Chapter Four: Programming the C167

1. C167 Programming Tools

The Keil IDE (Integrated Development Environment) is used to program the C167 in the C programming language. It writes to the board RAM an executable version of your code. This tool is used to design and test your program. Once you are satisfied it is working correctly, you can "permanently burn" the code onto FLASH using Phytec FLASH tools. But, FLASH is reprogrammable so you can erase FLASH memory using the software and download new code; the number of times this can be done is around 100,000.

2. Embedded Systems Programming Concepts

You MUST remember, you are writing code for a board that has no monitor, no keyboard or hard disk. So, it may feel weird at first. Nevertheless, such code is usually very clean and simple as you will see.

This kind of programming is called *embedded systems programming*. You are writing code that is going to run on an independent system, the CalBOT. The system need not be a robot, for instance your microcontroller may control the flight control systems of a Boeing 747. Hence, unlike writing code on a computer, you have to make SURE your program is not buggy. For instance, if your flight controller program crashes, the outcome could be disastrous!

3. Downloading the Skeleton Project File

Lets start writing programs! We have already configured Keil for you and made a skeleton project file that you can use when writing programs for the C167. There is a link in the Robotics Notes web page (http://www-inst.eecs.berkeley.edu/~ee40/calbot/webpage/index.htm) to the CalBOT project skeleton. But, first:

1. Create a new directory in your **network home directory**[1]. Name it 'CalBOT_project_skeleton'.
2. Download all the files under the CalBOT_project_skeleton link in the Robotics Notes web page. Make SURE you save the files as of type 'All files'.
3. Again, if you have any questions, email me (mbharat@cory.eecs.berkeley.edu), ask your TAs, ask your peers etc etc.

[1] This is NOT the desktop. You need to save it under your login in the network drive (usually the U drive).

4. Understanding the Keil IDE

First, start up Keil. Double click on the  icon on your desktop. Then:

1. Click on **Project** and then **Open Project**. Navigate to the CalBOT_project_skeleton directory and choose the CalBOT project.

You should see the screen in figure 1.

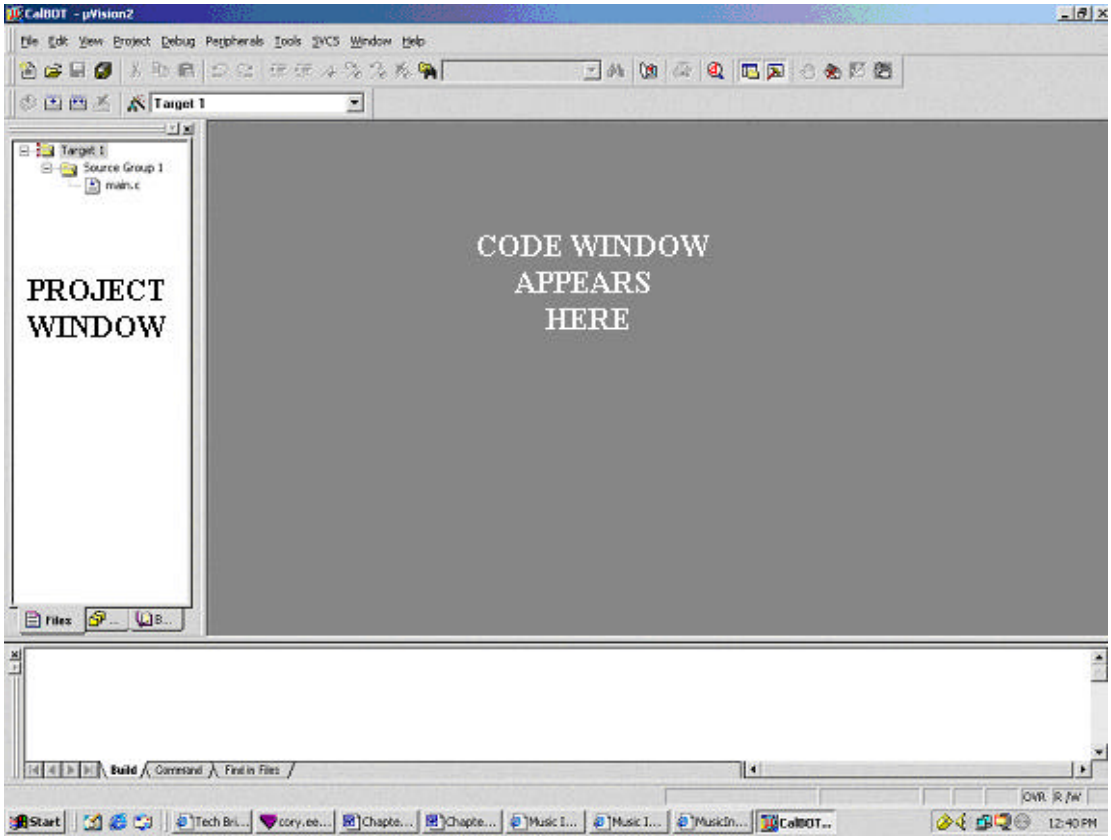

Figure 1. The Keil IDE startup screen

The two important windows are shown in figure 1. The PROJECT WINDOW displays the list of files in the project. The CODE WINDOW shows the code you are currently working on. Double-click on main.c to open the file in the code window. The code for main.c is shown below.

```
/* Header files */
#include <reg167.h>                                          (1)

/* Global variables and defines */


/* Initialization functions */
```

```
void InitLEDs(void) {
      DP2 = 0xFFFF; /* Turns on all LEDs */                (2)
      P2 = 0x0000;                                         (3)
}

void Initialize(void) {
      InitLEDs();

      IEN = 1;          /* Enable Interrupts */            (4)
}

/* Utility functions */

void kludgyWait(void) {                                    (5)
      float i;                                             (6)

      /* Approx. 1/2 sec delay */                          (7)
      for(i = 0;i < 2500;i+=0.1) {                         (8)
            ;
      }
}

void main(void) {                                          (9)
      Initialize();

      for(;;)                                             (10)
            ;
}
```

Program 1. Turning LEDs on

The numbers in parentheses on the right of the code are line numbers inserted by me for reference. Look at lines (2) and (3). These two lines are fundamental to programming the C167[2] and it involves a central microcontroller concept called ports. Figure 2 shows how ports help a microcontroller talk to the outside world.

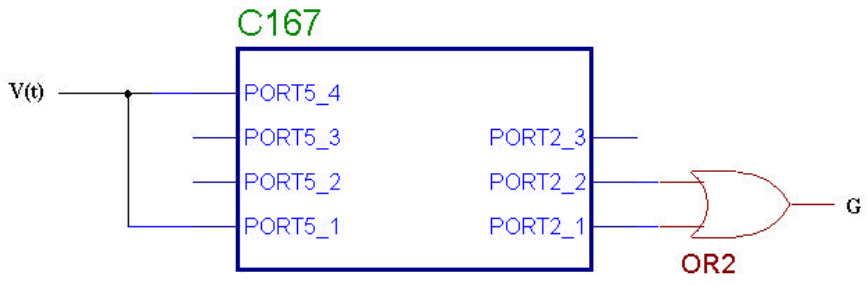

Figure 2. Microcontroller ports

[2] Refer to appendix A if you do not understand hexadecimal and binary numbers

5. Ports and C Programming

You can think of a port as a gateway for data flow (hence the name, "port"). Data can either come into the microcontroller from a port, or flow out of a microcontroller through a port to the outside world. For instance in figure 2 above, port 5 is configured as an input port and port 2 is configured as an output port. Data comes into the microcontroller as a time varying voltage, V(t). Analog to digital converters (A/D converters) in the C167 convert this analog voltage into digital data. The C167 then operates on this digital data and the result is sent to an OR gate through port 2.

What does PORT5_1 mean? Data in the digital world simply consists of two numbers: 0 and 1[3]. The 0 usually refers to a switch (say a transistor) being off (that is, connected to the lowest voltage in a circuit, usually ground) and the 1 refers to a switch being turned on (say a transistor that is connected to the highest voltage in a circuit).

So, PORT5_1 refers to bit 1 on port 5 in the C167. The maximum number of bits within a port in the C167 is 16 (hence the C167 is a 16-bit microcontroller). Notice that this is the maximum number, some ports in the C167 have only 8 bits (Port 8 and Port 7). Some ports are special and can handle only specific types of data, for instance Port 5 can handle only analog data. Chapter 7 has a list of which ports on the C167 are available to the user.

How do you say if the port receives data or sends data? By setting a direction port. For instance, in program 1, DP2 is the direction port for port 2. Setting a bit to 1 in DP2 tells the C167 that the corresponding bit in port 2 is used to be used as an output port, that is send data only.

Hence, referring program 1, line (2):

```
DP2 = 0xFFFF
```

tells the C167 that all the 16-bits in port 2 are to be used as output and line (3)

```
P2 = 0x0000
```

tells the C167 that all the 16-bits in port 2 have a value of 0.

Before we move on, look closely at figure 2. Can you spot the redundant component in figure 2? How could you avoid using it?

[3] Yes, your computer works because a bunch of 0s and 1s happen to be at the right place at the right time.

The OR gate in figure 2 is redundant. Instead, you could just write an if statement in your C program to test if Port2_1 OR Port2_2 is logic 1. (more on the if statement in chapter 5). This is the main reason for using digital control – avoiding hardware redundancy. Suppose you wanted to check for the condition: Port2_1 AND Port2_2. If you had used the external OR gate, then you probably would have to unsolder the OR gate and put an AND gate in its place. With software its easy, you just change the clause in the if statement to an AND.

The rest of program 1 illustrates some other concepts of C programming. First almost every line in C should end with a semicolon, the exceptions are loops, function implementations and preprocessor directives. Do not worry about these terms, they will become clear as you read this manual.

Second, blocks of C code are enclosed in curly braces { }. You don't have to worry about which blocks of C code to enclose in curly braces, it will become clear as you read the manual.

Line 1: #include <reg167.h> means include information about ports etc in your program. This helps you refer to port 2 as P2 and also has information about control and data registers. Control and data registers are the subject of chapter 5. This line is known as a PREPROCESSOR DIRECTIVE. You don't have to remember this term, just don't put a semicolon at the end of this line!

Line 4: IEN = 1 means enable interrupts. The topic of interrupts is beyond the scope of this manual. You don't need to understand this line to build the CalBOT, as a matter of fact you can build pretty advanced robots without interrupts. Nevertheless, for the curious an excellent reference is Patterson and Hennessy's *Computer Organization and Design*.

Line 5: void kludgyWait(void) is the start of a FUNCTION. Functions are used to split up your program into manageable pieces. The format of a function in C is:

```
Return-type functionName(arguments) {
        function statements
        ……
}
```

The functionName should be obvious; it can consist of upto 32 characters and must not begin with a number. A function takes some arguments, does something (hopefully useful) with them and returns a value. In our case, our kludgyWait function doesn't take any arguments and doesn't return any, hence the type **void**. We will cover functions in more detail in chapter 5.

By the way, I named my function kludgyWait because it doesn't wait exactly for 0.5 seconds. What do you mean by "waiting"?

It means if you run this function, it takes approximately 0.5 seconds for this function to finish executing. How does the function accomplish this? By using lines (6) and (8).

Line 6: float i is the DECLARATION of a VARIABLE. A variable (as the name implies) is used to hold different data at different times in your program. The declaration tells C what kind of data it can hold, here a **float** means the variable i can have a decimal number.

Line 7: /* Approximately ½ sec delay */ is a comment. Anything between a /* and */ is ignored by C, feel free to use them to make your program more understandable.

Line 8: for(i = 0;i < 2500;i+=0.1) is a LOOP in C. A loop (as the name implies) tells C to repeat the statements in the loop BODY (the lines following the loop and enclosed by { }) to be repeated. In this case, there are no statements in the body. This is indicated by the single semicolon.

How do you tell C how many times to repeat the statements in the body? This is accomplished by the loop statement. You read the statement as:

1. Start with i = 0
2. keep executing the body of the loop as long as i < 2500.
3. Once you execute the body once, increment i by 0.1.

Number 1 is called the initialization step, number 2 is called the conditional and number 3 is called the increment.

So, back to our original question: how does the kludgyWait function wait? Well, it takes time to increment i by 0.1 until we reach 2500, this time is approximately 0.5 seconds. Hmmm, it takes 0.5 seconds for a 20 MHz controller to count to 2500, 1/10 step at a time? Well, i is a float variable. Floating point calculations are EXTREMELEY slow.

Line 9: void main(void) is the start of the main function. The main function is special; C begins executing your code immediately after this statement.

In our case, the main function simply calls another function Initialize(). After this is the last statement in your embedded program.

Line 10: for(;;) is an INFINITE loop. It tells C to keep on executing your loop statements forever. Now why on earth would you want to do that?

Remember, you are writing an embedded program. Your code is going to execute on a microcontroller that may then be used in a Boeing 747 flight control computer. So, you want to be absolutely sure that your program doesn't stop executing.

6. The KitCON-167

Great. What the heck does this program do? As I mentioned in chapter 2, the KitCON-167 is a like a "motherboard" your C167. It contains a plethora of components like RAM, serial ports and power circuitry that make your microcontroller tick.

Among these components are 18 LEDs (light emitting diodes) out of which you can use 16[4]. Figure 2 in chapter 2 shows the location of these LEDs, which I refer to as "status LEDs". As the name implies, an LED emits light when turned on. What makes the 16 status LEDs so special? They are connected to port 2 of the C167 through the KitCON connector!

Aha, this program simply turns on the LEDs connected to port 2. So, why do we write a 0 to the Port 2 to turn the LEDs on in program 1? Well, the designers of the KitCON-167 decided to make these LEDs turn on when you write a 0 to Port 2, NOT a 1. Why, I don't know! So, in program 1 we write a 0 to Port 2 to turn on all the LEDs.

8. Downloading a Program to the KitCON-167

What do you do to put this program on the KitCON-167? First, you have to *compile* the program: send it through Keil's C compiler that translates your program into object code that is then translated by a linker into executable code.

It is very easy to do this, just click on **Project** as shown in figure 3. Then click on **Rebuild all Target files**. You should not receive any warnings or error messages. If you do, contact your TA, ask your peers or email me (mbharat@cory.eecs.berkeley.edu).

In order to actually download the program, make sure your serial cable is connected to the RS-232 (serial interface) port of the KitCON, refer to figure 2 in chapter 2. Once you have done this, turn on the KitCON-167. Insert the adaptor to the connector and make sure the dummy plug is removed, as shown in figure 4. Once you remove the dummy plug, the power LED should light up as shown in figure 5.

Now, click on **Debug** followed by **Start/Stop Debug Session**. You should see a blue progress bar at the bottom left corner of your screen. Once the program has finished downloading to the KitCON RAM, you will see figure 6. Click on **Debug** again and click **Go**. The LEDs should light up as shown in figure 7.

[4] The other two are used to indicate if your board is on and if it is connected to your computer.

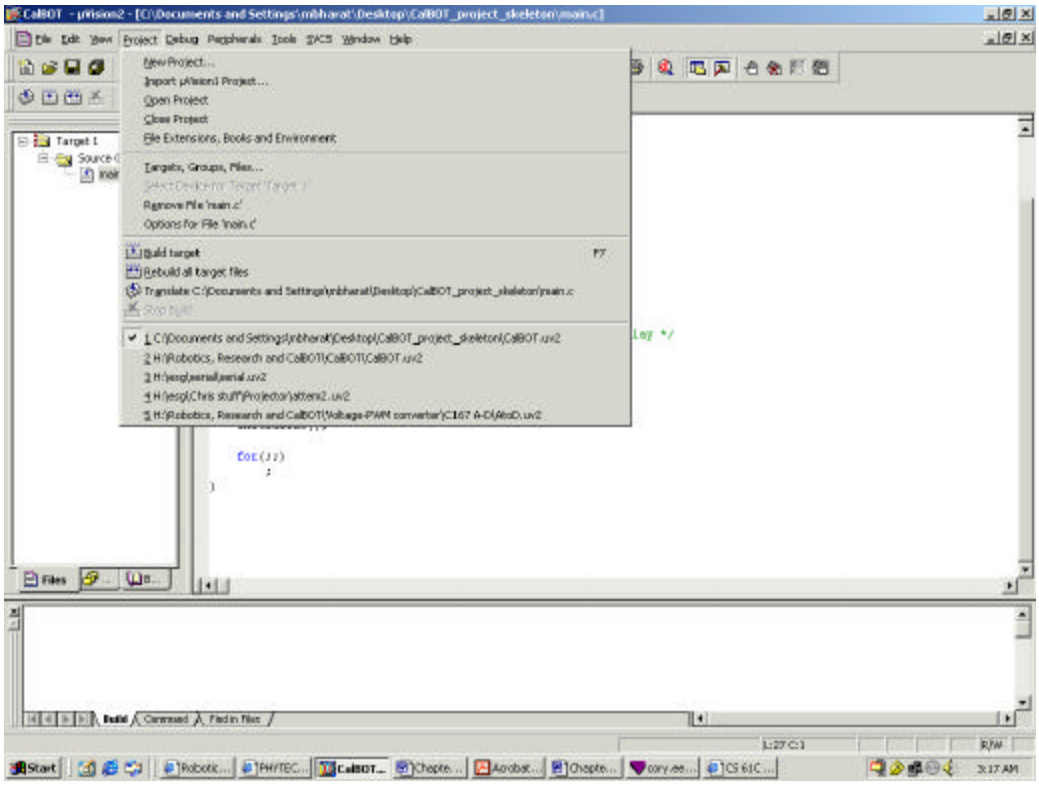

Figure 3. Compiling and linking your program

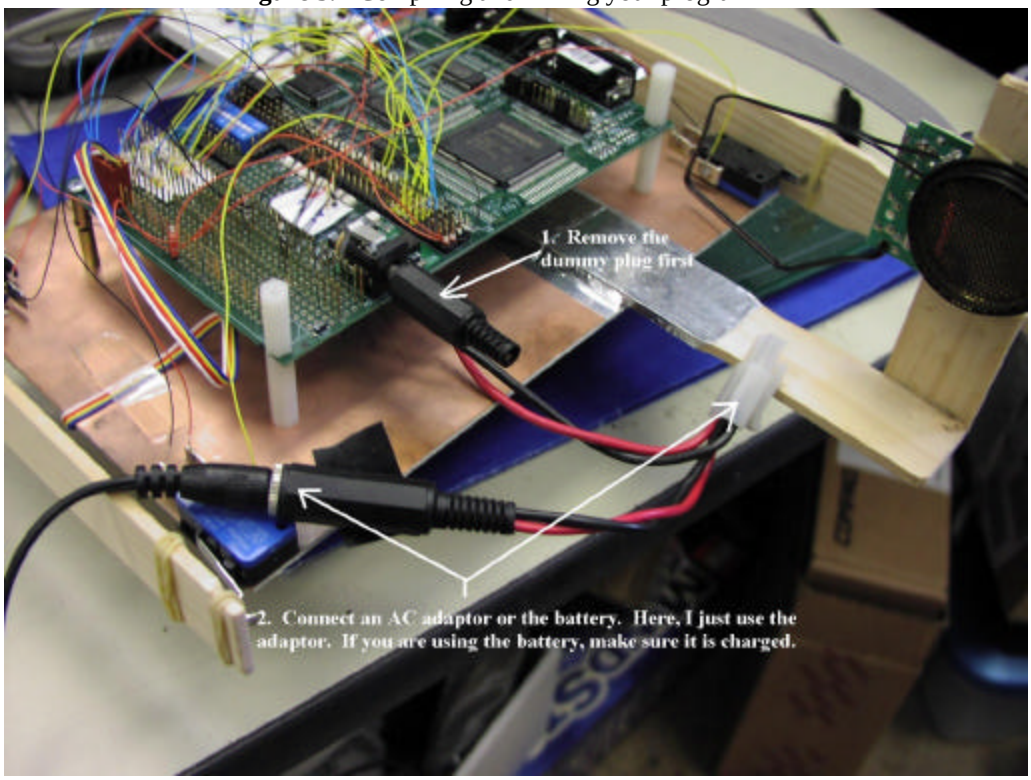

Figure 4. Power supply connections

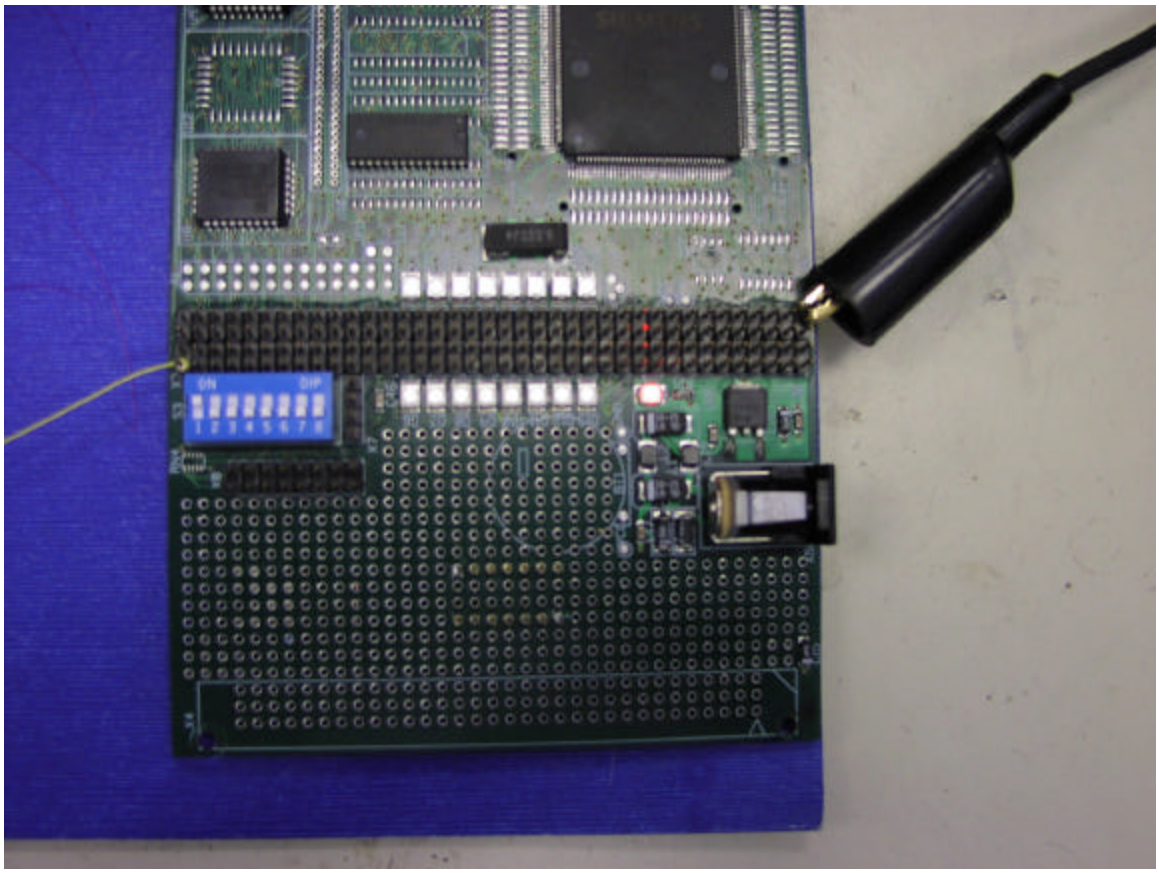

Figure 5. The red power LED turns on indicating the board is receiving power

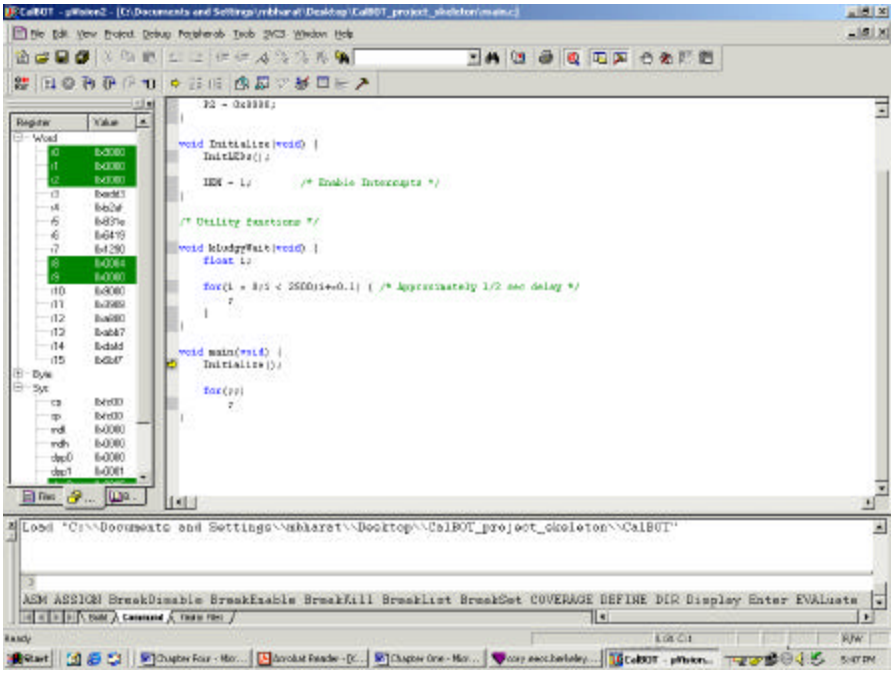

Figure 6. Keil ready to run your program

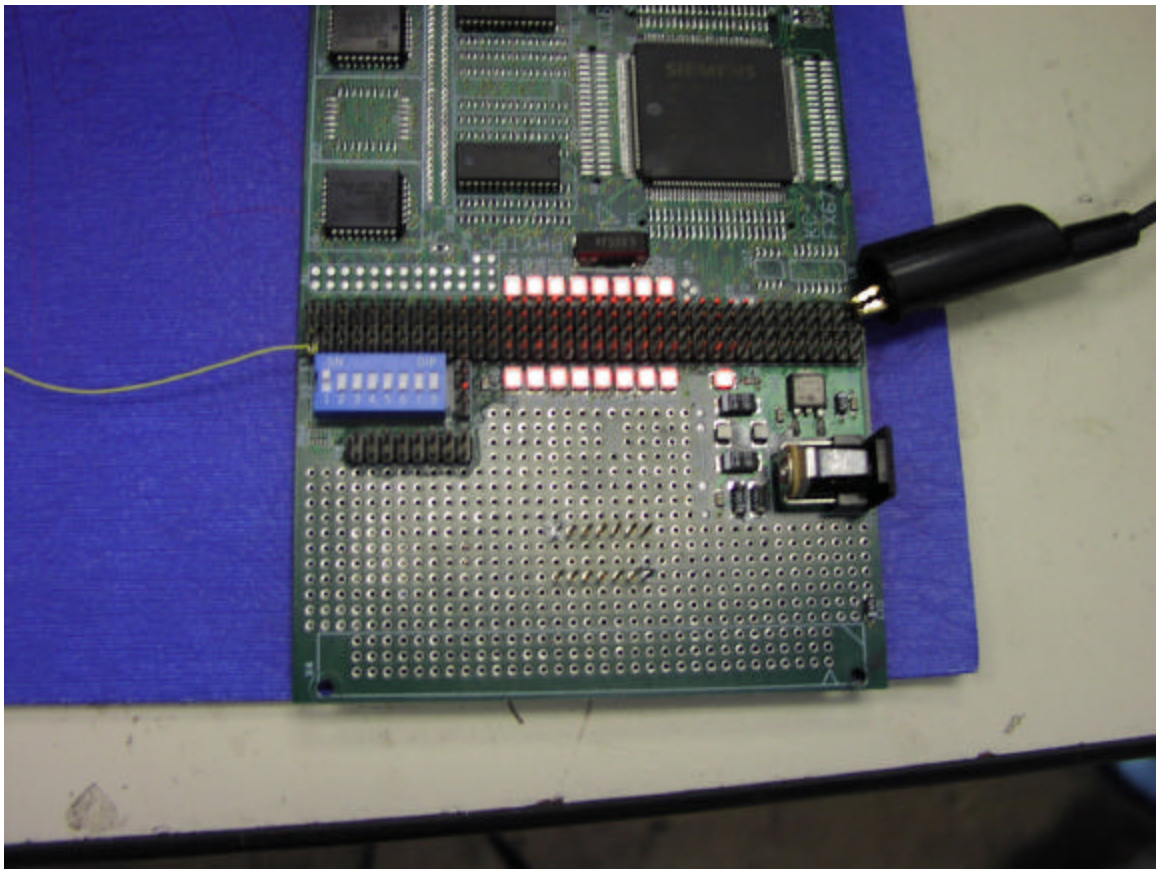

Figure 7. LEDs all lighting up!

8. Summary

This chapter covered a lot of ground. In order to fully understand these concepts, I suggest the following:

1. Learn how to turn on and off each LED in port 2.
2. Use the kludgyWait function to write a program that blinks the port 2 LEDs.

In the next chapter, we will add more tools to your microcontroller know-how, namely using control and data registers, using PWM to control the speed of the motor and the concept of a motor driver. After completing chapter 5, you should be able to control the speed and direction of your motor.