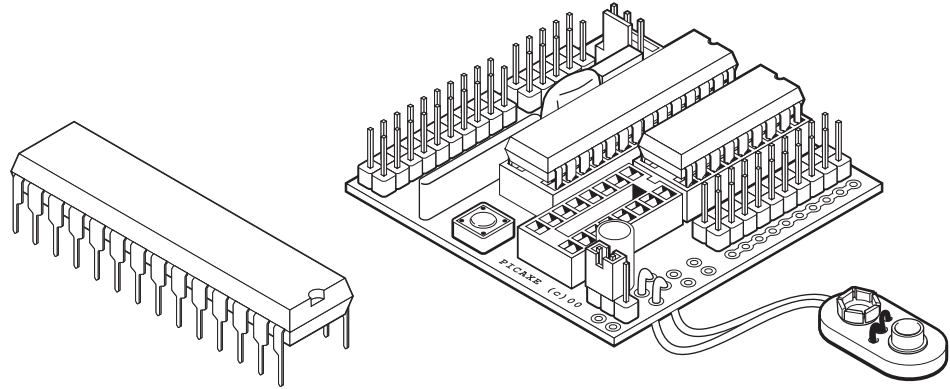1

PICAXE MICROCONTROLLER PROGRAMMING SYSTEM

The 'PICAXE' is an easy-to-program microcontroller system that exploits the unique characteristics of the new generation of low-cost 'FLASH' memory based microcontrollers. These microcontrollers can be programmed over and over again without the need for an expensive programmer.

The power of the PICAXE system is its simplicity. No programmer, eraser or complicated electronic system is required - the microcontroller is programmed (with a simple 'BASIC' program) via a 3-wire connection to the computer's serial port. The operational PICAXE circuit uses just 3 components and can be easily constructed on a prototyping breadboard, strip-board or PCB design.

The PICAXE microcontroller provides 22 input/output pins - 8 digital outputs, 8 digital inputs, 4 analogue inputs and 2 serial interface pins.

- *low-cost, simple to construct circuit*
- *8 inputs, 8 outputs and 4 analogue channels*
- *rapid download via serial cable*
- *free, easy to use Programming Editor software*
- *simple to learn BASIC language*
- *can also be programmed via flowcharts with Crocodile Technology software.*
- *free manuals and online support forum*
- *infrared remote control upgrade pack available*
- *servo controller upgrade pack available*

The comprehensive starter pack includes the following items:

- *interface board*
- *download cable*
- *CDROM containing software and manuals*
- *PICAXE-28 microcontroller chip*

Downloading your first program

This first simple program can be used to test your system. It requires the connection of an LED (and 330R resistor) to output pin 7. If connecting the LED directly to a PICAXE chip on a home-made board, connect the LED between the output pin and 0V. When using the interface board (as supplied with the starter pack), connect the LED between V+ and the pin, as the output is buffered by the darlington driver chip on the interface board. (Make sure the LED is connected the correct way around!).

Direct connection to PIC	Connection to PIC via interface board
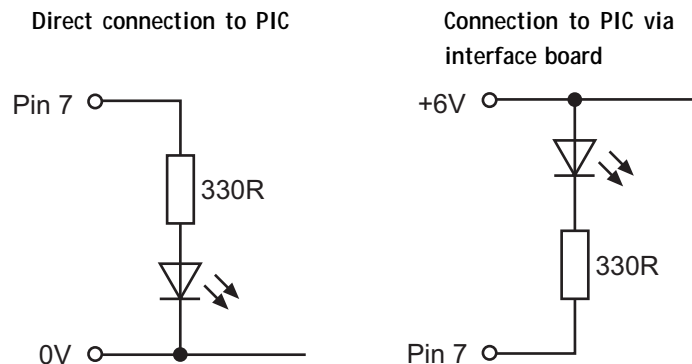
Note: The PICAXE microcontroller can be connected to a computer via the 28 pin project board OR via a simple self-made circuit board. Circuit connections are provided later in this booklet.

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-28

5. Click on the 'Serial Port' tab and select the serial port that the PICAXE cable is connected to.

6. Click 'OK'

7. Type in the following program:

```
main:   high 7
        pause 1000
        low 7
        pause 1000
        goto main
```

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

8. Make sure the PICAXE circuit is connected to the serial cable, and that the batteries are connected. Make sure the LED and 330R resistor are connected to output 7.

9. 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 7 should flash on and off every second.

3

Trouble-shooting

If an error message appears check the on-screen warning description. Common mistakes are:

- *Not connecting the cable to the PICAXE circuit or computer serial port.*

- *Not selecting the correct COM port setting within the Programming Editor software.*

- *Not connecting, or trying to use a flat battery.*

The PICAXE Circuit

The basic PICAXE circuit is shown below.
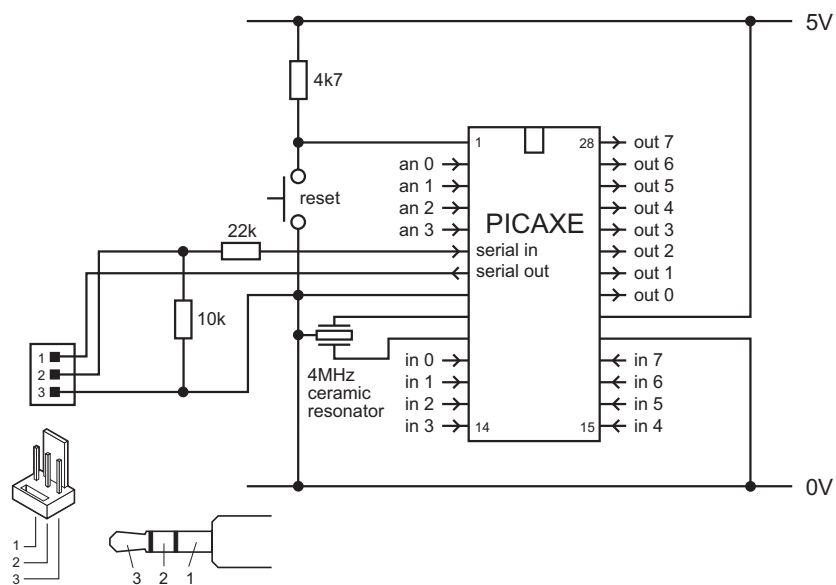
The 4MHz 3-pin ceramic resonator sets the 'clock speed' of the PICAXE microcontroller. To ensure correct timings it is essential that a 4MHz device is used. Each PICAXE microcontroller is supplied with a suitable 3 pin ceramic resonator. These devices are not polarised and so can be connected either way around.

The 4k7 resistor is used to pull the PICAXE microcontrollers reset pin (pin 1) high. If desired, a reset switch can also be connected between the reset pin (pin 1) and 0V. When the switch is pushed the PICAXE microcontroller 'resets' to the first line in the program.

Note that the microcontroller has two 0V connections (pin 8 and pin 19). These pins are internally connected within the microcontroller.

4

The PICAXE computer interface circuit

The PICAXE system uses a very simple interface to the computer serial port. Although this interface does not use true RS232 voltages, it is very low-cost and has proved to work reliably on almost all modern computers.
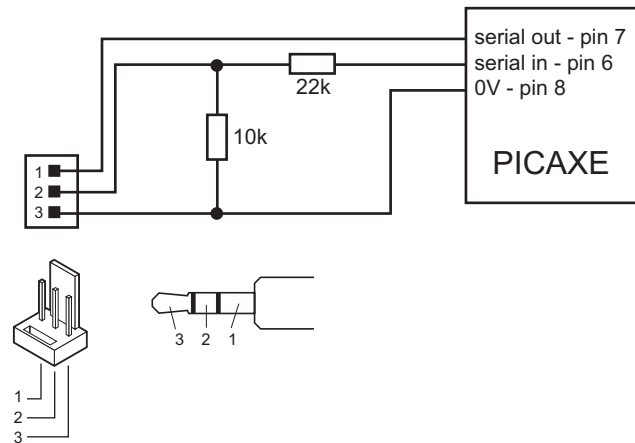
It is strongly recommended that this interfacing circuit is included on every PCB designed to be used with the PICAXE microcontroller. This enables the PICAXE microcontroller to be re-programmed without removing from the PCB.

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

When using a computer with the older 25-pin serial port connector, use a 9-25 way mouse adapter to convert the 9-pin PICAXE cable. These adapters can be purchased from all good high street computer stores.

The PICAXE Interface

The PICAXE interface is designed to allow rapid prototyping with the PICAXE system. It consists of a small PCB that contains the PICAXE circuit and basic interfacing circuits, leading to input/output ribbon cable connectors.

Power

The interface can be powered by a single, or two separate, power supplies. When two power supplies are used the second separate supply (V2+) is used to power the output devices. This is useful when a higher voltage is required (e.g. when driving a 12V stepper motor) or when it is necessary to isolate the outputs to avoid noise problems (e.g. when using radio-control servos).

The primary power supply (V1+) for the interface is a 6V or 4.5V DC. This is connected via the battery connector. DO NOT use a 9V PP3 battery, as the interface does not include a voltage regulator.

When a single power supply is used a jumper link is required on the external power jumper link 'E' as shown below.
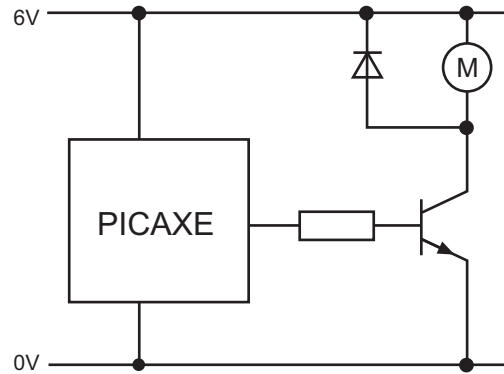
these two pins connected
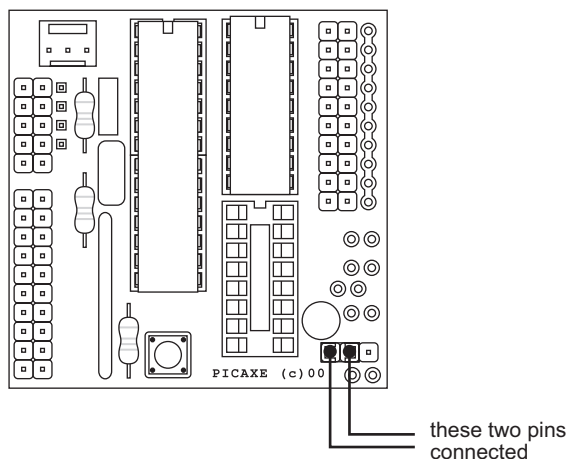
When two power supplies are required the second supply is connected to jumper E as shown below.
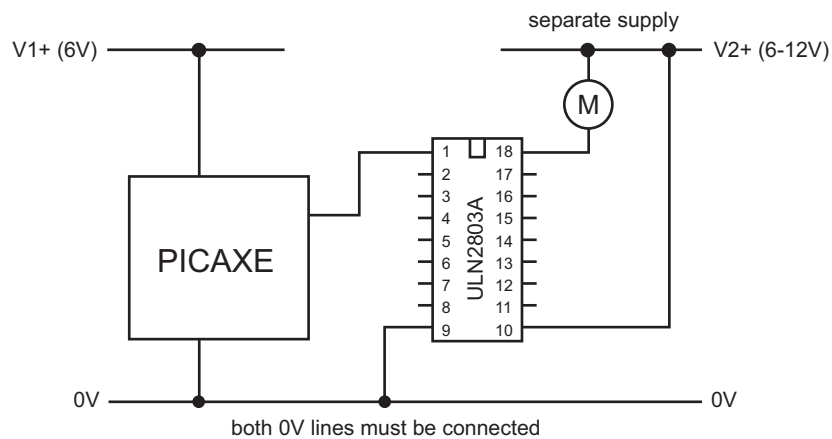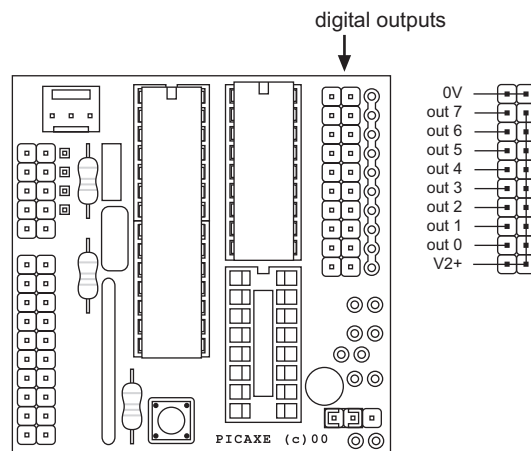
0V

V2+
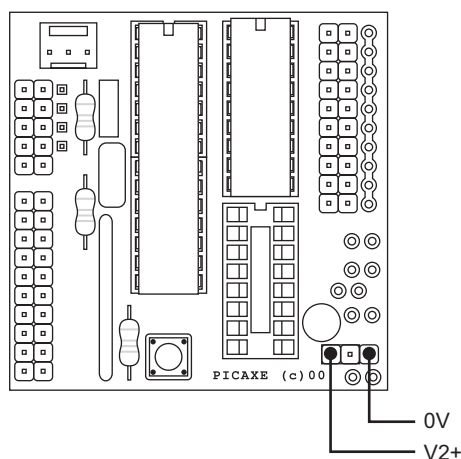
Digital Outputs

Each digital output is connected to a transistor as shown below.

On the board the transistors are contained in a single ULN2803A darlington driver IC. The equivalent output circuit is shown below.

both 0V lines must be connected

The digital output pins on the board are identified as follows:

digital outputs

```
0V
out 7
out 6
out 5
out 4
out 3
out 2
out 1
out 0
V2+
```

The output ribbon cable connector has the following pin arrangement (wire 1 is marked with red ink).

| 1	0V
2	0V
3	7
4	V2+
5	6
6	V2+
7	5
8	V2+
9	4
10	V2+
11	3
12	V2+
13	2
14	V2+
15	1
16	V2+
17	0
18	V2+
19	V2+
20	V2+

wire 1

PICAXE (c)00
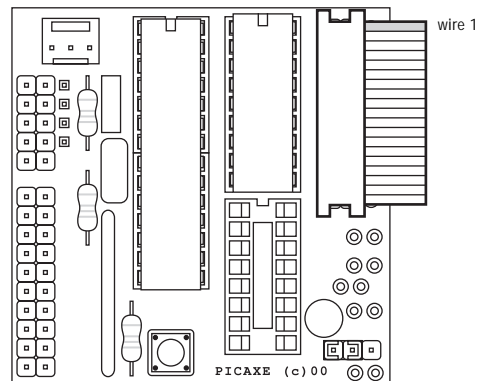
Motor Driver

An optional L293D motor driver IC (not supplied) may be added to the basic interface. This provides forward-backward control of two DC motors, controlled by outputs 4 to 7. Naturally, if only one motor is to be controlled then only two output lines are used.

optional L293D motor driver IC
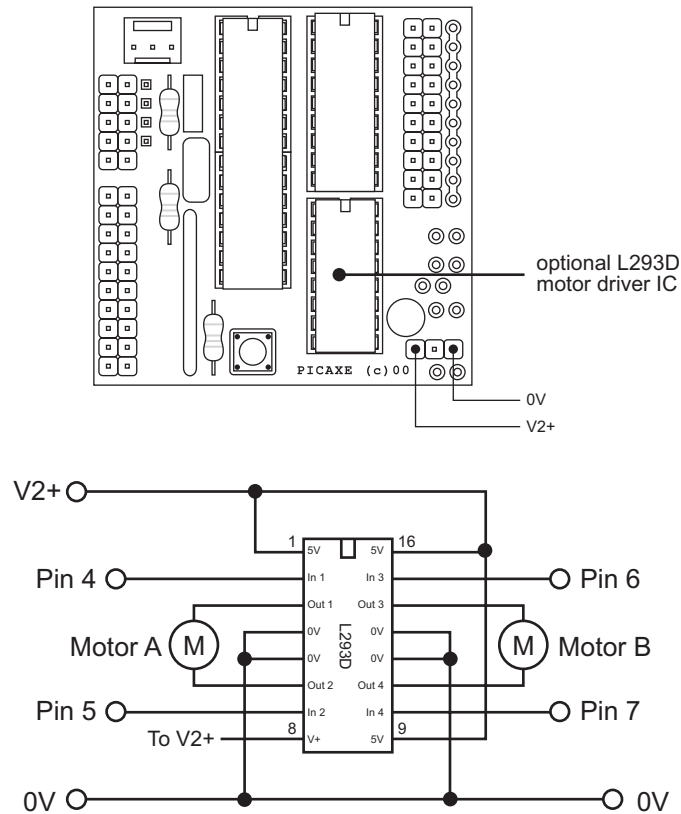
Both inputs low	- motor halt
First output high, second output low	- motor forward
First output low, second output high	- motor reverse
Both inputs high	- motor halt

Changing the states of the input pins has the effect of altering the direction of current flow through the motor, as shown below.
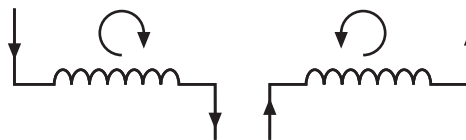

Note that the L293D will become warm with continuous use. A heatsink bonded onto the top of the chip will help keep it cool.

Motor A and B are connected to the interface via the pads provided. It is necessary to solder connectors or wires into these pads.

Digital Inputs

The digital inputs have a 10k pull down resistor on each input pin.
The equivalent input circuit is shown right.

digital inputs
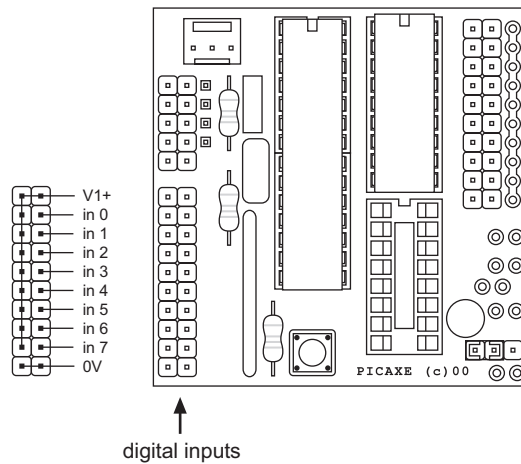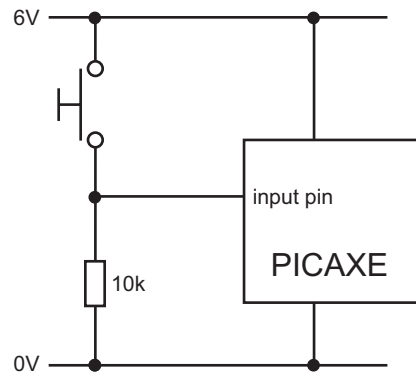
The input ribbon cable connector has the following pin arrangement (wire 1 is marked with red ink).

20	V1+
19	V1+
18	V1+
17	0
16	V1+
15	1
14	V1+
13	2
12	V1+
11	3
10	V1+
9	4
8	V1+
7	5
6	V1+
5	6
4	V1+
3	7
2	0V
1	0V
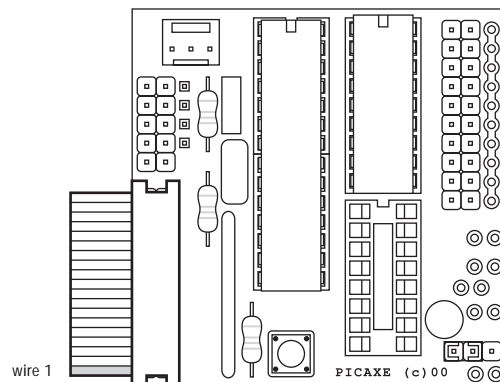
wire 1

Analogue Input Channels

The analogue input channels are left floating (no connection).
The equivalent input circuit is shown below.
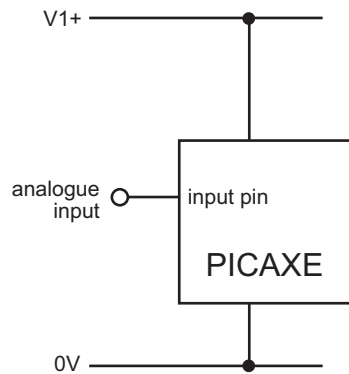
The analogue input ribbon cable connector has the following pin arrangement (wire 1 is marked with red ink).

10	V1+
9	0
8	V1+
7	1
6	V1+
5	2
4	V1+
3	3
2	0V
1	0V
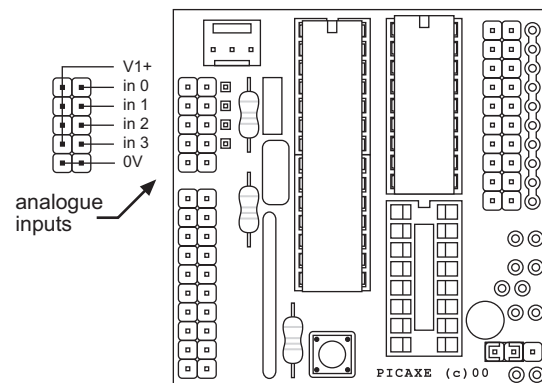
Radio Control Servo Upgrade

An upgrade pack is available that enables the PICAXE to be used to control a servomotor. Please see the separate servo upgrade datasheet.

Infra-red Remote Control Upgrade

An upgrade pack is available that enables the PICAXE to be used for remote control applications. See the separate infra-red remote control upgrade datasheet.

BASIC Programming Basics

The following programs are included to provide a brief introduction to a few of the main programming techniques. All programs can be tested by connecting an LED (with a 330R resistor) to outputs 6 and 7, a switch to input 0, and a variable resistor to analogue channel 0.

For further details about each program see the Commands section.

Switching outputs on and off

The following program switches output 7 on and off every second. The program demonstrates how to use the high, low, wait, pause and goto commands. When you download this program the red LED on output pin 7 should flash on and off continuously.

```
main:                   ' make a label called 'main'
     high 7             ' switch output 7 on
     wait 1             ' wait 1 second
     low 7              ' switch output 7 off
     pause 1000         ' wait 1000ms (= 1 second)
     goto main          ' jump back to start
```

The first line simply adds a label called 'main' to the program. A label is used as a positional marker in the program. In this program the last line uses the command 'goto main' to jump back to the first line. This creates a loop that continues forever.

A label can be any word (apart from keywords such as 'high'), but must begin with a letter. When the label is first defined it must end with a colon (:). The colon 'tells'the computer that the word is a new label.

It is usual to put some spaces (or a tab) at the start of each line, apart from where the line starts with a label. This makes the program easier to read and understand. Comments can also be added after an apostrophe (') symbol, to make the program operation easier to understand.

Note that the commands wait and pause both create time delays. However wait can only be used with whole seconds, pause can be used for shorter time delays (measured in milliseconds).

Detecting Inputs

The following program makes output pin 7 flash every time a switch on input pin 0 is pushed.

```
main:                       ' make a label called 'main'
      if pin0 = 1 then flash   ' jump if the input is on
      goto main             ' else loop back around

flash:                      ' make a label called 'flash'
      high 7                ' switch output 7 on
      pause 500             ' wait 0.5 second
      low 7                 ' switch output 7 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 7 on for half a second before returning to the main loop.

Note carefully the spelling in the if…then line – pin0 is all one word (without a space). Note also that only the label is placed after the command then – no other commands apart from a label are allowed after then.

Using Symbols

Sometimes it can be hard to remember which pins are connected to which devices. The 'symbol' command can then be used at the start of a program to rename the inputs and outputs.

```
symbol red = 7          ' rename output7 'red'
symbol green = 5        ' rename output5 'green'

main:                   ' make a label called 'main'
      high red          ' red LED on
      wait 2            ' wait 2 seconds
      low red           ' red LED off
      high green        ' green LED on
      wait 1            ' wait 1 second
      low green         ' green LED off
      goto main         ' jump back to the start
```

For…Next Loops

It is often useful to repeat the same part of a program a number of times, for instance when flashing a light. In these cases a for…next loop can be used.

```
symbol red = 7            ' rename output7 'red'
symbol counter = b0       ' define a counter using variable b0

main:                     ' make a label called 'main'
    for counter = 1 to 25 ' start a for…next loop
        high red          ' red LED on
        pause 500         ' wait 0.5 second
        low red           ' red LED off
        pause 500         ' wait 0.5 second
    next counter          ' next loop

    end                   ' stop the program
```

In this program all the code between the for and next lines is repeated 25 times. The number of times the code has been repeated is stored in a variable called 'counter', which in turn is a symbol for the variable 'b0'. There are 14 available variables, b0 to b13, which can be used in this way. A variable is a location in the memory where numbers can be stored.

Using Variables

The variables are commonly used to store 'numbers' as a program runs. This program would light up all the outputs (if an LED was connected) in different combinations

```
main:                     ' make a label called 'main'
    let b0 = b0 + 1       ' add one to b0
    let pins = b0         ' make outputs = value of b0
    pause 100             ' wait 0.1 second
    goto main             ' jump back to the start
```

Note that b0 is a byte variable. This means it supports any value between 0 and 255. This means that the program above will eventually 'overflow' at the highest number i.e. …253-254-255-0-1-2… This is an important fact to remember when performing mathematics with byte variables.

For further details about the mathematical capabilities of the PICAXE microcontroller see the Commands section.

Reading Analogue Input Channels

The value of an analogue input can be easily copied into a variable by use of the 'readadc' command. The variable value (0 to 255) can then be tested. The following program switches on one LED if the value is greater than 200 and a different LED if the value is less than 100. If the value is between 100 and 200 both LEDS are switched off.

```
main:                           ' make a label called 'main'
      readadc 0,b0              ' read channel 0 into variable b0
      if b0 > 200 then red      ' if b0 > 200 then do red
      if b0 < 100 then green     ' if b0 < 100 then do green
      low 7                     ' else switch off 7
      low 6                     ' and switch off 6
      goto main                 ' jump back to the start

red:                            ' make a label
      high 7                    ' switch on 7
      low 6                     ' switch off 6
      goto main                 ' jump back to start

green:                          ' make a label
      high 6                    ' switch on 6
      low 7                     ' switch off 7
      goto main                 ' jump back to start
```

Note that the PICAXE microcontroller has 4 analogue channels labeled 0 to 3. When using analogue sensors is often necessary to calculate the 'threshold' value necessary for the program (ie the values 100 and 200 in the program above). The debug command provides an easy way to see the 'real-time' value of a sensor, so that the threshold value can be calculated by experimentation.

```
main:                           ' make a label called 'main'
      readadc 0,b0              ' read channel 0 into variable b0
      debug b0                  ' transmit value to computer screen
      pause 100                 ' short delay
      goto main                 ' jump back to the start
```

After this program is run a 'debug' window showing the value of variable b0 will appear on the computer screen. As the sensor is experimented with the variable value will show the current sensor reading.

Sub-procedures

A sub-procedure is a separate 'mini-program' that can be called from the main
program. Once the sub-procedure has been carried out the main program continues.
Sub-procedures are often used to separate the program into small sections to make it
easier to understand. Sub-procedures that complete common tasks can also be copied
from program to program to save time.

The following program uses two sub-procedures to separate the two main sections of
the program('flash' and 'blink').

```
symbol red = 7              ' rename output7 'red'
symbol green = 6            ' rename output6 'buzzer'
symbol counter = b0         ' define a counter using variable b0

main:                       ' make a label called 'main'
      gosub flash           ' call the sub-procedure flash
      gosub blink           ' call the sub-procedure blink
      goto main             ' loop back

      end                   ' end of the main program

blink:                      ' make a sub-procedure called blink
      for counter = 1 to 25 ' start a for...next loop
         high red           ' red LED on
         pause 50           ' wait 0.05 second
         low red            ' red LED off
         pause 50           ' wait 0.05 second
      next counter          ' next loop
      return                ' return from the sub-procedure

flash:                      ' make a sub-procedure called flash
      high green            ' green on
      wait 2                ' wait 2 seconds
      low green             ' green off
      return                ' return from the sub-procedure
```

The main program is very simple as it simply calls the two sub-procedures via the
gosub command.

At the end of each sub-procedure a return command is used to return program flow
back to the main program loop. Note that at the end of the main program (but before
the sub-procedures) an end command has been added. This is a good idea when using
sub-procedures as it stops the main program accidentally 'falling' into a sub-procedure
at the end of the main program.

The PICAXE-28 Commands

The list below is a full summary of the commands supported by the PICAXE system. For syntax details and example programs see the Commands section.

DIGITAL OUTPUT

HIGH	Switch an output pin high (on).
LOW	Switch an output pin low (off).
TOGGLE	Toggle the state of an output pin.
PULSOUT	Output a pulse on a pin for given time.
SERVO	Provide pulses to control radio-control style servos.

ANALOGUE OUTPUT

PWM	Provide a PWM output pulse.
SOUND	Output a sound.

DIGITAL INPUT

IF… THEN	Jump to new program line depending on input condition..
PULSIN	Measure the length of a pulse on an input pin.

ANALOGUE INPUT

READADC	Read analogue channel into a variable.

PROGRAM FLOW

FOR.. NEXT	Establish a FOR-NEXT loop
BRANCH	Jump to address specified by offset
GOTO	Jump to address
GOSUB	Jump to subroutine at address.
RETURN	Return from subroutine
IF.. THEN	Compare and conditionally jump

VARIABLE MANIPULATION

{LET}	Perform variable mathematics.
LOOKUP	Lookup data specified by offset and store in variable.
LOOKDOWN	Find target's match number (0-N) and store in variable
RANDOM	Generate a pseudo-random number
PEEK	Read from additional RAM registers
POKE	Write to additional RAM registers

SERIAL I/O

SEROUT	Output serial data from output pin. Up to 2400 baud.
SERIN	Serial input data with qualifiers on input pin. Up to 2400 baud.
SERTXD	Output serial data from serial output pin. Fixed 4800 baud.
SERRXD	Serial input data with qualifiers on serial input pin. Fixed 4800 baud.

INTERNAL EEPROM ACCESS

|	
EEPROM	Store data in data EEPROM before downloading BASIC program
READ	Read data EEPROM location into variable
WRITE	Write variable into data EEPROM location
READMEM	Read program EEPROM location into variable
WRITEMEM	Write variable into program EEPROM location

POWER DOWN

NAP	Enter low power mode for short period (up to 2.3 sec)
SLEEP	Enter low power mode for period (up to 65535 sec)
END	Power down until reset

MISCELLANEOUS

PAUSE	Wait for up to 65535 milliseconds
WAIT	Wait for up to 65 seconds
INFRA	Receive infrared signals from an infrared controller.
SYMBOL	Rename variables.
DEBUG	Transmit variables to PC for debugging purposes.

Crocodile Technology software

The Programming Editor is a text based programming system that uses BASIC commands or flowcharts. The electronics simulation package 'Crocodile Technology' from Crocodile Clips Ltd can also be used to program the PICAXE microcontroller. In this system programs are also generated by drawing on-screen flowcharts.
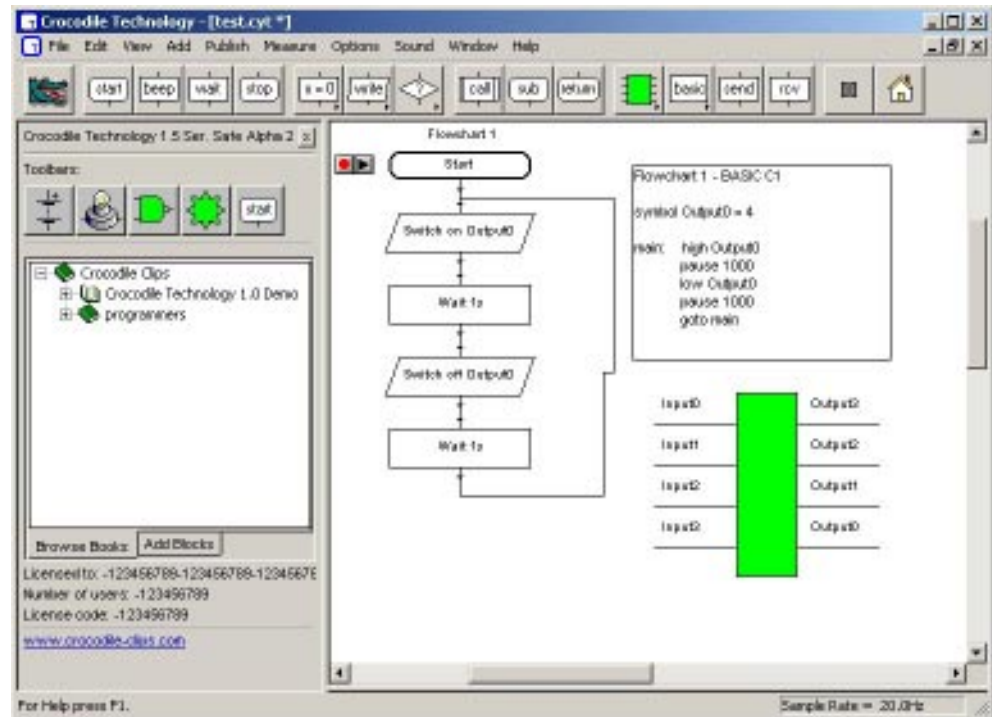
For further details about the crocodile technology software visit www.crocodile-clips.com

The Crocodile Technology software (student version) can be purchased from:
www.tech-supplies.co.uk