**Lab 10 Latches and Flip-Flops**

The objective of this lab is to model and investigate the behavior of various latches and flip-flops.

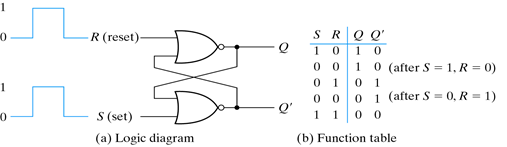
Sequential circuits are the digital circuits in which the output depends not only on the present input (like combinatorial circuits), but also on the past sequence of inputs. In effect, these circuits must be able to remember something about the past history of the inputs. Thus the timing concept is introduced and the clock signal provides the timing essence to sequential circuits. Latches and flip-flops are memory devices in sequential circuits, and are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. A flip-flop or latch is a circuit that has two stable states and can be used to store state information. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs.

**Laches**

Latch is a device with exactly two stable states: high-output and low-output. A latch has a feedback path, so information can be retained by the device. Therefore latches are volatile memory devices, and can store one bit of data for as long as the device is powered. As the name suggests, latches are used to "latch onto" information and hold in place.

1. **SR Latch**

An SR latch (Set/Reset) is an asynchronous device: it works independently of control signals and relies only on the state of the S and R inputs. The SR Latch a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled S for set and R for reset. The latch has two useful states. When output Q = 1, the latch is said to be in the set state. When Q = 0, it is the reset state. Outputs Q and Q’ are normally the complement to each other. However, when both inputs are equal to 1 at the same time, a condition in when both outputs are equal to 0 (rather than be mutually complementary) occurs. If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state. Therefore, setting both inputs to 1 is forbidden. Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed.



**Procedure**

1. Crate a Verilog module for the SR Latch using structural(gate)-level modeling

**module** SR\_latch (R, S, Q, Qprime);

**input** R, S;

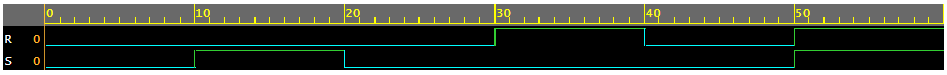
**output** Q, Qprime;

nor n0(Q,R,Qprime);

nor n1(Qprime,S,Q);

**endmodule**

1. Develop a testbench to test and validate the design. You should generate the input stimuli as shown in the below timing diagram.



**module test\_SR\_latch;**

**reg R, S; // inputs**

**wire Q, QPrime; // outputs**

**// module SR\_latch (R, S, Q, Qprime);**

**SR\_latch uut(R, S, Q, Qprime);**

**initial**

**begin**

**$dumpfile("dump.vcd");**

**$dumpvars(1, test\_SR\_latch);**

**// display the inputs and outputs**

**$monitor("R=%b S=%b Q=%b Q'=%b", S, R, Q, Qprime);**

**R = 0; S = 0;**

**#10 R = 0; S = 1;**

**#10 R = 0; S = 0;**

**#10 R = 1; S = 0;**

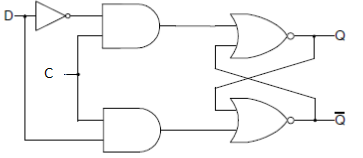
**#10 R = 1; S = 1;**

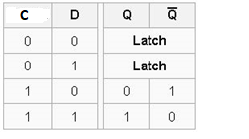
**#10 $finish**

**end**

1. **D Latch**

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 C (Control/Enable) line is high, otherwise the output is whatever the D input was when the C input was last high. This is why it is also known as a transparent latch - when C is asserted, the latch is said to be "transparent" - it signals propagate directly through it as if it isn't there.





**Procedure**

1. Model the D Latch in behavioral modeling

**module D\_Latch (D, C, Q, Qprime);**

**input D, C;**

**output Q, Qprime;**

**// redeclare Q AND Q' TO BE THE TYPE OF REG**

**reg Q = 0, Qprime;**

**always @ (D or C)**

**begin**

**if (C)**

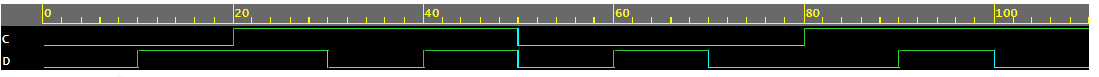
**Q <= D;**

**Qprime <= ~D;**

**end**

**endmodule**

1. Develop a testbench to test and validate the design. You should generate the input stimuli as shown in the below timing diagram.



**module test\_D\_Latch;**

**reg D, C;**

**wire Q, QPrime;**

**// module D\_Latch (D, C, Q, Qprime);**

**D\_Latch uut(D, C, Q, QPrime)**

**initial**

**begin**

**$dumpfile("dump.vcd");**

**$dumpvars(1, test\_D\_Latch);**

**// display the inputs and outputs**

**$monitor( "D=%b C=%b Q=%b Q'=%b", D,C,Q,QPrime);**

**C=0; D=0;**

**#10 C=0; D=1;**

**#10 C=1; D=0;**

**#10 C=1; D=1;**

**#10 $finish;**

**end**

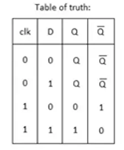
**endmodule**

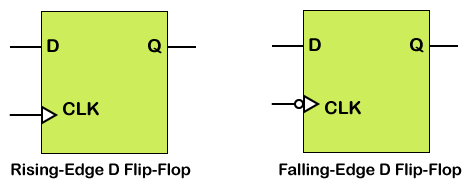
**Flip-Flops**

Flip-flops are clocked circuits whose output may change on an active edge of the clock signal based on its input. Unlike latches, which are transparent and in which output can change when the gated signal is asserted upon the input change, flip-flops normally would not change the output upon input change even when the clock signal is asserted. Flip-flops are widely used in synchronous circuits.

1. **D Flip-Flop**

The D flip-flop is a widely used type of flip-flop. It is also known as a data or delay flip-flop. The D flip-flop captures the value of the D-input at a definite portion of the clock cycle (such as the rising edge of the clock). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell or a delay line. The active edge in a flip-flop could be rising or falling. The following figure shows rising (also called positive) edge triggered D flip-flop and falling (negative edge) triggered D flip-flop.





**Procedure**:

1. Crate a Verilog module for a rising edge triggered D flip-flop in behavioral modeling

**module D\_ff (Clk, D, Q, Qprime);**

**input Clk, D;**

**output reg Q, Qprime;**

**always @ (posedge Clk)**

**begin**

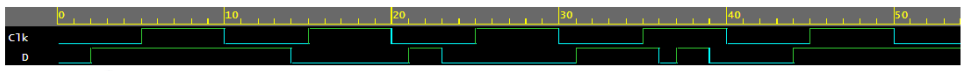
**Q <= D;**

**Qprime <= ~D;**

**end**

**endmodule**

1. Develop a testbench to test and validate the design. You should generate the input stimuli as shown in the below timing diagram.



module testDFF;

reg Clk, D;

wire Q, Qprime;

D\_ff uut(Clk, D, Q, Qprime);

// generate the clock signal

initial

begin

Clk = 0;

forever #5 Clk = ~Clk;

#200 $finish;

end

initial

begin

$dumpfile("dump.vcd");

$dumpvars(1, testDFF);

// display the inputs and outputs

$monitor( "Clk = %b D = %b Q = %b Q' = %b", Clk,D,Q,Qprime);

D=0;

#10 D = 1;

#10 D = 0;

#10 D = 1;

#10 D = 0;

#10 D = 1;

#10 D = 0;

#10 $finish;

end

endmodule

1. **D Flip-Flop with Set/Reset**

Often it is necessary to have the synchronous element to start with a defined output. It is also desired and required in some circuits to force the synchronous element to a known output ignoring input at the D input. The D flip-flop discussed above can be modified to have such functionality. Such D flip-flop is known as D flip-flop with synchronous set and reset capabilities if the desired output is obtained on the active edge of the clock, otherwise it is viewed to have asynchronous preset and clear. The model of the DFF with synchronous reset is shown below.

**module** D\_ff\_with\_synch\_reset (D, Clk, reset, Q, Qprime);

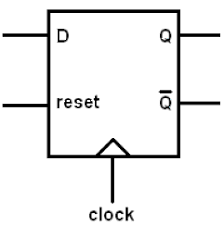
**input** D, Clk, reset;

**output** Q, Qprime;

**reg** Q =0, Qprime;

**always** @(**posedge** Clk)

**if** (reset)

**** **begin**

Q <= 1'b0;

Qprime <= 1'b1;

**end**

**else**

**begin**

Q <= D;

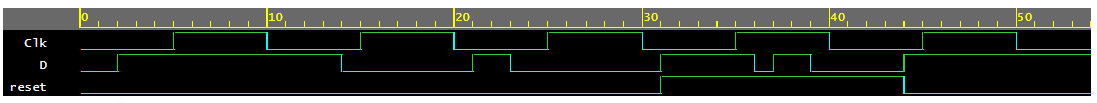
Qprime <= ~D;

**end**

**endmodule**

**Procedure**:

Develop a testbench to test and validate the design. You should generate the input stimuli as shown in the below timing diagram. **You can simply copy the test cases for the DFF above and then add the test cases for the reset.**



**module test\_Dff\_with\_synch\_reset;**

**reg D, Clk, reset;**

**wire Q, Qprime;**

**D\_ff\_with\_synch\_reset uut(D, Clk, reset, Q, Qprime);**

**initial**

**begin**

**Clk = 0;**

**forever #5 Clk = ~Clk;**

**#200 $finish;**

**end**

**initial**

**begin**

**$dumpfile("dump.vcd");**

**$dumpvars(1, test\_Dff\_with\_synch\_reset);**

**$monitor( "D = %b Clk = %b reset = %b Q = %b Q' = %b", D, Clk, reset, Q, Qprime);**

**D = 0; reset = 0;**

**#10 D= 1;**

**#10 D=0;**

**#10 D = 1; reset = 1;**

**#10 D = 0;**

**#10 D = 1;**

**#10 D = 0;**

**#10 D = 1; reset = 0;**

**#10 D = 0;**

**#10 $finish;**

**end**

**endmodule**

**Submission Instructions:**

Lab work submission

1. Take screenshots of all your waveforms.
2. Add the following information as comments to the beginning of your code. Make sure to click the “Save” button to save your project, then take a screenshot of your code.

// Author: Name

// Lab 10: put the title here

// Link to your project

1. Copy the link of your design from the address bar of the browser.
2. On the Blackboard, click on Lab 10. Attach the screenshot from the first two steps and paste the link from Step 3 into the Comments area, then click the “Submit” button.

Shape

Description automatically generated with low confidence

Rectangle

Description automatically generated with low confidence

Graphical user interface

Description automatically generated

Graphical user interface

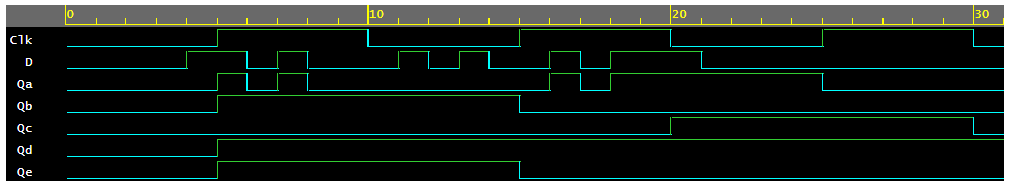
Description automatically generated with low confidence

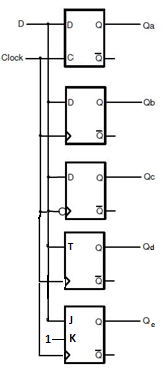
|  |
| --- |
| // Author: Gianna Rose  // Lab 10  // https://www.edaplayground.com/x/NLTd  // module test\_SR\_latch;  // reg R, S;  // wire Q, Qprime;    // // module SR\_latch (R, S, Q, Qprime);  // SR\_latch uut(R, S, Q, Qprime);    // initial  // begin  // $dumpfile("dump.vcd");  // $dumpvars(1, test\_SR\_latch);    // // display the inputs and outputs  // $monitor("R = %b S = %b Q = %b Q' = %b", S, R, Q, Qprime);    // R = 0; S = 0;  // #10 R = 0; S = 1;  // #10 R = 0; S = 0;  // #10 R = 1; S = 0;  // #10 R = 1; S = 1;  // #10 $finish;    // end  // endmodule  // OUTPUT  // R = 0 S = 0 Q = x Q' = x  // R = 1 S = 0 Q = 1 Q' = 0  // R = 0 S = 0 Q = 1 Q' = 0  // R = 0 S = 1 Q = 0 Q' = 1  // R = 1 S = 1 Q = 0 Q' = 0  // --------------------------------------------------  // module test\_D\_Latch;  // reg D, C;  // wire Q, Qprime;  // // module D\_Latch (D, C, Q, Qprime);  // D\_Latch uut(D, C, Q, Qprime);  // initial  // begin  // $dumpfile("dump.vcd");  // $dumpvars(1, test\_D\_Latch);    // // display the inputs and outputs  // $monitor( "D = %b C = %b Q = %b Q' = %b", D, C, Q, Qprime);  // C=0; D=0;  // #10 C = 0; D = 1;  // #10 C = 1; D = 0;  // #10 C = 0; D = 0;  // #10 C = 1; D = 1;  // #10 $finish;    // end  // endmodule  // OUTPUT  // D = 0 C = 0 Q = 0 Q' = x  // D = 1 C = 0 Q = 0 Q' = x  // D = 0 C = 1 Q = 0 Q' = 1  // D = 0 C = 0 Q = 0 Q' = 1  // D = 1 C = 1 Q = 1 Q' = 0  // --------------------------------------------------  // module testDFF;  // reg Clk, D;  // wire Q, Qprime;  // D\_ff uut(Clk, D, Q, Qprime);  // // generate the clock signal  // initial  // begin  // Clk = 0;  // forever #5 Clk = ~Clk;  // #200 $finish;  // end  // initial  // begin  // $dumpfile("dump.vcd");  // $dumpvars(1, testDFF);  // // display the inputs and outputs  // $monitor( "Clk = %b D = %b Q = %b Q' = %b", Clk, D, Q, Qprime);  // D=0;  // #10 D = 1;  // #10 D = 0;  // #10 D = 1;  // #10 D = 0;  // #10 D = 1;  // #10 D = 0;  // #10 $finish;  // end  // endmodule  // OUTPUT  // Clk = 0 D = 0 Q = x Q' = x  // Clk = 1 D = 0 Q = 0 Q' = 1  // Clk = 0 D = 1 Q = 0 Q' = 1  // Clk = 1 D = 1 Q = 1 Q' = 0  // Clk = 0 D = 0 Q = 1 Q' = 0  // Clk = 1 D = 0 Q = 0 Q' = 1  // Clk = 0 D = 1 Q = 0 Q' = 1  // Clk = 1 D = 1 Q = 1 Q' = 0  // Clk = 0 D = 0 Q = 1 Q' = 0  // Clk = 1 D = 0 Q = 0 Q' = 1  // Clk = 0 D = 1 Q = 0 Q' = 1  // Clk = 1 D = 1 Q = 1 Q' = 0  // Clk = 0 D = 0 Q = 1 Q' = 0  // Clk = 1 D = 0 Q = 0 Q' = 1  // Clk = 0 D = 0 Q = 0 Q' = 1  // --------------------------------------------------  // module test\_Dff\_with\_synch\_reset;  // reg D, Clk, reset;  // wire Q, Qprime;    // D\_ff\_with\_synch\_reset uut(D, Clk, reset, Q, Qprime);    // initial  // begin  // Clk = 0;  // forever #5 Clk = ~Clk;  // #200 $finish;  // end  // initial  // begin  // $dumpfile("dump.vcd");  // $dumpvars(1, test\_Dff\_with\_synch\_reset);  // $monitor( "D = %b Clk = %b reset = %b Q = %b Q' = %b", D, Clk, reset, Q, Qprime);  // D = 0; reset = 0;  // #10 D= 1;  // #10 D=0;  // #10 D = 1; reset = 1;  // #10 D = 0;  // #10 D = 1;  // #10 D = 0;  // #10 D = 1; reset = 0;  // #10 D = 0;  // #10 $finish;  // end  // endmodule  // OUTPUT  // D = 0 Clk = 0 reset = 0 Q = 0 Q' = x  // D = 0 Clk = 1 reset = 0 Q = 0 Q' = 1  // D = 1 Clk = 0 reset = 0 Q = 0 Q' = 1  // D = 1 Clk = 1 reset = 0 Q = 1 Q' = 0  // D = 0 Clk = 0 reset = 0 Q = 1 Q' = 0  // D = 0 Clk = 1 reset = 0 Q = 0 Q' = 1  // D = 1 Clk = 0 reset = 1 Q = 0 Q' = 1  // D = 1 Clk = 1 reset = 1 Q = 0 Q' = 1  // D = 0 Clk = 0 reset = 1 Q = 0 Q' = 1  // D = 0 Clk = 1 reset = 1 Q = 0 Q' = 1  // D = 1 Clk = 0 reset = 1 Q = 0 Q' = 1  // D = 1 Clk = 1 reset = 1 Q = 0 Q' = 1  // D = 0 Clk = 0 reset = 1 Q = 0 Q' = 1  // D = 0 Clk = 1 reset = 1 Q = 0 Q' = 1  // D = 1 Clk = 0 reset = 0 Q = 0 Q' = 1  // D = 1 Clk = 1 reset = 0 Q = 1 Q' = 0  // D = 0 Clk = 0 reset = 0 Q = 1 Q' = 0  // D = 0 Clk = 1 reset = 0 Q = 0 Q' = 1  // D = 0 Clk = 0 reset = 0 Q = 0 Q' = 1 |

|  |
| --- |
| // Author: Gianna Rose  // Lab 10  // https://www.edaplayground.com/x/NLTd  module SR\_latch (R, S, Q, Qprime);  input R, S;  output Q, Qprime;  nor n0(Q,R,Qprime);  nor n1(Qprime,S,Q);  endmodule  module D\_Latch (D, C, Q, Qprime);  input D, C;  output Q, Qprime;  reg Q = 0, Qprime;    always @ (D or C)  begin  if (C) begin  Q <= D;  Qprime <= ~D;  end  end  endmodule  module D\_ff (Clk, D, Q, Qprime);  input Clk, D;  output reg Q, Qprime;    always @ (posedge Clk)  begin  Q <= D;  Qprime <= ~D;  end  endmodule    module D\_ff\_with\_synch\_reset (D, Clk, reset, Q, Qprime);  input D, Clk, reset;  output Q, Qprime;  reg Q =0, Qprime;  always @(posedge Clk)  if (reset)  begin  Q <= 1'b0;  Qprime <= 1'b1;  end  else  begin  Q <= D;  Qprime <= ~D;  end  endmodule |

**You do not need to write a lab report for this lab but do the following homework and submit the screenshots and the link to the Blackboard under Lab 10 report submission.** It is due one week after the lab is done.

**Homework**: the following circuit and timing diagrams illustrate the differences between D-latch, rising edge triggered D flip-flop, falling edge triggered D flip-flop, T flip-flop, and JK flip-flop (the input K hardwired to 1). Write Verilog design codes and a testbench to reproduce the following waveforms. You need to initialize Q to 0 at the declaration (e.g. reg Q = 0;) for each flip-flop module. You can test all the flip-flops at once in one testbench.





Qa = D (if C = 1)

Qb = D (positive edge triggered)

Qc = D (negative edge triggered)

Qd = T ⊕ Qd (positive edge triggered)

Qe = J Qe’+ K’ Qe (positive edge triggered), K =1