Simon

Final Project Report
December 11, 2009
E155

Julia Karl and Kirsten McAfee

Abstract:

Simon is a game for children distributed by Milton Bradley. To play the game Simon, the player watches the buttons light up and then repeats the same pattern back by pressing the buttons. The game will consist of four buttons and four corresponding LEDs of different colors. The buttons are arranged in a square around an LCD display that shows the level and score. The first level of the game begins by lighting up one button, then the game waits for the user to press that same button. If the button is pressed correctly, the game will light up that same LED followed by another random LED, and the user must mimic that pattern. Each time the pattern pressed is correct, the level increases. When the level increases, an additional step is added to the sequence, so that the player has to press three buttons on level three. This continues up to level 16 at which point the user wins.

Introduction

The motivation for this project is to recreate the game Simon, which is distributed by Milton Bradley. Simon is based on the game Simon Says. In this game, the player watches LEDs light up and then the player mimics the same pattern back by pressing the corresponding buttons. If the player does not repeat the exact pattern back, they loose the game. A player who can complete all 16 levels of the game will win. The physical game will consist of four buttons and four corresponding LEDs of different colors. The buttons are arranged in a square around an LCD display that shows the level and score.

The first level of the game begins by lighting up one button; the game then waits for the user to press that same button. If the user is correct, the game will light up that same LED followed by another random LED, and then the user must mimic that pattern. Each time the pattern is pressed correctly, the level increases and is displayed on the LCD display. When the level increases, an additional step is added to the sequence and the player has to press two buttons on level two. This continues up to level 16 at which point the player wins.

The PIC controls the LCD display, keeps track of the level that the game is on, generates the random numbers for the sequence of LEDs for each game and does the timing for the LEDs.

The FPGA controls which of the LEDs light up when the random numbers are sent from the PIC. It stores the numbers that it receives and uses a finite state machine to check if the correct buttons are pressed by the player.

The following sections discuss how the LCD display was implemented and a more detailed discussion of how the PIC and FPGA work both together and separately.

Located below in Figure 1 is a block diagram of the final design.

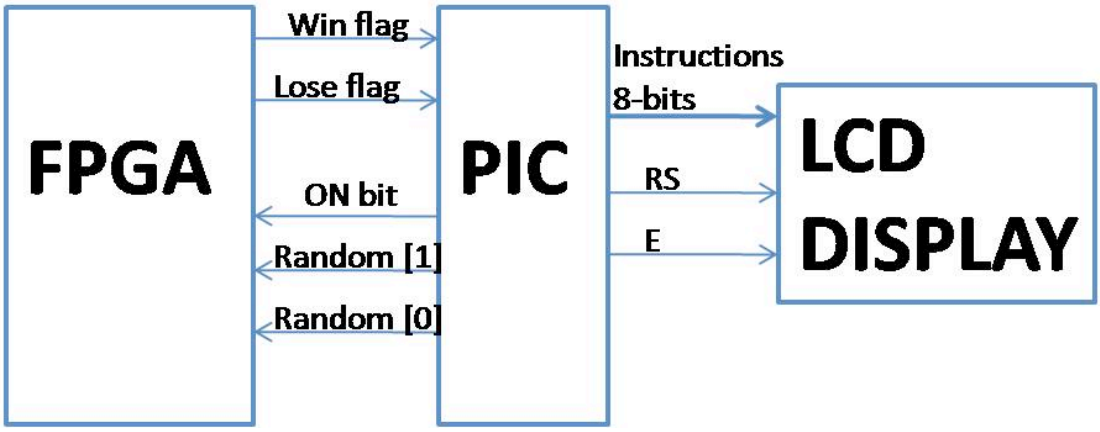

Figure 1 Block Diagram of Simon

LCD Display

The LCD used in this project is the CFAH1602JYYBJP. The data sheet can be found in References. The LCD has 16 pins. It communicates with the PIC via an 8-bit parallel connection. The PIC sends instructions as well as data using these 8-bits. The LCD uses 2 pins for power, 2 pins for backlight power, 1 ground, 2 to indicate mode, and 1 enable. The 8 pins used for data are connected to PORTD on the PIC and the enable and mode signals are connected to PORTC. The LCD is powered by 5V and both V_{dd} and V_0 pins are connected to 5V through a 1K potentiometer in order to control power and contrast of the display. The schematic in Figure 4 shows the connections.

To initialize the LCD, a series of instructions are sent to the LCD in order to configure its settings. For more information on initialization settings, refer to the data sheet. We initialized our LCD to have 8-bit instructions, 2 lines of display, and 5x11 pixel characters. Then the cursor is aligned to the first spot by sending the address of the DDRAM for that spot. Next the assembly code sets the table pointer to the program memory location defined in the C code for the corresponding message. The assembly code then sends the letters one-by one to the LCD to be displayed. Once a 0x00 is found, the cursor is aligned to the second line and the table pointer is set to the location of the second message and is displayed. The messages displaying the level and score of the player are put into the program memory in the C code and called as external variables in the assembly code. The assembly code is defined as a global function and called as an external function in the C code when the player advances a level or lose. A photo of the working LCD display can be seen below in Figure 2.

Figure 2 Working LCD display

User Interface

As part of the project proposal a casing for the game was needed. The box that that was created for the game is made out of sheet metal and was spot welded together. It has holes in the top for the buttons, LEDs and LCD display. It is a

10”x8.25”x3” box with a lid, which can be removed to access the breadboard inside. This box provides a user-friendly interface as well as portability for the game. Located in a photo below in Figure 3, is the final design in it’s casing.

Figure 3 Final design in casing

Schematics

Located below in Figure 4 is a full schematic of the final design of Simon. The FPGA and PIC are located in the Harris Board.

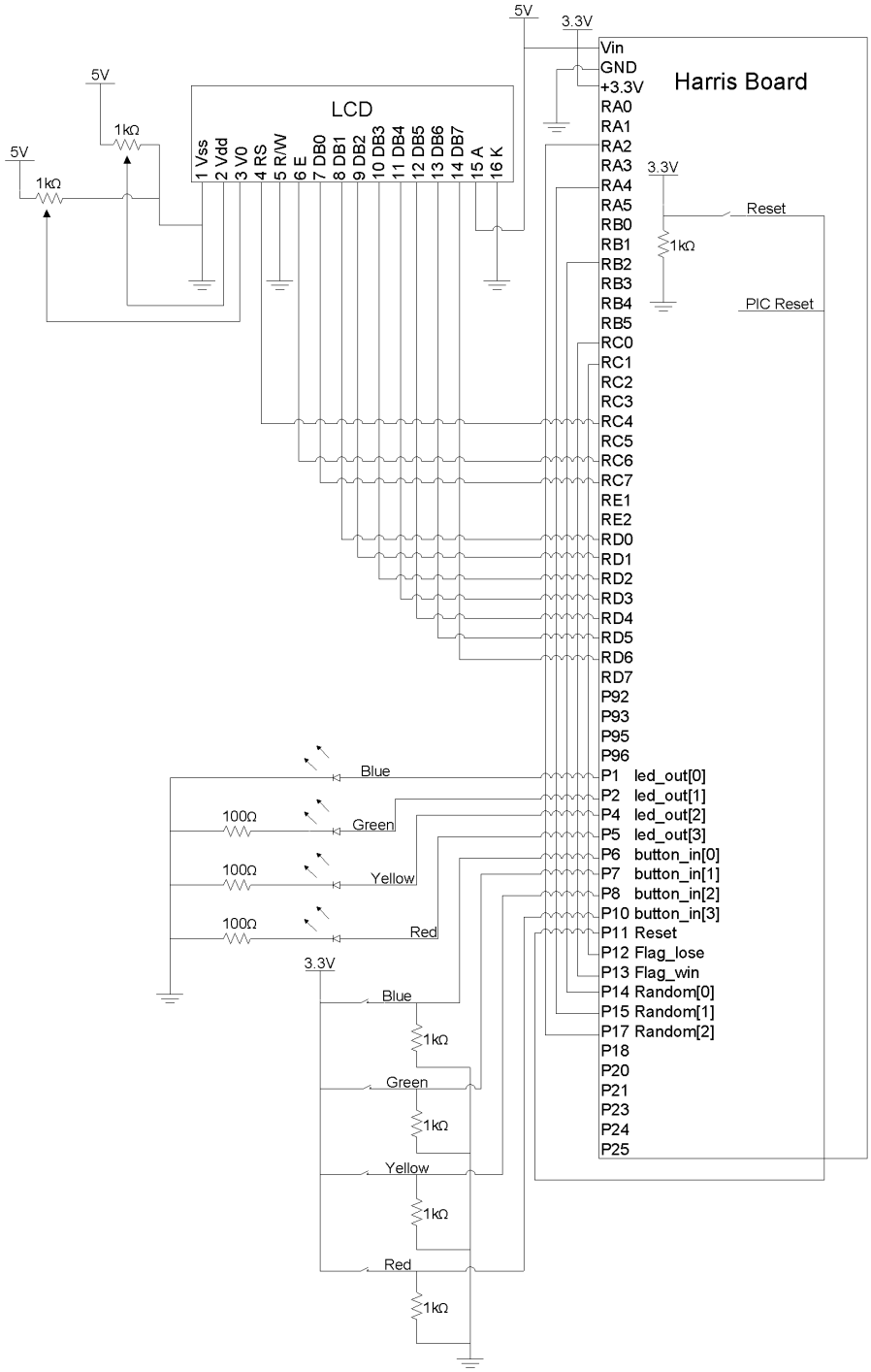

Figure 4 Schematic of Simon

Microcontroller

The main C code used to control the PIC microcontroller consists of 22 sub-functions. There are 18 sub-functions that put the messages into variables to be read by the assembly function controlling the LCD (16 for each level, a win message, and a lose message). One function is the external assembly function that turns on the LCD. The other three functions control the output to the FPGA. The C code first calls the pattern_init function that places 16 random integers into the data memory as a globally defined array called pattern. These 16 numbers correspond to the 16 levels for that game. The random number generator on the PIC was seeded using timer1. Next the main function calls disp_level1 and display_on to show "HI IM SIMON LEVEL 1 $:0" on the LCD display.

To start the game, the code calls pattern_out, which uses timer 0 to send two bits of a certain number of the random numbers in pattern depending on the level. Each output is sent for a certain amount of time along with an "on" bit to indicate to the FPGA that that LED should be lit. It then sends all zeros (turning the LED off) for the same amount of time before moving to the next number. Once the pattern is sent, the code enters a while loop to wait for the user to win or lose the level. level_inc determines the current level based on the win and lose input flags from the FPGA. Then the code displays the lose message, the win message, or the next level on the LCD depending on the output of level_inc. The code for the PIC is located in Appendix A.

FPGA

The FPGA code currently works off of six modules: a top module, really_slow_clock, slow_clk, leds, buttons and check. The top module connects all of the modules together. Located in Figure 5 below, is a block diagram of the FPGA code. Refer to Appendix B for the actual code.

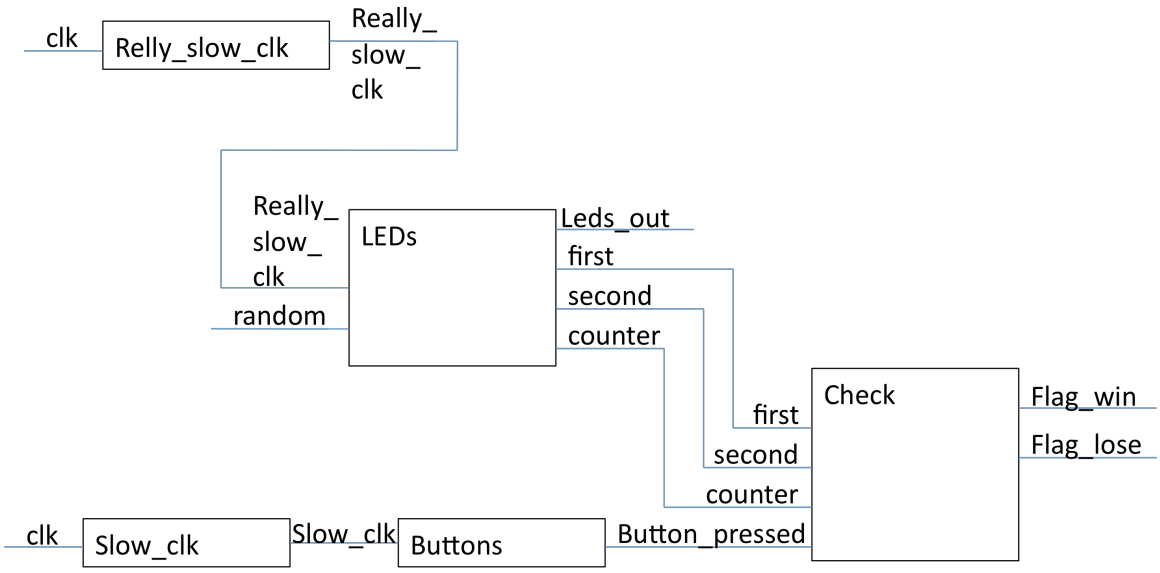

Figure 5 Block diagram of FPGA code

Slow_clock is the clock that is used to take the buttons input from the breadboard. This clock has to be slower to avoid switch bounce. The module buttons takes in the input of the buttons, and decodes it to figure out which button was pressed.

The check module is a very large state machine. Each level of the game has 2 states and there are 2 states for winning and losing the level. The first state of each level checks to see if a button is being pressed. If no button is being pressed, then the state stays the same, but if a button is being pressed, the state machine

determines if the correct button is being pressed. If the wrong button is pressed, then the state changes to the lose state. It is in this state that the lose flag is raised to alert the FPGA. If the button pressed is correct, the state machine moves to the next state where it checks if this is the last button in the sequence and waits until the player lets go of the button. If it is the last button in the sequence, the state machine jumps to the win state, where a flag is sent to the PIC.

The really_slow_clock module is what the leds module runs off of. The leds module takes the random numbers input from the PIC. This clock has to be slow to account for bouncing in the signals from the PIC. The leds module takes in bits from the PIC and turns the LEDs on or off. It also saves all of the previous values that the PIC has sent in order to determine what level the user is on. It does this by taking the first and second bit of each random number and putting them into the 16 bit registers first and second. These registers are outputted to the check module and it is how the check module determines if the correct button has been pressed. The level is also outputted to the check module.

Results

Our project was successful in that we created a playable Simon game and stuck to all of the original requirements of our project proposal. We ended up with a fully functional game that was playable up to 16 levels and had its own casing so it could be played outside of the lab. There are a few improvements that could be made. Currently the game is hooked up to an external power supply, it could be altered to run on a 5V wall transformer or batteries. Additionally we realized that

the game might be more fun if it was faster and if the LEDs lit up when the buttons were pressed.

The main difficulties of this project were coding the FPGA to check if the correct buttons were pressed and using the LCD assembly code in the C code. After multiple attempts at creating a compact finite state machine for the FPGA to check if the correct buttons were pressed, we realized that we could just have a finite state machine with many states. We ended up with a state machine that had two states for each button press and additional states for when the player loses and wins. Additionally, we had difficulty using the LCD assembly code in the C code. We wanted to take advantage of the assembly code provided on the Microtoys website (see references) but we had to change the message on the display at each level. What we ended up doing was altering the assembly code to be a global function and calling the function in the C code. Additionally, we created the messages and put them in the program memory in the C code and created global variables with their locations to be called in the assembly code.

References

LCD datasheet http://www.crystalfontz.com/products/1602j/CFAH1602JYYBJP.pdf
MicroToys LCD.pdf http://www4.hmc.edu:8001/Engineering/mocrotoys/LCD.pdf

Parts List

Bill of Materials
Prices Checked 12/10/2009

Description	Part Number	Manufacturer	Supplier	Supplier Part Number	Quantity	Unit Price	Total
Harris Board	N/A	Harvey Mudd College (HMC)	HMC	N/A	1	50.00	50.00
Blue LED	LVB3330	JAMECO VALUEPRO	Jameco	138691	1	0.99	0.99
Green LED	LTL4233	Lite-On	Jameco	1956653	1	0.09	0.09
Yellow LED	LTL307Y	Lite-On	Jameco	1956645	1	0.10	0.10
Red LED	LTL-307E-3	Lite-On	Jameco	2006730	1	0.10	0.10
Blue Button	30-784	Philmore Electronics	RVAC	PHI30784	1	2.87	2.87
Green Button	30-785	Philmore Electronics	RVAC	PHI30785	1	2.87	2.87
Yellow Button	30-781	Philmore Electronics	RVAC	PHI30781	1	3.05	3.05
Red Button	30-783	Philmore Electronics	RVAC	PHI30783	1	3.05	3.05
LCD Display	CFAH1602JYYBJP	CrystalFontz	CrystalFontz	CFAH1602JYYBJP	1	21.12	21.12
1 kΩ potentiometer	296UD102B1N	CTS Electrocomponents	Digikey	CT2262-ND	2	1.55	3.10
1 kΩ resistor	CFR-25JB-1K0	Yageo	Digikey	1.0KQBK-ND	4	0.32	1.28
100 Ω resistor	CFR-25JB-100R	Yageo	Digikey	100QBK-ND	3	0.32	0.96
Bread board	922318	3M	Digikey	922318-ND		48.19	48.19
White Button	30-10064	Philmore Electronics	HMC	Assorted Switches Bin	1	1.47	1.47
							139.24

Appendix A: Microcontroller Code

```
/*Julia Karl and Kirsten McAfee*/
/*created 11/08/09*/

#include <p18f4520.h>
#include <stdlib.h>

int pattern[16];
char row1_loc_low; //variables to be called in LCD.asm
char row1_loc_hi;
char row2_loc_low;
char row2_loc_hi;

rom char message1[] = "HI IM SIMON";
               // put messages into program memory
rom char level1[] = "Level 1 $:0 ";
               // to be called by table ptr in LCD.asm
rom char level2[] = "Level 2 $:5 ";
rom char level3[] = "Level 3 $:10";
rom char level4[] = "Level 4 $:15";
rom char level5[] = "Level 5 $:20";
rom char level6[] = "Level 6 $:25";
rom char level7[] = "Level 7 $:30";
rom char level8[] = "Level 8 $:35";
rom char level9[] = "Level 9 $:40";
rom char level10[] = "Level 10 $:45";
rom char level11[] = "Level 11 $:50";
rom char level12[] = "Level 12 $:55";
rom char level13[] = "Level 13 $:60";
rom char level14[] = "Level 14 $:65";
rom char level15[] = "Level 15 $:70";
rom char level16[] = "Level 16 $:75";
rom char gameover[] = "GAME OVER";
rom char win[] = "YOU WIN!";

extern void display_on(void);

int pattern_init(void) {
        int *patternptr;
        int i;
        patternptr = &pattern[0];
        srand( (unsigned int) TMR1L); //use TMR1L
        for(i=0; i<16; i++){
                pattern[i] = rand();
                        //put 16 random integers into pattern
        }
        //return pattern;
}

void pattern_out(int level){
        int i;
        for (i=0; i<level; i++){
                        // cycle though as many levels
                PORTAbits.RA4 = pattern[i];
                        // output two bits from random
                PORTBbits.RB2 = pattern[i]/3;
                PORTAbits.RA2 = 1;
                        // send bit to turn on LED
                TMR0L = 0x00;
                TMR0L = 0x00;
                while (TMR0L != 0xFF | TMR0H != 0x4F){
                }

                TMR0H = 0x00;
                TMR0L = 0x00;
                PORTAbits.RA4 = 0;                    // turn off LEDs
                PORTBbits.RB2 = 0;
                PORTAbits.RA2 = 0;
```

```
                while (TMR0L != 0xFF | TMR0H != 0x4F){
                }

                TMR0H = 0x00;
                TMR0L = 0x00;
        }
}

int level_inc(int levelflag, int loseflag, int prevlevel) {
        if (loseflag == 1) return 0;
        else if (levelflag == 1) return prevlevel + 1;
                        // if the level is won, move to next
        else return prevlevel;
}

void disp_level1(){             // display level one
        int mesptr1 = &message1;
        int mesptr2 = &level1;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level2(){             // display level two
        int mesptr1 = &message1;
        int mesptr2 = &level2;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level3(){
        int mesptr1 = &message1;
        int mesptr2 = &level3;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level4(){
        int mesptr1 = &message1;
        int mesptr2 = &level4;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level5(){
        int mesptr1 = &message1;
        int mesptr2 = &level5;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level6(){
        int mesptr1 = &message1;
        int mesptr2 = &level6;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}
```

```
void disp_level7(){
        int mesptr1 = &message1;
        int mesptr2 = &level7;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;

}void disp_level8(){
        int mesptr1 = &message1;
        int mesptr2 = &level8;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level9(){
        int mesptr1 = &message1;
        int mesptr2 = &level9;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level10(){
        int mesptr1 = &message1;
        int mesptr2 = &level10;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level11(){
        int mesptr1 = &message1;
        int mesptr2 = &level11;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level12(){
        int mesptr1 = &message1;
        int mesptr2 = &level12;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level13(){
        int mesptr1 = &message1;
        int mesptr2 = &level13;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level14(){
        int mesptr1 = &message1;
        int mesptr2 = &level1;
        row1_loc_low = mesptr1;
        row1_loc_hi = (mesptr1&0xff00)/256;
        row2_loc_low = mesptr2;
        row2_loc_hi = (mesptr2&0xff00)/256;
}
```

```
void disp_level15(){
	int mesptr1 = &message1;
	int mesptr2 = &level15;
	row1_loc_low = mesptr1;
	row1_loc_hi = (mesptr1&0xff00)/256;
	row2_loc_low = mesptr2;
	row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_level16(){
	int mesptr1 = &message1;
	int mesptr2 = &level16;
	row1_loc_low = mesptr1;
	row1_loc_hi = (mesptr1&0xff00)/256;
	row2_loc_low = mesptr2;
	row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_gameover(){
	int mesptr1 = &message1;
	int mesptr2 = &gameover;
	row1_loc_low = mesptr1;
	row1_loc_hi = (mesptr1&0xff00)/256;
	row2_loc_low = mesptr2;
	row2_loc_hi = (mesptr2&0xff00)/256;
}

void disp_win(){
	int mesptr1 = &message1;
	int mesptr2 = &win;
	row1_loc_low = mesptr1;
	row1_loc_hi = (mesptr1&0xff00)/256;
	row2_loc_low = mesptr2;
	row2_loc_hi = (mesptr2&0xff00)/256;
}

void main(void){
	int level = 1;		//start at level one
	TRISA = 0;		// A output
	PORTA = 0x00;
	INTCON = 0x00;
	T0CON = 0x87;
	T1CON = 0xA1;
	TRISB = 0;		// B output
	TRISC = 0b00000011;	// RC0 and RC1 inputs
	TMR0H = 0x00;
	TMR0L = 0x00;
	pattern_init();		// put 16 random ints into pattern
	disp_level1();		// display level one
	display_on();
	while(1){
		pattern_out(level); // send pattern of LEDs to FPGA
		while(PORTCbits.RC0 == 0 & PORTCbits.RC1 == 0){
			// wait until the user wins or loses
		}
		level = level_inc(PORTCbits.RC0, PORTCbits.RC1,
				level); // check level of user
		if (level == 0) disp_gameover(); // if lose, display game over
		else if (level == 1) disp_level1(); // else, display the level
		else if (level == 2) disp_level2();
		else if (level == 3) disp_level3();
		else if (level == 4) disp_level4();
		else if (level == 5) disp_level5();
		else if (level == 6) disp_level6();
		else if (level == 7) disp_level7();
		else if (level == 8) disp_level8();
		else if (level == 9) disp_level9();
		else if (level == 10) disp_level10();
		else if (level == 11) disp_level11();
		else if (level == 12) disp_level12();
		else if (level == 13) disp_level13();
```

```
                else if (level == 14) disp_level14();
                else if (level == 15) disp_level15();
                else if (level == 16) disp_level16(); // if past level 16, display win
                else   disp_win();
                display_on();
        }
}
```

```
; LCD.asm
; Kirsten McAfee and Julia Karl
; modified from Scrolling.asm by
; Kevin Lloyd and Rajdeep Roy
; Use the 18F4520 PIC microprocessor

; Overview:
; This code displays the message contained in row1_loc and
; row2_loc these variables are defined in simon.c
; The following at the pin connections used in this code

;PORTD(7) - DB7 - Pin 14
;PORTD(6) - DB6 - Pin 13
;PORTD(5) - DB5 - Pin 12
;PORTD(4) - DB4 - Pin 11
;PORTD(3) - DB3 - Pin 10
;PORTD(2) - DB2 - Pin 09
;PORTD(1) - DB1 - Pin 08
;PORTD(0) - DB0 - Pin 07
;PORTC(6) - E - Pin 6
;PORTC(4) - RS - Pin 4

LIST p=18F4520

include "p18f4520.inc"

; allocate variables
;LEDS EQU 0x00;
extern row1_loc_low
extern row1_loc_hi
extern row2_loc_low
extern row2_loc_hi

CODE
display_on
L_Init EQU 0x21
L_8bit EQU 0x22
L_Off EQU 0x23
L_Clear EQU 0x24
L_EntryMode EQU 0x25
L_On EQU 0x26
temp EQU 0x27
;begin main program
        clrf TRISD
        clrf TRISC
        #define LCD_E PORTC,0x06 ;should be 6
        #define LCD_RS PORTC,0x04 ;should be 7 ; #define LCD_RW PORTD,0x03
        movlw b'10000111' ; Setup the timer to enumerate every 256 cycles
        movwf T0CON
        movlw b'00110000' ; The standard Initilization code
        movwf L_Init
        movlw b'00111000' ; The code indicate 8-bit operating mode
        movwf L_8bit
        movlw b'00001000' ; Turn off the LCD
        movwf L_Off
        movlw b'00000001' ; Clear the LCD
        movwf L_Clear
        movlw b'00000110' ; Set the cursor increments when written to the DRAM
        movwf L_EntryMode
        movlw b'00001111' ; Turn the LCD back on
        movwf L_On

        ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        call LCD_Init
        call moveRow1
        clrf TBLPTRU ; Set the pointer to address 1000, address for the 1st Message
        movff row1_loc_hi,TBLPTRH ; Sets high bits of TBLPTR
        movff row1_loc_low,TBLPTRL ; Sets low bits of TBLPTR
        call DispMes
        call moveRow2
```

```
        clrf TBLPTRU
        movff row2_loc_hi, TBLPTRH
        movff row2_loc_low, TBLPTRL
        call DispMes
        bra done

LCD_Init
        call Dlay20 ; The below lines are needed if you want to re-initialize the LCD
                ; while on.
;;
        movff L_Init, PORTD
        ; call Pulse_E
        ; call Dlay5
        ;;
        movff L_Init, PORTD
        ; call Pulse_E
        ; call Dlay5
        ;;
        movff L_Init, PORTD
        ; call Pulse_E
        ; call Dlay5
        movff L_8bit, PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        movff L_Off, PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        movff L_Clear, PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        movff L_EntryMode, PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        movff L_On, PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        return

moveRow1
        movlw 0x80 ; Move to the first line, first spot
        movwf PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay5
        return

moveRow2
        movlw 0xC0 ; Move to the second line, first spot
        movwf PORTD
        bcf LCD_RS ; Set RS low, indicating an Instruction
        call Pulse_E ; Pulse E to tell the LCD that a new Instruction is there
        call Dlay20
        return

DispMes
        mes2
                TBLRD*+ ; Pointer initialization
                movf TABLAT, 0 ; move the value from the TBLPTR to the w-reg
                xorlw 0x00 ; if the end of message (return is 0) then
                bz mes2d ; branch to done
                call DispChar
                bra mes2
        mes2d
                return

DispChar ; Send a character to the LCD (assume char in w-reg)
```

```
        movwf PORTD ; Send the char code to PORTD and thus to LCD pins
        bsf LCD_RS ; Set the RS port high to indicate incomming Data
        call Pulse_E ; Pulse Enable so that knows to work with the data being sent
        call Dlay5 ; Wait 5 ns so that the LCD doesn't get overloaded
        return

Pulse_E ; This pulses the Enable bit so that the LCD pays
        ; attention
        bsf LCD_E
        nop
        nop
        bcf LCD_E
        return

Dlay20 ; Dealy 20 ms by delaying 5ms four times
        call Dlay5
        call Dlay5
        call Dlay5
        call Dlay5
        return

Dlay5 ; Dealy 5 ms by dealying 1ms five times
        call Dlay1
        call Dlay1
        call Dlay1
        call Dlay1
        call Dlay1
        return

Dlay1 ; Delay 1ms (little more than)
                movlw b'1001111' ; The Number x 256 to count up
                        ; to in order to have a 1ms delay
                        ; found by doing
                        ; (delay_time/clockcycle_time)/ 256
                        ; prescalar
                        ; movlw b'0000100' ; 10011111 is for 20 MHz,
                        ; 00000100 is for 1 MHz
                clrf TMR0L ; Reset the Timer
        Dlay1b
                cpfsgt TMR0L ; If the timer counter is less than the w-reg, keep going
                bra Dlay1b
                return ; Goes here when timer is greater than w-reg, done
done
GLOBAL display_on
end
```

Appendix B: FPGA Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineers: Julia Karl and Kirsten McAfee
// Create Date:    17:16:53 11/14/2009
// Module Name:    FPGA_top
//////////////////////////////////////////////////////////////////////////////////
module FPGA_top(
    input clk,
    input reset,
         input [2:0] random,
    input [3:0] button_in,
    output [3:0] leds_out,
    output flag_win,
    output flag_lose
         );

        wire [15:0] first;
        wire [15:0] second;
        wire [3:0] counter;
        wire slow_clock;
        wire really_slow_clock_out;
        wire [2:0] button_pressed;
        wire tenclk;
        wire reset_in;

        really_slow_clock really_slow(clk, reset,really_slow_clock_out);
        leds led(really_slow_clock_out, reset, reset_in,random, leds_out, first, second,
               counter);
        slow_clk slow(clk, reset, slow_clk);
        buttons button(slow_clock, button_in, button_pressed);
        check chec(slow_clock, reset, first, second, counter,button_pressed, flag_win,
               flag_lose, reset_in);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Create Date:    13:34:53 11/22/2009
// Design Name:        Julia Karl and Kirsten McAfee
// Module Name:    really_slow_clock
//////////////////////////////////////////////////////////////////////////////////

// This module is the input to the led module. It samples slow enough for the
// sometimes bouncy signal from the PIC.

module really_slow_clock(
    input clk,
    input reset,
    output really_slow_clock
    );

reg [17:0] counter;

assign really_slow_clock = counter[17]; // assign the most significant bit of the
                                        // counter to become the output to
                                        // really_slow_clock.


always @ (posedge clk) begin
        if (reset) begin
                counter <= 0; // always want the counter to start
                              // at zero. so you want to include a reset in this signal.
                end

        else begin
                counter <= counter + 1; // add one to counter every time there is a new
                                        // clock cycle.
                end
        end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineers: Julia Karl and Kirsten McAfee
// Module Name:    clock_delay
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////

// Module that provides a slow clock to the buttons module. It samples slow
// so there is no bounce in the button presses.

Module slow_clk(
    input clk,
         input reset,
         output slow_clock
    );

reg [20:0] counter;

assign slow_clock = counter[17]; // assign the most significant bit of the counter
                                 // to become the output slow_clock.

always @ (posedge clk) begin
        if (reset) begin
                counter <= 0; // always want the counter to start at zero. so you want to
                              // include a reset in this signal.
                end

        else begin
                counter <= counter + 1; // add one to counter every time there is a new
                                        // clock cycle.
                end
        end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineers: Julia Karl and Kirsten McAfee
// Module Name:    clock_delay
// Additional Comments:
//////////////////////////////////////////////////////////////////////////////////

// This module controls the LEDs. It takes input from the PIC and then stores the
// values so they can be checked later in the game. It also lights up the
// appropriate LEDs according to the numbers sent from the PIC. It also outputs a
// counter which is how many numbers this module has received from the PIC.

module leds(
    input clk,
    input reset,
        input reset_in,
    input [2:0] random,
    output reg [3:0] leds,
        output reg [15:0] first,
        output reg [15:0] second,
        output reg [3:0] counter
    );

        // parameters for the states.
        parameter S0 = 4'b0000;
        parameter S1 = 4'b0001;
        parameter S2 = 4'b0010;
        parameter S3 = 4'b0011;
        parameter S4 = 4'b0100;
        parameter S5 = 4'b0101;

        reg [2:0] random_2;
        reg [3:0] nextstate;
        reg [3:0] state;

        // register for input from the PIC
        always @(posedge clk)
                random_2 <= random;

        // Reseting logic. If the game is lost or won, this module has to reset. So,
        // there is an internal reset_in signal sent from the check module.
        always @(posedge clk or posedge reset) begin
                if (reset|reset_in)           state <= S0;
                else                          state <= nextstate;
        end

        // state machine
        always @(*) begin
                case (state)
                        S0: if (reset)                nextstate = S0;
                            else                      nextstate = S1;
                        S1: if (random_2[2] == 1)     nextstate = S2;  // If the leds
                                // should be on the random 3rd bit of random should be 1,
                                // so move onto the next state.
                            else                      nextstate = S1;
                                // need to wait a few states
                        S2:                           nextstate = S3; // before actually
                                // lighting
                        S3:                           nextstate = S4; // up the LEDs or
                                // sampling the signal from the PIC because of bouncing.
                        S4:                           nextstate = S5;
                        S5: if (random_2 != random)   nextstate = S1; // once the random
                                // bit changes move back to the first state, waiting for
                                // the LEDs to be turned back on.
                            else                      nextstate = S5;
                        default:                      nextstate = S0;
                endcase
        end
```

```
	always @(posedge clk) begin
		case (state)
			S0: begin
				counter <= 0;
				first <= 0;
				leds <= 0;
				second <= 0;
				end

			S1: begin
				counter <= counter;
				first <= first;
				second <= second;
				leds <= 0;
				end

			S2: begin
				counter <= counter;
				first <= first;
				second <= second;
				leds <= 0;
				end

			S3: begin
				counter <= counter;
				first <= first;
				second <= second;
				leds <= 0;
				end

			S4: begin
				counter <= counter + 1; // add one to the counter
				first[counter] <= random_2[0]; // store the first random
					// bit.
				second[counter]  <= random_2[1];// store the second random
					// bit.
				leds <= 0;
				end

			S5: begin
				// logic for lighting up the LEDs.
				counter <= counter;
				first <= first;
				second <= second;
				if (random == 4)
					leds <= 4'b0001;
				else if (random == 5)
					leds <= 4'b0010;
				else if (random == 6)
					leds <= 4'b0100;
				else if (random == 7)
					leds <= 4'b1000;
				else
					leds <= 0;
				end

			default: begin
				counter <= 0;
				first <= 0;
				leds <= 0;
				second <= 0;
				end
		endcase
	end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Julia Karl and Kirsten McAfee
// Create Date:    18:01:10 11/14/2009
// Module Name:    buttons
//////////////////////////////////////////////////////////////////////////////////

// This module takes in input from the buttons. and changes them into integers
// that will match the random input from the PIC. The output of this module is
// used in the check module.
module buttons(
    input clk,
    input [3:0] button_in,
         output reg [2:0] button_pressed
    );

        reg [3:0] button_in_2;


        always @(posedge clk) begin
                button_in_2 <= button_in;
                end

        always @(*)
                if (button_in_2 == 4'b0001)
                        button_pressed <= 0;
                else if (button_in_2 == 4'b0010)
                        button_pressed <= 1;
                else if (button_in_2 == 4'b0100)
                        button_pressed <= 2;
                else if (button_in_2 == 4'b1000)
                        button_pressed <= 3;
                else
                        button_pressed <= 4;
endmodule
```

```
'timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineers: Julia Karl and Kirsten McAfee
// Create Date:    17:22:56 11/14/2009
// Module Name:    check
//////////////////////////////////////////////////////////////////////////////////

// Takes input from other modules. It knows what the input from the pic has been
// and which buttons have been pressed. It checks every button against the actual
// values that the pic has sent. It sends out a win flag if the level is passed,
// a lose flag if the level is lost. It also sends out a reset_in when the level
// is either lost or won. It resets the LED module.

module check(
    input slow_clock,
    input reset,
         input [15:0] first,
         input [15:0] second,
         input [3:0] counter,
    input [2:0] button_pressed,
    output flag_win,
    output flag_lose,
         output reset_in
    );

        parameter S0 = 0; // parameters for all of the states.
        parameter S1 = 1;
        parameter S2 = 2;
        parameter S3 = 3;
        parameter S4 = 4;
        parameter S5 = 5;
        parameter S6 = 6;
        parameter S7 = 7;
        parameter S8 = 8;
        parameter S9 = 9;
        parameter S10 = 10;
        parameter S11 = 11;
        parameter S12 = 12;
        parameter S13 = 13;
        parameter S14 = 14;
        parameter S15 = 15;
        parameter S16 = 16;
        parameter S17 = 17;
        parameter S18 = 18;
        parameter S19 = 19;
        parameter S20 = 20;
        parameter S21 = 21;
        parameter S22 = 22;
        parameter S23 = 23;
        parameter S24 = 24;
        parameter S25 = 25;
        parameter S26 = 26;
        parameter S27 = 27;
        parameter S28 = 28;
        parameter S29 = 29;
        parameter S30 = 30;
        parameter S31 = 31;
        parameter S32 = 32;
        parameter S33 = 33;
        parameter S34 = 34;
        parameter S35 = 35;
        parameter S36 = 36;
        parameter S37 = 37;
        parameter S38 = 38;
        parameter S39 = 39;
        parameter S40 = 40;
        parameter S41 = 41;
        parameter S42 = 42;
        parameter S43 = 43;
        parameter S44 = 44;
```

```
reg [5:0] nextstate;
reg [5:0] state;

        // state register
always @(posedge slow_clock or posedge reset) begin
        if (reset)      state <= S44;
        else            state <= nextstate;
end

        // state machine
always @(posedge slow_clock) begin
        case (state)
                S0: if (reset)                          nextstate = S0;
                    else                                nextstate = S1;
                S1: if (button_pressed == 4)            nextstate = S1;
                    else if ((button_pressed[0] != first[0]) | (button_pressed[1]
                        != second[0]))                  nextstate = S33;
                                // when button pressed = 4, no button is being
                                // pressed. So stay in the same state check to see
                                // if the button that was pressed was correct
                    else                                nextstate = S2;
                                // if the button pressed was correct, move to the
                                // next state
                S2: if (button_pressed == 4)            nextstate = S3;
                                // wait until the button changes back to 4 and then
                                // change states.
                    else if (counter == 1)              nextstate = S34;
                                // if the level matches, then consider this level
                                // won.
                    else                                nextstate = S2;
                S3: if (button_pressed == 4)            nextstate = S1;
                    else if ((button_pressed[0] != first[1]) | (button_pressed[1]
                        != second[1]))                  nextstate = S33;
                    else                                nextstate = S4;
                S4: if (button_pressed == 4)            nextstate = S5;
                    else if (counter == 2)              nextstate = S34;
                    else                                nextstate = S4;
                S5: if (button_pressed == 4)            nextstate = S5;
                    else if ((button_pressed[0] != first[2]) | (button_pressed[1]
                        != second[2]))                  nextstate = S33;
                    else                                nextstate = S6;
                S6: if (button_pressed == 4)            nextstate = S7;
                    else if (counter == 3)              nextstate = S34;
                    else                                nextstate = S6;
                S7: if (button_pressed == 4)            nextstate = S7;
                    else if ((button_pressed[0] != first[3]) | (button_pressed[1]
                        != second[3]))                  nextstate = 33;
                    else                                nextstate = S8;
                S8: if (button_pressed == 4)            nextstate = S9;
                    else if (counter == 4)              nextstate = S34;
                    else                                nextstate = S8;
                S9: if (button_pressed == 4)            nextstate = S9;
                    else if ((button_pressed[0] != first[4]) | (button_pressed[1]
                        != second[4]))                  nextstate = S33;
                    else                                nextstate = S10;
                S10: if (button_pressed == 4)           nextstate = S11;
                     else if (counter == 5)             nextstate = S34;
                     else                               nextstate = S10;
                S11: if (button_pressed == 4)           nextstate = S11;
                     else if ((button_pressed[0] != first[5]) | (button_pressed[1]
                        != second[5]))                  nextstate = S33;
                     else                               nextstate = S12;
                S12: if (button_pressed == 4)           nextstate = S13;
                     else if (counter == 6)             nextstate = S34;
                     else                               nextstate = S12;
                S13: if (button_pressed == 4)           nextstate = S13;
                     else if ((button_pressed[0] != first[6]) | (button_pressed[1]
                        != second[6]))                  nextstate = S33;
                     else                               nextstate = S14;
                S14: if (button_pressed == 4)           nextstate = S15;
                     else if (counter == 7)             nextstate = S34;
```

```
     else                                nextstate = S14;
S15: if (button_pressed == 4)            nextstate = S15;
     else if ((button_pressed[0] != first[7]) | (button_pressed[1]
       != second[7]))                    nextstate = S33;
     else                                nextstate = S16;
S16: if (button_pressed == 4)            nextstate = S17;
     else if (counter == 8)              nextstate = S34;
     else                                nextstate = S16;
S17: if (button_pressed == 4)            nextstate = S17;
     else if ((button_pressed[0] != first[8]) | (button_pressed[1]
       != second[8]))                    nextstate = S33;
     else                                nextstate = S18;
S18: if (button_pressed == 4)            nextstate = S19;
     else if (counter == 9)              nextstate = S34;
     else                                nextstate = S18;
S19: if (button_pressed == 4)            nextstate = S19;
     else if ((button_pressed[0] != first[9]) | (button_pressed[1]
       != second[9]))                    nextstate = S33;
     else                                nextstate = S20;
S20: if (button_pressed == 4)            nextstate = S21;
     else if (counter == 10)             nextstate = S34;
     else                                nextstate = S20;
S21: if (button_pressed == 4)            nextstate = S21;
     else if ((button_pressed[0] != first[10]) | (button_pressed[1]
       != second[10]))                   nextstate = S33;
     else                                nextstate = S22;
S22: if (button_pressed == 4)            nextstate = S23;
     else if (counter == 11)             nextstate = S34;
     else                                nextstate = S22;
S23: if (button_pressed == 4)            nextstate = S23;
     else if ((button_pressed[0] != first[11]) | (button_pressed[1]
       != second[11]))                   nextstate = S33;
     else                                nextstate = S24;
S24: if (button_pressed == 4)            nextstate = S25;
     else if (counter == 12)             nextstate = S34;
     else                                nextstate = S24;
S25: if (button_pressed == 4)            nextstate = S25;
     else if ((button_pressed[0] != first[12]) | (button_pressed[1]
       != second[12]))                   nextstate = S33;
     else                                nextstate = S26;
S26: if (button_pressed == 4)            nextstate = S27;
     else if (counter == 13)             nextstate = S34;
     else                                nextstate = S26;
S27: if (button_pressed == 4)            nextstate = S27;
     else if ((button_pressed[0] != first[13]) | (button_pressed[1]
       != second[13]))                   nextstate = S33;
     else                                nextstate = S28;
S28: if (button_pressed == 4)            nextstate = S29;
     else if (counter == 14)             nextstate = S34;
     else                                nextstate = S28;
S29: if (button_pressed == 4)            nextstate = S29;
     else if ((button_pressed[0] != first[14]) | (button_pressed[1]
       != second[14]))                   nextstate = S33;
     else                                nextstate = S30;
S30: if (button_pressed == 4)            nextstate = S31;
     else if (counter == 15)             nextstate = S34;
     else                                nextstate = S30;
S31: if (button_pressed == 4)            nextstate = S31;
     else if ((button_pressed[0] != first[15]) | (button_pressed[1]
       != second[15]))                   nextstate = S33;
     else                                nextstate = S32;
S32: if (button_pressed == 4)            nextstate = S33;
     else if (counter == 16)             nextstate = S34;
     else                                nextstate = S32;
S33:                                     nextstate = S35; //lose
S34: if (button_pressed == 4)            nextstate = S40; //win
     else                                nextstate = S34;
S35:                                     nextstate = S36;
               // hold lose flag for a few clock cycles
S36:                                     nextstate = S37;
S37:                                     nextstate = S38;
```

```
                        S38:                                    nextstate = S39;
                        S39:                                    nextstate = S1;
                        S40:                                    nextstate = S41;
                                                // hold win flag for a few clock cycles
                        S41:                                    nextstate = S42;
                        S42:                                    nextstate = S43;
                        S43:                                    nextstate = S44;
                        S44:                                    nextstate = S1;
                        default:                                nextstate = S1;
                endcase
        end

                // output logic
        assign flag_win = (state == S39); // output flag_win when in state 33.
        assign flag_lose = (state == S32); // output flag_lose when in stae 32.
        assign reset_in = ((state == 42) | (state == 37));

endmodule
```