Programming for Performance

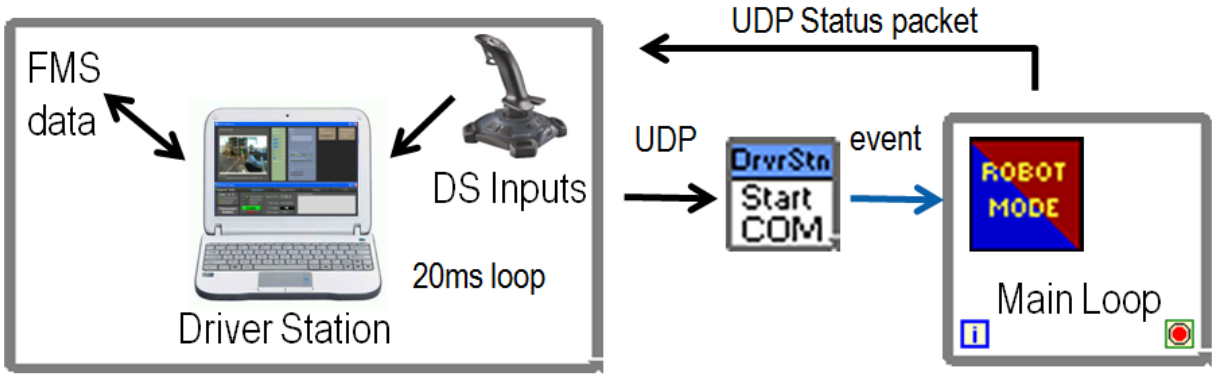
This document discusses performance concerns when programming with LabVIEW for FRC. It also outlines techniques for troubleshooting performance problems and techniques for programming efficiently.

Table of Contents

Overview

The compactRIO is a powerful tool for running programs in your embedded system. Despite its efficiency, the cRIO does have a limited amount of processing power available for running programs. This tutorial will cover the principles of programming for performance and how to determine if you are having problems based on performance.

How the System Works



The Driver Station reads inputs and publishes data every 20 ms. The cascade of events controls the Main Loop speed through UDP packets. The cRIO DLL receives these packets and sets an occurrence for the Start COM VI. Start COM reads input data and then sets an occurrence for the Robot Mode VI, controlling the execution speed of the Main loop and teleop loop. The Vision and Timed Task loops run independently.

Loop Rates on a Real-Time System

In order to be considered real-time, a program must execute within strict constraints on response time. There are two things that can keep this from happening:

1. Too much code to execute. If the CPU usage is very high, there may simply be too much code to execute within the given constraints.

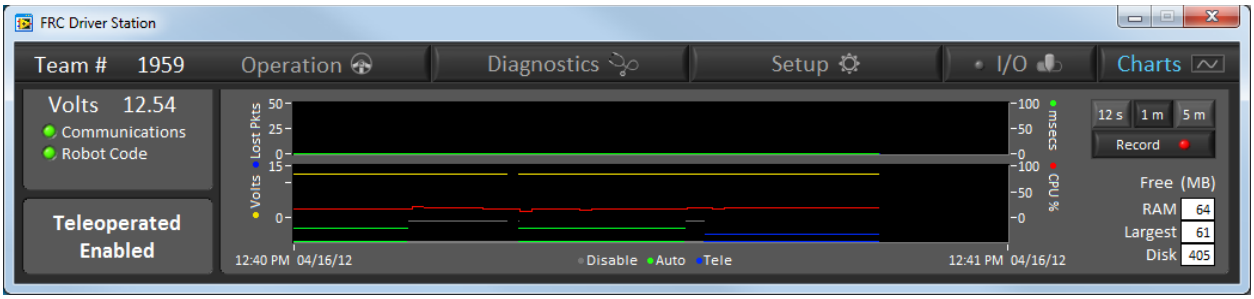
2. Delays are too long. Even if the CPU usage is low, there may be delays in the code that keep it from running within the real-time constraints.

Ideally, CPU usage should be below 80%. Common causes of CPU overload are loops without delays in them or loops with code that takes longer to run than the defined loop delay.

Tools

This section will discuss some of the tools you can use to troubleshoot performance problems.

The **Charts tab in the Driver Station** displays information on parameters such as Lost Data packets, Data packet trip time, CPU usage, Robot battery voltage, Free RAM on the cRIO, and the Robot state.



The Driver Station also has a **Log File Viewer** that you can use to review data after a match. The Viewer can be found at Program Files\FRC Driver Station\Driver Station Log File Viewer.exe and the Log Files will be stored at \Users\Public\Documents\FRC\Log Files.

These log files will include:

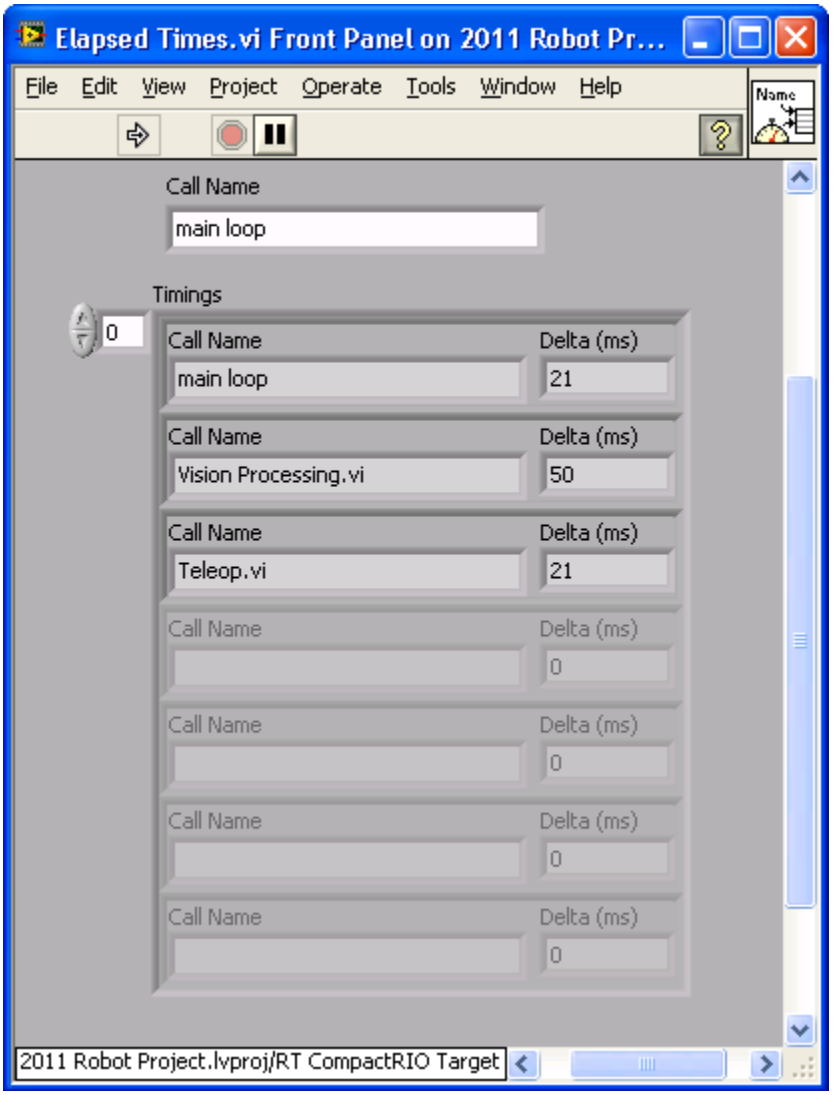
- data packet trip time
- number of lost packets
- robot battery voltage
- robot CPU usage %
- Disable, Auto, Teleop state according to the DS
- Disable, Auto, Teleop state according to the robot

Logging is enabled by default, so beware of large log files being created after days of testing.

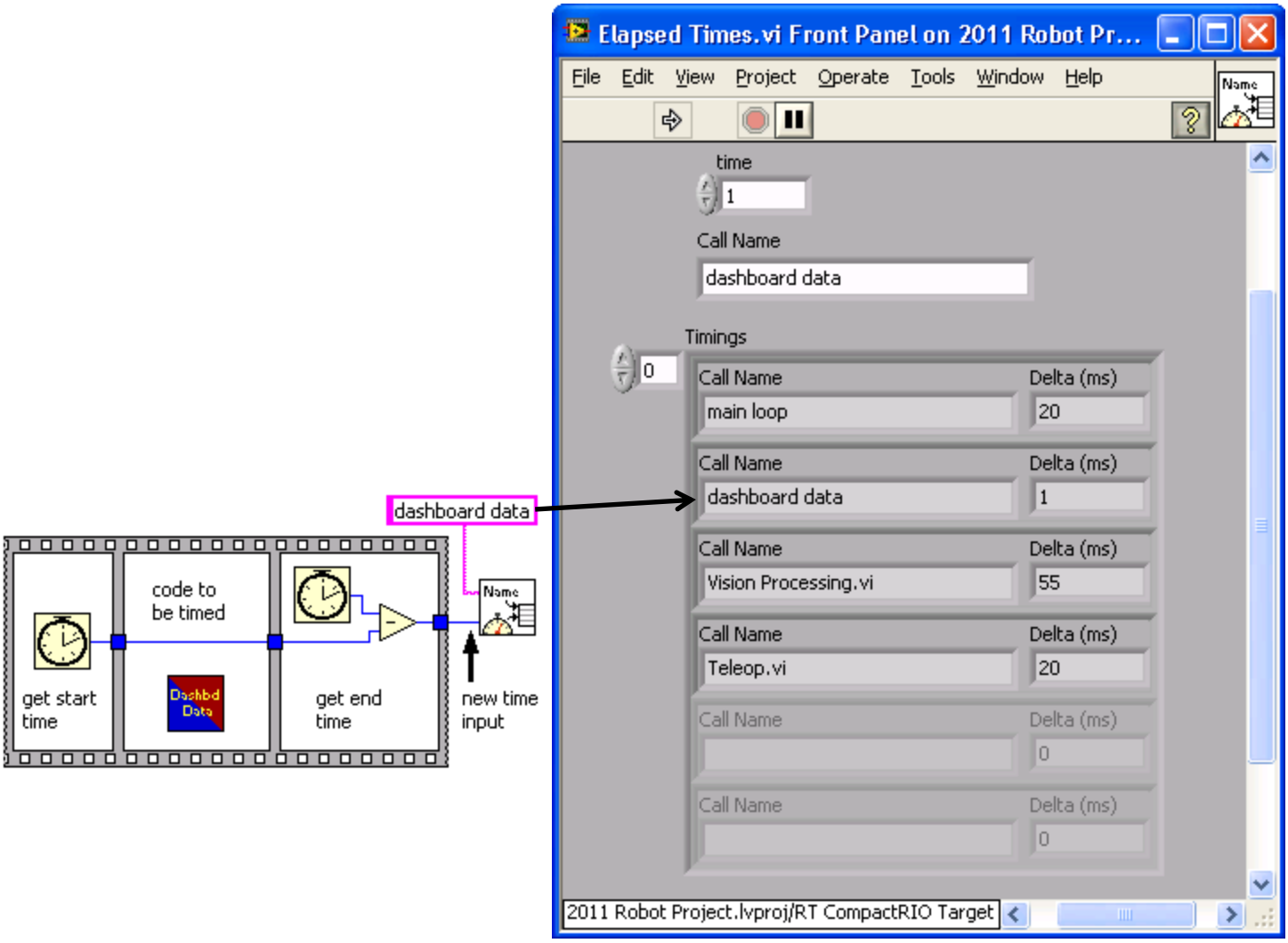
The **Elapsed Times.vi** can be used to measure time between calls in one or more locations of you code. In the LabVIEW Getting Started Window, the Troubleshooting the Robot tutorial explains details of how to use this VI. The Elapsed Times.vi can be found in the robot project under the Support Code folder.

You can use this VI throughout your robot code as a tool to show you how long it has been since a loop or a VI previously ran. Drop the Elapsed Times VI into a loop in your robot code or onto the block diagram of a robot code VI. Then, run your robot code, and open the Elapsed Times VI to see a list of all the locations in which you placed the VI, as well as the amount of time each part of the code takes to run. The Elapsed Times VI includes a **Call Name** input you can use to identify where you put the VI in

your code. If you do not wire a value to this input, the Elapsed Times VI uses the name of the VI you placed it within.

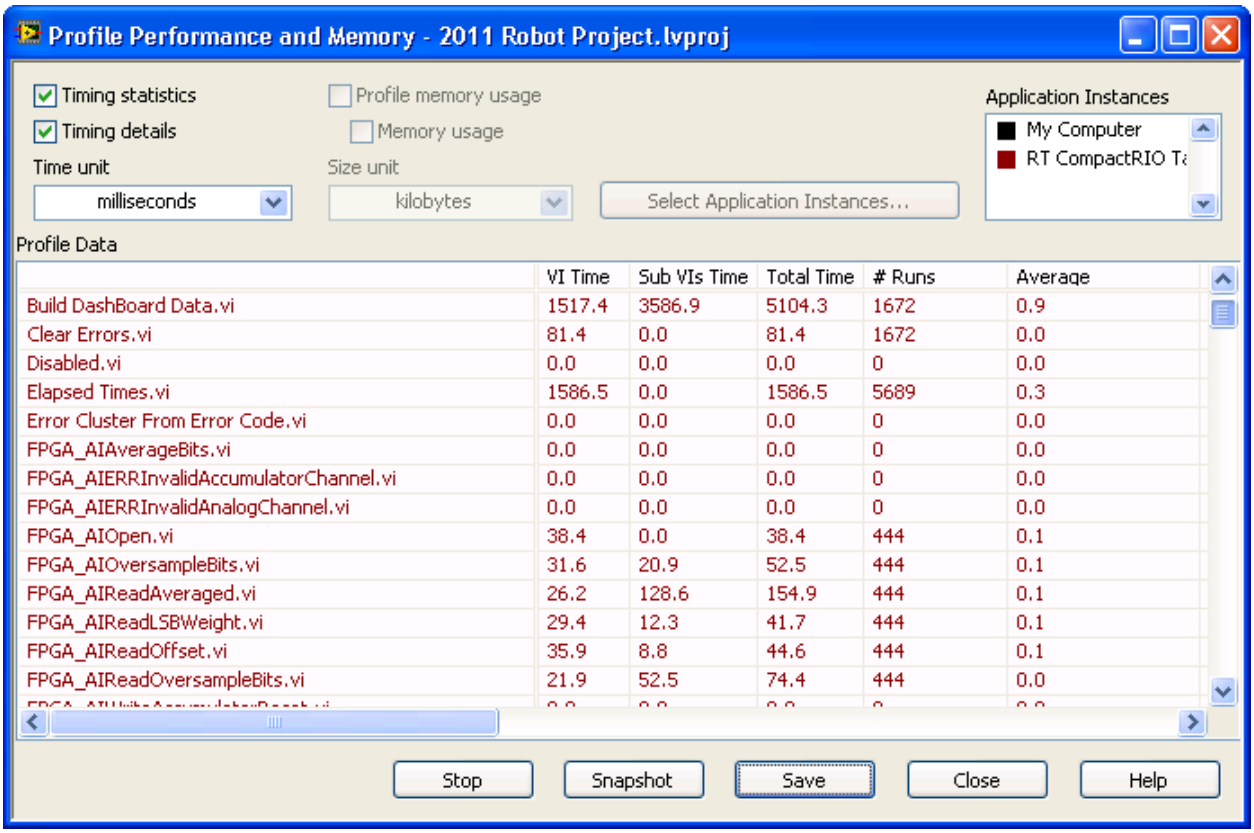


Elapsed Times.vi can be edited. In the picture below (Figure X), we have added a time input to manually define the returned Delta (ms) value instead of using the last call time. This adds the capability to time specific blocks of code. While in this example we time a single VI, this could be used to time larger snippets of code as well.



The **VI Profiler** will display timing information for individual Vis. Use this tool if you only need timing information for entire Vis and not their components. This tool can be found in LabVIEW by clicking **Tools>>Profile>>Performance and Memory.** Use the following steps to profile your Vis:

1. Open the Profiler
2. Configure it
3. Run your VI (Robot main.vi)
4. Click the Snapshot button on the Profiler



It is important to realize that all these tools themselves use resources. So, for example, once you debug using several instances of the Elapsed Times.vi, you will want to remove it from your code before you build your robot exe.

Debugging

There are a number of tools you can use to detect problems caused by performance issues. These include Watchdogs, Error explanations, and LabVIEW debugging functions.

The System Watchdog monitors communication from the Driver Station and turns off outputs if there is no robot communication for 100ms. This provides a safety mechanism for the robot in case there are system problems. Common problems that trigger the System Watchdog include:

- No communication – possibly a radio issue
- Slow communication – possibly a CPU overload
- Driver station running too slowly – Dashboard using too much CPU

The Safety Config Vis can be used as a User Watchdog alternative. They are output specific, meaning they can be turned on or off for individual outputs and configured to only act on unresponsive outputs. These Vis monitor specific update calls and, like the System Watchdog, shut off outputs if a 100ms

deadline is missed. Safety Config is enabled by default for the RobotDrive Vis and availiable for RobotDrive, MotorControl, PWM, Relay, and Solenoid. Timeout issues can commonly be caused by:

- Starved Teleop loop
- Running in Highlight Execution mode
- Hitting a breakpoint

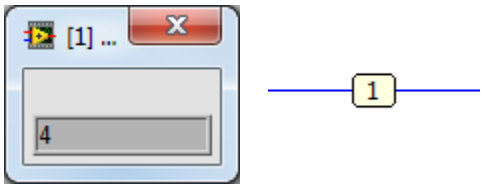
Runtime Errors also help in the debugging process. The Diagnostics tab of the Driver Station has a list of errors that occurred, including the -44061 (Safety Timeout) error code. Errors that reoccur here should be fixed with the error tools available:

- Help>>Explain Error
- Open VI Panels
- Probes

LabVIEW also has many built-in debugging tools to help in finding and resolving errors.

A broken run arrow will show you the details and location of an error.

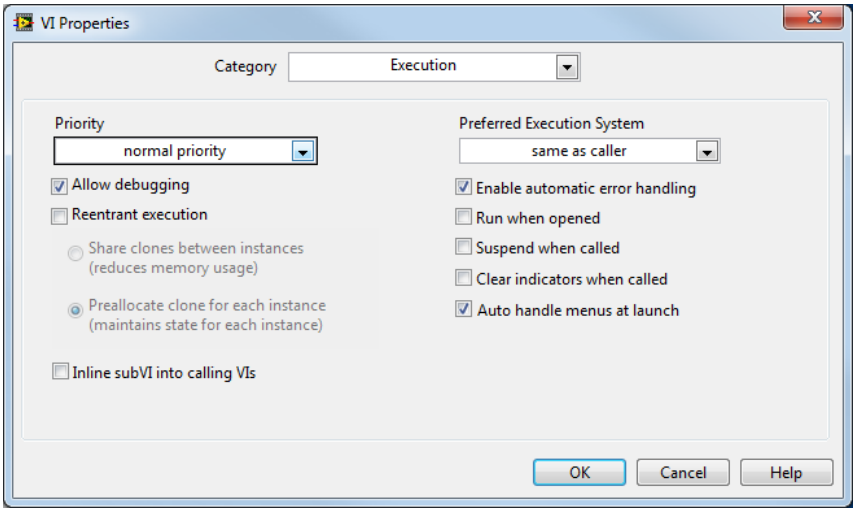
Highlight Execution animates the Block Diagram and traces the flow of the data, allowing you to view intermediate values.



Probes display values going through wires on the Block Diagram. Right-click on a wire and select Probe to place.

Programming Techniques

VI Execution Priority can be used to ensure adequate processor time for important tasks. You can find this setting by clicking File>>VI Properties and selecting the Execution tab. Here you can raise important subVIs to a higher priority for execution. If a VI with higher priority calls a subVI, the priority of the subVI rises to match the priority of the calling VI. Priority never drops below the level you set in the VI Properties dialog box. Note: Be careful not to mix VI priority with Timed Loops. Only one priority scale should be used at a time.



State Machines

State Machines allow you to transition between different pieces of code based on specific events or conditions. In LabVIEW a state machine is typically a case structure within a While Loop, with a shift register used to pass state information.

The state machine is in Teleop.vi. Note that Teleop.vi is already called in the Robot Main loop, so we don't need to add a While Loop.

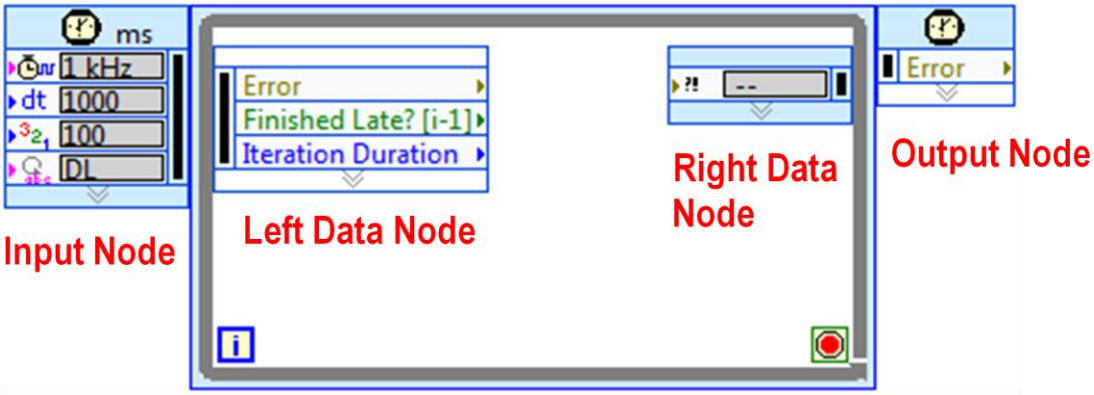
Periodic Tasks

The Periodic Tasks.vi is good for tasks that run repeatedly like control loops. You can use the existing loops in this VI to run your code or create new loops to fit your timing needs. You can pass the latest value of your data between loops using global variables, but you should only write to these from one location to avoid race conditions. Information on other data transfer methods can be found later in this article.

Timed Loops

Timed Loops allow you to run critical pieces of code that require exact timing. These should be used only when needed. The priority setting allows you to define which timed loops are more important; be careful not to mix this setting with VI Execution Priority found in the VI properties menu. Double-click on the blue section to bring up the Configuration Page.

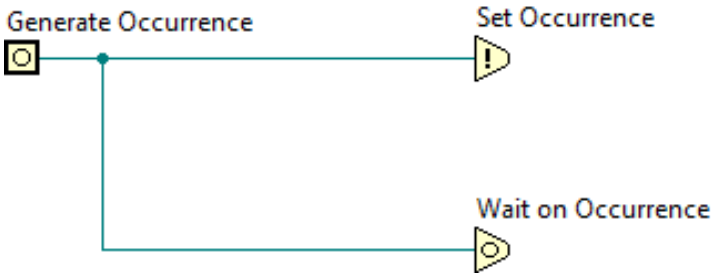
You can also change configuration settings and obtain execution information from the terminals on the Timed Loop. Input Terminals on the left set options for the loop before it starts running. These can be changed between iterations with the Right Data Node inside the loop. The Left Data Node inside the loop outputs information on how your loop and contained code are executing. Check the Context Help (Ctrl+H) to see how each of these parameters function.



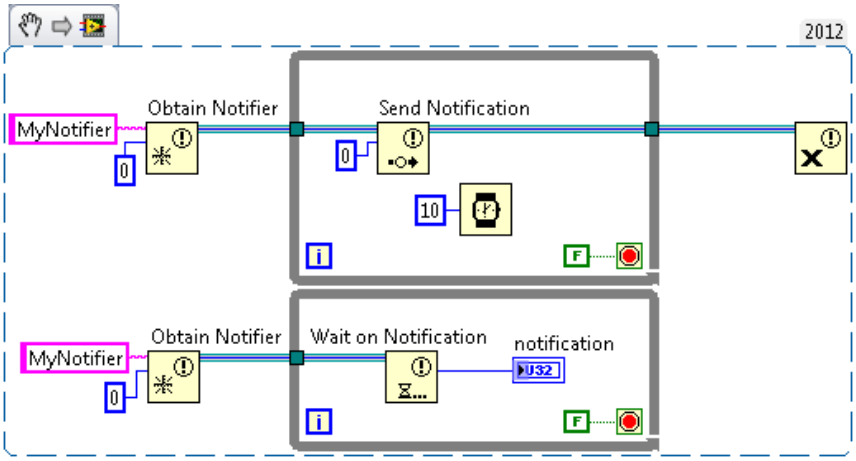
Event Driven Programming

Programming with events allows you to synchronize different parts of your code and control when loops execute.

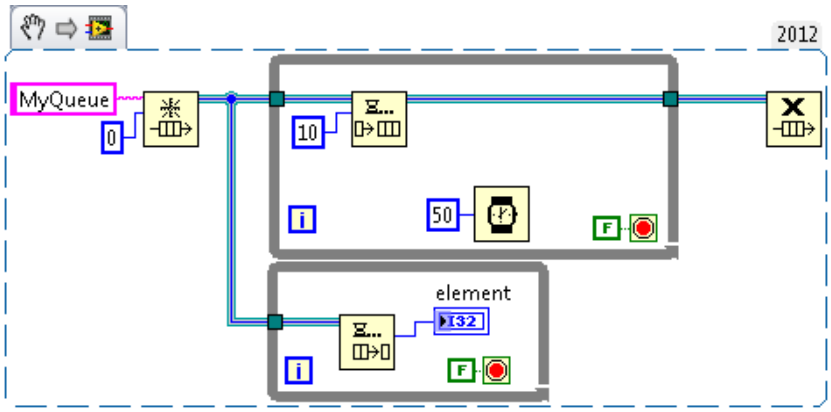
Occurrences control separate but synchronous activities. These are useful for improving an inefficient polling loop that is constantly looking for updates. Occurrences can be set in only one location, but the Wait On Occurrence VI can be used in multiple locations. These functions do not pass data; they merely control code execution.



Notifiers are similar to occurrences but can be used to pass data. These are similar to variables that do not need to be polled, as they can carry any data type but will suspend execution of a loop that is waiting on notification. Only the latest value sent with the Send Notification function will be read by the Wait on Notification function. Notifiers can also be used to synchronize loops and transfer data between Vis.



Queues store data in a buffer that can be read from another loop or VI. These functions will allow you to transfer multiple elements of data without losing any data points. Queues can transfer any data type and will hold execution of a loop until an element is placed in the queue.



Conclusion

- Make sure your loops run at expected rates
- Don’t overload the CPU
- Check for errors
- Consider using more efficient programming techniques
- Ask for help and use your resources