Lab 2: Simon Says Game

EE 234: Section 2

Michael Mentele

Partner: Jonathan Quissenberry

September 11, 2012

*Abstract*

The purpose of this report is to explain code written for a Simon Says game using peripheral modules on the Cerebot MX4cK board. A predetermined sequence of LEDs are turned on and off for a total of eight on and eight off states. The user must then replicate the LED sequence using the four switched provided on the switch module. This is done by mapping the state of SWm to LEDm-1 using read/write operations on programmable pins in Port B.

**Introduction**

In this lab we are exploring the functionality of the Cerebot MX4cK peripheral I/O. The pins are programmable as inputs and outputs. Using read and write operations we can do useful work by mapping our inputs to certain outputs. The purpose of the lab is the exploration of the boards method of communication with the outside world (ports).

The application of the material in this lab is production of a simplified Simon Says game that plays a sequence of LEDs and then waits for the user to replicate the sequence. To do this it uses peripheral switches as inputs and peripheral LEDs as outputs. The table below shows the sequence that will be played and that the user will have to perform.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **LD0 (on)** | **LD1 (on)** | **LD2 (on)** | **LD3 (on)** |
| **SW1** | On | Off | Off | Off |
| **SW2** | Off | On | Off | Off |
| **SW3** | Off | Off | On | Off |
| **SW4** | Off | Off | Off | On |
| **SW1** | On | Off | Off | Off |
| **SW2** | Off | On | Off | Off |
| **SW3** | Off | Off | On | Off |
| **SW4** | Off | Off | Off | On |

**Table 1: Peripheral Switch and Peripheral LED Sequence Mapping**

Figure 2 shows which bits are mapped to the switches and on-board LEDs. Using this information I know where to read the value that I will be writing to the peripheral LEDs.

|  |  |  |
| --- | --- | --- |
| **Cerebot Pin** | **MCU Port/bit** | **Switch** |
| JK-01 | RB10 | SW1 |
| JK-02 | RB11 | SW2 |
| JK-03 | RB12 | SW3 |
| JK-04 | RB13 | SW4 |

**Table 2: Port to Switch Mapping**

Table 3 illustrates (similarly to Table 2) which bits in Port B map to the LEDs on the peripheral LED unit. Knowing the bits I can write to enables me to pass the state of the switches to the corresponding LEDs.

|  |  |  |
| --- | --- | --- |
| **Cerebot Pin** | **Port/bit** | **Peripheral LED** |
| JJ-01 | RB0 | LD0 |
| JJ-02 | RB1 | LD1 |
| JJ-03 | RB2 | LD2 |
| JJ-04 | RB3 | LD3 |

**Table 3: Port to Peripheral LED Mapping**

|  |
| --- |
| **Software Design**  **C:\Users\cmentele\Desktop\EE234\Mike 234\Labs\Lab2\Lab2DONE.jpgDesign Overview**  The program plays the following LED sequence: LD0, LD1, LD2, LD3, LD0, LD1, LD2, and LD3. Illuminated LEDs are de-asserted between LED assertions.  After the sequence is played the user must input the same sequence using the peripheral switches. The program checks that sequence against the played sequence.  Figure 1 illustrates the functionality of my program. In simple terms it plays the sequence and then checks by state for it to be replayed exactly by checking sequentially which switches are turned on and off. It does this in real time meaning you get immediate feedback when incorrect.  If it is misplayed by the user it displays an “error” sequence of three flashes of all four LEDs. When all sixteen states are successful it plays a “success” sequence of of 0&1 the 1&3 and finally 1, 2, 3, and 4.  **How it is Accomplished**  There are sixteen indefinite while loops nested within a master while loop that counts to sixteen by incrementing within each sub loop. If the wrong switch is pushed on or if two are turned on or otherwise acts outside of the explicit delineation provided then the program displays an “error” sequence where all four LEDs flash three times. It then sets the counter of the master while loop to 17 ending the master loop.  If the user successfully repeats the sequence then the board plays a “success” sequence where LEDs 0 and 2 and LEDs 1 and 3 alternate flashing twice and then LEDs 0-3 turn on and off in succession. |

**Hardware Design**

1. Plug the USB cable into the Cerebot MX4cK “DEBUG”port and into a USB port on your computer.
2. Plug in the switch module into the JK-01:04 connector of the Cerebot board.
3. Plug the peripheral LED module into the JJ-01:04 connector.
4. Turn on the power by flipping the switch adjacent to the “DEBUG” port.

**Test Procedure and Results**

**Procedure**

1. Test correct sequence

This tests that it can run in the “correct” manner successfully. There is only one case that is correct so it is simplest to test and make a good starting point.

1. Test incorrect sequences

Here we evaluate whether the program realizes when the sequence has not been played as specified. There are many cases where this is possible. The wrong switch being asserted or not being de-asserted is on such case.

Typically this is where the issues would come up until my final iteration of code. One should check that an error is displayed at any deviation from the sequence.

1. Let someone else play it!

Using Murphy’s law… and another perspective this is an effective way to achieve unexpected results.

**Results**

Testing is part of an iterative design process. My results are as expected after several code adjustments; the LEDs played in the correct order and when you replicated the sequence the board displayed the success sequence. When deviated from the specified path an error display was displayed on the LEDs.

**Methodology**

Each function was tested before implementing overall functional testing. In this way I narrowed down what the possible sources of problems could be. The idea is the shortest feedback time will allow you to find root cause efficiently.

**Answers to Lab Questions**

1. Purpose of some .h files contained in the MPLAB library.
   1. **Alloca.h**

Defines the macro alloca(). This function allocates space in the stack frame of whatever your caller may be. Alloca() returns a pointer to the beginning of the stack that was allocated. The space is then freed when the function that called alloca() returns to *its* caller. You cannot take the address of this function nor can you change its functionality.

* 1. **Assert.h**

Defines the preprocessor macro “assert().” This means that the program makes assumptions about unspecified conditions. When you run it if your expression is 0 it stores information for a diagnostic message on stderr and then call the abort() function. Information it stores in stderr is: the source filename, the source line number, the source function, and the text of the expression

That evaluated to 0.

* 1. **Conio.h**

This header provides many functions for console I/O. You can use functions within this header to manipulate a screen or monitor, move text, or check buttons states.

* 1. **Ieee754.h**

This specifies a format for representing floating point numbers. It has three parts:

1. A sign bit
2. An order of magnitude (exponent)
3. And a mantissa specifying the number of digits.
   1. **Errno.h**

Defines several macros that are used to define and track “runtime errors.” The definitions are used with the perror() function and include an external integer named errno. The external integer is assigned a value from many C library functions when an error occurs during function execution. Errno also defines many constants to describe errors.

2)

**Conclusion**

The project worked as I wanted although there are all sorts of features I wish I had time to add. From here the next step I would take is being able to first program the sequence into the board with the switches so that the user could change the sequence. Something like Btn 1 would send the user to settings mode and Btn 2 would be play mode.

**Feedback**

Exact project specifications would be nice to clarify the project. Some kind of standard that we could hold our project to and be able to measure how well we did. I just wanted to know what functionality my work was being graded on.

**Appendix**

**A. Pre-Lab**

1. PortA => 12 Bidirectional Pins
2. PortB => 16 Bidirectional Pins
3. PortC => 8 Bidirectional Pins
4. PortD => 16 Bidirectional Pins
5. PortE => 10 Bidirectional Pins
6. PortF => 11 Bidirectional Pins
7. PortG => 10 Bidirectional Pins and 2 Input Pins

Registers are the quickest memory device. The function of a control register specifically is to control the pin settings. Registers configure the I/O pins for the desired application.

The three port registers are the TRIS, PORT, and LAT registers.

The TRIS (tri-state) registers determine whether a discrete PORT I/O pin is an input or an output. If you set a TRIS bit to ‘1’ then the pin is an input and if you set it to ‘0’ it is an output. By default all I/O port pins are set as inputs.

The PORT registers allow your I/O pins to be ‘read.’ When you write to a port register you essentially are writing to your PORT data latch or LAT register.

LAT registers (PORT data latch) contain date that you write to your I/O pins. It reads the values from your PORT registers.

**B. Code Listing**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Programmer: Michael R. Mentele \***

**\* Class: EE 234\_2 \***

**\* Lab Project: Lab 2 \***

**\* Date: 09/04/2012 \***

**\* \***

**\* Description: This program runs a simple Symon Says game \***

**\* \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// #include all necessary standard and user-defined libraries**

**#include <plib.h> // Includes all major functions and macros required to develop**

**// programs for the PIC32MX4**

**// Yes, don't forget your prototypes**

**// Prototypes go here, or in a .h file, which you would also need to #include**

**void setup\_LEDs (void);**

**void setup\_switches (void);**

**void delay (void);**

**void play\_sequence (void);**

**void error (void);**

**void success (void);**

**//First two loops are repeated eight times. Checks for ON followed by OFF until it reaches the 16th check where if correct plays the "success" sequence.**

**int main (void)//check sequence**

**{**

**//initiate variables**

**unsigned int switch\_states = 0;**

**unsigned int iteration = 0;//Place in loop so it knows when to end**

**while (iteration < 16)**

**{**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1) //wait for user to turn a switch on**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;//get switch states**

**//Is the right switch on?**

**if (switch\_states == 0x00000400)//Yes so turn LED on**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);//Pass SW value to LED (ON)**

**iteration++;//track place**

**break;//next loop**

**}**

**else if (switch\_states != 0)//No so return error!**

**{**

**error();//play error light sequence**

**iteration = 17;//go to end of main on break**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;//get switch states**

**//Did the switch that got turned on get turned off?**

**if (switch\_states == 0x00000000)//Yes so turn LED off**

**{**

**PORTWrite (IOPORT\_B, 0);//Turn LED (OFF)**

**iteration++;//track place**

**break;//next loop**

**}**

**else if (switch\_states != 0x00000400)//No they turned someother switch on**

**{**

**error();//play error sequence**

**iteration = 17;//go to end of main on break**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000800)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00000800)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00001000)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00001000)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00002000)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00002000)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000400)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00000400)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000800)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00000800)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00001000)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0x00001000)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00002000)**

**{**

**PORTWrite (IOPORT\_B, switch\_states>>10);**

**iteration++;**

**break;**

**}**

**else if (switch\_states != 0)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**while (1)**

**{**

**switch\_states = PORTRead(IOPORT\_B) & 0x00003C00;**

**if (switch\_states == 0x00000000)**

**{**

**PORTWrite (IOPORT\_B, 0);**

**iteration++;**

**success();//You made it to the last iteration huzzah!**

**break;**

**}**

**else if (switch\_states != 0x00002000)**

**{**

**error();**

**iteration = 17;**

**break;**

**}**

**}**

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**}**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: setup\_LEDs \***

**\* Date Created: 9/4/12 \***

**\* Date Last Modified: 9/4/12 \***

**\* Description: Enable LED pins as outputs \***

**\* Input parameters: None \***

**\* Returns: Nothing \***

**\* Usages: must be called once before writing to LEDs \***

**\* Preconditions: None \***

**\* Postconditions: Pins to the LEDs are output pins \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void setup\_LEDs (void)**

**{**

**// Setup the four periphial LEDs for write; output pins**

**// According to the Digilent Cerebot MX4cK Reference Manual**

**// LED1 -> PB0, LED2 -> PB1, LED3 -> PB2, LED4 -> PB3; NOTE: PB indicates PORTB**

**PORTSetPinsDigitalOut (IOPORT\_B, BIT\_0 | BIT\_1 | BIT\_2 | BIT\_3);**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: setup\_switches \***

**\* Date Created: 9/4/12 \***

**\* Date Last Modified: 9/4/12 \***

**\* Description: Maps the inputs to switches \***

**\* Input parameters: none \***

**\* Returns: nothing \***

**\* Usages: use once before reading from switches \***

**\* Preconditions: none \***

**\* Postconditions: Pins for the switches are input pins \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void setup\_switches (void)**

**{**

**// Setup the two on-board buttons for read; input pins**

**// According to the Digilent Cerebot MX4cK Reference Manual**

**// SW1 -> PB10, SW2 -> PB11, SW3-> PB12, SW4 -> PB13; NOTE: PB indicates PORTB**

**PORTSetPinsDigitalIn (IOPORT\_B, BIT\_10 | BIT\_11 | BIT\_12 | BIT\_13);**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: play\_sequence \***

**\* Date Created:9/4/2012 \***

**\* Date Last Modified: 9/4/2012 \***

**\* Description: Plays LED sequence to be replicated by user. \***

**\* Input parameters: none \***

**\* Returns: nothing \***

**\* Usages: use once before checking user sequence \***

**\* Preconditions: LEDs are set up \***

**\* Postconditions: none \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void play\_sequence (void)**

**{**

**// sequence is 1,2,3,4,1,2,3,4 corresponding to bits 0-3 and LEDs 0-3**

**PORTWrite (IOPORT\_B, BIT\_0);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_1);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_2);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_1);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_2);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, 0);**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: delay \***

**\* Date Created: 9/4/2012 \***

**\* Date Last Modified: 9/4/2012 \***

**\* Description: Sets appropriate delay period to be reused \***

**\* for other functions such as play\_sequence. \***

**\* Input parameters: none \***

**\* Returns: nothing \***

**\* Usages: Between assertion and deassertion of LEDs \***

**\* Preconditions: none \***

**\* Postconditions: none \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**void delay (void)**

**{**

**unsigned int count = 0;**

**while (count < 40000)**

**{**

**count++;**

**}**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: error \***

**\* Date Created: 9/4/2012 \***

**\* Date Last Modified: 9/4/2012 \***

**\* Description: Plays "error" sequence (All LEDs flash 4 times)\***

**\* Input parameters: none \***

**\* Returns: nothing \***

**\* Usages: Play when user input is incorrect \***

**\* Preconditions: setup LEDs \***

**\* Postconditions: none \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Put other function definitions below main (), or they**

**// may go in another .c source file; Functions most likely**

**// will include port and device setups/initalizations;**

**// Be sure to comment all functions with the above block**

**void error(void)**

**{**

**PORTWrite (IOPORT\_B, BIT\_0 | BIT\_1 | BIT\_2 | BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 & BIT\_1 & BIT\_2 & BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 | BIT\_1 | BIT\_2 | BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 & BIT\_1 & BIT\_2 & BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 | BIT\_1 | BIT\_2 | BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 & BIT\_1 & BIT\_2 & BIT\_3);**

**delay();**

**}**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Function: success \***

**\* Date Created:9/4/2012 \***

**\* Date Last Modified:9/4/2012 \***

**\* Description:Plays "success" sequence 0&2,1&3,1,2,3,4 \***

**\* Input parameters: none \***

**\* Returns: nothing \***

**\* Usages: Play if last the loop is reached and passes \***

**\* Preconditions: Setup leds \***

**\* Postconditions: none \***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Put other function definitions below main (), or they**

**// may go in another .c source file; Functions most likely**

**// will include port and device setups/initalizations;**

**// Be sure to comment all functions with the above block**

**void success(void)**

**{**

**PORTWrite (IOPORT\_B, BIT\_0 | BIT\_2);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_1 | BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0 | BIT\_2);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_1 | BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_0);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_1);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_2);**

**delay();**

**PORTWrite (IOPORT\_B, BIT\_3);**

**delay();**

**PORTWrite (IOPORT\_B, 0);**

**delay();**

**}**