

**EHB436E**

**DIGITAL SYSTEM DESIGN APPLICATIONS**

**2022 FALL**

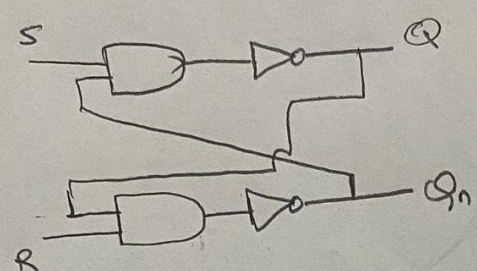
**Muhammed Erkmen**

**040170049**

**EXPERIMENT-5 REPORT**

# SR LATCH

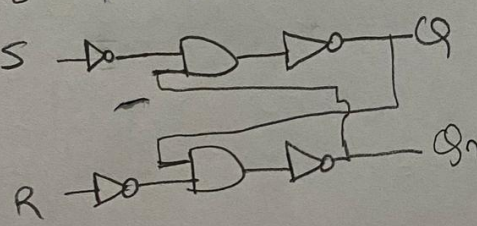
SR-Latch



$Q = (S \cdot Q_n)'$   
 $Q_n = (Q \cdot R)'$   
 $Q = S' + Q_n \rightarrow [Q_n = (Q \cdot R)']$   
 $Q = S' + Q \cdot R \rightarrow \text{Characteristic Equation.}$

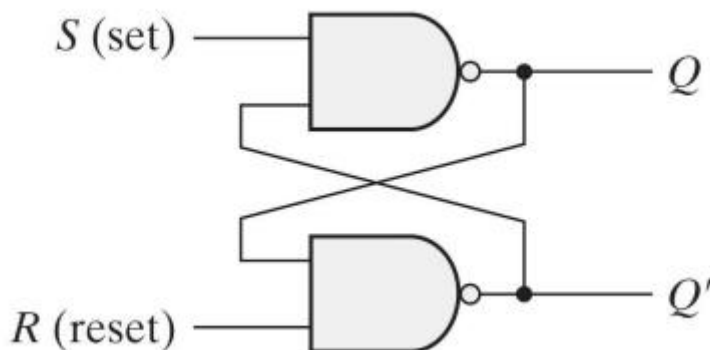
$Q_n = (Q \cdot R)'$   
 $Q_n = Q' + R'$   
 $Q_n = R' + S \cdot Q_n \rightarrow \text{inverse characteristic equation?}$

Didn't understand what "inverse characteristic equation". This circuit is low-active. When  $R \Rightarrow 0$  means Reset is enabled. When  $S \Rightarrow 0$  means SET is enabled. If we put not gates in their input, this will be high active.



$Q = (\bar{S} \cdot Q_n)'$   
 $Q_n = (\bar{R} \cdot Q)'$   
 $Q = S + Q_n \rightarrow \text{Now it became high-active. (function)}$

$$Q(t+1) = Q(t) \cdot R' + S$$



SR Latch with NAND gates Truth Table:

S	R	Q	Q'
0	0	Invalid	Invalid
0	1	1	0
1	0	0	1
1	1	No change	No change

Source Code of SR Latch:

```
`timescale 1ns / 1ps
module SR(
    input s,
    input r,
    output q,
    output qn
);
    wire q_fake;
    wire qn_fake;
    assign q_fake = q;
    assign qn_fake = qn;
    nand_gate NAND1 (.I1(s), .I2(qn_fake), .O(q));
    nand_gate NAND2 (.I1(r), .I2(q_fake), .O(qn));
endmodule
```

I designed SR Latch by using my NAND gates from SSI\_Library.

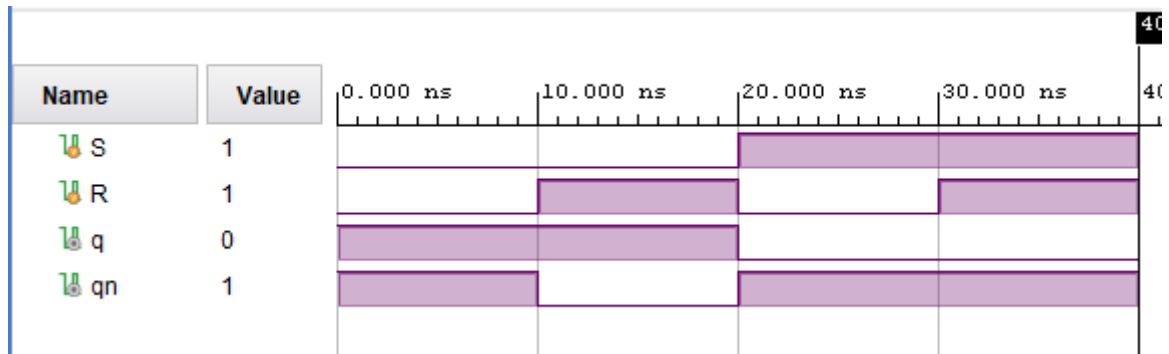
Testbench code of SR Latch:

```
`timescale 1ns / 1ps
module sr_tb;

reg S;
reg R;
wire q;
wire qn;

SR srl (.s(S) , .r(R) , .q(q) , .qn(qn));
initial begin
S=0;
R=0;
#10;
S=0;
R=1;
#10;
S=1;
R=0;
#10;
S=1;
R=1;
#10;
end
endmodule
```

Behavioral simulation results:



As expected in truth table, we were expecting when  $SR=0$ , there should be a invalid state. Q and Qnegative can not be 1 at the same time so this state is invalid.

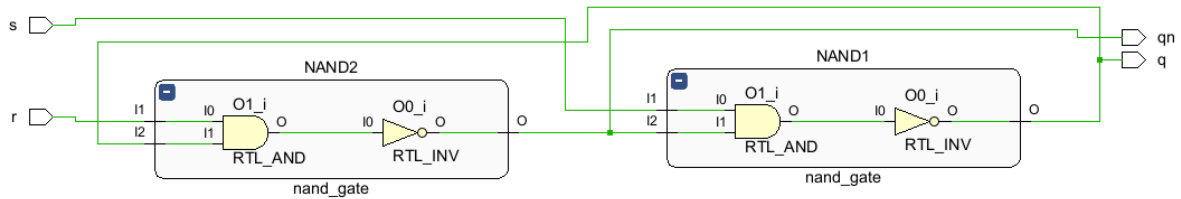
When  $S=0$  and  $R=1$ , Q goes to 1 and Qn goes to 0 as expected in truth table.

When  $S=1$  and  $R=0$ , Q goes to 0 and Qn goes to 1 as expected in truth table.

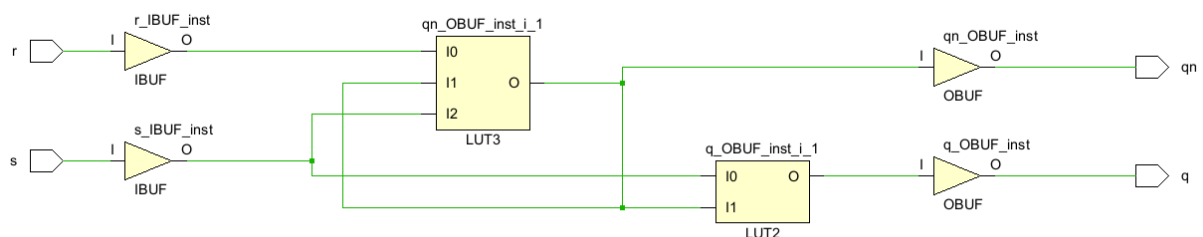
When  $S=1$  and  $R=1$ , Q stays as the last value as expected. In simulation last value before  $\{S,R\} = 2'b11$  is 0 so Q stays as 0.

So that means when we  $SET=0$  and  $RESET=1$ , we set the Q value 1. That means the circuit is low-active.

RTL schematic of SR Latch:



Technology schematic of SR Latch:



While generating bitfile, i get an error which is:

Combinational Loop (1 error)  
 [DRC LUTLP-1] Combinational Loop Alert: 1 LUT cells form a combinational loop. This can create a race condition. Timing analysis may not be accurate. The preferred resolution is to modify the design to remove combinational logic loops. If the loop is known and understood, this DRC can be bypassed by acknowledging the condition and setting the following XDC constraint on any one of the nets in the loop: 'set\_property ALLOW\_COMBINATORIAL\_LOOPS TRUE [get\_nets <myHiermyNet>]'. One net in the loop is qn\_OBUF.  
 Please evaluate your design. The cells in the loop are: qn\_OBUF\_inst\_i\_1.

After that error, i added 2 constraints for allowing loops for Q and Qn.

```
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets q_OBUF ];
set_property ALLOW_COMBINATORIAL_LOOPS TRUE [get_nets qn_OBUF ];
```

Then i generated bitstream and there was no problem.

# D LATCH

D-FlipFlop

$$Q = (Q_n \cdot (D \cdot CLK)')' = \overline{Q_n} + D \cdot CLK$$

$$Q_n = (Q \cdot (D' \cdot CLK)')' = Q \cdot (D' \cdot CLK)' + D \cdot CLK$$

$$Q_{t+1} = Q_t (D + \overline{CLK}) + D \cdot CLK$$

$$Q_{t+1} = D \cdot CLK + Q_t \cdot \overline{CLK} + D \cdot Q_t$$

TRUTH TABLE

CLK	D	$Q_t$	$Q_{t+1}$	$Q_n(t+1)$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

$$Q_n = (Q \cdot (D' \cdot CLK)')' = Q' + D \cdot CLK$$

$$Q_n(t+1) = Q_t' + D \cdot CLK$$

→ When CLK=1 goes D', when CLK=0 goes  $\overline{Q_t}$ .

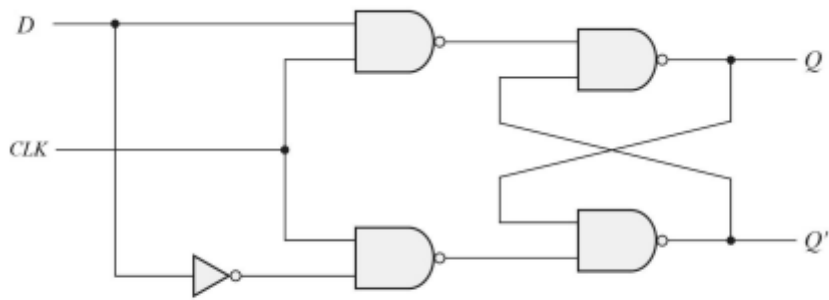
$$Q_{t+1} = CLK \cdot D + Q_t(\overline{CLK} + D)$$

So  $Q_{t+1}$  is dependent on clock, D and  $Q_t$   
 When clock=1,  $Q_{t+1} \leftarrow D$   
 When clock=0,  $Q_{t+1} \leftarrow Q_t$

$$Q(t+1) = D \cdot CLK + Q(t) \cdot CLK' + D \cdot Q_t$$

$$Q(t+1) = D(CLK + Q(t)) + Q(t) \cdot CLK'$$

$$Q(t+1) = D \cdot CLK + Q(t) \cdot CLK'$$

**Fig.2: D Latch**

Truth table of D Latch:

CLK	D	Q	Qn
0	0	Q	Qn
0	1	Q	Qn
1	0	0	1
1	1	1	0

Source Code of D Latch:

```
`timescale 1ns / 1ps
module dflipflop(
    input D,
    input clk,
    output q,
    output qn
);
    wire dnot;
    wire q_b;
    wire qn_b;
    not_gate n1 (.I(D),.O(dnot));
    nand_gate n2(.I1(dnot),.I2(clk),.O(qn_b));
    nand_gate n3(.I1(q),.I2(qn_b),.O(qn));
    nand_gate n4(.I1(D),.I2(clk),.O(q_b));
    nand_gate n5(.I1(q_b),.I2(qn),.O(q));
endmodule
```

## Testbench Code of D Latch:

```

`timescale 1ns / 1ps
module dflipflop_tb;

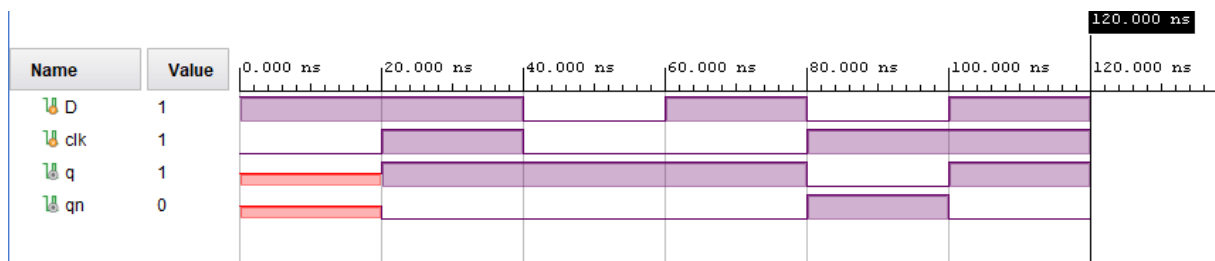
reg D;
reg clk;
wire q;
wire qn;

dflipflop DUT (.D(D) ,.clk(clk) ,.q(q) ,.qn(qn) );

initial begin
D=1;
clk=0;
#20;
D=1;
clk=1;
#20;
clk=0;
D=0;
#20;
clk=0;
D=1;
#20;
clk=1;
D=0;
#20;
clk=1;
D=1;
#20;
$finish;
end
endmodule

```

## Behavioral simulation results:

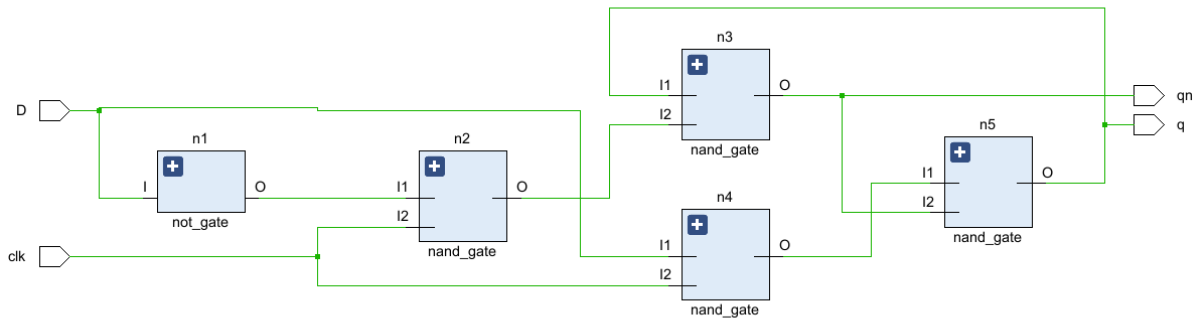


So until clk=1; q and qn is unknown (X). I loaded Q as 1 with using CLK=1 and D=1.

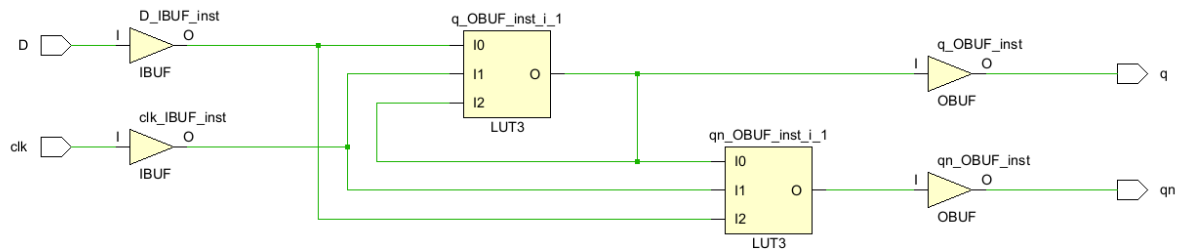
After that i look into the simulation results. According to truthtable, when CLK=0, there should be no change. In simulation that expectation comes true. When CLK=1, Q should be equal to D as expected in truth table. So that expectation comes true too.



## RTL schematic of D Latch:



## Technology schematic of D Latch: Timing



## Timing summary of D Latch:

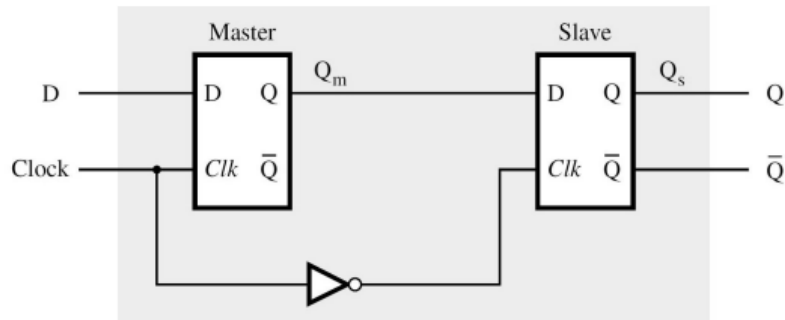
### Combinational Delays

From Port	To Port	Max Process Corner	Min Delay	Min Process Corner
clk	qn	9.718	2.835	FAST
clk	q	8.926	2.692	FAST
D	qn	8.720	2.434	FAST
D	q	7.929	2.306	FAST

Maximum delay is between clk and qn. That means we need clock period > 9.718 nanoseconds.

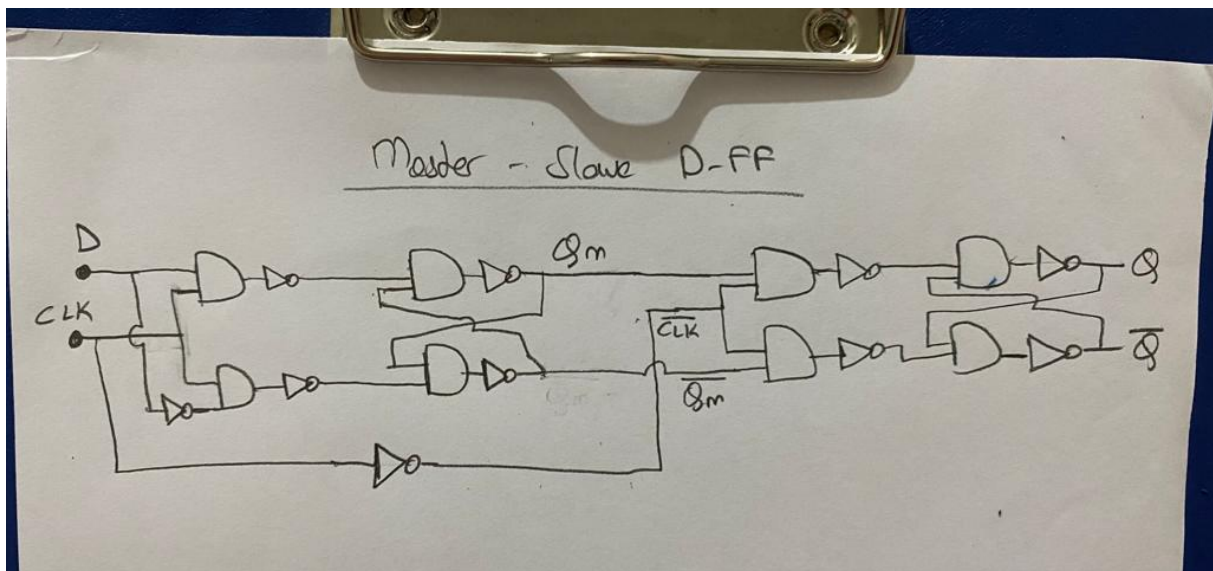
Our clock limit is 102,9018 MHz. We can work maximum in this clock range.

## Master-Slave D flip-flop



**Fig.3:** Master-Slave D-FF

Circuit diagram



How this works?

So, a D latch output Q goes D when Clock = 1. When we give CLK=1 to MASTER D, Qm goes to D, but clock of slave is Clock=0. When we make CLK 0, clock of slave goes to 1 and Qs goes to Qm. So that means, when CLK goes 1 to 0, Q output goes to Qm. According to these, when CLOCK goes 1 to 0 which means negative edge of clock, output goes D.

Source Code:

```
`timescale 1ns / 1ps

module masterslaved(
    input D,
    input clk,
    output q,
    output qn
);
    wire qm;
    wire qm_n;
    wire clk_n;
    wire qs;
    wire qs_n;

    assign q = qs;
    assign qn = qs_n;
    not_gate not1 (.I(clk),.O(clk_n));
    dflipflop MASTER (.D(D),.clk(clk),.q(qm),.qn(qm_n));
    dflipflop SLAVE (.D(qm),.clk(clk_n),.q(qs),.qn(qs_n));
endmodule
```

I used dflipflop module which designed in step 2.

## Testbench Code:

```

`timescale 1ns / 1ps

module masterslaved_tb;

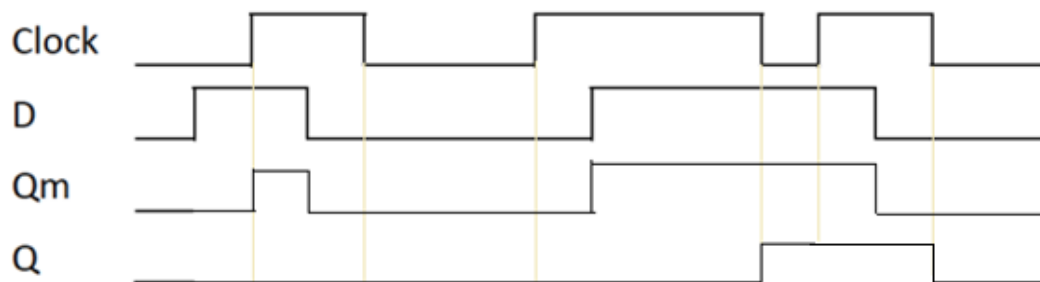
reg clk;
reg D;
wire qs;
wire qs_n;

masterslaved MASTER_SLAVE_D_FF (.D(D),.clk(clk),.q(qs),.qn(qs_n));

initial begin
clk = 0;
D= 0;
#20;
D=1;
#20;
clk=1;
#15;
D=0;
#5;
clk=0;
#20;
clk=1;
#10;
D=1;
#10;
clk = 0;
#20;
$finish;
end
endmodule

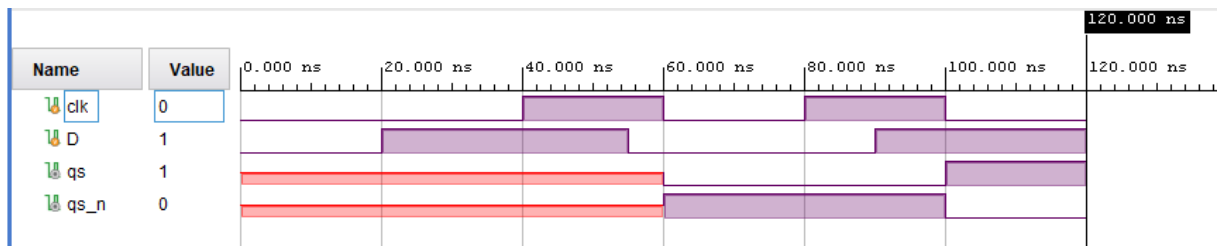
```

In experiment sheet, there was an assignment to draw Qm and Q. This is the solution:



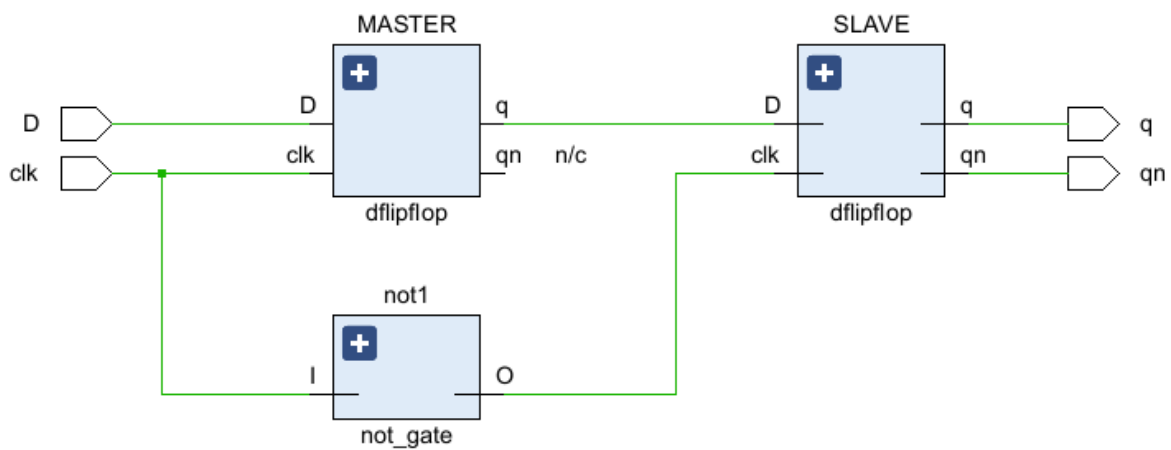
**Fig.4:** Time diagram for Master-Slave Flip-Flop.

Behavioral Simulation results:

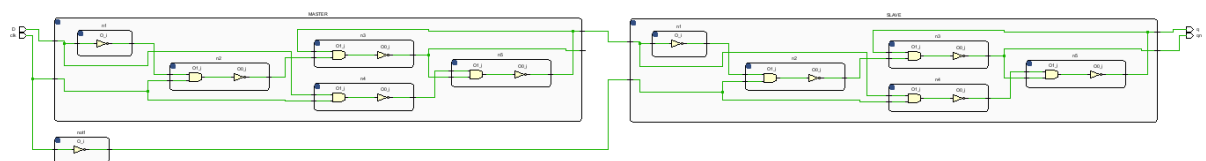


As we can see, NEGEDGE is the edge of clock which is effecting output.

RTL Schematic of Master-Slave D FF:



Circuit diagram:



## Behavioral D flip-flop

Source Code:

```
`timescale 1ns / 1ps

module dflipflopbehav(
    input D,
    input clk,
    output Q,
    output Qn
);
    reg FF;
    assign Q = FF;
    assign Qn = ~FF;
    always @(posedge clk) begin
        FF <= D;
    end

endmodule
```

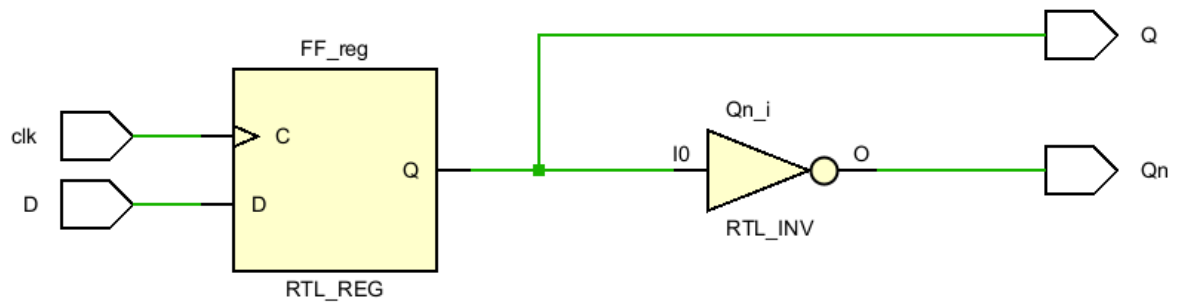
Testbench Code:

```
`timescale 1ns / 1ps
module dflipflopbehav_tb;
    reg D;
    reg clk=0;
    wire Q;
    wire Qn;

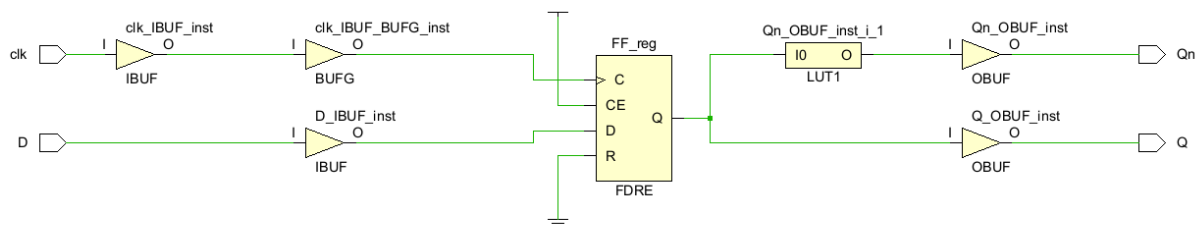
    dflipflopbehav DUT (.D(D), .clk(clk), .Q(Q), .Qn(Qn));
    always begin
        clk = ~clk;
        #10;
    end

    initial begin
        #50;
        D=1;
        #200;
        D=0;
        #100;
        D=1;
        #5;
        D=0;
        #5;
    end
endmodule
```

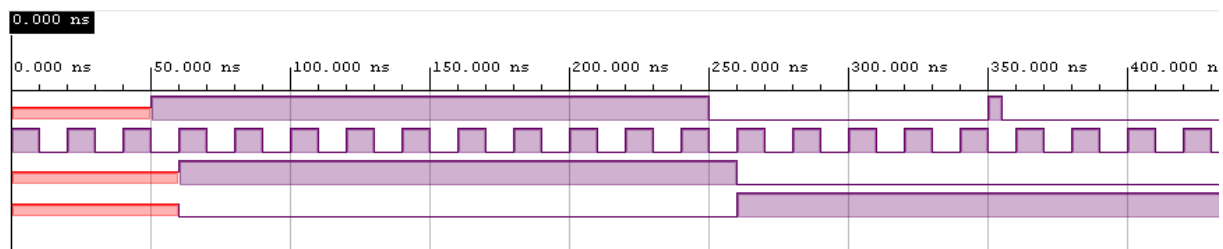
## RTL Schematic



## Technology Schematic:



## Behavioral Simulation Results:



## 8 bit register

Source Code:

```
`timescale 1ns / 1ps

module eightbitreg
(
    input [7:0] IN,
    input clk,
    input clear,
    output [7:0] OUT
);
reg [7:0] out_fake;

assign OUT = clear? 0:out_fake;
always @(posedge clk) begin
    if(!clear)
        out_fake <= IN;
    end

endmodule
```

Testbench Code:

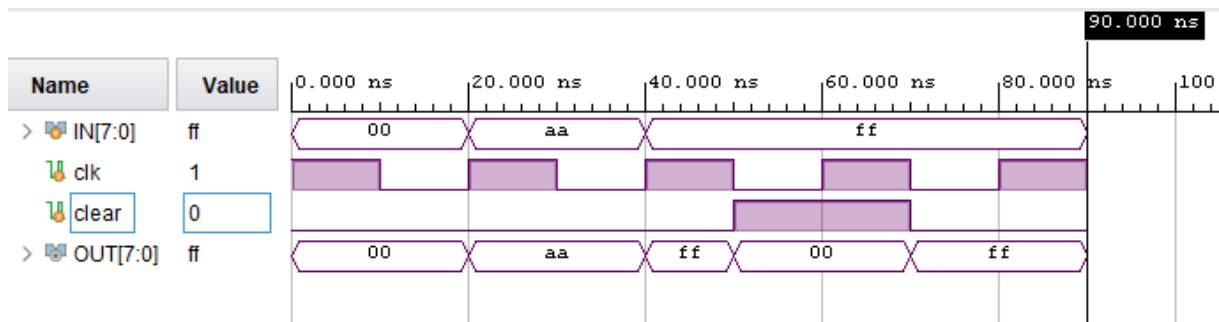
```
`timescale 1ns / 1ps
module eightbitreg_tb;
reg [7:0] IN;
reg clk=0;
reg clear;
wire [7:0] OUT;
eightbitreg DUT (
    .IN(IN),
    .clk(clk),
    .clear(clear),
    .OUT(OUT)
);

always begin
    clk = ~clk;
    #10;
end
initial begin
    IN = 8'b0000_0000;
    clear = 0;
    #20;
    IN = 8'b1010_1010;
    #20;
    IN = 8'b1111_1111;
    #10;
    clear = 1;
    #20;
    clear = 0;
    IN=8'b0000_0100;
    #20;
end

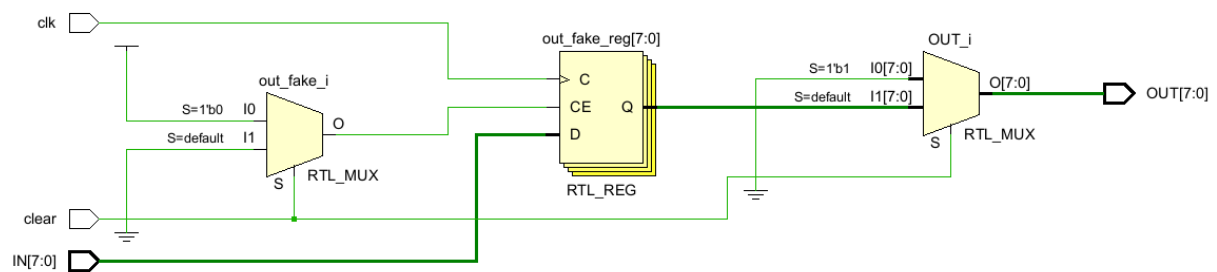
endmodule
```



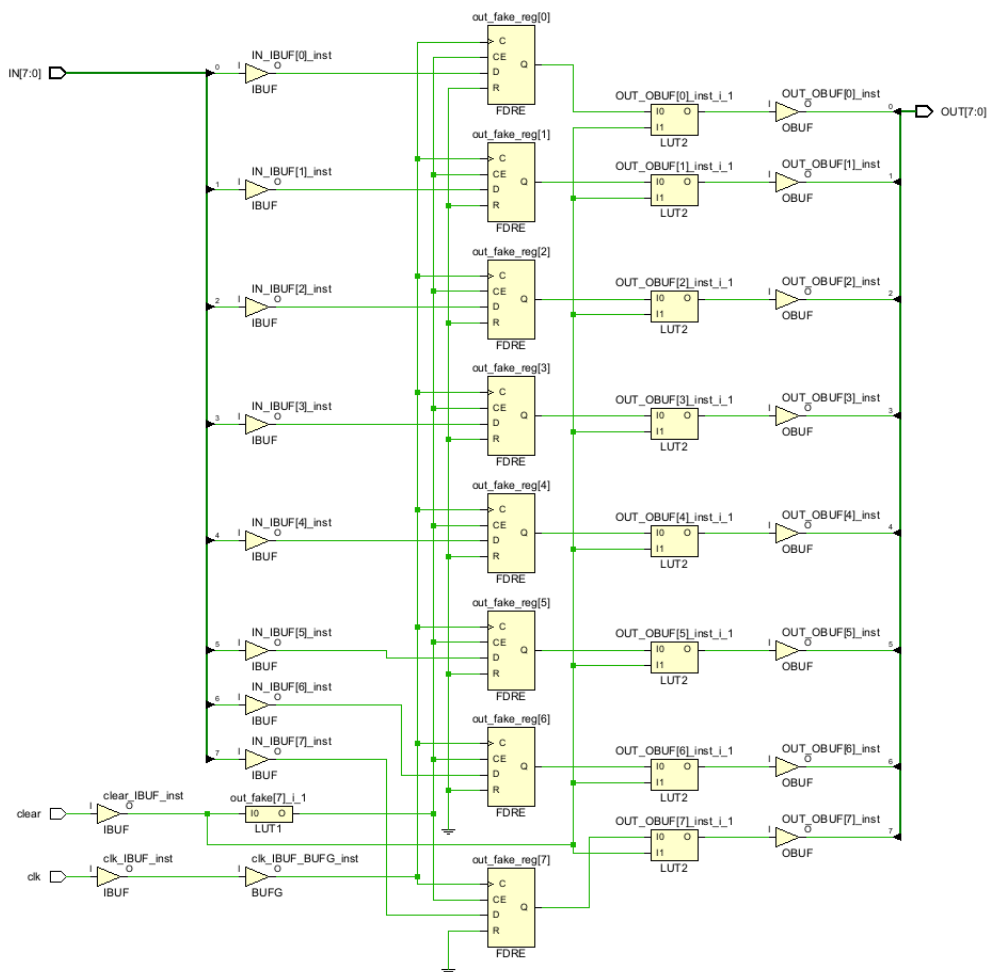
Behavioral Simulation of 8 bit register with clear button:



RTL Schematic of 8 bit register:



## Technology Schematic of 8 bit register:



Defining 4x8-bit register:

An 4x8bit array can be defined as:

```
reg [7:0] reg_array [3:0];
```

means there will be 4 – 8 bit register in reg\_array. If we want to create more than one register or wire set, we can create arrays with declaring its size before the name of the register/wire.

## 8 bit register reset reg

Source Code:

```
`timescale 1ns / 1ps

module eightbitreg_resetreg(
    input [7:0] IN,
    input clk,
    input clear,
    output reg [7:0] OUT

);

always @(posedge clk or posedge clear)
begin
if(clear)
OUT <= 0;
else
OUT <= IN;
end
endmodule
```

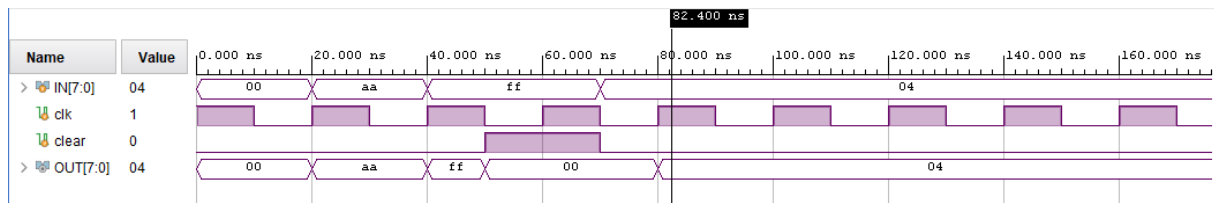
Testbench Code:

```
`timescale 1ns / 1ps
module eightbitreg_tb;
reg [7:0] IN;
reg clk=0;
reg clear;
wire [7:0] OUT;
eightbitreg DUT (
    .IN(IN),
    .clk(clk),
    .clear(clear),
    .OUT(OUT)
);

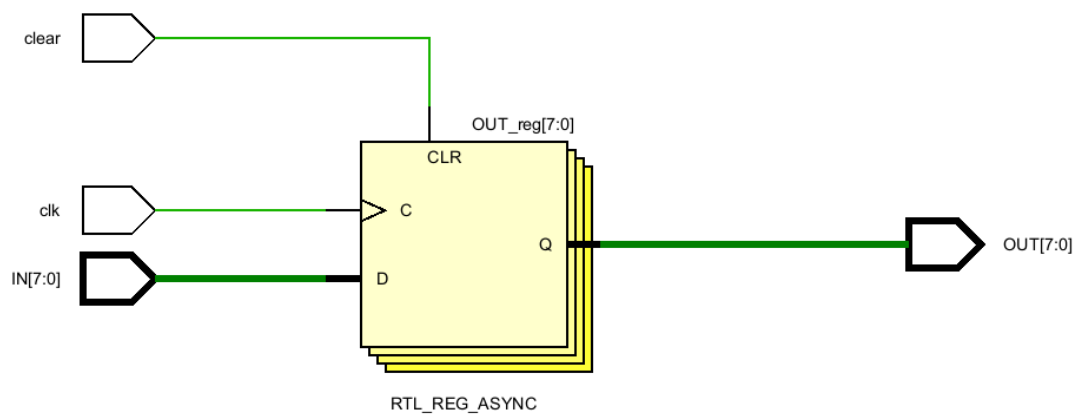
always begin
clk = ~clk;
#10;
end
initial begin
IN = 8'b0000_0000;
clear = 0;
#20;
IN = 8'b1010_1010;
#20;
IN = 8'b1111_1111;
#10;
clear = 1;
#20;
clear = 0;
IN=8'b0000_0100;
#20;
end

endmodule
```

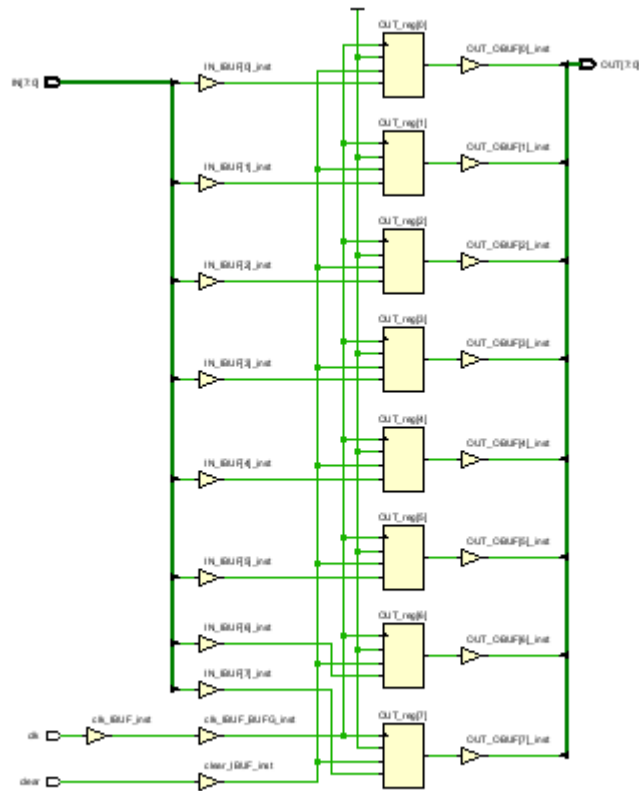
## Behavioral Simulation:



## RTL Schematic



Technology Schematic:



# Block Ram

Source Code:

```
`timescale 1ns / 1ps

module lastmodule(
    input clk,
    input wea,
    input [3:0] addra,
    output [7:0] douta);

wire clka;
clkdiv clkd (.clk(clk),.clko(clka));
top_block_ram ss (.clk(clka),.wea(wea),.addra(addra),.douta(douta));
endmodule

module top_block_ram(
    input clk,
    input wea,
    input [3:0] addra,
    output [7:0] douta
);

wire [7:0] wire_din;
blk_mem_gen_0 MEM1
(.clka(clk),.dina(wire_din),.wea(wea),.addra(addra),.douta(douta));
endmodule

module clkdiv(input clk,
               output reg clko=1);
reg counter=0;
always @(posedge clk)
if(counter==0)
counter <=counter+1;
else
clko<=~clko;
begin

end
endmodule
```

My memory.coe file (coded 40170049 for two times in binary):

```
MEMORY_INITIALIZATION_RADIX=2;
MEMORY_INITIALIZATION_VECTOR=
00000100,
00000000,
00000001,
00000111,
00000000,
00000000,
00000100,
00001001,
00000100,
00000000,
00000001,
00000111,
00000000,
00000000,
00000100,
00001001;
```

.coe file which is used for coefficients in block ram has the RADIX which shows the number set of the numbers binary, decimal, hexa etc. Memory initialization vector works for putting initial datas to block ram respectively to increasing addresses.

## Testbench Codes:

```

`timescale 1ns / 1ps
module top_block_ram_tb;
reg clk=0;
reg wea;
reg [3:0] addra;
wire [7:0] douta;
lastmodule DUT (.clk(clk),
                .wea(wea),
                .addra(addra),
                .douta(douta)
                );

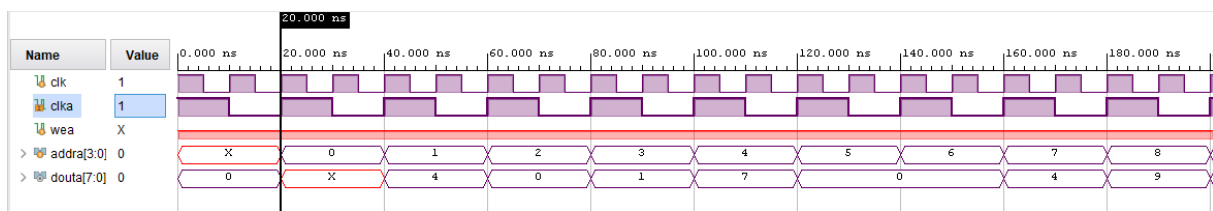
always begin
clk = ~clk;
#5;
end

initial begin
#20;
addra = 4'd0; #20;
addra = 4'd1; #20;
addra = 4'd2; #20;
addra = 4'd3; #20;
addra = 4'd4; #20;
addra = 4'd5; #20;
addra = 4'd6; #20;
addra = 4'd7; #20;
addra = 4'd8; #20;
addra = 4'd9; #20;
addra = 4'd10; #20;
addra = 4'd11; #20;
addra = 4'd12; #20;
addra = 4'd13; #20;
addra = 4'd14; #20;
addra = 4'd15; #20;
#20;
wea=1;
#640;
wea=0;
#20;
addra = 4'd0; #20;
addra = 4'd1; #20;
addra = 4'd2; #20;
addra = 4'd3; #20;
addra = 4'd4; #20;
addra = 4'd5; #20;
addra = 4'd6; #20;
addra = 4'd7; #20;
addra = 4'd8; #20;
addra = 4'd9; #20;
addra = 4'd10; #20;
addra = 4'd11; #20;
addra = 4'd12; #20;
addra = 4'd13; #20;
addra = 4'd14; #20;
addra = 4'd15; #20;
end

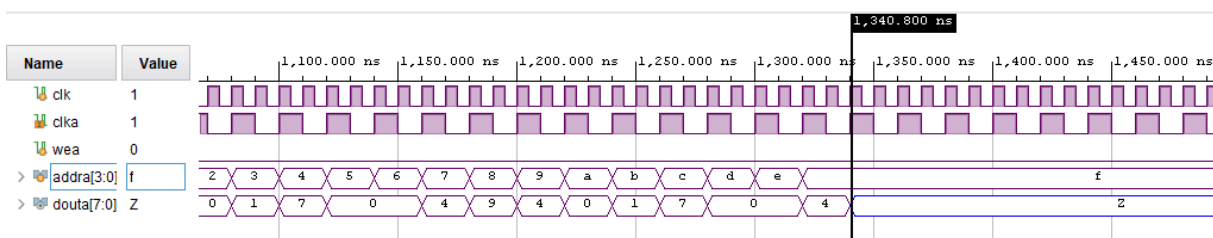
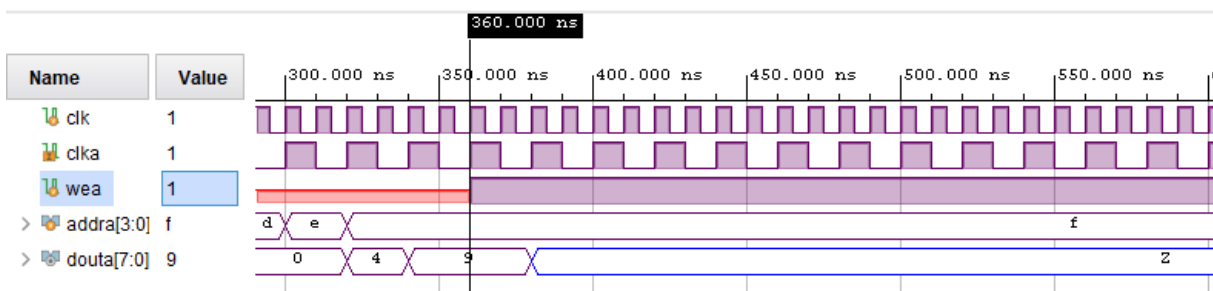
endmodule

```



**Behavioral Simulation Results:**

This simulation result shows that i can get my student id correct from the memory with 1 clock cycle delay. I connected my top module to 100MHz but memory to 50MHz because when i make the clock constraint in constraint file, i get error about clock pin and couldn't find any solution. With some research, people suggested clock wizard to get 50MHz clock, but that is equal to coding a clockdivider module. So i did it my own and created clock divider module.



This simulation results both show that our writing operation is succeed because we can read the high impedance value from f because we did not connect the datain wires inside the module to any port.

To get 50 MHz clock, i added to my constraint file this line:

```
create_clock -add -name sys_clk_pin -period 20.00 -waveform {0 5} [get_ports {clka}];
```

**Functionalities of Block RAM ports:**

addra port works for giving address to the RAM which is we are going to read / write.

wen port works for the info that we want to write data to RAM.

douta is the port we get the read data.

dina is the port we give the write data.

clka is the clock port.