Create an Embedded State Machine

Project

In this tutorial, create a real, working program to help you learn how to develop programs for your own applications in the future. This tutorial guides you through writing a program that illuminates an LED on an NI CompactRIO controller when a switch is turned on by programming the FPGA and the real-time processor. Before starting this project, you need to complete the Embedded Programming in LabVIEW module.

Project Setup

1. At your local electronics store, purchase a single-pole double-throw switch.
2. Wire the COM pin of the switch to DIO0 of the digital module.
3. Wire one side of the switch to AO0, which is used to provide the switch with more than 2 V.

   The specifications of the NI 9401 digital module explain that the module reads anything over 2 V as high, so when the switch is positioned so that DIO0 is connected to AO0, the digital line reads high.
4. Wire the other side of the switch to ground, which is used to provide the switch with 0 V.

   The specifications of the NI 9401 digital module explain that the module reads anything under 0.8 V as low, so when the switch is positioned so that DIO0 is connected to ground, the digital line reads low.

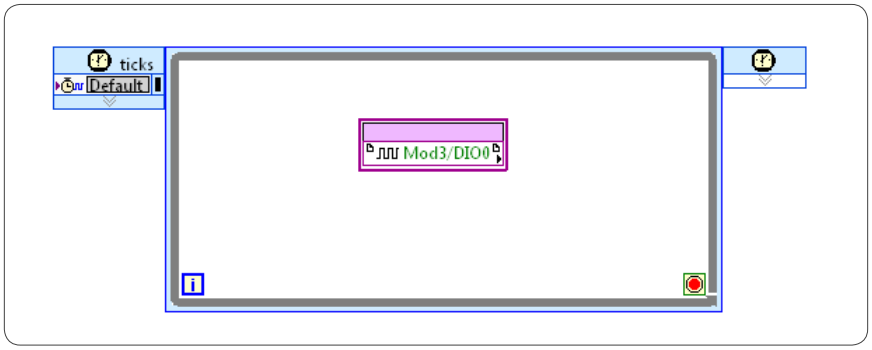
Setup summary: When the position of the switch is changed, the digital module reads either high or low depending on whether the switch is connected to the analog output module or to ground.

Programming

To begin this project, create a new LabVIEW project and add your reconfigurable I/O (RIO) target in FPGA Interface Mode. Embedded Programming Tutorial.

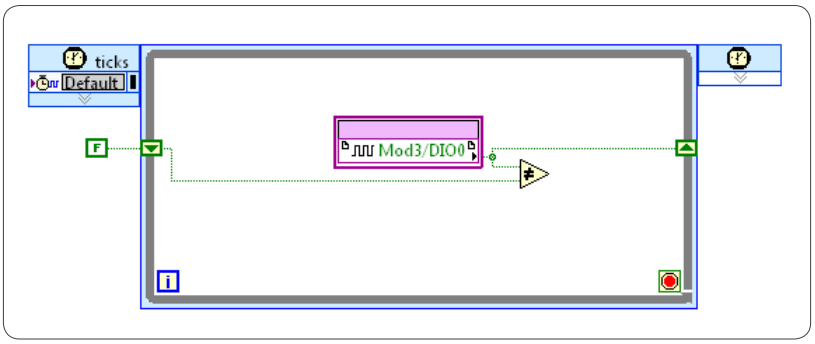
I/O Programming on the FPGA

1. Create a new VI under the FPGA target in the LabVIEW project by right-clicking the FPGA target and selecting **New»VI**.
2. Save the VI as **"State Machine FPGA."**
3. Place an FPGA I/O Node onto the block diagram.
4. Click on the FPGA I/O Node and choose DIO0.
5. Place the FPGA I/O Node in a single-cycle Timed Loop so that it runs continuously.

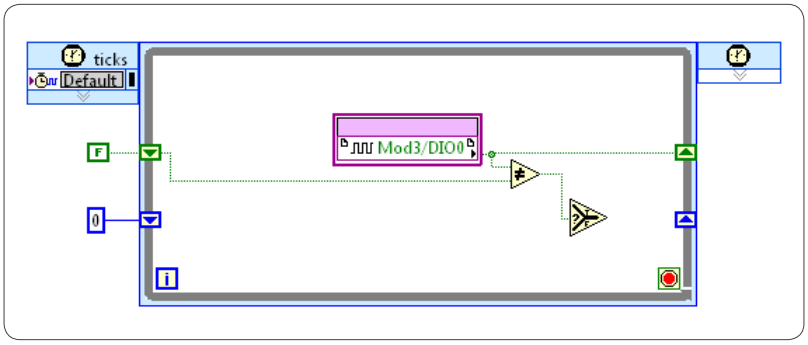


Remember that the goal of this program is to illuminate the controller's built-in LED every time the position of the switch is changed. One way to accomplish this is to simply monitor the state of the DIO0 and if the state of DIO0 changes, the LED is turned on or off. This logic works in theory, but you need to consider the special nature of FPGAs when thinking about how to implement ideas. Remember that the FPGA by default runs at 40 MHz. A loop on the real-time side where this information is processed cannot possibly run that fast, so some changes may not be detected. A possible solution is to increment a counter on the FPGA every time the switch position changes and then read this count on the real-time side. An increase in the count indicates that the switch position has been changed.

6. Place down a not equal function on the block diagram.
7. Wire the output of the FPGA I/O Node to one terminal of the not equal function.
8. Wire the output of the FPGA I/O Node to the loop border and replace the terminal with a shift register.
9. Wire from the left shift register to the other terminal of the not equal function. This allows the current value to be compared to the previous value.
10. Initialize the shift register as **"False."**

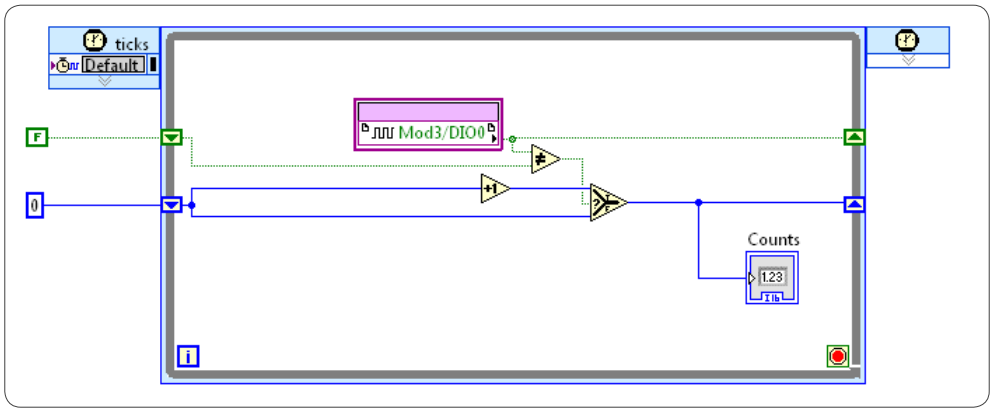


11. Place a Select function on the block diagram.
12. Wire the output of the not equal function to the selector terminal.
13. It is necessary to remember the value of the counter for each loop iteration so that 1 can be added to the most recent value. To implement this, place another shift register on the loop border.
14. Initialize the new shift register at zero to ensure that it starts with a fresh count each time the FPGA is powered with this code deployed to it.



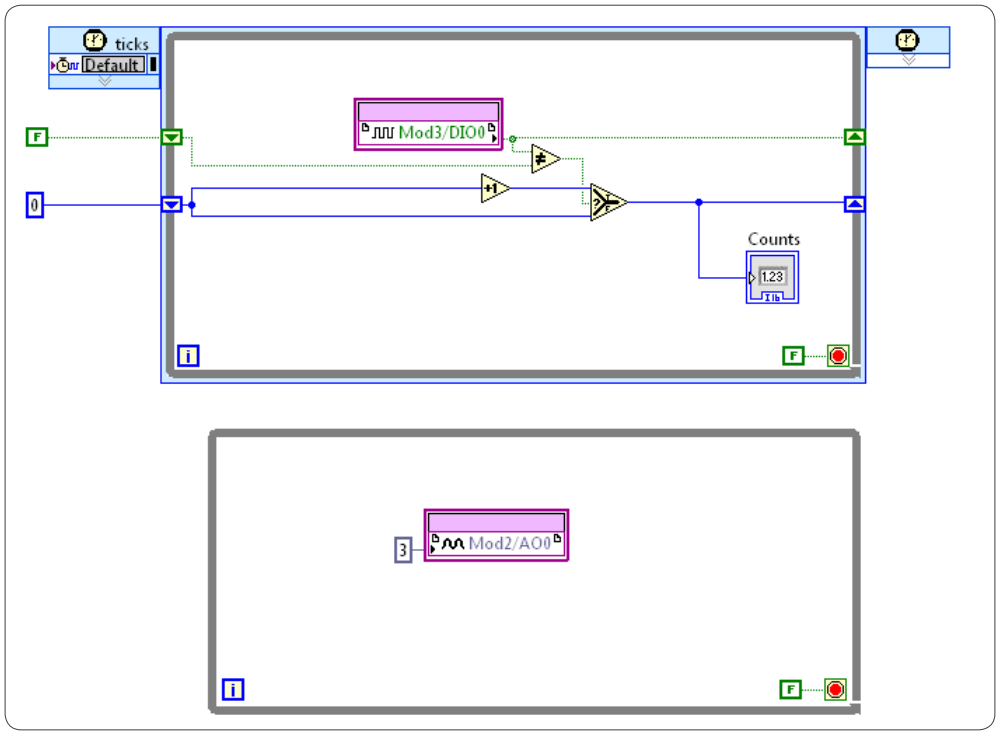
At this point, you can implement some logic based on the value of the count.

15. Place an Increment on the block diagram.
16. Wire the output of the left shift register to the input of the Increment.
17. If the state of the digital line has changed, meaning the Select is true, you want to increment the count, so wire the output of the Increment to the True terminal of the Select Function.
18. If the state of the digital input has not changed, meaning the Select is false, you do not need to increment the count, so wire the output of the left shift register to the false terminal of the select function.
19. Wire the output of the Select Function to the right shift register.
20. Display the current count with an indicator called **"Counts."**



The final step is to write a voltage to analog output 0. Remember that AO0 is used to supply the switch with a voltage value that the digital module registers as high.

21. Place an FPGA I/O Node on the block diagram outside the single-cycle Timed Loop.
22. Click on the FPGA I/O Node and choose AO0.
23. Wire a constant of 3 to the input of the FPGA I/O Node to ensure a value over 2 V is provided to the switch so the digital module reads high.
24. Place the analog code in a While Loop.
25. Wire the constants to the Loop Condition terminals of both loops.

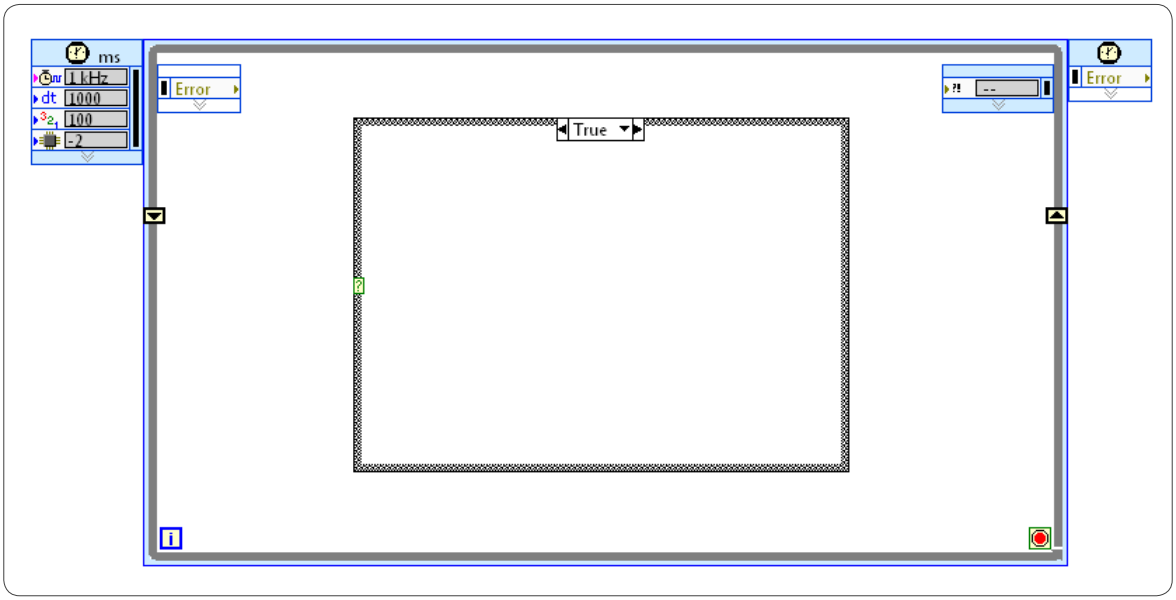


26. Compile the FPGA code. For Steps to do this, see the embedded programming tutorial.

Programming the Processor

Now that the functionality has been defined for the FPGA, switch to the real-time side and use the functionality from the FPGA to do further programming.

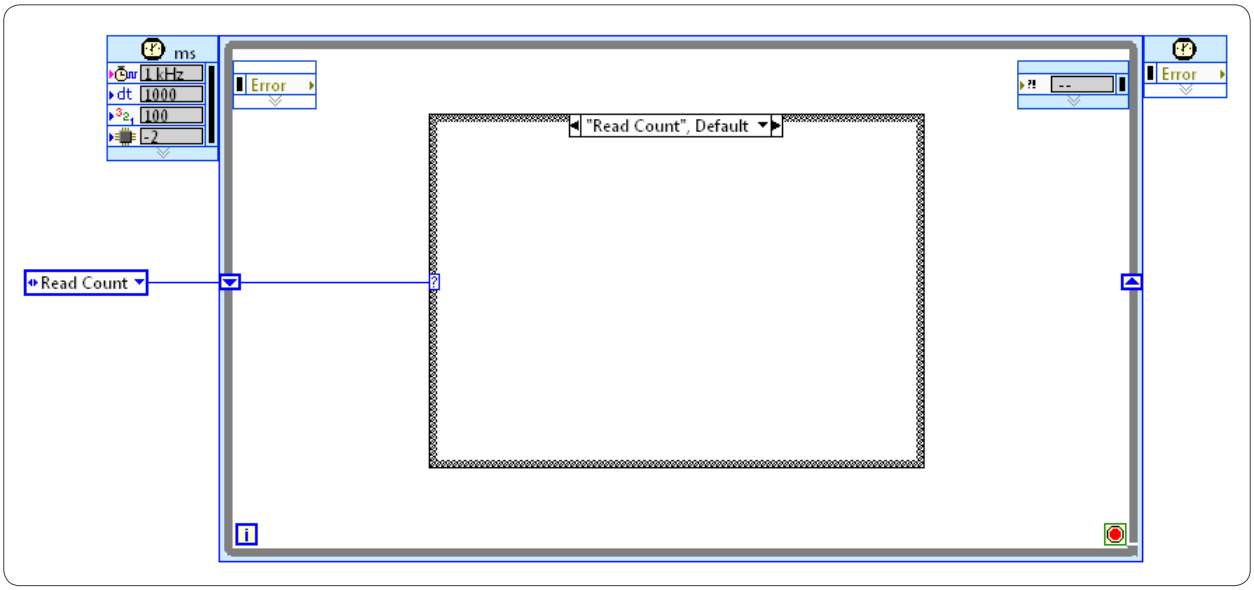
1. Create a new VI to run on the real-time processor by right-clicking on the CompactRIO controller and selecting **New»VI**.
2. Save this VI as **"State Machine RT."**
   Remember that the goal of this project is to change the state of an LED built into the CompactRIO controller every time the position of a switch is changed. You have a few ways to implement this. One way is using a state machine architecture. State machines are a common architecture in embedded programming. You can access state machine templates in NI LabVIEW by selecting **"Create Project"** on the LabVIEW Getting Started screen. This tutorial walks through each step of building a state machine. State machines fundamentally consist of a While Loop, a Case structure, and a shift register. Because you are programming a processor that runs a real-time OS, use a Timed Loop instead of a While Loop even though both work.
3. Place a Timed Loop, Case structure, and shift register on the block diagram as seen in the image below.



This state machine should have two cases. One is **"Read Count"** and the other is **"Set LED."**

4. Place an enum constant on the block diagram outside the loop and define **"Read Count"** and **"Set LED"** cases.
5. Wire the enum into the input of the left shift register.

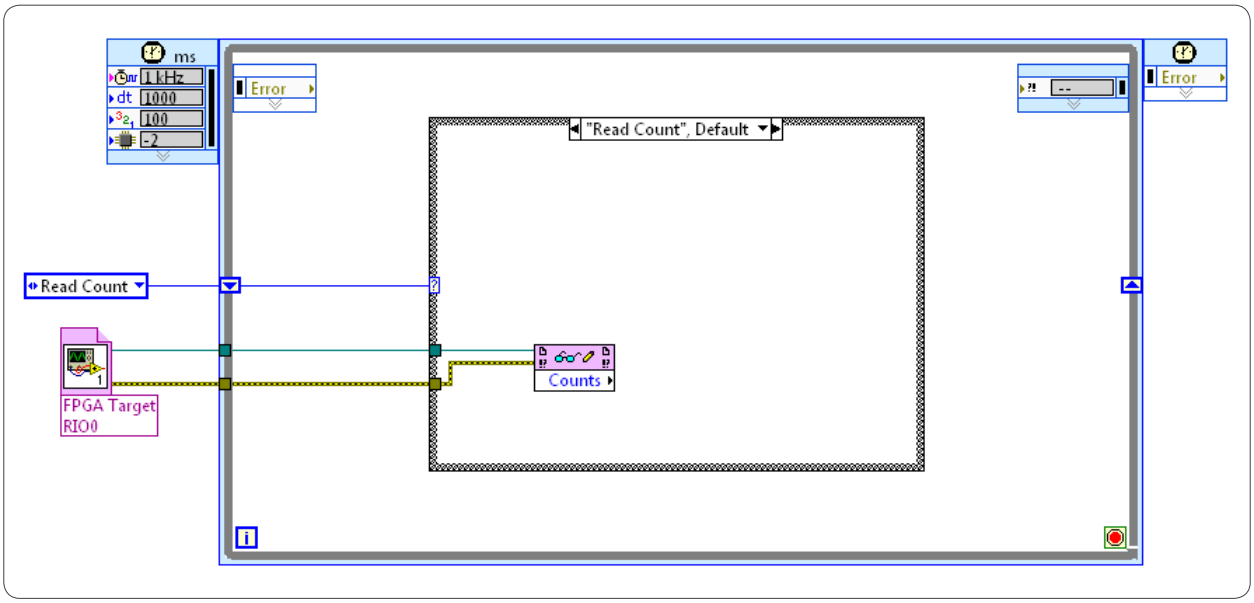
6. Wire the output of the left shift register into the case selector of the Case structure. Remember that state machines normally have more than two cases and you may want to edit these cases as you program. It is a best practice to create a type definition so that all instances of the enum are updated if the cases are edited. Because this tutorial features only two cases, it omits this step, but you should remember it for more complex future projects.



Recall that you programmed the FPGA to increment a counter when the digital input detects a change. Now use that count information to change the state of the LED.

7. Place an Open FPGA VI Reference and double-click on it to point it to the VI programmed for the FPGA.
8. Place an FPGA Read/Write Control on the block diagram.
9. Wire the reference and error wire from the Open FPGA VI Reference to the FPGA Read/Write Control.

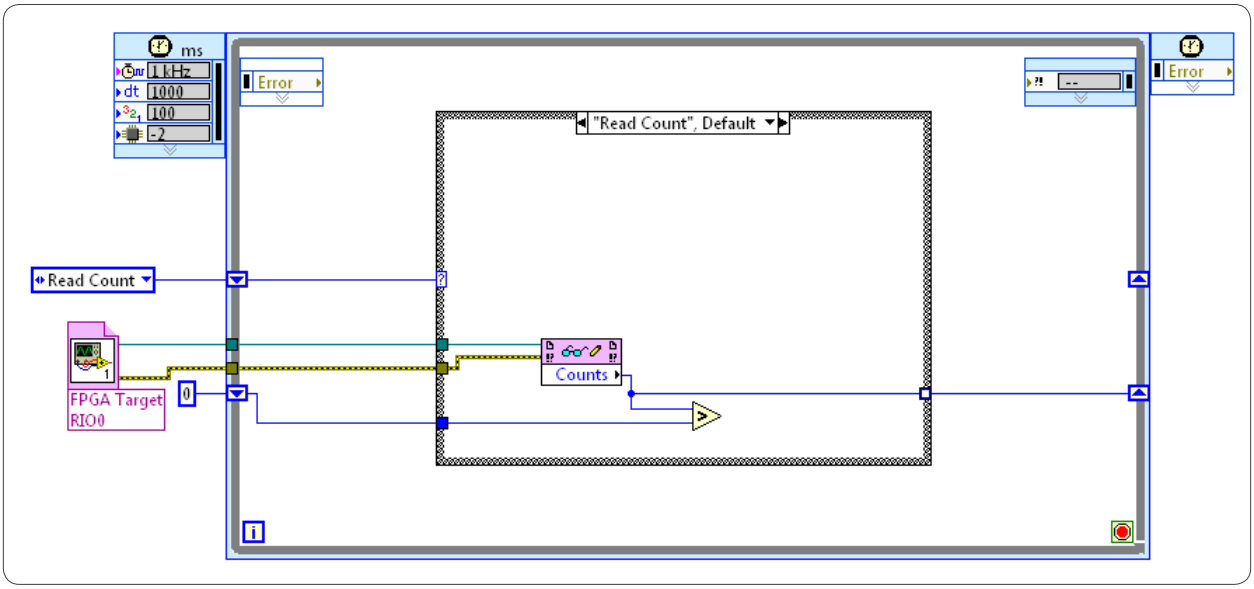
10. Left-click on the FPGA Read/Write Control and choose **"Counts."**



The next step is to write code to determine if the count has increased. To do this, you need to compare the current count to the count from the previous loop iteration.

11. Place a Greater? function on the block diagram inside the Read Count case.
12. Wire the output of Count to the top input of the Greater?.
13. Place an additional shift register on the loop border.
14. Wire the output of Count to the new shift register.
15. Wire the output of the left shift register to the bottom input of the Greater? to compare the current Count value to the previous value.

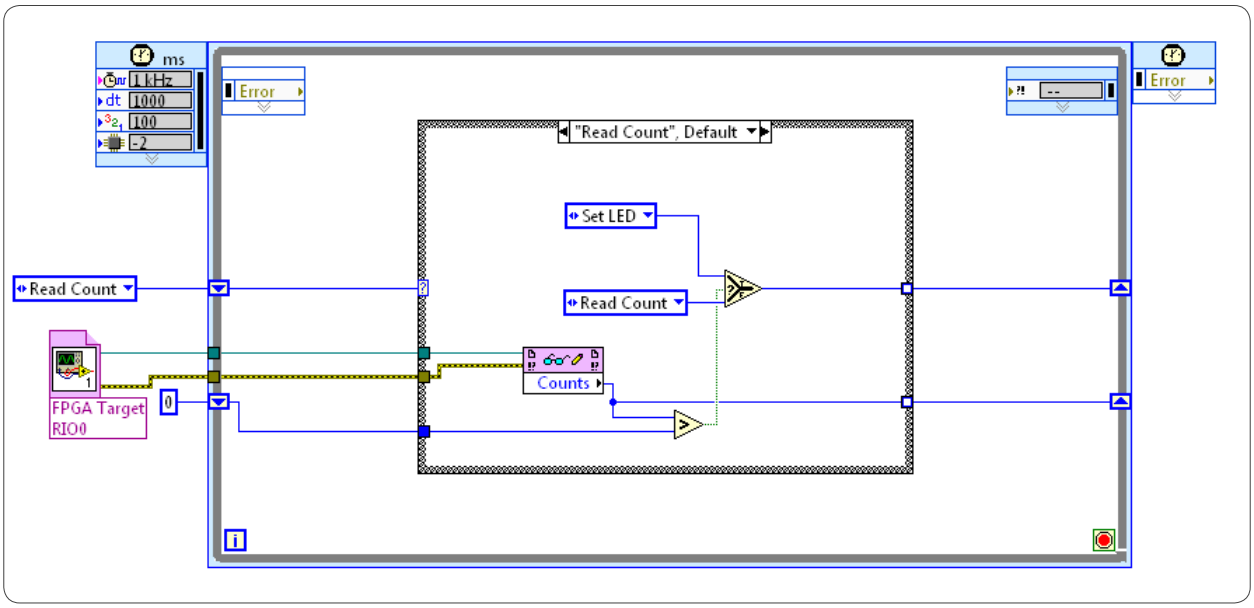
16. Now that you know whether the count increased, this information will help you make decisions in the code that turns the LED on and off.



Now that you know whether the count increased, you can begin to make decisions in the code with that information that turns the LED on and off.

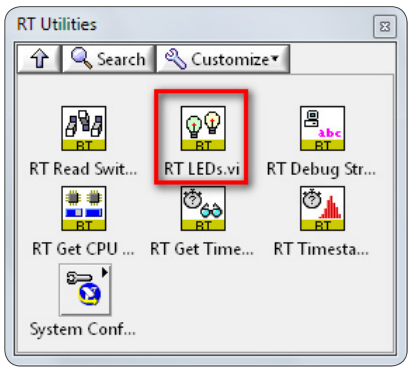
17. Place a Select on the block diagram inside the Read Count case.
18. Wire the output of the Greater? function to the Select.
19. If the Greater? is false, no action should be taken because the switch position was not changed. To ensure no action is taken, wire an instance of the enum constant to the False terminal of the Select and choose to stay in the **"Read Counts"** state.
20. If the Greater? is true, you need to move into the Set LED state. To implement this, wire another instance of the enum constant to the True terminal of the Select and set the enum to **"Set LED."**

21. Wire the output of the Select to the shift register, which is keeping track of the cases specified by the enum.

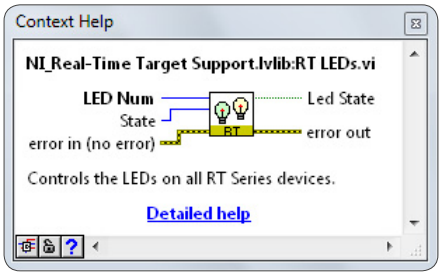


Move to the **"Set LED"** case. Because this tutorial uses an LED built into the CompactRIO controller, you can use a VI that is already built into LabVIEW to program the LED.

22. On the Real-Time palette under Utilities, locate the **RT LEDs.vi**.
23. Place this VI inside the Set LED case.



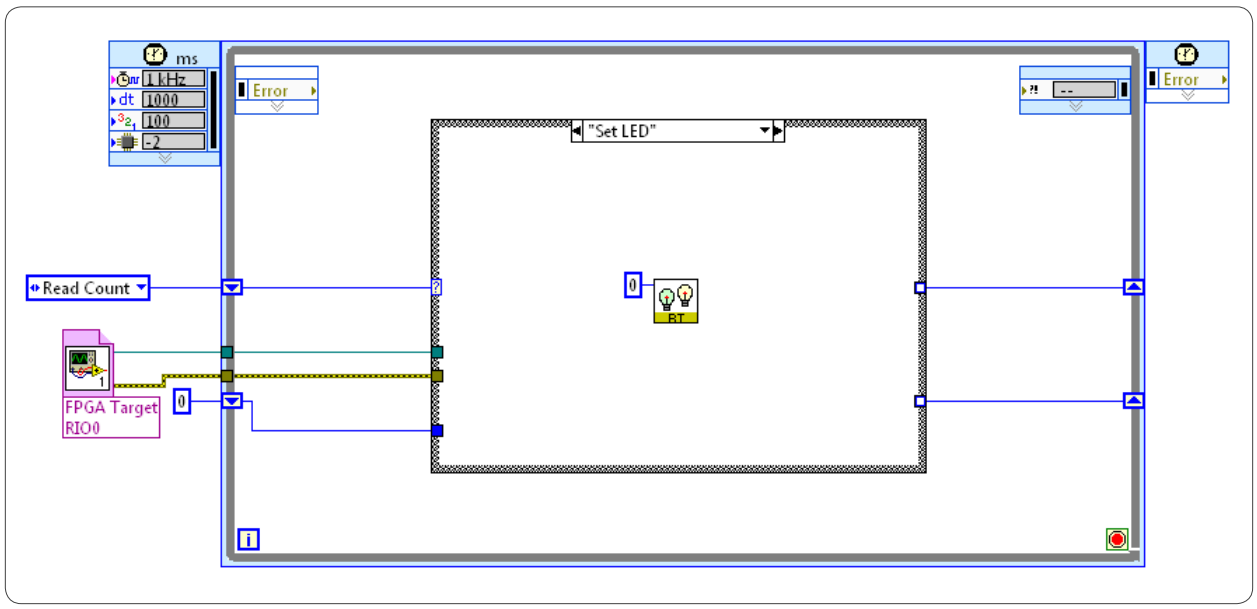
24. Open the LabVIEW Help by using the keyboard shortcut **<Ctrl-H>** and hovering over the **RT LEDs.vi**. In the Context Help window, click on **"Detailed Help"** to learn more about this VI.



25. Notice that the Detailed Help offers information related to the purpose of each terminal of this VI including how to specify which LED to use and how to specify which state the LED is in.

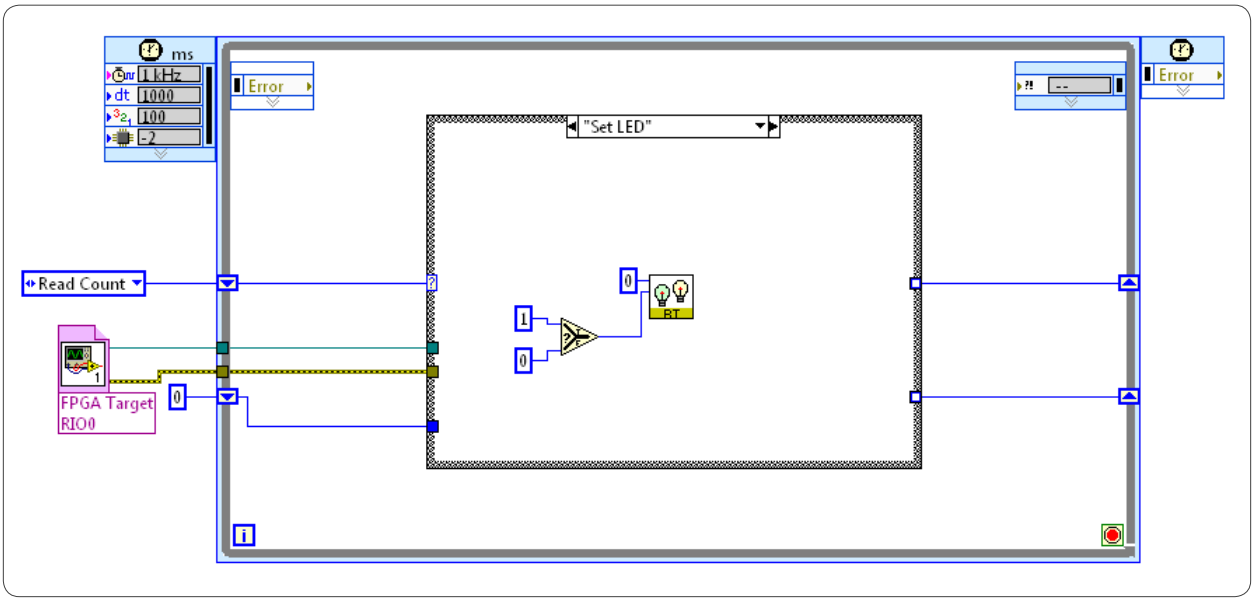
The NI cRIO-9024 embedded real-time controller has one user-defined LED which is LED 0. This can be different on different controllers.

26. Specify 0 as the LED number.



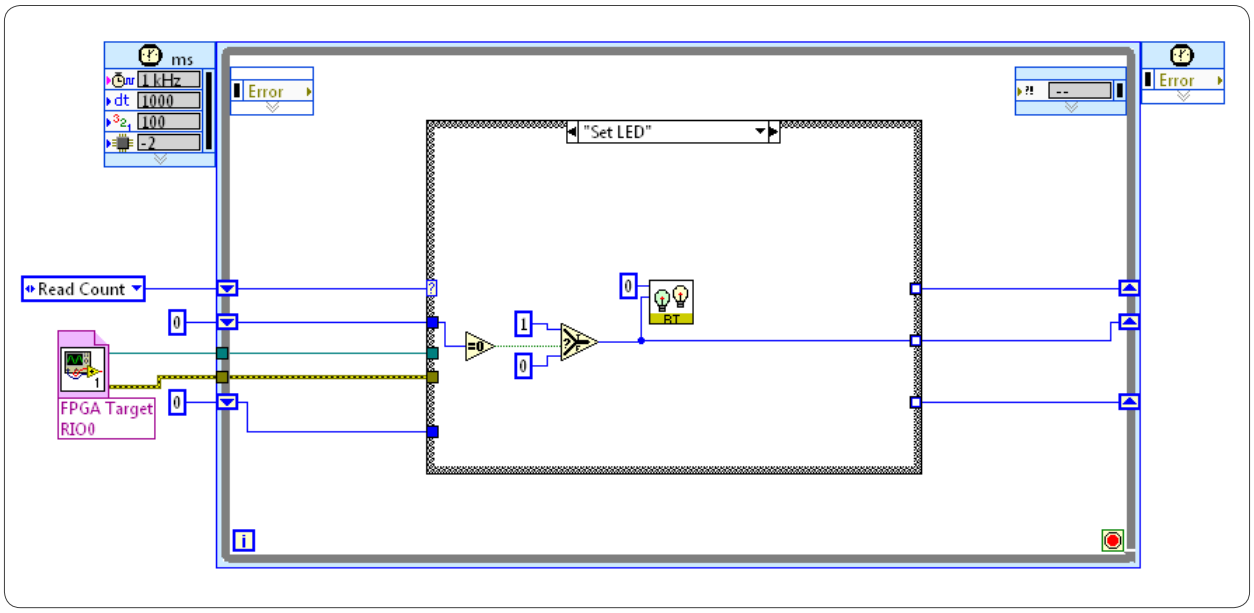
27. Place a Select inside the Set LED case. With this function, you can change the state of the LED depending on if the switch position has changed.
28. Wire a constant of 1 to the True terminal of the Select.

29. Wire a constant of 0 to the False terminal. The LabVIEW Help specifies that 1 turns the LED on and 0 turns it off.
30. Wire the output of the Select function to the State terminal of the RT LEDs VI.



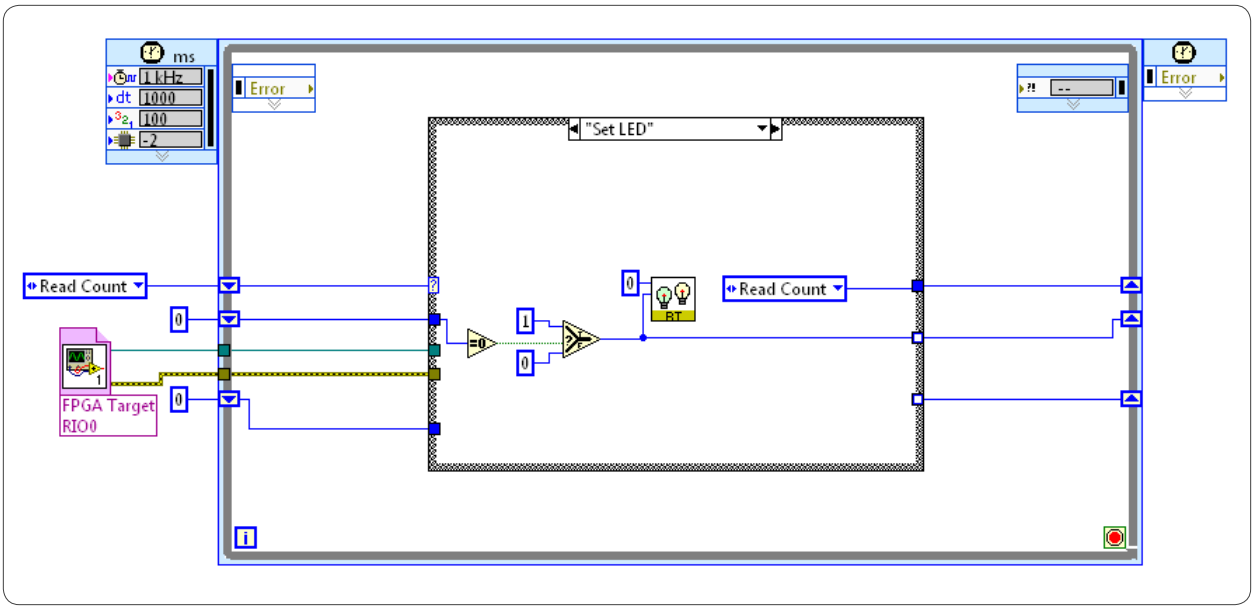
31. Add a shift register to the border of the Timed Loop to compare the current state of the LED to the previous state.
32. Wire the output of the select to the new shift register.
33. Initialize the shift register at 0.

34. Place an Equal to 0? function inside the Set LED case and wire it to the selector terminal of the Select. With this in place, if the last value is equal to zero when the switch position is changed, the Select reads true and turns on the LED by writing the number 1 to the LED state. On the next iteration, 1 is in the shift register that does not equal 0, so the Select registers False and writes 0 to the LED state, which turns off the LED.

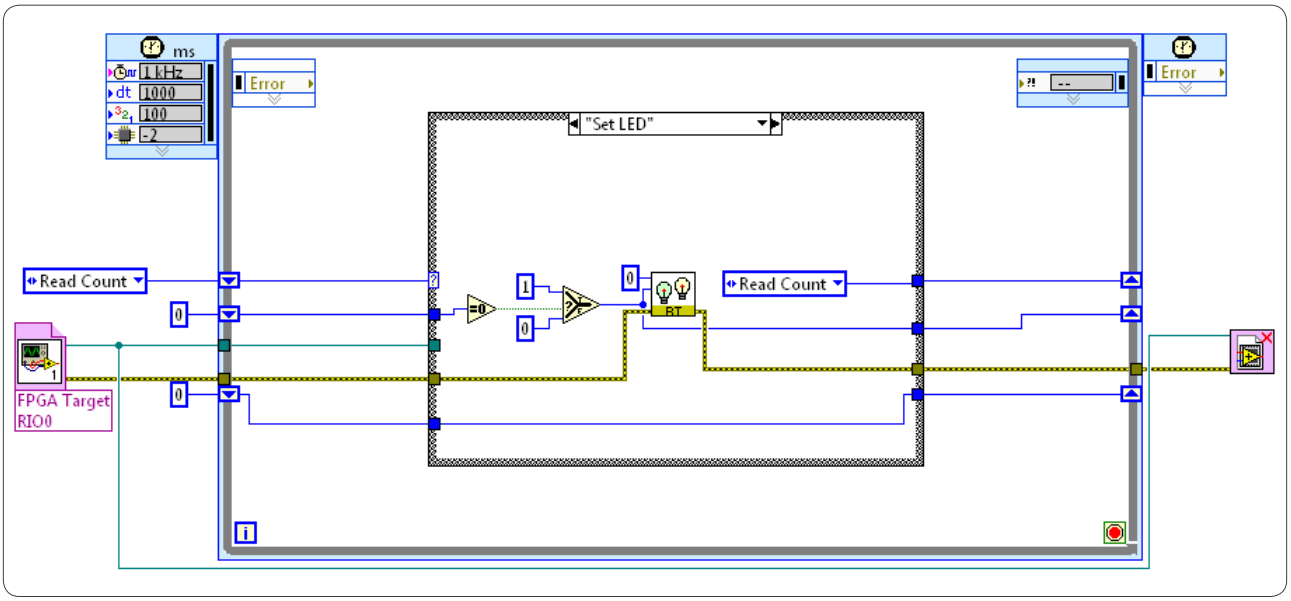


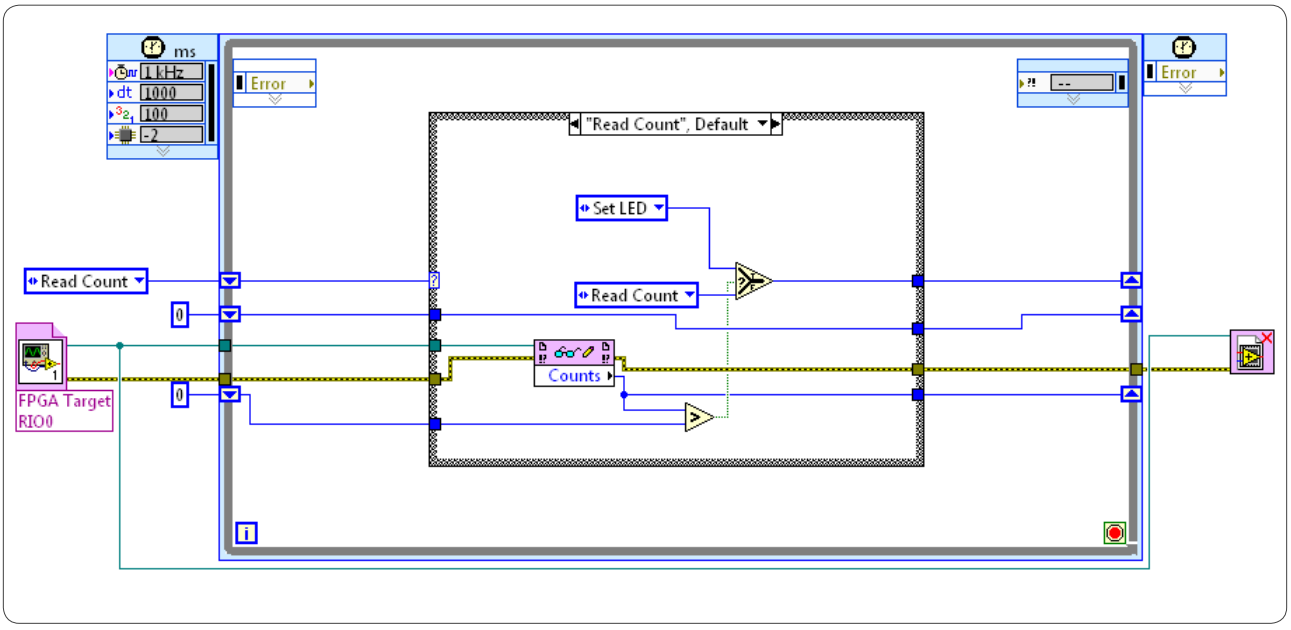
The final step is to program a way to exit the case and return to reading the count after the LED state has been changed.

35. Place an instance of the enum inside the Set LED case and set the enum to **"Read counts."**



36. Wire all error wires and close the FPGA reference.





37. Run the real-time code.

38. Unplug the Ethernet cable to see your code run headlessly.