HACETTEPE UNIVERSITY
Computer Engineering Department
BBM233 Logic Design Laboratory
Fall 2019

Experiment 5
Sequential Circuits in Verilog

Deadline for report submission:
Wednesday, 11.12.2019 at 13:00 for all sections.

AIM

In this experiment you will design a sequential circuit and implement it in Verilog HDL.

BACKGROUND

Sequential circuits are circuits whose outputs depend on both the present inputs and the sequence of past inputs (sequential circuits include memory elements). That is, the previous inputs or state of the circuit will have an effect on its present state.

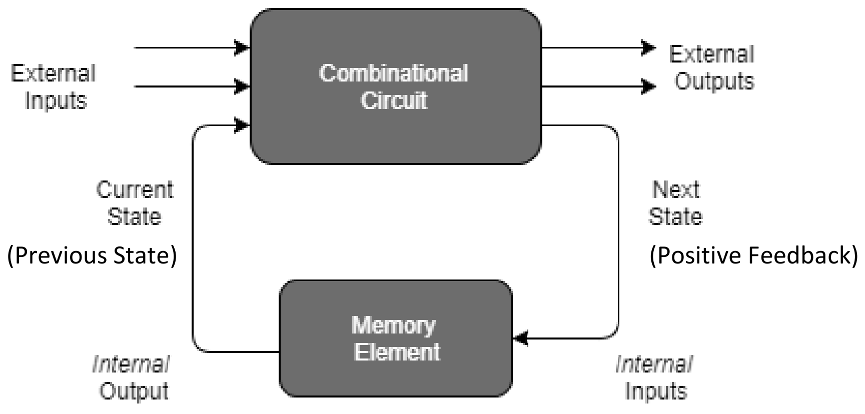
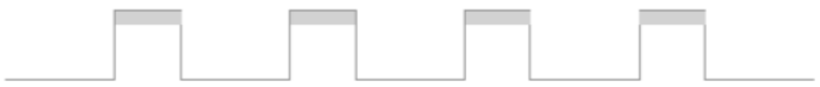


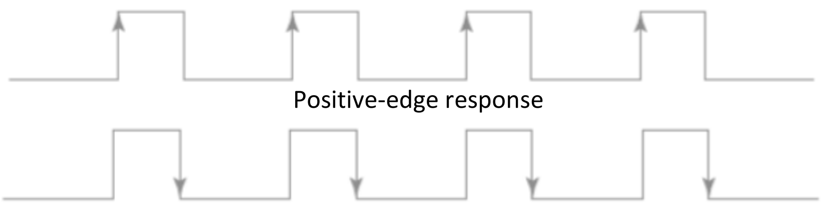
Figure: Sequential Circuit

Storage Elements

Latches: level-sensitive



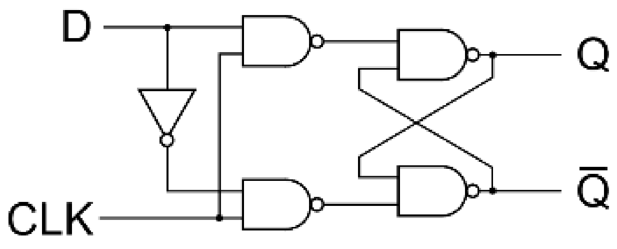
Flip-flops: edge-sensitive



Negative-edge response

A flip flop is a basic building block of sequential logic circuits. It is a circuit that has two stable states and can store one bit of state information. The output changes state by signals applied to one or more control inputs.

Edge-Triggered D Flip Flop



Inputs	Present Output	Next Output
D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Characteristic Equation

$$Q(t+1) = D$$

A basic D Flip Flop has a D (data) input, a clock (CLK) input and outputs Q and Q’ (the inverse of Q). Optionally it may also include the PR (Preset) and CLR (Clear) control inputs. A few possible Verilog implementations are given below.

Verilog code for Rising Edge D Flip Flop:

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

Verilog code for Falling Edge D Flip Flop:

```
module FallingEdge_DFlipFlop(D,clk,Q);
input D; // Data input
input clk; // clock input
output reg Q; // output Q
always @(negedge clk)
begin
 Q <= D;
end
endmodule
```

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

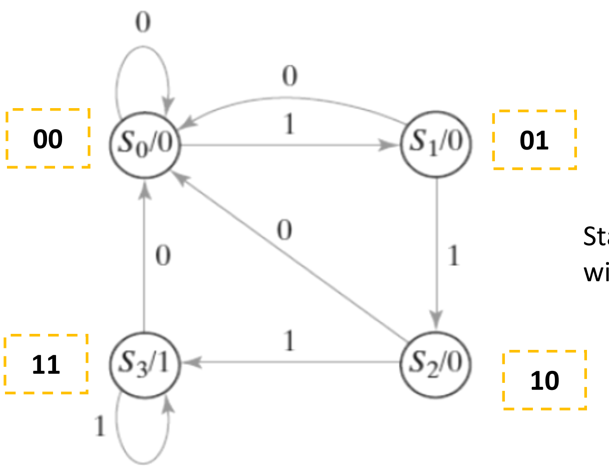
```
module RisingEdge_DFlipFlop_SyncReset(D,clk,sync_reset,Q);
input D; // Data input
input clk; // clock input
input sync_reset; // synchronous reset
output reg Q; // output Q
always @(posedge clk)
begin
 if(sync_reset==1'b1)
  Q <= 1'b0;
 else
  Q <= D;
end
endmodule
```

Sequential Circuit Design (With an Example)

Here we state the steps of designing a sequential circuit with an example circuit specification:
111 Sequence Detector: Design a circuit that outputs 1 when a sequence three consecutive 1's is applied to input, and 0 otherwise.

To design a sequential circuit, start with the problem definition and use the following steps:

- **Step-1**: **Create a state transition diagram from the description. Reduce the number of states if necessary, and assign binary values to the states**.



State diagram for 111 Sequence Detector with assigned binary values to the states.

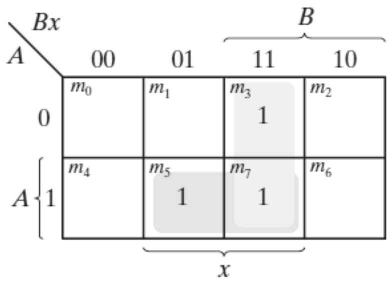
- **Step-2**: **Convert the state transition diagram into a state transition table (binary coded state table).**

State Table for Sequence Detector

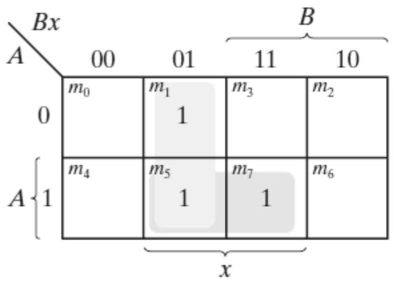
Present State		Input	Next State		Output
A	***B***	***x***	***A***	***B***	***y***
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

- **Step-3: Determine the number of flip-flops needed and choose flip-flop types. Derive their excitation tables (if designing a sequential circuit with flip-flops other than the D type), and derive input and output equations from the state table. Minimize the functions for the flip-flop inputs (e.g. using Karnaugh Maps).**

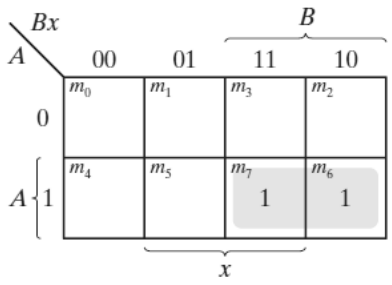
We choose D-type flip-flops. Since there are four states, we need two D flip-flops, and we label their outputs as A and B. The characteristic equation of the D flip-flop is `Q(t + 1) = D`, which means that the next-state values in the state table specify the D input condition for the flip-flop. From the state table we obtain the following input equations:



$D_A = Ax + Bx$

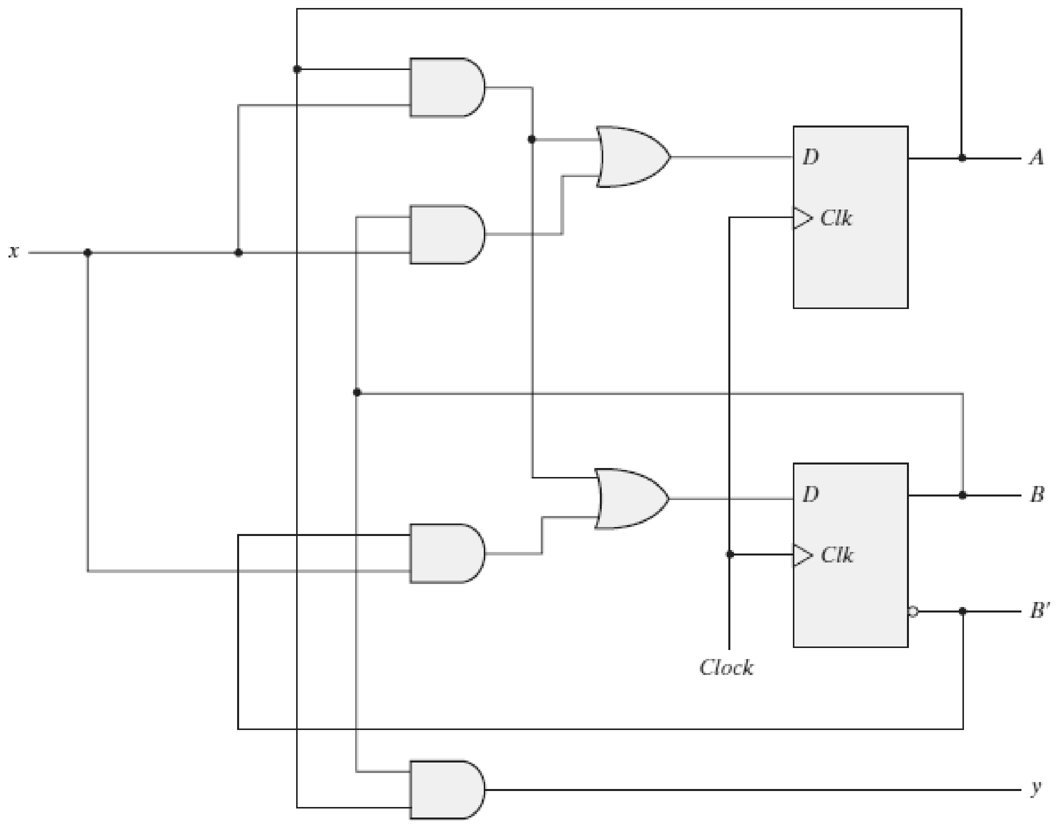


$D_B = Ax + B'x$



$y = AB$

- **Step-4: Use simplified functions for D1 and D2 to design sequential circuit.**



- **Step-5: Finally determine the combinatorial circuit to represent the output (if any).**
  Not necessary for this example.

Verilog Code for the 111 Sequence Detector

Structural design using flip-flops:

```
module D_ff(D, clk, Q); // Rising-edge D flip flop
    input D; // Data input
    input clk; // clock input
    output reg Q; // output Q

    always @(posedge clk)
    begin
        Q <= D;
    end
endmodule
```

Remember the equations:

$D_A = Ax + Bx$ $D_B = Ax + B'x$ $y = AB$

```
`include "D_ff.v"
module three_1s_seq_detector(
    input x_in,
    input clock,
    input reset,
    output y);

    reg [1:0] present_state = 2'b00;
    wire [1:0] next_state;

      D_ff D1 (
            .D((present_state[1]&x_in)|(present_state[0]&x_in)),
            .clk(clock),
            .Q(next_state[1])
      );

      D_ff D0 (
            .D((present_state[1]&x_in)|(~present_state[0]&x_in)),
            .clk(clock),
            .Q(next_state[0])
      );

      always @(reset or next_state) begin
        if (reset) begin present_state <= 2'b00; end
        else begin present_state <= next_state; end
      end
      assign y = present_state[1] & present_state[0];
endmodule
```

An alternative behavioral design in Verilog:

```
module three_1s_seq_detector_behavioral(
    input x_in,
    input clock,
    input reset,
    output y);
    parameter s0 = 2'b00, s1 = 2'b01, s2 = 2'b10, s3 = 2'b11;
    reg [1:0] state, next_state;

    always@(posedge clock, negedge reset)
        if (reset == 0) state <= s0;
        else state <= next_state;

    always@(state, x_in) begin

        case(state)
            s0: if (x_in == 0) next_state = s0;
            else if (x_in == 1) next_state = s1;
            s1: if (x_in == 0) next_state = s0;
            else if (x_in == 1) next_state = s2;
            s2: if (x_in == 0) next_state = s0;
            else if (x_in == 1) next_state = s3;
            s3: if (x_in == 0) next_state = s0;
            else if (x_in == 1) next_state = s3;
            default: next_state = s0;
        endcase
    end
    assign y = state[1] & state[0];
endmodule
```

A sample testbench:

```
module three_1s_seq_detector_tb;
    // Inputs
      reg x;
      reg clk;
      reg rst;
      // Outputs
      wire y;
      // Instantiate the Unit Under Test (UUT)
      three_1s_seq_detector uut (.x_in(x), .clock(clk), .reset(rst), .y(y));
      reg [19:0] input_data;
      integer shift_amount;
      initial begin
            // Initialize Inputs
            input_data = 20'b00001110100111100000;
            shift_amount=0;
            rst = 1; #50;
            rst = 0; #500; $finish;
      end
      initial begin // Generate clock
            clk = 0;
            forever begin
                  #10;
                  clk = ~clk; // change every 10ns
            end
      end
      always @ (posedge clk) begin
            x = input_data>>shift_amount; // get the next input bit
            shift_amount=shift_amount+1;
      end
endmodule
```

LAB EXPERIMENT

By following the steps discussed above, you will design a **Synchronous 3-Bit Binary Up/Down Counter** which counts either up or down depending on the counting mode.

- Counter outputs will be from 000 to 111 (from 0 to 7).
- The initial state of the counter is 000. There is a single input M (mode selecting signal) such that:
  - When M = 0, the counter counts down (once 0 is reached, it should start counting down from 7 again - cyclically),
  - When M = 1, the counter counts up (once 7 is reached, it should start counting up from 0 again - cyclically).

Experiment Steps:

1. Follow the given steps for designing sequential circuits starting with drawing the state transition diagram. For each step show your work clearly, as it is shown in these instructions with the example of sequence detector. Use D flip-flops in your implementation.

2. Once you have designed the circuit, write a Verilog code implementing the counter. Use behavioral design approach as shown in the alternative implementation. Write an appropriate testbench which clearly shows that your counter works properly.

3. Write a report that includes all design steps, final circuit design, Verilog codes, and proof of correct results (e.g. screenshots of waveform, variable changes from the console, etc.).

Report Submission:

The deadline for submission is Wednesday, 11.12.2019 at 13:00 for all sections. Zip your files (not .rar, only .zip files are supported by the system) and submit your work through https://submit.cs.hacettepe.edu.tr/index.php with the following file hierarchy:

```
- <studentID>.zip
    - counter.v
    - counter_testbench.v
    - report.pdf
```

Important notes:

- **One submission per group is enough. Clearly state the names and student IDs of the group members on the first page of your reports.**
- **Don’t share your codes with other groups as we will perform the plagiarism check.**