8 MPLAB ICD & PIC 16F877 tutorial

In the previous exercise, the simulation and debugging facilities of the MPLAB IDE were explored. The speed at which a simulator runs depends on the speed of your computer and the number of other tasks running in the background. The software simulator must update all of the simulated registers and RAM, monitor I/O, set and clear flags, check for break and trace points in software, and simulate the PICmicro MCU instruction with instructions being executed on your computer's CPU. MPLAB-SIM executes on instruction cycle boundaries, and resolutions shorter than one instruction cycle (T_{CY}) can not be simulated. MPLAB-SIM is a discrete-event simulator where all stimuli are evaluated, and all responses are generated, at instruction boundaries, or $T_{CY} = 4$ T_{OSC}, where T_{OSC} is the input clock period. Therefore some physical events can not be accurately simulated. These fall into the following categories:

- Purely asynchronous events
- Events that have periods shorter than one instruction cycle

In summary, the net result of instruction boundary simulation is that all events get synchronized at instruction boundaries, and events smaller than one instruction cycle are not recognized.

An in-circuit debugger is a system which allows the user to interact with/monitor the processor during actual operation. Because an actual processor is used, it is possible to observe the effects of physical events which cannot be simulated. The MPLAB ICD is a programmer for the PIC16F87X family, as well as an in-circuit debugger. It programs hex files into the PIC16F87X and offers basic debugging features like real-time code execution, stepping, and breakpoints. MPLAB ICD is intended to be used as an evaluation and debugging aid in a laboratory environment.

MPLAB ICD utilizes the In-Circuit Debugging capability of the PIC16F87X and Microchips In-Circuit Serial Programming (ICSP) protocol to both program and act as an in-circuit debugger for PIC16F87X devices. PIC16F87X microcontrollers can be serially programmed while in the end application circuit. This is simply done with two lines for clock and data and three other lines for power, ground, and the programming voltage. This allows customers to manufacture boards with unprogrammed devices, and then program the microcontroller just before shipping the product. This also allows the most recent firmware, or a custom firmware to be programmed.

To enable in-circuit debugging, the Debug Code (residing in the microcontroller in the MPLAB ICD Module) is programmed into the target PIC16F87X. The code is an MPASM module that will be programmed into the PIC16F87X automatically by MPLAB IDE. This code will reside at the end of program memory. It will reside in program memory locations 0x1F00 to 0x1FFF of the PIC16F877 processor. During execution, the Debug Code assumes responsibility for monitoring the microcontroller/user program, and communicates with the host using the same lines used for ICSP. Due to the built-in in-circuit debugging capability of the PIC16F87X, the ICSP function, and the Debug code, the MPLAB ICD uses the following on-chip resources:

- MCLR/VPP is shared for programming
- Low-voltage ICSP programming is disabled
- I/O pins RB6 and RB7 reserved for programming and in-circuit debugging
- Six general purpose file registers are reserved for debug monitor: Data Memory Address 0x070 (0x0F0, 0x170, 0x1F0), and 0x1EB to 0x1EF

- First program memory location (address 0x0000)(i.e in the user code the first instruction must be a NOP instruction)
- Last 100h (256) words of program memory area are reserved for Debug Code
- One stack level is not available.

In this tutorial, the timer, and A/D conversion facilities of the PIC16F877 will be introduced. The code will be run both in the simulator, and the in-circuit debugger. This tutorial is based on Tut877 in [ICD00, Ch. 3].

Hardware Overview

The PIC16F877 has 5 digital I/O ports (A-E) each between 3 and 8 bits wide. Each port is mapped into the register file space, and may be read/written to like any other register. The circuitry is such that it is not possible to physically input to and output from a particular pin simultaneously. For most ports, the I/O pins direction (input or output) is controlled by the data direction register, called the TRIS register. `TRIS<x>` controls the direction of `PORT<x>`. A '1' in the `TRIS` bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. An easy way to remember is that a '1' looks like an I (input) and a '0' looks like an O (output). The `PORT` register is the latch for the data to be output. When the `PORT` is read, the device reads the levels present on the I/O pins (not the latch). This means that care should be taken with read-modify-write commands on the ports and changing the direction of a pin from an input to an output.

The pins on the PIC16F877 are multiplexed so that one of several functions may be selected (e.g. pin 2 may be used as either bit 0 of I/O port A, or as channel 0 of the A/D converter). The TRIS registers control the direction of the port pins, even when they are being used as analog inputs. The user must ensure the TRIS bits are maintained set when using the pins as analog inputs.

The `OPTION_REG` register is a readable and writable register which contains various control bits to configure Timer0. Timer mode is selected by clearing bit T0CS (`OPTION_REG<5>`). In Timer mode, the Timer0 module will increment every instruction cycle without the prescaler, and every `n` cycles with the prescaler, where n is determined by the pre-scale ratio. The PSA and PS2:PS0 bits (`OPTION_REG<3:0>`) determine the prescaler assignment and prescale ratio.

bit 3 PSA: Prescaler Assignment bit 0 = Prescaler is assigned to the Timer0 module

bit 2-0 PS2:PS0: Prescaler Rate Select bits

Bit Value	Timer0 Rate
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

When the `TMR0` register overflows from FFh to 00h, this overflow sets bit `T0IF` (`INTCON<2>`). Bit T0IF must be cleared in software.

The A/D allows conversion of an analog input signal to a corresponding 8-bit digital number. The output of the analog sample and hold is the input into the converter, which generates the result via successive approximation. The analog reference voltage is software selectable to either the positive supply voltage (5V), or the voltage level on the AN3/VREF pin. For the A/D converter to meet its specified accuracy, the charge holding capacitor must be allowed to fully charge to the input channel voltage level. After the analog input channel is selected (changed) this acquisition must be done before the conversion can be started. The minimum acquisition time is $20\mu s$. The A/D conversion time per bit is defined as T_{AD}. The A/D conversion requires a minimum $12T_{AD}$ per 10-bit conversion. The source of the A/D conversion clock is software selected as a multiple of the clock frequency T_{OSC}. For correct A/D conversions, the A/D conversion clock (T_{AD}) must be selected to ensure a minimum T_{AD} time of $1.6\mu s$ (i.e. total conversion time approximately $20\mu s$). For a 4MHz clock this must be at least $8T_{OSC}$. It follows that overall A/D sampling cannot occur more frequently than 25kHz. Timer0 with a prescale value can be used to time the interval between samples. For a 4MHz clock, the prescale value must be at least 160 i.e. 256 must be used.

In the Lab

Simulating A/D Using the above information, we can write and simulate a program which will sample data from the A/D channel 0, and put the digital value out on port D. Follow the instructions on page III 16. You may wish to refer to the previous tutorial and the data sheet for the PIC16F877[PIC01]. Build the project you have made, and using the simulator, examine the operation of the timer, the A/D converter and PortD.

Using the ICD In order to observe this code running on a microprocessor, build the circuit shown on page III 17 using the PIC16F877 header, and have it checked by the technician/demonstrator. Follow the instructions on page III 18 to connect the ICD module, download and execute the code.

Debugging Using the MPLAB ICD and MPLAB IDE debugging functions, you can successfully find and fix the bugs in your code. Introduce some bugs and see how they affect the operation of the program; for example:

- comment out the line which clears TRISD
- comment out the line which clears the completion flag ADIF

Try to make use of all the IDE features, i.e. breakpoints, watch windows etc. Are there any differences between breakpoint behaviour for the simulator and the debugger?

References

[ICD00] MPLAB ICD USERS GUIDE. Manual DS51184D, Microchip Technology Inc., 2000.

[MPL00] MPLAB IDE, SIMULATOR, EDITOR USERS GUIDE. Manual DS51025D, Microchip Technology Inc., 2000.

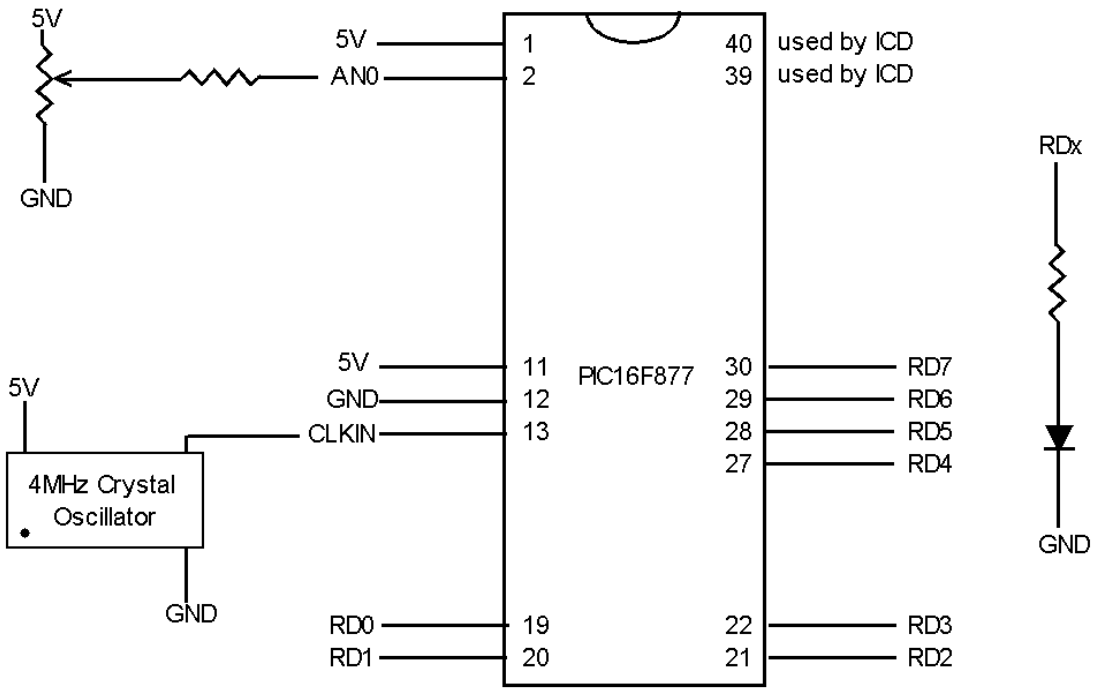
[PIC01] PIC16F87X Data Sheet: 28/40-Pin 8-Bit CMOS FLASH Microcontrollers. Technical Reference Document 30292c, MicroChip Technology Inc., 2001.

A/D SAMPLING SIMULATION

1. Create a new project, using MPLAB-SIM as the development tool for the PIC 16F877, and add a single assembly language node.
2. Enter the following code in the .asm file:

```
        LIST p=16f877

    ; Include file, change directory if needed
        INCLUDE "p16f877.inc"

    ; Start at the reset vector
        ORG     0x000
        nop

Start   BANKSEL PORTD
        clrf    PORTD       ;Clear PORTD
        movlw   B'01000001' ;Fosc/8, A/D enabled, Sample Channel 0
        movwf   ADCON0

        BANKSEL OPTION_REG
        movlw   B'10000111' ;TMR0 prescaler, 1:256
        movwf   OPTION_REG
        clrf    TRISD       ;PORTD all outputs
        movlw   B'00001110' ;Left justify, 1 analog channel
        movwf   ADCON1      ;VDD and VSS references

        BANKSEL PORTD

Main    bcf     INTCON,T0IF

Loop    btfss   INTCON,T0IF ;Wait for Timer0 to timeout
        goto    Loop

        bsf     ADCON0,GO   ;Start A/D conversion for channel 0

Wait    btfss   PIR1,ADIF   ;Wait for conversion to complete
        goto    Wait

        movf    ADRESH,W    ;Write A/D result to PORTD
        movwf   PORTD       ;LEDs
        bcf     PIR1,ADIF   ;Clear the completion flag

        goto    Loop        ;Do it again

        END
```

3. Set up a register stimulus file (.reg) with the values 25, 2e, 85, 30, 55, 4F for register `ADRESH`, to be injected at the program label `Main`.
4. You may wish to watch the values of `ADRESH`, `PORTD`, `INTCON`, `ADCON0`, and `PIR1`.



Parts List

- (8) LED's
- (9) 390Ω resistors
- 5KΩ potentiometer
- 4MHz Crystal Oscillator
- PIC16F877
- ICD header + RJ12 cable + ICD debug module + serial cable
- Breadboard, wire and 5V power supply

Figure 7: Circuit for ICD tutorial

A/D SAMPLING In-Circuit Debugging

1. **After your circuit has been checked!!** Connect one end of the serial cable to the computer, and the other to the ICD module. Then plug the header into the ICD module using the RJ12 cable.
2. To use the ICD instead of the simulator, choose Project > Edit Project and click on the Change button next to the development mode. Select MPLAB ICD Debugger under the Tools Menu. Make sure you select the PIC16F877 processor. Click OK. A window will inform you that no hex file has been built for this project. We will build the hex file later. Click OK. MPLAB IDE will now establish communications with the MPLAB ICD. The MPLAB ICD dialog will momentarily appear during this process. If you receive an error message, double-check the connections for power supply, socket seating, and cable seating. To debug, **the MPLAB ICD dialog must remain open**. Do not close the MPLAB ICD dialog when it appears. If you want, you can move the dialog. Notice that the correct development mode and processor are shown in the Edit Project dialog. Click OK to close the Edit Project dialog.
3. At this point, the MPLAB ICD dialog should be on your desktop. The Status bar displays each executed MPLAB ICD function and the status. When you program a device, you can watch the progress in this area. When the operation is complete, the Status box displays the message "Waiting for user command". To program the target PIC16F877 device, the ICD Options dialog must first be set up for programming. Click Options in the MPLAB ICD dialog to open the ICD Options dialog.
   - Use Checksum as ID by selecting the checkbox.
   - Oscillator XT is used in this tutorial.
   - the Watchdog Timer (WDT) should be off/disabled.
   - the Power-Up Timer (PWRT) should be off/disabled.
   - the Brown-Out Detect (BOD) should be off/disabled.
   - Low voltage ICSP programming should be disabled when using the MPLAB ICD.
   - Code Protect Data EE Turn off code protection for this tutorial.
   - Flash Memory Write No memory will be written to EECON for this tutorial.
   - Code Protect Turn off code protection for this tutorial.
   - Make sure that all checkboxes under Program Options are checked.
   - Click Def. Addr to set the address range to the maximum program memory available based on the device you selected.
4. Click the Program button to program the hex-file and debug code into the PIC16F87X in the MPLAB ICD Header. Programming may take a couple of minutes. During programming, the Status box shows the current phase of the operation. When programming is complete, the Status box displays the message "Waiting for user command."
5. The MPLAB ICD executes in real-time mode or in step mode. Real-time execution occurs when the PIC16F87X in the MPLAB ICD Header is put in MPLAB IDE's Run mode. Step mode execution can be accessed after the processor is halted. We will begin in real-time mode. Click the Run toolbar button or issue the Debug > Run > Runcommand. The Status bar at the bottom of the MPLAB IDE desktop should turn yellow. Turn the potentiometer and observe the LEDs. If the program is working correctly, you should see a binary representation of the voltage value across the potentiometer.

9 CourseWork B - 10%

There are two laboratory exercises to be submitted. Each is worth 5% of your final course mark. For each exercise:

- Design the algorithm.
- Write the assembly language code, assemble the program using the MPLAB assembler, and test it in the simulator.
- Connect the circuit and have it checked by the technician for correctness.
- Power up the circuit, download the hex file to the PIC16F877 using the ICD, and test that it works correctly.
- Demonstrate the working system to the lab demonstrator. At the time of demonstration, please submit your report containing:
  - the algorithm design,
  - printout of commented code,
  - copy of the code (*.asm; *.hex) on disk, and
  - a set of 5 test cases and the results

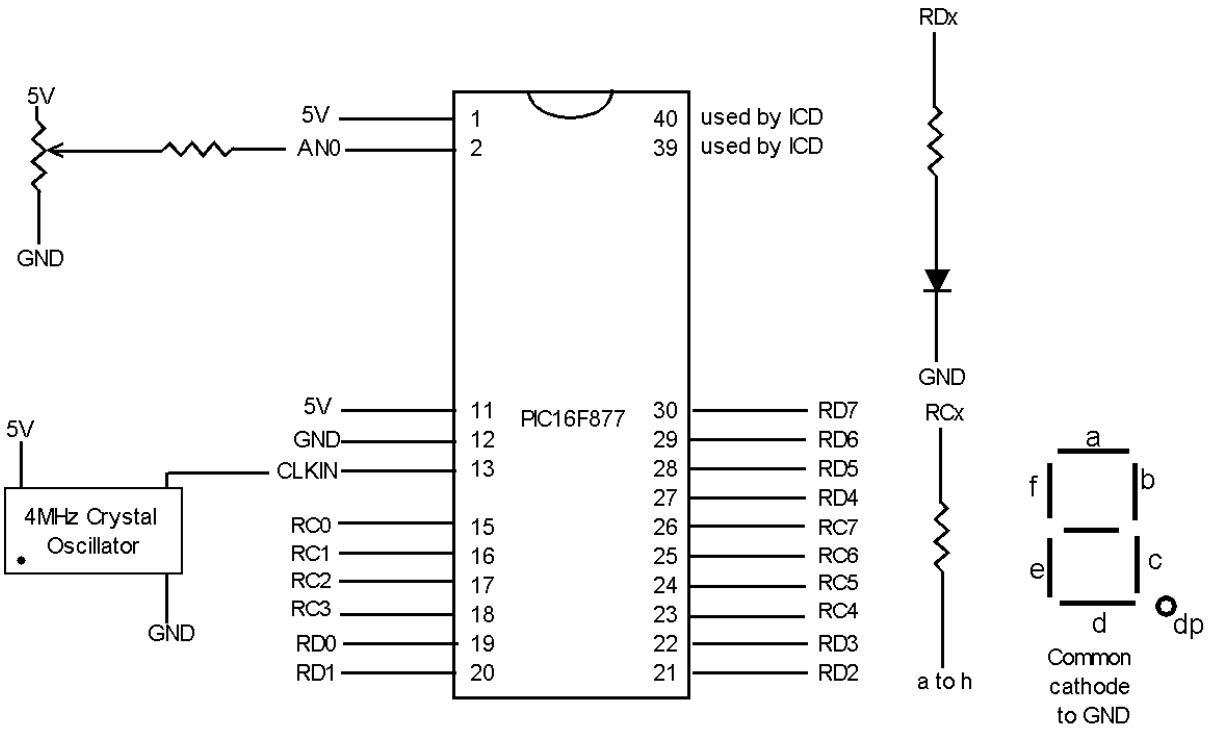
The demonstrations/submissions should take place before the end of Week 8 (15/3/02) for Exercise 1, and before the end of Week 10 (5/4/02) for Exercise 2.

Laboratory Exercise 1

This first exercise continues from the tutorial, and is intended to demonstrate how the 16F877 can be used to read an external input and display it via the output ports. The circuit is the same as for the tutorial, with the addition of a common-cathode 7-segment LED digit, which should be connected to PORTC (RC0 –> a; RC1 –> b; etc). Please remember to use appropriate resistances for the 7-segment digit. The value of the voltage drop over the potentiometer is read using the A/D and this value should be displayed as:

- a moving bar using 8 leds;
- a digit between 0 and F on an LED display.

The maximum voltage drop should be shown as all leds lit (or digit F) and the minimum voltage drop when only one of the leds is lit (or digit 0). (HINT: Think about using a lookup table).



Additional parts: 7 segment LCD digit display and resistors.

Figure 8: Circuit for laboratory exercise 1

Laboratory Exercise 2

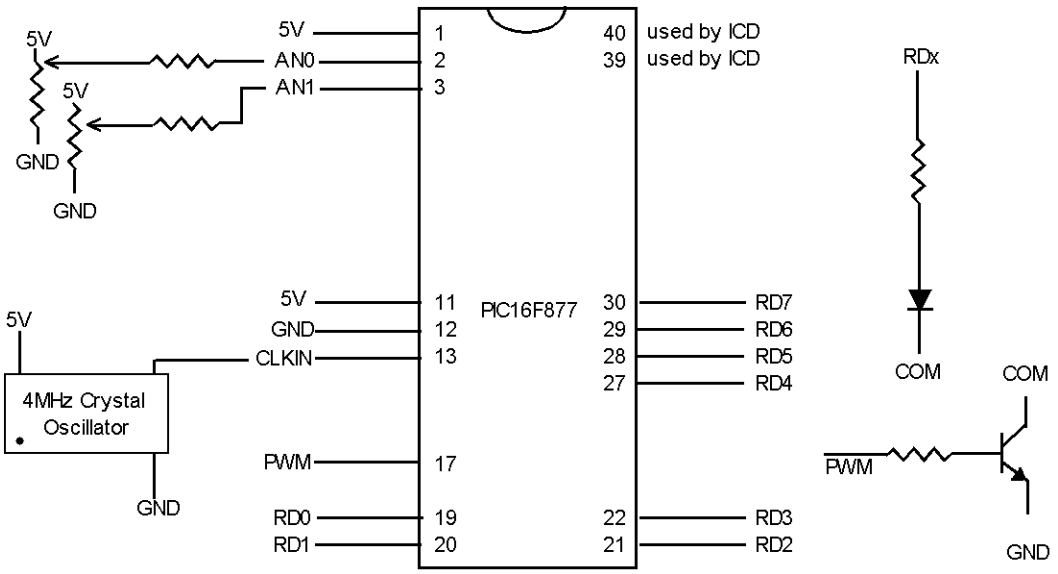
The second exercise also builds upon the tutorial, by adding a second 5K potentiometer and a transistor to change the intensity of the leds, and changing the A/D sampling so that it is interrupt driven.

In order to change the intensity of the leds, the Pulse Width Modulation (PWM) output may be used to turn a transistor, connected between the leds and ground, rapidly on and off. There are two things that are necessary to use the PWM output, the setting of the total period value and the high period value. The total period value, tp, and the high period value, hp, are related as follows:

$$\begin{aligned} tp &= \frac{125000}{f} \\ hp &= \text{Duty Cycle} \times tp \end{aligned}$$

Where f is the frequency. A minimum frequency of 20 Hz is needed for the flashing of the led display not to be noticed. The value of the second potentiometer is used to determine the high period value such that the duty cycle should vary approximately from 0% to 100%.

In the tutorial, a "busy poll" was used to determine when Timer0 had overflowed, and when the A/D conversion was complete. For this lab exercise, use interrupts instead of the busy poll, for Timer0 (start the next sample) and the A/D Completion flag (write sample to port or change PWM frequency).



Additional parts: $5k\Omega$ Potentiometer, (2) 390Ω resistors, Q2N3904 transistor.

Figure 9: Circuit for laboratory exercise 2