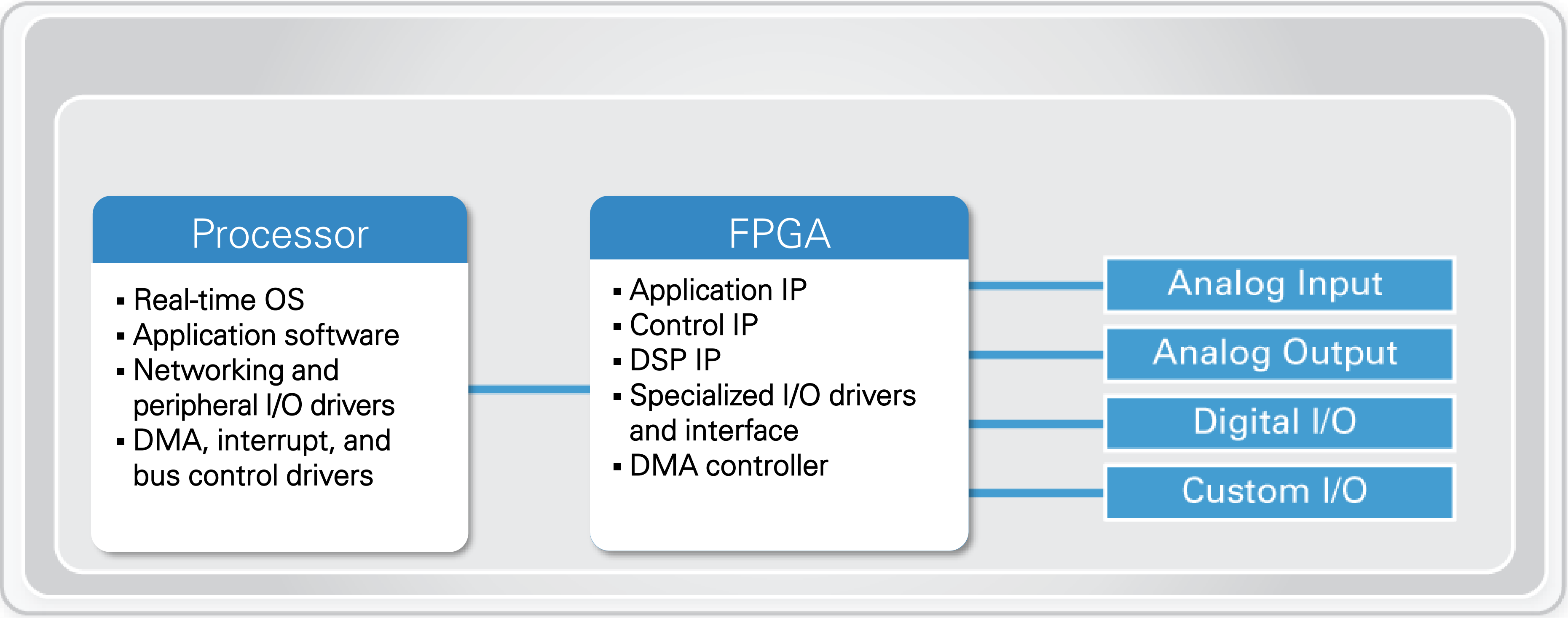
FPGA design with National Instuments

Rémi DA SILVA

Systems Engineer - Embedded and Data Acquisition Systems - MED Region

The NI Approach to Flexible Hardware



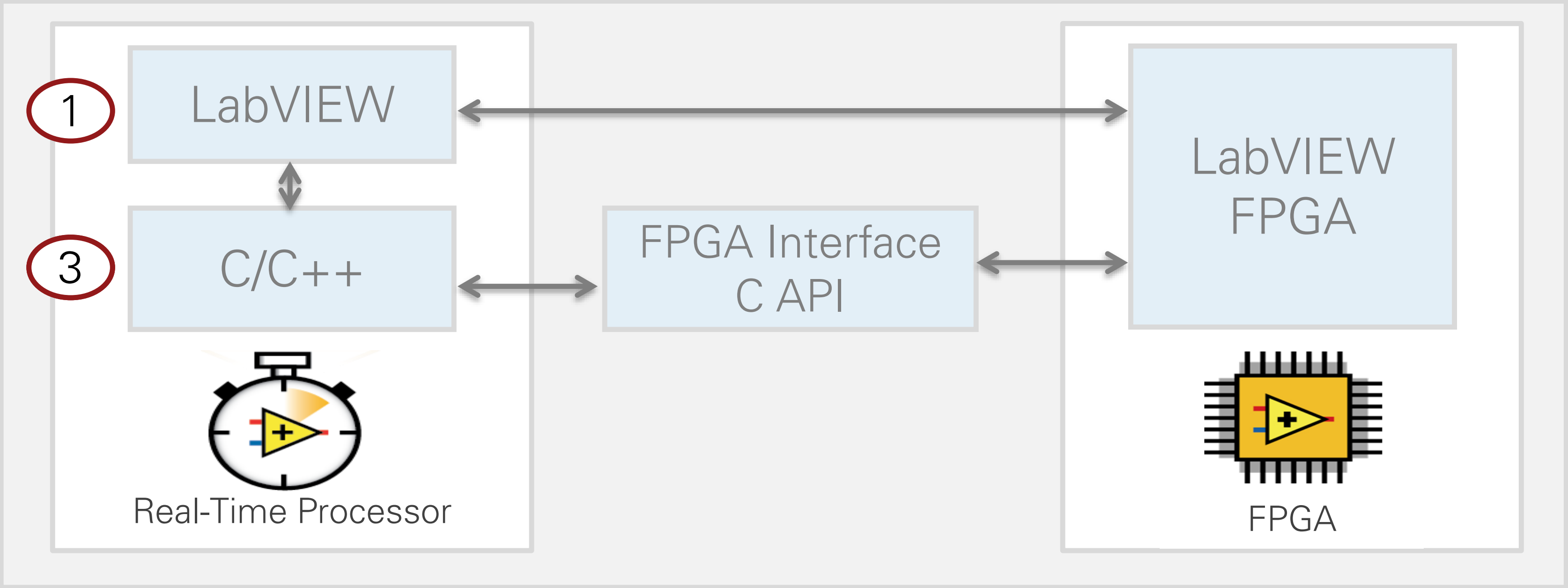
Processor

- Real-time OS
- Application software
- Networking and peripheral I/O drivers
- DMA, interrupt, and bus control drivers

FPGA

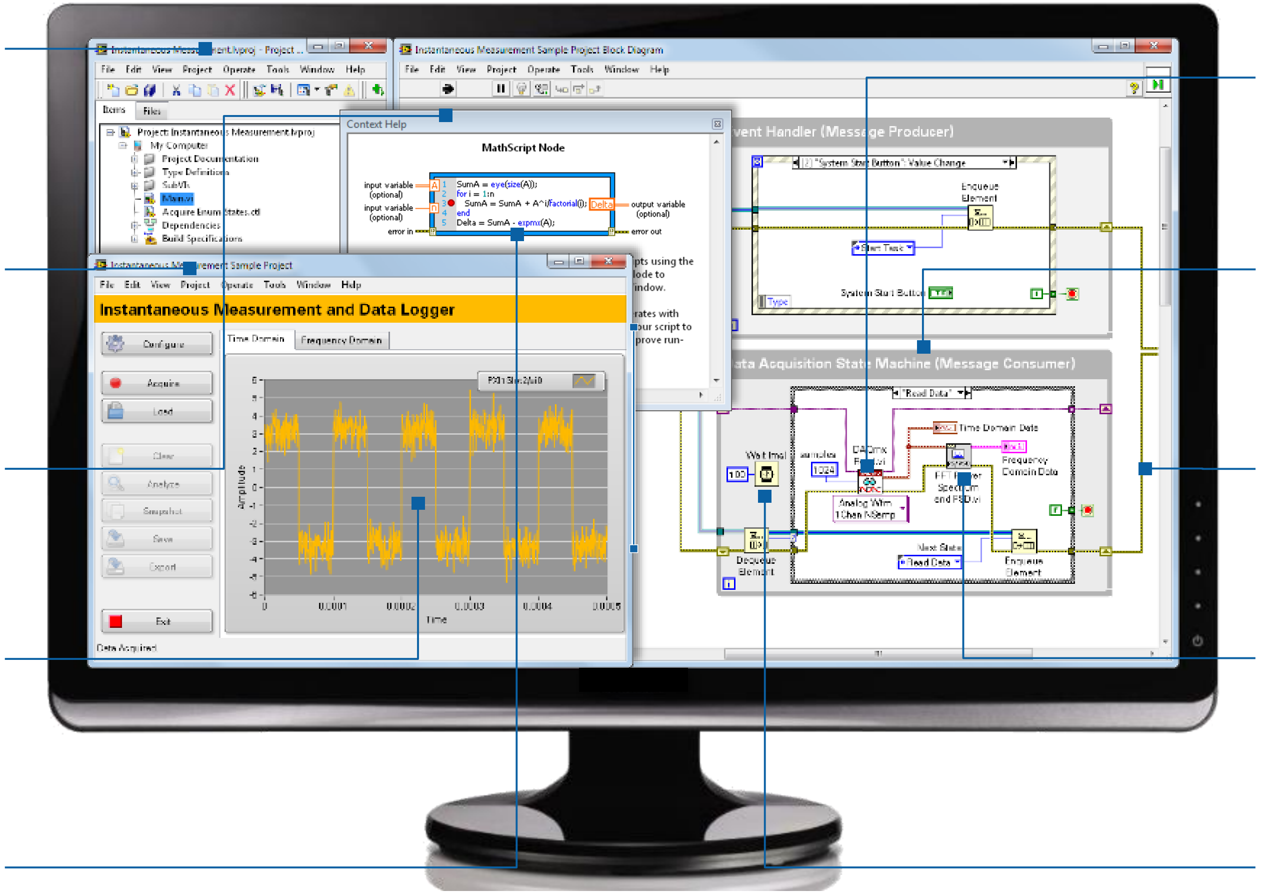
- Application IP
- Control IP
- DSP IP
- Specialized I/O drivers and interface
- DMA controller

NI Embedded Software Architecture Options



1. LabVIEW RT and FPGA
2. LabVIEW RT app for I/O, with C/C++ app or library
3. C/C++ on RT, LabVIEW FPGA

LabVIEW System Design Software



```
SumA = eye(size(A));
for i = 1:n
  SumA = SumA + A^i/factorial(i);
end
Delta = SumA - expm(A);
```

Instantaneous Measurement Sample Project Block Diagram
Context Help
MathScript Node
input variable (optional)
output variable (optional)
error in
error out
Event Handler (Message Producer)
Data Acquisition State Machine (Message Consumer)
Instantaneous Measurement and Data Logger
Configure
Acquire
Load
Clear
Analyze
Snapshot
Save
Export
Exit
Time Domain
Frequency Domain
Amplitude
Time
Data Acquired
Enqueue Element
Dequeue Element
Time Domain Data
Frequency Domain Data
Read Data

Project Explorer

Manage and organize all system resources, including I/O and deployment targets

Deployment Targets

Deploy LabVIEW code to the leading desktop, real-time, and FPGA hardware targets

Instant Compilation

See the state of your application at all times, instantly

Front Panel

Create event-driven user interfaces to control systems and display measurements

Models of Computation

Combine and reuse .m files, C code, and HDL with graphical code

Hardware Connectivity

Bring real-world signals into LabVIEW from any I/O on any instrument

Parallel Programming

Create independent loops that automatically execute in parallel

Block Diagram

Define and customize the behavior of your system using graphical programming

Analysis Libraries

Use high-performance analysis libraries designed for engineering and science

Timing

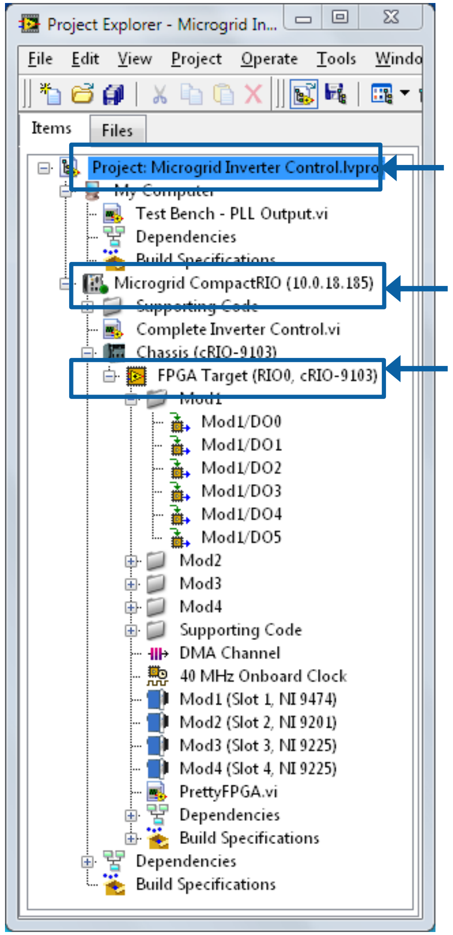
Define explicit execution order and timing with sequential data flow

Accelerates Your Success

By abstracting low-level complexity and integrating all of the tools you need to build any measurement or control system

LabVIEW System Development Environment

Complete
System IDE



```
Project: Microgrid Inverter Control.lvproj
  My Computer
    Test Bench - PLL Output.vi
    Dependencies
    Build Specifications
  Microgrid CompactRIO (10.0.18.185)
    Supporting Code
    Complete Inverter Control.vi
    Chassis (cRIO-9103)
      FPGA Target (RIO0, cRIO-9103)
        Mod1
          Mod1/DO0
          Mod1/DO1
          Mod1/DO2
          Mod1/DO3
          Mod1/DO4
          Mod1/DO5
        Mod2
        Mod3
        Mod4
        Supporting Code
        DMA Channel
        40 MHz Onboard Clock
        Mod1 (Slot 1, NI 9474)
        Mod2 (Slot 2, NI 9201)
        Mod3 (Slot 3, NI 9225)
        Mod4 (Slot 4, NI 9225)
        PrettyFPGA.vi
        Dependencies
        Build Specifications
    Dependencies
    Build Specifications
```

Windows Desktop PC Application

Real-Time Processor Application

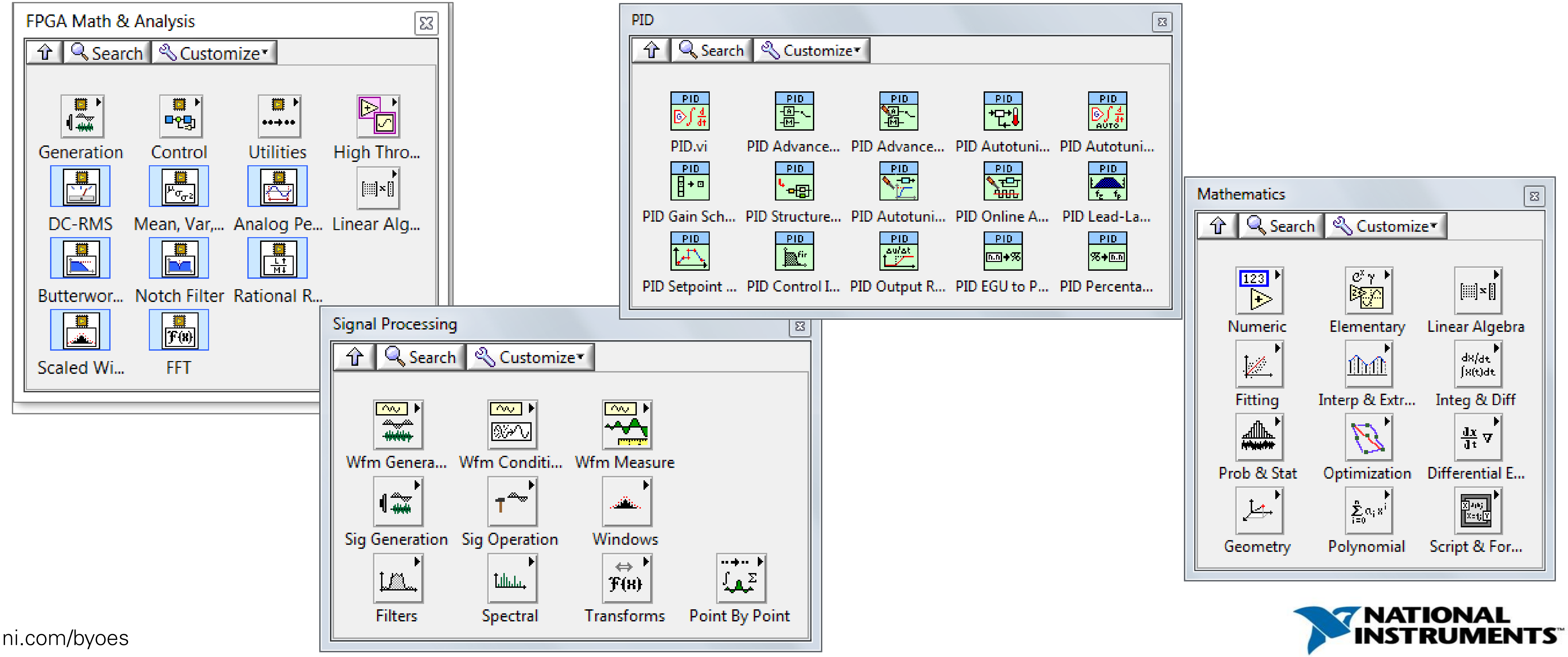
FPGA Application

System Design Tool

LabVIEW System Development Environment

Complete System IDE

Math and Analysis

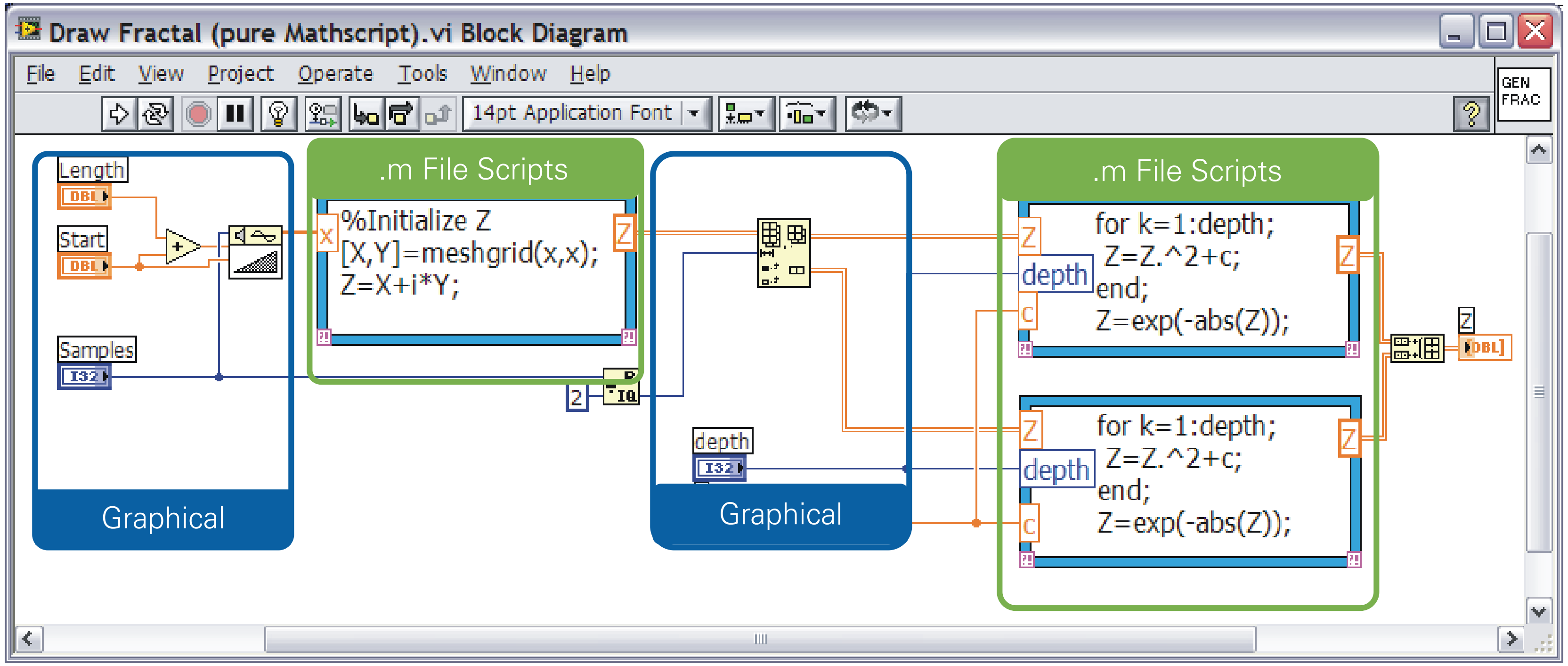


LabVIEW System Development Environment

Complete System IDE

Math and Analysis

Reuse of Existing Code



```
%Initialize Z
[X,Y]=meshgrid(x,x);
Z=X+i*Y;
```

```
for k=1:depth;
 Z=Z.^2+c;
end;
Z=exp(-abs(Z));
```

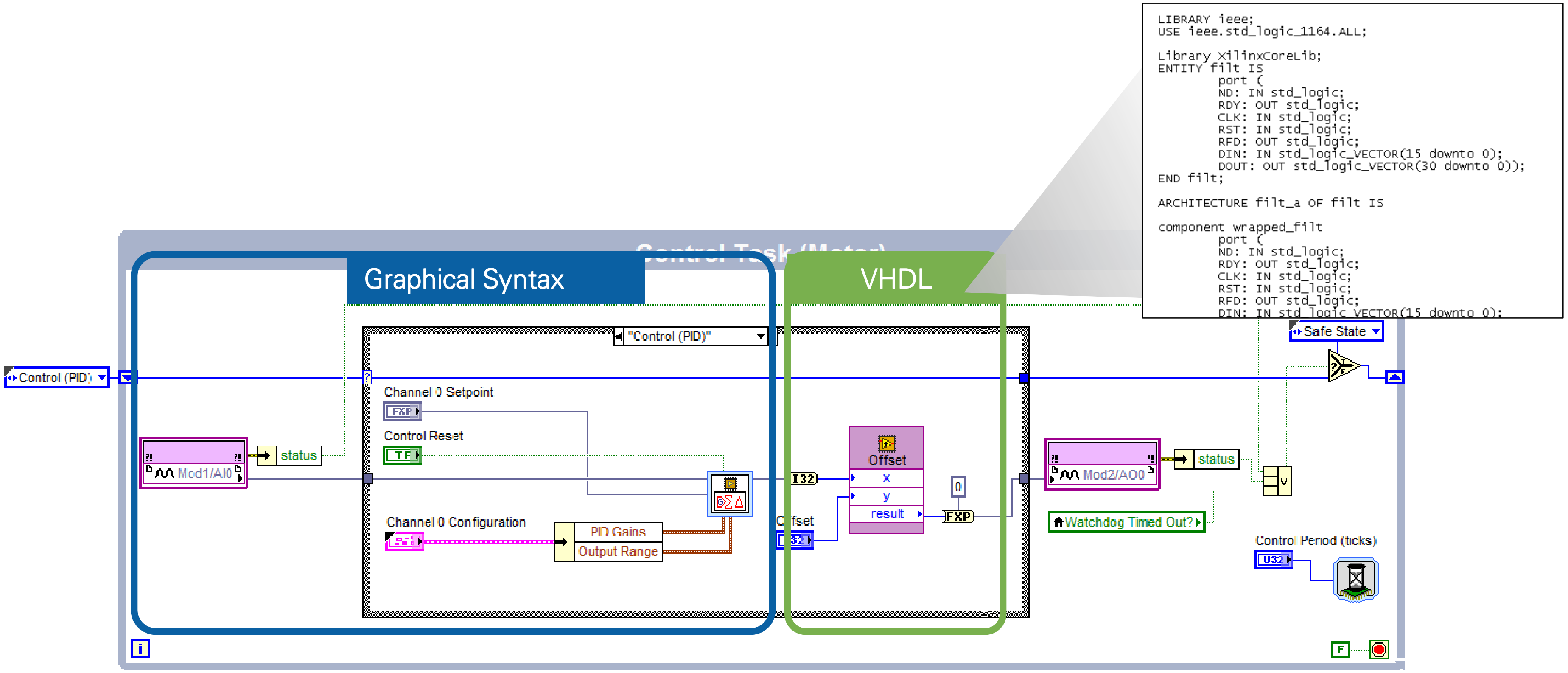
```
for k=1:depth;
 Z=Z.^2+c;
end;
Z=exp(-abs(Z));
```

LabVIEW System Development Environment

Complete System IDE

Math and Analysis

Reuse of Existing Code



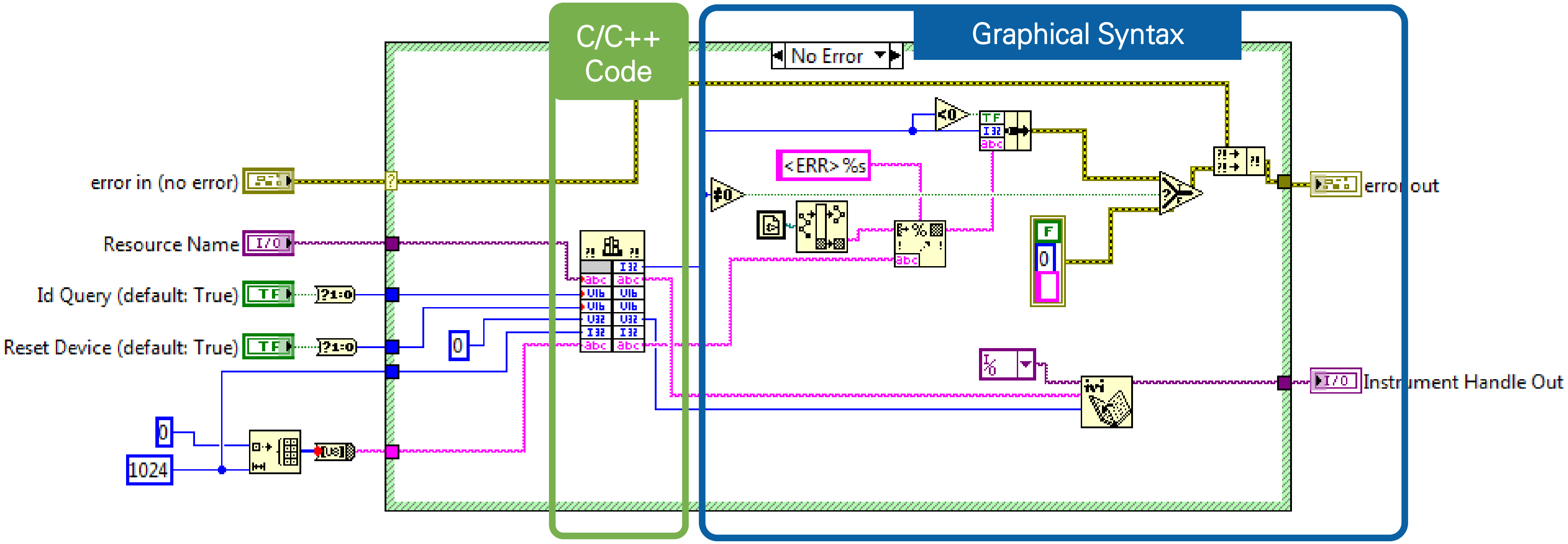
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

Library XilinxCoreLib;
ENTITY filt IS
        port (
        ND: IN std_logic;
        RDY: OUT std_logic;
        CLK: IN std_logic;
        RST: IN std_logic;
        RFD: OUT std_logic;
        DIN: IN std_logic_VECTOR(15 downto 0);
        DOUT: OUT std_logic_VECTOR(30 downto 0));
END filt;

ARCHITECTURE filt_a OF filt IS

component wrapped_filt
        port (
        ND: IN std_logic;
        RDY: OUT std_logic;
        CLK: IN std_logic;
        RST: IN std_logic;
        RFD: OUT std_logic;
        DIN: IN std_logic_VECTOR(15 downto 0);
```

LabVIEW System Development Environment

Complete System IDE

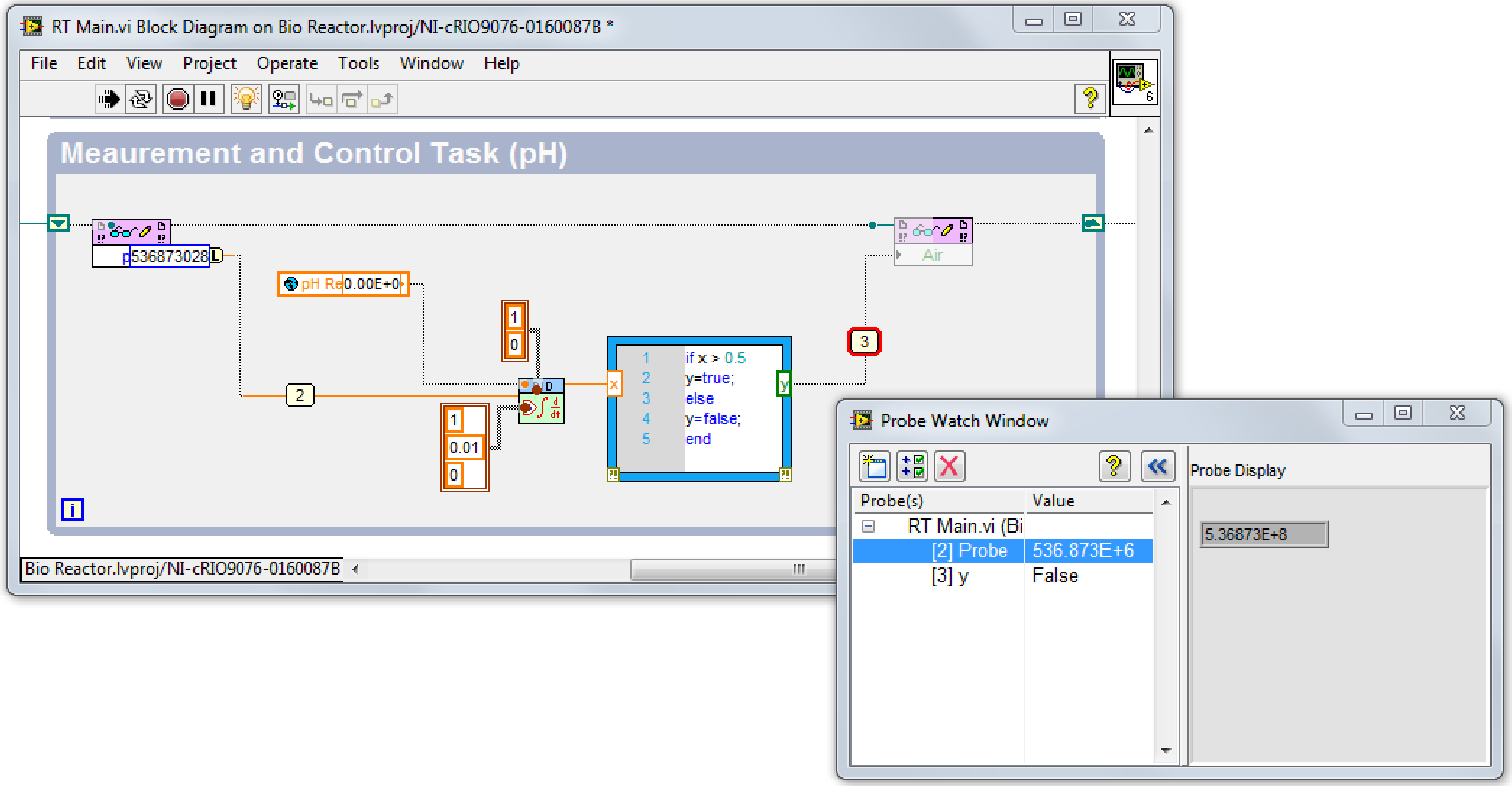
Math and Analysis

Reuse of Existing Code



LabVIEW System Development Environment

Complete System IDE	Math and Analysis	Reuse of Existing Code	Graphical Debugging



RT Main.vi Block Diagram on Bio Reactor.lvproj/NI-cRIO9076-0160087B *
File Edit View Project Operate Tools Window Help
Meaurement and Control Task (pH)
536873028
pH Re 0.00E+0
Air
Bio Reactor.lvproj/NI-cRIO9076-0160087B
Probe Watch Window
Probe Display

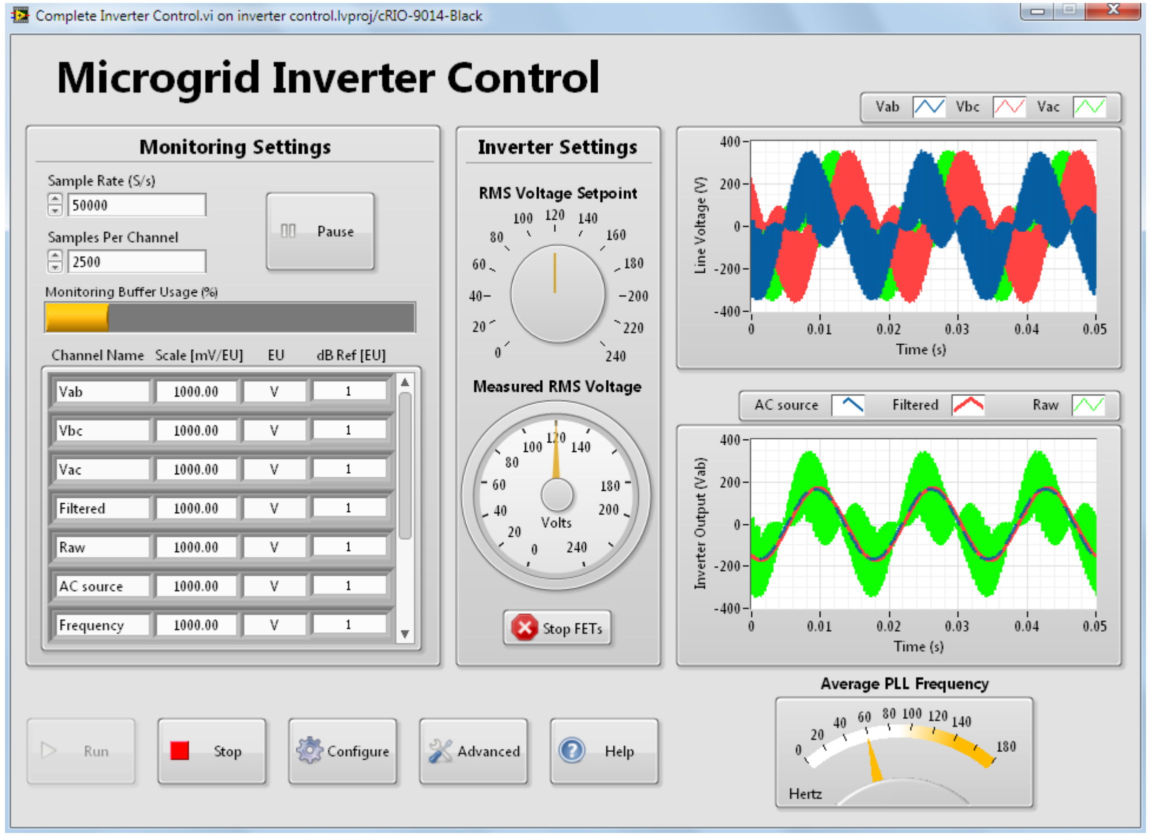
```
if x > 0.5
y=true;
else
y=false;
end
```

Probe(s)	Value
RT Main.vi (Bi	
[2] Probe	536.873E+6
[3] y	False

5.36873E+8

LabVIEW System Development Environment

Complete System IDE	Math and Analysis	Reuse of Existing Code	Graphical Debugging	User Interface



Complete Inverter Control.vi on inverter control.lvproj/cRIO-9014-Black

Microgrid Inverter Control

Monitoring Settings

Sample Rate (S/s): 50000

Samples Per Channel: 2500

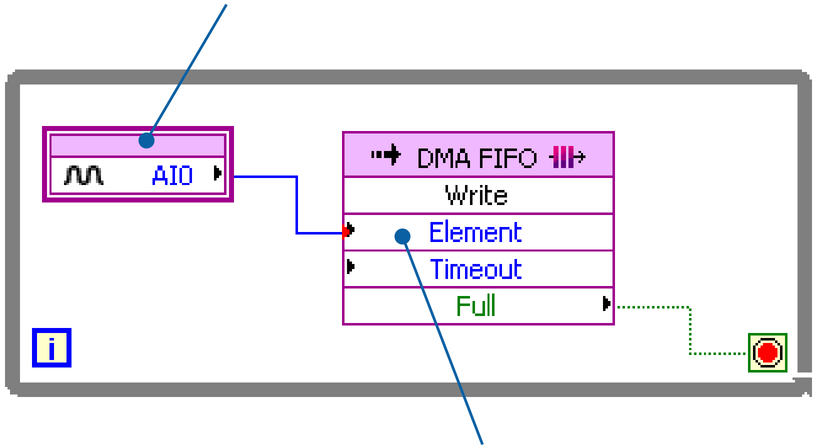
Monitoring Buffer Usage (%)

Channel Name	Scale [mV/EU]	EU	dB Ref [EU]
Vab	1000.00	V	1
Vbc	1000.00	V	1
Vac	1000.00	V	1
Filtered	1000.00	V	1
Raw	1000.00	V	1
AC source	1000.00	V	1
Frequency	1000.00	V	1

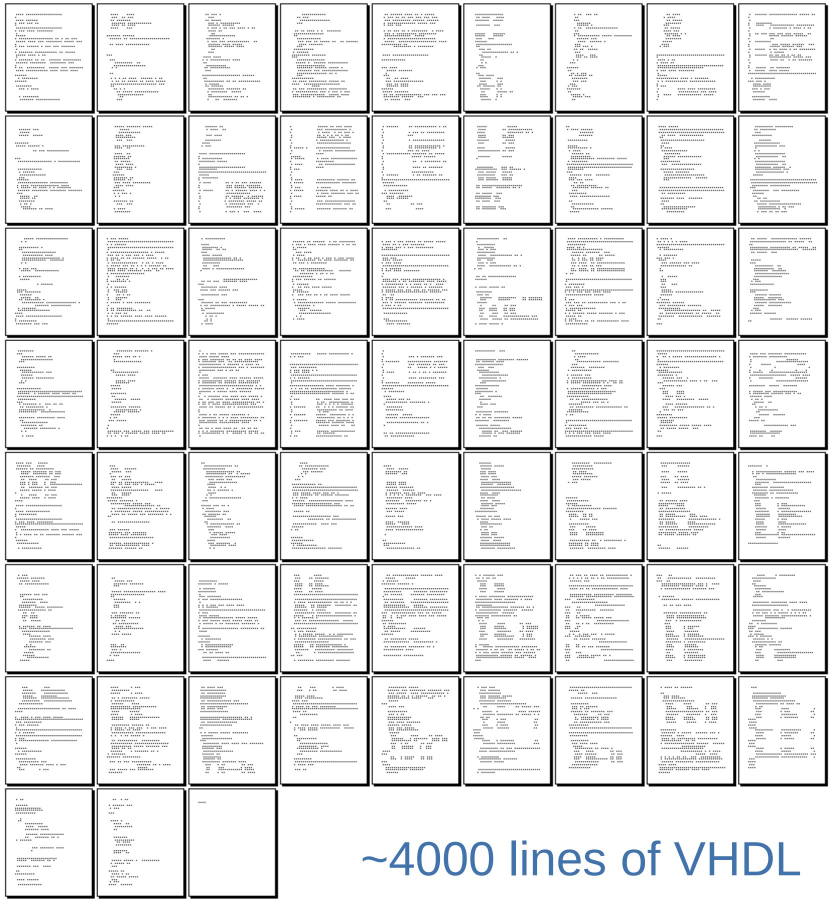
Inverter Settings

Abstraction of Hardware Complexities

Acquire analog data point-by-point



Directly transfer analog data to processor memory via FIFO for data logging, display, etc.



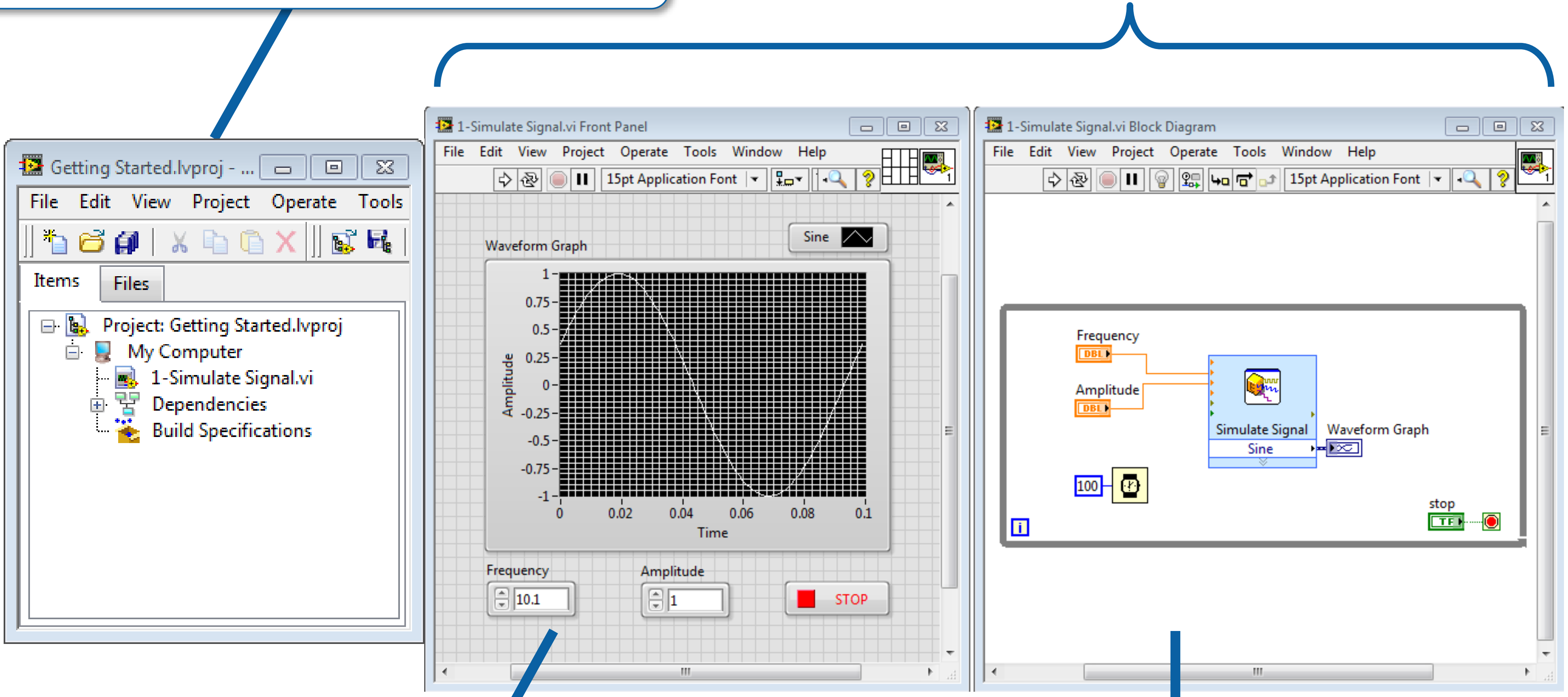
~4000 lines of VHDL

LabVIEW FPGA vs. VHDL

LabVIEW Environment Basics

“Project” = System Configuration

“VI” = Program or Function

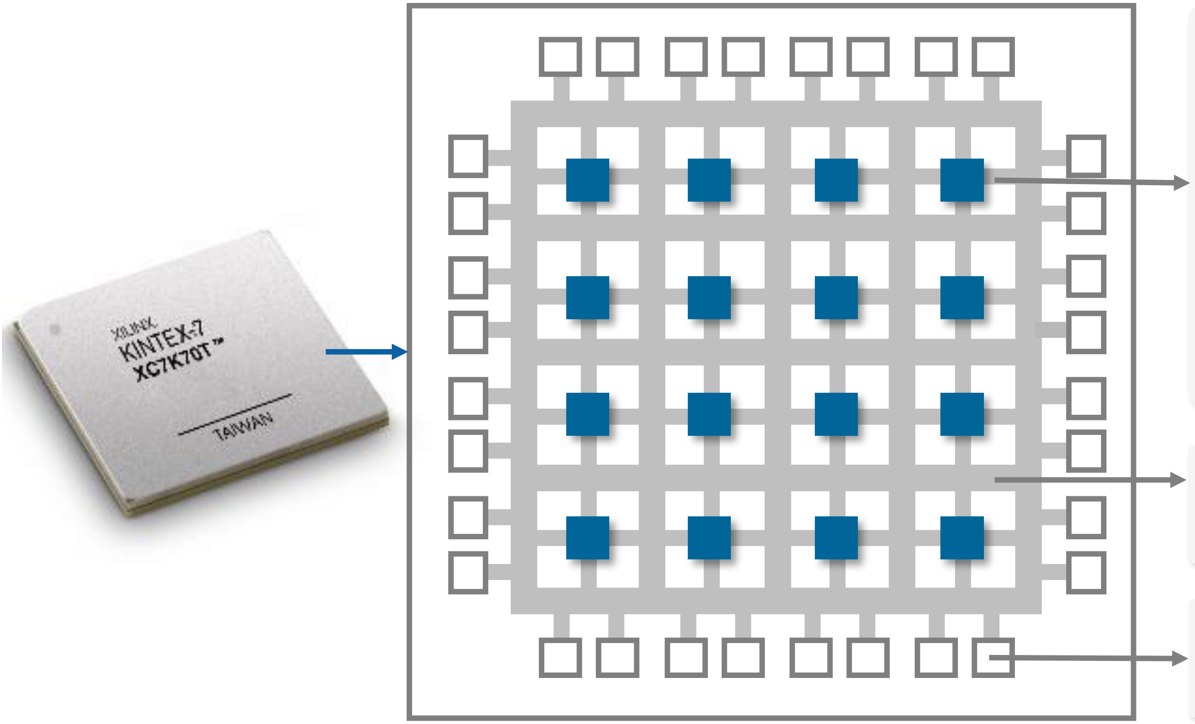


“Front Panel” = User Interface

“Block Diagram” = Code

Embedded systems – LabVIEW FPGA

Field-Programmable Gate Array (FPGA)



Memory Blocks
Store data sets or values in user defined RAM

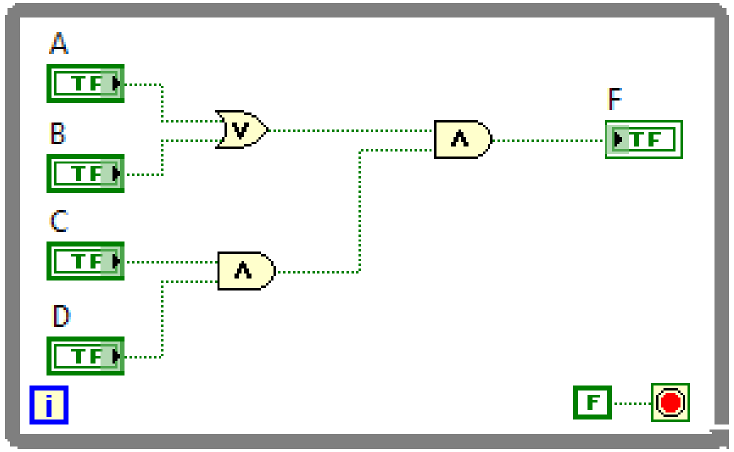
Configurable Logic Blocks (CLBs)
Implement logic using flip-flops and LUTs

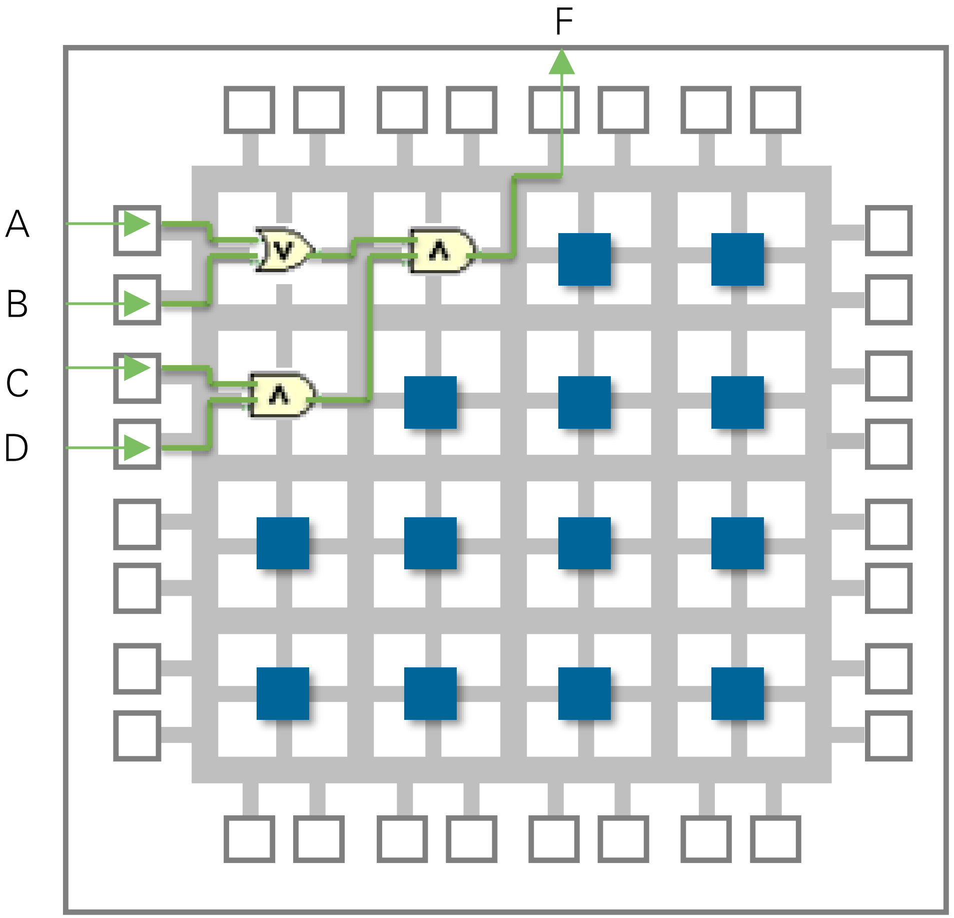
Multipliers and DSPs
Implement signal processing using multiplier and multiplier-accumulate circuitry

Programmable Interconnects
Route signals through the FPGA matrix

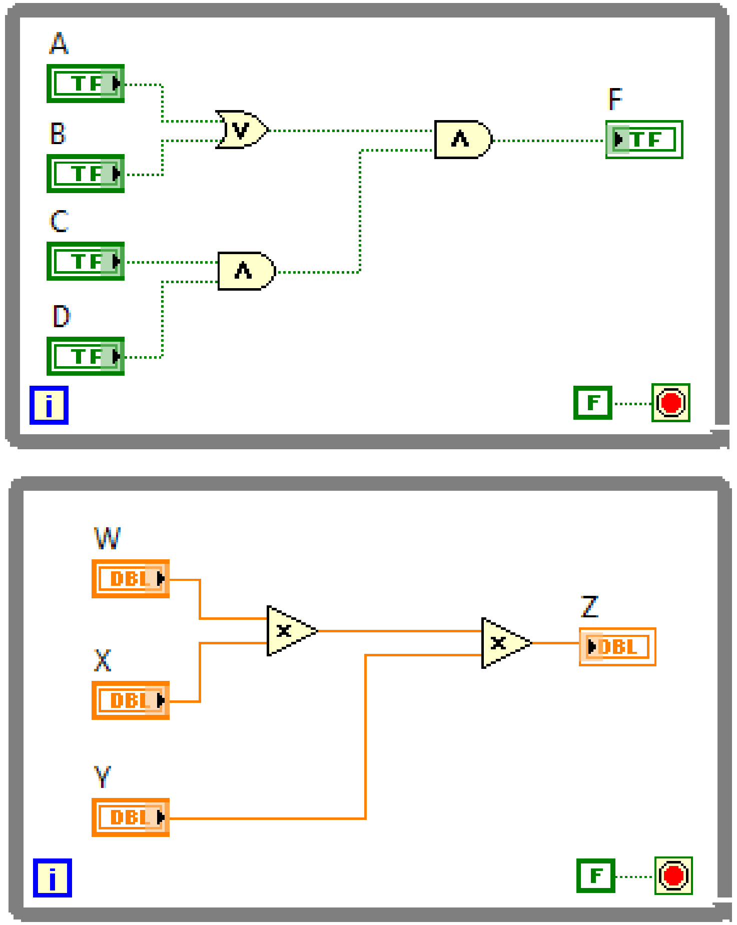
I/O Blocks
Directly access digital and analog I/O

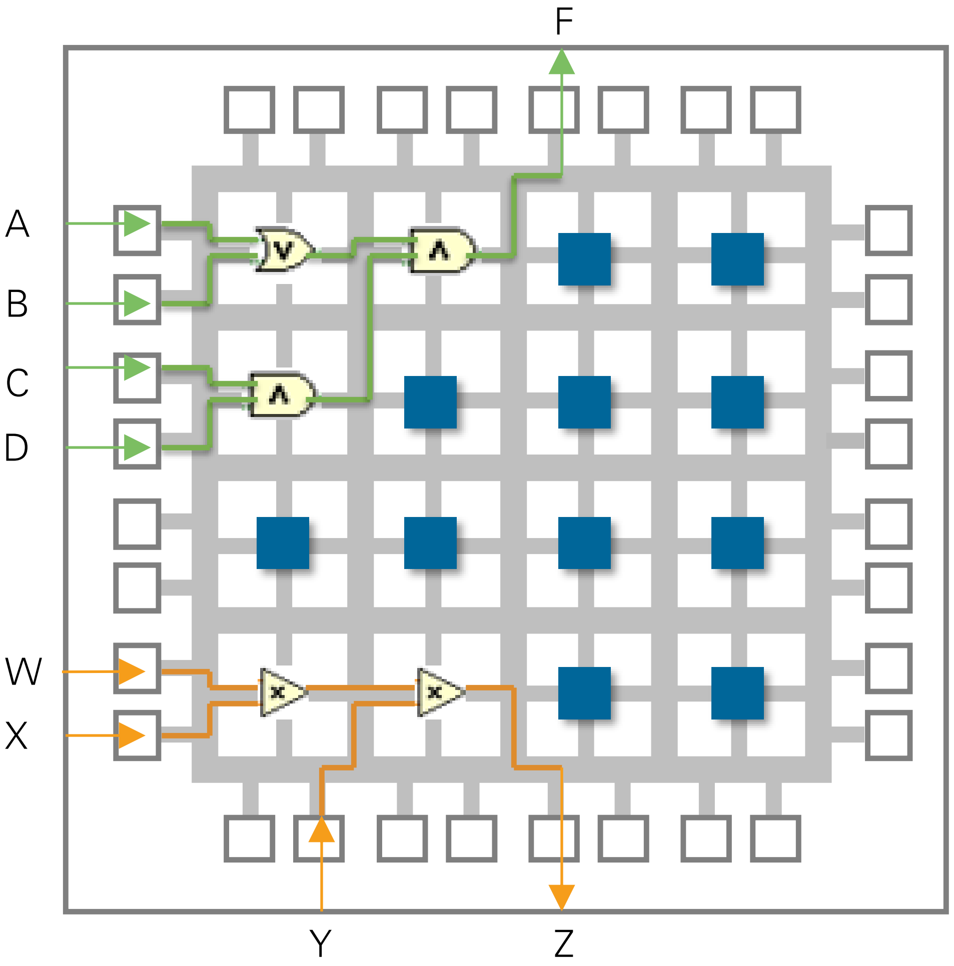
FPGAs Are Dataflow Systems





Parallel Processing

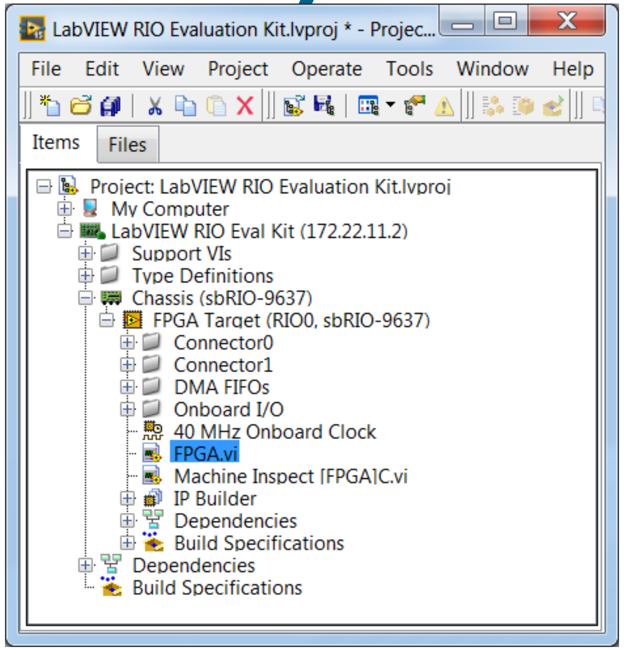




LabVIEW FPGA

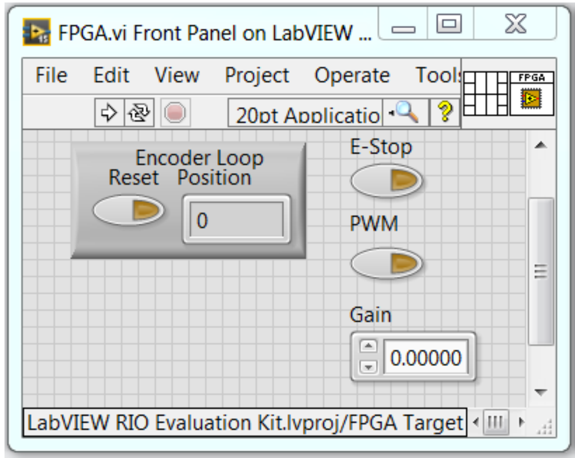
“Project” = System Configuration

“VI” = Application

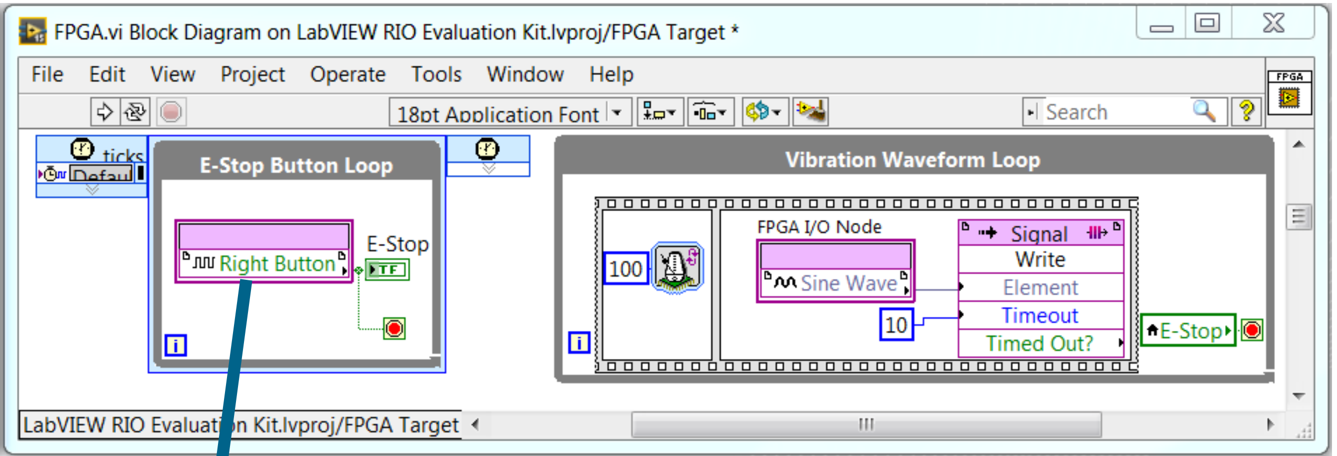


LabVIEW RIO Evaluation Kit.lvproj * - Projec...
File Edit View Project Operate Tools Window Help
Items Files

```
Project: LabVIEW RIO Evaluation Kit.lvproj
  My Computer
  LabVIEW RIO Eval Kit (172.22.11.2)
    Support VIs
    Type Definitions
    Chassis (sbRIO-9637)
      FPGA Target (RIO0, sbRIO-9637)
        Connector0
        Connector1
        DMA FIFOs
        Onboard I/O
        40 MHz Onboard Clock
        FPGA.vi
        Machine Inspect [FPGA]C.vi
        IP Builder
        Dependencies
        Build Specifications
    Dependencies
    Build Specifications
```



FPGA.vi Front Panel on LabVIEW ...
File Edit View Project Operate Tools
20pt Applicatio
Encoder Loop
Reset Position
0
E-Stop
PWM
Gain
0.00000
LabVIEW RIO Evaluation Kit.lvproj/FPGA Target



“Front Panel”= Interface Elements

I/O Node

“Block Diagram”= Code

LabVIEW FPGA vs. VHDL: Blink an LED

VHDL Implementation

```
begin

  LED <= LED_local;

  process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
       if ToggleLED then
         LED_local <= not LED_local;
       end if;
    end if;
  end process;

  CounterProc: process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
      if CounterValue = kCounterTC then
        CounterValue <= (others => '0');
        ToggleLED <= true;
      else
        CounterValue <= CounterValue + 1;
        ToggleLED <= false;
      end if;
    end if;
  end process CounterProc;

end rtl;
```

Physical wire connection to “LED”

LabVIEW FPGA vs. VHDL: Blink an LED

VHDL Implementation

```
begin

  LED <= LED_local;

  process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
       if ToggleLED then
         LED_local <= not LED_local;
       end if;
    end if;
  end process;

  CounterProc: process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
      if CounterValue = kCounterTC then
        CounterValue <= (others => '0');
        ToggleLED <= true;
      else
        CounterValue <= CounterValue + 1;
        ToggleLED <= false;
      end if;
    end if;
  end process CounterProc;

end rtl;
```

Physical wire connection to “LED”

Toggle the physical LED when internal timing signal “ToggleLED” is true. Executes every tick of the 50Mhz clock.

LabVIEW FPGA vs. VHDL: Blink an LED

VHDL Implementation

```
begin
  LED <= LED_local;
```

Physical wire connection to “LED”

```
  process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
       if ToggleLED then
         LED_local <= not LED_local;
       end if;
    end if;
  end process;
```

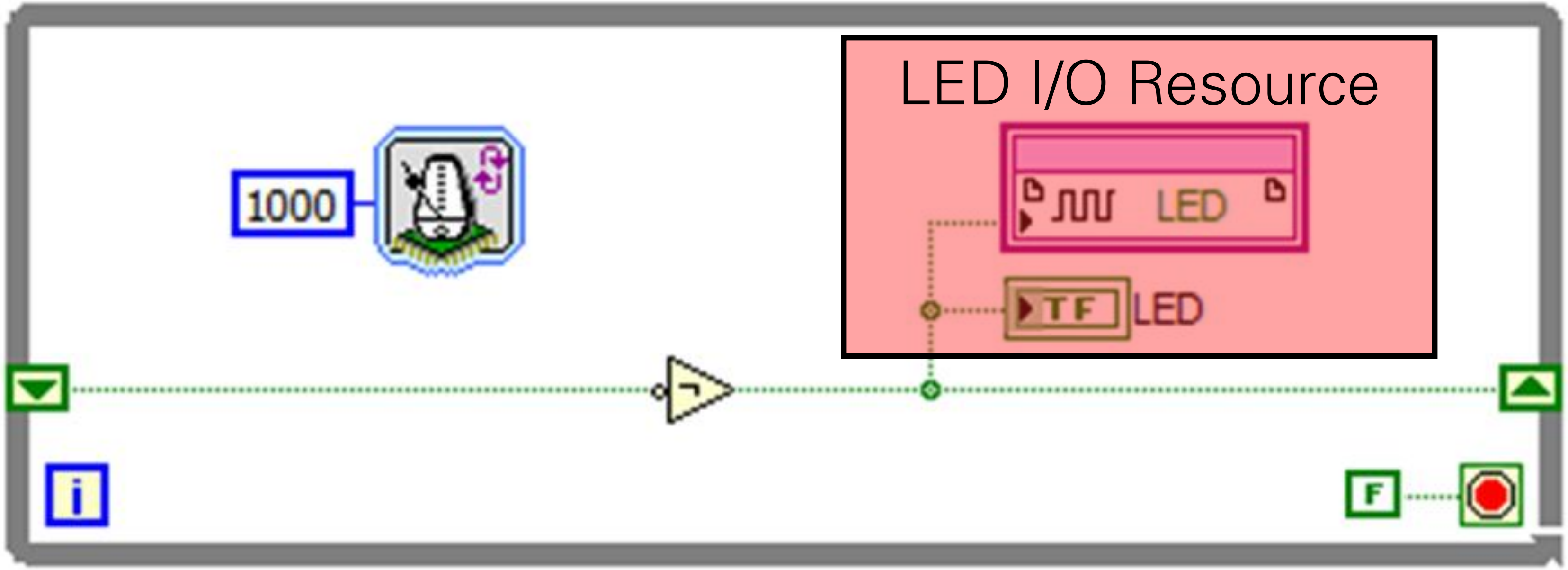
Toggle the physical LED when internal timing signal “ToggleLED” is true. Executes every tick of the 50Mhz clock.

```
  CounterProc: process (CLK_50MHZ)
  begin
    if rising_edge(CLK_50MHZ) then
      if CounterValue = kCounterTC then
        CounterValue <= (others => '0');
        ToggleLED <= true;
      else
        CounterValue <= CounterValue + 1;
        ToggleLED <= false;
      end if;
    end if;
  end process CounterProc;

end rtl;
```

Counter establishes the timing of the “ToggleLED” signal. Goes “true” when the counter reaches 50,000,000 (1 second) and resets counter.

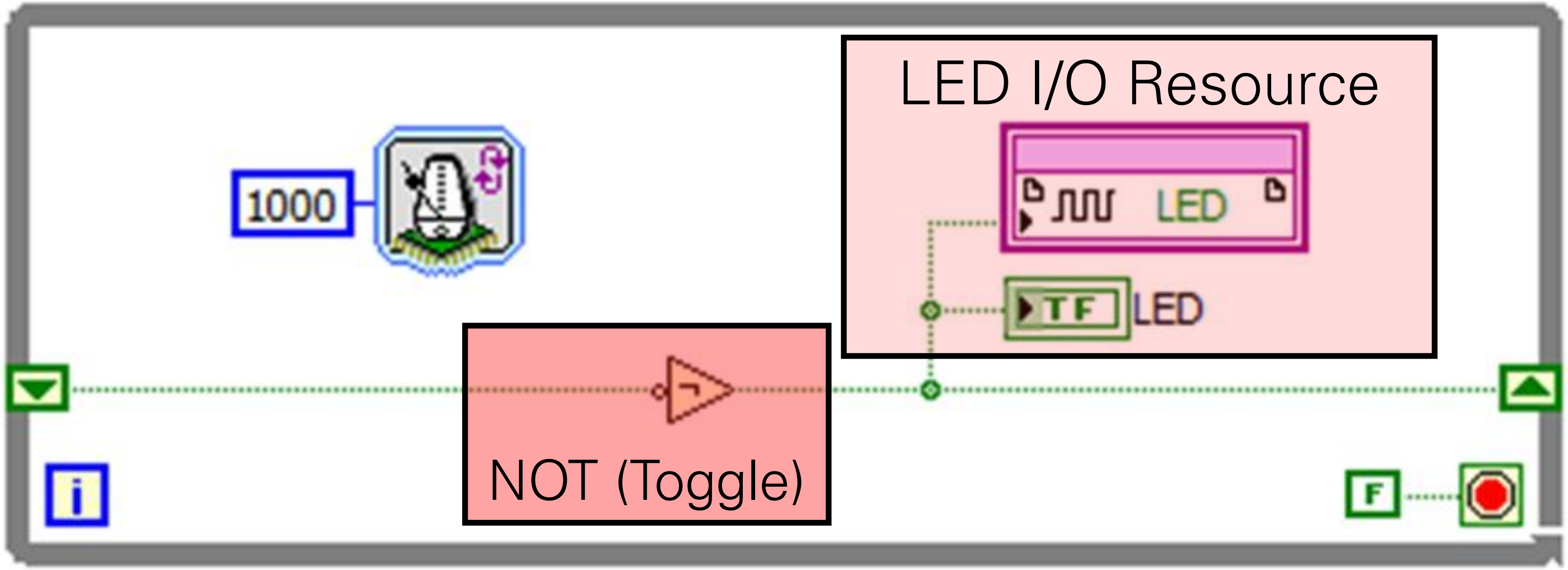
LabVIEW FPGA vs. VHDL: Blink an LED

LabVIEW Implementation



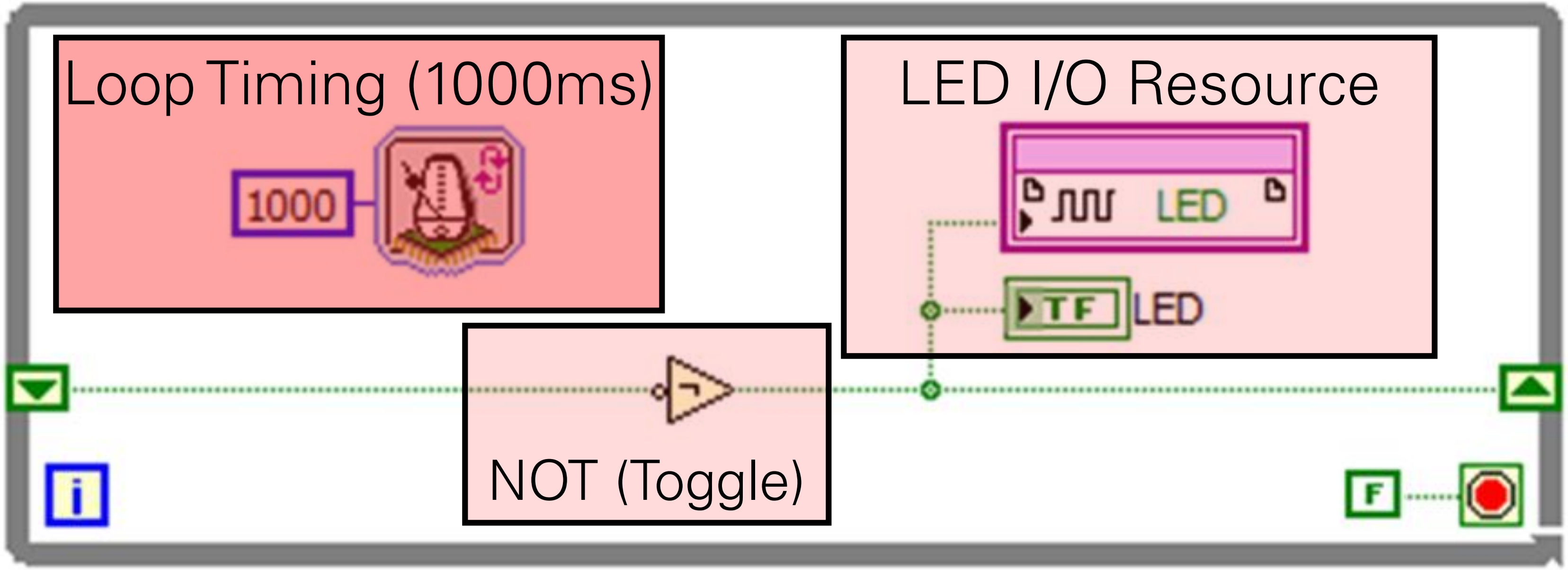
LabVIEW FPGA vs. VHDL: Blink an LED

LabVIEW Implementation



LabVIEW FPGA vs. VHDL: Blink an LED

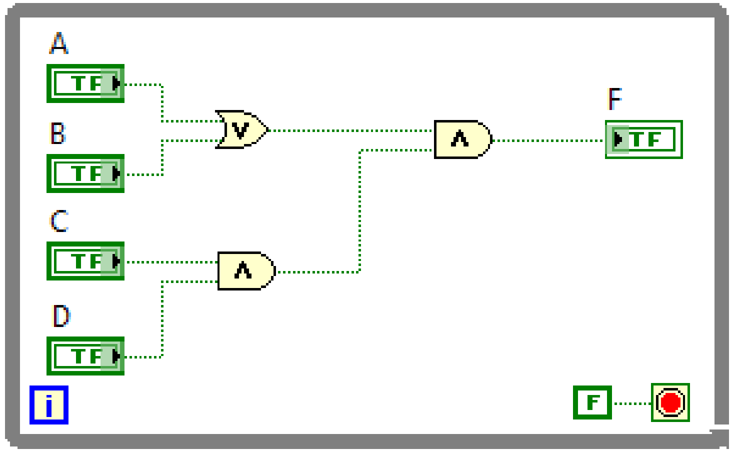
LabVIEW Implementation

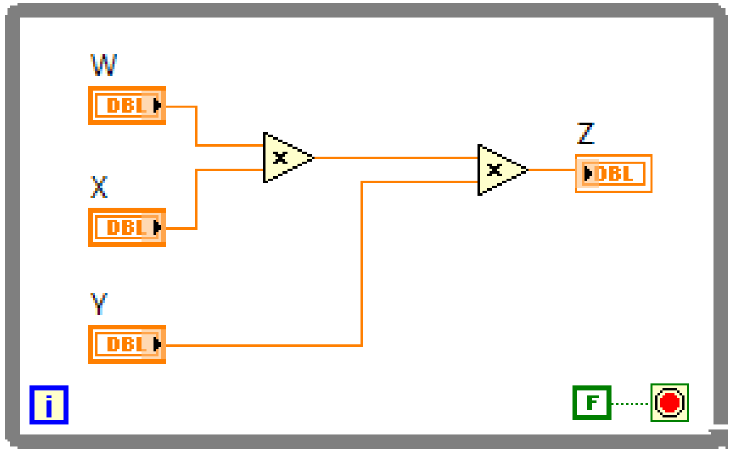


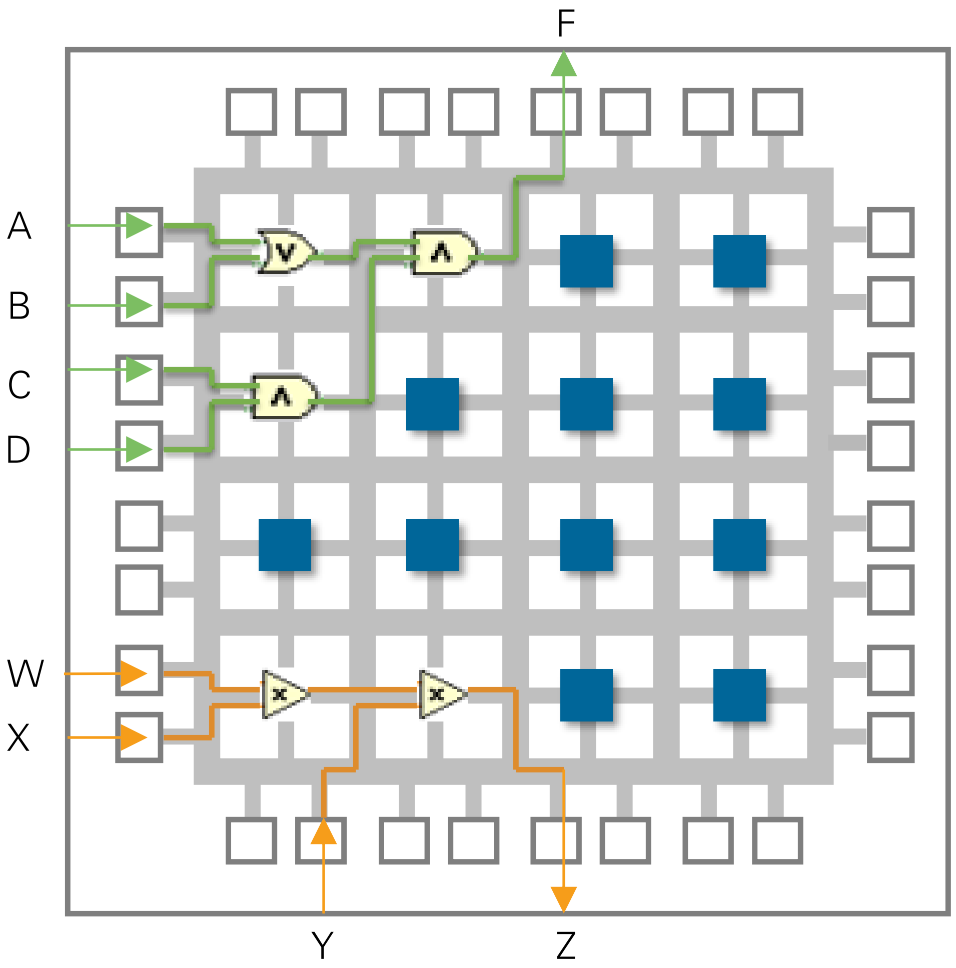
Why Are FPGAs Useful?

- *True Parallelism* – Provides parallel tasks and pipelining
- *High Reliability* – Designs become a custom circuit
- *High Determinism* – Runs algorithms at deterministic rates down to 25 ns (faster in many cases)
- *Reconfigurable* – Create new and alter existing task-specific personalities

Parallel Processing

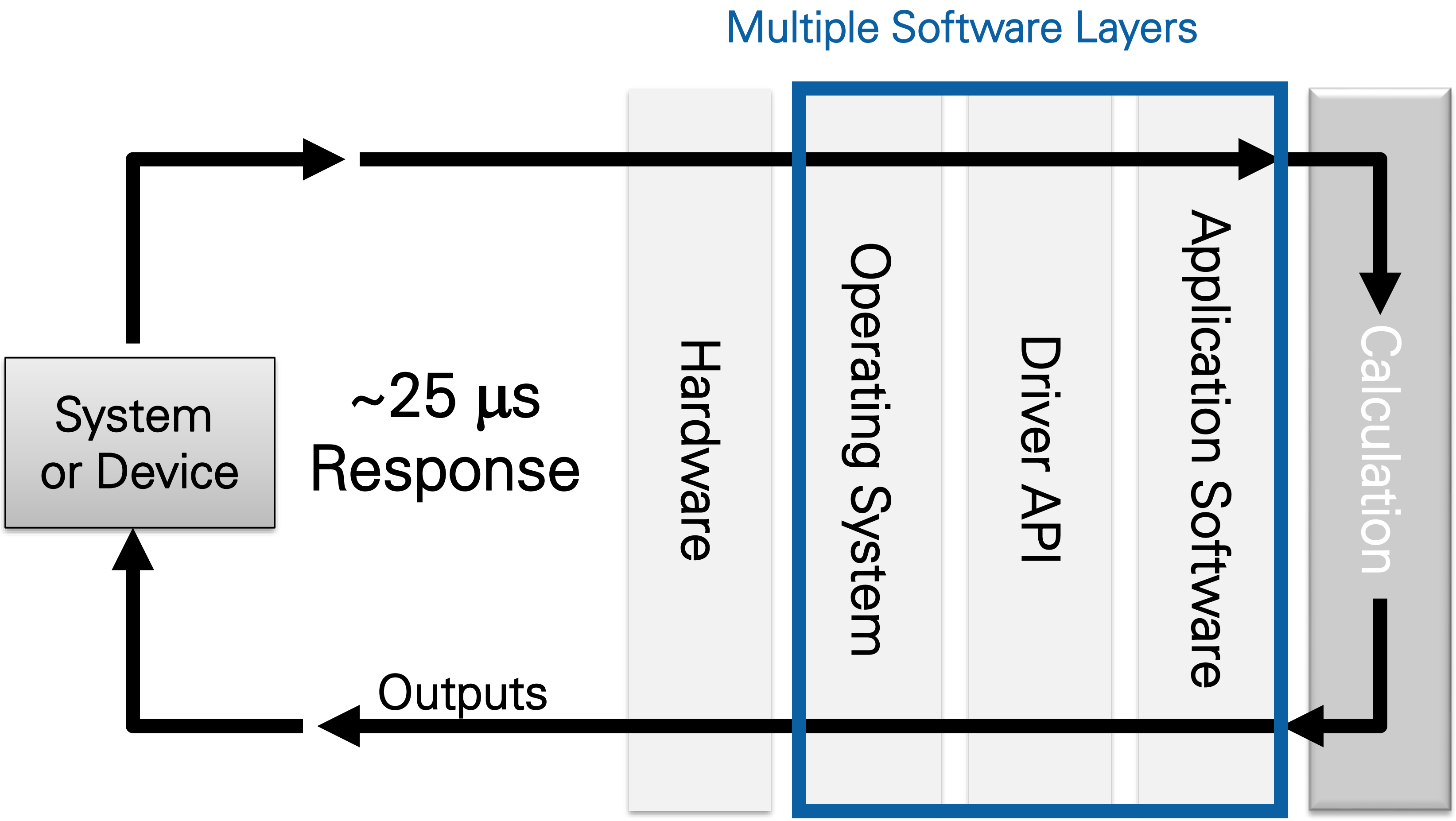






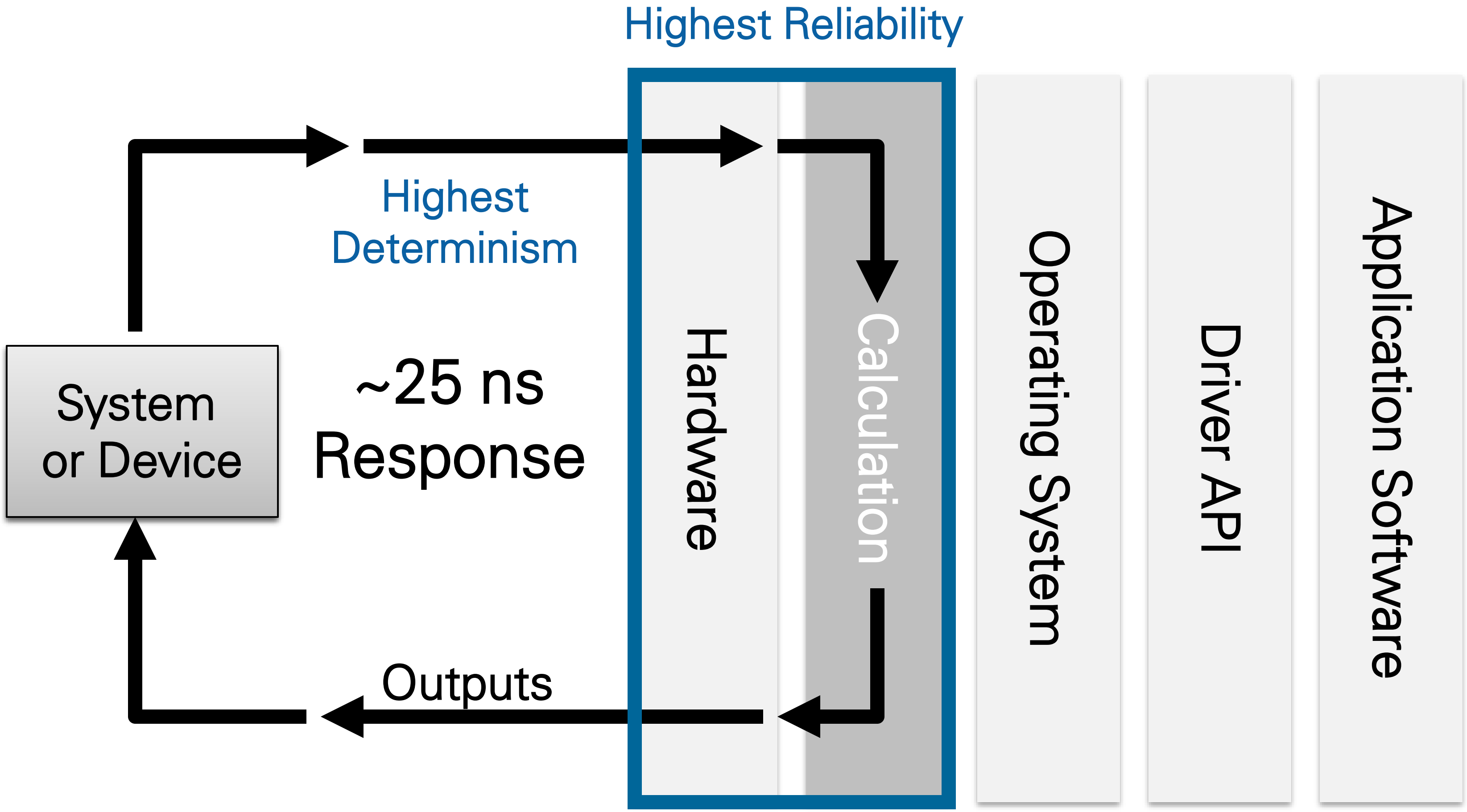
High Reliability and Determinism

Decision Making in Software



High Reliability and Determinism

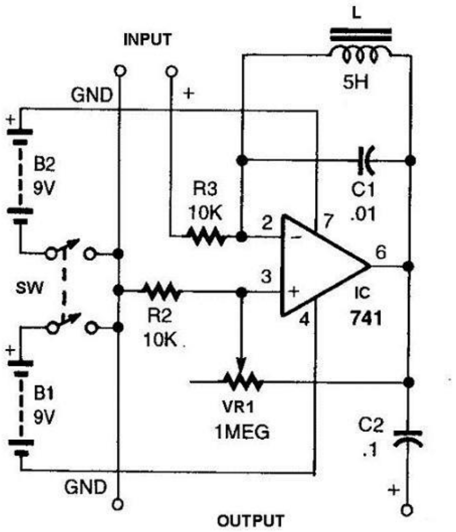
Decision Making in Hardware

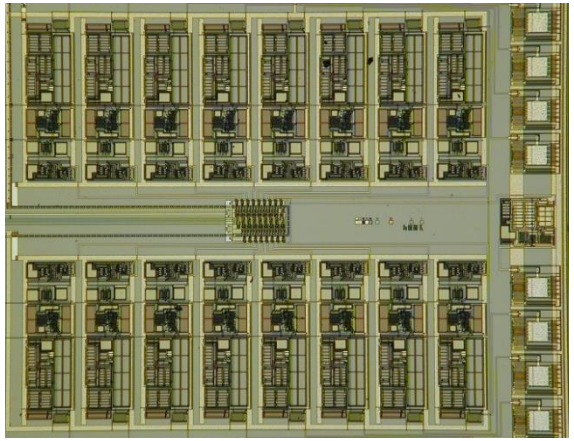


Reconfigurable

- Enables rapid development iterations
- Reduces overall design cost, taking NRE into account
- Decreases long-term maintenance

vs.





FPGAs

Custom Circuits

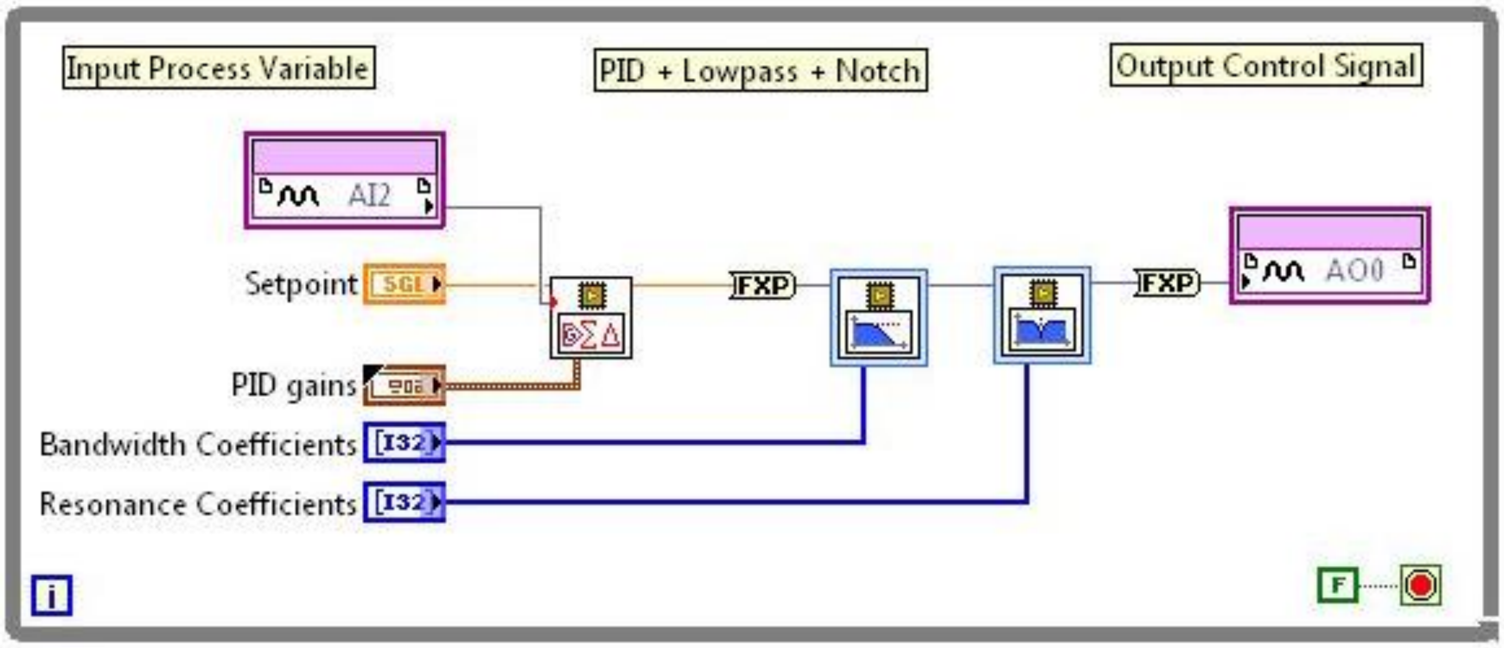
ASICs

Common application target

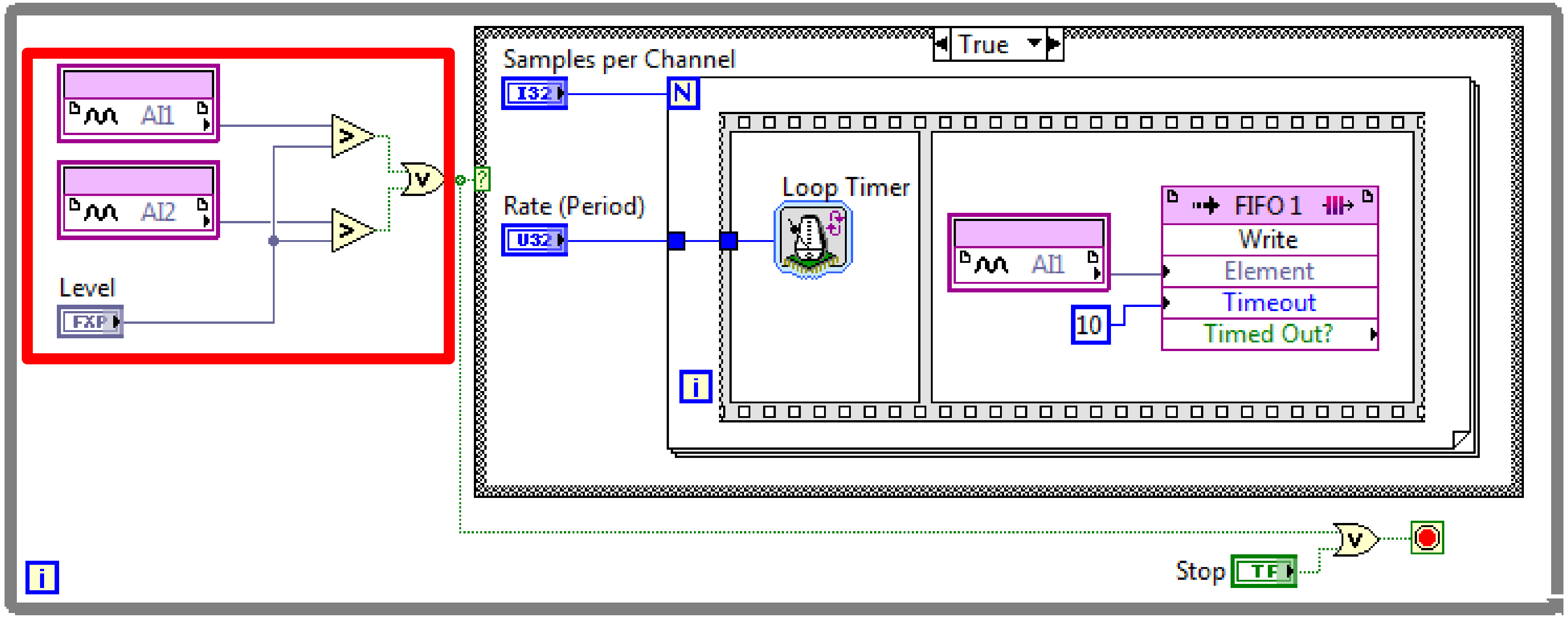
Common Applications

- High-speed control
- Custom data acquisition
- Digital communication protocols
- Inline signal processing

High-Speed Control



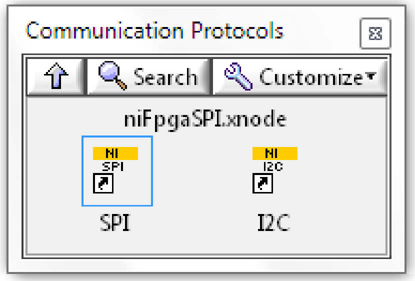
Custom Triggered Analog Input

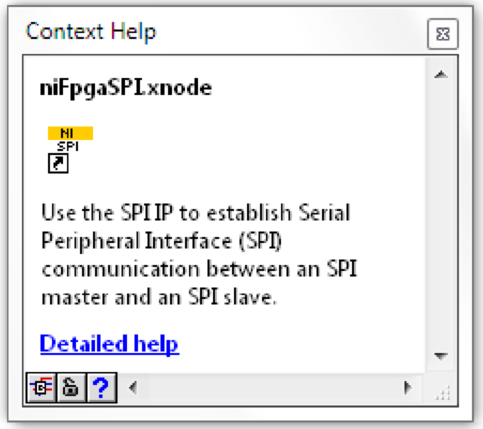


- Custom timing & synchronization
- Multi-rate sampling
- Custom counters
- Flexible PWM
- Flexible encoder interface

Digital Communication Protocol APIs

- Communications Protocols Palette: SPI/I2C




Context Help

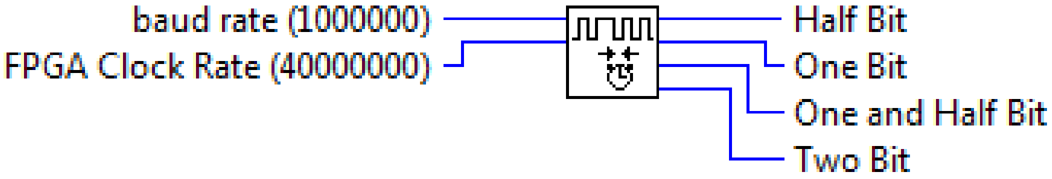
niFpgaSPI.xnode

Use the SPI IP to establish Serial Peripheral Interface (SPI) communication between an SPI master and an SPI slave.

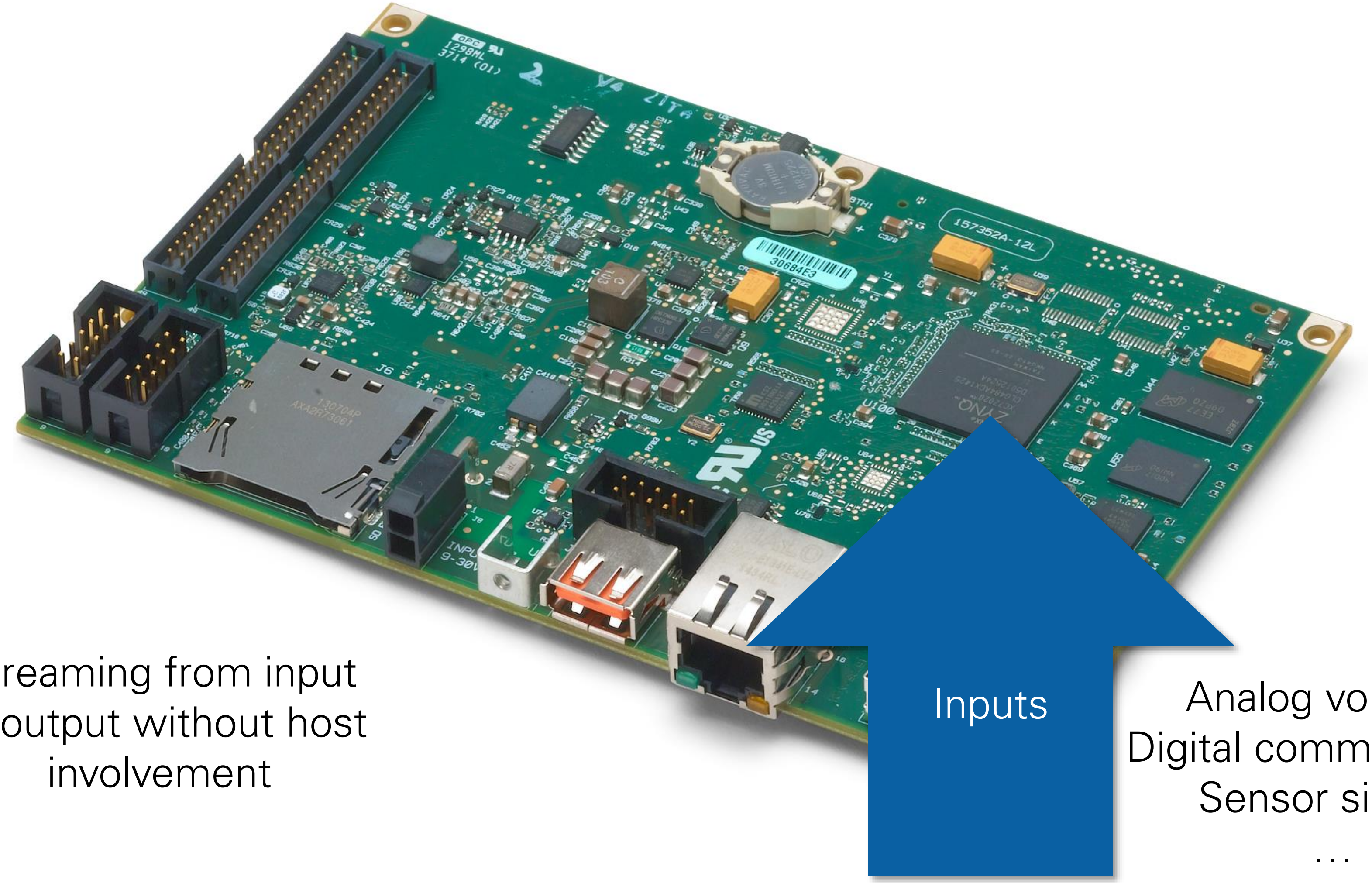
Detailed help

- Serial:

LV FPGA Serial Calc Timing Values.vi



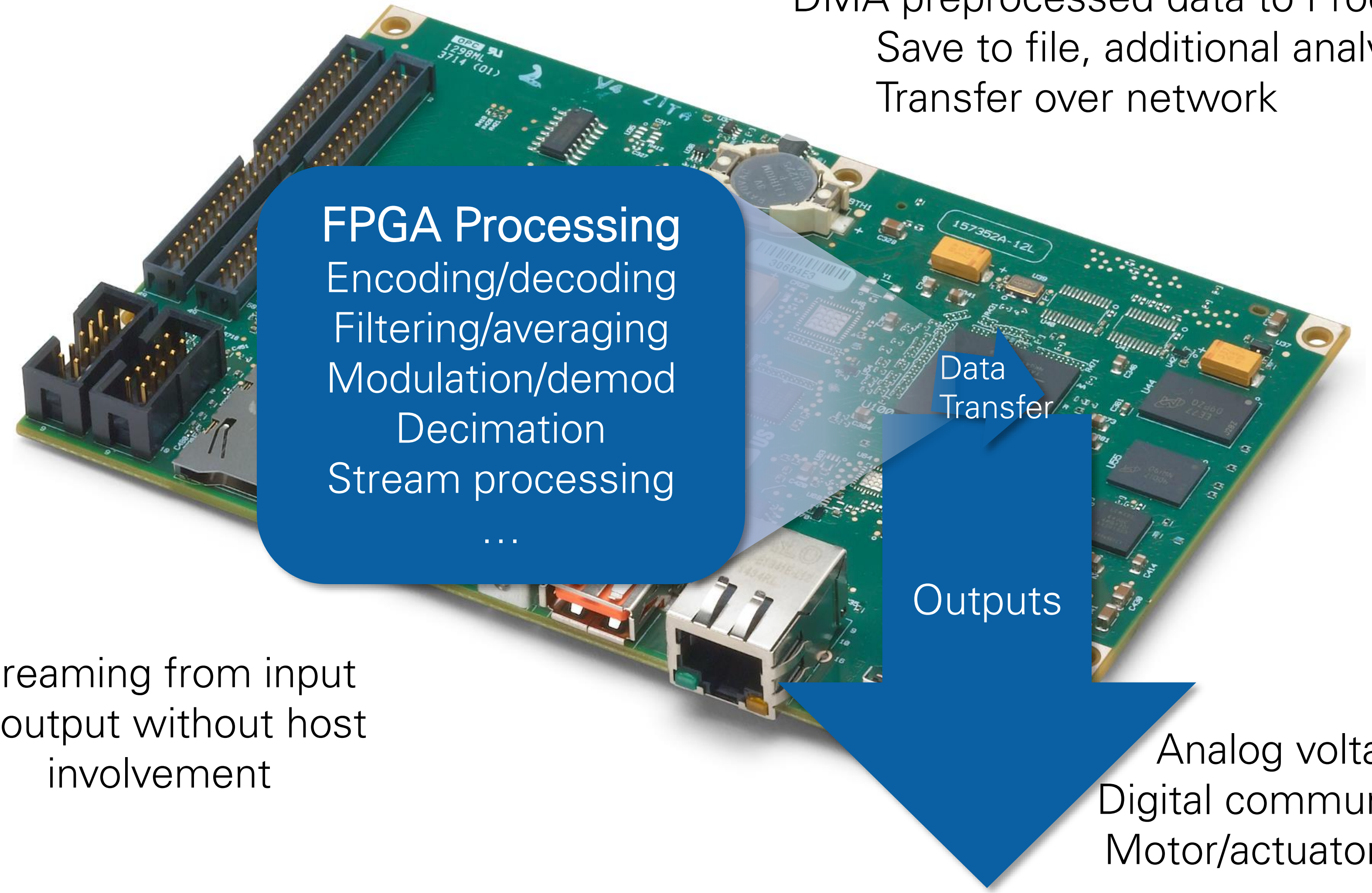
Inline Signal Processing and Data Reduction



Streaming from input
to output without host
involvement

Analog voltages
Digital communication
Sensor signals
. . .

Inline Signal Processing and Data Reduction



DMA preprocessed data to Processor:
Save to file, additional analysis,
Transfer over network

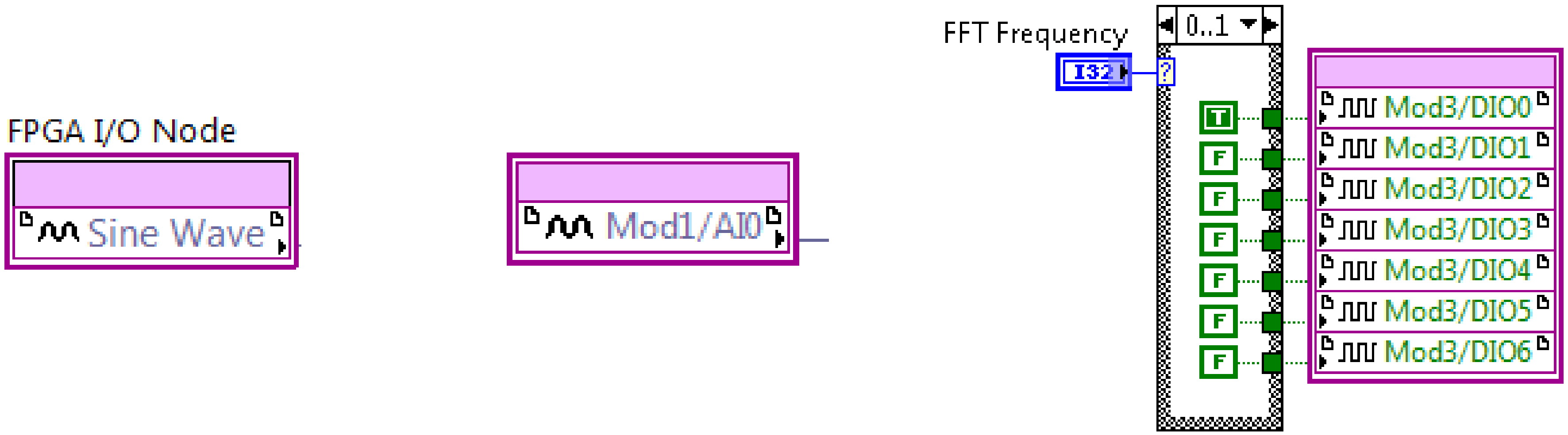
FPGA Processing
Encoding/decoding
Filtering/averaging
Modulation/demod
Decimation
Stream processing
. . .

Streaming from input
to output without host
involvement

Analog voltages
Digital communication
Motor/actuator drives
. . .

Data Transfer : I/O ←→ FPGA

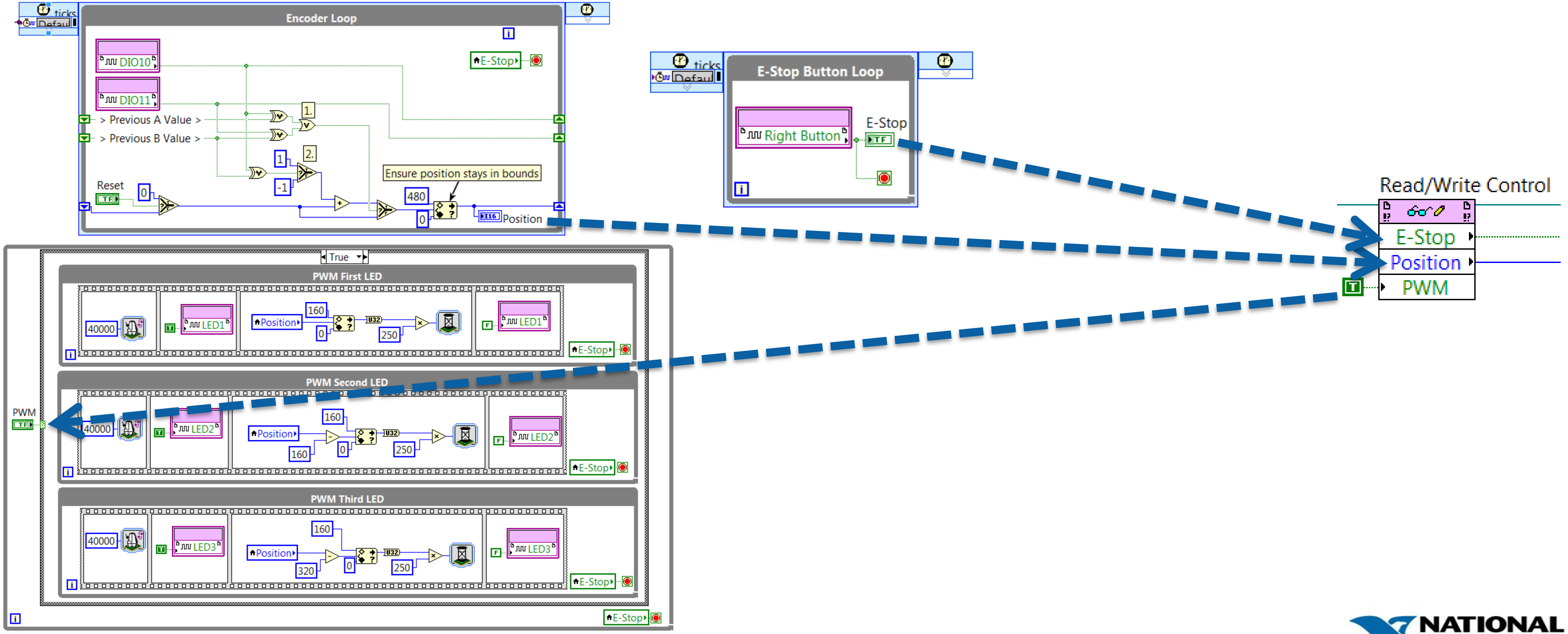
- FPGA I/O Nodes acquire and generate data
- Directly connected to I/O pins
- Data rates are defined by the AIO/DIO modules
- FPGA acquires one data point per loop iteration
- Can rename channels to be application-specific



FPGA ←→ RT: FPGA Read/Write Controls

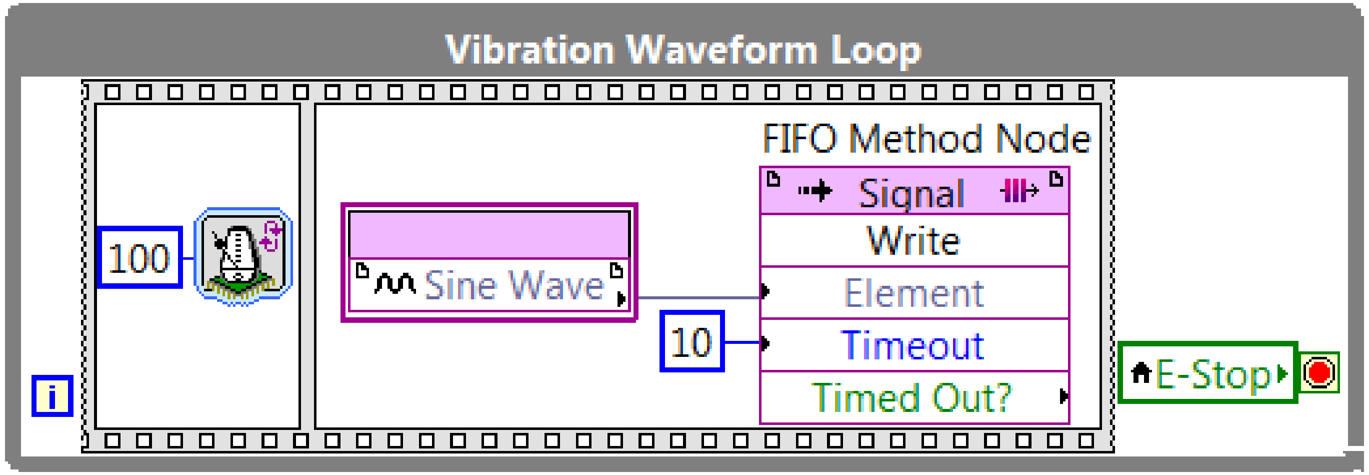
FPGA VI

Real-Time VI

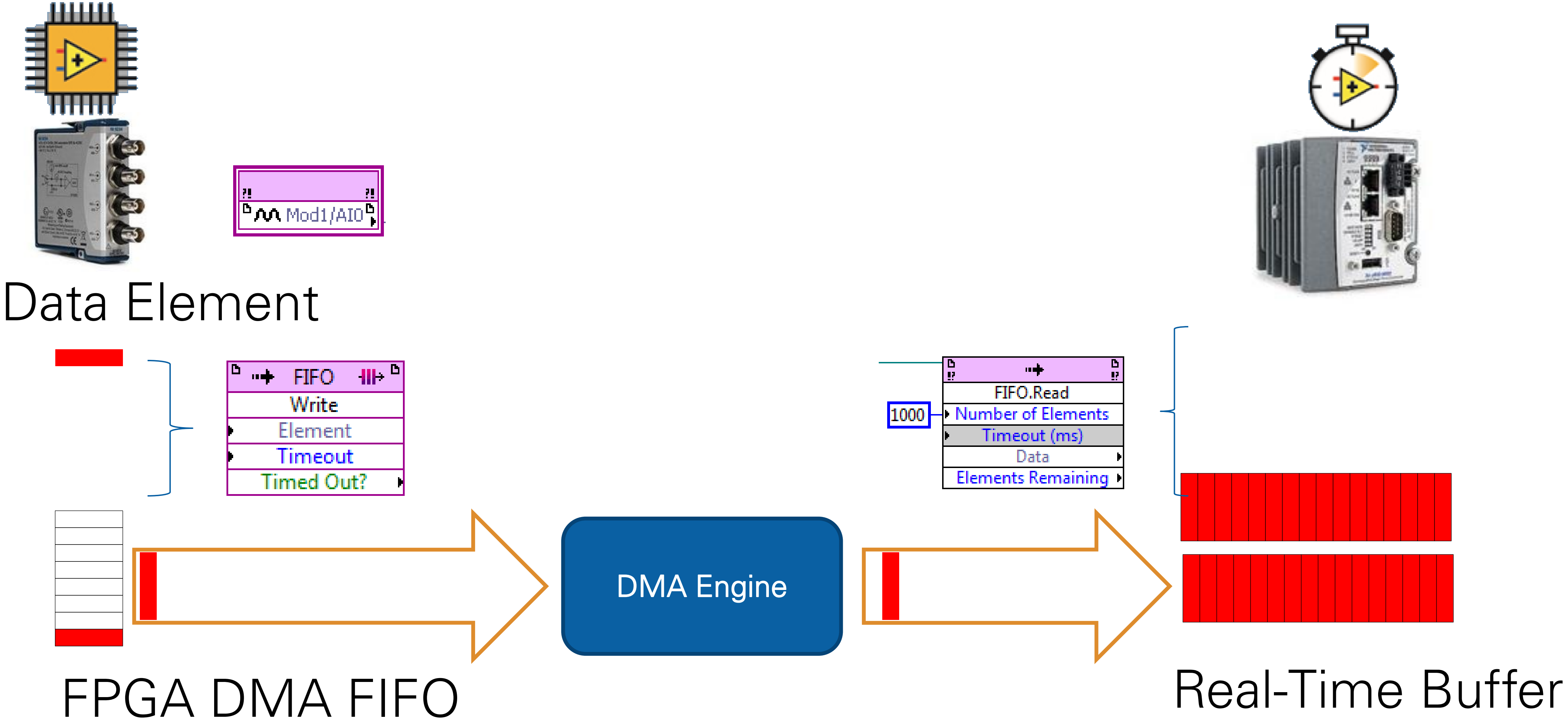


FPGA ←→ RT: Direct Memory Access (DMA) FIFOs

- DMA FIFOs are an efficient mechanism for streaming data to/from the FPGA to/from a Real-Time or Windows Processor
- RIO hardware targets have between 3 to 16 dedicated DMA channels, depending on the FPGA
- Target-Scoped FIFOs can transfer data between different portions of an FPGA VI or between VIs on an FPGA Target



FPGA ←→ RT: DMA FIFOs



Mod1/AIO0

Data Element

FIFO

Write

Element

Timeout

Timed Out?

1000

FIFO.Read

Number of Elements

Timeout (ms)

Data

Elements Remaining

DMA Engine

FPGA DMA FIFO

Real-Time Buffer

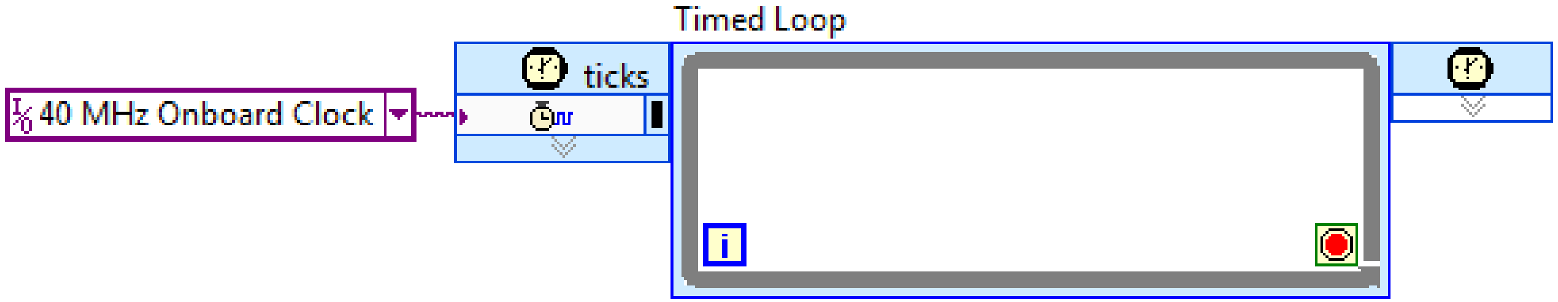
Clocking

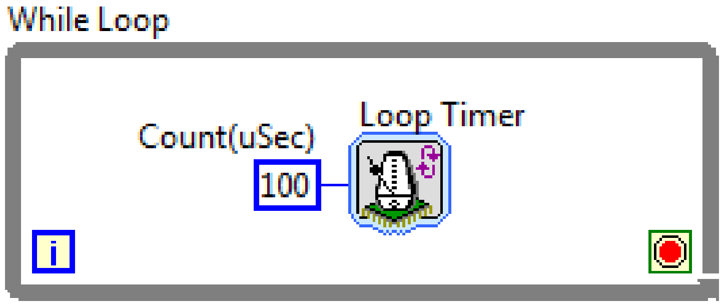
Process event, and registers

Understanding Clocks and Hardware Concurrency

- A **Timed Loop** on FPGA runs at 40MHz by default, based on the Onboard Clock

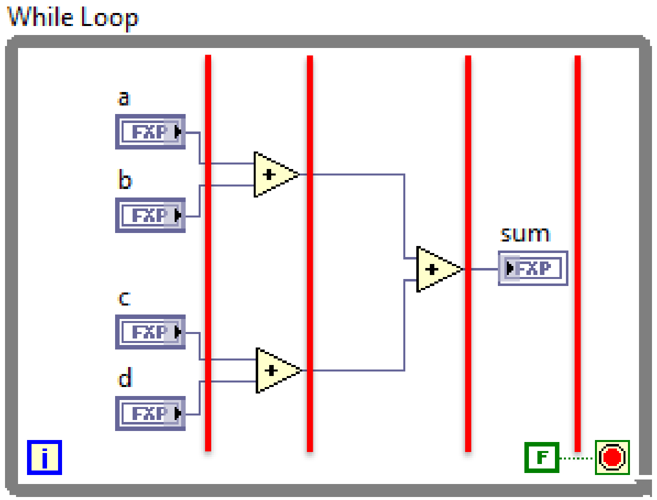
- A **While Loop** will execute at the rate specified in the **Loop Timer** function, either in ticks, ms, or µs.

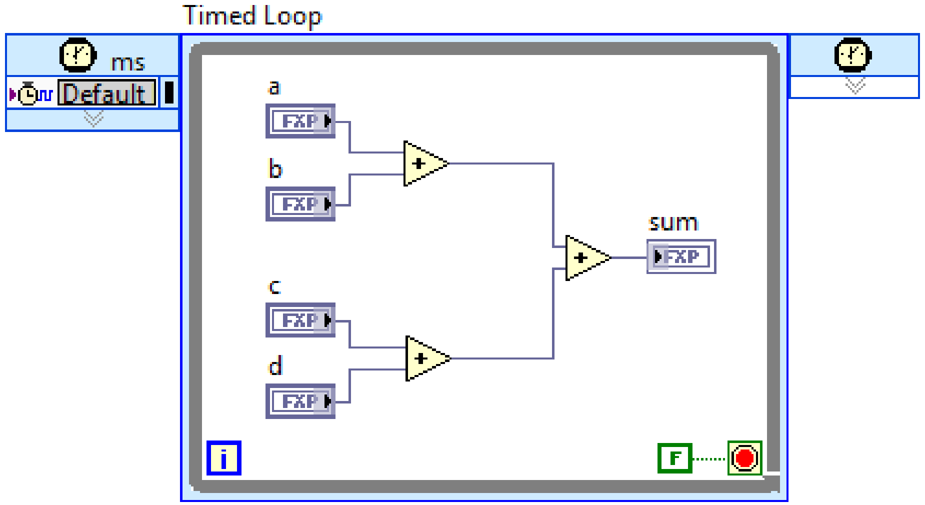




Understanding Clocks and Hardware Concurrency

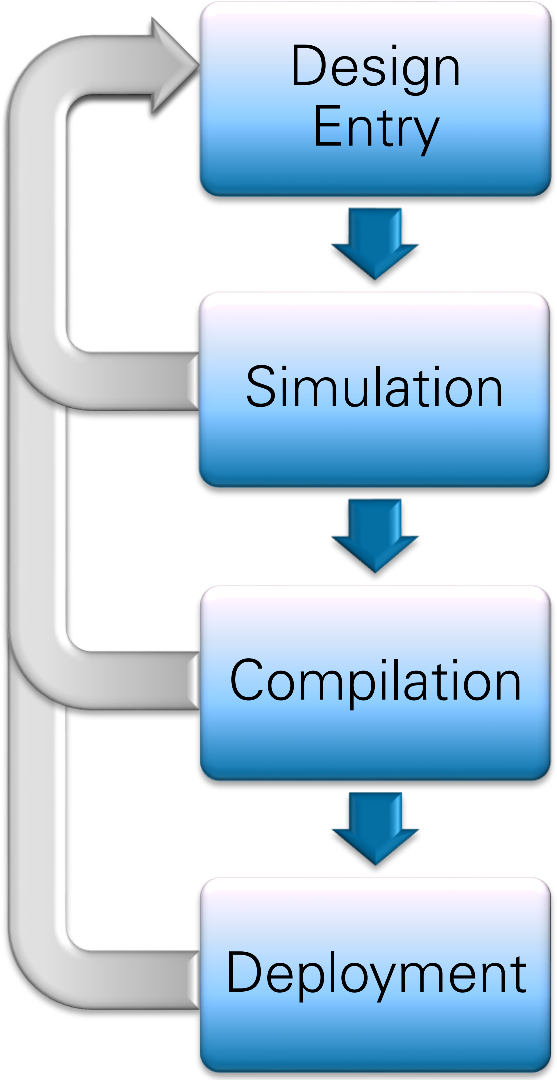
- The enable chain includes registers between each node that store values and execute at the rising edge of the clock
- A Timed Loop on FPGA is called a **Single Cycle Timed Loop (SCTL)**
    - Code executes in 1 clock cycle
    - Removes registers
    - Uses less resources
    - Not all functions are supported



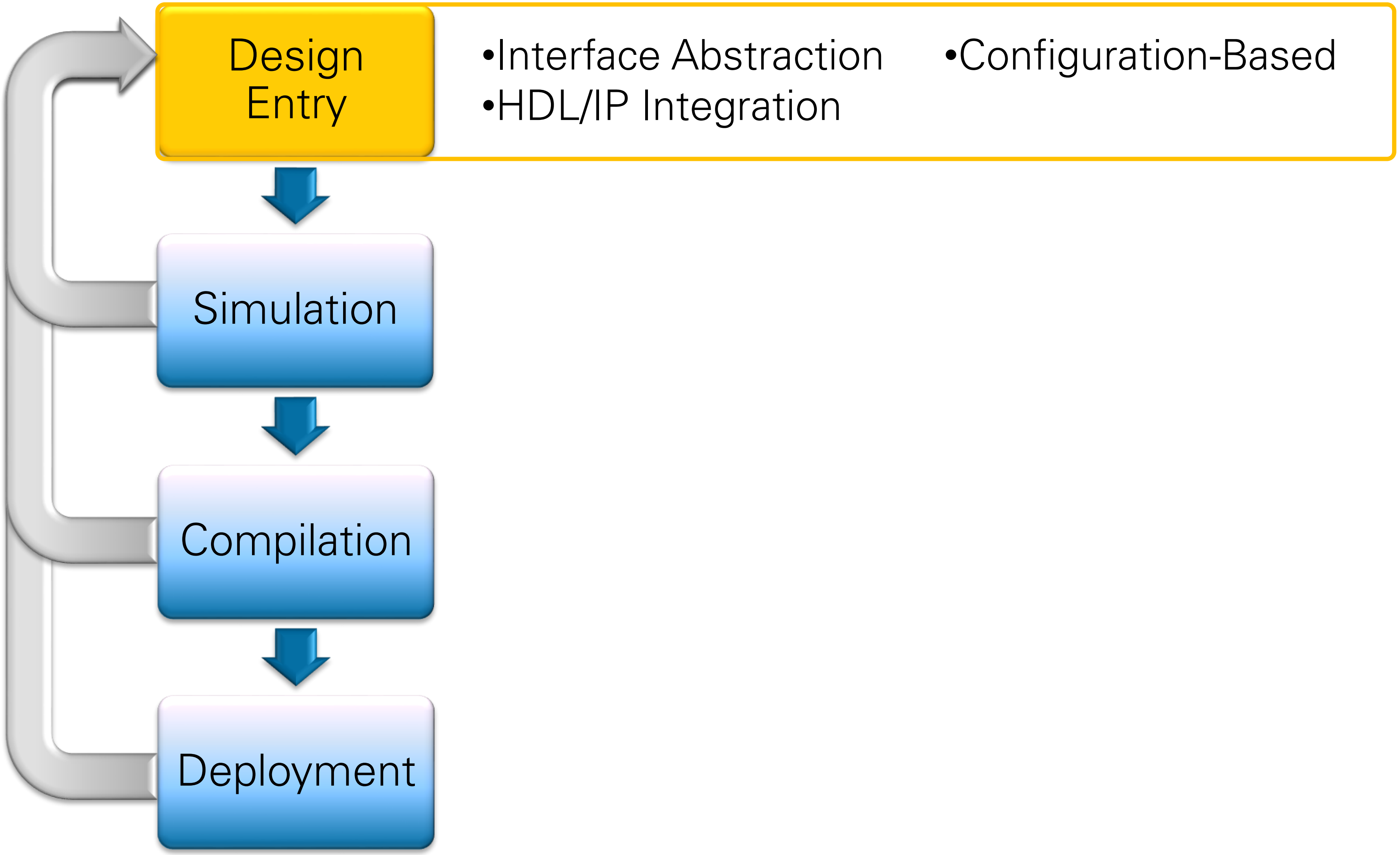


LabVIEW design flow

Simplified FPGA Design Flow



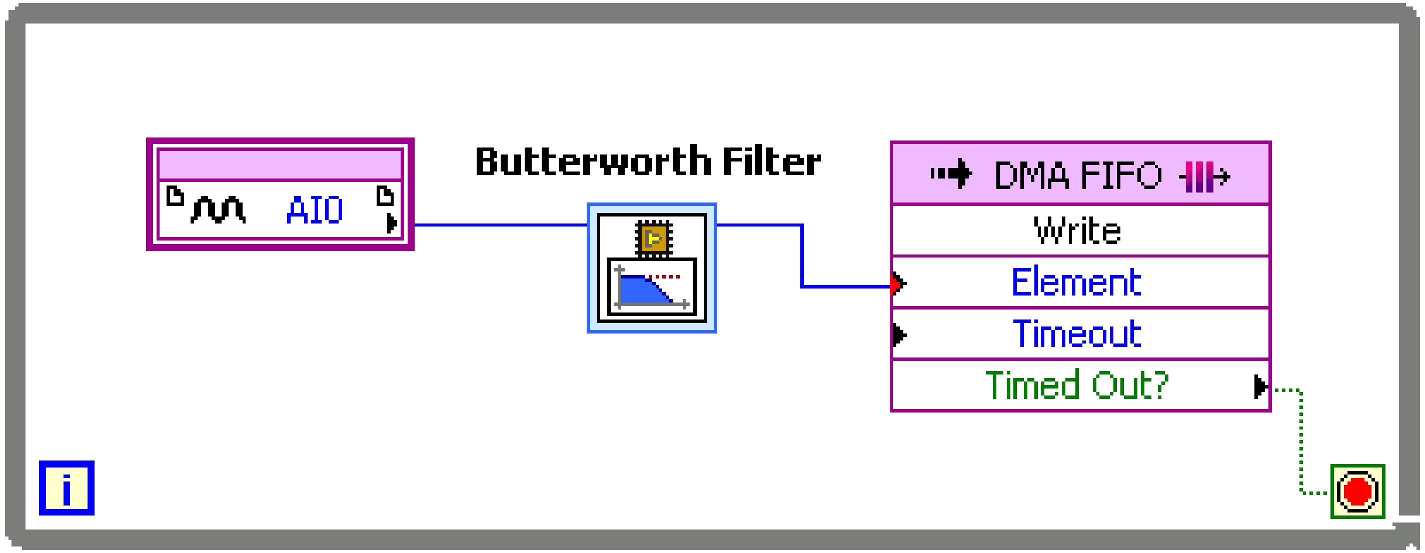
Simplified FPGA Design Flow



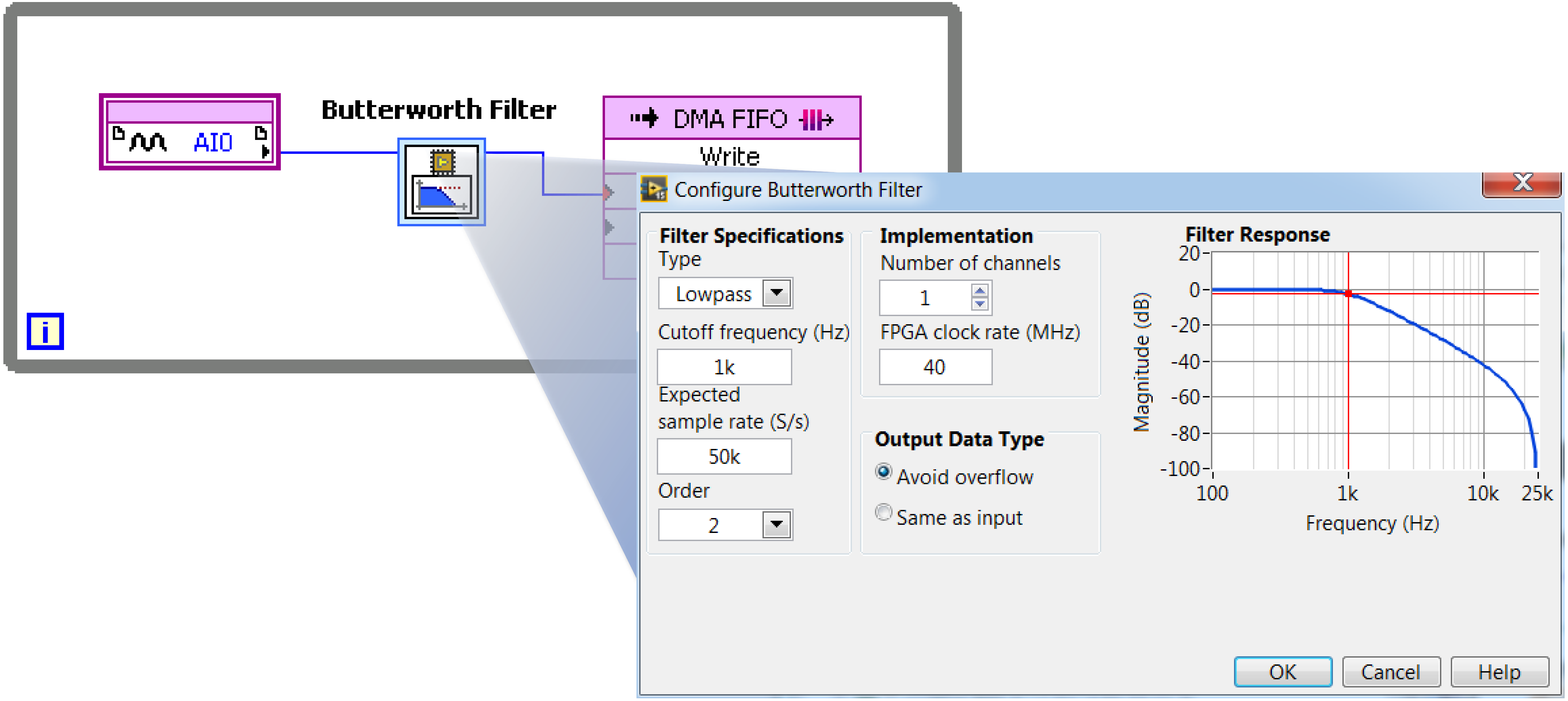
- •Interface Abstraction
- •HDL/IP Integration
- •Configuration-Based

Interface Abstraction

- I/O Interfaces to NI and 3rd party I/O modules, custom daughtercards
- Built-in DMA FIFO and memory interfaces



Configuration-Based Design



Configure Butterworth Filter

Filter Specifications

Type: Lowpass

Cutoff frequency (Hz): 1k

Expected sample rate (S/s): 50k

Order: 2

Implementation

Number of channels: 1

FPGA clock rate (MHz): 40

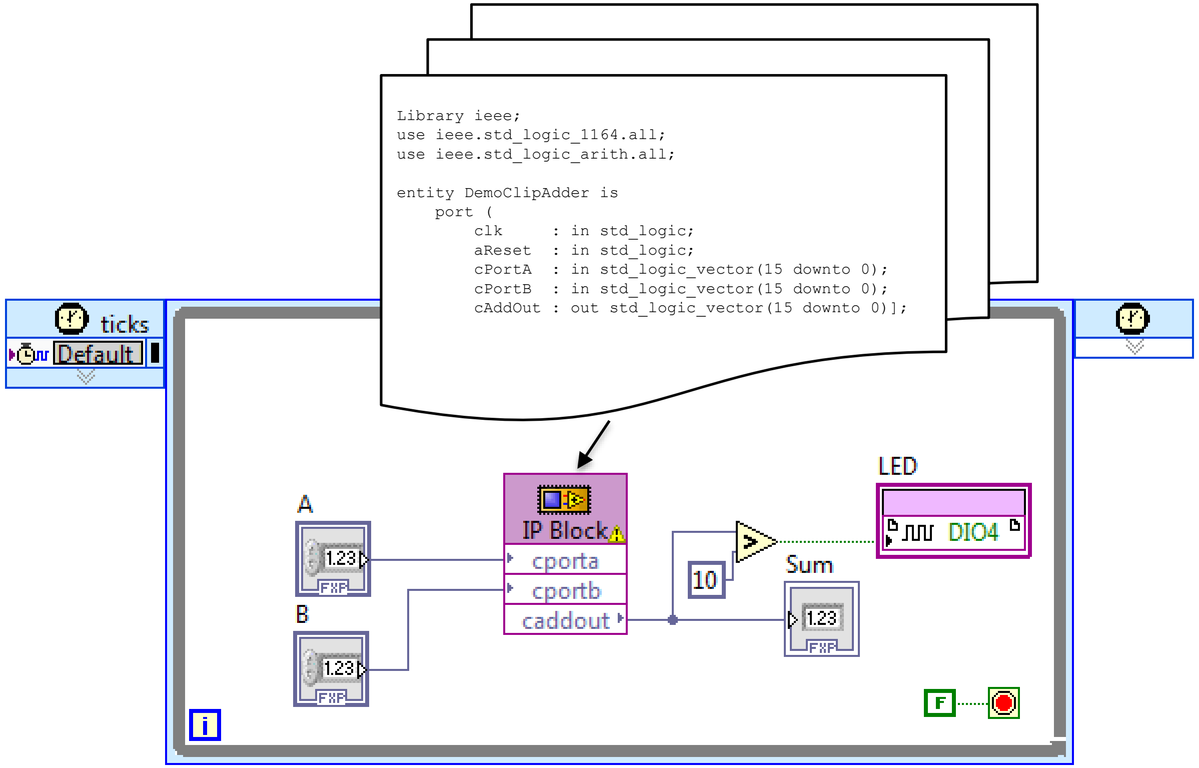
Output Data Type

● Avoid overflow

○ Same as input

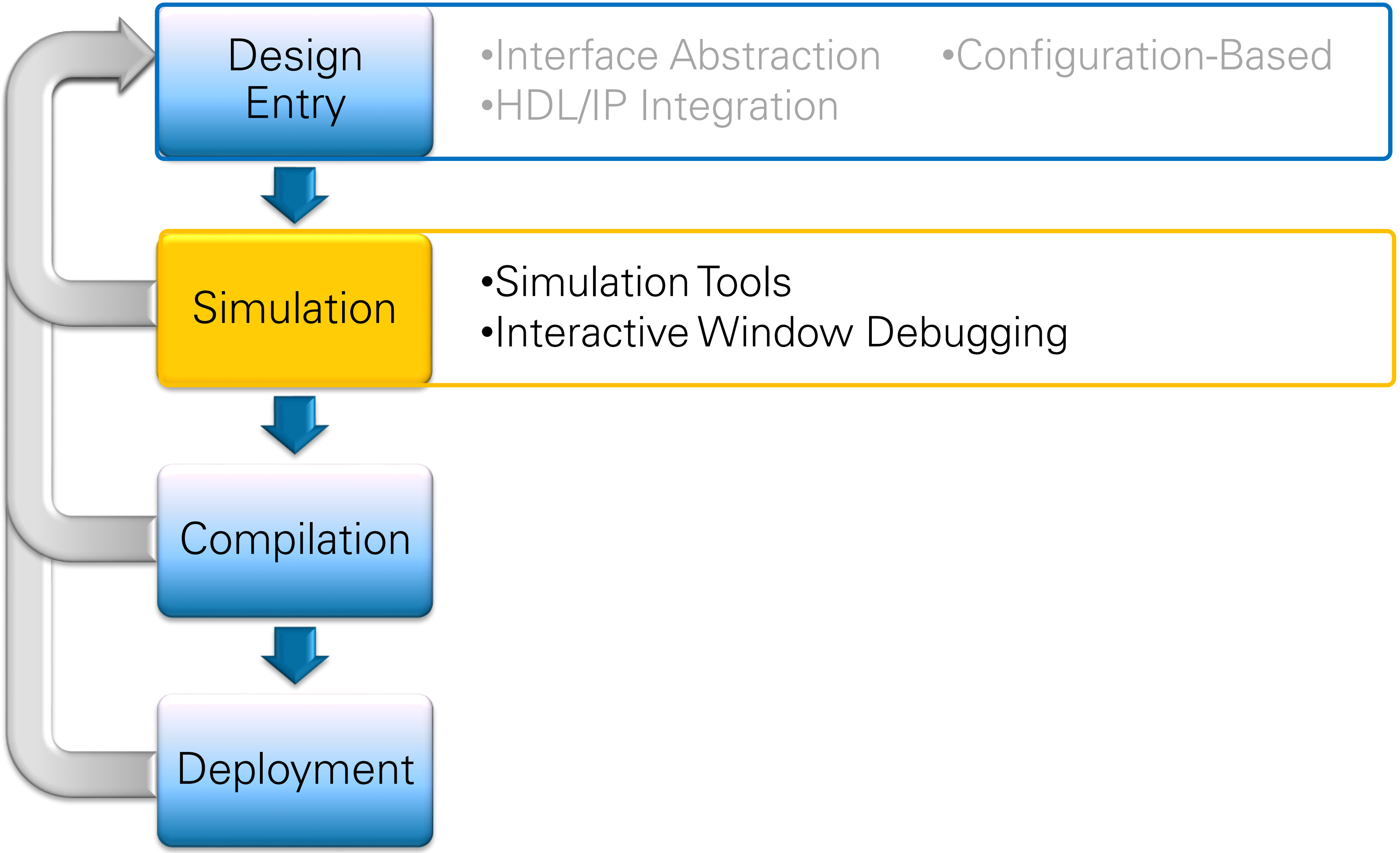
Reuse of Existing HDL Algorithms

- Use LabVIEW as the glue of your application
- Leverage existing digital design team expertise
- Similar to calling a shared library in LabVIEW on Windows or Real-Time



```
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)];
```

Simplified FPGA Design Flow



- Interface Abstraction
- HDL/IP Integration
- Configuration-Based

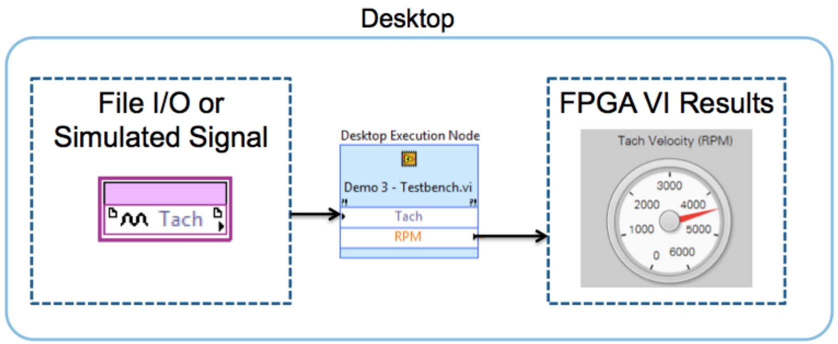
- Simulation Tools
- Interactive Window Debugging

Be More Productive with LabVIEW FPGA

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



Desktop

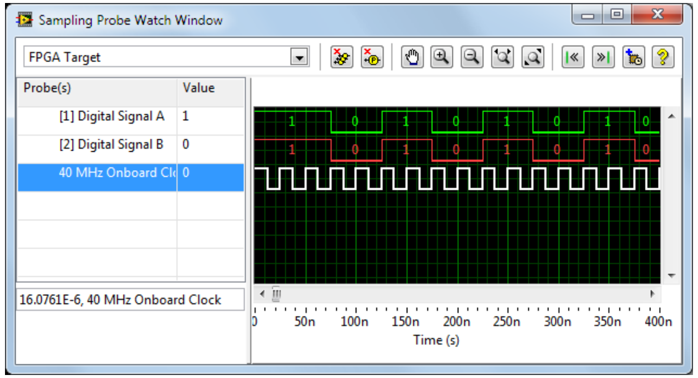
File I/O or Simulated Signal

Desktop Execution Node

FPGA VI Results

Verify Signal Timing with Waveform Probe

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



Sampling Probe Watch Window

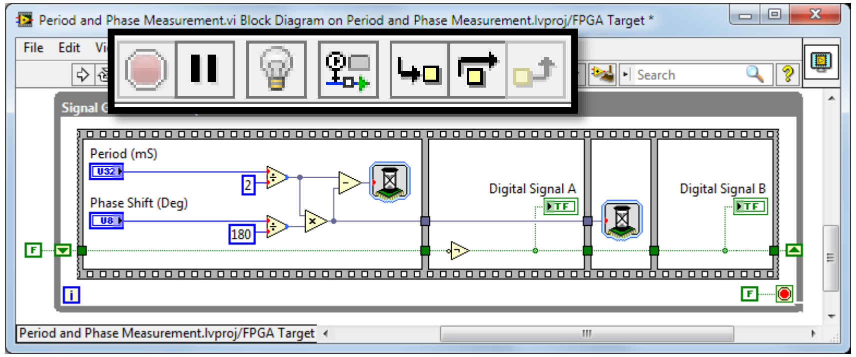
FPGA Target

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

16.0761E-6, 40 MHz Onboard Clock

Debug with Standard LabVIEW Features in Simulation

Highlight execution, breakpoints, and stepping features

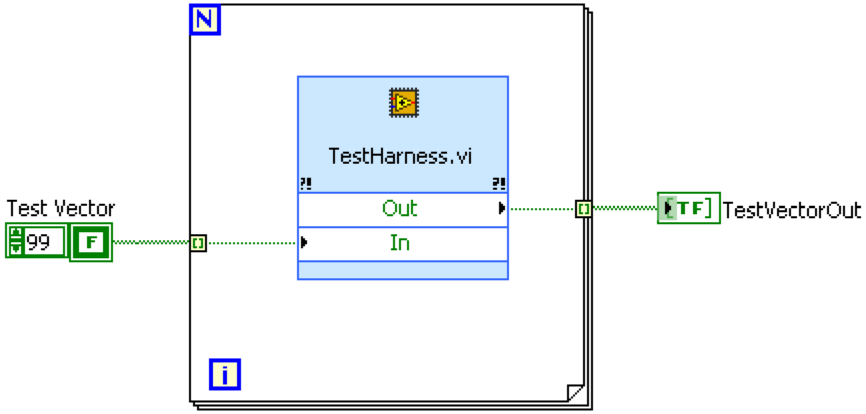


Period and Phase Measurement.vi Block Diagram on Period and Phase Measurement.lvproj/FPGA Target *

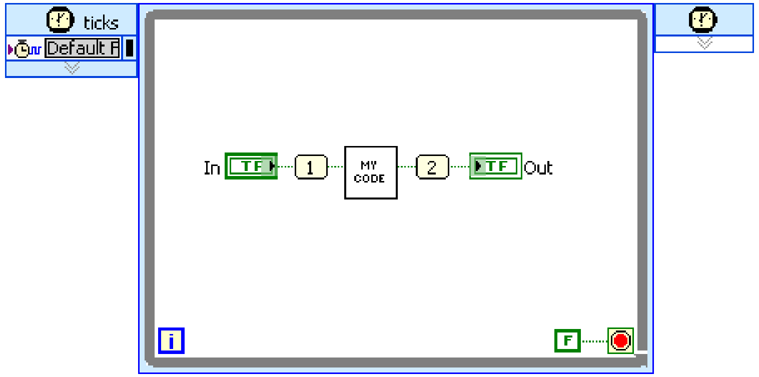
Period and Phase Measurement.lvproj/FPGA Target

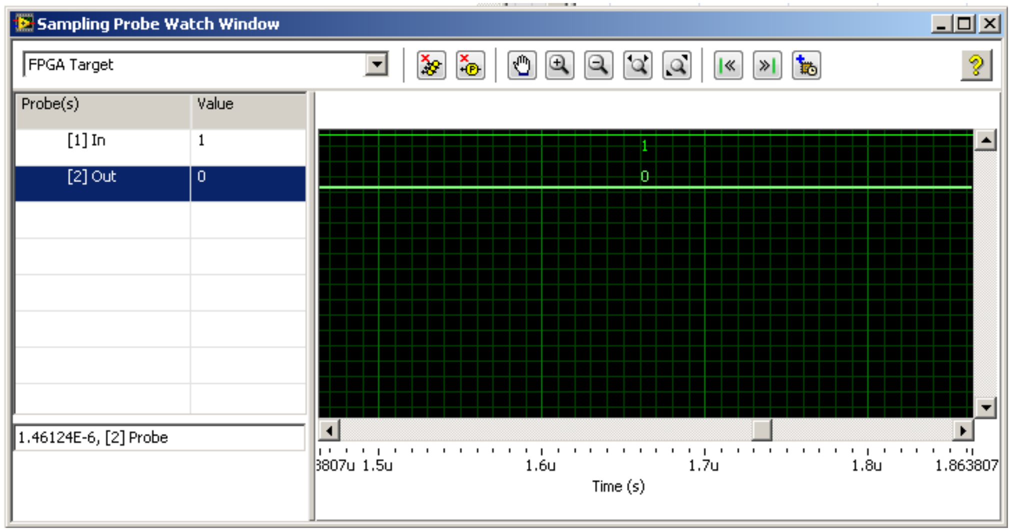
LabVIEW FPGA Desktop Execution Node

Unit Test



Test Harness

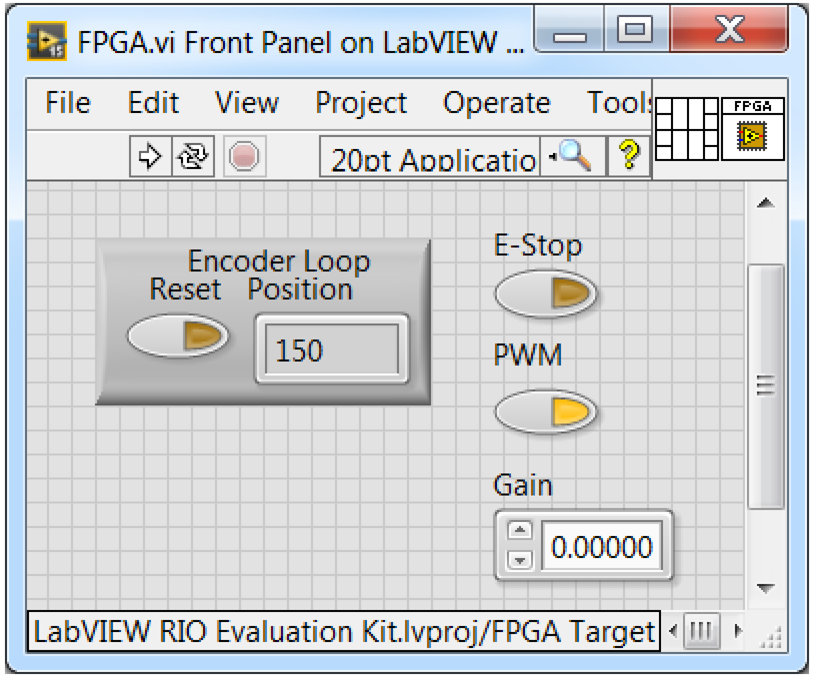




Probe(s)	Value
[1] In	1
[2] Out	0

1.46124E-6, [2] Probe

Interactive Front Panel "User Interface"



Encoder Loop

Reset: ● Position: 150

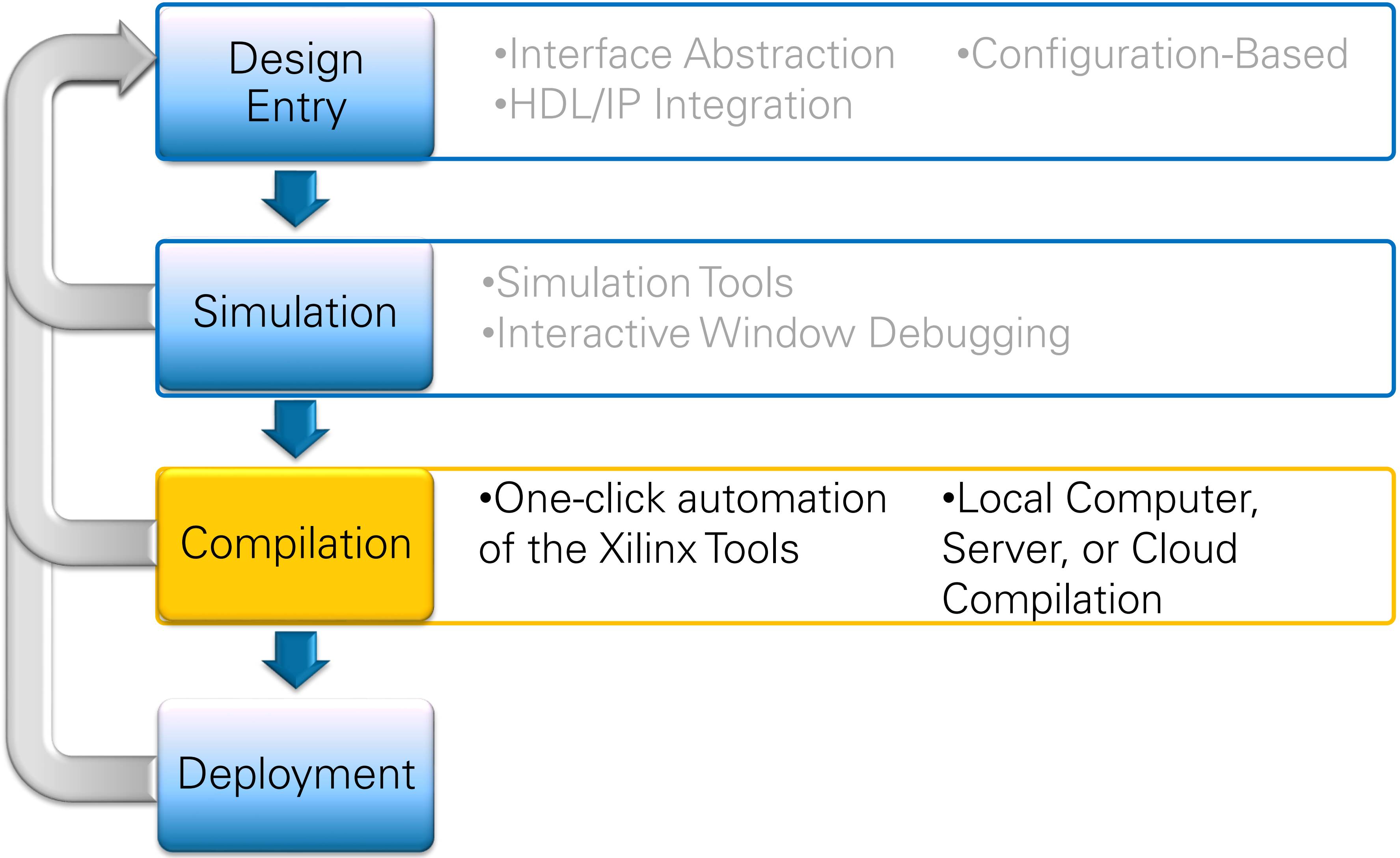
E-Stop: ●

PWM: ●

Gain: 0.00000

LabVIEW RIO Evaluation Kit.lvproj/FPGA Target

Simplified FPGA Design Flow



Design Entry

- Interface Abstraction
- HDL/IP Integration
- Configuration-Based

Simulation

- Simulation Tools
- Interactive Window Debugging

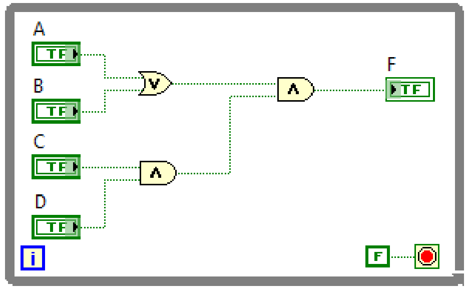
Compilation

- One-click automation of the Xilinx Tools
- Local Computer, Server, or Cloud Compilation

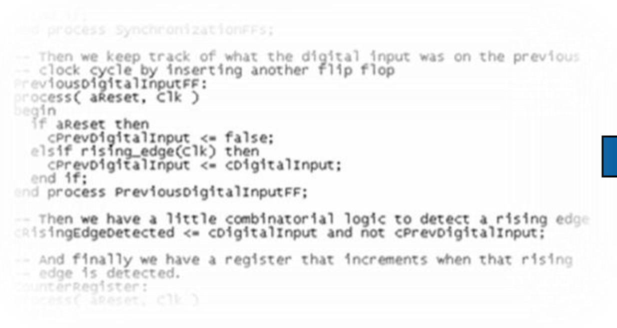
Deployment

Compilation Process

LabVIEW FPGA Code

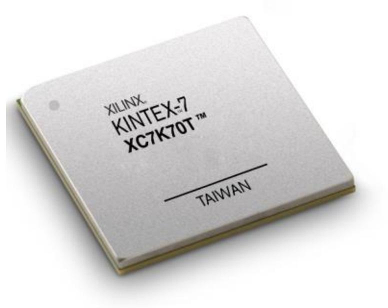


Compile VHDL through Xilinx



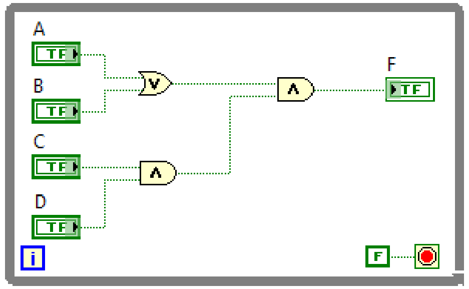
```
  end if;
end process SynchronizationFFs;

-- 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

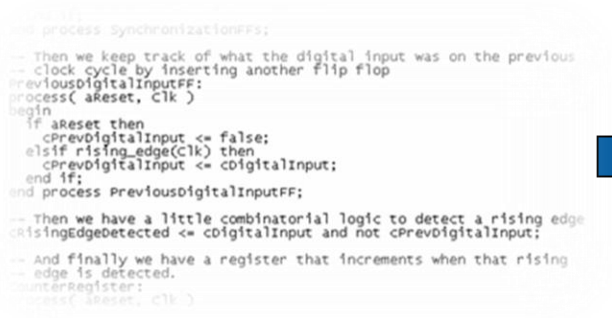


Compilation Process

LabVIEW FPGA Code

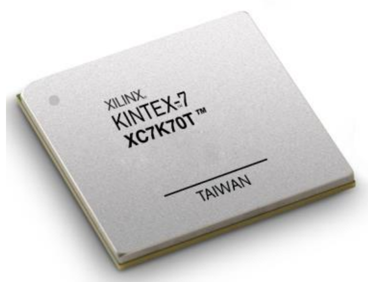


Compile VHDL through Xilinx



```
  end if;
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



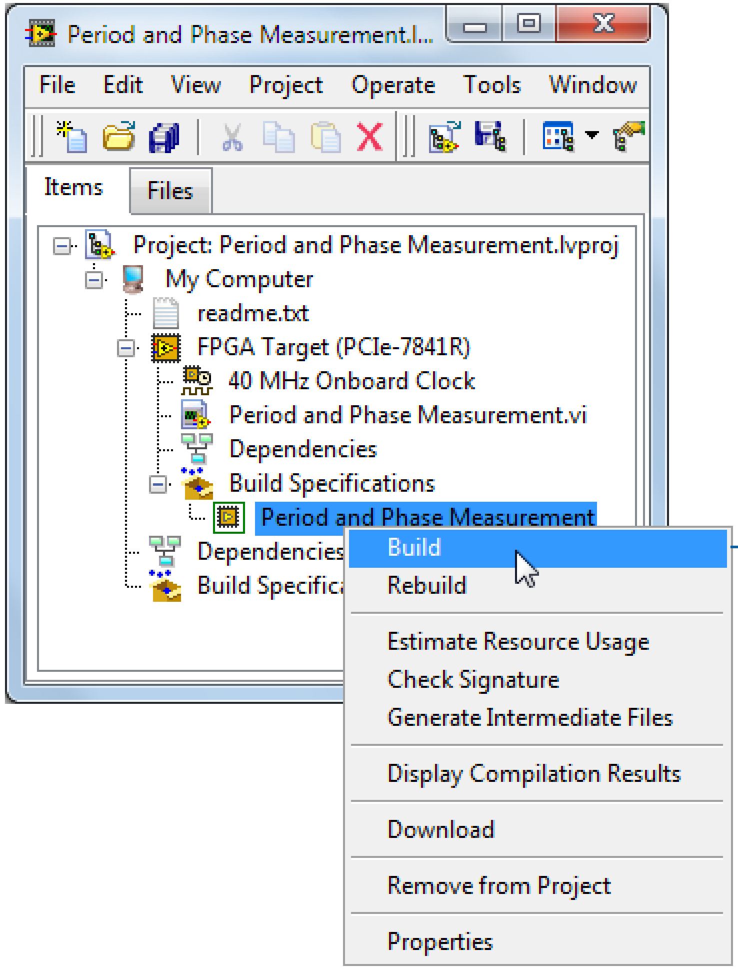
Translation
VHDL Generation

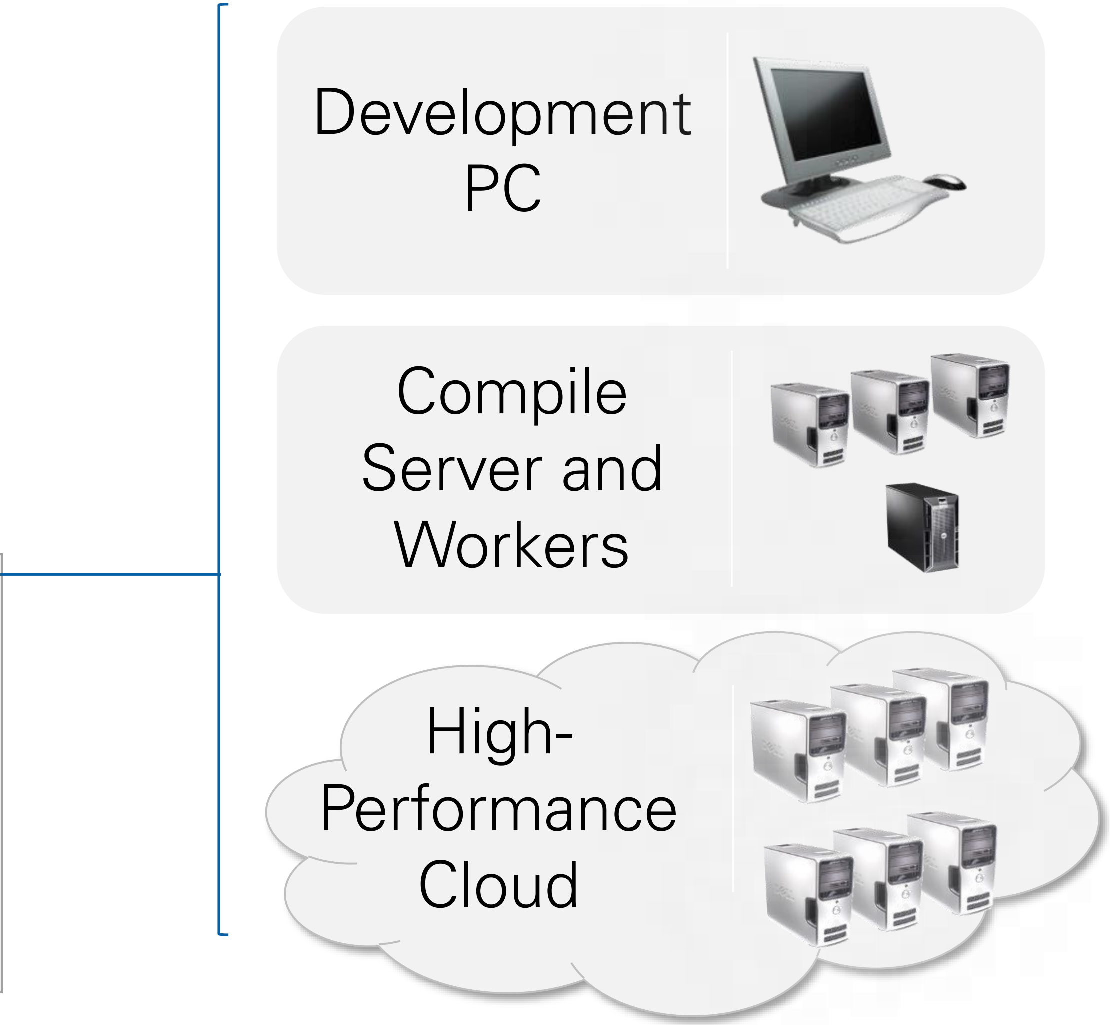
Optimization
Analysis & Logic Reduction

Synthesis
Place & Route Timing Verification

Bit Stream Generation
Download & Run

One-Click Deployment and Compilation



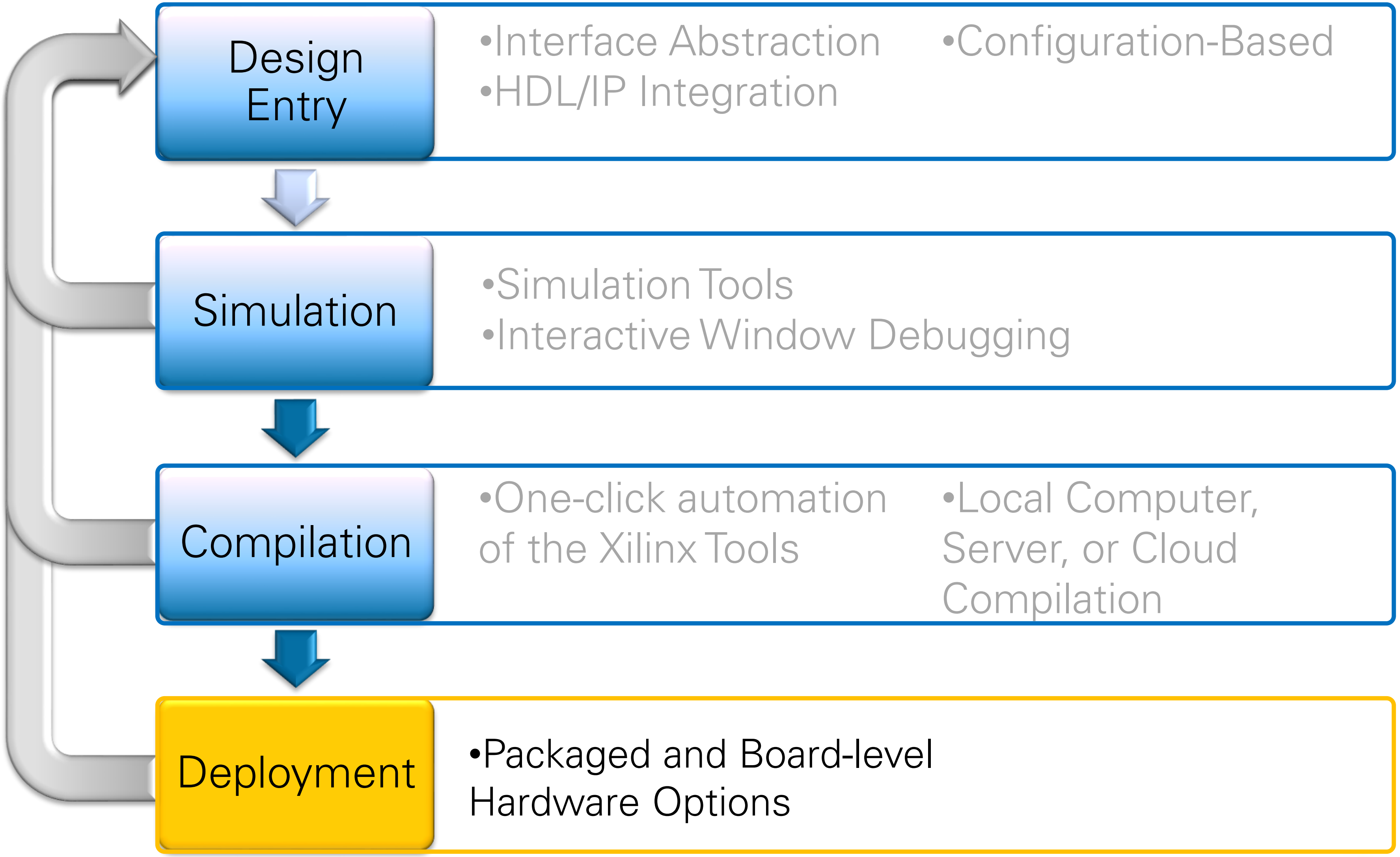


Development PC

Compile Server and Workers

High-Performance Cloud

Simplified FPGA Design Flow



Design Entry

- Interface Abstraction
- HDL/IP Integration
- Configuration-Based

Simulation

- Simulation Tools
- Interactive Window Debugging

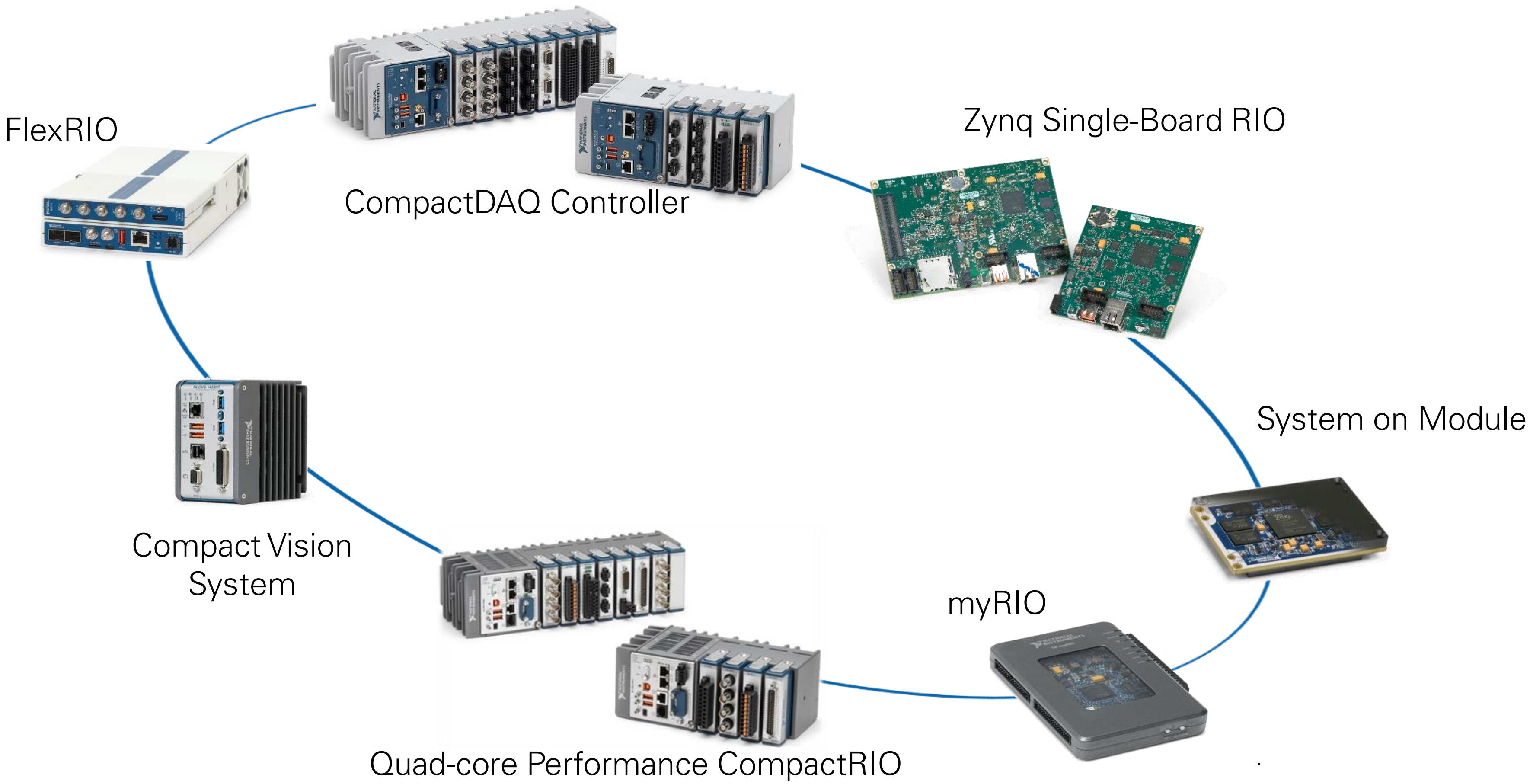
Compilation

- One-click automation of the Xilinx Tools
- Local Computer, Server, or Cloud Compilation

Deployment

- Packaged and Board-level Hardware Options

Integration with the Latest Hardware Products



FlexRIO

CompactDAQ Controller

Zynq Single-Board RIO

System on Module

myRIO

Quad-core Performance CompactRIO

Compact Vision System

NI System on Module

Core processing unit for an embedded system

- Minimizes design time and risk
  - Save time and risk with off-the-shelf hardware and software
  - Quickly prototype with off-the-shelf NI embedded targets and I/O
- Develop high-speed and advanced applications with an FPGA without HDL expertise
- Designed, tested, and validated for reliable deployments

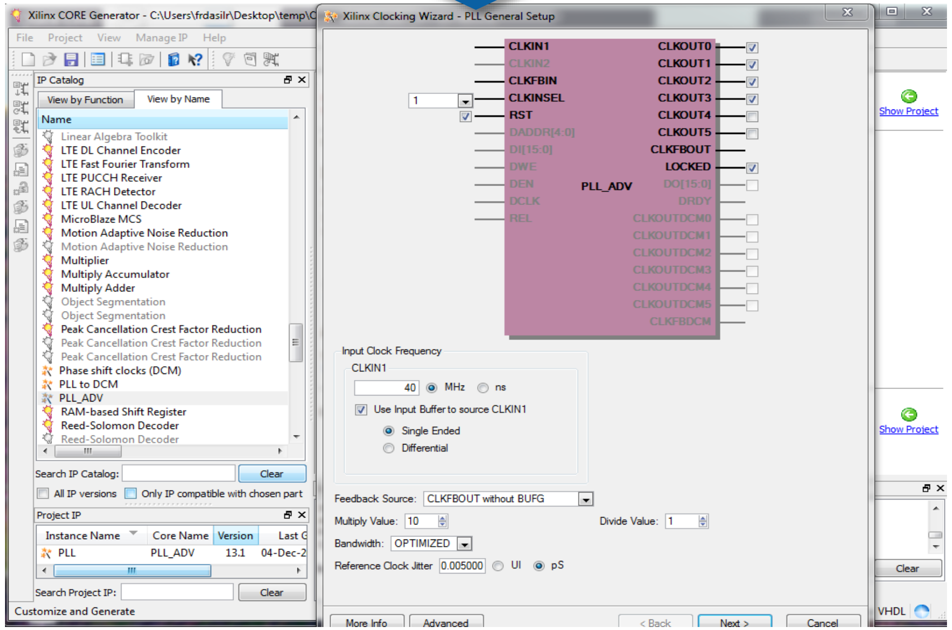
Xilinx® Tools

IDE (ISE, VIVADO) and primitive functions

Xilinx® Tools

- NI provides some IP from LV FPGA like FFT, FIRs…

Does Xilinx® has other IP for our FPGA?
-> **Coregen**



Xilinx CORE Generator - C:\Users\frdasilr\Desktop\temp\C

IP Catalog — View by Function | View by Name

Name
- Linear Algebra Toolkit
- LTE DL Channel Encoder
- LTE Fast Fourier Transform
- LTE PUCCH Receiver
- LTE RACH Detector
- LTE UL Channel Decoder
- MicroBlaze MCS
- Motion Adaptive Noise Reduction
- Motion Adaptive Noise Reduction
- Multiplier
- Multiply Accumulator
- Multiply Adder
- Object Segmentation
- Object Segmentation
- Peak Cancellation Crest Factor Reduction
- Peak Cancellation Crest Factor Reduction
- Peak Cancellation Crest Factor Reduction
- Phase shift clocks (DCM)
- PLL to DCM
- PLL_ADV
- RAM-based Shift Register
- Reed-Solomon Decoder
- Reed-Solomon Decoder

Search IP Catalog: Clear

□ All IP versions □ Only IP compatible with chosen part

Project IP

Instance Name	Core Name	Version	Last G
PLL	PLL_ADV	13.1	04-Dec-2

Search Project IP: Clear

Customize and Generate

Xilinx Clocking Wizard - PLL General Setup

CLKINSEL: 1

☑ RST

☑ CLKOUT0 ☑ CLKOUT1 ☑ CLKOUT2 ☑ CLKOUT3 □ CLKOUT4 □ CLKOUT5 ☑ LOCKED □ DO[15:0] □ CLKOUTDCM0 □ CLKOUTDCM1 □ CLKOUTDCM2 □ CLKOUTDCM3 □ CLKOUTDCM4 □ CLKOUTDCM5

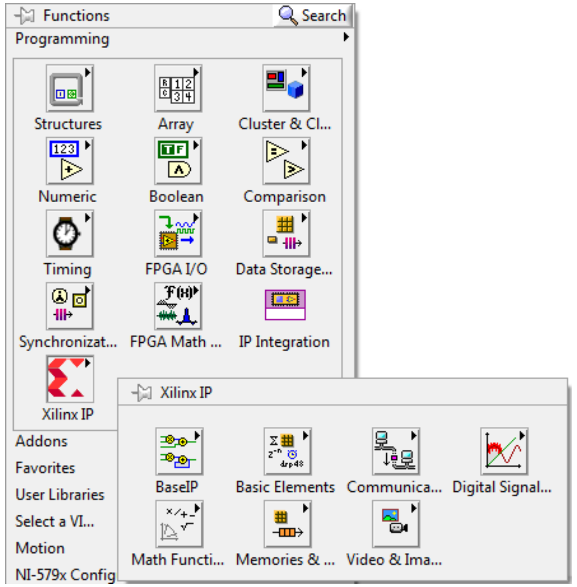
Input Clock Frequency
CLKIN1: 40 ● MHz ○ ns
☑ Use Input Buffer to source CLKIN1
● Single Ended
○ Differential

Feedback Source: CLKFBOUT without BUFG

Multiply Value: 10 Divide Value: 1

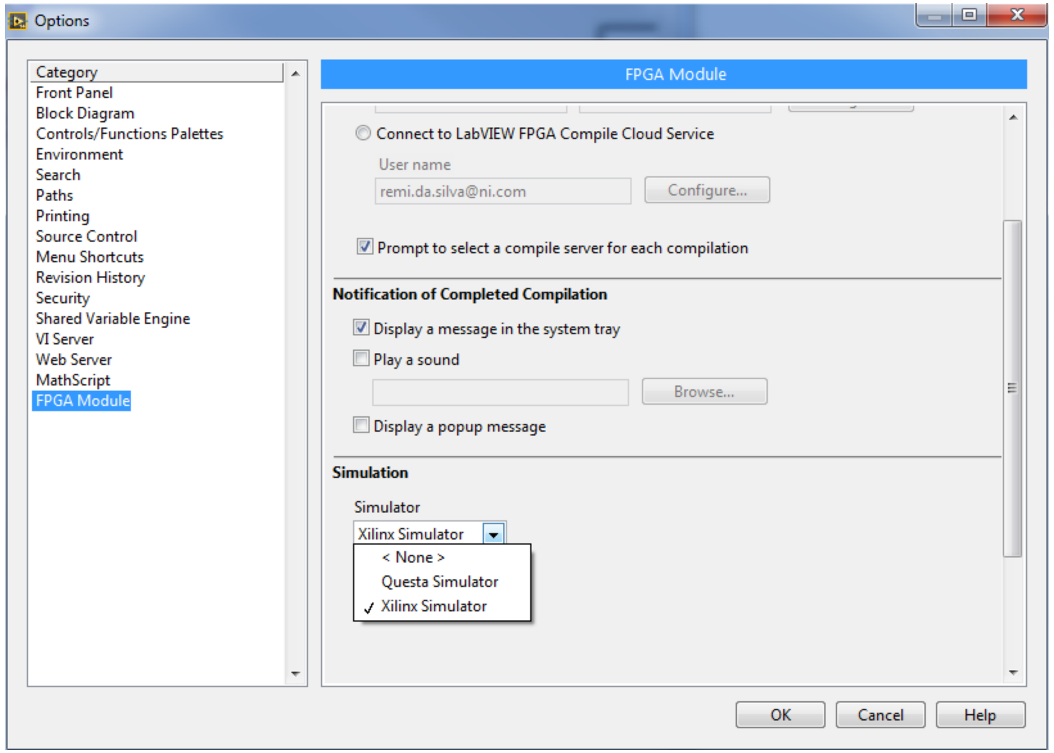
Bandwidth: OPTIMIZED

Reference Clock Jitter 0.005000 ○ UI ● pS



Third Party Simulation

- Used to create detailed models of timing and functional behavior of designs
- Xilinx ISIM is shipped with the Xilinx Tools
- ModelSim/Questa



Options

FPGA Module

○ Connect to LabVIEW FPGA Compile Cloud Service

User name: remi.da.silva@ni.com   Configure...

☑ Prompt to select a compile server for each compilation

Notification of Completed Compilation

☑ Display a message in the system tray

□ Play a sound   Browse...

□ Display a popup message

Simulation

Simulator: Xilinx Simulator

- < None >
- Questa Simulator
- ✓ Xilinx Simulator