Exercise 2-2: Event-Driven State Machine

Goal

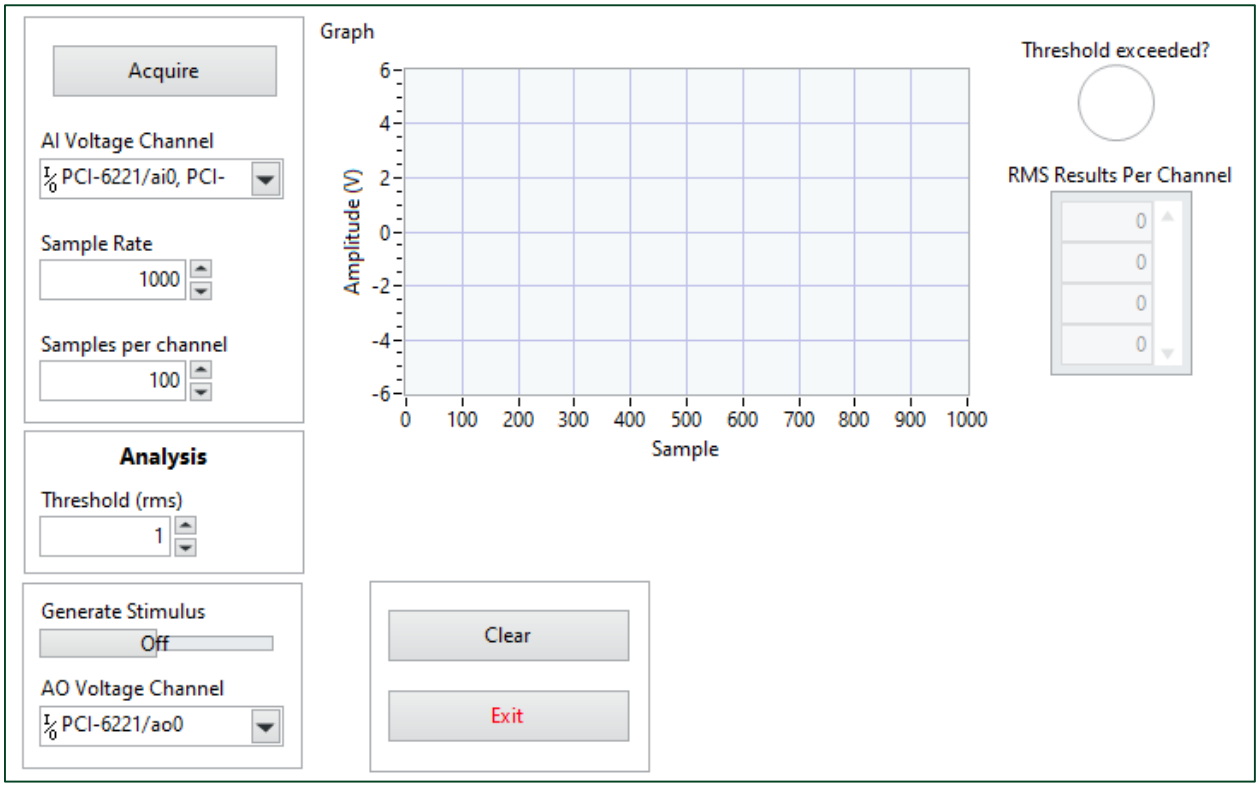
- In this exercise, you will configure the VI that uses the Event-Driven State Machine design pattern and explore its limits and the caveats. After that, you will analyze the Event-Driven State Machine template provided by LabVIEW.

Hardware Setup

(Hardware) In the exercises where we work with Analog Input/Output channels, we use PCI-6221/USB-6212 multifunction I/O device paired with the BNC-2120 shielded connector block. Analog Input 2 should be connected to the Sine/Triangle BNC connector. Analog Input 3 should be connected to the TTL Square Wave BNC connector. The Sine/Triangle waveform switch should be set to Sine.

Scenario

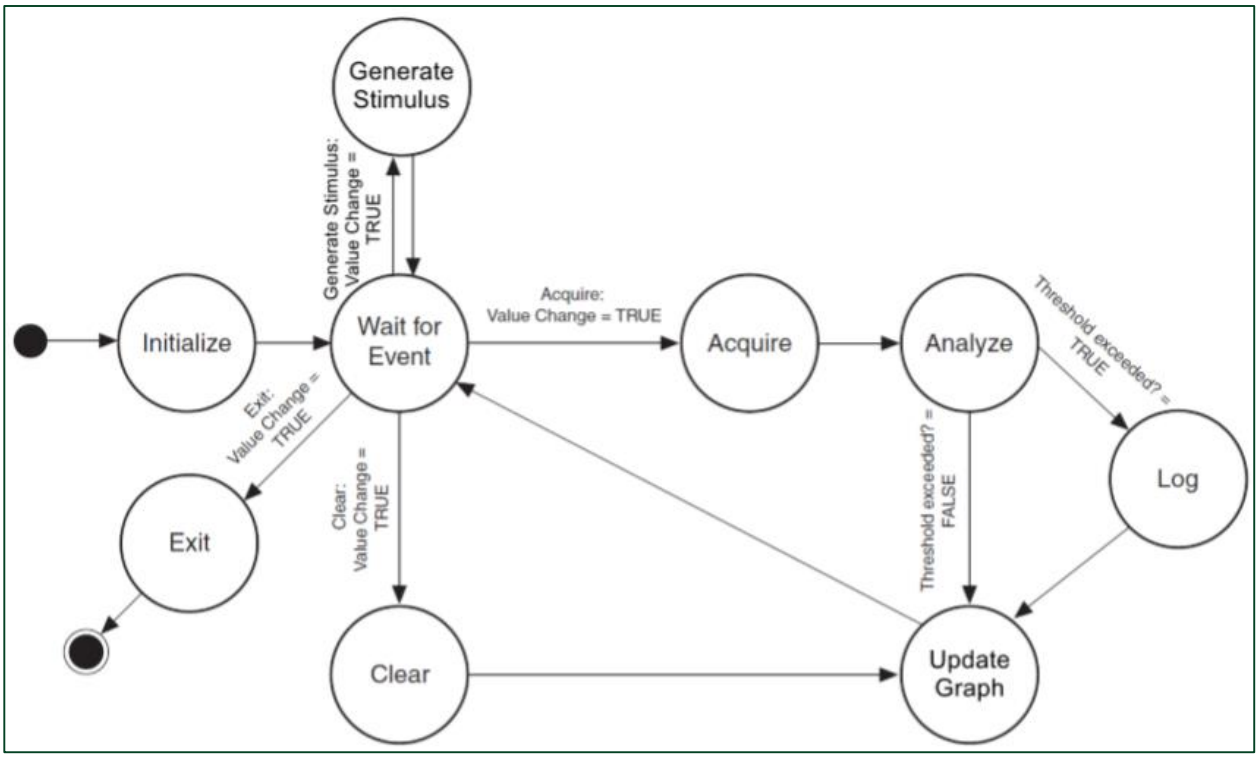
You will use an unconfigured Event-Driven State Machine the front panel of which is shown in the following figure.



The following table lists the events in this application and their corresponding state diagram logic.

Event	State Diagram Logic
“Acquire”: Value Change	Acquires data, analyzes the data, and logs data if it exceeds the threshold, updates the graph, and then waits for another event.
“Clear”: Value Change	Clears the graph data, and then waits for another event.
“Generate Stimulus”: Value Change	Generates either 5V (ON position) or 0V (OFF position) at the selected analog output, and displays a dialog box informing the user about it, and then waits for another event.
“Exit”: Value Change	Exists the application.

Examine the state diagram for this application in the following figure.



This application uses the Event-Driven State Machine design pattern for the following reasons.

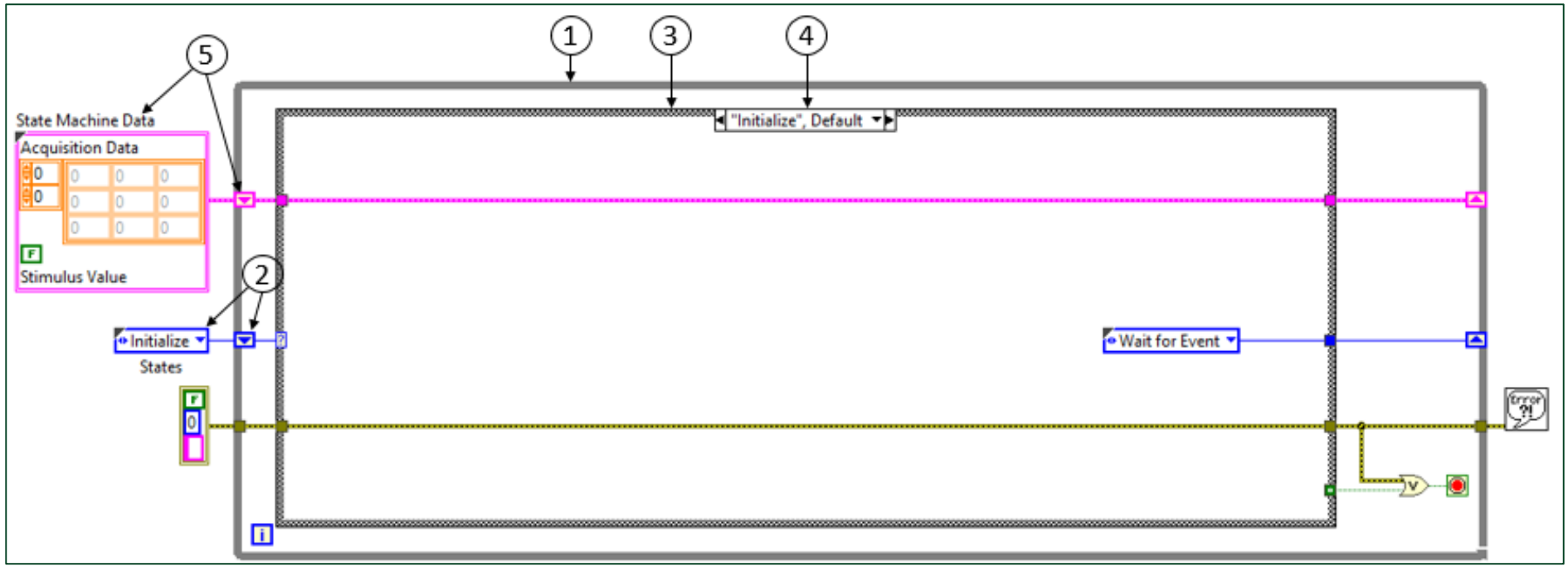
- This application is event driven. The user will click **many** buttons while the application is running, and those Value Change events must trigger the corresponding code to execute.
- The code that corresponds to Value Change events can be described by a state diagram.

Guided Instructions

In this section, you will implement the state transition logic described by the arrows in the previous figure.

1. Open `C:\Exercises\LabVIEW Core 2\Event-Driven State Machine\Event-Driven State Machine.lvproj`.
2. From the **Project Explorer** window, open the Event-Driven State Machine VI.
3. Examine the front panel. In this application, clicking the buttons described in the table on the first page will execute the corresponding code.

4. Examine the Event-Driven State Machine design pattern.
   - Explore the block diagram and notice the state machine components as shown in the following figure.



1. **While Loop**
2. **Type Definition enum** – Click on this **enum** and notice that it contains all the states described in the state diagram shown in the previous figure.
3. **Case structure** – Click on the **Case Selector** and notice that there is a case for every state described in the state diagram shown in the previous figure.
4. **Cases** – Each case contains both state code and the state transition code.
5. **State Machine Data** – The Type Definition cluster constant defines the state machine data. The shift register stores the data and allows multiple states to access and modify the data.

Question 1: What is the first case that this VI will execute?

_____

5. Examine the **Initialize** case.
   Question 2: What case will the next iteration of the While Loop execute after the Initialize case?

   _____

   _____

6. Examine the Wait for Event case. This case exists in every application based on the Event-Driven State Machine design pattern.
   - Notice that this case contains an Event structure.
   - Based on the state transition diagram, what events do you expect to see defined in the Event structure?

     _____

     _____

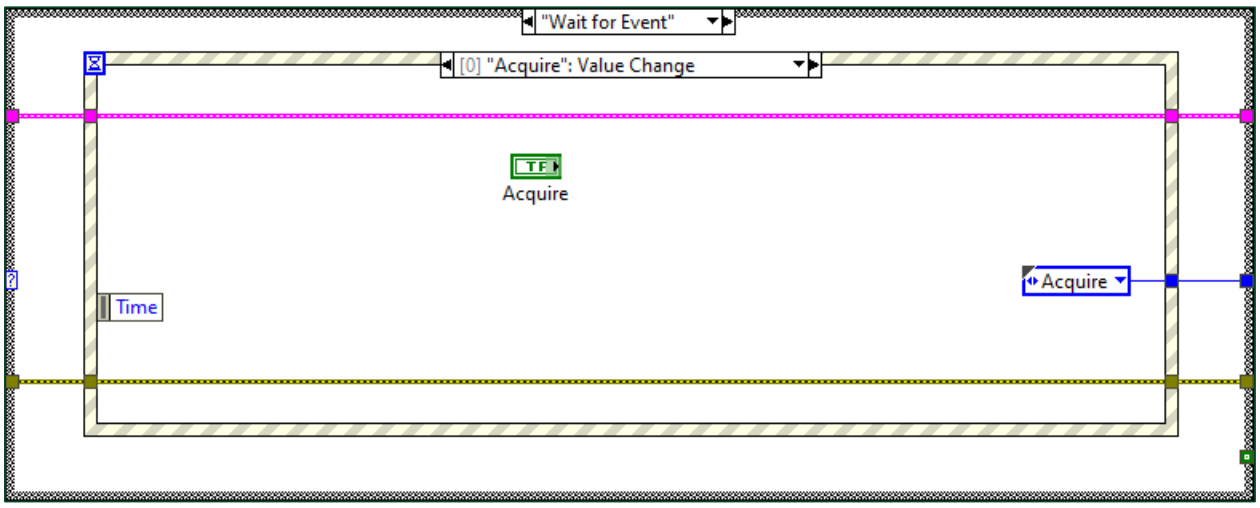
     _____

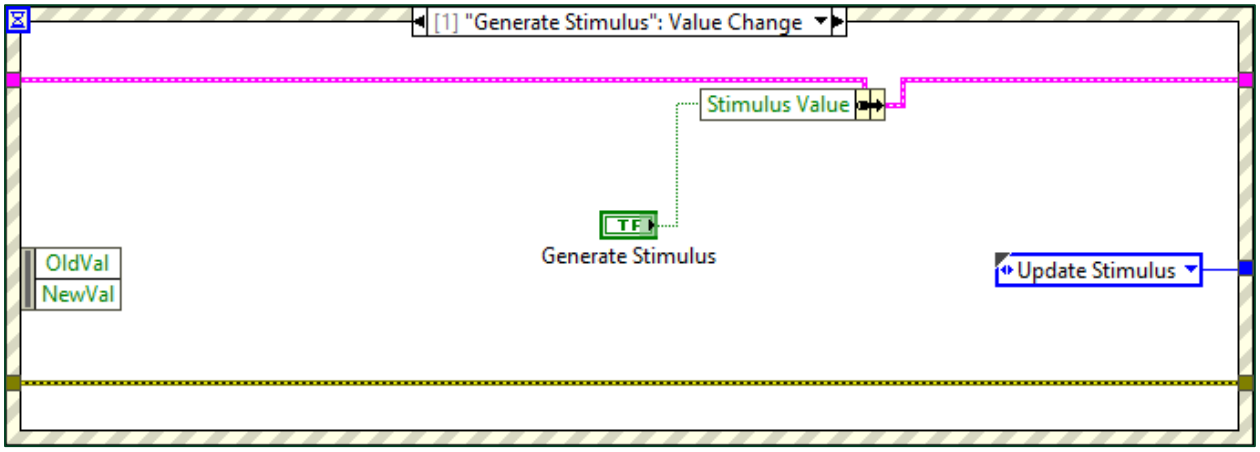
     _____

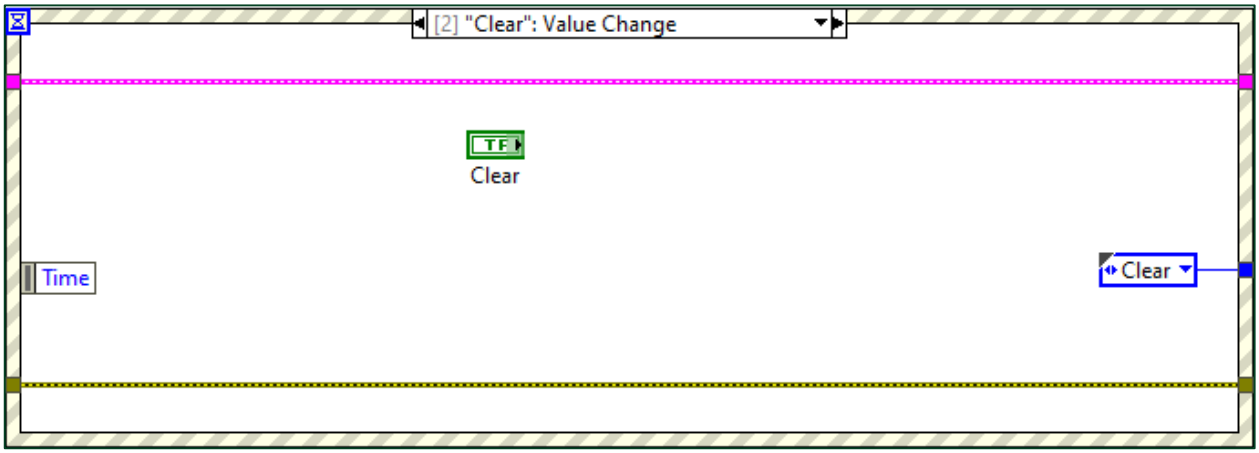
   - Click the **event selector** of the Event structure to view the defined event cases.
   - Based on the state transition diagram, write the state transition enum value you need in each event case.

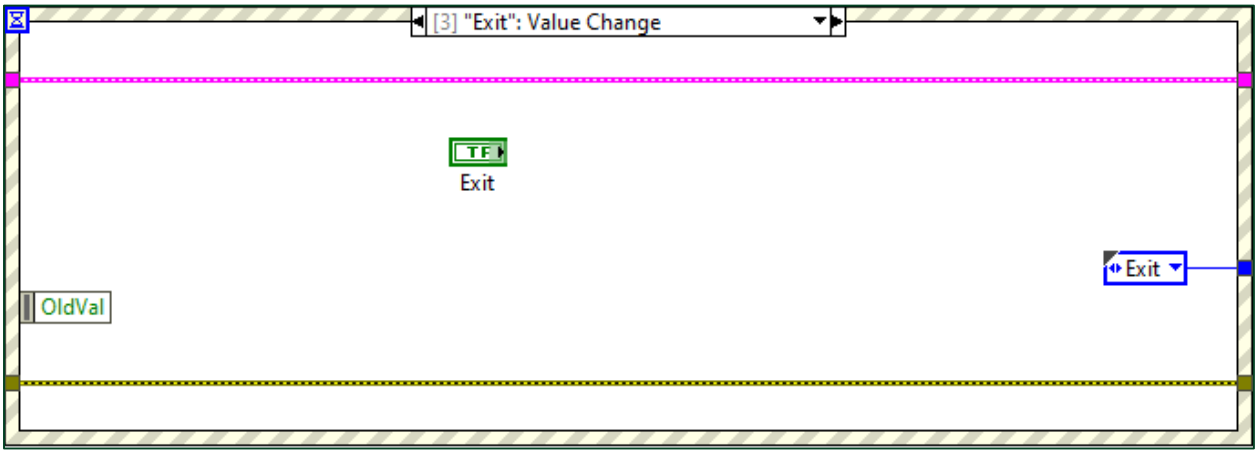
Event Case	States Enum Value (Initialize, Wait for Event, Acquire, Analyze, Log, Clear, Update Graph, Update Stimulus, Exit)
“Acquire”: Value Change	
“Clear”: Value Change	
“Generate Stimulus”: Value Change	
“Exit”: Value Change	

- In the **Project Explorer** window expand **support** folder, and drag the **States.ctl** enum to the following event cases on the block diagram. Set this enum to the values (Acquire, Update Stimulus, Clear, Exit) shown in the following figures. This defines which state the state machine will transition to next.

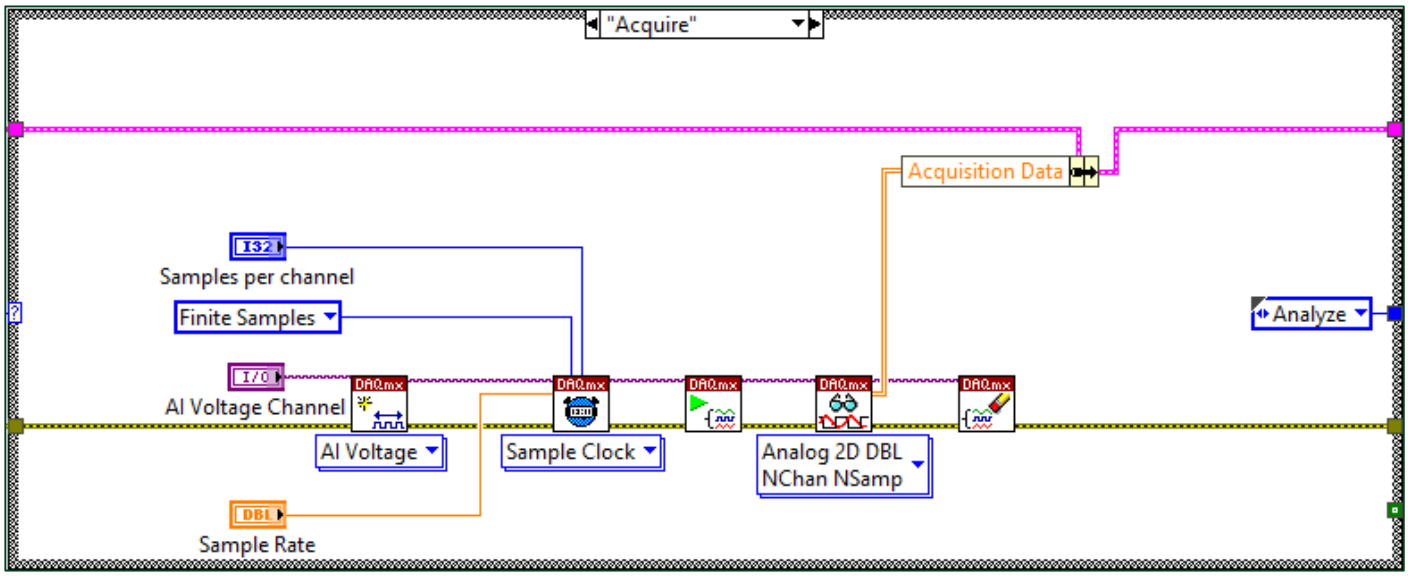





[2] "Clear": Value Change
TF
Clear
Time
Clear


[3] "Exit": Value Change
TF
Exit
OldVal
Exit

7. Define what happens when you click the **Acquire** button.
   - As seen in the state transition diagram, when you click the **Acquire** button, the next iteration of the While Loop will execute the Acquire case.
   - Go to the Acquire case of the Case structure, which corresponds to the Acquire state in the state transition diagram.
   - Based on the state transition diagram, which state should the state machine go to after the Acquire state? _____
   - Modify the Acquire case to go to the Analyze state next, as shown in the following figure.

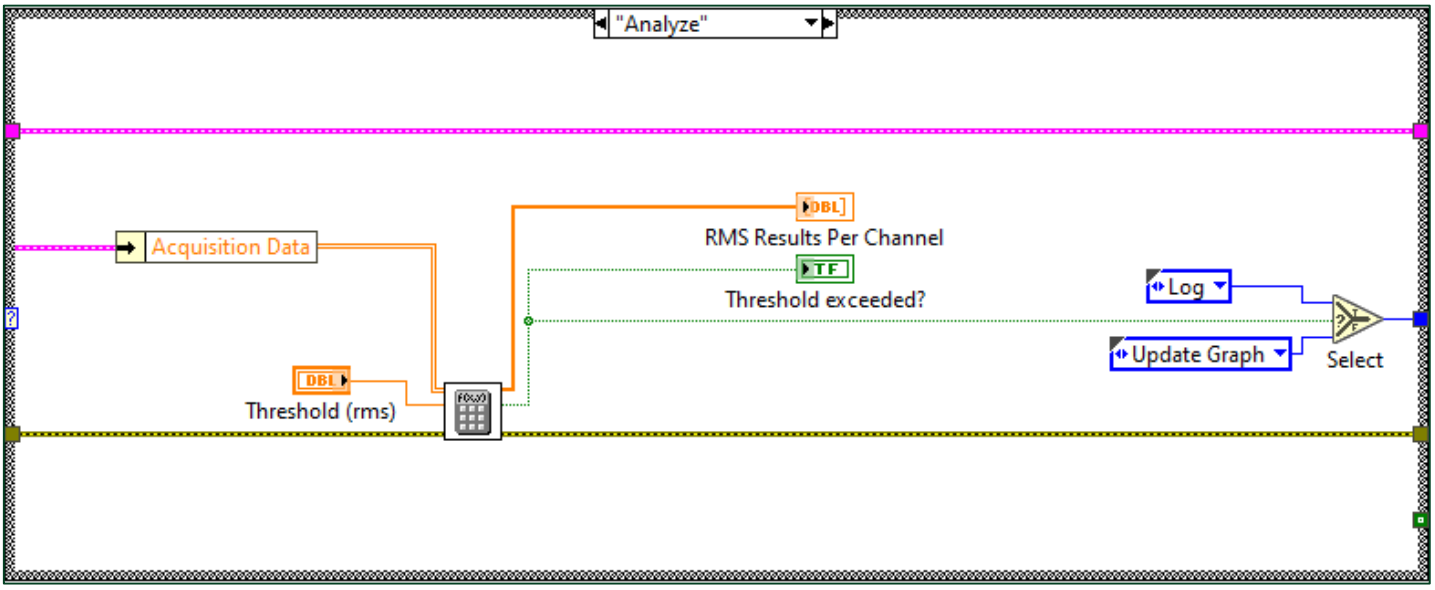


   - Which state should the state machine go to after the Analyze state?

     _____

     _____

- Modify the Analyze case to go to either the Log or Update Graph state next, as shown in the following figure.

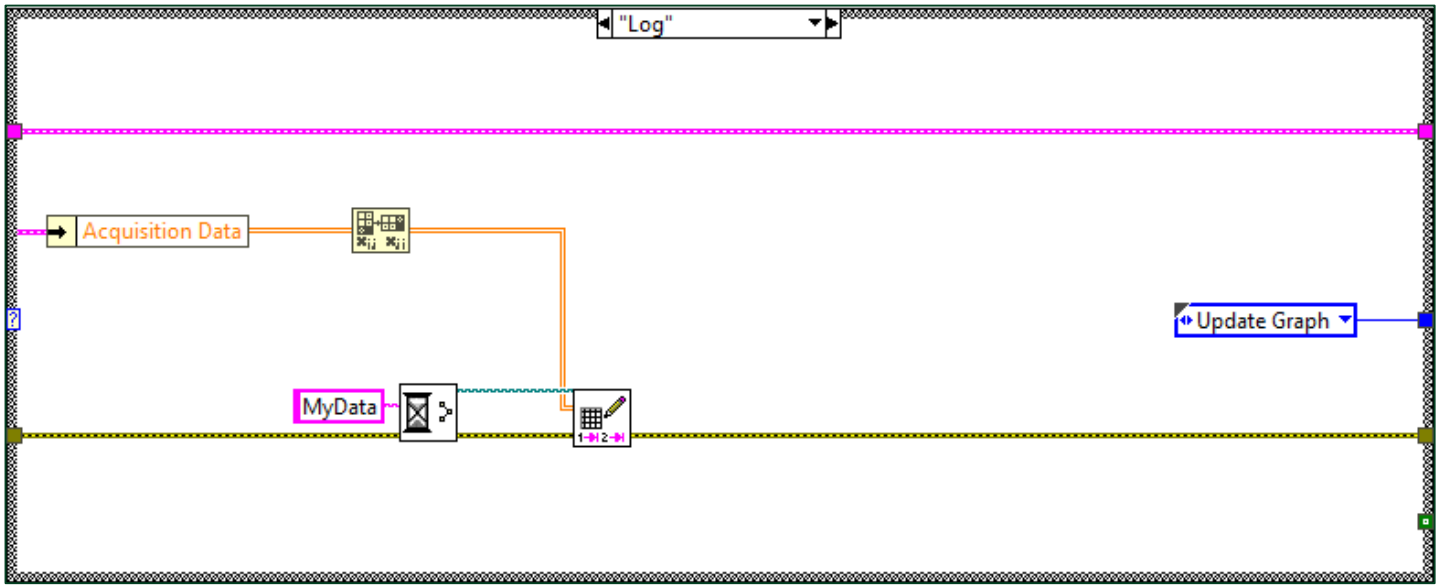


- Which state should the state machine go to after the Log state?

  _____

  _____

- Modify the Log case to go to the Update Graph state next, as shown in the following figure.

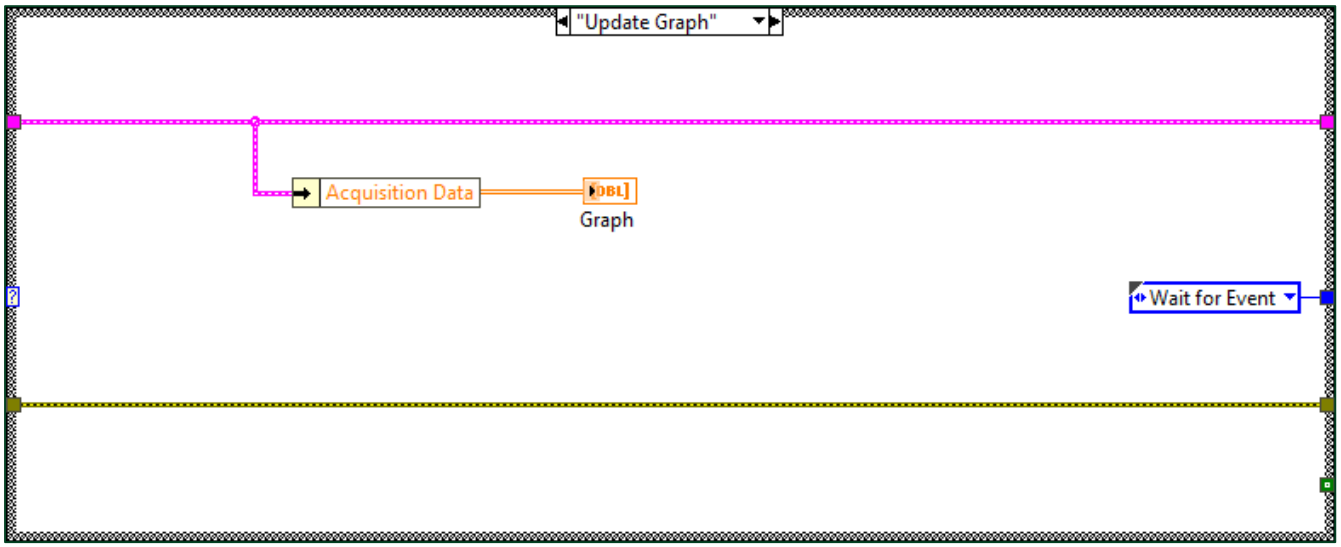


- Which state should the state machine go to after the Update Graph state?

  _____

  _____

- Modify the Update Graph case to go back to the Wait for Events state next, as shown in the following figure.

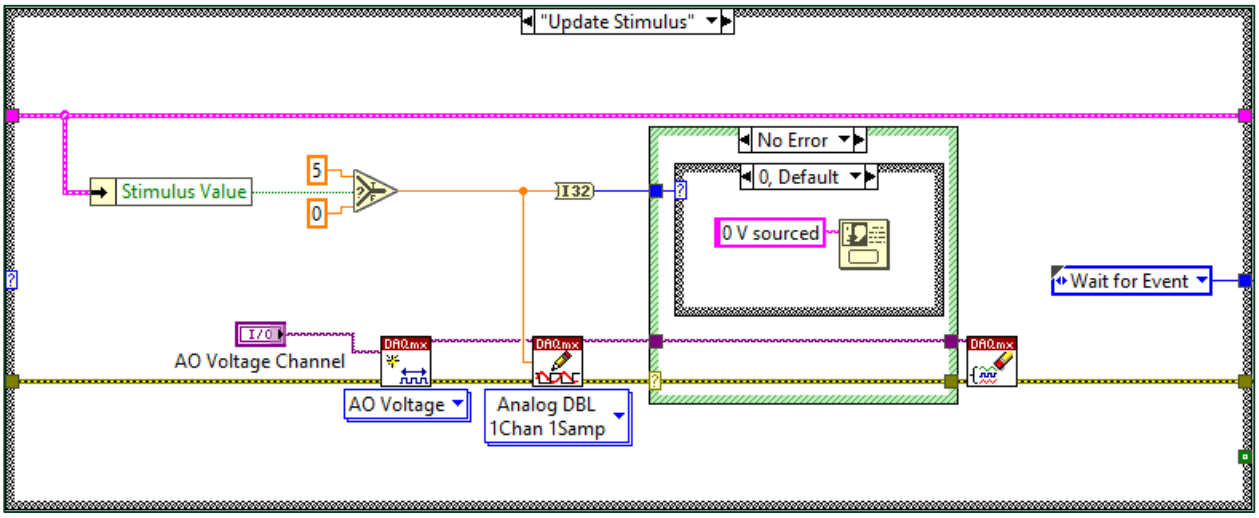


8. Define what happens when you click the **Generate Stimulus** button.
   - Examine the Wait for Event state.
     - Select the **Wait for Event** case of the Case structure, and then select the "**Generate Stimulus"**: Value Change event case of the Event structure.
     - Notice that the enum tells the state machine to execute the Update Stimulus case next.
   - Examine the Update Stimulus state.
     - Select the **Update Stimulus case** of the Case structure.
   - Based on the state transition diagram, which state should the state machine go to after the Update Stimulus state?

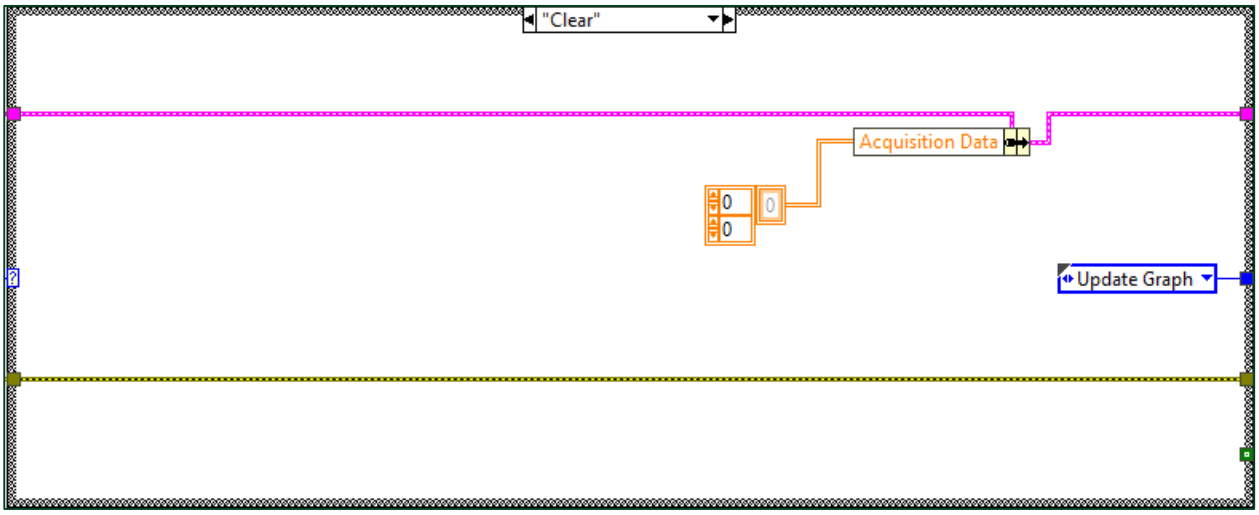
     _____

     _____

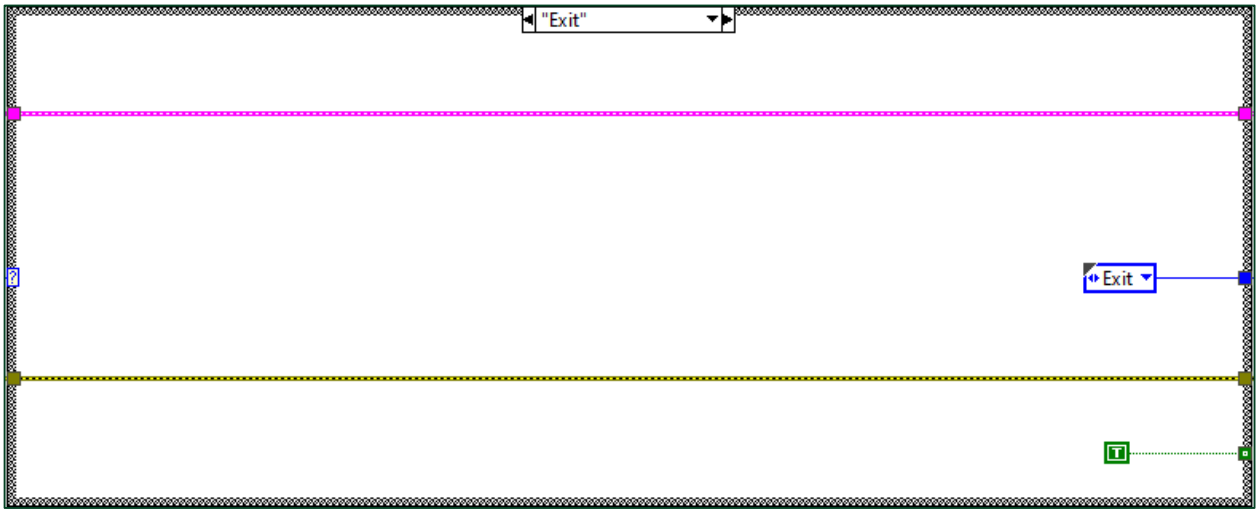
- Modify the Update Stimulus case to go back to the Wait for Events state next, as shown in the following figure.



9. Define what happens when you click the **Clear** button.
   - Examine the Wait for Event state.
     - Select the **Wait** for Event case of the Case structure, and then select the **"Clear"**: Value Change event case of the Event structure.
     - Notice that the enum tells the state machine to execute the Clear case next.
   - Examine the Clear state.
     - Select the **Clear** case of the Case structure.
     - Based on the state transition diagram, which state should the state machine go to after the Clear state?
   - Modify the Clear case to go to the Update Graph state next, as shown in the following figure.



10. Define what happens when you click the **Exit** button.
    - Examine the Wait for Event state.
        - Select the **Wait** for Event case of the Case structure, and then select the **“Exit”**: Value Change event case of the Event structure.
        - Notice that the enum tells the state machine to execute the Exit case next.
    - Examine the Exit state.
        - Select the **Exit** case of the Case structure.
        - Notice that this case outputs a True constant that stops the While Loop and exits the application.
        - Even though the application will exit after the Exit case, you still need to wire a value into the enum output tunnel of the Case structure to ensure that the output tunnel value is defined in all cases.
        - Modify the Exit case to wire an Exit enum value to the enum output tunnel, as shown in the following figure, even though the state machine will exit immediately after the Exit case.



11. Examine how the state machine allows multiple states to access the **State Machine Data** cluster data.
    - Notice that the Acquire, Analyze, Log, Clear, and Update Graph states all write to or read from the Acquisition Data element of the **State Machine Data** cluster.
    - Notice how the Wait for Event» ”Generate Stimulus”: Value Change and Update Stimulus states both access the Stimulus Value element of the **State Machine Data** cluster.
12. Save the VI.

Test

1. Run the VI.
2. Set the controls to the following values.

AI Voltage Channel	PCI-6221/ai0, PCI-6221/ai2:3
Sample Rate (Hz)	1000
Samples per channel	100
Threshold (rms)	1
AO Voltage Channel	PCI-6221/ao1

3. Click the **Generate Stimulus** button to set it to True.
4. If you have a real DAQ device, the analog output channel should now be outputting 5 V.
5. Click the **Acquire** button.
   - Verify that the graph indicator displays the acquired data.
   - If necessary, adjust the **Threshold (rms)** value to a low enough number, so that the acquired data is above the threshold. Click **Acquire** again. This will cause the VI to log the data to a file.
   - In Windows Explorer, navigate to the `C:\Exercises>\LabVIEW Core 2\ Event-Driven State Machine` directory. Notice that a log file is created each time the acquisition data exceeds the threshold.
6. Click **Clear** button.
   - Verify that the VI clears the graph.
7. Click the **Exit** button.
   - Verify that the VI exits.

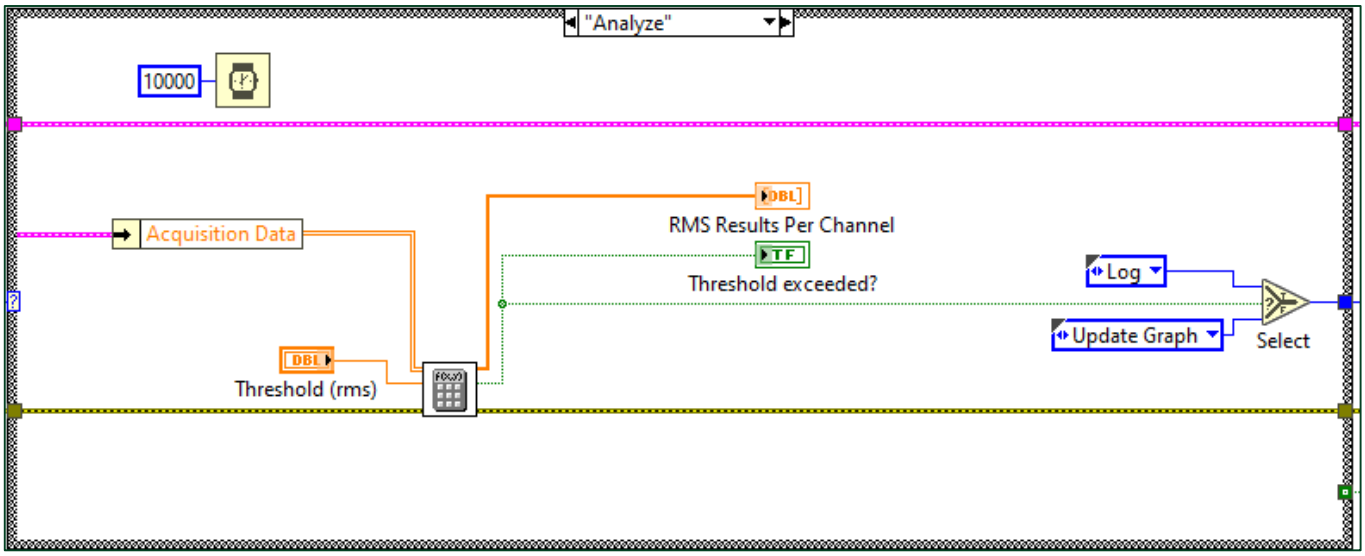
Explore Limits/Caveats of Event-Driven State Machine Design Pattern

The Event-Driven State Machine design pattern is good for applications where users click **controls** on the front panel to trigger corresponding code to execute and the code logic can be described by a state diagram. This design works best when the corresponding code duration is short.

If the code duration is long, such as 10 seconds, that means the user interface (front panel) will appear unresponsive to the user for those 10 seconds.

Explore what happens when corresponding code duration is long.

1. Modify the Analyze case code execution duration to be 10 seconds.
   - Add a Wait (millisecond) node in the Analyze case. Right-click the **input** and select **Create» Constant**. Set the constant to 10000.



2. Examine the behavior of the VI.
   - Run the VI.
   - Click the **Acquire** button. This causes the state machine to go into the Acquire case and then the Analyze case, which will execute for 10 seconds.
   - Click the **Clear** button. Usually, the Clear button would show a latching mechanical action (ON, and then reset back to OFF). Notice that the Clear button appears unresponsive because it appears like it is still pressed (ON). This happens because the code is still in the Analyze case. When the code eventually goes back to the Wait for Event state, it will read the Clear button click in the "**Clear": Value Change** event case, and the Clear button switches back off. The code to clear the graph will then execute.
   - Click **Exit** to stop the **VI**.

3. Delete the Wait node from the block diagram to return to the original behavior of the VI. There are two techniques to solve this problem and make the user interface still appear responsive while code is taking a longer time to execute. You will learn about these techniques later.
    - Method 1: If code is taking a longer time to execute, disable controls to let the user know that the use of those controls id impossible. When the code is done executing and the application can once again process events, then re-enable those controls.
    - Method 2: Use a design pattern based on the Producer/Consumer (Events) design pattern. This design pattern uses two loops:
        - Loop 1: Handle events to keep the user interface responsive.
        - Loop 2: Execute the code that corresponds to the events.

Explore an Event-Driven State Machine Template

LabVIEW includes an Event-Driven State Machine template. Open the template by following these steps:

1. Open LabVIEW.
2. From the Getting Started window click the **Create Project** button.
3. Select **Simple State Machine** from the list and click **Finish** to create a project based on an event-driven state machine.
4. Explore the project.

Your Turn

Try making the following improvements to the application that you created in this exercise.

Challenge #1

When developing an application that outputs signals, you should ensure that the application sets the outputs to a safe value when the application exits.

Modify this VI, so that the Exit case sets the analog output channel (PCI-6221/ao1) to 0 V.

Challenge #2

Modify this VI, so that every time the VI starts executing, the VI initializes the Generate Stimulus control to False.

Refer to the *Challenge Hints section* at the end of this exercise if you need a hint.

On the Job

Do any of your applications have a need for an event-driven state machine? If so, draw the state diagram for the application below.

Challenge Hints

Challenge #1

Method 1 - Shift Register: Add an AO Voltage Channel item to the State Machine Data shift register, so you can access the most current AO Voltage Channel in multiple event cases (Generate Stimulus, Exit).

Method 2 - Local Variable: Use a local variable in the Exit case to access the current value of the AO Voltage Channel control. Using duplicate terminals during initialization or shutdown generally does not result in race conditions if you properly sequence application.

Challenge #2

Use a local variable in the Initialize case to initialize the value of the Generate Stimulus control.

Refer to the `C:\Solutions\LabVIEW Core 2\Exercise 2-2 (Challenge Solution)` for the solution code.

Answers

Question 1 - Answer: The first case this VI executes is the Initialize case. Notice that the enum shift register is initialized with Initialize.

Question 2 - Answer: The next case the While Loop executes is the **Wait for Event** case.

End of Exercise 2-2