then things can easily get out of control. Say two programmers are writing the same program and the first wanted a variable to control the area of the front patio of a house and another wanted a variable to control the size for the flat roof for the porch and both used 'f_size'! What a mess a large program could easily become. OOP allows programmers to manage names and structure for programs easily.

CPP (or C++ or C plus plus) is a version of the C programming language we can use to program the AVR and learn about OOP. There are many other OOP languages including common ones such as Java and C# (C sharp).

62.1 The black box concept

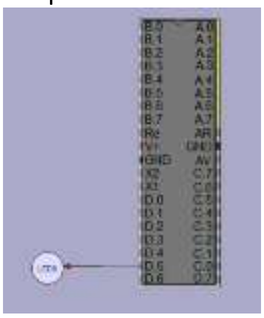
In technology we use the term 'black box' to represent the idea that we know about an objects inputs and outputs (structure) and its behaviour but not all the detail of how it does it. In OOP we use the same concept; we can have a routine, function or block of program code, we know its inputs and outputs and what it does; but we don't know anything about what is going on inside it. We only need to know how to interact with it, what its inputs and outputs are.
For example `int factorial(int n)` is a function that takes an integer, calculates the factorial for it and returns the answer; we don't know how it actually does the calculation just what it does it and its inputs and outputs.

The same situation exists with our AVR hardware; in an AVR we have I/O pins that we don't know anything about except what they do and how to use them; we are not interested in their internal workings just how to configure and how to use them. They can be thought of as black boxes to us in technology and in programming terms we can call these **objects**.

62.2 The concept of a class

In OOP we consider an I/O pin as a **class**, and one specific pin e.g. portD.5 is an **object** or an **instance of the class** of I/O pin. The word 'class' refers to a definition for an object, just like 'car' is a class that defines objects with 4 wheels and a motor and my green 1500cc manual 2009 Toyota Eco 5-door registration numberABC123 is an object or a specific instance of class car.

OutputPin class



Before we can use an object such as an output pin we have two important things to do:

1. Define the class by defining the **properties** (attributes and characteristics of the objects that the class represents) and the **methods** we use to interact or communicate with the object (these define operations, functions, abilities or behaviours); we do this in the definition of the class; for instance each output pin of a microcontroller should have **properties** of being high or low and a number of **methods** such as:
    a. set it high
    b. reset it low,
    c. toggle it,

    d. find out if it is high
    e. find out if it is low.
2. **Construct** (create, **instantiate**) one instance of the I/O pin. Think of the class as the design or template for an object (e.g. drawings for a stapler or pattern for a shirt) and constructing it is making one instance (making one stapler, making a shirt). We need to instantiate the output pin on portD.5 before we can use it.

62.3 First CPP program

In CPP we write this first program that creates an **instance** of an output port and then toggles it on and off to make an LED flash (this code can be autogenerated by System Designer using your block diagram)

```
#include "OutputPin.h"   //the definition of the class OutputPin is in this file
#include "Util.h"         //some useful functions we can use are found in this file

int main (void)
{
   //*********************************************
   //Hardware definitions
   //construct an instance of class output, on port D pin 5
    OutputPin led0('D',5);

   //Program starts here
   while(1)
   {
         led0.toggle();
         delay_ms(750);
   }
}
```

The various methods we can use to communicate with the object are found in the definition for class OuputPin in the header file “OutputPin.h”. (In C we have two files “OutputPin.h” and “OutputPin.c”; we will focus on the .h file now and the .c file a little later.) In “OutputPin.h” the methods we can use are defined for us.

The constructor is the first method (we can’t use an object until it is **instantiated**)

```
        OutputPin (char port, char pin);
```

The methods we use to communicate with the instance

```
void set(bool high);
        void setHigh();
        void setLow();
        void toggle();
```

Note
Just because we have used an I/O pin as our first object don’t just think that objects can only model concrete things they can model conceptual things like a ‘meeting’ or a ‘time’ and they can model processes such as ‘sort’ or ‘read’, or ‘send’.

Class InputPin
Before we can connect a switch we have to instantiate (create an instance of) the class InputPin.

```
InputPin sw0('A',0, 1);
```

In the file “InputPin.h” we have the methods available to us:
The constructor for class InputPin is:

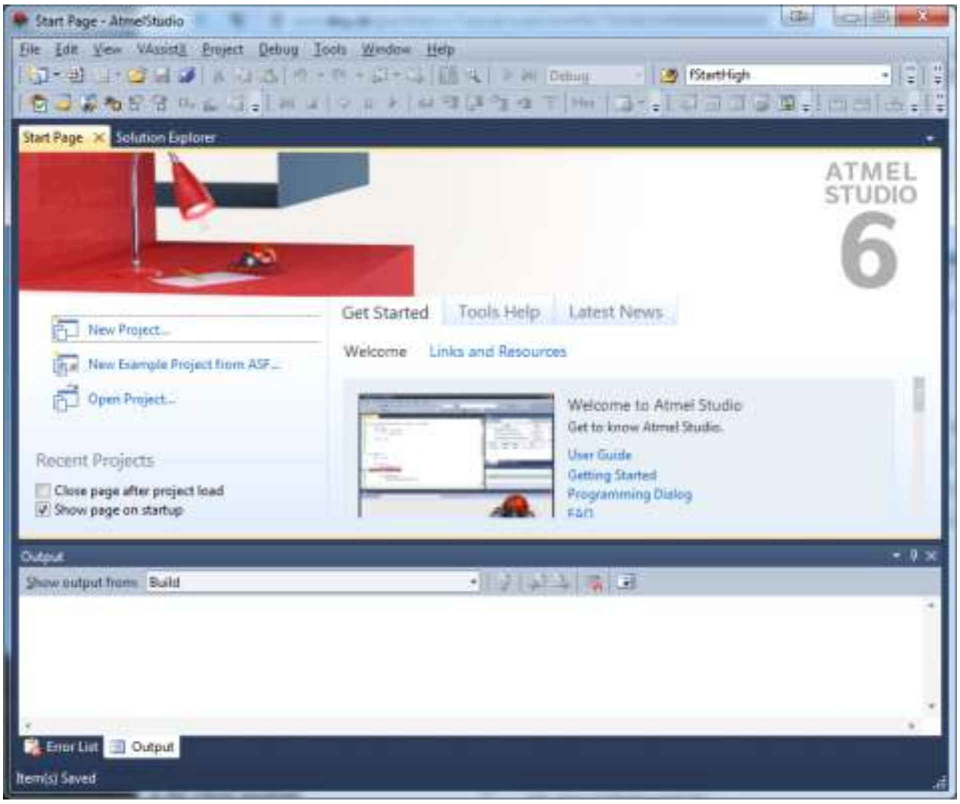
```
InputPin (char port, char pin, bool pullup);
```

This describes the input pin in terms of its register e.g PortB and its pin e.g. 3 and whether the pullup resistor is active or not. The methods to communicate with the instance of each InputPin are:

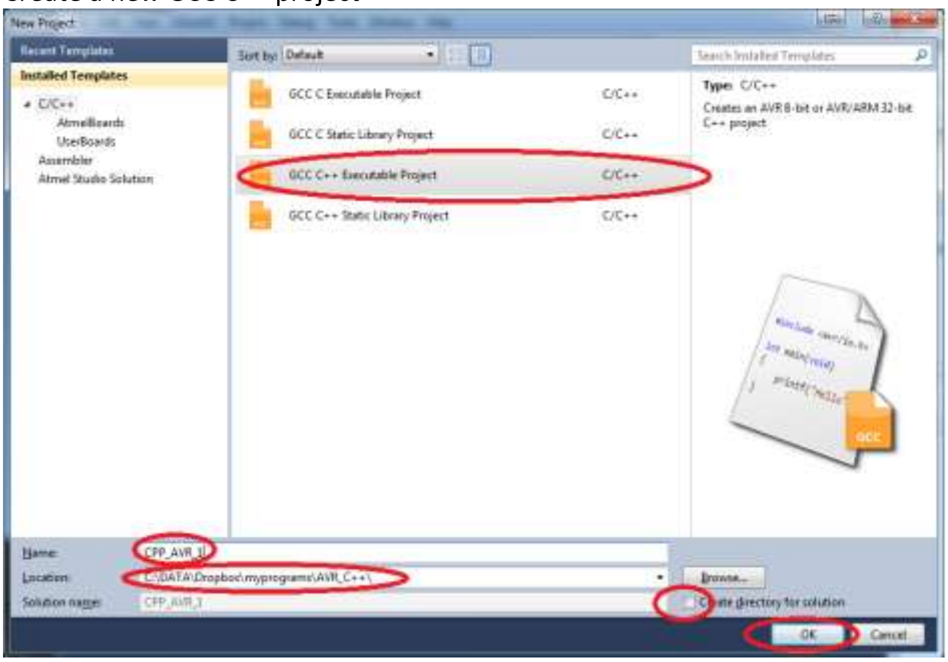
```
bool isHigh();
        bool isLow();
```

62.4 Creating an AVR CPP program in Atmel Studio 6

Start Atmel Studio

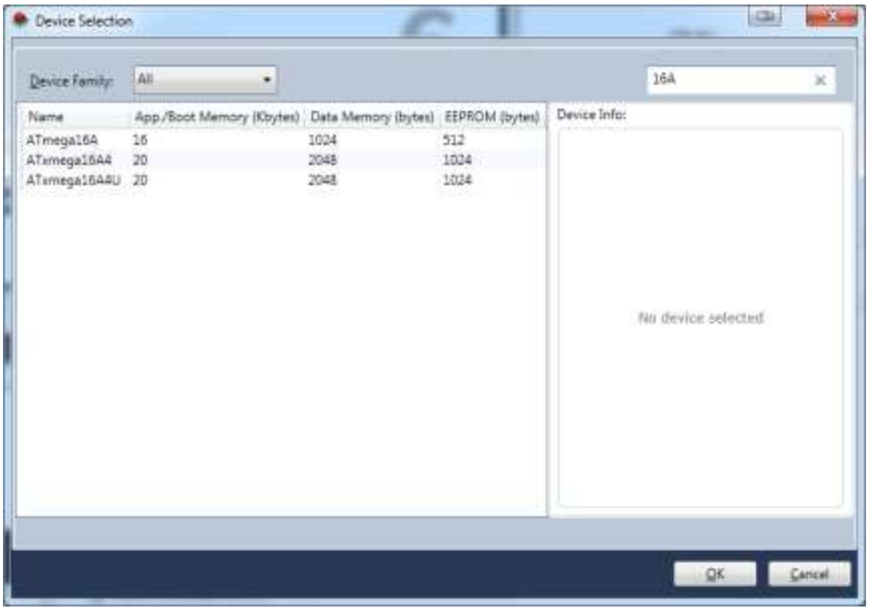


Create a new GCC C++ project

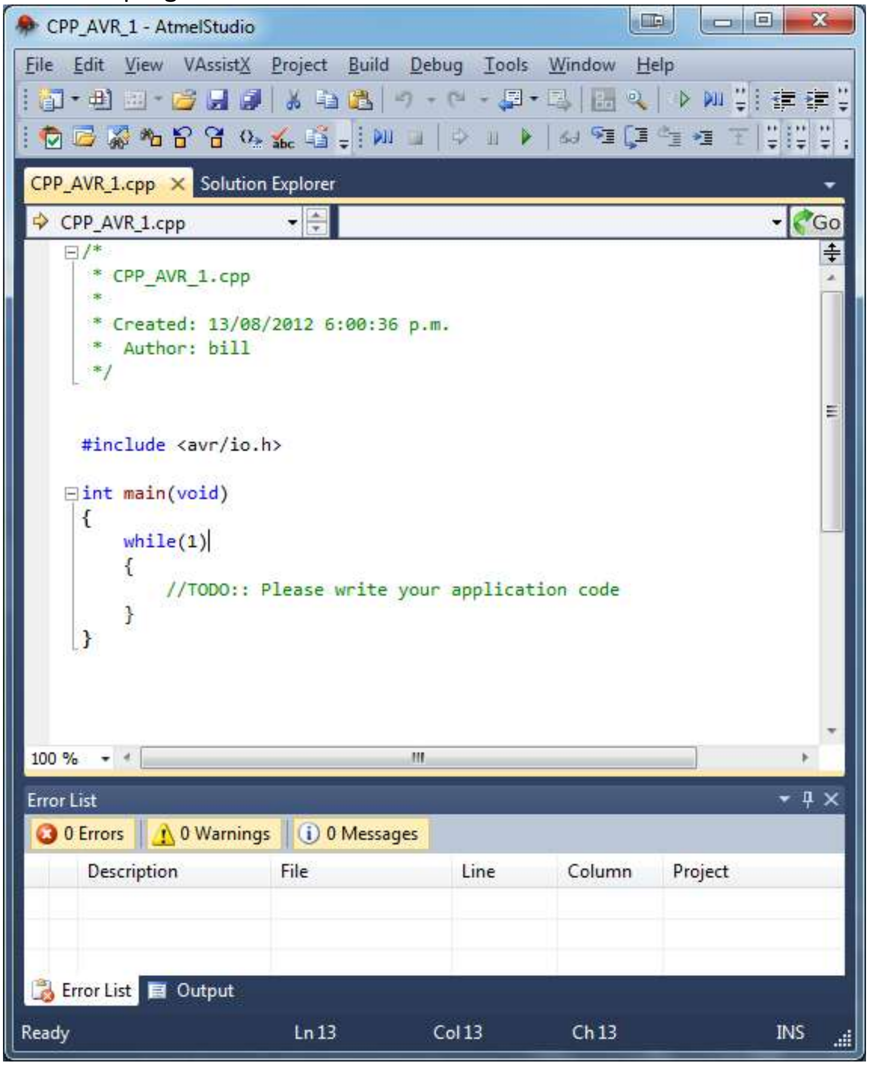


It's best to start thinking about order for your projects early on; I choose to save all my AVR projects into different folders depending upon the language I write them in. So this will go into my folder AVR_C++. Note that even though I don't select Create a directory for the solution it still creates a directory CPP_AVR_1, but it doesn't create the subdirectories within that directory.

Next choose the microcontroller you will be using. You can select it from the list or type in part of the name and then select it.



Your new program will start like this…

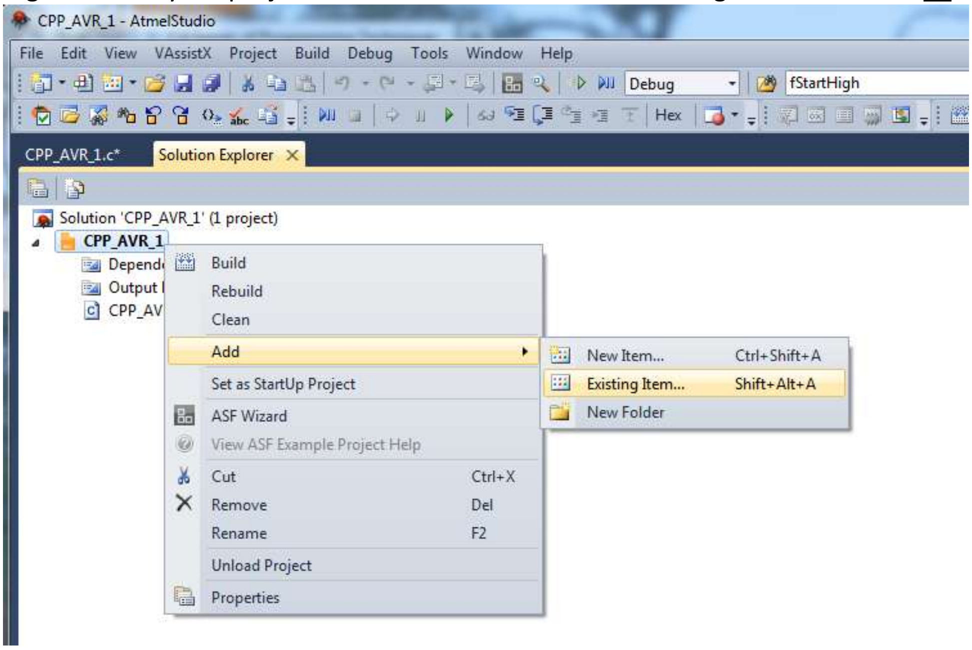


This program can be compiled by pressing F7.

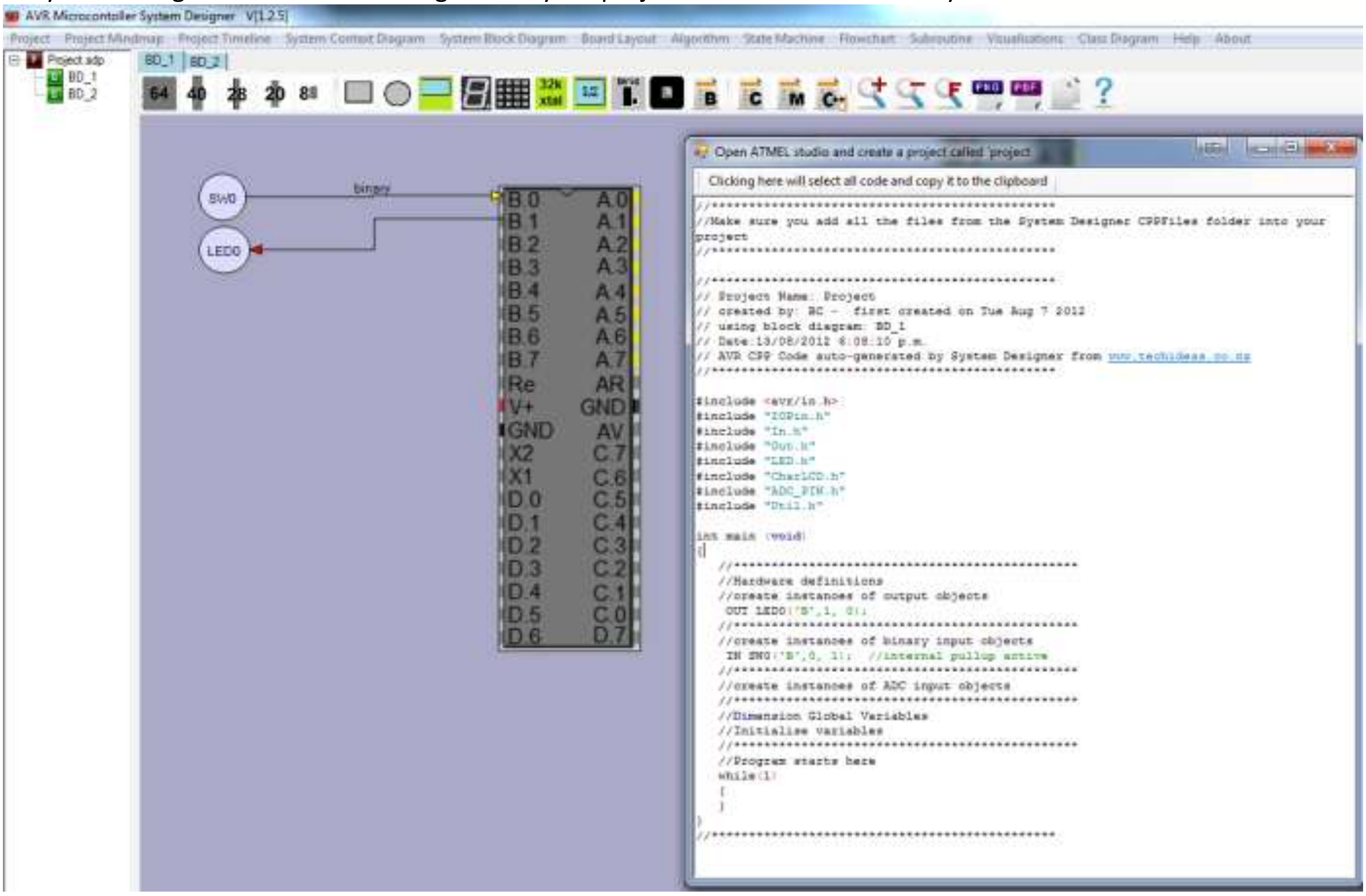
62.5 Adding our class files to the project

In the folder CPPFiles under System Designer you will find a number of files, these must be added to your solution. If you don't see a tab labelled Solution Explorer go onto the menu and choose View then Solution Explorer.

Right click on your project and choose ADD then and Existing Item and select all the files in the folder.



In System Designer create a block diagram for your project and then automatically create the CPP code for it.



Then copy all of this program or the parts of it you want through to your CPP project.

62.6 First Input and output program

```
#include <avr/io.h>
#include "IOPin.h"
#include "InputPin.h"
#include "OutputPin.h"
#include "Led.h"
#include "CharLCD.h"
#include "AdcPin.h"
#include "Util.h"

int main (void)
{
   //***********************************************
   //Hardware definitions
   //create instances of output objects
    OutputPin led0('B',1, 0); //initially off
   //***********************************************
   //create instance of binary input objects
    IinputPin sw0('B',0, 1);  //internal pullup active
   //***********************************************
   //Program starts here
   while(1)

   {
       if (sw0.IsLow())
       {
              led0.setHigh(); //on
              delay_ms(1500);
              ledD0.setLow(); //off
       }
   }
}
```

Take note of how we use the methods to interact with each object, this is the dot operator, it allows us to **reference** the methods of the object.

Overloading

If you have been following carefully so far you will may have noticed that OutputPin has been used differently in the above examples

In the first example the constructor was

```
       OutputPin (char port, char pin);
```

While in the second example it was

```
       OutputPin (char port, char pin, bool start);
```

This is called overloading of methods and is a neat feature of OOP that allows us a wide range of control.

Say in C we have a function that adds two 8 bit numbers, e.g.

```
char add_2_numbers (char X, char Y);
```

What if we want a function that adds 2 16 bit numbers as well, we would have to call it a different name e.g.

```
signed int add_2_16bit_numbers (signed int X, signed int Y);
```

This gets kind of annoying and can become confusing; in CPP there is a better way and it is called overloading. So we can define all of these methods within one class

```
char add_2_numbers (char X, char Y);
signed char add_2_numbers (signed char X, signed char Y);
signed int add_2_numbers (signed int X, signed int Y);
signed long int add_2_numbers (signed long int X, signed long int Y);
```

62.7 Class OutputPin

```
/*	OutputPin.h
*	Definition of class that handles output pins.
*/

#pragma once
#ifndef OUTPUT_PIN_DEFINED
#define OUTPUT_PIN_DEFINED

#include <avr/io.h>
#include "IOPin.h"

class OutputPin : public IOPin
{
public:
	/* constructors */
	OutputPin(char port, char pin, bool start);
	OutputPin(char port, char pint);

	void set(bool high); // set high if true, low if false
	void setHigh();
	void setLow();
	void toggle();

};

#endif //OutputPin
```

62.8 Class InputPin

```
/*	InputPin.h
*	definition of class that handles input pins
*/
#pragma once
#ifndef INPUT_PIN_DEFINED
#define INPUT_PIN_DEFINED

#include <avr/io.h>
#include "IOPin.h"

class InputPin: public IOPin
{
public:
	//constructor
	InputPin (char port, char pin, bool pullup);


};

#endif //INPUT_PIN_DEFINED
```

62.9 Inheritance

Do you notice that the class InputPin has the ability to read whether the PIN is high or low but that the two functions to do these are not actually inside In.h

The classes InputPin and OutputPin are actually **derived** from another class called IOPin, and so any functionality in IOPin is **inherited** by the classes INPUT_PIN and OutputPin

This inheritance relationship is declared in the lines in the header files (.h files)

```
class InputPin: public IOPin
class OutputPin : public IOPin
```

62.10 Class IOPin

The file IOPin.h is

```
/*      IOPin.h
*       definition for class that handles control of i/o pins
*/
#pragma once
#ifndef IOPIN_DEFINED
#define IOPIN _DEFINED

#include <avr/io.h>

class IOPin
{
public:
        // constructor
        IOPin(char port, char pin);

        bool isHigh();
        bool isLow();
        inline uint8_t getBit() const {return avrBit;}; // we won't cover inline or const yet
        inline bool isValid() const {return pinValid;};

protected:
        void setAsInput();
        void setAsOutput();
        void activatePullup();
        void deActivatePullup();

private:
        uint8_t             avrPin;       // The bit mask for this pin
        volatile uint8_t*   portReg;      // The port register
        volatile uint8_t*   pinReg;       // The PIN register
        volatile uint8_t*   dataDirReg;   // The data direction register
        bool                pinValid;     // Does the constructor define a valid pin
};

#endif
```

Any instance of class INPUT_PIN and class OutputPin has access to the functions IsHigh() and IsLow() because they **inherit** those methods from class IOPin.

62.11 Encapsulation

The methods and properties of class IOPin are declared in two different sections of the class. Public and Private. A programmer doesn't normally give direct access to the data or properties within a class they do it via methods that define the exact nature of the control we want to release to the outside world. This is the concept of **encapsulation** and is one of the most powerful aspects of OOP; the programmer who writes the class defines what information is **public** and what is **private**. The programmer who uses the object can have access to public things inside the object; e.g. with a car we have **public** access to the brake, hand brake, accelerator etc. We don't have access to the distributor, brake pads and wiper motors directly these are **private**. This shows us one major power of OOP in that there is the ability to hide or protect properties or internal data from being set into some incorrect state.

62.12 Access within a class

In the IOPin class there are three different levels of access given to users of the class: **Public**, **Protected** and **Private**. Why? Well it's all about control, what attributes we give users of the class.

There are some things that anyone who uses the class must have access to, these are the public things.
We don't want any old program or programmer changing the IO port or pin or validity of the pin directly so we make these private.

But classes that **inherit** IOPin can have some other protected access as well, so class OutputPin and InputPin can set the pin as input or output and activate the pullup resistor or deactivate it; whereas the programmer who writes a normal application and just uses IOPins doesn't need this level of control.

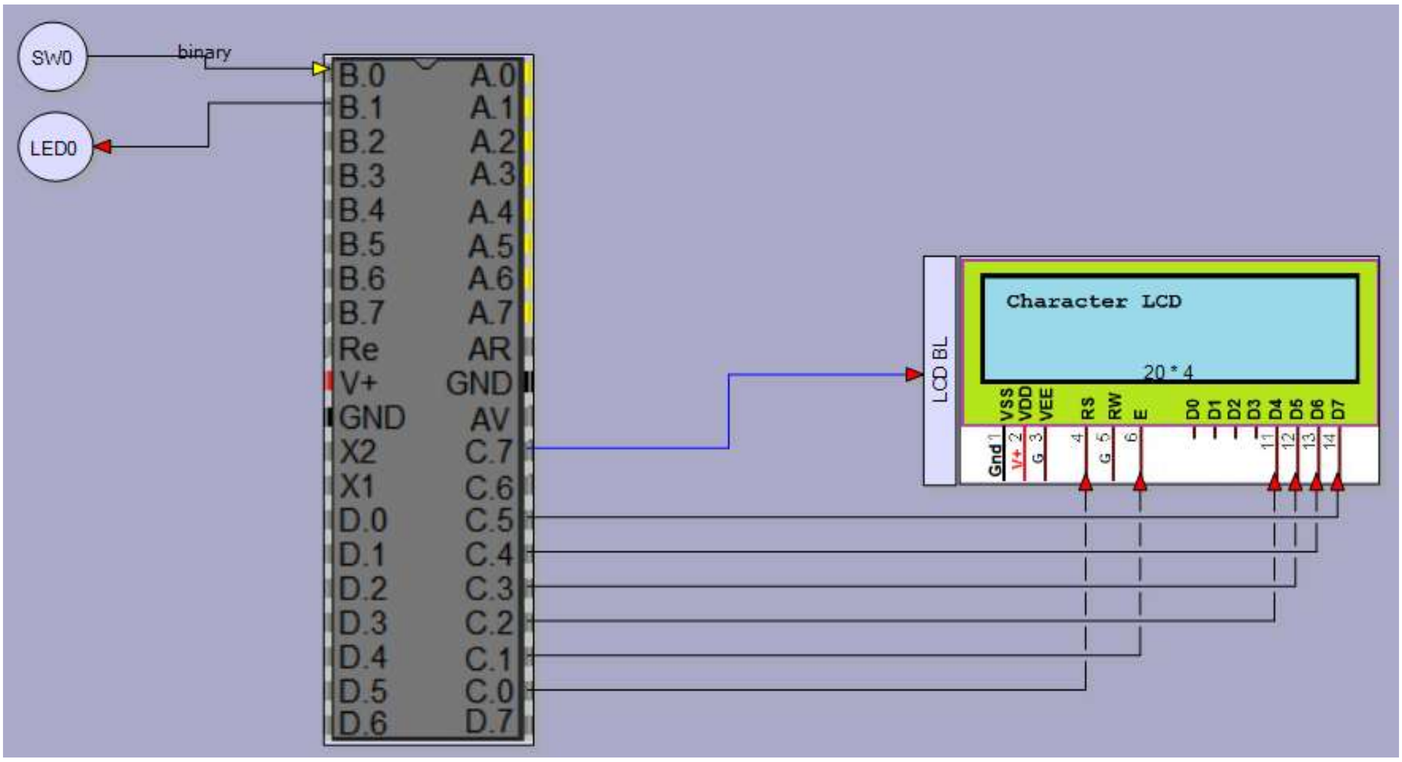
Access	public	protected	private
same class	Yes	Yes	Yes
derived class	Yes	Yes	No
Not related	Yes	No	No

Notice in the above descriptions the different words **'uses'** and **'inherits'**. This distinction is very important in OOP. We often refer to it as **'is a' (inherits)** or **'has a' (uses)** type of relationship. InputPin inherits from IOPin so it 'is a' therefore it has access to all the public and protected things in IOPin. A class that uses an IOPin will not have access to the protected things inside IOPin just the public things.

So when you define InputPin you want to be able to change the IOPin to an input using the method setAsInput(); but when you make some device that uses an InputPin you don't want it to be able to change it from input to output.

62.13 Class Char_LCD

Here we will look at a 'uses a' type of relationship. A character LCD uses 6 pins of the microcontroller (it doesn't extend or inherit from the OutputPin class).




```
/*      CharLCD.h
*       definition of class that handles Character LCDs.
        note that he rw pin is unused
*/

#pragma once
#ifndef CharLCD_DEFINED
#define CharLCD_DEFINED

#include <avr/io.h>
#include <stdlib.h>
#include "Output_pin.h"
#include "IOPin.h"
#include "Util.h"

//some commands take time for the LCD to process
const int LCD_DELAY_US=100;
const int LCD_DELAY_MS=2;


//LCD commands
#define LCD_CLR             0x01        // clear LCD
#define LCD_HOME            0x02        // cursor to home position
// see animations http://www.geocities.com/dinceraydin/lcd/commands.htm for the following commands
#define LCD_SHIFTDEC        0x04        // decrement address counter, display shift off
#define LCD_DEC             0x05        // decrement address counter, display shift on
#define LCD_INC             0x06        // Increment address counter, display shift off - default
#define LCD_SHIFTINC        0x07        // Increment address counter, display shift on

#define LCD_ALL             0x0F        // LCD On, LCD display on, cursor on and blink on
#define LCD_ON              0x0C        // turn lcd on/no cursor
#define LCD_OFF             0x08        // turn lcd off
#define LCD_ON_DISPLAY      0x04        // turn display on
#define LCD_ON_CURSOR       0x0E        // turn cursor on
#define LCD_ON_BLINK        0x0F        // cursor blink
```

```
#define LCD_X0Y0          0x80        // cursor Pos on line 1 (or with column)
#define LCD_X0Y1          0xC0        // cursor Pos on line 2 (or with column)
#define LCD_X0Y2          0x94        // cursor Pos on line 3 (or with column)
#define LCD_X0Y3          0xD4        // cursor Pos on line 4 (or with column)
#define LCD_CURSOR_LEFT   0x10        //move cursor one place to left
#define LCD_CURSOR_RIGHT  0x14        //move cursor one place to right
#define LCD_BUSY 0x80      /* DB7: LCD is busy- not used      */

class CharLCD
{
public:
       /* constructor  */
       CharLCD(OutputPin* lcdRs, OutputPin* lcdEn, OutputPin* lcdD4, OutputPin* lcdD5, OutputPin*
lcdD6, OutputPin* lcdD7, char lcdChars, char lcdLines);

       void init();               //Initialise the display, this is called by the constructor so
users do not normally need to call it
       void Off();
       void On();
       void cursorOn();
       void lcd_cursorOff();      //default
       void cursorBlink();
       void cls();
       void home();
       void cursorXY(char x, char y);
       void line0();
       void line1();
       void line2();
       void line3();
       void disp(const char *str);
       void disp(signed char n);
       void disp(uint8_t n);
       void disp_bin(uint8_t n);
       void disp_bin(signed char n);
       void disp(unsigned int n);
       void disp(signed int n);
       void disp_bin(unsigned int n);
       void disp_bin(signed int n);

private:
       void command(uint8_t dat);
       void lcd_data(uint8_t dat);
       void lcd_write(uint8_t dat, char rs);
       void lcd_pulse_en();
       uint8_t rows;
       uint8_t cols;
       OutputPin* _rs;
       OutputPin* _rw;
       OutputPin* _en;
       OutputPin* _dat4;
       OutputPin* _dat5;
       OutputPin* _dat6;
       OutputPin* _dat7;
};

#endif
```

There are a great many methods in this class, note that not all are public. The methods: command, lcd_data, lcd_write, lcd_pulse are private, so that means only this class has access to them.

Take note of the constructor it is quite long because we have to pass 6 pins and 2 other variables as parameters to it for the connections and the size of the LCD.

```
CharLCD(OutputPin* lcdRs, OutputPin* lcdEn, OutputPin* lcdD4, OutputPin* lcdD5, OutputPin* lcdD6,
OutputPin* lcdD7, char lcdChars, char lcdLines);
```

Note that the class has pointers to other classes passed to it. So to use the CharLCD class we need to instantiate 6 OutputPin objects and pass their addresses to CharLCD instance like this

```
    OutputPin lcd_rs('C',0);
    OutputPin lcd_en('C',1);
    OutputPin lcd_dat4('C',2);
    OutputPin lcd_dat5('C',3);
    OutputPin lcd_dat6('C',4);
    OutputPin lcd_dat7('C',5);
    CharLCD lcd(&lcd_rs, &lcd_en, &lcd_dat4, &lcd_dat5, &lcd_dat6, &lcd_dat7, 20, 4);
```

After that we can use the LCD with the 'disp' methods e.g.

```
lcd.disp("Hello Planet Earth");
```

If using System Designer the code can be automatically generated and produces this:

```
//**********************************************
// Project Name: Project
// created by: BC -  first created on Tue Aug 7 2012
// using block diagram: BD_3
// Date:19/08/2012 8:54:31 p.m.
// AVR CPP Code auto-generated by System Designer from www.techideas.co.nz
//**********************************************

#include <avr/io.h>
#include "IOPin.h"
#include "Input_pin.h"
#include "Output_pin.h"
#include "LED.h"
#include "CharLCD.h"
#include "AdcPin.h"
#include "Util.h"
#include "Timer2.h"

int main (void)
{
   //**********************************************
   //Hardware definitions
   //create instances of output objects
    OutputPin LCD_BL('C',7, 1);
    OutputPin led0('B',1, 0);
   //**********************************************
   //create instances of binary input objects
    INPUT_PIN sw0('B',0, 1);  //internal pullup active
   //**********************************************
   //create instances of ADC input objects
   //**********************************************
   //**********************************************
   //Character LCD config
    OutputPin lcdRs('C',0);
    OutputPin lcdEn('C',1);
    OutputPin lcdD4('C',2);
    OutputPin lcdD5('C',3);
    OutputPin lcdD6('C',4);
    OutputPin lcdS7('C',5);
    CharLCD lcd(&lcdRs, &lcdEn, &lcdD4, &lcdD5, &lcdD6, &lcdD7, 20, 4);
    lcd.init();
    lcd.disp("Project");
   //Dimension Global Variables
   //Initialise variables
   //**********************************************
   //Program starts here
   while(1)
   {
   }
}
//**********************************************
```

62.14 Exercise – create your own Led class.

An Led is really just an output pin and we don't really need to create a class for it but it is a useful first exercise in creating a class.

We will need 2 files Led.h and Led.cpp

Led.h
This tells us about the constructor and that it requires a port and a pin
It defines two funtions for us on() and off(), both return no value.

```
/*
	Led.h
	definition of class that controls pin as an Led
*/

#pragma once
#ifndef LED_DEFINED
#define LED_DEFINED

#include "OutputPin.h"

class Led : public OutputPin{

public:
	Led(char port, char pin);

	void on();
	void off();
};

#endif
```

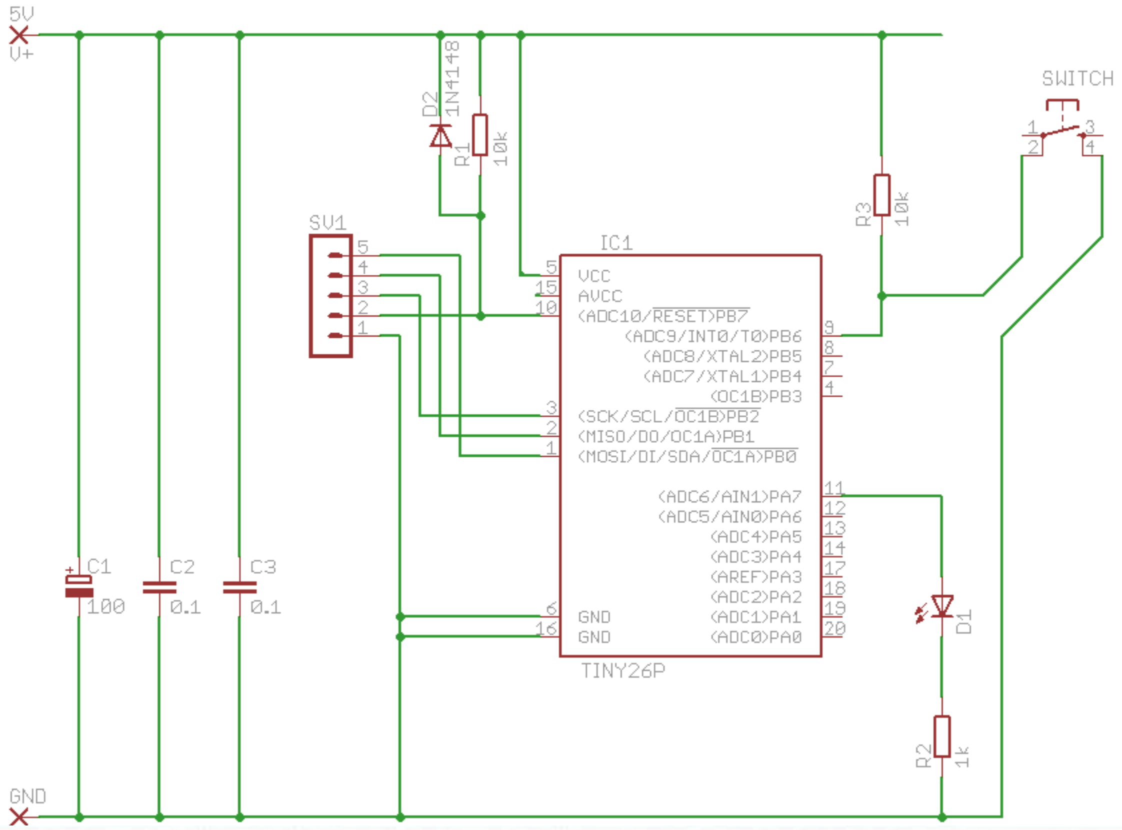
Led.cpp
This is the file with the actual code in it.

```
/*	Led.h
*	Definition of class that handles an Led.
*/

#include <avr/io.h>
#include "Led.h"

Led::Led(char port, char pin)
: OutputPin(port, pin, false)
{
	off(); //initially turn it off
}

void Led::on()
{
	setHigh();
}

void Led::off()
{
	setLow();
}
```

Here you will see that an on command just sets the pin to high and an off command just sets the pin to low.

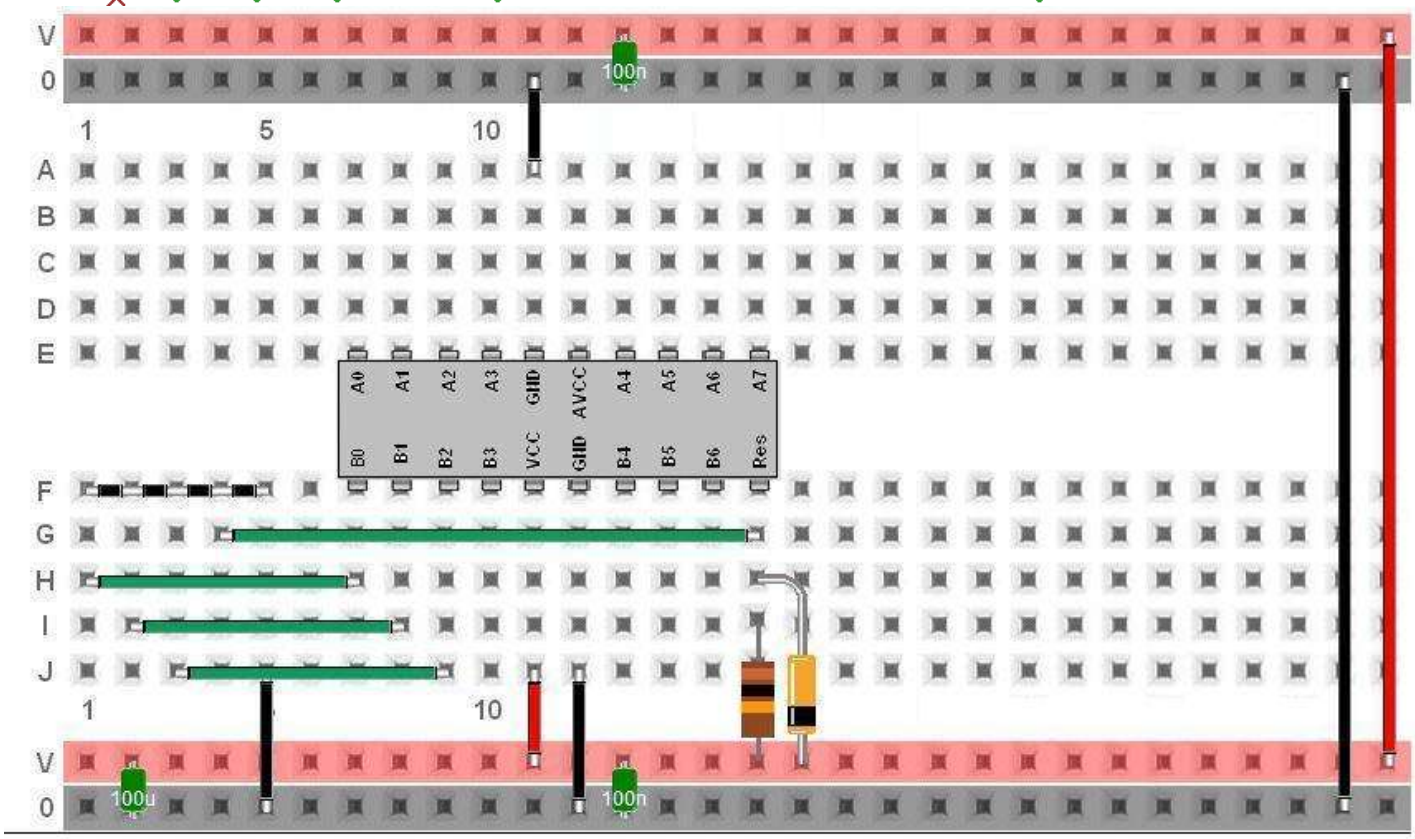
63 Alternative AVR development PCBS

63.1 ATTiny461 breadboard circuit

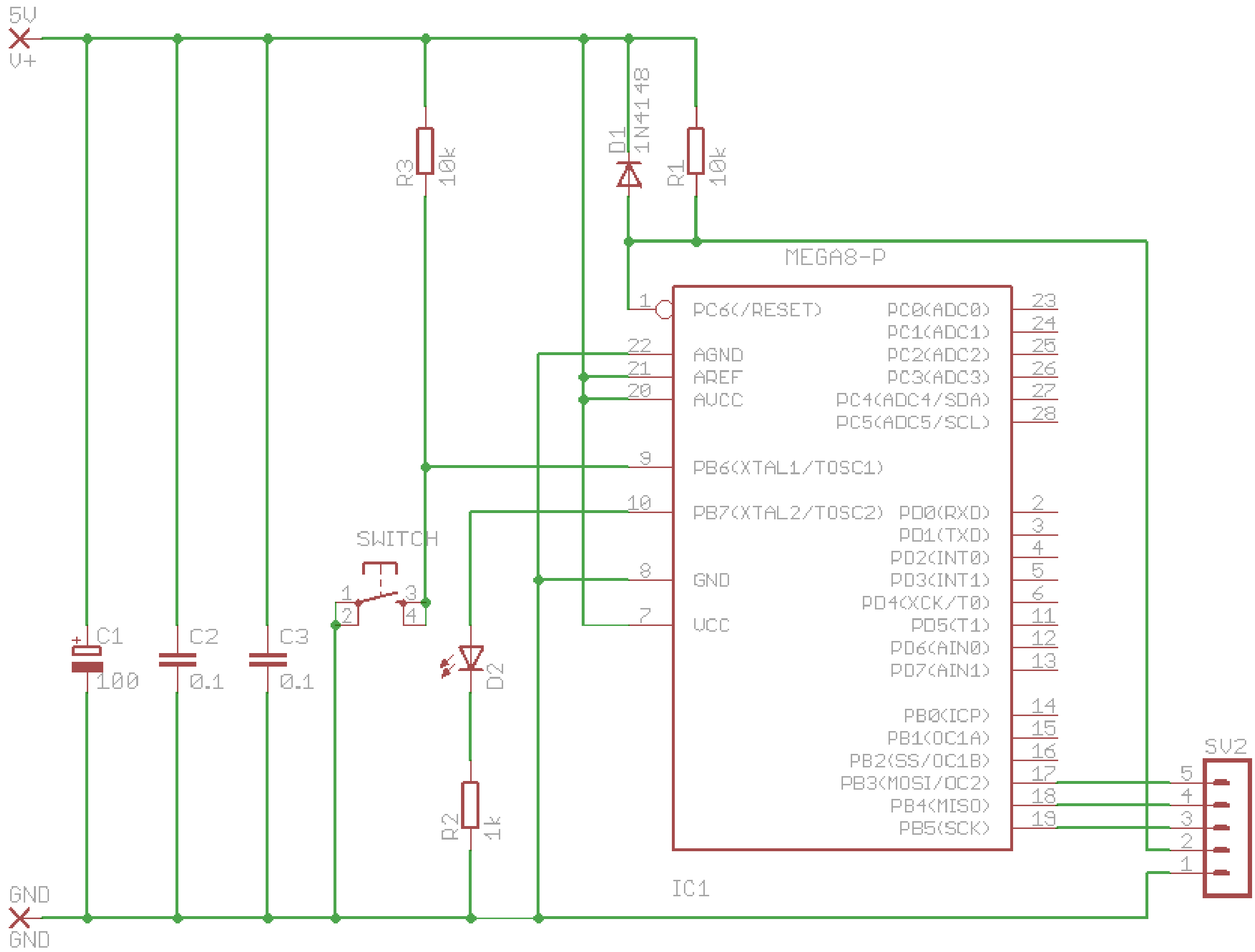


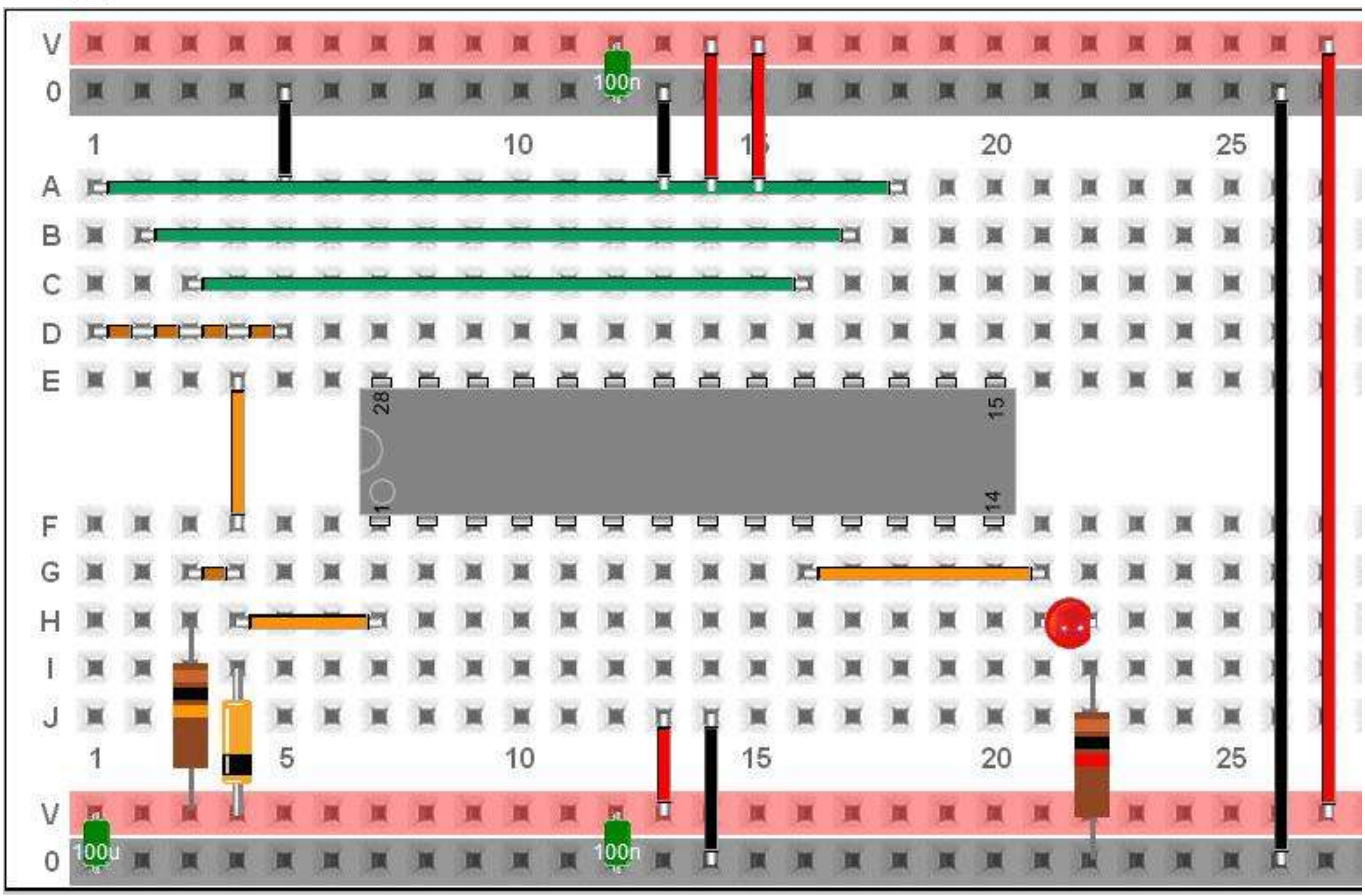
Complete the layout design for the above circuit on the breadboard diagram.

It helps to put a label on top of the IC with the pin connections on it so that you can easily identify which pin is which.



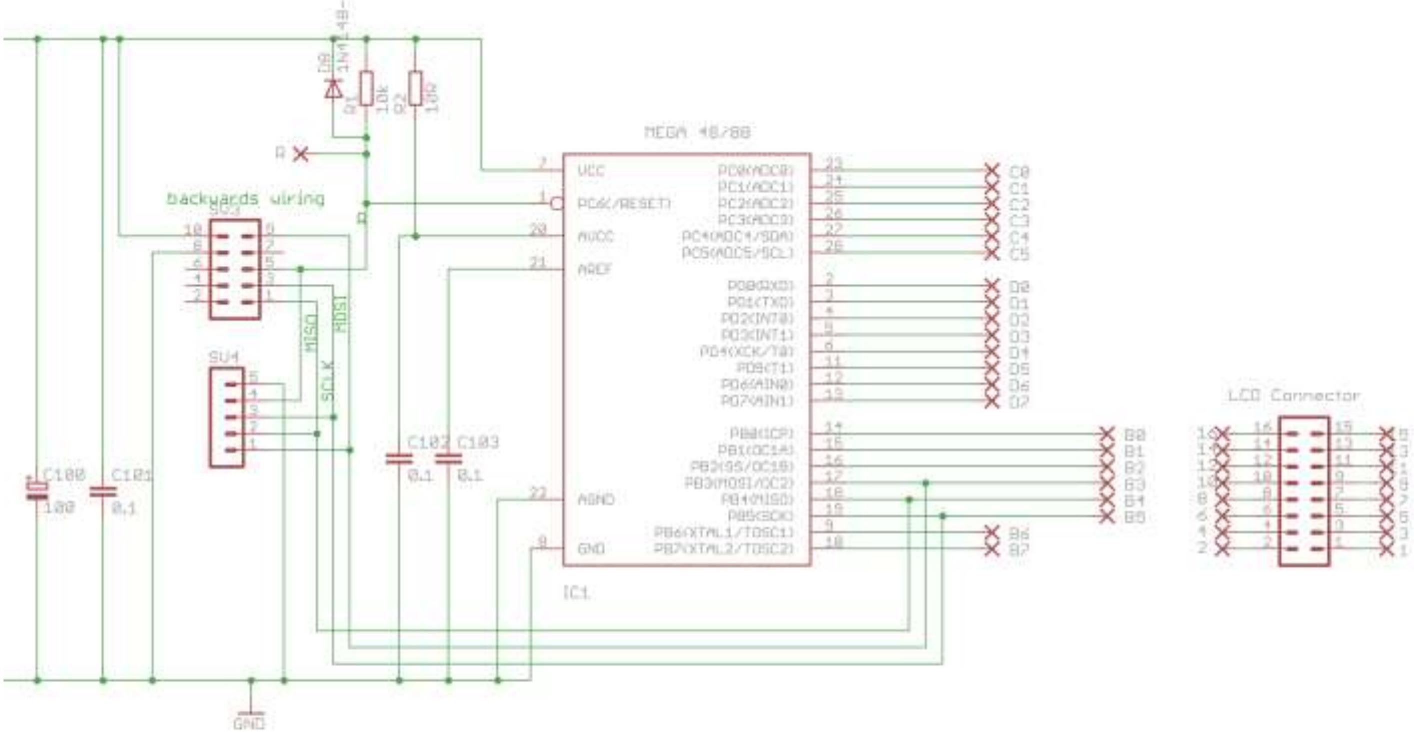
63.2 Alternative ATMega48 breadboard circuit



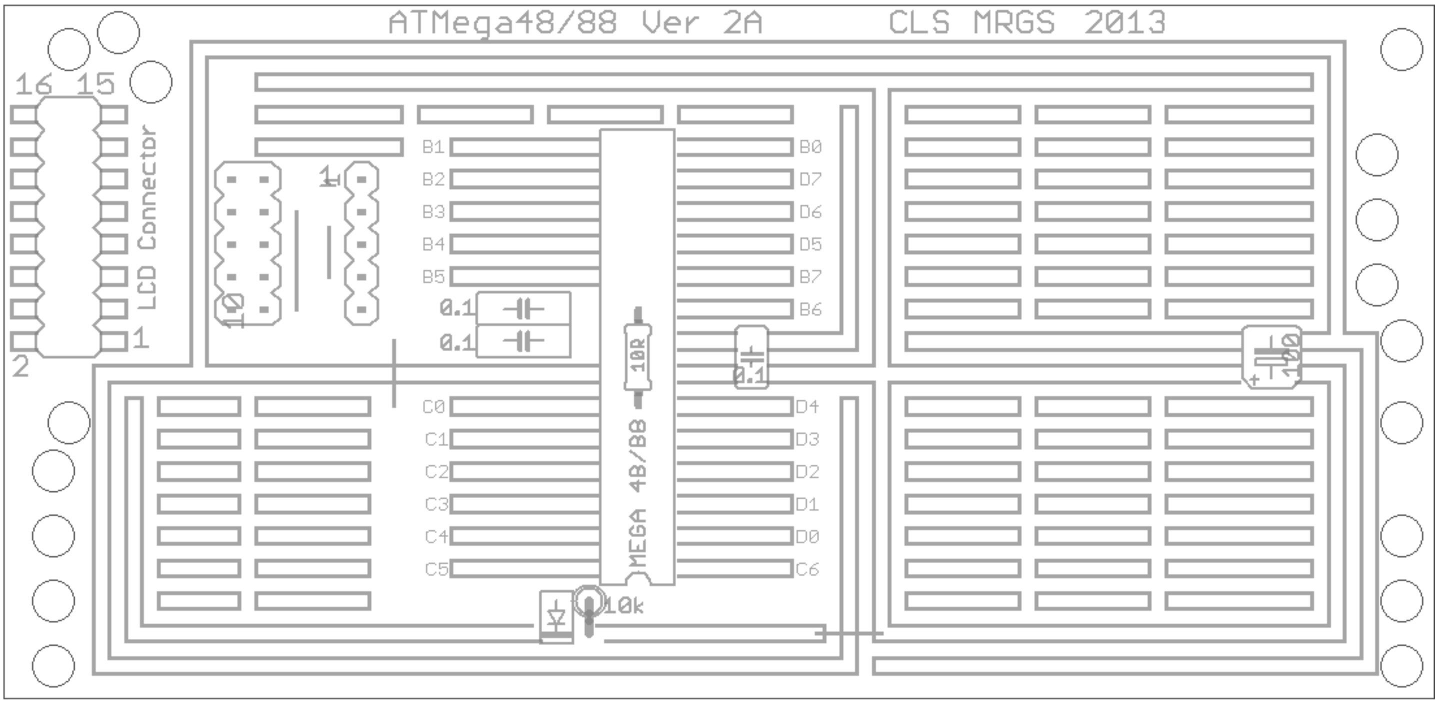


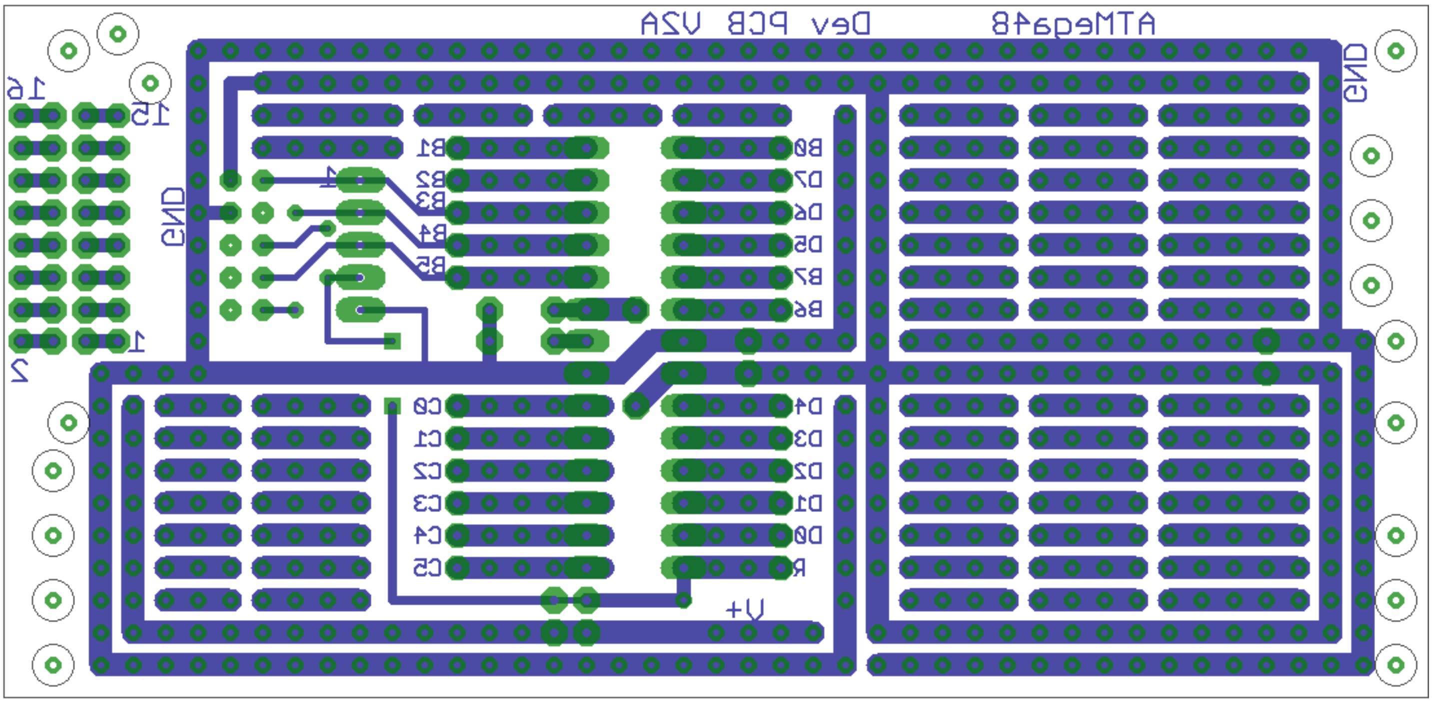
Breadboard layout

ATMega48/88 Devel PCB V2A

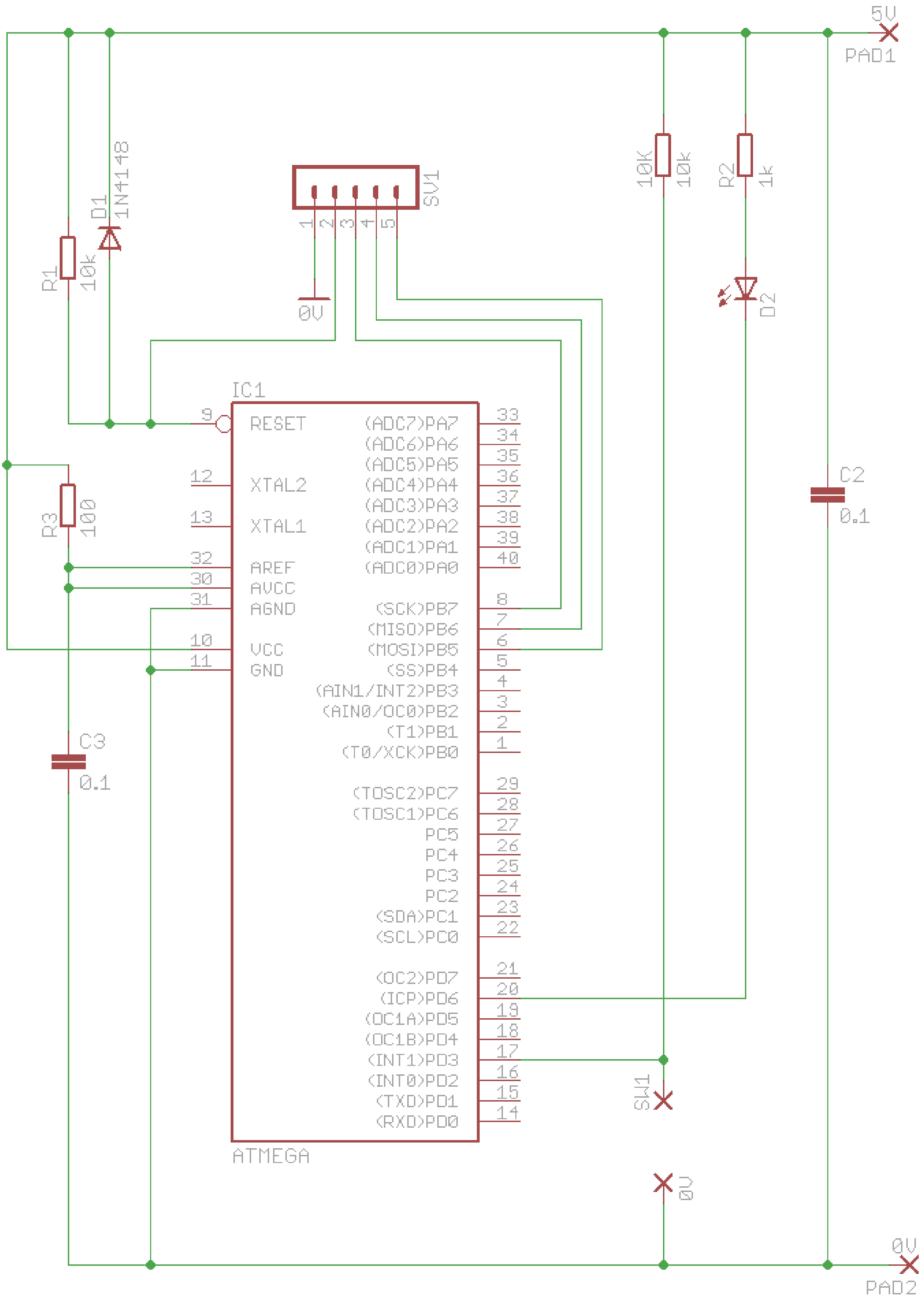


This schematic shows two programming connectors, the upper one is the standard ATMEL 10 pin programming connector as used in the USBASP programmer. The second is the modified version we have used at MRGS for many years with the parallel port. These connectors are designed to be soldered to the bottom or track side of the PCB so that the LCD doesn't interfere with the programming connector

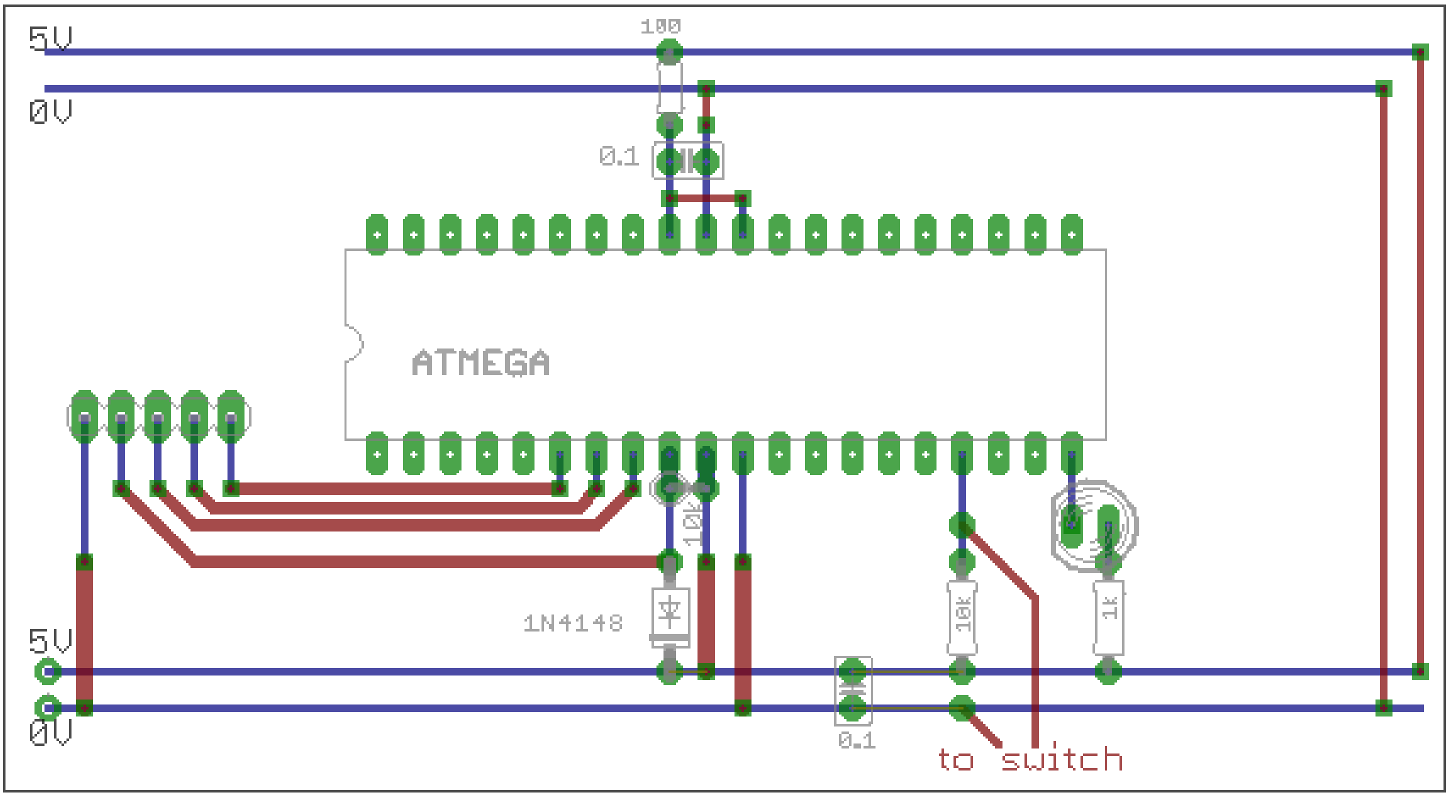

ATMega48/88 Ver 2A
CLS MRGS 2013
LCD Connector
MEGA 48/88
10k
10R


GND
+V

63.3 Alternative ATMega breadboard circuit



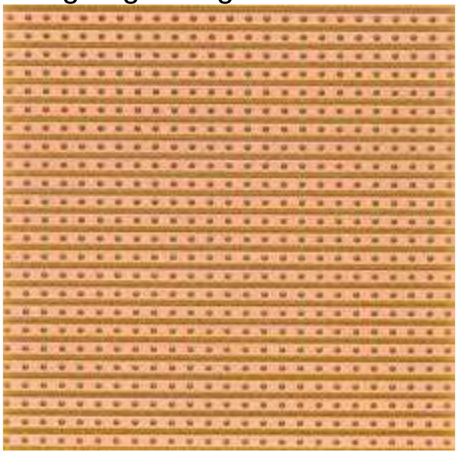
63.4 AVR circuit description



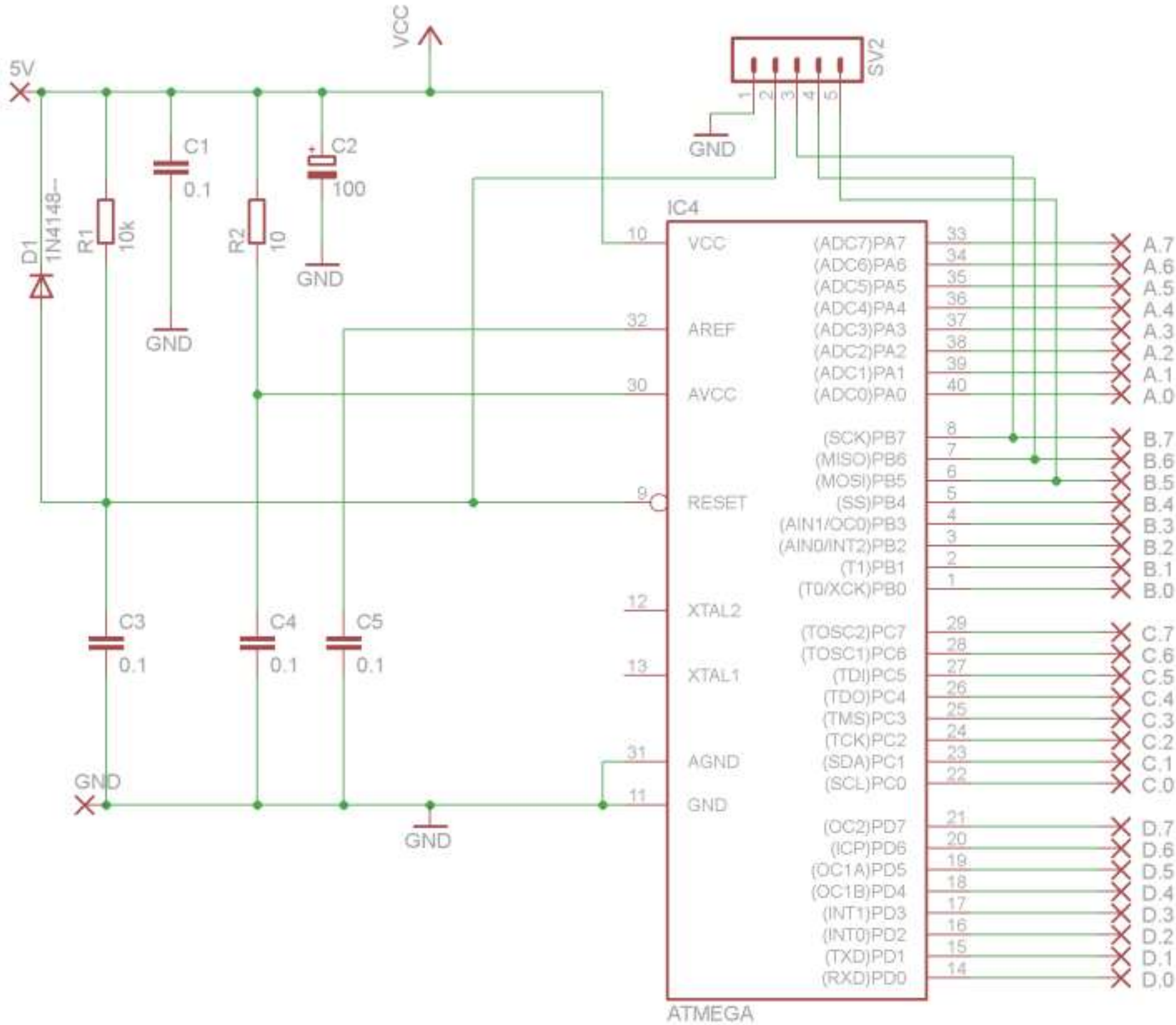
- The 5 pin header (connector) is for programming the AVR from a PC.
- The 0.1uF capacitor between 0V and VCC is to reduce any variations in power supply voltage.
- The 10k is a pull-up resistor for the reset pin, a low (connection to ground) on this pin will halt the microcontroller and when it is released(pulled high by the resistor) the program will run from the beginning again.
- The 1N4148 is a protection diode that will stop high voltages possibly damaging the microcontroller (it is only required on the reset pin because all the other microcontroller pins have built in protection diodes).
- The 0.1uF capacitor and 100R resistor are the power supply for the ADC circuit

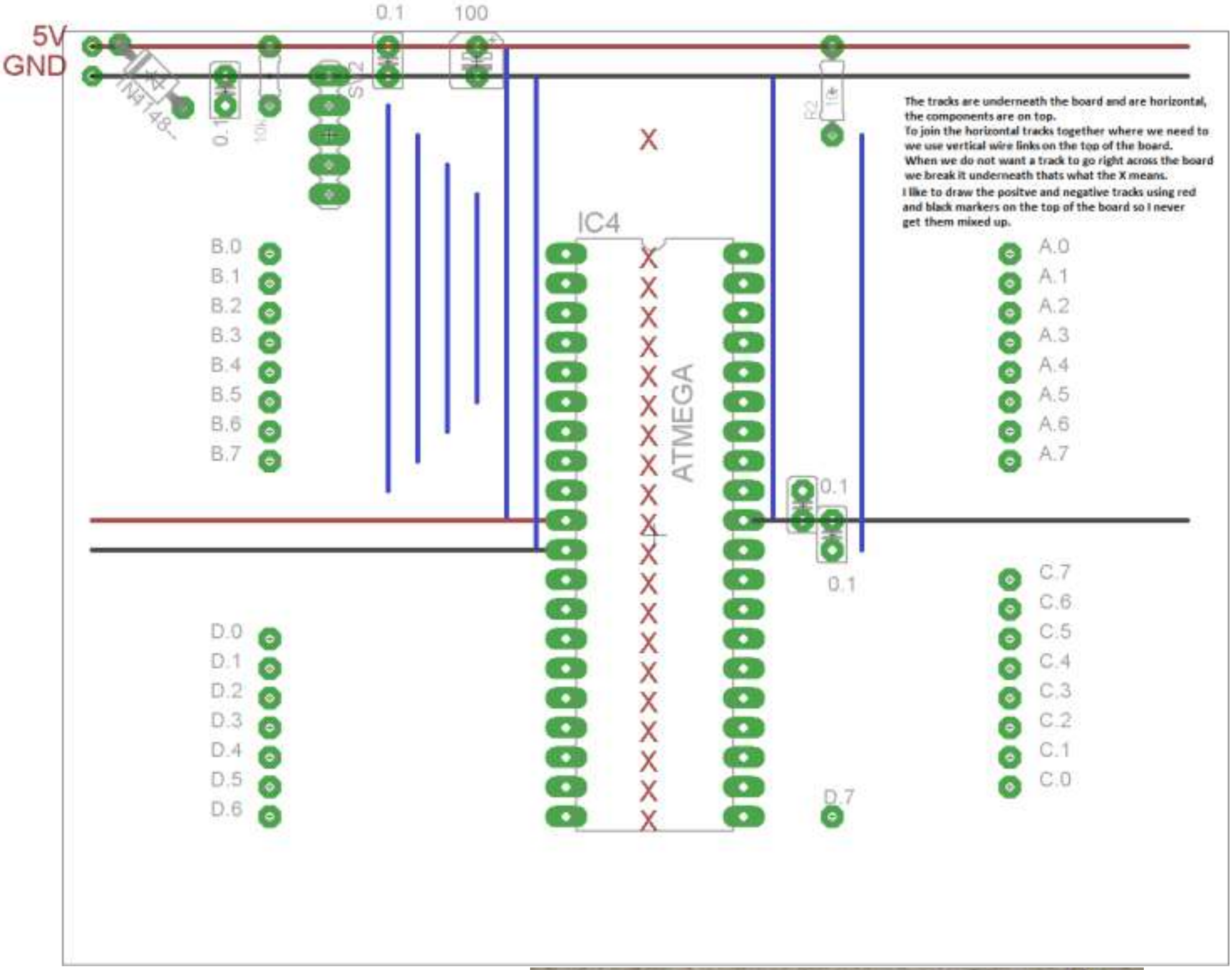
63.5 ATMega on Veroboard

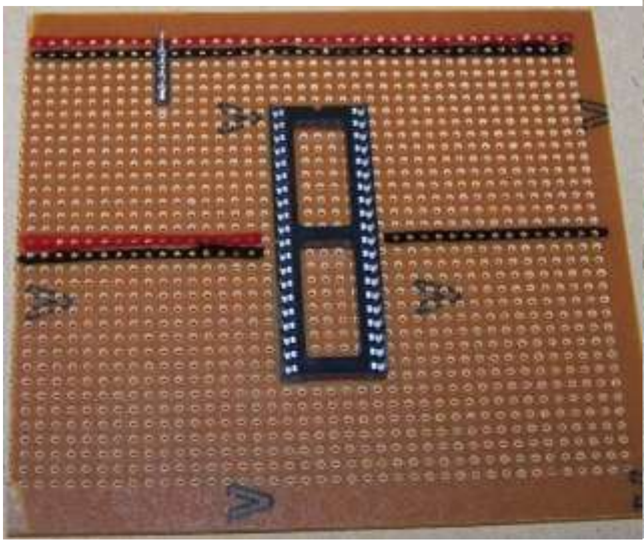
Veroboard is a smart PCB that already has holes and track. It is mre reliable and permanent than breadboard but not so easy to modify. It is much quicker to use than making a PCB when you are designing a single circuit.

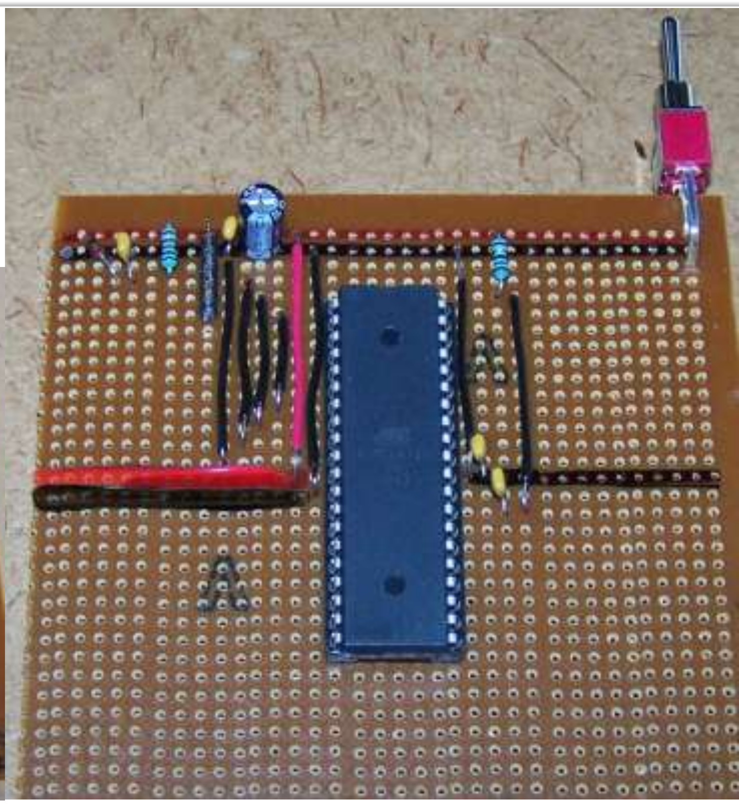


Here is an AVR schematic using an ATMega




5V
GND
1N4148
0.1
10k
SV2
0.1
100
X
R2
10k
The tracks are underneath the board and are horizontal, the components are on top.
To join the horizontal tracks together where we need to we use vertical wire links on the top of the board.
When we do not want a track to go right across the board we break it underneath thats what the X means.
I like to draw the positve and negative tracks using red and black markers on the top of the board so I never get them mixed up.
IC4
ATMEGA
B.0
B.1
B.2
B.3
B.4
B.5
B.6
B.7
A.0
A.1
A.2
A.3
A.4
A.5
A.6
A.7
0.1
0.1
D.0
D.1
D.2
D.3
D.4
D.5
D.6
D.7
C.7
C.6
C.5
C.4
C.3
C.2
C.1
C.0





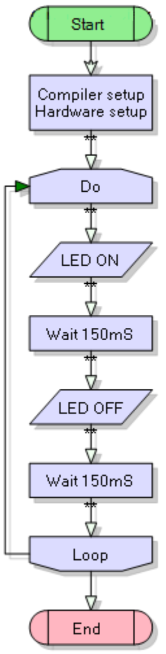
63.6 Different microcontroller starter circuit

In the code throughout this book different AVR microcontrollers are referred to in different places. So take special note of the $regfile and $crystal commands used and make sure they match the micro you are using. Note that the microcontroller all run by default at 1,000,000MHz however many of the ATMega circuits have been changed to run at 8,000,000 MHz.

You will also have to make changes to the ports used in the program: the ATTiny461 has ports A & B, the ATMega48 has B, C & D, the ATMega chips have ports A, B, C & D.

63.7 Getting started code for the ATMega48

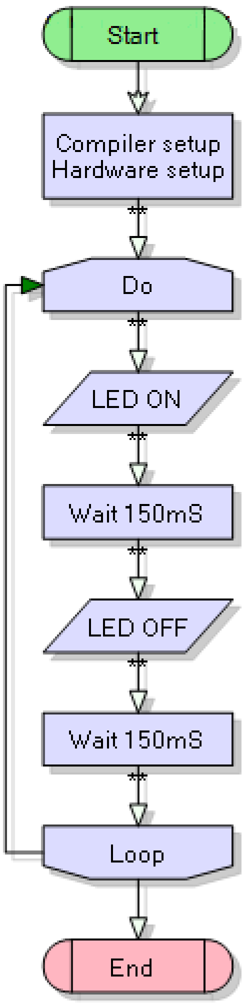
The code for the ATmega48 is similar to the ATTiny461; the code changes are underlined below




```
'Flash1led.bas  for the ATMEGA48
'B.Collis June 1009

$regfile = "m48def.dat"            ' bascom needs to know our micro
$crystal = 1000000              ' bascom needs to know how fast it is
going


Config Portb = Output            'make these 8 micro pins outputs
Const Flashdelay = 100            ' preset how long a wait will be

Do
   Set PortB.7              ' LED 7 on
   Waitms Flashdelay        'wait a preset time
   Reset PortB.7             'LED 7 off
   Waitms Flashdelay              'wait a preset time
Loop                               'return to do and start again
End
```

63.8 Getting started code for the ATMega16

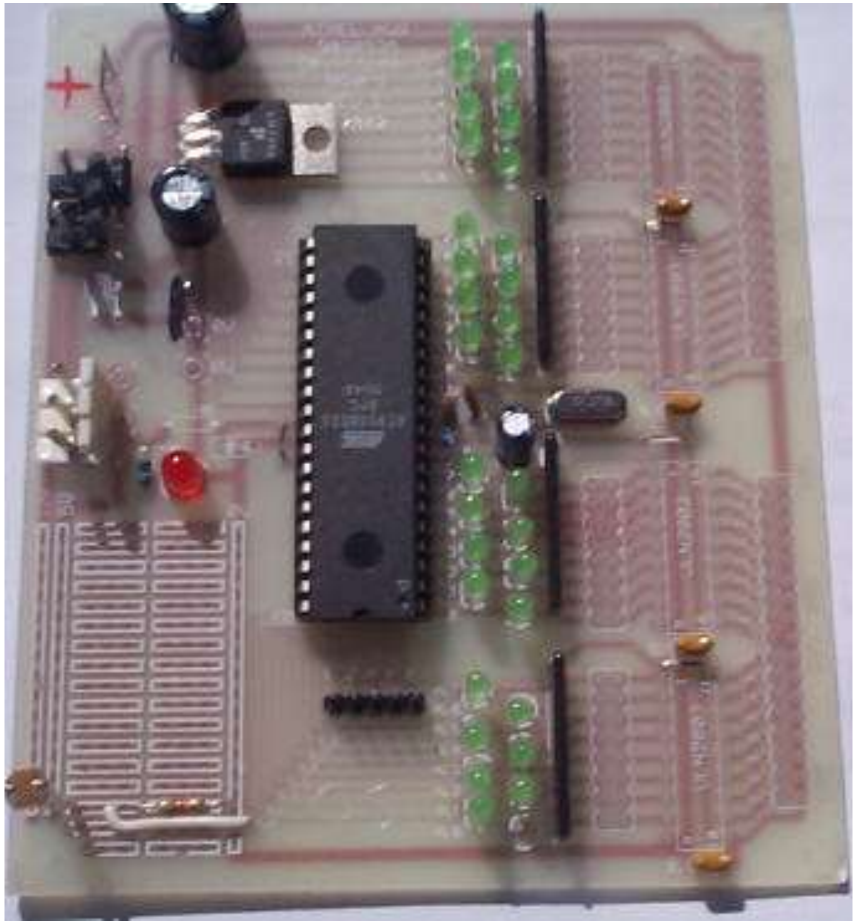
The code for the ATMega is similar to the ATTiny461; the code changes are underlined below



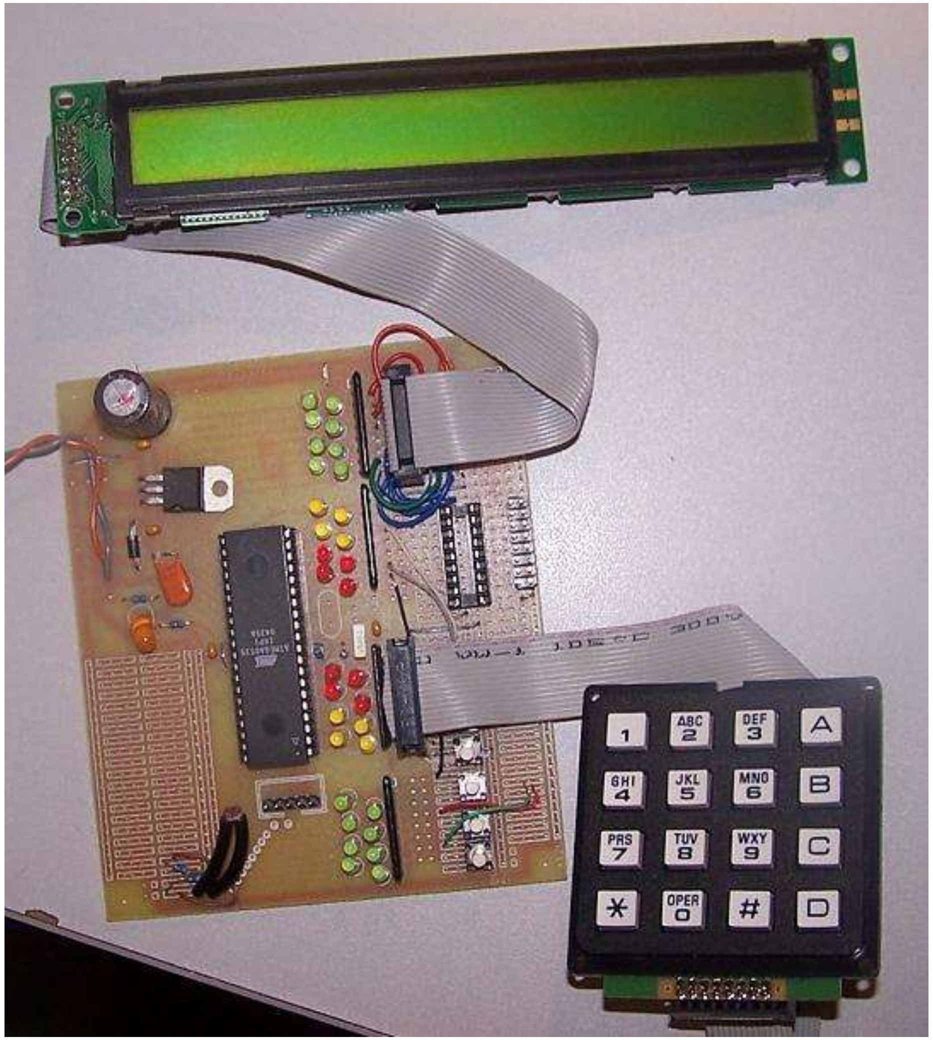

```
'Flash1led.bas  for the ATMEGA16I
'B.Collis June 1009

$regfile = "m16def.dat"          ' bascom needs to know our micro
$crystal = 1000000              ' bascom needs to know how fast it is
going


Config Portd = Output            'make these 8 micro pins outputs
Const Flashdelay = 100            ' preset how long a wait will be

Do
      Set PortD.7            ' LED 7 on
      Waitms Flashdelay      'wait a preset time
      Reset PortD.7          'LED 7 off
   Waitms Flashdelay          'wait a preset time
Loop                          'return to do and start again
End
```

Note the change to $regfile

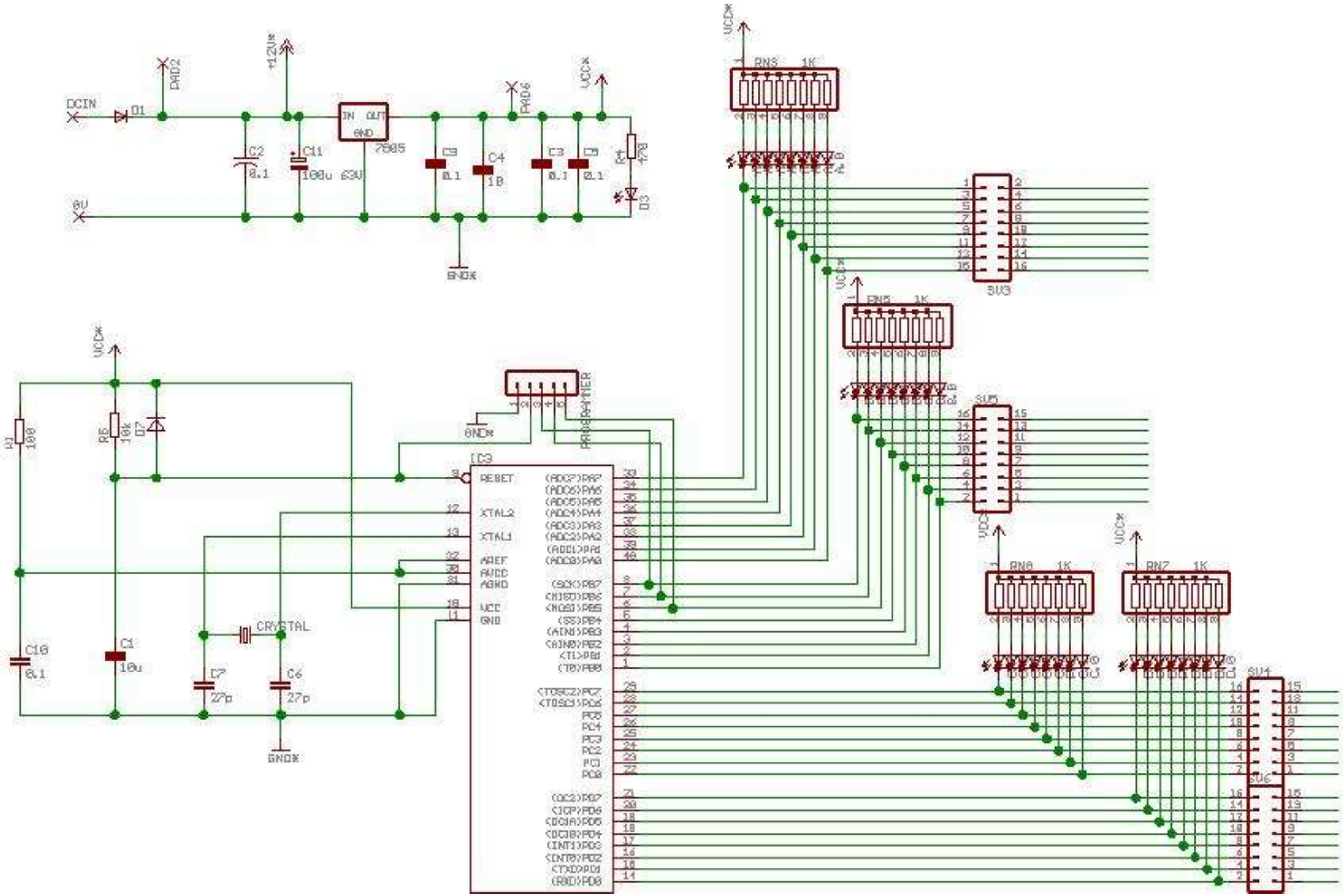
63.9 Early ATMega boards



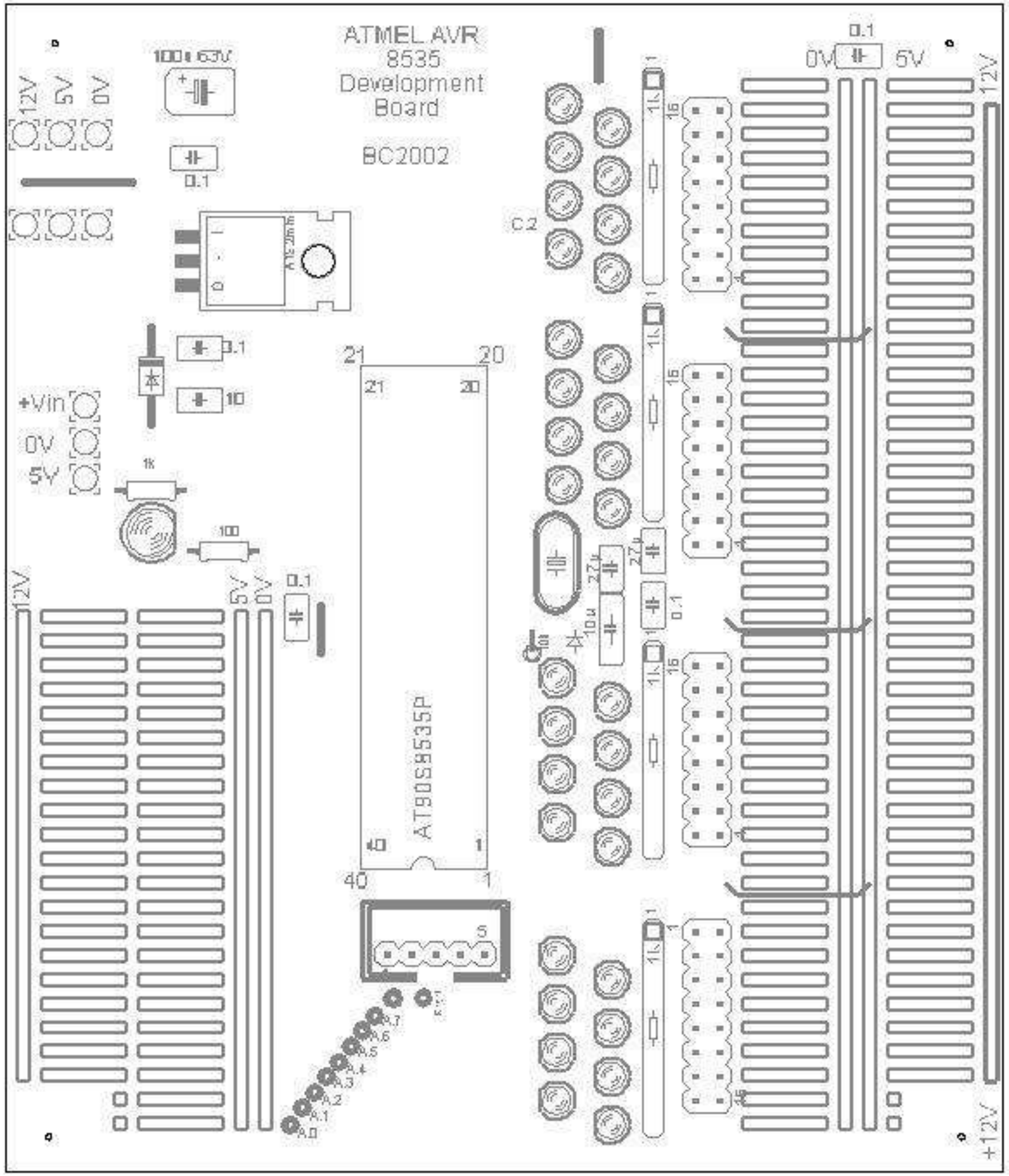
8535 Version 1A (OLD)



8535 Version 1 (OLD)

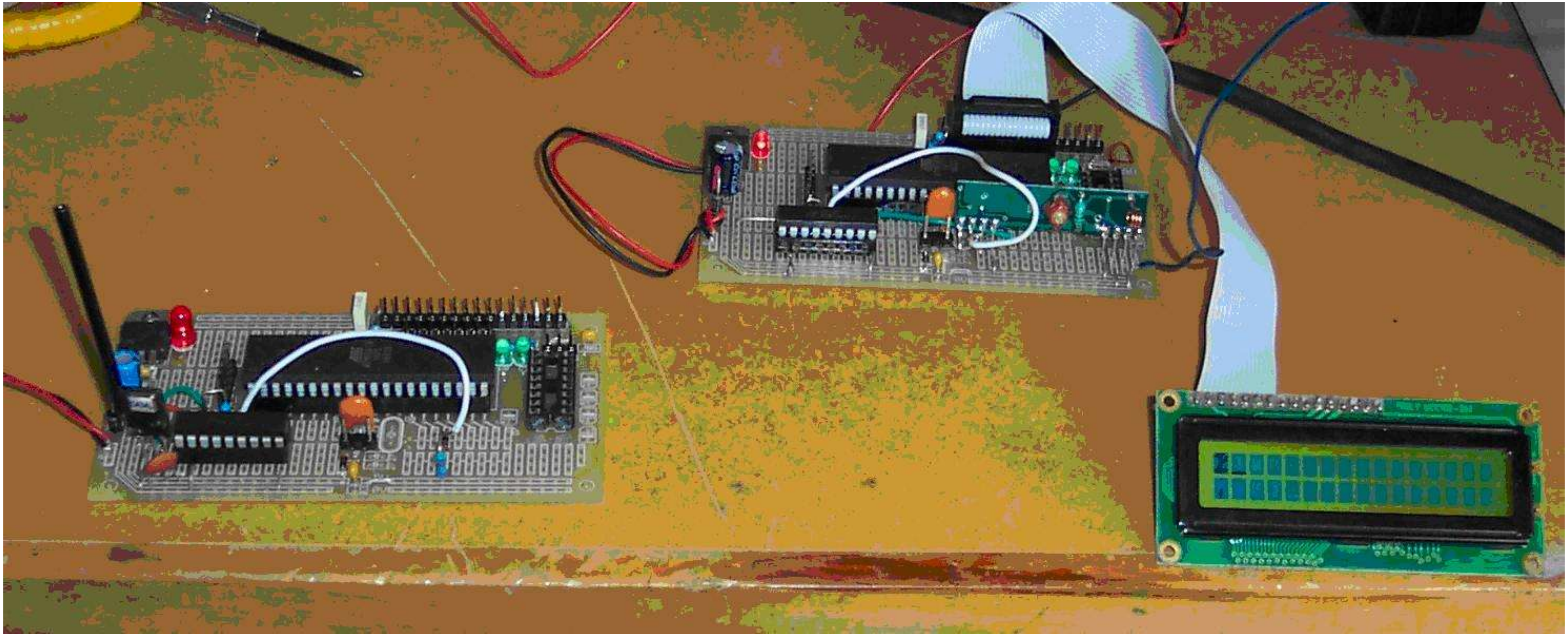


Version 1A schematic (OLD)



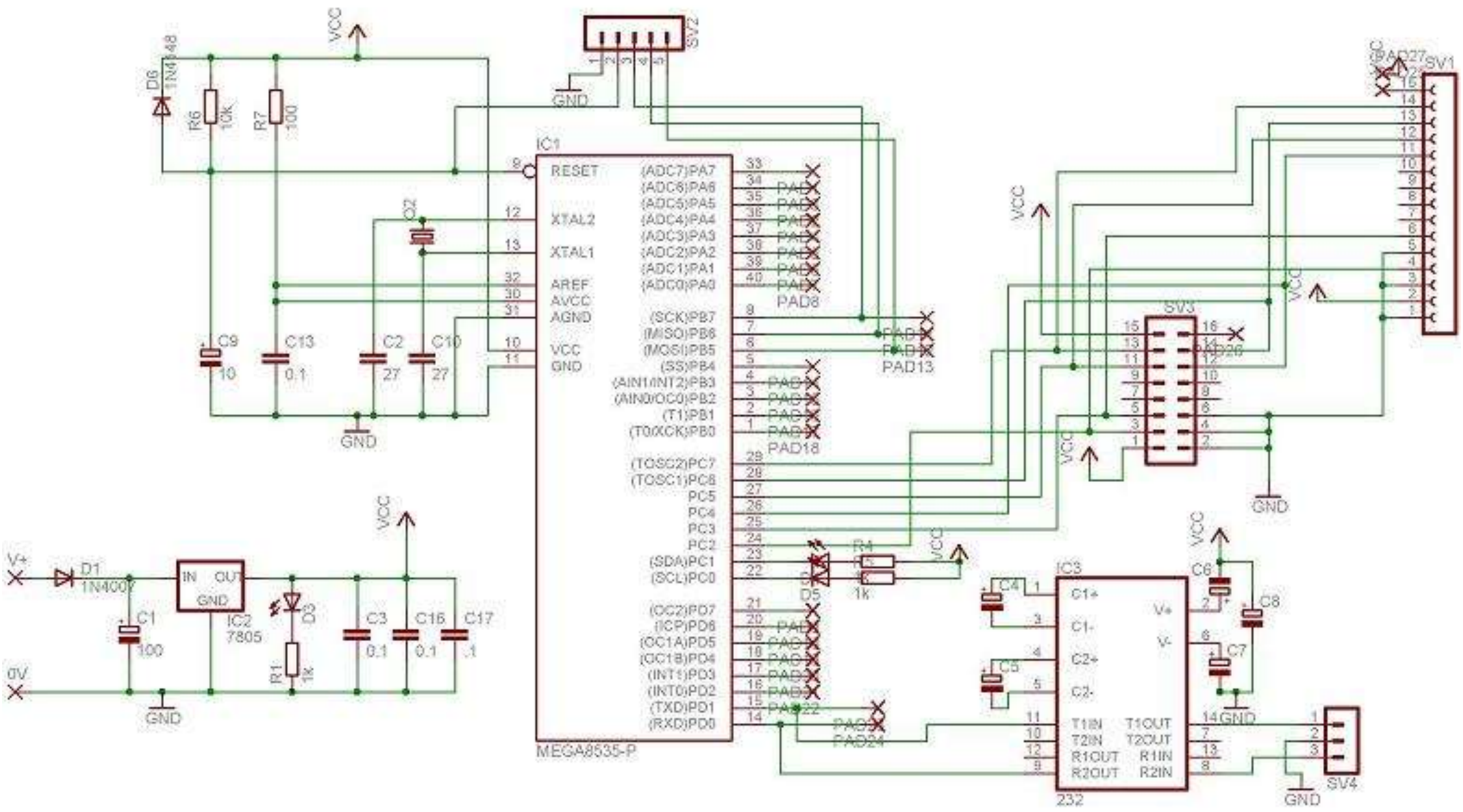
Version 1A pcb layout

63.10 AVR Development Board 2

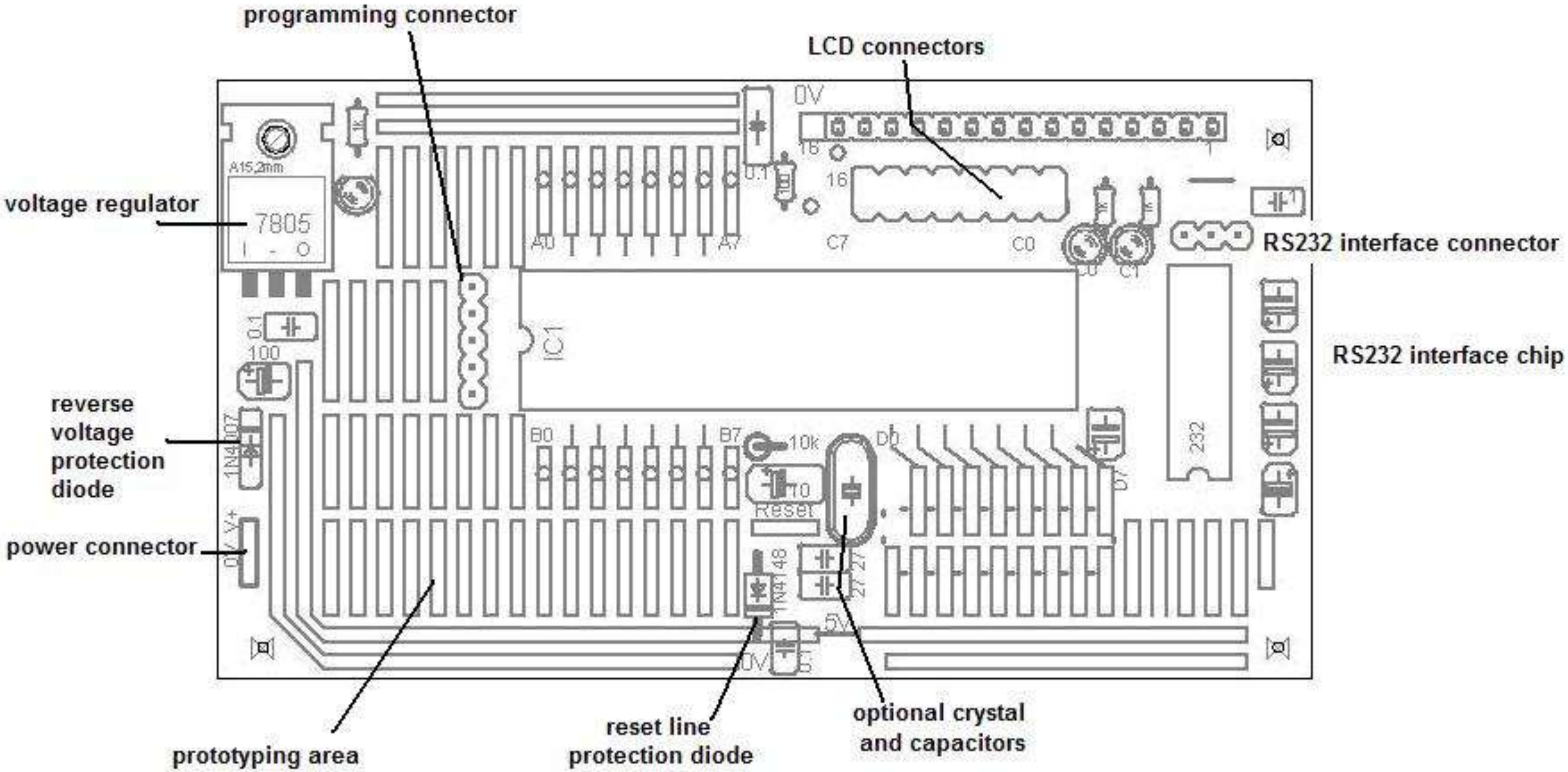


These are useful for small projects and are still available in the workshop

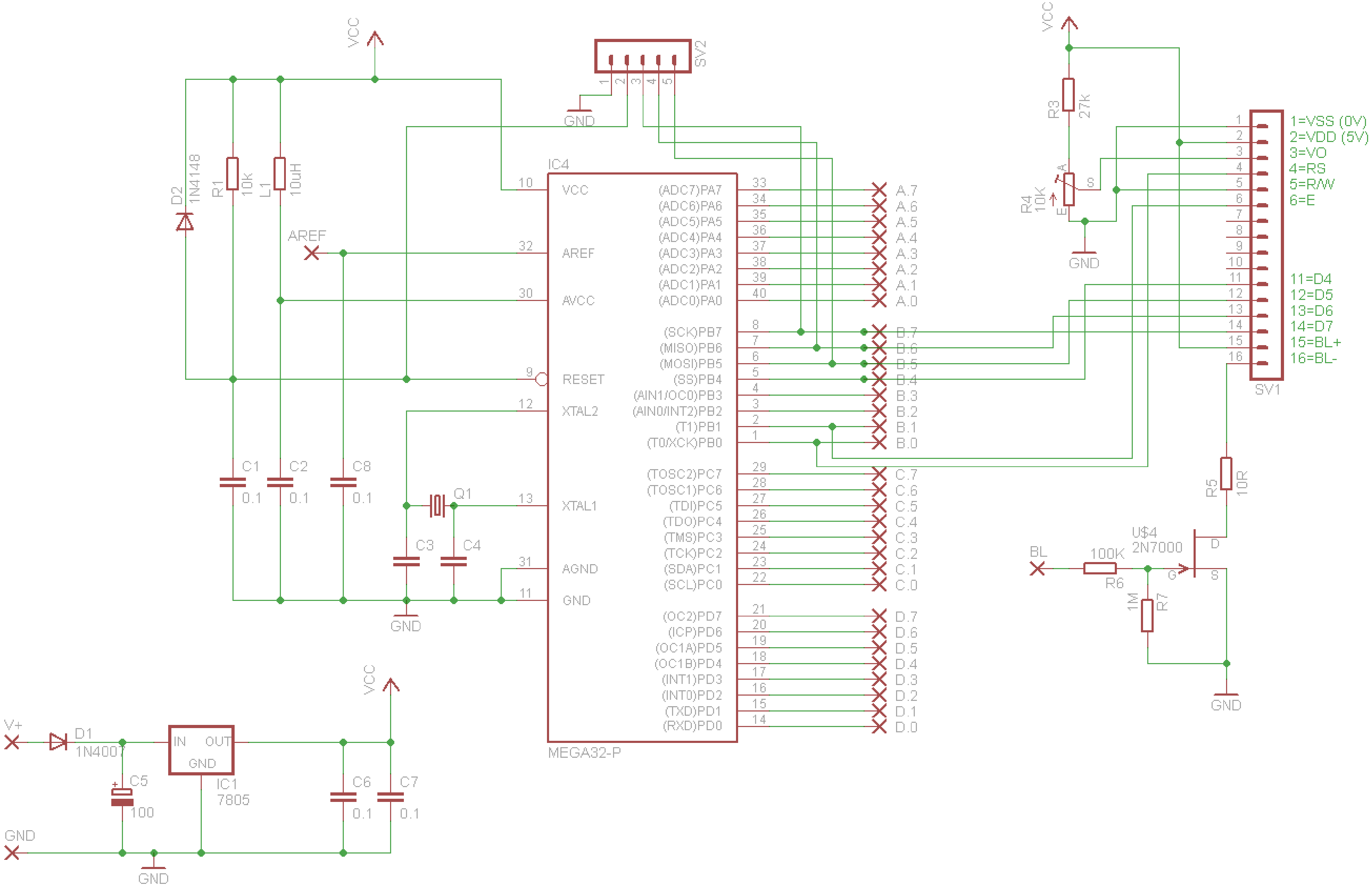
63.11 Dev board version 2 circuit diagram



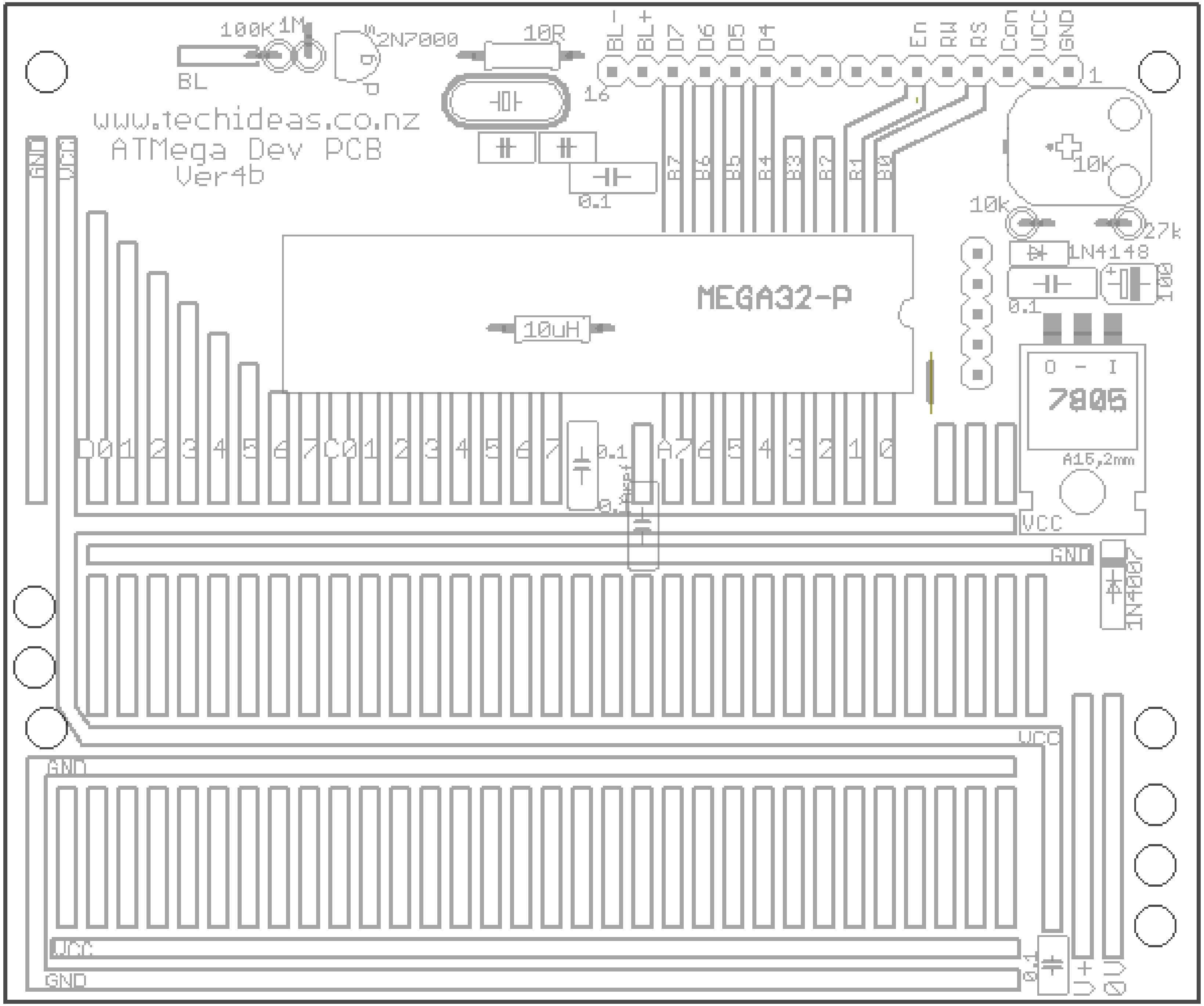
63.12 Dev board pcb layout version 2



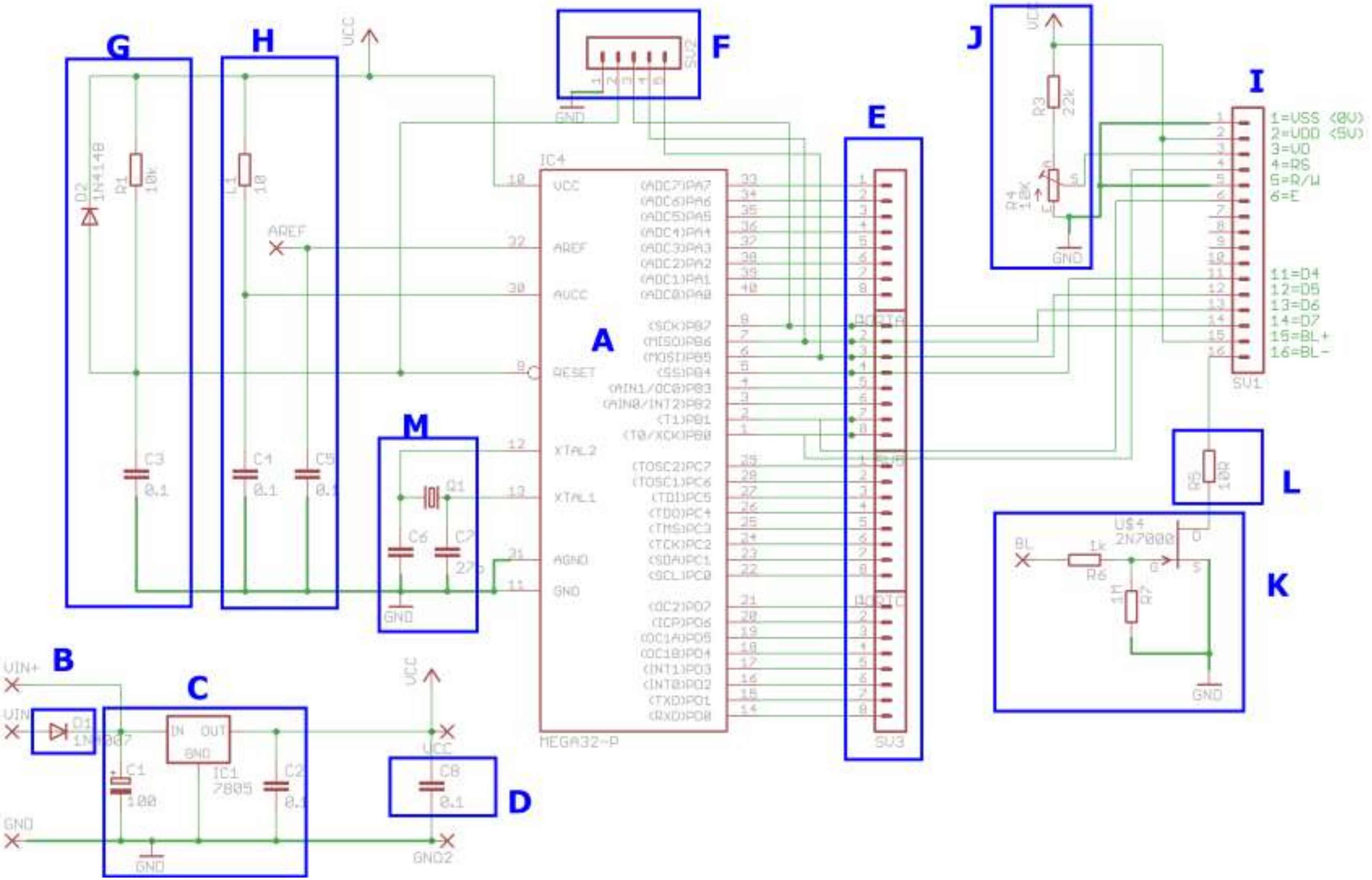
63.13 ATMEGA V4b development board circuit – 12TCE 2011



63.14 V4b devboard layout 12TCE 2011

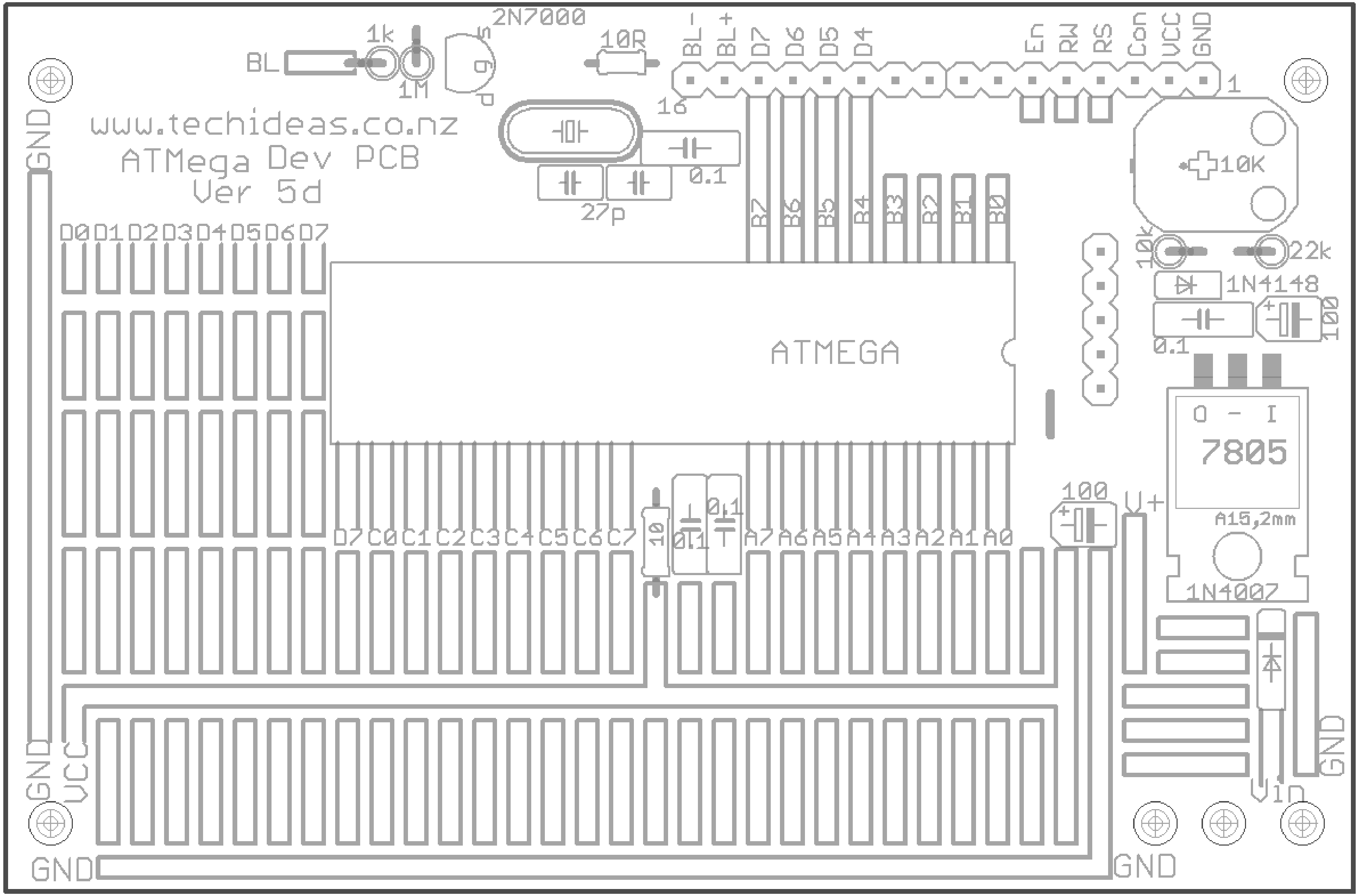


63.15 ATMega Dev PCB V5DSchematic (2012)

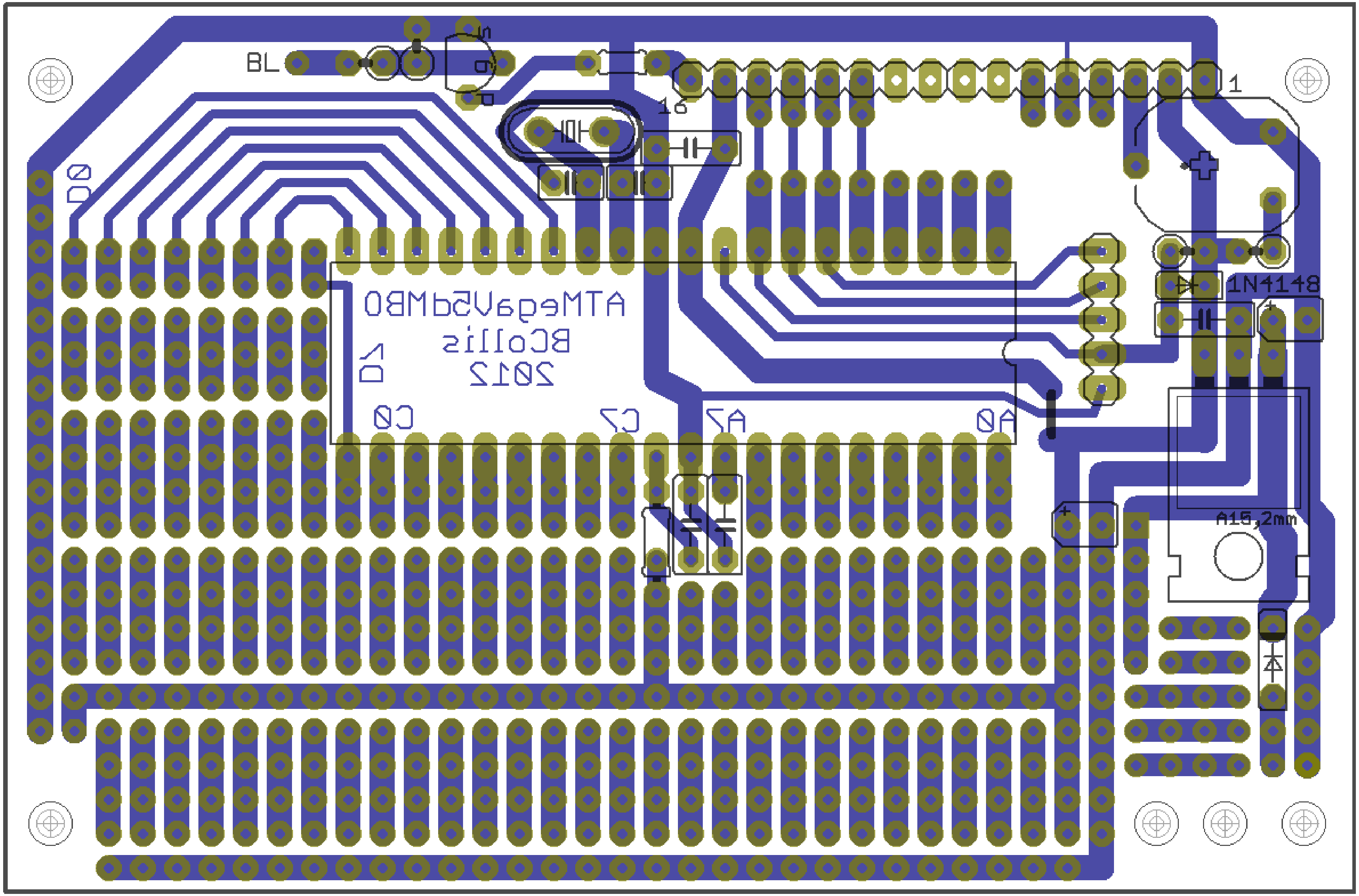


A – ATMega16
B – Reverse polarity protection diode
C – 7805 voltage regulator with input and output filte capacitors
D – 0.1uF (100nF) power supply filter capacitors around the board
E – I/O ports (32)
F – Programming Connector
G – Reset circuit
H – ADC power supply circuit
I – LCD connector
J – Contrast adjustment
K – FET backlight control circuit
L – backlight current limit resistor

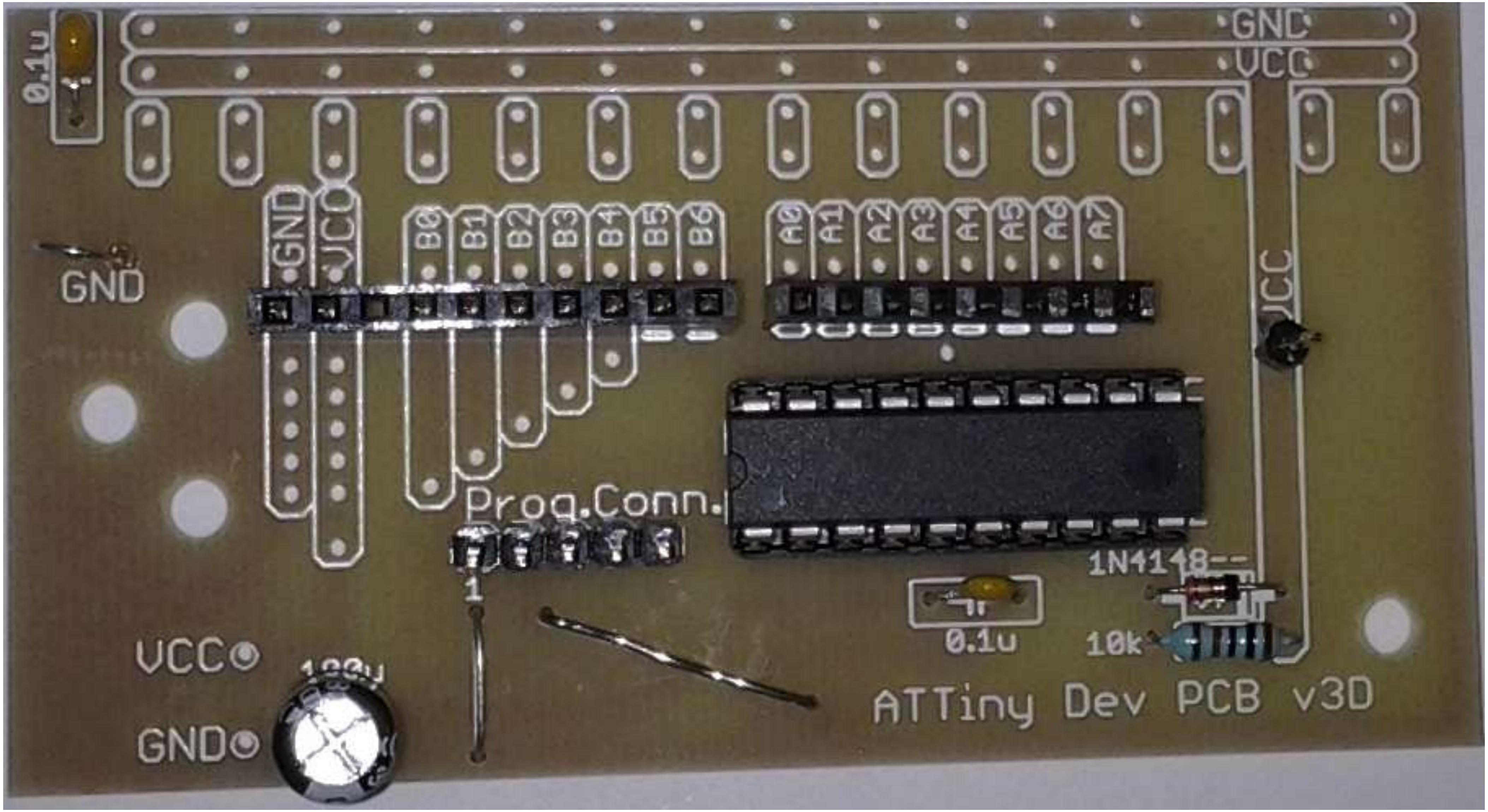
63.16 ATMega Dev PCB V5DLayout (2012)

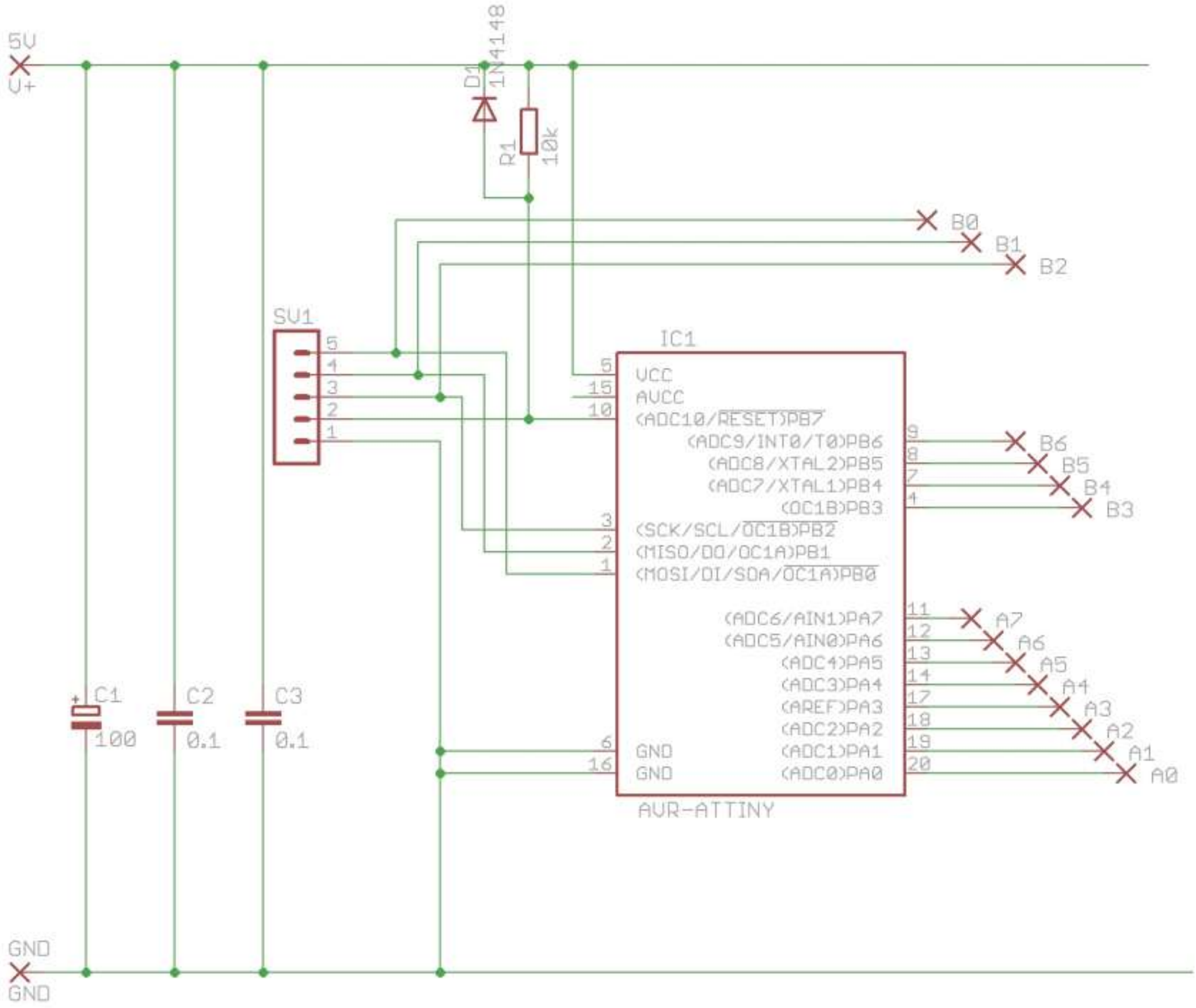


63.17 ATMega Dev PCB V5D Copper (2012)

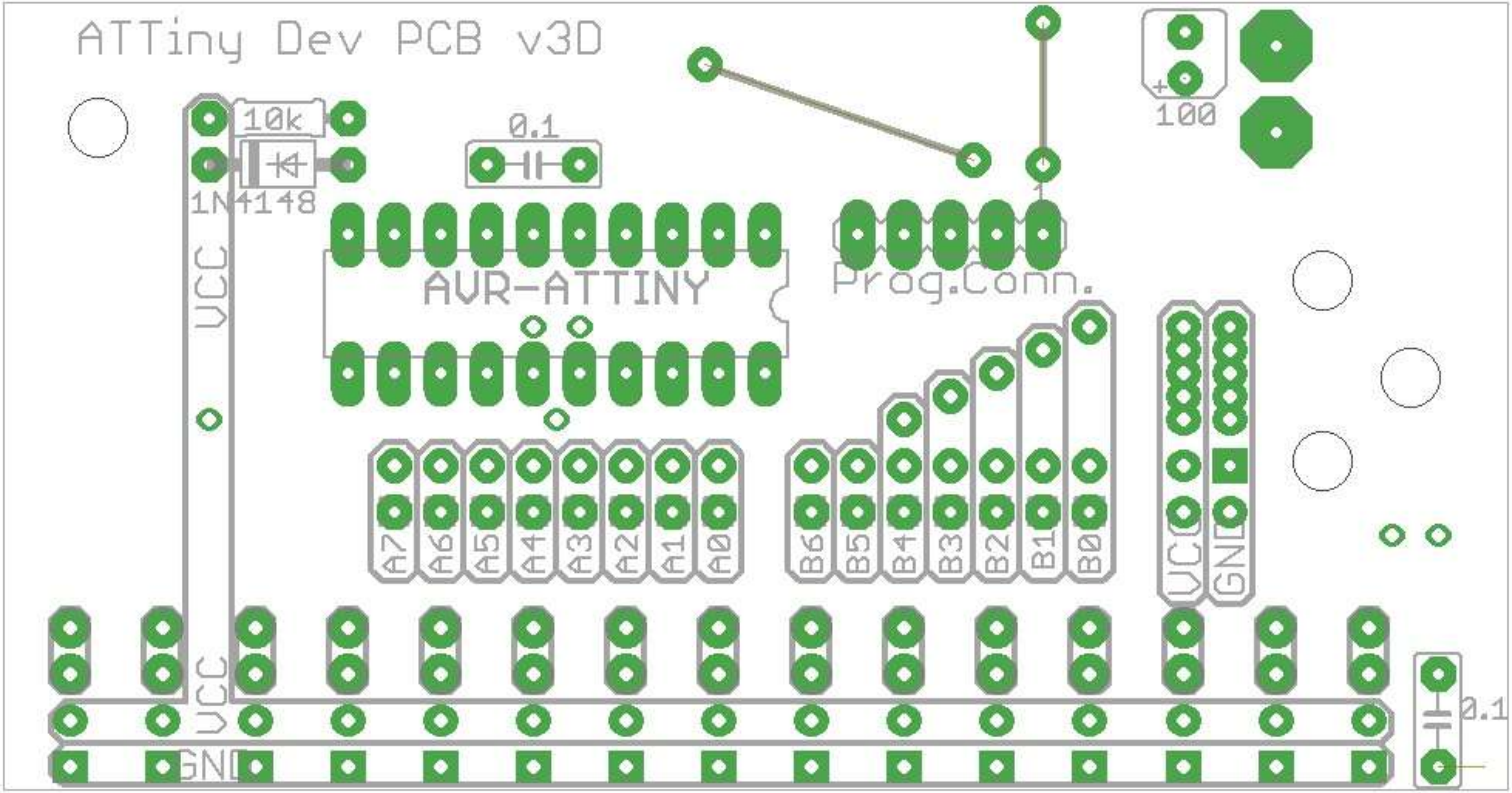


63.18 Year10 ATTiny461 V3d development board

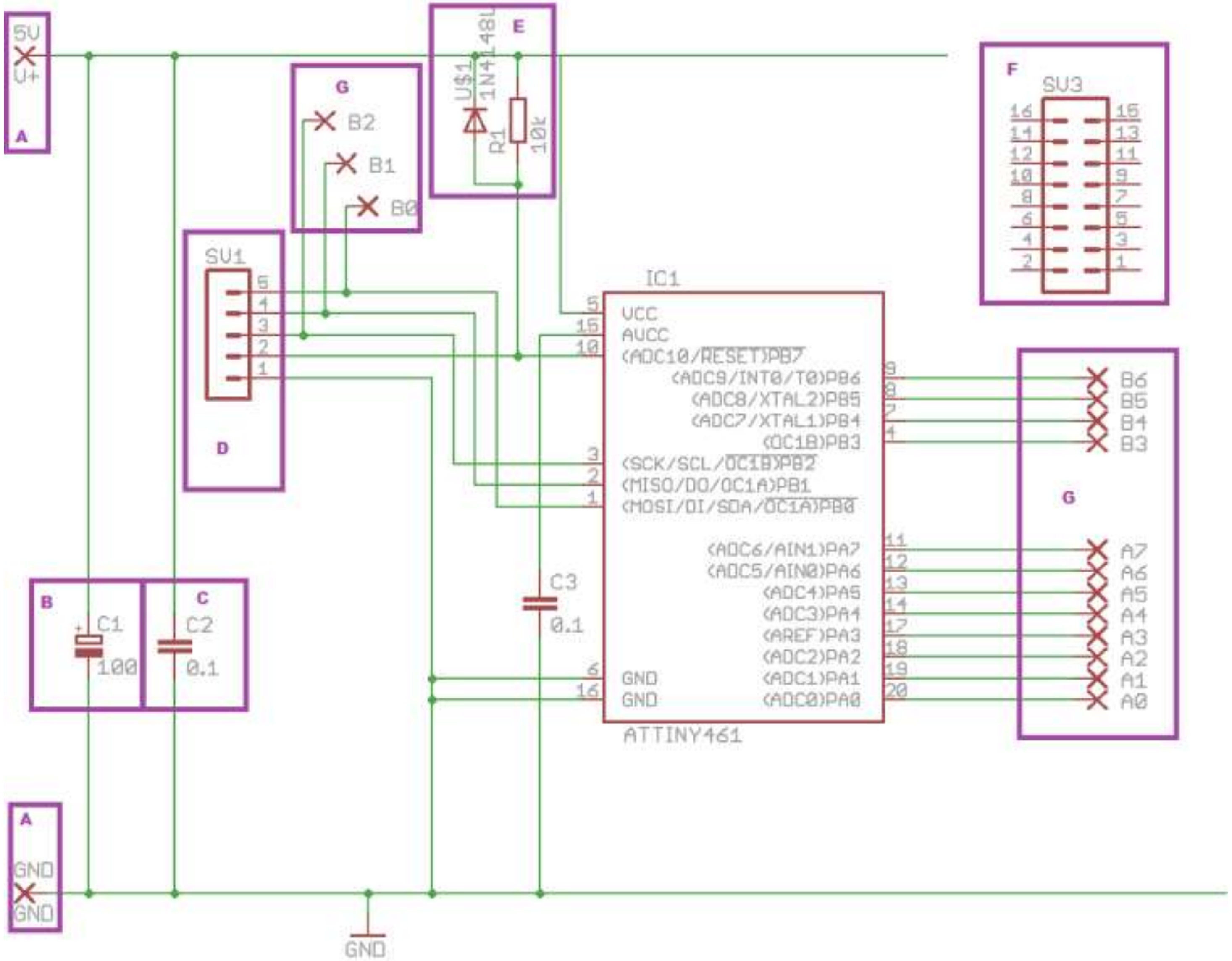




5V
V+
D1
1N4148
R1
10k
B0
B1
B2
SV1
IC1
VCC
AVCC
(ADC10/RESET)PB7
(ADC9/INT0/T0)PB6
(ADC8/XTAL2)PB5
(ADC7/XTAL1)PB4
(OC1B)PB3
(SCK/SCL/OC1B)PB2
(MISO/DO/OC1A)PB1
(MOSI/DI/SDA/OC1A)PB0
(ADC6/AIN1)PA7
(ADC5/AIN0)PA6
(ADC4)PA5
(ADC3)PA4
(AREF)PA3
(ADC2)PA2
(ADC1)PA1
(ADC0)PA0
GND
GND
B6
B5
B4
B3
A7
A6
A5
A4
A3
A2
A1
A0
C1
100
C2
0.1
C3
0.1
AVR-ATTINY
GND
GND

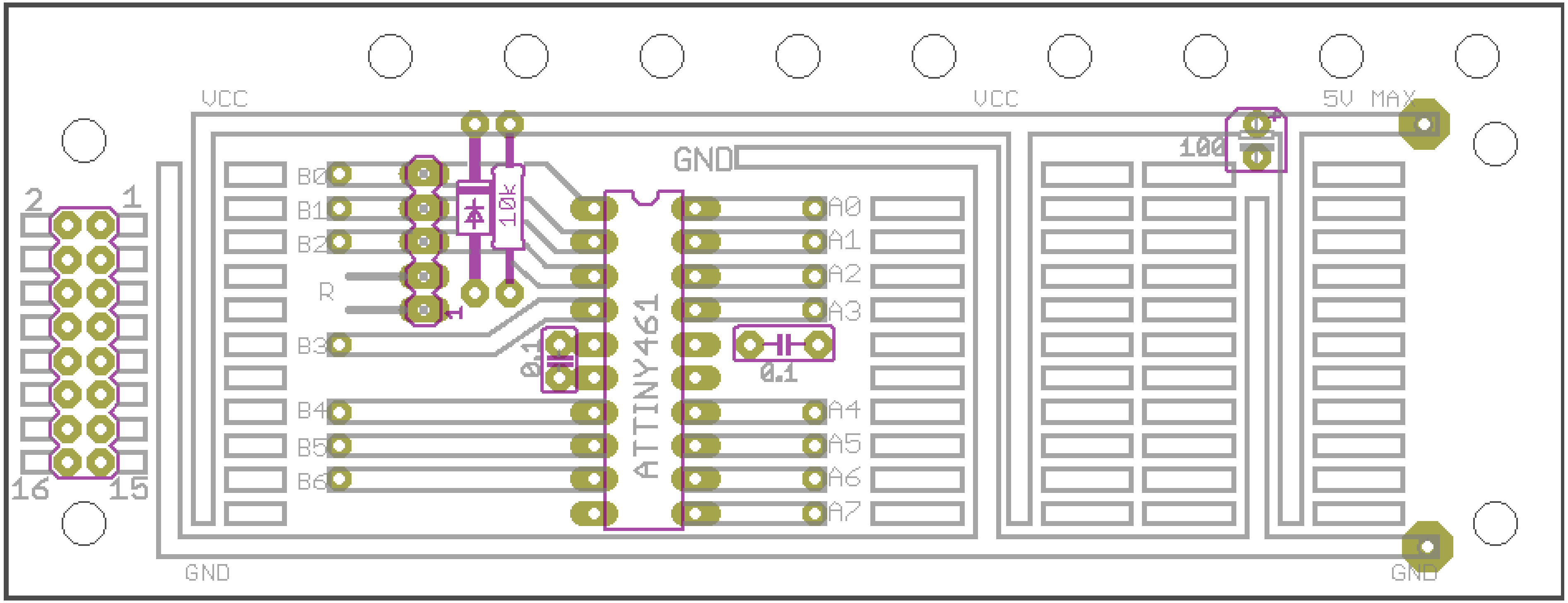

ATTiny Dev PCB v3D
10k
1N4148
0.1
100
+
AVR-ATTINY
Prog.Conn.
VCC
A7
A6
A5
A4
A3
A2
A1
A0
B6
B5
B4
B3
B2
B1
B0
VCC
GND
VCC
GND
0.1

63.19 Year11 ATTiny461 V6d development board

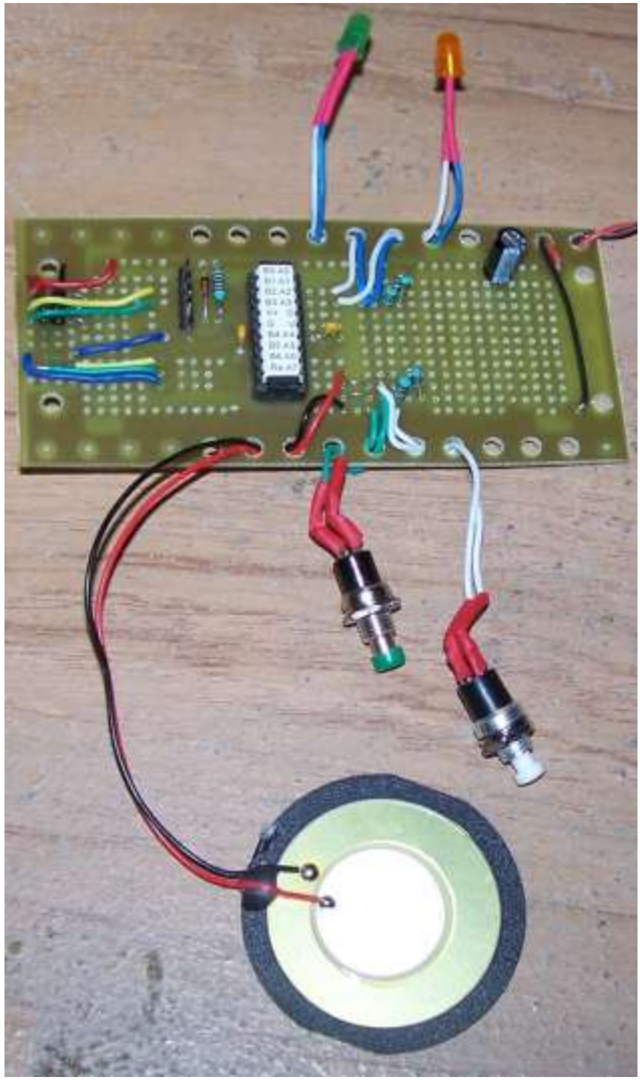


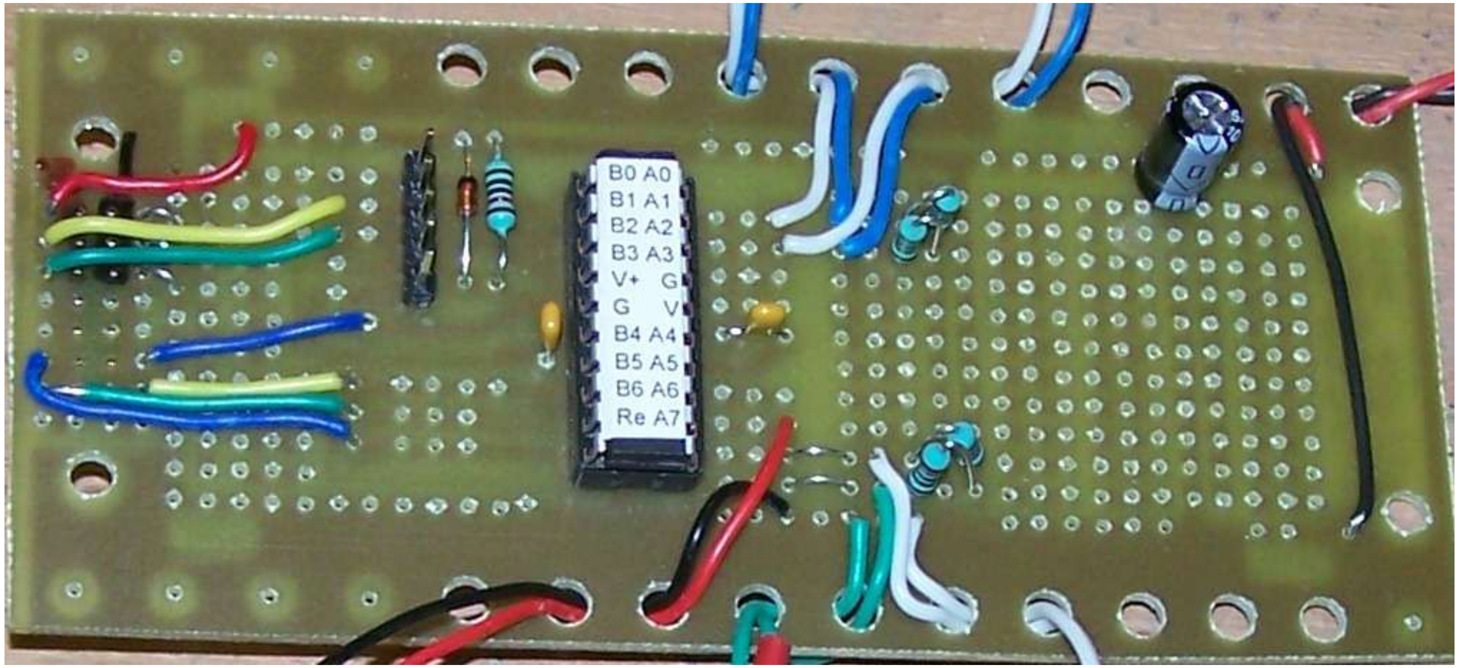
A – see both connections, 5V and GNd (ground or negative), make sure the maximum voltage is 5V.
B – the 100uF capacitor is to absorb any variations in voltage from the power supply.
C – the 0.1uF capacitor is to absorb and fast variations in the voltage caused by the switching of the very rapid microcontroller
D – the programming connector
E – the reset circuit: the 10k resistor connects the reset line to 5V allowing the microcontroller to run programs, if left unconnected it might drift, if shorted to gnd then the micro will stop.
The diode is a protection diode to stop any voltage higher then 5V on the reset line damaging the microcontroller (all the other pins have diodes built in)
F the connector for the LCD, no wiring is shown
G – the I/O connections – note that B0, B1 & B2 are shared with the programming connector which somewhat limits what can be connected to them

63.20 ATTiny461 V6d development board layouts

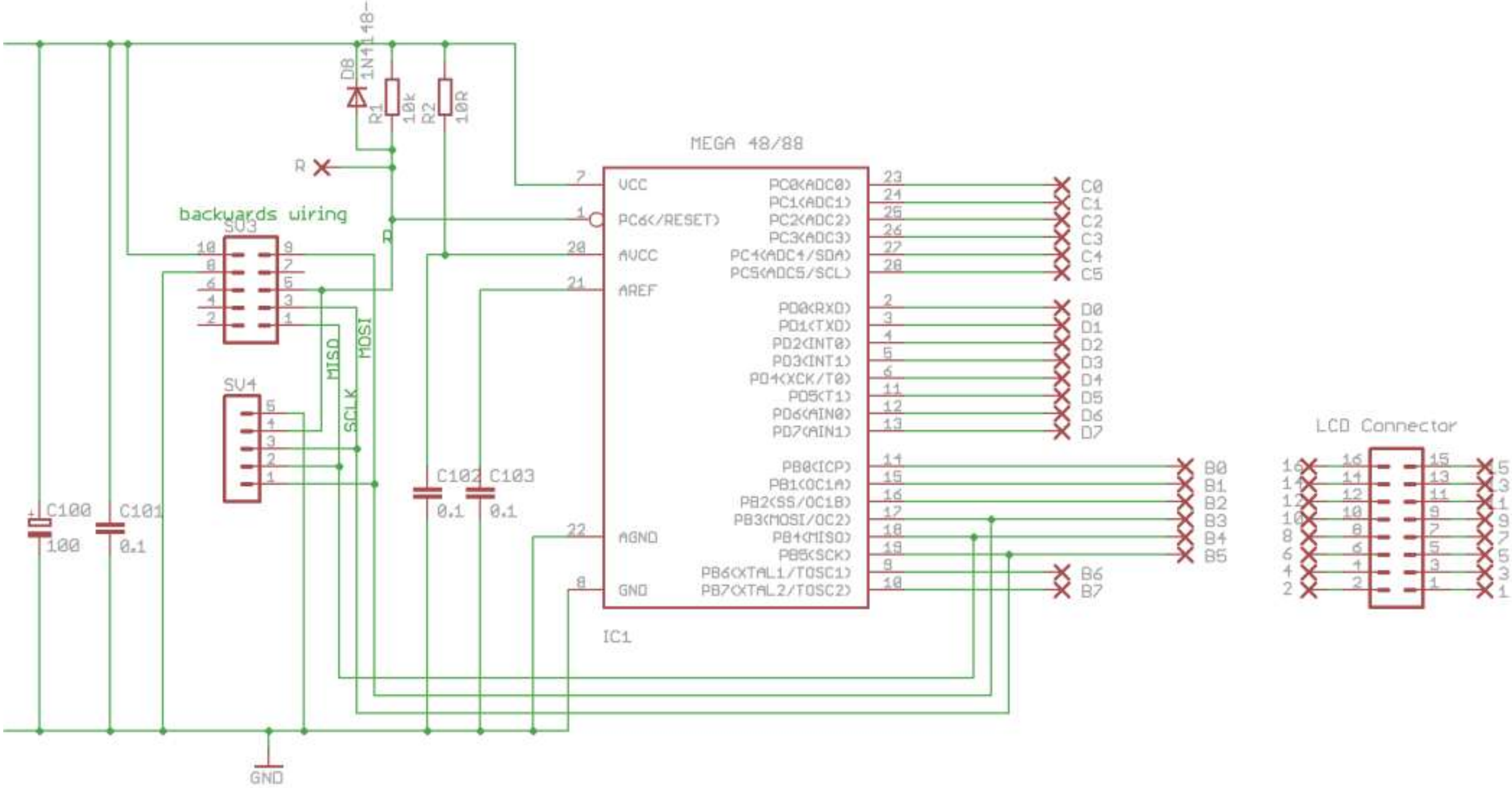


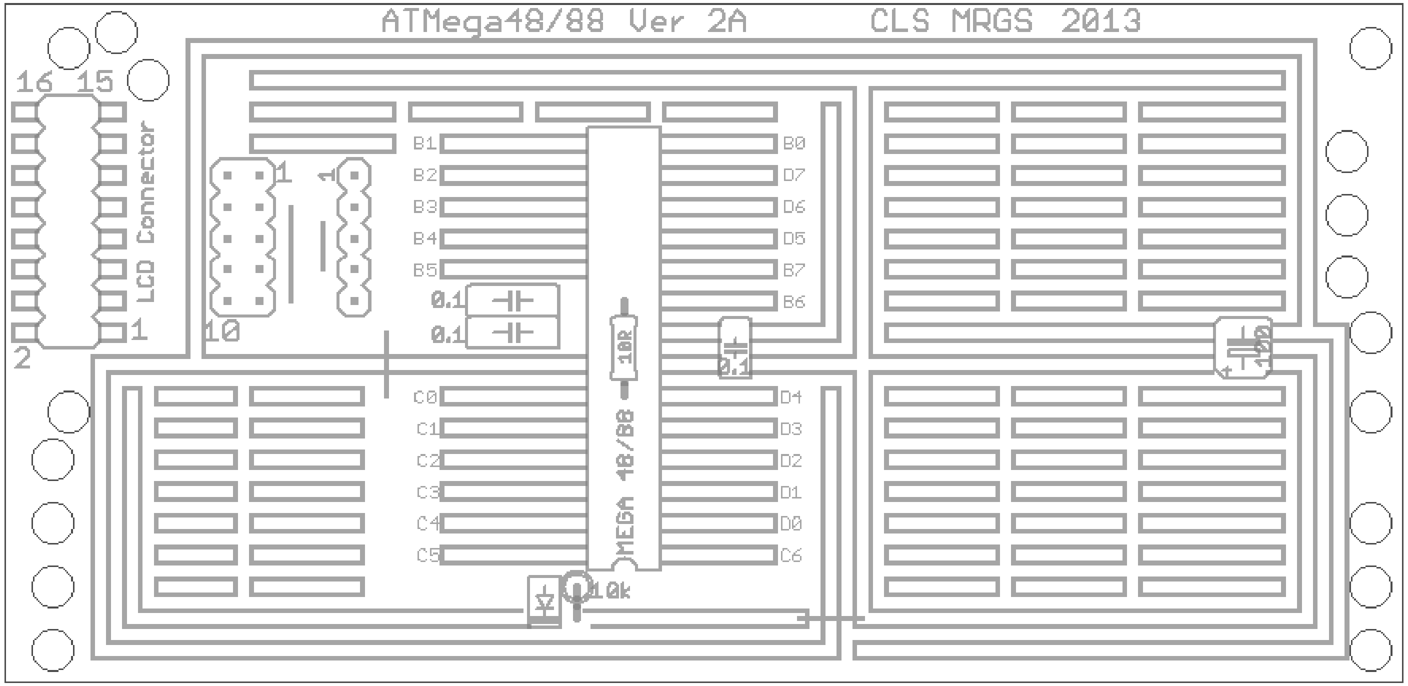
63.21 ATTiny461 V6b development board images

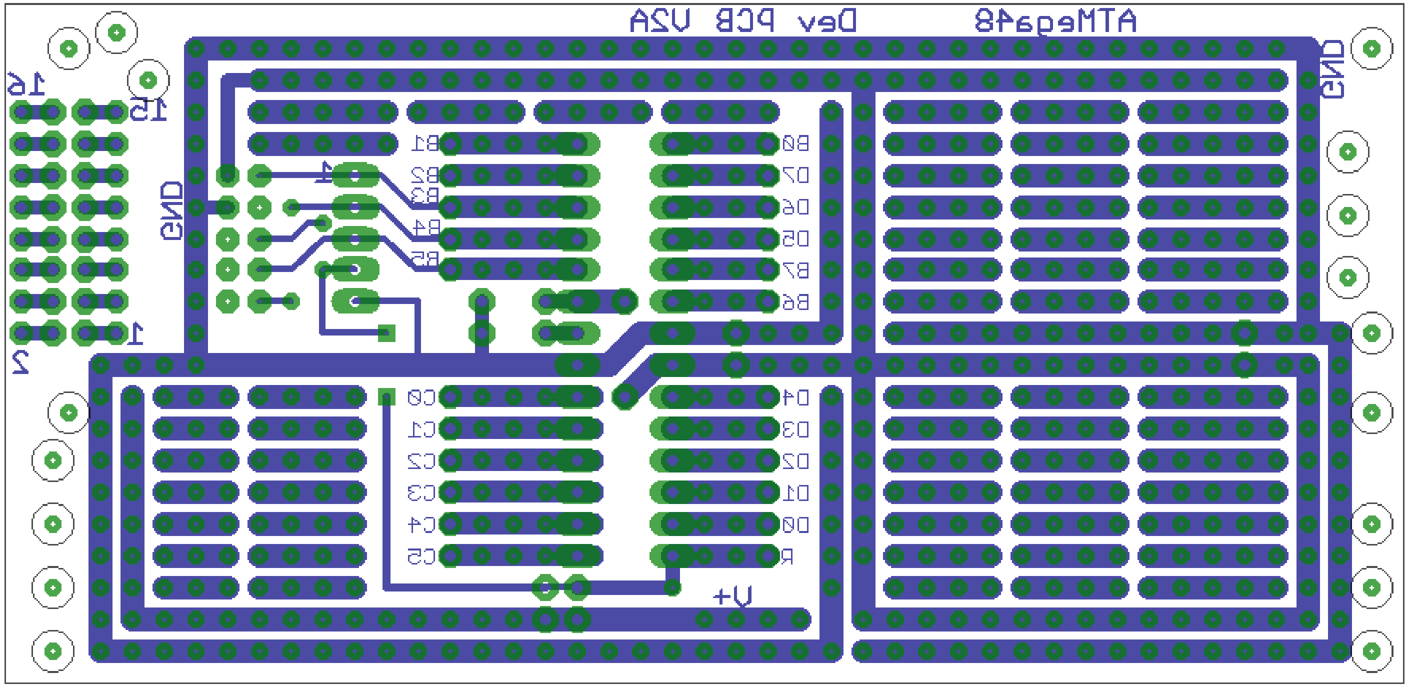




63.22 ATMega 48 Dev PCB 2A



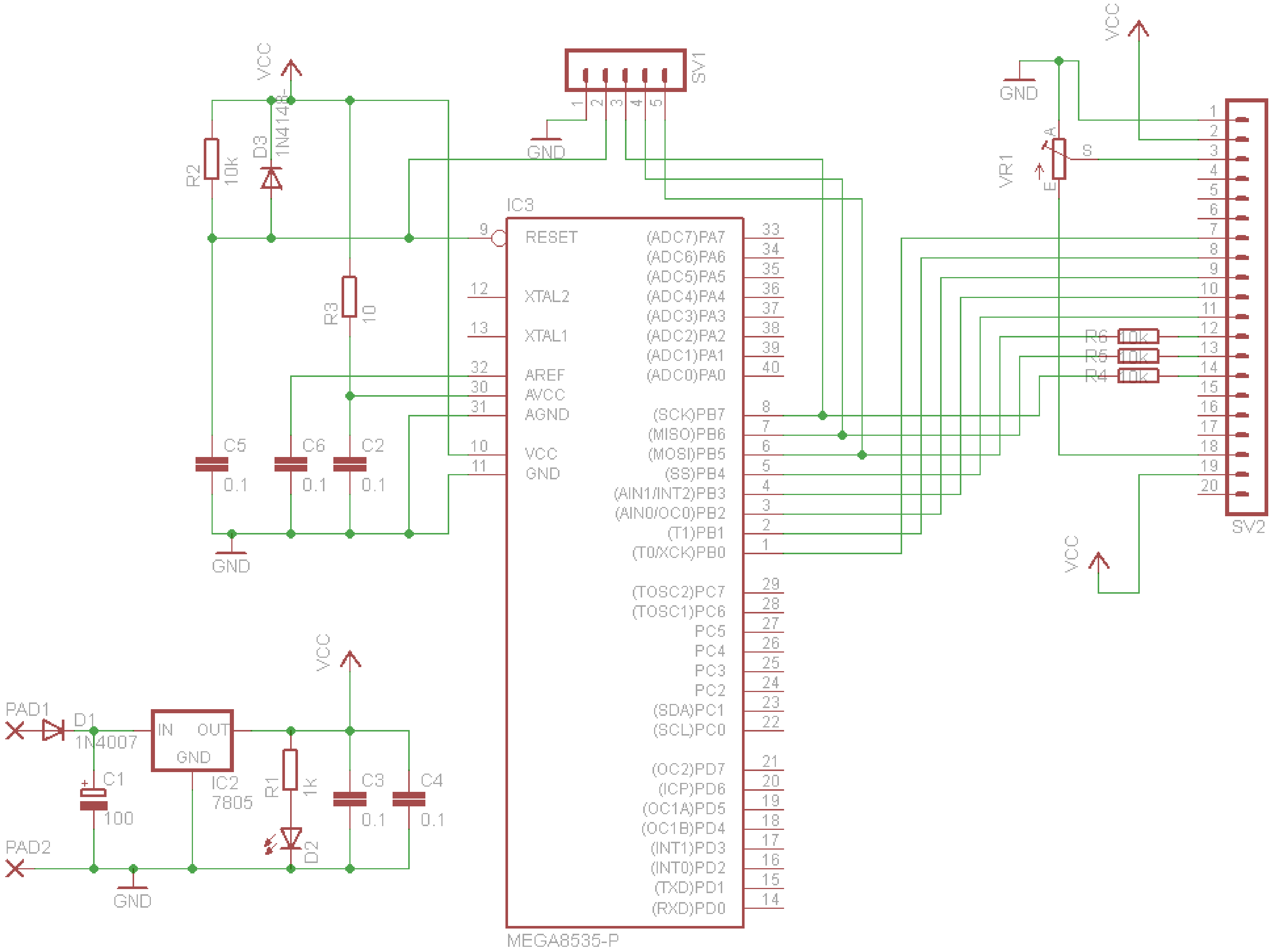

ATMega48/88 Ver 2A
CLS MRGS 2013
16 15
LCD Connector
1
2
10
B1
B2
B3
B4
B5
B0
D7
D6
D5
B7
B6
0.1
0.1
10R
MEGA 48/88
C0
C1
C2
C3
C4
C5
D4
D3
D2
D1
D0
C6
10k



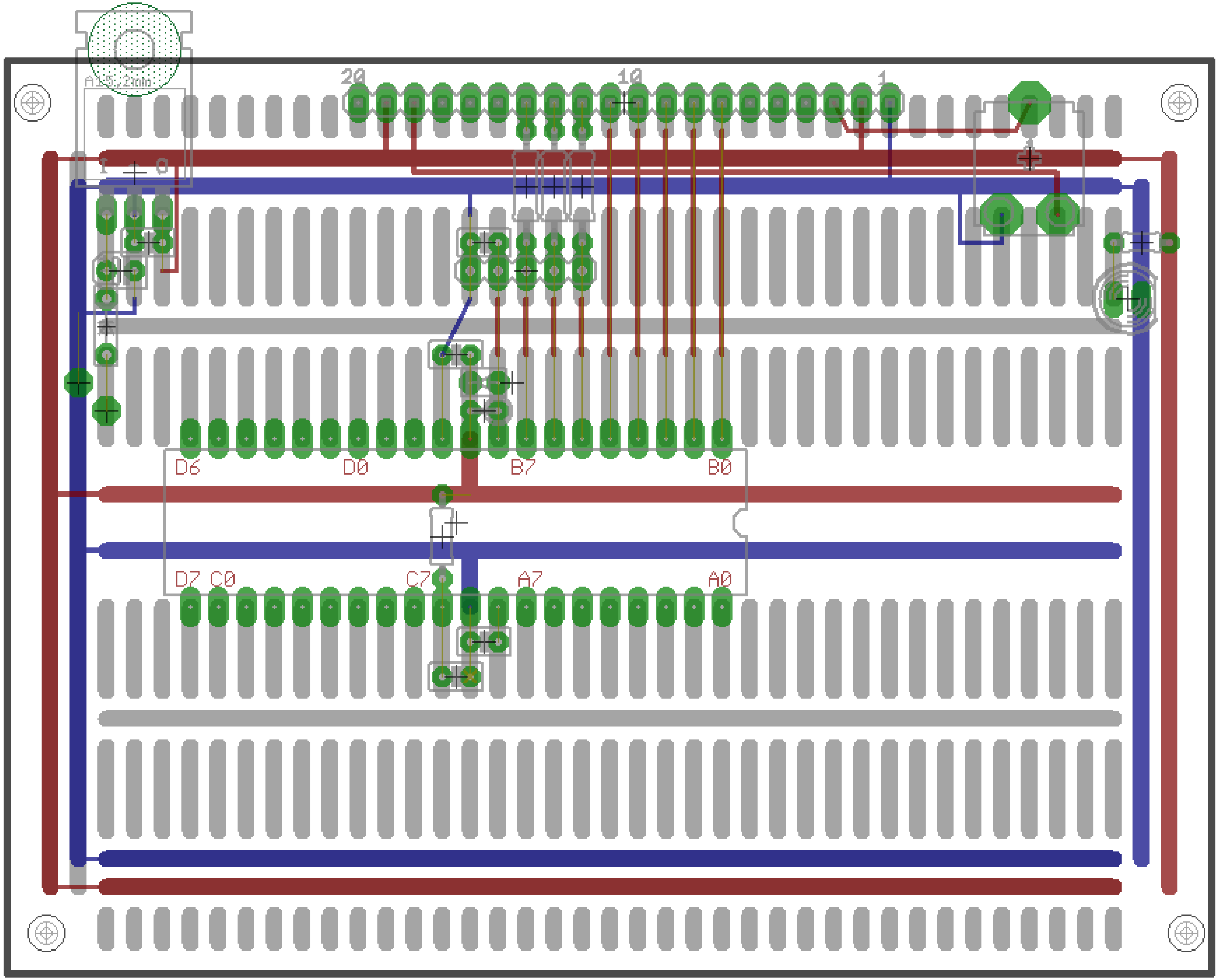
63.23 ATMEGA Protoyping board

This board is available from Sure Electronics

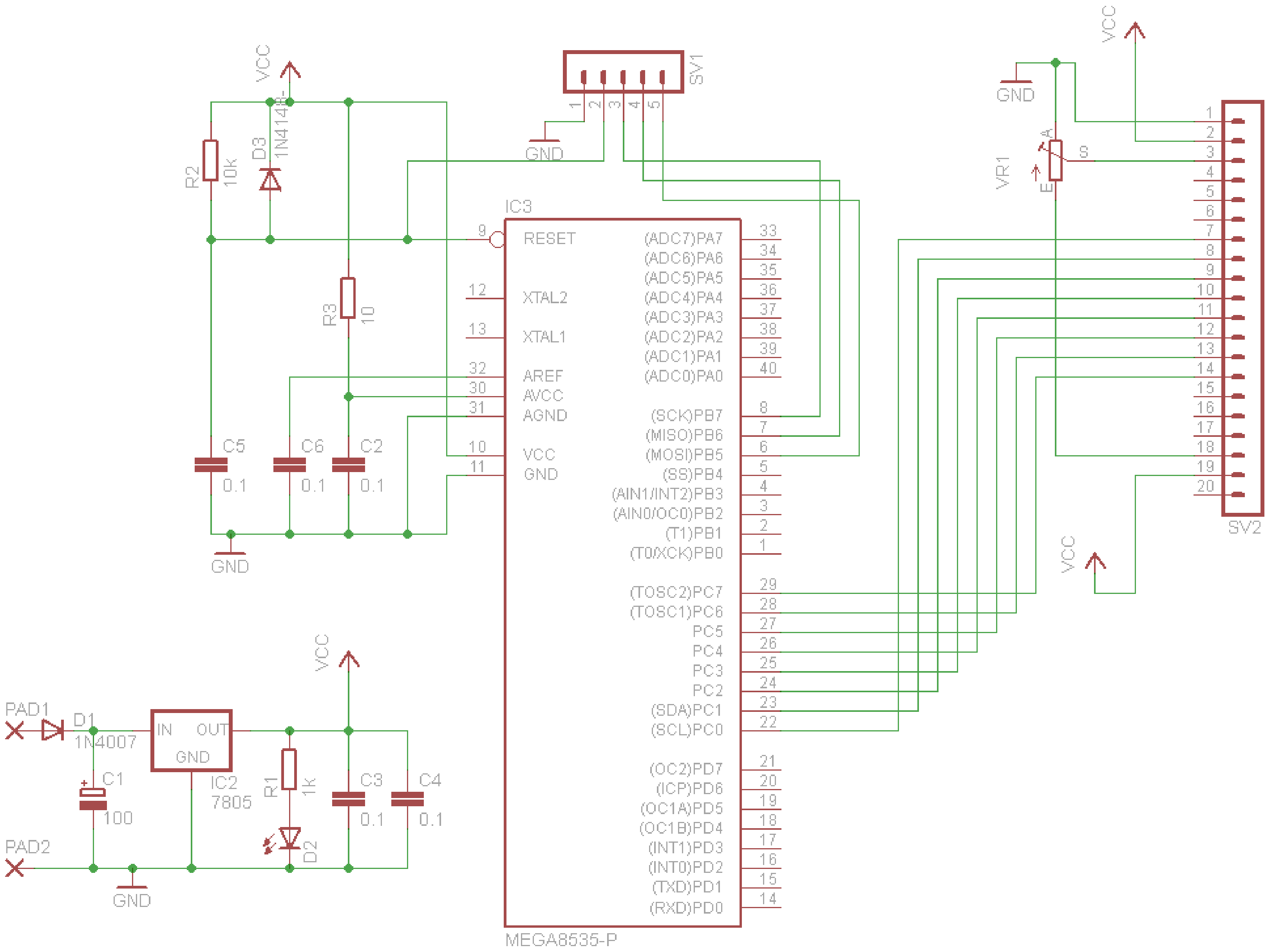
63.24 128x64 GLCD Schematic – VerC -data on portB



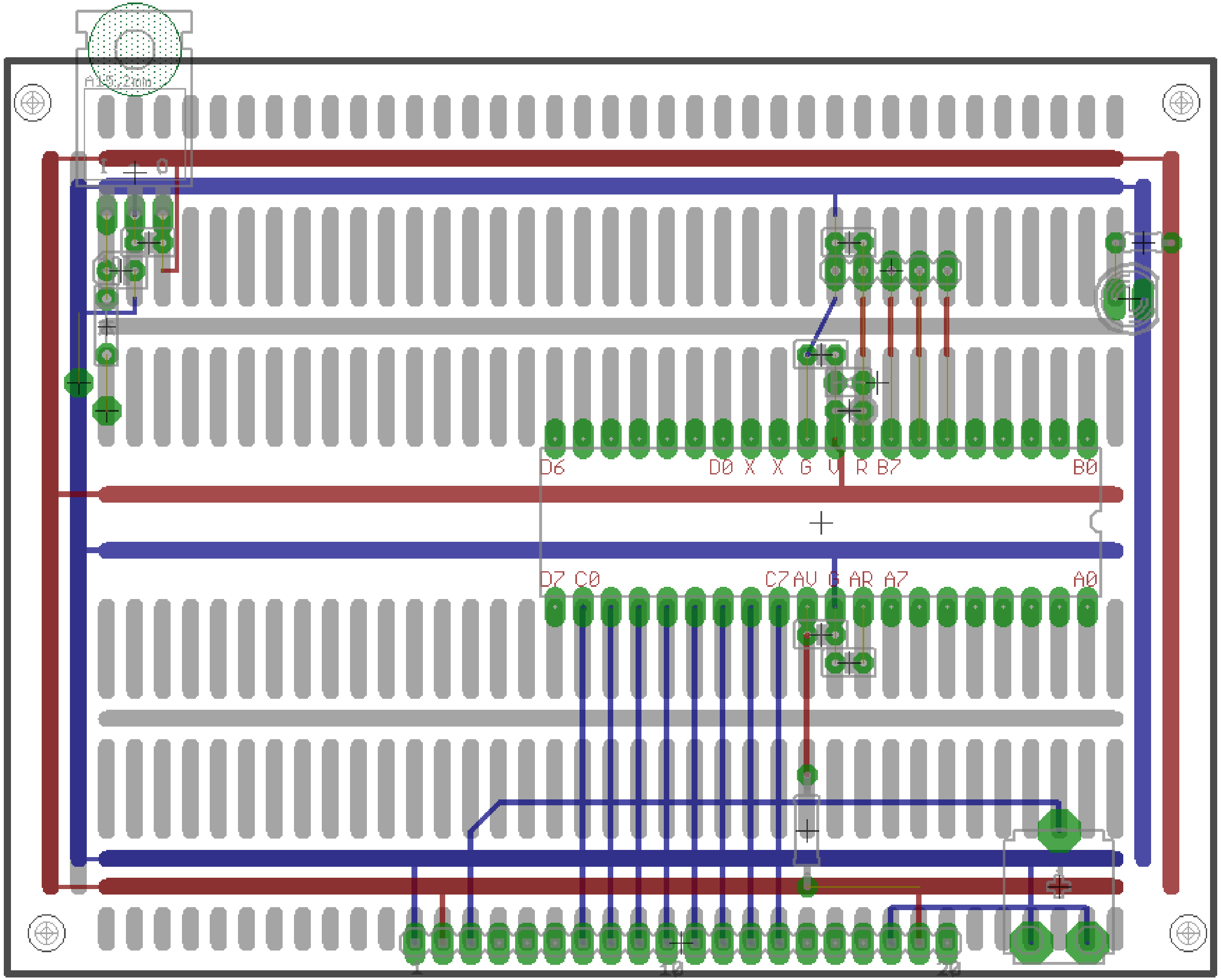
63.25 128x64 GLCD Layout – VerC –data on portB



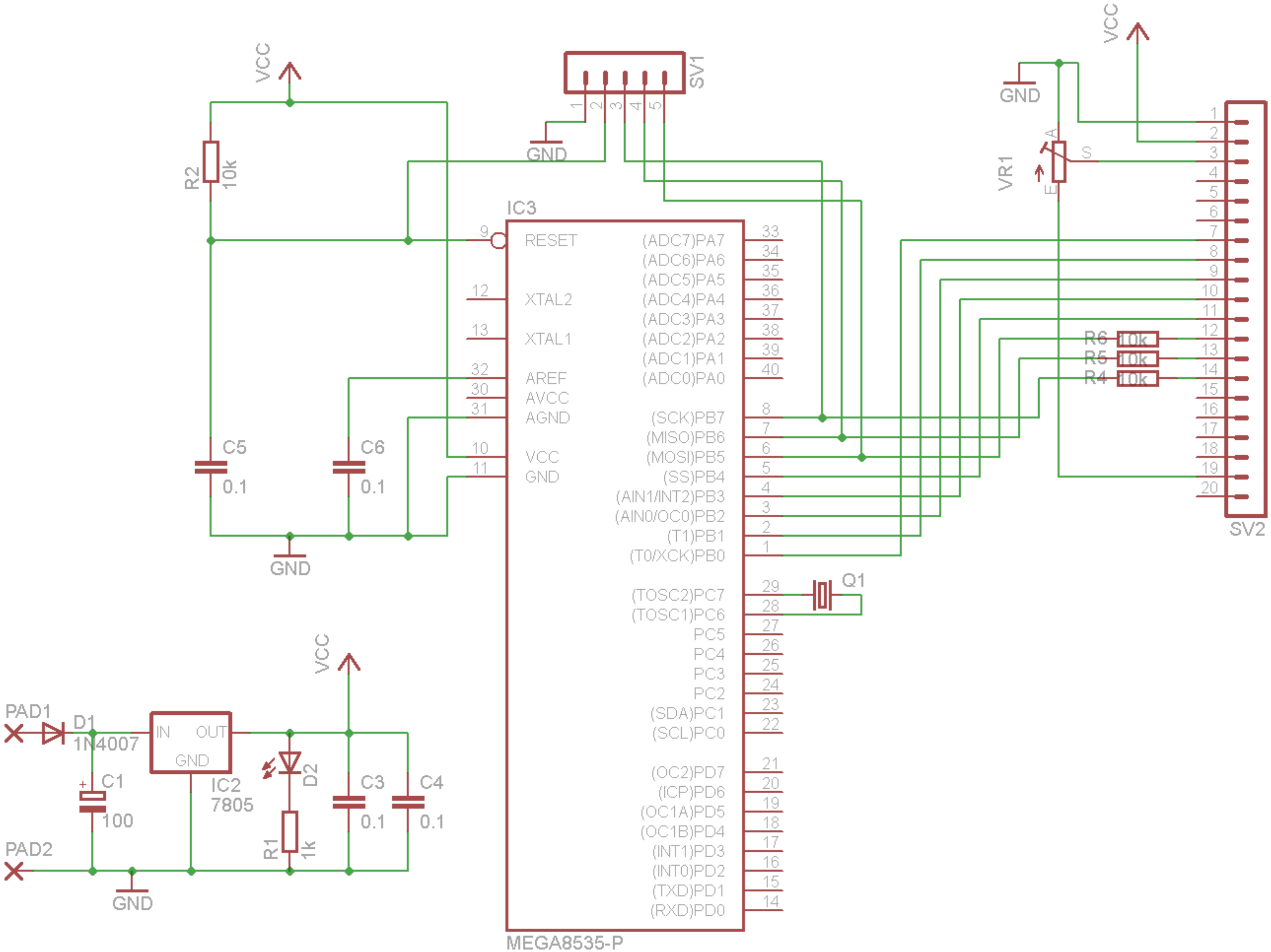
63.26 128x64 GLCD Schematic – VerD -data on portB



63.27 128x64 GLCD Layout –VerD -data on portB

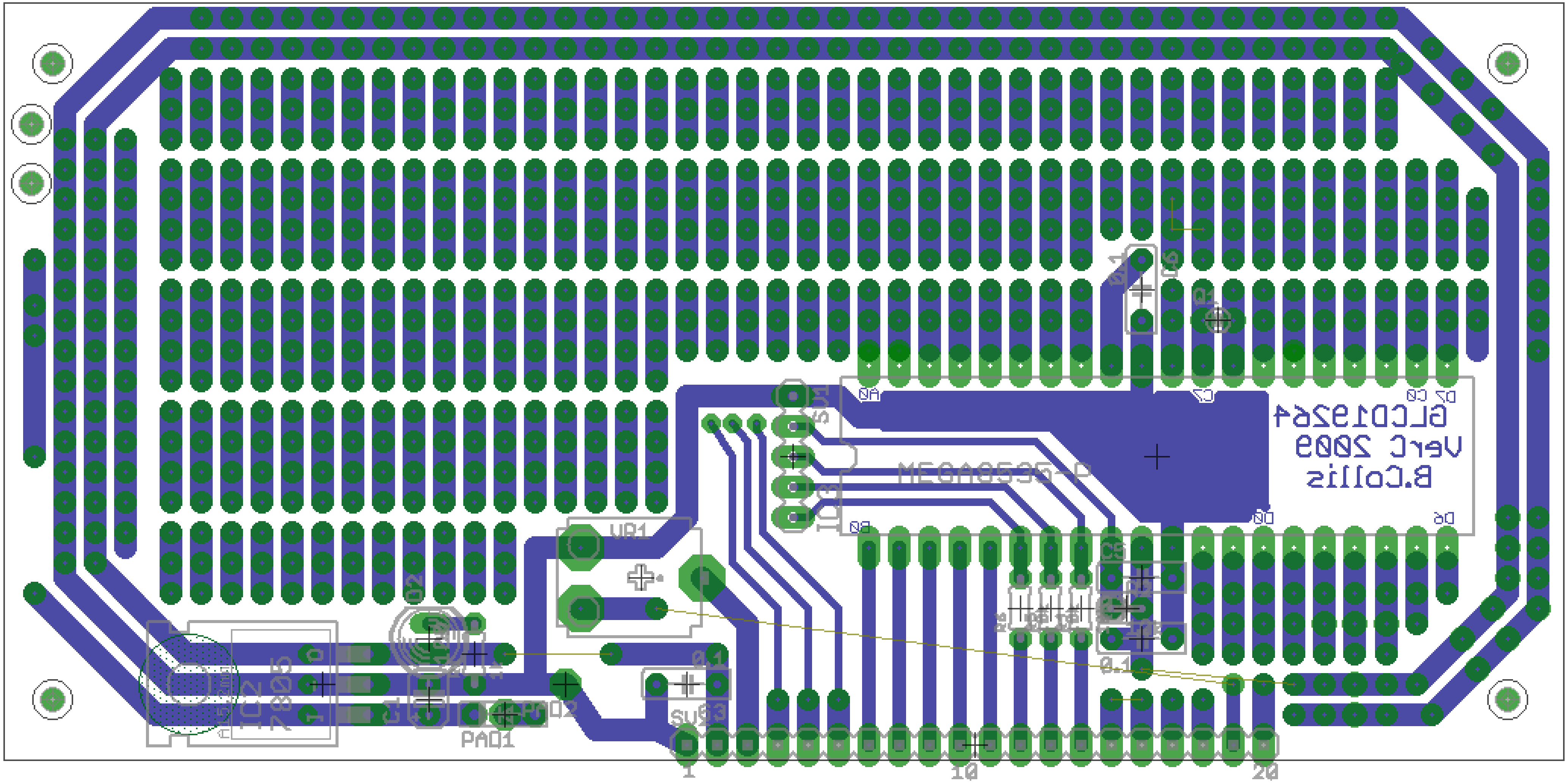


63.28 GLCD 192x64 schematic



The 7 data lines for the GLCD are not shown connected on this, they were added to the board later as per the software setup in the Temperature Tarcker project earlier in the book.

63.29 GLCD 192x64 layout

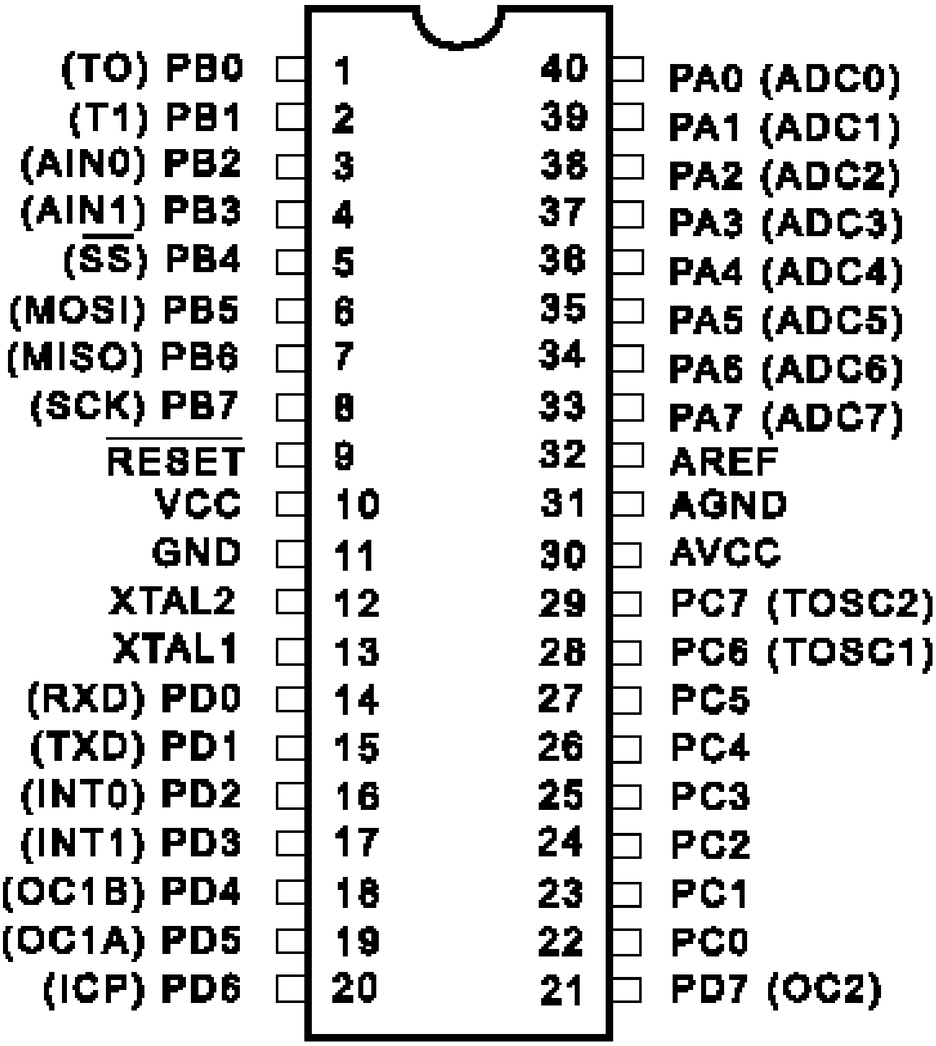


63.30 ATMEGA microcontroller pin connections

Fill out this form for your development board as you use it

Port Pin	Second Function	Direction	Connected to	To control/sense
A.0	ADC 0	I / O		
A.1	ADC 1	I / O		
A.2	ADC 2	I / O		
A.3	ADC 3	I / O		
A.4	ADC 4	I / O		
A.5	ADC 5	I / O		
A.6	ADC 6	I / O		
A.7	ADC 7	I / O		
B.0	Timer0	Input		
B.1	Timer1	Input		
B.2		I / O		
B.3		I / O		
B.4		I / O		
B.5	MOSI-Prog	I / O		
B.6	MISO-Prog	I / O		
B.7	SCK-Prog	I / O		
C.0		I / O		
C.1		I / O		
C.2		I / O		
C.3		I / O		
C.4		I / O		
C.5		I / O		
C.6	32768 xtal	I / O		
C.7	32768 xtal	I / O		
D.0	Serial rx	I / O		
D.1	Serial tx	I / O		
D.2	Interrupt0	I / O		
D.3	Interrupt1	I / O		
D4	T1 out	I / O		
D.5		I / O		
D.6	ICP	I / O		
D.7		I / O		

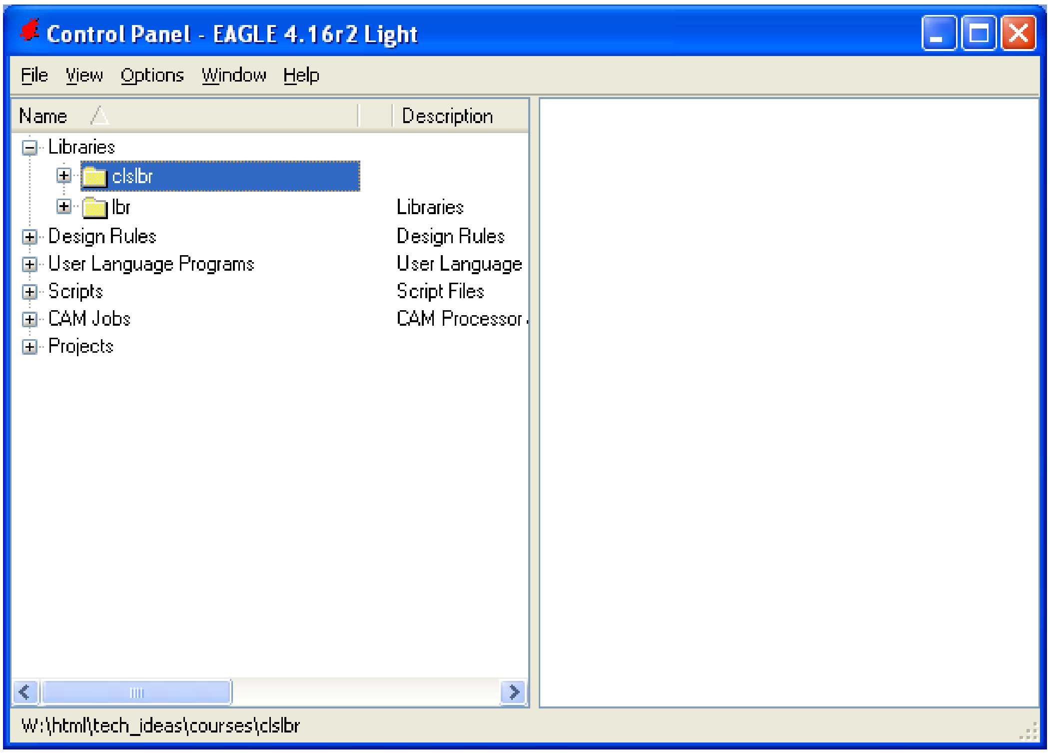
63.31 ATMEGA16/644 40pin DIP package– pin connections



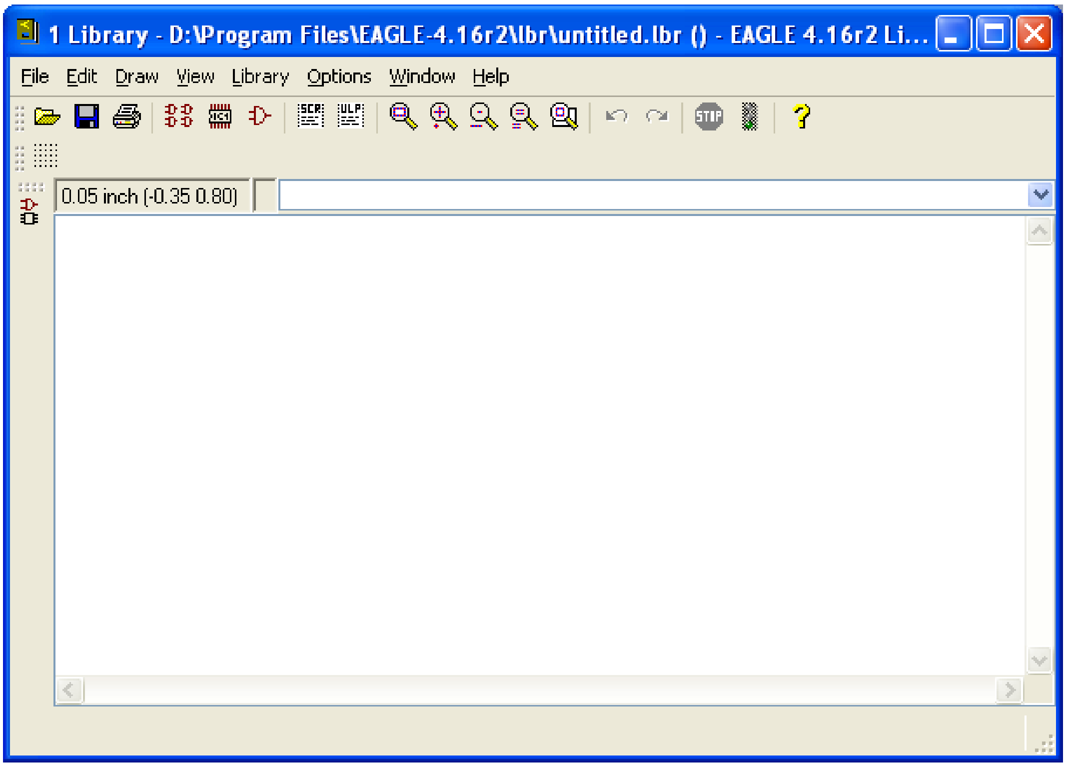
64 Eagle - creating your own library

This requires copying components from an existing library into a new library and then modifiying them.

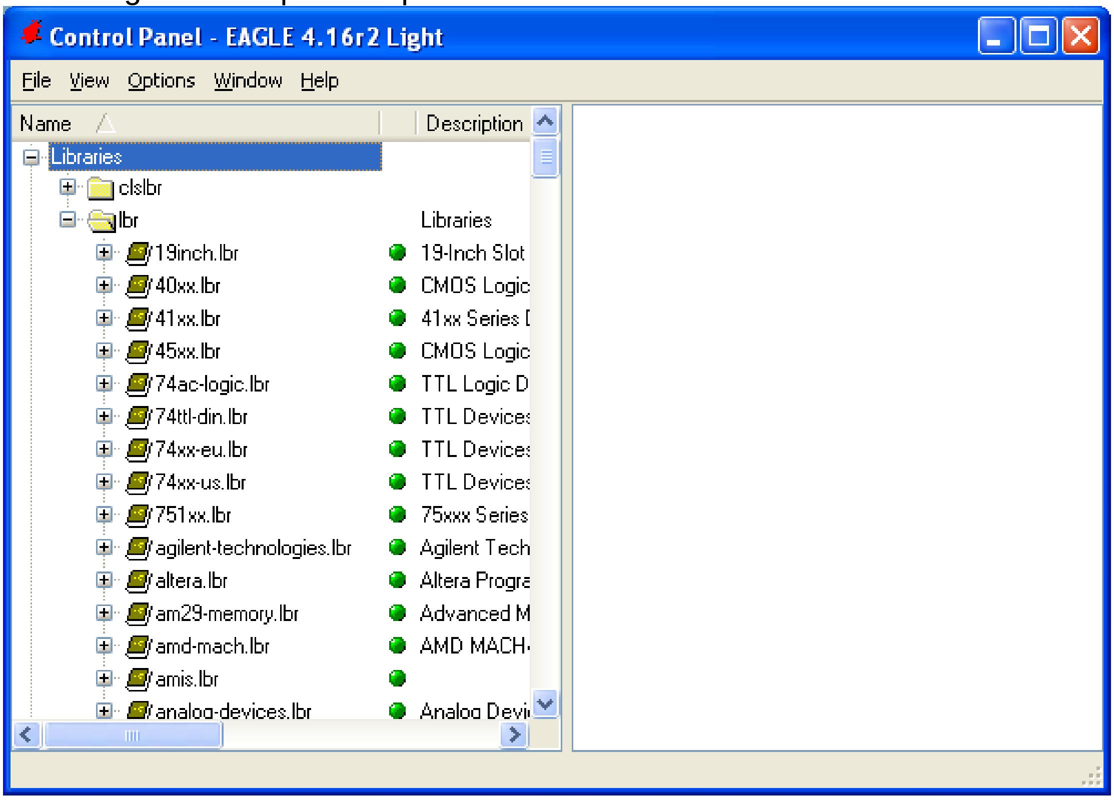
From the Eagle Control Panel Select File – New – Library



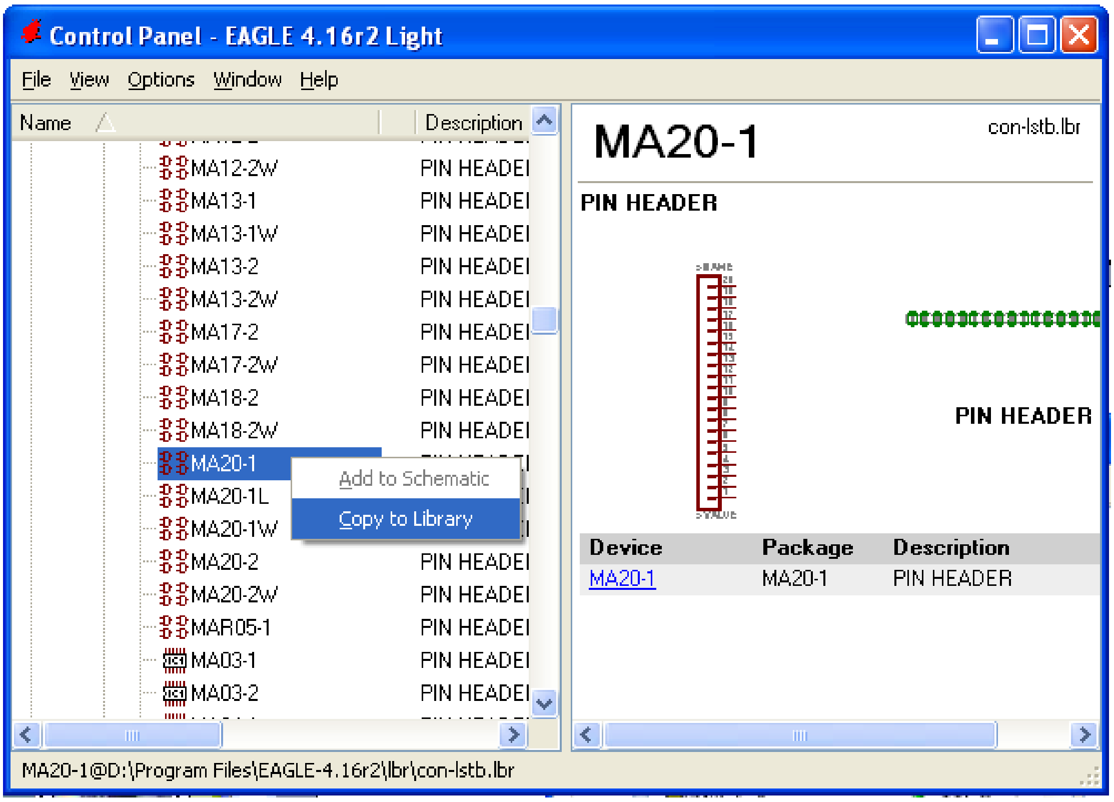
A new [empty] library will open Save it into a suitable location



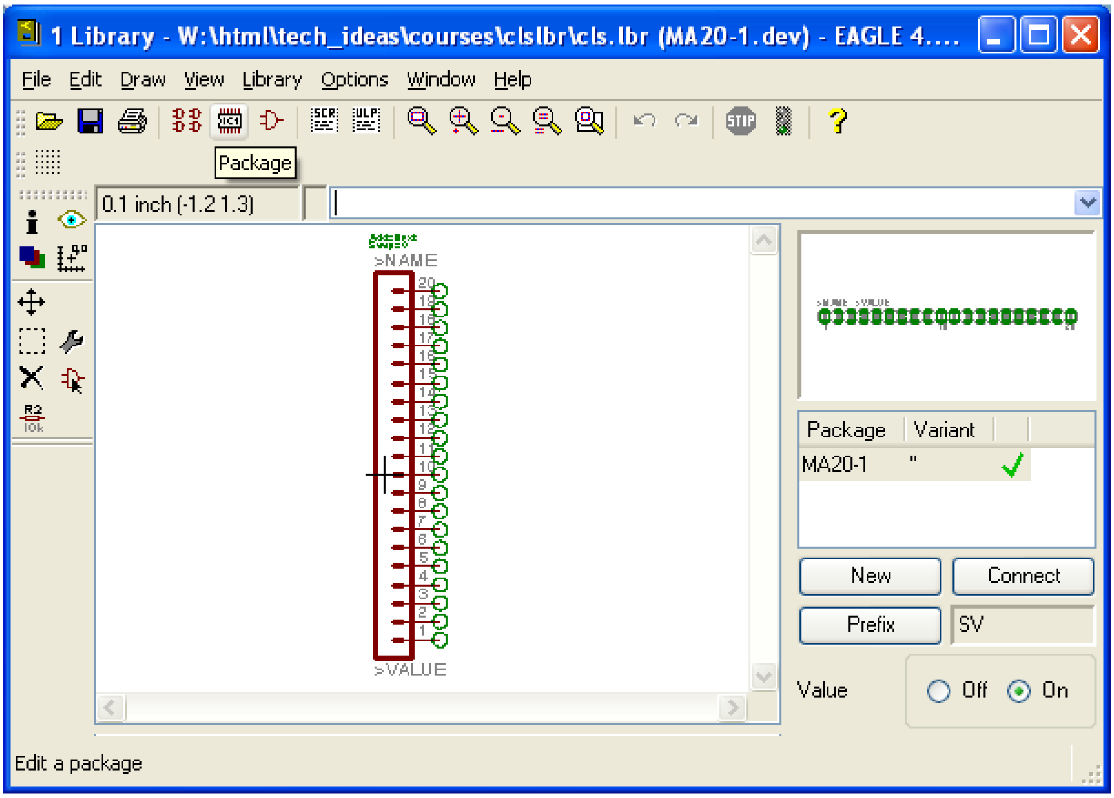
In the eagle control panel expand the libraries



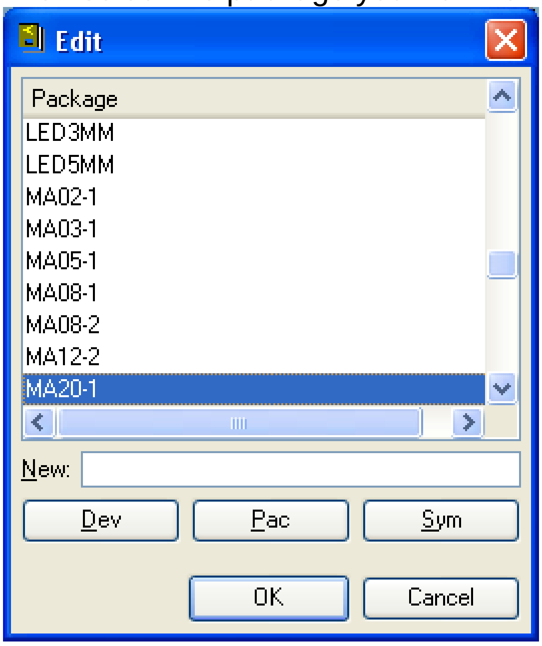
Then expand the library you want to copy a device from and right click on the device, and it can be copied to the open library (it will copy the symbol, package and device)



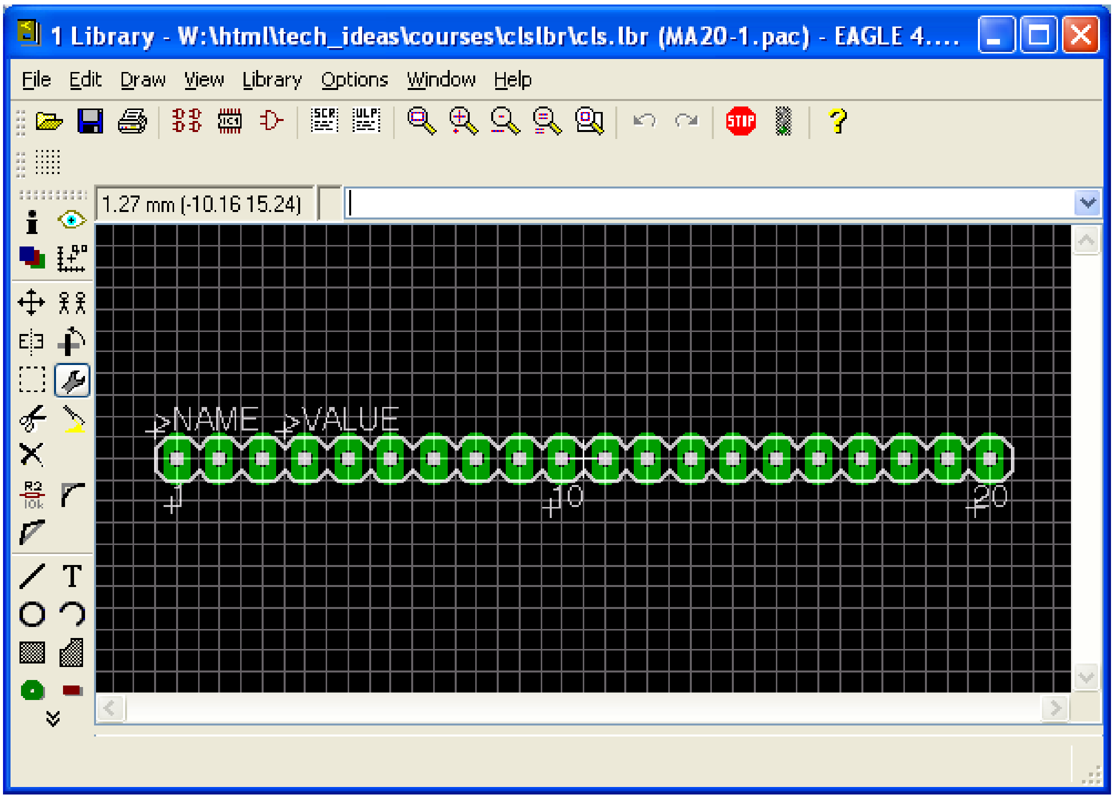
Within your own library you can now modify the package by selecting the package button from the toolbar



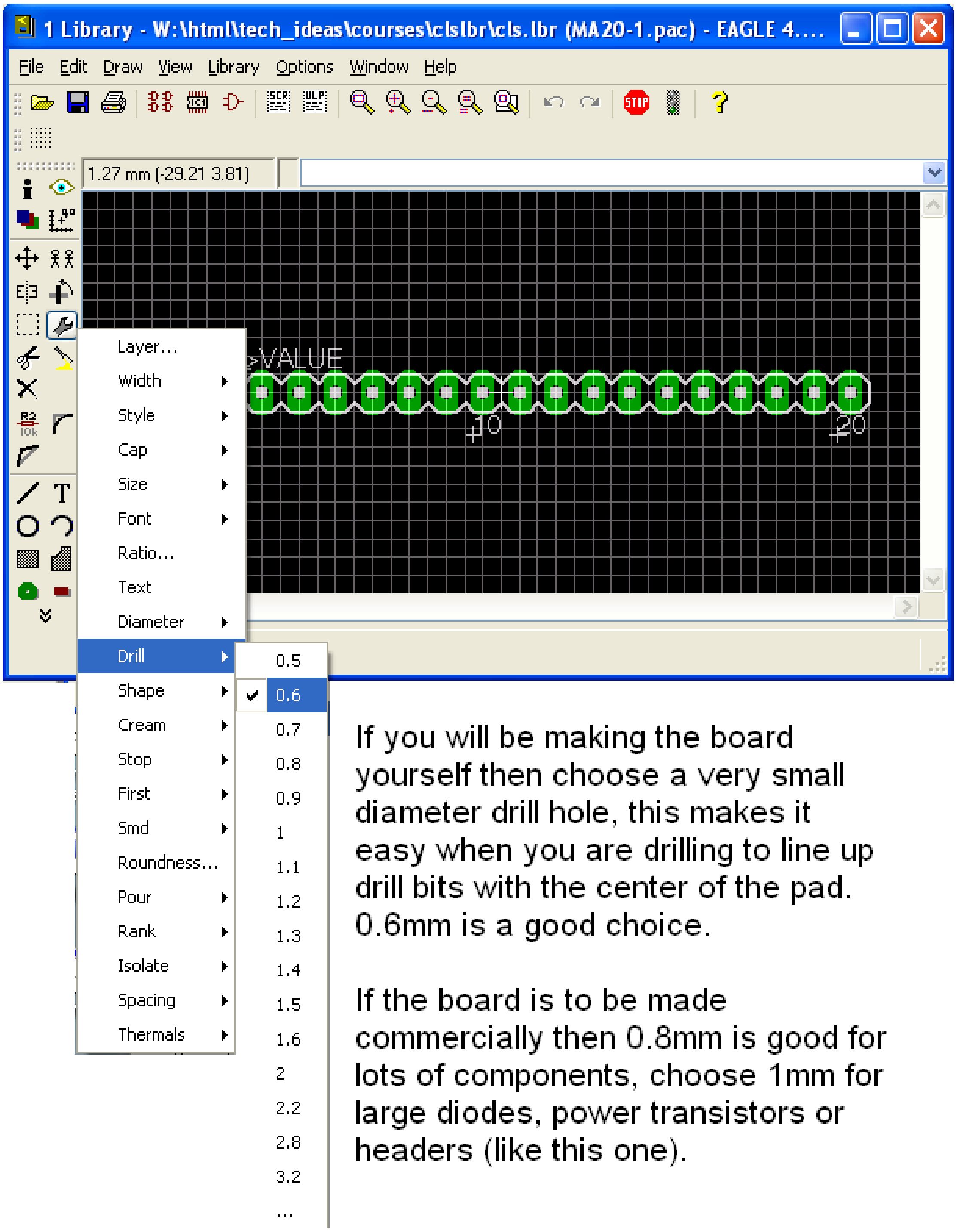
Then select the package you want to modify from the next window and click on OK



The package editor opens, type **grid mm** into the text area and press enter

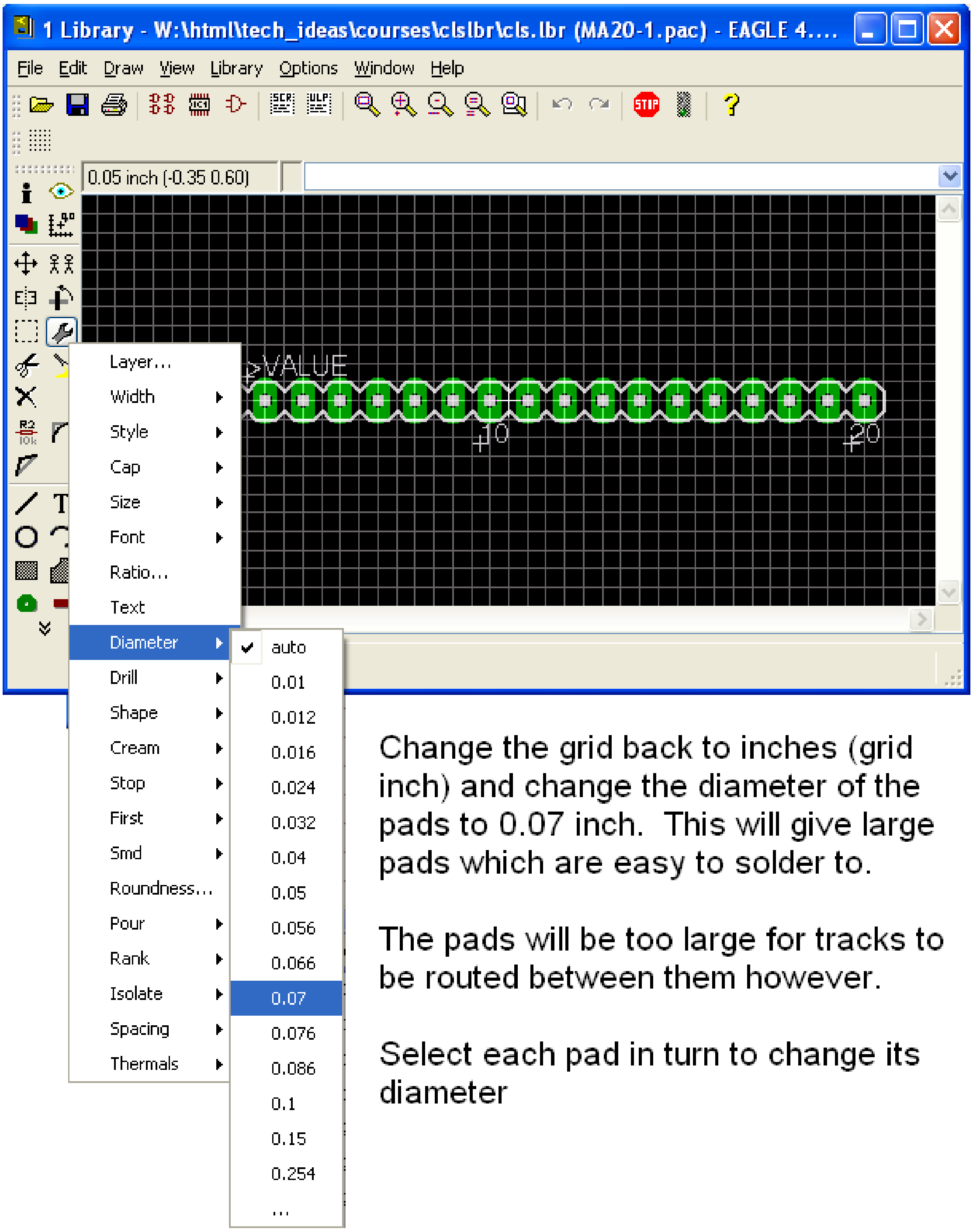


Change the drill hole size for the pads, and **click on each pad** to change its hole size.



If you will be making the board yourself then choose a very small diameter drill hole, this makes it easy when you are drilling to line up drill bits with the center of the pad. 0.6mm is a good choice.

If the board is to be made commercially then 0.8mm is good for lots of components, choose 1mm for large diodes, power transistors or headers (like this one).

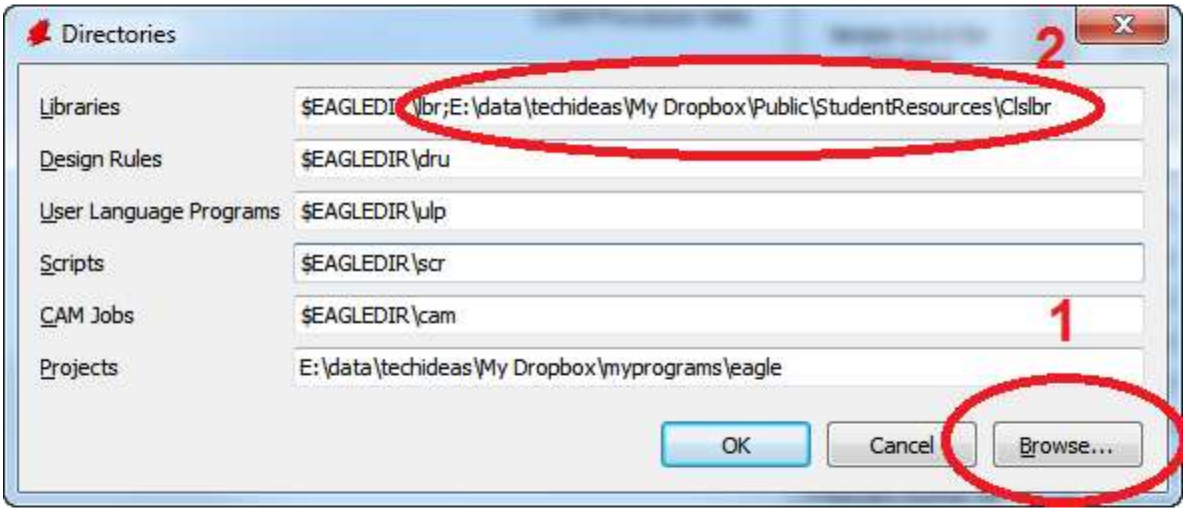


Change the grid back to inches (grid inch) and change the diameter of the pads to 0.07 inch. This will give large pads which are easy to solder to.

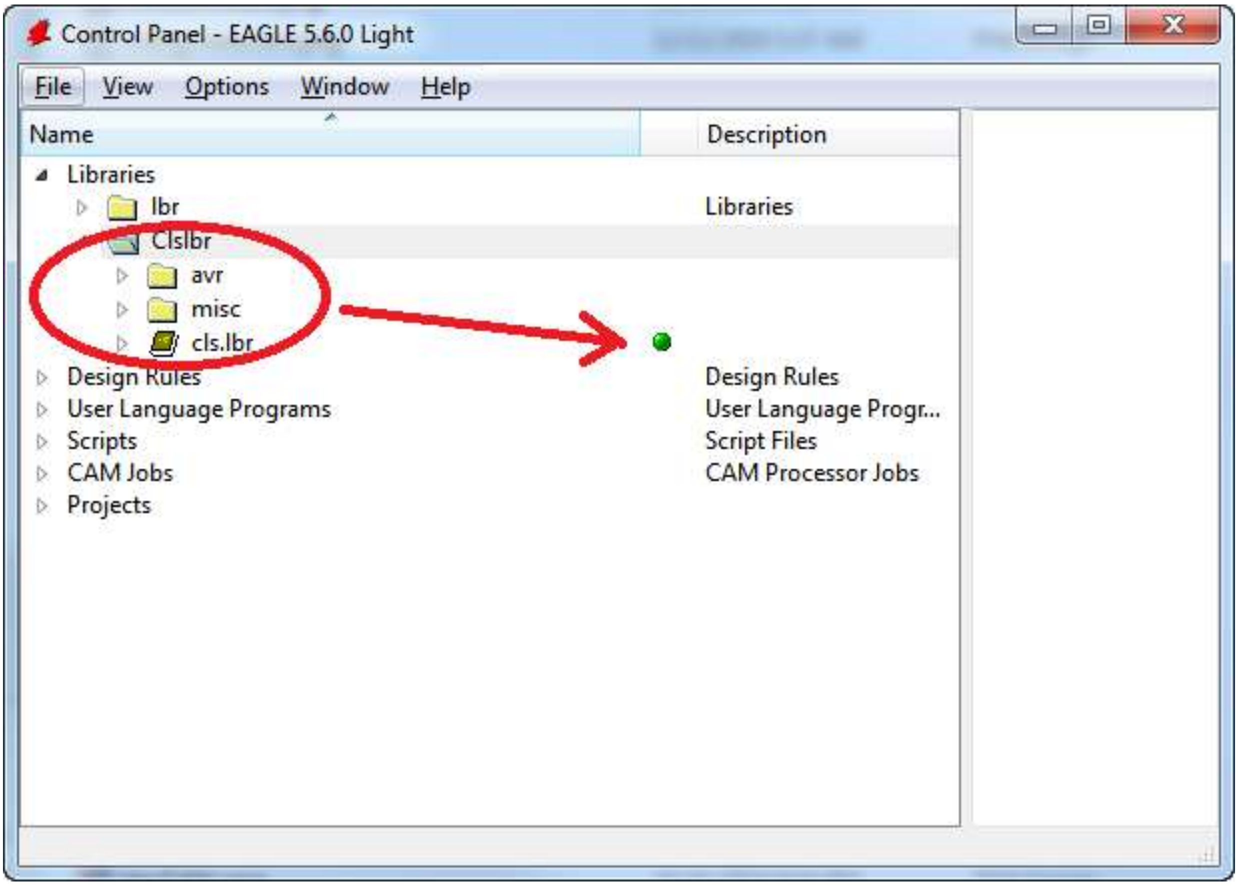
The pads will be too large for tracks to be routed between them however.

Select each pad in turn to change its diameter

When you have finished adding components and editing them, save and close your library.



In Eagle control panel open Options then Directories from the menu and then 1. Click in the librbaires area, browse to your new folder (you can select the folder, you don't select the library itself) and 2 the link will appear in your libraries path.

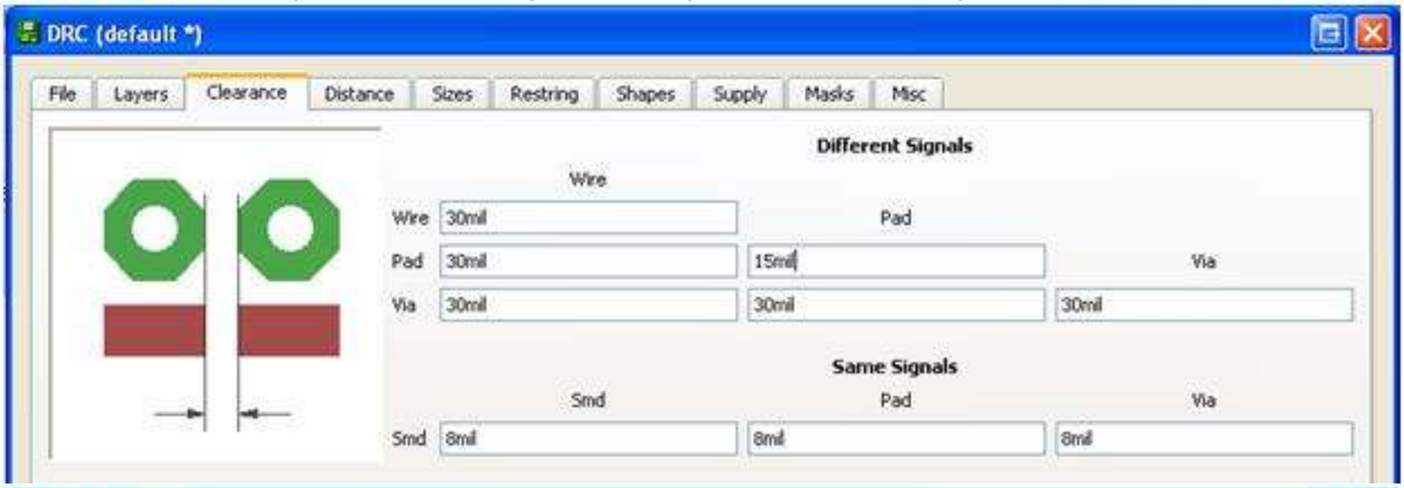


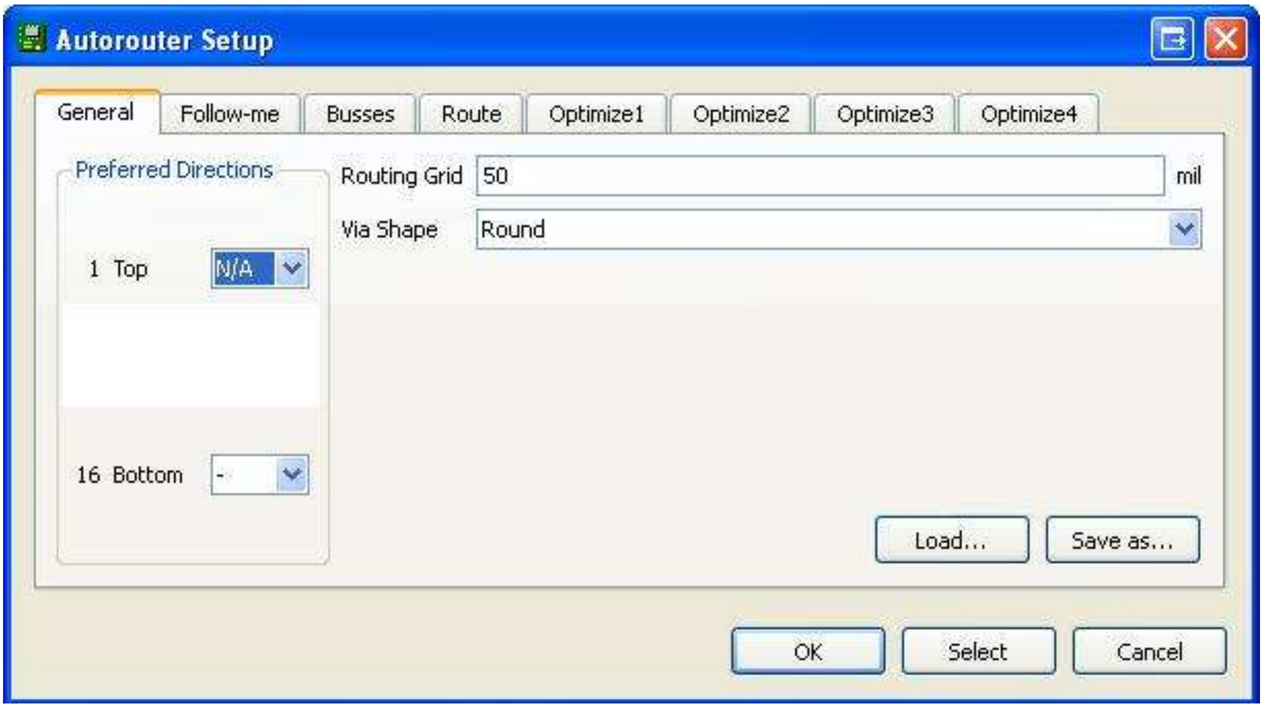
In the main control panelbrose under libraries to your new library and make sure the dot is green, if it's not then right click on the librbay and select USE.

64.1 Autorouting PCBS

Learning to autoroute pcbs is like learning to drive a big truck; it can be a bit dangerous in the hands of someone who can't already drive! So don't think you will auto route your first few boards, learn the basics about laying out PCBs especially about minimising cross overs at the ratsnest stage.

You will have to setup the DRC in Eagle in the layout editor. Before you run the autorouter.

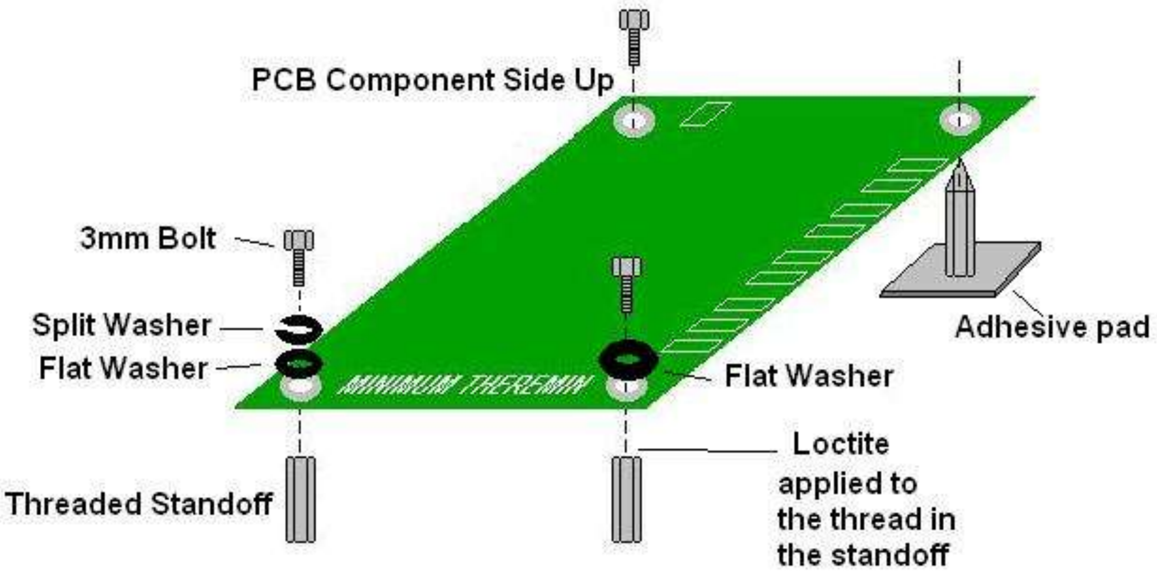




And in the layout editor choose route and make sure Top is set to N/A.

65 Practical Techniques

65.1 PCB Mounting

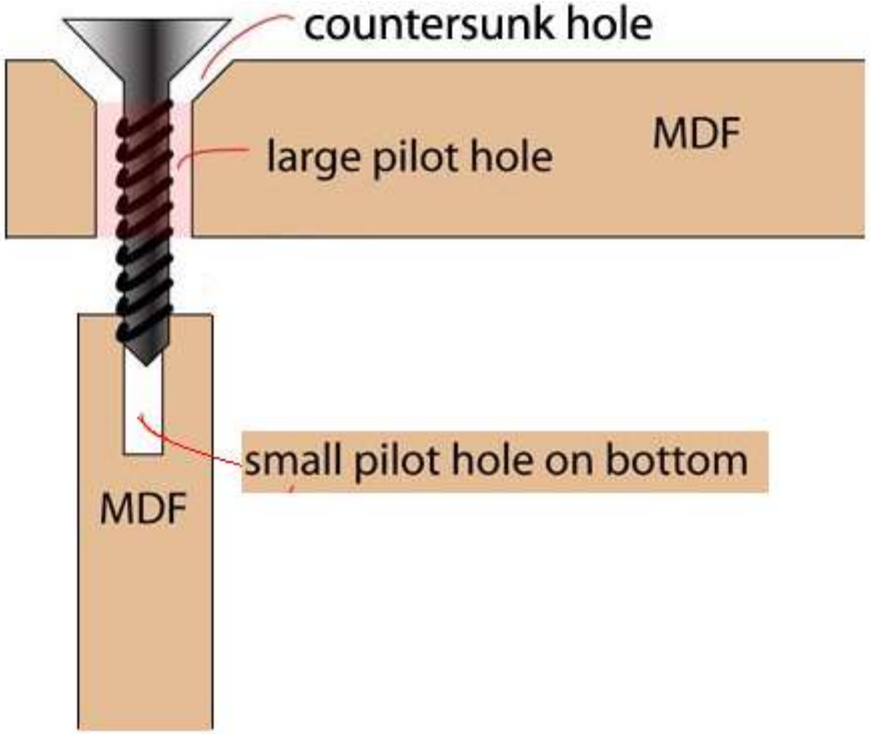


The PCB needs to mounted inside the case.
Usually on some type of standoff.
To keep the bolts from becoming loose with vibration they need to be secured with either a split washer or loctite

Advantage of split washer it is removable	Advantage of loctite it is very strong	Advantage of adhesive quick to fix, Disadvantage - not strong

Using the provided printouts make comments in your journal as to which mounting method you chose and why

65.2 Countersink holes and joining MDF/wood



For CSK POZI Twinthread 4x1/2 Zinc plated screws

- Hold or clamp both pieces together in their final position
- Drill 2mm pilot hole through both pieces (this helps it all line up properly)
- Dill 3mm pilot hole through top piece
- Countersink the top piece so that the screw head sits flush in the MDF
- Use pozidrive screwdriver to drive screw
  - DO NOT OVER TIGHTEN

65.3 MDF

3mm/10mm thick, manmade, composite or engineered wood

Properties:

- Less expensive than many natural woods
- No grain on the surface, so no tendency to split. It does have grain into the edge and screws will generally cause it to split
- Consistent in strength and size
- Flexible - can be used for curved surfaces, will bend under weight.
- Edges are smooth and need no filling like ply when finishing
- Shapes well.
- Heavier than plywood or chipboard (the resins are heavy)
- Swells and breaks when waterlogged
- May warp or expand if not sealed and exposed to moisture
- Dulls blades more quickly than many woods
- Made with urea formaldehyde resins which may cause eye and lung irritation when cutting and sanding. Repeated exposure over many years to dust (all wood dusts) increases the risk of nasal cancers

Processes available in class:

- Marking out, pencil, square, ruler – measure to within 1/2mm
- Drilling – drill press or battery drill. Drill bits, spade bits, holesaws
- Cutting - band saw for longer cuts, large pieces straight cuts only , scroll saw for small cuts and radius, sanding (hand, machine)
- Sanding – hand sanding preferred, belt sander is too aggressive as the wood is soft.
- Gluing and Nailing – done together, nailing on its own will not hold, glue used is PVA, make sure glue covers the full edge, quickly wipe excess of with a damp (not wet) cloth. Nails or screws should be 25mm from end when screwing into and edge
- Milling – circles, slots, needs solid surface underneath to avoid cracking out, timber can burn

http://www.nelsonpine.co.nz/School.htm
http://www.thelaminexgroup.co.nz/pdf/products/TLG6013%20LakepineBroc.pdf

3mm cost: \$3.75 per m^2 10mm cost:\$9.30 per m^2

65.4 Plywood

manmade, composite or engineered wood

Properties:

- 9mm thickness
- Very strong
- resistance to cracking, shrinkage, twisting/warping,
- can be manufactured in sheets far wider than the trees from which it was made
- Economical and effective utilisation of wood
- Light weight
- Veneered – nicer timber on the outer
- This ply is suitable for both internal and external use (water proof glues)
- Marking out, square, ruler – measure within 1/2mm accuracy

Processes available in class:

- Drilling – drill press or battery drill. Drill bits, spade bits, hole saws
- Cutting - band saw for large pieces, longer cuts, straight cuts only – safety, scroll saw or fret saw for small cuts and radius, sanding (hand, machine)
- Sanding – hand sanding or belt sander
- Milling – circles, slots, needs solid surface underneath to avoid cracking out
- Gluing and Nailing – done together, nailing on its own will not hold well together, glue used is PVA, make sure glue covers the full edge, quickly wipe excess off with a damp (not wet) cloth
- Wood screws - can screw into the edge of plywood if the screw is no larger than 1/3 the thickness of the timber

9mm ply cost: \$18.28 per m^2

65.5 Acrylic

(Polymethyl methacrylate)- 3mm thick clear

Properties
- Thermoplastic - it will soften when heated so it can be formed into different shapes easily.
- Hard and Rigid
- Good surface finish
- Scratches easily
- Liable to crack not bend when cold
- Has a thin covering on both faces to protect it from scratching, do not remove

Processes available in class:
- Marking out, pen, square, ruler – measure within 0.5mm accuracy
- Drilling – drill press or battery drill. Drill bits, no spade bits or hole saws, needs solid surface underneath to avoid cracking
- Milling – circles, slots, needs very solid surface underneath to avoid cracking
- Cutting - band saw for large pieces, longer cuts, straight cuts only – scroll saw for small cuts however as the scroll saw cuts the material the cut is very thin and the acrylic is heated causing the cut to melt back together
- Sanding – hand sanding or belt sander (larger pieces on edges only), use progressively finer sand paper to polish to get a glassy edge finish
- Bending – use the strip heater, support the material on both sides until soft enough to bend easily. Hold in shape required until cool, can be cooled under cold water, remove the clear covering before heating
- Gluing, special glue is required, avoid skin contact
- Nuts and bolts washers
- Tapping

3mm Acrylic cost: $43.72 per m^2

65.6 Electrogalv

0.8mm thick, zinc coated mild steel **in lengths of 1200 and widths of 120, 150, 180, 220, 250, 280, 300mm**

Properties
- Very strong
- Avoid scratching the surface to remove the coating
- Will bend under excess weight
- Strengthened by bending
- can be painted easily

Processes available in class:

- Marking out, scriber, square, ruler – measure within 0.5mm accuracy
- Drilling – drill press or battery drill. Drill bits, no spade bits or hole saws, needs solid surface underneath to avoid distorting the steel. Must centre punch before drilling too stop the drill wandering while drilling. With holes larger than 5mm use a small (3mm) drill to make a pilot hole first.
- Cutting – guillotine, shears, tin snips (absolutely not the bandsaw)
- Bending – use magnabender
- Filing, file all edges to remove burrs (sharp points) and smooth corners
- Nuts and bolts - removable
- Machine screws – removable
- Rivnuts - removable
- Pop-rivets - permanent
- Spot welding – permanent
- Nibbler – rectangular holes

0.8mm Electrogalv cost: $33 per m^2

65.7 Choosing fasteners

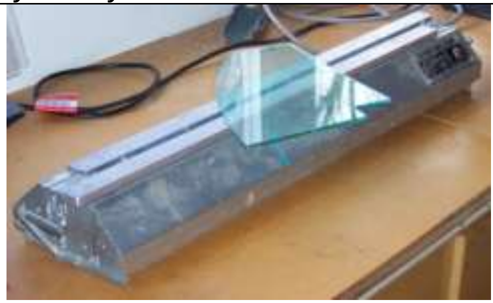
Countersunk Machine Screw 	Pan Head Machine Screw 	Nyloc (nylon locking insert) Nut 	Rivnut 
Self Tapping Screw 	Countersunk Wood screw 	Jolt head panel pin (nail) 	Hinge 

Name the different fasteners used

	From Wood	From Metal	From Acrylic
To Wood			
To Metal			
To Acrylic			

65.8 Workshop Machinery

Give its name – materials used for and key safety considerations

65.9 Glues/Adhesives

PVA – (wood to wood)
Polyvinyl acetate is a water based adhesive which is coloured white.
PVA works when it soaks into the surface of the wood and sets once all the water is absorbed.
PVA makes an extremely strong bond and is often stronger than the actual wood fibres itself. PVA is good to for gluing wood to fabric.

Solvent cement – (plastic to plastic)
There are many types of solvent cement however the most common is dichloromethane.
Dichloromethane works by dissolving the surface of hard plastics such as Acrylic and High impact polystyrene.
Solvent cement is very dangerous and will give off fumes so it is important to use this within a well ventilated room.
Solvent cement is good to for gluing plastic to plastic.

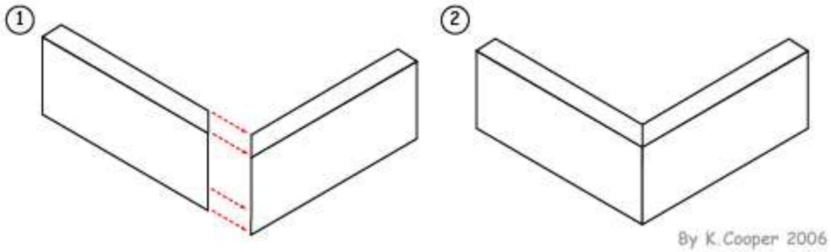
Hot glue guns – (Card to card / modelling)
Hot Glue guns are used a lot in schools for quick modelling of work.
However these can be rarely used on final products as it is not strong enough.
Hot Glue guns are good to for gluing card to card and modelling materials together.

ADOS F2 – Almost anything to anything
ADOS is a contact adhesive, you apply it to both parts and then leave them to dry
Line up the two parts extremely accurately and press firmly together, once they touch they cannot be moved
It can be messy so not good for things that need to look nice.

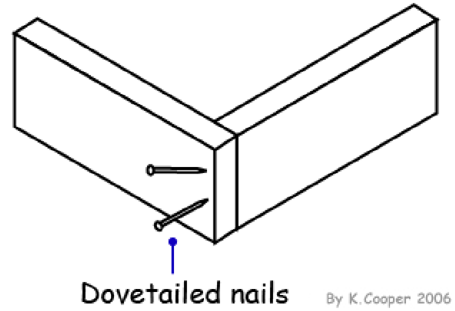
65.10 Wood Joining techniques

When using a style of wood joint choose the most appropriate and say why you chose it.

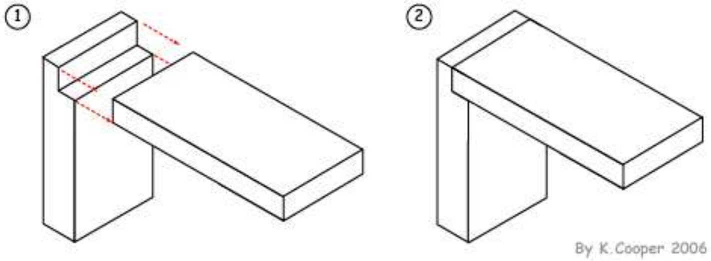
Wood - Mitre Joint



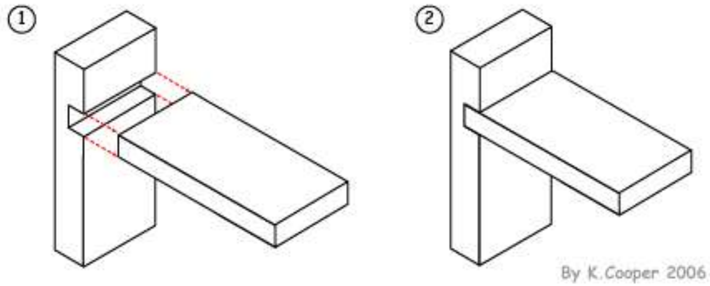
Wood - Butt Joint



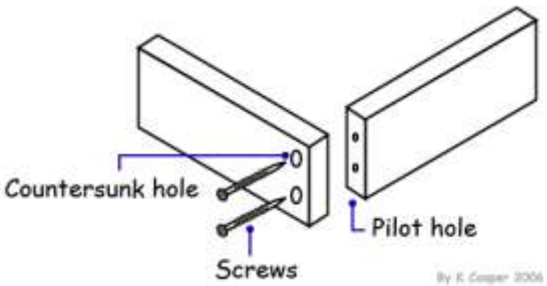
Wood – Lap Joint



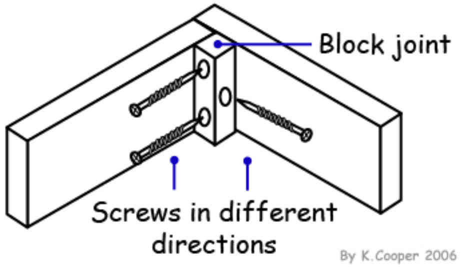
Wood - Housing Joint



Wood – Screws



Wood – Block Joint



USEFUL WEBSITE www.mr-dt.com

65.11 Codes of Practice for student projects

<table>
<tr><td colspan="3">Codes of practice are industry recognized ways of carrying out work on your project, so that it is safe for users and provides reliable operation.</td></tr>
<tr>
<td><u>Materials and processes</u>
materials used suit the final situation
processes used (e.g. joints) suit the final situation
no sharp parts or sharp corners
no loose or small parts

<u>Environmental</u>
discuss any recycled or recyclable materials used
discuss any hazardous materials used
discuss any hazardous waste generated

<u>Legal</u>
laws/regulations e.g. for children's toys
electrical laws ECP50 & AS/NZS3820
copyright laws, is your work original?
If another’s logo is used, was it authorised?
note owners do not like their logos modified.

<u>Documentation / user instructions</u>
clear explanations for end users
care instructions for the product
warnings of hazards</td>
<td><u>PCB CAD design</u>
System block diagram is drawn first
Schematic layout guidelines
Circuit is laid out to follow block diagram
0V or GND wires are at bottom of schematic
+V or battery connections are at top of schematic
use European symbol standards
minimise crossovers of lines in schematic
components correctly chosen for size and rating
use nets not wires in schematics
use junctions to show joining of nets
all components named and given values
name, date and version on schematic
PCB layout guidelines
layout size and shape to suit case limitations
place large components first
minimize ratsnest crossovers
no tracks between pins of an IC
track width minimum 0.04”
track spacing minimum 0.025”
large pad sizes for wires off the board
add pcb mounting holes to layout
add places for cable stress relief to layout
name, date and version on layout
avoid excessive track length
minimize board size
placing of decoupling capacitors next to IC’s</td>
<td><u>Electronic work</u>
good solder joints
wire insulation stripped correctly
no loose or cut strands of wire
no splashes of solder
no holes in solder joints
circuit boards securely mounted
batteries securely mounted
no stress on wires
no sharp edges of case to damage wires
heat shrink used to cover solder joints to stop shorting and provide mechanical strength
label all user controls

<u>Software</u>
intuitive operation for users
files are backed up often to other locations
files are progressively kept
title block at beginning of code to explain operation
comments used throughout code to explain function
constants are used instead of values
code broken up into subroutines or procedures
labels and variables have useful names
modifications are recorded</td>
</tr>
</table>

65.12 Fitness for purpose definitions and NZ legislation

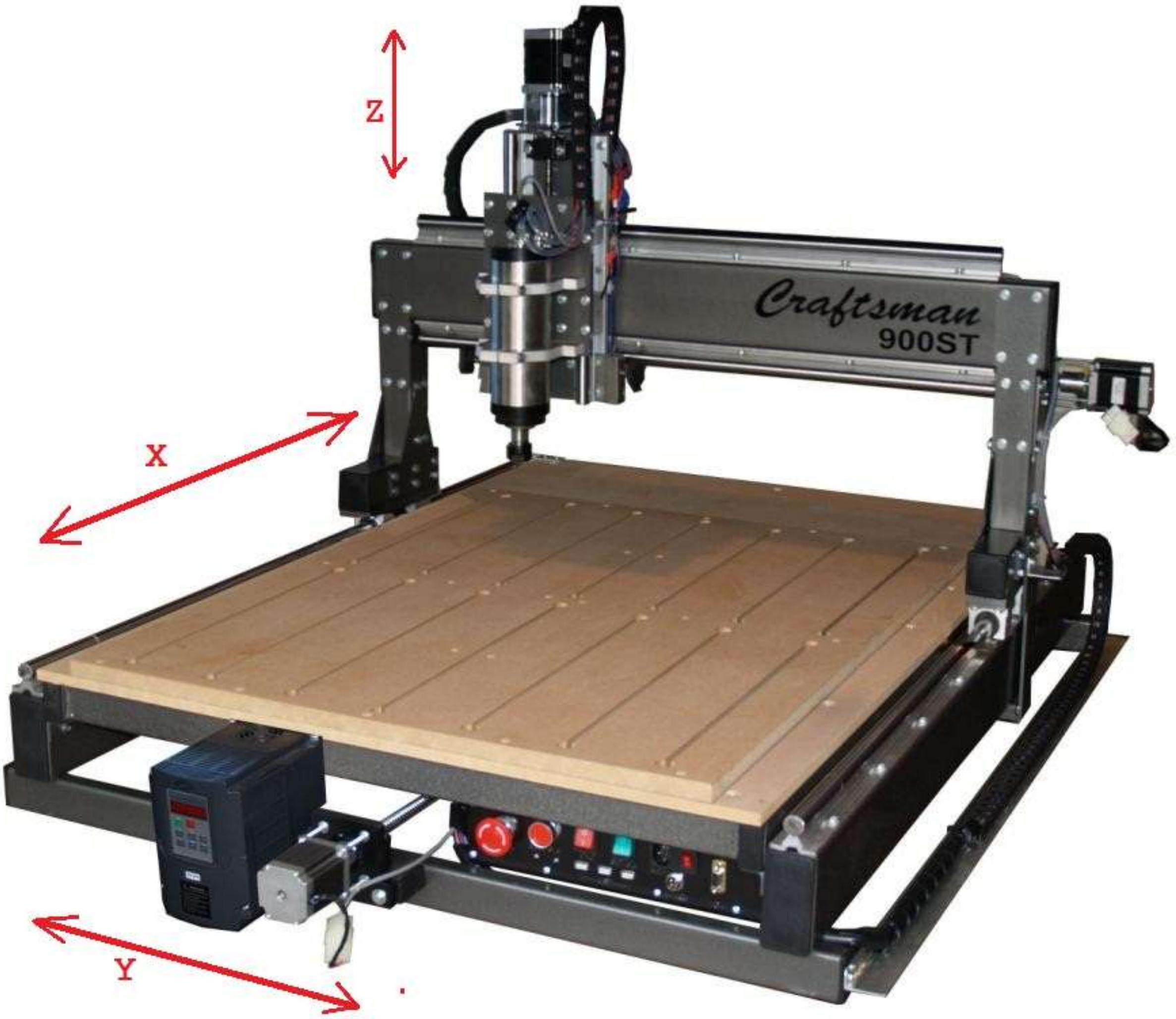
A product that has been manufactured to a standard that is acceptable to the end user.
http://www.sinclair-consultancy.sagenet.co.uk/glossary.htm
A criteria used in evaluating a product; the evaluator asks how well the product performs the function for which it was designed. If the product performs well then the product is said to have fitness for purpose
http://www.primarydandt.org/learn/glo_0000000323.asp
The notion derives from manufacturing industry that purportedly assesses a product against its stated purpose. The purpose may be that as determined by the manufacturer or, according to marketing departments, a purpose determined by the needs of customers. http://www.qualityresearchinternational.com/glossary/fitnessforpurpose.htm

'Fitness for purpose' is commonly used to judge the ability of an outcome to serve its purpose in 'doing the job' within the intended location, where the 'job to be done' is clearly defined by the brief. Referring to 'fitness for purpose' in its broadest sense within technology education, correlates to an extension of this usage to include the determination of the 'fitness' of the practices involved in the development of the outcome, as well as the 'fitness' of the outcome itself, for the identified purpose. Extending the concept in this way is an attempt to locate both the concept and its application within a sociocultural understanding of the nature of technological practice whereby the performance of outcome is but one of the factors that justifies a positive 'fitness for purpose' judgment.
http://www.techlink.org.nz/glossaryitem.htm?GID=2

NZ Legislation: Guarantee of fitness for particular purpose under the Consumer Guarantees Act
The Consumer Guarantees Act (CGA) is about the quality of goods and services. It offers protection to customers who have had poor quality work carried out for them by a tradesperson or purchased goods, from a person in trade, that do not meet reasonable expectations. The work you do must achieve any particular result the customer wants and has told you about.
e.g., John wants a drainage system that will stop his lawn from flooding every time it rains.
You must tell the customer before you start the job if you can't guarantee that the job you do will achieve the purpose or the result they want. Otherwise you will be liable under the Act for not having achieved the desired purpose. This guarantee applies to particular purposes that the customer has told you about. Normal purposes for the work you are doing will be covered by the guarantee that you will use reasonable care and skill.
Does the customer have to specifically tell me what they want?
If the purpose they want to achieve is a normal purpose then the customer does not have to specifically state it.
e.g., if a customer wants a tap replaced it is obvious that they will want the tap to turn on and off and to deliver a reasonable flow of water. Where the result wanted is less ordinary the customer must let you know exactly what they want.
e.g., if Rita wants a particular pattern for her paving stones she must tell you exactly how she wants it done.
Writing down exactly what you have agreed to do in a written quote or contract is a good way of avoiding any debate about what was agreed.
What if I can't be expected to know if it will work?
Sometimes it will be obvious that the customer can't expect to rely on your skill to achieve the desired result.
e.g. , Julie ask the painter to cut back a tree that will get in the way of the painting. The painter agrees and charges for the time it takes. The tree dies and Julie wants the painter to pay compensation. Julie knew that the painter was not a tree surgeon and that she couldn't rely on the painter having the skill to trim the tree successfully.
Sometimes you may want to tell the customer that you can't guarantee that you have the skills to do the job.
e.g., Fran's car has a recurrent problem with the generator. The mechanic at her local garage has looked at it once and told her it is a job for an auto-electrician. Fran asks him to have another look at it anyway as she doesn't want to have to take the car to an auto-electrician in town. In this case the mechanic has told the customer that they may not have the specialist skills needed. Fran will not be able to claim that the work was not fit for the purpose.
If you are in a similar situation you must make it clear to your customers that you may not have the skills required.
What if the customer has chosen the cheapest option?
Sometimes the customer will ask you to use the cheapest option. *e.g., Jan asks her painter to put only one topcoat on her house as she plans to sell it.* Bruce is told that his radiator needs a new core. Bruce says he can't afford it and asks the garage to just solder up the leak. In these cases the result may be less fit for its normal purpose than if the customer had been prepared to pay the extra money for the second coat of paint or the new radiator core. You may want to get the customer's agreement in writing that they have chosen the cheaper option.
e.g., we have repaired this radiator by soldering the leak as requested. In our opinion the radiator core needs replacing.
You must still guarantee the quality of the work done but clearly there will be a lower expectation on the work. You should not use wording such as "This work is not guaranteed". This could be interpreted as an attempt to contract out of the Consumer Guarantees Act.
http://www.consumeraffairs.govt.nz/businessinfo/cga/services.html#purpose
Using the "definitions and comments above, comment on your projects "fitness for purpose" with regard to your stakeholder's specifications.
N.B. Even though you may not be selling your product to an end consumer and it may not even meet the definition of a personal or household use item under the CGA the explanations above help our understanding of fitness for purpose.

66 CNC

The CNC (computer numerical control) machine in class is a useful tool which allows an automated approach to drilling and milling PCBs, cases etc.



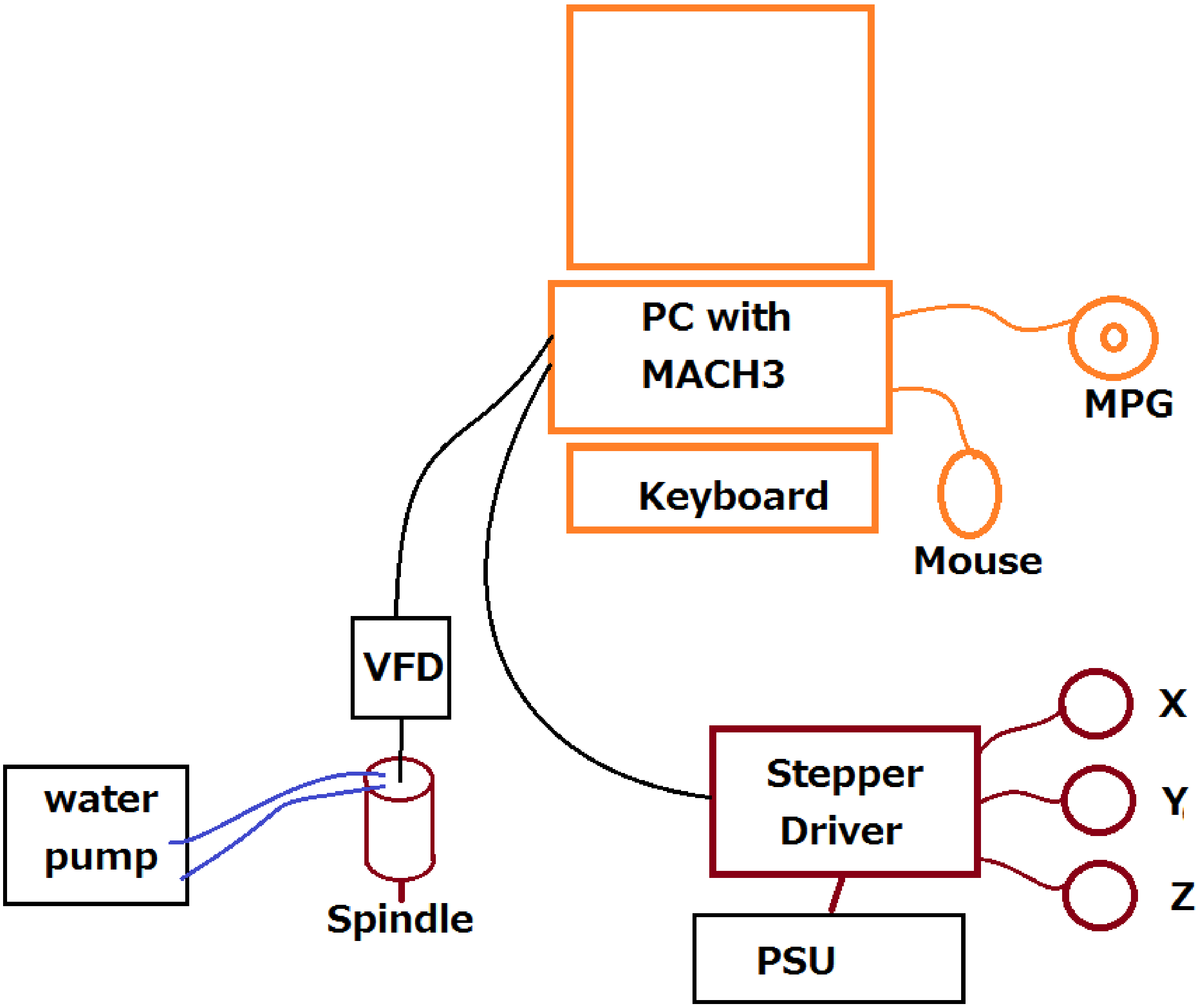
The machine contains its own PC and is connected to the network so that files can be transferred to it directly. The PC in the cnc machine runs **MACH3** cnc software, which interprets **gcode** commands to control the machine.

Gcode consists of a text file with commands to control the machine e.g.

```
G90 G21 G49
M3 S15000
G0  Z15.000           Y100.000     Z15.000
G1  Z-4.500     F400
X26.473      Y129.066     Z-4.500
X57.024      Y153.810     Z-4.500
X352.976     Y153.810     Z-4.500
X383.527     Y129.066     Z-4.500
X410.000     Y100.000     Z-4.500
G0  Z15.000
G0 X0 Y0
M05
G0 Z0
M30
```

It is not necessary to write gcode it is best to draw what you want in a graphics program that creates gcode.

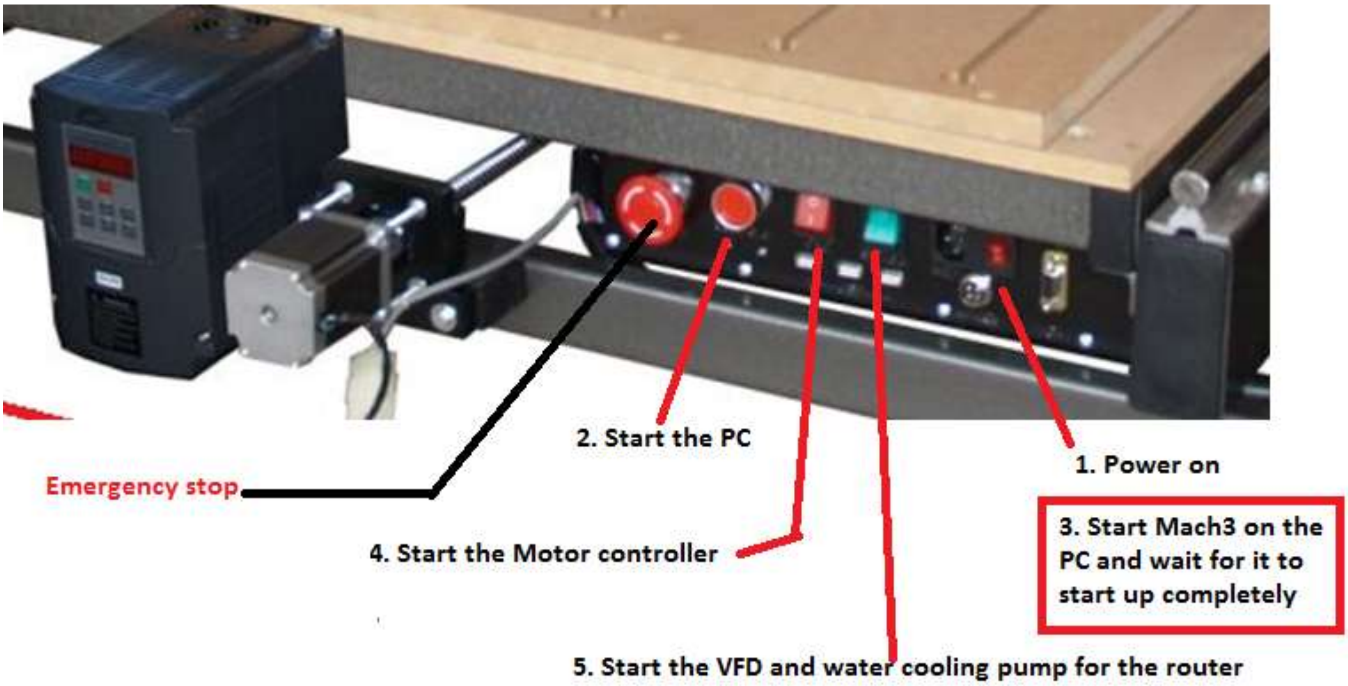
66.1 Machine overview



The 3 stepper motors (X/Y/Z) are driven by a stepper driver circuit board connected to the PC (MACH3 software must be running before turning this on)
The spindle (router) itself is controlled by the PC via a VFD (variable frequency drive), which outputs a high voltage at different frequencies to vary the speed
400Hz output = 24,000 rpm (revolutions per minute)
300Hz output = 18,000 rpm
200Hz output = 12,000 rpm

The keyboard and mouse control the PC, however another controller the MPG (manual pulse generator) controls some features of the device as well.

66.2 Starting the CNC machine

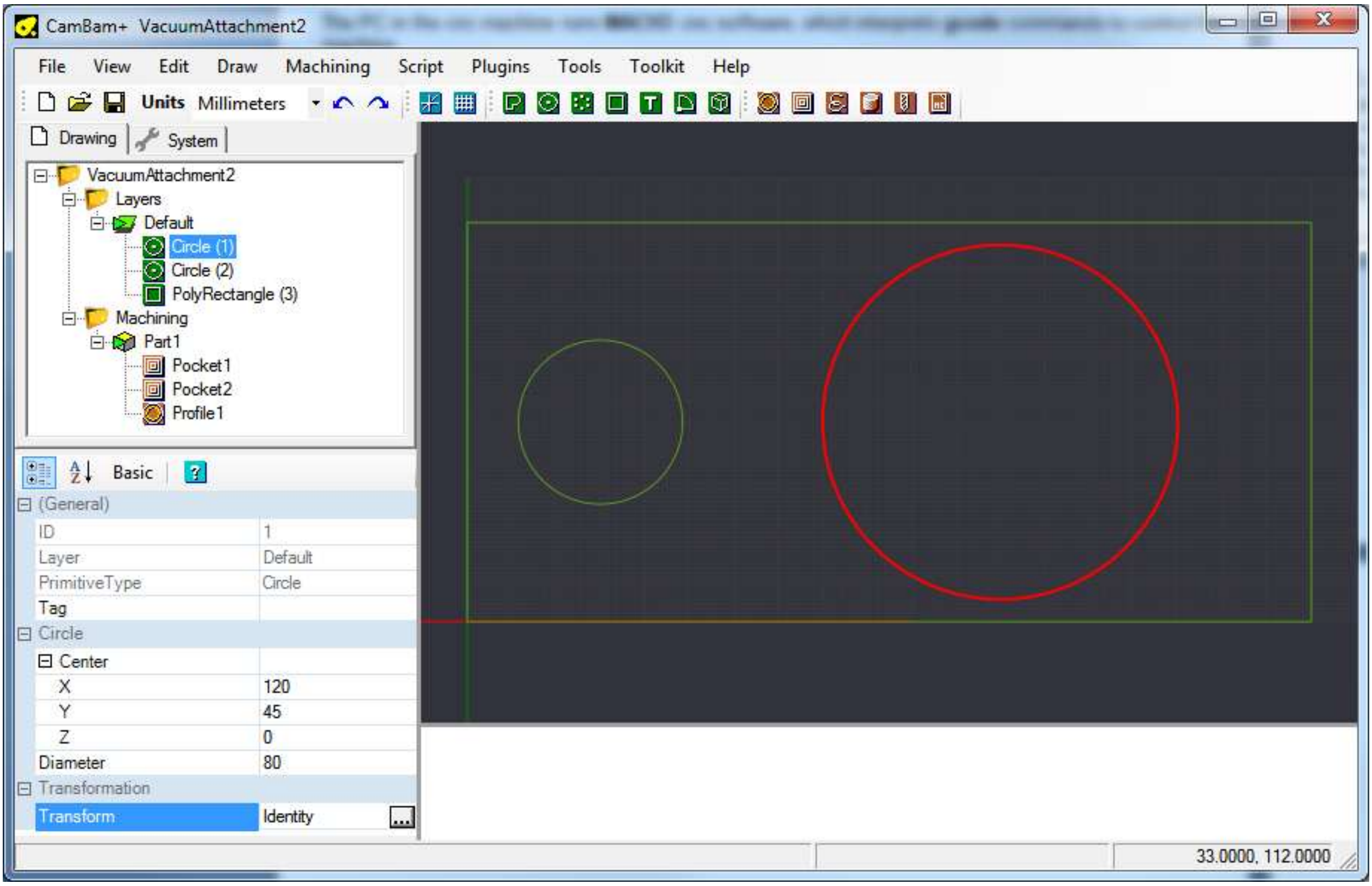


You must wait for the program MACH3 to start before turning on the motor controller, otherwise the motor controller could start in an unreliable state.

The most convenient way of controlling the machine manually is via the shuttleexpress controller

Four of the buttons have been programmed into the mach3 software, press X to move the machine on the X axis, and then change the outer and inner rotating dials to move the machine. When the machine is in the position you want it to be then press the 4th button to zero the axes in mach3.

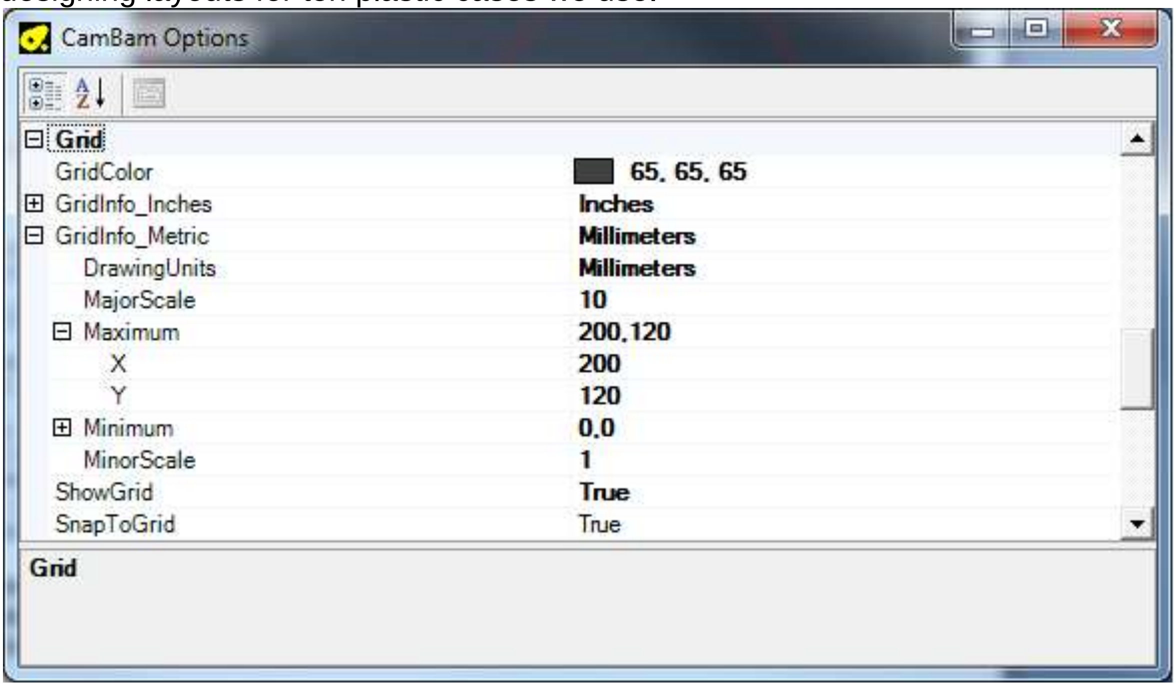
66.3 CamBam



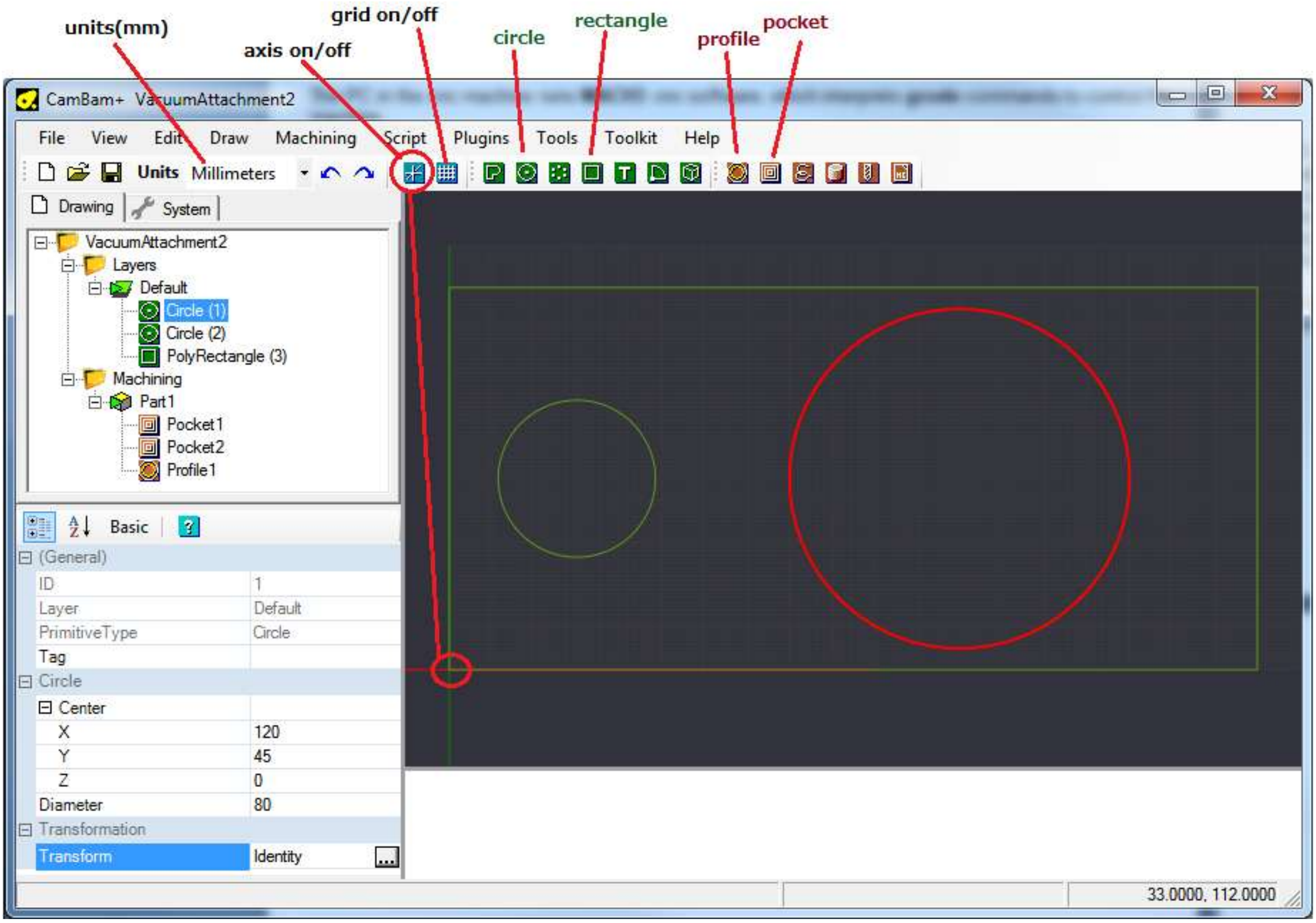
This software allows users to easily create gcode.
There are 4 stages to using this software.
Stage 1: draw shapes
Stage 2: determine the machining for each shape.
Stage 3: review the machining
Stage 4: export gcode.

66.4 CamBam options

Make sure you adjust the options (Menu-Tools-Options), I find the size of 200x120 useful when designing layouts for teh plastic cases we use.

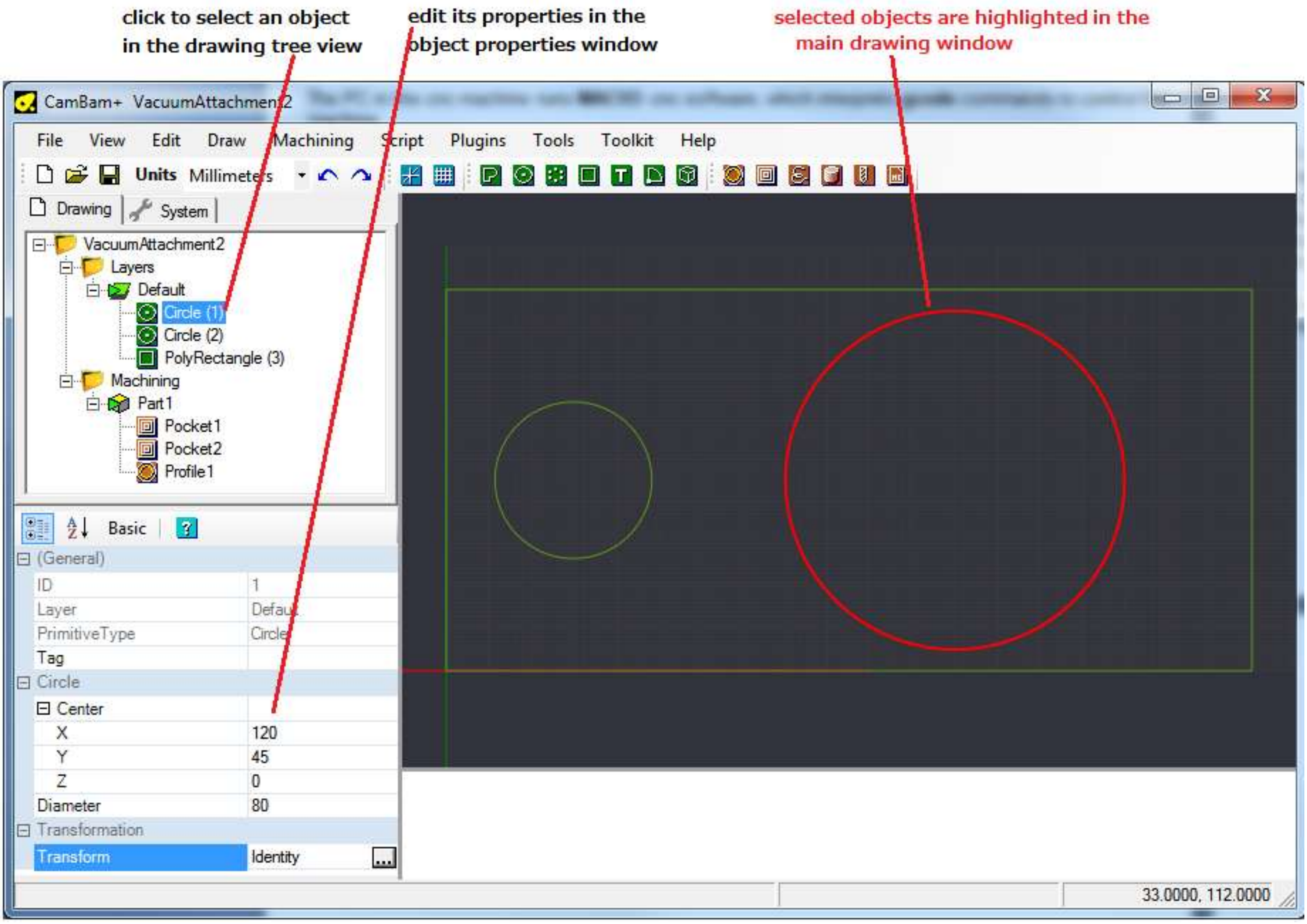


66.5 Drawing shapes in CamBam



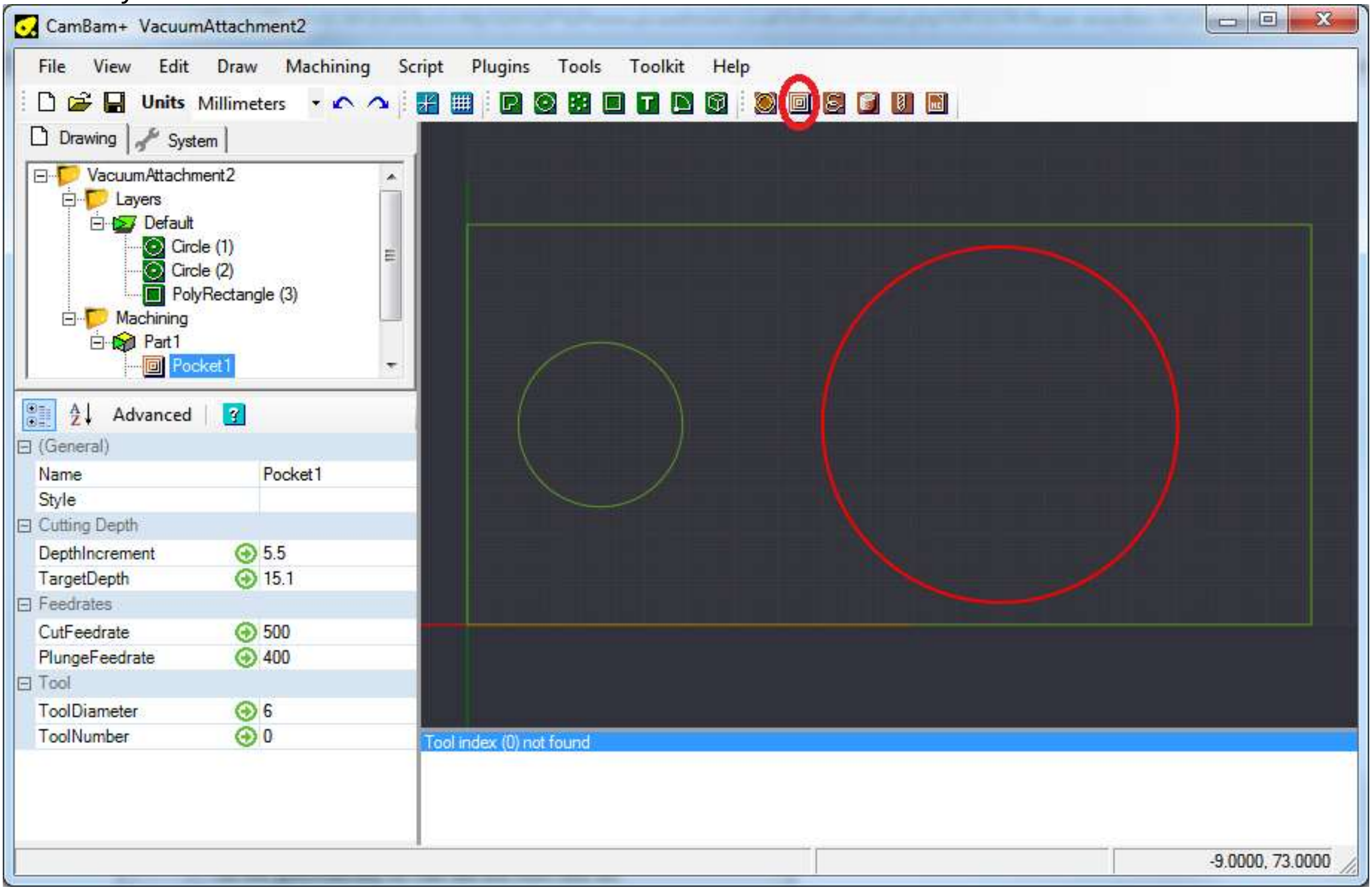
Make sure the units are in millimietres, and the axis and grids are both on.
Select the circle tool and draw the circle in the drawing window
Select the rectangle tool and draw the rectangle in the drawing window

To edit an object, click it in the object tree view and then edit its properties, such as location (X,Y,Z) and diameter.



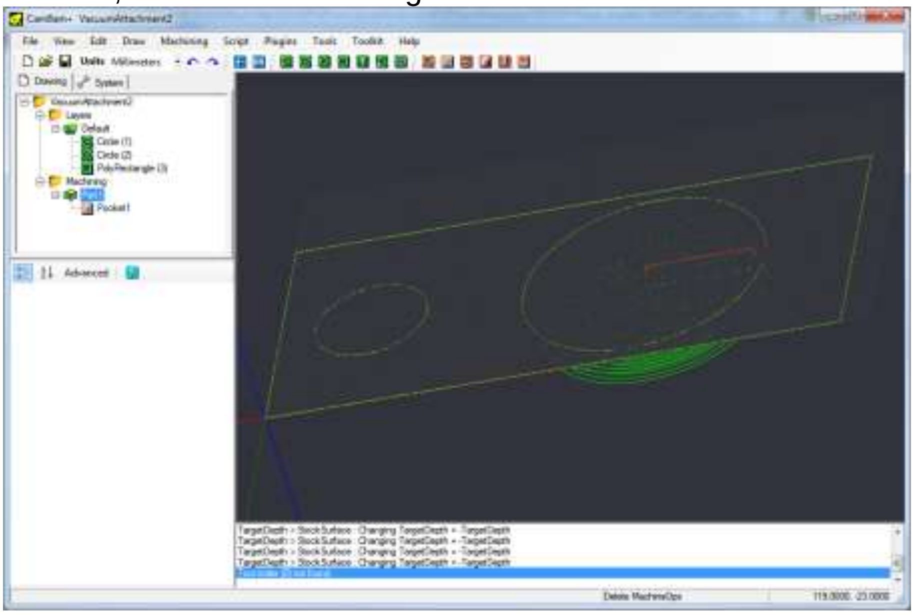
66.6 Machining commands

Once a shape is created it needs to have machining added to it. Here you wll tell the cnc machine what you want down to the shape, it could become a profile or a pocket.
Highlight the shape(s) you want to attach the machining to and click on the pocket tool to cut the whole area away.



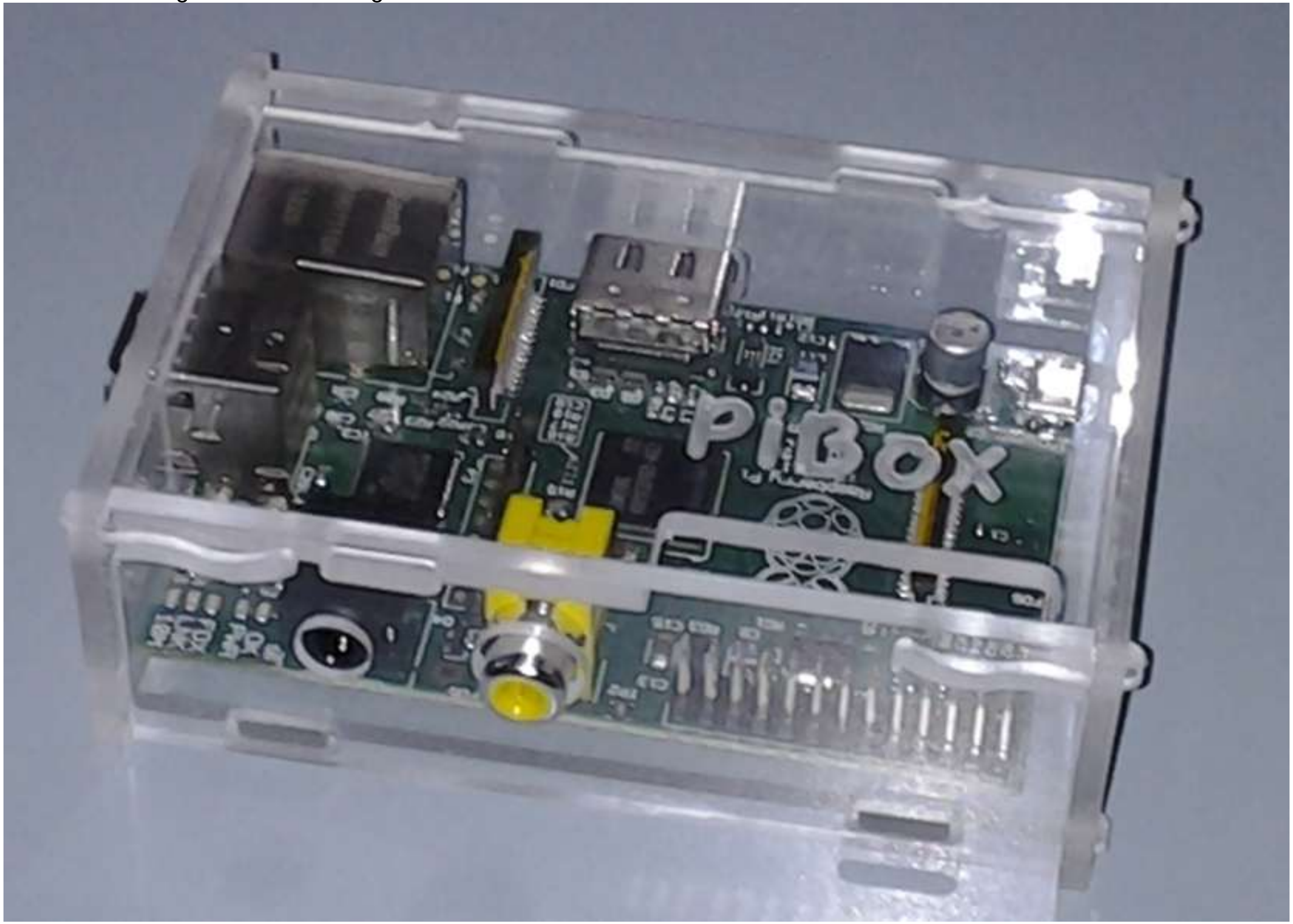
Set the target depth, for 15mm thick material I cut a 15.1mm hole. And I di it in three passes each of 5.5mm depth. This was acrylic so I set the cut rate high at 500mm/minute. I was using a 6mm tool as well.

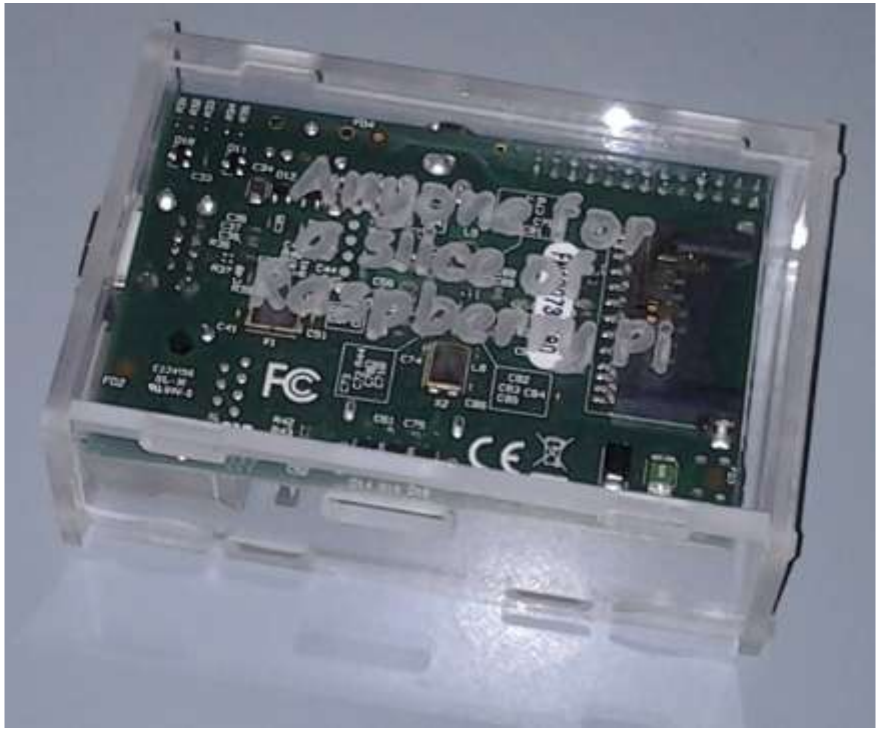
If you right click on the drawing and select Machining – Generate Toolpaths you cansee the path the tool will take, to rotate the drawing hold the ALT button and left mouse button while moving the mouse



66.7 A Box of Pi

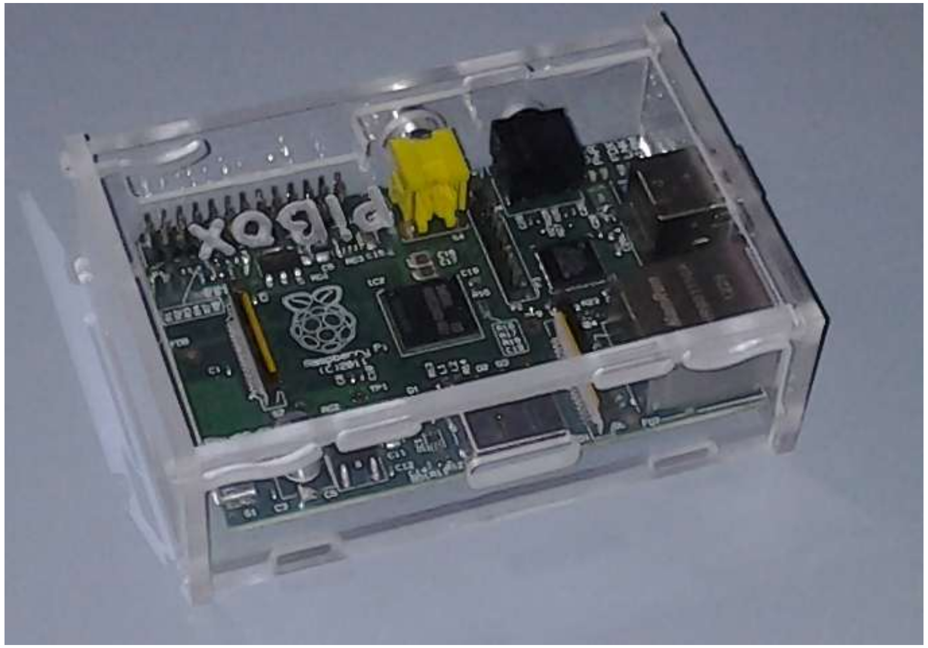
The Raspberry Pi is a single board ARM computer that comes without a case.
Here is the design for a case using Cambam.

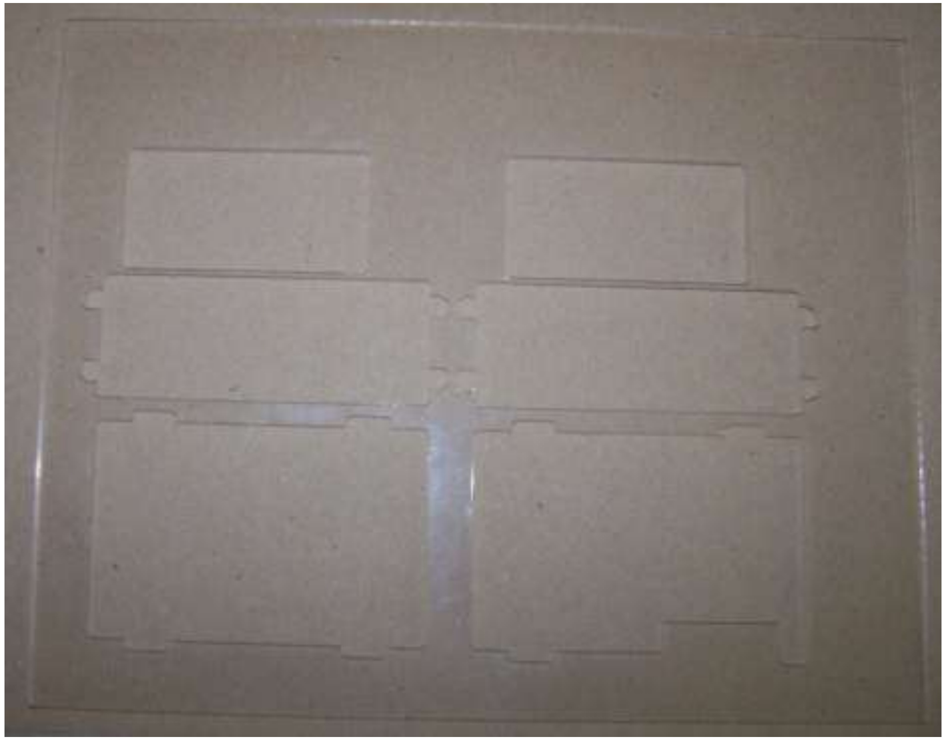


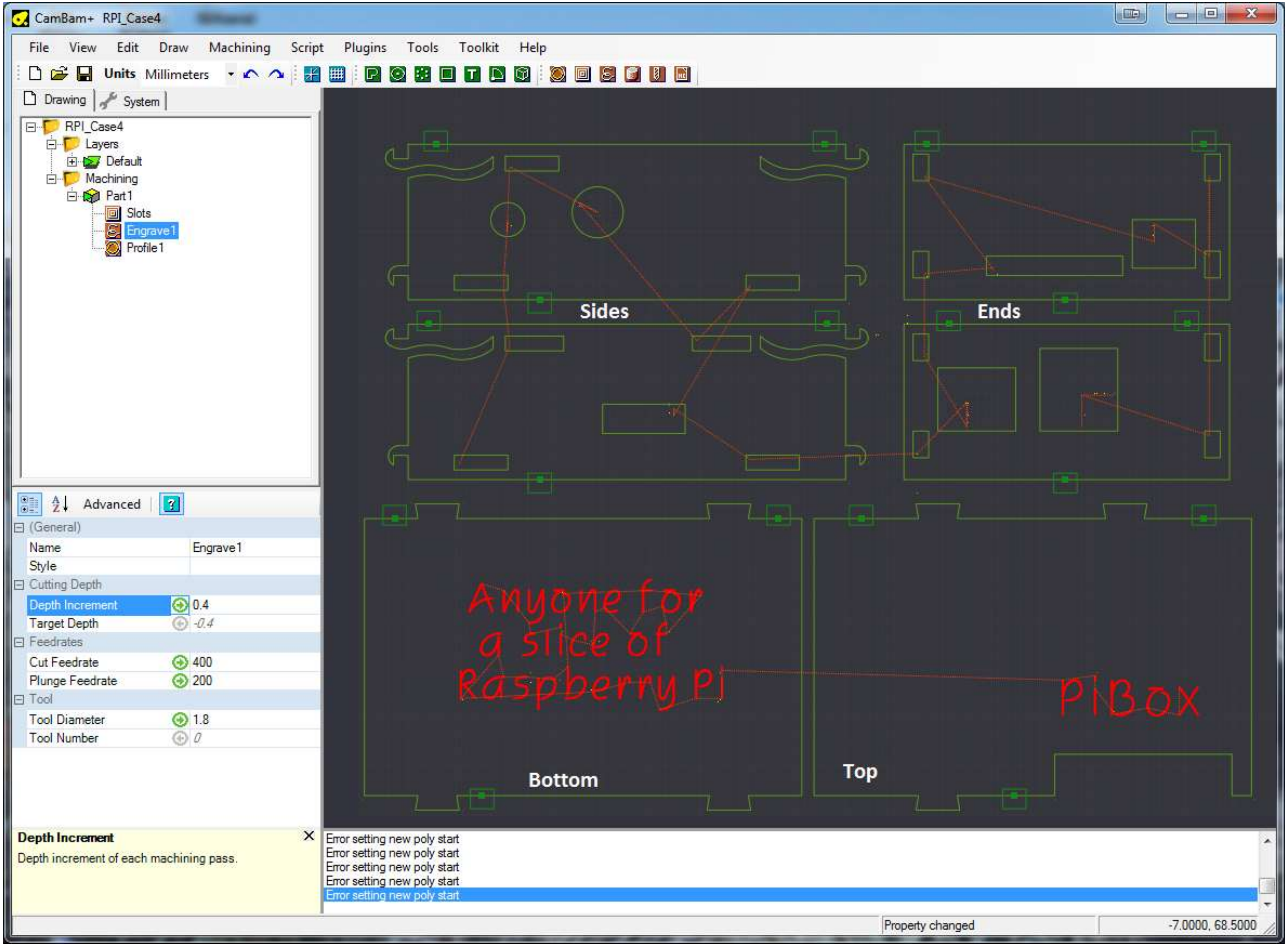

Anyone for a slice of Raspberry Pi


PiBox


pibox

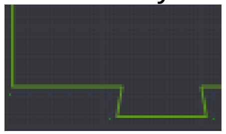

pibox





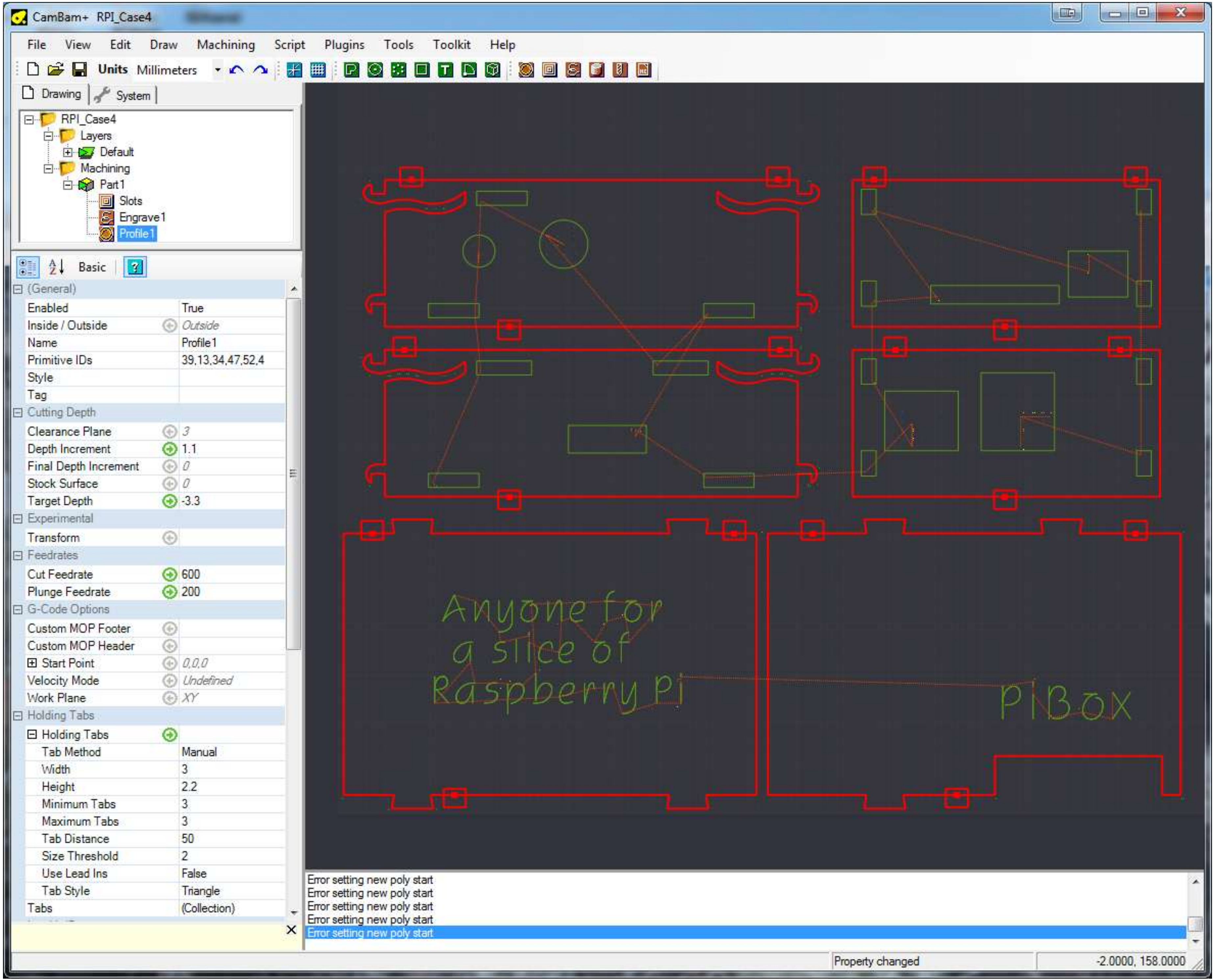
It took quite a lot of trialling to get to the finished design. There are a few design features that are worth highlighting.

The shape of the keys on the top and bottom pieces that fit into the slots on the sides are not square but cut in by 0.5mm. The reason this was done is that the mill bit is round so cannot cut a square inside corner, it always cuts a radius. This interfered with the side when the two parts were put together so rather than making the slots longer on the sides, these keys were undercut by 0.5mm



Actual key if not undercut

hole in side for key to fit into, is also not square so must be larger than the key

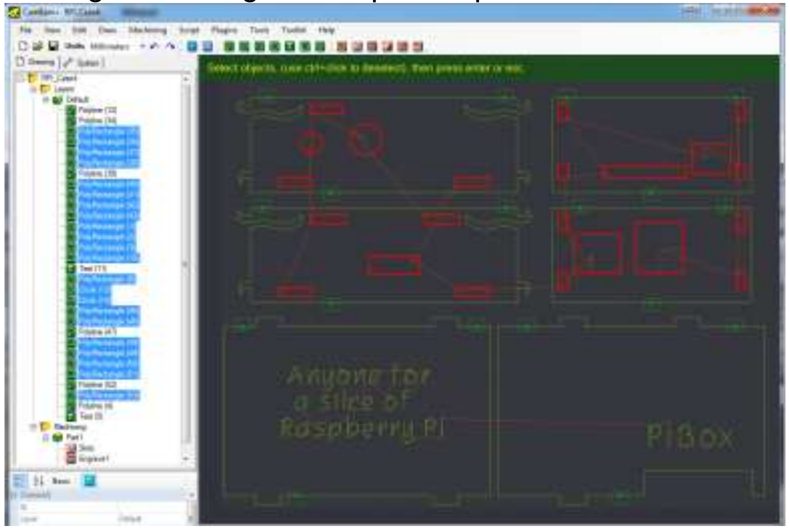


The thickness of the acrylic is 3mm however the depth of all cuts was finally settled on as 3.3mm. This allows for a little misadjustment in the height of the router bit when starting the cuts. So some mdf is placed between the acrylic and the cnc table to protect the cnc table.

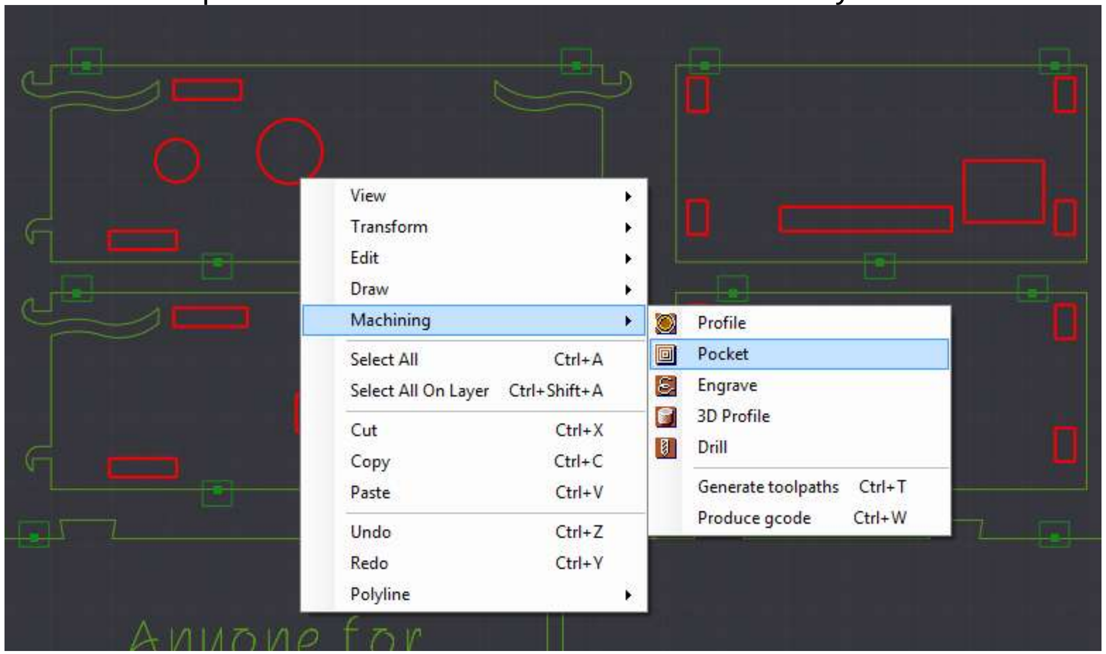
The cut depth was settled on as 1.1mm increments and so the bit does three passes over every cut.

The slots in the sides that the keys on the top and bottom fit into was made to be 3.05mm, just a little wider than the 3mm acrylic thickness.

Adding machining to multiple shapes.



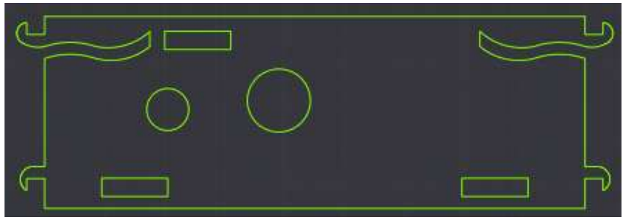
All the rectangles are selected, then right click on one of the rectangles to add machining.
In this case a pocket so that all the material will be cut away.



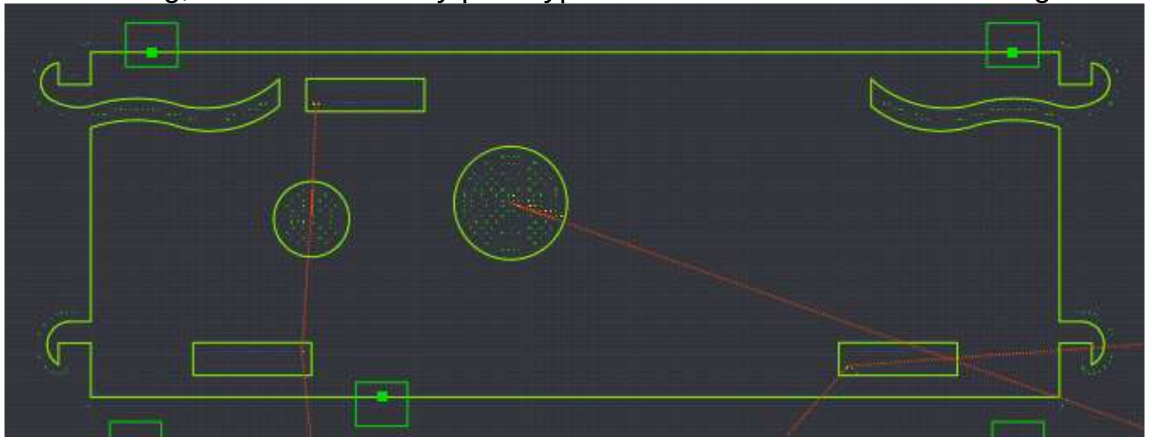
Engraving requires text to be added to a shape.

66.8 Holding Tabs

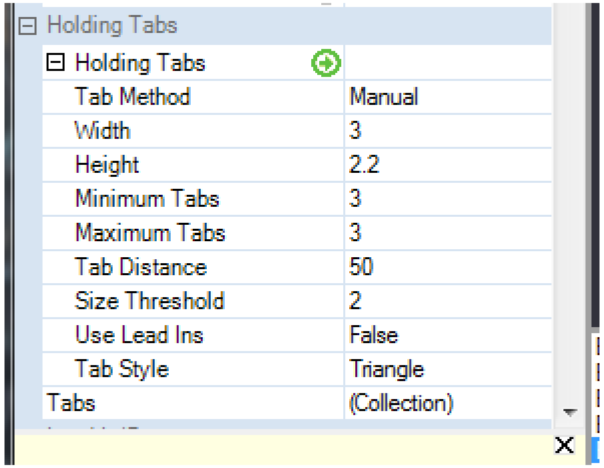
These are the small attachments between the part you are machining and the stock material.
You cannot just let the machine cut out a shape like this without supporting it, or towards the end of the curring it will move and get caught by the mill bit.



Three tabs were added to each of the 6 pieces, the tabs on these 2 side pieces were strategically placed so that the flexible pieces were well supported while cutting, in one of the early prototypes one broke off as it was moving a little with the router bit.



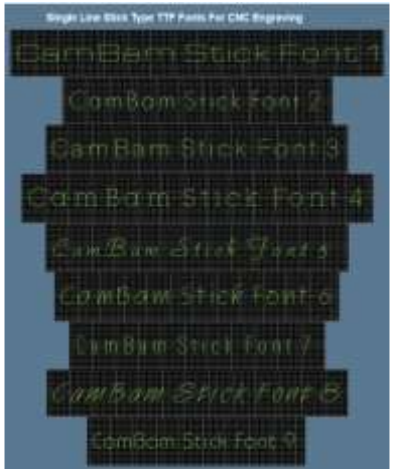
To create tabs, select the profile you want to add the tabs to. Then click Basic at the top of the properties window to get the Advanced properties settings. In the properties window expand HoldingTabs.



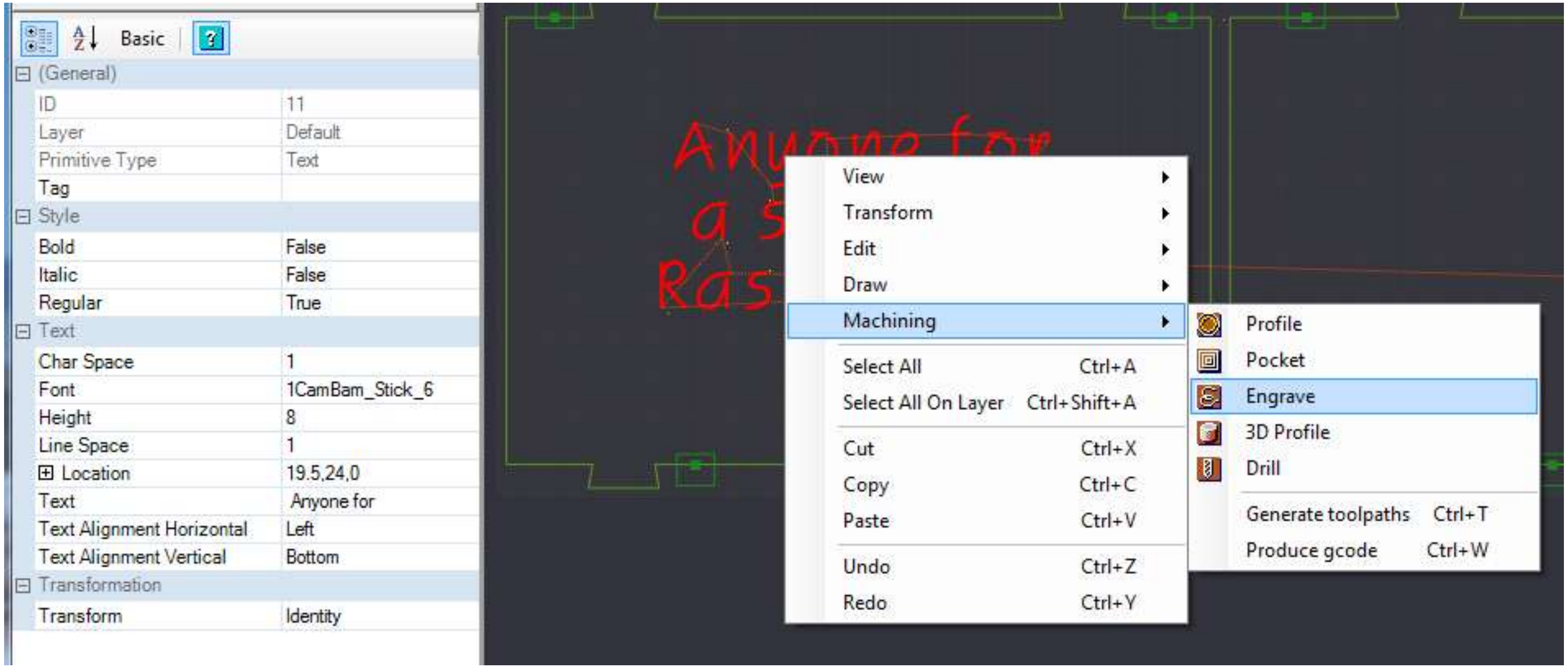
I chose a minium of 3 and a maximum of 3, height of 2.2, triangle shape.

Then change Manual to Automatic, go back to the drawing window and right click to select Machining from the context menu and then choose Generate Toolpath. The tabs will appear on the pieces, once you can see them they can be dragged around with the mouse to where youwant them to be.

66.9 Engraving



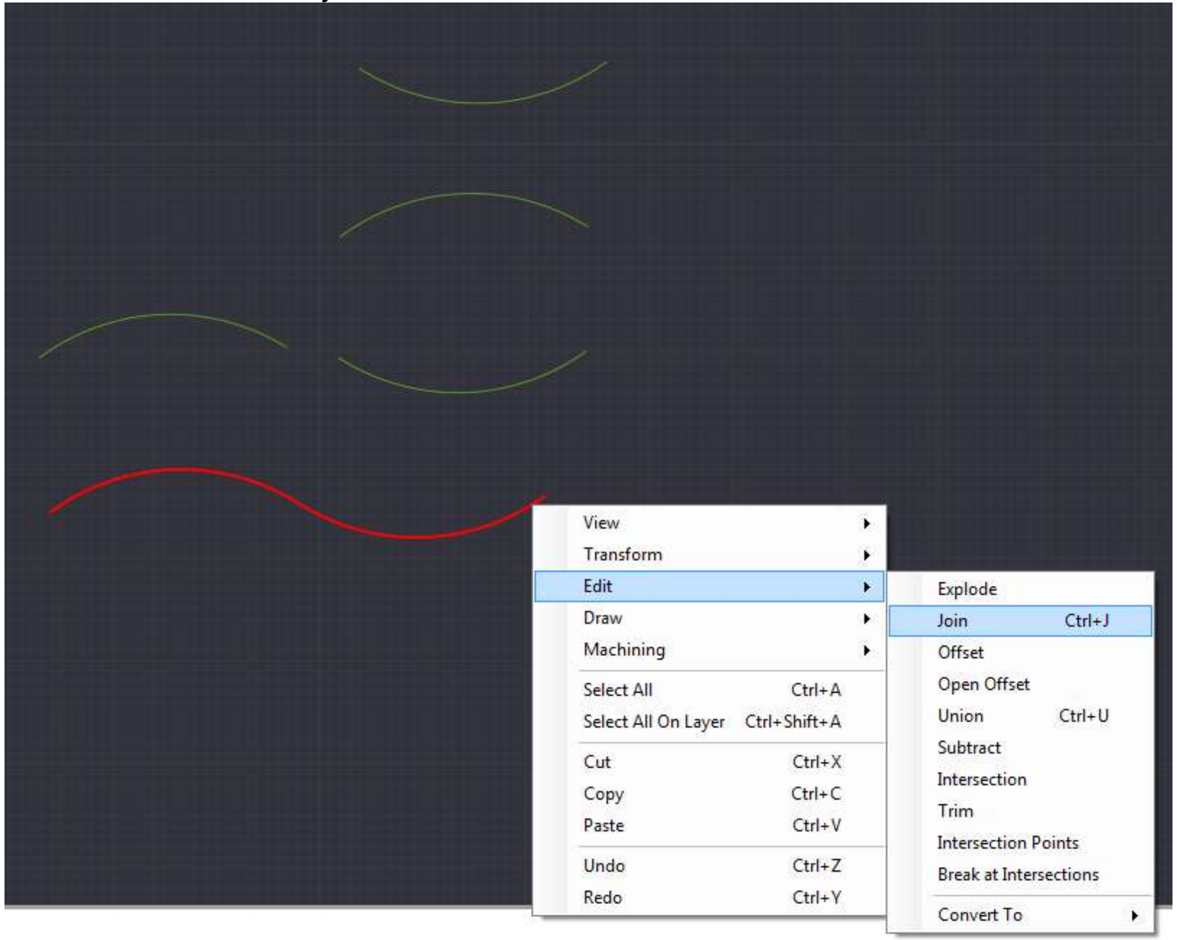
These fonts have been downloaded into the windows font directory, they are very useful as they are only cut to one thickness of the mill bit no matter what size they are made. Normal windows fonts are not.



Once text has been added it can be moved and editted and resized from the properties window.

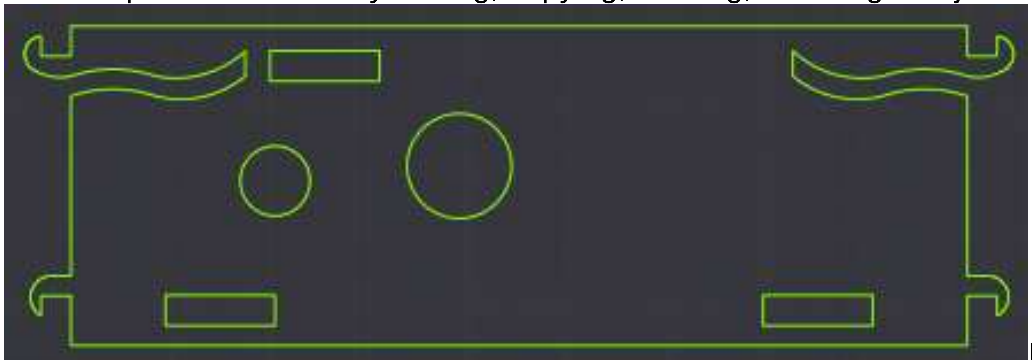
66.10 Polylines

These require you to draw freehand on the screen to make the shape you want.
In this case an arc was created, then copied and rotated, the two arcs were moved together and then joined.
When you go to join shapes Cambam will ask for a distance to show how close you want the pieces to be.
If this window is blank just enter 0.1 into it.

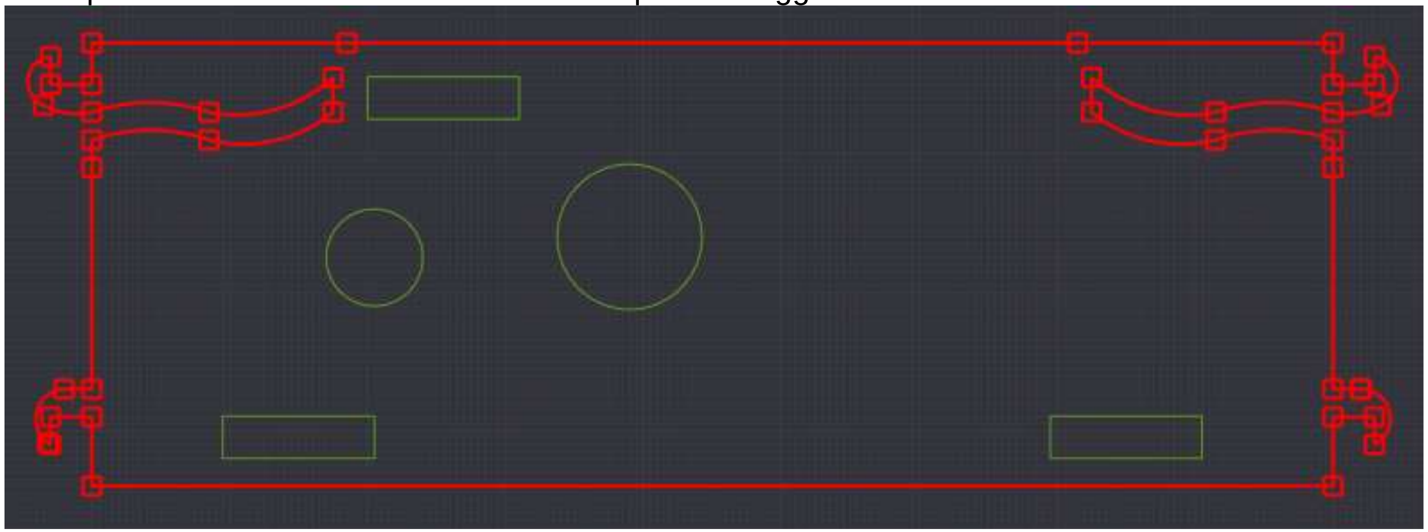


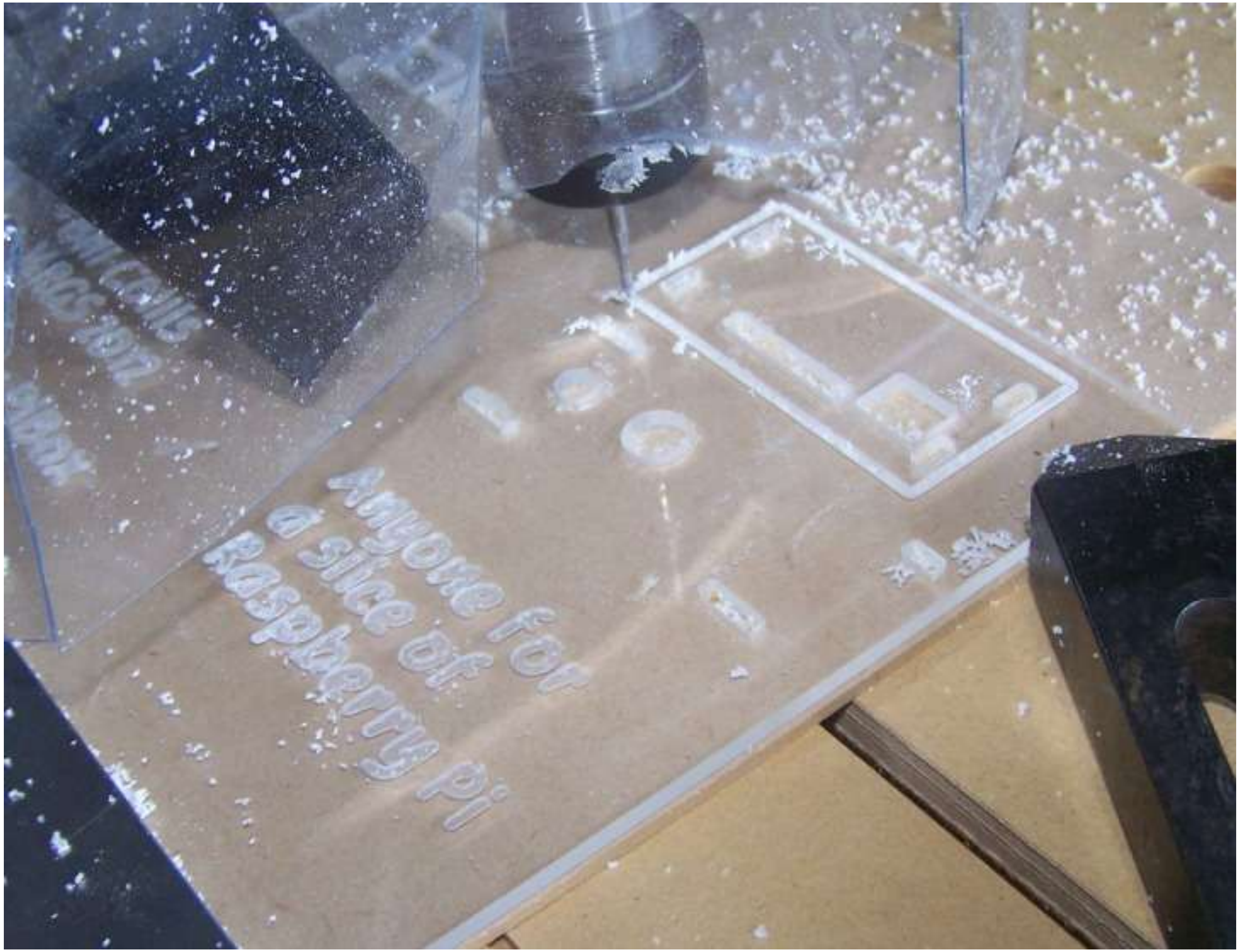
Different polylines can be broken at their intersections and then parts deleted that you don't want, then the rest joined together.

This shape was created by adding, copying, rotating, breaking and joining lots of shapes.

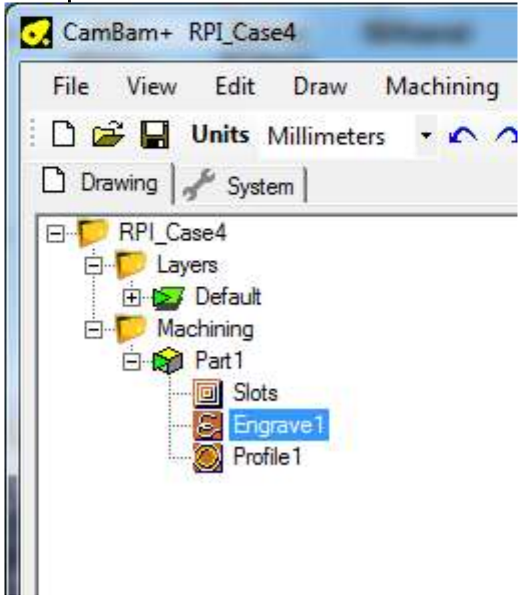

note the hooks are 3mm wide inside the same as the material thickness.

A shape can be double clicked and then the points dragged as well





Before commencing cutting, make sure that the order of the operations is correct, here the slots, then engraving then profile (outside cuts) are completed in that order.



Finally the machine was started and the cutting commences!

67 Index

The book covers both hardware interfacing and software design. It is based around the Atmel AVR range of microcontrollers and the Bascom AVR cross compiler from MCS Electronics.

This book started out as a collection of notes used in courses in electronics and microcontrollers for my secondary school students. It has been put together in one book to encourage students to see their knowledge as something that does not exist as discreet and separate during their years at school but as a complete 4 year course. It is hoped it will benefit other beginners who want a gentle paced but comprehensive introduction to practical design of projects using microcontrollers.

The language and understandings communicated through out the book reflect the use of microcontrollers as the basis for student projects to meet the requirements of their assessments in Technology Education in the New Zealand Currciulum and the Achievement Standards at Level 1, 2 & 3 of NCEA as well as New Zealand Scholarship. Examples of successful student projects are included and more will be included as time permits to increase students knowledge and understandings of requirements at these various levels.

The materials in the book are under constant review as they are in use daily with students at Mount Roskill Grammar School, Auckland,New Zealand; hence feedback on their usefulness is immediate and changes are made often.

Students who might not go on to careers in electronics should also consider this course at school as a year10 option as it develops beginning understandings and develops skills in computer programming which have significant benefit to many careers in the modern world.

As students develop their understandings of microcontrollers it is hoped that they will also develop an understanding of the power of the microcontroller to make possible the Information revolution and also to make our world become either a better or a worse place to live in. It would be great tohear from people how they used the knowledge they gain from this book to help others.

(All mistakes are mine let me know about them so I can correct them)