ELEX 7660
Project Report

Simon

Mike Bagheri

Jeremy Barbour

Alexander Bowers

ELEX 7660: Digital System Design

Final Project Report

Simon

Date Prepared: April 16, 2017

Table of Contents

0 Introduction

We as a group came up with an idea of making a game that is called Simon. Simon is a mixture of hardware and software. For each level, the device creates a pattern of tones and lights and waits until a user repeats the exact same pattern by pressing the pushbuttons. Upon user's success, the game advances to the next level, and the number of tones and lights keep increasing. Upon user's failure, the game goes back to level 1.

1 Objectives

At the beginning of the project, we set two sets of goals; primary and secondary. Primary goals are the most essential; these are necessary for the game to function properly. Secondary goals are additional add-ons that require more time. Due to the limited time that we had, we mainly focused on primary goals as being our target.

1.1 Primary Goals

- Display the colour pattern to be pressed
- Display different number of colours per pattern, increasing difficulty and length
- Be able to differentiate between the push buttons that the user presses
- Recognize if the pattern entered from the user matches the generated colour pattern
- Upon entering an incorrect colour pattern the game will restart
- Display the number of consecutive rounds correctly entered on the 7-segment display

We were able to achieve all of the primary goals by the project deadline.

1.2 Secondary Goals

- Being able to limit the amount of time the user has to press the correct combination
- Upgrade the 7-segment display to an LCD display
- Making a basic game menu that would welcome a player includes; start, instruction, difficulty
- Being able to generate a unique tone for each colour as it shows the colour pattern and as the player presses the push buttons

We complete a sound addition to the game, and an appropriate sound is played for each pushbutton when the colours are being displayed. We also added a feature where it would light up the pushbuttons and play the associated sound when the user was entering the pattern but this made gameplay confusing, and so was removed. Unfortunately, due to the limited time we were not able to achieve many secondary goals.

2 Software and Hardware

Software and hardware used for this project are as follows:

Hardware		Software
Item	Quantity	Quartus Prime 2016
Lamps/Pushbutton	4	
4 X 7 Segment LEDS	1	
Speaker	1	
Transistors	4	
10K Resistors	4	
Soldering Board	1	
FPGA	1	

3 Game Interfacing

3.1 Human Interface

3.1.1 Display

For displaying the score on the 4X7 segment display we used 3 distinct modules.

1- Decode2
2- Decodebin
3- Decode7

This module activates the appropriate VCC

1- Decode2:

```
module decode2 (input logic [1:0] digit,
            output logic [3:0] ct);

      always_comb begin
            case (digit)
                                        // Enable appropriate Vcc
            0: ct = 4'b_0001;
            1: ct = 4'b_0010;
            2: ct = 4'b_0100;
            3: ct = 4'b_1000;

            endcase
      end
```

```
endmodule
```

This module converts binary into seperated decimals

2- Decodebin

```
module Decodebin (input logic [1:0] digit,            // Clock going from 0 to 3
             output logic[3:0] idnum,
                   input [31:0] scount);        // Single digit from my ID number
            reg [3:0] hundreds = 4'd0;
            reg [3:0] tens = 4'd0;
            reg [3:0] ones = 4'd0;
            reg [19:0] level;
            integer i;

            always@(digit) begin
                  begin

      level[19:8] = 0;
      level[7:0] = scount[7:0];
      // this is the double dabble algorithm it is used to convert binary to
seperated decimal values
      for (i=0; i<8; i=i+1) begin
         if (level[11:8] >= 5)
            level[11:8] = level[11:8] + 3;

         if (level[15:12] >= 5)
            level[15:12] = level[15:12] + 3;

         if (level[19:16] >= 5)
            level[19:16] = level[19:16] + 3;

         level = level << 1;
      end

      hundreds = level[19:16];
      tens     = level[15:12];
      ones     = level[11:8];
   end
            // updates score to correct stage number

            unique case (digit)

            0: idnum = ones;
            1: idnum = tens;
            2: idnum = 4'd0;
            3: idnum = 4'd0;

            endcase
      end

endmodule
```

This module turns on the correct leds depending on the score number

3- Decode7

```
module decode7 ( input logic [3:0] num,          // Input num that will display on
led array
                              output logic [7:0] leds);           // leds that will
turn on
                  always_comb begin
                              unique case (num)
                              // Assignment of leds for each num
                              0: leds <= ~(8'b_0011_1111);
                              1: leds <= ~(8'b_0000_0110);
                              2: leds <= ~(8'b_0101_1011);
                              3: leds <= ~(8'b_0100_1111);
                              4: leds <= ~(8'b_0110_0110);
                              5: leds <= ~(8'b_0110_1101);
                              6: leds <= ~(8'b_0111_1101);
                              7: leds <= ~(8'b_0000_0111);
                              8: leds <= ~(8'b_0111_1111);
                              9: leds <= ~(8'b_0110_1111);
                              endcase

                    end

endmodule
```

3.1.2 Audio

The following module was used as a tone generator for the project. The tone generator is given a frequency to play when each pushbutton is being lit up and is given a frequency of zero (off) when the pushbuttons are not supposed to be lit up.

```
module tonegen
  #( logic [31:0] fclk = 50 * 1000 * 1000)           // clock frequency, Hz
   ( input logic [31:0] writedata, // Avalon MM bus, data
     input logic write,             // " write strobe
     output logic spkr,             // on/off output for audio
     input logic reset, clk );

      enum logic [1:0] {reset_s, write_s, decr, restart} state;
      reg [31:0] frequency;
      reg signed [31:0] count;

      always_comb begin
            if (reset)
                  state = reset_s;
            else if (write)
                  state = write_s;
            else if (count < 1)
                  state = restart;
            else
                  state = decr;
```

```
	end

	always_ff @(posedge clk) begin
		if (spkr)							//define register
			spkr <= 1;
		else
			spkr <= 0;
		unique case (state)
			reset_s: begin
				frequency <= 0;
				count <= fclk;
				end
			write_s:
				frequency <= writedata;			// tone frequency
			decr:
				count <= count - (2 * frequency);	// delay
			restart: begin
				if (!spkr)
					spkr <= 1;
				else
					spkr <= 0;
				count <= fclk * 10;
			end
		endcase
	end

	endmodule
```

3.1.3 Pushbuttons

The following module found on FPGA4fun [1] was used to debounce the pushbuttons, as without a pushbutton debouncer the FPGA would think that the pushbuttons were pressed multiple times each press due to contact bounce. We opted for a software solution for the debouncers and not hardware solution due to ease of trouble shooting a software solution and for an easier manufacturing process.

```
module PushButton_Debouncer(
    input clk,
    input PB,  // "PB" is the glitchy, asynchronous to clk, active low push-button
signal

    // from which we make three outputs, all synchronous to the clock
    output reg PB_state,  // 1 as long as the push-button is active (down)
    output PB_down,  // 1 for one clock cycle when the push-button goes down (i.e.
just pushed)
    output PB_up   // 1 for one clock cycle when the push-button goes up (i.e. just
released)
);

// First use two flip-flops to synchronize the PB signal the "clk" clock domain
reg PB_sync_0;  always @(posedge clk) PB_sync_0 <= ~PB;  // invert PB to make
PB_sync_0 active high
reg PB_sync_1;  always @(posedge clk) PB_sync_1 <= PB_sync_0;

// Next declare a 16-bits counter
```

```
reg [15:0] PB_cnt;

// When the push-button is pushed or released, we increment the counter
// The counter has to be maxed out before we decide that the push-button state has
changed

wire PB_idle = (PB_state==PB_sync_1);
wire PB_cnt_max = &PB_cnt;     // true when all bits of PB_cnt are 1's

always @(posedge clk)
if(PB_idle)
    PB_cnt <= 0;  // nothing's going on
else
begin
    PB_cnt <= PB_cnt + 16'd1;  // something's going on, increment the counter
    if(PB_cnt_max) PB_state <= ~PB_state;  // if the counter is maxed out, PB
changed!
end

assign PB_down = ~PB_idle & PB_cnt_max & ~PB_state;
assign PB_up   = ~PB_idle & PB_cnt_max &  PB_state;

endmodule
```

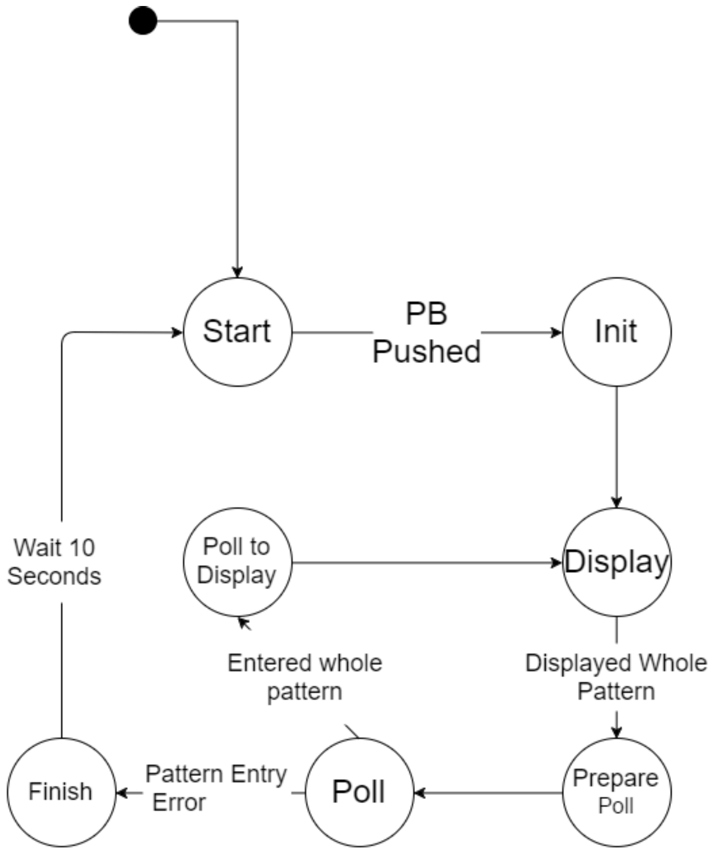
Once the pushbuttons have been successfully debounced, they can be polled and then it can be determined if the correct pattern was entered. The controller simply determines if the correct button is released each time one of the pushbuttons is released. Released was chosen and not pushed for this part of the project because if it was when the button was pressed the next pattern would start right as the last button was pressed and this was confusing for the user.

```
            if ((lcount > 0) && PBUP) begin
             lcount <= lcount -1;
             case(out % (NUM_LAMPS))
             0: loser <= !PBY_up;
             1: loser <= !PBR_up;
             2: loser <= !PBB_up;
             3: loser <= !PBG_up;
             endcase // out % NUMLAMPS
             lfsrclk <= 1;

            end // end if(lcount > 0) %% PBUP
```

4 Digital Design

It was decided that the game would be realized using solely hardware, and so a state machine must be created for the game. The following algorithm was implemented for the game. To randomize the game each time a linear feedback shift register (LFSR) [2] was used. This is one of the reasons for a start stage, the LFSR was "seeded" during the time is takes the user to press the start button to begin the game.



Simon Game State Machine

Once the game enters the display stage of the game, the LFSR is seeded with the seed and the first colour is displayed. Once this colour has been displayed for enough time, the LFSR is clocked and then the new colour is displayed. Once the entire pattern is displayed, the game gets ready to poll the pushbuttons to see if the user presses the correct pattern.

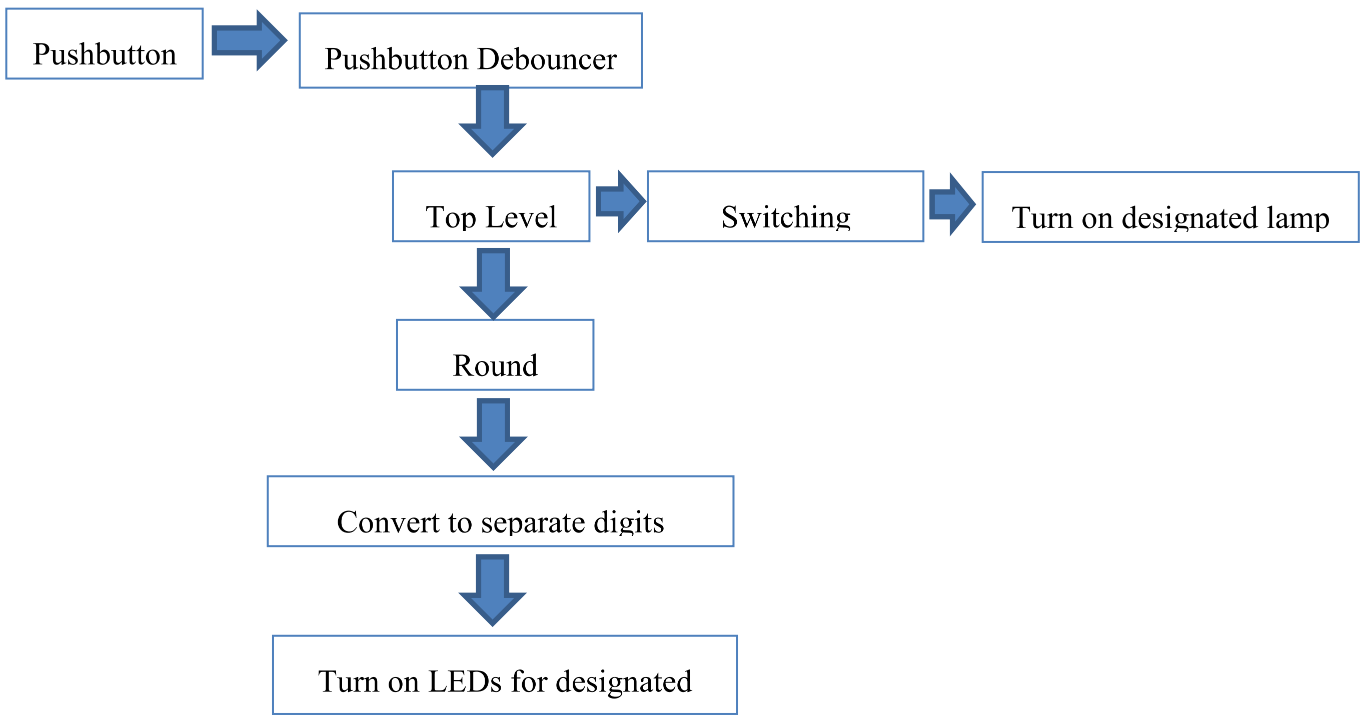
First step is the prepare poll state, where the tone generator is turned off, and the LFSR is reseeded, an important characteristic of LFSR is used in this game, if the seed is the same to the LFSR then the resulting pattern will be the same, so there is no need to store the displayed bits in an array, they are stored within the algorithm of the LFSR.

The next step is to poll the push buttons and determine if the entered pattern matches the displayed pattern. The pushbuttons Are tested for when they are released, and this signal is high for one clock cycle when the pushbutton is just released. When the pushbutton is released the algorithm determines if the press matches the correct one from the LFSR and if it does it clocks the LFSR and continues. If the button pressed does not match the game restarts.

Once the player has Entered the whole pattern the game does back to the displaying stage and the game displays the pattern again, but this time with one more colour appended on the end.

5 Physical Design

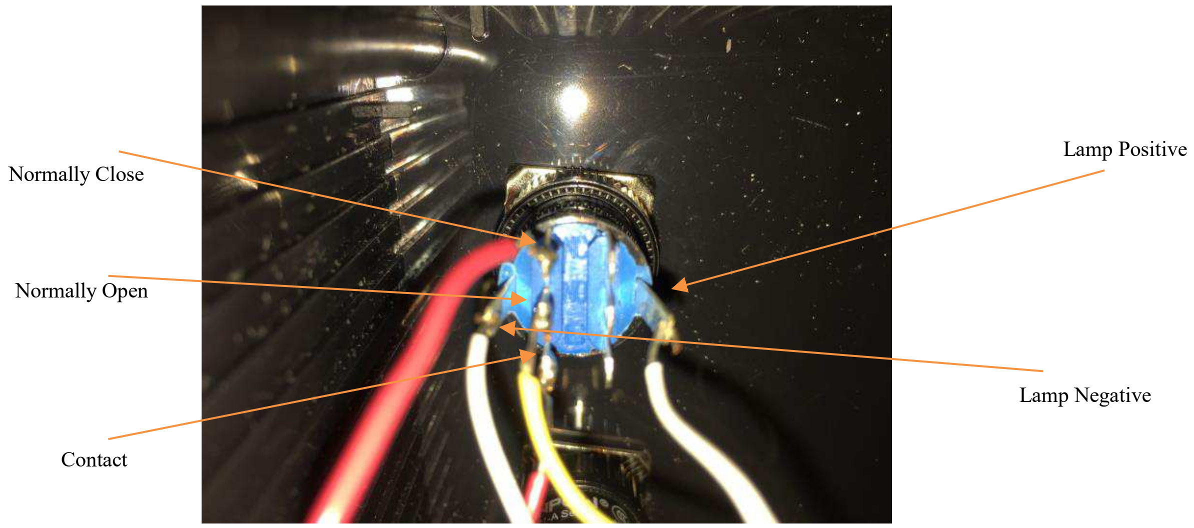
We have decided to use an enclosure for our project in order to make it neat. We have used a soldering board to solder components such as resistors, transistors. In addition, there would be common Ground and VCC.



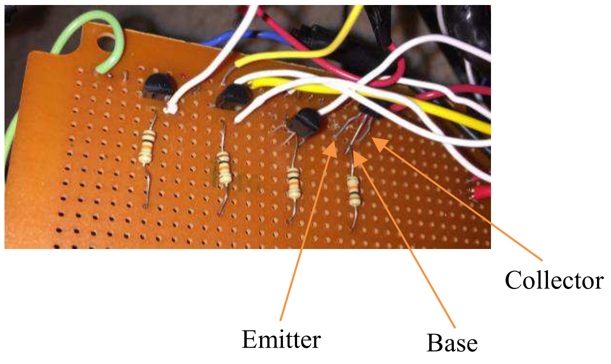
5.1 Push Buttons

The holes for pushbuttons were drilled first and filled to achieve the right size. The pins of each pushbuttons were carefully solder to VCC, Ground, and bread board. The use of transistor for this project is to provide enough voltage to turn the lamps inside of the pushbuttons.

5 pins of each pushbutton were used for this project.

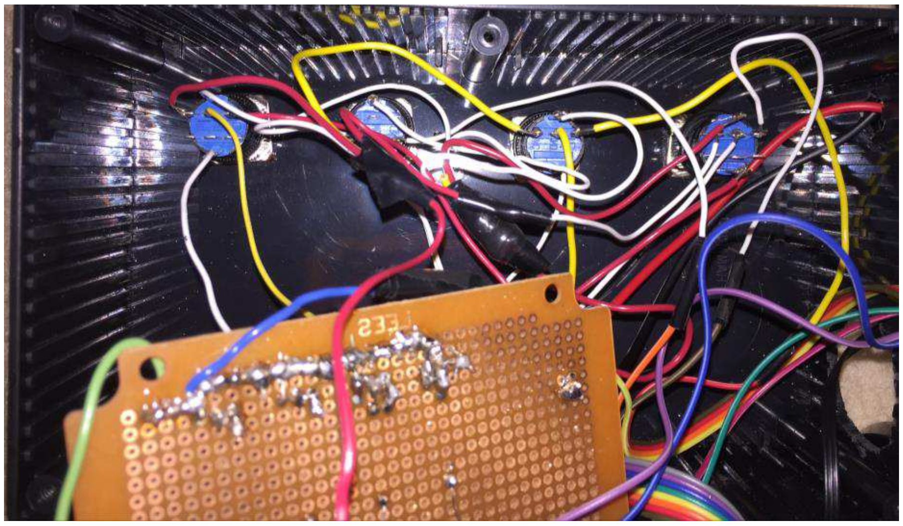


Pushbuttons	Normally Close	Normally Open	Contact	Lamp Negative	Lamp Positive
Yellow	Ground of FPGA	VCC of FPGA	FPGA input	Collector of Transistor	VCC
Red	Ground of FPGA	VCC of FPGA	FPGA input	Collector of Transistor	VCC
Green	Ground of FPGA	VCC of FPGA	FPGA input	Collector of Transistor	VCC
Blue	Ground of FPGA	VCC of FPGA	FPGA input	Collector of Transistor	VCC



Shows the pins related to data transfer from FPGA.

Pushbuttons	Input to FPGA	Output from FPGA for turning on Lamps
Yellow	PIN_j14	PIN_L14
Red	PIN_K15	PIN_M10
Green	PIN_L13	PIN_J16
Blue	PIN_N14	PIN_J13

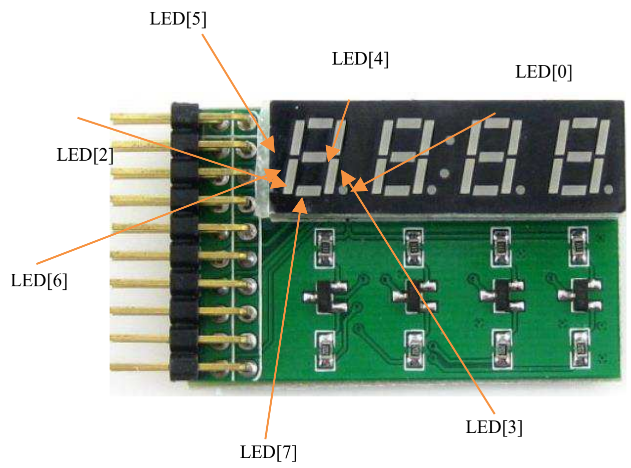


5.2 Display

The same process that was used for pushbuttons were also used here, we used drill and file to achieve the appropriate size for the display.

Used For	Output Pins from FPGA
LED[0]	PIN_A5
LED[1]	PIN_B6
LED[2]	PIN_B7
LED[3]	PIN_A7
LED[4]	PIN_C8

LED[5]	PIN_E7
LED[6]	PIN_E8
LED[7]	PIN_F9
CT[0]	PIN_A12
CT[1]	PIN_C11
CT[2]	PIN_E11
CT[3]	PIN_C9



5.3 Audio

We have decided to add a speaker in order to hear a unique tone specifically for different colours. The speaker plays a distinct sound when the colours are displayed to aid memory.

6 References

[1] "FPGA fun," [Online]. Available: http://www.fpga4fun.com/Debouncer2.html. [Accessed 3 April 2017].

[2] D. K. Tala, "asic world," 9 February 2014. [Online]. Available: http://www.asic-world.com/examples/verilog/lfsr.html. [Accessed 28 March 2017].

7 Appendix: Source Code

The following code is all of the modules in the game, including the top level module.

```
      parameter NUM_LAMPS = 4;
      parameter SEED_BITS = 16;
      parameter NOTE_C = 2616;
      parameter NOTE_D = 2937;
      parameter NOTE_E = 3296;
      parameter NOTE_F = 3492;
      parameter NOTE_G = 3920;

module lab1 ( input logic CLOCK_50,     // 50 MHz clock
              output logic [3:0] LAMPS, output logic [7:0] leds,
                               output logic [3:0] ct,
                           input logic PBY, input logic PBR, input logic PBB, input
logic PBG,
                            output logic spkr) ;

      reg         [31:0]count = 0;
      reg         [(SEED_BITS - 1):0]out = 0;
      reg         rst = 0, loser = 0, lfsrclk, reset = 0, write = 1, PBPRESS,
PBDOWN, PBUP, PBY_state, PBY_down, PBY_up, PBR_state, PBR_down, PBR_up, PBB_state,
PBB_down, PBB_up ,PBG_state, PBG_down, PBG_up;
      reg               [31:0] scount = 0, speakerfreq = 0, finishcount = 0;
      reg signed  [31:0] lcount = 0;
      reg         [(SEED_BITS - 1):0] seed = 8'b0101_0101;       // lfsr seed
      enum              {start, init, display, preparepoll, poll, polltodisplay,
finish} state = start, statenext;

      lab1clk lab1clk_0 ( CLOCK_50, clk ) ;
      decode2 decode2_0 (.digit,.ct) ;
      bcitid  bcitid_0  (.digit,.idnum, .scount) ;
      decode7 decode7_0 (.num(idnum),.leds) ;

                        logic [1:0] digit ;
                              logic [3:0] idnum ;
      lfsr lfsr_0(.out, .clk(lfsrclk), .rst, .seed);

      PushButton_Debouncer YellowPB( .clk(CLOCK_50), .PB(PBY),
.PB_state(PBY_state), .PB_down(PBY_down), .PB_up(PBY_up) );
      PushButton_Debouncer RedPB( .clk(CLOCK_50), .PB(PBR),  .PB_state(PBR_state),
.PB_down(PBR_down), .PB_up(PBR_up) );
      PushButton_Debouncer BluePB( .clk(CLOCK_50), .PB(PBB),  .PB_state(PBB_state),
.PB_down(PBB_down), .PB_up(PBB_up) );
      PushButton_Debouncer GreenPB( .clk(CLOCK_50), .PB(PBG),
.PB_state(PBG_state), .PB_down(PBG_down), .PB_up(PBG_up) );

      tonegen tonegen_0( .clk(CLOCK_50), .reset, .spkr, .write,
.writedata(speakerfreq));
            logic       clk ;


                              always_ff @(posedge clk)
                                digit <= digit + 1'b1 ;
   always_ff @(posedge CLOCK_50) begin

        write <= 0;
```

```
    rst <= 0;
    lfsrclk <= 0;
    state <= statenext;

    case(state)

    start: begin

    seed <= seed + 1;
    count <= count + 1;
    if (count > 50*1000*1000 ) begin
    count <= 0;
    if((LAMPS >= (1 << 3))  || (LAMPS == 0))
     LAMPS <= 1;
    else
        LAMPS <= LAMPS << 1;
    end

    end   // end case start:

    init: begin
    write <= 1;
    rst <= 1;
    lfsrclk <= 1;
    count = 0;
    end   // end begin

    display: begin
 count <= count + 1;

    if(count < 10 * 1000 * 1000) begin
        LAMPS <= '0;
        write <= 1;
        case(out % (NUM_LAMPS))
         0: speakerfreq <= NOTE_C;
         1: speakerfreq <= NOTE_D;
         2: speakerfreq <= NOTE_E;
         3: speakerfreq <= NOTE_F;
        endcase // endcase
        end // end if
    else begin
     LAMPS <= (1 << (out % (NUM_LAMPS)));
    end // end else


    if (count > 35 * 1000 * 1000 ) begin
     lfsrclk <= 1;
        count <= 0;
        lcount <= lcount + 1;
    end



    end // end display

  preparepoll: begin
        rst <= 1;              // reset the LFSR
        write <= 1;
        speakerfreq <= 0;
  end
```

```
      poll: begin
            if ((lcount > 0) && PBUP) begin
             lcount <= lcount -1;
             case(out % (NUM_LAMPS))
             0: loser <= !PBY_up;
             1: loser <= !PBR_up;
             2: loser <= !PBB_up;
             3: loser <= !PBG_up;
             endcase // out % NUMLAMPS
             lfsrclk <= 1;
            end // end if(lcount > 0) %% PBUP


       end // end poll

      polltodisplay: begin
       scount <= scount + 1;  // next level!
       rst <= 1;              // reseed lfsr
       count <= 0;            // reset timer
      end // endpolltodisplay

      finish: begin
       scount <= 0;
       loser <= 0;
       LAMPS <= '1;
      end

      endcase
end

always_comb begin
      PBPRESS = PBY_state || PBR_state || PBB_state || PBG_state;
      PBDOWN = PBY_down || PBR_down || PBB_down || PBG_down;
      PBUP = PBY_up || PBR_up || PBB_up || PBG_up;
      if ((state == start) && (PBY_state || PBR_state || PBB_state || PBG_state) )
       statenext = init;
      else if (state == init)
       statenext = display;
      else if ((state == display) && (lcount > (scount)))
       statenext = preparepoll;
      else if (state == preparepoll)
       statenext = poll;
      else if ((state == poll) && (loser || (lcount < 1)))
       statenext = loser ? finish : polltodisplay;
      else if (state == polltodisplay)
       statenext = display;
      else if ((state == finish) && (finishcount < 50 * 1000 * 1000 * 10))
       statenext = start;
      else
       statenext = state;
end

endmodule


module decode2 (input logic [1:0] digit,
            output logic [3:0] ct);

      always_comb begin
            case (digit)
                                          // Enable appropriate Vcc
            0: ct = 4'b_0001;
            1: ct = 4'b_0010;
            2: ct = 4'b_0100;
            3: ct = 4'b_1000;

            endcase
      end

endmodule



module bcitid (input logic [1:0] digit,                 // Clock going from 0 to 3
             output logic[3:0] idnum,
                   input [31:0] scount);          // Single digit from my ID number
            reg [3:0] hundreds = 4'd0;
            reg [3:0] tens = 4'd0;
            reg [3:0] ones = 4'd0;
            reg [19:0] level;
            integer i;

            always@(digit) begin
                  begin

      level[19:8] = 0;
      level[7:0] = scount[7:0];

      for (i=0; i<8; i=i+1) begin
         if (level[11:8] >= 5)
            level[11:8] = level[11:8] + 3;

         if (level[15:12] >= 5)
            level[15:12] = level[15:12] + 3;

         if (level[19:16] >= 5)
            level[19:16] = level[19:16] + 3;

         level = level << 1;
      end

      hundreds = level[19:16];
      tens     = level[15:12];
      ones     = level[11:8];
   end


            unique case (digit)

            0: idnum = ones;                    // Last ID number
            1: idnum = tens;
            2: idnum = 4'd0;
            3: idnum = 4'd0;        // Fist ID number

            endcase
      end
endmodule
```

```
module decode7 ( input logic [3:0] num,          // Input num that will display on
led array
                              output logic [7:0] leds);          // leds that will
turn on
                  always_comb begin
                              unique case (num)
                              // Assignment of leds for each num
                              0: leds <= ~(8'b_0011_1111);
                              1: leds <= ~(8'b_0000_0110);
                              2: leds <= ~(8'b_0101_1011);
                              3: leds <= ~(8'b_0100_1111);
                              4: leds <= ~(8'b_0110_0110);
                              5: leds <= ~(8'b_0110_1101);
                              6: leds <= ~(8'b_0111_1101);
                              7: leds <= ~(8'b_0000_0111);
                              8: leds <= ~(8'b_0111_1111);
                              9: leds <= ~(8'b_0110_1111);
                              endcase

                    end

endmodule





module lfsr (output reg [(SEED_BITS - 1):0]out, input clk, input rst, input
[(SEED_BITS - 1):0] seed);

  wire feedback;




  assign feedback = ~(out[7] ^ out[2]);


always @(posedge clk, posedge rst)
  begin
    if (rst)
      out = seed;
    else
      out = {out[(SEED_BITS - 2):0],feedback};
  end
endmodule



module PushButton_Debouncer(
    input clk,
    input PB,  // "PB" is the glitchy, asynchronous to clk, active low push-button
signal

    // from which we make three outputs, all synchronous to the clock
    output reg PB_state,  // 1 as long as the push-button is active (down)
    output PB_down,  // 1 for one clock cycle when the push-button goes down (i.e.
just pushed)
    output PB_up   // 1 for one clock cycle when the push-button goes up (i.e. just
released)
);
```

```
// First use two flip-flops to synchronize the PB signal the "clk" clock domain
reg PB_sync_0;  always @(posedge clk) PB_sync_0 <= ~PB;  // invert PB to make
PB_sync_0 active high
reg PB_sync_1;  always @(posedge clk) PB_sync_1 <= PB_sync_0;

// Next declare a 16-bits counter
reg [15:0] PB_cnt;

// When the push-button is pushed or released, we increment the counter
// The counter has to be maxed out before we decide that the push-button state has
changed

wire PB_idle = (PB_state==PB_sync_1);
wire PB_cnt_max = &PB_cnt;    // true when all bits of PB_cnt are 1's

always @(posedge clk)
if(PB_idle)
    PB_cnt <= 0;  // nothing's going on
else
begin
    PB_cnt <= PB_cnt + 16'd1;  // something's going on, increment the counter
    if(PB_cnt_max) PB_state <= ~PB_state;  // if the counter is maxed out, PB
changed!
end

assign PB_down = ~PB_idle & PB_cnt_max & ~PB_state;
assign PB_up   = ~PB_idle & PB_cnt_max &  PB_state;
endmodule



module tonegen
  #( logic [31:0] fclk = 50 * 1000 * 1000)           // clock frequency, Hz
   ( input logic [31:0] writedata, // Avalon MM bus, data
     input logic write,            // " write strobe
     output logic spkr,            // on/off output for audio
     input logic reset, clk );

      enum logic [1:0] {reset_s, write_s, decr, restart} state;
      reg [31:0] frequency;
      reg signed [31:0] count;

      always_comb begin
            if (reset)
                  state = reset_s;
            else if (write)
                  state = write_s;
            else if (count < 1)
                  state = restart;
            else
                  state = decr;
      end

      always_ff @(posedge clk) begin
            if (spkr)                                                  //define register
                  spkr <= 1;
            else
                  spkr <= 0;
            unique case (state)
                  reset_s: begin
```

```
                        frequency <= 0;
                        count <= fclk;
                        end
                write_s:
                        frequency <= writedata;             // tone frequency
                decr:
                        count <= count - (2 * frequency);   // delay
                restart: begin
                        if (!spkr)
                                spkr <= 1;
                        else
                                spkr <= 0;
                        count <= fclk * 10;
                end
        endcase
end

endmodule
```

8 Appendix: PINOUTS

The following file is the pinouts assigned for the project.

CLOCK_50	Location	PIN_R8	Yes
LED[4]	Location	PIN_D1	Yes
LED[5]	Location	PIN_F3	Yes
LED[6]	Location	PIN_B1	Yes
LED[7]	Location	PIN_L3	Yes
KEY[0]	Location	PIN_J15	Yes
KEY[1]	Location	PIN_E1	Yes
SW[0]	Location	PIN_M1	Yes
SW[1]	Location	PIN_T8	Yes
SW[2]	Location	PIN_B9	Yes
SW[3]	Location	PIN_M15	Yes
DRAM_ADDR[0]	Location	PIN_P2	Yes
DRAM_ADDR[1]	Location	PIN_N5	Yes
DRAM_ADDR[2]	Location	PIN_N6	Yes
DRAM_ADDR[3]	Location	PIN_M8	Yes
DRAM_ADDR[4]	Location	PIN_P8	Yes
DRAM_ADDR[5]	Location	PIN_T7	Yes
DRAM_ADDR[6]	Location	PIN_N8	Yes
DRAM_ADDR[7]	Location	PIN_T6	Yes
DRAM_ADDR[8]	Location	PIN_R1	Yes
DRAM_ADDR[9]	Location	PIN_P1	Yes
DRAM_ADDR[10]	Location	PIN_N2	Yes
DRAM_ADDR[11]	Location	PIN_N1	Yes
DRAM_ADDR[12]	Location	PIN_L4	Yes
DRAM_BA[0]	Location	PIN_M7	Yes
DRAM_BA[1]	Location	PIN_M6	Yes
DRAM_CKE	Location	PIN_L7	Yes
DRAM_CLK	Location	PIN_R4	Yes
DRAM_CS_N	Location	PIN_P6	Yes
DRAM_DQ[0]	Location	PIN_G2	Yes
DRAM_DQ[1]	Location	PIN_G1	Yes
DRAM_DQ[2]	Location	PIN_L8	Yes

DRAM_DQ[3]	Location	PIN_K5	Yes
DRAM_DQ[4]	Location	PIN_K2	Yes
DRAM_DQ[5]	Location	PIN_J2	Yes
DRAM_DQ[6]	Location	PIN_J1	Yes
DRAM_DQ[7]	Location	PIN_R7	Yes
DRAM_DQ[8]	Location	PIN_T4	Yes
DRAM_DQ[9]	Location	PIN_T2	Yes
DRAM_DQ[10]	Location	PIN_T3	Yes
DRAM_DQ[11]	Location	PIN_R3	Yes
DRAM_DQ[12]	Location	PIN_R5	Yes
DRAM_DQ[13]	Location	PIN_P3	Yes
DRAM_DQ[14]	Location	PIN_N3	Yes
DRAM_DQ[15]	Location	PIN_K1	Yes
DRAM_DQM[0]	Location	PIN_R6	Yes
DRAM_DQM[1]	Location	PIN_T5	Yes
DRAM_CAS_N	Location	PIN_L1	Yes
DRAM_RAS_N	Location	PIN_L2	Yes
DRAM_WE_N	Location	PIN_C2	Yes
I2C_SCLK	Location	PIN_F2	Yes
I2C_SDAT	Location	PIN_F1	Yes
G_SENSOR_CS_N	Location	PIN_G5	Yes
G_SENSOR_INT	Location	PIN_M2	Yes
GPIO_2[0]	Location	PIN_A14	Yes
GPIO_2[1]	Location	PIN_B16	Yes
GPIO_2[2]	Location	PIN_C14	Yes
GPIO_2[3]	Location	PIN_C16	Yes
GPIO_2[4]	Location	PIN_C15	Yes
GPIO_2[5]	Location	PIN_D16	Yes
GPIO_2[6]	Location	PIN_D15	Yes
GPIO_2[7]	Location	PIN_D14	Yes
GPIO_2[8]	Location	PIN_F15	Yes
GPIO_2[9]	Location	PIN_F16	Yes
GPIO_2[10]	Location	PIN_F14	Yes
GPIO_2[11]	Location	PIN_G16	Yes
GPIO_2[12]	Location	PIN_G15	Yes

GPIO_2_IN[0]	Location	PIN_E15	Yes
GPIO_2_IN[1]	Location	PIN_E16	Yes
GPIO_2_IN[2]	Location	PIN_M16	Yes
GPIO_0_IN[0]	Location	PIN_A8	Yes
GPIO_0[0]	Location	PIN_D3	Yes
GPIO_0_IN[1]	Location	PIN_B8	Yes
GPIO_0[1]	Location	PIN_C3	Yes
GPIO_0[2]	Location	PIN_A2	Yes
GPIO_0[3]	Location	PIN_A3	Yes
GPIO_0[4]	Location	PIN_B3	Yes
GPIO_0[5]	Location	PIN_B4	Yes
GPIO_0[6]	Location	PIN_A4	Yes
GPIO_0[7]	Location	PIN_B5	Yes
GPIO_0[8]	Location	PIN_A5	Yes
GPIO_0[9]	Location	PIN_D5	Yes
GPIO_0[10]	Location	PIN_B6	Yes
GPIO_0[11]	Location	PIN_A6	Yes
GPIO_0[12]	Location	PIN_B7	Yes
GPIO_0[13]	Location	PIN_D6	Yes
GPIO_0[14]	Location	PIN_A7	Yes
GPIO_0[15]	Location	PIN_C6	Yes
GPIO_0[16]	Location	PIN_C8	Yes
GPIO_0[17]	Location	PIN_E6	Yes
GPIO_0[18]	Location	PIN_E7	Yes
GPIO_0[19]	Location	PIN_D8	Yes
GPIO_0[20]	Location	PIN_E8	Yes
GPIO_0[21]	Location	PIN_F8	Yes
GPIO_0[22]	Location	PIN_F9	Yes
GPIO_0[23]	Location	PIN_E9	Yes
GPIO_0[24]	Location	PIN_C9	Yes
GPIO_0[25]	Location	PIN_D9	Yes
GPIO_0[26]	Location	PIN_E11	Yes
GPIO_0[27]	Location	PIN_E10	Yes
GPIO_0[28]	Location	PIN_C11	Yes
GPIO_0[29]	Location	PIN_B11	Yes

GPIO_0[30]	Location	PIN_A12	Yes
GPIO_0[31]	Location	PIN_D11	Yes
GPIO_0[32]	Location	PIN_D12	Yes
GPIO_0[33]	Location	PIN_B12	Yes
GPIO_1_IN[0]	Location	PIN_T9	Yes
GPIO_1[0]	Location	PIN_F13	Yes
GPIO_1_IN[1]	Location	PIN_R9	Yes
GPIO_1[1]	Location	PIN_T15	Yes
GPIO_1[2]	Location	PIN_T14	Yes
GPIO_1[3]	Location	PIN_T13	Yes
GPIO_1[4]	Location	PIN_R13	Yes
GPIO_1[5]	Location	PIN_T12	Yes
GPIO_1[6]	Location	PIN_R12	Yes
GPIO_1[7]	Location	PIN_T11	Yes
GPIO_1[8]	Location	PIN_T10	Yes
GPIO_1[9]	Location	PIN_R11	Yes
GPIO_1[10]	Location	PIN_P11	Yes
GPIO_1[11]	Location	PIN_R10	Yes
GPIO_1[12]	Location	PIN_N12	Yes
GPIO_1[13]	Location	PIN_P9	Yes
GPIO_1[14]	Location	PIN_N9	Yes
GPIO_1[15]	Location	PIN_N11	Yes
GPIO_1[16]	Location	PIN_L16	Yes
GPIO_1[17]	Location	PIN_K16	Yes
GPIO_1[18]	Location	PIN_R16	Yes
GPIO_1[19]	Location	PIN_L15	Yes
GPIO_1[20]	Location	PIN_P15	Yes
GPIO_1[21]	Location	PIN_P16	Yes
GPIO_1[22]	Location	PIN_R14	Yes
GPIO_1[23]	Location	PIN_N16	Yes
GPIO_1[24]	Location	PIN_N15	Yes
GPIO_1[25]	Location	PIN_P14	Yes
GPIO_1[26]	Location	PIN_L14	Yes
GPIO_1[27]	Location	PIN_N14	Yes
GPIO_1[28]	Location	PIN_M10	Yes

GPIO_1[29]	Location	PIN_L13	Yes
GPIO_1[30]	Location	PIN_J16	Yes
GPIO_1[31]	Location	PIN_K15	Yes
GPIO_1[32]	Location	PIN_J13	Yes
GPIO_1[33]	Location	PIN_J14	Yes
qspb	Location	PIN_A2	Yes
qsa	Location	PIN_A8	Yes
qsb	Location	PIN_B8	Yes
ct[0]	Location	PIN_A12	Yes
leds[0]	Location	PIN_A5	Yes
ct[1]	Location	PIN_C11	Yes
leds[1]	Location	PIN_B6	Yes
ct[2]	Location	PIN_E11	Yes
leds[2]	Location	PIN_B7	Yes
ct[3]	Location	PIN_C9	Yes
leds[3]	Location	PIN_A7	Yes
leds[4]	Location	PIN_C8	Yes
leds[5]	Location	PIN_E7	Yes
leds[6]	Location	PIN_E8	Yes
leds[7]	Location	PIN_F9	Yes
kpc[3]	Location	PIN_D5	Yes
kpc[2]	Location	PIN_A6	Yes
kpc[1]	Location	PIN_D6	Yes
kpc[0]	Location	PIN_C6	Yes
kpr[0]	Location	PIN_E9	Yes
kpr[1]	Location	PIN_F8	Yes
kpr[2]	Location	PIN_D8	Yes
kpr[3]	Location	PIN_E6	Yes
rgb_din	Location	PIN_D9	Yes
rgb_clk	Location	PIN_E10	Yes
rgb_cs	Location	PIN_B11	Yes
rgb_dc	Location	PIN_D11	Yes
rgb_res	Location	PIN_B12	Yes
jstk_sel	Location	PIN_G15	Yes
adc_cs_n	Location	PIN_A10	Yes

adc_saddr	Location	PIN_B10	Yes	
adc_sdat	Location	PIN_A9	Yes	
adc_sclk	Location	PIN_B14	Yes	
spkr	Location	PIN_B3	Yes	
point	Location	PIN_D12	Yes	
jstk_sel	Weak Pull-Up Resistor	On	Yes	lab1
PBY	Location	PIN_J14	Yes	
PBR	Location	PIN_K15	Yes	
PBG	Location	PIN_L13	Yes	
PBB	Location	PIN_N14	Yes	
LAMPS[0]	Location	PIN_L14	Yes	
LAMPS[1]	Location	PIN_M10	Yes	
LAMPS[2]	Location	PIN_J16	Yes	
LAMPS[3]	Location	PIN_J13	Yes	