Nanyang Technological University
School of Electrical and Electronic Engineering

NTU-TUM MSc (IC Design) Programme
Digital IC Design Course: NM6008

Laboratory Manual

Module 4

Verilog State Machine

Course Instructors: A/Prof Lim Meng Hiot
A/Prof Gwee Bah Hwee
Dr Cheng Deruo
Cheng Juncheng
Sheng Shirui

2nd – 13th October 2023

Section 1: Introduction

This tutorial style laboratory manual document is meant for NTU-TUM Master of Science (MSc) course – Integrated Circuit (IC) Design, NM6008. It is a step-by-step hands-on exercise to familiarize students with HDL state machine implementation.

The following sub-sections state the objectives and the experimental design information of this module – *Verilog State Machine*.

[Important Note: It is a pre-requisite that you have gone through the tutorial in Module 3. You are expected to refer to the Module 3 laboratory manual document for relevant information, if necessary.]

1.1 Objectives:

Module 4 mainly pertains to design and implementation of a state machine. The specific objectives include:

(i) To reinforce students' understanding and familiarization with IC design methodologies by repeating all the design flows/steps in Module 3;

(ii) To understand the sequential logic and the synchronous-logic design methodologies;

(iii) To construct state machine.

1.2 Design Information

1.2.1 General Background

Combinational logic, e.g. the full-adder, which is mentioned in the Module 3, is a type of logic circuit whose output is solely a function of its inputs. Conversely, sequential logic is a type of logic circuit whose output depends not only on its present inputs but also on the past history of its input(s)/output(s). Put simply, sequential logic has a 'memory' function embedded in it. In reality, almost all circuits and systems in practical digital devices are a mixture of combinational logic and sequential logic.

The most basic sequential logic components are latches and flip-flops. Fig. 1 (a) shows a latch example with its corresponding waveforms. Particularly, the latch is a 'logic l' sensitive latch such that when the *CLK* signal is '1' (see the shaded parts), the input *D* is transferred over the output *Q*. When the *CLK* signal is '0', the output *Q* is latched, and hence remains unchanged. Fig. 1 (b) shows a flip-flop example with its corresponding waveforms. Particularly, the flip-flop is a 'positive-edge' sensitive flip-flop where when the *CLK* signal is triggered from logic '0' to '1' (see the short shaded transition parts), the input *D* is transferred over the output *Q*. Otherwise, the output *Q* is latched and hence remains unchanged. In practice, flip-flops are preferable to latches due to their edge-sensitive attribute so that the former will have lesser timing issues. Hence, it is not surprising that some reference books even recommend **NOT** adopting latches in your design(s) if possible.

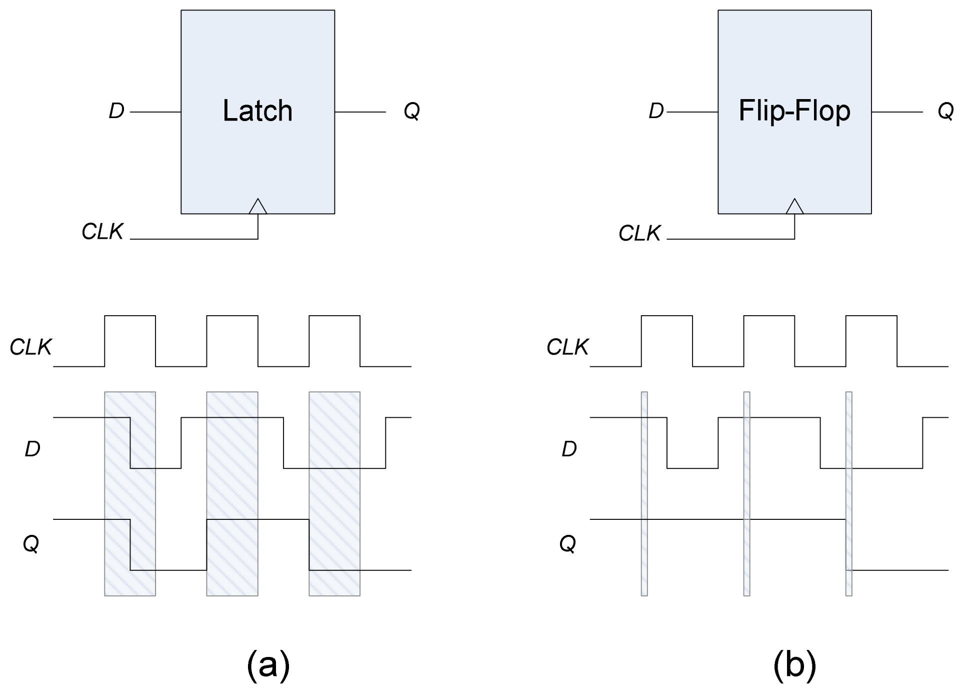

Fig. 1 Example: (a) a Latch with its waveforms, (b) a flip-flop with its waveforms.

The *CLK* signal in the Fig. 1 is essentially a (global) clock signal to synchronize digital circuits, i.e. the transfer of data is done within a well-synchronized timing. Such design methodology is called the synchronous-logic methodology, and is the most prevalent methodology adopted by the circuits and systems community. The main advantage of adopting the synchronous-logic methodology is to mitigate timing issues (and hence the associated design complexity). A good reference book to understand the synchronous-logic methodology is "Digital Integrated Circuits, A Design Perspective" authored by Jan. M. Rabaey *et al.* Design manuals of the Electronic Design Automation (EDA) vendors (e.g. Synopsys, Cadence, Mentor Graphics, Xilinx, Altera, etc.) are also good reading materials if students wish to learn more in the methodology.

In this module, students will learn to design a simple sequential logic design – a state machine.

1.2.2 Design Example – A State Machine

Fig. 2 shows the primary inputs and outputs of the State Machine; for consistency, please use "StateMachine" as the top module name. Consider first the 7 inputs, namely *CLK*, *NRST*, *start*, *rst*, *CIN*, *A*, and *B*, and their associated function(s). The *CLK* signal triggers the state machine to operate state by state, and the *NRST* signal is an asynchronous signal to reset all the flip-flops embodied in the State Machine. The *CIN*, *A*, and *B* signals are the input signals of a full-adder embedded in it. The *start* and *rst* signals are two further control signals for the State Machine. Fig. 3 further shows the state transition diagram. There are four states: S0, S1 S2 and S3. S0 is the default state. When the *start* signal is enabled in S0, the State Machine will jump to S1, or otherwise remains in the same state. In S1, when the *rst* signal is not enabled, the State Machine will jump to S2, or otherwise will jump to S0. In S2, when the *rst* signal is not enabled, the State Machine will jump to S3, or otherwise will jump to S0. In S3, when the *rst* signal is not enabled, the State Machine will jump to S1, or otherwise will jump to S0.

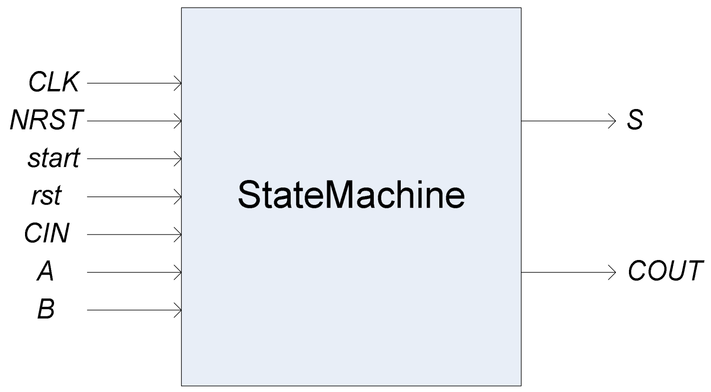

Fig. 2 The primary inputs and outputs of the State Machine.

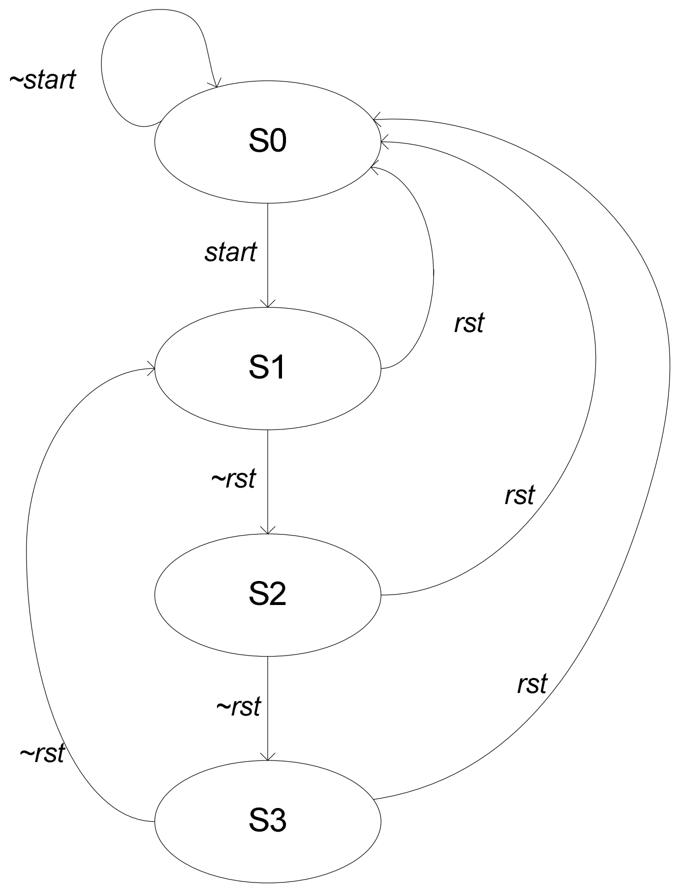

Fig. 3 The state transition diagram

Consider now the 2 outputs, namely *S* and *COUT*. In this exercise, the outputs are required to be realized as combinational logic. In both S1 and S3 stage, the *S* output will function as a Sum signal according to the three input signals, *A*, *B*, and *CIN*. In both S2 and S3 stage, the *COUT* signal will function as a Carry signal according to the three input signals, *A*, *B*, and *CIN*. Otherwise, both the *S* and *COUT* signals are '0'. Refer to the Module 3 on how the *S* and *COUT* signals can be generated.

Section 2: Procedures/Tasks

2.1 Behaviour Modelling and Simulations

A. <u>Working Environment:</u>

1. Create your working environment; please refer to the Module 3.

B. <u>Creating a Behavioural Verilog Code for Implementation of State Machine</u>

1. Create a behaviour code to design the State Machine mentioned in Section 1.2.2. You can follow the tips presented in steps 2 to 5.

2. Tip 1: Define your state parameters

```
module StateMachine (//define your inputs and outputs);

parameter S0 = 2'b00;
parameter S1 = 2'b01;
parameter S2 = 2'b10;
parameter S3 = 2'b11;

//define your inputs and outputs

reg [1:0] state, next_state:
```

3. Tip 2: Synchronous-logic CLK and asynchronous-logic NRST

```
// Sequential Logic for the current state logic
always @ (posedge CLK or negedge NRST)
  if (!NRST)
     begin
       state <= #1 //What is your default state?
     end
     else
     begin
       state <= #1 next_state;
     end
```

4. Tip 3: Combinational Outputs

```
assign S    = //Construct this output by yourself;
assign COUT = //Construct this output by yourself;
```

5. Tip 4: State Machine Coding

```
always @ (state or start or rst)
  case (state)
    S0 : begin
           if (start)
              next_state = S1;
           else
              next_state = S0;
         end
    S1 : begin
           if (rst)
              next_state = S0;
           else
              next_state = S2;
//continue your coding …
```

C. Creating a Behavioural Verilog Code for Creating a Test-bench

1. Create a test-bench to test the State Machine. You can follow the tips below in order to generate a periodical clock signal.

```
parameter CLK_T = 50;

…

 //Clock Generator
initial
  begin
    CLK = 0;
    forever #CLK_T CLK = !CLK;
  end
```

D. Running Verilog Simulations

1. Run your simulations -refer to Module 3.

2.2 Synthesis

A. Working Environment:

1. Create your working environment - refer to Module 3.

B. Modify the Basic Setups/Files

1. Modify the names of the basic setups/files - refer to Module 3.
2. Make sure the clock frequency is 100ns.

C. <u>Using the Synopsys Tool – Design Vision</u>

1. Perform synthesis - refer to Module 3**.**
2. Answer the questions in Section 3.1.

2.3 Post-Synthesis Simulations

A. <u>Working Environment:</u>

1. Create your working environment refer to Module 3.

B. <u>Modifying a Behavioural Verilog Code for Creating a Test-bench to Test the State Machine</u>

1. Modify your test-bench - refer to Module 3.

C. <u>Running Verilog Simulations</u>

1. Run your simulations - refer to Module 3.

2.4 Power Simulations

A. <u>Working Environment:</u>

1. Create your working environment - refer to Module 3.

B. <u>Simulating the Power Dissipation:</u>

1. Run your simulations - refer to Module 3.

Section 3: Assignment Questions

1. Basic Questions

1. Refer to the file "report.rpt" generated in Section 2.2 and answer the following questions:

   What are the gates used for realizing the State Machine?

   How many registers were used?

   Check the timing information, and explain what "Slack" means.

2. Optional Questions

Since the *S* and *COUT* signals are outputs of a combinational logic circuit, you may observe glitches in these signals. Suppose you are asked to use flip-flops to eliminate these glitches. Design your new circuits, and verify them. What are the additional hardware and power costs compared to the original design?