UNIVERSITI TEKNOLOGI MALAYSIA

Finite State Machines with Quartus State Machine Editor

Muhammad Mun'im Ahmad Zabidi, Musa Mohd Mokji, Izam Kamisian, Norhafizah Ramli

Abstract
This articles introduces the reader to the concepts of finite state machines and their rapid implementation using Quartus state machine editor.

[1] *Department of Electronic and Computer Engineering, School of Electrical Engineering, Faculty of Engineering*
*__Corresponding author__: munim@utm.my

1. Introduction

Designing a finite state machine (FSM) is a common task for a digital logic engineer. The two basic types of FSM are Moore and Mealy, shown in Figure 1 and 2. Any FSM contains three basic components: state register, next state logic and output logic.

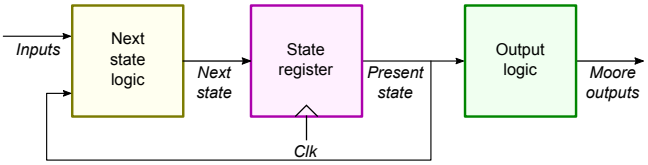

Figure 1. *Basic Moore type state machine.*

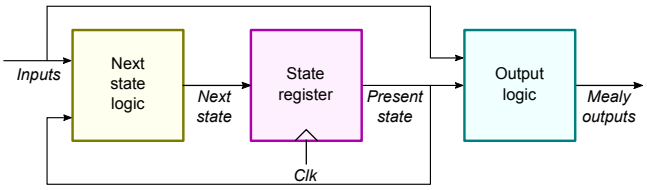

Figure 2. *Basic Mealy type state machine.*

State Register

The state register is simply a few flip-flops which store the present state. The **present state** is the value of the flip-flop at any given time while the **next state** is the value after receiving a clock edge. A FSM with n flip-flops has 2^n different states. The number of states is limited or finite, thus the name *finite* state machine. For example, an FSM with 2 flip-flops are limited to 4 states because they can only have 4 values: 00, 01, 10 and 11.

Next State Logic

The next state logic determines what the next state will be when a clock pulse arrives. The decision is based on the present state and input received from the outside world.

Output Logic

Output logic controls the outside world. Moore type FSM determines the system output based only on the present state (or the current value of flips-flops). Mealy type FSM determines the output based on present state and the present input. Due to this difference, Moore type FSM is easier to design, while Mealy type FSM is usually more efficient. Both types are widely used. Advanced FSMs may have both kinds of outputs such as in Figure 3.

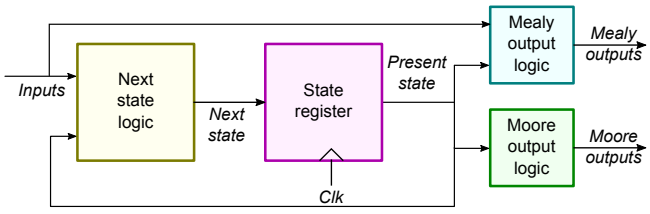

Figure 3. *FSM with both Moore and Mealy outputs.*

Finite state machines are modeled by state diagrams which describe the state changes that happen as inputs are received.

2. Moore-type Vending Machine

The vending machine in Figure 4 is a very common example used to demonstrate the concept of finite state machines. The vending machine releases a package of product (gum, soda, etc) after it has received 15 cents in coins [1].

The machine has a single coin slot that accepts *nickels* (5¢) and *dimes* (10¢), one coin at a time. A mechanical sensor indicates whether a dime or a nickel has been inserted into the coin slot. The controller's output causes a item of product to be released down a chute to the customer. The machine does not give change.

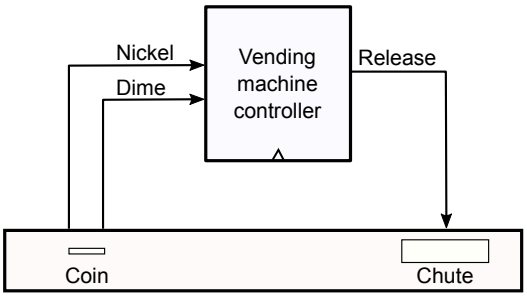

Figure 4. *Vending machine abstract view.*

The Moore machine symbolic state diagram for the vending machine is shown in Fig. 5. Each state represents how much money has been deposited into the vending machine.

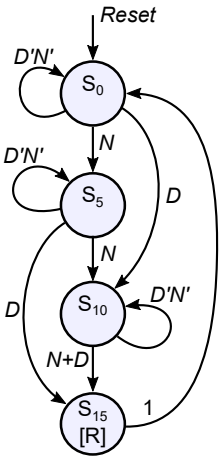

Figure 5. *The state diagram for vending machine.*

The FSM starts at initial state S_0 which means no money has been deposited. If a nickel (N)) was inserted, the FSM goes to state S_5. This is shown by the arrow labeled N. If a dime (D) was inserted in state S_0, the FSM goes to state S_{10}, represented by the arrow labeled D. If nothing was received, the FSM stays at state S_0, represented by the arrow looping to itself labeled $D'N'$.

The state machine assumes N and D can never occur at the same time. Although this assumption may simplify manual design of the state machine, it may cause conflicts.

Table 1. *State description.*

Symbol	Meaning
S_0	No money received
S_5	5¢ received
S_{10}	10¢ received
S_{15}	15¢ received

More on this later.

From either state S_5 or state S_{10}, more coins can be inserted. A nickel causes the FSM to advance to the immediately succeeding state. A dime causes the FSM to advance two states ahead. At state S_{10}, either a dime or a nickel advances the FSM to state S_{15}, represented by the arrow labeled $N+D$.

At state S_{15}, the FSM outputs the *Release* (R) signal which causes the gum to be delivered. Then the FSM is restored to the initial condition by an unconditional transition to state S_0. This unconditional action is represented by the arrow labeled 1 (or unlabeled).

An enhanced vending machine state diagram is shown in Figure 6. In this version, only arrows representing state transitions are shown. An unlabeled arrow is assumed to be an unconditional transition, e.g. $S_{15} \rightarrow S_0$. The enhanced state diagram updates the labels for $S_0 \rightarrow S_5$ and $S_5 \rightarrow S_{10}$ transitions to resolve conflicts.

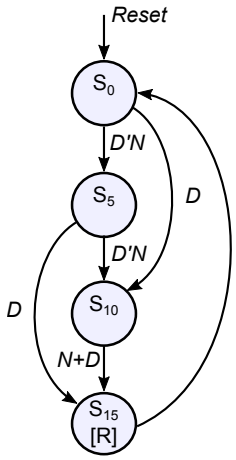

Figure 6. *Enhanced state diagram for vending machine.*

The next step in the design process is to enter the design using Quartus State Machine Editor.

3. Create a New Project

Launch the new project wizard in Quartus.

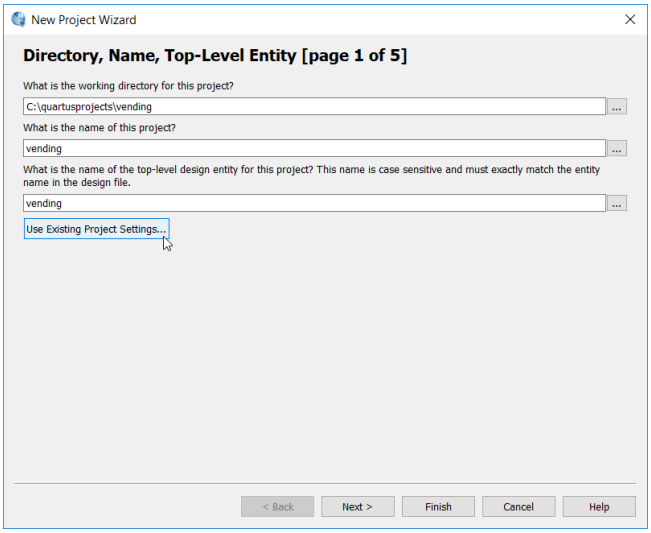

Figure 7. *New project wizard Page 1.*

After entering the folder name and project name, click Finish.

You can skip all the other pages. For this experiment, the device type is not important.

4. Design Entry

1. From the top menu, choose **File ➥ New ➥ State Machine File**.

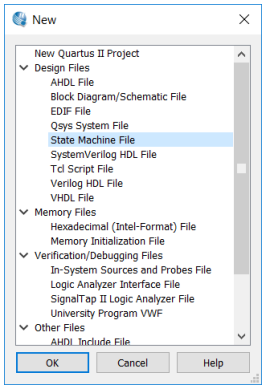

Figure 8. *New state machine.*

2. From the toolbar, select the **State Tool**.

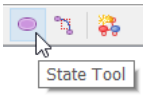

Figure 9. *State tool.*

3. When the icon changes, click four times to get four states. These are placeholders. You can place them any way, any where in the editor.

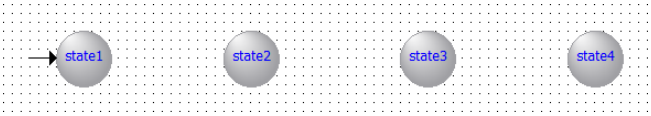

Figure 10. *Four placeholder states.*

You can skip Step 2 and 3. You can always add or delete states during Step 8.

4. From the toolbar, select the **State Machine Wizard** tool. Confirm editing of an existing design then click OK.

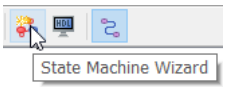

Figure 11. *Select the State Machine Wizard.*

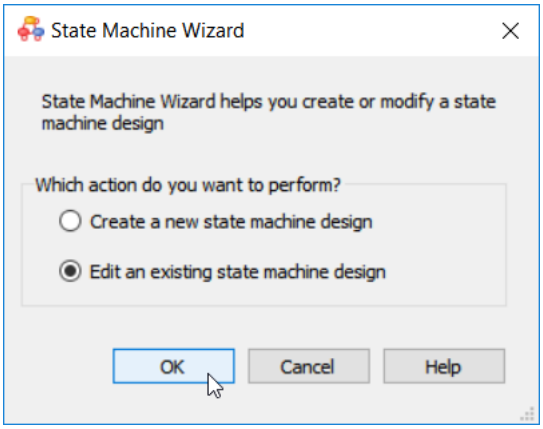

Figure 12. *Confirm edit of current state machine.*

5. In dialog box that appears, choose the General tab. Make sure the bottom checkbox is checked. This allows you to simplify the state diagram by not showing arrows pointing back to the source state.

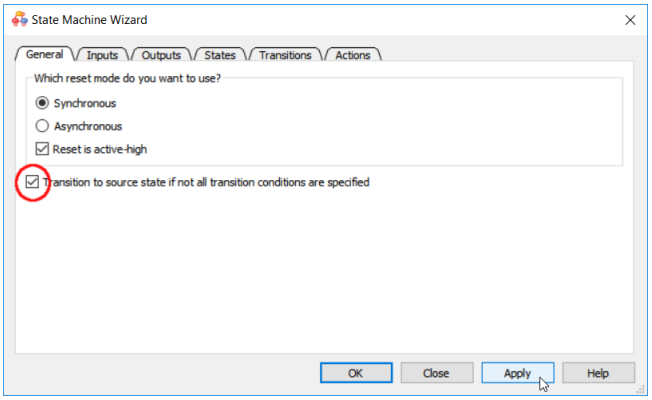

Figure 13. *General tab of State Machine Wizard.*

6. In the Inputs tab, click once on the <New> cell.

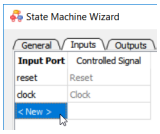

Figure 14. *The Inputs tabs initial condition.*

Add two new inputs, N and D.

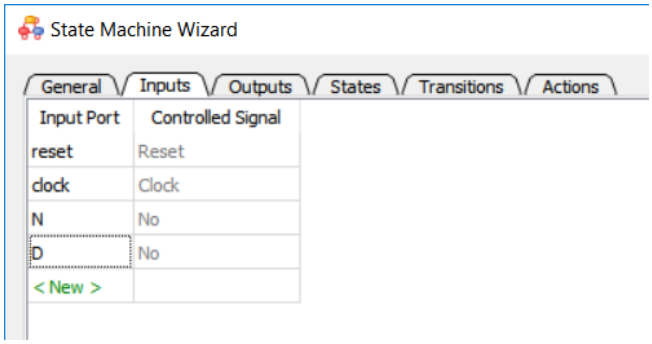

Figure 15. *Inputs N and D are added.*

7. In the Outputs tab, click on the <New> cell and add the output R.

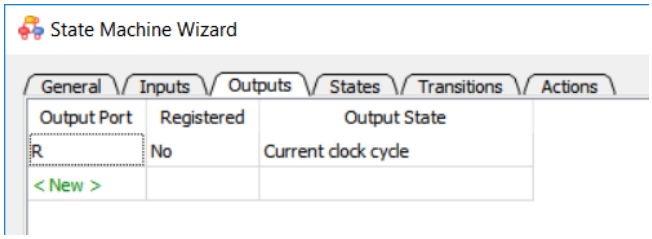

Figure 16. *The output R is added.*

8. The States tab defines all states in the machine. Here, rename all states to S0, S5, S10 and S15. The table should appear as in Figure.

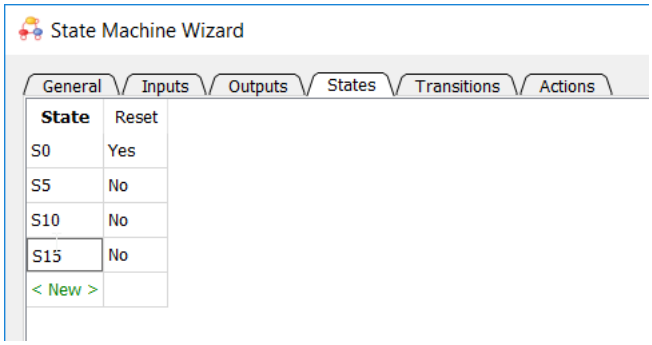

Figure 17. *States tab of State Machine Wizard.*

9. The Transitions tab define state-to-state transitions. Enter the transitions to get the table in Figure 18.

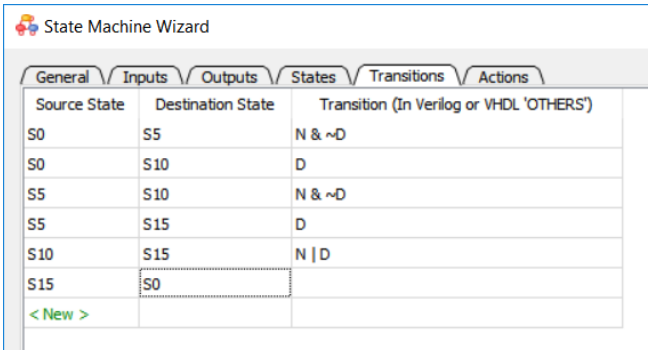

Figure 18. *Transitions tab of State Machine Wizard.*

The vending machine has only six transitions. Each row in the table represents one transition with a source state, destination state and the Verilog expression representing the Boolean equation. In Verilog, AND, OR and NOT use the operators `&`, `|` and `~`, respectively.

From S_0 in the original state diagram, the arrow labeled N goes to S_5 and the arrow labeled D goes to S_{10}. In the original design, we assume N and D are never active together. Quartus does not accept this assumption. The State Machine Editor will give an error unless we label the transitions as `D` and `N & ~D` (Boolean ND'). Alternatively, we can label the transitions as `N` and `D & ~N` (Boolean DN').

The non-ambiguity rule for S_0 is also applied to transitions from state S_5.

For the S_{10} to S_{15} transition, the expression `N | D` represents Boolean $N + D$.

The transition from S_{15} to S_0 is unconditional. It is labeled "1" in the state diagram (or unlabeled). In the Transitions table, leave the cell blank.

10. The Actions tab defines the outputs. In the original state diagram, the output R is active in state S_{15}. Enter the actions like in Figure 19. The value "1" in the Output Value cell means the output is active unconditionally during S_{15}. This is also called a Moore output.

    (To create a Mealy output, enter the Boolean expression the Output Value cell.)

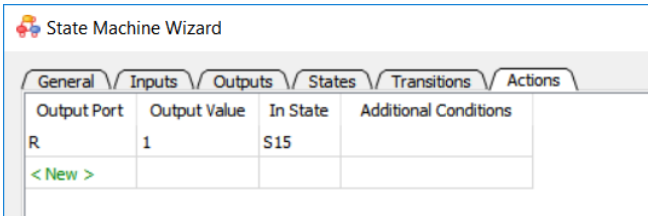

Figure 19. *Actions tab defines state machine outputs.*

11. All information have been entered. Click OK! We should get Figure 20.

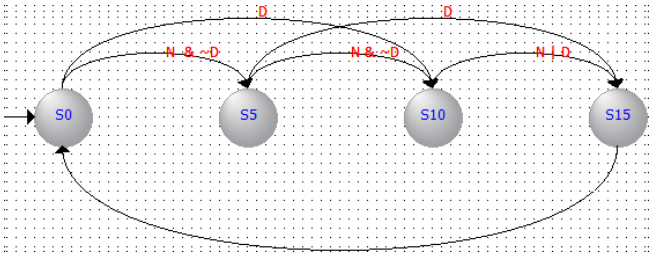

Figure 20. *Quartus rearranged states.*

12. Save the state machine file.

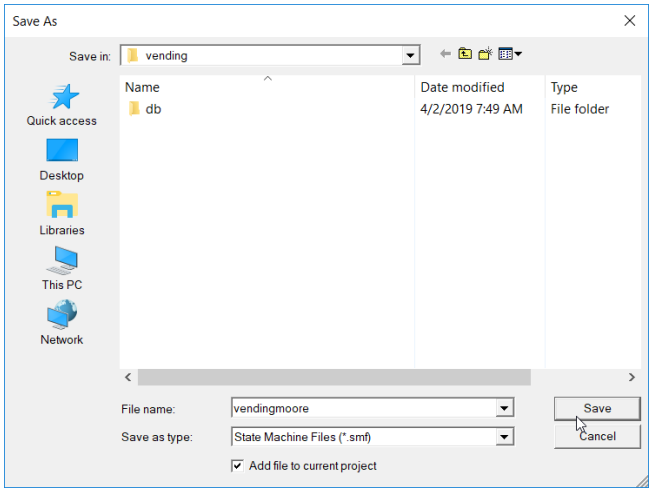

Figure 21. *Save the file.*

13. From the toolbar, click **Generate HDL File**. A dialog box will appear. The choice would not make any difference, so choose Verilog HDL anyway.

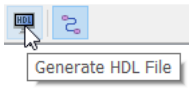

Figure 22. *Generate HDL.*

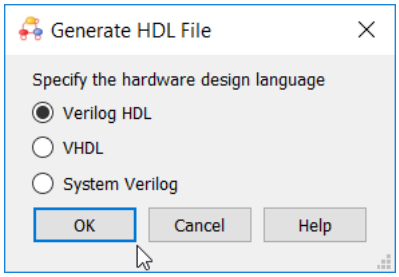

Figure 23. *Choose any language you like.*

13. After the HDL file is generated, add the file to the project.

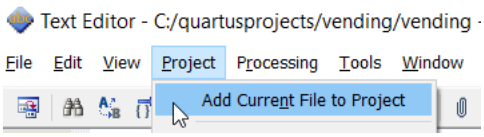

Figure 24. *Add the Verilog file to current project.*

14. Set the state machine file as the top-level entity. Note: you can do this step any time after creating a new state machine file but it must be before compiling the HDL file.

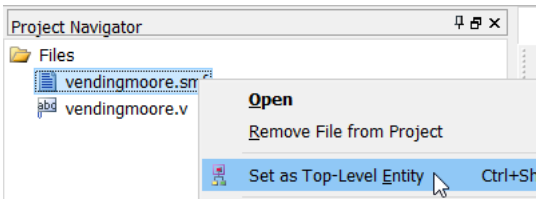

Figure 25. *Set vendingmoore.smf as top level entity.*

15. Compile the generated HDL code.

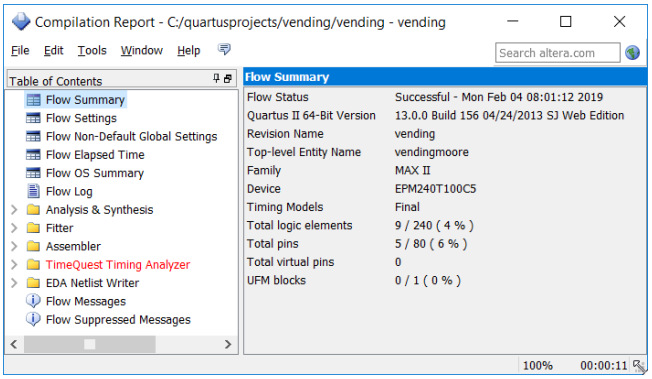

Figure 26. *Compilation report.*

After a successful compilation, you are now ready to simulate the vending machine.

5. Simulation

In order to simulate the design, we will enter the input simulation waveform.

1. Choose **File ➥ New ➥ University Program VWF** to create a new file (see Figure 27) then click OK.

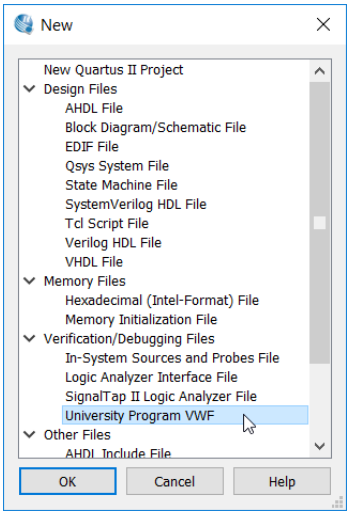

Figure 27. *Create new waveform file.*

2. Simulation waveform editor window will appear such shown in Figure 28.

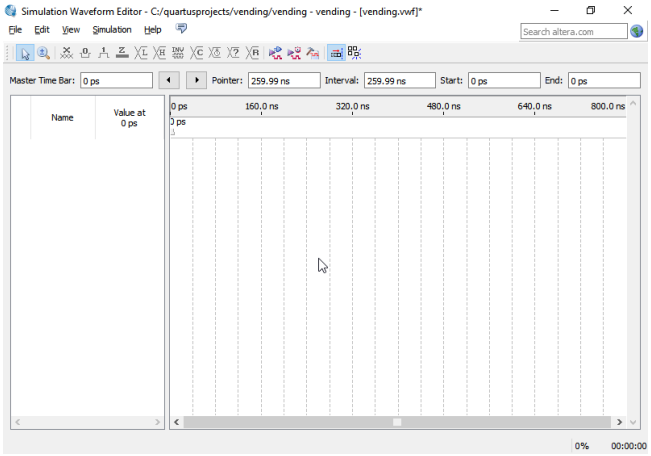

Figure 28. *Blank waveform.*

3. Choose **Edit ➥ Grid Size...**

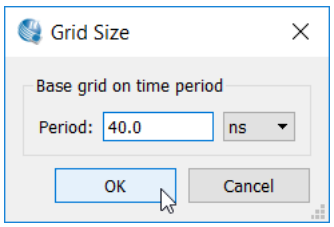

Figure 29. *Set Grid size to 40 ns.*

4. Choose **Edit ➥ Set End Time...**

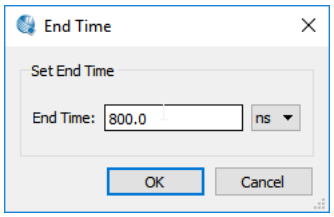

Figure 30. *Set end time to 800 ns.*

5. Go to **View ➥ Fit in Window** to get the Vector Waveform at the workspace. We can also adjust the view at our convenience through Zoom In and Zoom Out tool.

6. Go to **Edit ➥ insert** and click insert node or bus. A pop-up dialog box will appear as shown in Figure 31.

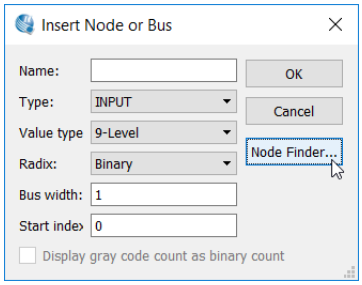

Figure 31. *Insert node dialog.*

7. Click Node Finder.. . A pop-up box will appear as shown in Figure 32.

8. Follow these steps:

   (a) Set the Filter to Pins:all
   (b) Click List
   (c) Click the >> button
   (d) Click OK on the Node Finder window

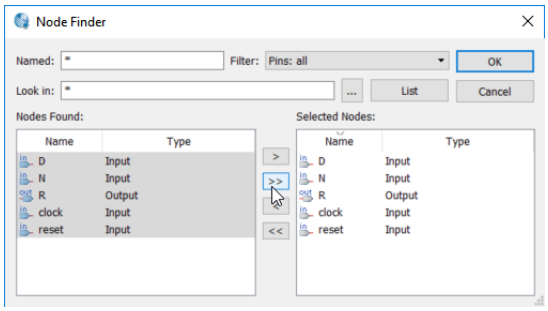

Figure 32. *Node finder dialog.*

9. We will be brought back at the Insert Node or Bus box as shown in Figure 33, click OK .

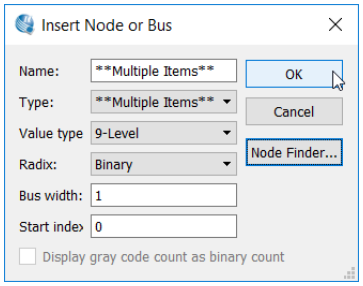

Figure 33. *Adding multiple nodes.*

10. At this stage we will see five signals on the waveform editor.

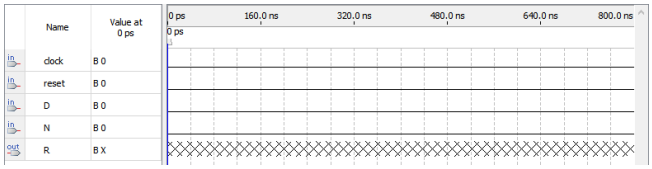

Figure 34. *Waveform after inserting nodes.*

11. Click on the clock signal, then the Overwrite Clock tool. Next, set the clock period to 40 ns.

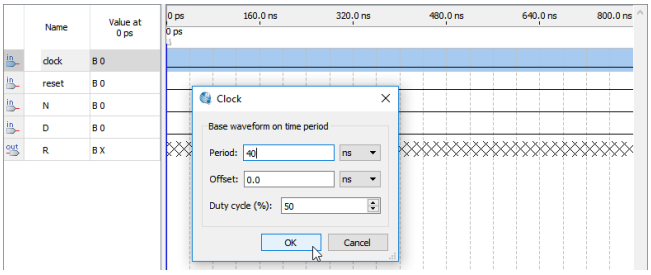

Figure 35. *Set the clock period to 40 ns.*

12. For the remaining signals, set the values according to Figure 36.

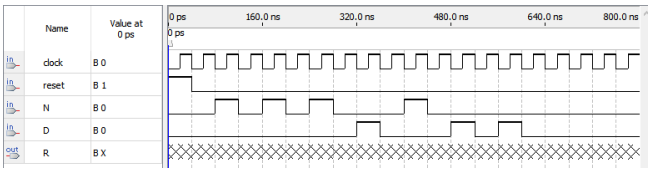

Figure 36. *Simulation input waveform.*

13. Choose **File ➥ Save as...**. A pop-up box will appear as shown in Figure 37, click Save.

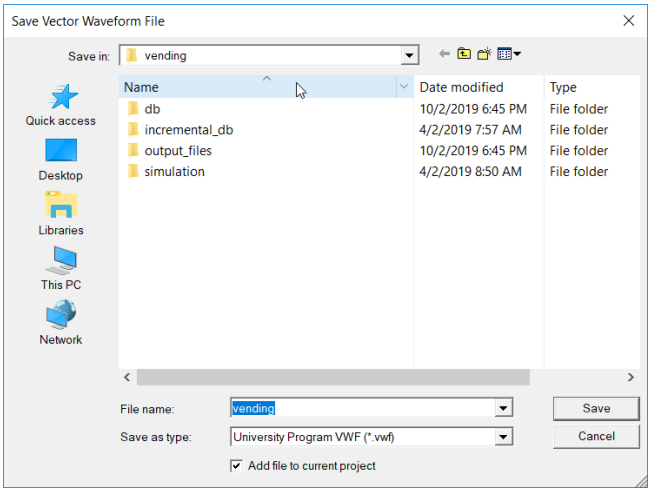

Figure 37. *Save the vector waveform.*

14. Click run functional simulation icon on the toolbar. If the simulation is successful, a new window will appear showing the result of simulated waveforms.

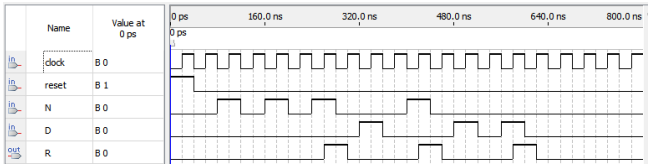

Figure 38. *Simulation output waveform.*

6. Mealy-type Vending Machine

The Mealy type vending machines solves the same task using only three states (Figure 39). This section of the application note only points the differences. Unless otherwise mentioned, use the same settings as the Moore machine.

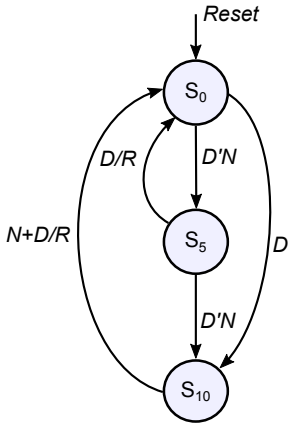

Figure 39. *The state diagram for vending machine.*

1. From the top menu, choose **File ➥ New ➥ State Machine File**.
2. In the State Machine Wizard, use the same settings in General and Inputs tabs as the Moore type vending machine.
3. In the Outputs tab, enter the *R* as output. Make sure the output is registered.

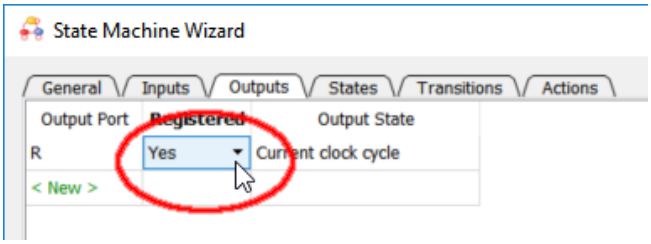

Figure 40. *Mealy output must be registered.*

4. In the States tab, enter the following states.

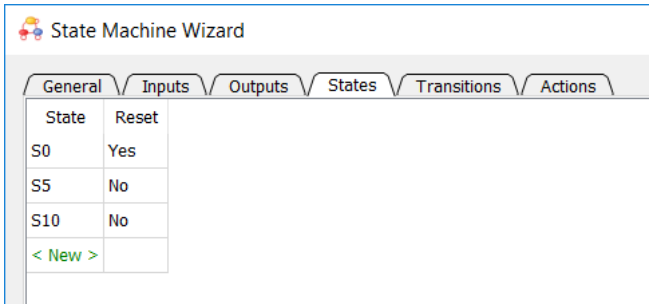

Figure 41. *Mealy states.*

5. In the Transitions tab, enter the following states.

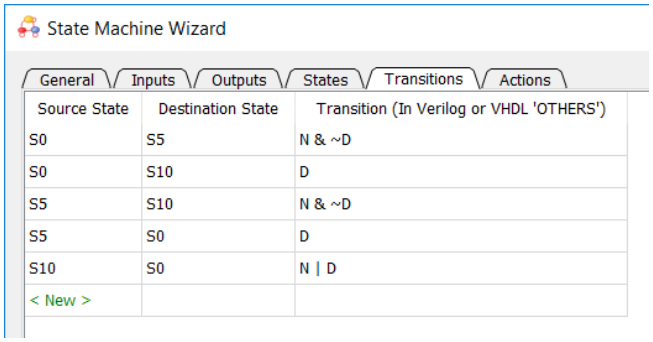

Figure 42. *Mealy transitions.*

6. In the Actions tab, enter the following states.

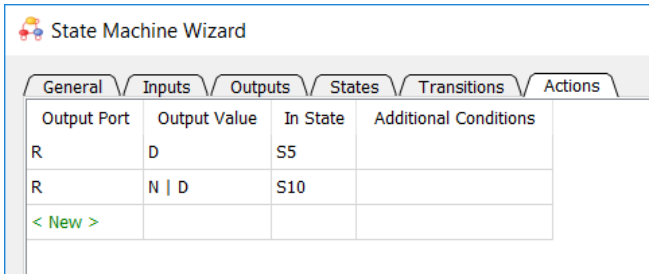

Figure 43. *Mealy actions.*

7. Clicking OK aftering entering all the information above gives the following state diagram.

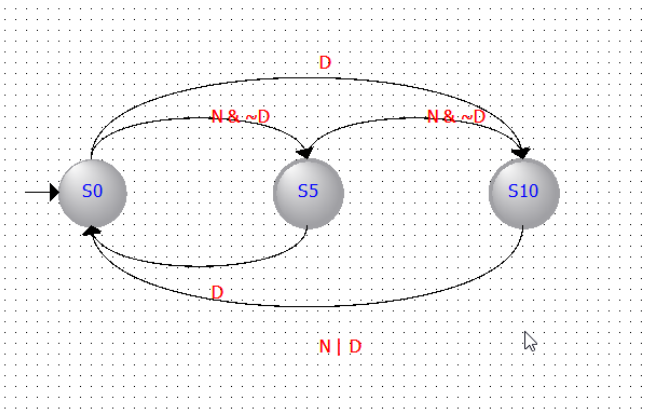

Figure 44. *Mealy state diagram.*

8. Save the file as **vendingmealy.smf**
9. Set vendingmealy.smf as **Top Level Entity**
10. General HDL.
11. Compile.
12. Simulate. You should get the same output as Figure 38.

Moore vs Mealy

This summary assumes the reader knows the difference between Mealy and Moore outputs.

Moore outputs are locked to the state. When asserted, Moore outputs are high as long as the FSM is in the particular state regardless of input changes. In *Actions* tab of the *State Machine Wizard*, enter 1 in the *Output Value* column. Refer Figure 19.

Mealy outputs asserted when certain conditions are met in a state. In *Actions* tab of the *State Machine Wizard*, enter the required Boolean expression in the *Output Value* column. Refer Figure 43.

Mealy outputs are glitchy. To make the outputs easier to analyze, set Mealy outputs as *Registered* in the *Outputs* tab in the *State Machine Wizard*. If you do not set it as registered, you must understand the effects of propagation delay, setup time and hold time of flip-flops to predict when the outputs are valid.

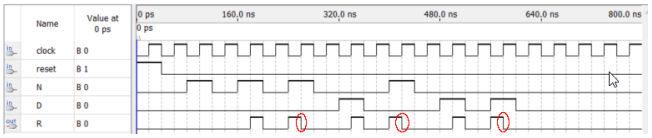

Figure 45. *Unregistered Mealy outputs: the circles indicate valid outputs. In Quartus, zoom in to inspect actual output values during the rising edge of the clock.*

References

[1] Randy H. Katz and Gaetano Borriello. *Contemporary Logic Design*. 2nd ed. Pearson, 2004.

[2] *Designing State Machines for FPGAs*. Application Note 130. Actel Corporation. Sept. 1997. URL: https://www.microsemi.com/document-portal/doc_view/130043-state-machine-an.

[3] Munim Zabidi, Izam Kamisian, and Ismahani Ismail. *The Art of Digital Design*. 2019.

[4] *Introduction to the Quartus® II Software*. Version 10. Altera. 2010. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/intro_to_quartus2.pdf.