UNIVERSITI TEKNOLOGI MALAYSIA

Implementing One-Hot State Machines Using Schematics

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

Abstract
This articles introduces the reader to the concepts of one hot state machines and their implementation on Quartus software.

[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. Usually the most important decision to make when designing a state machine is what state encoding to use. A poor choice of codes will result in a state machine that uses too much logic, or is too slow, or both.

In the *one-hot encoding* only one bit of the state vector is asserted for any given state. All other state bits are zero. Thus if there are n states then n state flops are required. State decoding is simplified, since the state bits themselves can be used directly to indicate whether the machine is in a particular state. No additional logic is required.

The first discussion of one-hot state machines was given by Huffman [1]. He analyzed asynchronous state machines implemented with electromechanical relays, and introduced a "one-relay-per-arrow" realization of his flow tables.

According to Golson [2], there are numerous advantages to using the one-hot design methodology:

- Maps easily into register-rich FPGA architectures such as Xilinx and Intel.
- One-hot state machines are typically faster. Speed is independent of the number of states, and instead depends only on the number of transitions into a particular state. A highly-encoded machine may slow dramatically as more states are added.
- Don't have to worry about finding an "optimal" state encoding. This is particularly beneficial as the machine design is modified, for what is "optimal" for one design may no longer be best if you add a few states and change some others. One-hot is equally "optimal" for all machines.
- One-hot machines are easy to design. Schematics can be captured and HDL code can be written directly from the state diagram without coding a state table.
- Modifications are straightforward. Adding and deleting states, or changing excitation equations, can be implemented easily without affecting the rest of the machine.
- Easily synthesized from VHDL or Verilog.
- There is typically no area penalty over highly-encoded machines.
- Critical paths are easy to find using static timing analysis.
- Easy to debug. Bogus state transitions are obvious, and current state display is trivial.

2. Vending Machine Example

The vending machine in Figure 1 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. 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.

The Moore machine symbolic state diagram is shown in Fig. 2. Each state represents how much money has been deposited into the vending machine.

The meaning of each state is listed in Table 1. The FSM starts at state S_0 which means no money has been deposited. If a nickel was inserted, the FSM goes to state S_5. This is shown by the arrow labeled with the variable N. If a dime was inserted in state S_0, the FSM goes to state S_{10}, represented by the arrow label D. If nothing was received, the FSM stays at state S_0, represented by the arrow looping to

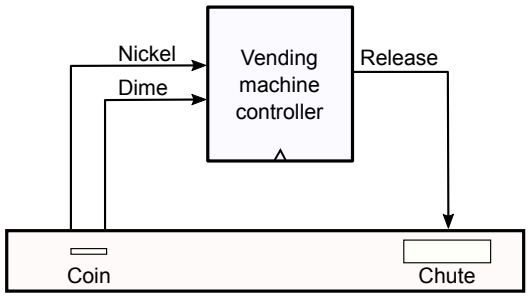

Figure 1. *Vending machine abstract view.*

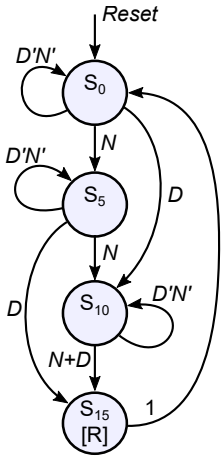

Figure 2. *The state diagram for vending machine.*

Table 1. *State description.*

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

itself labeled $D'N'$.

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* 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 action is represented by the arrow labeled 1.

An arrow labeled 1 in a multi-input state diagram has a different meaning compared to the state diagrams from the previous chapter. Previously, the state machines have only a single input, therefore there is no need to label the variables, and the arrows were labeled 0 or 1 according to the value of the single input variable. This state machine has two inputs, N and D. We must label each arrow with the Boolean expression formed by the input variables, hence the labels $D'N'$, $D+N$, D etc. An arrow labeled 1 on multi-input FSMs simply means the TRUE Boolean condition or a transition that always happens.

3. Deriving the Next State Equations

The vending machine controller can be implemented using one-hot state assignment by using the following state assignment. Remember that in one-hot state assignment, each state is represented by one flip-flop. The state the machine is in is indicated by the one and only flip-flop that is hot. Thus the name one-hot.

Symbolic	One-Hot Encoded
S_0 = 0¢	0001
S_5 = 5¢	0010
S_{10} = 10¢	0100
S_{15} = 15¢	1000

The next state equations are found by inspecting the state diagram. For each state, each incoming arrow produces one term in the next state expression. Fig. 3 shows how to derive the expression for S_0^+. In the state diagram, two arrows come into state S_0. This implies two ways to make the S_0 flip-flop hot. The first arrow comes from S_0 itself labeled $D'N'$. This generates the term $S_0D'N'$ where S_0 is the originating state and $D'N'$ is the condition. The second arrow comes from S_{15} unconditionally. This produces the second term $S_{15} \cdot 1$ or simply S_{15}.

The full list of next state equations are:

$$S_0^+ = S_0D'N' + S_{15}$$
$$S_5^+ = S_0N + S_5D'N'$$
$$S_{10}^+ = S_0D + S_5N + S_{10}D'N'$$
$$S_{15}^+ = S_5D + S_{10}N + S_{10}D$$

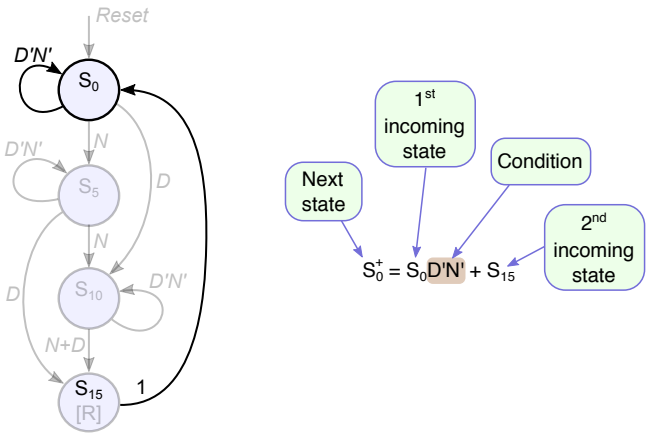

Figure 3. *Deriving the next state equations for one-hot controllers.*

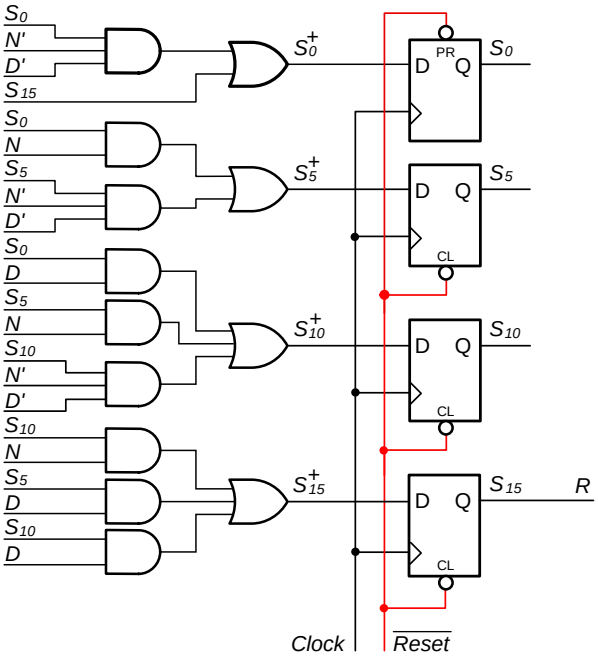

Figure 4. *One-hot Moore-type implementation of vending machine controller.*

Getting the output equation is straightforward. The output *Release* is only asserted when the FSM is in S_{15}. The *Release* signal is simply the output of the hot FF. Therefore, the output equation is:

$$Release = S_{15}$$

The controller implementation is shown in Fig. 4. The $\overline{Reset}$ input is connected to the preset input of the first flip-flop and to the clear inputs of the remaining flip-flops. This forces the system to state S_0 when $\overline{Reset}$ is asserted. The controller solved using one-hot encoding looks more complex than binary encoding, but the steps saved in deriving the next state logic reduce the chance for errors and encourage experimentation.

4. Automatic Reset Handling

An FSM starts operating from an **initial state**, such as S_0 in the case of the vending machine. The circuit in Fig. 4 has a manual reset input to put the FSM in the initial state. Manual reset is not practical for a system which is expected to run as soon as it is turned on.

4.1 Power-on-Reset (POR) Circuit

On circuits where the flip-flop values are unknown or random on power up, the FSM can be forced to the initial state by a reset pulse generated by an external power-on-reset (POR) circuit.

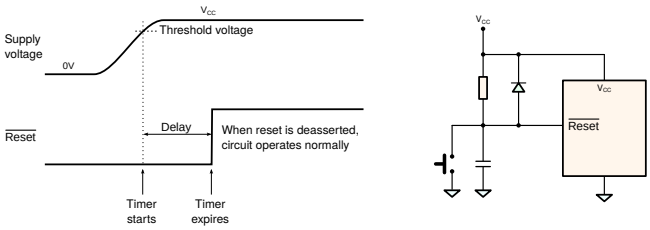

Figure 5. *POR holds the digital system in its reset state until the supply voltage exceeds the POR threshold and a specific delay period has elapsed.*

Fig. 5 shows the timing diagram for a POR. When the system is turned on, the supply voltage increases gradually until it reaches the full value. The POR signal is low at this time. On the way to reaching its full value, the voltage passes a threshold value. A timer is started when the threshold value is crossed. The POR signal is only released to high when timer has expired. This arrangement causes all flip-flops to be forced to the initial state for a predetermined delay. When POR is high, the FSM can jump to any other state.

Note that a human may reset the system by pressing the reset button at any time.

4.2 Inverted First Flip-Flop

On field programmable logic (CPLD and FPGA), all flip-flops contain zeros when a chip is powered up. For a highly encoded FSM, the all-zero condition (or the **none-hot state**) is usually the initial state and the POR circuit is unnecessary.

The none-hot state is *not* a valid operating state for one-hot encoded FSMs. But it can be made valid! The trick is to use negative logic for the first flip-flop. Simply add a NOT gate at the input and another NOT gate at the output. The following table lists the flip-flop values for our Moore vending machine.

Symbolic	Actual FF State	Interpretation
S_0 = 0¢	0000	0001
S_5 = 5¢	0011	0010
S_{10} = 10¢	0101	0100
S_{15} = 15¢	1001	1000

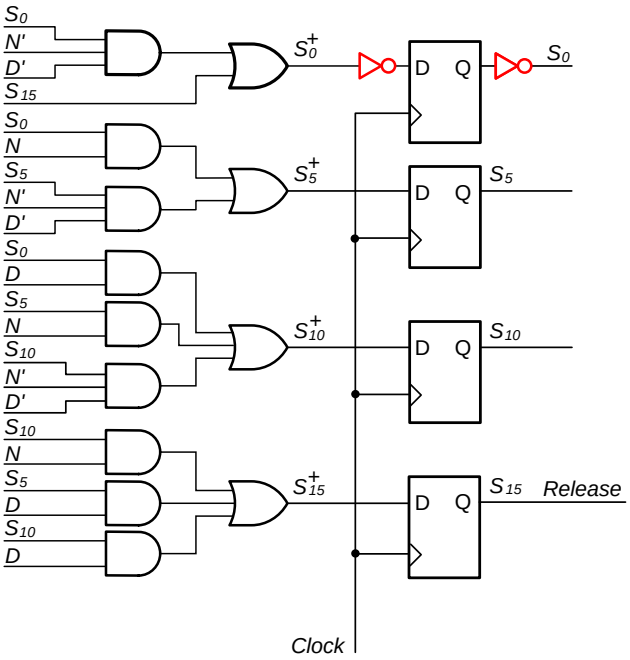

Figure 6. *Automatic reset handling of one-hot vending machine controller with negative logic first-flip-flop.*

4.3 The Ghost State

The none-hot state can be used to steer the FSM to the first valid state automatically. A NOR gate connected to the outputs of all flip-flops detects the all-zero condition. From all-zero state, the FSM goes to main state diagram. Consider the none-hot state to be a transient extra state (or ghost state) in the state diagram. The FSM never returns to the ghost state after the first clock cycle. The following table lists the flip-flop values for our Moore vending machine.

Symbolic	FF State
Init	0000
S_0 = 0¢	0001
S_5 = 5¢	0010
S_{10} = 10¢	0100
S_{15} = 15¢	1000

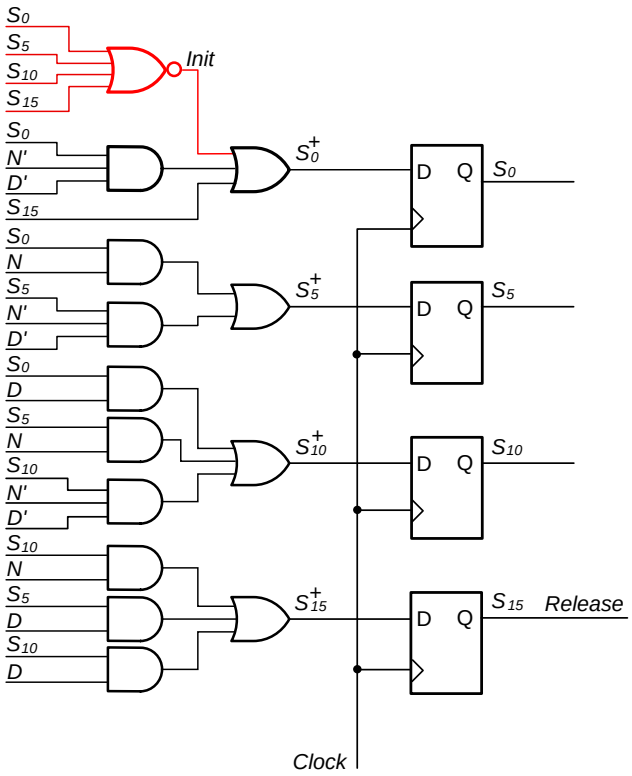

Figure 7. *Automatic reset handling of one-hot vending machine controller with none-hot state detector.*

References

[1] David A Huffman. "The synthesis of sequential switching circuits". In: *Journal of the Franklin Institute* 257.3 (1954).

[2] Steve Golson. "One-hot state machine design for FPGAs". In: *Proc. 3rd Annual PLD Design Conference & Exhibit*. Vol. 1. 3. 1993.

[3] *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.

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

[5] *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.