ECE 5727

Lab 2: FSM Design Options

Due Wednesday, March 17
Total Points: 32

Objectives

In this lab you will:

1. Learn to use the Vivado simulator
2. Look at the effects of the various ways to code an FSM

Lab Overview

As discussed in class, there are two types of FSMs: Moore machines have outputs solely dependent on the current state, while Mealy machine outputs are also influenced by the input to the system. Additionally, we covered the two ways to code the FSM. The first way is to use a single clocked procedural block (always_ff). The second is to use a set of two procedural blocks, one that is clocked (alway_ff) to maintain the state and one that handles all the combinational logic for the machine (always_comb).

Instructions

A note for Linux users: in my Vivado installation instructions, I forgot that you will need to install gcc. Make sure to do that (sudo apt install gcc) prior to attempting to launch the simulator.

Download the lab project files either from Canvas (under Files -> Labs -> Lab2) or from the course GitHub (https://github.com/daberman/CT-ECE5727-SP21)

1. I have already created the Vivado project, so all you need to do is select *Open Project* under Quick Start. Navigate to where you have the Lab 2 files are and select Lab2.xpr
2. Under **Design Sources** there is the source code for the FSM_Example module that we will be simulating the behavior of. As shown in class, there is also now an FSM_Example_TB.sv source file located under **Simulation Sources**. This is the file we will be working with for this lab. You are welcome to take a look at the FSM_Example.sv source code if you'd like, but we won't be touching it.
3. If you expand the FSM_Example_TB under **Simulation Source**, you will notice there are now four modules being instantiated instead of just one as I showed during class. This is being done through the use of the *generate* command within the source code and *parameters*, and I will go talk about them when I do a Lab 2 review in class. For now, just know that we will be using the testbench code to simulate four FSM_Example modules, each implementing a different FSM coding style. They are named *Generate_FSM[0:3].DUT*.

4. The first thing we want to do is launch the simulator. You do this by selecting **SIMULATION->Run Simulation** from the Flow Navigator. This will pop up a selection dialog for the type of simulation to run. As neither synthesis nor implementation have been run, all options except for the first (Behavioral Simulation) should be greyed out. That's ok, since all we care about is viewing the behavior of the code; we don't need to simulate the expected real-world functionality. The post-synthesis/implementation simulations have extremely narrow use-cases that are rarely encountered (I've never had to run anything other than behavioral simulations).

5. The simulator will attempt to compile all relevant source files and begin the simulation. If for some reason it fails, check the Console to see what the Error messages are (you will need to scroll up a little). If you can't figure out what's going wrong or how to fix it, definitely reach out to me!

6. Once the simulation launches, it will automatically load the top-level testbench signals and localparams into the waveform viewer and run for a default amount of time (the project should have it configured for 250 ns). If you scroll up a couple lines in the console, you should see timestamps of when each FSM detected the input bit sequence. We will now set up the waveform viewer to take a look at all the signals of interest. At the end of the lab instructions is a screenshot example of what the end result should look like.

   (a) The viewer opens with an extremely zoomed view. We can get a better view by zooming out using either the Zoom Out button or the Zoom Fit button to its left in the viewer toolbar. The latter will fit the entire simulation to your screen, so I typically always start by using that button.

   (b) The first three signals can be left alone. Note that *din* is only a single bit as the same input data is being fed to all four modules, so only the single signal is needed. However, each FSM has its own *dout*, so the top-level *dout* is 4 bits with the index value corresponding to the FSM module. Expand it by clicking the arrow to the left of the signal name so that all four bits are visible at once.

   (c) The 16-bit data signal is next in the viewer. By default, it is shown as a hex value, but since it is shifting binary data, it might be more helpful to see it in that format. Right-click the signal name, then select **Radix->Binary**. If you were to zoom in, you would now be able to see the bits shifting to the left with each clock cycle. In my example waveform, I moved data to sit above din since I would prefer to view them together that way. You can drag the signal up if you want to do the same.

   (d) Shift-click and ctrl-click work as one might expect for selecting multiple signals. Select the three localparams added to the waveform and then either hit the *Delete* key (not backspace) on your keyboard or right-click and **Delete** to remove them.

   (e) At this point, those of you with prior experience might already be able to determine the combination of FSM type and code style used for FSMs 1 and 2.

However, there is no way to know what was used for FSMs 0 and 3, as their douts are asserted at the same time. So, we are going to need to take a look at each module's individual signals.

(f) From the **Scope** tab, select Generate_FSM[0].DUT. From the **Objects** tab, select din, dout and the three state signals and drag them into the viewer. You will notice that only din pops up with info. The other four signals are all currently blank. This is because to reduce log file size, the simulator only stores the data for signals already in the waveform viewer when the simulation is run. We will deal with this shortly.

(g) You will notice you now have two din and two dout signals. How do we keep track of which module each signal came from? To the best of my (limited) experience with the Vivado simulator, there is no option to toggle showing the full path for each signal like in Modelsim. As such, we have two options to keep stuff organized: dividers and groups. You can use whichever you prefer, my example has both options shown.

- If you select one or more signals and then right-click, there is an option to create a **New Group**. This gives you the ability to collapse/expand a group of signals to hide/view them based on what you are doing. Signals can be dragged in/out of a group as needed.
- If you right click anywhere in the waveform viewer, the option to create a **New Divider** is near the bottom. I used this to mark the signals for FSMs 2 and 3 in my example.
- Whichever option you go with, you can name the group/divider whatever you'd like to keep track of them.

(h) Repeat 6(f-g) for the other three FSMs.

7. With all the signals of interest added to the waveform it is time to re-run the simulation so we can actually see what was going on. From the main toolbar click the **Restart** button. By default, the toolbar is set for a 10us runtime, which is far more time than we need. Reduce it to .25us (or 250ns), then click the **Run for** button. *Note: this is the right arrow with the little (t) below it. Clicking the Run All button to its left will cause the simulation to run indefinitely until you click stop.*

8. The simulation should run pretty quickly. If you look at the state signals for each FSM, you should now be able to determine what type and style is being used for each. Take a screenshot of your completed waveform for submission. *Note: my example is colored, if you'd like to color your signals/dividers for easier viewing you can do so by selecting and right-clicking whatever signal(s) you want and selecting desired* ***Signal Color****.*

9. You can save this "waveform configuration" for future use. This will allow you to immediately load the waveform viewer with this set of signals. Click the **Save** button from the waveform viewer toolbar and a pop up save dialog for creating a .wcfg file will appear. Upon saving, Vivado will ask if you want to add this file to your project.

Doing so will allow the Simulator to automatically load the waveform viewer from the config file instead of using the default top-level signals when it launches.

10. I now want you to update the testbench, so open the source code for FSM_Example_TB.sv if you haven't already. Alter the BIT_SEQUENCE local param to contain two instances of the desired bit sequence. Note that your total bit sequence must be no more than 16 bits total (or update the size of *data*). The underscores are a visual delimiter to help see how long the sequence is. They can be used to group the 1s and 0s however you'd like, but by convention typically groups of 4 or sometimes 8 are used.

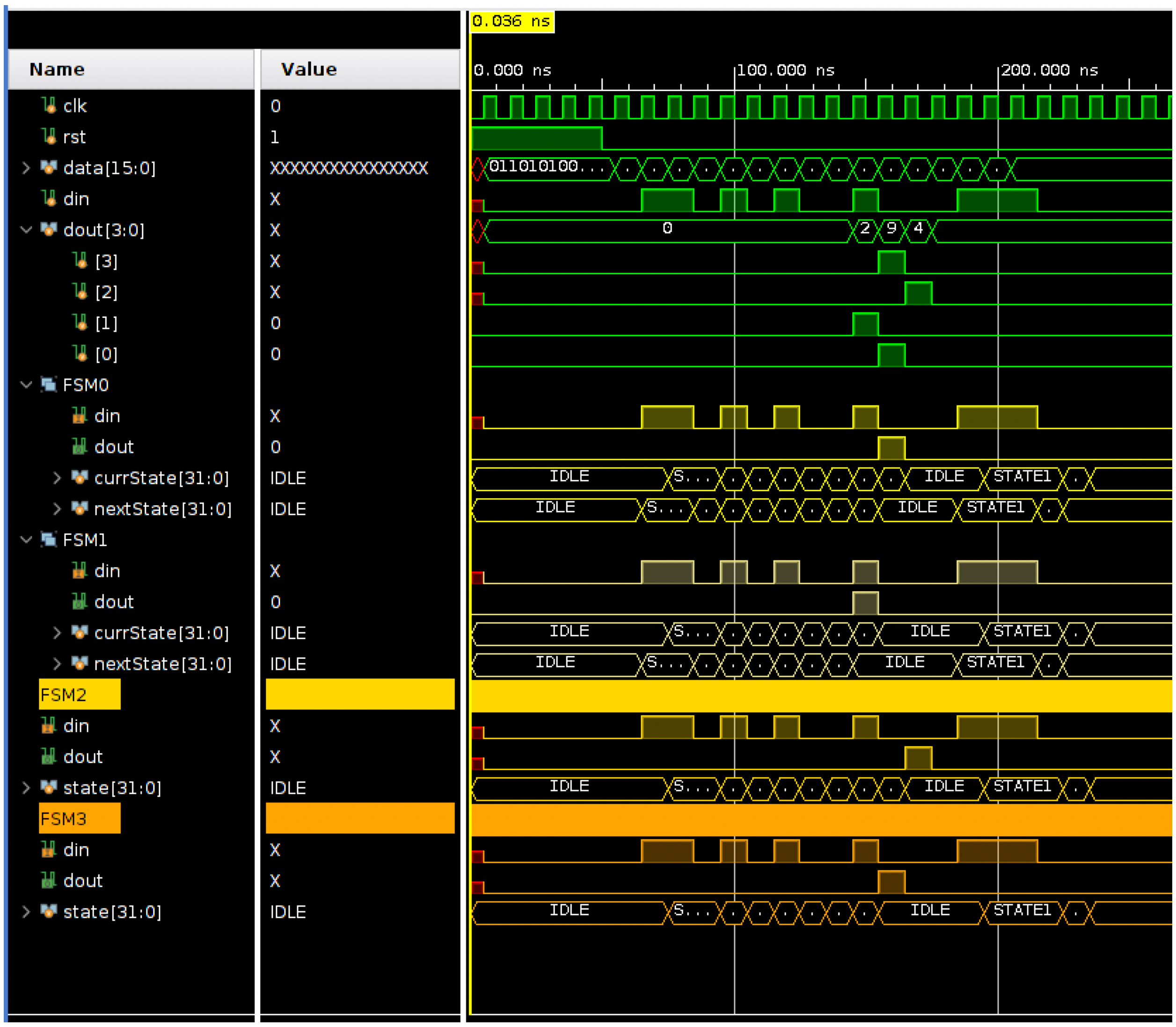
11. If you click the **Relaunch Simulation** button at this point, you will run the simulation with your updated sequence. This will lead to Error messages being printed to the Console due to the assert statement at line 38 failing when the FSMs detect the second iteration of the sequence. If you lengthened *data* you may need to run the simulation for longer than 250ns before this happens.

12. Update the code within that initial block (starting at line 30) to check that each FSM detects the sequence twice, and **Relaunch Simulation** again.

13. You have finished the lab. Answer the lab questions on Canvas, and upload your waveform screenshot and updated testbench code.

Lab Questions

Answer the questions for the lab on Canvas.



Name	Value
clk	0
rst	1
data[15:0]	XXXXXXXXXXXXXXXX
din	X
dout[3:0]	X
[3]	X
[2]	X
[1]	0
[0]	0
FSM0	
din	X
dout	0
currState[31:0]	IDLE
nextState[31:0]	IDLE
FSM1	
din	X
dout	0
currState[31:0]	IDLE
nextState[31:0]	IDLE
FSM2	
din	X
dout	X
state[31:0]	IDLE
FSM3	
din	X
dout	X
state[31:0]	IDLE

Example Waveform