



UiO : University of Oslo

FYS3240

PC-based instrumentation and microcontrollers

LabVIEW Real-Time and Embedded

Spring 2011 – Lecture #10



Embedded Computing

- An **embedded system** is a **computer system designed to perform one or a few dedicated functions, often with real-time computing constraints.**
- Embedded processors can be microprocessors, microcontrollers or FPGAs.
- Embedded systems run with limited computer hardware resources: limited memory, small or non-existent keyboard and/or screen

General Purpose Operating Systems

- Windows, Linux, MacOS, Unix
 - Processor time shared between programs
 - OS can preempt high priority threads
 - Service interrupts –keyboard, mouse, Ethernet...
 - Cannot ensure that code finish within specified time limits!

Selecting an Operating System

General Purpose OS

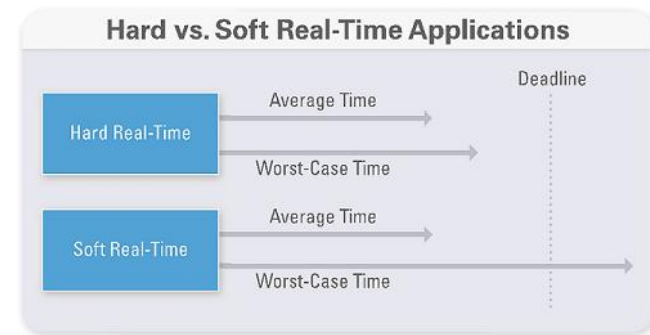
- Features
 - User interface
 - Enterprise connectivity
 - Peripheral interrupts
 - Background applications
 - OS that controls all scheduling
- Applications
 - Buffered data acquisition
 - Offline analysis
 - Data presentation

Real-Time OS

- Features
 - Embedded
 - Deterministic
 - Control over OS
 - Schedule that ensure that high-priority tasks execute first
- Applications
 - Closed Loop Control
 - Time-critical decision making
 - Extended run time
 - Increased reliability
 - Standalone operation

What is a real-time system




- A real-time system gives you determinism
 - real-time does not mean “real fast” (it can be slower)!
 - real-time means that you can determine (predict) accurately when a section of your program will execute
- Hard real-time
 - systems where it is absolutely imperative that responses occur within the required deadline (Example: Flight control systems)
- Soft real-time
 - allows for some deadlines to be missed with only a slight degradation in performance but not a complete failure (example: DAQ-systems)
- In contrast, on an ordinary desktop PC (with Windows) the OS operates on a fairness basis
 - Each application gets time on the CPU regardless of its priority
 - Even our most time-critical application can be suspended for some routine maintenance



LabVIEW Real-time (RT) systems

- The **LabVIEW Real-Time Module** extends LabVIEW to be able to target off-the-shelf real-time targets
 - LabVIEW code can be made to execute with hard real-time performance
- The application is developed under Windows on a regular PC, and then downloaded to run on the real-time target

LabVIEW Real-Time Module

✉ [E-mail this Page](#) Configure Page for:  [Print](#)  [PDF](#)  [Rich Text](#)



[\[+\] Enlarge Picture](#)

- Design real-time applications with graphical programming
- Download to a dedicated target for reliable, deterministic performance
- Deploy as a distributed, stand-alone, or embedded system
- Use built-in PID control functions or create your own control algorithms
- Purchase individually or as part of the NI Developer Suite

Build vs. Buy for Embedded systems

- Buy COTS (Commercial-off-the-shelf) hardware when possible
- Examples of when a custom build is necessary:
 - High volumes (10,000+)
 - An iteration on an existing custom design
 - Custom size or shape required
 - Very stringent technical requirements (such as ultralow power consumption)

[NI paper](#)

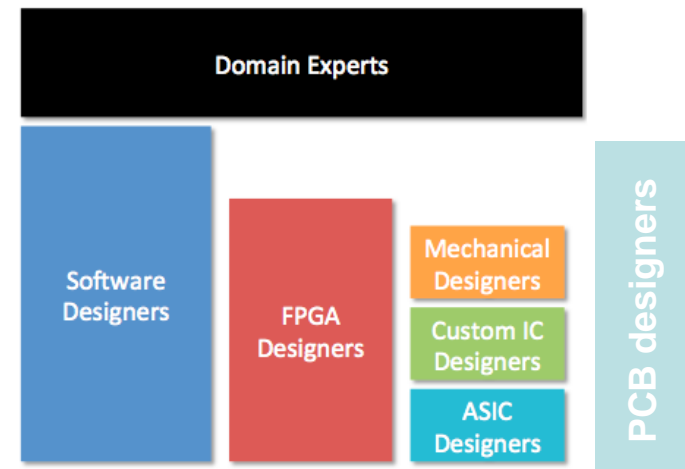
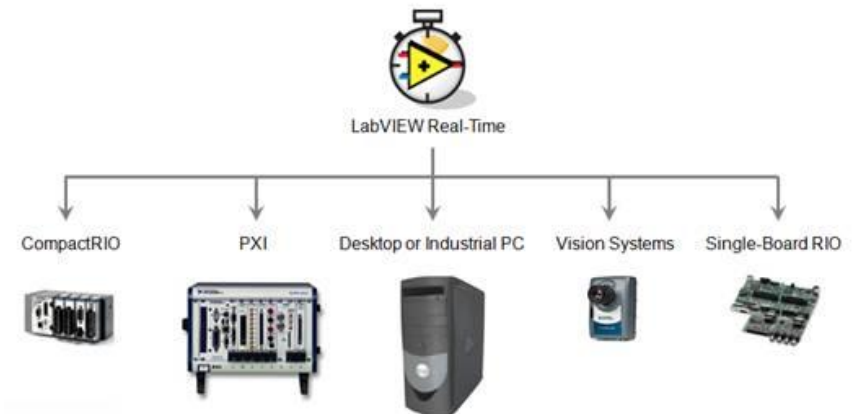


Figure 1. Custom design traditionally takes a large design team with different levels of hardware, software and application expertise

Real-time hardware platforms

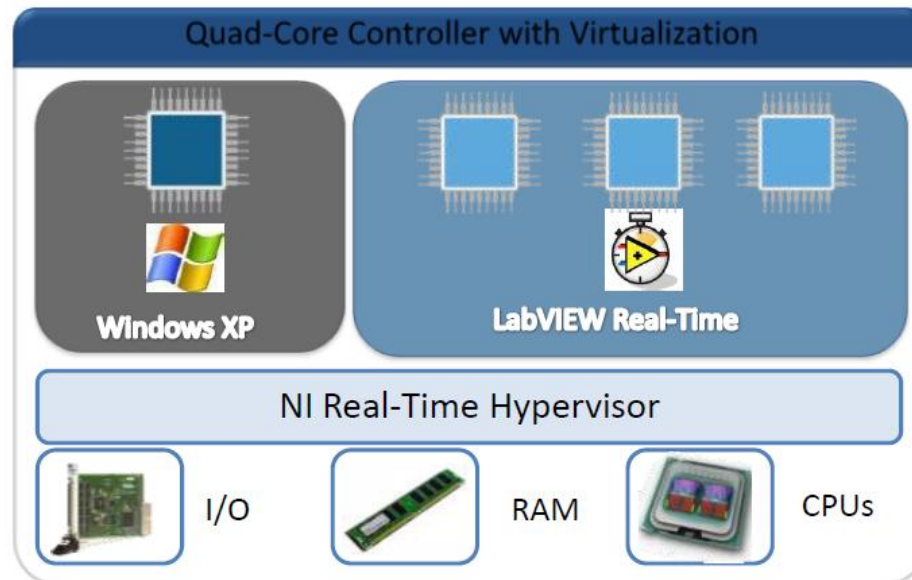
- Desktop PC with real-time OS (RTOS)
 - as long as the hardware meets certain system requirements
- **PXI with real-time controller**
 - often used for high-performance real-time systems such as hardware-in-the-loop testing
- **NI FPGA**
- **NI CompactRIO**
- **NI Single-Board RIO**
- NI CompactVision
- Industrial PCs/Controllers
- NI Compact FieldPoint
 - a PLC (programmable logic controller)



PXI - NI Real-Time Hypervisor

Can run real-time OS and Windows on the same controller!

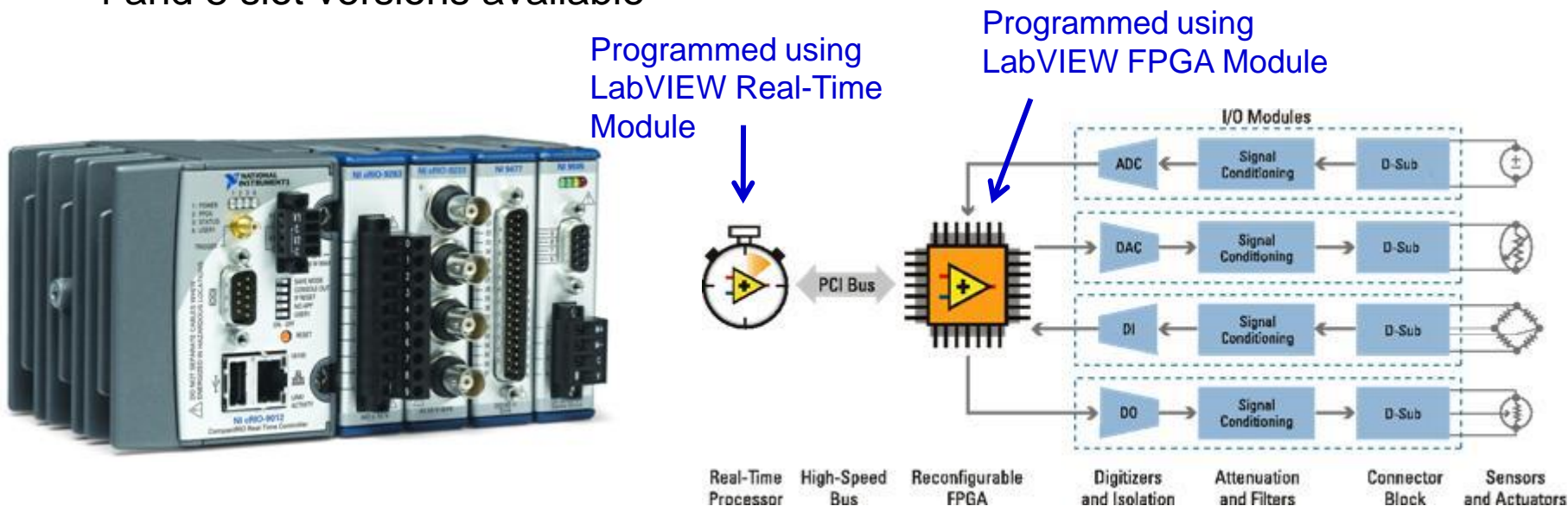
Run LabVIEW
RealTime and
Windows XP
simultaneously



Available on multicore PXI
and industrial controllers

NI CompactRIO platform

- **CompactRIO (cRIO)** combines a real-time processor, a Field-Programmable Gate Array (FPGA), and I/O modules in a small, rugged form factor.
- Serial, USB, and Ethernet ports are built in to the controller. When using CompactRIO, your I/O modules (e.g. for digital I/O, bus communication, A/D conversion) are connected to the FPGA for fast processing in hardware, and then you exchange data between the FPGA and the real-time processor as desired.
- 4 and 8 slot versions available






NI Single Board RIO




- NI Single-Board RIO systems are identical in architecture to CompactRIO systems, only in a single circuit board form factor
- Single-Board RIO hardware features a real-time processor and programmable FPGA just as with CompactRIO, and several I/O modules are also available in a board-only form factor.
- Users can easily port applications prototyped on NI CompactRIO hardware to the Single Board RIO (e.g. for high-volume applications)






Input/Output Device comparison

I/O Availability	PXI	CompactRIO	Standard or Industrial PCs
<p>○ Good</p> <p>◐ Better</p> <p>● Best</p>			
Variety	●	◐	●
Standard Driver APIs	◐	◐	◐
Customizability	◐	●	◐
Expandability	●	◐	●

Performance comparison

Performance	PXI	CompactRIO	Standard or Industrial PCs
<input type="radio"/> Good <input checked="" type="radio"/> Better <input checked="" type="radio"/> Best			
Deterministic Execution	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
Timing, Triggering, and Synchronization	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
Processor Speed	<input checked="" type="radio"/>	<input checked="" type="radio"/>	<input checked="" type="radio"/>
Multicore Processing	<input checked="" type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

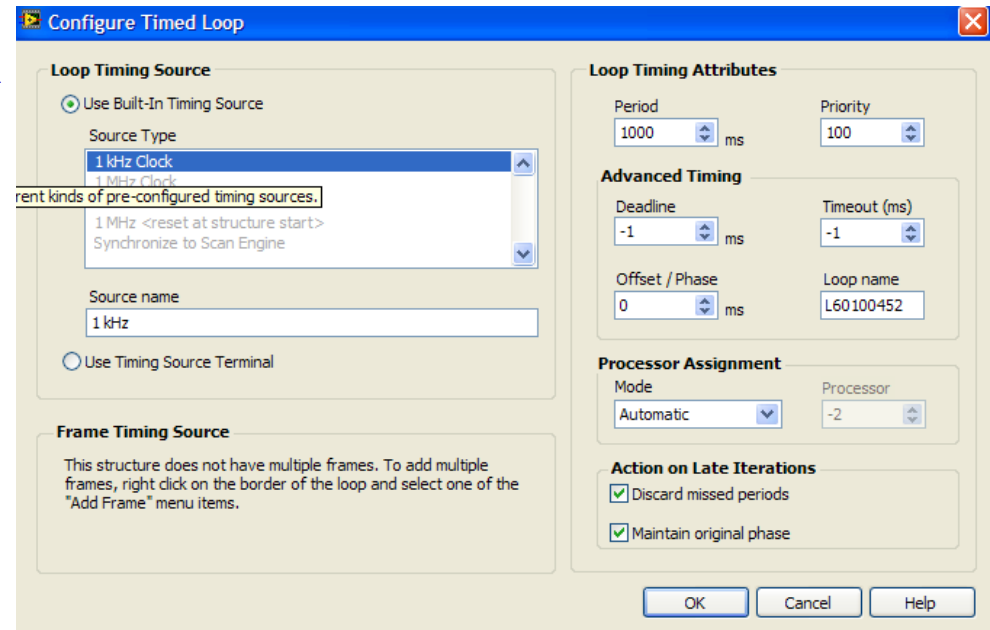
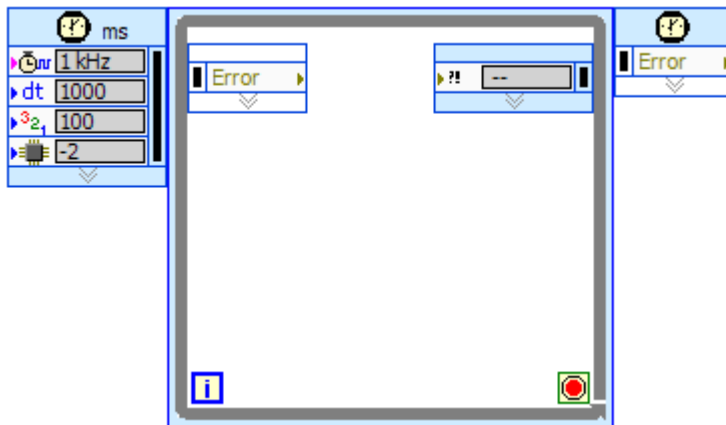
Ruggedness and portability comparison

Physical Attributes	PXI	CompactRIO	Standard or Industrial PCs
<p>○ Good</p> <p>◐ Better</p> <p>● Best</p>			
Ruggedness	◐	●	Varies
Portability	◐	●	◐

LabVIEW – Timed loops

- Simplifies the way you schedule real-time execution order (by giving it a priority, a periode/frequency and offset) for parallel loops

double-click



Common Pitfalls of Data Communication

Race conditions- two requests made to the same shared resource

Deadlock- two or more depended processes are waiting for each other to release the same resource

Data loss- gaps or discontinuities when transferring data

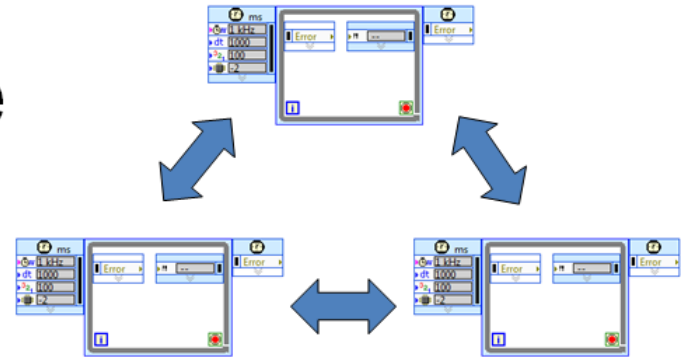
Performance degradation- poor processing speed due to dependencies on shared resources

Buffer overflows- writing to a buffer faster than it is read from the buffer

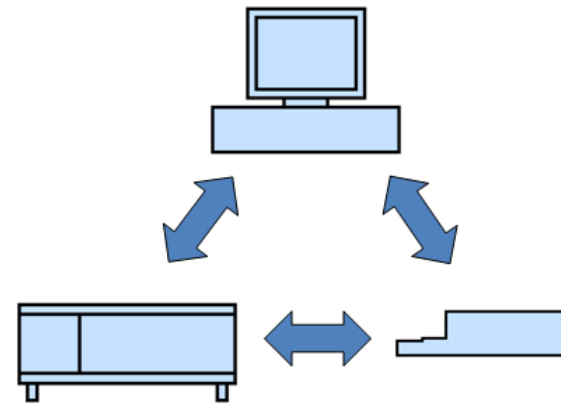
Stale data- reading the same data point more than once

Scope of Communication

Inter-process: the exchange of data takes place within a single application context

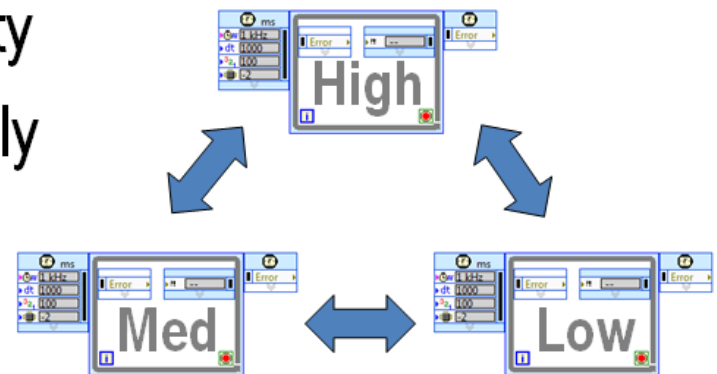
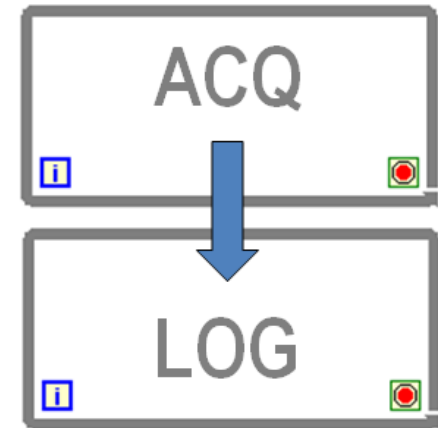


Inter-target: communication between multiple physical targets, often over a network layer



Defining Inter-process Communication

- Communication on same PC or Target
- Communicate between parallel processes or loops
- Offload data logging or processing to another CPU/Core/Thread within same VI/executable
- Loops can vary in processing priority
- Used to communicate synchronously and asynchronously



Inter-process Communication Options

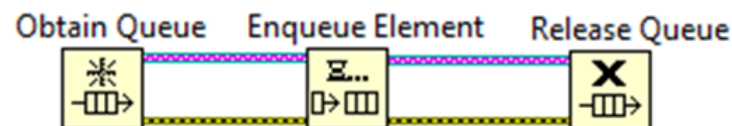
Shared Variables

Update GUI loop with latest value



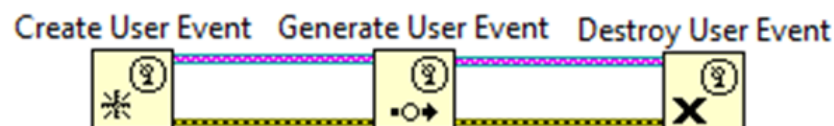
Queues

Stream continuous data between loops on a non-deterministic target



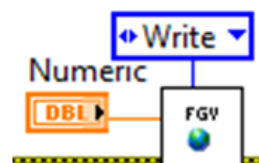
Dynamic Events

Register Dynamic Events to execute sections of code



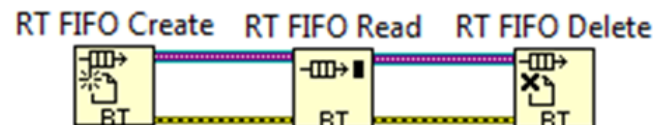
Functional Global Variables (FGV)

Use a non-reentrant subVI to protect critical data



RT FIFOs

Stream continuous data between time critical loops on a single RT target



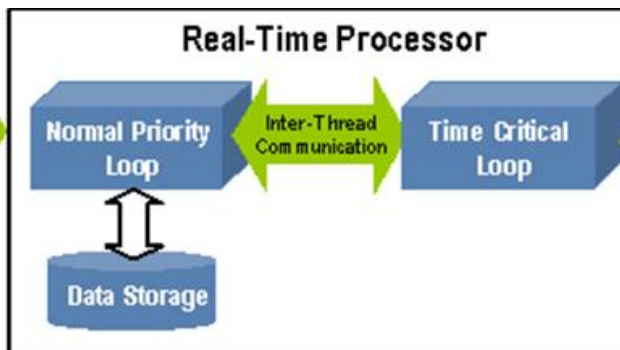
RT FIFOs vs. Queues

- Queues can handle string, variant, and other variable size data types, while RT FIFOs can not
- RT FIFOs are pre-determined in size, queues can grow as elements are added to them
- Queues use blocking calls when reading/writing to a shared resource, RT FIFOs do not
- RT FIFOs do not handle errors, but can produce and propagate them

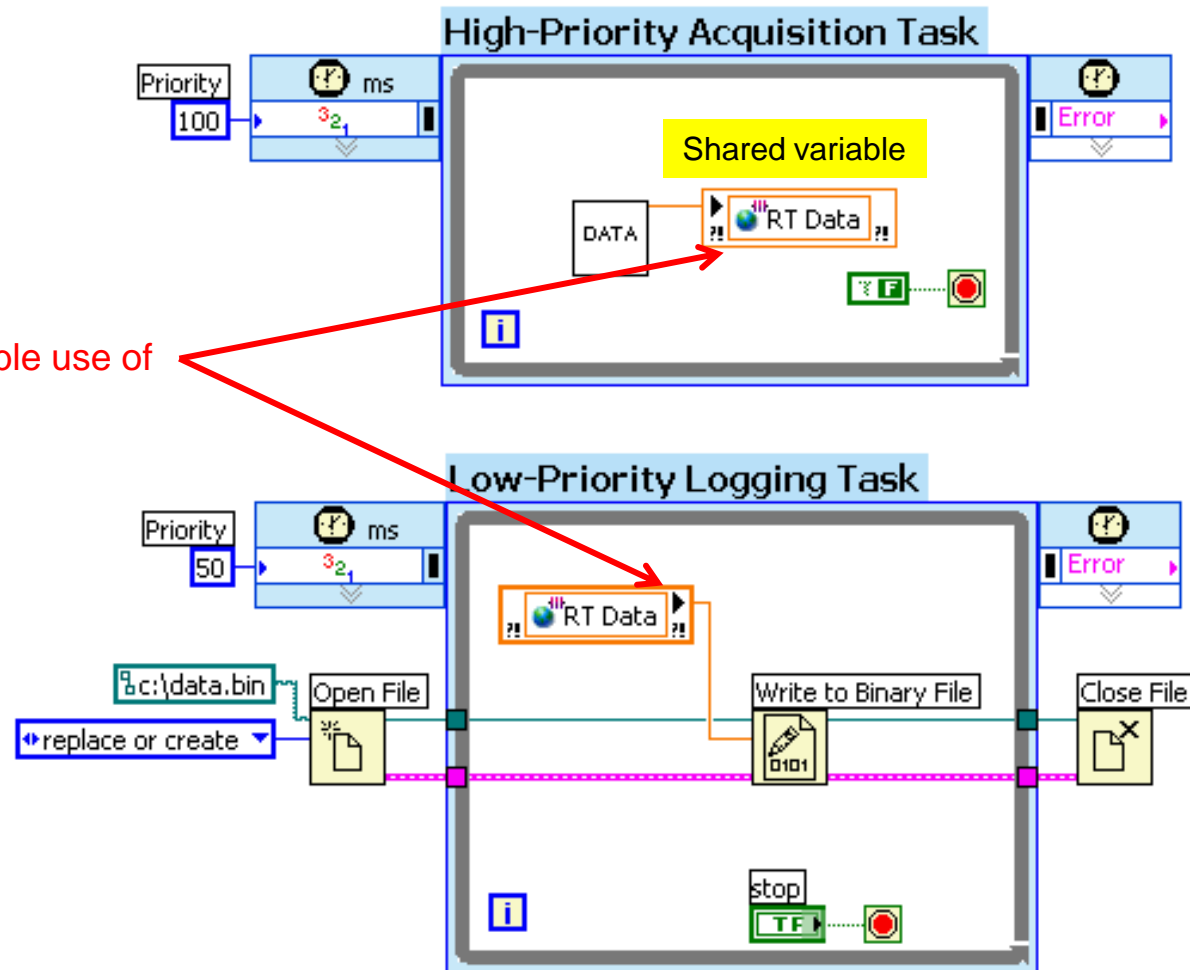
Key Takeaway:

RT FIFOs are more deterministic for the above reasons

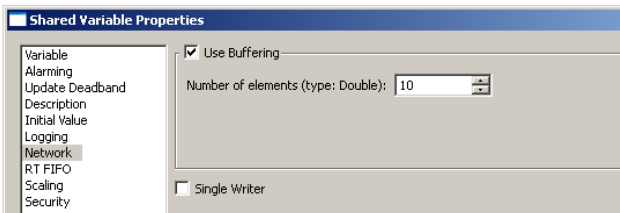
Deterministic communication between real-time threads with shared variables



Can enable use of
RT FIFO



Shared Variables: Can enable buffering (to avoid losing data)

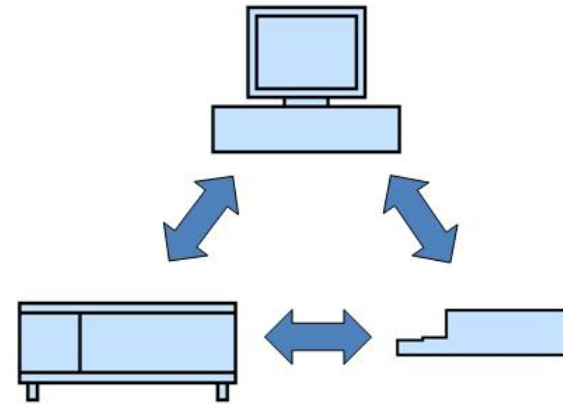


Single-Process Shared Variables and LabVIEW Real-Time FIFO

- In order to maintain determinism, a real-time application requires the use of a nonblocking, deterministic mechanism to transfer data from deterministic sections of the code, such as higher-priority timed loops and time-critical priority VIs, to nondeterministic sections of the code. When you install the LabVIEW Real-Time Module, you can configure a shared variable to use real-time FIFOs by enabling the real-time FIFO feature from the **Shared Variable Properties** dialog box. National Instruments recommends using real-time FIFOs to transfer data between a time-critical and a lower-priority loop. You can avoid using the low-level real-time FIFO VIs by enabling the real-time FIFO on a single-process shared variable.

Defining Inter-target Communication

- PC, RT, FPGA, Mobile Device
- Offload data logging and data processing to another target
- Multi-target/device application
- Network based



Inter-Target Communication Options

TCP/IP and UDP

Define low-level communication protocols to optimize throughput and latency

Shared Variables

Access latest value for a network published variable

Network Streams

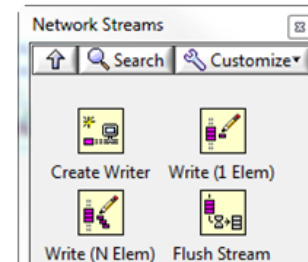
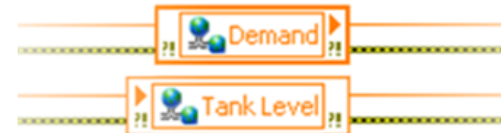
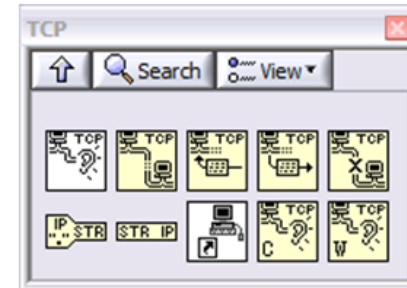
Point to Point streaming in LabVIEW with high throughput and minimal coding

Web UI Builder

Create a thin client to communicate with a LabVIEW Web Service

DMAs

Direct memory access between to different components of a system



NI & LabVIEW Embedded products







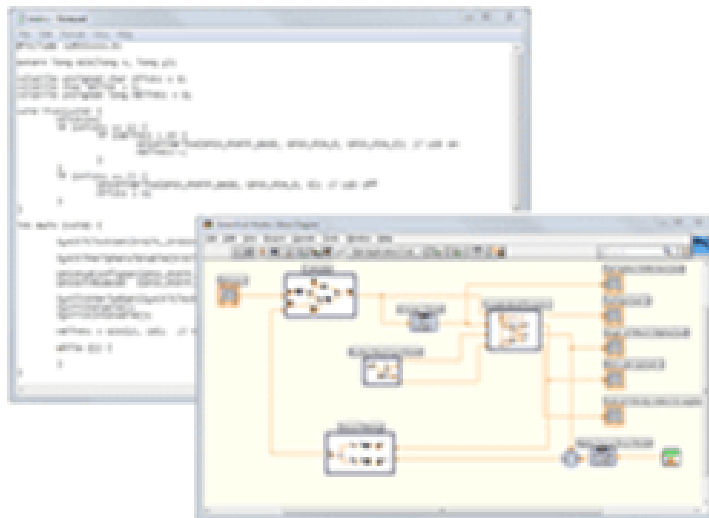
With the NI **LabVIEW C Code Generator**, you can port your algorithm designed using the LabVIEW programming environment to any processor of your choice.

LabVIEW Embedded

NI LabVIEW C Generator

From Algorithm to Embedded Target

 [E-mail this Page](#) [Configure Page for:](#)  [Print](#)  [PDF](#)  [Rich Text](#)







- Generate ANSI C code from LabVIEW VIs
- Compatible with 8-, 16-, and 32-bit microprocessors
- Use with any embedded OS or barebone

 [Download Eval](#)

LabVIEW Embedded

NI LabVIEW Embedded Module for ARM Microcontrollers Graphical Programming for ARM7, ARM9, and Cortex-M3

 [E-mail this Page](#) Configure Page for:  [Print](#)  [PDF](#)  [Rich Text](#)

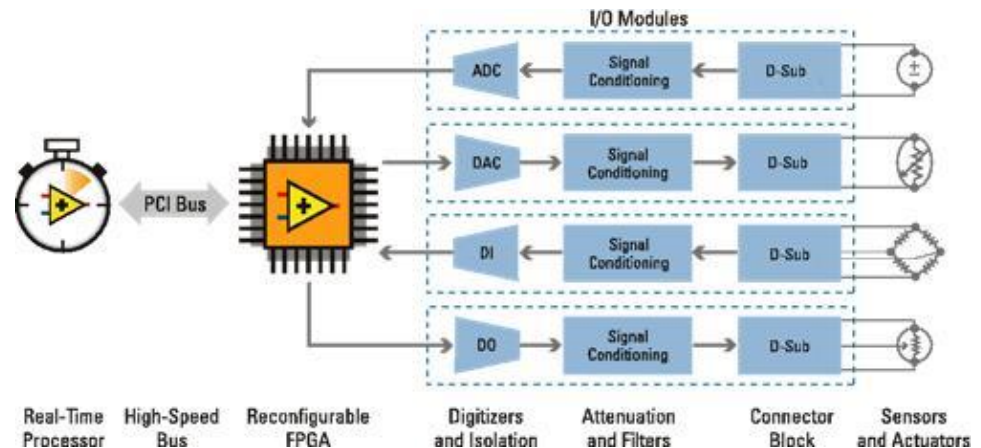


[\[+\] Enlarge Picture](#)

- Works with more than 260 ARM7, ARM9, and Cortex-M3 microcontrollers
- Integrated drivers for analog and digital I/O, PWM, TCP/IP, serial, I2C, and SPI
- Simulate your application on the desktop including peripheral I/O for stimulus/response
- Simple API for integrating C code with graphical code for a hybrid programming approach

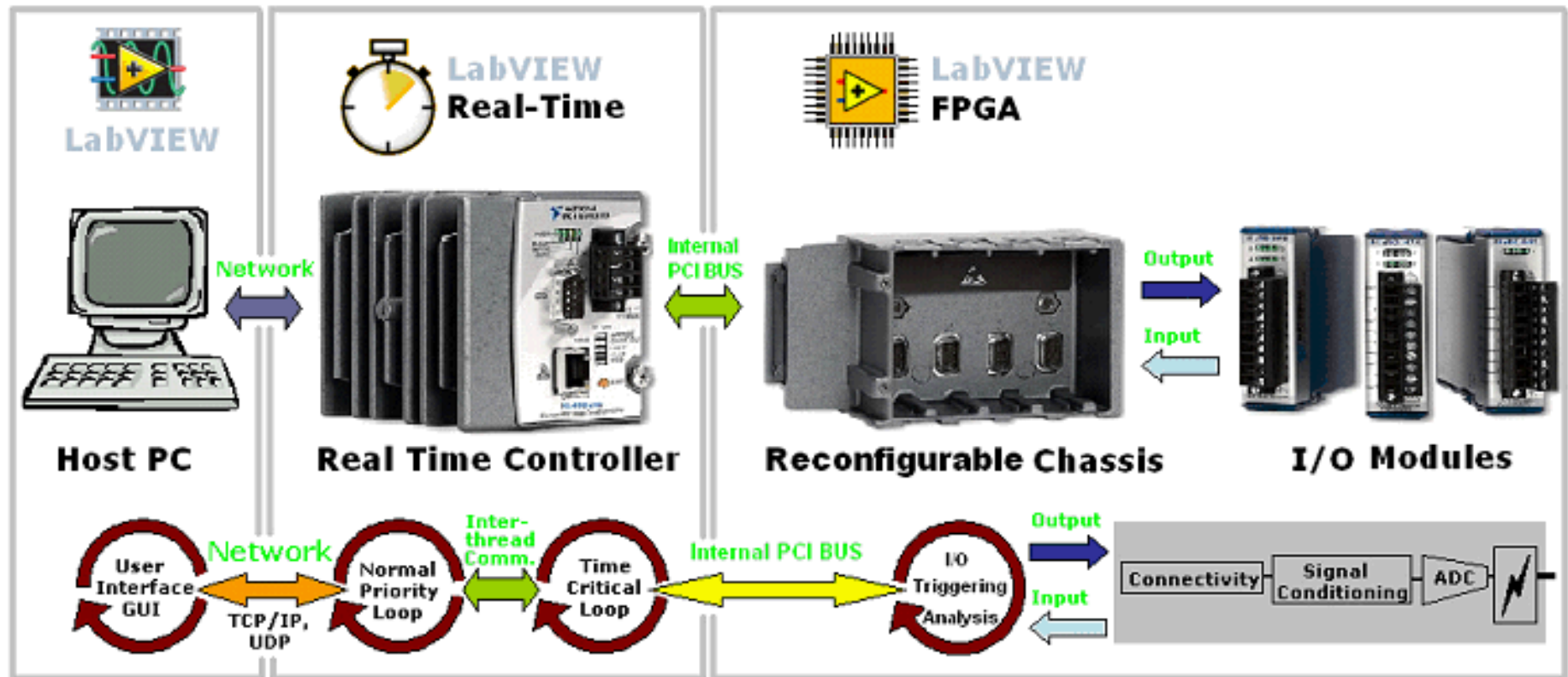
LabVIEW Embedded system application development

- Developing the **LabVIEW FPGA application** for Input/Output (I/O), timing, synchronization, high speed control and signal processing.
- Developing the **LabVIEW Real-Time application** for deterministic floating point analysis and control as well as communication with a networked host computer.
- Developing the **LabVIEW for Windows application** for graphical user interfaces, supervisory control and data logging.

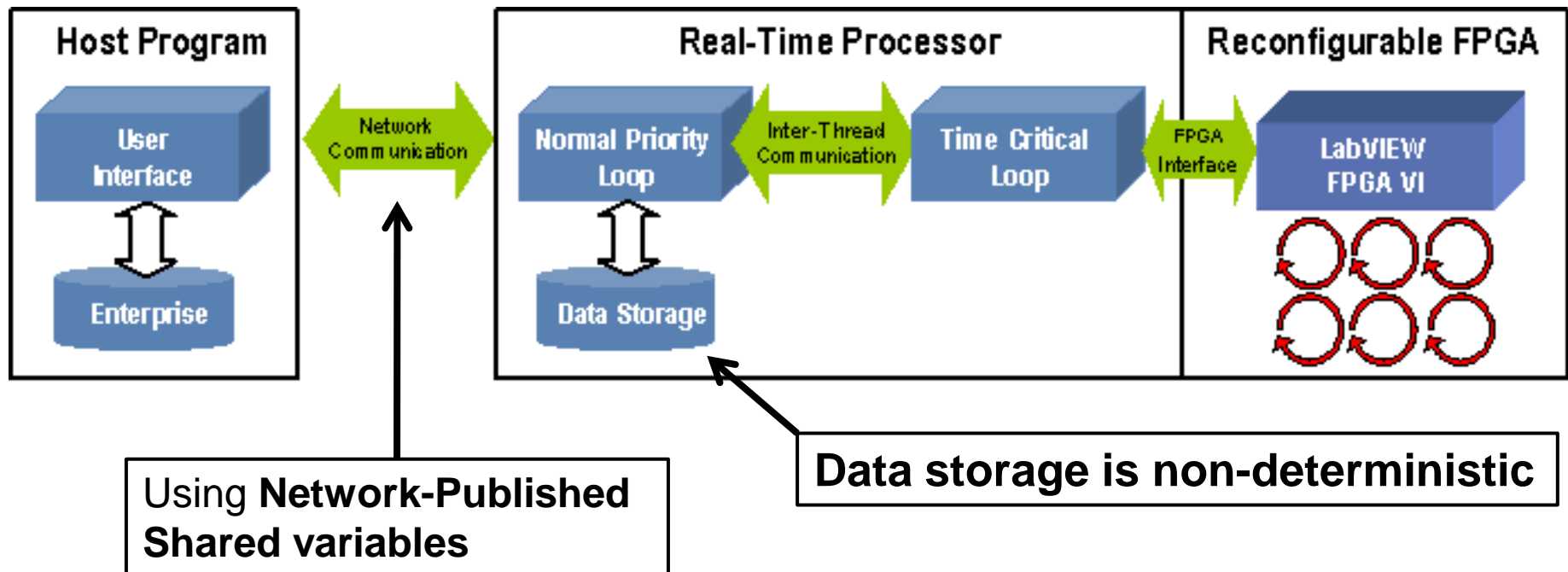


NI CompactRIO Reconfigurable Embedded System

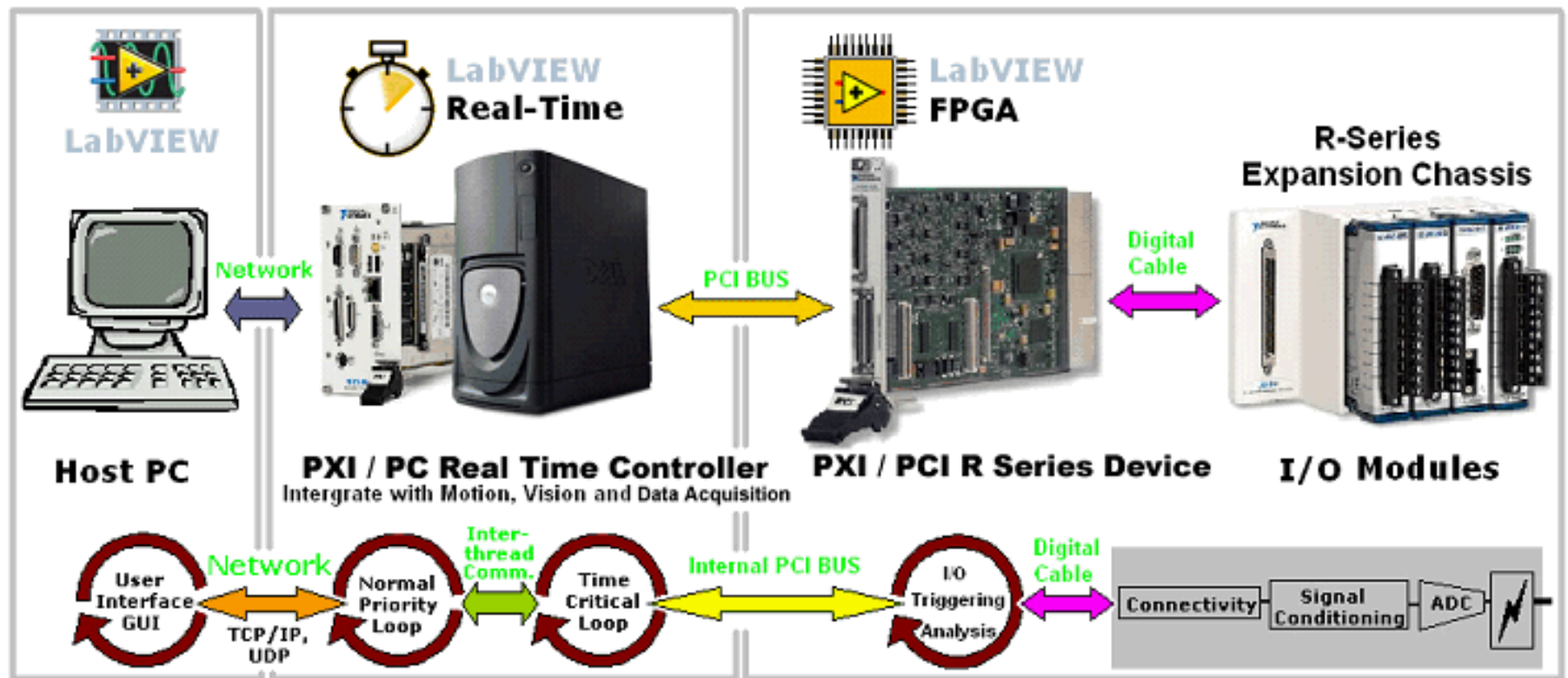
Note that most communication protocols are non-deterministic, so, in order to ensure deterministic performance in your time-critical code, you should not perform communication from within the time-critical VI. Transfer the data to a normal priority VI also running on the RT side to perform your communication.



Architecture for Advanced CompactRIO Applications



R-series Intelligent DAQ System Embedded System



What to avoid in high-priority code?

- Operations that allocated memory:
 - Array functions such as Build array, Append array
 - String manipulation
- Non-deterministic functions:
 - File I/O Operations
 - Networking functions
 - Some I/O Driver calls

Interrupt-Driven Programming

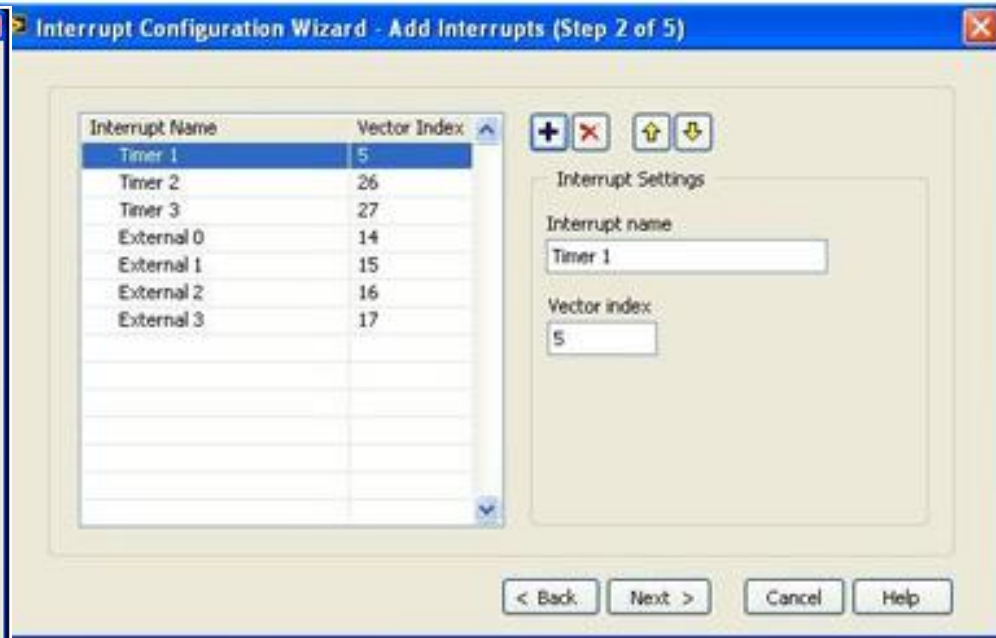
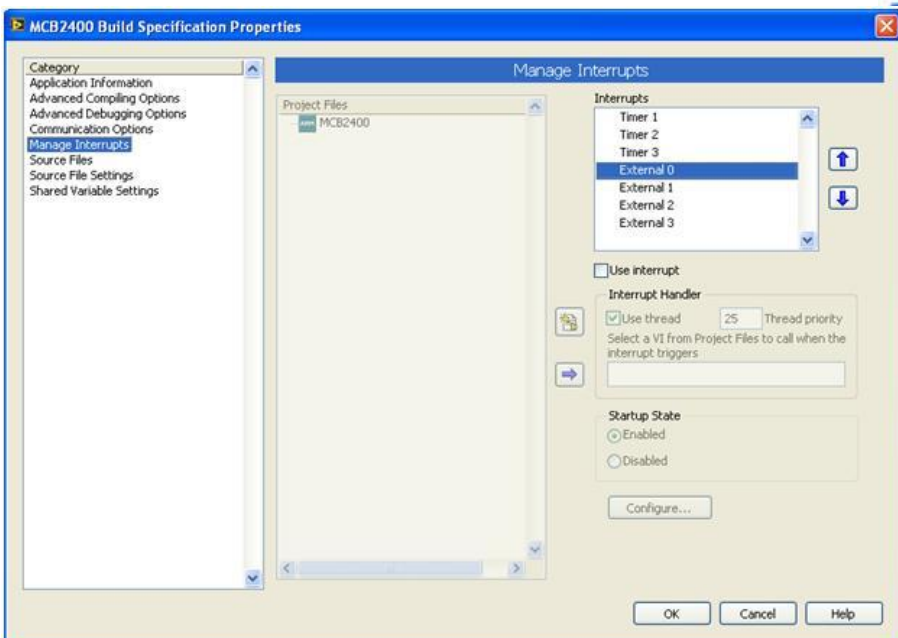
- In interrupt-driven systems software is designed such that when a registered event, such as a timer, is received, a response is fired to respond to this event.
- There are two components of any interrupt-driven system: **the interrupt** and **the interrupt handler**.
- An interrupt is a signal that is generated by hardware, which indicates an event has occurred that should halt the currently executing program.
- Interrupt handlers (also referred to as interrupt service routines - ISRs) are portions of code that are registered with the processor to execute once a particular interrupt has occurred. Once the processor is aware of an interrupt, it halts the currently executing process, performs a context switch to save the state of the system, and executes the interrupt handler. Once the interrupt handler code has executed, the processor returns control to the previously running program.

Interrupt-Driven Programming II

- For **Interrupt-Driven Programming** hardware events are detected and responded to, compared to **event driven programming** (on a PC) where user interface events trigger some code to be executed

LabVIEW: Interrupts on ARM processors

- The **Interrupt Configuration Wizard** makes it easier for you to add hardware interrupt support to hardware targets supported in LabVIEW.
- Using the project build specifications, select an interrupt and a LabVIEW VI to act as the interrupt handler in your application
- Use the **Manage Interrupts** page of the ARM Build Specification Properties dialog box to enable interrupts and create and configure ISR VIs.
- Using a series of dialogs, you can customize any available hardware interrupt in LabVIEW



Interrupts for Data Acquisition

- In general, there are three approaches to acquiring data from an external device or synchronizing communication between devices. These three approaches are described as follows:
- Polling – This method involves periodically reading the status of the device to determine whether the device needs attention.
- **Direct Memory Access (DMA)** – A dedicated processor, the DMA controller, transparently transfers data from the device to computer memory, or vice versa.
- Interrupts – the device is configured to interrupt the processor whenever the device requires attention.