Introduction to LabVIEW FPGA

Carlos Pazos

Embedded Software Product Marketing Manager

Embedded Systems Challenges

High-speed I/O and analysis

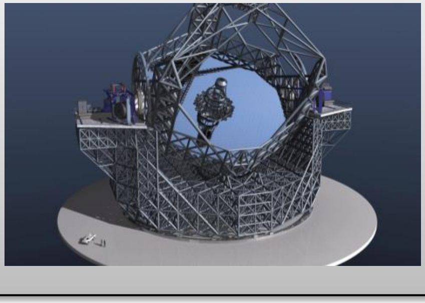
An oil well pump monitoring system requires high-speed I/O and analysis to catch momentary pressure spikes and vibration indications

System uptime and reliability

A bio-refinery requires long system uptimes and high reliability for failsafe control systems

High-speed or deterministic control

An Extremely Large Telescope (ELT) requires control of nanometric position actuators



Other Systems Challenges

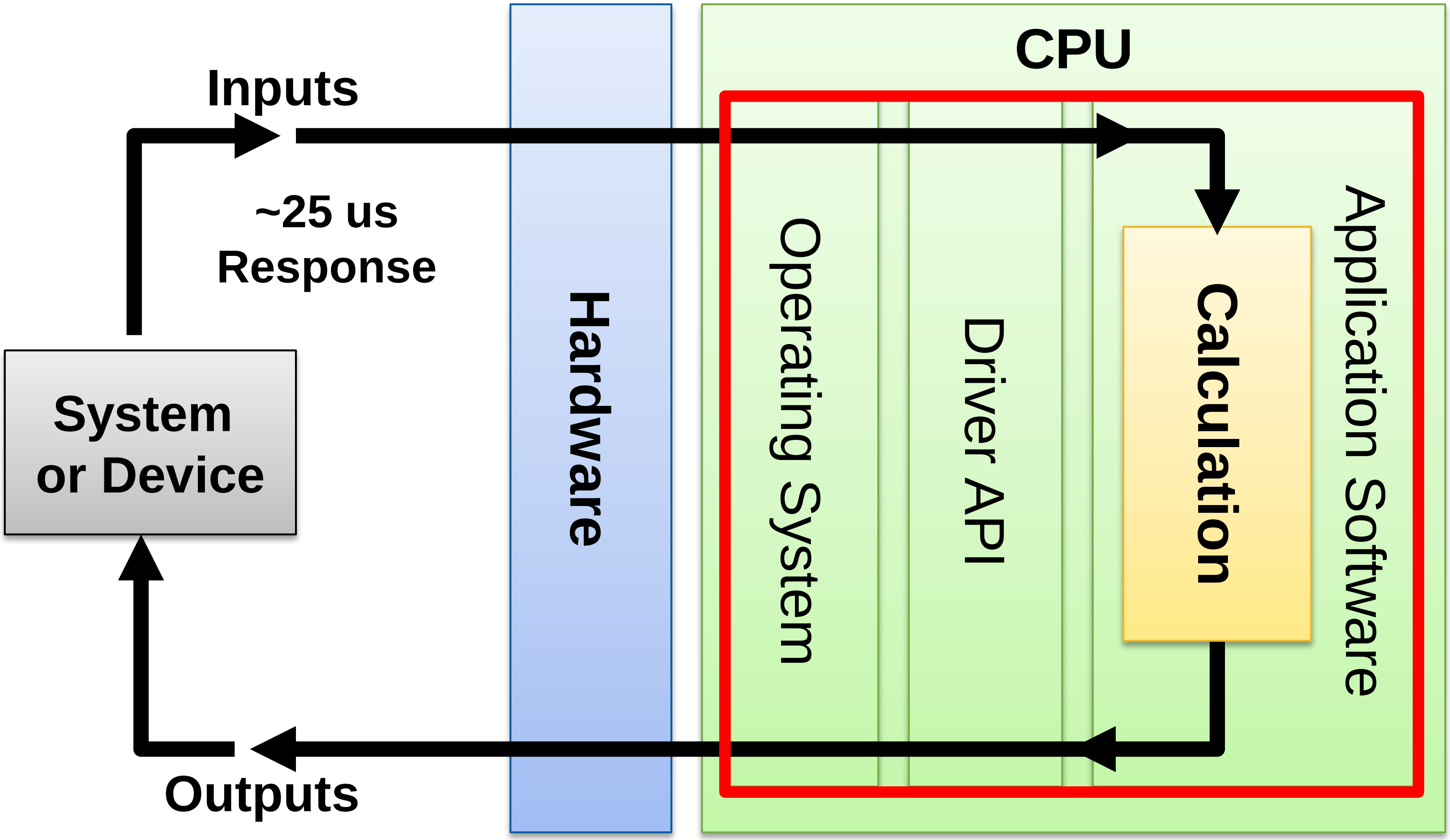
	Channel Density	Programmable FPGA	Analog Performance
Automated Test General Purpose Test	Lower Capitol Cost	Faster Filtering and Spectrum Analysis	High Accuracy, Repeatability, and Calibration
	Improved Test Time with More Test Points	Complex and Deterministic Triggers	Best Channel and Device Synch with PXI
High-Performance Embedded LIDAR RADAR Signal Intelligence Beam Position Monitoring General Physics	More Signals for More Precise Results	Reduce Data Processing time	>10 ENOB on 8 Channels
	High Density for Compact Design	Customizable Algorithms and Memory Control	High Bandwidth for Spectral Analysis

A Processor Based Approach

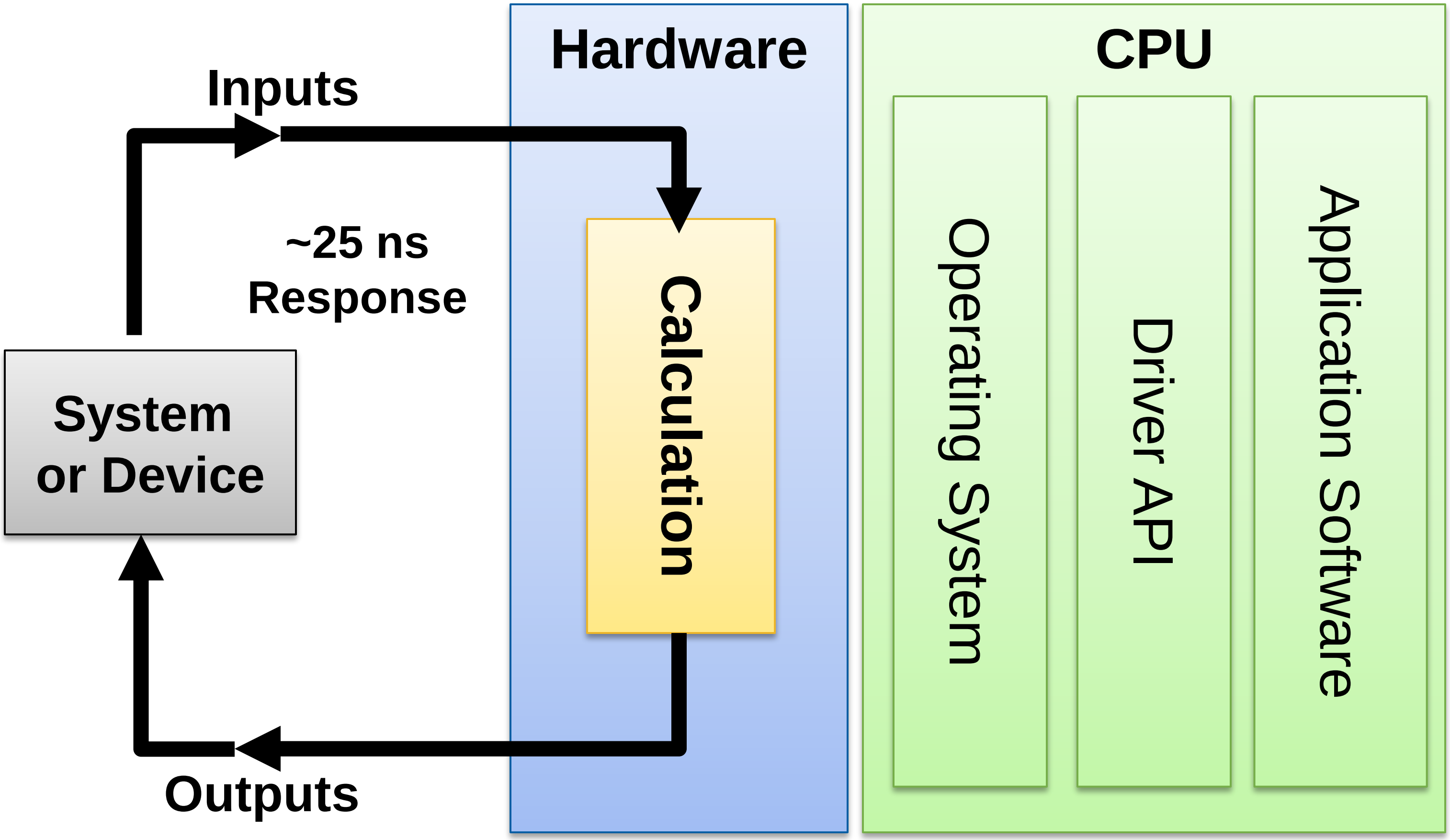
Benefits:

- Can easily be programmed
- Well-suited for high-precision floating point calculations
- Well-suited for networking and peripheral I/O

Processor Based Approach



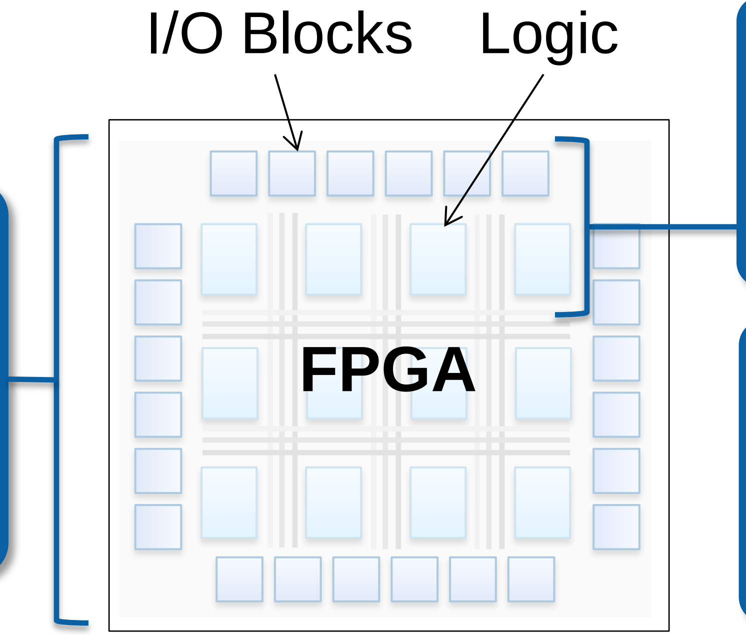
Decision Making in Hardware



FPGA Technology

What is an FPGA?

- Software defined hardware
- No operating system is needed for execution of logic



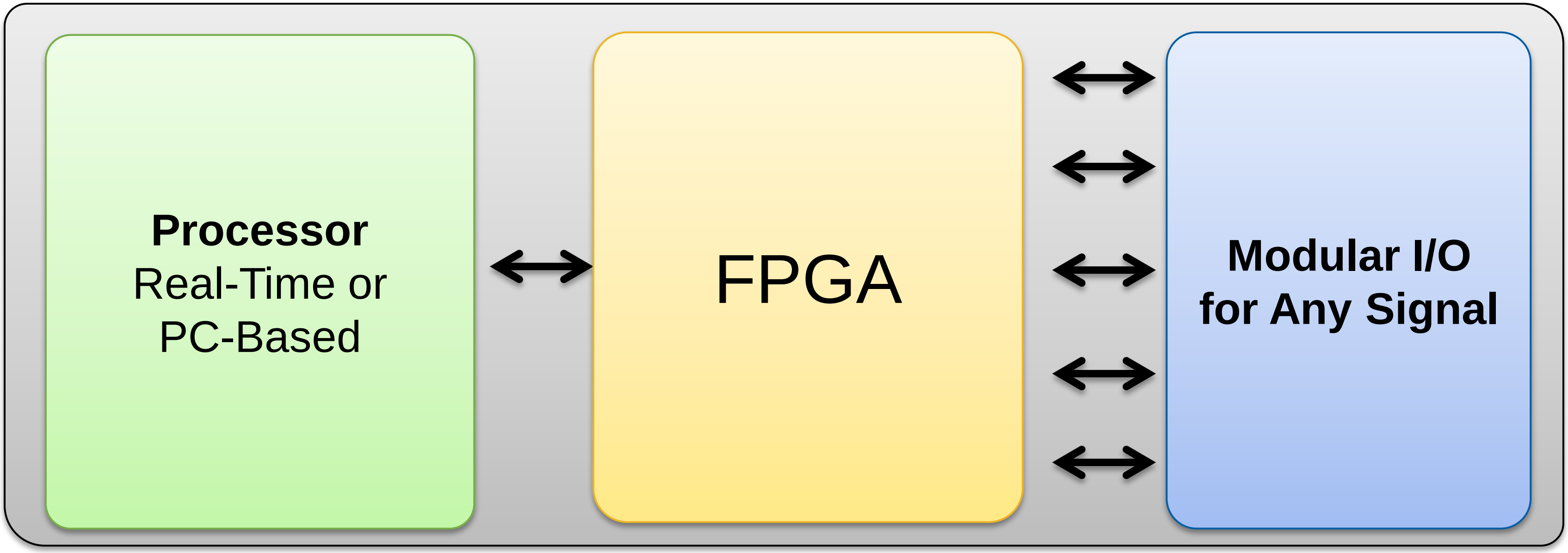
Dedicated logic in silicon for highest reliability

Inline signal processing with minimal latency

High-speed and deterministic control

The NI Approach

We call this the LabVIEW RIO architecture.



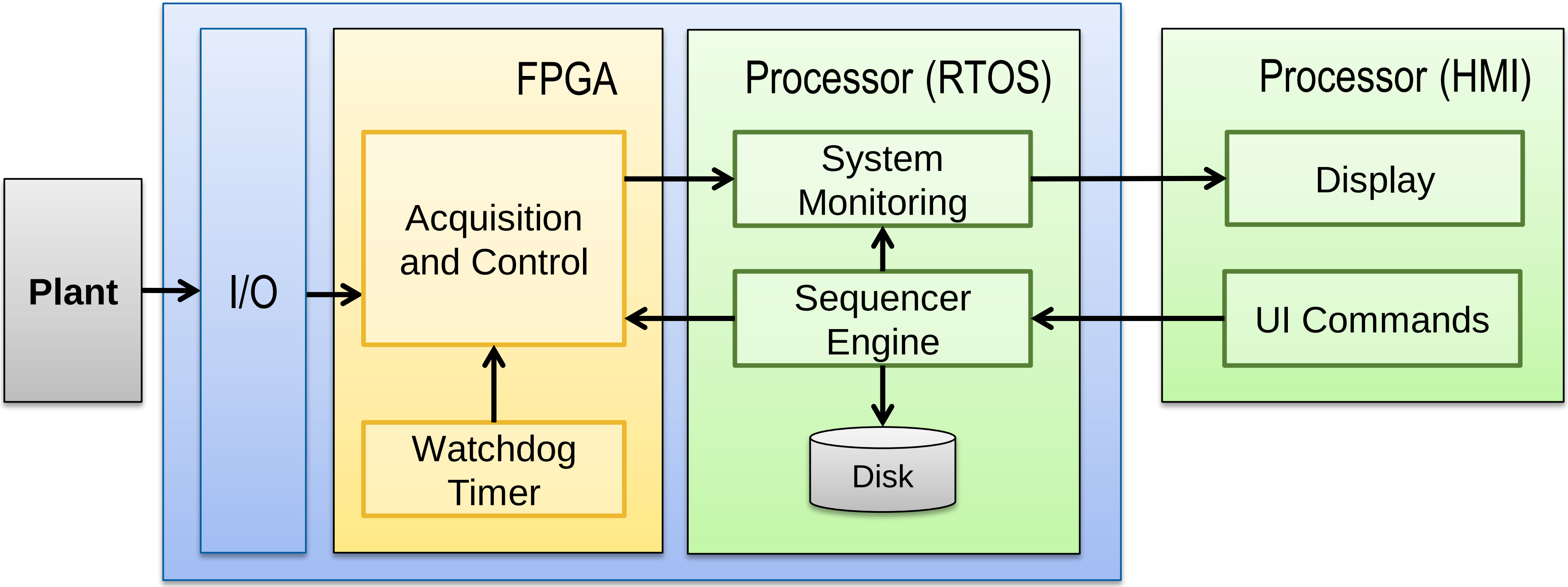
Processor
Real-Time or
PC-Based

FPGA

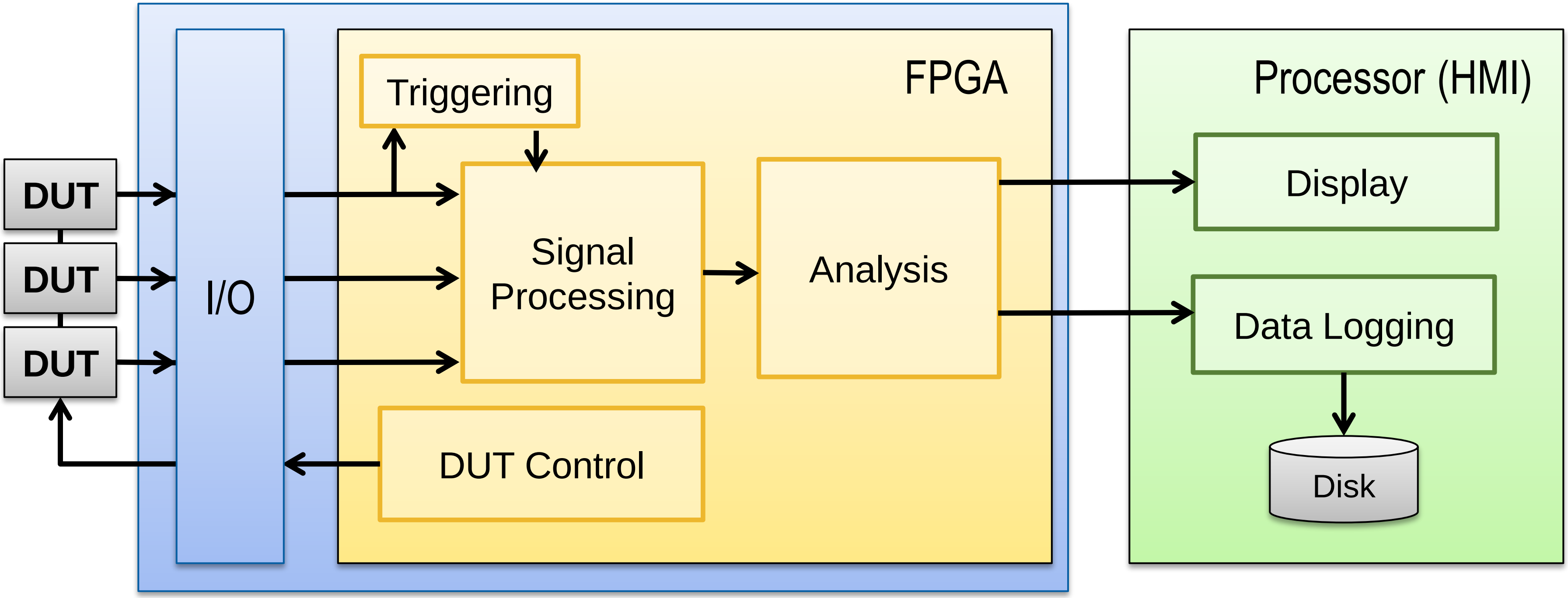
Modular I/O
for Any Signal

Highly Productive **LabVIEW** Graphical Programming Environment
for Programming Host, FPGA, I/O, and Bus Interfaces

High-Speed Control Application



High-Throughput Test Application



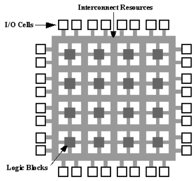
Embedded Software Development Challenge

Tools

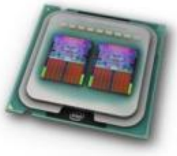
- Math (.m file script)
- Simulation (Hybrid)
- User Interface (HTML)
- FPGA (VHDL, Verilog)

- Host Control (C, C++, .NET)
- DSP (Fxd pt C, Assembly)
- H/W Driver (C, Assembly)
- System Debug

Targets



FPGAs



Multicore Processors

- Embedded development requires multiple software tools
- Parallel processing increases system complexity
- Software tools don’t address system design

Long learning curves
Limited reuse
Need for “specialists”

Increased costs
Increased time-to-result

Graphical System Design for FPGAs

NATIONAL INSTRUMENTS LabVIEW™

Graphical System Design Platform

LabVIEW FPGA

Xilinx IP XILINX®

HLS Technology

VHDL

Value

Rugged

Performance

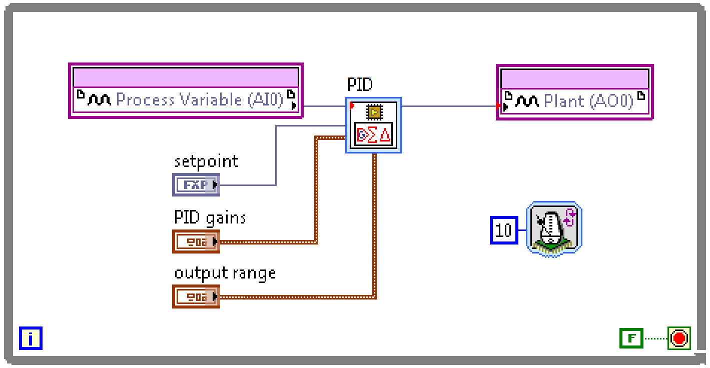
High Performance

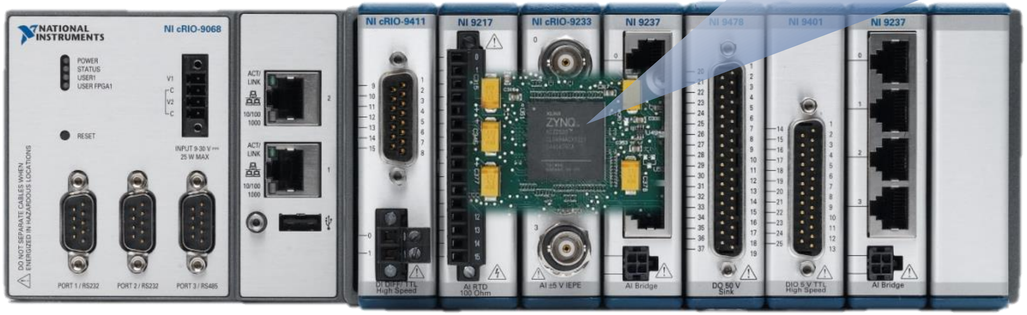
High Performance

NI RIO Hardware

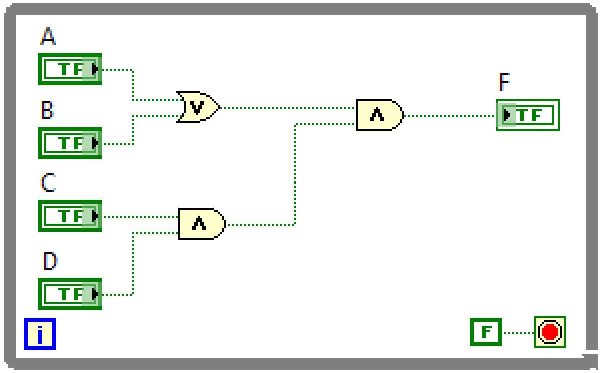
LabVIEW FPGA Module

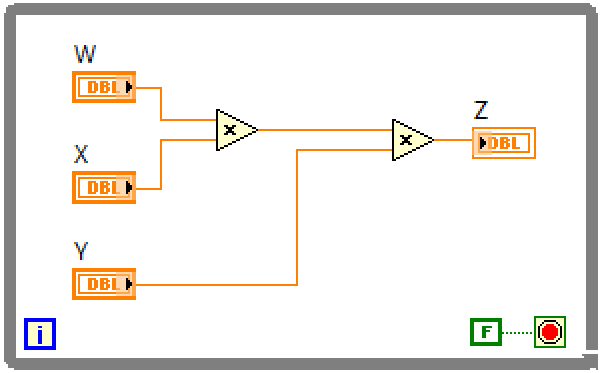
- Use LabVIEW to design hardware
- Offload the most critical pieces of your application
  - High speed control
  - Inline signal processing
  - Custom protocols
  - Custom timing, triggering, and synchronization
  - Fast stimulus/response testing

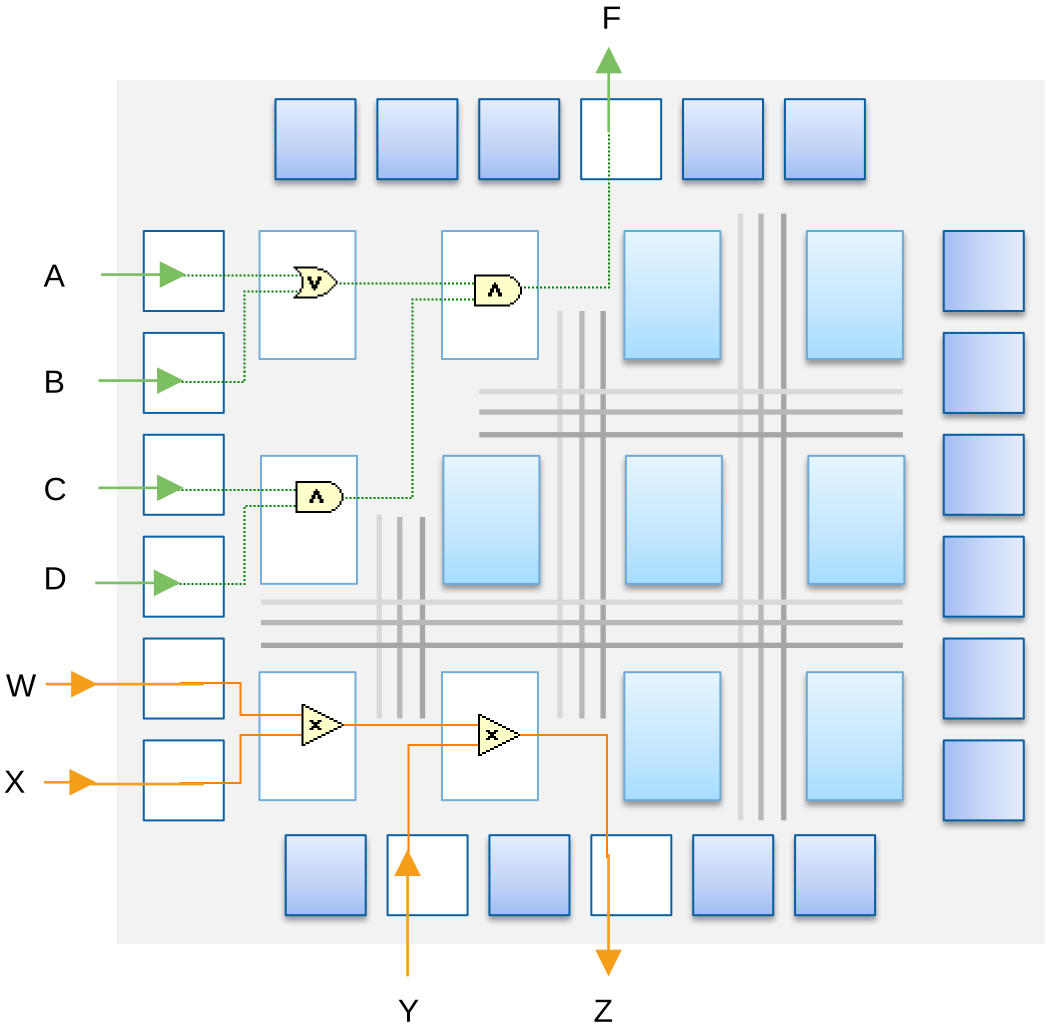




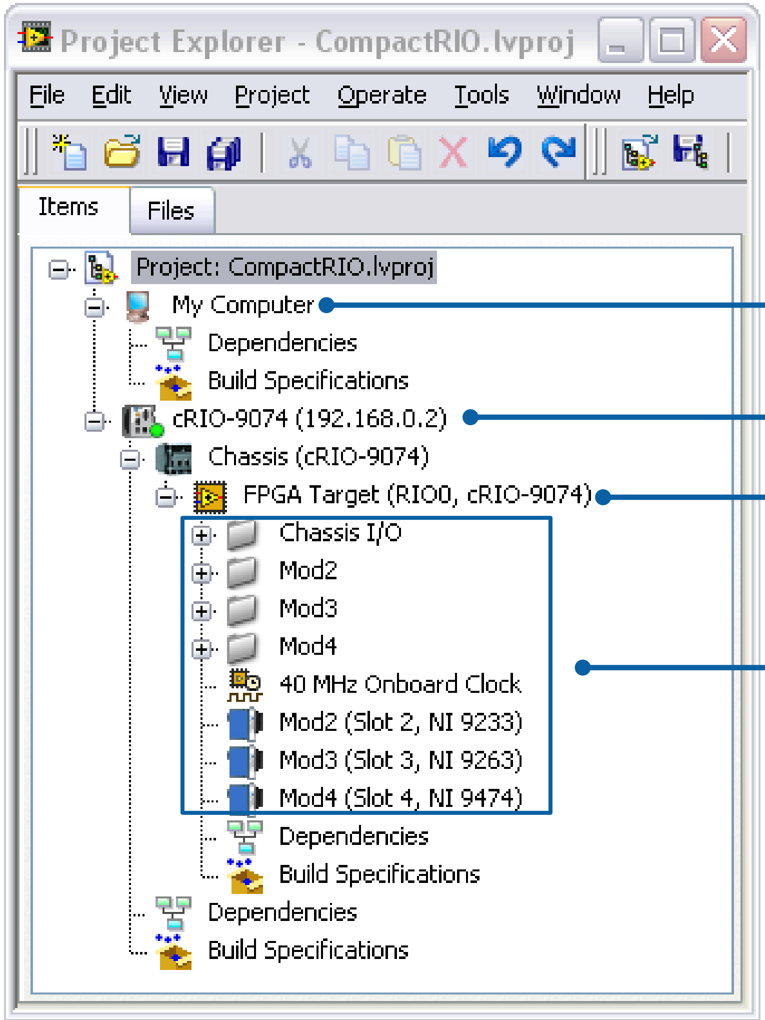
Mapping LabVIEW to an FPGA







Using the LabVIEW Project


Project Explorer - CompactRIO.lvproj
File Edit View Project Operate Tools Window Help
Items
Files

- Project: CompactRIO.lvproj
  - My Computer
    - Dependencies
    - Build Specifications
  - cRIO-9074 (192.168.0.2)
    - Chassis (cRIO-9074)
      - FPGA Target (RIO0, cRIO-9074)
        - Chassis I/O
        - Mod2
        - Mod3
        - Mod4
        - 40 MHz Onboard Clock
        - Mod2 (Slot 2, NI 9233)
        - Mod3 (Slot 3, NI 9263)
        - Mod4 (Slot 4, NI 9474)
        - Dependencies
        - Build Specifications
    - Dependencies
    - Build Specifications

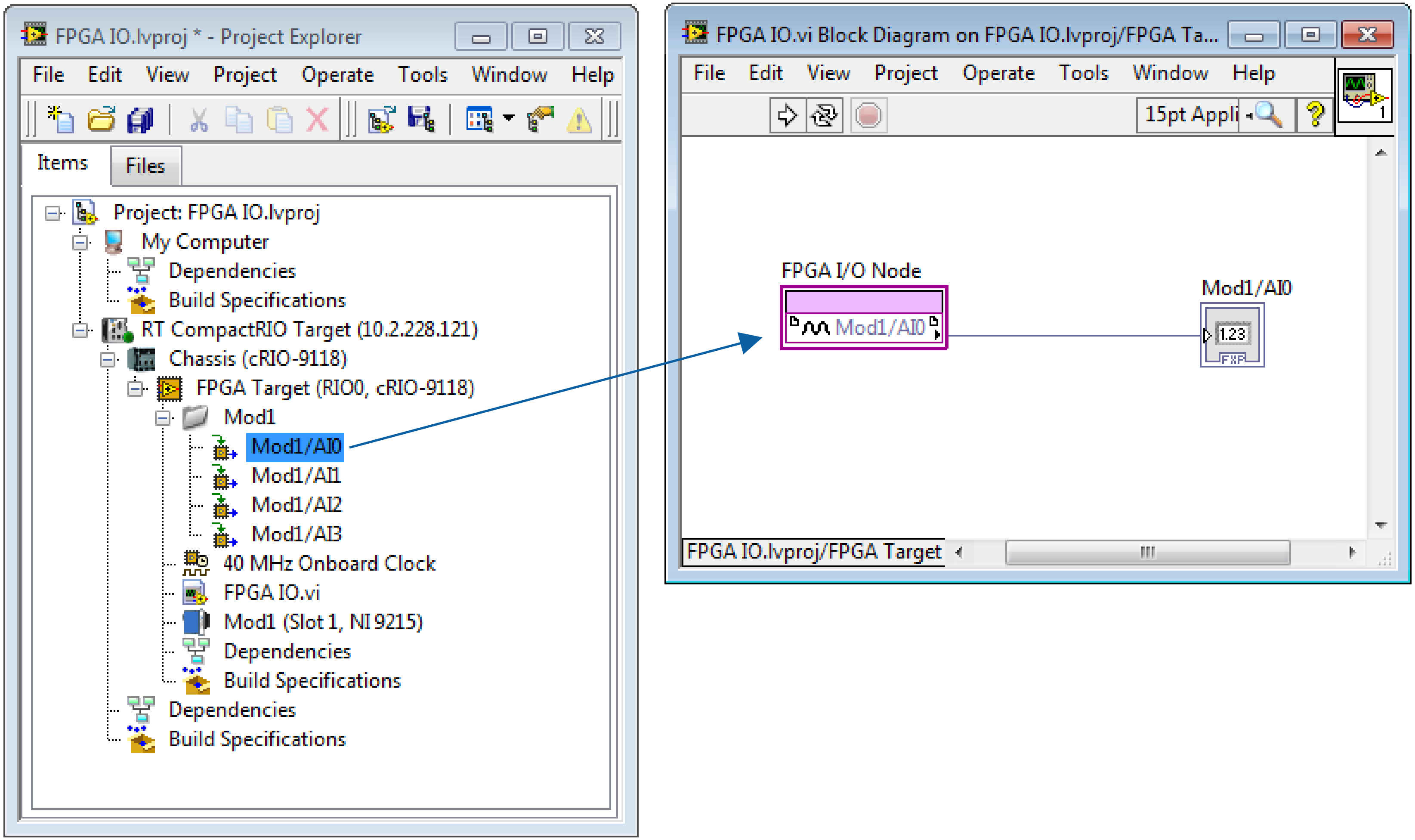
Development PC Target

Embedded Processor Target

FPGA Target

FPGA Resources (I/O, clocks, IP, etc.)

Abstraction of Hardware Complexities



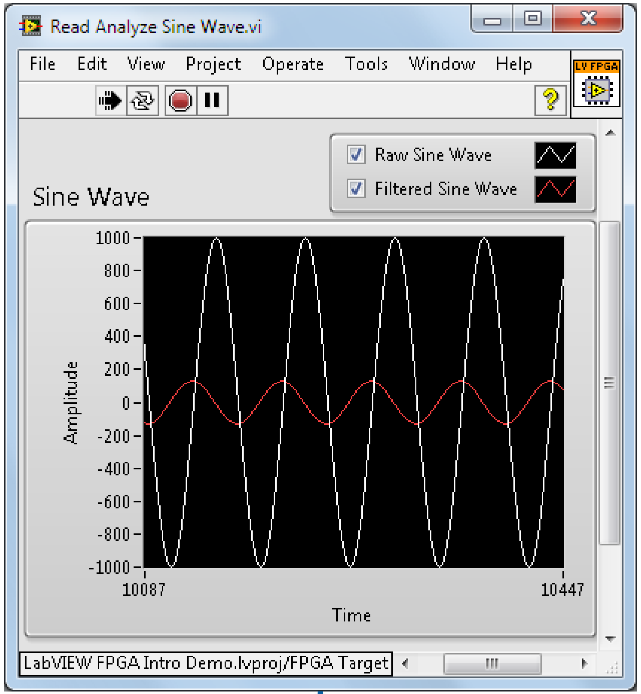
FPGA IO.lvproj * - Project Explorer

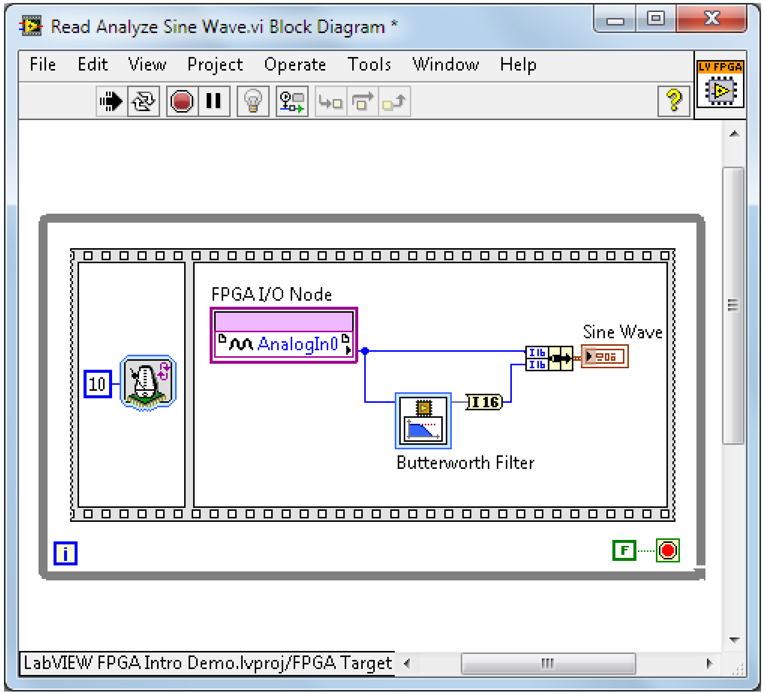
Project: FPGA IO.lvproj
- My Computer
  - Dependencies
  - Build Specifications
- RT CompactRIO Target (10.2.228.121)
  - Chassis (cRIO-9118)
    - FPGA Target (RIO0, cRIO-9118)
      - Mod1
        - Mod1/AI0
        - Mod1/AI1
        - Mod1/AI2
        - Mod1/AI3
      - 40 MHz Onboard Clock
      - FPGA IO.vi
      - Mod1 (Slot 1, NI 9215)
      - Dependencies
      - Build Specifications
  - Dependencies
  - Build Specifications

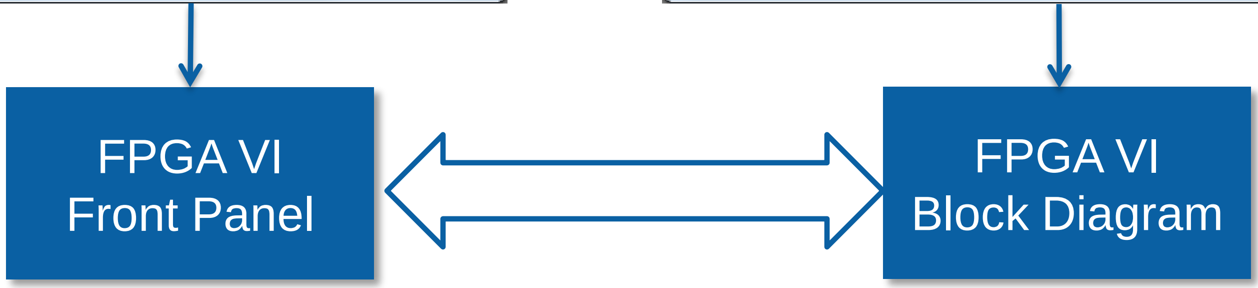
FPGA IO.vi Block Diagram on FPGA IO.lvproj/FPGA Ta...

FPGA IO.lvproj/FPGA Target

Interactive Front Panel Communication


Read Analyze Sine Wave.vi
File Edit View Project Operate Tools Window Help
LV FPGA
☑ Raw Sine Wave
☑ Filtered Sine Wave
Sine Wave
Amplitude
1000
800
600
400
200
0
-200
-400
-600
-800
-1000
10087
10447
Time
LabVIEW FPGA Intro Demo.lvproj/FPGA Target


Read Analyze Sine Wave.vi Block Diagram *
File Edit View Project Operate Tools Window Help
LV FPGA
FPGA I/O Node
AnalogIn0
Sine Wave
10
I16
Butterworth Filter
i
F
LabVIEW FPGA Intro Demo.lvproj/FPGA Target

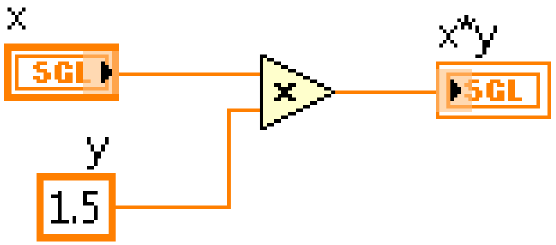


Host Computer

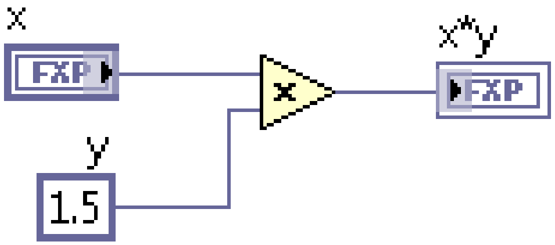
FPGA Target

LabVIEW FPGA Datatypes

- The fixed-point datatype is very efficient for hardware applications (DSPs, FPGAs, etc.)
- Uses less hardware resources than floating-point
- Single precision floating-point datatype is available and recommended for certain use cases



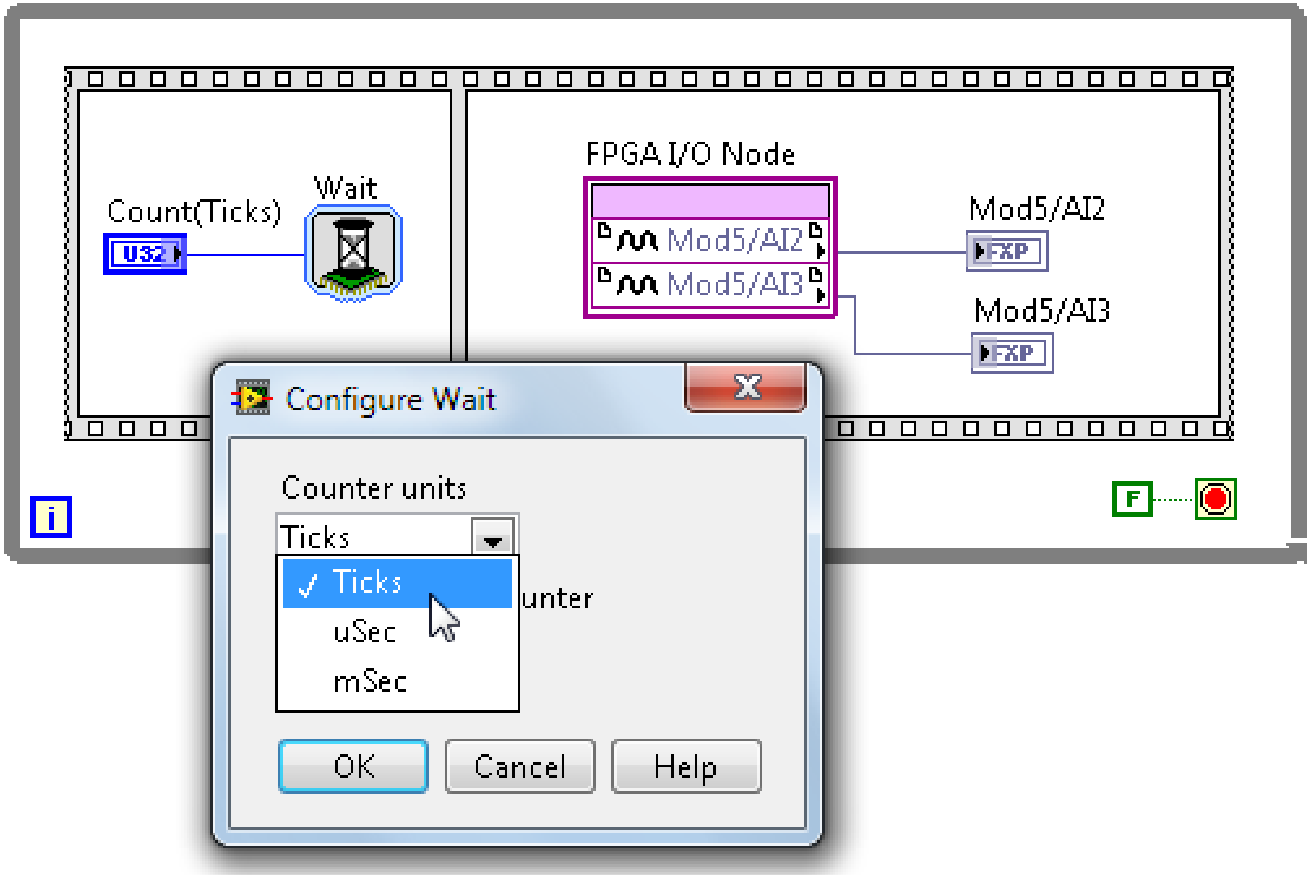
Single-Precision Floating Point



Fixed-Point

Abstraction of Timing

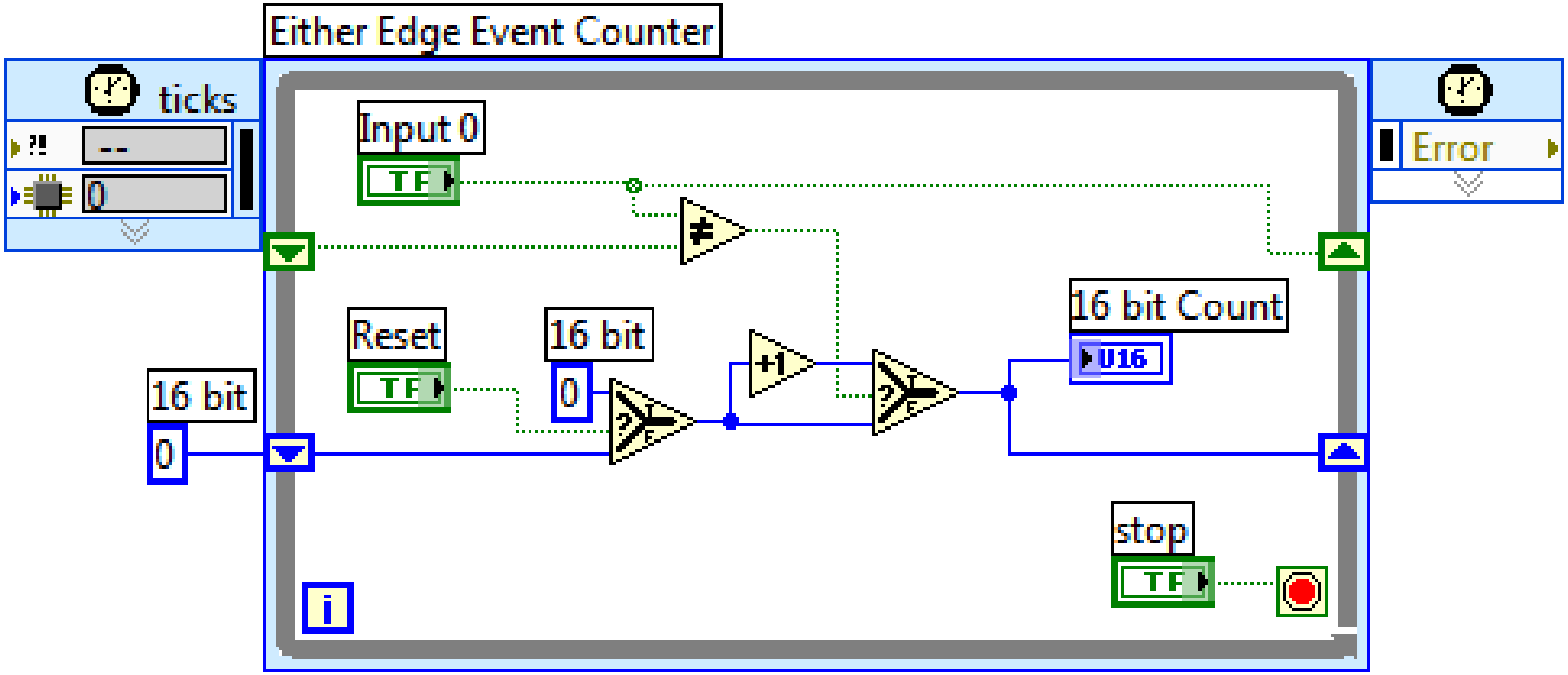
- Loops can execute on the order of ticks of the 40 MHz clock (nanoseconds), microseconds, and/or milliseconds



Counter units: Ticks

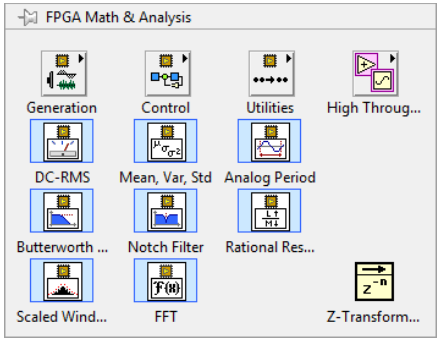
Single-Cycle Timed Loop

- Executes code within 1 cycle of the FPGA clock
- Can be used to optimize the performance of your code

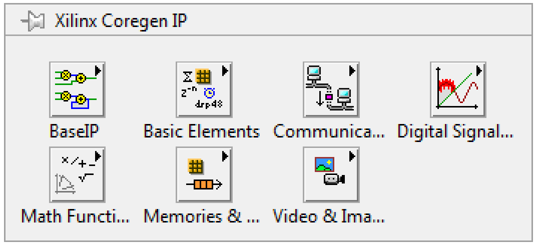


LabVIEW FPGA Functions and IP

In Product



XILINX®



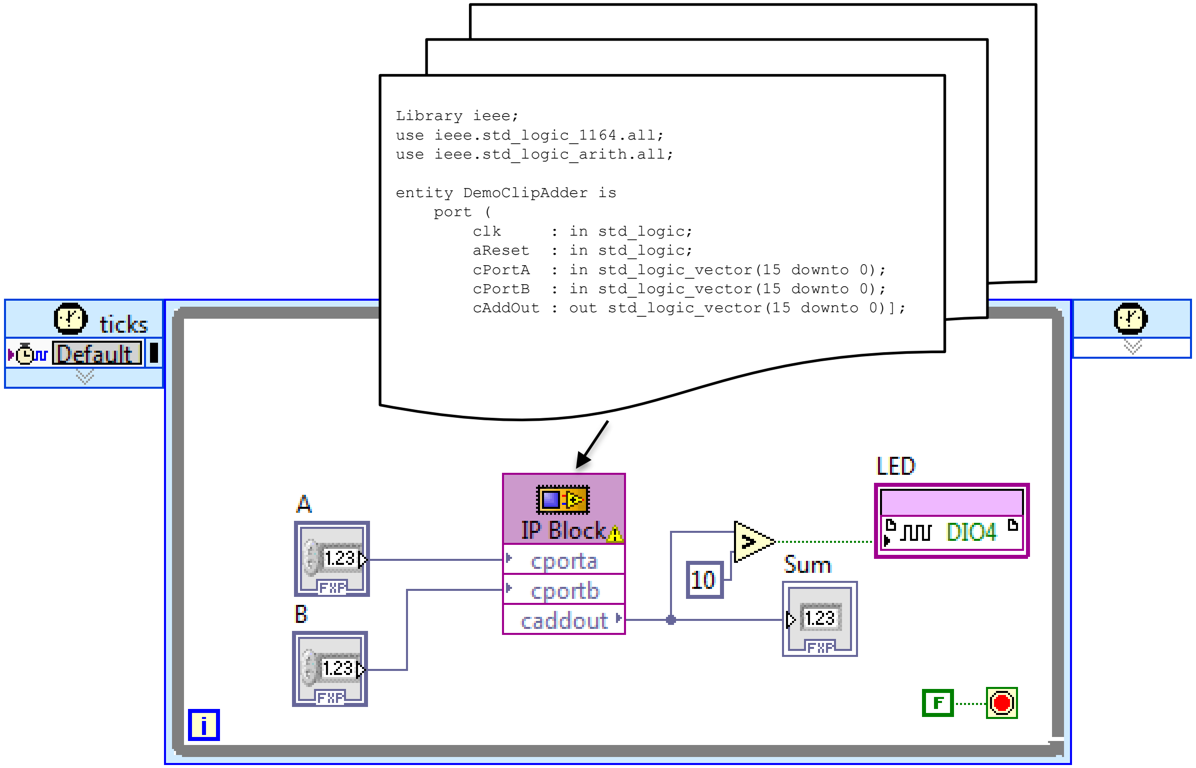
Online

ni.com/IPnet

- Filters (Butterworth, Notch, DC-RMS, etc.)
- PID control
- Control of brushless DC motors
- Digital buses and protocols (SPI, I2C, UART, etc.)
- Image Processing
- RF communications
- Linear and nonlinear systems
- PWM
- Encryption
- Data manipulation
- Device drivers (LCD display, IR sensors, etc.)
- Video processing
- Basic elements (counters, accumulators, etc.)
- Signal generation
- High-throughput math
- Transforms
- Trig functions
- Digital signal processing
- ….**and more**

Reuse of Existing HDL Algorithms

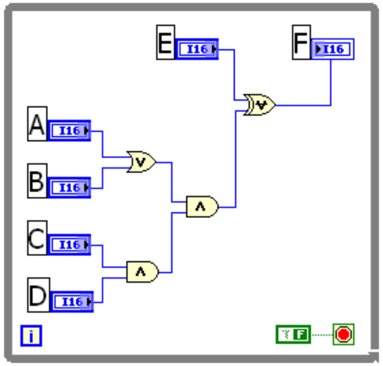
- Increase application development efficiency and leverage existing team expertise
- Similar to calling a DLL in LabVIEW for the desktop



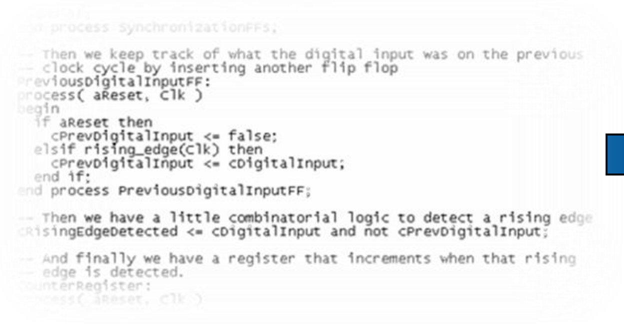
```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity DemoClipAdder is
    port (
        clk     : in std_logic;
        aReset  : in std_logic;
        cPortA  : in std_logic_vector(15 downto 0);
        cPortB  : in std_logic_vector(15 downto 0);
        cAddOut : out std_logic_vector(15 downto 0)];
```

Compilation Process

LabVIEW FPGA Code



Compile VHDL through Xilinx

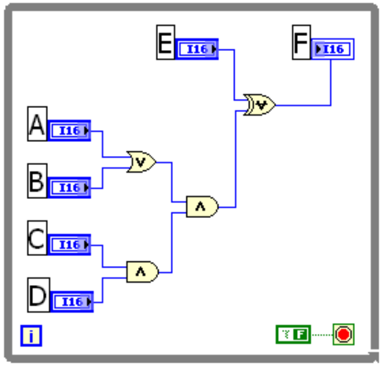


```
end process SynchronizationFFs;

-- Then we keep track of what the digital input was on the previous
-- clock cycle by inserting another flip flop
PreviousDigitalInputFF:
process( aReset, Clk )
begin
  if aReset then
    cPrevDigitalInput <= false;
  elsif rising_edge(Clk) then
    cPrevDigitalInput <= cDigitalInput;
  end if;
end process PreviousDigitalInputFF;

-- Then we have a little combinatorial logic to detect a rising edge
cRisingEdgeDetected <= cDigitalInput and not cPrevDigitalInput;

-- And finally we have a register that increments when that rising
-- edge is detected.
CounterRegister:
process( aReset, Clk )
```

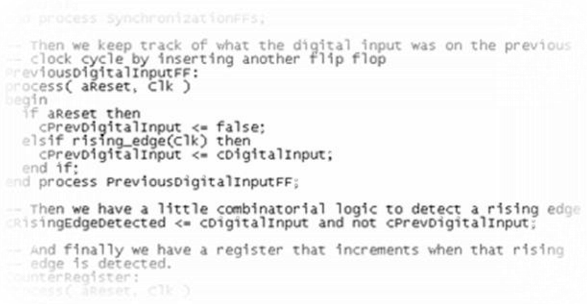
FPGA Logic Implementation

Compilation Process

LabVIEW FPGA Code

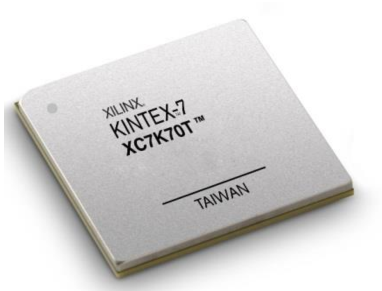


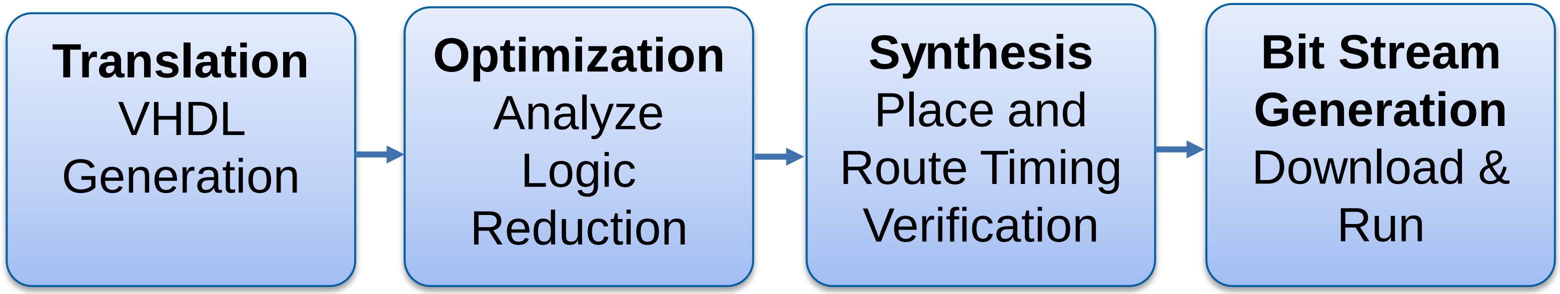
Compile VHDL through Xilinx



```
end process SynchronizationFFs;

-- Then we keep track of what the digital input was on the previous
-- clock cycle by inserting another flip flop
PreviousDigitalInputFF:
process( aReset, Clk )
begin
  if aReset then
    cPrevDigitalInput <= false;
  elsif rising_edge(Clk) then
    cPrevDigitalInput <= cDigitalInput;
  end if;
end process PreviousDigitalInputFF;

-- Then we have a little combinatorial logic to detect a rising edge
cRisingEdgeDetected <= cDigitalInput and not cPrevDigitalInput;

-- And finally we have a register that increments when that rising
-- edge is detected.
CounterRegister:
process( aReset, Clk )
```

FPGA Logic Implementation





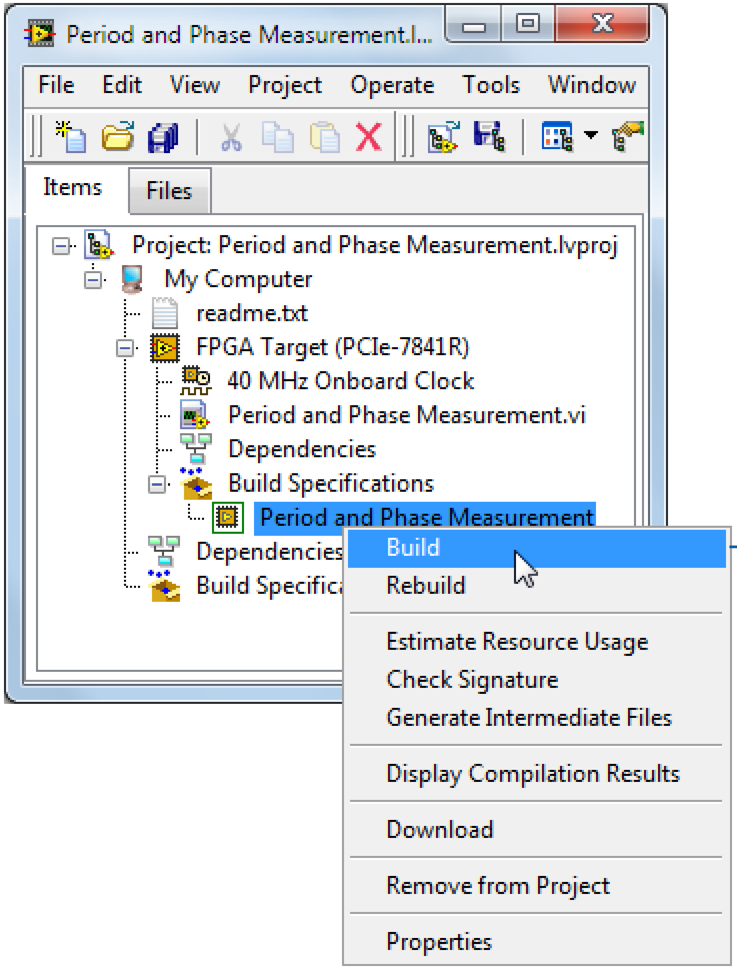
Translation VHDL Generation

Optimization Analyze Logic Reduction

Synthesis Place and Route Timing Verification

Bit Stream Generation Download & Run

One-Click Deployment and Compilation


Period and Phase Measurement.l...
File Edit View Project Operate Tools Window
Items
Files
Project: Period and Phase Measurement.lvproj
My Computer
readme.txt
FPGA Target (PCIe-7841R)
40 MHz Onboard Clock
Period and Phase Measurement.vi
Dependencies
Build Specifications
Period and Phase Measurement
Dependencies
Build Specific
Build
Rebuild
Estimate Resource Usage
Check Signature
Generate Intermediate Files
Display Compilation Results
Download
Remove from Project
Properties

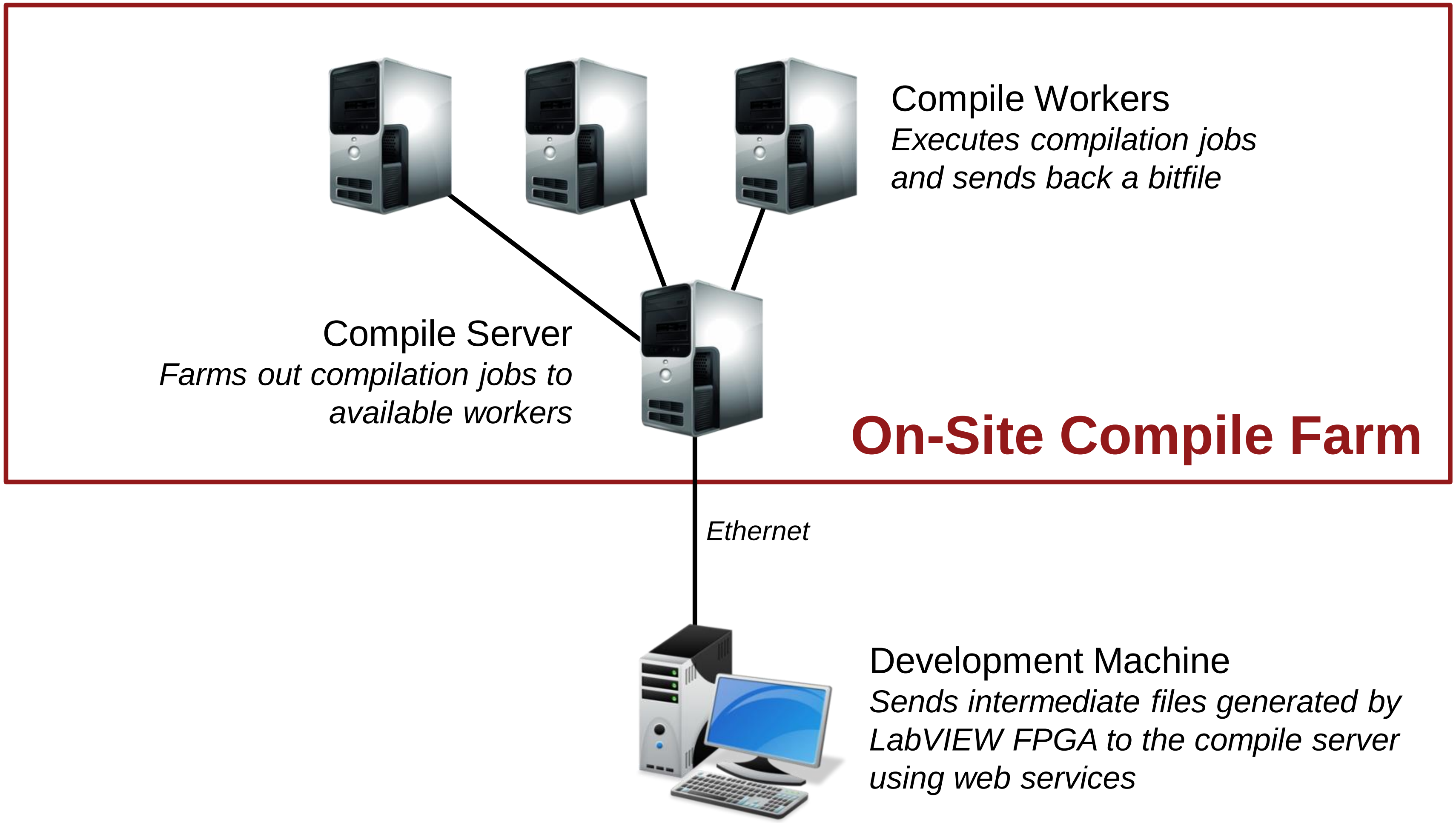
Development PC

Compile Server and Workers

High-Performance Cloud

LabVIEW FPGA Compile System

LabVIEW FPGA Compile Farm Toolkit



Compile Workers

Executes compilation jobs and sends back a bitfile

Compile Server

Farms out compilation jobs to available workers

On-Site Compile Farm

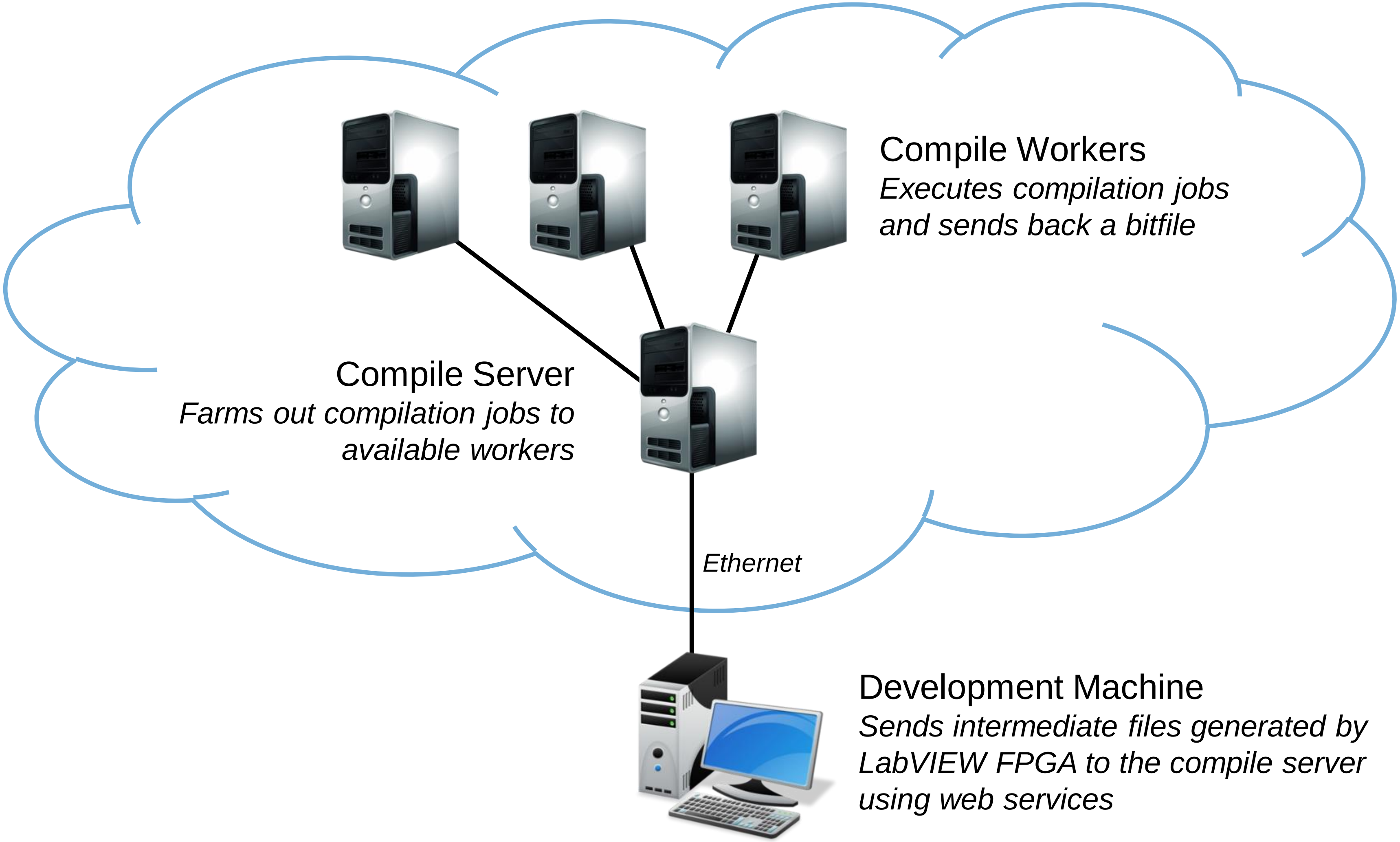
Ethernet

Development Machine

Sends intermediate files generated by LabVIEW FPGA to the compile server using web services

LabVIEW FPGA Compile System

LabVIEW FPGA Compile Cloud Service



Compile Workers

Executes compilation jobs and sends back a bitfile

Compile Server

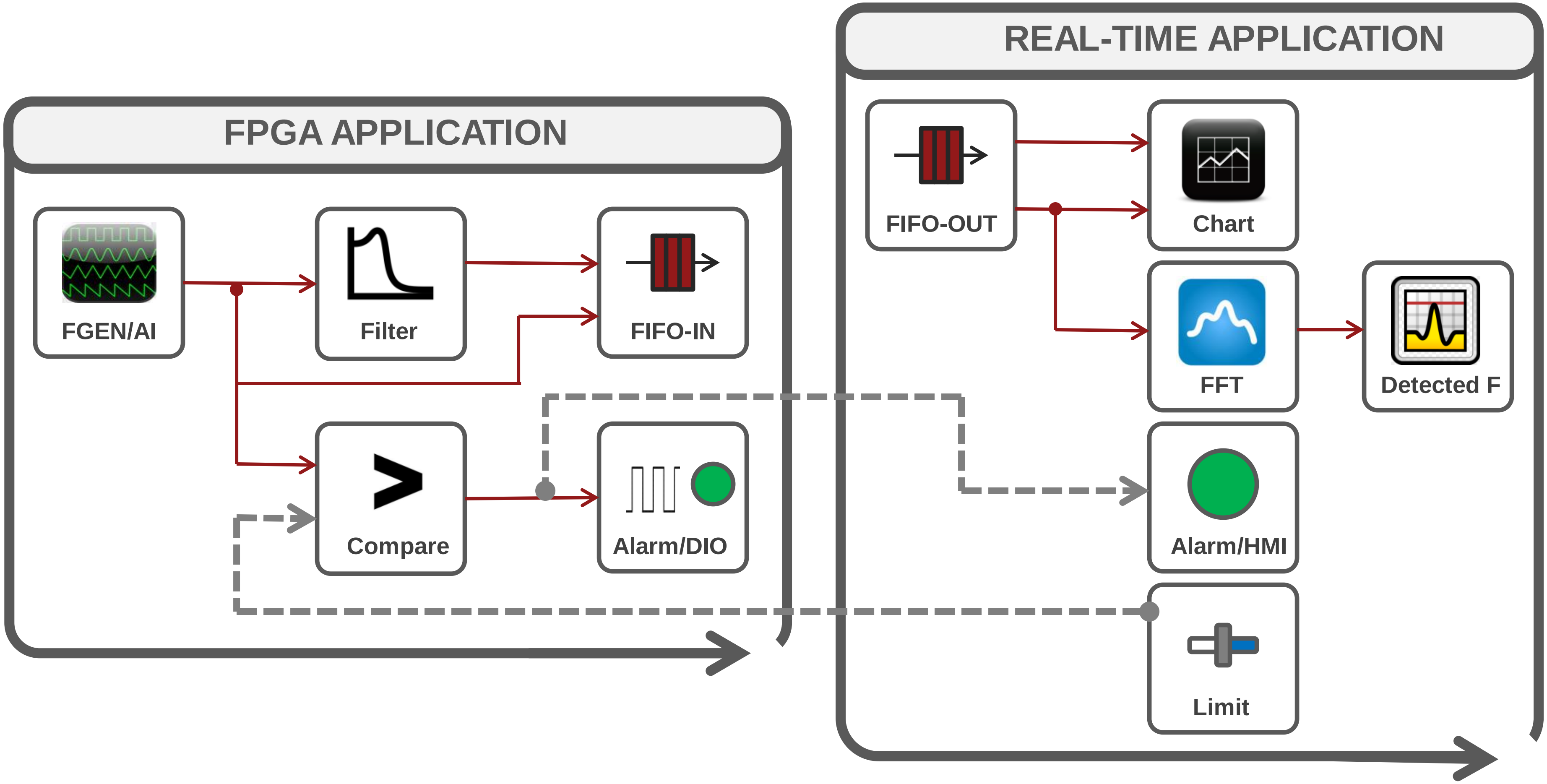
Farms out compilation jobs to available workers

Development Machine

Sends intermediate files generated by LabVIEW FPGA to the compile server using web services

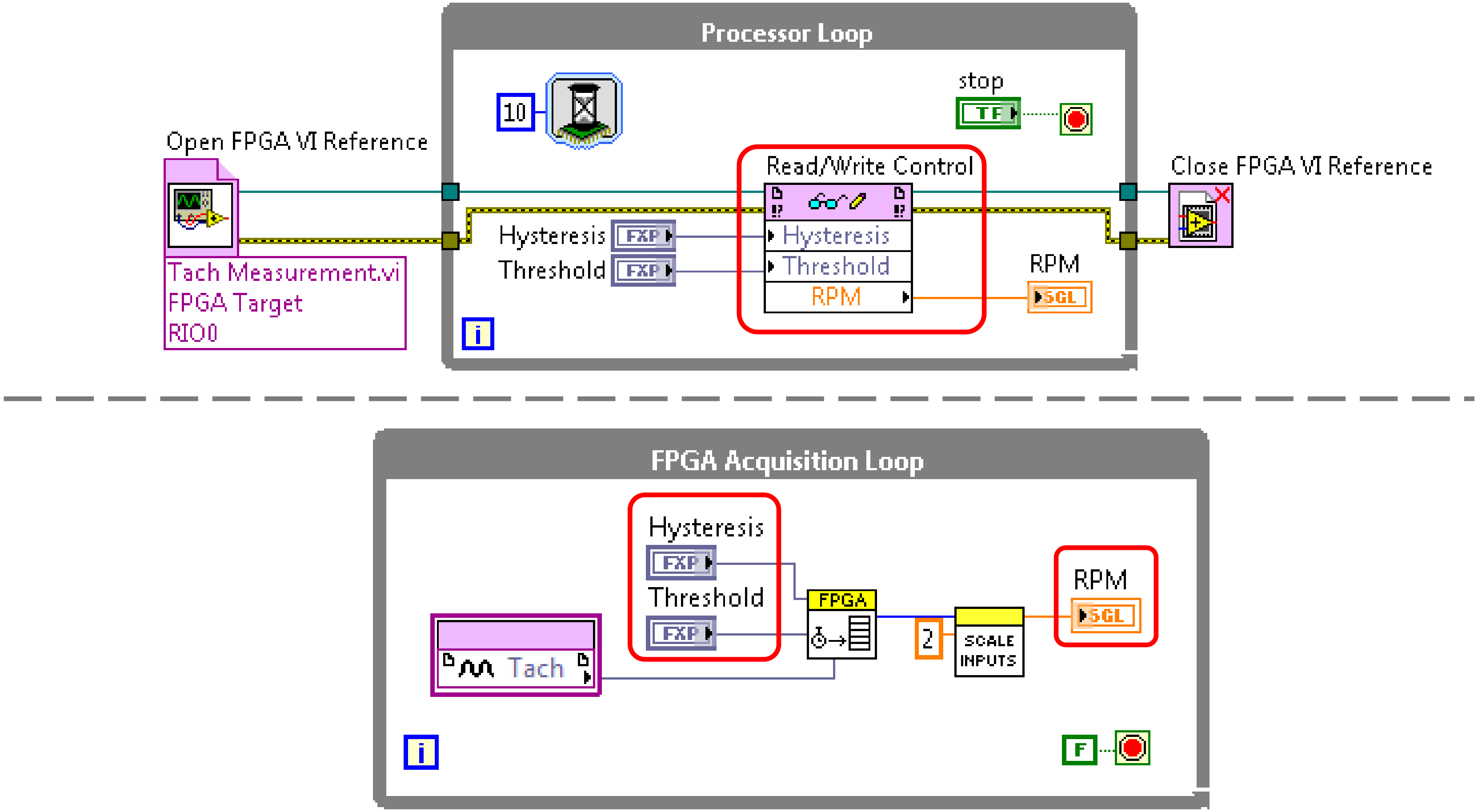
DEMO

CREATING A TESTBENCH



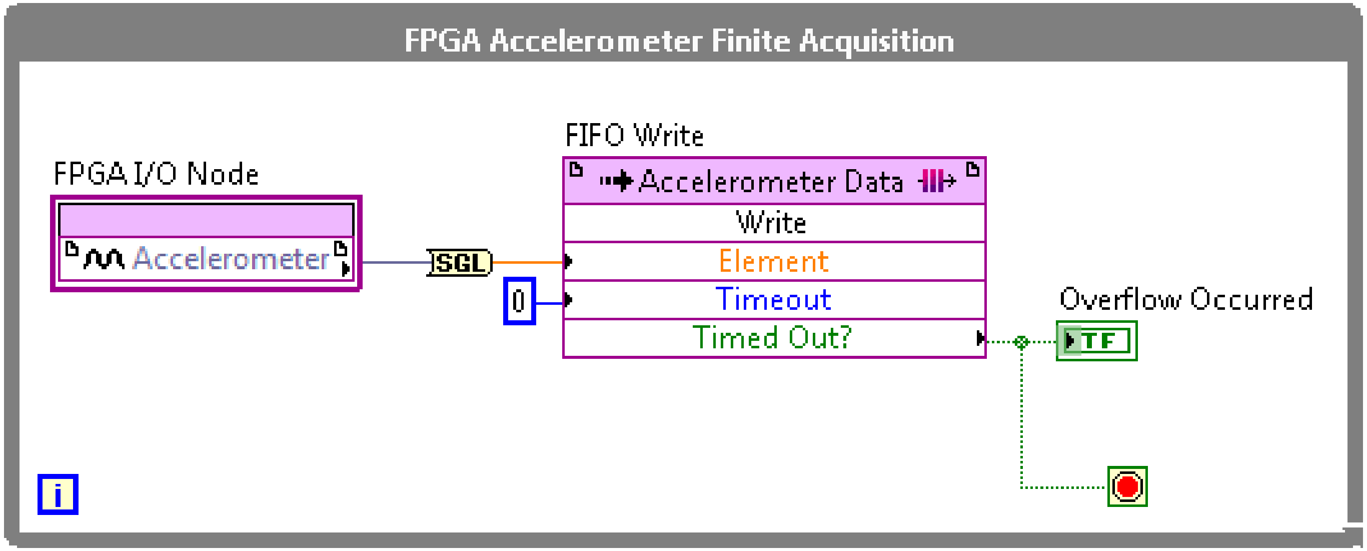
Host Synchronization

- The Read/Write Controls method can be used for communicating current value data



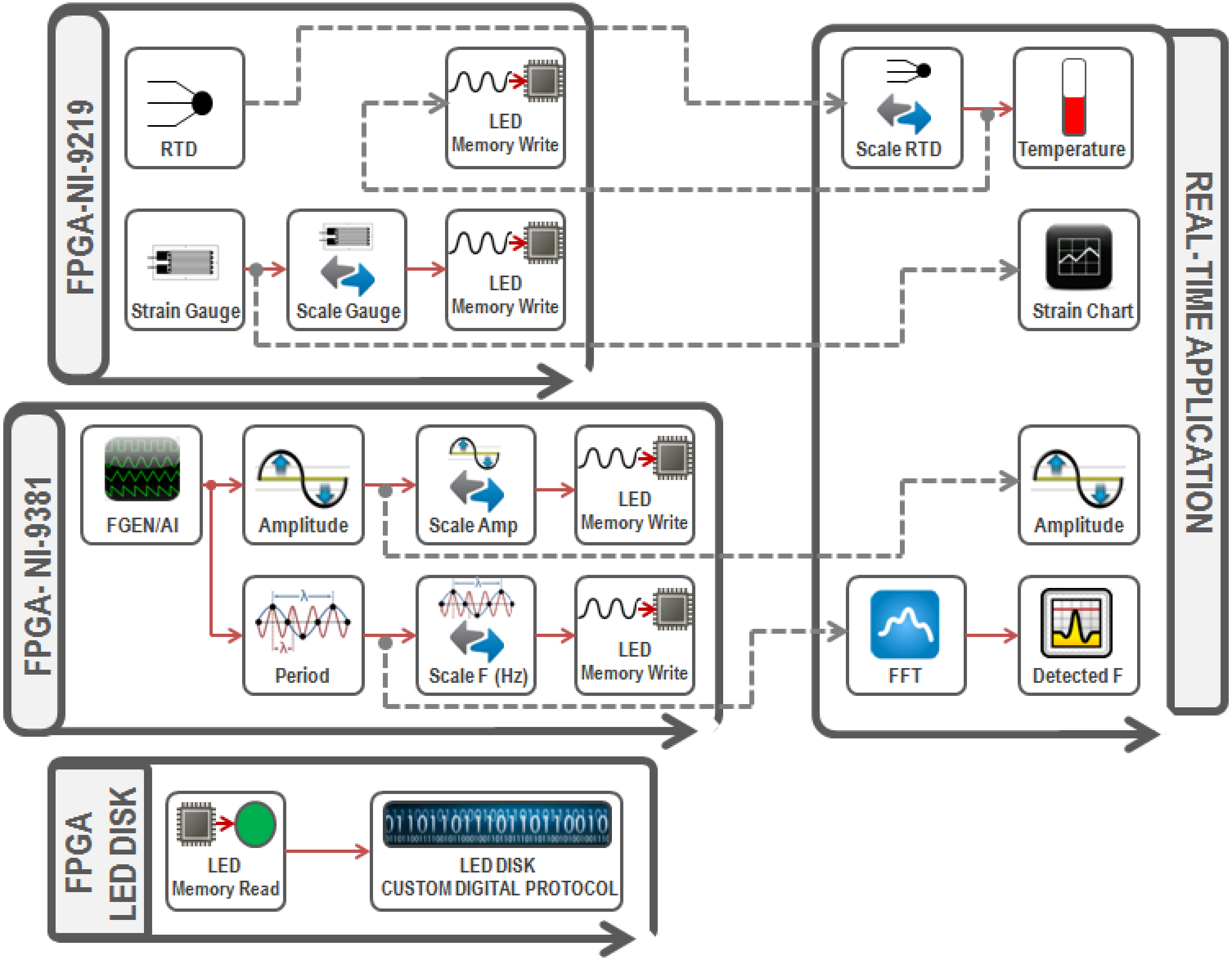
Host Synchronization

- Direct Memory Access (DMA) FIFOs are an efficient mechanism for streaming data from the FPGA to the host processor
- Does not involve processor resources



DEMO

HOST SYNCHRONIZATION



Unrivaled Integration with the Latest Technology

Software Designed Oscilloscope

System on a Module (SoM)

LabVIEW™ 2014

New FPGA Hardware Targets

Performance CompactRIO

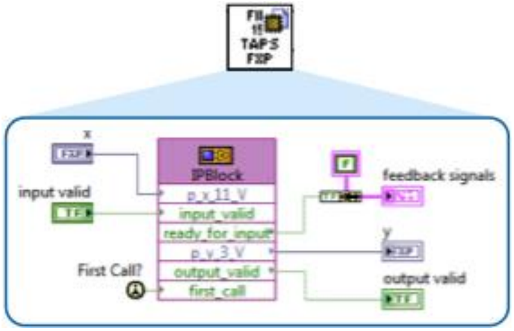
USB3 CVS

Be More Productive with LabVIEW FPGA 2014

Design Faster

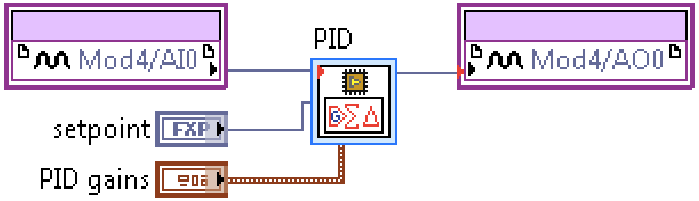
Design High-Performance Algorithms

Use **LabVIEW FPGA IP Builder** to design optimized, high-performance algorithms using high-level programming constructs



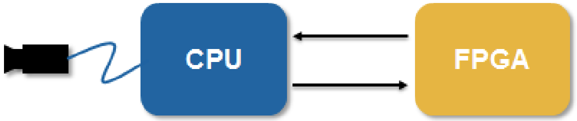
Design PID Controllers

Use the **PID Control VI** to quickly prototype high-speed or high-determinism control algorithms



Design Image Processing Applications

Offload over **50 image processing functions** to the FPGA for maximum performance with the NI Vision Development Module 2014

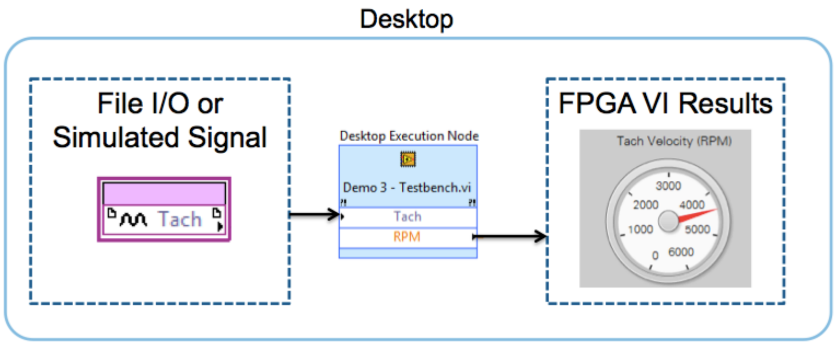


Be More Productive with LabVIEW FPGA 2014

Verify Faster

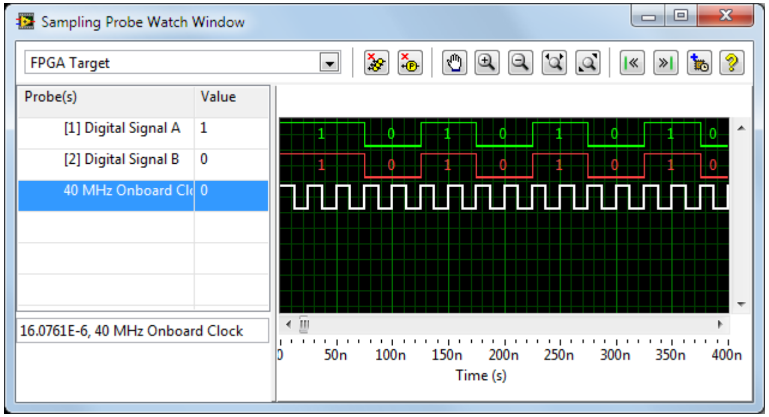
Verify Code using Simulated I/O

Use the **Desktop Execution Node** to verify code by developing test benches using simulated or file generated I/O



Verify Signal Timing with Waveform Probe

Use the **Digital Waveform Probe** to probe your signals relative to one another and view history



Probe(s)	Value
[1] Digital Signal A	1
[2] Digital Signal B	0
40 MHz Onboard Clo	0

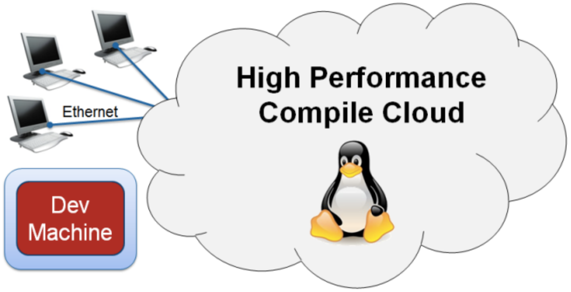
16.0761E-6, 40 MHz Onboard Clock

Be More Productive with LabVIEW FPGA 2014

Compile Faster

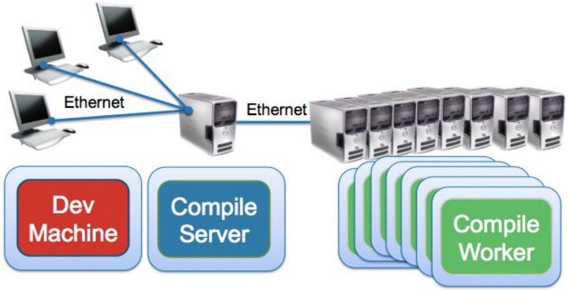
Send Your Compiles to the Cloud

Use the **LabVIEW FPGA Compile Cloud Service** (free with SSP) to reduce your compile times up to 60%



Manage FPGA Compilations On-Site

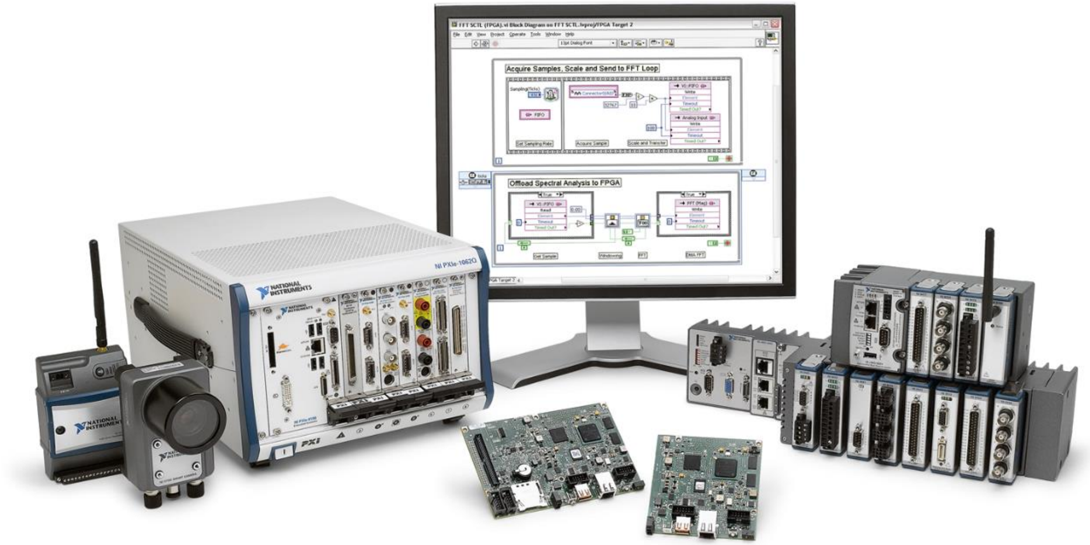
Use the **LabVIEW FPGA Compile Farm Toolkit** to create an on-site server to manage FPGA compilations



Increase Compilation Performance with Vivado

Use **Xilinx Vivado** included with LabVIEW FPGA 2014 to compile faster and more reliably for Kintex-7 FPGAs and Zynq SoCs

The Benefits of a Platform Based Approach


Acquire Samples, Scale and Send to FFT Loop
Offload Spectral Analysis to FPGA
NI PXIe-1062Q
NATIONAL INSTRUMENTS
PXI

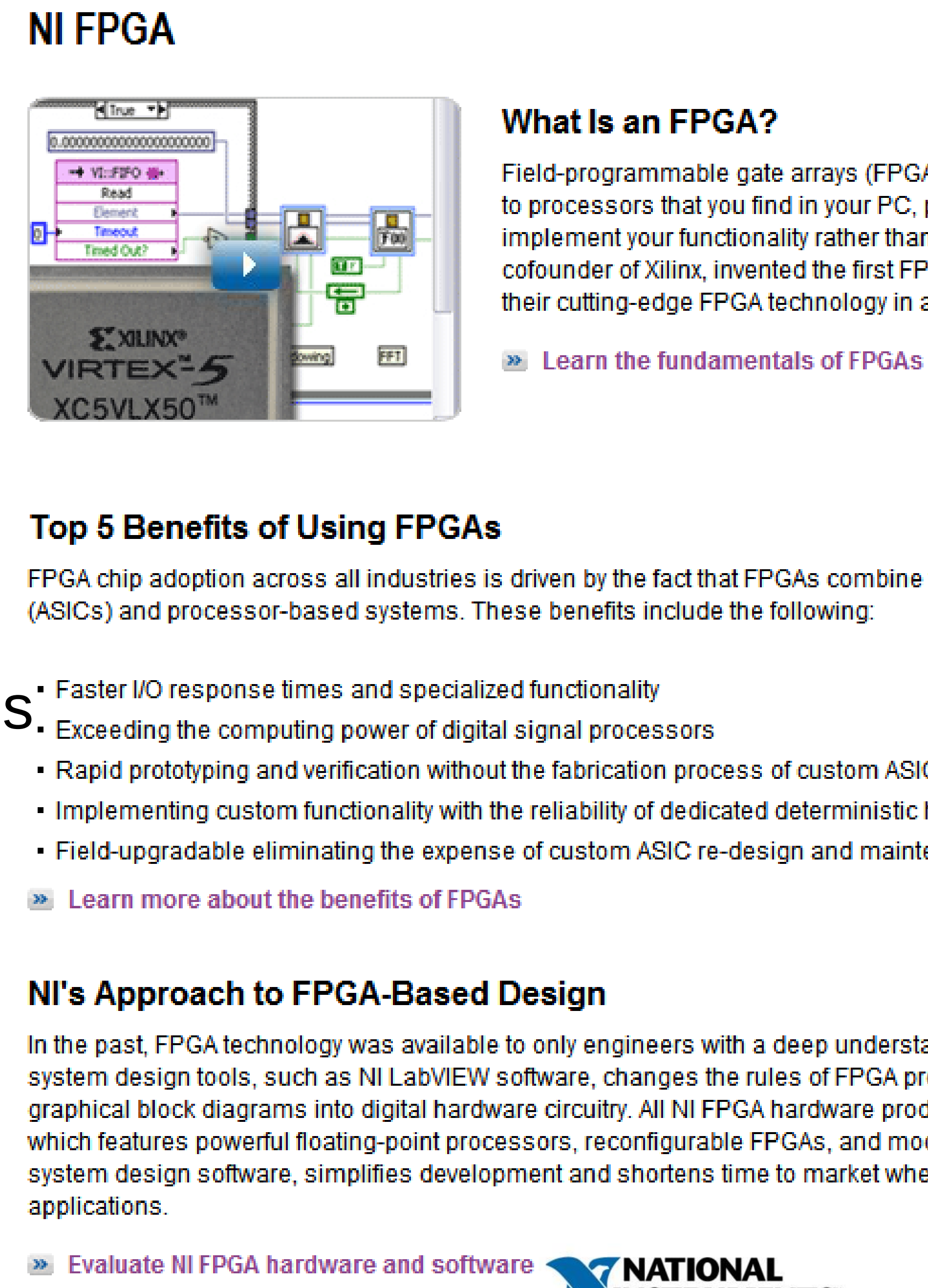
High-Level Software

Flexible Hardware

Integrated Hardware and Software Platform

ni.com/fpga

- FPGA Fundamentals
- Benefits of FPGAs
- NI FPGA-based Case Studies
- Learn more about the tools
  - LabVIEW FPGA
  - FPGA-based RIO hardware



NI FPGA

What Is an FPGA?

Field-programmable gate arrays (FPGA
to processors that you find in your PC, p
implement your functionality rather than
cofounder of Xilinx, invented the first FP
their cutting-edge FPGA technology in a

» Learn the fundamentals of FPGAs

Top 5 Benefits of Using FPGAs

FPGA chip adoption across all industries is driven by the fact that FPGAs combine
(ASICs) and processor-based systems. These benefits include the following:

- Faster I/O response times and specialized functionality
- Exceeding the computing power of digital signal processors
- Rapid prototyping and verification without the fabrication process of custom ASIC
- Implementing custom functionality with the reliability of dedicated deterministic h
- Field-upgradable eliminating the expense of custom ASIC re-design and mainte

» Learn more about the benefits of FPGAs

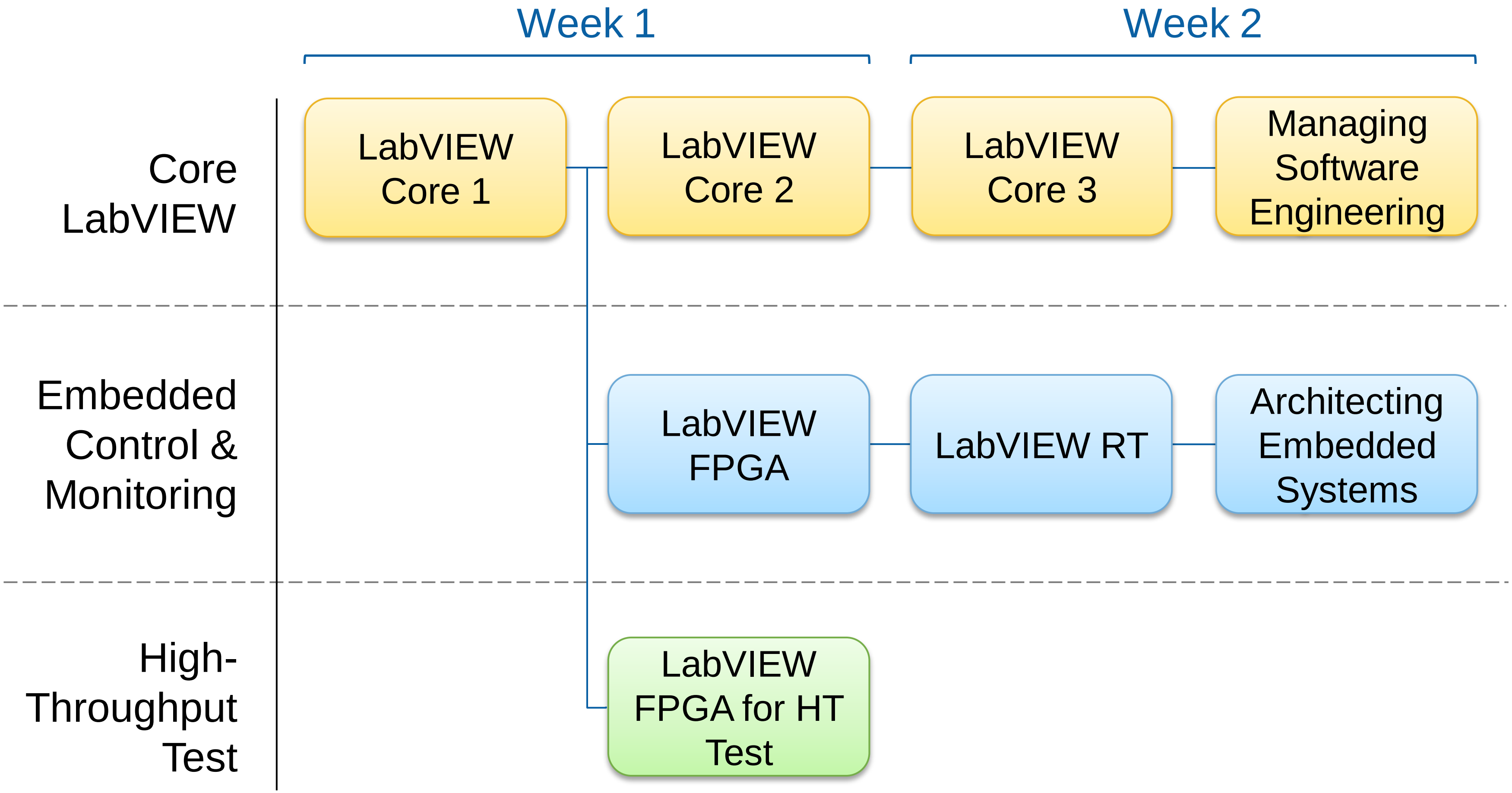
NI's Approach to FPGA-Based Design

In the past, FPGA technology was available to only engineers with a deep understa
system design tools, such as NI LabVIEW software, changes the rules of FPGA pro
graphical block diagrams into digital hardware circuitry. All NI FPGA hardware prod
which features powerful floating-point processors, reconfigurable FPGAs, and mod
system design software, simplifies development and shortens time to market whe
applications.

» Evaluate NI FPGA hardware and software

NI Instructor Led Training

Classroom, Virtual, or Online



Week 1

Week 2

Core LabVIEW

LabVIEW Core 1

LabVIEW Core 2

LabVIEW Core 3

Managing Software Engineering

Embedded Control & Monitoring

LabVIEW FPGA

LabVIEW RT

Architecting Embedded Systems

High-Throughput Test

LabVIEW FPGA for HT Test

LabVIEW for CompactRIO Developer’s Guide

- Best practices for designing embedded control and monitoring systems with LabVIEW
- Recommended architectures and frameworks
- Downloadable example code throughout

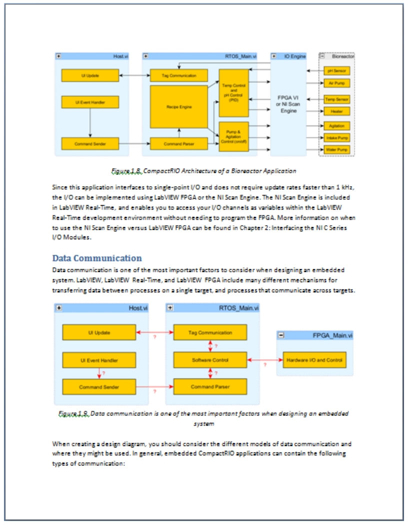


Figure 1.8. CompactRIO Architecture of a Bioreactor Application

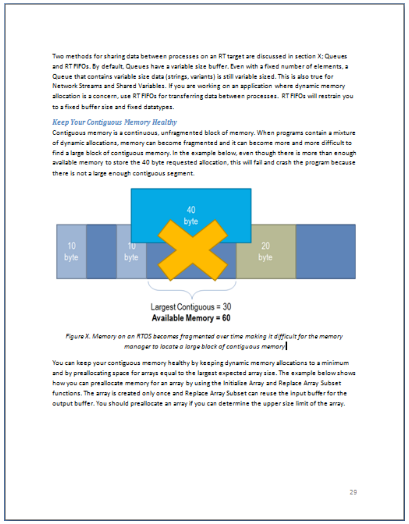
Since this application interfaces to single-point I/O and does not require update rates faster than 1 kHz, the I/O can be implemented using LabVIEW FPGA or the NI Scan Engine. The NI Scan Engine is included in LabVIEW Real-Time, and enables you to access your I/O channels as variables within the LabVIEW Real-Time development environment without needing to program the FPGA. More information on when to use the NI Scan Engine versus LabVIEW FPGA can be found in Chapter 2: Interfacing the NI C Series I/O Modules.

Data Communication

Data communication is one of the most important factors to consider when designing an embedded system. LabVIEW, LabVIEW Real-Time, and LabVIEW FPGA include many different mechanisms for transferring data between processes on a single target, and processes that communicate across targets.

Figure 1.9. Data communication is one of the most important factors when designing an embedded system

When creating a design diagram, you should consider the different models of data communication and where they might be used. In general, embedded CompactRIO applications can contain the following types of communication:



Two methods for sharing data between processes on an RT target are discussed in section X; Queues and RT FIFOs. By default, Queues have a variable size buffer. Even with a fixed number of elements, a Queue that contains variable size data (strings, variants) is still variable sized. This is also true for Network Streams and Shared Variables. If you are working on an application where dynamic memory allocation is a concern, use RT FIFOs for transferring data between processes. RT FIFOs will restrain you to a fixed buffer size and fixed datatypes.

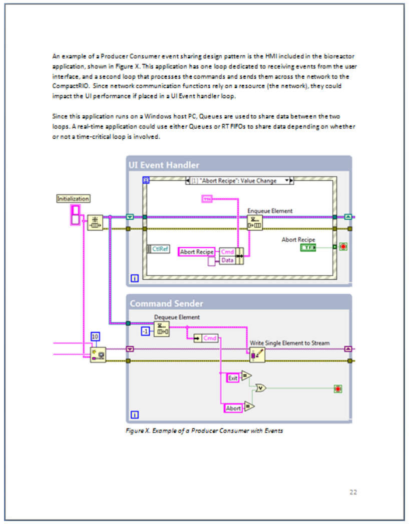
Keep Your Contiguous Memory Healthy

Contiguous memory is a continuous, unfragmented block of memory. When programs contain a mixture of dynamic allocations, memory can become fragmented and it can become more and more difficult to find a large block of contiguous memory. In the example below, even though there is more than enough available memory to store the 40 byte requested allocation, this will fail and crash the program because there is not a large enough contiguous segment.

Figure X. Memory on an RTOS becomes fragmented over time making it difficult for the memory manager to locate a large block of contiguous memory

You can keep your contiguous memory healthy by keeping dynamic memory allocations to a minimum and by preallocating space for arrays equal to the largest expected array size. The example below shows how you can preallocate memory for an array by using the Initialize Array and Replace Array Subset functions. The array is created only once and Replace Array Subset can reuse the input buffer for the output buffer. You should preallocate an array if you can determine the upper size limit of the array.

29

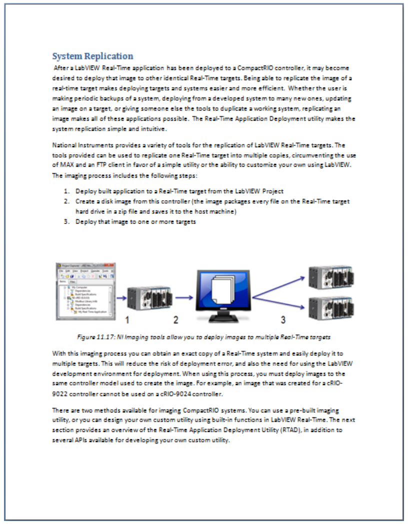


An example of a Producer Consumer event sharing design pattern is the HMI included in the bioreactor application, shown in Figure X. This application has one loop dedicated to receiving events from the user interface, and a second loop that processes the commands and sends them across the network to the CompactRIO. Since network communication functions rely on a resource (the network), they could impact the UI performance if placed in a UI Event handler loop.

Since this application runs on a Windows host PC, Queues are used to share data between the two loops. A real-time application could use either Queues or RT FIFOs to share data depending on whether or not a time-critical loop is involved.

Figure X. Example of a Producer Consumer with Events

22



System Replication

After a LabVIEW Real-Time application has been deployed to a CompactRIO controller, it may become desired to deploy that image to other identical Real-Time targets. Being able to replicate the image of a real-time target makes deploying targets and systems easier and more efficient. Whether the user is making periodic backups of a system, deploying from a developed system to many new ones, updating an image on a target, or giving someone else the tools to duplicate a working system, replicating an image makes all of these applications possible. The Real-Time Application Deployment utility makes the system replication simple and intuitive.

National Instruments provides a variety of tools for the replication of LabVIEW Real-Time targets. The tools provided can be used to replicate one Real-Time target into multiple copies, circumventing the use of MAX and an FTP client in favor of a simple utility or the ability to customize your own using LabVIEW. The imaging process includes the following steps:

1. Deploy built application to a Real-Time target from the LabVIEW Project
2. Create a disk image from this controller (the image packages every file on the Real-Time target hard drive in a zip file and saves it to the host machine)
3. Deploy that image to one or more targets

Figure 11.17: NI Imaging tools allow you to deploy images to multiple Real-Time targets

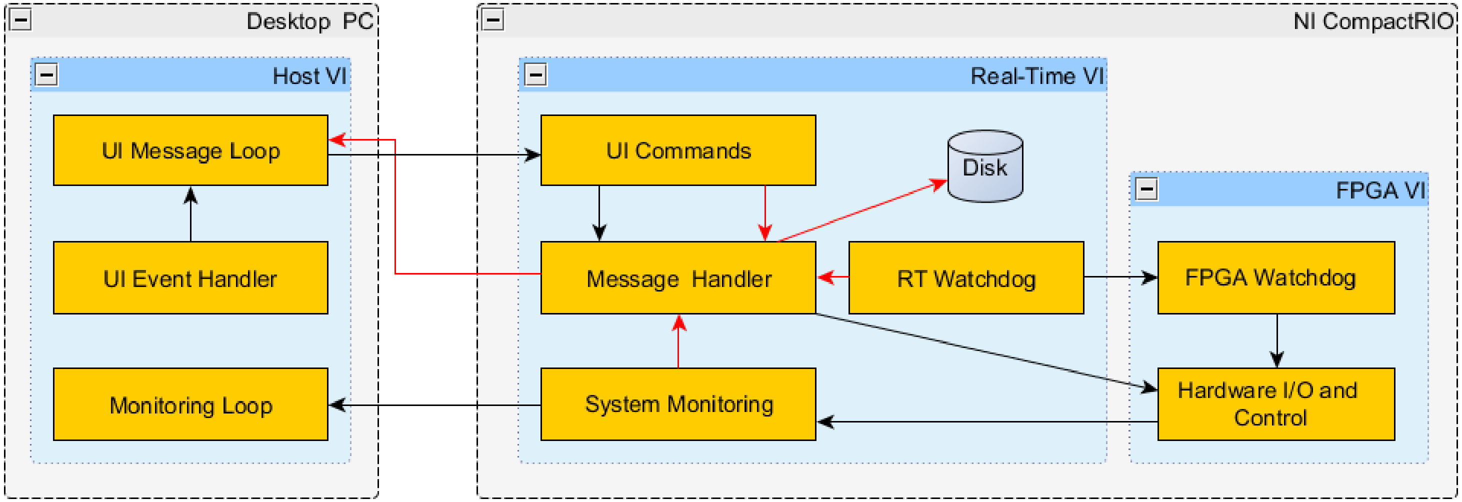
With this imaging process you can obtain an exact copy of a Real-Time system and easily deploy it to multiple targets. This will reduce the risk of deployment error, and also the need for using the LabVIEW development environment for deployment. When using this process, you must deploy images to the same controller model used to create the image. For example, an image that was created for a cRIO-9022 controller cannot be used on a cRIO-9024 controller.

There are two methods available for imaging CompactRIO systems. You can use a pre-built imaging utility, or you can design your own custom utility using built-in functions in LabVIEW Real-Time. The next section provides an overview of the Real-Time Application Deployment Utility (RTAD), in addition to several APIs available for developing your own custom utility.

ni.com/compactriodevguide

LabVIEW for CompactRIO Sample Projects

- Recommended starting points designed to ensure the quality and scalability of a system



- LabVIEW FPGA Control & Monitoring (above)
- LabVIEW FPGA Control with Sequencer Engine
- LabVIEW Real-Time Control & Monitoring
- LabVIEW FPGA Waveform Acquisition and Logging