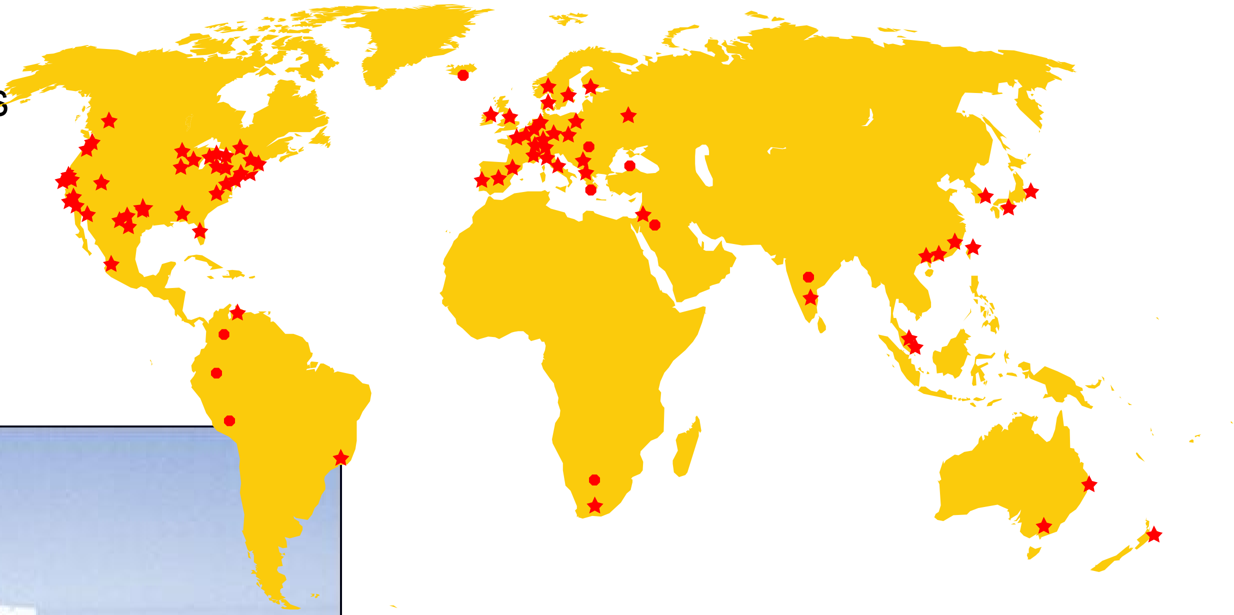
EE249 Lab Graphical System Design

October 4, 2012

Hugo A. Andrade, Kaushik Ravindran, Jeff C. Jensen

National Instruments

- More than 50 international branches in over 45 countries
- Corporate headquarters in Austin, TX



Dr. James Truchard

- 6,400+ employees
- More than 1,000 products

National Instruments

Offering graphical system design solutions to the Test and Measurement and Industrial Embedded

Revenue: $1.04B revenue in 2011, $292M revenue in Q2 2012

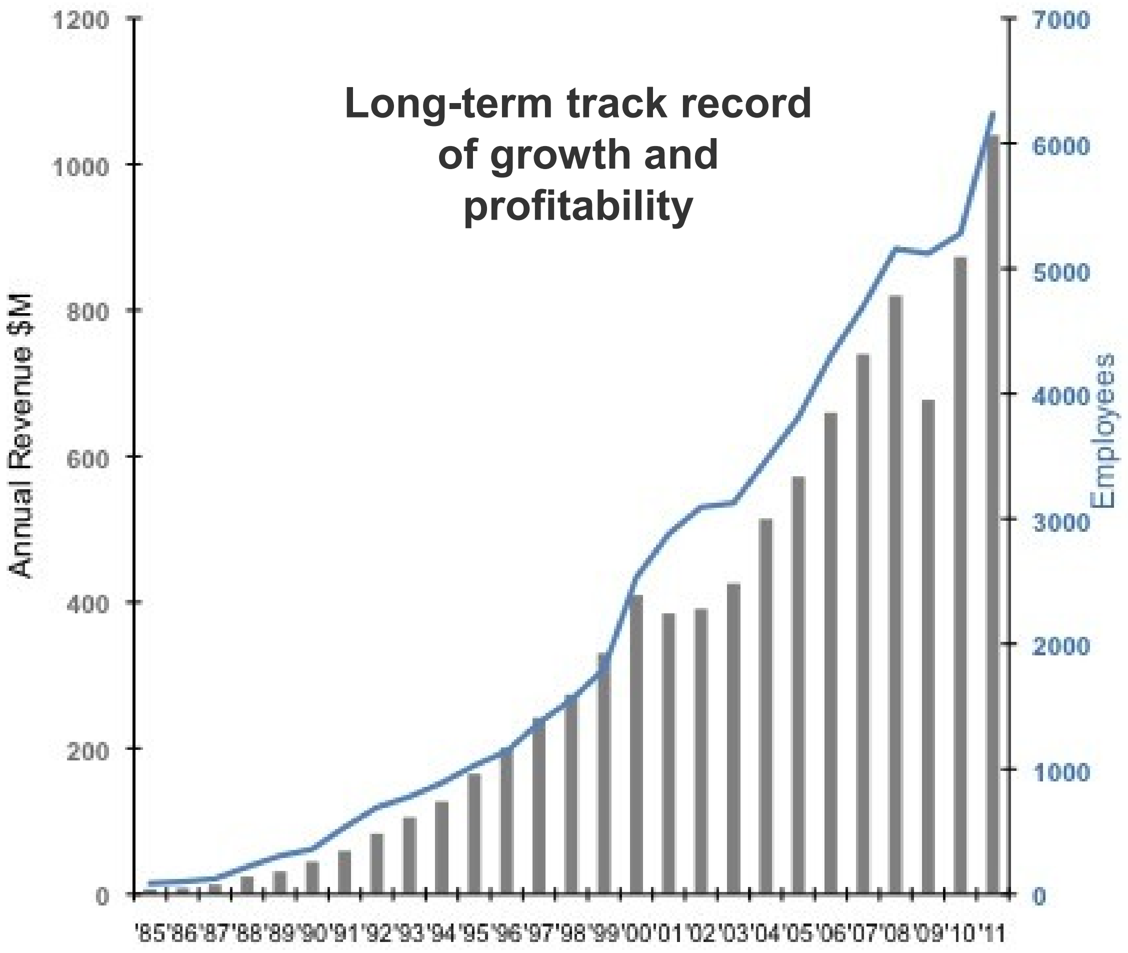
Global Operations: Approximately 6,400 employees; operations in more than 45 countries

Broad customer base: More than 35,000 companies served annually

Diversity: No industry >15% of revenue

Culture: *FORTUNE's* 100 Best Companies to Work For list for 13 consecutive years and top 25 companies to work for worldwide by FORTUNE Magazine and the Great Places to Work Institute

Strong Cash Position: Cash and short-term investments of $351M at June 30, 2012

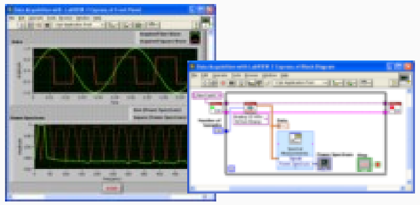


What We Do

National Instruments equips engineers and scientists with tools that accelerate productivity, innovation, and discovery

Low-Cost Modular Measurement and Control Hardware

Productive Software Development Tools



Highly Integrated Systems Platforms

Diversity of Customers

hp Cisco Systems U.S. DEPARTMENT OF DEFENSE U.S. DEPARTMENT OF ENERGY EXXON ALCOA bp SIEMENS Lucent Technologies Bell Labs Innovations 3com SONY GOODYEAR MOTOROLA Ferrari TRW YOKOGAWA ALCATEL BAE SYSTEMS

LOCKHEED MARTIN Honeywell CORNING Discovering Beyond Imagination NOKIA ABB Kodak Microsoft IBM TOSHIBA Seagate Agilent Technologies 3M NASA BOEING DUPONT Ford ROHDE&SCHWARZ Fisher-Rosemount

AMD P&G Dow FOXBORO Rockwell PHILIPS XEROX FUJI ELECTRIC AT&T Toyota HONDA Tektronix NISSAN ACTERNA intel TERADYNE ERICSSON

- Top 100 customers ≈ 35% of revenue
- More than 30,000 customers in more than 90 countries
- 95% of Fortune 500 manufacturing companies have adopted Virtual Instrumentation

Diversity of Applications

No Industry >15% of Revenue in 2011

Academic

Advanced Research

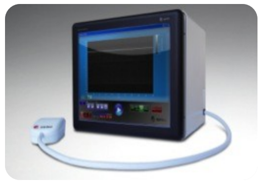
Automotive

Big Physics

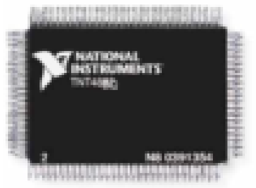
Consumer Electronics

Defense/Aerospace

Energy


Life Sciences

Mobile Devices


Semiconductors

Technology Overview

NAE: Engineering Grand Challenges

Advance health informatics

Engineer the tools of scientific discovery

Reverse-engineer the brain

Provide energy from fusion

Engineer better medicines

Provide access to clean water

Enhance virtual reality

Restore and improve urban infrastructure

Develop carbon sequestration methods

Advance personalized learning

Make solar energy economical

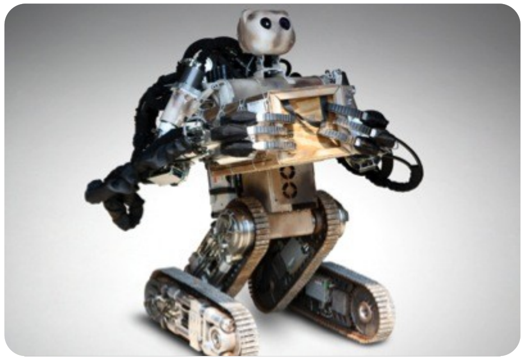
Prevent nuclear terror

Secure cyberspace

Manage the nitrogen cycle

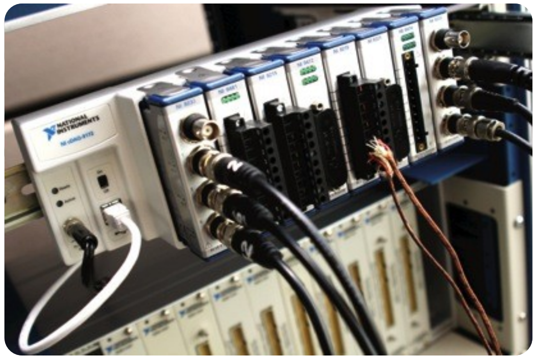
http://www.engineeringchallenges.org/

Build Better Systems Faster



Higher Performance

Lower Costs

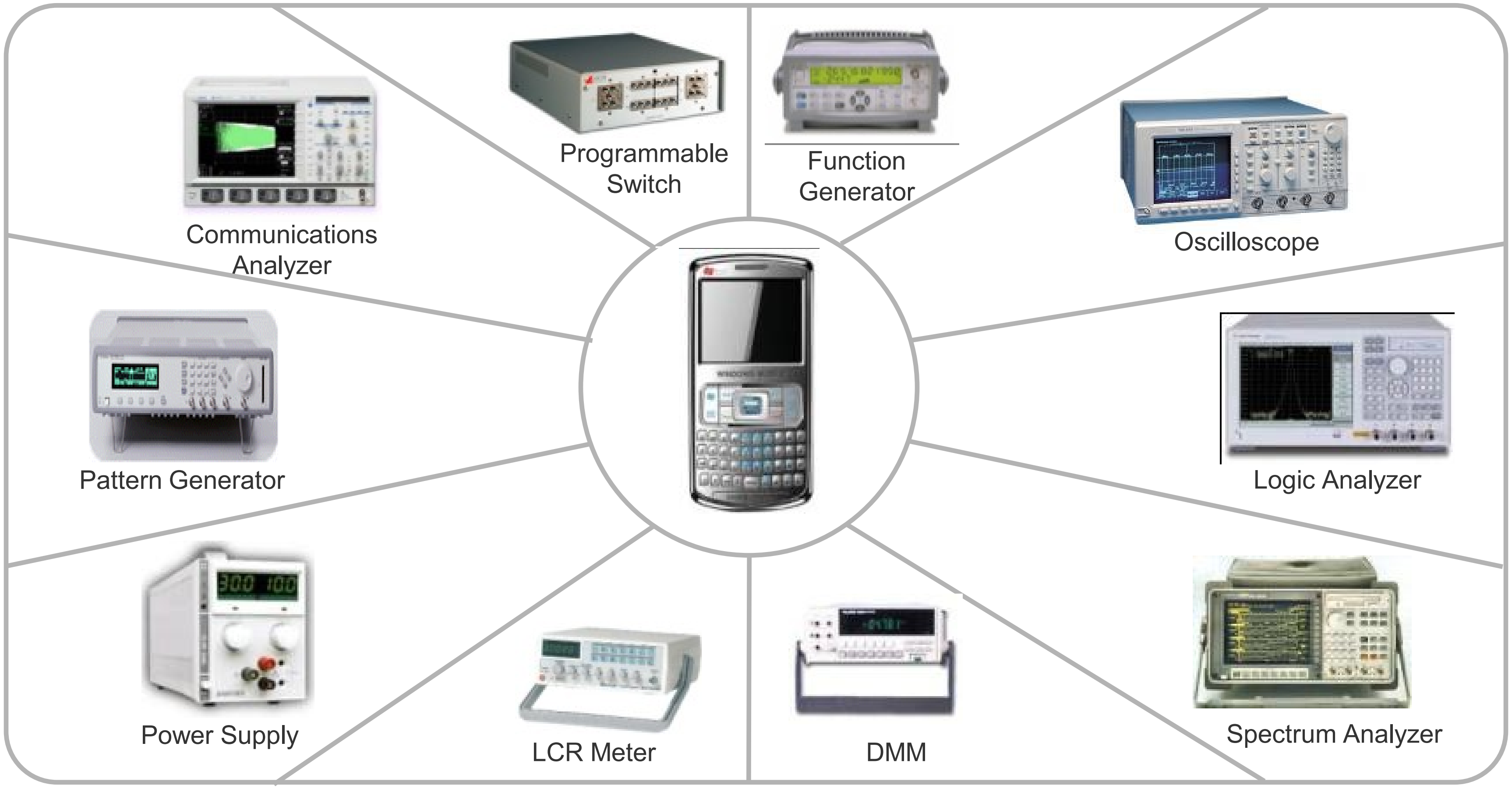


Better Integration

We equip engineers and scientist with the tools that accelerate productivity, innovation and discovery

NATIONAL INSTRUMENTS™

The Traditional Approach to Automated Test



Source: Agilent, Keithley, and Nicolet

The Customer Decision: Build or Buy in Embedded

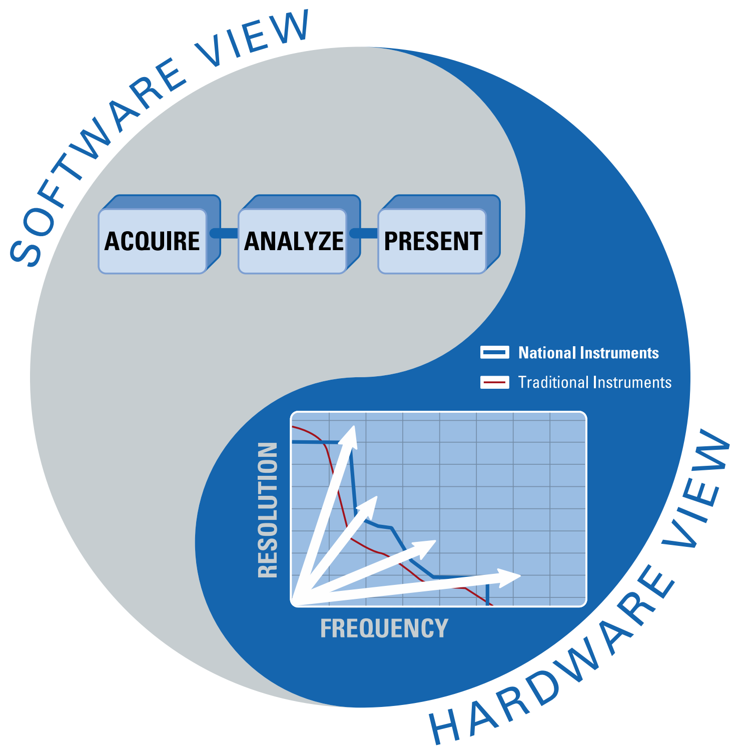
Build

- Custom HW/SW solution
- Long lead times for new product
- Significant resource requirements

Buy

- Off-the-shelf hardware with LabVIEW
- Use less resources because systems are pre-built
- Faster time to market

The Virtual Instrumentation Approach

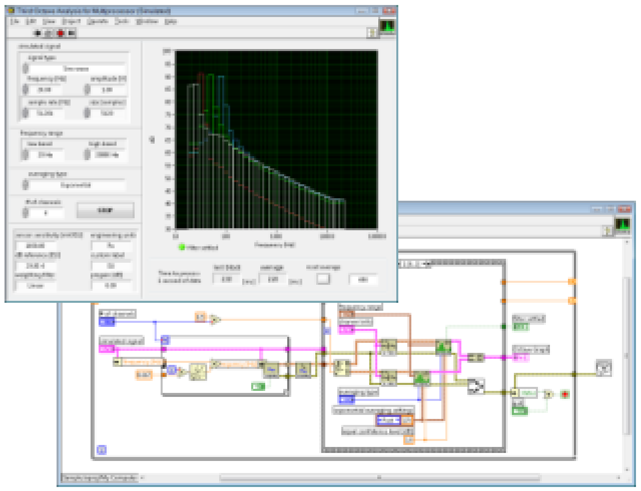


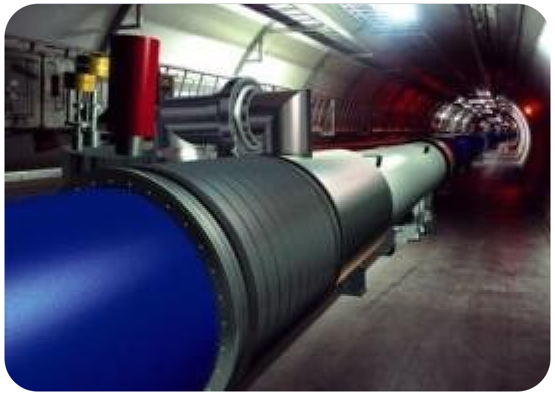
The Software Is the Instrument

Empowering Users Through Software

Providing unique differentiation and preserving customer investments

LEGO® MINDSTORMS® NXT
"the smartest, coolest toy of the year"





CERN Large Hadron Collider
"the most powerful instrument on earth"

Graphical System Design

A Platform-Based Approach

Test

Monitor

Embedded

Control

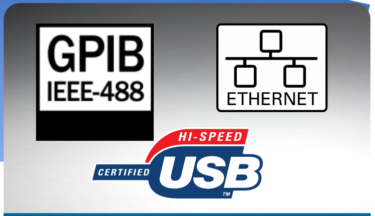
Cyber Physical

NATIONAL INSTRUMENTS LabVIEW™

Desktops and PC-Based DAQ

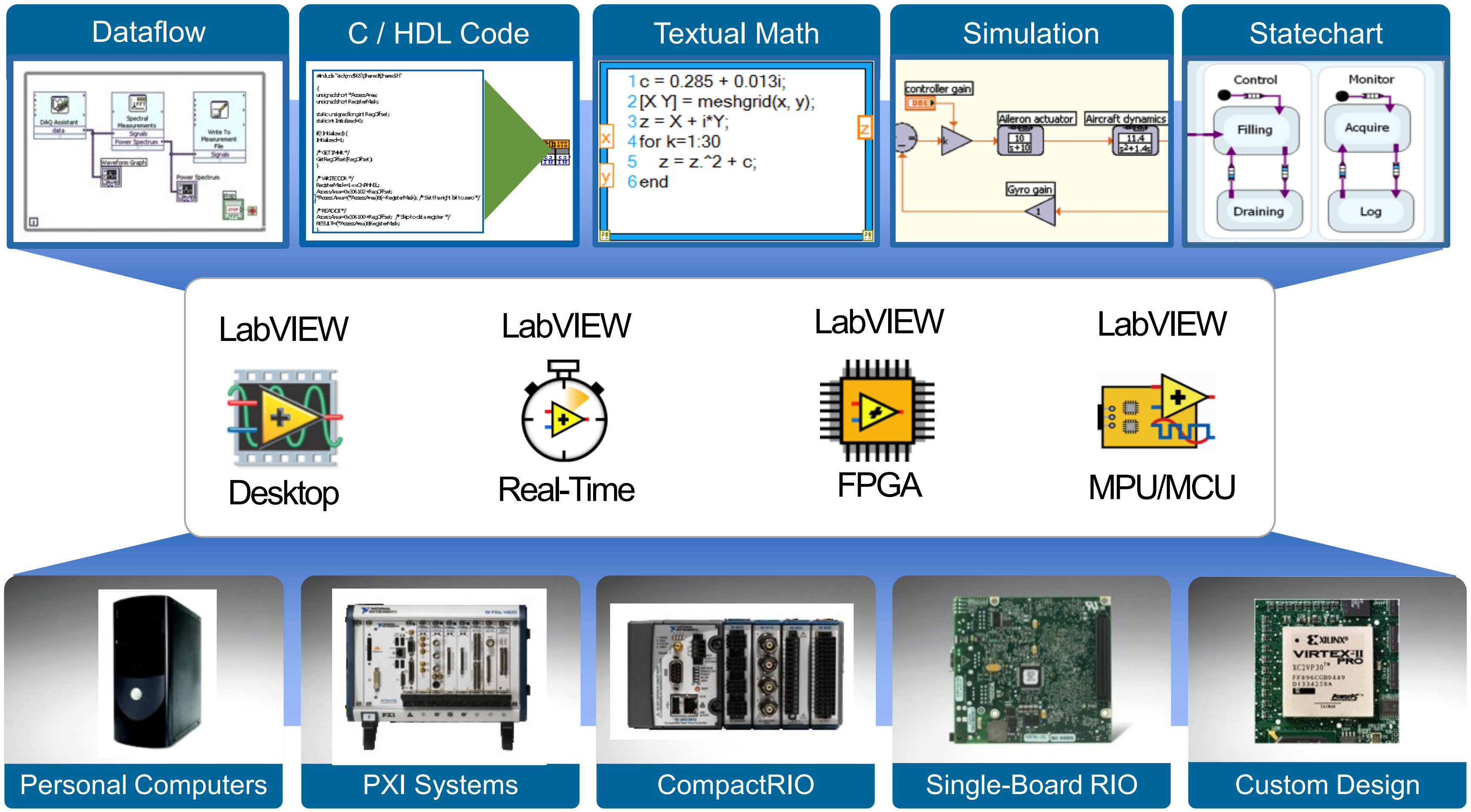
PXI and Modular Instruments

RIO and Custom Designs



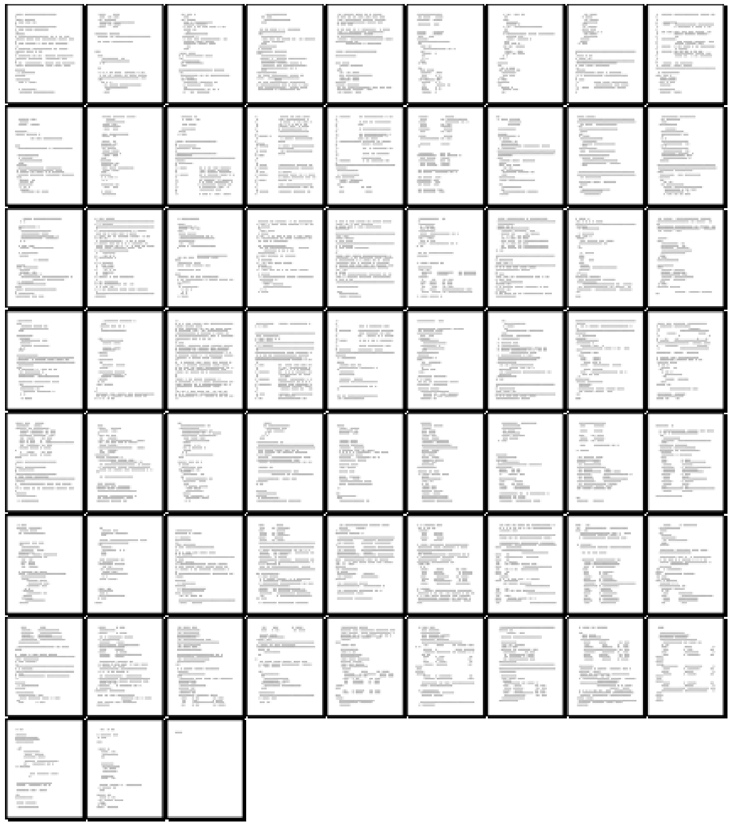
Open Connectivity with 3rd Party I/O

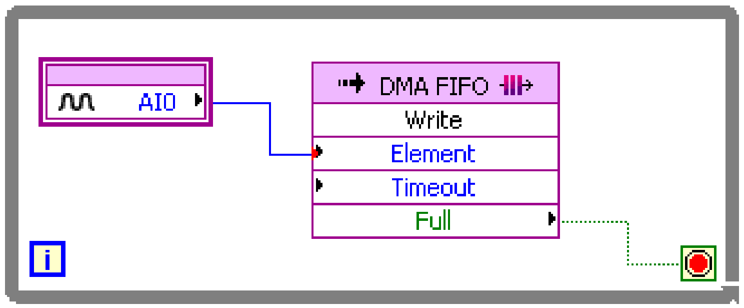
System Design to Deployment



```
c = 0.285 + 0.013i;
[X Y] = meshgrid(x, y);
z = X + i*Y;
for k=1:30
   z = z.^2 + c;
end
```

Abstraction to the Pin





VHDL

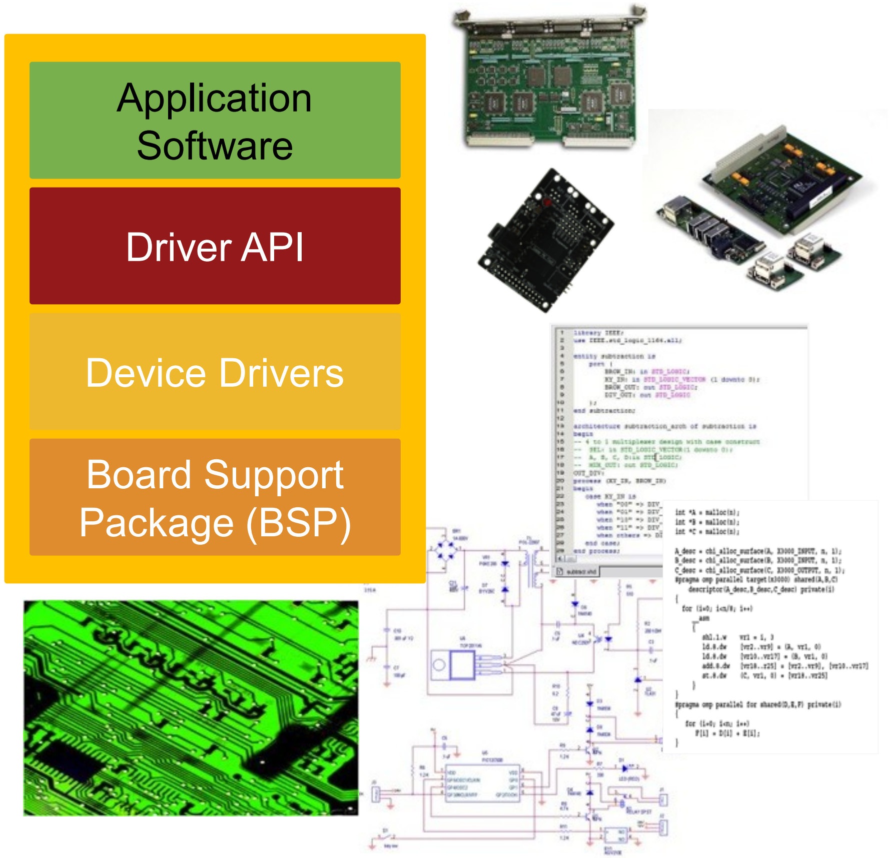
LabVIEW FPGA

Integration of Modular I/O and Commercial Technology

Box Instruments

PXI Modular Instruments

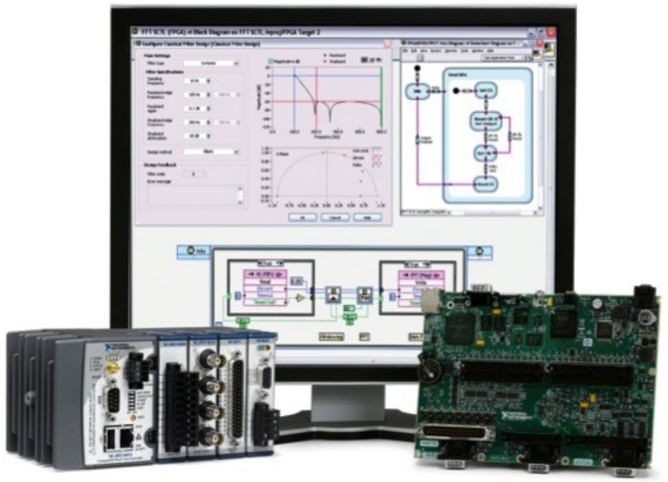
Faster System Development



```
library IEEE;
use IEEE.std_logic_1164.all;

entity subtraction is
    port (
        BROW_IN: in STD_LOGIC;
        XY_IN: in STD_LOGIC_VECTOR (1 downto 0);
        BROW_OUT: out STD_LOGIC;
        DIV_OUT: out STD_LOGIC
    );
end subtraction;

architecture subtraction_arch of subtraction is
begin
-- 4 to 1 multiplexer design with case construct
--  SEL: in STD_LOGIC_VECTOR(1 downto 0);
--  A, B, C, D:in STD_LOGIC;
--  MUX_OUT: out STD_LOGIC;
OUT_DIV:
process (XY_IN, BROW_IN)
begin
   case XY_IN is
      when "00" => DIV_
      when "01" => DIV_
      when "10" => DIV_
      when "11" => DIV_
      when others => DI
   end case;
end process;
```

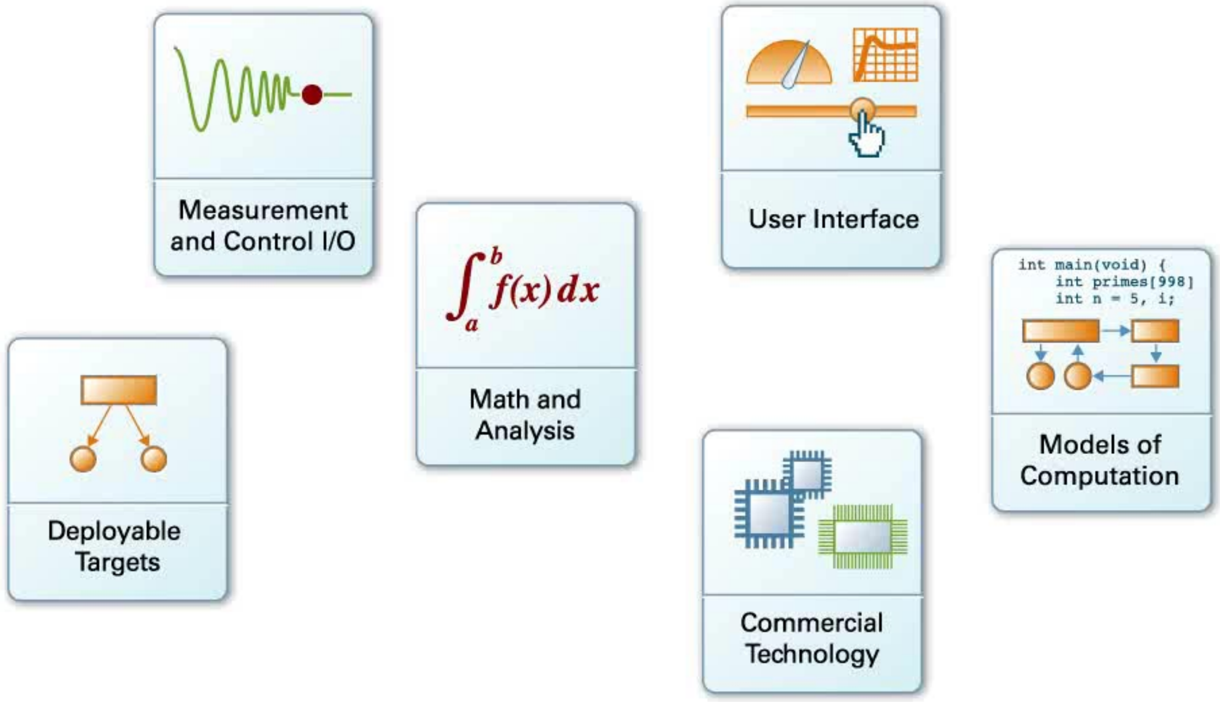
```
int *A = malloc(n);
int *B = malloc(n);
int *C = malloc(n);

A_desc = chi_alloc_surface(A, X3000_INPUT, n, 1);
B_desc = chi_alloc_surface(B, X3000_INPUT, n, 1);
C_desc = chi_alloc_surface(C, X3000_OUTPUT, n, 1);
#pragma omp parallel target(x3000) shared(A,B,C)
   descriptor(A_desc,B_desc,C_desc) private(i)
{
  for (i=0; i<n/8; i++)
     __asm
     {
       shl.1.w    vr1 = i, 3
       ld.8.dw    [vr2..vr9] = (A, vr1, 0)
       ld.8.dw    [vr10..vr17] = (B, vr1, 0)
       add.8.dw   [vr18..r25] = [vr2..vr9], [vr10..vr17]
       st.8.dw    (C, vr1, 0) = [vr18..vr25]
     }
}
#pragma omp parallel for shared(D,E,F) private(i)
{
   for (i=0; i<n; i++)
      F[i] = D[i] + E[i];
}
```

Integrating Components



Integrated System Platform

Integrating Software and Hardware Elements



Productive software and reconfigurable hardware for any system that needs measurement and control

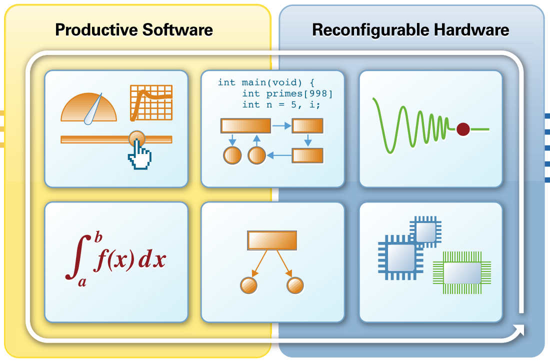
Software

Hardware

COMMUNITY
140,000+ online members
250+ registered user groups
1000+ job postings online
400K+ children through LEGO

CONNECTIVITY
9000+ instrument drivers
8000+ example programs
1000+ motion drives
1000+ smart sensors
1000+ Third-party PAC devices

COLLABORATION
280+ third-party add-ons
400+ Solution partners
1000+ value added resellers
35+ training courses



PROCESSOR
Intel, Microsoft, Freescale, Wind River
Multi-core and real-time technology

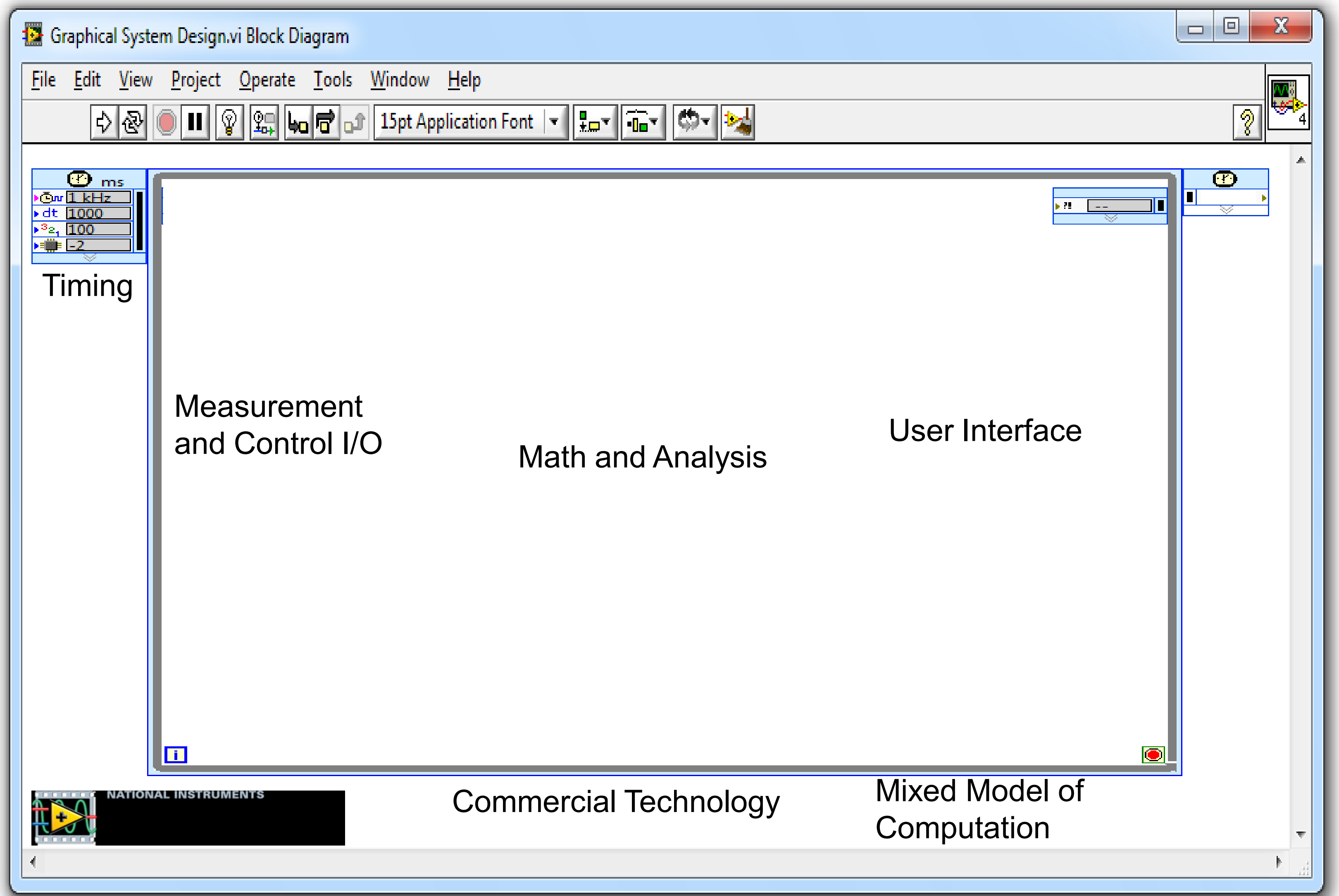
FPGA
Xilinx Virtex & Spartan
Reconfigurable hardware

IP
Control & signal processing IP & I/O drivers
Built-in graphical IP, integrate user IP

I/O
Analog Devices, Texas Instruments
Connect to any sensor & actuator

BUS
PCI/PCIe, Enet, USB, wireless, deterministic Enet, Open architecture

Productive software and reconfigurable hardware for any system that needs measurement and control


Graphical System Design.vi Block Diagram
File Edit View Project Operate Tools Window Help
15pt Application Font
ms
1 kHz
dt 1000
100
-2
NATIONAL INSTRUMENTS

Timing

Measurement and Control I/O

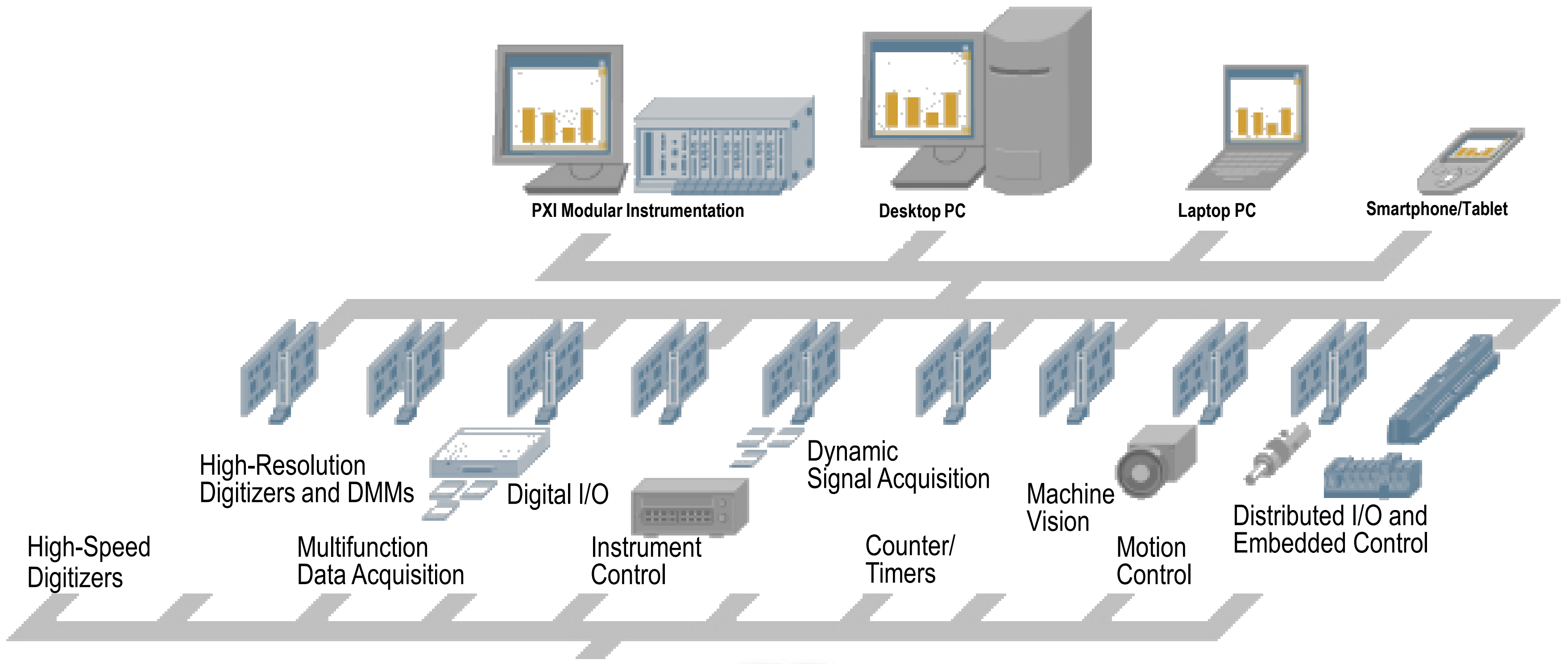
Math and Analysis

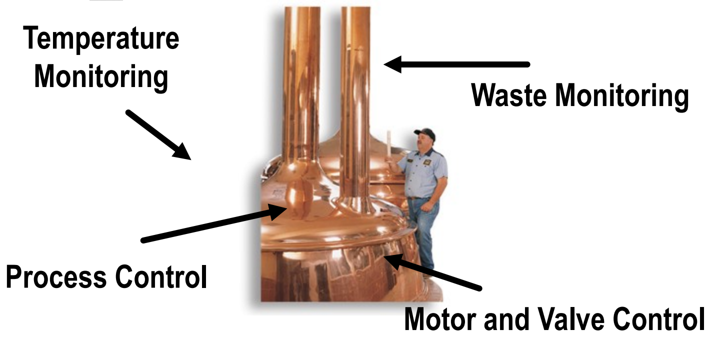
User Interface

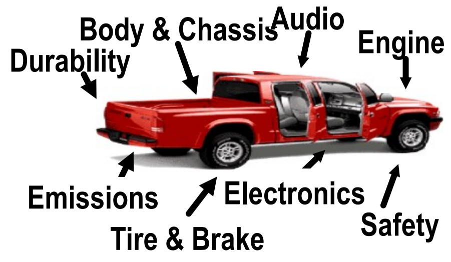
Commercial Technology

Mixed Model of Computation

Integrated Distributed Heterogeneous Platform

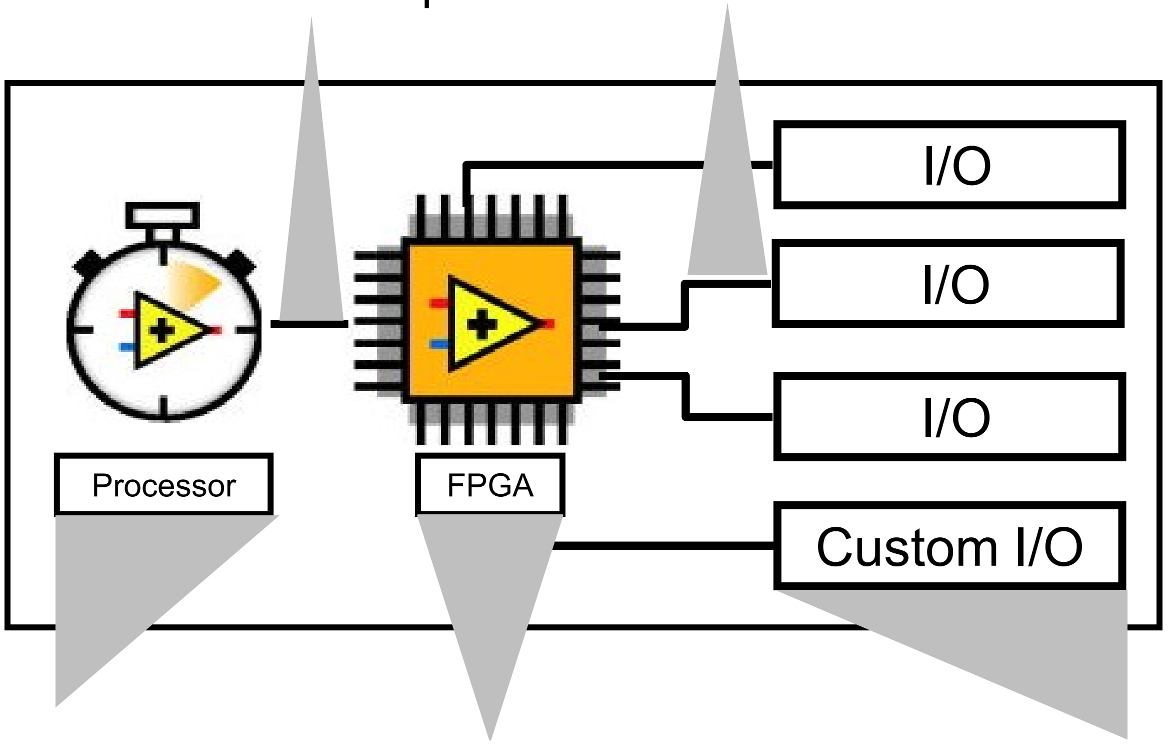

PXI Modular Instrumentation
Desktop PC
Laptop PC
Smartphone/Tablet
High-Resolution Digitizers and DMMs
Digital I/O
Dynamic Signal Acquisition
Machine Vision
Distributed I/O and Embedded Control
High-Speed Digitizers
Multifunction Data Acquisition
Instrument Control
Counter/ Timers
Motion Control





High-Speed Data Streaming
- Synchronize memory access
- Fast data links for maximum performance

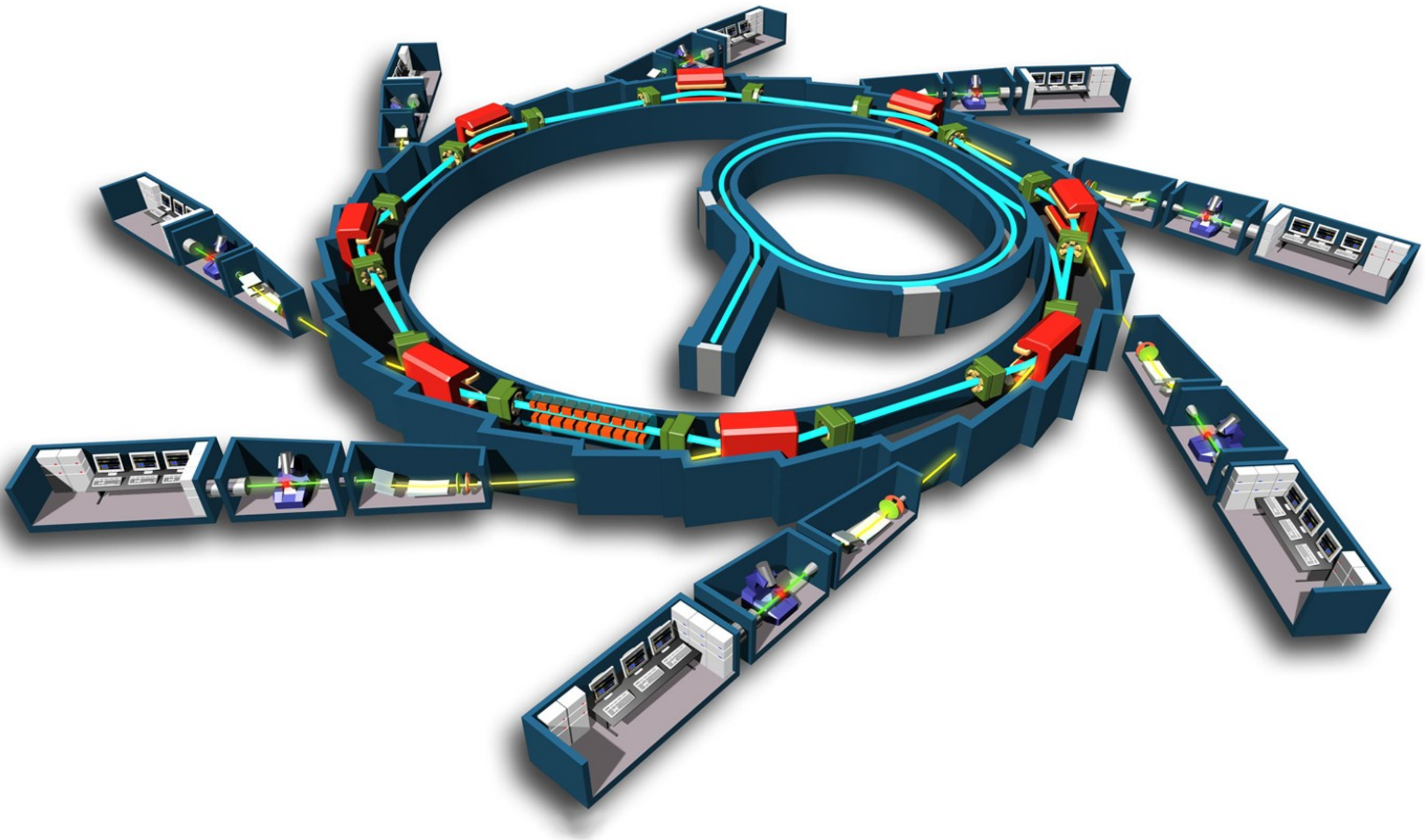
A/D Technology
- Multirate sampling
- Individual channel triggering



Microprocessors
- Floating-point processing
- Communications
- Multicore technology
- Reprogrammable

FPGAs
- High-speed control
- High-speed processing
- Reconfigurable
- True Parallelism
- High Reliability

I/O
- Custom timing & triggering
- Modular I/O
- Calibration
- Custom modules

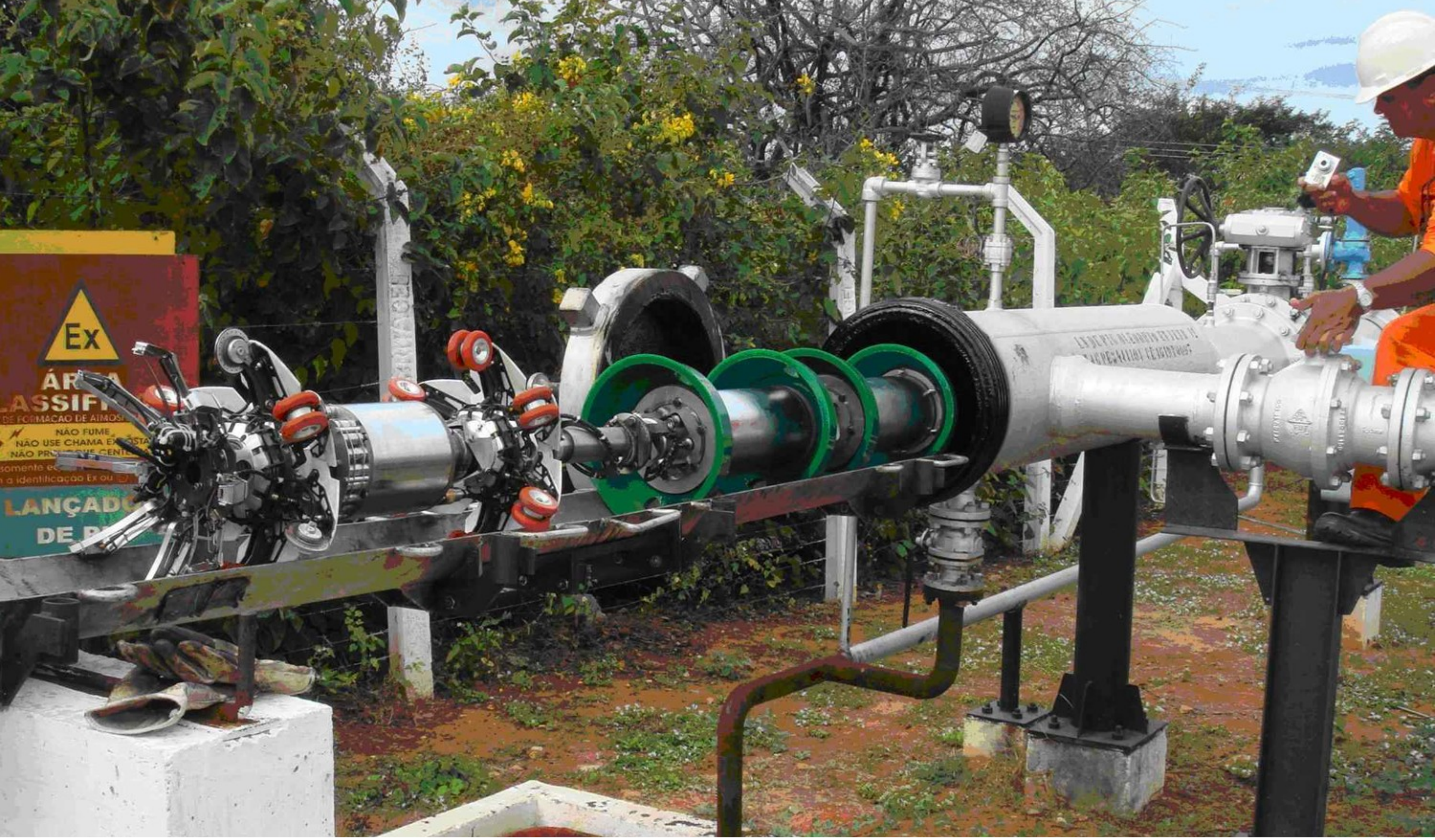


Advanced Data Acquisition

ISIS Proton Synchrotron

Semiconductor Test

Analog Devices

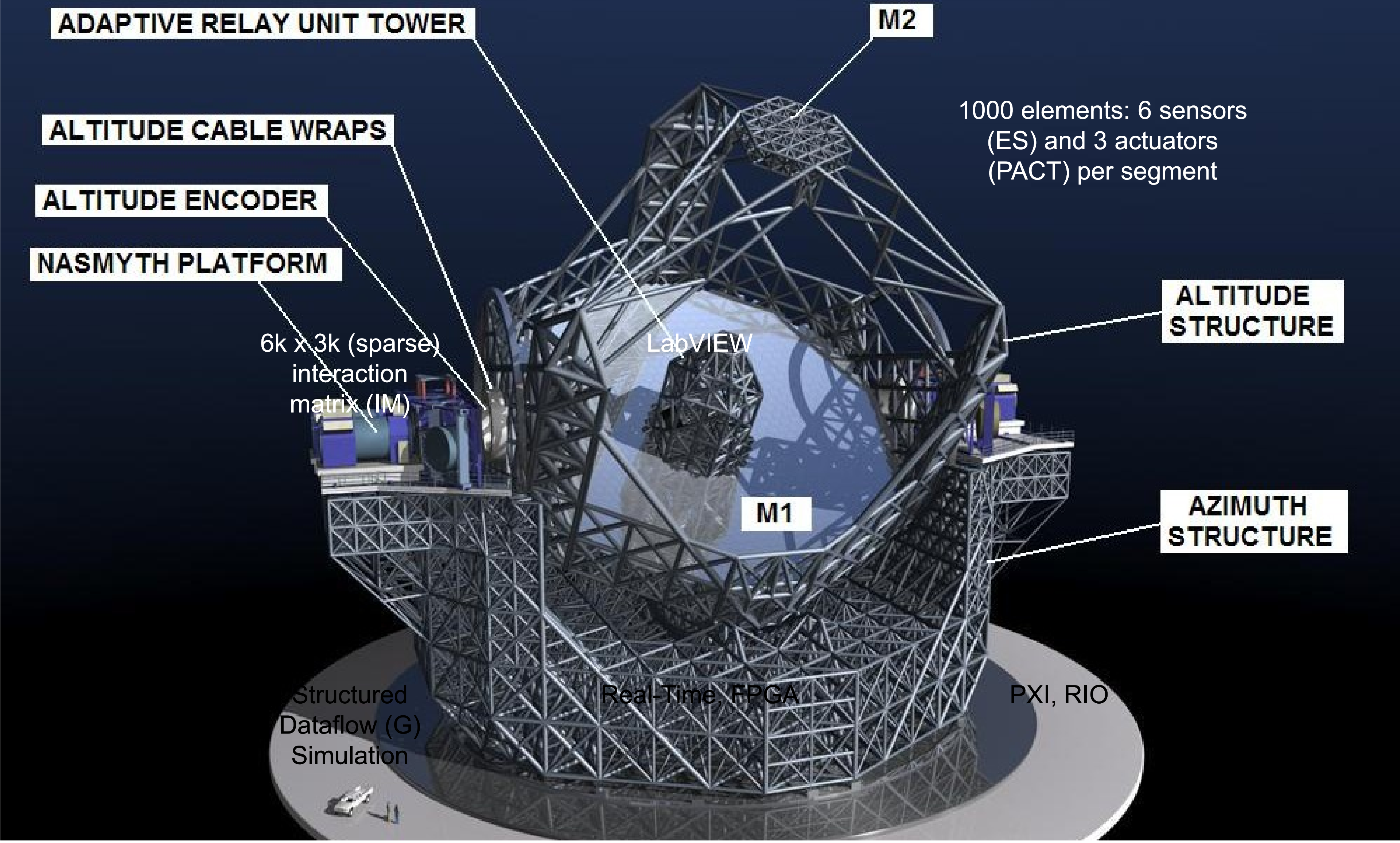


Pipeline Test and Validation

Inertial Pipeline Inspection Gauge

EcoCAR Challenge

Virginia Tech – 1st Place 2011



1000 elements: 6 sensors (ES) and 3 actuators (PACT) per segment

6k x 3k (sparse) interaction matrix (IM)

LabVIEW

Structured Dataflow (G) Simulation

Real-Time, FPGA

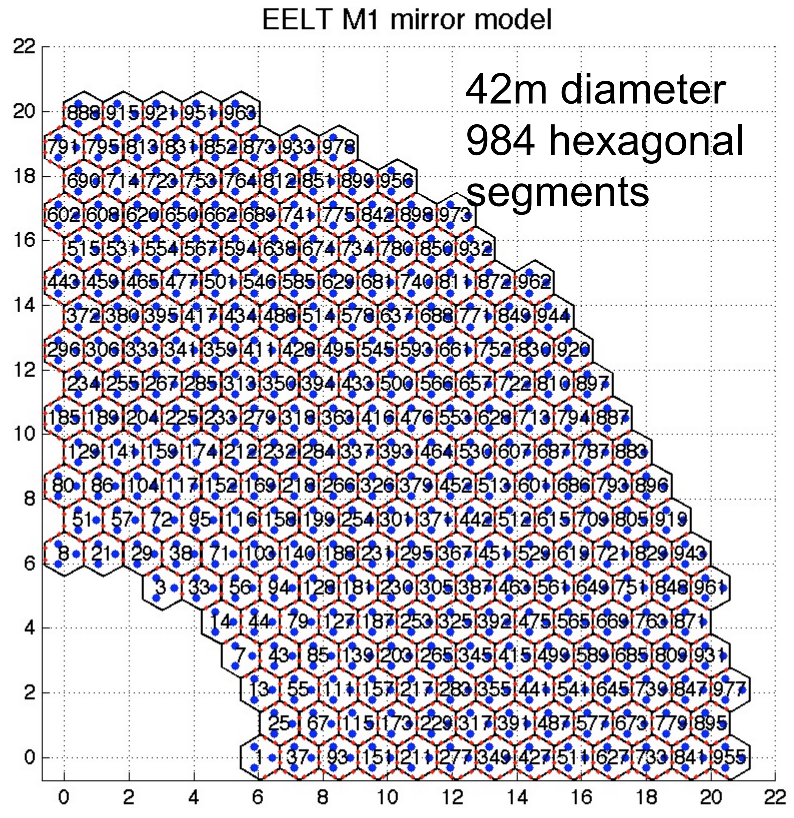
PXI, RIO

Large-scale RT Applications

European Extremely Large Telescope

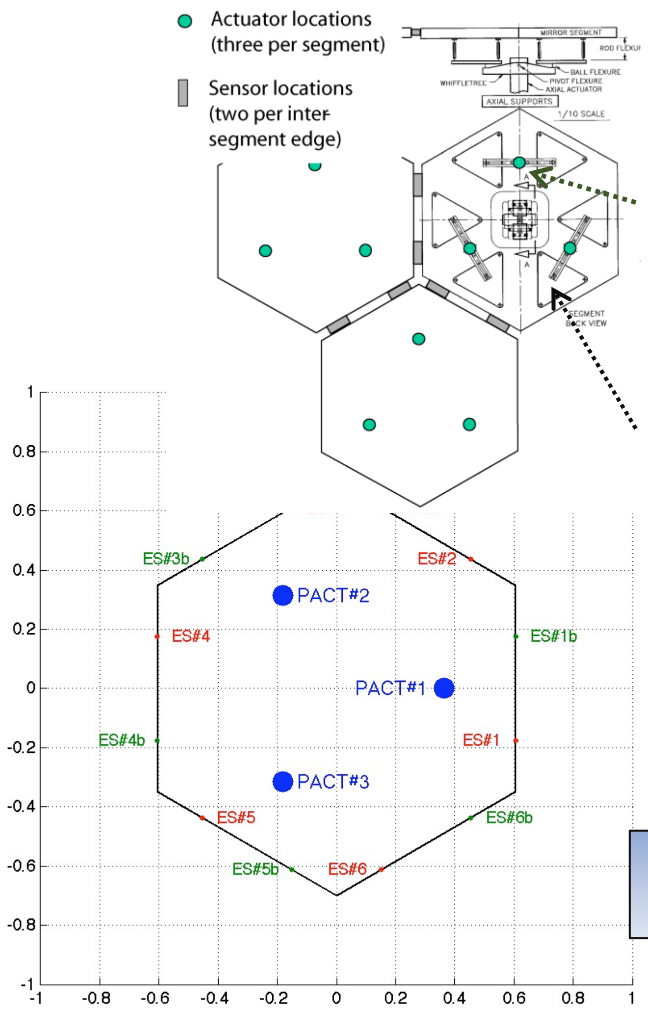
E-ELT Primary Mirror (M1): Mirror and Segment Models

Mirror Model



42m diameter
984 hexagonal segments

Segment Model



3 axial Position ACTuators

6 Edge Sensors

Total: 2952 PACTs, 5604 ESs

ni.com

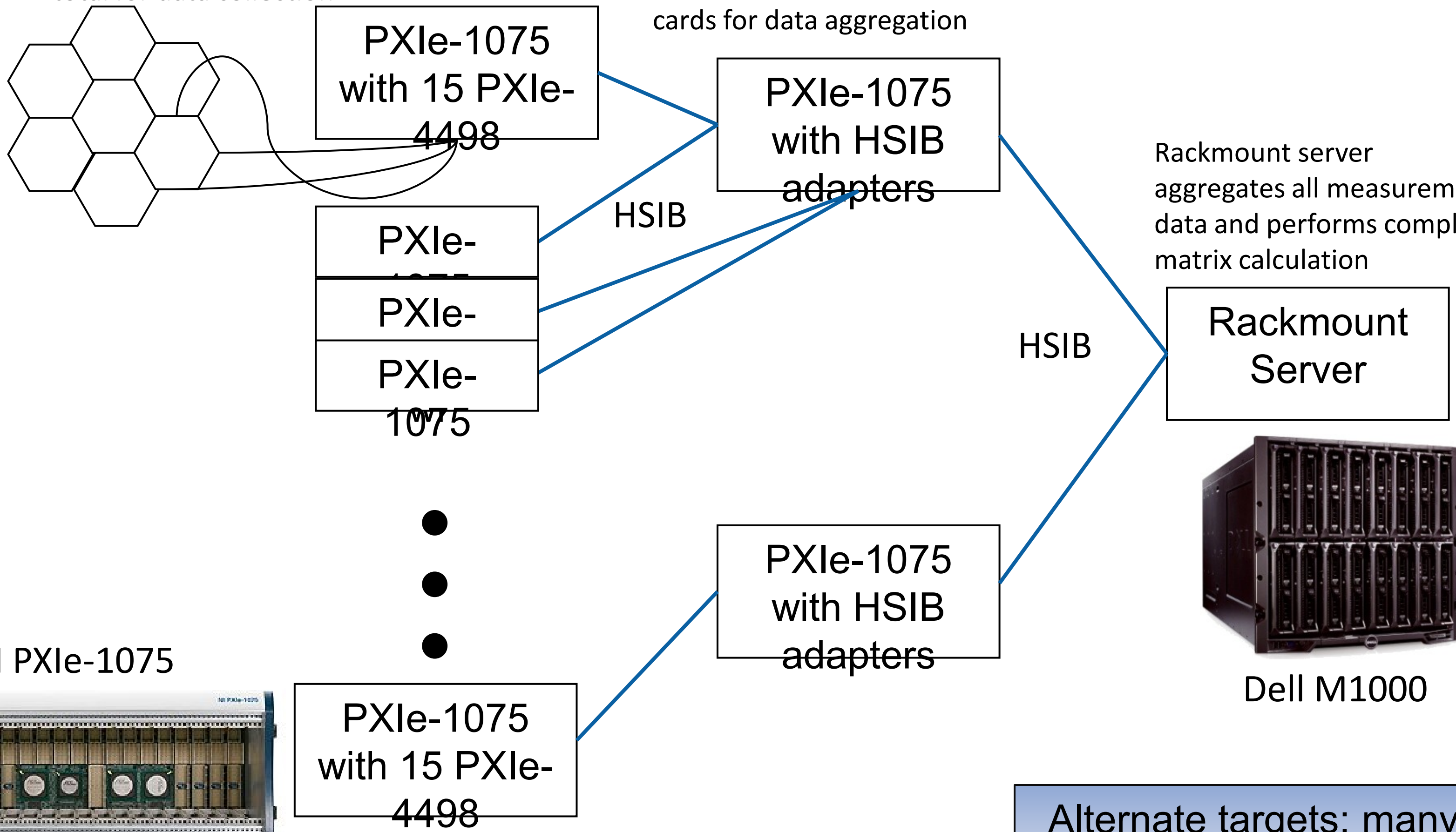
NATIONAL INSTRUMENTS™

LabVIEW^{TM} based Control Platform

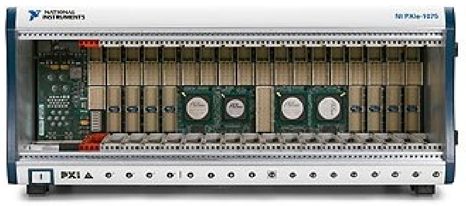
Each leaf node can measure data for 40 mirrors. 25 PXIe-1075 chassis needed in total for data collection

Need 2 PXIe-1075 chassis with embedded controller and HSIB cards for data aggregation

Rackmount server aggregates all measurement data and performs complex matrix calculation


PXIe-1075 with 15 PXIe-4498
PXIe-1075
PXIe-1075
PXIe-1075
HSIB
PXIe-1075 with HSIB adapters
HSIB
Rackmount Server
PXIe-1075 with HSIB adapters
PXIe-1075 with 15 PXIe-4498

NI PXIe-1075



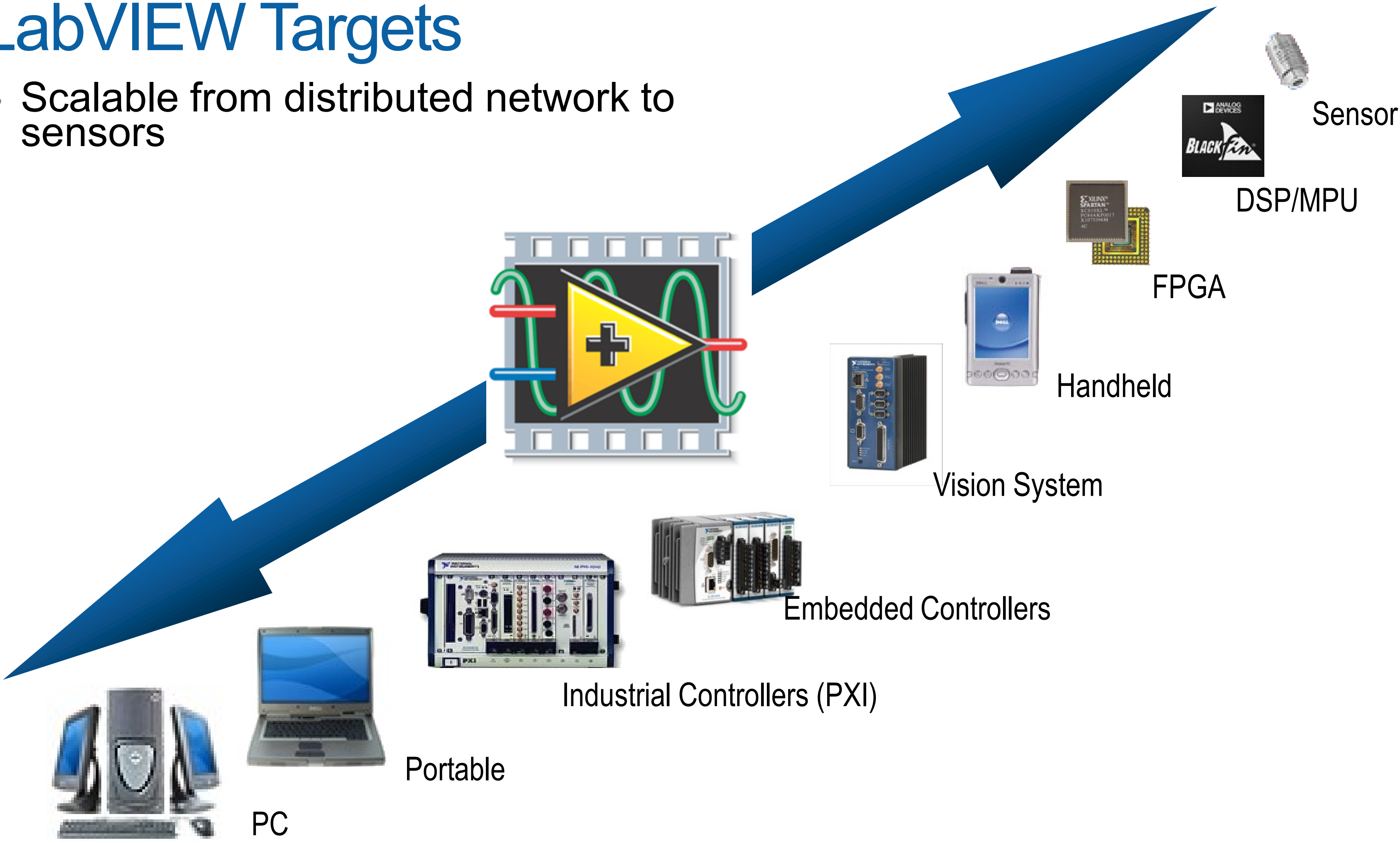
NI PXIe-4498

Dell M1000

Alternate targets: many-core, CPU+GPU, FPGA accelerators

LabVIEW Targets

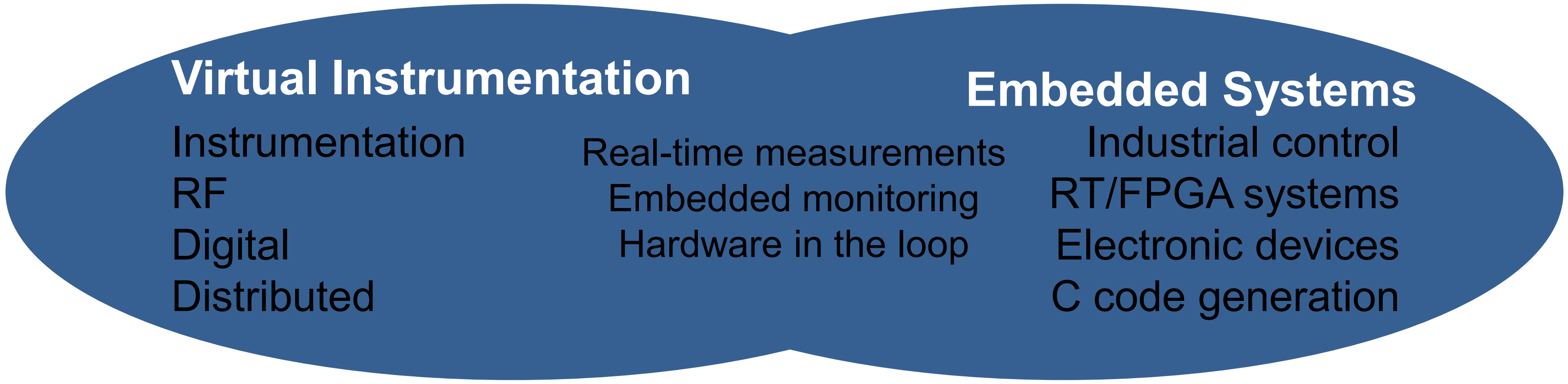
- Scalable from distributed network to sensors

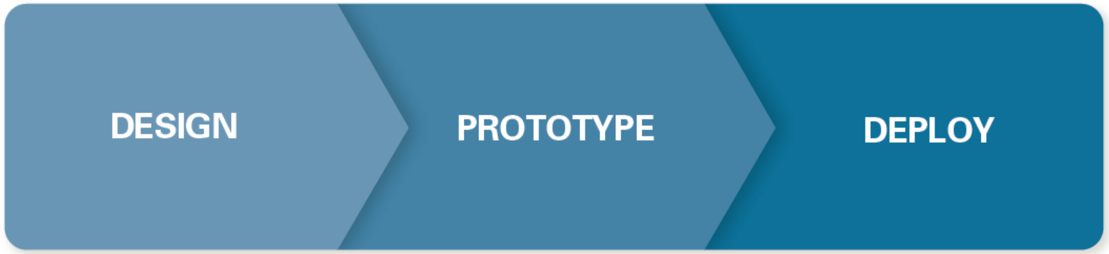


National Instruments Vision *Evolved*

"To do for embedded what the PC did for the desktop."

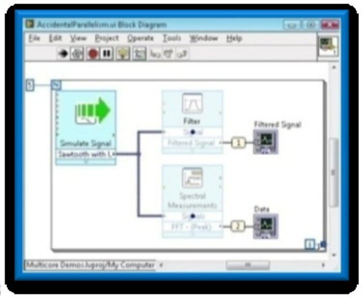
Graphical System Design



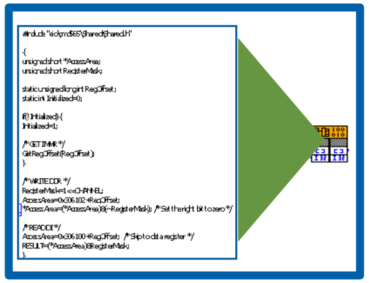


High-Level Development Tools

Data Flow

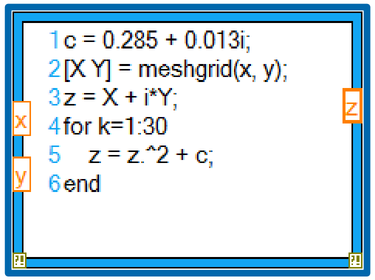


C Code



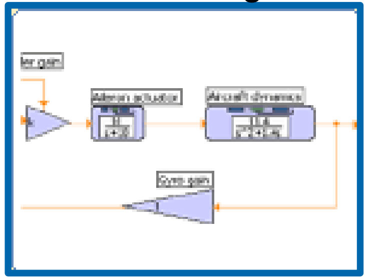
```
#include "sio\pmd565\Shared\Shared.H"

{
unsigned short *AccessArea;
unsigned short RegisterMask;

static unsigned long int RegOffset;
static int Initialized=0;

if(!Initialized){
Initialized=1;

/*GET IMMR */
GetRegOffset(RegOffset);
}

/*WRITE DDR */
RegisterMask=1<<CHANNEL;
AccessArea=0x306102+RegOffset;
*AccessArea=(*AccessArea)&(~RegisterMask); /*Set the right bit to zero*/

/*READ DDI*/
AccessArea=0x306100+RegOffset; /*Skip to data register */
RESULT=(*AccessArea)&RegisterMask;
}
```

Textual Math

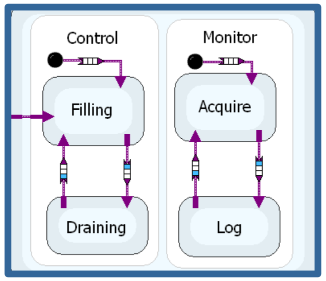


```
c = 0.285 + 0.013i;
[X Y] = meshgrid(x, y);
z = X + i*Y;
for k=1:30
    z = z.^2 + c;
end
```

Modeling



Statechart



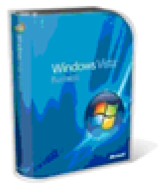
NATIONAL INSTRUMENTS LabVIEW™

Graphical System Design Platform

Linux®

Macintosh

Windows



Desktop Platform

Real-Time

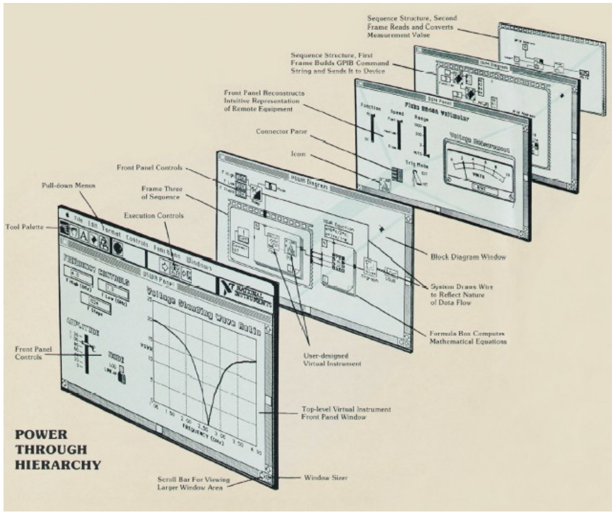
FPGA

Micro

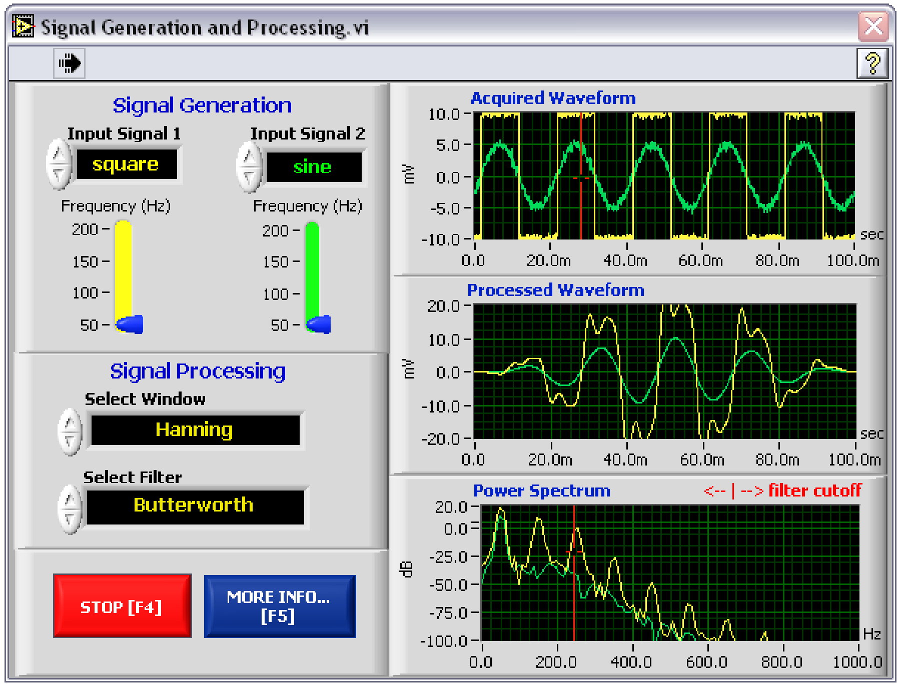
Embedded Platform

NATIONAL INSTRUMENTS™

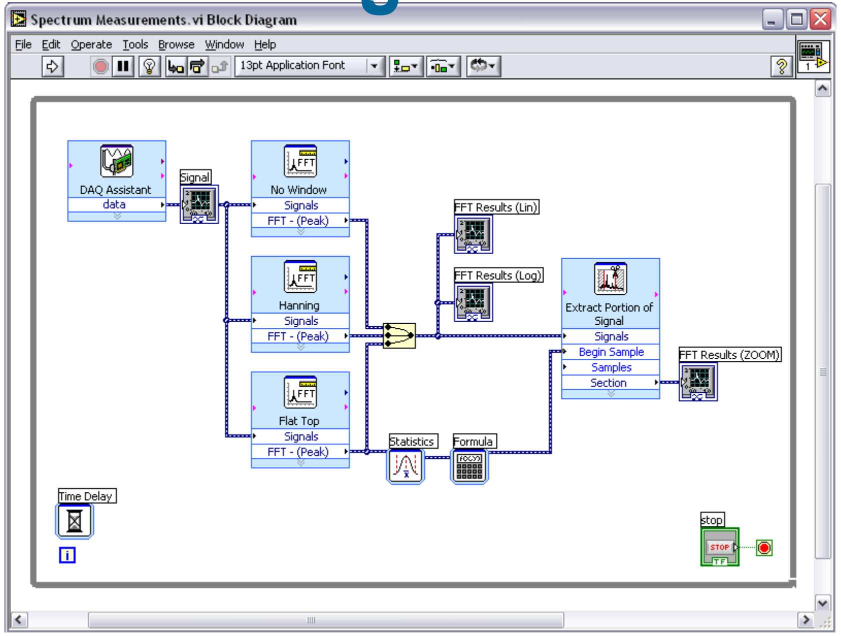
LabVIEW Virtual Instrument


Sequence Structure, Second Frame Reads and Converts Measurement Value
Sequence Structure, First Frame Builds GPIB Command String and Sends It to Device
Front Panel Reconstructs Intuitive Representation of Remote Equipment
Connector Pane
Icon
Front Panel Controls
Pull-down Menus
Frame Three of Sequence
Execution Controls
Tool Palette
Block Diagram Window
System Draws Wire to Reflect Nature of Data Flow
Formula Box Computes Mathematical Equations
Front Panel Controls
User-designed Virtual Instrument
Top-level Virtual Instrument Front Panel Window
POWER THROUGH HIERARCHY
Scroll Bar For Viewing Larger Window Area
Window Sizer

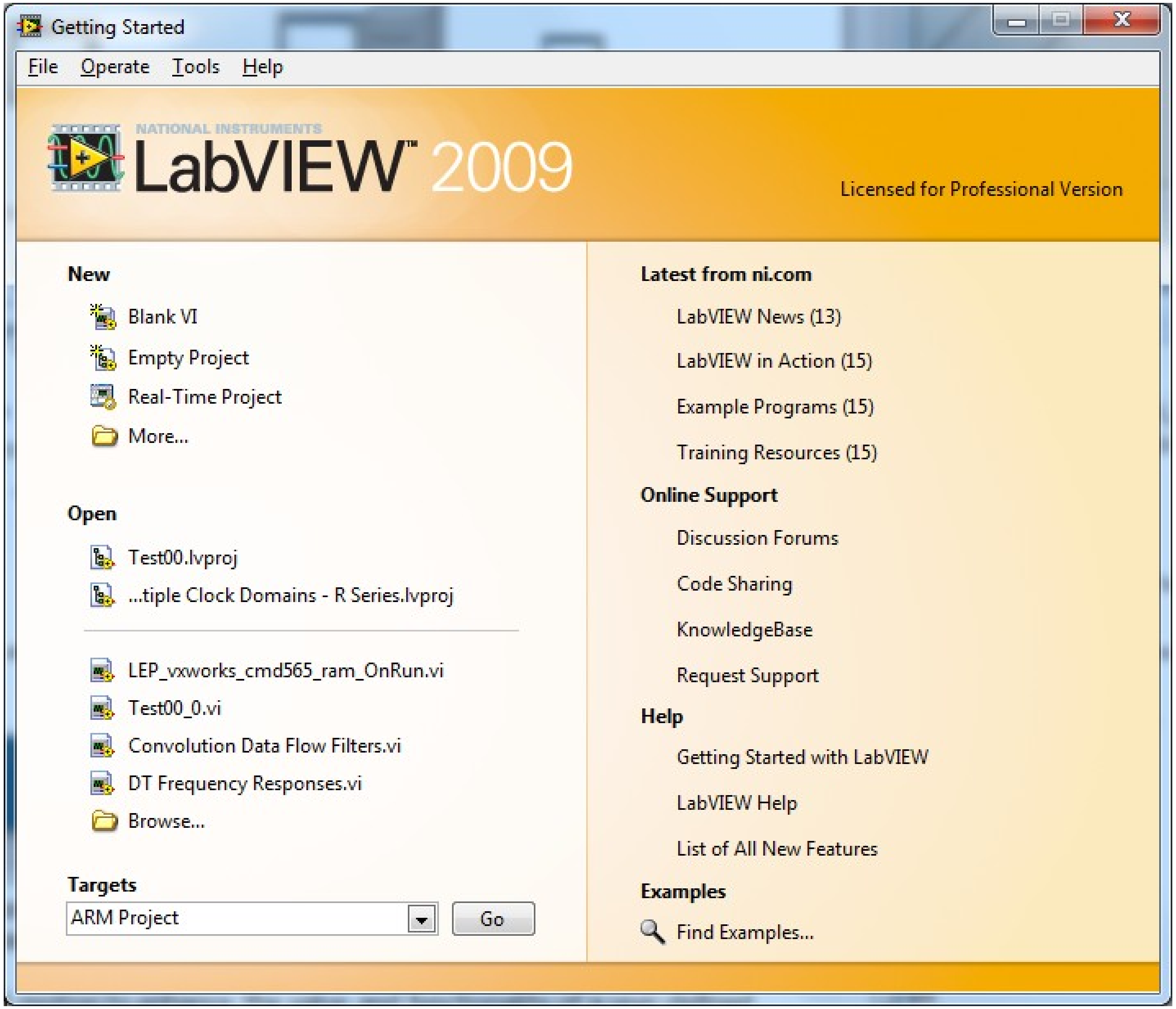
Front Panel



Block Diagram



ni.com


Getting Started

File Operate Tools Help

NATIONAL INSTRUMENTS LabVIEW™ 2009

Licensed for Professional Version

New

- Blank VI
- Empty Project
- Real-Time Project
- More...

Open

- Test00.lvproj
- ...tiple Clock Domains - R Series.lvproj
- LEP_vxworks_cmd565_ram_OnRun.vi
- Test00_0.vi
- Convolution Data Flow Filters.vi
- DT Frequency Responses.vi
- Browse...

Targets

ARM Project | Go

Latest from ni.com

- LabVIEW News (13)
- LabVIEW in Action (15)
- Example Programs (15)
- Training Resources (15)

Online Support

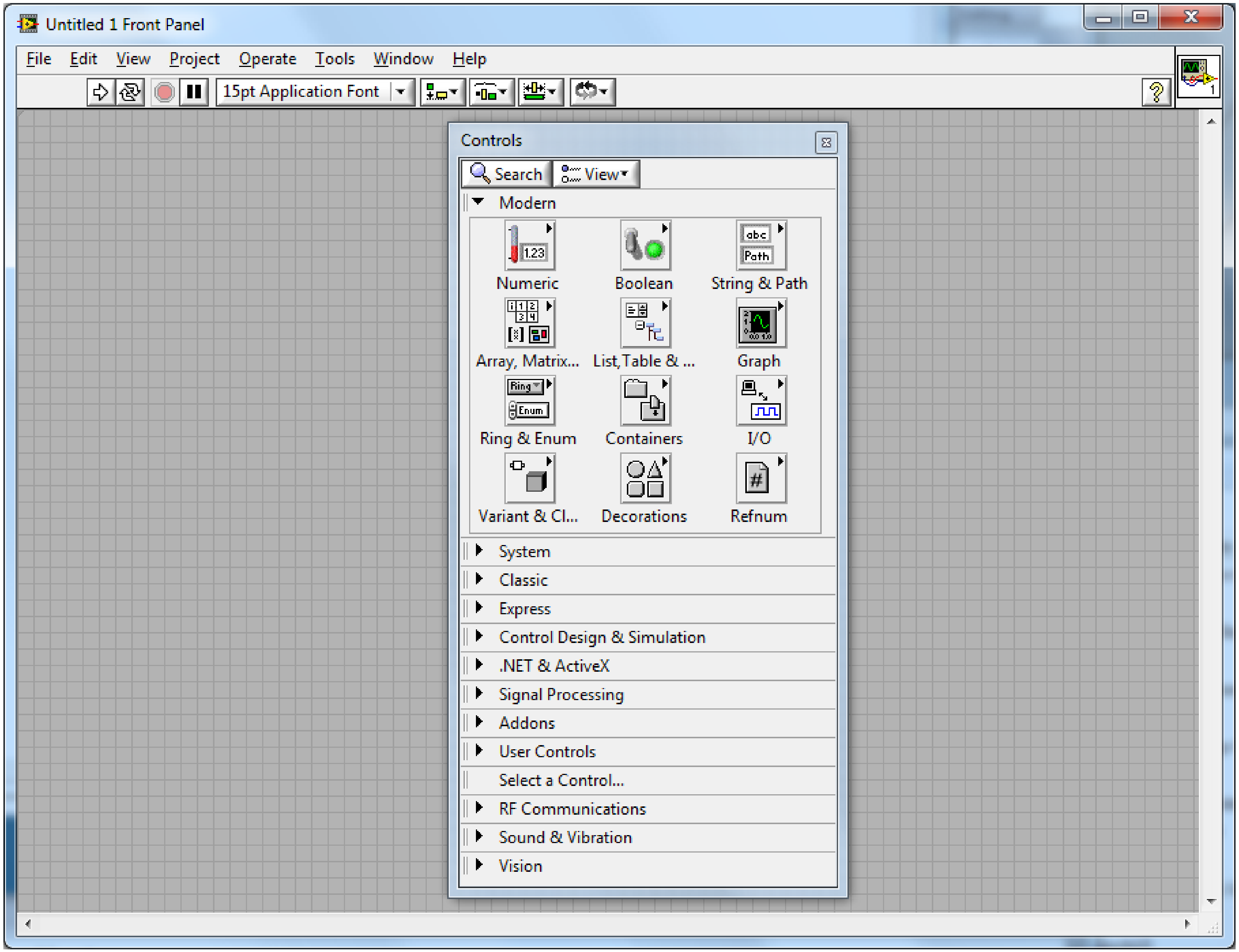
- Discussion Forums
- Code Sharing
- KnowledgeBase
- Request Support

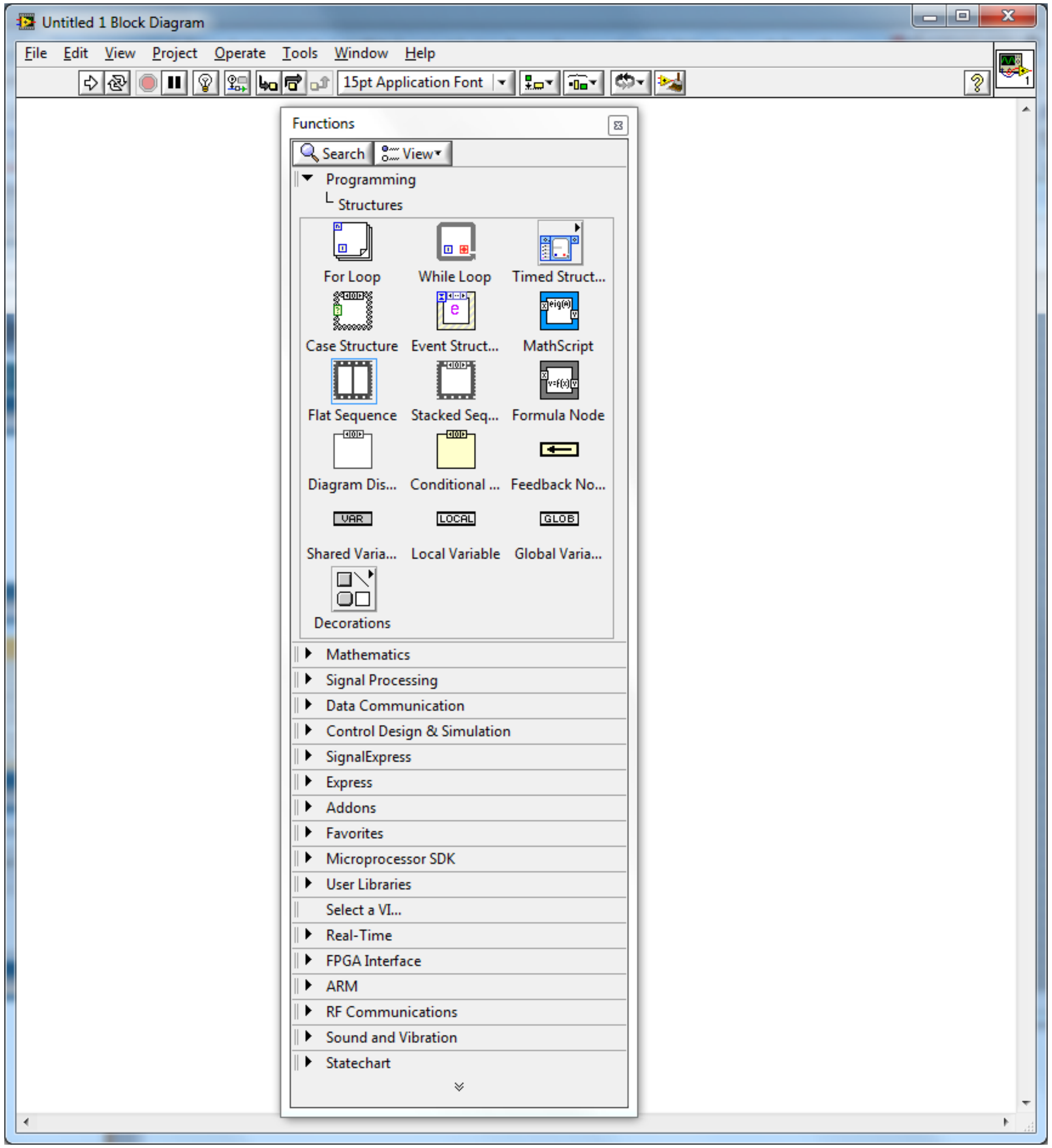
Help

- Getting Started with LabVIEW
- LabVIEW Help
- List of All New Features

Examples

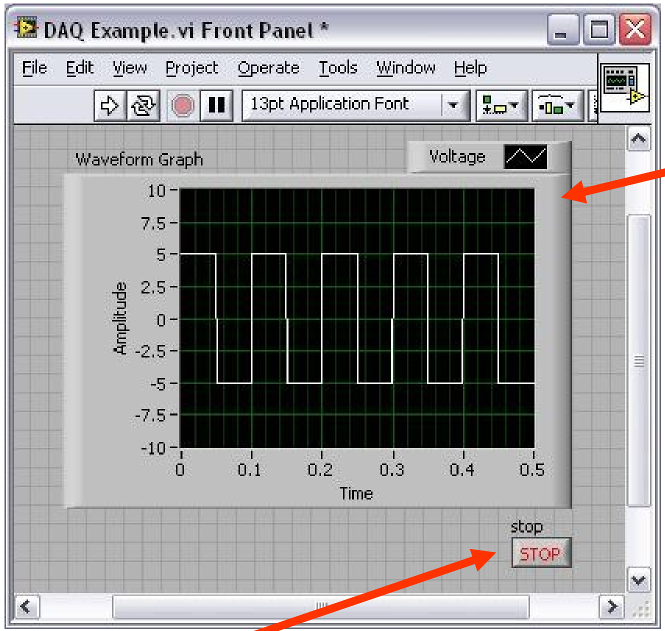
- Find Examples...


Untitled 1 Front Panel
File
Edit
View
Project
Operate
Tools
Window
Help
15pt Application Font
Controls
Search
View
Modern
Numeric
Boolean
String & Path
Array, Matrix...
List, Table & ...
Graph
Ring & Enum
Containers
I/O
Variant & Cl...
Decorations
Refnum
System
Classic
Express
Control Design & Simulation
.NET & ActiveX
Signal Processing
Addons
User Controls
Select a Control...
RF Communications
Sound & Vibration
Vision


Untitled 1 Block Diagram
File
Edit
View
Project
Operate
Tools
Window
Help
15pt Application Font
Functions
Search
View
Programming
Structures
For Loop
While Loop
Timed Struct...
Case Structure
Event Struct...
MathScript
Flat Sequence
Stacked Seq...
Formula Node
Diagram Dis...
Conditional ...
Feedback No...
Shared Varia...
Local Variable
Global Varia...
Decorations
Mathematics
Signal Processing
Data Communication
Control Design & Simulation
SignalExpress
Express
Addons
Favorites
Microprocessor SDK
User Libraries
Select a VI...
Real-Time
FPGA Interface
ARM
RF Communications
Sound and Vibration
Statechart

Creating a VI

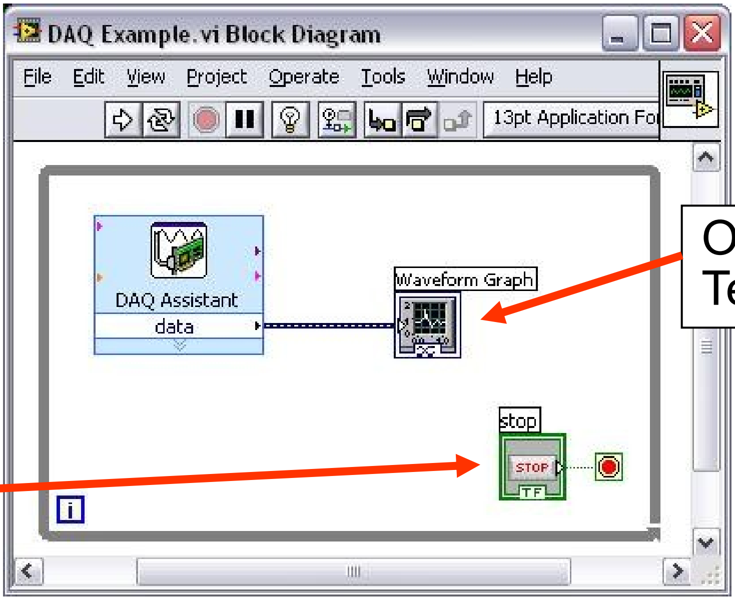
Front Panel Window


DAQ Example.vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font
Waveform Graph
Voltage
Amplitude
Time
stop
STOP

Graph Indicator

Boolean Control

Block Diagram Window

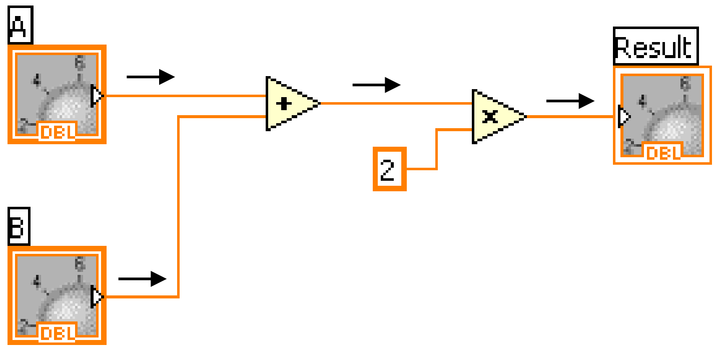

DAQ Example.vi Block Diagram
File Edit View Project Operate Tools Window Help
13pt Application Font
DAQ Assistant
data
Waveform Graph
stop
STOP
TF

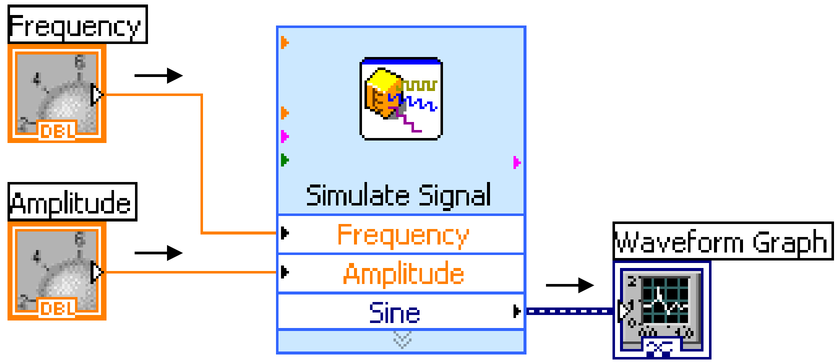
Output Terminal

Input Terminals

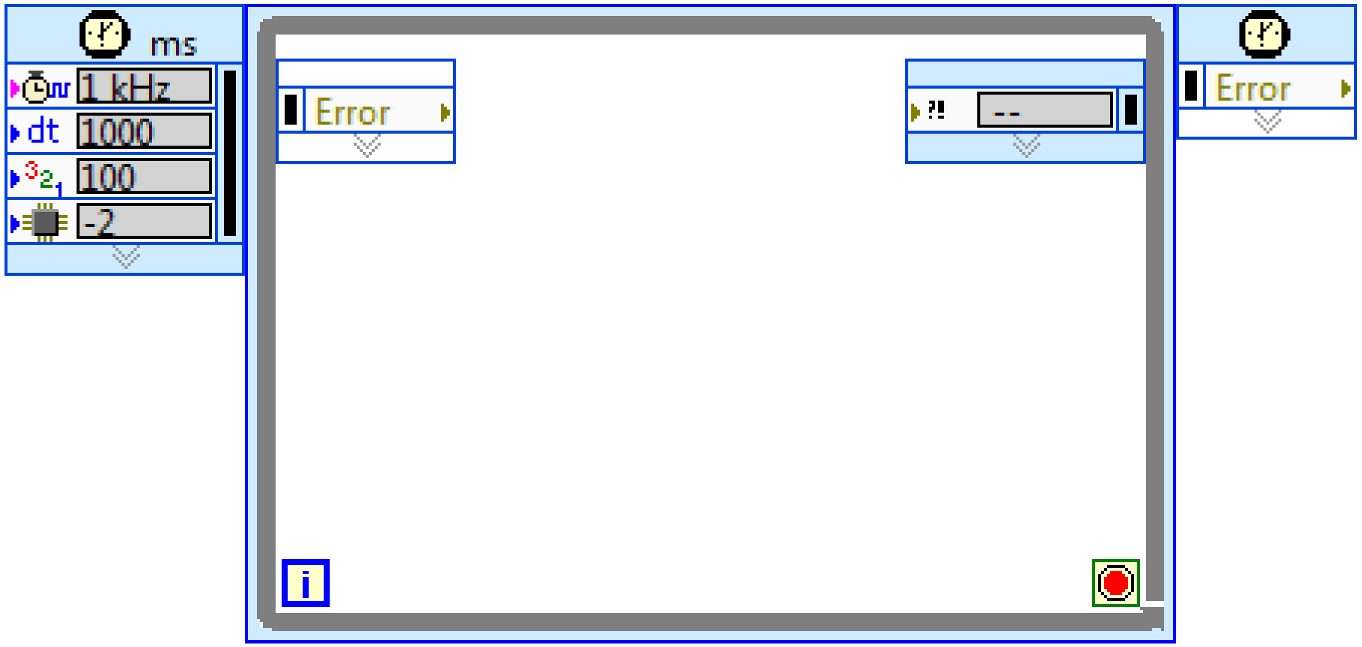
Dataflow Programming

- Block diagram execution
  - Dependent on the flow of data
  - Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done





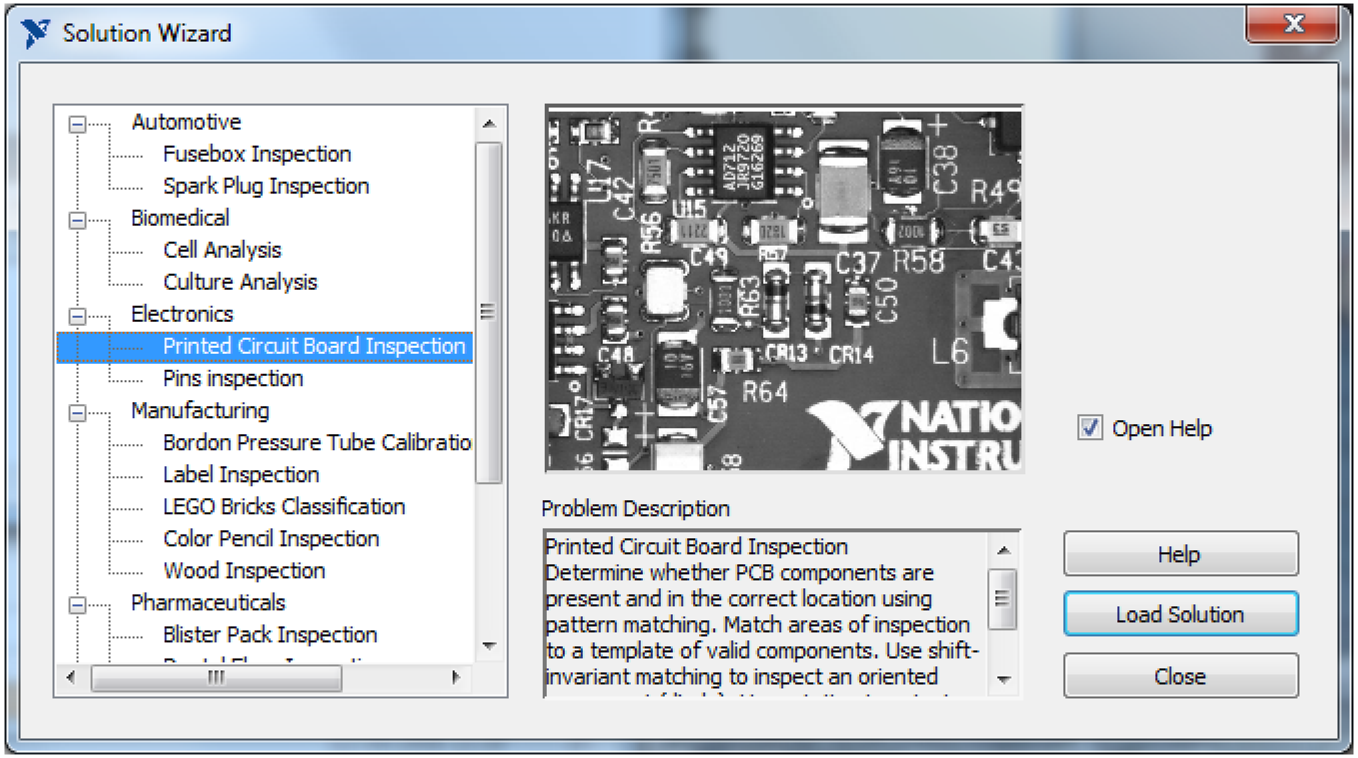
Structured Dataflow



LabVIEW as a Target Language

- Application Wizards – Patterns
- StateCharts
- MathScript
- Control and Simulation Diagram
- Express Nodes and X-nodes
- I/O Nodes

Application Wizards - Patterns



Solution Wizard

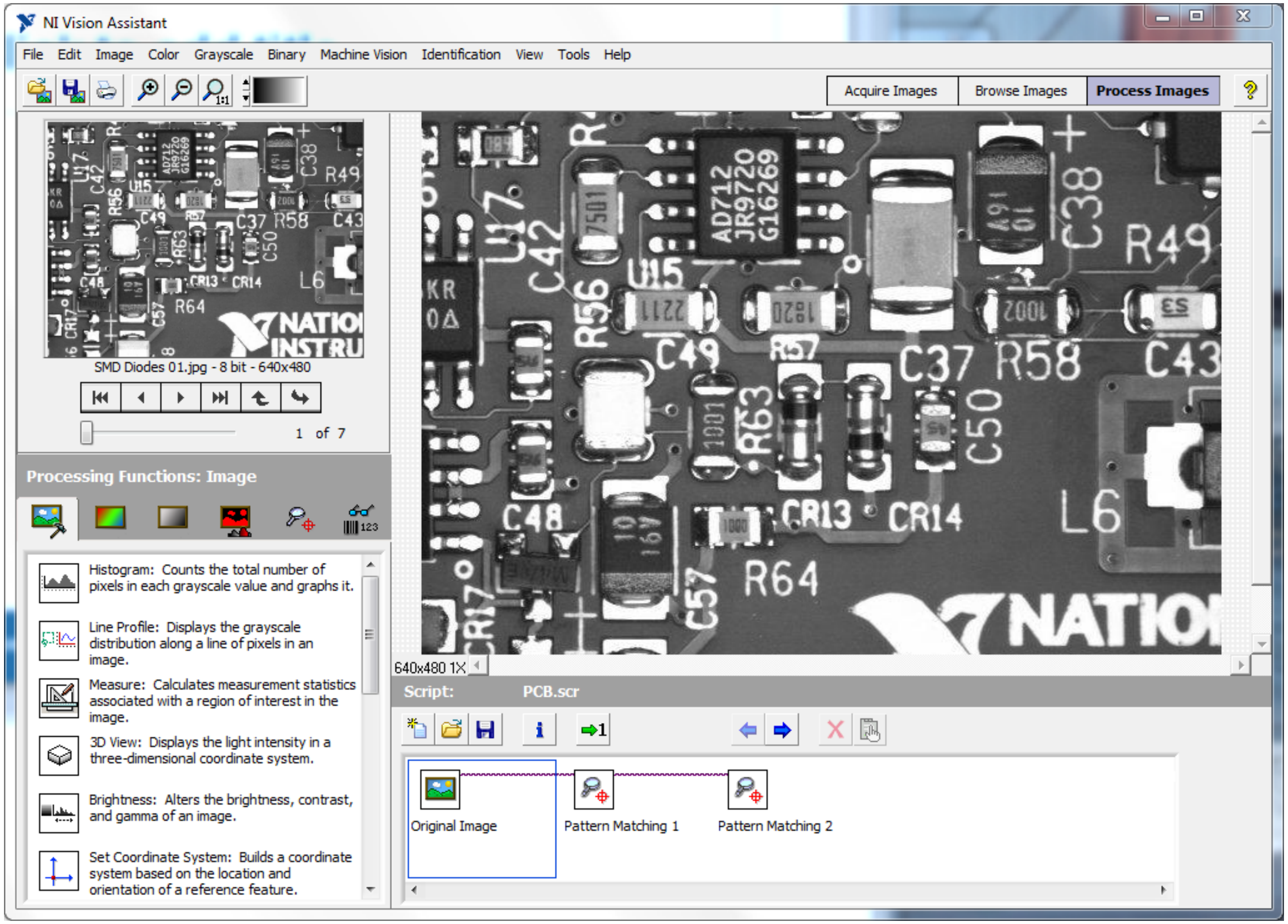
- Automotive
  - Fusebox Inspection
  - Spark Plug Inspection
- Biomedical
  - Cell Analysis
  - Culture Analysis
- Electronics
  - Printed Circuit Board Inspection
  - Pins inspection
- Manufacturing
  - Bordon Pressure Tube Calibratio
  - Label Inspection
  - LEGO Bricks Classification
  - Color Pencil Inspection
  - Wood Inspection
- Pharmaceuticals
  - Blister Pack Inspection

[✓] Open Help

Problem Description

Printed Circuit Board Inspection
Determine whether PCB components are present and in the correct location using pattern matching. Match areas of inspection to a template of valid components. Use shift-invariant matching to inspect an oriented

Application Wizards - Patterns


NI Vision Assistant
File Edit Image Color Grayscale Binary Machine Vision Identification View Tools Help
Acquire Images
Browse Images
Process Images
SMD Diodes 01.jpg - 8 bit - 640x480
1 of 7
Processing Functions: Image
640x480 1X
Script: PCB.scr
Original Image
Pattern Matching 1
Pattern Matching 2

Histogram: Counts the total number of pixels in each grayscale value and graphs it.

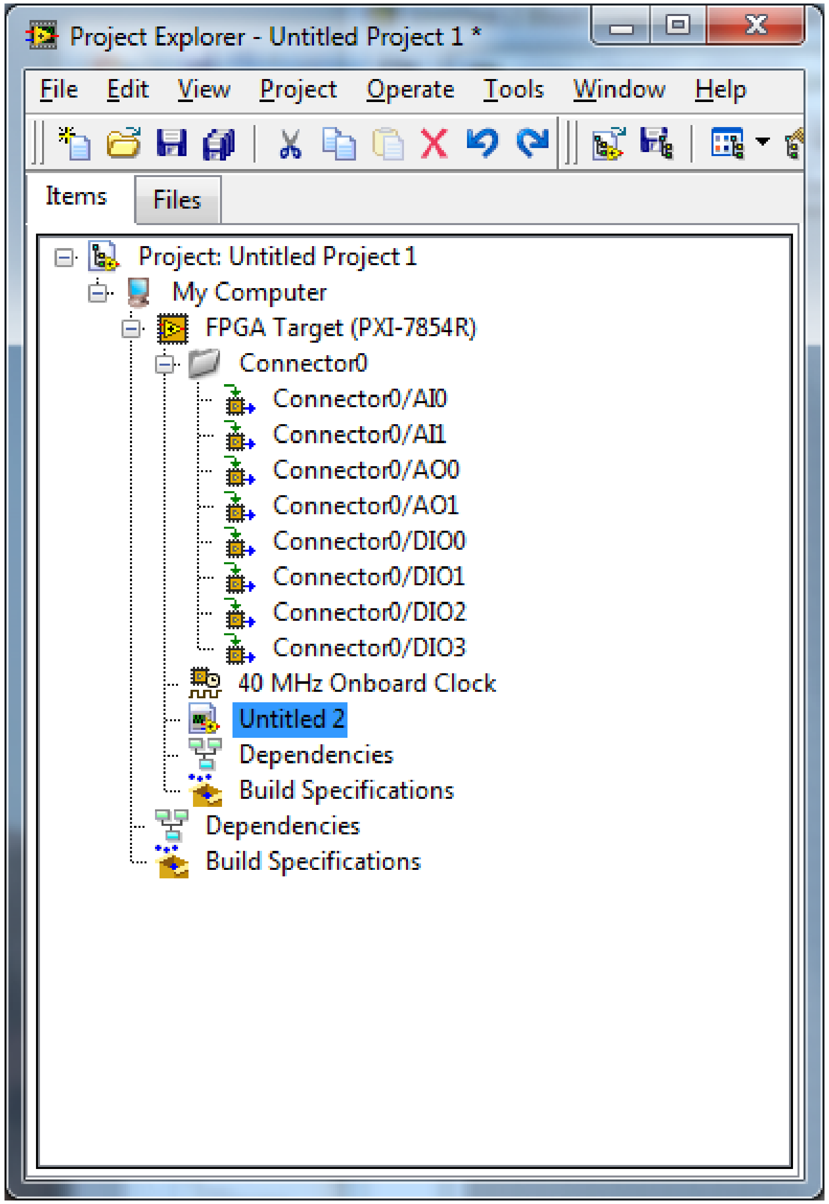
Line Profile: Displays the grayscale distribution along a line of pixels in an image.

Measure: Calculates measurement statistics associated with a region of interest in the image.

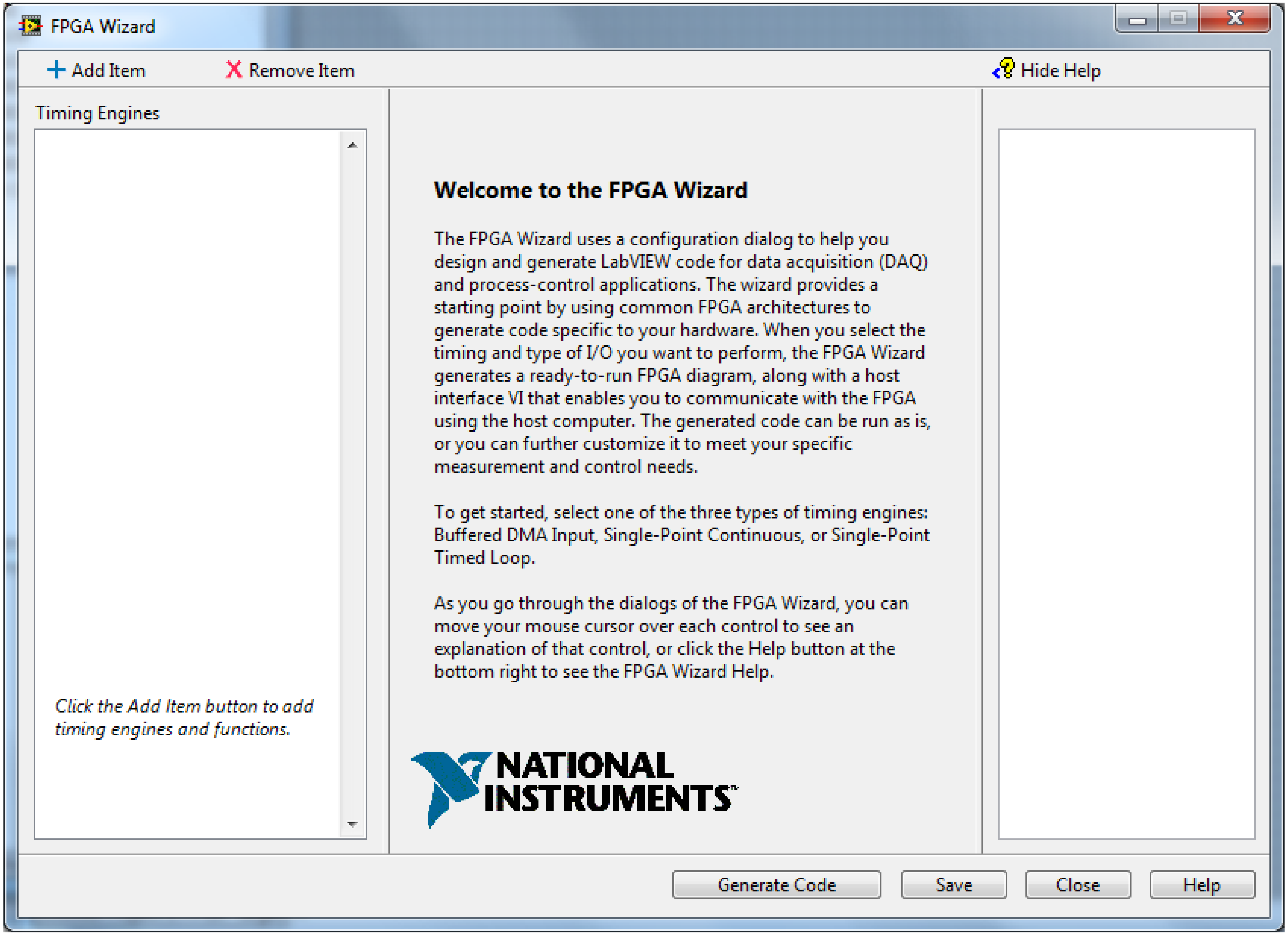
3D View: Displays the light intensity in a three-dimensional coordinate system.

Brightness: Alters the brightness, contrast, and gamma of an image.

Set Coordinate System: Builds a coordinate system based on the location and orientation of a reference feature.



Project: Untitled Project 1
My Computer
FPGA Target (PXI-7854R)
Connector0
Connector0/AI0
Connector0/AI1
Connector0/AO0
Connector0/AO1
Connector0/DIO0
Connector0/DIO1
Connector0/DIO2
Connector0/DIO3
40 MHz Onboard Clock
Untitled 2
Dependencies
Build Specifications
Dependencies
Build Specifications



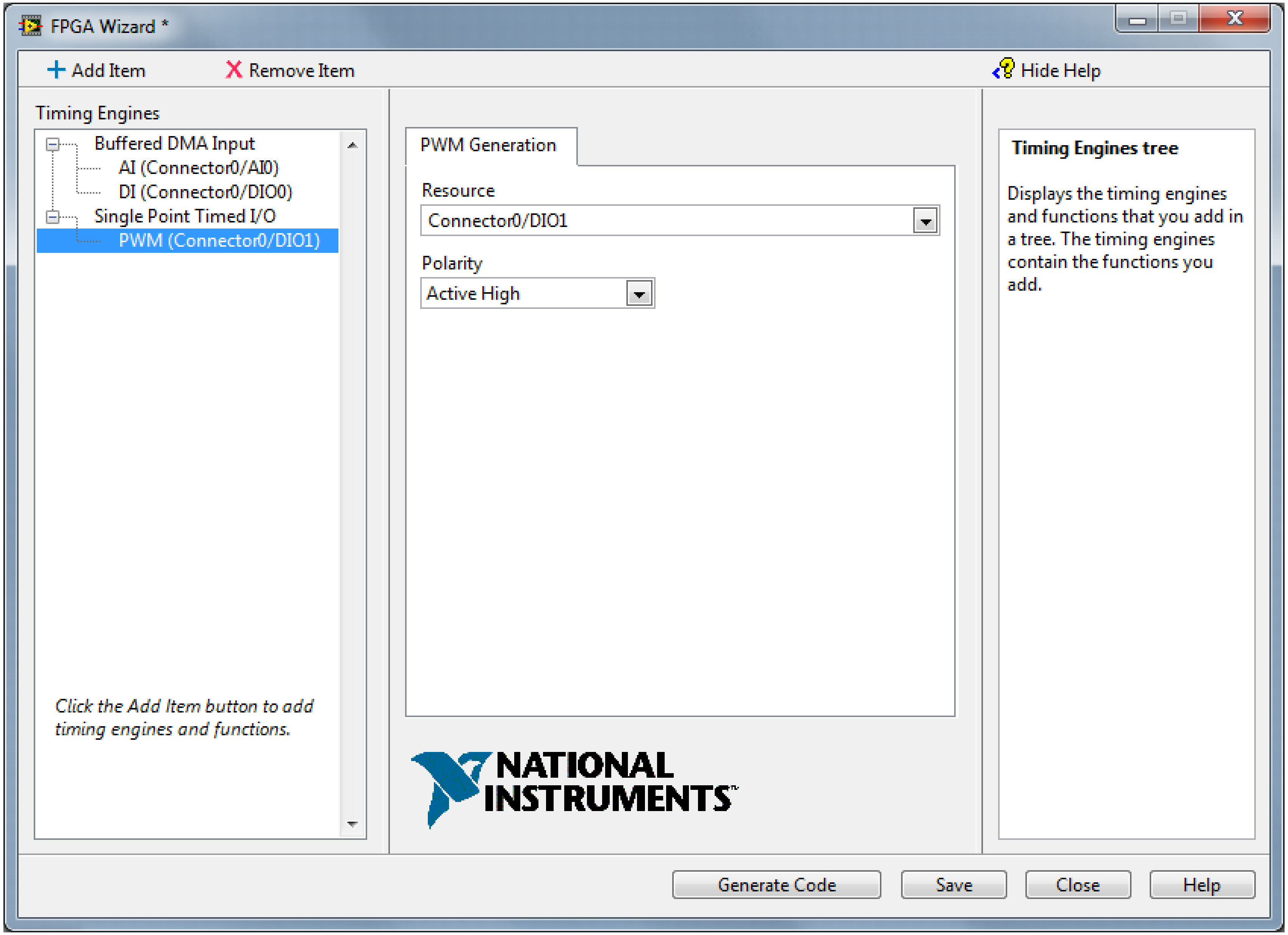
Welcome to the FPGA Wizard

The FPGA Wizard uses a configuration dialog to help you design and generate LabVIEW code for data acquisition (DAQ) and process-control applications. The wizard provides a starting point by using common FPGA architectures to generate code specific to your hardware. When you select the timing and type of I/O you want to perform, the FPGA Wizard generates a ready-to-run FPGA diagram, along with a host interface VI that enables you to communicate with the FPGA using the host computer. The generated code can be run as is, or you can further customize it to meet your specific measurement and control needs.

To get started, select one of the three types of timing engines: Buffered DMA Input, Single-Point Continuous, or Single-Point Timed Loop.

As you go through the dialogs of the FPGA Wizard, you can move your mouse cursor over each control to see an explanation of that control, or click the Help button at the bottom right to see the FPGA Wizard Help.

Click the Add Item button to add timing engines and functions.



FPGA Wizard *

Add Item   Remove Item   Hide Help

Timing Engines

- Buffered DMA Input
  - AI (Connector0/AI0)
  - DI (Connector0/DIO0)
- Single Point Timed I/O
  - PWM (Connector0/DIO1)

Click the Add Item button to add timing engines and functions.

PWM Generation

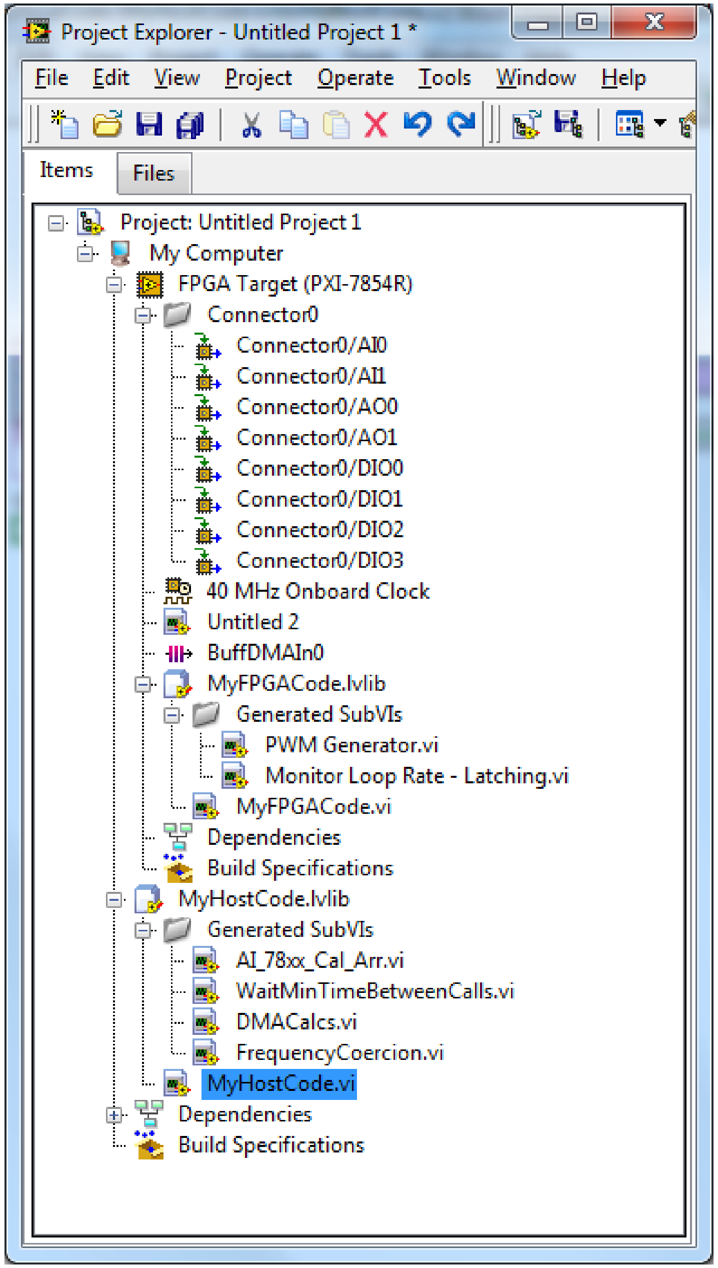
Resource: Connector0/DIO1

Polarity: Active High

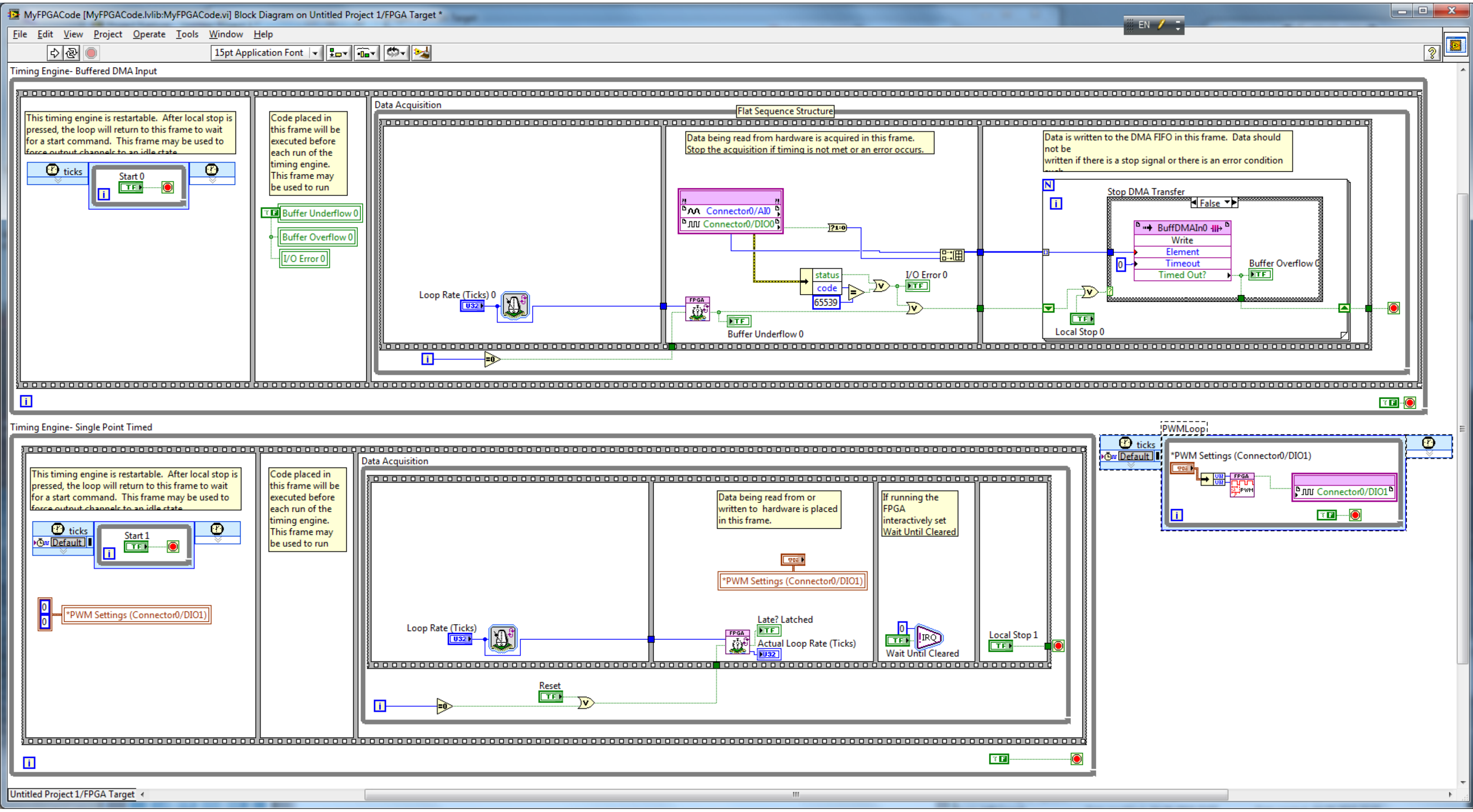
Timing Engines tree

Displays the timing engines and functions that you add in a tree. The timing engines contain the functions you add.

Generate Code   Save   Close   Help


Project Explorer - Untitled Project 1 *
File Edit View Project Operate Tools Window Help
Items Files

Project: Untitled Project 1
My Computer
FPGA Target (PXI-7854R)
Connector0
Connector0/AI0
Connector0/AI1
Connector0/AO0
Connector0/AO1
Connector0/DIO0
Connector0/DIO1
Connector0/DIO2
Connector0/DIO3
40 MHz Onboard Clock
Untitled 2
BuffDMAIn0
MyFPGACode.lvlib
Generated SubVIs
PWM Generator.vi
Monitor Loop Rate - Latching.vi
MyFPGACode.vi
Dependencies
Build Specifications
MyHostCode.lvlib
Generated SubVIs
AI_78xx_Cal_Arr.vi
WaitMinTimeBetweenCalls.vi
DMACalcs.vi
FrequencyCoercion.vi
MyHostCode.vi
Dependencies
Build Specifications


MyFPGACode [MyFPGACode.lvlib:MyFPGACode.vi] Block Diagram on Untitled Project 1/FPGA Target *

Timing Engine- Buffered DMA Input

This timing engine is restartable. After local stop is pressed, the loop will return to this frame to wait for a start command. This frame may be used to force output channels to an idle state

Code placed in this frame will be executed before each run of the timing engine. This frame may be used to run

Data being read from hardware is acquired in this frame. Stop the acquisition if timing is not met or an error occurs.

Data is written to the DMA FIFO in this frame. Data should not be written if there is a stop signal or there is an error condition such

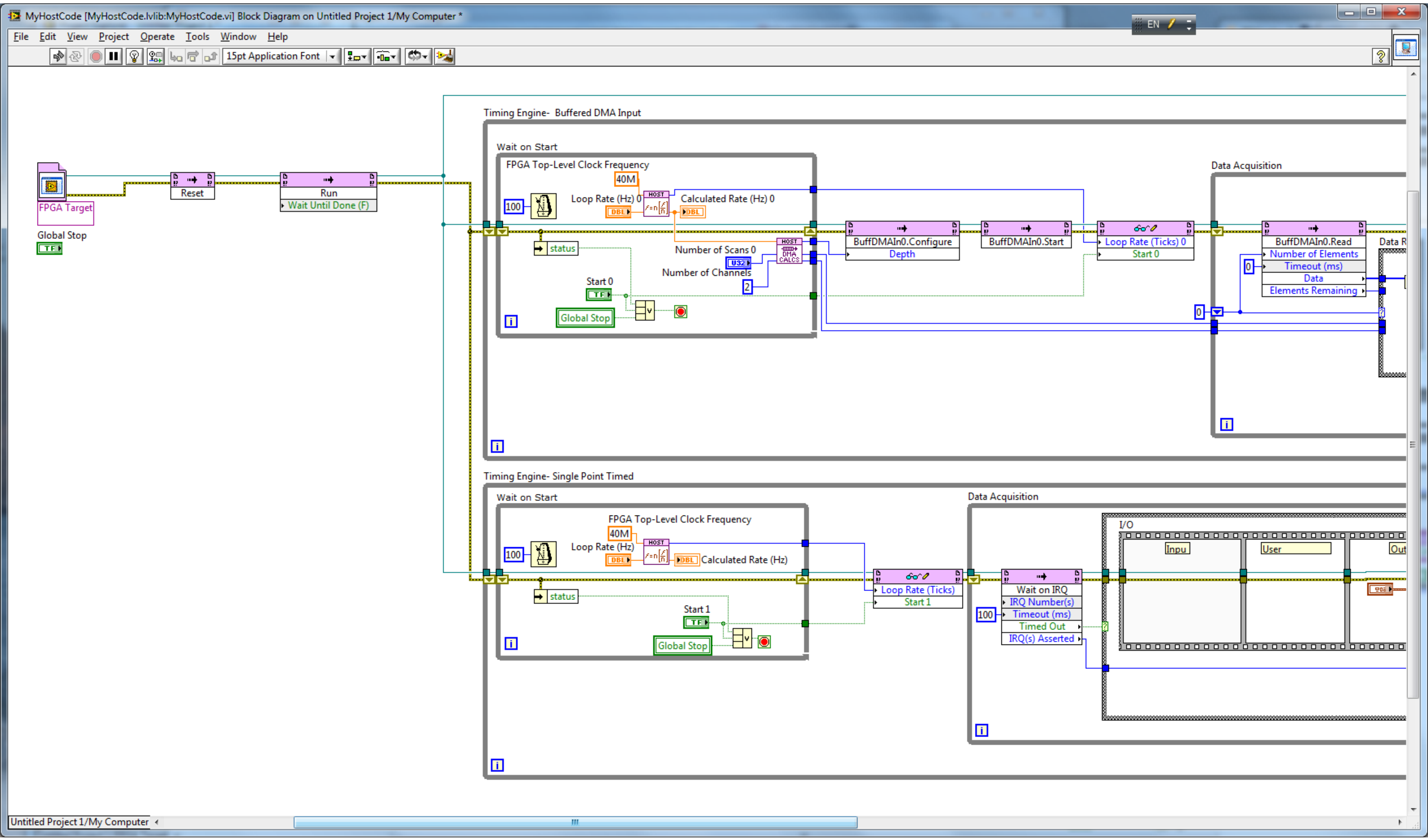
Timing Engine- Single Point Timed

This timing engine is restartable. After local stop is pressed, the loop will return to this frame to wait for a start command. This frame may be used to force output channels to an idle state

Code placed in this frame will be executed before each run of the timing engine. This frame may be used to run

Data being read from or written to hardware is placed in this frame.

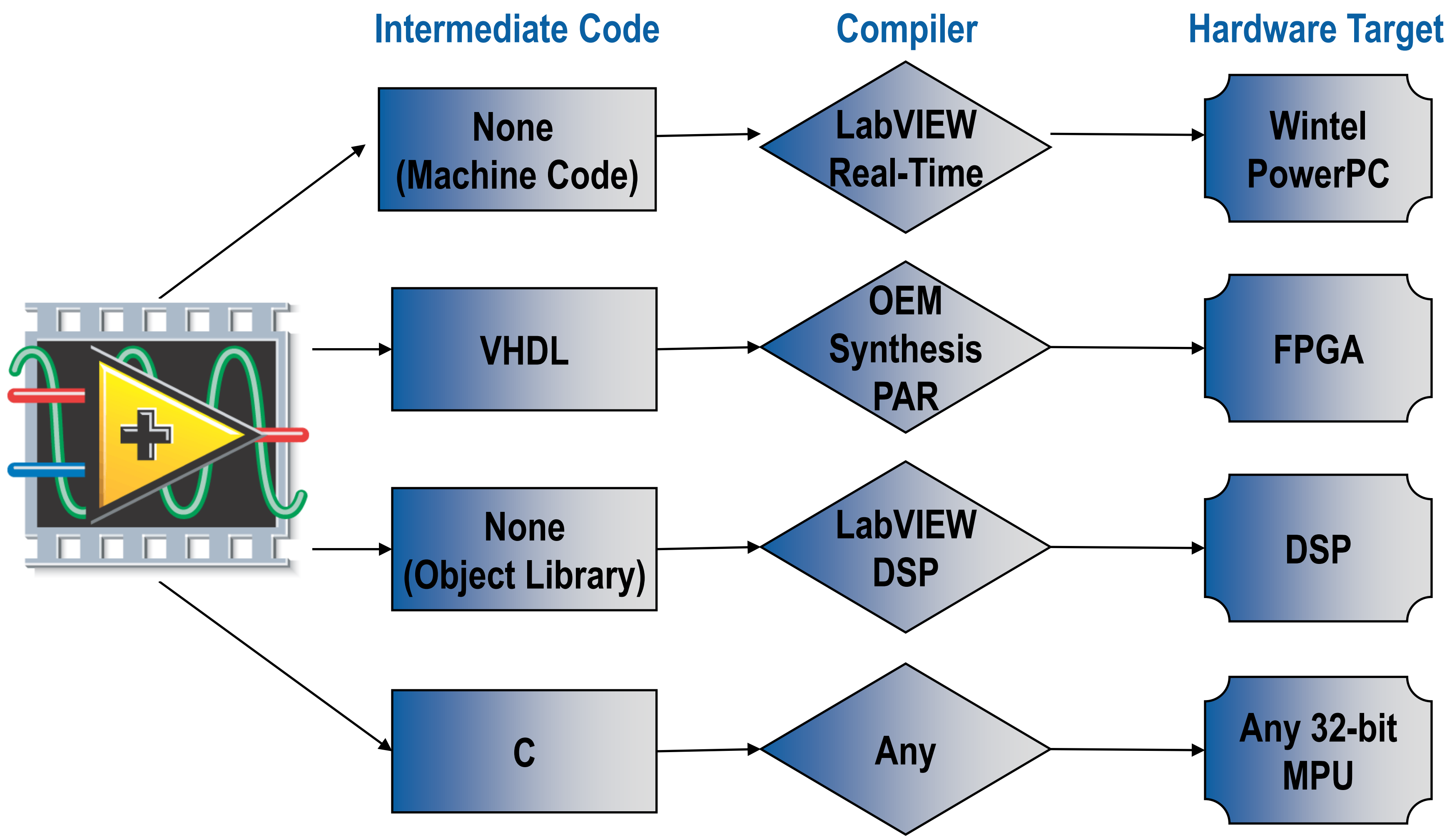
If running the FPGA interactively set Wait Until Cleared


MyHostCode [MyHostCode.lvlib:MyHostCode.vi] Block Diagram on Untitled Project 1/My Computer *
File Edit View Project Operate Tools Window Help
15pt Application Font
FPGA Target
Global Stop
Reset
Run
Wait Until Done (F)
Timing Engine- Buffered DMA Input
Wait on Start
FPGA Top-Level Clock Frequency
40M
100
Loop Rate (Hz) 0
Calculated Rate (Hz) 0
status
Number of Scans 0
Number of Channels
2
Start 0
Global Stop
BuffDMAIn0.Configure
Depth
BuffDMAIn0.Start
Loop Rate (Ticks) 0
Start 0
Data Acquisition
BuffDMAIn0.Read
Number of Elements
Timeout (ms)
Data
Elements Remaining
Data R
0
Timing Engine- Single Point Timed
Wait on Start
FPGA Top-Level Clock Frequency
40M
100
Loop Rate (Hz)
Calculated Rate (Hz)
status
Start 1
Global Stop
Loop Rate (Ticks)
Start 1
Data Acquisition
Wait on IRQ
IRQ Number(s)
Timeout (ms)
Timed Out
IRQ(s) Asserted
100
I/O
Inpu
User
Out
Untitled Project 1/My Computer

The G (LabVIEW) Language Model

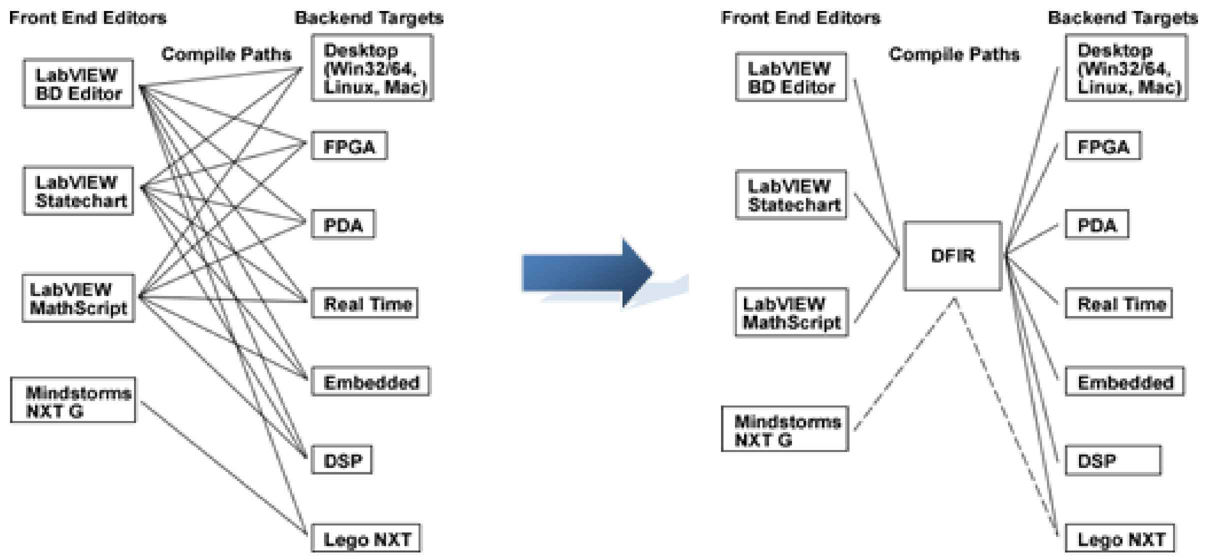
- Homogenous dataflow language
  - Structured case (switch, select) and loops
    - “Structured dataflow”
- Run-time scheduling
  - Explicit task level parallelism
  - Implicit parallelism heuristically identified
- Synthesizable language
  - To machine code on x86 and PPC processors
  - To VHDL for FPGAs
  - To C for embedded processors
- Turing complete

Evolution of LabVIEW Code Generation



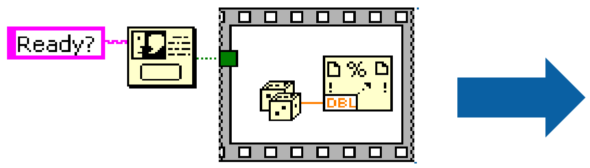
DFIR - Background

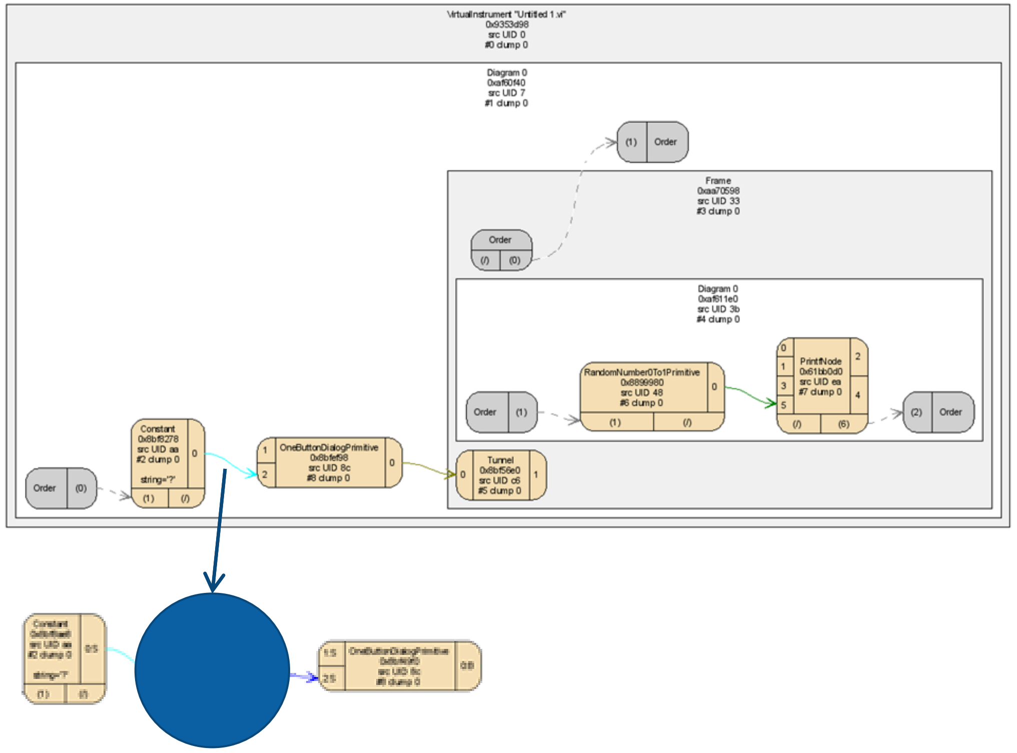
Data **F**low **I**ntermediate **R**epresentation (**DFIR**) is used today to separate front-end editors from back-end compilers (as illustrated below) and to provide a consistent framework for managing code generation and optimizations.



DFIR - Background

DFIR models existing G data flow semantics with arbitrary VI hierarchy. Wires are also modeled as nodes, which can generate custom code if needed.





System Deployment

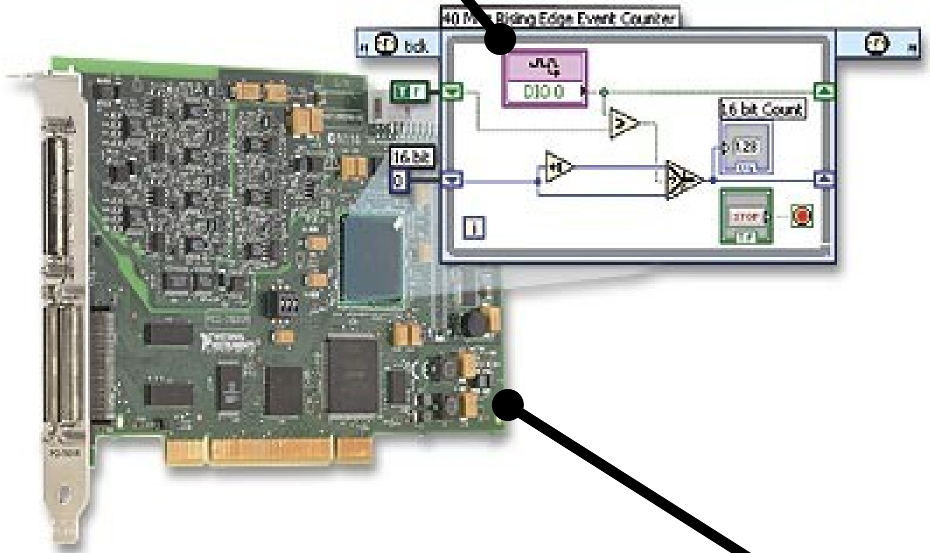
- Target aware synthesis
- I/O Port Abstraction
  - I/O Classes
  - Protocol generation
- Channel Abstraction
  - FIFO
  - Loop-to-loop
  - Peer-to-peer
  - Board-to-host (DMA)

System Deployment

- Timing
    - Expressing an order
        - Language constructs
        - Operating Environments
    - Reality of Platform timing
        - Static analysis

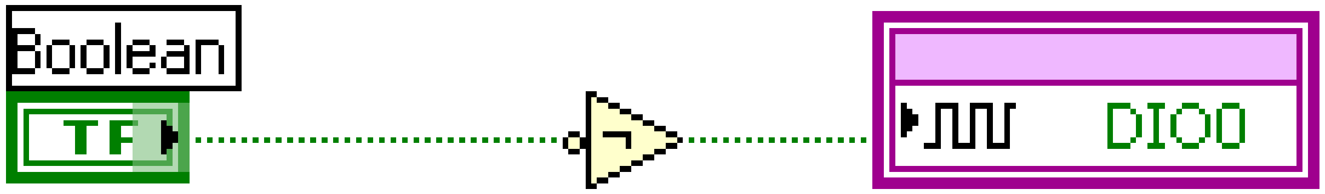
What is LabVIEW FPGA

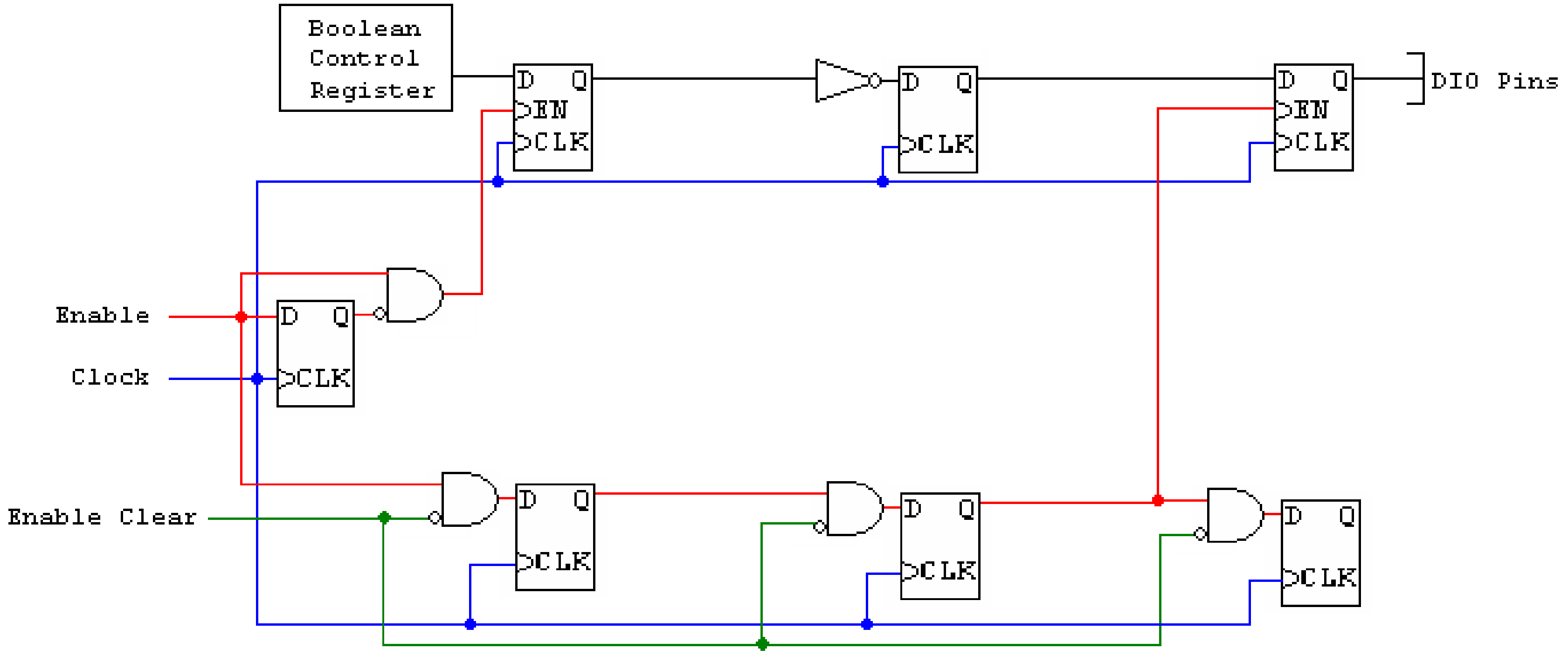
LabVIEW FPGA Module



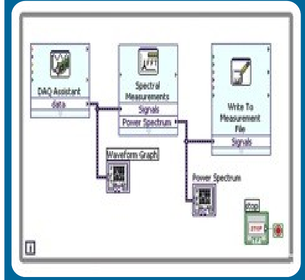
Reconfigurable I/O (RIO) Hardware

Enforcing Dataflow in FPGA

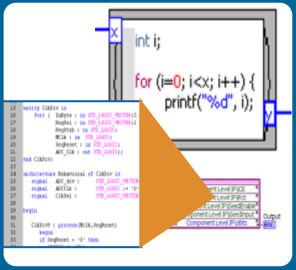




Data Flow

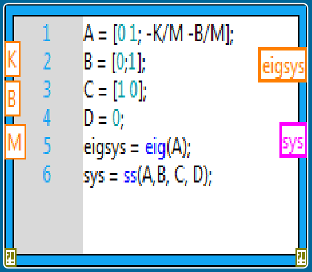


C/HDL Code



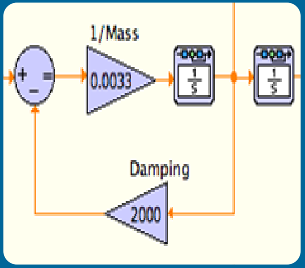
```
int i;
for (i=0; i<x; i++) {
    printf("%d", i);
```

Textual Math

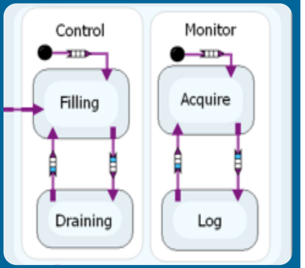


```
A = [0 1; -K/M -B/M];
B = [0;1];
C = [1 0];
D = 0;
eigsys = eig(A);
sys = ss(A,B, C, D);
```

Simulation



State Chart



NATIONAL INSTRUMENTS LabVIEW™

Personal Computers

PXI Systems

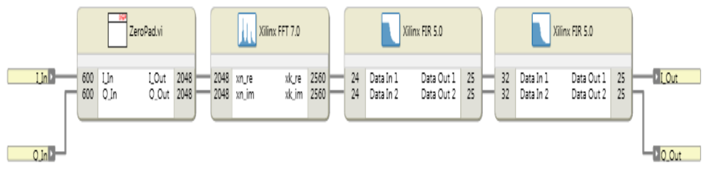
NI CompactRIO

NI Single-Board RIO

NI USRP

System-Level Design

Concurrent Application

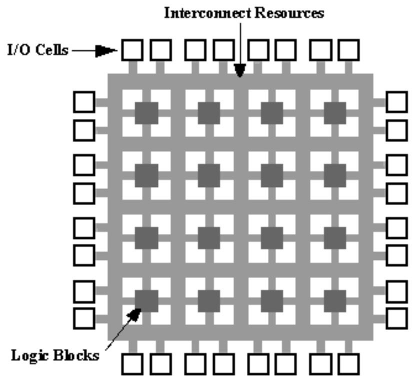


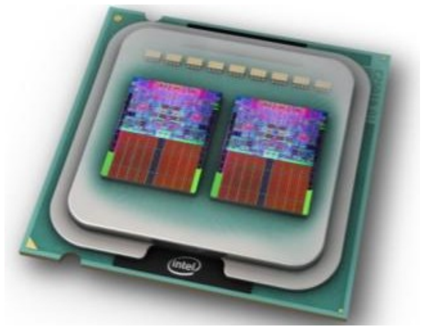
Application trends

- **Large # of parallel tasks**
- Large node/channel counts
- High performance requirements
- E.g. streaming DSP applications

Implementation Gap

Parallel Platform





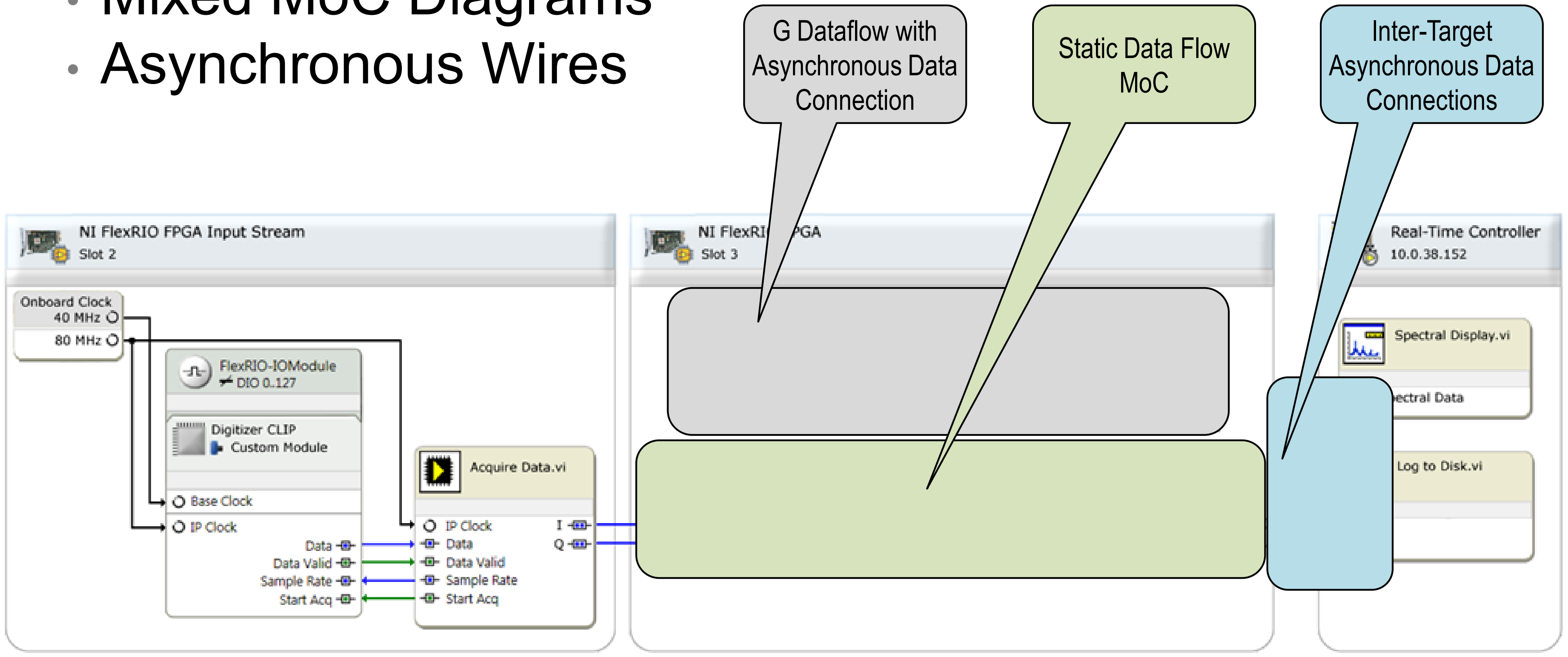
Platform trends

- **Large # of processing elements**
- Heterogeneous processors and memories
- Distributed I/O
- E.g. Heterogeneous FPGA targets

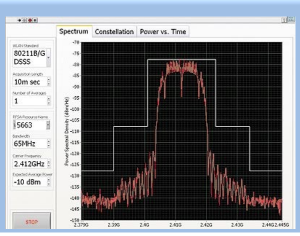
Modeling System-Level Designs

System-level designs introduce new modeling constructs:

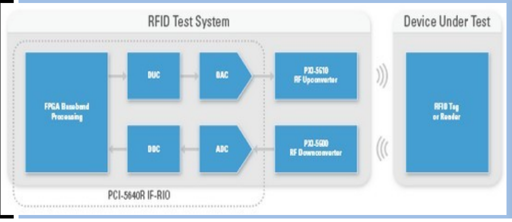
- Systems
- Targets
- Mixed MoC Diagrams
- Asynchronous Wires



RF Communications Applications Overview



WLAN Standard: 80211B/G DSSS
Acquisition Length: 10m sec
Number of Averages: 1
RFSA Resource Name: 5663
Bandwidth: 65MHz
Carrier Frequency: 2.412GHz
Expected Average Power: -10 dBm



Research

Network Optimization
Interference Alignment
Single Ant Diversity
Cognitive Radio

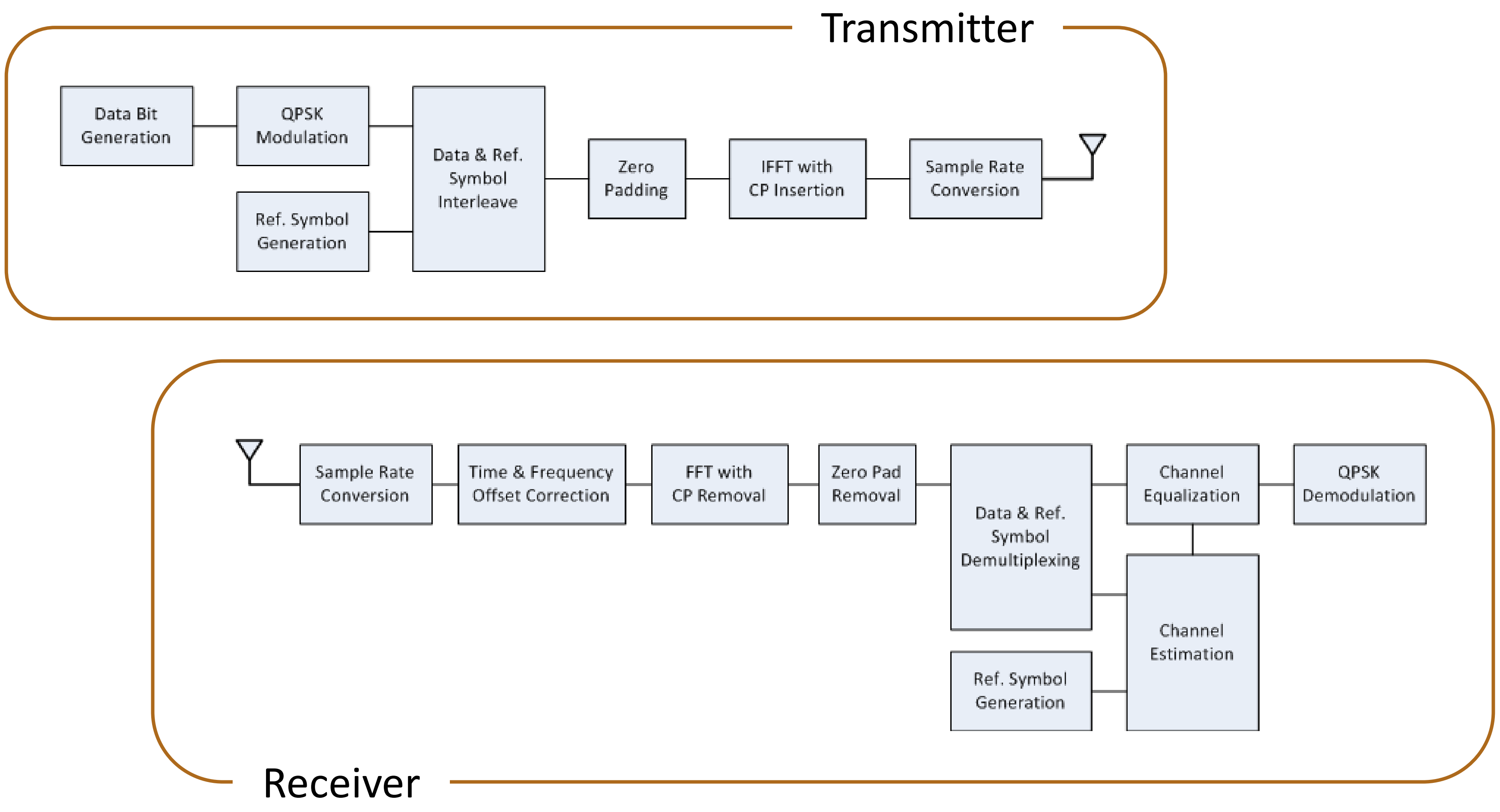
Surveillance

Signal INT
Communications INT
Electronic INT
Spectrum Sniffer

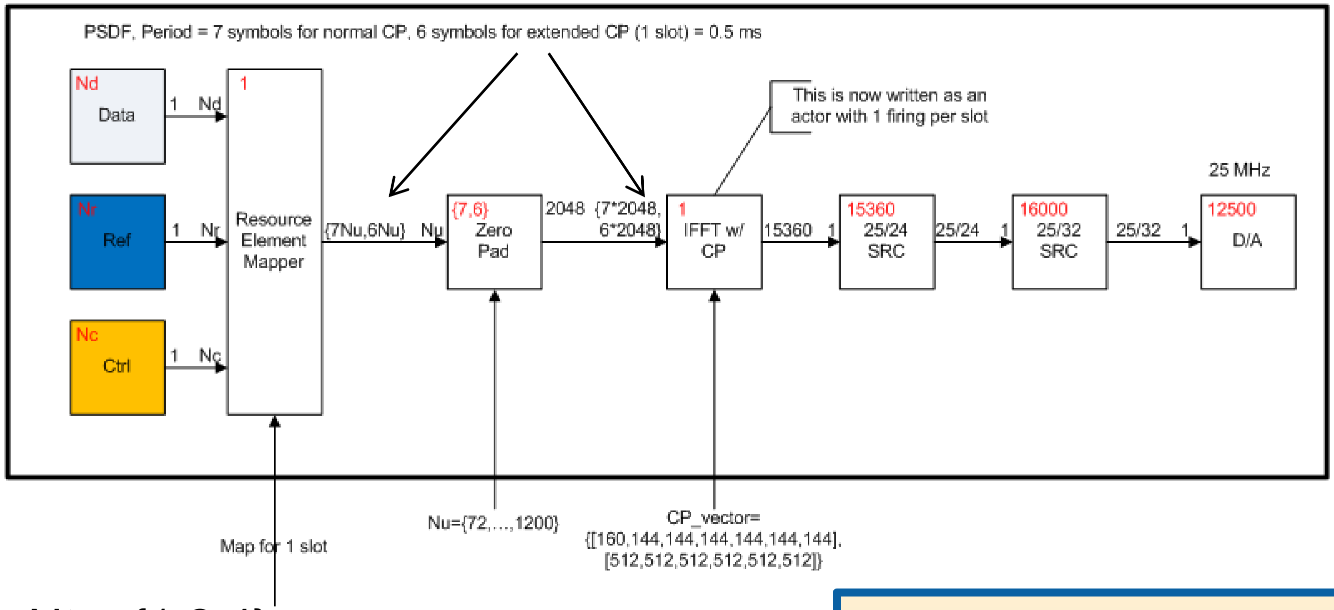
Advanced Test

RF Channel Emulation
BTS Emulation
Protocol Analysis
Next Gen Test

OFDM TX/RX Block Diagram



Streaming Model of the OFDM Transmitter

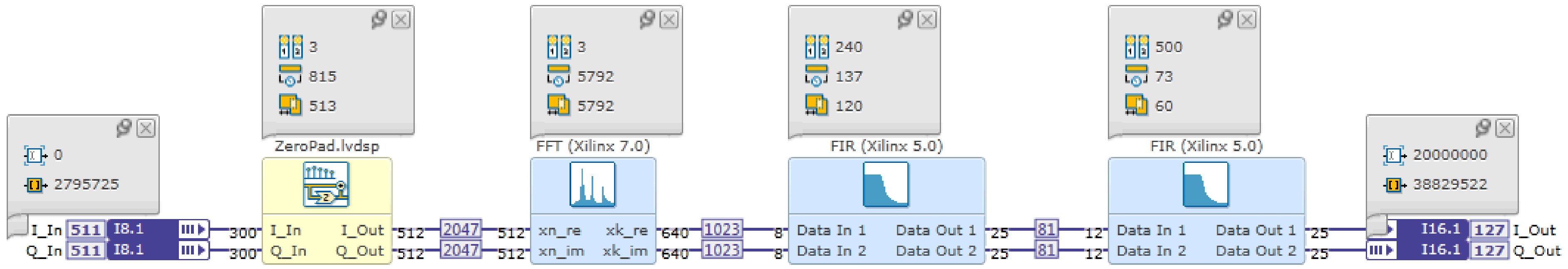


PSDF, Period = 7 symbols for normal CP, 6 symbols for extended CP (1 slot) = 0.5 ms
Nd
Data
1 Nd
Nr
Ref
1 Nr
Nc
Ctrl
1 Nc
1
Resource Element Mapper
{7Nu,6Nu} Nu
{7,6}
Zero Pad
2048 {7*2048, 6*2048}
1
IFFT w/ CP
This is now written as an actor with 1 firing per slot
15360 1
15360
25/24 SRC
25/24 1
16000
25/32 SRC
25/32 1
25 MHz
12500
D/A
Map for 1 slot
Nu={72,...,1200}
CP_vector= {[160,144,144,144,144,144,144], [512,512,512,512,512,512]}

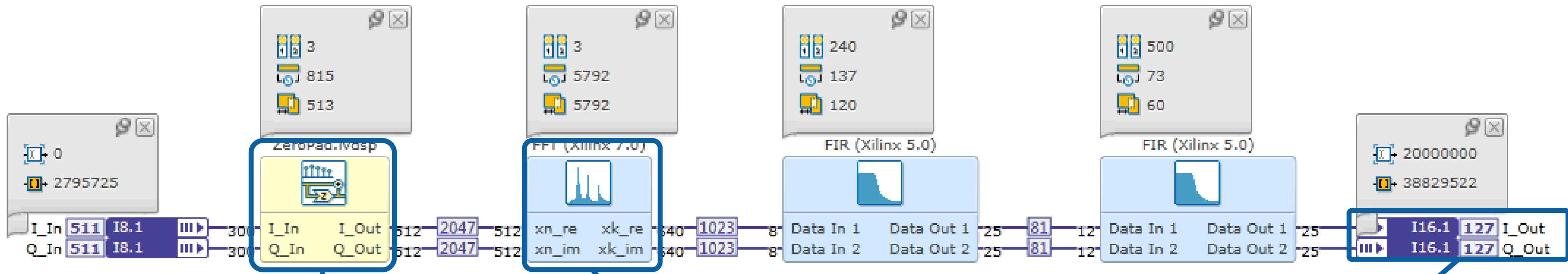
- Nt = {1,2,4}
  - Compile time - # transmitters
- Nu = {72, 180, 300, 600, 900, 1200}
  - Initialization time - Bandwidth
- CP mode = {'Normal', 'Extended'}
  - Run time, To overcome Inter-symbol-interference, Can be applied at symbol boundary
- CP Vector
  - Selection based on CP mode, Elements must be applied at symbol boundary

Challenge: How to express a domain expert's algorithm specification in a model that is viable for analysis and implementation?

LabVIEW DSP Design Module



LabVIEW DSP Design Module

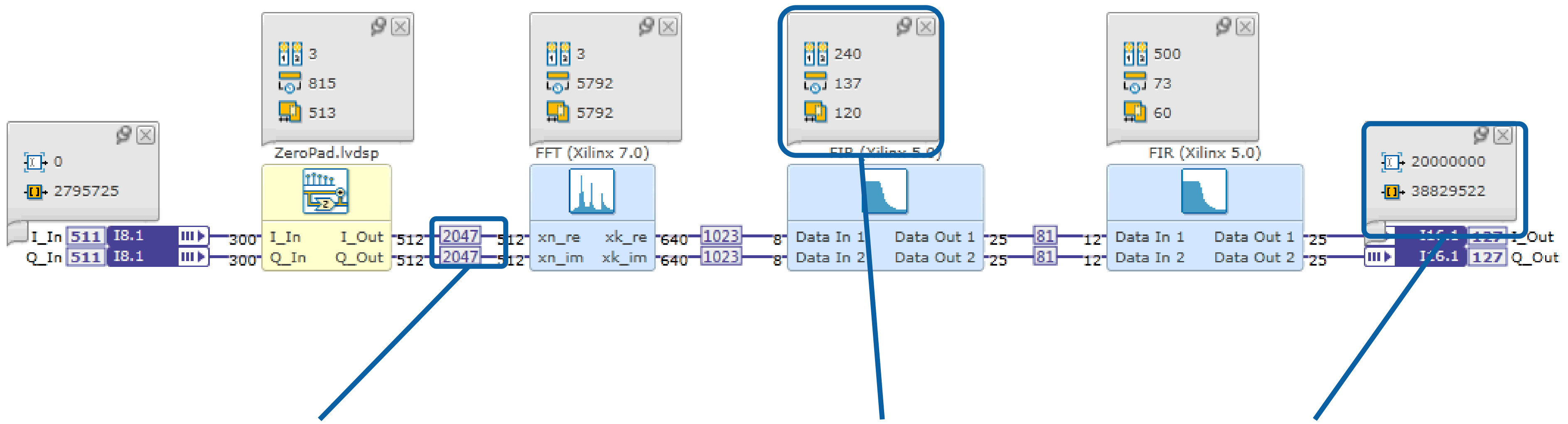


Sub Diagrams

Xilinx CoreGen Blocks

Data Ports

LabVIEW DSP Design Module

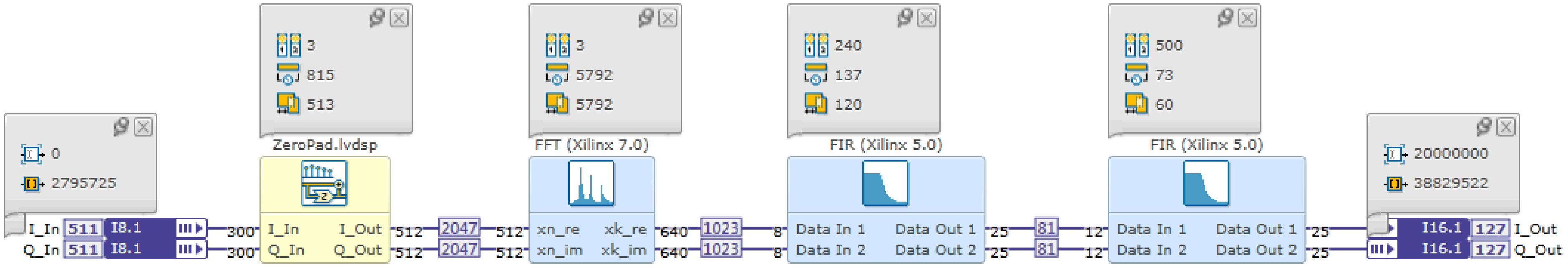


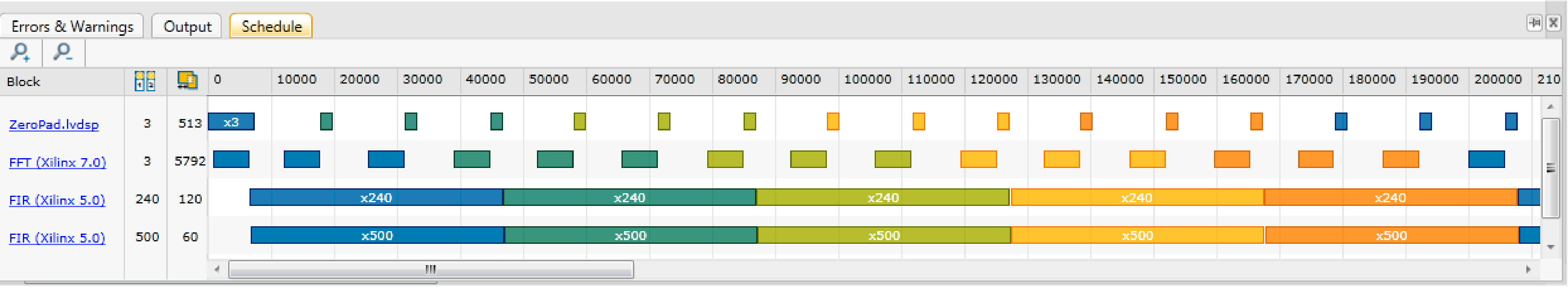
Auto buffer sizing to minimize resources

Calculated firing counts and timing data

Throughput constraints

LabVIEW DSP Design Module

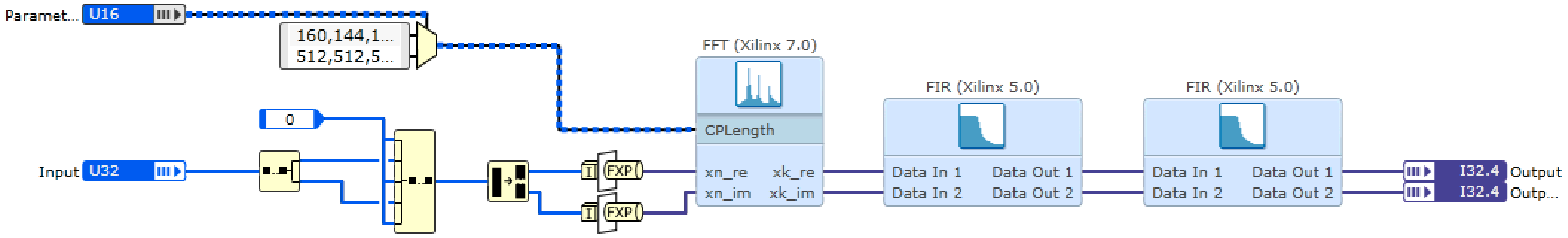


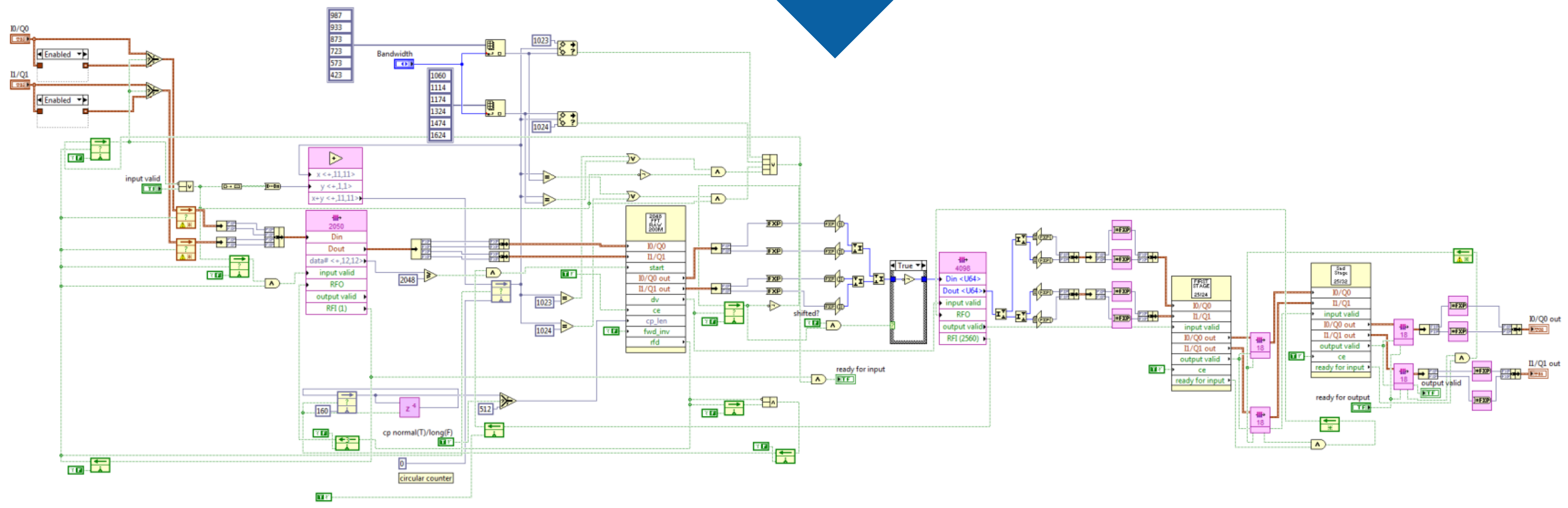


Block		
ZeroPad.lvdsp	3	513
FFT (Xilinx 7.0)	3	5792
FIR (Xilinx 5.0)	240	120
FIR (Xilinx 5.0)	500	60

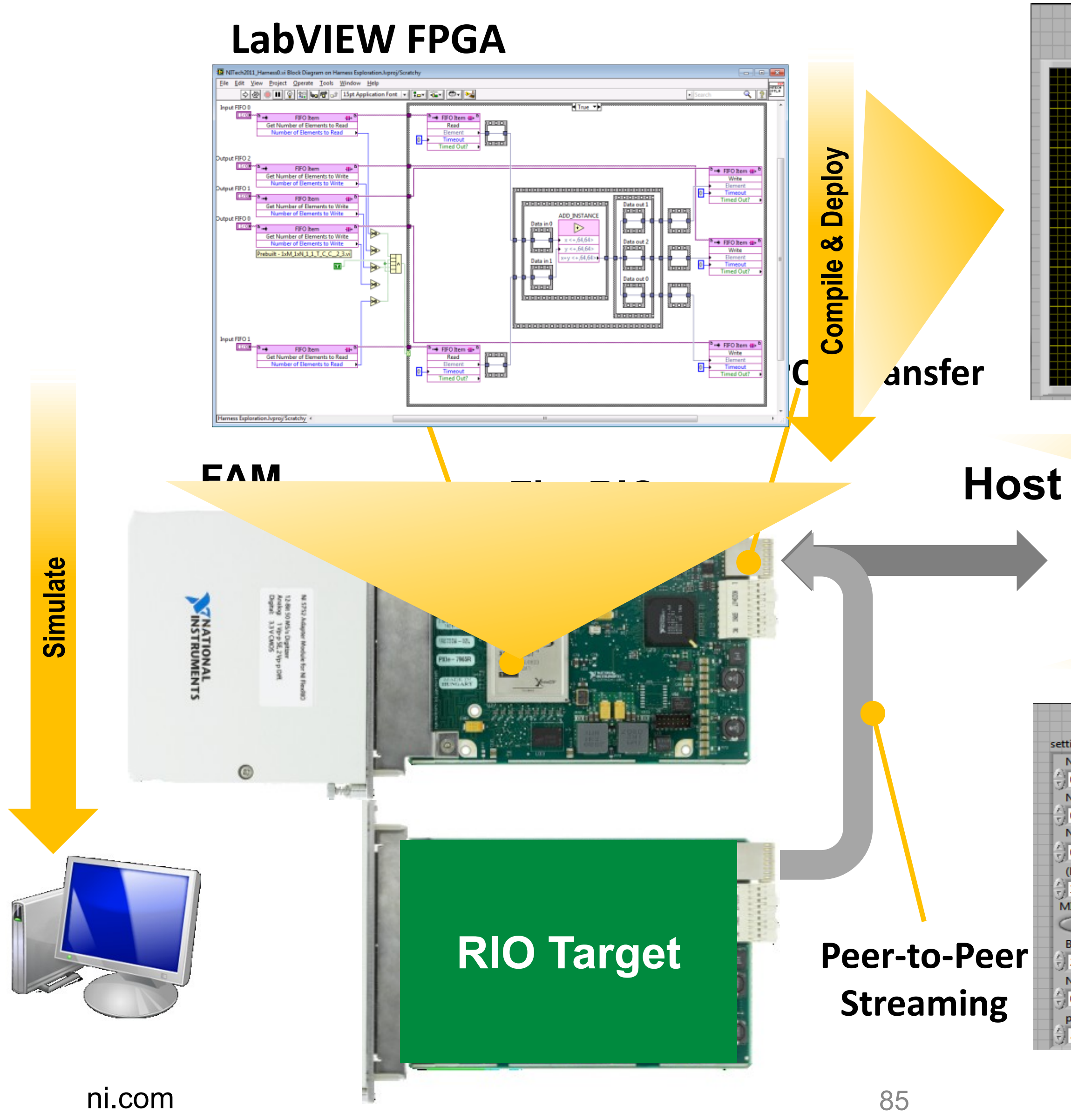
Calculated Schedule View

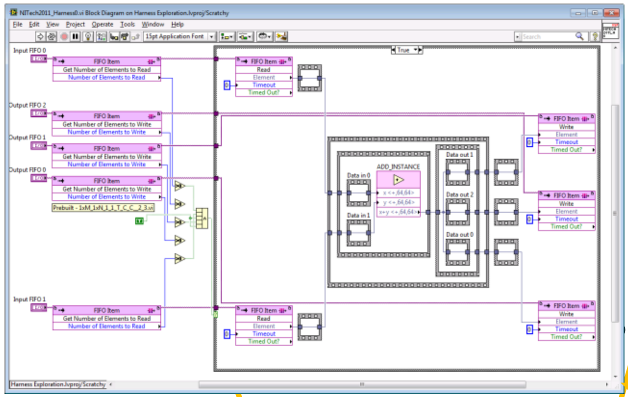
High-Level Model to FPGA blocks

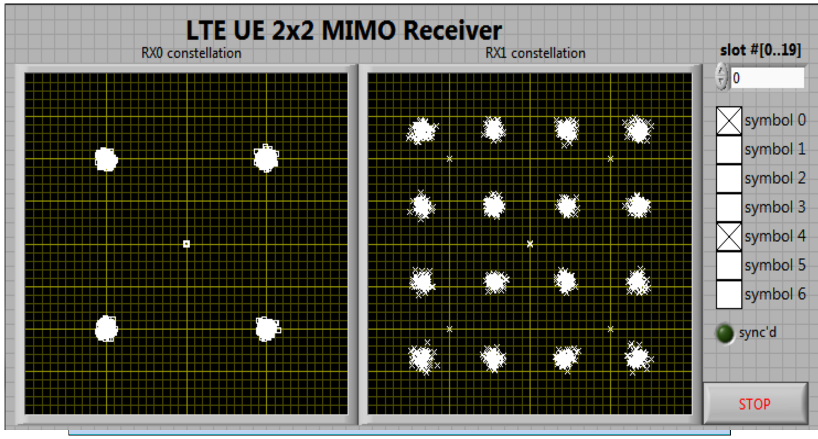




RF Design Flow







LTE UE 2x2 MIMO Receiver

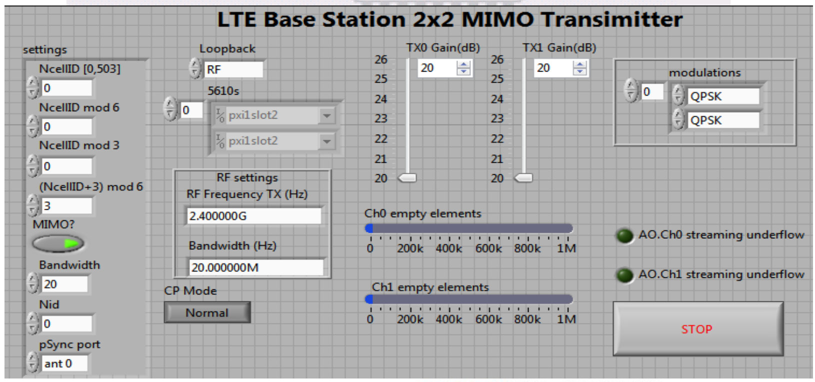
RX0 constellation · RX1 constellation

slot #[0..19]: 0

- ☒ symbol 0
- ☐ symbol 1
- ☐ symbol 2
- ☐ symbol 3
- ☒ symbol 4
- ☐ symbol 5
- ☐ symbol 6

● sync'd

STOP



LTE Base Station 2x2 MIMO Transimitter

settings
- NcellID [0,503]: 0
- NcellID mod 6: 0
- NcellID mod 3: 0
- (NcellID+3) mod 6: 3
- MIMO?
- Bandwidth: 20
- Nid: 0
- pSync port: ant 0

Loopback: RF

5610s: 0 — pxi1slot2, pxi1slot2

RF settings
- RF Frequency TX (Hz): 2.400000G
- Bandwidth (Hz): 20.000000M

CP Mode: Normal

TX0 Gain(dB): 20 · TX1 Gain(dB): 20

Ch0 empty elements · Ch1 empty elements

modulations: 0 — QPSK, QPSK

● AO.Ch0 streaming underflow

● AO.Ch1 streaming underflow

STOP

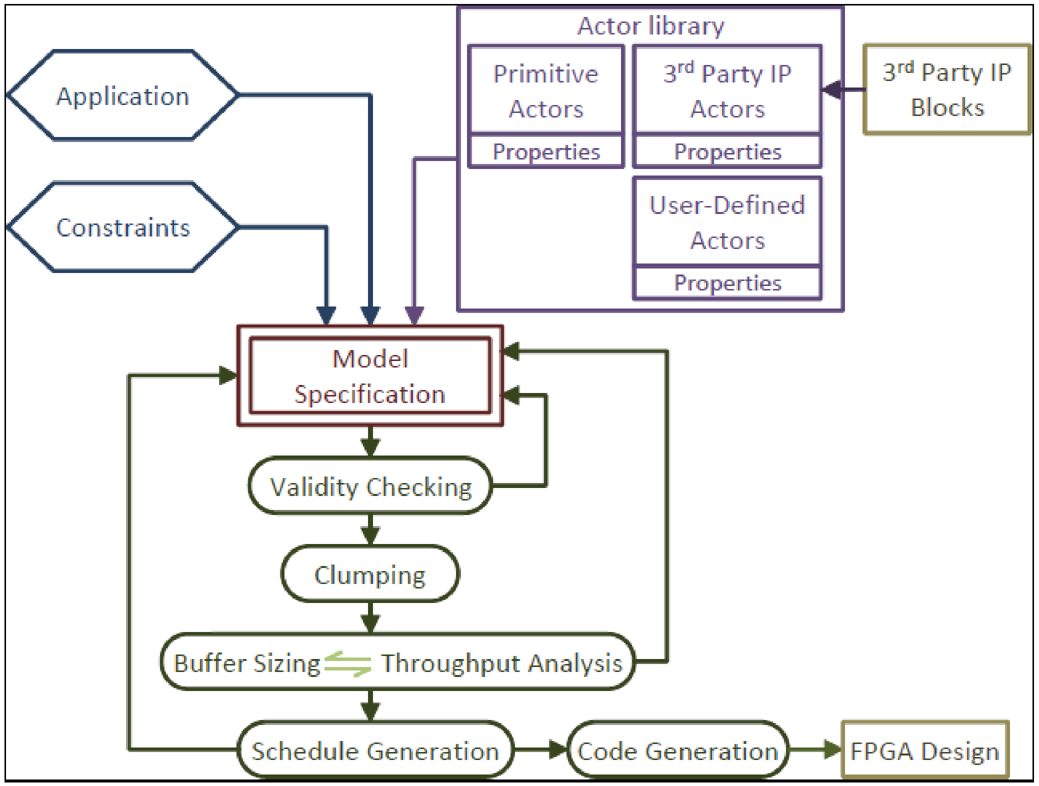
Related Frameworks

- Ptolemy II
  - + Prominent framework for exploring MoCs
  - – Code generation for HW not fully developed
- Grape-II
  - + Facilitates emulation of SDF/CSDF on FPGAs
  - – Lacking in smooth integration of IP
- LabVIEW FPGA
  - + Commercial framework for generation of HW from dataflow models
  - – No synthesis and optimization of multi-rate models
- Xilinx System Generator
  - + Commercial framework for HW generation from SR and DT models
  - – Not suitable for dataflow models/ limited HW optimization
- Agilent System Vue
  - + Support expressive dataflow models/ libraries for RF+DSP applications
  - – No path to implementation on specific HW targets
- Open DF and CAL
  - + HW generation from dataflow models/ generates VHDL
  - – Less analysis options/ Limited support for integration of commercial IP

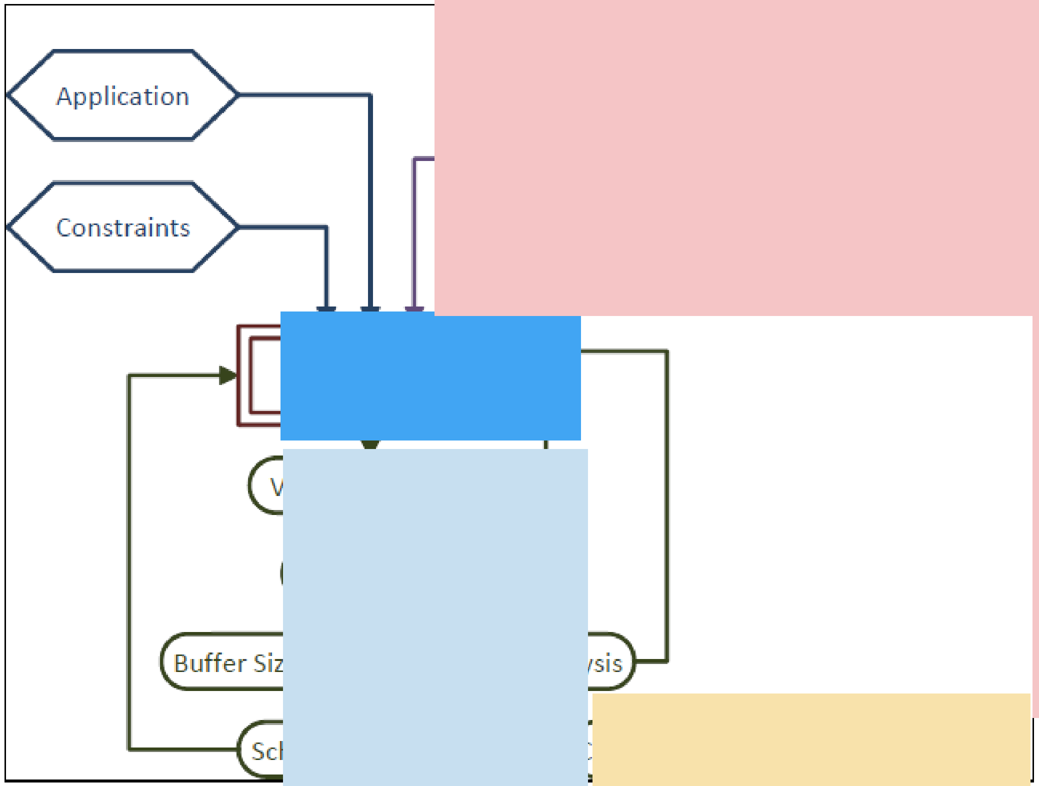
DSP Design Module – Summary

- Simplifies creation of complex DSP subsystems targeted for FPGA deployment, allowing
  - Fast prototype of real-time FPGA-based DSP subsystems
  - Integration of rich signal processing IP libraries that exploit FPGA and surrounding DSP fabric
  - Design of signal processing IP blocks with LabVIEW FPGA or by importing third-party IP
  - Exploration of design trade-offs between timing requirements and resource constraints

Tool Flow


Application
Constraints
Actor library
Primitive Actors
Properties
3rd Party IP Actors
Properties
3rd Party IP Blocks
User-Defined Actors
Properties
Model Specification
Validity Checking
Clumping
Buffer Sizing
Throughput Analysis
Schedule Generation
Code Generation
FPGA Design

Tool Flow (Focus Areas)



Models of Computation

Analysis and Optimization Back End

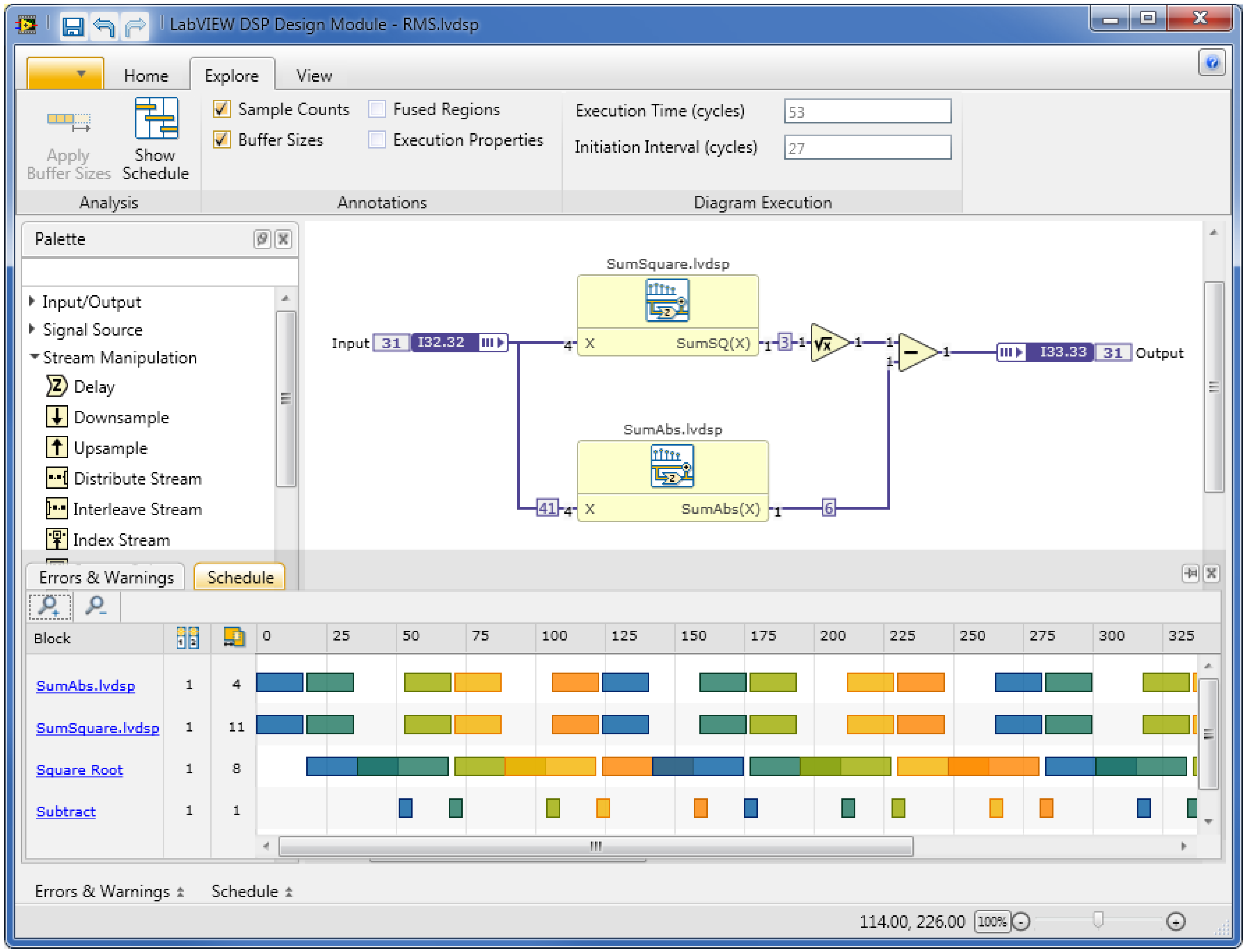
Simulation and Verification

Actor Definition

- Performance Models
- and Timing Library
- IP Modeling and
- Integration

- Code Generation and
- Implementation

GUI Screenshot

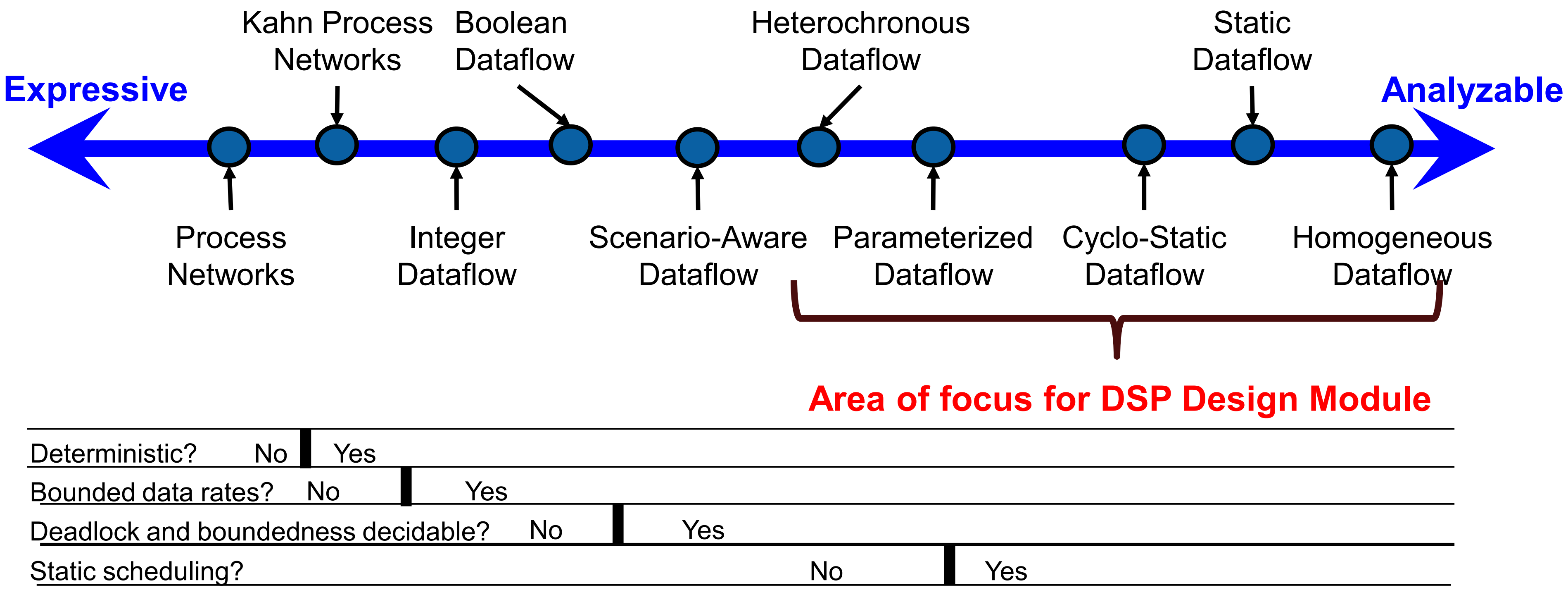


[✓] Sample Counts
[✓] Buffer Sizes
[ ] Fused Regions
[ ] Execution Properties

Execution Time (cycles): 53
Initiation Interval (cycles): 27

Block		
SumAbs.lvdsp	1	4
SumSquare.lvdsp	1	11
Square Root	1	8
Subtract	1	1

MoCs for Streaming Applications



Deterministic? No | Yes

Bounded data rates? No | Yes

Deadlock and boundedness decidable? No | Yes

Static scheduling? No | Yes

<u>Key trade-off</u>: Analyzability vs. Expressability

[1] Edward A. Lee, "Concurrent Models of Computation for Heterogeneous Software", EECS 290, 2004

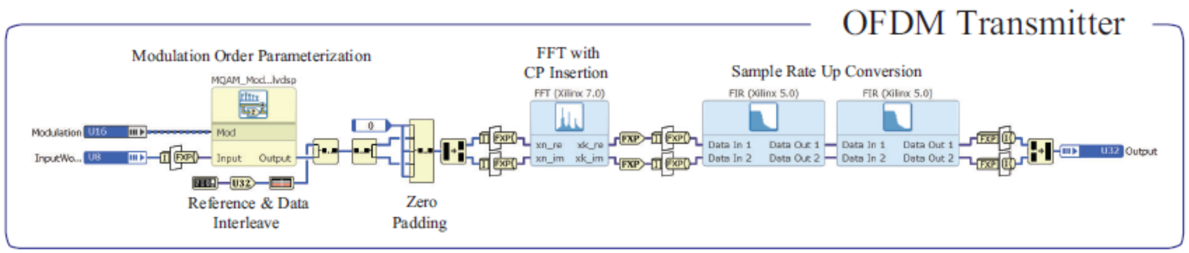
Analysis and Optimization Features

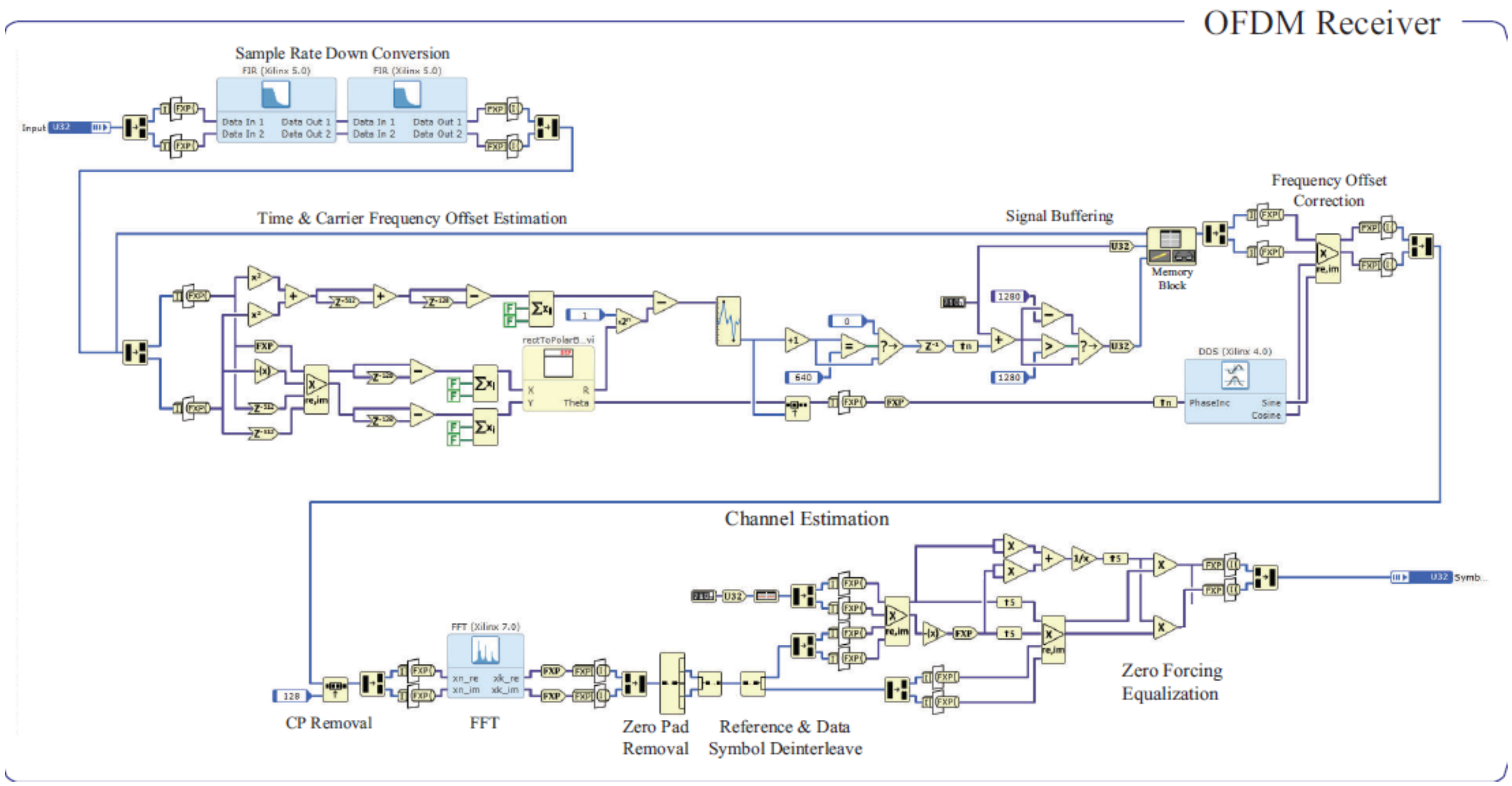
- Core dataflow optimizations
  - Model validation
    - Deadlock detection and boundedness check
  - Throughput and latency computation
  - Buffer size optimization (under throughput constraints)
  - Schedule computation

- Hardware specific optimizations
  - Resource constrained schedule computation
  - Retiming and fusion
  - Rate matching
  - IP interface synthesis

[1] S. S. Bhattacharyya, P. K. Murthy and E. A. Lee, "Software Synthesis from Dataflow Graphs," Kluwer Academic Publishers, Norwell, Mass, 1996.

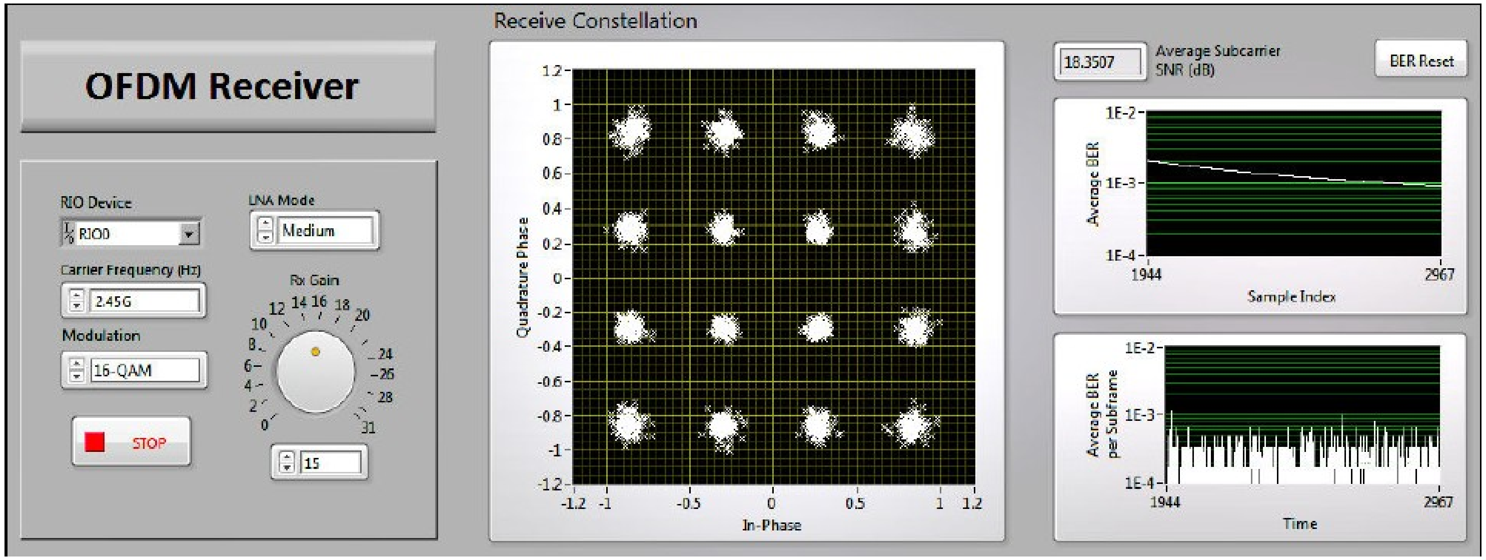
Design Capture in DSP Design Module





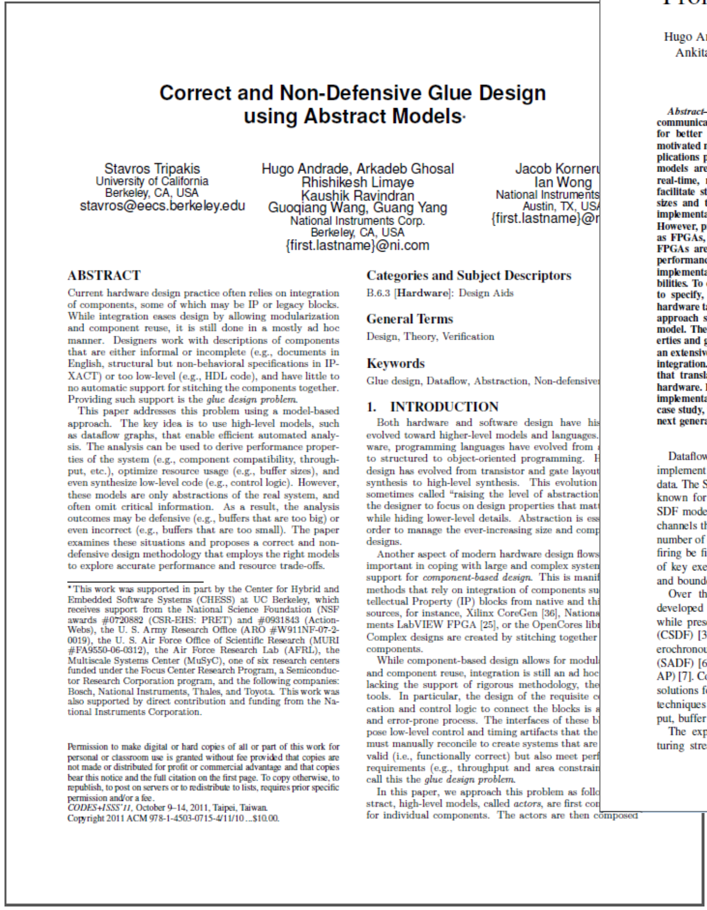
Synthesis Results for OFDM Rx/Tx Example

Resource Name	Available Resource Elements	Transmitter Utilization	Receiver Utilization
Slices	14,720	43.1%	79.2%
Slice Registers	58880	21.6%	54.6%
Slice LUTs	58880	24.7%	57.3%
DSP48s 640	640	2.7%	8.3%
Block RAM	244	8.2%	19.7%



RIO Device: RIO0
LNA Mode: Medium
Carrier Frequency (Hz): 2.45G
Modulation: 16-QAM
Rx Gain: 15
Average Subcarrier SNR (dB): 18.3507

Successful collaboration with UCB



Correct and Non-Defensive Glue Design using Abstract Models*

Stavros Tripakis
University of California
Berkeley, CA, USA
stavros@eecs.berkeley.edu

Hugo Andrade, Arkadeb Ghosal
Rhishikesh Limaye
Kaushik Ravindran
Guoqiang Wang, Guang Yang
National Instruments Corp.
Berkeley, CA, USA
{first.lastname}@ni.com

Jacob Kornerup
Ian Wong
National Instruments
Austin, TX, USA
{first.lastname}@n

ABSTRACT

Current hardware design practice often relies on integration of components, some of which may be IP or legacy blocks. While integration eases design by allowing modularization and component reuse, it is still done in a mostly ad hoc manner. Designers work with descriptions of components that are either informal or incomplete (e.g., documents in English, structural but non-behavioral specifications in IP-XACT) or too low-level (e.g., HDL code), and have little to no automatic support for stitching the components together. Providing such support is the *glue design problem*.

This paper addresses this problem using a model-based approach. The key idea is to use high-level models, such as dataflow graphs, that enable efficient automated analysis. The analysis can be used to derive performance properties of the system (e.g., component compatibility, throughput, etc.), optimize resource usage (e.g., buffer sizes), and even synthesize low-level code (e.g., control logic). However, these models are only abstractions of the real system, and often omit critical information. As a result, the analysis outcomes may be defensive (e.g., buffers that are too big) or even incorrect (e.g., buffers that are too small). The paper examines these situations and proposes a correct and non-defensive design methodology that employs the right models to explore accurate performance and resource trade-offs.

*This work was supported in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF awards #0720882 (CSR-EHS: PRET) and #0931843 (Action-Webs), the U. S. Army Research Office (ARO #W911NF-07-2-0019), the U. S. Air Force Office of Scientific Research (MURI #FA9550-06-0312), the Air Force Research Lab (AFRL), the Multiscale Systems Center (MuSyC), one of six research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program, and the following companies: Bosch, National Instruments, Thales, and Toyota. This work was also supported by direct contribution and funding from the National Instruments Corporation.

Categories and Subject Descriptors
B.6.3 [**Hardware**]: Design Aids

General Terms
Design, Theory, Verification

Keywords
Glue design, Dataflow, Abstraction, Non-defensive

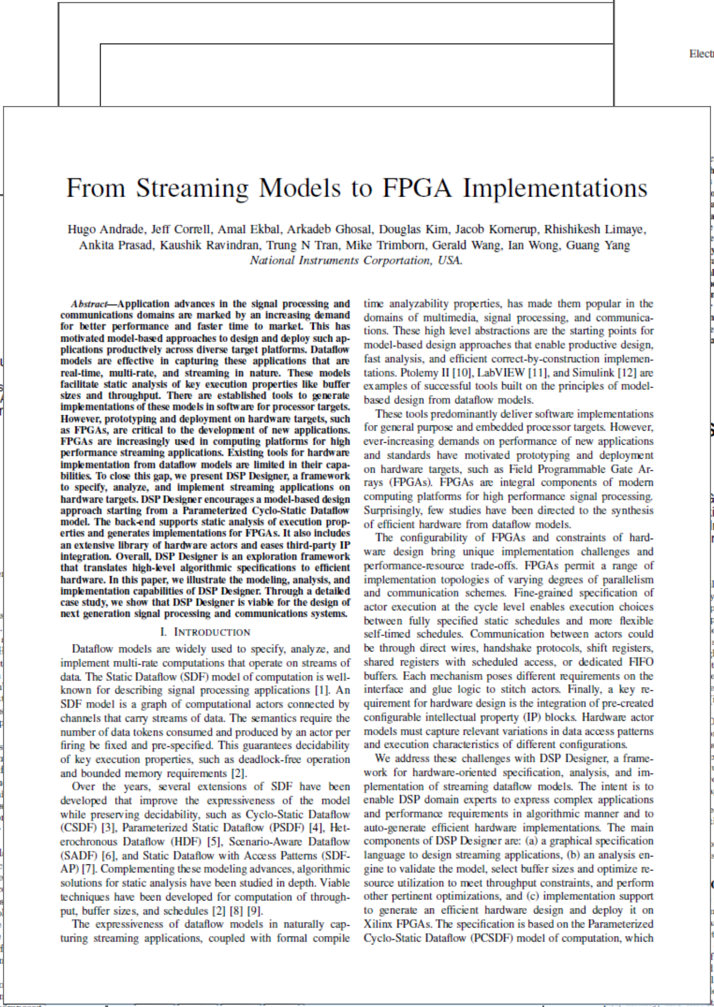
1. INTRODUCTION

Both hardware and software design have his
evolved toward higher-level models and languages.
ware, programming languages have evolved from
to structured to object-oriented programming. H
design has evolved from transistor and gate layout
synthesis to high-level synthesis. This evolution
sometimes called "raising the level of abstraction
the designer to focus on design properties that matt
while hiding lower-level details. Abstraction is ess
order to manage the ever-increasing size and comp
designs.

Another aspect of modern hardware design flows
important in coping with large and complex systen
support for *component-based design*. This is manif
methods that rely on integration of components su
tellectual Property (IP) blocks from native and thi
sources, for instance, Xilinx CoreGen [36], Nationa
ments LabVIEW FPGA [25], or the OpenCores libr
Complex designs are created by stitching together
components.

While component-based design allows for modul
and component reuse, integration is still an ad hoc
lacking the support of rigorous methodology, the
tools. In particular, the design of the requisite c
cation and control logic to connect the blocks is s
and error-prone process. The interfaces of these bl
pose low-level control and timing artifacts that the
must manually reconcile to create systems that are
valid (i.e., functionally correct) but also meet perf
requirements (e.g., throughput and area constrain
call this the *glue design problem*.

In this paper, we approach this problem as follo
stract, high-level models, called *actors*, are first con
for individual components. The actors are then composed



From Streaming Models to FPGA Implementations

Hugo Andrade, Jeff Correll, Amal Ekbal, Arkadeb Ghosal, Douglas Kim, Jacob Kornerup, Rhishikesh Limaye, Ankita Prasad, Kaushik Ravindran, Trung N Tran, Mike Trimborn, Gerald Wang, Ian Wong, Guang Yang
National Instruments Corportation, USA.

Abstract—**Application advances in the signal processing and communications domains are marked by an increasing demand for better performance and faster time to market. This has motivated model-based approaches to design and deploy such applications productively across diverse target platforms. Dataflow models are effective in capturing these applications that are real-time, multi-rate, and streaming in nature. These models facilitate static analysis of key execution properties like buffer sizes and throughput. There are established tools to generate implementations of these models in software for processor targets. However, prototyping and deployment on hardware targets, such as FPGAs, are critical to the development of new applications. FPGAs are increasingly used in computing platforms for high performance streaming applications. Existing tools for hardware implementation from dataflow models are limited in their capabilities. To close this gap, we present DSP Designer, a framework to specify, analyze, and implement streaming applications on hardware targets. DSP Designer encourages a model-based design approach starting from a Parameterized Cyclo-Static Dataflow model. The back-end supports static analysis of execution properties and generates implementations for FPGAs. It also includes an extensive library of hardware actors and eases third-party IP integration. Overall, DSP Designer is an exploration framework that translates high-level algorithmic specifications to efficient hardware. In this paper, we illustrate the modeling, analysis, and implementation capabilities of DSP Designer. Through a detailed case study, we show that DSP Designer is viable for the design of next generation signal processing and communications systems.**

I. Introduction

Dataflow models are widely used to specify, analyze, and implement multi-rate computations that operate on streams of data. The Static Dataflow (SDF) model of computation is well-known for describing signal processing applications [1]. An SDF model is a graph of computational actors connected by channels that carry streams of data. The semantics require the number of data tokens consumed and produced by an actor per firing be fixed and pre-specified. This guarantees decidability of key execution properties, such as deadlock-free operation and bounded memory requirements [2].

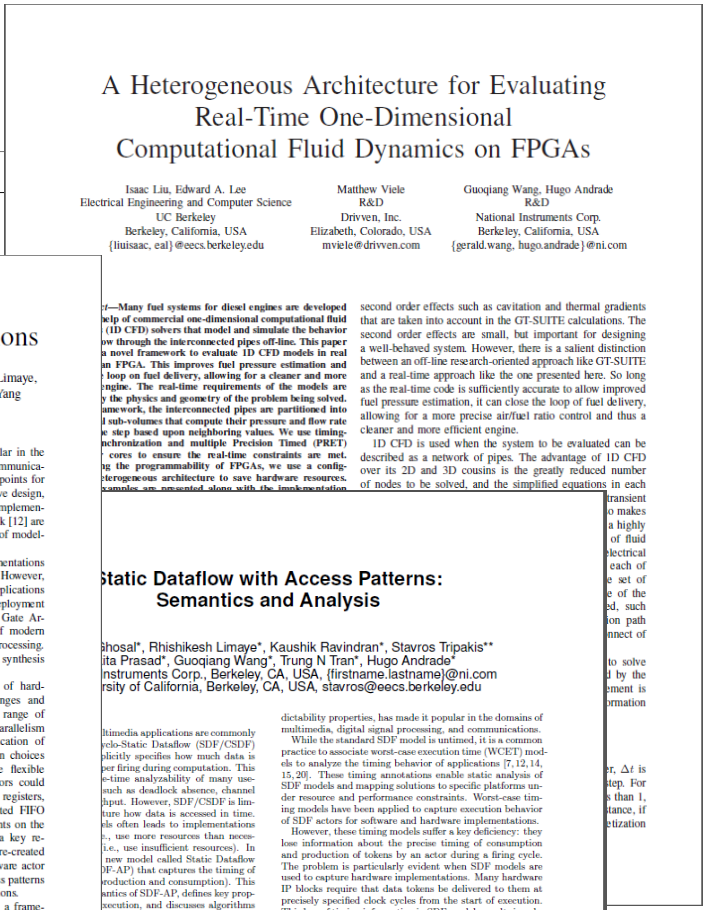
Over the years, several extensions of SDF have been developed that improve the expressiveness of the model while preserving decidability, such as Cyclo-Static Dataflow (CSDF) [3], Parameterized Static Dataflow (PSDF) [4], Heterochronous Dataflow (HDF) [5], Scenario-Aware Dataflow (SADF) [6], and Static Dataflow with Access Patterns (SDF-AP) [7]. Complementing these modeling advances, algorithmic solutions for static analysis have been studied in depth. Viable techniques have been developed for computation of throughput, buffer sizes, and schedules [2] [8] [9].

The expressiveness of dataflow models in naturally capturing streaming applications, coupled with formal compile time analyzability properties, has made them popular in the domains of multimedia, signal processing, and communications. These high level abstractions are the starting points for model-based design approaches that enable productive design, fast analysis, and efficient correct-by-construction implementations. Ptolemy II [10], LabVIEW [11], and Simulink [12] are examples of successful tools built on the principles of model-based design from dataflow models.

These tools predominantly deliver software implementations for general purpose and embedded processor targets. However, ever-increasing demands on performance of new applications and standards have motivated prototyping and deployment on hardware targets, such as Field Programmable Gate Arrays (FPGAs). FPGAs are integral components of modern computing platforms for high performance signal processing. Surprisingly, few studies have been directed to the synthesis of efficient hardware from dataflow models.

The configurability of FPGAs and constraints of hardware design bring unique implementation challenges and performance-resource trade-offs. FPGAs permit a range of implementation topologies of varying degrees of parallelism and communication schemes. Fine-grained specification of actor execution at the cycle level enables execution choices between fully specified static schedules and more flexible self-timed schedules. Communication between actors could be through direct wires, handshake protocols, shift registers, shared registers with scheduled access, or dedicated FIFO buffers. Each mechanism poses different requirements on the interface and glue logic to stitch actors. Finally, a key requirement for hardware design is the integration of pre-created configurable intellectual property (IP) blocks. Hardware actor models must capture relevant variations in data access patterns and execution characteristics of different configurations.

We address these challenges with DSP Designer, a framework for hardware-oriented specification, analysis, and implementation of streaming dataflow models. The intent is to enable DSP domain experts to express complex applications and performance requirements in algorithmic manner and to auto-generate efficient hardware implementations. The main components of DSP Designer are: (a) a graphical specification language to design streaming applications, (b) an analysis engine to validate the model, select buffer sizes and optimize resource utilization to meet throughput constraints, and perform other pertinent optimizations, and (c) implementation support to generate an efficient hardware design and deploy it on Xilinx FPGAs. The specification is based on the Parameterized Cyclo-Static Dataflow (PCSDF) model of computation, which



A Heterogeneous Architecture for Evaluating Real-Time One-Dimensional Computational Fluid Dynamics on FPGAs

Isaac Liu, Edward A. Lee
Electrical Engineering and Computer Science
UC Berkeley
Berkeley, California, USA
{liuisaac, eal}@eecs.berkeley.edu

Matthew Viele
R&D
Drivven, Inc.
Elizabeth, Colorado, USA
mviele@drivven.com

Guoqiang Wang, Hugo Andrade
R&D
National Instruments Corp.
Berkeley, California, USA
{gerald.wang, hugo.andrade}@ni.com

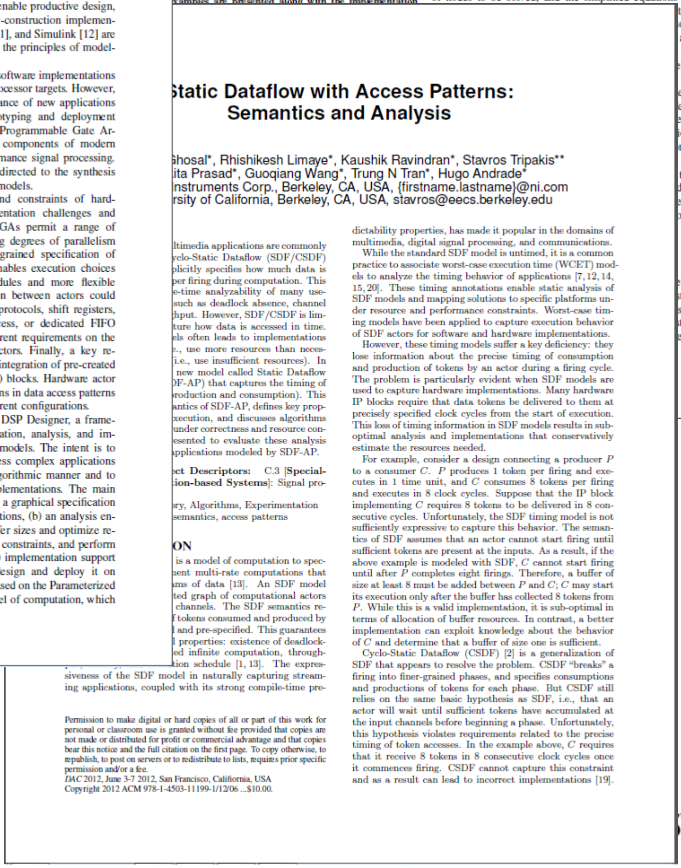
ct—**Many fuel systems for diesel engines are developed
help of commercial one-dimensional computational fluid
(1D CFD) solvers that model and simulate the behavior
ow through the interconnected pipes off-line. This paper
a novel framework to evaluate 1D CFD models in real
an FPGA. This improves fuel pressure estimation and
loop on fuel delivery, allowing for a cleaner and more
engine. The real-time requirements of the models are
y the physics and geometry of the problem being solved.
amework, the interconnected pipes are partitioned into
l sub-volumes that compute their pressure and flow rate
e step based upon neighboring values. We use timing-
nchronization and multiple Precision Timed (PRET)
cores to ensure the real-time constraints are met.
ng the programmability of FPGAs, we use a config-
eterogeneous architecture to save hardware resources.
xamples are presented along with the implementation**

second order effects such as cavitation and thermal gradients that are taken into account in the GT-SUITE calculations. The second order effects are small, but important for designing a well-behaved system. However, there is a salient distinction between an off-line research-oriented approach like GT-SUITE and a real-time approach like the one presented here. So long as the real-time code is sufficiently accurate to allow improved fuel pressure estimation, it can close the loop of fuel delivery, allowing for a more precise air/fuel ratio control and thus a cleaner and more efficient engine.

1D CFD is used when the system to be evaluated can be described as a network of pipes. The advantage of 1D CFD over its 2D and 3D cousins is the greatly reduced number of nodes to be solved, and the simplified equations in each
transient
o makes
a highly
of fluid
electrical
each of
e set of
e of the
ed, such
ion path
nnect of

to solve
d by the
ement is
ormation

er, Δt is
tep. For
s than 1,
tance, if
etization



Static Dataflow with Access Patterns: Semantics and Analysis

hosal*, Rhishikesh Limaye*, Kaushik Ravindran*, Stavros Tripakis**
ita Prasad*, Guoqiang Wang*, Trung N Tran*, Hugo Andrade*
Instruments Corp., Berkeley, CA, USA, {firstname.lastname}@ni.com
rsity of California, Berkeley, CA, USA, stavros@eecs.berkeley.edu

ltimedia applications are commonly
yclo-Static Dataflow (SDF/CSDF)
plicitly specifies how much data is
per firing during computation. This
e-time analyzability of many use-
such as deadlock absence, channel
hput. However, SDF/CSDF is lim-
ture how data is accessed in time.
els often leads to implementations
e., use more resources than neces-
(i.e., use insufficient resources). In
new model called Static Dataflow
DF-AP) that captures the timing of
production and consumption). This
antics of SDF-AP, defines key prop-
xecution, and discusses algorithms
under correctness and resource con-
esented to evaluate these analysis
applications modeled by SDF-AP.

ect Descriptors: C.3 [**Special-
tion-based Systems**]: Signal pro-

ory, Algorithms, Experimentation
semantics, access patterns

ON

is a model of computation to spec-
nent multi-rate computations that
ams of data [13]. An SDF model
ted graph of computational actors
channels. The SDF semantics re-
f tokens consumed and produced by
l and pre-specified. This guarantees
l properties: existence of deadlock-
ed infinite computation, through-
tion schedule [1, 13]. The expres-
siveness of the SDF model in naturally capturing streaming applications, coupled with its strong compile-time pre-

dictability properties, has made it popular in the domains of multimedia, digital signal processing, and communications.

While the standard SDF model is untimed, it is a common practice to associate worst-case execution time (WCET) models to analyze the timing behavior of applications [7, 12, 14, 15, 20]. These timing annotations enable static analysis of SDF models and mapping solutions to specific platforms under resource and performance constraints. Worst-case timing models have been applied to capture execution behavior of SDF actors for software and hardware implementations.

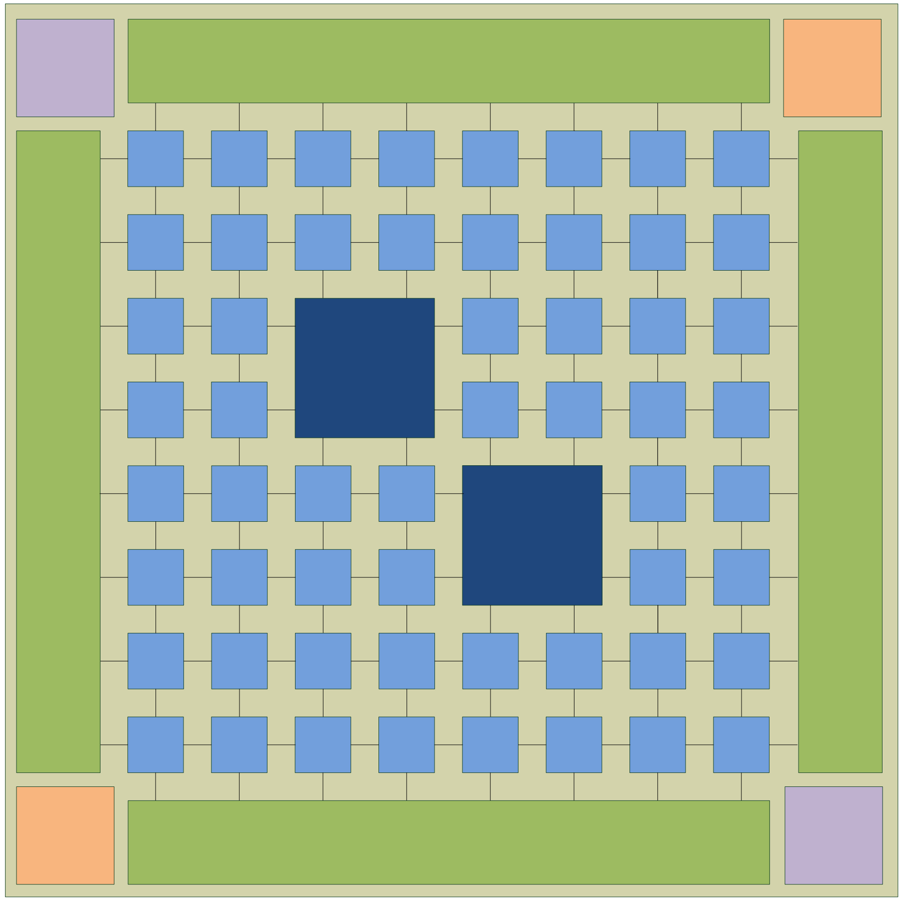
However, these timing models suffer a key deficiency: they lose information about the precise timing of consumption and production of tokens by an actor during a firing cycle. The problem is particularly evident when SDF models are used to capture hardware implementations. Many hardware IP blocks require that data tokens be delivered to them at precisely specified clock cycles from the start of execution. This loss of timing information in SDF models results in sub-optimal analysis and implementations that conservatively estimate the resources needed.

For example, consider a design connecting a producer P to a consumer C. P produces 1 token per firing and executes in 1 time unit, and C consumes 8 tokens per firing and executes in 8 clock cycles. Suppose that the IP block implementing C requires 8 tokens to be delivered in 8 consecutive cycles. Unfortunately, the SDF timing model is not sufficiently expressive to capture this behavior. The semantics of SDF assumes that an actor cannot start firing until sufficient tokens are present at the inputs. As a result, if the above example is modeled with SDF, C cannot start firing until after P completes eight firings. Therefore, a buffer of size at least 8 must be added between P and C; C may start its execution only after the buffer has collected 8 tokens from P. While this is a valid implementation, it is sub-optimal in terms of allocation of buffer resources. In contrast, a better implementation can exploit knowledge about the behavior of C and determine that a buffer of size one is sufficient.

Cyclo-Static Dataflow (CSDF) [2] is a generalization of SDF that appears to resolve the problem. CSDF "breaks" a firing into finer-grained phases, and specifies consumptions and productions of tokens for each phase. But CSDF still relies on the same basic hypothesis as SDF, i.e., that an actor will wait until sufficient tokens have accumulated at the input channels before beginning a phase. Unfortunately, this hypothesis violates requirements related to the precise timing of token accesses. In the example above, C requires that it receive 8 tokens in 8 consecutive clock cycles once it commences firing. CSDF cannot capture this constraint and as a result can lead to incorrect implementations [19].

Trends in Future Computational Platforms

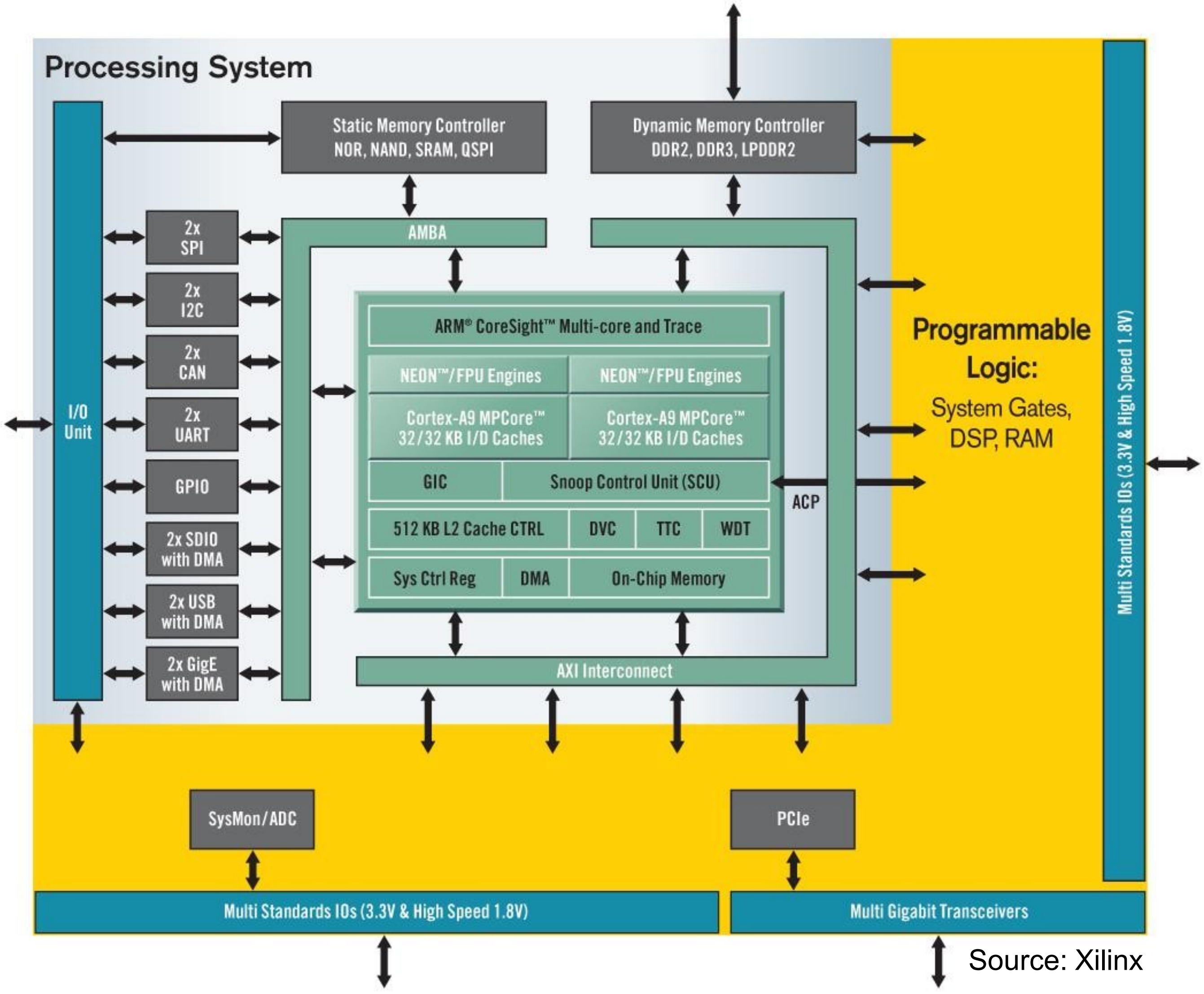
- Rapid increase in multi/ many core processors
- Convergence of architectures
- Gain in performance (speed, memory etc)
- Sophisticated power/ thermal management
- Unreliability from manufacturing technology
- NoC, high speed memory interface, specialized IO, reconfigurable fabric etc …



Future Heterogeneous Architectures

XILINX

Zynq Extensible Processing Platform



Processing System
I/O Unit
Static Memory Controller NOR, NAND, SRAM, QSPI
Dynamic Memory Controller DDR2, DDR3, LPDDR2
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA
ARM® CoreSight™ Multi-core and Trace
NEON™/FPU Engines
NEON™/FPU Engines
Cortex-A9 MPCore™ 32/32 KB I/D Caches
Cortex-A9 MPCore™ 32/32 KB I/D Caches
GIC
Snoop Control Unit (SCU)
ACP
512 KB L2 Cache CTRL
DVC
TTC
WDT
Sys Ctrl Reg
DMA
On-Chip Memory
AXI Interconnect
Programmable Logic:
System Gates, DSP, RAM
Multi Standards IOs (3.3V & High Speed 1.8V)
SysMon/ADC
PCIe
Multi Standards IOs (3.3V & High Speed 1.8V)
Multi Gigabit Transceivers

Source: Xilinx

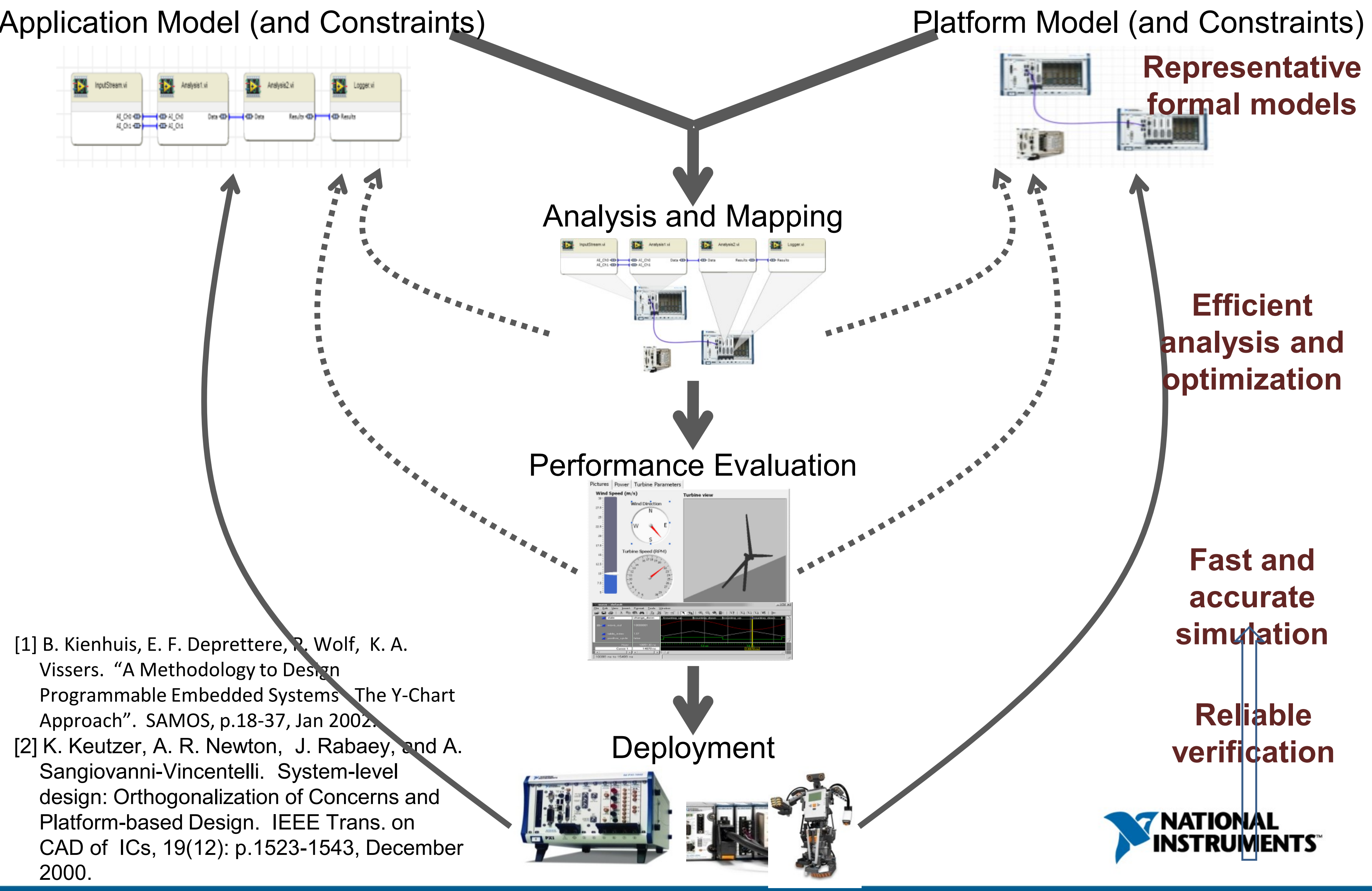
NATIONAL INSTRUMENTS™

Synthesis on Heterogeneous Platforms

- **Motivation**: To develop automatic system-level synthesis and exploration framework to deploy **high-level application** specifications onto **heterogeneous platforms**

- Goals:
  - Develop system-level language for the domain expert
  - Improve productivity while maintaining performance
  - Provide exploration framework to evaluate cost/quality, and derive optimal platform/ mapping
  - Allow system-level simulation/verification/validation to ensure model requirements

Y-Chart: A Disciplined System Design Methodology


Application Model (and Constraints)
Platform Model (and Constraints)
InputStream.vi
Analysis1.vi
Analysis2.vi
Logger.vi
Analysis and Mapping
Performance Evaluation
Deployment
Representative formal models
Efficient analysis and optimization
Fast and accurate simulation
Reliable verification

[1] B. Kienhuis, E. F. Deprettere, P. Wolf, K. A. Vissers. "A Methodology to Design Programmable Embedded Systems - The Y-Chart Approach". SAMOS, p.18-37, Jan 2002.

[2] K. Keutzer, A. R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of Concerns and Platform-based Design. IEEE Trans. on CAD of ICs, 19(12): p.1523-1543, December 2000.

Challenges

- Models for heterogeneous platform architectures
  - Computation, communication, I/O, Storage, UI, Cloud
- Mapping and Optimization
  (for distributed computation and communication)
  - Allocation, binding, reusing and scheduling
- Appropriate application description level
  - Models of computation, Domain specific knowledge
- System level validation
  - Testing, simulation, verification