University Of Engineering and Technology, Lahore
Computer Engineering Department

Course Name: Computer Architecture	**Course Code:** CMPE-421L
Assignment Type: Lab	**Dated:** 11th September 2023
Semester: 7th	**Session:** 2020
Lab/Project/Assignment #: 2	**CLOs to be covered: CLO 1**
Lab Title: Sequential Design in System Verilog	**Teacher Name:** Engr. Afeef Obaid

Lab Evaluation

CLO1	Understand the design and simulation of basic digital circuits with HDL and HDL simulation tools.					
Levels (Marks)	**Level1**	**Level2**	**Level3**	**Level4**	**Level5**	**Level6**
(10)						
					Total	**/10**

Rubrics for Current Lab Evaluation

Scale	**Marks**	**Level**	**Rubric**
Excellent	**9-10**	L1	Submitted all lab tasks, BONUS task, have good understanding.
Very Good	**7-8**	L2	Submitted the lab tasks but have good understanding
Good	**5-6**	L3	Submitted the lab tasks but have weak understanding.
Basic	**3-4**	L4	Submitted the lab tasks but have no understanding.
Barely Acceptable	**1-2**	L5	Submitted only one lab task.
Not Acceptable	**0**	L6	Did not attempt

Lab #2

Lab Goals

By reading this manual, students will be able to:

- Understand the basics of sequential design.
- Create SV modules to implement sequential components like flip-flops, registers, latches, and FSMs.
- Simulate and verify the behaviors of sequential designs.

Equipment Required

- Computer system with ModelSim or Xcelium, installed on it.

Sequential Design in System Verilog

Introduction

In contrast to a combinational circuit, the output of a sequential circuit can depend on the current as well as previously applied inputs. Consequently, to design sequential circuits, we need memory elements that retain the information about previous inputs. Sequential systems can be implemented using synchronous and asynchronous circuits. However, synchronous circuits are preferred due to the ease of their implementation as meeting timing requirements become easier compared to meeting them in asynchronous circuits. In this section, we will study synchronous sequential circuit design and their verification.

Basic Memory Elements and their SV Description

All sequential circuits require some sort of memory element to store information about previous inputs. There are two basic memory elements,

- D flip-flop
- Latch

Each of these elements can store 1-bit of data. We can combine multiple flip-flops to store a word (data of width more than 1) and the resulting storage element is called a **register**.

In this session, we will learn about the behavior of these basic memory elements and their SV descriptions.

Latch SV Description

```
module latch
(
    input wire D,  // Data input
    input wire EN, // Enable input
    output reg Q   // Latch output
);

    always_latch
    begin
        if (EN)
        begin
            Q = D;
        end
    end

endmodule
```

D flip-flop SV Description

```
module d_flip_flop
(
    input  logic clk, // Clock input
    input  logic rst, // Reset input
    input  logic D,   // Data input
    output logic Q    // Flip-flop output
);

    always_ff @(posedge clk or posedge rst)
    begin
        if (rst)
        begin
            Q <= 0; // Reset the flip-flop to 0 when the reset signal is asserted.
        end
        else
        begin
            Q <= D; // Update the flip-flop with the D input on the rising edge of
the clock.
        end
    end

endmodule
```

Basic Testbench Design for Synchronous Circuits

The testbench design for a synchronous circuit is different from a combinational circuit as all the outputs are changed with respect to the synchronizing clock's edge. As a result, the test signals that we need to apply to synchronous circuits should also be synchronized to the clock. In this section, we will learn how to design stimulus for synchronous circuits.

SV Testbench for a D flip-flop

```
module tb_d_flip_flop;

  // Define signals for connecting to the D flip-flop module
  logic clk; // Clock signal
  logic rst; // Reset signal
  logic D;   // Data input
  logic Q;   // Flip-flop output

  // Instantiate the D flip-flop module
  d_flip_flop uut
  (
    .D(D),
    .clk(clk),
    .rst(rst),
    .Q(Q)
  );

  // Clock generation
  always
  begin
    #5 clk = ~clk; // Invert the clock every 5 time units
  end

  // Initialize signals
  initial begin
    clk = 0; // Initialize the clock signal
    rst = 0; // Initialize the reset signal
    D = 0;   // Initialize the data input

    // Apply reset and data input
    rst = 1; // Assert reset
    D = 1;   // Set data input to 1
    #10;     // Wait for 10 time units
    rst = 0; // Deassert reset
    #10;     // Wait for 10 time units

    // Monitor the flip-flop's output
    $display("Time\tD\tQ");
    $display("------------------");

    // Simulate for 20 clock cycles
    repeat (20)
    begin
      D = ~D; // Toggle the data input
      #10;    // Wait for 10 time units
      $display("%d\t%d\t%d", $time, D, Q);
    end

    $finish; // End the simulation
  end

endmodule
```

Designing a 2-Bit Binary Counter

Description

In this exercise, you will design a 2-bit binary counter using SV. A binary counter is a sequential circuit that counts in binary, starting from 00 and incrementing by one with each clock cycle. This counter will help you understand the concept of sequential logic and how to implement it in SV.

Instructions

- Open your text editor or IDE.
- Create a design document for your module to be made.
- Create a new SV module named "binary_counter" compliant with your design document.
- Inside the module, use basic memory elements to implement the module.
- Create a testbench for your binary counter module.
- Apply clock pulses and verify the counter's behavior by comparing it with design document.
- Simulate the design and observe the output.

Review Questions

Review Question 1

Explain the difference between combinational and sequential logic.

Lab Tasks

Lab Task

Modify the binary counter design to create a 3-bit binary counter and update the testbench accordingly. Simulate the 3-bit counter and observe its behavior.