PIC Tutorial Hardware

The hardware required consists of a number of small boards (built on Veroboard), which connect together via ten pin leads using Molex connectors. The first board (Main Board) carries the PIC16F628 processor and 5V regulator - the board can be fed from a simple 9V battery. Some of the later tutorials will require two processor boards, this is the reason for the second connector on PortB - the two processors will communicate with each other over a standard 9600 baud serial bus, the second board can be either powered from the first (using a four wire connection lead), or powered from it's own supply (using a three wire connection lead). The lead consists of a ground wire, RB1 to RB2, RB2 to RB1, and an optional 5V wire. RB1 and RB2 cross over so we can experiment with the built-in hardware USART as well as software serial communications.

I've added a second processor board, based on the PIC16F876, this adds a third port, and includes 5 channels of 10 bit analogue input - the existing tutorials based on the PIC16F628 should work with a few slight changes, these are explained on the changes page, as I'm running the 16F876 at 20MHz (5 times faster than the 16F628) the delay routines will need altering as well.

The second board (LED Board) carries eight LED's with associated series resistors, and is used in the first series of tutorials. The third board (Switch Board) provides a row of four switches, and four LED's (so you can do some exercises without needing the previous LED board). The fourth board (LCD Board) only has a variable resistor (contrast) and a single resistor (pull-up for RA4), the actual LCD module is mounted off board and connected via another 10 way Molex connector, this allows you to plug different LCD's in. The fifth board (Joystick Board) provides an interface for a standard PC analogue joystick, giving access to the two analogue controls and the two buttons. The sixth board (IR Board) has an Infrared transmitter and receiver, using two of them with two processor boards we can experiment with Infrared communication. The seventh board (I2C EEPROM Board) uses a standard EEPROM 24Cxx series (I used a 24C04 and a 24C256). With I2C there are a great many components you can connect to the bus, the basic software interface remains pretty well the same, except that some chips (like the 24C256) use an extended addressing mode to access more memory, the standard addressing mode can only access 2kB (8 x 256 byte pages). I'll be adding some other I2C based boards later, they will use the same basic I2C routines as the existing I2C EEPROM board does. The eighth board (I2C Clock Board) implements a battery backed clock, using a PCF8583P chip, and the ninth one (I2C A2D Board) introduces analogue to digital conversion, using a PCF8591P chip. The tenth board (I2C Switch Board) is very simple, it provides four push button switches for use with the other I2C boards. The eleventh board is the PIC16F876 processor board, and the twelfth is an RS232 interface board using the standard MAX232 chip.

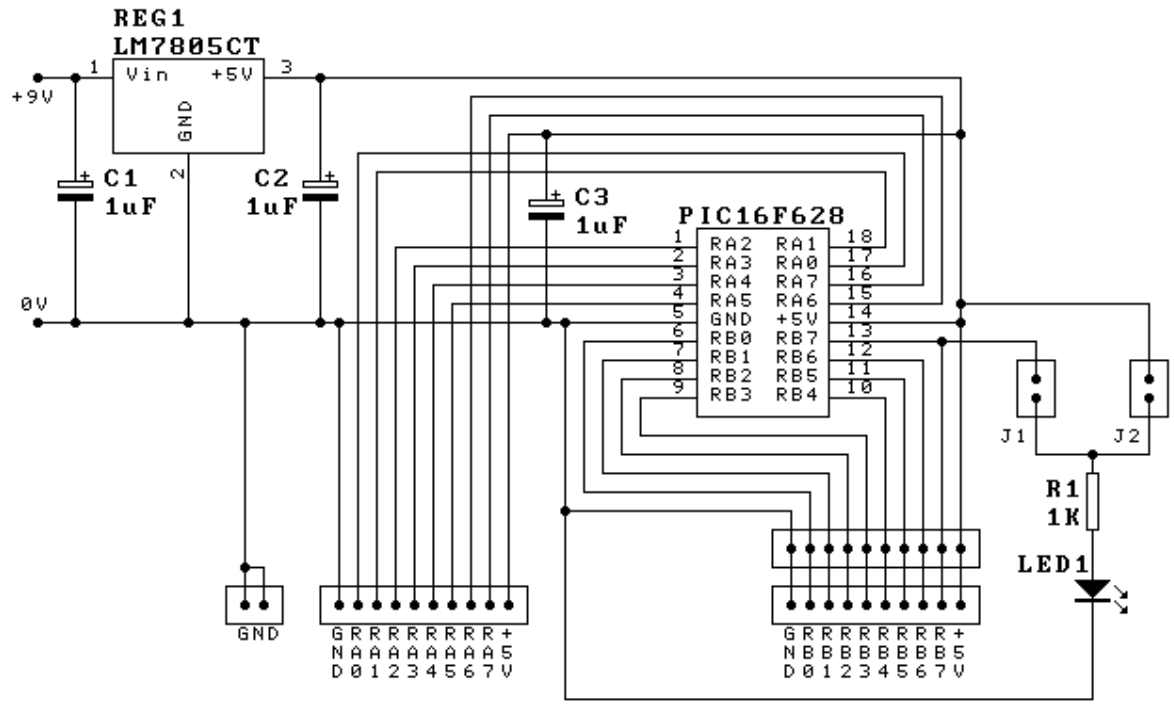
The various boards.	
Main Board	The main 16F628 processor board (two required later).
Main Board Two	A 16F876 based processor board.
LED Board	Eight LED's for displaying the outputs from one port.
Switch Board	Four pushbutton switches for connecting to one port.
LCD Board	An LCD text display board, in 4 bit mode, connecting to one port.
Joystick Board	A board for connecting an analogue PC joystick.
IR Board	An Infrared transmitter/receiver board (two required).
I2C EEPROM Board	An I2C EEPROM board.
I2C Clock Board	An I2C battery backed clock board.
I2C A2D Board	A four channel A2D converter via the I2C bus.
I2C Switch Board	Four push buttons for use with the I2C boards.
RS232 Board	An RS232 interface board.
Next Board	To be arranged!.

I obtained the Molex connector parts from RS Components, for the PCB part there are two options, the first has fully open pins, the second has plastic locking guides at the back, which means you can't get it on the wrong way round or out of step - use which ever you prefer, I initially used the open ones, but used locking ones on my second processor board and the IR Board. You can buy an expensive crimping tool for fitting the Socket Terminals to the wire, but I simply soldered them in place - it's a little fiddly, but reasonably easy - once the terminals are fitted on the wire they are easily pushed into place in the socket housing. I used a blue wire to mark pin one, and the rest were all white. I made a number of leads up, about 12cm long, with connectors at both ends, and a single ended one which solders to the LCD module. A special longer one, with only 4 wires (two of them crossed over) was made for cross connecting the two processor boards.

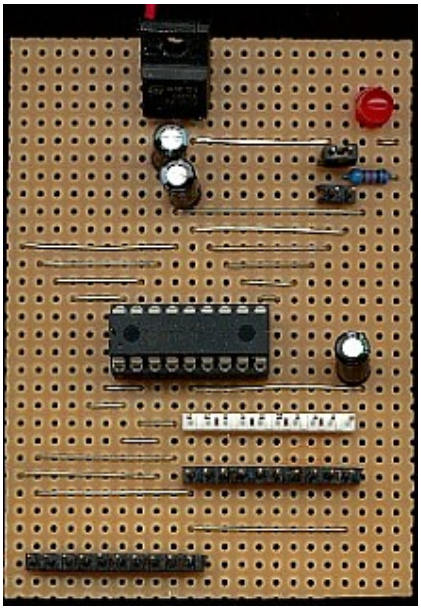
Connector parts used.		
Part Description	**RS Part Number**	**Quantity**
PCB Header (non-locking)	467-582	1 Pack (10)
PCB Header (locking)	453-230	1 Pack (10)
Socket Housing	467-633	1 Pack (10)
Socket Terminals	467-598	1 Pack (100)

PIC Tutorial Main Board

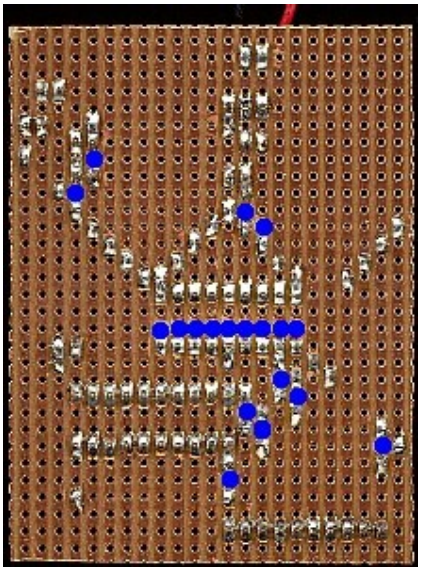
Main Board



This is the circuit of the main board for the tutorials, it consists of the PIC16F628, a 7805 regulator, 3 capacitors, three ten pin connectors, one for PortA, and two for PortB (the second for connecting two of these boards together), and a two pin ground test connection - optionally it also includes an LED, a resistor, and two 2 pin jumpers. Each of the three ten pin connectors is wired identically, with a ground connection at the left side, and a 5V connection at the right - this will allow you to plug the same extension board into either port, and help to demonstrate their differences. The capacitors C1 and C2 are to keep the 7805 stable, they have a tendency to oscillate without them, and C3 is just a decoupling capacitor placed near the chip, always a good practice (although PIC's do seem very tolerant of them). The jumpers J1 and J2 allow the LED to be connected either to 5V (J2) as a 'power on' indicator, or to RB7 (J1) where it can be switched by the port pin - this allows you to do something before you build any further boards. **Under no circumstances connect both J1 and J2 at the same time, it's likely to damage the chip.**



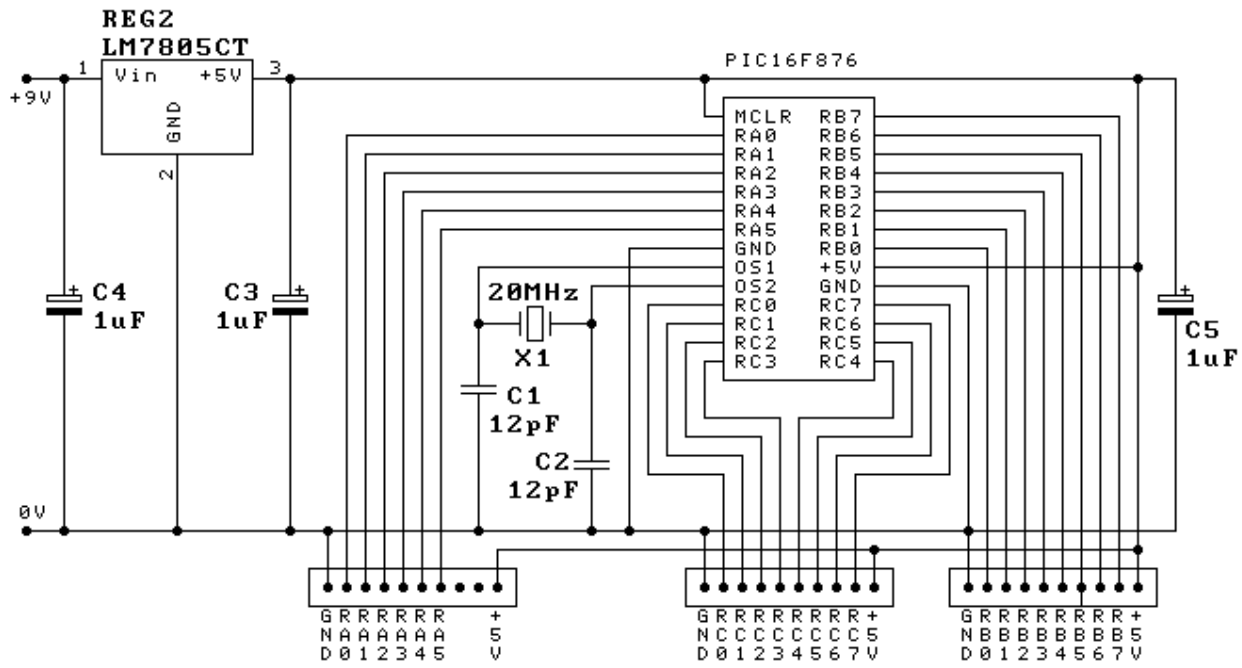
This is a photo of the main board, it's built on a piece of Veroboard 23 strips wide, by 31 holes high. The left of the two black connectors at the bottom is PortA, the right pair are PortB. All the wire jumpers are required to line the connectors up neatly. In order to prevent the pins of the PIC getting damaged, the PIC is permanently inserted in a 'turned pin' socket, this is then plugged into a normal socket on the board. To program it the PIC, complete with turned pin socket, is unplugged and inserted in the programmer, programmed and then returned. This is very easy to do, and the 'turned pin' socket prevents any damage. The PIC is capable of being programmed in-circuit, but it adds circuit complications and uses up I/O pins, so I haven't implemented that. J1 is the upper of the two jumpers, nearest the LED. Although it's not very easy to see in this picture, pin one of the PIC is to the left. The 2 pin ground test connection isn't fitted in this picture, it fits vertically just above C3, on the ground rail connecting to the negative end of C3.



This is a bottom view of the board, I've indicated the track cuts (19 of them) with blue circles, with this picture, and the one above, it should be fairly easy to duplicate the board - remember - there are 19 track cuts, and 21 wire links.

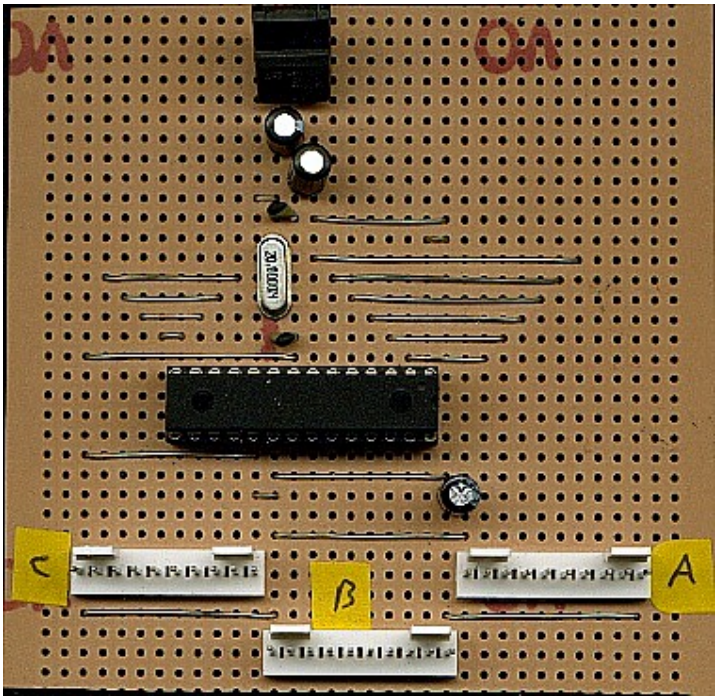
PIC Tutorial Main Board Two

Main Board Two

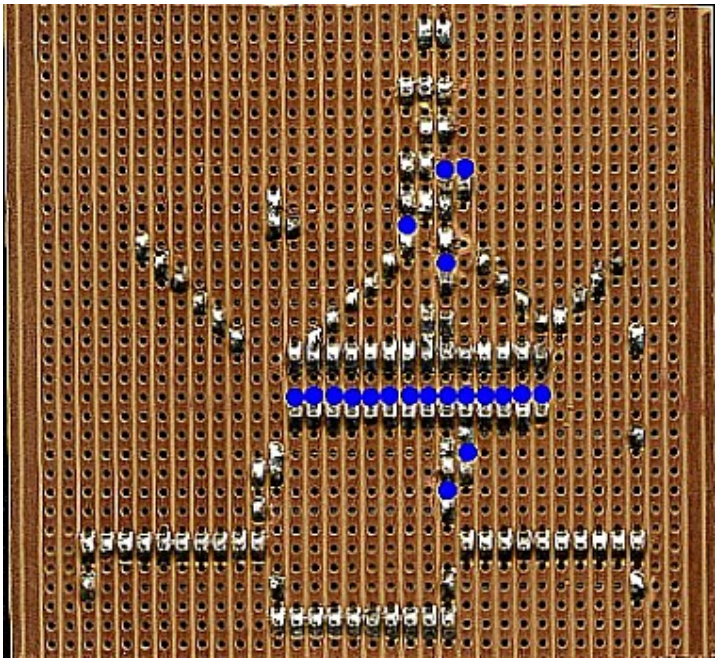


This is the circuit of the second main board for the tutorials, it consists of the PIC16F876, a 7805 regulator, a 20MHz crystal, 5 capacitors, three ten pin connectors, one for PortA, one for PortB, and one for PortC . Each of the three ten pin connectors is wired identically, with a ground connection at the left side, and a 5V connection at the right - this will allow you to plug the same extension board into any port, and help to demonstrate their differences - the most obvious difference is that PortA only has 6 I/O lines, which can be either digital I/O or analogue inputs, with 10 bit resolution.

Basically it's very similar to the 16F628 tutorial board, but has an extra port and added facilities - as the 16F876 doesn't have an internal oscillator a crystal is required for the clock oscillator - I choose a 20MHz crystal for this, if you can't get a 20MHz chip the 4MHz 16F876's seem perfectly happy to run at 20MHz - I suspect they are exactly the same chip, and graded to provide the two different versions.



This is a photo of the main board, it's built on a piece of Veroboard 34 strips wide, by 34 holes high. The left of the three white connectors at the bottom is PortC, the right one is PortA, and the middle one PortB (I stuck little labels on them as I keep forgetting which is which). All the wire jumpers are required to line the connectors up neatly. In order to prevent the pins of the PIC getting damaged, the PIC is permanently inserted in a 'turned pin' socket, this is then plugged into a normal socket on the board. To program it the PIC, complete with turned pin socket, is unplugged and inserted in the programmer, programmed and then returned. This is very easy to do, and the 'turned pin' socket prevents any damage. The PIC is capable of being programmed in-circuit, but it adds circuit complications and uses up I/O pins, so I haven't implemented that.



This is a bottom view of the board, I've indicated the track cuts (20 of them) with blue circles, with this picture, and the one above, it should be fairly easy to duplicate the board - remember - there are 20 track cuts, and 20 wire links.

PIC Tutorial Changes

Changes for the PIC16F876 board

The PIC16F876 is very similar to the 16F628, and uses the same 14 bit command set, so the differences are pretty small, but a few changes to the existing tutorial code is required.

1. **Initialisation code** - the processor type, include file, and configuration fuse settings need changing.
2. **Setup code** - the 16F628 requires the CMCON register setting to disable the comparator hardware, the 16F876 doesn't have this (although the 16F876A does, but isn't set by default). However, the 16F876 does have PortA set as analogue inputs by default, so these require setting as digital inputs in the setup code.
3. **Delay routines** - as we're running the 16F876 five times as fast, the delay routines require modifying to use five times as many cycles.
4. **PORT changes** - the 16F628 has two 8-bit ports A and B, the 16F876 has three ports - but only B and C are 8-bit, port A only has 6 pins available RA0-RA5, five of the six can be used as analogue inputs. So it's probably easiest to change all references to PortA and TrisA to PortC and TrisC, and connect to PortC in place of PortA.

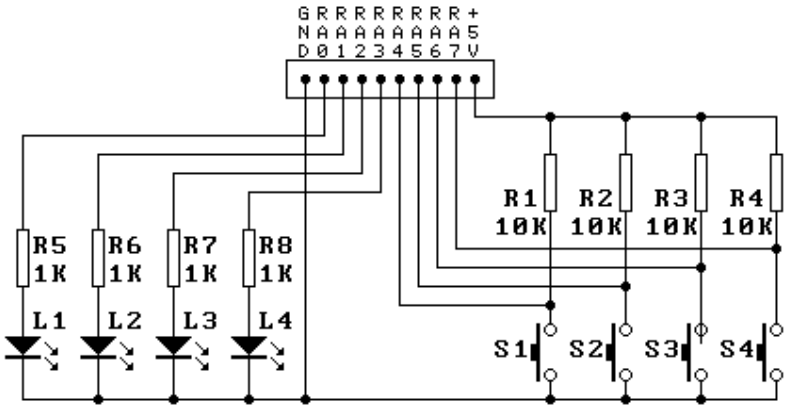
PIC16F628-4MHz	**PIC16F876-20MHz**
Initialisation code	
LIST p=16F628 include "P16F628.inc" __config 0x3D18	LIST p=16F876 include "P16F876.inc" __config 0x393A
Setup code	
movlw 0x07 movwf CMCON	BANKSEL ADCON1 movlw 0x06 movwf ADCON1 BANKSEL PORTA
Delay routines	
Delay255 movlw 0xff ;delay 255mS goto d0 Delay100 movlw d'100' ;delay 100mS goto d0 Delay50 movlw d'50' ;delay 50mS goto d0 Delay20 movlw d'20' ;delay 20mS goto d0 Delay10 movlw d'10' ;delay 10mS goto d0 Delay1 movlw d'1' ;delay 1mS goto d0	Delay255 movlw 0xff ;delay 255mS goto d0 Delay100 movlw d'100' ;delay 100mS goto d0 Delay50 movlw d'50' ;delay 50mS goto d0 Delay20 movlw d'20' ;delay 20mS goto d0 Delay10 movlw d'10' ;delay 10mS goto d0 Delay1 movlw d'1' ;delay 1mS goto d0

Delay5 5ms	movlw	0x05	;delay	Delay5 5ms	movlw	0x05	;delay
d0	movwf	count1		d0	movwf	count1	
d1	movlw	0xC7		d1	movlw	0xE7	
	movwf	counta			movwf	counta	
	movlw	0x01			movlw	0x04	
	movwf	countb			movwf	countb	
Delay_0	decfsz	counta, f		Delay_0	decfsz	counta, f	
	goto	$+2			goto	$+2	
	decfsz	countb, f			decfsz	countb, f	
	goto	Delay_0			goto	Delay_0	
	decfsz	count1 ,f			decfsz	count1 ,f	
	goto	d1			goto	d1	
	retlw	0x00			retlw	0x00	
PORT changes							
TRISA PORTA				TRISC PORTC			

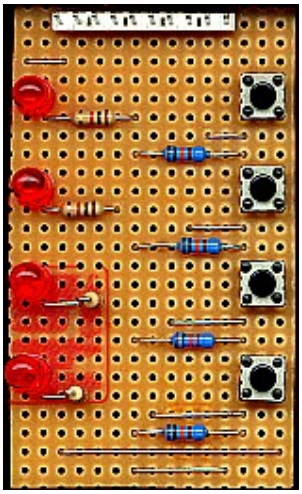
The changes above should allow the existing 16F628 tutorials to work on the 16F876 board, however the 16F876 has greater hardware capabilities than the 16F628, for example - 5x10 bit analogue inputs and two PWM outputs, both of these will be used in later tutorials, and will obviously not be possible on the 16F628.

PIC Tutorial - Switch Board

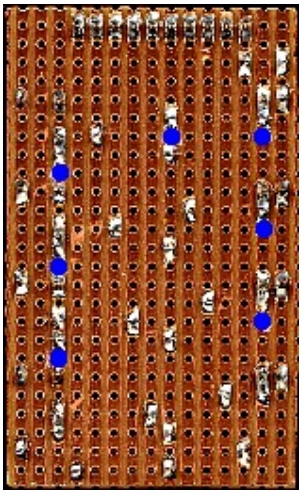
Switch Board



This is the Switch Board, a simple array of four pushbutton switches connected to the top four pins of one port, with four LED's connected to the bottom four pins of the same port (so you don't require the LED board as well). The switches connect to the top four pins of PortA, this is because RA5 can only be an input, and RA4 is an open-collector output - by using the top four pins it leaves the others available as general purpose I/O pins. Although it's labelled as connecting to PortA, it can also be connected to PortB if required.



This is a top view of the Switch Board, it consists of four switches, with pull-up resistors, and four LED's with associated current limiting resistors - two of which are mounted vertically.

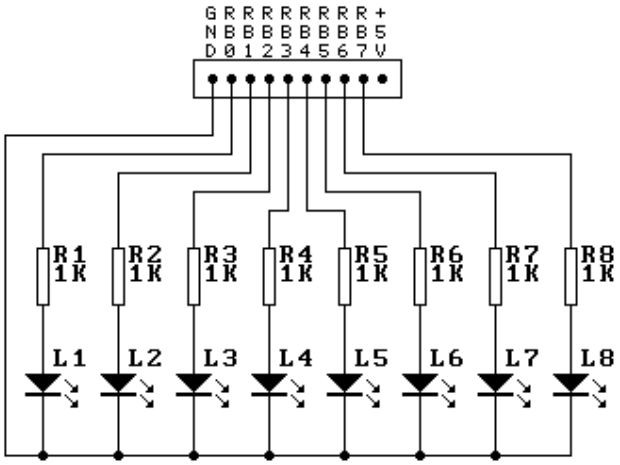


A bottom view of the Switch Board, the seven track cuts are marked with blue circles, and it has seven wire links on the top.

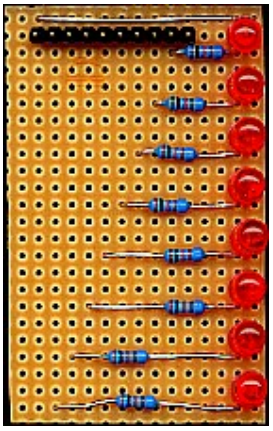
PIC Tutorial One - LED's

PIC Tutorial - LED Board

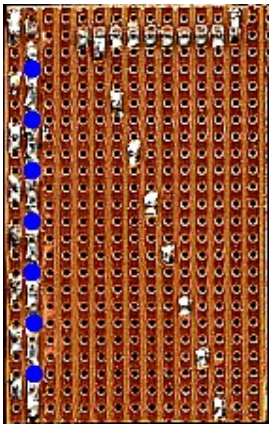
LED Board



This is the LED Board, a simple array of eight LED's connected to all pins of one port. As well as being used for simple output tutorials, it can be used as a debugging aid, by using it to display values at certain points in a program. Although it's labelled as connecting to PortB, it can also be connected to PortA if required.



This is the top side of the LED board, it's built on a piece of Veroboard 15 strips wide by 24 holes high, it consists simply of eight LED's, eight resistors, and the connector. This board has only one wire link, across the top side of the connector, to connect the ground connection of the LED's.



The bottom of the board, it has only seven track cuts, again they are marked with blue circles.

For the first parts of this tutorial you can use the Main Board LED, with jumper J1 set, or you can use the LED board on PortB, the later parts use more than one LED and the LED board will be required. Download zipped tutorial files.

Tutorial 1.1

This simple program repeatedly switches all the output pins high and low.

```
        ;Tutorial 1.1 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets the configuration settings
                                        ;(oscillator type etc.)

        org     0x0000                  ;org sets the origin, 0x0000 for the
16F628,
                                        ;this is where the program starts
running
        movlw   0x07
        movwf   CMCON                   ;turn comparators off (make it like a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   TRISB
        movwf   TRISA                   ;set PortA all outputs
        bcf     STATUS,         RP0     ;select bank 0

Loop
```

```
        movlw   0xff
        movwf   PORTA                   ;set all bits on
        movwf   PORTB
        nop                             ;the nop's make up the time taken by
the goto
        nop                             ;giving a square wave output
        movlw   0x00
        movwf   PORTA
        movwf   PORTB                   ;set all bits off
        goto    Loop                    ;go back and do it again

        end
```

The first three lines are instructions to the assembler, and not really part of the program at all, they should be left as they are during these tutorials - the __Config line sets the various configuration fuses in the chip, in this case it selects the internal 4MHz oscillator. The next line 'org 0x0000' sets the start address, it does vary across the PIC range, but most modern ones start from the lowest address - zero.

Lines 5 and 6 are specific to the 16F628, 'movlw 0x07' means 'MOVe the Literal value 7 into the W register', the W register is the main working register, 'movwf CMCON' means 'MOV the value in W to File CMCON', CMCON is a register in the 16F628 that is used to select the operation of the comparator hardware. So these two lines set CMCON to 7, this disables the comparator, and makes their I/O lines available for general use.

The next five lines set the direction of the I/O pins, first we have to select 'bank 1', some registers are in 'bank 0' and some in 'bank 1', to select 'bank 1' we need to set the bit RP0 in the STATUS register to '1' - the 'bsf' (Bit Set File) command sets a bit to one. The 'bcf' (Bit Clear File) at the end of these five lines, sets RP0 back to '0' and returns to 'bank 0'. The 'movlw', as before, moves a literal value into the W register, although this time the value passed is a binary value (instead of the hexadecimal 0x00), signified by the 'b' at the start of the value, in this case it's simply zero, and this value is then transferred to the two TRIS registers (TRIState) A and B. This sets the direction of the pins, a '0' sets a pin as an Output, and a '1' sets a pin as an Input - so b'00000000' (eight zeros) sets all the pins to outputs, b'10000000' would set I/O pin 7 as an input, and all the others as outputs - by using a binary value it's easy to see which pins are inputs (1) and which are outputs (0).

This completes the setting up of the chip, we can now start the actual 'running' part of the program, this begins with a label 'Loop', the last command 'goto Loop' returns the program to here, and it loops round for ever. The first instruction in this section 'movlw 0xff' moves the hexadecimal number 0xff (255 decimal, 11111111 binary) to the W register, the second and third then transfer this to the PortA and PortB I/O ports - this 'tries' to set all 16 pins high (I'll explain more later!). The next two instructions are 'nop' 'NO Operation', these simply take 1uS to execute, and do nothing, they are used to keep the outputs high for an extra 2uS. Following that we have a 'movlw 0x00' which moves 0x00 (0 decimal, 00000000 binary) to the W register, then we transfer them to the ports as before, this sets all 16 outputs low. The last 'goto Loop' instruction goes back and runs this section of the program again, and thus continues switching the port pins high then low.

Tutorial 1.2

As you will have noticed from the first part, the LED's don't flash!. This isn't strictly true, they do flash - but much too quickly to be visible. As the PIC runs at 4MHz each instruction only takes 1uS to complete (except for 'jump' instructions, which take 2uS), this causes the LED's to flash tens of thousands of times per second - much too quick for our eyes!. This is a common 'problem' with PIC programming, it runs far faster than the world we are used to, and often we need to slow things down!.

This second program also repeatedly switches all the output pins high and low, but this time introduces a time delay in between switching.

```
;Tutorial 1.2 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   TRISB
        movwf   TRISA                   ;set PortA all outputs
        bcf     STATUS,         RP0     ;select bank 0

Loop
        movlw   0xff
        movwf   PORTA                   ;set all bits on
        movwf   PORTB
        nop                             ;the  nop's  make  up  the  time  taken  by
the goto
        nop                             ;giving a square wave output
        call    Delay                   ;this waits for a while!
        movlw   0x00
        movwf   PORTA
        movwf   PORTB                   ;set all bits off
        call    Delay
        goto    Loop                    ;go back and do it again

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
```

```
goto    $+2
decfsz  countb, f
goto    Delay_0

decfsz  count1  ,f
goto    d1
retlw   0x00

end
```

This simply adds a couple of extra lines in the main program, 'call Delay', this is a call to a subroutine, a part of the program which executes and then returns to where it was called from. The routine is called twice, once after the LED's are turned on, and again after they are turned off. All the 'Delay' subroutine does is waste time, it loops round and round counting down until it finishes and returns. The extra part added at the beginning of the program (cblock to endc) allocates a couple of variables (count1 and count2) to two of the 'general purpose file registers', these start at address 0x20 - the cblock directive allocates the first variable to 0x20, and subsequent ones to sequential addresses.

The 'Delay' routine delays 250mS, set in it's first line (movlw d'250') - the 'd' signifies a decimal number, easier to understand in this case - so we turn on the LED's, wait 250mS, turn off the LED's, wait another 250mS, and then repeat. This makes the LED's flash 2 times per second, and is now clearly visible. By altering the value d'250' you can alter the flash rate, however as it's an eight bit value it can't go any higher than d'255' (0xff hexadecimal).

This routine introduces a new command 'decfsz' 'Decrement File and Skip on Zero', this decrements the file register specified (in this case either counta, countb, or count1) and if the result equals zero skips over the next line. So for an example using it,

```
decfsz  count1  ,f
goto    d1
```

this decrements count1 (storing the result back in count1, because of the ',f' - for 'file' at the end of the line), checks if it equals zero, and if not continues to the 'goto d1' line, which jumps back, runs the intervening code, and decrements count1 again, this continues until count1 equals zero, then the 'goto d1' is skipped over and the subroutine is exited at 'retlw 0x00'. The entire Delay routine is called a 'nested loop', the inner loop (using counta and countb) takes 1mS to run, and the outer loop calls the inner loop the number of times specified in count1 - so if you load 0x01 into count1 the entire Delay routine will take 1mS, in the example used we load d'250' (hexadecimal 0xfa) into count1, so it takes 250mS (1/4 of a second). The other new command introduced is 'retlw' 'RETurn from subroutine with Literal in W', this returns to where the subroutine was called from, and returns an optional value in the W register (it's not used to return a value here, so we assign 0x00 to it).

A line which might cause some confusion is the one 'goto $+2', this is basically an assembler instruction, the '$' represents the current program address, and the '+2' adds 2 to that address. So 'goto $+2' jumps to the line after the next line, in this case the line 'goto Delay_0', it simply saves giving the line it's own label. Another common use of this technique is 'goto $+1', which at first glance doesn't seem to do anything (as the program would continue to the next line anyway), but it's a longer replacement for 'nop' (NO oPeration) - this is often used to provide a small delay, but at 4MHz one 'nop' only provides a 1uS delay, a 'goto' instruction takes 2uS, so the single

word instruction 'goto $+1' can take the place of two 'nop' instructions, giving a 50% space saving.

I mentioned above that the routine (as written) can only delay a maximum of 255mS, if we wanted a longer delay we could introduce another outer loop which calls 'Delay' the required number of times, but if we just wanted to make it flash once per second (instead of twice) we could simply duplicate the 'call Delay' lines,

```
        nop                     ;giving a square wave output
        call    Delay           ;this waits for a while!
        call    Delay           ;second delay call added
        movlw   0x00
```

this gives a 500mS delay, leaving the LED on for 1/2 a second, by adding a second 'call Delay' to the 'off time' the LED will stay off for 1/2 a second as well. There's no requirement to keep these symmetrical, by using one 'call Delay' for the 'on time', and three for the 'off time' the LED will still flash once per second, but only stay on for 1/4 of the time (25/75) - this will only use 50% of the power that a 50/50 flash would consume.

There are huge advantages in using subroutines, a common routine like this may be required many times throughout the program, by storing it once as a subroutine we save lots of space. Also, if you need to alter the routine for any reason, you only need to alter it in one place, and the change will affect all the calls to it. As your PIC programming skills develop you will find you create a library of useful little routines, these can be 'stitched together' to create larger programs - you'll see the 'Delay' subroutine appearing quite a lot in later tutorials, and other subroutines will also make many appearances - why keep reinventing the wheel?.

Tutorial 1.3

The previous two examples simply turn all pins high or low, often we only want to affect a single pin, this is easily achieved with the 'bcf' and 'bsf' commands, 'bcf' 'Bit Clear File' clears a bit (sets it to 0), and 'bsf' 'Bit Set File' sets a bit (sets it to 1), the bit number ranges from 0 (LSB) to 7 (MSB). The following example flashes PortB, bit 7 (RB7) only, the rest of the pins remain at 0.

```
;Tutorial 1.3 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        org     0x0000                  ;org sets the origin, 0x0000 for the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn comparators off (make it like a
16F84)
```

```
        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   TRISB
        movwf   TRISA                   ;set PortA all outputs
        bcf     STATUS,         RP0     ;select bank 0
        clrf    PORTA
        clrf    PORTB                   ;set all outputs low

Loop
        bsf     PORTB,  7               ;turn on RB7 only!
        call    Delay                   ;this waits for a while!
        bcf     PORTB,  7               ;turn off RB7 only!.
        call    Delay
        goto    Loop                    ;go back and do it again

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7                    ;delay 1mS
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

The 'movwf PORTA' and 'movwf PORTB' lines have been replaced by the single line 'bsf PORTB, 7' (to turn the LED on), and 'bcf PORTB, 7' (to turn the LED off). The associated 'movlw 0xff' and 'movlw 0x00' have also been removed, as they are no longer required, the two 'nop' commands have also been removed, they are pretty superfluous - it's not worth adding 2uS to a routine that lasts 250mS!.

Tutorial 1.4

If you want to use a different pin to RB7, you could simply alter the '7' on the relevant two lines to whichever pin you wanted, or if you wanted to use PortA, alter the PortB to PortA - however, this requires changing two lines (and could be many more in a long program). So there's a better way! - this example (functionally identical to the previous one) assigns two constants at the beginning of the program, LED, and LEDPORT - these are assigned the values '7' and 'PORTB' respectively, and these constant names are used in the 'bsf' and 'bcf' lines. When the assembler is run it replaces all occurrences of the constant names with their values. By doing this is makes it MUCH! easier to change pin assignments, and it will be used more and more in the following tutorials. In fact, if you look at the 'P16F628.INC' which sets the defaults for the chip, this is simply a list of similar assignments which take a name and replace it with a number (PORTB is actually 0x06).

```
;Tutorial 1.4 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
```

```
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        LED     Equ     7               ;set constant LED = 7
        LEDPORT Equ     PORTB           ;set constant LEDPORT = 'PORTB'

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this   is   where   the   program   starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   TRISB
        movwf   TRISA                   ;set PortA all outputs
        bcf     STATUS,         RP0     ;select bank 0
        clrf    PORTA
        clrf    PORTB                   ;set all outputs low

Loop
        bsf     LEDPORT,        LED     ;turn on RB7 only!
        call    Delay                   ;this waits for a while!
        bcf     LEDPORT,        LED     ;turn off RB7 only!.
        call    Delay
        goto    Loop                    ;go back and do it again

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7                    ;delay 1mS
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

This works exactly the same as the previous version, and if you compare the '.hex' files produced you will see that they are identical.

Suggested exercises:

- Alter the number of Delay calls, as suggested above, to produce asymmetrical flashing, both short flashes and long flashes.

- Change the pin assignments to use pins other than RB7 (requires LED board).
- Flash more than one (but less than 8) LED's at the same time - TIP: add extra 'bsf' and 'bcf' lines.
- Introduce extra flashing LED's, using different flashing rates -TIP: flash one on/off, then a different one on/off, adding different numbers of calls to Delay in order to have different flashing rates. If required change the value (d'250') used in the Delay subroutine.

Tutorials below here require the LED board

Tutorial 1.5

This uses the LED board, and runs a single LED across the row of eight.

```
;Tutorial 1.5 - Nigel Goodwin 2002
       LIST    p=16F628              ;tell assembler what chip we are using
       include "P16F628.inc"         ;include the defaults for the chip
       __config 0x3D18               ;sets   the   configuration   settings
(oscillator type etc.)

       cblock  0x20                  ;start of general purpose registers
               count1                ;used in delay routine
               counta                ;used in delay routine
               countb                ;used in delay routine
       endc

       LEDPORT Equ     PORTB         ;set constant LEDPORT = 'PORTB'
       LEDTRIS Equ     TRISB         ;set constant for TRIS register

       org     0x0000                ;org  sets  the  origin,  0x0000  for  the
16F628,
                                     ;this  is  where  the  program  starts
running
       movlw   0x07
       movwf   CMCON                 ;turn  comparators  off  (make  it  like  a
16F84)

       bsf     STATUS,         RP0   ;select bank 1
       movlw   b'00000000'           ;set PortB all outputs
       movwf   LEDTRIS
       bcf     STATUS,         RP0   ;select bank 0
       clrf    LEDPORT               ;set all outputs low

Loop
       movlw   b'10000000'
       movwf   LEDPORT
       call    Delay                 ;this waits for a while!
       movlw   b'01000000'
       movwf   LEDPORT
       call    Delay                 ;this waits for a while!
       movlw   b'00100000'
       movwf   LEDPORT
       call    Delay                 ;this waits for a while!
       movlw   b'00010000'
       movwf   LEDPORT
       call    Delay                 ;this waits for a while!
       movlw   b'00001000'
```

```
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000100'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000010'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000001'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        goto    Loop                    ;go back and do it again

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

Tutorial 1.6

We can very easily modify this routine so the LED bounces from end to end, just add some more 'movlw' and 'movwf' with the relevant patterns in them - plus the 'call Delay' lines.

```
;Tutorial 1.6 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        LEDPORT Equ     PORTB           ;set constant LEDPORT = 'PORTB'
        LEDTRIS Equ     TRISB           ;set constant for TRIS register

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
```

```
        movlw   b'00000000'             ;set PortB all outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low

Loop
        movlw   b'10000000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'01000000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00100000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00010000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00001000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000100'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000010'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000001'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000010'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00000100'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00001000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00010000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'00100000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        movlw   b'01000000'
        movwf   LEDPORT
        call    Delay                   ;this waits for a while!
        goto    Loop                    ;go back and do it again

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0
```

```
        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

Tutorial 1.7

Now while the previous two routines work perfectly well, and if this was all you wanted to do would be quite satisfactory, they are rather lengthy, crude and inelegant!. Tutorial 1.5 uses 24 lines within the loop, by introducing another PIC command we can make this smaller, and much more elegant.

```
;Tutorial 1.7 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        LEDPORT Equ     PORTB           ;set constant LEDPORT = 'PORTB'
        LEDTRIS Equ     TRISB           ;set constant for TRIS register

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low

Start   movlw   b'10000000'             ;set first LED lit
        movwf   LEDPORT

Loop    bcf     STATUS, C               ;clear carry bit
        call    Delay                   ;this waits for a while!
        rrf     LEDPORT,        f
        btfss   STATUS, C               ;check  if  last  bit  (1  rotated  into
Carry)
        goto    Loop                    ;go back and do it again
        goto    Start

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
```

```
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

This introduces the 'rrf' 'Rotate Right File' command, this rotates the contents of the file register to the right (effectively dividing it by two). As the 'rrf' command rotates through the 'carry' bit we need to make sure that is cleared, we do this with the 'bcf STATUS, C' line. To check when we reach the end we use the line 'btfss PORTB, 0' to check when bit 0 is high, this then restarts the bit sequence at bit 7 again.

Tutorial 1.8

We can apply this to tutorial 1.6 as well, this time adding the 'rlf' 'Rotate Left File' command, this shifts the contents of the register to the left (effectively multiplying by two).

```
;Tutorial 1.8 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        LEDPORT Equ     PORTB           ;set constant LEDPORT = 'PORTB'
        LEDTRIS Equ     TRISB           ;set constant for TRIS register

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low

Start   movlw   b'10000000'             ;set first LED lit
        movwf   LEDPORT

Loop    bcf     STATUS, C               ;clear carry bit
        call    Delay                   ;this waits for a while!
        rrf     LEDPORT,        f
        btfss   STATUS, C               ;check  if  last  bit  (1  rotated  into
Carry)
```

```
        goto    Loop                    ;go back and do it again
        movlw   b'00000001'             ;set last LED lit
        movwf   LEDPORT

Loop2   bcf     STATUS, C               ;clear carry bit
        call    Delay                   ;this waits for a while!
        rlf     LEDPORT,        f
        btfss   STATUS, C               ;check  if  last  bit  (1  rotated  into
Carry)
        goto    Loop2                   ;go back and do it again
        goto    Start                   ;finished, back to first loop

Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

Tutorial 1.9

So far we have used two different methods to produce a 'bouncing' LED, here's yet another version, this time using a data lookup table.

```
;Tutorial 1.9 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count                   ;used in table read routine
                count1                  ;used in delay routine
                counta                  ;used in delay routine
                countb                  ;used in delay routine
        endc

        LEDPORT Equ     PORTB           ;set constant LEDPORT = 'PORTB'
        LEDTRIS Equ     TRISB           ;set constant for TRIS register

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this   is   where   the   program   starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
```

```
        movlw   b'00000000'             ;set PortB all outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low


Start   clrf    count                   ;set counter register to zero
Read    movf    count, w                ;put counter value in W
        call    Table
        movwf   LEDPORT
        call    Delay
        incf    count,  w
        xorlw   d'14'                   ;check for last (14th) entry
        btfsc   STATUS, Z
        goto    Start                   ;if start from beginning
        incf    count,  f               ;else do next
        goto    Read

Table   ADDWF    PCL, f                 ;data table for bit pattern
        retlw   b'10000000'
         retlw   b'01000000'
         retlw   b'00100000'
         retlw   b'00010000'
         retlw   b'00001000'
         retlw   b'00000100'
         retlw   b'00000010'
         retlw   b'00000001'
         retlw   b'00000010'
         retlw   b'00000100'
         retlw   b'00001000'
         retlw   b'00010000'
         retlw   b'00100000'
         retlw   b'01000000'


Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

This version introduces another new command 'addwf' 'ADD W to File', this is the command used to do adding on the PIC, now with the PIC only having 35 commands in it's RISC architecture some usual commands (like easy ways of reading data from a table) are absent, but there's always a way to do things. Back in tutorial 1.2 we introduced the 'retlw' command, and mentioned that we were not using the returned value, to make a data table we make use of this returned value. At the Label 'Start' we first zero a counter we are going to use 'clrf Count', this is then moved to the W register 'movf Count, w', and the Table subroutine called. The first line in

the subroutine is 'addwf PCL, f', this adds the contents of W (which has just been set to zero) to the PCL register, the PCL register is the Program Counter, it keeps count of where the program is, so adding zero to it moves to the next program line 'retlw b'10000000' which returns with b'10000000' in the W register, this is then moved to the LED's. Count is then incremented and the program loops back to call the table again, this time W holds 1, and this is added to the PCL register which jumps forward one more and returns the next entry in the table - this continues for the rest of the table entries. It should be pretty obvious that it wouldn't be a good idea to overrun the length of the table, it would cause the program counter to jump to the line after the table, which in this case is the Delay subroutine, this would introduce an extra delay, and return zero, putting all the LED's out. To avoid this we test the value of count, using 'xorlw d14' 'eXclusive Or Literal with W', this carries out an 'exclusive or' with the W register (to which we've just transferred the value of count) and the value 14, and sets the 'Z' 'Zero' flag if they are equal, we then test the Z flag, and if it's set jump back to the beginning and reset count to zero again.

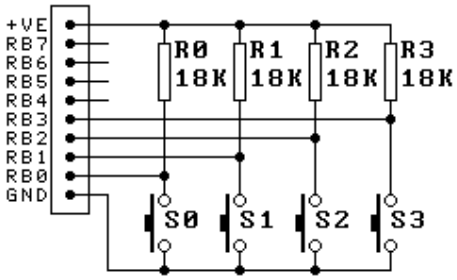
The big advantage of this version is that we can have any pattern we like in the table, it doesn't have to be just one LED, and we can easily alter the size of the table to add or remove entries, remembering to adjust the value used to check for the end of the table!. A common technique in this tables is to add an extra entry at the end, usually zero, and check for that rather than maintain a separate count, I haven't done this because doing it via a counter allows you to have a zero entry in the table - although this example doesn't do that you may wish to alter the patterns, and it could make a simple light sequencer for disco lights, and you may need a zero entry.

Suggested exercises:

- Using the table technique of 1.9, alter the pattern to provide a moving dark LED, with all others lit.
- Alter the length of the table, and add your own patterns, for a suggestion try starting with LED's at each end lit, and move them both to the middle and back to the outside.

PIC Tutorial Two – Switches

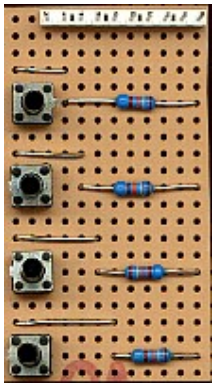
I2C EEPROM Board



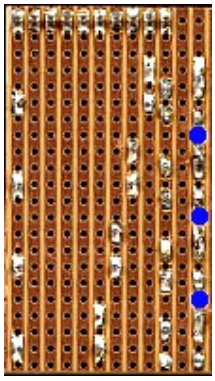
This is the I2C Switch Board, although it's not really an I2C based board, I've named it as such because it's intended to provide some input buttons for the other I2C boards. As those boards use RB6 and RB7 for the I2C bus we can't use the original switch board, however this board only has switches (no LED's like the original switch board), and they connect to the bottom four port pins. It plugs in to the second connector for PortB, the first being used for the relevant I2C board.

The first use of this board is to provide adjustment buttons for the I2C Clock Board, enabling easy adjustment of the time and date. It will also be used with the I2C A2D board, to allow various adjustments and settings to made, such as 'set the sample time', 'dump data to PC' etc. I think most projects can benefit from the availability of a few push buttons to control their action.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.



This is the top view of the I2C Switch Board, it has only 4 wire links.



The bottom of the I2C Switch Board, there are 3 track cuts.

For the first parts of this tutorial you require the Main Board and the Switch Board, the later parts will also use the LED Board, as written the tutorials use the Switch Board on PortA and the LED Board on PortB. Download zipped tutorial files.

Tutorial 2.1 - requires Main Board and Switch Board.

This simple program turns the corresponding LED on, when the button opposite it is pressed, extinguishing all other LED's.

```
;Tutorial 2.1 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)


LEDPORT Equ     PORTA                   ;set constant LEDPORT = 'PORTA'
SWPORT  Equ     PORTA                   ;set constant SWPORT = 'PORTA'
LEDTRIS Equ     TRISA                   ;set constant for TRIS register
SW1     Equ     7                       ;set constants for the switches
SW2     Equ     6
SW3     Equ     5
SW4     Equ     4
LED1    Equ     3                       ;and for the LED's
LED2    Equ     2
LED3    Equ     1
LED4    Equ     0

;end of defines

        org     0x0000                  ;org  sets  the  origin,  0x0000  for  the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'11110000'             ;set PortA 4 inputs, 4 outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low
```

```
Loop    btfss   SWPORT, SW1
        call    Switch1
        btfss   SWPORT, SW2
        call    Switch2
        btfss   SWPORT, SW3
        call    Switch3
        btfss   SWPORT, SW4
        call    Switch4
        goto    Loop

Switch1 clrf    LEDPORT                 ;turn all LED's off
        bsf     SWPORT, LED1            ;turn LED1 on
        retlw   0x00

Switch2 clrf    LEDPORT                 ;turn all LED's off
        bsf     SWPORT, LED2            ;turn LED2 on
        retlw   0x00

Switch3 clrf    LEDPORT                 ;turn all LED's off
        bsf     SWPORT, LED3            ;turn LED3 on
        retlw   0x00

Switch4 clrf    LEDPORT                 ;turn all LED's off
        bsf     SWPORT, LED4            ;turn LED4 on
        retlw   0x00


        end
```

As with the previous tutorials we first set things up, then the main program runs in a loop, the first thing the loop does is check switch SW1 with the 'btfss SWPORT, SW1' line, if the switch isn't pressed the input line is held high by the 10K pull-up resistor and it skips the next line. This takes it to the 'btfss SWPORT, SW2' line, where SW2 is similarly checked - this continues down checking all the switches and then loops back and checks them again. If a key is pressed, the relevant 'btfss' doesn't skip the next line, but instead calls a sub-routine to process the key press, each switch has it's own sub-routine. These sub-routines are very simple, they first 'clrf' the output port, turning all LED's off, and then use 'bsf' to turn on the corresponding LED, next the sub-routine exits via the 'retlw' instruction. As the switch is likely to be still held down, the same routine will be run again (and again, and again!) until you release the key, however for this simple application that isn't a problem and you can't even tell it's happening.

Tutorial 2.2 - requires Main Board and Switch Board.

This program toggles the corresponding LED on and off, when the button opposite it is pressed. It introduces the concept of 'de-bouncing' - a switch doesn't close immediately, it 'bounces' a little before it settles, this causes a series of fast keypresses which can cause chaos in operating a device.

```
;Tutorial 2.2 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count1                  ;used in delay routine
                counta                  ;used in delay routine
```

```
                countb                  ;used in delay routine
        endc

LEDPORT Equ     PORTA                   ;set constant LEDPORT = 'PORTA'
SWPORT  Equ     PORTA                   ;set constant SWPORT = 'PORTA'
LEDTRIS Equ     TRISA                   ;set constant for TRIS register
SW1     Equ     7                       ;set constants for the switches
SW2     Equ     6
SW3     Equ     5
SW4     Equ     4
LED1    Equ     3                       ;and for the LED's
LED2    Equ     2
LED3    Equ     1
LED4    Equ     0

SWDel   Set     Del50                   ;set the de-bounce delay (has to use
'Set' and not 'Equ')

;end of defines

        org     0x0000                  ;org sets the origin, 0x0000 for the
16F628,
                                        ;this is where the program starts
running
        movlw   0x07
        movwf   CMCON                   ;turn comparators off (make it like a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'11110000'             ;set PortA 4 inputs, 4 outputs
        movwf   LEDTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low


Loop    btfss   SWPORT, SW1
        call    Switch1
        btfss   SWPORT, SW2
        call    Switch2
        btfss   SWPORT, SW3
        call    Switch3
        btfss   SWPORT, SW4
        call    Switch4
        goto    Loop

Switch1 call    SWDel                   ;give switch time to stop bouncing
        btfsc   SWPORT, SW1             ;check it's still pressed
        retlw   0x00                    ;return is not
        btfss   SWPORT, LED1            ;see if LED1 is already lit
        goto    LED1ON
        goto    LED1OFF

LED1ON  bsf     LEDPORT,        LED1    ;turn LED1 on
        call    SWDel
        btfsc   SWPORT, SW1             ;wait until button is released
        retlw   0x00
        goto    LED1ON

LED1OFF bcf     LEDPORT,        LED1    ;turn LED1 on
        call    SWDel
        btfsc   SWPORT, SW1             ;wait until button is released
        retlw   0x00
```

```
        goto    LED1OFF

Switch2 call    SWDel                   ;give switch time to stop bouncing
        btfsc   SWPORT, SW2             ;check it's still pressed
        retlw   0x00                    ;return is not
        btfss   SWPORT, LED2            ;see if LED2 is already lit
        goto    LED2ON
        goto    LED2OFF

LED2ON  bsf     LEDPORT,        LED2    ;turn LED2 on
        call    SWDel
        btfsc   SWPORT, SW2             ;wait until button is released
        retlw   0x00
        goto    LED2ON

LED2OFF bcf     LEDPORT,        LED2    ;turn LED2 on
        call    SWDel
        btfsc   SWPORT, SW2             ;wait until button is released
        retlw   0x00
        goto    LED2OFF

Switch3 call    SWDel                   ;give switch time to stop bouncing
        btfsc   SWPORT, SW3             ;check it's still pressed
        retlw   0x00                    ;return is not
        btfss   SWPORT, LED3            ;see if LED3 is already lit
        goto    LED3ON
        goto    LED3OFF

LED3ON  bsf     LEDPORT,        LED3    ;turn LED3 on
        call    SWDel
        btfsc   SWPORT, SW3             ;wait until button is released
        retlw   0x00
        goto    LED3ON

LED3OFF bcf     LEDPORT,        LED3    ;turn LED3 on
        call    SWDel
        btfsc   SWPORT, SW3             ;wait until button is released
        retlw   0x00
        goto    LED3OFF

Switch4 call    SWDel                   ;give switch time to stop bouncing
        btfsc   SWPORT, SW4             ;check it's still pressed
        retlw   0x00                    ;return is not
        btfss   SWPORT, LED4            ;see if LED4 is already lit
        goto    LED4ON
        goto    LED4OFF

LED4ON  bsf     LEDPORT,        LED4    ;turn LED4 on
        call    SWDel
        btfsc   SWPORT, SW4             ;wait until button is released
        retlw   0x00
        goto    LED4ON

LED4OFF bcf     LEDPORT,        LED4    ;turn LED4 on
        call    SWDel
        btfsc   SWPORT, SW4             ;wait until button is released
        retlw   0x00
        goto    LED4OFF

;modified Delay routine, direct calls for specified times
;or load W and call Delay for a custom time.
```

```
Del0    retlw   0x00                    ;delay 0mS - return immediately
Del1    movlw   d'1'                    ;delay 1mS
        goto    Delay
Del5    movlw   d'5'                    ;delay 5mS
        goto    Delay
Del10   movlw   d'10'                   ;delay 10mS
        goto    Delay
Del20   movlw   d'20'                   ;delay 20mS
        goto    Delay
Del50   movlw   d'50'                   ;delay 50mS
        goto    Delay
Del100  movlw   d'100'                  ;delay 100mS
        goto    Delay
Del250  movlw   d'250'                  ;delay 250 ms
Delay   movwf   count1
d1      movlw   0xC7                    ;delay 1mS
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00
        end
```

In order to de-bounce the keypresses we delay for a short time, then check that the button is still pressed, the time delay is set by the variable SWDel, which is defined as Del50 in the defines section at the start of the program. I've extended the Delay routine to provide a selection of different delays (from 0mS to 250mS), called by simple 'call' instructions, the Delay routine itself can also be called directly - simply load the required delay into the W register and 'call Delay'. We then check to see if the corresponding LED is lit, with 'btfss SWPORT, LEDx', and jump to either 'LEDxON' or 'LEDxOFF', these routines are almost identical, the only difference being that the first turns the LED on, and the second turns it off. They first switch the LED, on or off, depending on which routine it is, and then delay again (calling SWDel as before), next they check to see if the button is still pressed, looping back around if it is. Once the key has been released the routine exits via the usual 'retlw' and returns to waiting for a keypress. I've used the variable SWDel (and provided the various delay times) so that you can easily try the effect of different delay times - in particular try setting SWDel to Del0, and see how the button pressing isn't reliable, you will probably find one of the buttons is worse than the others - particularly if you use old switches, wear makes them bounce more.

Tutorial 2.3 - requires Main Board, Switch Board, and LED Board.

Now for a more realistic example - this combines Tutorial 2.1 with Tutorial 1.9, the result is an LED sequencing program with four different patterns, selected by the four keys, with the key selected indicated by the corresponding LED.

```
;Tutorial 2.3 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
```

```
        __config 0x3D18             ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                ;start of general purpose registers
                count               ;used in table read routine
                count1              ;used in delay routine
                counta              ;used in delay routine
                countb              ;used in delay routine
        endc

LEDPORT Equ     PORTB               ;set constant LEDPORT = 'PORTB'
LEDTRIS Equ     TRISB               ;set constant for TRIS register
SWPORT  Equ     PORTA
SWTRIS  Equ     TRISA
SW1     Equ     7                   ;set constants for the switches
SW2     Equ     6
SW3     Equ     5
SW4     Equ     4
LED1    Equ     3                   ;and for the LED's
LED2    Equ     2
LED3    Equ     1
LED4    Equ     0

        org     0x0000              ;org  sets  the  origin,  0x0000  for  the
16F628,
                                    ;this   is   where   the   program   starts
running
        movlw   0x07
        movwf   CMCON               ;turn  comparators  off  (make  it  like  a
16F84)

        bsf     STATUS,         RP0 ;select bank 1
        movlw   b'00000000'         ;set PortB all outputs
        movwf   LEDTRIS
        movlw   b'11110000'         ;set PortA 4 inputs, 4 outputs
        movwf   SWTRIS
        bcf     STATUS,         RP0 ;select bank 0
        clrf    LEDPORT             ;set all outputs low
        clrf    SWPORT
        bsf     SWPORT, LED1        ;set initial pattern


Start   clrf    count               ;set counter register to zero
Read    movf    count, w            ;put counter value in W
        btfsc   SWPORT, LED1        ;check which LED is lit
        call    Table1              ;and read the associated table
        btfsc   SWPORT, LED2
        call    Table2
        btfsc   SWPORT, LED3
        call    Table3
        btfsc   SWPORT, LED4
        call    Table4
        movwf   LEDPORT
        call    Delay
        incf    count,  w
        xorlw   d'14'               ;check for last (14th) entry
        btfsc   STATUS, Z
        goto    Start               ;if start from beginning
        incf    count,  f           ;else do next
        goto    Read

Table1  ADDWF   PCL, f              ;data table for bit pattern
```

```
        retlw   b'10000000'
        retlw   b'01000000'
        retlw   b'00100000'
        retlw   b'00010000'
        retlw   b'00001000'
        retlw   b'00000100'
        retlw   b'00000010'
        retlw   b'00000001'
        retlw   b'00000010'
        retlw   b'00000100'
        retlw   b'00001000'
        retlw   b'00010000'
        retlw   b'00100000'
        retlw   b'01000000'

Table2  ADDWF   PCL, f               ;data table for bit pattern
        retlw   b'11000000'
        retlw   b'01100000'
        retlw   b'00110000'
        retlw   b'00011000'
        retlw   b'00001100'
        retlw   b'00000110'
        retlw   b'00000011'
        retlw   b'00000011'
        retlw   b'00000110'
        retlw   b'00001100'
        retlw   b'00011000'
        retlw   b'00110000'
        retlw   b'01100000'
        retlw   b'11000000'

Table3  ADDWF   PCL, f               ;data table for bit pattern
        retlw   b'01111111'
        retlw   b'10111111'
        retlw   b'11011111'
        retlw   b'11101111'
        retlw   b'11110111'
        retlw   b'11111011'
        retlw   b'11111101'
        retlw   b'11111110'
        retlw   b'11111101'
        retlw   b'11111011'
        retlw   b'11110111'
        retlw   b'11101111'
        retlw   b'11011111'
        retlw   b'10111111'

Table4  ADDWF   PCL, f               ;data table for bit pattern
        retlw   b'00111111'
        retlw   b'10011111'
        retlw   b'11001111'
        retlw   b'11100111'
        retlw   b'11110011'
        retlw   b'11111001'
        retlw   b'11111100'
        retlw   b'11111100'
        retlw   b'11111001'
        retlw   b'11110011'
        retlw   b'11100111'
        retlw   b'11001111'
        retlw   b'10011111'
        retlw   b'00111111'
```

```
ChkKeys btfss   SWPORT, SW1
        call    Switch1
        btfss   SWPORT, SW2
        call    Switch2
        btfss   SWPORT, SW3
        call    Switch3
        btfss   SWPORT, SW4
        call    Switch4
        retlw   0x00

Switch1 clrf    SWPORT                  ;turn all LED's off
        bsf     SWPORT, LED1            ;turn LED1 on
        retlw   0x00

Switch2 clrf    SWPORT                  ;turn all LED's off
        bsf     SWPORT, LED2            ;turn LED2 on
        retlw   0x00

Switch3 clrf    SWPORT                  ;turn all LED's off
        bsf     SWPORT, LED3            ;turn LED3 on
        retlw   0x00

Switch4 clrf    SWPORT                  ;turn all LED's off
        bsf     SWPORT, LED4            ;turn LED4 on
        retlw   0x00


Delay   movlw   d'250'                  ;delay 250 ms (4 MHz clock)
        movwf   count1
d1      movlw   0xC7                    ;delay 1mS
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

        end
```

The main differences here are in the Delay routine, which now has a call to check the keys every milli-second, and the main loop, where it selects one of four tables to read, depending on the settings of flag bits which are set according to which key was last pressed.

Tutorial 2.4 - requires Main Board, Switch Board, and LED Board.

Very similar to the last tutorial, except this one combines Tutorials 2.2 and 2.3 with Tutorial 1.9, the result is an LED sequencing program with three different patterns, selected by three of the keys, with the key selected indicated by the corresponding LED - the difference comes with the fourth switch, this selects slow or fast speeds, with the fast speed being indicated by a toggled LED.

```
;Tutorial 2.4 - Nigel Goodwin 2002
        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)

        cblock  0x20                    ;start of general purpose registers
                count                   ;used in table read routine
                count1                  ;used in delay routine
                count2                  ;used in delay routine
                counta                  ;used in delay routine
                countb
                countc
                countd
                speed
        endc

LEDPORT Equ     PORTB                   ;set constant LEDPORT = 'PORTB'
LEDTRIS Equ     TRISB                   ;set constant for TRIS register
SWPORT  Equ     PORTA
SWTRIS  Equ     TRISA
SW1     Equ     7                       ;set constants for the switches
SW2     Equ     6
SW3     Equ     5
SW4     Equ     4
LED1    Equ     3                       ;and for the LED's
LED2    Equ     2
LED3    Equ     1
LED4    Equ     0


SWDel   Set     Del50

        org     0x0000                  ;org sets the origin, 0x0000 for the
16F628,
                                        ;this  is  where  the  program  starts
running
        movlw   0x07
        movwf   CMCON                   ;turn comparators off (make it like a
16F84)

        bsf     STATUS,         RP0     ;select bank 1
        movlw   b'00000000'             ;set PortB all outputs
        movwf   LEDTRIS
        movlw   b'11110000'             ;set PortA 4 inputs, 4 outputs
        movwf   SWTRIS
        bcf     STATUS,         RP0     ;select bank 0
        clrf    LEDPORT                 ;set all outputs low
        clrf    SWPORT                  ;make sure all LED's are off
        bsf     SWPORT, LED1            ;and turn initial LED on
        movlw   d'250'
        movwf   speed                   ;set initial speed



Start   clrf    count                   ;set counter register to zero
Read    movf    count, w                ;put counter value in W
        btfsc   SWPORT, LED1            ;check which LED is on
        call    Table1                  ;and call the associated table
        btfsc   SWPORT, LED2
        call    Table2
        btfsc   SWPORT, LED3
```

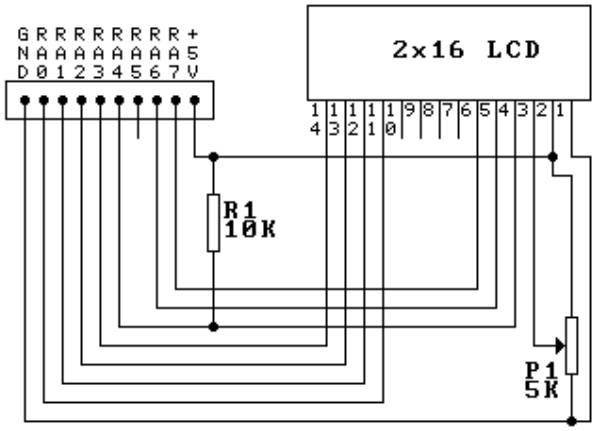
```
        call    Table3
        movwf   LEDPORT
        call    DelVar
        incf    count,  w
        xorlw   d'14'                   ;check for last (14th) entry
        btfsc   STATUS, Z
        goto    Start                   ;if start from beginning
        incf    count,  f               ;else do next
        goto    Read

Table1  ADDWF   PCL, f                  ;data table for bit pattern
        retlw   b'10000000'
        retlw   b'01000000'
        retlw   b'00100000'
        retlw   b'00010000'
        retlw   b'00001000'
        retlw   b'00000100'
        retlw   b'00000010'
        retlw   b'00000001'
        retlw   b'00000010'
        retlw   b'00000100'
        retlw   b'00001000'
        retlw   b'00010000'
        retlw   b'00100000'
        retlw   b'01000000'

Table2  ADDWF   PCL, f                  ;data table for bit pattern
        retlw   b'11000000'
        retlw   b'01100000'
        retlw   b'00110000'
        retlw   b'00011000'
        retlw   b'00001100'
        retlw   b'00000110'
        retlw   b'00000011'
        retlw   b'00000011'
        retlw   b'00000110'
        retlw   b'00001100'
        retlw   b'00011000'
        retlw   b'00110000'
        retlw   b'01100000'
        retlw   b'11000000'

Table3  ADDWF   PCL, f                  ;data table for bit pattern
        retlw   b'01111111'
        retlw   b'10111111'
        retlw   b'11011111'
        retlw   b'11101111'
        retlw   b'11110111'
        retlw   b'11111011'
        retlw   b'11111101'
        retlw   b'11111110'
        retlw   b'11111101'
        retlw   b'11111011'
        retlw   b'11110111'
        retlw   b'11101111'
        retlw   b'11011111'
        retlw   b'10111111'

ChkKeys btfss   SWPORT, SW1
        call    Switch1
        btfss   SWPORT, SW2
        call    Switch2
```

```
        btfss   SWPORT, SW3
        call    Switch3
        btfss   SWPORT, SW4
        call    Switch4
        retlw   0x00

Switch1 bcf     SWPORT, LED2            ;turn unselected LED's off
        bcf     SWPORT, LED3            ;turn unselected LED's off
        bsf     SWPORT, LED1            ;turn LED1 on
        retlw   0x00

Switch2 bcf     SWPORT, LED1            ;turn unselected LED's off
        bcf     SWPORT, LED3            ;turn unselected LED's off
        bsf     SWPORT, LED2            ;turn LED2 on
        retlw   0x00

Switch3 bcf     SWPORT, LED1            ;turn unselected LED's off
        bcf     SWPORT, LED2            ;turn unselected LED's off
        bsf     SWPORT, LED3            ;turn LED3 on
        retlw   0x00

Switch4 call    SWDel                   ;give switch time to stop bouncing
        btfsc   SWPORT, SW4             ;check it's still pressed
        retlw   0x00                    ;return is not
        btfss   SWPORT, LED4            ;see if LED4 is already lit
        goto    FASTON
        goto    FASTOFF

FASTON  bsf     SWPORT, LED4            ;turn LED4 on
        movlw   d'80'
        movwf   speed                   ;set fast speed
        call    SWDel
        btfsc   SWPORT, SW4             ;wait until button is released
        retlw   0x00
        goto    FASTON

FASTOFF bcf     SWPORT, LED4            ;turn LED4 on
        movlw   d'250'
        movwf   speed                   ;set slow speed
        call    SWDel
        btfsc   SWPORT, SW4             ;wait until button is released
        retlw   0x00
        goto    FASTOFF

DelVar  movfw   speed                   ;delay set by Speed
        movwf   count1
d1      movlw   0xC7                    ;delay 1mS
        movwf   counta
        movlw   0x01
        movwf   countb
Delay_0
        decfsz  counta, f
        goto    $+2
        decfsz  countb, f
        goto    Delay_0

        decfsz  count1  ,f
        goto    d1
        retlw   0x00

;use separate delay routines, as Del50 is called from ChkKeys
;which is called from within DelVar
```

```
Del50   movlw   d'50'                  ;delay 50mS
        movwf   count2
d3      movlw   0xC7                   ;delay 1mS
        movwf   countc
        movlw   0x01
        movwf   countd
Delay_1
        decfsz  countc, f
        goto    $+2
        decfsz  countd, f
        goto    Delay_1

        decfsz  count2  ,f
        goto    d3
        retlw   0x00


      end
```

PIC Tutorial Three - LCD Modules

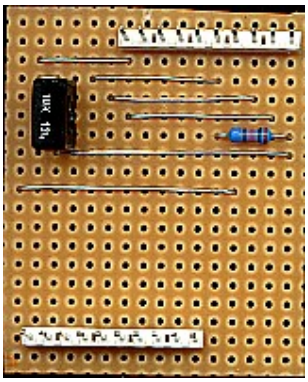
LCD Board



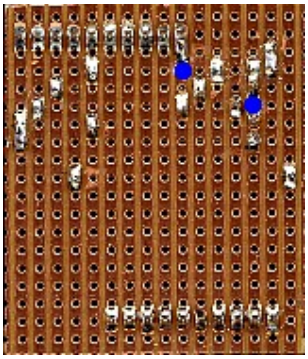
LCD Pin Functions		
Pin	**Function**	**Description**
1	Vss	Ground
2	Vdd	+ve supply
3	Vee	Contrast
4	RS	Register Select
5	R/W	Read/Write
6	E	Enable
7	D0	Data bit 0 (8 bit)
8	D1	Data bit 1 (8 bit)
9	D2	Data bit 2 (8 bit)
10	D3	Data bit 3 (8 bit)
11	D4	Data bit 4
12	D5	Data bit 5
13	D6	Data bit 6
14	D7	Data bit 7

This is the LCD Board, using an LCD module based on the industry standard Hitachi HD44780, it connects to 7 pins of one port, and operates in 4 bit 'nibble' mode to save I/O pins. By connecting to PortA we have to use a pull-up resistor (R1) on RA4, and are unable to use RA5 (which is only an input), however this frees all of PortB which will allow us to use some of the extra hardware available on PortB, along with the LCD, in a later tutorial. The potentiometer

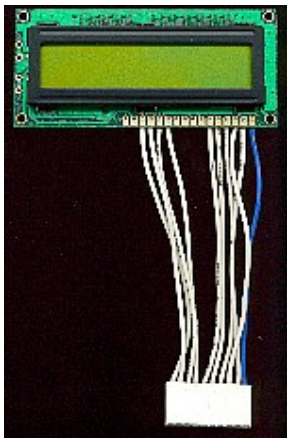
P1, is for adjusting the contrast of the display, and if incorrectly adjusted can cause the display to be invisible. Although it's labelled as connecting to PortA, as with most of the boards, it can also be connected to PortB if required. By using 4 bit mode we can connect the entire LCD module to one port, it uses exactly 10 pins (just right for our Molex connectors). In 4 bit mode we don't use pins 7-10, which are used as the lower 4 data bits in 8 bit mode, instead we write (or read) to the upper 4 pins twice, transferring half of the data each time - this makes the program slightly more complicated, but is well worth it for the pins saved - particularly as it allows us to use just the one 10 pin connector.



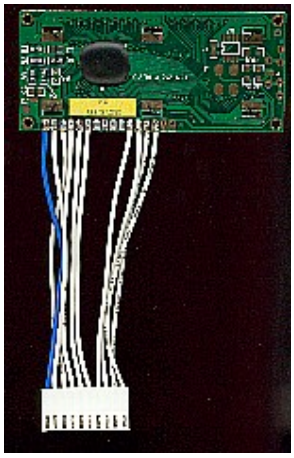
This is the top view of the LCD board, the upper connector goes to the main processor board, and the lower one is where the LCD module plugs in - you could solder the wires from the LCD directly to the board, but I chose to use a Molex plug and socket - so I can plug different LCD's into the same board. The LCD module is wired as the circuit above, with pins 7-10 of the module being ignored and being in sequence from pin 1 to pin 14.



The bottom of the LCD board, the two track breaks are marked with blue circles, and it only has six wire links on the top. The vertically mounted preset resistor is for setting the contrast of the display.



Front and rear views of the wired 2x16 LCD module, I used about 3 inches of wire to the Molex socket, notice the four connections left blank in the middle, these are the extra pins used for 8 bit mode. Also notice the single blue wire to the socket, I've done this on all the leads I've made up - it signifies pin 1 of the connector.



For the first parts of this tutorial you require the Main Board and the LCD Board, the later parts will also use the Switch Board, as written the tutorials use the LCD Board on PortA and the Switch Board on PortB. Although the hardware diagram shows a 2x16 LCD, other sizes can be used, I've tested it with a 2x16, 2x20, and 2x40 - all worked equally well. The intention is to develop a useful set of LCD routines, these will be used in the later parts of the tutorials to display various information.

Download zipped tutorial files.

LCD Command Control Codes

Command	Binary								Hex
	D7	D6	D5	D4	D3	D2	D1	D0	
Clear Display	0	0	0	0	0	0	0	1	01
Display and Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	I/D	S	01 to 07
Display On/Off and Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

I/D:	1=Increment*	0=Decrement	R/L:	1=Right Shift / 0=Left Shift
S:	1=Display Shift On	0=Display Shift off*	8/4:	1=8 bit interface* / 0=4 bit interface
D:	1=Display On	0=Display Off*	2/1:	1=2 line mode / 0=1 line mode*
U:	1=Cursor Underline On	0=Cursor Underline Off*	10/7:	1=5x10 dot format / 0=5x7 dot format*
B:	1=Cursor Blink On	0=Cursor Blink Off*		
D/C:	1=Display Shift	0=Cursor Move		*=initialisation setting / x=don't care

This table shows the command codes for the LCD module, it was taken from an excellent LCD tutorial that was published in the UK magazine 'Everyday Practical Electronics' February 1997 - it can be downloaded as a PDF file from the EPE website. The following routines are an amalgamation of a number of routines from various sources (including the previously mentioned tutorial), plus various parts of my own, the result is a set of reliable, easy to use, routines which work well (at least in my opinion!).

Tutorial 3.1 - requires Main Board and LCD Board.

This program displays a text message on the LCD module, it consists mostly of subroutines for using the LCD module.

```
;LCD text demo - 4 bit mode
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
```

```
       __config 0x3D18              ;sets   the   configuration   settings
(oscillator type etc.)



               cblock  0x20                 ;start   of   general   purpose
registers
                       count                ;used in looping routines
                       count1               ;used in delay routine
                       counta               ;used in delay routine
                       countb               ;used in delay routine
                       tmp1                 ;temporary storage
                       tmp2
                       templcd              ;temp store for 4 bit mode
                       templcd2
               endc

LCD_PORT       Equ     PORTA
LCD_TRIS       Equ     TRISA
LCD_RS         Equ     0x04                 ;LCD handshake lines
LCD_RW         Equ     0x06
LCD_E          Equ     0x07

               org     0x0000

               movlw   0x07
               movwf   CMCON                ;turn  comparators  off  (make  it
like a 16F84)

Initialise     clrf    count
               clrf    PORTA
               clrf    PORTB



SetPorts       bsf     STATUS,        RP0   ;select bank 1
               movlw   0x00                 ;make all pins outputs
               movwf   LCD_TRIS
               bcf     STATUS,        RP0   ;select bank 0

               call    Delay100             ;wait for LCD to settle


               call    LCD_Init             ;setup LCD


               clrf    count                ;set counter register to zero
Message        movf    count, w             ;put counter value in W
               call    Text                 ;get  a  character  from  the  text
table
               xorlw   0x00                 ;is it a zero?
               btfsc   STATUS, Z
               goto    NextMessage
               call    LCD_Char
               call    Delay255
               incf    count, f
               goto    Message

NextMessage    call    LCD_Line2            ;move to 2nd row, first column

               clrf    count                ;set counter register to zero
```

```
Message2        movf    count, w                ;put counter value in W
                call    Text2                   ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                goto    EndMessage
                call    LCD_Char
                incf    count, f
                goto    Message2

EndMessage

Stop            goto    Stop                    ;endless loop



;Subroutines and text tables

;LCD routines

;Initialise LCD
LCD_Init        movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd

                call    LCD_Clr                 ;clear display

                retlw   0x00

; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 0
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 0
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_CharD       addlw   0x30
LCD_Char        movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
```

```
                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line1W      addlw   0x80                    ;move to 1st row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_Line2W      addlw   0xc0                    ;move to 2nd row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00

LCD_HEX         movwf   tmp1
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7                    ;delay 1mS
                movwf   counta
```

```
                movlw   0x01
                movwf   countb
Delay_0
                decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

;end of LCD routines

HEX_Table       ADDWF   PCL       , f
                RETLW   0x30
                RETLW   0x31
                RETLW   0x32
                RETLW   0x33
                RETLW   0x34
                RETLW   0x35
                RETLW   0x36
                RETLW   0x37
                RETLW   0x38
                RETLW   0x39
                RETLW   0x41
                RETLW   0x42
                RETLW   0x43
                RETLW   0x44
                RETLW   0x45
                RETLW   0x46


Text            addwf   PCL, f
                retlw   'H'
                retlw   'e'
                retlw   'l'
                retlw   'l'
                retlw   'o'
                retlw   0x00

Text2           ADDWF   PCL, f
                RETLW   'R'
                RETLW   'e'
                RETLW   'a'
                RETLW   'd'
                RETLW   'y'
                RETLW   '.'
                RETLW   '.'
                RETLW   '.'
                RETLW   0x00
                end
```

As usual, first we need to set things up, after the normal variable declarations and port setting we reach 'call LCD_Init', this sets up the LCD module. It first waits for 100mS to give the

module plenty of time to settle down, we then set it to 4 bit mode (0x20) and set the various options how we want them - in this case, Display Shift is On (0x28), Character Entry Mode is Increment (0x06), and Block Cursor On (0x0D). Once the LCD is setup, we can then start to send data to it, this is read from a table, exactly the same as the LED sequencer in the earlier tutorials - except this time we send the data to the LCD module (using LCD_Char) and use a 0x00 to mark the end of the table, thus removing the need to maintain a count of the characters printed. Once the first line is displayed we then sent a command to move to the second line (using call LCD_Line2), and then print the second line from another table. After that we enter an endless loop to leave the display as it is.

This program introduces a new use of the 'goto' command, 'goto $+2' - '$' is an MPASM arithmetic operator, and uses the current value of the program counter, so 'goto $+2' means jump to the line after the next one - 'goto $+1' jumps to the next line, and may seem pretty useless (as the program was going to be there next anyway), but it can be extremely useful. A program branch instruction (like goto) uses two instruction cycles, whereas other instructions only take one, so if you use a 'nop' in a program it takes 1uS to execute, and carries on from the next line - however, if you use 'goto $+1' it still carries on from the next line, but now takes 2uS. You'll notice more use of the 'goto $' construction in later tutorials, if you are checking an input pin and waiting for it to change state you can use 'goto $-1' to jump back to the previous line, this saves allocating a label to the line that tests the condition.

This is a table of the LCD subroutines provided in these programs, you can easily add more if you wish - for instance to set a line cursor rather than a block one, if you find you are using a particular feature a lot you may as well make a subroutine for it.

LCD Subroutines	
LCD_Init	Initialise LCD Module
LCD_Cmd	Sent a command to the LCD
LCD_CharD	Add 0x30 to a byte and send to the LCD (to display numbers as ASCII)
LCD_Char	Send the character in W to the LCD
LCD_Line1	Go to start of line 1
LCD_Line2	Go to start of line 2
LCD_Line1W	Go to line 1 column W
LCD_Line2W	Go to line 2 column W
LCD_CurOn	Turn block cursor on
LCD_CurOff	Turn block cursor off
LCD_Clr	Clear the display
LCD_HEX	Display the value in W as Hexadecimal

Tutorial 3.2 - requires Main Board and LCD Board.

This program displays a text message on the top line and a running 16 bit counter on the bottom line, with the values displayed in both decimal and hexadecimal , it consists mostly of the previous subroutines for using the LCD module, plus an extra one for converting from 16 bit hexadecimal to decimal.

```
;LCD 16 bit counter
```

```
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)



                cblock  0x20                    ;start   of   general   purpose
registers
                        count                   ;used in looping routines
                        count1                  ;used in delay routine
                        counta                  ;used in delay routine
                        countb                  ;used in delay routine
                        tmp1                    ;temporary storage
                        tmp2
                        templcd                 ;temp store for 4 bit mode
                        templcd2

                        NumL                    ;Binary   inputs   for   decimal
convert routine
                        NumH

                        TenK                    ;Decimal  outputs  from  convert
routine
                        Thou
                        Hund
                        Tens
                        Ones
                endc

LCD_PORT        Equ     PORTA
LCD_TRIS        Equ     TRISA
LCD_RS          Equ     0x04                    ;LCD handshake lines
LCD_RW          Equ     0x06
LCD_E           Equ     0x07

                org     0x0000

                movlw   0x07
                movwf   CMCON                   ;turn  comparators  off  (make  it
like a 16F84)

Initialise      clrf    count
                clrf    PORTA
                clrf    PORTB
                clrf    NumL
                clrf    NumH



SetPorts        bsf     STATUS,         RP0     ;select bank 1
                movlw   0x00                    ;make all pins outputs
                movwf   LCD_TRIS
                bcf     STATUS,         RP0     ;select bank 0

                call    LCD_Init                ;setup LCD
```

```
                clrf    count                   ;set counter register to zero
Message         movf    count, w                ;put counter value in W
                call    Text                    ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                goto    NextMessage
                call    LCD_Char
                incf    count, f
                goto    Message

NextMessage     call    LCD_Line2               ;move to 2nd row, first column

                call    Convert                 ;convert to decimal
                movf    TenK,   w               ;display decimal characters
                call    LCD_CharD               ;using LCD_CharD to convert to
ASCII
                movf    Thou,   w
                call    LCD_CharD
                movf    Hund,   w
                call    LCD_CharD
                movf    Tens,   w
                call    LCD_CharD
                movf    Ones,   w
                call    LCD_CharD
                movlw   ' '                     ;display a 'space'
                call    LCD_Char
                movf    NumH,   w               ;and counter in hexadecimal
                call    LCD_HEX
                movf    NumL,   w
                call    LCD_HEX
                incfsz  NumL,   f
                goto    Next
                incf    NumH,   f
Next            call    Delay255                ;wait so you can see the digits
change
                goto    NextMessage


;Subroutines and text tables

;LCD routines

;Initialise LCD
LCD_Init        call    Delay100                ;wait for LCD to settle

                movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0c                    ;Set display on/off and cursor
command
                call    LCD_Cmd                 ;Set cursor off

                call    LCD_Clr                 ;clear display

                retlw   0x00
```

```
; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_CharD       addlw   0x30                    ;add 0x30 to convert to ASCII
LCD_Char        movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line1W      addlw   0x80                    ;move to 1st row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_Line2W      addlw   0xc0                    ;move to 2nd row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00
```

```
LCD_HEX         movwf   tmp1
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7                    ;delay 1mS
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0
                decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

;end of LCD routines

HEX_Table       ADDWF   PCL       , f
                RETLW   0x30
                RETLW   0x31
                RETLW   0x32
                RETLW   0x33
                RETLW   0x34
                RETLW   0x35
                RETLW   0x36
                RETLW   0x37
                RETLW   0x38
                RETLW   0x39
                RETLW   0x41
                RETLW   0x42
                RETLW   0x43
                RETLW   0x44
                RETLW   0x45
                RETLW   0x46
```

```
Text            addwf   PCL, f
                retlw   '1'
                retlw   '6'
                retlw   ' '
                retlw   'B'
                retlw   'i'
                retlw   't'
                retlw   ' '
                retlw   'C'
                retlw   'o'
                retlw   'u'
                retlw   'n'
                retlw   't'
                retlw   'e'
                retlw   'r'
                retlw   '.'
                retlw   0x00

;This routine downloaded from http://www.piclist.com
Convert:                        ; Takes number in NumH:NumL
                                ; Returns decimal in
                                ; TenK:Thou:Hund:Tens:Ones
        swapf   NumH, w
        iorlw   B'11110000'
        movwf   Thou
        addwf   Thou,f
        addlw   0XE2
        movwf   Hund
        addlw   0X32
        movwf   Ones

        movf    NumH,w
        andlw   0X0F
        addwf   Hund,f
        addwf   Hund,f
        addwf   Ones,f
        addlw   0XE9
        movwf   Tens
        addwf   Tens,f
        addwf   Tens,f

        swapf   NumL,w
        andlw   0X0F
        addwf   Tens,f
        addwf   Ones,f

        rlf     Tens,f
        rlf     Ones,f
        comf    Ones,f
        rlf     Ones,f

        movf    NumL,w
        andlw   0X0F
        addwf   Ones,f
        rlf     Thou,f

        movlw   0X07
        movwf   TenK

                    ; At this point, the original number is
                    ; equal to
                    ; TenK*10000+Thou*1000+Hund*100+Tens*10+Ones
```

```
                    ; if those entities are regarded as two's
                    ; complement binary.  To be precise, all of
                    ; them are negative except TenK.  Now the number
                    ; needs to be normalized, but this can all be
                    ; done with simple byte arithmetic.

        movlw   0X0A                            ; Ten
Lb1:
        addwf   Ones,f
        decf    Tens,f
        btfss   3,0
        goto   Lb1
Lb2:
        addwf   Tens,f
        decf    Hund,f
        btfss   3,0
        goto   Lb2
Lb3:
        addwf   Hund,f
        decf    Thou,f
        btfss   3,0
        goto   Lb3
Lb4:
        addwf   Thou,f
        decf    TenK,f
        btfss   3,0
        goto   Lb4

        retlw   0x00


               end
```

Tutorial 3.3 - requires Main Board and LCD Board.

This program displays a text message on the top line and a running 16 bit counter on the bottom line, just as the last example, however, instead of using the Delay calls this version waits until the LCD Busy flag is clear. The LCD module takes time to carry out commands, these times vary, and the previous tutorials used a delay more than long enough to 'make sure' - however, the modules have the capability of signalling when they are ready, this version uses that facility and avoids any unnecessary delays. I've also used the LCD_Line2W routine to position the numbers further to the right and demonstrate the use of the routine, another slight change is that the tables have been moved to the beginning of program memory, this was done because it's important that tables don't cross a 256 byte boundary, so putting them at the start avoids this.

```
;LCD 16 bit counter - using LCD Busy line
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL     0,      -302     ;suppress bank selection messages
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)
```

```
                cblock  0x20                    ;start   of   general   purpose
registers
                        count                   ;used in looping routines
                        count1                  ;used in delay routine
                        counta                  ;used in delay routine
                        countb                  ;used in delay routine
                        tmp1                    ;temporary storage
                        tmp2
                        templcd                 ;temp store for 4 bit mode
                        templcd2

                        NumL                    ;Binary   inputs   for   decimal
convert routine
                        NumH

                        TenK                    ;Decimal  outputs  from  convert
routine
                        Thou
                        Hund
                        Tens
                        Ones
                endc

LCD_PORT        Equ     PORTA
LCD_TRIS        Equ     TRISA
LCD_RS          Equ     0x04                    ;LCD handshake lines
LCD_RW          Equ     0x06
LCD_E           Equ     0x07

                org     0x0000
                goto    Start

HEX_Table       ADDWF   PCL       , f
                RETLW   0x30
                RETLW   0x31
                RETLW   0x32
                RETLW   0x33
                RETLW   0x34
                RETLW   0x35
                RETLW   0x36
                RETLW   0x37
                RETLW   0x38
                RETLW   0x39
                RETLW   0x41
                RETLW   0x42
                RETLW   0x43
                RETLW   0x44
                RETLW   0x45
                RETLW   0x46


Text            addwf   PCL, f
                retlw   '1'
                retlw   '6'
                retlw   ' '
                retlw   'B'
                retlw   'i'
                retlw   't'
                retlw   ' '
                retlw   'C'
                retlw   'o'
                retlw   'u'
```

```
                retlw   'n'
                retlw   't'
                retlw   'e'
                retlw   'r'
                retlw   '.'
                retlw   0x00

Start           movlw   0x07
                movwf   CMCON                   ;turn comparators off (make it
like a 16F84)

Initialise      clrf    count
                clrf    PORTA
                clrf    PORTB
                clrf    NumL
                clrf    NumH



SetPorts        bsf     STATUS,         RP0     ;select bank 1
                movlw   0x00                    ;make all pins outputs
                movwf   LCD_TRIS
                movwf   TRISB
                bcf     STATUS,         RP0     ;select bank 0

                call    LCD_Init                ;setup LCD


                clrf    count                   ;set counter register to zero
Message         movf    count, w                ;put counter value in W
                call    Text                    ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                goto    NextMessage
                call    LCD_Char
                incf    count, f
                goto    Message

NextMessage     movlw   d'2'
                call    LCD_Line2W              ;move to 2nd row, third column

                call    Convert                 ;convert to decimal
                movf    TenK,   w               ;display decimal characters
                call    LCD_CharD               ;using LCD_CharD to convert to
ASCII
                movf    Thou,   w
                call    LCD_CharD
                movf    Hund,   w
                call    LCD_CharD
                movf    Tens,   w
                call    LCD_CharD
                movf    Ones,   w
                call    LCD_CharD
                movlw   ' '                     ;display a 'space'
                call    LCD_Char
                movf    NumH,   w               ;and counter in hexadecimal
                call    LCD_HEX
                movf    NumL,   w
                call    LCD_HEX
                incfsz  NumL,   f
                goto    Next
```

```
                incf    NumH,   f
Next            call    Delay255                ;wait so you can see the digits
change
                goto    NextMessage


;Subroutines and text tables

;LCD routines

;Initialise LCD
LCD_Init        call    LCD_Busy                ;wait for LCD to settle

                movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0c                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd                 ;Set cursor off

                call    LCD_Clr                 ;clear display

                retlw   0x00

; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    LCD_Busy
                retlw   0x00

LCD_CharD       addlw   0x30                    ;add 0x30 to convert to ASCII
LCD_Char        movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    LCD_Busy
                retlw   0x00
```

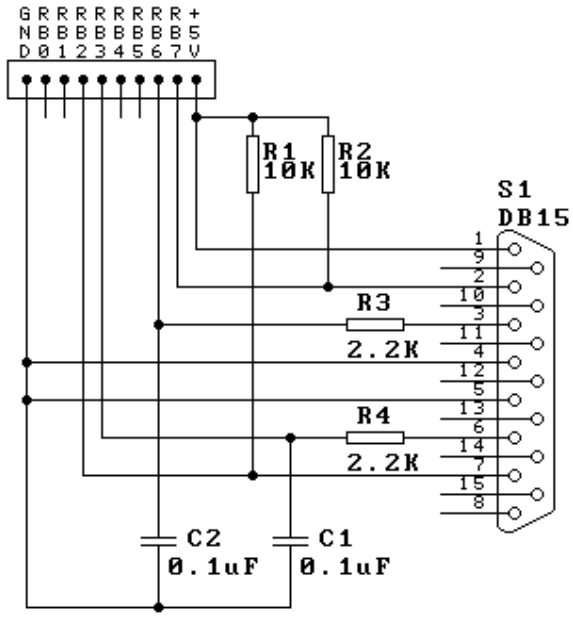
```
LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line1W      addlw   0x80                    ;move to 1st row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_Line2W      addlw   0xc0                    ;move to 2nd row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00

LCD_HEX         movwf   tmp1
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Delay255        movlw   0xff                    ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'                  ;delay 100mS
                goto    d0
Delay50         movlw   d'50'                   ;delay 50mS
                goto    d0
Delay20         movlw   d'20'                   ;delay 20mS
                goto    d0
Delay5          movlw   0x05                    ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7                    ;delay 1mS
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0
                decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
```

```
                goto    d1
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

LCD_Busy
                bsf     STATUS, RP0             ;set bank 1
                movlw   0x0f                    ;set Port for input
                movwf   LCD_TRIS
                bcf     STATUS, RP0             ;set bank 0
                bcf     LCD_PORT, LCD_RS        ;set LCD for command mode
                bsf     LCD_PORT, LCD_RW        ;setup to read busy flag
                bsf     LCD_PORT, LCD_E
                swapf   LCD_PORT, w             ;read upper nibble (busy flag)
                bcf     LCD_PORT, LCD_E
                movwf   templcd2
                bsf     LCD_PORT, LCD_E         ;dummy read of lower nibble
                bcf     LCD_PORT, LCD_E
                btfsc   templcd2, 7             ;check busy flag, high = busy
                goto    LCD_Busy                ;if busy check again
                bcf     LCD_PORT, LCD_RW
                bsf     STATUS, RP0             ;set bank 1
                movlw   0x00                    ;set Port for output
                movwf   LCD_TRIS
                bcf     STATUS, RP0             ;set bank 0
                return

;end of LCD routines



;This routine downloaded from http://www.piclist.com
Convert:                        ; Takes number in NumH:NumL
                                ; Returns decimal in
                                ; TenK:Thou:Hund:Tens:Ones
        swapf   NumH, w
        iorlw   B'11110000'
        movwf   Thou
        addwf   Thou,f
        addlw   0XE2
        movwf   Hund
        addlw   0X32
        movwf   Ones

        movf    NumH,w
        andlw   0X0F
        addwf   Hund,f
        addwf   Hund,f
        addwf   Ones,f
        addlw   0XE9
        movwf   Tens
        addwf   Tens,f
        addwf   Tens,f

        swapf   NumL,w
        andlw   0X0F
        addwf   Tens,f
        addwf   Ones,f
```

```
        rlf     Tens,f
        rlf     Ones,f
        comf    Ones,f
        rlf     Ones,f

        movf    NumL,w
        andlw   0X0F
        addwf   Ones,f
        rlf     Thou,f

        movlw   0X07
        movwf   TenK

                ; At this point, the original number is
                ; equal to
                ; TenK*10000+Thou*1000+Hund*100+Tens*10+Ones
                ; if those entities are regarded as two's
                ; complement binary.  To be precise, all of
                ; them are negative except TenK.  Now the number
                ; needs to be normalized, but this can all be
                ; done with simple byte arithmetic.

        movlw   0X0A                            ; Ten
Lb1:
        addwf   Ones,f
        decf    Tens,f
        btfss   3,0
        goto   Lb1
Lb2:
        addwf   Tens,f
        decf    Hund,f
        btfss   3,0
        goto   Lb2
Lb3:
        addwf   Hund,f
        decf    Thou,f
        btfss   3,0
        goto   Lb3
Lb4:
        addwf   Thou,f
        decf    TenK,f
        btfss   3,0
        goto   Lb4

        retlw  0x00


               end
```

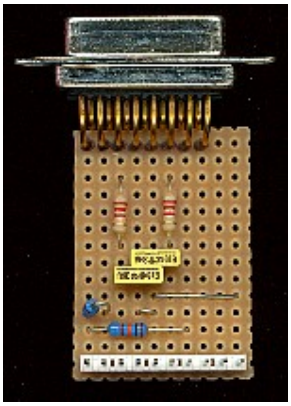
PIC Tutorial Four – Joysticks

Joystick Board

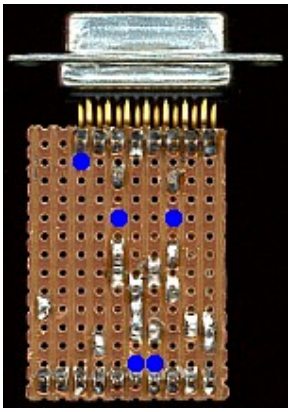


This is the Joystick Board, used for connecting a standard PC analogue joystick. It connects to 4 pins of one port and uses a simple capacitor charging technique to read the analogue resistance of the joystick. The circuit is nice and simple, R1 and R2 are pull-up resistors for the two trigger buttons on the joystick (which connect either pin 2 or pin 7 of the 15 way D connector to ground). The analogue inputs are on pins 3 and 6, and consist of 100K variable resistors from these pins to pin 1 (the 5V supply). From the analogue controls we feed through R3 or R4, these are to set the minimum resistance (2.2K when the joystick controls are at minimum). The current through these resistors is used to charge C2 (or C1), and the charging time is dependent on the value of the joystick + R3 (or R4). To read the controls we discharge the capacitor (by setting the relevant port pin to an output and setting it low), then reset the port pin to be an input and wait until the capacitor charges enough to make the input switch high - during this time we maintain a 16 bit count - this gives us a value based on the position of the joystick.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.



This is the top view of the Joystick Board.



The bottom of the Joystick Board.

For this tutorial you require the Main Board, LCD Board and the Joystick Board, as written the tutorials use the LCD Board on PortA and the Joystick Board on PortB - although these could easily be swapped over, as the Joystick Board doesn't use either of the two 'difficult' pins for PortA, pins 4 and 5. Download zipped tutorial files.

Tutorial 4.1 - requires Main Board, LCD Board and Joystick Board.

This program uses the LCD module to give a hexadecimal display of the values of the X and Y resistors in the joystick, and an indication if the buttons are pressed or not. It uses the LCD subroutines from the previous tutorial, and subroutines written to read the joystick.

```
;Joystick routines with LCD display
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)



                cblock  0x20                    ;start   of   general   purpose
registers
                        count                   ;used in looping routines
                        count1                  ;used in delay routine
                        counta                  ;used in delay routine
                        countb                  ;used in delay routine
```

```
                        tmp1                    ;temporary storage
                        tmp2
                        templcd                 ;temp store for 4 bit mode
                        templcd2
                        HiX                     ;result for X pot
                        LoX
                        HiY                     ;result for Y pot
                        LoY
                        Flags
                endc

LCD_PORT        Equ     PORTA
LCD_TRIS        Equ     TRISA
LCD_RS          Equ     0x04                    ;LCD handshake lines
LCD_RW          Equ     0x06
LCD_E           Equ     0x07

JOY_PORT        Equ     PORTB
JOY_TRIS        Equ     TRISB
PotX            Equ     0x06                    ;input assignments for joystick
PotY            Equ     0x03
SW1             Equ     0x07
SW2             Equ     0x02
SW1_Flag        Equ     0x01                    ;flags used for key presses
SW2_Flag        Equ     0x02

                org     0x0000
                goto    Start

;TABLES - moved to start of program memory to avoid paging problems,
;a table must not cross a 256 byte boundary.
HEX_Table       ADDWF   PCL       , f
                RETLW   0x30
                RETLW   0x31
                RETLW   0x32
                RETLW   0x33
                RETLW   0x34
                RETLW   0x35
                RETLW   0x36
                RETLW   0x37
                RETLW   0x38
                RETLW   0x39
                RETLW   0x41
                RETLW   0x42
                RETLW   0x43
                RETLW   0x44
                RETLW   0x45
                RETLW   0x46


Xtext           addwf   PCL, f
                retlw   'J'
                retlw   'o'
                retlw   'y'
                retlw   '-'
                retlw   'X'
                retlw   ' '
                retlw   0x00

Ytext           addwf   PCL, f
                retlw   'J'
                retlw   'o'
```

```
                retlw   'y'
                retlw   '-'
                retlw   'Y'
                retlw   ' '
                retlw   0x00

presstext       addwf   PCL, f
                retlw   'C'
                retlw   'l'
                retlw   'o'
                retlw   's'
                retlw   'e'
                retlw   0x00

nopresstext     addwf   PCL, f
                retlw   'O'
                retlw   'p'
                retlw   'e'
                retlw   'n'
                retlw   ' '
                retlw   0x00

;end of tables

Start           movlw   0x07
                movwf   CMCON                   ;turn comparators off (make it
like a 16F84)

Initialise      clrf    count
                clrf    PORTA
                clrf    PORTB
                bcf     Flags,  SW1_Flag        ;clear button pressed flags
                bcf     Flags,  SW2_Flag



SetPorts        bsf     STATUS,         RP0     ;select bank 1
                movlw   0x00                    ;make all LCD pins outputs
                movwf   LCD_TRIS
                movlw   0xff                    ;make all joystick pins inputs
                movwf   JOY_TRIS
                bcf     STATUS,         RP0     ;select bank 0

                call    JOY_Init                ;discharge timing capacitors
                call    Delay100                ;wait for LCD to settle
                call    LCD_Init                ;setup LCD module

Main
                call    ReadX                   ;read X joystick
                call    ReadY                   ;read Y joystick
                call    ReadSW                  ;read switches

                call    LCD_Line1               ;set to first line
                call    XString                 ;display Joy-X string
                movf    HiX,    w               ;display high byte
                call    LCD_HEX
                movf    LoX,    w               ;display low byte
                call    LCD_HEX
                movlw   ' '
                call    LCD_Char
                call    DisplaySW1
```

```
                call    LCD_Line2               ;set to second line
                call    YString                 ;display Joy-Y string
                movf    HiY,    w               ;display high byte
                call    LCD_HEX
                movf    LoY,    w               ;display low byte
                call    LCD_HEX
                movlw   ' '
                call    LCD_Char
                call    DisplaySW2
                goto    Main                    ;loop for ever



;Subroutines and text tables

DisplaySW1      btfsc   Flags,  SW1_Flag
                goto    Press_Str
                btfss   Flags,  SW1_Flag
                goto    NoPress_Str
                retlw   0x00

DisplaySW2      btfsc   Flags,  SW2_Flag
                goto    Press_Str
                btfss   Flags,  SW2_Flag
                goto    NoPress_Str
                retlw   0x00

XString         clrf    count                   ;set counter register to zero
Mess1           movf    count, w                ;put counter value in W
                call    Xtext                   ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess1

YString         clrf    count                   ;set counter register to zero
Mess2           movf    count, w                ;put counter value in W
                call    Ytext                   ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess2

Press_Str       clrf    count                   ;set counter register to zero
Mess3           movf    count, w                ;put counter value in W
                call    presstext               ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess3

NoPress_Str     clrf    count                   ;set counter register to zero
```

```
Mess4           movf    count, w                ;put counter value in W
                call    nopresstext             ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess4

;LCD routines

;Initialise LCD
LCD_Init        movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd

                call    LCD_Clr                 ;clear display

                retlw   0x00

; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_CharD       addlw   0x30                    ;convert   numbers   to   ASCII
values
LCD_Char        movwf   templcd                 ;display    character    in    W
register
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
```

```
                retlw   0x00

LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set block cursor on
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set block cursor off
                call    LCD_Cmd
                retlw   0x00

LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00

LCD_HEX         movwf   tmp1                    ;display W as hexadecimal byte
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

;end of LCD routines

;joystick routines

JOY_Init                                ;setup joystick port
        bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotX    ;make PotX an output
        bcf     JOY_PORT,       PotX    ;discharge capacitor
        bcf     JOY_TRIS,       PotY    ;make PotY an output
        bcf     JOY_PORT,       PotY    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadX
        clrf    HiX                     ;reset counter registers
        clrf    LoX
        bsf     STATUS,         RP0     ;select bank 1
        bsf     JOY_TRIS,       PotX    ;make PotX an input
        bcf     STATUS,         RP0     ;select bank 0

x1
        btfsc   JOY_PORT,       PotX    ;keep going until input high
        goto    EndX
        incfsz  LoX,f
```

```
        goto    x1
        incfsz  HiX,f
        goto    x1

EndX    bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotX    ;make PotX an output
        bcf     JOY_PORT,       PotX    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadY
        clrf    HiY                     ;reset counter registers
        clrf    LoY
        call    Delay5
        bsf     STATUS,         RP0     ;select bank 1
        bsf     JOY_TRIS,       PotY    ;make PotY an input
        bcf     STATUS,         RP0     ;select bank 0

y1
        btfsc   JOY_PORT,       PotY    ;keep going until input high
        goto    EndY
        incfsz  LoY,f
        goto    y1
        incfsz  HiY,f
        goto    y1

EndY    bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotY    ;make PotY an output
        bcf     JOY_PORT,       PotY    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadSW  btfss   JOY_PORT,       SW1
        call    Sw1On
        btfss   JOY_PORT,       SW2
        call    Sw2On
        btfsc   JOY_PORT,       SW1
        call    Sw1Off
        btfsc   JOY_PORT,       SW2
        call    Sw2Off
        retlw   0x00

Sw1On   bsf     Flags,  SW1_Flag
        retlw   0x00

Sw2On   bsf     Flags,  SW2_Flag
        retlw   0x00

Sw1Off  bcf     Flags,  SW1_Flag
        retlw   0x00

Sw2Off  bcf     Flags,  SW2_Flag
        retlw   0x00

;end of joystick routines

;Delay routines

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
```

```
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7                    ;delay 1mS
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0
                decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

;end of Delay routines


                end
```

Tutorial 4.2 - requires Main Board, LCD Board and Joystick Board.

This second program is very similar to the previous one, except it uses a different method of counting the time taken to charge the capacitor. Whereas the first example used a simple software counter, this one uses a hardware timer for the lower byte, and an interrupt driven routine for the upper byte. This has the major advantage of being more accurate, giving 1uS resolution.

```
;Joystick routines with LCD display
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)



                cblock  0x20                    ;start   of   general   purpose
registers
                        count                   ;used in looping routines
                        count1                  ;used in delay routine
                        counta                  ;used in delay routine
                        countb                  ;used in delay routine
                        tmp1                    ;temporary storage
                        tmp2
                        templcd                 ;temp store for 4 bit mode
                        templcd2
                        HiX                     ;result for X pot
                        LoX
                        HiY                     ;result for Y pot
                        LoY
```

```
                Timer_H
                Flags
            endc

LCD_PORT    Equ     PORTA
LCD_TRIS    Equ     TRISA
LCD_RS      Equ     0x04                ;LCD handshake lines
LCD_RW      Equ     0x06
LCD_E       Equ     0x07

JOY_PORT    Equ     PORTB
JOY_TRIS    Equ     TRISB
PotX        Equ     0x06                ;input assignments for joystick
PotY        Equ     0x03
SW1         Equ     0x07
SW2         Equ     0x02
SW1_Flag    Equ     0x01                ;flags used for key presses
SW2_Flag    Equ     0x02

            org     0x0000
            goto    Start

            ORG     0x0004
            BCF     INTCON,     T0IF
            INCF    Timer_H,    f
            RETFIE



Start       movlw   0x07
            movwf   CMCON               ;turn comparators off (make it
like a 16F84)

Initialise  clrf    count
            clrf    PORTA
            clrf    PORTB
            bcf     Flags,  SW1_Flag    ;clear button pressed flags
            bcf     Flags,  SW2_Flag



SetPorts    bsf     STATUS,     RP0     ;select bank 1
            movlw   0x00                ;make all LCD pins outputs
            movwf   LCD_TRIS
            movlw   0xff                ;make all joystick pins inputs
            movwf   JOY_TRIS
            MOVLW   0x88                ;assign prescaler to watchdog
            MOVWF   OPTION_REG
            bcf     STATUS,     RP0     ;select bank 0
            CLRF    INTCON
            BSF     INTCON  , T0IE      ;enable timer interrupts

            call    JOY_Init            ;discharge timing capacitors
            call    Delay100            ;wait for LCD to settle
            call    LCD_Init            ;setup LCD module

Main
            call    ReadX               ;read X joystick
            call    ReadY               ;read Y joystick
            call    ReadSW              ;read switches

            call    LCD_Line1           ;set to first line
```

```
                call    XString                 ;display Joy-X string
                movf    HiX,    w               ;display high byte
                call    LCD_HEX
                movf    LoX,    w               ;display low byte
                call    LCD_HEX
                movlw   ' '
                call    LCD_Char
                call    DisplaySW1

                call    LCD_Line2               ;set to second line
                call    YString                 ;display Joy-Y string
                movf    HiY,    w               ;display high byte
                call    LCD_HEX
                movf    LoY,    w               ;display low byte
                call    LCD_HEX
                movlw   ' '
                call    LCD_Char
                call    DisplaySW2
                goto    Main                    ;loop for ever



;Subroutines and text tables

DisplaySW1      btfsc   Flags,  SW1_Flag
                goto    Press_Str
                btfss   Flags,  SW1_Flag
                goto    NoPress_Str
                retlw   0x00

DisplaySW2      btfsc   Flags,  SW2_Flag
                goto    Press_Str
                btfss   Flags,  SW2_Flag
                goto    NoPress_Str
                retlw   0x00

XString         clrf    count                   ;set counter register to zero
Mess1           movf    count, w                ;put counter value in W
                call    Xtext                   ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess1

YString         clrf    count                   ;set counter register to zero
Mess2           movf    count, w                ;put counter value in W
                call    Ytext                   ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess2

Press_Str       clrf    count                   ;set counter register to zero
Mess3           movf    count, w                ;put counter value in W
```

```
                call    presstext               ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess3

NoPress_Str     clrf    count                   ;set counter register to zero
Mess4           movf    count, w                ;put counter value in W
                call    nopresstext             ;get a character from the text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess4

;LCD routines

;Initialise LCD
LCD_Init        movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0d                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd

                call    LCD_Clr                 ;clear display

                retlw   0x00

; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_CharD       addlw   0x30                    ;convert   numbers   to   ASCII
values
LCD_Char        movwf   templcd                 ;display    character    in    W
register
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
```

```
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    Delay5
                retlw   0x00

LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set block cursor on
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set block cursor off
                call    LCD_Cmd
                retlw   0x00

LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00

LCD_HEX         movwf   tmp1                    ;display W as hexadecimal byte
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

;end of LCD routines

;joystick routines

JOY_Init                                ;setup joystick port
        bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotX    ;make PotX an output
        bcf     JOY_PORT,       PotX    ;discharge capacitor
        bcf     JOY_TRIS,       PotY    ;make PotY an output
        bcf     JOY_PORT,       PotY    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadX   clrf    Timer_H                 ;clear timer hi byte
```

```
        bsf     STATUS,         RP0     ;select bank 1
        bsf     JOY_TRIS,       PotX    ;make PotX an input
        bcf     STATUS,         RP0     ;select bank 0
        clrf    TMR0
        bcf     INTCON,         T0IF    ;start timer
        bsf     INTCON,         GIE     ;start interrupts
        btfss   JOY_PORT,       PotX
        goto    $-1                     ;loop until input high
        cltw
        iorwf   TMR0,           f       ;stop timer (for 3 cycles)
        movf    TMR0,           W
        movwf   LoX                     ;and read immediately
        movf    Timer_H,        W
        movwf   HiX
        bcf     INTCON,         GIE     ;turn off interrupts
        btfsc   INTCON,                 GIE
        goto    $-2
        bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotX    ;make PotX an output
        bcf     JOY_PORT,       PotX    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadY   clrf    Timer_H                 ;clear timer hi byte
        bsf     STATUS,         RP0     ;select bank 1
        bsf     JOY_TRIS,       PotY    ;make PotY an input
        bcf     STATUS,         RP0     ;select bank 0
        clrf    TMR0
        bcf     INTCON,                 T0IF    ;start timer
        bsf     INTCON,                 GIE     ;start interrupts
        btfss   JOY_PORT,       PotY
        goto    $-1                     ;loop until input high
        clrw
        iorwf   TMR0,           f       ;stop timer (for 3 cycles)
        movf    TMR0,           W
        movwf   LoY                     ;and read immediately
        movf    Timer_H,        W
        movwf   HiY
        bcf     INTCON,         GIE     ;turn off interrupts
        btfsc   INTCON,                 GIE
        goto    $-2
        bsf     STATUS,         RP0     ;select bank 1
        bcf     JOY_TRIS,       PotY    ;make PotY an output
        bcf     JOY_PORT,       PotY    ;discharge capacitor
        bcf     STATUS,         RP0     ;select bank 0
        retlw   0x00

ReadSW  btfss   JOY_PORT,       SW1
        call    Sw1On
        btfss   JOY_PORT,       SW2
        call    Sw2On
        btfsc   JOY_PORT,       SW1
        call    Sw1Off
        btfsc   JOY_PORT,       SW2
        call    Sw2Off
        retlw   0x00

Sw1On   bsf     Flags,  SW1_Flag
        retlw   0x00

Sw2On   bsf     Flags,  SW2_Flag
        retlw   0x00
```

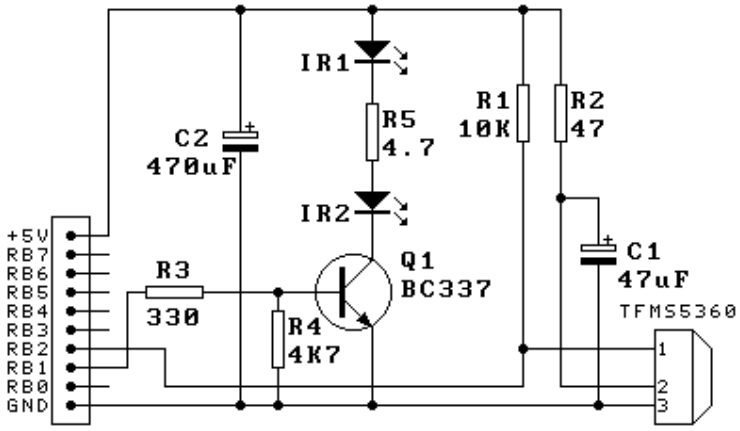
```
Sw1Off  bcf     Flags,  SW1_Flag
        retlw   0x00

Sw2Off  bcf     Flags,  SW2_Flag
        retlw   0x00

;end of joystick routines

;Delay routines

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7                    ;delay 1mS
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0
                decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

;end of Delay routines

        ORG     0x0100

;TABLES - moved to avoid paging problems,
;a table must not cross a 256 byte boundary.
HEX_Table       ADDWF   PCL       , f
                RETLW   0x30
                RETLW   0x31
                RETLW   0x32
                RETLW   0x33
                RETLW   0x34
                RETLW   0x35
                RETLW   0x36
                RETLW   0x37
                RETLW   0x38
                RETLW   0x39
                RETLW   0x41
                RETLW   0x42
                RETLW   0x43
                RETLW   0x44
                RETLW   0x45
                RETLW   0x46


Xtext           addwf   PCL, f
                retlw   'J'
```

```
                retlw   'o'
                retlw   'y'
                retlw   '-'
                retlw   'X'
                retlw   ' '
                retlw   0x00

Ytext           addwf   PCL, f
                retlw   'J'
                retlw   'o'
                retlw   'y'
                retlw   '-'
                retlw   'Y'
                retlw   ' '
                retlw   0x00

presstext       addwf   PCL, f
                retlw   'C'
                retlw   'l'
                retlw   'o'
                retlw   's'
                retlw   'e'
                retlw   0x00

nopresstext     addwf   PCL, f
                retlw   'O'
                retlw   'p'
                retlw   'e'
                retlw   'n'
                retlw   ' '
                retlw   0x00

;end of tables

                end
```

The first change in this program is the main program start address, previously we started as immediately after the reset vector (0x0000) as possible, but the interrupt vector is located at 0x0004 so we need to skip over this with a 'goto Start' command. The small interrupt routine itself is located an 0x0004 and simply increments the high byte counter every time the hardware low byte counter overflows. The other two lines in the interrupt routine re-enable interrupts (they are cancelled automatically when called) and the return from the routine, this time using 'retfie' (Return From Interrupt) rather than 'retlw'.

PIC Tutorial Five - Infrared Communication

Infrared Board



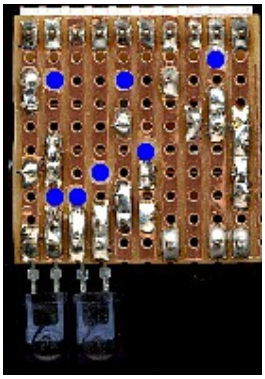
This is the Infrared Board, we need two of these, so that we can communicate between two main boards, it consists of two distinct parts. Firstly the IR receiver, comprising R1, R2, C1, and the IR receiver I/C itself (feeding port pin 2), and secondly the IR transmitter, comprising Q1, R3, R4, R5,C2, IR1, and IR2 (fed from port pin 1). If you only want to do one way communication you could build just the transmitter on one board, and just the receiver on the other, but building both on both boards gives the possibility of two way communication.

The receiver I/C detects IR signals modulated with a 38KHz signal, R2 and C1 are to provide decoupling for the supply (to avoid instability problems), and R1 is just a pull-up resistor as the I/C has an open-collector output (just like RA4 on the PIC).

The transmitter is a simple single transistor digital switch, when pin RB1 goes high this turns the transistor on, passing current through the IR LED's, with the current limited by R5 between the LED's. This passes quite a high current through the LED's and it's important that they are pulsed and not left on permanently or damage will probably occur - C2 is fitted to provide the required high current pulses without upsetting the main 5V rail. By pulsing the LED's with high current we increase the range and lower the current requirements - this is standard practice in IR remote controls, R5 limits the current through the LED's. As the receiver detects 38KHz modulation, we need to pulse the LED's at 38KHz, this can be done by feeding the LED's with a 13uS pulse followed by a 13uS space - in actual fact I decrease the pulse length, and increase the space length (keeping the total length at 26uS) - this reduces the power consumption.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.

This is the top view of the Infrared Board, there are only two wire links.



The bottom of the Infrared Board, it has seven track breaks, marked with blue circles (as usual).

To complete all of these tutorials you will require two Main Boards, two IR Boards, the LCD Board, the Switch Board, and the LED Board, as written the first two tutorials use the LCD Board and Switch Board on PortA and the IR Boards on PortB - although these could easily be swapped over, as the IR Board doesn't use either of the two 'difficult' pins for PortA, pins 4 and 5. The third tutorial uses the IR Board on PortA and the LED Board on PortB (as we require all 8 pins to be outputs). Download zipped tutorial files.

IR transmission has limitations, the most important one (for our purposes) being that the receiver doesn't give out the same width pulses that we transmit, so we can't just use a normal, RS232 type, serial data stream, where we simply sample the data at fixed times - the length of the received data varies with the number of ones sent - making receiving it accurately very difficult. Various different schemes are used by the manufacturers of IR remote controls, and some are much more complicated than others.

I've chosen to use the Sony SIRC (Sony Infra Red Control) remote control system, many of you may already have a suitable Sony remote at home you can use, and it's reasonably easy to understand and implement. Basically it uses a pulse width system, with a start bit of 2.4mS, followed by 12 data bits, where a '1' is 1.2mS wide, and a '0' is 0.6mS wide, the bits are all separated by gaps of 0.6mS. The data itself consists of a 7 bit 'command' code, and a 5 bit 'device' code - where a command is Channel 1, Volume Up etc. and a device is TV, VCR etc. This is how the same remote system can be used for different appliances, the same command for 'Power On' is usually used by all devices, but by transmitting a device ID only a TV will respond to 'TV Power On' command.

The table to the right shows the data format, after the Start bit the command code is send, lowest bit first, then the device code, again lowest bit first. The entire series is sent

Start	Command Code							Device Code				
S	D0	D1	D2	D3	D4	D5	D6	C0	C1	C2	C3	C4
2.4mS	1.2 or 0.6mS							1.2 or 0.6mS				

repeatedly while the button is held down, every 45mS. In order to decode the transmissions we need to measure the width of the pulses, first looking for the long 'start' pulse, then measuring the next 12 pulses and deciding if they are 1's or 0's. To do this I'm using a simple software 8 bit counter, with NOP's in the loop to make sure we don't overflow the counter. After measuring one pulse we then test it to see if it's a valid pulse, this routine provides four possible responses 'Start Pulse', 'One', 'Zero', or 'Error', we initially loop until we get a 'Start Pulse' reply, then read the next 12 bits - if the reply to any of these 12 is other than 'One' or 'Zero' we abort the read and go back to waiting for a 'Start Pulse'.

Device ID's	
TV	1
VTR1	2
Text	3
Widescreen	4
MDP	6
VTR2	7
VTR3	11
Effect	12
Audio	16
Pro-Logic	18
DVD	26

The device codes used specify the particular device, but with a few exceptions!, while a TV uses device code 1, some of the Teletext buttons use code 3, as do the Fastext coloured keys - where a separate Widescreen button is fitted, this uses code 4. The table to the left shows some of the Device ID codes I found on a sample of Sony remotes. Five bits gives a possible 32 different device ID's, and some devices respond to more than one device ID, for example some of the current Sony VCR's have the Play button in a 'cursor' type of design, surrounded by 'Stop', 'Pause', 'Rewind', and 'Fast Forward' - the ones I tested actually send a DVD ID code when these keys are pressed (along with a different command ID to that used normally used for 'Play' etc.). However, they still respond to an older Sony remote which sends the VTR3 device ID, which despite being labelled VTR3 on TV remotes seems to be the normal standard Sony VCR device ID. It's quite common for Sony remotes to use more than one device ID, a Surround Sound Amplifier Remote I tried used four different device ID's.

If you don't have a Sony remote you can use, I've also built a transmitter, using the second Main Board, second IR Board, and the Switch Board, the four buttons allow you to send four different command codes - I've chosen TV as the device, and Volume Up, Volume Down, Program Up, and Program Down as my four commands, I've confirmed this works on various Sony TV's. Transmitting the SIRC code is quite simple to do, I generate the 38KHz modulation directly in software, and to reduce current consumption don't use a 50/50 on/off ratio - by using a longer off than on time we still get the 38KHz, but with a reduced power requirement.

Tutorial 5.1 - requires one Main Board (with LED set to RB7), one IR Board and LCD Board.

This program uses the LCD module to give a decimal display of the values of the Device and Command bytes transmitted by a Sony SIRC remote control, it can be easily altered to operate port pins to control external devices, as an example the main board LED is turned on by pressing button 2, turned off by pressing button 3, and toggled on and off by pressing button 1 (all on a TV remote, you can change the device ID for a different remote if you need to). As it stands it's very useful for displaying the data transmitted by each button on your Sony remote control - the **Device ID's** table above was obtained using this design.

```
;Tutorial 5_1
;Read SIRC IR with LCD display
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
```

```
       __config 0x3D18              ;sets   the   configuration   settings
(oscillator type etc.)



               cblock  0x20                 ;start   of   general   purpose
registers
                       count                ;used in looping routines
                       count1               ;used in delay routine
                       counta               ;used in delay routine
                       countb               ;used in delay routine
                       LoX
                       Bit_Cntr
                       Cmd_Byte
                       Dev_Byte
                       Timer_H
                       Flags
                       Flags2
                       tmp1                 ;temporary storage
                       tmp2
                       tmp3
                       lastdev
                       lastkey

                       NumL                 ;Binary   inputs   for   decimal
convert routine
                       NumH

                       TenK                 ;Decimal  outputs  from  convert
routine
                       Thou
                       Hund
                       Tens
                       Ones

                       templcd              ;temp store for 4 bit mode
                       templcd2
               endc

LCD_PORT       Equ     PORTA
LCD_TRIS       Equ     TRISA
LCD_RS         Equ     0x04                 ;LCD handshake lines
LCD_RW         Equ     0x06
LCD_E          Equ     0x07

IR_PORT        Equ     PORTB
IR_TRIS        Equ     TRISB
IR_In          Equ     0x02                 ;input assignment for IR data

OUT_PORT       Equ     PORTB
LED            Equ     0x07

ErrFlag        Equ     0x00
StartFlag      Equ     0x01                 ;flags used for received bit
One            Equ     0x02
Zero           Equ     0x03

New            Equ     0x07                 ;flag used to show key released

TV_ID          Equ     0x01                 ;TV device ID
```

```
But1            Equ     0x00                    ;numeric button ID's
But2            Equ     0x01
But3            Equ     0x02
But4            Equ     0x03
But5            Equ     0x04
But6            Equ     0x05
But7            Equ     0x06
But8            Equ     0x07
But9            Equ     0x08

                org     0x0000
                goto    Start

                org     0x0004
                retfie

;TABLES - moved to start of page to avoid paging problems,
;a table must not cross a 256 byte boundary.
HEX_Table       addwf   PCL       , f
                retlw   0x30
                retlw   0x31
                retlw   0x32
                retlw   0x33
                retlw   0x34
                retlw   0x35
                retlw   0x36
                retlw   0x37
                retlw   0x38
                retlw   0x39
                retlw   0x41
                retlw   0x42
                retlw   0x43
                retlw   0x44
                retlw   0x45
                retlw   0x46

Xtext           addwf   PCL, f
                retlw   'D'
                retlw   'e'
                retlw   'v'
                retlw   'i'
                retlw   'c'
                retlw   'e'
                retlw   ' '
                retlw   ' '
                retlw   ' '
                retlw   'C'
                retlw   'o'
                retlw   'm'
                retlw   'm'
                retlw   'a'
                retlw   'n'
                retlw   'd'
                retlw   0x00

;end of tables

Start           movlw   0x07
                movwf   CMCON                   ;turn comparators off (make it
like a 16F84)

Initialise      clrf    count
```

```
                clrf    PORTA
                clrf    PORTB
                clrf    Flags
                 clrf    Dev_Byte
                clrf    Cmd_Byte


SetPorts        bsf     STATUS,         RP0     ;select bank 1
                movlw   0x00                    ;make all LCD pins outputs
                movwf   LCD_TRIS
                movlw   b'01111111'             ;make  all  IR  port  pins  inputs
(except RB7)
                movwf   IR_TRIS
                bcf     STATUS,         RP0     ;select bank 0

                call    LCD_Init                ;setup LCD module
                call    Delay255                ;let IR receiver settle down

Main
                call    LCD_Line1               ;set to first line
                call    String1                 ;display IR title string

                call    ReadIR                  ;read IR signal
                movlw   d'2'
                call    LCD_Line2W              ;set cursor position
                clrf    NumH
                movf    Dev_Byte,       w       ;convert device byte
                movwf   NumL
                call    Convert
                movf    Tens,   w
                call    LCD_CharD
                movf    Ones,   w
                call    LCD_CharD

                movlw   d'11'
                call    LCD_Line2W              ;set cursor position
                clrf    NumH
                movf    Cmd_Byte,       w       ;convert data byte
                movwf   NumL
                call    Convert
                movf    Hund,   w
                call    LCD_CharD
                movf    Tens,   w
                call    LCD_CharD
                movf    Ones,   w
                call    LCD_CharD

                call    ProcKeys                ;do   something   with   commands
received

                goto    Main                    ;loop for ever

ProcKeys
                btfss   Flags2, New
                retlw   0x00                    ;return if not new keypress
                movlw   TV_ID                   ;check for TV ID code
                subwf    Dev_Byte,      w
                btfss    STATUS    , Z
                retlw   0x00                    ;return if not correct code

                movlw   But1                    ;test for button 1
```

```
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                goto    Key1                    ;try  next  key  if  not  correct
code

                movf    OUT_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED             ;and test LED bit for toggling
                bsf     OUT_PORT,       LED     ;turn on LED
                btfsc   tmp3,   LED
                bcf     OUT_PORT,       LED     ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                retlw   0x00

Key1            movlw   But2                    ;test for button 2
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                goto    Key2                    ;try  next  key  if  not  correct
code
                                                ;this time just turn it on
                bsf     OUT_PORT,       LED     ;turn on LED
                bcf     Flags2, New             ;and cancel new flag
                retlw   0x00

Key2            movlw   But3                    ;test for button 3
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                retlw   0x00                    ;return if not correct code
                                                ;this time just turn it off
                bcf     OUT_PORT,       LED     ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                retlw   0x00

String1         clrf    count                   ;set counter register to zero
Mess1           movf    count, w                ;put counter value in W
                call    Xtext                   ;get  a  character  from  the  text
table
                xorlw   0x00                    ;is it a zero?
                btfsc   STATUS, Z
                retlw   0x00                    ;return when finished
                call    LCD_Char
                incf    count, f
                goto    Mess1

;IR routines

ReadIR          call    Read_Pulse
                btfss   Flags,  StartFlag
                goto    ReadIR                  ;wait for start pulse (2.4mS)

Get_Data        movlw   0x07                    ;set up to read 7 bits
                 movwf   Bit_Cntr
                 clrf    Cmd_Byte
Next_RcvBit2     call   Read_Pulse
                btfsc   Flags,  StartFlag       ;abort if another Start bit
                goto    ReadIR
                btfsc   Flags,  ErrFlag         ;abort if error
                goto    ReadIR

                 bcf     STATUS    , C
                 btfss   Flags,        Zero
                 bsf     STATUS    , C
```

```
                rrf     Cmd_Byte  , f
                decfsz  Bit_Cntr  , f
                goto    Next_RcvBit2

                rrf     Cmd_Byte  , f         ;correct  bit  alignment  for  7
bits

Get_Cmd         movlw   0x05                  ;set up to read 5 bits
                movwf   Bit_Cntr
                clrf    Dev_Byte
Next_RcvBit     call    Read_Pulse
                btfsc   Flags,  StartFlag     ;abort if another Start bit
                goto    ReadIR
                btfsc   Flags,  ErrFlag       ;abort if error
                goto    ReadIR

                bcf     STATUS    , C
                btfss   Flags,        Zero
                bsf     STATUS    , C
                rrf     Dev_Byte  , f
                decfsz  Bit_Cntr  , f
                goto    Next_RcvBit

                rrf     Dev_Byte  , f         ;correct  bit  alignment  for  5
bits
                rrf     Dev_Byte  , f
                rrf     Dev_Byte  , f

                retlw   0x00

;end of ReadIR


;read pulse width, return flag for StartFlag, One, Zero, or ErrFlag
;output from IR receiver is normally high, and goes low when signal received

Read_Pulse      clrf    LoX
                btfss   IR_PORT,      IR_In   ;wait until high
                goto    $-1
                clrf    tmp1
                movlw   0xC0                  ;delay to decide new keypress
                movwf   tmp2                  ;for keys that need to toggle

Still_High      btfss   IR_PORT,      IR_In   ;and wait until goes low
                goto    Next
                incfsz  tmp1,f
                goto    Still_High
                incfsz  tmp2,f
                goto    Still_High
                bsf     Flags2, New           ;set  New  flag  if  no  button
pressed
                goto    Still_High

Next            nop
                nop
                nop
                nop
                nop                           ;waste time to scale pulse
                nop                           ;width to 8 bits
                nop
                nop
                nop
```

```
                nop
                nop
                nop
                incf    LoX,    f
                btfss   IR_PORT,       IR_In
                goto    Next                    ;loop until input high again

; test if Zero, One, or Start (or error)

Chk_Pulse       clrf    Flags

TryError        movf    LoX,    w               ; check if pulse too small
                addlw   d'255' - d'20'          ; if LoX <= 20
                btfsc   STATUS    , C
                goto    TryZero
                bsf     Flags,  ErrFlag         ; Error found, set flag
                retlw   0x00

TryZero         movf    LoX,    w               ; check if zero
                addlw   d'255' - d'60'          ; if LoX <= 60
                btfsc   STATUS    , C
                goto    TryOne
                bsf     Flags,  Zero            ; Zero found, set flag
                retlw   0x00

TryOne          movf    LoX,    w               ; check if one
                addlw   d'255' - d'112'         ; if LoX <= 112
                btfsc   STATUS    , C
                goto    TryStart
                bsf     Flags,  One             ; One found, set flag
                retlw   0x00

TryStart        movf    LoX,    w               ; check if start
                addlw   d'255' - d'180'         ; if LoX <= 180
                btfsc   STATUS    , C
                goto    NoMatch
                bsf     Flags,  StartFlag       ; Start pulse found
                retlw   0x00
NoMatch                                         ; pulse too long
                bsf     Flags,  ErrFlag         ; Error found, set flag
                retlw   0x00

;end of pulse measuring routines

;LCD routines

;Initialise LCD
LCD_Init        call    LCD_Busy                ;wait for LCD to settle

                movlw   0x20                    ;Set 4 bit mode
                call    LCD_Cmd

                movlw   0x28                    ;Set display shift
                call    LCD_Cmd

                movlw   0x06                    ;Set display character mode
                call    LCD_Cmd

                movlw   0x0c                    ;Set  display  on/off  and  cursor
command
                call    LCD_Cmd                 ;Set cursor off
```

```
                call    LCD_Clr                 ;clear display

                retlw   0x00

; command set routine
LCD_Cmd         movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bcf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    LCD_Busy
                retlw   0x00

LCD_CharD       addlw   0x30                    ;add 0x30 to convert to ASCII
LCD_Char        movwf   templcd
                swapf   templcd,        w       ;send upper nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high

                movf    templcd,        w       ;send lower nibble
                andlw   0x0f                    ;clear upper 4 bits of W
                movwf   LCD_PORT
                bsf     LCD_PORT, LCD_RS        ;RS line to 1
                call    Pulse_e                 ;Pulse the E line high
                call    LCD_Busy
                retlw   0x00

LCD_Line1       movlw   0x80                    ;move to 1st row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line2       movlw   0xc0                    ;move to 2nd row, first column
                call    LCD_Cmd
                retlw   0x00

LCD_Line1W      addlw   0x80                    ;move to 1st row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_Line2W      addlw   0xc0                    ;move to 2nd row, column W
                call    LCD_Cmd
                retlw   0x00

LCD_CurOn       movlw   0x0d                    ;Set display on/off and cursor
command
                call    LCD_Cmd
                retlw   0x00

LCD_CurOff      movlw   0x0c                    ;Set display on/off and cursor
command
                call    LCD_Cmd
                retlw   0x00
```

```
LCD_Clr         movlw   0x01                    ;Clear display
                call    LCD_Cmd
                retlw   0x00

LCD_HEX         movwf   tmp1
                swapf   tmp1,   w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                movf    tmp1, w
                andlw   0x0f
                call    HEX_Table
                call    LCD_Char
                retlw   0x00

Pulse_e         bsf     LCD_PORT, LCD_E
                nop
                bcf     LCD_PORT, LCD_E
                retlw   0x00

LCD_Busy
                bsf     STATUS, RP0             ;set bank 1
                movlw   0x0f                    ;set Port for input
                movwf   LCD_TRIS
                bcf     STATUS, RP0             ;set bank 0
                bcf     LCD_PORT, LCD_RS        ;set LCD for command mode
                bsf     LCD_PORT, LCD_RW        ;setup to read busy flag
                bsf     LCD_PORT, LCD_E
                swapf   LCD_PORT, w             ;read upper nibble (busy flag)
                bcf     LCD_PORT, LCD_E
                movwf   templcd2
                bsf     LCD_PORT, LCD_E         ;dummy read of lower nibble
                bcf     LCD_PORT, LCD_E
                btfsc   templcd2, 7             ;check busy flag, high = busy
                goto    LCD_Busy                ;if busy check again
                bcf     LCD_PORT, LCD_RW
                bsf     STATUS, RP0             ;set bank 1
                movlw   0x00                    ;set Port for output
                movwf   LCD_TRIS
                bcf     STATUS, RP0             ;set bank 0
                return

;end of LCD routines


;Delay routines

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0         decfsz  counta, f
```

```
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

;end of Delay routines

;This routine downloaded from http://www.piclist.com
Convert:                        ; Takes number in NumH:NumL
                                ; Returns decimal in
                                ; TenK:Thou:Hund:Tens:Ones
        swapf   NumH, w
        iorlw   B'11110000'
        movwf   Thou
        addwf   Thou,f
        addlw   0XE2
        movwf   Hund
        addlw   0X32
        movwf   Ones

        movf    NumH,w
        andlw   0X0F
        addwf   Hund,f
        addwf   Hund,f
        addwf   Ones,f
        addlw   0XE9
        movwf   Tens
        addwf   Tens,f
        addwf   Tens,f

        swapf   NumL,w
        andlw   0X0F
        addwf   Tens,f
        addwf   Ones,f

        rlf     Tens,f
        rlf     Ones,f
        comf    Ones,f
        rlf     Ones,f

        movf    NumL,w
        andlw   0X0F
        addwf   Ones,f
        rlf     Thou,f

        movlw   0X07
        movwf   TenK

                    ; At this point, the original number is
                    ; equal to
                    ; TenK*10000+Thou*1000+Hund*100+Tens*10+Ones
                    ; if those entities are regarded as two's
                    ; complement binary.  To be precise, all of
                    ; them are negative except TenK.  Now the number
                    ; needs to be normalized, but this can all be
                    ; done with simple byte arithmetic.

        movlw   0X0A                            ; Ten
Lb1:
```

```
        addwf   Ones,f
        decf    Tens,f
        btfss   3,0
        goto   Lb1
Lb2:
        addwf   Tens,f
        decf    Hund,f
        btfss   3,0
        goto   Lb2
Lb3:
        addwf   Hund,f
        decf    Thou,f
        btfss   3,0
        goto   Lb3
Lb4:
        addwf   Thou,f
        decf    TenK,f
        btfss   3,0
        goto   Lb4

        retlw  0x00


               end
```

Tutorial 5.2 - requires one Main Board, one IR Board and Switch Board.

This program implements a Sony SIRC IR transmitter, pressing one of the four buttons sends the corresponding code, you can alter the codes as you wish, for this example I chose Volume Up and Down, and Program Up and Down. In order to use this with the LED switching above, I would suggest setting the buttons to transmit '1', '2', '3' and '4', where '4' should have no effect on the LED - the codes are 0x00, 0x01, 0x02, 0x03 respectively (just to confuse us, the number keys start from zero, not from one).

```
;Tutorial 5.2 - Nigel Goodwin 2002
;Sony SIRC IR transmitter
       LIST    p=16F628              ;tell assembler what chip we are using
       include "P16F628.inc"         ;include the defaults for the chip
       __config 0x3D18               ;sets   the   configuration   settings
(oscillator type etc.)

       cblock  0x20                  ;start of general purpose registers
               count1                ;used in delay routine
               counta                ;used in delay routine
               countb
               count
               Delay_Count
               Bit_Cntr
               Data_Byte
               Dev_Byte
               Rcv_Byte
               Pulse
       endc


IR_PORT Equ     PORTB
IR_TRIS Equ     TRISB
IR_Out  Equ     0x01
IR_In   Equ     0x02
```

```
Ser_Out Equ     0x01
Ser_In  Equ     0x02
SW1     Equ     7                       ;set constants for the switches
SW2     Equ     6
SW3     Equ     5
SW4     Equ     4

TV_ID           Equ     0x01                    ;TV device ID

But1            Equ     0x00                    ;numeric button ID's
But2            Equ     0x01
But3            Equ     0x02
But4            Equ     0x03
But5            Equ     0x04
But6            Equ     0x05
But7            Equ     0x06
But8            Equ     0x07
But9            Equ     0x08
ProgUp          Equ     d'16'
ProgDn          Equ     d'17'
VolUp           Equ     d'18'
VolDn           Equ     d'19'


        org     0x0000                  ;org sets the origin, 0x0000 for the
16F628,
        goto    Start                   ;this is where the program starts
running

        org     0x005

Start           movlw   0x07
                movwf   CMCON                   ;turn comparators off (make it
like a 16F84)

                clrf    IR_PORT                 ;make PortB outputs low


                bsf     STATUS,         RP0     ;select bank 1
                movlw   b'11111101'             ;set PortB all inputs, except
RB1
                movwf   IR_TRIS
                movlw   0xff
                movwf   PORTA
                bcf     STATUS,         RP0     ;select bank 0

Read_Sw
                btfss   PORTA,  SW1
                call    Switch1
                btfss   PORTA,  SW2
                call    Switch2
                btfss   PORTA,  SW3
                call    Switch3
                btfss   PORTA,  SW4
                call    Switch4
                call    Delay27
                goto    Read_Sw

Switch1         movlw   ProgUp
                call    Xmit_RS232
                retlw   0x00
```

```
Switch2         movlw   ProgDn
                call    Xmit_RS232
                retlw   0x00

Switch3         movlw   VolUp
                call    Xmit_RS232
                retlw   0x00

Switch4         movlw   VolDn
                call    Xmit_RS232
                retlw   0x00

TX_Start        movlw   d'92'
                call    IR_pulse
                movlw   d'23'
                call    NO_pulse
                retlw   0x00

TX_One          movlw   d'46'
                call    IR_pulse
                movlw   d'23'
                call    NO_pulse
                retlw   0x00

TX_Zero         movlw   d'23'
                call    IR_pulse
                movlw   d'23'
                call    NO_pulse
                retlw   0x00

IR_pulse
                MOVWF   count           ;  Pulses the IR led at 38KHz
irloop          BSF     IR_PORT,        IR_Out
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                BCF     IR_PORT,        IR_Out
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP
                NOP
                NOP                     ;
                NOP                     ;
                DECFSZ  count,F
                GOTO    irloop
                RETLW   0

NO_pulse
                MOVWF   count           ;  Doesn't pulse the IR led
irloop2         BCF     IR_PORT,        IR_Out
```

```
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                BCF     IR_PORT,        IR_Out
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP                     ;
                NOP
                NOP
                NOP                     ;
                NOP                     ;
                DECFSZ  count,F
                GOTO    irloop2
                RETLW   0

Xmit_RS232      MOVWF   Data_Byte               ;move W to Data_Byte
                MOVLW   0x07                    ;set 7 DATA bits out
                MOVWF   Bit_Cntr
                call    TX_Start                ;send start bit
Ser_Loop        RRF     Data_Byte , f           ;send one bit
                BTFSC   STATUS    , C
                call    TX_One
                BTFSS   STATUS    , C
                call    TX_Zero
                DECFSZ  Bit_Cntr  , f           ;test if all done
                GOTO    Ser_Loop
                                                ;now send device data
                movlw   D'1'
                movwf   Dev_Byte                ;set device to TV
                MOVLW   0x05                    ;set 5 device bits out
                MOVWF   Bit_Cntr
Ser_Loop2       RRF     Dev_Byte , f            ;send one bit
                BTFSC   STATUS    , C
                call    TX_One
                BTFSS   STATUS    , C
                call    TX_Zero
                DECFSZ  Bit_Cntr  , f           ;test if all done
                GOTO    Ser_Loop2
                retlw   0x00



;Delay routines

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay27         movlw   d'27'           ;delay 27mS
```

```
                goto    d0
Delay20         movlw   d'20'            ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0         decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

;end of Delay routines

        end
```

Tutorial 5.3 - requires one Main Board, one IR Board and LED Board.

This program implements toggling the 8 LED's on the LED board with the buttons 1 to 8 on a Sony TV remote control, you can easily change the device ID and keys used for the LED's. I've also used a (so far unused) feature of the 16F628, the EEPROM data memory - by using this the program remembers the previous settings when unplugged - when you reconnect the power it restores the last settings by reading them from the internal non-volatile memory. The 16F628 provides 128 bytes of this memory, we only use one here (address 0x00, set in the EEPROM_Addr constant).

```
;Tutorial 5_3
;Read SIRC IR and toggle LED display, save settings in EEPROM data memory.
;Nigel Goodwin 2002

        LIST    p=16F628                ;tell assembler what chip we are using
        include "P16F628.inc"           ;include the defaults for the chip
        ERRORLEVEL      0,      -302    ;suppress bank selection messages
        __config 0x3D18                 ;sets   the   configuration   settings
(oscillator type etc.)



                cblock  0x20                    ;start   of   general   purpose
registers
                        count                   ;used in looping routines
                        count1                  ;used in delay routine
                        counta                  ;used in delay routine
                        countb                  ;used in delay routine
                        LoX
                        Bit_Cntr
                        Cmd_Byte
                        Dev_Byte
                        Flags
                        Flags2
                        tmp1                    ;temporary storage
                        tmp2
```

```
                tmp3
                lastdev
                lastkey

            endc

LED_PORT    Equ     PORTB
LED_TRIS    Equ     TRISB

IR_PORT     Equ     PORTA
IR_TRIS     Equ     TRISA
IR_In       Equ     0x02                ;input assignment for IR data

OUT_PORT    Equ     PORTB
LED0        Equ     0x00
LED1        Equ     0x01
LED2        Equ     0x02
LED3        Equ     0x03
LED4        Equ     0x04
LED5        Equ     0x05
LED6        Equ     0x06
LED7        Equ     0x07

EEPROM_Addr Equ     0x00                ;address of EEPROM byte used

ErrFlag     Equ     0x00
StartFlag   Equ     0x01                ;flags used for received bit
One         Equ     0x02
Zero        Equ     0x03

New         Equ     0x07                ;flag used to show key released

TV_ID       Equ     0x01                ;TV device ID

But1        Equ     0x00                ;numeric button ID's
But2        Equ     0x01
But3        Equ     0x02
But4        Equ     0x03
But5        Equ     0x04
But6        Equ     0x05
But7        Equ     0x06
But8        Equ     0x07
But9        Equ     0x08

            org     0x0000
            goto    Start

            org     0x0004
            retfie

Start       movlw   0x07
            movwf   CMCON               ;turn comparators off (make it
like a 16F84)

Initialise  clrf    count
            clrf    PORTA
            clrf    PORTB
            clrf    Flags
             clrf    Dev_Byte
            clrf    Cmd_Byte
```

```
SetPorts        bsf     STATUS,         RP0     ;select bank 1
                movlw   0x00                    ;make all LED pins outputs
                movwf   LED_TRIS
                movlw   b'11111111'             ;make all IR port pins inputs
                movwf   IR_TRIS
                bcf     STATUS,         RP0     ;select bank 0

                call    EE_Read                 ;restore previous settings

Main
                call    ReadIR                  ;read IR signal
                call    ProcKeys                ;do   something   with   commands
received

                goto    Main                    ;loop for ever

ProcKeys
                btfss   Flags2, New
                retlw   0x00                    ;return if not new keypress
                movlw   TV_ID                   ;check for TV ID code
                subwf    Dev_Byte,      w
                btfss    STATUS    , Z
                retlw   0x00                    ;return if not correct code

                movlw   But1                    ;test for button 1
                subwf    Cmd_Byte, w
                btfss    STATUS    , Z
                goto    Key1                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED0            ;and test LED bit for toggling
                bsf     LED_PORT,       LED0    ;turn on LED
                btfsc   tmp3,   LED0
                bcf     LED_PORT,       LED0    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key1            movlw   But2                    ;test for button 1
                subwf    Cmd_Byte, w
                btfss    STATUS    , Z
                goto    Key2                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED1            ;and test LED bit for toggling
                bsf     LED_PORT,       LED1    ;turn on LED
                btfsc   tmp3,   LED1
                bcf     LED_PORT,       LED1    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key2            movlw   But3                    ;test for button 1
                subwf    Cmd_Byte, w
                btfss    STATUS    , Z
                goto    Key3                    ;try  next  key  if  not  correct
code
```

```
                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED2            ;and test LED bit for toggling
                bsf     LED_PORT,       LED2    ;turn on LED
                btfsc   tmp3,   LED2
                bcf     LED_PORT,       LED2    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key3            movlw   But4                    ;test for button 1
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                goto    Key4                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED3            ;and test LED bit for toggling
                bsf     LED_PORT,       LED3    ;turn on LED
                btfsc   tmp3,   LED3
                bcf     LED_PORT,       LED3    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key4            movlw   But5                    ;test for button 1
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                goto    Key5                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED4            ;and test LED bit for toggling
                bsf     LED_PORT,       LED4    ;turn on LED
                btfsc   tmp3,   LED4
                bcf     LED_PORT,       LED4    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key5            movlw   But6                    ;test for button 1
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                goto    Key6                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED5            ;and test LED bit for toggling
                bsf     LED_PORT,       LED5    ;turn on LED
                btfsc   tmp3,   LED5
                bcf     LED_PORT,       LED5    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key6            movlw   But7                    ;test for button 1
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
```

```
                goto    Key7                    ;try  next  key  if  not  correct
code

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED6            ;and test LED bit for toggling
                bsf     LED_PORT,       LED6    ;turn on LED
                btfsc   tmp3,   LED6
                bcf     LED_PORT,       LED6    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

Key7            movlw   But8                    ;test for button 1
                subwf   Cmd_Byte, w
                btfss   STATUS    , Z
                retlw   0X00

                movf    LED_PORT,       w       ;read PORTB (for LED status)
                movwf   tmp3                    ;and store in temp register
                btfss   tmp3,   LED7            ;and test LED bit for toggling
                bsf     LED_PORT,       LED7    ;turn on LED
                btfsc   tmp3,   LED7
                bcf     LED_PORT,       LED7    ;turn off LED
                bcf     Flags2, New             ;and cancel new flag
                call    EE_Write                ;save the settings
                retlw   0x00

EE_Read         bsf     STATUS, RP0             ; Bank 1
                movlw   EEPROM_Addr
                movwf   EEADR                   ; Address to read
                bsf     EECON1, RD              ; EE Read
                movf    EEDATA, W               ; W = EEDATA
                bcf     STATUS, RP0             ; Bank 0
                movwf   LED_PORT                ; restore previous value
                retlw   0x00

EE_Write        movf    LED_PORT,       w       ; read current value
                bsf     STATUS, RP0             ; Bank 1
                bsf     EECON1, WREN            ; Enable write
                movwf   EEDATA                  ; set EEPROM data
                movlw   EEPROM_Addr
                movwf   EEADR                   ; set EEPROM address
                movlw   0x55
                movwf   EECON2                  ; Write 55h
                movlw   0xAA
                movwf   EECON2                  ; Write AAh
                bsf     EECON1, WR              ; Set WR bit
                                                ; begin write
                bcf     STATUS, RP0             ; Bank 0

                btfss   PIR1,   EEIF            ; wait for write to complete.
                goto    $-1
                bcf     PIR1,   EEIF            ;   and   clear   the   'write
complete' flag
                bsf     STATUS, RP0             ; Bank 1
                bcf     EECON1, WREN            ; Disable write
                bcf     STATUS, RP0             ; Bank 0
                retlw   0x00
```

```
;IR routines

ReadIR          call    Read_Pulse
                btfss   Flags,  StartFlag
                goto    ReadIR                  ;wait for start pulse (2.4mS)

Get_Data        movlw   0x07                    ;set up to read 7 bits
                 movwf   Bit_Cntr
                 clrf    Cmd_Byte
Next_RcvBit2     call   Read_Pulse
                btfsc   Flags,  StartFlag       ;abort if another Start bit
                goto    ReadIR
                btfsc   Flags,  ErrFlag         ;abort if error
                goto    ReadIR

                 bcf     STATUS    , C
                 btfss   Flags,        Zero
                 bsf     STATUS    , C
                 rrf     Cmd_Byte  , f
                 decfsz  Bit_Cntr  , f
                 goto    Next_RcvBit2

                 rrf     Cmd_Byte  , f          ;correct  bit  alignment  for  7
bits

Get_Cmd         movlw   0x05                    ;set up to read 5 bits
                 movwf   Bit_Cntr
                 clrf    Dev_Byte
Next_RcvBit      call   Read_Pulse
                btfsc   Flags,  StartFlag       ;abort if another Start bit
                goto    ReadIR
                btfsc   Flags,  ErrFlag         ;abort if error
                goto    ReadIR

                 bcf     STATUS    , C
                 btfss   Flags,        Zero
                 bsf     STATUS    , C
                 rrf     Dev_Byte  , f
                 decfsz  Bit_Cntr  , f
                 goto    Next_RcvBit

                 rrf     Dev_Byte  , f          ;correct  bit  alignment  for  5
bits
                 rrf     Dev_Byte  , f
                 rrf     Dev_Byte  , f

                retlw   0x00

;end of ReadIR


;read pulse width, return flag for StartFlag, One, Zero, or ErrFlag
;output from IR receiver is normally high, and goes low when signal received

Read_Pulse       clrf   LoX
                btfss   IR_PORT,       IR_In   ;wait until high
                goto    $-1
                clrf    tmp1
                movlw   0xC0                    ;delay to decide new keypress
                movwf   tmp2                    ;for keys that need to toggle

Still_High      btfss   IR_PORT,       IR_In   ;and wait until goes low
```

```
                goto    Next
                incfsz  tmp1,f
                goto    Still_High
                incfsz  tmp2,f
                goto    Still_High
                bsf     Flags2, New             ;set  New  flag  if  no  button
pressed
                goto    Still_High

Next            nop
                nop
                nop
                nop
                nop                             ;waste time to scale pulse
                nop                             ;width to 8 bits
                nop
                nop
                nop
                nop
                nop
                nop
                incf    LoX,    f
                btfss   IR_PORT,        IR_In
                goto    Next                    ;loop until input high again

; test if Zero, One, or Start (or error)

Chk_Pulse       clrf    Flags

TryError        movf    LoX,    w               ; check if pulse too small
                addlw   d'255' - d'20'          ; if LoX <= 20
                btfsc   STATUS    , C
                goto    TryZero
                bsf     Flags,  ErrFlag         ; Error found, set flag
                retlw   0x00

TryZero         movf    LoX,    w               ; check if zero
                addlw   d'255' - d'60'          ; if LoX <= 60
                btfsc   STATUS    , C
                goto    TryOne
                bsf     Flags,  Zero            ; Zero found, set flag
                retlw   0x00

TryOne          movf    LoX,    w               ; check if one
                addlw   d'255' - d'112'         ; if LoX <= 112
                btfsc   STATUS    , C
                goto    TryStart
                bsf     Flags,  One             ; One found, set flag
                retlw   0x00

TryStart        movf    LoX,    w               ; check if start
                addlw   d'255' - d'180'         ; if LoX <= 180
                btfsc   STATUS    , C
                goto    NoMatch
                bsf     Flags,  StartFlag       ; Start pulse found
                retlw   0x00
NoMatch                                         ; pulse too long
                bsf     Flags,  ErrFlag         ; Error found, set flag
                retlw   0x00

;end of pulse measuring routines
```

```
;Delay routines

Delay255        movlw   0xff            ;delay 255 mS
                goto    d0
Delay100        movlw   d'100'          ;delay 100mS
                goto    d0
Delay50         movlw   d'50'           ;delay 50mS
                goto    d0
Delay20         movlw   d'20'           ;delay 20mS
                goto    d0
Delay5          movlw   0x05            ;delay 5.000 ms (4 MHz clock)
d0              movwf   count1
d1              movlw   0xC7
                movwf   counta
                movlw   0x01
                movwf   countb
Delay_0         decfsz  counta, f
                goto    $+2
                decfsz  countb, f
                goto    Delay_0

                decfsz  count1  ,f
                goto    d1
                retlw   0x00

;end of Delay routines


                end
```

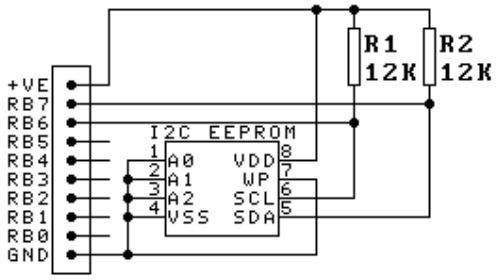
The EEPROM data is accessed by two new routines, EE_Read and EE_Write, the EE_Read routine is called as the program powers up, before we enter the main loop, and the EE_Write routine is called after every LED change. The EE_Read routine is very straightforward, we simply set the address we wish to read in the EEADR register, set the RD flag in the EECON1 register, and then read the data from the EEDATA register. Writing is somewhat more complicated, for a couple of reasons:

1. Microchip have taken great care to prevent accidental or spurious writes to the data EEPROM. In order to write to it we first have to set the 'Write Enable' bit in the EECON1 register, and then make two specific writes (0x55 and 0xAA) to the EECON2 register, only then can we set the WR bit in EECON1 and start the actual writing. One of the most common problems in domestic electronics today is data EEPROM corruption, hopefully the efforts of Microchip will prevent similar problems with the 16F628.
2. Writing to EEPROM takes time, so we have to wait until the 'Write Complete' flag is set, it doesn't really matter in this application as the time spent waiting for the next IR command gives more than enough time to write to the data EEPROM, but it's good practice to do it anyway.

The extra work involved makes the EE_Write routine a lot longer than the EE_Read routine, it also doesn't help that we need to access registers in different banks, so we do a fair bit of bank switching.

PIC Tutorial Six - I2C EEPROM Programming

I2C EEPROM Board



This is the I2C (or IIC) Board, it stands for 'Inter I/C Communications' and is a standard two wire bidirectional bus used for communicating between chips in most modern electronic equipment - for instance in a modern TV receiver almost everything is controlled via I2C, and all the settings are usually stored in a small 8 pin EEPROM (Electrically Erasable Read Only Memory). This project uses a standard EEPROM, and can be various types, which will all drop in the socket. The two signals to the chip are called SDA (Serial DatA) and SCL (Serial CLock), and are open-collector outputs (like the infamous RA4), but we can easily simulate that in software - and in any case, we don't need to do so for the SCL signal for a 'single master system', which is what we will be using. As we are only using a 'single master' system, resistor R1 isn't really needed, but I've included it as a matter of form.

I haven't labelled the EEPROM chip, as it can be a number of different ones, basically the 24C02 (256 bytes), 24C04 (512 bytes), 24C08 (1024 bytes) or 24C16 (2048 bytes), these all use 'standard addressing', you can also use a 24C32 (4096 bytes) or 24LC64 (8192 bytes) which use 'extended addressing' to access the greater memory. On the smaller chips memory is in 256 byte 'pages', with standard addressing only allowing 8 pages. The page addressing is also related to the address lines A0, A1 & A2, the 24C02 uses all three lines, so you can have eight connected to the bus with the addresses set differently, the 24C04 only uses two address lines, allowing 4 chips to be connected. The 24C16 uses none of the address lines, it already has all eight pages internally, so only one 24C16 can be connected to the same I2C bus. So 'standard addressing' only allows 2048 bytes to be addressed, in eight 256 byte 'pages'. The 'extended addressing' mode on the larger chips uses two eight bit address registers (giving a possible 65,536 bytes per chip), plus the same three bit page/address allocation, so you can have up to eight of these larger chips on the same bus. A 24LC256 is also available, which gives 32,768 bytes of space, and you can also address 8 of these on the same I2C bus, giving 262,144 bytes of memory storage which would make a very useful data-logger (over 72 hours at one sample per second!).

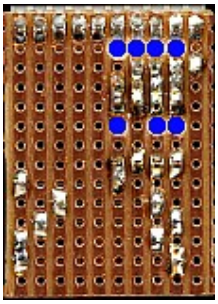
ChipType				Address			
1	0	1	0	x	x	x	R/W

Each I2C chip type has it's own individual 4 bit chip address, for EEPROM memory chips it's '1010', this is combined with a 3 bit address from the address input pins to give a 7 bit unique chip address - this is then made into an 8 bit word by the addition of a R/W bit, '1' for read' and '0' for write, and this is the complete

value written as the 'Slave Address' for the chip. Each chip on the bus must have a unique address or problems are going to occur.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.

This is the top view of the I2C EEPROM Board, it has 7 wire links.



The bottom of the I2C EEPROM Board, there are 7 track cuts, please note that there are only 3 between the I/C pins, one isn't cut as it's used to ground the WP pin.

These tutorials require the Main Board, the LCD Board, and various of the I2C Boards, as written the tutorials use the LCD Board on PortA and the I2C Boards on PortB - although these could easily be swapped over, as the I2C Boards don't use either of the two 'difficult' pins for PortA, pins 4 and 5, as outputs. Download zipped tutorial files.

As with the LCD Tutorial, the idea is to implement a reusable set of I2C routines.

Rather than showing the routines on the page as with earlier tutorials (they are getting quite lengthy now), I'm only going to store them in the pages download ZIP file so you will need to download them. As the I2C tutorials use a number of different boards, each section is headed by the I2C boards required in **bold type**.

I2C is a protocol designed by Philips Semiconductors, and is for communications between I/C's over a two wire synchronous serial bus, devices are classed as either 'Master' or 'Slave', for our purposes the Main Board processor is the 'Master', and any other devices are 'Slaves'. The initial tutorials use a 24C04, a 512 byte EEPROM memory chip, commonly used for storing the settings in modern TV's and VCR's, where they are used to store all the customer settings (tuning, volume, brightness etc.) and the internal calibration values for the set, which are normally accessed through a special 'service mode'. These chips provide non-volatile memory as a series of 256 byte 'pages', so the 24C04 provides two 'pages' giving 512 bytes of memory. The 24C02 uses 'standard addressing', which gives the 256 byte page limit, other larger chips use 'extended addressing', giving a possible 16 bit address space, I've used a 24C256 which uses a 15 bit address space, giving 32,768 of memory space. To address these you require two bytes of address data, the programs already include these (but commented out), I've uncommented them

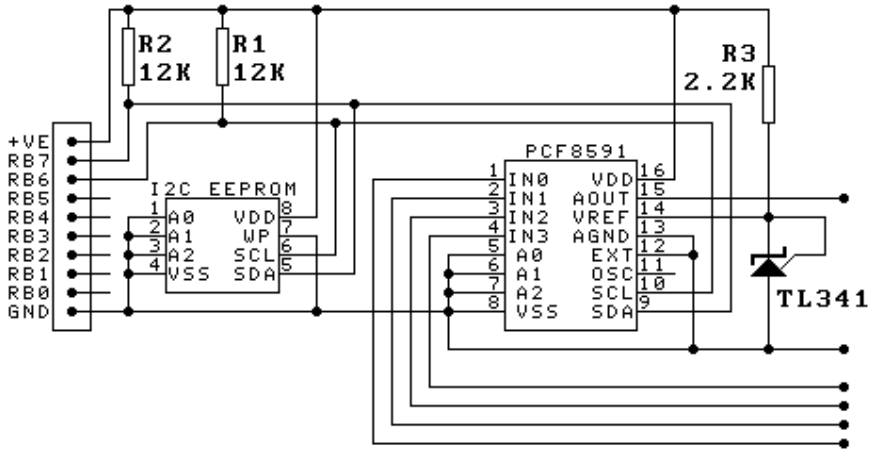
for copies of the first two tutorials and called them 6_1a and 6_2a, if you want to use an EEPROM larger than a 24C16 you will need to use these extended addressing versions.

I2C EEPROM Board

The first tutorial writes sequential numbers through one entire page of memory, and then reads them back, 4 bytes at a time, displaying them on the LCD, separated by about half a second between updates. The second tutorial demonstrates 'sequential writing', the first tutorial uses 'byte writing' (which is why it displays 'Writing..' for a couple of seconds) - a write is fairly slow to EEPROM, taking around 10mS to complete - 'sequential writing' allows you to store a number of bytes in RAM inside the EEPROM chip (a maximum of 8 for the 24C04) and then write them all to EEPROM with a single write delay. Tutorial 2 writes 4 bytes at once, to demonstrate how this is done. The third tutorial is simply a cut-down version of the first, I've included it as a useful tool, it simply reads one page of the EEPROM and displays it as tutorial 1 does - useful for checking the contents of an EEPROM. You can deal with the EEPROM write delay in a couple of ways, firstly you can introduce a software delay, and this option is included in the tutorials (but commented out), or you can keep checking until the chip is ready, this is the method I've used in these tutorials, although if you want you can comment that line out and un-comment the 'call delay10' line instead.

PIC Tutorial - I2C A2D Board

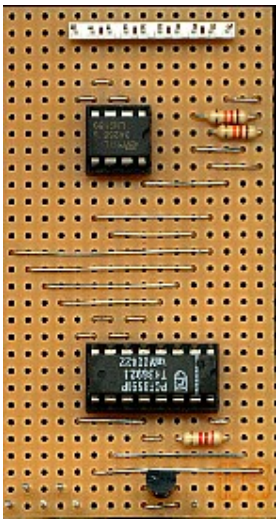
I2C A2D Board



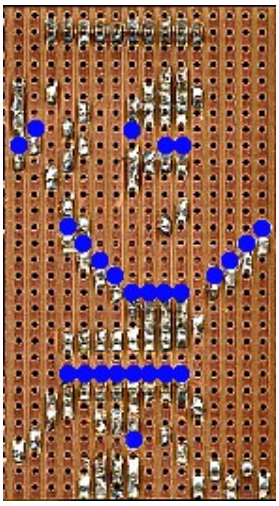
This is the I2C A2D (Analogue to Digital converter) Board, it uses a Philips PCF8591P, which is an I2C chip providing 4 analogue inputs, and 1 analogue output, all having 8 bit resolution. There are actually very few support components required, I've chosen to use an external 2.5V precision voltage reference, which feeds in at pin 14, but this could be simply connected to the 5V rail - though it would be less accurate. By using the 2.5V reference we set the range of the conversion from 0-2.5V, however this can easily be scaled by feeding from a suitable attenuator.

Notice the circuit also shows an EEPROM, the idea being to give the option of storing samples in it's non-volatile memory, and I'll be using a 24C256 to give 32,768 bytes of storage. Notice both chips connect to the same port pins via the I2C bus - by having different chip addresses we can address either one independently.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.



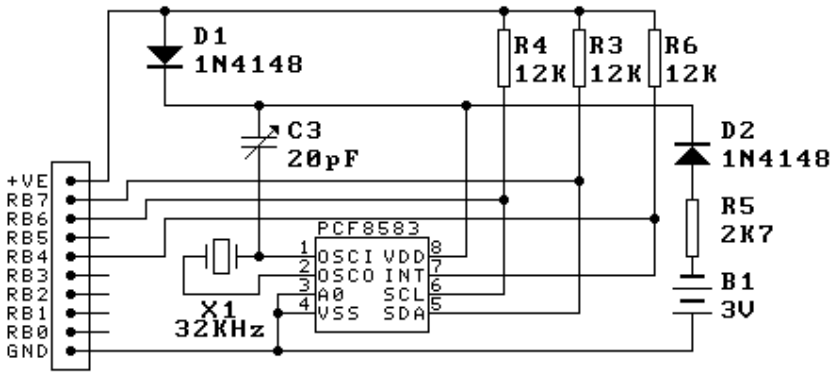
This is the top view of the I2C A2D Board, there are 23 wire links.



The bottom of the I2C A2D Board, there are 26 track cuts.

PIC Tutorial - I2C Clock Board

I2C Clock Board

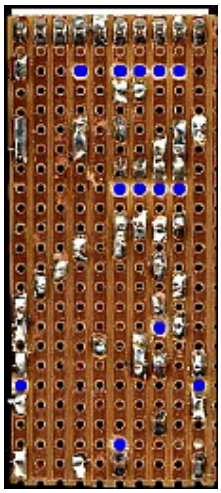


This is the I2C Clock Board, it uses a PCF8583P, which is a real time, battery backed, CMOS I2C clock chip in an 8 pin DIL package. The actual chip I'm using here (as shown in the picture) is labelled 'Intersil 7313', and came from a Grundig VS920 video recorder, but it's pin compatible with the original Philips chip (which is what's actually listed on the circuit). Notice that this chip only has one address line, so can only be mapped as either page 0, or page 1.

The circuit is very similar to the previous I2C EEPROM board, with a few additions, a 32KHz clock crystal and trimmer (using two of the previous address lines), an extra alarm output complete with 12K pull-up resistor (connected to RB4), and components for the battery backup circuit (D1 and D2 are isolating diodes). When the board is powered up the chip is supplied with 5V through D1 (which drops 0.7V leaving 4.3V on the chip), D2 is reverse biased and passes no current. When the board isn't powered, D2 passes current from the battery (only around 2uA, giving a long battery life) to the chip, and D1 is reverse biased, isolated the rest of the circuit. The 3V battery shown is a lithium disk type, and usually lasts around 5 years in the Grundig VCR's that use this same chip. The trimmer is for setting the accuracy of the clock, and if accurately adjusted should keep good time.

Although it's labelled as connecting to PortB, as with most of the boards, it can also be connected to PortA if required.

This is the top view of the I2C Clock Board, it has 7 wire links.



The bottom of the I2C Clock Board, there are 13 track cuts.

I2C Clock Board, and I2C Switch Board

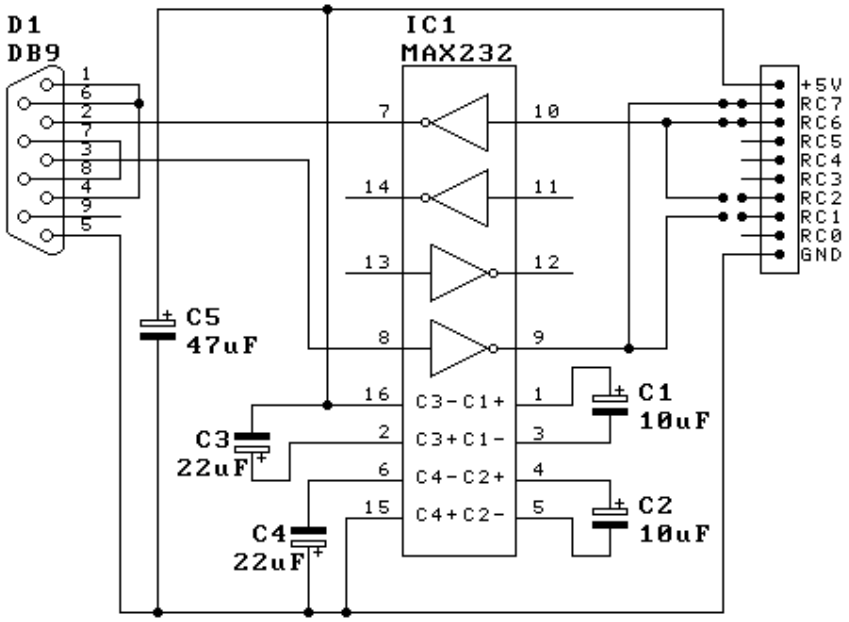
Now we move onto the I2C Clock board, basically we use exactly the same I2C routines, the only difference being in the way we manipulate the data, we need to read the clock registers from the chip (using a sequential read), apply a little processing, and then display them on the LCD. Actually setting the clock is somewhat more complicated, and the biggest difference is the routines for reading the switch board, and setting the clock chip values - which are then written back to the chip with a sequential write. The four buttons used are (from left to right), 'Set', 'Up', 'Down', and 'Next' - in the initial display mode the only button which has an effect is the 'Set' button, this jumps to the 'Clock Set' mode, and starts a flashing cursor on the tens of hours. From this point all four buttons work, pressing 'Set' again will return to display mode, updating the clock values (and zeroing the seconds). Pressing 'Up' will increase the value under the cursor, and 'Down' will decrease the value, with '0' being the lower limit, and '9' being the upper one - I don't currently take account of the different maximum values for particular digits (i.e. tens of hours doesn't go higher than 2), but rely on setting them sensibly. The 'Next' button moves on to the next digit, and if pressed while on the last digit (years units) will return to display mode, just like pressing the 'Set' button. I also don't currently take any account of the correct years, the PCF8583 only provides 0-3 for the years, with 0 being a leap year - extra software routines will be required to do this, with the actual values stored in spare PCF8583 EEPROM memory, and updated when the year changes (remembering that the year might change while the processor is powered down, and the clock is running on it's back-up battery).

I2C A2D Board, and I2C Switch Board

Again, the A2D board uses the same basic I2C routines as before (but with a different chip address for the PCF8591) as with the I2C Clock Board the differences come in the manipulation of the data. As the board also includes an EEPROM socket this can be used to store samples from the A2D chip - with a single 24C256 we can store up to 32,768 eight bit samples - this introduces a slight 'snag', the 24C256 uses 'extended addressing', while the PCF8591 only uses 'standard addressing', however we can still use the same I2C routines by using a flag to tell the routines which addressing mode to use, simply switching the flag for the different chips - this flag switching becomes part of the reusable I2C routines.

PIC Tutorial Seven - RS232

RS232 Board

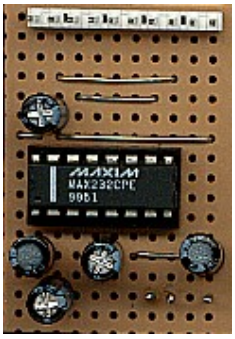


This is the RS232 board, it uses a MAX232 5V to RS232 converter chip, this converts the 0-5V TTL levels at the PIC pins to the +12V/-12V levels used in RS232 links. As is common with these devices it inverts the data during the conversion, the PIC USART hardware is designed to take account of this - but for software serial communications you need to make sure that you invert both the incoming and outgoing data bits.

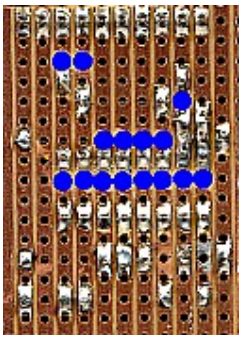
The two closed links on the RC7 and RC6 lines are for connection to the 16F876 board (the 16F876 uses RC6 and RC7 for it's USART connection), and are the two top wire links shown on the top view of the board below. The two open links on the RC1 and RC2 lines are for the 16F628 board (the 16F628 uses RB1 and RB2 for it's USART connection), and are the two top track breaks shown on the bottom view of the board below.

So, for use with the 16F876 board fit the top two wire links, and cut the top two tracks shown, for the 16F628 leave the top two links out, and don't cut the two top track breaks. This only applies if you are using the hardware USART, for software serial communications you can use any pins you like.

Although it's labelled as connecting to PortC for the 16F876 processor board (and is also designed to connect to PortB for the 16F628 processor board), as with most of the boards, it can also be connected to other ports if required, and if not using the hardware USART.



This is the top view of the RS232 Board, there are five wire links, the three veropins at the bottom right are the connections to the 9 pin D socket. As it's not too clear, pin one of the chip is at the left hand side of the board.



The bottom of the RS232 Board, it has fifteen track breaks, marked with blue circles (as usual).

For these tutorials you require the Main Board, Main Board 2, LCD Board, Serial Board, LED Board and switch board. Download zipped tutorial files, a number of examples for the 16F876 based Main Board 2 are provided, these have an 'a' at the end of the filename - the rest are left for the user to convert as an exercise.

RS232 is an asynchronous serial communications protocol, widely used on computers. Asynchronous means it doesn't have any separate synchronising clock signal, so it has to synchronise itself to the incoming data - it does this by the use of 'START' and 'STOP' pulses. The signal itself is slightly unusual for computers, as rather than the normal 0V to 5V range, it uses +12V to -12V - this is done to improve reliability, and greatly increases the available range it can work over - it isn't necessary to provide this exact voltage swing, and you can actually use the PIC's 0V to 5V voltage swing with a couple of resistors to make a simple RS232 interface which will usually work well, but isn't guaranteed to work with all serial ports. For this reason I've designed the Serial Board to use the MAX232 chip, this is a chip specially designed to interface between 5V logic levels and the +12V/-12V of RS232 - it generates the +12V/-12V internally using capacitor charge pumps, and includes four converters, two transmit and two receive, the Serial Board only makes use of one of each - the other two are clearly marked on the circuit, and can be used for something else if required.

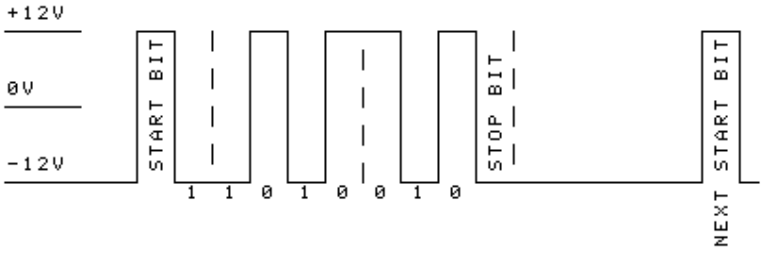
There are various data types and speeds used for RS232, I'm going to concentrate on the most common type in use, known as 8N1 - the 8 signifies '8 Data Bits', the N signifies 'No Parity' (can also be E 'Even Parity' or O 'Odd Parity'), the final 1 signifies '1 Stop Bit'. The total data sent consists of 1 start bit, 8 data bits, and 1 stop bit - giving a total of 10 bits. For the speed, I'm going to concentrate on 9600BPS (Bits Per Second), as each byte sent has 10 bits this means we can transfer a maximum of 960 bytes of data per second - this is fairly fast, but pretty easy to do in software, it's easily modified if you need faster or slower speeds, all you need to do is alter the delay timings - but I find 9600BPS is a pretty good speed to use.

We now know that we will be sending or receiving 960 ten bit data bytes per second, from that it's simple to calculate how long each bit is - simply divide 1 second by 9600 - this gives

104uS per bit. This value is crucial to successful RS232 communication, it doesn't have to be exact as the stop pulse allows resynchronisation after each data byte, but it must be accurate enough to maintain reading in the correct bit throughout each byte. The data is sent low bit first, so the example in the diagrams below is sending '01001011 Binary', '4B Hex', '75 Decimal'.

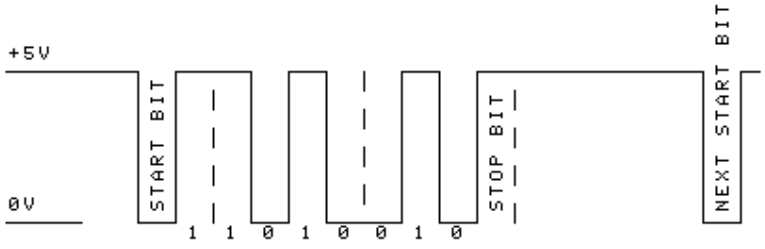
OK, now we know all the details of the protocol we are using, I'll explain how we transmit a byte:

1. The RS232 signal needs to be in the 'STOP CONDITION', at -12V, as the MAX232 inverts (a '1' is -12V and a '0' +12V) we need to make sure the PIC output pin is set HIGH, this should be done in the initialisation section of the program - this pin should always be high, **EXCEPT** when we are sending data, when it can be either high or low.
2. The RS232 line is now happily sat at -12V, and the receiving device is waiting for a 'START BIT', to generate this all we need to do is set the PIC output pin low, the MAX232 inverts the signal and takes the RS232 line up to +12V. As we know that all bits should be 104uS long we now delay 104uS, before we do anything else.
3. Now we can transmit the 8 data bytes, starting with the low bit, after each bit is set on the output pin we again wait 104uS, so each bit is the correct length.
4. That only leaves the 'STOP BIT', for this we set the PIC output pin HIGH (as in section 1 above), and wait 104uS to give the correct bit length - now the 'STOP BIT' doesn't have to be only 104uS long, it simply signifies the end of the data byte. If it is the last data byte it could be a considerable time before another 'START BIT' is sent - this is shown in the diagrams by the large gap between the end of the 'STOP BIT' (shown by the dotted line) and the next 'START BIT'. If you are sending data as fast as possible the next 'START BIT' will start on that dotted line, immediately after the 104uS 'STOP BIT'.



This is an example of a signal on an RS232 line, initially it sits at -12V, known as the 'STOP CONDITION', this condition lasts until a signal is sent. To send a signal we first need to let the receiving device know we are starting to send data, to do this we set the line to +12V, this is called the 'START BIT' - the receiving device is waiting for this to happen, once it does it then gets ready to read the next 9 bits of data (eight data bits and one stop bit).

This is the identical signal as it leaves (or enters) the PIC pin, as the MAX232 inverts the signal this looks to be inverted, but is actually the correct way up - RS232 logic levels are inverted compared to normal levels.



To receive a data byte is pretty straightforward as well:

1. Test the PIC input pin, and loop until it goes low, signifying the beginning of the 'START BIT'.
2. Now we wait just half a bit time (52uS) and check again to make sure it's still low - this 52uS delay means we are reading the 'START BIT' pretty well in the centre of the pulse, where it should be the most reliable.
3. Following a successful 'START BIT' we can now read the data bits, as we are currently in the centre of the 'START BIT' we can simply wait 104uS, which will take us to the centre of the first data bit, then read the input pin again, remembering to invert the polarity of the bit. We then read the next seven bits in the same way, waiting 104uS before each one.
4. Lastly we need to account for the 'STOP BIT', again we wait 104uS for the centre of the bit and could read the port pin if we wanted, if it isn't high there has obviously been an error, but for simplicity we just exit the routine.
5. We now can transfer the received byte to where we wish, and either wait for another byte or do something else.

Here are the actual serial routines we will be using, they consist of a number of small subroutines, and require four data registers allocating:

- Xmit_Byte - this is used to store the transmitted byte (passed in W).
- Rcv_Byte - this is used for the received byte, and is copied to W on exiting the routine.
- Bit_Cntr - used to count the number of bits sent or received, set to 8 and decremented.
- Delay_Count - used in the two delay routines.

The routines themselves consist of three subroutines that are called, and two internal subroutines, not normally called from elsewhere:

- **SER_INIT** - this is only ever called once, usually when the program first runs, as part of the normal initialisation stages, it sets the input and output pins to the correct direction, and sets the output pin to the correct polarity - high, so the RS232 line sets at -12V.
- **XMIT_RS232** - this is the transmit routine, simply load the byte to be transmitted into the W register and call this subroutine (CALL XMIT_RS232).
- **Rcv_RS232** - this is the receive routine, when you call this is waits for a received byte, there's no timeout, so it will wait for ever if it doesn't receive a byte. To use the subroutine simply call it (CALL Rcv_RS232) and it returns the received byte in the W register.
- **Start_Delay** - internal subroutine that delays 52uS, used by the Rcv_RS232 subroutine to delay half a bit length.
- **Bit_Delay** - used by both the transmit and receive subroutines, to provide a 104uS (one bit) delay.

```
;Serial routines
            Xmit_Byte    Equ  0x20        ;holds byte to xmit
            Rcv_Byte     Equ  0x21        ;holds received byte
            Bit_Cntr     Equ  0x22        ;bit counter for RS232
            Delay_Count  Equ  0x23        ;delay loop counter
SER_INIT
            BSF     STATUS, RP0           ;select bank 1
            BCF     TRISB, 6              ;set B6 as an output
            BSF     TRISB, 7              ;set B7 as an input
            BCF     STATUS, RP0           ;select bank 0
```

```
            BSF     PORTB, 6              ;set B6 high
            RETURN

XMIT_RS232  MOVWF   Xmit_Byte             ;move W to Xmit_Byte
            MOVLW   0x08                  ;set 8 bits out
            MOVWF   Bit_Cntr
            BCF     PORTB, 6
            CALL    Bit_Delay
Ser_Loop    RRF     Xmit_Byte , f         ;send one bit
            BTFSS   STATUS    , C
            BCF     PORTB, 6
            BTFSC   STATUS    , C
            BSF     PORTB, 6
            CALL    Bit_Delay
            DECFSZ  Bit_Cntr  , f         ;test if all done
            GOTO    Ser_Loop
            BSF     PORTB, 6
            CALL    Bit_Delay
            RETURN

Rcv_RS232   BTFSC   PORTB, 7              ;wait for start bit
            GOTO    Rcv_RS232
            CALL    Start_Delay               ;do half bit time delay
            BTFSC   PORTB, 7              ;check still in start bit
            GOTO    Rcv_RS232
            MOVLW   0x08                  ;set up to read 8 bits
            MOVWF   Bit_Cntr
            CLRF    Rcv_Byte
Next_RcvBit CALL    Bit_Delay
            BTFSS   PORTB, 7
            BCF     STATUS    , C
            BTFSC   PORTB, 7
            BSF     STATUS    , C
            RRF     Rcv_Byte  , f
            DECFSZ  Bit_Cntr  , f         ;test if all done
            GOTO    Next_RcvBit
            CALL    Bit_Delay
            MOVF    Rcv_Byte, W
            RETURN

Start_Delay MOVLW   0x0C
            MOVWF   Delay_Count
Start_Wait  NOP
            DECFSZ  Delay_Count , f
            GOTO    Start_Wait
            RETURN

Bit_Delay   MOVLW   0x18
            MOVWF   Delay_Count
Bit_Wait    NOP
            DECFSZ  Delay_Count , f
            GOTO    Bit_Wait
            RETURN
```

The routines presented here use PortB pin 6 as the output, and PortB pin 7 as the input, they are based on a 4MHz clock frequency. As it's all done in software you can easily change the port and pin designations, and simply alter the delay timings for different clock speeds or baud rates.

Tutorial 7.1 - required hardware, Main Board and Serial Board.

This first sample program simply transmits a few ASCII characters out of the serial board, it displays 'RS232'. In this example each character is individually loaded in to the W register and the XMIT_RS232 subroutine is called.

Tutorial 7.2 - required hardware, Main Board and Serial Board.

This second sample program transmits two lines of text, the text is stored as a string (terminated by '0x00') in the top page of memory, the two characters '0x0A' and '0x0D' in the string are LF and CR to move the cursor to the start of the next line. The XMIT_RS232 subroutine is called repeatedly in the loop which reads the string.

Tutorial 7.3 - required hardware, Main Board, LCD Board and Serial Board.

This third sample program receives data one character at a time and displays it on the LCD module. Please note that both this, and the next tutorial, can only handle one character at a time - as there's no handshaking involved the routine on the PIC must finish whatever it has to before the next character arrives - if a continuous stream of data is incoming it only has 52uS before the next byte arrives, and this is too fast for the LCD to have finished displaying the previous character. There are various ways of overcoming this - firstly, as long as you are typing the characters on the keyboard there won't be a problem (you can't type fast enough), secondly you could arrange for the transmitted protocol to have more than one stop bit (two stop bits would give three times as long to display the characters, three stop bits would give five times as long, and so on). Or you could buffer the characters in PIC data registers, this still wouldn't allow a continuous data stream, but would probably do all that's required. (For a further possibility see Tutorial 7.7a)

Tutorial 7.4 - required hardware, Main Board, LCD Board and Serial Board.

This fourth sample program receives data one character at a time, displays it on the LCD module (as in 7.3) and then echo's the character back to the PC screen.

Tutorial 7.5 - required hardware, Main Board, LED Board and Serial Board.

This fifth sample program receives data one character at a time, displays it on the LED board and then echo's the character back to the PC screen, the ports have been swapped around (PortA now connects to the serial board, and PortB connects to the LED board, because pin A4 is an open-collector output). This would make a nice simple way of providing eight switched outputs controlled from a serial port.

Tutorial 7.6 - required hardware, Main Board, Switch Board and Serial Board.

This sixth sample program receives one data byte from the PC (any byte - just to initiate a read), reads the switches connected to PortB, and sends the result back to the PC as a string of eight 1's and 0's - B0 first, and B7 last - followed by CRLF. If you examine the code, the routine for sending the PortB reading as a series of ASCII 1's and 0's is based on the XMIT_RS232 code, it reads the bits in turn in exactly the same way, only the action taken for each bit is different. As the previous example makes a nice easy way of writing eight bits, this one makes a nice easy way of reading eight bits.

Tutorial 7.7a - required hardware, Main Board 2, LCD Board and Serial Board.

This seventh sample program works exactly like Tutorial 7.4, but is based on the 16F876 at 20MHz, and uses the hardware USART rather than software emulation. As it uses hardware to receive the serial data, this gives a lot more time for processing and displaying characters, around 1mS or so. There isn't a 16F628 version of this tutorial yet as I have to change the serial board connections over, as soon as this is done I'll post a 16F628 version as well - if you want to do it, the values for the USART are SPBRG=25 and BRGH=1.