EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

Základy práce a programovania CompactRIO systémov pre meracie aplikácie

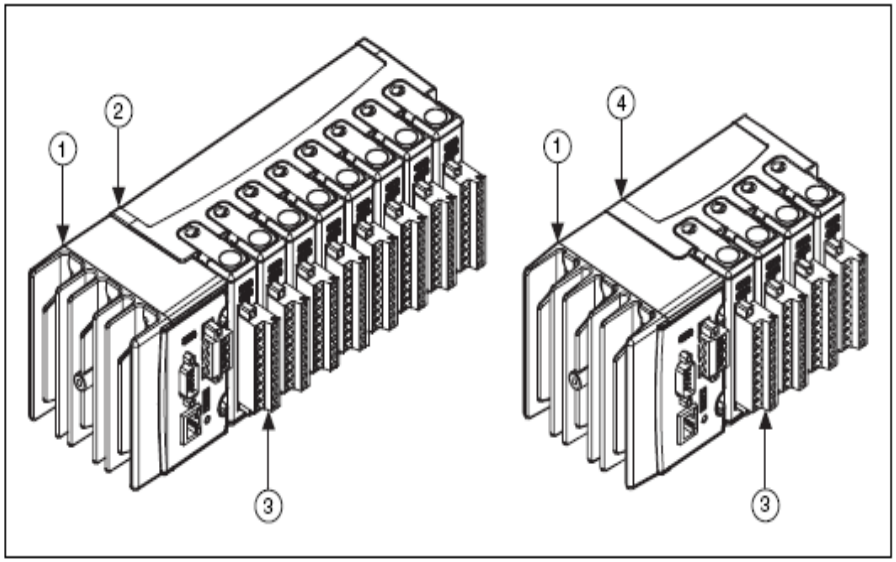
Ing. Mgr. Márk Jónás (ANV, s.r.o.)

29.6.2012

Tato prezentace je spolufinancována Evropským sociálním fondem a státním rozpočtem České republiky.

Obsah seminára

1. Úvod do CompactRIO platformy
2. Konfigurácia zariadenia
3. Architektúra aplikácií
4. Scan mode snímania dát
5. Práca s programovateľným hradlovým poľom
6. Vývoj Real-time aplikácií
7. Vývoj aplikácie na riadiacom počítači
8. Transfer dát



1 Controller
2 Eight-Slot Reconfigurable Embedded Chassis (cRIO-9102 and cRIO-9104)
3 I/O Module
4 Four-Slot Reconfigurable Embedded Chassis (cRIO-9101 and cRIO-9103)

Lekcia 1

ÚVOD DO COMPACTRIO PLATFORMY

A. CompactRIO

Programmable Automation Controller (PAC)

- Analog and digital I/O
- Floating-point processing
- Seamless connectivity

Embedded—A component in a larger system.

Headless—Operates without a user interface and when the host computer is unavailable

Rugged—50 g shock, –40 to 70 ° C

B. Applications

- Machine Control
  - High-speed motion control
  - Real-time signal processing and control of power electronics, hydraulic systems
  - Custom motion and vision inspection, material handling

B. Applications

Machine Condition Monitoring

- Bearing order analysis, lubrication monitoring

Mobile/portable DSA

- Noise, vibration, dynamic signal analysis, acoustics

Distributed Acquisition

- Central controller with distributed I/O nodes over Ethernet/wireless

B. Applications

Automotive & Aerospace

In-Vehicle Data Acquisition

- Automobiles, motorcycles, recreational vehicles, research aircraft, trains

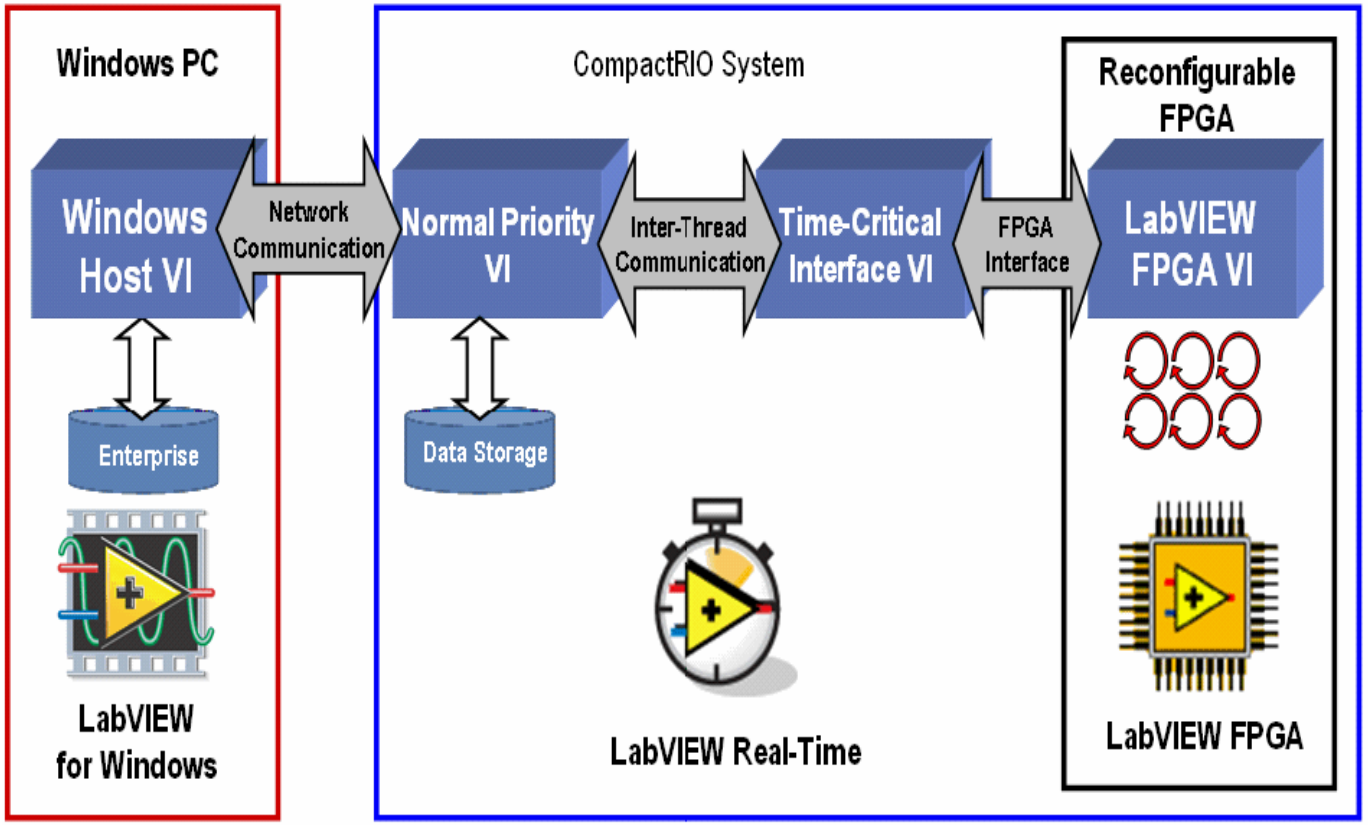
Engine and ECU test cells

- HIL testing of engines and engine controllers, sensor simulation using FPGA

Rapid Control Prototyping

- Automotive/aerospace control prototyping

C. Application Architecture



Alternate Architectures

- Windows host VI and RT controller VI
- Standalone RT controller VI
- Stand-alone FPGA VI
- Stand-alone FPGA VI and RT controller VI

C. Application Architecture

FPGA Functions

- I/O, Hardware-based timing and triggering
- Low-level signal processing

C. Application Architecture

Real-Time Controller Functions

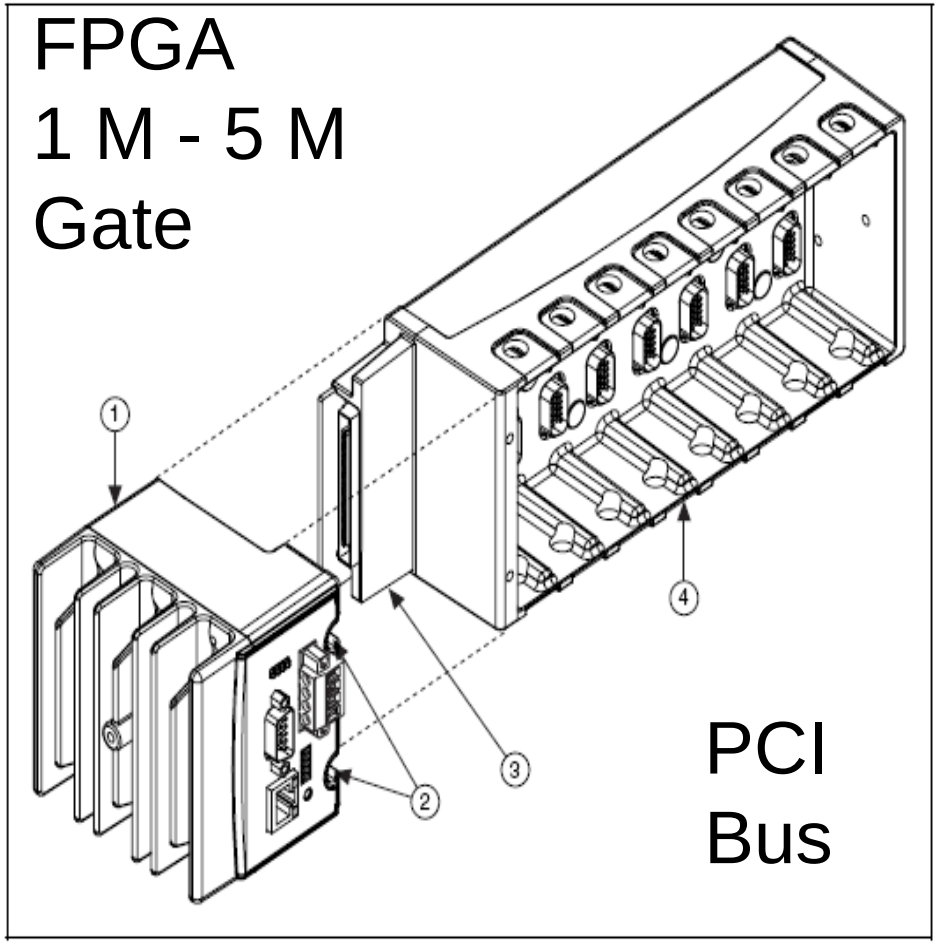
- FPGA interaction
  - Configuring
  - Communicating data
  - Controlling
- Processing data
- Logging data
- Communications with remote PC host

C. Application Architecture

Windows PC Functions

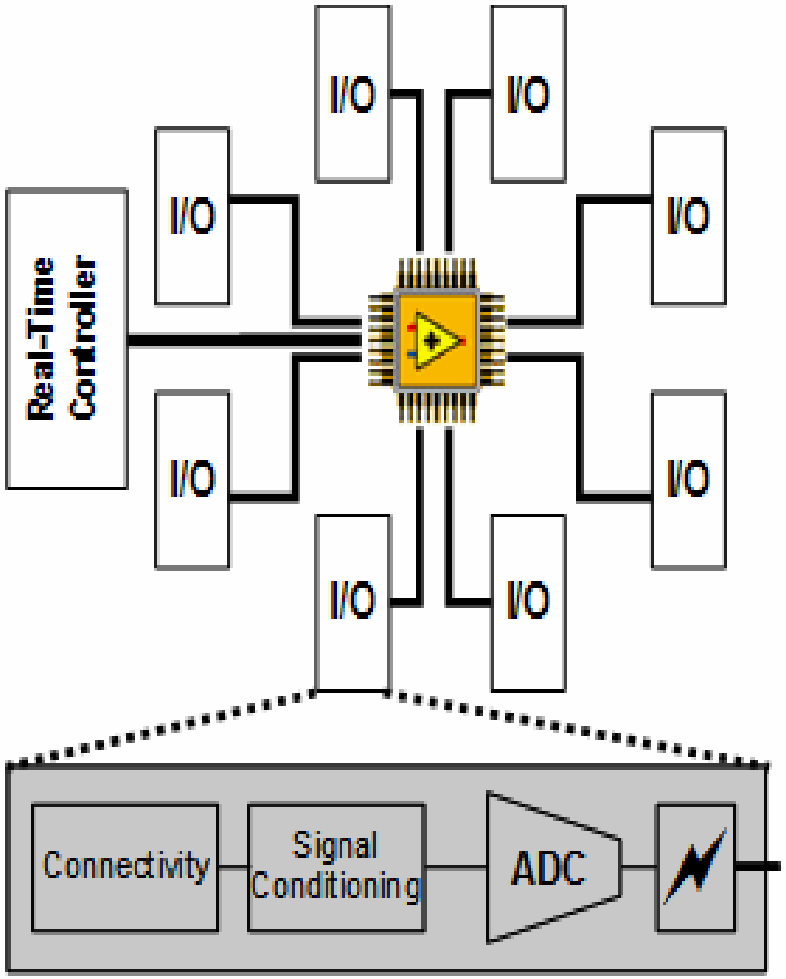
- Logging data
- Accessing databases
- Integrating with enterprise systems
- Providing a human-machine (User) interface (HMI) and display
- Supporting supervisory control

D. Components

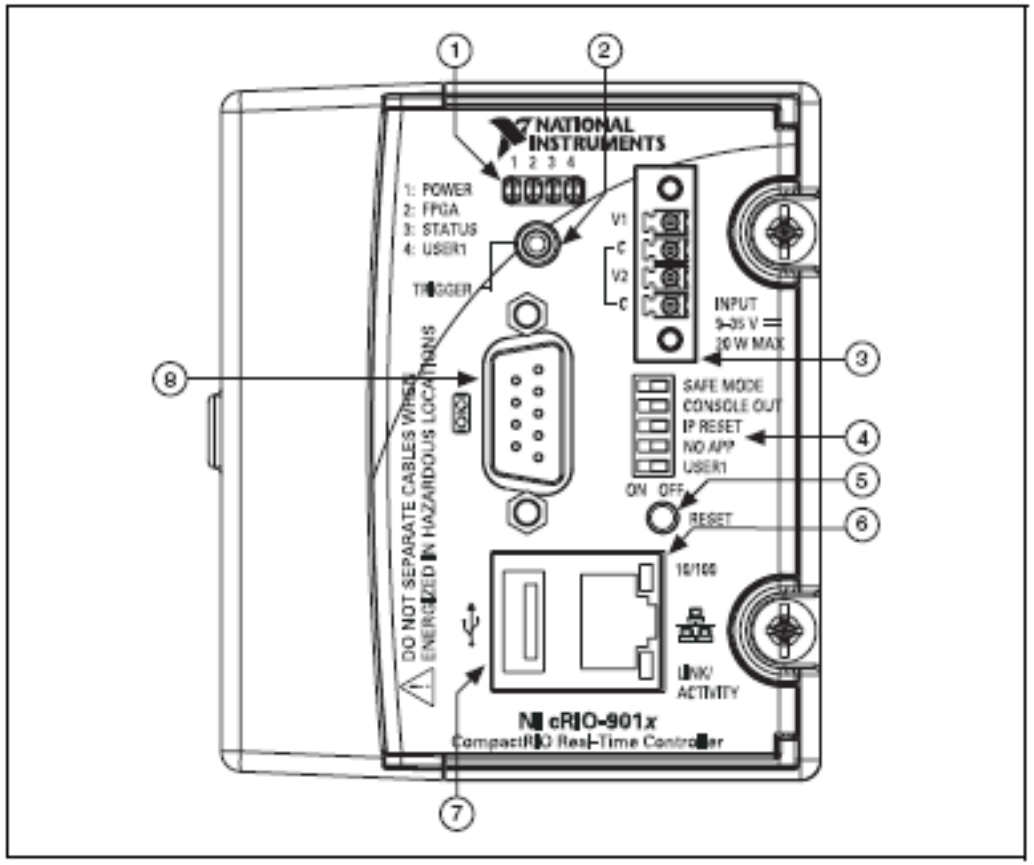


1	Controller	3	Controller Slot
2	Captive Screws	4	Reconfigurable Embedded Chassis

D. Components



D. Components



1	LEDs	5	Reset Button
2	SMB Connector	6	RJ-45 Ethernet Port
3	Power Connector	7	USB Port
4	DIP Switches	8	RS-232 Serial Port

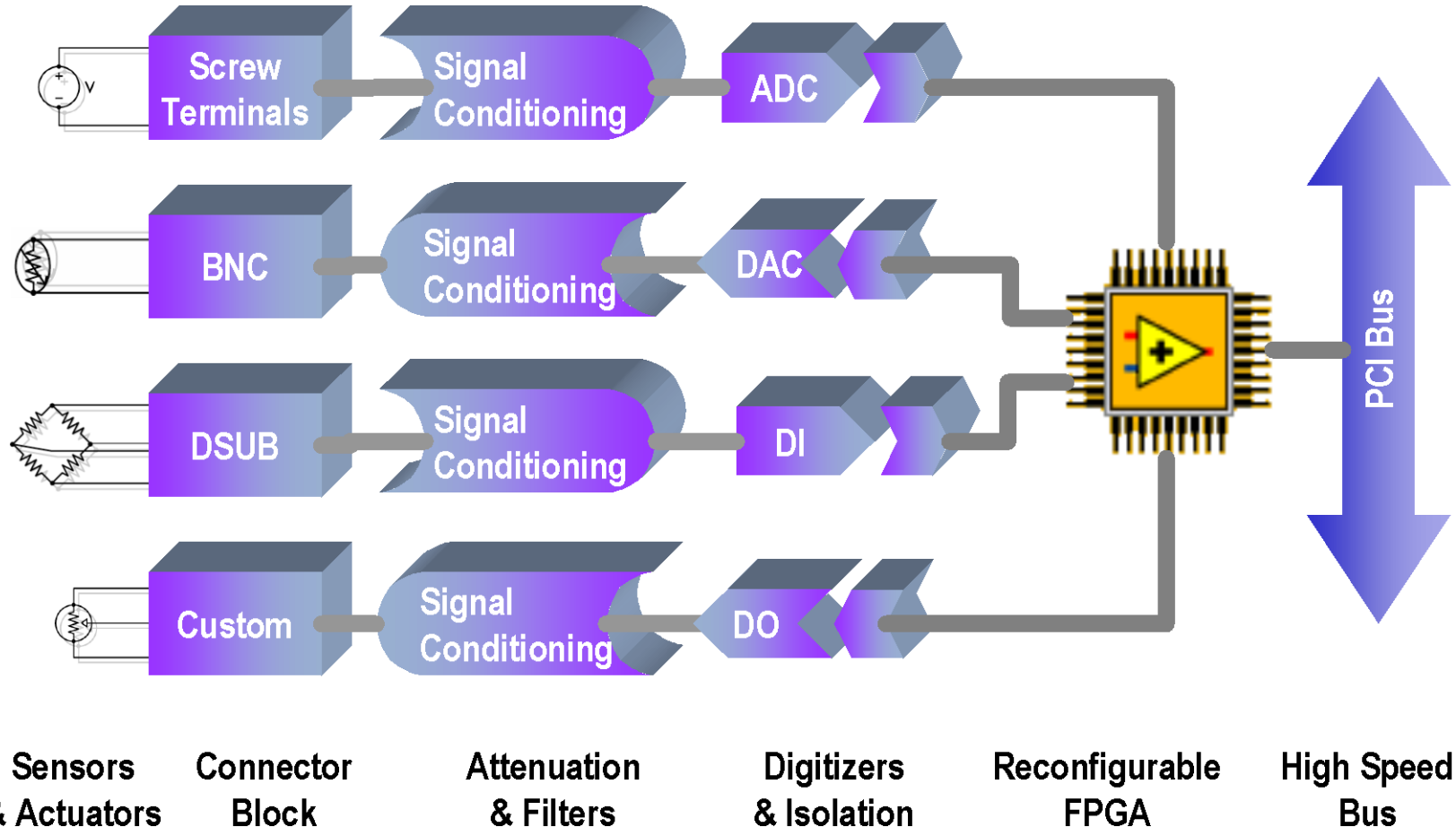
D. Components

9012 RT Controller Memory

- Nonvolatile memory stores information even when not powered
  - Used for drivers, storing VIs, and the Shared Variable Engine
- DRAM is volatile memory and will be lost on power down
  - Used for allocating memory and running applications

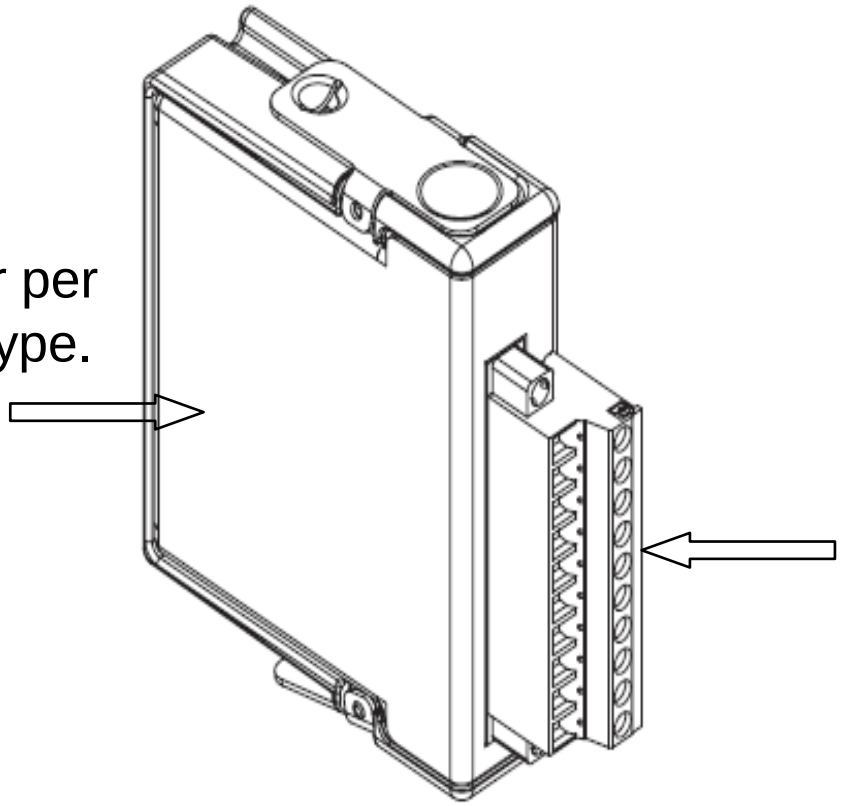
Nonvolatile	DRAM
128 MB	64 MB

D. Components



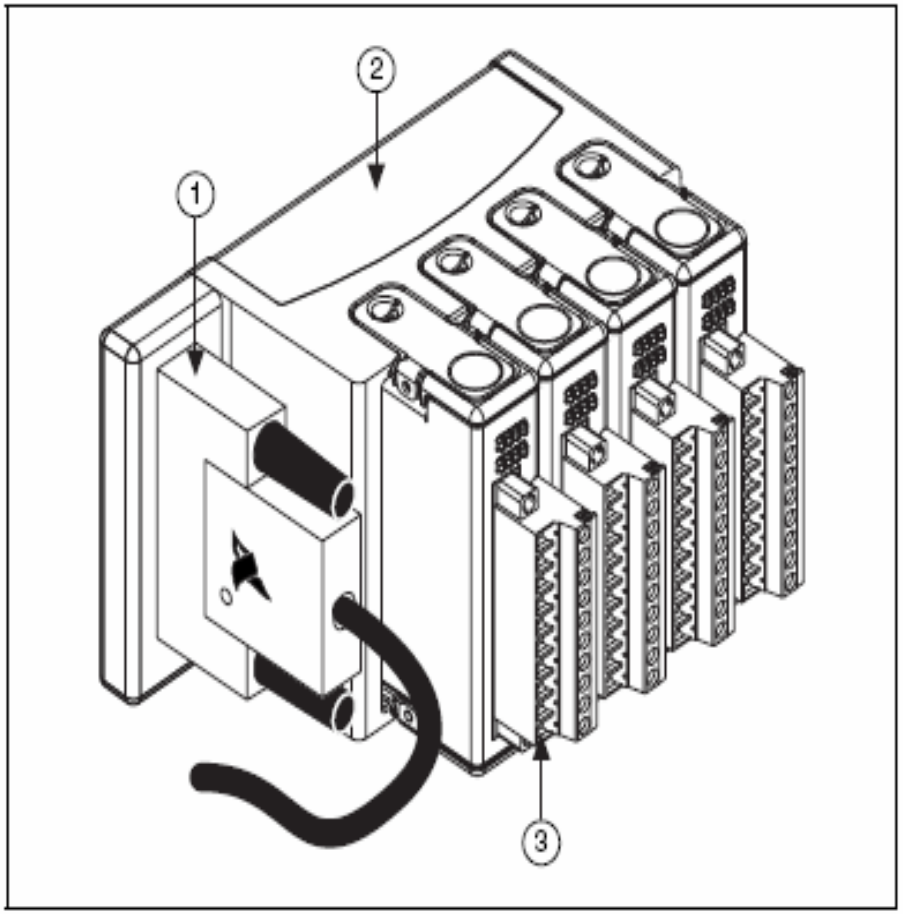
D. Components

Custom circuitry for per the measurement type.



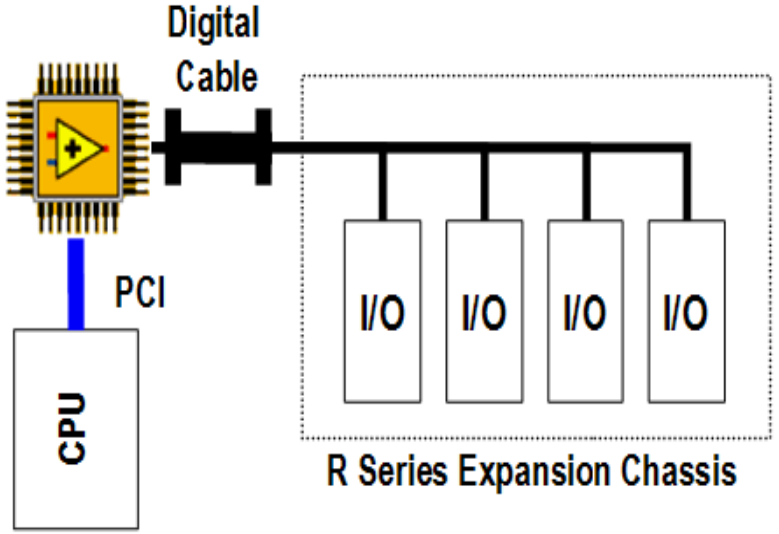
Direct connection to Industrial sensors and actuators

E. R Series Expansion System



1 Cable 2 R Series Expansion Chassis 3 I/O Module

- Used with PC-based R Series devices



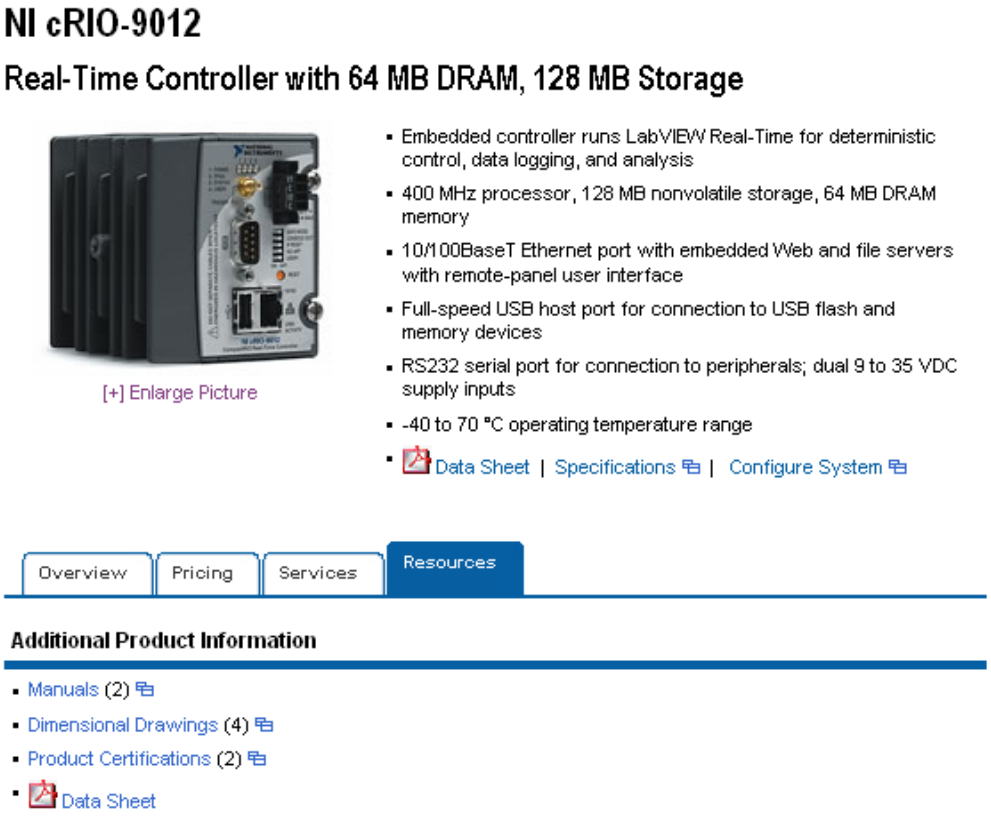
G. Accessories

- Industrial enclosures
- Flat-panel touch-screen industrial computers
- Power supplies

Find Specifications

- Hardware ships with
  - User Manual
  - Getting Started Guide
  - Installation Guide
- Can search for updated manuals online
  - ni.com/manuals
- All product listings on ni.com contain links

  Data Sheets and manuals



H. Operation

Power-on Self Test

Power-on Self Test (POST)

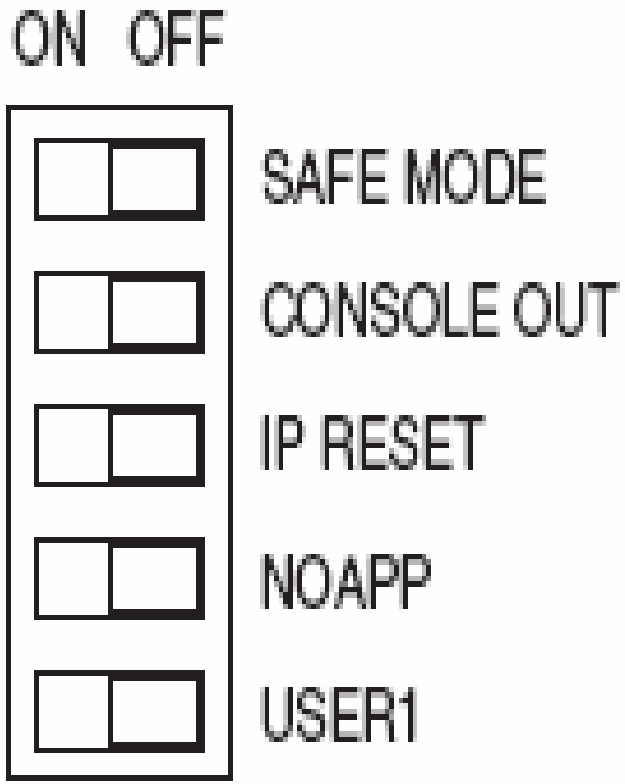
1. Power LED – ON
2. Status LED – ON
3. User1 LED – ON

POST Complete

1. Power LED – ON
2. Status LED – OFF
3. User1 LED – OFF

H. Operation

DIP switches



H. Operation

Safe Mode Switch

- Determines whether the embedded LabVIEW Real-Time engine launches when the controller boots.
- Normal operation is the OFF position.
- If ON only the essential services required for updating configuration and installing software are loaded.
- Use ON if the software on the controller is corrupted.
- ON required to reformat the drive
- Refer to MAX Help for more about installing software and reformatting the drive.

H. Operation

Console Out Switch

- Serial Communication
- Serial-port Configuration:
  - 9,600 bits per second
  - Eight data bits
  - No parity
  - One stop bit
  - No flow control
- Keep the Console Out switch in the OFF position during normal operation.

H. Operation

IP Reset Switch

- ON position and reboot resets the IP address to 0.0.0.0.
- If on your local subnet and the IP Reset switch is ON, the controller appears in MAX with IP address 0.0.0.0.
- Configure a new IP address for the controller in MAX.
- ON unlocks a controller that was locked in MAX.
- Keep the IP Reset switch in the OFF position during normal operation.

H. Operation

- Restore Defaults
  - If the controller is not able to communicate with the network
  - Changes the IP address, subnet mask, DNS address, and gateway to 0.0.0.0
  - Does not affect power-on defaults, watchdog settings, or VIs
- To restore defaults:
  - Move the IP Reset DIP switch to the ON position.
  - Push the Reset button to cycle power to the controller. The Status LED flashes once, indicating that the controller IP address is unconfigured.
  - Move the IP Reset switch to the OFF position.

H. Operation

No App Switch

- ON prevents a LabVIEW startup application from running when the controller powers on.
- To permanently disable the application from running at power up, disable it in LabVIEW.
- To run an application when the controller powers on, push the No App switch to the OFF position, and configure the application in LabVIEW to launch when the controller powers on.

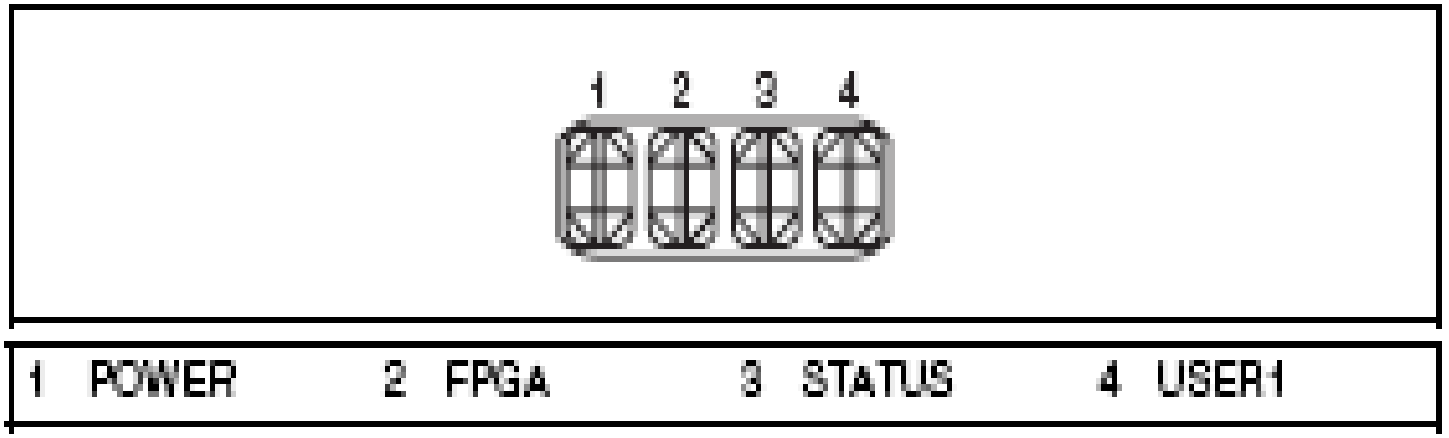
H. Operation

User1 Switch

- User defines the behavior of the User1 switch with the
  RT Read Switch VI in a LabVIEW Real-Time embedded application.

H. Operation

9012 Controller LEDs



H. Operation

- Power LED – Lit indicates the power supply is adequate, and the controller is supplying power to the CompactRIO system.
- FPGA LED – Programmable for application debugging or status. Refer to LabVIEW Help for information about programming.
- User1 LED – Programmatically definable to meet the needs of your application in the RT VI.

H. Operation

Status LED

- Off during normal operation.
- Indicates errors by flashing certain patterns:
  - Slow, continuous flashing. The controller is unconfigured. Use MAX to configure.
  - 2 flashes – Software error. Reinstall software. Use MAX Help.
  - 3 flashes – Safe Mode DIP switch is in the ON position.
  - 4 flashes – The controller software has crashed twice without rebooting or cycling power between crashes. Controller may be out of memory. Review your RT VI.
  - Continuous flashing – Unrecoverable error. Contact NI.
  - Continuously lit – The flash memory card is corrupt. Reformat the hard drive on the controller. Refer to MAX Help.

Lekcia 2

KONFIGURÁCIA ZARIADENIA

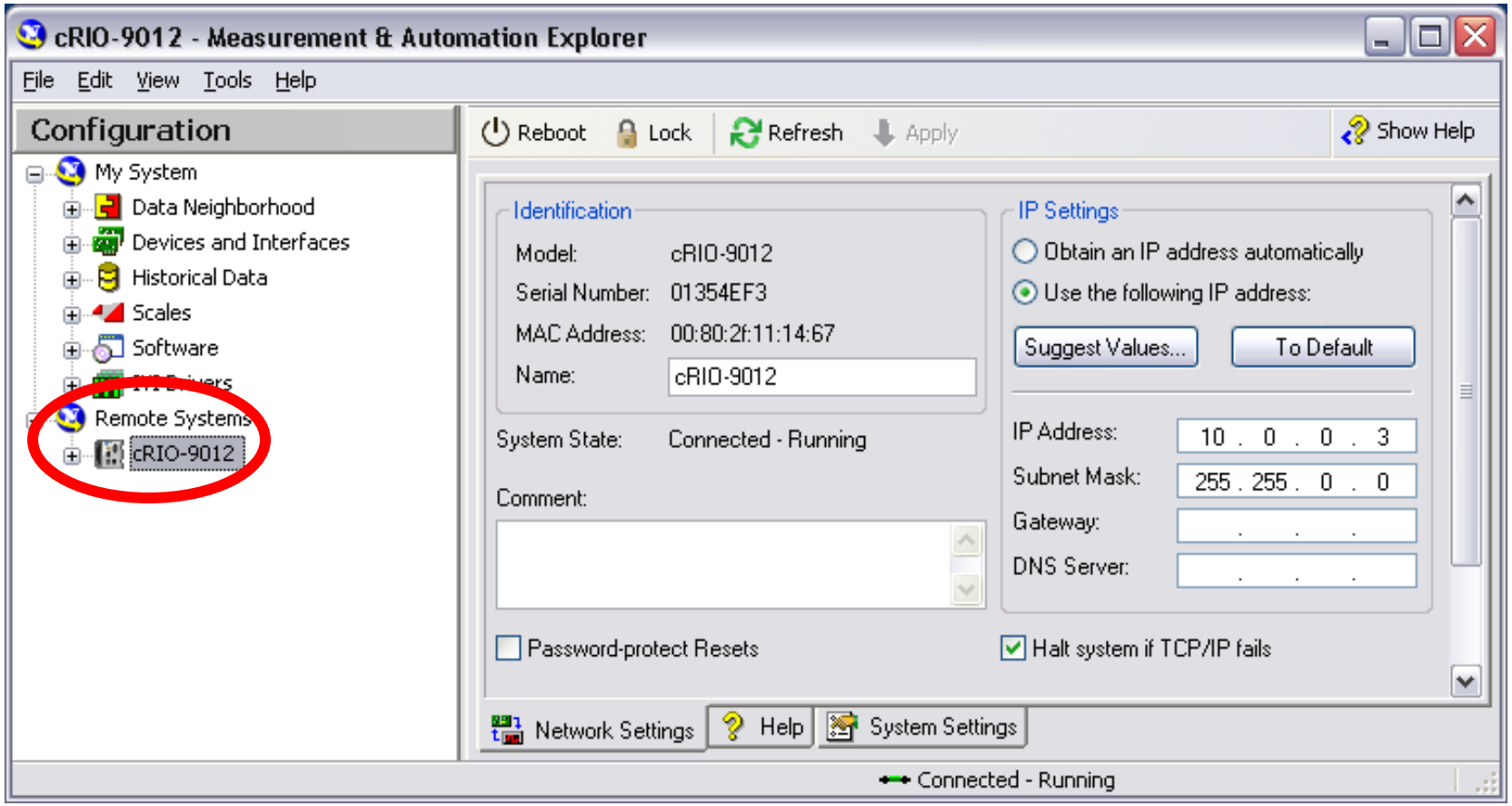
Konfigurácia zariadenia

- Detect the Remote Target
- Configure Network Settings
- View Devices and Interfaces
- Add/Remove Software

A. Detect Remote Target

- Install NI-RIO software from the shipping CD.
- Connect CompactRIO to PC with crossover Ethernet cable or to a network.
- Power up CompactRIO.
- Not configured CompactRIO – Status light blinks slowly.
- Open MAX (Start»Programs»National Instruments).
- Remote Systems in MAX– Devices connected over the Ethernet.

A. Detect the Remote Target

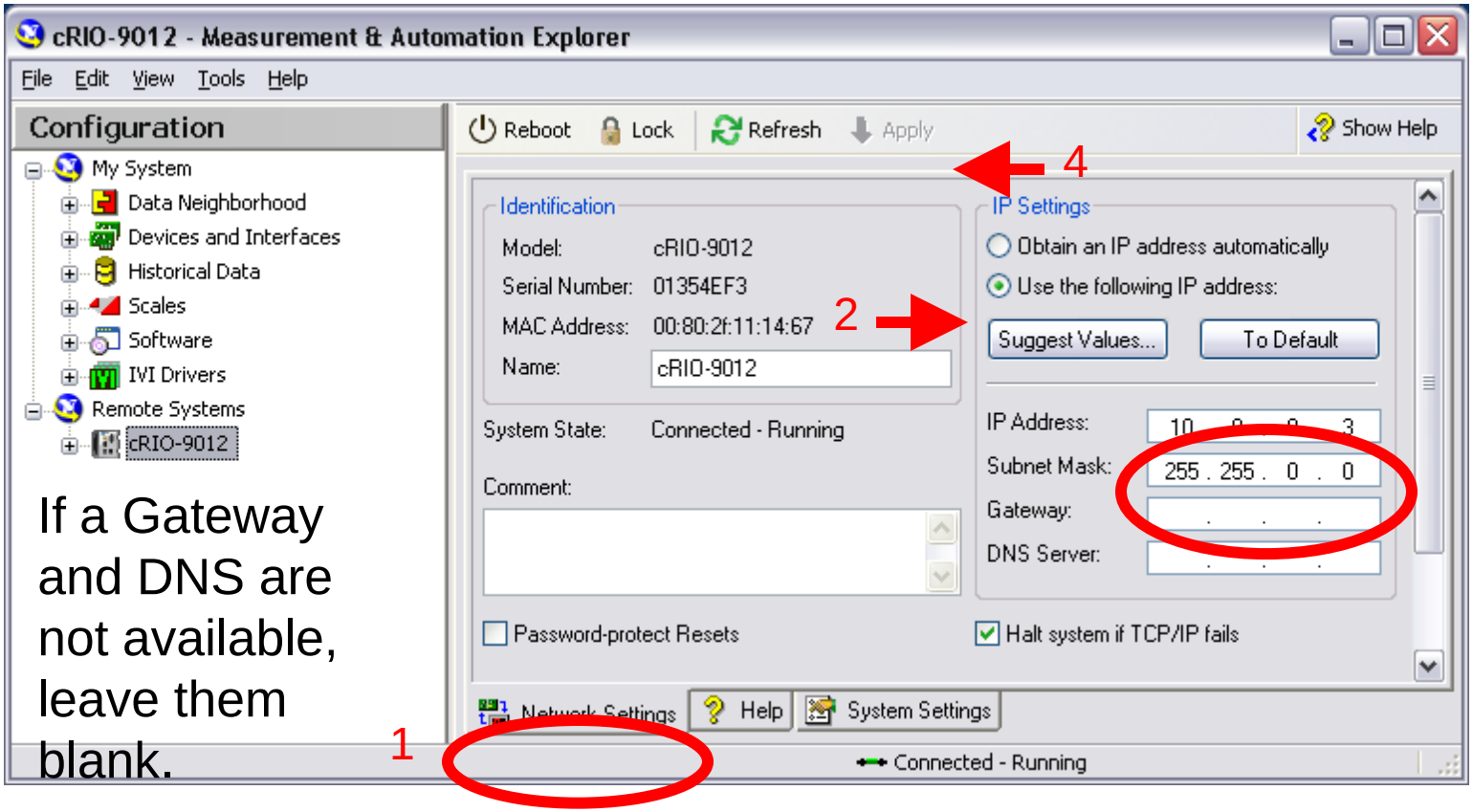


B. Configure Network Settings

- IP Address
- Subnet Mask
- Gateway
- DNS Address

B. Configure Network Settings

- Static IP Address



B. Configure Network Settings

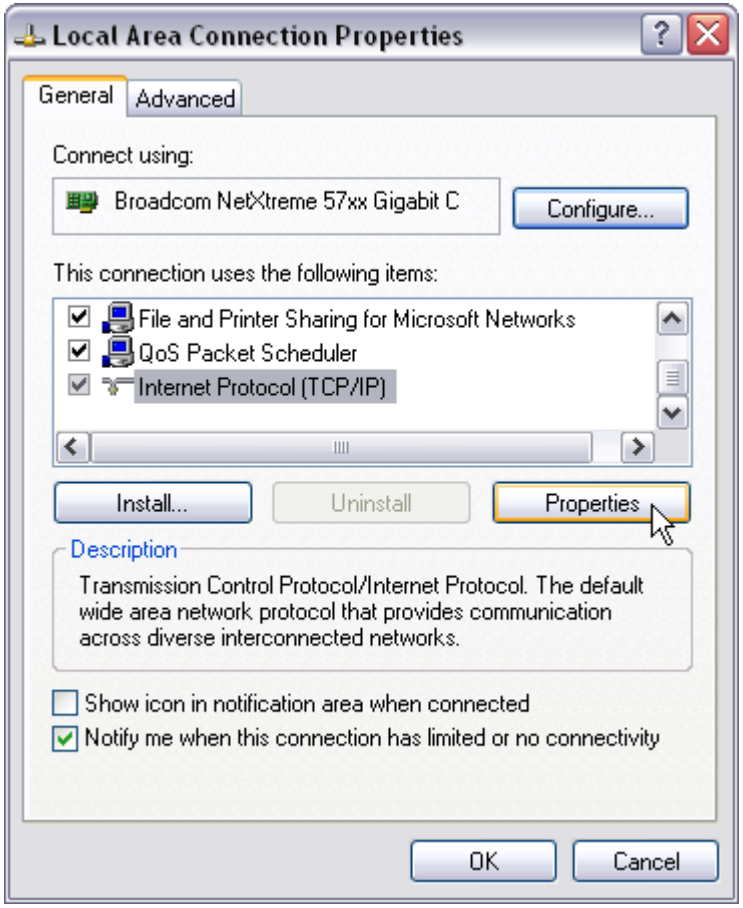
Automatic IP Address from DHCP Server

- Select “Obtain IP address from DHCP server”
- Click Apply
- Allocate on each reboot. New address might not the be the same. You must check the address frequently.
- To avoid this, use a static IP.
- Some DHCP servers are not compatible. Returns 0.0.0.0 after three failed attempts.

B. Configure Network Settings

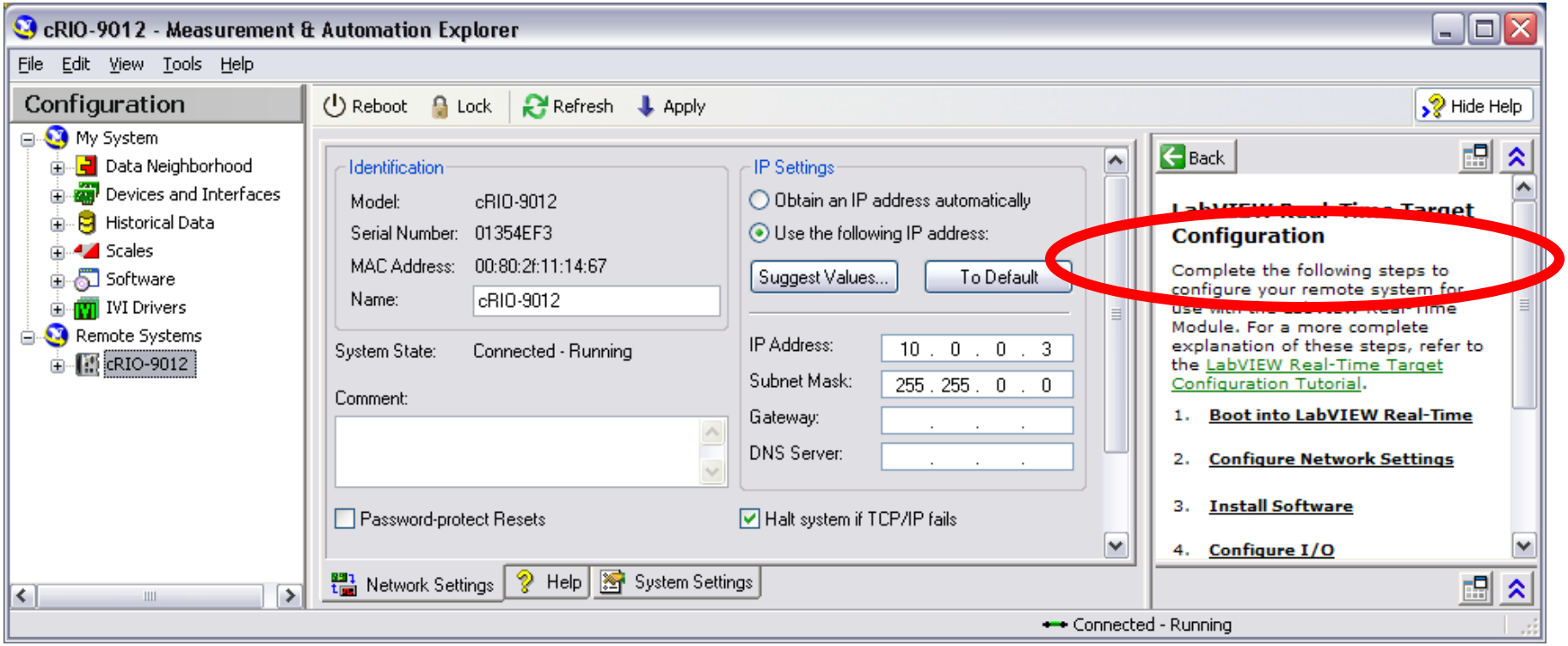
Static IP Address

- Crossover Cable requires PC and CompactRIO set to Static IP Address
  - Control Panel»Network Connections
  - Local Area Connection» Internet Protocol»Properties



B. Configure Network Settings

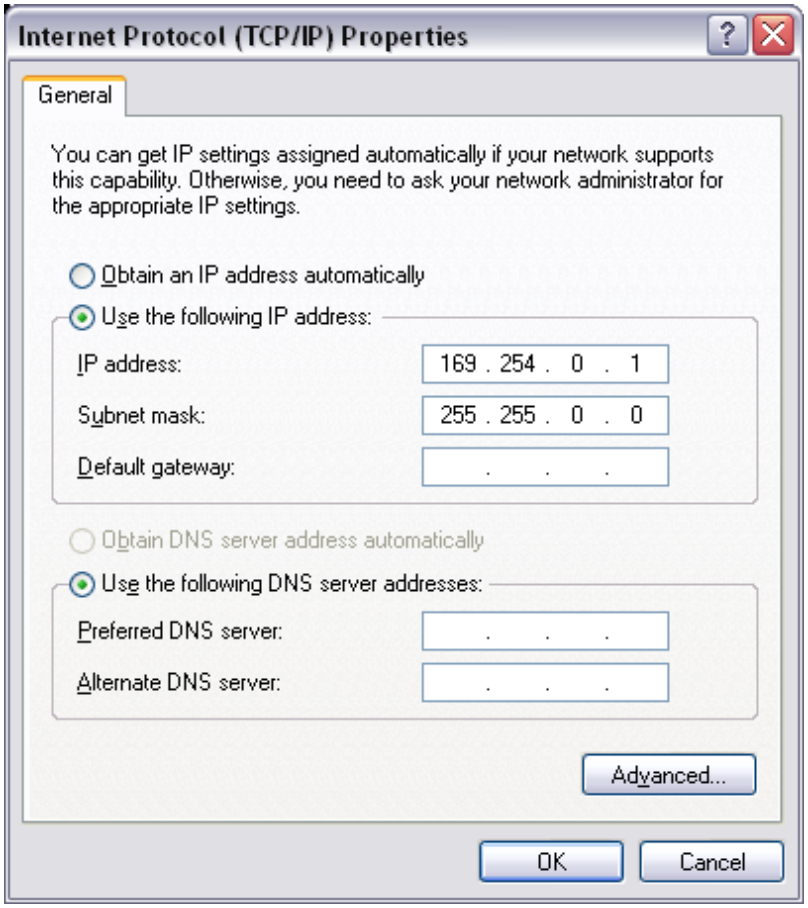
Display Configuration Tree, Network Settings, Help Simultaneously



B. Configure Network Settings

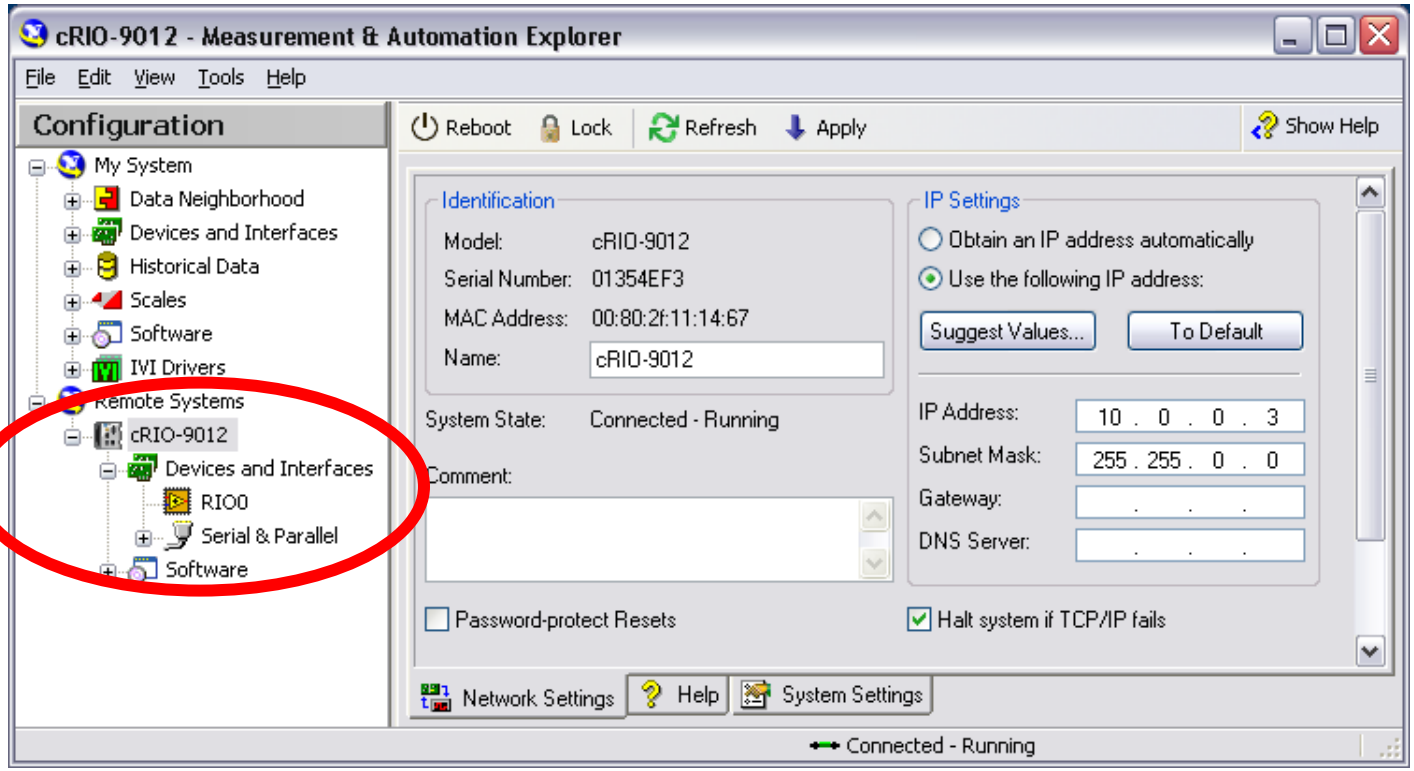
Static IP Address with Crossover

- Set IP to desired address



C. View Devices and Interfaces

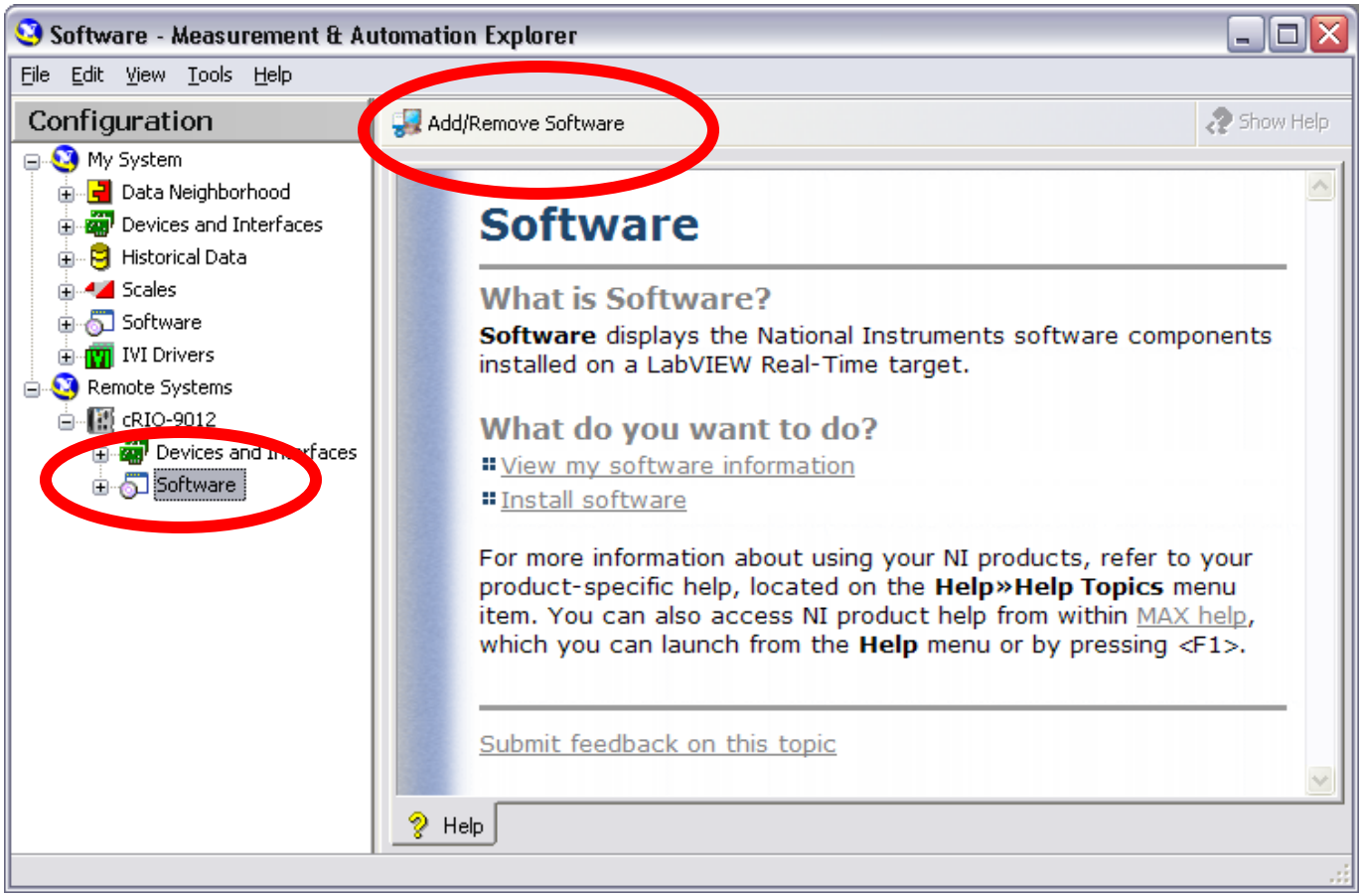
Expand Devices and Interfaces to view the chassis



C. View Devices and Interfaces

- If you do not see the chassis under devices and interfaces:
  - Check the hardware connections
  - Refresh the Configuration tree <F5>

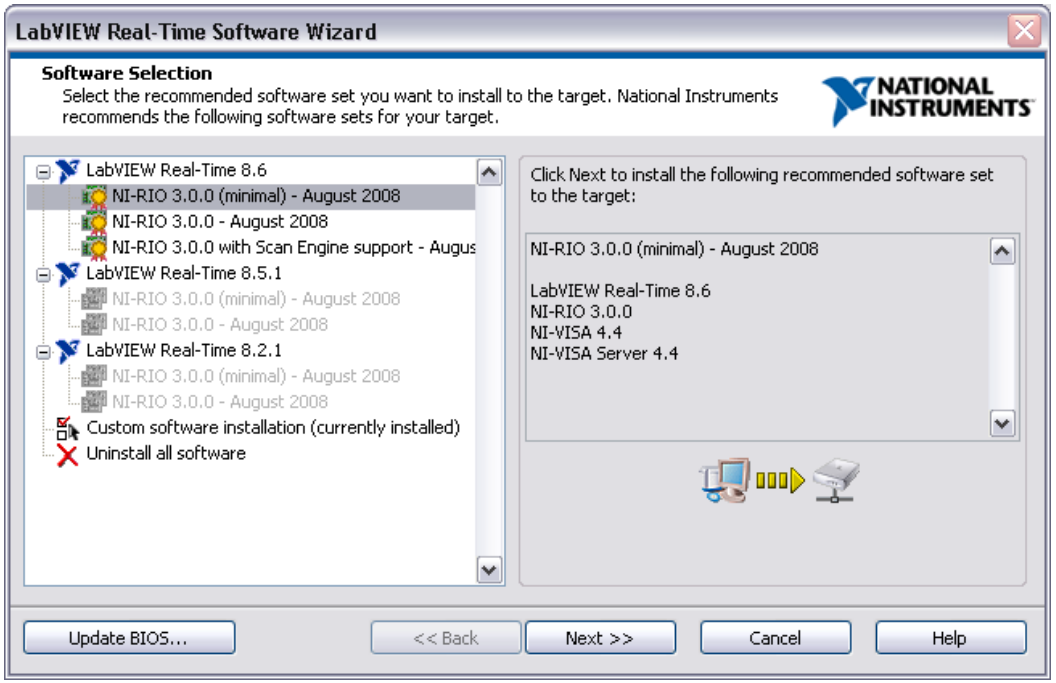
D. Add/Remove Software



D. Add/Remove Software

Default is to install minimal software necessary

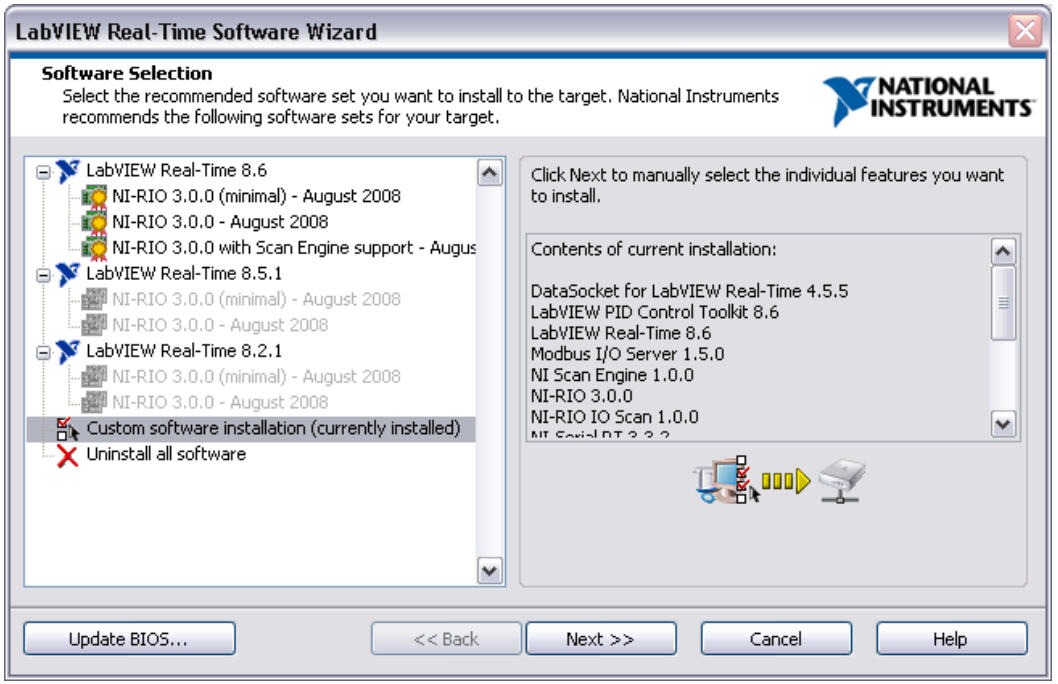
- Good for simple applications without
  - Network Variables
  - Watchdogs
  - PID Loops
  - Modbus Communication



D. Add/Remove Software

Custom software installation is for advanced applications

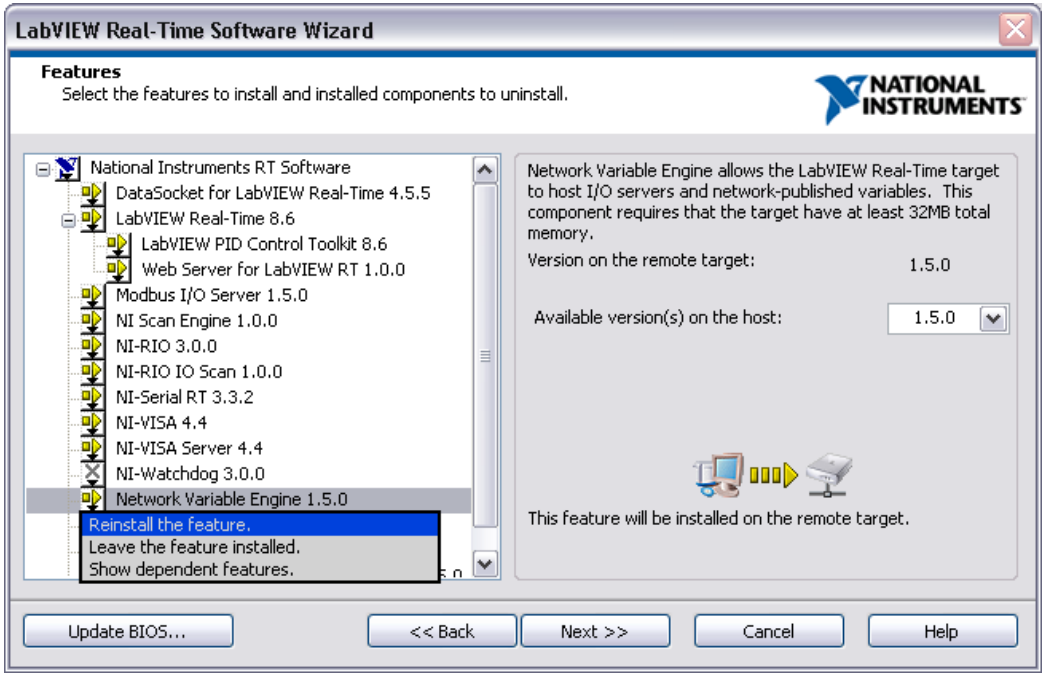
- Uses more nonvolatile memory
  - Only install what is necessary to avoid wasting space



D. Add/Remove Software

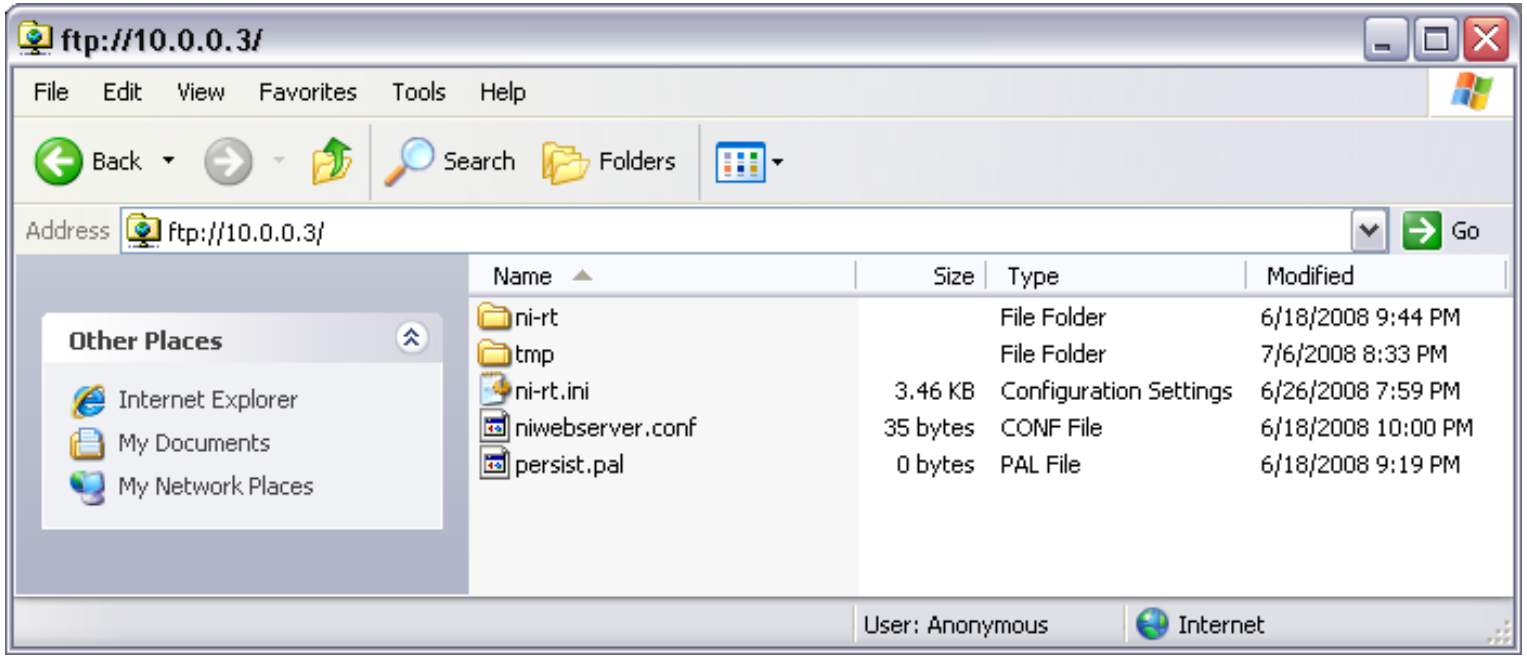
Verify proper installation settings

- After proper configuration is set click **Next** to install
  - After install, reboot the RT Controller



D. Add/Remove Software

Use FTP to display software on CompactRIO
Can check Modified column to see last update



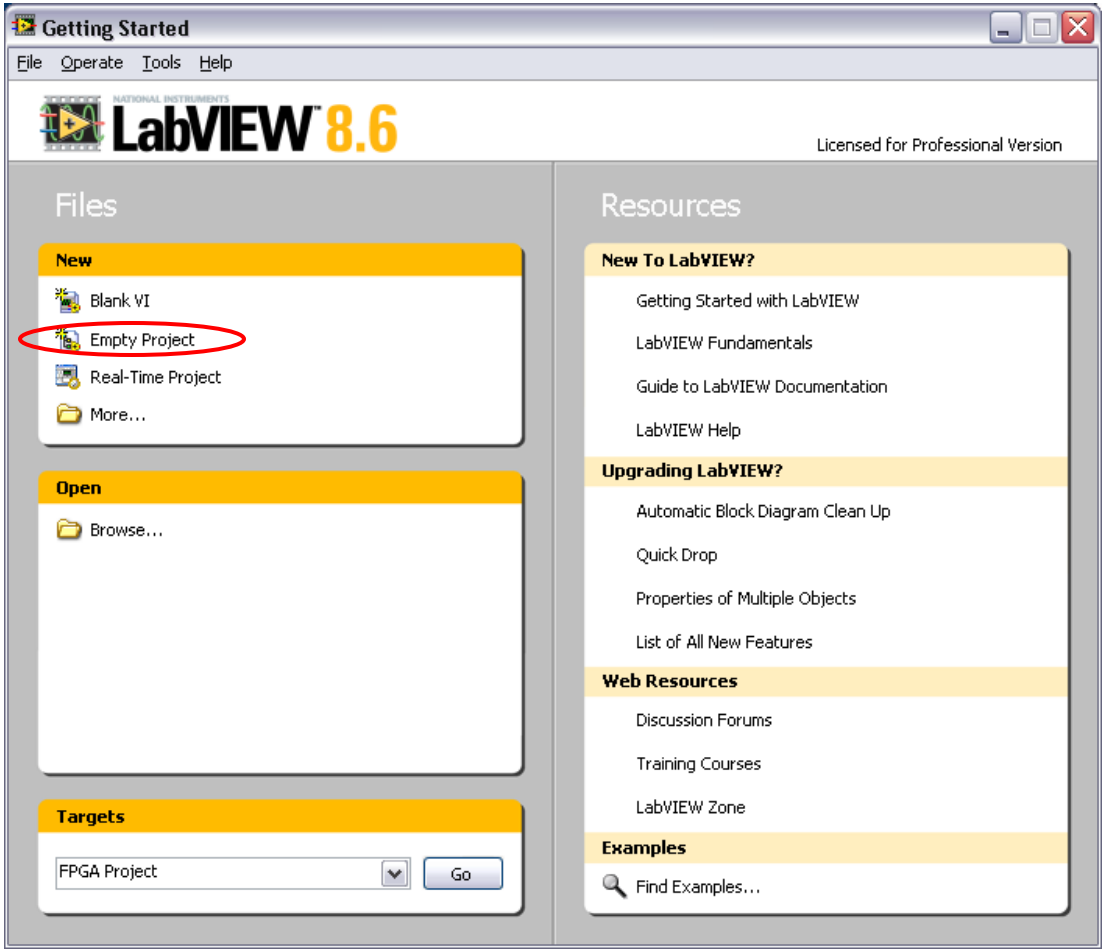
Lekcia 3

ARCHITEKTÚRA APLIKÁCIÍ

Architektúra aplikácií

- Create a Project
- Add the CompactRIO Target

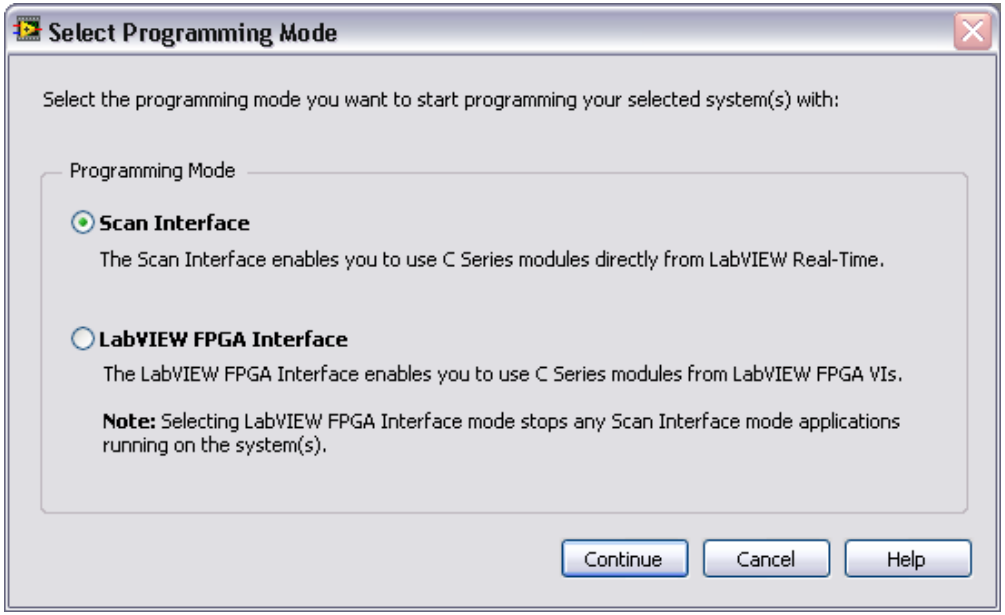
A. Create a Project



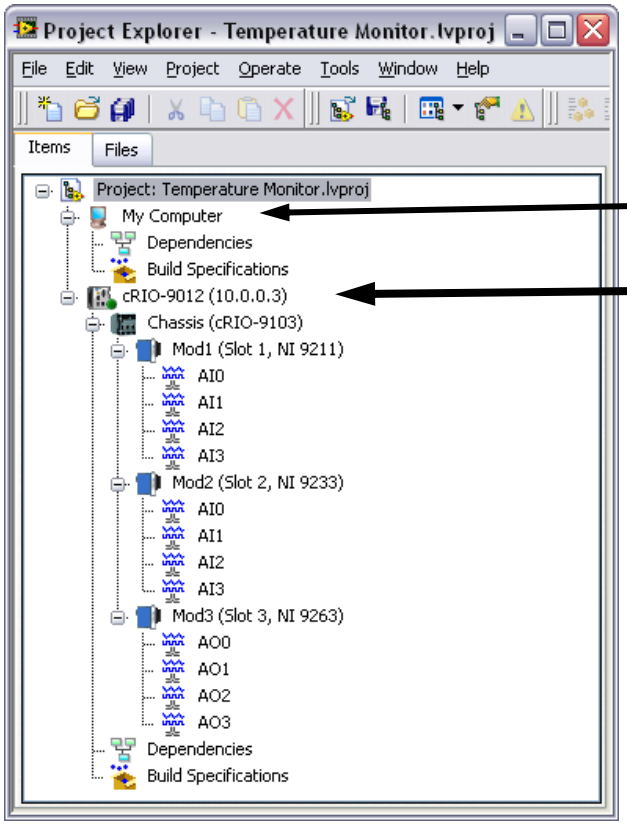
A. Create a Project

Two programming modes available:

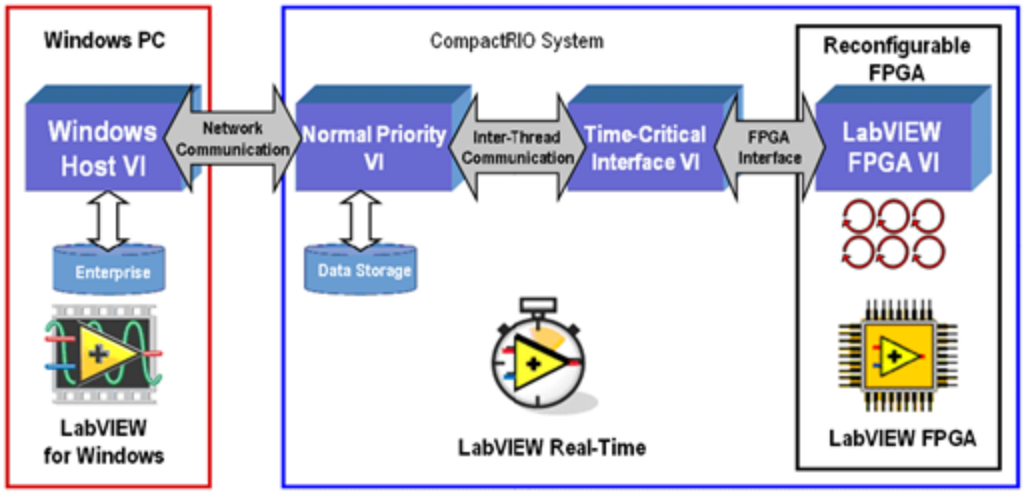
- Scan Interface
- LabVIEW FPGA Interface



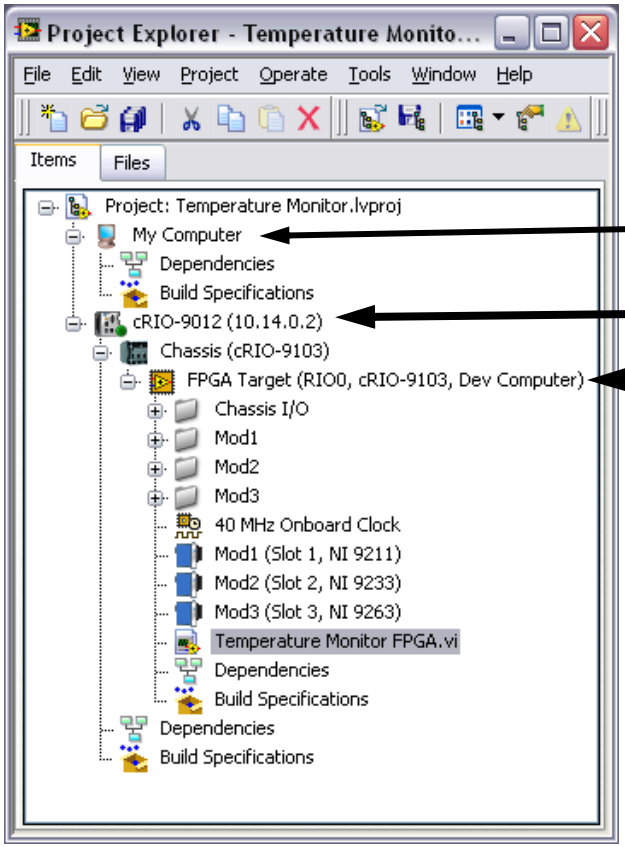
A. Create a Project – Scan Mode

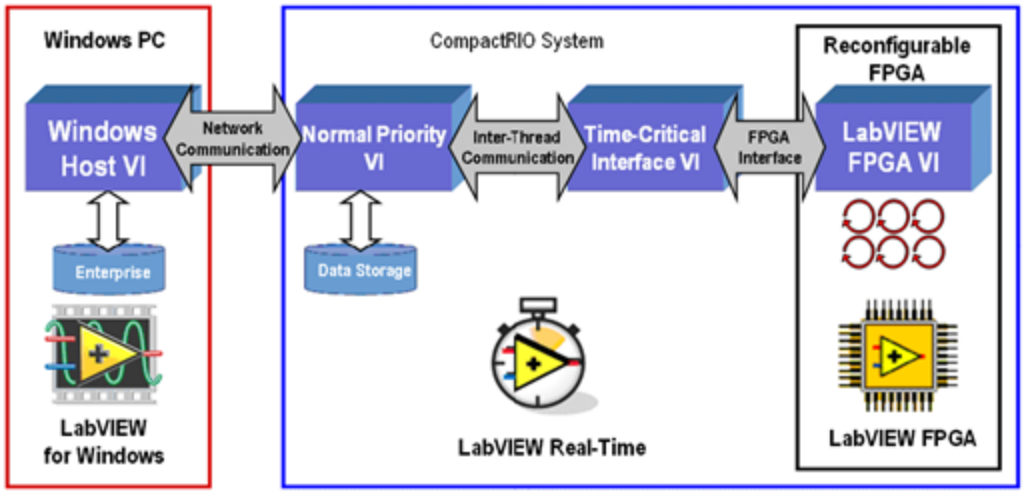


FPGA is handled behind the scenes



A. Create a Project – FPGA Mode

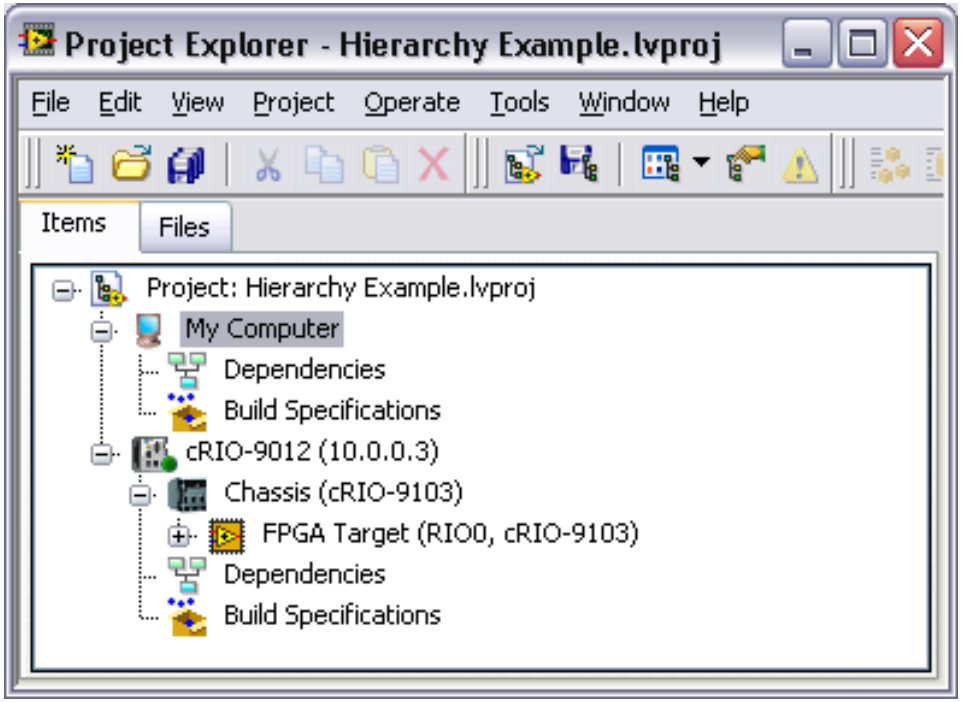




A. Create a Project – FPGA Mode

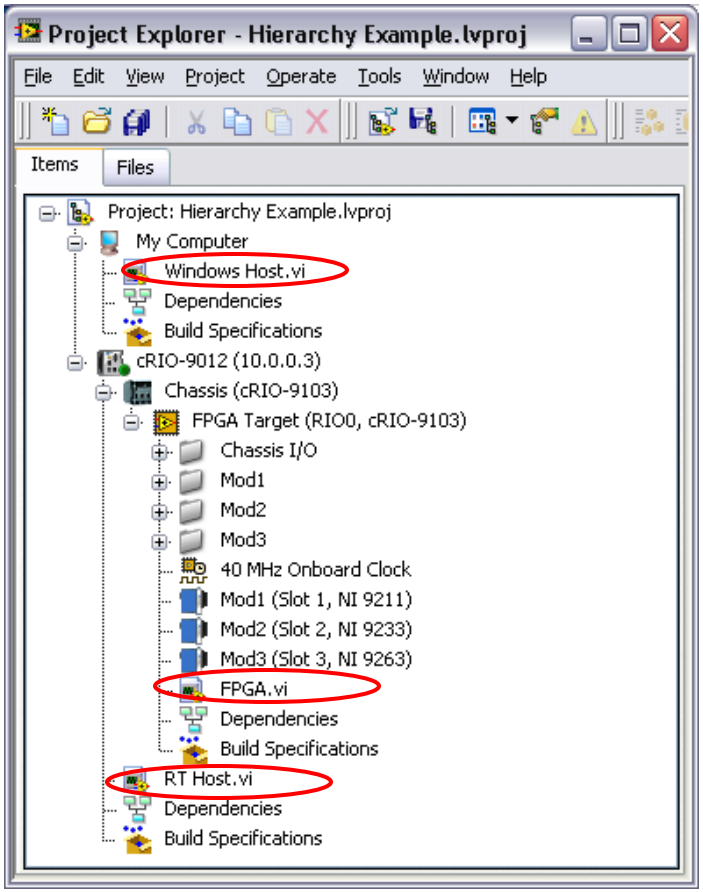
Three levels - indented

1. Project
2. My Computer and RT
3. FPGA



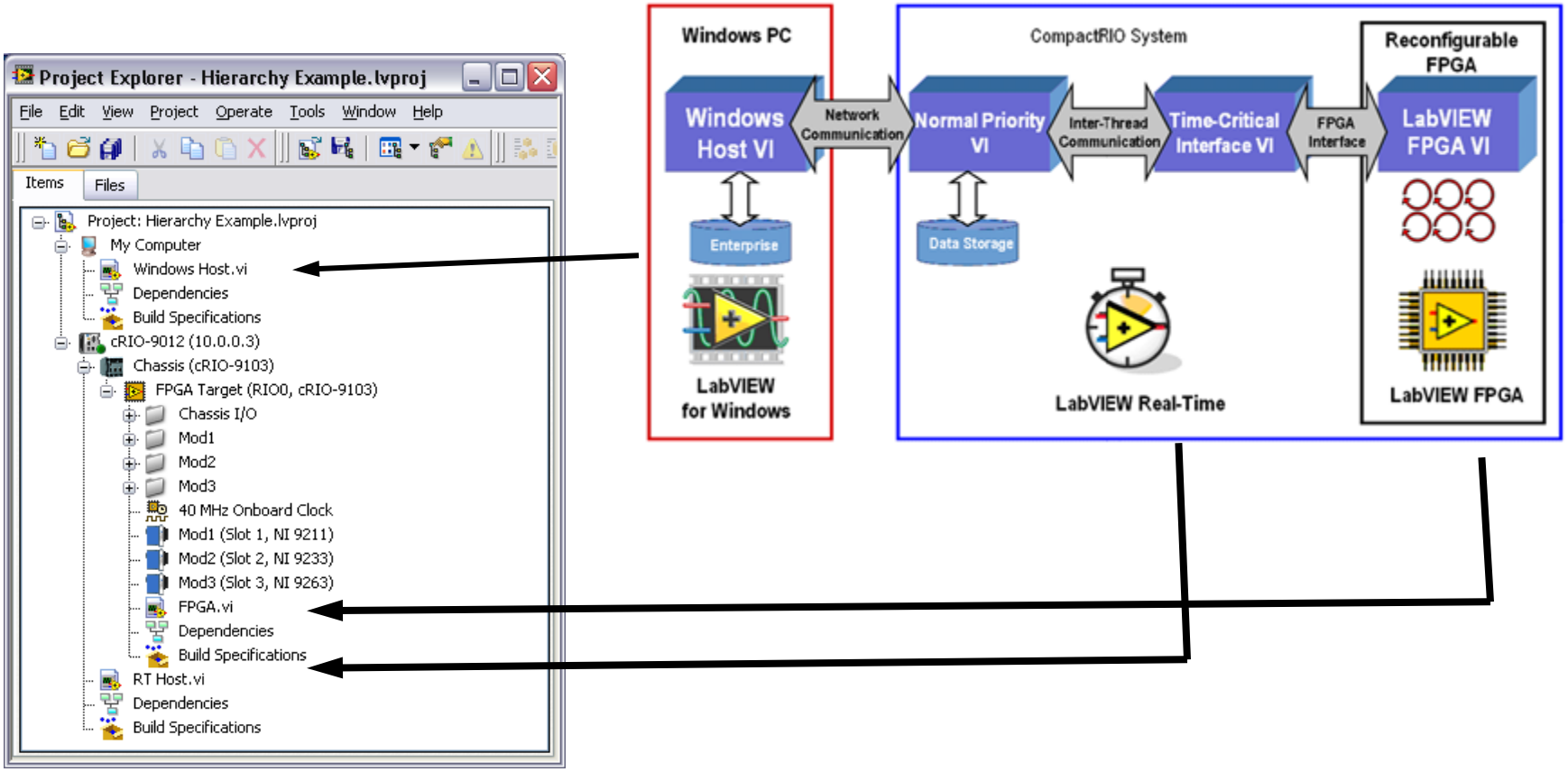
A. Create a Project

VIs reside under their target

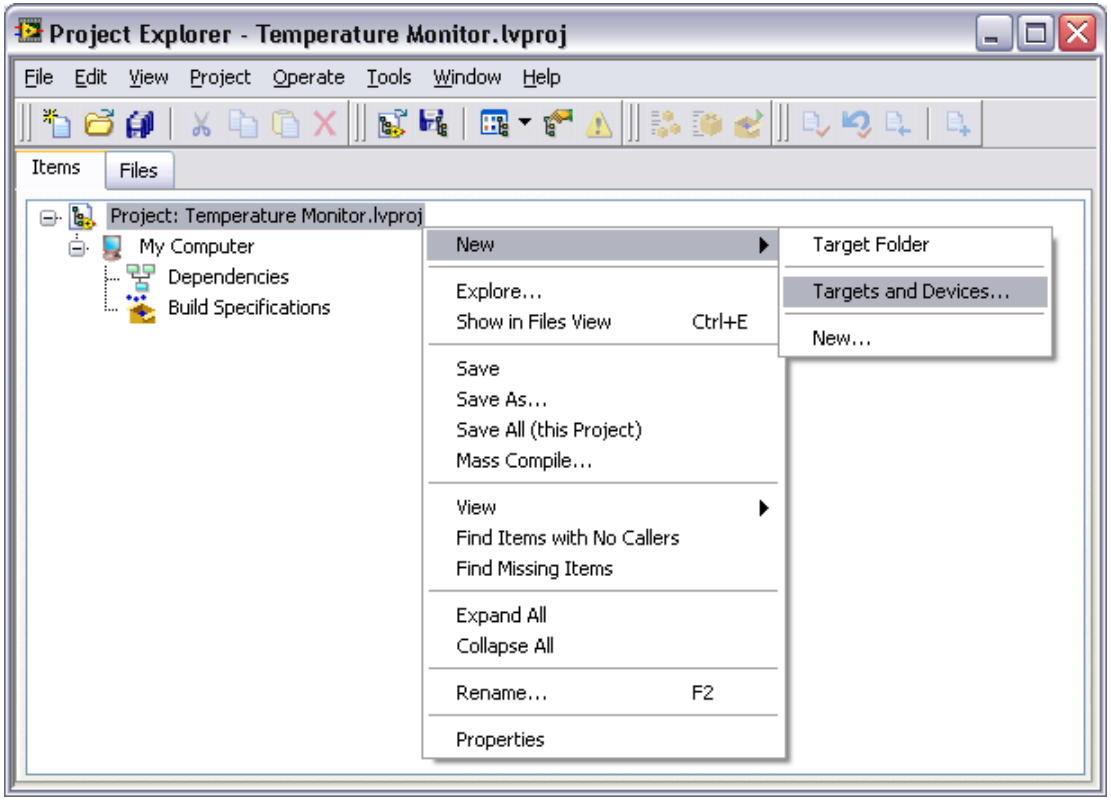


A. Create a Project

Software architecture mirrors hardware architecture

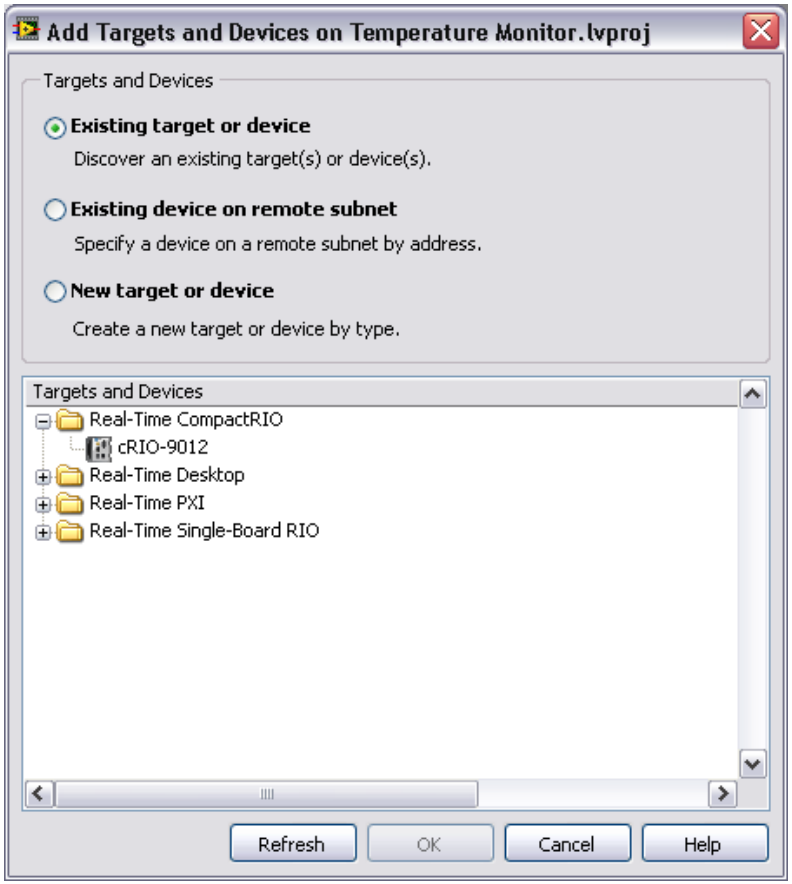


B. Add the CompactRIO Target



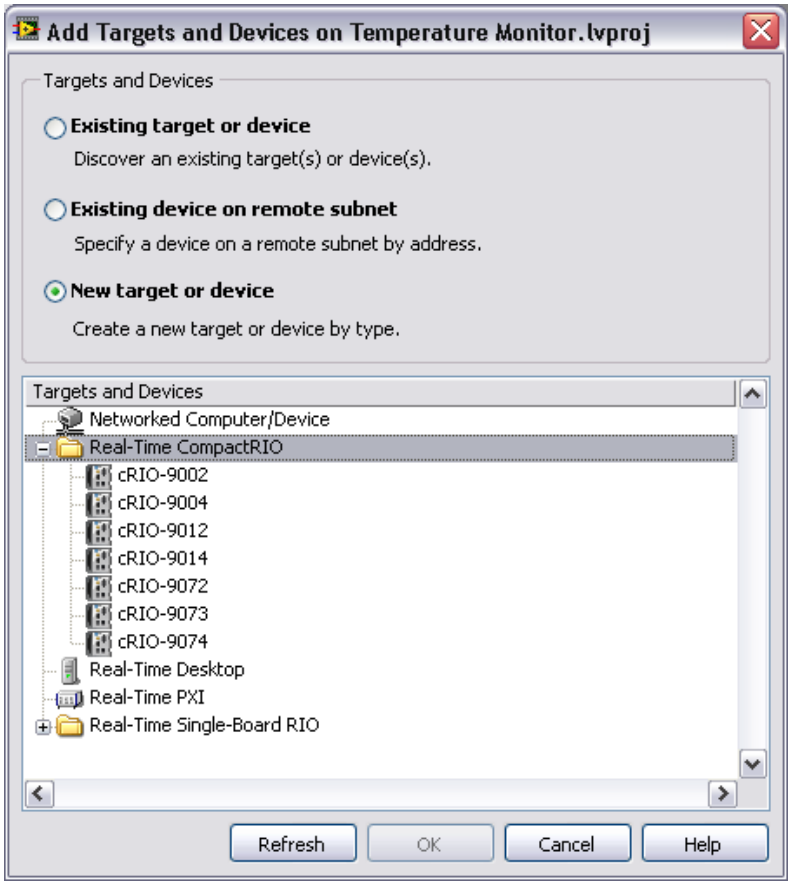
B. Add CompactRIO Target

- Existing target or device
  - Finds already configured device
- Existing device on remote subnet
  - Need to point to target if outside the local network



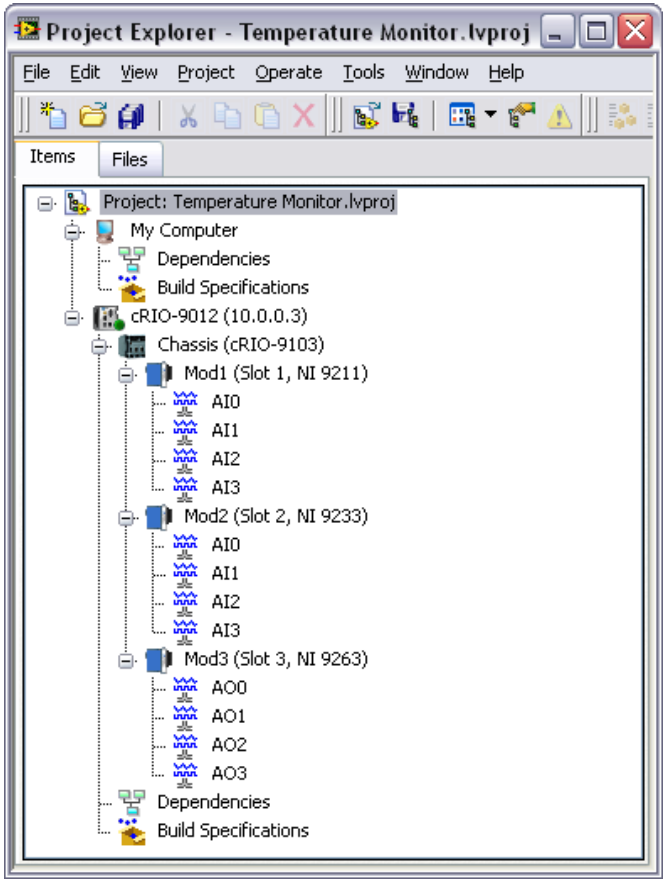
B. Add CompactRIO Target

- New Target or Device
  - Great for testing of equipment before it is in-hand
  - Must choose each component individually as it will be set up in final configuration
    - cRIO Controller
    - FPGA Target
    - C Series Modules



B. Add CompactRIO Target

- Verify and Save the Project
- Add the CompactRIO device
  - Select a programming mode
    - Scan mode
    - FPGA
- When all desired information is in the project, Save the Project

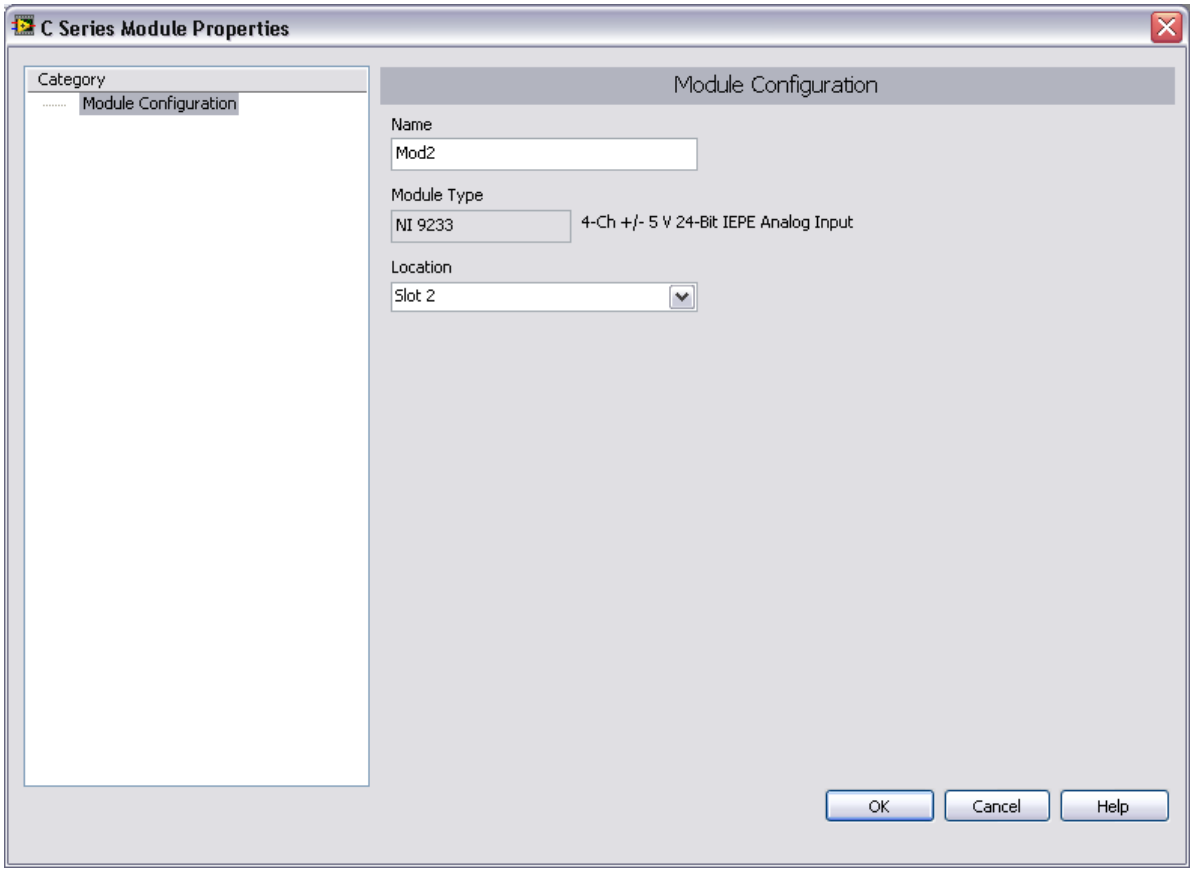


I/O Modules

Change Properties of a Module

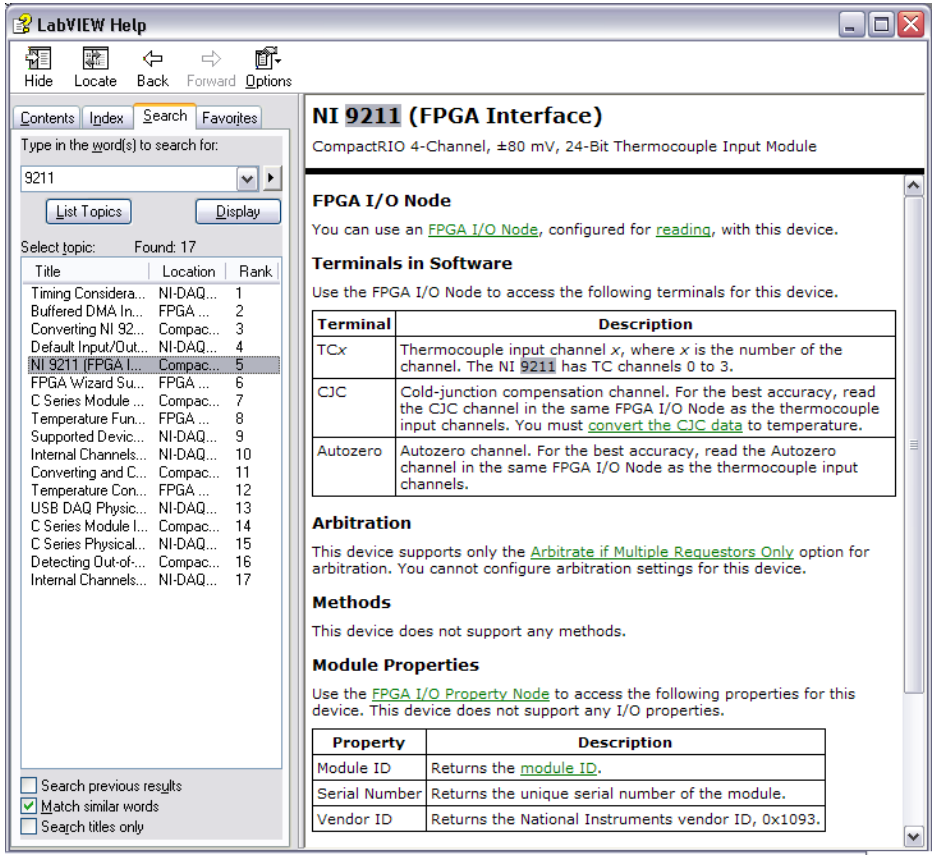
Can configure

- Alias in Project



I/O Modules

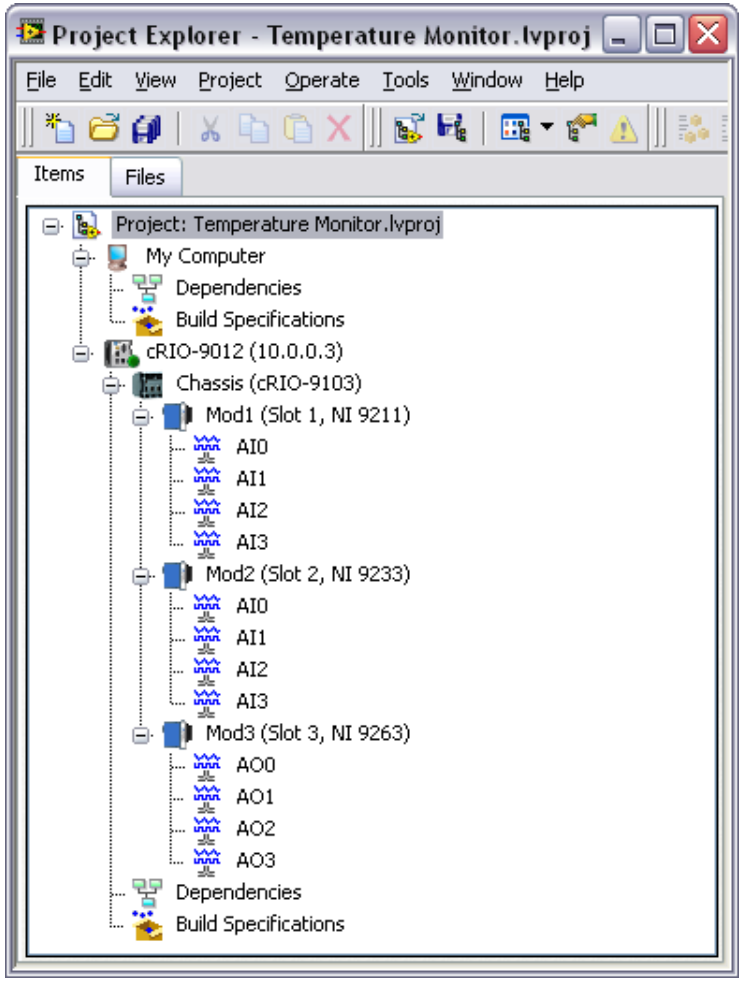
Module-specific information in LabVIEW Help



View FPGA I/O Items

Categorized by module

- Default Naming Convention
  - Scan Mode: ModX » ChannelY
  - FPGA Mode: ModX » ModX/ChannelY



Lekcia 4

SCAN MODE SNÍMANIA DÁT

Scan mode snímania dát

- What is CompactRIO Scan Mode?
- Using CompactRIO Scan Mode
- NI Distributed System Manager
- Programming with CompactRIO Scan Mode

A. What is CompactRIO Scan Mode?

- Enables access to I/O modules directly in LabVIEW Real-Time and host applications without FPGA programming
- Powered by two technologies
  - RIO Scan Interface (RSI)
  - NI Scan Engine

A. What is CompactRIO Scan Mode?

Uses

- Allows you to get an application up and running quickly
- Applications with synchronous I/O updates at rates up to 1 kHz
- Add counter, PWM, or quadrature encoder functionality to any eight-channel digital C Series module

A. What is CompactRIO Scan Mode?

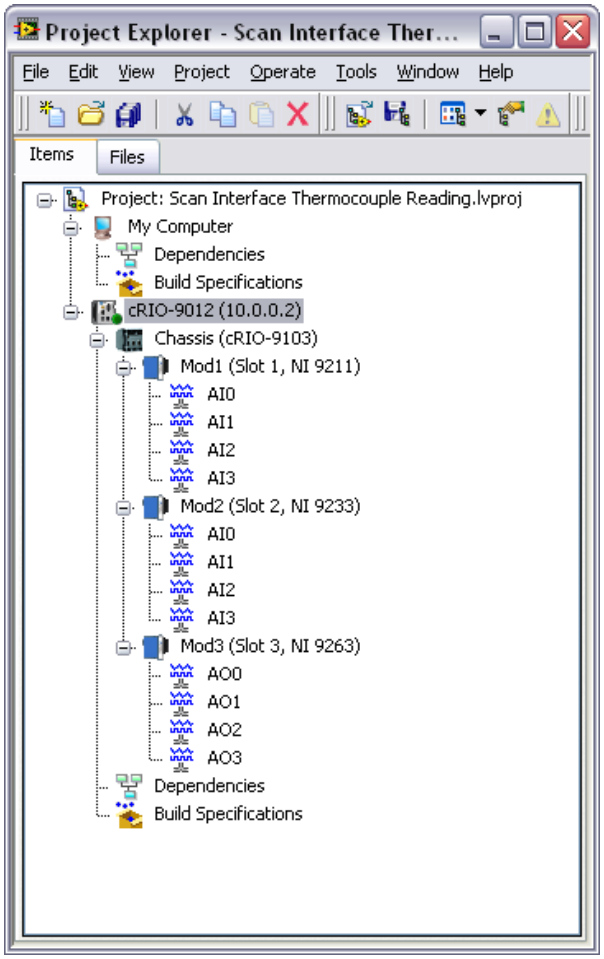
Requirements

- Controller
  - VxWorks Real-Time Operating System
  - 2M gate FPGA
  - Examples: NI 9012, 9014, 9073, 9074
  - Not supported: Pharlap controllers (NI 9002, 9004)
- Supported backplanes
  - Examples : 9103, 9104, 9073, 9074

B. Using CompactRIO Scan Mode

Getting Started

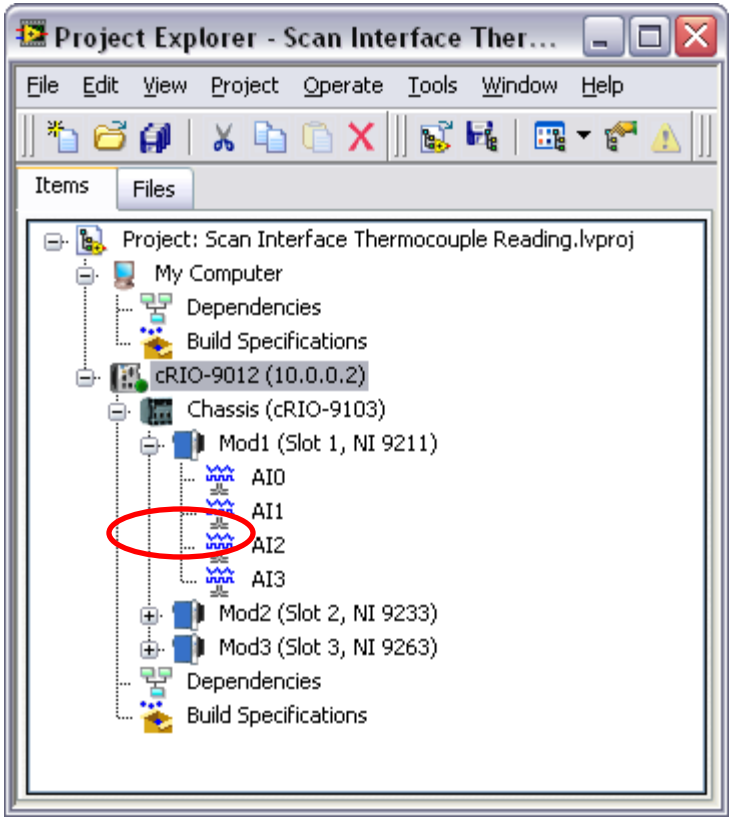
- Connect I/O modules to Real-Time target
- Install NI Scan Engine on the target
- Add the target to your LabVIEW project
- LabVIEW automatically detects modules and create an I/O variable for each channel



B. Using CompactRIO Scan Mode

What is an I/O Variable?

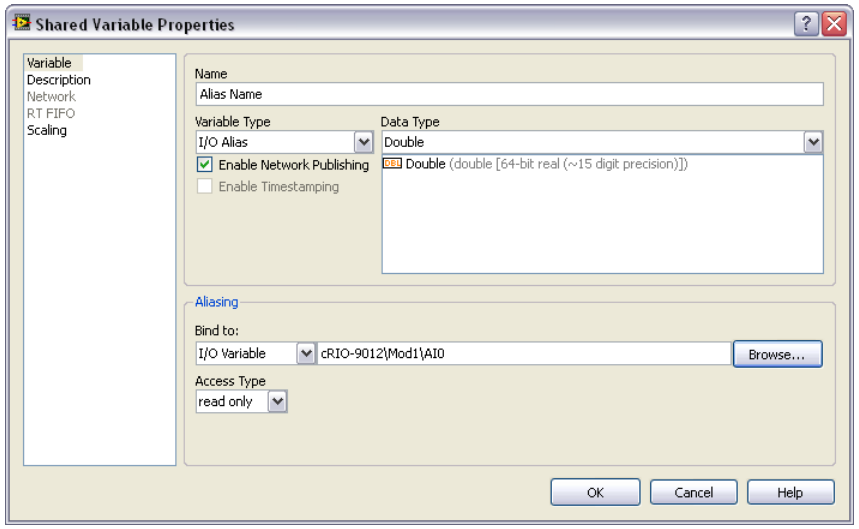
- A type of shared variable that is tied directly to a physical I/O channel
- Can be used to read/write directly to an I/O channel



B. Using CompactRIO Scan Mode

Creating I/O Aliases

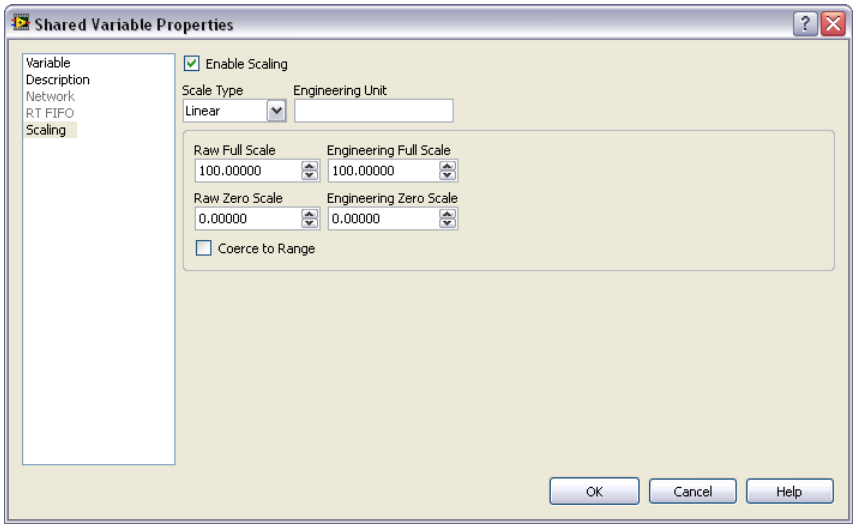
- Provides an extra layer of abstraction from the physical I/O channel
- Created in the Project Explorer
  - Create a new variable on the RT target
  - Modify the Shared Variable Properties dialog box



B. Using CompactRIO Scan Mode

Custom Scaling

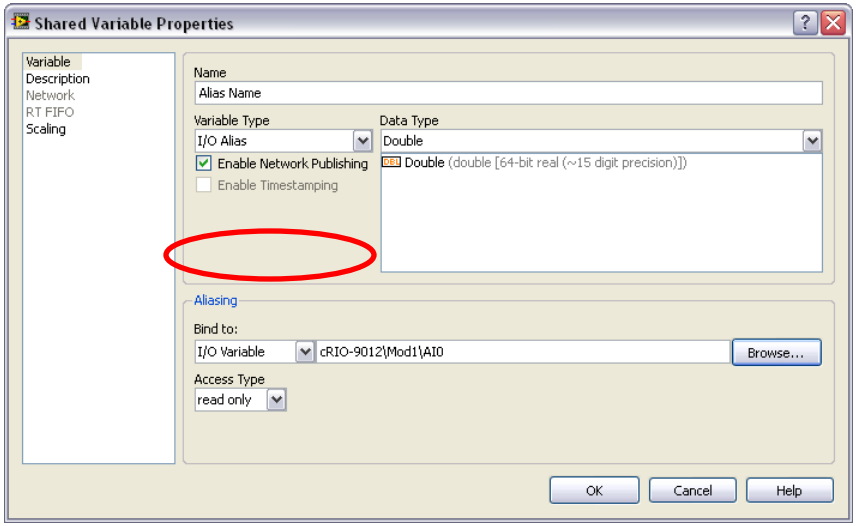
- You can enable scaling on an I/O variable or alias on the Scaling page of the Shared Variable Properties dialog box
- Example: I/O variable associated with a thermocouple input
  - Create a Celsius alias and a Fahrenheit alias
  - Scale each alias and use them to display temperature data in both scales



B. Using CompactRIO Scan Mode

Network Publishing of I/O Variables

- Use to monitor I/O values on a host computer or to access an I/O variable from a remote target
- Enable/disable network publishing in the Shared Variable Properties dialog box



B. Using CompactRIO Scan Mode

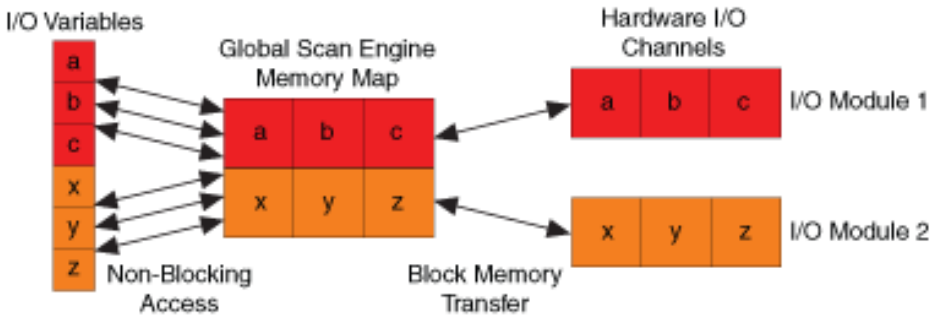
I/O Variable Access Methods

- LabVIEW adds I/O variables to a global scan engine memory map and updates all values concurrently
- Two access methods
  - Scanned I/O access
  - Direct I/O access
- Right-click on the I/O variable and select either **Change to Scanned** or **Change to Direct**, depending on the current state of the variable

B. Using CompactRIO Scan Mode

Scanned I/O Access

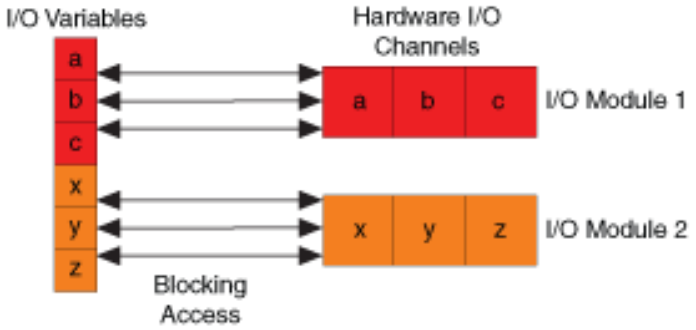
- Default setting for each I/O variable
- Use for groups of I/O channels that update at a single rate
- Uses scan engine memory map to perform non-blocking I/O reads and writes
- For each read, the scan engine returns the most recent value stored in the memory map



B. Using CompactRIO Scan Mode

Direct I/O Access

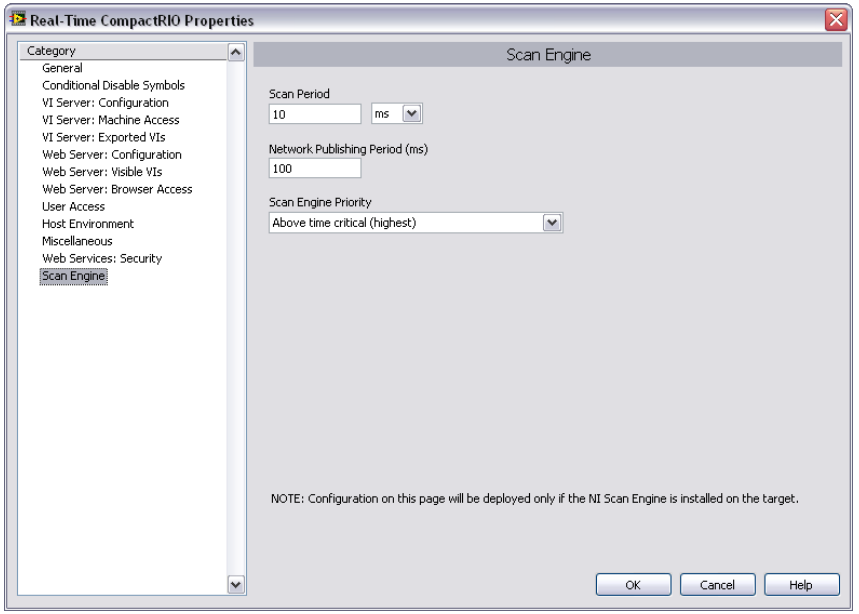
- Use for single-point local I/O channels with rates asynchronous to the scan period
- Bypasses the scan engine memory map and communicates directly with the I/O device driver



B. Using CompactRIO Scan Mode

Configuring Scan Mode Settings

- Open the LabVIEW project
- Right-click on the Real-Time target and select Properties to show the Real-Time CompactRIO Properties dialog box
- Select Scan Engine from the Category list to display the Scan Engine Page
  - Set the Scan Period, Network Publishing Period (ms) and Scan Engine Priority



B. Using CompactRIO Scan Mode

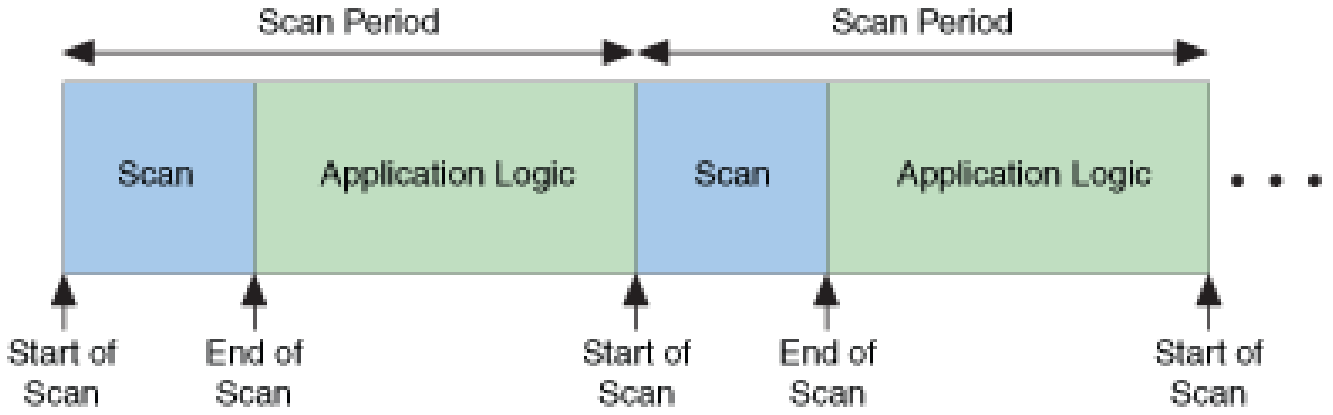
Scan Engine Page

- **Scan Period** – Period of the NI Scan Engine
- **Network Publishing Period (ms)–** How often the target updates published values on the network
  - Should not be faster than the scan period
- **Scan Engine Priority** – Priority of the NI Scan Engine thread on the Real-Time target

B. Using CompactRIO Scan Mode

Scan Period

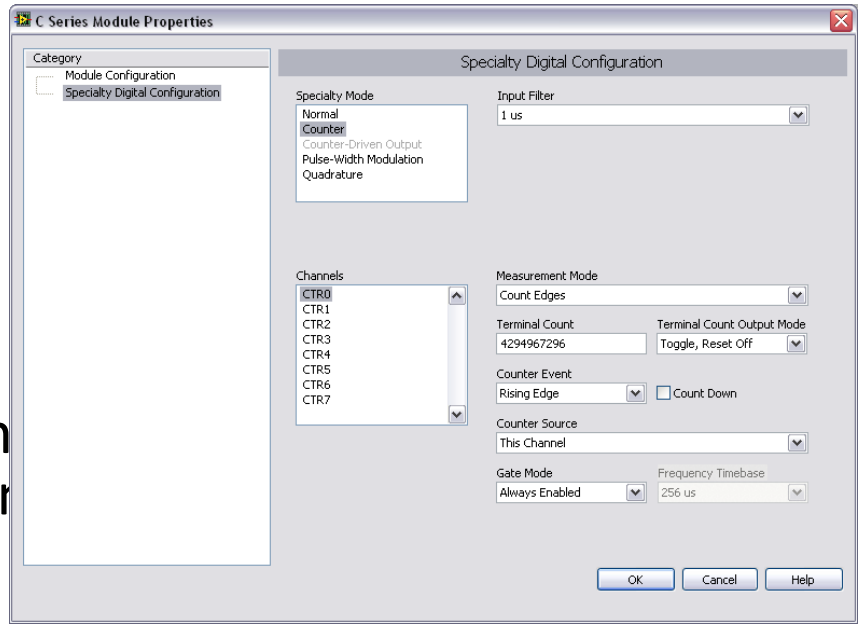
- NI Scan Engine executes at regular intervals determined by the scan period
- Choose a period long enough to accommodate both the scan itself and application logic



B. Using CompactRIO Scan Mode

Configuring Digital I/O Modules

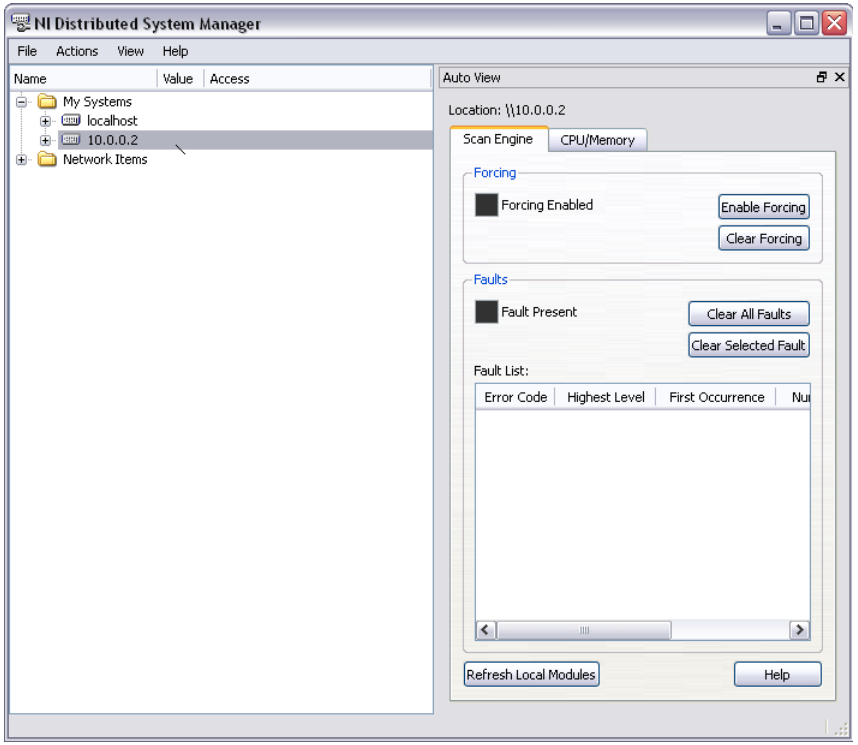
- Scan mode adds functionality to any 8-channel digital C-Series module
  - Counter
  - Quadrature encoder
  - PWM
- No programming required
- Settings are configured from the LabVIEW project, but run on the FPGA for accuracy and speed



C. NI Distributed System Manager

What is the NI Distributed System Manger?

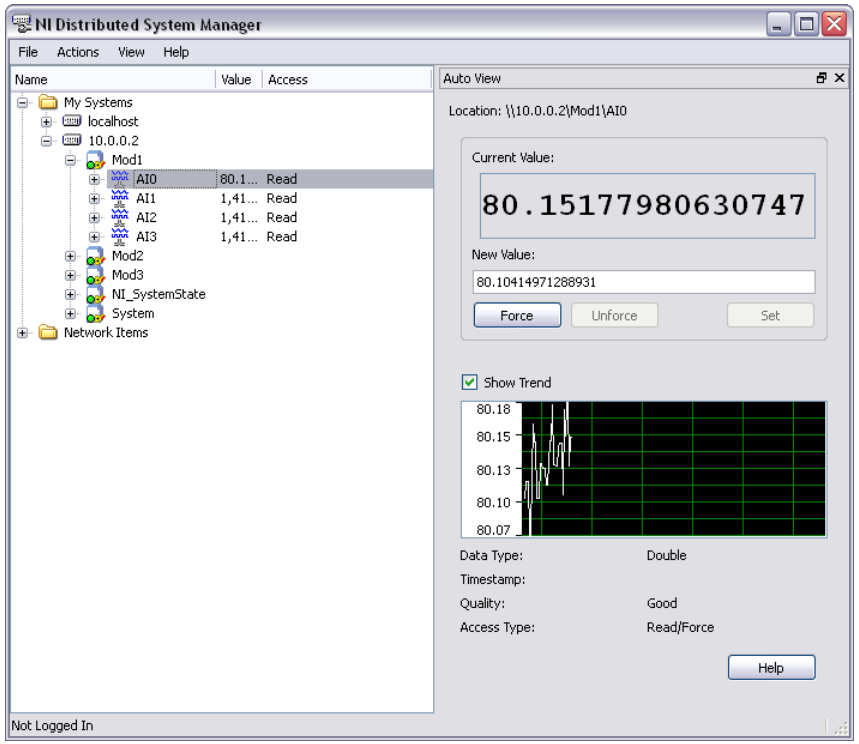
- Provides a central location for monitoring systems on the network and managing published data
- Provides test panels for CompactRIO modules



C. NI Distributed System Manager

Test Panel Usage

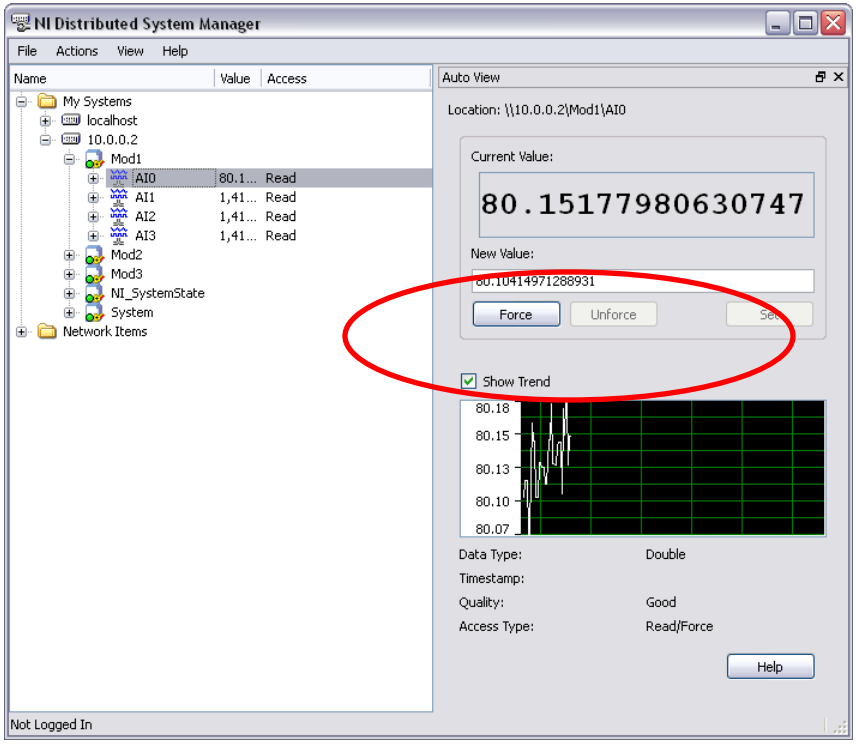
- Read/write data directly to the I/O variables
- Gives visibility into memory usage and processor load for cRIO modules
- Enables you to force the values of I/O variables



C. NI Distributed System Manager

Forcing Values on I/O Variables

- Causes associated I/O channel to assume the value you specify
  - Don’t have to stop or change the real-time application
  - Holds the specified value until:
    - You unforce the variable
    - Target is rebooted
    - You force the variable to a different value



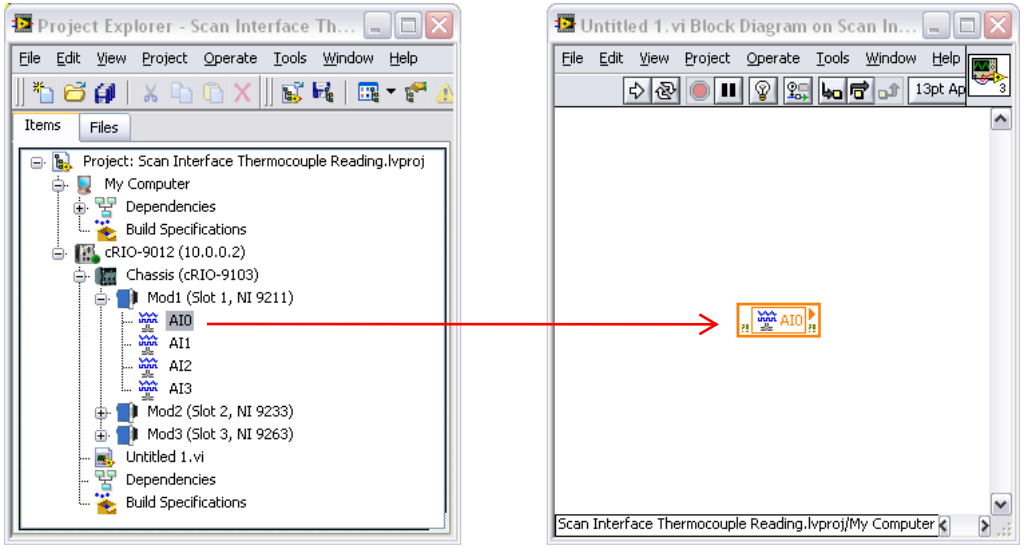
C. NI Distributed System Manager

- Forcing Values on I/O Variables
  - NI Scan Engine does not update values for a forced variable
  - Unforcing the variable returns control to the scan engine
  - Forced values apply to aliases as well as standard I/O variables
    - Forced I/O variable forces all associated aliases
    - Forced alias forces parent I/O variable and all of its associated aliases

D. Programming with CompactRIO Scan Mode

Using the I/O Variable on the Block Diagram

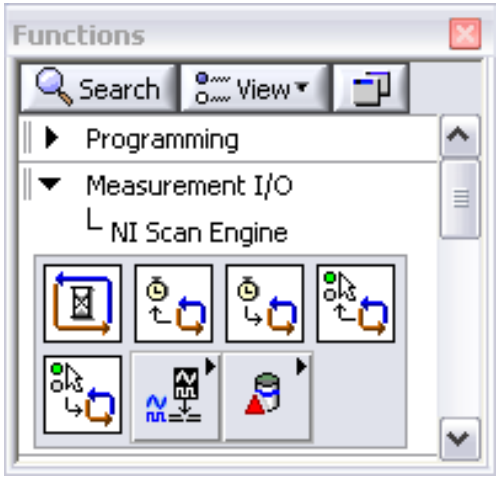
- Drag the I/O variable or alias from the Project Explorer window onto your LabVIEW Real-Time or host VI block diagram
- Use the placed I/O variable to read from or write to the I/O channel programmatically



D. Programming with CompactRIO Scan Mode

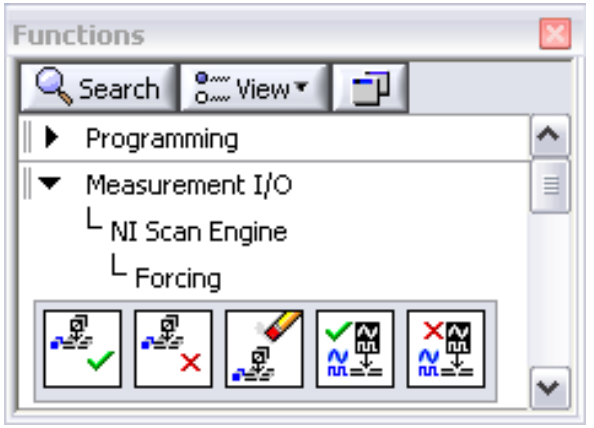
NI Scan Engine VIs (Installed with LabVIEW Real-Time)

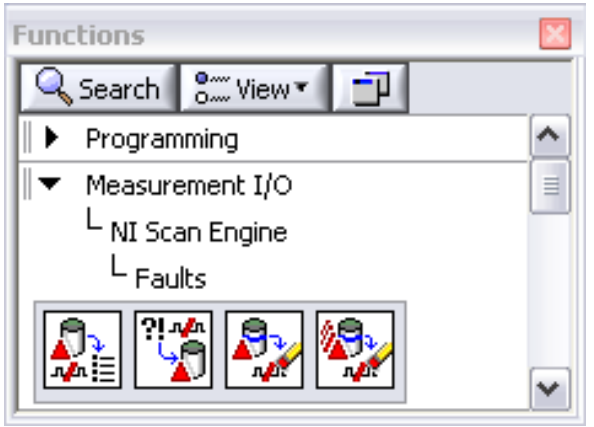
- Synchronize to Scan Engine
- Get Scan Engine Period
- Set Scan Engine Period
- Get Scan Engine Mode
- Set Scan Engine Mode
- Forcing Subpalette
- Faults Subpalette



D. Programming with CompactRIO Scan Mode

- Forcing VIs
  - Disable Variable Forcing
  - Enable Variable Forcing
  - Force Variable
  - Unforce Variable
- Faults VIs
  - Get Fault List
  - Set Fault
  - Clear Fault
  - Clear all Faults





D. Programming with CompactRIO Scan Mode

- Synchronization of LabVIEW Code to NI Scan Engine
  - Two methods
    - Synchronize to Scan Engine.vi
    - Synchronize Timed Structure to the scan engine
      - Use the **Synchronize to Scan Engine** timing source

Lekcia 5

PRÁCA S PROGRAMOVATEĽNÝM HRADLOVÝM POĽOM

Práca s programovateľným hradlovým poľom

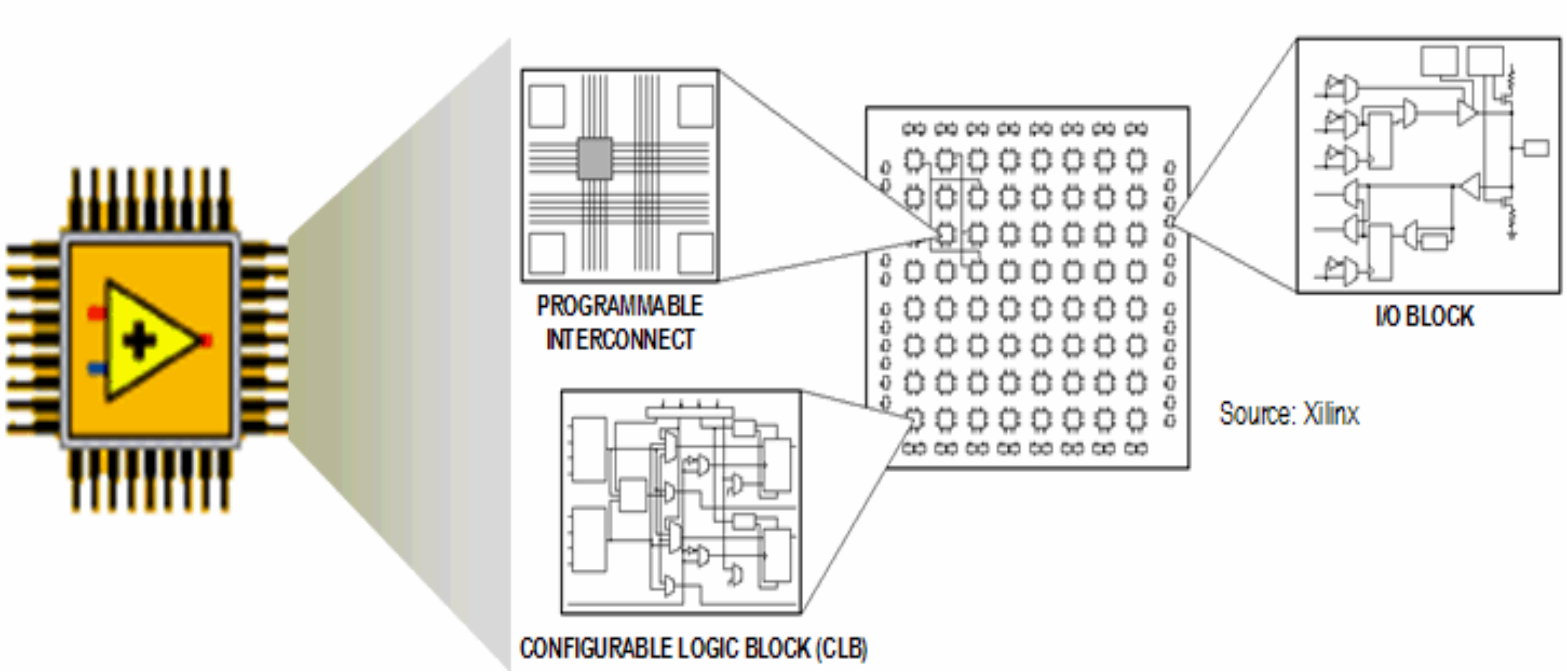
A. Introduction
B. Fixed Point Math
C. Defining FPGA Logic with LabVIEW
D. Developing the FPGA VI
E. Testing with the Development Machine

E.Interactive Front Panel Communication
F.Wiring the Modules
G.Compiling the FPGA VI
H.Downloading to Flash Memory
I. Using LabVIEW FPGA with CompactRIO Scan Mode

A. Introduction

- An FPGA is a chip that can be reconfigured with software.
- FPGAs are used in NI R Series DAQ, Compact Vision, and CompactRIO.
- An FPGA replaces millions of logic gates.

A. Introduction



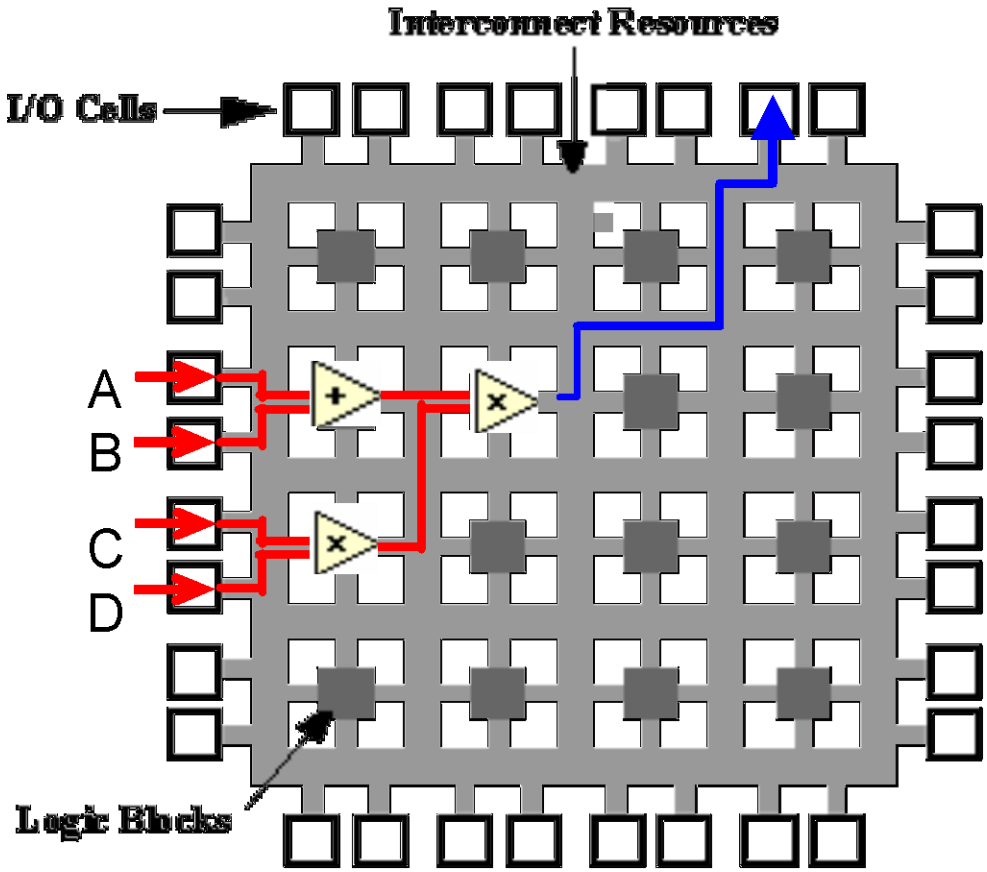
FPGA Layout and Components

A. Introduction

- Programmable interconnect switches and wires route signals in an FPGA.
- Switches are known as Register Flip-Flops.
- Flip-Flops pass data on a rising edge of the clock.
- Compiled LabVIEW code produces a Look-up Table (LUT) that defines outputs from all possible input values to a logic function.
- The LUT defines the interconnections between configurable logic blocks (CLBs).

A. Introduction

Implements a VI that calculates a value for F from inputs A, B, C, and D where F= CD(A+B)

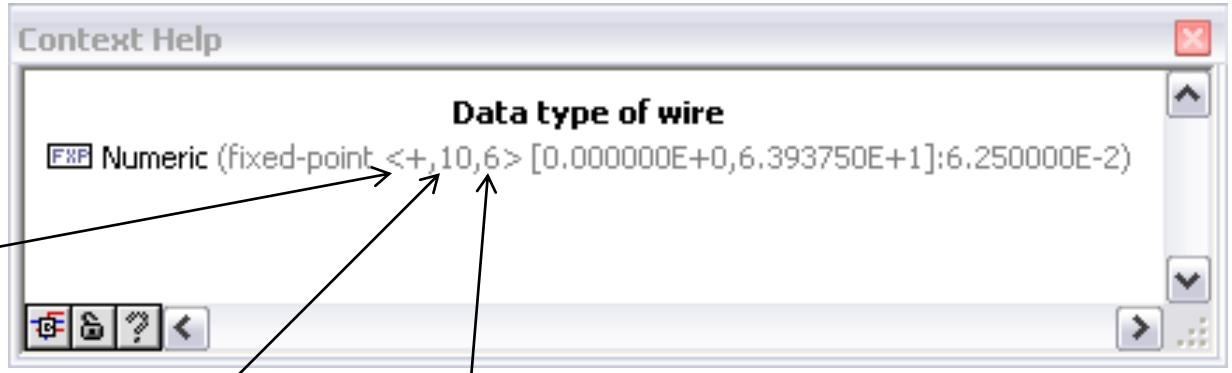


B. Fixed-Point Math

- Greatly simplifies arithmetic on the FPGA
  - Simplifies computations
  - Size and speed advantages of integer math
  - Must configure the appropriate range when using fixed-point math

Encoding, Word Length and Integer Word Length

•Encoding – Fixed Point numbers are either Signed (±) or Unsigned (+).



•Word Length – The total number of bits used for the Fixed-Point data.

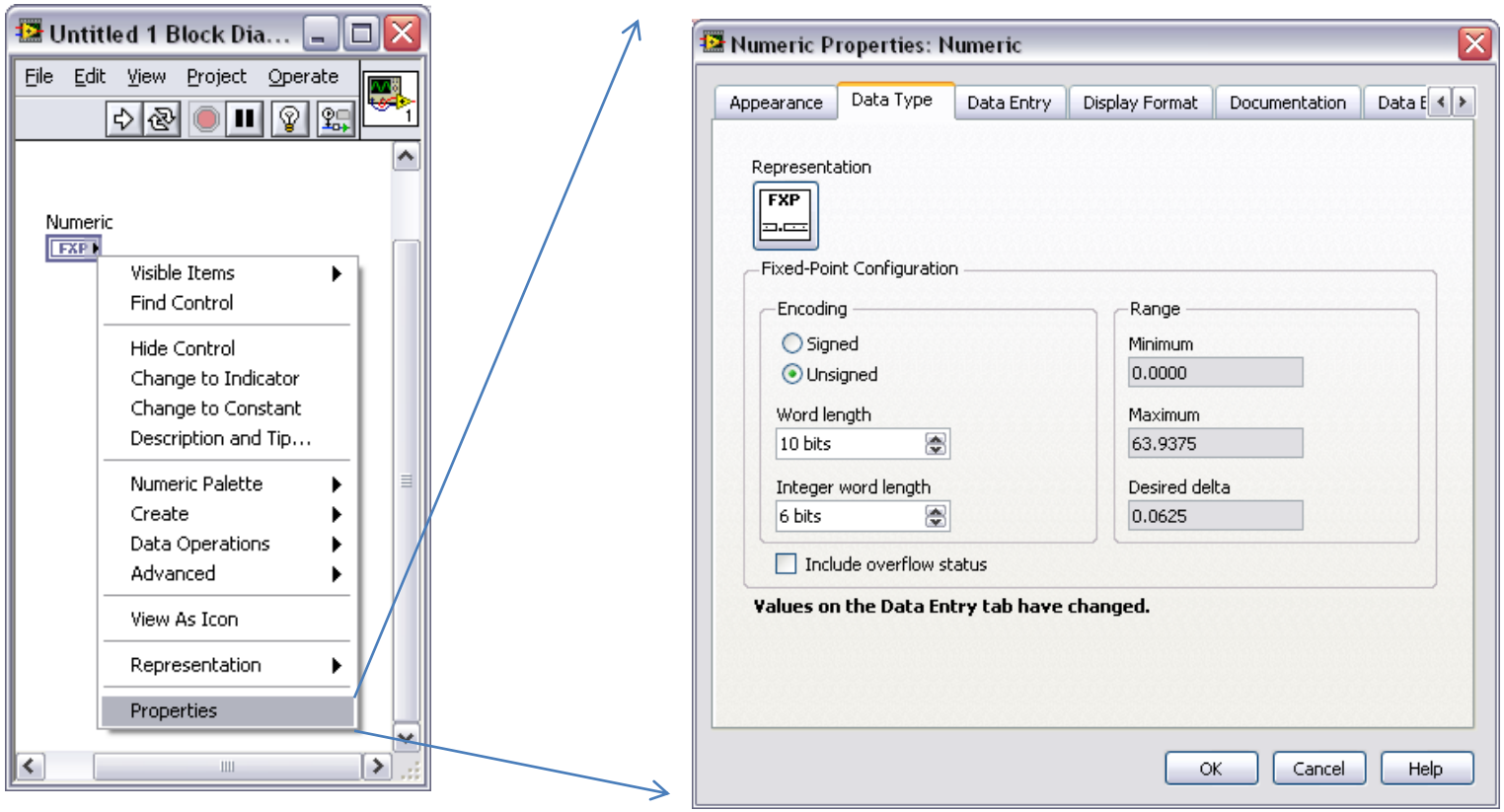
•Integer Word Length – The number of bits used in the integer portion of the Fixed-Point data.

B. Fixed-Point Math

- Numeric Representation Examples

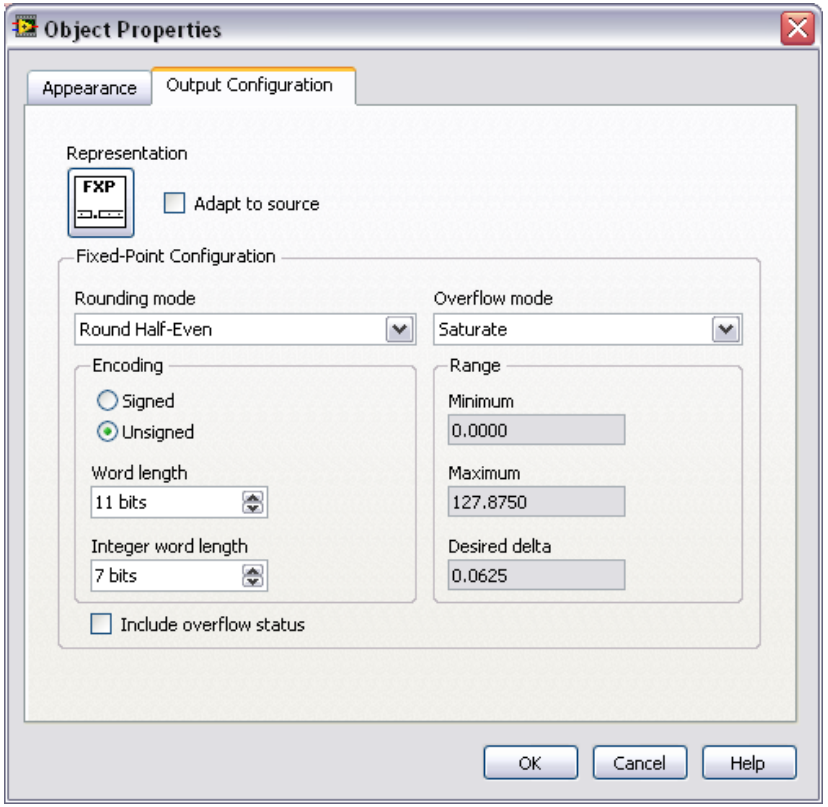
Representation	delta	Min Value	Max Value
U8	1	0	255
I8	1	-128	127
FXP <+,8,7>	0.5	0	127.5
FXP <+,8,6>	0.25	0	63.75
FXP <±,8,7>	0.5	-64	63.5
FXP <±,8,6>	0.25	-32	31.75

Fixed Point Configuration



Configuring Fixed Point Functions

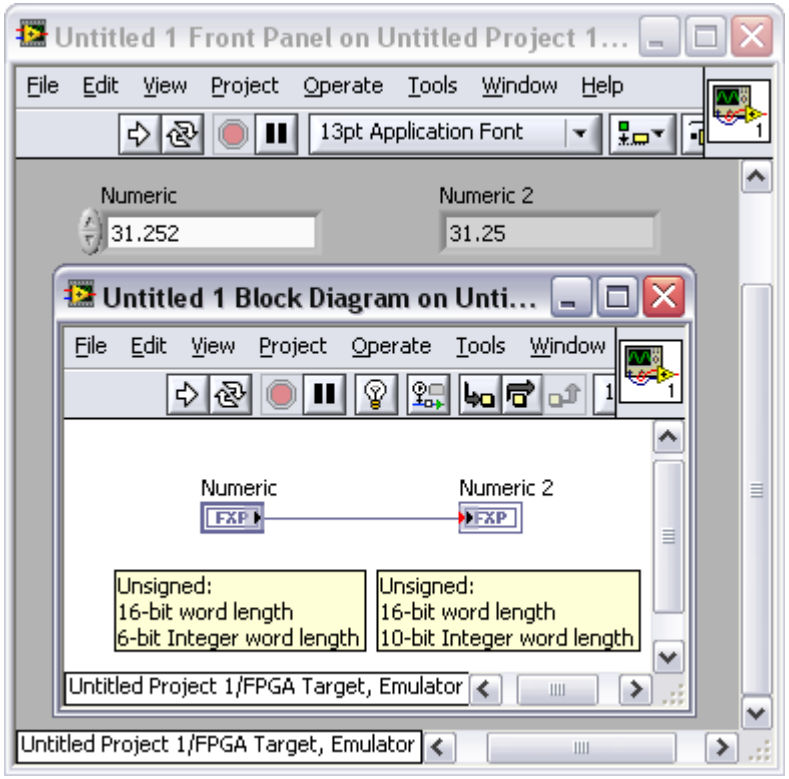
- Overflow – occurs when the result of an operation is outside the range of values that the output type can represent
- Rounding – occurs when the precision of the input value or the result of an operation is greater than the precision of the output type.



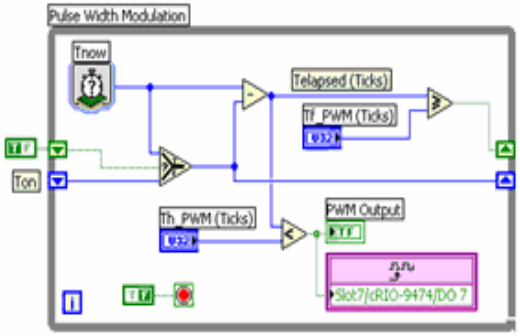
B. Fixed-Point Math

Selecting an Overflow and Rounding Mode

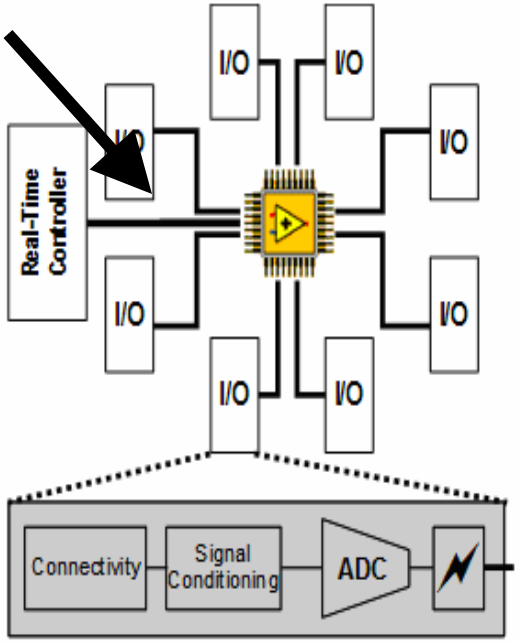
- Use same format for all related functions
- Mixing fixed-point configuration types can cause data loss



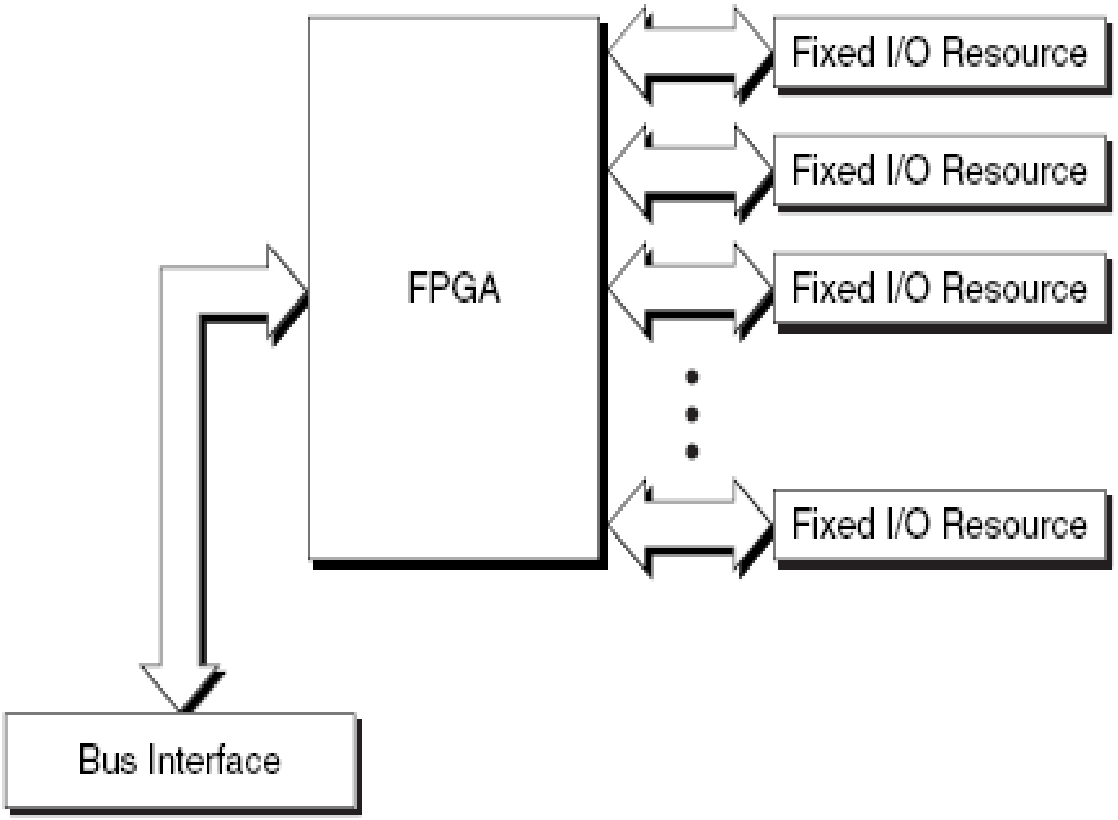
C. Defining FPGA Logic with LabVIEW



- FPGA Module
- Do not have to learn VHDL
- True parallel execution
- Deterministic



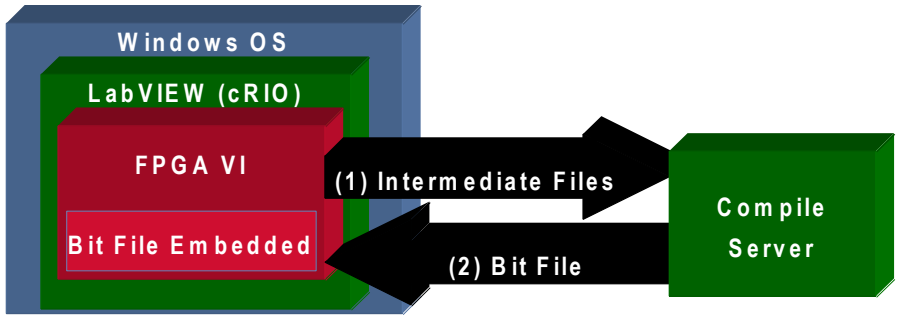
C. Defining FPGA Logic with LabVIEW



Communications between I/O, FPGA, and Controller

C. Defining FPGA Logic with LabVIEW

- Turn the FPGA VI into executable code
  - FPGA Module compiles VI
  - Graphical code translated to VHDL
  - Xilinx ISE compiler creates circuit from VHDL
  - Compiler optimizes the implementation
- A bitstream file results
- Bitstream loads at run time
- Bitstream reloads at power-up
  - On-board flash memory
  - Controller over PCI Bus



C. Defining FPGA Logic with LabVIEW

FPGA provides:

- Timing
- Triggering
- Processing

Each fixed I/O uses a portion of the FPGA logic

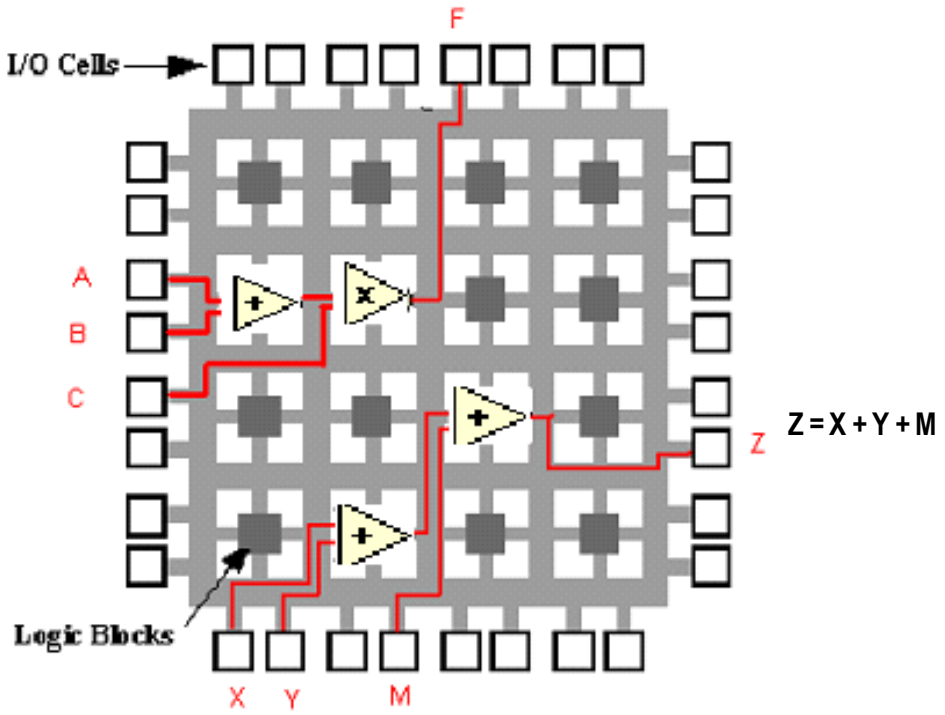
The PCI interface also uses a portion of the FPGA logic

C. Defining FPGA Logic with LabVIEW

- Multi-loop analog PID loop rates exceed 100 kHz on embedded RIO FPGA hardware whereas they run at 30 kHz in real-time without FPGA hardware.
- Digital control loop rates execute at up to 1 MS/s.
- Single-cycle while loops execute at 40 MHz (25 ns).
- Due to parallel processing ability, adding additional computation does not necessarily reduce the speed of the FPGA application.

C. Defining FPGA Logic with LabVIEW

True simultaneous parallel implementation of
F = (A+B)C and Z = X+Y+M in separate gates on an FPGA



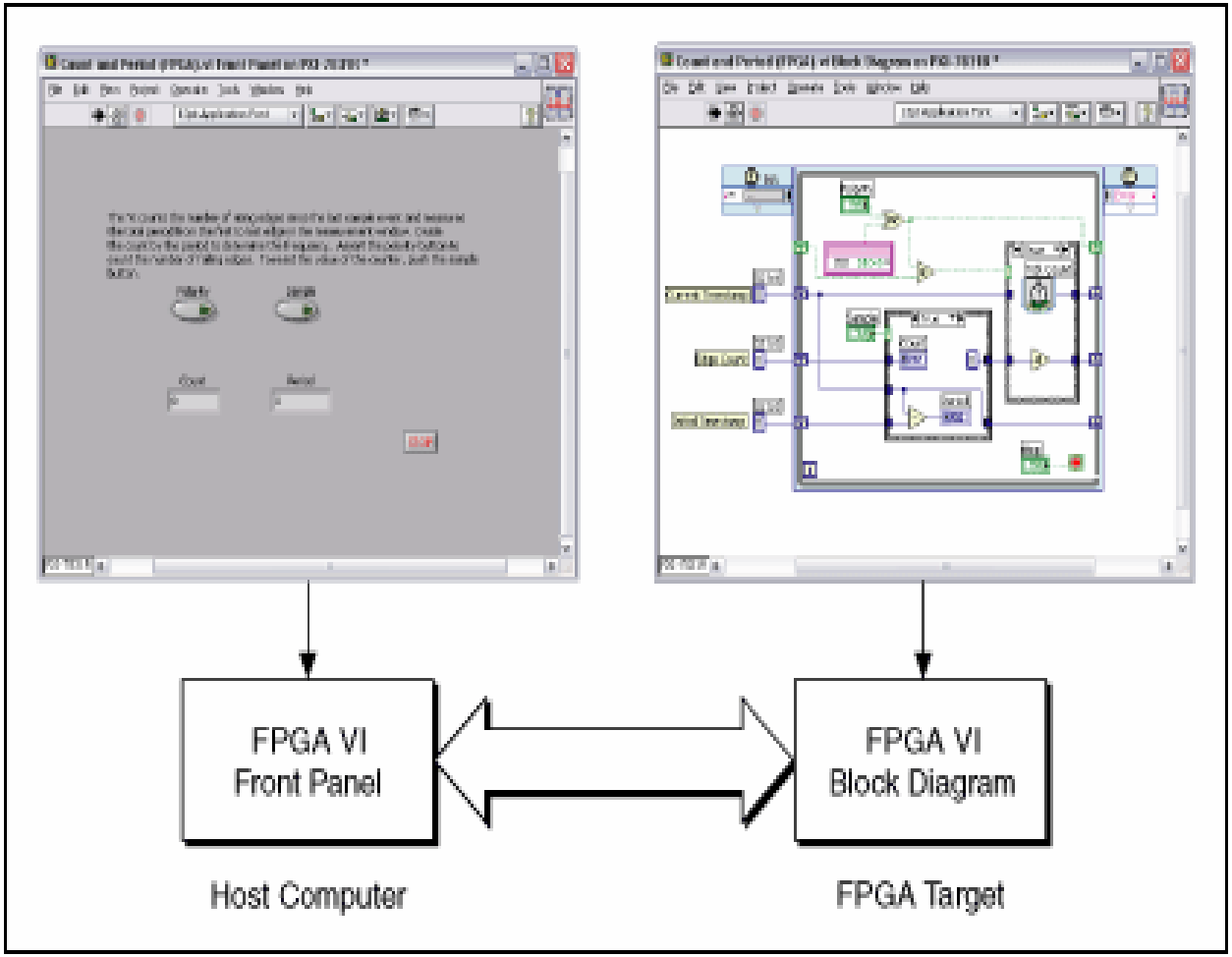
D. Developing the FPGA VI

FPGA Development Process

- No operating system on the FPGA
- Download and run only 1 VI at a time
- FPGA can run independently of the host
- FPGA can store data
- Add a VI in the FPGA Target activates the FPGA palette

D. Developing the FPGA VI

Interactive Front Panel Communication



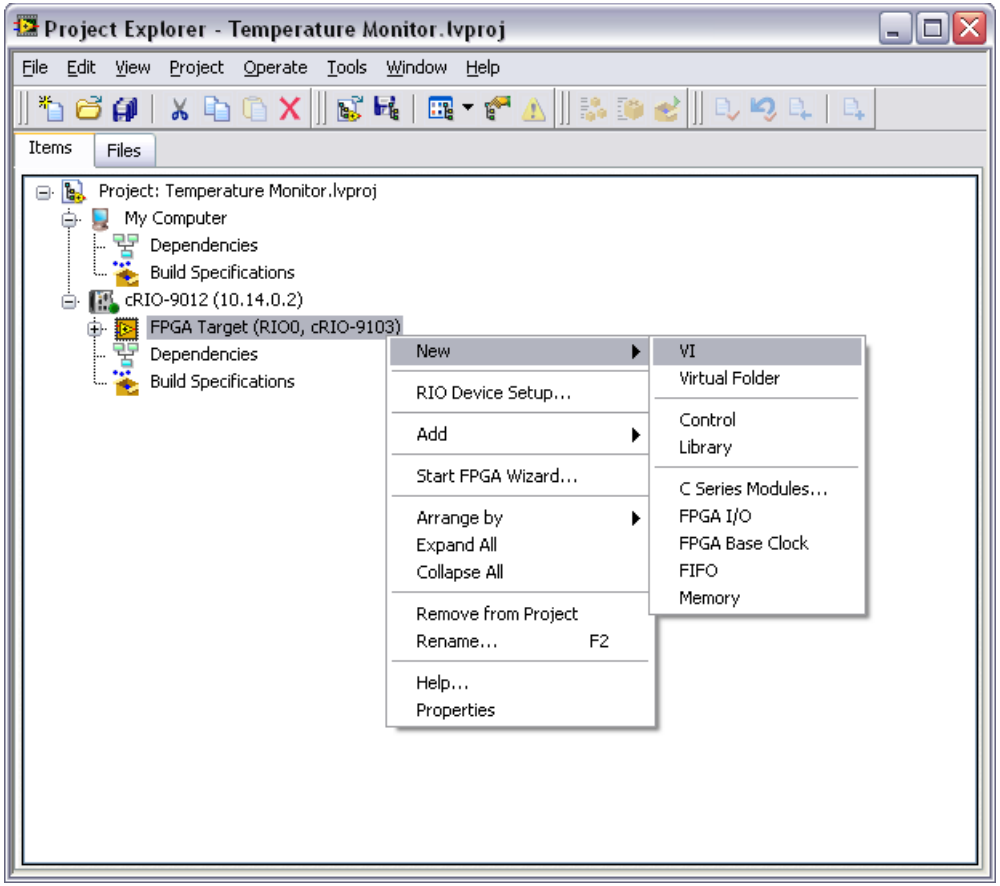
D. Developing the FPGA VI

- FPGA is fast and reliable
- FPGA has limited space

FPGA	RT or PC
Time critical control	Data analysis
Acquisition	File I/O
Timing in the FPGA	User interface

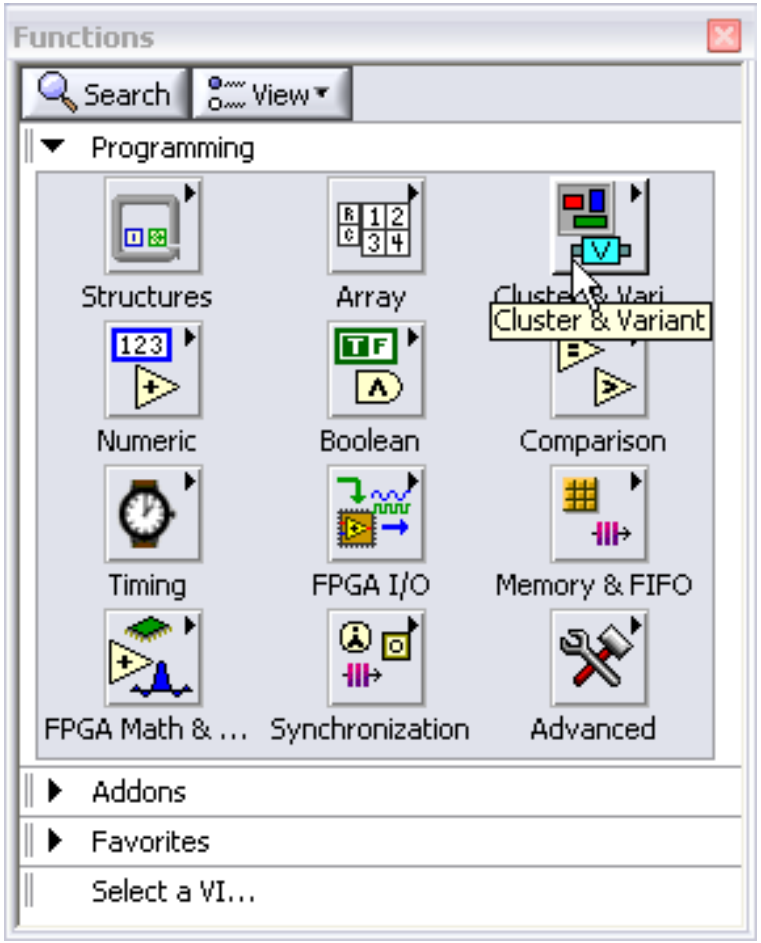
D. Developing the FPGA VI

Add a VI under the FPGA target



D. Developing the FPGA VI

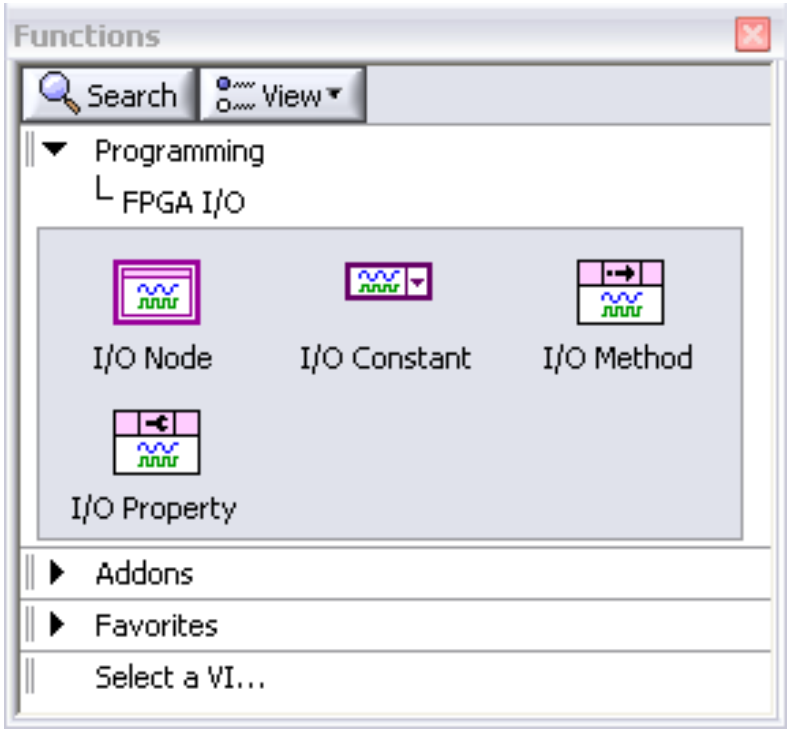
- VIs under the FPGA target inherit the FPGA Function Palette.
- Many palettes are similar to LabVIEW for Windows except the FPGA I/O and FPGA Math & Analysis Palettes.



D. Developing the FPGA VI

FPGA I/O palette

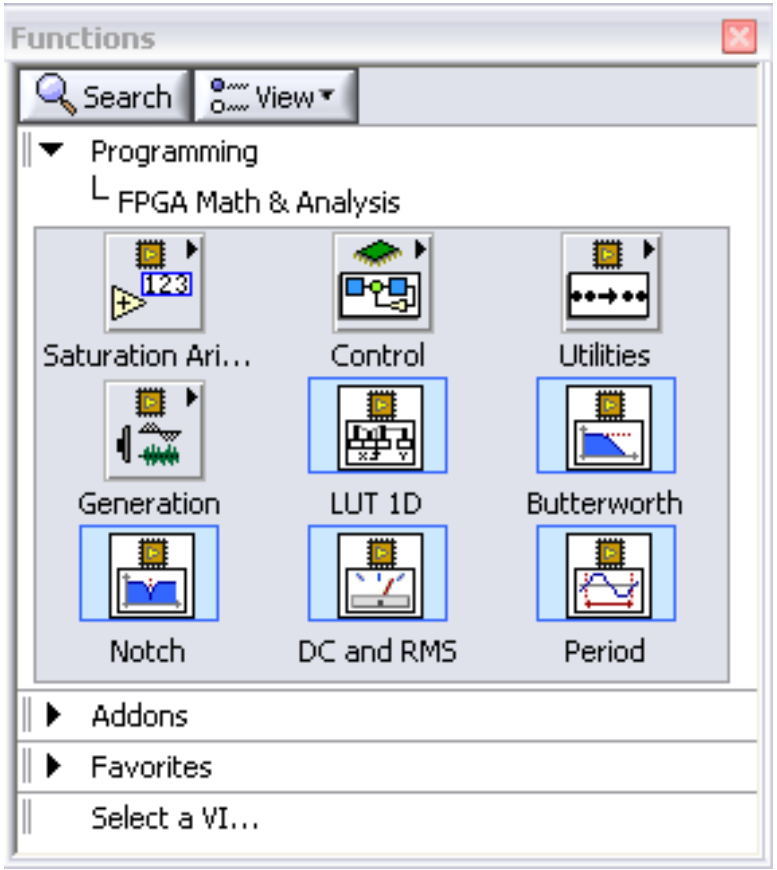
- Provide communication with I/O modules.



D. Developing the FPGA VI

FPGA Math & Analysis

Keep calculations as simple as possible to preserve space on FPGA



D. Developing the FPGA VI

FPGA I/O items connect I/O to the FPGA logic

Each FPGA I/O item has a type like analog or digital

FPGA VIs can have multiple types of I/O items

D. Developing the FPGA VI

FPGA I/O terms:

- Terminal – a hardware connection on a CompactRIO module
- Channel – a logical representation in LabVIEW FPGA of a terminal
- Alias – a name assigned by the developer to a particular channel

D. Developing the FPGA VI

Digital I/O:

- Individual Lines – Boolean data type
- Ports (collection of 8 lines) – U8 data type (1 bit per line)
- Digital I/O can write or read. Disable with I/O Method Node before switching between I and O.

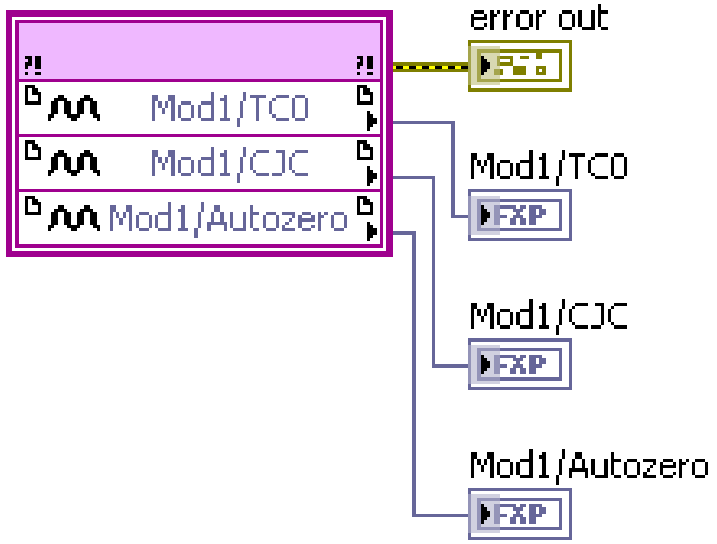
D. Developing the FPGA VI

Analog I/O

- Data Type – I32
- AI – Converts binary representation of the voltage to a signed integer.
- AO – Writes the binary representation of the voltage to the D/A converter.

D. Developing the FPGA VI

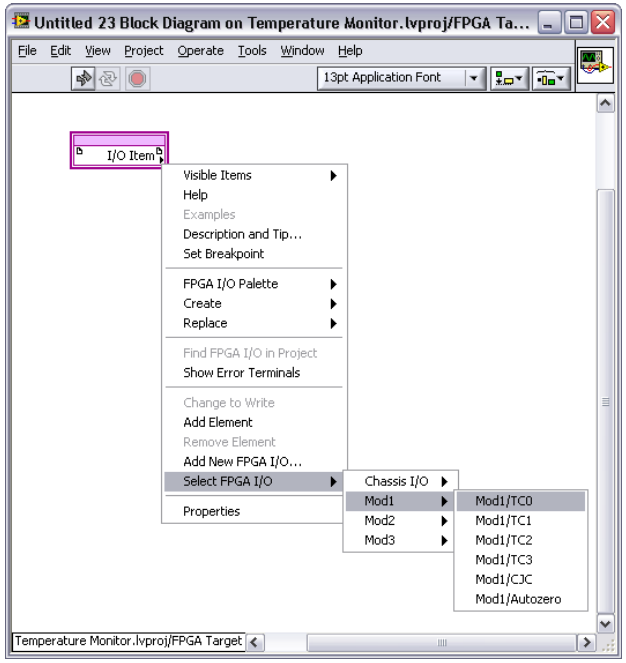
Error Handling

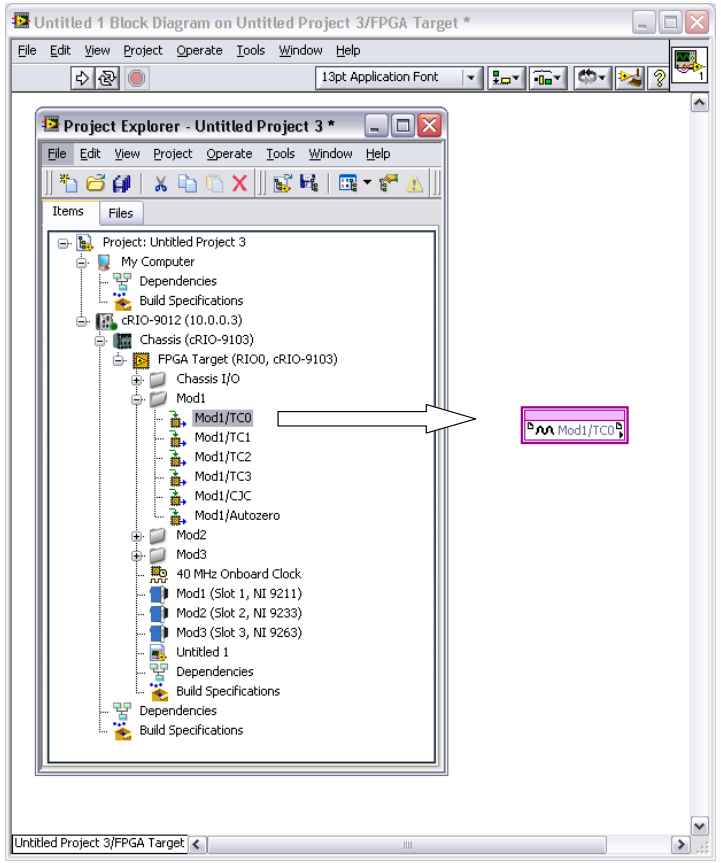


- Right-click the I/O Node to show error terminals
- Error information is useful for debugging and validating data, but it uses space on the FPGA and can slow execution

D. Developing the FPGA VI

Place I/O Nodes from palette or Project Explorer



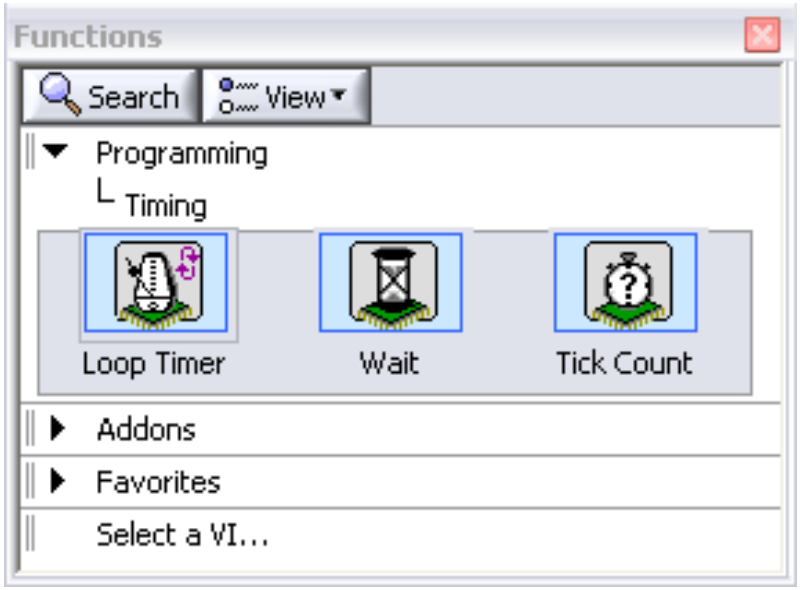


D. Developing the FPGA VI

Loop Timer Function
- Times loop rate

Wait
- Adds explicit delay
- Pulse length control

Tick Count
- Returns current value of FPGA clock.
- Benchmark loop rates
- Create custom timers

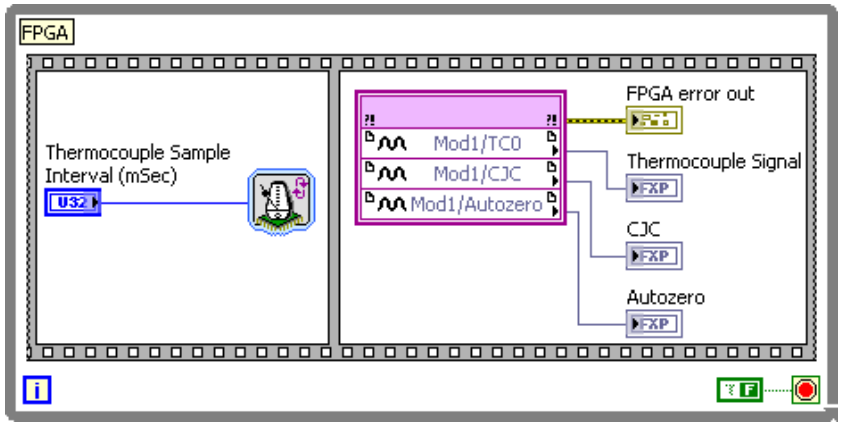


D. Developing the FPGA VI

Loop Timing with Loop Timer Function

- Records current time as initial time during first iteration
  - No delay for Loop Timer,

    then reads channels
- Second execution adds timer count to initial time and waits until count has elapsed
  - Subsequent iterations match desired timing synchronization.



D. Developing the FPGA VI

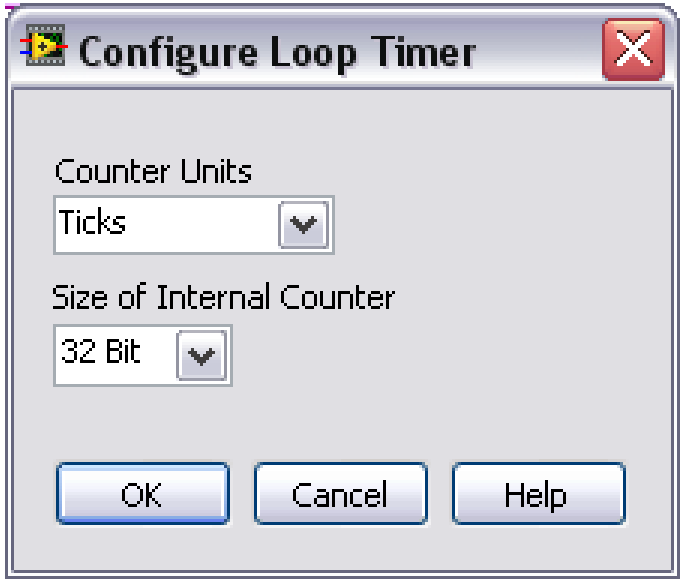
Configure Loop Timer

Counter Units:

Ticks — clock cycles (40 MHz)

µSec — microseconds

mSec — milliseconds

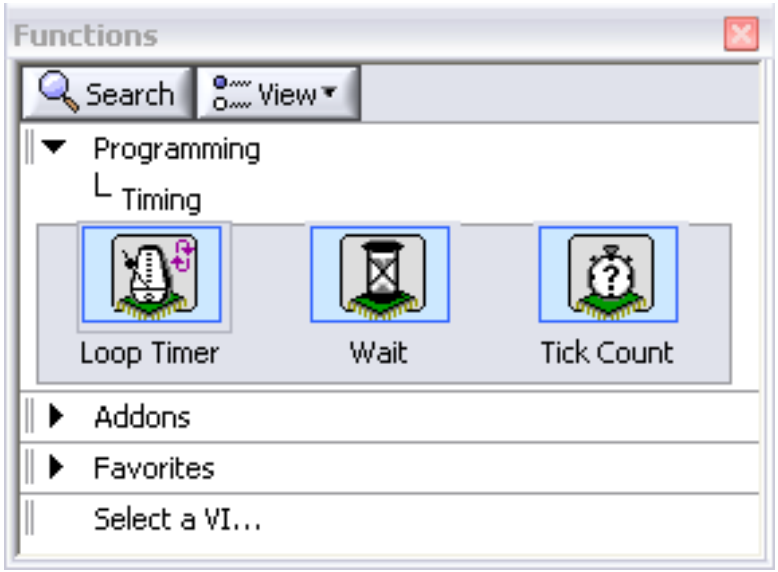


Size of Internal Counter – 8, 16, or 32 bit = maximum time a timer can track. To save space on the FPGA, use the smallest Size of Internal Counter possible.

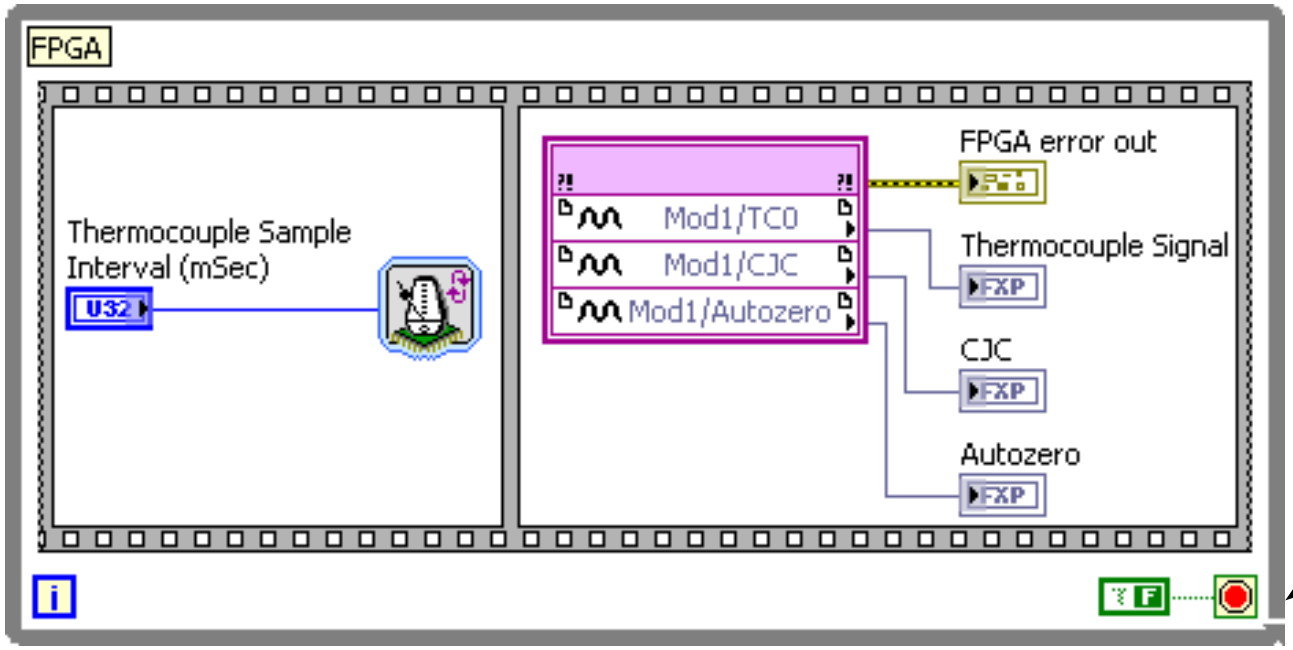
D. Developing the FPGA VI

Loop Timer function vs. Wait function

- Loop Timer function allows code to execute right away during the first iteration
- Wait function delays the code execution.



D. Developing the FPGA VI



Host VI starts and stops FPGA

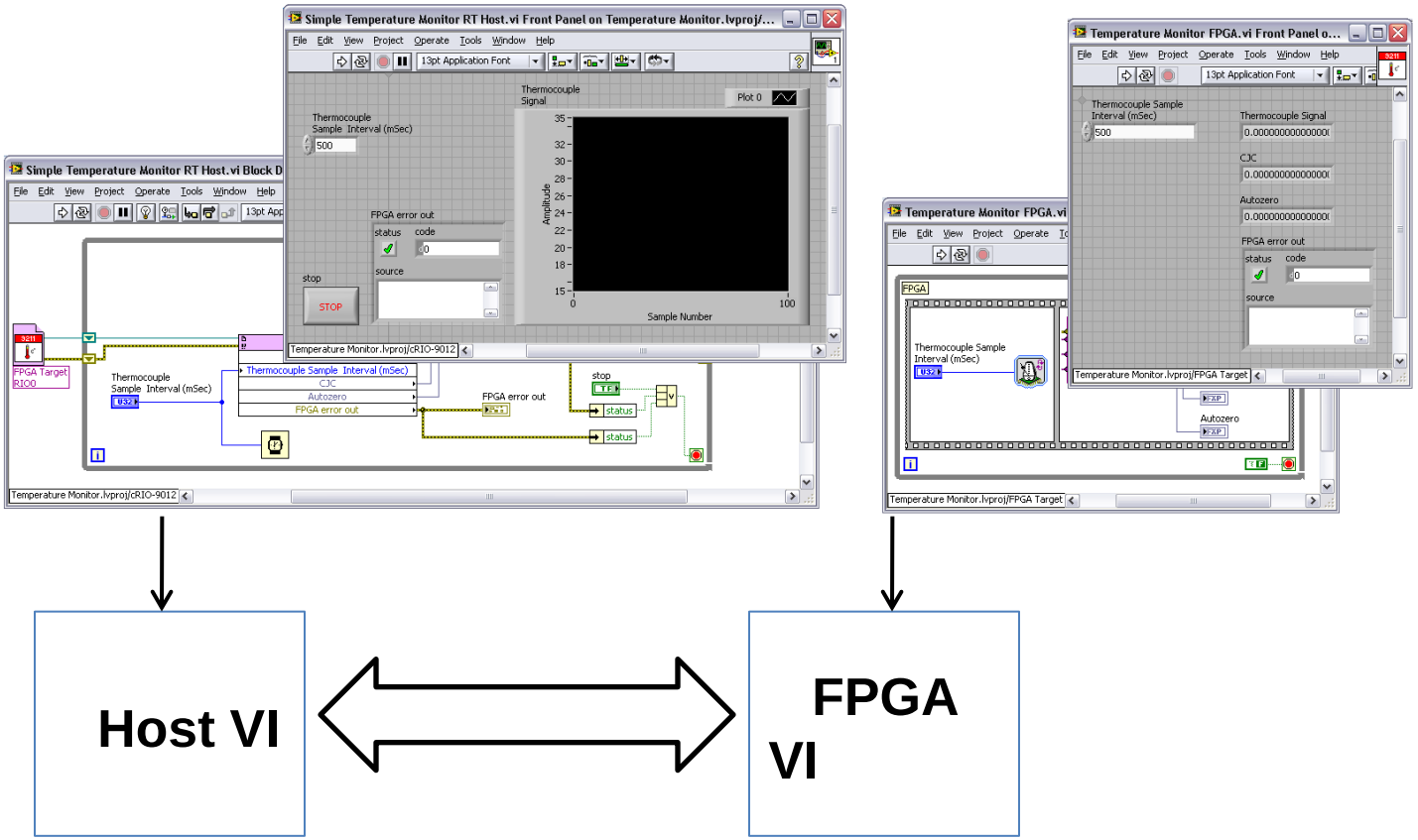
D. Developing the FPGA VI

When designing an FPGA VI, consider communications with an RT or Windows host. Host FPGA Interface functions allow you to:

- Establish and terminate communication with the FPGA.
- Download, abort, reset, and run the FPGA.
- Read and write data to the FPGA.
- Wait for and acknowledge FPGA interrupts.
- Read DMA FIFOs (direct memory access first-in first-out buffers).

D. Developing the FPGA VI

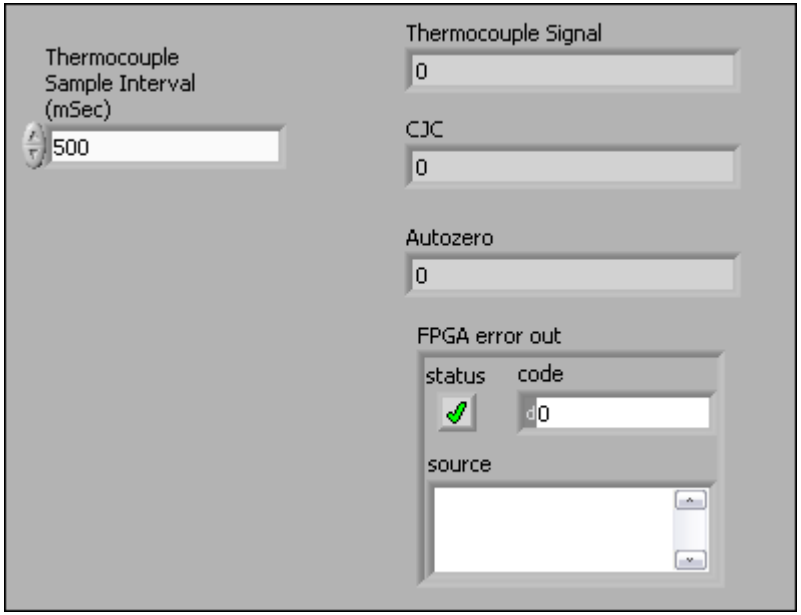
Programmatic Front Panel Communication



D. Developing the FPGA VI

FPGA Front Panel

- FPGA has limited memory
- Use simple controls and indicators
- Use the minimum necessary controls and indicators for programmatic Front-panel communication to a host
- Add temporary controls and indicators for debugging if necessary, but remove them



D. Developing the FPGA VI

Modular Code

- Segment code into function blocks (subVIs)
- Develop and test blocks independently
- Takes advantage of parallel execution ability of FPGA

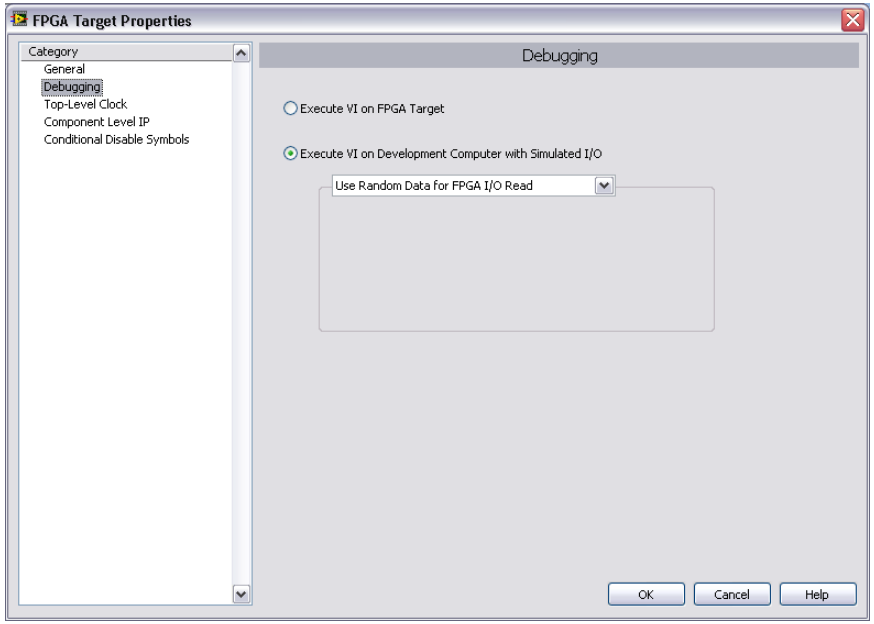
E. Testing with the Development Machine

- Compiling – few minutes to several hours
- Verify logic before compiling
- LabVIEW bit-accurate emulation mode
- The emulator generates random data
- Executes logic on the Windows PC
- Traditional debugging tools are available

E. Testing with the Development Machine

FPGA Target Properties dialog box

- Right-click the FPGA Target in the Project Explorer window



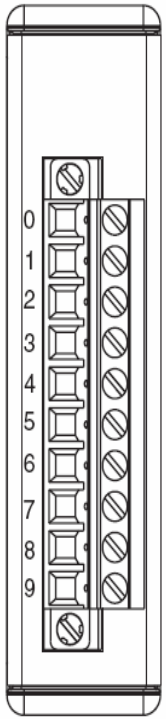
When testing is complete, set the FPGA Target to execute VI on the FPGA target.

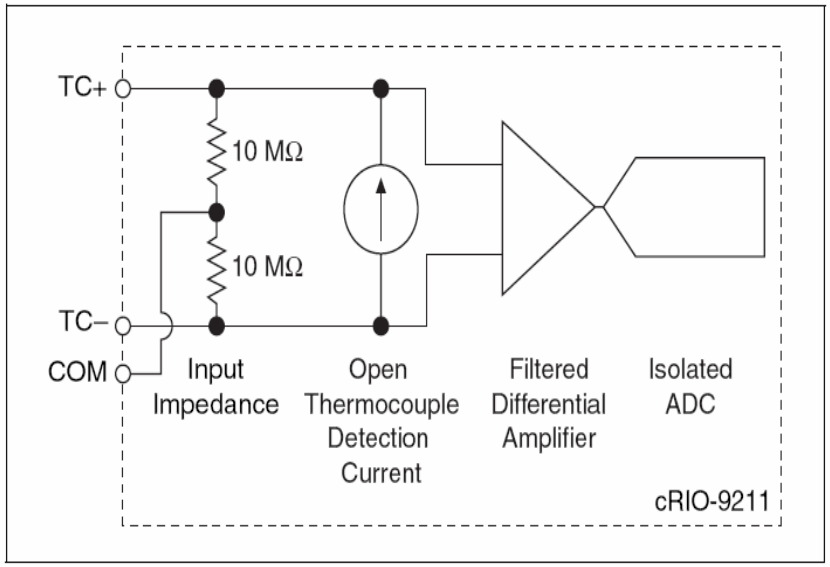
F. Interactive Front Panel Communication

FPGA has no user interface

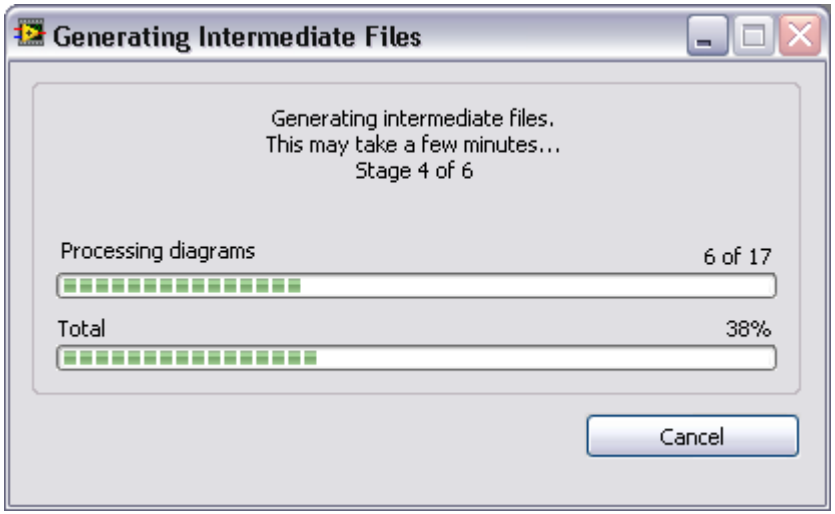
- Must communicate data from FPGA to Host PC
  - Requires no additional programming
- Front Panel displayed on Host PC
- Block Diagram executes on FPGA as compiled
- Communication layer shares all control and indicator values
- Cannot use debugging tools when running FPGA VI
  - Test with Emulator first, or add indicators as probes

G. Wiring the Modules

Module	Terminal	Signal
	0	TC0+
	1	TC0–
	2	TC1+
	3	TC1–
	4	TC2+
	5	TC2–
	6	TC3+
	7	TC3–
	8	No connection
	9	Common (COM)

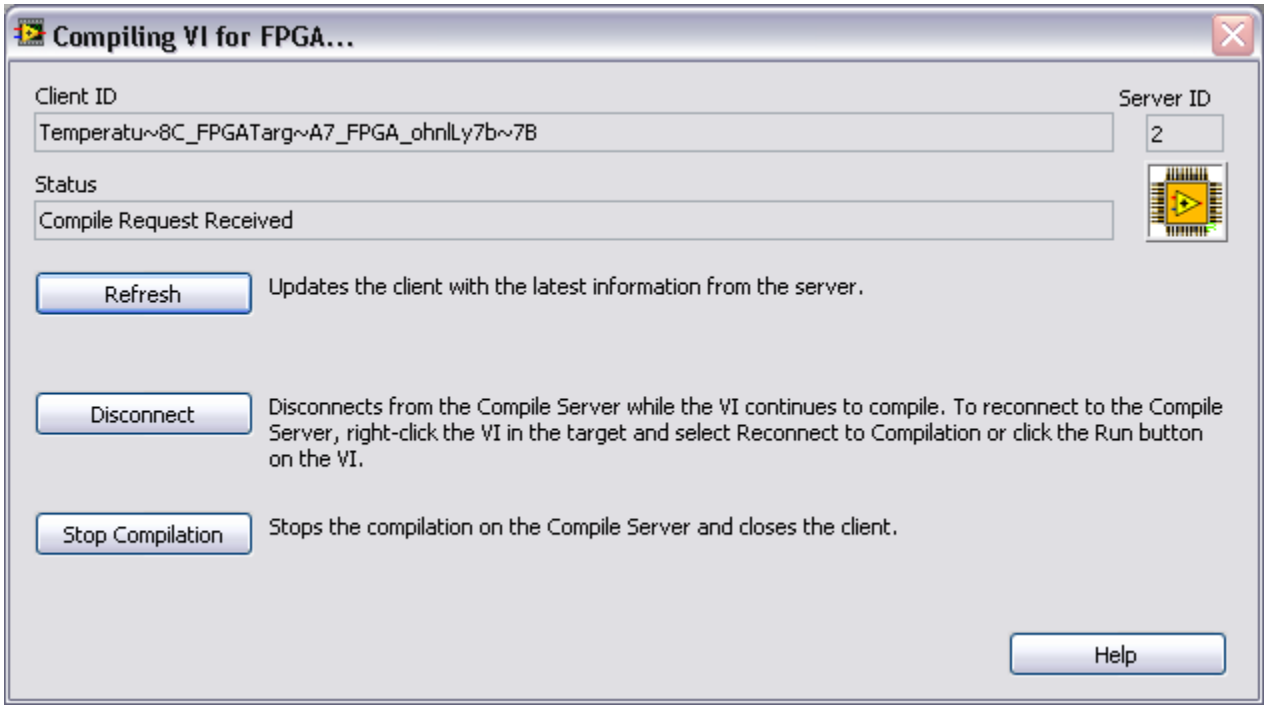


H. Compiling the FPGA VI



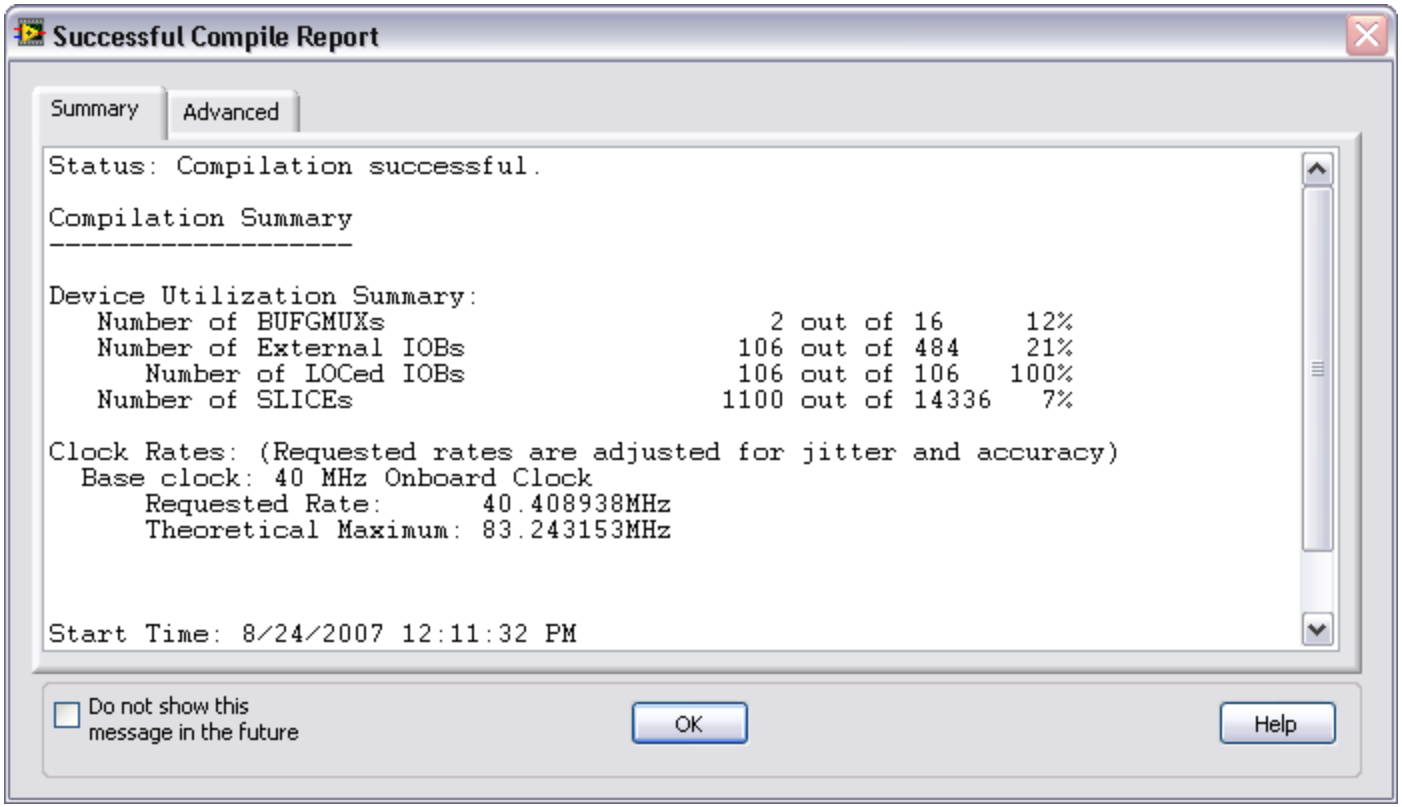
– Click the Run button
– Converts graphical code to VHDL
– Generates intermediate files

H. Compiling the FPGA VI



- Disconnect – Disconnects from the Compile Server to continue working in LabVIEW
- Run the VI again to reconnect

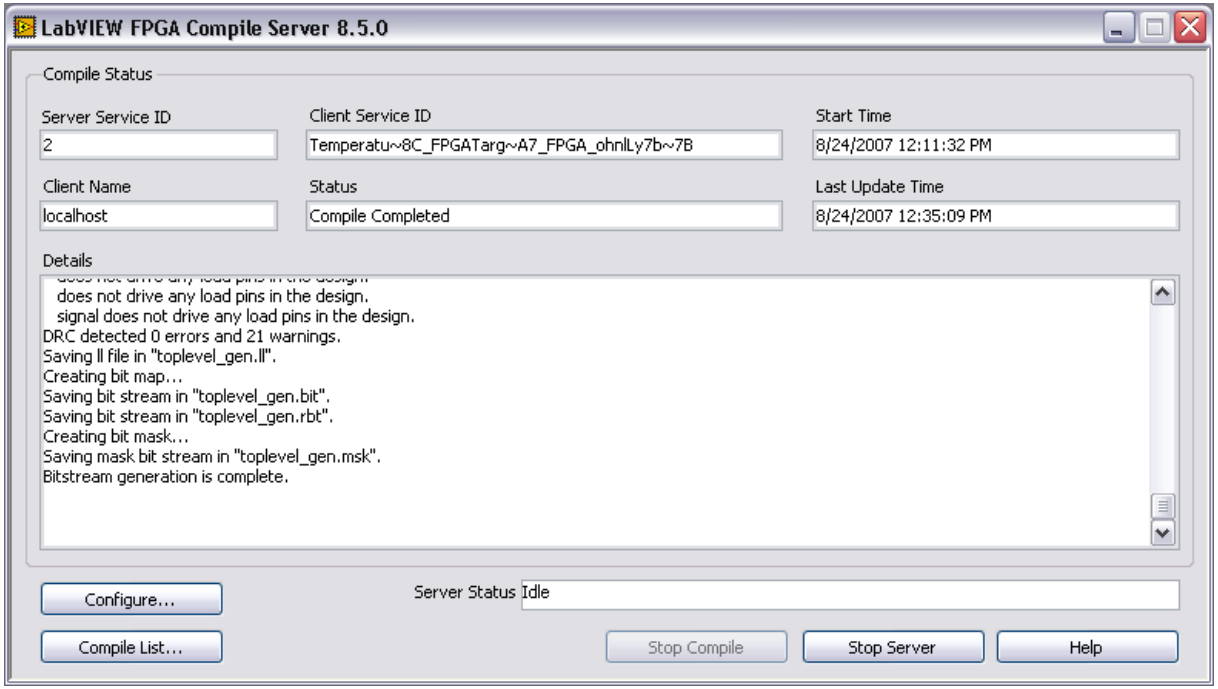
H. Compiling the FPGA VI



H. Compiling the FPGA VI

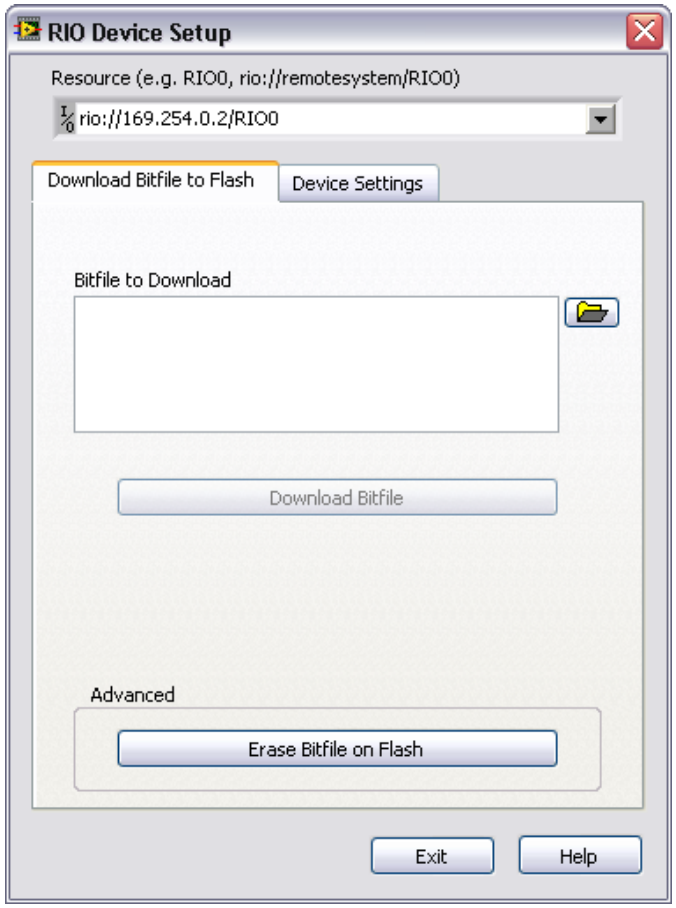
View Previous Compile Reports Tools»FPGA Module»
Start Local Compile Server

Click the Compile List button



I. Downloading to Flash Memory

– Configure the target – automatic load at power up



I. Downloading to Flash Memory

Run when loaded option

1. Right-click the FPGA target in the Project Explorer window and select Properties from the shortcut menu. The FPGA Target Properties dialog box appears.
2. Place a checkmark in the “Run when loaded to FPGA” checkbox.
3. Click the OK button. All FPGA VIs you create for the FPGA target will run when loaded.
4. Configure the FPGA target to load the VI automatically when powered on.

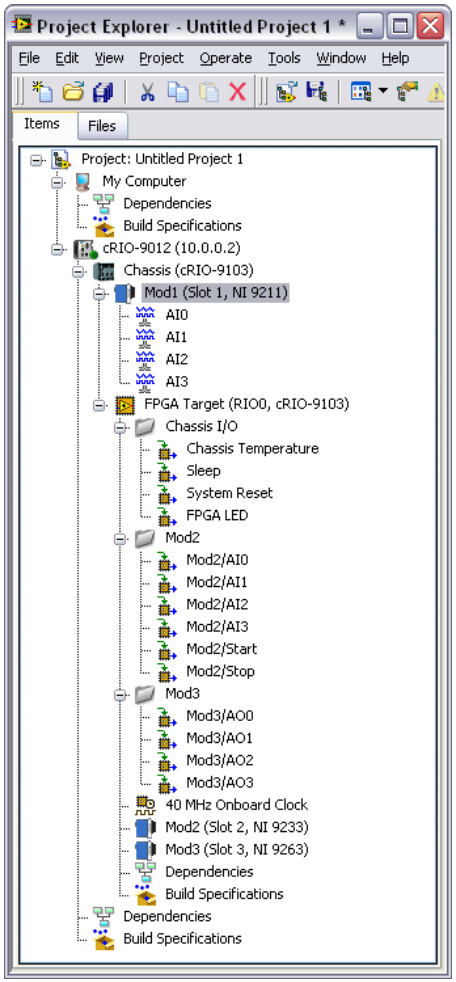
J. Using LabVIEW FPGA with CompactRIO Scan Mode

What if you want to share the I/O workload between the FPGA and the real-time processor?

- Need FPGA for signal processing on some channels but only single-point acquisition from others?
- Using some modules that aren't supported by scan mode with others that are?

J. Using LabVIEW FPGA with CompactRIO Scan Mode

- FPGA mode can be used in conjunction with the scan mode on a per-module basis
- Modules can be moved from FPGA mode to scan mode in the LabVIEW project
  - Drag from the FPGA target to the RT target and vice versa.



J. Using LabVIEW FPGA with CompactRIO Scan Mode

Compiling when using both FPGA mode and scan mode

- Scan mode logic will be compiled along with the LabVIEW FPGA VI into a single FPGA application
- Space used by the RSI on the FPGA scales with the number of modules using scan mode
  - If there are no modules running in scan mode, then RSI will not be included in the compile

J. Using LabVIEW FPGA with CompactRIO Scan Mode

When to use FPGA mode instead of scan mode

- Analog acquisition at rates above 1 kHz
- High-speed PID control loops running faster than 1 kHz
- Custom hardware analysis is required
- Hardware-based signal processing is required
- I/O modules used are not supported by scan mode
  - e.g. DSA modules
- Need to offload processing from the real-time controller

J. Using LabVIEW FPGA with CompactRIO Scan Mode

Performance tradeoffs of using scan mode instead of FPGA

- Specialty digital functionality in scan mode only supports up to 1 MHz counters vs the 20 MHz counters achievable on FPGA
- NI Scan Engine uses system resources
  - FPGA space requirements
  - Minimum of two DMA channels needed
  - System memory resources used
  - CPU time scales with the scan period

Lekcia 6

REAL-TIME CONTROLLER

Real-Time Controller

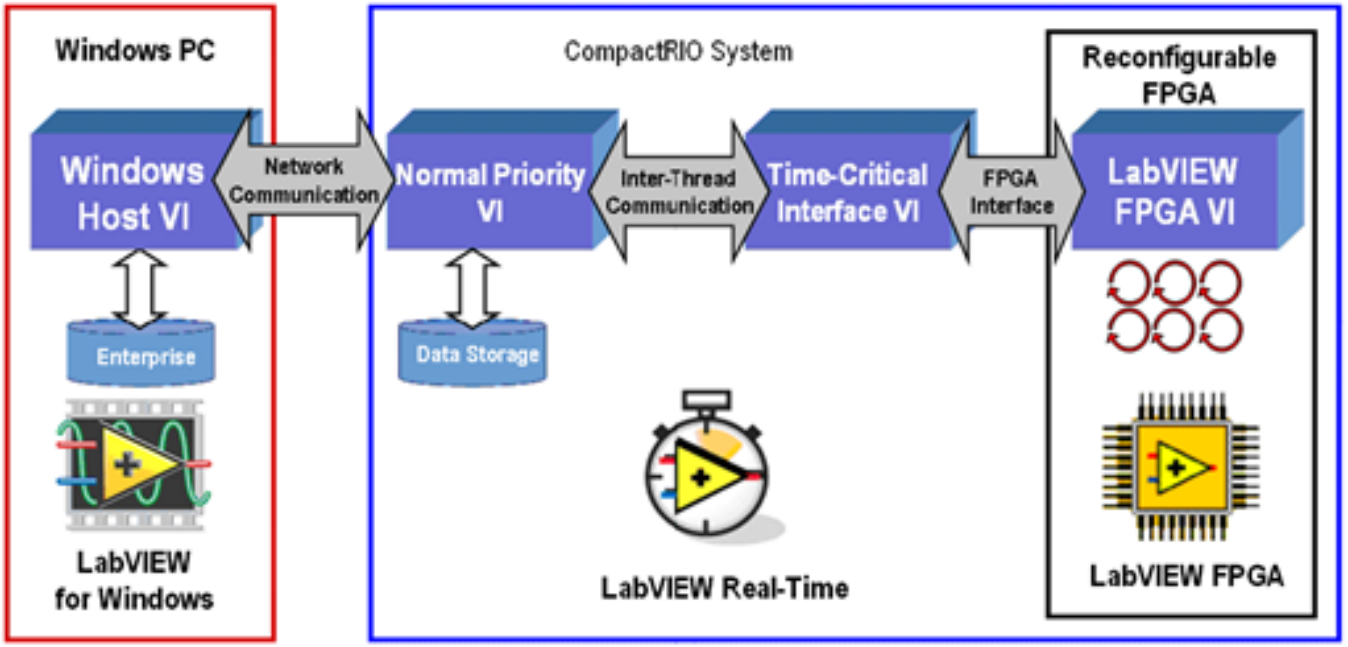
- A. Introduction
- B. LabVIEW Real-Time Applications
- C. Development Course
- D. Deterministic Operating Systems
- E. Timing Methods

E. Developing an RT Host VI

F. Rebooting the CompactRIO RT Controller

E. Reusing Code in Multiple Targets

A. Introduction



A. Introduction

- RT VI common tasks:
  - Data processing
  - Perform operations not available on the FPGA target
  - Log data
  - Run multiple VIs
  - Control the timing and sequencing of data transfer

Common RT Loop Types

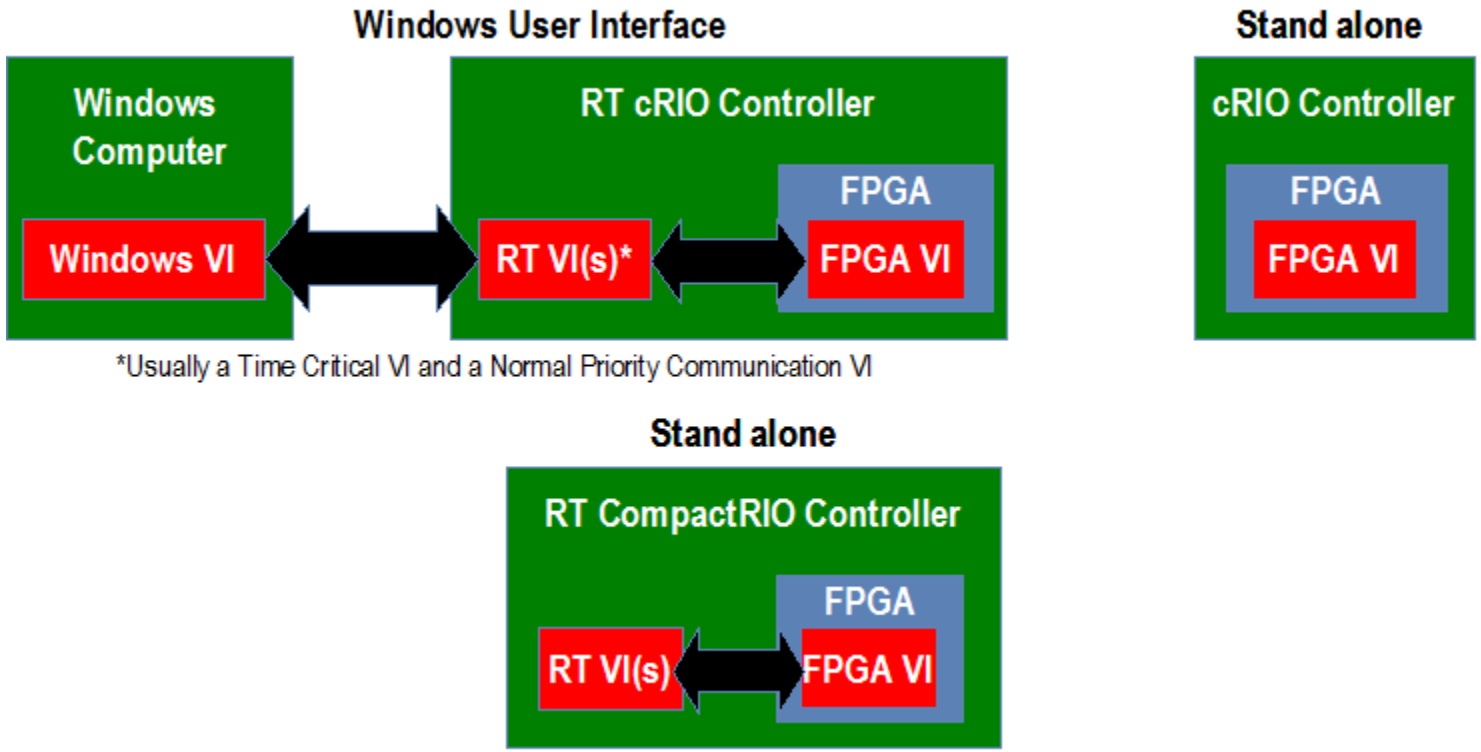
A time-critical loop (higher priority)

- floating-point control
- signal processing
- analysis
- point-by-point decision making

Normal-priority loop (executes when time-critical loop waits)

- embedded data logging
- remote panel Web interface
- Ethernet/serial communication

A. Introduction



Common RT Configurations

B. The LabVIEW Real-Time Application Development Course

- Configuring PXI and a Compact FieldPoint system
- Real-time application design
- Multithreading
- Passing data between threads
- Improving determinism
- Priority levels
- Memory management
- Timed structures
- Event response
- TCP/IP communications
- Verifying timing and memory usage
- Execution Trace Tool Kit
- Deploying an application
- Remote panels
- RT Wizard

C. Deterministic Operating Systems

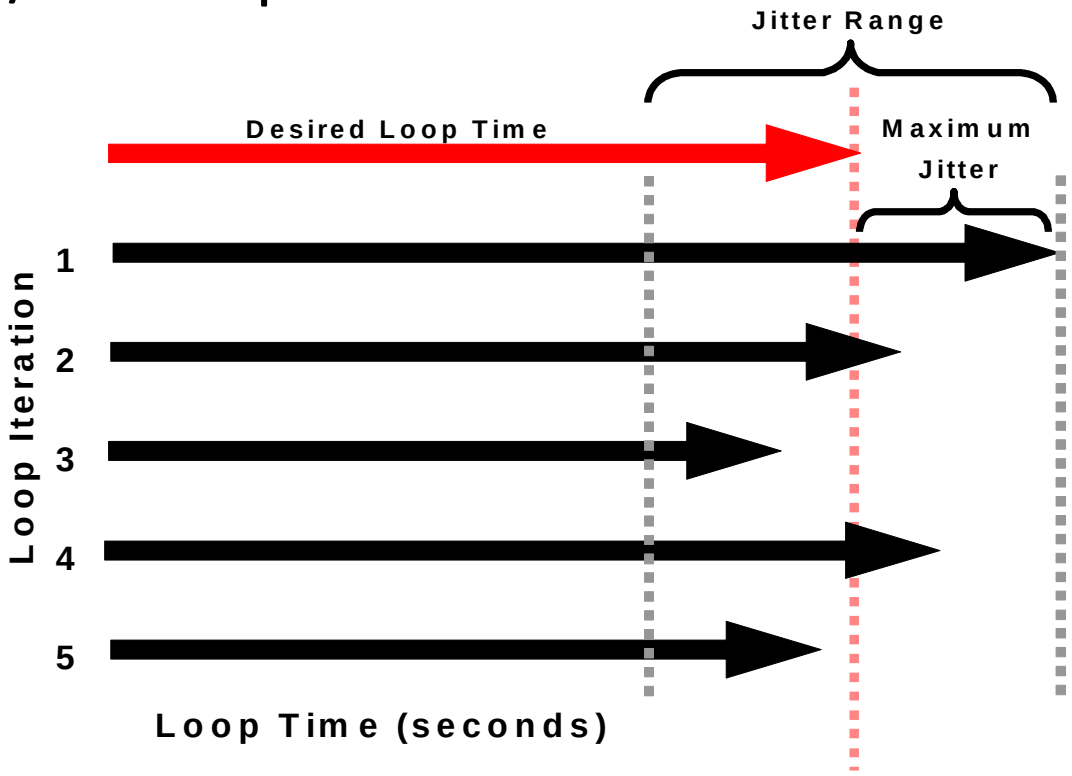
- Ardence Phar Lap Embedded Tool Suite (ETS) real-time or Wind River's VxWorks
- LabVIEW Real-Time Module to develop VIs
- MS Windows is not a real-time OS.

Deterministic: responds reliably to an event, or performs an operation, within a guaranteed time period.

C. Deterministic Operating Systems

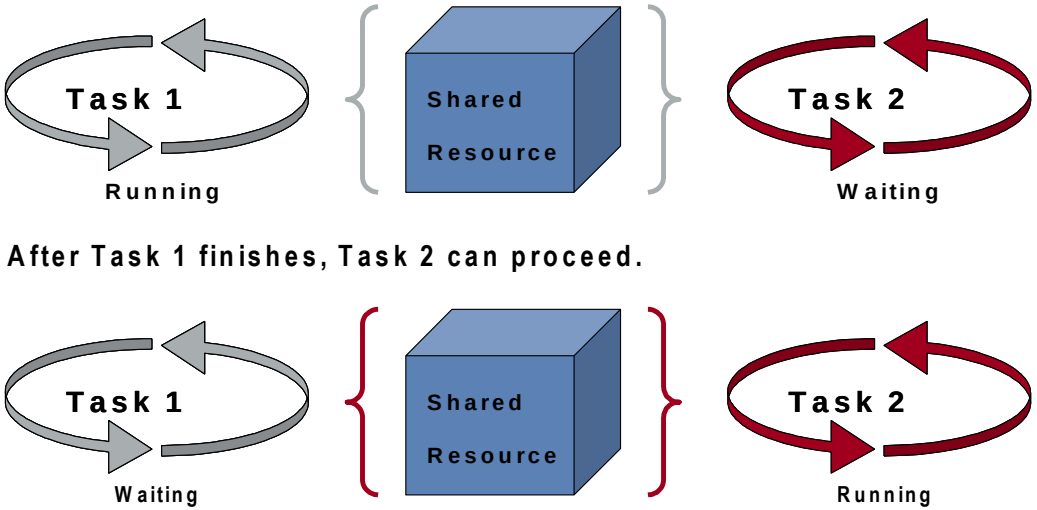
Jitter = Variation in Loop Cycle Time

T = Loop Cycle Time. 1/T = Loop Rate



C. Deterministic Operating Systems

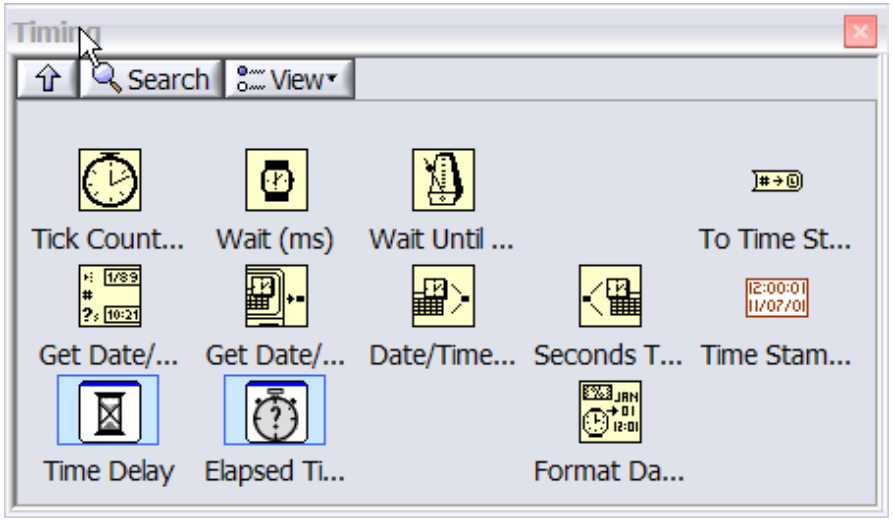
Some shared resources can only be accessed by one process at a time. These resources can cause jitter.



- Examples: I/O, memory (variables and FIFOs), non-reentrant VIs
- FPGA does not need to share resources – jitter can be as low as 250 picoseconds

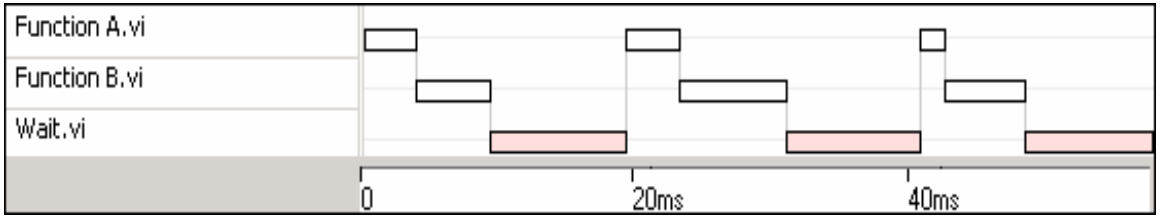
D. Timing Methods

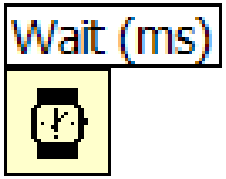
- Deterministic tasks can consume all processor resources.
- Use Wait (ms) and Wait Until Next ms Multiple VIs to pause deterministic loops and allow non-deterministic tasks to execute.



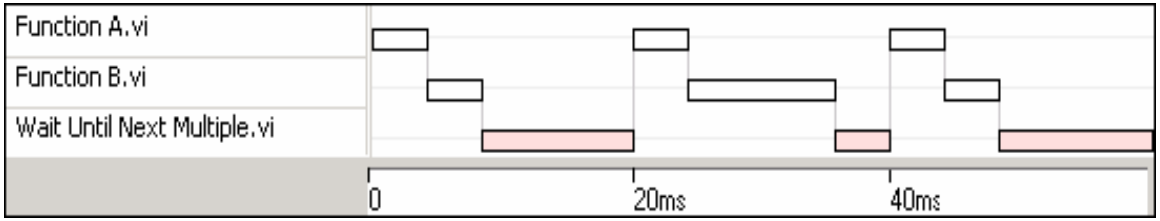
D. Timing Methods

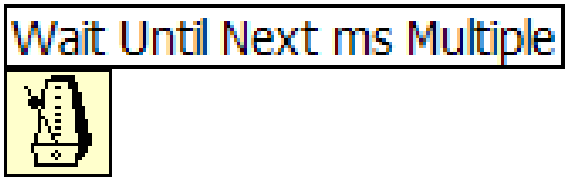
- Wait VI – Constant time of execution
- Execute A, Execute B, sleep 10 ms





- Wait Until Next Multiple VI – Variable time of execution
- Execute A, Execute B, sleep until OS timer reaches next multiple of 20 ms

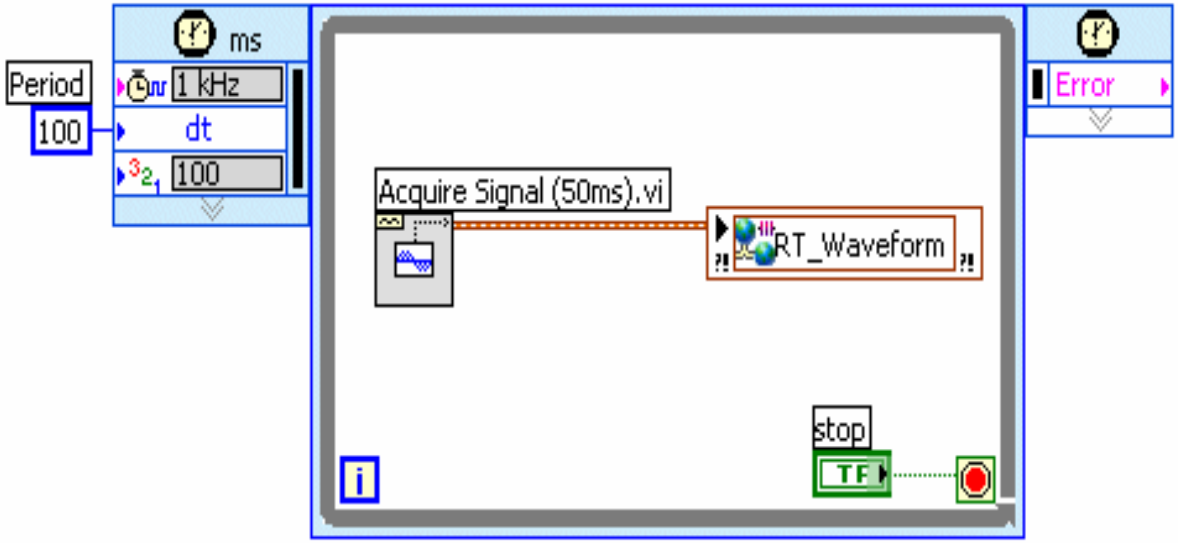




D. Timing Methods

Timed Structures – Timed Loop and Timed Sequence

– Priority Configurable

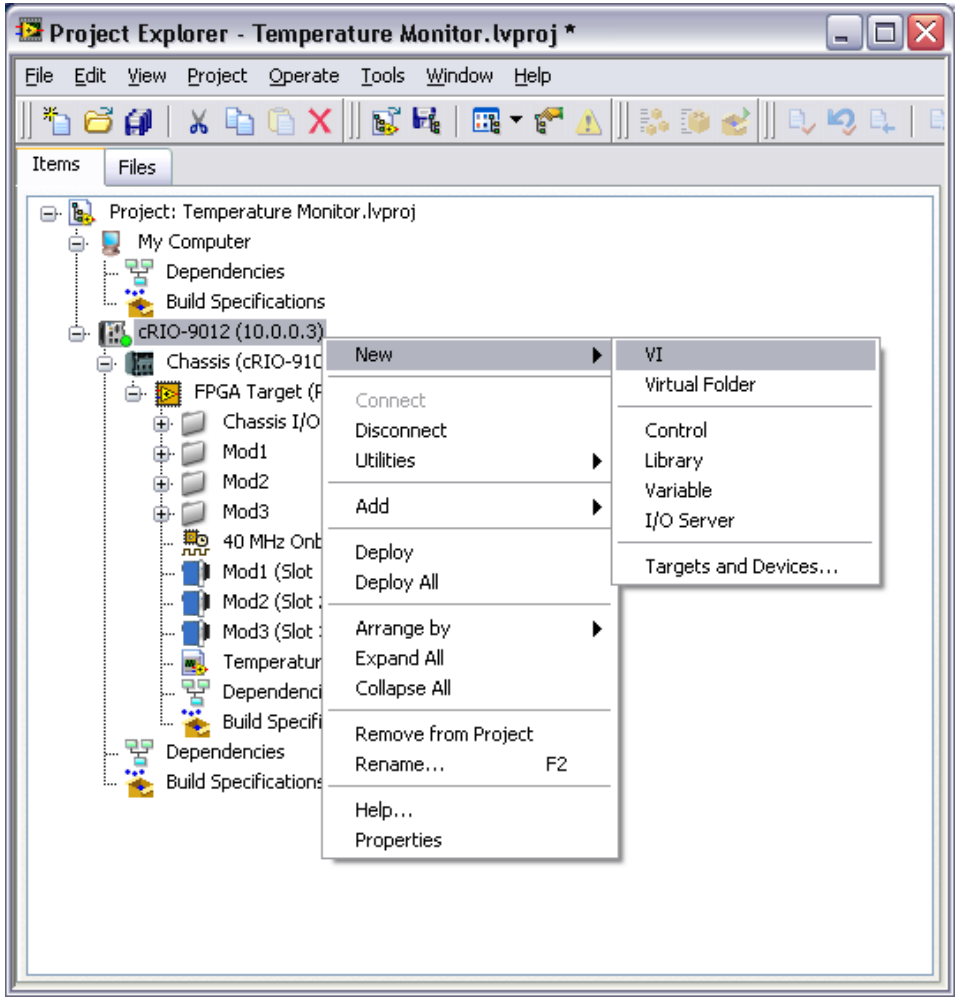


Period = 100 ms, Acquire = 50 ms, Loop idle = 50 ms

LabVIEW executes lower priority tasks during idle time.

E. Developing an RT Host VI

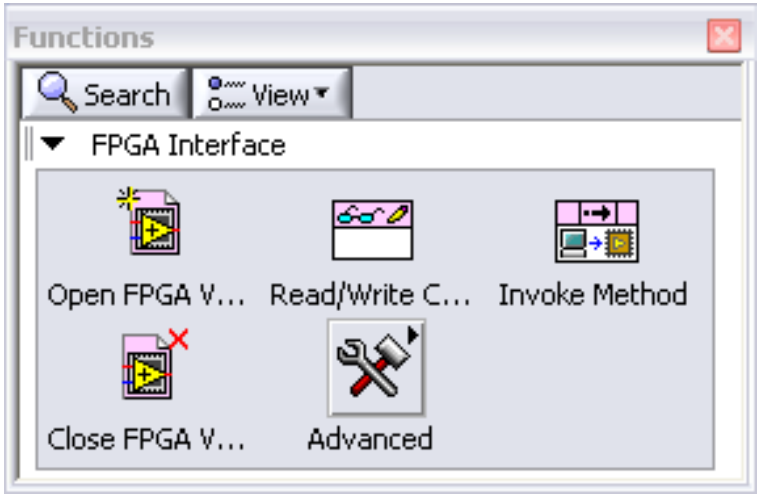
Add a VI under the cRIO target



E. Developing an RT Host VI

FPGA Interface Functions Palette

- Establish and terminate communication with the FPGA VI.
- Download, abort, and run the FPGA VI on the FPGA target.



- Read and write data to the FPGA VI.
- Wait for and acknowledge FPGA VI interrupts.
- Read DMA FIFOs.

E. Developing an RT Host VI

Open FPGA VI Reference

- Opens a reference to:
  - the FPGA VI or
  - bitfile
  - and FPGA target
- Select VI and target in the shortcut menu
  - Or, specify target with the resource name input
    - Right-click and choose show resource name input
- Must open a reference to the FPGA target before you can communicate between the host VI and the FPGA VI.
- Configure for Open or Open and Run
  - Use free label to describe functionality

E. Developing an RT Host VI

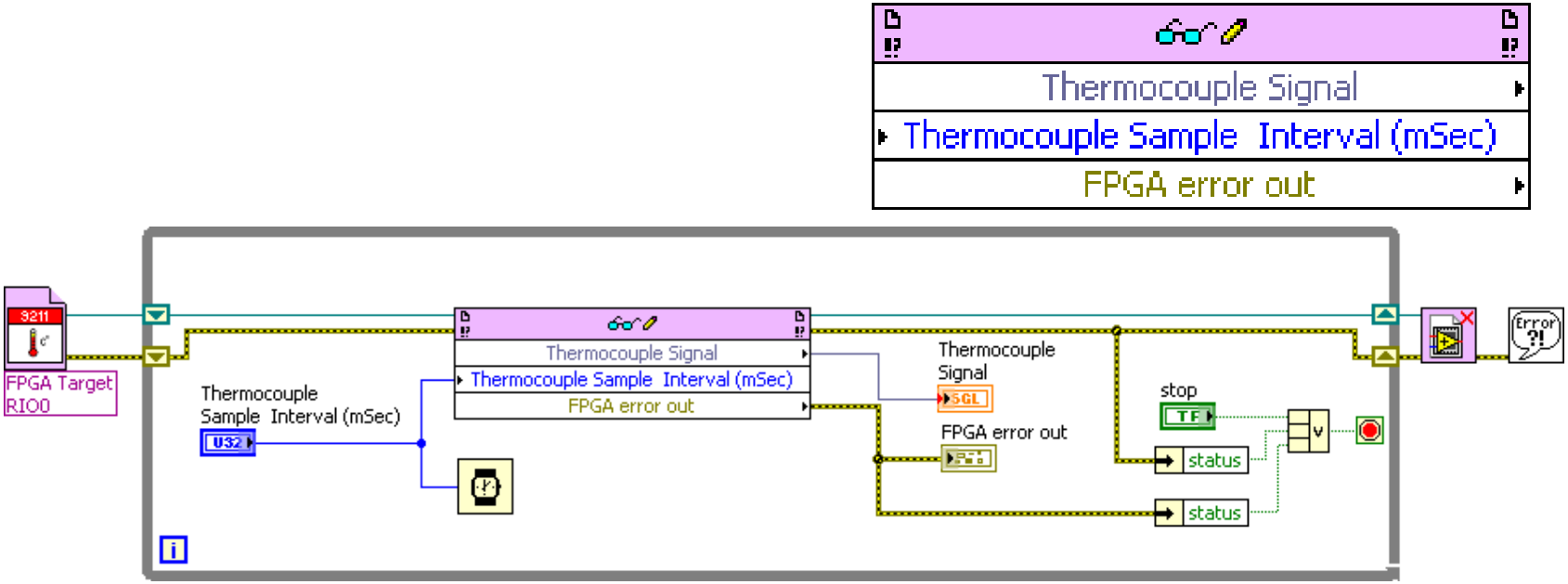
Invoke Method

- Invokes method or action from a host VI
- Methods:
  - download
  - abort
  - run
  - wait for and acknowledge interrupts,
  - read DMA FIFOs
  - write DMA FIFOs

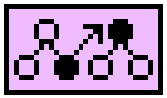
E. Developing an RT Host VI

Read/Write Control

– Reads a value from or writes a value to a control or indicator in the FPGA VI on the FPGA target.

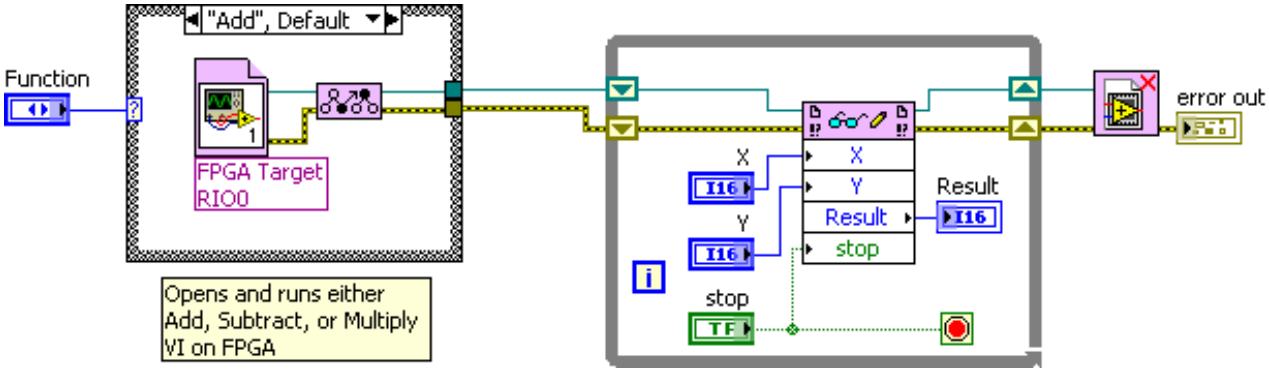


E. Developing an RT Host VI

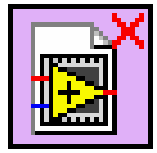


Upcast

- Converts an FPGA VI-specific reference to a more generic reference
- Enables using common code to interact with different FPGA VIs. The FPGA targets must be of the same class.
- You cannot use this function with DMA FIFOs



E. Developing an RT Host VI



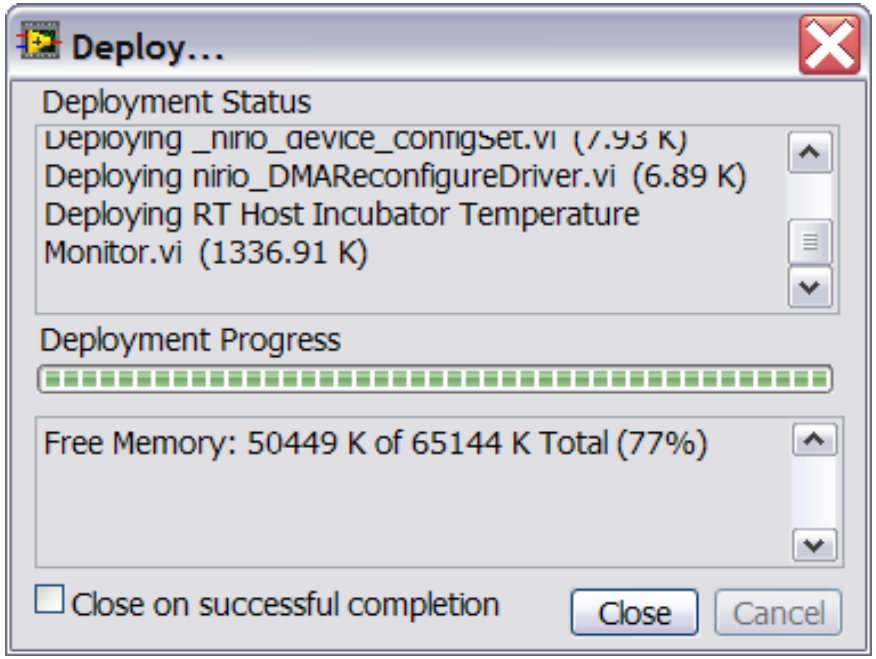
Close FPGA VI Reference

- Stops and resets the FPGA
- Right-click and select Close from the shortcut menu to close the reference without resetting
- Default is Close and Reset, which closes the reference, stops the FPGA VI, and resets the FPGA
- Use free label to describe functionality

E. Developing an RT Host VI

After developing the RT host VI, run it

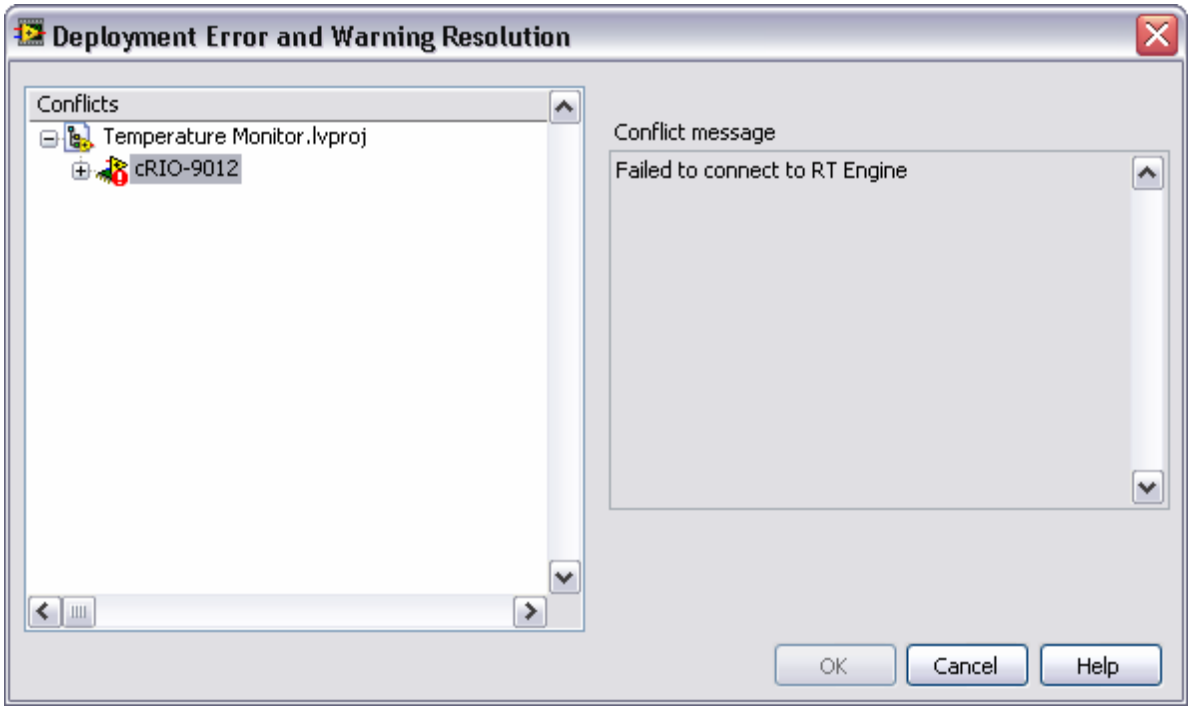
– Deploying Status Window:



F. Rebooting the CompactRIO RT Controller

Reboot the CompactRIO

- Right-click target
- Choose Utilities»Reboot
- Wait for Power-up sequence



G. Reusing Code in Muliple Targets

- NI Example Finder – Toolkits and Modules » FPGA » CompactRIOExamples configured for another target. Copy code to your target.
  - Open a project where the hardware has been configured.
  - Open a new project.
  - Copy (drag) the cRIO target to the new project.
  - Save the new project and close the original project.

G. Reusing Code in Muliple Targets

Continued…

- Open the NI Example Finder project, and drag the FPGA VI, the RT host VI, and all other items in the example project to appropriate locations in the new project.
- Remove any unneeded VIs from the original project.
- Save the new project.
- Update the FPGA VI references in the RT Host.
- Recompile the FPGA VI.

G. Reusing Code in Muliple Targets

- The new target must support all items.

  - If an item is not supported, it appears with this icon.
  - Unsupported items could be a(n)
    - I/O Resource
    - Clock Resource
    - FIFO
    - Memory
  - Remove or replace the unsupported item.

Lekcia 7

WINDOWS PC HOST

Windows PC Host

- Overview
- Shared Variable Network Communications

A. Overview

- Interactive Front Panel Communication
  - Works between PC and RT controller, like it did between the PC and FPGA
  - Is good for development and debugging
  - Is not deterministic
- PC runs front panel, RT runs block diagram

A. Overview

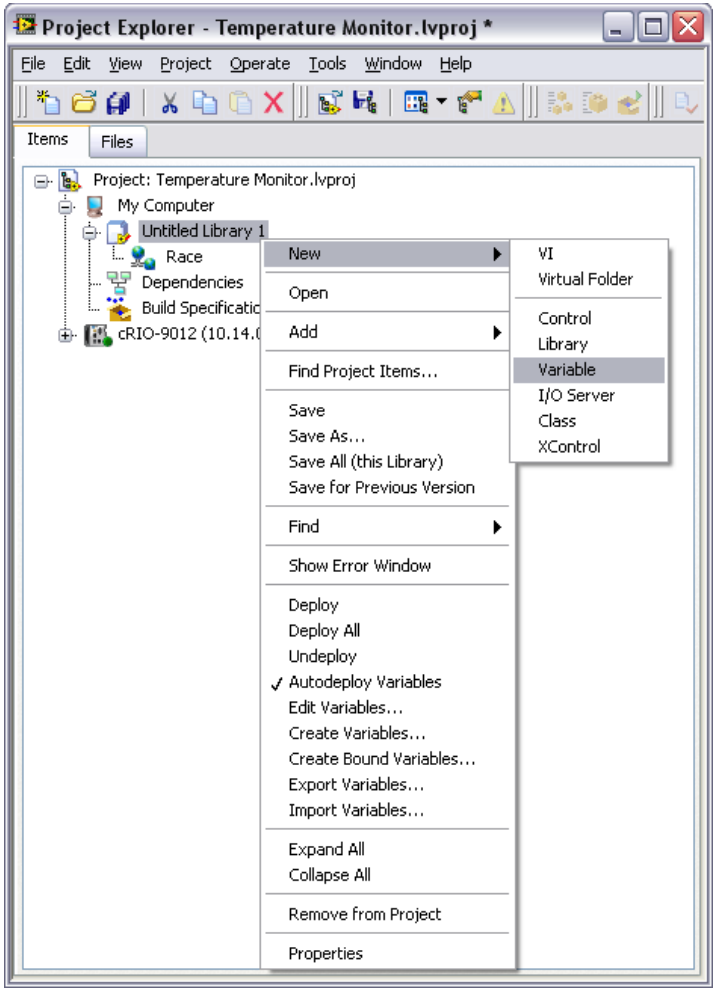
- Use Network Communications for deterministic communication between PC and RT Controller
- Network communication allows you to
  - Run another VI on the host computer.
  - Control the data exchanged
    - which front panel objects get updated and when
    - which components are visible on the front panel
  - Control timing and sequencing of the data transfer
  - Perform additional data processing or logging

B. Shared Variable Network Communication

- Shared variables automatically publish data values over an Ethernet network
- Shared variables communicate between
  - Parallel processes
  - Between VIs
  - Between computers

B. Shared Variable Network Communication

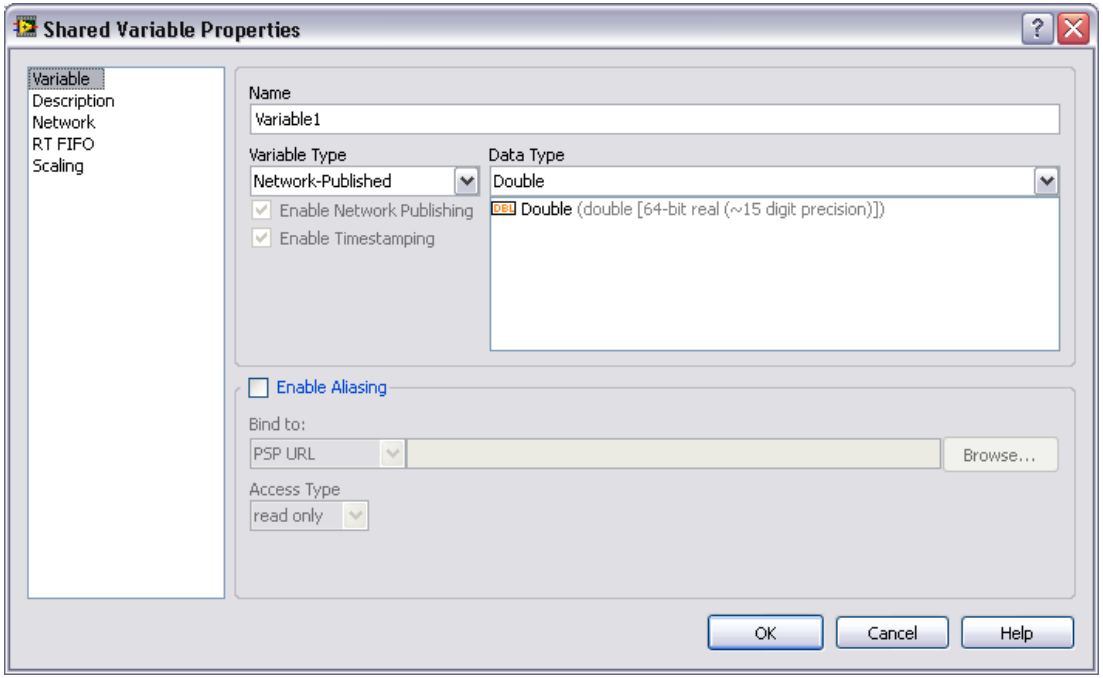
- Create Shared Variables from the Project Explorer window
- Shared Variables must be inside project libraries.
- LabVIEW automatically creates the library with the shared variable inside.



B. Shared Variable Network Communication

Shared Variable Properties Dialog box

- Name
- Data Type
- Variable Type

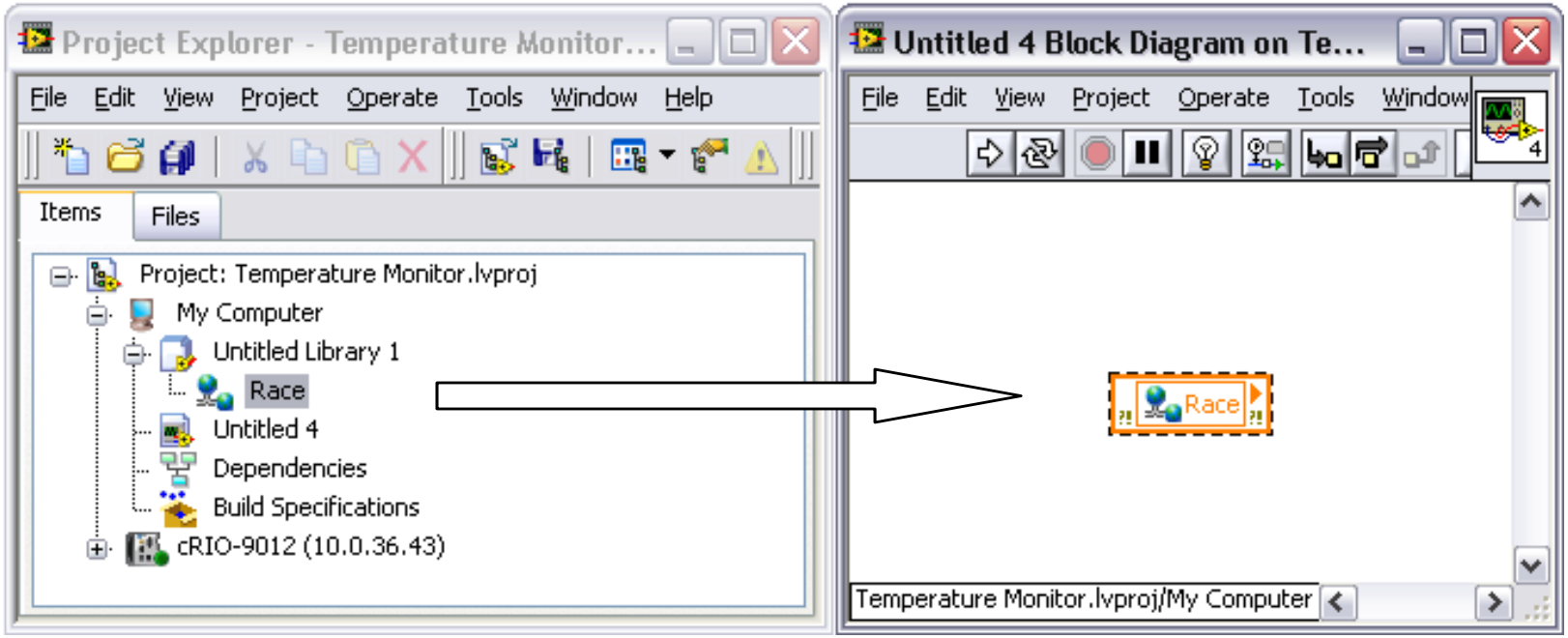


B. Shared Variable Network Communication

The target you right-click will host the variable. There are advantages for hosting on each target.

Variables on Target	Variables on Host
Delta filtering can reduce updates and network traffic for non-buffered shared variable write operations	Less memory and processor use on RT target
	LabVIEW DSC features available
Less chance of engine being unavailable due to crash or user intervention	Other computers reading the data do not use resources on target

B. Shared Variable Network Communication



B. Shared Variable Network Communication

Race Conditions – Timing or events, or scheduling of tasks may unintentionally affect an output or data value.

- Common problem for
  - Parallel execution
  - Sharing data between tasks
- Difficult to debug
  - Can return the same value 1000s of times
  - Capable of returning a different value at any time
- Refer to information presented later on avoiding race conditions

Race Conditions

Buffering and Synchronization

FPGA FIFOs

Handshaking

Interrupts

Direct Memory Access

Hardware-Timed Data Acquisition

Lekcia 8

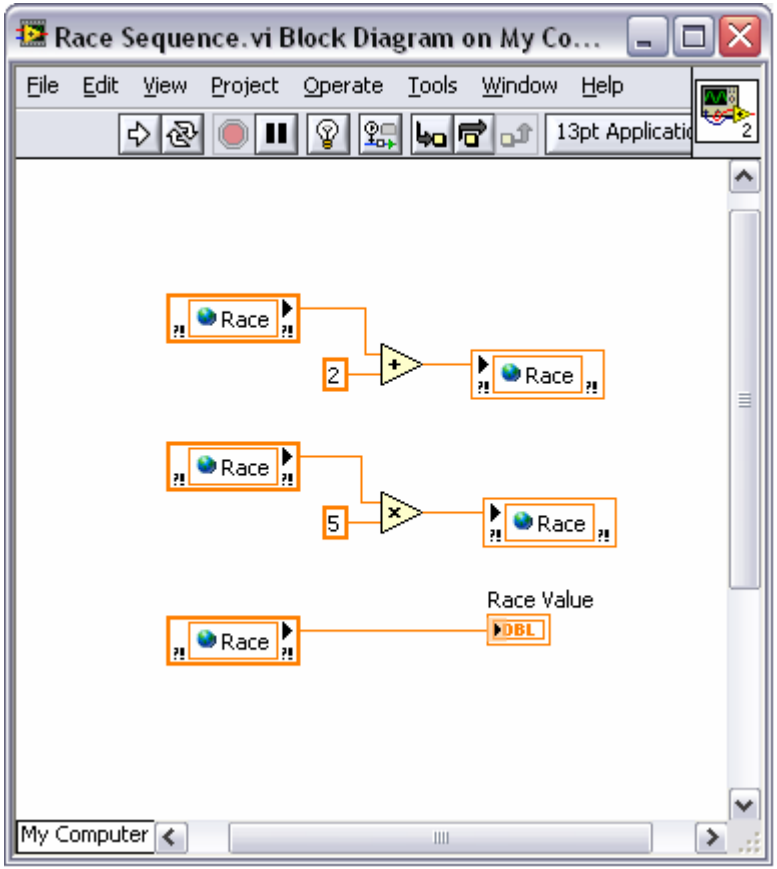
DATA TRANSFER

A. Race Conditions

- FPGA, RT and PC applications run asynchronously on their own.
- The FPGA can run at nanosecond loop times.
- The RT controller can run at microsecond loop times.
- The Windows PC can run at millisecond loop times.
- This presents problems for communicating data.
  - Data can be lost.
  - Data can be read multiple times.

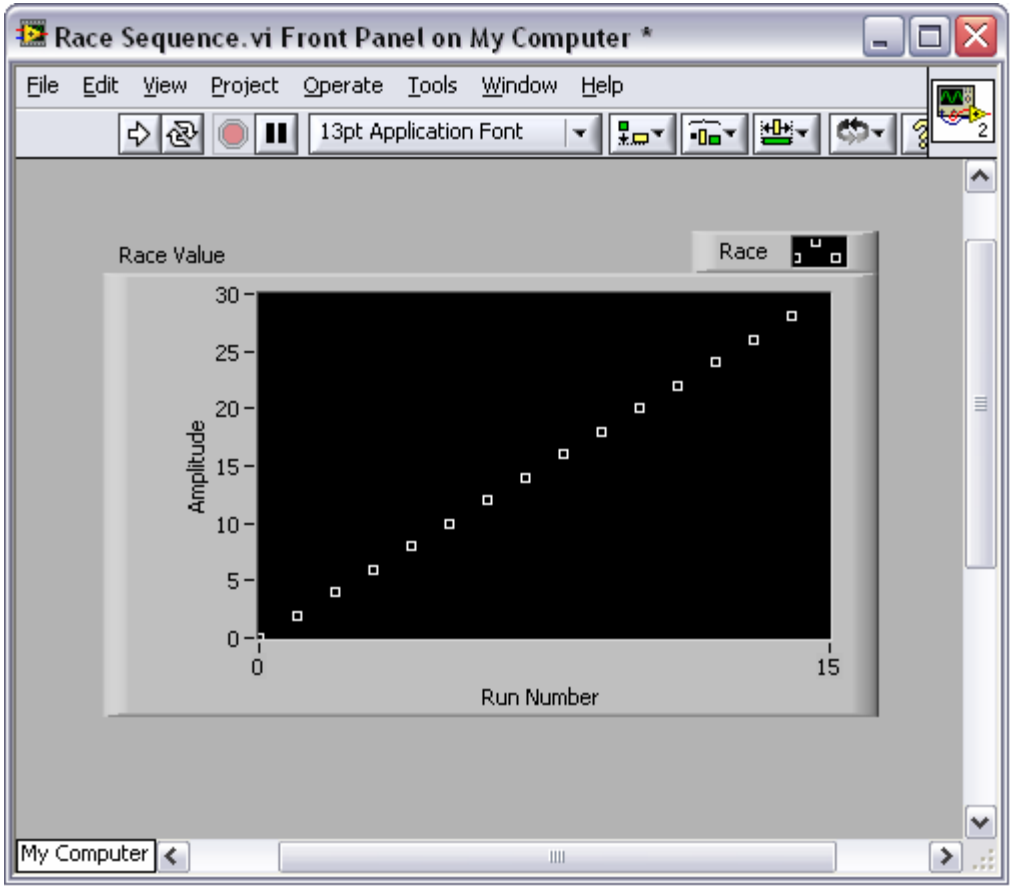
A. Race Conditions

1. What will the value of the Race Sequence VI be after the first run?
2. After the second?
3. After the third?



A. Race Conditions

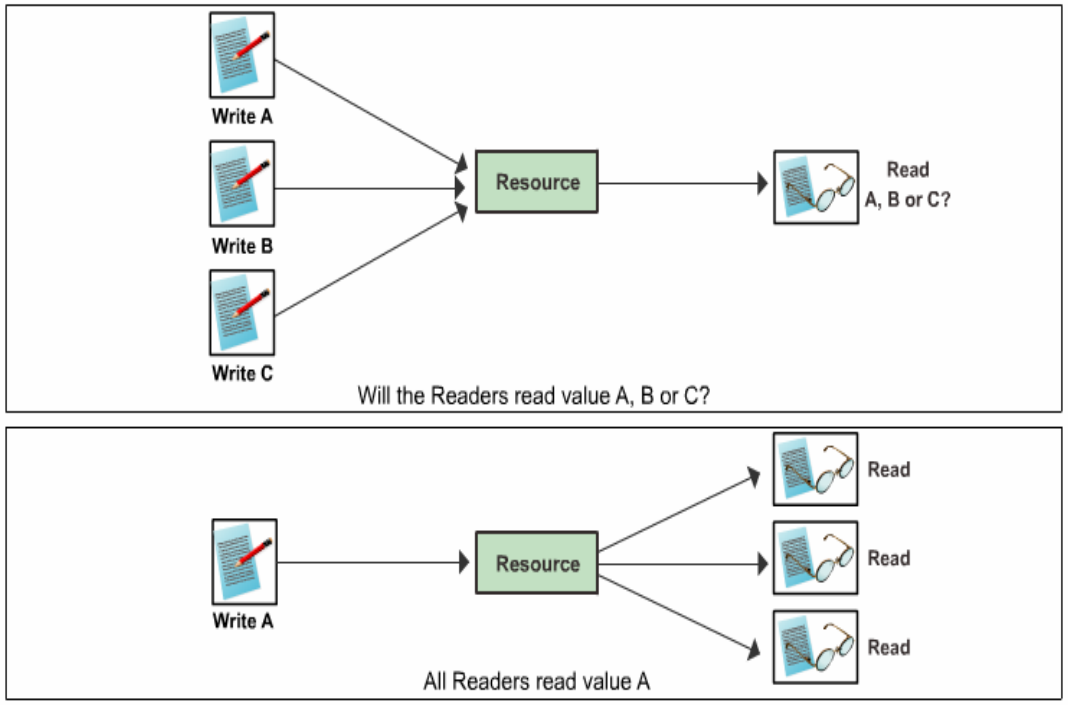
A possible result



A. Race Conditions

The Race Sequence code has several problems:

- It does not control the sequence of operations.
- There are multiple writers to the shared variable.
- The shared variable is not initialized.



A. Race Conditions

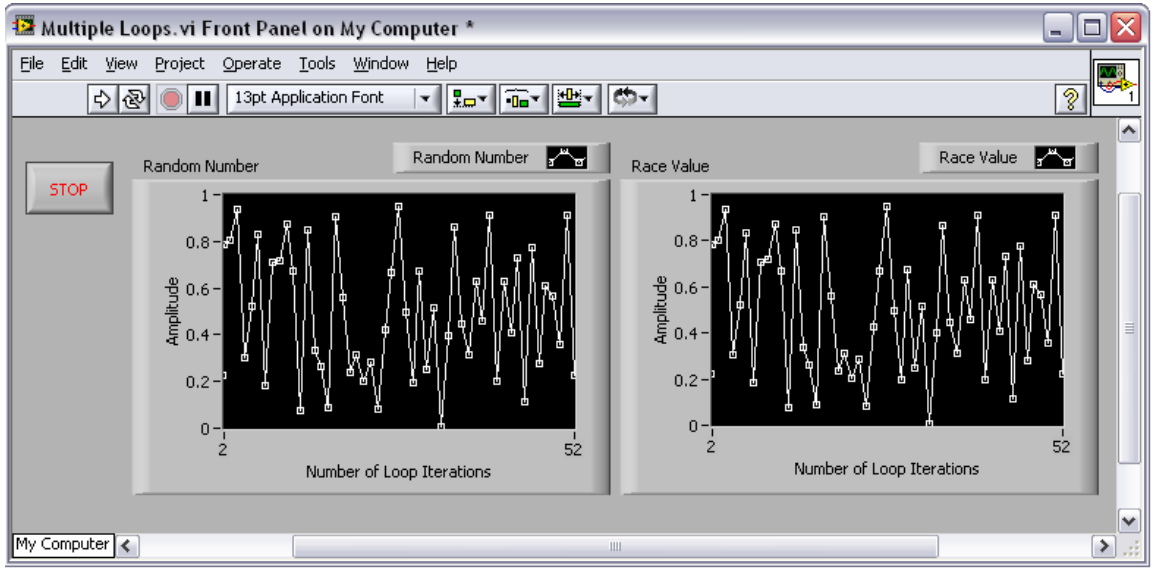
Avoid race conditions by:

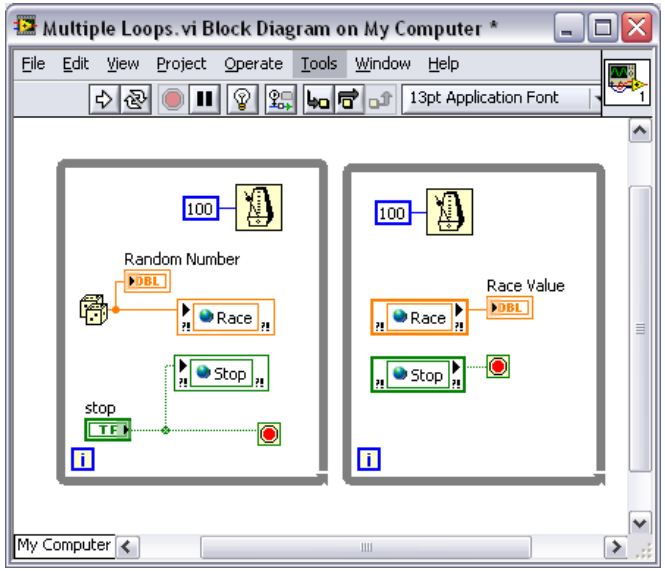
- Controlling shared resources
- Properly sequencing instructions
- Identifying and protecting critical sections within your code
- Reducing use of variables

Initialize shared variables. Otherwise, they retain values from previous executions or default values of associated front panel objects.

A. Race Conditions

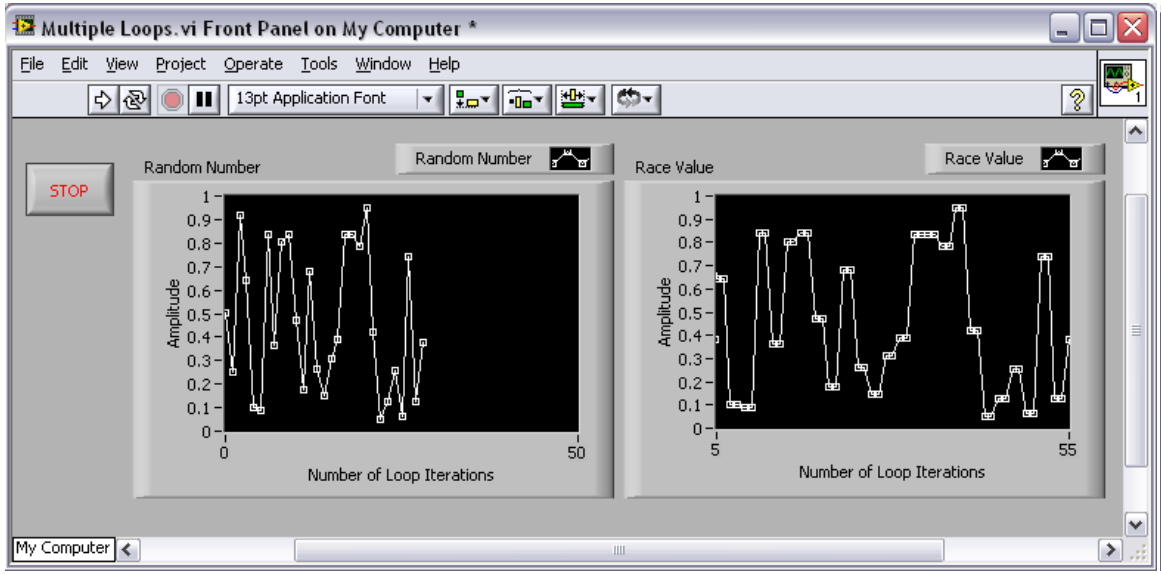
Parallel loops: Write Loop running at the same rate as the Read Loop

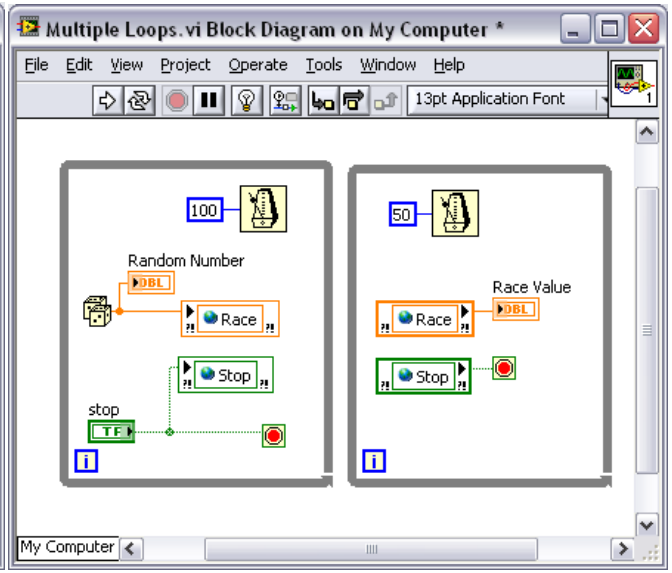




A. Race Conditions

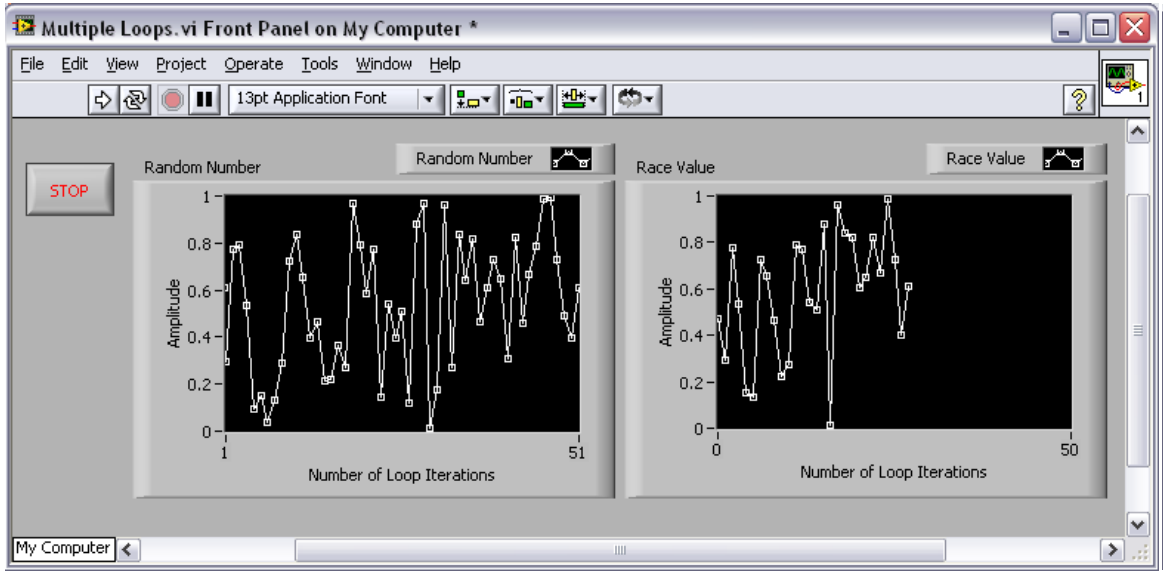
Parallel loops: Write loop running slower than Read Loop (Underflow)

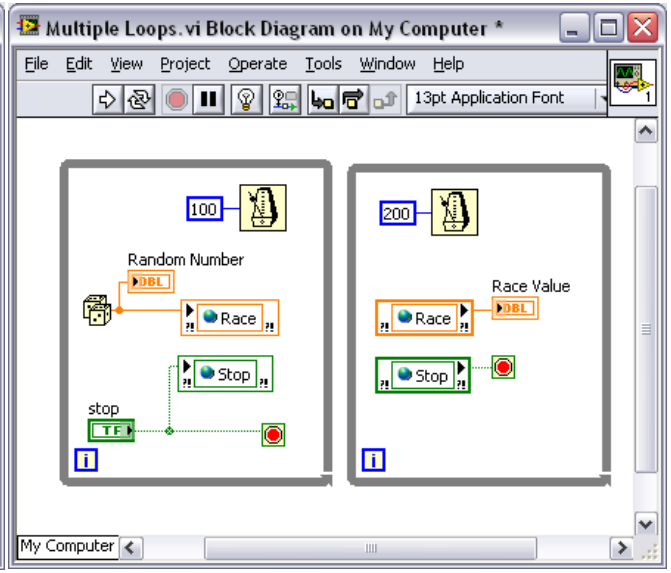




A. Race Conditions

Parallel loops: Write loop running faster than Read Loop (Overflow)





B. Buffering and Synchronization

The level of synchronization needed is application dependent.

- Synchronization is not required in some slower control applications..
- Synchronization is required if you need to acquire data with a known rate and transfer all data without loss.

Buffer – An area of computer memory where multiple data items are stored.

Synchronization – Tasks occur at the same time or in unison using clock, triggers, or events.

B. Buffering and Synchronization

Methods

- Handshaking
- Interrupts
- Direct Memory Access

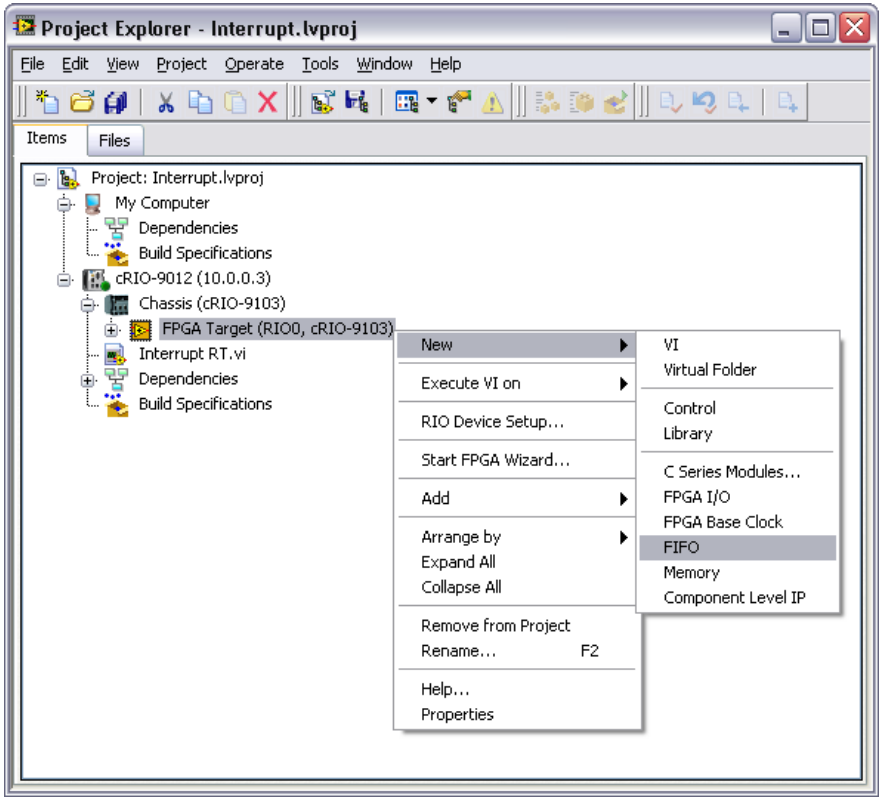
C. FPGA FIFOs

FIFO – First-in first-out buffer. The first data item written to memory is the first item read and removed from memory.

- VI-Scoped – A single FIFO transfers data between multiple loops in the same VI.
- Target-Scoped – A single FIFO transfers data between loops on multiple VIs running on the same target.
- DMA – A single FIFO transfers data to and from RT VIs by directly accessing memory.

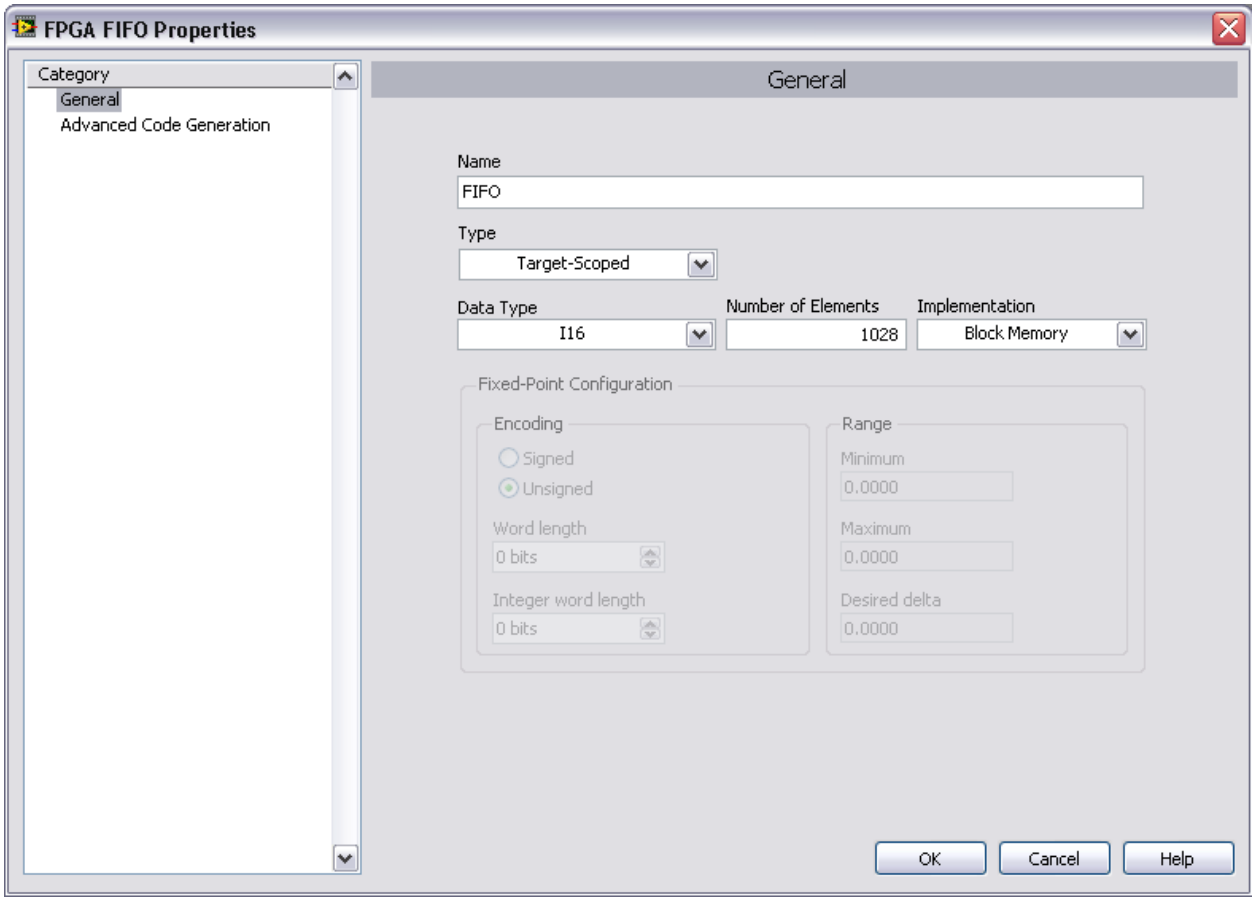
C. FPGA FIFOs

Create Target-Scoped or DMA FIFOs from the Project Explorer window



C. FPGA FIFOs

Configure



C. FPGA FIFOs

Type

- Select Target-Scoped from the Type pull-down menu in the Properties Dialog if you want to use a target-scoped FIFO.
- Select Host to Target-DMA or Target to Host-DMA from the Type pull-down menu if you want to use a DMA FIFO.

C. FPGA FIFOs

Implementation (disabled for DMA FIFOs)

- Flip-Flops – Use gates on the FPGA to provide the fastest performance. Recommended only for very small FIFOs < 100 bytes.
- Look-up Tables – Store data in look-up tables available on the FPGA (2 per slice). Recommended only for small FIFOs < 300 bytes.

Allocate the FIFO in user (block) memory to preserve FPGA space for your VI.

- 80 kB on a 1M gate FPGA
- 192 kB on a 3M gate FPGA

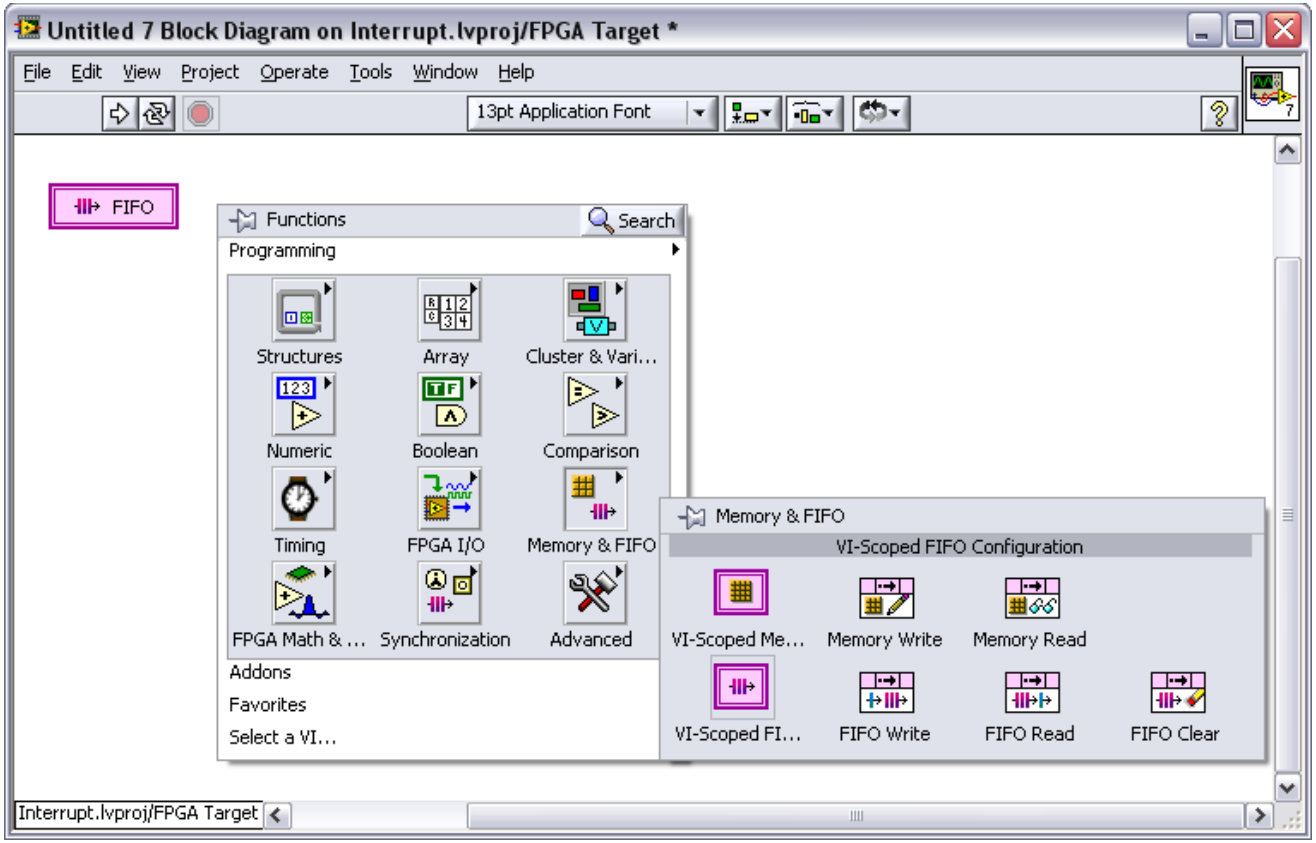
C. FPGA FIFOs

Number of Elements

- Number of elements FIFO can hold
- Maximum depends on the **Implementation** that has been selected and the amount of space available on the FPGA for the implementation

C. FPGA FIFOs

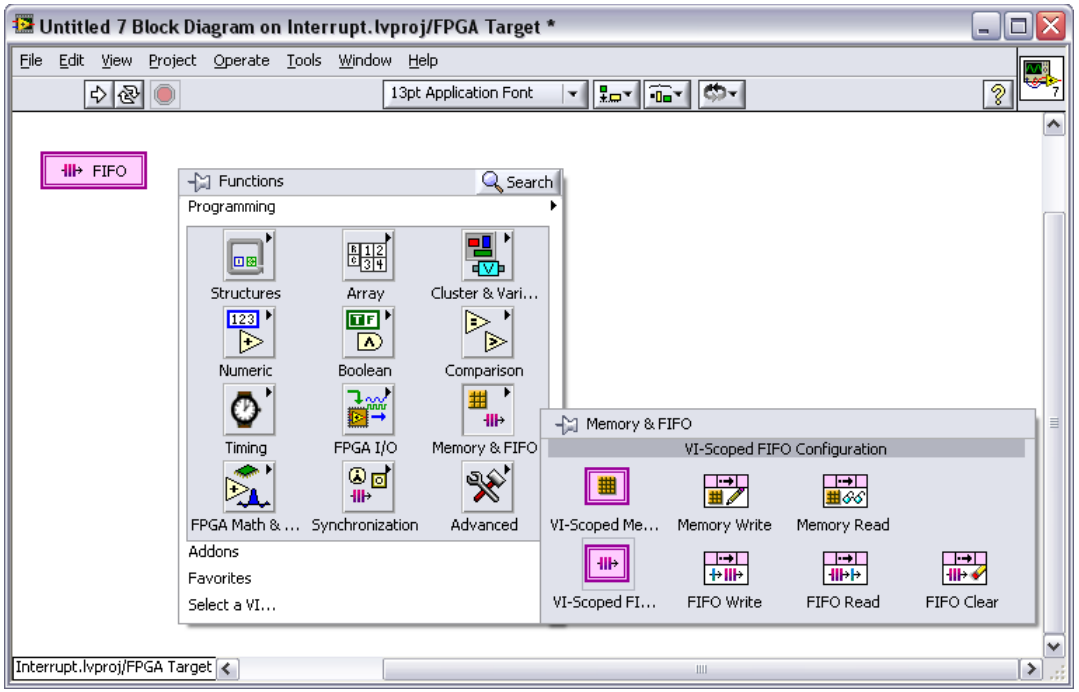
Create a VI-Scoped FIFO from the Functions palette



C. FPGA FIFOs

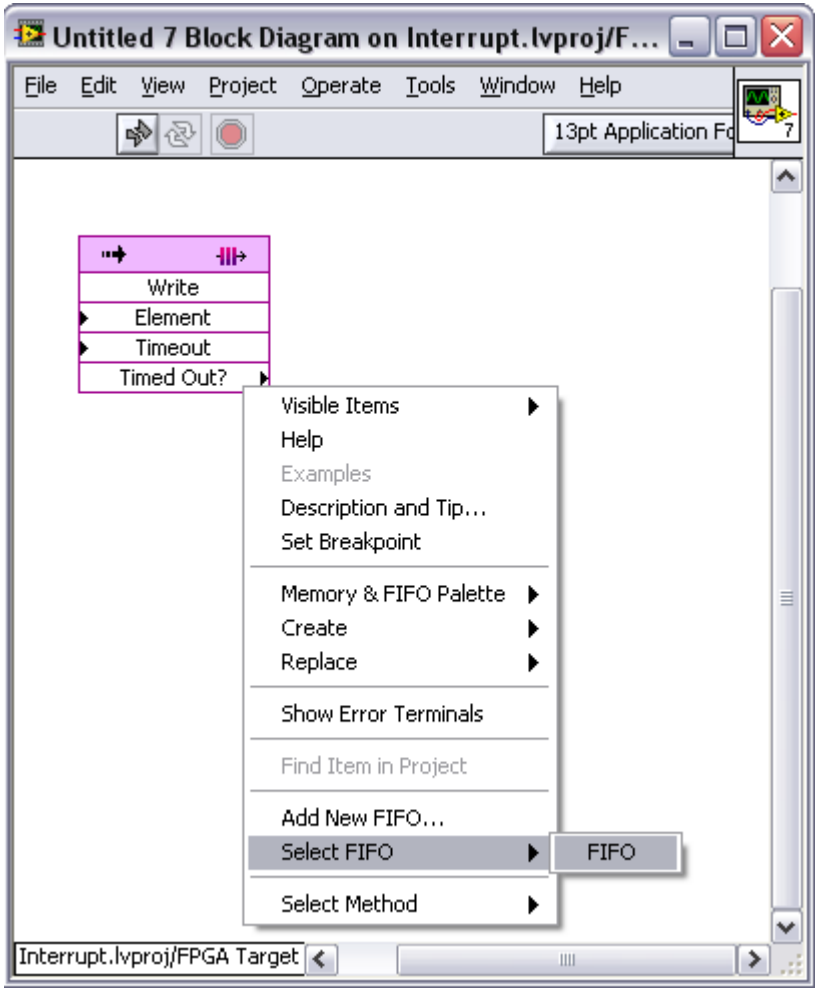
Put data into the FIFO with the FIFO Write function

Retrieve data with the FIFO Read function



C. FPGA FIFOs

- Place an FPGA FIFO Write or Read from the Functions palette
- Right-click the object and choose Select FIFO
  - Associates node with a defined FIFO



C. FPGA FIFOs

FPGA FIFO Write Terminals

- Element – Inputs the data element to store
- Timeout – Sets number of ticks the function waits for space if the FIFO is full. 0 (default) = no wait, –1 prevents timeout.
- Timed Out?– Is True if FIFO is not available. Does not overwrite. Does not add new element.
- Can be lossy

C. FPGA FIFOs

Resetting a FIFO

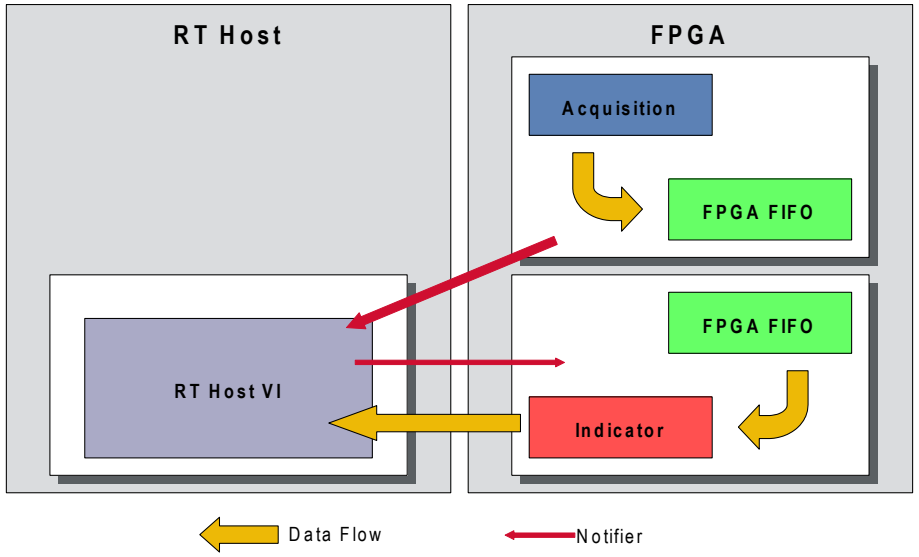
- FIFOs reset when the VI stops and restarts using the development machine
- To reset programmatically:
  - Use the Invoke Method function

    Or
  - Use the Close FPGA VI function configured to Close and Reset

C. FPGA FIFOs

VI scoped FIFO with Programmatic Front Panel Communication:

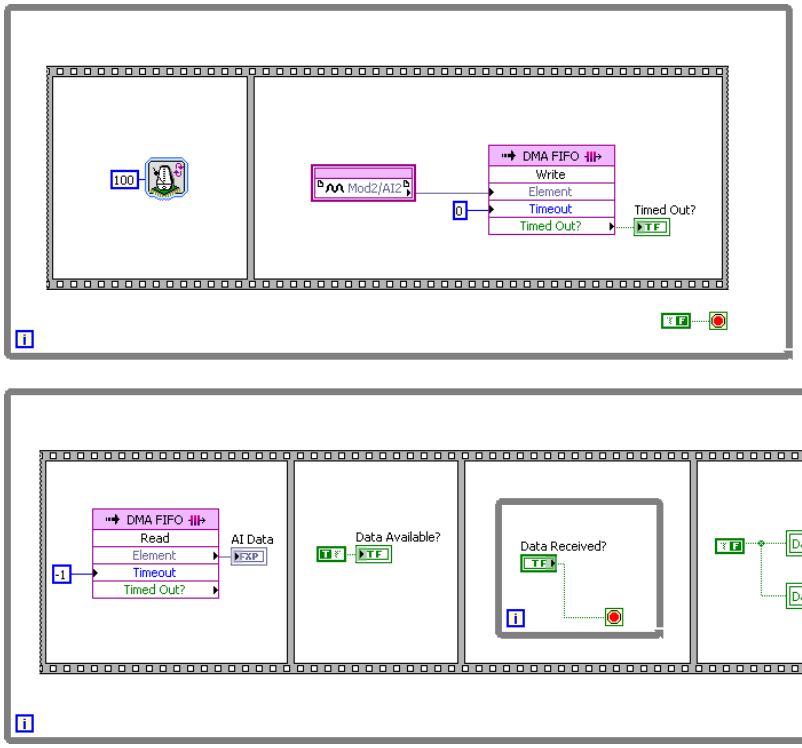
1. I/O writes to FIFO and notifies host
2. Transfer loop reads FIFO and passes data to host using Front Panel Communication
3. Host acknowledges transfer and transfer loop sends another data value



C. FPGA FIFOs

Separate I/O and data transfer to speed I/O

- Top loop acquires data.
- Transfer loop reads FIFO and passes data to host using Front Panel Communication.



C. FPGA FIFOs

Clear Method

- Use the Clear Method to empty a target- or VI-scoped FIFO
  - Or call Reset VI method from host.

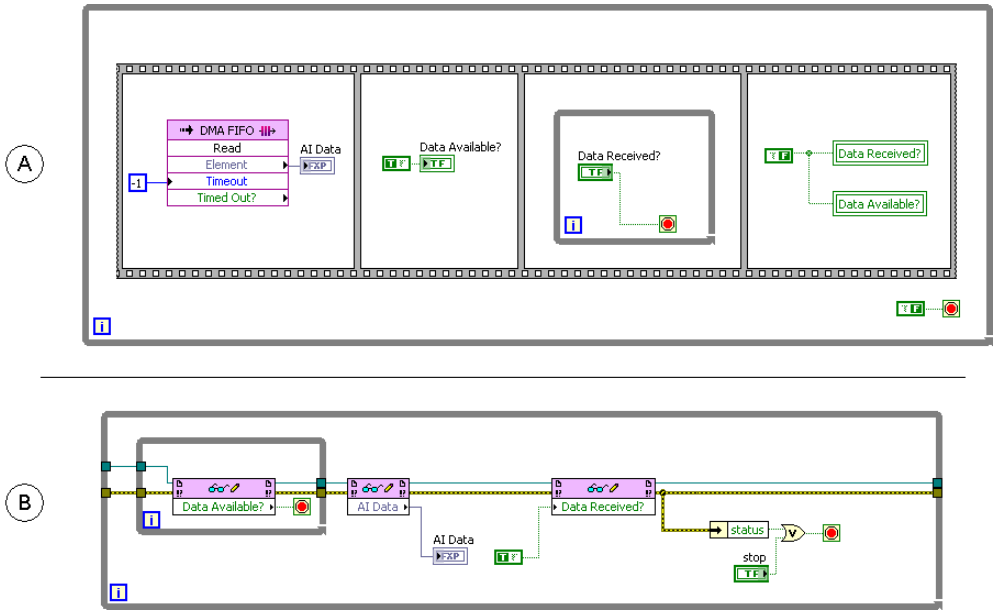
D. Handshaking

Handshaking – Checking that a receiving device is ready to receive or a transmitting device is ready to transmit

- Host uses Boolean controls and indicators (data available and data read flags) on the target to coordinate operations
- Requires polling

D. Handshaking

1. FPGA acquires data and writes True to Data Available?
2. When Data Available? is True, host reads data.
3. After data is read, host writes True to Data Received.
4. FPGA loop iterates when Data Received is True.

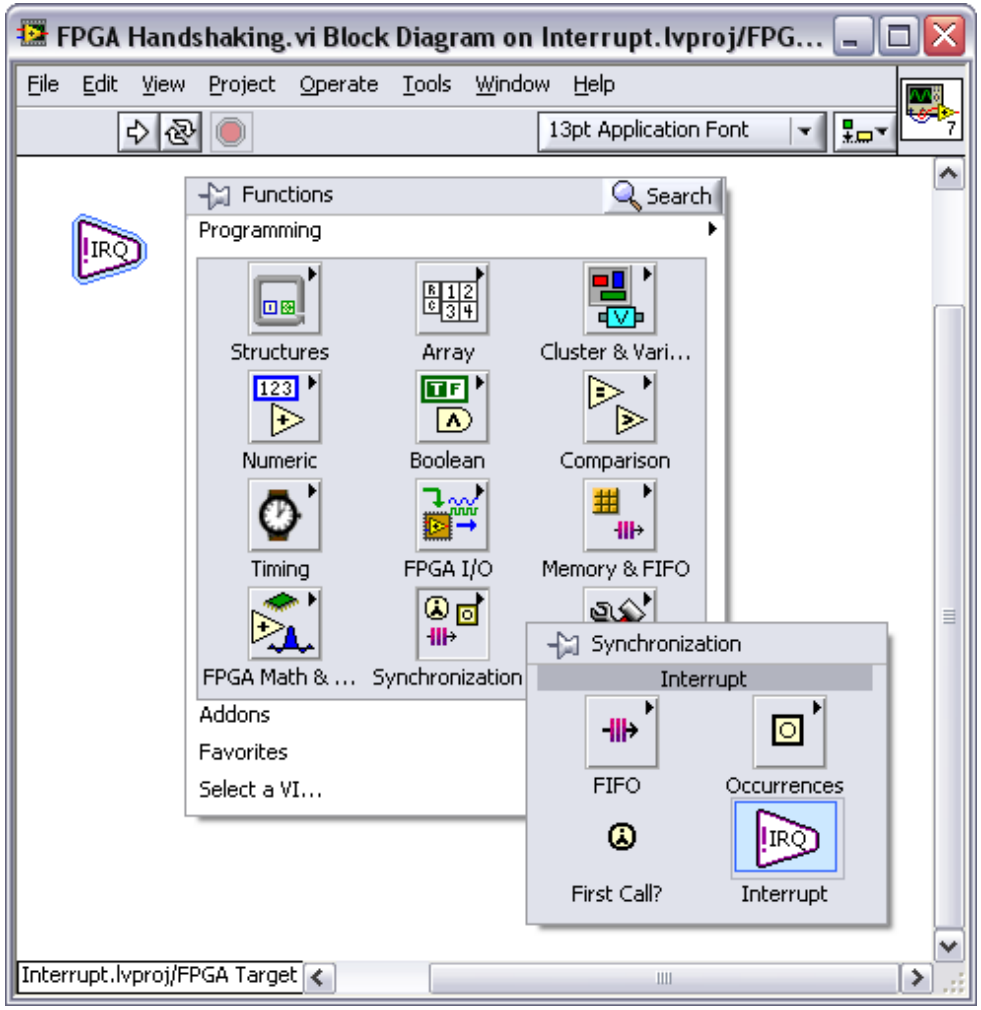


Note: Use small arrays on FPGAs

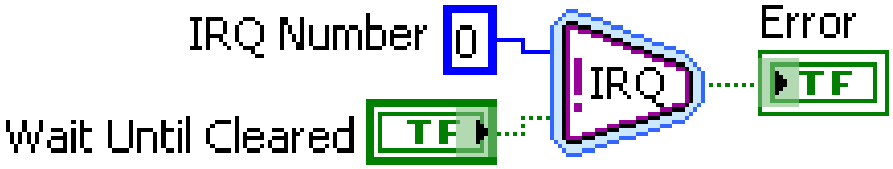
E. Interrupts

- Communicate over a physical hardware line.
  - FPGA Interrupts send a trigger to the host.
  - Eliminate polling.
  - Allow host to perform other operations while waiting for the Interrupt signal.

E. Interrupts

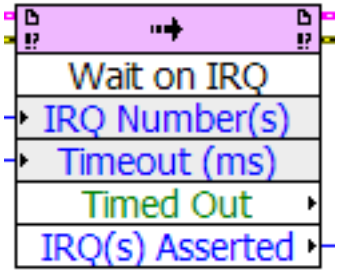


E. Interrupts



- IRQ Number – specifies which logical interrupt (0 – 31) to assert. The default is 0.
- Wait Until Cleared – specifies if this VI waits until the host VI acknowledges the logical interrupt. The default is FALSE.

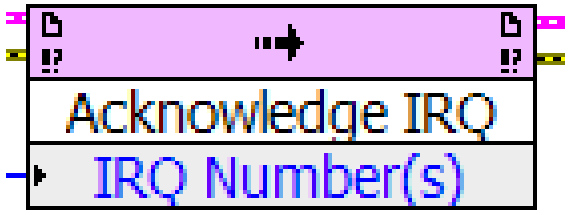
E. Interrupts



RT Host and Interrupts

- RT host must acknowledge interrupts.
- Use Wait on IRQ and Acknowledge IRQ methods.
  - IRQ Number(s) – specifies the logical interrupt for which the function waits.
  - Timeout (ms) – specifies the number of milliseconds the VI waits before timing out. –1 = infinite timeout.
  - Timed Out – returns TRUE if this method has timed out.
  - IRQ(s) Asserted – returns the asserted interrupts. Empty or –1 array indicates that no interrupts were received.

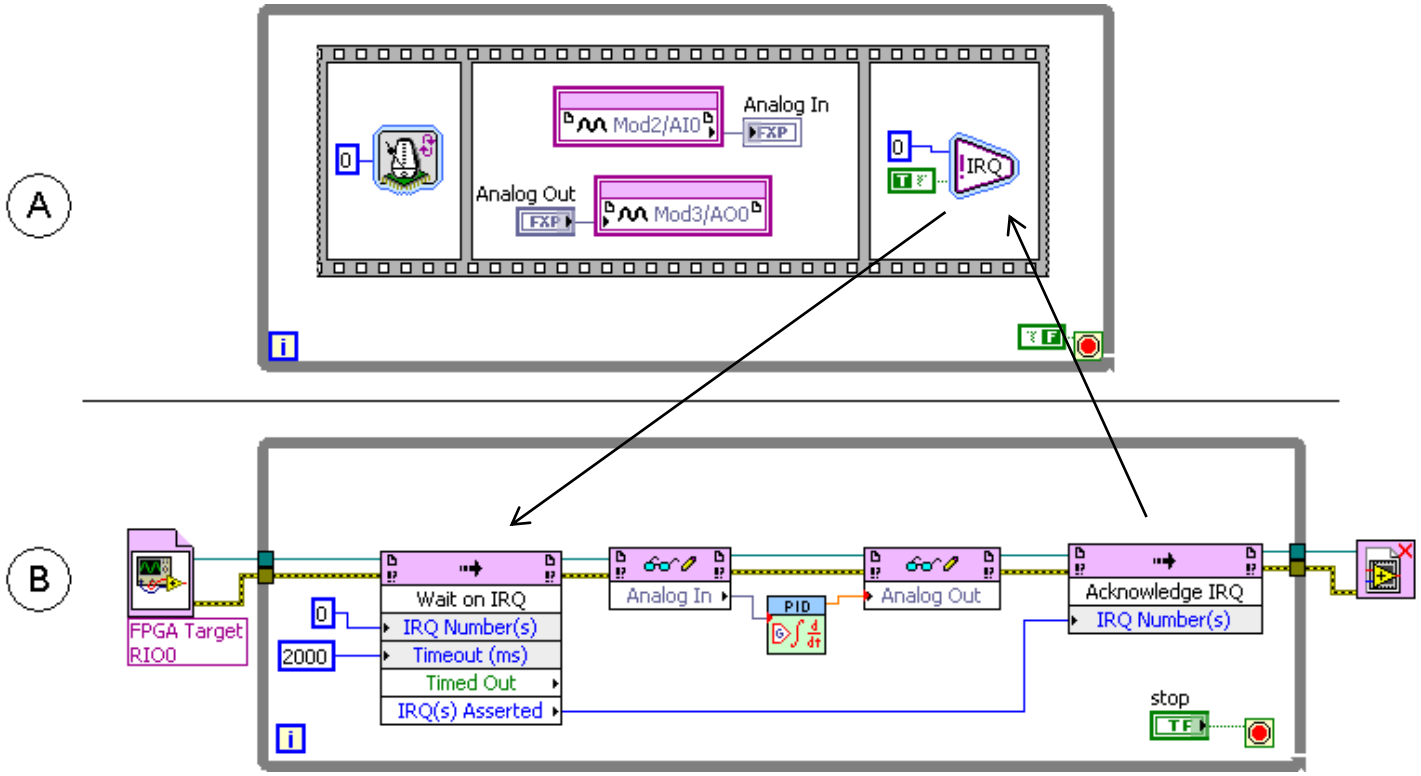
E. Interrupts



- Acknowledge IRQ method
  - Acknowledges and resets any interrupts that have occurred to their default values. Wire after the Wait on IRQ method.
  - IRQ Number(s) specifies the logical interrupt or array of logical interrupts for which the function waits.

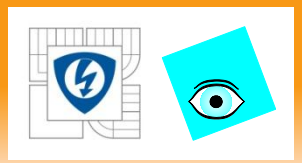
E. Interrupts

Interrupt implementation.



F. Direct Memory Access

- Transfers data from FPGA directly to memory on the RT controller through bus mastering.
- Streams large amounts of data.
- Provides better performance than using local FIFO and reading indicators.
- Host processes data while FPGA transfers data to host memory.
- Without DMA, processor must read data.
- Consists of two parts – FIFO part and host part.
- DMA engine transfers data along PCI bus to host memory.



Demonstration: Target-to-Host DMA FIFO

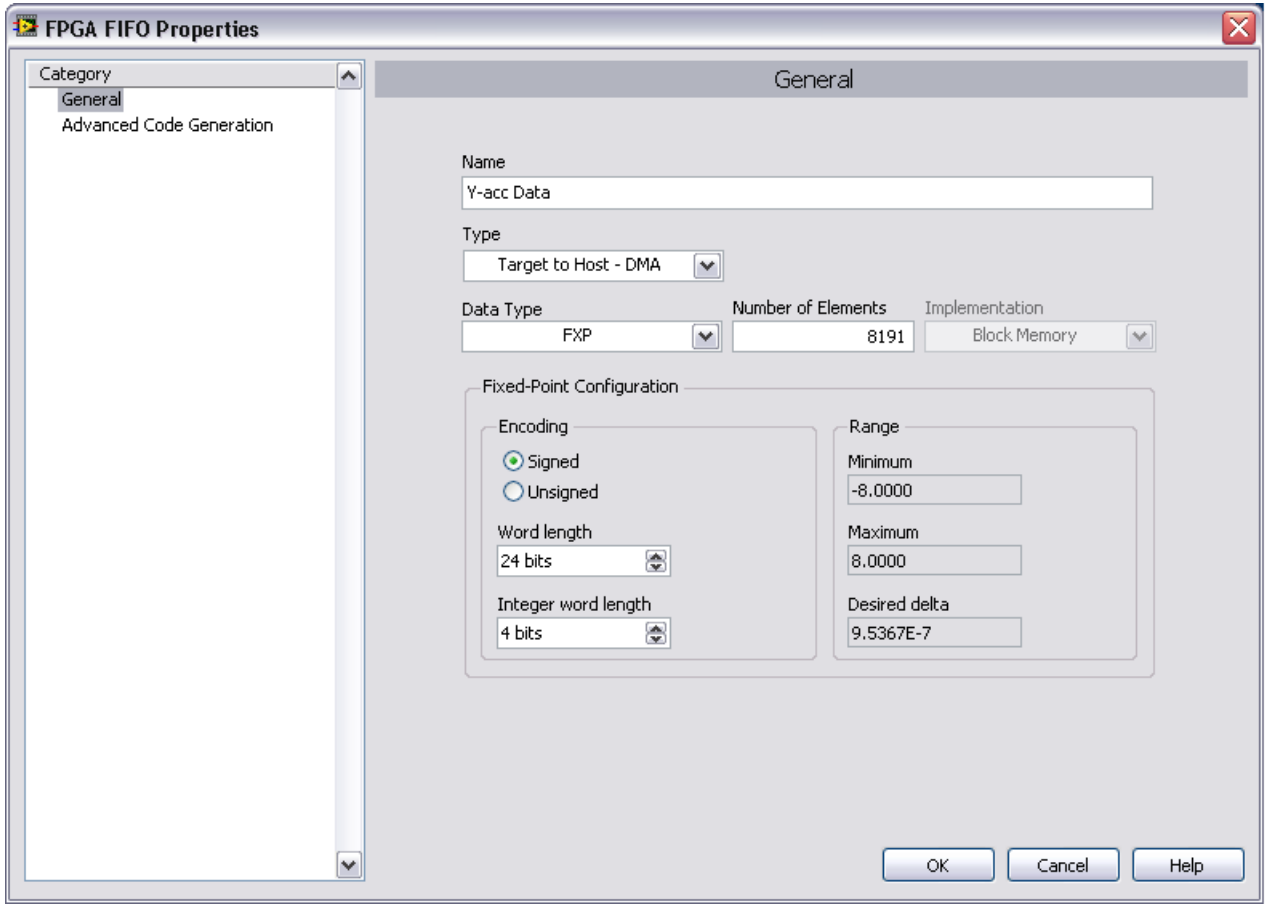
Show how data is transferred from the FPGA to the host.

Demonstration: Host-to-Target DMA FIFO

Show how data is transferred from the host to the FPGA.

- <Exercises>/CompactRIO Fundamentals/Demonstrations

F. Direct Memory Access



F. Direct Memory Access

- The Number of Elements determines the size of the FPGA part of the FIFO.
  - Save resources by choosing the smallest size that is reasonable for your application.
  - Compare the rate at which data points are put into the FPGA FIFO to the rate at which the host application will be able to read points out.

F. Direct Memory Access

Specify the size, or depth of the RT host part of the FIFO

- Configure Method
- Host computer DMA FIFO size = 2 X FPGA DMA FIFO size if unconfigured
- Consider longest delay and configure sizes large enough so they do not fill

F. Direct Memory Access

If the FIFO number of elements is too small and FIFO fills:

- Acquire slower
- Increase number of elements to read on host
- Increase FIFO/host buffer sizes

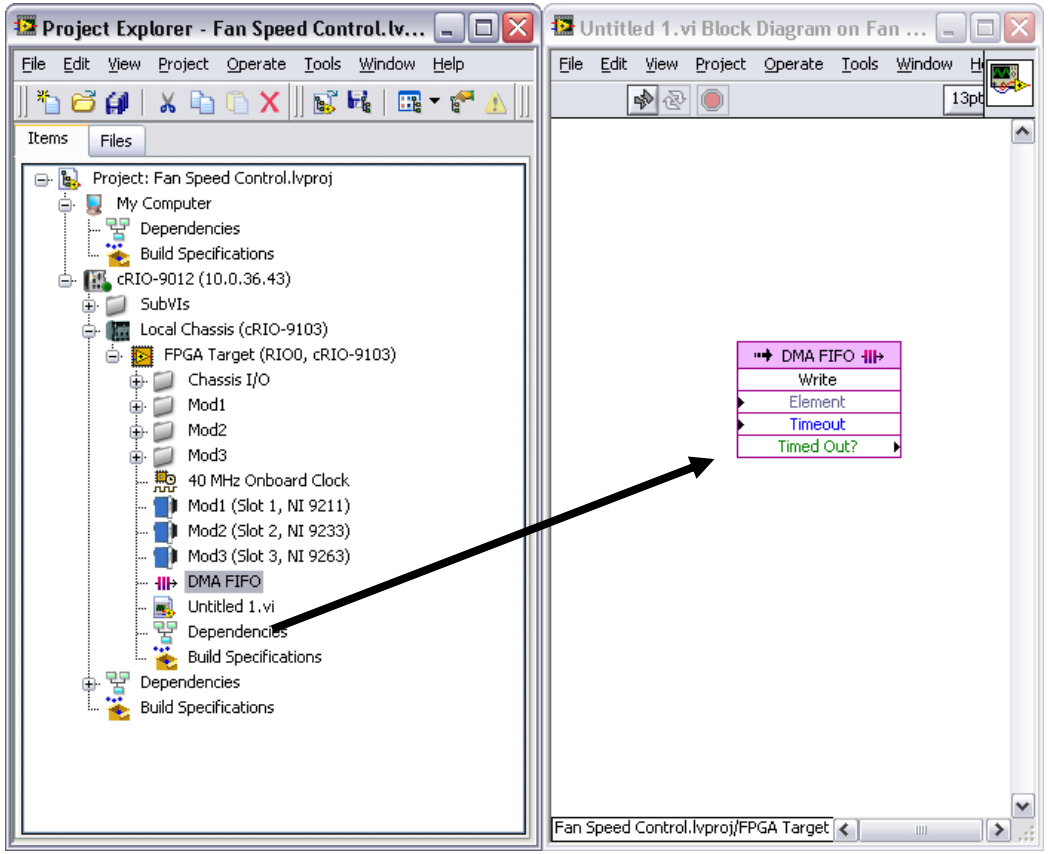
F. Direct Memory Access

FIFO number of elements is too large:

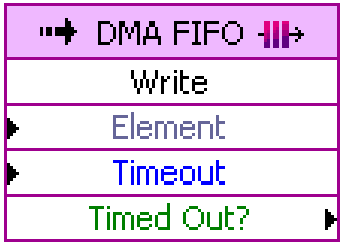
- Other functions on the FPGA cannot use the memory
- Other functions on the host cannot use the memory.

F. Direct Memory Access

Drag FIFO from Project Explorer window.



F. Direct Memory Access



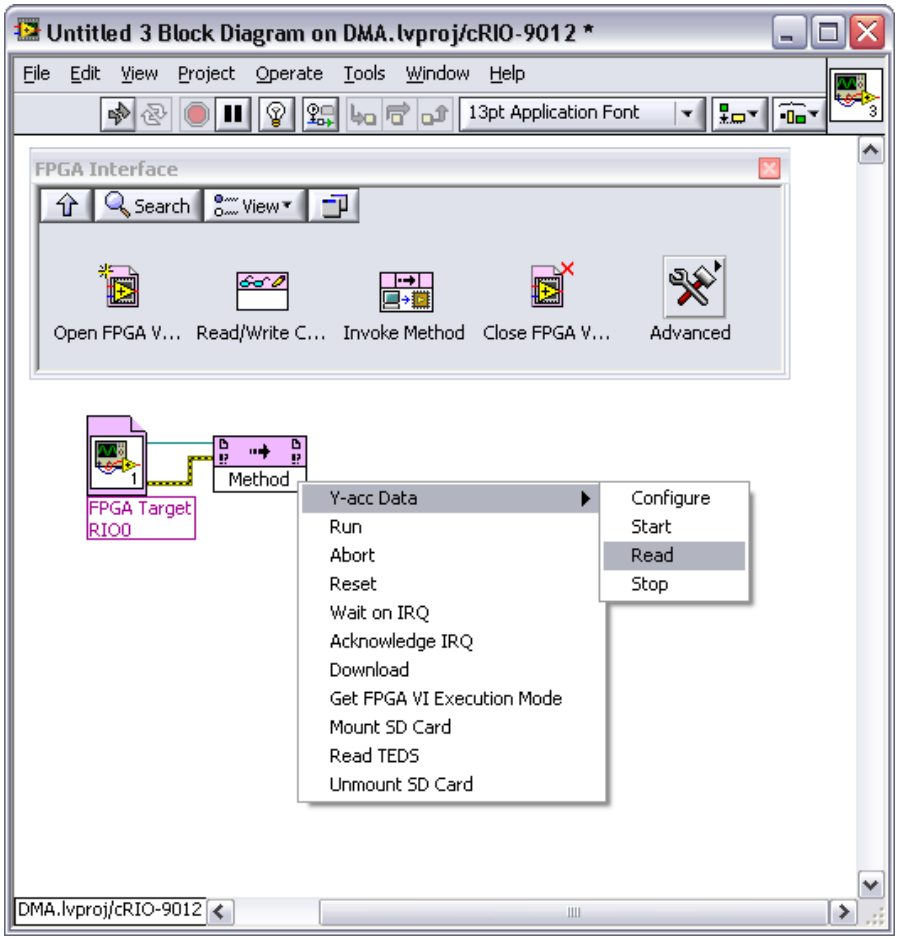
FIFO Write Function

- Element – inputs the data element to be stored.
- Timeout – inputs the number of clock ticks the function waits for available space in the FIFO if the FIFO is full. The default is 0, or no wait. A value of –1 prevents the function from timing out.
- Timed Out? – returns TRUE if space in the FIFO is not available before the function completes execution.
  - If Timed Out? is TRUE, the function does not add Element to the FIFO.

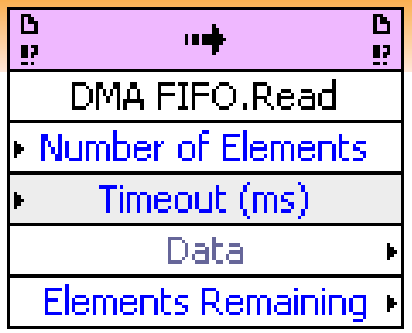
F. Direct Memory Access

Read a DMA FIFO in the host:

1. Open a reference to the FPGA.
2. Add an Invoke Method function.
3. Choose the Data Read Method.



F. Direct Memory Access



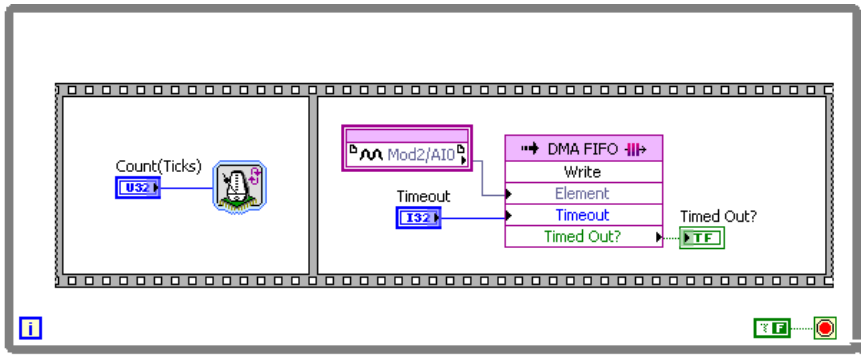
- Read a DMA FIFO in the Host:
  - Number of Elements – determines the number of elements to read in each iteration.
    - Function completes when elements are read or when timeout is reached.
    - Same overhead for reading a large number as a small number of elements. So read more elements if host is too slow.
  - Timeout – length of time to wait before timeout. Default is 5,000 ms.
    –1 = indefinite wait.
  - Data – returns data read
  - Elements Remaining – indicates elements remaining in the host part of the DMA FIFO

F. Direct Memory Access

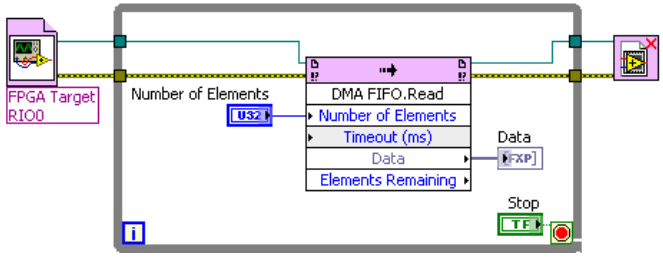
Implementing a DMA FIFO with blocking:

- User specifies number of elements
- DMA Engine attempts to transfer within the Timeout

A

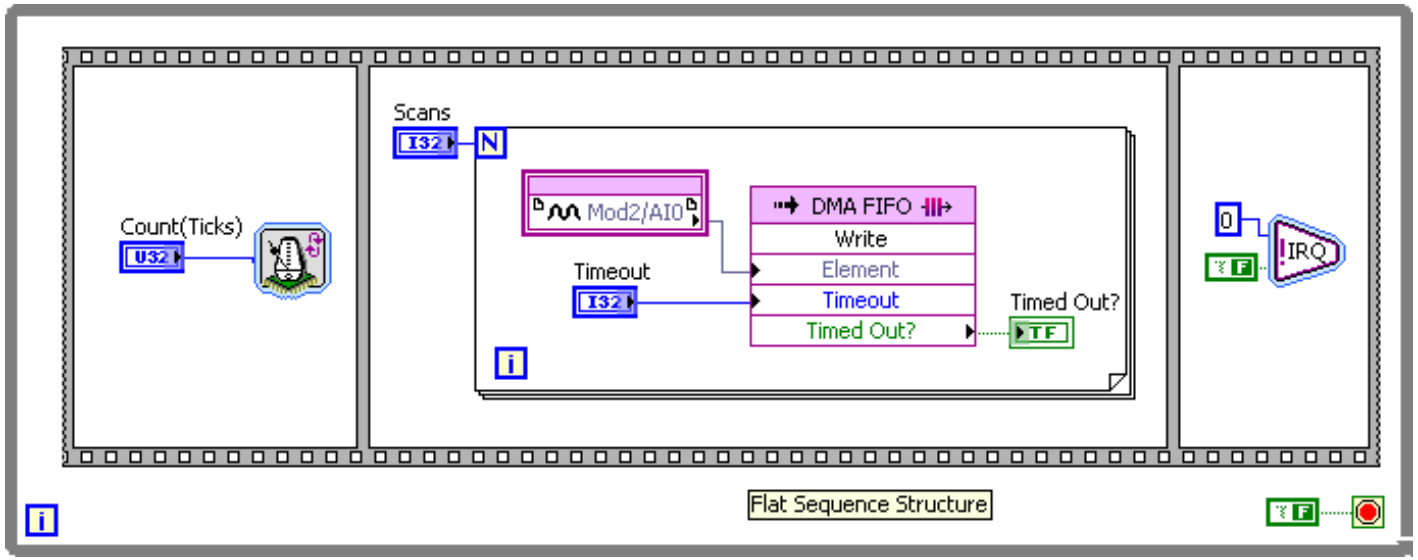


B



F. Direct Memory Access

Implementing a DMA FIFO with an Interrupt

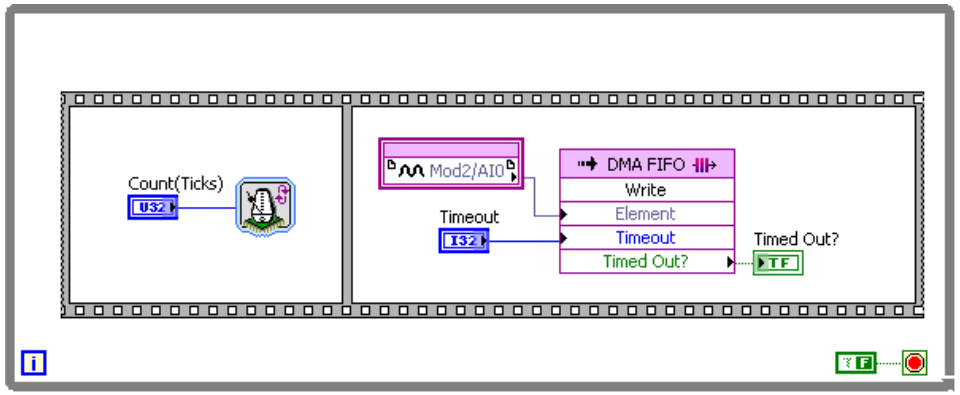


F. Direct Memory Access

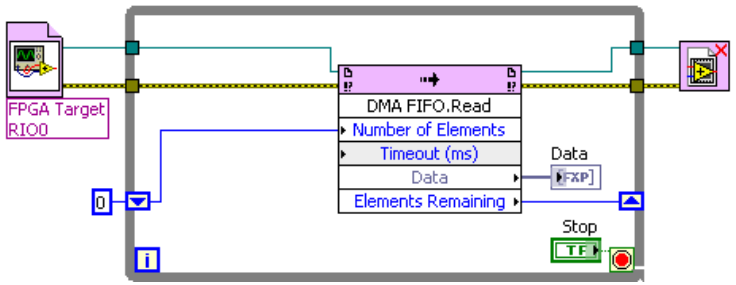
Implementing a DMA FIFO with Polling:

- 0 to number elements
- Returns elements to read

A

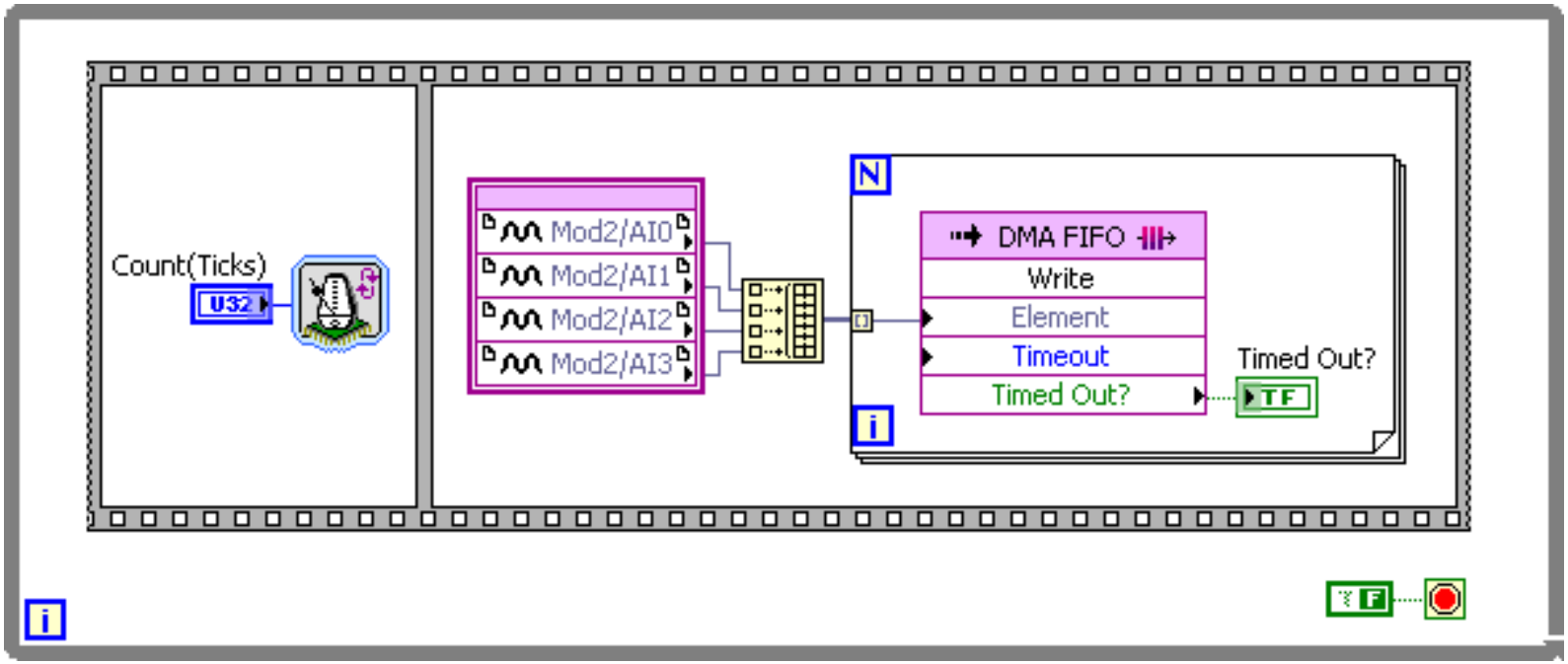


B



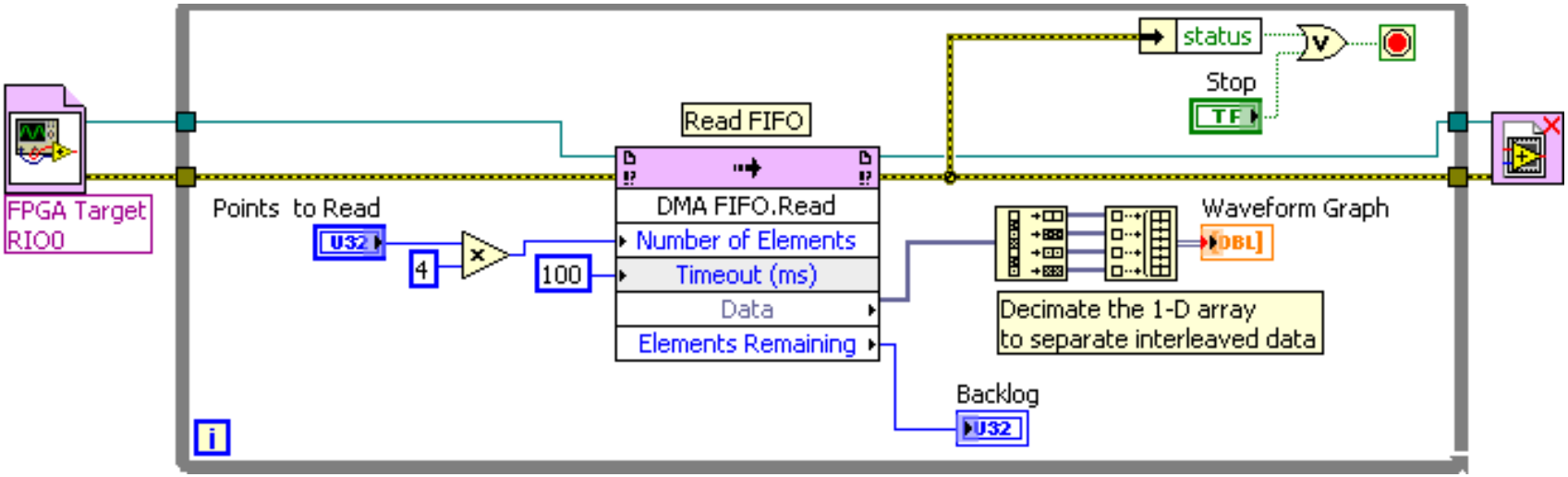
F. Direct Memory Access

Writing multiple channels of data to a DMA FIFO by interleaving



F. Direct Memory Access

Reading multiple channels from a DMA FIFO in the host

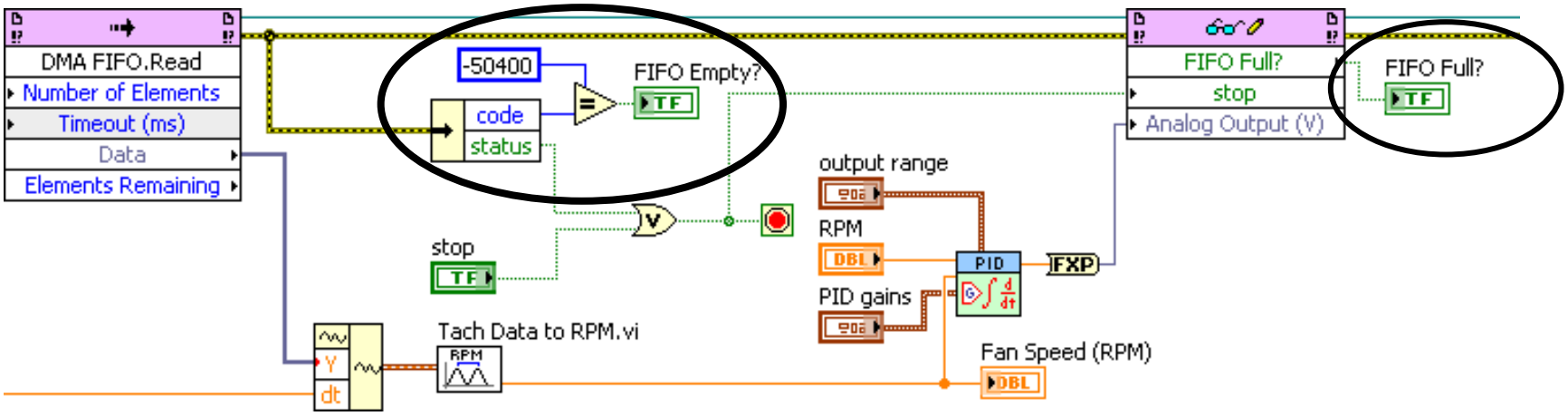


F. Direct Memory Access

Lossless Application

Overflow – DMA Full?

Underflow – Check for –50400



F. Direct Memory Access

Overflow – too many data points for buffer, may lose data

- Acquire data slower
- Increase number of elements to read on the host
- Increase the rate that the host reads data
- Increase buffer sizes on FPGA and host

Demonstration:
Target-to-Host DMA FIFO - Overflow

Show how an overflow of the DMA FIFO can occur if the RT host VI code is too complex

F. Direct Memory Access

Underflow – not enough data to read, FIFO read times out

Increase timeout

Read slower

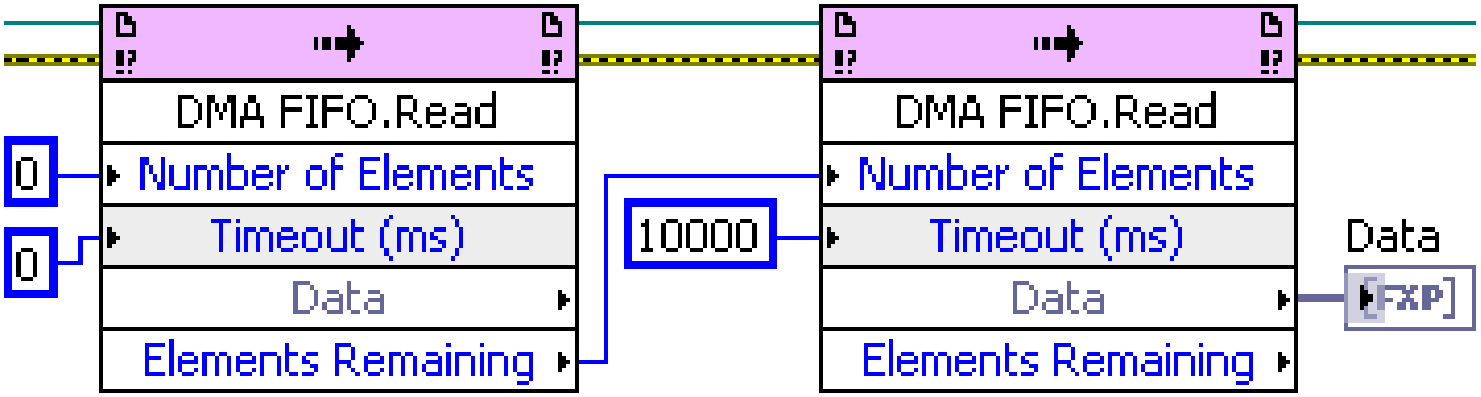
Read smaller sets of elements

Demonstration: Target-to-Host DMA FIFO – Underflow

Show how an underflow occurs when the DMA FIFO tries to read data before it is available.

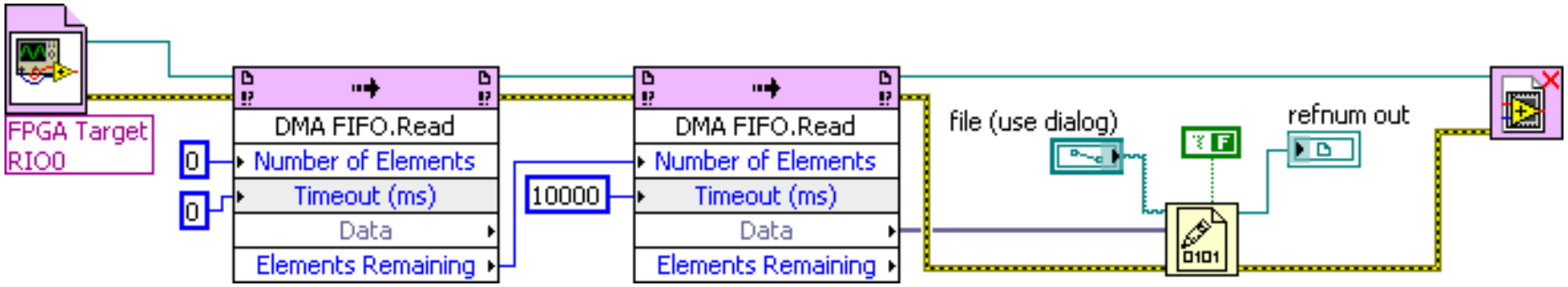
F. Direct Memory Access

Recover data and flush FIFO to find out how much data is remaining



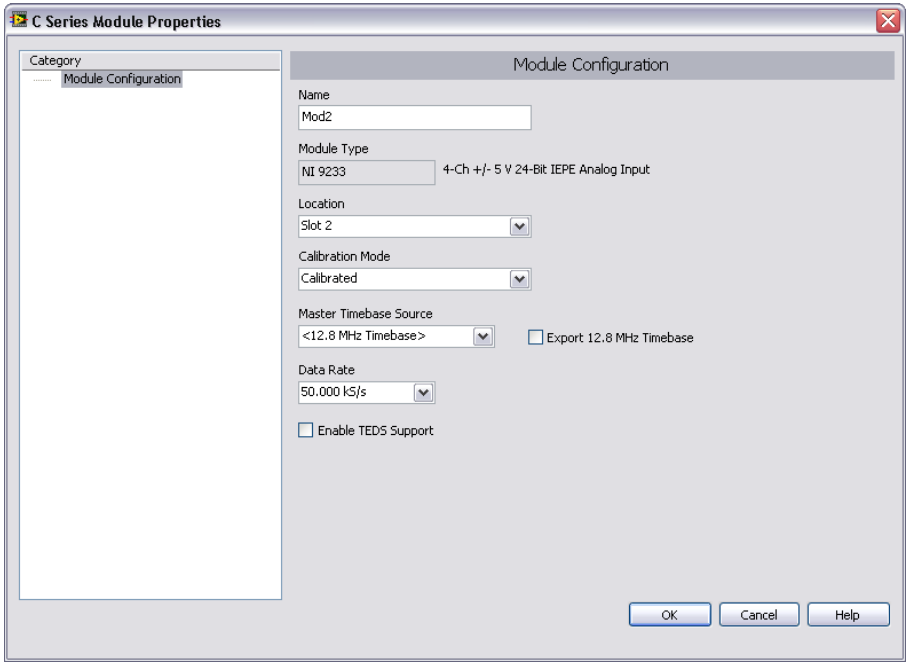
F. Direct Memory Access

DMA Flush Buffer subVI



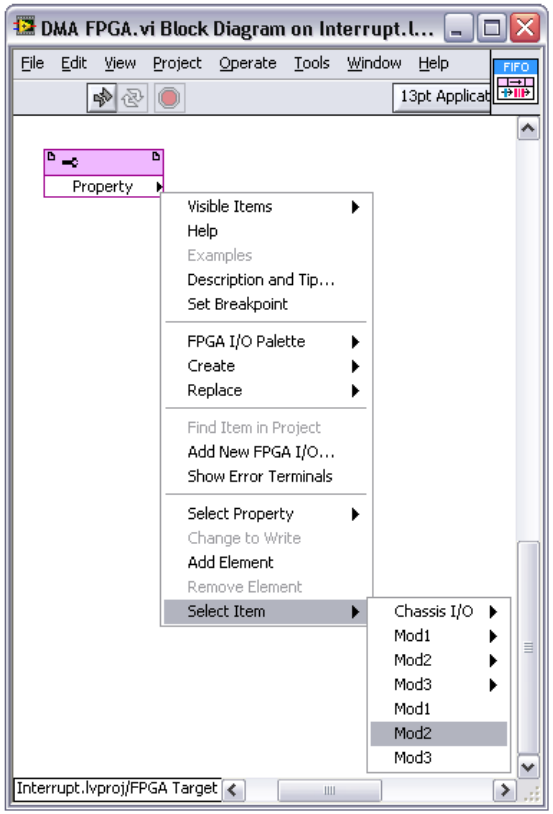
G. Hardware-Timed Data Acquisition

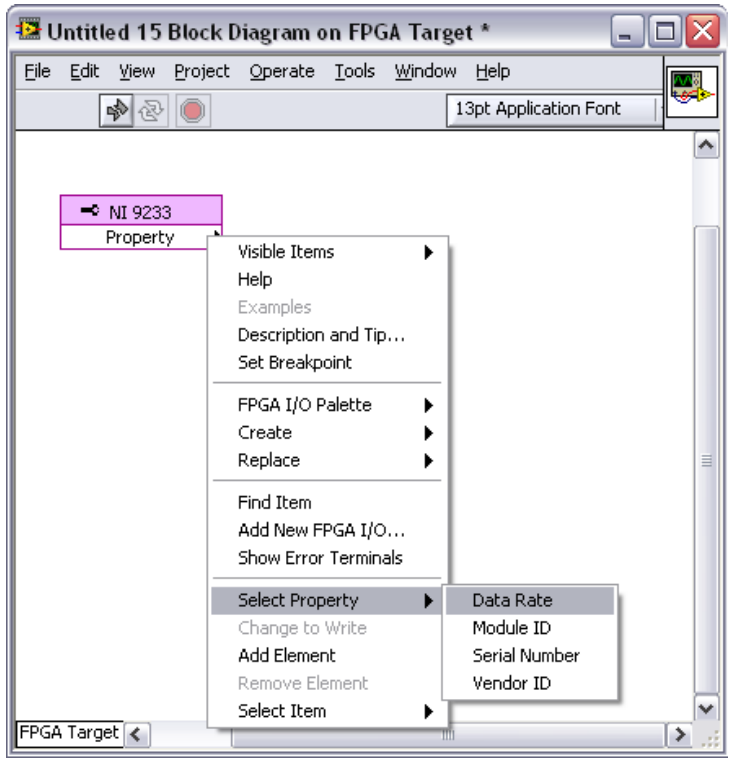
- NI 9233 does hardware timed data acquisition
- Search LabVIEW Help – NI 9233 or right-click in Project Explorer window
- 4 AI Channels sampled simultaneously and two digital channels (start/stop)
- Set Data Rate statically or programmatically



G. Hardware-Timed Data Acquisition

Create an NI 9233 Property Node and Data Rate property

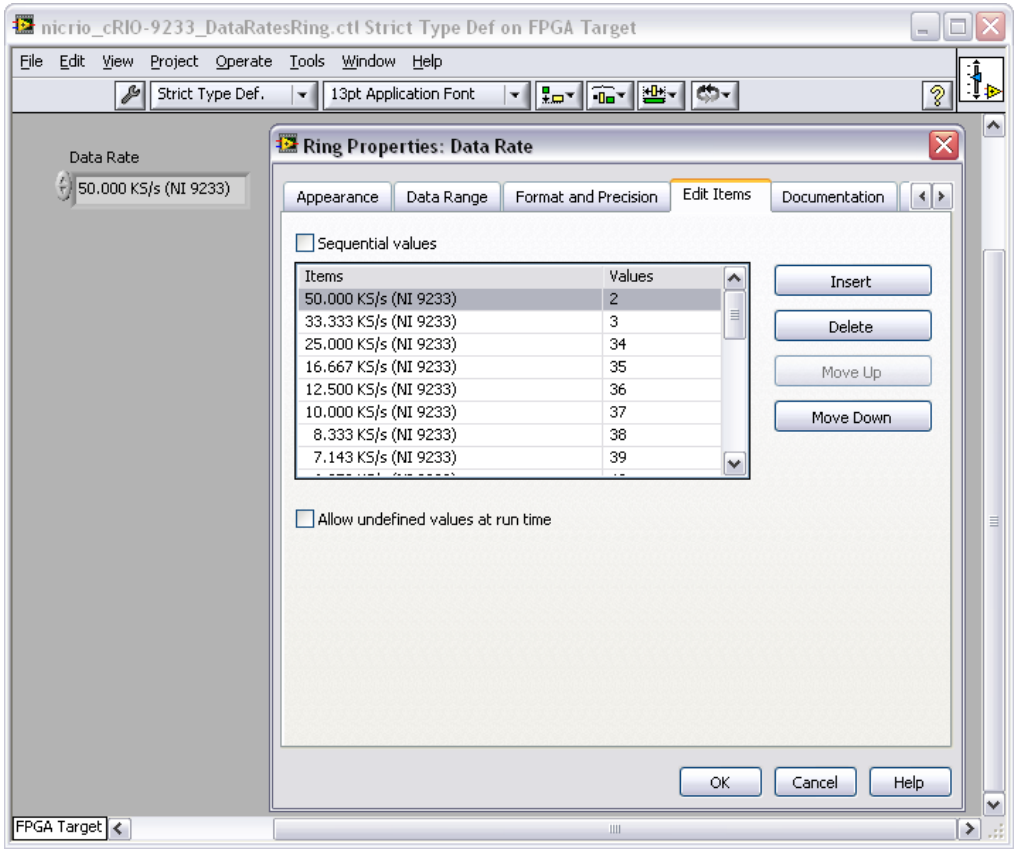




G. Hardware-Timed Data Acquisition

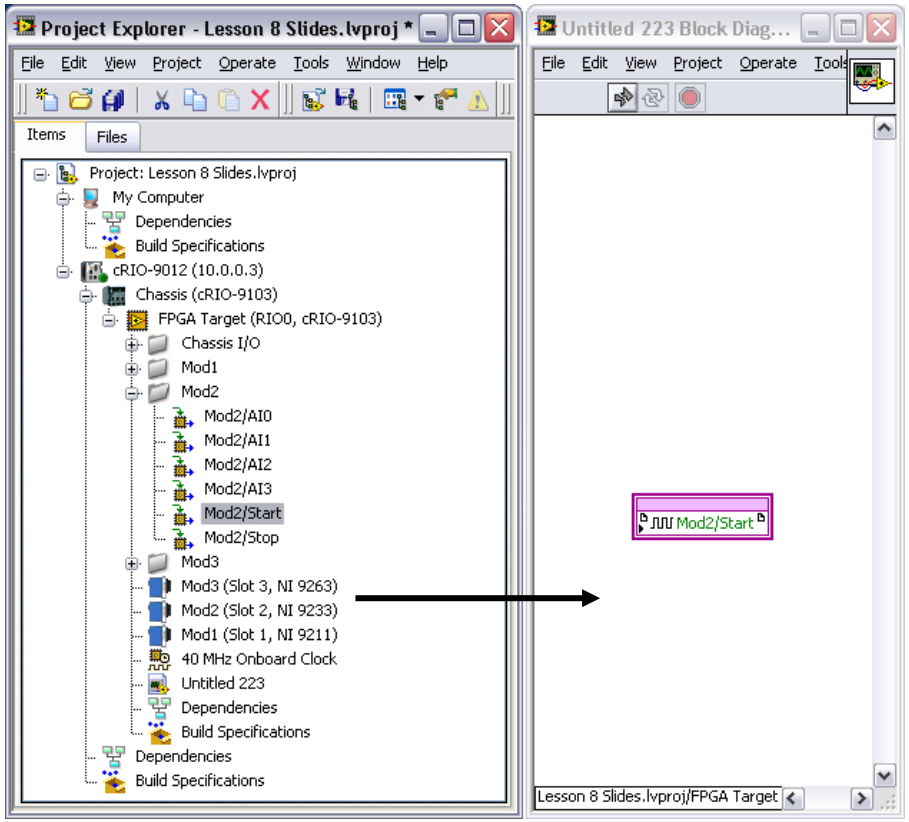
The NI 9233 Data Rate Control

- Custom Ring
- Strict Type



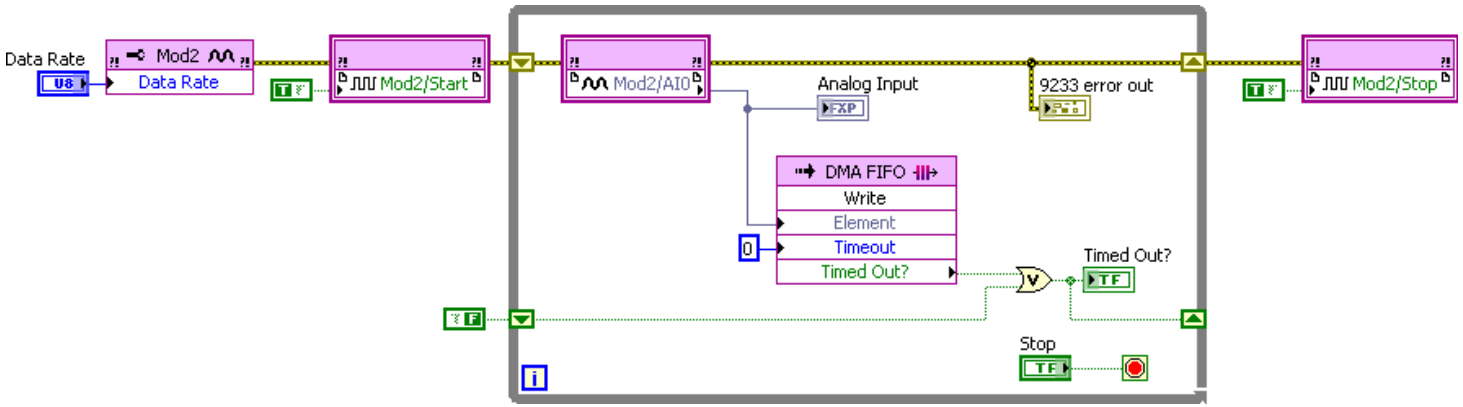
G. Hardware-Timed Data Acquisition

Drag the Start and Stop items from the Project Explorer window



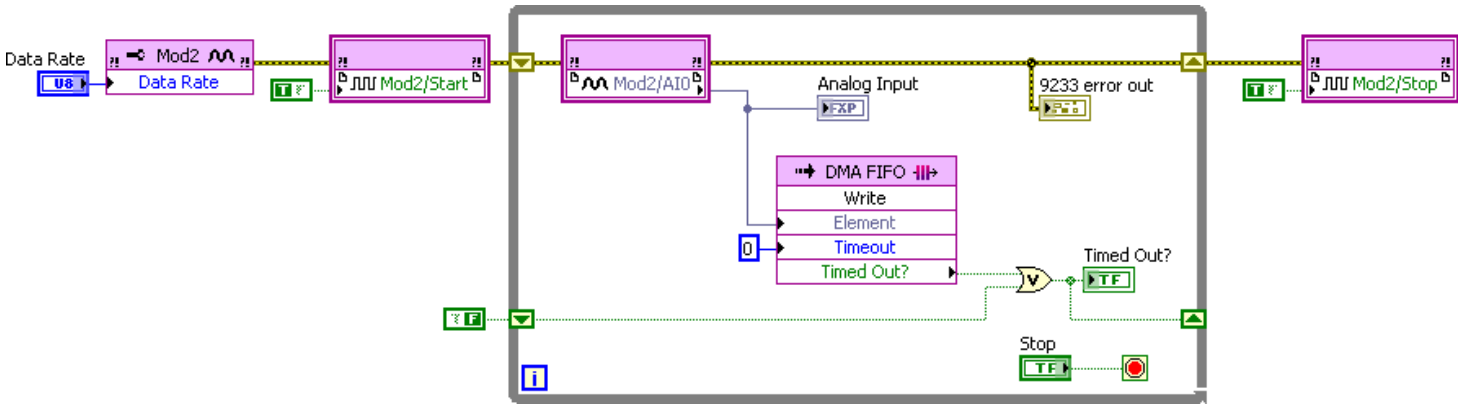
G. Hardware-Timed Data Acquisition

- If you read multiple channels place all in the same node to ensure synchronous reads.
  - The I/O Node does not return until new data has been acquired.
  - If the NI 9233 did not start, or stops while the node is waiting, the node returns a timeout error.
  - You cannot perform other operations such as accessing TEDS info or properties while the NI 9233 is in acquisition mode.



G. Hardware-Timed Data Acquisition

- NI 9233 is internally timed. Do not put timing functions in the loop.
  - If the loop time is slower than the NI 9233 data rate, the I/O Node returns an overrun warning and continues to read data.
  - The warning indicates one or more points were missed.
  - When reading from multiple modules in the same loop and the non-NI 9233 is slower, either reduce the NI 9233 rate or use two loops.



Optimization Techniques

Benchmarking FPGA VIs

Basic Optimizations

Architectural Optimizations

Advanced Optimizations

DODATOK

FPGA OPTIMALIZÁCIA

A. Optimization Techniques

FPGA VIs are limited primarily in two areas:

– Speed
  - Execution rate too slow for specifications

– Size
  - Requires too much space on the FPGA
  - Uses so much RAM that it will not compile

To increase speed and reduce size, optimize FPGA code

A. Optimization Techniques

- Some techniques sacrifice speed for size and vice versa
- Can use multiple techniques in one VI

FPGA Optimization Technique	Speed	Size
Limit Font Panel Objects		X
Use Small Data Types		X
Avoid Large VIs	X	X
Use Non-reentrant subVIs		X
Use Reentrant subVIs	X	
Use Parallel Operations	X	
Use Pipelining	X	
Use Single-Cycle Timed Loops	X	X

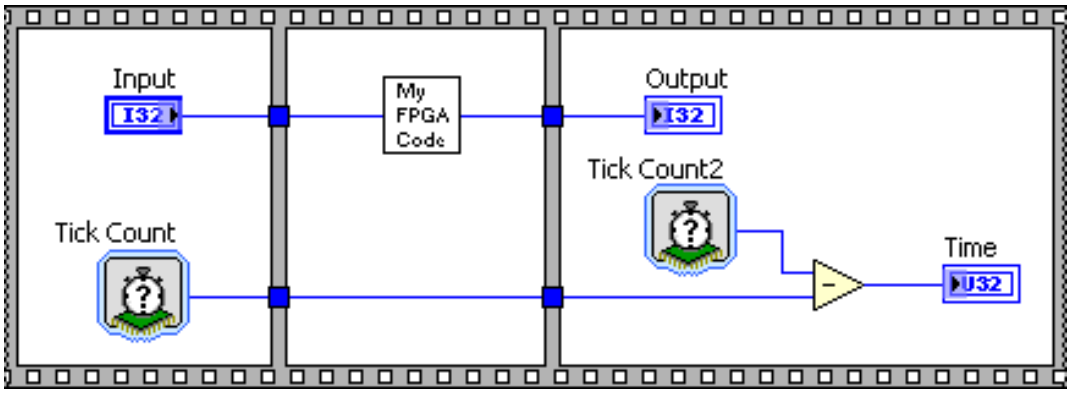
B. Benchmarking FPGA VIs

- To benchmark FPGA VI Speed use Tick Count VIs
  - Requires additional code for testing
    - Typically removed in final application
- To benchmark FPGA VI Size analyze the Compile Report
  - Compile Size unknown until entire compile is complete

B. Benchmarking FPGA VIs

Benchmark the Execution Speed of a VI

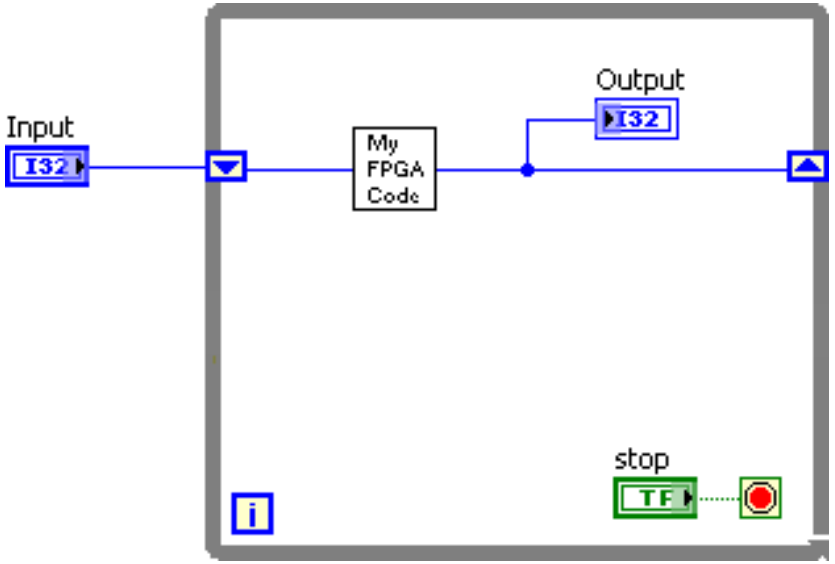
- Use Tick Count VI to determine execution speed
  - Get initial time
  - Execute code
  - Get final time
  - Calculate the difference
    - Code can be removed later
  - Timestamp measurements done in parallel
    - Does not affect execution speed



B. Benchmarking FPGA VIs

Benchmark the Loop Rate of a VI

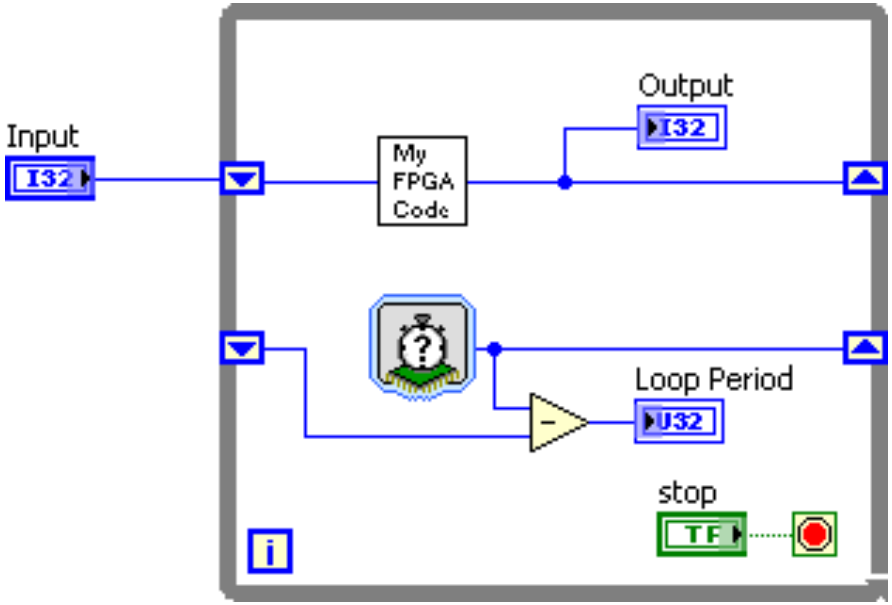
- Loop Rate limited by maximum speed of code in loop
- Maximum loop rate limited by code execution time plus 2 ticks
  - 1 Tick = 1 Clock cycle
  - Clock cycle depends on compile rate (Default 40 MHz)



B. Benchmarking FPGA VIs

Benchmark the Loop Rate of a VI

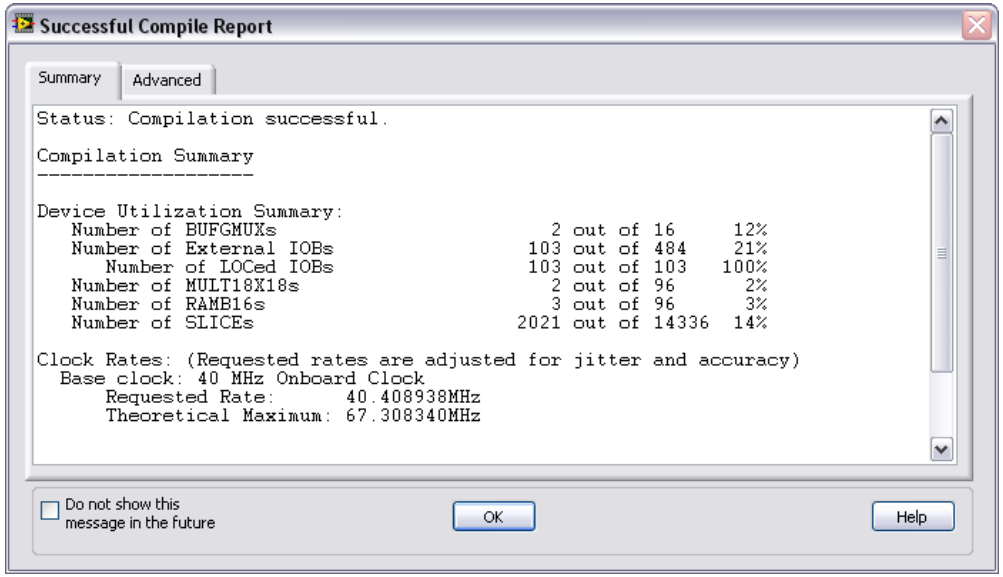
- Timestamp each iteration
- Calculate the difference
  - Code can be removed later
- Timestamp measurements done in parallel
  - Does not affect execution speed



B. Benchmarking FPGA VIs

Benchmark the Size of a VI

- Compilation Summary



C. Basic Optimizations

These types of optimizations are relatively easy to implement

- Require no major changes in code architecture
- Should be basic programming practice for all FPGA VIs
- Basic Optimizations primarily affect FPGA size

C. Basic Optimizations

Types of Basic Optimizations:

- Limit Front Panel Objects
- Bitpack Boolean Logic
- Use Small Data Types
- Avoid Large Functions
- Optimize Comparisons
- Reentrant vs. Non-Reentrant SubVIs

C. Basic Optimizations

Limit Front Panel Objects

- Each Front Panel Object on the Top-Level VI must have logic to interact with the host VI.
- Each read and write from the host to the FPGA is broken down into 32-bit packets to transfer across the bus.

C. Basic Optimizations

Limit Front Panel Objects

- Avoid using Arrays on the Front Panel
  - Compile fails if more bytes in array than are available in RAM

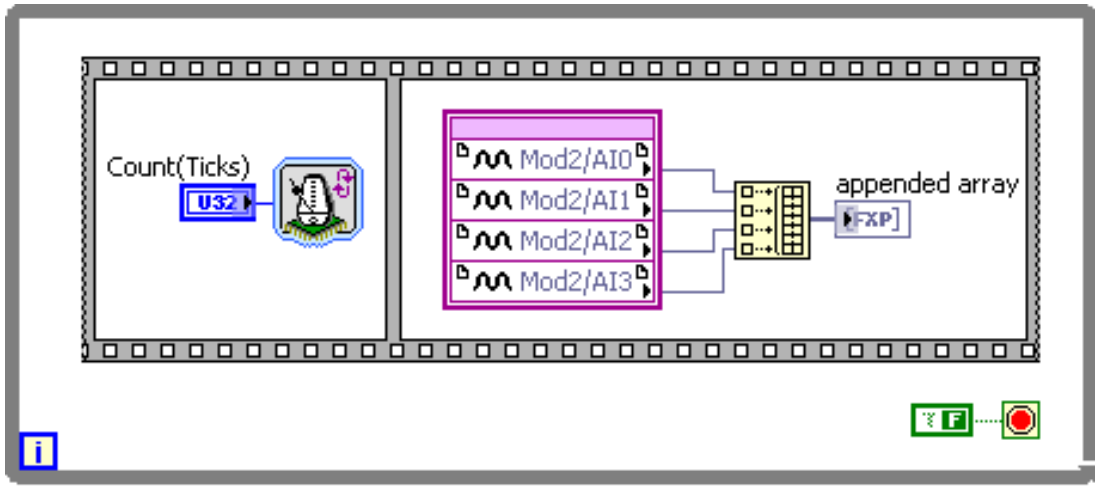
Device	RAM
1M Gate FPGA	81,920
3M Gate FPGA	196,608

- **If only enough time to do 1 optimization, do this optimization.**

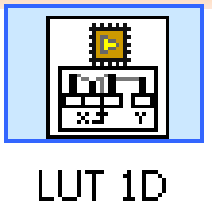
C. Basic Optimizations

Limit Front Panel Objects

- Avoid using Arrays on the Front Panel
  - Can quickly use large amounts of FPGA size
    - Each bit in the array uses its own flip-flop on the FPGA



C. Basic Optimizations

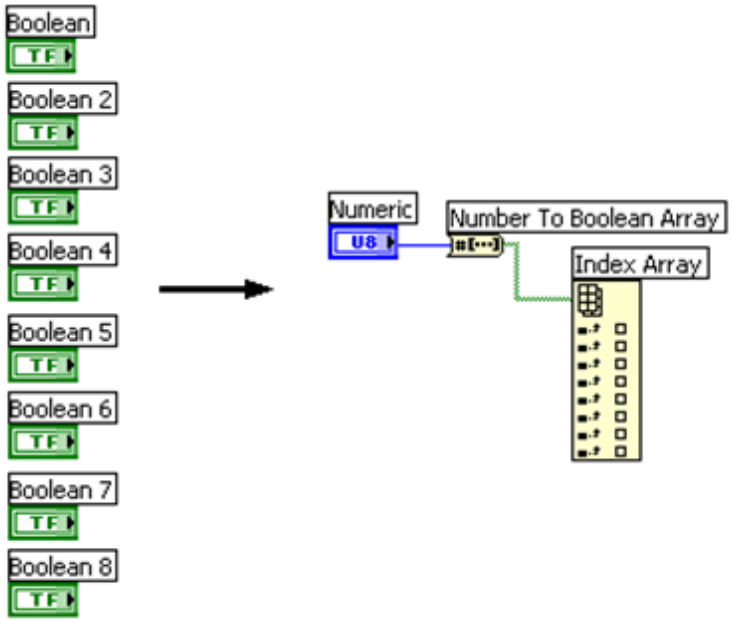


- Limit Front Panel Objects
  - Replace Large Front Panel Array Controls with Look-Up Table VI.
    - Provides a general-purpose block of initialized memory.
    - Use look-up tables to store waveforms for
      - Signal generation
      - Model nonlinear systems
      - Arithmetic computations.
  - Replace the array with a DMA FIFO to stream data

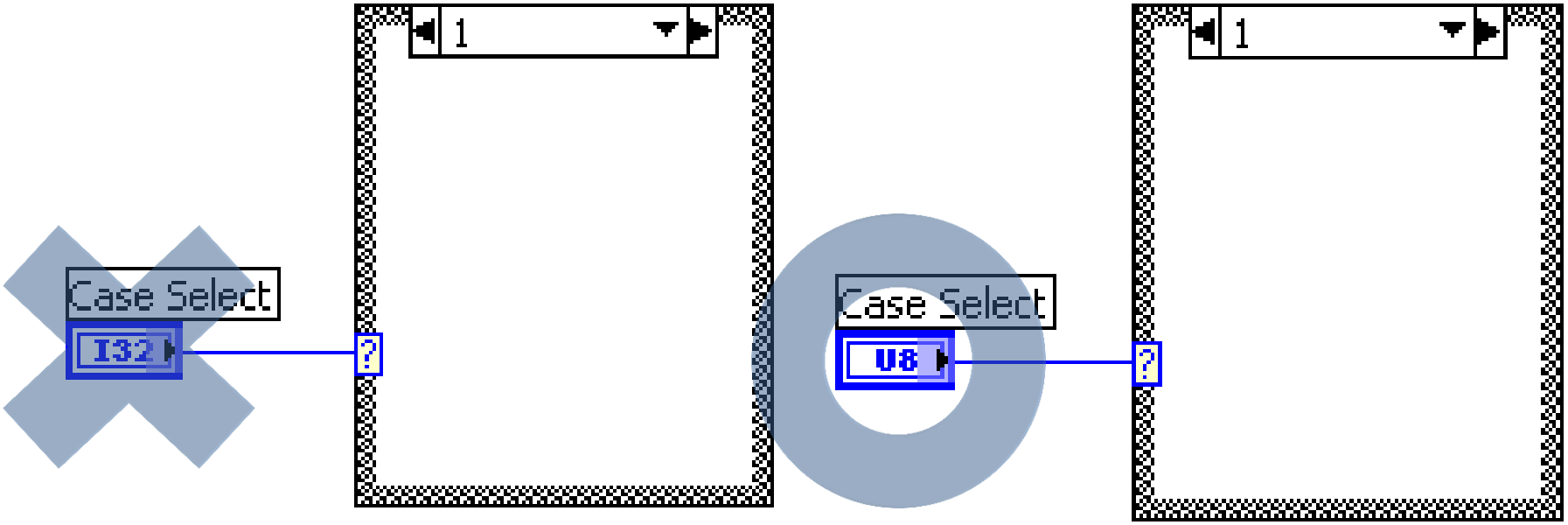
C. Basic Optimizations

Bitpack Boolean Logic

- Display data in integer as binary data
  - U8 Numeric Control can replace eight Boolean Controls
  - Maintains same information using 1/8th as many controls
  - Use functions to still manipulate data in same manner



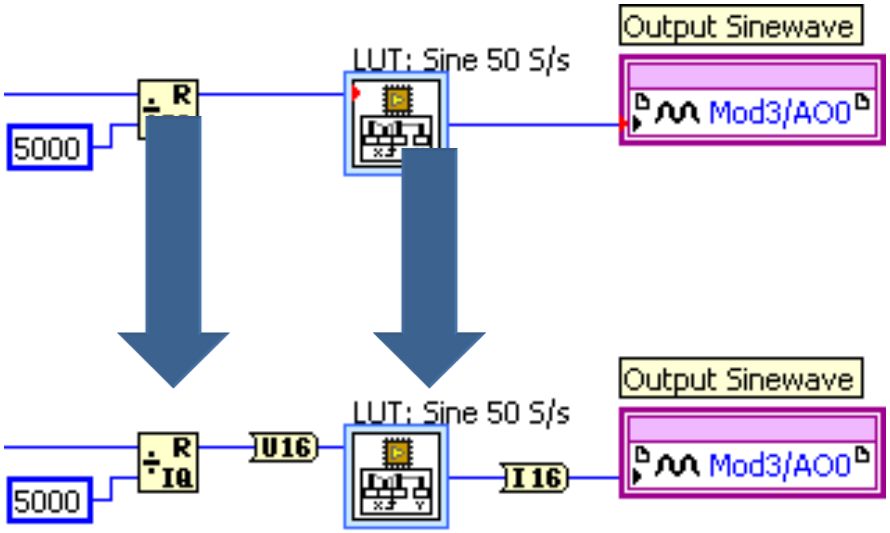
Use Small Data Types



C. Basic Optimizations

Use Small Data Types

- Eliminate coercion dots
  - Determine necessary input format
  - Insert conversion function
  - Intentional coercion creates a more efficient compile

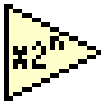


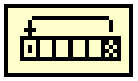
C. Basic Optimizations

Avoid Large Functions

- Not all functions are equal

-  Quotient Remainder

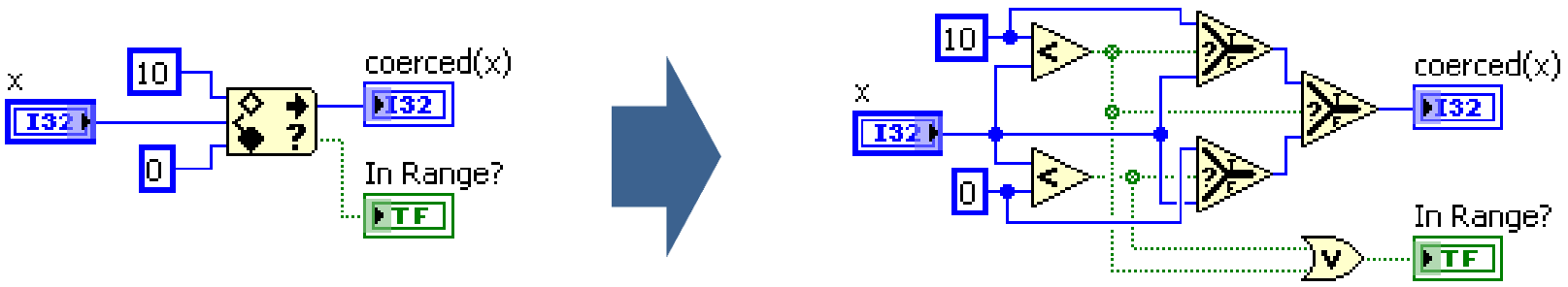
-  Scale By Power of 2 (free if use constant for power)

-  Array Functions (should use constants where possible)

C. Basic Optimizations

Optimize Comparisons

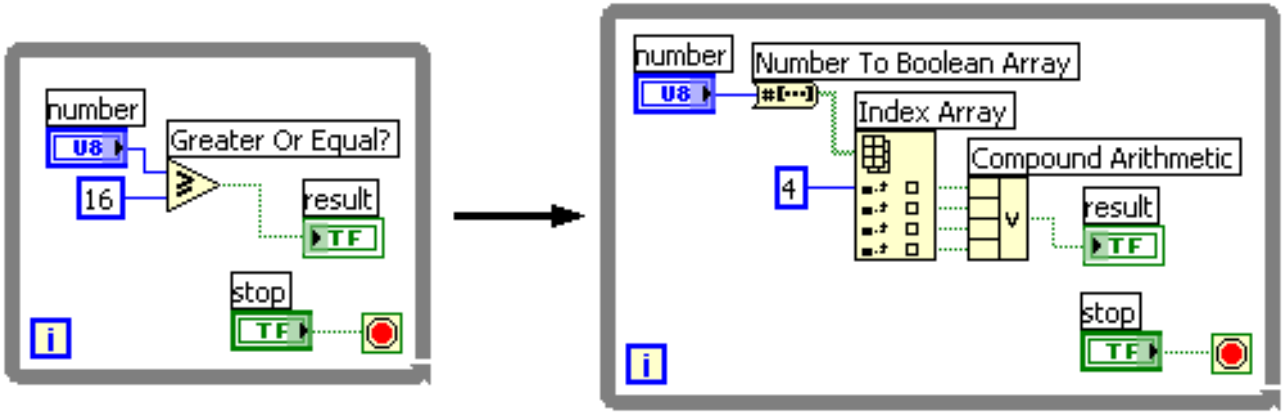
- Often comparisons can be replaced with lower level functions.
  - Refactor the code with simplified comparisons



C. Basic Optimizations

Optimize Comparisons

- Can replace comparison functions with bit logic
  - Easiest to compare power of 2
  - Must restructure code to change comparison value
- Same result, but uses half the FPGA resources and executes almost twice as fast



Reentrant vs. Non-Reentrant SubVIs

VI Type	FPGA Speed	FPGA Utilitization
Non-reentrant	Slower—Each call to the subVI waits until the previous call ends	Lower—Only one instance of the subVI exists on the FPGA
Reentrant	Faster—Multiple calls to the same subVI run in parallel	Higher—Each instance of the subVI uses space on the FPGA

C. Basic Optimizations

Reentrant vs. Non-Reentrant SubVIs

- By default, VIs created under an FPGA target are reentrant.
  - To make a subVI non-reentrant change VI Properties
  - Multiple copies of reentrant VIs allow for quick creation of similar code

D. Architecture Optimizations

- Dataflow within the FPGA
  - Parallel Operations
  - Pipelining
  - Single Cycle Timed Loops
  - Combining Optimizations

D. Architecture Optimizations

Dataflow within the FPGA

- Each function or VI takes a minimum of 1 clock tick
- Functions can run in parallel
- Some dependent functions must run in sequence
- Application can only run as quickly as the sum of items in a sequence.
- While Loops have a 2 clock tick overhead
  - If prior slide was in a loop
    - Required 3 clock ticks, plus 2 clock ticks for loop
    - Max Rate = 40 MHz / 5 = 8 MHz

D. Architecture Optimizations

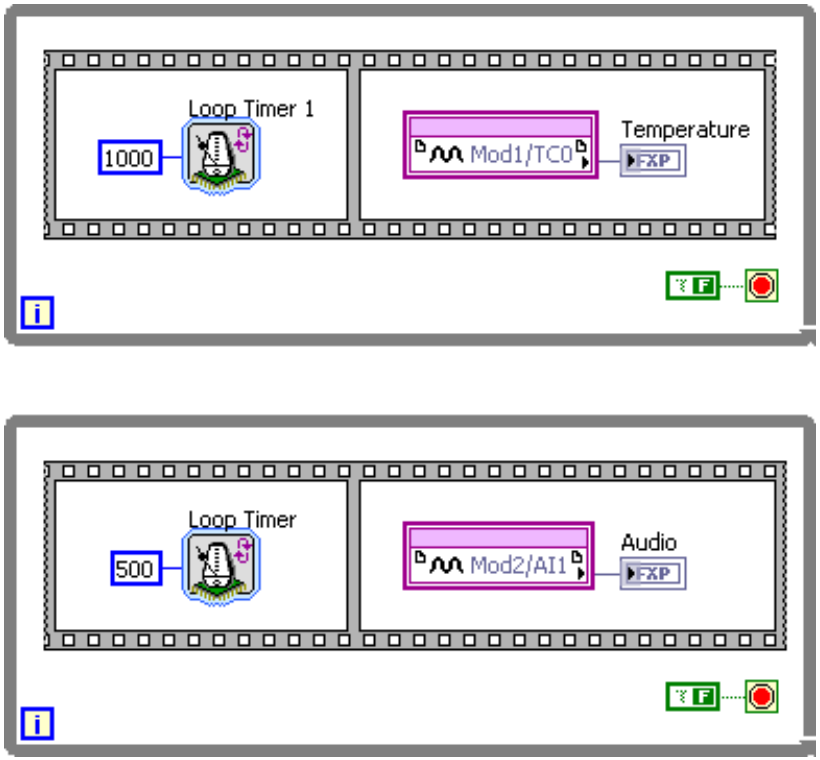
Parallel Operations

- Graphical programming promotes parallel code architectures
- LabVIEW Windows and Real-Time **serialize** execution
- LabVIEW FPGA implements **truly parallel** execution

D. Architecture Optimizations

Parallel Operations

- Two parallel loops with different sampling rates
  - Run in parallel because there are no shared resources between the two loops.

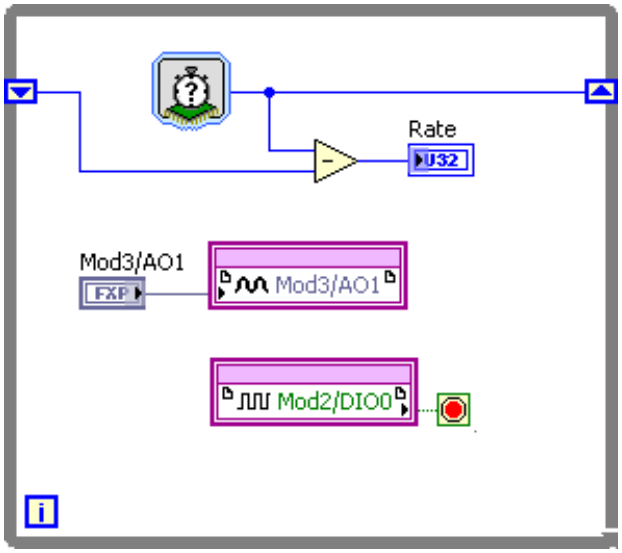


D. Architecture Optimizations

Parallel Operations

- Loop rates limited by longest path
  - A0 takes about 35 ticks, DI takes 1 tick (HW Specific)
  - DI limited by AO when in same loop

38 Ticks ~ 1uSec

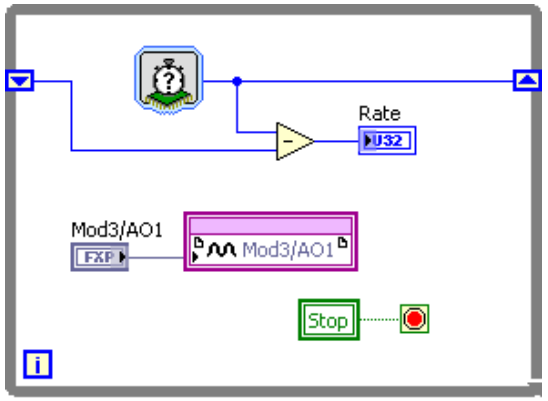


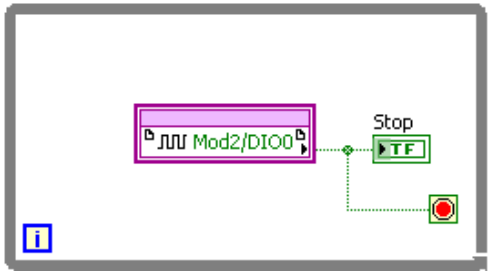
D. Architecture Optimizations

Parallel Operations

- Loop rates limited by longest path
  - A0 takes about 35 ticks, DI takes 1 tick (HW Specific)
  - Separate functions to allow DI to run independent of AO
  - This allows DI to be sampled 10 times faster by using a separate loop

38 Ticks ~ 1uSec





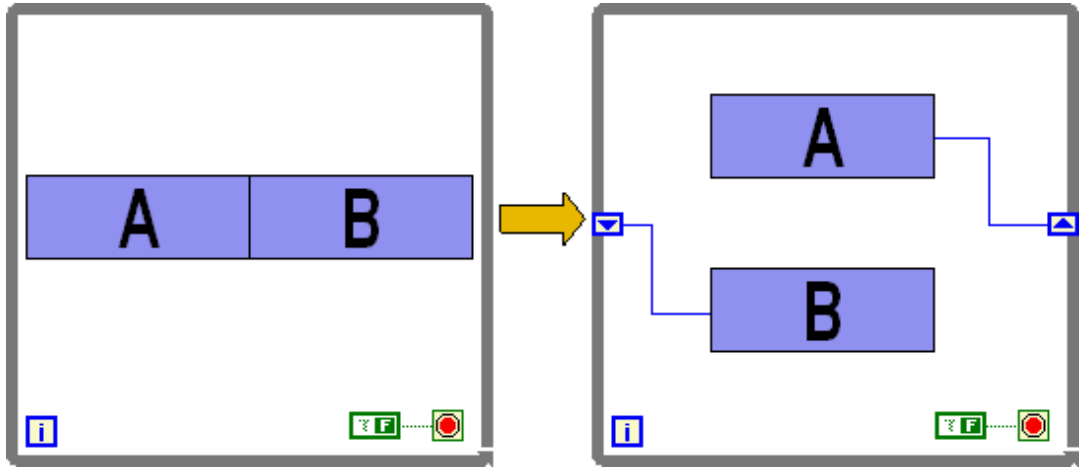
4 Ticks ~ .1 uSec

D. Architecture Optimizations

Pipelining

Within a loop you can split up your code into different loop iterations to reduce the duration of each iteration

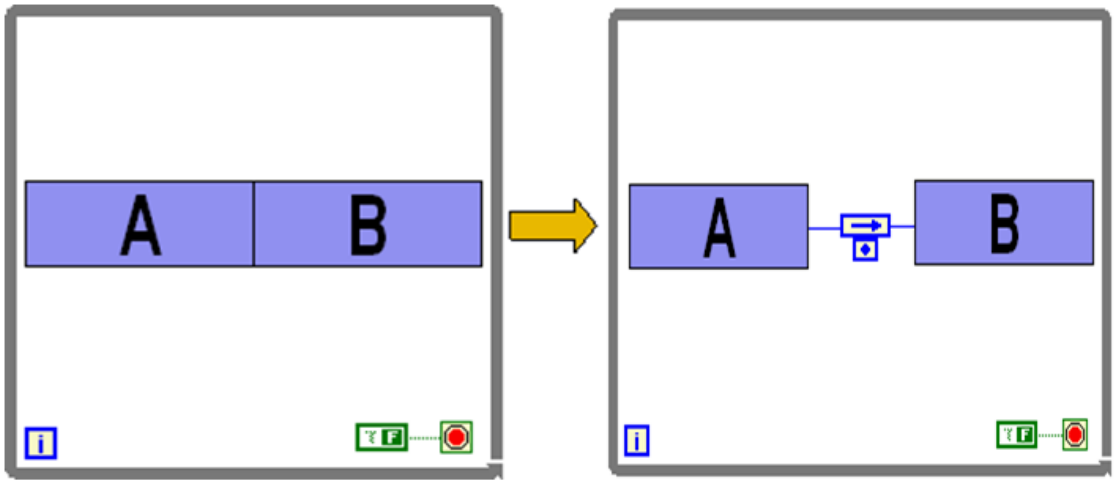
- Handle different parts of the process flow in parallel within one loop iteration
- Pass data to next piece of code using shift registers



Pipelining

Can maintain look and feel of original application by using Feedback Nodes

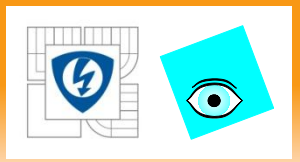
- Same functionality as a Shift Register
- Maintains more congruous VI appearance



D. Architecture Optimizations

Pipelining is a technique you can use to increase the throughput or speed of the FPGA VI.

- Takes advantage of the parallel processing capabilities of the FPGA
  - By using parallel processing increases the efficiency of sequential code
- Must divide code into discrete steps
  - Wire inputs and outputs of each step to Feedback Nodes or shift registers



Pipelining

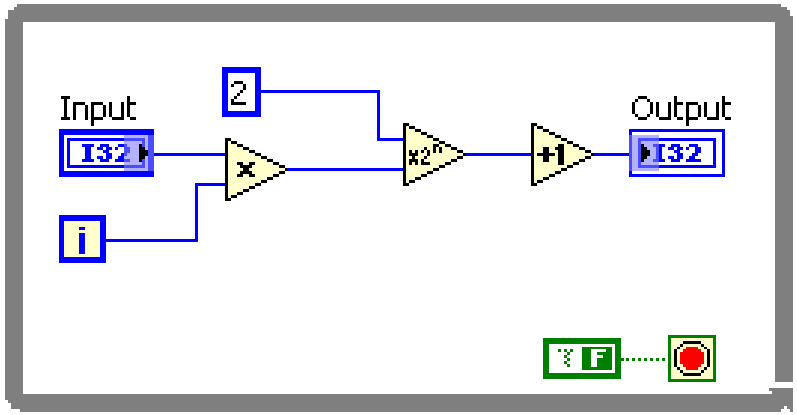
Pipelining Drawbacks

- Require code to be restructured
  - Greatly reduced by using Feedback Nodes
- N-cycle delay before valid output data
  - Where *N = #Discrete Steps Used – 1*

Pipelining

Basic Example

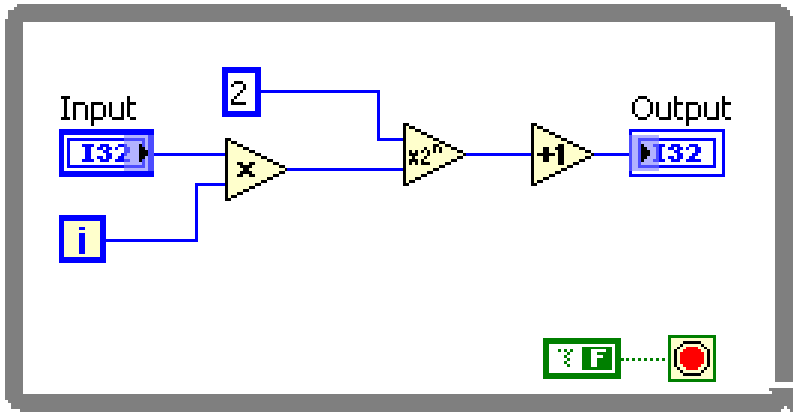
- Simple set of code with inputs and outputs
- Takes 7 clock cycles to execute
  - 5 cycles for the code, 2 for the loop
- If Input = 1 for i = 0..3
  - Output = 1, 5, 9, 13

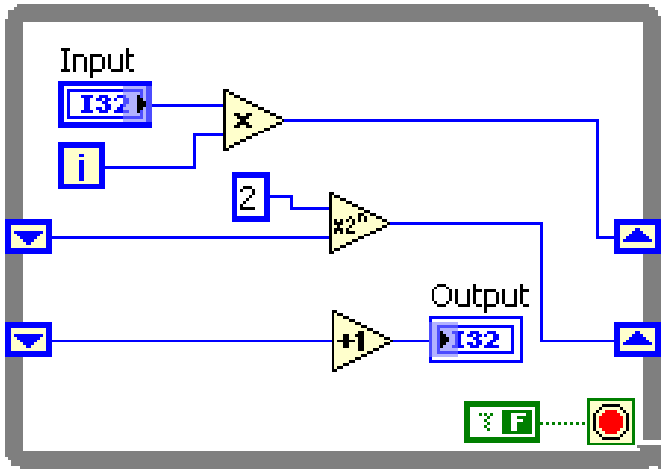


Pipelining

Basic Example

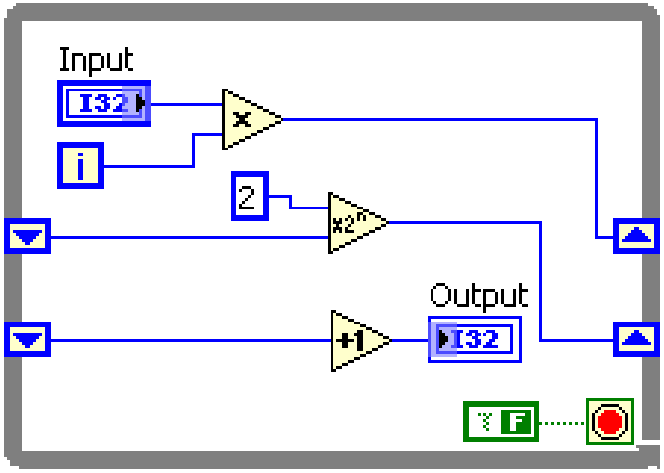
- Pass data to next step of code by using shift registers
- Takes 5 clock cycles to execute
  - 3 cycles for the code, 2 for the loop
- If Input = 1 for i = 0..3
  - Output = ?, ?, 1, 5



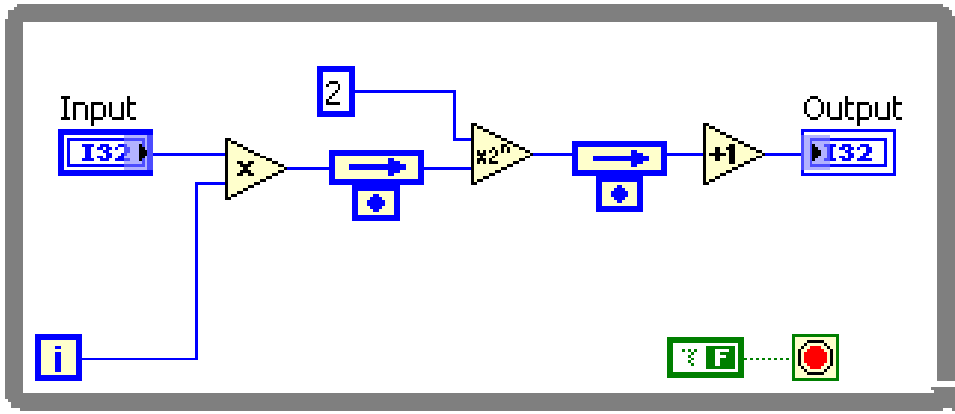


Pipelining

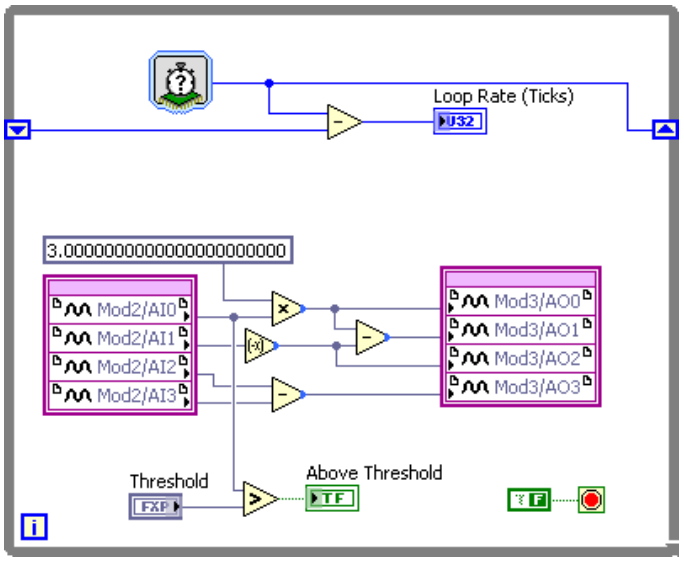
- Basic Example
- Can maintain look and feel of original application by using Feedback Nodes in sequence with code
  - Same functionality as a Shift Register
  - Maintains more congruous VI appearance

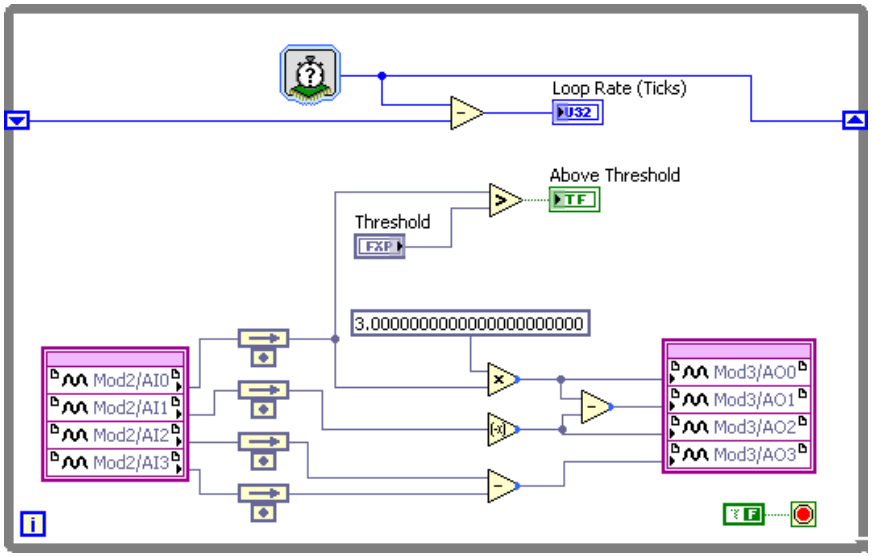


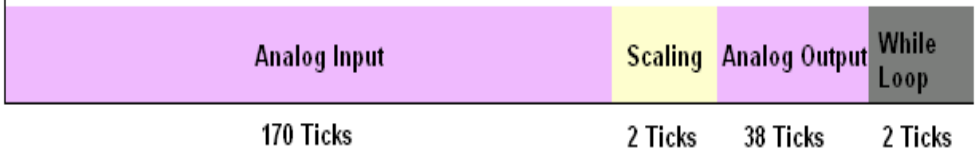
=



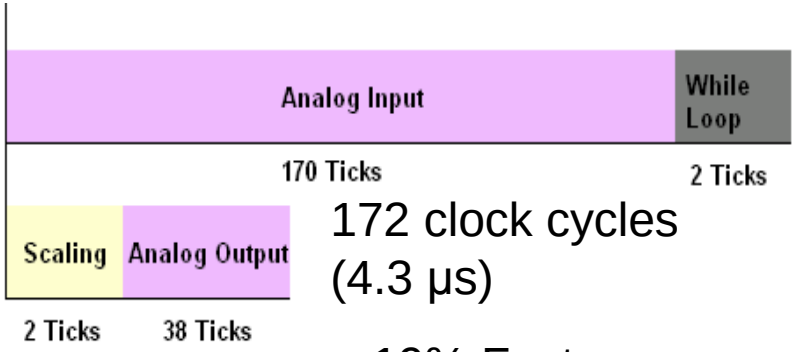
Pipelining







212 clock cycles (5.3 µs)



172 clock cycles (4.3 µs)

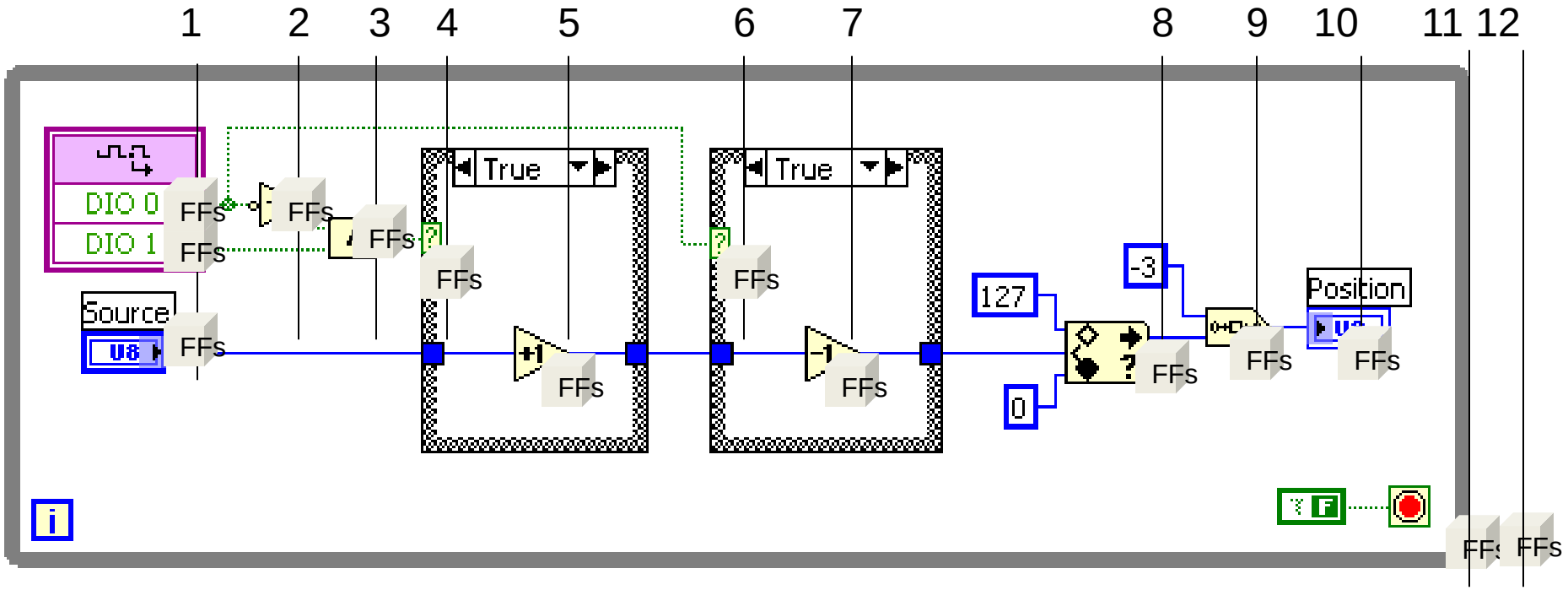
~ 19% Faster

D. Architecture Optimizations

Pipelining

– What to do if your diagram executes too slowly?

– 12 clock cycles

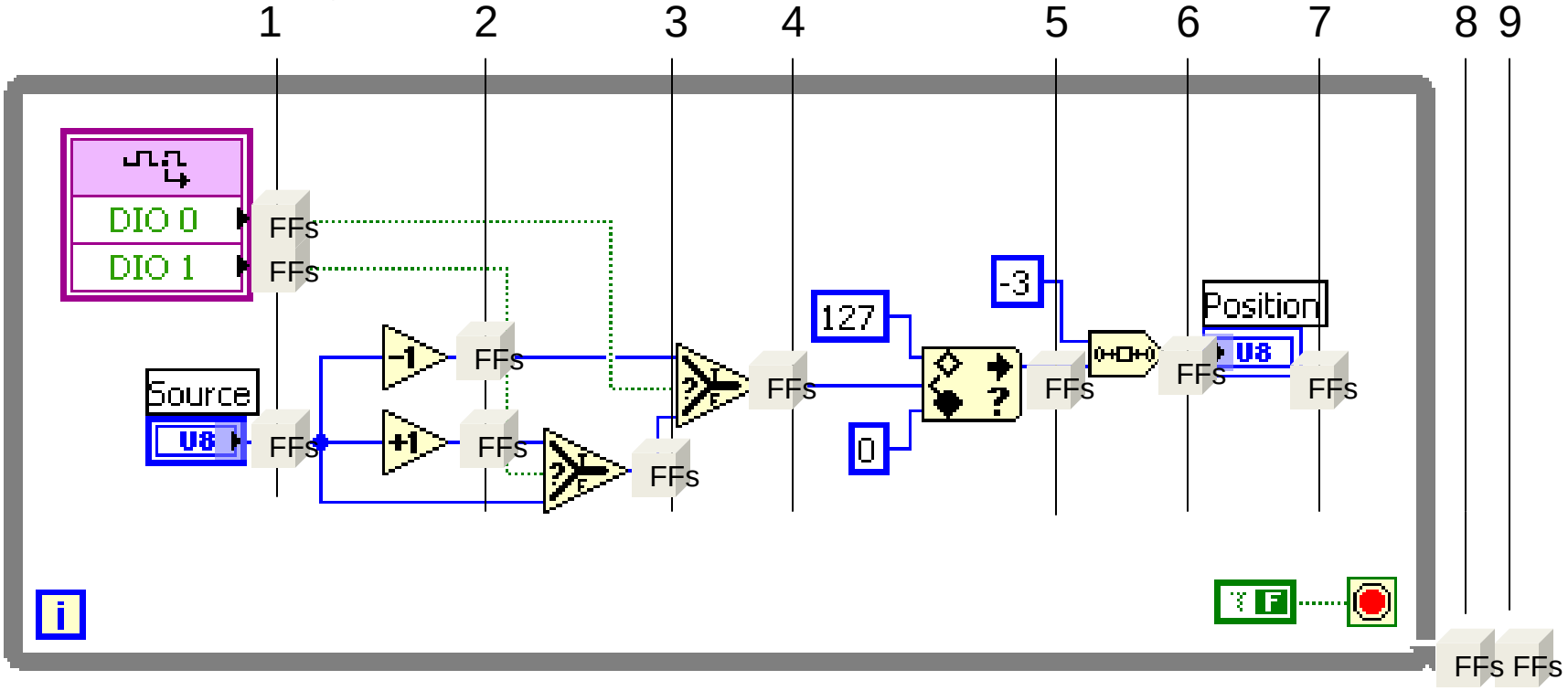


D. Architecture Optimizations

Pipelining

– Shorten the longest path

– 9 clock cycles



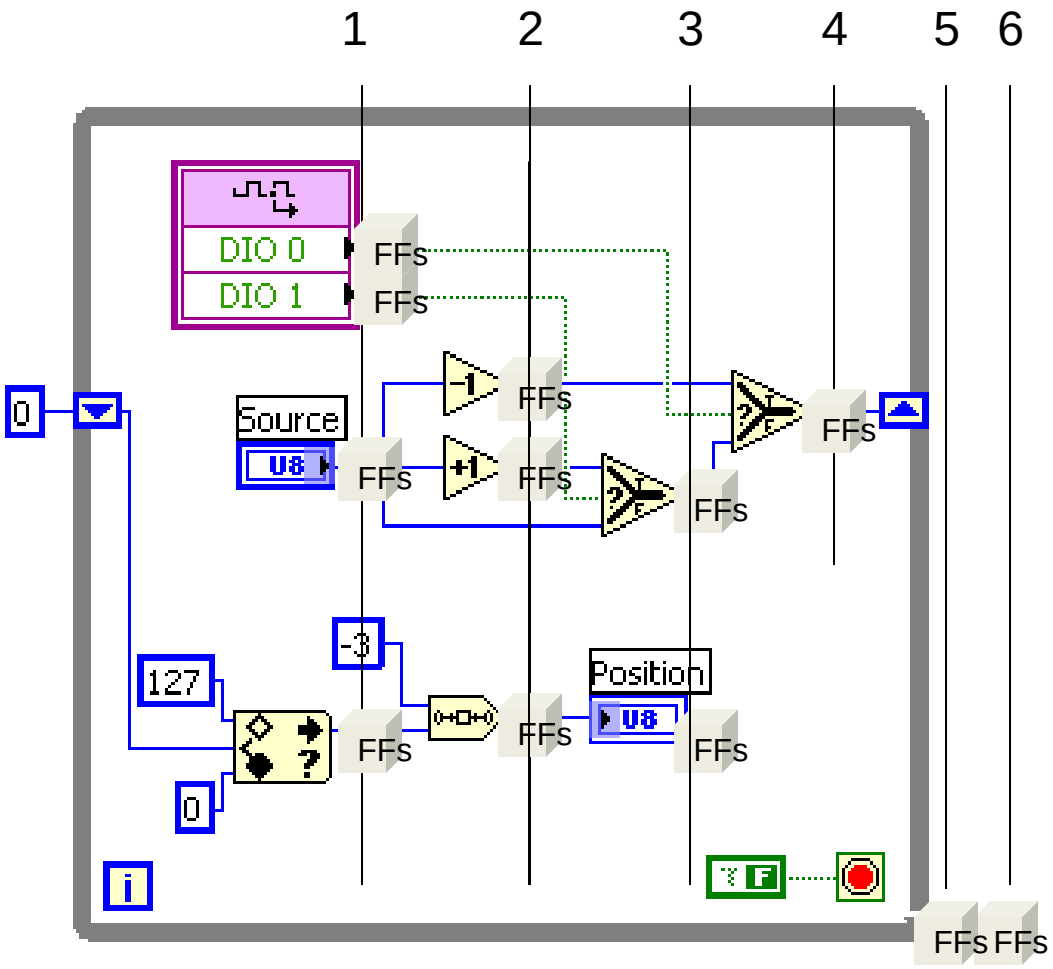
D. Architecture Optimizations

Pipelining

– Watch out for

  pipeline
  effects including
  increased latency

– 6 clock cycles

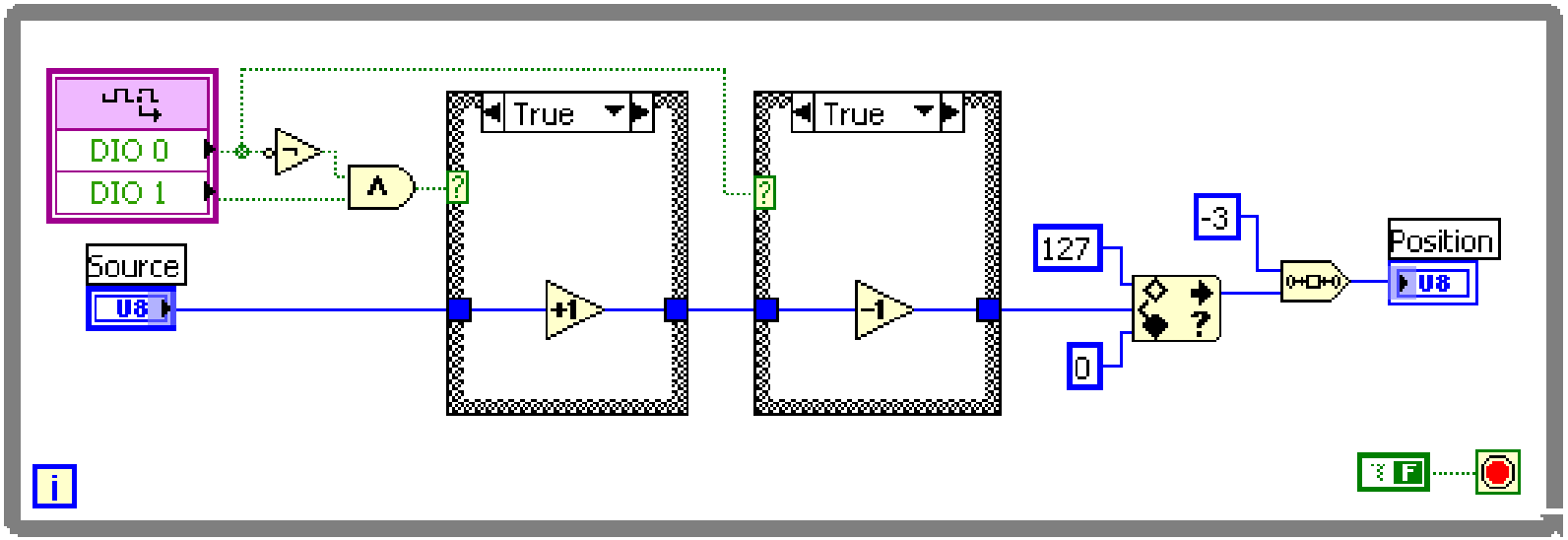


But can we go faster?

D. Architecture Optimizations

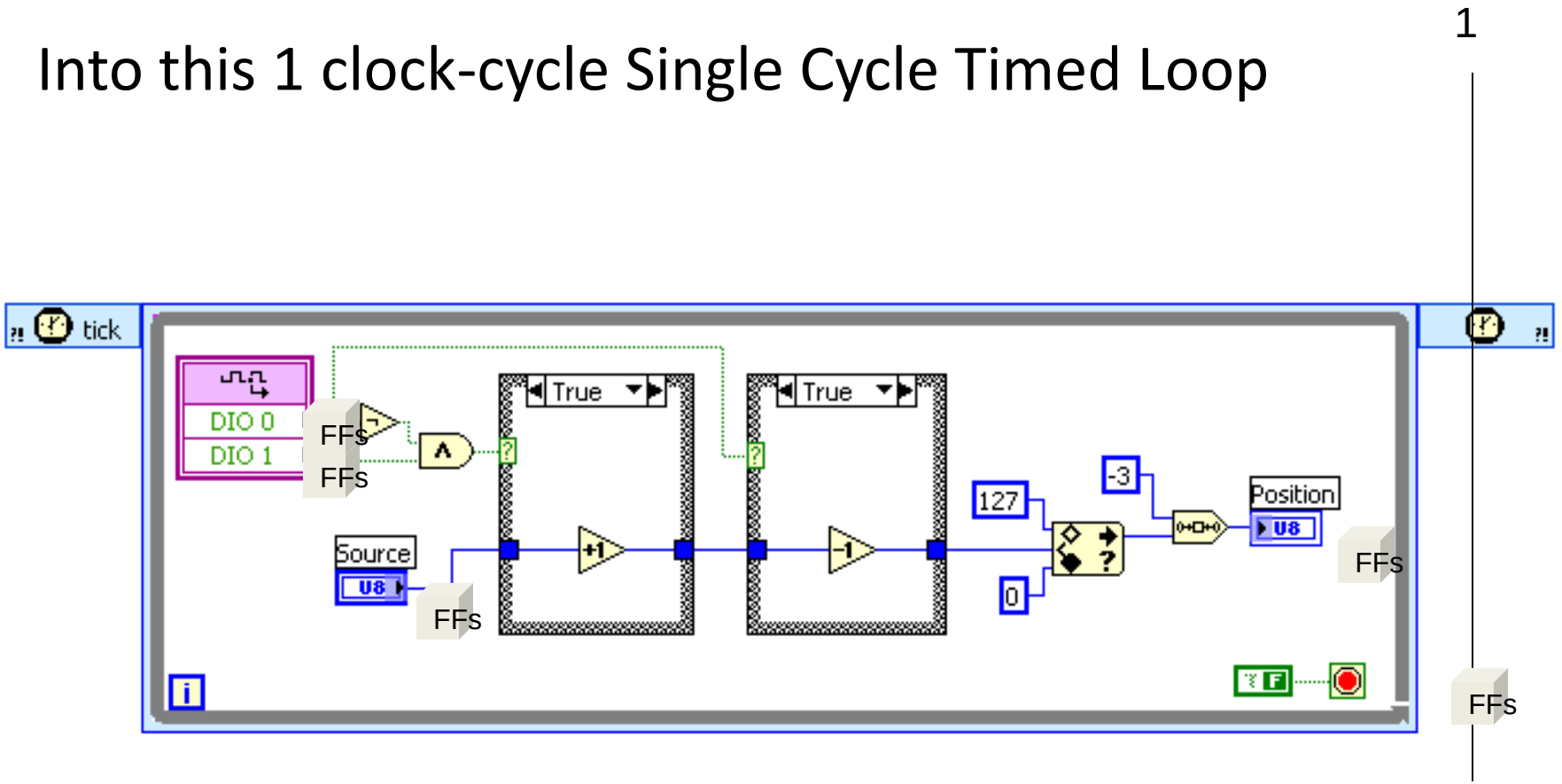
Single Cycle Timed Loop (SCTL)

Can use a Single-Cycle Timed Loop to turn this 12 clock-cycle While Loop…



Single Cycle Timed Loop

Into this 1 clock-cycle Single Cycle Timed Loop



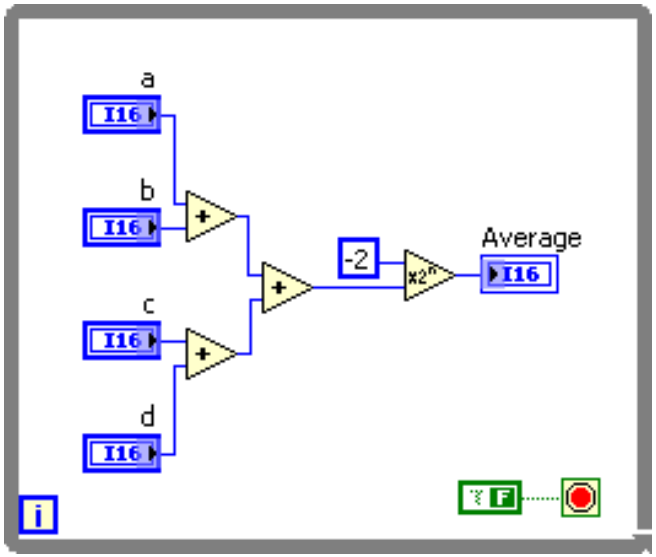
Single Cycle Timed Loop

Loop contents execute in a single clock period

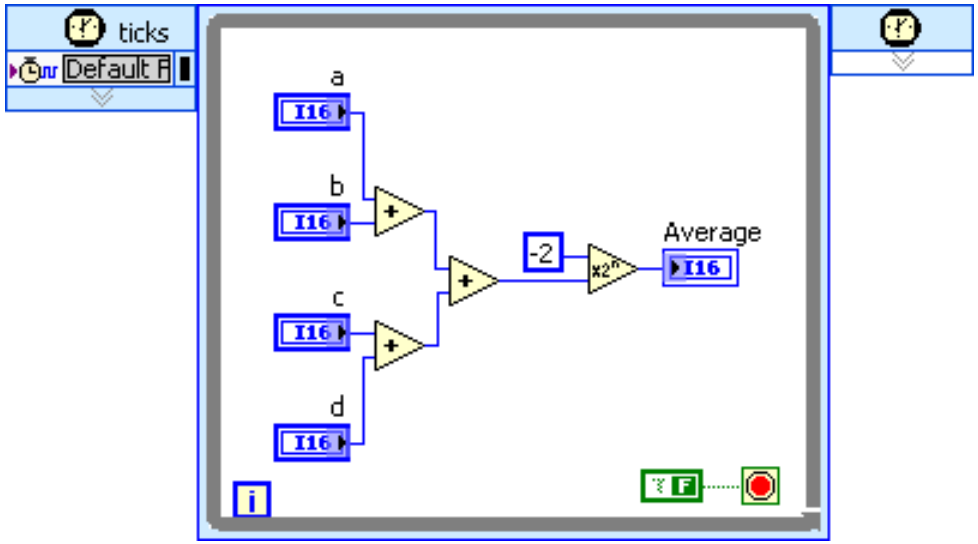
- Some VIs and functions cannot be used in the loop at all
  - Analog input, analog output (Most HW)
  - Nested loops
  - Any that require more than a single clock cycle to execute
  - Shared resources (Arbitration)
  - Loop Timer
  - Wait
- Combinatorial path length becomes critical
  - May require combining optimizations

Single Cycle Timed Loop

Saved 5 Ticks by placing this code in a SCTL



7 Ticks
512 out of 5120 Slices 10%



1 Tick
454 out of 5120 Slices 8%

Single Cycle Timed Loop

SCTL limited by propagation delays through the FPGA circuitry

- If total combinatorial path propagation takes longer than 1 clock cycle, compile fails
  - Try to reduce path as much as possible before using SCTL