PICAXE Manual
www.picaxe.co.uk

revolution

Contents

About this manual

The PICAXE manual is divided into three separate sections:

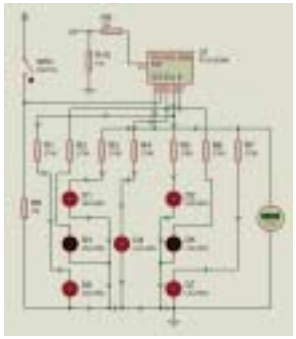
Section 1 -	Getting Started	(picaxe_manual1.pdf)
Section 2 -	BASIC Commands	(picaxe_manual2.pdf)
Section 3 -	Microcontroller interfacing circuits	(picaxe_manual3.pdf)

This first section provides general information for getting started with the PICAXE system. No prior understanding of microcontrollers is required. A series of tutorials introduce the main features of the system.

For more specific information, syntax and examples of each BASIC Command please see section 2 'BASIC Commands'.

For microcontroller interfacing circuits, and example programs, for most common input/output transducers, please see section 3

Software Overview



Revolution Education Ltd publish 4 software titles for use with the PICAXE microcontroller chips. Two are free, the other two are low cost options.

PICAXE Programming Editor
The PICAXE Programming Editor is the main Windows application used for programming PICAXE chips. This software is free of charge to PICAXE users. The Programming Editor supports both textual (BASIC) and flowchart (graphical) methods of developing programs. This manual was prepared using version **5.3.0** of the Programming Editor software.

AXEpad
AXEpad is a simpler, free version of the Programming Editor software for use on the Linux and Mac operating systems. It supports the BASIC programming method.

Logicator for PIC micros
Logicator is a flowcharting application designed for educational use. Programs are developed as graphical flowcharts on screen. These flowcharts are then automatically converted into BASIC files for download into the PICAXE chips.

PICAXE VSM
PICAXE VSM is a full Berkeley SPICE circuit simulator, which will simulate complete electronic circuits using PICAXE chips. The BASIC program can be stepped through line by line whilst watching the input/output peripheral react to the program.

This manual focuses on the BASIC textual programming language, as used by Programming Editor, AXEpad and PICAXE VSM.

Please see the separate Logicator manual for more details about the Logicator flowchart programming method.

Software Comparison

	Prog. Editor	AXEpad	Logicator	PICAXE VSM
BASIC programming option	X	X	(X)	X
Flowchart programming option	(X)		X	
Assembler code option	X			
Windows Version	X	(X)	X	X
Linux Version		X		
Mac OSX Version		X		
On Screen Simulation	X		X	X
Berkeley SPICE Circuit Simulation				X
Support of all PICAXE Types	X	X	X	X
Cost / Distribution	Free	Free	Shareware (£15)	Cost Option (£50)

Key:

X = Supported

(X) = Supported, but more suitable product also available,
e.g. for the flowchart method of programming 'Logicator for PIC micros' is recommended, but the 'PICAXE Programming Editor' may also be used in the legacy flowchart mode.

Software Quick Choice Guide

Windows	-> Textual BASIC programming	-> Programming Editor
	-> Flowchart programming	-> Logicator for PIC
	-> SPICE Circuit Simulation	-> PICAXE VSM
Mac	-> Textual BASIC programming	-> AXEpad
Linux	-> Textual BASIC programming	-> AXEpad

Third Party Software

Revolution produce royality free PICAXE drivers that can be used to add PICAXE support to third party products. Current third party software products include:

Win/Mac/Linux	-> Flowchart programming	-> Yenka PICs
	->Circuit Simulation	-> Yenka Electronics
	->PCB Artwork	-> Yenka PCB
Win/Mac	-> Flowchart programming	-> Flowol

Technical Support Forum

If you have a question about any aspect of the PICAXE system please post a question on the very active (and friendly!) support forum at this website **www.picaxeforum.co.uk**

Quick Start

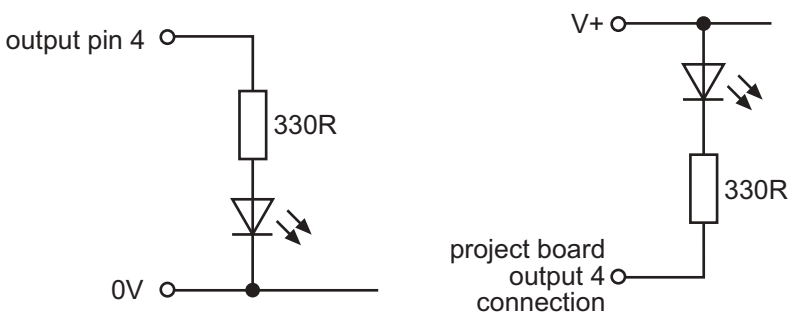
It is strongly recommended that you read the first few chapters of this manual before using the PICAXE system. However if you cannot wait to get going, this quick start guide provides a summary of the information explained in much more detail later in this manual!

1. Install the Programming Editor software from the CDROM (or download from www.picaxe.co.uk).
2. Insert the AXE026 serial cable into the 9 pin serial COM socket at the rear of the computer, or the AXE027 USB cable into an available USB port and install the USB driver when prompted.
3. Start the Programming Editor software (click Start>Programs>Revolution Education>Programming Editor). Then click View>Options menu to display the Options panel (this may also automatically appear on startup). On the 'Mode' tab select the correct type of PICAXE chip. On the 'Ports' tab also select the appropriate serial COM port (the port where you connected the serial / USB cable).
4. Connect an LED and 330 ohm resistor to the output pin 4 of the PICAXE chip. On 'home-made' or prototype circuits connect the LED/resistor between the output pin and 0V. On project boards (which have a Darlington transistor buffered output) connect the LED/resistor between V+ and the output pin. Ensure correct polarity of the LED.
5. Connect the PICAXE cable to the hardware.
6. Connect the 4.5V (3xAA battery) or 5V regulated power supply to the project board. Do NOT use a 9V PP3 battery.
7. Using the software, type in the following program:

```
main:     high 4
          pause 1000
          low 4
          pause 1000
          goto main
```

8. Click the PICAXE>Program menu to download the program to the hardware. After the download the output LED should flash on and off very second.

Congratulations! You have now programmed a microcontroller using the PICAXE System!



At a glance - specifications:

Power Supply:
4.5V or 5V DC is recommended. Do not use 6V, 7.2V or 9V battery packs, these could permanently damage the chip. For trouble-shooting use 3xAA cells only.

28X2/40X2 parts are also optionally available in special low power 1.8V to 3.3V variants. Note that 4.5V or 5V will permanently damage these special low power parts.

Outputs:
Each output can sink or source 20mA. This is enough to light an LED but will not, for instance, drive a motor. Total maximum current per chip is 90mA.

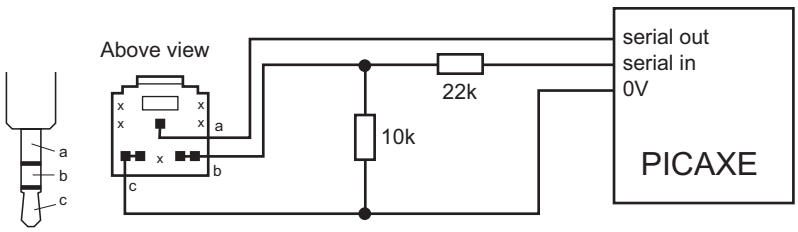
Inputs:
An input should be above (0.8 x power supply voltage) to be high, below (0.2 x power supply voltage) to be low. It is recommended, but not essential, to tie unused inputs low via a 10k resistor.

ADC:
The ADC range is the power supply voltage range. The maximum recommended input impedance is 20k. Unconnected ADC will 'float' giving varying false readings. However 'touch sensor' pins must float (no pullup/pulldown).

Serial download pin:
The serial download pin must **never** be left floating. This will give unreliable operation. Always use the 10k/22k resistors as shown below, even if the chip was programmed on a different board.

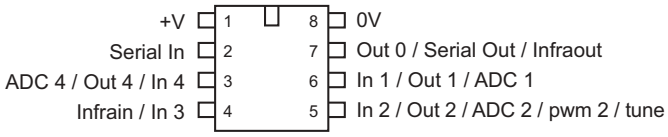
Reset pin:
The reset pin (if present) must **never** be left floating. This will give unreliable operation. Always tie high (ie to the positive supply) via a 4k7 or 10k resistor.

At a glance - download circuit:

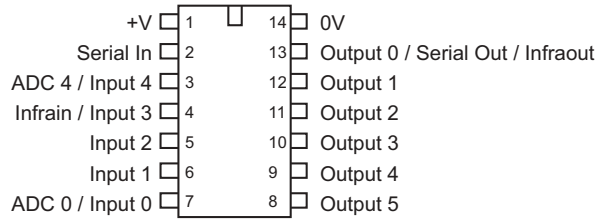


At a glance - pinout diagrams (older parts):

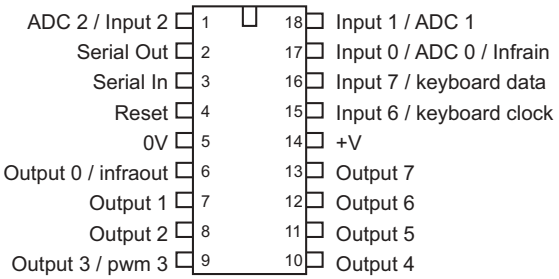
PICAXE-08M



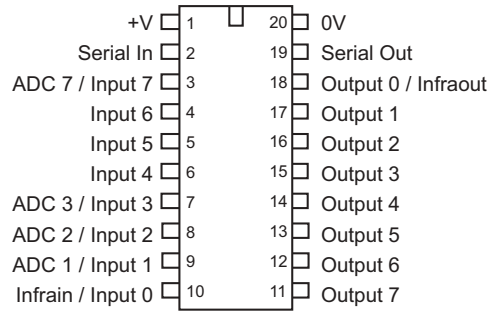
PICAXE-14M



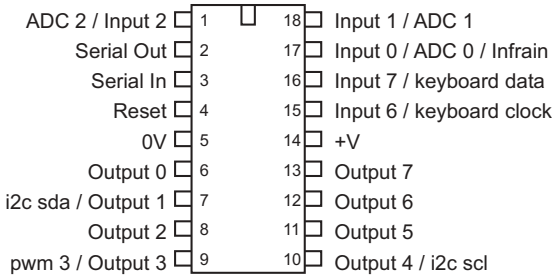
PICAXE-18M



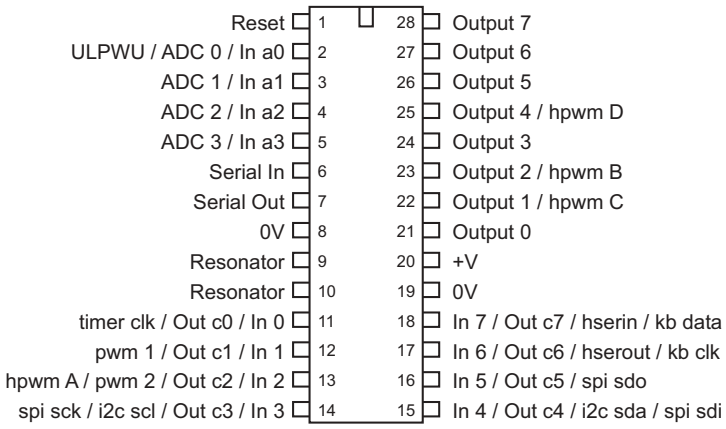
PICAXE-20M



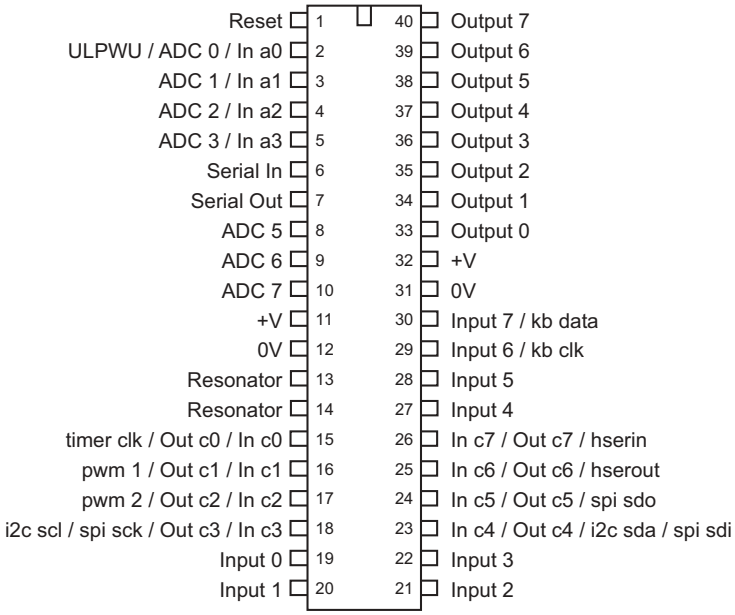
PICAXE-18X



PICAXE-28X1

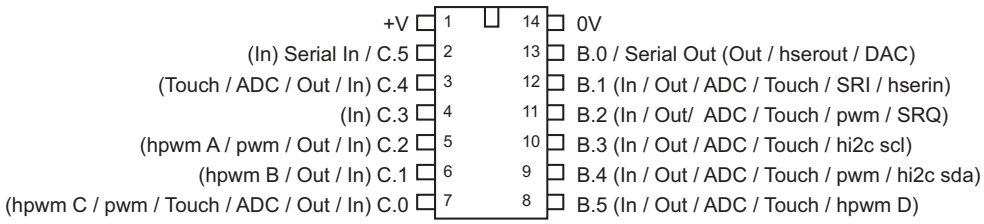


PICAXE-40X1

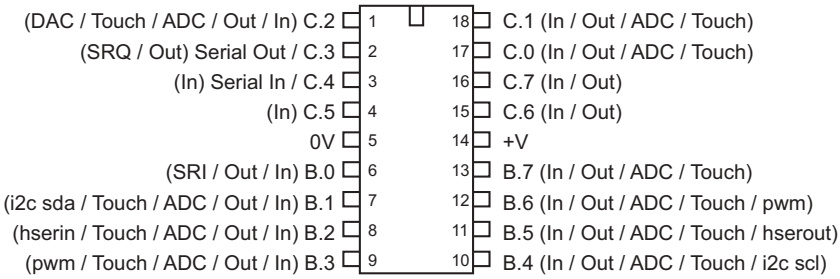


At a glance - pinout diagrams (M2 parts):

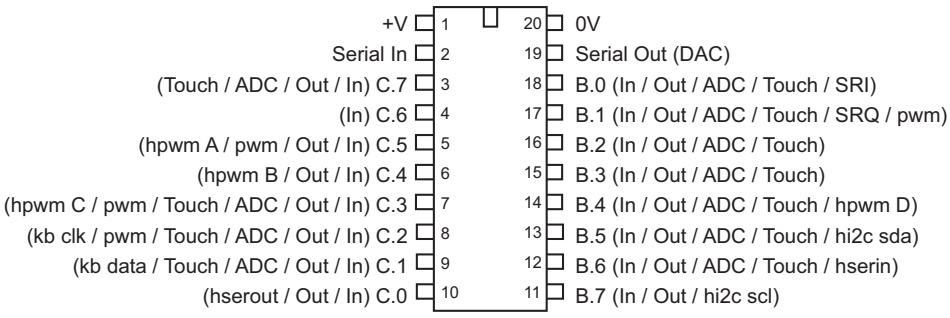
PICAXE-14M2



PICAXE-18M2



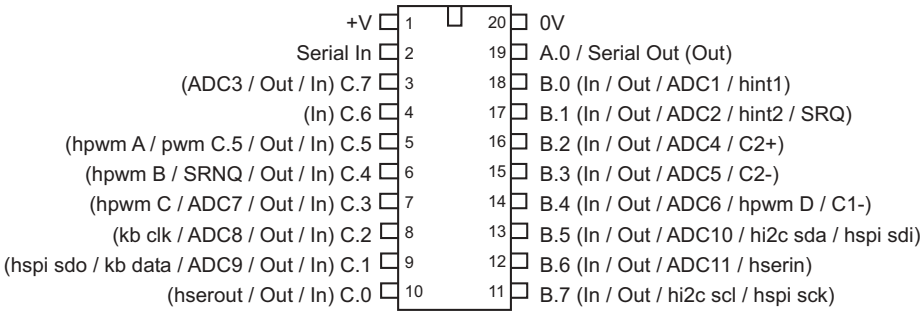
PICAXE-20M2



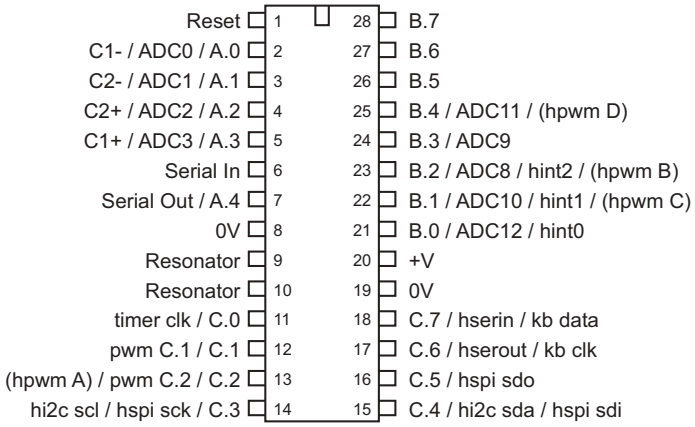
(14M2 and 20M2 are future parts, check availability)

At a glance - pinout diagrams (X2 parts):

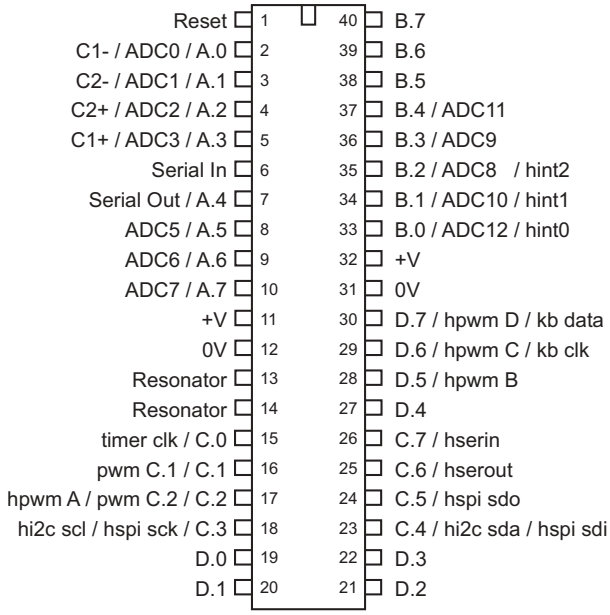
PICAXE-20X2



PICAXE-28X2

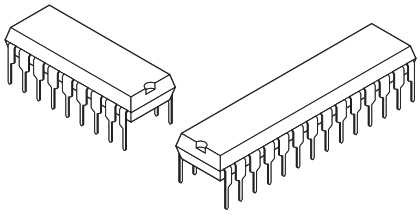


PICAXE-40X2



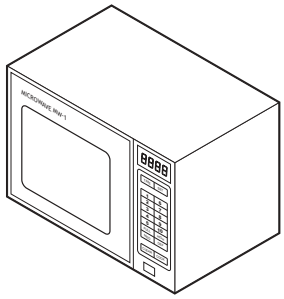
What is a microcontroller?

A microcontroller is often described as a 'computer-on-a-chip'.



It is a low-cost 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 build into a product to make the product more intelligent and easier to use.



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).

One microcontroller can often replace a number of separate parts, or even a complete electronic circuit.

Some of the advantages of using microcontrollers in a product design are:

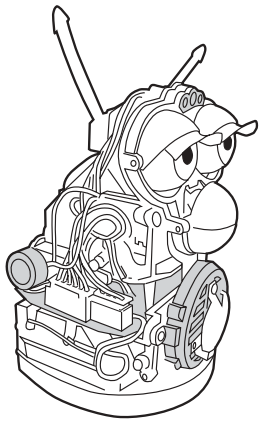
- increased reliability through a smaller part count
- reduced stock levels, as one microcontroller replaces several parts
- simplified product assembly and smaller end products
- greater product flexibility and adaptability since features are programmed into the microcontroller and not built into the electronic hardware
- rapid product changes or development by changing the program and not the electronic hardware

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!

In industry microcontrollers are usually programmed using the assembler or 'C' programming languages. However the complexity of these languages means that it is often not realistic for younger students in education, or many home hobbyists without formal training, to use these programming methods.

The PICAXE system overcomes this problem by use of a much simpler, easy to learn, BASIC programming language. Programs can also be created graphically by use of the flowchart editor.

Microcontrollers, input and outputs



A popular children's electronic toy is shown in the diagram. This is a good example of a *mechatronic* system, as it uses an electronic circuit to control a number of mechanisms. It also contains a number of sensors so that it can react to changes when it is moved (for example being put in a dark place or being turned upside down).

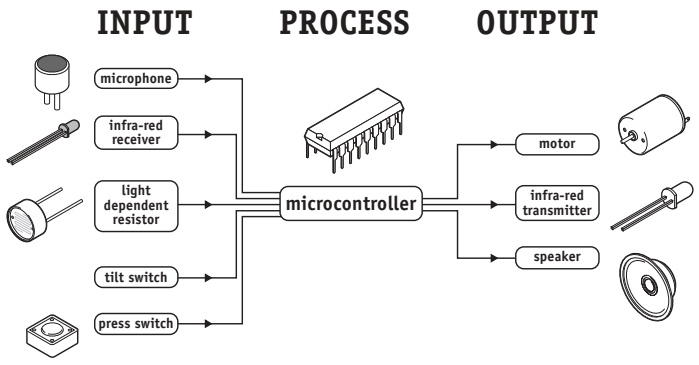
Input transducers are electronic devices that detect changes in the 'real world' and send signals into the process block of the electronic system.

Some of the input transducers for the electronic toy are:

- push switches on the front and back to detect when the toy is being 'stroked', and a switch in the mouth to detect when the toy is being 'fed'
- a light-dependent resistor (LDR) between the eyes to detect if it is light or dark
- a microphone to detect noises and speech
- a tilt switch to detect when the toy is being turned upside down
- an infrared detector to detect infrared signals from other toys

Output transducers are electronic devices that can be switched on and off by the process block of the electronic system. Some of the output transducers of the electronic toy are:

- a motor to make the eyes and mouth move
- a speaker to produce sounds
- an infrared LED (light-emitting diode) to send signals to other toys.



The microcontroller uses information from the input transducers to make decisions about how to control the output devices. These decisions are made by the control program, which is downloaded into the microcontroller. To change the 'behaviour' of the toy it is simply a process of changing and downloading a new program into the microcontroller

What is the PICAXE system?

The PICAXE system exploits the unique characteristics of the new generation of low-cost 'FLASH' memory based microcontrollers. These microcontrollers can be programmed over and over again (typically 100 000 times) without the need for an expensive programmer.

The PICAXE uses a simple BASIC language (or graphical flowcharts) that younger students can start generating programs with within an hour of first use. It is much easier to learn and debug than industrial programming languages (C or assembler code).

Unlike other BASIC 'module' based systems, all PICAXE programming is at the 'chip' level. Therefore instead of buying an expensive pre-assembled (and difficult to repair) surface mount module, with the PICAXE system you simply purchase a standard chip and use it directly in your project board.

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

The PICAXE 'Programming Editor' software is free and so the only cost per computer is the low-cost download cable. In the educational environment this enables students to buy their own cable and for schools to equip **every** single computer with a download cable. Other systems that require an expensive programmer or 'module' are generally too expensive to implement in this way.

Finally as the PICAXE chip never leaves the project board, all leg damage (as can occur when the chip is moved back and forth from a programmer) is eliminated.

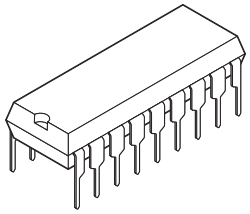
Building your own circuit / PCB

The PICAXE system has been designed to allow students / hobbyists to build their own PCB circuits for the PICAXE system. However if you do not wish to make your own circuit a number of project board kits and PCBs are available - please see the current PICAXE catalogue for more details.

If you wish to make your own PCB some reference designs are available at the PCB section of the PICAXE website at www.picaxe.co.uk
PCB samples are available for educational use in the popular realPCB and PCB Wizard formats.

If you wish to 'bread-board' a prototype circuit the AXE091 Development kit is highly recommended.

What is a PICAXE microcontroller?



A PICAXE microcontroller is a standard Microchip PICmicro™ microcontroller that has been pre-programmed with the PICAXE bootstrap code. The bootstrap code enables the PICAXE microcontroller to be re-programmed directly via a simple serial connection. This eliminates the need for an (expensive) conventional programmer, making the whole download system a very low-cost simple serial cable!

The pre-programmed bootstrap code also contains common routines (such as how to generate a pause delay or a sound output), so that each download does not have to waste time downloading this commonly required data. This makes the download time much quicker.

As the blank microcontrollers purchased to 'make' PICAXE microcontrollers are purchased in large volumes, it is possible for the manufacturer to program the bootstrap code and still sell the PICAXE microcontroller at prices close to standard catalogue process for single un-programmed PIC microcontrollers. This means the cost of the PICAXE microcontroller to the end user is very economical.

The PICAXE bootstrap code is not available for programming into blank microcontrollers. You must purchase PICAXE microcontrollers (rather than blank, un-programmed microcontrollers) for use in the PICAXE system.

PICAXE chip labels

PICAXE chips are pre-programmed and tested Microchip PICmicro™ microcontrollers.

More recent M2 parts are custom parts 'factory engraved' with the full PICAXE name. Older parts are simply 'engraved' with the Microchip part name.

PICAXE Type	*Engraving on top of chip*	
• PICAXE-08M	PIC12F683	
• PICAXE-14M	PIC16F684	
• PICAXE-14M2	PICAXE-14M2	*(future part, check availability)*
• PICAXE-18M2	PICAXE-18M2	
• PICAXE-20M	PIC16F677	
• PICAXE-20M2	PICAXE-20M2	*(future part, check availability)*
• PICAXE-20X2	PIC18F14K22	
• PICAXE-28X1	PIC16F886	
• PICAXE-28X2-5V	PIC18F2520	
• PICAXE-28X2-3V	PIC18F25K20	
• PICAXE-40X1	PIC16F887	
• PICAXE-40X2-5V	PIC18F4520	
• PICAXE-40X2-3V	PIC18F45K20	

Superseded older PICAXE chip

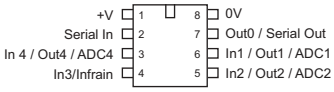
PICAXE Type	*Engraving*	*Replacement Type*
• PICAXE-08	PIC12F629	*Superseded by 08M*
• PICAXE-18	PIC16F627(A)	*Superseded by 18M2*
• PICAXE-18A	PIC16F819	*Superseded by 18M2*
• PICAXE-18M	PIC16F819	*Superseded by 18M2*
• PICAXE-18X	PIC16F88	*Superseded by 18M2*
• PICAXE-28A	PIC16F872	*Superseded by 28X1*
• PICAXE-28X	PIC16F873A	*Superseded by 28X1*
• PICAXE-40X	PIC16F874A	*Superseded by 40X1*

Which PICAXE chip?

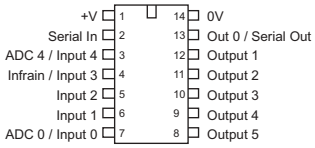
The PICAXE system can be used with different physical sizes of PICAXE chip (8, 14, 18, 20, 28 and 40 pin). The primary difference between the sizes of chips is the number of input/output pins available – the larger chips cost a bit more but have more available input/output pins. The same BASIC language is common to all size chips.

Within a chip size there are also different variants (e.g. for the 20 pin PICAXE the 20M and 20X2 variants are available). The principal difference between the variants is the amount of memory (ie how long a program can be downloaded into the chip). The higher specification variants also have some increased functionality (e.g. high resolution analogue inputs and i2c compatibility, as described in the next section). Any project can be upgraded to the next level variant at any point (e.g. if your program is too long for the variant of chip used) by simply replacing the microcontroller in your circuit with the upgraded variant. All upgraded variants are pin and program compatible with the lower specification device.

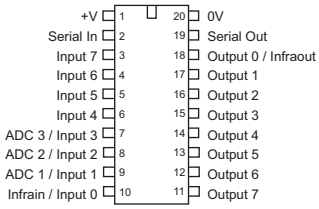
PICAXE-08M



PICAXE-14M



PICAXE-20M



The recommended part for new designs is:

Educational:

08	PICAXE-08M
14	PICAXE-14M (14M2 release in 2011)
18	PICAXE-18M2
20	PICAXE-20M (20M2 release in 2011)

Standard:

18	PICAXE-18M2
28	PICAXE-28X1
40	PICAXE-40X1

Advanced:

20	PICAXE-20X2
28	PICAXE-28X2
40	PICAXE-40X2

The following table shows the primary functional differences between the available PICAXE microcontrollers.

For general 'hobbyist' the M2 and X2 series are recommended.

Budget:	(40 - 200 line memory)			
08	5 inputs/outputs		1 low-res ADC	4MHz
08M	5 inputs/outputs		3 ADC	8MHz
Education:	(800 - 1800 line memory)			
14M2	11 configurable i/o		0-7 ADC	32MHz
18M2	16 configurable i/o		0-10 ADC	32MHz
20M2	16 configurable i/o		0-11 ADC	32MHz
Standard:	(800 - 1800 line memory)			
28X1	0-12 inputs	9-17 outputs	0-4 ADC	20MHz
40X1	8-20 inputs	9-17 outputs	3-7 ADC	20MHZ
Advanced:	(2000 - 3200 line memory in up to 4 separate slots)			
20X2	18 configurable i/o		0-8 ADC	64MHz
28X2	22 configurable i/o		0-8 ADC	40MHz
40X2	33 configurable i/o		0-10 ADC	40MHZ

All parts default to operation at 4MHz (8MHz for X2 parts). For use at higher speeds please see the 'setfreq' command in part 2 of the manual.

** 14M2 and 20M2 are future products - please check availability!*

The older 18, 18A, 18M, 18X parts are no longer manufactured as they have now been superseded by the 18M2 parts.
The older 28, 28A, 28X and 40X parts are no longer manufactured as they have now been superseded by the X1 parts.

Using the PICAXE system.

To use the PICAXE system you will require:

- A PICAXE microcontroller
- A PICAXE circuit board
- A power supply (e.g. 4 rechargeable AA cells (4.8V) or 3 alkaline AA cells (4.5V))
- A download cable (USB or serial)
- The free 'Programming Editor' software or 'AXEpad' software.

All these items are included within all the PICAXE 'starter' packs.

To run the Programming Editor software you require a computer running Windows XP or later. Any computer that runs the Windows operating system will work in textual 'BASIC' mode, however a Pentium processor or later is recommended for graphical flowchart work.

To run the AXEpad software you require a PC with a x386 Linux distribution or Mac with OSX (10.2 or later).

The computer also requires a USB port (for AXE027 USB cable) or 9 pin serial port for connecting the AXE026 serial download cable. See the USB/Serial Port setup section for more details.

PICAXE Starter Packs

To get started with the PICAXE system a starter pack is recommended. All 5 starter packs contain the same CDROM (containing the manuals and free programming software), USB (or serial) download cable and battery box. However the project board and type of PICAXE chip varies in each starter pack as indicated below.
3 x AA batteries are also required (not included).

PICAXE-08M Starter Pack (AXE003U)
PICAXE-08 protoboard, PICAXE-08M chip, CDROM, USB download cable and battery box. Self assembly kit.

PICAXE-14M Starter Pack (AXE004U)
PICAXE-20M Starter Pack (AXE005U)
PICAXE-14 projectboard, PICAXE-14M chip, CDROM, USB download cable and battery box. Self assembly kit.

PICAXE-18M2 Starter Pack (AXE002U)
PICAXE-18 standard project board, PICAXE-18M2 chip, CDROM, USB download cable and battery box. Pre-assembled (18M2 chip supplied).

PICAXE-28X1 Starter Pack (AXE001U)
PICAXE-28 project board, connector cables, PICAXE-28X1 chip, CDROM, USB download cable and battery box. Pre-assembled (28X1 chip supplied).

Development Starter Pack (AXE091U)
Specifically designed for hobbyists with large breadboarding area and inputs/ outputs for experimentation.
The development PCB can support all sizes of PICAXE chips and is supplied with a PICAXE-18M2 chip. Pre-assembled.

Tutorial Starter Pack (AXE050U)
The tutorial pack is designed for school use to enable students to rapidly learn the PICAXE language by a series of structured tutorials (provided on the CDROM). Pre-assembled board with LDR, switches and output display.

PICAXE Project Boards

Individual project boards/kits are also available for users who do not wish to manufacture their own pcb. All boards have the serial download connector for programming the PICAXE chip via the serial / USB download cable.

PICAXE-08 Proto Board (AXE021)
Small self-assembly board to allow rapid prototyping of PICAXE-08 circuits. The board provides the basic circuit and download connector, with a small prototyping area to allow connection of input and output circuits.

PICAXE-08 Motor Driver (AXE023)
The motor driver board can be used to drive 4 individual on/off outputs (e.g. buzzers), or the outputs can be used in pairs to allow forward-reverse-stop control of two motors. Pre-assembled with PICAXE-08 chip included.

PICAXE-14 Project Board (AXE117)
PICAXE-20 Project Board (AXE118)
The project board PCB is a professional quality PCB that enables students to construct a project board that has 6 outputs and 5 inputs. The board provides space for the PICAXE-14M2 chip, download socket and darlington driver. Self assembly kit (including PCB).

PICAXE-18 Project Board (CHI030)
The PICAXE-18 standard interface board is a pre-assembled board fitted with a darlington driver chip so that output devices such as motors and buzzers can be connected directly to the board. Supports 5 inputs and 8 outputs.

PICAXE-18 High Power Project Board (CHI035)
The pre-assembled high power interface board provides 4 FET drivers to drive high current output devices. By addition of the optional L293D motor driver chip, an additional 2 motor control outputs can be added.

PICAXE-28 Project Board (AXE020)
A pre-assembled board fitted with a darlington driver chip for 8 output devices. By addition of the optional motor driver chip, an additional 2 motor control outputs can be added to the board. Supplied with connector ribbon cables.

PICAXE-28/40 Proto Board (AXE022)
The PICAXE-28/40 protoboard kit allows rapid development of PICAXE-28X1/X2 and 40X1/X2 projects. The board provides the basic circuit and download connector, with connections for inputs & outputs. EEPROM socket included.

Each project board has it's own datasheet containing connection details, circuit diagram etc. These datasheets are accessed via the 'Help' menu of the software.

Software Installation

Computer Requirements:

To install the software you require a computer running Windows 95 or later with approximately 50MB free space. Any computer that runs the Windows operating system will work in textual 'BASIC' mode, however a Pentium 4 processor or later is recommended for graphical flowchart work.

Installation:

1) Start up and log into your computer (some operating systems require that you log in as 'Administrator' to install software).
2) Insert the CD, or download and run the installation file from the software page at www.picaxe.co.uk
3) Follow the on-screen instructions to install the software. On older computers you may be instructed to restart the computer after installation.
4) Insert the AXE026 cable into the 9 pin serial port at the back of the computer., or the AXE027 USB cable into the USB port. The AXE027 will require a software driver when first used, a 'New hardware found' wizard will automatically start (see the AXE027 datasheet for more details).
5) Click Start>Programs>Revolution Education>Programming Editor to start the software.
6) If the Options screen does not automatically appear, click the View>Options menu. On the 'Mode' tab select the size and type of PICAXE microcontroller you are using. On the 'Port' tab select the appropriate serial COM port then click OK.

You are now ready to use the system.

Installation on RM CC3 networks

The software will run on all school networks, including RM CC3.

1) It is recommended you use the uncompressed MSI install provided on the CDROM, rather than the internet download.
2) Log on as System Admin and use your preferred distribution software (e.g. RM Application Wizard) to build a distribution package using the MSI install found within the /progedit folder on the CDROM. If preferred you can also manually copy the MSI files into the RMPackages\Applications area.
3) Update the package list of the appropriate workstations using the RM Management Console and generate shortcuts as required.
4) XP users - note that you may have to create two Software Restrictions 'hash' rules - one to the progedit.exe executable and another to the shortcut. To do this log on as System Admin on an XP workstation, click Start>Programs>System Management>Software Restriction settings. Open Computer Configuration>Windows Settings>Software Restriction Policies>Additional Rules. From the Action menu select 'New Hash Rule' and browse to the progedit.exe executable. Click OK.
5) The default save/open folder paths can be edited as required in the file called network.ini found in the main installation folder.

Installing the AXE027 USB cable drivers

Many desktop computers have a 9 pin serial connector for connection of the PICAXE download cable. However some modern laptop computers do not have a 9 pin serial connector to save space, in this case the USB port must be used instead.

The USB interface system is an intelligent system that requires the connected device to automatically configure itself when connected to the computers USB port. Although it is theoretically possible to build a USB version of the PICAXE, the extra memory required would increase the cost of every single PICAXE chip by almost £3 ($5).

Therefore an alternate system is used. The user purchases a one-off low-cost 'USB to serial' cable (part AXE027), which is a special intelligent PICAXE cable that allows chips to be programmed via the USB port.

USB Cable Installation procedure:

(Please see the USB Cable (AXE027.pdf) help file for more detailed instructions. This is available on the software page at www.picaxe.co.uk or the \USB folder of the CDROM).

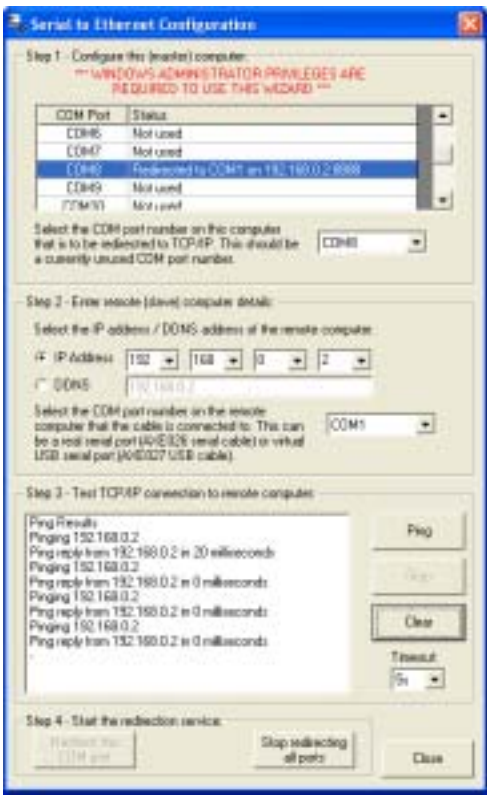
1) Purchase the AXE027 USB cable.
2) Connect to the USB port of the computer
3) Insert the CDROM supplied with the USB adapter to install the latest driver
4) Note the serial port COM number allocated to the USB adapter.
5) Connect the standard PICAXE cable to the USB adapter.
6) Start up the Programming Editor software and select the appropriate COM port from the View>Options>Port menu.
7) Click 'Refresh' to refresh the available port list.
8) Use the software and hardware as normal.

Notes:

- Windows 95 and NT do not support USB devices.

Downloading over a network using TCP/IP

The Programming Editor software supports COM port redirection over a TCP/IP "ethernet" connection. This connection can be a local network or even the internet.



To use this feature a 'virtual' COM port is created on the local computer (the computer that is running the Programming Editor software) and creates a TCP/IP connection. At the remote computer (where the download cable is connected to the USB/ serial port) a small redirection service application is installed and then redirects the real COM port to the TCP/IP connection.

This system allows the Programming Editor software to use the serial port on the remote computer exactly as if it was on the local computer - new program downloads and even serial data can be transmitted seamlessly back and forward over the TCP/IP connection.

To setup this connection two steps are required:
1) Run the wizard (PICAXE>WIzards>COM to TCP/IP menu) on the local computer to setup the local connection.
2) Install the SEC software on the remote computer and run it's Wizard to select the serial port to be used. This software runs as a service and so can be configured to always start when the computer is powered up. This allows it to be installed on unattended machines (e.g. in a museum).

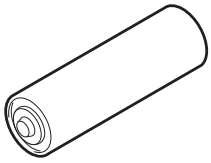
For further details please see the Serial Ethernet Connection software datasheet.

PICAXE Power Supply

All PICAXE chips will **run** programs at voltages between 3 and 5.5V DC.
The later generation parts (M2 and X2 parts) will also run down to 1.9V.

IMPORTANT NOTE - this manual describes use of the standard range (3-5.5V) parts. The 28X2 and 40X2 parts are also optionally available in special low power (1.8V to 3.3V) variants. Use of a 5V supply on a 3.3V part will permanently damage it!

It is recommended that the power supply is provided in one of the 3 following ways:



- 3 x AA alkaline AA cells (4.5V)
- 4 x rechargeable AA cells (4.8V)
- 5V regulated from a 9V DC regulated supply (5V)

Do not use a 9V PP3 battery, this is above the maximum rating of the PICAXE chip and will cause permanent damage. Note that most 3xAA and 4xAA battery boxes use the same 'press-stud' style connector and battery snap/clip as a PP3 9V battery. Note the provision of this style of clip does not mean that a project board should use a PP3 9V battery, it is just unfortunate that all battery boxes use the same style connector.

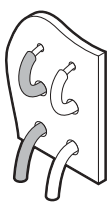
PP3 9V batteries are designed for very low-current, long term applications (e.g. a smoke alarm or multi-meter). Although a PP3 9V supply regulated to 5V will work for short periods with a microcontroller, it will drain very quickly when an output device (e.g. LED, motor or buzzer) is connected. Therefore always use AAA or AA battery packs rather than 9V PP3 batteries in microcontroller projects (as used with many portable consumer goods e.g. CD players, LED torches etc.) Take care when inserting PICAXE chips into your circuit to ensure they are the correct way around. Take extra care with 18 pin chips, as if inserted 'upside-down' the power supply connections will be reversed causing permanent damage to the chip.

AA Battery Packs

Alkaline AA cells have a nominal voltage of 1.5V, so 3 cells will give 4.5V. If you wish to use 4 cells, also use a 1N4001 diode in series with the battery pack. The diode provides voltage polarity protection, and as the diode has a 0.7V drop the microcontroller voltage will be an acceptable 5.3V (6V-0.7V).
Rechargeable AA cells have a nominal voltage of 1.2V, so 4 cells will give 4.8V.
Take care not to short circuit any battery pack, as the large short circuit current may cause considerable heat damage or start a fire.

Using battery snaps.

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

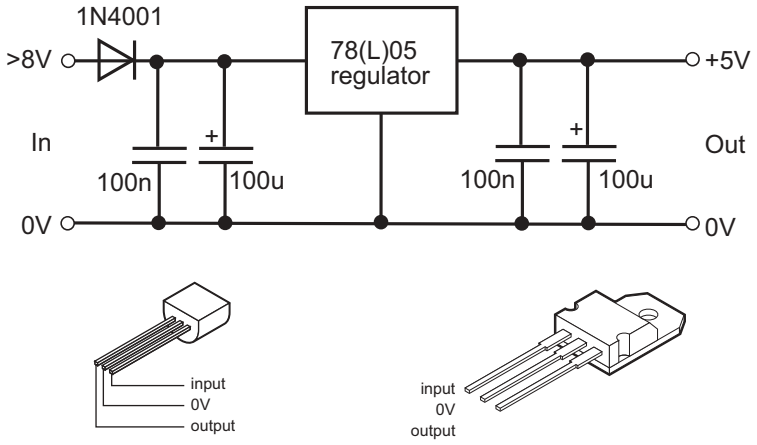


Regulated Power Supply.

Some users may wish to use a 'wall adapter' style power supply (e.g. part PWR009). It is essential that a good quality regulated 9V DC device is used with a 5V regulator. Unregulated devices may give excessive voltages (under low load conditions) that will damage the microcontroller. Old computer 12V/5V supplies are not suitable for PICAXE microcontroller work.

The 9V DC supply must be regulated to 5V using a voltage regulator (e.g. 7805 (1A capability) or 78L05 (100mA capability)). The full regulation circuit is shown below. The 1N4001 diode provides reverse connection protection, and the capacitors help stabilise the 5V supply. Note that voltage regulators do not generally function correctly unless the input supply in this circuit is approximately 8V or higher. The capacitors shown are also essential.

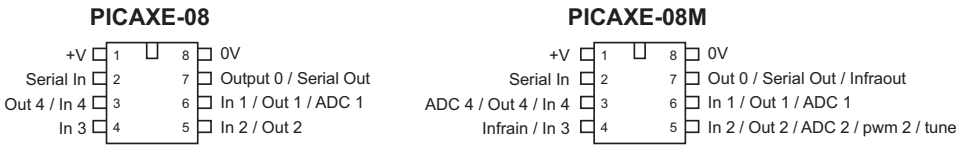
Never try to use a 9V PP3 battery with this circuit. The PP3 battery has insufficient current capability and is not recommended for any PICAXE project work.



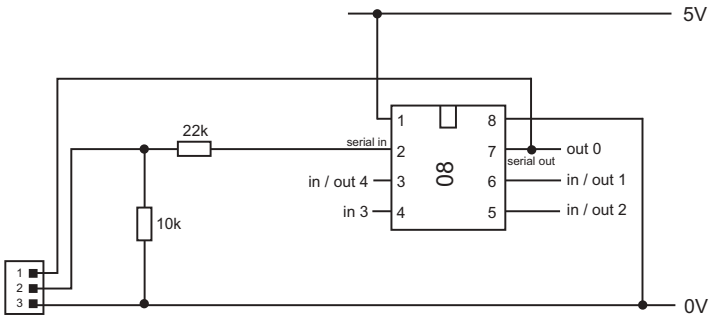
IMPORTANT NOTE - this manual describes use of the standard range (3-5V) parts. The X2 parts are also optionally available in special low power (1.8V to 3.3V) variants. Use of a 5V supply on a 3.3V part will permanently damage it!

PICAXE-08M/08 Pinout and Circuit

The pinout diagrams for the 8 pin devices are as follows:
(0.3" DIL or 150mil SOIC)



The minimum operating circuit for the 8 pin devices is:



See the Serial Download Circuit section of this manual for more details about the download circuit.

Notes:

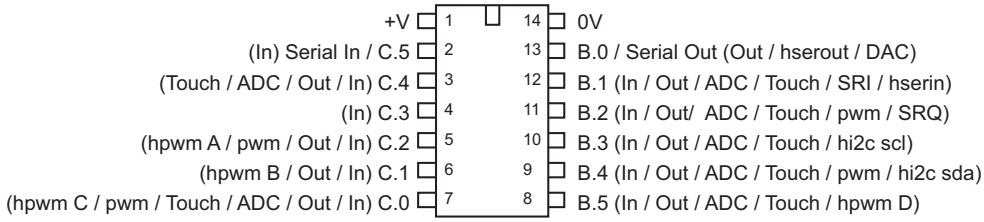
1) The 10k/22k resistors must be included for reliable operation.
   DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!
2) Output pin 0 (leg 7) is used during the program download, but can also be used as a general purpose output once the download is complete. On the project boards a jumper link allows the microcontroller leg to either be connected to the download socket (PROG position) or to the output (OUT position). Remember to move the jumper into the correct position when testing your program!

If you are making your own pcb you can include a similar jumper link or small switch, or you may prefer to connect the microcontroller leg to both the output device and the program socket at the same time. In this case you must remember that your output device will rapidly switch on and off as the download takes place (not a problem with simple outputs like LEDs, but could cause problems with other devices such as motors).

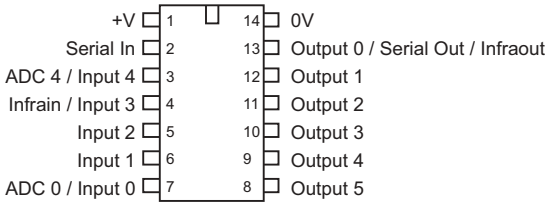
PICAXE-14M2/14M Pinout and Circuit

The pinout diagrams for the 14 pin devices are as follows:
(0.3" DIL or 150mil SOIC)

PICAXE-14M2

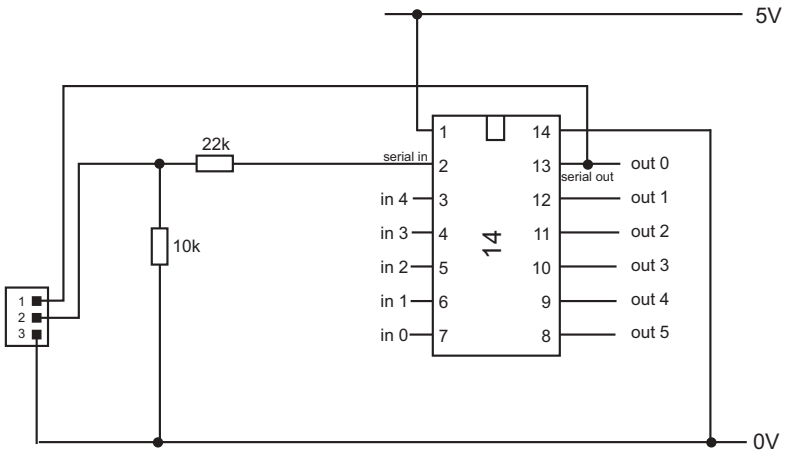


PICAXE-14M



14M2 is a future part - check availability!
Please see appendix C for information on how the 14M i/o pins can be reconfigured by advanced users.

The minimum operating circuit for the 14 pin devices is:



See the USB / Serial Download Circuit section of this manual for more details about the download circuit.

Notes:

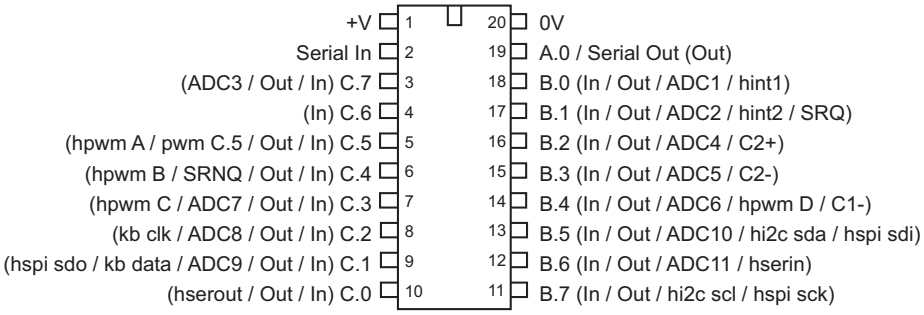
1) The 10k/22k resistors must be included for reliable operation.
   DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!
2) Output pin 0 (leg 7) is used during the program download, but can also be used as a general purpose output once the download is complete. On the project boards a jumper link allows the microcontroller leg to either be connected to the download socket (PROG position) or to the output (OUT position). Remember to move the jumper into the correct position when testing your program!

If you are making your own pcb you can include a similar jumper link or small switch, or you may prefer to connect the microcontroller leg to both the output device and the program socket at the same time. In this case you must remember that your output device will rapidly switch on and off as the download takes place (not a problem with simple outputs like LEDs, but could cause problems with other devices such as motors).

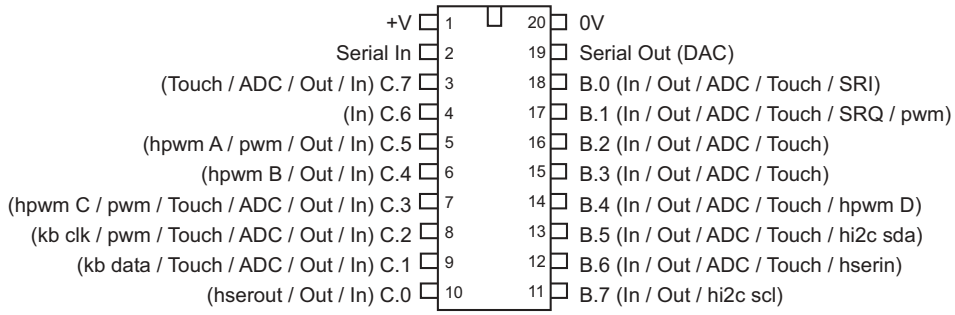
PICAXE-20X2/20M2/20M Pinout and Circuit

The pinout diagrams for the 20 pin devices are as follows:
(0.3" DIL or 300mil SOIC)

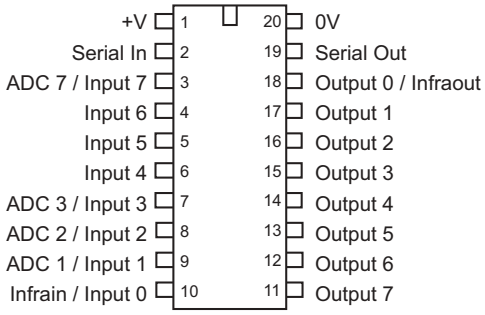
PICAXE-20X2



PICAXE-20M2



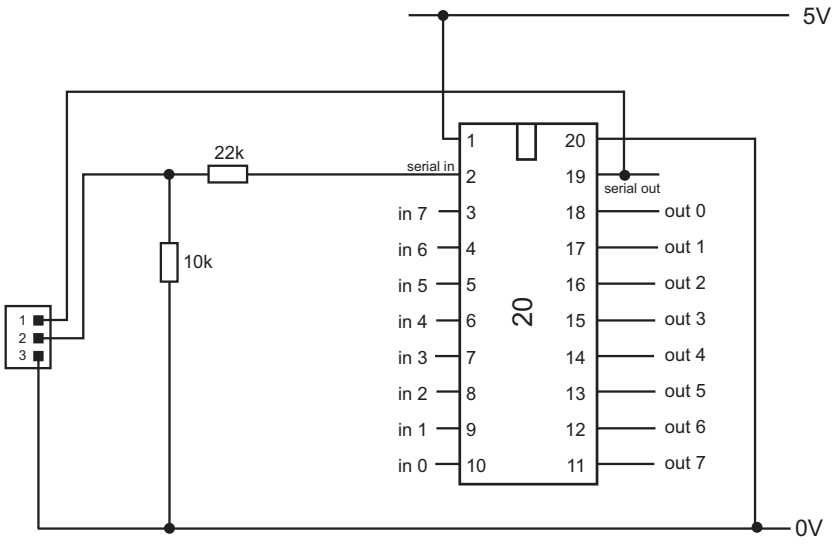
PICAXE-20M



Note pin C.6 is **input only** on the 20M2 and 20X2 parts. This is due to the internal silicon design of the chip and cannot be altered.

20M2 is a future part - check availability!

The minimum operating circuit for the 20 pin devices is:



See the USB / Serial Download Circuit section of this manual for more details about the download circuit.

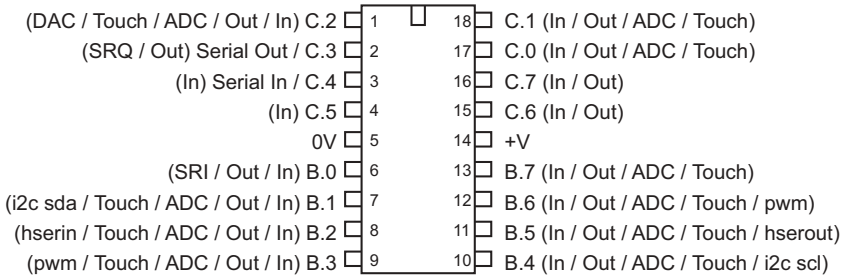
Notes:

1) The 10k/22k resistors must be included for reliable operation.
DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!

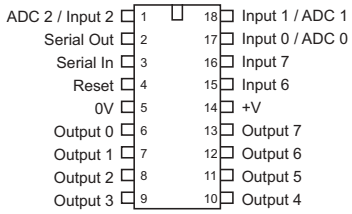
PICAXE-18M2/18X/18M/18A/18 Pinout and Circuit

The pinout diagrams for the 18 pin devices are as follows:
(0.3" DIL or 300mil SOIC)

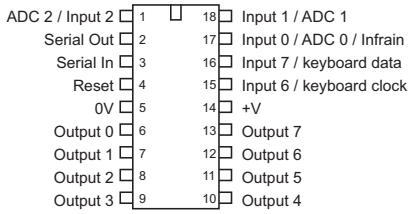
PICAXE-18M2



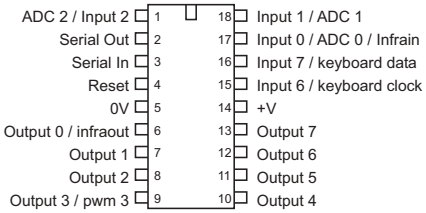
PICAXE-18



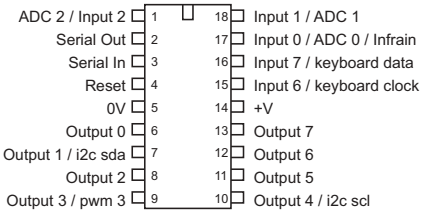
PICAXE-18A



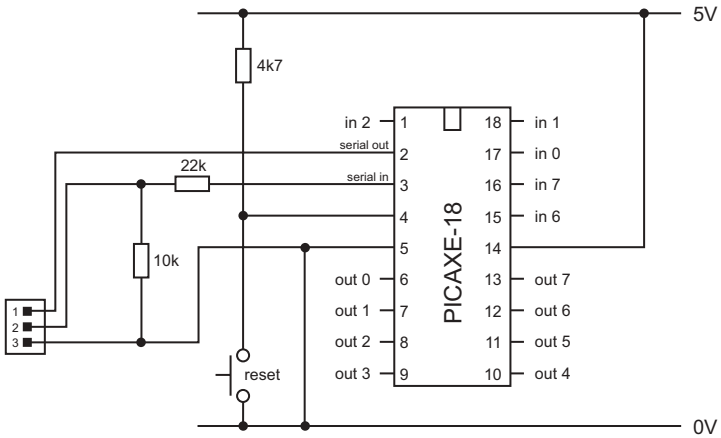
PICAXE-18M



PICAXE-18X



The minimum operating circuit for the 18 pin devices is:



See the USB / Serial Download Circuit section of this manual for more details about the download circuit.

Notes:

1) The 10k/22k resistors must be included for reliable operation.
DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!

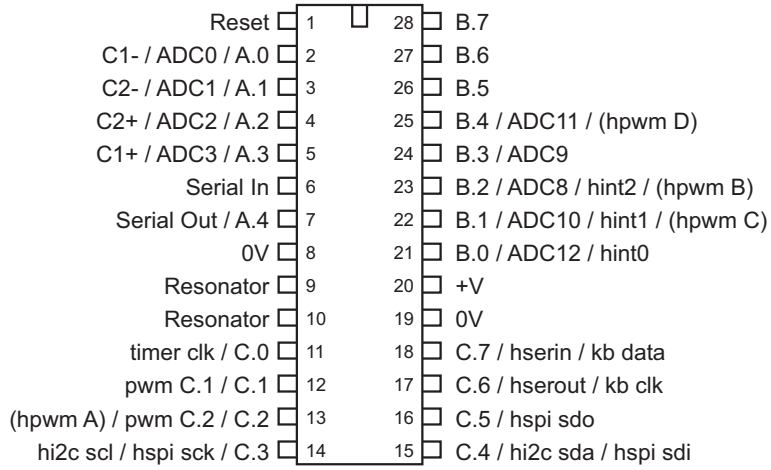
2) The reset pin must be tied high with the 4k7 resistor to operate on non-M2 parts. On 18M2 parts there is no reset pin, this is a general purpose input.

3) No external resonator is required as the chips have an internal resonator.

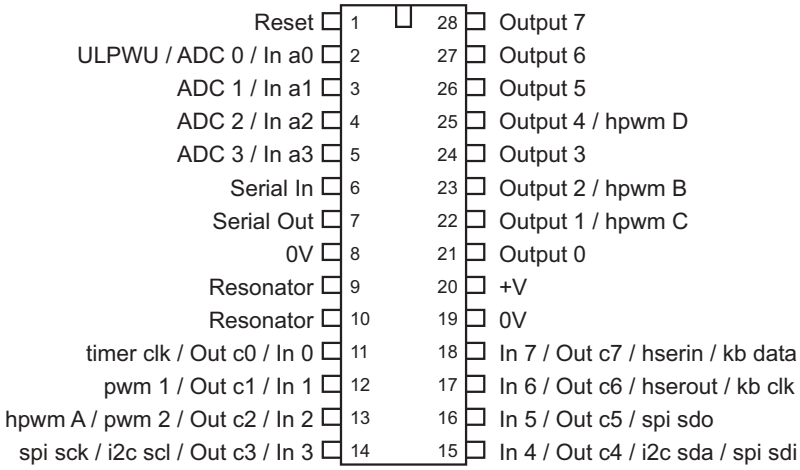
PICAXE-28X2/28X1/28X/28A Pinout and Circuit

The pinout diagrams for the 28 pin devices are as follows:
(0.3" DIL or 300mil SOIC)

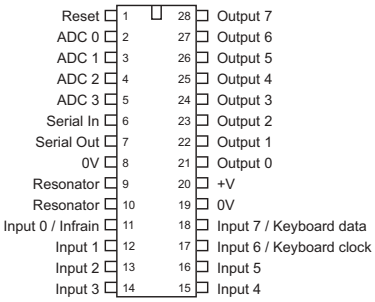
PICAXE-28X2



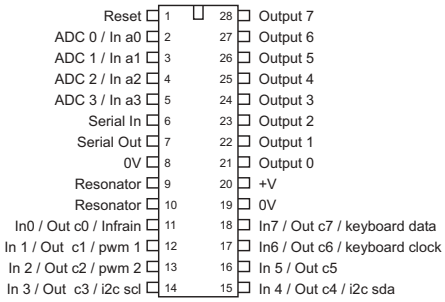
PICAXE-28X1



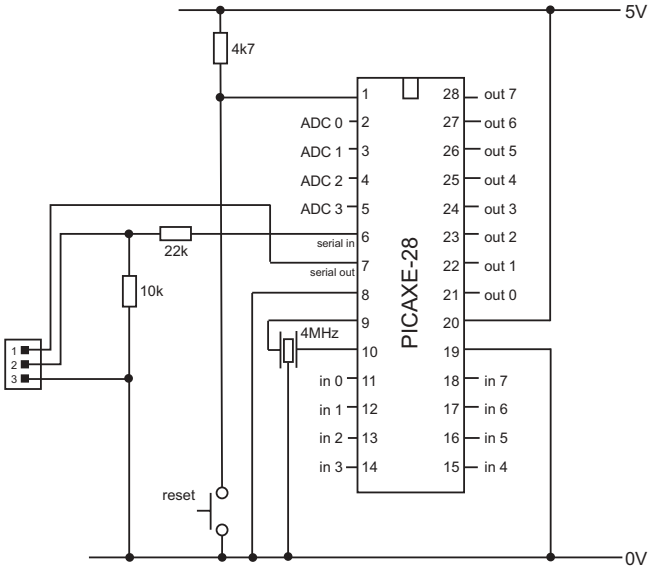
PICAXE-28A



PICAXE-28X



The minimum operating circuit for the 28 pin devices is:



See the USB / Serial Download Circuit section of this manual for more details about the download circuit.

Notes:

1) The 10k/22k resistors must be included for reliable operation.
DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!

2) The reset pin must be tied high with the 4k7 resistor to operate.

3) Resonator:

28X2-5V	(optional)	4 (16), 8(32), or 10(40) MHz
28X2-3V	(optional)	4 (16), 8(32), 10 (40) or 16(64) MHz
28X1	(optional)	16MHz
28X		4, 8 or 16MHz
28 / 28A		4MHz

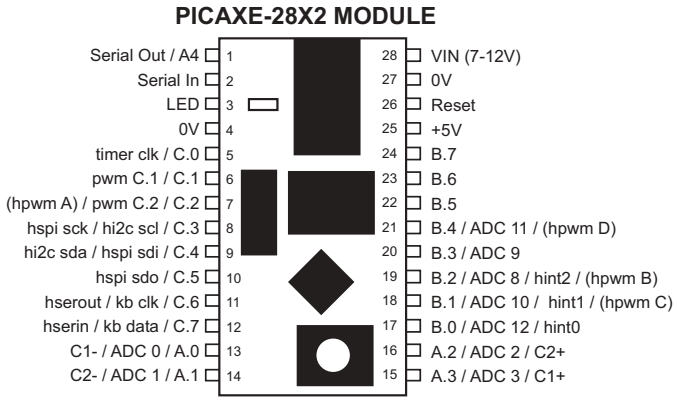
The 28X1 and 28X2 have an internal resonator (4 or 8MHz) and so the external resonator is optional. On 28A and 28X parts it is compulsory.
The 28X2 has an internal 4xPLL circuit. This multiplies the external clock speed by 4. Therefore an external 8MHz resonator gives an actual internal operating clock frequency of 4x8MHz=32MHz.

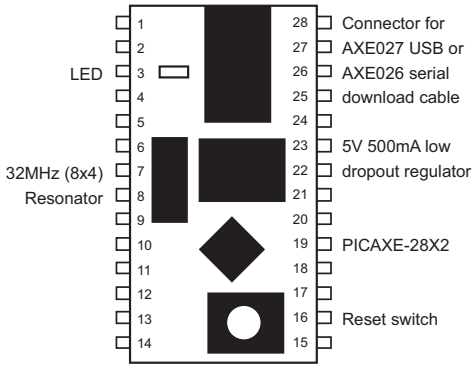
IMPORTANT NOTE - this manual describes use of the standard range (3-5V) parts. The X2 parts are also available in special low power (1.8V to 3.3V) variants. Use of a 5V supply on a 3.3V part will permanently damage it!

PICAXE-28X2 Module (AXE200)

28X2 Module - part AXE200

The 28X2 module is a complete PICAXE circuit in convenient 28 pin (0.6" wide) DIL package. The module is designed to be placed in a 'turned pin' style IC socket on the end user project board (e.g. socket part ICH028W).





Notes:
The module is supplied in a 28 pin carrier socket. It is highly recommended that the module is left in this socket at all times - ie use a separate socket on the project board. Then if a leg is accidentally snapped off the carrier socket it is possible to very carefully remove and replace the low-cost carrier socket.

Power can be supplied at 7-12V DC via pin 28. This is then regulated on-board via a 5V 500mA low drop out regulator. The 5V output is available at pin 25. Alternately a 4.5V or 5V supply can be connected directly to pin 25, leaving pin 28 unconnected.

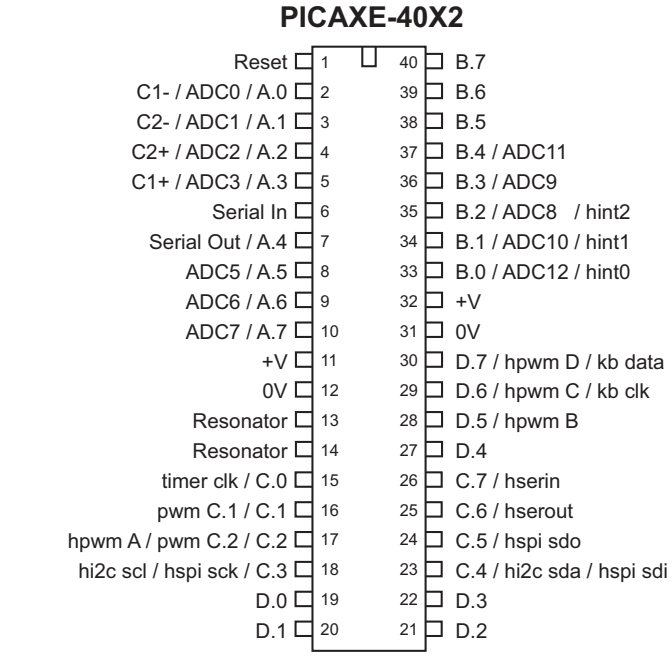
There is an on -board reset switch (with 4k7 pull up included on -board). The module can also be reset by connecting the reset pin (pin 26) to 0V.

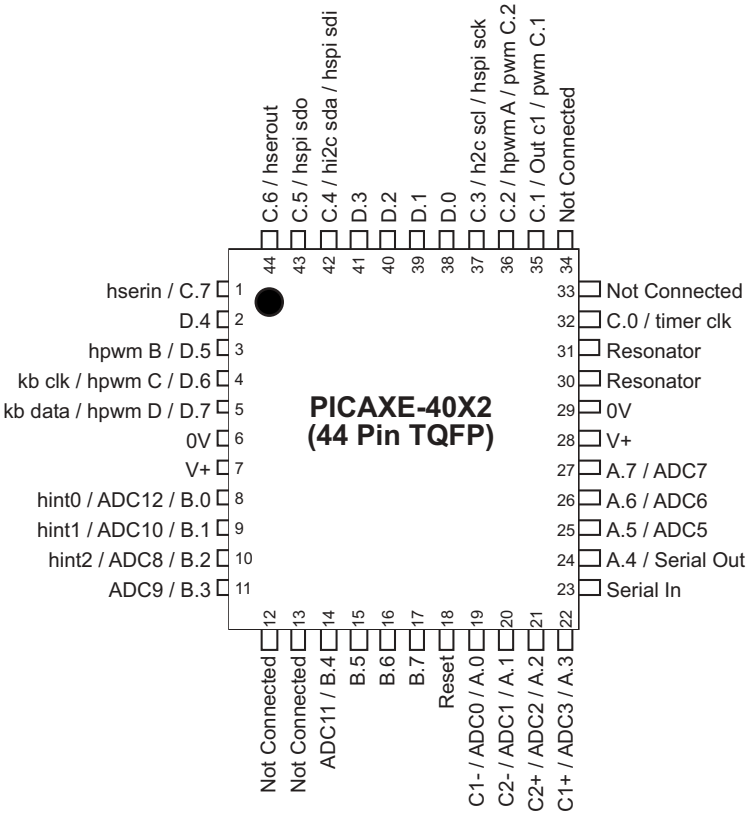
Download can be made via the on-board socket (AXE027 USB or AXE026 serial download cable) or via the Serial In / Serial Out pins.

The LED pin (pin 3) connects to an on-board LED/330R resistor which then connects to 0V. If left unconnected the LED does not operate, and hence draws no current (sometimes desirable in battery based systems). To use the LED as a power indicator simply connect the LED pin (pin 3) to 5V (pin 25). Alternately the LED pin can be connected to an output pin and hence controlled by high / low commands within the user program.

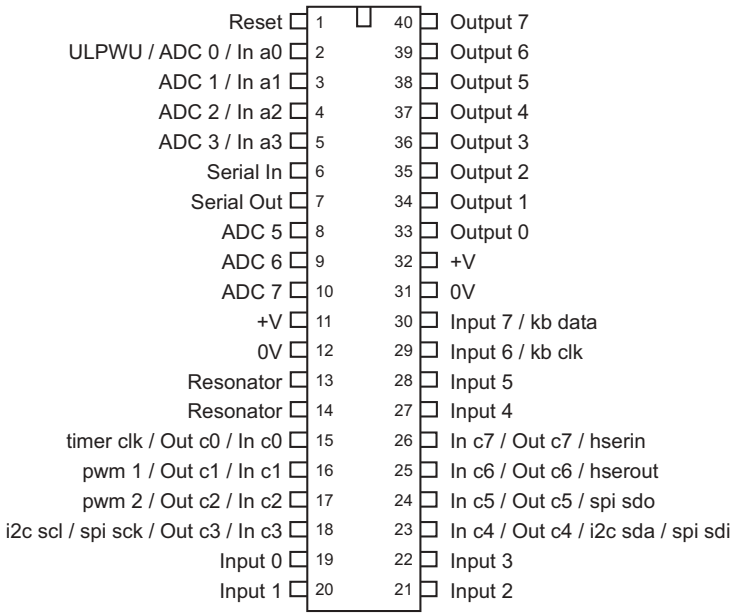
PICAXE-40X2/40X1/40X Pinout and Circuit

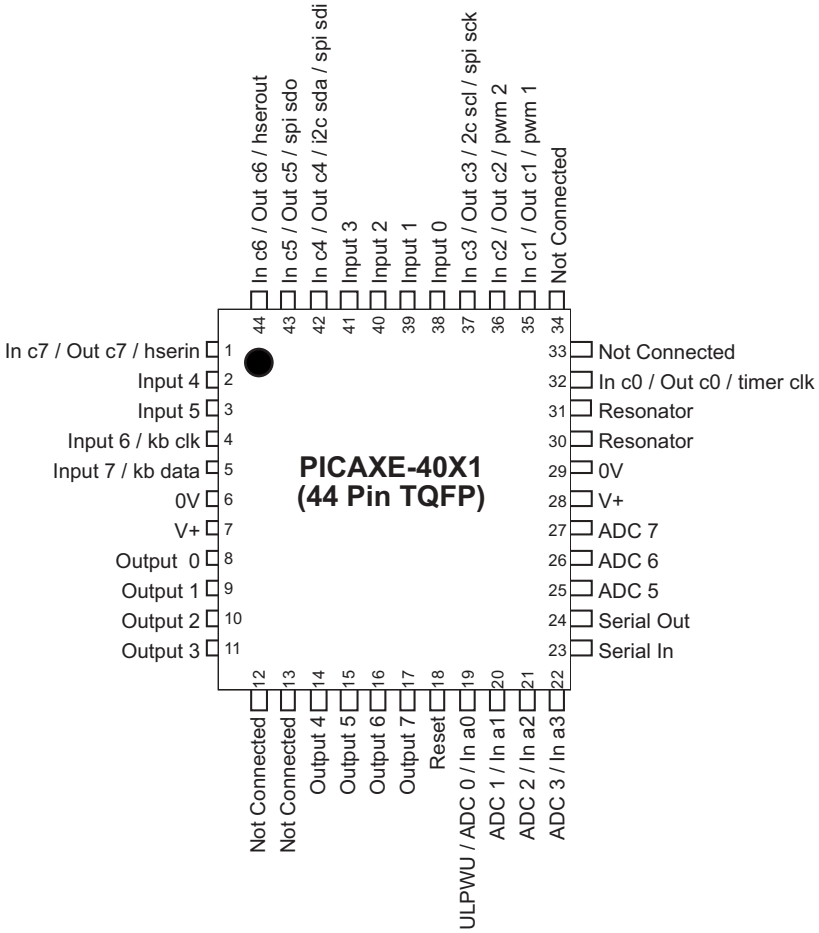
The pinout diagram for the 40 pin device is as follows:
(0.6" DIL or 44pin TQFP)



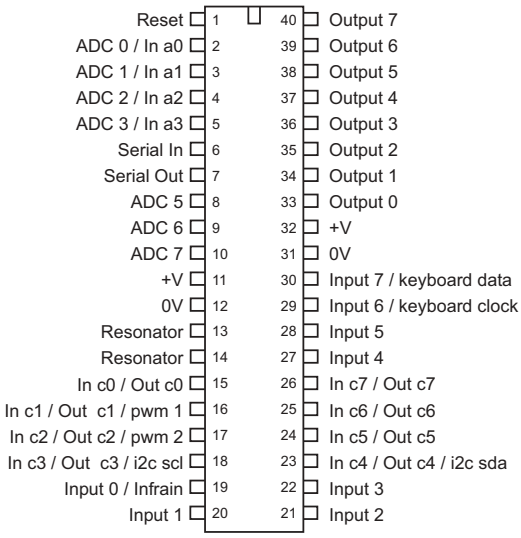


PICAXE-40X1





PICAXE-40X



The minimum operating circuit for the 40 pin device is the same as the 28 pin minimum circuit (altering the appropriate pin numbers as required).

See the USB / Serial Download Circuit section of this manual for more details about the download circuit.

Notes:

1) The 10k/22k resistors must be included for reliable operation.
DO NOT leave the serial in pin floating as THE PROGRAM WILL NOT RUN!
2) The reset pin must be tied high with the 4k7 resistor to operate.
3) Resonator:

40X2	(optional)	4 (16), 8(32), or 10(40) MHz
40X2-3V	(optional)	4 (16), 8(32), 10 (40) or 16(64) MHz
40X1	(optional)	16MHz
40X		4, 8 or 16MHz

The 40X1 and 40X2 have an internal resonator (4 or 8MHz) and so the external resonator is optional. On 40X parts it is compulsory.
The 40X2 has an internal 4xPLL circuit. This multiplies the external clock speed by 4. Therefore an external 4MHz resonator gives an actual internal operating clock frequency of 4x4MHz=16MHz.

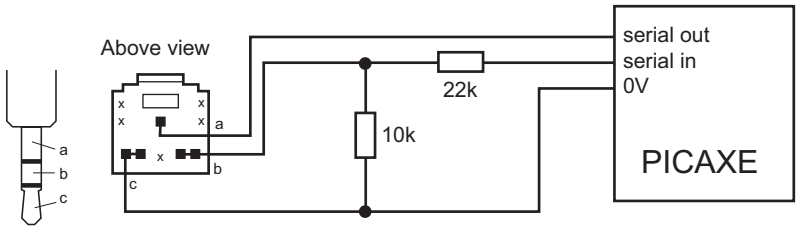
IMPORTANT NOTE - this manual describes use of the standard range (3-5V) parts. The X2 parts are also available in special low power (1.8V to 3.3V) variants. Use of a 5V supply on a 3.3V part will permanently damage it!

USB Download Circuit

The USB download circuit is identical for all PICAXE chips. It consists of 3 wires from the PICAXE chip to the AXE027 USB cable. One wire sends data from the computer to the serial input of the PICAXE, one wire transmits data from the serial output of the PICAXE to the computer, and the third wire provides a common ground.

Note this circuit can also be used for the AXE026 serial cable. Therefore the same circuit can be used with either USB or serial cable.

The minimum download circuit is shown here.



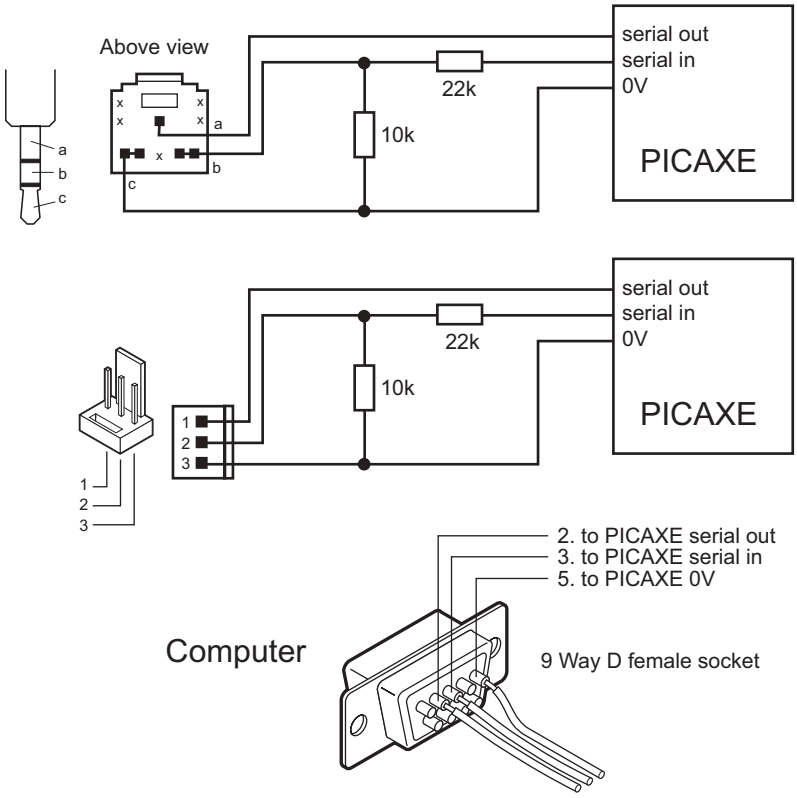
Note that the two resistors are not a potential divider. The 22k resistor works with the internal microcontroller diodes to clamp the serial voltage to the PICAXE supply voltage and to limit the download current to an acceptable limit. The 10k resistor stops the serial input 'floating' whilst the download cable is not connected. This is essential for reliable operation.

The two download resistors must be included on every PICAXE circuit (i.e. not built into the cable). The serial input must never be left unconnected. If it is left unconnected the serial input will 'float' high or low and will cause unreliable operation, as the PICAXE chip will receive spurious floating signals which it may interpret as a new download attempt.

Serial Download Circuit

The serial download circuit is identical for all PICAXE chips. It consists of 3 wires from the PICAXE chip to the AXE026 serial cable. One wire sends data from the computer to the serial input of the PICAXE, one wire transmits data from the serial output of the PICAXE to the computer, and the third wire provides a common ground. See the USB adapter section for details on how to use the USB port adapter.

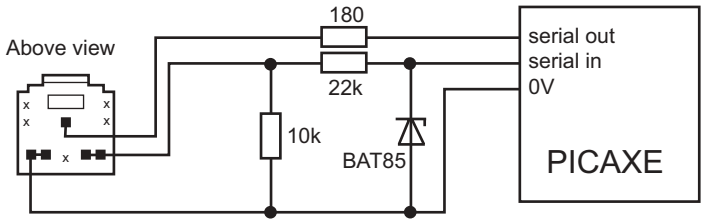
The minimum download circuit is shown here. This circuit is appropriate for most educational and hobbyist work.



Note that the two resistors are not a potential divider. The 22k resistor works with the internal microcontroller diodes to clamp the serial voltage to the PICAXE supply voltage and to limit the download current to an acceptable limit. The 10k resistor stops the serial input 'floating' whilst the download cable is not connected. This is essential for reliable operation.

The two download resistors must be included on every PICAXE circuit (i.e. not built into the cable). The serial input must never be left unconnected. If it is left unconnected the serial input will 'float' high or low and will cause unreliable operation, as the PICAXE chip will receive spurious floating signals which it may interpret as a new download attempt.

Enhanced Serial Download Circuit



The BAT85 Shokkty diode operate at a lower device voltage than the internal microcontroller diodes, providing a more accurate voltage reference. The additional 180R resistor provides additional preventative short circuit and static protection on the serial output pin.

Not required when the AXE027 USB cable is used.

Download Cables

The USB download cable (AXE027) is recommended for all modern computers. It is compatible with any computer with a USB port.

The standard serial download cable (part AXE026) consists of a 3.5mm stereo plug, which mates with a stereo socket (part CON039) on the project board. This type of connection is more robust and reliable than the Molex header in the educational environment.

Individual hobbyists may prefer a standard 3 pin Molex 0.1" (2.54 mm) 3 pin header. A matching cable (part AXE025) is available. This cable is not recommended for the educational environment.

All serial computer connection is via the serial port (9 pin D connector).If you have a very old computer with a 25pin serial port, you require a 25-9 pin adapter (part ADA010), which are also available from most high street computer stores.

Reset Circuit

All 18 (excluding 18M2), 28 and 40 pin PICAXE have a 'reset' pin. This pin must be in the high condition for the PICAXE microcontroller to function. If this pin is left unconnected the microcontroller will not operate reliably. To tie this pin high connect a 4.7k resistor between the reset pin and V+ supply rail (do not connect the pin directly to V+, always use a resistor). A reset switch is optional, but highly recommended. This should be a 'push to make' type and connected between the reset pin and 0V.

All 8, 14 and 20 pin PICAXE do not have a reset pin. Therefore to reset the microcontroller the power supply must be disconnected and then reconnected. Note that, when using capacitors in your supply circuit, these capacitors may hold enough charge to keep the microcontroller powered for several seconds after the power supply is disconnected.

Resonator

Different PICAXE chips have internal or external (or both) options:

PICAXE	INTERNAL	EXTERNAL
08, 18	4	-
'A' parts	4,8	-
'M' parts	4,8	-
'X' parts	4,8	-
'M2' parts	4,8,16,32	-
20X2	4,8,16,32,64	-
28A	-	4
28X	-	4,8,16
28X1	4,8	4,8,16
28X2-5V	4,8	4 (=16), 8 (=32), 10 (=40)
28X2-3V	4,8,16	4 (=16), 8 (=32), 10 (=40), 16 (=64)
40X	-	4,8,16
40X1	4,8	4,8,16
40X2-5V	4,8	4 (=16), 8 (=32), 10 (=40)
40X2-3V	4,8,16	4 (=16), 8 (=32), 10 (=40), 16 (=64)

All 28 and 40 pin PICAXE can use an external resonator (the resonator is internal within the 08, 14, 20 and 18 pin PICAXE). Note that the internal resonator within the 08,14,20 and 18 PICAXE is not quite as accurate as an external resonator. Although this does not cause any issues with the majority of projects, if a specialised project requires very high precision a 28 or 40pin PICAXE should be used.

An 3 pin ceramic resonator is recommended when required. This device consists of a resonator and two loading capacitors in a single 3 pin package. The centre pin is connected to 0V and the outer two pins to the two PICAXE resonator pins (the resonator can be used either way around).

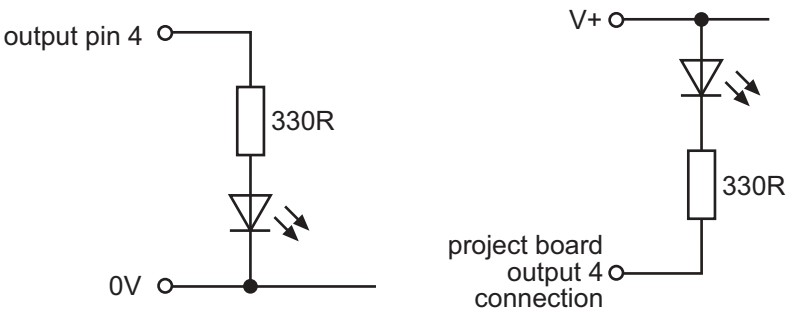
All parts default to 4MHz internal operation, apart from the X2 parts which default to 8MHz internal operation.

The 28X2 and 40X2 contain an internal 4xPLL circuit. This means that the internal operating frequency is 4x the external resonator frequency. The maximum speed of these devices is therefore 64MHz (using a 16MHz resonator).

If desired a 2 pin resonator, or 2 pin crystal, can be used with X, X1 or X2 parts. In this case two appropriate loading capacitors must also be used with the resonator/crystal. See the crystal manufacturer's datasheet for more information.

Testing the System

This first simple program can be used to test your system. It requires the connection of an LED (and 330R resistor) to output pin 4. If connecting the LED directly to a PICAXE chip on a proto (or home-made) board, connect the LED between the output pin and 0V. When using the project boards (e.g. as supplied within the 18 and 28 starter packs), connect the LED between V+ and the output connector, as the output is buffered by the darlington driver chip on the project board. (Make sure the LED is connected the correct way around!).



1. Connect the PICAXE cable to the computer USB/serial port. Note which port it is connected to (e.g. 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 the correct PICAXE type.
5. Click on the 'Serial Port' tab and select the serial port (virtual COM port for USB cable) that the PICAXE cable is connected to.
6. Click 'OK'
7. Type in the following program:

```
main: high 4
      pause 1000
      low 4
      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 4.
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 4 should flash on and off every second.

If your program does not download use the check list and hard-reset procedure described in the next two sections to isolate the mistake.

Hard-reset procedure

The download process involves the PICAXE microcontroller regularly checking the serial input line for a new download signal from the computer. This is automatic and not noticed by the PICAXE user. However there can be rare occasions when the PICAXE does not check the serial input line quickly enough whilst running its program. These situations can include:

- Corrupt program in PICAXE (e.g. if power or cable removed part way through a new download)
- Incorrect clock frequency (set by setfreq command)
- Pause or wait commands longer than 5 seconds used in program.
- Use of serin, infrain or keyin within program.

Fortunately it is very simple to resolve this issue, as the very first thing any PICAXE chip does on power reset is check for a new computer download. Therefore if you reset the PICAXE whilst a download is being started by the computer, the new download will always be recognised. This process is called a hard-reset.

To perform a hard-reset using the reset switch (28, 40 pin PICAXE):

1) Press and hold down the reset switch.
2) Click the PICAXE>Program menu to start a download.
3) Wait until the progress-bar appears on screen.
4) Wait 1 second then release the reset switch.

To perform a hard reset using the power supply (all sizes):

1) Disconnect the power supply.
2) Wait until all power supply decoupling capacitors have discharged (can take up to 30 seconds or more depending on circuit design).
3) Click the PICAXE>Program menu to start a download.
4) Wait until the progress-bar appears on screen.
5) Reconnect the power supply

Download CheckList

If you cannot download your program, check the following items. Remember that all new PICAXE are pre-programmed and tested, therefore if a new chip does not download it is generally a hardware setup issue.

If the program fails part way through a download this is generally a power supply issue (or loose cable connection). Try with 3 new alkaline cells giving exactly 4.5V.

PICAXE microcontroller

- Is the correct PICAXE chip correctly inserted in socket
- Is a PICAXE chip (not blank un-programmed PIC chip) being used.
- Is a damaged PICAXE chip being used (e.g. chip that has had over-voltage or reverse power supply applied)
- Is a smooth 4.5V to 5.5V DC supply correctly connected. TEST ON ACTUAL CHIP V+ and 0V pins with a multimeter!
- Is the reset pin connected to V+ via 4.7k resistor (18 / 28 / 40 pin chips)
- Is the correct 3 pin resonator connected if required (28 / 40 pin chips)
- Are the serial download 10k/22k resistors correctly connected.

Software

- Latest version Programming Editor installed (v5.3.0 or later, see software page at www.picaxe.co.uk for up to date information)
- Correct serial port selected (View>Options>Port menu).
- Correct resonator speed selected (if appropriate) (View>Options>Mode menu)
- No conflicting serial port software running on computer (in particular PDA type 'hotsync' software and interactive whiteboard software)

Serial Download Cable(part AXE026)

- Correctly wired download cable.
- Correctly wired download socket with 10k/22k resistors.
- All download socket pins correctly soldered to PCB.
- Download cable correctly connected between computer and microcontroller.
- Download cable inserted fully into socket.

USB Download Cable (part AXE027)

- USB cable configured to use correct serial port
- USB cable installed with correct driver (Vista / XP users - ensure you are using the correct XP specific driver (also valid for Vista), available from the software page at www.picaxe.co.uk)

USB adapter (part USB010)

- USB adapter configured to use correct serial port
- USB adapter installed with correct driver (XP users - ensure you are using the correct XP specific driver, available from the software page at www.picaxe.co.uk)

Understanding the PICAXE memory.

The PICAXE memory consists of three different areas. The amount of memory varies between PICAXE types.

Program Memory.

Program memory is where the program is stored after a new download. This is 'FLASH' rewritable memory that can be reprogrammed up to (typically) 100,000 times. The program is not lost when power is removed, so the program will start running again as soon as the power is reconnected.

It is not generally required to erase a program, as each download automatically over-writes the whole of the last program. However if you want to stop a program running you can use the PICAXE>Clear Hardware Memory menu to download an 'empty' program into the PICAXE.

On standard PICAXE chips (M2, X, X1) you can download around 600 to 1000 lines of BASIC code. On A or M revision parts you can download around 80 lines and on educational parts around 40 lines. X2 parts support up to 4 programs of 1000 lines. Note these values are approximate as different commands require different amounts of memory space. To check your memory usage use the PICAXE>Check Syntax menu option.

On X1 and X2 parts 256 bytes of the program memory can also be 'reserved' as a lookup table (e.g. for LCD messages). See the table / readtable commands in part 2 of the manual for more details.

Data Memory

Data memory is additional storage space within the microcontroller. The data is also not lost when power is removed. Each download resets all data bytes to 0, unless the EEPROM command has been used to 'preload' data into the data memory. See the EEPROM, read and write command descriptions for more details.

On the PICAXE-08 / 08M / 14M / 20M / 18 / 18M / 18M2 the data memory is 'shared' with the program memory Therefore larger programs will result is a smaller available data memory area.

On all other PICAXE chips the data and program memory are completely separate.

RAM (Variables)

The RAM memory is used to store temporary data in variables as the program runs. It loses all data when the power is removed. There are four types of variable - general purpose, storage, scratchpad and special function.

Variables are memory locations within the PICAXE microcontroller that store data whilst the program is running. All this information is lost when the microcontroller is reset.

For information about variable mathematics see the 'let' command information in part 2 of the manual.

General Purpose Variables.

There are 14 or more general purpose byte variables. These byte variables are labelled b0, b1 etc. Byte variables can store integer numbers between 0 and 255. Byte variables cannot use negative numbers or fractions, and will 'overflow' without warning if you exceed the 0 or 255 boundary values (e.g. 254 + 3 = 1) (2 - 3 = 255)

However for larger numbers two byte variables can be combined to create a word variable, which is capable of storing integer numbers between 0 and 65535. These word variables are labelled w0, w1 etc, and are constructed as follows:

w0 = b1 : b0
w1 = b3 : b2
w2 = b5 : b4
w3 = b7 : b6
w4 = b9 : b8
w5 = b11 : b10
w6 = b13 : b12 etc.

Therefore the most significant byte of w0 is b1, and the least significant byte of w0 is b0.

In addition bytes b0 and b1 (w0) are broken down into individual bit variables. These bit variables can be used where you just require a single bit (0 or 1) storage capability.

b0 = bit7: bit6: bit5: bit4: bit3: bit2: bit1: bit0
b1 = bit15: bit14: bit13: bit12: bit11: bit10: bit9: bit8

M2, X1 and X2 parts also support bit16-bit31 (b2-b3)

You can use any word, byte or bit variable within any mathematical assignment or command that supports variables. However take care that you do not accidentally repeatedly use the same 'byte' or 'bit' variable that is being used as part of a 'word' variable elsewhere.

All general purpose variables are reset to 0 upon a program reset.

Storage Variables.

Storage variables are additional memory locations allocated for temporary storage of byte data. They cannot be used in mathematical calculations, but can be used to temporarily store byte values by use of the peek and poke commands.

The number of available storage locations varies depending on PICAXE type. The following table gives the number of available byte variables with their addresses. These addresses vary according to technical specifications of the microcontroller. See the poke and peek command descriptions for more information.

'M2' parts	228	28 to 255 ($1C to $FF)
'M' parts	48	80 to 127 ($50 to $7F)
'A' parts	48	80 to 127 ($50 to $7F)
'X' parts	96	80 to 127 ($50 to $7F), 192 to 239 ($C0 to $EF)
PICAXE-20X2	72	56 to 127 ($38 to $7F)
PICAXE-28X1	95	80 to 126 ($50 to $7E), 192 to 239 ($C0 to $EF)
PICAXE-28X2	200	56 to 255 ($38 to $FF)
PICAXE-40X	112	80 to 127 ($50 to $7F), 192 to 255 ($C0 to $FF)
PICAXE-40X1	95	80 to 126 ($50 to $7E), 192 to 239 ($C0 to $EF)
PICAXE-40X2	200	56 to 255 ($38 to $FF)
PICAXE-08	none	

Scratchpad

PICAXE-20X2	128	0 to 127 ($00 to $7F)
PICAXE-28X1	128	0 to 127 ($00 to $7F)
PICAXE-28X2	1024	0 to 1023 ($00 to $3FF)
PICAXE-40X1	128	0 to 127 ($00 to $7F)
PICAXE-40X2	1024	0 to 1023 ($00 to $3FF)

Special Function Variables (SFR)

The special function variables available for use depend on the PICAXE type.

PICAXE-08 / 08M SFR

pins = the input / output port
dirs = the data direction register (sets whether pins are inputs or outputs)
infra = another term for variable b13, used within the infrain2 command

The variable pins is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented.

pins = x : x : x : pin4 : pin3 : pin2 : pin1 : x

The variable dirs is also broken down into individual bits.
Only valid bi-directional pin configuration bits are implemented.

dirs = x : x : x : dir4 : x : dir2 : dir1 : x

PICAXE-14M2/18M2/20M2 SFR

pinsB	- the portB input pins
outpinsB	- the portB output pins
dirsB	- the portB data direction register
pinsC	- the portC input pins
outpinsC	- the portC output pins
dirsC	- the portC data direction register
bptr	- the byte scratchpad pointer
@bptr	- the byte scratchpad value pointed to by ptr
@bptrinc	- the byte scratchpad value pointed to by ptr (post increment)
@bptrdec	- the byte scratchpad value pointed to by ptr (post decrement)
time	- the current time
task	- the current task

When used on the left of an assignment 'pins' applies to the 'output' pins e.g.

```
let outpinsB = %11000000
```

will switch outputs 7,6 high and the others low.

When used on the right of an assignment 'pins' applies to the input pins e.g.

```
let b1 = pinsB
```

will load b1 with the current state of the input pin on portB.

The variable pinsX is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented e.g.

```
pinsB =   pinB.7 : pinB.6 : pinB.5 : pinB.4 :
          pinB.3 : pinB.2 : pinB.1 : pinB.0
```

The variable outpinX is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented. e.g.

```
outpinsB =  outpinB.7 : outpinB.6  : outpinB.5 : outpinB.4 :
            outpinB.3 : outpinB.2 : outpinB.1 : outpinB.0
```

The variable dirsX is broken down into individual bit variables for setting inputs/ outputs directly e.g.

```
dirsB =   dirB.7 : dirB.6  : dirB.5 : dirB.4 :
          dirB.3 : dirB.2 : dirB.1 : dirB.0
```

See the 'Variables - General' section in Part 2 of the manual for more information about @bptr, @bptrinc, @bptrdec

PICAXE-14M/20M SFR (Not M2 parts)

pins = the input port when reading from the port
(out)pins = the output port when writing to the port
infra = a separate variable used within the infrain command
keyvalue = another name for infra, used within the keyin command

Note that pins is a 'pseudo' variable that can apply to both the input and output port.

When used on the left of an assignment pins applies to the 'output' port e.g.

```
let pins = %11000011
```

will switch outputs 7,6,1,0 high and the others low.

When used on the right of an assignment pins applies to the input port e.g.

```
let b1 = pins
```

will load b1 with the current state of the input port.

Additionally, note that

```
let pins = pins
```

means 'let the output port equal the input port'

To avoid this confusion it is recommended that the name 'outpins' is used is this type of statement e.g.

```
let outpins = pins
```

The variable pins is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented.

pins = x : x : x : pin4 : pin3 : pin2 : pin1 : pin0 (14M)
pins = pin7 : pin6 : pin5 : pin4 : pin3 : pin2 : pin1 : pin0 (20M)

The variable outpins is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented.

outpins = x : x : outpin5 : outpin4 :
outpin3 :out pin2 : outpin1 : outpin0 (14M)

outpins = outpin7 : outpin6 : outpin5 : outpin4 :
outpin3 :out pin2 : outpin1 : outpin0 (20M)

PICAXE-18 / 18A / 18M / 18X SFR (not 18M2)

pins = the input port when reading from the port
(out)pins = the output port when writing to the port
infra = a separate variable used within the infrain command
keyvalue = another name for infra, used within the keyin command

Note that pins is a 'pseudo' variable that can apply to both the input and output port.

When used on the left of an assignment pins applies to the 'output' port e.g.

```
let pins = %11000011
```

will switch outputs 7,6,1,0 high and the others low.

When used on the right of an assignment pins applies to the input port e.g.

```
let b1 = pins
```

will load b1 with the current state of the input port.

Additionally, note that

```
let pins = pins
```

means 'let the output port equal the input port'

To avoid this confusion it is recommended that the name 'outpins' is used is this type of statement e.g.

```
let outpins = pins
```

The variable pins is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented.

pins = pin7 : pin6 : x : x : x : pin2 : pin1 : pin0

The variable outpins is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented.

PICAXE-28 / 28A / 28X SFR

pins	= the input port when reading from the port
(out)pins	= the output port when writing to the port
infra	= a separate variable used within the infrain command
keyvalue	= another name for infra, used within the keyin command

Note that pins is a 'pseudo' variable that can apply to both the input and output port.

When used on the left of an assignment pins applies to the 'output' port e.g.

```
let pins = %11000011
```

will switch outputs 7,6,1,0 high and the others low.

When used on the right of an assignment pins applies to the input port e.g.

```
let b1 = pins
```

will load b1 with the current state of the input port.

Additionally, note that

```
let pins = pins
```

means 'let the output port equal the input port'

To avoid this confusion it is recommended that the name 'outpins' is used is this type of statement e.g.

```
let outpins = pins
```

The variable pins is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented.

pins = pin7 : pin6 : pin5 : pin4 : pin3 : pin2 : pin1 : pin0

The variable outpins is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented.

outpins = outpin7 : outpin6 : outpin5 : outpin4 :
outpin3 : out pin2 : outpin1 : outpin0

PICAXE-28X1 / 40X1 SFR

pins	= the input port when reading from the port
outpins	= the output port when writing to the port
ptr	= the scratchpad pointer
@ptr	= the scratchpad value pointed to by ptr
@ptrinc	= the scratchpad value pointed to by ptr (post increment)
@ptrdec	= the scratchpad value pointed to by ptr (post decrement)
flags	= system flags

When used on the left of an assignment 'outpins' applies to the 'output' port e.g.

```
let outpins = %11000011
```

will switch outputs 7,6,1,0 high and the others low.

When used on the right of an assignment 'pins' applies to the input port e.g.

```
let b1 = pins
```

will load b1 with the current state of the input port.

The variable pins is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented.

pins = pin7 : pin6 : pin5 : pin4 : pin3 : pin2 : pin1 : pin0

The variable outpins is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented.

outpins = outpin7 : outpin6 : outpin5 : outpin4 :
outpin3 : out pin2 : outpin1 : outpin0

The scratchpad pointer variable is broken down into individual bit variables:

ptr = ptr7 : ptr6 : ptr5 : ptr4 : ptr3 : ptr2 : ptr1 : ptr0

See the 'Variables - Scratchpad' section for more information about @ptr, @ptrinc, @ptrdec

The system 'flags' byte is broken down into individual bit variables. If the special hardware feature of the flag is not used in a program the individual flag may be freely used as a user defined bit flag.

Name	Special	Special function
flag0	-	*reserved for future use*
flag1	-	*reserved for future use*
flag2	-	*reserved for future use*
flag3	-	*reserved for future use*
flag4	-	*reserved for future use*
flag5	hserflag	hserial background receive has occurred
flag6	hi2cflag	hi2c write has occurred (slave mode)
flag7	toflag	timer overflow flag

PICAXE-20X2 / 28X2 / 40X2 SFR

pinsA	-the portA input pins
dirsA	- the portA data direction register
pinsB	- the portB input pins
dirsB	- the portB data direction register
pinsC	- the portC input pins
dirsC	- the portC data direction register
pinsD	- the portD input pins
dirsD	- the portD data direction register
bptr	- the byte scratchpad pointer
@bptr	- the byte scratchpad value pointed to by ptr
@bptrinc	- the byte scratchpad value pointed to by ptr (post increment)
@bptrdec	- the byte scratchpad value pointed to by ptr (post decrement)
ptr	- the scratchpad pointer (ptrh : ptrl)
@ptr	- the scratchpad value pointed to by ptr
@ptrinc	- the scratchpad value pointed to by ptr (post increment)
@ptrdec	- the scratchpad value pointed to by ptr (post decrement)
flags	- system flags

When used on the left of an assignment 'pins' applies to the 'output' pins e.g.

```
let pinsB = %11000000
```

will switch outputs 7,6 high and the others low.

When used on the right of an assignment 'pins' applies to the input pins e.g.

```
let b1 = pinsB
```

will load b1 with the current state of the input pin on portB.

The variable pinsX is broken down into individual bit variables for reading from individual inputs with an if...then command. Only valid input pins are implemented e.g.

```
pinsB  =    pinB.7 : pinB.6 : pinB.5 : pinB.4 :
            pinB.3 : pinB.2 : pinB.1 : pinB.0
```

The variable outpinX is broken down into individual bit variables for writing outputs directly. Only valid output pins are implemented. e.g.

```
outpinsB =  outpinB.7 : outpinB.6  : outpinB.5 : outpinB.4 :
            outpinB.3 : outpinB.2 : outpinB.1 : outpinB.0
```

The variable dirsX is broken down into individual bit variables for setting inputs/ outputs directly e.g.

```
dirsB =     dirB.7 : dirB.6  : dirB.5 : dirB.4 :
            dirB.3 : dirB.2 : dirB.1 : dirB.0
```

The byte scratchpad pointer variable is broken down into individual bit variables:

```
bptrl =     bptr7 : bptr6 : bptr5 : bptr4 : bptr3 : bptr2 : bptr1 : bptr0
```

See the 'Variables - General' section (manual part 2) for more information about @bptr, @bptrinc, @bptrdec

The scratchpad pointer variable is broken down into individual bit variables:

ptrl = ptr7 : ptr6 : ptr5 : ptr4 : ptr3 : ptr2 : ptr1 : ptr0
ptrh = xxxx : xxxx : xxxx : xxxx : xxxx : xxxx : ptr9 : ptr8

See the 'Variables - Scratchpad' section (manual part 2) for more information about

@ptr, @ptrinc, @ptrdec

The system 'flags' byte is broken down into individual bit variables. If the special hardware feature of the flag is not used in a program the individual flag may be freely used as a user defined bit flag.

Name	Special	Special function
flag0	hint0flag	hardware interrupt on pin INT0
flag1	hint1flag	hardware interrupt on pin INT1
flag2	hint2flag	hardware interrupt on pin INT2
flag3	hintflag	hardware interrupt on any pin 0,1,2
flag4	compflag	hardware interrpt on comparator
flag5	hserflag	hserial background receive has occurred
flag6	hi2cflag	hi2c write has occurred (slave mode)
flag7	toflag	timer overflow flag

Parallel Task Processing

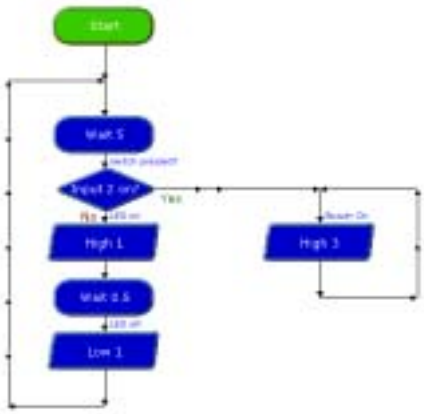
The M2 educational PICAXE chips support parallel task processing.

A PICAXE microcontroller is a single core controller and can therefore only process one instruction at any one time. The only exception to this rule is where the chip also contains a separate on-chip peripheral which runs independently of the main core. The main example of a separate peripheral is the pwm peripheral, which is activated via the pwmout command. This generates pulses completely separately to the main processing task, and so requires none of the core processing time to keep working in the background.

However the new PICAXE microcontroller educational range (M2 parts) can now simulate 'parallel processing' by repeatedly switching between a number of tasks at very high speed. This is made possible by the increased operating speeds of the newer parts - for instance running 4 tasks at 16MHz is approximately equal to running one task at 4MHz. All tasks therefore 'appear' to be processed in parallel. Parallel tasks are designed for educational use to simplify programming by younger students. This is best demonstrated by example.

A student wants to build a bicycle alarm. Every 5 seconds an LED briefly flashes to indicate the alarm is active. However when the chain (wire) is cut a buzzer must immediately sound.

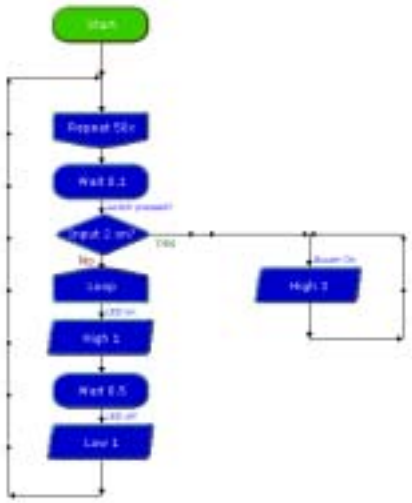
```
start0:
      pause 5000
      if pinC.2 = 1 then alarm
      high B.1
      pause 500
      low B.1
      goto start0
alarm:
      high B.3
      goto alarm
```



The student's first attempt at programming is shown in flowchart 1 above. This is the obvious solution, but does not work as expected. This is because the alarm does not sound immediately when the wire is cut - as the input is only checked once every 5 second there can be up to 5 seconds 'lag' before the alarm sounds.

The correct single task solution is to break the large 5 second delay into much smaller 'chunks' e.g. 50 loops of 0.1s (100ms) as shown in flowchart 2 overleaf. Therefore the input is checked much more frequently and the alarm sounds almost instantaneously. However this solution is not so easy to understand and most students will not initially reach this solution without teacher guidance.

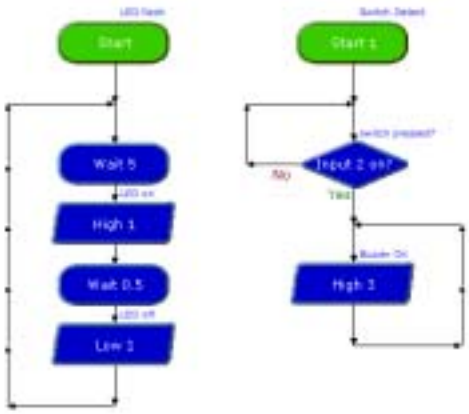
```
start0:
      for b4 = 1 to 50
         pause 100
         if pinC.2 = 1 then alarm
      next b4
      high B.1
      pause 500
      low B.1
      goto start0
alarm:
      high B.3
      goto alarm
```



Flowchart 3 below shows a simpler way to achieve the correct outcome using parallel tasks. Task 1 simply flashes the LED whilst task 2 checks the switch. In this solution the input is detected even faster than with the single task solution above. This parallel task solution is also generally easier for students to understand than flowchart 2.

```
start0:
      pause 5000
      high B.1
      pause 500
      low B.1
      goto start0

start1:
      if pinC.2 = 1 then alarm
      goto start1
alarm:
      high B.3
      goto alarm
```



How Parallel Tasks Operate.

All M2 parts can operate in single task mode or parallel task mode. In single task mode the M2 part operates as a traditional PICAXE part and follows the program sequentially (line by line) as expected.

By inclusion of additional 'start' labels within the user flowchart/BASIC program the compiler automatically switches the downloaded program into parallel task mode instead. In parallel task mode there are two (or more) program start positions and the PICAXE chip starts all tasks after a reset.

The commands are processed in a circular manner, for example with two tasks the first command in task 0 is processed, then the first command in task 1 is processed, then the second command in task 0 is processed and so on. Therefore

the processing core 'cycles' between the different tasks. During task 'dead' processing time (e.g. during delays such as within a pause command) the core automatically realises that there is no current processing to be carried out within that task and moves immediately onto the next task. Therefore the response of another task is not affected by a pause delay.

All variables/user RAM/EEPROM are shared between all tasks. Therefore, if required, tasks can interact and influence each other's behaviour by transferring data in particular bytes.

Multi task mode labels
Each task can be any length. The only restriction is that all tasks must fit in the total program memory area of that M2 chip.

The start ("top") of the program is always task 0. This is the first task that is started when the chip is reset. If desired an optional 'start0:' label may be used, but this is also implied by default if not used. If used, 'start0:' must always be on the very first line of program code.

Within BASIC programs the second task, task 1, is indicated by use of the 'start1:' label. Likewise task 2 is indicated by 'start2:' and so on. Within Logicator flowcharts a new 'start' cell is simply dragged onto the flowchart to indicate the new start positions.

The compiler automatically recognises the extra 'start' labels and therefore switches the part into multi-task mode upon the new program download.
Each task has its own program counter and stack. Therefore sub-procedures can be shared between different tasks if required, however this is not generally recommended. There is one interrupt (setint command) which will interrupt all current tasks.

Suspending tasks
It is possible to disable tasks during the program by use of the 'suspend' command. A suspended task can later be resumed by a 'resume' command within a different task.

The task currently being processed is stored in a special user variable called 'task'. The 'task' variable is updated very time the core switches to a new task. Therefore the command 'suspend task' will always suspend the current task. To have a particular task suspended at reset simply make sure that 'suspend task' is the first command within that task.

Take care not to suspend all tasks at the same time, or no processing will take place! A particular task can also be restarted by using the 'restart' command. Note that 'restart' does not reset the entire chip (use the 'reset' command to do this), so variables etc. are not cleared by the 'restart' command.

'Sleep' and 'nap' commands switch off the core and place the chip into low power mode. Therefore a nap or sleep command within any task will suspend all of the tasks.

BASIC simulation

When simulating BASIC program the 'Programming Editor' software process all tasks in parallel as expected, but only normally 'traces' (highlights on screen) the BASIC code for one task at a time to avoid confusion. Task 0 is traced by default, however a different task can be traced by including a '#simtask X' directive within the program. This directs the software to trace task X instead. '#simtask all' is also accepted and will rapidly highlight the line being processed in each task, which can be confusing to watch!

Limitations.

Parallel tasking is primarily designed for simpler educational projects. It works very well using simple input/output commands and programs that contain pause commands within the tasks. This covers the vast majority of school educational projects.

Parallel tasking is not designed for complex parallel tasks with each task trying to use different advanced features simultaneously e.g. trying to use serial / infra-red / 1-wire communication protocols simultaneously in 3 different tasks! In this situation the end user should use a single core program to process each advanced feature in turn.

Commands that require total core processing to maintain critical timing integrity (e.g. readtemp, sertxd, debug, serin, irin etc.) will 'block' the parallel tasking until that command has finished/timed out. Therefore the other tasks will appear to momentarily 'hang' during the processing of that command.

Due to the task cycling the timing between each command in a particular task cannot be guaranteed, because different length commands within the other tasks will be processed in the interval. This also means that the accuracy of pause commands will be slightly decreased. If a program sepcifically requires high timing accuracy a single task should be used instead.

The 'setfreq' command is not available in parallel task programs, as the core will automatically switch the frequency as required to maintain a high parallel task processing speed. However most commands will 'appear' to be operating at 4MHz, so commands such as pulsout/pulsin, serout/serin, count etc. should be calibrated as for 4MHz operation. Background servo pulsing will continue to operate, but may have a decreased accuracy with occasional 'twitches'. The change in background frequency may also affect background pwm pulse generation, so it is recommended that single task mode is used for programs containing pwmout commands.

Processing multiple tasks is much more complex than a single task and so in parallel task mode the core requires use of additional dedicated RAM memory. Therefore, on the 18M2 in parallel task mode only, bytes 128 to 255 of the user RAM are reserved as additional RAM for use by the core. Bytes 0-127 are still available to the end user via peek/poke commands. In parallel task mode the byte pointer (bptr) on the 18M2 will therefore overflow back to 0 after 127 (it overflows at 255 in single task mode). This does not apply to other M2 chips (e.g. 14M2, 20M2) as the silicon of these later developed chips was designed to include more RAM for this purpose.

Flowchart or BASIC?

The **Programming Editor** and **AXEpad** software support textual BASIC programming.Programming Editor also has a legacy simple flowcharting interface, but this is no longer developed as it has been superseded by the more user friendly Logicator flowcharting software.

The **Logicator for PIC micros** software provides a graphical flowchart method of programming. It is widely used within schools.

All programming methods use the same BASIC commands and syntax. The flowchart method simply provides a graphical way of joining the BASIC commands together, to save typing in programs. Flowcharting uses a smaller sub-set of the BASIC commands, and is generally used by younger students in the educational environment.

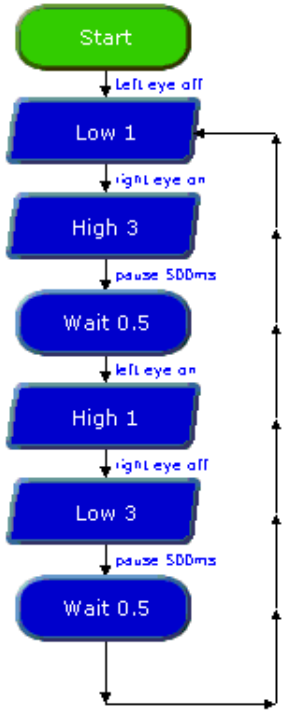
One advantage of flowchart programming is the very graphical on-screen simulation. This allows students to 'see' their program in operation before downloading to the PICAXE.

Most hobbyist and experienced educational users prefer the textual BASIC method of programming. It is much more powerful than flowcharts, which can become very complicated for large programs.

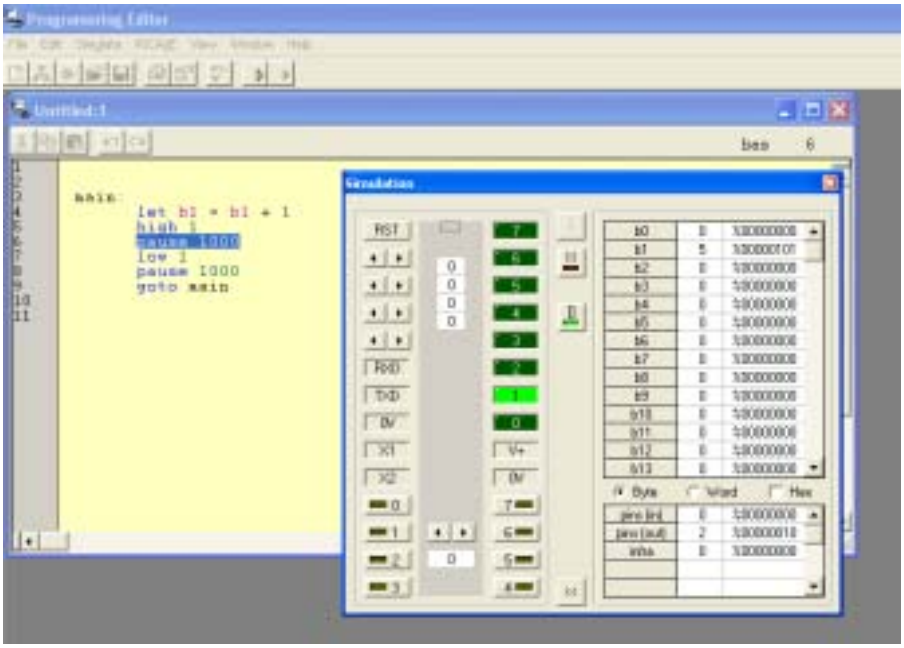
All flowcharts are automatically converted into BASIC programs prior to download to the PICAXE microcontroller. Therefore the main focus of this manual is on textual BASIC programming.

Logicator has its own, separate, instruction manual. For further information on the (now discontinued) simple flowchart programming method within Programming Editor, please see the flowchart appendix at the end of this manual.

```
main:
      low 1
      high 3
      pause 500
      high 1
      low 3
      pause 500
      goto main
```



BASIC Simulation



Simulations are available when using the 'colour syntax' mode. This option is selected from the View>Options>Editor menu.

When a BASIC program has been entered, the simulation is started by clicking the Simulate >Run menu (or pressing <Ctrl> + <F5>)

The main simulation panel is always displayed during a simulation, but varies in appearance to match the current PICAXE chip mode (View > Options menu). However similar functions apply in each case.

Outputs are shown as numbered LED indicators. LED on indicates a logic level 1 .

Inputs are shown as buttons with LED indicator. To change condition simply click on the button. LED on indicates a logic level 1.

Analogue input values are shown in a grid and can be altered by the scroll up/ down buttons or by typing over the value directly (0-255). These are the values used by the 'readadc' command.

The 'generic' value operates in a similar manner (0-65535) and is used by the following commands as the input value: count, pulsin, readadc10, readtemp, readtemp12 etc.

The reset (RST) button resets the simulation (program flow starts again from top of the program and all variables are reset).

Program Flow Control and Breakpoints

Three buttons on the main simulation panel are shortcut buttons for the Simulation menu functions.

> start / stop the simulation

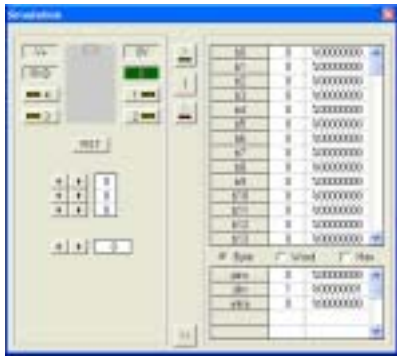
} single step through the simulation

| | pause the simulation at the current line

Breakpoints can be placed in (removed from) the program by simply clicking over the line number in the margin. Alternatively the Simulation > Toggle breakpoint menu may be used to insert/remove a breakpoint at the current cursor position. Breakpoints are indicated by a red bar in the margin.

To single step a program place a breakpoint on the first line you want to study and then click Run. From that point on use } to step through the program.

Display of the other available panels (upon simulation start) is determined by the Simulate > Simulation Panels options.



To change an input value click on the 'switch' beside the pin layout. To change an analogue value use the slider to alter the analogue value.

The 'generic' value is used to enter data for commands such as count, pulsin etc.

Variables Panel

The variables panel shows the current byte (b0, b1 etc) or word (w0, w1 etc) variable values. To change a variable value double click over the variable whilst the program is paused. You can then enter a new value.

Memory Panel

The memory panel displays the current values of the data EEPROM or SFR or scratchpad memory areas. For PICAXE chips that also store the user program in the data EEPROM the memory locations currently used by the program are indicated with a 'P'.

Serial Output Panel

The serial output panel displays output from the serout and sertxd commands. Debug commands are not simulated because the variable values are always available in the 'variables' panel.

Simulation Options

Use View>Options>Simulation menu to select the various simulation options.

Simulation Delay
This slider sets the time delay over each line as the program is simulated.

Highlight Labels
This option highlights and delays over labels that are on a line by themselves. This slows down the simulation but enables the user to clearly see which label has been jumped to.

Automatically Hide Panels
This option hides the panels as soon as the simulation ends. If unchecked the panels will remain on screen until program editing commences.

Beep
This option simulates sounds with a beep instead of a sound. This is only for use on older computers without a soundcard fitted.

Simulate LCD
Serout commands on the selected output will display a simulated LCD panel. This simulation matches the standard AXE033 or FRM010 serial LCD commands (custom characters, AXE033 clock and AXE033 alarm functions are not simulated).

Simulate EEPROM
Adds a simulated 24LC16B or 24LC256 EEPROM for i2c commands.

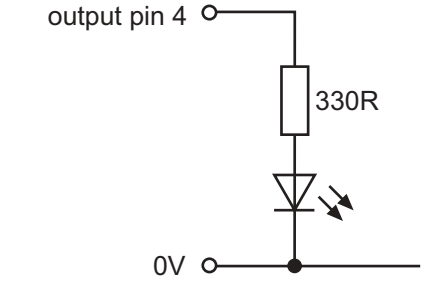
Simulate DS1307 RTC
Adds a simulated DS1307 real time clock for i2c commands. Time and date register reads use the values from the computers internal clock. Writes to change these time/date registers are ignored under simulation.

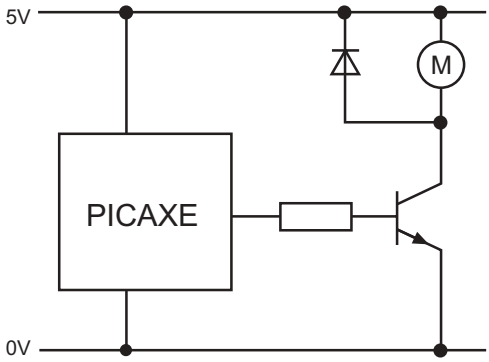
Interfacing Circuit Summary

This section provides a very brief overview of input/output interfacing to the PICAXE microcontroller. For more detailed explanations see section 3 of the manual - Interfacing Circuits. Section 3 provides detailed connection diagrams and sample programs for most common input / output transducers.

Digital Outputs

The microcontroller can sink or source 20ma on each output pin, maximum 90mA per chip. Therefore low current devices such as LEDs can be interfaced directly to the output pin. Higher current devices can be interfaced via a transistor, FET or darlington driver array.

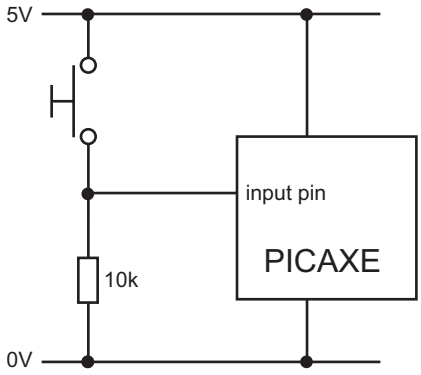




Digital Inputs

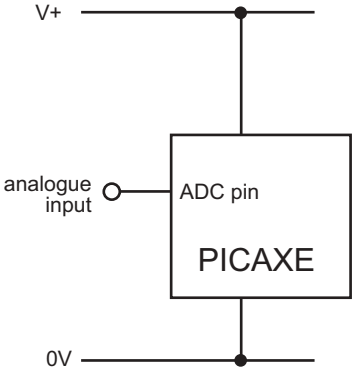
Digital input switches can be interfaced with a 10k pull down resistor. The resistor is essential as it prevents the input 'floating' when the switch is in th open position. This would give unreliable operation.

Note the 10k resistor is pre-fitted to the project board inputs.



Analogue Inputs

Analogue inputs can be connected in a potential divider arrangement between V+ and 0V. The analogue reference is the supply voltage, and the analogue signal must not exceed the supply voltage.



Tutorial 1 – Understanding and using the PICAXE System

The PICAXE chip, the 'brain' of the PICAXE system, when supplied new without a control program, does not do anything! The user must write a control program on the computer, and then download this program into the PICAXE chip. Therefore the PICAXE system consists of three main components:

The 'Programming Editor' software

This software runs on a computer and allows you to use the computer keyboard to type in programs in a simple BASIC language. Programs can also be generated by drawing flowcharts. Alternately third party software applications may be used (e.g. 'PIC-Logicator' or 'Crocodile Technology' software may be used to simulate complete PICAXE electronic circuits, programmed via flowcharts).

The Serial / USB Download Cable

This is the cable that connects the computer to the PICAXE system. The cable only needs to be connected when downloading programs. It does not have to be connected when the PICAXE is running because the program is permanently stored on the PICAXE chip – even when the power supply is removed!

The PICAXE chip and board

The PICAXE microcontroller chip 'runs' program that have been downloaded to it. However the chip needs to be mounted on an electronic board that provide connection to the microcontroller chip.

The electronic board can be designed by the user on a piece of stripboard or printed circuit board, or a pre-made interface or tutorial board may be used for speed and convenience.

Summary - Programming Procedure

1. Write the program on the computer using the Programming Editor software.
2. Connect the download cable from the computer to the PICAXE.
3. Connect the power supply to the PICAXE board.
4. Use the Programming Editor software to download the program. The download cable can then be removed after the download.

The program will start running on the PICAXE automatically. However the program can also be restarted at any time by pressing the reset switch (if available) or by cycling the power off and back on.

Input / Output Pin Naming Conventions

The first PICAXE chips had a maximum of 8 input and 8 output pins, so there was no need for a port naming scheme, as there was only one default input port and one default output port for each chip.

Therefore input and outputs pins were just refered to by their pin number

e.g.	*Output commands*	*Input Commands*
	high 1	count 2, 100, w1
	sound 2, (50,50)	pulsin 1, 1, w1
	serout 3, N2400, (b1)	serin 0, N2400, b3

However on later M2 and X2 PICAXE parts more flexibility was added by allowing almost all of the pins to be configured as inputs or outputs as desired. This creates more than 8 inputs or outputs and an amended naming scheme is therefore required. Therefore the pins on these parts are referred to by the new PORT.PIN notation. Up to 4 ports (A, B, C, D) are available, depending on chip pin count.

e.g.	*Output commands*	*Input Commands*
	high B.1	count A.2, 100, w1
	sound C.2, (50,50)	pulsin B.1, 1, w1
	serout A.3, N2400, (b1)	serin C.0, N2400, b3

In the case of if...then statements which check the status of the input pin variable, the naming convention of these input pin variables have changed in a similar style from

if pin1 =1 then...

to

if pinC.1 = 1 then...

The name of the input pins byte for each port is changed from

pins

to

pinsA, pinsB, pinsC, pinsD

The name of the output pins byte for each port is changed from

outpins

to

outpinsA, outpinsB, outpinsC, outpinsD

The name of the data direction register for each port is changed from

dirs

to

dirsA, dirsB, dirsC, dirsD

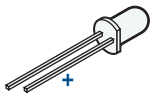
This manual generally uses the newer PORT.PIN format in the examples unless an example is specifically for an older part.

Please see the pinout diagrams for the chip you are using. Note that input / output pin numbers used within commands are not the same as the physical leg numbers!

Downloading a BASIC program

The following program switches output 4 on and off every second. When you download this program the LED should flash on and off every second.

```
main:
      high B.4
      pause 500
      low B.4
      pause 500
      goto main
```



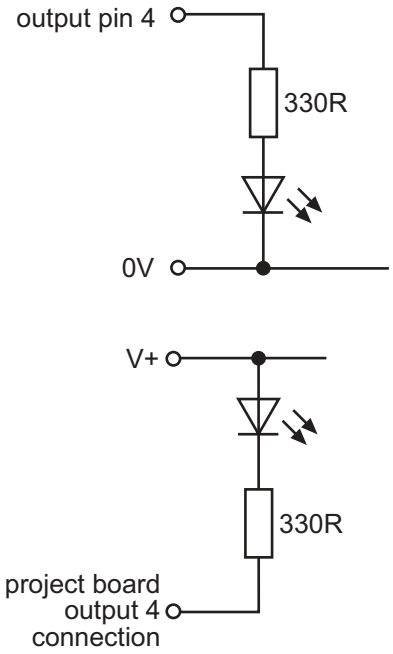
This program uses the **high** and **low** commands to control output pin 4, and uses the **pause** command to make a delay (1000 ms = 1 second).

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 / USB port. Note which COM port it is connected to.
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 the appropriate PICAXE chip.
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:

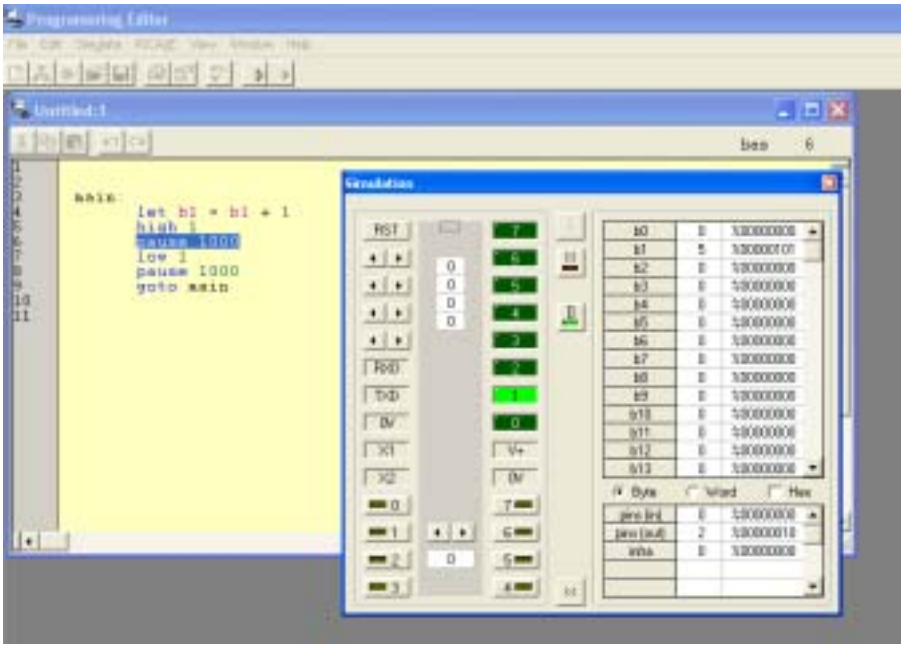
```
main:
      high B.4
      pause 1000
      low B.4
      pause 1000
      goto main
```



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

6. Connect an LED (and 330R resistor) to output pin 4. If connecting the LED directly to a PICAXE chip on a **proto** (or home-made) board, connect the LED **between the output pin and 0V**. When using the **project** boards (as supplied within the 14, 18 and 28 starter packs), connect the LED **between V+ and the output** connector, as the output is buffered by the darlington driver chip on the project board. (Make sure the LED is connected the correct way around!).
7. Make sure the PICAXE circuit is connected to the serial cable, and that the batteries are connected.
8. Select PICAXE>Run. A download bar should appear as the program downloads. When the download is complete the program will start running automatically – the LED should flash on and off every second.

Simulating a BASIC program



To simulate the program simply click the Simulate>Run menu. Each line of the BASIC code will be highlighted as it is processed, and an on-screen graphic shows the status of all input and output pins.

To change the status of an input simply click on the input button which is beside the corresponding leg of the chip on the graphic.

.

Tutorial 2 - Using Symbols, Comments & White-space

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 LED  = B.4           ; rename output4 'LED'
symbol buzzer = B.2         ; rename output2 'buzzer'

main:                       ; make a label called 'main'
      high LED              ; LED on
      low buzzer            ; buzzer off
      pause 1000            ; wait 1 second (1000 ms)
      low LED               ; LED off
      high buzzer           ; buzzer on
      wait 1                ; wait 1 second
      goto main             ; jump back to the start
```

Remember that **comments** (an explanation after the ; symbol) can make each line of a program much easier to understand. These comments are ignored by the computer when it downloads a program to the PICAXE

A label (e.g. **main:** in the program above) can be any word (apart from keywords such as 'switch'), 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.

This program uses the **wait** command. 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 (1000th of a second)).

Wait can be followed by a number between 1 and 65.
Pause can be followed by a number between 1 and 65535.

It is also a good programming technique to use tabs (or spaces) at the start of lines without labels so that all the commands are neatly aligned. The term **'white-space'** is used by programmers to define tabs, spaces and blank lines, and the correct use of white-space can make the program listing much easier to read and understand. See the example program on the next page, where code between the for...next commands is also indented with a tab for clarity.

Note:
Some early BASIC languages used **'line numbers'** rather than **labels** for 'goto' commands. Unfortunately this line number system can be inconvenient to use, because if you modify your program by later adding, or removing, lines of code you then have to modify all the line numbers within the 'goto' commands accordingly. The label system, as used in most modern BASIC languages, overcomes this problem automatically.

Tutorial 3 - For...Next Loops

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

This program flashes the LED connected to output pin 1 on and off 15 times. The number of times the code has been repeated is stored in the general purpose RAM memory of the PICAXE chip using variable b1 (the PICAXE contains 14 general purpose byte variables labelled b0 to b13). These variables can also be renamed using the **symbol** command to make them easier to remember.

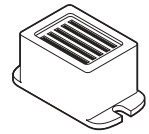
```
symbol counter = b1     ; define the variable b1 as "counter"
symbol LED = B.4        ; define pin 4 with the name "LED"

main:
      for counter = 1 to 15         ; start a for...next loop
            high LED                ; switch pin 4 high
            pause 500               ; wait for 0.5 second
            low LED                 ; switch pin 4 low
            pause 500               ; wait for 0.5 second
      next counter                  ; end of for...next loop

      end                           ' end program
```

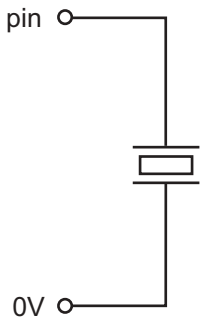
Note again how white-space (extra spaces) has been used to clearly show all the commands that are contained between the **for** and **next** commands.

Tutorial 4 - Making Sounds



Buzzers will make a fixed frequency noise when switched on. However the PICAXE system can automatically create noises of different frequencies by use of the sound, play and tune commands with a piezo sounder. All PICAXE chips support the sound command, which is designed to make warning 'beeps' etc. This is recommended instead of using buzzers.

To play musical tunes, the sound command is not recommended. Some PICAXE chips support the play and tune commands, specifically designed for playing musical tunes. Refer to the 'Tune' command in part 2 of the manual for more details.



Example sound program:

```
main:
      sound B.2,(50,100)      ; freq 50, length 100
      sound B.2,(100,100)     ; freq 100, length 100
      sound B.2,(120,100)     ; freq 120, length 100
      pause 1000              ; wait 1 second
      goto main               ; loop back to start
```

To test this program you must add a piezo sounder between the output pin (in this case output 2) and 0V. Note that on the project boards (supplied in the PICAXE-14, 18 and 28 starter packs) fitted with a Darlington driver the piezo must be connected directly to the PICAXE output pin (not the buffered output connection).

The first number provides the pin number (in this case output 2). The next number (in brackets) is the tone, followed by the duration. The higher the tone number the higher pitch the sound (note that only valid tones are 0 to 127).

The following program uses a for…next loop to produce 120 different sounds.

```
main:
      for b0 = 1 to 120       ; start a for...next loop
         sound B.2,(b0,50)    ; make a sound, freq from b0
      next b0                 ; next loop
      end
```

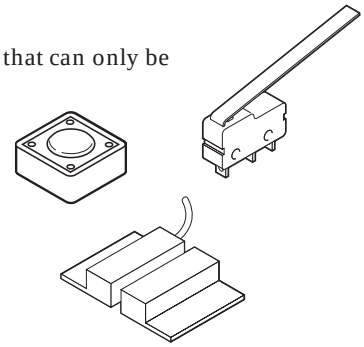
The number stored in variable b0 increase by 1 in every loop (1-2-3 etc.) Therefore by using the variable name b0 in the tone position, the note can be changed on each loop.

The following program does the same task but backwards, by using step value of -1 (rather than the default of +1 as above).

```
main:
      for b0 = 120 to 1 step -1     ; count down in loop
         sound B.2,(b0,50)          ; make a sound. freq from b0
      next b0                       ; next loop
      end
```

Tutorial 5 – Using Digital Inputs

A digital sensor is a simple 'switch' type sensor that can only be 'on' or 'off'.



Common examples of a digital sensor are:

- microswitches
- push and rocker switches
- reed switches

This program below shows how to react to switch pushes. In this program output pin 4 flashes every time the push switch on input pin 3 is pushed. Note that if using an 18 pin chip you should select a different input pin (e.g. pin0, as pin3 does not exist on this size chip).

```
main:                           ; make a label called 'main'
      if pinC.3 = 1 then flash  ; jump if the input is on
      goto main                 ; else loop back around

flash:                          ; make a label called 'flash'
      high B.4                  ; switch output 4 on
      pause 2000                ; wait 2 seconds
      low B.4                   ; switch output 4 off
      goto main                 ; jump back to start
```

In this program the first three lines make up a continuous loop. If the input is off (=0) the program just loops around time and time again. If the switch is on (=1) the program jumps to the label called **'flash'**. The program then flashes output 4 on for two seconds before returning to the main loop.

Note carefully the spelling in the **if…then** line – **pin3** is all one word (without a space). This is because pin3 is the name of a variable that contains the data from the input pin. Note also that only the label is placed after the command **then.**

Two switches (or more) can be combined by the AND or OR key words.

A 2-input AND gate is programmed as

```
if pinC.2 = 1 and pinC.3 = 1 then flash
```

A 3-input OR gate is programmed as

```
if pinC.1 = 1 or pinC.2 = 1 or pinC.3 = 1 then flash
```

To read the whole input port use the following command

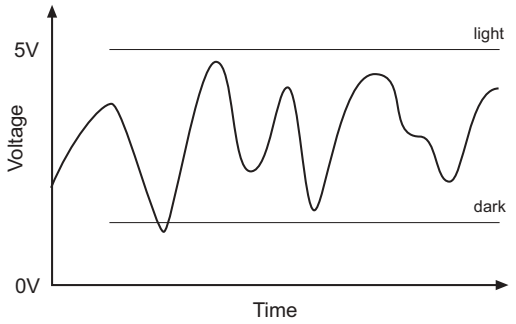
```
let b1 = pinsC
```

To isolate individual pins (e.g. 6 and 7) within the port, mask the pins variable with an & (logical AND) statement

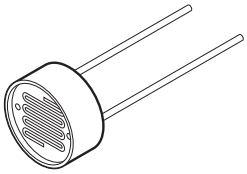
```
let b1 = pinsC & %11000000
```

Tutorial 6 – Using Analogue Sensors

An analogue sensor measures a continuous signal such as light, temperature or position. 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. dark = 0, light = 255).



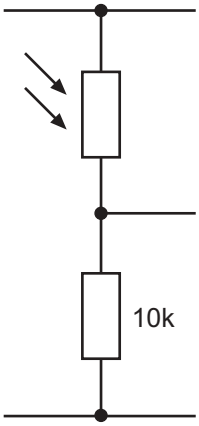
Common examples of analogue sensors are:



- LDR (Light Dependent Resistor)
- Thermistor
- Variable Resistor (potentiometer)

Using a Light Dependent Resistor (LDR)

The LDR is an example of an analogue sensor. It is connected to the PICAXE ADC input in a potential divider arrangement (e.g. input 1). Note that not all inputs have ADC capabilities - see the pinout diagrams for more information.



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 120 and a different LED if the value is less than 70. If the value is between 70 and 120 both LEDS are switched off.

```
main:                           ; make a label called ,main
      readadc C.1,b0            ; read ADC1 into variable b0
      if b0 > 120 then top      ; if b0 > 120 then do top
      if b0 < 70 then bot       ; if b0 < 70 then do bot
      low B.0                   ; else switch off 0
      low B.4                   ; and switch off 4
      goto main                 ; jump back to the start

top:                            ; make a label
      high B.0                  ; switch on 0
      low B.4                   ; switch off 4
      goto main                 ; jump back to start

bot:                            ; make a label
      high B.4                  ; switch on 4
      low B.0                   ; switch off 0
      goto main                 ; jump back to start
```

Tutorial 7 - Using Debug

When using analogue sensors it is often necessary to calculate the 'threshold' value necessary for the program (ie the values 70 and 120 in the tutorial 6 program). 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 C.1,b0 ; read channel 1 into variable b0
      debug b0       ; transmit value to computer screen
      pause 500      ; 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 Light falling on the LDR sensor is altered, the variable value will show the current sensor reading.

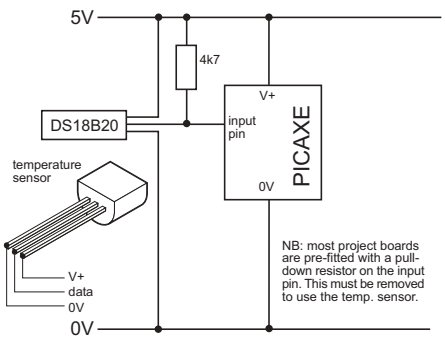
The debug window opens automatically after a download, but can also be opened manually at any time via the PICAXE>Debug menu.

Tutorial 8 - Using Serial Terminal with Sertxd

All PICAXE variants support the debug command. However the M and X parts also support more complex serial debug messages by use of the sertxd command, which sends a user defined serial string to the computer (at baud rate 4800). This can be displayed by the included Serial Terminal function (PICAXE>Terminal menu). The Serial Terminal can also be automatically opened every time a download takes place by the View>Options>Options menu.

```
main:                ; make a label called main
      readtemp C.1,b0 ; read input 1 into variable b0
      sertxd ("The value is ",#b0,cr,lf)
      pause 500      ; short delay
      goto main      ; jump back to the start
```

The sertxd command transmits the string "The value is" followed by the ASCII string of the current value of the variable b1 (the # prefix to the variable indicates an ASCII string representing the correct value is to be transmitted). The CR and LF constants are pre-defined values (13 and 10) that cause the serial terminal to display a newline for each value so that the display updates correctly.



This program uses the readtemp command to read the temperature from a DS18B20 digital temperature sensor connected to input 1.

Tutorial 9 - Number Systems

A microcontroller operates by performing a large number of commands in a very short space of time by processing electronic signals. These signals are coded in the binary system – the signal either being **high** (1) and **low** (0)

The counting system used in everyday activities is the **decimal system**. This number system uses the ten familiar digits 0 to 9 to explain how big or small the number is.

However when working with microcontrollers it is sometimes easier to work in binary. This is especially true when trying to control multiple outputs at the same time.

A single binary digit is referred to a **bit** (binary digit). The PICAXE systems use 8 bits (1 **byte**), with the least significant bit (LSB), bit 0, on the right hand side and the most significant bit (MSB), bit 7, on the left hand side.

Therefore the binary number %11001000 means set bits 7,6,3 high (1) and the others low (0). The % sign tells the computer you are working in binary instead of decimal.

This means that all 8 outputs can be controlled at the same time, instead of multiple high and low commands. The following program demonstrates how to make the seven segment display on the AXE050 tutorial board count from 0 to 9.

```
main:
      let pinsB = %00111111   ; digit 0
      pause 250               ; wait 0.25 second
      let pinsB = %00000110   ; digit 1
      pause 250               ; wait 0.25 second
      let pinsB = %01011011   ; digit 2
      pause 250               ; wait 0.25 second
      let pinsB = %01001111   ; digit 3
      pause 250               ; wait 0.25 second
      let pinsB = %01100110   ; digit 4
      pause 250               ; wait 0.25 second
      let pinsB = %01101101   ; digit 5
      pause 250               ; wait 0.25 second
      let pinsB = %01111101   ; digit 6
      pause 250               ; wait 0.25 second
      let pinsB = %00000111   ; digit 7
      pause 250               ; wait 0.25 second
      let pinsB = %01111111   ; digit 8
      pause 250               ; wait 0.25 second
      let pinsB = %01101111   ; digit 9
      pause 250               ; wait 0.25 second
      goto main
```

Each 'let pins=' line changes the number of bars that are lit on the seven segment display on the tutorial board. This is quicker, and more memory efficient, than using lots of 'high' and 'low' commands.

Tutorial 10 - 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 X part PICAXE microcontrollers support 255 sub-procedures. All other parts support 15 sub-procedures.

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

```
symbol LED = B.4              ; rename output4 'LED'
symbol buzzer = B.2           ; rename output2 'buzzer'
symbol counter = b1           ; define a counter using variable b1

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

      end                     ; end  of the main program


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

noise:
      high buzzer             ; buzzer on
      pause 2000              ; wait 2 seconds
      low buzzer              ; buzzer off
      return                  ; return from the sub-procedure
```

This second program shows how a variable can be used to transfer information into a sub-procedure. In this case variable b2 is used to tell the microcontroller to flash 5, and then 15, times.

```
symbol LED = B.4              ; rename output4 'LED'
symbol counter = b1           ; define a counter using variable b1

main:                         ; make a label called 'main'
      let b2 = 5              ; preload b2 with 5
      gosub flash             ; call the sub-procedure flash
      pause 500               ; wait a while
      let b2 = 15             ; preload b2 with 15
      gosub flash             ; call the sub-procedure flash
      pause 500               ; wait a while
      goto main               ; loop back

      end                     ; end  of the main program


flash:                        ; make a sub-procedure called flash
      for counter = 1 to b2   ; start a for…next loop
         high LED             ; LED on
         pause 250            ; wait 0.25 second
         low LED              ; LED off
         pause 250            ; wait 0.25 second
      next counter            ; next loop
      return                  ; return from the sub-procedure
```

Tutorial 11 - Using Interrupts

An interrupt is a special case of a sub-procedure. The sub-procedure immediately occurs when a particular input (or combination of inputs) is activated.

A polled interrupt is a quicker way of reacting to a particular input combination. It is the only type of interrupt available in the PICAXE system. The inputs port is checked between execution of each command line in the program, between each note of a tune command, and continuously during any pause command. If the particular inputs condition is true, a 'gosub' to the interrupt sub-procedure is executed immediately. When the sub-procedure has been carried out, program execution continues from the main program.

The interrupt inputs condition is any pattern of '0's and '1's on the input port, masked by the byte 'mask'. Therefore any bits masked by a '0' in byte mask will be ignored.

e.g.
to interrupt on input1 high only

```
setint %00000010,%00000010
```

to interrupt on input1 low only

```
setint %00000000,%00000010
```

to interrupt on input0 high, input1 high and input 2 low

```
setint %00000011,%00000111
```

etc.

Only one input pattern is allowed at any time. To disable the interrupt execute a SETINT command with the value 0 as the mask byte.

Notes:

1) Every program which uses the SETINT command must have a corresponding interrupt: sub-procedure (terminated with a return command) at the bottom of the program.
2) When the interrupt occurs, the interrupt is permanently disabled. Therefore to re-enable the interrupt (if desired) a SETINT command must be used within the interrupt: sub-procedure itself. The interrupt will not be enabled until the 'return' command is executed.
3) If the interrupt is re-enabled and the interrupt condition is not cleared within the sub-procedure, a second interrupt may occur immediately upon the return command.
4) After the interrupt code has executed, program execution continues at the next program line in the main program. In the case of the interrupted pause, wait, play or tune command, any remaining time delay is ignored and the program continues with the next program line.

More detailed SETINT explanation.
The SETINT must be followed by two numbers - a 'compare with value' (input) and an 'input mask' (mask) in that order. It is normal to display these numbers in binary format, as it makes it more clear which pins are 'active'. In binary format input7 is on the left and input0 is on the right.

The second number, the 'input mask', defines which pins are to be checked to see if an interrupt is to be generated ...
- %00000001 will check input pin 0
- %00000010 will check input pin 1
- %01000000 will check input pin 6
- %10000000 will check input pin 7
- etc

These can also be combined to check a number of input pins at the same time...
- %00000011 will check input pins 1 and 0
- %10000100 will check input pins 7 and 2

Having decided which pins you want to use for the interrupt, the first number (inputs value) states whether you want the interrupt to occur when those particular inputs are on (1) or off (0).

Once a SETINT is active, the PICAXE monitors the pins you have specified in 'input mask' where a '1' is present, ignoring the other pins.

An input mask of %10000100 will check pins 7 and 2 and create a value of %a0000b00 where bit 'a' will be 1 if pin 7 is high and 0 if low, and bit 'b' will be 1 if pin 2 is high and 0 if low.

The 'compare with value', the first argument of the SETINT command, is what this created value is compared with, and if the two match, then the interrupt will occur, if they don't match then the interrupt won't occur.

If the 'input mask' is %10000100, pins 7 and 2, then the valid 'compare with value' can be one of the following ...

- %00000000 Pin 7 = 0 and pin 2 = 0
- %00000100 Pin 7 = 0 and pin 2 = 1
- %10000000 Pin 7 = 1 and pin 2 = 0
- %10000100 Pin 7 = 1 and pin 2 = 1

So, if you want to generate an interrupt whenever Pin 7 is high and Pin 2 is low, the 'input mask' is %10000100 and the 'compare with value' is %10000000, giving a SETINT command of ...

- SETINT %10000000,%10000100

The interrupt will then occur when, and only when, pin 7 is high and pin 2 is low. If pin 7 is low or pin 2 is high the interrupt will not happen as two pins are 'looked at' in the mask.

Example:

```
setint %10000000,%10000000
‘ activate interrupt when pinC.7 only goes high

main:
      low B.1                       ; switch output 1 off
      pause 2000                    ; wait 2 seconds
      goto main                     ; loop back to start

interrupt:
      high B.1                      ; switch output 1 on
      if pinC.7 = 1 then interrupt  ; loop here until the
                                    ; interrupt cleared
      pause 2000                    ; wait 2 seconds
      setint %10000000,%10000000    ; re-activate interrupt
      return                        ; return from sub
```

In this example an LED on output 1 will light immediately the input is switched high. With a standard if pin7 =1 then.... type statement the program could take up to two seconds to light the LED as the if statement is not processed during the pause 2000 delay time in the main program loop (standard program shown below for comparison).

```
main:
      low B.1                ; switch output 1 off
      pause 2000             ; wait 2 seconds
      if pinC.7 = 1 then sw_on
      goto main              ; loop back to start

sw_on:
      high B.1               ; switch output 1 on
      if pinC.7 = 1 then sw_on
                  ; loop here until the condition is cleared
      pause 2000             ; wait 2 seconds
      goto main              ; back to main loop
```

The next step - your own PICAXE project!

You should now have a good idea about how the PICAXE system works and should be able to start designing your own project.

Make sure you also study sections 2 (BASIC Commands) and 3 (Microcontroller Interfacing Circuits) of the manual for additional information.

There are a large range of project ideas and examples within the Help files of the Programming Editor software. Studying these exemplar projects will provide further ideas, as will looking at the very active forum within the technical support section of the main PICAXE website (www.picaxe.forumco.uk). The forum has a very large community of helpful PICAXE enthusiasts who can always lend a hand if you are struggling with a project!

There is no limit to how creative PICAXE users can be! Have a go at your own project, you might be surprised how quickly you can start developing exciting microcontroller based electronic projects!

Appendix A – BASIC Commands Summary

This appendix provides an overview of available commands. Refer to section 2 of the manual for more specific information and examples for each BASIC Command

PICAXE-08 / 08M/14M/20M Commands

Output	high, low, toggle, pulsout, let pins =
ADC	readadc
I/O Config.	input, output, reverse, let dirs =
PWM	pwm
Sound	sound
Input	if...then, readadc, pulsin, button
Serial	serin, serout
Program Flow	goto, gosub, return, branch
Loops	for...next
Mathematics	let (+, -, *, **, /, //, max, min, &, |, ^, &/, |/, ^/)
Variables	if...then, random, lookdown, lookup
Data memory	eeprom, write, read
Delays	pause, wait, nap, sleep, end
Miscellaneous	symbol, debug

PICAXE-08M/14M/18M Additional Commands:

Input	count
ADC	readadc10
Interrupt	setint
PWM	pwmout
Music	play, tune
RAM	peek, poke
Servo Control	servo
Infrared	infrain2, infraout
Temperature	readtemp, readtemp12
1-wire Serial No	readowsn
Clock Frequency	setfreq

PICAXE-18 / 18A / 18M / 18X Commands

Output	high, low, toggle, pulsout, let pins =
Input	if...then, readadc, pulsin, button
ADC	readadc
RAM	peek, poke
Sound	sound
Serial	serin, serout
Program Flow	goto, gosub, return, branch
Loops	for...next
Mathematics	let (+, -, *, **, /, //, max, min, &, |, ^, &/, |/, ^/)
Variables	if...then, random, lookdown, lookup
Data memory	eeprom, write, read
Delays	pause, wait, nap, sleep, end
Miscellaneous	symbol, debug

PICAXE-18A / 18M / 18X Additional Commands:

Interrupt	setint
Servo Control	servo
Infrared	infrain
Temperature	readtemp
1-wire Serial No	readowsn
1-wire Clock	readowclk, resetowclk (18A only)
Keyboard	keyin, keyled
Clock Frequency	setfreq

PICAXE-18M Additional Commands:

Input	count
ADC	readadc10
Music	play / tune
Temperature	readtemp12
PWM	pwmout
Infrared	infrain2 / infraout

PICAXE-18X Additional Commands:

Input	count
ADC	readadc10
I2C	readi2c, writei2c, i2cslave
Temperature	readtemp12
PWM	pwmout

PICAXE-28A / 28X / 28X1 / 28X2 Commands

Output	high, low, toggle, pulsout, let pins =
Input	if...then, pulsin, button
ADC	readadc
RAM	peek, poke
Sound	sound
Serial	serin, serout
Program Flow	goto, gosub, return, branch
Loops	for...next
Mathematics	let (+, -, *, **, /, //, max, min, &, |, ^, &/, |/, ^/)
Variables	if...then, random, lookdown, lookup
Data memory	eeprom, write, read
Delays	pause, wait, nap, sleep, end
Miscellaneous	symbol, debug
Interrupt	setint
Servo Control	servo
Infrared	infrain
Temperature	readtemp
1-wire Serial No	readowsn
Keyboard	keyin, keyled

PICAXE-28X Additional Commands:

Input	count, if portA….then
ADC	readadc10
Portc config.	let dirsc =
Portc output	high portC, low portC, let pinsc =
I2C	readi2c, writei2c, i2cslave
Temperature	readtemp12
PWM	pwmout

PICAXE-28X1 Additional Commands:

Scratchpad	put, get, @ptr, @ptrinc, @ptrdec
ADC	calibadc, calibadc10
Serial	hsersetup, hserout, hserin, serrxd
SPI	spiin, spiout, hspisetup, hspiin, hpsiout
I2C	hi2csetup, hi2cin, hi2cout
One-wire	owin, owout
PWM	hpwm
Timer	settimer
Power control	hibernate, enablebod, disablebod

PICAXE-40X/40X1/40X2 Commands

Output	high, low, toggle, pulsout, let pins = ,
Input	if...then, if portA…then, if portC then…, pulsin, button,
Counting	count
ADC	readadc, readadc10
Portc config.	let dirsc =
Portc output	high portc, low portc, let pinsc =
PWM	pwmout
RAM	peek, poke
Sound	sound
Serial	serin, serout
Program Flow	goto, gosub, return, branch
Loops	for...next
Mathematics	let (+, -, *, **, /, //, max, min, &, |, ^, &/, |/, ^/)
Variables	if...then, random, lookdown, lookup
Data memory	eeprom, write, read
Delays	pause, wait, nap, sleep, end
Miscellaneous	symbol, debug
Interrupt	setint
Servo Control	servo
Infrared	infrain
Temperature	readtemp, readtemp12
1-wire Serial No	readowsn
Keyboard	keyin, keyled

PICAXE-40X1 Additional Commands:

Scratchpad	put, get, @ptr, @ptrinc, @ptrdec
ADC	calibadc, calibadc10
Serial	hsersetup, hserout, hserin, serrxd
SPI	spiin, spiout, hspisetup, hspiin, hpsiout
I2C	hi2csetup, hi2cin, hi2cout
One-wire	owin, owout
PWM	hpwm
Timer	settimer
Power control	hibernate, enablebod, disablebod

Appendix B – Over-clocking at higher frequencies

All main PICAXE functions are based upon a 4MHz resonator frequency (8MHz on X2 parts). However the user may choose to 'overclock' some of the PICAXE parts to achieve faster operation

With the -08, -18 the internal resonator is fixed at 4MHz and cannot be altered.

With the -08M, -14M, -18A, -18M, -18X the internal resonator has a default value of 4MHz. However it can be increased by the user to 8MHz via use of the 'setfreq m8' command.

With the -28 and -28A an external 4MHz resonator must be used.

With the -28X / -40X an external 4MHz 3 pin ceramic resonator is normally used, but it is also possible to use a faster resonator (8 or 16Mhz), although this will affect the operation of some of the commands.

With the -28X1 / -40X1 the internal resonator has a default value of 4MHz. However it can be increased by the user to 8MHz via use of the 'setfreq m8' command or to an external 16/20MHz 3 pin ceramic resonator via use of the 'setfreq em16 (em20)' command.

The Programming Editor software supports resonator frequencies of 4, 8, 16MHz only. No other frequencies are recommended. If any other frequency is used it may not be possible to download a new program into the PICAXE microcontroller.

To change the frequency:

All 8, 14, 18 and 20 pin chips
Download a program containing the command `setfreq m4` (for 4 MHz) or `setfreq m8` (for 8Mhz). If no setfreq command is used in a program the frequency will default to 4MHz (8MHz on X2 parts). Note the new frequency occurs immediately after the command is run. When downloading new programs, you must ensure the correct frequency (View>Options>Mode) is used to match the last program running in the PICAXE chip. If in doubt perform a 'hard-reset' at 4Hz.

PICAXE-28X and PICAXE-40X
Solder the appropriate external 3 pin ceramic resonator into the project board.

PICAXE-28X1/28X2 and PICAXE-40X1/40X2
Solder the appropriate external 3 pin ceramic resonator into the project board. Use the setfreq command to switch between internal 4 or 8 or external frequency.

Downloading programs at 4, 8, 16MHz

After changing frequency you must select the correct frequency via the View>Options>Mode software menu. If the wrong frequency is selected the program will not download. This is not required on M2, X1 and X2 parts as they default back to the internal resonator for the download.

Commands affected by resonator frequency.

Many of the commands are affected by a change in resonator frequency. A summary of the commands affected are given below (see section 2 of the manual - BASIC Commands for detailed command syntax and information).

When using devices with an internal resonator, remember that it is sometimes possible to change back to 4MHz to run the command dependent on this speed e.g.

```
setfreq m4
readtemp 1,b1
setfreq m8
```

This is not possible with devices with an external resonator. This process is automatic on M2, X1 and X2 parts.

Commands for which operation is affected by change in resonator speed:

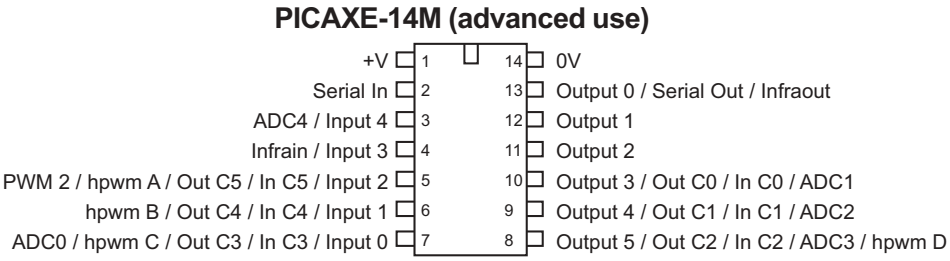
- count
- debug
- readi2c, writei2c, i2cin, i2cout
- pause, wait
- pulsin, pulsout
- pwm. pwmout
- serin, serout, sertxd, serrxd, hsersetup, hserin, hserout
- sound

Note that nap, doze and sleep are not affected by resonator speed as they use their own, separate, internal timer.

The following commands will not work at 8 or 16MHz due to timing issues with the external device listed. Note that M2, X1 and X2 parts automatically switch to internal 4MHz operation to process these commands, so the external frequency can be higher.

- irin, infrain, infrain2, irout, infraout (infrared receiver)
- kbin, keyin (keyboard)
- kbled, keyled (keyboard)
- readtemp / readtemp12 (DS18B20 temperature sensor)
- readowsn, owin, owout (1-wire device)
- servo (servo)
- play, tune (music)

Appendix C – Configuring the PICAXE-14M Input-Output Pins



The PICAXE-14M is a very versatile device. In it's default state, which is designed primarily for educational use, it has a simple, clean 'inputs on left' - 'outputs on right' layout.

However more advanced users can re-configure the bottom 3 pins on each side to be either inputs or outputs. This has added advantages as follows:

- more flexible quantity of inputs and outputs
- more ADC channels become available
- the option to use pwmout via the pwmout and hpwm commands

The diagram above shows the advanced function of each pin. The 6 pins are arranged in a 'port' (portC) with bits labelled C0-C5. Note that the portC bit numbers do not correspond to the normal input/output numbers (or even the leg numbers!). Study the pinout diagram very carefully!

Using portc pins as outputs

Any portc pin can be configured to be used as a digital output.

To convert the pin C3 to output and make it high

```
high portc 3
```

To convert the pin C3 to output and make it low

```
low portc 3
```

To convert all the pins to outputs

```
let dirsc = %00111111
```

To convert all the pins to inputs

```
let dirsc = %00000000
```

It is not possible to access the portc pins C3-C5 with any other 'output' type commands (serout, pulsout etc). Therefore when used as outputs these pins should be reserved as simple on/off outputs. Remember that C0-C2 are normal outputs (3-5) anyway, and so can be used with any output command.

Using portc as digital inputs

The portc pins C0, C1, C2 are, by default, configured as outputs. They can however be reconfigured as inputs, but you must ensure your hardware design allows for the fact that the pin will be an output upon powerup. A simple 1k resistor in series with the pin will normally resolve this issue.

To make the pin an input you must use 'let dirsc = ' as described above.

The following syntax is used to test the input condition:

```
if portC pin0 = 1 then jump
```

i.e. the additional keyword 'portC' is inserted after the 'if' command.

to test if two (or more) portc inputs are on

```
if portC pin0 = 1 AND pin1 = 1 then jump
```

to test if either of two (or more) portc inputs are on

```
if portC pin0 = 1 OR pin1 = 1 then jump
```

Note the portc command is only required once after the 'if' command.
It is not possible to test inputs on two different ports within the same if…then statement.

It is not possible to access the portc pins with any other 'input' type commands (count, pulsin etc). Therefore these pins should be reserved as simple on/off switches.

Note that 'dirsc' uses the common BASIC notation 0 for input and 1 for output.

Using portc as analogue inputs

Three additional ADC pins, ADC1,2,3, are available AFTER the corresponding pin has been converted to an input. You must ensure your hardware design allows for the fact that the pin will be an output upon powerup. A simple 1k resistor in series with the pin will normally resolve this issue.

Using portc as pwm outputs

C5 can be used with the pwmout command, but will make this pin an output. Pins C2-5 (hpwm A-D) can all be used with the hpwm command, but will also make the corresponding pins outputs.

Special Note - Output Pin 0

Pin 0 (leg 13) is used during the program download, but can also be used as a normal output once the download is complete. Therefore you must remember that your output device will rapidly switch on and off as the download takes place (not a problem with simple outputs like LEDs, but could cause problems with other devices such as motors).

Appendix D – Configuring PICAXE-08 / 08M Input-Output Pins

The PICAXE-08 microcontroller has 5 input/output pins. Unlike the larger PICAXE microcontroller (where the pins are pre-defined) the user can select whether some of the pins are used as input or as outputs.

Pin 0 must always be an output, and pin 3 must always be an input (this is due to the internal construction of the microcontroller). The other 3 pins can be selected to be inputs or outputs, and so the user can select any input/output combination between the limits of 1 input-4 outputs and 4 inputs-1 output.

In addition pin 1 also contains a low-resolution analogue to digital converter and so can be used as an analogue input pin if required.

Important - Don't Get Confused!
The input/output pin numbers are NOT the same as the external 'leg' numbers, as the input/output pin numbering follows the microcontrollers manufacturers port allocation. To avoid confusion this manual always talks about 'legs' where referring to the external physical location of the input/output pin.

Leg	Description	Notes
1	Positive Supply, V	Use a 3V to 5V battery pack/supply
2	Serial In	Used for the program download
3	Pin 4	Input or output
4	Pin 3	Input only
5	Pin 2	Input or output
6	Pin 1	Input or output
7	Pin 0 / Serial Out	Output only. Also used for download
8	Ground, G	Connect to the power supply (0V)

Special Note - Output Pin 0

Pin 0 (leg 7) is used during the program download, but can also be used as a normal output once the download is complete. On the project boards a jumper link allows the microcontroller leg to either be connected to the download socket (PROG position) or to the output (OUT position). Remember to move the jumper into the correct position when testing your program!

If you are making your own pcb you can include a similar jumper link or small switch, or you may prefer to connect the microcontroller leg to both the output device and the program socket at the same time. In this case you must remember that your output device will rapidly switch on and off as the download takes place (not a problem with simple outputs like LEDs, but could cause problems with other devices such as motors).

Selecting Inputs or Outputs.

When the PICAXE-08 first powers up, all pins are configured as input pins (except pin0, which is always an output). There are three methods of setting the other pins to be outputs (if required)

Method 1 – use a command that requires the pin to be an output.

This is the simplest method, used by most educational users. As soon as a command that involves an output pin (such as high, low, toggle, serout or sound) is used, the PICAXE-08 microcontroller automatically converts the pin to an output (and leaves the pin as an output).

Therefore the simplest way to setup outputs is just to put a 'low' command at the start of the program for each output pin. This tells the microcontroller to make the pin an output, and to make sure the output is condition low (off).

Method 2 – use the input and output command.

The command 'output ?' (where ? is the pin number) can also be used to tell the pin to be an output at the start of a program. Likewise the 'input ?' command can be used to set the pin as an input, although this is not normally necessary as most of the pins are set as inputs by default. Note that the output command does not set the pin into a known high or low state, so it is often preferable to use the 'low' command instead.

The input and output commands have no effect on pin 0 (output) and pin 3 (input), which cannot be altered.

Method 3 – (advanced) use the let dirs = command

The 'let dirs = %000100111' command can be used to simultaneously set all the pins at the same time. This is quicker than using multiple input/output commands but requires an understanding of binary bits (explained in tutorial 9). Placing a 0 for the pin number bit will make the corresponding pin an input, a 1 will make the pin an output. The value of bits 0,3,5,6,7 can be either 0s or 1s as they have no effect on the microcontroller and are simply ignored.

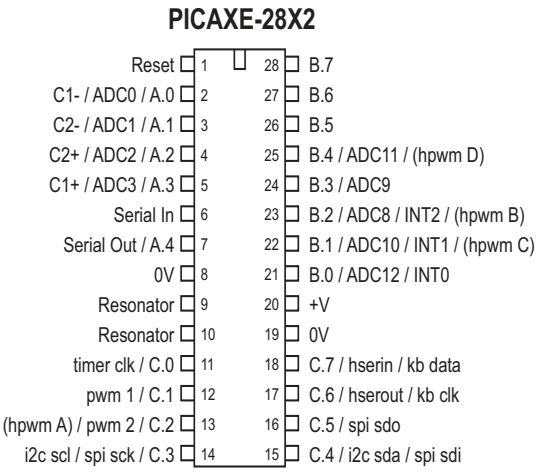
Selecting pins to be an analogue input.

Use of the readadc command will automatically configure the pin to be an analogue input. Therefore use the command 'readadc 1,b2' whenever you wish to take an analogue reading (presuming use of variable b2 to store the analogue reading).

Appendix E – Configuring PICAXE-28X / 28X1 Input-Output Pins

To provide greater flexibility, the input / output pin configuration of the PICAXE-28X can be varied by the user.
The default power up settings are the same as the other PICAXE-28 parts (8 in, 8 out, 4 analogue).

PORTA (legs 2 to 5) provide 4 analogue inputs (default) or up to 4 digital inputs.
PORTB (leg 21 to 28) provide 8 fixed outputs.
PORTC (leg 11 to 18) provide 8 digital inputs (default) or up to 8 outputs.



This gives a maximum of 12 digital inputs, 16 outputs and 4 analogue inputs

PORTA Functions

Leg	Default Function	Second Function
2	analogue 0	porta input 0
3	analogue 1	porta input 1
4	analogue 2	porta input 2
5	analogue	porta input 3

PORTB Functions

PORTB pins are fixed as outputs and cannot be altered.

PORTC Functions

Leg	Default	Second Function	Special Function
11	input 0	output portc 0	infrared (input)
12	input 1	output portc 1	pwm 1 (output)
13	input 2	output portc 2	pwm 2 (output)
14	input 3	output portc 3	i2c scl clock (input)
15	input 4	output portc 4	i2c sda data (input)
16	input 5	output portc 5	
17	input 6	output portc 6	keyboard clock (input)
18	input 7	output portc 7	keyboard data (input)

The portC pins can be used as the default inputs, changed to outputs, or used with their special function via use of the infrain, keyin, i2cslave, or pwmout command as appropriate.

Using porta as digital inputs

The porta pins 0 to 3 (legs 2 to 5) are, by default, configured as analogue inputs. However they can also be used as simple digital inputs.

The following syntax is used to test the input condition:

```
if portA pin0 = 1 then jump
```

i.e. the additional keyword 'portA' is inserted after the 'if' command.

to test if two (or more) porta inputs are on

```
if portA pin0 = 1 AND pin1 = 1 then jump
```

to test if either of two (or more) porta inputs are on

```
if portA pin0 = 1 OR pin1 = 1 then jump
```

Note the portA command is only required once after the 'if' command.
It is not possible to test inputs on two different ports within the same if…then statement.

It is not possible to access the portA pins with any other 'input' type commands (count, pulsin etc). Therefore these pins should be reserved as simple on/off switches.

Using portc as outputs

The portc pins are, by default, digital input pins.
However they can also be configured to be used as digital outputs.

To convert the pin to output and make it high

```
high portc 1
```

To convert the pin to output and make it low

```
low portc 1
```

To convert all the pins to outputs

```
let dirsc = %11111111
```

To convert all the pins to inputs

```
let dirsc = %00000000
```

Note that 'dirsc' uses the common BASIC notation 0 for input and 1 for output.

(Advanced - If you are more familiar with assembler code programming you may prefer to use the command 'let trisc =' instead, as this uses the inverted assembler notation - 1 for input and 0 for output. Do not attempt to directly poke the trisc register (poke command) as the PICAXE bootstrap refreshes the register setting regularly).

To switch all the outputs on portc high

```
let pinsc = %11111111
(or)  let portc = %11111111
```

To switch all the outputs on portc low

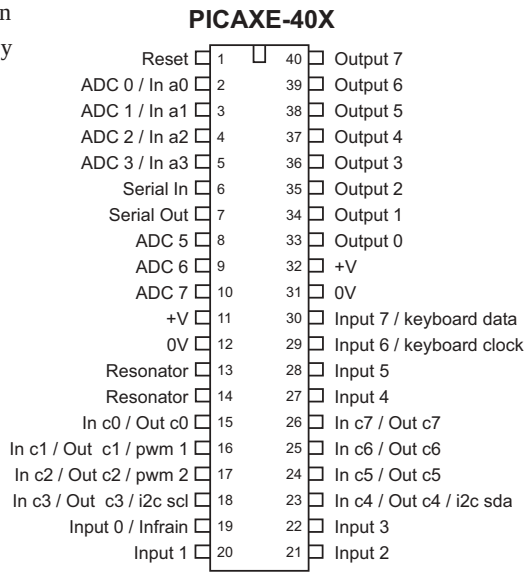
```
let pinsc = %00000000
(or)  let portc = %00000000
```

Appendix F – Configuring PICAXE-40X / 40X1 Input-Output Pins

To provide greater flexibility, the input/output pin configuration of the PICAXE-40X can be varied by the user.

PORTA (legs 2 to 5) provide 4 analogue inputs (default) or up to 4 digital inputs.
PORTB (leg 33 to 40) provide 8 fixed outputs.
PORTC (leg 15-18. 23-26) provide 8 digital inputs (default) or up to 8 outputs.
PORTD (leg 19-22, 27-30) provide 8 digital inputs
PORTE (leg 8 to 10) provide 3 analogue inputs

This gives a maximum of 20 digital inputs, 16 outputs, 7 analogue inputs



PORTA Functions

Leg	Default Function	Second Function
2	analogue 0	porta input 0
3	analogue 1	porta input 1
4	analogue 2	porta input 2
5	analogue	porta input 3

PORTB / PORTE Functions

PORTB pins are fixed as outputs and cannot be altered.
PORTE pins are fixed as analogue inputs and cannot be altered.

PORTC Functions

Leg	Default	Second Function	Special Function
15	input portc 0	output portc 0	
16	input portc 1	output portc 1	pwm 1 (output)
17	input portc 2	output portc 2	pwm 2 (output)
18	input portc 3	output portc 3	i2c scl clock (input)
23	input portc 4	output portc 4	i2c sda data (input)
24	input portc 5	output portc 5	
25	input portc 6	output portc 6	
26	input portc 7	output portc 7	

The portC pins can be used as the default inputs, changed to outputs, or used with their special function via use of the i2cslave or pwmout command

PORTD Functions

Leg	Default Function	Special Function
19	input 0	infrared (input)
20	input 1	
21	input 2	
22	input 3	
27	input 4	
28	input 5	
29	input 6	keyboard clock (input)
30	input 7	keyboard data (input)

Using porta as digital inputs

The porta pins 0 to 3 (legs 2 to 5) are, by default, configured as analogue inputs. However they can also be used as simple digital inputs.
The following syntax is used to test the input condition:

```
if porta pin0 = 1 then jump
```

i.e. the additional keyword 'portA' is inserted after the 'if' command.

to test if two (or more) porta inputs are on

```
if porta pin0 = 1 AND pin1 = 1 then jump
```

to test if either of two (or more) porta inputs are on

```
if porta pin0 = 1 OR pin1 = 1 then jump
```

Note the portA command is only required once after the 'if' command.
It is not possible to test inputs on two different ports within the same if…then statement.

It is not possible to access the portA pins with any other 'input' type commands (count, pulsin etc). Therefore these pins should be reserved as simple on/off switches.

Using portc as digital inputs

On the PICAXE-40X **portD** are the standard inputs, and hence use the standard `if pin0 =` command. Therefore for **portC** inputs the extra keyword `portC` must be used (as in the `if portA pin0 =` example above).

Using portc as outputs

The portc pins are, by default, digital input pins.
However they can also be configured to be used as digital outputs.
To convert the pin to output and make it high

```
high portc 1
```

To convert the pin to output and make it low

```
low portc 1
```

To convert all the pins to outputs

```
let dirsc = %11111111
```

To convert all the pins to inputs

```
let dirsc = %00000000
```

Note that 'dirsc' uses the common BASIC notation 0 for input and 1 for output.

(Advanced - If you are more familiar with assembler code programming you may prefer to use the command 'let trisc =' instead, as this uses the inverted assembler notation - 1 for input and 0 for output. Do not attempt to directly poke the trisc register (poke command) as the PICAXE bootstrap refreshes the register setting regularly).

To switch all the outputs on portc high

```
let pinsc = %11111111
(or)  let portc = %11111111
```

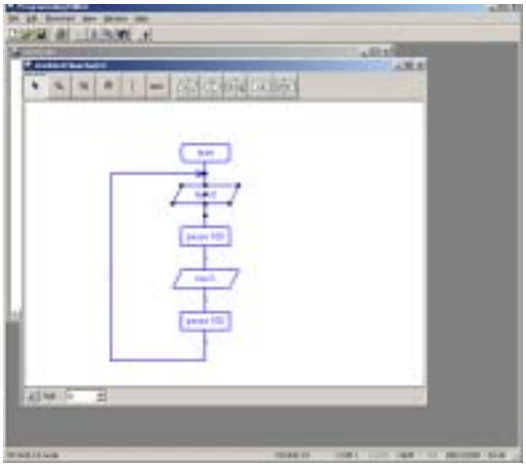
To switch all the outputs on portc low

```
let pinsc = %00000000
(or)  let portc = %00000000
```

Appendix G - Using Flowcharts in Programming Editor

This flowcharting method is no longer supported or developed, please consider using the much more powerful and user friendly Logicator flowcharting software instead. Logicator can be downloaded from the software pages at www.picaxe.co.uk

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. Click File>New Flowchart to start a new flowchart.



Flowchart Tools

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.

Select Area
Use to select a particular area.

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 shape.

Connection Points
Use instead of a long complicated line. This gives two 'connection points' that can be placed at different points within the flowchart.

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.

After dropping a shape the cursor will remain in that shape mode until either:
1) the select tool is selected
2) the right hand mouse button is clicked to move back to select mode

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.

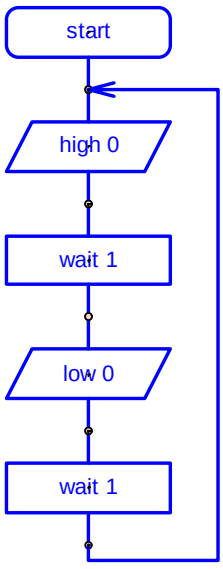
For further information about each command see section 2 of the manual - 'BASIC Commands'. Any BASIC Command not supported by the simulation can be added via the 'generic' shape, indicated by three dots.

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.



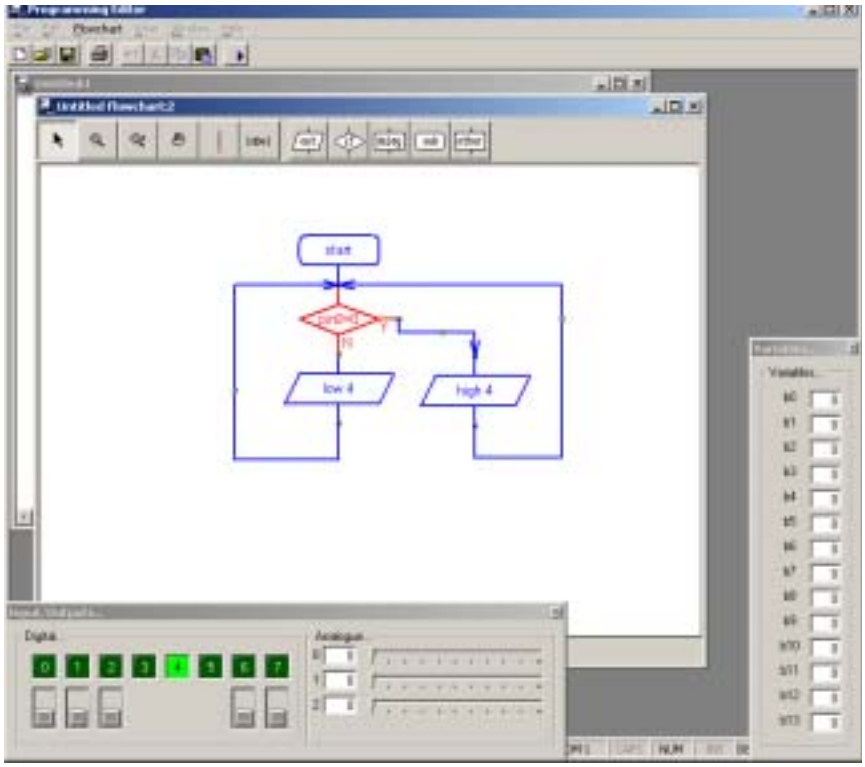
On Screen Simulation

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 can be directly downloaded to the microcontroller. Alternately the flowchart can be 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.

Zooming

To zoom in and out you can use the two toolbar zoom features, or the rapid zoom button at the bottom right hand corner of the screen. Left click this button to zoom-in and right click to zoom-out.

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.

Appendix H - Frequently Asked Questions (FAQ).

Where can I purchase PICAXE microcontrollers?

All microcontrollers can be purchased from within the PICAXE section of the online store at www.tech-supplies.co.uk or from our distributors (see www.picaxe.co.uk)

Which cable - serial or USB?

Many modern computers do not have a 9 pin legacy serial port and so we always recommend the USB download cable part AXE027. However the AXE026 serial cable is a more economical option for multiple computers that still have serial ports - e.g. in a school IT room.

There appears to be two PICAXE serial download cables - which should I use?

The standard serial PICAXE cable (part AXE026) ends with a stereo style 3.5mm plug. If making your own board we recommend this stereo cable cheaper as it is cheaper, better quality, and our sample PCB files use this connector (part CON039). The original PICAXE-28 cable (part AXE025) ended with a 3 pin in-line connector, but this cable is no longer used on any of our project boards or sample pcbs.

I've built a second pcb (without the download circuit) and the PICAXE program will not run!

If you program a PICAXE chip in a different board, and then move the chip to a board without the download circuit, you must ensure that the 'serial in' pin is tied to ground (0V) on the second board for reliable operation.

I've bought some blank PICs and they don't work in the PICAXE system!

The PICAXE microcontroller is not a blank PICmicro! It is a microcontroller that has been pre-programmed with a 'bootstrap' program that enables the download via the direct cable link (the bootstrap program tells the microcontroller how to interpret the direct cable programming commands). Therefore you must buy 'PICAXE' microcontrollers, rather than blank microcontrollers, to use with the PICAXE system. However we sell PICAXE microcontrollers at approx. the same price as blank devices, so there is very little price difference for the end user, particularly if you purchase the multi-packs.

I've programmed a PICAXE microcontroller using a conventional programmer and it will now not work in the PICAXE system!

You have overwritten, and hence deleted, the PICAXE bootstrap program (see above). The microcontroller can no longer be used as a PICAXE microcontroller, but you can naturally continue using it with your conventional programmer.

Can you reprogram microcontrollers (that I have accidentally erased) with the bootstrap program?

No. We do not accept microcontrollers from unknown sources due to the correct storage/handling procedures required by these devices. We use gang programmers costing several thousand pounds to program the bootstrap code into the blank microcontrollers, and so must protect this expensive equipment from damage. It is also likely that if we did offer this service the handling cost would end up more expensive than new PICAXE microcontrollers anyway!

Can you supply the bootstrap program so that I can make my own PICAXE?

No. The small royalty made on each PICAXE chip sold is the only financial benefit to our company to support the PICAXE system - the software is free and the cables/development kits are sold at very low cost. Therefore we do not allow anyone else to manufacture PICAXE microcontrollers.

Can I see the assembler code that is downloaded into the PICAXE?

If you own a Revolution Serial PIC Programmer, you can convert PICAXE BASIC programs into assembler code, to program blank PICs or to just learn how assembler code works by 'disassembly'. However some of the more complex commands (e.g. serin) are not supported, and the assembler code program generated is optimised for sequential learning (not optimised for compactness as with the PICAXE system) and so the code is not 100% identical to that downloaded to the PICAXE.

Can you alter the input/output pin arrangement of the PICAXE microcontroller?

The PICAXE-08/08M, all M2 and all X2 parts have configurable pins. The other parts have mainly fixed i/o, although some pins can be changed - see the appendices at the end of Manual part 1 for more details.

How long a program can I download into the PICAXE microcontroller?

This varies on the commands used, as not all commands use the same amount of memory. As a general rule you can download about

40-110 lines of code into PICAXE-08/18
80-220 lines of code into PICAXE-08M/14M/20M/18A/18M/28/28A
600-1800 lines of code into PICAXE-14M2/18M2/20M2/18X/28X/40X
2000-3200 lines of code into PICAXE-20X2/28X1/28X2/40X1/40X2

However some commands, such as sound and serout use more memory and so will reduce this count. In our experience most educational programs that are too long to download are generally badly composed, and can be greatly reduced in size by use of sub-procedures etc.

Do I need to erase the device?

How do I stop a program in the PICAXE microcontroller running?

Each download automatically overwrites the whole of the previous program. There is generally no need to erase the memory at any point. However if you want to stop a program running you can select the 'Clear Hardware Memory' menu to download an 'empty' program into the PICAXE memory.

How often can the PICAXE microcontroller be reprogrammed?

PICAXE chips can be reprogrammed at least 100,000 times. Note these are minimum values and the actual values may be much greater.

How vulnerable to damage are the microcontrollers?

The microcontrollers have a high level of static protection built into each pin and so handling them without any personal static protection in an educational environment is perfectly acceptable.

Can I control servos using the PICAXE?

Yes, many parts have a 'servo' command that allows control of up to 8 servos (one on each output).

Can I control an LCD display?

Yes, the PICAXE supports serial LCD modules (like the Serial LCD/Clock Module AXE033) via the serout command. Note that the AXE033 module can also be pre-programmed with up to 8 messages to reduce the memory usage of the PICAXE microcontroller.

How fast does the PICAXE operate?

The PICAXE-08/18 microcontrollers have an internal 4MHz resonator, and the PICAXE-28/40 uses an external 4MHz ceramic resonator. This means the microcontroller processes 1 million assembler commands a second, which equates to roughly about 1,000 BASIC commands per second.
The M and X parts can be overclocked to 8 or 16MHz (multiplies speed by x2 or x4).

Does the PICAXE support interrupts?

Yes. Many parts support a polled interrupt on the input port. Use the 'setint' or 'setintflags' command to setup the desired interrupt port setting.

How do I create time delays longer than 65 seconds?

The best way of creating long delays is to do minute delays with a loop, e.g. to wait an hour (60 minutes)

```
for b2 = 1 to 60  'start a for..next loop
pause 60000       'wait 1 minute
next b2           'next loop
```

The PICAXE microcontroller works at a nominal 4MHz, but due to device manufacturing tolerances there is likely to be a drift of a few seconds over long time periods (e.g. a day). Note that the Serial LCD/Clock module (AXE033) has a precision clock and 'alarm clock' function that can be used to trigger the PICAXE at predefined interval or at certain time/dates with much greater precision. The X parts can also be linked to the i2c DS13097 real time clock.

My program is too long! What can I do?

Tips for reducing program length (see BASIC Commands help file for more details):

1) Use 'let pins =' instead of multiple high/low commands
2) Use sub-procedures for repeated code
3) Try to reduce the use of sound and serout commands, which use a lot of memory
4) If using an LCD, store the messages in the AXE033 Serial LCD Module, rather than in the program
5) Use eeprom and read commands to store messages in data memory (see next page)
6) Restructure your program to reduce the number of 'goto' commands
7) Use a PICAXE chip with the largest memory (X1 or X2 parts)

You can use the 'PICAXE>Check Syntax' menu to test the length of your program without a download.

Do symbols increase the program length?

No, all symbols are converted back to 'numbers' by the computer software prior to download and so have no affect on program length. You can use as many symbol commands as you wish.

What notes are generated by the sound command?

The sound command generates different 'beep' sounds for the values 1-127. The tune and play commands on the PICAXE-08M are specifically designed to play tunes. See the tune command in section 2 of the manual for more details.

I need more outputs - what can I do?

Use the PICAXE-28X/28X1 or 40X/40X1 which can have up to 16 outputs. Or connect a single output (e.g. output7) from a first PICAXE chip to input0 of a second PICAXE-18 chip. Program the second PICAXE-18 chip with this simple program:

```
main: serin 0,N2400,b1
      let pins = b1
      goto main
```

The eight outputs of the second chip can now be controlled with a serout 7,N2400,(b2) command by the first chip, where b2 contains the 'pins' value (0 to 255) desired on the second chip. This gives you a total of 15 useable outputs.

I need more inputs - what can I do?

Use a PICAXE-28X1 or 40X1, which can be configured to have a large number of inputs. Remember that analogue inputs can also be used as digital inputs if required, just see if the 'readadc' value is greater or less than 100. In many applications switches can also be connected in parallel on a single input pin.

How do I test more than one input at once?

Use the following command to test two inputs together

```
if pin0 = 1 and pin1 = 1 then...
```

or either of two inputs

```
if pin0 = 1 or pin1 = 1 then...
```

Appendix I - Advanced Technical Information and FAQ

This appendix provides advanced technical data for users who wish to understand more advanced technical data about the PICAXE microcontrollers. This information is not required for normal PICAXE use.

These notes presume the user is familiar with PIC microcontrollers, their configuration fuse settings and programming in assembler code.

What is a PICAXE microcontroller?

A PICAXE microcontroller is a Microchip PIC microcontroller that has been pre-programmed with the PICAXE bootstrap code. The bootstrap code enables the microcontroller to be reprogrammed without the need for an (expensive) conventional programmer, making the whole download system a very low-cost simple serial cable!

The bootstrap code also contains common routines (such as how to generate a pause delay or a sound output), so that each download does not have to waste time downloading this commonly required data. This makes the download time much quicker.

Why use the PICAXE instead of assembler / C?

The PICAXE uses a simple BASIC language (or flowcharts) that younger students can start generating programs with within an hour of first use. It is much easier to learn and debug than either C or assembler code.

The second advantage is the direct cable download method. The software is free and so the only cost per computer is a low-cost download cable. This enables students to buy their own cable and for schools to equip **every** single computer with a download cable. Other systems that require an expensive programmer are generally too expensive to implement in this way.

Finally as the PICAXE chip never leaves the board, all leg damage (as can occur when the chip is moved back and forth from a programmer) is eliminated.

How is the program stored within the microcontroller?

The program is stored in either data or program memory depending on the microcontroller type. The following table shows how program, read/write/eeprom data and readmem/writemem data is stored.

	Program	Read/Write	Readmem/Writemem
PICAXE-08M	Data	Data	N/A
PICAXE-14M2	Program	Data (256)	N/A (use i2c)
PICAXE-18M2	Program/Data	Data (256)	N/A (use i2c)
PICAXE-20M2	Program	Data (256)	N/A (use i2c)
PICAXE-18X	Program	Data (256)	N/A (use i2c)
PICAXE-28X	Program	Data (128)	N/A (use i2c)
PICAXE-28X1	Program	Data (256)	N/A (use readtable or i2c)
PICAXE-28X2	Program	Data (256)	N/A (use readtable or i2c)
PICAXE-40X	Program	Data (128)	N/A (use i2c)
PICAXE-40X2	Program	Data (256)	N/A (use readtable or i2c)

The program and read/write memory is overwritten with every download. Use the eeprom command to preload data (within the program) for the read/write commands. The readmem/writemem memory is not changed during a download.

How many times can the microcontroller be reprogrammed?

PICAXE chips can be reprogrammed at least 100,000 times. Note these are minimum values and the actual values may be much greater.

How is a download started?

When the computer starts a download an interrupt is generated on the serial input pin on the PICAXE. This interrupts the main program and puts the PICAXE into a state for a new download to be received. Therefore you must ensure that the 'serial in' pin is tied to ground (0V) via the 22k/10k resistors on ALL project boards for reliable operation of the microcontroller (to prevent unwanted 'floating pin' interrupt signals).

What are the electrical characteristics of the PICAXE (e.g. operating voltage range etc.)?

The electrical characteristics of the PICAXE microcontroller is dependent upon the base PIC microcontroller that is programmed with the PICAXE bootstrap code to create the PICAXE microcontroller. Therefore see the Microchip datasheet (from www.microchip.com) for the appropriate microcontroller characteristics. The lowest recommended operating voltage from these datasheets is 3V (Note this is the 'operating voltage' only. You may require a higher voltage (minimum 4.5V recommended) whilst doing the actual serial download from the computer to ensure accurate memory programming of the chip). X2 parts are also available in special 1.8V to 3.3V variants.

Does the PICAXE set the watchdog timer fuse?

Yes, the watchdog timer is set and used within a number of commands such as sleep and nap. The user cannot alter it's settings.

Does the PICAXE set the power-up timer fuse?

Yes.

Does the PICAXE set the brown-out fuse?

Yes for the M, M2, X1 and X2 parts, no for other parts. An unfortunate side effect of the brown-out fuse on the other parts is that it restricts the lowest operating voltage of the micro-controller to about 4.2V. As many users wish to use 3V battery packs, the brown-out fuse is not set on the PIC microcontrollers with a 4.2V brown-out.

The enablebod/disabledbod command can enable/disable the brown out function on M, M2, X1 and X2 parts.

How does the PICAXE do ADC (analogue-to-digital) conversions?

The (discontinued) PICAXE-08 and PICAXE-18 used the internal comparator to do a low resolution ADC step comparison, providing 16 discrete analogue values. The other PICAXE microcontrollers all use the internal ADC to do a full 256 step (8 bit) conversion. Although the microcontrollers are technically capable of 10 bit conversions, this is converted by the readadc command into byte (8 bit) values for ease of use via the byte (b1 etc.) variables, which makes the maths easier for students. This gives a resolution of about 0.02V (at 5V supply) which is adequate for almost all educational projects. Most parts also have a separate 10 bit adc read option (1024 steps), via the readadc10 command.

Can you supply the bootstrap program so that I can make my own PICAXE?

No. The small royalty made on each PICAXE chip sold is the only financial benefit to our company to support the PICAXE system - the software is free and the cables/development kits are sold at very low cost. Therefore we do not allow anyone else to manufacture PICAXE microcontrollers.

Can I mix assembler in with the BASIC code?

No. The program and bootstrap code cannot be 'mixed' with assembler code, this is not good programming practice. However you can achieve the same goal by converting your BASIC into assembler code using the automatic conversion feature, and then editing the converted assembler code program (see below).

Can I see the assembler code that is downloaded into the PICAXE?

If you own a Revolution Serial PIC Programmer (part BAS800), you can convert PICAXE BASIC programs into assembler code, to program blank PICs or to just learn how assembler code works by 'disassembly'. However some of the more complex commands (e.g. serin) are not supported, and the assembler code program generated is optimised for sequential learning (not optimised for compactness as with the PICAXE system) and so the code is not identical to that downloaded to the PICAXE.

Can you alter the input/output pin arrangement of the PICAXE microcontroller?

The PICAXE-08 has 5 pins that can be configured as desired. The 28 and 40 pin PICAXE can also be altered to give more inputs or outputs. The 18 pin input/ output pin arrangements are fixed and cannot be altered.

How long a program can I download into the PICAXE microcontroller?

This varies on the commands used, as not all commands use the same amount of memory.

There is no fixed 'byte' formula as to memory usage e.g. pause 5, pause 50 and pause 500 will all take different amounts of memory space! To calculate memory usage use the 'Check Syntax' option from the PICAXE menu. This will report the amount of memory used.

Do symbols increase the program length?

No, all symbols are converted back to 'numbers' by the computer software prior to download and so have no affect on program length. You can use as many symbol commands as you wish.

Do I need to erase the device?
How do I stop a program in the PICAXE microcontroller running?

Each download automatically overwrites the whole of the previous program. There is generally no need to erase the memory at any point. However if you want to stop a program running you can select the 'Clear Hardware Memory' menu to download an 'empty' program into the PICAXE memory.

Why is an 'empty' program not 0 bytes long?

Each downloaded program contains some configuration data, and an 'end' command is always added automatically to the end of each downloaded program. Therefore an 'empty' program on screen will not generate a zero byte program. To preventhe automatic end use the #no_end directive.

How vulnerable to damage are the microcontrollers?

The microcontrollers have a high level of static protection built into each pin and so generally handling them without any personal static protection in an educational (non-production) environment is acceptable.

Can I use i2c EEPROMs with the PICAXE?

The M2, X, X1 and X2 parts support all i2c parts via the hi2cin and hi2cout commands.

Can the PICAXE count pulses?

The M, M2, X, X1 and X2 parts support the count command which can count the number of pulses in a defined period. All parts support the pulsin command to measure the length of a pulse.

Can I control servos using the PICAXE?
Can I do PWM control of a motor using the PICAXE?

The M, M2, X, X1 and X2 parts have a dedicated pwmout command which acts on one or two of the pins for full pwm control.

These parts also have a 'servo' command that allows control of up to 8 servos (one on each output). The servo command users the internal timer and an interrupt, so that the pulses are maintained 'in the background' all the time that the PICAXE is running the main program.

The servo command produces a pulse of length 0.01ms to 2.55 ms approximately every 20ms. Therefore it can also be used as a simple background PWM output with PWM mark:space ratios between 1:2000 and 1:8 (approx).

How fast does the PICAXE operate?
Can I overclock the PICAXE?

All parts have an internal 4MHz/8MHz resonator, and the PICAXE-28/40 family can optionally also use an external ceramic resonator. This means the microcontroller processes 1 million assembler commands a second, which equates to roughly about 1000 BASIC commands per second. Different commands take different times to execute depending on how complex their 'assembler code' is.

All parts can be overclocked up to 64MHz (see the Over-clocking Appendix for restrictions).

Why does the PICAXE only support up to 4800 baud rate on serout/serin commands? Can I send and receive serial data via the download cable?

The maximum baud rates were originally selected for reliable operation with microcontrollers with internal resonator. The early internal resonators were not as accurate as an external device, and a slower baud rate ensures reliable operation. The M2, X1 and X2 parts support much higher baud rates via the hardware EUSART using the hserout command.

Many parts can send data via the download cable via a 'sertxd' command and receive data via the 'serrxd' command.

Does the PICAXE support interrupts?

The PICAXE uses the internal microcontroller interrupts for some of it's BASIC commands (e.g. servo). Therefore the internal interrupts are not available for general use. However the A, M and X parts all support a single 'polled' interrupt on the input port. Use the 'setint' BASIC command to setup the desired interrupt port setting to enable the polled interrupt. The polled interrupt scans the input port between every BASIC command (and constantly during pause commands), and so activates very quickly.

Software Version

The latest version of the Programming Editor and all other titles can be downloaded from the following website:

www.picaxe.co.uk

A very active forum for the discussion of PICAXE projects, and for technical support, also exists at

www.picaxeforum.co.uk

Contact Address

Revolution Education Ltd
http://www.rev-ed.co.uk/

Acknowledgements

Revolution Education would like to thank the following:

Clive Seager
John Bown
LTScotland
Higher Still Development Unit
UKOOA

