

Introduction

This document is an attempt to answer your programming questions and queries relating to Microchip's PIC microcontrollers. Currently the topics covered in this document relate only to the PIC18F series of microcontrollers and to the PIC18F4580 used on our P0701 USB-PIC module.

If you have a programming problem then send us an email or enter your question on our 'Questions and answers' web page.

Email: questions@compactcontrol.co.uk

web: www.compactcontrol.co.uk

Table of Contents

Introduction.....	1
Q1: I have set the TRIS register for port A to be all inputs but when I try to read from the port the bits always read as 0.....	1
Q2: I have set the TRIS register for port D to be all inputs but when I try to read from port D bits 0 to 3 always read as 0.....	2
Q3: How do I use a timer to get a regular interrupt.....	2
Using the internal clock.....	2
Basic timer interrupts.....	2
Using the timer associated with the CCP module.....	3
Using an external clock.....	3

Q1: I have set the TRIS register for port A to be all inputs but when I try to read from the port the bits always read as 0.

You can get similar problems with port E bits. The problem is the A to D converter. The A to D converter register ADCON1 defines which port pins are analogue or digital inputs and if a pin is defined as an analogue input then it will always read as 0.

If you are not using the ADC then the easiest way to stop this problem is to set the PBADEN bit in the configuration register 3H, this will cause the PCFG3:PCFG0 bits in ADCON1 to be initialised as 0 hence setting the ADC channels to 0 and enabling all shared port pins to be used as digital I/O. If you leave the PBADEN bit clear then the PCFG3:PCFG0 bits in ADCON1 will be set to 0111 and this means that the following port pins will be assigned to the ADC and will read as 0.

Port A	RA0	Port A	RA5
Port A	RA1	Port E	RE0
Port A	RA2	Port E	RE1
Port A	RA3	Port E	RE2

If you are using the ADC then you should set ADCON1 before trying to use any of the affected port pins, e.g.

```
MOVLW 3
MOVWF ADCON1, A
```

This will set up the ADC with 3 analogue inputs (RA0, RA1, RA2), the remaining port pins shared with the ADC will all be available as digital I/O.

Q2: I have set the TRIS register for port D to be all inputs but when I try to read from port D bits 0 to 3 always read as 0

This problem is caused by the comparator module configuration. If you look at the CMCON register there are 3 bits CM2:CM0, these set the comparator mode and are initialised to 0 (setting mode 0) after a reset or power up. Mode 0 has port bits RD3:RD0 assigned to the comparator module as analogue inputs so they will always read as 0 when you read the port. To use port bits RD3:RD0 as digital I/O you must set the comparator mode, see the PIC data sheet for more details.

If you want to use all 4 shared inputs as digital I/O then turn off the comparator module first, e.g.

```
MOVLW 7
MOVWF CMCON, A
```

This sets the comparator mode to 7 which disables the comparator module and assigns RD3:RD0 as digital I/O.

Q3: How do I use a timer to get a regular interrupt.

Most of the timer modules in a PIC consist of a prescaler and a timer that can be configured as 8 or 16 bits, some timers also include the ability to use an additional oscillator, we will look at all of these options.

Using the internal clock.

Basic timer interrupts

The easiest way to get a regular interrupt is to set up a timer with a prescaler running from the internal clock, assuming the master clock source is an accurate timebase, e.g. a crystal, then the interrupts will be accurate too, however there is a problem in the way the timer is reset.

```
MOVLW 0xb0
MOVWF T1CON, A
MOVLW 0xfb
MOVWF TMR1H, A
MOVLW 0x1e
MOVWF TMR1L, A
BSF T1CON, TMR1ON, A
BSF PIE1, TMR1IE, A
```

Assuming Fosc is 40MHz then the above code sets up timer 1 to be 16 bits with a prescaler of 8:1 and a count of 1250, and its source is Fosc/4, so it will rollover after 10000 counts (or 1mS). If the timer 1 interrupt is enabled and the handler reloads the timer as below.

Handler

```
BTFSS PIR1, TMR1IF, A
bra notTimer1
MOVLW 0xfb
MOVWF TMR1H, A
```

```
MOVLW 0x1e
MOVWF TMR1L, A
BCF PIR1, TMR1IF, A
```

Then you will get a regular interrupt occurring 1000 times a second. The big problem with this is interrupt latency and other interrupts. For many timers the prescaler can be only set to 1:8 maximum, this means that you have only 8 instruction cycles available to handle the interrupt and reset the timer otherwise you will lose one or more counts and the timebase will be slow. You can compensate for this by reloading the timer with a slightly higher value and this will be good enough for most applications, but where you need a really accurate clock source this still may not be accurate enough.

Using the timer associated with the CCP module

Some PIC's have a CCP module (capture compare PWM module) and a timer associated with the module, this timer is different. The timer associated with CCP also has a register that it is compared against and the timer counts from 0 until it matches the register value and then resets back to 0, also an interrupt can be generated at the reset point. This can be used to generate an accurate timebase since the interrupt handler is not reloading the timer.

```
MOVLW 0x22
MOVWF T2CON, A
MOVLW .125
MOVWF PR2, A
BSF T2CON, TMR2ON, A
BSF PIE1, TMR2IE, A
```

The above example sets timer 2 to have a prescaler of 1:16 and a post scaler of 1:5. The PR2 register is set to 125, these 3 values when multiplied give an interrupt every 10000 counts. With $F_{osc} = 40\text{MHz}$, the input to timer 2 is 10MHz and an interrupt will be generated every 1mS.

You can select the frequency of the main clock to be a frequency that would not require a timer to be reloaded on every interrupt. If you are using an external crystal of 4.194304MHz, a common frequency, and you have enabled the PLL, so $F_{osc}/4 = 4.194304\text{MHz}$ then the example below can be used.

```
MOVLW 0x05
MOVWF T0CON, A
BSF T0CON, TMR0ON, A
BSF INTCON, TMR0IF, A
```

This sets timer 0 to be 16 bits and have a prescaler of 64, 16 bits means a roll-over every 65536 counts and with a prescaler of 64 this is 4194304 counts, so you will get an interrupt once per second and you do not need to reload the timer, so the interrupt will be accurate.

Using an external clock

In most PIC18F microcontrollers there will be at least one timer module that is capable of working with an external crystal. The range of crystal frequencies is vast and most requirements can be accommodated. The best approach is to select a crystal frequency that allows the timer to roll-over to generate the required interrupt rate, if you choose a frequency that requires the timer to be reloaded on every interrupt then there is the possibility of missing timer ticks and compromising the

accuracy of the interrupts. The below examples show various crystal frequencies, timer prescaler values and timer sizes (8 or 16 bit) to achieve various useful interrupt rates.

Crystal Frequency	Prescaler	Timer size 8 or 16 bit	Resulting Interrupt frequency
4.194304 MHz	8	16	8 Hz
32.768 KHz	8	8	16 Hz
3.2768 MHz	2	16	25 Hz
3.2768 MHz	1	16	50 Hz
4.194304 MHz	1	16	64 Hz
6.5536 MHz	1	16	100 Hz
8.192 MHz	1	16	125 Hz
32.768 KHz	1	8	128 Hz
9.8304 MHz	1	16	150 Hz
307.2 KHz	8	8	150 Hz
307.2 KHz	4	8	300 Hz
307.2 KHz	2	8	600 Hz
307.2 KHz	1	8	1200 Hz
3.6864 MHz	8	8	1800 Hz
4.096 MHz	8	8	2000 Hz
6.144 MHz	8	8	3000 Hz
6.5536 MHz	8	8	3200 Hz
7.3728 MHz	8	8	3600 Hz
7.68 MHz	8	8	3750 Hz
8.192 MHz	8	8	4000 Hz
4.608 MHz	4	8	4500 Hz
9.8304 MHz	8	8	4800 Hz
7.68 MHz	4	8	7500 Hz
1.8432 MHz	8	8	9000 Hz

To use timer 1 with an external clock, connect the crystal between pins T1OSO and T1OSI, you will probably need small value capacitors from each pin to ground, the value of the capacitor depends on the crystal used and can be found in the data sheet for the crystal.

The below code will set up timer 1 with as an 8 bit timer with a prescaler of 8 and using the oscillator with an external crystal.

```

MOVLW 0x7A      ; Set up the timer
MOVWF T1CON, A
BSF T1CON, TMR1ON, A      ; Turn on the timer.

```