Digital System Design Laboratory Exercise 9

Finite State Machines

This is an exercise in using finite state machines.

# Part I

We wish to implement a finite state machine (FSM) that recognizes two specific sequences of applied input symbols, namely four consecutive 1s or four consecutive 0s. There is an input *w* and an output *z*. Whenever *w* = 1 or *w* = 0 for four consecutive clock pulses the value of *z* must be 1; otherwise, *z* = 0. Overlapping sequences are allowed, so that if *w* = 1 for five consecutive clock pulses the output *z* will be equal to 1 after the fourth and fifth pulses. Figure [1](#_bookmark0) illustrates the required relationship between *w* and *z*.

Clock

*w z*

Figure 1: Required timing for the output *z*.

* Draw the state diagram for the FSM that recognizes these sequences.
* Construct the binary-coded state table for the FSM.

# Part II

Design and implement your FSM circuit of part I using Verilog HDL as follows:

* Your Verilog code should use the coding style of FSM as discussed in the class. i.e.,
  + Use a case statement inside an always block to define the state computation.
  + Use another always block to implement the state flip-flops (state assignment).
  + Specify the output z either using a separate always block or using the always block used for state computation.
  + Finally, write a testbench module to apply different input sequences and check if z goes high when four consecutive 0s or 1s are detected. Simulate your testbench and verify the functionality of behavioral model of your code
  + Synthesis your code and read the synthesis report to identify the type of FSM and encoding used by Vivado.
  + As a default, the Vivado synthesis tool chooses an encoding protocol for state machines based on internal algorithms that determine the best solution for most designs. However, the FSM\_ENCODING property lets you specify the state machine encoding of your choice. The Verilog syntax to set this property is as shown below:

(\* fsm\_encoding = "one\_hot" \*) reg [7:0] my\_state;

You can apply one of the following encodings:

* + - AUTO: This is the default behavior when FSM\_ENCODING is not specified. It allows the Vivado synthesis tool to determine the best state machine encoding method. In this case, the tool might use different encoding styles for different state machine registers in the same design.
    - ONE\_HOT
    - SEQUENTIAL
    - JOHNSON
    - GRAY
    - NONE: This disables state machine encoding within the Vivado synthesis tool for the specified state machine registers. In this case the state machine is synthesized as logic.
  + Apply an encoding type other than that used by VIVADO automatically. Synthesis your code again and analyze how the implementation changes.
  + Finally, apply NONE encoding, synthesize your code and analyze the results again.

# Part III

The sequence detector can also be implemented in a straightforward manner using shift registers, instead of using the more formal approach described above.

* Create Verilog code that instantiates two 4-bit shift registers; one is for recognizing a sequence of four 0s, and the other for four 1s.
* Include the appropriate logic expressions in your design to produce the output z.
* Check the functionality of your code through a testbench.
* Synthesize your code and analyze how does this implementation differ from the implementation of part II with FSM encoding style set to NONE.

# Part IV

In this part of the exercise, you are required to implement a signal generator (LED Blinking) using an FSM. Write separate Verilog codes to implement the following three modes. Each part incrementally builds up on the previous part. *For each part, first* ***design appropriate FSM****, then* ***write its corresponding Verilog code****, and finally* ***implement your design on a NEXYS A7 FPGA development board*** *to test the functionality of each part.* Thestate diagram that clearly illustrates all the states and transitions involved in both blinking modes

* Implement an FSM that controls the blinking of an LED on the FPGA board. The circuit should have an enable input and an output to drive the LEDs. The LED remains off until the enable input is asserted. Once the enable input is asserted (turned on), the LED blinks such that it remains ON for 1 second and OFF for 1 second. This blinking operation continues until the enable signal is turned off.
* Modify your design such that the blinking operation is modified such that the LED turns on for 1 second, then OFF for 1 second, then ON for 2 seconds, and OFF for 1 second. This blinking operation continues until the enable signal is turned off.
* Finally, modify your FSM and corresponding code to activate the signal generator activated in two different modes. The first mode is the Short Blink Mode, where the LED blinks three times with each blink consisting of one second ON and one second OFF. The second mode is the Long Blink Mode, where the LED blinks three times with each blink consisting of three seconds ON and one second OFF. This blinking operation continues until the enable signal is turned off.
* Additionally, you are required to create a complete state diagram that clearly illustrates all the states and transitions involved in both blinking modes. Finally, you will implement and test your FSM design on a NEXYs A7 FPGA development board to verify its functionality and timing accuracy.