Digital Design and Computer Architecture

Lab 6: C Programming

Introduction

In this lab, you will learn to program a PIC32 microcontroller in C by following a tutorial and then writing several of your own programs.

The µMudd32 Board

In this lab and the next two, you will be using the µMudd32 board developed by Leo Altmann and Christian Jolivet at HMC in 2010. The board contains a PIC32MX675F-512H microcontroller, LEDs, switches, a port for programming the microcontroller, a row of male header pins for connecting to a breadboard, and many other goodies that you'll explore if you take E155. <Note: for *Digital Design and Computer Architecture* textbook users, the lab can be completed through simulation without the board.>

We will be using the PIC32-series microcontroller because PIC is one of the market-leading microcontroller vendors, their development tools are fairly good, and because the PIC32-series of processors is based on the MIPS architecture that we will study later.

Your particular PIC32 microcontroller has 512 KB of Flash memory for programs, 64 KB of static RAM for data, dozens of digital input/output pins, 16 analog input pins, and oodles of special peripherals such as counters, comparators, timers, serial ports, and Ethernet ports. It is rated to operate up to 80 MHz, but runs at 40 MHz on this board. It costs about $7 in low quantities; a version with less memory is available for $3.25 in quantities of 10,000.

You will use the In Circuit Debugger (ICD3) module, affectionately known as the hockey puck, to program your PIC from a PC in the E85 lab. The ICD3 connects to the PC via a USB cable and to the µMudd32 board via a short RJ-11 cable.

Be sure that the µMudd32 board is plugged into the ProtoBoard. The Vin and GND pins on the µMudd32 board should be wired to 5V and Ground on the protoboard, respectively. The ICD3 should be connected to the PC and the board. The protoboard should be turned on. The ICD power and status lights should be green and the active light should be blue. When you are all done with the lab, turn off the protoboard.

You'll be using the DIP switches and LEDs to interact with your program. The four DIP switches in the lower left corner produce a 1 when the bottom side of the switch is down and a 0 when the top side is down. There are 10 LEDs on the right side of the board. The top one indicates that power is ON. The next is always OFF. The remaining 8 display a byte of information, with the least significant bit at the top and the most significant bit at the bottom.

PIC32 C Compiler Tutorial

PIC microcontrollers are programmed from the Microchip MPLAB Integrated Development Environment (IDE). The IDE supports programming in both C and assembly language; you'll use C in this lab and the next, and assembly language in Lab 8.

In this tutorial, you will learn to write and compile a simple program that uses the DIP switches, LEDs, and input and output from the MPLAB console. You'll also learn to step through a program and debug it if you have errors.

Launch MPLAB IDE from the Start menu or from the desktop. Choose Project → New… Create a new project named lab6tut in your Charlie directory (e.g. H:\e85\lab6_xx). Choose Configure -> Select Device and pick the PIC32MX675F512H under the Device pull-down menu. Click OK. Choose Project -> Select Language Toolsuite. Make sure the Microchip C-Compiler Toolsuite is the active Toolsuite. Click OK. Then choose Project -> Build Options… -> Project. In the MPLAB PIC32 C Compiler tab, check "Use Alternate Settings" at the bottom of the window and enter `-g -mappio-debug`. Click OK. This is necessary to let a program print messages through the ICD back to the MPLAB console.

Copy lab6tut.c and e85.c from \\charlie.hmc.edu\Courses\Engineering\E85\Labs\Lab6 to your lab6_xx directory. Choose Project -> Add Files to Project… and add these two C files to your project. Look over the files and see what they do by using File → Open.

Choose Debugger -> Select Tool -> MPLAB ICD3 to test your program on the PIC32 chip using the ICD3. Then choose Project -> Build All to compile the C program. Look at the Build tab in the Output window. You should see no warnings and a report that "BUILD SUCCEEDED." Choose Debugger -> Program to download your program to the PIC. You should see a message in the MPLAB ICD3 tab of the Output window saying "Programming /Verify complete".

If at any point you have trouble communicating with the PIC or the status light on the puck turns red, try unplugging and replugging the USB cable, then choosing Debugger -> Reconnect. MPLAB may hang up for a while after this happens. If you have to kill the program, you may find that your project has been corrupted when you reopen it, and you'll have to recreate the project (but not your C file).

Choose View -> Application In/Out to open a console window for input and output from the program. In the Format section of the console window, set Output and Input to Text.

You will be using the DIP switches and LEDs on the µMudd32 board to interact with your program. The top of the board is the side with the ICD connector.

Choose Debugger -> Run to run the program. In the Input/Output window, you should see the program print the number read from the switches. It will prompt you to enter a number. Type your number in the User Input field at the bottom of the Input/Output window. The program will display the number from the DIP switches in the least significant nibble of the LEDs, and the number you entered in the most significant nibble. It will then pause for 3 seconds (3000 ms), then repeat. Try entering different numbers and changing the value on the DIP switches. Check that everything works as you expect. The console is not entirely reliable at printing messages; it will sometimes print part of the message late and sometimes skip printing part of the message. **Choose Debugger -> Halt when you are done.**

The next step is to learn to use the debugger to trace through your program. Choose View -> Watch to set some variables to watch. In the box at the top center, type `sw` and hit enter, then click Add Symbol. Do the same for `num`.

Click the cursor on the `initIO()` line in lab6tut.c. Right click and choose Run to Cursor. The program will start up and run to this first line. Choose Debuggger -> Step Into (F7). You can watch the program step through the `initIO` function by repeatedly pressing F7. Do the same with `readSwitches`. Check that `sw` gets the value you expect in the Watch window. Choose Debugger -> Step Over (F8) to step over the `printf` statements. Stepping into them would make no sense because you don't have the source code to watch how they work. If you accidentally do step into a library function like `printf`, put the cursor on the next line and choose Run to Cursor again. Step over the `scanf` statement and enter a value. Check that num gets this value. Step into the writeLEDs function. Add `val` to the watch Window. Check that it has the value you expect when you step into the function. Note that `sw` and `num` will indicate Out of Scope because they are not defined within `writeLEDs`.

To start the program again, choose Debugger -> Reset -> Processor Reset (F6). Then step through the program again.

Finally, learn to modify the program. Comment out the line with the delay. Recompile your code and program it over the ICD. Test it again and observe that there is no delay.

You now know your way around the C compiler and in-circuit debugger. In the next sections, you can write some programs of your own.

Pocket Hypnotizer

You have been dispatched to the Atacama desert to obtain secret information from a rebel leader. You'll have to get past the border guards to reach your destination. For this mission, you need to build a pocket hypnotizer.

Create a new project called lab6ph_xx. Remember to set the Build Options and choose the ICD3 as the debugger. Create a new file called lab6ph_xx.c for your program. Remember to add e85.c to your project and to call `initIO()` before running the rest of your program.

Write a program that causes a pattern on the LED bar to zip back and forth. When your program starts, it should turn on one of the LEDs. Then it should turn off that LED and turn on the next. Continue until reaching the end of the LED bar, then go back, then repeat indefinitely. You'll need to choose a suitable delay between steps to get the desired effect.

Remember to use Build All, Program, View Application In/Out, and Run to test your program. If you have difficulties, use the debugger and watch window to step through your code and compare against your expectations. Tune the delay until it looks mesmerizingly good. Stare into the blinking lights as you repeat to yourself "I will ace E85."

Fibonacci Numbers

Your next goal is to calculate and print the first 16 Fibonacci numbers to the screen. Recall that each number in the Fibonacci series is the sum of the previous two numbers. Table 1 lists the first few numbers in the series.

n	1	2	3	4	5	6	7	8	9	10	11	…
fib(n)	1	1	2	3	5	8	13	21	34	55	89	…

Table 1: Fibonacci Series

We can also define the fib function for negative values of n. To be consistent with the definition of the Fibonacci series, what would the following values be?

fib(0) = ____

fib(-1) = ____

These values are useful when writing a loop to compute fib(n) for all non-negative values of n.

Create a new project called lab6fib_xx. Create a new file called lab6fib_xx.c and write your program. Remember to add e85.c to your project and to call `initIO()` before running the rest of your program. Compute and print the Fibonacci numbers for n = 1…16.

Number Guessing Game

Your final project is to write a game to guess a random number between 1 and 100. The game should play something like the one below, with bold indicating user input.

```
I'm thinking of a number between 1 and 100.
Your guess: 40
Too high.  Try again.
Your guess: 20
Too low.  Try again.
Your guess: 33
You got it in only 3 guesses!
```

Call your project lab6guess_xx.

You'll find the `rand()` function to be helpful. It returns a pseudorandom integer. You'll need to `#include <stdlib.h>` to use the function.

Unfortunately, it uses the same random number seed each time your program starts, making the game rather boring to play more than once. The `srand()` function changes the random number seed. For extra credit, develop a way to make your game less predictable.

What to Turn In

Include each of the following items in your submission **in the following order**. Clearly label each part by number.

1. Please indicate how many hours you spent on this lab**.** This will be helpful for calibrating the workload for next time the course is taught
2. Your neatly written, commented lab6ph_xx.c file. Did it work?
3. Your lab6fib_xx.c file. What is the 16th Fibonacci number that it calculates?
4. Your lab6guess_xx.c file. Does it work?
5. What were the most difficult bugs you encountered and how did you fix them?

Hints

If you are having trouble, check for the following common problems:

- Be sure your files are on your Charlie H drive and that there are no spaces or special characters in the file name.
- Be sure you've included e85.c in each of your projects.
- Remember to follow each of the steps below to compile and test a new program:
  - Check that the current program is halted
  - Build All
  - Program
  - Run
- If `printf()` or `scanf()` isn't working correctly, check that:
  - The compiler settings should include `-g -mappio-debug`
  - Your program must call `initIO()`
  - View Application In/Out and set the format to text
  - Even with this, part of the print statements may not display until the next print is called – and sometimes they may not display at all. This is due to a bug in the Microchip software and isn't your fault.