Tutorial: State Machines

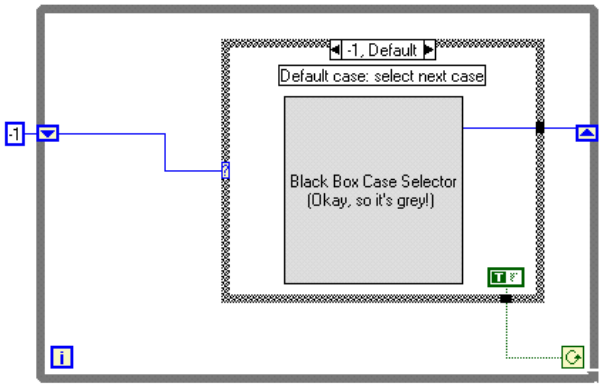
Publish Date: May 08, 2013 | 31 Ratings | **2.94** out of 5

Overview

The state machine is one of the fundamental architectures NI LabVIEW developers frequently use to build applications quickly. Developers use state machines in applications where distinguishable states exist. Each state can lead to one or multiple states and can end the process flow. A state machine relies on user input or in-state calculation to determine which state to go to next. Many applications require an “initialize” state followed by a default state, where you can perform many different actions. These actions depend on previous and current inputs as well as states. You can use a “shutdown” state to perform cleanup actions.

Table of Contents

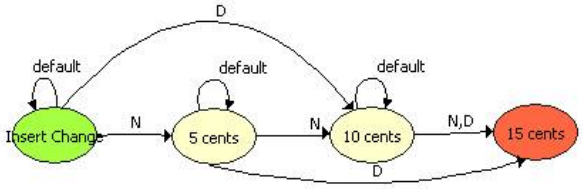
In LabVIEW software, you can create a basic state machine with a while loop, a shift register, a case statement, and some form of case selector (case selectors are discussed in a later section). The while loop is the main program loop, which executes until the conditions for exiting the program are met. The while loop’s main responsibility is to call the case selector and then execute the appropriate case. The shift register keeps track of which case should execute next. Finally, each case of the case statement contains the action for one specific use action. Often the default case is used as the place to check the case selector (in other words, if the user did nothing, check again to see if he has done something yet).



State Diagram

When designing state machines, you can create a state diagram to graphically represent the different states and how they interact. Use state diagrams, the design frameworks for state machines, to model the control algorithms you need with discrete logical states. State Diagrams make it easy to develop and understand the functionality of an application that uses a state machine.

The figure below is an example of a state diagram. The ovals represent the states and the arrows represent the possible transitions between states.



All applications require an initial state, or starting point, followed by transition states that perform different actions. A terminal state, or ending point, is the final state executed and performs cleanup actions.

State diagrams are useful in simplifying the design process of applications that use complex decision-making algorithms. To create an effective state diagram, you must know the various states of the application and how they relate to one another. By visualizing the various execution states of the application, you improve the overall design of the application.

Design a State Machine with a State Diagram

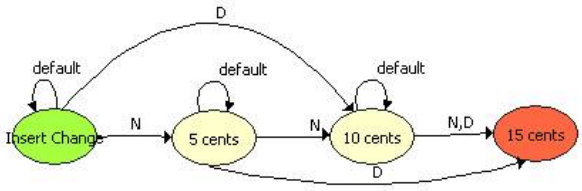
Imagine a vending machine that accepts combinations of nickels and dimes to get a coke. The cost of a coke is 15 cents and the machine does not return change. First, establish the states that the vending machine might be in:

- Start: No money inserted
- 5 cents
- 10 cents
- Done: 15 cents

Now think about the possible ways or paths that the vending machine can take to get into these states. Don’t forget to consider how the vending machine starts in the initial state.

- Nickel is inserted
- Dime is inserted
- Default: Nothing is inserted

From these lists, you see that there are four states and three possible paths from each state. You need to depict which states are connected by which paths. For example, when the vending machine is in the initial start state, the total change inserted is 0 cents. When a nickel is inserted, the vending machine must go to the 5 cent state. Therefore, the start state leads to the 5 cent state by the nickel path. By considering all states and paths, you can create a state diagram for the vending machine:



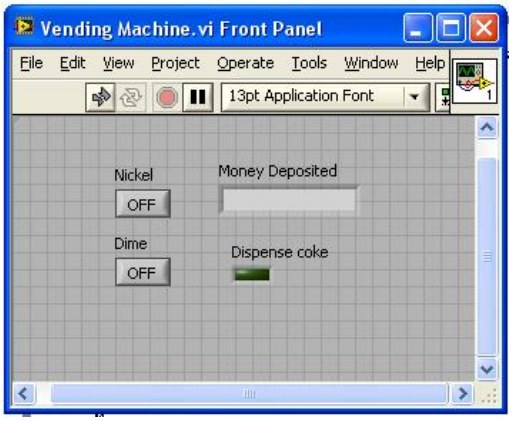
With a state diagram, you can better understand how to create a state machine.

Building a State Machine

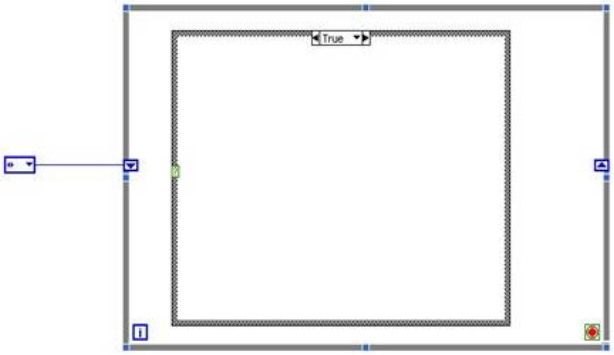
Using the state diagram above, create a state machine.

1. Create a new blank VI.
2. On the front panel place:

- Two text buttons named “Nickel” and “Dime”
- Text indicator named “Money Deposited”
- Boolean indicator named “Dispense coke”

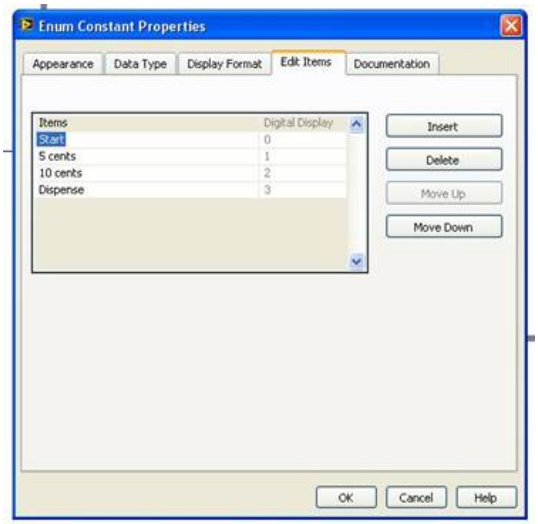


3. Place a while loop on the block diagram.
4. Place a case structure in the while loop.
5. Create a shift register on the while loop.
6. Create an Enum and wire it to the shift register to initialize it.

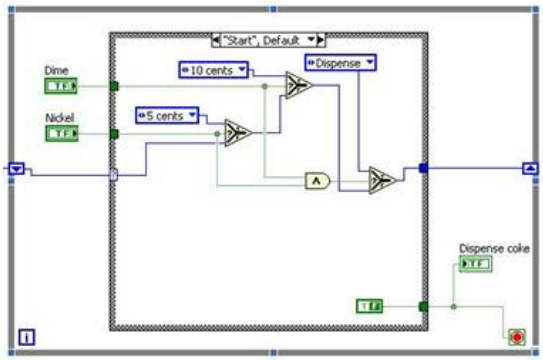


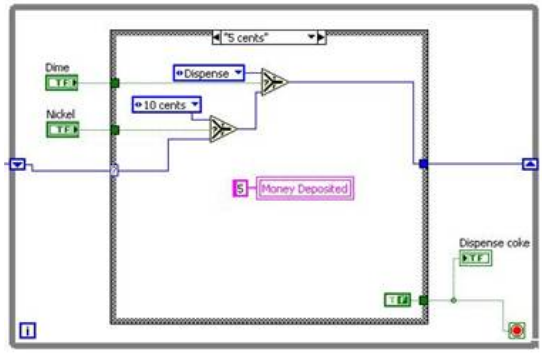
7. Right-click the Enum, select Edit Items, and add the following “states”:

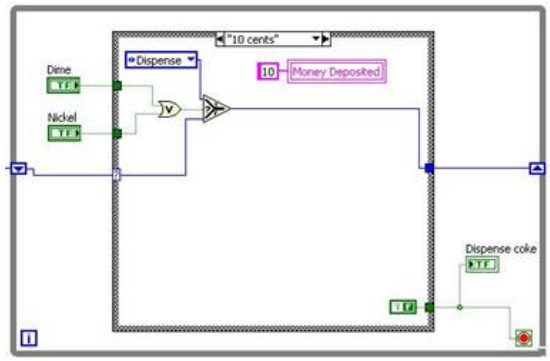
- Start
- 5 cents
- 10 cents
- Dispense

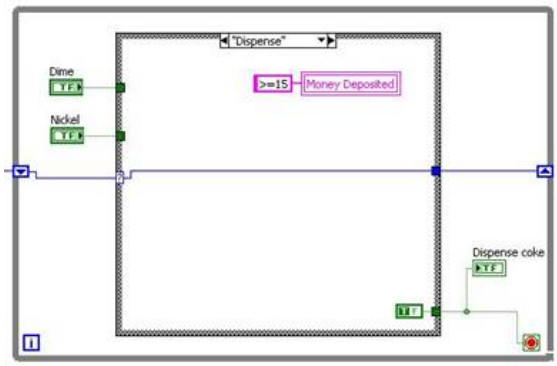


8. Wire the shift register to the conditional input of the case structure.
9. Right-click the case box at the top of the case structure and select **Add Case for Every Value**.
10. Wire the different cases as depicted in the following figures.









11. Wire **Dispense coke** to the Boolean output of the case structure.
12. Inside the while loop, place a wait function with a constant.
13. Outside of the while loop, wire a constant of 0 to the **Money Deposited** string indicator.

The finished VI should look like this: