Document Type: Tutorial
NI Supported: Yes
Publish Date: Mar 30, 2009

Considerations in Implementing Real-Time Drivers

Overview

This document is intended for anyone who is interested in implementing a Real-Time driver for a third-party PXI, cPCI or PCI device. It should help you to choose the best implementation approach for your particular situation.

Table of Contents

Background

The implementation approaches recommended in this document are based on the VISA application programming interface (API).

VISA is a set of open standards developed and supported by more than 50 of the test and measurement industry's leading equipment vendor, user, and system integrator organizations. The VISA standards were created to improve the efficiency of integrating and maintaining multivendor test and measurement systems based on the VXI and VME hardware platforms or controlled through IEEE 488 (GPIB) and RS232/422 cabled interfaces.

The National Instruments implementation of the VISA software standards is called NI-VISA. In addition to the device types listed above, NI-VISA also works with CompactPCI and PXI devices on the Windows and LabVIEW Real-Time platforms. Through the NI-VISA API you can perform control/status register (CSR) read/write operations, interrupt handling, and shared memory allocation for direct memory access (DMA). NI-VISA can be used to create instrument drivers for third-party devices, using the architecture shown in Figure 1.

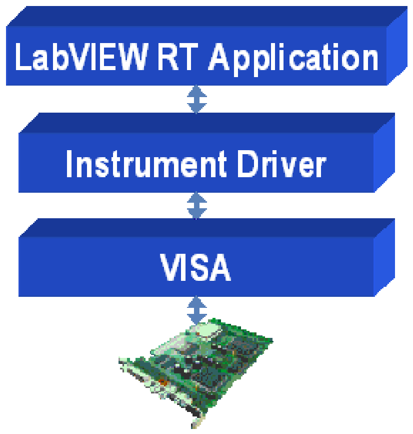

Figure 1. LabVIEW Real-Time Instrument Driver Architecture

NI-VISA licenses are included with National Instruments system control and instrumentation hardware, as well as with NI LabVIEW, LabWindows/CVI, Measurement Studio, and other selected NI application development tools. NI-VISA deployment and distribution licenses are also available separately from National Instruments.

Device Complexity

To evaluate the best driver implementation strategy, it is important to examine some key properties of the device. The complexity of the driver is usually related directly to the complexity of the device, which can vary widely.

Some devices are very simple in function, such as a switch module in which a driver directly controls the state of each relay by setting individual bits in its control/status registers (CSRs). A driver for this type of device can usually be implemented with relatively little effort.

Other devices have more complicated register maps that do not directly map to the functionality desired in a typical application. For example, the configuration, status, and data information related to a single I/O function may be mapped to several registers. Conversely multiple I/O functions can be mapped to several bit fields within a single register. An instrument driver for this type of device must translate the functionality provided through its API to the programming logic required by the hardware registers.

Devices that generate asynchronous events or transfer large amounts of data often use interrupts and/or DMA to reduce the workload on the processor. These features will add another level of complexity to the driver.

In summary, as device complexity increases, the driver must do more work to translate the driver API functions to sequences of register operations.

Note: For core I/O interfaces (keyboard, mouse, video, disk storage, networking, serial port, printer port, etc.) the operating system provides software infrastructure that handles standard protocols and APIs. It is important to understand that under VISA a driver communicates with the device, VISA does not provide access to the device through standard interfaces such as file I/O or TCP/IP. For this reason, core I/O interfaces are typically not good candidates for VISA-based driver development.

Creating a Native LabVIEW Driver

An instrument driver can be implemented in native LabVIEW code using the NI- VISA API for LabVIEW . For simple devices, it may be possible to create a driver using only the hardware reference manual provided by the device manufacturer.

For complex devices, it is important to secure support from the manufacturer. Existing driver source code is a useful reference, even though you will be rewriting it in LabVIEW.

National Instruments provides standardized support for source-level debugging of LabVIEW code in the LabVIEW Real-Time environment.

Porting an Existing Driver to LabVIEW

Rather than recreating the driver in LabVIEW, it may be more efficient to port an existing C driver from its original platform to VISA, and then to build wrapper VIs around the API function calls. This approach will require some patience as well as good technical support from the device manufacturer. It is useful when the existing driver provides a lot more functionality besides just communication with the device. Because reimplementing all that functionality in LabVIEW would be complicated and time-consuming, porting the existing driver would result in a more efficient effort.

***Note**:* At the time of this writing, the LabVIEW Real-Time execution environment cannot handle source-level debugging for C code. National Instruments does not provide standard support for this approach. C code can be ported to LabVIEW and debugged under Windows, and then executed on the target Real-Time system. Because NI-VISA works under both Windows and LabVIEW Real-Time, the Windows-debugged driver should operate under LabVIEW Real-Time without modification. Because of differences in the Windows and LabVIEW Real-Time execution environments, there is no guarantee that this method will work. However, numerous complex drivers have been implemented successfully using this approach

Determinism

LabVIEW Real-Time system implementations are usually driven by requirements related to deterministic execution. As a driver becomes more complex, it is increasingly difficult to predict or characterize the impact that it will have on these characteristics.

Only drivers that give up the processor regularly are approapriate for LabVIEW Real-Time applications. When designing a LabVIEW Real-Time driver make sure that the driver releases the CPU often so that other tasks can be completed.

When porting a C-based driver, examine the existing design and make any changes that are necessary.

Where to Go from Here

With this application note you should have a good idea of the trade-offs related to choosing a driver implementation. For additional information on driver code development, read the following documents:

Configuring the Real-Time Environment to Recognize a Third Party Device describes the process of configuring NI-VISA to recognize the PCI device.

If you would like to learn more about writing a LabVIEW-based driver, you can read the application note Developing a LabVIEW Real-Time Driver for a PXI or Compact PCI Device.

If you would like to learn more about porting an existing Windows device driver to the Real-Time environment, you can read the application note Porting a Windows Device Driver to LabVIEW Real-Time.