2018

ENGN6213/3213

Assignment 1 Report

Australian National University

CHAOYUN GONG U6329142

KENNETH GODWIN ZHANG U5624541

COLLEGE OF ENGINEERING AND COMPUTER SCIENCE

Contents

Part One: Fundamentals

Question 1

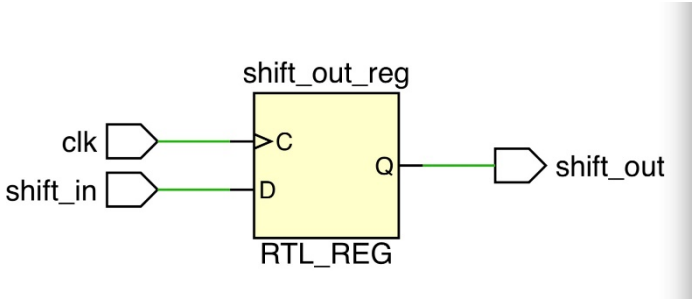
Blocking statements use "=", the following statements will be executed until this statement has completed. It is sequential. Non-blocking statements use "<=", all the non-blocking statements are executed parallel.

Figure 1 is gate-level schematic of blocking statements. In one always () statement, the reg a, b, c and output shift_out are all integrated in one D flip-flop. Because all the sentences are executed one by one in one posedge clk. So shift_out will get the value of shift_in after one posedge clk.

Figure 2 is gate-level schematic of non-blocking statements. Each reg has a D flip-flop. In one posedge clk, all the D flip-flop will active together. So after one posedge clk, all the number will be shifted to the output by one bit. and it costs three clock for shift_out to get the value of shift_in firstly.

The code of blocking statements:

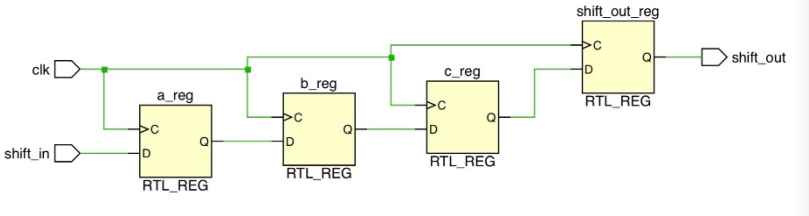
```
always @(posedge clk) begin
        a = shift_in;
        b = a;
        c = b;
        shift_out = c;
    end
```



(Figure 1)

The code of blocking statements:

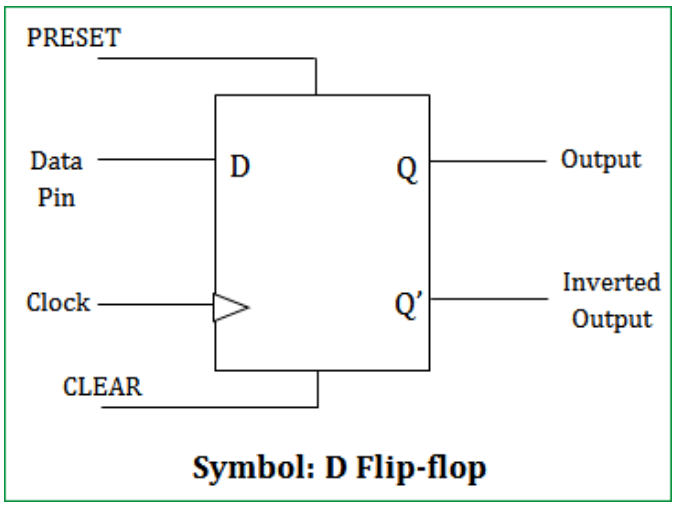
```
always @(posedge clk) begin
        a <= shift_in;
        b <= a;
        c <= b;
        shift_out <= c;
end
```



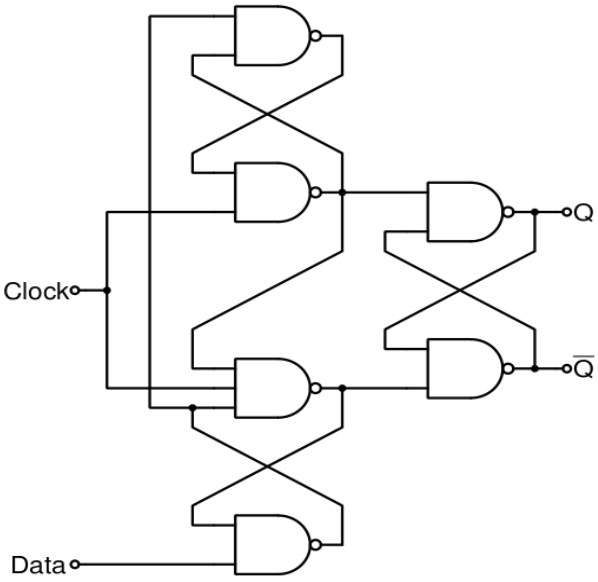
(Figure 2)

Question 2

D-type flip-flop has two stable states and can store state information. As showed in Figure 3, the data comes from Pin D, the values output from Pin Q and Q'.



(Figure 3)



(Figure 4)

Using two D latch can constitute a simple edge-triggered D flip flop. In Figure 4. It shows the classic gate-level of a D flip flop. It includes six NAND-gates. We divide it to two parts (left and right). Left is the inputs D, CLK and the left four NAND-gates. The truth table is below:

D	CLK	Out1	Out2
0	0	1	1
0	Rising edge	1	0
1	0	1	1
1	Rising edge	0	1

Right part is a RS latch. The input S'and R' are Out1 and Out2 of left part. The equation is $Q^{n+1} = S' + RQ^n$. The truth table is below:

S'（Out1）	R'（Out2）	Q	Q^{n+1}
1	1	0/1	hold
0	1	0/1	1
1	0	0/1	0
0	0	0/1	unstable

Because in high-level of CLK, the input will be blocked in circuit. When CLK=1, the change of input will not affect output. The total truth table is below:

D	CLK	Q	Q^{n+1}
0	0/1	0/1	hold
0	Rising edge	0/1	0
1	0/1	0/1	Hold
1	Rising edge	0/1	1

We can find that when CLK = 0 or 1, d flip-flop can store state. In the rising edge, the next output will be changed by input

D. In sequential circuit. The next state depends on the previous state and input. D flip flop has memory to store the state information. So it is very suitable for sequential logic designs. And the stable of D flip flop is stable.

Question 3

Answer 1

The minimum size of the resultant signal means the maximum size of the calculation result. For complement code, the scope of a signed 18-bit is from -2^{17} to $(2^{17}-1)$. The resultant signal equals $2^{17} * 2^{17}$ and add one signed bit. So the minimum size of resultant signal is 35-bit.

Answer 2

The scope of a signed 3-bit is from -4 to 3. The maximum result is 3 – (-4) =7. It needs 3 bits, and one bit for sign. Thus, the minimum size of resultant signal is 4-bit.

Answer 3

$2^{14}<17000<2^{15}$. Thus, the minimum size of resultant signal is 15-bit.

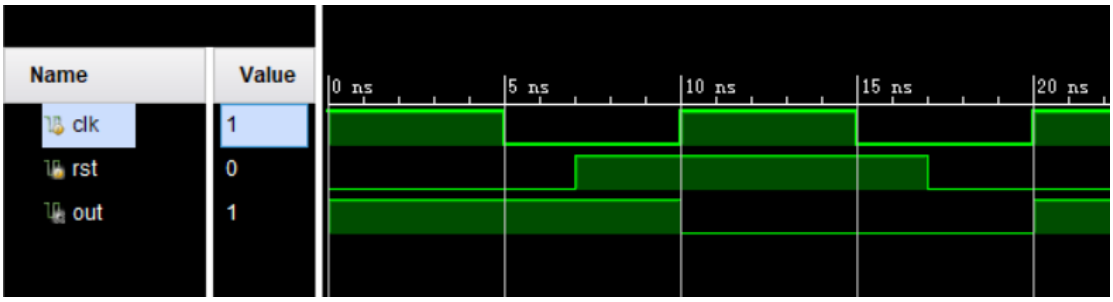
Answer 4

The maximum 10-bit has 2^{10}numbers. $\frac{2^{10}}{2^5} = 2^5$. Thus, the minimum size of resultant signal is 5-bit.

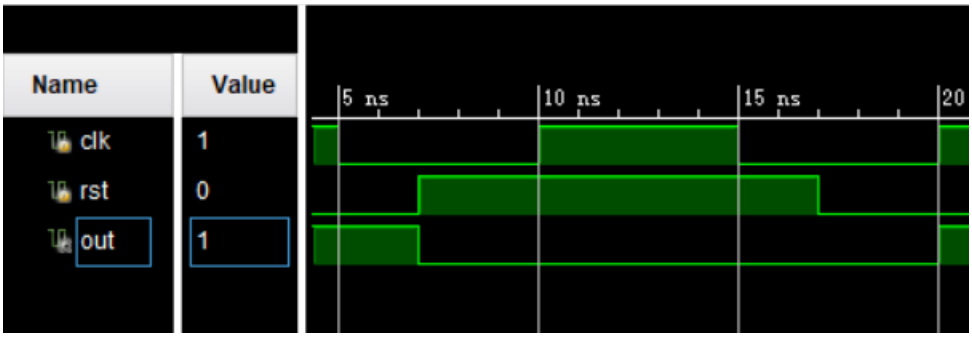
Question 4

Synchronous reset: the reset signal can be valid on the rising edge of clock only.

Asynchronous reset: the system will be reset when reset signal is valid, whatever the clock state.



(Figure 5)



(Figure 6)

Figure 5 is the simulation wave for synchronous reset. When reset becomes valid, the output has not changed until the next rising edge of clock. The synchronous reset code and simulation code are shown below.

Figure 6 is the simulation wave for asynchronous reset. When reset becomes valid, the output is reset at the same time. When using an asynchronous reset that is driven by a physical push button, the signal which is from physical push button will always has bounce, and asynchronous reset will reset system at the same time, it will cause that the system has been reset several times. The asynchronous reset code and simulation code are shown below.

Using an asynchronous reset is more convenient and can save resource. most devices' libraries have port for asynchronous reset. And it easy to be recognized and Using global reset port GSR of FPGA.

Synchronous reset code:

```
always @ (posedge clk) begin
      if(rst) begin
          out = 0;
      end else begin
          out = 1;
      end
end
```

Asynchronous reset code:

```
always @(posedge clk or posedge rst) begin
      if( rst) begin
          out = 0;
      end else begin
          out = 1;
      end
   end
```

The simulation code:

```
always begin
   #5
   clk = ~clk;
end
initial begin
   clk =1 ;
   rst =0;
   #7
   rst =1;
   #10
   rst =0;
end
```

Question 5

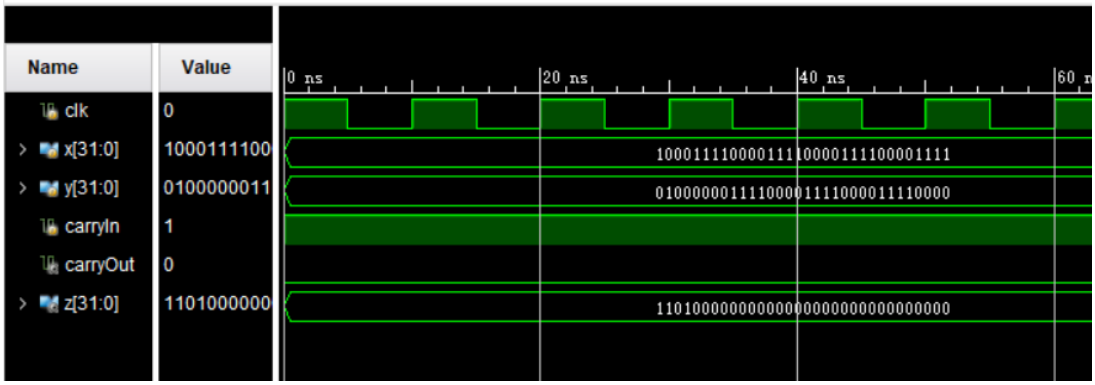
Combinatorial and Sequential logic circuit

In combinatorial logic circuit, the outputs only depend on the inputs anytime. There are no memory elements and feedback lines in circuit.
In sequential logic circuit, outputs depend on the inputs and previous state of the circuit. The circuit has memory elements or feedback lines.
Combinatorial logic circuit: a, d, e.
Sequential logic circuit: b, c.

a. 32 Bit adder

```
module threetwo_bit_adder(
input wire clk,
input wire [31:0] x,
input wire [31:0] y,
input wire carryIn,
output reg carryOut,
output reg [31:0] z
    );
reg [30:0]tem_carry;
reg [31:0] v;
integer i;
always @(*) begin
   for (i=0; i<=31; i=i+1) begin
      if (i==0) begin
         v[0] = x[0]^y[0];
         z[0] = carryIn^v[0];
         tem_carry[0] =(x[0] & y[0]) | (carryIn & v[0]);
       end else if (i==31) begin
          v[31] = x[31]^y[31];
          z[31] = tem_carry[30]^v[31];
          carryOut =(x[31] & y[31]) | (tem_carry[30] & v[31]);
       end else begin
       v[i] = x[i]^y[i];
       z[i] = tem_carry[i-1]^v[i];
       tem_carry[i] =(x[i] & y[i]) | (tem_carry[i-1] & v[i]);
```

```
        end
    end
end
endmodule
```

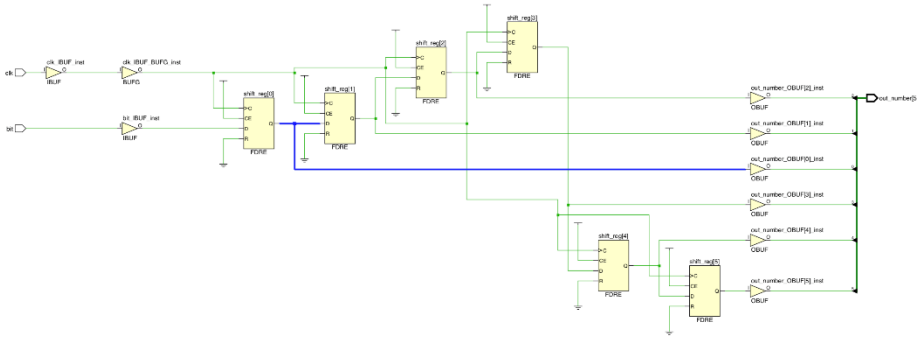


(Figure 7)

The simulation wave shows that the output result just depends on the two 32-bit adders and a 1-bit carryIn in anytime. So, it is combinatorial logic circuit.

b. 6-element shift-register

```
always @(posedge clk) begin
        shift[0] <= bit;
          shift[1] <= shift[0];
        shift[2] <= shift[1];
        shift[3] <= shift[2];
        shift[4] <= shift[3];
        shift[5] <= shift[4];
end
assign out_number = shift;
```

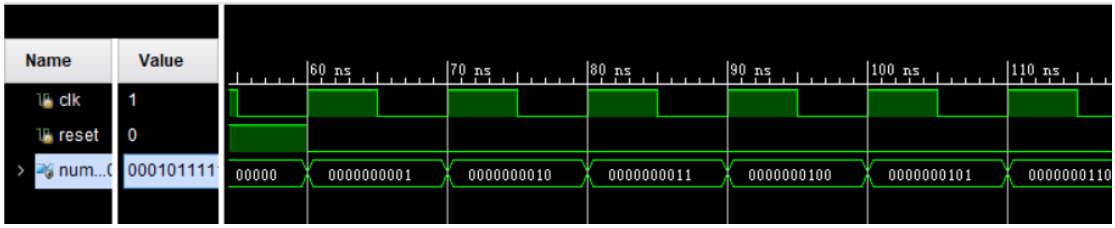


(Figure 8)

Figure 8 shows the hardware of the 6-element shift register. The circuit includes six D flip flop, and the output of D Flip flop need to rely on the previous state. Thus, 6-element shift-register is sequential logic circuit.

c. 10-bit overflow counter

```
always @(posedge clk) begin
    if (reset==1) begin
        counter <= 0;
    end else begin
        counter <= counter + 1'b1;
    end
end
assign number = counter;
```

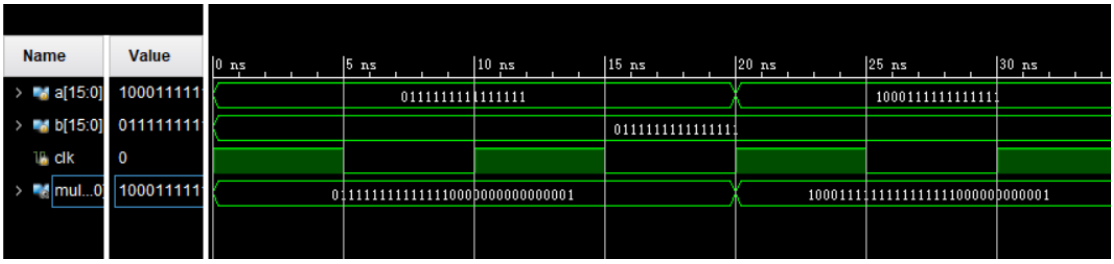


(Figure 9)

Figure 9 shows the simulation wave of 10-bit overflow counter. Each time, output changes its state depended on the input clk and the previous state. Thus, 10-bit overflow counter is sequential logic circuit.

d. 16-bit signed multiplier

```
module sixteen_signed_multiplier(
    input wire signed [15:0] a,b,
    input wire clk,
    output wire signed [30:0] mul
    );
    assign mul = a*b;
endmodule
```

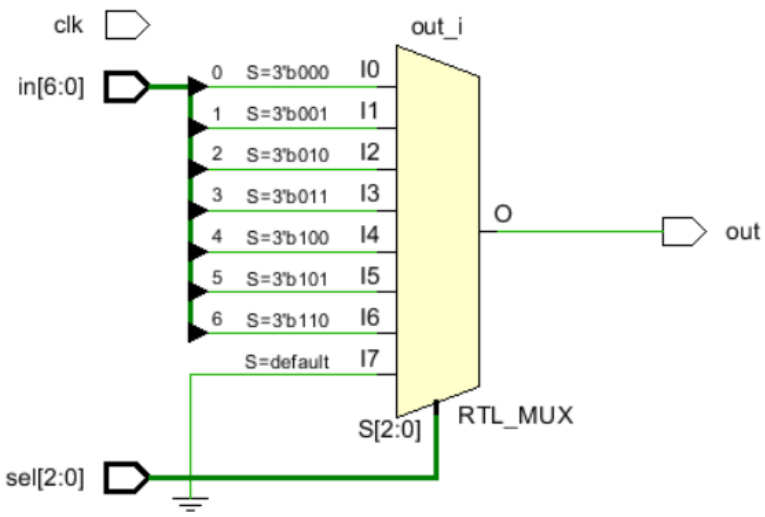


(Figure 10)

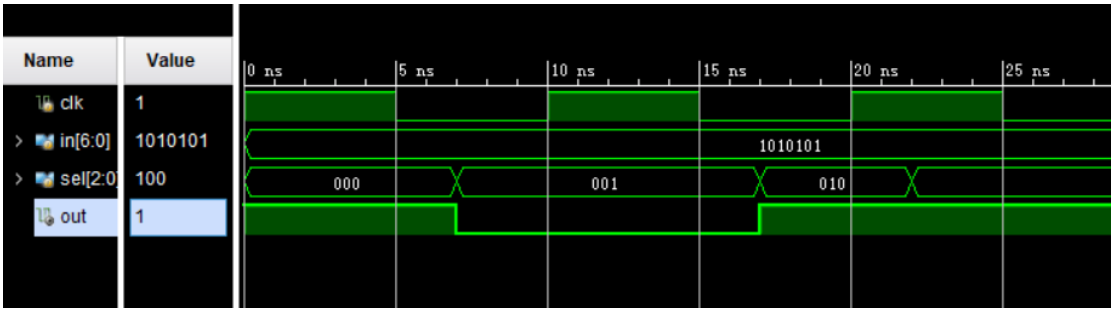
Figure 10 shows the simulation wave of 16-bit signed multiplier. We can see that the output only depends on the two inputs a, b. The previous state cannot affect the output. Thus, 16-bit signed multiplier is combinatorial logic circuit.

e. 7-bit multiplexer

```
always @(*) begin
        case (sel)
        3'b000 : out <= in[0];
        3'b001 : out <= in[1];
        3'b010 : out <= in[2];
        3'b011 : out <= in[3];
        3'b100 : out <= in[4];
        3'b101 : out <= in[5];
        3'b110 : out <= in[6];
        default : out <= 1'b0;
        endcase
```



(Figure 11)

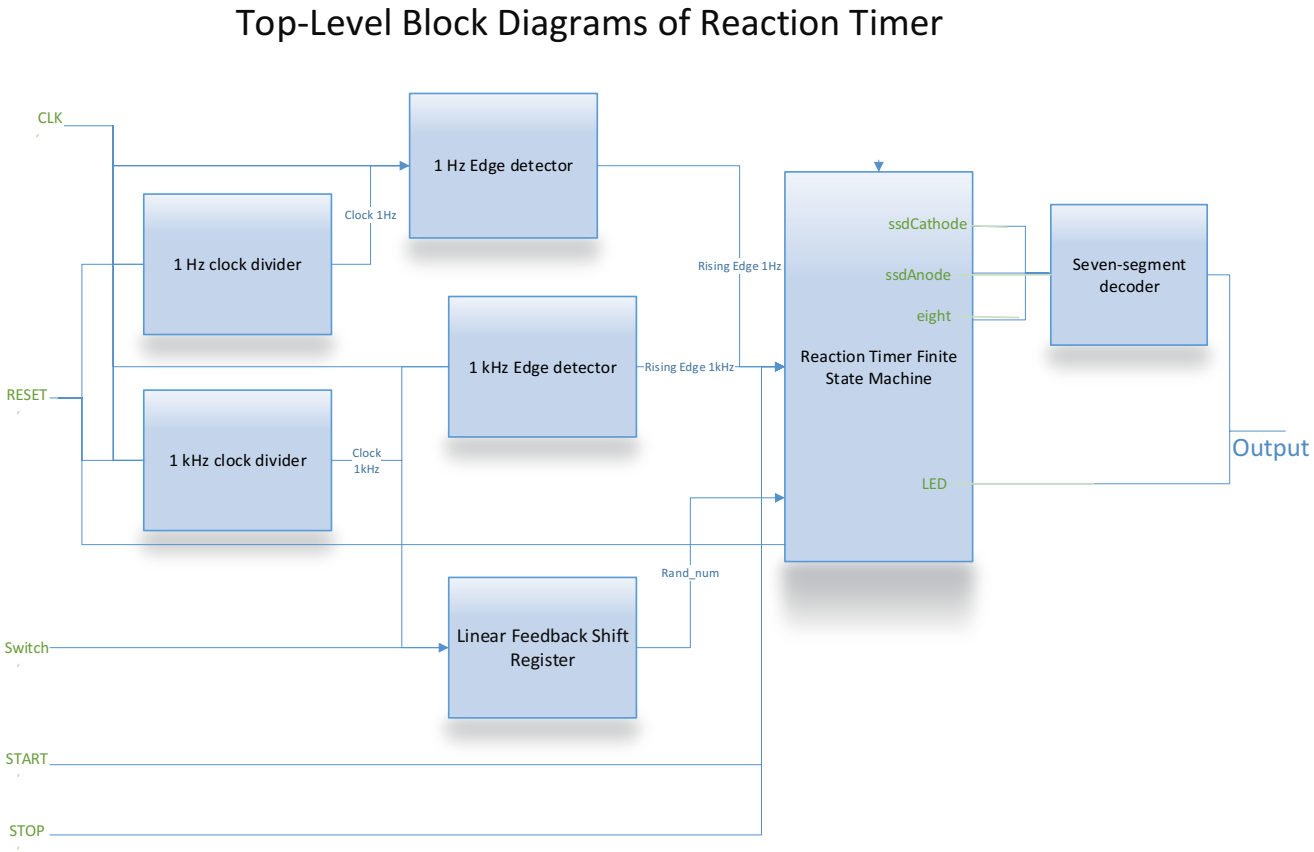


(Figure 12)

Figure 11 and Figure 12 are hardware and simulation wave of 7-bit multiplexer respectively. The output depends on the current input sel to select the current input in. thus, 7-bit multiplexer is combinatorial logic circuit.

Part Two: Reaction Timer

1. Reaction Timer High-level block diagram



(Figure 13)

Figure 13 illustrates the detailed top-level block diagram of reaction timer we built for Basys3 FPGA.

There are five inputs and 2 outputs designed for the reaction timer. CLK is the FPGA clock for timing purpose. RESET is used for resetting the system. START is used to activate the system to working. STOP is used to terminate the system. SWITCH is used to control the linear feedback shift register for generating random number. LED lights are used to indicate timing process. The seven-segment decoder is used to scan the timing result.

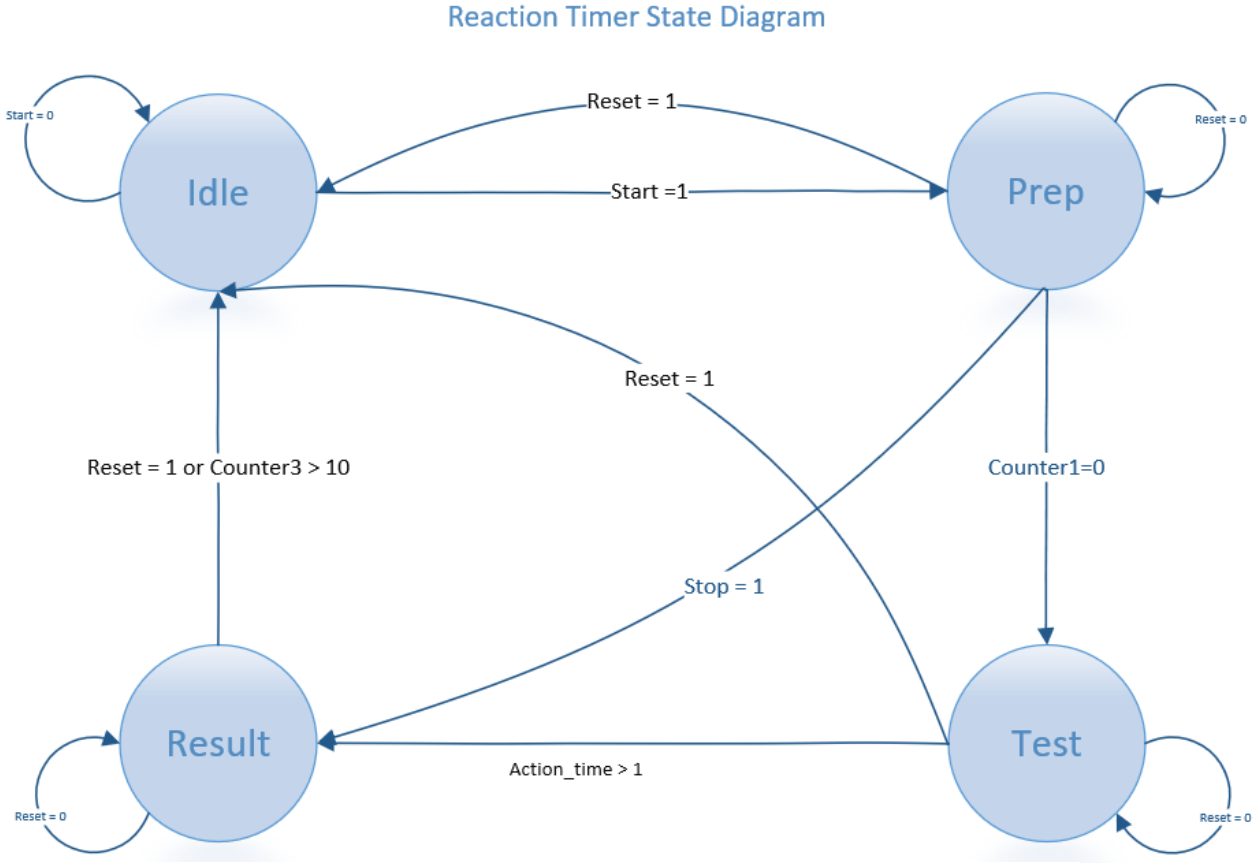
To generate 1kHz and 1Hz clock frequency for timing purpose, the frequency of the original 100 MHz FPGA clock is divided separately by both 1 kHz and 1 Hz clock dividers. Then two Edge detectors collect the divided clock frequency to generate the rising edge of 1kHz and 1Hz clock signal. These signals are passed to the finite state machine for counting purpose. After a series of working process, data are passed to the seven-segment decoder and LED lights for displaying the desired result.

2. Reaction Timer State Diagram

Figure 14 shows the state diagram of our designed reaction timer which contains four states. They are Idle, Prep, Test and Result. There are 3 separate counters used for Prep, Test and Result states. At any state of the system, when the reset button is on, the system will move to default state Idle.

- At Idle state, the state moves to Prep state when the start button is pushed.

- At prep state, when countdown counter (counter1) is counted to 0, the state moves to test state. If the stop button is pushed before counter1 decreases to 0, the state moves to result and display ‘FAIL’.



(Figure 14)

- At test state, when the value of counting result “action time” is saved and greater than 1, then state moves to result state to display the result.
- At the result state, when count-up counter (counter3) is greater than 10 seconds, the state will return the Idle.

3. Reaction Timer Operating Principle

To realize all required function in each state of the reaction timer, we have designed and programmed the Verilog code carefully. The detailed operation and verification process of the reaction timer are elaborated on the following paragraphs.

a) IDLE

At Idle state, the LED lights are turned off. The seven-segment decoder displays ‘----’, which means that the device is powered on and be ready to work. It is implemented by sending bcd and ssdAnode data to the seven-segment decoder. Meanwhile, initialize counter1 to 3 for the countdown process of the Prep state. All other variables are also set to their initial values.

During the design, we use the Basys3 FPGA to test whether the result is correct.

b) PREP

At Prep state, the LED lights are turned off, the seven-segment tube sequential displays 3, 2 and 1 at once in 3 seconds. Then the seven-segment decoder turns off and it is the signal for FSM to change the state. The countdown process is implemented by using 1Hz rising edge signal. Variable counter1 decreases 1'd1 every second until it reaches 0. Meanwhile, the case statement transfers the value of counter1 to bcd. Based on the bcd value, the seven-segment decoder displays the result during the countdown process.

c) TEST

At test state, the seven-segment displayer are turned off. All LED lights will turn on after a special waiting period. By using a linear feedback shift-register (LFSR) to generate a pseudo-random number. Because the feedback bits are constant, the next random number is predictable. Thus, it is pseudo. This code used an 8- bit shift-register. [3] The input and output number are 8 bits and the random number ranges from 0 to 2^8. The aim of the special period is between 1 second and 5 seconds, and the clock frequency is 100 MHz, so the count number should between 49_999_999 and 250_000_000. (The random number * 784_314 + 49_999_999) approximate (49_999_999, 250_000_000). When the switch is off, the period is constant. Because the LFSR is not working. When it is turned on, the LFSR will generate random number every 1 second. The reaction timer starts counting by using the 1kHz clock signal. At each 1kHz rising edge, variable counter2 increases 1'd1 every 0.001 second. When the STOP button is on, the current result of counter2 is stored into the variable "action time", then counter2 is then reset to 0 for next reaction test. The variable "action time" is then passed to the state Result.

d) RESULT

At result state, if the action time is greater than 9.999 seconds or the STOP button is on during the Prep state (counter1!= 0), the seven-segment tube displays "FAIL". On the other hand, the value of action_time will be divided each bit in decimal and it will be stored into variable a, b, c, d. The function is moding 10 to get the lowest significant bit (action_time%10). Exact diving 10 and mod 10 to get the second significant bit ((action_time/10)%10). Exact diving 100 and mod 10 to get the third significant bit ((action_time/100)%10). Exact diving 1000 and mod 10 to get the highest significant bit ((action_time/1000)%10). In result displaying, by setting eight =1 at the highest bit, and display dot in the seven segments. eight =0 at other bits and no dot in seven segments. Thus, the seven-segment tube displays a four digits number with decimal dot next to the highest bit.

For judging whether the current action time is the fastest, a new variable "mini action time' is used to compare with current action time. If the current action time is smaller than the mini action time, then change the pattern of LED and store the value of current action time to mini action time. Otherwise turn off all LED lights and remain mini action time to its own value.

Meanwhile, if the reset button is not pushed, the reaction timer starts counting by using the 1Hz clock signal. At every 1Hz rising edge, counter3 increase 1 bit per second until its value is greater than 10, then the reaction timer returns to Idle state.

e) Seven-segment decoder

The seven-segment decoder has a 4 bits input "bcd" and a 1-bit input "eight". The output is 8 bits variable "ssd". If eight =1, then the number with the dot is displayed on the segment tube. Other no dot is displayed on the tube.

f) Constraint design

In the target FPGA device (Basys3):
The RESET button is assigned to pin W19.
The START button is assigned to pin U18.
The STOP button is assigned to pin T18.
The SWITCH button is assigned to pin V17.
Eight seven-segment cathodes and four anodes are all connected.
16 LED lights are assigned to variable led.

References

[1]"D Flip-Flop Circuit Diagram: Working & Truth Table Explained". [Online]. Available: https://circuitdigest.com/electronic-circuits/d-flip-flops.

[2]"Need help with D Flip Flop". [Online]. Available: https://www.physicsforums.com/threads/need-help-with-d-flip-flop.534311/.

[3]”BitArt. (2012). Available: https://www.cnblogs.com/BitArt/archive/2012/12/22/2827005.html

Appendix

The code for Part2 is in file. It includes assignment_reaction_timer1.v, sevenSegmentDecoder.v, integer_clockDivider.v, edgeDetector.v, sevenSegmentDecoder.v, assignment_reaction_timer1_constraint.xdc.