MATLAB EXPO 2019

Adopting Model-Based Design for FPGA, ASIC, and SoC Development

Hitu Sharma
Application Engineer- Signal Processing, HDL & Communication

Agenda

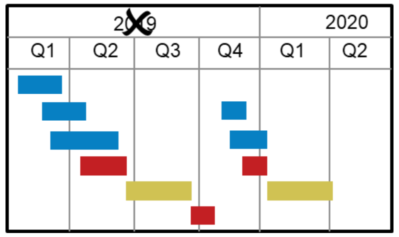
- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results

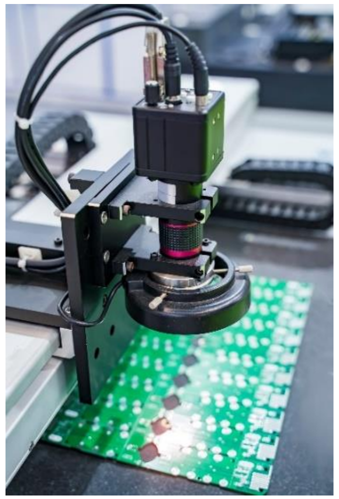
FPGA, ASIC, and SoC Development Projects

67% of ASIC/FPGA projects are **behind schedule**

Over **50% of project time** is spent on **verification**

75% of ASIC projects require a **silicon re-spin**

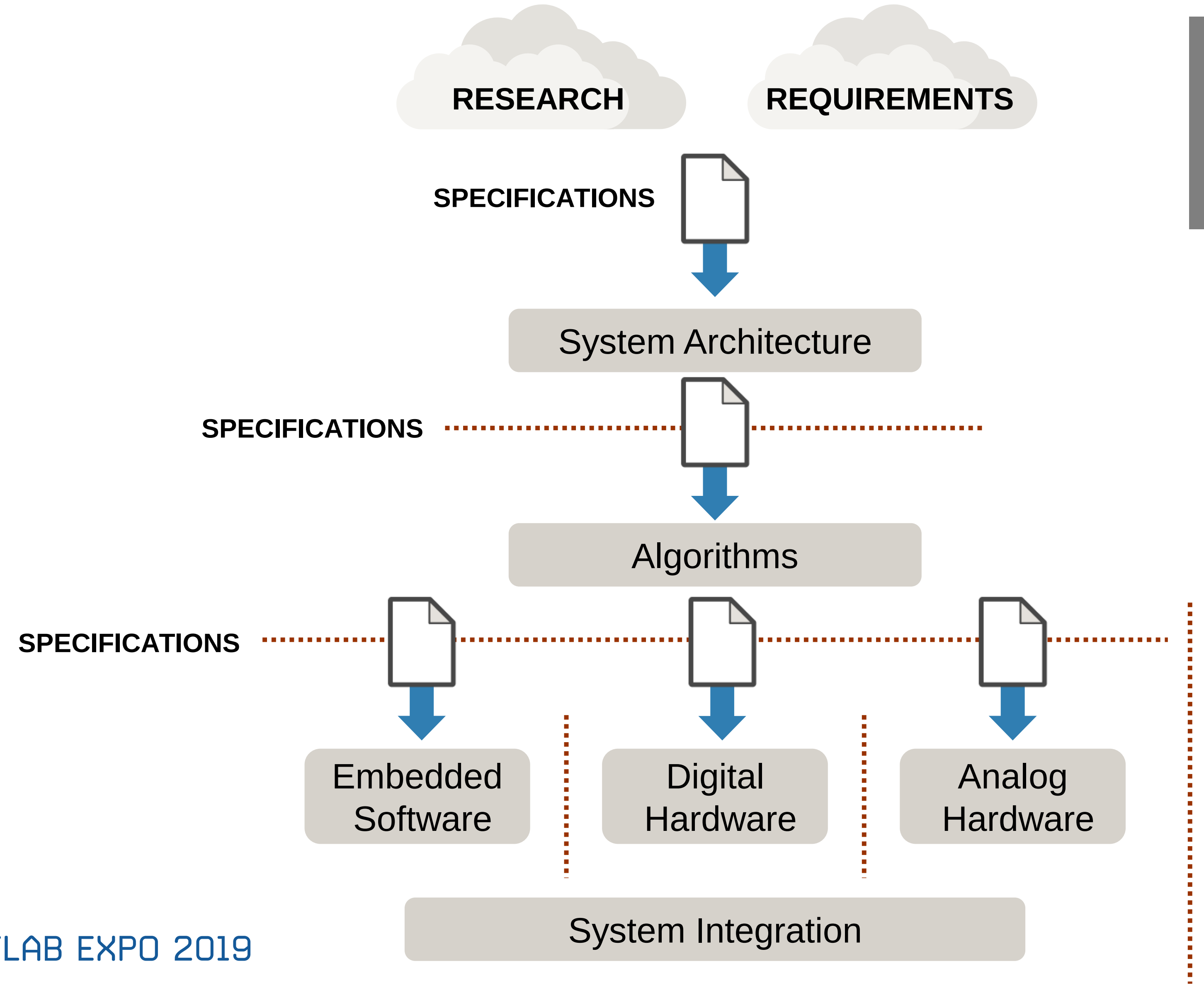




84% of FPGA projects have non-trivial **bugs escape into production**

Statistics from 2018 Mentor Graphics / Wilson Research survey, averaged over FPGA/ASIC

Many Different Skill Sets Need to Collaborate



- Poor communication across teams
- Key decisions made in silos
- System-level issues found in late stages
- Hard to adapt to changing requirements

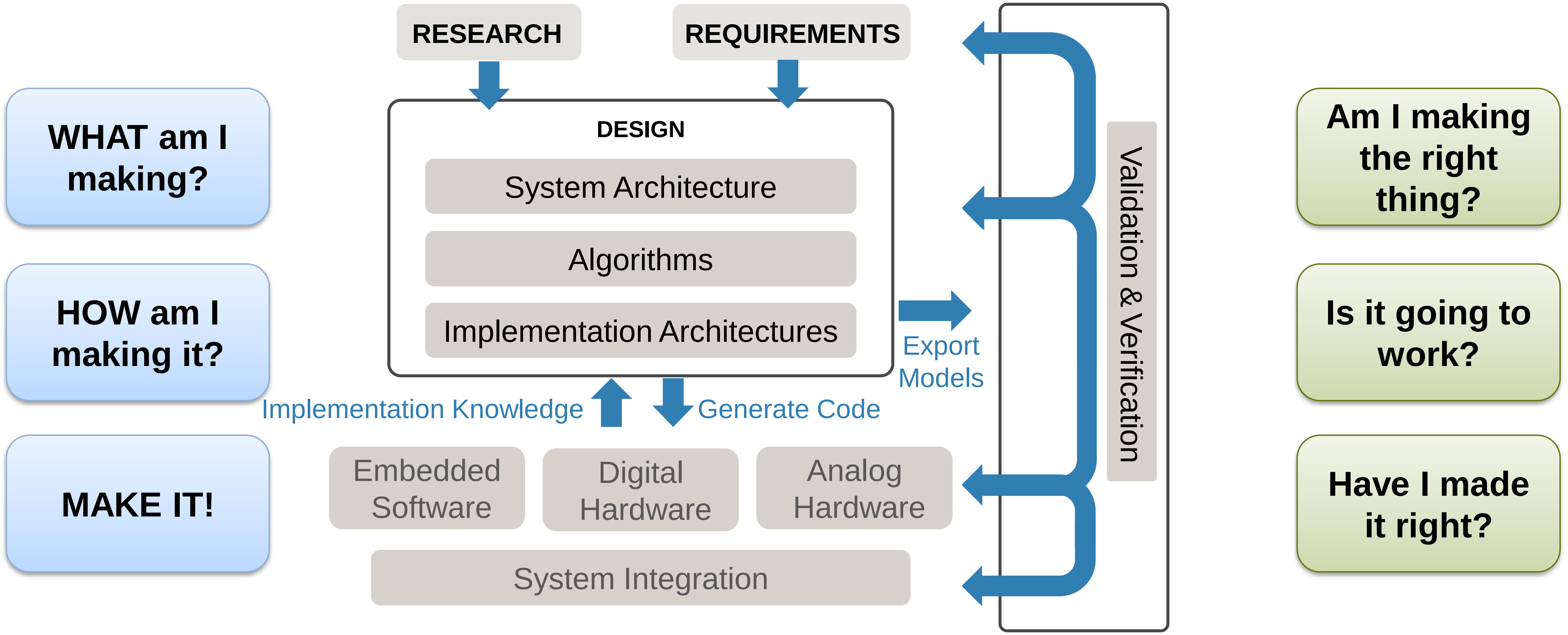
"In our previous, document-based design workflow, each team developed its own specification. This created a communication gap between the teams, as well as delays and the increased risk of error"

Noritaka Kosugi, Kazuyuki Hori, and Yuji Ishida

Hitachi

Verification

SoC Collaboration with Model-Based Design



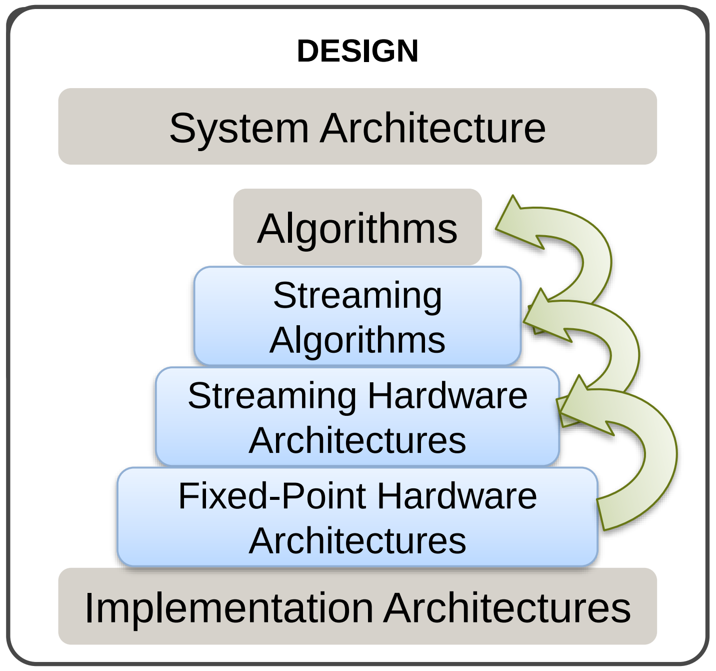
Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results

General Approach: Use the Strengths of MATLAB and Simulink

MATLAB

- ✓ Large data sets
- ✓ Explore mathematics
- ✓ Control logic
- ✓ Data visualization


DESIGN
System Architecture
Algorithms
Streaming Algorithms
Streaming Hardware Architectures
Fixed-Point Hardware Architectures
Implementation Architectures

Simulink

- ✓ Parallel architectures
- ✓ Timing
- ✓ Data type propagation
- ✓ Mixed-signal modeling

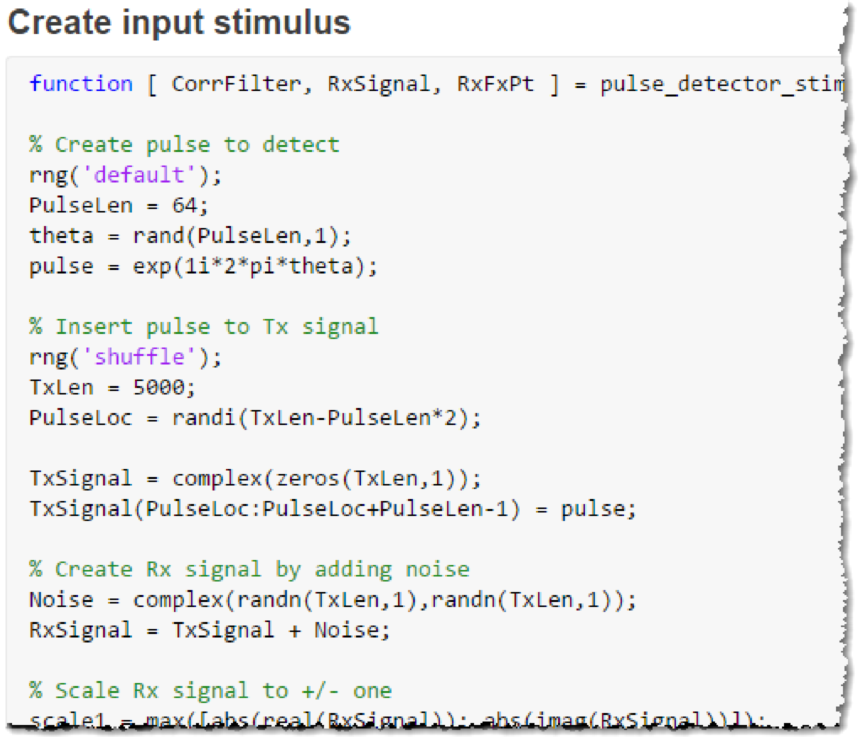
Partition Hardware-Targeted Design, System Context, Testbench

Algorithm Stimulus

Hardware Algorithm

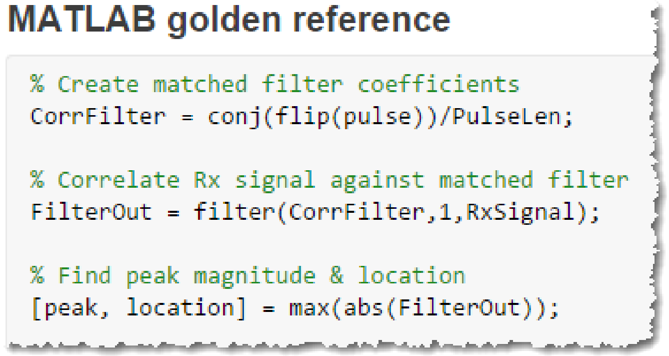
Software Algorithm

Analysis



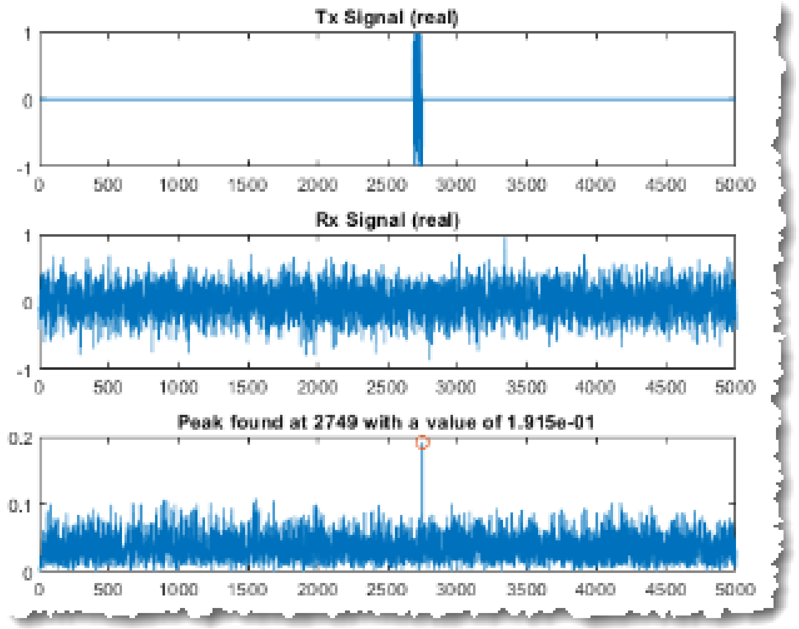
Create input stimulus

```
function [ CorrFilter, RxSignal, RxFxPt ] = pulse_detector_stim

% Create pulse to detect
rng('default');
PulseLen = 64;
theta = rand(PulseLen,1);
pulse = exp(1i*2*pi*theta);

% Insert pulse to Tx signal
rng('shuffle');
TxLen = 5000;
PulseLoc = randi(TxLen-PulseLen*2);

TxSignal = complex(zeros(TxLen,1));
TxSignal(PulseLoc:PulseLoc+PulseLen-1) = pulse;

% Create Rx signal by adding noise
Noise = complex(randn(TxLen,1),randn(TxLen,1));
RxSignal = TxSignal + Noise;

% Scale Rx signal to +/- one
scale1 = max([abs(real(RxSignal)); abs(imag(RxSignal))]);
```

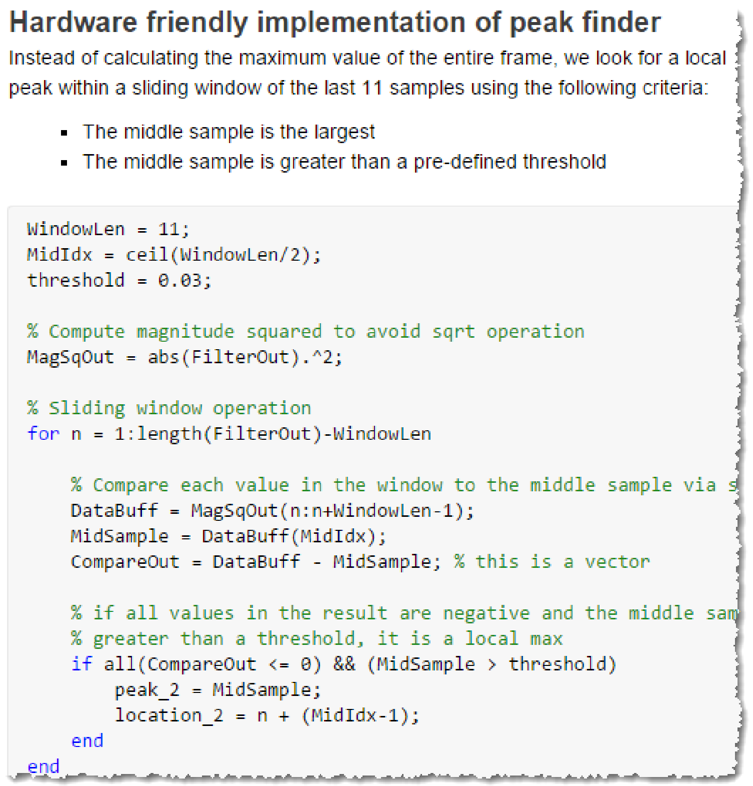


MATLAB golden reference

```
% Create matched filter coefficients
CorrFilter = conj(flip(pulse))/PulseLen;

% Correlate Rx signal against matched filter
FilterOut = filter(CorrFilter,1,RxSignal);

% Find peak magnitude & location
[peak, location] = max(abs(FilterOut));
```



Streaming Algorithms: MATLAB or Simulink…or Both

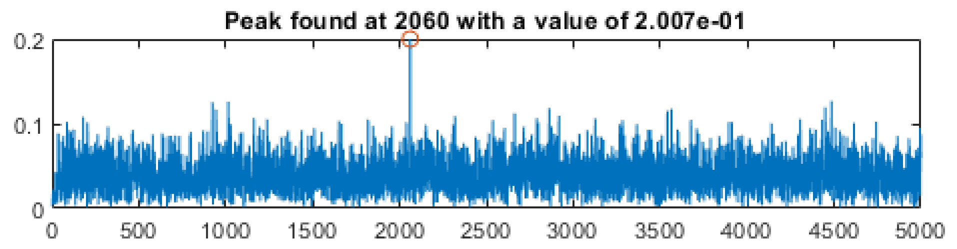


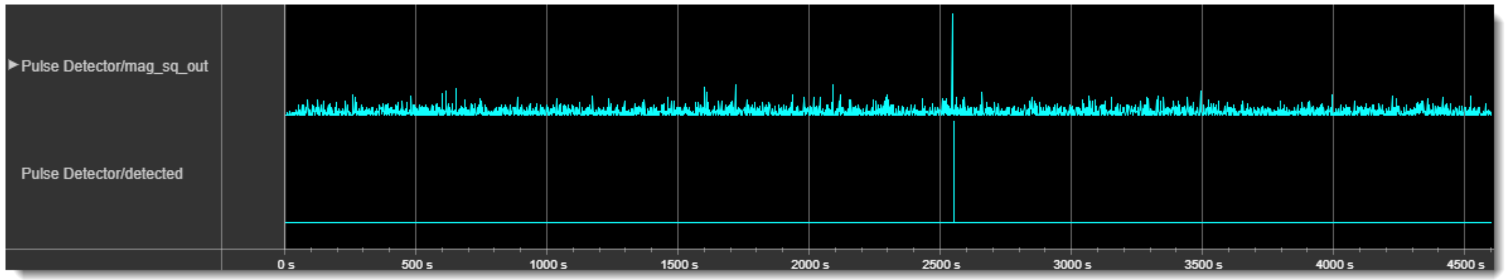
Hardware friendly implementation of peak finder

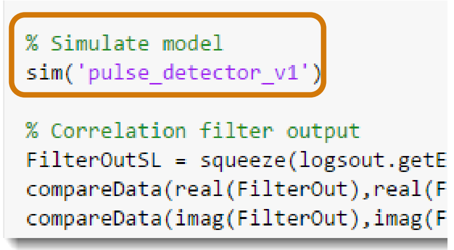
Instead of calculating the maximum value of the entire frame, we look for a local peak within a sliding window of the last 11 samples using the following criteria:

- The middle sample is the largest
- The middle sample is greater than a pre-defined threshold

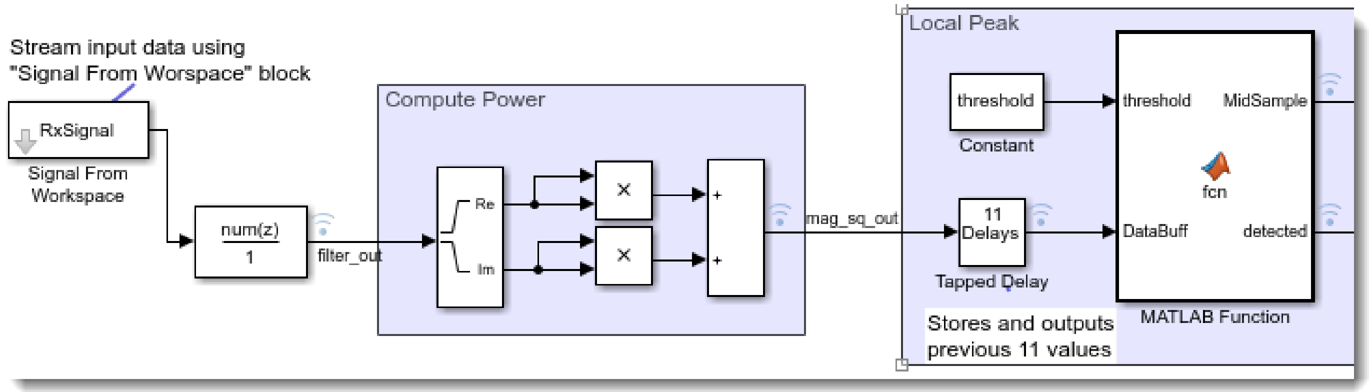
```
WindowLen = 11;
MidIdx = ceil(WindowLen/2);
threshold = 0.03;

% Compute magnitude squared to avoid sqrt operation
MagSqOut = abs(FilterOut).^2;

% Sliding window operation
for n = 1:length(FilterOut)-WindowLen

    % Compare each value in the window to the middle sample via s
    DataBuff = MagSqOut(n:n+WindowLen-1);
    MidSample = DataBuff(MidIdx);
    CompareOut = DataBuff - MidSample; % this is a vector

    % if all values in the result are negative and the middle sam
    % greater than a threshold, it is a local max
    if all(CompareOut <= 0) && (MidSample > threshold)
        peak_2 = MidSample;
        location_2 = n + (MidIdx-1);
    end
end
```


Peak found at 2060 with a value of 2.007e-01

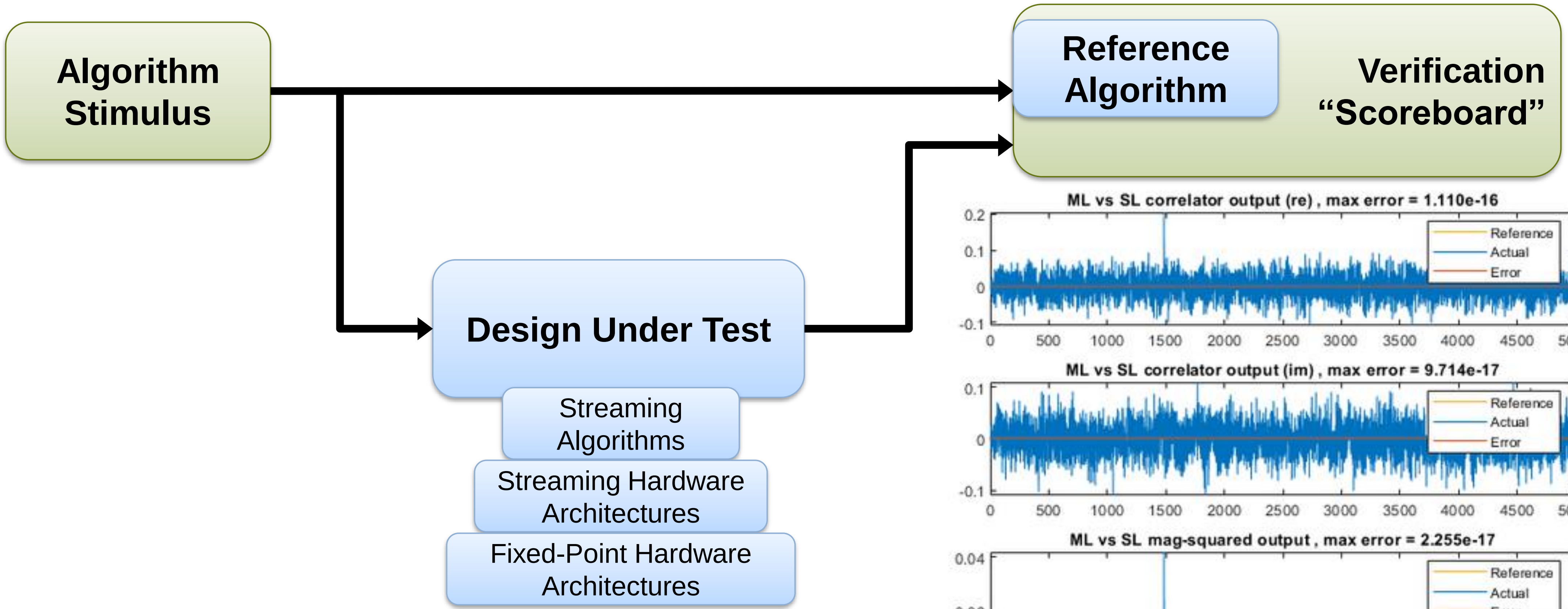




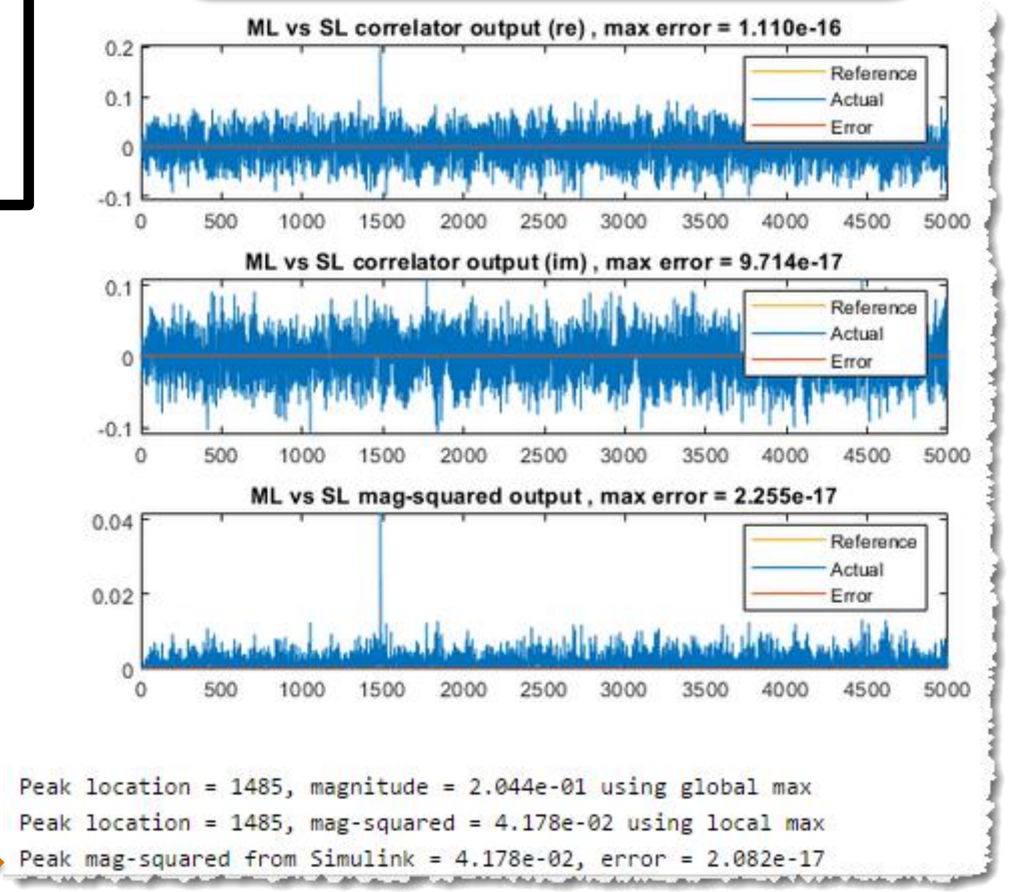
```
% Simulate model
sim('pulse_detector_v1')

% Correlation filter output
FilterOutSL = squeeze(logsout.getE
compareData(real(FilterOut),real(F
compareData(imag(FilterOut),imag(F
```



Refine Algorithm and Verify Against Golden Reference



Algorithm Stimulus
Reference Algorithm
Verification "Scoreboard"
Design Under Test
Streaming Algorithms
Streaming Hardware Architectures
Fixed-Point Hardware Architectures



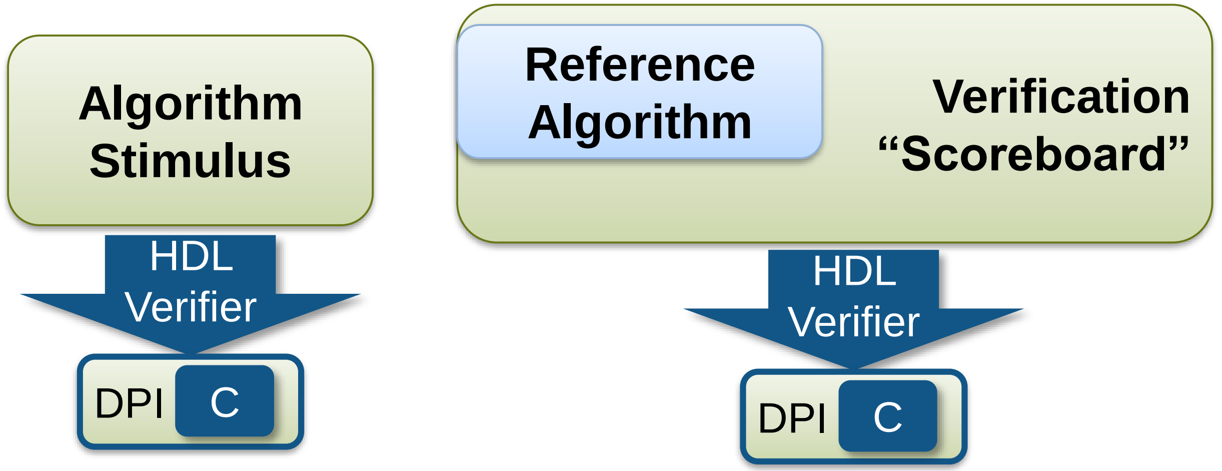
ML vs SL correlator output (re) , max error = 1.110e-16
ML vs SL correlator output (im) , max error = 9.714e-17
ML vs SL mag-squared output , max error = 2.255e-17
Reference
Actual
Error
0 500 1000 1500 2000 2500 3000 3500 4000 4500 5000

```
Peak location = 1485, magnitude = 2.044e-01 using global max
Peak location = 1485, mag-squared = 4.178e-02 using local max
Peak mag-squared from Simulink = 4.178e-02, error = 2.082e-17
```

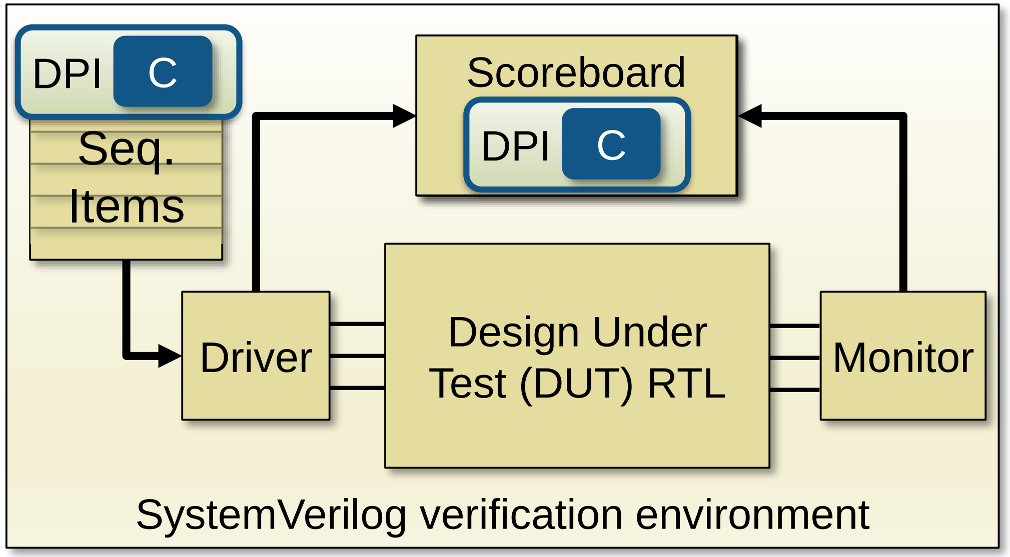
Self-checking

MATLAB EXPO 2019

Generate SystemVerilog DPI Components for RTL Verification

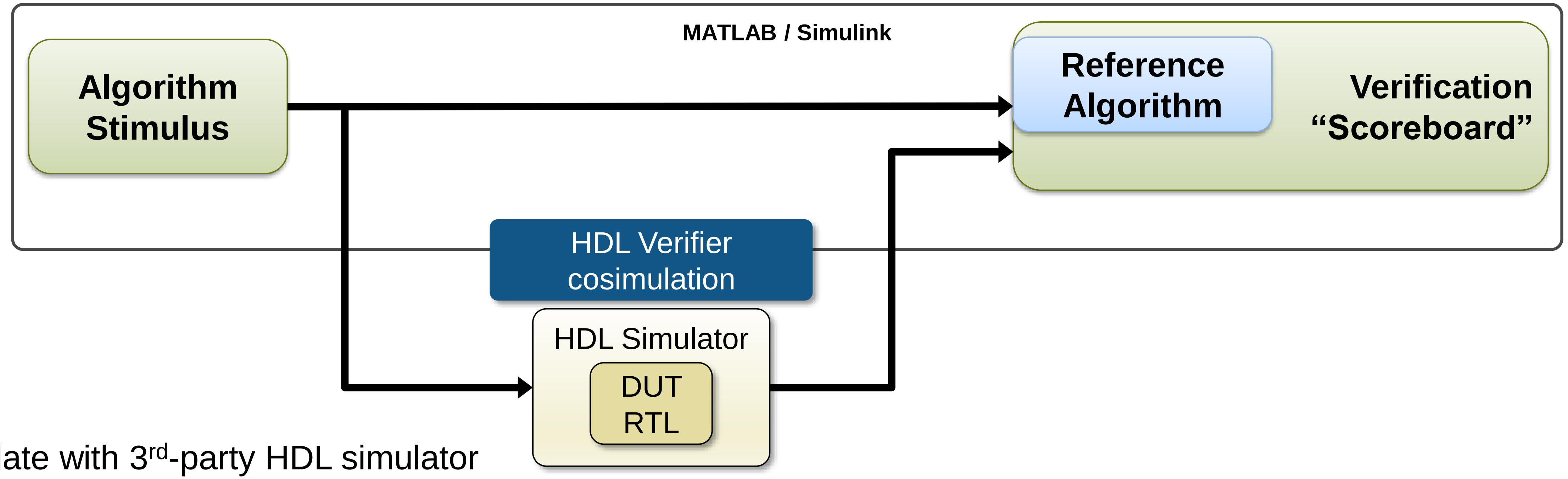


- Reuse MATLAB/Simulink models in verification
  - Scoreboard, stimulus, or models external to the RTL
    - Generate from frame-based or streaming algorithm
    - Floating-point or fixed-point
    - Individual components or entire testbench
  - Runs natively in SystemVerilog simulator
  - Eliminate re-work and miscommunication
  - Save testbench development time
  - Easy to update when requirements change

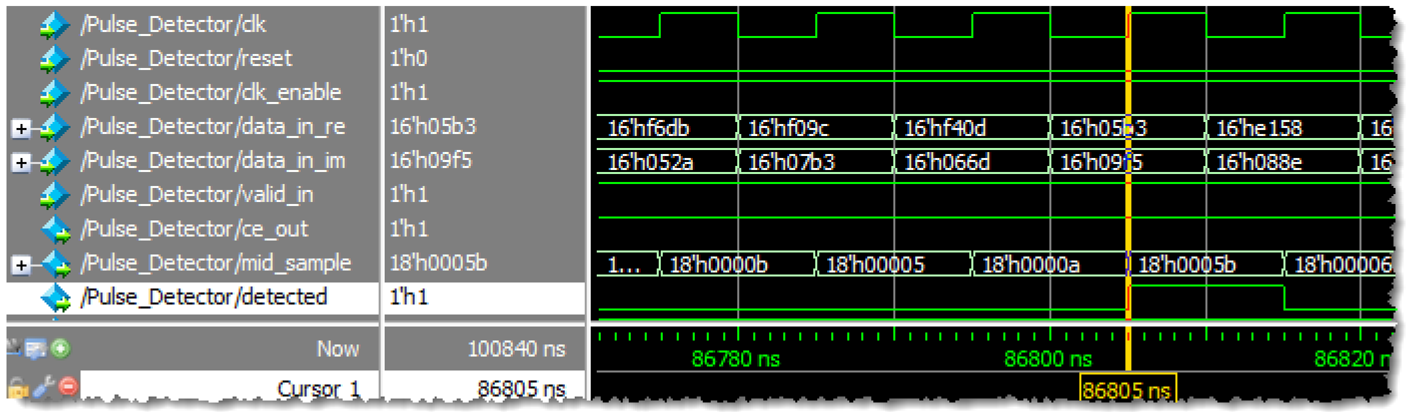


DPI
C
Seq. Items
Scoreboard
DPI
C
Driver
Design Under Test (DUT) RTL
Monitor
SystemVerilog verification environment

What if there’s a mismatch?



- Co-simulate with 3rd-party HDL simulator
  - Reuse MATLAB/Simulink test environment
  - Run HDL design in a supported simulator*
  - Generate co-simulation infrastructure and handshaking
  - Analyze both the design and test environment

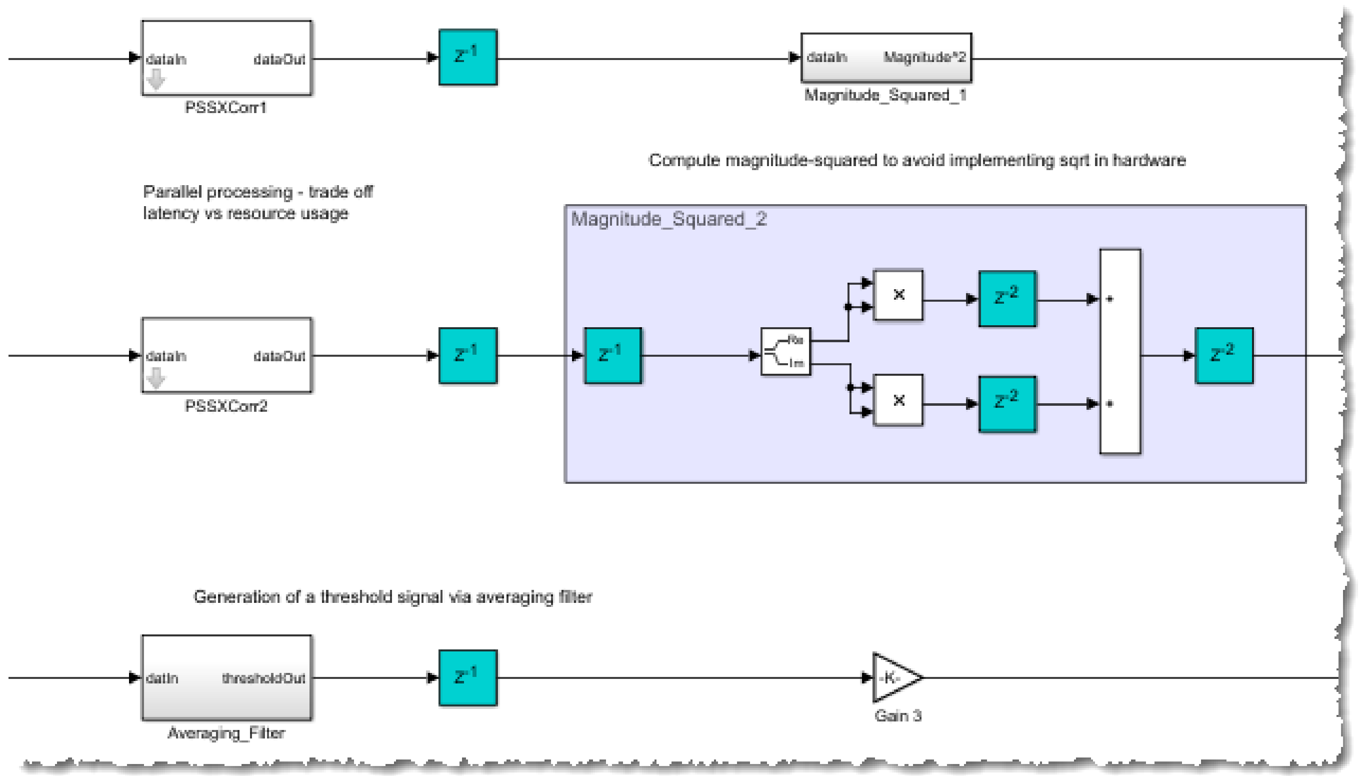


```
/Pulse_Detector/clk          1'h1
/Pulse_Detector/reset        1'h0
/Pulse_Detector/clk_enable   1'h1
/Pulse_Detector/data_in_re   16'h05b3    16'hf6db  16'hf09c  16'hf40d  16'h05b3  16'he158  16
/Pulse_Detector/data_in_im   16'h09f5    16'h052a  16'h07b3  16'h066d  16'h09f5  16'h088e  16
/Pulse_Detector/valid_in     1'h1
/Pulse_Detector/ce_out       1'h1
/Pulse_Detector/mid_sample   18'h0005b   1...  18'h0000b  18'h00005  18'h0000a  18'h0005b  18'h00006
/Pulse_Detector/detected     1'h1
Now                          100840 ns   86780 ns   86800 ns   86820 n
Cursor 1                     86805 ns    86805 ns
```

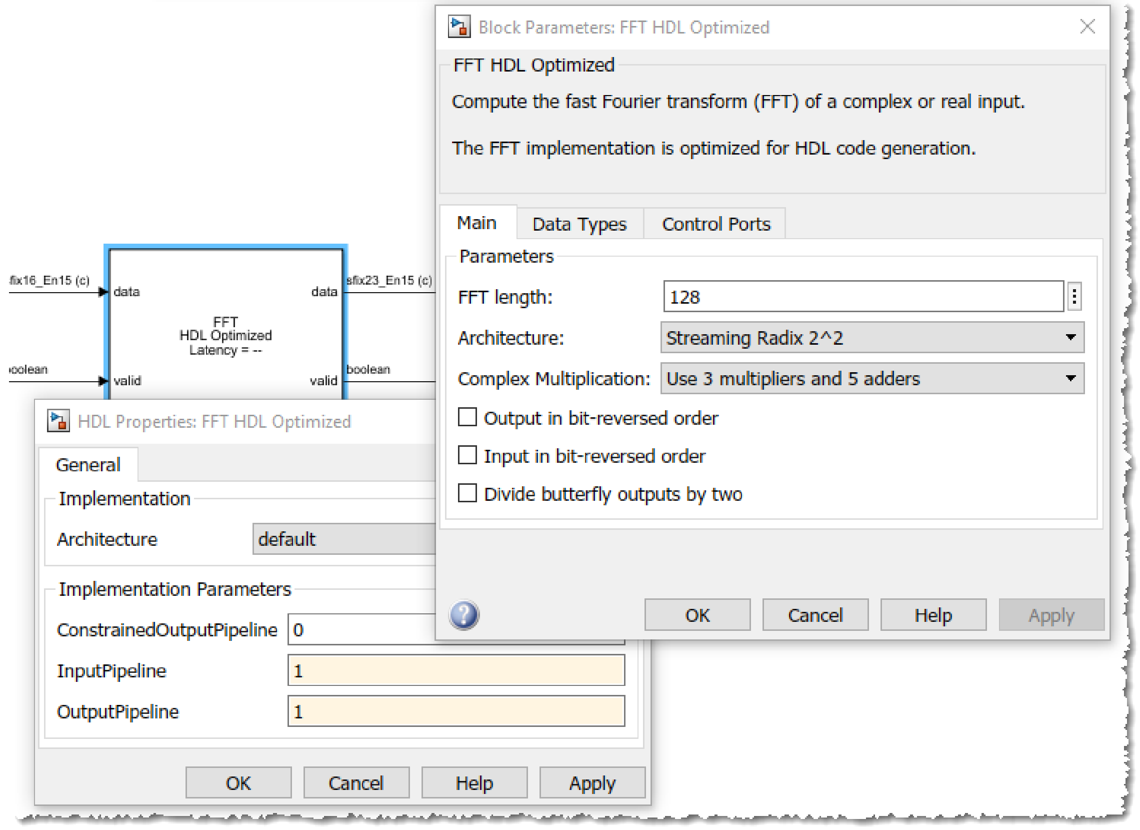
* Mentor Graphics® ModelSim® or Questa®
  Cadence® Incisive® or Xcelium™

Collaborate to Add Hardware Architecture

Optimize architecture design for hardware goals


dataIn
dataOut
PSSXCorr1
z⁻¹
dataIn
Magnitude^2
Magnitude_Squared_1
Compute magnitude-squared to avoid implementing sqrt in hardware
Parallel processing - trade off latency vs resource usage
Magnitude_Squared_2
dataIn
dataOut
PSSXCorr2
Re
Im
×
z⁻²
Generation of a threshold signal via averaging filter
dataIn
thresholdOut
Averaging_Filter
-K-
Gain 3

Specify HDL implementation options



Block Parameters: FFT HDL Optimized

FFT HDL Optimized

Compute the fast Fourier transform (FFT) of a complex or real input.

The FFT implementation is optimized for HDL code generation.

Main | Data Types | Control Ports

Parameters

FFT length:	128
Architecture:	Streaming Radix 2^2
Complex Multiplication:	Use 3 multipliers and 5 adders

☐ Output in bit-reversed order
☐ Input in bit-reversed order
☐ Divide butterfly outputs by two

OK | Cancel | Help | Apply

HDL Properties: FFT HDL Optimized

General

Implementation

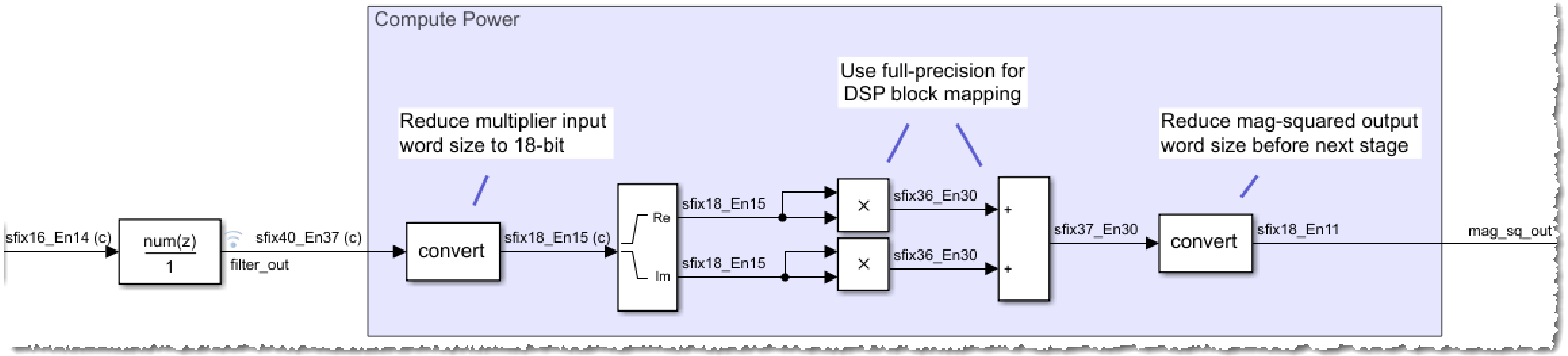
Architecture	default

Implementation Parameters

ConstrainedOutputPipeline	0
InputPipeline	1
OutputPipeline	1

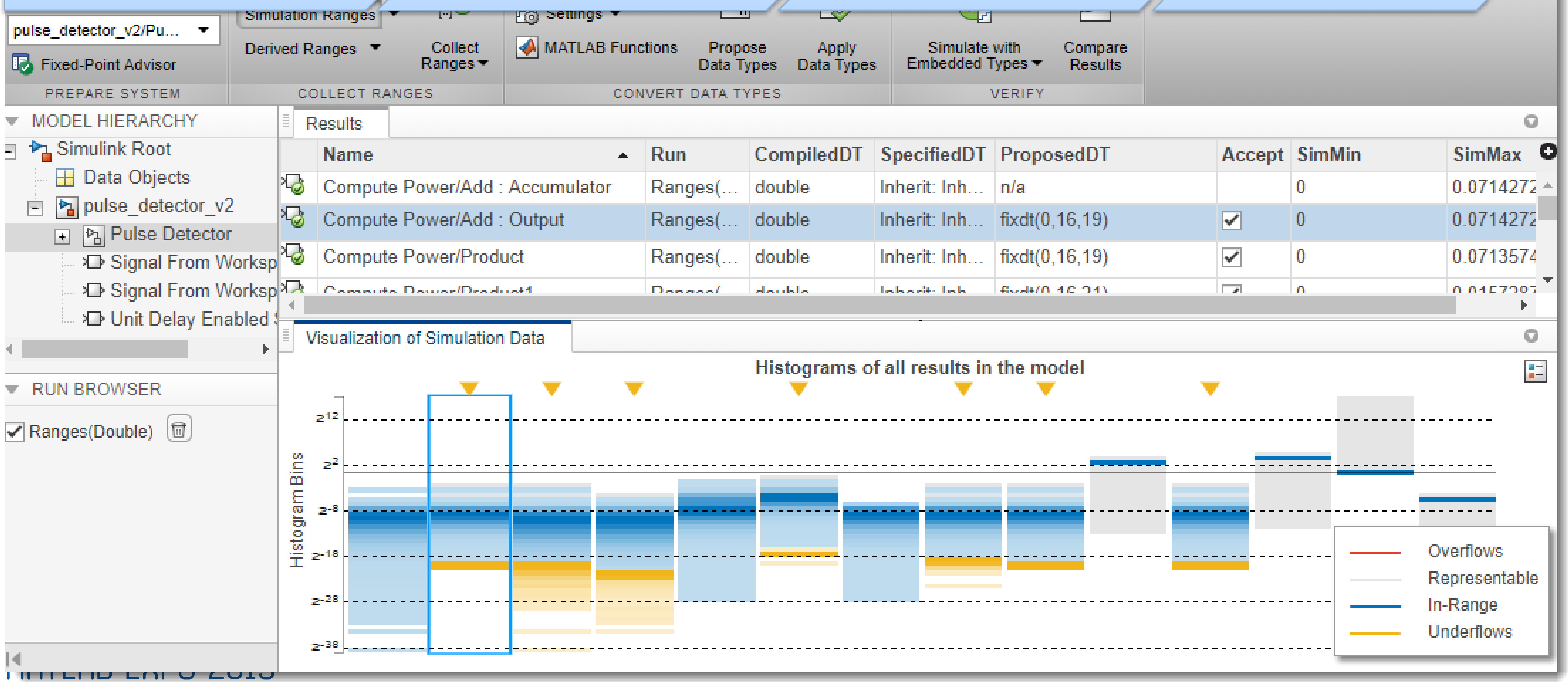
OK | Cancel | Help | Apply

Fixed-Point Streaming Algorithms: Manual Approach



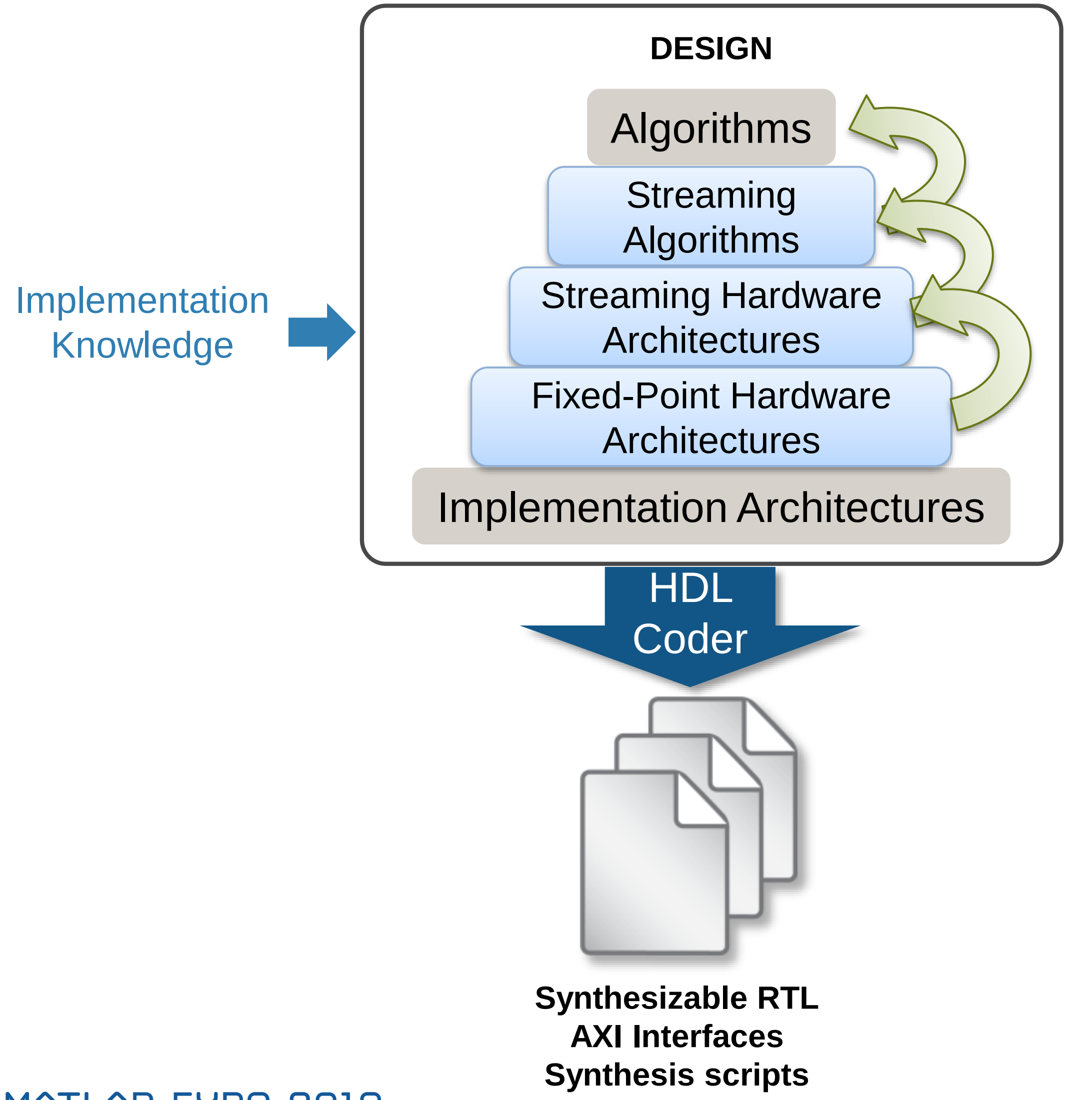
Fixed-Point Streaming Algorithms: Automated Approach

Simulate with representative data to collect required ranges → Fixed-Point Designer proposes data types → Choose to apply proposed types or set your own → Simulate and compare results



Name	Run	CompiledDT	SpecifiedDT	ProposedDT	Accept	SimMin	SimMax
Compute Power/Add : Accumulator	Ranges(...	double	Inherit: Inh...	n/a		0	0.0714272
Compute Power/Add : Output	Ranges(...	double	Inherit: Inh...	fixdt(0,16,19)	✓	0	0.0714272
Compute Power/Product	Ranges(...	double	Inherit: Inh...	fixdt(0,16,19)	✓	0	0.0713574

Automatically Generate Production RTL

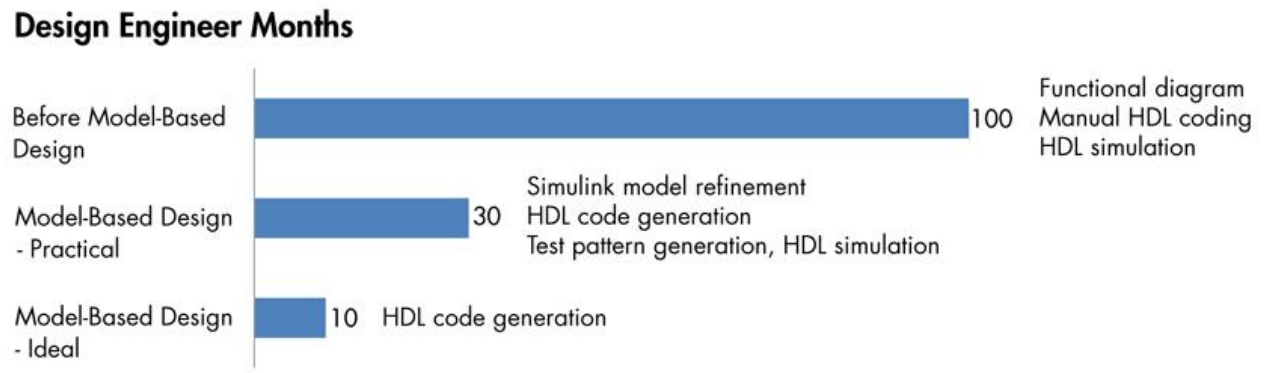


- Choose from over 250 supported blocks
  - Including MATLAB functions and Stateflow charts
- Quickly explore implementation options
  - Micro-architectures
  - Pipelining
  - Resource sharing
  - Fixed-point or native floating point
- Generate readable, traceable Verilog/VHDL
  - Optionally generate AXI interfaces with IP core
- Quickly adapt to changes and re-generate
- Production-proven across a variety of applications and FPGA, ASIC, and SoC targets

Agenda

- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer results

Model-Based Design Results for Communications System Development at Hitachi



Solution

Adopted Model-Based Design to enable teams to verify the specification via a model in a shared simulation environment. The system design and FPGA design teams use the model as an executable specification. The model is refined and elaborated throughout the design process, and HDL code is automatically generated for logic synthesis and implementation.

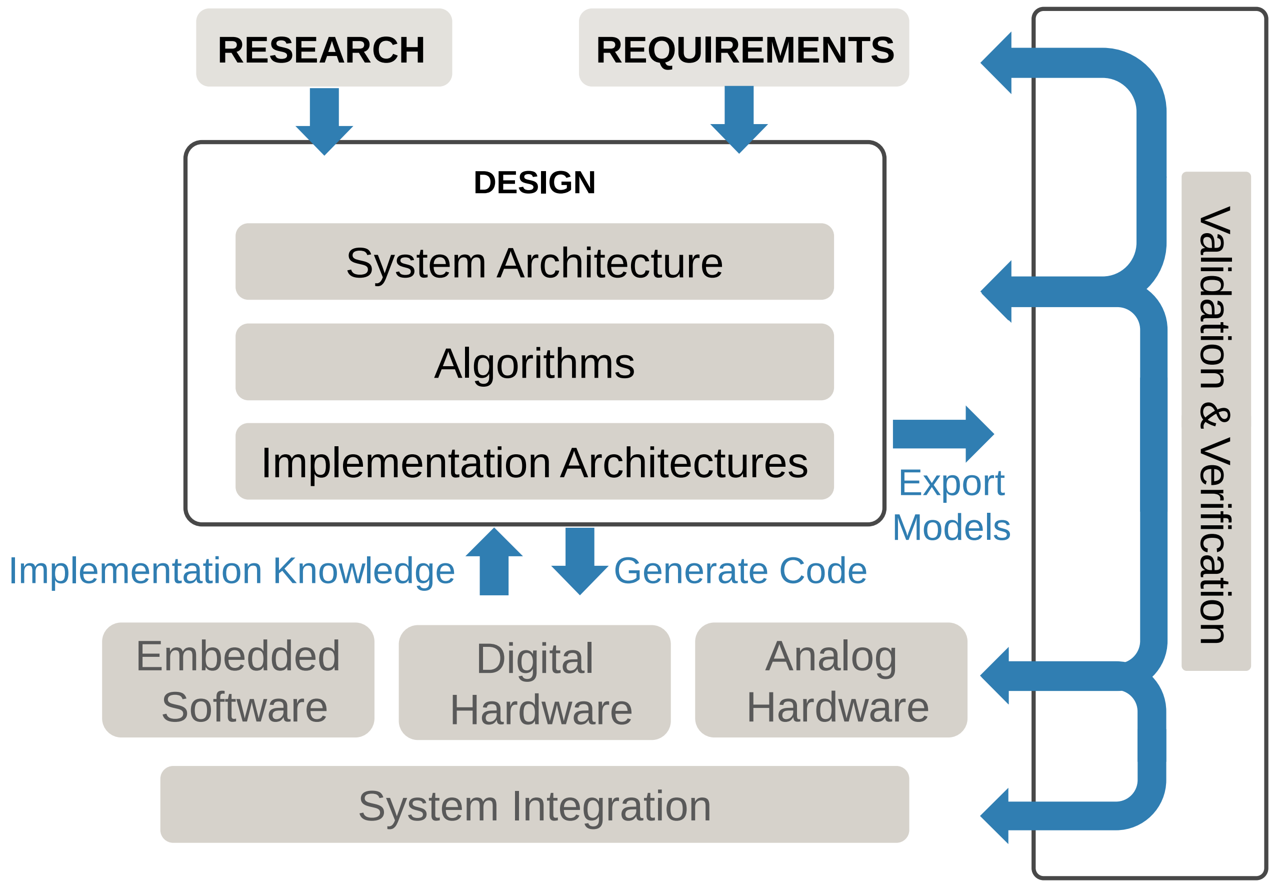
Results

- 70% effort reduction for design projects
- Nearly equivalent FPGA performance, power, and resource usage
- Adopted across more than 10 product development projects

"We have adopted Model-Based Design with MATLAB® and Simulink® as our standard development workflow for FPGA design. As a result, we have improved communication between teams, reduced development time, and reduced risk by evaluating system performance early in the design process."

Link to article

Getting Started Collaborating with Model-Based Design



- ❑ Refine algorithm toward implementation
- ❑ Verify refinements versus previous versions
- ❑ Generate verification models
- ❑ Add hardware implementation detail and generate optimized RTL
- ❑ Simulate System-on-Chip architecture

- ➢ Eliminate communication gaps
- ➢ Key decisions made via cross-skill collaboration
- ➢ Identify and address system-level issues before implementing subsystems
- ➢ Adapt to changing requirements with agility

MATLAB EXPO 2019

What's New!

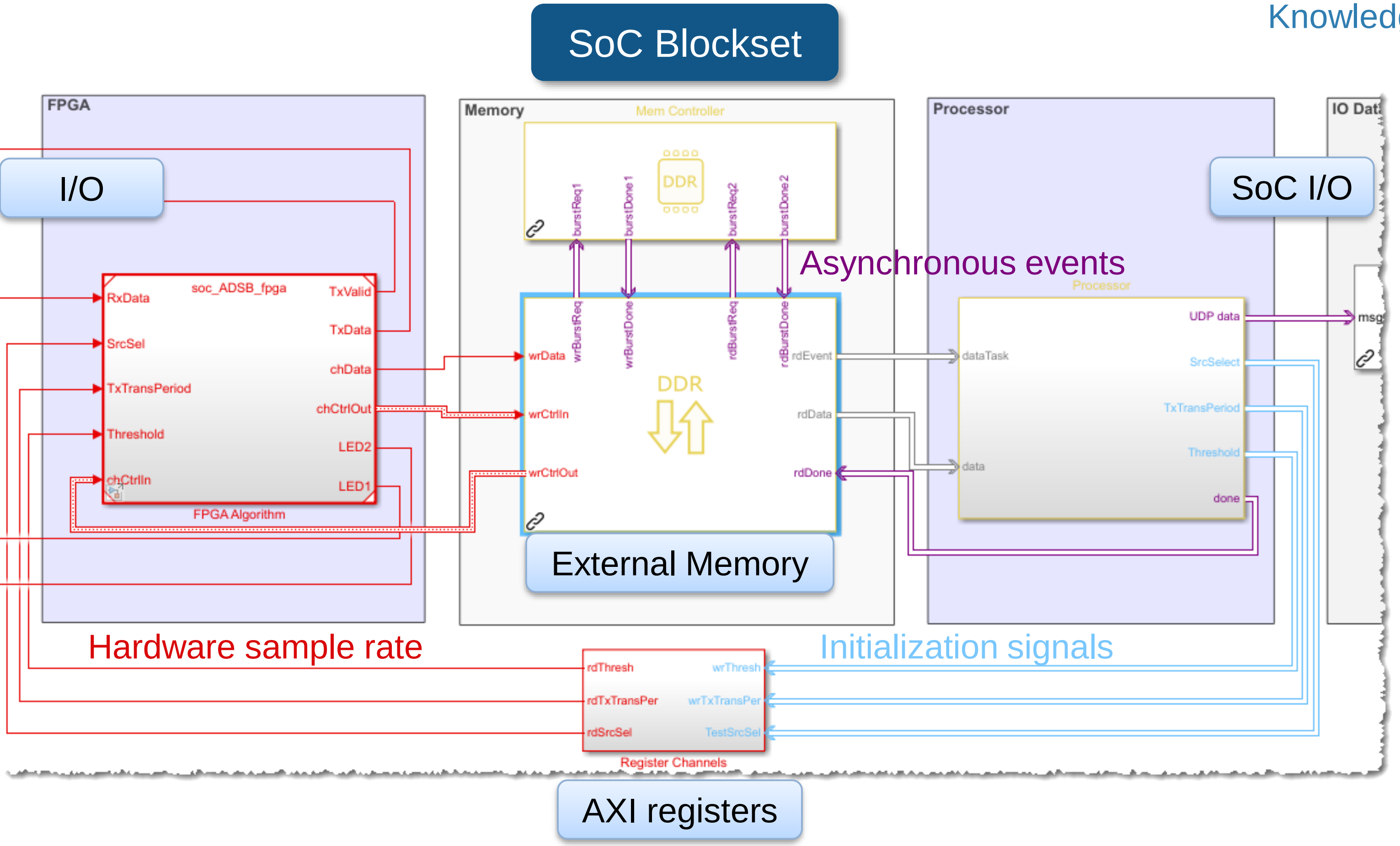
R2019a

Model and Simulate SoC Architecture

DESIGN

- System Architecture
- Algorithms
- Implementation Architectures

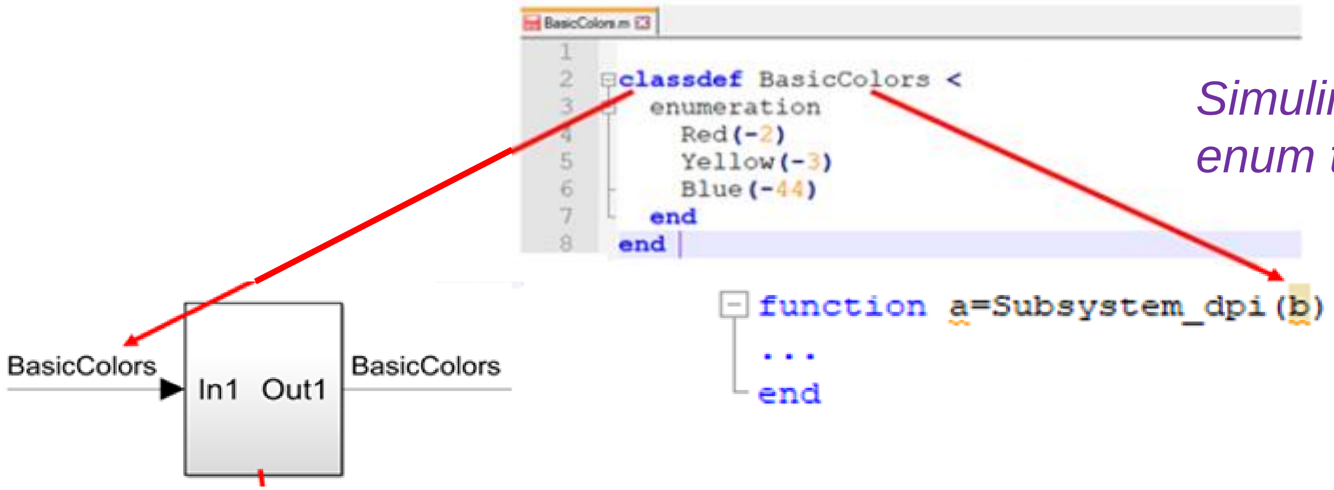
Implementation Knowledge



- Simulate behavior and latency
  - Algorithm, memory, internal and external connectivity
  - Scheduling and OS effects
  - Real streaming I/O data
- Diagnose software performance and hardware utilization
- Adjust core algorithms so they work in the actual hardware context

Enhanced Data Type Support for SystemVerilog DPI Generation: Enum

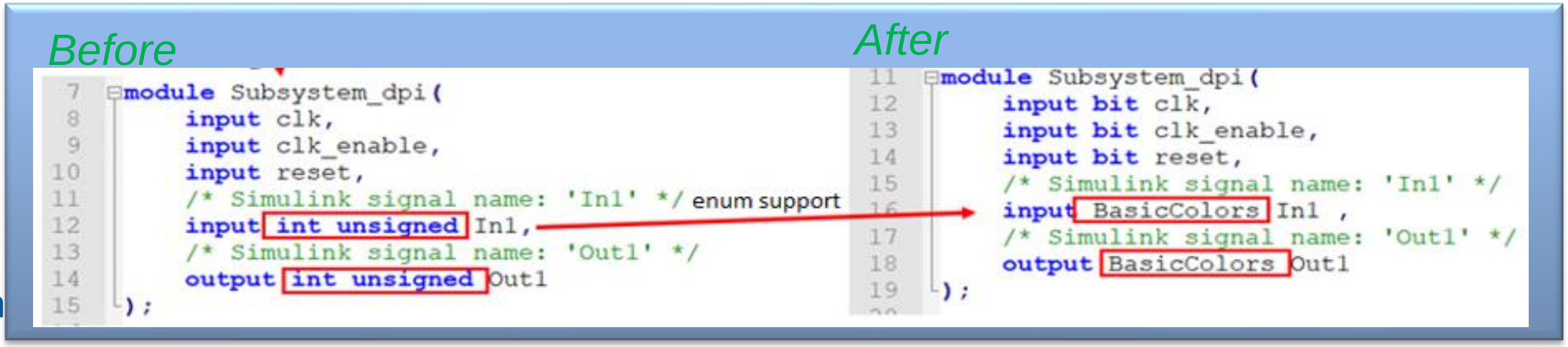
DPI feature will generate SystemVerilog enum from Simulink and MATLAB enum data types on interface.



```
classdef BasicColors <
    enumeration
        Red(-2)
        Yellow(-3)
        Blue(-44)
    end
end
```

```
function a=Subsystem_dpi(b)
...
end
```

Simulink or MATLAB enum type



Before

```
module Subsystem_dpi(
        input clk,
        input clk_enable,
        input reset,
        /* Simulink signal name: 'In1' */
        input int unsigned In1,
        /* Simulink signal name: 'Out1' */
        output int unsigned Out1
);
```

enum support

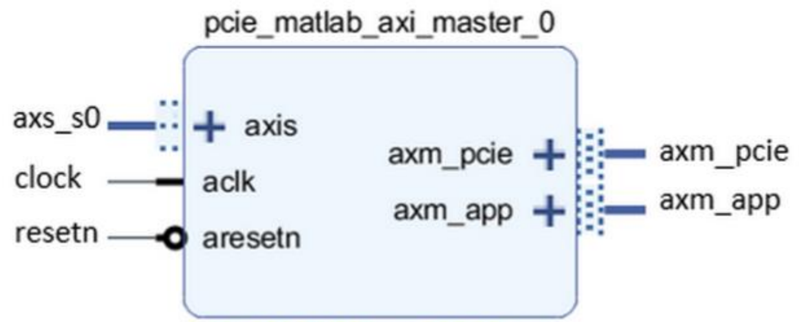
After

```
module Subsystem_dpi(
        input bit clk,
        input bit clk_enable,
        input bit reset,
        /* Simulink signal name: 'In1' */
        input BasicColors In1 ,
        /* Simulink signal name: 'Out1' */
        output BasicColors Out1
);
```

PCI Express based MATLAB as AXI Master for Xilinx FPGA Boards

Provide HDL IP core for Xilinx Vivado designs to support AXI4 read and write operations over a PCI Express connection.

- Provides faster performance than JTAG and Ethernet based MATLAB as AXI Master
- Not board specific.
- Demo available:
  - *copyXilinxFPGAExampleFiles('pcieaximaster')*
- To use on host:
  - mem = aximaster('Xilinx', 'Interface', 'PCIe');
  - writememory(mem, 'C000000', *wrdata*);
  - rddata = readmemory(mem, 'C00000000', 'uint8');



PCIe MATLAB as AXI Master

Learn More

- Next steps to get started with:
  - Verification: Improve RTL Verification by Connecting to MATLAB webinar
  - Fixed-point quantization: Fixed-Point Made Easy webinar
  - Incremental refinement, HDL code generation: HDL self-guided tutorial

- Visit us at the demo booth for detailed update on SoC block set and HDL Verifier workflow.

Training Services

Exploit the full potential of MathWorks products

Flexible delivery options:

- Public training available in several cities
- Onsite training with standard or customized courses
- Web-based training with live, interactive instructor-led courses

More than 48 course offerings:

- Introductory and intermediate training on MATLAB, Simulink, Stateflow, code generation, and Polyspace products
- Specialized courses in control design, signal processing, parallel computing, code generation, communications, financial analysis, and other areas

www.mathworks.in/training

MathWorks | *Training Services*

Generating HDL Code from Simulink

This two-day course shows how to generate and verify HDL code from a Simulink® model using HDL Coder™ and HDL Verifier™

Topics include:

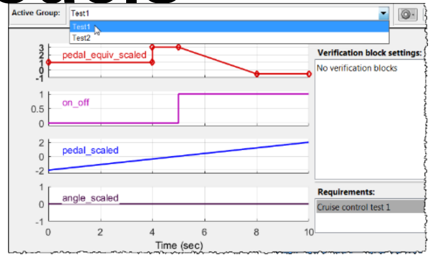
- Preparing Simulink models for HDL code generation
- Generating HDL code and testbench for a compatible Simulink model
- Performing speed and area optimizations
- Integrating handwritten code and existing IP
- Verifying generated HDL code using testbench and cosimulation

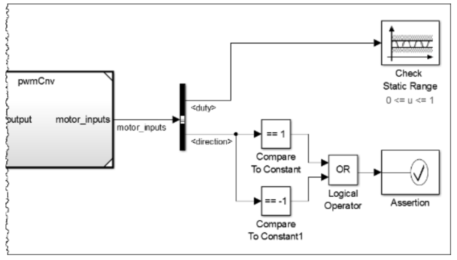
MathWorks | *Training Services*

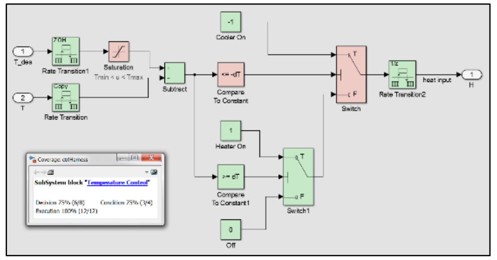
Verification and Validation of Simulink Models

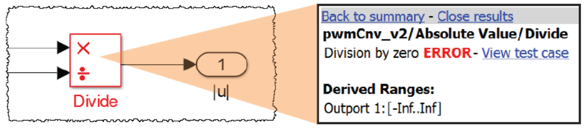
After this 1-day course you will be able to:

- Verify models using simulation test cases.
- Verify models using formal methods.
- Automate the execution and documentation of test suites.



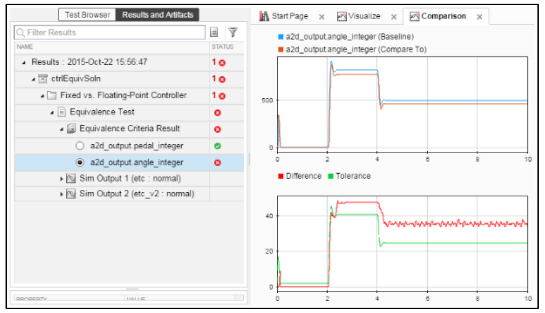






Back to summary - Close results
pwmCnv_v2/Absolute Value/Divide
Division by zero **ERROR** - View test case

Derived Ranges:
Outport 1:[-Inf..Inf]



Results : 2015-Oct-22 15:56:47
ctrlEquivSoln
Fixed vs. Floating-Point Controller
Equivalence Test
Equivalence Criteria Result
a2d_output.pedal_integer
a2d_output.angle_integer
Sim Output 1 (etc : normal)
Sim Output 2 (etc_v2 : normal)

MathWorks®

Accelerating the pace of engineering and science

Speaker Details

Email: hitu.sharma@mathworks.in

Lined in :https://www.linkedin.com/in/hitu-sharma-99095218/

Contact MathWorks India

Call: 080-6632-6000

Email: info@mathworks.in

Your feedback is valued.

Please complete the feedback form provided to you.