DIGITAL SYSTEM DESIGN APPLICATIONS

(CRN: 11275)

THE REPORT OF EXPERIMENT – 5

Faculty of Electrical and Electronics Engineering

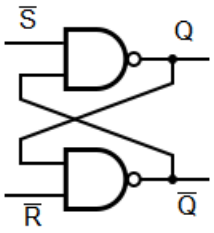
Electronics and Communication Engineering

Yusuf Tekin – 040200043

Yusuf Tekin
040200043

1. D Flip-Flop

SR LATCH:



TRUTH TABLE

INPUTS		OUTPUTS	
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	X	X
0	1	1	0
1	0	0	1
1	1	Q_0	$\overline{Q}_0$

Logic Diagram and Truth Table of SR NAND Latch[1]

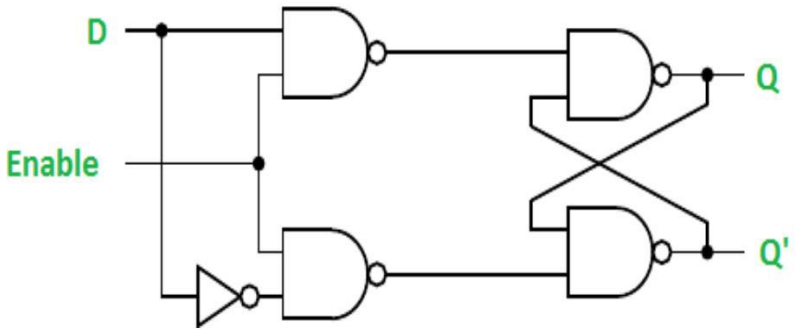
$$Q = (S \cdot Q')' \longrightarrow Q = (S \cdot (R \cdot Q)')' = S' + (R \cdot Q)$$
$$Q' = (R \cdot Q)' \qquad Q_{next} = S' + (R \cdot Q)$$

Characteristic Function of the SR NAND Latch

As it can be seen in the truth table the SR Latch has two inputs which are S' as *SET* and R' as *RESET.* Due to the designing with NAND gates to avoid affecting latching actions, the inputs are inverted in this circuit (active low). The circuit has 4 conditions which are set (S = 1, R = 0), reset (S = 0, R= 1), hold (S = 1, R = 1) and forbidden (S = 0, R = 0). The circuit sets Q = 1 in set condition, Q = 0 in reset condition, Q_{next} = Q in hold condition. However, in the forbidden condition the Q and Q' outputs are both forced to be "1" which creates a logical error and unstable condition. Designer should avoid this condition via never setting S = R = 0.

Yusuf Tekin
040200043

GATED D LATCH:



Logic Diagram of the Gated D Latch[2]

Enable	D	Q(n)	Q(n+1)	STATE
1	0	x	0	RESET
1	1	x	1	SET
0	x	x	Q(n)	No Change

Truth Table of the Gated D Latch[2]

Enable = 1 ⟶ $Q_{next} = D$

Enable = 0 ⟶ $Q_{next} = Q$

$$Q_{next} = (E \cdot D) + (\bar{E} \cdot Q)$$

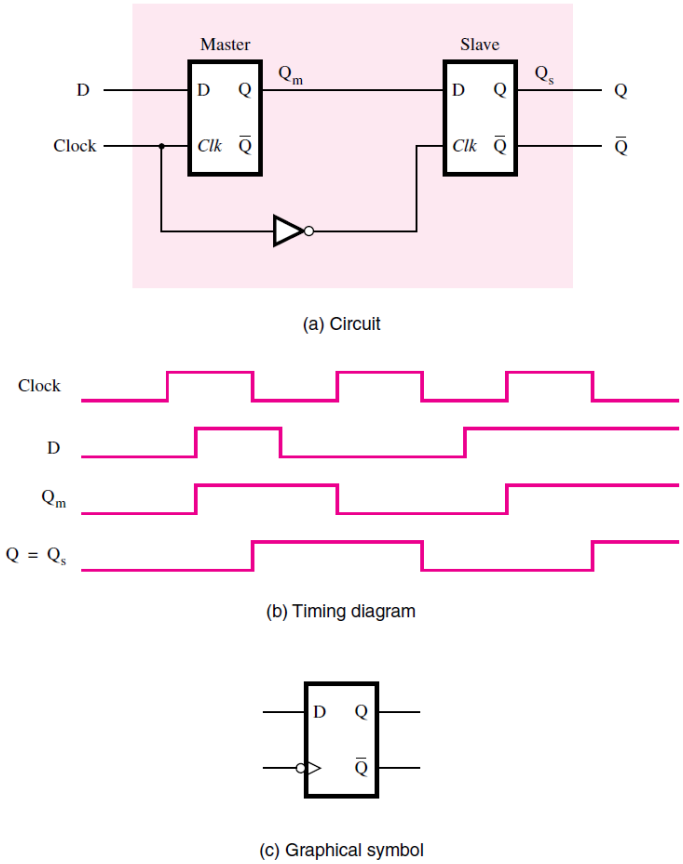
Characteristic Function of the Gated D Latch

Yusuf Tekin
040200043

EDGE-TRIGGERED D FLIP-FLOPs:

An Edge-Triggered D Flip-Flop is a sequential circuit that stores data on the rising or falling edge of a clock signal. It is a key component in digital systems used for synchronizing data and for creating registers, counters, and memory elements. Unlike a latch, which is level-sensitive, a flip-flop changes its output only on the edge of the clock signal.

A Master-Slave Flip-Flop is a type of edge-triggered flip-flop that uses two latches which are called the master latch, and the slave latch connected in series. In D Flip-Flop Master latch is a level-sensitive D latch controlled by the clock and accepts the input when the clock is low and latches the input when the clock transitions to high state. However, Slave latch is controlled by the inverted clock signal and becomes transparent when the clock is high, latching the output of the master latch.



Master-slave D Flip-Flop [3]

When CLK = 0, the input D is propagated to the master latch output Q_m and the slave latch is in the latched state, so the final output Q remains unchanged. On the contrary when CLK

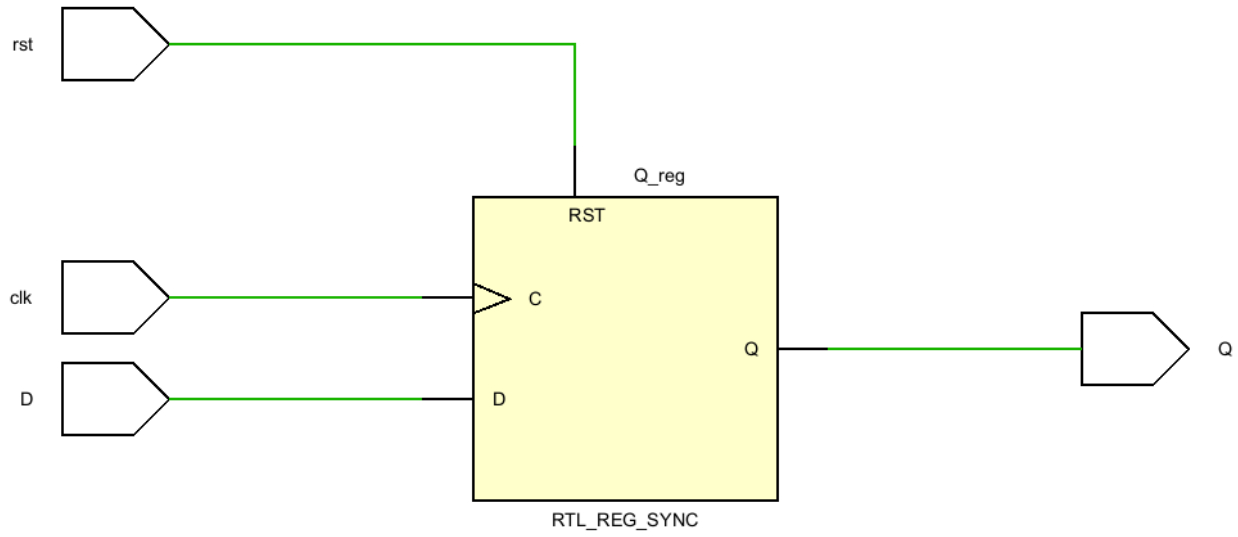
= 1, the master latch is in the latched state, so that Q_m stays the same and the slave latch propagates Q_m to the final output Q.

In an edge-sensitive circuit, the output changes only on the edge of a clock signal, either the rising edge (low-to-high transition) or the falling edge (high-to-low transition). In Edge-Triggered D Flip-Flops the input D is sampled and stored only at the clock edge (e.g., rising edge for a positive-edge-triggered D flip-flop). This provides more precise control of data timing, making it ideal for synchronous systems and minimizes the risk of glitches caused by noise or unintended input transitions.[3]

In a level-sensitive circuit, the output changes continuously if the clock signal is active (high or low). A latch is level sensitive. When the enable signal (or clock) is active, the input propagates to the output immediately. When the enable signal is inactive, the latch holds its previous state. This provides a simpler design and faster response compared to edge-sensitive circuits. However, it has disadvantages over edge-sensitive design like less robust in synchronous designs because inputs can inadvertently propagate to the output multiple times within one clock cycle.[3]

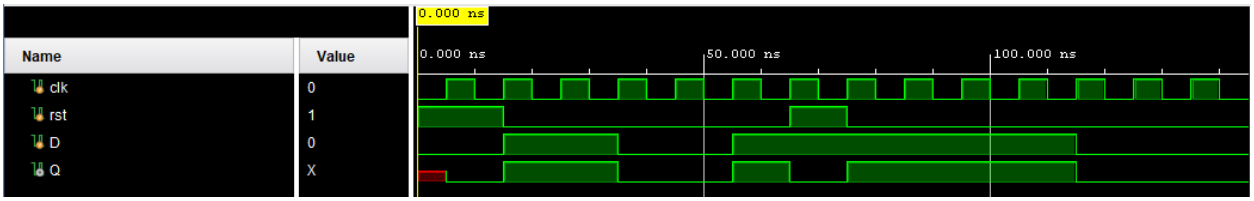
Due to such differences edge-triggered D Flip-Flops are commonly used in applications such as registers, counters, Finite State Machines (FSM), pipelined system design, etc.

DFF with Synchronous Reset:



D Flip-Flop with Synched Reset RTL Schematic

Yusuf Tekin
040200043



D Flip-Flop with Synched Reset Behavioral Simulation

```
`timescale 1ns / 1ps

module DFF_sync(
    input clk,
    input rst,
    input D,
    output reg Q
    );

always @(posedge clk) begin
    if(rst) Q <= 1'b0;
    else Q <= D;
end

endmodule
```

D Flip-Flop Synched Reset Verilog Code

```
`timescale 1ns / 1ps

module DFF_sync_tb;

    reg clk, rst, D;
    wire Q;

    DFF_sync uut (
        .clk(clk),
        .rst(rst),
        .D(D),
        .Q(Q)
    );

    always begin
        clk = 1'b0; #5;
        clk = 1'b1; #5;
    end

    initial begin
        rst = 1'b0;
        D = 1'b0;

        rst = 1'b1; #15;
        rst = 1'b0;

        D = 1'b1; #20;
        D = 1'b0; #20;

        D = 1'b1; #10;
        rst = 1'b1; #10;
        rst = 1'b0; #10;

        D = 1'b1; #30;
        D = 1'b0; #30;
        $finish();
    end
endmodule
```

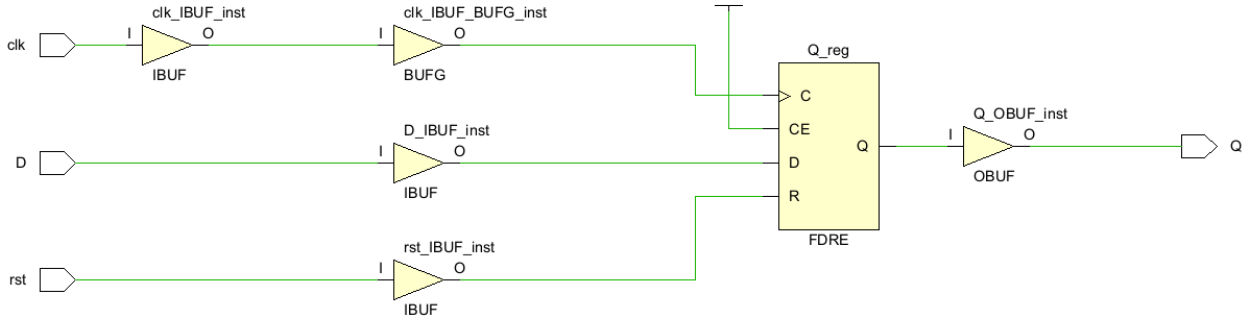
D Flip-Flop Synched Reset Testbench Code

Primitives

Ref Name	Used	Functional Category
IBUF	3	IO
OBUF	1	IO
FDRE	1	Flop & Latch
BUFG	1	Clock

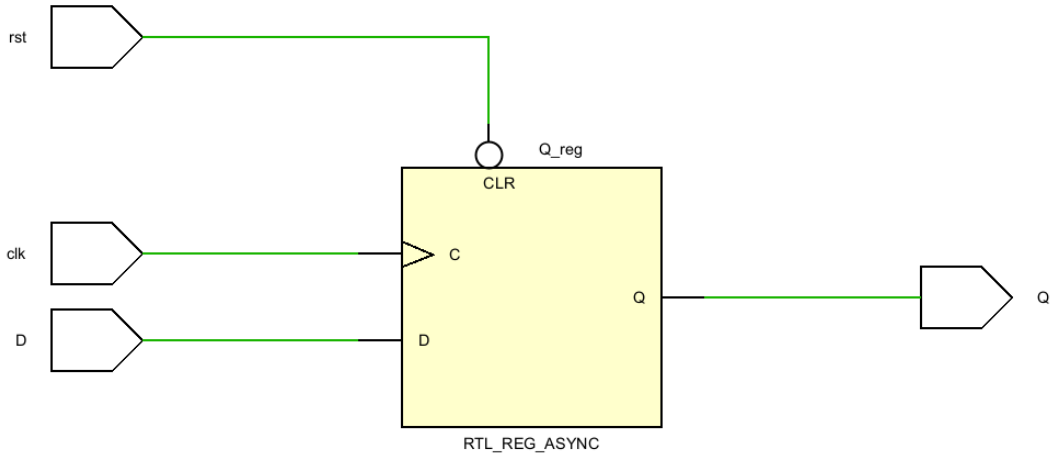
D Flip-Flop Synched Reset Primitive Usages

Yusuf Tekin
040200043



D Flip-Flop Synched Reset Technoogy Schematic

DFF with Asynchronous Reset:

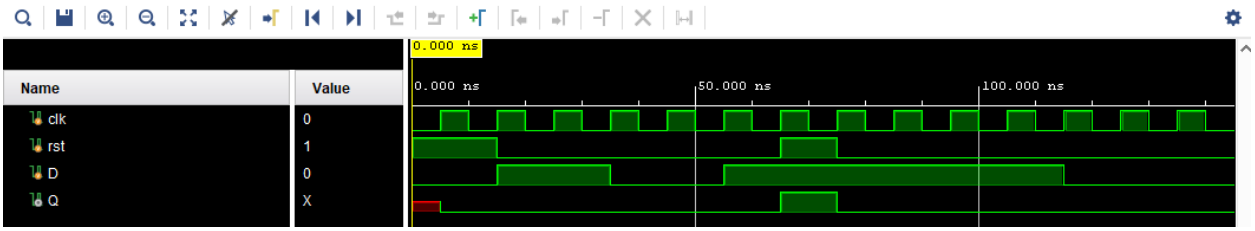


D Flip-Flop Asynchronous Reset RTL Schematic

```
module DFF_async(

    input clk,
    input rst,
    input D,
    output reg Q
    );

always @(posedge clk or negedge rst) begin
    if(!rst) Q <= 1'b0;
    else Q <= D;
end
endmodule
```

D Flip-Flop Asynchronous Reset Verilog Code

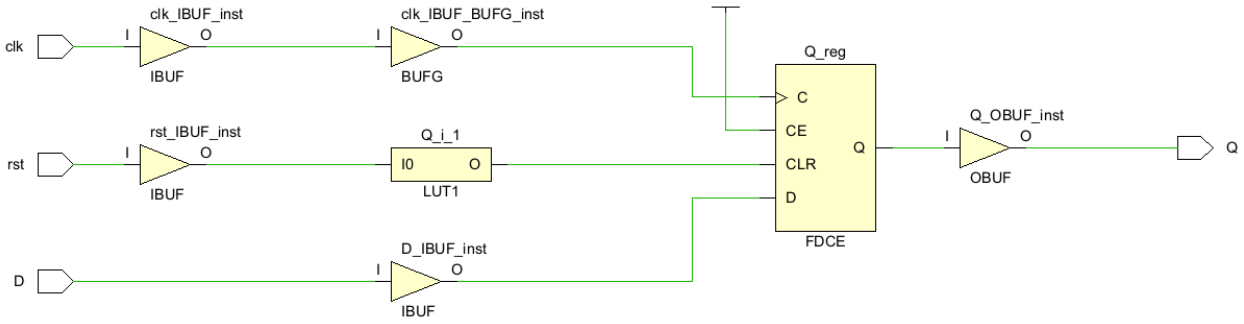
Yusuf Tekin
040200043



D Flip-Flop Asynchronous Reset Behavioral Simulation

```
`timescale 1ns / 1ps

module DFF_async_tb;

    reg clk, rst, D;
    wire Q;

    DFF_async uut (
        .clk(clk),
        .rst(rst),
        .D(D),
        .Q(Q)
    );

    always begin
        clk = 1'b0; #5;
        clk = 1'b1; #5;
    end

    initial begin
        rst = 1'b0;
        D = 1'b0;

        rst = 1'b1; #15;
        rst = 1'b0;

        D = 1'b1; #20;
        D = 1'b0; #20;

        D = 1'b1; #10;
        rst = 1'b1; #10;
        rst = 1'b0; #10;

        D = 1'b1; #30;
        D = 1'b0; #30;
        $finish();
    end
endmodule
```

D Flip-Flop Asynchronous Reset Testbench Code



D Flip-Flop Asynchronous Reset Technology Schematic

Primitives

Ref Name	Used	Functional Category
IBUF	3	IO
OBUF	1	IO
LUT1	1	LUT
FDCE	1	Flop & Latch
BUFG	1	Clock

D Flip-Flop Asynchronous Reset Primitive Usages

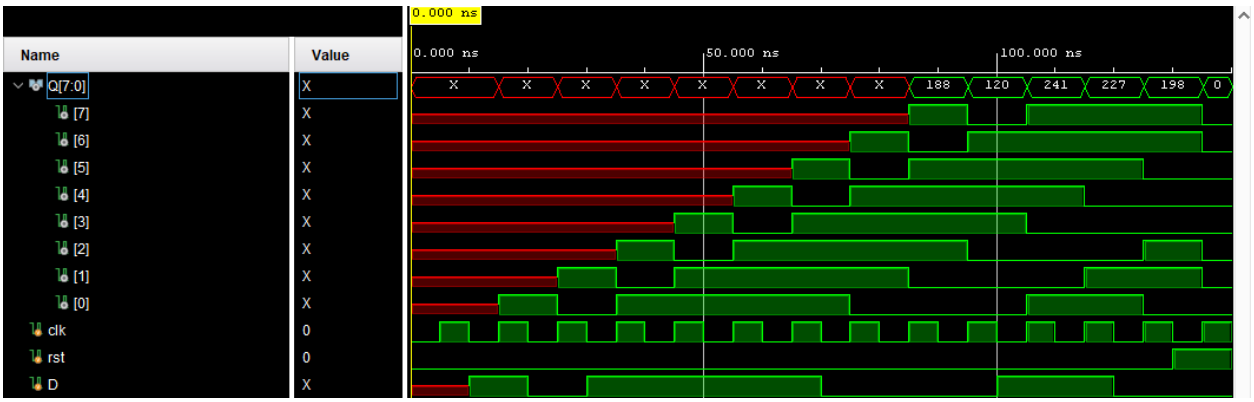
In the asynchronous design, there is an extra LUT1 used for the design. This is because the asynchronous design requires to be affected by “reset” input independent from clock. A LUT1 is a simple 1-input LUT (look-up table), often used to directly implement a constant or simple logic, such as forcing Q = 0 when rst = 1. However, in a synchronous flip-flop, the reset signal is handled within the clocked data path of the flip-flop itself. It is combined with the data (D) input using internal gates and does not require extra external logic (e.g., a LUT).

Yusuf Tekin
040200043

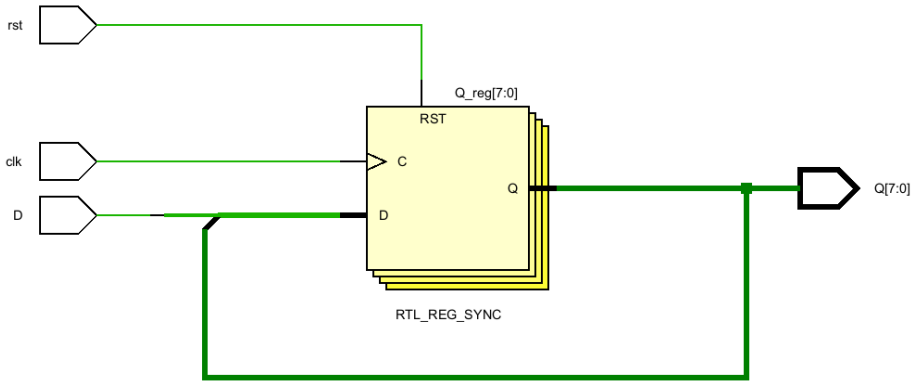
2. 8-bit Shift Register

```
`timescale 1ns / 1ps

module shift8(
    input clk,
    input rst,
    input D,
    output reg [7:0] Q
    );

always @(posedge clk) begin
    if(rst) Q <= 8'b0;
    else begin
        Q <= {Q[6:0], D};
    end
end
endmodule
```

8-bit Shift Register Verilog Code

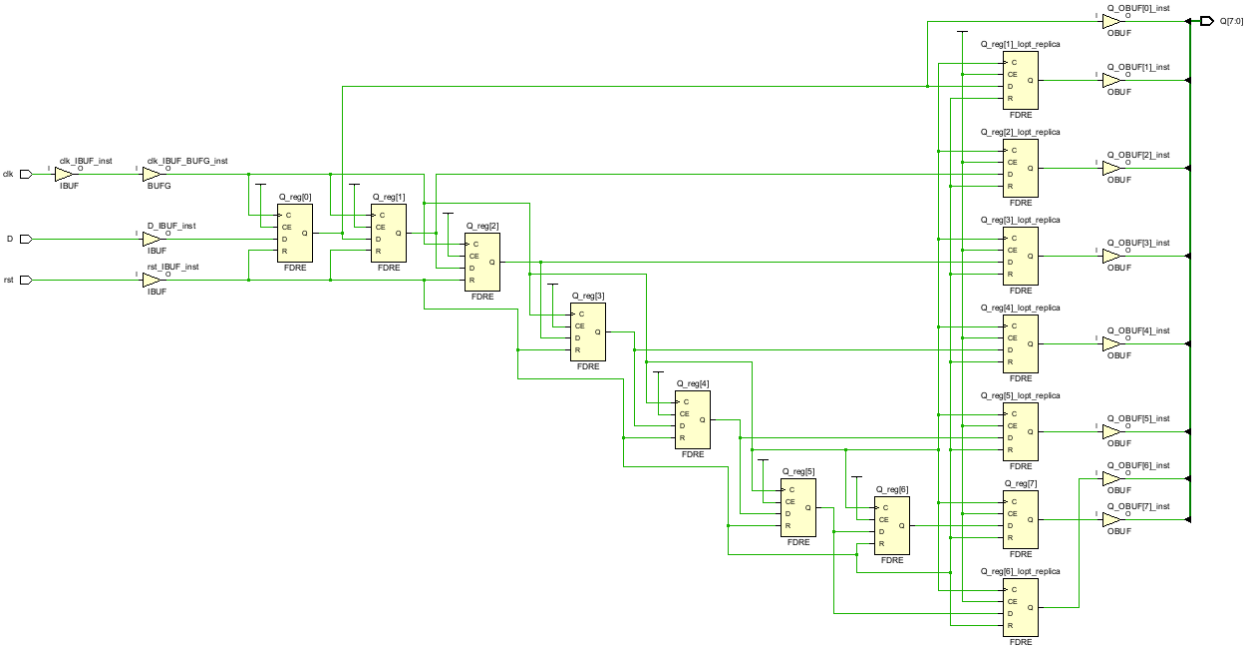


8-bit Shift Register Behavioral Simulation



8-bit Shift Register RTL Schematic

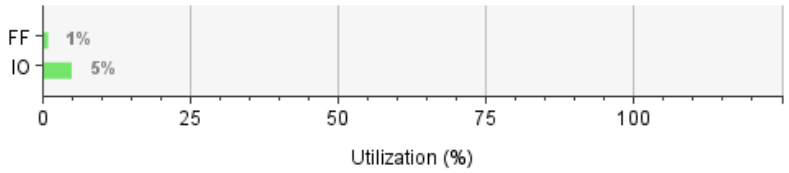
Yusuf Tekin
040200043



8-bit Shift Register Technology Schematic

Summary

Resource	Utilization	Available	Utilization %
FF	14	65200	0.02
IO	11	210	5.24



8-bit Shift Register Utilization Summary

Primitives

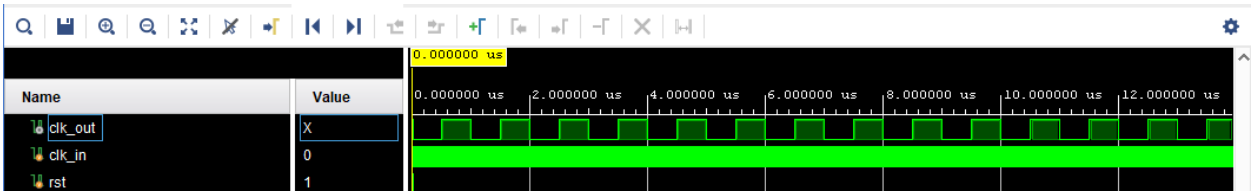
Ref Name	Used	Functional Category
FDRE	14	Flop & Latch
OBUF	8	IO
IBUF	3	IO
BUFG	1	Clock

8-bit Shift Register Primitives

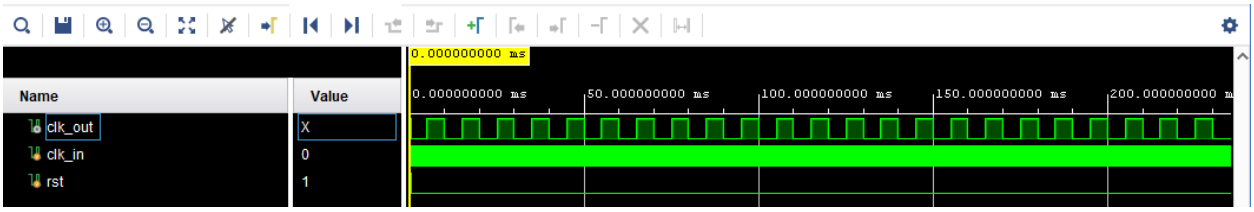
3. Clock Divider with Stopwatch Example

```
`timescale 1ns / 1ps

module clk_divider #(parameter [27:0] CLK_DIV = 100)(
    input clk_in,
    input rst,
    output reg clk_out
    );

reg [27:0] counter;

always @(posedge clk_in) begin
    if(rst) begin
        counter <= 28'b0;
        clk_out <= 1'b0;
    end
    else begin
        if(counter == CLK_DIV - 1) begin
            counter <= 28'b0;
            clk_out <= ~clk_out;
        end else begin
            counter <= counter + 1;
        end
    end
end
endmodule
```

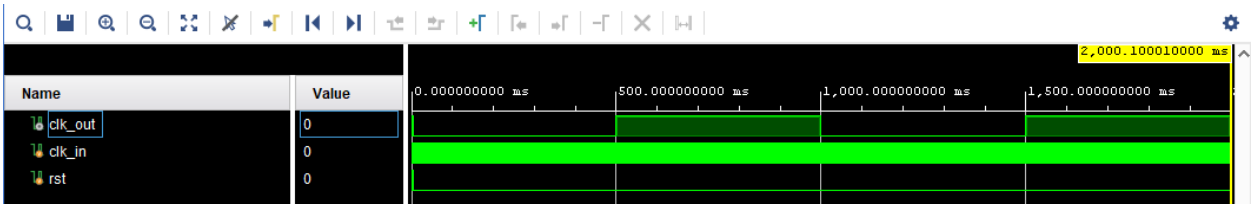
Clock Divider Verilog Code



Clock Divider Behavioral Simulation for 1 MHz



Clock Divider Behavioral Simulation for 100 Hz

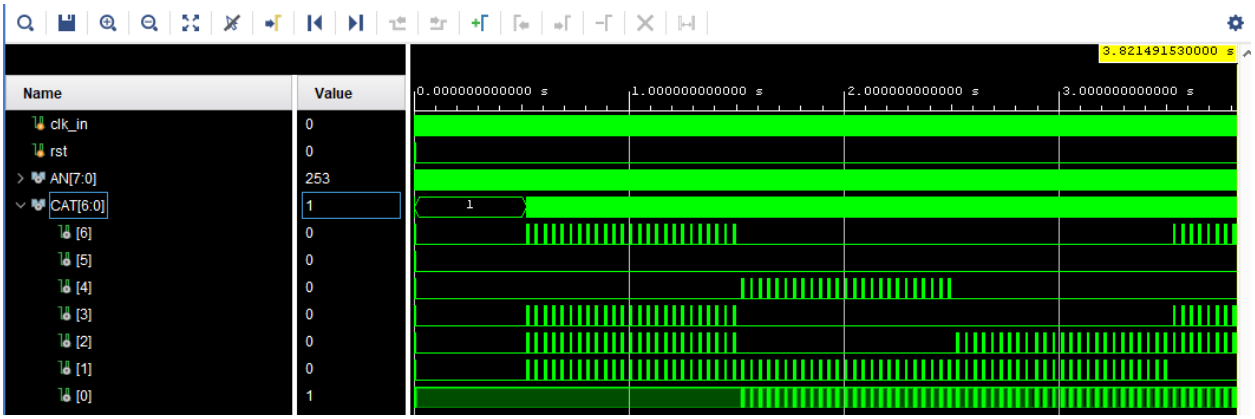


Clock Divider Behavioral Simulation for 1 Hz

Yusuf Tekin
040200043

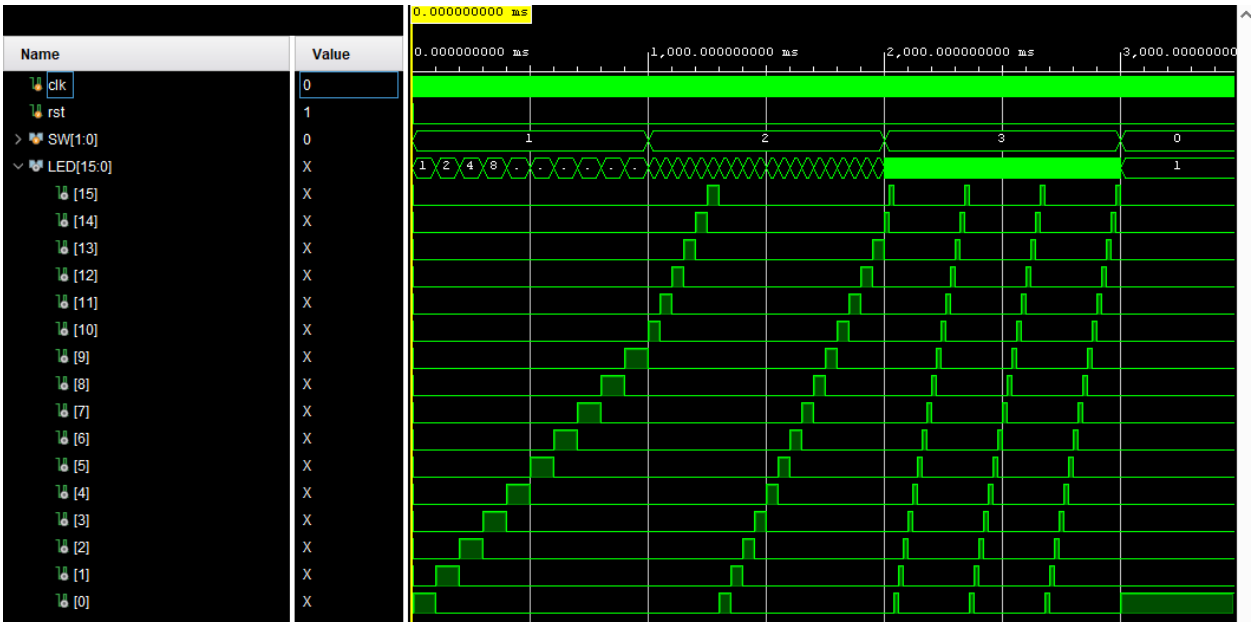
```
`timescale 1ns / 1ps

module stopwatch(
    input             clk_in,
    input             rst,
    output reg [7:0]  AN,       // Anode  : select 7-segment displays
    output reg [6:0]  CAT       // Catode : select 7-segment LEDs
);
    wire        clk_out1;              // 1sec clk
    wire        clk_out100;            // 10msec clk (100Hz)

    reg  [13:0] cnt;                    // 1sec timer
    wire [15:0] cnt_bcd;                // BCD converted
    reg  [3:0]  display;                // display number
    reg  [1:0]  refresh_cnt;            // refresh counter

    clk_divider #( .CLK_DIV(50000000) )
    CLKDIV1(
        .clk_in (clk_in),
        .rst    (rst),
        .clk_out(clk_out1)
    );
       // CLK DIVIDER 100Hz
    clk_divider #( .CLK_DIV(500000) )
    CLKDIV100(
        .clk_in (clk_in),
        .rst    (rst),
        .clk_out(clk_out100)
    );

    // Binary to Decimal Conversion
    bin2bcd BIN2BCD(
        .bin(cnt),
        .bcd(cnt_bcd)
    );
    // Timer 1sec
    always @(posedge clk_out1, posedge rst)
    begin
        if( rst )
        begin
            cnt     <= 0;
        end
        else
        begin
            if( cnt == 9999 )
                cnt <= 0;
            else
                cnt <= cnt + 1;
        end
    end
    // Refresh 7-segment displays
    always @(posedge clk_out100, posedge rst)
    begin
        if( rst )
        begin
            AN          <= 8'b1111_0000;
            display     <= 4'b0000;
            refresh_cnt <= 0;
        end
        else
        begin
            case( refresh_cnt )
                2'b00: begin
                    AN          <= 8'b1111_1110;
                    display     <= cnt_bcd[3:0];
                    refresh_cnt <= refresh_cnt + 1;
                end
                2'b01: begin
                    AN = 8'b1111_1101;
                    display     <= cnt_bcd[7:4];
                    refresh_cnt <= refresh_cnt + 1;
                end
                2'b10: begin
                    AN = 8'b1111_1011;
                    display     <= cnt_bcd[11:8];
                    refresh_cnt <= refresh_cnt + 1;
                end
                2'b11: begin
                    AN = 8'b1111_0111;
                    display     <= cnt_bcd[15:12];
                    refresh_cnt <= refresh_cnt + 1;
                end
            endcase
       end
    end
    // Select 7-segment LEDs
    always @(*)
    begin
         case(display)
             4'b0000: CAT = 7'b0000001; // "0"
             4'b0001: CAT = 7'b1001111; // "1"
             4'b0010: CAT = 7'b0010010; // "2"
             4'b0011: CAT = 7'b0000110; // "3"
             4'b0100: CAT = 7'b1001100; // "4"
             4'b0101: CAT = 7'b0100100; // "5"
             4'b0110: CAT = 7'b0100000; // "6"
             4'b0111: CAT = 7'b0001111; // "7"
             4'b1000: CAT = 7'b0000000; // "8"
             4'b1001: CAT = 7'b0000100; // "9"
             default: CAT = 7'b0000001; // "0"
         endcase
    end
endmodule
```

Stopwatch Verilog Code



Stopwatch Behavioral Simulation

4. Sliding LEDs



Sliding LEDs Behavioral Simulation

Yusuf Tekin
040200043

```
`timescale 1ns / 1ps

module sliding_leds #(parameter [23:0] CLK_DIV = 10000000)(
    input clk,
    input rst,
    input [1:0] SW,
    output reg [15:0] LED
    );

reg [23:0] counter;
reg [23:0] desired_rate;

always @(posedge clk) begin
    if(rst) begin
        counter <= 24'b0;
        LED <= 16'b0000_0000_0000_0001;
    end else begin
        case(SW)
            2'b00: desired_rate <= 24'b0;
            2'b01: desired_rate <= CLK_DIV;        //10 Hz
            2'b10: desired_rate <= CLK_DIV / 2;    //20 Hz
            2'b11: desired_rate <= CLK_DIV / 5;    //50 Hz
            default: desired_rate <= 24'b0;
        endcase

        if(desired_rate != 24'b0) begin
            if(counter == desired_rate - 1) begin
                counter <= 24'b0;
                LED <= {LED[14:0],LED[15]};
            end else begin
                counter <= counter + 1;
            end
        end
    end
end
endmodule
```

Sliding LEDs Verilog Code

Unconstrained Paths - sys_clk_pin - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 41	∞	1	1	1	LED_reg[14]_lopt_replica/C	LED[14]	6.712	4.216	2.496	∞	sys_clk_pin			0.025
Path 42	∞	1	1	1	LED_reg[0]_lopt_replica/C	LED[0]	6.682	3.980	2.702	∞	sys_clk_pin			0.025
Path 43	∞	1	1	1	LED_reg[2]_lopt_replica/C	LED[2]	6.624	3.989	2.635	∞	sys_clk_pin			0.025
Path 44	∞	1	1	1	LED_reg[13]_lopt_replica/C	LED[13]	6.618	4.052	2.565	∞	sys_clk_pin			0.025
Path 45	∞	1	1	1	LED_reg[15]_lopt_replica/C	LED[15]	6.558	4.066	2.493	∞	sys_clk_pin			0.025
Path 46	∞	1	1	1	LED_reg[10]_lopt_replica/C	LED[10]	6.493	3.991	2.502	∞	sys_clk_pin			0.025
Path 47	∞	1	1	1	LED_reg[5]_lopt_replica/C	LED[5]	6.354	4.053	2.301	∞	sys_clk_pin			0.025
Path 48	∞	1	1	1	LED_reg[8]_lopt_replica/C	LED[8]	6.283	4.002	2.281	∞	sys_clk_pin			0.025
Path 49	∞	1	1	1	LED_reg[1]_lopt_replica/C	LED[1]	6.261	3.999	2.262	∞	sys_clk_pin			0.025
Path 50	∞	1	1	1	LED_reg[11]_lopt_replica/C	LED[11]	6.229	3.989	2.240	∞	sys_clk_pin			0.025

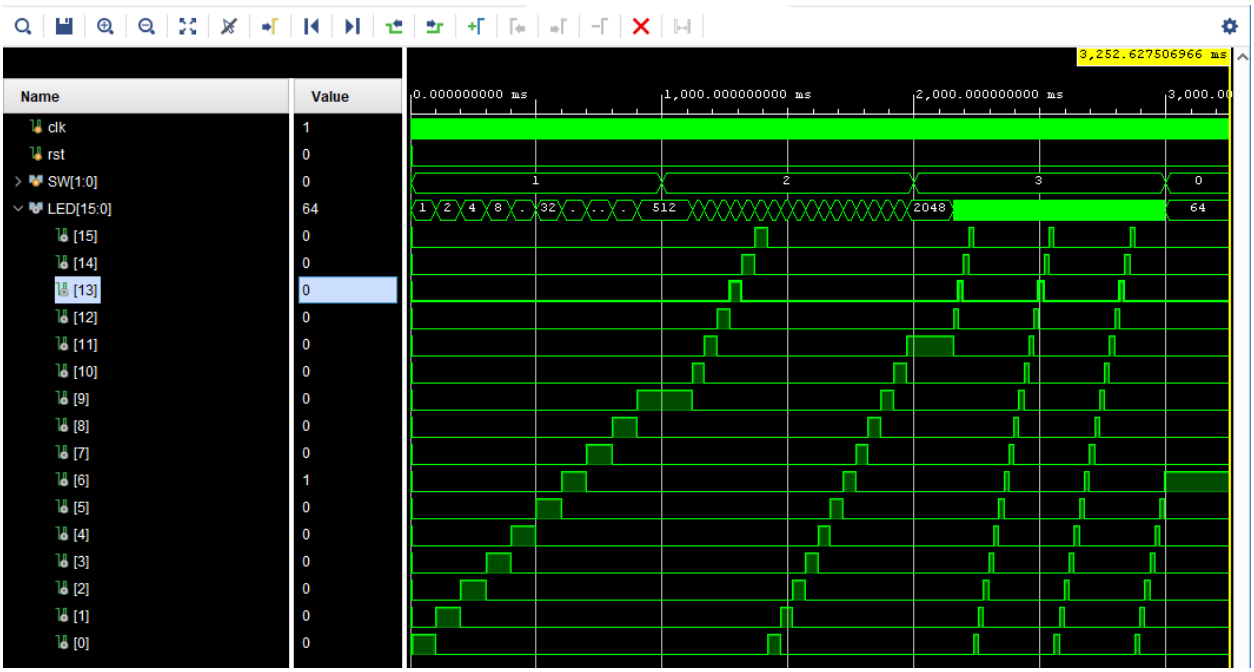
clk_pin Path Delays Setup

Yusuf Tekin
040200043

Unconstrained Paths - sys_clk_pin - NONE - Hold

Name	Slack	Levels	Routes	High Fanout	From	To	Total ...	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 60	∞	1	1	1	LED_reg[8]_lopt_replica/C	LED[8]	2.014	1.388	0.626	-∞	sys_clk_pin			0.025
Path 59	∞	1	1	1	LED_reg[5]_lopt_replica/C	LED[5]	2.001	1.400	0.601	-∞	sys_clk_pin			0.025
Path 58	∞	1	1	1	LED_reg[1]_lopt_replica/C	LED[1]	1.979	1.385	0.594	-∞	sys_clk_pin			0.025
Path 57	∞	1	1	1	LED_reg[11]_lopt_replica/C	LED[11]	1.954	1.375	0.579	-∞	sys_clk_pin			0.025
Path 55	∞	1	1	1	LED_reg[7]_lopt_replica/C	LED[7]	1.939	1.379	0.560	-∞	sys_clk_pin			0.025
Path 56	∞	1	1	1	LED_reg[12]_lopt_replica/C	LED[12]	1.938	1.412	0.526	-∞	sys_clk_pin			0.025
Path 54	∞	1	1	1	LED_reg[9]_lopt_replica/C	LED[9]	1.929	1.438	0.491	-∞	sys_clk_pin			0.025
Path 53	∞	1	1	1	LED_reg[6]_lopt_replica/C	LED[6]	1.928	1.381	0.546	-∞	sys_clk_pin			0.025
Path 52	∞	1	1	1	LED_reg[3]_lopt_replica/C	LED[3]	1.759	1.368	0.391	-∞	sys_clk_pin			0.025
Path 51	∞	1	1	1	LED_reg[4]_lopt_replica/C	LED[4]	1.739	1.386	0.353	-∞	sys_clk_pin			0.025

clk_pin Path Delays Setup

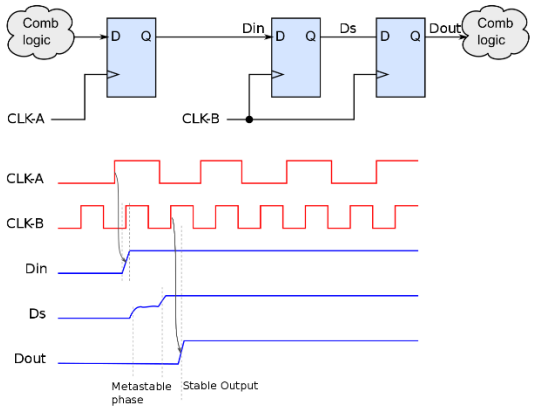


Post-Implementation Timing Simulation

5. Research

Static Timing Analysis (STA) is a technique used to validate the timing of a digital circuit without requiring simulation. It calculates the propagation delays through combinational paths and verifies that setup and hold time constraints are satisfied for all sequential elements, such as flip-flops and latches. Setup time is the minimum period before a clock edge during which the input data must remain stable to be correctly captured. Conversely, hold time refers to the minimum period after the clock edge that data must remain stable to ensure proper latching. Violating these constraints can lead to errors, making STA critical for ensuring reliable operation. The maximum clock frequency of a sequential circuit is determined by the total propagation delay, clock-to-Q delay, setup time, and any clock skew. This frequency is calculated using fmax=1/Tclockf_{max} = 1 / T_{clock}fmax=1/Tclock, where TclockT_{clock}Tclock includes all timing delays.[5]

Metastability occurs when a flip-flop enters an unstable state due to a violation of setup or hold times. This situation often arises when asynchronous signals or signals from different clock domains interact with the clocked system. In a metastable state, the flip-flop's output hovers unpredictably between logic levels, potentially causing incorrect behavior. To prevent metastability, designers use techniques such as double-flopping, where a chain of flip-flops stabilizes asynchronous inputs. Proper timing margins, slower clock speeds, and encoding multi-bit signals with Gray Code further reduce the likelihood of metastable states. Additionally, asynchronous FIFO designs are effective for managing clock domain crossings.[4]



Metastability Figure[4]

References

1. https://electronics-course.com/sr-nand-latch
2. https://evertutorial.com/articles/DigitalDesign/Gated_D_Latch
3. Brown, S. D., & Vranesic, Z. G. (2013). Fundamentals of Digital Logic with Verilog Design. (p. 247-331).
4. https://en.wikipedia.org/wiki/Metastability_(electronics)#:~:text=In%20electronics%2C%20metastability%20is%20the,unstable%20equilibrium%20or%20metastable%20state.
5. ChatGPT