LabVIEW™ Real-Time 1 Course Manual

Course Software Version 2010
September 2010 Edition
Part Number 373246A-01

Worldwide Technical Support and Product Information
ni.com

National Instruments Corporate Headquarters
11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 358 (0) 9 725 72511, France 01 57 66 24 24, Germany 49 89 7413130, India 91 80 41190000, Israel 972 3 6393737, Italy 39 02 41309277, Japan 0120-527196, Korea 82 02 3451 3400, Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466, New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 328 90 10, Portugal 351 210 311 210, Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222, Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Additional Information and Resources* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the Info Code feedback.

Contents

Lesson 6
Verifying Your Application

Lesson 7
Deploying Your Application

Appendix A
Additional Information about LabVIEW Real-Time

Appendix B
Instructor's Notes

Appendix C
Additional Information and Resources

4

Timing Applications and Acquiring Data

In this lesson, you develop the deterministic loop of a target application. This typically involves control parameters, hardware input and output, and timing. This lesson focuses on software and hardware methods of timing a loop in a real-time application.

Topics

A. Timing Control Loops
B. Software Timing
C. Hardware Timing
D. Event Response – Monitoring for Events

A. Timing Control Loops

The preemptive nature of the RTOS on RT series devices can cause a deterministic loop to monopolize the processor on the device. On a single-core system, a deterministic loop might use all processor resources and not allow lower priority threads in the application to execute. Unless the deterministic loop is isolated on its own core, the deterministic loop must periodically yield processor resources to the lower-priority tasks so they can execute. By properly separating the deterministic task from lower priority non-deterministic tasks, you can reduce application jitter.

You can use software methods or hardware methods to time control loops.

B. Software Timing

LabVIEW provides multiple software timing methods. You can insert the LabVIEW Wait function, Wait Until Next Multiple function, or Wait Express VI in your code to add sleep time. Alternately, you can use a Timed Structure, such as a Timed Loop, which controls execution speed and adds other benefits. Each of these methods has a millisecond resolution when used for software timing.

Wait

The Wait Express VI causes a VI to sleep for the specified amount of time. For example, if the operating system millisecond timer value is 112 ms when the Wait Express VI executes, and the **Count (mSec)** input equals 100, then the Express VI returns when the millisecond timer value equals 142 ms.

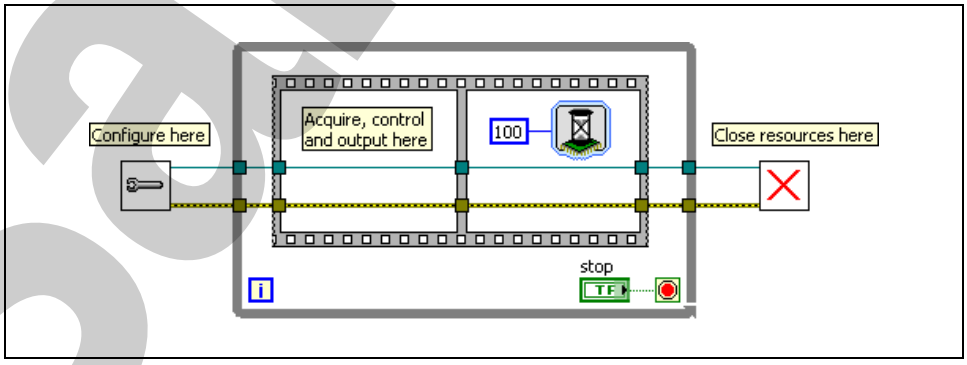

Figure 4-1. The Wait Express VI

Avoid using this Express VI in parallel with deterministic code. If the Wait Express VI executes first, the whole thread sleeps until the VI finishes, and the code in parallel does not execute until the Wait Express VI finishes. The resulting loop period is the code execution time plus the **Count (mSec)** time.

Wait Until Next Multiple

When you use the Wait Until Next Multiple Express VI, you can choose ticks, msec, or µsec resolution. The name of the **Count** input reflects the resolution you choose. When configured for ms resolution, the Wait Until Next Multiple Express VI causes a thread to sleep until the operating system ms timer value equals a multiple of the **Count (mSec)** input. For example, if the Wait Until Next Multiple Express VI executes with a **Count (mSec)** input of 100 ms and the operating system millisecond timer value is 112 ms, the VI sleeps until the millisecond timer value equals 200 ms because 200 ms is the first multiple of 100 ms after the Wait Until Next Multiple Express VI executes.

Use the Wait Until Next Multiple Express VI to synchronize a loop with the operating system millisecond timer value multiple. A loop has a period of **Count (mSec)** if the Wait Until Next Multiple Express VI executes in parallel with other code in the same loop. However, the loop does not have the period of **Count (mSec)** if the code takes longer to execute than the **Count (mSec)**.

However, avoid placing the Wait Until Next Multiple Express VI in parallel with other code because doing so can cause incorrect timing of a control system. The dataflow properties of LabVIEW programming can cause the Wait Until Next Multiple Express VI to execute before, after, or between the execution of the analog input and output. The behavior of the loop differs depending on when the Express VI executes. Instead, use a Sequence structure to control when the Express VI executes, as shown in Figure 4-2.

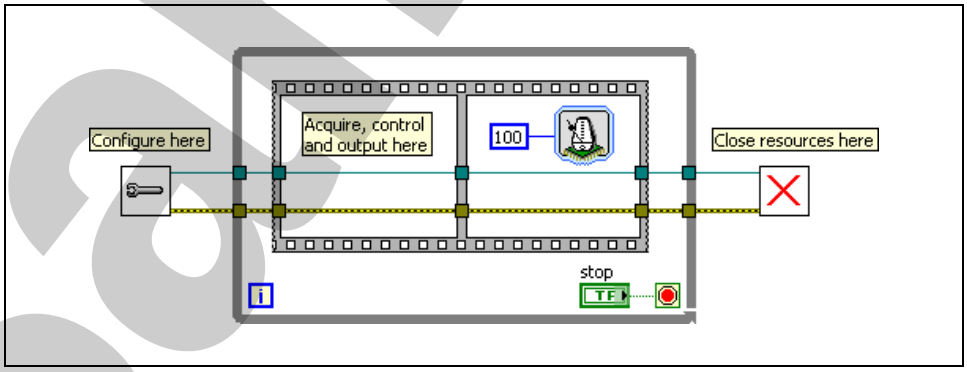

Figure 4-2. The Wait Until Next Multiple Express VI

In the Figure 4-2, the code may take a variable amount of time to finish executing, but calling the Wait Until Next Multiple Express VI afterwards enforces a loop frequency of 10 Hz (1/100 ms). The maximum achievable loop rate is 1 kHz with a wait multiple of 1 ms.

Because the Wait Until Next Multiple Express VI accepts only integers, loop rates are limited to only a few frequencies: 1000, 500, ~333, 250, 200, ~167 Hz, and so on.

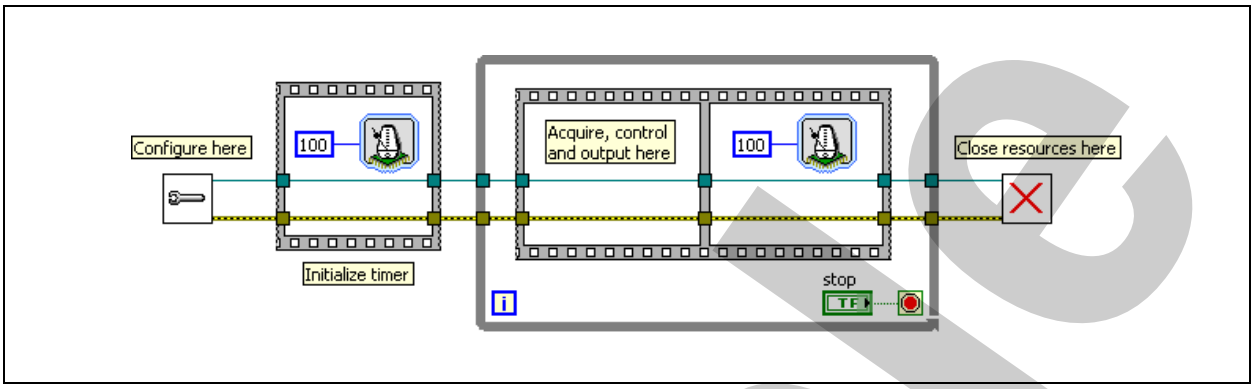

Figure 4-3. Initialized Wait Until Next Multiple Express VI

In Figure 4-3, the 100 ms timer is initialized by calling the Wait Until Next Multiple Express VI immediately before the While Loop begins. Otherwise, the loop time for the first iteration would be indeterminate. In the While Loop, placing the Wait Until Next Multiple Express VI in a sequence structure adds the delay after the code has finished. This guarantees the order of execution.

Before deciding on a **Count** value, you must ensure that the code in your loop can execute faster than the wait multiple. If the code inside the loop takes longer than the **Count** value, the loop must wait a second multiple of **Count**, because code was running when the first ms multiple arrived. In this case, the Wait Until Next Multiple Express VI is not aware that the first multiple occurred and waits until a second multiple occurs before returning.

In addition to controlling loop rates, the Wait Until Next Multiple Express VI forces time-critical VIs to sleep until the wait finishes. When a VI sleeps, it relinquishes the CPU, allowing other VIs or threads to execute. Unless the time-critical VI is isolated on its own core, sleep time is required, because the user interface and other background processes need CPU time to survive.

The Wait Until Next Multiple Express VI masks software jitter within the loop. The Express VI has some inherent jitter, which is acceptable for many real-time applications. In this example, the Wait Until Next Multiple Express VI synchronizes with each 100 ms tick of the OS clock, allowing the loop to achieve 10 Hz software-timed analog input. The timeline in Figure illustrates the 5 ms wait multiple at ΔT, the actual time required to execute the code at T_e, worst case jitter at T_j, and worst case time at T_{wc}. As long as the worst case time is smaller than the wait multiple, the actual loop

period equals the wait multiple, plus or minus the jitter incurred by the Express VI itself.

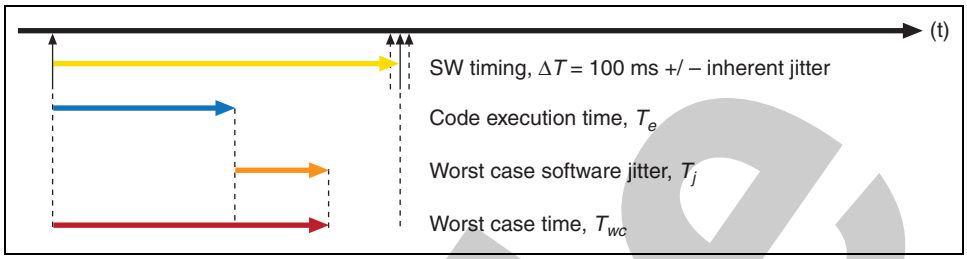

Figure 4-4. Software Timing Timeline

µs Timing

If you are targeted to an RT target that allows microsecond timing, you can use the microsecond clock for the wait Express VIs. If you are not targeted to an acceptable target, you can still select microsecond (µs) timing, but the program uses the millisecond (ms) operating system clock instead.

Using the microsecond clock allows for more loop rate options:

- With a ms clock, loop rates are *1/X* ms = 1 KHz, 500 Hz, ~333 Hz, 250 Hz, and so on
- With a µs clock, loop rates are *1/X* µs = 1 MHz, 500 KHz, ~333 KHz, 250 KHz, and so on

To use microsecond timing, double-click a Wait Express VI to open a configuration window, and set Counter Units to **µSec**.

Timed Loop

A Timed Loop executes an iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multi-rate timing capabilities, feedback on loop execution, timing characteristics that change dynamically, or manual processor assignment.

Because the Timed Loop automatically imposes sleep as needed to achieve the loop rate you specify, there is no need to use a Wait or Wait Until Next Multiple Express VI to add sleep time in the loop.

Because of the preemptive nature of Timed Loops, they can monopolize processor resources. A Timed Loop might use all of the processor resources, not allowing other tasks on the block diagram to execute. You must configure the highest priority Timed Loop with a period large enough to perform the deterministic task and have idle time during every iteration to allow lower priority loops to execute.

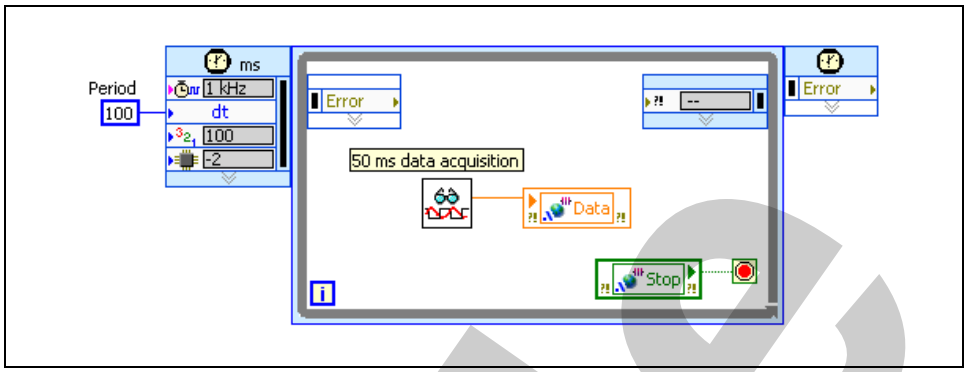

Figure 4-5. Timed Loop

Figure 4-5 contains a subVI that performs a data acquisition for 50 ms. The Timed Loop has a period of 100 ms that allows the loop to remain idle for 50 ms during each iteration. During the time when the Timed Loop remains idle, LabVIEW can execute lower priority tasks on the block diagram.

To configure the Timed Loop for microsecond timing, choose the MHz clock (µs timer) in the Loop Timing Source section of the Timed Loop configuration window if you are targeted to an appropriate target. Access the Timed Loop configuration window by double-clicking the Input node of the Timed Loop.

C. Hardware Timing

You can implement hardware timing in your real-time application by using external timing sources. The National Instruments drivers that run on RT targets support VIs or functions that can cause sleep in the current LabVIEW thread and return when the driver detects a specific event. For example, you can use NI-DAQmx and NI data acquisition hardware to time real-time applications. Refer to the specific NI driver documentation for information about VIs or functions that you can use to sleep and wait for driver events.

DAQmx

You can use NI data acquisition hardware and NI-DAQmx to achieve a sleep resolution much finer than 1 kHz. Hardware timing uses the DAQ device internal clock or an external clock to control timing. You can use the DAQmx VIs to control when a Read VI or a Write VI executes within a loop. Alternately, you can wire a DAQmx task to a Timed Loop to tie the loop rate to the hardware clock.

Use the DAQmx Timing VI to configure the sample clock, which controls the loop rate. You can configure the DAQmx task according to your application, however, the Hardware Timed Single Point option for the **sample mode** input provides the best access to the hardware clock. Notice that when the requested scan rate is too fast relative to the code execution time, you may miss ticks from the clock. In other words, if the RT target is not powerful enough to execute the code within the loop at least as fast as the scan rate, the clock rate will be slower than configured.

You can use NI data acquisition hardware with NI-DAQmx to match loop rates to match the rate of the hardware clock. With NI-DAQmx, you can use the following methods to time real-time applications:

- **Hardware-Timed Single-Point**—NI-DAQmx supports hardware-timed, single-point sample mode in which samples are acquired or generated continuously using hardware timing and no buffering. You can use hardware-timed, single-point mode for control applications that require input and/or output within a deterministic period of time. Refer to the *NI-DAQmx Single-Point Real-Time Applications* topic of the *NI-DAQmx Help* for information about using hardware-timed, single-point operations to time your deterministic application.
- **Counter Timers**—NI-DAQmx supports using hardware-timed counter input operations to drive a control loop. Use the Wait For Next Sample Clock VI to synchronize the counter operations with the counter's sample clock. Refer to the *Hardware-Timed Counter Tasks* topic of the *NI-DAQmx Help* for information about using counter input operations to time deterministic applications.
- **DAQmx Timing Sources for Timed Structures**—Timed structures can be hardware-timed and are ideal for multirate applications. By default, timed structures use the 1 kHz clock on Windows or the real-time operating system of an RT target as a timing source. You also can use an external signal on a DAQ device as the timing source of a timed structure using NI-DAQmx. Use the DAQmx Create Timing Source VI to create a timing source that can synchronize the timed structure with the hardware clock. Refer to the *Hardware-Timed Simultaneously Updated I/O Using the Timed Loop* topic of the *NI-DAQmx Help* for information about using an external signal on a DAQ device to control a timed structure.

You can create external timing sources for controlling a timed structure with NI-DAQmx. Use the DAQmx Create Timing Source VI to programmatically select an external timing source. You also can use several types of NI-DAQmx timing sources, including frequency, digital edge counters, digital change detection, and signals from task sources, to control timed structures. Use the DAQmx - Data Acquisition VIs to create the following types of NI-DAQmx timing sources to control a timed structure.

- **Frequency**—Creates a timing source that causes a timed structure to execute at a constant frequency.
- **Digital Edge Counter**—Creates a timing source that causes a timed structure to execute on rising or falling edges of a digital signal.
- **Digital Change Detection**—Creates a timing source that causes a timed structure to execute on rising or falling edges of one or more digital lines.
- **Signal from Task**—Creates a timing source that uses the signal you specify to determine when a timed structure executes.

Refer to the *NI-DAQmx Help*, available by selecting **Start»All Programs» National Instruments»NI-DAQ»NI-DAQmx Help**, for information about using NI-DAQmx VIs and functions to control timed structures.

D. Event Response – Monitoring for Events

With real-time event response, you can respond to a single event within a given amount of time. Some common events include detecting a peak in a measurement or detecting when a threshold has been reached. You can use the Point-by-Point Signal Analysis VIs to detect these types of events as shown in Figure 4-6.

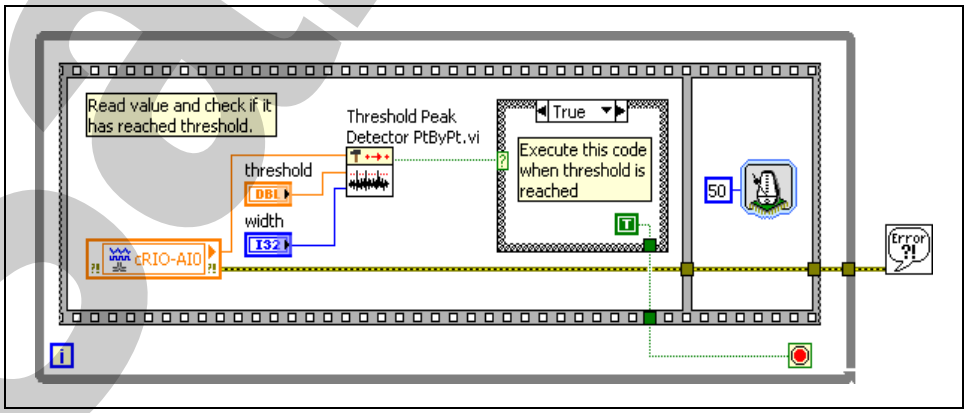

Figure 4-6. Monitoring for An Event Using A Point-by-Point VI

Event Response – Digital Change Detection

(NI-DAQmx only) Another common event response application involves watching for a digital line change, which is useful when watching for an alarm trigger. Figure 4-7 uses the DAQmx digital change functionality with the Timed Loop. The digital line is connected to the **Source Name** terminal of the Timed Loop. When the digital change happens or a timeout occurs, the Timed Loop wakes up and executes the code in the loop.

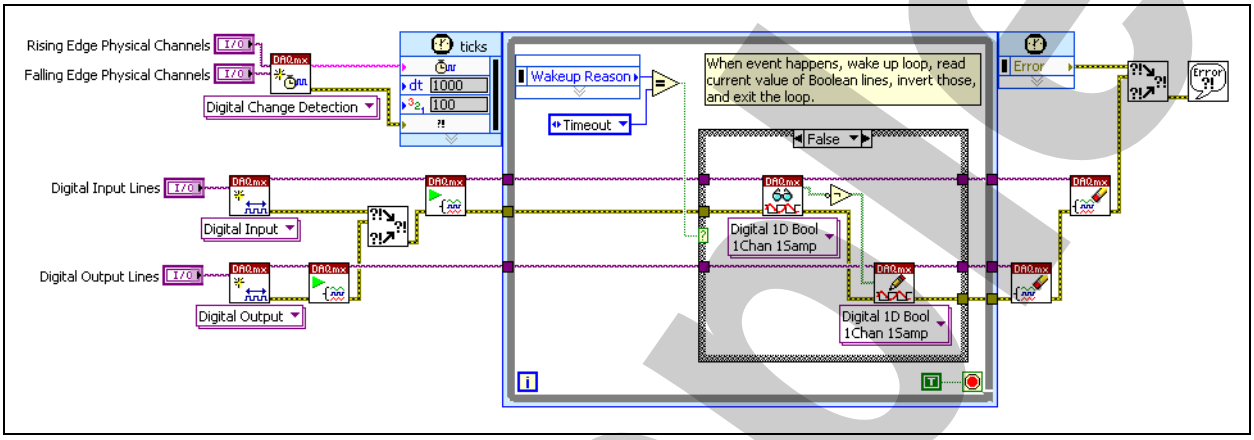

Figure 4-7. Digital Change Detection

Summary – Quiz

1. Which of the following are benefits of using timing in a control loop?
   a. Provide sleep so lower priority threads can execute
   b. Reduce application jitter
   c. Both a & b

2. True or False? It is good programming practice to use wait functions in parallel with time critical code.

3. Which of the following typically provides finer resolution?
   a. Hardware timing
   b. Software timing

4. Which of the following methods use hardware timing?
   a. Timed Loop linked to a µs clock
   b. DAQmx VIs connected to an external clock
   c. Wait Express VI with µs resolution
   d. Timed Loop linked to a ms clock

Summary – Quiz Answers

1. Which of the following are benefits of using timing in a control loop?
   a. Provide sleep so lower priority threads can execute
   b. Reduce application jitter
   c. Both a & b

2. True or False? It is good programming practice to use wait functions in parallel with time critical code.
   False

3. Which of the following typically provides finer resolution?
   a. Hardware timing
   b. Software timing

4. Which of the following methods use hardware timing?
   a. Timed Loop linked to a µs clock
   b. DAQmx VIs connected to an external clock
   c. Wait Express VI with µs resolution
   d. Timed Loop linked to a ms clock

Notes