EN 580.471 Fall 2008
Principles of Design of Biomedical Instrumentation

Lab 4: Programmable Microcontrollers – Introduction to PIC Chips

Introduction

The PIC microcontroller is a device that plays an important role in many of today's products and will likely be at the center of your Challenge and Honors projects. While learning the capabilities of the PIC will be an ongoing topic of the semester, you should begin to familiarize yourself with the documentation. This lab will be a gentle introduction to the basic use of the PIC, which we will continue to build on in the coming weeks.

To begin with, download the PIC16F87X Datasheet and the C Compiler Reference Manual in WebCT. You should pay attention to the Pin Diagram for the PIC on page 3 of the manual, as this diagram will be a frequent reference for you.

The C Compiler Reference will be your guidebook for programming and controlling the PIC and ultimately controlling your future labs / projects. It is important that you browse the table of contents, particularly the Built-in Functions, Common Questions and Answers and Example Programs sections. While it is not necessary to memorize any of the built-in functions, you should know where to find information on functions you will need to use.

The programmers we have available to us in the lab use the C language for programming the PIC. While it helps if you know some C already, you can certainly get by without knowing any at all. There are some useful functions and sample code included at the end of this lab. Read through these addendums carefully before attempting the pre-lab.

You will be using the PIC in nearly every remaining lab and almost certainly in your chosen projects. With that in mind, build your circuit with care and ensure that you have a firm grasp on the basics of programming your PIC before moving on to the next lab.

Pre-lab Questions [30 points]

We will be using the PIC16F877 – keep this in mind when referring to the PIC16F87X datasheet.

Use the datasheet, the C Compiler manual the class notes and the notes at the end of this document to complete the following:

1. Draw the schematic for basic PIC operation, in particular, show where capacitors should be placed, how the clock is wired to the PIC and how to handle the master reset pin. You do not have to draw and label all pins, only those vital for setting up the PIC. **[10]**

2. How many I/O ports are available on the 16F877? What are their names / pin numbers? Which pins (eg. RA0) are capable of receiving analog input? **[6]**

3. Write a program to check the input on a particular pin and light an LED when that pin receives a high voltage and turn it off when it receives a low voltage **[10]**

4. Using the sample code attached at the end of this document as well as the C Compiler Manual, provide brief explanations in your own words of what the following functions do and what their roles are in using the PIC's A/D converter: setup_adc(), setup_adc_ports(), set_adc_channel(), and read_adc(). **[4]**

Lab Procedure

Setup the PIC

1. Using the schematic you designed in the prelab, correctly place and wire your PIC on your breadboard.
    a. Be careful when placing and wiring components during this stage – confusing wiring and poor placement will create many problems for you down the road.

Test your PIC (Code provided)

1. Once your circuit is setup properly, compile the included Blink LED Program code and download it to the PIC.
2. Power up the PIC, replace it in your circuit, wire the LED to the proper output pin and ensure that the LED is blinking with the proper delay

Interact with the PIC

1. Using the code you wrote for the pre-lab, program the PIC to light up an LED when it receives an input on a particular pin.

Analog-to-Digital Converter Module:

1. Using the sample code as a template, write a program to receive an analog input on a particular pin and check whether that value is above a given threshold.
2. Building on the code from above, modify the code so that it lights an LED whenever the threshold is above 128.
3. Check the output of your program by feeding in a slow ~100 Hz sine wave into the pin. The output from your test pin should be a square wave.

This is the end of this PIC lab. However, it is strongly recommended that if you have any free time you try different tests to assure yourself that you can program the PIC – you will use these chips in most things from now on.

Lab Write-Up **[30 points]**

1. Include the 3 programs used during the lab. Include brief comments describing what the program does. **[5 x 3 =15]**
2. You will be using various types of sensors in your upcoming labs/projects. Write a program that reads the input analog signal from the sensor and based on its magnitude, bins it into 3 ranges (e.g. low, medium and high). Based on the binning, you have to generate square waves of 3 different duty cycles: 0.25, 0.5 and 0.75 respectively. **[15]**

Setting up MPLAB

The easiest way to setup a project in MPLab is to use the Project Wizard.

1. Start MPLab
2. Click on Project → Project Wizard
3. Click Next at the Welcome screen
4. Select **PIC16F877** as the device
5. Select **CCS C Compiler for PIC12/14/18** as the Active Toolsuite

5b. If the Location field is blank, type in: "C:\Program Files\PICC\Ccsc.exe"

6. Create a name for your project and establish a directory where your project will reside
7. If you already have a source file you'd like to add to the project, you can add it now or it can always be done later.
8. Review that the selections are correct and click Finish

To create a source file, click on File → New. Click on File → Save As and save the file with a '.c' extension to indicate that it is a source file. (By default, all files are saved with a '.c' extension). Save this file in your project directory. In the project directory tree window, right click on Source Files and select Add Files to include additional source files to the project.

Once your source file is complete, the next step is to setup the programmer.

1. Click on Programmer → Select Programmer → PICStart Plus
2. To start the programmer, select Programmer → Enable Programmer
3. Next, the bits need to be configured as follows:

   Click on Configure → Configuration Bits

   Oscillator - HS
   Watchdog Timer - Off
   Power Up Timer - Off
   Brown Out Detect - Off
   Low Voltage Program - Disabled
   Flash Program Write - Enabled
   Data EE Read Protect - Off
   Code Protect - Off

Now you are ready to proceed to compiling and building the source file. Click on Project → Build All.

Once a Build is successful, you can write the code onto the PIC chip. Click on Programmer → Program to copy the source code (now in assembly language) onto the PIC microcontroller.

Useful Functions

These are a list of some very useful built in functions that will make programming in PIC much easier. For more details on each function or for additional functions that may also be useful, consult the C Compiler manual.
==

output_high(PIN)
output_low(PIN)

These two functions are used to turn on or off a port. PIN would be the actual PIN or Port number such as PIN_A1 or PIN_C3.

Ex:
output_high(PIN_A1)
output_low(PIN_C1)

Another way to do this is the *output_bit(pin, value)* function.

output_bit(PIN_B1, 1)

is the same as

output_high(PIN_B1)

However, *output_bit(pin, value)* is a little more versatile. If we want the output of B1 to be the same as the input of A1, it would be tedious to use the output_high function since we need to first test if A1 is high, and if it is, then set B1 to high. This can be done in one line with output_bit:

output_bit(PIN_B1, input(PIN_A1))

==

input(PIN)

This determines whether the pin is in the high or low state (whether it's getting 5V or Gnd/0V). If it is receiving 5V, input(pin) will output a 1 and 0 if otherwise.

Ex. We want to see if a switch (connected to pin_c1) is turned on. If it is turned on, we will blink the LED connected to pin_b1.

```
while(true)
{
  if( input(PIN_C1) ) //Note: 1 = true, 0 = false so if PIN_C1 is high, this if statement will
execute
        {
        output_high(PIN_B1);
        delay_ms(1000);
        output_low(PIN_B1);
        delay_ms(1000);
        }
}
```

==

output_toggle(pin)

This is used to toggle the high/low state of a pin.

We can redo the blinking LED code using this function:

```
while(true)
{
        output_toggle( PIN_B1 );
        delay_ms(1000);
        output_toggle( PIN_B1 );
        delay_ms(1000);
}
```

==

delay_ms(time)

This causes the program to stop for a set amount of time. An example of this is if we want to turn a LED on and off. If there wasn't a delay, the on/off cycle would be so fast that we won't be able to see it change. We want the LED (connected to port B1) to turn on, delay for 5 seconds, then turn off and then repeat the cycle.

```
while(true)  // this makes the program run continuously
{
output_high( PIN_B1 );
delay_ms(5000);
output_low( PIN_B1 );
delay_ms(5000);
}
```

delay_us(time) works in a similar fashion but in micro-seconds of time.

==

Blinking LED Program

```
/* The following 3 lines will need to be included at the beginning of every program */
#include <16F877.h>
#include <stdio.h>
#use delay(clock=10000000)  //Note that this clock speed may need to be changed

main()
 {
        while(true)
        {
                output_high( PIN_B1 );
                delay_ms(1000);
                output_low( PIN_B1 );
                delay_ms(1000);
        }
}
```

Sample ADC program

The best way to explain how to do analog to digital conversion is to go through an actual program. The following code tests analog port A1 and sees if the input is greater or less than 2.5V. If it is greater, a LED is set to blink.

```
/*
Analog to Digital Conversion Sample Program

Start every program with the following three lines:
#include <16F877.h>
#include <stdio.h>
#use delay(clock=10000000)

The include statements tell the compiler that we are using the PIC16F877 chip and we want to
take advantage of the standard C input/output libraries. For each chip, there are specific functions
and definitions that are pre-made to make your job easier. Take a look in C:\Program
Files\PICC\Devices\16F877.h . Note that every PIN name (ex: PIN_A2) is linked to a number that
the compiler understands. There are a number of other pre-defined functions specific for each
chip.

The clock chip we normally use is 10 MHz. If you use a different clock, set the proper clock speed
using clock=<speed in Hz>
*/


#include <16F877.h>
#include <stdio.h>
#use delay(clock=10000000)

/*
```

```
It is good programming practice to make things you use often into functions. Blinking a LED is
something we will use a number of times so we define a function called blink_led which contains
all the code.
*/
void blink_led() {
        output_high( PIN_B2 );
        delay_ms(100);
        output_low( PIN_B2 );
        delay_ms(100);
        }


void main()
{
        int threshold = 128;        //128 corresponds to 2.5v
        int value;                  //The converted value returned from ADC will be stored here


/*
The following are two functions used to setup the A/D conversion. There is no need to make any
changes to the first line. The second lines tells the compiler which ports we want to be analog.
There are a number of options available. Refer to the 16F877.h file to determine exactly which
parameters the compiler will understand (this is toward the bottom of the page under the Analog
to Digital category). Some other choices are "ALL_ANALOG" which turns all the possible analog
ports on (A0 A1 A2 A3 A5 E0 E1 E2). Note that not all the combinations are possible.
*/

        setup_adc(ADC_CLOCK_INTERNAL);
        setup_adc_ports(AN0_AN1_AN3);

        while(TRUE) //We always want the program to run continuous until we turn off power
        {

/* The first line selects which channel to do the A/D conversion on. Since the ports we initiated
are ANO, AN1, AN3, set_adc_channel(1) indicates that AN1 is selected. (set_adc_channel(0)
would mean AN0 and set_adc_channel(2) would be AN3).

As a general rule, always have a 10us delay after a channel is selected.
*/

                set_adc_channel(1);
                delay_us(10);

/* When the read_adc() function is called, the conversion takes place and a value between 0-255
is outputted and saved in the variable value.
*/
                value = read_adc();

/* If value is numerically greater or equal to the threshold value of 128, the LED is set to blink
*/
                if (value>=threshold)
                {
                        blink_led();
                }
/* It is good practice to always have a delay at the end of each loop. A 3ms delay is not noticeable
but gives enough time for the chip to reset and start over
*/
        delay_ms(3);
        }
}
```