**Department of Electrical Engineering**

**UET, Lahore**

**Digital Systems Laboratory**

# Lab # 11(b)

# Implementation of HDL Models of the Latches and Flip Flops

**Objective:**

In this lab you will implement the HDL models of the latches and flip flops.

**Theoretical Background:**

Read the article 5.6 of the book: **"Digital Design by M. Morris Mano and Michael D. Ciletti 5th Edition"**

**Quick overview:**

There are two types of 'Abstract Behaviors' in Behavioral Modelling, viz:

- Single-pass behavior, keyword: **'initial'**, one-time execution of block statement, non-synthesizable, usually used for stimulus signal in test bench/ test fixture file.
- Cyclic behavior, keyword: **'always'**, indefinite repetitive execution of block statement unless simulation is halted, synthesizable, used for both stimulus signal and circuit modelling.

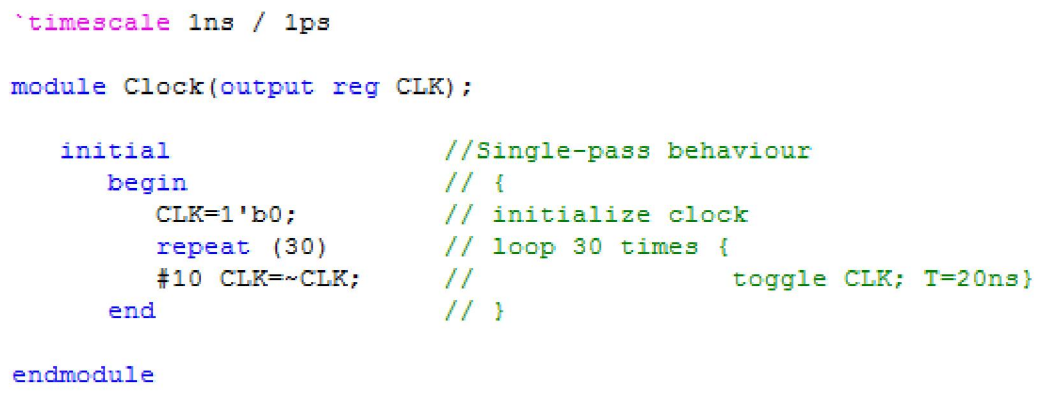
Note: A module may contain an arbitrary number of initial or always behavioral statements. They execute concurrently with respect to each other, starting at time 0, and may interact through common variables.

Note: **'begin'** and **'end'** keywords mark the start and end of a block of code.

We'll start by developing most fundamental stimulus signal of sequential digital logic circuits, the clock.

**Lab Task (a):**

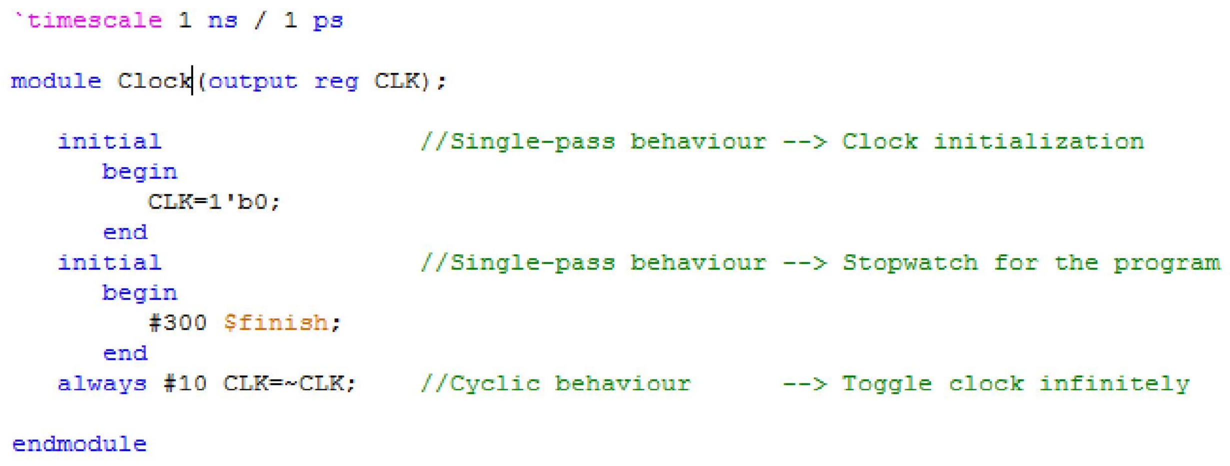
Simulate behavioral model of the following Verilog HDL module in Test Fixture.



```
`timescale 1ns / 1ps

module Clock(output reg CLK);

   initial                  //Single-pass behaviour
      begin                 // {
         CLK=1'b0;          // initialize clock
         repeat (30)        // loop 30 times {
         #10 CLK=~CLK;      //                toggle CLK; T=20ns}
      end                   // }

endmodule
```

**Lab Task (b):**

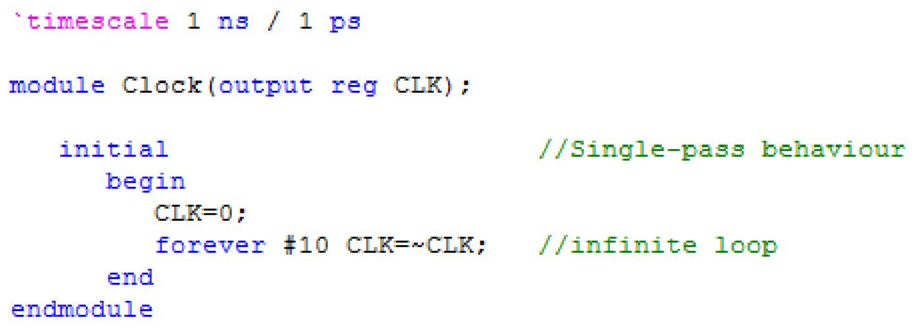
Simulate behavioral model of the following Verilog HDL module in Test Fixture.



```
`timescale 1 ns / 1 ps

module Clock(output reg CLK);

   initial                  //Single-pass behaviour --> Clock initialization
      begin
         CLK=1'b0;
      end
   initial                  //Single-pass behaviour --> Stopwatch for the program
      begin
         #300 $finish;
      end
   always #10 CLK=~CLK;     //Cyclic behaviour      --> Toggle clock infinitely

endmodule
```

**Lab Task (c):**

Simulate behavioral model of the following Verilog HDL module (Free-running clock) in Test Fixture.



```
`timescale 1 ns / 1 ps

module Clock(output reg CLK);

   initial                        //Single-pass behaviour
      begin
         CLK=0;
         forever #10 CLK=~CLK;    //infinite loop
      end
endmodule
```

There are two 'Control Operators', viz:

- Delay control operator, key symbol: **'#'** suspends execution of statements until a specified time has elapsed. We have already seen the use of this operator in previous examples.
- Event control operator, key symbol: **'@'**, suspends an activity until an event has occurred. The general form of statement containing event control operator is:

```
always @ (event control expression) begin
    // Procedural assignment statements that execute when the condition is met
end
```

An event can be:

- Unconditional change in value of a variable (e.g. @A, for level triggered devices)
- A specified transition of a signal value (e.g. **@ (posedge clock),** for edge triggered devices)

There are two types of variable assignments, viz:

- Procedural assignment: made only when an assignment statement is executed
- Continuous Assignment: triggered by event occurrence on other variables [Beyond your scope for now]

<u>Note:</u> The variables in the left-hand side of the procedural statements must be of the **'reg'** data type and must be declared as such.

There are two types of Procedural assignments, viz:

- Blocking, key operator: **'='**, executed sequentially in the order they are listed, cannot be used for modelling latched behavior.
- Non-blocking, key operator: **'<='**, executed concurrently by evaluating the set of expressions, used for modelling latched behavior. (We'll be using this assignment mostly)

Now we are ready to model various latches and flip-flops in HDL. We'll employ all of the above mentioned concepts in our lab. You are required to show output of all the models presented below using a clock stimulus in test fixture file.

Starting with the D latch and flip-flop:

- For D latch we know that: Output = Input when latch is enabled.
- For D flip-flop we have the characteristic equation: Q (t+1) =D

**Lab Task (d):**

Simulate behavioral model of the following Verilog HDL module (D Latch).

```
// Description of D latch
module D_latch (Q, D, enable);
   output Q;
   input D, enable;
   reg Q;
   always @ (enable or D)
   if (enable) Q <= D; // Same as: if (enable == 1)
endmodule
// Alternative syntax (Verilog 2001, 2005)
module D_latch (output reg Q, input enable, D);
   always @ (enable, D)
   if (enable) Q <= D; // No action if enable not asserted
endmodule
```

**Lab Task (e):**

Simulate behavioral model of the following Verilog HDL module (D Flip-flop without reset).

```
// D flip-fl op without reset
module D_FF (Q, D, Clk);
   output Q;
   input D, Clk;
   reg Q;
   always @ ( posedge Clk)
   Q <= D;
endmodule
```

**Lab Task (f):**

Simulate behavioral model of the following Verilog HDL module (D Flip-flop with asynchronous reset).

```
// D flip-fl op with asynchronous reset (V2001, V2005)
module DFF ( output reg Q, input D, Clk, rst);
   always @ ( posedge Clk, negedge rst)
   if (!rst) Q <= 1'b0; // Same as: if (rst == 0)
   else Q <= D;
endmodule
```

From now on, we'll prefer to have an asynchronous reset input in all of our sequential circuits. Moreover, since we know that:

$$Q(t+1) = Q \oplus T \qquad \text{for a } T \text{ flip-flop}$$
$$Q(t+1) = JQ' + K'Q \qquad \text{for a } JK \text{ flip-flop}$$

We can make T and JK flip-flops using D flip-flops.

**Lab Task (g):**

Simulate behavioral model of the following Verilog HDL module (T Flip-flop with asynchronous reset).

```
// T flip-flop from D fl ip-fl op and gates
module TFF (Q, T, Clk, rst);
   output Q;
   input T, Clk, rst;
   wire DT;
   assign DT = Q ^ T ; // Continuous assignment
// Instantiate the D fl ip-fl op
DFF TF1 (Q, DT, Clk, rst);
endmodule
```

**Lab Task (h):**

Simulate behavioral model of the following Verilog HDL module (JK Flip-flop with asynchronous reset).

```
// JK fl ip-fl op from D fl ip-fl op and gates (V2001, 2005)
module JKFF ( output reg Q, input J, K, Clk, rst);
   wire JK;
   assign JK = (J & ~Q)|(~K & Q);
   // Instantiate D fl ip-fl op
   DFF JK1 (Q, JK, Clk, rst);
endmodule
```

You can also describe the flip-flop by using the characteristic table rather than the characteristic equation.

**Lab Task (i):**

Simulate behavioral model of the following Verilog HDL module (JK Flip-flop).

```
// Functional description of JK fl ip-fl op (V2001, 2005)
module JK_FF ( input J, K, Clk, output reg Q, output Q_b);
   assign Q_b = ~ Q ;
   always @ ( posedge Clk)
   case ({J,K})
      2'b00: Q <= Q;
      2'b01: Q <= 1'b0;
      2'b10: Q <= 1'b1;
      2'b11: Q <= !Q;
   endcase
endmodule
```

**Assignment:**

- Add an asynchronous reset input to the module of lab task (i).
- Implement the T flip-flop using its characteristic table.