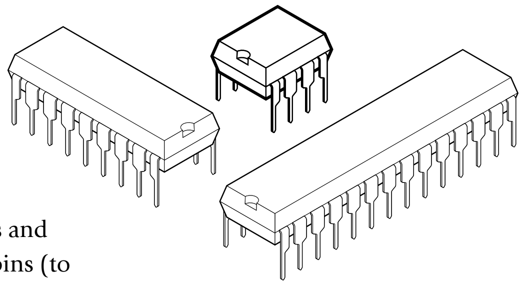
Electronic Game

What is a microcontroller?

A microcontroller is often described as a 'computer-on-a-chip'. It can be used as an 'electronic brain' to control a product, toy or machine.



The microcontroller is an integrated circuit ("chip") that contains memory (to store the program), a processor (to process and carry out the program) and input/output pins (to connect switches, sensors and output devices like motors).

Microcontrollers are purchased 'blank' and then programmed with a specific control program. This program is written on a computer and then 'downloaded' into the microcontroller chip. Once programmed the microcontroller is built into a product to make the product more intelligent and easier to use.

Early Electronic Games.

'Simon' was one of the very first portable electronic games, made by the games company MB in 1978. It was quite heavy and needed both 2 'D' 1.5V cells and a 9V PP3 battery to make it work!

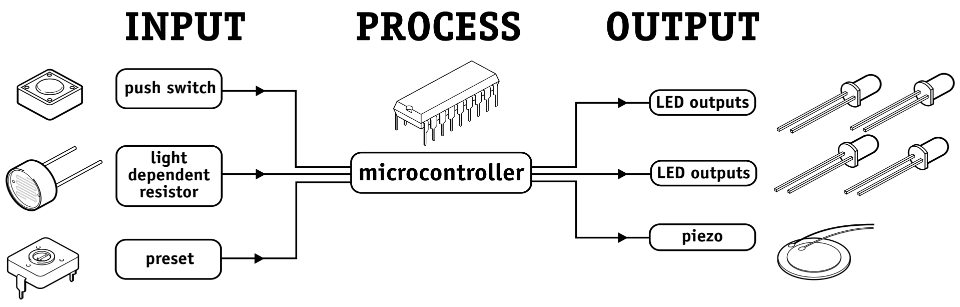
The idea behind the Simon game was quite simple. It was based on the old game 'Simon Says'. The game was made up of a big round plastic case with four coloured panels – under each panel was a switch and a light bulb. You would start the game and the microprocessor would light up one of the four panels and sound a tone. The game was then to press the panel that lit up. Simple enough! Then Simon would repeat, lighting that panel and adding another. Now your job was to press the two panels in the correct order. The number of panels would continue to get longer until you could no longer remember the sequence, which would cause Simon to issue a harsh buzz and end the game.

In 1979 MB released Super Simon which had two sets of panels so that two people could play against each other. In 1980 Pocket Simon, a smaller version of the original game, was released. There was also a Simon released with a clear casing so the electronics could be seen inside.

Simon was quite expensive as it used a complicated circuit controlled by a micro-processor, and needed big batteries to provide power for the light bulbs. It is now possible to reproduc the Simon game using 4 LEDs and a microcontroller at very low cost, and it will run off just two small 'AAA' 1.5V cells!

Block Diagrams

The electronic system for an electronic game can be drawn as a 'block diagram'.



The switches are electronic device that can detect movement and are known as 'inputs'. The microcontroller can then 'decide' how to behave and may then switch the 'output' LEDs on in different patterns or generate a sound using a piezo-transducer. If desired a Light Dependent Resistor or a temperature sensor can also be used as an input. A preset resistor may be used to set the 'speed' or 'time delay' in a game.

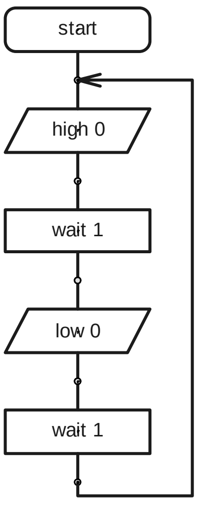
What is the PICAXE System?

The microcontrollers used in devices such as electronic games can be difficult to program, as they generally use a complicated programming language called 'assembler code', which can be quite difficult to learn.

The PICAXE system makes the microcontrollers much easier to program. The control sequence can be drawn (and simulated) on the computer as a flowchart, or written in a simpler programming language called BASIC. This makes it much easier to use the microcontroller as the complicated 'assembler code' does not need to be learnt.

A sample BASIC program and flowchart are shown here. In this case both programs do the same thing - flash a light (connected to output 0) on and off every second.

```
main:
        high 0
        wait 1
        low 0
        wait 1
        goto main
```



Supported by: UKOOA Oil and Gas for Britain

Building your own Electronic Game

Design Brief

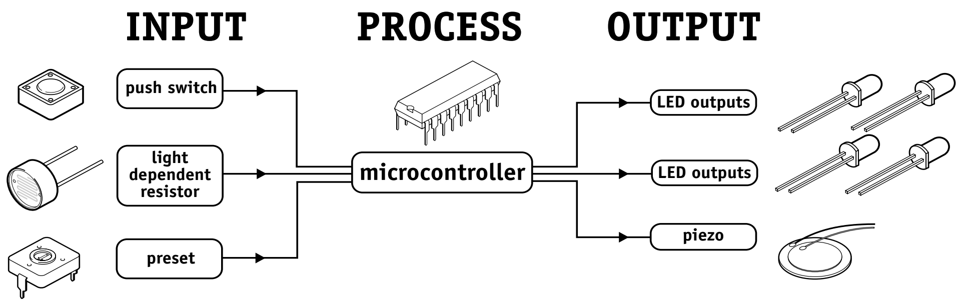
Design and make an electronic game. The game may have 4 or 5 LED indicators and a large push switch. It will also use a preset resistor to provide an adjustable input (e.g. time). You may also include other sensors such as more switches, light or temperature sensors. You may also choose to add a piezo sounder to generate sounds.

Design Specification Points

1) The design will use a PICAXE-18 microcontroller as it's controller.
2) The design will include LED outputs, a switch input and a preset resistor input.
3) The design will be programmed with a game you must design.

Block Diagram

The block diagram for your electronic game may look like this:



Personalising Your Electronic Game

Here are some things to think about:

1) How is your game going to function? It could be a simple timing or 'dice' device used as part of a bigger game, or a complete game by itself.

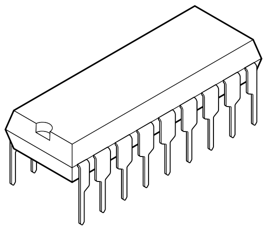
2) What colour and size of LEDs are you going to use? The most common LEDs are red, but many other sizes and colours are available (e.g. blue).

3) How will you activate the game? Normally a large push switch on the case is used, but you could also connect other switches on leads. You could also choose to use many other types of sensor, for example, you could have a light sensor (LDR) that can detect changes in light level when you put your hand over it.

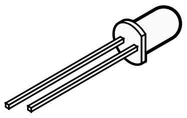
Electronic Components

The main electronic components you may need for your electronic game are shown here. The next few pages describe each of these components in more detail, and also provide some programming ideas that may be useful when you are later programming your electronic game patterns.

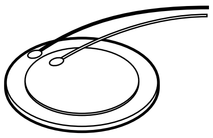
PICAXE-18 microcontroller



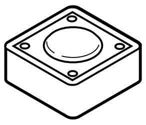
light emitting diode (LED)



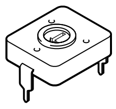
piezo sounder



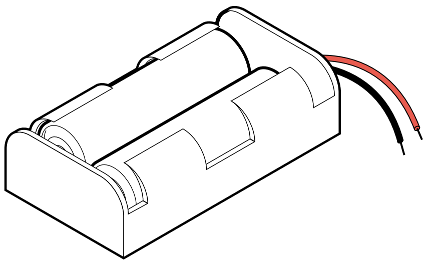
push switch



preset resistor

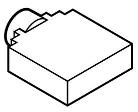


2 x AAA battery box

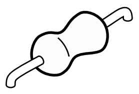


and you will also need

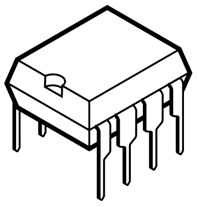
picaxe download socket



resistors



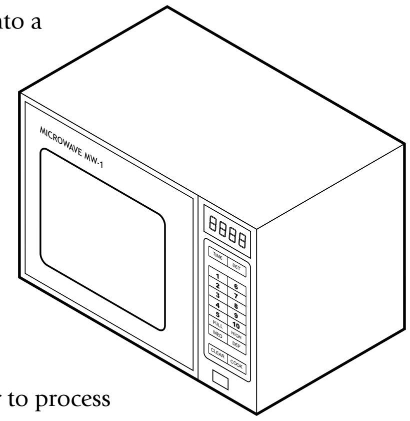
Section 2 - Electronic Components
Microcontrollers



What is a microcontroller?

A microcontroller is often described as a 'computer-on-a-chip'. It is an integrated circuit that contains memory, processing units, and input/ output circuitry in a single unit.

Microcontrollers are purchased 'blank' and then programmed with a specific control program. Once programmed the microcontroller is built into a product to make the product more intelligent and easier to use.



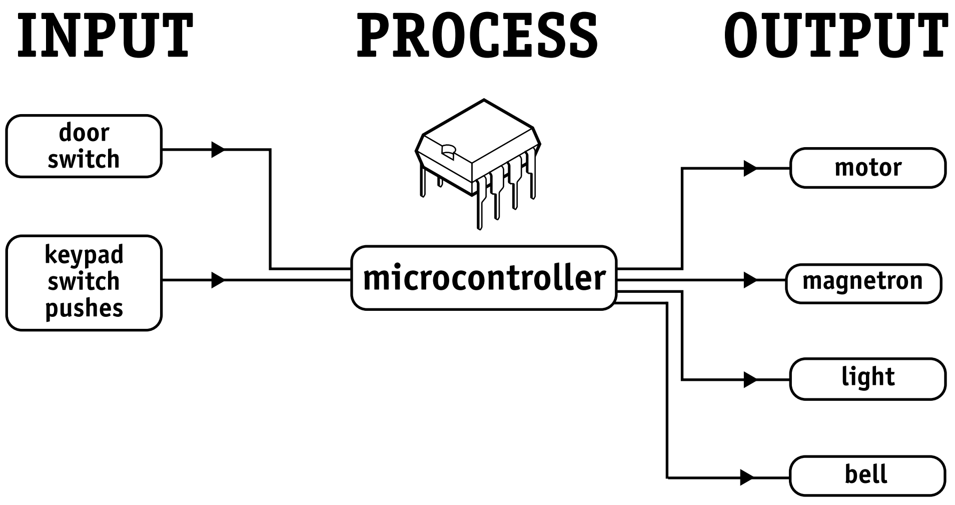
Where are microcontrollers used?

Applications that use microcontrollers include household appliances, alarm systems, medical equipment, vehicle subsystems, and electronic instrumentation. Some modern cars contain over thirty microcontrollers - used in a range of subsystems from engine management to remote locking!

As an example, a microwave oven may use a single microcontroller to process information from the keypad, display user information on the seven segment display, and control the output devices (turntable motor, light, bell and magnetron).

How are microcontrollers used?

Microcontrollers are used as the 'brain' in electronic circuits. These electronic circuits are often drawn visually as a 'block diagram'. For instance a simplified block diagram for the microwave above could be drawn like this:



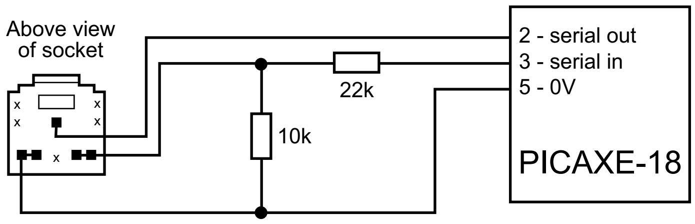
The program for the microcontroller is developed (and tested) on the computer and then downloaded into the microcontroller. Once the program is in the microcontroller it starts to 'run' and carries out the instructions.

How are programs written?

Programs are drawn as flowcharts or typed as 'BASIC' listings. This is is explained in the programming section (section 3) later in this booklet.

How is the program transferred to the microcontroller?

The PICAXE-18 microcontroller is programmed by connecting a cable from the serial port at the back of the computer to a socket on the printed circuit board (PCB) beside the microcontroller. This socket (which looks like a headphone socket as found on a portable CD player) connects to two legs of the microcontroller and to 0V from the battery. This allows the computer and the microcontroller to 'talk' to allow a new



program to be downloaded into the microcontroller's memory.

The socket and interfacing circuit is included on every PCB designed to be used with the PICAXE-18 microcontroller. This enables the PICAXE microcontroller to be re-programmed without removing the chip from the PCB - simply connect the cable whenever you want to download a new program!

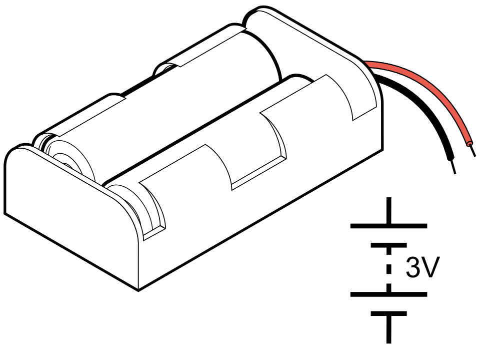
The circuit diagrams of PICAXE circuits often do not include the components above to make it easier to understand the input/output connections. However the two resistors and the socket are always built onto every PICAXE project board!

Note:
Most modern computers have two serial ports, normally labelled COM1 and COM2. The Programming Editor software used to create the programs must be configured for the correct serial port – select **View>Options>Serial Port** to select the correct serial port for your machine.

If you are using a new laptop computer it may only have the newer 'USB' type connector. In this case you must buy a USB to serial adapter to use the PICAXE system. These are available from most high street computer stores or online from www.tech-supplies.co.uk (part USB010).

Batteries

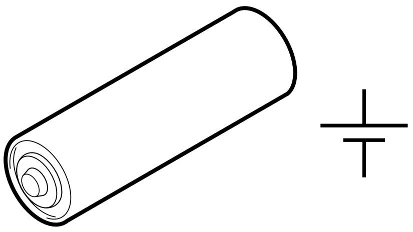
What is a battery?



A battery is a self-contained source of electronic energy. It is a portable power supply.
Batteries contain chemicals that store energy. When connected into a circuit this chemical energy is converted to electrical energy that can then power the circuit.

Which battery size should I use?

Batteries come in all sorts of types and sizes. Most battery packs are made up of a number of 'cells', and each cell provides about 1.5V. Therefore 4 cells will generate a 6V battery and 3 cells a 4.5V battery.



As a general rule, the larger the battery the longer it will last (as it contains more chemicals and so will be able to convert more energy). A higher voltage battery does not last longer than a lower voltage battery Therefore a 6V battery pack made up of 4 AA cells will last much longer than a 9V PP3 battery, as it contains a larger total amount of chemical energy as it is physically larger. Therefore items that require more power to work (e.g. a CD walkman which contains a motor and laser to read the CD's) will always use AA cells rather than PP3 batteries.

Microcontrollers generally require 3 to 6V to work, and so it is better to use a battery pack made up of two, three or four AAA or AA size cells. Never use a 9V PP3 battery directly as the 9V supply will damage the microcontroller. If you want to use a 9V PP3 battery you must use it together with a 78L05 5V regulator.

Which battery type should I use?

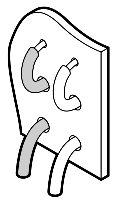
Different batteries are made of different chemicals. Zinc-carbon batteries are the cheapest, and are quite suitable for many microcontroller circuits. Alkaline batteries are more expensive, but will last much longer when driving devices like motors that require larger currents. Lithium batteries are much more expensive but have a long life, and so are commonly used in computer circuits to provide a clock backup.

Rechargeable batteries can be recharged when they 'run-down'. They are generally made up of nickel and cadmium (Ni-cad) or nickel metal hydroxide (NiMH) chemicals.

Safety!

Never 'short circuit' any battery. Alkaline and rechargeable batteries can provide a very large current, and can get so hot that they will actually melt the battery box if you short circuit them! Always make sure you connect the battery around the correct way (red positive (V+) and black negative (0V or ground). The microcontroller chip will get hot and be damaged if the battery is connected the wrong way around.

Using battery snaps.



Battery packs are often connected to electronic printed circuit boards by battery snaps or wires. Always ensure you get the red and black wires the correct way around. It is also useful to thread the wires through holes on the board before soldering it in place - this provides a much stronger joint that is less likely to snap off.

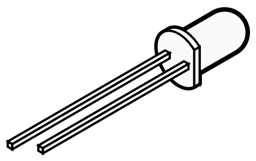
Never accidentally connect a 9V PP3 battery to the battery snap - this will damage the microcontroller, which only works between 3 and 6V.

Soldering to battery boxes.

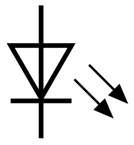
Some small battery boxes require wires to be soldered to metal contacts on the battery box. In this case you must be very careful not to overheat the metal contact. If the contacts gets very hot they will melt the plastic and fall off. A good way of stopping this happening is to ask a friend to hold the metal contact with a pair of small pliers. The pliers will act as a 'heat-sink' and help stop the plastic melting.

Light Emitting Diode (LED)

What is an LED?



A Light Emitting Diode (LED) is an electronic component that gives out light when current passes through it. An LED is a special type of diode. A diode is a component that only allows current to flow in one direction. Therefore when using a diode, it must always be connected the correct way around.



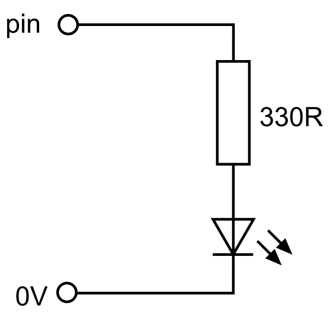
The positive (anode) leg of an LED is longer than the negative (cathode) leg (shown by the bar on the symbol). The negative leg also has a flat edge on the plastic casing of the LED.

What are LEDs used for?

LEDs are mainly used as indicator lights. Red and green LEDs are commonly used on electronic appliances like televisions to show if they are switched on or in 'standby' mode. LEDs are available in many different colours, including red, yellow, green and blue. Special 'ultrabright' LEDs are used in safety warning devices such as the 'flashing lights' used on bicycles. Infra-red LEDs produce infra-red light that cannot be seen by the human eye but can be used in devices such as video remote-controls.

Using LEDs.

LEDs only require a small amount of current to work, which makes them much more efficient than bulbs (this means, for instance, that if powered by batteries the LEDs will light for a much longer time than a bulb would). If too much current is passed through an LED it will be damaged, and so LEDs are normally used together with a 'series' resistor that protects the LED from too much current.



The value of the resistor required depends on the battery voltage used. For a 4.5V battery pack a 330R resistor can be used, and for a 3V battery pack a 120R resistor is appropriate.

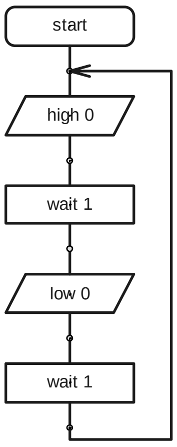
Connecting the LED to a microcontroller.

Because the LED only requires a small amount of current to operate, it can be directly connected between the microcontroller output pin and 0V (with the series protection resistor). Two LEDs can be driven from the same pin if you use a resistor for each LED.

Testing the LED connection.

After connecting the LED it can be tested by a simple program like this:

```
main:
        high 0
        wait 1
        low 0
        wait 1
        goto main
```



This program would switch the LED (connected to output pin 0) on and off every second. If the LED does not work check:

1) the LED is connected the correct way around
2) the correct resistor is used
3) the correct output pin number is being used in the program
4) all the solder joints are good

This program flashes the LED connected to output pin 0 on and off 15 times using a BASIC programming technique called a for...next loop (this technique cannot be used with flowcharts). The number of times the code has been repeated is stored in the memory of the PICAXE chip using a 'variable' called b1 (the PICAXE contains 14 variables labelled b0 to b13). A variable is a 'number storage position' inside the microcontroller than the microcontroller can use to store numbers as the program is carried out.

```
main:   for b1 = 1 to 15        ' start a for...next loop
                high 0          ' switch pin 0 high
                pause 500       ' wait for 0.5 second
                low 0           ' switch pin 0 low
                pause 500       ' wait for 0.5 second
        next b1                 ' end of for...next loop

        end                     ' end program
```

Switching more than one LED at once.

Sometimes it is useful to switch more than one LED on or off at the same time. This saves time when lots of high and low commands would have to be used together.

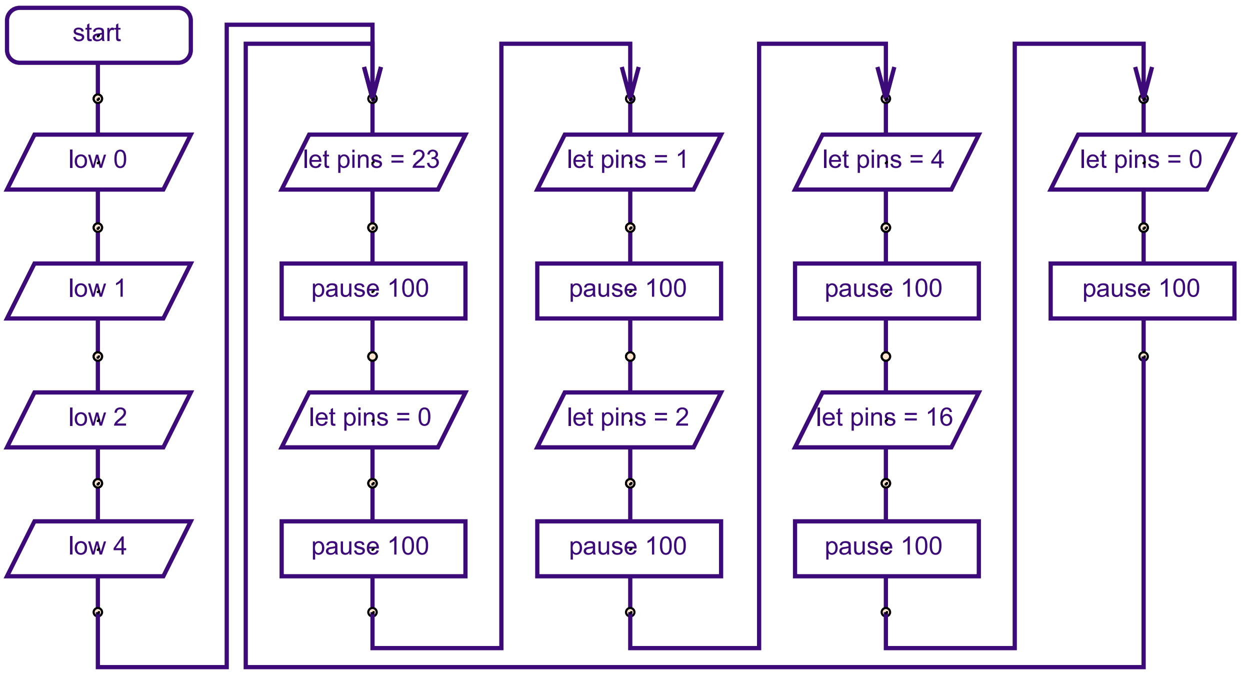
The command that does this is called let pins =

Pin	5	4	2	1	0
Value	32	16	4	2	1

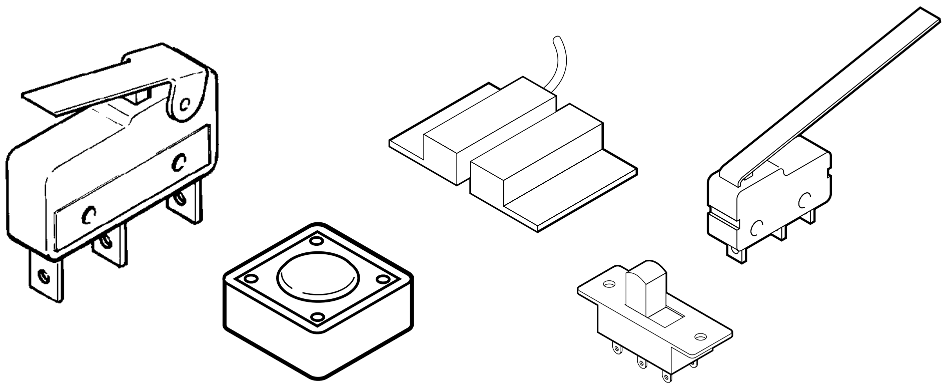
After the equals sign a number is used. Each output pin is given a value, and the number used in the program is the sum of these values.

Therefore this program switches all of the outputs on, and then all off, and then one on at a time, in sequence.

```
main:
        let pins = 23   ' switch first four LEDs on (16+4+2+1)
        pause 100       ' wait for 0.1 second
        let pins = 0    ' switch all off
        pause 100       ' wait for 0.1 second
        let pins = 1    ' switch pin 0 on, others off
        pause 100       ' wait for 0.1 second
        let pins = 2    ' switch pin 1 on, others off
        pause 100       ' wait for 0.1 second
        let pins = 4    ' switch pin 2 on, others off
        pause 100       ' wait for 0.1 second
        let pins = 16   ' switch pin 4 on, others off
        pause 100       ' wait for 0.1 second
        let pins = 0    ' switch all off
        pause 100       ' wait for 0.1 second
        goto main       ' loop back to start
```

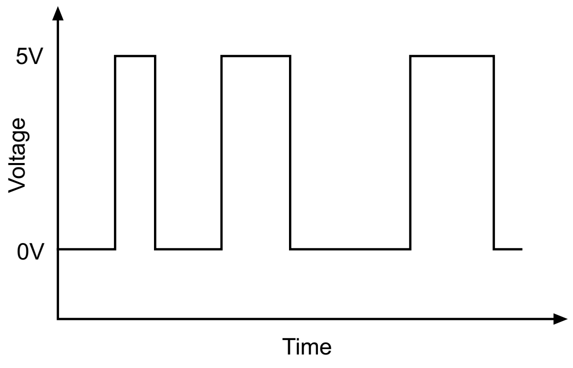


Digital Sensors (Switches)



What are switches?

A digital sensor is a simple 'switch' type sensor that can only be 'on' or 'off'. If a graph is drawn of the on-off signals as the switch is pushed it will look like this:



Switches are electronic components that detect movement. There are a large number of different types of switches e.g:

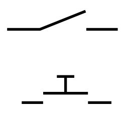
push switches that detect a momentary 'push'
micro-switches with long levers that detect small movements
tilt-switches that detect jolting
reed-switches that detect a magnet being moved

What are switches used for?

Push switches are commonly used on device like keypads. Micro-switches are used in burglar alarms to detect if the cover is removed from the alarm box. Reed switches are used to detect doors and windows being opened and tilt switches are often used to detect movement in devices such as toys, hair-dryers and tool-box alarms.

Switch Symbols.

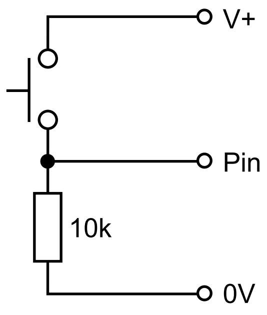
The symbols for a slide switch
and a push switch are shown here.



Using switches

A switch is used with a resistor as shown in the diagram. The value of the resistor is not that important, but a 10k resistor is often used. When the switch is 'open' the 10k resistor connects the microcontroller input pin down to 0V, which gives an off (logic level 0) 0V signal to the microcontroller input pin.

When the switch is activated, the input pin is connected to the positive battery supply (V+). This provides an on (logic level 1) signal to the microcontroller.



Testing the switch

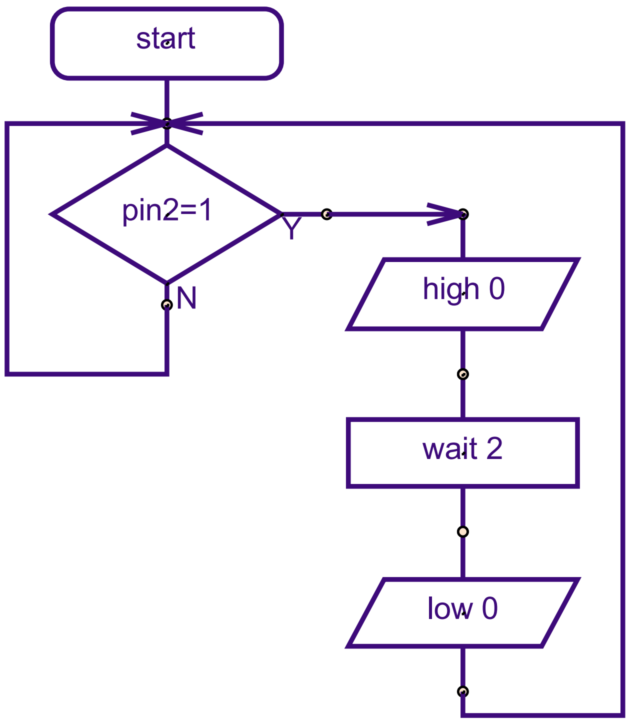
After connecting the switch it can be tested by a simple program like this. This program will switch an output on and off according to if the switch is pushed or not.

```
main:                                  ‘ make a label called ‘main’
        if input2 is on then flash     ‘ jump if the input is on
        goto main                      ‘ else loop back around

flash:                                 ‘ make a label called ‘flash’
        high 0                         ‘ switch output 0 on
        wait 2                         ‘ wait 2 seconds
        low 0                          ‘ switch output 0 off
        goto main                      ‘ jump back to start
```

In this program the first three lines make up a continuous loop. If the input is off the program just loops around time and time again.

If the switch is then pushed the program jumps to the label called **‘flash’**. The program then flashes output 0 on for two seconds before returning to the main loop.

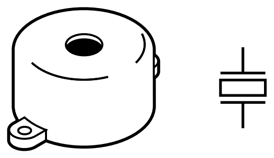
Note carefully the spelling in the **if…then** line – **input2** is all one word (without a space). You can use the word **pin2** or **input2** to mean the same thing. Note also that only the label is placed after the command **then** – no other words apart from a label are allowed at this point.



Buzzers and Piezo-Transducers

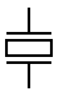
What is a piezo transducer?

A piezo transducer is a low-cost 'mini-speaker' that can used to make sounds. The sound that the piezo makes can be changed by altering the electronic signals provided by the microcontroller.



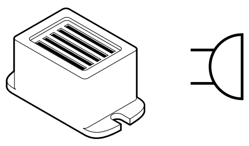
Where are piezos used for?

Piezos are used in many different consumer goods to provide 'feedback' to the user. A good example is a vending machine which will 'beep' whenever a keypad switch is pressed to select a drink or snack. The 'beep' provides the user with feedback to tell them their switch push has been successful. Uncased piezos are also often used in musical birthday cards to play a tune when the card is opened.



What is the difference between a piezo and a buzzer?

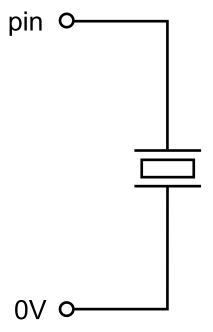
A buzzer contains a small electronic circuit that generates the electronic signal needed to make a noise. Therefore when a buzzer is connected to a battery it will always make the same sound. A piezo does not contain this circuit, and so therefore needs an external signal. This signal can be supplied by the output pin of a microcontroller. A piezo also requires less current to operate and so will last longer in battery powered circuits.

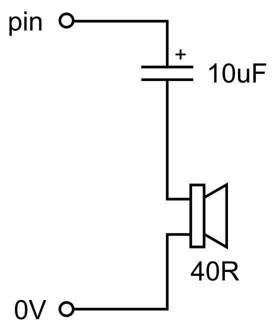


Using piezos.

A piezo is very simple to connect. Simply connect the red wire to the microcontroller output pin and the black wire to 0V (ground).

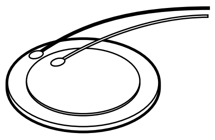
Note that the cheapest piezos do not have a plastic casing to them. In this case it is necessary to mount the piezo on a piece of board (with a sticky pad) to create a noise that can be heard. The board acts as a 'sound-box' to amplify the sound made by the piezo. Make sure the sticky pad is stuck on the correct side of the piezo (the brass side without the wires!).

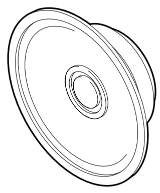




Making More Noise.

Some times you might want to make a louder noise. In this case it is possible to use a speaker instead of the piezo. When using a speaker it is also necessary to use a capacitor (e.g. 10uF electrolytic capacitor) to prevent damage to the microcontroller.
Remember that, like the piezo, a speaker only works correctly when mounted in a 'sound-box'.

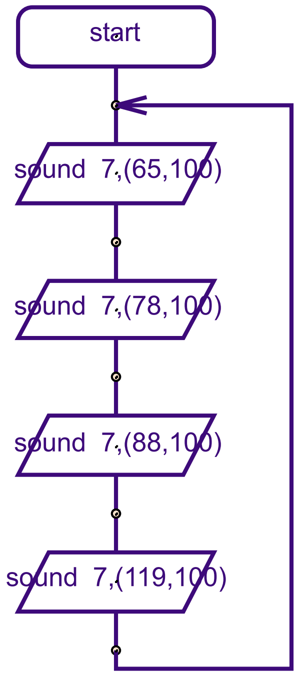




Testing the piezo connection.

After connecting the piezo it can be tested by a simple program like this:

```
main:
        sound 7,(65,100)
        sound 7,(78,100)
        sound 7,(88,100)
        sound 7,(119,100)
        goto main
```



This program would make the piezo (connected to output pin 7) make 4 different sounds (value 65, 78, 88 and 119). If the piezo does not work check:

1) the sound value is between 0 and 127
2) the correct output pin number is being used in the program
3) all the solder joints are good

With the sound command, the first number provides the pin number (on projects output7 is often used). The next number is the tone, followed by the duration. The higher the tone, the higher the pitch of the sound (note that some sounders cannot produce very high tones and so number greater than 127 may not be heard).

When using multiple sounds you may also include them all on the same line e.g.

```
        sound 7,(65,100,78,100,88,100,119,100)
```

The following table shows some common notes with the appropriate sound value:
A(49), As(51), B(54), C(57), Cs(61), D(65), Ds(71), E(78), F(88), Fs(101), G(119)

The following BASIC program uses a for…next loop to produce 120 different sounds, using variable b1 to store the sound value.

```
main:
        for b1 = 1 to 120       ‘ start a for...next loop
           sound 7,(b1,50)      ‘ make a sound, freq value from b1
        next b1                 ‘ next loop
        end
```

The number stored in variable b1 id increased by 1 in every loop (1-2-3 etc.) Therefore by using the variable name b1 in the tone position, the note can be changed on each loop.

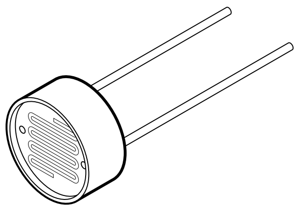
The following program does the same task but backwards (counting down instead of up).

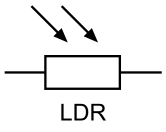
```
main:
        for b1 = 120 to 1 step -1 ‘ start a for...next loop
           sound 7,(b1,50)         ‘ make a sound, freq value from b1
        next b1                    ‘ next loop
        end
```

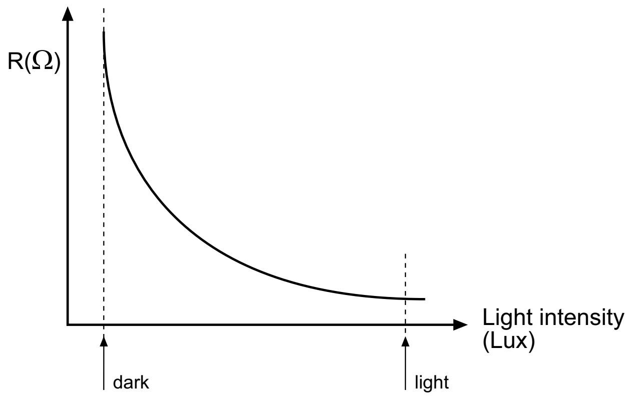
Light Dependent Resistor (LDR)

What is an LDR?

A Light Dependent Resistor (LDR) is special type of resistor that reacts to changes in light level. The resistance of the LDR changes as different amounts of light fall on the top 'window' of the device. This allows electronic circuits to measure changes in light level.



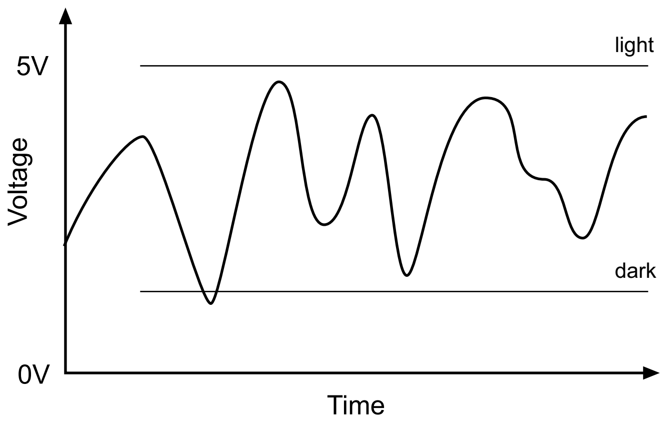




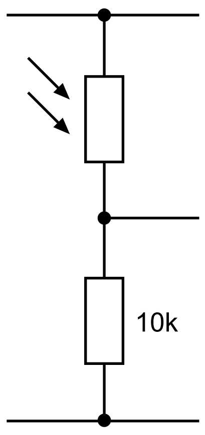
What are LDRs used for?

LDRs are used in automatic street lamps to switch them on at night and off during the day. They are also used within many alarm and toys to measure light levels.

The LDR is a type of **analogue sensor**. An analogue sensor measures a continuous signal such as light, temperature or position (rather than a digital on-off signal like a switch). The analogue sensor provides a varying voltage signal. This voltage signal can be represented by a number in the range 0 and 255 (e.g. very dark = 0, bright light = 255).



Using LDRs.



A LDR can be used in two ways. The simplest way to use an LDR is as a simple on-off ("digital") switch - when the light level is above a certain value (called the 'threshold value') the LDR will provide an on signal, when the light level is below a certain value the LDR will provide an off signal.

In this case the LDR is used in a potential divider with a standard resistor. The value of the standard resistor sets the 'threshold value'. For miniature LDRs a suitable value may be 1k or 10k, for larger ORP12 type LDRs 10k is more appropriate. If desired the fixed resistor can be replaced by a variable resistor so that the threshold value can be 'tuned' to different light values.

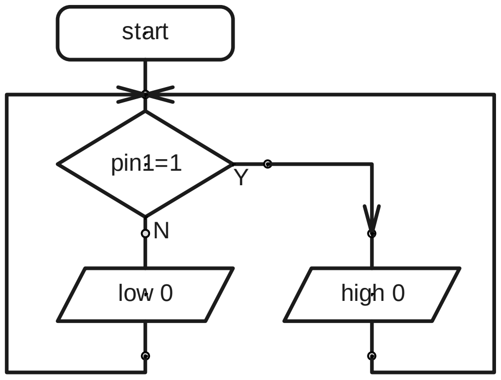
A more versatile way of using the LDR is to measure a number of different light values, so that decisions can be made at varying light levels rather than just one fixed threshold value. A varying value is known as an 'analogue' value, rather than a digital 'on-off' value. To measure analogue values the microcontroller must contain an 'analogue to digital converter (ADC)' and the programming software must support use of this ADC. Most microcontrollers only contain ADC on certain input pins, and so the input pin connection must be carefully selected. With the 18 pin microcontroller pins 0,1,2 can be used.

The electronic circuit for using the ADC is a potential divider identical to the circuit above. The analogue 'measurement' is carried out within the microcontroller itself.

Testing the LDR (digital)

After connecting the LDR it can be tested as a digital switch by a simple program like this:

```
main:
        if input1 is on then dohigh
        low 0
        goto main

dohigh:
        high 0
        goto main
```

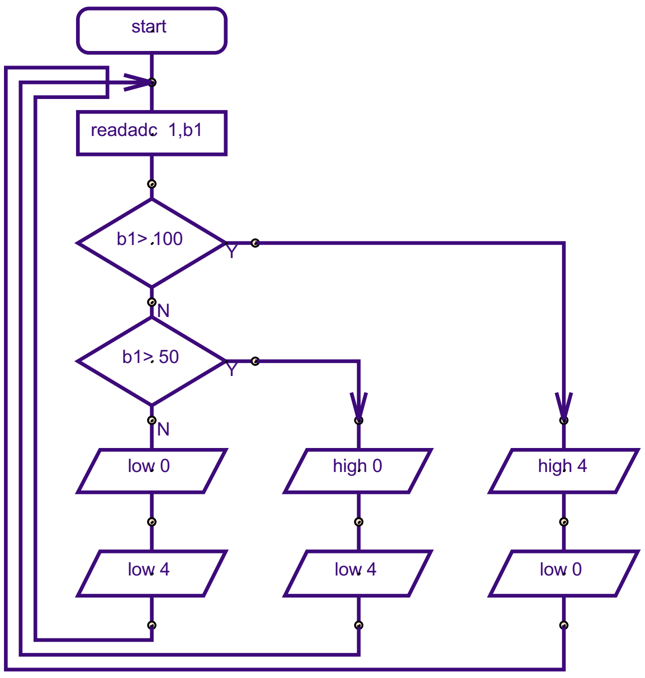


This program will switch output 0 on and off according to the light level.

Testing the LDR (analogue)

After connecting the LDR it can be tested as an analogue sensor by a program like this:

```
main:
        readadc 1,b1
        if b1 > 100 then do4
        if b1 > 50 then do0
        low 0
        low 4
        goto main

do4:
        high 4
        low 0
        goto main
do0:
        high 0
        low 4
        goto main
```

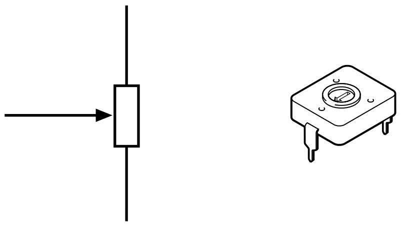


The 'readadc' command is used to read the analogue value (a number between 0 and 255) into variable b1. Once the number is in variable b1 it can be tested to see if it is greater than 100 or greater than 50. If it is greater than 100 output 4 is switched on, if it is between 50 and 100 output 0 is switched on, and if it is less than 50 both outputs are switched off.

Preset Resistor

What is a preset resistor?

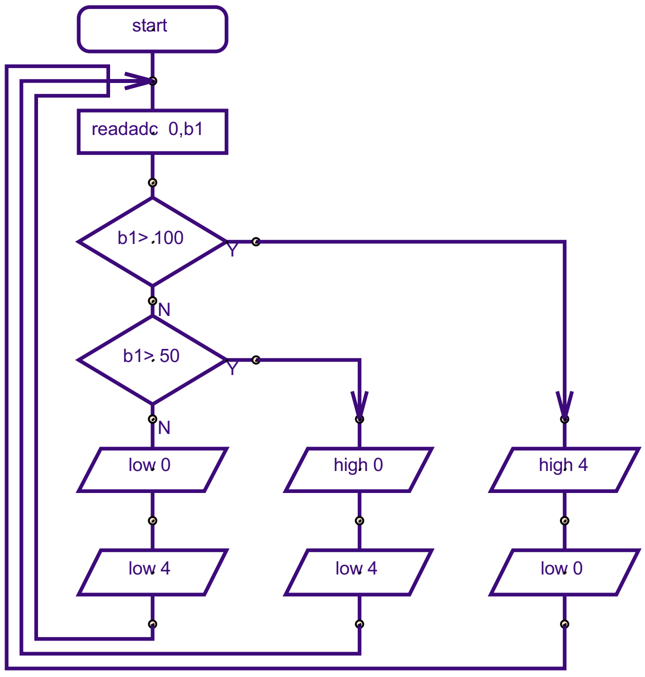
A preset resistor is a resistor that has a slot that can be twisted to alter the value of the resistance of the resistor. This enables adjustment to circuits e.g. to alter the time delay within a game.



Testing the preset resistor

The preset resistor is an analogue sensor like the lDR, and so can be tested by the same program as follows:

```
main:
        readadc 0,b1
        if b1 > 100 then do4
        if b1 > 50 then do0
        low 0
        low 4
        goto main

do4:
        high 4
        low 0
        goto main
do0:
        high 0
        low 4
        goto main
```

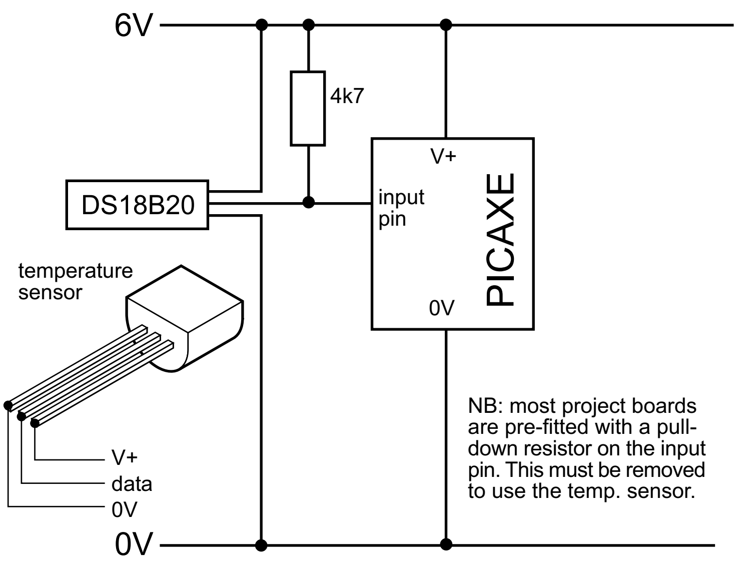


The 'readadc' command is used to read the analogue value (a number between 0 and 255) into variable b1. Once the number is in variable b1 it can be tested to see if it is greater than 100 or greater than 50. If it is greater than 100 output 4 is switched on, if it is between 50 and 100 output 0 is switched on, and if it is less than 50 both outputs are switched off.

DS18B20 Digital Temperature Sensor

What is a digital temperature sensor?

A DS18B20 is special type of digital temperature sensor that reacts to changes in temperature level. The temperature sensor can be connected to the PICAXE-18A (note the enhanced 18A part is required) to enable accurate temperature readings to be made. This allows the electronic circuit to measure changes in temperature.


6V
4k7
V+
DS18B20
input
pin
PICAXE
temperature
sensor
0V
NB: most project boards are pre-fitted with a pull-down resistor on the input pin. This must be removed to use the temp. sensor.
V+
data
0V
0V

Testing the digital temperature sensor

After connecting the DS18B20 it can be tested by a simple program like this:

```
main:
       readtemp 6,b1   ' (presuming connection to input 6)
       debug b1
       pause 1000
       goto main
```

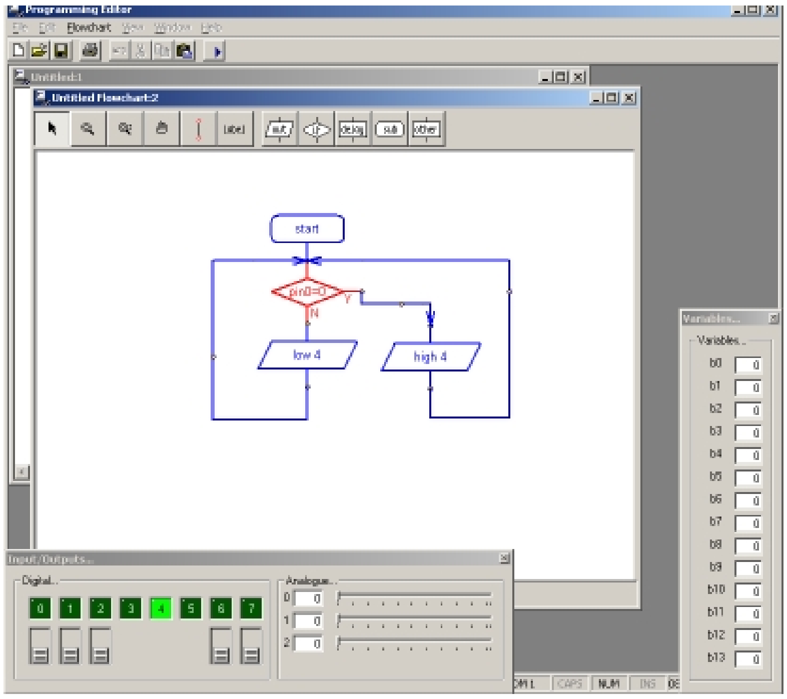
This program will display the temperature level on screen, updating every second.

Section 3
Programming - Drawing Flowcharts

Flowcharts are a useful tool that allow programs to be drawn graphically to make them easier to understand. The Programming Editor software includes a flowchart editor that allows flowcharts to be drawn on screen. These flowcharts can then be converted to BASIC listings for download into the PICAXE. The flowcharts can also be printed or exported as graphics files for inclusion within project portfolios.

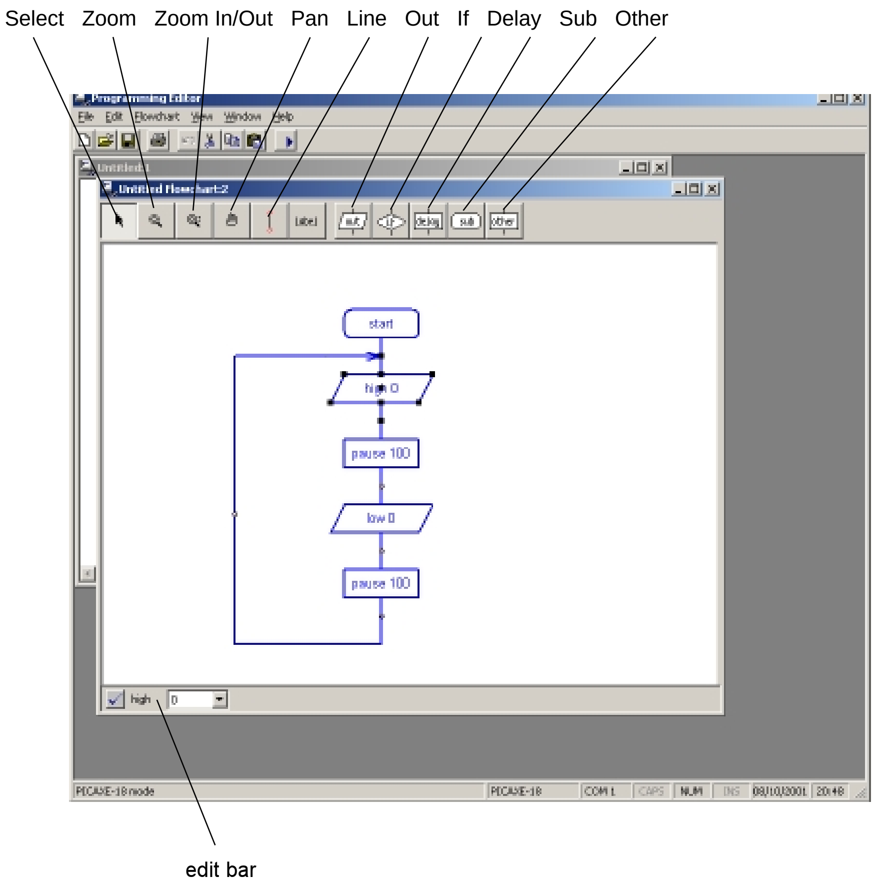
Detailed instructions for drawing/downloading a flowchart:

1. Connect the PICAXE cable to the computer serial port. Note which port it is connected to (normally labelled COM1 or COM2).
2. Start the Programming Editor software.
3. Select View>Options to select the Options screen (this may automatically appear).
4. Click on the 'Mode' tab and select PICAXE-18
5. Click on the 'Serial Port' tab and select the serial port that the PICAXE cable is connected to. Click 'OK'
6. Start a new flowchart by clicking the File>New Flowchart menu.
7. Draw the flowchart by dragging the correct symbols onto the screen, and then using the mouse to draw arrows between the symbols.
8. Once the flowchart is complete it can be converted into a BASIC program by selecting Flowchart>Convert Flowchart to BASIC. The BASIC program can then be downloaded into the PICAXE by clicking the PICAXE>Run menu.
9. To print or save the flowchart, use the File menu options. To export the flowchart as a graphic file, use the File>Export menu. To publish the image in a Word document select file type EMF. To publish the flowchart on an internet web page use the GIF file type.



Flowchart Screen

The Flowchart Editor allows flowcharts to be drawn and simulated on-screen. The flowchart can then be automatically converted into a BASIC program for downloading into the microcontroller.



Flowchart Screen

Select Tool
Use this to select and move shapes. When a single shape is selected it's BASIC code can be edited in the edit bar at the bottom of the window.

Zoom
Use to zoom in to an area of the graph. Right click to zoom out.

Zoom In/Out
To zoom in click and move the mouse up. To zoom out click and move the mouse down.

Pan
Use this tool to move around the flowchart.

Line Tool
Use this tool to draw lines between shapes. Corners can be added by clicking once. When the line is near to a shape it will 'snap' to the connection point.

Label Tool
Use this tool to add descriptive labels or titles to the flowchart.

Out / If / Delay / Sub / Other
Click on these buttons to move to the command sub-menu to select commands.

Drawing Flowcharts

To draw a flowchart click on one of the command menu buttons (out / if / delay / sub / other) on the toolbar to move to the appropriate command sub-menu. Select the appropriate command and then click on the screen where the shape is required. Do not try to locate the shape precisely at first – just drop it in the general area and then use the select tool to move the shape to the correct position.

Once the shape is in position click on it so that it is highlighted. The BASIC code for the shape will then appear in the edit bar at the bottom of the screen. Edit the code as required.

For further information about each command see the 'BASIC Commands' help file.

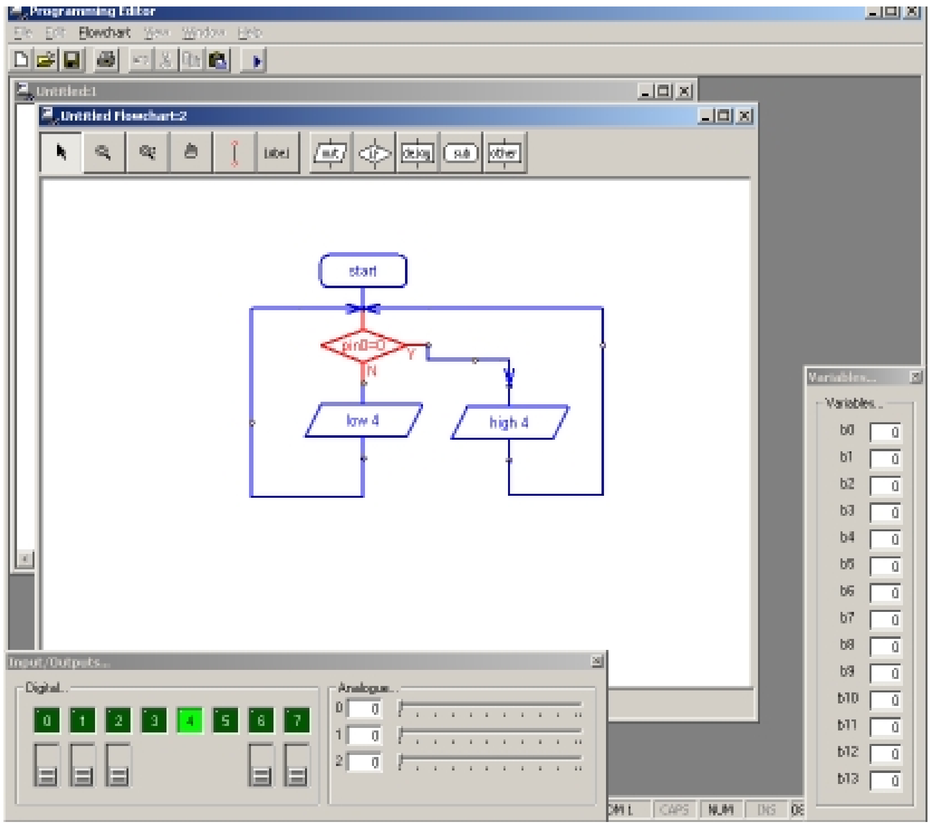
Joining Shapes

Shapes are joined by moving them close together until they 'snap' together. Alternately lines can be drawn between the shapes using the 'line tool' from the main toolbar. Note that it is only possible to join the bottom (side) of shapes to the top of other shapes (you cannot connect lines to lines). Only one line is allowed out of the bottom of each shape.

To enable neat diagrams, corners to the lines can be added by clicking with the mouse. When a line moves close to a connection point it will snap into position and then a click will finish the line.

Lines cannot be moved. If you try to move a line it will be deleted and a new line must be created.

On Screen Simulation



To simulate the flowchart, click ‘Simulate’ from the Flowchart menu. The program will then start to run on-screen.

As the program runs each cell is highlighted red as it is carried out. The ‘Inputs/Outputs’ and ‘Variables’ windows also appear when a simulation is being carried out. To adjust the input values click the on-screen switch (shown beneath the output LED) or slide the analogue input slider.

The time delay between shapes can be adjusted via the Flowchart options (View>Options>Flowchart menu).

Note that certain commands have no on-screen simulation equivalent feature. In this case the command is simply ignored as the flowchart runs.

Downloading Flowcharts

Flowcharts are not directly downloaded to the microcontroller. First the flowchart is converted into a BASIC program, which is then downloaded.

To convert a program select 'Convert' from the Flowchart menu. The BASIC program for downloading will then be created.

Shapes that are not connected to the 'start' or 'sub' shapes in the flowchart are ignored when the conversion takes place. The conversion will stop if an unconnected shape is found. Therefore always use a 'stop' shape or line to complete the flowchart before simulation or conversion.

Note that it is possible to quickly convert and then download a flowchart by pressing the shortcut key <F5> twice.

Using Symbols

Inputs, Outputs and Variables can all be renamed using the 'Symbol Table' from the Flowchart menu. When a symbol is renamed the new name appears in the drop-down menus on the edit bar. Note that you should not use commands (e.g. switch or sound) as a symbol as this will generate errors in your converted BASIC program.

Saving and Printing Flowcharts

Flowcharts can be saved, printed and exported as graphic files (for adding to word processor documents) via the File menu. Flowcharts can also be copied to the Windows clipboard (for pasting into other applications) via the Edit menu.

Section 4
Programming - BASIC

Programming in BASIC is more powerful than using flowcharts. This is because BASIC contains more commands, eg. for...next loops, which cannot be used with the graphical flowchart methods. However you have to be more accurate in your 'typing' as no spelling mistakes are allowed!

The following program is a sample BASIC program which switches output 0 on and off every second. When you download the program an LED connected to output 0 would flash on and off every second..

```
main:
        high 0
        pause 1000
        low 0
        wait 1
        goto main
```

This program uses the **high** and **low** commands to control output pin 0, and uses the **pause** and wait commands to make a delay. Wait uses whole second units, whilst pause uses 1 millisecond (ms) units (1000 ms = 1 second). Therefore in this program both the delays are the same, just written in different ways.

The last **goto main** command makes the program 'jump' back to the label **main:** at the start of the program. This means the program loops forever. Note that the first time the label is used it must be followed by the colon (:) symbol. This tells the computer the word is a new label.

Detailed instructions:

1. Connect the PICAXE cable to the computer serial port. Note which port it is connected to (normally labelled COM1 or COM2).
1. Start the Programming Editor software.
2. Select View>Options to select the Options screen (this may automatically appear).
3. Click on the 'Mode' tab and select PICAXE-18
4. Click on the 'Serial Port' tab and select the serial port that the PICAXE cable is connected to. Click 'OK'
5. Type in the following program:

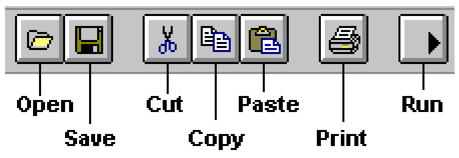
```
main:
        high 0
        pause 1000
        low 0
        wait 1
        goto main
```

(NB note the colon (:) directly after the label 'main' and the spaces between the commands and numbers)

6. Make sure the PICAXE circuit is connected to the serial cable, and that the batteries are connected.
7. Select PICAXE>Run. A download bar should appear as the program downloads. When the download is complete the program should start running automatically – the LED on output 0 should flash on and off every second.

Programming Editor Software Reminders:

Toolbar short-cuts:



To download/run a BASIC program:
1. Check the download cable is connected to the PICAXE and the computer's serial port
2. Check that the battery is connected to the PICAXE
3. Make sure the Programming Editor software is in the correct mode (look for 'PICAXE-18' in the statusbar at the bottom left of the screen).
4. Click **PICAXE>Run** (or the toolbar icon) (or press the shortcut key F5)

To save a program/flowchart:
1. Click **File - Save As...** (or the toolbar icon)
2. Type in a filename
3. Click <OK>

To open a saved program/flowchart:
1. Click **File - Open...** (or the toolbar icon)
2. Select the file type (BASIC or flowchart)
3. Select a filename from the list by clicking on it
4. Click <OK>

To start a new BASIC program:
1. Click **File - New**

To start a new flowchart:
1. Click **File - New Flowchart** (or the toolbar icon)

To on-screen simulate a flowchart:
1. Click **Flowchart - Simulate...** (or the toolbar icon)
2. Click on the flowchart to stop the simulation

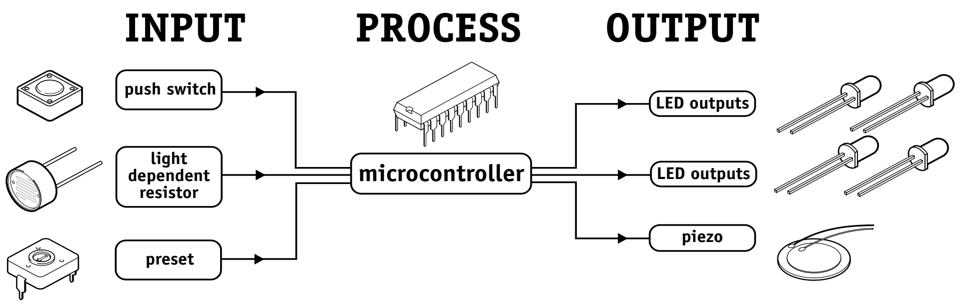
To convert a flowchart to BASIC:
1. Click **Flowchart - Convert to BASIC...** (or press the shortcut key F5)

To print a program/flowchart:
1. Click **File - Print...** (or the toolbar icon)
2. If you want each program line printed in A BASIC program to have a number, make sure the 'Print Line Numbers' box is checked
3. Click <OK>

SECTION 5 - THE ELECTRONIC GAME PCB

The Electronic Game project uses a PICAXE-18 microcontroller with LED outputs. The project also uses a switch to activate the LEDs.

The electronic block diagram is shown below.

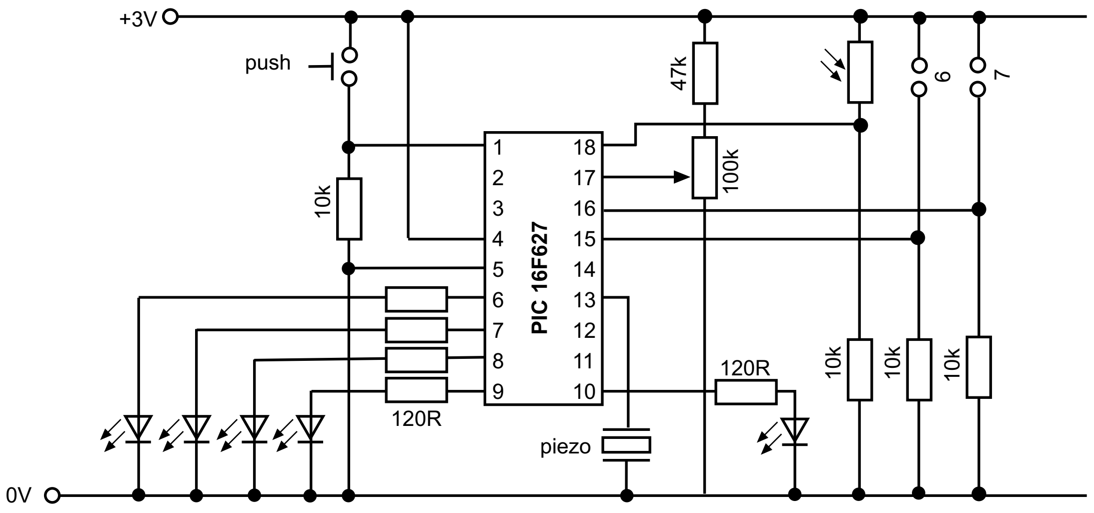


output - pin0 (leg 6) is connected to LED1
output - pin1 (leg 7) is connected to LED2
output - pin2 (leg 8) is connected to LED3
output - pin3 (leg 9) is connected to LED4
output - pin4 (leg 10) is connected to LED5
output - pin7 (leg 7) is connected to the piezo sounder
input - pin0 (leg 17) is connected to the preset resistor
input - pin1 (leg 18) is connected to LDR (optional)
input - pin2 (leg 1) is connected to the push switch

if desired two extra switches/sensors can be connected to inputs 6 and 7.

Circuit Diagram

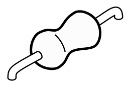
The circuit diagram for the basic electronic game is shown below:

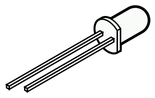


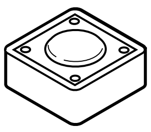
Building the Electronic Game PCB

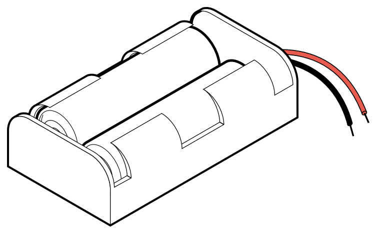
What you will need:

R1 to 5	120R resistor	(brown red brown gold)
R7	22k resistor	(red red orange gold)
R6, R8-9, R10-11	10k resistor	(brown black orange gold)
R10	47k resistor	(yellow violet orange gold)
LED1 to 5	5mm LEDs	
SW	push switch	
VR1	100k variable preset	
IC1	18 pin IC socket	
IC1	PICAXE-18 microcontroller	
CT1	PICAXE download 3.5mm socket	
BT1	3V (2xAAA) battery box	
PCB	printed circuit board	
PZ	piezo sounder	
single core wire	to connect switches and battery box	



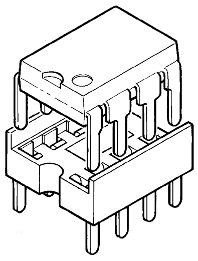




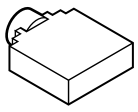


Optional:
LED1 may be replaced with a miniature LDR if desired. This provides an extra input instead of the output LED.

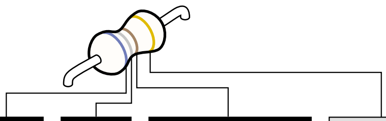
If a 9V PP3 battery is preferred, a 78L05 voltage regulator is required in position RG1. If a 3V battery box is used (recommended) simply use a wire link across the two outer holes of position RG1.



Tools:
soldering iron and solder
side cutters



Resistor colour codes



Black	0	0	Black x1	Silver ±10%
Brown	1	1	Brown x10	Gold ±5%
Red	2	2	Red x100	
Orange	3	3	Orange x1000	
Yellow	4	4	Yellow x10,000	
Green	5	5	Green x100,000	
Blue	6	6	Blue x1,000,000	
Violet	7	7		
Grey	8	8		
White	9	9		

Example shown:
blue, grey, brown, gold
= 680R ±5%

Soldering the PCB.

The printed circuit board (PCB) is specially manufactured with a 'solder resist' layer to make it simpler to solder. This is the green 'lacquer' layer that covers the tracks so that the solder does not stick to these tracks. However for successful assembly the PCB must be carefully assembled and soldered.

When soldering always make sure the solder iron tip is hot and clean. To test if it is hot enough try to melt a piece of solder on the tip. The solder should melt almost instantly. Then clean off the melted solder by wiping the tip on a damp sponge.

Remember that solder will only 'stick' to hot surfaces. Therefore never melt the solder on the soldering iron tip and then try to 'drop' it onto the joint – this won't work as the joint will be cold and so the solder won't stick.

To successfully solder you must hold the soldering iron in one hand and the solder in the other. Therefore make sure the board is held on the table so it won't move (e.g. use a bulldog clip or get someone else to hold it for you).

Steps to soldering:

1) Clean the soldering iron tip on the damp sponge
2) Press the soldering iron tip against the pad on the PCB AND the leg of the component. Count to 3 to give the joint time to warm up.
3) Keep the soldering iron in position and touch the solder against the joint. Allow enough solder to melt to cover the joint.
4) Take the solder away first, then the soldering iron
5) Allow the solder to cool for about 5 seconds before trying to move the board.

After each joint is made make sure it does not accidentally 'bridge' across to other joints. However be aware that some solder joints (e.g. on the two sides of the PICAXE download socket) have two wires very close together that are already connected by a track (line) on the PCB. In this case it does not matter if the solder joins together.

Tips!

1) Always start with the smallest components like the resistors. Then move onto larger components like the IC socket and then finish with the tall components like capacitors and transistors. Do not try to put all the components in position at once, only do two or three at a time.
2) Always make sure that the components lie flat on the board before they are soldered. When using components with long legs like resistors and LEDs, bend the legs so that the component is held firmly in position before soldering.
3) Make sure the PICAXE stereo download socket 'snaps' into position flat on the board before it is soldered.
4) Make sure that the components that only work one way around (LEDs, diodes, transistors and capacitors) are correctly aligned before soldering (see the marks on the PCB).
5) Piezo sounder wires are very thin. Make sure you do not overheat them or they may melt.
6) Always thread the battery snap wires down and up through the thread holes before soldering. This helps make a much stronger joint which is less likely to snap off.

With the electronic game the LEDs must be mounted slightly above the PCB so they fit through the case correctly. Try the position in the case before soldering.

1) Place the 22k (red red orange gold) resistor and the five 10k (brown black orange gold) resistors in position. Bend the legs to hold the resistors in position and then solder.
2) Place the five 120R (brown red brown gold) resistors and 47k (yellow violdet orange gold) resistor in position and solder. Bend the legs to hold the resistors in position and then solder. Note if using an enhanced PICAXE-18A microcontroller the 47k resistor is not required and should be replaced with a wire link.
3) Use an off-cut resistor leg to make a wire link in the positions LINK1 and LINK2.
4) If using a 9V PP3 battery, solder a 78L05 regulator in position RG1. If using a 3V battery pack the regulator is not required. In this case use a wire link to solder across the two outer contacts in position RG1 (as marked on the PCB).
5) Push the PICAXE stereo download socket onto the PCB and **make sure it clicks into position** (so that it lies flat on the board). Solder the five metal square contacts (the five round plastic support post holes do not have to be soldered). Do not worry if the solder joins on the two metal contacts either side of the socket as they are supposed to be joined anyway.
6) Push the IC socket into position. Make sure the notch at one end points towards the top. Fold the legs over to hold the socket in position and then solder.
7) Solder the variable resistor in position.
8) Solder the LEDs into position. Make sure that the flat on one side of the LED aligns with the flat marked on the PCB
9) Thread the battery clip down through the large hole by the letters AXE. Thread it back up through the large hole by the letters 105 then solder the black wire into the hole marked 0v and the red wire into the hole marked V+
10) Carefully check the board to make sure there are no missed joints or accidental solder bridges.
11) Insert the microcontroller into the socket, ensuring pin1 faces the correct way.
12) Solder the piezo sounder to the holes marked PZ.
13) Solder the +ve battery connector to the minature on/off switch (side contact). Solder a wire from the centre switch contact to the BT+ connection on the PCB. Solder a third wire from the -ve battery connector to the BT- connection on the PCB.
14) Solder two wires from the SW connection on the PCB and connect to the push switch connector in the casing via screws.

If using the optional miniature LDR instead of LED1, this is soldered in the pads beside R8 and R1 (at right angles to the LED position).

If using the two optional extra inputs on inputs 6 and 7, these can be connected via direct wire links to the rectangles marked 6 and 7 on the PCB, or a second stereo headphone socket can be used (requires modification to casing to fit).

Testing your circuit.

Step 1 – Check the solder joints.

Check that the solder does not accidentally bridge between two pads. This is most likely to happen on the LEDs. On the stereo socket the two round pads close together on each side can be joined as they are already joined by a track on the board. However they must not be joined to the central round hole.

Step 2 - Check the components.

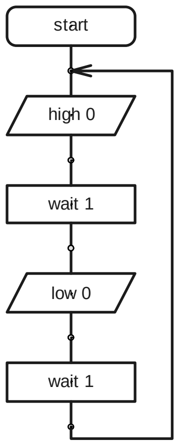
1) Check that the black battery clip wire is in the hole marked 'BT-' and the red battery clip wire is in the hole marked 'BT+' (via the on-off switch)
2) Check that the PICAXE-18 chip is in the socket correctly, with the dent (showing pin1) closest to the stereo socket.
3) Check that the flat edge of the LEDs is connected to the correct hole on the PCB.
4) Check that the stereo socket is correctly soldered, including the middle round pad which is often forgotten by mistake.

Step 3 - Connect the battery.

Check the 2 AAA batteries are in the battery box correctly. Put your finger on the microcontroller and switch the power switch on. If it starts to get hot switch off immediately as there is a problem – most likely that the chip or the battery wires are around the wrong way.

Step 4 – Download a program to test LED 0.

Connect the cable to the back of the computer and to the PICAXE socket on the PCB. Make sure the cable is pushed fully into the socket on the PCB.



Make sure the software is in the PICAXE-18 mode and the correct serial port is selected (see section 4 of this booklet for more information).

Type in and download the following program:
program like this:

```
main:
        high 0
        wait 1
        low 0
        wait 1
        goto main
```

The LEDs should flicker as the program downloads. After the download is complete the LEDs should flash on and off every second. If the LED does not flash check that it is around the correct way and that the 120R resistors are in the correct positions on the PCB.

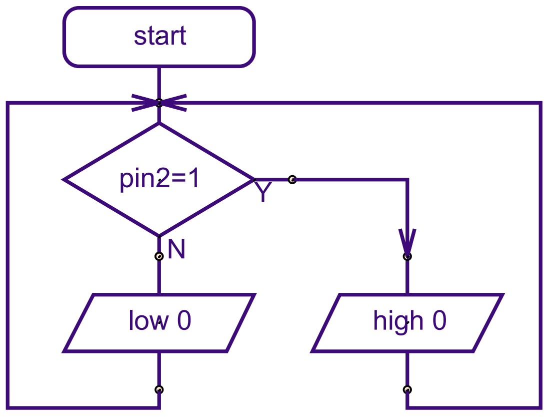
If the program does not download check that the 22k, 10k, socket and IC socket are all soldered correctly. Use a multimeter to make sure you are getting 4.5V across the top legs (1 and 8) of the microcontroller. Check that the cable is pushed firmly into the socket and that the correct serial port is selected within the software.

Step 5 – Test LEDs 1-4.

Repeat the program in step 4, but use high 1 and low 1 instead of high 0 and low 0. Then repeat on outputs 2, 3 and 4. This will test all the other LEDs.

Step 6 – Test The switch.

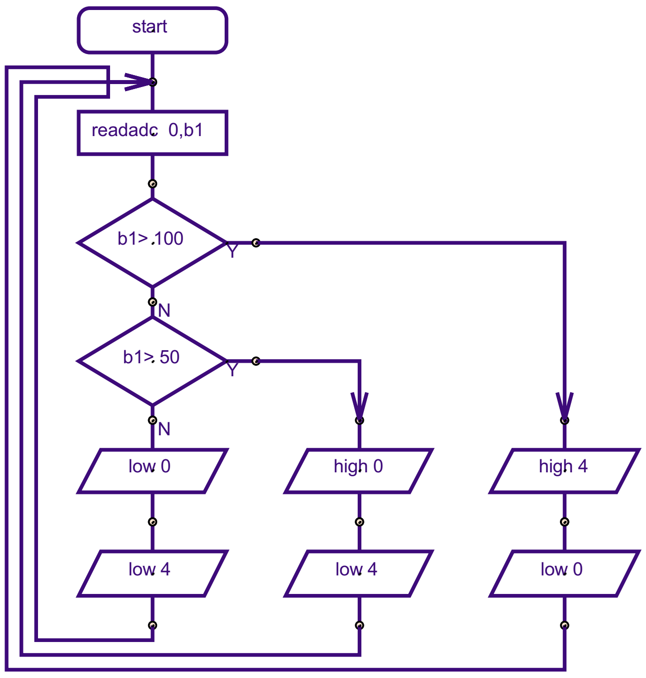
```
main:
        if pin2 = 1 then LEDon
        low 0
        goto main

LEDon:
        high 0
        goto main
```



The LED should light as you press the switch. If they do not check that the switch and 10k resistor are correctly soldered.

Step 7 – Test the preset resistor.

```
main:
        readadc 0,b1
        if b1 > 100 then do4
        if b1 > 50 then do0
        low 0
        low 4
        goto main

do4:
        high 4
        low 0
        goto main
do0:
        high 0
        low 4
        goto main
```



The LEDs should light and go out as you rotate the preset resistor back and forward.

If all these tests pass, you can be congratulated as you have correctly built and assembled your Electronic Game! It is now time to develop and test your own program for your own game!

Section 6 - Program Ideas.

Now that you have assembled and tested your Electronic Game, it is time to develop your own program.

Included on the next pages are four example programs. These are designed to give you a starting point for your program. You may choose to modify them or to start a completely new program if you prefer.

Be creative!

Program 1 Explanation - Timing Device

This program is a simple timer. After the switch is pressed, the 5 LEDs light in turn before the piezo sounder makes a noise. The preset resistor can be used to adjust the time delay.

Program 2 Explanation - Quiz Indicator

This program uses three switches (large switch on input 2 and two additional switches connected to inputs 6 and 7) and three LEDs (LEDs 1-3) to show which team presses their switch first. The LED lights for a short period of time, which can be adjusted by the preset resistor.

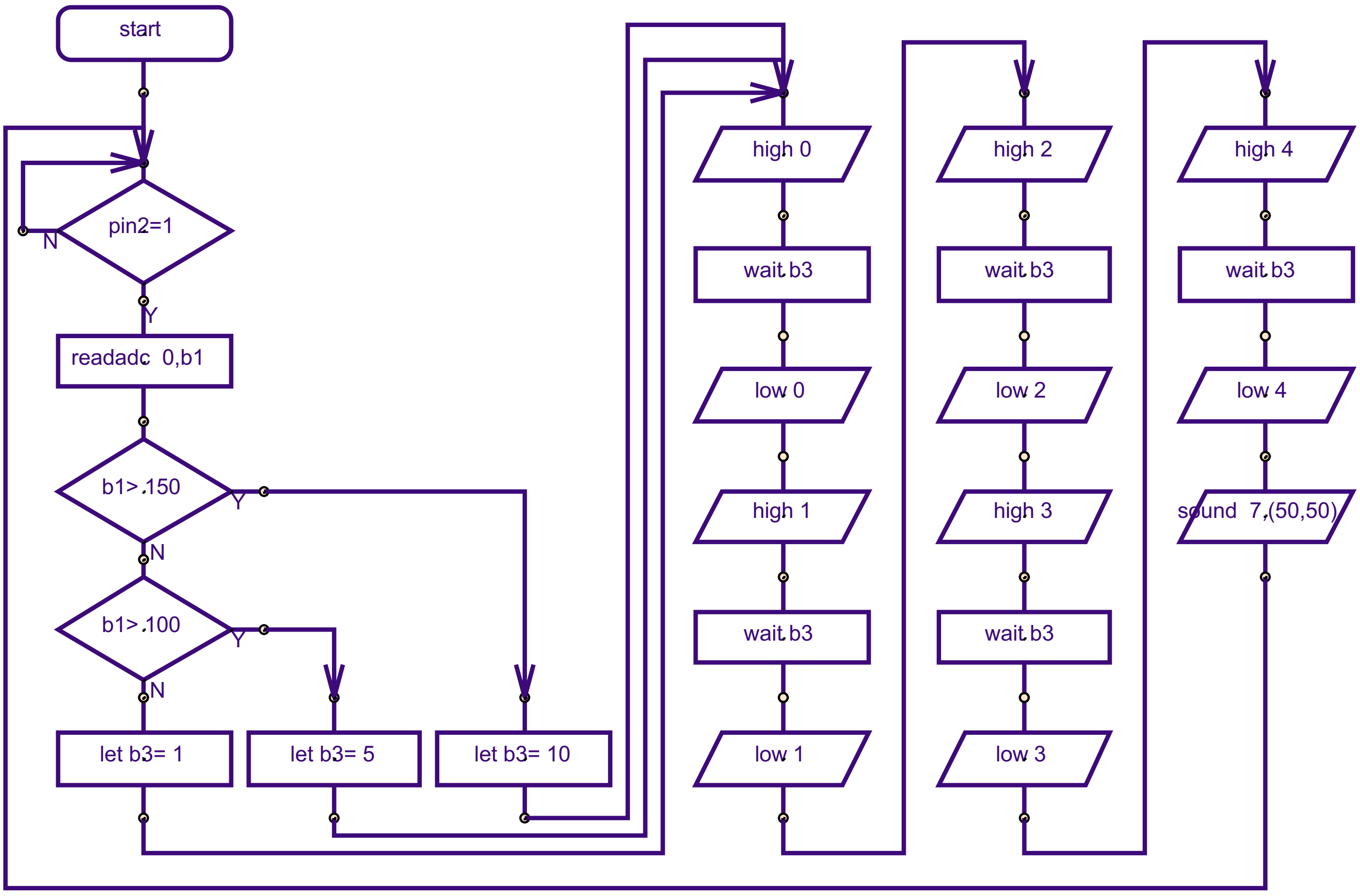
Program 3 Explanation - Temperature Alarm

This program is more of an alarm than a game. When the switch is pressed, a temperature reading is taken. Different LEDs are then lit depending on the temperature, and if it is too cold the buzzer sounds. This could be used as a hyper or hypothermia alarm for the elderly.

Program 4 Explanation - Simon Game

This program is quite complicated beacuse it mimics the Simon game described on page 1. Four LEDs (1-4) are used and 4 switches. One switch is the main large push switch on input 2. The second switch is connected to input1 in the LDR position (instead of LED0 which is not required). The other two switches are connected to inputs 6 and 7.

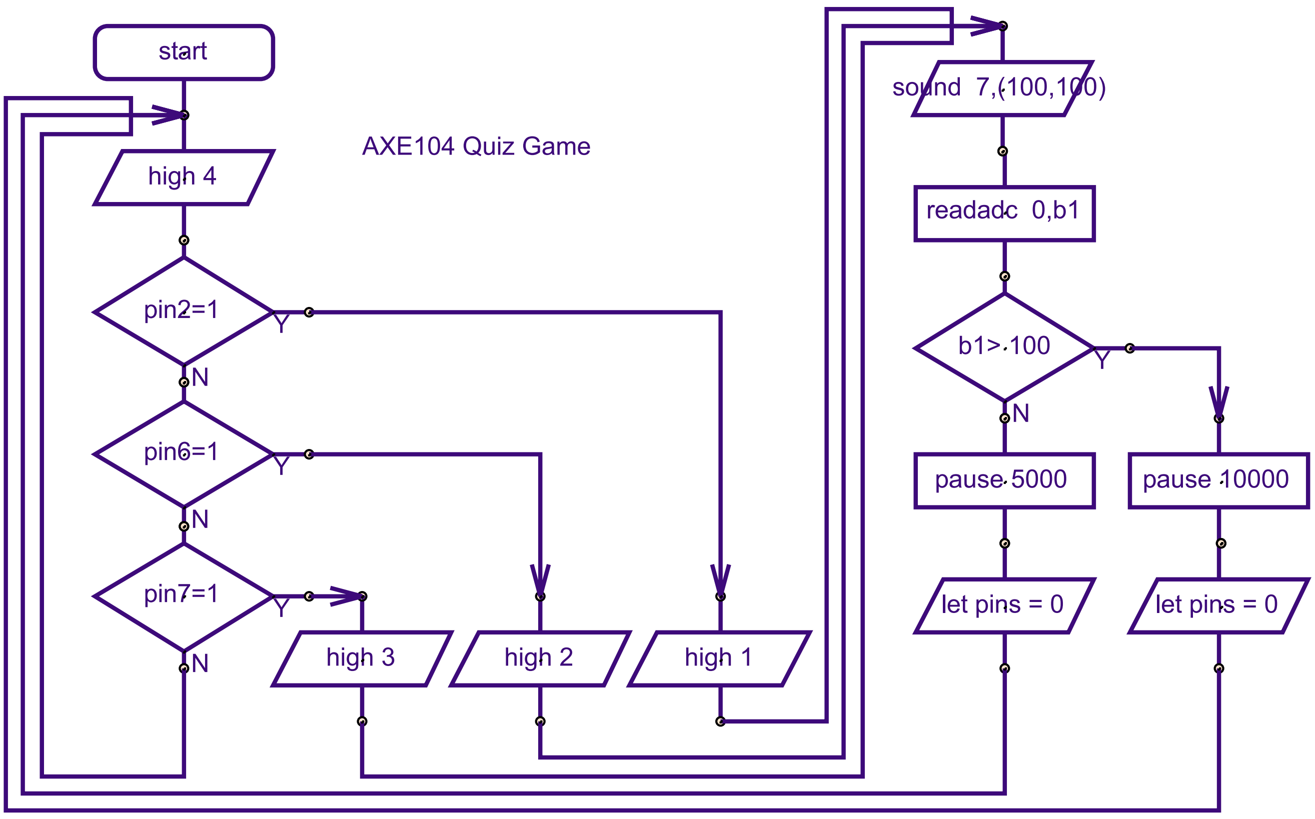
Program 1


start
pin2=1
N
Y
readadc 0,b1
b1>.150
Y
N
b1>.100
Y
N
let b3= 1
let b3= 5
let b3= 10
high 0
wait b3
low 0
high 1
wait b3
low 1
high 2
wait b3
low 2
high 3
wait b3
low 3
high 4
wait b3
low 4
sound 7,(50,50)

Program 1

```
'' AXE104 Timer Program
` Note the use of a word variable (w3)
` as the pause values are greater than
` a byte value (255)
main:
        if pin2 = 1 then start ' test switch
        goto main

start:  readadc 0,b1            ' read preset value
        if b1 > 150 then time10        ' if bigger than 150 jump
        if b1 > 100 then time5 ' if bigger than 100 jump

time1:  let w3 = 1000           ' 1 second
        goto flash

time5:  let w3 = 5000           ' 5 second
        goto flash

time10: let w3 = 10000          ' 10 second
        goto flash

flash:  high 0                  ' light LEDs
        pause w3
        low 0
        high 1
        pause w3
        low 1
        high 2
        pause w3
        low 2
        high 3
        pause w3
        low 3
        high 4
        pause w3
        low 4
        sound 7,(50,50)                 ' beep
        goto main               ' back to start
```

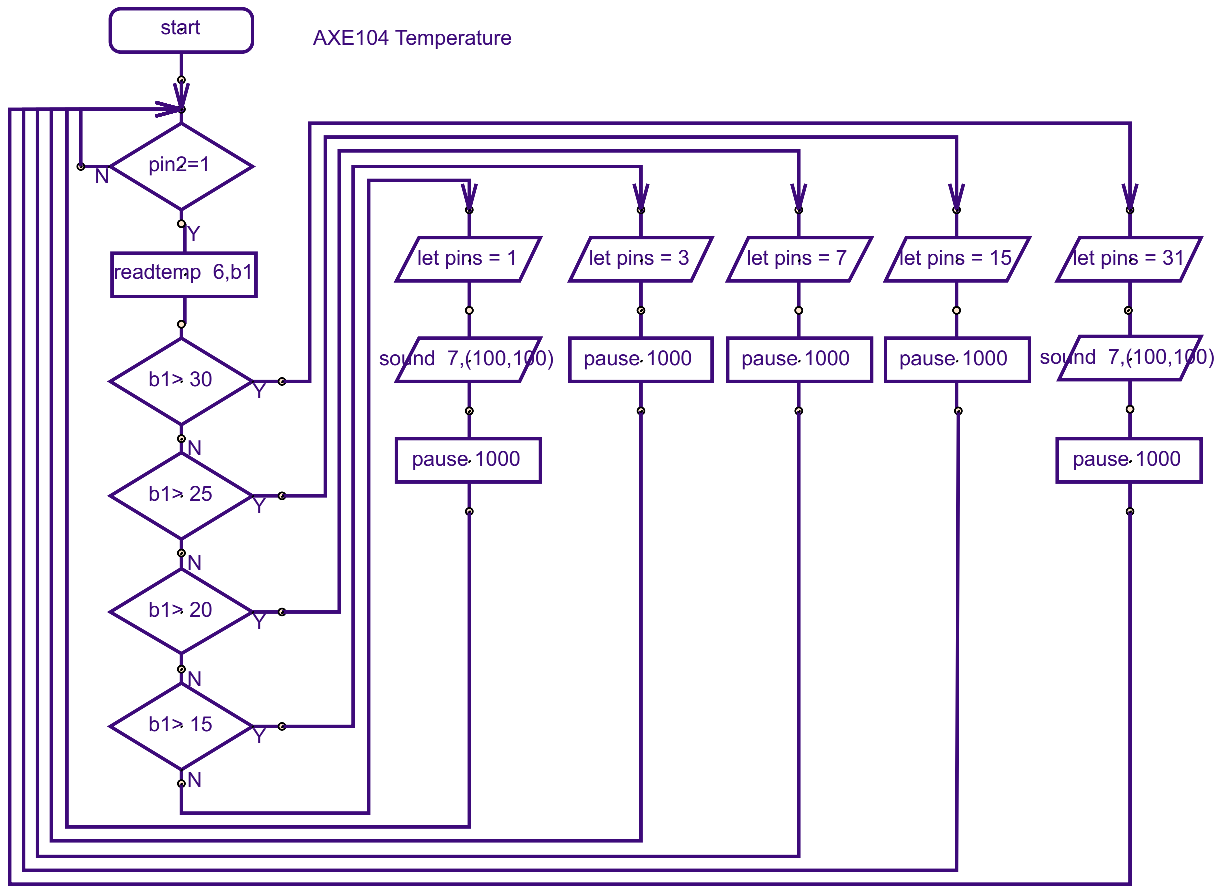
Program 2


start
high 4
AXE104 Quiz Game
pin2=1
Y
N
pin6=1
Y
N
pin7=1
Y
N
high 3
high 2
high 1
sound 7,(100,100)
readadc 0,b1
b1>100
Y
N
pause 5000
pause 10000
let pins = 0
let pins = 0

Program 2

```
' AXE104 Quiz Program

main:   high 4                  ' use LED4 as power indicator
        if pin2 = 1 then team1 ' test switch
        if pin6 = 1 then team2 ' test switch
        if pin7 = 1 then team3 ' test switch
        goto main

team1:                          ' team 1 LED
        high 1
        goto timer

team2:                          ' team 2 LED
        high 2
        goto timer

team3:                          ' team 3 LED
        high 3
        goto timer

timer:  sound 7,(100,100)       ' make beep
        readadc 0,b1            ' read preset value
        if b1 > 100 then time10        ' if biiger than 100 jump

time5:  pause 5000              ' 5 seconds
        low 1                   ' LEDs off
        low 2
        low 3
        goto main               ' back to start

time10: pause 10000             ' 10 seconds
        low 1                   ' LEDs off
        low 2
        low 3
        goto main               ' back to start
```

Program 3


start
AXE104 Temperature
pin2=1
N
Y
readtemp 6,b1
b1> 30
Y
N
b1> 25
Y
N
b1> 20
Y
N
b1> 15
Y
N
let pins = 1
sound 7,(100,100)
pause 1000
let pins = 3
pause 1000
let pins = 7
pause 1000
let pins = 15
pause 1000
let pins = 31
sound 7,(100,100)
pause 1000

Program 3

```
' AXE104 Temperature Program
' Note requires PICAXE-18A microcontroller
' with DS18B20 temp sensor on input6

main:   if pin2 = 1 then temp  ' test switch
        goto main

temp:   readtemp 6,b1          ' read temperature value
        if b1 > 30 then temp30 ' test for above 30 degrees
        if b1 > 25 then temp25 ' test for 25-30 degrees
        if b1 > 20 then temp20 ' test for 20-25 degrees
        if b1 > 15 then temp15 ' test for 15-20 degrees

temp10: let pins = %00000001   ' light one LED - too cold
        sound 7,(100,100)      ' make warning sound
        pause 1000
        goto main              ' back to start

temp15: let pins = %00000011   ' light two LEDs
        pause 1000
        goto main              ' back to start

temp20: let pins = %00000111   ' light three LEDs
        pause 1000
        goto main              ' back to start

temp25: let pins = %00001111   ' light four LEDs
        pause 1000
        goto main              ' back to start

temp30: let pins = %00011111   ' light five LEDs - too hot
        sound 7,(100,100)      ' make warning sound
        pause 1000
        goto main              ' back to start
```

Program 3

The Simon Says program is fairly complex and makes extensive use of variables to remember how many lights are in the sequence, and how many switches the player has pressed. The light sequence is stored in the EEPROM data memory of the PICAXE-18A microcontroller, and in theory up to 256 steps are available, although only 64 are used here!

The first job is to wait for the start switch to be pressed. When this switch is pushed the data memory is loaded with a random sequence of 64 values, which can be the values 1, 2, 3 or 4. As the random command returns a number between 0 and 255, a simple comparison test is made between 0-60, 60-120, 120 –180, 180-255 to load just the values 1 to 4 into memory. Note how the values are made up as a series of 1+1+1+1 steps, this is a useful programming technique which can reduce the total number of lines in a program.

A new game is then started with just two steps in the sequence. This is achieved by loading the value 1 into a variable called counter, so that there will at first be two steps in the playback loop (step 0 and step 1).

The playback loop then reads the memory and makes the beep noise/lights the LED as appropriate via the 'beep' sub-procedure.

It is then turn for the player to respond. The number of presses a player has made is stored in the variable called 'position', and so this variable is first reset to 0. Then the correct response is read back from memory and saved into variable 'key'.

The PICAXE then waits for a key to be pressed. If the key is correct the beep/LED are shown, the position is incremented, and the game continues. When the sequence is completed program flows jumps down to 'success', which makes a noise and then adds one to counter so that the game can be continued with one extra step in the sequence.

If the incorrect key is pressed program flow jumps to 'fail' and a failure noise is made before the program jumps back to the start.

Note this program is too complicated to be drawn as a flowchart.

```
` Simon Says Game

` Push swithes on inputs 1,2,6,7
` LEDs on outputs 1-4
` Piezo on output 7

symbol rand = b1           ` random for loading memory
symbol key = b2            ` switch 1-2-3-4
symbol position = b3       ` position of player in game
symbol freq = b4           ` sound variable
symbol counter = b5        ` number of steps in sequence

` *** This section waits for start ***

` wait for any switch to be pushed
` with all four LEDs lit
` preload rand with any number by repeatedly
` using the random command in the loop

init:
       let pins = %00011110
       random rand
       if pin1 = 1 then preload
       if pin2 = 1 then preload
       if pin6 = 1 then preload
       if pin7 = 1 then preload
       goto init

` *** This section loads memory for game ***

` load EEPROM data memory with 64 numbers
` first get the random number (0 to 255)
` and then change to either 1,2,3 or 4
` and then save into data memory

preload:
       for position = 0 to 63     ' for..next loop

       let key = 0
       random rand          ' get random number 0-255
       if rand > 180 then set1
       if rand > 120 then set2
       if rand > 60 then set3

set4:  let key = key + 1   '1+1+1+1 = 4
set3:  let key = key + 1   '1+1+1 = 3
set2:  let key = key + 1   '1+1 = 2
set1:  let key = key + 1   '1

       write position,key  ' save in data memory

       next position       ' next loop
```

```
' *** This section plays back a sequence ***

' switch off the LEDs and then start
' a game with the end counter as 1
       let pins = %00000000
       let counter = 1

' playback the game sequence
playback:
       for position = 0 to counter
          read position,key
          gosub beep
          pause 500
       next position

' *** This section detects the players reply sequence ***

' now the user responds
' reset the players position to 0
       position = 0

' get the correct key player is supposed to hit
' from the EEPROM memory
gameloop:
       read position,key

' if position is equal to counter then all done
       if position = counter then success

' else wait for switch to be pressed
loop:  if pin1 = 1 then played1
       if pin2 = 1 then played2
       if pin6 = 1 then played3
       if pin7 = 1 then played4
       goto loop

' switch pressed so check it is the correct one
' if it is make a beep sound and then continue
' else fail the game
played1:
       if key <> 1 then fail
       let position = position + 1
       gosub beep
       goto gameloop

played2:
       if key <> 2 then fail
       let position = position + 1
       gosub beep
       goto gameloop
```

```
played3:
       if key <> 3 then fail
       let position = position + 1
       gosub beep
       goto gameloop

played4:
       if key <> 4 then fail
       let position = position + 1
       gosub beep
       goto gameloop

` *** Failed so make noise and jump back to start ***
` failed so make failed noise, light all LEDs
` and go back to start
fail:
       pause 250
       let pins = %00011110
       sound 7,(100,300)
       goto init

` *** Succeeded so add another step to sequence and loop ***

` success so make a success sound
` and then increment counter and do another sequence
success:
       pause 50
       sound 7,(50,100)
       pause 50
       sound 7,(50,100)
       pause 500
       let counter = counter + 1
       goto playback

` *** sub light LED and beep ***

'sub-procedure to light correct LED
'and make a different beep sound for each LED
'key always contains number 1,2,3 or 4
'multiply sound freq by 10 to give larger difference

beep:
       high key
       freq = key * 10
       sound 7,(freq,100)
       low key
       return
```

ACKNOWLEDGEMENT

This project development was partly funded by the UK Offshore Oil and Gas Industry.
www.oilandgas.org.uk/education/

Supported by: