Tutorial: Timing, Shift Registers, and Case Structures

Publish Date: Jul 02, 2008 | 34 Ratings | **4.65** out of 5

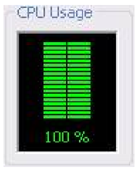
Overview

In this tutorial you will learn to expand the potential of your VIs with timing functions, iterative data transfer, and case selectors. After you complete this tutorial, you will be able to time for and while loops, pass data between iterations of a loop, and programmatically select which sections of your code to run.

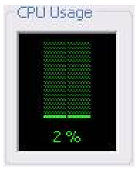
Table of Contents

Loop Timing in LabVIEW

Normally when a loop, such as a while loop, finishes executing one iteration, it immediately begins running the next. It is often beneficial to control how often a loop executes, or its frequency. For example, if you wanted to acquire data in a loop, you would need a method to control the frequency of the data acquisition. Timing a loop also allows the processor time to complete other tasks such as updating and responding to the user interface. In the following figures, the processor usage for a simple VI with a while loop running untimed and timed are shown. Timing a loop can drastically increase performance.



Untimed Loop

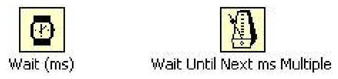


Timed Loop Executing at 1000 Times a Second

Wait Functions

After you create a loop, you can place a wait VI inside of the loop to control how long it waits before performing the next iteration.

There are two basic wait functions in LabVIEW: **Wait (ms)** and **Wait Until Next ms Multiple**.



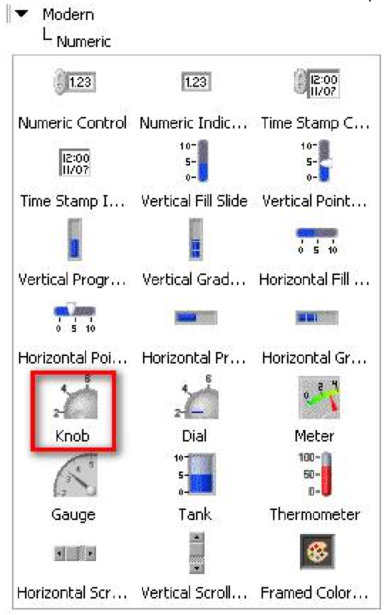
The **Wait (ms)** function forces the loop to wait for a user-specified amount of time, in milliseconds, before running the next iteration.

The **Wait Until Next ms Multiple** function watches the millisecond counter and waits for it to reach a multiple of the user-specified time, in milliseconds, before running the next iteration of the loop. You can use this VI to synchronize different activities. For example, you can configure multiple loops to execute at each multiple of 200 ms.

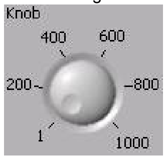
Tutorial: Implementing a Wait Function in a Loop

Consider using one of the wait functions in a while loop. Using the information from previous tutorials, create a blank VI in LabVIEW.

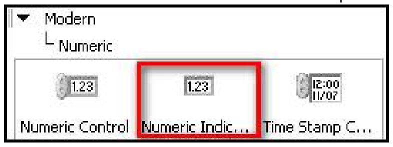
1. Place a knob numeric control on the front panel by right-clicking on the front panel and navigating to **Controls»Modern»Numeric»Knob.**



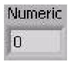
2. Change the knob's limits to 1 and 1000 by double-clicking on the knob's current limits and entering the new values. You will use the knob to control a while loop's wait time.



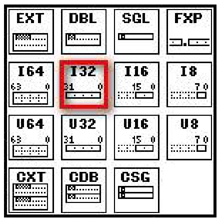
3. Place a numeric indicator on the front panel by right-clicking on the front panel and navigating to **Controls»Modern»Numeric»Numeric Indicator**.



4. This indicator displays the while loop iterations.

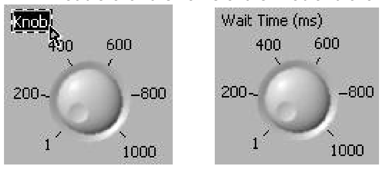


5. Change the Representation of the numeric indicator to I-32 (long integer) by right-clicking on the indicator and selecting Representation. Click on I32.

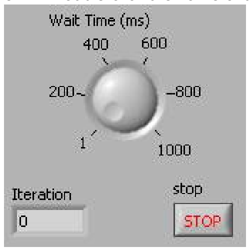


6. Place a Stop Boolean control on the front panel. You can find this at **Controls»Modern»Boolean»Stop Button**. Use this **Stop** button to stop the while loop.

7. Double-click the name of the knob and change its name to "Wait Time (ms)."

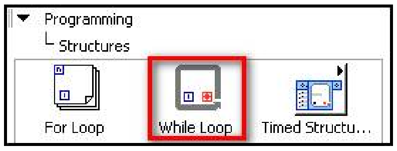


8. Double-click the name of the numeric indicator and change its name to "Iteration."

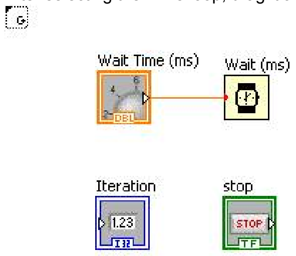


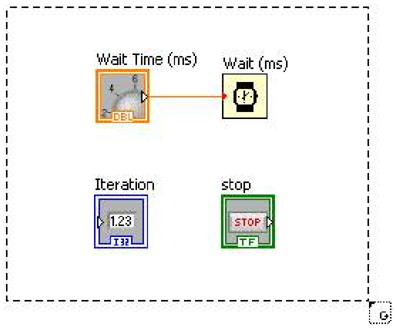
9. View the block diagram by selecting **Window»Show Block Diagram** or pressing **<ctr-E>**.

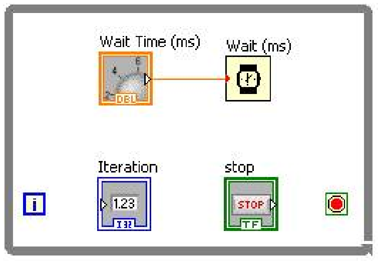
10. On the block diagram, drag a while loop around the front panel controls and indicator. Find the while loop at **Functions»Programming»Structures»While loop**.



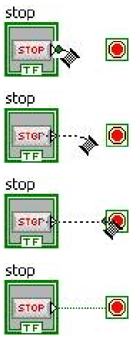
After selecting the while loop, drag it around the three icons. If you miss one icon, just click and drag it into the loop.





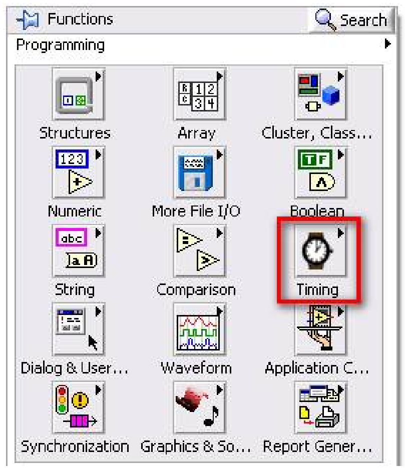


11. Wire the Stop control to the while loop's stop conditional terminal. Click on the right terminal of the **Stop** button's icon, drag to the left input terminal of the stop conditional terminal, and click to complete the wire.

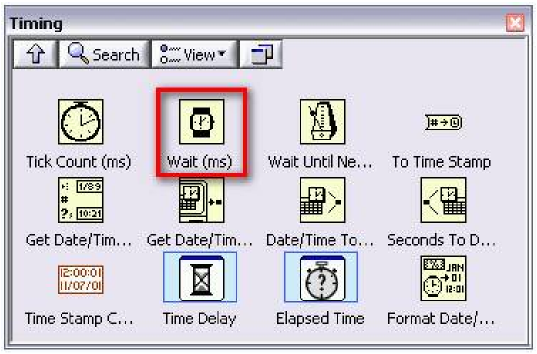


12. Wire the numeric indicator to the while loop iteration terminal.

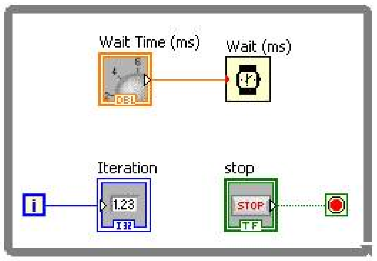
13. Right-click on the block diagram to open the **Functions** palette and then navigate to and open the **Timing** palette, which is located at **Functions»Programming»Timing**.



14. Place the **Wait (ms)** function, located in the **Timing** palette, inside the while loop.

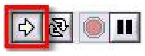


15. Wire the knob control **Wait Time (ms)** to the input of the **Wait (ms)** function. The knob value specifies how long, in milliseconds, the loop waits before running the next iteration.

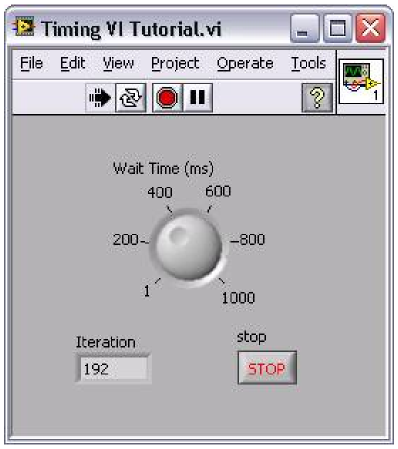


16. View the block diagram by selecting **Window»Show Front Panel** or pressing **<ctr-E>**.

17. Run the VI.



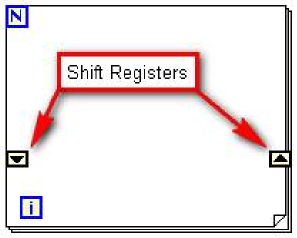
18. As the VI runs, change the knob value by clicking and dragging the knob; note that the speed of the loop, as shown by the iteration indicator, changes accordingly.



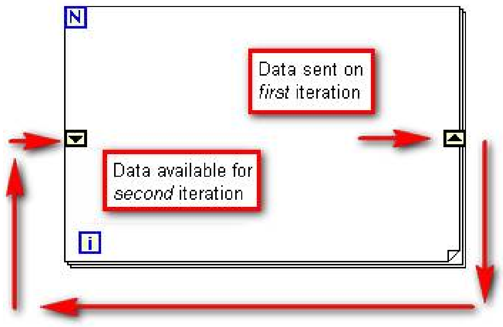
19. Stop the VI using the Stop Boolean control.

Iterative Data Transfer

When programming with loops, sometimes you need to call data from previous iterations of the loop. In LabVIEW, you can use shift registers, which are similar to static variables in text-based programming languages, to pass values from one loop iteration to the next.



Data enters the right shift register and is passed to the left shift register on the next iteration of the loop.


Data sent on *first* iteration
Data available for *second* iteration

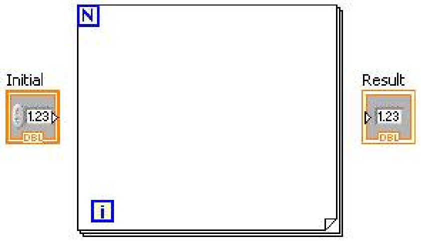
Tutorial: Using Shift Registers

The next section of this tutorial guides you through creating and using shift registers in a simple LabVIEW VI.

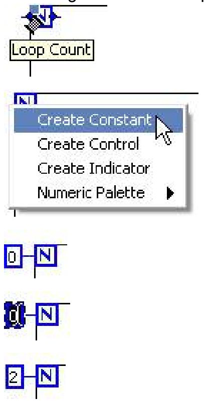
1. Create a new LabVIEW VI by navigating to **File»New VI**.
2. Place a numeric control on the front panel and change its value to 2.
3. Double-click on the control's name and change it to "Initial."

4. Place a numeric indicator on the front panel and name it "Result."

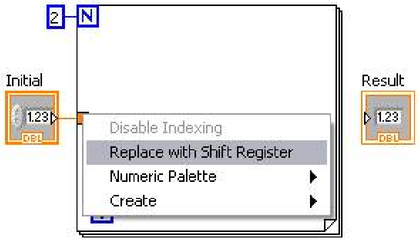
5. View the block diagram by selecting **Window»Show Block Diagram** or pressing **<ctr-E>**.
6. Place a for loop on the block diagram between the numeric control and indicator. The for loop is located at **Functions»Programming»Structures»For Loop**.



7. Right-click on the input of the count terminal of the for loop and select **Create Constant**. Change the value of this constant to 2.

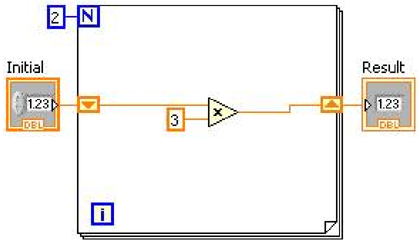


8. Wire the output of the Initial control to the right edge of the for loop to create a tunnel.
9. Right-click on the tunnel that you just created and select **Replace with Shift Register**.



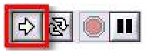
10. Wire the output of the right shift register to the Result indicator.
11. Place a multiply function in the for loop.
12. Place a numeric constant in the for loop, assign it a value of 3, and connect it to one of the input terminals of the multiply function.

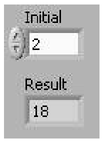
13. Wire the left shift register to the remaining input of the multiply function, and wire the output of the function to the right shift register.



14. View the block diagram by selecting **Window»Show Front Panel** or pressing **<ctr-E>**.

15. Run the VI. The VI changes the value of the Result indicator to 18.





Explanation

Shift registers are integral to this VI. To understand how the VI works, you can step through the code.

Because the for loop's counter terminal is wired to a constant of 2, it runs twice. On the first iteration of the for loop, the value of Initial, 2, is multiplied by 3. The result is 6, and this value is passed to the right shift register. On the second iteration of the for loop, the left shift register receives the value that was sent to the right shift register on the previous iteration, 6. The value of 6 is multiplied by 3, for a result of 18. Because the for loop completed all of its iterations, it stops running and the value of 18 is sent to the Result indicator on the front panel.

The mathematical formula for this simple VI looks like this:

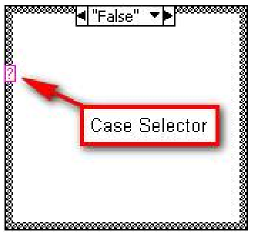
Result = ((Initial * 3) * 3)

If you changed the value of the for loop's count terminal to 4, the mathematical formula looks like this:

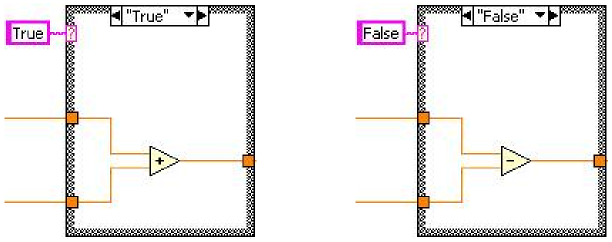
Result = ((((Initial * 3) * 3) * 3) * 3)

Case Structures

When programming in LabVIEW, you may want to choose between multiple sections of code based on an input. Based on the input it receives on the case selector terminal, the case structure chooses which "case," or section of code, to execute. The case selector terminal appears as a small question mark (?) on the left side of the case structure.



If you change the input to the case selector terminal, the section of code that is executed changes. In the figures below, the case structure executes different code for the input strings "True" and "False."



Selector Terminal

The case selector terminal can receive multiple data types. You can use the following data types as inputs to the case selector terminal:

- Integers
- Boolean values
- Strings
- Enumerated type values (Enums)

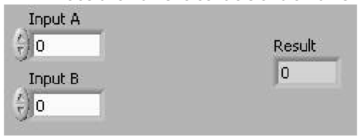
A case structure with a Boolean wired to its selector terminal has a maximum of two cases; all other data types allow two or more cases.

Tutorial: Programming with a Case Structure

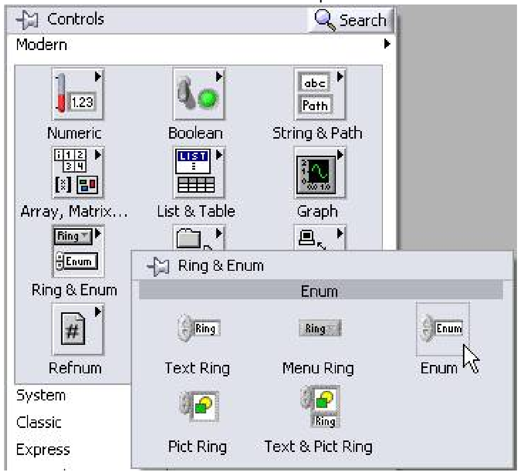
The following tutorial demonstrates how you can use a case structure to choose between multiple sections of code.

1. Create a new LabVIEW VI by navigating to **File»New VI**.

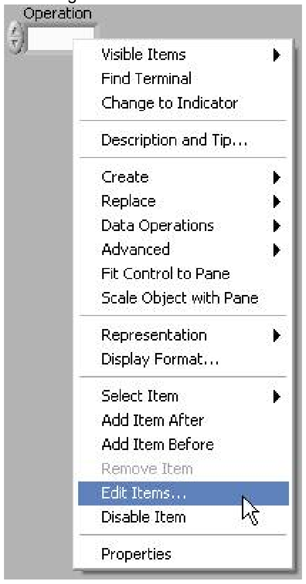
2. Place two numeric controls and a numeric indicator on the front panel. Name the numeric controls "Input A" and "Input B." Name the indicator "Result."

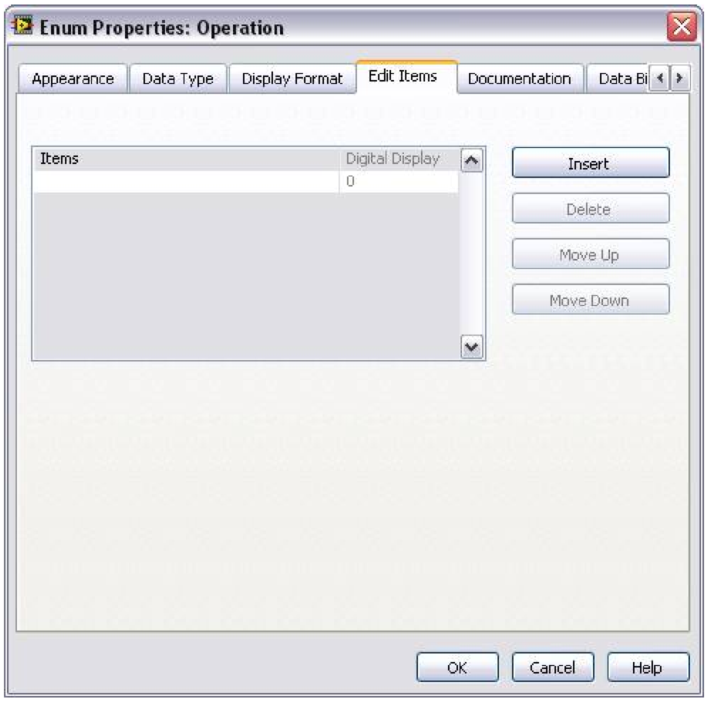


3. Place an Enum control on the front panel. This control is located in **Controls»Modern»Ring & Enum**. Rename the control "Operation."

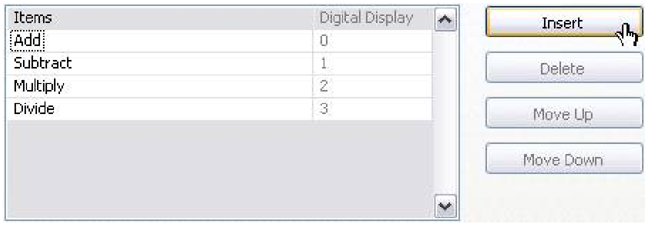


4. Right-click on the Enum and select **Edit Items…** to open the Enum Properties.





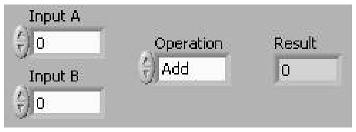
5. Click the **Insert** button and add the following Items: Add, Subtract, Multiply, and Divide.



Items	Digital Display
Add	0
Subtract	1
Multiply	2
Divide	3

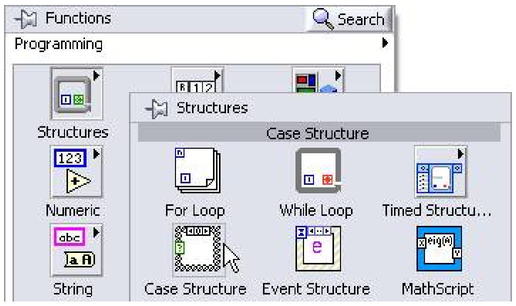
6. Click the **OK** button at the bottom of the Enum Properties box to close it.

Your front panel should look similar to the following figure.

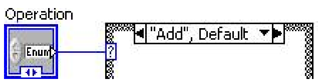


7. View the block diagram by selecting **Window»Show Block Diagram** or pressing **<Ctrl-E>**.

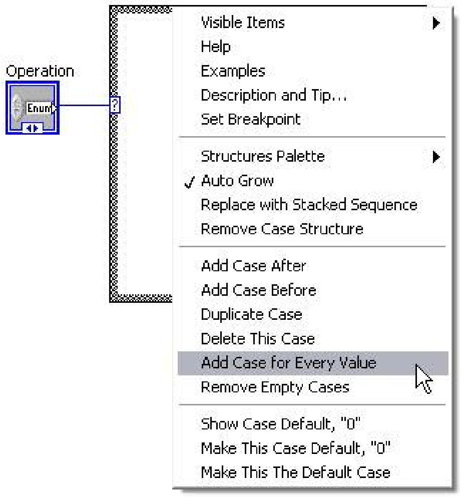
8. Place a case structure on the block diagram, between your controls and indicator. Find the case structure at **Functions»Programming»Structures**.



9. Wire the Operation Enum control to the case structure's selector terminal, located on the left side of the case structure.



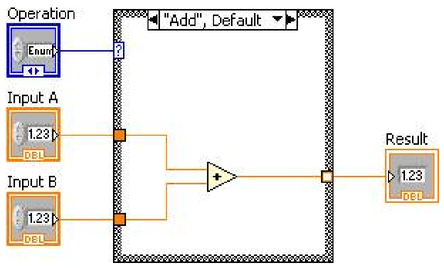
10. Right-click on the border of the case structure and select **Add Case for Every Value**. The case structure now has a case for every value of the Enum that is wired to the case selector terminal. This tutorial demonstrates four cases: Add, Subtract, Multiply, and Divide.



11. Switch to the "Add" case of the case selector terminal by clicking the right or left arrows at the top of the case structure, or by placing your mouse inside the case structure and pressing **<ctr>** while scrolling with the mouse.

12. Place the add function in the Case Statement when the "Add" case is selected.

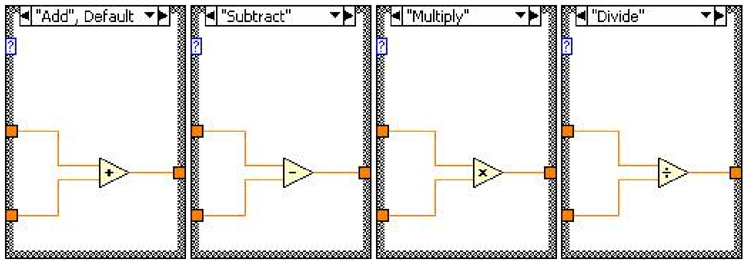
13. Wire the controls Input A and Input B to the inputs of the add function. Wire the output of the add function to the Result indicator. You can wire through the case structure, which acts much like a loop and creates tunnels automatically.



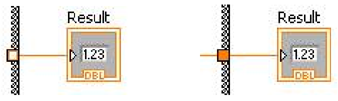
14. Switch to the “Subtract” case of the case selector terminal by clicking the right or left arrows at the top of the case structure, or by placing your mouse inside the case structure and pressing <ctr> while scrolling with the mouse.

15. Add the subtract function to the “Subtract” case. Wire the function to the controls and indicators in the same way you wired the add function.

16. Add the multiply function to the “Multiply” case and the divide function to the “Divide” case. Wire these functions to the controls and indicators in the same way you wired the add and subtract functions. Your case structure should now contain four cases similar to the following:



17. When you have successfully wired all the outputs, the tunnel to the Result indicator changes from a hollow square to a filled square, as shown in the following figures.

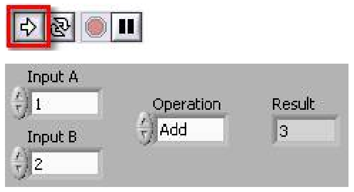


18. View the block diagram by selecting **Window»Show Front Panel** or pressing **<Ctrl-E>**.

19. Change the value of Input A to 1 and the value of Input B to 2.

20. Change the value of the Operation Enum to “Add.”

21. Run the VI, which changes the value of the Result indicator to 3.



22. Experiment with different values of the Operation Enum and run the VI again.

Video Exercise Timing, Structures, and Storing Data Modules Home FIRST Community