ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

# BLG 242E
# DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

| | | |
|---|---|---|
| **EXPERIMENT NO** | : | 4 |
| **EXPERIMENT DATE** | : | 07.04.2021 |
| **LAB SESSION** | : | FRIDAY - 10.30 |
| **GROUP NO** | : | G17 |

**GROUP MEMBERS:**

| | | |
|---|---|---|
| 150180024 | : | ŞULE BEYZA KARADAĞ |
| 150190710 | : | SENİHA SERRA BOZKURT |
| 150190024 | : | AHMET FURKAN KAVRAZ |

**SPRING 2021**

## Contents

## 1 INTRODUCTION [10 points]

In the following experiment, we are going to implement a positive-edge-triggered pulse generator using the Verilog.

1. We will implement a NAND gate module.
2. By using it, we will implement an SR Latch WITHOUT & WITH enable input.
3. Then we will implement a D Latch and by using it we will implement a D Flip-Flop.
4. Finally, we will design a positive-edge-triggered pulse generator by using circular shift register.

## 2 MATERIALS AND METHODS [40 points]

### 2.1 Preliminary

#### 2.1.1 Flip-Flop

A Flip-flop is a circuit that has two stable states. It can be used as storage elements. Flip-flops are triggered by a clock signal. Therefore, the stored data can be changed by clock signal. Also we can use the positive or negative edge triggering of the clock signal to change the output.

#### 2.1.2 Latches & Flip-Flops

The main difference between latches and a flip-flops is clocking process. Latches does not have clock inputs. They can be change by only enable input. Flip-flops are triggered by a clock signal. The output is changed only when an active clock signal is given.

#### 2.1.3 SR Latch

A SR latch can be created with two NAND gates that have a cross-feedback loop. It has two stable states, one state is referred to as complement of set and the other as complemented of reset, $\overline{S}$ and $\overline{R}$ respectively. Qn output must be the complement of Q output.

The input S=1, R=0 sets the output Q=1, the output Qn = 0; when S=0, R=0 outputs do not change, our latch preserves its status; S=0, R=1 resets the output Q. If the both inputs are "1" our state is invalid. Because it will try to produce a high Q and a low Q at the same time. But the circuit gives us the same outputs for both Q and Qn.

| Q(t) | S’ | R’ | Q(t+1) |
|---|---|---|---|
| 0 | 0 | 0 | x |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | x |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 1: 4. Truth table of $\overline{S}$ $\overline{R}$ Latch WITHOUT Enable

| E | Q(t) | S | R | Q(t+1) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | x |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | x |

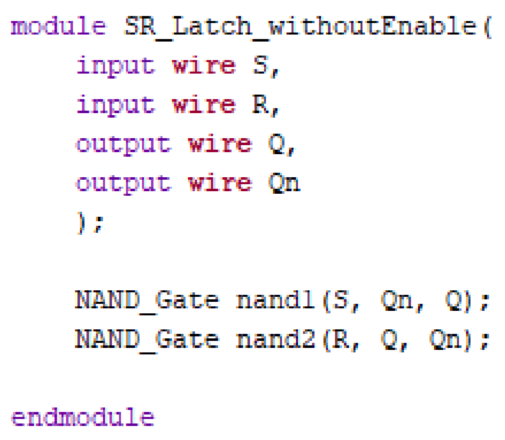
Table 2: 5. Truth table of $\overline{S}$ $\overline{R}$ Latch WITH Enable

| D | CLK | Q(t+1) | Qn(t+1) |
|---|---|---|---|
| 0 | $\uparrow$ | 0 | 1 |
| 0 | $\uparrow$ | 1 | 0 |
| x | 0 | Q(t) | Qn(t) |
| x | 1 | Q(t) | Qn(t) |

Table 3: 6. Truth table of D Flip Flop

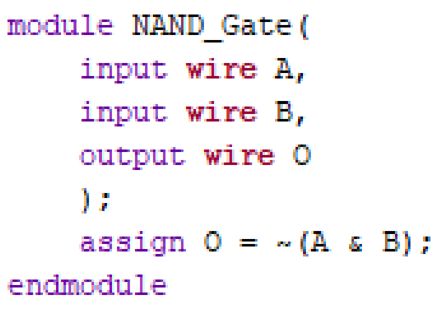
## 2.2 Experiment

### 2.2.1 Part 1



```
module SR_Latch_withoutEnable(
    input wire S,
    input wire R,
    output wire Q,
    output wire Qn
    );

    NAND_Gate nand1(S, Qn, Q);
    NAND_Gate nand2(R, Q, Qn);

endmodule
```

(a) SR Latch without Enable Module Code



```
module NAND_Gate(
    input wire A,
    input wire B,
    output wire O
    );
    assign O = ~(A & B);
endmodule
```

(b) NAND gate Module Code

Figure 1: Implemented SR Latch with NAND gates

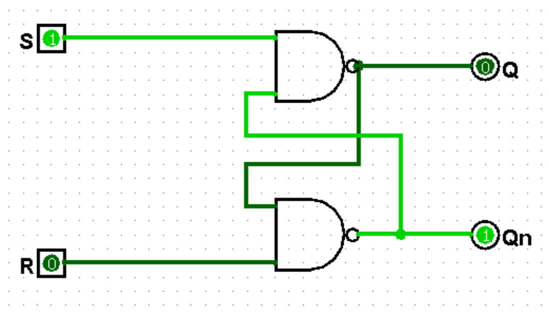


Figure 2: Circuit of SR latch without enable

In this part, we are required to implement a S-R latch with NAND gates. In order to do that we connected the NAND gates in such a way that the output of one gate feeds back to the input of other gate (see Figure 2).

As you can see in the Figure 3; we implement the true points in the Karnaugh Map according to truth table in **Table 1** which we constructed before for SR latch and we take all don't care values as "1". Then, we obtained the following expression for $Q(t+1)$ in terms of S, R, Q(t):

$$S + Q(t).\overline{R}$$

When $\overline{S}$ and $\overline{R}$ inputs are both "0", value "1" appears on both outputs. The reason of this situation is that the value "0" is dominant in the "AND" gate, so 2 "NAND" gates give "1" output regardless of the other input. But the outputs must be complement of

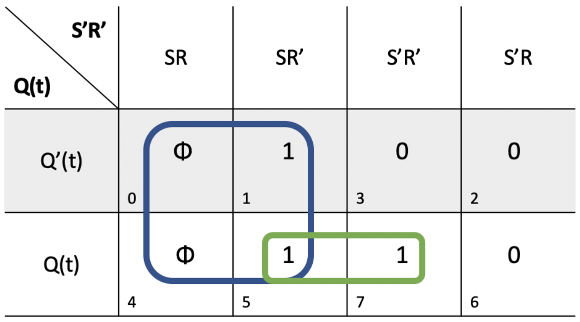


Figure 3: Karnaugh Map for the truth table in Figure 1

each other. Therefore, $\overline{S} = 0$ $\overline{R} = 0$ are called as disallowed inputs.

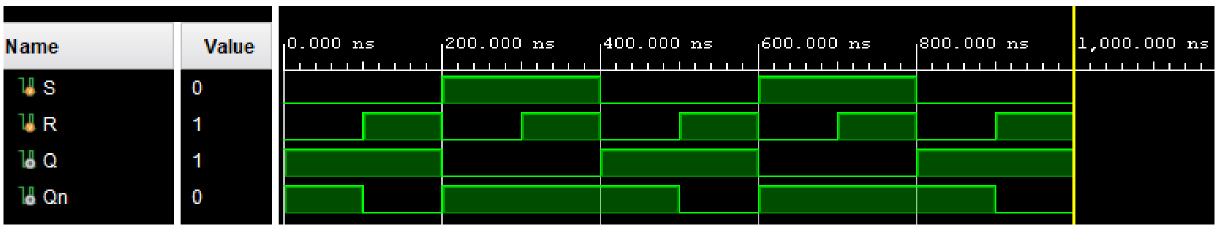
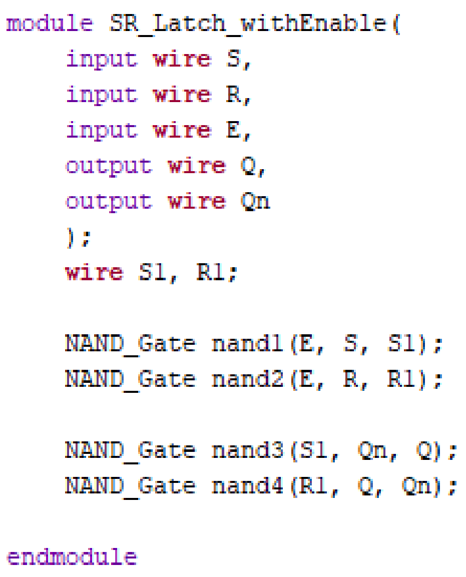


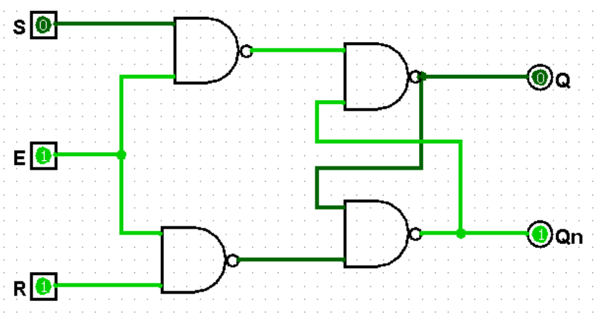
Figure 4: Simulation Results of SR Latch without Enable

### 2.2.2 Part 2



```
module SR_Latch_withEnable(
    input wire S,
    input wire R,
    input wire E,
    output wire Q,
    output wire Qn
    );
    wire S1, R1;

    NAND_Gate nand1(E, S, S1);
    NAND_Gate nand2(E, R, R1);

    NAND_Gate nand3(S1, Qn, Q);
    NAND_Gate nand4(R1, Q, Qn);

endmodule
```

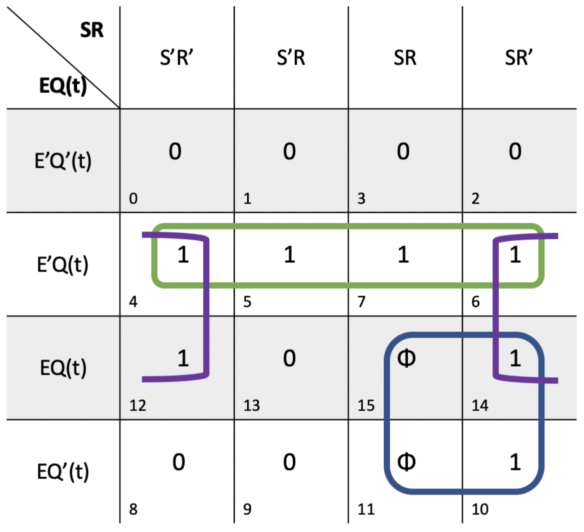
(a) SR Latch with Enable Module Code



(b) Circuit of SR Latch with Enable Input

Figure 5: Implemented SR Latch with enable input

In this part of the experiment, we are required to implement a SR latch with an additional Enable input. Therefore, we add an E input to the circuit shown in **Figure 2** and connected S, R, E inputs to the leftover parts of NAND Gates (see Figure 5.b).



| SR / EQ(t) | S′R′ | S′R | SR | SR′ |
|---|---|---|---|---|
| E′Q′(t) | 0 0 | 0 1 | 0 3 | 0 2 |
| E′Q(t) | 1 4 | 1 5 | 1 7 | 1 6 |
| EQ(t) | 1 12 | 0 13 | Φ 15 | 1 14 |
| EQ′(t) | 0 8 | 0 9 | Φ 11 | 1 10 |

Figure 6: Karnaugh Map for the truth table in Figure 2

As you may see in the Figure 6; we implement the true points in the Karnaugh Map according to truth table in **Table 2** which we constructed before for SR latch with enable input and we take all don't care values as "1". Then, we obtained the following expression for Q(t + 1) in terms of S, R, E, Q(t):

$$E.S + Q(t).\overline{R} + \overline{E}.Q(t)$$

The addition of enable input "E" allowed the S-R latch to change its state only if the enable input is high(1). If the S-R latch is disabled by giving a "0" value to input E, latch would preserve its state. When the S-R latch is enabled, if we give "1" value to both S and R inputs, both outputs would be "1". Disallowed inputs are different from the inputs in **Part 1** because we used additional NAND gates. The output of the NAND gates on the left hand side is "0", and therefore the same situation occurs with the forbidden "0" input in the **Part 1**.

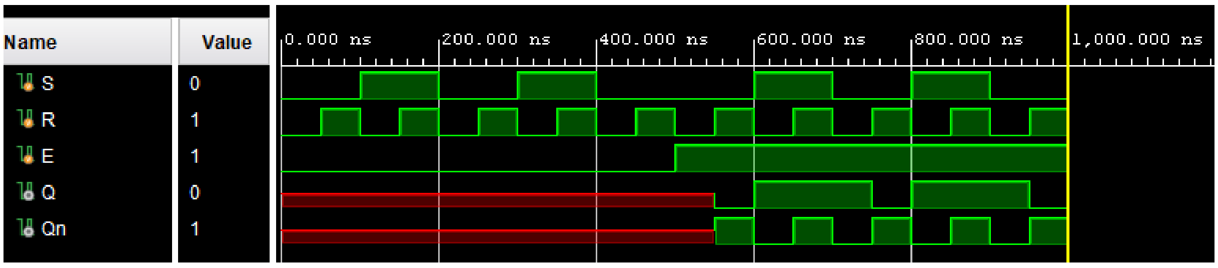
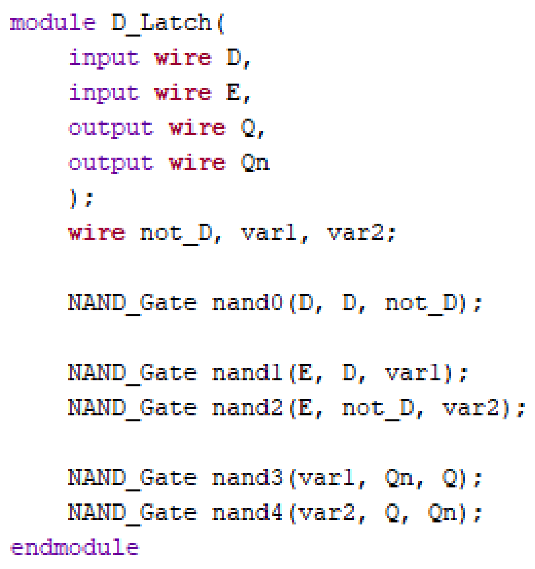


Figure 7: Simulation result of S-R latch WITH enable

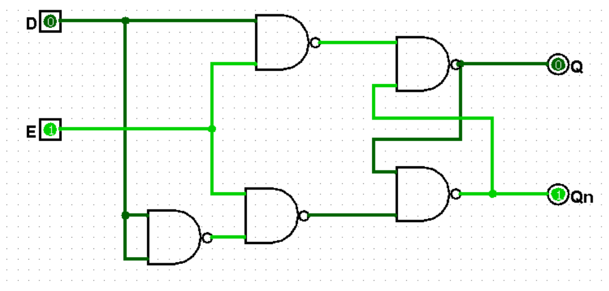
### 2.2.3 Part 3

In this part, we implemented a positive edge triggered D flip-flop module with D input and for $Q$ and $Q_{neg}$ outputs using D latches. We implemented the D latches ourselves using only 2-input NAND gates as a separate module we implemented in the above parts. To demonstrate better that the clock is only effective at rising edge, see Figure-10.



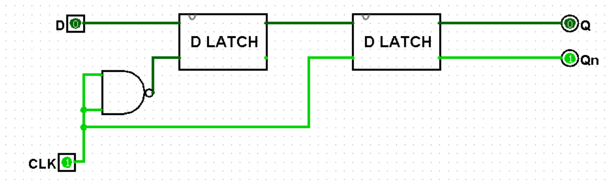
```
module D_Latch(
    input wire D,
    input wire E,
    output wire Q,
    output wire Qn
    );
    wire not_D, var1, var2;

    NAND_Gate nand0(D, D, not_D);

    NAND_Gate nand1(E, D, var1);
    NAND_Gate nand2(E, not_D, var2);

    NAND_Gate nand3(var1, Qn, Q);
    NAND_Gate nand4(var2, Q, Qn);
endmodule
```

(a) D Latch Module Code

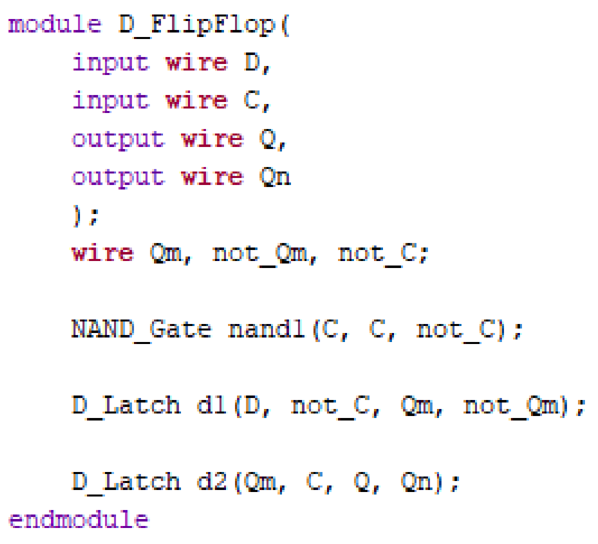


(b) D Latch Circuit

Figure 8: Implemented D Latch



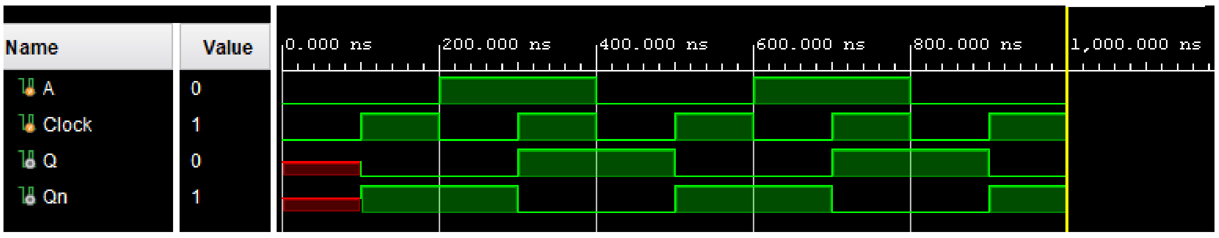
(a) D Flip-Flop Circuit



```
module D_FlipFlop(
    input wire D,
    input wire C,
    output wire Q,
    output wire Qn
    );
    wire Qm, not_Qm, not_C;

    NAND_Gate nand1(C, C, not_C);

    D_Latch d1(D, not_C, Qm, not_Qm);

    D_Latch d2(Qm, C, Q, Qn);
endmodule
```

(b) D Flip-Flop Module Code

Figure 9: Implemented D Flip-Flop



| Name | Value |
|---|---|
| A | 0 |
| Clock | 1 |
| Q | 0 |
| Qn | 1 |

Figure 10: Simulation Results of Rising-Edge D Flip-Flop

### 2.2.4 Part 4

In this part, we implemented a positive edge triggered pulse generator using a circular shift register. The circuit took 16-bit input for the loaded value, 1-bit input for the clock signal, 1-bit input for the load flag and gave 1-bit output. Basically, when $Load = 0$ (that is, $Shift = 1$), with the clock signal, circular shift operation is done; when $Load = 1$, with the clock signal, a 16-bit input value is loaded.

We designed our circuit in a way that the output of this circuit is the most significant bit (MSB) of the loaded value (so, firstly we determined what the direction of the shift should be = that is **from Least Significant Bit to Most Significant Bit** to maintain MSB as the output).

Our design supports variable pulse frequencies and duration listed below. We built the circuit and generate given signals. For each signal, we observed both input and output. To show better that our design supports those gaps and frequencies, we implemented the simulation results in figures below.

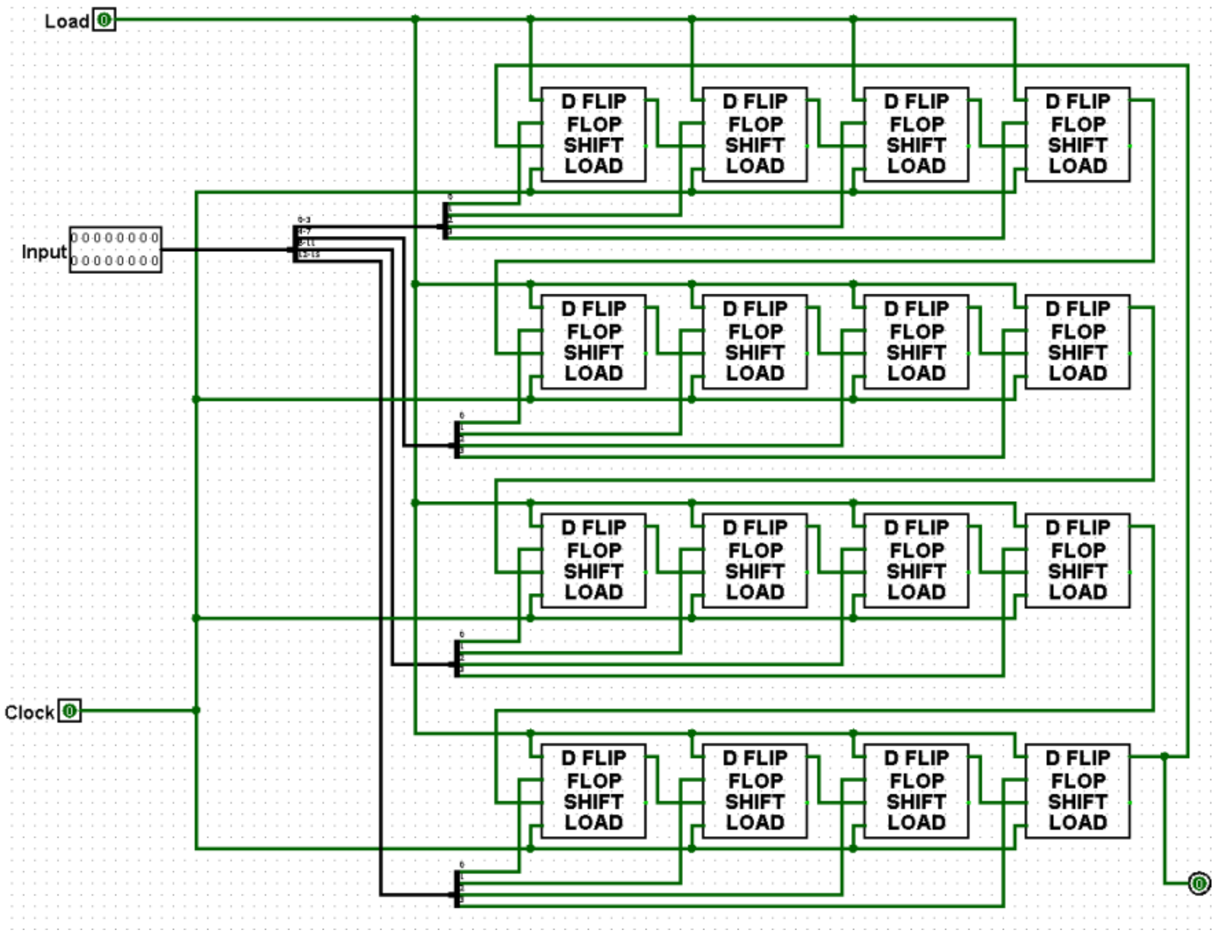
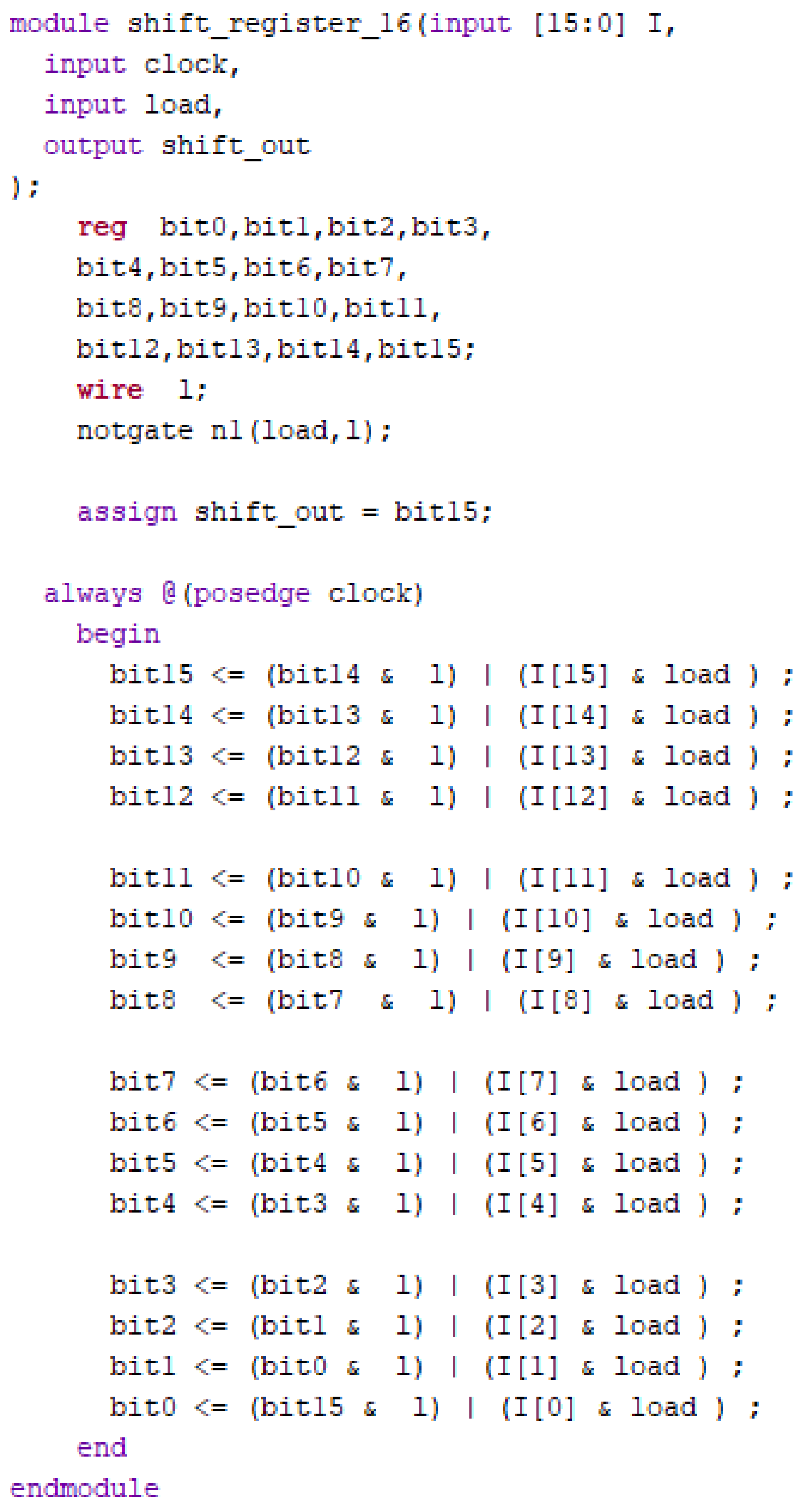


Figure 11: 16 Bit Circular Left Shift Register



```
module shift_register_16(input [15:0] I,
  input clock,
  input load,
  output shift_out
);
    reg  bit0,bit1,bit2,bit3,
    bit4,bit5,bit6,bit7,
    bit8,bit9,bit10,bit11,
    bit12,bit13,bit14,bit15;
    wire  l;
    notgate n1(load,l);

    assign shift_out = bit15;

  always @(posedge clock)
    begin
      bit15 <= (bit14 &  l) | (I[15] & load ) ;
      bit14 <= (bit13 &  l) | (I[14] & load ) ;
      bit13 <= (bit12 &  l) | (I[13] & load ) ;
      bit12 <= (bit11 &  l) | (I[12] & load ) ;

      bit11 <= (bit10 &  l) | (I[11] & load ) ;
      bit10 <= (bit9 &  l) | (I[10] & load ) ;
      bit9  <= (bit8 &  l) | (I[9] & load ) ;
      bit8  <= (bit7  &  l) | (I[8] & load ) ;

      bit7 <= (bit6 &  l) | (I[7] & load ) ;
      bit6 <= (bit5 &  l) | (I[6] & load ) ;
      bit5 <= (bit4 &  l) | (I[5] & load ) ;
      bit4 <= (bit3 &  l) | (I[4] & load ) ;

      bit3 <= (bit2 &  l) | (I[3] & load ) ;
      bit2 <= (bit1 &  l) | (I[2] & load ) ;
      bit1 <= (bit0 &  l) | (I[1] & load ) ;
      bit0 <= (bit15 &  l) | (I[0] & load ) ;
    end
endmodule
```

Figure 12: Code of Left Shift Register

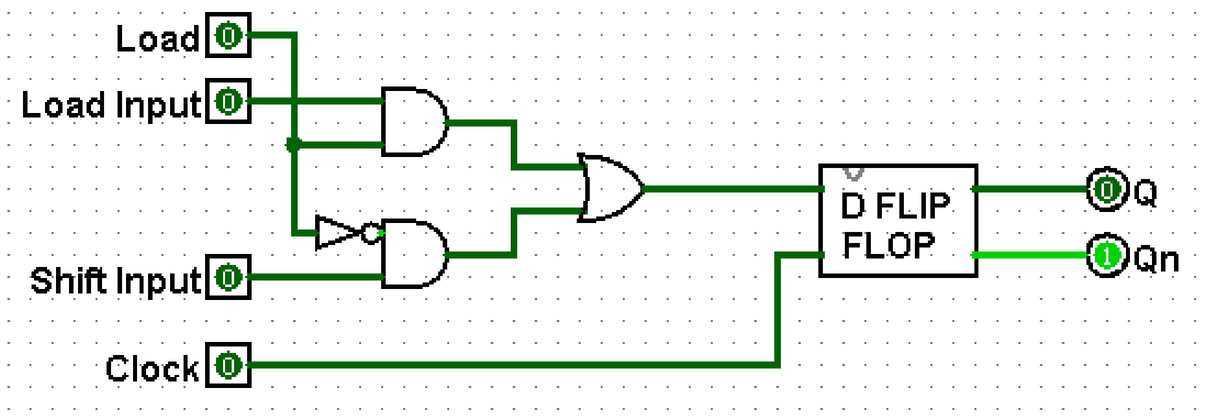


Figure 13: D Flip Flop with Shift and Load

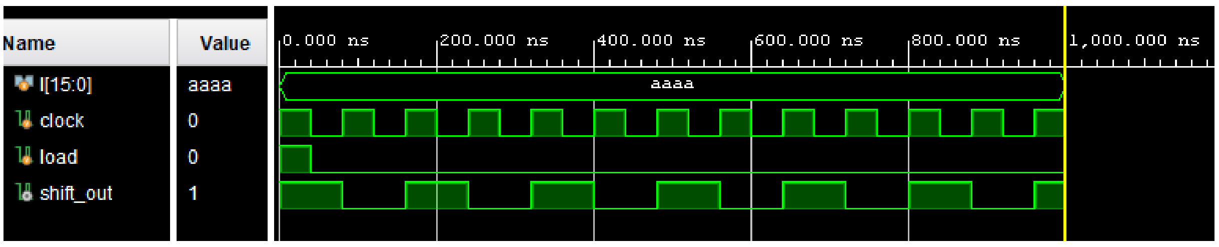


Figure 14: With the 1/2 frequency of clock signal

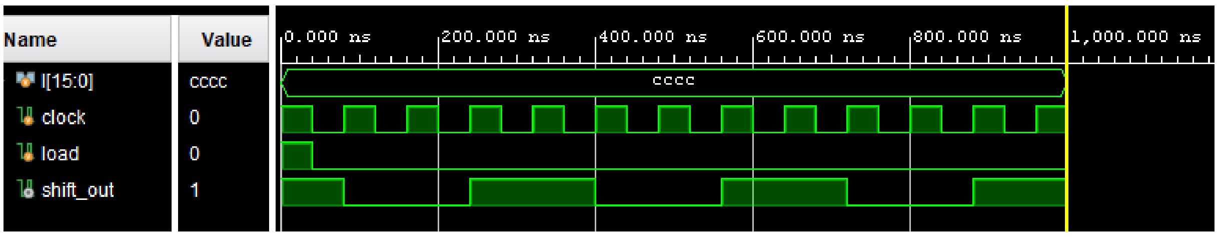


Figure 15: With the 1/4 frequency of clock signal

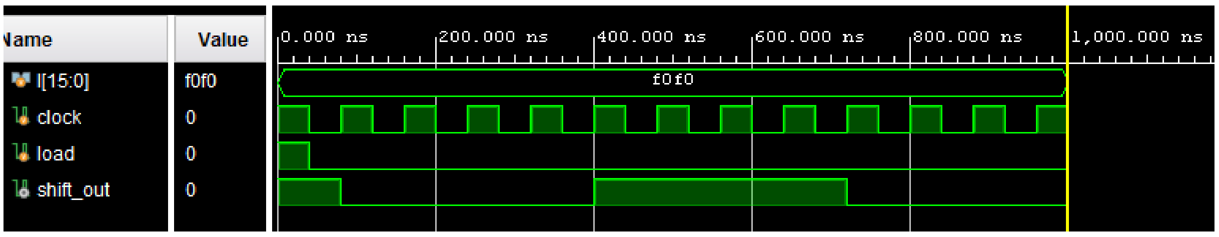


Figure 16: With the 1/8 frequency of clock signal

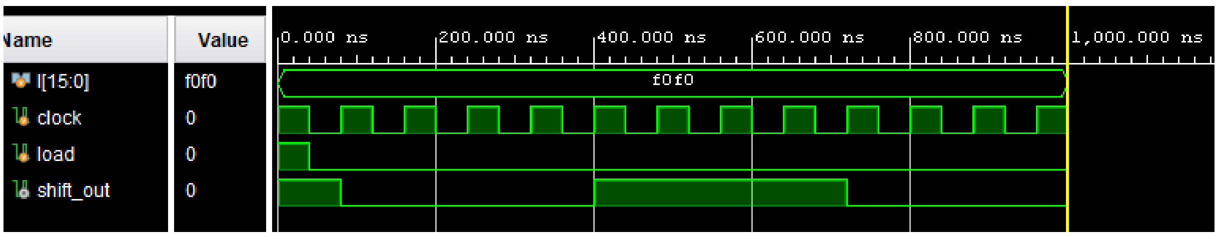


Figure 17: With 1/7 pulse-gap duration rate

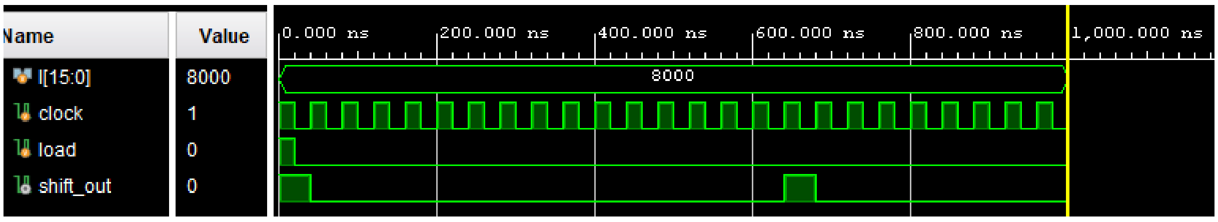


Figure 18: With 1/15 pulse-gap duration rate

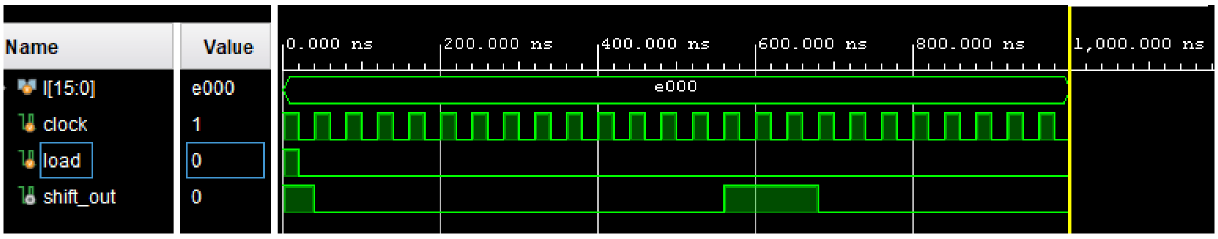


Figure 19: With 3/13 pulse-gap duration rate

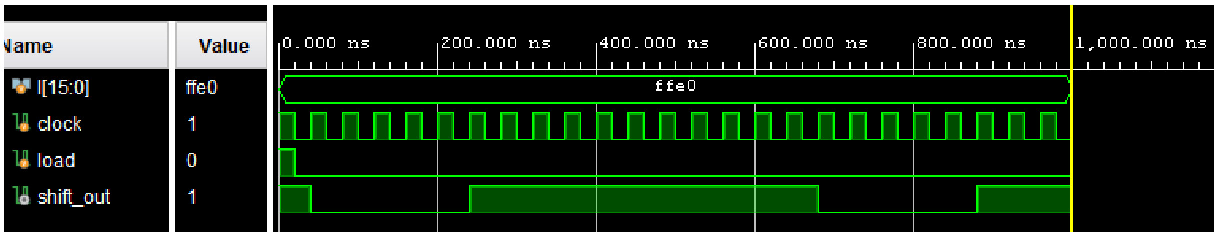
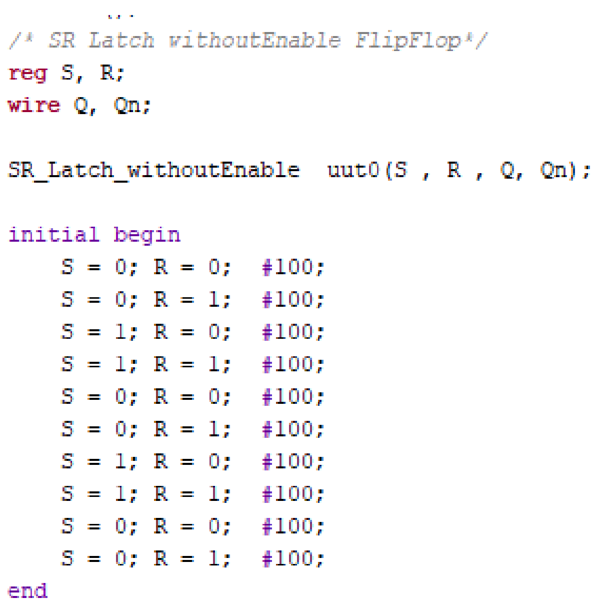


Figure 20: With 11/5 pulse-gap duration rate

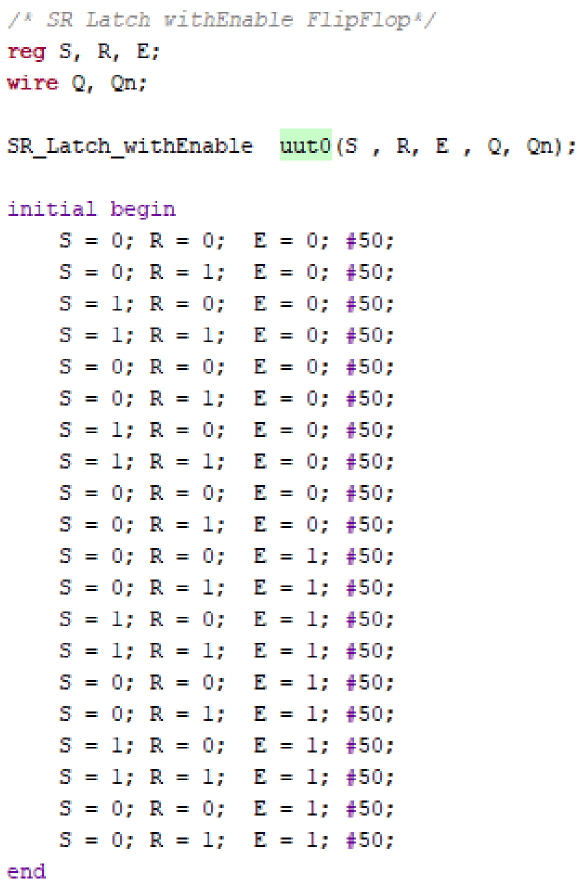
## 3 RESULTS [15 points]

You can see the simulation codes we used for all the Experiments' parts and their simulation results, all separately ran one after another, while the other are in the commented version to prevent the program from crush. To carry out each part of the experiment, we arranged inputs and the time slots of each simulation according to part or frequency or pulse/gap duration it belongs. The simulation results we obtained after completing each step of the experiments resulted as we expected. This concludes the fact that our experiment ended up being coherent.



```
        ...
/* SR Latch withoutEnable FlipFlop*/
reg S, R;
wire Q, Qn;

SR_Latch_withoutEnable  uut0(S , R , Q, Qn);

initial begin
    S = 0; R = 0;  #100;
    S = 0; R = 1;  #100;
    S = 1; R = 0;  #100;
    S = 1; R = 1;  #100;
    S = 0; R = 0;  #100;
    S = 0; R = 1;  #100;
    S = 1; R = 0;  #100;
    S = 1; R = 1;  #100;
    S = 0; R = 0;  #100;
    S = 0; R = 1;  #100;
end
```

(a) SR Latch without Enable



```
/* SR Latch withEnable FlipFlop*/
reg S, R, E;
wire Q, Qn;

SR_Latch_withEnable  uut0(S , R, E , Q, Qn);

initial begin
    S = 0; R = 0;  E = 0; #50;
    S = 0; R = 1;  E = 0; #50;
    S = 1; R = 0;  E = 0; #50;
    S = 1; R = 1;  E = 0; #50;
    S = 0; R = 0;  E = 0; #50;
    S = 0; R = 1;  E = 0; #50;
    S = 1; R = 0;  E = 0; #50;
    S = 1; R = 1;  E = 0; #50;
    S = 0; R = 0;  E = 0; #50;
    S = 0; R = 1;  E = 0; #50;
    S = 0; R = 0;  E = 1; #50;
    S = 0; R = 1;  E = 1; #50;
    S = 1; R = 0;  E = 1; #50;
    S = 1; R = 1;  E = 1; #50;
    S = 0; R = 0;  E = 1; #50;
    S = 0; R = 1;  E = 1; #50;
    S = 1; R = 0;  E = 1; #50;
    S = 1; R = 1;  E = 1; #50;
    S = 0; R = 0;  E = 1; #50;
    S = 0; R = 1;  E = 1; #50;
end
```

(b) SR Latch with Enable

Figure 21: Codes of SR Latches

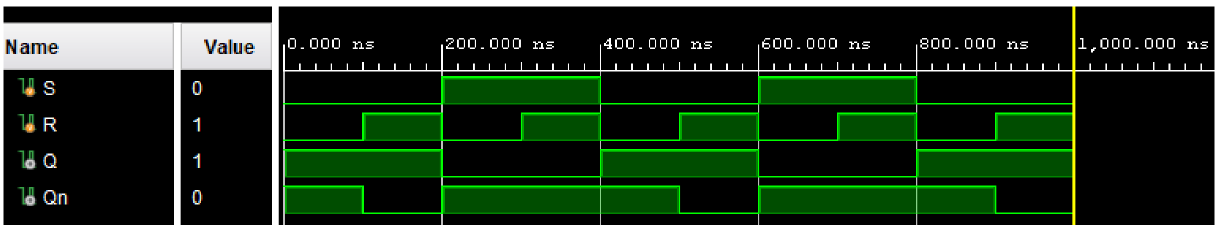


Figure 22: Simulation Result of SR Latch WITHOUT Enable

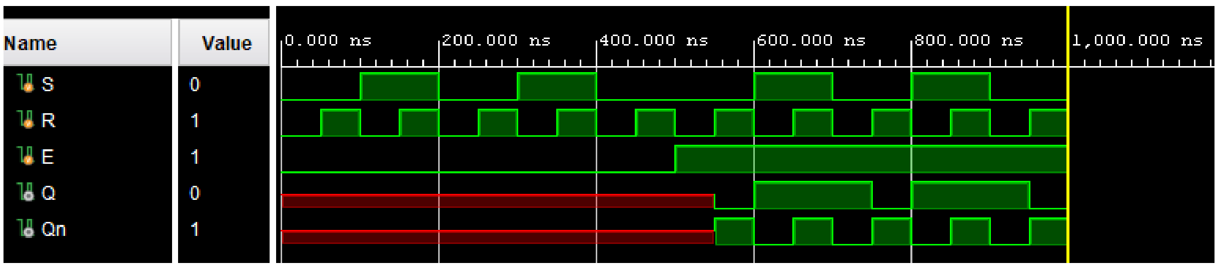
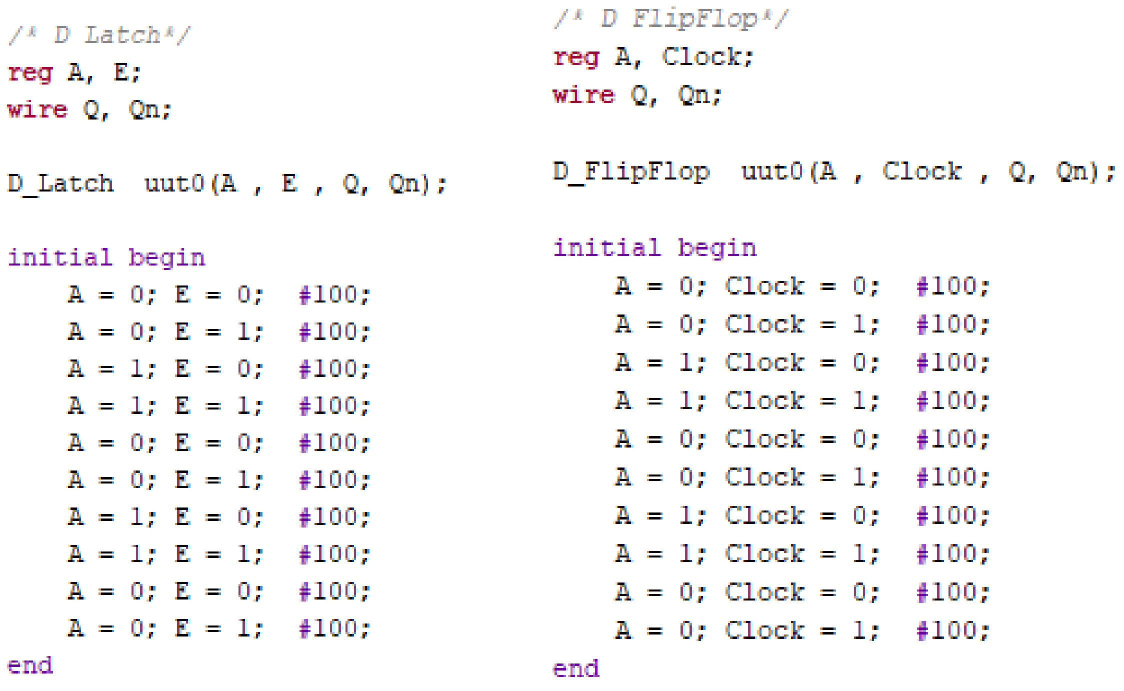


Figure 23: Simulation Result of SR Latch WITH Enable



```
/* D Latch*/
reg A, E;
wire Q, Qn;

D_Latch  uut0(A , E , Q, Qn);

initial begin
    A = 0; E = 0;  #100;
    A = 0; E = 1;  #100;
    A = 1; E = 0;  #100;
    A = 1; E = 1;  #100;
    A = 0; E = 0;  #100;
    A = 0; E = 1;  #100;
    A = 1; E = 0;  #100;
    A = 1; E = 1;  #100;
    A = 0; E = 0;  #100;
    A = 0; E = 1;  #100;
end
```

(a) D Latch

```
/* D FlipFlop*/
reg A, Clock;
wire Q, Qn;

D_FlipFlop  uut0(A , Clock , Q, Qn);

initial begin
    A = 0; Clock = 0;  #100;
    A = 0; Clock = 1;  #100;
    A = 1; Clock = 0;  #100;
    A = 1; Clock = 1;  #100;
    A = 0; Clock = 0;  #100;
    A = 0; Clock = 1;  #100;
    A = 1; Clock = 0;  #100;
    A = 1; Clock = 1;  #100;
    A = 0; Clock = 0;  #100;
    A = 0; Clock = 1;  #100;
end
```

(b) D FlipFlop

Figure 24: Simulation Codes of D Latch and FlipFlop

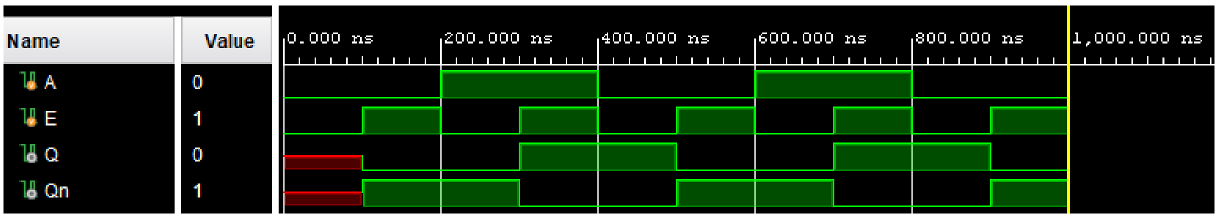


Figure 25: Simulation Result of D Latch

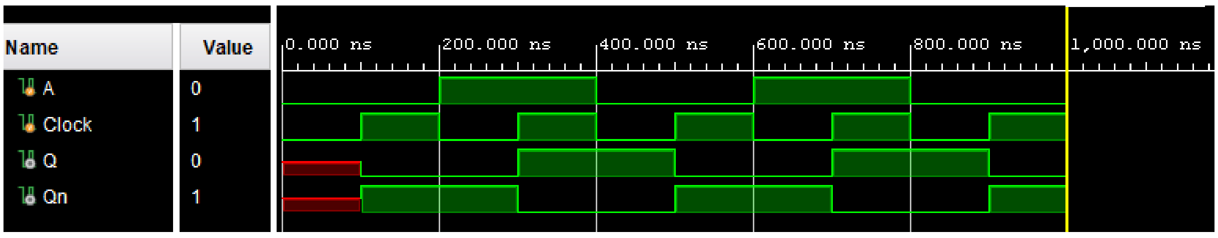
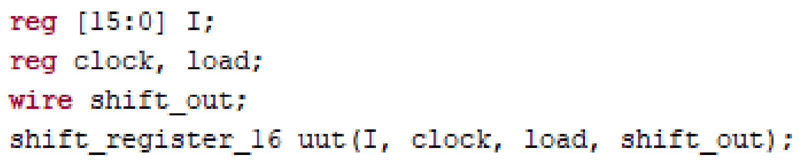


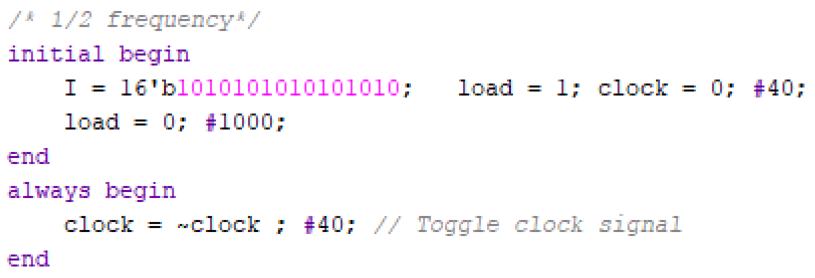
Figure 26: Simulation Result of D Flip-Flop

In Figure 27, we have a general code for circular shift registers. The other codes are about frequencies and pulse-gap durations. When we run this program we should use always Figure 26 and one of the other codes about frequencies and pulse-gap durations.



```
reg [15:0] I;
reg clock, load;
wire shift_out;
shift_register_16 uut(I, clock, load, shift_out);
```

Figure 27: Simulation Code of Shift Register



```
/* 1/2 frequency*/
initial begin
    I = 16'b1010101010101010;   load = 1; clock = 0; #40;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #40; // Toggle clock signal
end
```

Figure 28: Generated pulse with the 1/2 frequency of clock signal

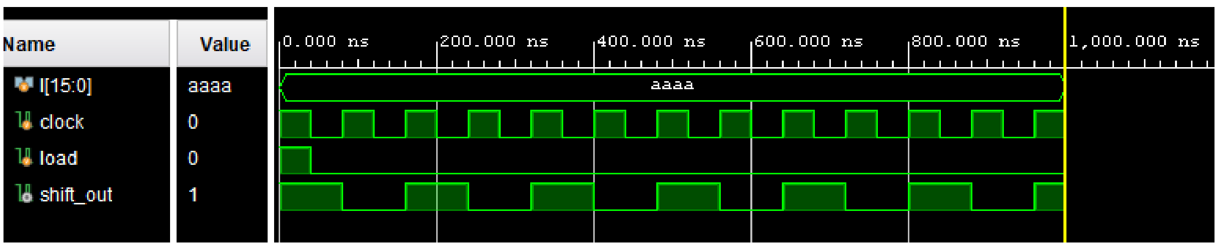


Figure 29: Simulation result with the 1/2 frequency of clock signal

```
/* 1/4 frequency*/
initial begin
    I = 16'b1100110011001100;   load = 1; clock = 0; #40;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #40; // Toggle clock signal
end
```

Figure 30: Generated pulse with the 1/4 frequency of clock signal

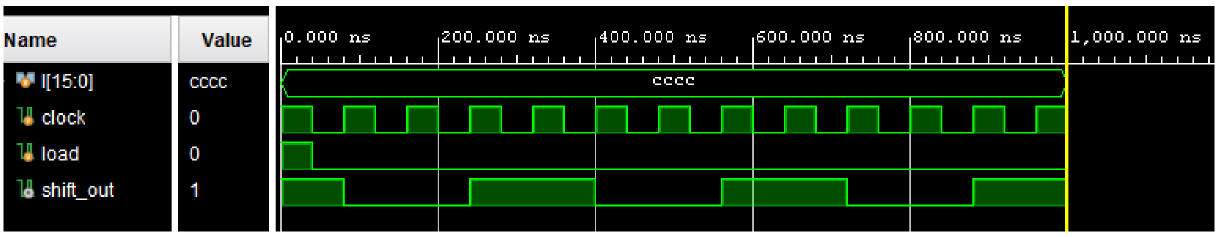


Figure 31: Simulation result with the 1/4 frequency of clock signal

```
/* 1/8 frequency*/
initial begin
    I = 16'b1111000011110000;   load = 1; clock = 0; #40;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #40; // Toggle clock signal
end
```

Figure 32: Generated pulse with the 1/8 frequency of clock signal

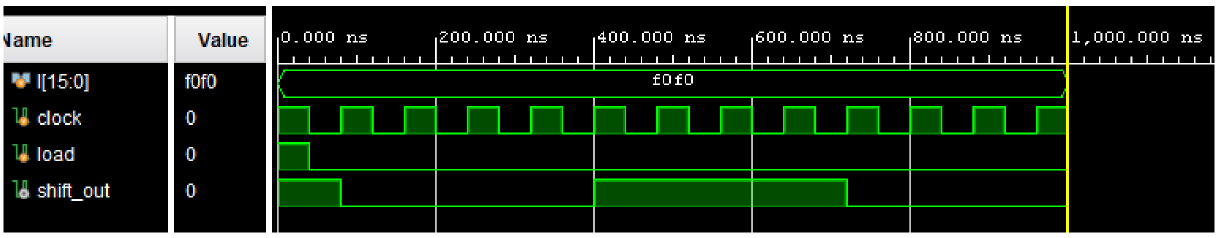
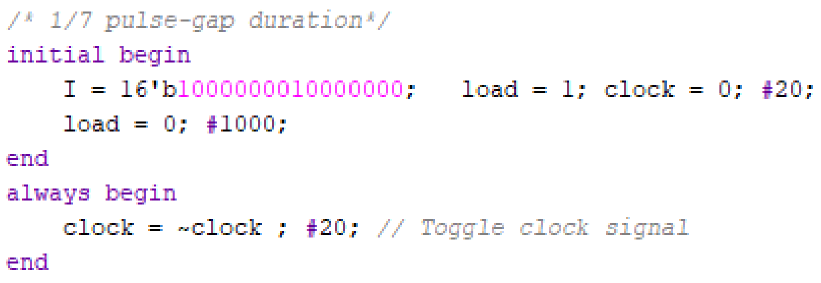


Figure 33: Simulation result with the 1/8 frequency of clock signal



```
/* 1/7 pulse-gap duration*/
initial begin
    I = 16'b1000000010000000;   load = 1; clock = 0; #20;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #20; // Toggle clock signal
end
```

Figure 34: Generated pulse with 1/7 pulse-gap duration rate

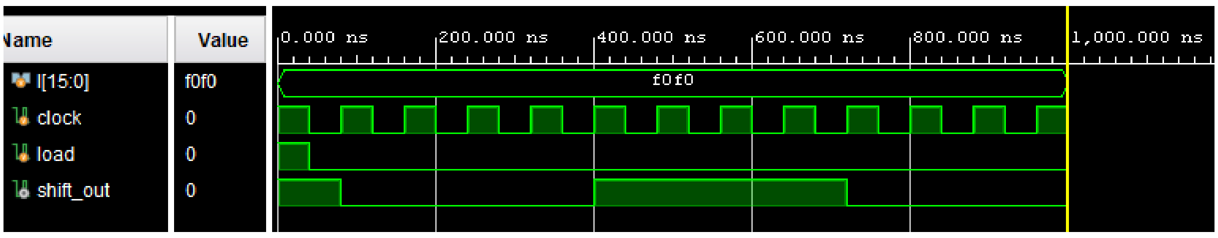
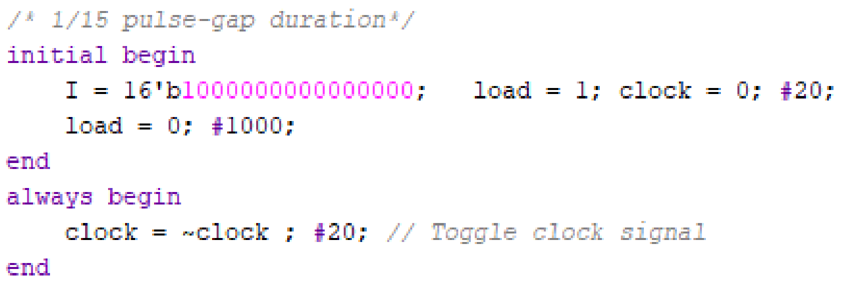


Figure 35: Simulation result with 1/7 pulse-gap duration rate



```
/* 1/15 pulse-gap duration*/
initial begin
    I = 16'b1000000000000000;   load = 1; clock = 0; #20;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #20; // Toggle clock signal
end
```

Figure 36: Generated pulse with 1/15 pulse-gap duration rate

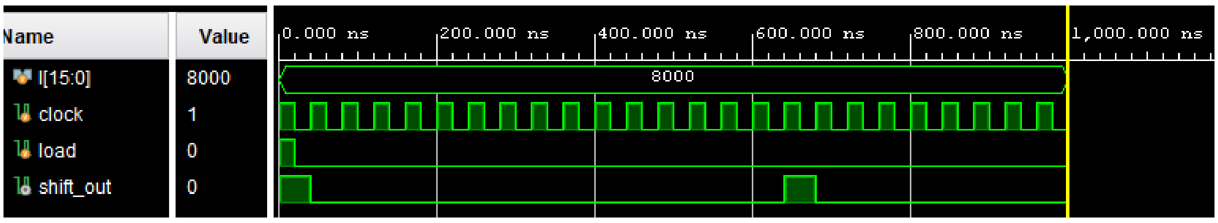


Figure 37: Simulation result with 1/15 pulse-gap duration rate

```
/* 3/13 pulse-gap duration*/
initial begin
    I = 16'b1110000000000000;   load = 1; clock = 0; #20;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #20; // Toggle clock signal
end
```

Figure 38: Generated pulse with 3/13 pulse-gap duration rate

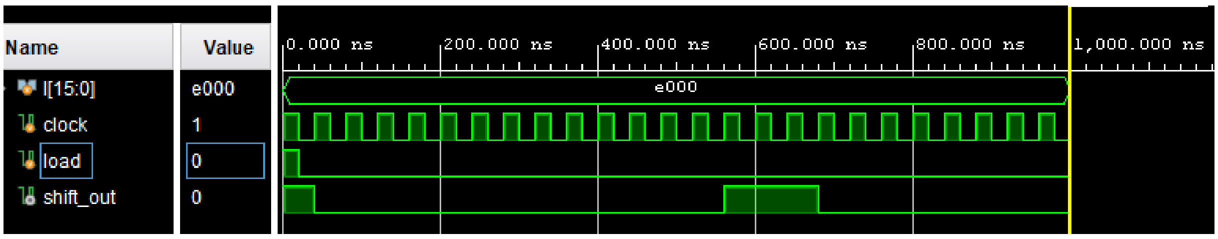


Figure 39: Simulation result with 3/13 pulse-gap duration rate

```
/* 11/5 pulse-gap duration*/
initial begin
    I = 16'b1111111111100000;   load = 1; clock = 0; #20;
    load = 0; #1000;
end
always begin
    clock = ~clock ; #20; // Toggle clock signal
end
```

Figure 40: Generated pulse with 11/5 pulse-gap duration rate

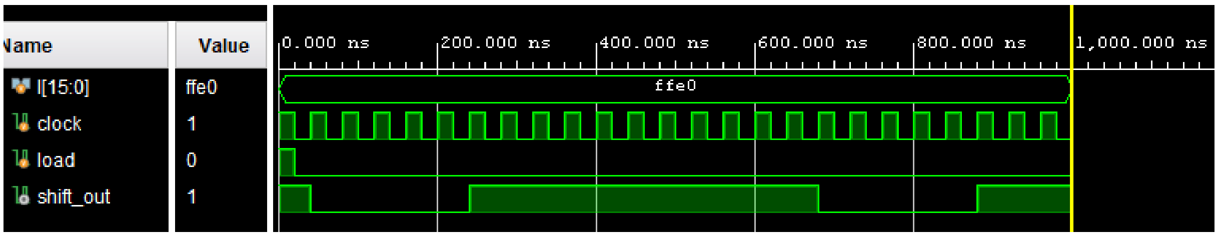


Figure 41: Simulation result with 11/5 pulse-gap duration rate

## 4 DISCUSSION [25 points]

**In Part-1**, we implemented S'R' Latch without Enable with using NAND Gates. First we implemented NAND gates and then we implemented the design of SR Latch as in the Figure 2.

**In Part-2**, we implemented S'R' Latch with Enable with using NAND Gates. We implemented the design of S'R' Latch (with Enable) as in the Figure 5-b.

**In Part-3**, first we implemented D Latch with Enable by using NAND Gates. Normally D Latch contains S'R' Latch, so we can also use S'R' Latch, but the question says that we should implement D Latch using only NAND Gates. So we only used NAND Gates. Also, we coded D Latch according to Figure 8-b.

After that, we implemented D Flip-Flop by using D Latches and NAND Gates. Also, we have a clock input which provides us positive or negative edge trigger. The D Flip-Flop is coded as in the Figure 9-a. Using 2 D Latches, we could have a Flip-Flop if we connect the Clock inputs to opposite signs (when one is 0 the other one will be 1). If we connect positive Clock value to first D Latch and negative Clock value to second D Latch, we have a negative edge triggered Flip Flop. So we just connect negative Clock value to first one and positive Clock value to second one. As a result, we code D Flip Flop according to Figure 9-a as positive edge triggered.

**In Part-4**, the question is about implementing Pulse Generator with using 16-bit Circular Shift Register. Circular Shift Register is a circuit which we can load an input or shift all bits left circularly. Circularly means that ,for left shifting, the value of Most Significant Bit goes to Least Significant Bit. For using load, we gave initial inputs our circuit and then for using circular shift, we can create pulses. The output value of the circuit which is the Most Significant Bit provides creating pulses. In Figure 11, we create this circular left shift register with the load. In circuit, we have "D Flip Flop Shift Load" circuit element for choosing which is only one of the Shift or Load Inputs. When we coding this, there is no need to use D Flip Flops. Because Verilog allows Shifting or Loading Inputs with using "reg"s, "always" block and parallel assignment which is "¡=".

When we implement the Positive Edge Triggered Pulse Generator, this is all about writing simulation codes with using circular shift register and giving inputs in simulation code.

- 1/2 frequency means that our inputs are sequentially one by one "1"s and "0"s, like (1010101010101010).
- 1/4 frequency means that our inputs are sequentially one by one "11"s and "00"s, like (1100110011001100).

- 1/8 frequency means that our inputs are sequentially one by one "1111"s and "0000"s, like (1111111100000000).
- 1/7 pulse-gap duration means that our inputs have 1 pulse (which is 1 in binary) and 7 gaps (which is 0 in binary) sequentially, like (1000000010000000).
- 1/15 pulse-gap duration means that our inputs have 1 pulse and 15 gaps sequentially, like (1000000000000000).
- 3/13 pulse-gap duration means that our inputs have 3 pulse and 13 gaps sequentially, like (1110000000000000).
- 11/5 pulse-gap duration means that our inputs have 11 pulse and 5 gaps sequentially, like (1111111111100000).

In final, we have Positive Edge Triggered Pulse Generator.

## 5 CONCLUSION [10 points]

We learned how to implemented $\overline{S}\ \overline{R}$ Latch without Enable or with Enable using only NAND Gates.

We already knew that the D latches and so do D flip-flops could be implemented using SR latches but in the question we had restrictions so we learned how to implemented D Latch using only NAND Gates, and we also learned how to implement rising-edge triggered D flip-flop using only NAND gates and D latches.

We learned how to implement a Pulse Generator with using 16-bit Circular Shift Register. We learned that Verilog allows Shifting or Loading Inputs with using "reg"s, "always" block and parallel assignment which is "<=". So we also became familiar with using them.

When we implemented the Positive Edge Triggered Pulse Generator, we simulated it for different frequencies and pulse-gap ratios given to us.

We learned how to respect the memory in our computers because the process on the back-end was so hard for us to implement, even for just 16 bit numbers and a circular shifting operation to get MOST SIGNIFICANT BIT was complicated. So we will never complain about the slow computers of ours, even during the games and stuff, and we will never say bad words to task manager, we will just compliment to him about all the good work he managed, and respect him.

The table generator provided to us doesn't really fit into all empty spaces in Latex, and is not fittable to any multicolumns and stuff. So we just tried to do the best we can.