PUE 3141: ADVANCED DIGITAL SYSTEMS DESIGN

LECTURE 3

20/07/2022

BY DR. MUTUGI KIRUKI

Topics:

i) Sequential Logic Designs with Verilog HDL

References:

i) Mano, M.M., & Ciletti, M.D. (2007) *Digital Design* (4th Edition). Pearson.

Sequential Logic HDL

Recap on Verilog syntax and keywords

initial:

- Declares a single-pass behavior and specifies a single statement or a block statement.
- Expires after the associated statement executes.
- Usually used to prescribe stimulus signals in a test bench – never to model the behavior of a circuit.

always:

- Declares a cyclic behavior.
- Executes and re-executes indefinitely, until the simulation is stopped.

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.

Simulating clock in sequential circuit

In simulating a sequential circuit, it is necessary to generate a clock source for triggering the flip-flops. Two ways of doing so:

```
initial
  begin
    clock = 1'b0;
    repeat (30)
        #10 clock = ~clock;
  end
```

Sets clock to 0 at time 0

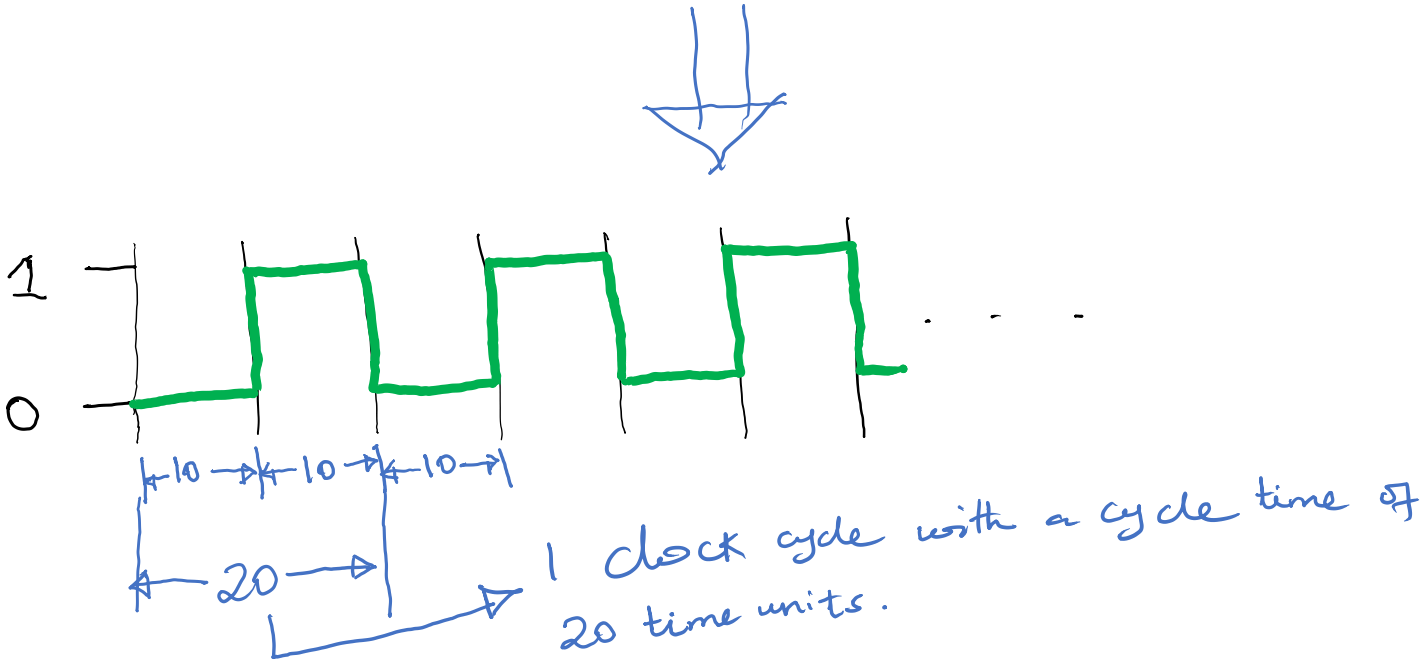
Loop that executes 30 times

Wait 10 minutes then complement the value of clock.

```
initial
  begin
    clock = 1'b0;
  end

initial 300 $finish;
always #10 clock = ~clock;
```

Terminate the simulation after 300 time units

After 10 time units, the always statement repeatedly complements clock.



Event control operator, @

- Used to suspend activity until an event occurs.
- The event can be:
  - ✓ An unconditional change in a signal value - e.g., **@** A
  - ✓ A specified transition of a signal value - e.g., **@** (**posedge** clock)

- General form of this type of statement is:

  always @ (event control expression) **begin**

  //Procedural assignment statements that execute when the condition is met

  end

- **event control expression (sensitivity list):**
  - Specifies the condition that must occur to launch execution of the procedural assignment statements. Variables on the LHS of the procedural statement must be of the **reg** data type and must be declared as such.
  - Specifies the events that must occur to initiate execution of the procedural statements associated with the **always** block. Statements within the block execute sequentially from top to bottom.

The sensitivity list can specify:

- ✓ **Level-sensitive events –** occur in combinational circuits and latches.

  - E.g.: **always @** (A **or** B **or** C) — Will initiate execution of the procedural statements in the associated always block if a change occurs in A, B, or C.

- ✓ **Edge-sensitive events** – occur in sequential circuits
  - ➢ In synchronous sequential circuits, changes in flip-flops occur only in response to a transition of a clock pulse.
  - ➢ The transition may be either a positive edge (**posedge**) or a negative edge (**negedge**) of the clock.

  - E.g.: **always @** (**posedge** clock, **negedge** reset) — Will initiate execution of the procedural statements only if the *clock* goes through a positive transition or if *reset* goes through a negative transition

Blocking and non-blocking assignments

- Blocking assignments use the symbol (=) as the assignment operator. They are executed sequentially in the order they are listed in a block of statements.
- Non-blocking assignments use (<=) as the operator. They are executed concurrently by evaluating the set of expressions on the RHS of the list of statements; they don't make assignments to their LHS till all of the expressions are evaluated.

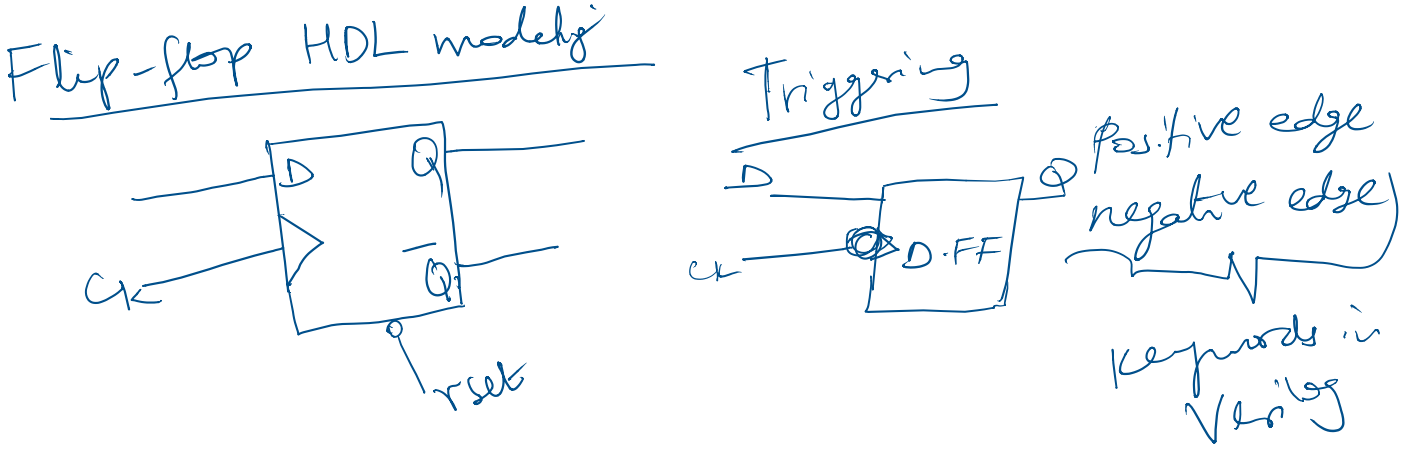
Example: If A=1 and B=3; the two types of assignments give the following results:

Blocking assignment:	**Non-blocking assignment**:
B = A	B <= A
C = B + 1	C <= B + 1
After execution, B = 1; C = 2	**After execution, B = 1; C = 4**

fork … join block

- Similar to begin … end block; but statements within it execute in parallel as opposed in sequence.
- More convenient to use fork …join block instead of begin … end in describing waveforms

Example 1:



Top Module

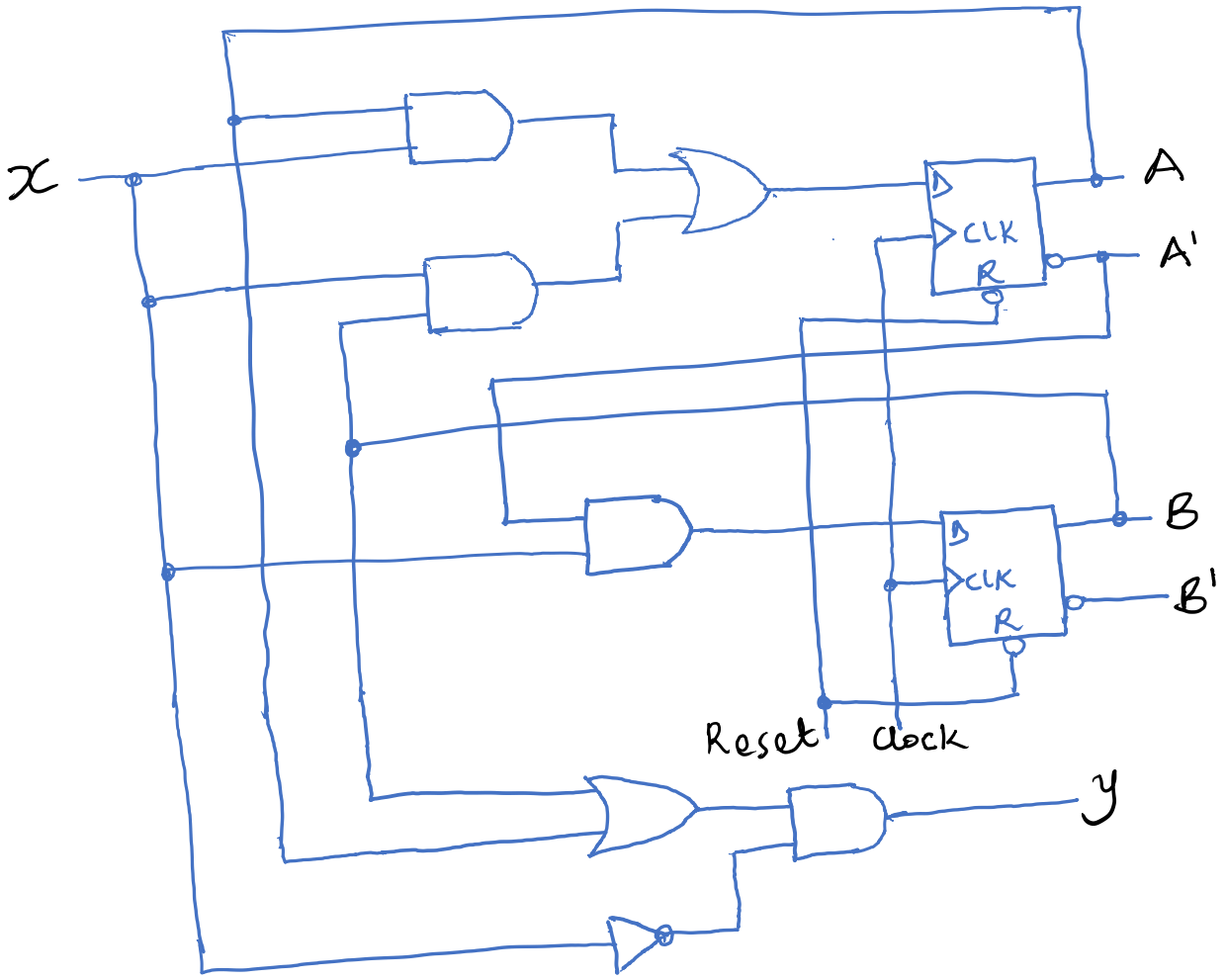
```
module example_three (D, clk, rst, Q);

input D, clk, rst;
output Q;

// Declare the output as a reg
reg Q;

//Implement the circuit
always @ (posedge clk, negedge rst)
//Assign the inputs to the outputs
//Our D-flipflop would operate normally as long as it is not in
reset
//If its in reset, the output would always be at 0
if(rst == 0) Q <= 1'b0; //Non-blocking assignment
else Q <= D;  //Normal operation of the circuit

endmodule
```

Test Bench

```
`timescale 1ns/1ns
`include "example_three.v"

module example_three_tb;
reg D, clk, rst; //inputs to simulate
wire Q;   //outputs to simulate

//Instantiate the module that we are simulating
example_three D_FlipFlop(D, clk, rst, Q); //Retain the same
order

//Simulate the clock
initial
    begin
    clk = 1'b0; //initialize the clock value
    repeat(25)
    #10 clk = ~clk;
    end

//simulate the circuit using the initial block
initial
    begin
    $dumpfile("example_three_tb.vcd");
    $dumpvars(0,example_three_tb);
    D = 1'b1; rst = 1'b0; //our flip flop is the reset state
    #35 D = 1'b1; rst = 1'b1; //Flip in normal operation
    #35 D = 1'b0; rst = 1'b1; //Flip in normal operation
    #35 D = 1'b1; rst = 1'b0; //flip flop is in reset state
    #45
    $display ("Simulation ended");
    end

endmodule
```

Example 2:

Consider a clocked sequential circuit as shown below:



State Equations

- State equation is an algebraic expression that specifies the condition for a flip-flop state transition.

  $A_{n+1} = A_n x_n + B_n x_n$

  $B_{n+1} = A'_n x_n$

  $y_n = (A_n + B_n) x'_n$

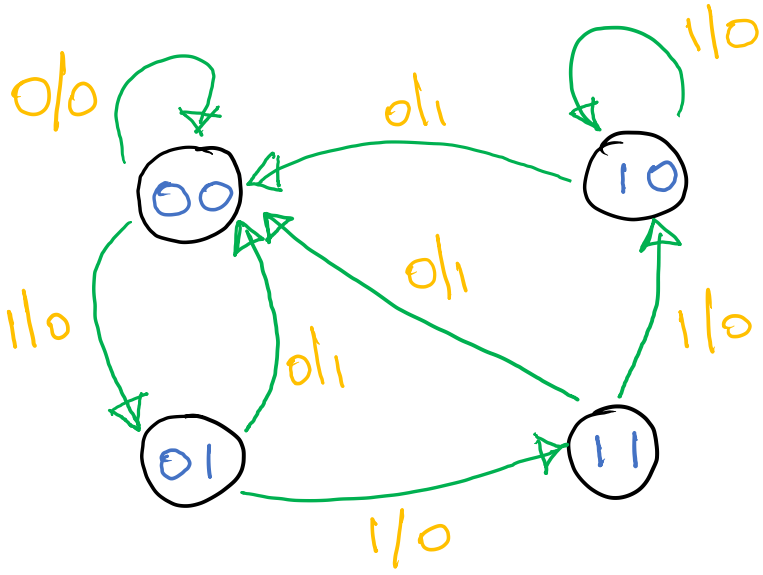
State/Transition Table

PS A_n	PS B_n	Input x_n	Output y_n	NS A_{n+1}	NS B_{n+1}
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	0	1	0

second form of state table

PS		NS				Output	
		$x_n=0$		$x_n=1$		$x_n=0$	$x_n=1$
A_n	B_n	A_{n+1}	B_{n+1}	A_{n+1}	B_{n+1}	y_n	y_n
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

State Diagram



Mealy and Moore state machines

There are two distinct types of sequential circuits:

- Mealy sequential circuit
  - External outputs are a function of the circuit present states and external inputs.
  - For example, the above circuit is a Mealy circuit i.e. $\mathbf{y} = \boldsymbol{f}\ (\boldsymbol{x}, \boldsymbol{Q_A}, \boldsymbol{Q_B})$
  - In the design & operation of Mealy machines, extra precaution must be taken to ensure that the external inputs do not change states at the same instant with the clock signal.

- Moore sequential circuit
  - External outputs are a function of the circuit present states only.
  - The outputs of a Moore machine can be a function of the flip-flop outputs but not directly connected to external inputs, i.e. $\mathbf{y} = \boldsymbol{f}\ (\boldsymbol{Q_A}, \boldsymbol{Q_B})$

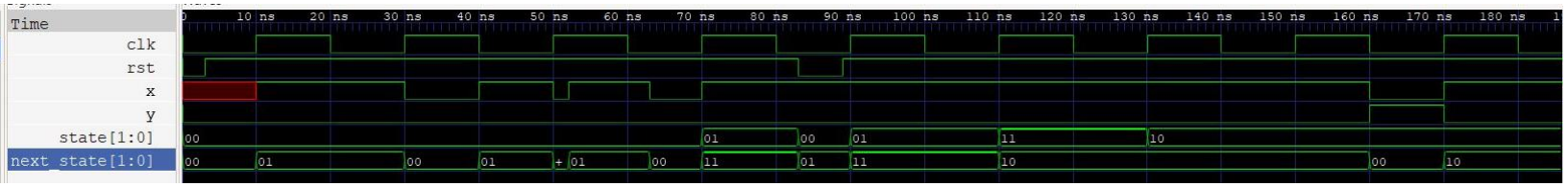
Top Module

```
module example_Mealy (x, clk, rst, y);
input x, clk, rst;
output y;

// Declare the output as a reg
reg y;

//Declare variables to hold the values of the present and next
states
reg [1:0] state, next_state;
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;

//Implement the circuit

//Check if in reset
always @ (posedge clk, negedge rst)
//If its in reset, circuit will always be in reset state i.e. 00
if(rst == 0) state <= S0; //reset state
else state <= next_state;  //Normal operation of the circuit

//Form the next state
always @ (state, x)
    case (state)
        S0: if(x) next_state = S1; else next_state = S0;
        S1: if(x) next_state = S3; else next_state = S0;
        S2: if(x) next_state = S2; else next_state = S0;
        S3: if(x) next_state = S2; else next_state = S0;
    endcase

//Form the output
always @ (state, x)
    case (state)
        S0: y = 0;
        S1, S2, S3: y = ~x;
    endcase

endmodule
```

Test Bench

```
`timescale 1ns/1ns
`include "example_four_Mealy.v"

module example_Mealy_tb;
reg x, clk, rst; //inputs to simulate
wire y;   //outputs to simulate

//Instantiate the module that we are simulating
example_Mealy Mealy_one(x, clk, rst, y); //Retain the same order

//Simulate the clock
initial
    begin
    clk = 1'b0; //initialize the clock value
    repeat(30)
    #10 clk = ~clk;
    end

//Value Change Dump (vcd)
initial begin
    $dumpfile("example_Mealy_tb.vcd");
    $dumpvars(0,example_Mealy_tb);
    #200 $dumpoff; // stop dumping
end

//Simulate the circuit
initial fork
    rst = 0;
    #3 rst = 1;
    #83 rst = 0;
    #89 rst = 1;
    #10 x = 1;
    #30 x = 0;
    #40 x = 1;
    #50 x = 0;
    #52 x = 1;
    #63 x = 0;
    #70 x = 1;
    #80 x = 1;
    #160 x = 0;
    #170 x = 1;
join

endmodule
```

Simulation


Time
clk
rst
x
y
state[1:0]
next_state[1:0]
10 ns
20 ns
30 ns
40 ns
50 ns
60 ns
70 ns
80 ns
90 ns
100 ns
110 ns
120 ns
130 ns
140 ns
150 ns
160 ns
170 ns
180 ns