

# USING I2C WITH PICAXE

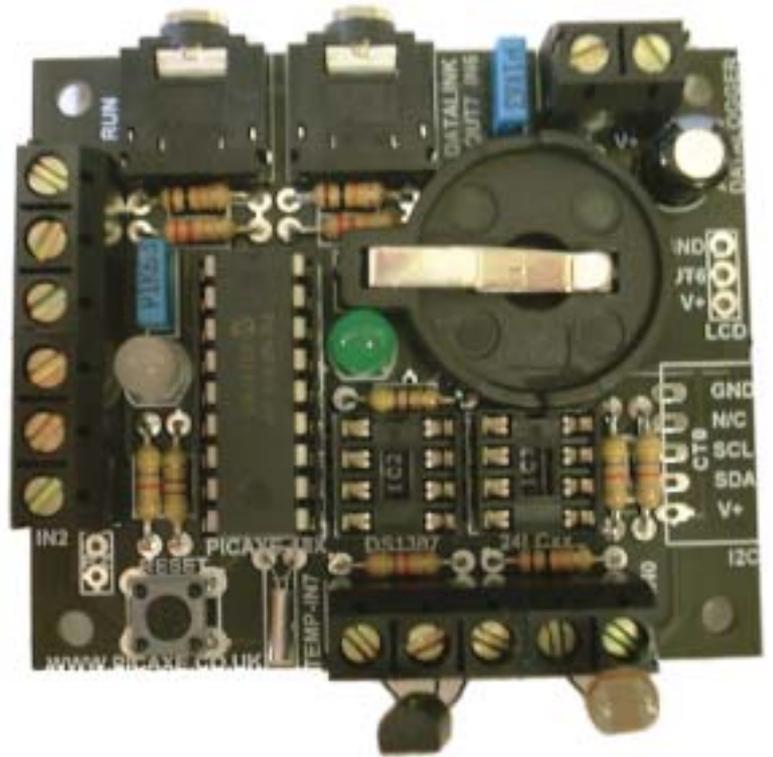
## Contents:

This article provides an introduction into how to use i2c parts with the PICAXE system.

This article:

- 1) Describes the i2c bus
- 2) Explains how the i2c bus is used with the PICAXE system
- 3) Gives an example of using the i2c bus with a 24LCxx series EEPROM
- 4) Gives an example of using the i2c bus with a DS1307 real time clock.
- 5) Gives an example of using the i2c bus with a SPE030 speech synthesizer.

All the information in this datasheet applies to the PICAXE-X parts (18X, 28X, 40X). If you wish to experiment with use of the i2c bus, we recommend use of the AXE110 Datalogger fitted with the AXE034 Real Time Clock Upgrade. This will provide you with a board that has the PICAXE-18X, a 24LC16B EEPROM memory chip, and a DS1307 real-time-clock chip fitted.

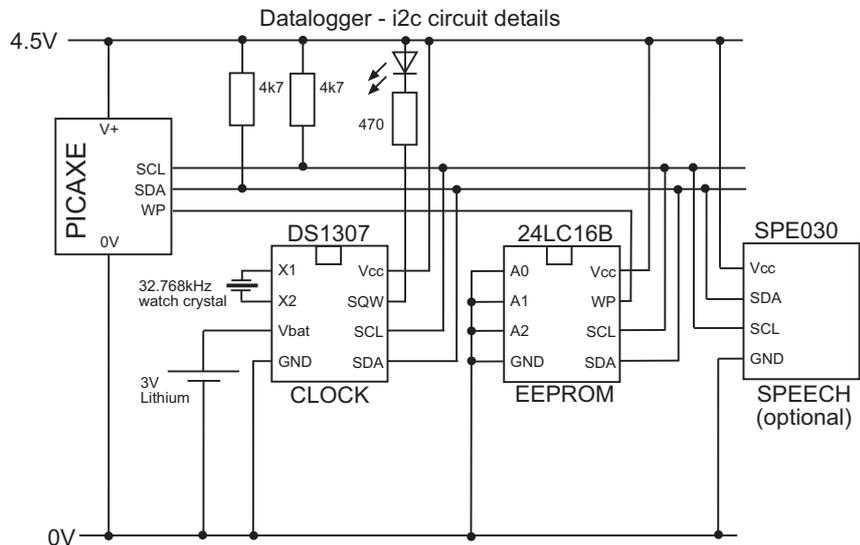


## Terms used in this article:

IC	- integrated circuit or 'chip'
Master	- a microcontroller IC that 'controls' the operation of a circuit
Slave	- a slave IC that does certain specialised tasks for the master IC
Byte	- a number between 0 and 255
Register	- a memory location within the slave that stores 1 byte of data
Register Address	- an address that 'points' to a particular memory register
Block	- group of 256 registers
EEPROM IC	- a slave IC that can store a large amount of data
RTC IC	- a slave IC that can maintain the date / time (real-time-clock)
ADC IC	- a slave IC that can perform analogue-to-digital conversions

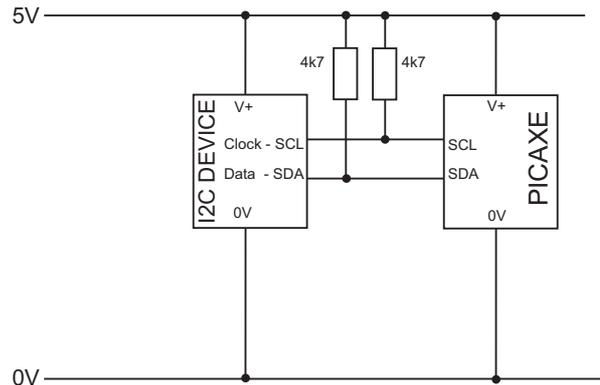
## Sample i2c circuit

PICAXE-18X datalogger circuit.  
Shows connections of  
- 24LC16B EEPROM  
- DS1307 RTC  
- SPE030 Speech Module



## What is the i2c bus?

The Inter-Integrated-Circuit (i2c) bus was originally developed by Phillips Inc. for the transfer of data between ICs at the PCB level. The physical interface of the communication bus consists of just two lines – one for the clock (SCL) and one for the data (SDA). These lines are pulled high by resistors connected to the V+ rail. 4k7 is a commonly used value for these resistors, although the actual value used is not that critical. When either of the master or slave ICs want to 'transmit', they pull the lines low by transistors built inside the IC.



The IC that controls the bus is called the Master, and is often a microcontroller – in this article a PICAXE-18X microcontroller will be used as the master device. The other ICs connected to the bus are called Slaves. There can be more than one slave on the bus, as long as each slave has been configured to have a unique 'slave address' so that it can be uniquely identified on the bus. In theory there are up to about 112 different addresses available, but most practical applications would generally have between 1 and 10 slave ICs.

## Why use the i2c bus?

### Advantages:

- Most major semiconductor manufacturers produce many low-cost i2c compatible ICs. The range of ICs available is quite extensive - memory EEPROMs, real-time-clocks, ADCs, DACs, PWM motor/fan controllers, LED drivers, digital potentiometers, digital temperature sensors etc. etc.
- Many of these ICs come in small 8 pin packages. This makes the circuit design very straight forward.
- Many slave devices can be connected to the same bus, which only uses two of the microcontroller pins. This is a very efficient use of the microcontroller pins.
- The bus design is very simple, using just two lines and two resistors.

### Disadvantages:

- The i2c bus communication protocol is quite complicated. However this can be easily overcome by using microcontroller systems such as PICAXE, which provide simple BASIC style commands for all the i2c data transfers, and therefore the end user needs no technical knowledge of the bus communication protocols.
- Each slave IC will have a few unique setup parameters (e.g. slave address), which must be extracted from the manufacturers datasheet. This is not normally that difficult, once you know the main parameters that you are looking for!

## Slave Configuration Parameters

Although all i2c slave devices work in roughly the same way, there are four parameters that must be checked from the manufacturer's datasheet for each slave device used.

### Parameter 1 - Slave Address

As already mentioned, each slave IC on the i2c bus must have a unique address. This is not generally a problem when using different types of IC on the same bus, as most ICs have a different default slave address.

The slave address is generally 7 bits long, with the 8<sup>th</sup> bit reserved to indicate whether the master wishes to write to (1) or read from (0) the slave. A 10-bit slave address is also possible, but is rarely used and so not covered in this article. This means the slave address is often quoted in datasheets as, for instance, 1010000x, with x indicating the read/write bit. When using the PICAXE system the state of this 8<sup>th</sup> bit is not important, as the PICAXE system will automatically set or clear the bit as necessary for a read or a write.

However it is also possible that you may want to use two or more of the same type of IC (e.g. memory EEPROM) on the same bus. This can be overcome by the use of external address pins on the slave device, which can be connected (on the PCB design) to either V+ or 0V to give each slave IC on the PCB a unique address. In the case of the popular 24LCxx series of EEPROMs there are 3 external address pins (A2, A1 and A0). By connecting these pins to V+ or GND on your circuit design, you can ensure that up to 8 parts can be uniquely identified on the same bus.

For these ICs the datasheet slave address may be quoted as, for instance, 1010ddd, where d is 1 or 0 depending on the state of the external address pin A2-A0.

### Parameter 2 -Bus Speed (100 or 400kHz)

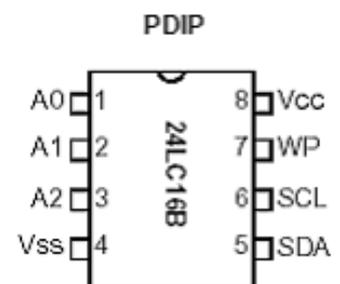
The maximum bus speed for data transfer between the master and slave is normally 400kHz. However some parts will only work up to 100kHz, and so the manufacturer datasheet should be checked for each slave IC used. Note this is the maximum speed - all parts can be run at the slower speed if desired.

### Parameter 3 - Register Address Size (Byte or Word)

All data transfer from the master to the slave is a 'write', and this means that a byte of data is transferred from the master to a 'register' within the slave IC. All data transfer from the slave to the master is a 'read'. Simpler slave devices have a maximum of 256 registers, and so a 'register address' of one byte length can be used to identify the particular register of interest. However larger devices, particularly memory EEPROMs, have more than 256 registers and so may need a 'word' (two byte) register address instead.

### Parameter 4 – Page Write Buffer

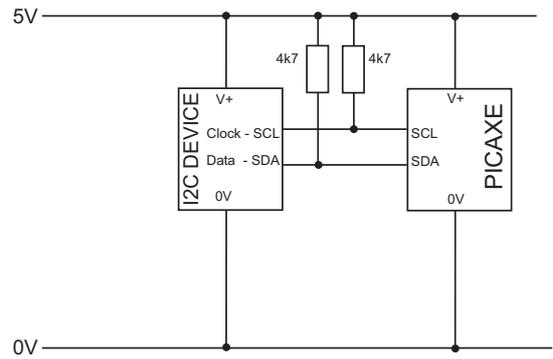
All EEPROM memory chips require a 'write time' to save the data in the chip. This is typically 5 or 10ms. When writing lots of data, this can cause a significant delay. To help overcome this issue, many ICs have a page write buffer that can accept more than one byte at once (typically 8, 16 or 32 bytes) so that all these bytes can be programmed at once. This means, for instance, in the case of 8 bytes you only have one 10ms delay, rather than 80ms of delay. Important Note: One of the biggest mistakes made by beginners is that they don't realise that page writes can only 'start' at a multiple of the buffer size, and cannot overflow the page buffer size. In effect this means (for an 8 byte buffer) you can write 8 bytes up from address 0 (or 8 or 16 etc.) but only up 6 bytes from address 2 (10, 18 etc.), or else you would overflow the 8 byte page write boundary.



## 2) Using i2c with the PICAXE System

### Hardware

All the PICAXE X parts (18X, 28X, 40X) have two pins which can be dedicated for the two i2c communication lines – SDA and SCL. The typical electrical configuration is shown here.



### Software

Communication with the slave device just requires three BASIC commands – `i2cslave`, `readi2c` and `writei2c`

#### `i2cslave`

The `i2cslave` command is used to set up the slave parameters for each slave IC. The syntax is

```
i2cslave slave_address, bus_speed, address_size
```

where `slave_address` is the address (e.g. %10100000)  
`bus_speed` is the keyword `i2cfast` (400kHz) or `i2cslow` (100kHz)  
`address_size` is the keyword `i2cbyte` or `i2cword` as appropriate

#### `writei2c`

The `writei2c` command is used to write data to the slave. The syntax is

```
writei2c start_address, (data,data,data,data...)
```

where `start_address` is the start address (byte or word as appropriate)  
`data` is bytes of data to be sent (either fixed values of variable contents)  
(multiple bytes of data can be sent, but care should be taken not to exceed the page buffer size)

#### `readi2c`

The `readi2c` command is used to read data back from the slave into variables in the PICAXE. The syntax is

```
readi2c start_address, (variable, variable,...)
```

where `start_address` is the start address (byte or word as appropriate)  
`variable` is where the returned data is stored in the master (b0, b1, b2 etc)

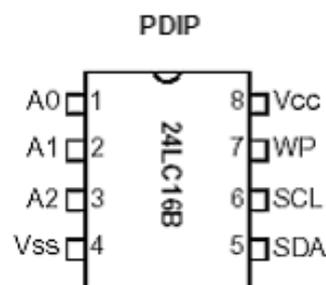
#### Example

To write the text “hello” (actually 5 bytes of data – one byte for each letter) to a 24LC16B memory IC and then read it back into variables, the program would be

```
i2cslave %10100000, i2cfast, i2cbyte    ` set slave parameters
writei2c 0, ("hello")                  ` write the text
pause 10                                ` wait 10ms write time
readi2c 0, (b0,b1,b2,b3,b4)            ` read the data back again
debug b0                                ` display data on screen
```

### 3) Practical Example – 24LC16B EEPROM

Many projects involve the storage of data. This may be data collected during a datalogging experiment, or pre-configured data built into the circuit at the time of build (e.g. messages in different languages to be displayed on an LCD screen). The PICAXE chips can generally store 128 or 256 bytes of data internally, but some projects may require much more than this, and so an external memory storage IC is required.



External EEPROM (electrically-erasable-programmable-read-only-memories) ICs can be used to store large amounts of data. Most EEPROMs store data in 'blocks' of 256 registers, each register storing one byte of data. Simplest EEPROMs may only have one block of 256 registers, more expensive EEPROMs can have up to 256 blocks, giving a total of  $256 \times 256 = 65536$  (64k) memory registers.

The 24LCxx series EEPROMs are probably the most commonly used i2c EEPROM devices. Many manufacturers make these parts, but we will only consider Microchip brand parts in this article because these tend to be readily available via mail order catalogues. These EEPROMs can be written to over 1 million times, and the EEPROM memory also retains data when the power is removed. Pin 7 of the IC is a write-enable pin that can prevent the data being corrupted (keep the pin high to prevent data being changed). Often this pin is connected to a microcontroller pin, so that the microcontroller can control when data can be written (pull pin low to enable writes).

The cheapest EEPROMs (e.g. Microchip parts ending in the letter 'B') only use a single byte register address, which by definition can only uniquely identify 256 registers. This means that the various blocks (if they exist) must be identified in a different way. The 24LC16B has 8 blocks, the other EEPROMs have less (see table below). The way these cheap EEPROMs overcome this address problem is by merging the block address into the slave address. This means, in effect, that a single 24LC16B 'appears' on the i2c bus as 8 different 'slaves', each slave having a unique address and containing 256 registers. This system might at first appear quite strange, but the IC is constructed this way to keep the cost of the IC to a minimum. However this system does have the downfall that only one part can be used per bus (the external IC pins A2-A0 are not actually physically connected within these cheaper 'B' parts).

The more expensive EEPROMs (24LC32 upwards) use a word register address, and so the block address can be incorporated within the normal register word address. This means the EEPROM appears on the i2c bus as a single slave, and so up to 8 identical devices can be connected to the bus by configuring the external A2 to A0 address pins accordingly. Using 8 of the commonly available 24LC256 EEPROMs will give a huge 2Mb of memory!

Device	Registers	Buffer	Slave	Speed	Address
24LC01B	128	8	%1010xxxx	i2cfast (400kHz)	i2cbyte
24LC02B	256	8	%1010xxxx	i2cfast (400kHz)	i2cbyte
24LC04B	512	16	%1010xxbx	i2cfast (400kHz)	i2cbyte
24LC08B	1k (1024)	16	%1010xbbx	i2cfast (400kHz)	i2cbyte
24LC16B	2k (2048)	16	%1010bbbx	i2cfast (400kHz)	i2cbyte
24LC32A	4k (4096)	32	%1010dddx	i2cfast (400kHz)	i2cword
24LC65	8k (8192)	64	%1010dddx	i2cfast (400kHz)	i2cword
24LC128	16k (16384)	64	%1010dddx	i2cfast (400kHz)	i2cword
24LC256	32k (32768)	64	%1010dddx	i2cfast (400kHz)	i2cword
24LC512	64k (65536)	128	%1010dddx	i2cfast (400kHz)	i2cword

where b = block address (internal to EEPROM)

d = device address (configured by external pins A2, A1, A0)

x = don't care

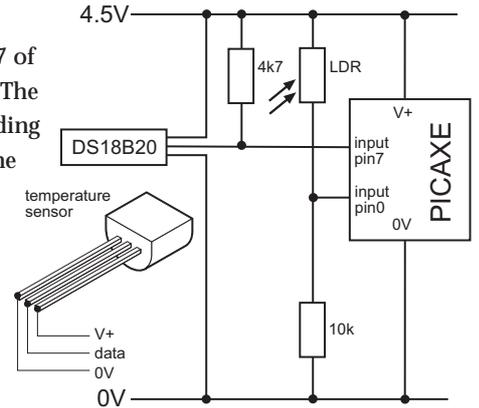
## Saving Data

The following example reads the data from a digital temperature sensor (on input 7 of the PICAXE master) and a LDR light sensor (on input 0) every minute for an hour. The data is saved in a 24LC04B, 24LC08B or 24LC16B EEPROM. Each temperature reading is saved in the first block (000) of the memory, and each light reading is saved in the second block (001) of the memory. A for..next loop is used to repeat the action 60 times, and the loop counter value (0 to 59) is used as the address to save the data within the appropriate memory block.

```

for b1 = 0 to 59                ` start for..next loop
  readtemp 7,b2                ` read temp value from 7
  i2cslave %10100000, i2cfast, i2cbyte ` set block0 parameters
  writei2c b1,(b2)             ` write the value
  pause 10                     ` wait EEPROM write time
  readadc 0,b3                 ` read light value from 0
  i2cslave %10100010, i2cfast, i2cbyte ` set block1 parameters
  writei2c b1,(b3)            ` write the value
  pause 60000                  ` wait 1 minute
next b1                         ` next loop

```



## Reading Data

The following example reads back the data saved in the example above, and displays the data on a Serial LCD Module (part AXE033). The serout command is the command that transmits the data from the PICAXE master to the serial LCD module (connected on output 6). The temperature value is shown on the top line of the display, the light value on the bottom line of the display. Each reading is displayed for 2 seconds.

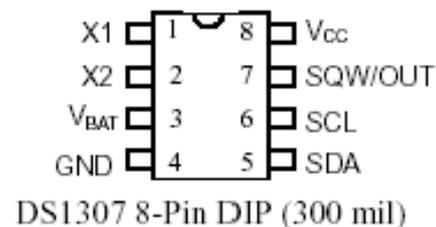
```

for b1 = 0 to 59                ` start for..next loop
  i2cslave %10100000, i2cfast, i2cbyte ` set block0 parameters
  readi2c b1,(b2)                ` read the temp value
  i2cslave %10100010, i2cfast, i2cbyte ` set block1 parameters
  readi2c b1,(b3)                ` read the light value
  serout 6,N2400,(254,128,"Temp Value =",#b2, " ") ` display temp
  serout 6,N2400,(254,192,"Light Value =",#b3, " ") ` display light
  pause 2000                     ` wait 2 seconds
next b1                           ` next loop

```

#### 4) Practical Example – DS1307 Real Time Clock

The Maxim/Dallas Semiconductor DS1307 is an accurate real-time-clock, which automatically maintains the current time and date, including compensation for months with less than 31 days and leap years. The DS1307 is an 8 pin device, and a standard low-cost 32.768 kHz 12pF quartz watch crystal is connected to pins 1 and 2 to provide the accurate time base. An optional 3V lithium 'backup' cell can also be connected to pin 3, this ensures that the time is kept up to date when the main circuit power is removed. The IC automatically detects removal of the main power source and moves to the lithium cell power as and when required. The cell should last at least ten years.



The DS1307 also has two additional features of interest. Pin 7 is an open collector output that can be programmed to 'flash' at 1Hz. This allows an LED to be attached as a 'seconds indicator' in clock applications. The IC also contains 56 bytes of general purpose RAM, which can be used as extra memory by the master if required.

From the manufacturers datasheet for the DS1307 ([www.dalsemi.com](http://www.dalsemi.com)), the following i2c details can be found:

slave address        - 1101000x  
 address size        - 1 byte  
 bus speed            - 100kHz

The registers of the DS1307 are defined as follows:

Address	Register
00	Seconds (0-59)
01	Minutes (0-59)
02	Hours (0-23)
03	Day of Week (1-7)
04	Date (1-31)
05	Month (1-12)
06	Year (00-99)
07	Control (set to 16 (\$10))
08-\$3F	General Purpose RAM

All the time/date data is in BCD (binary-coded-decimal) format, which makes it very easy to read and write using hex notation e.g. 11:35am will contain \$11 in the hours register and \$35 in the minutes register.

## Setting the Time / Date

To set the correct time after the circuit is first powered up, the current time must be written to the registers. The following example PICAXE program will setup the time to 11:59:00 on Thursday 25/12/03.

This is carried out by loading the registers in order from address 00 upwards i.e. seconds then minutes then hours etc.

```
i2cslave %11010000, i2cslow, i2cbyte
writei2c 0, ($00, $59, $11, $03, $25, $12, $03, $10)
end
```

## Reading the Time / Date

To read the current time you can use the following program to load variables within the PICAXE with the various register values from the DS1307. Calculations can then be carried out to see, for instance, if a particular alarm point has been reached. This example program acts as an alarm clock, checking the time every 30 seconds. If the time is exactly 07:00 then a buzzer, connected to output 7, will sound for 20 seconds.

```
i2cslave %11010000, i2cslow, i2cbyte      ` set slave parameter

loop:
  pause 30000                             ` wait 30 sec
  readi2c 0, (b0, b1, b2)                 ` read sec, min, hour
  if b2 <> $07 then loop                   ` if hour not 7 loop
  if b1 <> $00 then loop                   ` if min not 00 loop

  high 7                                  ` switch on buzzer
  pause 20000                             ` wait 20 sec
  low 7                                   ` switch off buzzer
  pause 60000                             ` wait 60 sec to prevent repeat
  goto loop                               ` loop
```

## 5) Practical Example – SPE030 Speech module.

The SPE030 module is a speech synthesizer that will speak the text sent to it over the i2c bus.

From the SPE030 datasheet (spe030.pdf), the following i2c details can be found:

```
slave address    - $C4
address size     - 1 byte
bus speed        - 400kHz
```

The following program will generate the speech "hello PICAXE user". For further information see the SPE030 datasheet (spe030.pdf).

```
i2cslave $C4,i2cfast,i2cbyte
writei2c 0,(0,0,5,3,"hello pickacks user",0)
writei2c 0,(64)
```