Document Type: Tutorial
NI Supported: Yes
Publish Date: Oct 20, 2011

Real-Time FIFO for Deterministic Data Transfer Between VIs

Table of Contents

Overview

In LabVIEW Real-Time(RT) applications, you may need to transfer data to or from a VI set to time critical priority (time critical VI). This data can be transferred to or from a non-deterministic system, such as over the network to another computer, or to the hard drive. Typically, the data is first transferred to or from a VI set to normal priority, or a priority lower than time critical, which we will call the Communication Loop. Non-deterministic operations can then be performed in the Communication Loop without harming the real-time performance of the Time-critical Loop. The Communication Loop can then communicate with an external application over the network, or perform File I/O.

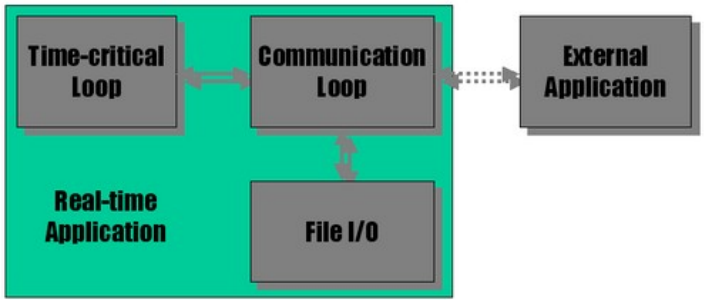

Figure 1: A common real-time application structure

Communicating Data Between VIs

There are three methods to communicate between VIs: global variables, functional globals (a VI that acts like a global variable), and a RT FIFO. FIFO stands for "**F**irst **I**n, **F**irst **O**ut". Global variables are a lossy form of communication since there can be many writes to the global variable before a read is ever performed, thus data can be lost. Since only one VI can access a global variable at a time, it can cause priority inversions, which in turn cause jitter, or increase execution time, in the time critical loop. Functional globals have a similar behavior. However, with the RT FIFO, a write and a read can be performed at the same time. Also, the RT FIFO acts like a fixed size queue, so that data elements that you write to an RT FIFO do not overwrite previous elements, unless the RT FIFO is full, in which case the oldest element is over written. The RT FIFO can be a lossy communication if the reader does not read elements from the FIFO before the FIFO fills up. The advantage of using the RT FIFO is that even if the reader pauses momentarily, and multiple writes to the RT FIFO occur during that time, data is not lost as long as the reader can catch up and read the elements out of the RT FIFO before it fills up. This document will focus on the third method of communication between VIs: the RT FIFO.

Using RT FIFO

The RT FIFO is simple to use. Its use involves creating the RT FIFO in one VI and either passing the RT FIFO reference to the other VI, or opening a reference to the RT FIFO in the another VI. Then writes to the RT FIFO are performed in one VI, and reads from the RT FIFO performed in another VI, and finally, the RT FIFO is deleted. Figure 2 shows an example of using the RT FIFO, although in this example data is transferred to and from the same VI.

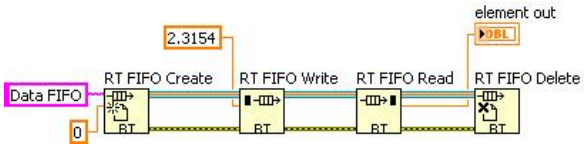

Figure 2. A Very Simple RT FIFO Example

Creating an RT FIFO

An RT FIFO is created using the RTFIFOCreate VI. The type of elements that the RT FIFO will contain is determined by type of the data wired to the **type** input terminal (the RT FIFO VIs are polymorphic). All of the elements in an RT FIFO must be of the same type. RT FIFOs can be created of elements of the following types:

- Numeric
- Waveform
- Boolean (Element only)
- Clusters of the above datatypes

RT FIFO elements can also be arrays of the same types as listed above. To specify the size of the arrays in each RT FIFO element, first wire an array to the **type** input terminal, then wire the size for the arrays to the **elements in array** input terminal. The reference output for the RT FIFO, **rt fifo**, is a cluster, and will be pink for RT FIFOs with array elements or booleans. In Figure 3, an RT FIFO called Data FIFO is created that contains array elements. In this case each array element has 5 elements.

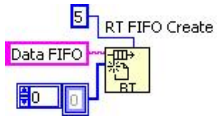

Figure 3. Creating an RT FIFO of Array Elements

The RT FIFO is fixed length, and the memory for the RF FIFO is allocated when the RT FIFO is created. If the RT FIFO was of unlimited length, then it would have to dynamically allocate more memory as the number of elements in the RT FIFO increased. Determinism, or real-time behavior, of time critical VIs would be harmed if this dynamic memory allocation occurred inside of the time critical VI. To specify the size of the RT FIFO, wire the desired size to the **size** input terminal of the RTFIFOCreate VI. In Figure 4, an RT FIFO of 20 elements is created.

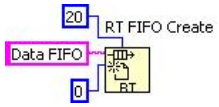

Figure 4. Specifying the RT FIFO Size

Opening a reference to an existing RT FIFO

To open a reference to an existing RT FIFO, use the RTFIFOCreate VI with the name of the RT FIFO wired to the **name** terminal. If an RT FIFO with that name does not exist, wiring a TRUE wired to the **create if not found** terminal as shown in Figure 3, will create a new FIFO. Wiring a FALSE to this will create an error code if the RT FIFO is not found.

Figure 5. Opening a Reference to an Existing RT FIFO

Writing when full or reading when empty

If you perform a write when the RT FIFO is full and a slot does not become available before the TIMEOUT, the oldest element is overwritten and the **timeout** output will be TRUE. To avoid overwriting elements, you need to either make sure that the reader will always read elements from the RT FIFO faster than they are written, you need to ensure that the RT FIFO is large enough to contain the elements that are written to it while the reader is unable to read or you can wire a FALSE into the **overwrite on timeout** input. In the case that **overwrite on timeout** is FALSE, the data to be written will be disregarded.

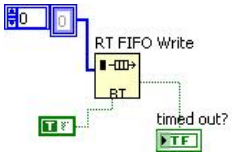

Figure 6. **Overwrite** Input.

If you perform a read when the RT FIFO is empty, then the **empty** output will be TRUE.

Figure 7. **Empty** Output.

Related Links:
Real-Time FIFO Example