CompactRIO

Embedded Measurement, Compute, and Control for Engineers

EMERSON

Daniel Eaton

Chief Field Applications Engineer

CLA, CLED, 20 years “doing” w/ NI

Daniel.eaton@emerson.com

We are here to help!

- Pull us in to design conversations
- Let us help de-risk projects
- Tell us what you want to learn about next

cRIO
What is it?

CompactRIO

A rugged, embedded computing platform with modular I/O, programmable FPGA, and a real-time OS all powered by LabVIEW.

Features

- NI Linux Real-Time OS
- Up to 1.91 GHz quad-core processors
- LabVIEW-programmable Xilinx FPGAs
- -40 °C to 70 °C operating temperature range
- 50 g shock and 5 g vibration operating range
- Over 200 modules for electrical signals, sensors, and communication protocols.
- Up to 16 GB of on-board SSD for data logging
- Computer ports including video out, USB, and Ethernet for peripherals and communication
- LabVIEW programmable main processor and FPGA

EMERSON | NI

Controller and Backplane

Controller

- Provides compute infrastructure for running DAQ and control applications
- Comprises typical “motherboard” components (CPU, memory, disk etc.)

Backplane

- Provides infrastructure for connecting compute resources (controller) to IO resources (C Series modules)
- Includes DAQ ASIC and user programmable FPGA

cRIO-904x – Maximize Performance
Up to 1.6 GHz Quad Core CPU and Kintex-7 325T FPGA

TSN-Enabled
Ethernet (x2)

PFI
Trigger

USB Type C
-USB 3.1
-Host/Device
-Display

Isolated Power, SD, USB 2.0, RS-232, RS-485, User Button,
Reset Button, NI Linux Real-Time, 4/8 slots C Series I/O

EMERSON | NI

Basic Measurements
(Full Catalog)

NI 9223

1 MHz, 4 independent ch
+/- 10V

NI 9206

250 kHz, 32 muxed chs
+/- 10V

NI 9220

100 kHz, 16 independent ch
+/- 10V

NI 9208

500 kHz, 16 muxed chs
+/- 20 mA

LabVIEW had Analysis and Control Blocks

- PID
- Fuzzy Control
- Digital Filtering

Basic Generation (Full Catalog)

NI 9263

100 kHz, 4 independent ch
+/- 10V

NI 9264

25 kHz, 16 independent ch
+/- 10V

NI 9262

1 MHz, 6 independent ch
+/- 10V

NI 9265

100 kHz, 4 independent ch
0-20 mA

LabVIEW had Analysis and Control Blocks

- PID
- Fuzzy Control
- Digital Filtering

Modules for Temperature Measurement (Full Catalog)

NI 9213

High-density, most popular TC input module

NI 9219

Universal module with Ch-Ch Isolation

NI 9214

High-density like NI 9213, but more accurate

NI 9210

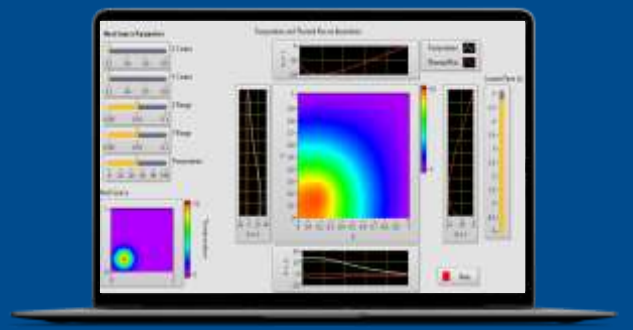
Miniature TC input (mini jack), lowest cost

NI 9212

Ch-Ch isolation, miniature TC connection option, best accuracy (screw terminal option)

LabVIEW Analysis Blocks for:

- Interpolation and Extrapolation
- Fitting
- Probability and Statistics



PDE THERMAL DISTRIBUTION.VI EXAMPLE

Modules for Power Measurement (Full Catalog)

Voltage

NI 9242/44

3-phase inputs (L1,L2,L3,N) for 220/440 VAC networks

NI 9225

300 V_{rms}, ch-ch isolated

Current

NI 9227

Good accuracy for low current (5 Arms) measurements
Minimal over-range

NI 9245/46

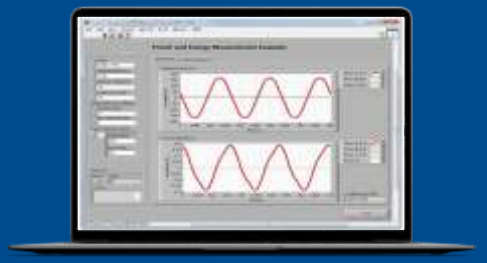
Use for CTs with 5A secondary outputs and large over-range

NI 9238/29

Use with CTs that have a low-voltage output (0.333V or 10V)

Electrical Power Suite for LabVIEW

- Signal processing palette
- Waveform measurements palette
- Dynamic signal filtering



POWER AND ENERGY MEASUREMENT VI
DEVELOPED USING THE ELECTRICAL POWER SUITE

Modules for Sound and Vibration

Full Catalog

NI 9234

Most popular IEPE module

NI 9250

Designed for
NVH applications

NI 9231

High channel-count

NI 9230

Low-cost version of NI 9232

NI 9232

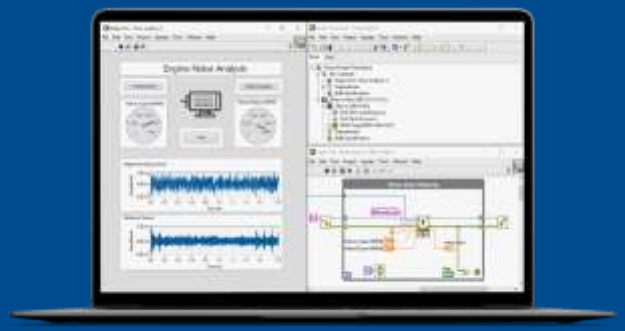
Designed for turbomachinery.
Higher speed with ±30V range.

All sound and vibration modules have:

- 24-bit resolution input circuitry
- Built-in excitation for sensors (IEPE)
- Sample rates between 12ks/s/ch and 102.4 ks/s/ch

Sound and Vibration Toolkit for LabVIEW

- Example Programs with S&V specificc user interface
- Analysis functions for S&V



Modules for Load/Force, Pressure, Strain, Torque (Full Catalog)

NI 9237

4-Channel, 24-bit
50kS/s/ch
½, ¼, Full Bridge

NI 9219

Universal module
with Ch-Ch Isolation

NI 9235/9236

8-Channel, 24-bit
50kS/s/ch
¼ bridge 120/350 Ohm Variants

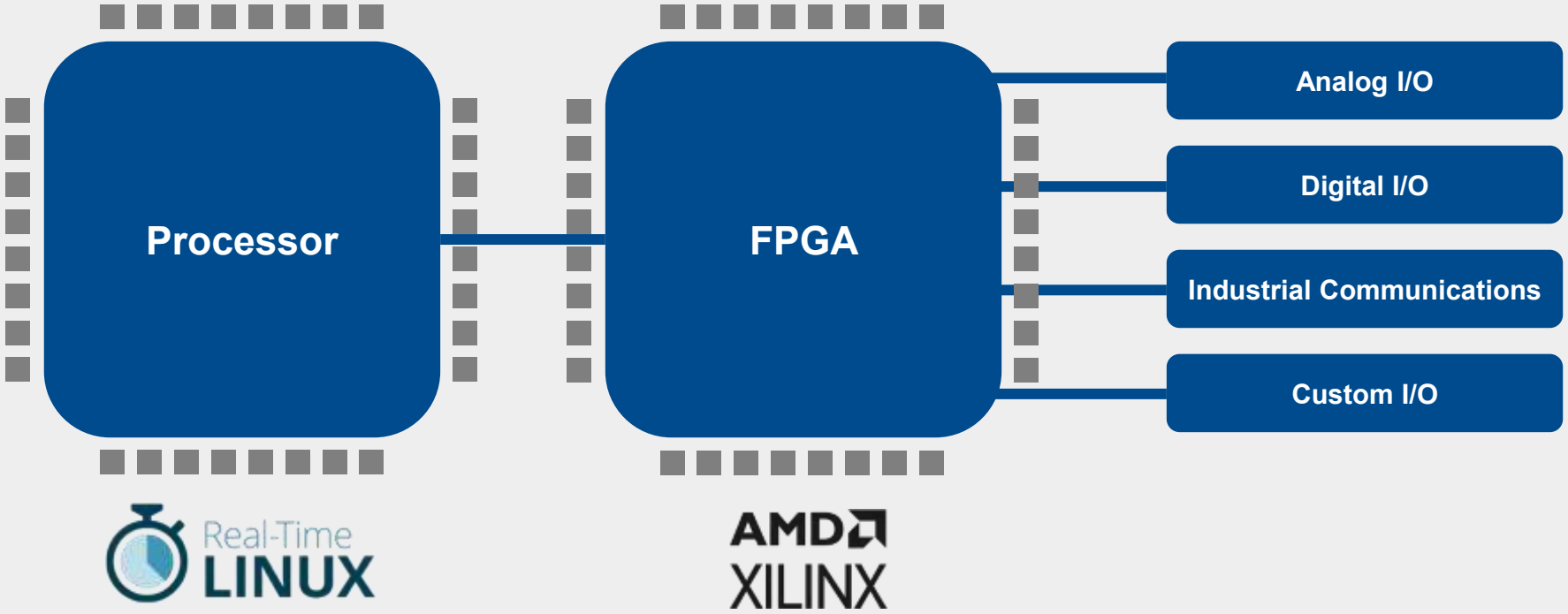
LabVIEW has Analysis Blocks for:

- Interpolation and Extrapolation
- Fitting
- Probability and Statistics

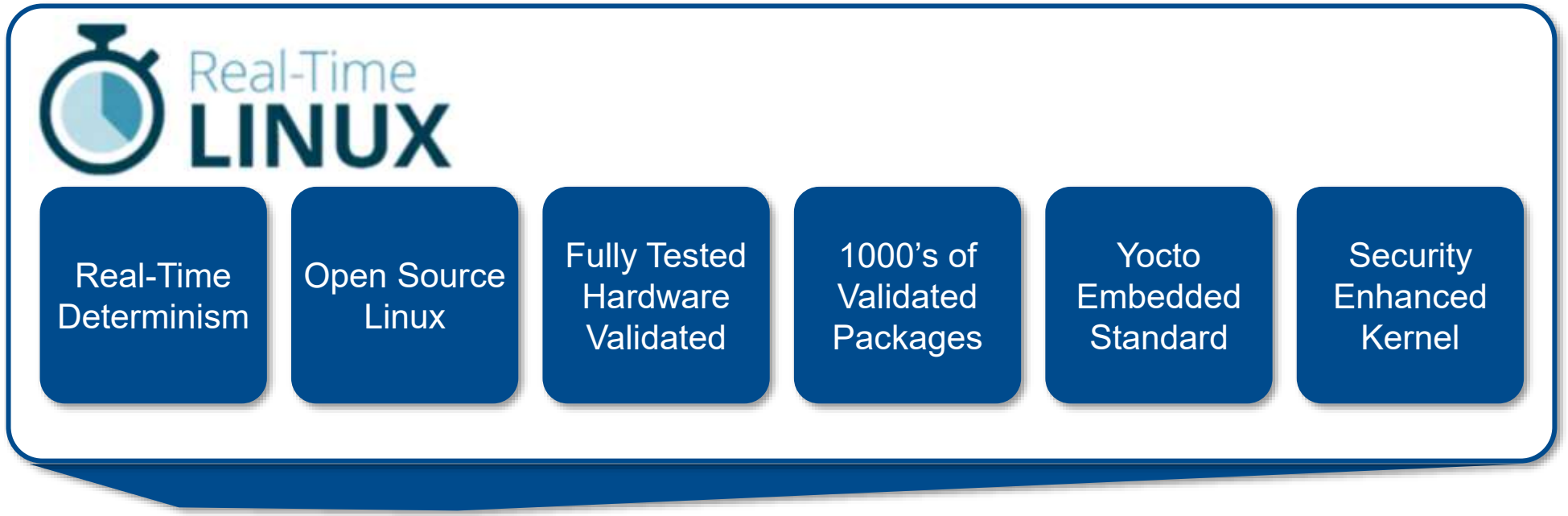
EMERSON

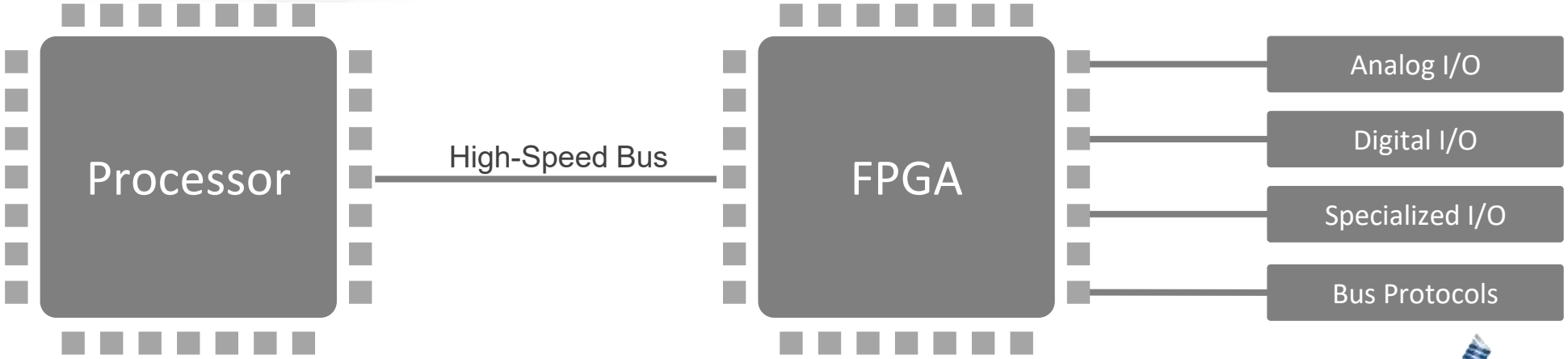
System Architecture

All CompactRIO systems share the same three key components

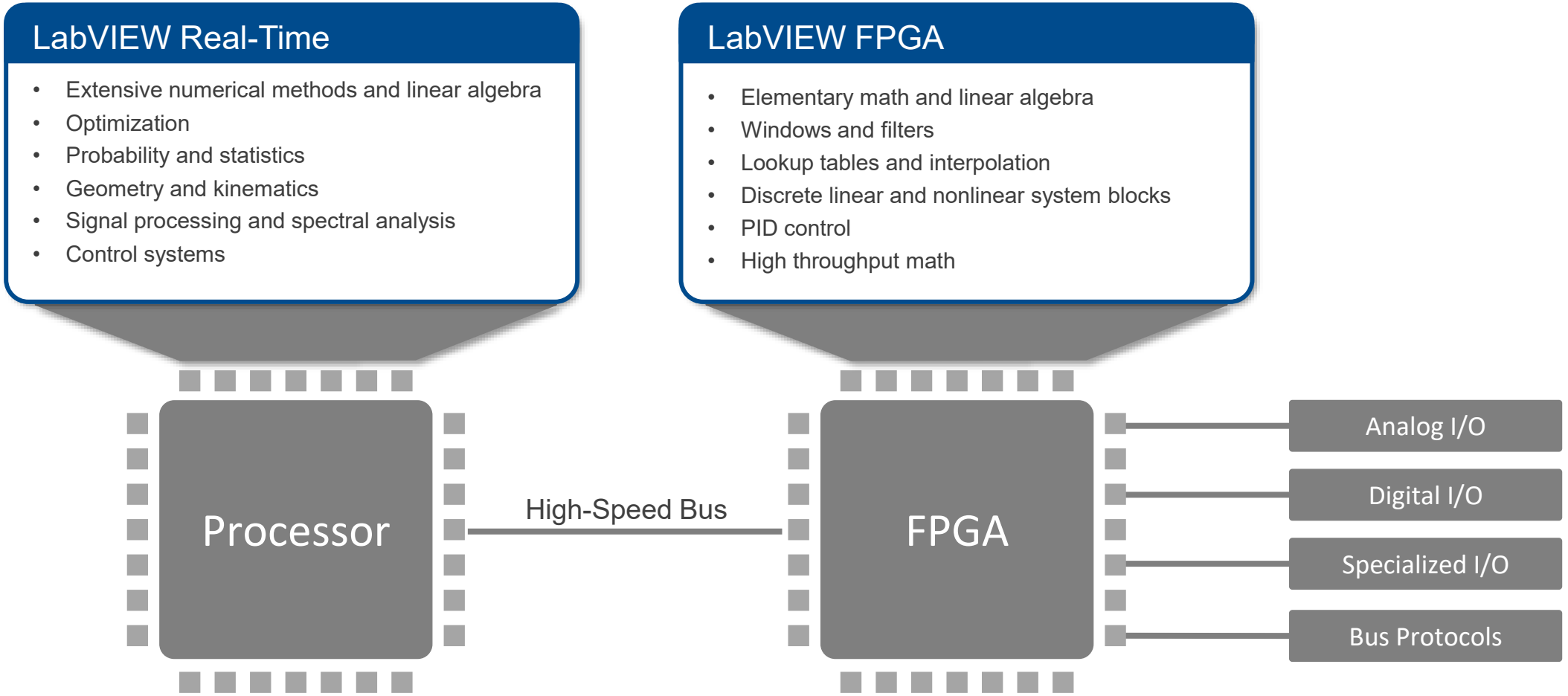


Open, Deterministic OS

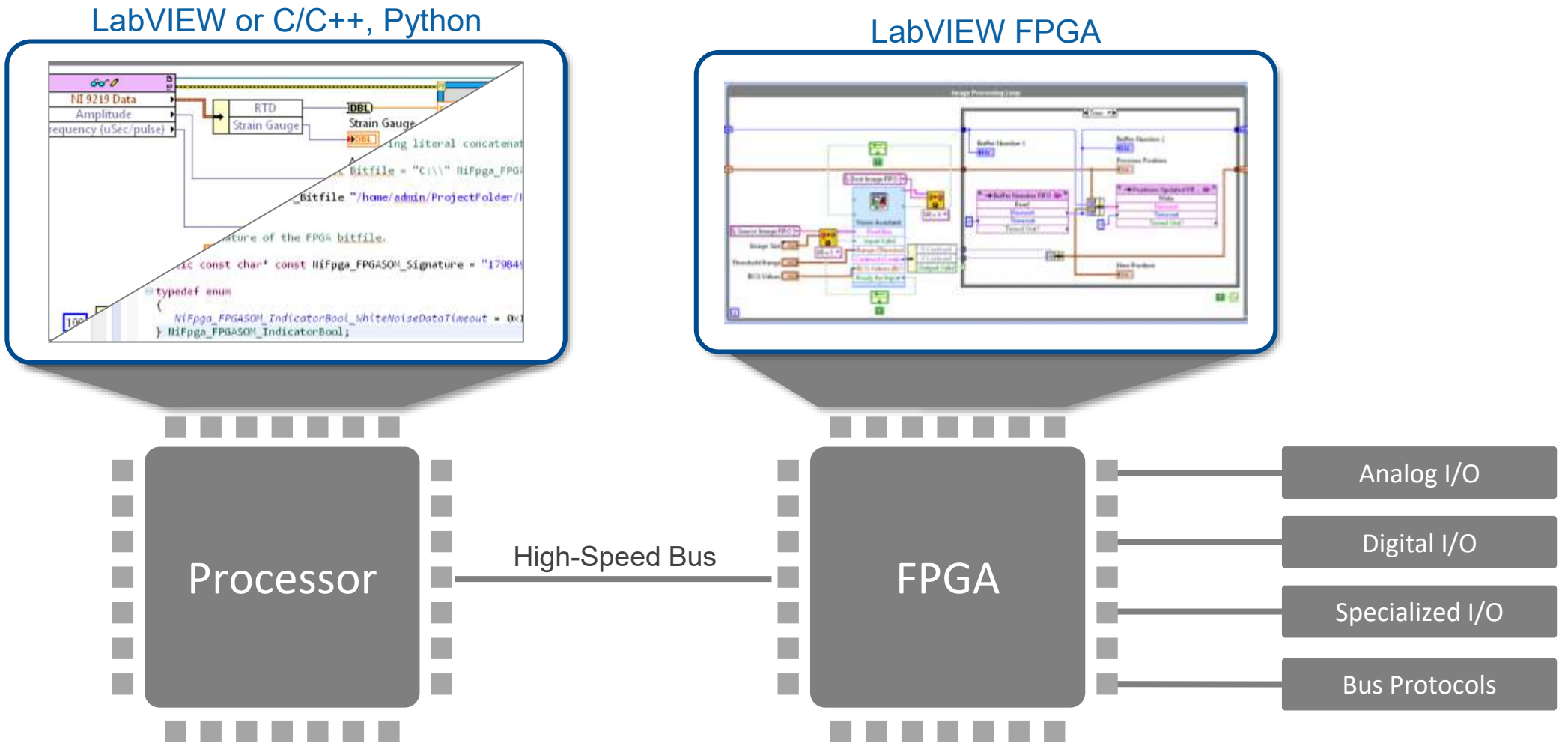




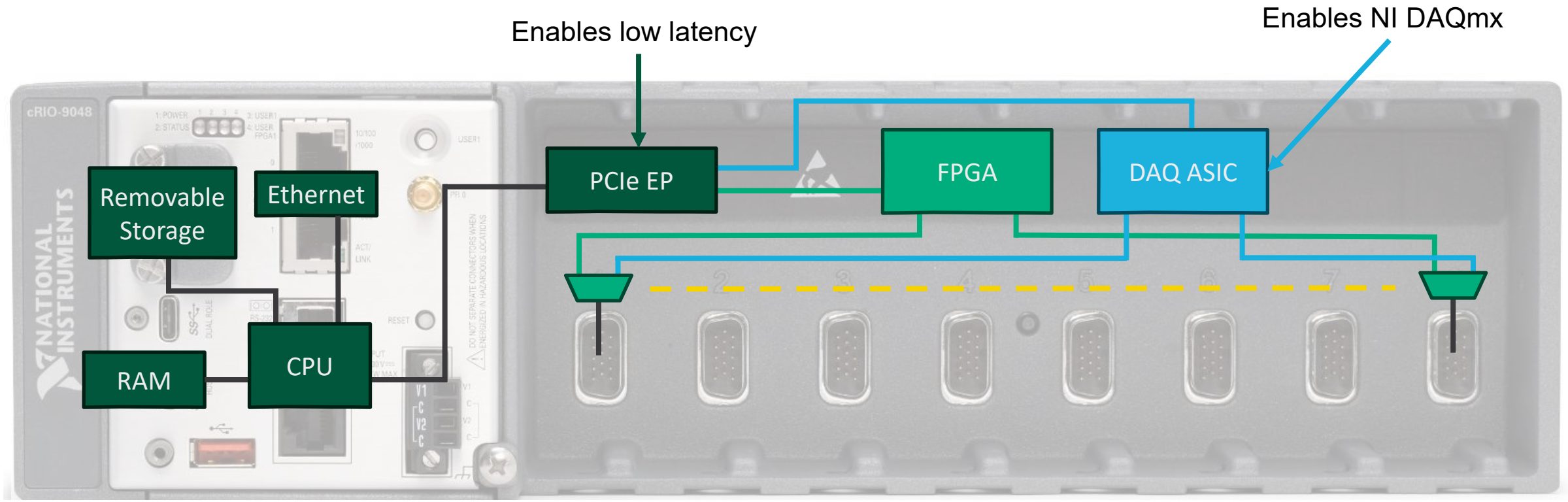
Application Software for the CompactRIO Platform



Application Software for the CompactRIO Platform



HW Architecture



CompactRIO Synchronization

TSN synchronizes multiple devices with no programming required

Formalized as IEEE 802.1 AS (an IEEE 1588 profile)

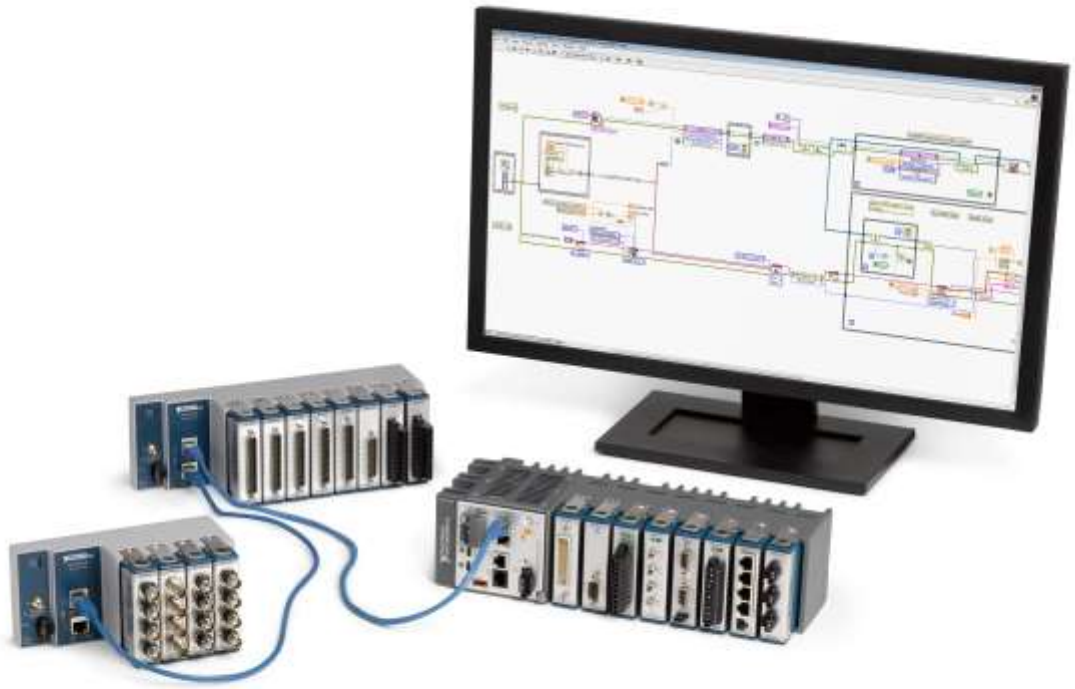
Synchronized networked systems within 1 µs of network time

Synchronize distributed measurements within 1 µs

- Same start time
- No drift over time

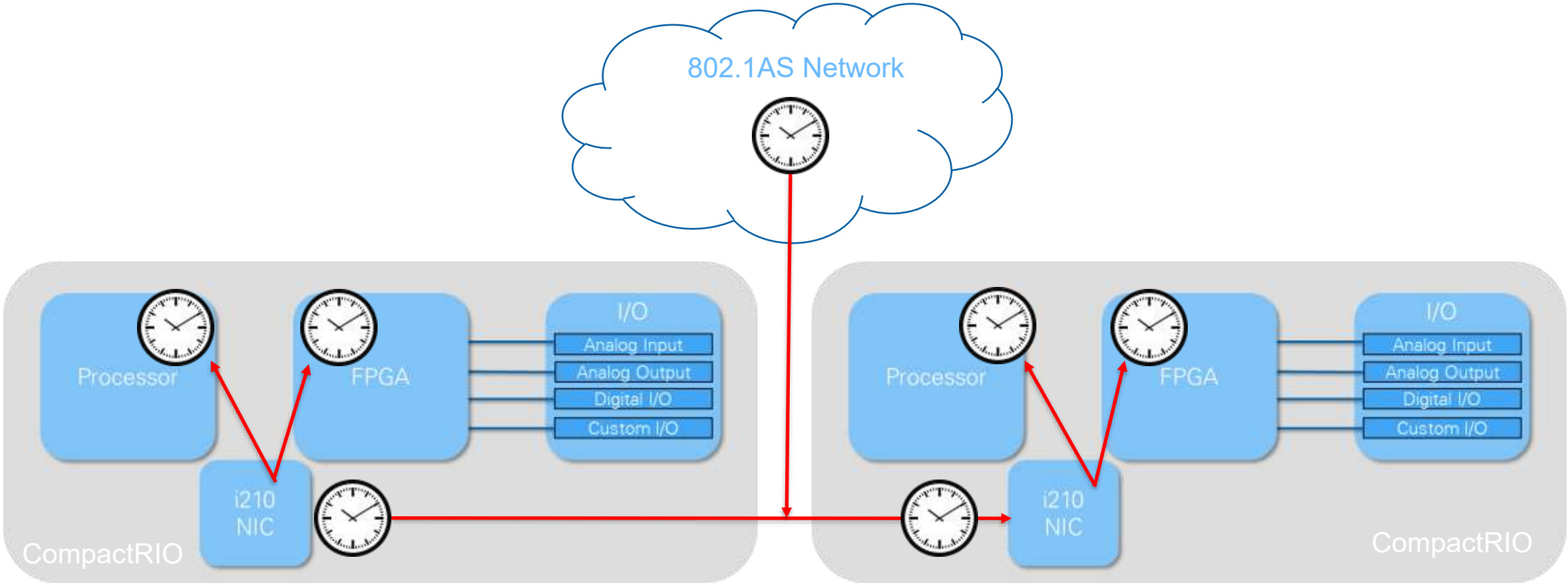
(Locally) Expand IO using CompactDAQ with TSN and FieldDAQ

Expand Processors & IO using additional cRIOs



IEEE 802.1AS on CompactRIO And Industrial Controllers

Time Synchronization



All CPUs and FPGAs agree on time (enables start time synchronization)

Local clocks are disciplined (eliminates drift)

EMERSON

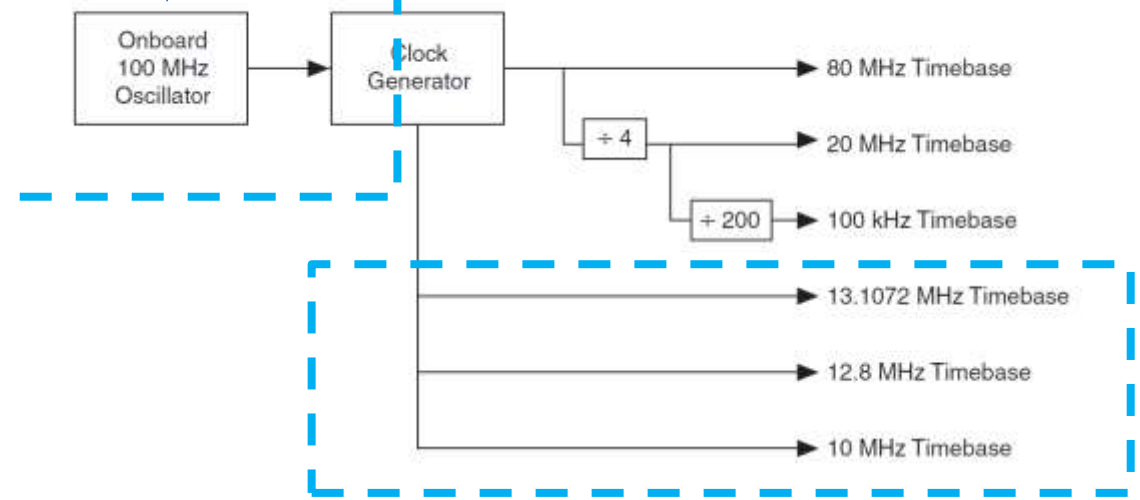
Chassis Clock Generation & AI Timing Engines

Chassis Clock Generation

Network Time

Adjusts

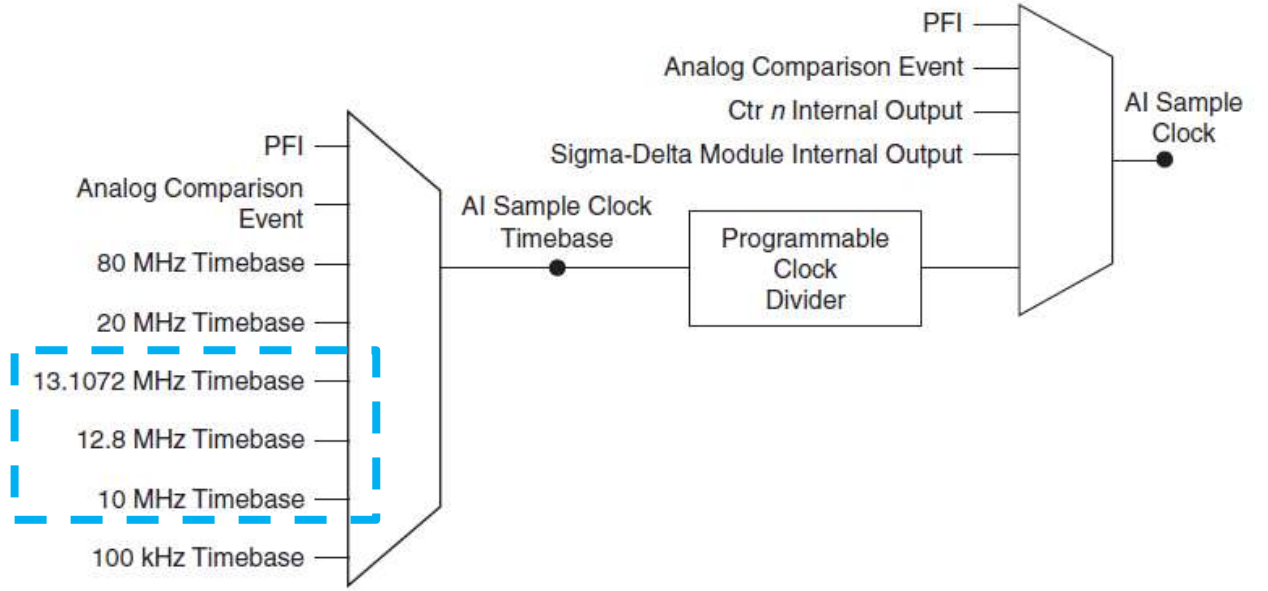
Figure 7-1. Clock Routing Circuitry



Only on TSN Chassis

AI Timing Engines

Figure 3-1. AI Sample Clock Timing Options



Rugged Specifications and Certifications

Environmental

Temperature (IEC 60068-2-1 and IEC 60068-2-2)	
Operating	-40 °C to 70 °C
Storage	-40 °C to 85 °C

Caution Failure to follow the mounting instructions in the user manual can cause temperature derating. Visit *ni.com/info* and enter Info Code `criomounting` for more information about mounting configurations and temperature derating.

Ingress protection	IP 40
Operating humidity (IEC 60068-2-56)	10% RH to 90% RH, noncondensing
Storage humidity (IEC 60068-2-56)	5% RH to 95% RH, noncondensing
Pollution Degree (IEC 60664)	2
Maximum altitude	5,000 m

Indoor use only.

Shock and Vibration

To meet these specifications, you must mount the system directly on a flat, rigid surface as described in the user manual, affix ferrules to the ends of the terminal lines, and provide strain relief for all cables.

Operating vibration	
Random (IEC 60068-2-64)	5 g_{rms}, 10 Hz to 500 Hz
Sinusoidal (IEC 60068-2-6)	5 g, 10 Hz to 500 Hz
Operating shock (IEC 60068-2-27)	30 g, 11 ms half sine; 50 g, 3 ms half sine; 18 shocks at 6 orientations

Hazardous Locations

U.S. (UL)	Class I, Division 2, Groups A, B, C, D, T4; Class I, Zone 2, AEx nA IIC T4
Canada (C-UL)	Class I, Division 2, Groups A, B, C, D, T4; Class I, Zone 2, Ex nA IIC T4
Europe (ATEX) and International (IECEx)	Ex nA IIC T4 Gc

Certifications

EMERSON

Security | Roadmap to Security Compliance

Short Term

Documentation to meet existing requirements

- ✓SBOMs / LOVs
- ✓Secure configuration instructions
- ✓User-configured TPM
- ✓NIST 800-171 compliance documentation
- ❑Vulnerability management (Ongoing)

Medium Term

Improve security experience with product investments

- ✓ Improved baseline security with 11.1 distribution (based on Kernel 6.6)
- ✓ SNAC 2.0 script and documentation
- ❑ Data encryption
- ❑ Network authentication
- ❑ Secure boot / TPM
- ❑ Vulnerability management (Ongoing)

Long Term

Security Maturity and certifications

- ❑ Faster processors
- ❑ Full toolchain security
- ❑ Industry Certifications
- ❑ CRA Compliance
- ❑ Updated hardware architecture
- ❑ Vulnerability management (Ongoing)

www.ni.com/security

EMERSON | NI

What can you do with CompactRIO?

Embedded Test and Monitoring

- Structural health monitoring with distributed, synchronized vibration and strain measurements
- Novel electrical power monitoring applications such as deploying algorithms for fault detection, equipment maintenance, and power quality.
- Industrial sensor measurements at the edge that need more speed, precision, processing, or synchronization than available from PLCs/PACs.
- Mobile or portable test systems that need quality measurements, logging, and processing in a rugged environment.
- Small-scale HIL

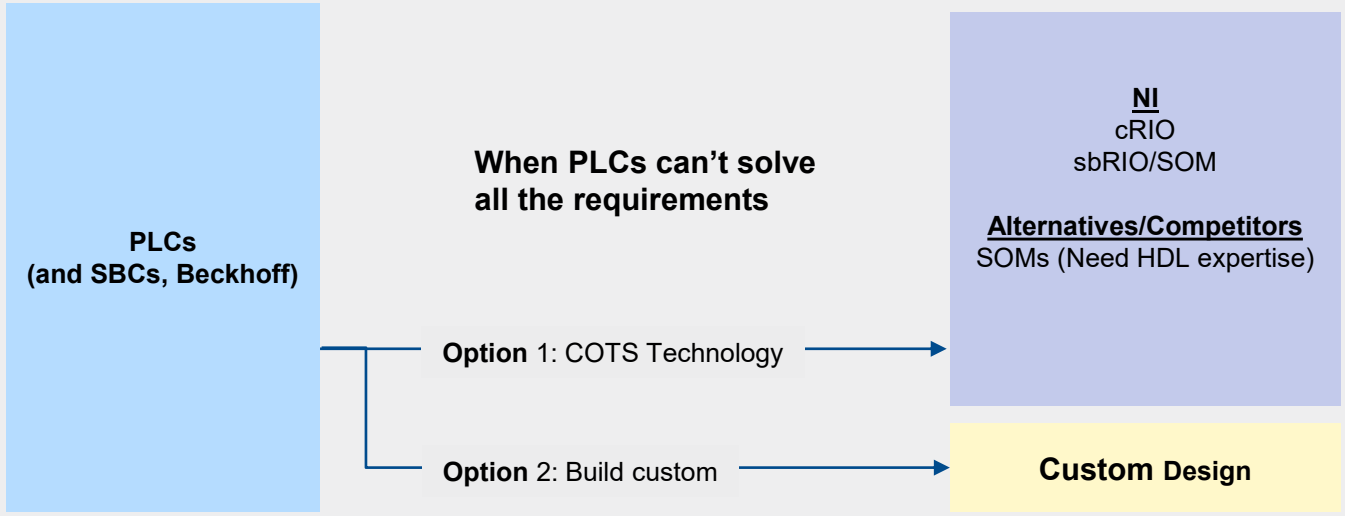
Control

- Engineer-designed, high-speed automation systems for high-cost assets deployed to fewer than 1,000 units.
  - Oil and gas applications
  - Off-highway ag equipment and Earth movers
- Rapid control prototyping
- Control test equipment and combine with data acquisition
  - Don’t add a PLC with ladder logic, use cRIO with LabVIEW.
  - Great for academic research

CompactRIO Ideal Use Cases

Industrial Domain

Sweet Spot: PLC's can't solve all requirements, rugged environment, lack of custom design expertise, deploying tens to hundreds/yr



DAQ Applications

Sweet Spot: Applications that need DAQ plus a little something extra

DAQ + Long-Term, Uninterrupted Operation
High-quality measurements with an RTOS for reliable, uninterrupted operation

DAQ + Advanced DUT Control
Provide more complex/high-speed control of a DUT to take measurements for ADV/APT

DAQ + Stand-Alone Operation
Small, headless data logger with high-quality measurements

DAQ + In-Line Signal Processing
Small, headless data logger with high-quality measurements

EMERSON

CompactRIO Differentiation Summary

Primary Differentiators

FPGA: Perform advanced high-speed control (>10 kHz) and in-line signal processing made easy to non-HDL developers with LV FPGA

RT: Reliable determinism for control without needing to be an RT expert with LabVIEW RT and an open RTOS based on Linux

Timing & Sync: Best-in-class timing and synchronization that is easy to use across IO types and scale across controllers

Ruggedness: Deploy with confidence in the world's most demanding environments with a wide temp, shock, and vibe range

IO Quality: Make decisions for control loops or perform data analysis on measurements you can trust

Dev Efficiency: Complete projects with advanced control or processing requirements 4x faster than custom design

Differentiators Against Specific Competition

PLC's/Beckhoff
- Faster Control: >1 kHz (CPU) and >10 kHZ (FPGA)
- High-quality, sensor-based measurements
- Ruggedness: wider temp, shock, vibe range

Custom Design
- 4x faster time to market and dev efficiency
- Reduce the stress of lifecycle management (10+ year life)
- Harness power of RT and FPGA with non-experts

Dataloggers/DAQ
- Reliable long-term DAQ with an RTOS
- Mixed-measurement DAQ with a modular IO approach
- Advanced in-line signal processing with CPU and FPGA

EMERSON

cRIO & sbRIO Roadmap

Unreleased product features/capabilities and technology explorations are subject to change without notice. Feedback is encouraged.

Next Generation CompactRIO & Single-Board RIO Refresh

NI has planned refreshes for both CompactRIO and Single-Board RIO

- Refresh highest performance controllers first (cRIO-904x)

Key focus areas for next generation remain consistent with our past

- Continuous performance improvements
- New features to support evolving trends

Additional focus to maintain compatibility with existing products

- Optimize for re-use within existing system architectures
- Continue to support key software programming paradigms and features

Next Generation CompactRIO Refresh

Exploration Areas for Product Enhancements

- Improved processor performance
- Improved FPGA performance
- Security enhancements
- AI
- Ease of Use

Additional Notes

- Compatibility with existing form factor (size, mounting, etc)
- Maintain DAQmx and FPGA programming options
- Maintain TSN support for synchronization

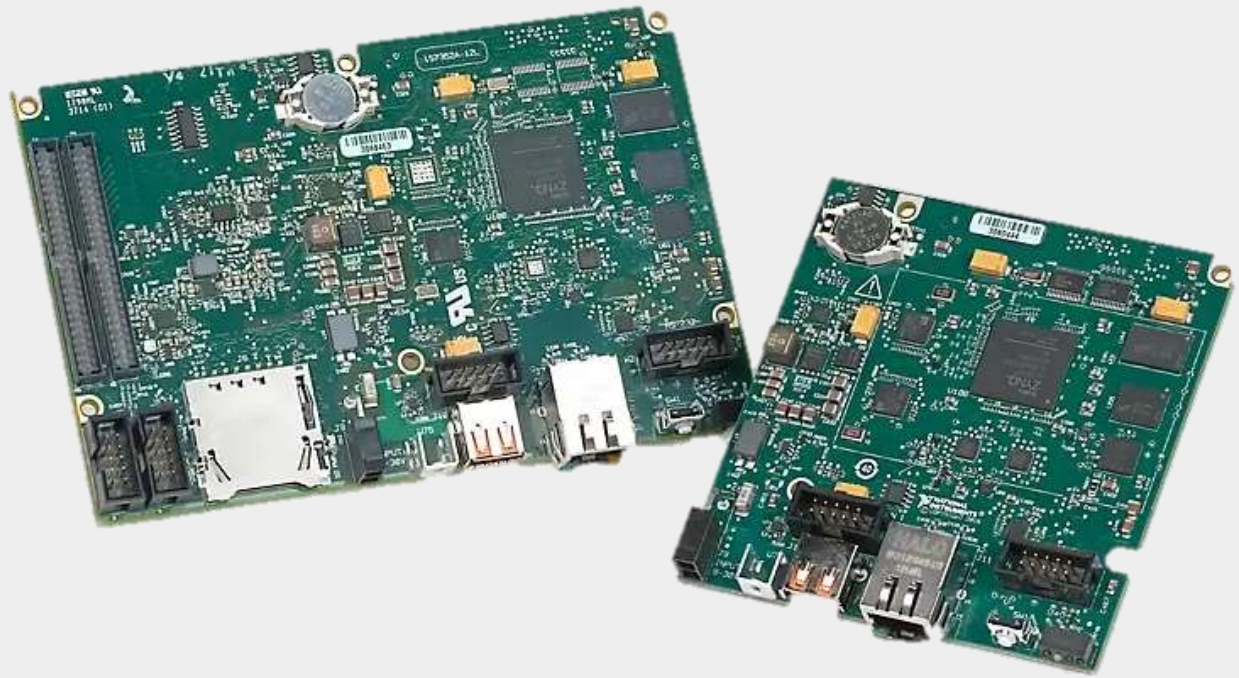
Single-Board RIO Refresh

Exploration Areas for Product Enhancements

- Improved processor performance
- Improved FPGA performance
- Security enhancements
- AI

Additional Notes

- Compatibility with existing form factor (size, mounting, etc)
- Thermal/cooling system integration requirements
- Maintain TSN support for synchronization



Getting Started

Up Front System Design

EMERSON

Create High Level Design

PC

HMI

- Display
- Send commands

Network

cRIO System

Real-Time **Processor**

- Sequencer
- System Monitoring
- Log data to file
- < 5kHz Control
- Embedded UI (on some models)
- Networking
- Data Acquisition

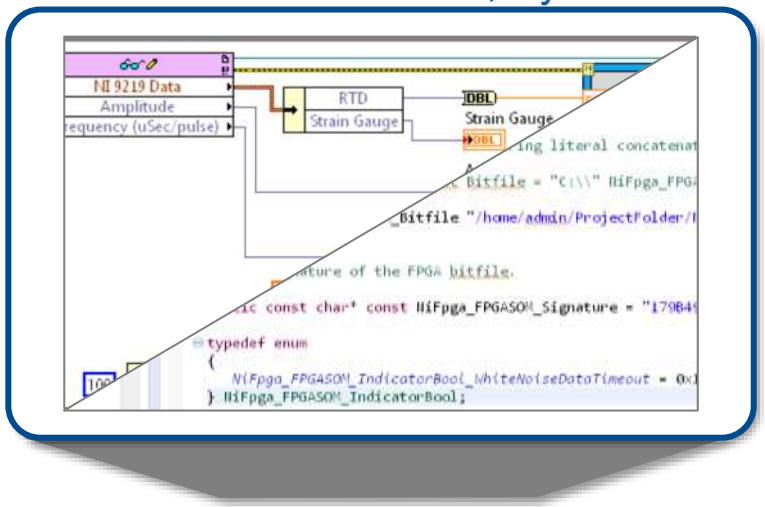
FPGA

- Acquisition
- > 5KHz Control
- Safe state logic
- Custom Protocols

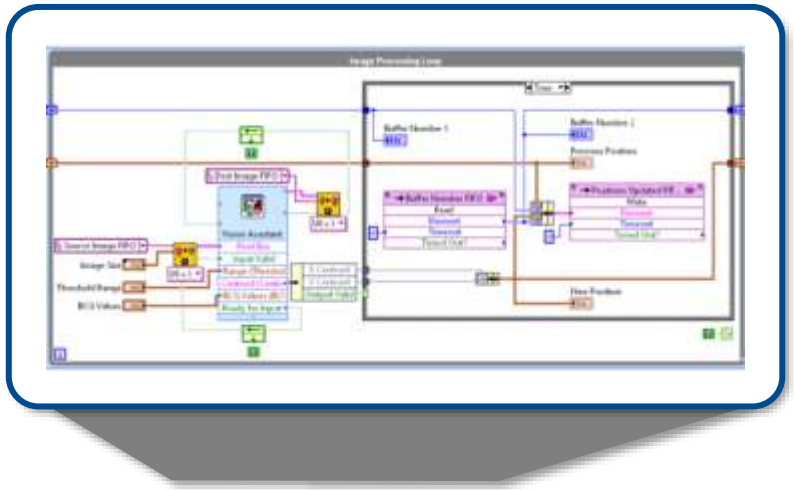
EMERSON

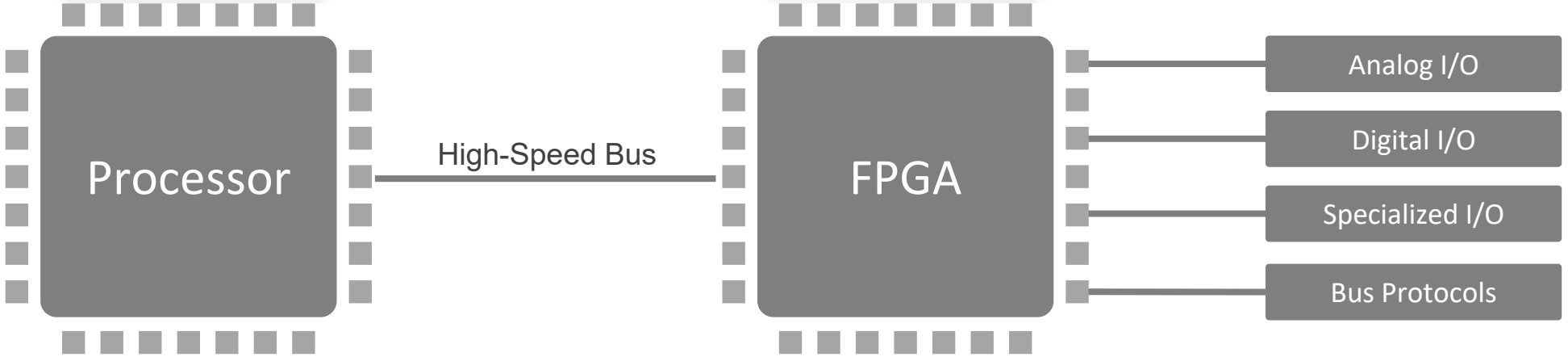
Pick the Application Software

LabVIEW or C/C++, Python



LabVIEW FPGA





Pick the IO

3 Things to Know Before Starting

Where are you going to access your IO

- Processor or FPGA

What type of ADC do you need for your application?

- SAR vs Delta Sigma
- Independent vs Multiplex

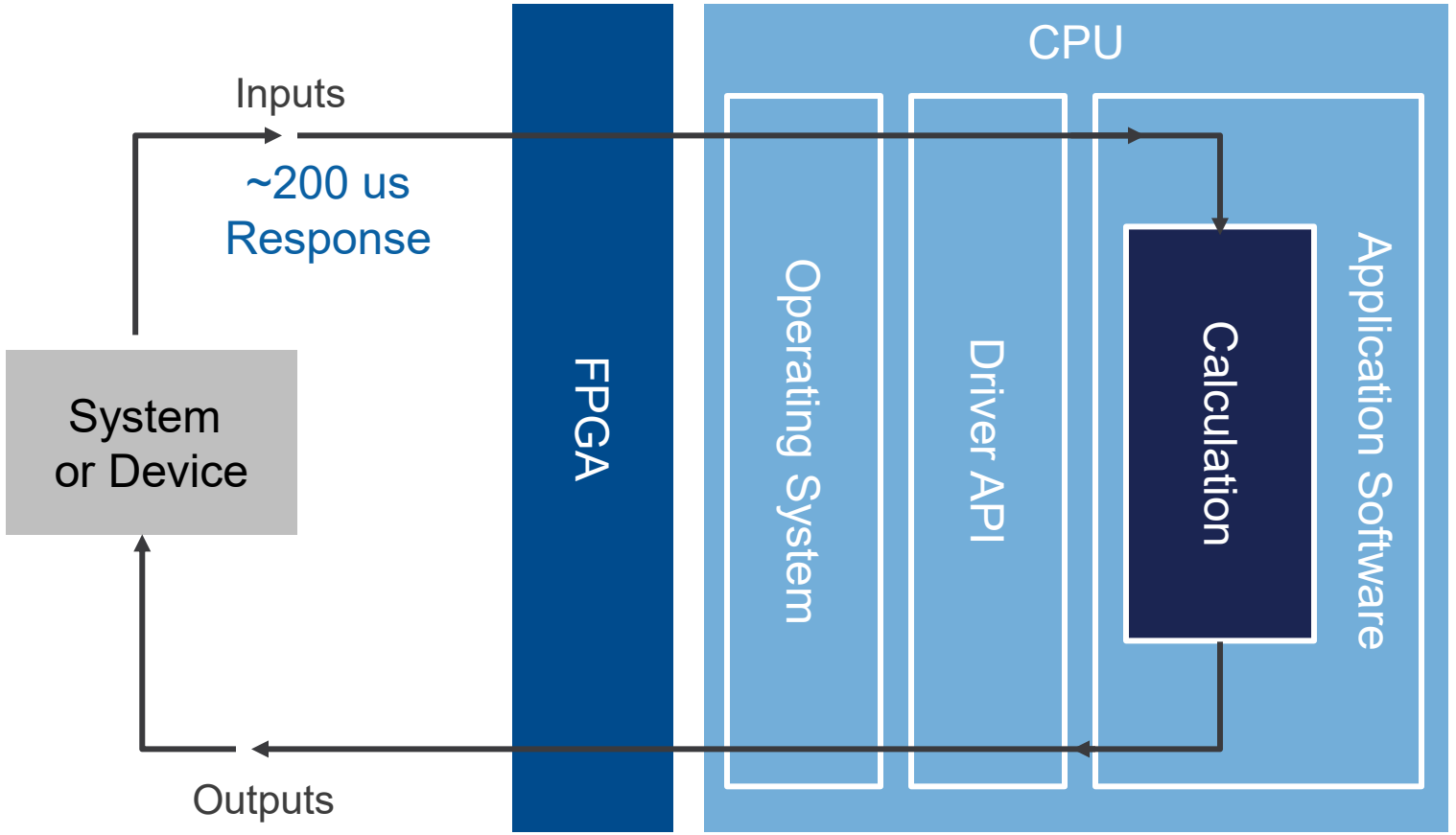
What IO API best fits your application?

- NI DAQmx vs Scan Engine vs Direct FPGA
- (Note: Scan Engine is LabVIEW Only)

Pick the IO Access Location

Option 1: Processor Based Access

Easiest

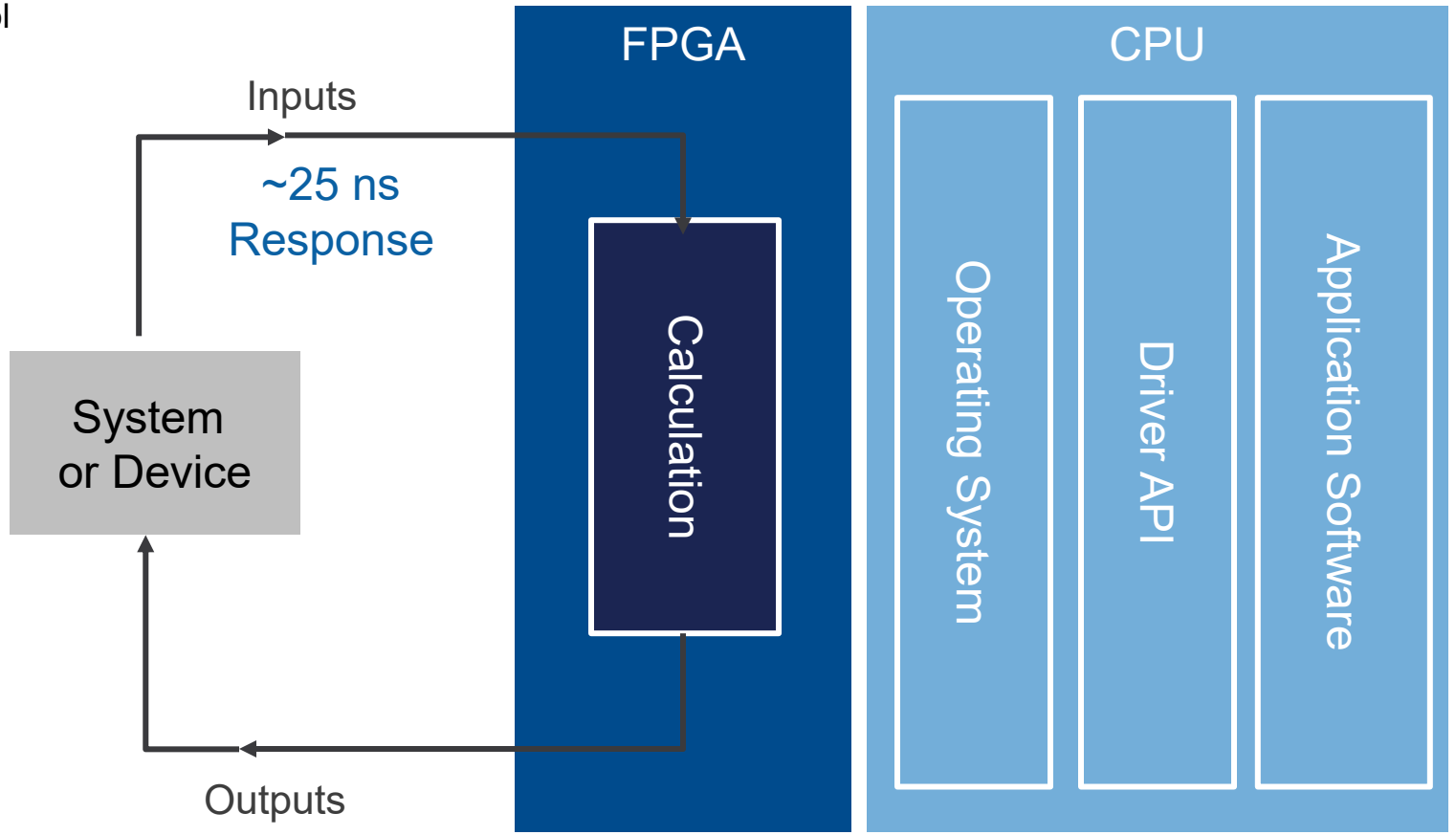


Pick the IO Access Location

Option 2: FPGA Based Decision Making

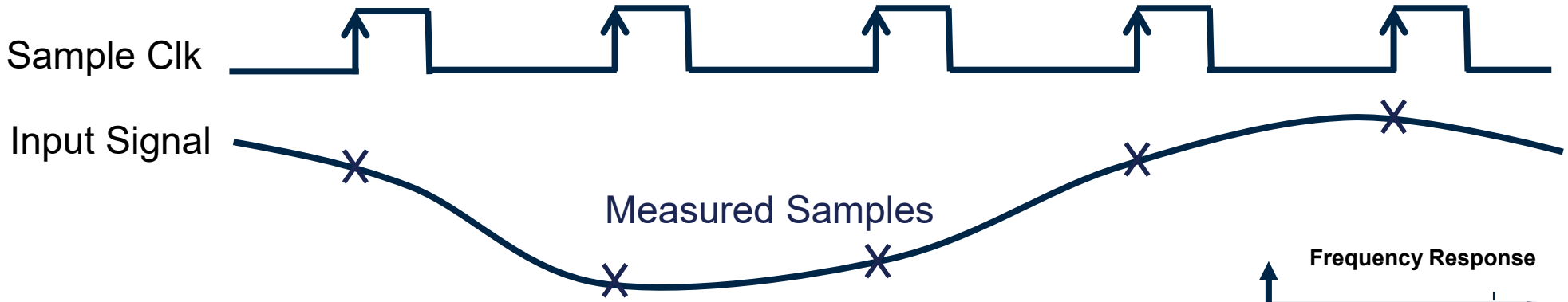
Safest, faster, most control

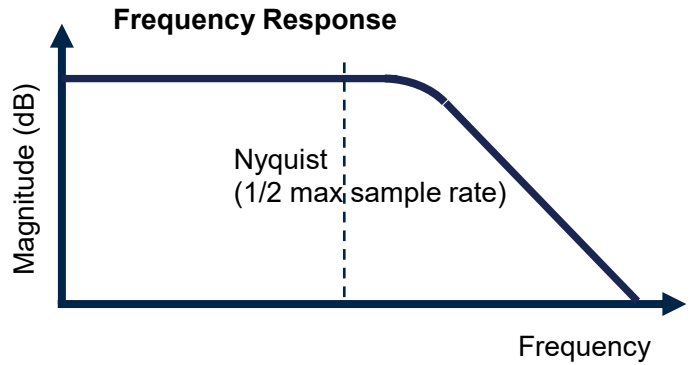
Most work

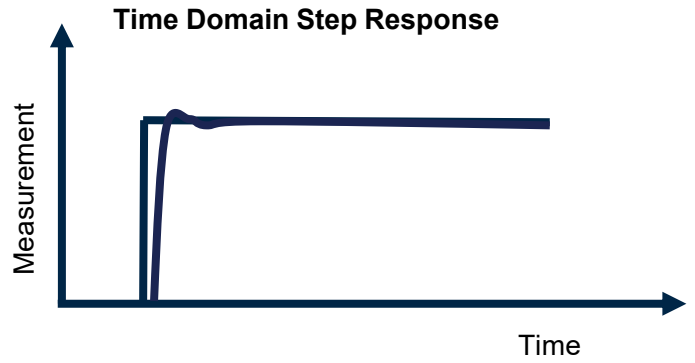


EMERSON

Pick the ADCs: SAR Architecture



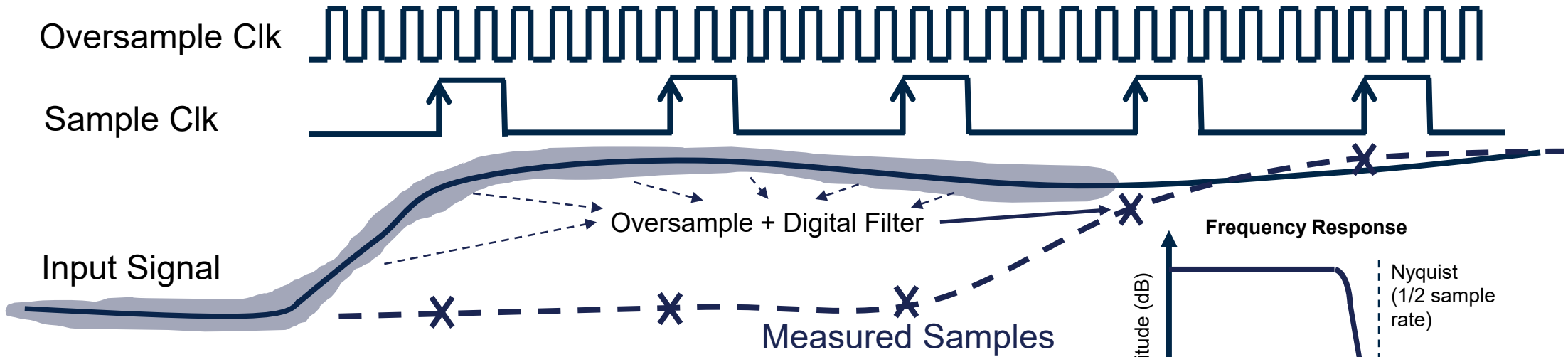


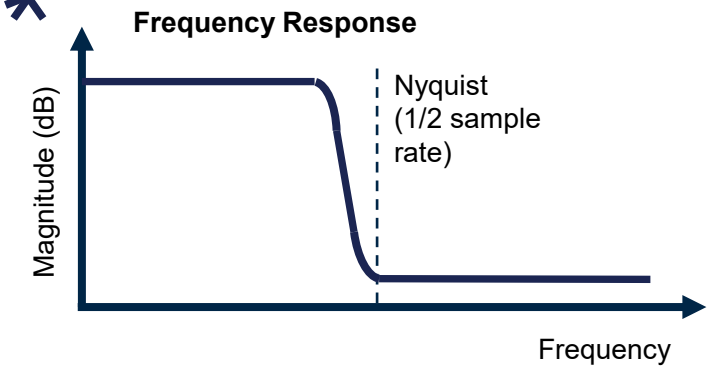


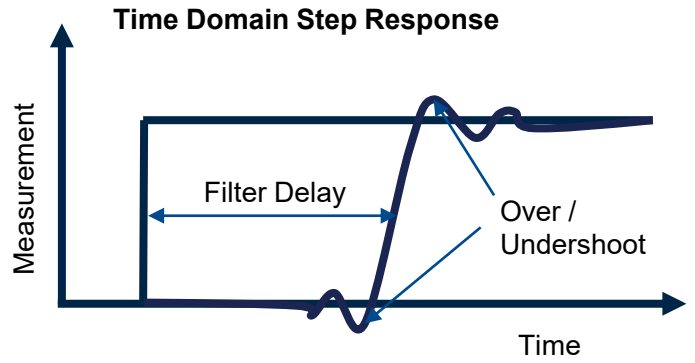
SAR (Successive Approximation Register) ADCs

- Takes an immediate measurement on each Sample Clock
- Fast sample rates (up to 1 MS/s on C Series)
- Wide Bandwidth, with Bandwidth > Sample rate
- Support arbitrary timing
- Fixed noise, independent of sample rate
- Best for control applications

Pick the ADCs: Delta-Sigma with Brickwall Filter



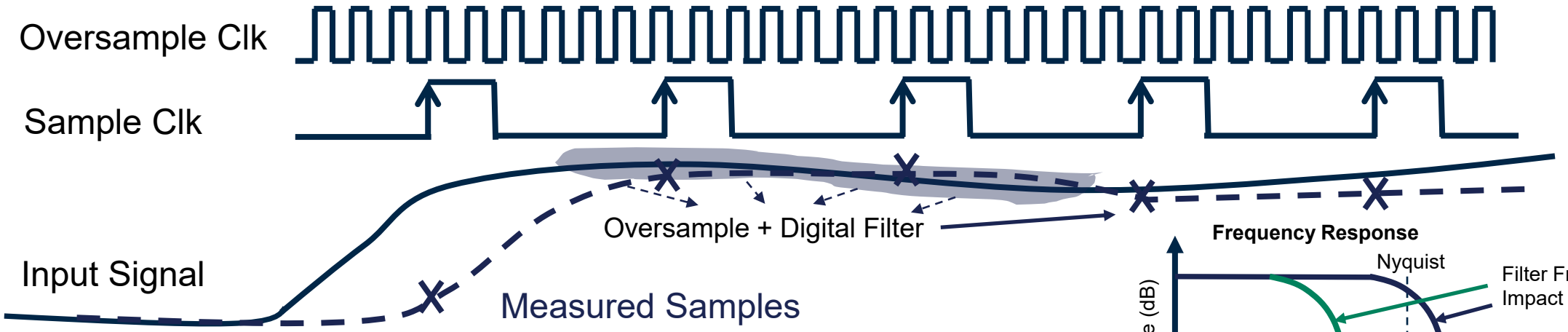


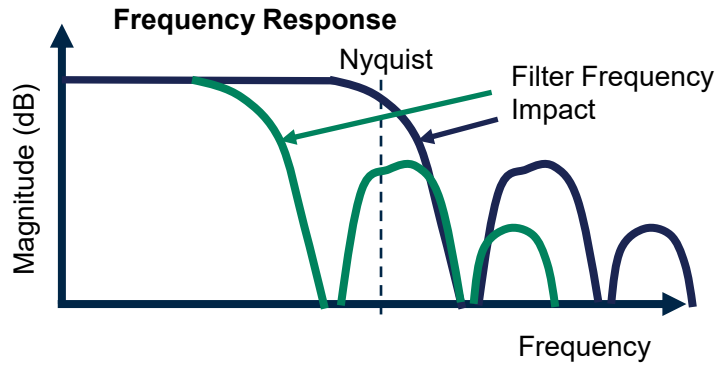


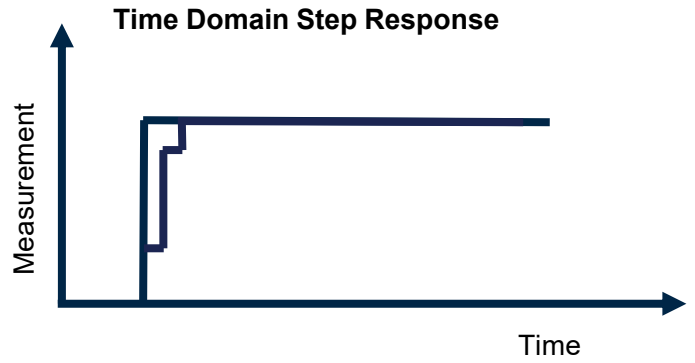
Delta-Sigma ADCs

- Moderate sample rates (up to 102.4 kS/s on C Series)
- Limited sample rates based on dividing down Oversample Clock
- Noise may improve with lower rates
- Full alias protection
- Great for waveform acquisitions, data logging
- Very high latency, bad for control applications

Pick the ADCs: Delta-Sigma with Comb Filter



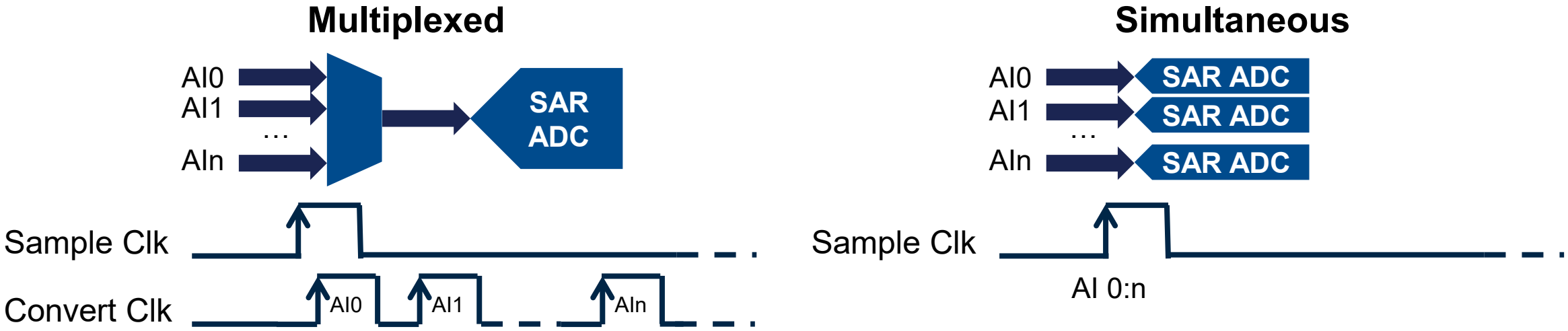




Delta-Sigma ADCs

- Moderate sample rates (up to 50 kS/s on C Series)
- Limited sample rates based on dividing down Oversample Clock
- Rejection and notch adjustable with Filter Frequency and sample rate
- Partial alias protection
- Great for waveform acquisitions, data logging
- Moderate latency, OK for control applications

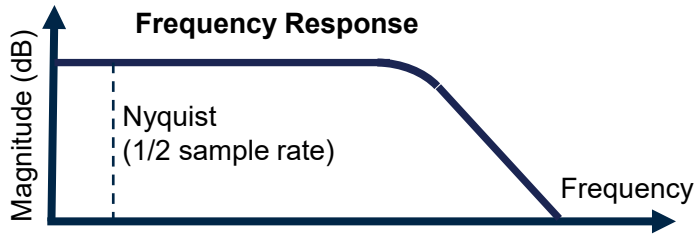
Pick the ADCs: Multiplexed Inputs Impact



Compared to Simultaneously Sampled Inputs:

- Not phase aligned
- Higher noise, crosstalk, and aliasing

Input bandwidth >> Sample Rate



Pick the DACs: Analog Output Architectures

Same Story: Choose based on application

C Series AO Voltage Module Comparison						
Product Name	**Number of channels**	**Output Range**	**Drive**	**Update Rate**	**Linearity**	**Galvanic Isolation**
Simultaneously Updated String DAC						
NI 9263	4	±10V	1 mA	100 kS/s	12 LSB	Bank
NI 9264	16	±10V	1 mA	25 kS/s	12 LSB	Bank
Simultaneously Updated R2R DAC						
NI 9262	6	±10V	2.5 mA	1 MS/s	2 LSB	Bank
NI 9269	4	±10V	5 mA	100 kS/s	2 LSB	Chan-Chan
Simultaneously Updated Delta-Sigma DAC						
NI 9260	2	3 Vrms	7 mA	51.2 kS/s	-110 dB	None

C Series AO Current Module Comparison						
Product Name	**Number of channels**	**Output Range**	**Compliance Voltage**	**Update Rate**	**Linearity**	**Galvanic Isolation**
Simultaneously Updated String DAC						
NI 9265	4	0 to 20 mA	12 V	100 kS/s	16 LSB	Bank
NI 9266	8	0 to 20 mA	12 V	24 kS/s	16 LSB	Bank

String DACs – best for control loops

- Excellent differential linearity
- Very low glitch energy
- Worse integral non-linearity and accuracy

R-2R DACs – best for DAQ/Waveform

- Very good integral non-linearity and accuracy
- High glitch energy at code transitions

Delta-Sigma DACs – best for audio waveform generation

- Excellent dynamic range and SFDR
- Moderate update rates (up to 51.2 kS/s in C Series)
- Long latencies and limited sample rates

EMERSON | NI

Pick the IO

Cheat Sheet

16 bit typically = SAR

24 bit typically = Delta Sigma

Always review input delay specification for Delta Sigma modules

Do try to match sample rates on modules

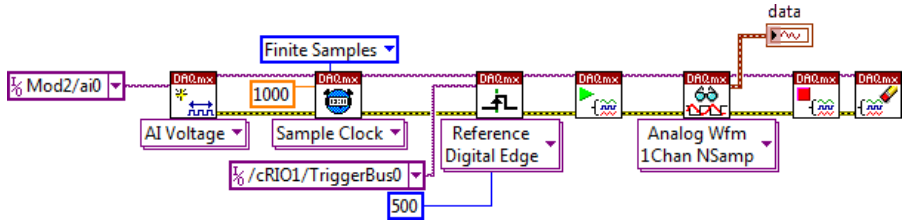
Do try to avoid multiplexed modules

Do steer towards SAR/String (typically 16 bit) modules for control

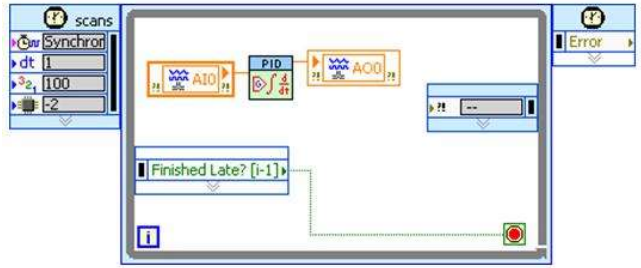
Do steer towards Delta Sigma for waveform / logging applications

Pick the IO API (per slot)

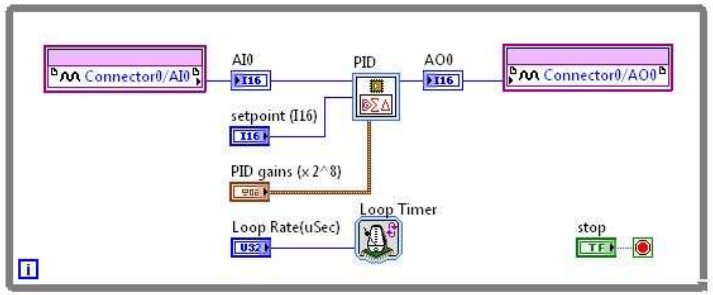
Real-Time (NI-DAQmx)
- Simple API for waveform and control from processor
- Supports control loop rates up to 5 kHz
- All channels in a task are synchronized
- Real-Time uses appropriate driver, such as NI-DAQmx or XNET



Real-Time Scan (IO Variables)
- Single-Point data acquisition and control from LabVIEW Real-Time
- Supports control loop rates up to 1 kHz
- Expandable IO w/ NI 9145 chassis
- LabVIEW Only



LabVIEW FPGA
- Single-point data acquisition and control from LabVIEW FPGA
- Supports control loop rates up to 10s of MHz
- Hardware-level determinism and reliability for control loops
- In-line signal processing for live filtering, calculations, and more



API Deep Dives

Better Understanding → Better Design

EMERSON

NI DAQmx IO on cRIO

Better Understanding → Better Design

EMERSON

DAQmx on CompactRIO

Faster Time to Measurement

Module identification and debugging in MAX

API ideal for measurements

Module calibration

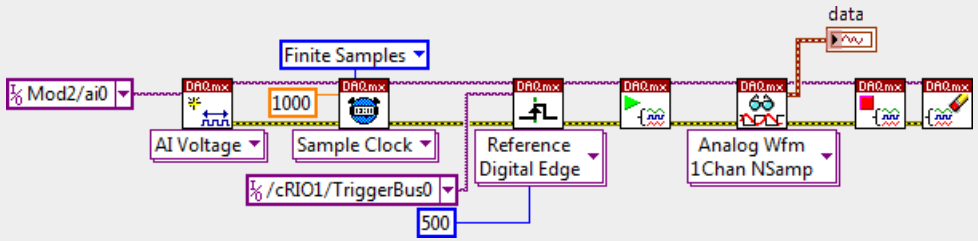
DAQmx Improvements on CompactRIO

Timing engine per slot for multi-rate and mixed measurement applications

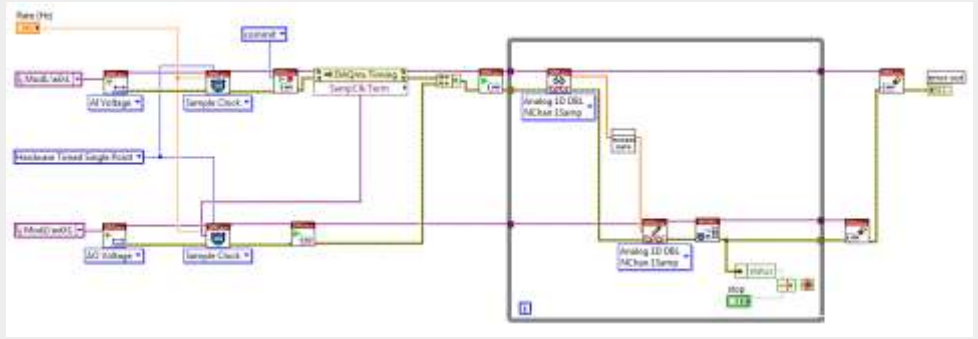
Deterministic control up to 5 kHz (1kHz actual) with Hardware-Timed Single Point

Measurement synchronization with TSN

Waveform Streaming



Deterministic Control



CompactRIO NI DAQmx Engines

One engine per slot

Input and output engine per slot

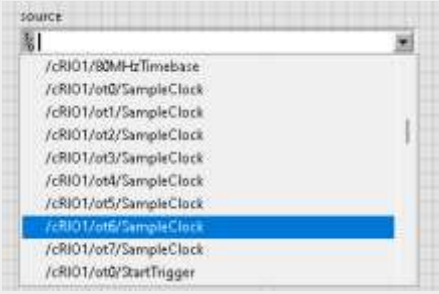
Four counters per chassis

Use for local synchronization

References:
CompactRIO Timing Engine

CompactDAQ Technology

SampTimingEngine Value	Analog Input, Digital Input, and NI 9361 Timing Engine Used	Analog Output and Digital Output Timing Engine Used
0	it0	ot0
1	it1	ot1
2	it2	ot2
3	it3	ot3
4	it4	ot4
5	it5	ot5
6	it6	ot6
7	it7	ot7



Get the drivers right…

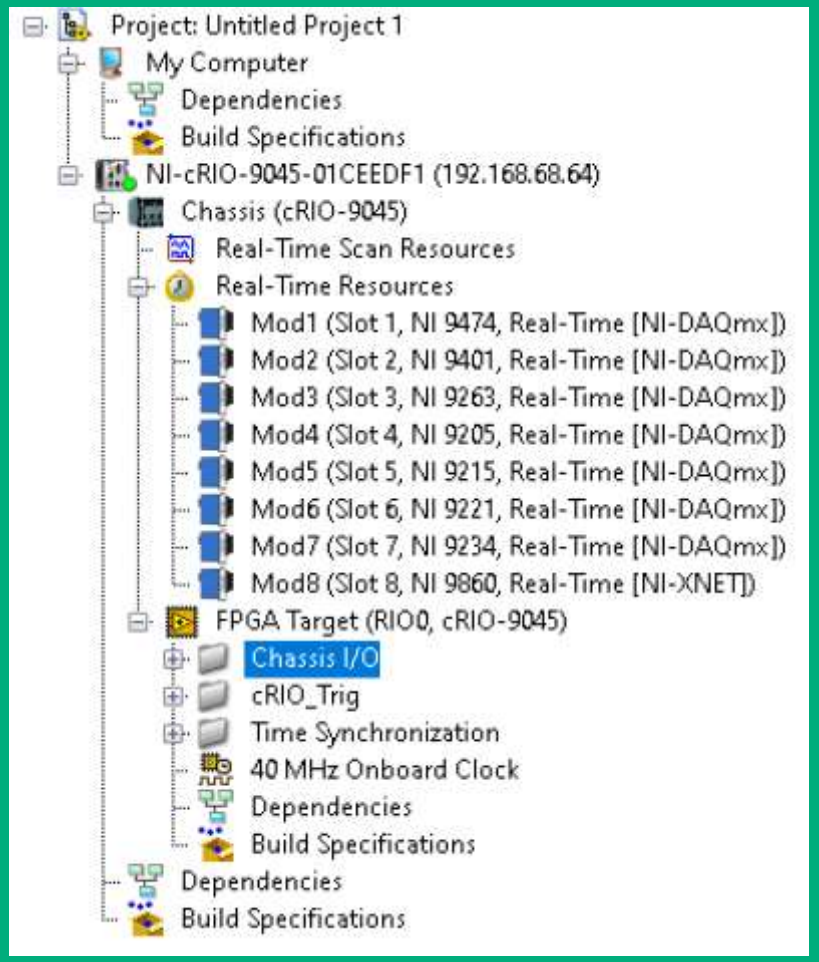
Host and cRIO driver version must match

Check versions in MAX

Update as necessary

cRIO + DAQmx requires:

- CompactRIO Support
- NI-DAQmx
- NI-Sync



Demo: Configuring cRIO for DAQmx

Create the LabVIEW Project

Create new blank LabVIEW Project

- Create Project → Blank Project

Add cRIO

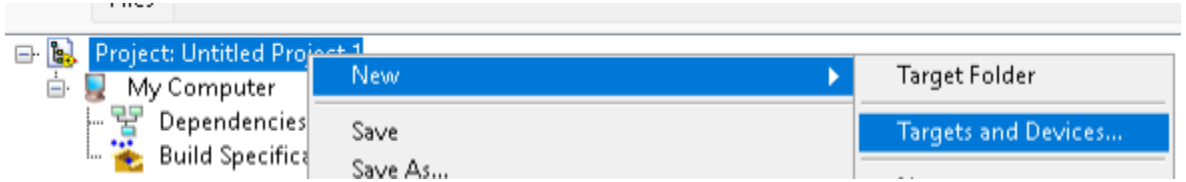
- Right click on Project
- New → Target and Devices

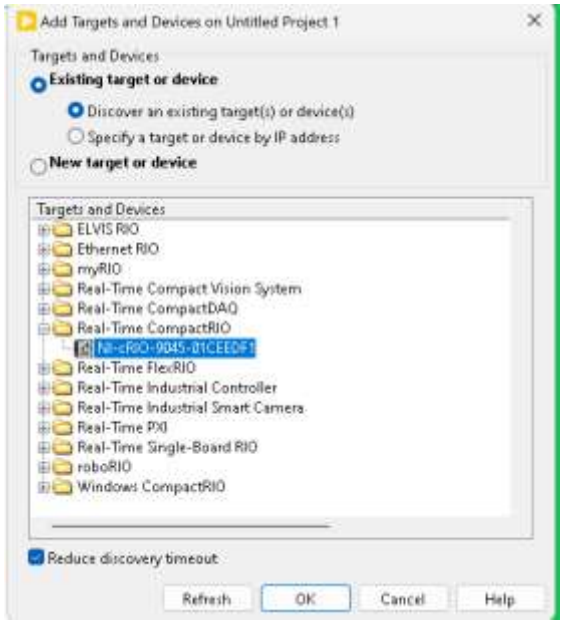
Select your cRIO from Real-Time CompactRIO

- Computer and cRIO must be on same subnet
- Modules will automatically be added with default settings

Delete all modules from:

- Real-Time Scan Resources
- FPGA Target





Add Modules to Real Time Resources

Make sure connected to cRIO

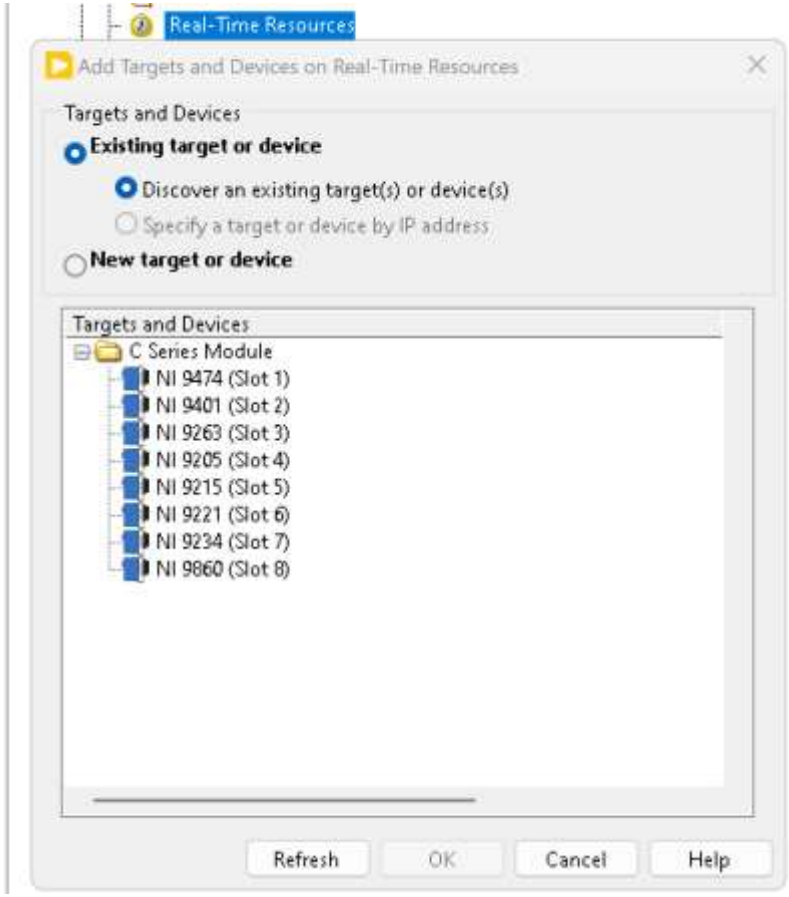
- Right click on cRIO
- Connect

Add modules to Real-Time Resources

- Right click on Real-Time Resources
- New → C Series Modules
- Select desired modules from GUI
- Select OK

Deploy settings from Chassis

- Right click on Chassis
- Deploy All



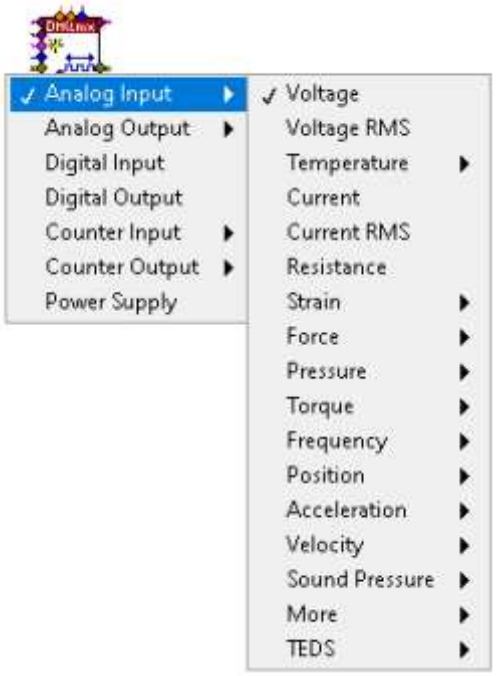
Configure the Modules / IO

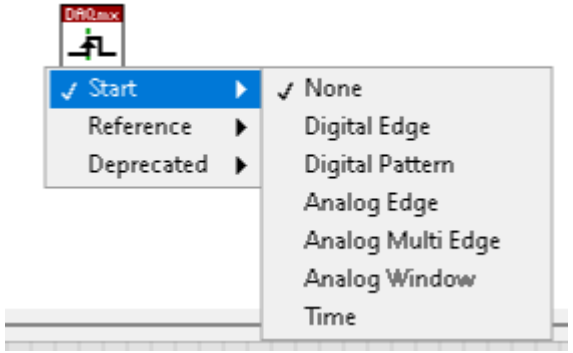
Use NI DAQmx API

Individual Channels: DAQmx Create Channel

Timing: DAQmx Timing

Triggering: DAQmx Start Trigger



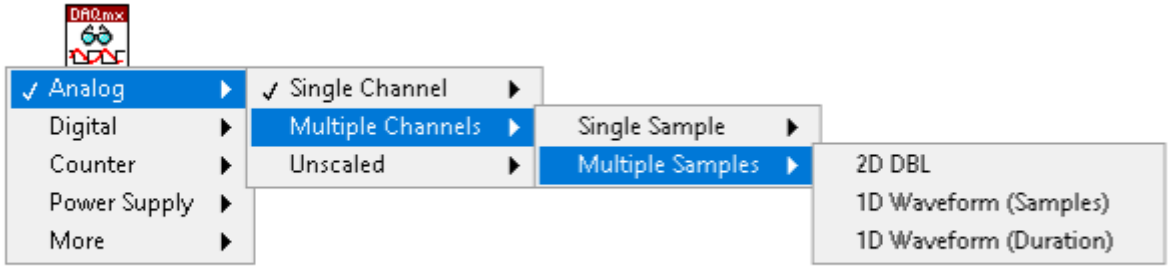


Access the IO

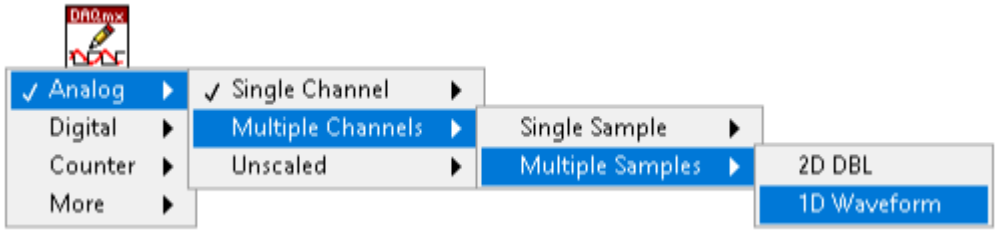
Use NI DAQmx API

Start Acq / Gen: DAQmx Start Task

Read Channels: DAQmx Read



Write Channels: DAQmx Write



Stop Acq / Gen: DAQmx Stop Task

Start with shipping examples

Open examples under cRIO

Open example finder

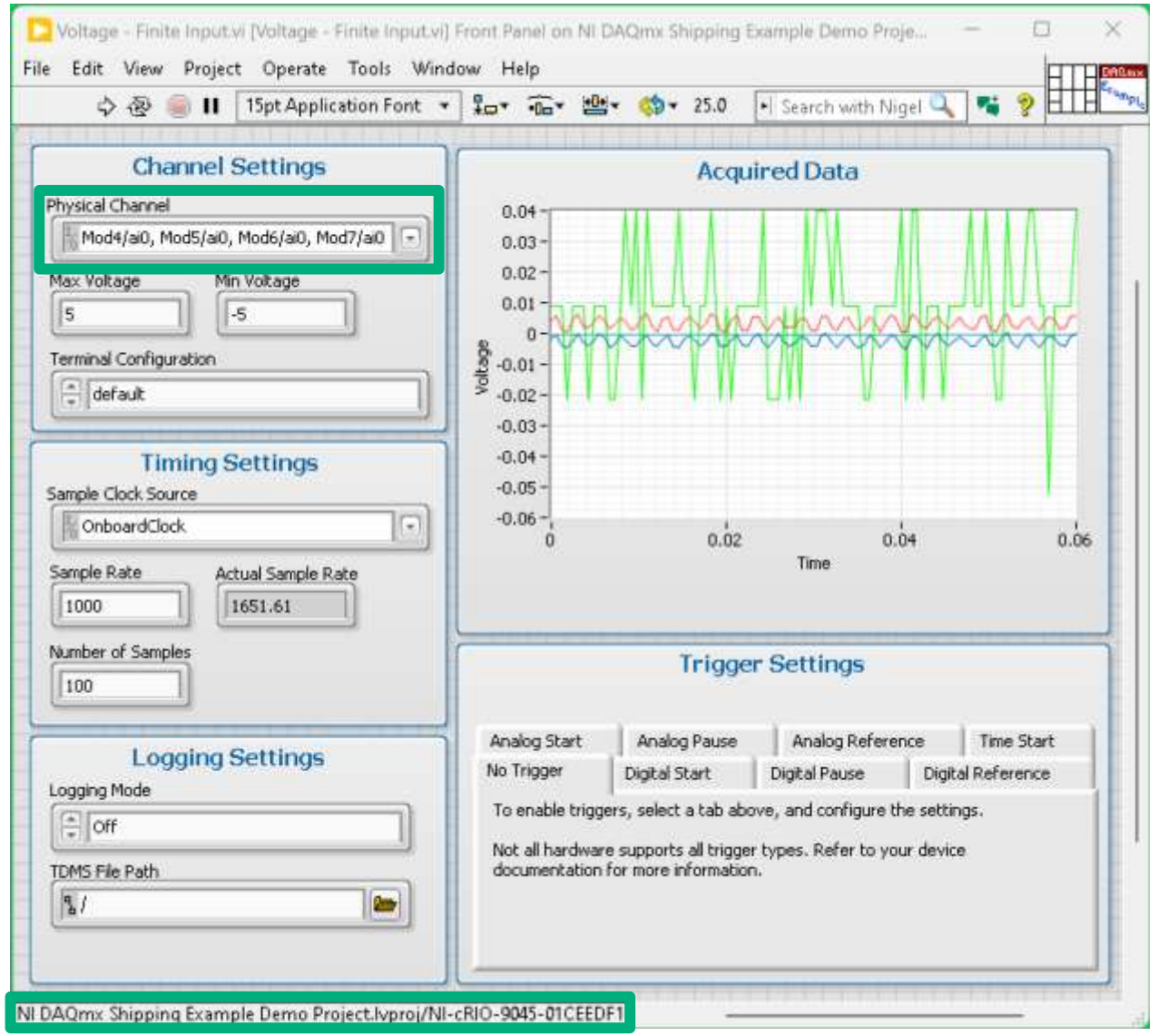
Open “Voltage – Finite Input.vi” under the cRIO

- Navigate to Hardware Input & Output → DAQmx → Analog Input → Voltage – Finite Input.vi
- Add it under your cRIO target in the project
- Close the example in memory
- “Connect” to the cRIO (get green connection status)
- Re-open the example under your cRIO target

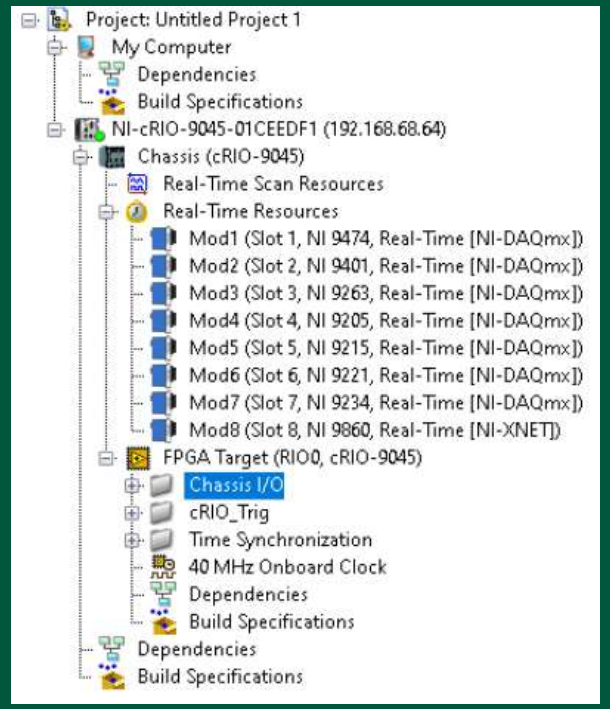
Setup multiple modules synchronization and run

- Select the first AI channel from each AI module
- Run

Show Block Diagram and highlight channel and clock config



Summary: Configuring cRIO for DAQmx

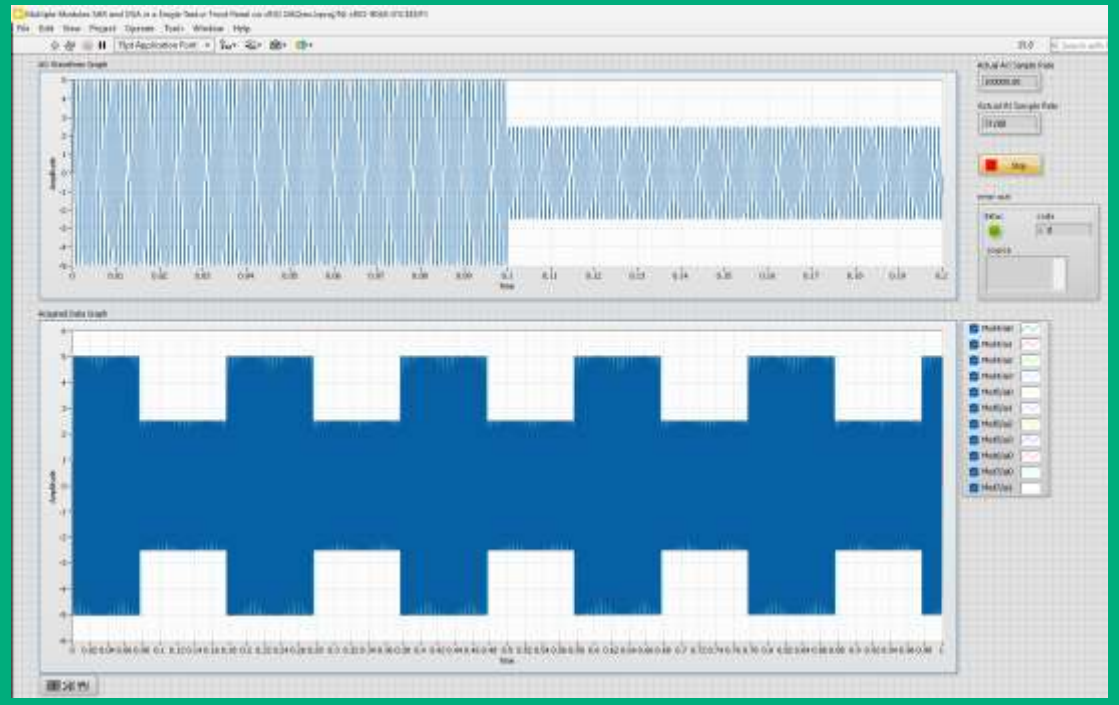


Quick getting started experience

Multiple modules synchronization

Access to all things DAQmx

- NI MAX
- Calibration
- Scaling
- etc



Demo: DAQmx Acquisition Synchronization

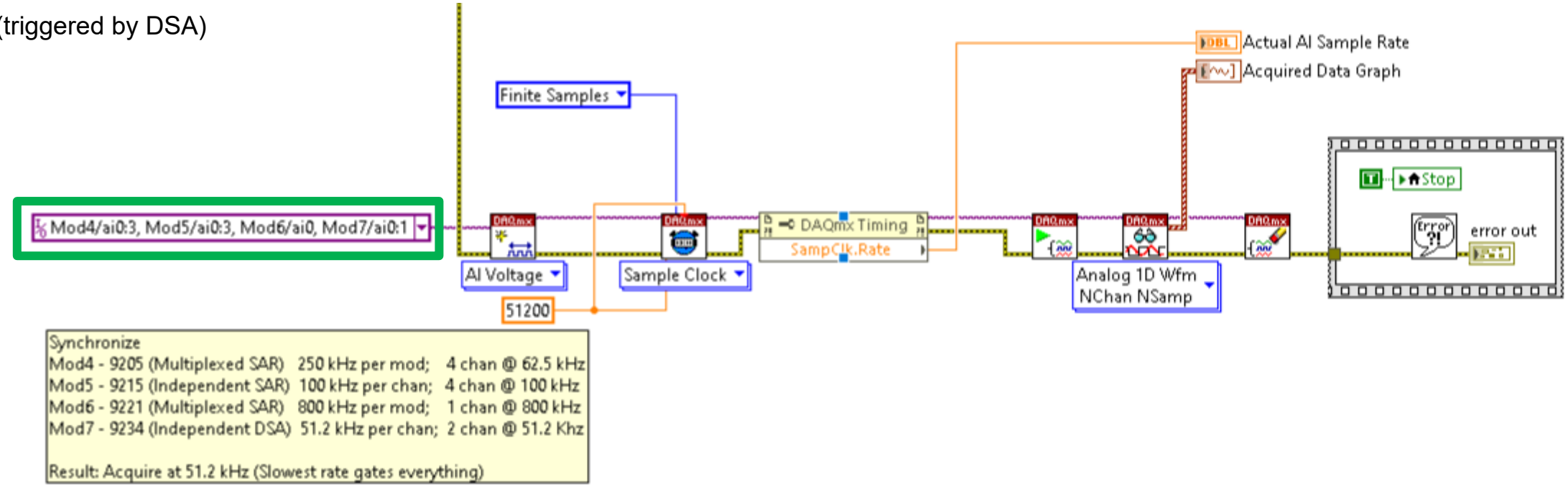
“Automatic” local synchronization

Multiple ADC types synchronized in a single task

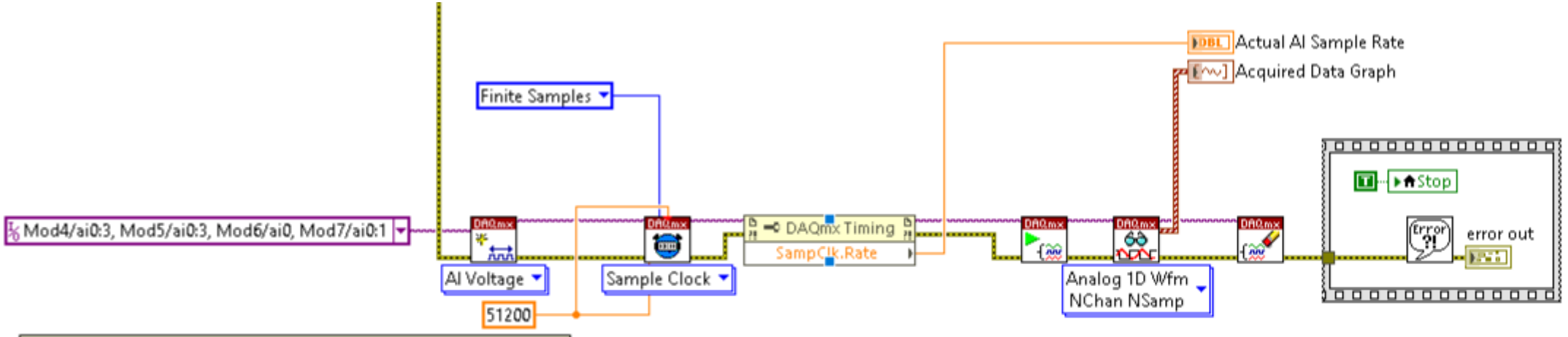
DSA (Used as master timing)

Independent SARs (triggered by DSA)

Muxed SARs (triggered by DSA)



Local Synchronization Test

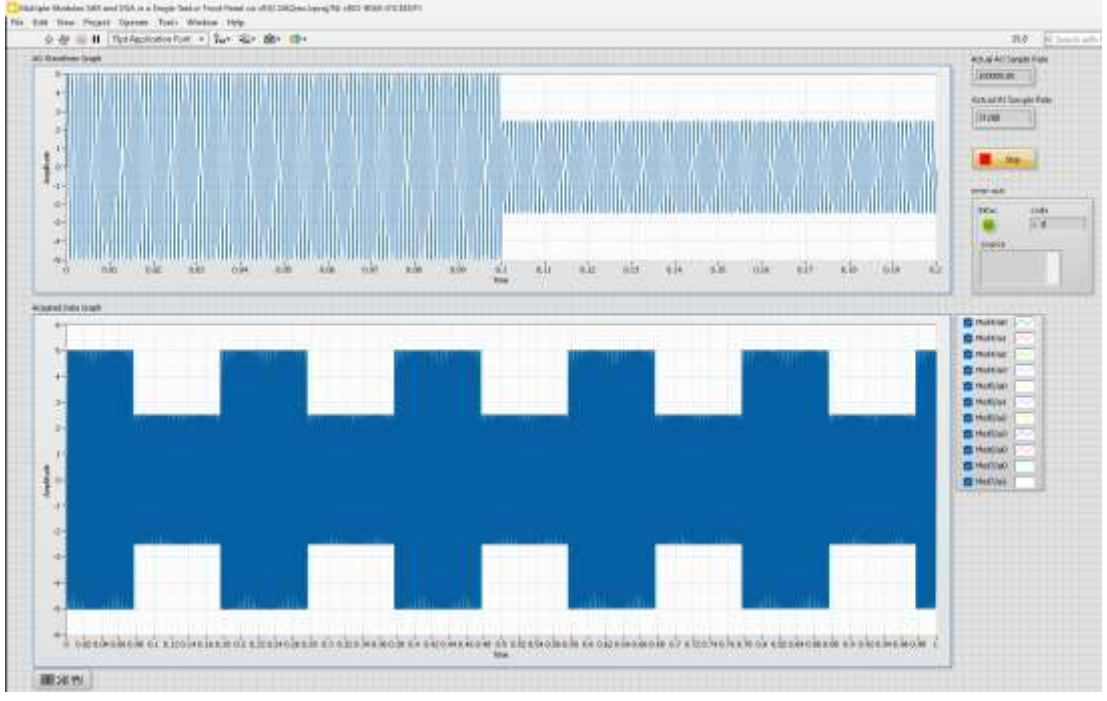


Synchronize
Mod4 - 9205 (Multiplexed SAR) 250 kHz per mod; 4 chan @ 62.5 kHz
Mod5 - 9215 (Independent SAR) 100 kHz per chan; 4 chan @ 100 kHz
Mod6 - 9221 (Multiplexed SAR) 800 kHz per mod; 1 chan @ 800 kHz
Mod7 - 9234 (Independent DSA) 51.2 kHz per chan; 2 chan @ 51.2 Khz

Result: Acquire at 51.2 kHz (Slowest rate gates everything)

1 kHz Sine Gen @ 100 kHz

51.2 kHz Acq



Demo: DAQmx Acquisition Synchronization

Steps

Open LabVIEW Project @ cRIO Intro → cRIO Example Code → DAQmx → Multiple ADC Types Synchronization Demo

Connect to the cRIO

Deploy the Chassis settings

Open SAR and DSA Module Synchronization Demo.vi

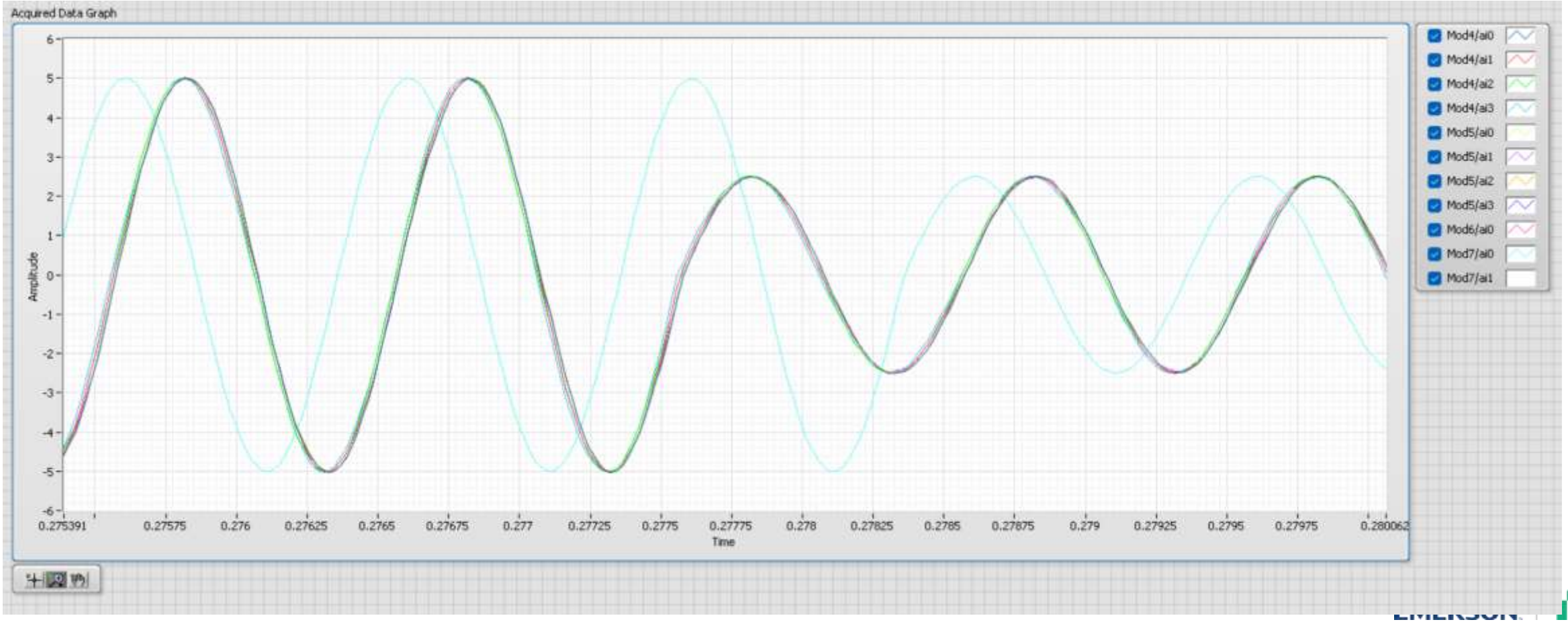
Go to the Block Diagram; Highlight

- 1kHz sine wave generation @ 100 kHz
- Acquisition @ 51200 Hz on Muxed SAR (9205), Independent SAR (9215), & DSA (9234)
- 9205 could go 100 kHz, 9205 could go 62.5 kHz → Synched to DSA at 51.2 kHz
- Logging on cRIO (locally on Linux RT)

Run Example

Review waveforms in slide form below

All Modules Together

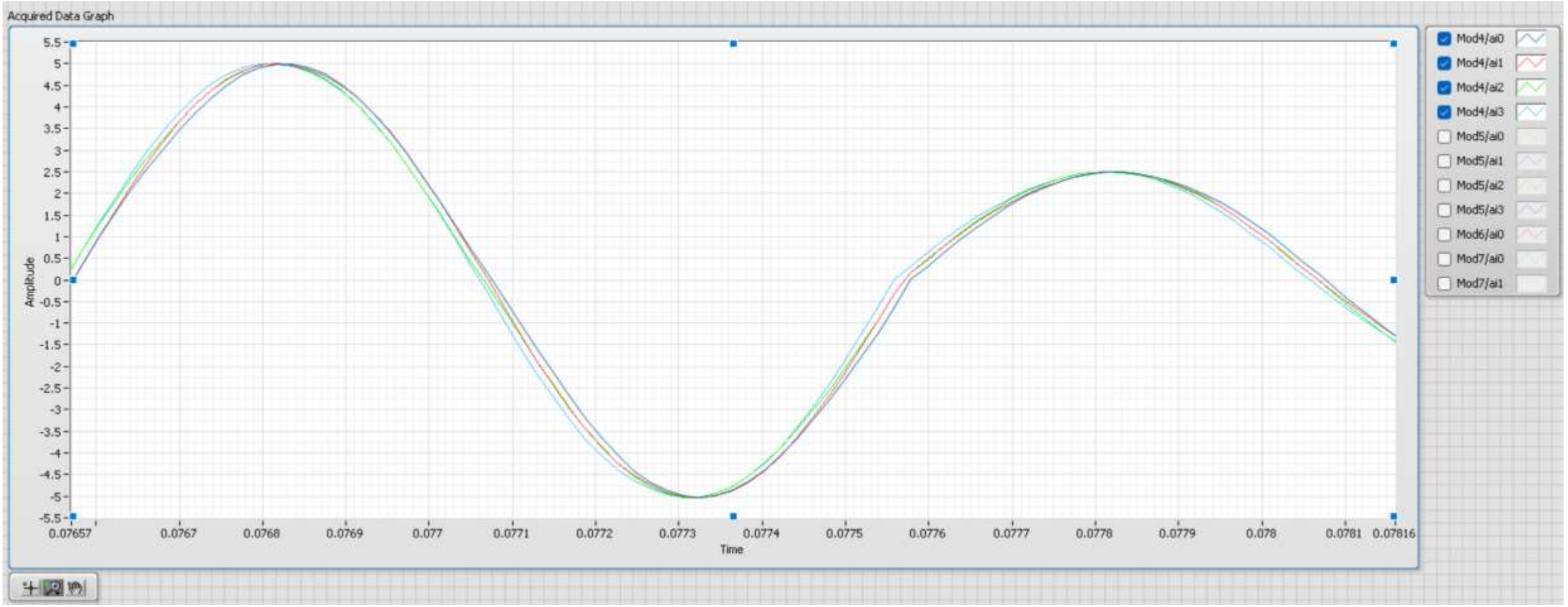


Multiplexed SAR Comparison (NI 9205 ai0:3)

Table 4. Conversion Time (Maximum Sampling Rate)

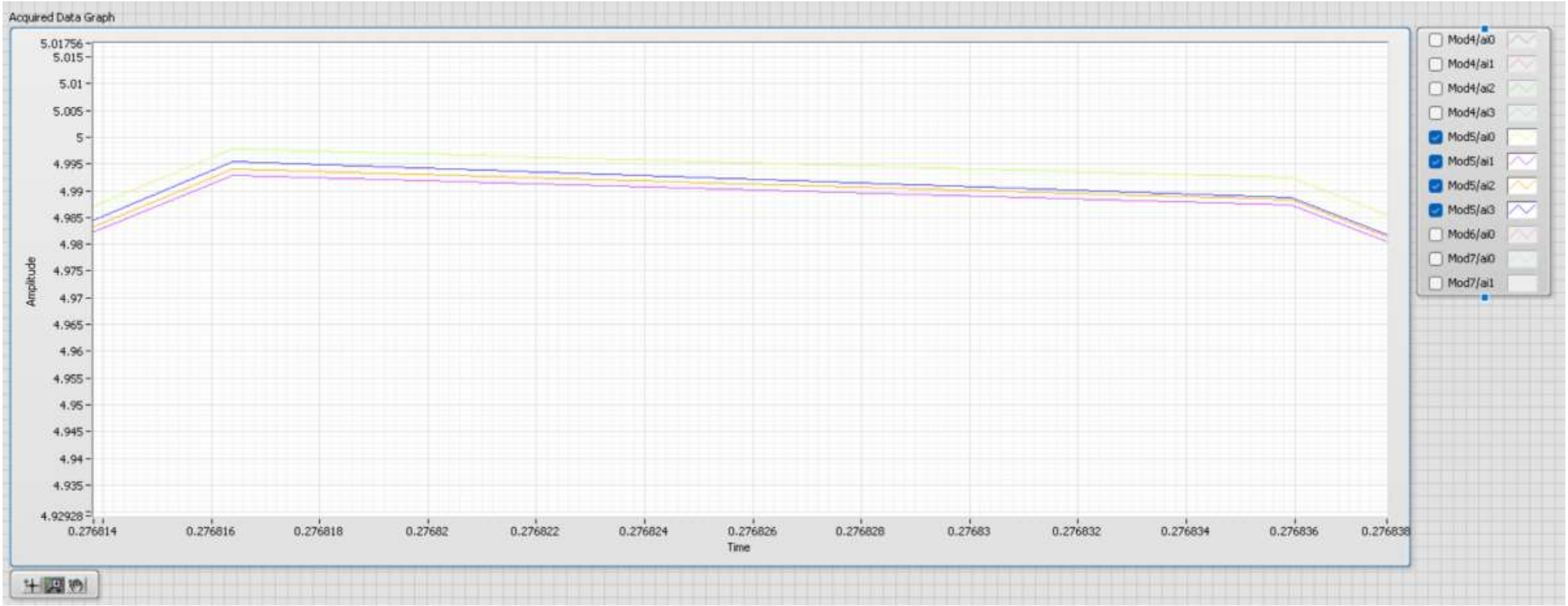
Chassis Type	Conversion Time
CompactRIO and CompactDAQ chassis	4.00 µs (250 kS/s)
R Series Expansion chassis	4.50 µs (222 kS/s)

Each channel “shifts” by conversion time (4 us)



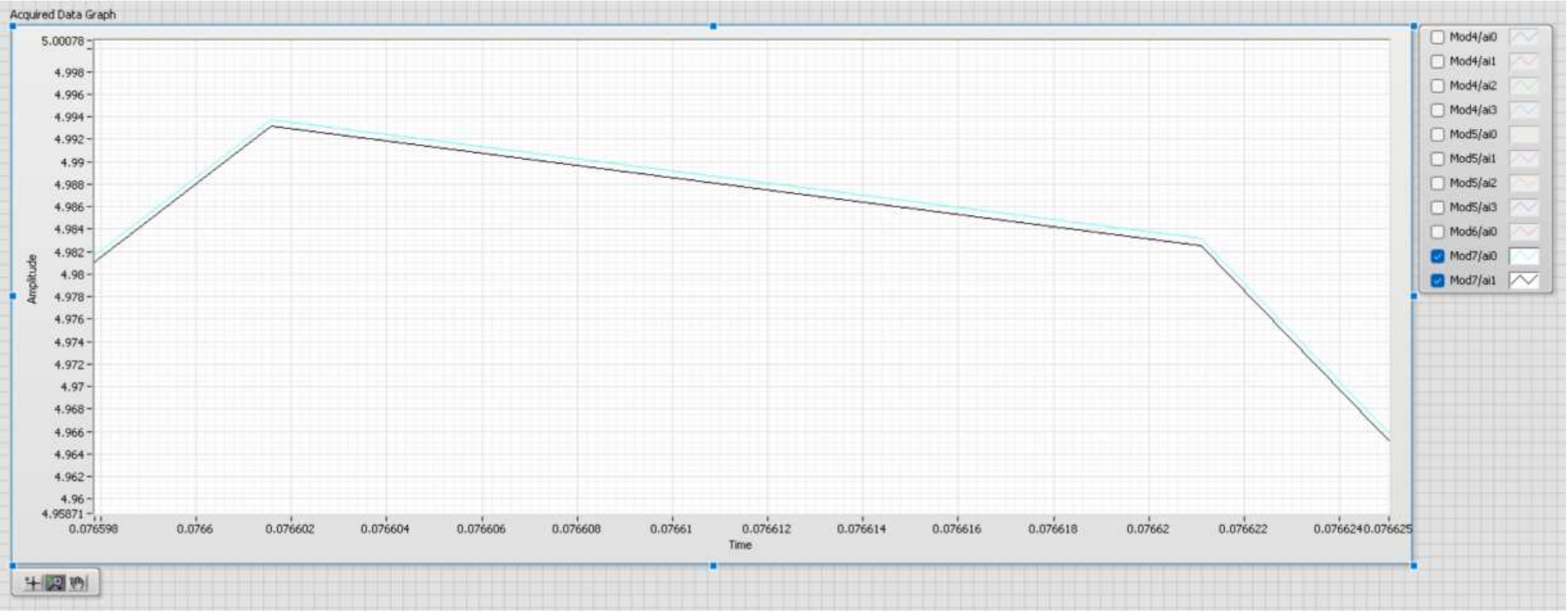
Independent SAR Comparison (NI 9215 ai0:3)

All channels perfectly synchronized in time



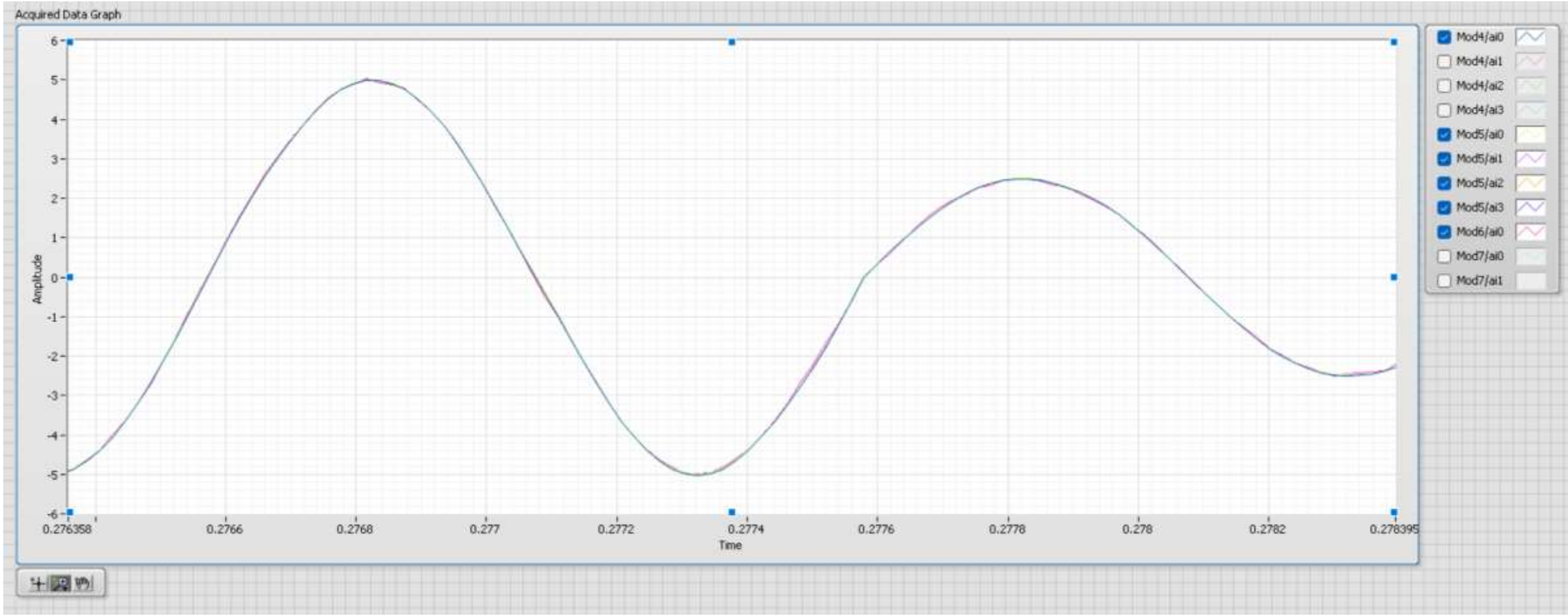
Independent DSA Comparison (NI 9234 ai0:1)

All channels perfectly synchronized in time



Independent & First Channel Muxed SAR Comparison (NI 9205 ai0 + NI 9215 ai0:3 + NI 9221 ai0)

All three modules perfectly synchronized in time



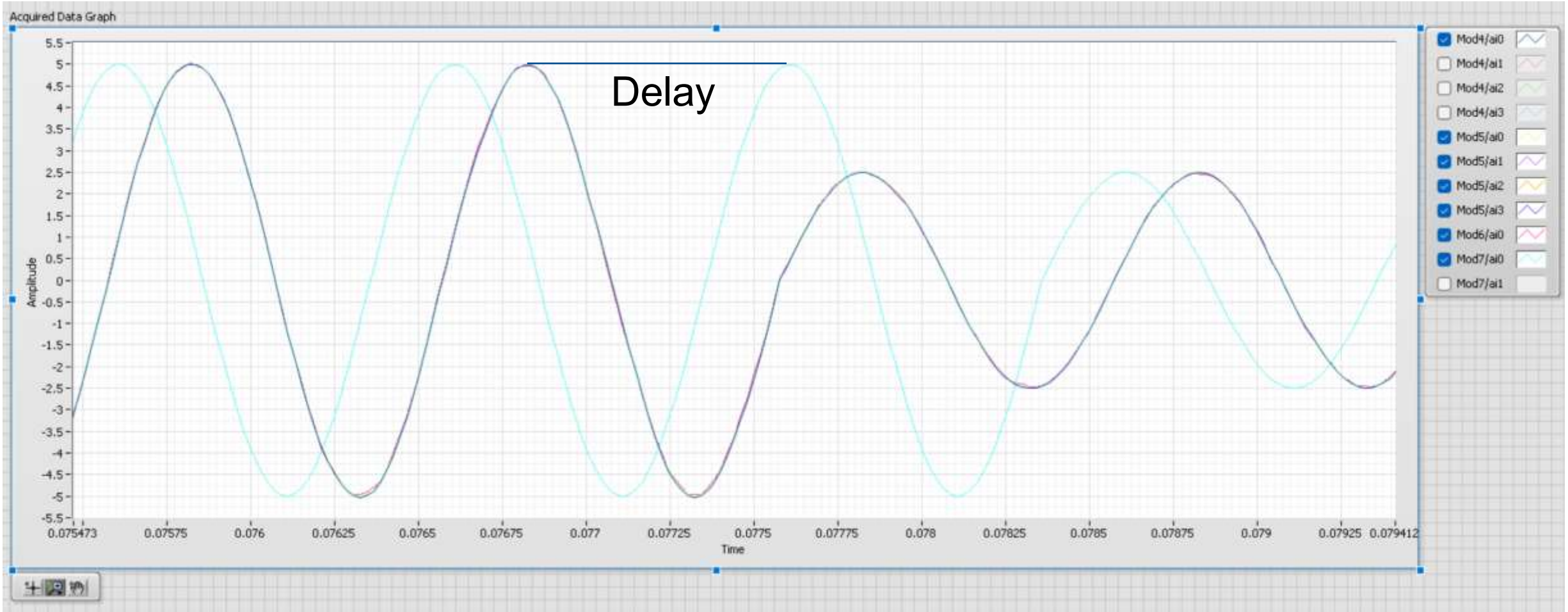
Independent & First Channel Muxed SAR Comparison vs DSA (NI 9205 ai0 + NI 9215 ai0:3 + NI 9221 ai0 + NI 9234 ai0)

Input delay	$(40 + 13/512)/f_s + 2.6\ \mu s$

=(40+13/512)/51200+0.0000026

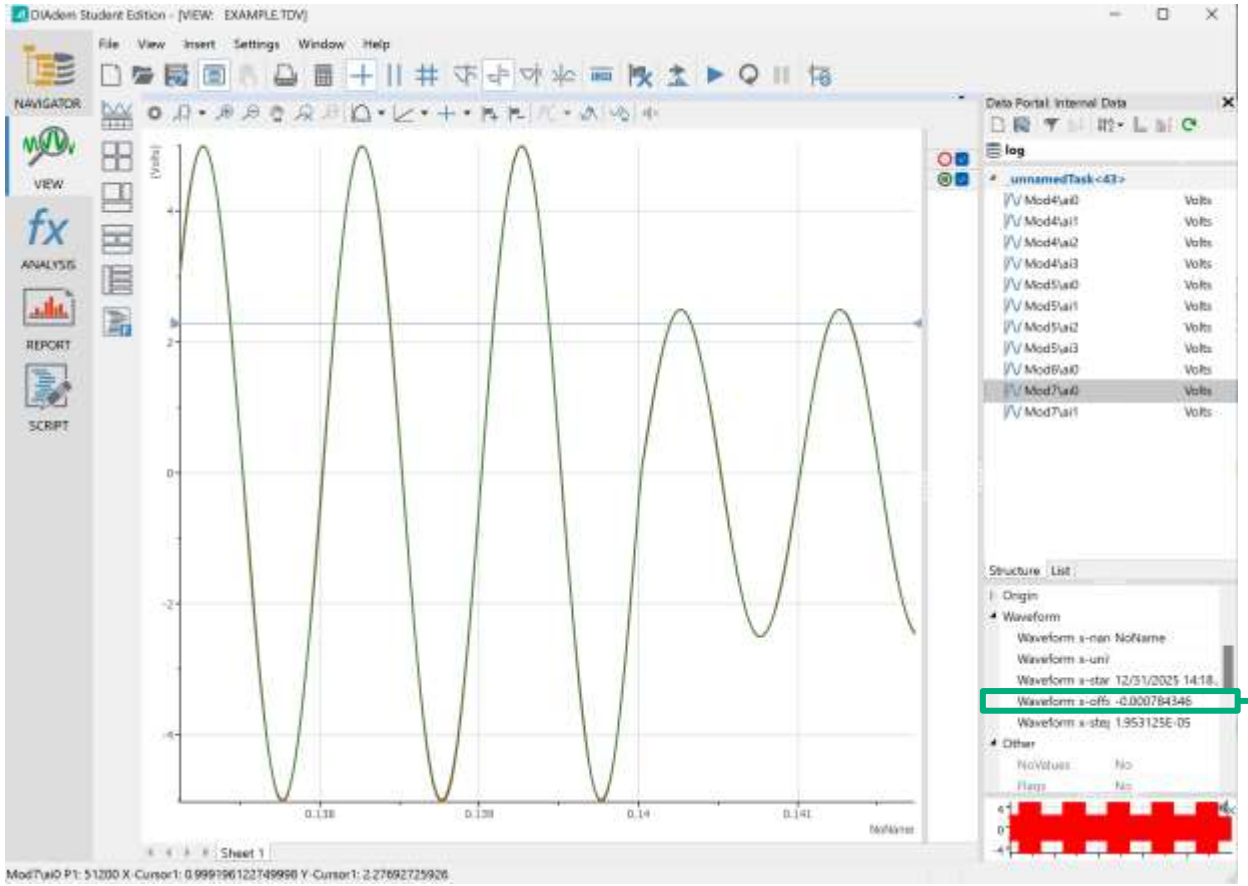
	A	B	C	D	E
1	0.000784346	seconds			

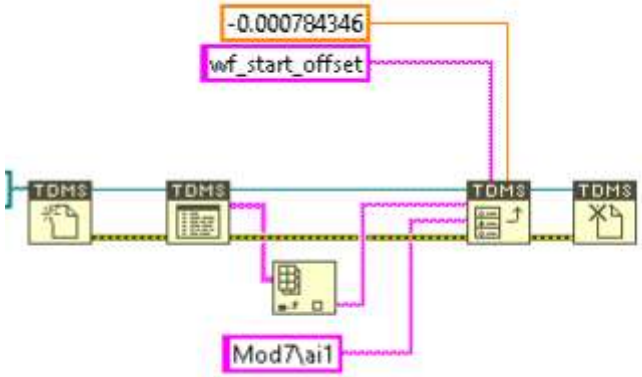
Phase shift due to Delta Sigma Chip

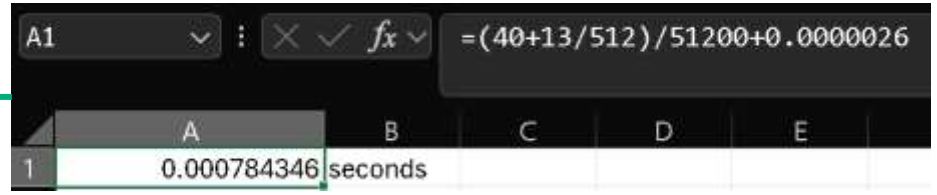


Correcting DSA Samples (NI 9205 ai0 + NI 9234 ai0)

Apply Waveform x-offset







Correcting DSA Samples (NI 9205 ai0 + NI 9234 ai0)

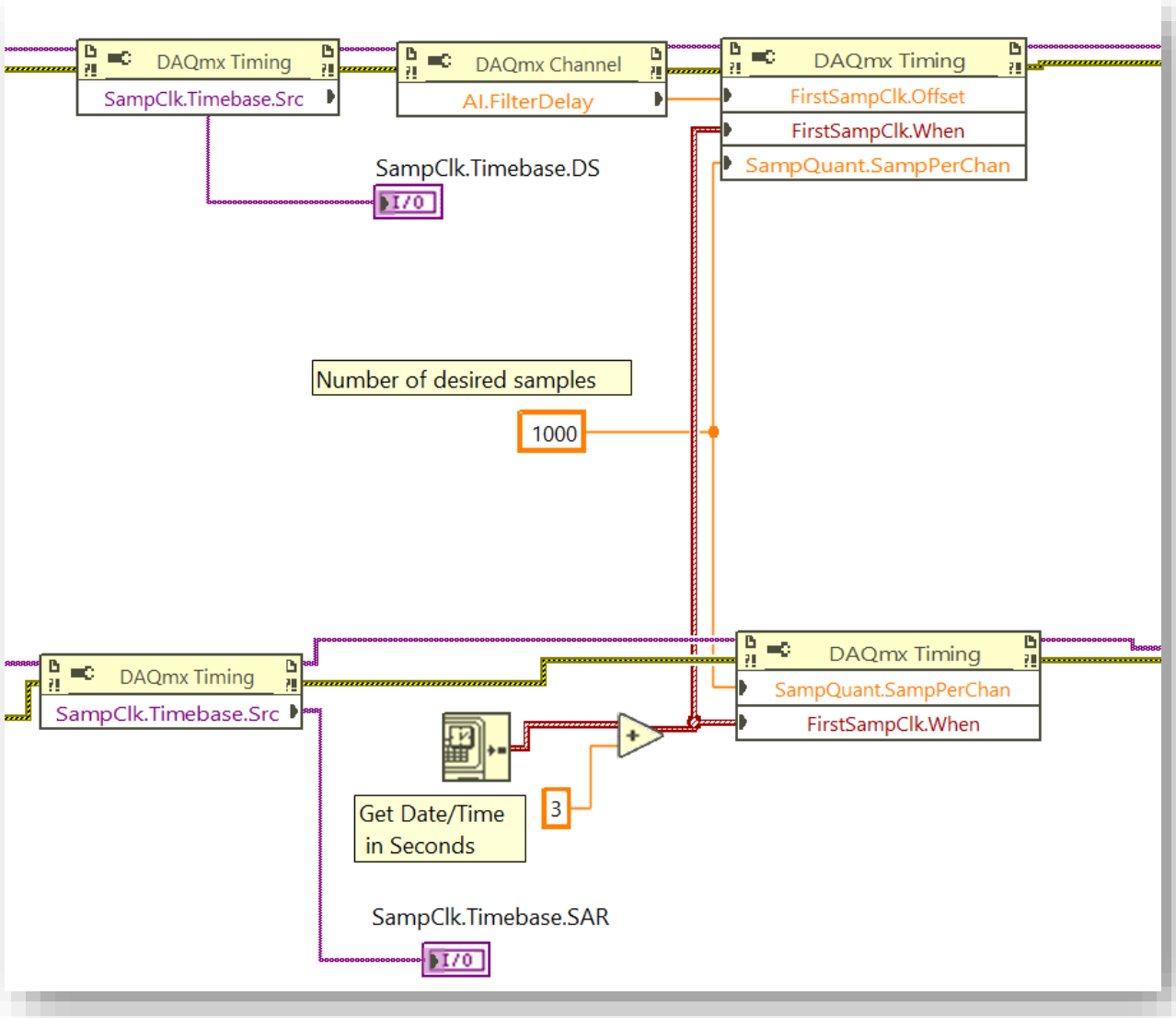
Live Option

Keep Delta Sigma Modules in separate task

Synchronize start time (FirstSampleClk.When)

On Delta Sigma, Get and Apply Filter Delay

- AI.FilterDelay → FirstSampleClk.Offset



Correcting Mux Samples…

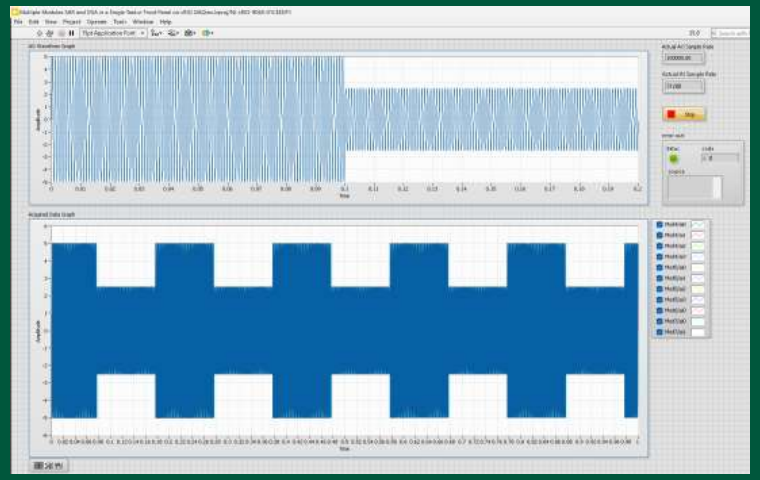
Pick a different module…

Technically might be able to resample…

With Delta Sigma, you latch the sample on the sample clock

With a Muxed module, you sample after the sample clock (ChNumber * ConvertTime – ConvertTime)

Summary: DAQmx Acquisition Synchronization



DAQmx synchronizes modules in a single task

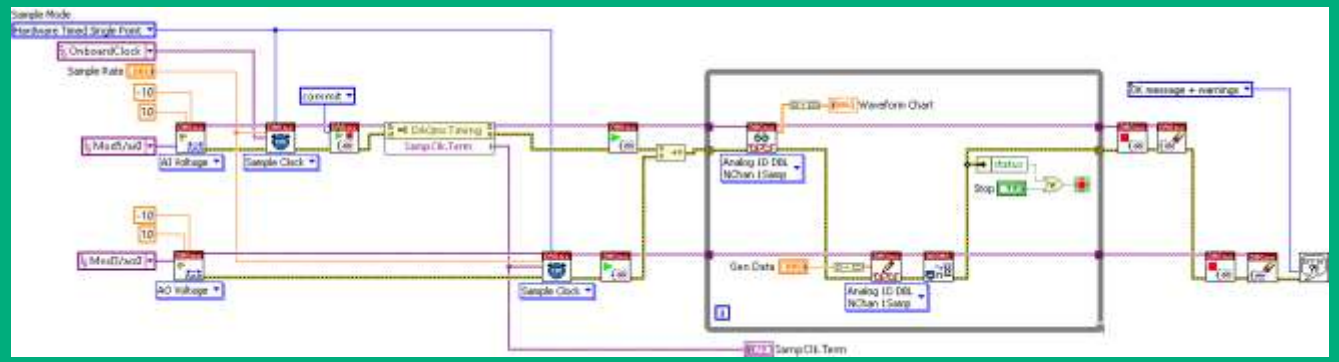
- Regardless of frequency
- Regardless of ADC type

DSA input delay is real

- Select modules accordingly
- Fix live or in post

Muxed modules = cheap channel count; not synchronization

- Select modules accordingly



Demo: DAQmx Synchronized Control Demo

Demo: DAQmx Synchronized Control

Steps

Open LabVIEW Project @ cRIO Intro → cRIO Example Code → DAQmx → Synchronized Control Demo

Connect to the cRIO

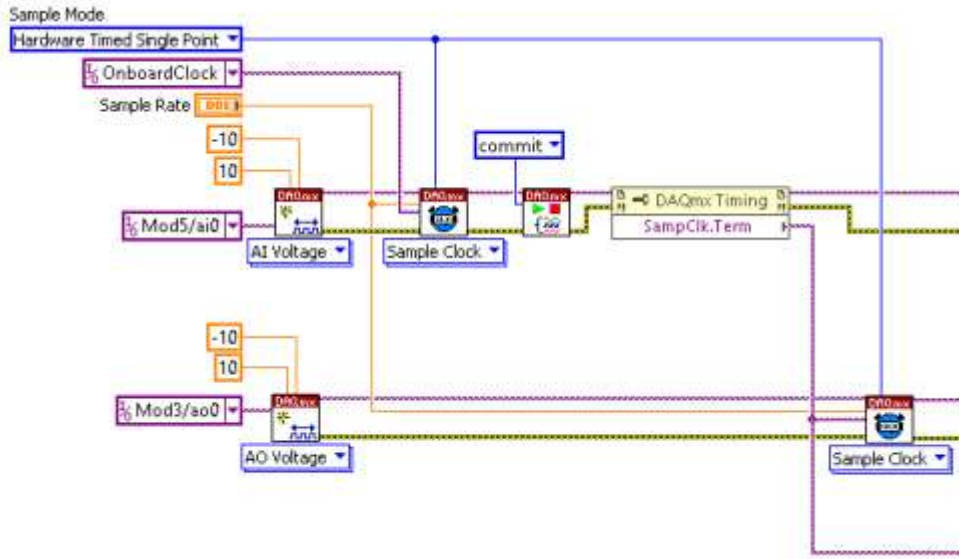
Deploy the Chassis settings

Open DAQmx Synchronized Control.vi

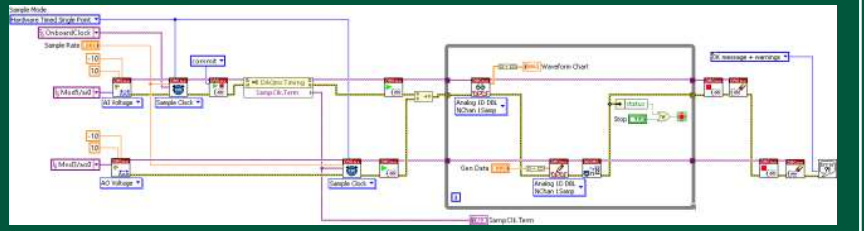
Go to the Block Diagram; Highlight

- Setting sample clock to Hardware Timed Single Point
- Committing one task (reserves a DAQ engine)
- Getting SampleClk.Term from DAQ engine
- Using SampleClk.Term as second task clock (AI & AO hardware synched)
- Reading / writing N Channels 1 Sample (single point control)
- Wait for Next Sample Clock (Loop timed to sample clock)

Run Example



Summary: DAQmx Synchronized Control Demo



DAQmx allows multiple task synchronization

- Shared sample clocks

Hardware timed single point is supported on cRIO

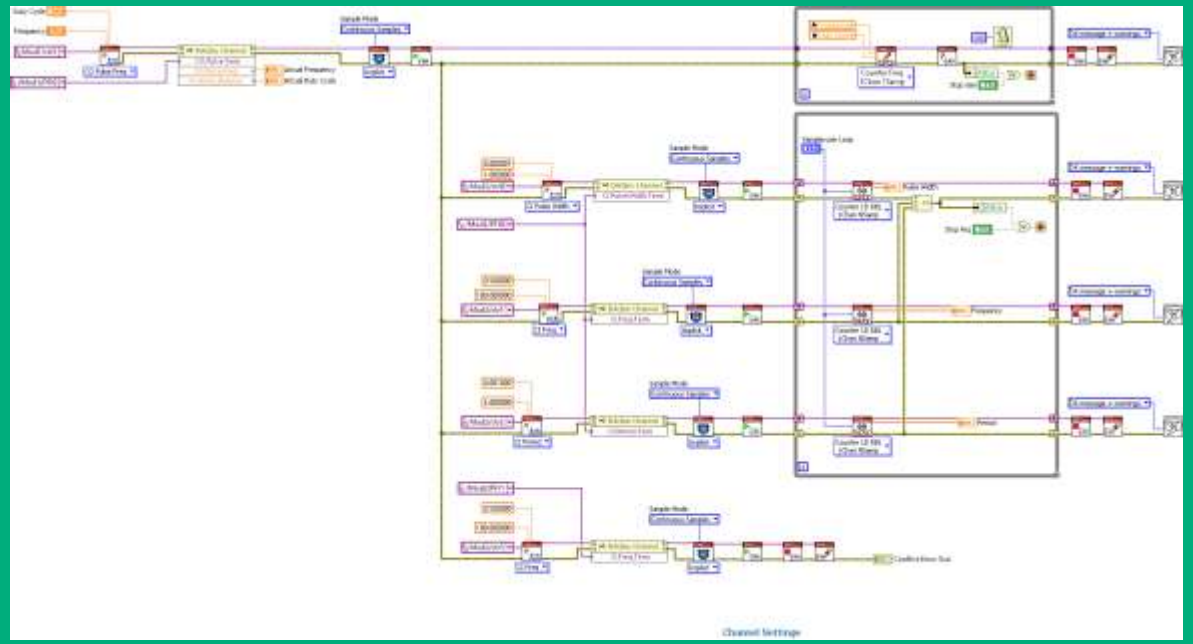
- Single sample per hardware clock tick

DAQmx provides timing function to keep in synch

- Wait until Next Sample Clock

Great for easy, slower speed control

- Up to 500 Hz; (marketing says faster…)



Demo: NI DAQmx Counters

(Core Weakness…)

NI DAQmx Counters on cRIO

Four Counters Total

Inputs can be augmented with:

- NI 9326 – 6 Embedded Frequency Inputs
- NI 9361 – 8 Embedded Counter Inputs

No additional output options

Measurements are not optimized for PWMs

- Read Frequency OR Duty Cycle (not both on the same channel)

Personal Recommendation: Use FPGA programming for PWMs…

Demo: DAQmx Counters

Steps

Open LabVIEW Project @ cRIO Intro → cRIO Example Code → DAQmx → Multiple Counters Demo

Connect to the cRIO

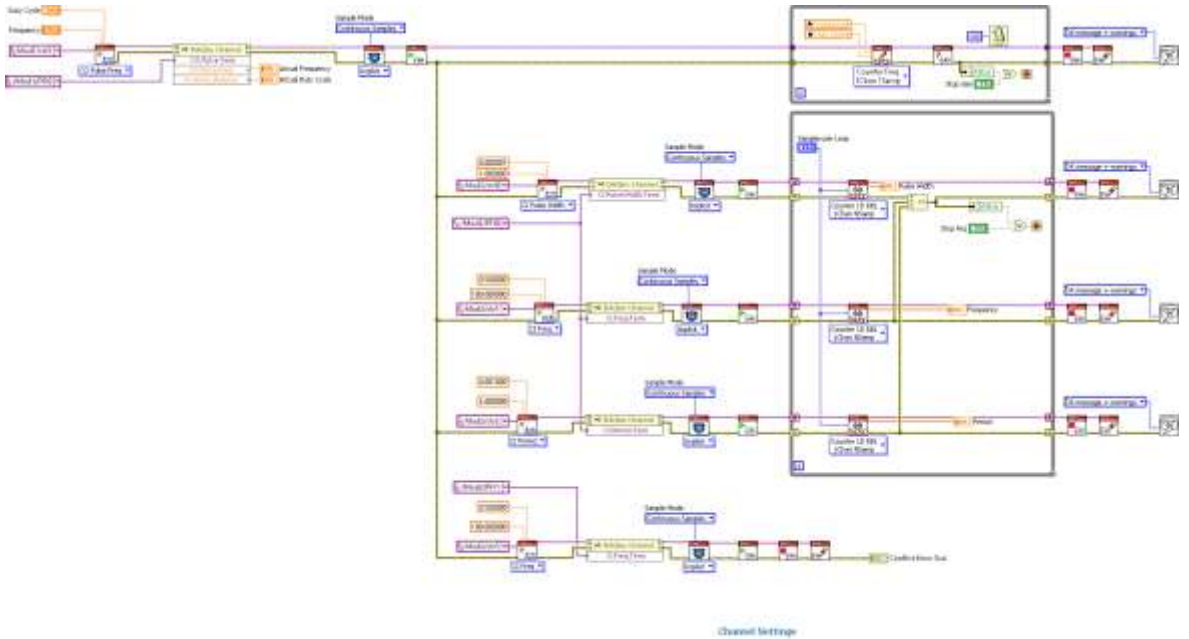
Deploy the Chassis settings

Open DAQmx Counters Control.vi

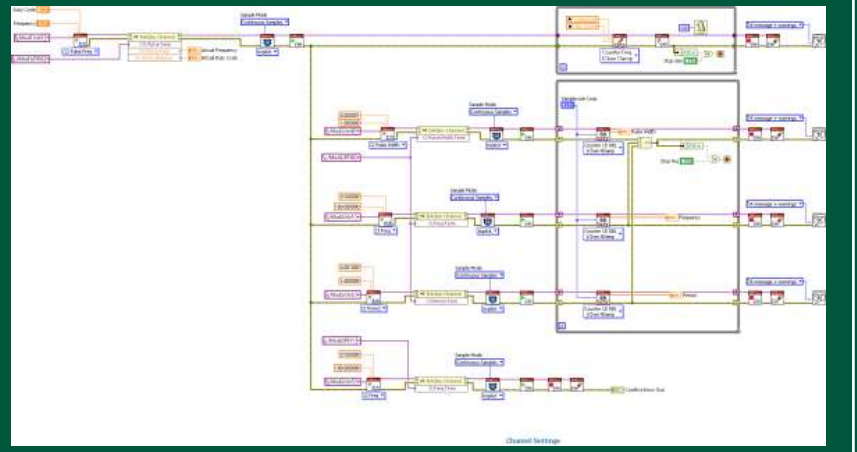
Go to the Block Diagram; Highlight

- Generating on one channel; acquiring on four channels
- Pulse Width, Frequency, & Period all on same PFI line; takes 3 counters
- 5th counter will error (proving only 4 counters)
- Counter naming (Mod/ctr#), but still only get 4

Run Example



Summary: DAQmx Counters Demo



DAQmx provides 4 counters to work with

One measurement per counter

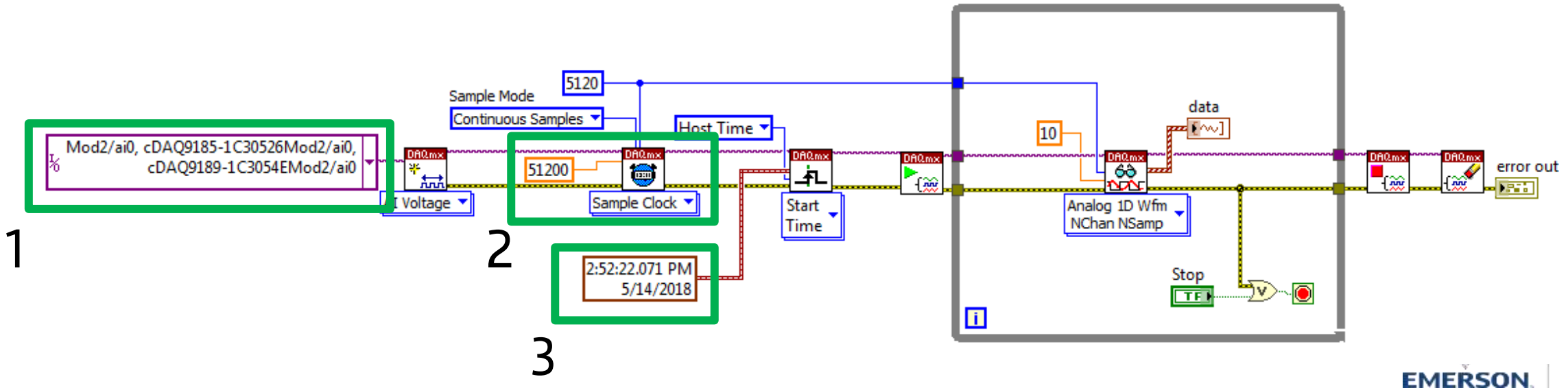
Input expansion possible with c series modules

Plan accordingly

Expand local IO w/ NI-DAQmx

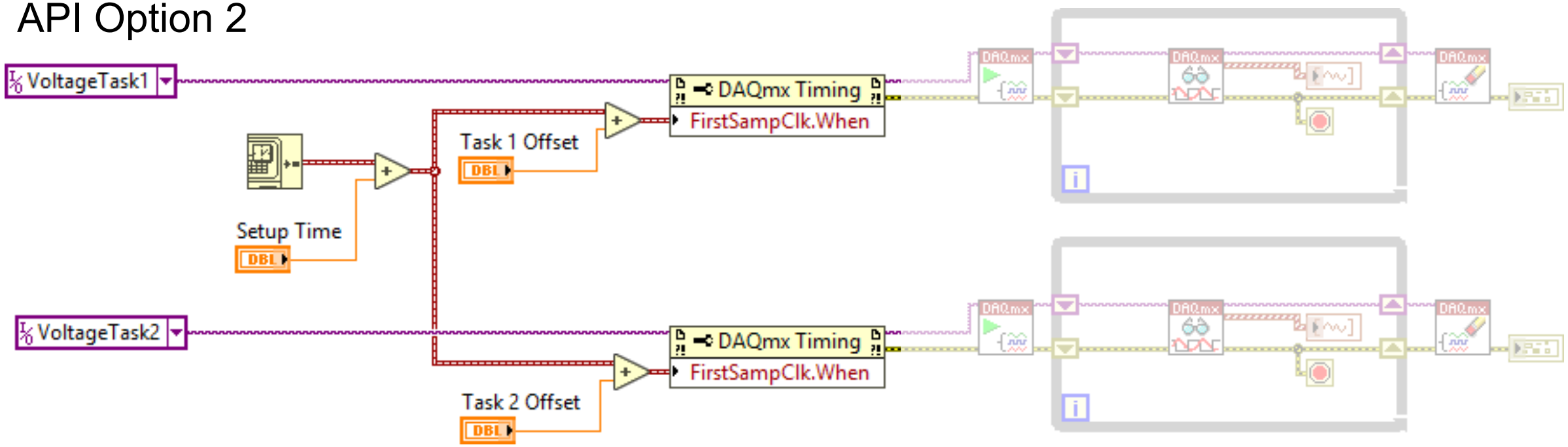
TSN Enabled cDAQ & FieldDAQ for Synchronized Expansion

1. Channel expansion (multi-device tasks)
2. Sample clock disciplined to network time
3. Ability to start a task at the same start time across synchronized devices



Expand local IO w/ NI-DAQmx

API Option 2



Expand remote processors & IO w/ NI-DAQmx

Drift and timestamps will automatically be handled

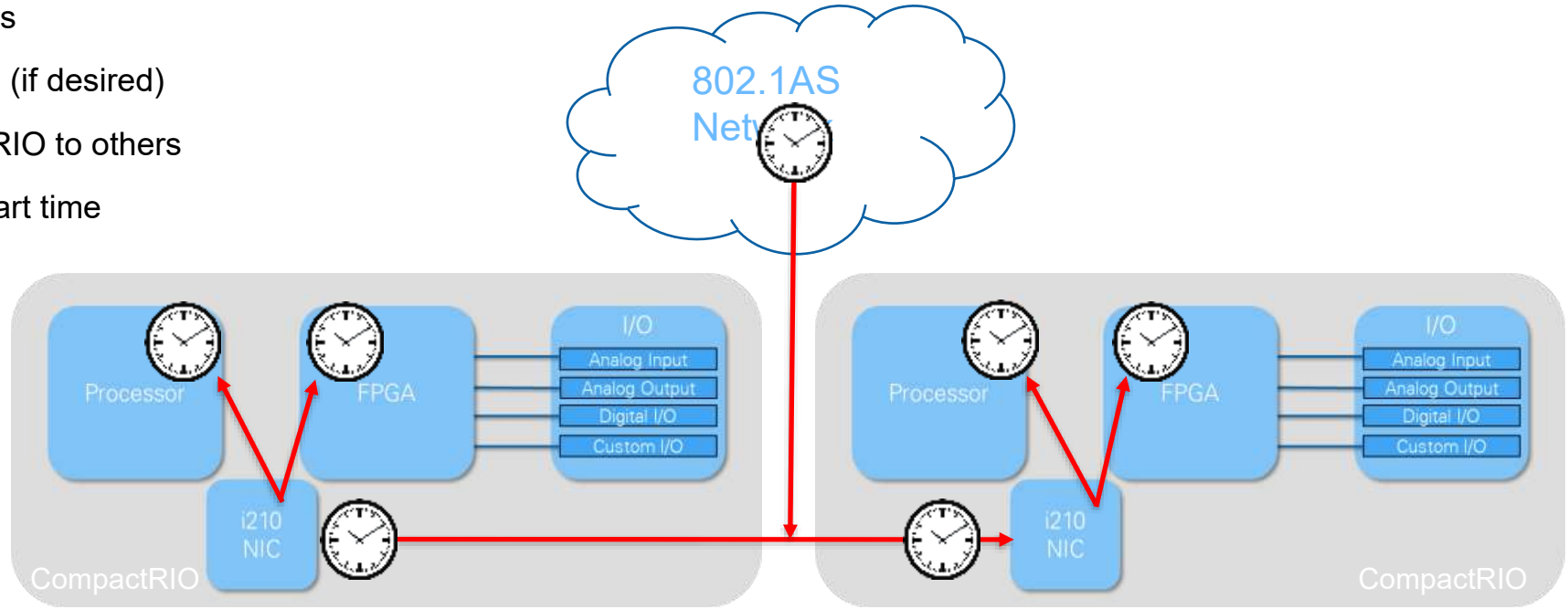
- TSN automatically synchronizes time
- TSN automatically disciplines local clocks

Use DAQmx timing to synchronize start time (if desired)

- Send “Start Time” command from one cRIO to others
- Use option 2 above to trigger DAQmx start time

Acquisition / logging still happens locally

- FTP files off
- Combine in Diadem



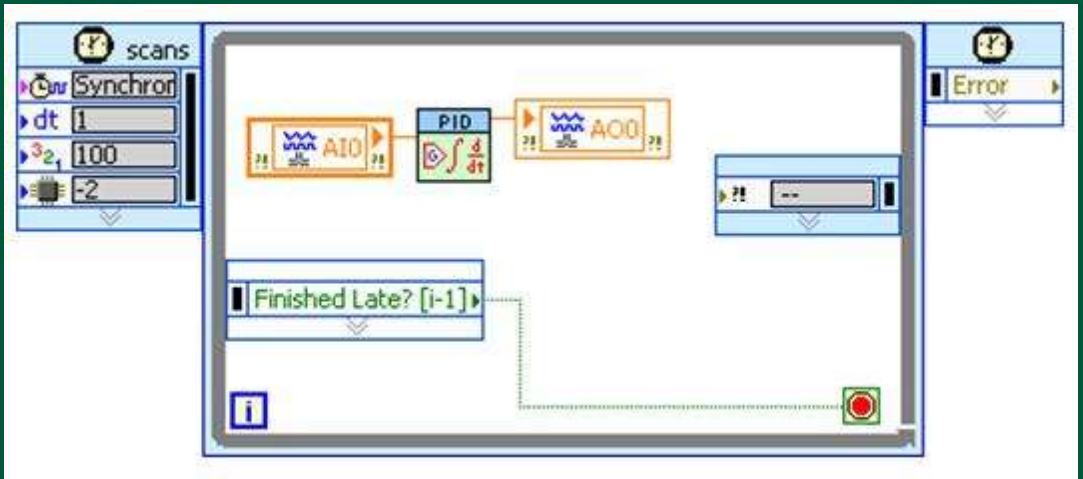
Summary: NI DAQmx IO on cRIO

Pros

- Ease of Use (Multiple Modules, Synchronization)
- Waveforms
- Basic Control
- Synchronized IO Expansion

Cons

- Counters
- Higher speed control



Scan Engine IO on cRIO

Better Understanding → Better Design

EMERSON

Scan Engine

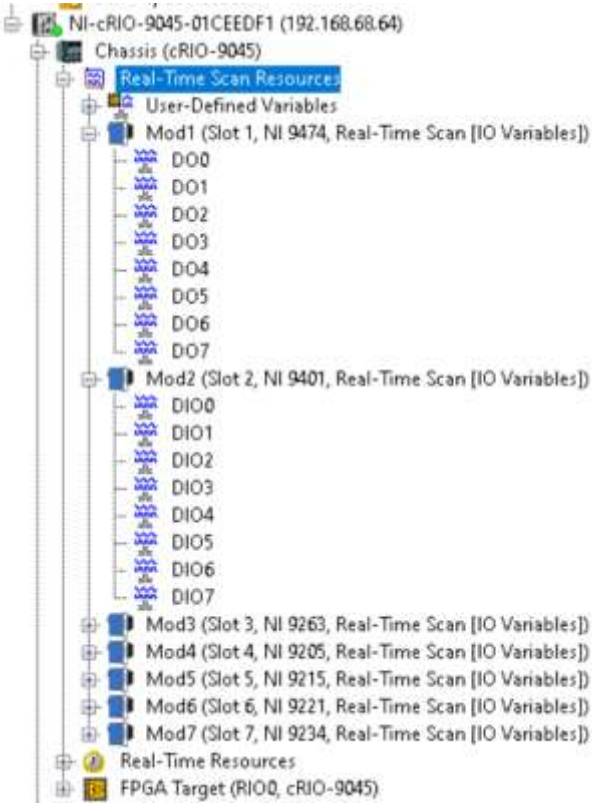
Simplified Control and Networking Experience

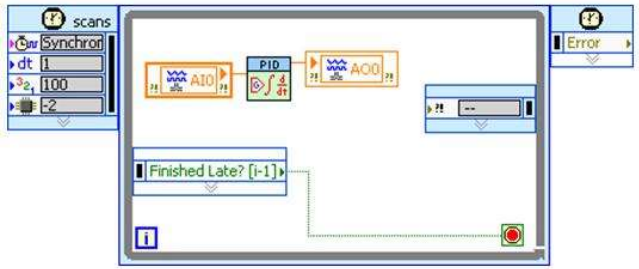
Single-Point data acquisition and control from LabVIEW Real-Time

Supports control loop rates up to 1 kHz

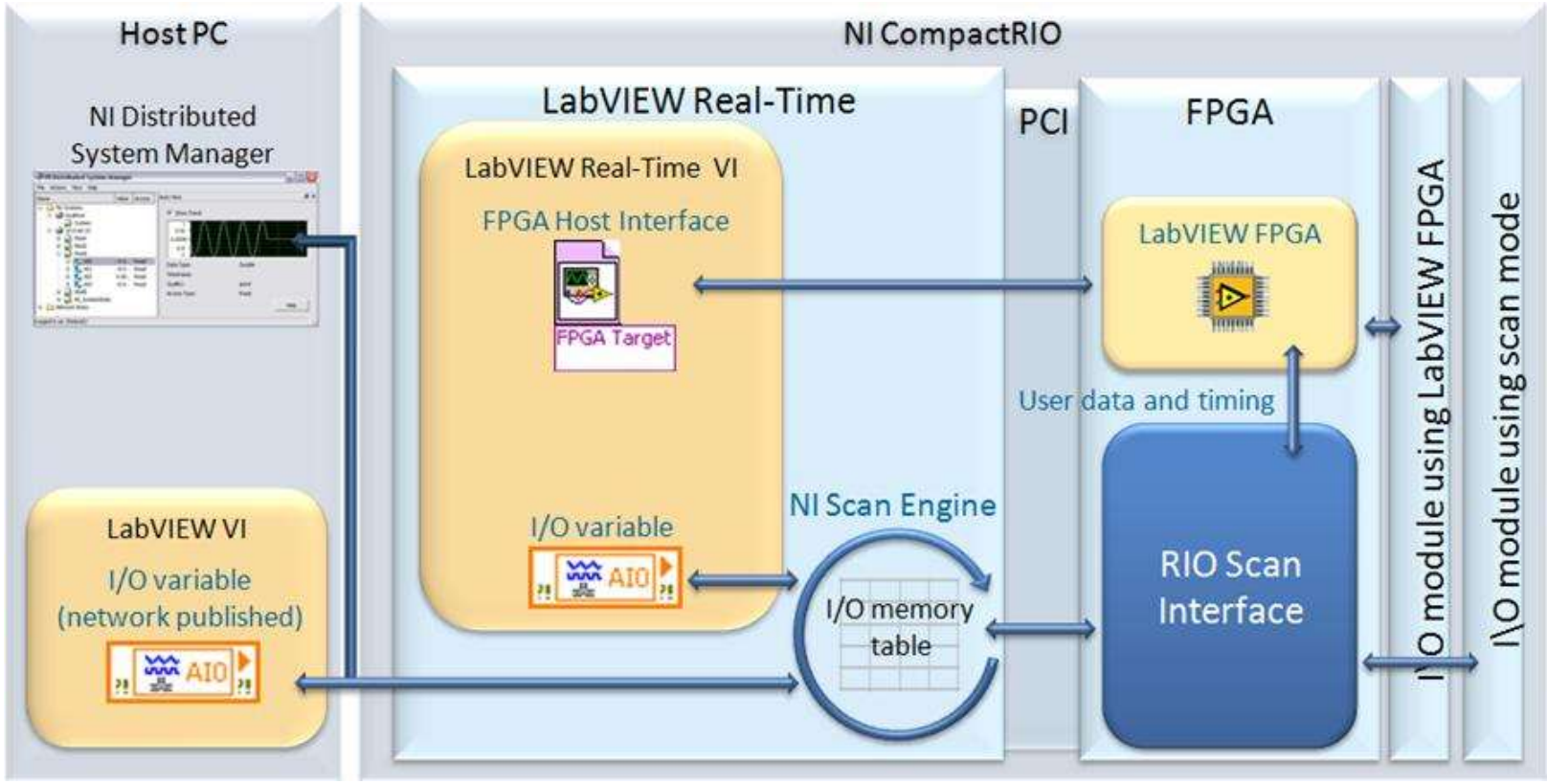
Expansion IO with NI 9145 (eCAT)

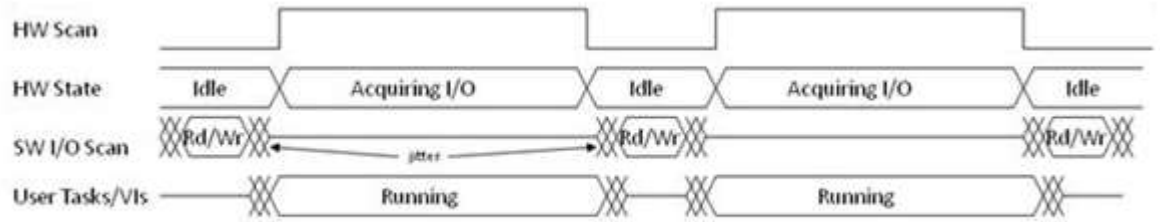
Variables can be automatically published via network





Scan Engine Architecture





Get the drivers right…

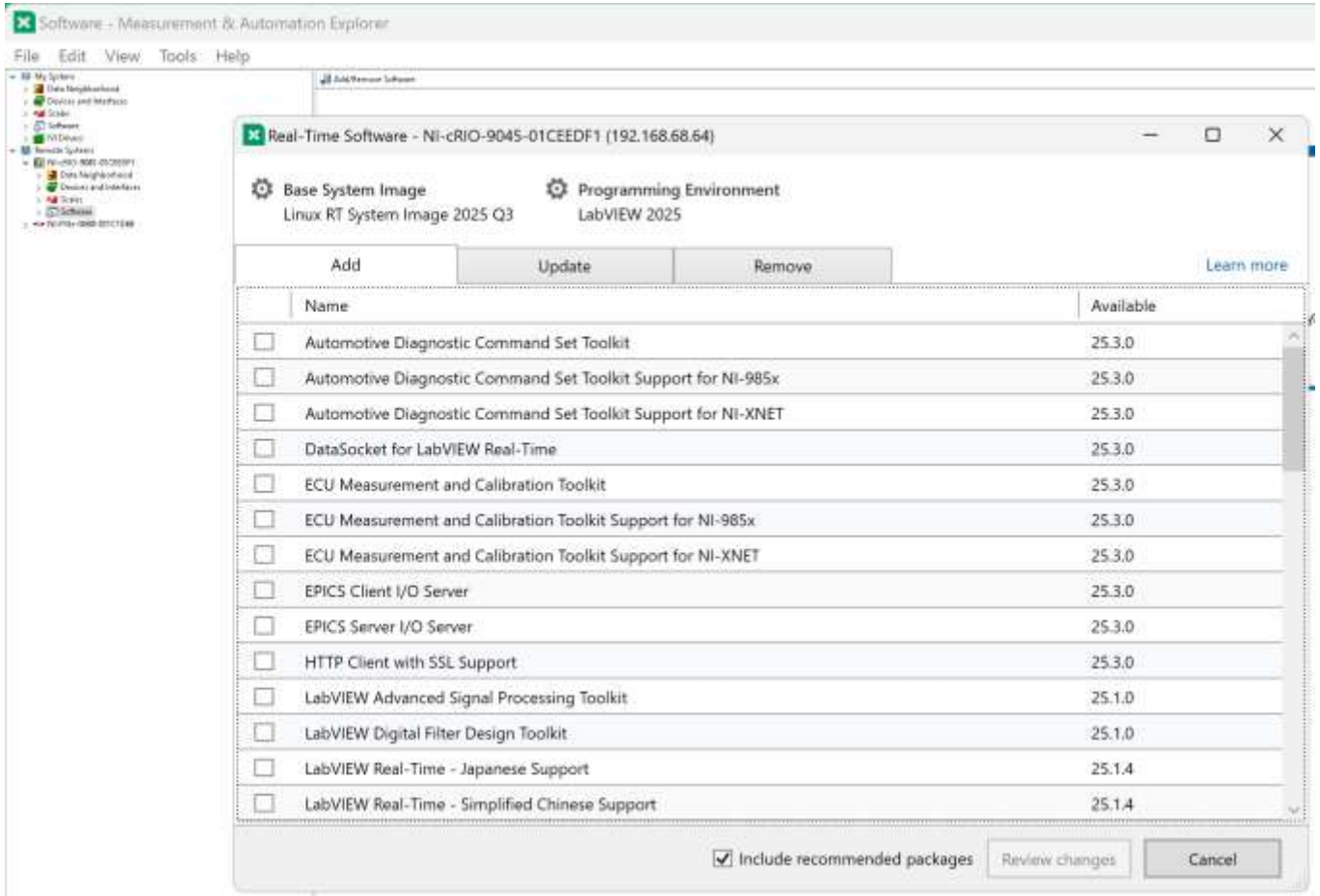
Host and cRIO driver version must match

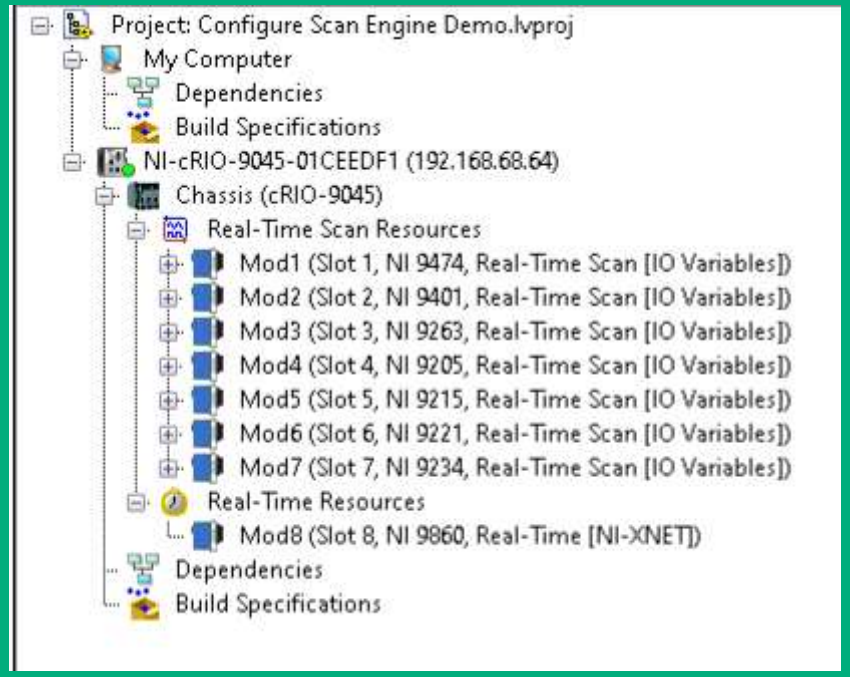
Check versions in MAX

Update as necessary

cRIO + Scan Engine requires:

- CompactRIO Support
- Network Variable Engine
- NI Scan Engine
- NI-Industrial Communications for EtherCAT
- NI-RIO / IO Scan / Server
- NI-Sync





Demo: Configuring cRIO for Scan Engine

Create the LabVIEW Project

Create new blank LabVIEW Project

- Create Project → Blank Project

Add cRIO

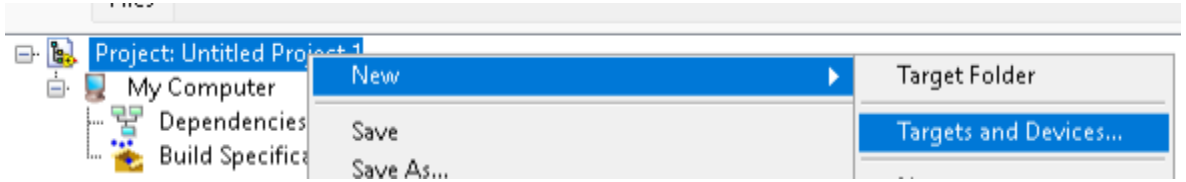
- Right click on Project
- New → Target and Devices

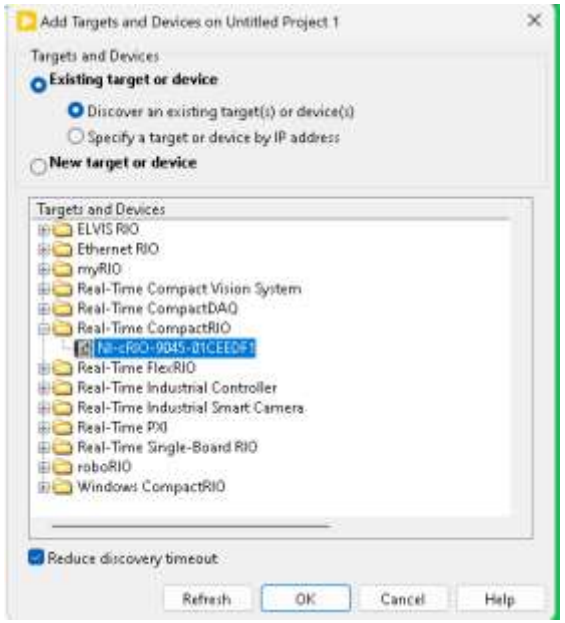
Select your cRIO from Real-Time CompactRIO

- Computer and cRIO must be on same subnet
- Modules will automatically be added with default settings

Delete all modules from:

- Real-Time Resources
- FPGA Target





Configure the Controller

Make sure connected to cRIO

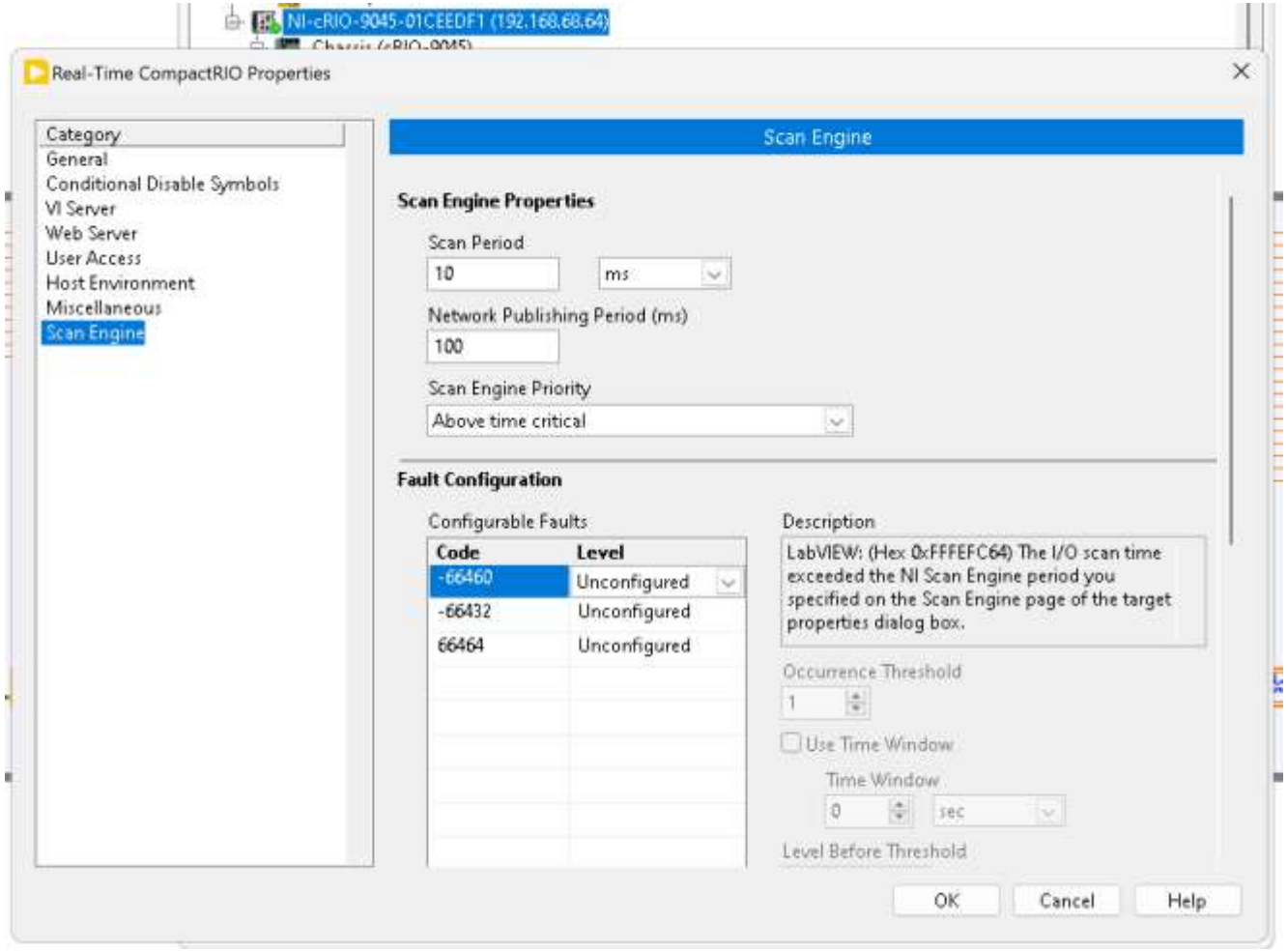
- Right click on cRIO
- Connect

Configure the Scan Engine

- Right click on cRIO
- Properties
- Select Scan Engine from list box
- Configure the Scan Period (IO rate)
- Configure the Network Period

Deploy settings from cRIO

- Right click on cRIO
- Deploy All



Configure the Chassis

Make sure connected to cRIO

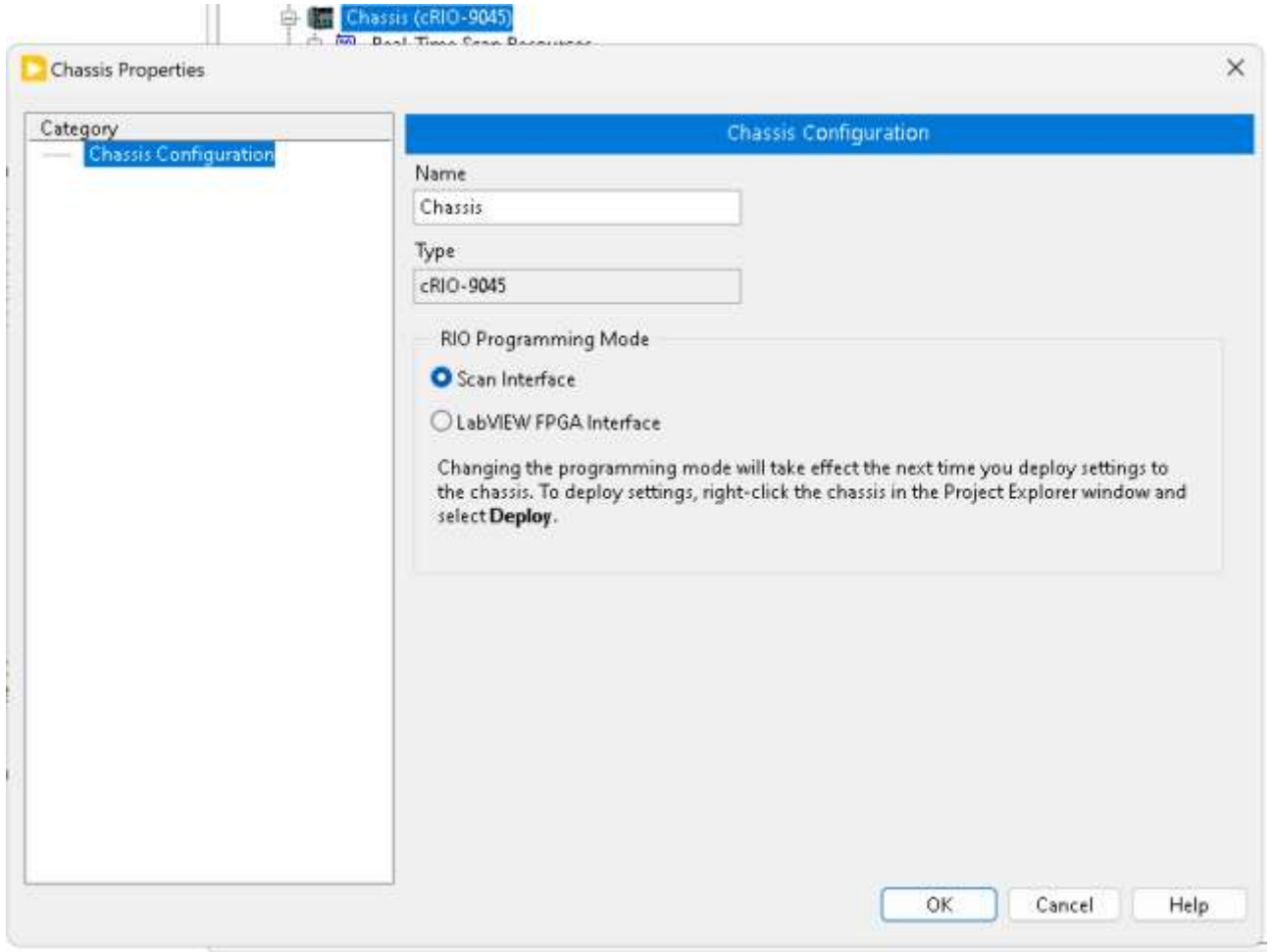
- Right click on cRIO
- Connect

Set RIO Programming Mode to Scan Interface

- Right click on Chassis
- Properties
- Set Mode to Scan Interface

Deploy settings from Chassis

- Right click on Chassis
- Deploy All



Add Modules to Real Time Scan Resources

Make sure connected to cRIO

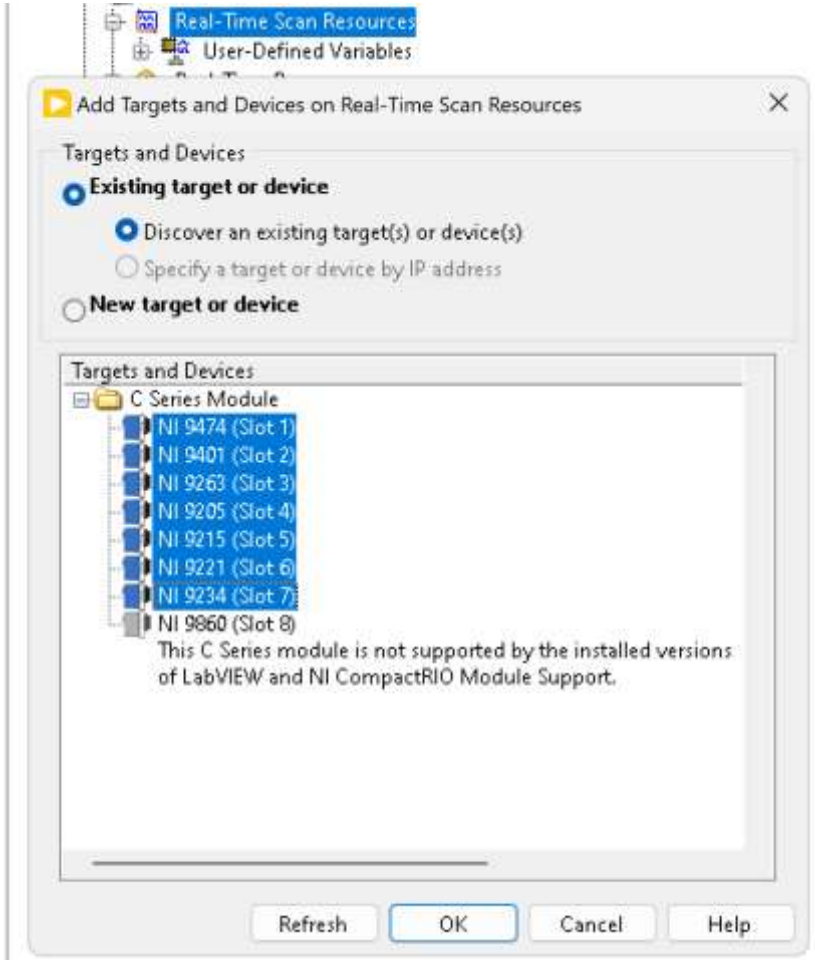
- Right click on cRIO
- Connect

Add modules to Real-Time Scan Resources

- Right click on Real-Time Scan Resources
- New → C Series Modules
- Select desired modules from GUI
- Select OK

Deploy settings from Chassis

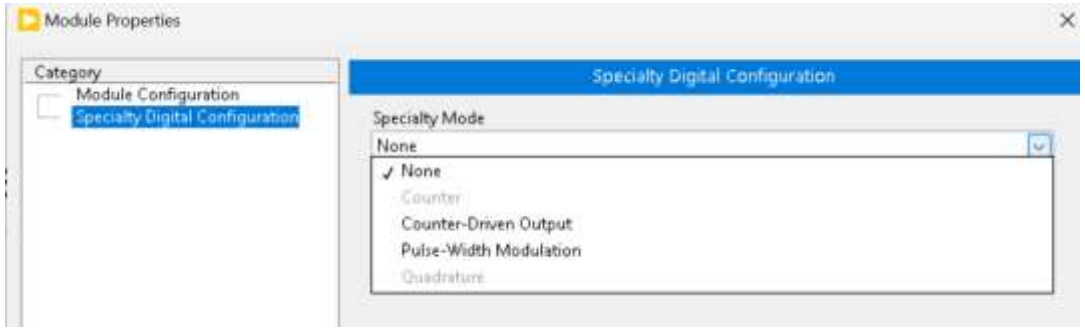
- Right click on Chassis
- Deploy All



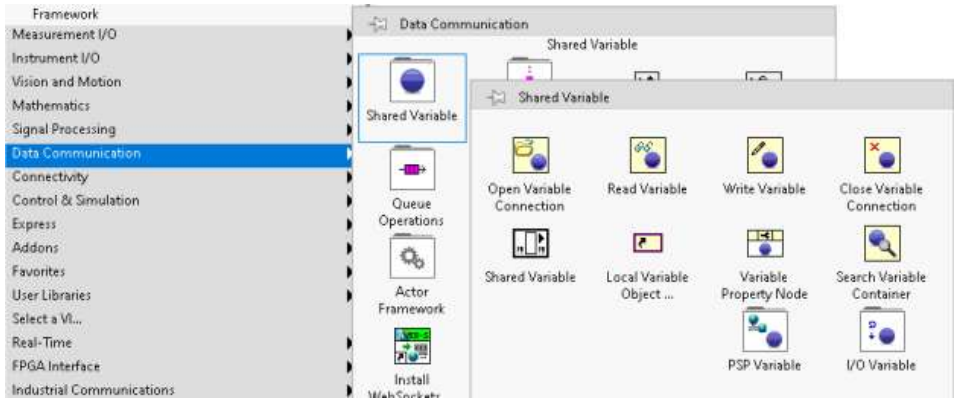
Configure the Modules

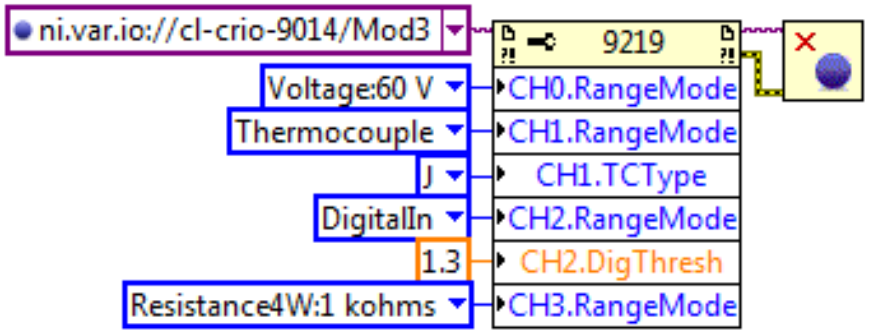
Two options

In project, on module: Right Click → Properties



Configure Shared Variable API





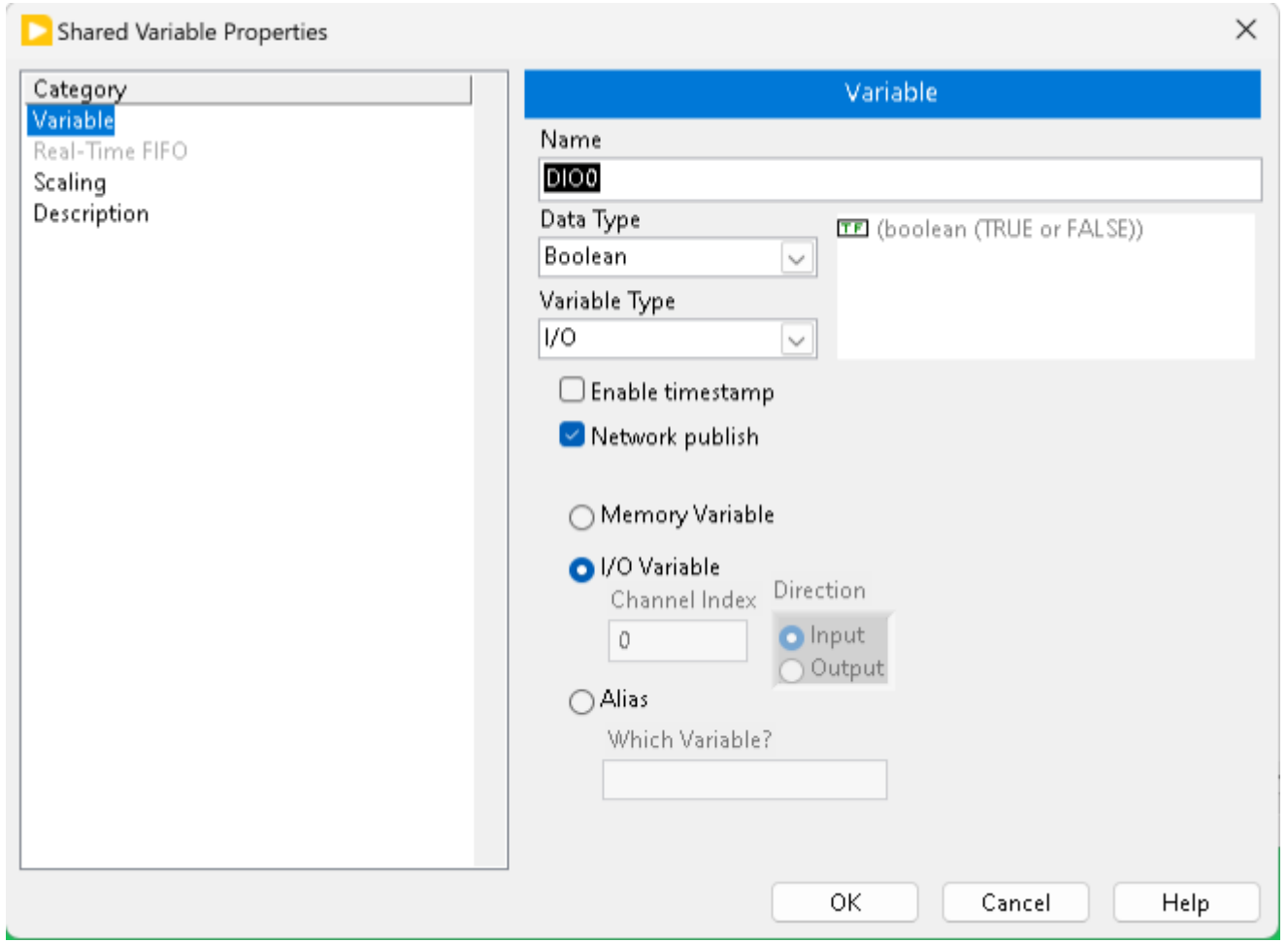
EMERSON

Configure the Variables

In project, on variable: Right Click → Properties

Enable / Disable

- Timestamps
- Network Publishing
- Scaling



Access the IO

Variable Node Method

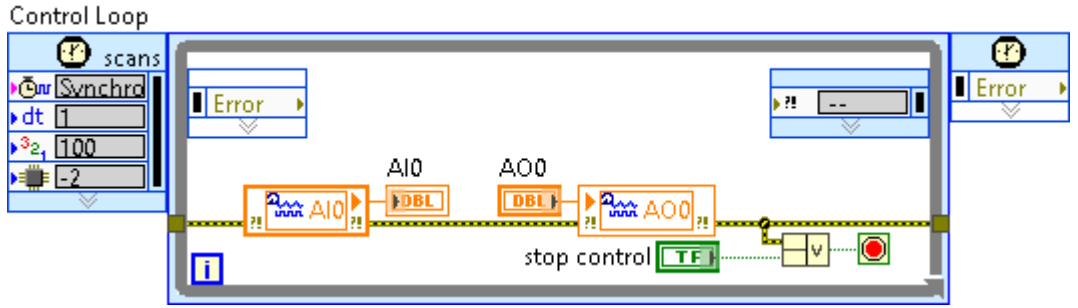
Drag the Variable to the Block Diagram

- Select Variable in Project
- Drag to block diagram
- Drop on the block diagram

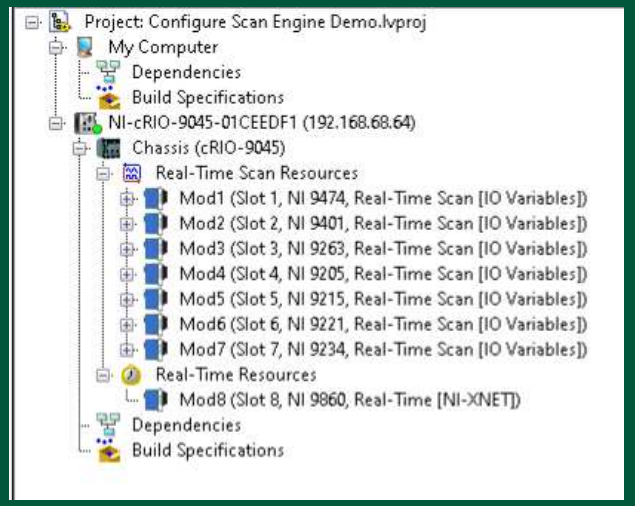
Set Variable Direction

- Right click on dropped variable
- Access Mode → Read / Write

Use a Timed Loop “Synchronize to Scan Engine”



Summary: Configuring cRIO for Scan Engine



Quick getting started experience

Multiple modules synchronization

Access the IO

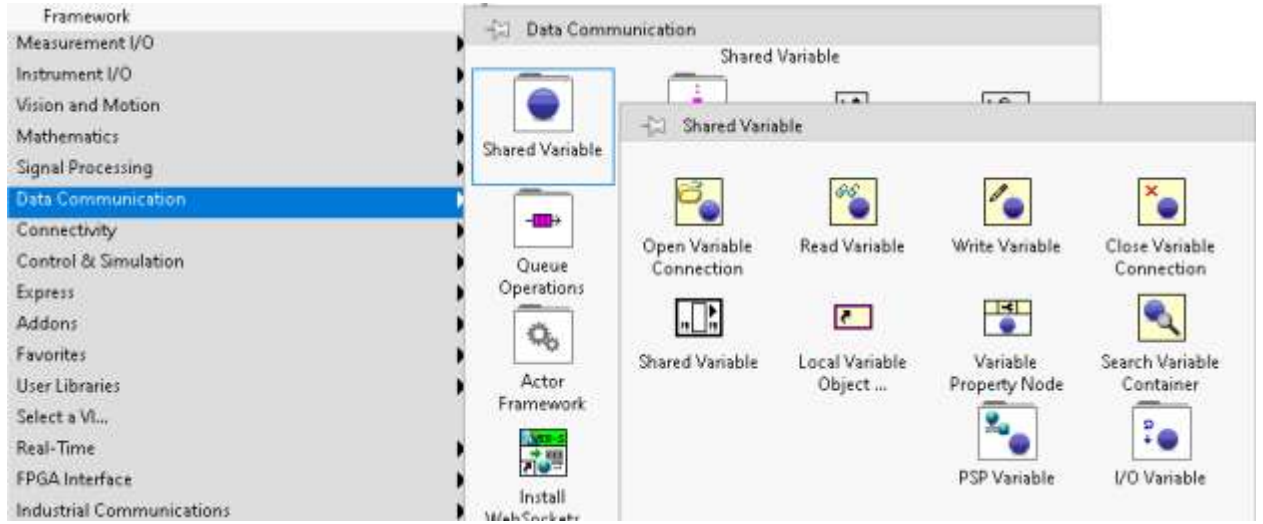
Variable API Method

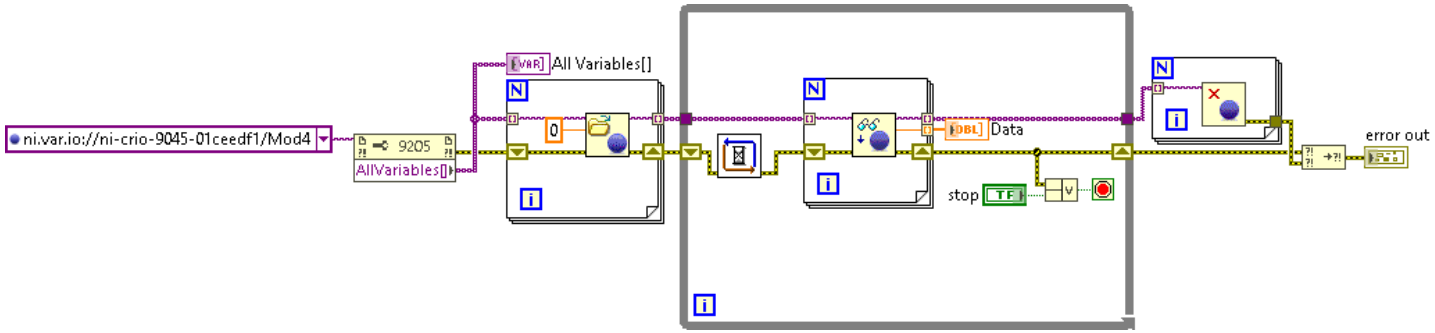
On the palette Data Communication → Shared Variable

- Use Open, Read / Write / Close per string

Access IO by “String” ni.var.io://localhost/Mod4/AI0

- Starts with: ni.var.io://
- Then ip address or name or “localhost” + “/”
- Then module name + “/”
- Then IO name

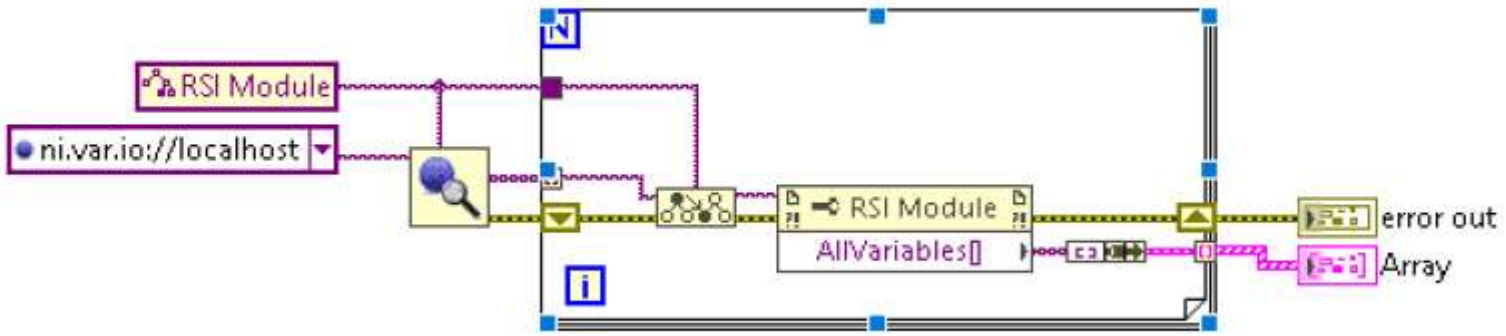




Access the IO

Variable API Method

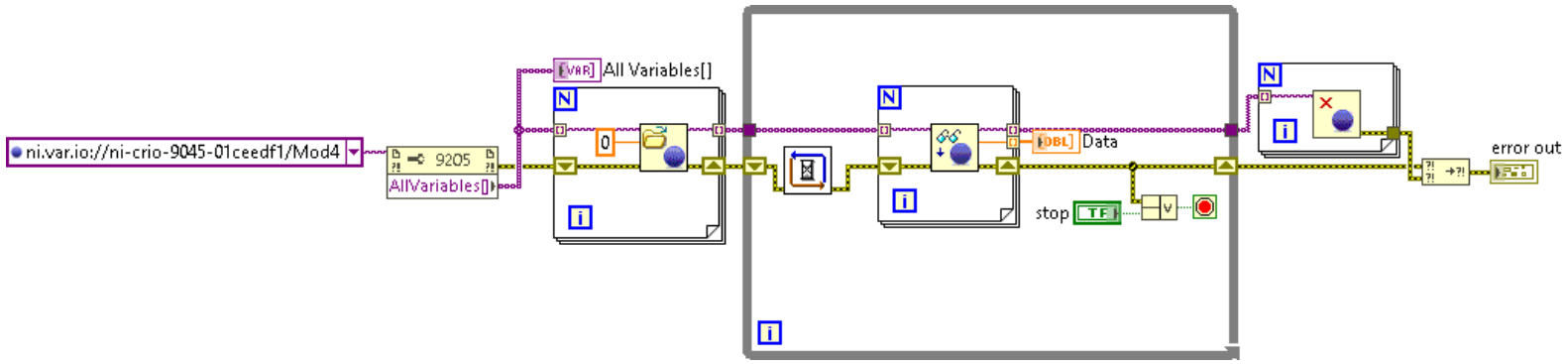
Finding all the IO Variables



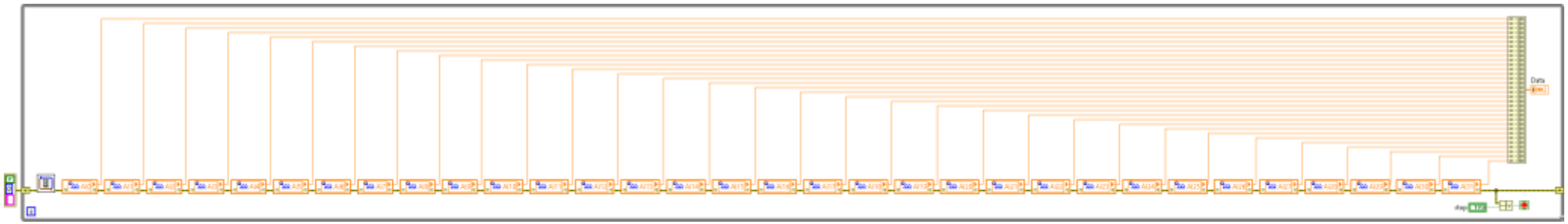
Access the IO

Variable API Method

Read an entire module



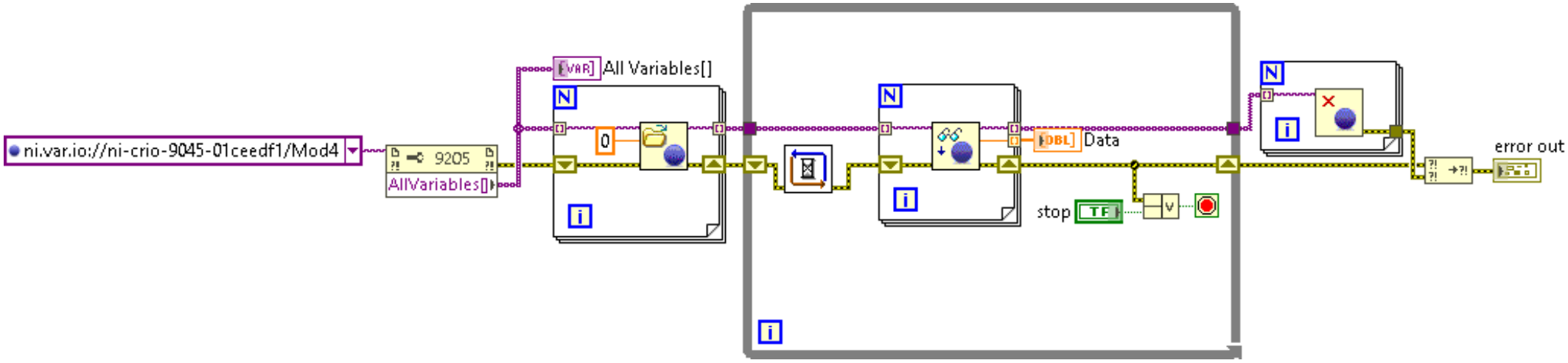
VS



EMERSON

Access the IO

Code at: cRIO Example Code\Scan Engine\Shared Variable API Demo

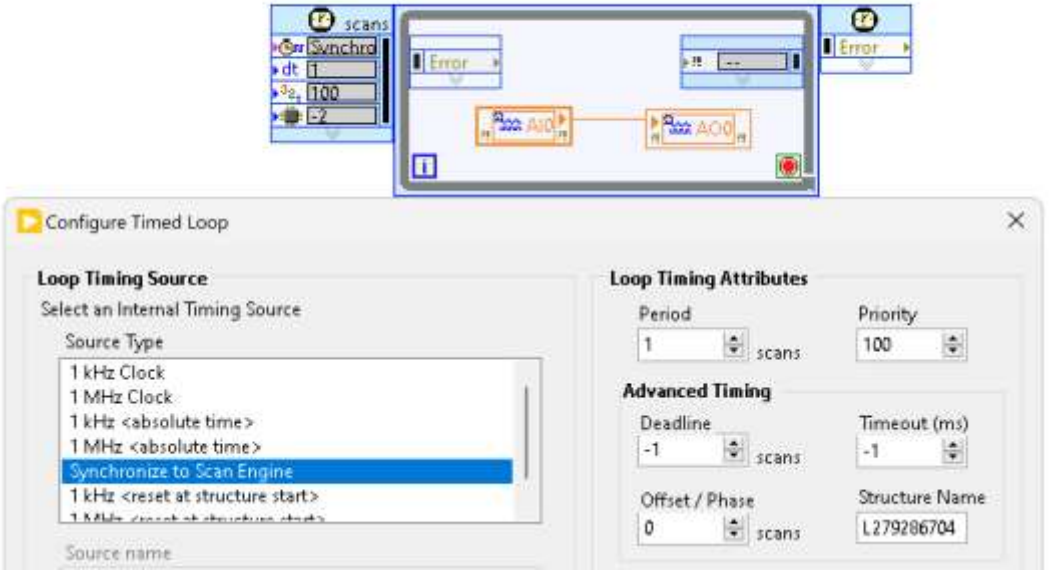


Synching IO Access to Scan Engine

Two methods

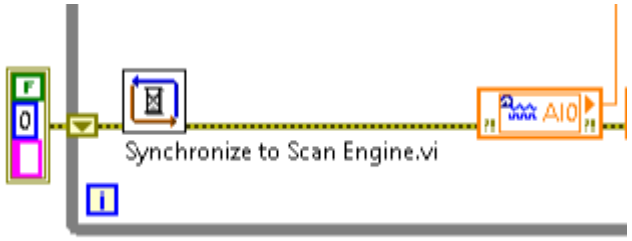
Timed Loop Synchronized to Scan Engine

- Ideal for control due to timed loop priority

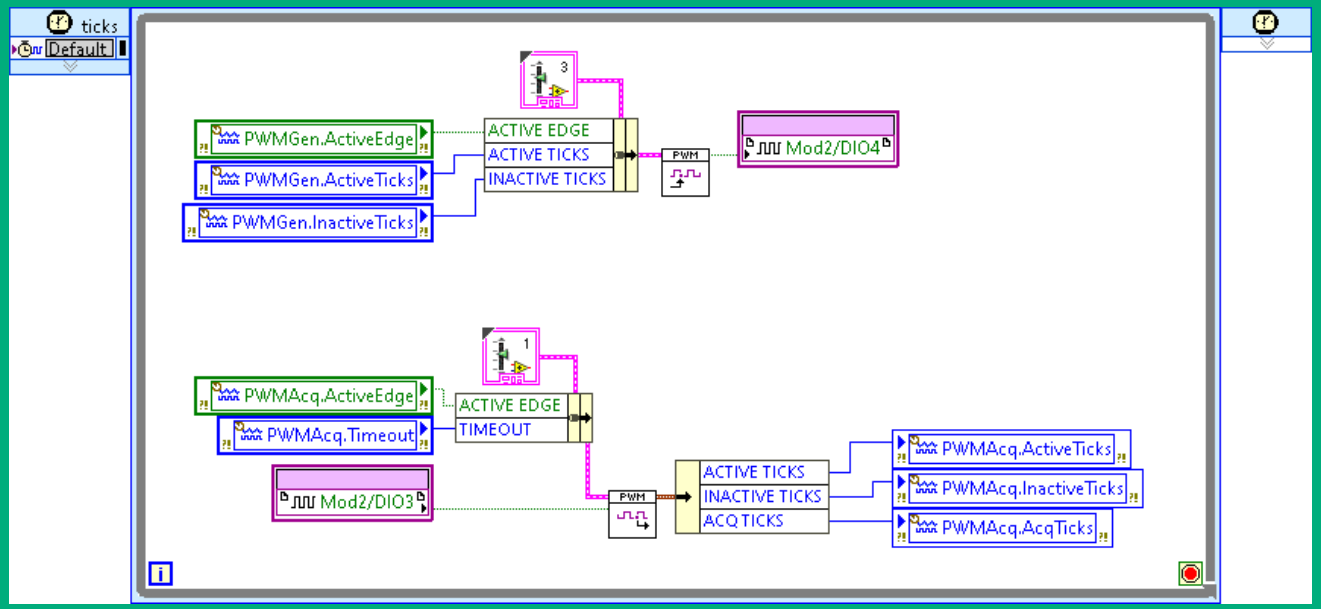


Use Synchronize to Scan Engine.vi

- Ideal for lower priority operations



- NOTE: Both work with variable node or variable API access



Demo: Accessing the FPGA w/ Scan Engine

Access FPGA with Scan Engine

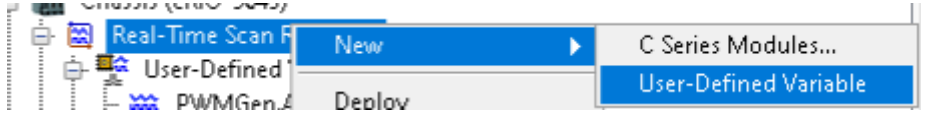
Create User Variables

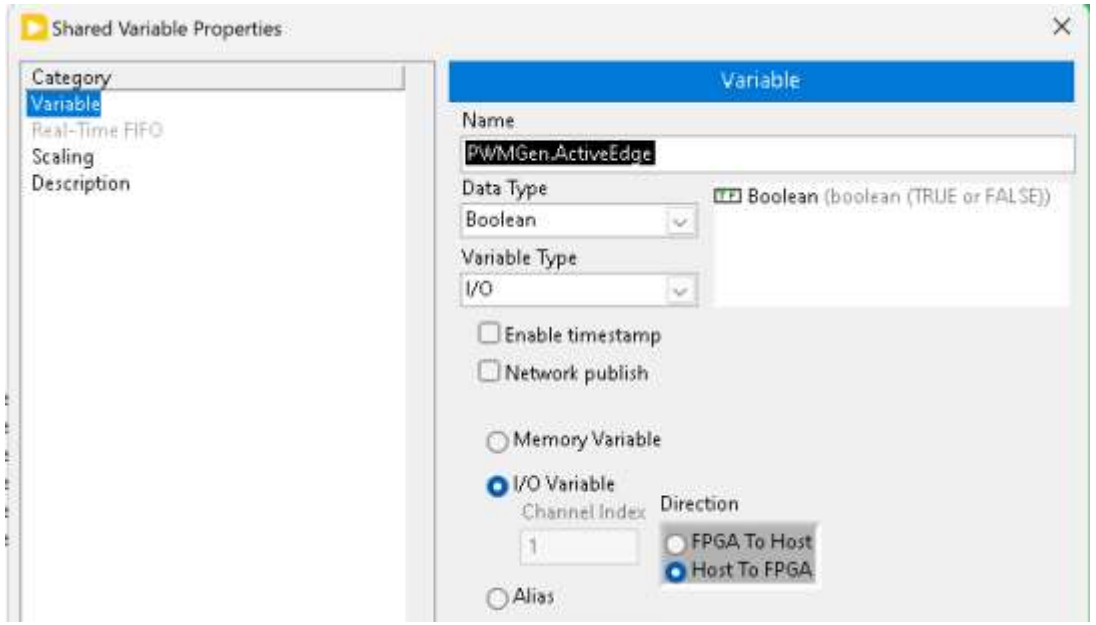
Create User Variable

- Right click on Real-Time Scan Resource
- New → User-Defined Variable
- Pick supported Data Type
- Set Variable Type to I/O
- Choose I/O Variable
- Set Direction (FPGA → Host or Host → FPGA)

Deploy settings from Chassis

- Right click on Chassis
- Deploy All





Access FPGA with Scan Engine

Access User Variables in FPGA App

Drag User Variable to FPGA app

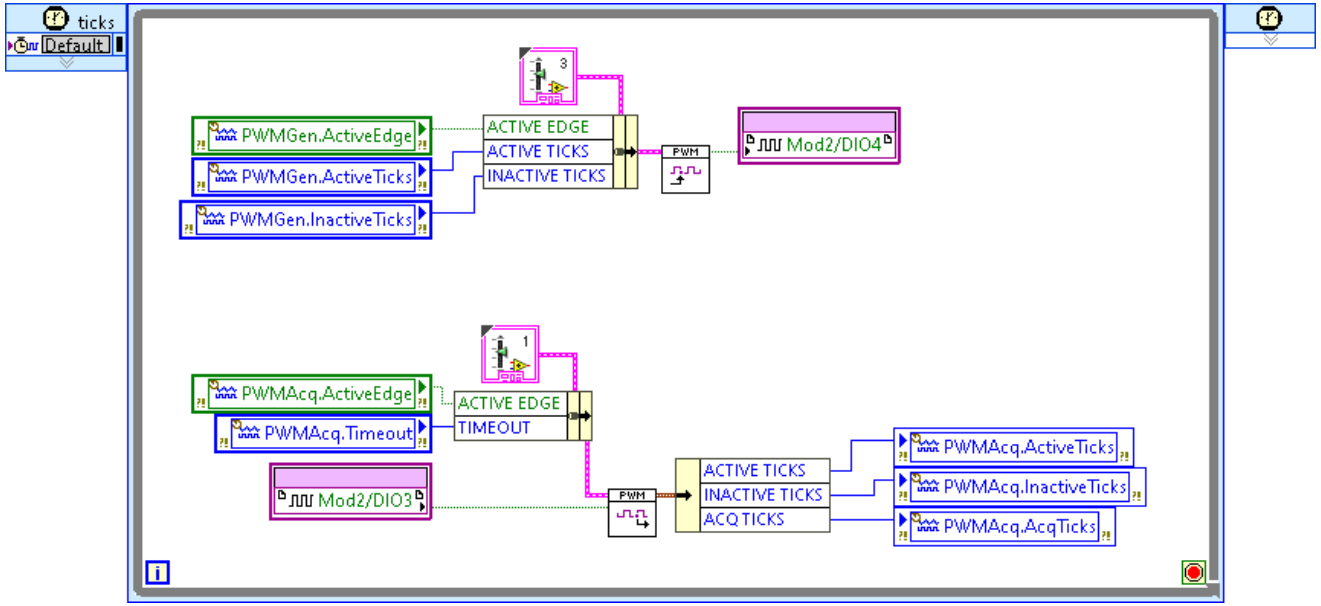
- In Project, From Real-Time Scan Resources → User-Defined Variables
- Drag variable
- Drop into FPGA Program

Set Variable Direction

- Right click on dropped variable
- Access Mode → Read / Write

Build FPGA program

Compile bitfile



Access FPGA with Scan Engine

Access User Variables in Host App

In project, Set RIO Programming Mode to FPGA Interface

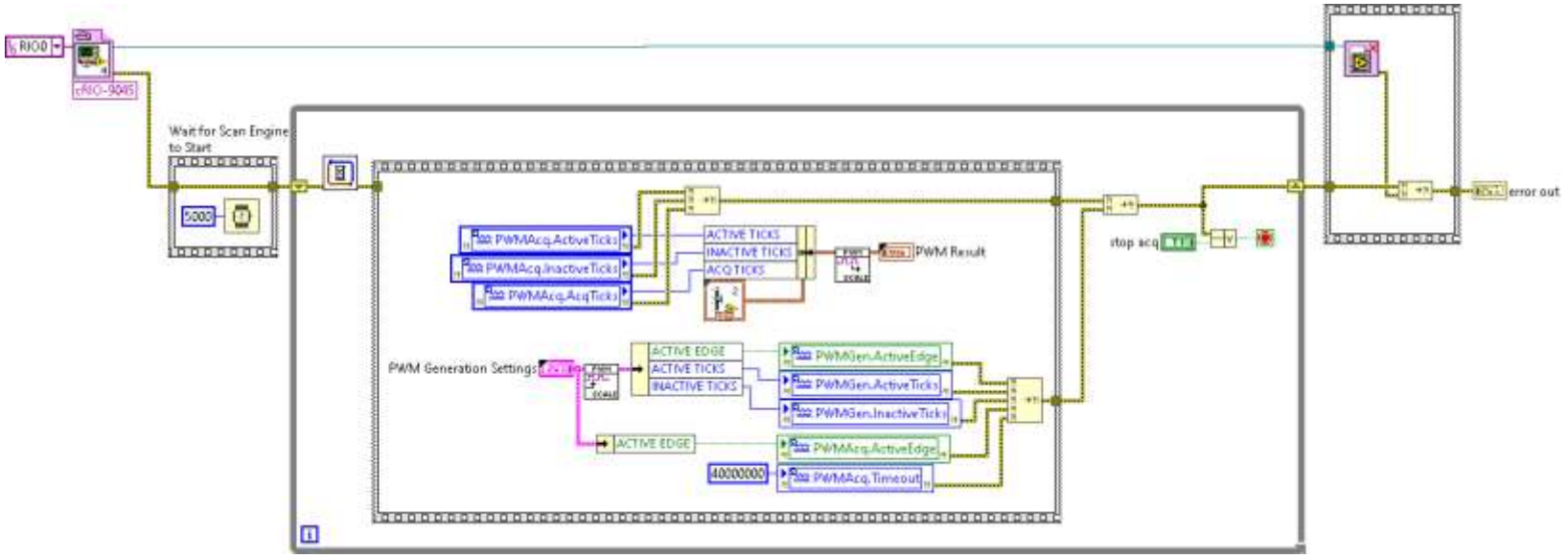
- Right click on Chassis
- Properties
- Set Mode to FPGA Interface

Deploy settings from Chassis

- Right click on Chassis
- Deploy All

In RT App, Start FPGA

- Open FPGA program (linked to bitfile)
- Start FPGA program (checkbox in Open FPGA)
- Wait for Scan Engine to start



Access FPGA with Scan Engine

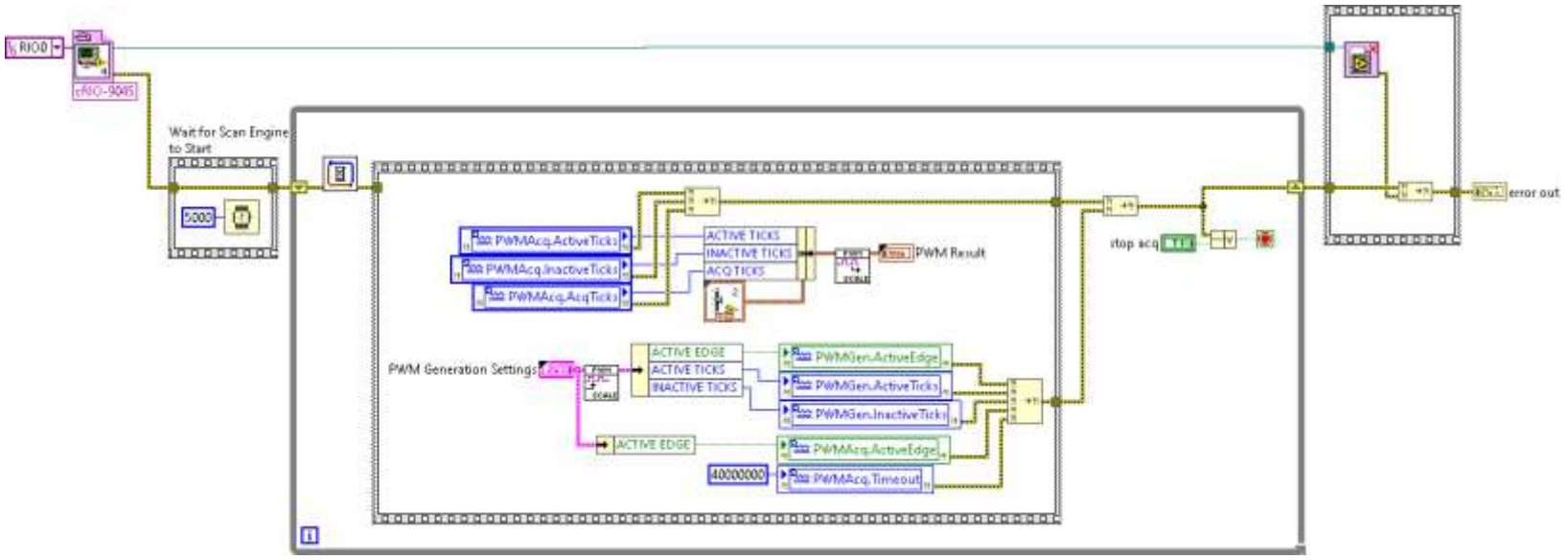
Access User Variables in Host App

Drag User Variable to RT app

- In Project, From Real-Time Scan Resources → User-Defined Variables
- Drag variable
- Drop into RT Program

Synchronize to Scan Engine (if needed)

Stop FPGA when done



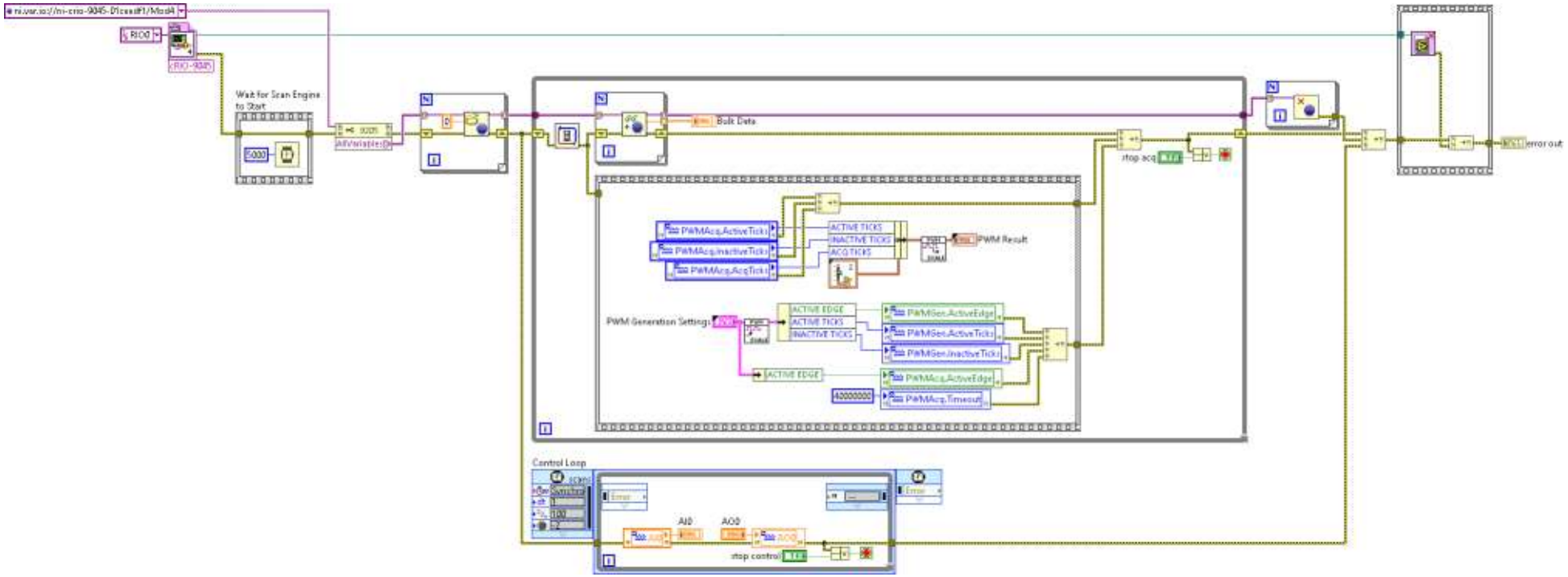
Host Example: cRIO Example Code\Scan Engine\Accessing the FPGA Demo\RT\FPGA User Variables Host.vi

EMERSON

Access FPGA with Scan Engine

Add in Scan Engine Variables

Add in additional scan engine variables after the Wait / Open Variable Connections:

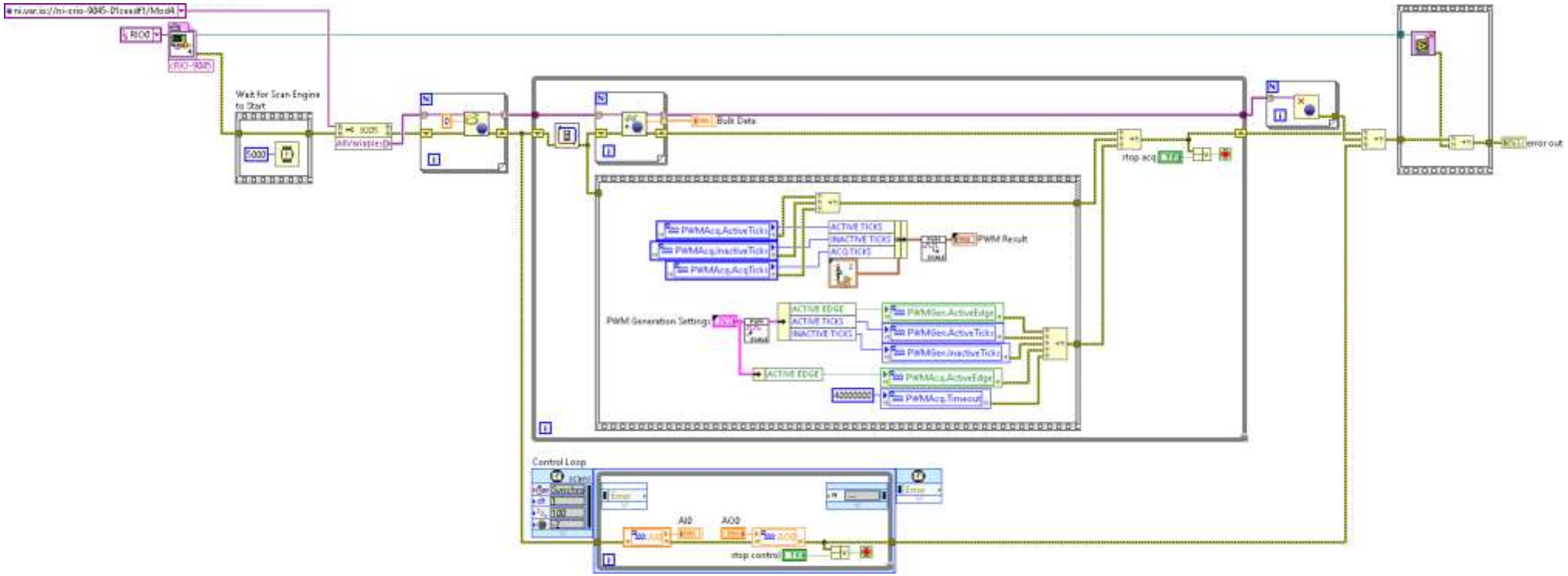


Example: cRIO Example Code\Scan Engine\Accessing the FPGA Demo\RT\Scan Engine and FPGA User Variables Host Demo.vi

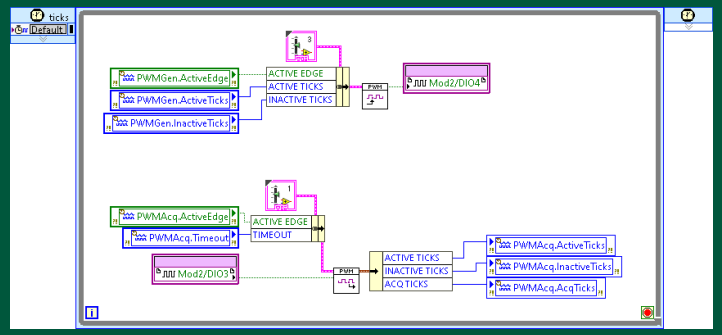
EMERSON

Access FPGA with Scan Engine

Code at: cRIO Example Code\Scan Engine\Accessing the FPGA Demo



Summary: Accessing the FPGA w/ Scan Engine



Create custom logic on FPGA

Pass values through Scan Engine

Expanding IO w/ Scan Engine

NI 9145 Chassis w/ FPGA

Connect to CompactRIO (Right Click → Connect)

Add EtherCAT Master

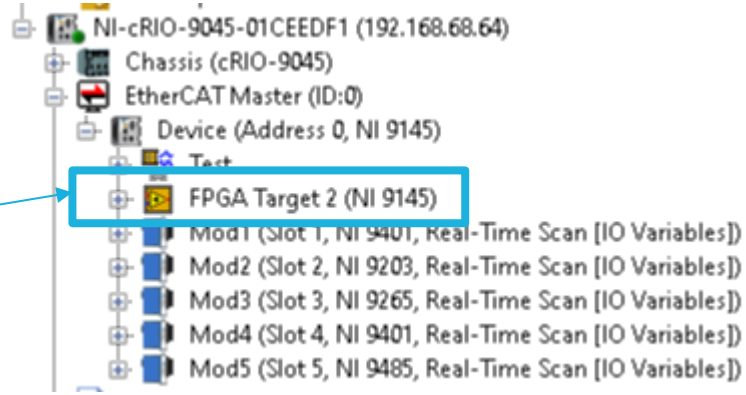
- Right Click → New → Targets and Devices
- Select EtherCAT Master Device

Devices will be automatically discovered and added

Works (mostly) like Scan Engine on cRIO

- Follow configure steps
- Drag and drop / Use API
- Including FPGA Access with User Variables!!

Synchronizes to Scan Engine via EtherCAT



Scan Engine Summary

Key Benefits

- By far, the easiest experience

Key Caveats

- Counter support better than NI DAQmx but still not optimized
- No waveform support
- Only supported in LabVIEW (no Python / C++)

Summary: Scan Engine IO on cRIO

Pros

- Ease of Use (Multiple Modules, Synchronization)
- Optimized for processor – based control
- Synchronized IO Expansion

Cons

- No waveform support
- LabVIEW Only
- Counters are better, not still not ideal

FPGA IO on cRIO

Better Understanding → Better Design

EMERSON

LabVIEW FPGA on CompactRIO

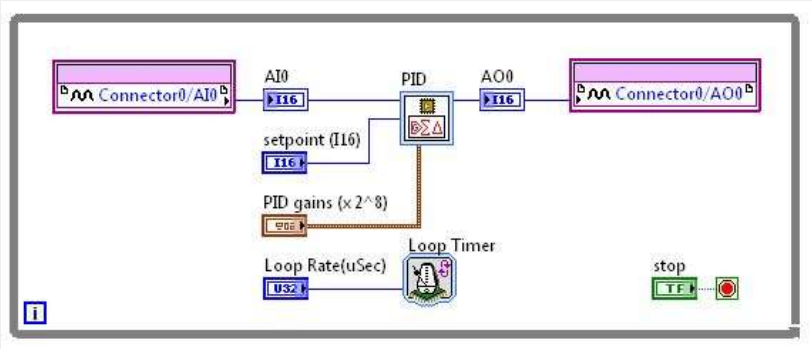
Performance and Flexibility to Solve Current and Future Challenges

Control loop rates up to 10s of MHz

In-line signal processing

Custom timing and triggering

High Speed and Advanced Control



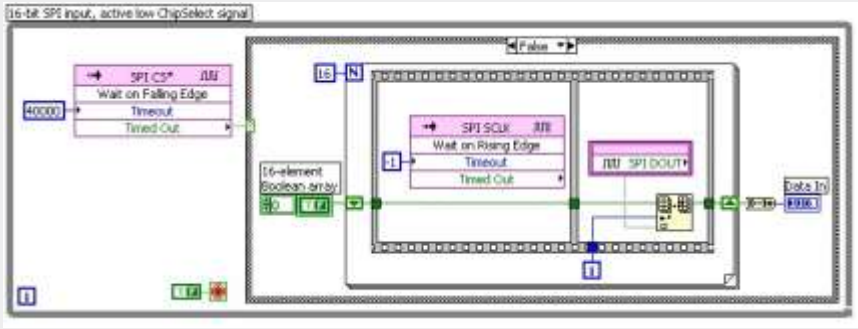
Expanded Measurement Options

Access to new NI modules (Ex: Serial, CAN, Profibus/Profinet, Functional Safety)

3rd party C Series modules

Connect to custom sensors with a high speed digital interface

Connect to New Sensors and Busses



Get your drivers right…

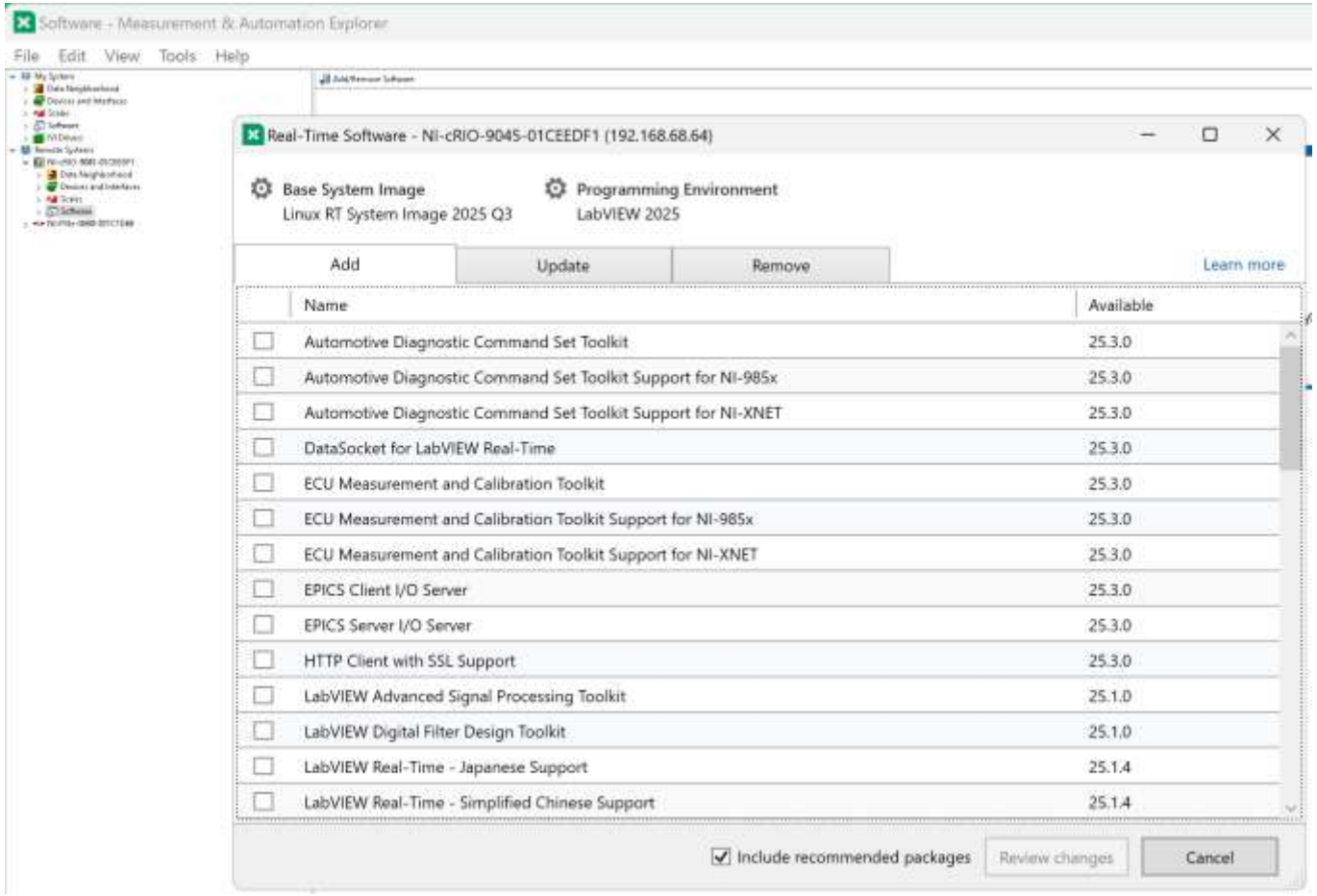
Host and cRIO driver version must match

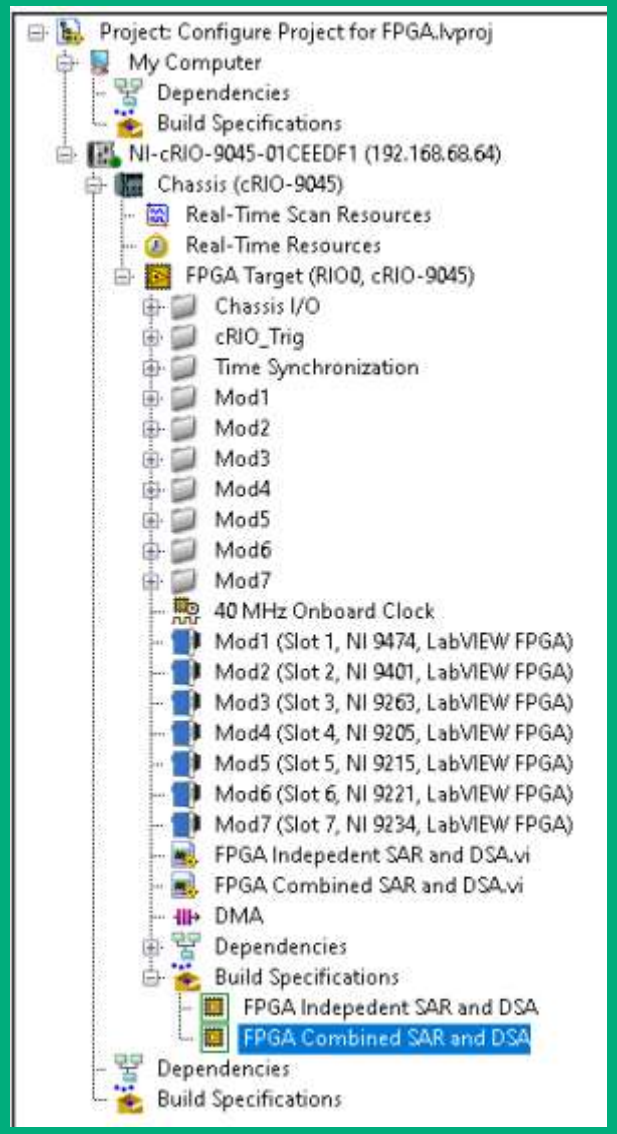
Check versions in MAX

Update as necessary

cRIO + FPGA requires:

- CompactRIO Support
- NI-RIO / IO Scan / Server
- NI-Sync





Demo: Configuring cRIO for FPGA

Create the LabVIEW Project

Create new blank LabVIEW Project

- Create Project → Blank Project

Add cRIO

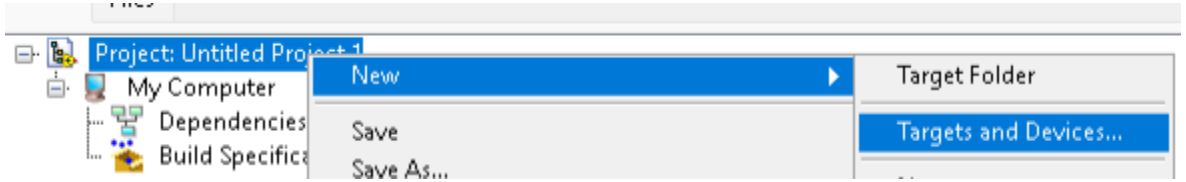
- Right click on Project
- New → Target and Devices

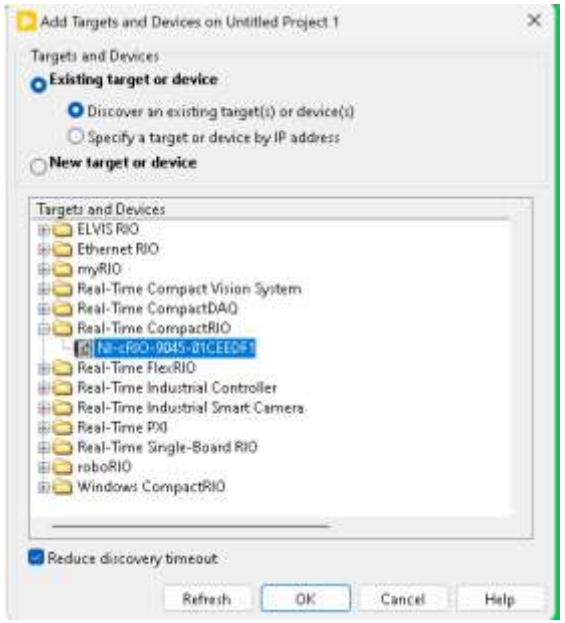
Select your cRIO from Real-Time CompactRIO

- Computer and cRIO must be on same subnet
- Modules will automatically be added with default settings

Delete all modules from:

- Real-Time Scan Resources
- Real-Time Resources





Configure the Chassis

Make sure connected to cRIO

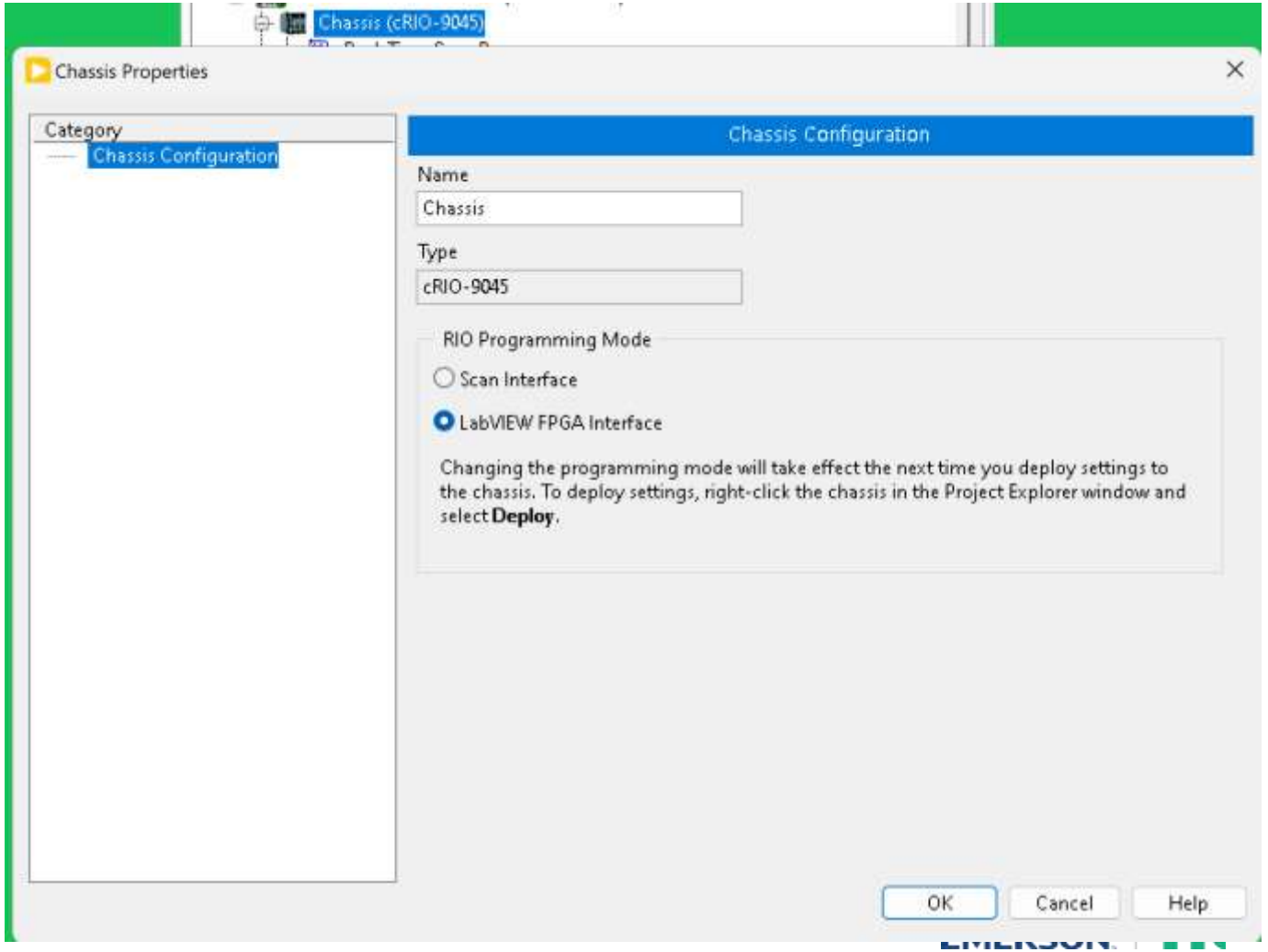
- Right click on cRIO
- Connect

Set RIO Programming Mode to LabVIEW FPGA Interface

- Right click on Chassis
- Properties
- Set Mode to LabVIEW FPGA Interface

Deploy settings from Chassis

- Right click on Chassis
- Deploy All



Add Modules to FPGA Target

Make sure connected to cRIO

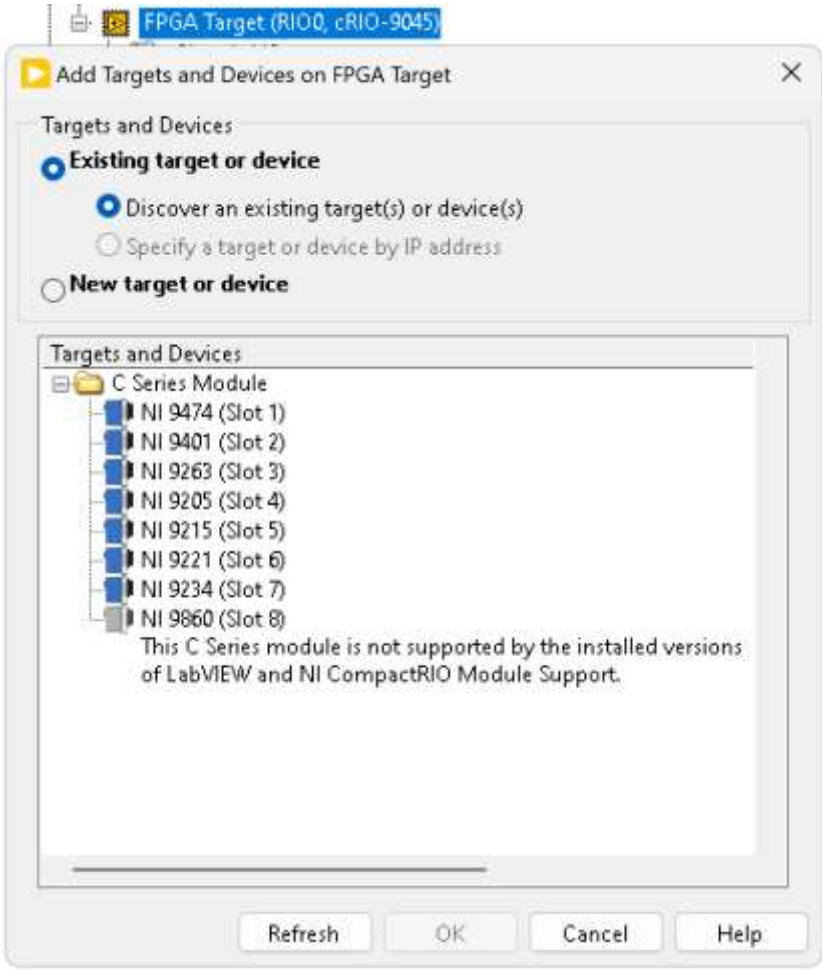
- Right click on cRIO
- Connect

Add modules to FPGA Target

- Right click on FPGA Target
- New → C Series Modules
- Select desired modules from GUI
- Select OK

Deploy settings from Chassis

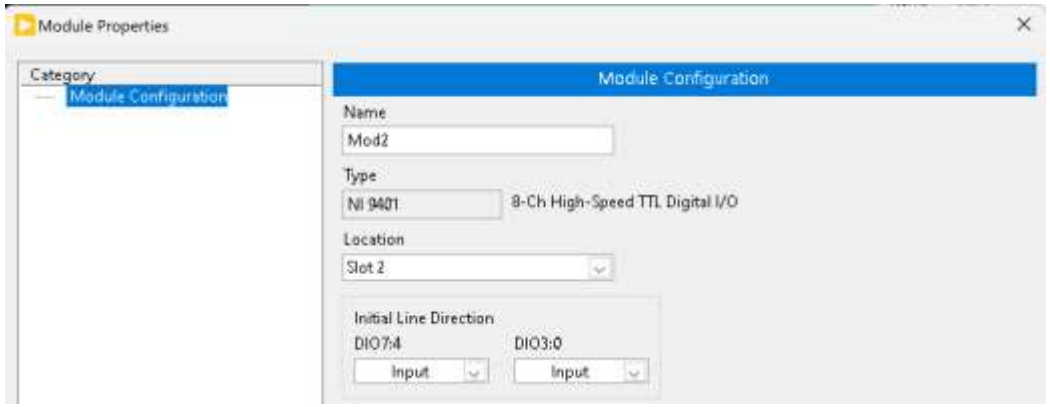
- Right click on Chassis
- Deploy All



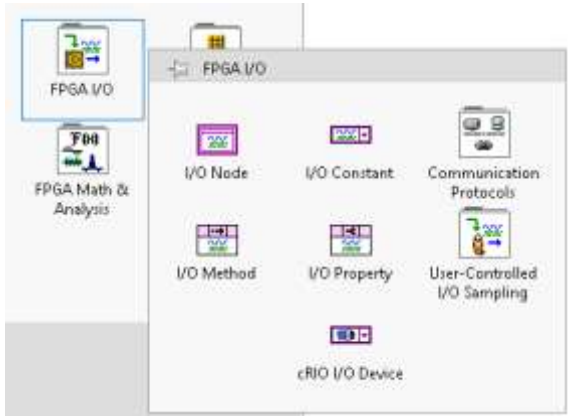
Configure the Modules

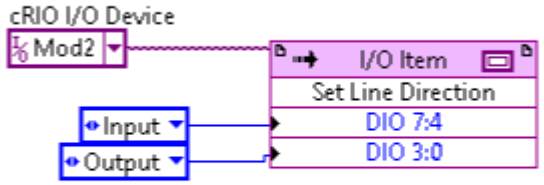
Two options

In project, on module: Right Click → Properties



Configure w/ FPGA I/O API





Access the IO in FPGA App

Drag the IO Node to the Block Diagram

- Select IO in Project
- Drag to block diagram
- Drop on the block diagram

Set IO Node Direction

- Right click on dropped node
- Change to Read / Write

Drag IO Node down to add more IO

Put a while loop around the IO node

Save VI

Access the IO in FPGA App

Compiling

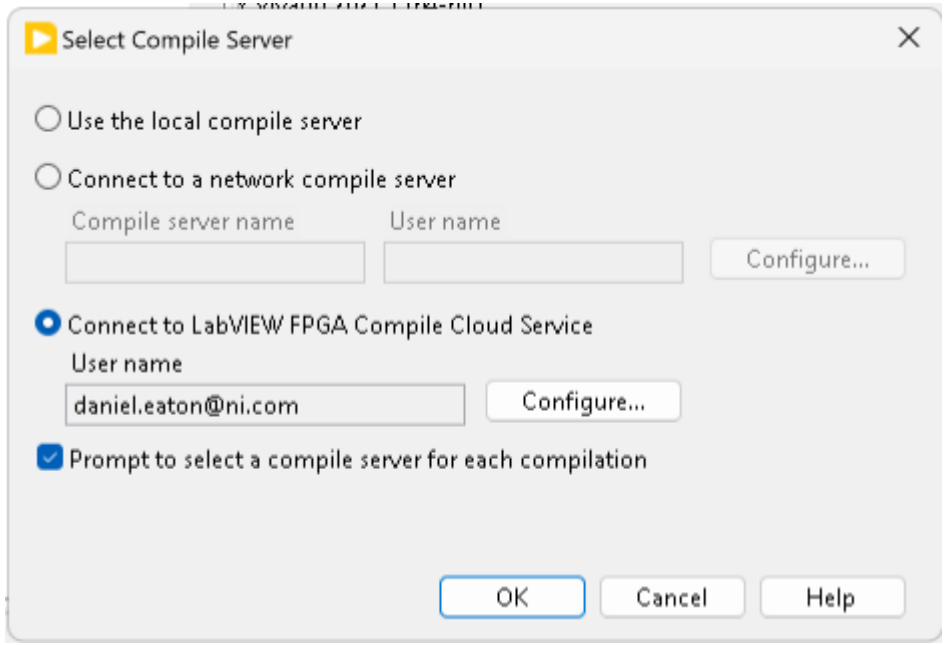
Compile

- Right click on FPGA VI → Create Build Specification
- Right click on build spec → Build

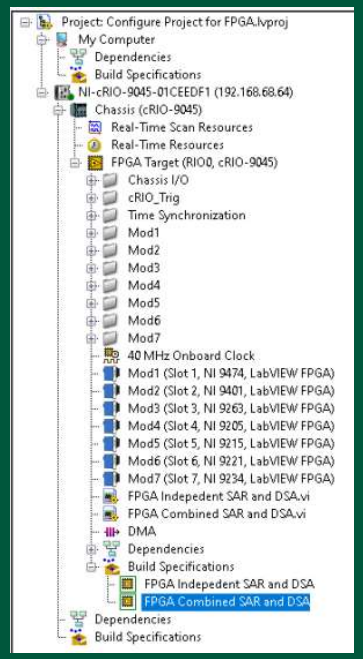
LabVIEW FPGA Compile Cloud Service for optimized workflow

- Transfers FPGA program to NI owned server
- Allows user to continue to work
- Secure but technically “off premises”

Use local compile server for “most secure”

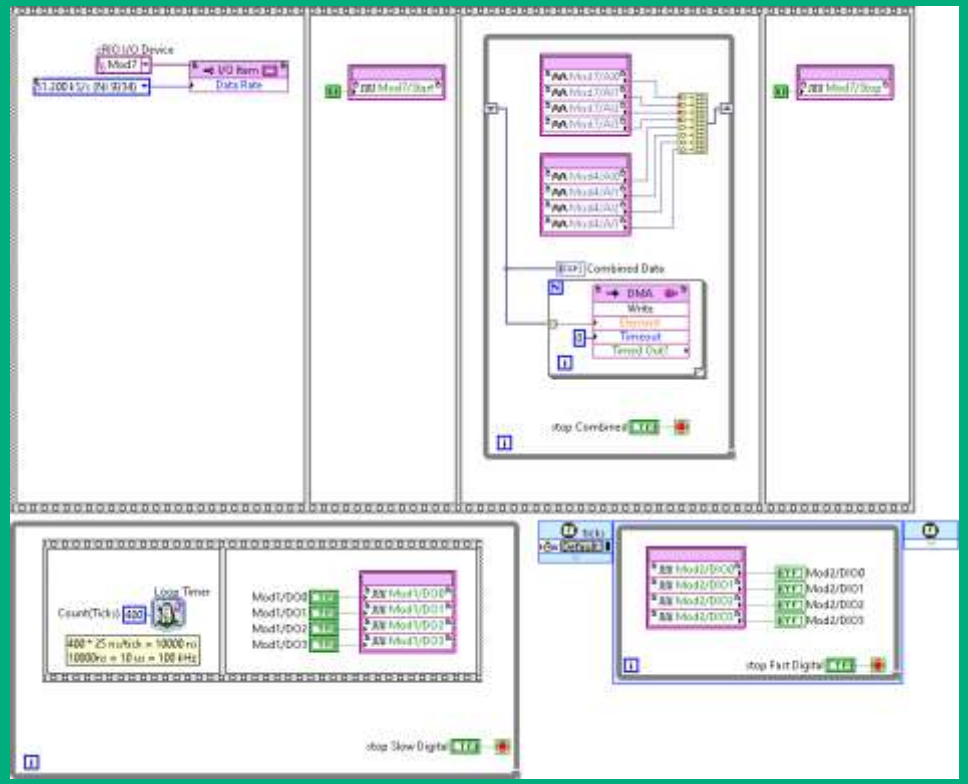


Summary: Configuring cRIO for FPGA



Lowest level of access → Ultimate control

Ability to offload compiling to cloud



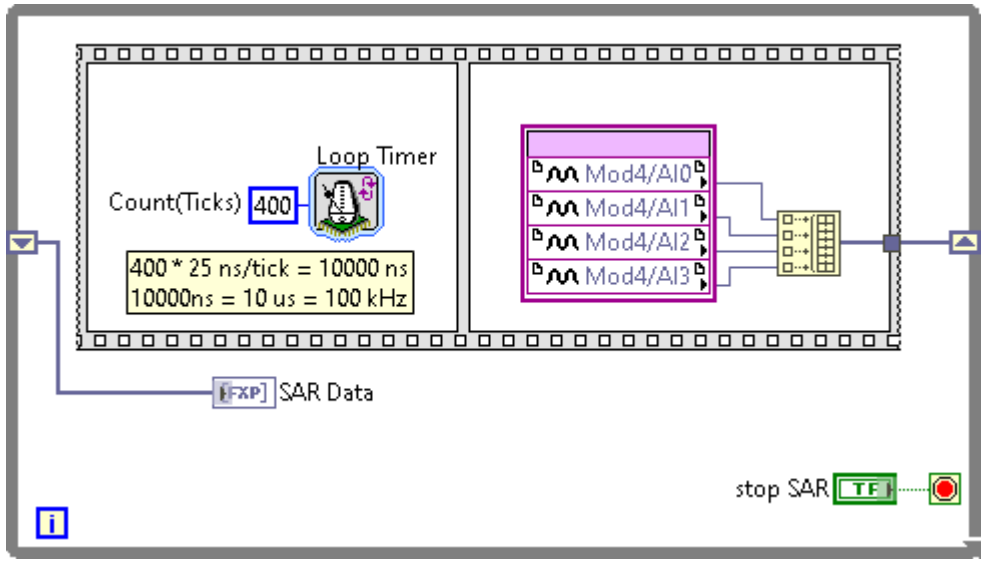
Demo: Basic Host + FPGA Example Walkthrough

Timing IO in FPGA App

SAR AI / AO

Explicitly time the loop

- Use Loop Timer to set period
- Use sequence structure to control data flow
- Pipeline data to optimize loop rate



Can't exceed module capability

Important Math

- 1 tick is 25 nanoseconds
- 400 ticks * 25 nanoseconds / tick = 10,000 nanoseconds
- 10,000 nanoseconds = 10 microseconds = .01 milliseconds = .00001 seconds
- 1 / .00001 seconds = 100 kHz

Timing IO in FPGA App

DSA AI

Configure the module timing

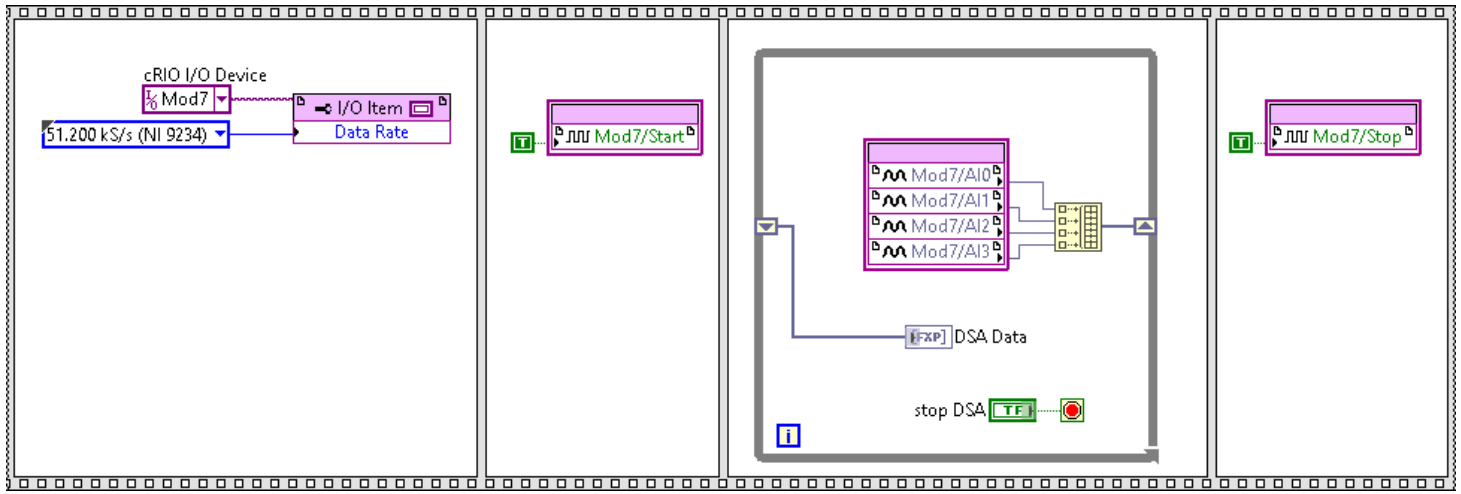
- FPGA I/O API
- Module properties in project

Start the module clock

Module times itself, no loop timer needed

Still pipeline for optimized loop rates

Stop the module clock when done



Timing IO in FPGA App

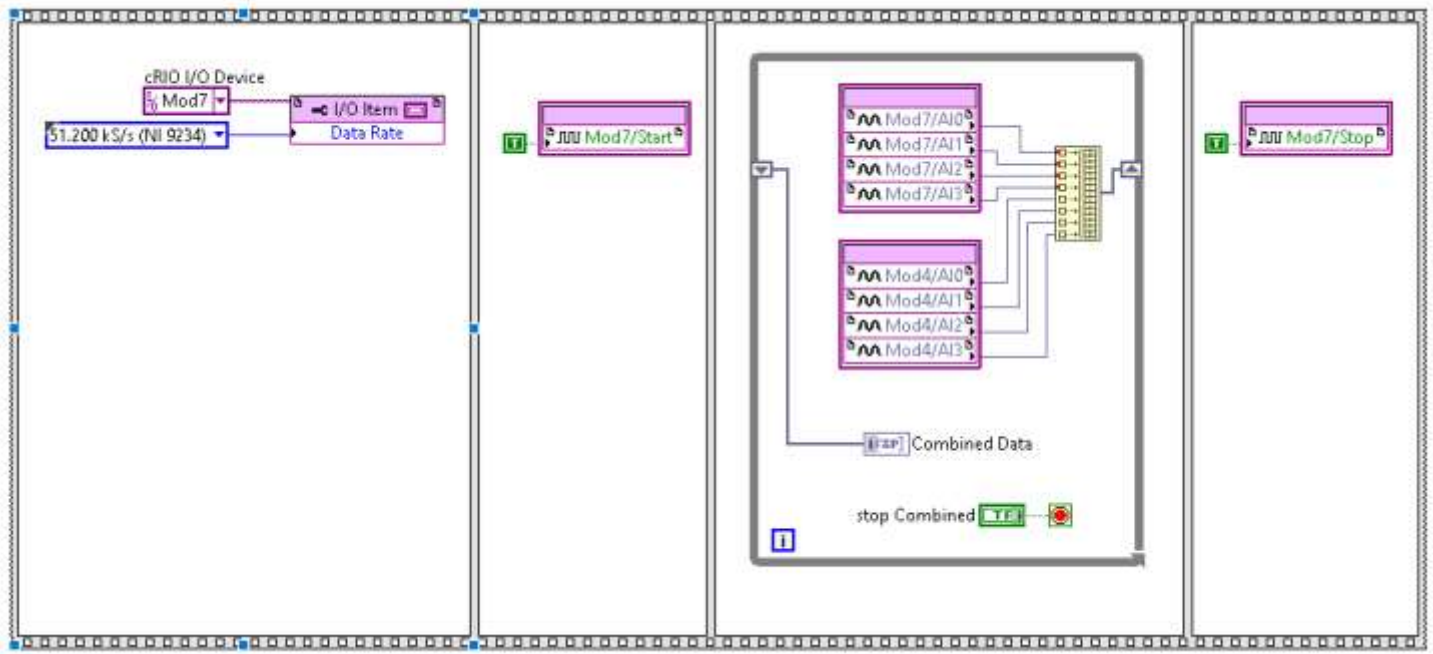
SAR + DSA

Allow DSA module to time SAR module

Keep SAR modules in different IO node

SAR per chan rate >= DSA desired rate

- NI 9215 (100 kHz) + NI 9234 (51.2 kHz) = Good
- NI 9205 (16 ch 15.625 kHz) + NI 9234 (51.2 kHz) = Bad
- NI 9205 (16 ch 15.625 kHz) + NI 9234 (10.24 kHz) = Good

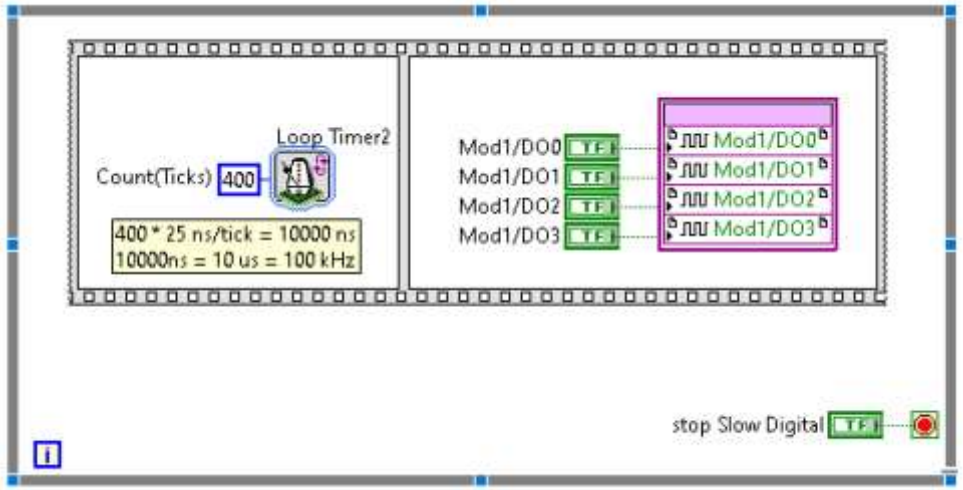


Timing IO in FPGA App

Digital IO

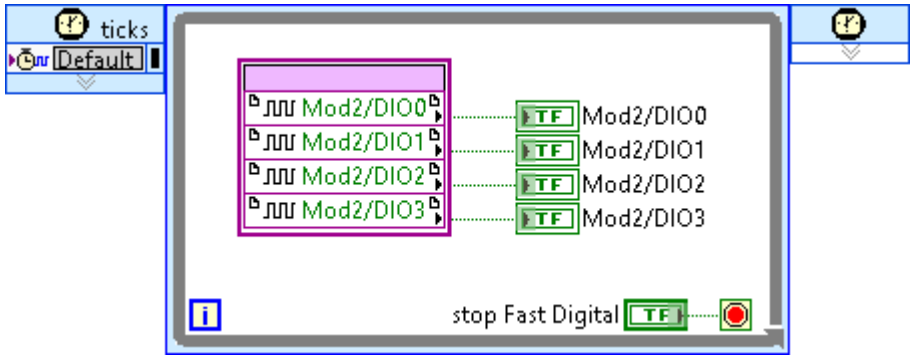
Slow Speed → Standard While Loop

- Time like analog input loop
- Slower loop rate = worse resolution
- All modules will compile in loop



High Speed → SCTL (Single Cycle Timed Loop)

- Timing in 1 tick (25 ns)
- Highest possible resolution
- Only certain modules will compile in SCTL



Timing IO in FPGA App

PWMs

FPGA IO is read or write; User builds the logic

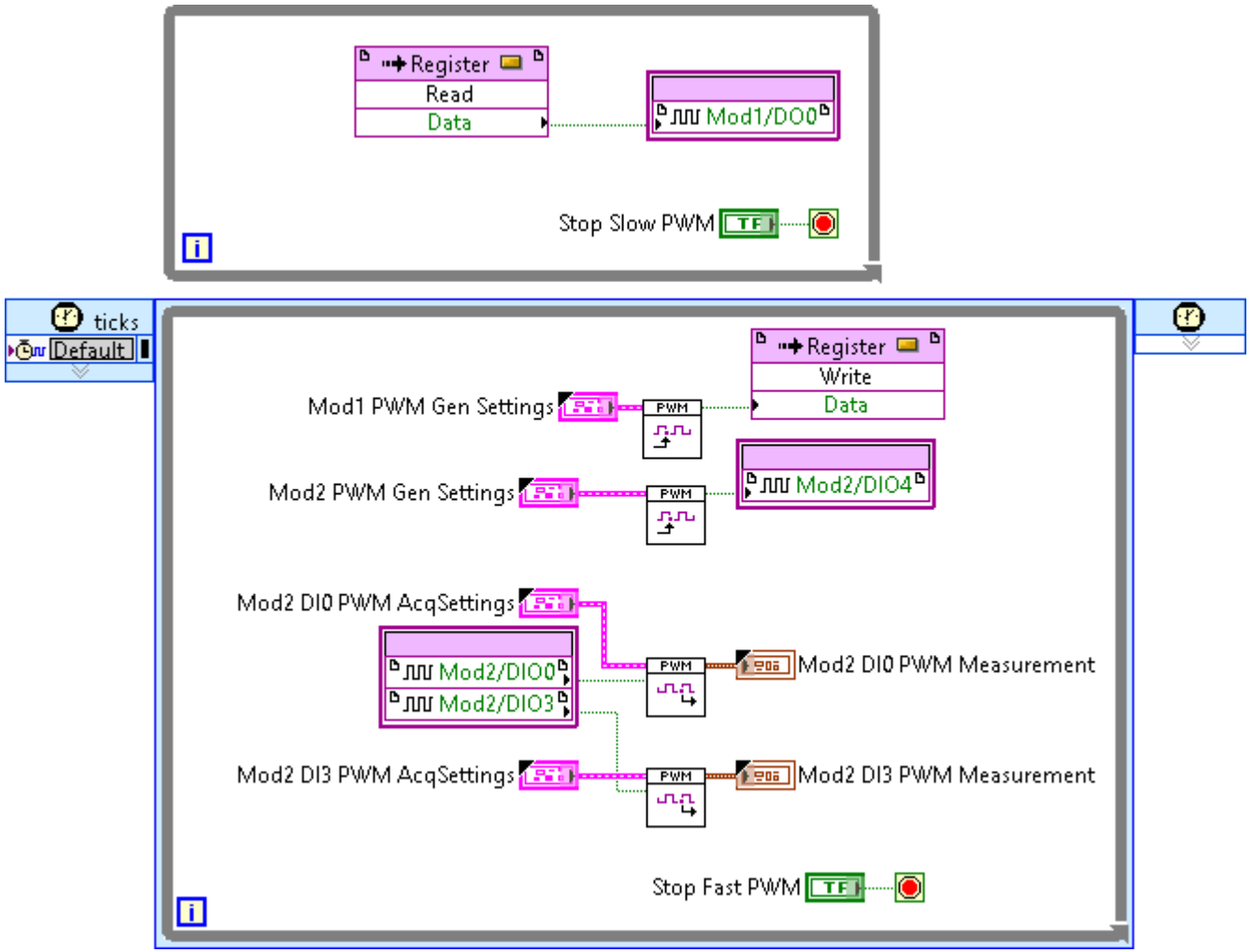
User responsible for PWM logic; typically

- Reads digital line
- Determines rising / failing edge
- Count FPGA ticks between edges
- Latches count on rising / failing edge

Same on generation (just reverse)

Lots of community made functions already out there

Example at: cRIO Example Code\FPGA\PWMs

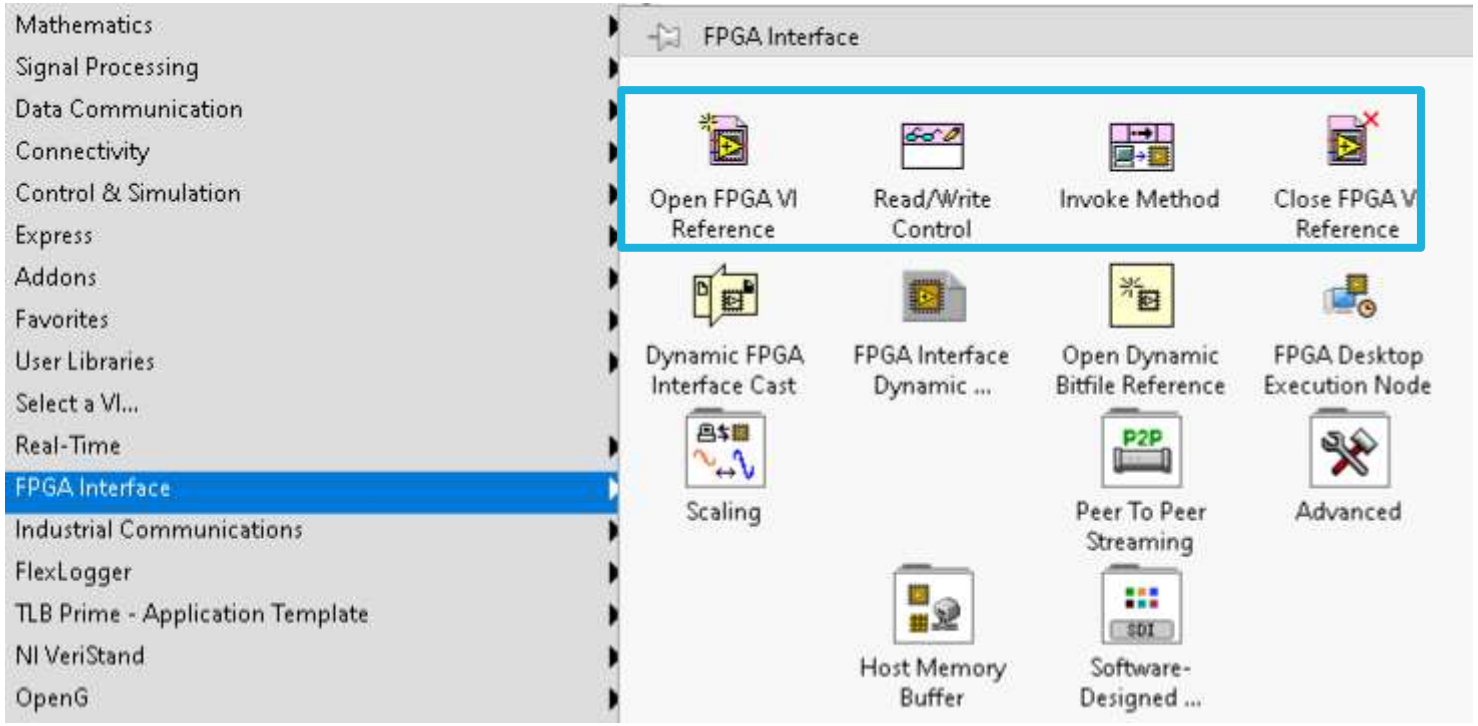


Accessing the IO From in RT App

FPGA Interface API

Core functions for accessing FPGA App from RT App

- Open
- Read/Write Control
- Invoke Method
- Close

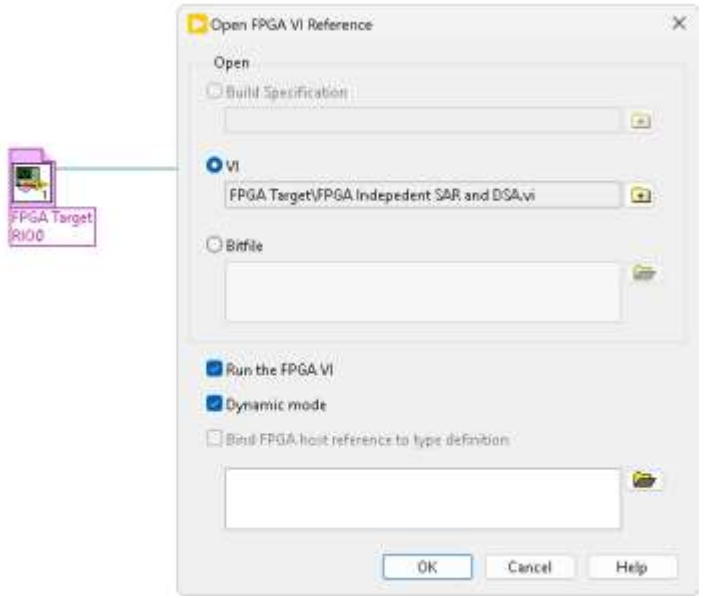


Accessing the IO From in RT App

Start and stop FPGA App from RT App

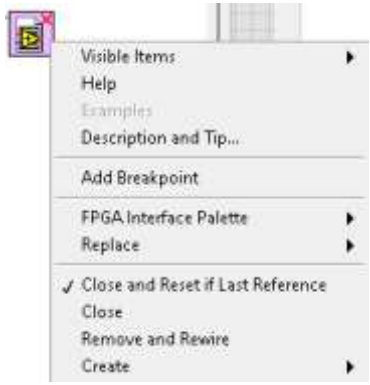
Start the FPGA Program from RT

- Drop Open FPGA VI Reference
- Right click → Configure Open FPGA VI Reference
- Select the FPGA VI, Build Spec, or Bitfile
- Set Run FPGA VI true (typically)
- Set Dynamic Mode true



Stop the FPGA Program from RT

- Drop Close FPGA VI Reference
- Right click → Close and Reset if Last Reference



Accessing the IO From in RT App

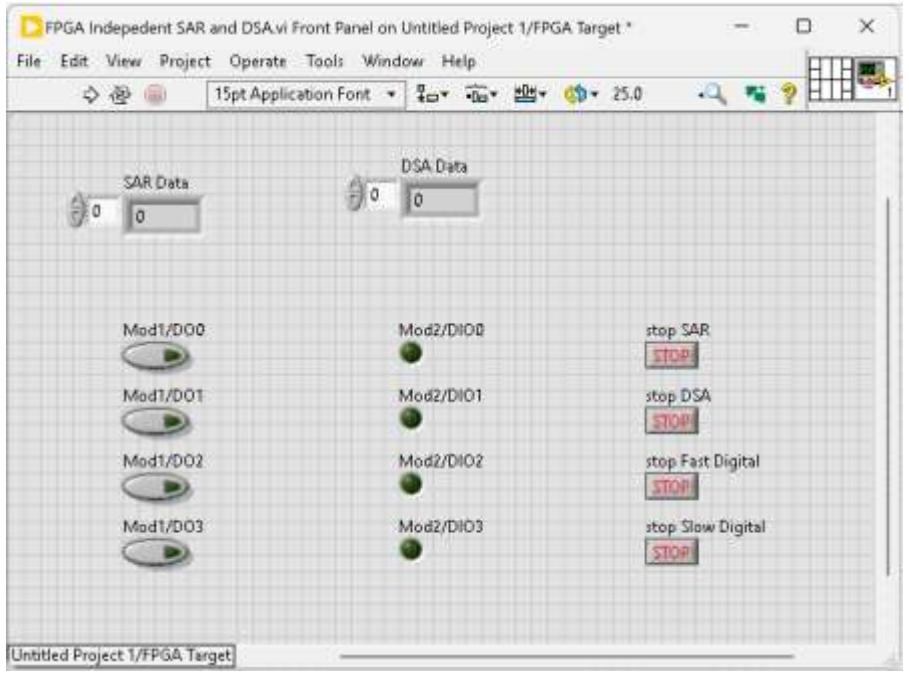
Read / Write Controls

Accesses FPGA Front Panel from RT App

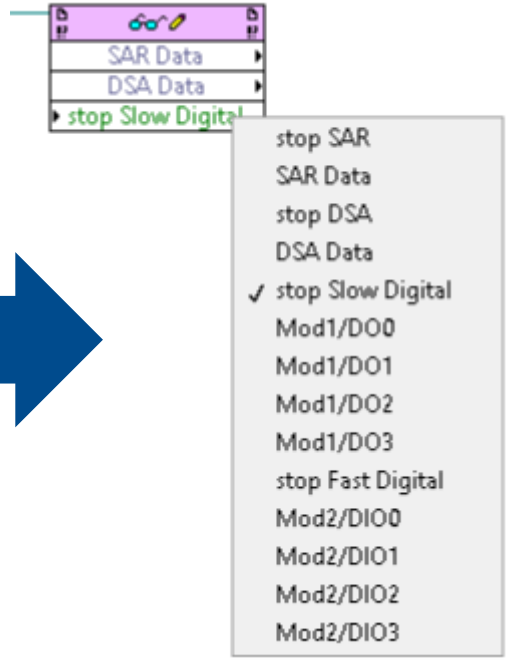
- Easiest access
- Lowest latency, lowest bandwidth
- Single point only
- Asynchronous access

To use:

- Drop Read/Write Control method
- Wire to Open FPGA VI Reference
- Drag down number of required items
- Left click item → Select control



FPGA Front Panel



RT Block Diagram

Accessing the IO From in RT App

DMA in FPGA

Add FIFO to FPGA Target

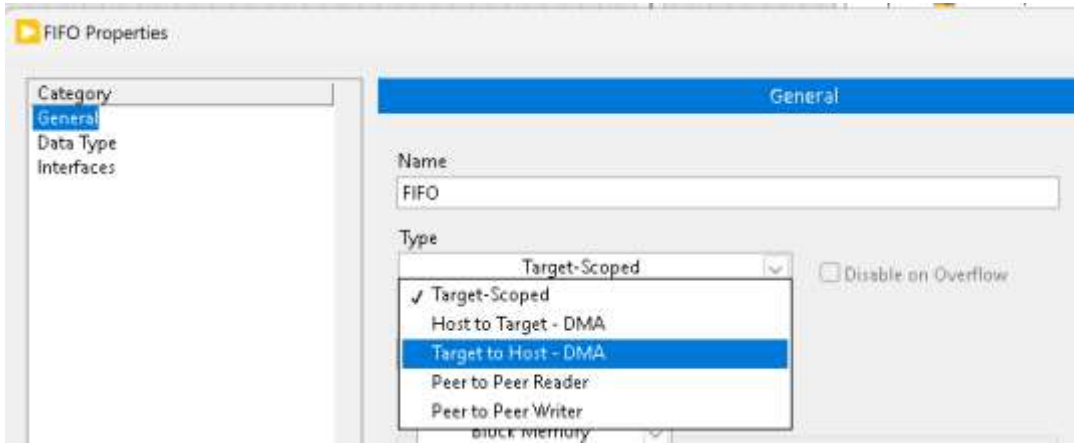
- In project, right click on FPGA Target
- New → FIFO

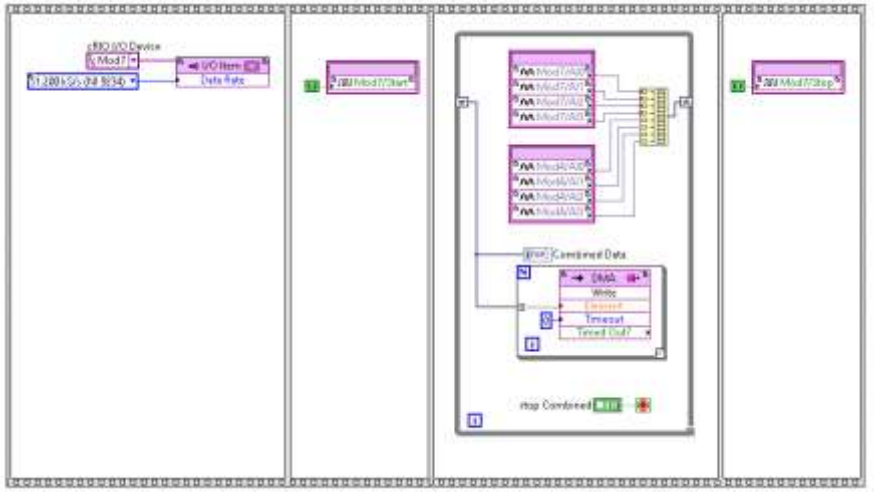
Configure FIFO properties

- Set Type to DMA
- Set Data Type

Place in FPGA App

- Drag FIFO from project to FPGA
- Drop in desired loop





Accessing the IO From in RT App

DMA Invoke Functions

Accesses FPGA DMA from RT App

- Complicated access
- Higher latency, highest bandwidth
- Lossless, waveform data
- Synchronous access

Configure DMA

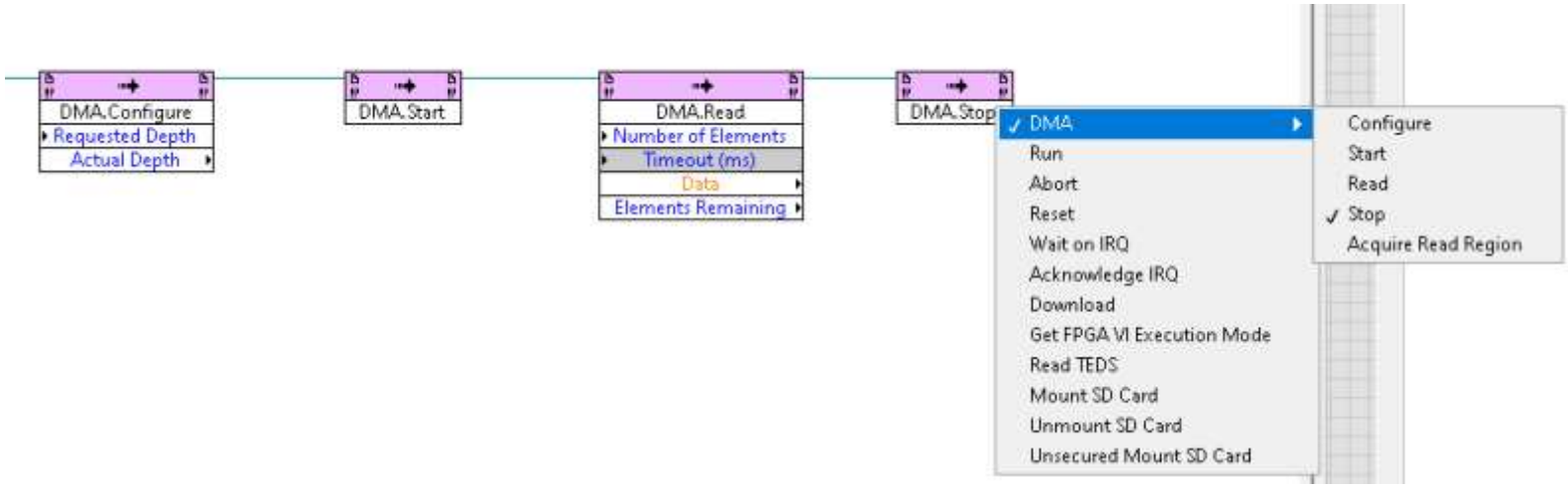
- Sets buffer (memory) size

Start

Read DMA

- Set number of elements to read & timeout
- Read elements, typically in a loop

Stop



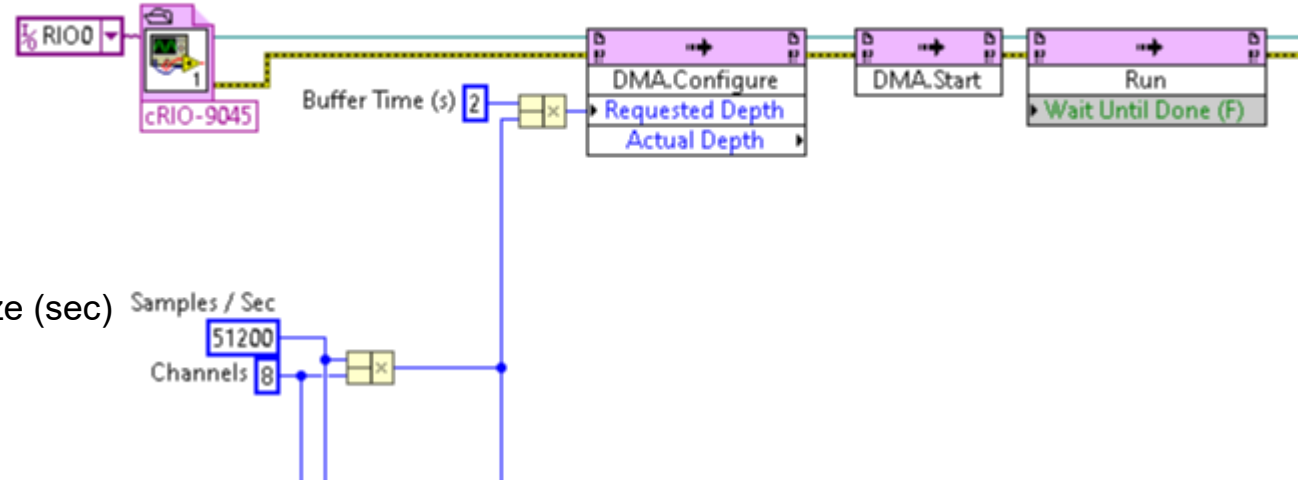
RT Block Diagram

Accessing the IO From in RT App

Host Side – DMA Best Practices

DMA Configuration

- Do not Run the FPGA with Open
- Configure DMA before running
- Depth = Channels * Acquired Samples/Channel * Desired Buffer size (sec)
- Start DMA
- Run the FPGA

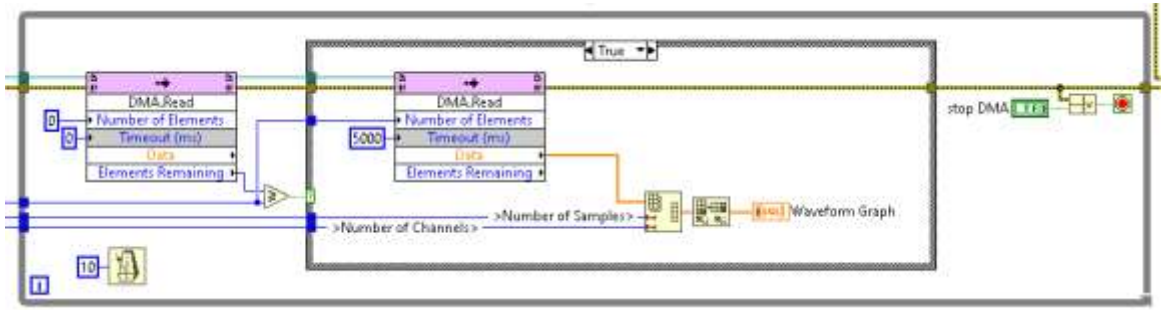


Accessing the IO From in RT App

Host Side – DMA Best Practices

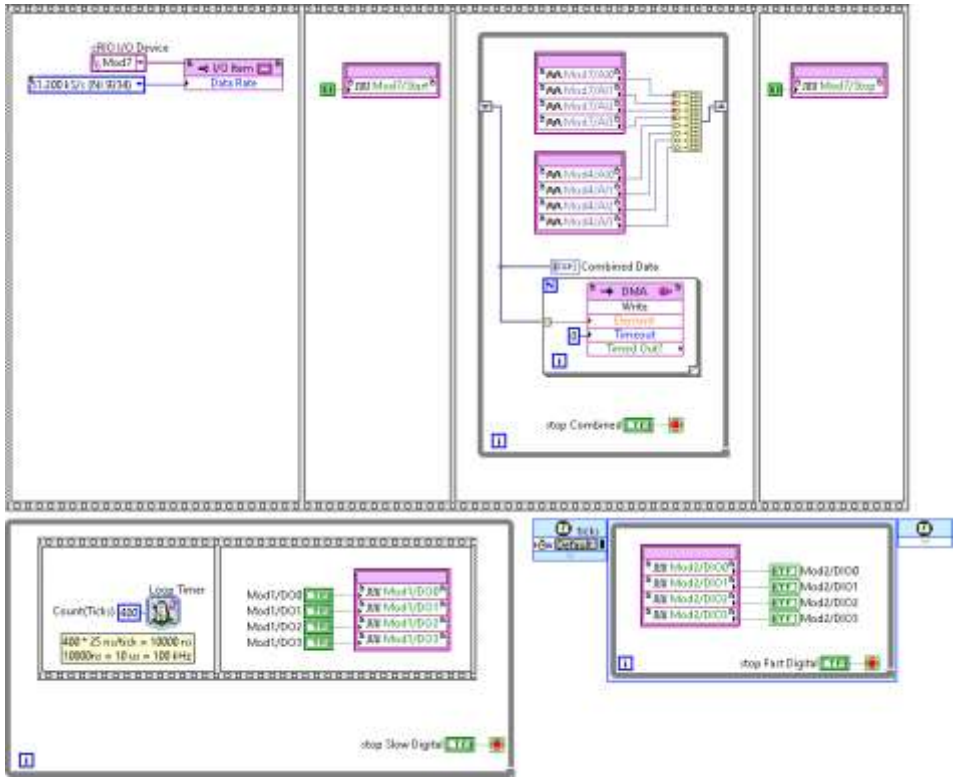
Reading DMA

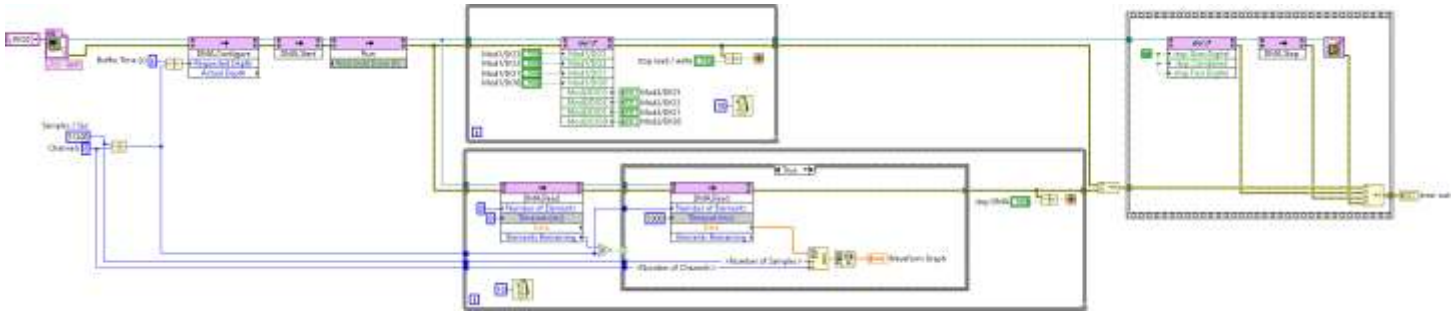
- Get number of elements in buffer (# of elements & timeout = 0)
- Compare Elements Remaining to (Channels * Acquired Samples/Channel)
- Do nothing if elements are not available (Loop waits)
- Read elements if they are available (Call DMA.Read again with Channels * Acquired Samples/Channel)
- Reshape Array (Size 0: Acquired Samples/Channel, Size 1:Channels)
- Transpose Array



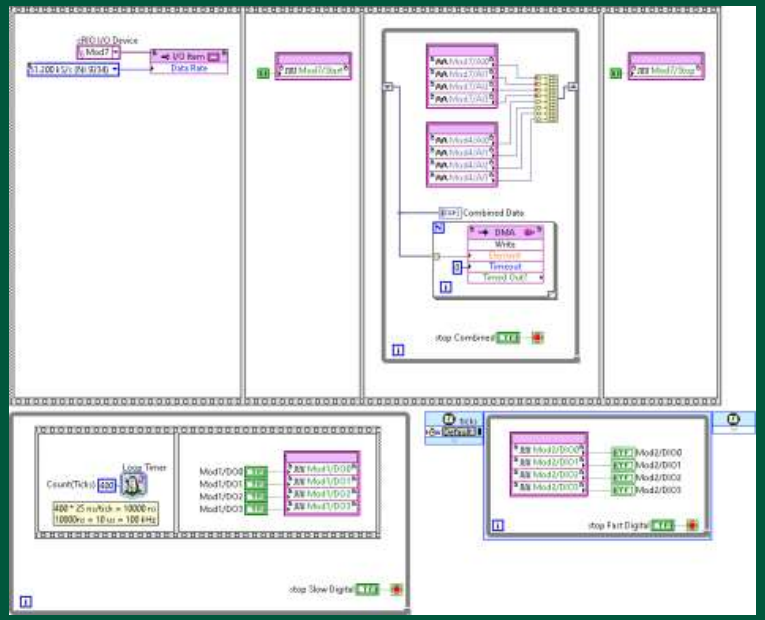
Basic Host + FPGA Example Walkthrough

Code at: cRIO Example Code\FPGA\Basic IO Access Example\





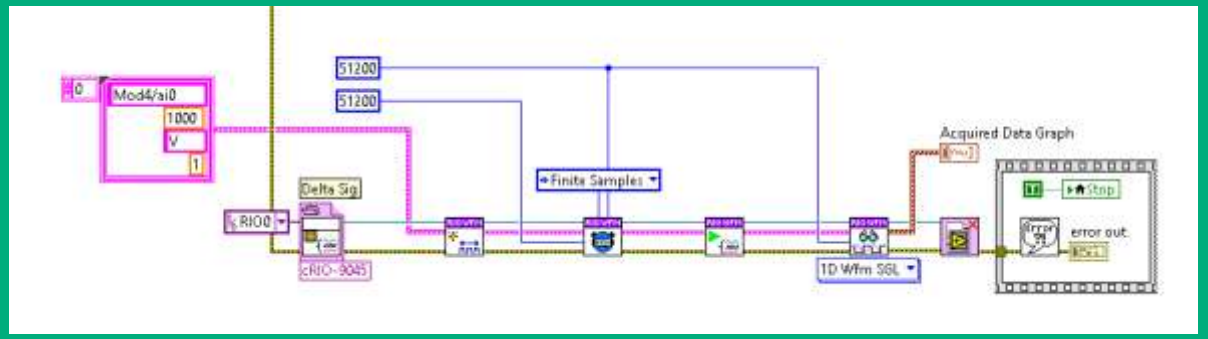
Summary: Basic Host + FPGA Example Walkthrough



Perform Low Level IO operations on FPGA

Communicate with RT OS

- Read/Write Controls
- Invoke Nodes → DMA

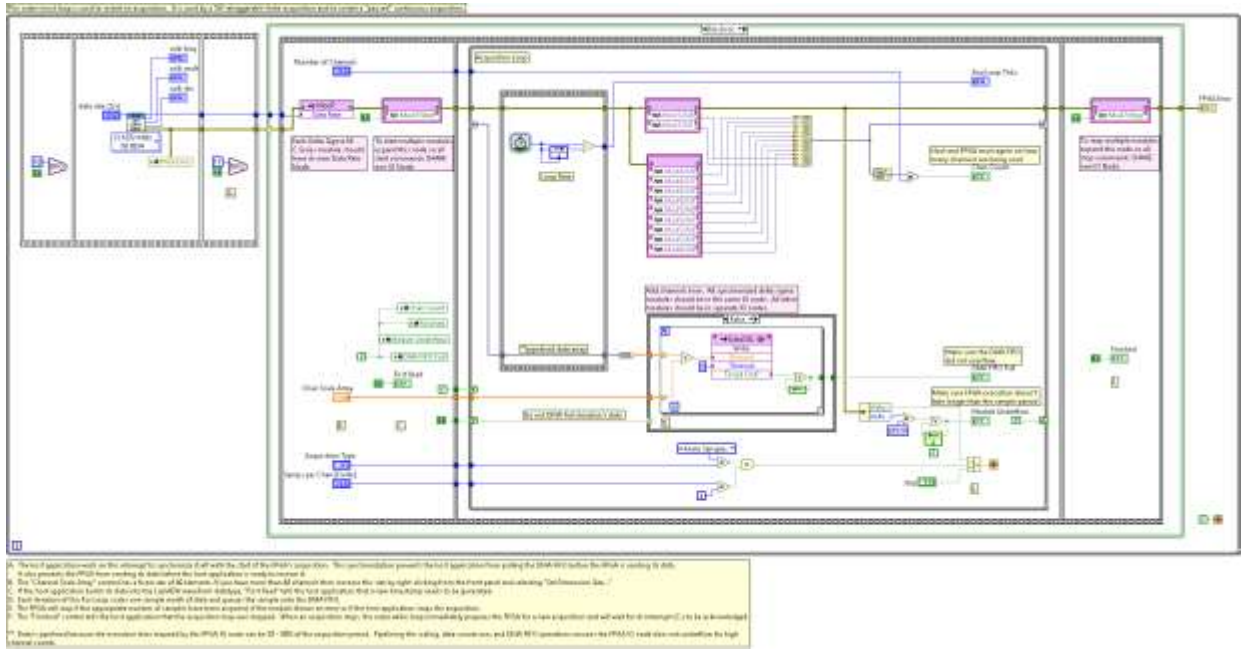


Demo: cRIO Waveform API Walkthrough

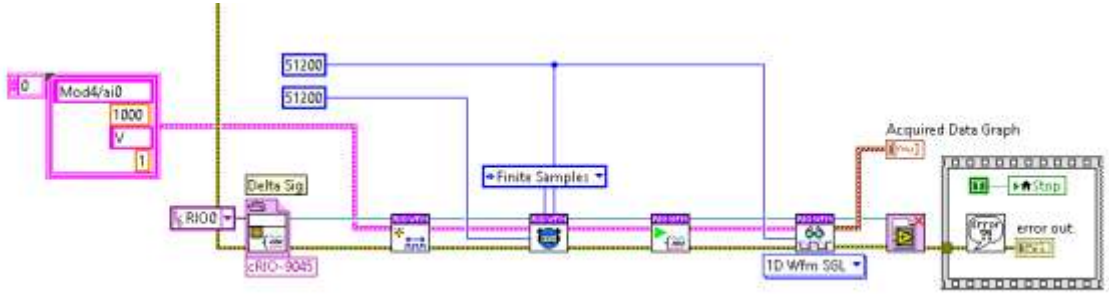
cRIO Waveform API – Easier Way

[Link: cRIO Waveform API]

Provides FPGA templates w/ DMA support

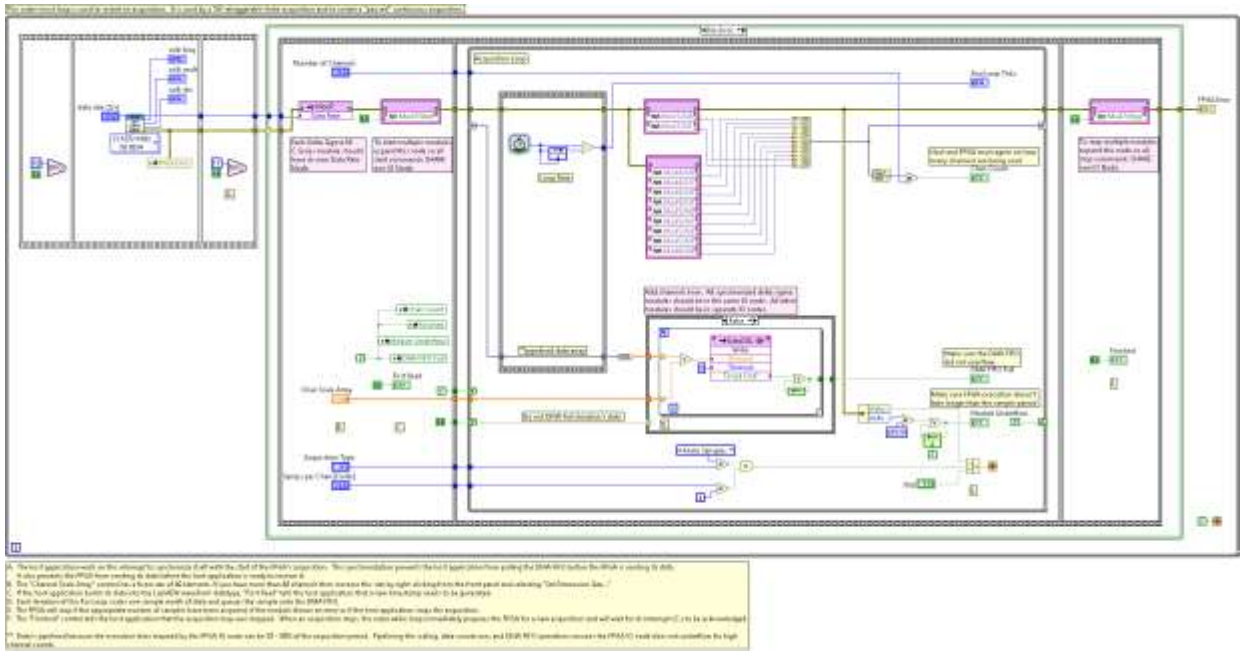


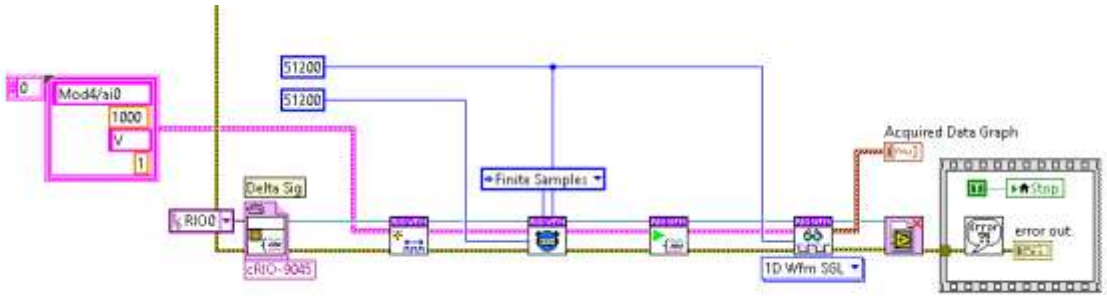
Provides easy Host API and Examples



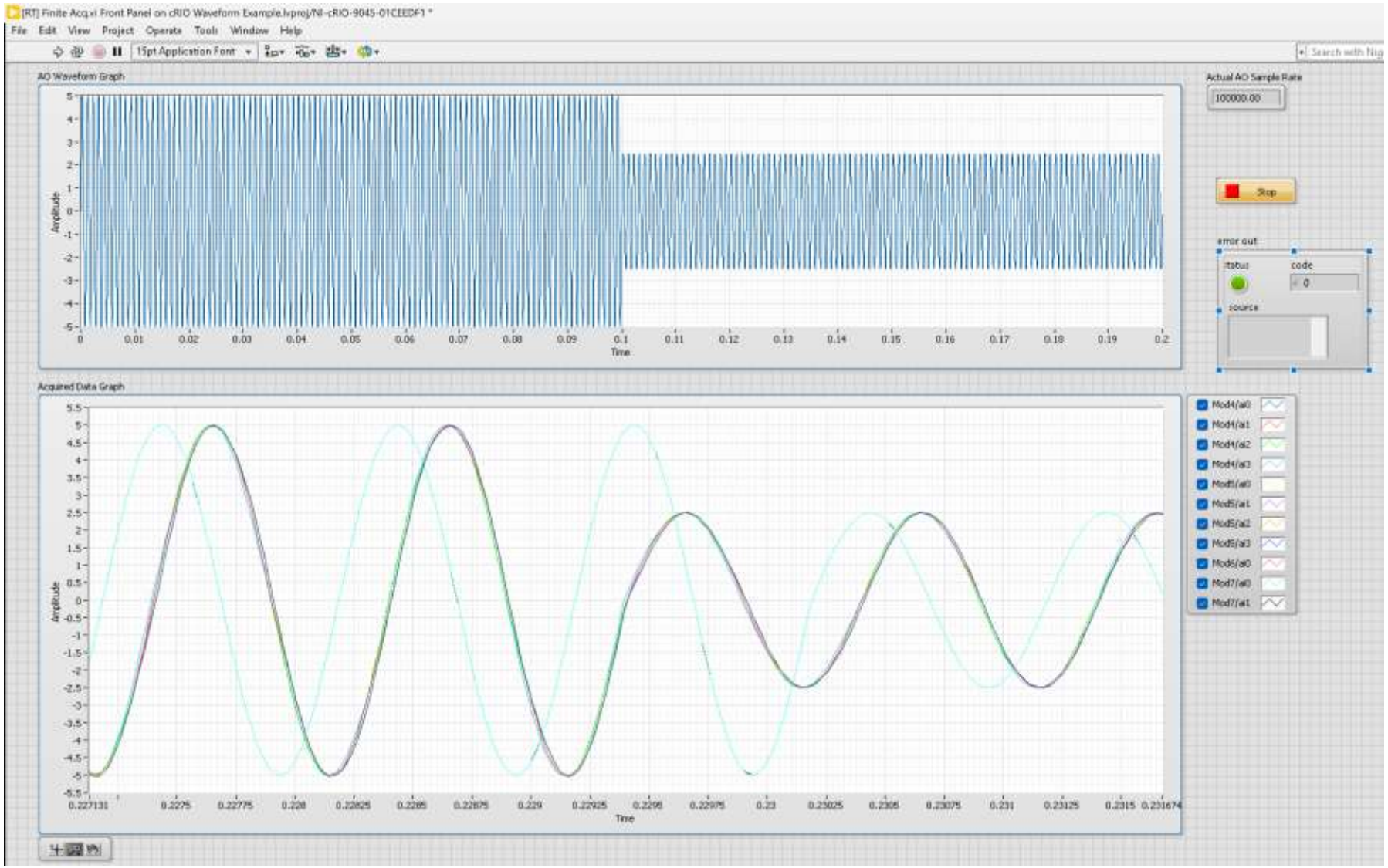
cRIO Waveform API – Example

Code at: cRIO Example Code\FPGA\cRIO Waveform Example

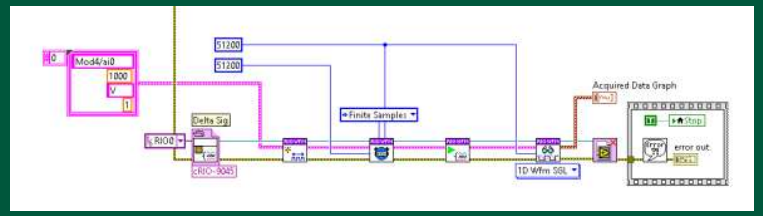




Synchronization vs NI DAQmx – Exact Same Result



Summary: cRIO Waveform API Walkthrough



FPGA waveforms made easier

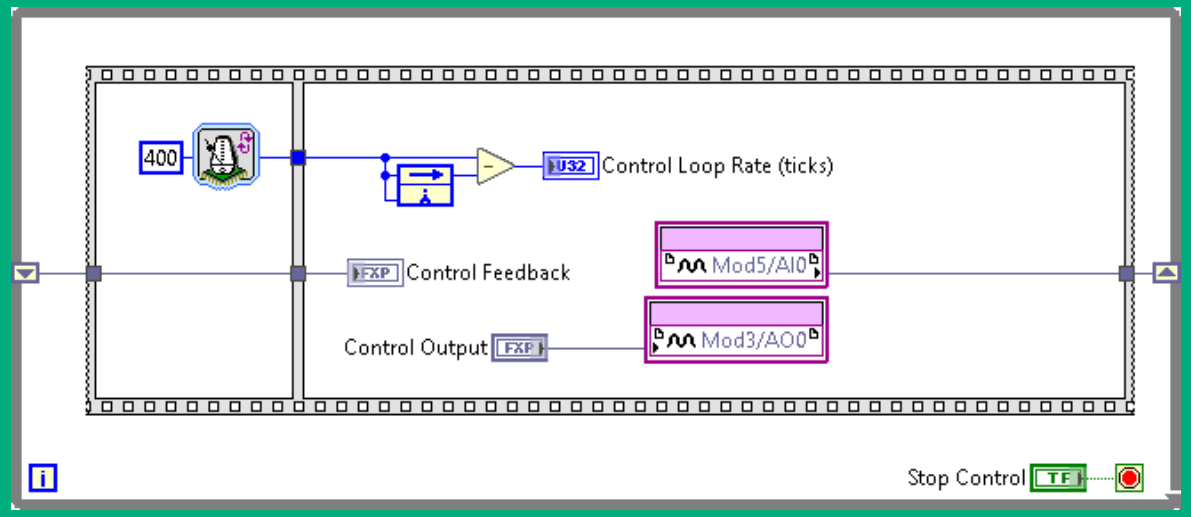
Communicate with RT OS

- Abstracted from the user
- Handles errors

Allows FPGA customization w/ NI DAQmx “like” experience

- Inline signal processing
- Split signal control + waveform acquisition

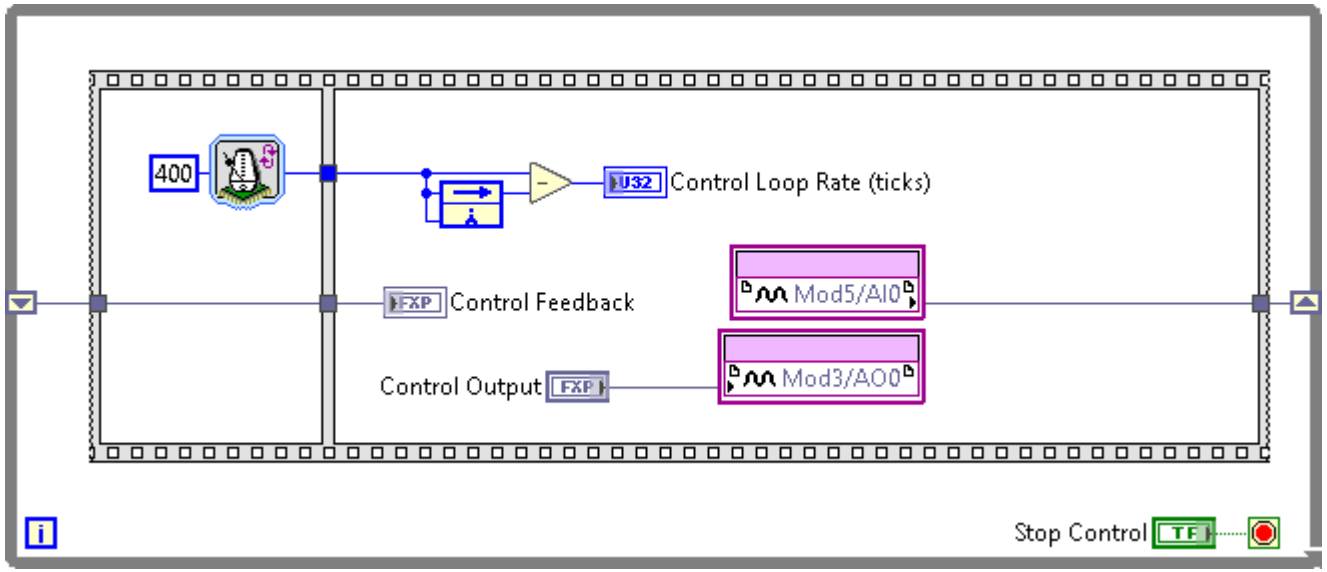
EMERSON



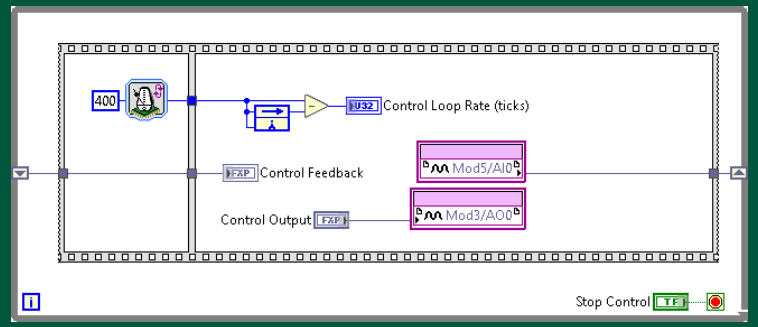
Demo: FPGA Control Example Walkthrough

Basic Host + FPGA Example Walkthrough

Code at: cRIO Example Code\FPGA\Basic Control Example

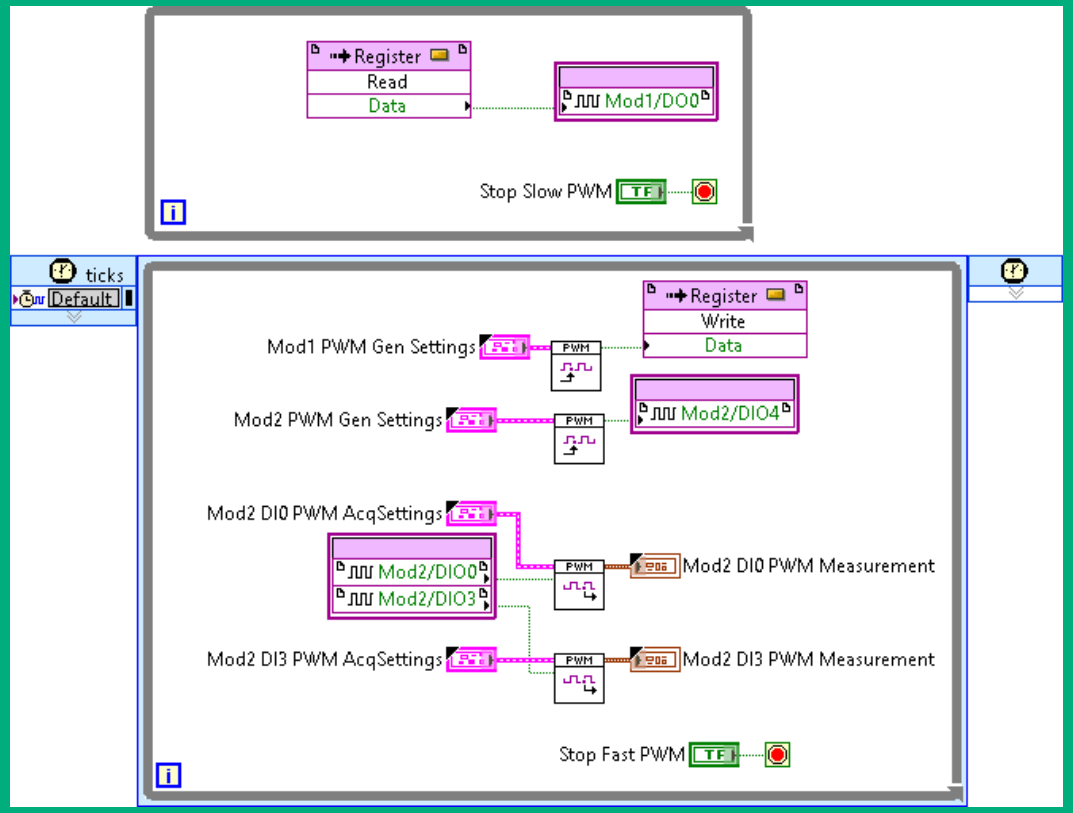


Summary: FPGA Control Example Walkthrough



High Speed Control – 100 kHz with Analog In / Out

All in Hardware – No OS needed, only power to cRIO

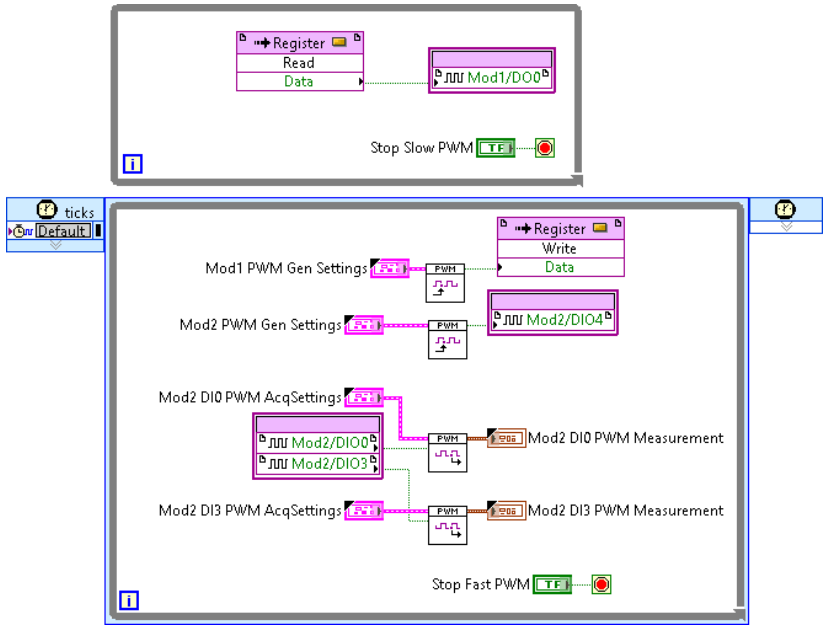


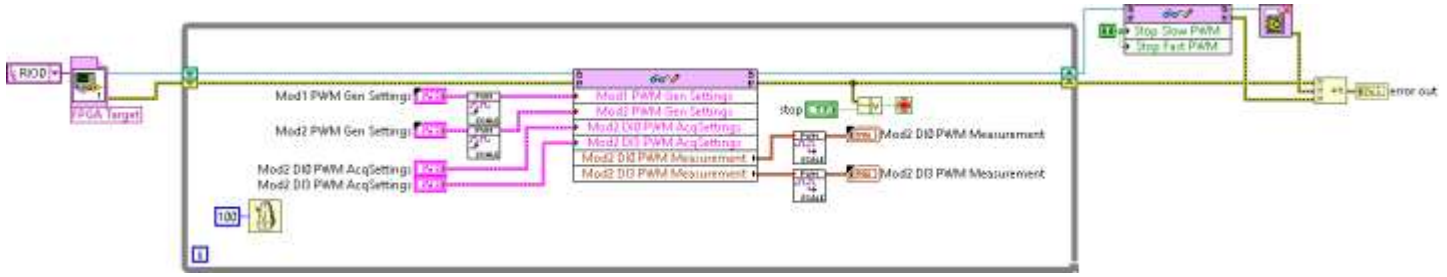
Demo: FPGA Counters Code Walkthrough

EMERSON

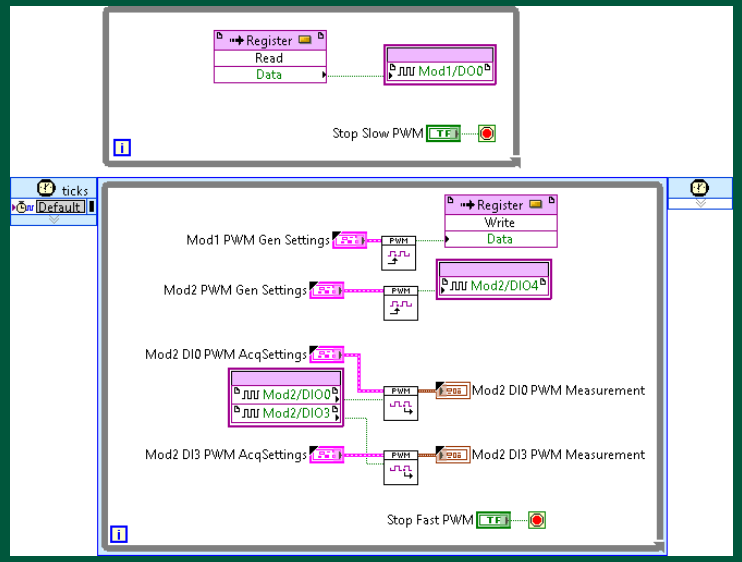
Basic Host + FPGA Example Walkthrough

Code at: cRIO Example Code\FPGA\Basic Control Example





Summary: FPGA Counters Code Walkthrough



FPGA is amazing at counters

- User creates them
- Typically uses RT for conversion (saves FPGA fabric)

Add any analysis per pin (e.g. period, frequency, duty cycle, active width, inactive width, etc)

- 8 Digital Inputs * 5 measurements types = 40 measurements

Again, FPGA “pin splitting” is awesome

Summary: FPGA IO on cRIO

Pros

- Lowest level of access
- Most flexible (in-line processing, parallel operations, pin splitting, etc)
- Fastest execution, control
- No operating system interaction required

Cons

- Harder to use (compared to other options, easier than VHDL)
- No (easy) IO expansion

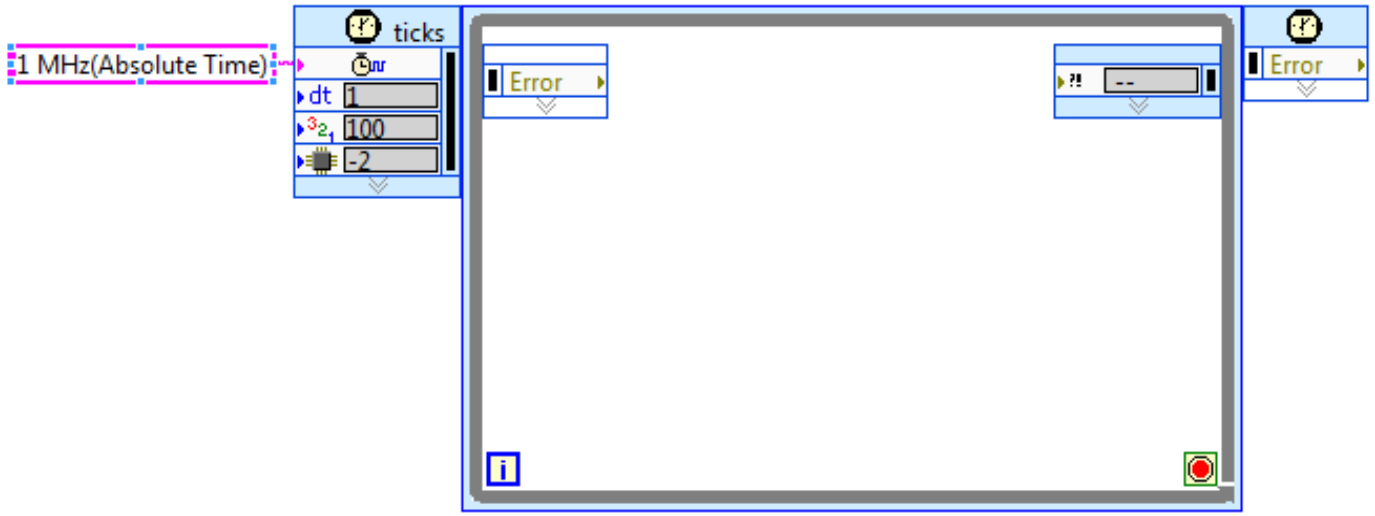
TSN Synchronization

Accessible everywhere

EMERSON

Access TSN Synchronization in LabVIEW Real-Time

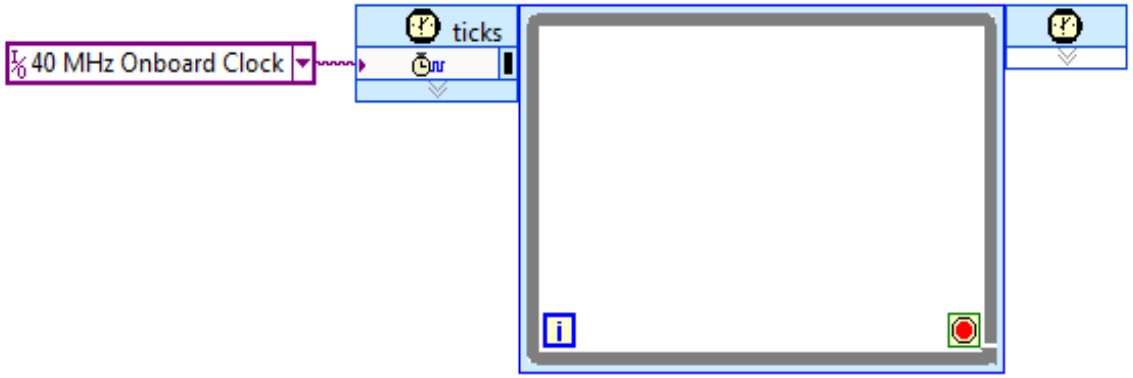
Synchronized time loops(using an absolute time base) disciplined to network time



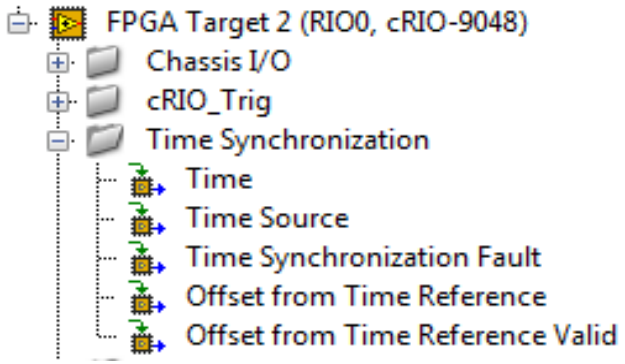
Get time functions return the current network time

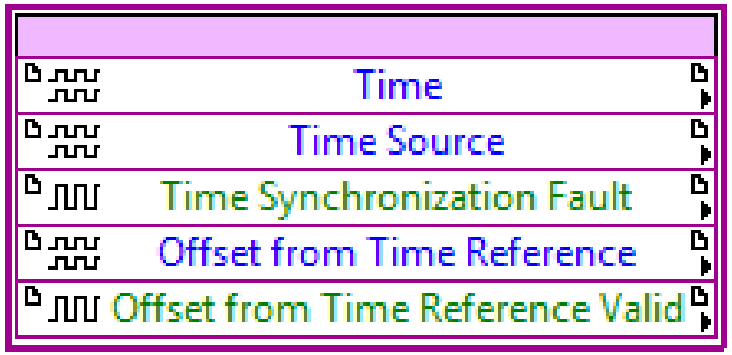
Access TSN Synchronization in LabVIEW FPGA

- FPGA clock disciplined to network time



- Time Synchronization Registers
  - Current Network Time
  - Time Source
  - Offset from Time Reference Valid
  - Fault





Recommendations

Best practices

Pick your module per slot for your application

Independent SAR ADC / DAC for high-speed control

MUX SAR ADC / DAC for slow speed acq or control

Don’t use delta sigma for high-speed control due to sample delay

Use Delta sigma for waveform logging

Best Practices

Use the right API

High speed analog and static digital data logging – NI DAQmx

Slow speed control – Scan Engine

Safety; high speed control; high speed digital processing (PWM, period, frequency) & comms, signal processing – FPGA

Best practices

TSN (802.1AS) is amazing…

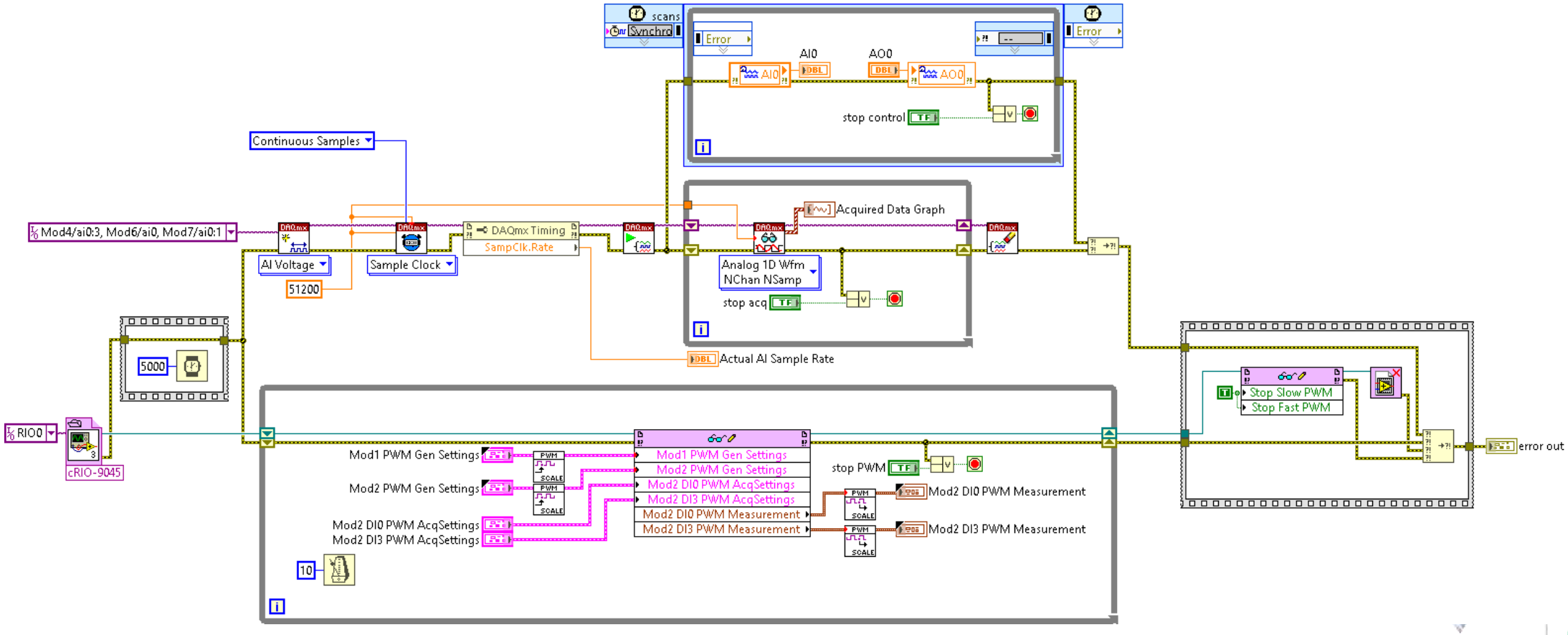
You get it for “free”

NI DAQmx has best overall experience

Demo: Using all Three APIs

Using all Three APIs

Code at: cRIO Example Code\Final App



Summary: Using all Three APIs

Design your application per slot!!

Questions?

Daniel Eaton

Chief Field Applications Engineer

Daniel.eaton@emerson.com

We are here to help!

Pull us in to design conversations

Let us help de-risk projects

- Tell us what you want to learn about next
  - Network Communication
  - FPGA Optimization
  - PXI

THANK YOU!!