DLD LAB - ASSIGNMENT

NAME : M.Taran

REG NO :19BCE7346

SLOT : L17+18

EXPERIMENT-1

Verification of Latches and Flip-Flops (D flip flop).

THEORY:

The D latch (D for "data") or transparent latch is a simple extension of the gated SR latch that removes the possibility of invalid input states (metastability). Since the gated SR latch allows us to latch the output without using the S or R inputs, we can remove one of the inputs by driving both the Set and Reset inputs with a complementary driver, i.e. we remove one input and automatically make it the inverse of the remaining input. The D latch outputs the D input whenever the Enable line is high, otherwise the output is whatever the D input was when the Enable input was last high. This is why it is also known as a transparent latch - when Enable is asserted, the latch is said to be "transparent" - it signals propagate directly through it as if it isn't there.

Aim:

To verify Latches and Flip-Flops (D flip flop)

Equipment required:

1. Xilinx software

2. Model Sim

CODE-

Verilog code for Rising Edge D Flip Flop:

```
module RisingEdge_DFlipFlop(D,clk,Q);
input D;
input clk;
output Q;
always @(posedge clk)
begin
 Q <= D;
end
endmodule
```

Verilog code for Rising Edge D Flip-Flop with Synchronous Reset :

```
module RisingEdge_DFlipFlop_SyncReset(D,clk,sync_reset,Q);
input D;
input clk;

input sync_reset;

output reg Q;

always @(posedge clk)

begin

 if(sync_reset==1'b1)

  Q <= 1'b0;

 else

  Q <= D;

end

endmodule
```

Verilog code for Rising Edge D Flip-Flop with Asynchronous Reset High Level :

```
module RisingEdge_DFlipFlop_AsyncResetHigh(D,clk,async_reset,Q);

input D;

input clk;

input async_reset;

output reg Q;

always @(posedge clk or posedge async_reset)

begin

 if(async_reset==1'b1)
```

```
  Q <= 1'b0;
 else
  Q <= D;
end
endmodule
```

Verilog code for Rising Edge D Flip-Flop with Asynchronous Reset Low Level :

```
module RisingEdge_DFlipFlop_AsyncResetLow(D,clk,async_reset,Q);
input D;
input clk;input async_reset;
output reg Q;
always @(posedge clk or negedge async_reset)
begin
 if(async_reset==1'b0)
  Q <= 1'b0;
 else
  Q <= D;
end
endmodule
```

Verilog code for Falling Edge D Flip Flop :

```
module FallingEdge_DFlipFlop(D,clk,Q);

input D;

input clk;

output reg Q; always @(negedge clk)

begin

 Q <= D;

end

endmodule
```

Verilog code for Falling Edge D Flip-Flop with Synchronous Reset :

```
module FallingEdge_DFlipFlop_SyncReset(D,clk,sync_reset,Q);

input D;

input clk;

input sync_reset;

output reg Q;

always @(negedge clk)

begin

 if(sync_reset==1'b1)

  Q <= 1'b0;
 else
  Q <= D;
end
endmodule
```

Verilog code for Falling Edge D Flip-Flop with Asynchronous Reset High Level :

```
module FallingEdge_DFlipFlop_AsyncResetHigh(D,clk,async_reset,Q);
input D; input clk;
input async_reset;
output reg Q;
always @(negedge clk or posedge async_reset)
begin
 if(async_reset==1'b1)
  Q <= 1'b0;
 else
  Q <= D;
end
endmodule
```

Verilog code for Falling Edge D Flip-Flop with Asynchronous Reset Low Level :

```
module FallingEdge_DFlipFlop_AsyncResetLow(D,clk,async_reset,Q);

input D;

input clk;

input async_reset;

output reg Q;

always @(negedge clk or negedge async_reset)

begin

 if(async_reset==1'b0)

  Q <= 1'b0;

 else

  Q <= D;

end

endmodule
```

Verilog Testbench code to simulate and verify D Flip-Flop:

```
timescale 1ns/1ps;

module tb_DFF();

reg D;

reg clk;

reg reset;

wire Q;
```

```
RisingEdge_DFlipFlop_SyncReset dut(D,clk,reset,Q);


initial begin
  clk=0;
     forever #10 clk = ~clk;
end
initial begin
 reset=1;
 D <= 0;
 #100;
 reset=0;
 D <= 1;
 #100;
 D <= 0;
 #100;
 D <= 1;
end
endmodule
```

Conclusion :

In this lab, we learned the functionality of D flipflop. We modeled and verified the functionality of these components. Xilinx also provides some *basic latches and flip-flops library components which a designer can instantiate and use instead of writing a model. Writing a model provides portability across vendors and technologies whereas instantiating library components enable a quick use of a component without re-inventing the wheel.*

EXPERIMENT-2:

Design and verification of 4-bit Shift Registers using Verilog code in Xilinx

THEORY:

When several flip-flops are grouped together, with a common clock, to hold related information the resulting circuit is called a register. Just like flip-flops, registers may also have other control signals. A register stores bits of information in such a way that systems can write to or read out all the bits simultaneously.

Examples of registers include data, address, control, and status. Simple registers will have separate data input and output pins but clocked with the same clock source.

Aim:

To implement shift-right and shift-left register using Verilog code in Xilinx.

Objectives :

In this lab, a shift-right and shift-left register are designed. The objective will be to test these designs on Xilinx simulation tool. The tests will be *performed for all the possible combinations of inputs to verify their functionality. Moreover, the knowledge gained will be used to design complex designs.*

The simple register will work where the information needs to be registered every clock cycle. However, there are situations where the register content should be updated only when certain condition occurs. For example, a status register in a computer system gets updated only when certain instructions are executed. In such case, a register clocking should be controlled using a control signal. Such registers will have a clock enable pin

Equipment required:

1. Xilinx software
2. Model Sim

Logic diagram(s): Logic diagram for a shift-right and shift-left register are shown in Fig

CODE FOR SHIFT RIGHT AND SHIFT LEFT

```
module slsr(sl, sr, din, clk, reset,Q);

 input sl, sr, din, clk, reset;

 output [7:0] Q; reg [7:0] Q;

 always @ (posedge clk) begin
 if (~reset) begin
 if (sl) begin
 Q <= #2 {Q[6:0],din};
End
else if (sr) begin
 Q <= #2 {din, Q[7:1]};
 End
End
 End
always @ (posedge reset) begin
 Q<= 8'b00000000;
End
Endmodule
```

TEST BENCH :

```
 module main;
 reg clk, reset, din, sl, sr;
wire [7:0] q;
 slsr slsr1(sl, sr, din, clk,
 reset, q);
 initial begin
forever begin
 clk <= 0;
 #5
 clk <= 1;
#5
 clk <= 0;
end
end
 initial begin
reset = 1;
 #12
reset = 0;
#90
reset = 1;
 #12
reset = 0;
 End
initial begin
sl = 1;
sr = 0;
#50
sl = 0;
#12
sr = 1;
end
initial begin
forever begin
```

```
din = 0;

#7

din = 1;

 #8

din = 0;

end

end

endmodule
```

Conclusion :

In this lab, we learned how various kinds of registers and counters work. We modeled and verified the functionality of these components. These components are widely used in a processor system design.

EXPERIMENT-3:

Design and verification of 4-bit Ring and Johnson counter using Verilog code in Xilinx

THEORY: *Ring counter is a typical application of Shift resister. Ring counter is almost same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in case of ring counter but in case of shift resister it is taken as output.*

Except this all the other things are same.So, for designing 4-bit Ring counter we need 4 flip-flop.

In this diagram, we can see that the clock pulse (CLK) is applied to all the flip-flop simultaneously. Therefore, it is a Synchronous Counter.

Also, here we use Overriding input (ORI) to each flip-flop. Preset (PR) and Clear (CLR) are used as ORI.

Aim:

To implement 4-bit Ring counter using Verilog code in Xilinx.

Equipment required:

1. Xilinx software

2. Model Sim

Ring Counter-

Code-

```
module
ring_count(q,clk,clr);
```

```
input clk,clr;

output [3:0]q;

 reg [3:0]q;

 always @(posedge clk)

      if(clr==1)

           q<=4′b1000;

       Else

           Begin

               q[3]<=q[0];

               q[2]<=q[3];

               q[1]<=q[2];

               q[0]<=q[1];

           end

endmodule

//testbench//

`timescale 1ns/1ps

module ring_count_test();

   reg clk_tb,clr_tb;
```

```
  wire [3:0]q_tb;

  ring_count dut1(q_tb,clk_tb,clr_tb);

  initial

     begin

     $display(“time,\t clk_tb,\t clr_tb,\t q_tb”);

     $monitor(“%g,\t %b,\t %b,\t
%b”,$time,clk_tb,clr_tb,q_tb);

     clr_tb=1′b0;

     #50 clr_tb=1′b1;

     #100 clr_tb=1′b0;

     end

     always

        begin

        #50 clk_tb=1′b1;

       #50 clk_tb=1′b0;

        end

endmodule
```

Johnson Counter-

THEORY: *Johnson counter also known as creeping counter, is an example of synchronous counter. In Johnson counter, the complemented output of last flip flop is connected to input of first flip flop and to implement n-bit Johnson counter we require n flip-flop.It is one of the most important type of shift register counter. It is formed by the feedback of the output to its own input.Johnson counter is a ring with an inversion.Another name of Johnson counter are:creeping counter, twisted ring counter, walking counter, mobile counter and switch tail counter.*

Aim :

To implement 4-bit Johnson counter using Verilog code in Xilinx.

Equipment required:

1. *Xilinx software*
2. *Model Sim*

Code -

```
module johnson_ctr #(parameter WIDTH=4)

 (

 input clk,
  input rstn,
    output reg [WIDTH-1:0] out
  );


  always @ (posedge clk) begin
      if (!rstn)
         out <= 1;
      else begin
        out[WIDTH-1] <= ~out[0];
        for (int i = 0; i < WIDTH-1; i=i+1) begin
          out[i] <= out[i+1];
        end
      end
  end
endmodule
```

Testbench -

```
module tb;
  parameter WIDTH = 4;


  reg clk;
  reg rstn;

  wire [WIDTH-1:0] out;


  johnson_ctr   u0 (.clk (clk),

               .rstn (rstn),

               .out (out));


  always #10 clk = ~clk;


  initial begin

    {clk, rstn} <= 0;


    $monitor ("T=%0t out=%b", $time, out);

    repeat (2) @(posedge clk);

    rstn <= 1;

    repeat (15) @(posedge clk);

    $finish;

  end

endmodule
```

Conclusion :

In this lab, we learned how ring and johnson counters work. We modeled and verified the functionality of these components. These components are widely used in a processor system design.