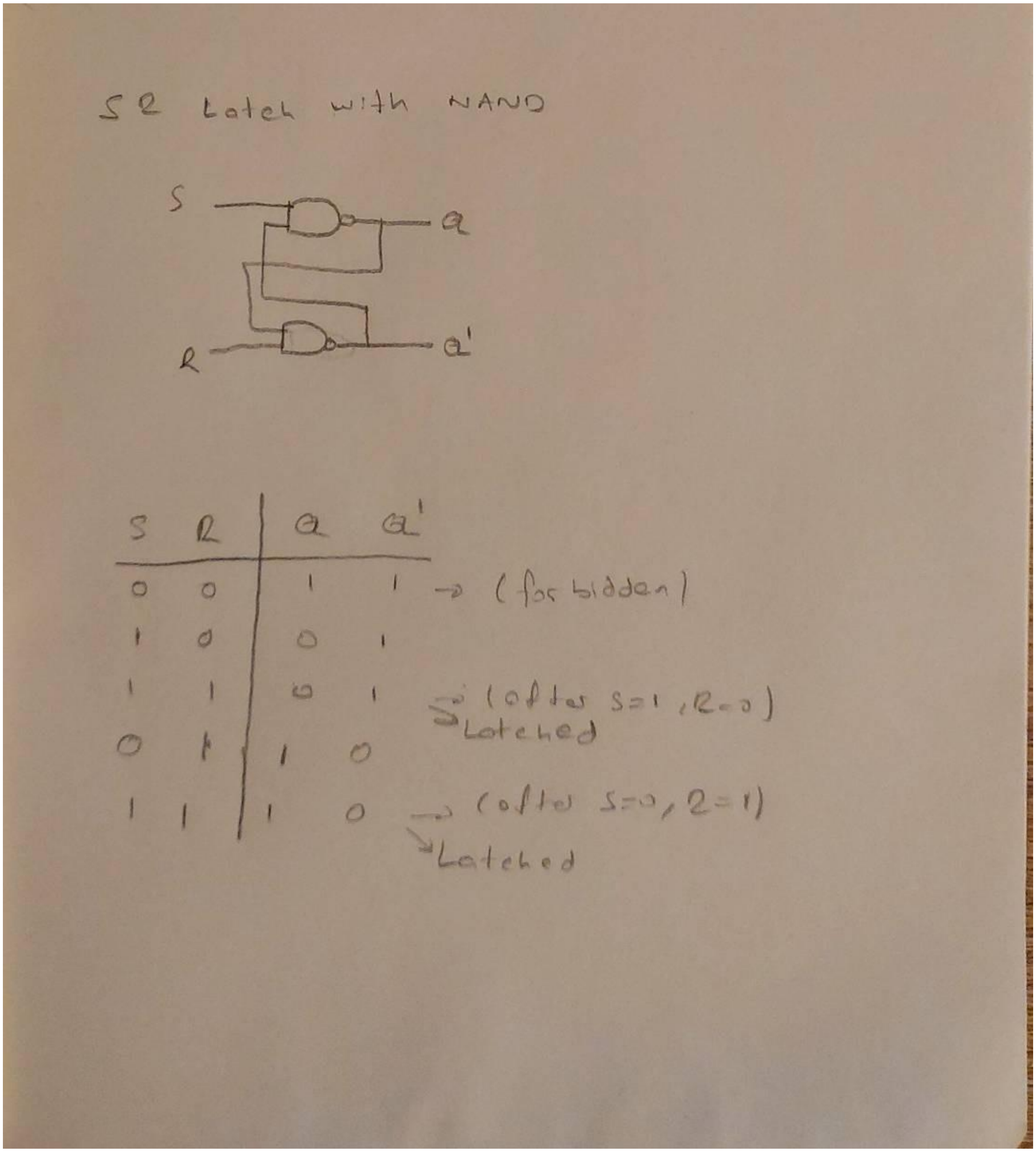
Ömer Faruk SERT

040170144

1-) SR LATCH

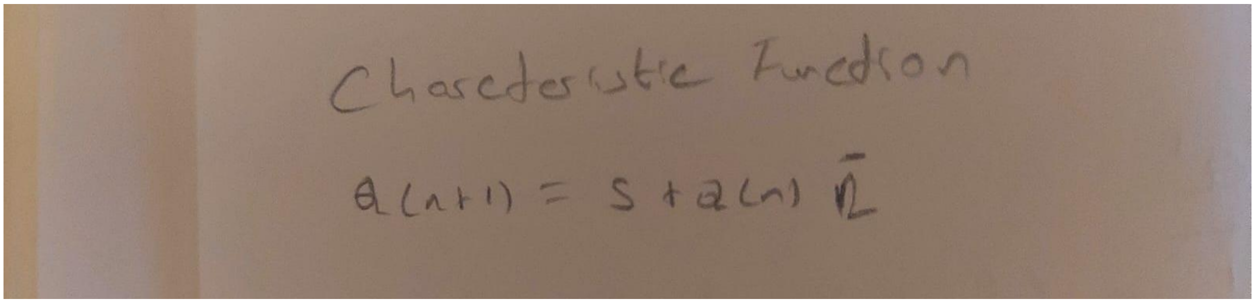
Circuit Diagram and Truth table of SR latch



SR Latch with NAND

S	R	Q	Q'	
0	0	1	1	→ (forbidden)
1	0	0	1	
1	1	0	1	→ (after S=1, R=0) ↘ Latched
0	1	1	0	
1	1	1	0	→ (after S=0, R=1) ↘ Latched

Characteristic Function of SR latch



Charecteristic Function

$Q(n+1) = S + Q(n)\bar{R}$

Verilog Code

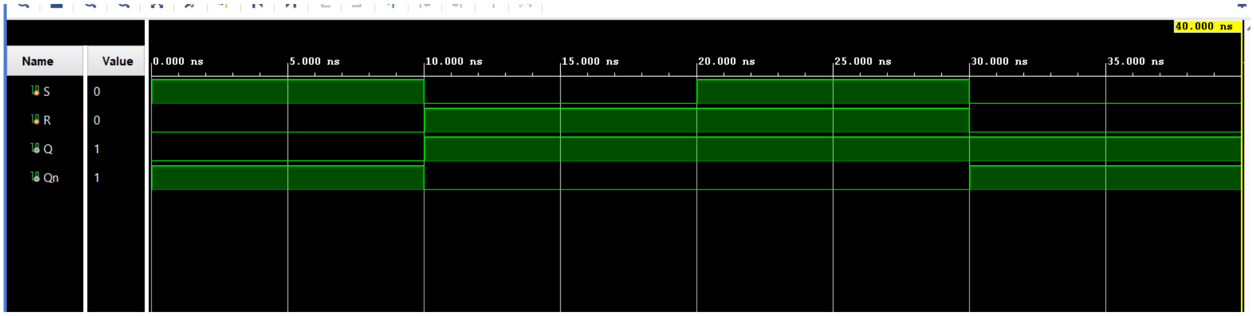
Module Code

```
`timescale 1ns / 1ps

module SR_Latch (
    input S,R,
    output Q,Qn
);
    wire [1:0]OX;
    NAND nand1(.I1(S),.I2(OX[1]),.O(OX[0]));
    NAND nand2(.I1(R),.I2(OX[0]),.O(OX[1]));

    assign Q=OX[0];
    assign Qn=OX[1];
endmodule
```

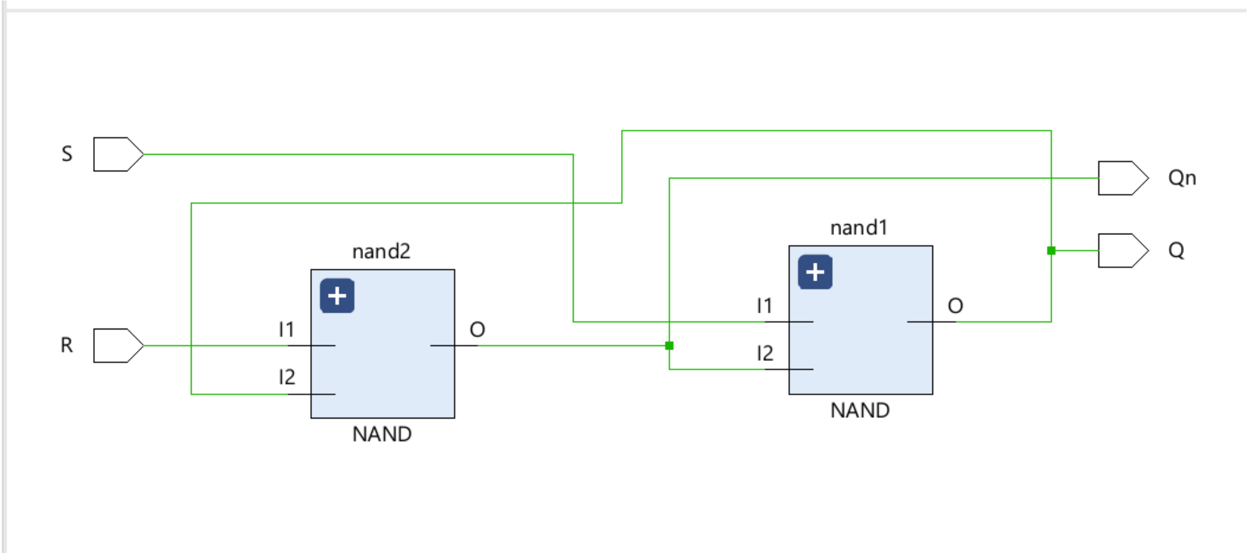
Testbench Code

```
`timescale 1ns / 1ps



module experiment5tb;

    reg S;
    reg R;
    wire Q,Qn;

    SR_Latch uut(.S(S),.R(R),.Q(Q),.Qn(Qn));

    initial
    begin

    R=1'b0;
    S=1'b1;
    #10

    R=1'b1;
    S=1'b0;
    #10

    R=1'b1;
    S=1'b1;
    #10


    R=1'b0;
    S=1'b0;
    #10
    $finish;
    end
endmodule
```

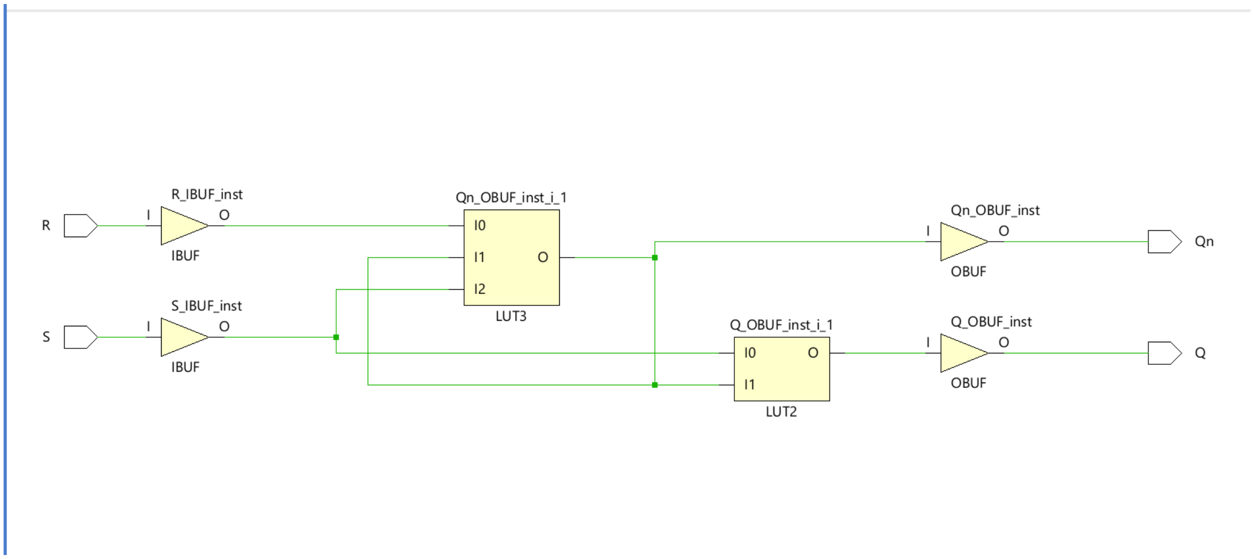
Simulation Wave



RTL Schematic

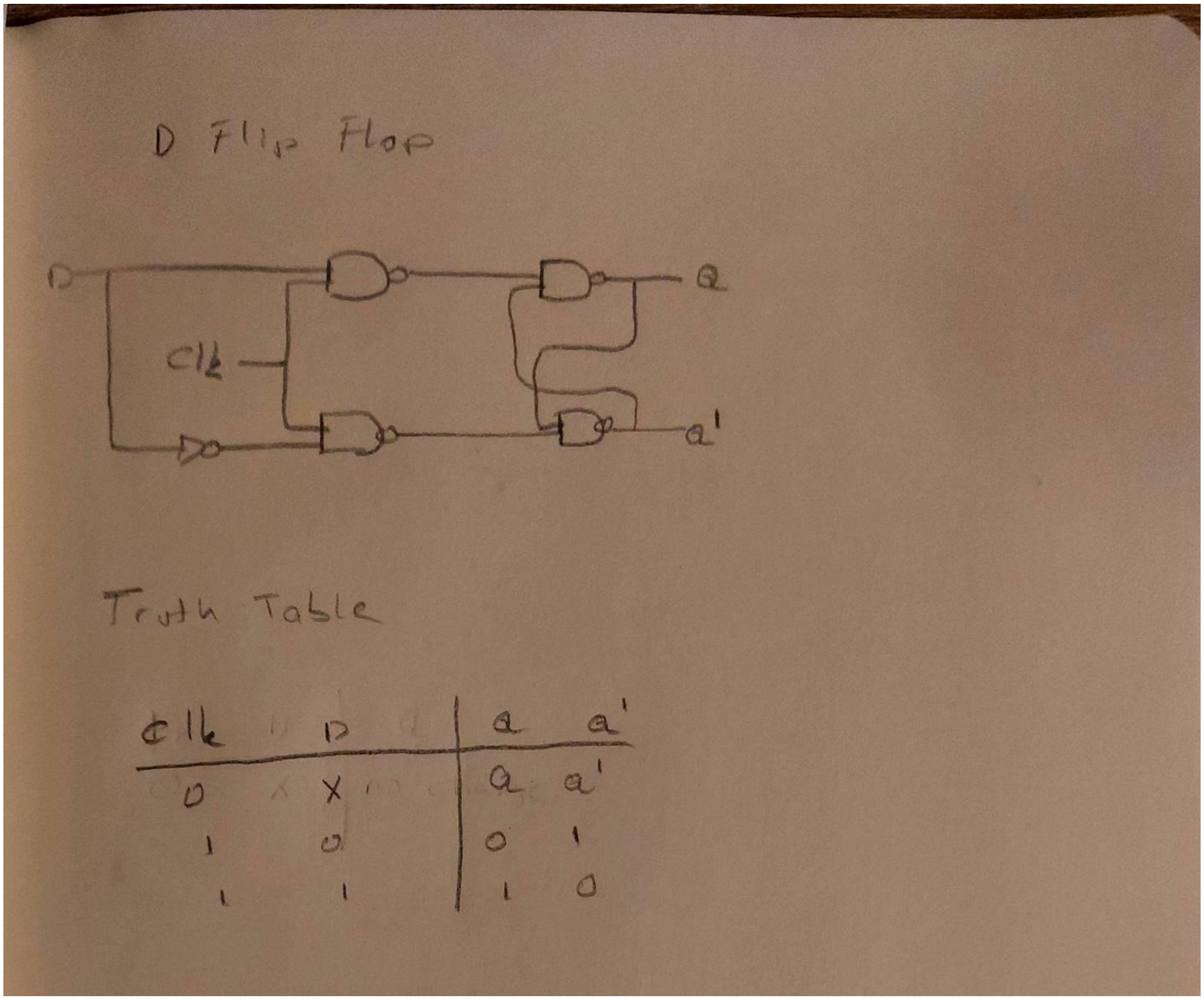


Technology Schematic



2-) D Flip Flop

Circuit Diagram and Truth Table of D Flip Flop



D Flip Flop

Truth Table

Clk	D	Q	Q'
0	X	Q	Q'
1	0	0	1
1	1	1	0

Verilog Code

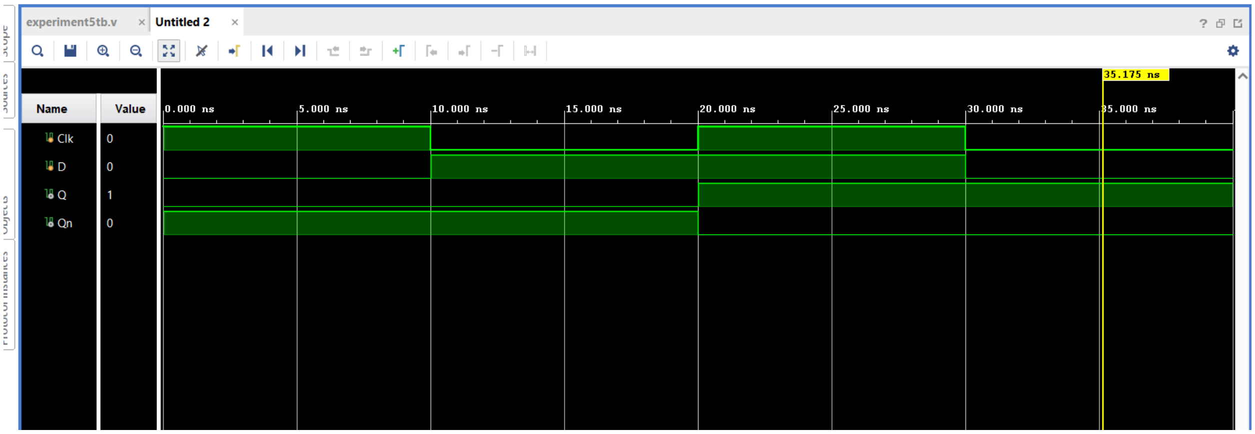
Module Code

```
`timescale 1ns / 1ps

module DFF (
    input D,Clk,
    output Q,Qn
);
    wire [4:0]OX;
    NOT not1(.I(D),.O(OX[0])); //Dnot

    NAND nand1(.I1(D),.I2(Clk),.O(OX[1]));
    NAND nand2(.I1(OX[1]),.I2(OX[2]),.O(OX[3])); //OX[2]=Qn , OX[3]=Q

    NAND nand3(.I1(OX[0]),.I2(Clk),.O(OX[4]));
    NAND nand4(.I1(OX[3]),.I2(OX[4]),.O(OX[2]));

    assign Qn=OX[2];
    assign Q=OX[3];

endmodule
```

Testbench Code

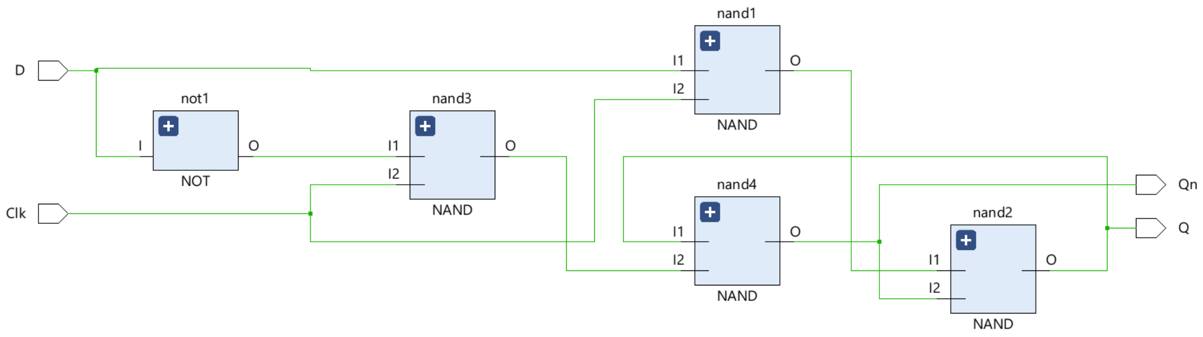
```
`timescale 1ns / 1ps



module experiment5tb;

    reg Clk;
    reg D;
    wire Q,Qn;

    DFF uut(.D(D),.Clk(Clk),.Q(Q),.Qn(Qn));

    initial
    begin

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

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

    D=1'b1;
    Clk=1'b1;
    #10
    D=1'b0;
    Clk=1'b0;
    #10
    $finish;
    end
endmodule
```

Simulation Waveform

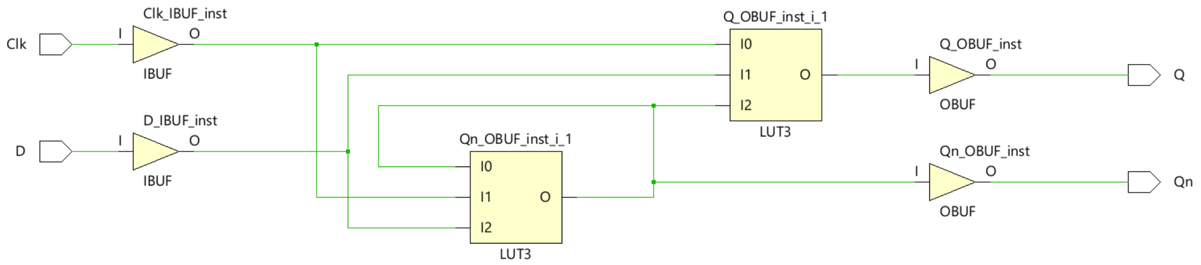


Name	Value
Clk	0
D	0
Q	1
Qn	0

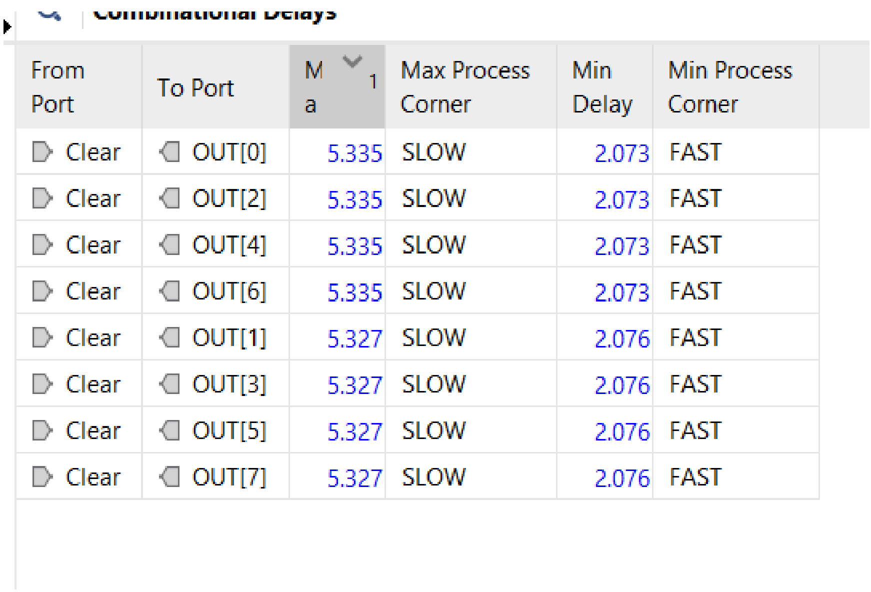
RTL Schematic



Technology Schematic



Combinationa Delays



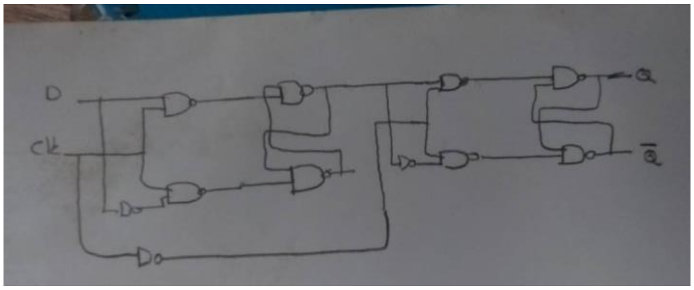
Combinational Delays

From Port	To Port	Ma 1	Max Process Corner	Min Delay	Min Process Corner
Clear	OUT[0]	5.335	SLOW	2.073	FAST
Clear	OUT[2]	5.335	SLOW	2.073	FAST
Clear	OUT[4]	5.335	SLOW	2.073	FAST
Clear	OUT[6]	5.335	SLOW	2.073	FAST
Clear	OUT[1]	5.327	SLOW	2.076	FAST
Clear	OUT[3]	5.327	SLOW	2.076	FAST
Clear	OUT[5]	5.327	SLOW	2.076	FAST
Clear	OUT[7]	5.327	SLOW	2.076	FAST

187.4414 MHz is the maximum limit of clock frequency because the longest delay is 5.335ns, so from F=1/T we can derive the maximum clock frequency.

3-) Master Slave D Flip Flop

Circuit Diagram

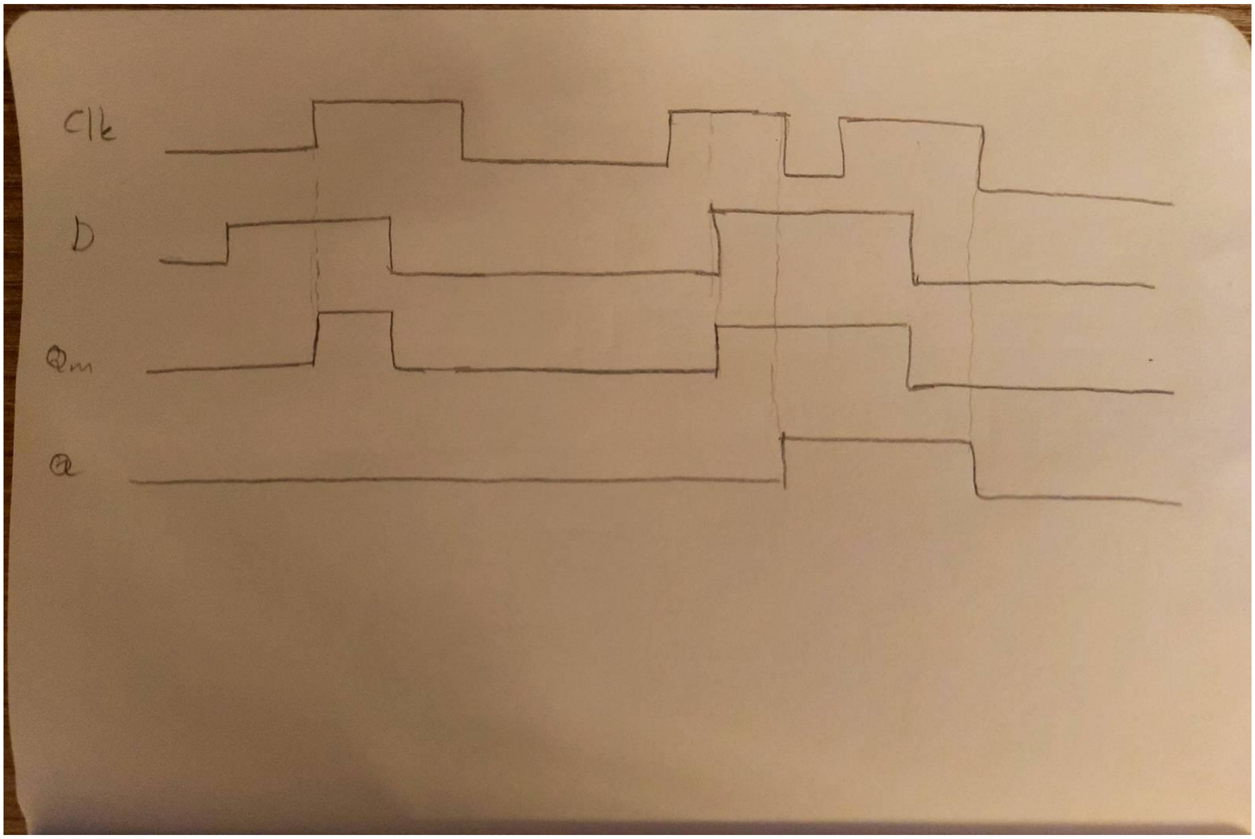


Explanation of how Master Slave D Flip Flop works

Normally a D flip flop would direct the input to the output when clock is high. When there is 2 D flip flop with one of them is connected to not clock, while clock is high it takes the input and stores in first flip flops output and after clock became low it holds in the second flip flop also.

At negating edge output will change because the output comes from slave latch, and it is active when clock is low.

Hand Drawn Simulation



Verilog Code

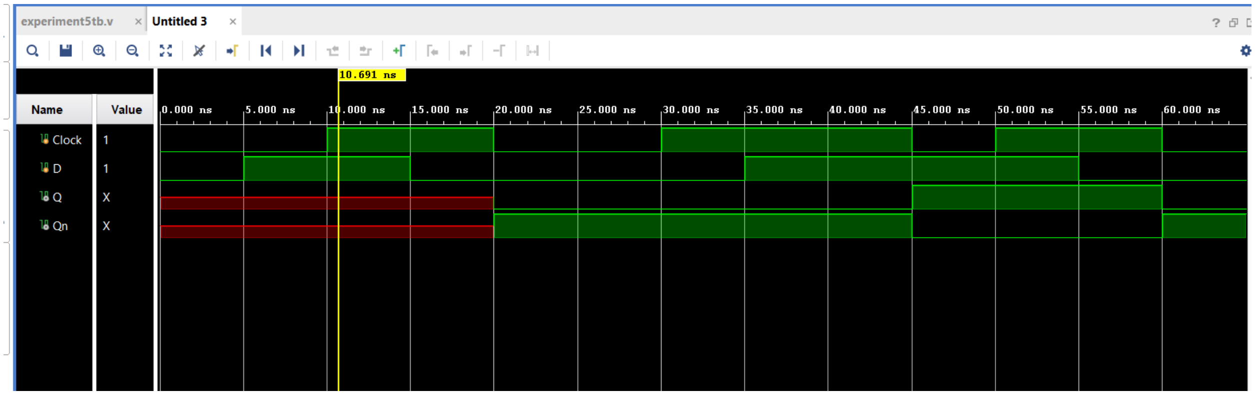
Module Code

```
`timescale 1ns / 1ps
module MSDF (
    input D,Clock,
    output Q,Qn
);
    wire [2:0]OX;
    wire Qm;

    NOT not1(.I(Clock),.O(OX[0])); //Clocknot
    DFF dff1(.D(D),.Clk(Clock),.Q(Qm));
    DFF dff2(.D(Qm),.Clk(OX[0]),.Q(OX[1]),.Qn(OX[2]));
    assign Q=OX[1];
    assign Qn=OX[2];


endmodule
```

Testbench Code

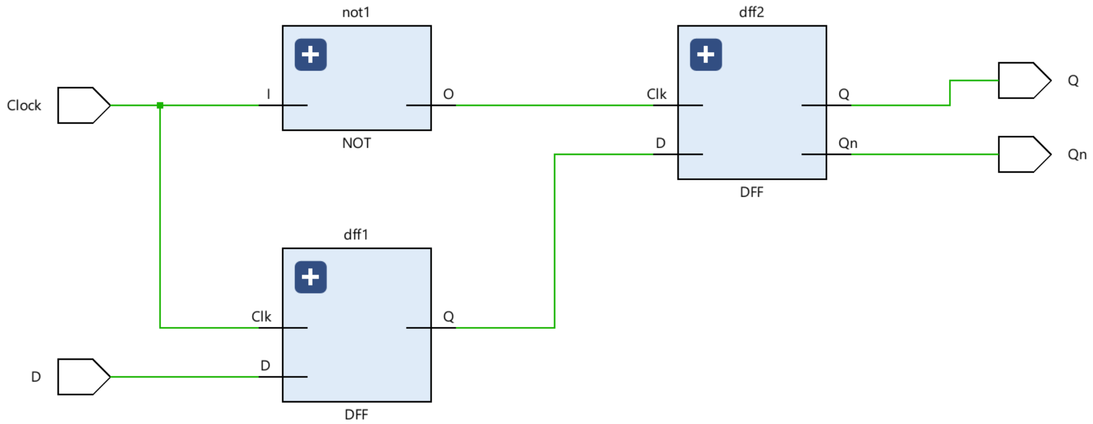
```
`timescale 1ns / 1ps



module experiment5tb;

    reg Clock;
    reg D;
    wire Q,Qn;

    MSDF uut(.D(D),.Clock(Clock),.Q(Q),.Qn(Qn));

    initial
    begin

    D=1'b0;
    Clock=1'b0;
    #5

    D=1'b1;
    #5
    Clock=1'b1;
    #5
    D=1'b0;
    #5
    Clock=1'b0;
    #10
    Clock=1'b1;
    #5
    D=1'b1;
    #10
    Clock=1'b0;
    #5;
    Clock=1'b1;
    #5
    D=1'b0;
    #5
    Clock=1'b0;
    #5

    $finish;
    end
endmodule
```

Simulation Waveform

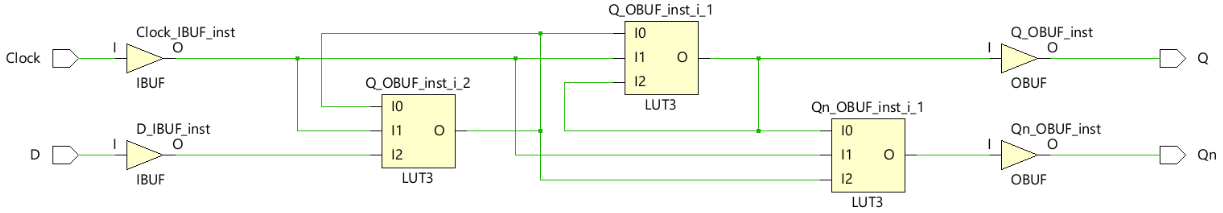


Name	Value
Clock	1
D	1
Q	X
Qn	X

RTL Schematic



Technology Schematic



4-) D Flip Flop Behavioral Design

Verilog Code

Module Code

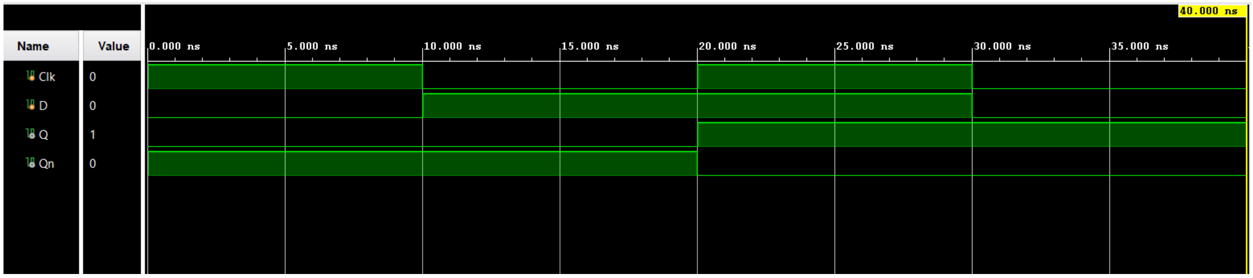
```
module DFFB(
    input D,Clk,
    output Q,Qn
);
    reg FF;
    always @(posedge Clk) begin
        FF=D;
    end
    assign Q=FF;
    assign Qn=~FF;
endmodule
```

Testbench Code

```
`timescale 1ns / 1ps


module experiment5tb;
```

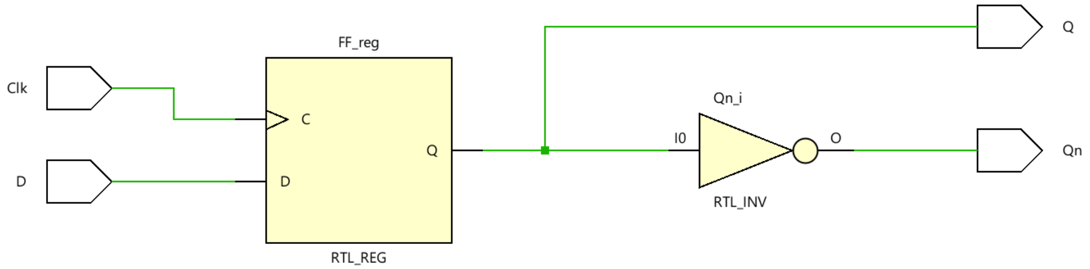
```
    reg Clk;
    reg D;
    wire Q,Qn;

    DFFB uut(.D(D),.Clk(Clk),.Q(Q),.Qn(Qn));

    initial
    begin

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

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

    D=1'b1;
    Clk=1'b1;
    #10


    D=1'b0;
    Clk=1'b0;
    #10
    $finish;
    end
endmodule
```

Simulation Waveform

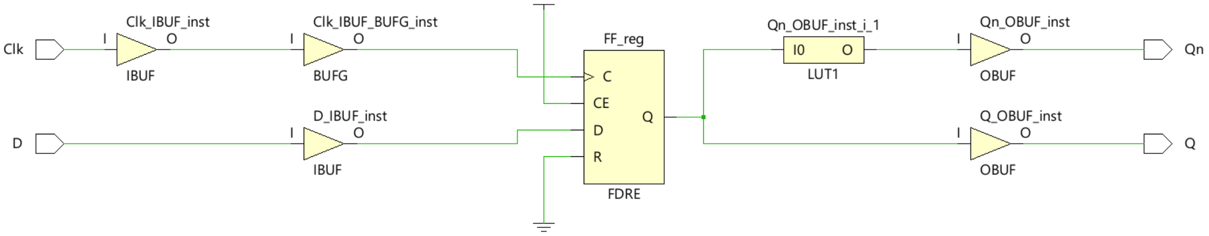


Name	Value
Clk	0
D	0
Q	1
Qn	0

RTL Schematic



Technology Schematic



5-) 8-bit Register

Verilog Code

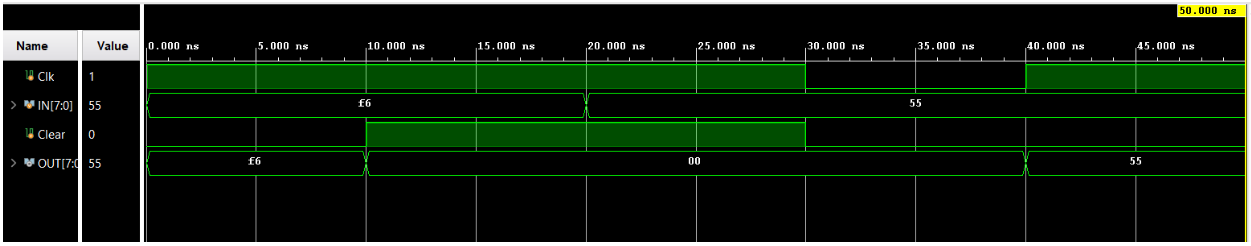
Module Code

```
module bit8Register(
    input [7:0]IN,
    input Clk,
    input Clear,
    output [7:0]OUT
);
    reg [7:0]FF;
    always@(posedge Clk)
    begin
        FF<=IN;
    end
    always @(Clear) begin
        FF <= FF & {~Clear,~Clear,~Clear,~Clear,~Clear,~Clear,~Clear,~Clear};
    end

    assign OUT = FF ;

endmodule
```

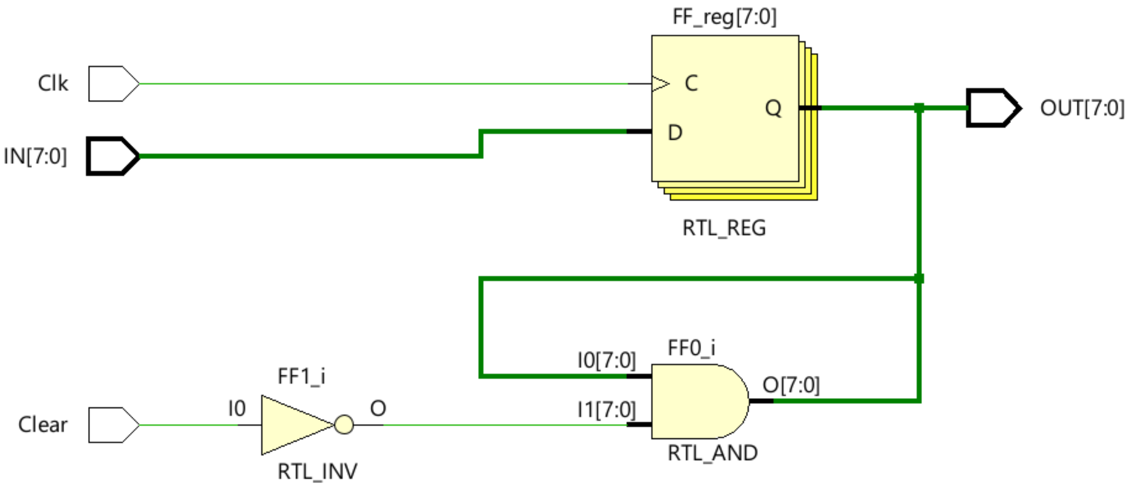
Simulation Code

```
`timescale 1ns / 1ps


module experiment5tb;

    reg Clk;
    reg [7:0]IN;
    reg Clear;
    wire [7:0]OUT;

    bit8Register uut(.Clk(Clk),.IN(IN),.Clear(Clear),.OUT(OUT)); //IN,Clk,Clear

    initial
    begin

    IN=8'b11110110;
    Clk=1'b1;
    Clear=1'b0;
    #10
    Clear=1'b1;
    #10
    IN=8'b01010101;
    Clk=1'b1;
    Clear=1'b1;
    #10
    Clear=1'b0;
    Clk=1'b0;
    #10
    Clk=1'b1;
    #10
    $finish;
    end
endmodule
```

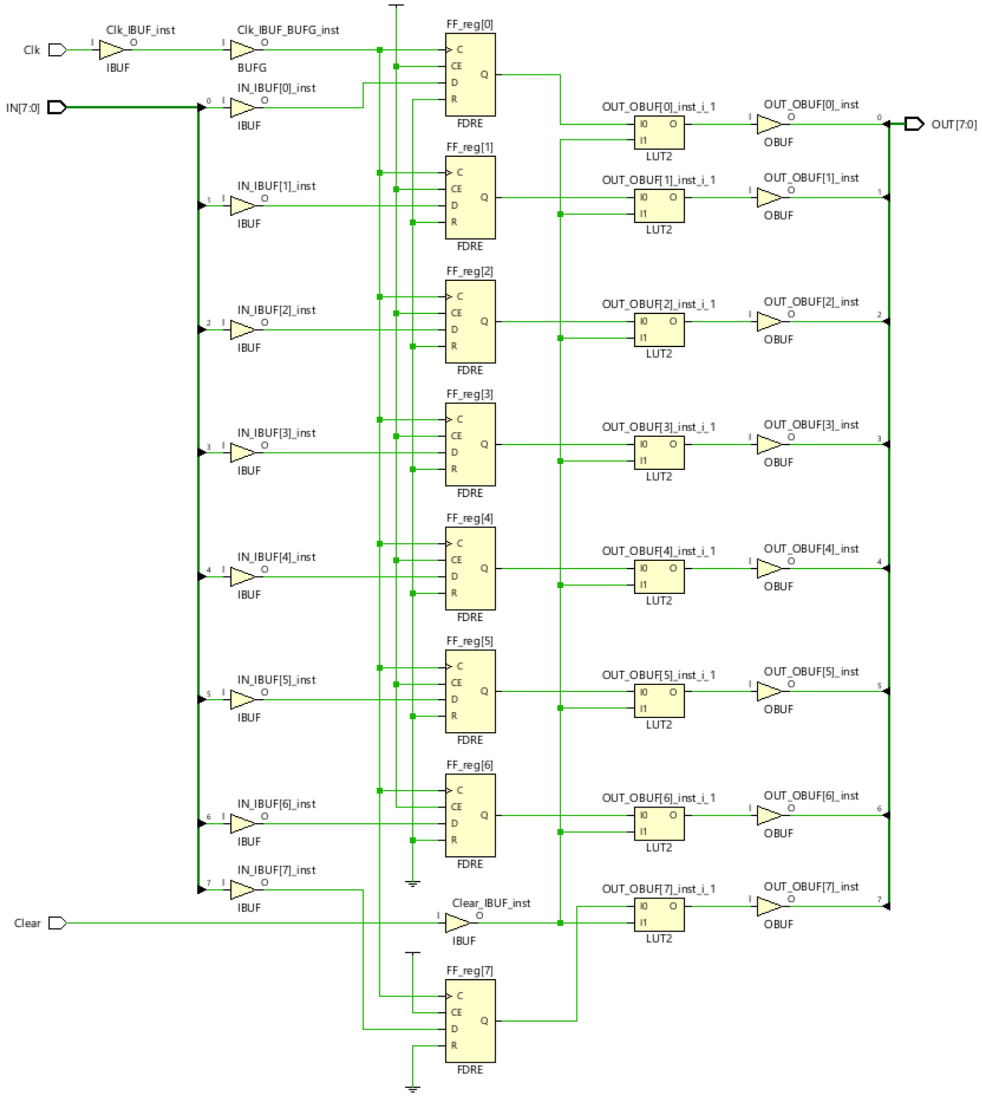
Simulation Waveform



RTL Design



Technology Schematic



We can define 4x8 array by using for loop in a for loop

```
for (int i=0; i< 3; i++)
{
    for (int j=0; j< 7; j++)
    {
        a_2D[i][j] = (byte)i*j;
    }
}
```

That is the example code

This code shows how to distribute inputs by 8 bit 8bit and we can use 4 8 bit register and distribute the one common clock to all 8 bit register

6-) Block Ram

Verilog Code

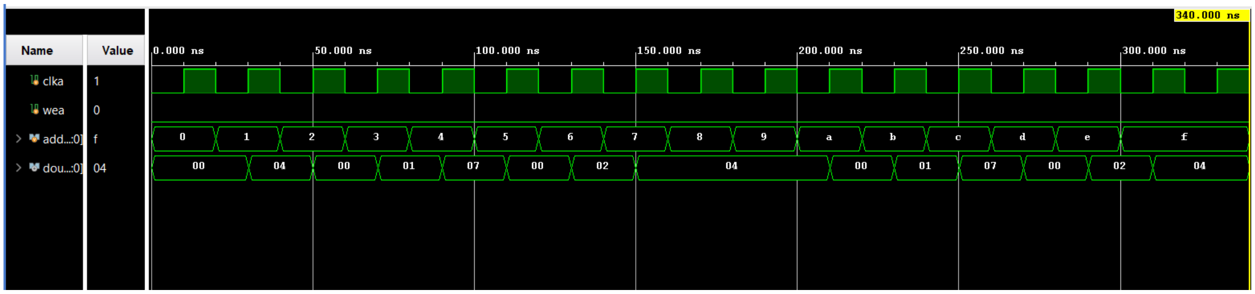
Module Code

```
module BRAM(
    input clka,
    input wea,
    input [3:0]addra,
    output [7:0]douta
);
    reg clka1;
    integer count;
    integer count_next;
    wire [7:0]dina;
    assign dina=8'b10101010;

    /*always@(posedge clka)
    begin
        if(count == 2)
        begin
            count_next<=0;
            clka1<= 1'b1;
        end
        else
        begin
        count_next=count+1;
        count=count_next;
        end
    end*/
    blk_mem_gen_0
BRAM(.clka(clka),.wea(wea),.addra(addra),.douta(douta),.dina(dina));

endmodule
```

Testbench Code

```
`timescale 1ns / 1ps


module experiment5tb;

    reg clka;
    reg wea;
    reg [3:0]addra;

    wire [7:0]douta;

    BRAM uut(.clka(clka),.wea(wea),.addra(addra),.douta(douta));

    initial
    begin
        clka=1'b0;
        wea=1'b0;
        addra=4'b0000;
        #10
        clka=1'b1;
        #10
        clka=1'b0;
        addra=4'b0001;
        #10
        clka=1'b1;
        #10
        addra=4'b0010;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b0011;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b0100;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b0101;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b0110;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b0111;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1000;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1001;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1010;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1011;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1100;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1101;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1110;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        addra=4'b1111;
        clka=1'b0;
        #10
        clka=1'b1;
        #10
        clka=1'b0;
        #10
        clka=1'b1;
        #10

    $finish;
    end
endmodule
```

Simulation Waveform



Name	Value
clka	1
wea	0
add...:0]	f
dou...:0]	04

Functionalities of Block Ram ports

wrea port is high when we want to change the data inside the blockram

clka when it is in rising edge circuit becomes active and changes the output

addra is the address port where we specify the address that we want to read or write

dina is the data what we want to write inside the blokram

douta is the output of the blockram which shows the data inside the address which is specified by addra

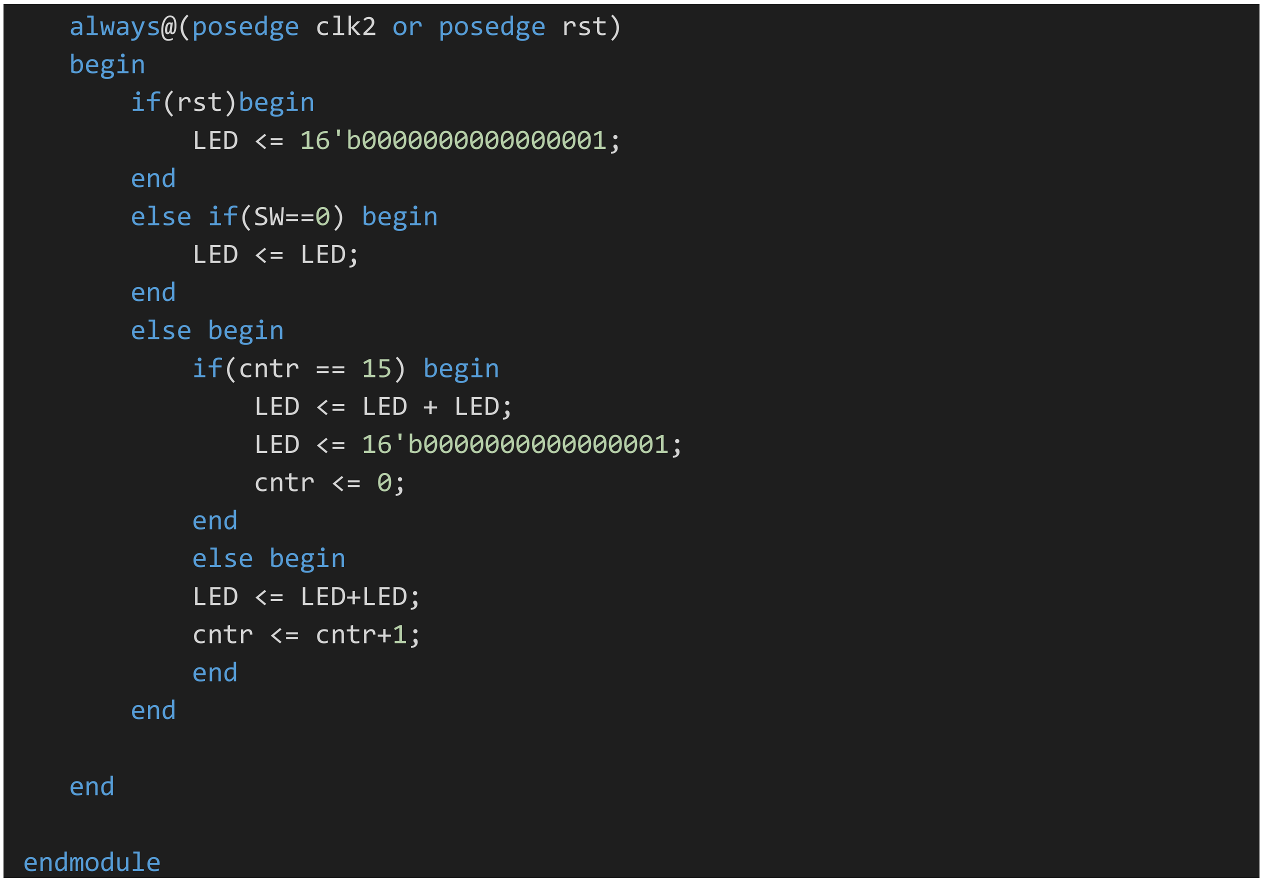
Structure of .coe file

```
MEMORY_INITIALIZATION_RADIX=2; // this indicates the radix
MEMORY_INITIALIZATION_VECTOR= // this part is the values that will be written
00000100,
00000000,
00000001,
00000111,
00000000,
00000010,
00000100,
00000100,
00000100,
00000000,
00000001,
00000111,
00000000,
00000010,
00000100,
00000100;
```

7-) Sliding LEDs

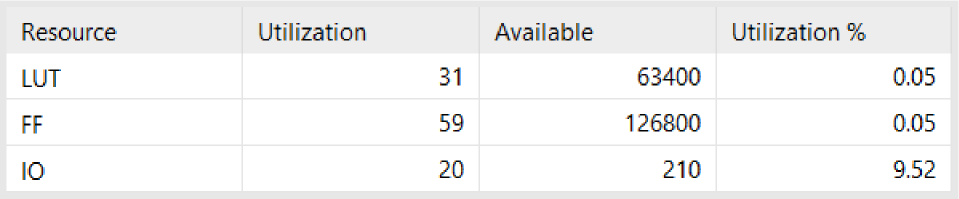
Verilog Code

```
`timescale 1ns / 1ps




 module SLED2
 (

    input clk,
    input rst,
    input [1:0]SW,
    output reg [15:0]LED

 );
    reg clk2=1;
    parameter  MAX_CNT_DEST = 5000000;
    reg [$clog2(MAX_CNT_DEST)-1:0]counter=0;
    always@(SW)
    begin
        case(SW)
        2'b00,2'b01: counter <= MAX_CNT_DEST;
        2'b10: counter <= MAX_CNT_DEST/2;
        2'b11: counter <= MAX_CNT_DEST/5;
        endcase
    end
    reg [$clog2(MAX_CNT_DEST)-1:0]counter2=0;

    always@(posedge clk)
    begin
        if(counter2 == counter)
        begin
            clk2 <= ~clk2;
            counter2 <= 0;
        end
        else begin
            counter2 <= counter2 +1;
        end
    end

    reg[3:0]cntr=0;
```

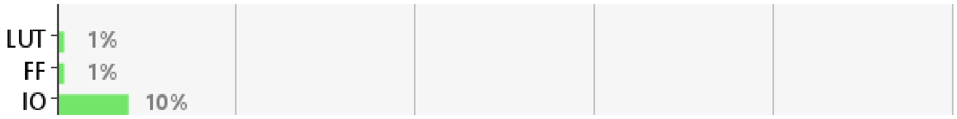


```
    always@(posedge clk2 or posedge rst)
    begin
        if(rst)begin
            LED <= 16'b0000000000000001;
        end
        else if(SW==0) begin
            LED <= LED;
        end
        else begin
            if(cntr == 15) begin
                LED <= LED + LED;
                LED <= 16'b0000000000000001;
                cntr <= 0;
            end
            else begin
            LED <= LED+LED;
            cntr <= cntr+1;
            end
        end

    end

endmodule
```

Utilization Report



Resource	Utilization	Available	Utilization %
LUT	31	63400	0.05
FF	59	126800	0.05
IO	20	210	9.52


LUT
FF
IO