Experiment No 12

Implementation of Sequential Logic using Verilog HDL

12.1 Objectives:

After completing this experiment, student will be able to:

- Implement Sequential Circuits in Verilog
- Writing of Design Module & Stimulus Module of Flip-flops in Verilog
- Result display and timing waveform generation in ModelSim

12.2 Background Theory

In a combinational circuit, the output depends only on the present value of the inputs. However, the output of a sequential circuit depends on not only the current value of the inputs but also the state of the circuit. The state of the circuit is determined by the previous values of the inputs. Hence, a sequential circuit has memory and its output depends on the sequence of the past inputs. These circuits use memory elements, such as flip-flops (FFs), to store the current state of the system.

12.3 Verilog Implementation of Sequential Circuits

There are two structured procedure statements in Verilog: always and initial. These statements are the two most basic statements in behavioral modeling. All other behavioral statements can appear only inside these structured procedure statements. Verilog is a concurrent programming language unlike the C programming language, which is sequential in nature. Activity flows in Verilog run in parallel rather than in sequence. Each always and initial statement represents a separate activity flow in Verilog. Each activity flow starts at simulation time 0. The statements always and initial cannot be nested. The fundamental difference between the two statements is explained in the following sections.

12.3.1 Initial Statement

All statements inside an initial statement constitute an initial block. An initial block starts at time 0, executes exactly once during a simulation, and then does not execute again. If there are multiple initial blocks, each block starts to execute concurrently at time 0. Each block finishes execution independently of other blocks as shown below. Multiple behavioral statements must be grouped, typically using the keywords begin and end. If there is only one behavioral statement, grouping is not necessary. This is similar to the begin-end blocks in Pascal programming language or the { } grouping in the C programming language. E.g.

```
module stimulus;
reg x,y, a,b, m;
initial
m = 1'b0; //single statement; does not need to be grouped
initial
begin
#5 a = 1'b1; //multiple statements; need to be grouped
#25 b = 1'b0;
end
initial
begin
#10 x = 1'b0;
#25 y = 1'b1;

end
```

In the above example, the two initial statements start to execute in parallel at time 0. If a delay #<delay> is seen before a statement, the statement is executed <delay> time units after the current simulation time.

12.3.2 Always Statement

All behavioral statements inside an always statement constitute an always block. The always statement starts at time 0 and executes the statements in the always block continuously in a looping fashion. This statement is used to model a block of activity that is repeated continuously in a digital circuit. An example is a clock generator module that toggles the clock signal every half cycle. In real circuits, the clock generator is active from time 0 to as long as the circuit is powered on.

```
module clock_gen (output reg clock);
//Initialize clock at time zero
initial
clock = 1'b0;
//Toggle clock every half-cycle (time period = 20)
always
#10 clock = ~clock;
initial
#1000 $finish;
Endmodule
```

The always statement starts at time 0 and executes the statement clock = ~clock every 10 time units. Notice that the initialization of clock has to be done inside a separate initial statement. If we put the initialization of clock inside the always block, clock will be initialized every time the always is entered. Also, the simulation must be halted inside an initial statement. If there is no $stop or $finish statement to halt the simulation, the clock generator will run forever.

12.4 Verilog Implementation of Flip Flops

When dealing with a large sequential circuit, the design problem becomes much more approachable if we use the synchronous methodology rather than asynchronous approach. In a synchronous circuit, all the storage elements are triggered by the same clock signal. This gives us a better control over the system because, in this case, we know when the data is going to be sampled by the storage elements. Since all of the storage elements of a synchronous system are connected to the same clock, we can model the system as shown in Figure 12.1.

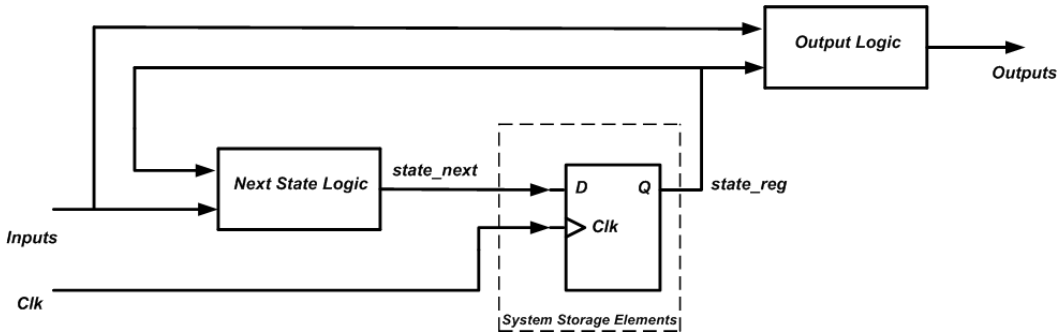


Figure 12.1: Synchronous System

In this model, the dashed box represents all the storage elements of the system (D-type FFs in this example). The blocks outside the dashed box are combinational circuits.

The "Next State Logic" processes the "Inputs" and the current state of the system, represented by "state_reg", to determine the next state of the system ("state next"). With the upcoming rising clock edge, "state next" will be stored on the FFs. The "Output Logic" block is again a combinational circuit that processes the "Inputs" and "state_reg" to determine the system outputs. Separating a synchronous system to storage elements and some combinational circuits as shown in Figure 12.1 helps us to more easily find the HDL description of the system. We only need to describe some combinational circuits and connect them to the memory elements.

Figure 12.2 below shows the symbol for positive-edge (Figure 12.2-a) and negative-edge (Figure 12.2-b) DFFs. At the clock edge, the input (D) of a DFF is sampled and passed on to the output (Q). For a positive-edge DFF, the sampling edge is the clock rising edge (Figure 12.2-a) while a negative-edge DFF is sensitive to the falling edge of clock. An FF usually has a reset (rst) that can be used to initialize the system to a known state.

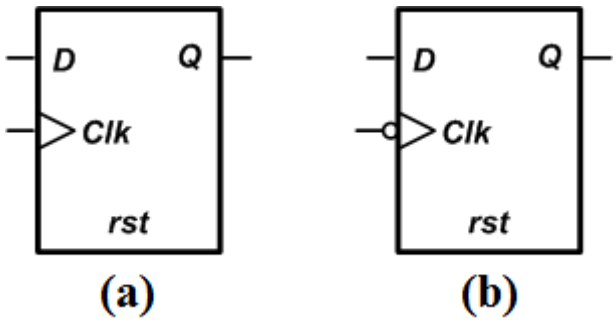


Figure 12.2 D-Flip flop block diagram representation

12.4.1 Verilog code for Flip flop

```
1       module D_FF
2               (
3                       input wire clk, reset,
4                       input wire d,
5                       output reg q
6                );


7               always @(posedge clk, posedge reset)
8                       if (reset)
9                               q <= 1'b0;
10                      else
11                              q <= d;

12      endmodule
```

Line 7 of this code uses the “posedge” keyword before the “clk” and “reset” inputs in the sensitivity list of the “always” block. “Posedge”, which stands for positive edge, tells the synthesis tool that the “always” block should be activated at the rising edge of the “clk” and “reset” signals. Note that the rising edge of both “clk” and “reset” are in the sensitivity list. Hence, a rising edge of “reset” will activate the “always” block irrespective of the “clk” signal condition. In other words, the “reset” of this DFF is asynchronous.

Note that we use non-blocking assignment (<=) when inferring flip-flops. Improper use of blocking (=) and non-blocking (<=) assignments can lead to undesired functionality. We can simply separate the combinational part of the system from its memory elements and follow these two guidelines:

- Use blocking assignments to describe combinational circuits.
- Use non-blocking assignments to infer FFs.

12.5 Generation of Clock and Reset Signal in Verilog

Clocks are the main synchronizing events to which all other signals are referenced. If the RTL is in Verilog, the Clock generator is written in Verilog even if the Test Bench is written in other languages like Vera, Specman or SystemC. Clock can be generated many ways. Some test benchs need more than one clock generator. So testbench need clock with different phases some other need clock generator with jitter. The very first transition of clock at time zero may be perceived as a transition because clock has an unknown value before time zero and gets assigned to a value at time zero. How this time zero clock transition is perceived is simulator dependent, and thus care must be taken. Following examples show simple clock generators with 50% duty cycles.

```
EXAMPLE:
initial clk = 0;
always #10 clk = ~clk;

EXAMPLE:
always
begin
clk = 0;
#10;
clk = 1;
#10;
end

EXAMPLE:
always
begin
clk = 0;
forever #10 clk = ~clk;
end
```

Different testbenchs need different clock periods. It is beneficial to use parameters to represent the delays, instead of hard coding them. For example, to generate a clock starting with zero that has a 50% duty cycle, the following code can be used:

```
EXAMPLE:
module Tb();
reg clock;
integer no_of_clocks;

parameter CLOCK_PERIOD = 5;
initial no_of_clocks = 0;
initial clock = 1'b0;

always #(CLOCK_PERIOD/2) clock = ~clock;

always@(posedge clock)
no_of_clocks = no_of_clocks +1 ;

initial
begin
#50000;
$display("End of simulation time is %d , total number of clocks seen is %d expected
is %d",$time,no_of_clocks,($time/5));
$finish;
end
endmodule
```

Sometimes a transition on any one of multiple signals or events can trigger the execution of a statement or a block of statements. Mostly Flip-flop triggered on two events such as clock and reset to handle multiple event in a circuit event or control is used. This is expressed as an OR of events or signals. The list of events or signals expressed as an OR is also known as a sensitivity list. Sensitivity lists can also be specified using the "," (comma) operator instead of the ‘or’ operator. Following example shows how the above example can be rewritten using the

comma operator. Comma operators can also be applied to sensitivity lists that have edge-sensitive triggers.

```
//A positive edge triggered D flipflop with asynchronous falling
//reset can be modeled as shown below
always @(posedge clk, negedge reset) //Note use of comma operator
if(!reset)
q <=0;
else
q <=d;
```

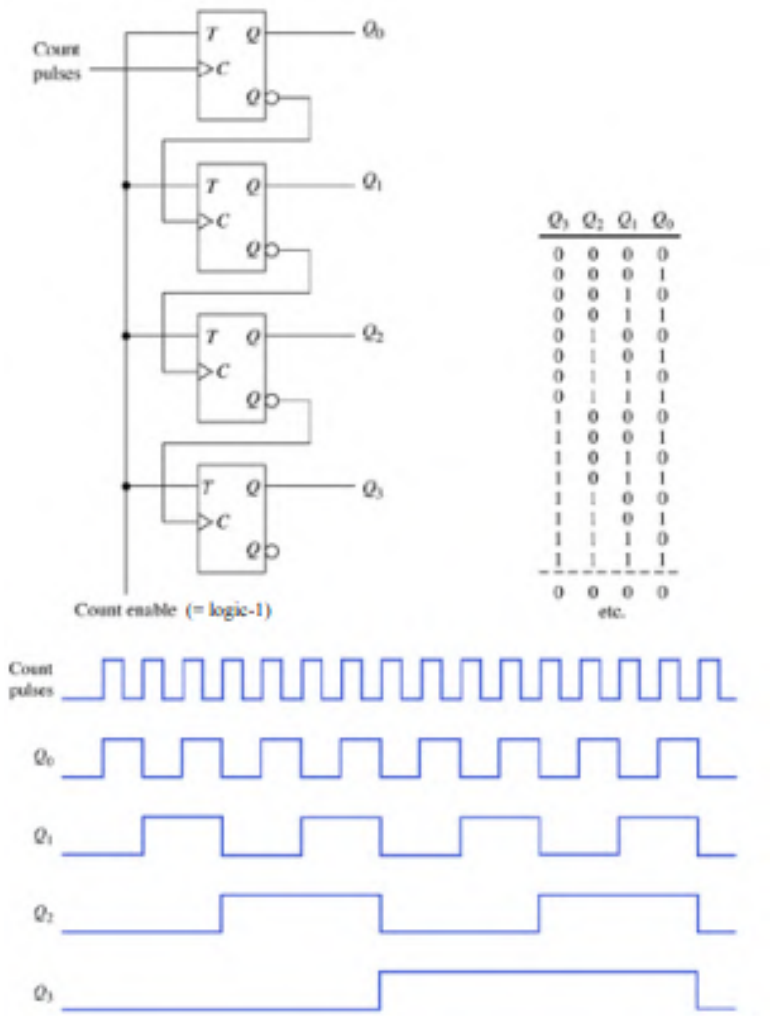
12.6 Lab Activities

12.6.1 Task-1:

Implement J-K and T-Flip-flop in Verilog and write Stimulus for testing and verification of the design.

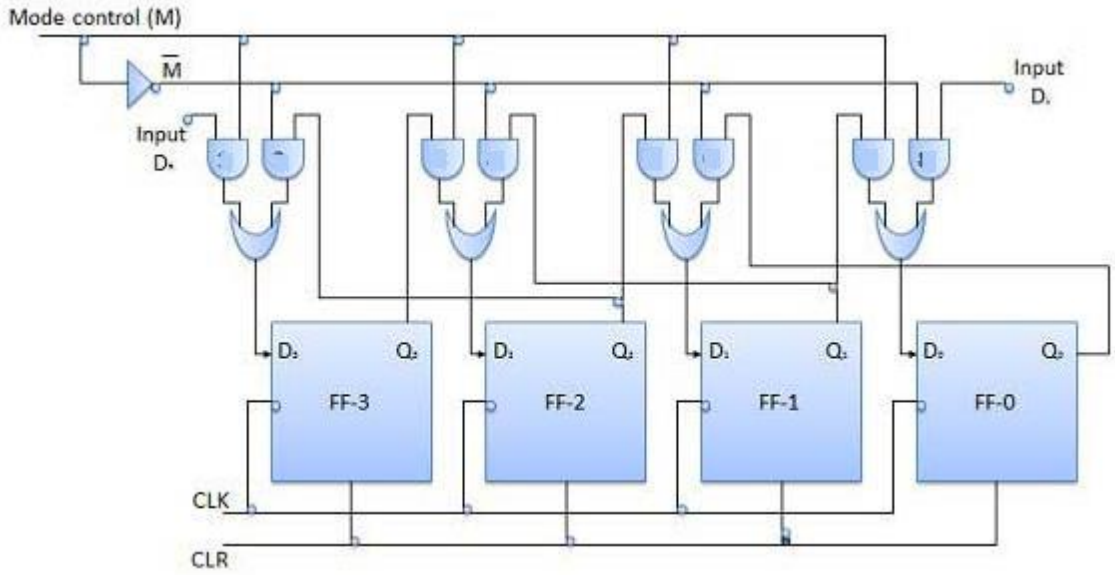
12.6.2 Task-2:

Implement a 4 bit Ripple counter as shown in the figure below in Verilog, Write Stimulus to test the design and attach output waveform for verification.



12.6.3 Task-3:

Implement bi-directional shift Register in Verilog, Write stimulus to test the design and attach output waveform for verification.



Lab Exercise and Summary

Summary should cover Introduction, Procedure, Data Analysis and Evaluation.

Student's Signature: _____ **Date: _____**

LABORATORY SKILLS ASSESMENT (Psychomotor)

Total Marks: 100

Criteria (Max Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S< 70%	Level 3 70% ≤ S< 90%	Level 4 90%≤ S ≤100%	Score (S)
Procedural Awareness (20)	Selects inappropriate skills and/or strategies to write and compile the programs.	Selects and applies partially appropriate skills and/or strategies required by the programs.	Selects and applies the considerably appropriate strategies and/or skills specific to the programs.	Selects and applies the completely appropriate strategies and/or skills specific to the programs.	
Practical Implementation (20)	Makes major critical errors in applying procedural knowledge related to the Implementation of Sequential Logic using Verilog HDL	Makes numerous critical errors in applying procedural knowledge related to the Implementation of Sequential Logic using Verilog HDL	Makes minor non-critical errors in applying procedural knowledge related to the Implementation of Sequential Logic using Verilog HDL	Applies the procedural knowledge in optimized ways the Implementation of Sequential Logic using Verilog HDL	
Program Logic (20)	Program logic has many errors with majority of contradictory conditions.	Program logic has some errors with several contradictory conditions.	Program logic is mostly correct, but may contain occasional errors or redundant/ contradictory conditions.	Program logic is correct, with no known errors, and no redundant or contradictory conditions.	
Syntax Correctness and Results (20)	Program does not follow proper syntax of the Implementation of Sequential Logic using Verilog HDL and does not produce desired results for most inputs.	Program partially follows the proper syntax of the Implementation of Sequential Logic using Verilog HDL and produces appropriate results for few inputs. .	Program adequately follows the proper syntax of the Implementation of Sequential Logic using Verilog HDL and produces appropriate results for most inputs.	Program completely follows proper syntax of the Implementation of Sequential Logic using Verilog HDL and produces appropriate results for all inputs tested.	
Use of Software Tool (10)	Uses software tool, with limited competence.	Uses software tool, with some competence.	Uses software tool, with considerable competence.	Uses software tool, with a high degree of competence.	
Safety (10)	Requires Constant reminders to follow safety procedures.	Requires some reminders to follow safety procedures.	Follows safety procedures with only minimal reminders.	Routinely follows safety procedures.	
				Marks Obtained	

Instructor's Signature: _____ **Date: _____**

LABORATORY SKILLS ASSESMENT (Affective)

Total Marks: 40

Criteria (Max. Marks)	Level 1 0% ≤ S < 50%	Level 2 50% ≤ S < 70%	Level 3 70% ≤ S < 90%	Level 4 90% ≤ S ≤ 100%	Score (S)
Introduction (5)	Very little background information provided or information is incorrect	Introduction is brief with some minor mistakes	Introduction is nearly complete, missing some minor points	Introduction complete and well-written; provides all necessary background principles for the experiment	
Procedure (5)	Many stages of the procedure are not entered on the lab report.	Many stages of the procedure are entered on the lab report.	The procedure could be more efficiently designed but most stages of the procedure are entered on the lab report.	The procedure is well designed and all stages of the procedure are entered on the lab report.	
Data Record (10)	Data is brief and missing significant pieces of information.	Data provides some significant information and has few critical mistakes.	Data is almost complete but has some minor mistakes.	Data is complete and relevant. Tables with units are provided. Graphs are labeled. All questions are answered correctly.	
Data Analysis (10)	Data is presented in very unclear manner. Error analysis is not included.	Data is presented in ways (charts, tables, graphs) that are not clear enough. Error analysis is included.	Data is presented in ways (charts, tables, graphs) that can be understood and interpreted. Error analysis is included.	Data are presented in ways (charts, tables, graphs) that best facilitate understanding and interpretation. Error analysis is included.	
Report Quality (10)	Report contains many errors.	Report is somewhat organized with some spelling or grammatical errors.	Report is well organized and cohesive but contains some grammatical errors.	Report is well organized and cohesive and contains no grammatical errors. Presentation seems polished.	
				Marks Obtained	

LABORATORY SKILLS ASSESSMENT (Cognitive)

Total Marks: 10

(If any) **Marks Obtained**	

Instructor's Signature: _____ **Date: _____**