

SANTA CLARA UNIVERSITY	ELEN 21L Fall 2022
Laboratory #8: Finite State Machines For lab sections Monday-Friday November 14- 18, 2022	

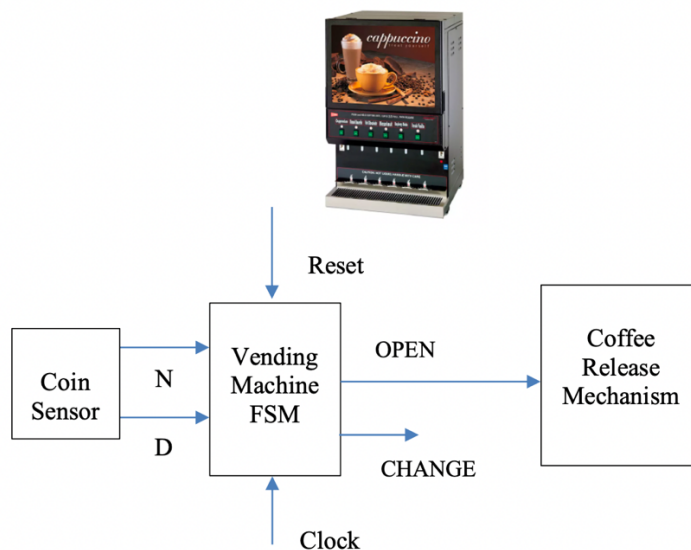
I. OBJECTIVES

- To translate a design problem statement into a Finite State Machine (FSM)
- To design, test, and implement an FSM based on a functional specification
- To explore and compare various designs using both Verilog and schematic for the design

PROBLEM STATEMENT

This lab is an extension of Example 8.7 in Section 8.6 of textbook *Fundamentals of Digital Logic Design with Verilog* 3rd edition by Brown and Vranesic.

You are to design the controller for a Vending Machine. The machine sells a cup of coffee for 15 cents. The machine ONLY takes Nickels (5 cents) and Dimes (10 cents). It doesn't take pennies or quarters or dollar bills. The machine cannot accept two inputs a time, only one input at a time. Once a sufficient amount of money (15 cents) has been entered, the machine will dispense the cup of coffee. The machine will provide change if more than 15 cents is entered.



The FSM design procedure includes the followings:

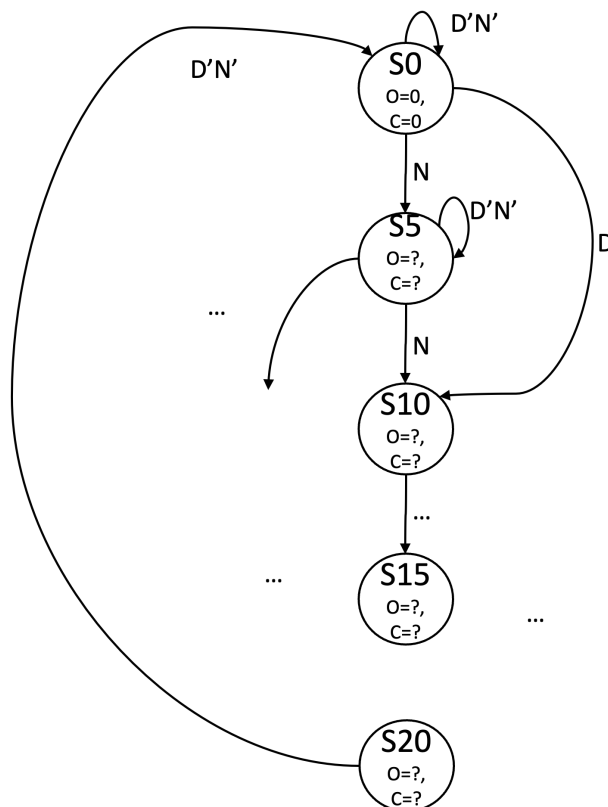
1. Describing a state diagram
2. Building a state transition table
3. State encoding
4. *Next State* and *Output* logic minimization (e.g., by using K-Maps)
5. Write a Verilog code

II. PRE-LAB

- (a) [State Diagram] We will use a Moore machine to implement this vending machine. Before building a state diagram, let us define five states as follows:

	Meaning	Output	
		<i>O</i> (Open)	<i>C</i> (Change)
<i>S0</i>	Balance = 0 cents	0	0
<i>S5</i>	Balance = 5 cents	0	0
<i>S10</i>	Balance = 10 cents	0	0
<i>S15</i>	Balance = 15 cents	1	0
<i>S20</i>	Balance = 20 cents	1	1

Based on this definition, complete the following state diagram:



In doing so, assume that

- Two inputs (*N* and *D*) are never asserted at the same time.
- In the states that there is enough balance for coffee, i.e., in *S15* and *S20*, none of the inputs is asserted.

- (b) [State Transition Table] Based on the state diagram obtained above, complete the state transition table below.

	Next States				Output	
	$N'D'$	$N'D$	ND'	ND	O (Open)	C (Change)
$S0$	$S0$	$S10$	$S5$	-	0	0
$S5$	$S5$?	$S10$	-	0	0
$S10$?	?	?	-	0	0
$S15$?	-	-	-	1	0
$S20$?	-	-	-	1	1

- (c) [State Encoding] As we have five states in our FSM, we need at least 3 bits to represent the state. We are going to use $S[2:0]$; perform the state encoding by completing the following table:

	$S[2:0]$
$S0$	000
$S5$	001
$S10$?
$S15$?
$S20$?

In doing so, try to minimize the number of gates to be used for the Output logic, i.e., the combinational logics that generate O and C .

Combining the above two tables, complete the final state transition table below:

	Current states ($S[2:0]$)	Next States ($S^*[2:0]$)				Output	
		$N'D'$	$N'D$	ND'	ND	O (Open)	C (Change)
$S0$	000	000	?	001	-	0	0
$S5$	001	001	?	?	-	0	0
$S10$?	?	?	?	-	0	0
$S15$?	?	-	-	-	1	0
$S20$?	?	-	-	-	1	1

- (d) [Logic Minimization] Now that we have the complete information for *Next State* and *Output* logics from the above table, let us perform optimizations. For each of

- $S^*[2]$,
- $S^*[1]$,
- $S^*[0]$,
- O , and
- C ,

obtain a minimized Boolean equation. You may or may not use K-maps.

- (e) [Implementation] Complete the Verilog implementation of your Moore machine design (based on `skeleton_moore.txt`).

Submit your Verilog code as your pre-lab assignment

III. LAB PROCEDURE


1. Reconcile your design choices with your lab partner

Compare the state diagram, state encoding, and transition table that you built in your pre-lab with what your partner has done. Decide between yourselves which version you will use.

2. Implement your Moore machine in a Verilog module.

- Create a Verilog file (*File* → *New...* → *Design Files* → *Verilog HDL File*). Then, copy and paste what is given in *module1.txt*.
- In order to make the port names consistent with the provided testbench, do NOT modify the input/output port specifications. For your information, the input/output port information is as follows in your schematic:

	Port name	
Inputs	Clock	Positive edge clock
	Reset	Positive edge reset
	N	Nickel sensor input (active-high)
	D	Dime sensor input (active-high)
Outputs	O	Open (Coffee out)
	C	Change
	S[2:0]	State (for debugging purpose)

- Complete the implementation and save it as *lab8.v*.
 - You need to complete “assign” statements for $S^*[2:0]$, O , and C .
 - Use the optimized Boolean equations you obtained in your pre-lab report.
 - See page 33 of Verilog Syntax Reference for an example Verilog assign statements.
- For compilation, use “Analysis & Elaboration” (), NOT a synthesis or full compilation (This lab is simulation-only).

3. Test your Moore machine (simulation).

- Use a testbench (*tb.v*, available to download on Camino) for testing your design, which we call a testbench. It will be a means for generating the clock and reset signals, and a framework for controlling the value of the N and D signal over time¹.
- The given test sequence is for the following scenario:
Reset → Nickel → Nickel → Nickel
- Modify the given testbench for testing the following scenarios:
 - Reset → Nickel → Nickel → Dime
 - Reset → Dime → Dime
 - Reset → Nickel → Dime
- Repeat the simulation for the above test sequences and demonstrate your work to your TA.

¹ Previously we used a tool (University Program VWF) for defining the input sequence to be simulated. But we can do this directly in a testbench Verilog module. If you are further interested, see pages 44-45 of Verilog Syntax Reference.

4. Another Verilog description method

- a. Instead of optimizing Boolean logic equations, we may use case or if-else statements.
- b. Replace the module description of *lab8.v* with what is given in *module2.txt*.
- c. Complete the implementation and repeat 2 and 3.
 - i. See pages 37-38 of Verilog Syntax Reference for case and if-else statements examples.
- d. Make sure this new implementation gives you the same simulation results.

When you have convinced yourself that it works, demonstrate to your TA.

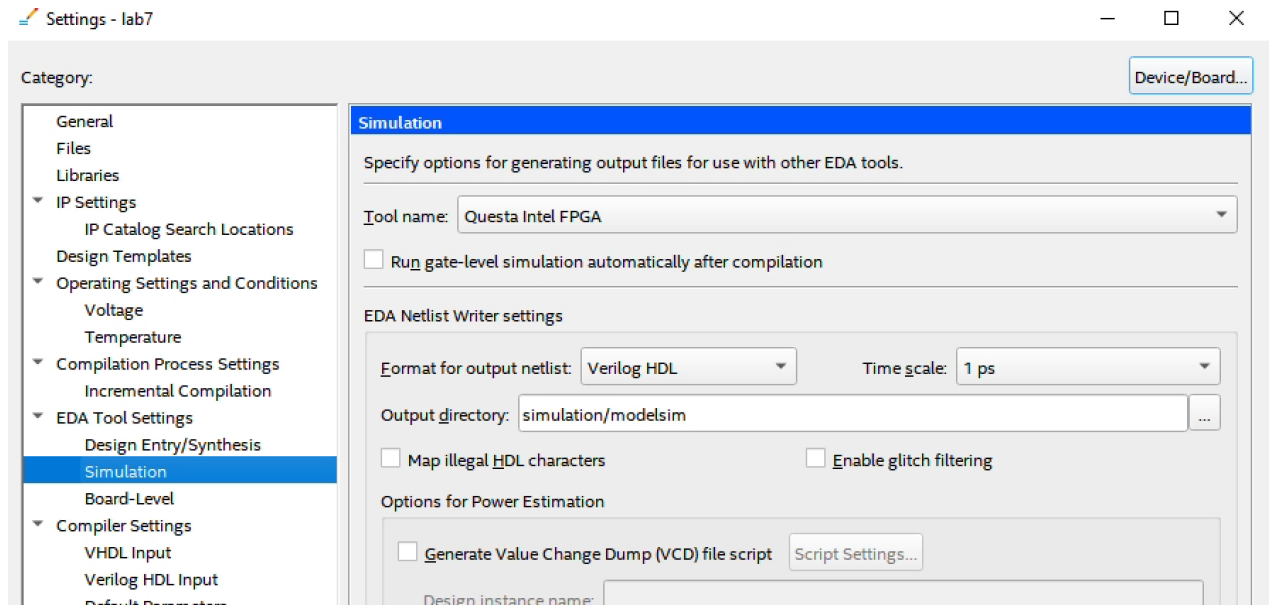
IV. REPORT

Your report should address the following:

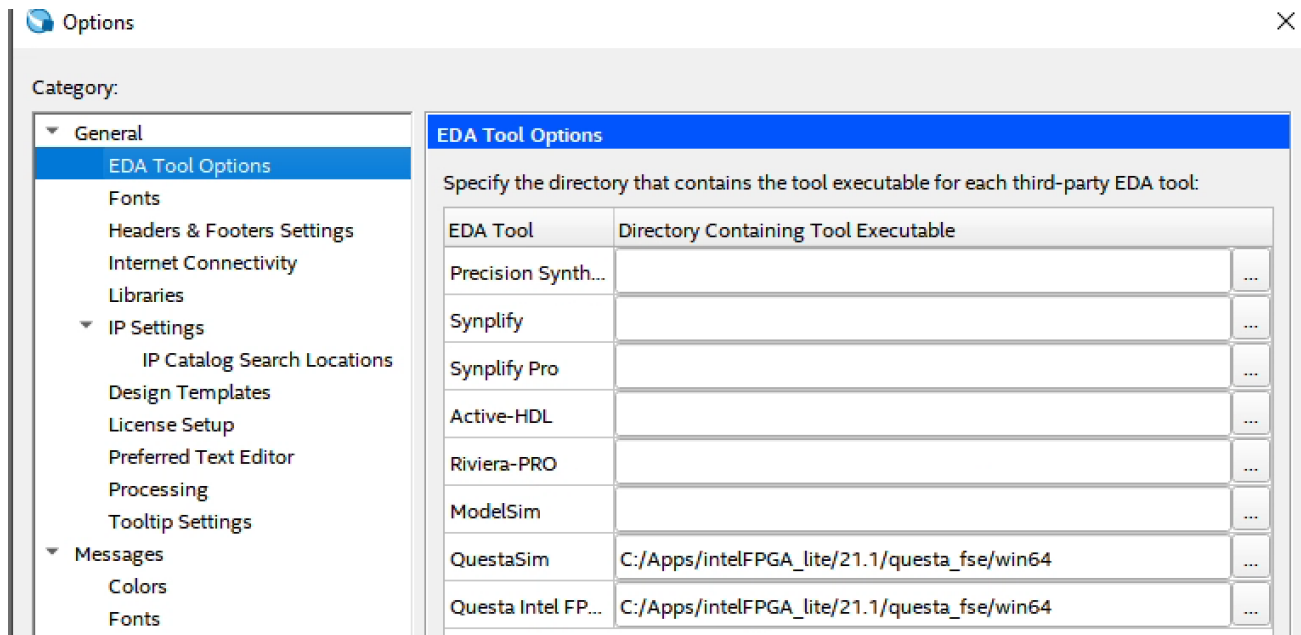
- Were the state diagrams/state tables you created in your pre-lab correct or not? If it was incorrect, in what way was it incorrect? What do you think led you to your incorrect diagram/table?
- If your pre-lab was incorrect, provide a corrected answer.
- Comparing the two different implementation methods, which one do you think is more productive (from the perspective of specification)? Why do you think so?
- How will your design change if you have to accept all types of coins (nickels, dimes, pennies and quarters). What about if you accept dollar bills?

V. REFERENCES

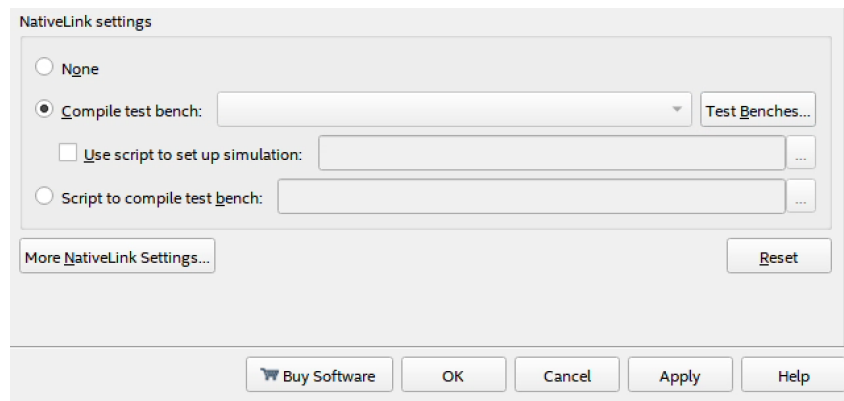
In order to simulate with Verilog code rather than the Waveform editor, make sure that simulation option has been set up correctly. To do this, first go to *Assignment* → *Settings...* → *EDA Tool Settings* → *Simulation*. For the “Tool Name” option, change it to “Questa Intel FPGA”.



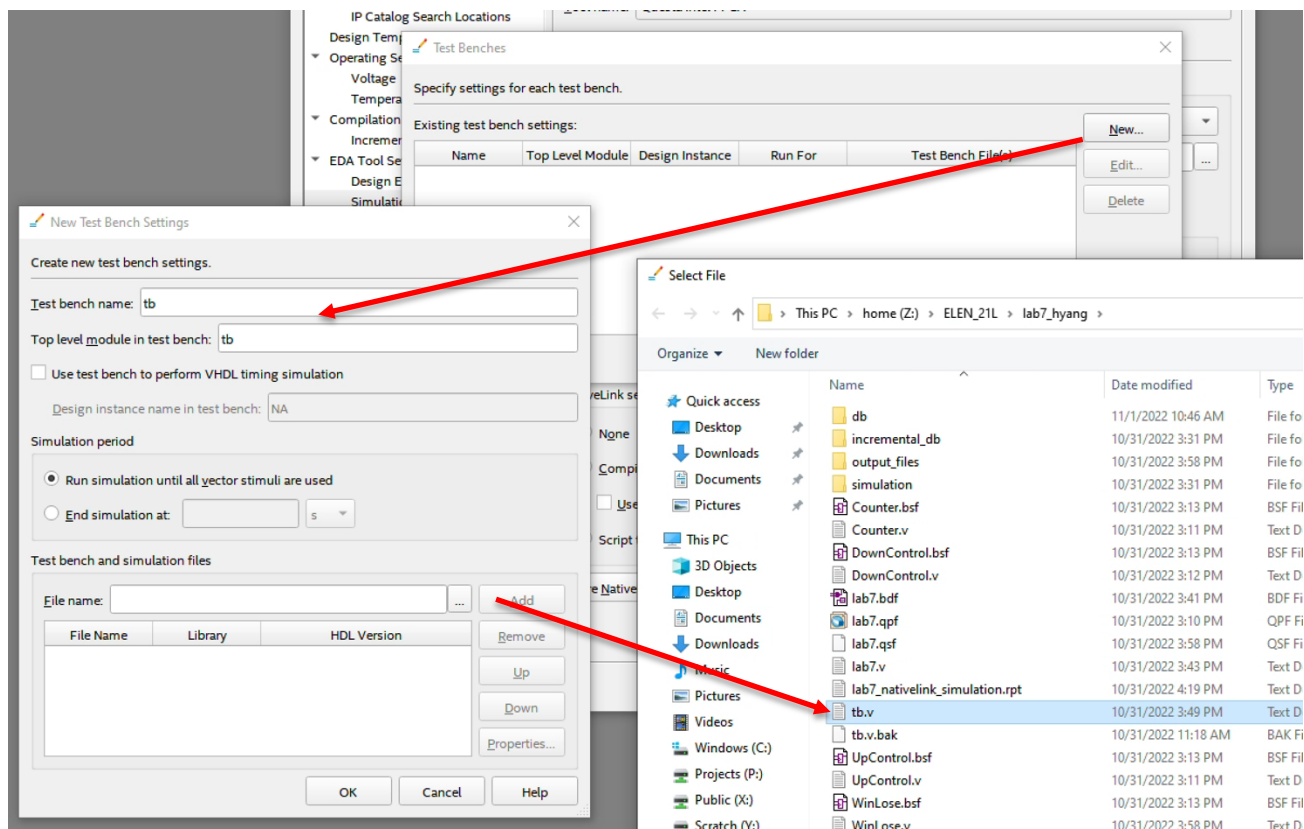
Also, make sure that the simulation tool paths are set correctly: *Tools* → *Options* → *General* → *EDA Tool Options*.



Put the provided testbench file (tb.v) in your project folder. Go to *Assignment* → *Settings...* → *EDA Tool Settings* -> *Simulation* and choose “*Compile test bench*” for the “*NativeLink settings*”. Then, click on “*Test Benches...*”



In the Test Benches panel, click on “*New...*” and set the “*Test bench name*” and “*Top level module in test bench*” to tb, which is the module name of the provided testbench (tb.v). Locate tb.v in your project folder and “*Add*” it in “*Test bench and simulation files*”.



When this is complete, go to *Tools* → *Run Simulation Tool* → *RTL Simulation*. (NOT *Gate Level Simulation...*) A simulation window should pop up after a delay.