ECE 271, Design Project

Benjamin Anderson II

Nov. 27th, 2022

1. Project Description

The inputs used for this design are the rightmost switches on the DE10-Lite, as well as the two buttons located above them. The switches correspond to the 6 seven segment displays, while the top button acts as a reset button, and the bottom button acts as the setting button.

The project itself is designed in such a way that upon start-up, or when the reset button is pressed, there will be an animation that plays in each of the seven segment displays. This animation is meant to tell the user that the value at that location has not been set yet. To set a number, the correct corresponding switch must be turned on (up position) and the set button must be clicked. When this set button is clicked all displays that have their switches on will activate and show the same pseudo-randomly generated number. This number will then remain until it is either reset or set to be a different random number. The random number generated will be between 1 and 6 as this is meant to be a dice rolling simulation.

Below is a video demonstration of the final product in action:
https://photos.app.goo.gl/jijkMkwhfCLyRpNv6

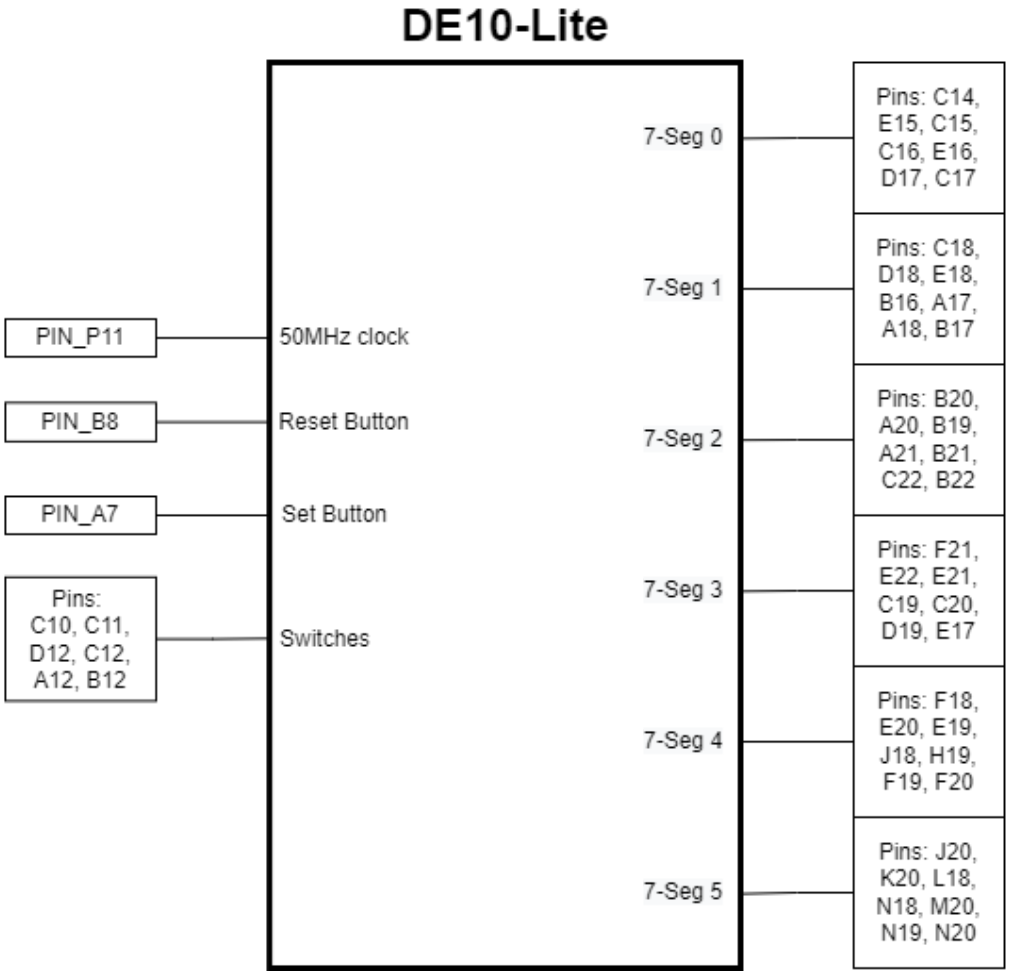

Figure 1: Hardware Diagram

The above figure shows the pins on the DE10-Lite that were used in this project. The output pins are listed from A to G according to the seven segment display standard.

2. High Level Description

Inputs: On board 50MHz clock oscillator, active low reset button, active low set button, 6 switches that correspond to the output displays.

Outputs: Six seven segment displays that correspond to the input pins

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

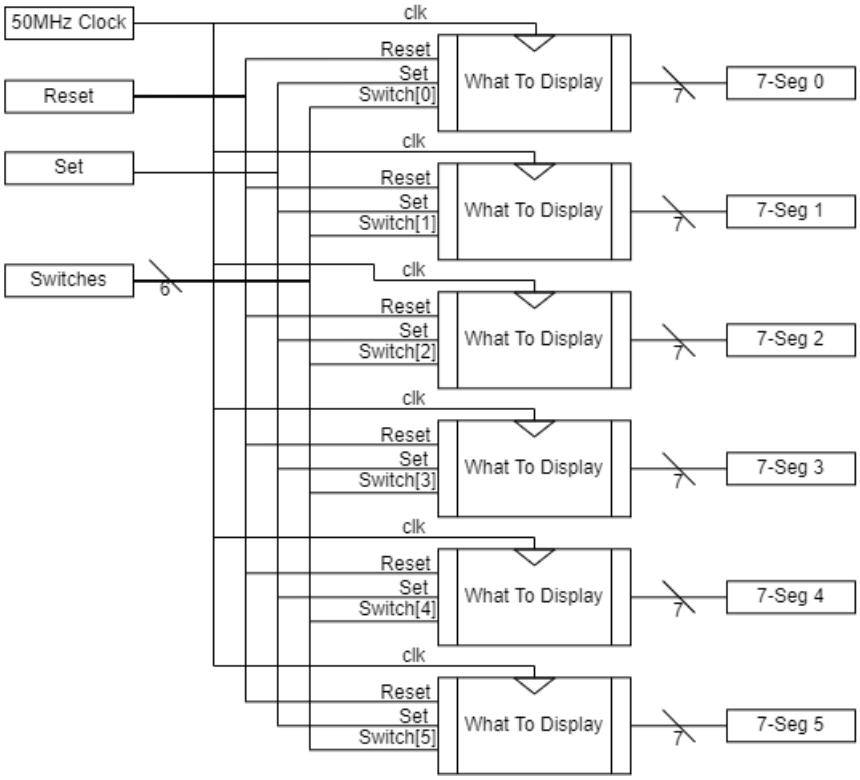

Figure 2: Top-Level Block Diagram

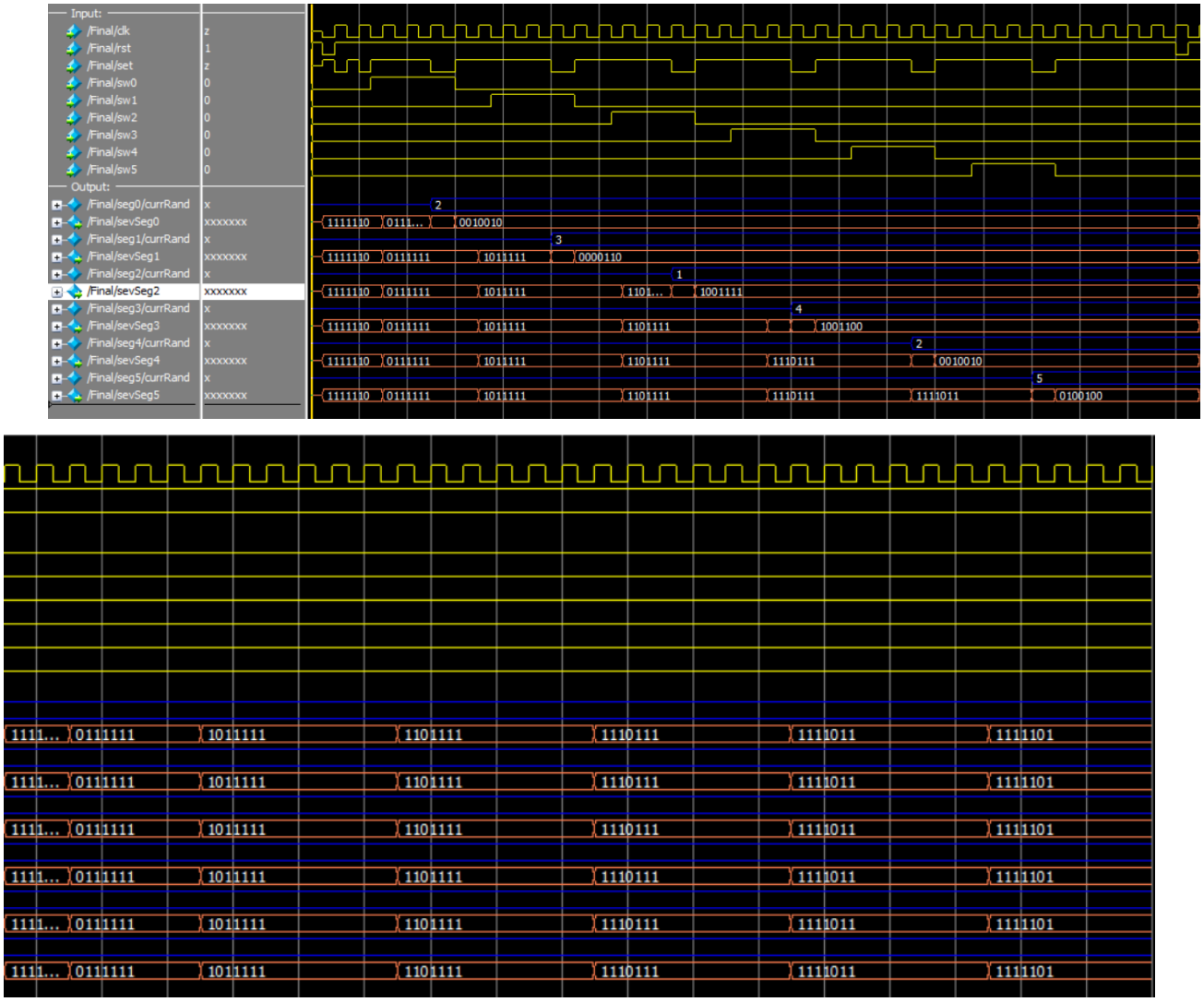

Figure 3: Top-Level Simulation

To explain the above simulation: First the reset and set buttons are pulsed to make sure the circuit is running properly. After this the first switch is activated, and the set button is pushed. This gives the random number 2, this random number is then encoded for the seven segment display (0010010). After this the switch is turned off and the next switch is activated. This cycle repeats until the last display gains its random value. Finally the reset button is pushed, telling the displays to cycle the loading animation, the same animation that they were displaying before being activated.

2.1. What To Display Unit

Inputs: Switch, active low reset button, active low set button, 50MHz clock

Outputs: Either a random number or a “loading animation.” If the set button is pushed while the switch is active, then a random number will be output to the seven segment display. If the reset button is pushed then the loading animation will restart.

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

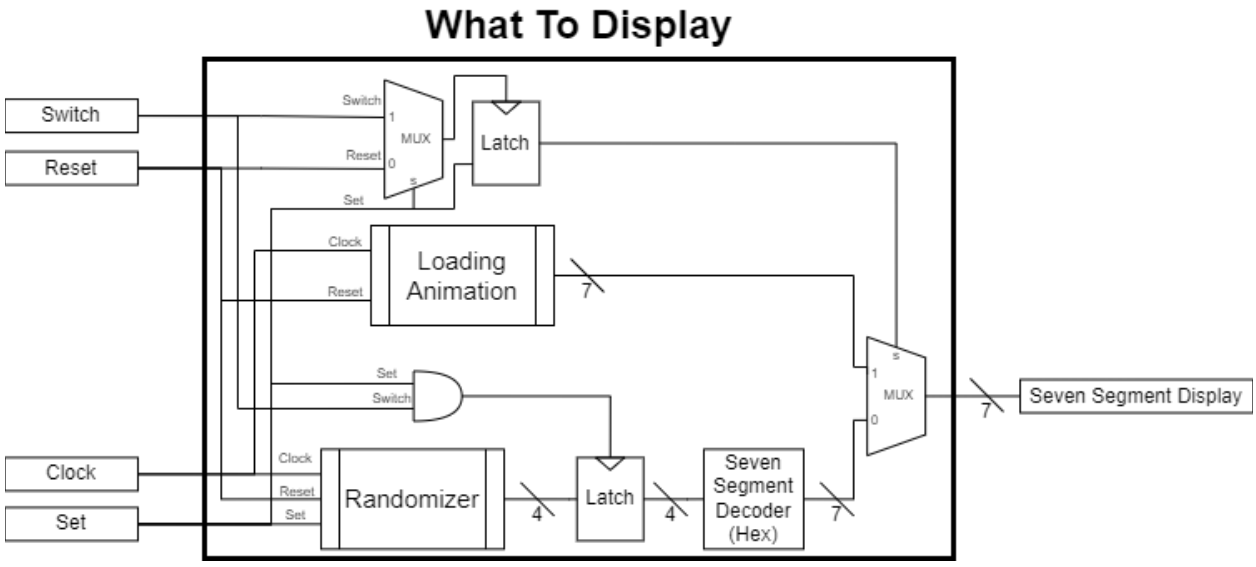

Figure 4: Logical Block Used in the Top Level Block Diagram

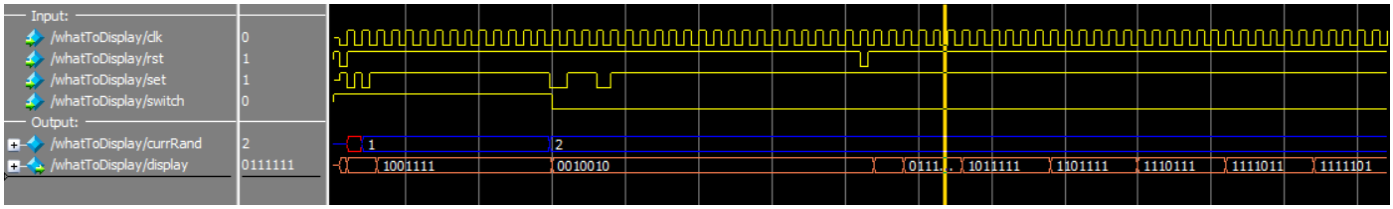

Figure 5: What To Display Simulation

This simulation is similar to the top-level simulation, the only difference is that there is only 1 switch to worry about, so the model is less confusing.

2.1.1. Loading Animation Unit

Input: 50MHz clock, active low reset button

Output: Animation that changes every half second in a pattern that slowly rotates around the seven segment display.

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

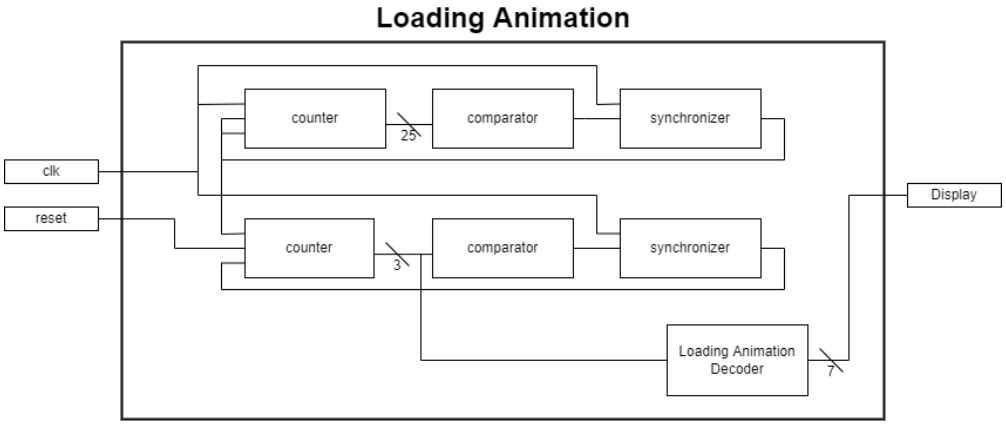

Figure 6: Loading Animation Block Diagram

This clock works by initially dividing the 50MHz clock signal into a 2Hz clock signal. It then repeatedly counts to 6 on the slower clock signal and outputs a frame of animation to the seven segment display depending on the current count value.

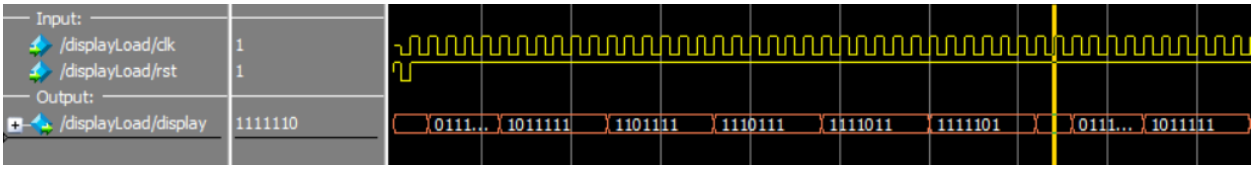

Figure 7: Loading Animation Simulation

The simulation for this file required me to speed up the clock, so it is running at 25MHz, rather than 2 Hz. This is the same for Figures 3 and 5. As you can see the default value is hit for a short period of time every cycle due to the lost time from the synchronizer, which will be discussed further later. However this is only 1/25000000th of a second, so I feel it is an acceptable loss.

2.1.1.1. Counter Unit

Input: Clock signal, two reset buttons, one resetting to a value and the other resetting to 0

Output: Continuously incrementing value

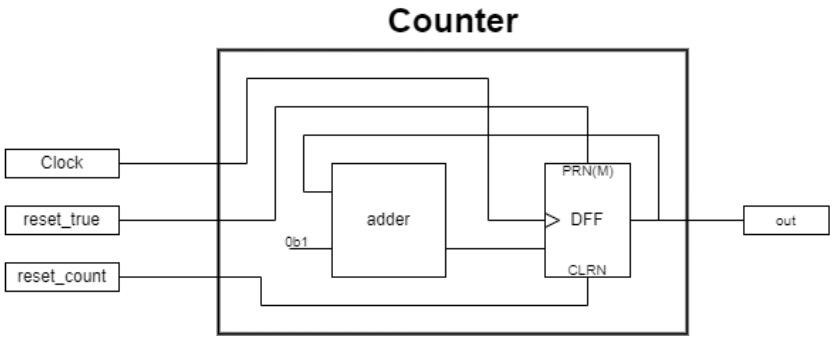

Figure 8: Modified Counter Block Diagram

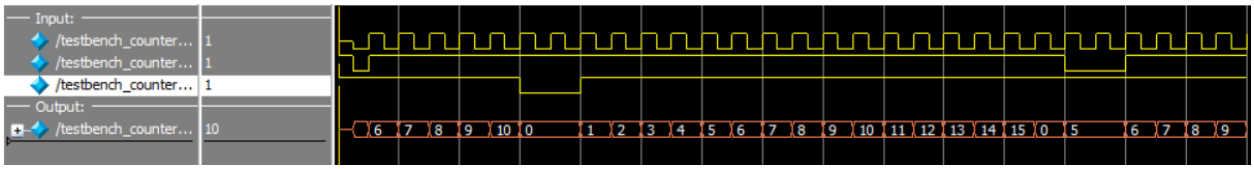

Figure 9: Modified Counter Simulation

For the sake of this simulation I set the preset for reset_true to be 5, as can be seen on the right side of the simulation when reset_true is pulsed after the counter naturally resets

2.1.1.2. Comparator Unit

Input: A and M

Output: Whether A is less than M

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

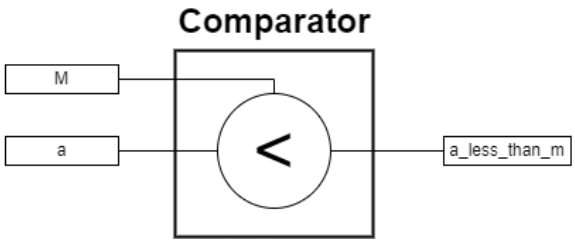

Figure 10: Comparator Block Diagram

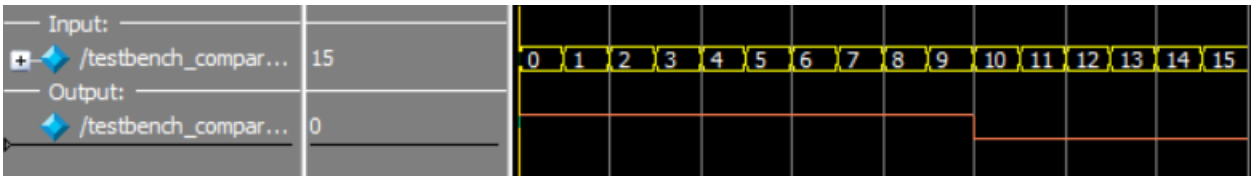

Figure 11: Comparator Simulation

This simulation is fairly self explanatory, as when ‘a’ reaches 10 (the preset M value) the output turns from being a 1 to being a 0, since ‘a’ is no longer less than M.

2.1.1.3. Synchronizer Unit

Input: Data to sync with clock, and clock signal

Output: Data that is now in sync with the clock

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

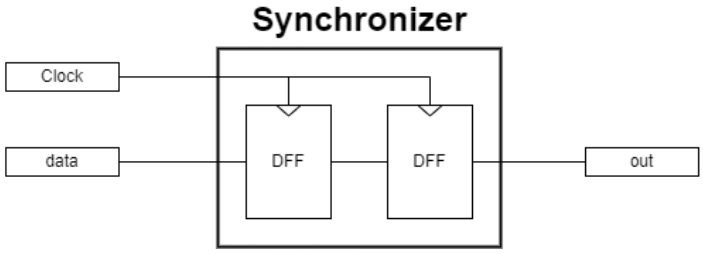

Figure 12: Synchronizer Block Diagram

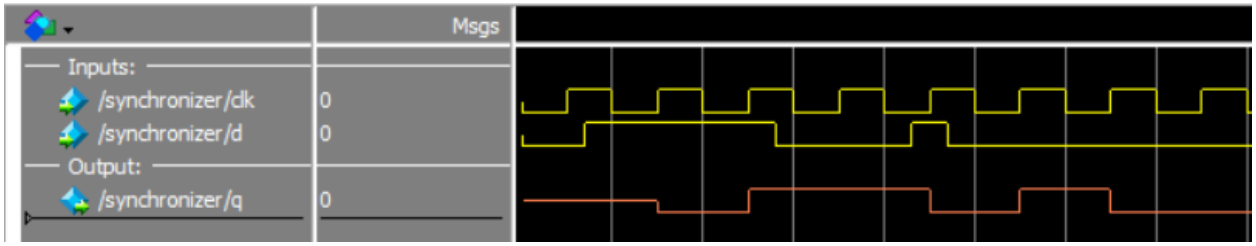

Figure 13: Synchronizer Simulation

This block shows the time delay of the synchronizer circuit. This time delay is 2 clock cycles, and will turn extremely short pulses, like the second, into pulses that last a full clock cycle.

2.1.1.4. Loading Animation Decoder Unit

Input: 3-bit number that ranges between 0 and 7

Output: 7-bit active low display, where segment A is the most significant bit.

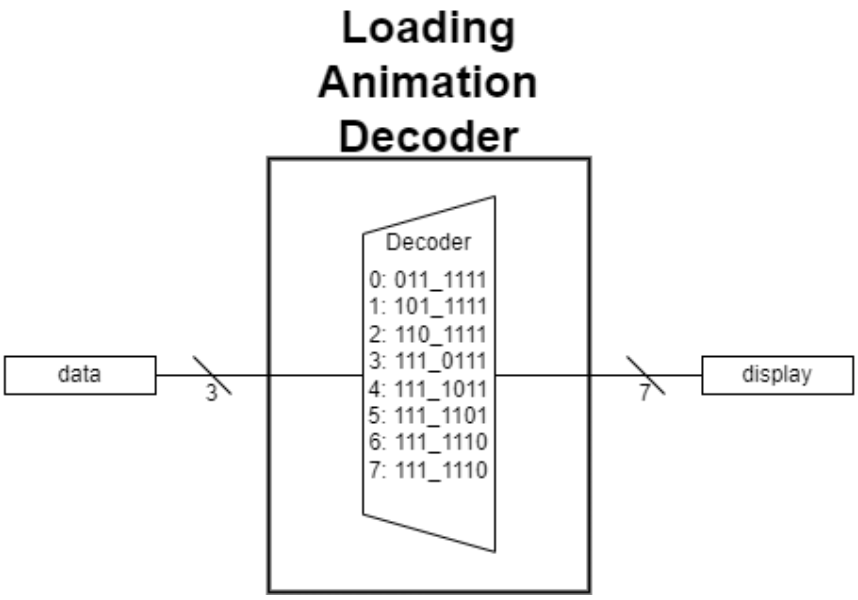

Figure 14: Loading Animation Decoder Block Diagram

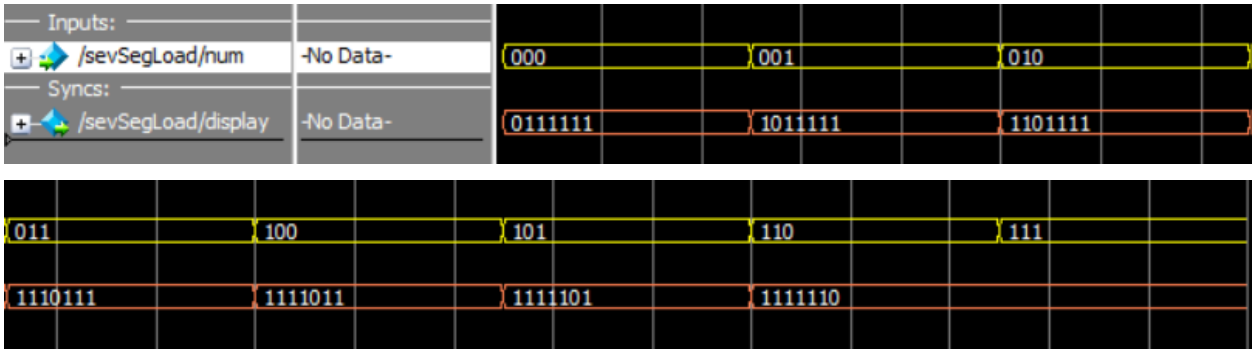

Figure 15: Loading Animation Decoder Simulation

This simulation image is different from those above in that it had to be broken into two images to be able to show all of the data properly. Though the data itself isn't very difficult to understand. As you can see the 0 moves to the right every time the input number increases by 1 (except for the last cycle, as there is no 8th position for the 0 to move to). This causes the animation to cycle an active light around the seven segment display.

2.1.2. Random Number Generator Unit

Input: Clock signal, active low reset button, active low set button
Output: 3-bit random number between 1 and 6

Nov. 30, 2022	FPGA Dice Machine	Benjamin Anderson II	934-353-159

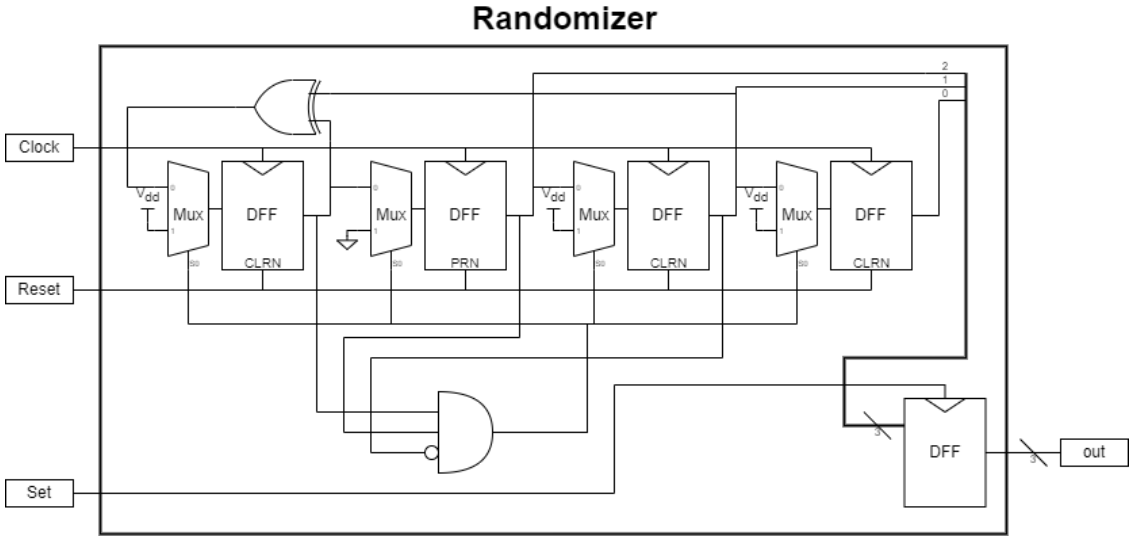

Figure 16: Random Number Generator Block Diagram

This 4-bit linear feedback shift register acts as my random number generator. The specifics of how it works are as follows: It begins initialized to a value (0100) and proceeds to shift the bits

to the right every clock cycle. The first bit is the XOR of the first D Flip-Flop and the 3rd, while there is also an AND gate checking to see if the shift register is going to run into a 7. If it is going to run into a 7, then the register skips it and goes to 3 instead. Finally, the output only changes when the set button is pushed.

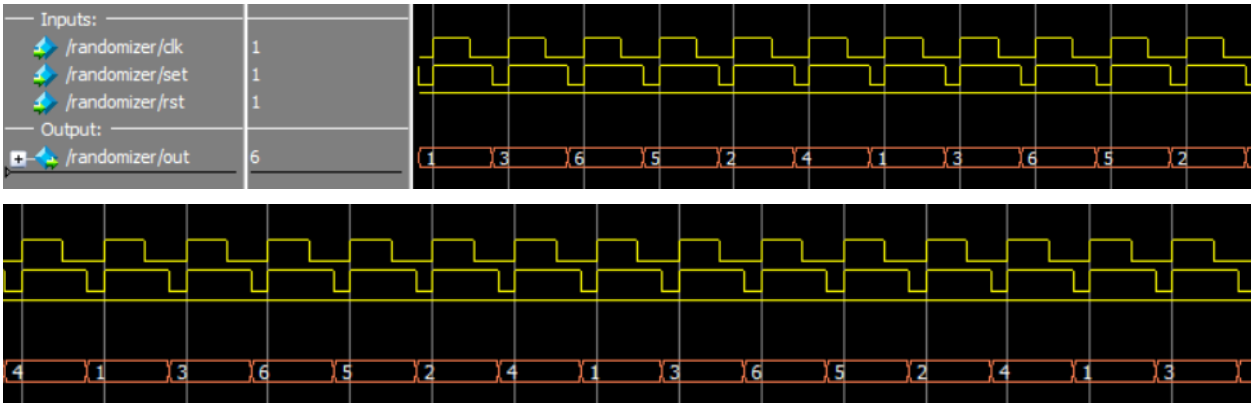

Figure 17: Random Number Generator Simulation

This simulation simply shows the cycling random numbers that the random number generator generates. As you can see the cycle never comes across 0 or 7, as they have either been skipped over or were never a part of the cycle to begin with.

2.1.3. Seven Segment Decoder Unit

Input: 4-bit number that is between 0 and 15

Output: 7-bit active low display that is capable of displaying the hexadecimal numbers 0-F, where segment A is the most significant bit

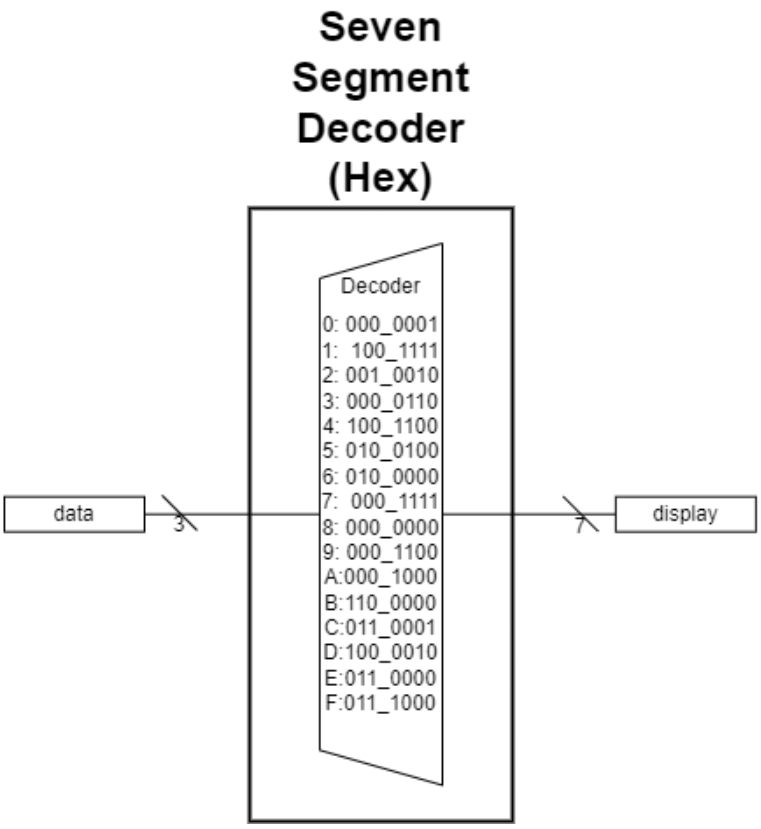

Figure 18: Seven Segment Decoder Block Diagram

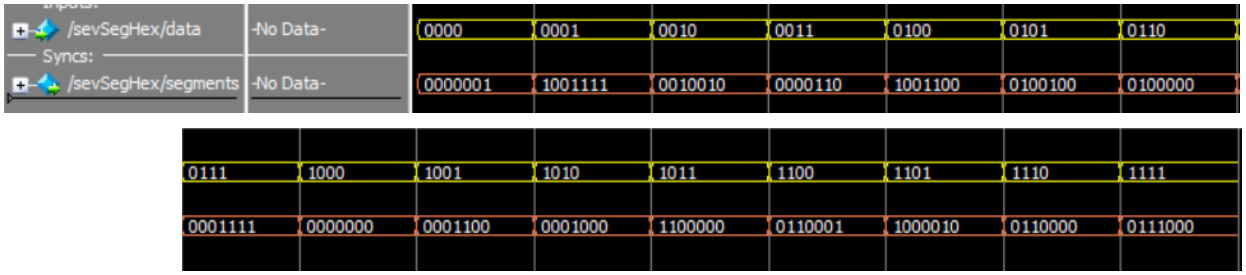

Figure 19: Seven Segment Decoder Simulation

This simulation simply shows that the seven segment decoder can work for all input values 0-15.

A. SystemVerilog Files

```
/**********************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      Final
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Top Level for connecting each seven segment
       display with each switch
**********************************************************************/
module Final
   (input logic clk,
    input  logic rst,
    input  logic set,
    input  logic sw0,
    input  logic sw1,
    input  logic sw2,
    input  logic sw3,
    input  logic sw4,
    input  logic sw5,
    output logic[6:0] sevSeg0,
    output logic[6:0] sevSeg1,
    output logic[6:0] sevSeg2,
    output logic[6:0] sevSeg3,
    output logic[6:0] sevSeg4,
    output logic[6:0] sevSeg5);

    whatToDisplay seg0(clk, rst, set, sw0, sevSeg0[6:0]);
    whatToDisplay seg1(clk, rst, set, sw1, sevSeg1[6:0]);
    whatToDisplay seg2(clk, rst, set, sw2, sevSeg2[6:0]);
    whatToDisplay seg3(clk, rst, set, sw3, sevSeg3[6:0]);
    whatToDisplay seg4(clk, rst, set, sw4, sevSeg4[6:0]);
    whatToDisplay seg5(clk, rst, set, sw5, sevSeg5[6:0]);
endmodule
```

A.1. What To Display

```
/*************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      whatToDisplay
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Module that informs the seven segment display
      what it is supposed to dispay at any given moment.
 *************************************************************/
module whatToDisplay
   (input logic clk,
    input  logic rst,
    input  logic set,
    input  logic switch,
    output logic[6:0] display);

    logic[3:0] randNum;     // The number outputted by the randomizer module
    logic[6:0] rng_display; // The seven segment value of the current random number
    logic[6:0] load_display;// The seven segment value of the loading animation
    logic[3:0] currRand;    // The random number that will be displayed as rng_display
    logic setter;

    randomizer randVal(clk, set, rst, randNum[3:0]);   // always randomizing a number (1-6) in the background
    sevSegHex dis(currRand[3:0], rng_display[6:0]);    // converts curr_rand into a seven segment display

    displayLoad load(clk, rst, load_display[6:0]);     // returns a loading animation

    // When set and switch are both active, setter turns on. Setter only turns off when rst is active
    always_ff @(negedge set, negedge rst, posedge clk) begin
       if(!set) begin
          if (switch) begin
             currRand[3:0] <= randNum[3:0];
             setter <= 1;
          end
       end else if(!rst) setter <= 0;
    end

    //checks to see if setter is active
    //when setter is active, the random number from currRand is displayed
    //when setter is not active, the loading animation is shown to the user.
    always_ff @(negedge rst, posedge setter, posedge clk) begin
       if(setter) begin
          display[6:0] <= rng_display[6:0];
       end else begin
          display[6:0] <= load_display[6:0];
       end
    end

endmodule
```

A.1.1. Loading Animation

```
/*************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      displayLoad
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Outputs a loading animation
 *************************************************************/
module displayLoad
   (input logic clk,
    input  logic rst,
    output logic[6:0] display);


    logic[24:0] fast_count; // output of the counter that goes to 25,000,000
    logic[2:0]  count_val;  // informs which stage of the animation is beign displayed

    logic clk_short;  // turns high every half second
    logic count_rst;  // tells the bottom counter when to reset
    logic less_than_6;// outputs HIGH, when bottom counter is less than 6
    logic half_sec;   // the clock used for the animation

    //The logical block that counts to a 25,000,000 and resets when it is hit
    counter #(25, 0) fast_counter(clk, half_sec, half_sec, fast_count[24:0]);
    comparator #(25, 25000000) fast_comp(fast_count[24:0], clk_short);
    synchronizer fast_sync(clk, clk_short, half_sec);

    //The logical block that counts to 6 and resets when it is hit.
    //(uses the output of the last block as the clock)
    counter #(3, 6) count(half_sec, rst, count_rst, count_val[2:0]);
    comparator #(3, 6) lessThan6(count_val[2:0], less_than_6);
    synchronizer sync(clk, less_than_6, count_rst);

    //displays the animation stage dependent on the current count value.
    sevSegLoad segs(count_val[2:0], display[6:0]);
endmodule
```

A.1.1.1. Counter

```
/*************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      counter
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      modified counter module that will reset to a
       given value upon `reset_true`
 *************************************************************/
module counter
   #(parameter N, // bit size of the output
     parameter M) // number to reset to upon `reset_true`
   (input logic clk,
    input  logic reset_true,
    input  logic reset_count,
    output logic[N-1:0] out);

    //resets output to 0 upon reset_count
    //resets output to M upon reset_true
    //increments output otherwise
    always_ff @(posedge clk, negedge reset_true, negedge reset_count) begin
       if(!reset_true) out <= M;
       else if (!reset_count) out <= 0;
       else out <= out + 1;
    end
endmodule
```

A.1.1.2. Comparator

```
/**********************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      comparator
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Simple logic block that outputs true so long as
      the input is less than the M parameter
**********************************************************************/
module comparator
   #(parameter N, // size of the input
   parameter   M) // number being compared to
   (input logic[N-1:0] a,
   output logic a_lt_M);

   assign a_lt_M = (a < M);
endmodule
```

A.1.1.3. Synchronizer

```
/**********************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      synchronizer
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Aligns an asynchronous signal to the clock, at
      the expense of 2 clock cycles
**********************************************************************/
module synchronizer
   (input logic clk,
   input  logic d,
   output logic q);

   logic n1;
   always_ff @(posedge clk) begin
      n1 <= d;
      q <= n1;
   end
endmodule
```

A.1.1.4. Loading Animation

```
/**********************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      sevSegLoad
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Seven segment decoder used to determine which
      stage of animation to display when loading
**********************************************************************/
module sevSegLoad
   (input logic[2:0] num,
   output logic[6:0] display);
   always_comb
      case(num)
         0:       display<=7'b011_1111;
         1:       display<=7'b101_1111;
         2:       display<=7'b110_1111;
         3:       display<=7'b111_0111;
         4:       display<=7'b111_1011;
         5:       display<=7'b111_1101;
         default: display<=7'b111_1110;
      endcase
endmodule
```

A.1.2. Randomizer

```
/**********************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      randomizer
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      Randomizes number between 1 and 6
**********************************************************************/
module randomizer
   (input logic clk,
   input  logic set,
   input  logic rst,
   output logic[2:0] out);

   logic dff1;
   logic dff2;
   logic dff3;
   logic dff4;

   //will never give a 0, but will give a 7
   always_ff@(posedge clk, negedge rst) begin
      if(!rst) begin
         dff1 <= 0;
         dff2 <= 1;
         dff3 <= 0;
         dff4 <= 0;
      end else begin
         dff1 <= dff1 ^ dff3;
         dff2 <= dff1;
         dff3 <= dff2;
         dff4 <= dff3;
         if(dff2 == 1 && dff3 == 1 && dff4 == 0) begin //7 will be hit (iterate past it)
            dff1 <= 1;
            dff2 <= 0;
            dff3 <= 1;
            dff4 <= 1;
         end
      end
   end
   always_ff @(negedge set)
      out[2:0] <= {dff4, dff3, dff2};
endmodule
```

A.1.3. Seven Segment Decoder

```
/*************************************************************************
 * Company:          Oregon State University
 * Engineer:         Benjamin Anderson II
 *
 * Create Date:      11/27/2022
 * Design Name:      Final
 * Module Name:      sevSegHex
 * Project Name:     ECE_272_Final
 * Target Devices:   DE10-Lite
 * Tool Versions:    Quartus Prime 18.0
 * Description:      A standard seven segment display decoder for
      hexadecimal numbers where the MSB is A, and the LSB is g
*************************************************************************/
module sevSegHex(input  logic[3:0] data,
                 output logic[6:0] segments);
  always_comb
    case(data)
      4'h0:    segments=7'b000_0001;
      4'h1:    segments=7'b100_1111;
      4'h2:    segments=7'b001_0010;
      4'h3:    segments=7'b000_0110;
      4'h4:    segments=7'b100_1100;
      4'h5:    segments=7'b010_0100;
      4'h6:    segments=7'b010_0000;
      4'h7:    segments=7'b000_1111;
      4'h8:    segments=7'b000_0000;
      4'h9:    segments=7'b000_1100;

      4'hA:    segments=7'b000_1000;
      4'hB:    segments=7'b110_0000;
      4'hC:    segments=7'b011_0001;
      4'hD:    segments=7'b100_0010;
      4'hE:    segments=7'b011_0000;
      4'hF:    segments=7'b011_1000;
      default: segments=7'b111_1111;
    endcase
endmodule
```

B. Simulation Files

```
quit -sim
vsim -gui -t ps -default_radix bin work.Final

# Inputs
add wave -divider Input:
add wave -color yellow clk rst set sw0 sw1 sw2 sw3 sw4 sw5

# Outputs
add wave -divider Output:
add wave -color blue -radix unsigned seg0.currRand
add wave -color coral sevSeg0
add wave -color blue -radix unsigned seg1.currRand
add wave -color coral sevSeg1
add wave -color blue -radix unsigned seg2.currRand
add wave -color coral sevSeg2
add wave -color blue -radix unsigned seg3.currRand
add wave -color coral sevSeg3
add wave -color blue -radix unsigned seg4.currRand
add wave -color coral sevSeg4
add wave -color blue -radix unsigned seg5.currRand
add wave -color coral sevSeg5


# Initialize
force rst 1 0ps, 0 5ps, 1 10ps
force set 1 5ps, 0 10ps, 1 15ps, 0 20ps, 1 25ps
force clk 0 5ps, 1 10ps -r 10ps
force sw0 0 0ps
force sw1 0 0ps
force sw2 0 0ps
force sw3 0 0ps
force sw4 0 0ps
force sw5 0 0ps

## FORCE SWITCHES

# Force Switch 0
force sw0 1 25ps
force set 0 50, 1 60ps
force sw0 0 60ps

# Force Switch 1
force sw1 1 75ps
force set 0 100, 1 110ps
force sw1 0 110ps

# Force Switch 2
force sw2 1 125ps
force set 0 150, 1 160ps
force sw2 0 160ps

# Force Switch 3
force sw3 1 175ps
force set 0 200, 1 210ps
force sw3 0 210ps

# Force Switch 4
force sw4 1 225ps
force set 0 250, 1 260ps
force sw4 0 260ps

# Force Switch 5
force sw5 1 275ps
force set 0 300, 1 310ps
force sw5 0 310ps

# Reset all switches
force rst 0 360ps, 1 365 ps
run 720ps
```

B.1. What To Display Simulation File

```
quit -sim
vsim -gui -t ps -default_radix bin work.whatToDisplay

# Inputs
add wave -divider Input:
add wave -color yellow clk rst set switch

# Outputs
add wave -divider Output:
add wave -color blue -radix unsigned currRand
add wave -color coral display


# Force Values
force rst 1 0ps, 0 5ps, 1 10ps
force set 1 5ps, 0 10ps, 1 15ps, 0 20ps, 1 25ps
force switch 1 0ps
force clk 0 5ps, 1 10ps -r 10ps

force set 0 148ps, 1 160ps
force switch 0 150ps

force set 0 180ps, 1 190ps
force rst 0 360ps, 1 365ps

run 720ps
```

B.1.1. Loading Animation Simulation File

```
quit -sim
vsim -gui -t ps -default_radix bin work.displayLoad

# Inputs
add wave -divider Input:
add wave -color yellow clk rst

# Outputs
add wave -divider Output:
add wave -color coral display


# Force Values
force rst 1 0ps, 0 5ps, 1 10ps
force clk 0 5ps, 1 10ps -r 10ps

run 1000ps
```

B.1.1.1. Counter Simulation Files

```
module testbench_counter();
    logic       clk, reset_true, reset_count;
    logic[3:0]  out;

    //instantiate DUT
    counter #(4, 5) dut(clk, reset_true, reset_count, out[3:0]);
endmodule
```

```
quit -sim
vsim -gui -t ps -default_radix unsigned work.testbench_counter

# Inputs
add wave -divider Input:
add wave -color yellow clk reset_true reset_count

# Outputs
add wave -divider Output:
add wave -color coral out


# Force Values
force reset_true 1 0ps, 0 5ps, 1 10ps
force reset_count 1 0ps, 0 60ps, 1 80ps
force clk 0 5ps, 1 10ps -r 10ps

force reset_true 0 240ps, 1 260ps

run 300ps
```

B.1.1.2. Comparator Simulation Files

```
module testbench_comparator();
    logic[3:0]  in;
    logic       out;
    comparator #(4, 10) dut(in[3:0], out); //compare against 10
endmodule
```

```
quit -sim
vsim -gui -t ps -default_radix unsigned work.testbench_comparator

# Inputs
add wave -divider Input:
add wave -color yellow in

# Outputs
add wave -divider Output:
add wave -color coral out


# Force Values
force in 0 0ps
force in 1 5ps
force in 2 10ps
force in 3 15ps

force in 4 20ps
force in 5 25ps
force in 6 30ps
force in 7 35ps

force in 8 40ps
force in 9 45ps
force in 10 50ps
force in 11 55ps

force in 12 60ps
force in 13 65ps
force in 14 70ps
force in 15 75ps

run 80ps
```

B.1.1.3. Loading Animation Decoder Simulation Files

```
quit -sim
vsim -gui -t ps -default_radix bin work.sevSegLoad

# Inputs
add wave -divider Inputs:
add wave -color yellow num

# Outputs
add wave -divider Syncs:
add wave -color coral display

# Force Values

force num 000 0ps
force num 001 5ps
force num 010 10ps
force num 011 15ps

force num 100 20ps
force num 101 25ps
force num 110 30ps
force num 111 35ps

run 40ps
```

B.1.2. Random Number Generator Simulation File

```
quit -sim
vsim -gui -t ps -default_radix unsigned work.randomizer

# Inputs
add wave -divider Inputs:
add wave -color yellow clk set rst

# Outputs
add wave -divider Output:
add wave -color coral out

# Flip-Flop Values
add wave -divider FlipFlops
add wave -color green dff1 dff2 dff3 dff4

# Force Values
force rst 1 0ps, 0 5ps, 1 10ps
force clk 0 5ps, 1 10ps -r 10ps
force set 1 0ps, 0 8ps -r 10ps


run 300ps
```

B.1.3. Seven Segment Decoder Simulation File

```
quit -sim
vsim -gui -t ps -default_radix bin work.sevSegHex

# Inputs
add wave -divider Inputs:
add wave -color yellow data

# Outputs
add wave -divider Syncs:
add wave -color coral segments

# Force Values

force data 0000 0ps
force data 0001 5ps
force data 0010 10ps
force data 0011 15ps

force data 0100 20ps
force data 0101 25ps
force data 0110 30ps
force data 0111 35ps

force data 1000 40ps
force data 1001 45ps
force data 1010 50ps
force data 1011 55ps

force data 1100 60ps
force data 1101 65ps
force data 1110 70ps
force data 1111 75ps

run 80ps
```