# ECE 271 Project, FPGA Dice Machine, Group 17

Niranjan Varma, Cameron Crutcher

Dec 3, 2021

## Contents

## 1. Project Description

Inputs: This design reads an FPGA board. The inputs for this project are reading 6 slide switches and 2 push buttons. There is also a 50MHz clock input for the design.

Outputs: The design displays LEDs in a loading sequence from right to left. It also displays the seven segments for the six possible digits on the FPGA.

As seen from figure 1, the design for the making of the machine was quite repetitive. We had made a Led_lighter_neg(see 2.1.1), which would invert the active low push buttons to make a true signal output. These push buttons are the clk_button which is an enable. The other is the reset which is used to reset the board. The output signal would be the enable for the counter(see 2.1.2). Once the counter is turned on, it also receives a 50 MHz clock and its own reset port. The counter outputs a 28 bit value that would become the divided clock bits.

These bits are what make the FPGA dice machine random. These bits are added in varying significance into Led_lighter_pos(see 2.1.2) into sequential LEDs to allow for the lighting up of the LEDs to be a loading motion. The last LED lighting up signifies the loading to be complete. Which is why it is used as a clock input for the random_mux(see 2.2.5).

The divided clock bits are also inputted into the random_counters which give out varying random values between 1 and 6. The output goes to the random_mux which checks if the toggle is true. If so, the seven segment display receives the number. If not, the seven segment display receives the default value.

When looking at figure 2, the pin assignments for the design can be seen. These pin assignments are all in the FPGA itself because it is the only device used in the execution of the project. Clearly the switches, LEDs and 7 segment displays are all 3.3 V LVTTL while the 2 push buttons are 3.3 V Schmitt Trigger.

**To see the Video execution of the project**
https://youtu.be/oV7mrANdz0Q

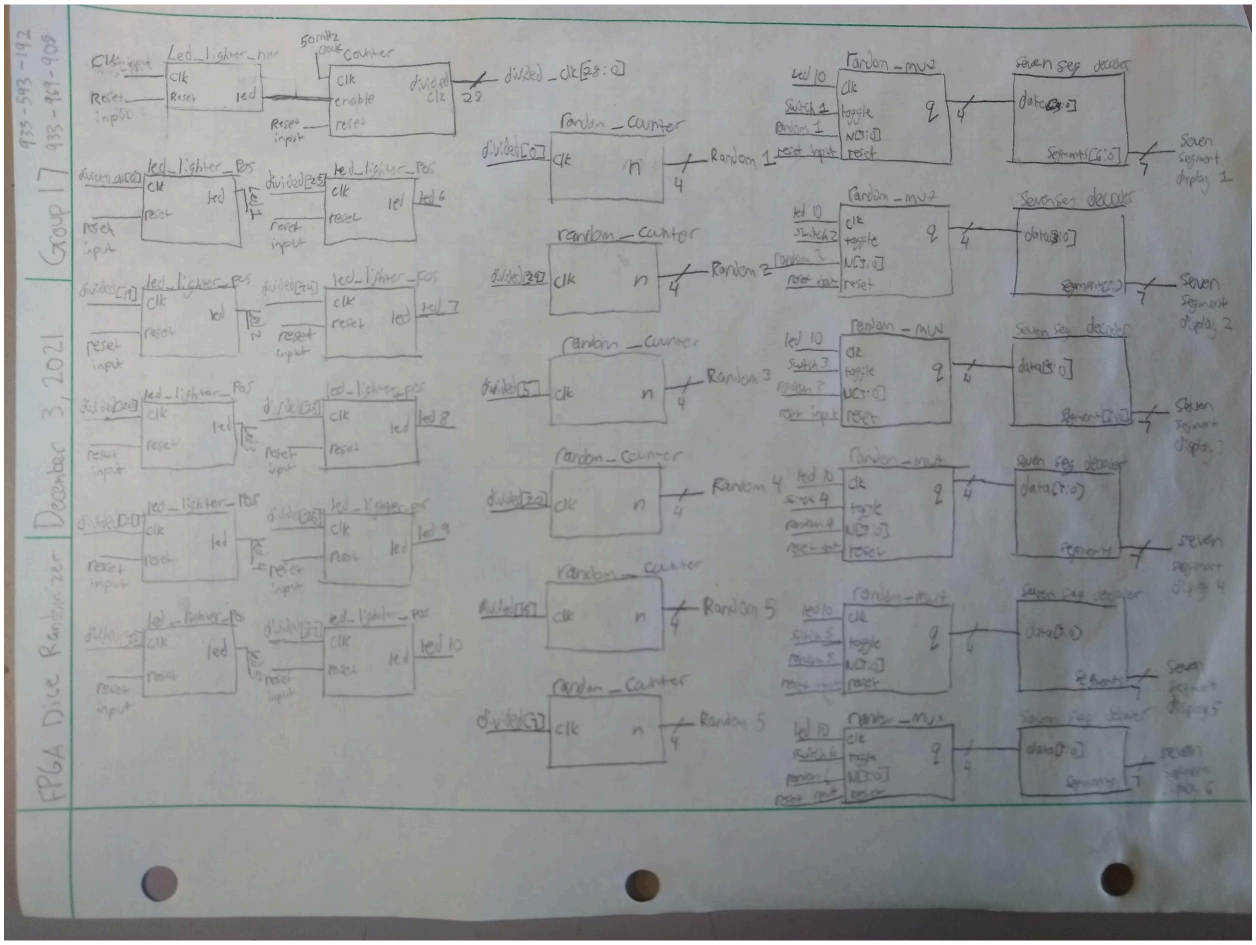


Figure 1: The top level diagram for the Dice Machine

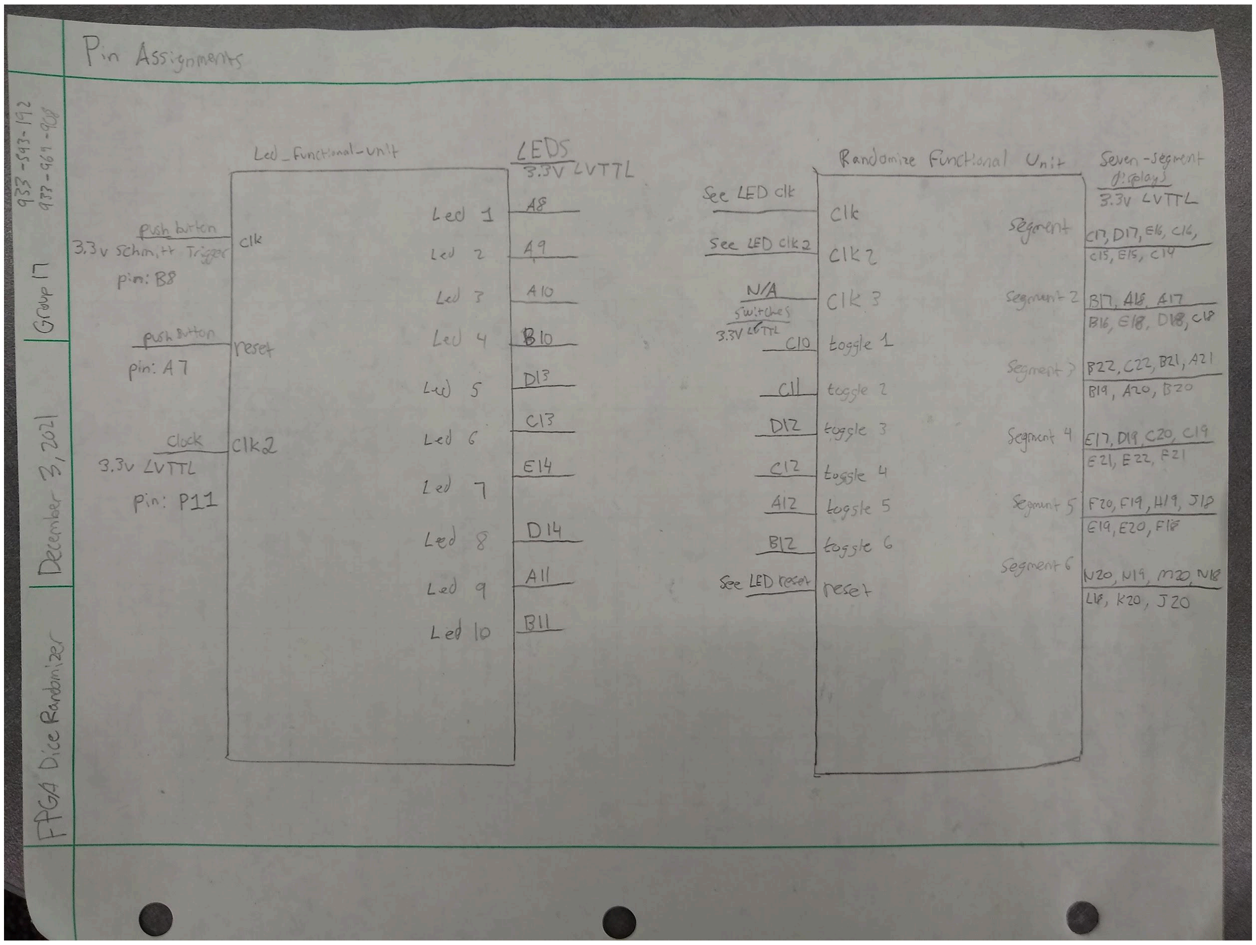


Figure 2: Hardware Diagram for the dice machine

## 2. High Level Descriptions

Inputs: This design reads an FPGA board. The inputs for this project are reading 6 slide switches and 2 push buttons. There is also a 50MHz clock input for the design.

Outputs: The design displays LEDs in a loading sequence from right to left. It also displays the seven segments for the six possible digits on the FPGA.

In the making of the project, we had the idea to use system verilog to implement a random number generator inside the randomize functional unit in figure 3(specifically in the random_mux). However, we realized that the $urandom function is not synthesizable. Thus the reason for creating a random number counter inside the particular functional unit.

Both randomize and LED functional units have only 2 similarities. One is that they both utilize bits from the clock divider, reset and enable inputs. As seen in Figure 5 and figure 13. They also have one connecting node where the output from the final LED becomes the clock input for the randomizer unit.

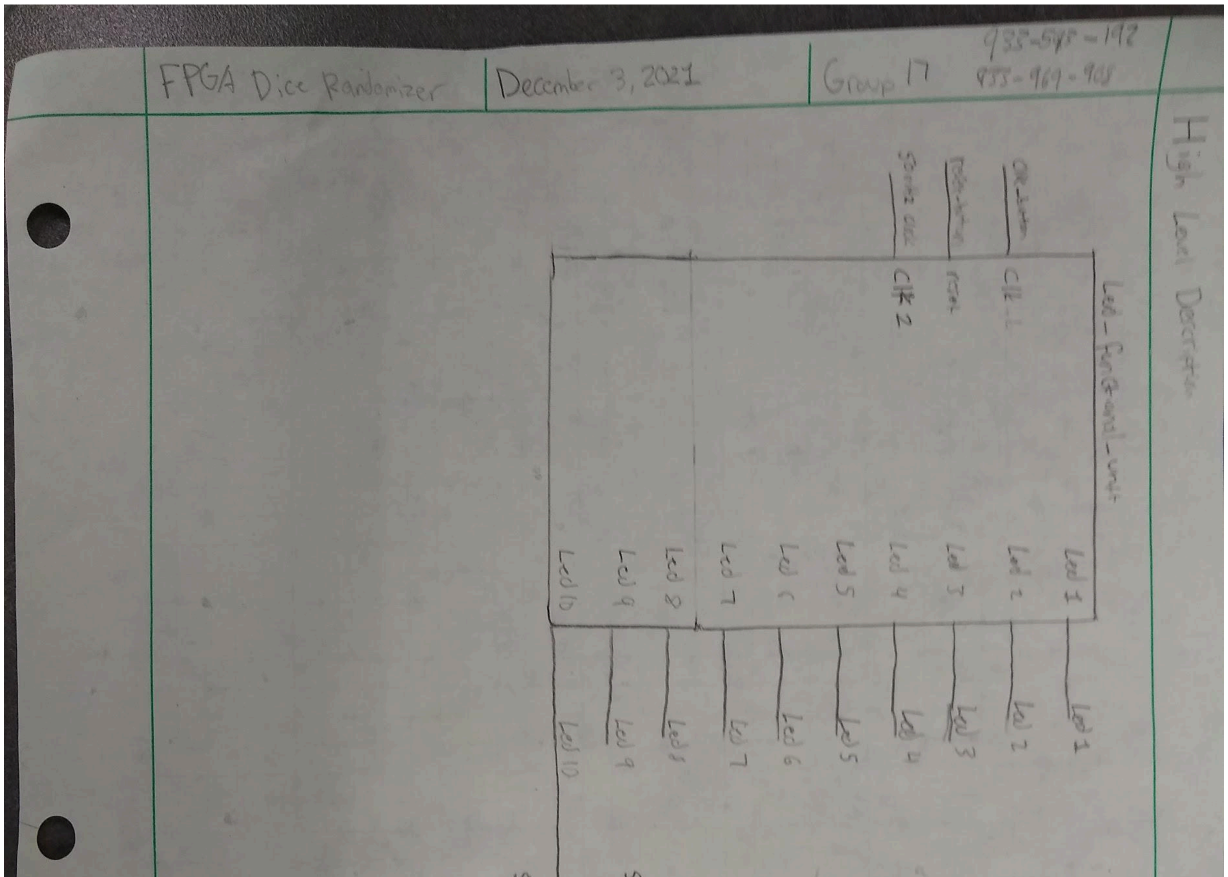
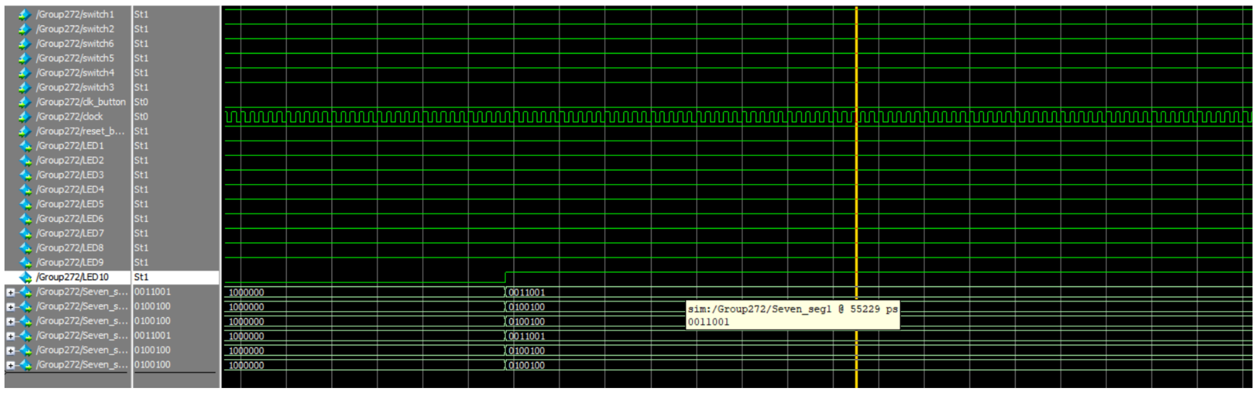


Figure 3: The high level description diagram for the dice machine. Refer to A.1



```
/Group272/switch1        St1
/Group272/switch2        St1
/Group272/switch6        St1
/Group272/switch5        St1
/Group272/switch4        St1
/Group272/switch3        St1
/Group272/clk_button     St0
/Group272/clock          St0
/Group272/reset_b...     St1
/Group272/LED1           St1
/Group272/LED2           St1
/Group272/LED3           St1
/Group272/LED4           St1
/Group272/LED5           St1
/Group272/LED6           St1
/Group272/LED7           St1
/Group272/LED8           St1
/Group272/LED9           St1
/Group272/LED10          St1
/Group272/Seven_s...     0011001    1000000    0011001
/Group272/Seven_s...     0100100    1000000    0100100
/Group272/Seven_s...     0100100    1000000    0100100
/Group272/Seven_s...     0011001    1000000    0011001
/Group272/Seven_s...     0100100    1000000    0100100
/Group272/Seven_s...     0100100    1000000    0100100

sim:/Group272/Seven_seg1 @ 55229 ps
0011001
```

Figure 4: Simulation for the Top Level diagram/ whole project. Refer to B.1

## 2.1 Clock divider/LED function unit

Inputs: This has a clk_button, a reset_button, and a 50 MHz clock signal.

Outputs: 10 separate LEDs

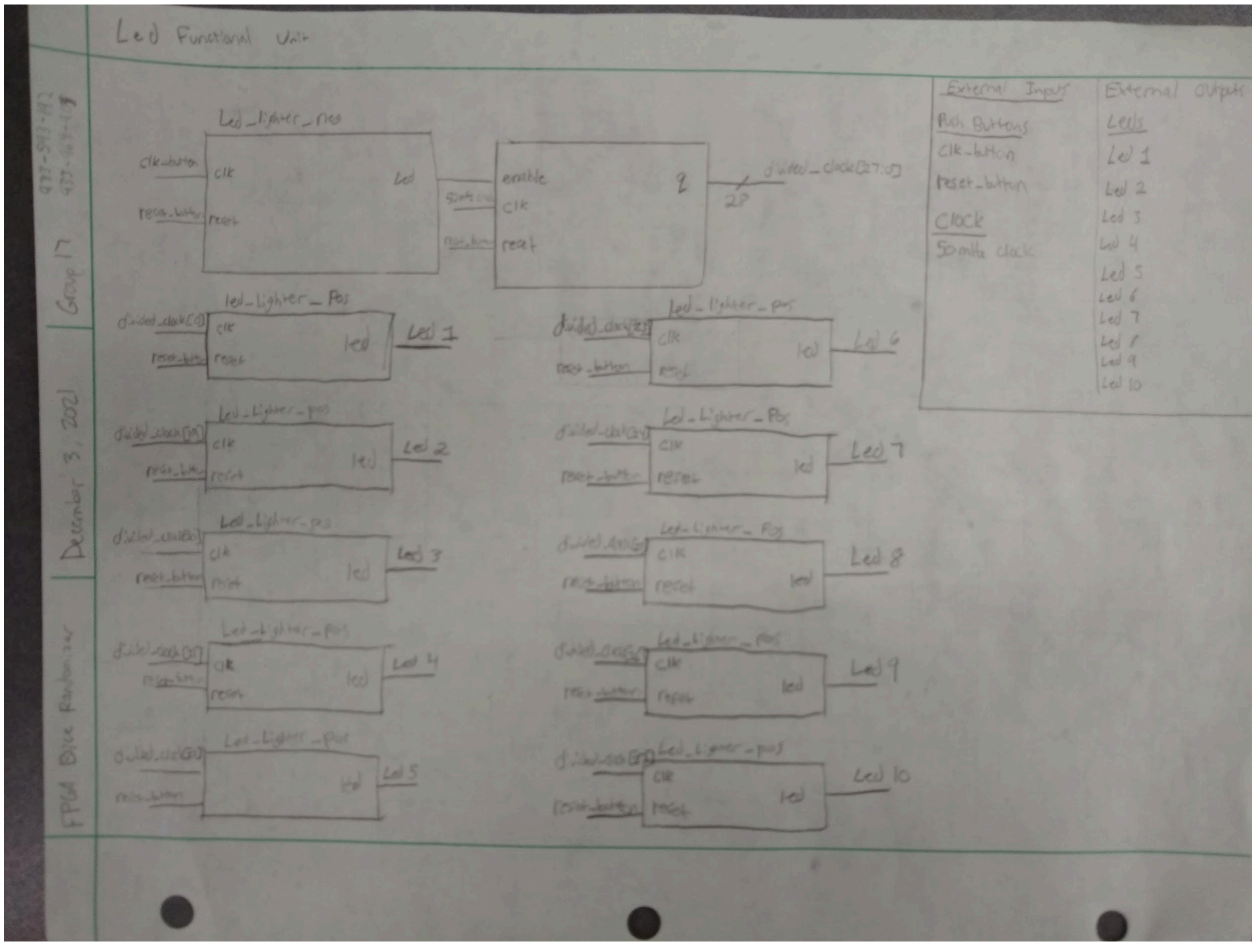


Figure 5: The workings of the clock divider/LED functional unit. Refer to A.8

As seen from figure 5, the functional unit is made up of a clock divider(led_lighter_neg and counter) with 10 different led_lighter_pos. This is because to allow for the lighting up motion of the LEDs they must have increasing significance of the divided clock bit taken in. This means that they take in different inputs and would not be very simple if put into one gate.

Figure 6 shows the simulation of one LED. The LED value becomes true when the reset_button is 1and the clk button is 0 (active low). The LED output actually lights up when there is a rising edge in the bit of divided clock in its input.

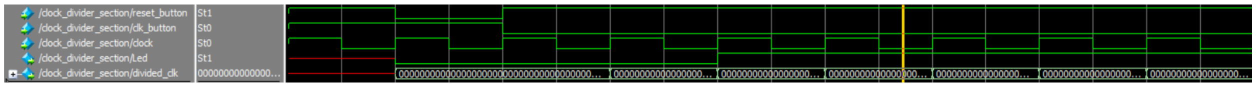

/clock_divider_section/reset_button St1
/clock_divider_section/clk_button St0
/clock_divider_section/clock St0
/clock_divider_section/Led St1
/clock_divider_section/divided_clk 00000000000000...

Figure 6: Simulation for the Clock divider/LED function unit. Refer to B .2

### 2.1.1 Led_lighter_neg

Inputs: clk and reset values

Outputs: Led output value

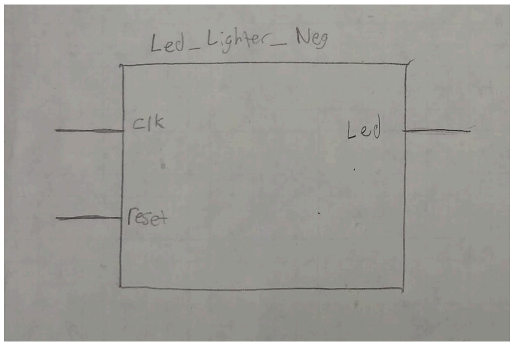


Figure 7: The Led_lighter_neg gate. Refer to A.5

The main purpose of the figure 7 gate is to be able to invert the push buttons which are active low into a true signal when the clk_button is pressed and a false signal when the reset_button is pressed.
As seen from figure 8, the LED value is true when the clk is 0 and the reset is 1.

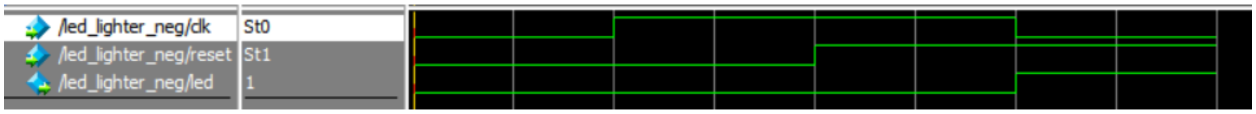


Figure 8: Simulation for the Led_lighter_neg. Refer to B.8

### 2.1.2 Counter

Inputs: clk input, enable value, reset value

Outputs: a 28 bit Q value

Parameter: N = 28

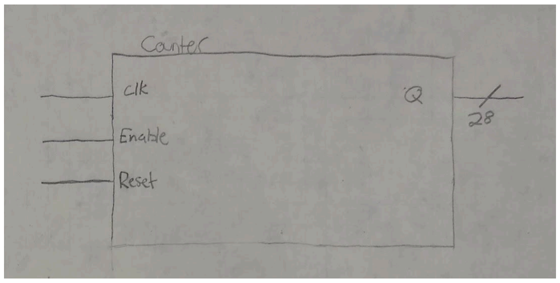


Figure 9: The Counter gate. Refer to A.6

The purpose of the counter is to be able to make enough varying clock signals through the vast number of bits outputted to be utilized in other blocks.
As seen in figure 10 the q output increases in value every rising edge of the clk and when the enable is true and the reset is false.

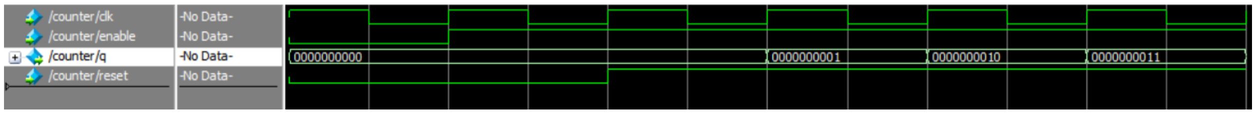


Figure 10: Simulation for the Counter. Refer to B.7

### 2.1.3 Led_lighter_pos

Inputs: clk and reset values

Outputs: Led output value

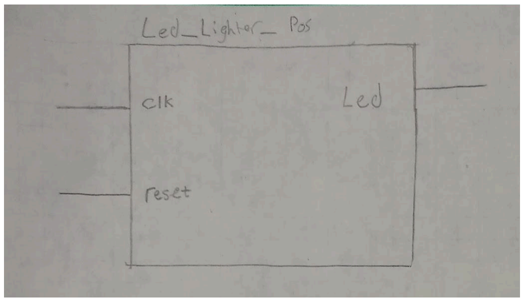


Figure 11: The Led_lighter_pos gate. Refer to A.4

The block in figure 11 is very similar to the 2.1.1 gate. The only difference is as seen in figure 12, the LED output is true only when the clk_button is 1 and reset_button is 1. This means that the reset is active low while the clk is active high. However, the purpose of this gate is to facilitate the lighting of the led.

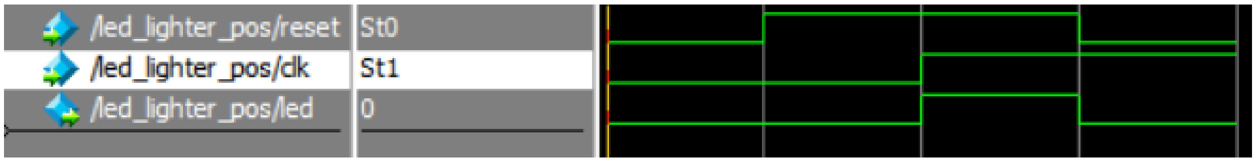


Figure 12: Simulation for the Led_lighter_pos. Refer to B.9

## 2.2 Randomizer sevenseg section

Inputs: clk_button, reset_button

Outputs: the 6 seven seg displays showing the random value produced

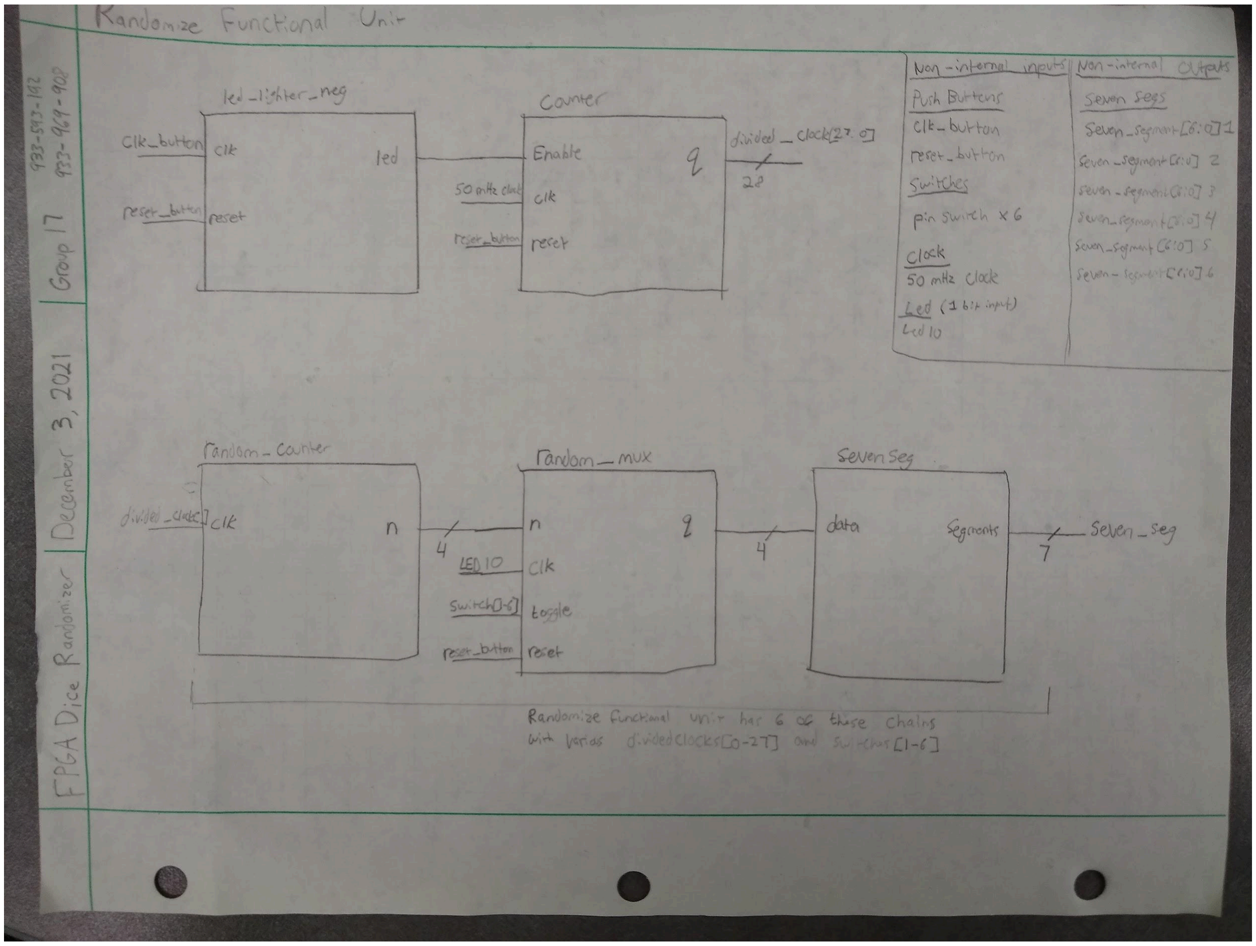


Figure 13: The workings of the randomizer sevenseg functional unit. Refer to A.9

The purpose of the functional unit represented by figure 13 is to make random values between 1 through 6 and display them when requested. As seen in the diagram it has a clock divider that outputs data. The random counter takes one bit from the divided clock and generates a random value that is taken into the random mux. The random mux makes sure the toggle value is true and enabled when the final LED is lit as seen with its clock input. Then this value is passed into the seven seg to be displayed.

Figure 13 and 14 represent the working of 1 unit. There is one clk divider and the bottom part of figure 13 is replicated 6 times to accommodate the 6 7 segment displays.

As seen in figure 14, the seven seg value is generated(changes) when there is a rising(at the rate of the intaken bit) edge, reset is 1(active low) and the clk_button is 1.

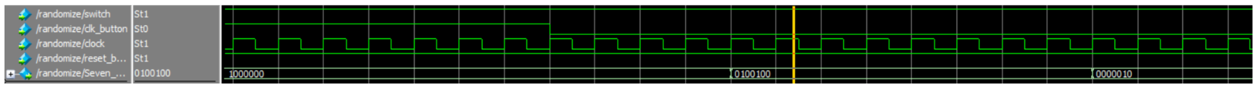


Figure 14: Simulation for the randomizer sevenseg functional unit. Refer to B.3

### 2.2.1 Led_lighter_neg

Please refer to 2.1.1 to learn about the neg_led_lighter

### 2.2.2 Counter

Please refer to 2.1.2 to learn about the counter

### 2.2.3 Sevenseg

Inputs: 4 bit data input binary value

Outputs: 7 bit value that is the representation of the input value in a 7 segment display as a decimal number

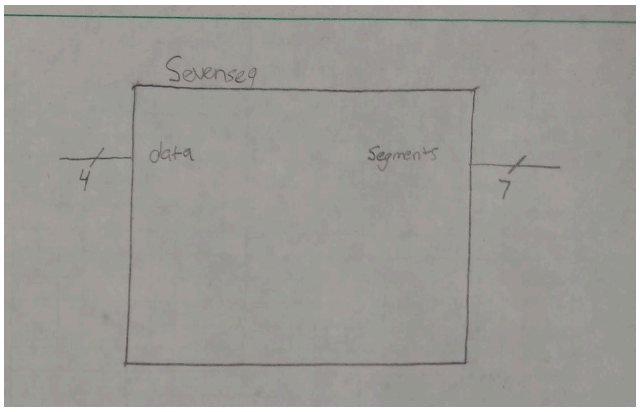


Figure 15: The seven seg display gate. Refer to A.2

The purpose of the seven seg gate in figure 15 is to take in a value of 4 bits and output the 7 bit segment display of the particular binary value in its denary form. As seen from figure 16, the seven segment value is given out for each 4 bit value.

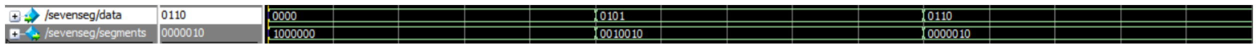


Figure 16: The simulation for the seven segment display. Refer to B.4

## 2.2.4 Random_counter

Inputs: clk input

Outputs: A 4 bit n value between 1 and 6

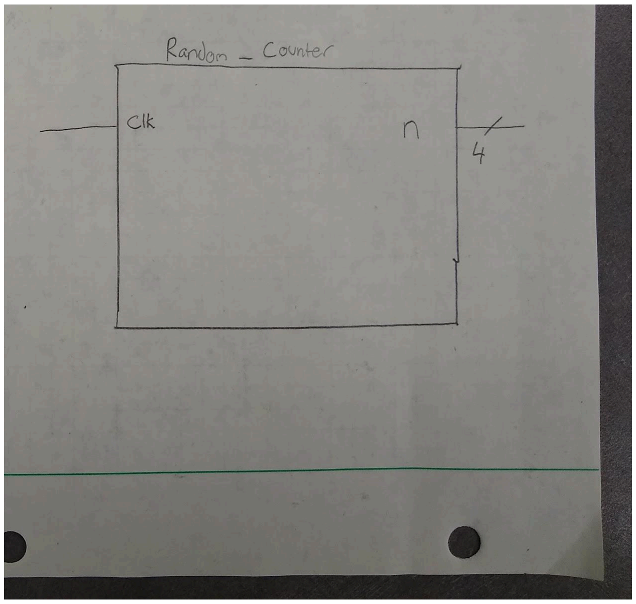


Figure 17: The random_counter gate. Refer to A.3

The random counter gate, in figure 17, is very much a counter. However, there are 6 random_counters to accommodate for the 6 different digits of seven seg. Each of the random_counters takes in a different divided clock bit in a jumbled sequence. This is where the random and unpredictability factor comes through. The random counter only outputs values between 1 and 6 to resemble the dice as much as possible. It counts from 1 to 6 and resets back to 1 after reaching 6.

The evidence of the counting from 1 through 6 and resetting back to 1 is evident in figure 18. As seen the value out the n output changes from 1 through 6 and back to 1. Increasing by 1 at each rising edge of the clock input.

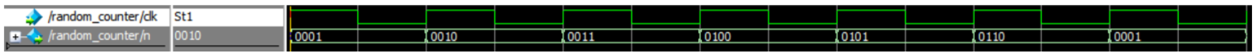


Figure 18: Simulation for the random_counter. Refer to B.6

## 2.2.5 Random_mux

Inputs: clk input, toggle value, 4 bit N value, reset value

Outputs: 4 bit a output value that passes binary value of N

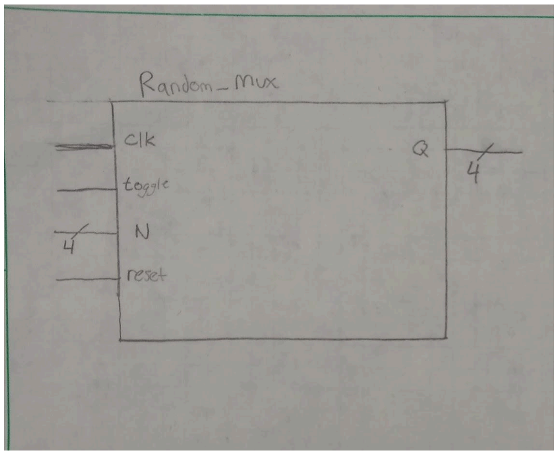


Figure 19: The random_mux gate. Refer to A.7

The purpose of the random_mux gate is to be able to make sure that the particular digit that its assigned to wants to output ratchet rather than the default. This is determined by the toggle. If the toggle is true then the n value is passed through but not if it is false. The toggle is represented by switches that correspond to a particular seven seg digit.

As seen from figure 20, the q output value is equivalent to the N value when the reset is 1(active low) and the toggle is 1(active high). The value changes at the rising edge of the clock.

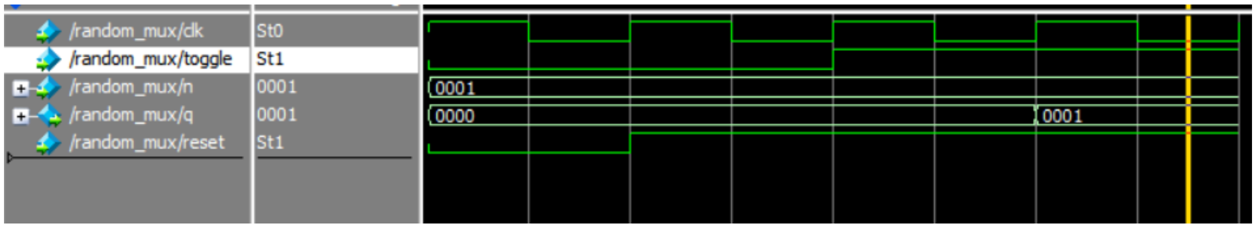
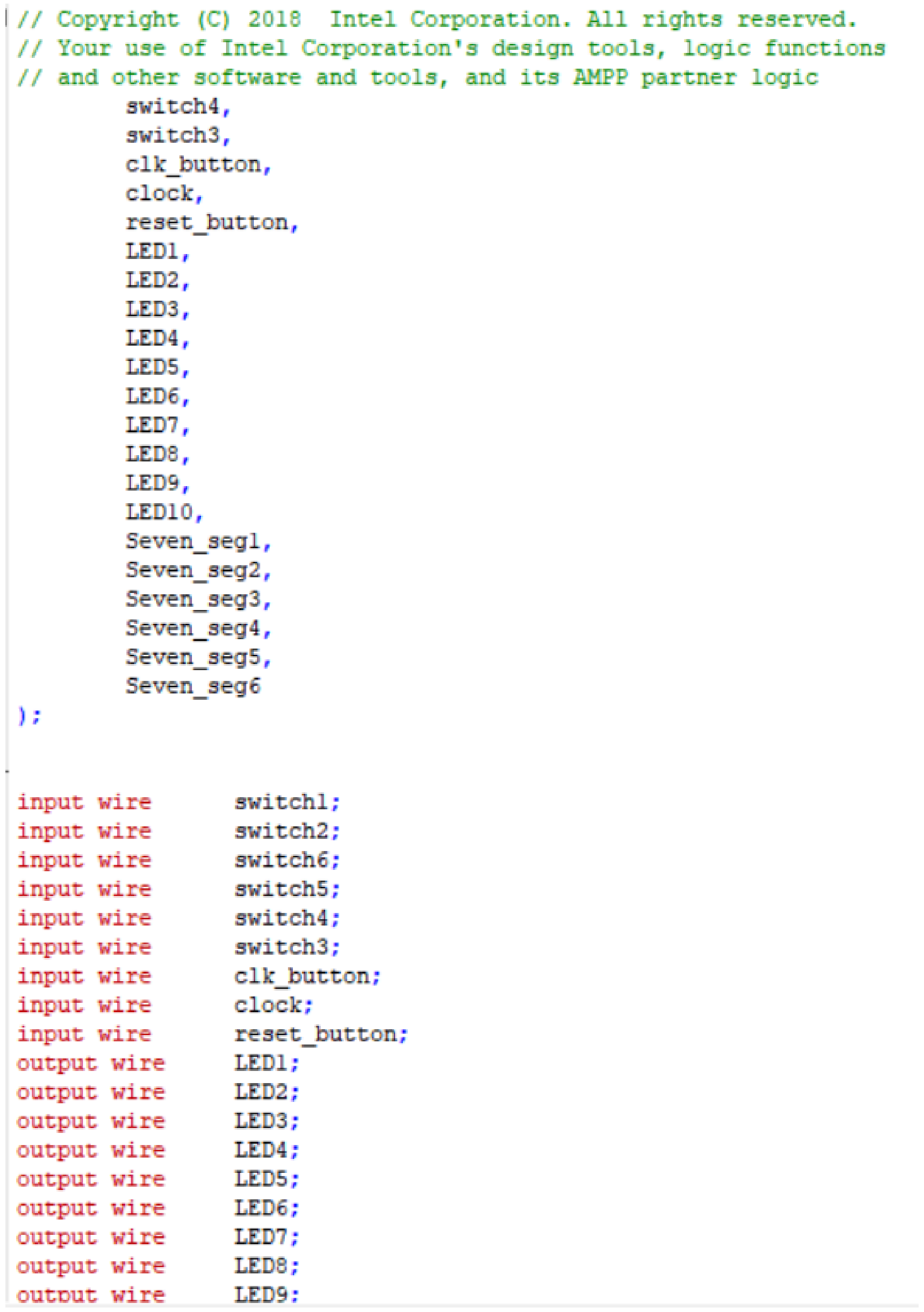


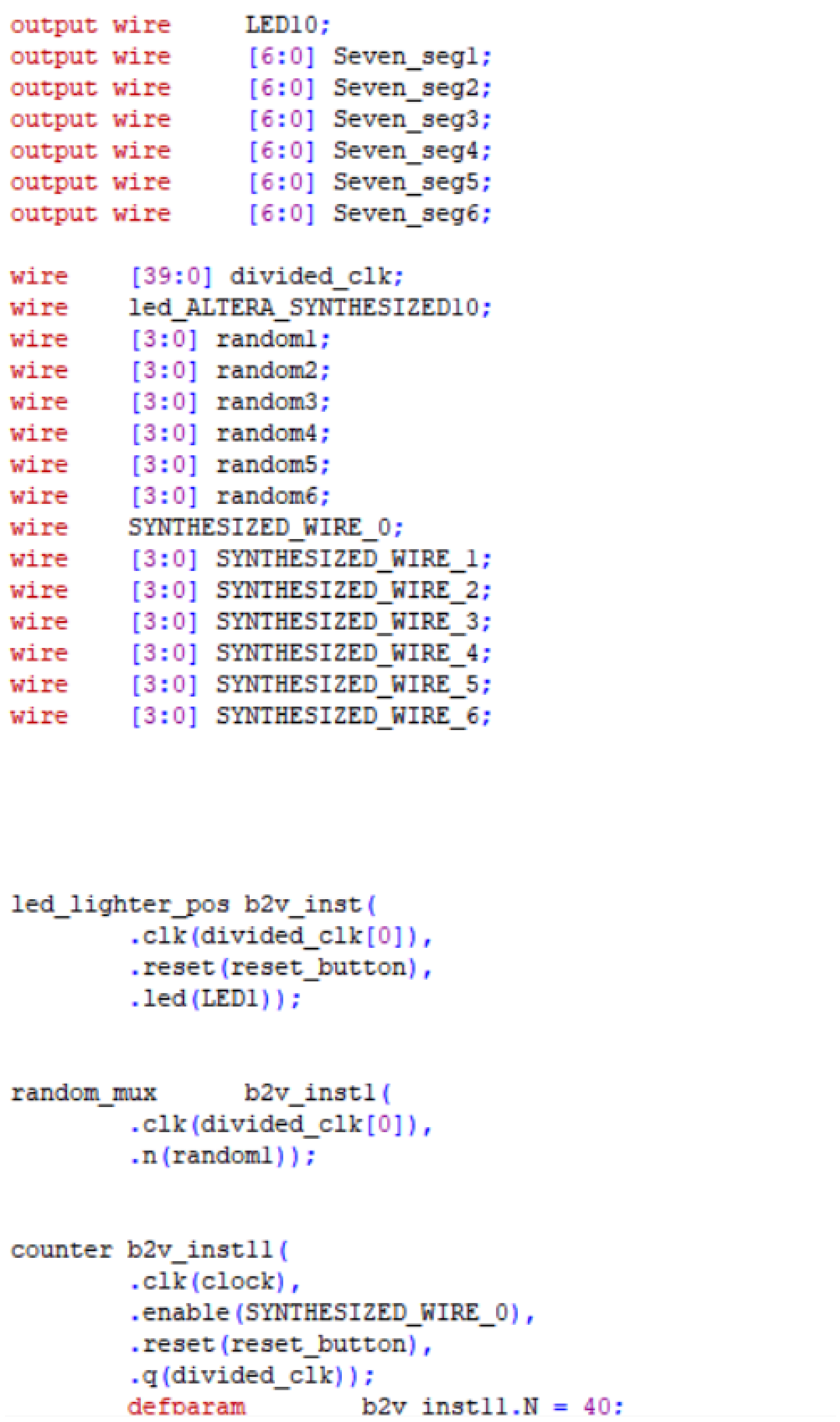
Figure 20: Simulation for the random_mux. Refer to B.5

## A. SystemVerilog Files

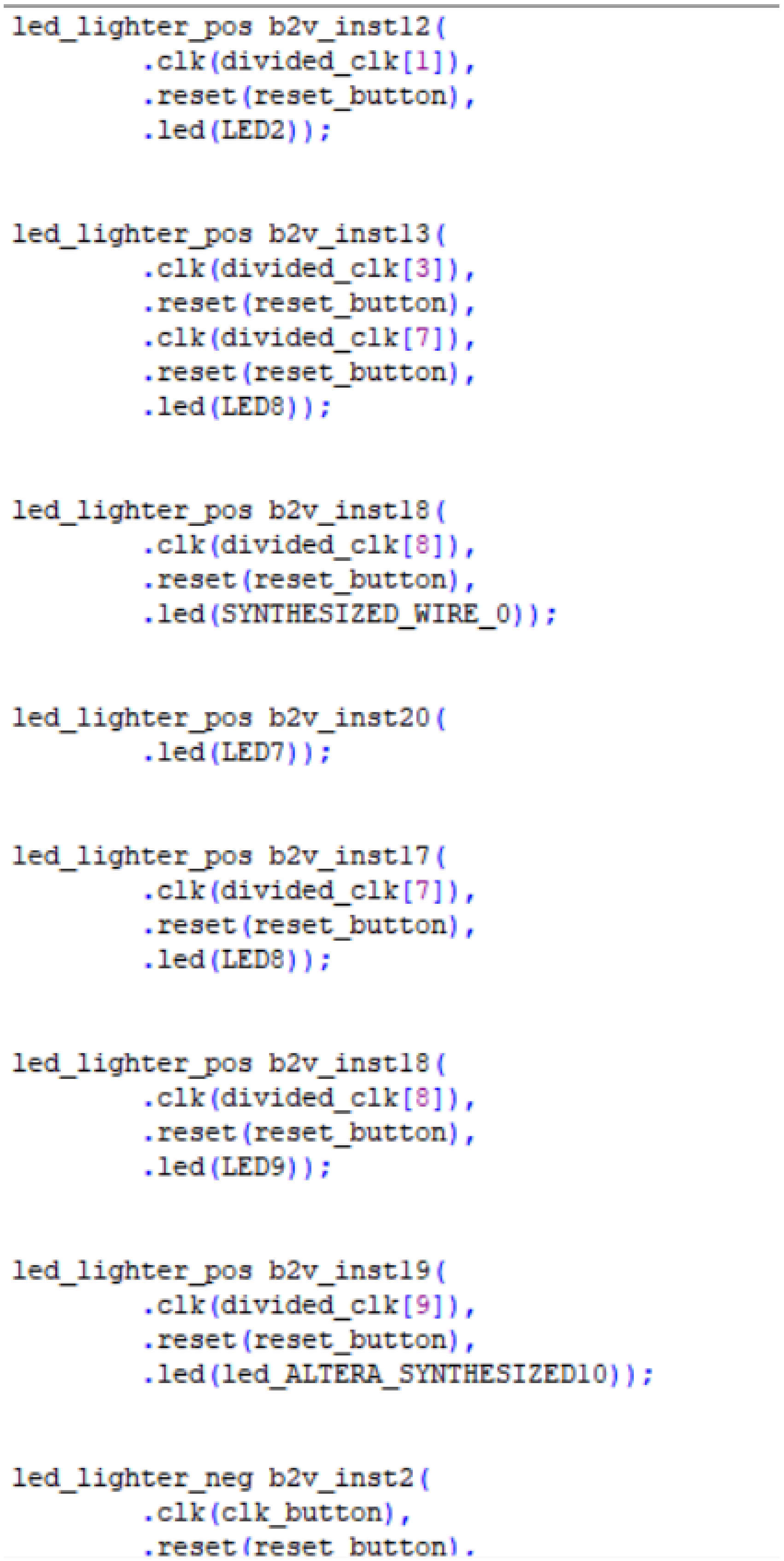
### A.1 Top Level(Verilog HDL)



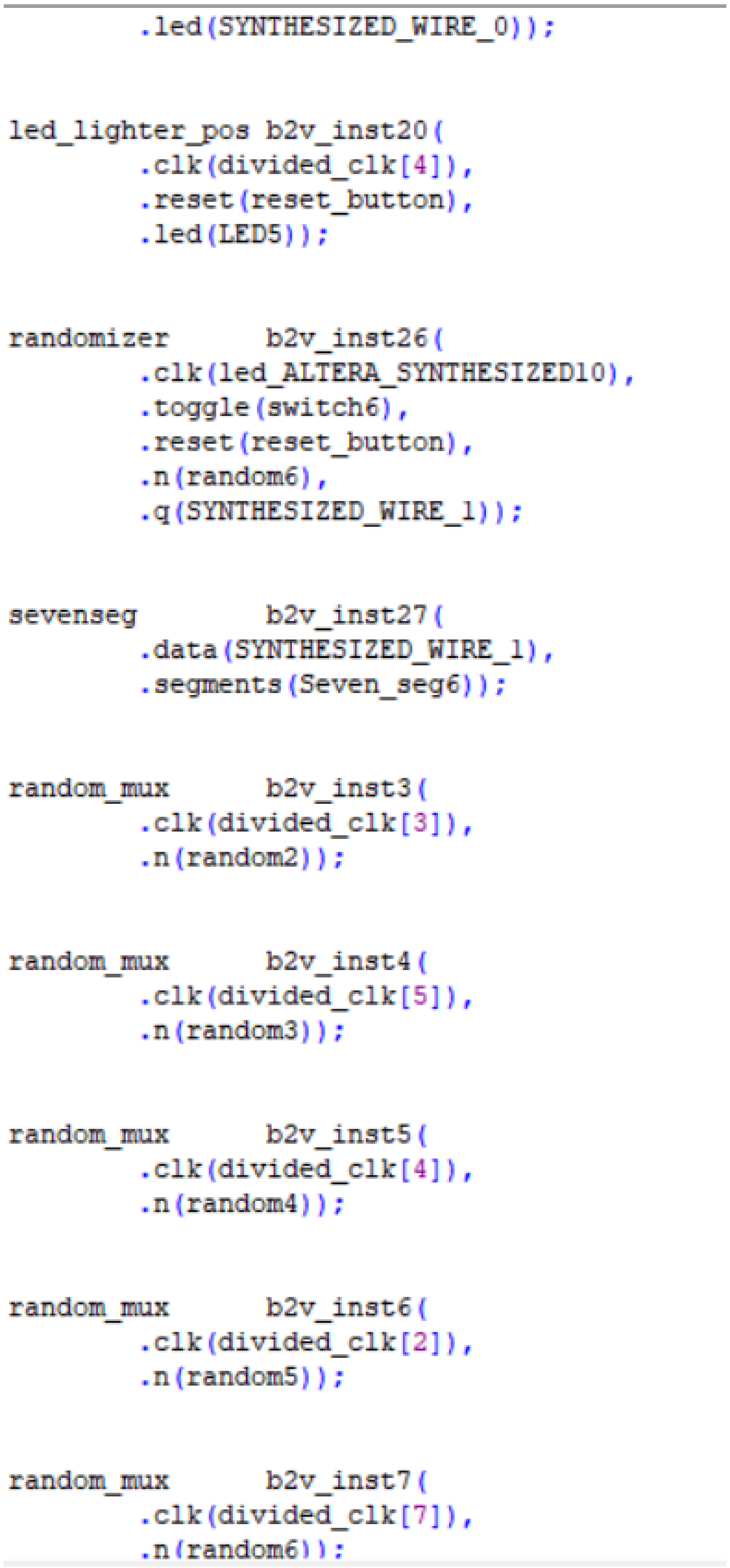
```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
        switch4,
        switch3,
        clk_button,
        clock,
        reset_button,
        LED1,
        LED2,
        LED3,
        LED4,
        LED5,
        LED6,
        LED7,
        LED8,
        LED9,
        LED10,
        Seven_seg1,
        Seven_seg2,
        Seven_seg3,
        Seven_seg4,
        Seven_seg5,
        Seven_seg6
);


input wire      switch1;
input wire      switch2;
input wire      switch6;
input wire      switch5;
input wire      switch4;
input wire      switch3;
input wire      clk_button;
input wire      clock;
input wire      reset_button;
output wire     LED1;
output wire     LED2;
output wire     LED3;
output wire     LED4;
output wire     LED5;
output wire     LED6;
output wire     LED7;
output wire     LED8;
output wire     LED9;
```



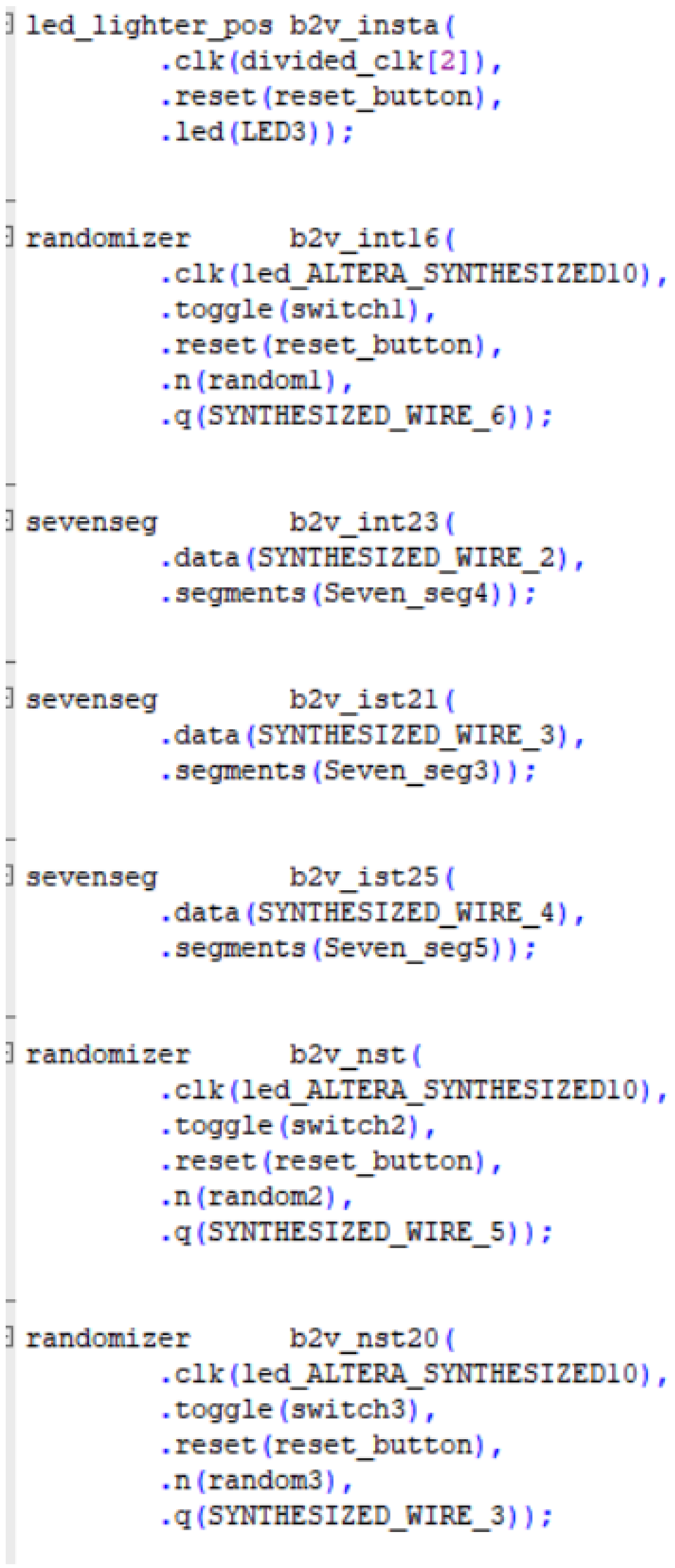
```
output wire     LED10;
output wire     [6:0] Seven_seg1;
output wire     [6:0] Seven_seg2;
output wire     [6:0] Seven_seg3;
output wire     [6:0] Seven_seg4;
output wire     [6:0] Seven_seg5;
output wire     [6:0] Seven_seg6;

wire    [39:0] divided_clk;
wire    led_ALTERA_SYNTHESIZED10;
wire    [3:0] random1;
wire    [3:0] random2;
wire    [3:0] random3;
wire    [3:0] random4;
wire    [3:0] random5;
wire    [3:0] random6;
wire    SYNTHESIZED_WIRE_0;
wire    [3:0] SYNTHESIZED_WIRE_1;
wire    [3:0] SYNTHESIZED_WIRE_2;
wire    [3:0] SYNTHESIZED_WIRE_3;
wire    [3:0] SYNTHESIZED_WIRE_4;
wire    [3:0] SYNTHESIZED_WIRE_5;
wire    [3:0] SYNTHESIZED_WIRE_6;




led_lighter_pos b2v_inst(
        .clk(divided_clk[0]),
        .reset(reset_button),
        .led(LED1));


random_mux      b2v_inst1(
        .clk(divided_clk[0]),
        .n(random1));


counter b2v_inst11(
        .clk(clock),
        .enable(SYNTHESIZED_WIRE_0),
        .reset(reset_button),
        .q(divided_clk));
        defparam        b2v_inst11.N = 40;
```



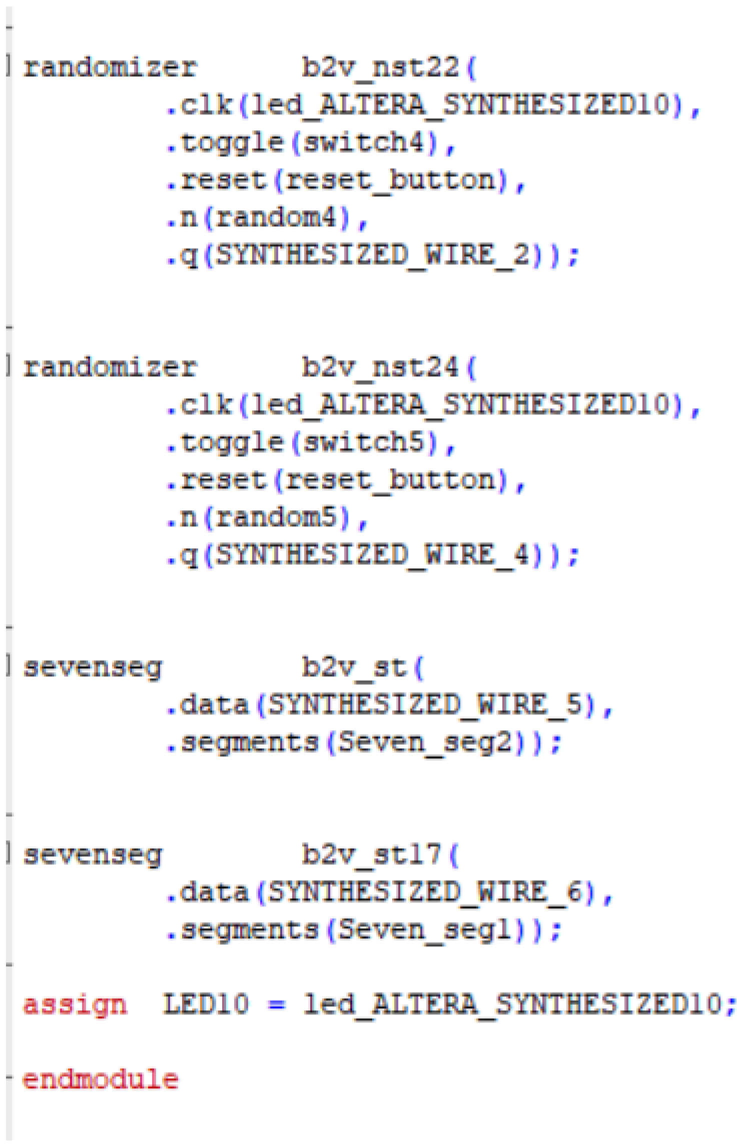
```
led_lighter_pos b2v_inst12(
        .clk(divided_clk[1]),
        .reset(reset_button),
        .led(LED2));


led_lighter_pos b2v_inst13(
        .clk(divided_clk[3]),
        .reset(reset_button),
        .clk(divided_clk[7]),
        .reset(reset_button),
        .led(LED8));


led_lighter_pos b2v_inst18(
        .clk(divided_clk[8]),
        .reset(reset_button),
        .led(SYNTHESIZED_WIRE_0));


led_lighter_pos b2v_inst20(
        .led(LED7));


led_lighter_pos b2v_inst17(
        .clk(divided_clk[7]),
        .reset(reset_button),
        .led(LED8));


led_lighter_pos b2v_inst18(
        .clk(divided_clk[8]),
        .reset(reset_button),
        .led(LED9));


led_lighter_pos b2v_inst19(
        .clk(divided_clk[9]),
        .reset(reset_button),
        .led(led_ALTERA_SYNTHESIZED10));


led_lighter_neg b2v_inst2(
        .clk(clk_button),
        .reset(reset_button),
```



```
        .led(SYNTHESIZED_WIRE_0));


led_lighter_pos b2v_inst20(
        .clk(divided_clk[4]),
        .reset(reset_button),
        .led(LED5));


randomizer      b2v_inst26(
        .clk(led_ALTERA_SYNTHESIZED10),
        .toggle(switch6),
        .reset(reset_button),
        .n(random6),
        .q(SYNTHESIZED_WIRE_1));


sevenseg        b2v_inst27(
        .data(SYNTHESIZED_WIRE_1),
        .segments(Seven_seg6));


random_mux      b2v_inst3(
        .clk(divided_clk[3]),
        .n(random2));


random_mux      b2v_inst4(
        .clk(divided_clk[5]),
        .n(random3));


random_mux      b2v_inst5(
        .clk(divided_clk[4]),
        .n(random4));


random_mux      b2v_inst6(
        .clk(divided_clk[2]),
        .n(random5));


random_mux      b2v_inst7(
        .clk(divided_clk[7]),
        .n(random6));
```



```
led_lighter_pos b2v_insta(
        .clk(divided_clk[2]),
        .reset(reset_button),
        .led(LED3));


randomizer      b2v_int16(
        .clk(led_ALTERA_SYNTHESIZED10),
        .toggle(switch1),
        .reset(reset_button),
        .n(random1),
        .q(SYNTHESIZED_WIRE_6));


sevenseg        b2v_int23(
        .data(SYNTHESIZED_WIRE_2),
        .segments(Seven_seg4));


sevenseg        b2v_ist21(
        .data(SYNTHESIZED_WIRE_3),
        .segments(Seven_seg3));


sevenseg        b2v_ist25(
        .data(SYNTHESIZED_WIRE_4),
        .segments(Seven_seg5));


randomizer      b2v_nst(
        .clk(led_ALTERA_SYNTHESIZED10),
        .toggle(switch2),
        .reset(reset_button),
        .n(random2),
        .q(SYNTHESIZED_WIRE_5));


randomizer      b2v_nst20(
        .clk(led_ALTERA_SYNTHESIZED10),
        .toggle(switch3),
        .reset(reset_button),
        .n(random3),
        .q(SYNTHESIZED_WIRE_3));
```



```
randomizer      b2v_nst22(
         .clk(led_ALTERA_SYNTHESIZED10),
         .toggle(switch4),
         .reset(reset_button),
         .n(random4),
         .q(SYNTHESIZED_WIRE_2));


randomizer      b2v_nst24(
         .clk(led_ALTERA_SYNTHESIZED10),
         .toggle(switch5),
         .reset(reset_button),
         .n(random5),
         .q(SYNTHESIZED_WIRE_4));


sevenseg        b2v_st(
         .data(SYNTHESIZED_WIRE_5),
         .segments(Seven_seg2));


sevenseg        b2v_st17(
         .data(SYNTHESIZED_WIRE_6),
         .segments(Seven_seg1));

assign  LED10 = led_ALTERA_SYNTHESIZED10;

endmodule
```

## A.2 Sevenseg

```
module sevenseg(input logic[3:0]data,
                output logic[6:0]segments);
   always_comb
      case(data)
      0:segments=7'b100_0000;
      1:segments=7'b111_1001;
      2:segments=7'b010_0100;
      3:segments=7'b011_0000;
      4:segments=7'b001_1001;
      5:segments=7'b001_0010;
      6:segments=7'b000_0010;
      7:segments=7'b111_1000;
      8:segments=7'b000_0000;
      9:segments=7'b001_1000;
      default: segments=7'b000_0000;
      endcase
endmodule
```

## A.3 Random_counter

```
module random_counter(input logic clk,
                     |   output logic[3:0] n);


   always_ff@ (posedge clk)
      begin
      if (n < 6) n <= n+1;
      else n = 1;
      end
endmodule
```

## A.4 Led_lighter_pos

```
module led_lighter_pos(input logic clk,
                       input logic reset,
                       output logic led);
   always_ff@ (posedge clk, negedge reset)
      begin
      if (!reset) led <= 0;
      else led <= 1;
      end
endmodule
```

## A.5 Led_lighter_neg

```
module led_lighter_neg(input logic clk,
                       input logic reset,
                       output logic led);
   always_ff@ (negedge clk, negedge reset)
      begin
      if (!reset) led <= 0;
      else led <= 1;
      end
endmodule
```

## A.6 Counter

```
module counter #(parameter N=10)
                (input logic clk,
                input logic enable,
                input logic reset,
                output logic [N-1:0]q);
   always_ff@(posedge clk, negedge reset)
      if (!reset) q <= 0;
      else if (enable)        q <= q + 1;
endmodule
```

## A.7 Random_mux

```
module random_mux(input logic clk,
                  input logic toggle,
                  input logic[3:0] n,
                  input logic reset,
                  output logic[3:0] q);


   always_ff@ (posedge clk,negedge reset)
      begin
      if (!reset) q<= 0;
      else if (toggle) q <= n;
      else q <= 0;
      end
endmodule
```

## A.8 Clock divider section

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// PROGRAM              "Quartus Prime"
// VERSION              "Version 18.0.0 Build 614 04/24/2018 SJ Lite Edition"
// CREATED              "Fri Dec 03 01:33:16 2021"

module clock_divider_section(
        clk_button,
        clock,
        reset_button,
        Led,
        divided_clk
);


input wire      clk_button;
input wire      clock;
input wire      reset_button;
output wire     Led;
output wire     [39:0] divided_clk;

wire    [39:0] divided_clk_ALTERA_SYNTHESIZED;
wire    SYNTHESIZED_WIRE_0;
```

```
led_lighter_pos b2v_inst(
        .clk(divided_clk_ALTERA_SYNTHESIZED[1]),
        .reset(reset_button),
        .led(Led));


counter b2v_inst11(
        .clk(clock),
        .enable(SYNTHESIZED_WIRE_0),
        .reset(reset_button),
        .q(divided_clk_ALTERA_SYNTHESIZED));
        defparam        b2v_inst11.N = 40;


led_lighter_neg b2v_inst2(
        .clk(clk_button),
        .reset(reset_button),
        .led(SYNTHESIZED_WIRE_0));

assign  divided_clk = divided_clk_ALTERA_SYNTHESIZED;

endmodule
```

## A.9 Randomizer sevenseg

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// PROGRAM		"Quartus Prime"
// VERSION		"Version 18.0.0 Build 614 04/24/2018 SJ Lite Edition"
// CREATED		"Fri Dec 03 02:17:05 2021"

module randomize(
	switch,
	clk_button,
	clock,
	reset_button,
	Seven_seg1
);


input wire	switch;
input wire	clk_button;
input wire	clock;
input wire	reset_button;
output wire	[6:0] Seven_seg1;

wire	[39:0] divided_clk;
wire	[3:0] random1;
wire	SYNTHESIZED_WIRE_0;
wire	[3:0] SYNTHESIZED_WIRE_1;
```

```
random_mux      b2v_inst(
        .clk(divided_clk[0]),
        .n(random1));


counter b2v_inst11(
        .clk(clock),
        .enable(SYNTHESIZED_WIRE_0),
        .reset(reset_button),
        .q(divided_clk));
        defparam        b2v_inst11.N = 40;


led_lighter_neg b2v_inst2(
        .clk(clk_button),
        .reset(reset_button),
        .led(SYNTHESIZED_WIRE_0));


randomizer      b2v_int16(
        .clk(divided_clk[2]),
        .toggle(switch),
        .reset(reset_button),
        .n(random1),
        .q(SYNTHESIZED_WIRE_1));


sevenseg        b2v_st17(
        .data(SYNTHESIZED_WIRE_1),
        .segments(Seven_seg1));


endmodule
```

## B Simulation Files

### B.1 Top Level

```
vsim -gui work.Group272
add wave -position insertpoint  \
sim:/Group272/switch1 \
sim:/Group272/switch2 \
sim:/Group272/switch6 \
sim:/Group272/switch5 \
sim:/Group272/switch4 \
sim:/Group272/switch3 \
sim:/Group272/clk_button \
sim:/Group272/clock \
sim:/Group272/reset_button \
sim:/Group272/LED1 \
sim:/Group272/LED2 \
sim:/Group272/LED3 \
sim:/Group272/LED4 \
sim:/Group272/LED5 \
sim:/Group272/LED6 \
sim:/Group272/LED7 \
sim:/Group272/LED8 \
sim:/Group272/LED9 \
sim:/Group272/LED10 \
sim:/Group272/Seven_seg1 \
sim:/Group272/Seven_seg2 \
sim:/Group272/Seven_seg3 \
sim:/Group272/Seven_seg4 \
sim:/Group272/Seven_seg5 \
sim:/Group272/Seven_seg6
force -freeze sim:/Group272/switch1 1 0
force -freeze sim:/Group272/switch2 1 0
force -freeze sim:/Group272/switch6 1 0
force -freeze sim:/Group272/switch5 1 0
force -freeze sim:/Group272/switch4 1 0
force -freeze sim:/Group272/switch3 1 0
force -freeze sim:/Group272/clk_button 1 0
force -freeze sim:/Group272/reset_button 1 0
force -freeze sim:/Group272/clock 1 0, 0 {50 ps} -r 100
run 100 ps
force reset_button 0
run 100 ps
force reset_button 1
force clk_button 0
run 80000 ps
```

## B.2 Clock divider/Led function unit

```
vsim -gui work.clock_divider_section
add wave divided_clk
add wave Led
add wave reset_button
add wave clk_button
add wave clock
force -freeze sim:/clock_divider_section/clock 1 0, 0 {50 ps} -r 100
force reset_button 1
force clk_button 1
run 100 ps
force reset_button 0
run 100 ps
force reset_button 1
force clk_button 0
run 100 ps
run 1000 ps
```

## B.3 Randomizer sevenseg

```
vsim -gui work.randomize
add wave switch
add wave clk_button
add wave reset_button
add wave Seven_seg1
add wave clock
force -freeze sim:/randomize/clock 1 0, 0 {50 ps} -r 100
force clk_button 1
force reset_button 0
force switch 1
run 1000 ps
force reset_button 1
run 1000 ps
force clk_button 0
run 5000 ps
```

## B.4 Sevenseg

```
vsim -gui work.sevenseg
add wave segments
force data 0000
run 100 ps
force data 0101
run 100 ps
force data 0110
run 100 ps
```

## B.5 Random_mux

```
vsim -gui work.random_mux
add wave clk
add wave toggle
add wave n
add wave q
add wave reset
force -freeze sim:/random_mux/clk 1 0, 0 {50 ps} -r 100
force reset 0
force toggle 0
force n 0001
run 100 ps
force reset 1
run 100 ps
force toggle 1
run 200 ps
```

## B.6 Random_counter

```
vsim -gui work.random_counter
add wave clk
add wave n
force -freeze sim:/random_counter/clk 1 0, 0 {50 ps} -r 100
run 700 ps
```

## B.7 Counter

```
vsim -gui work.counter
add wave clk
add wave enable
add wave q
add wave reset
force -freeze sim:/counter/clk 1 0, 0 {50 ps} -r 100
force enable 0
force reset 0
run 100 ps
force enable 1
run 100 ps
force reset 1
run 100 ps
run 300 ps
```

## B.8 Led_lighter_neg

```
vsim -gui work.led_lighter_neg
add wave clk
add wave reset
add wave led
force reset 0
force clk 0
run 100 ps
force clk 1
run 100 ps
force reset 1
run 100 ps
force clk 0
run 100 ps
```

## B.9 Led_lighter_pos

```
vsim -gui work.led_lighter_pos
add wave reset
add wave led
add wave clk
force clk 0
force reset 0
run 100 ps
force reset 1
run 100 ps
force clk 1
run 100 ps
force reset 0
run 100 ps
```