MATLAB EXPO 2019

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

Robert Anderson
Principal Application Engineer - MathWorks

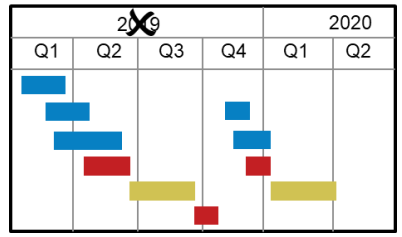
Agenda

- 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

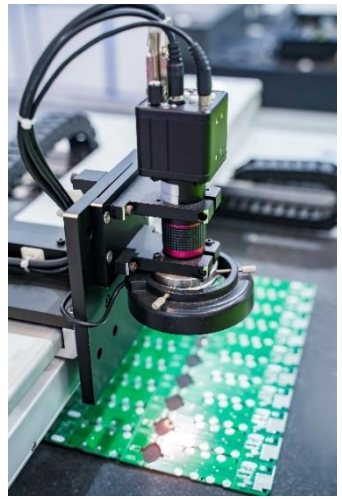
FPGA, ASIC, and SoC Development Projects

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

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



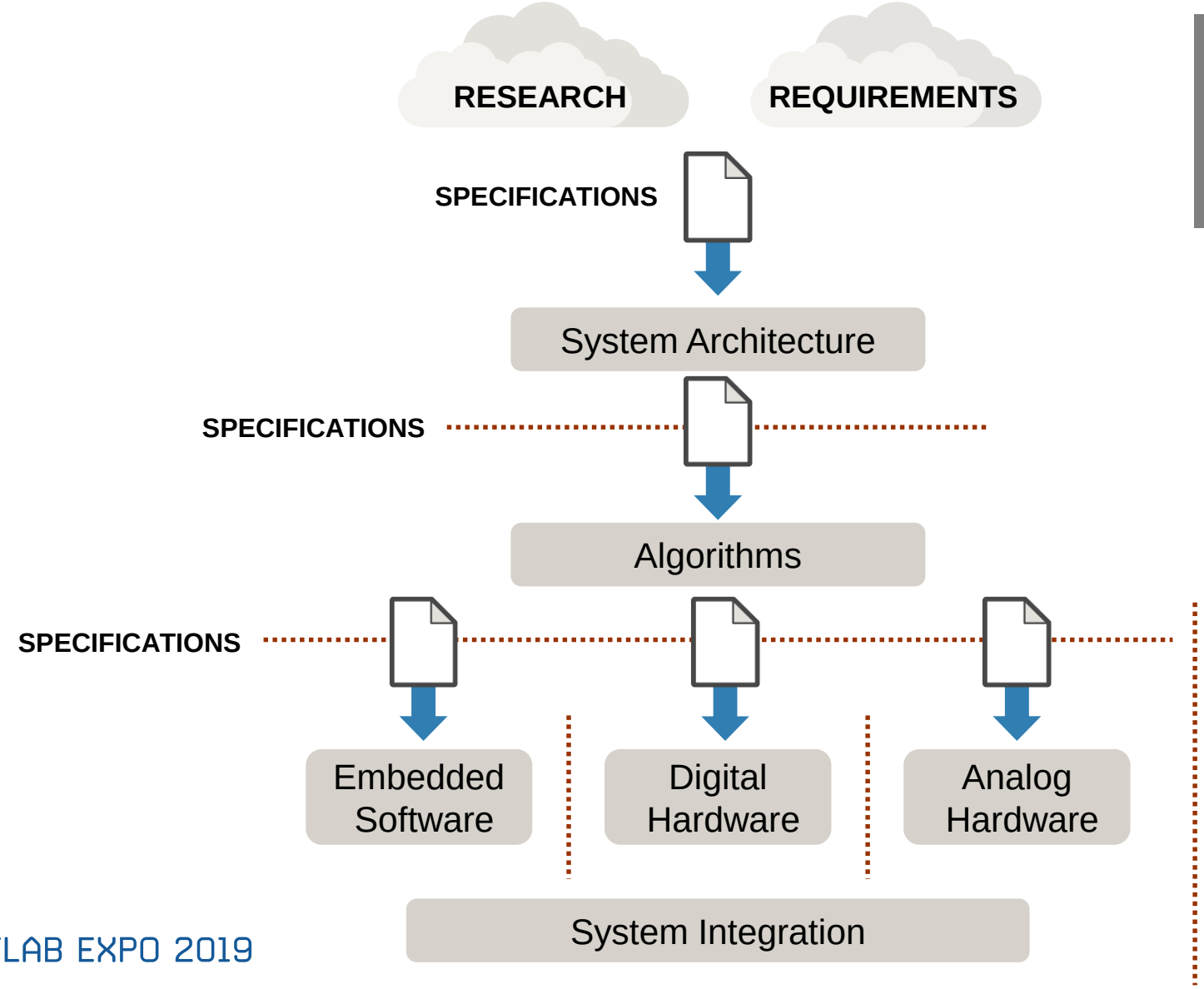
75% of ASIC projects require a **silicon re-spin**



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

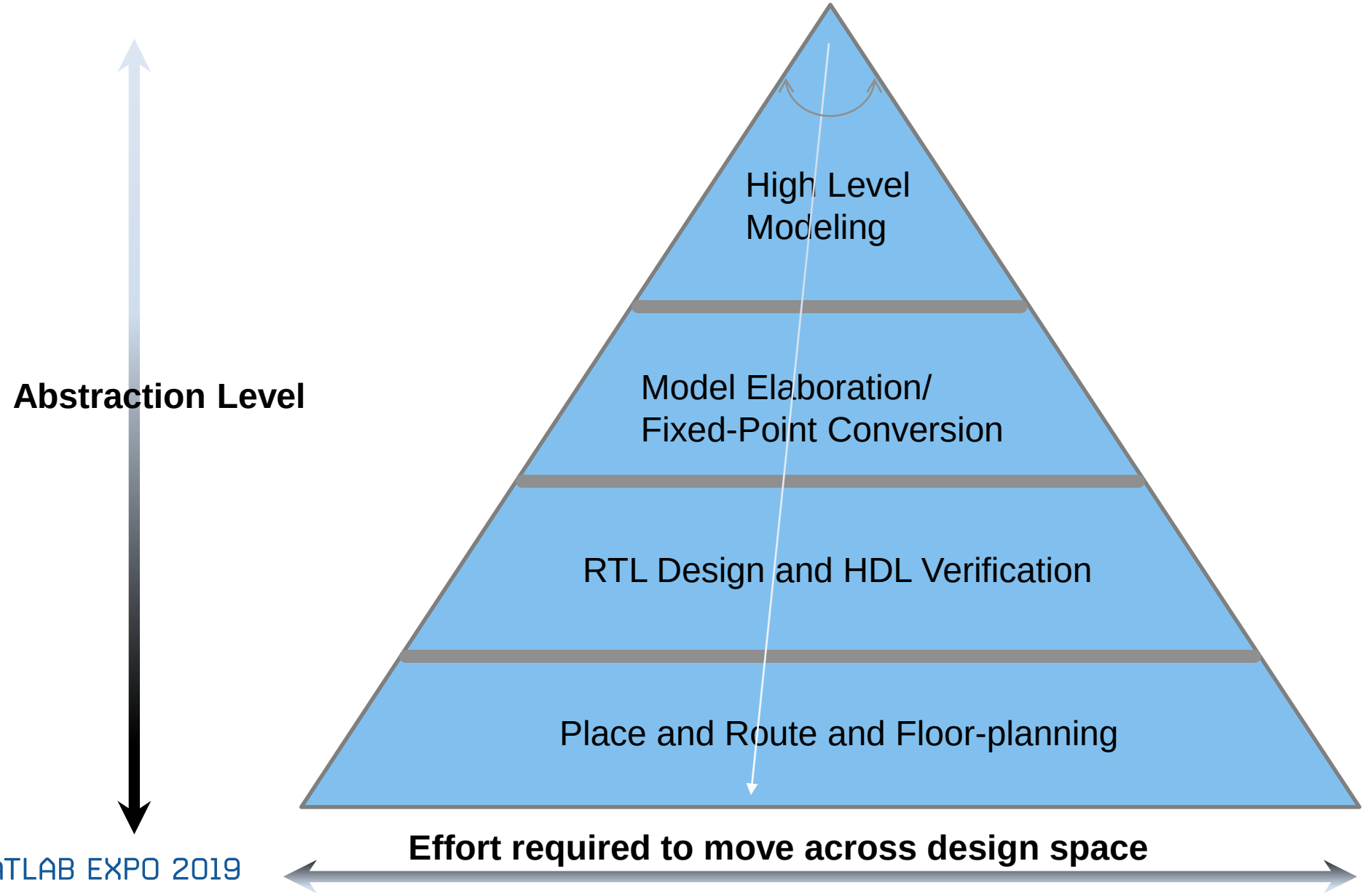


- Poor communication across teams
- Key decisions made in silos
- System-level issues found in late stages
- Hard to adapt to changing requirements

"Rapid innovation under a rapid timeline – that's when this flow falls apart."

Jamie Haas
Allegro Microsystems

Abstraction vs Design Space Exploration



Cost of Finding a Bug vs Location in Design Cycle

Location in Design Cycle

Requirements

High Level Modeling/Verification

Model Elaboration/Fixed-Point Conversion

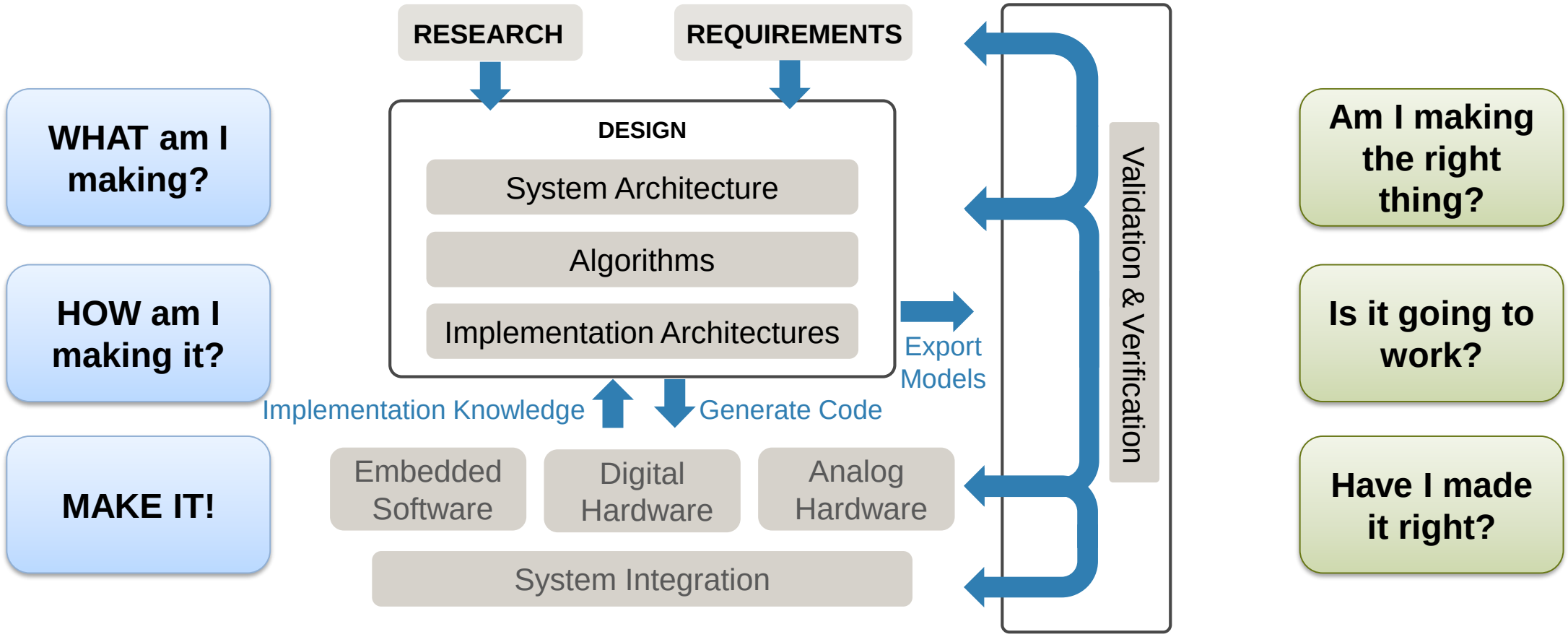
RTL Design and HDL Verification

Place and Route/Floor-planning

Integration and Validation Test

Post – Production/Product Launch

SoC Collaboration with Model-Based Design



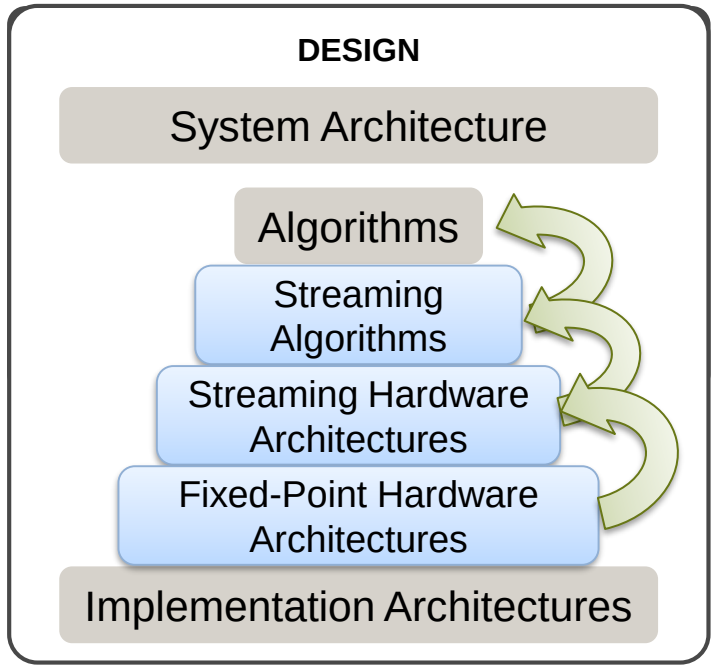
Agenda

- 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



Simulink

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

Bit Accurate

Cycle Accurate

Partition Hardware-Targeted Design, System Context, Testbench

Algorithm Stimulus

Hardware Algorithm

Software Algorithm

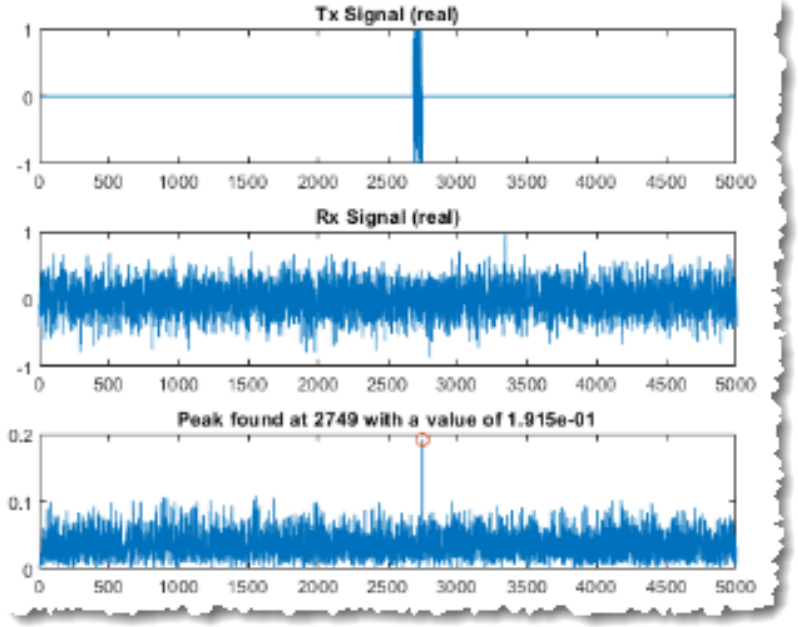
Analysis

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));
```



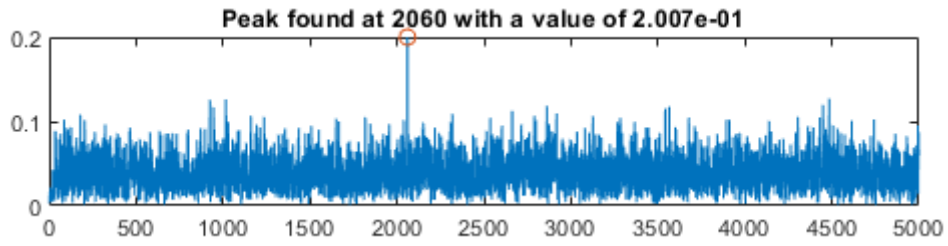
Streaming Algorithms: MATLAB or Simulink…or Both

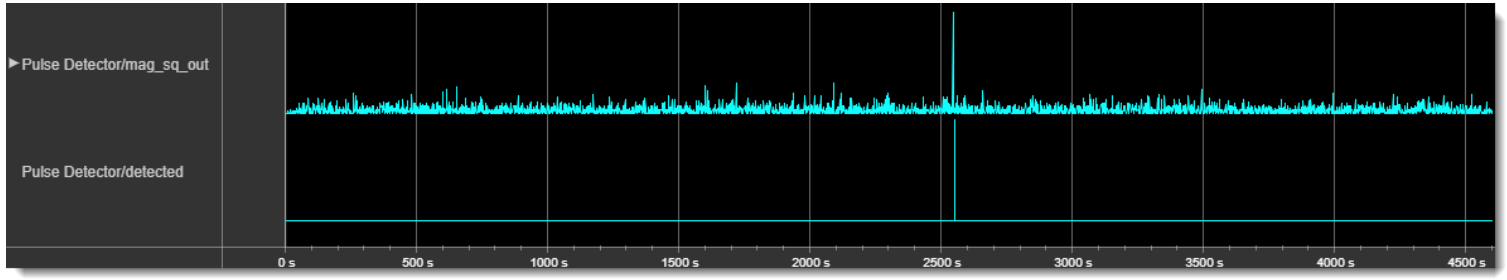
Hardware friendly implementation of peak finder

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:

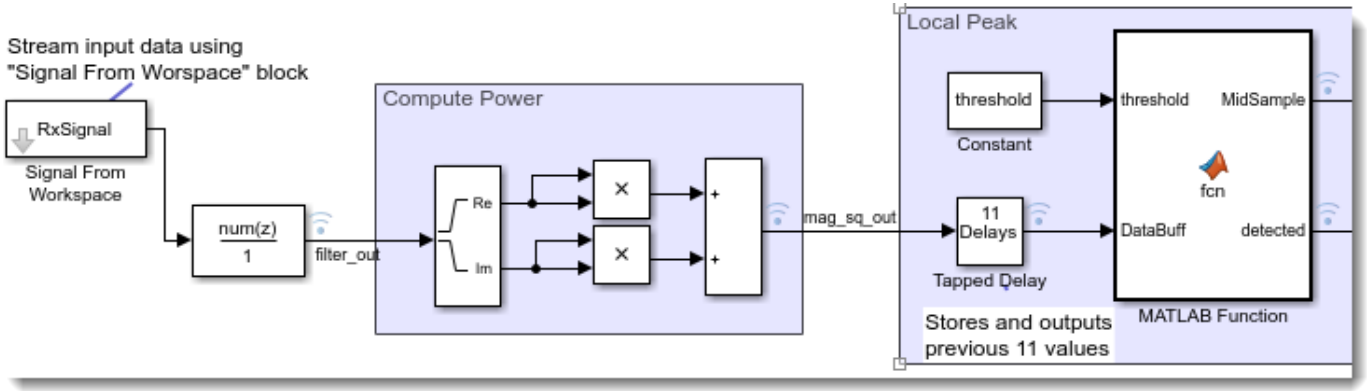
- The middle sample is the largest
- The middle sample is greater than a pre-defined threshold

```
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
```

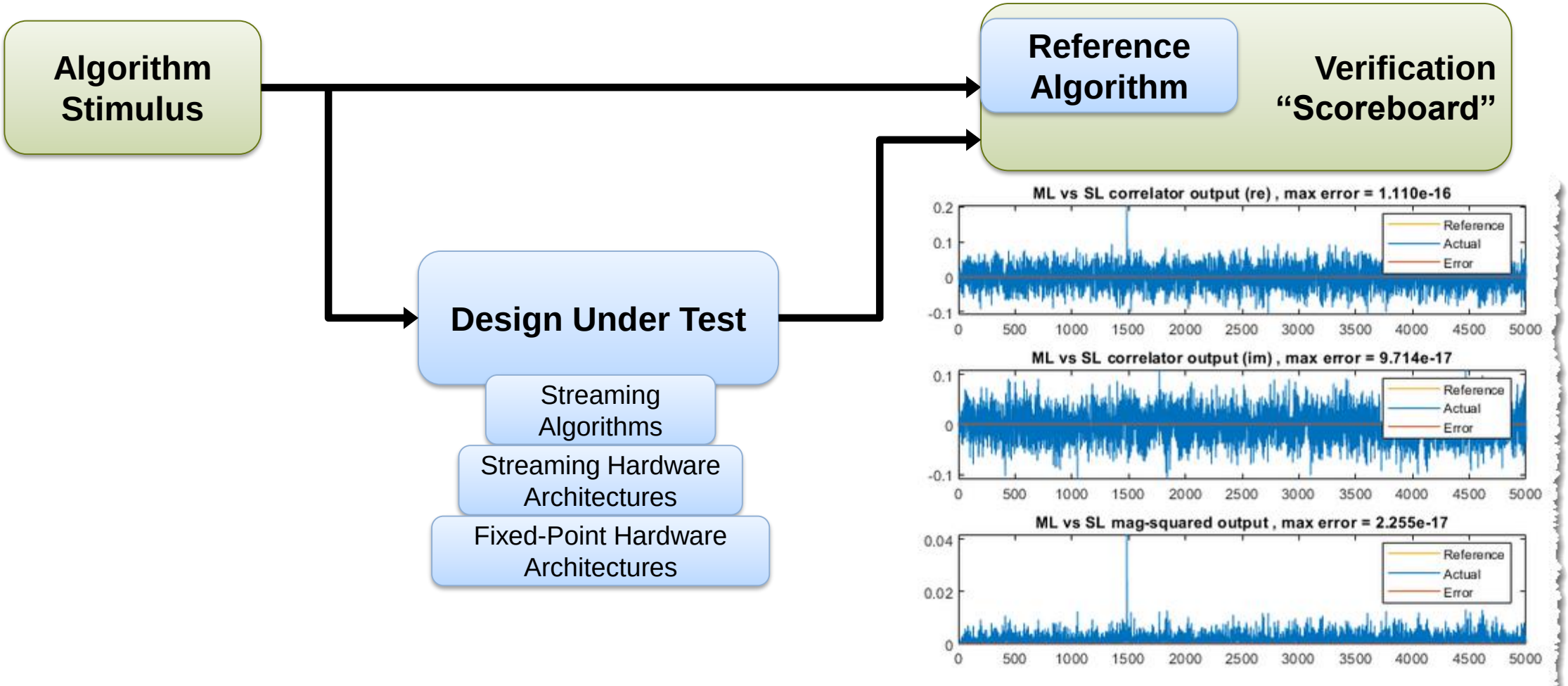





```
% Simulate model
sim('pulse_detector_v1')

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



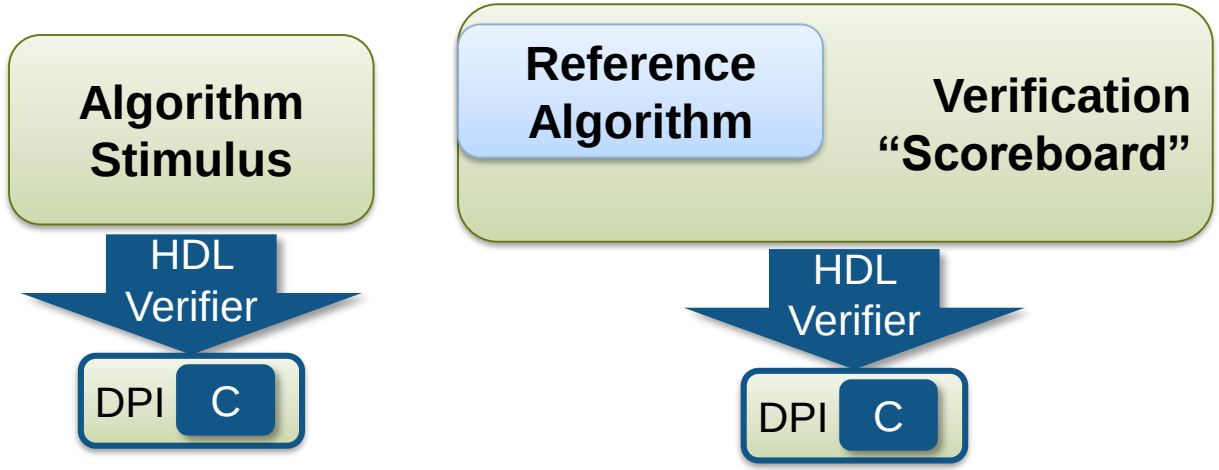
Refine Algorithm and Verify Against Golden Reference



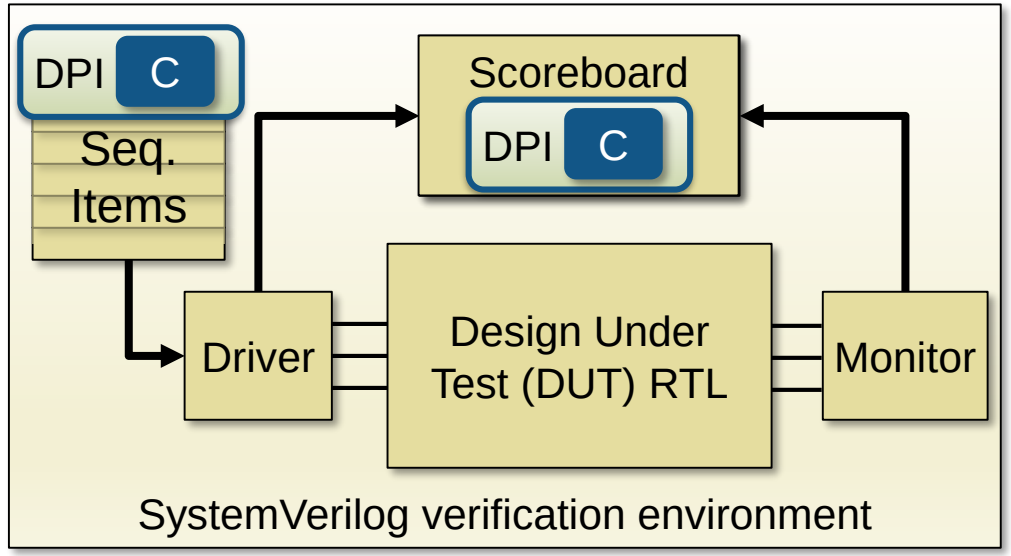

```
Peak location = 1485, magnitude = 2.044e-01 using global max
Peak location = 1485, mag-squared = 4.178e-02 using local max
Peak mag-squared from Simulink = 4.178e-02, error = 2.082e-17
```

Self-checking

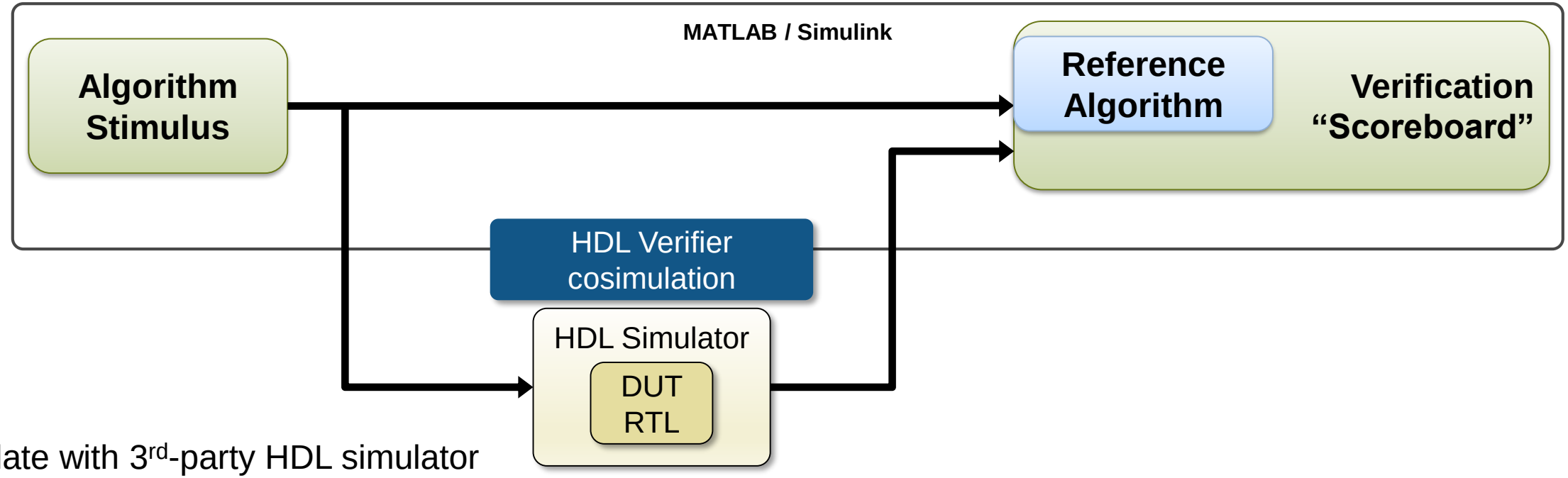
Generate SystemVerilog DPI Components for RTL Verification



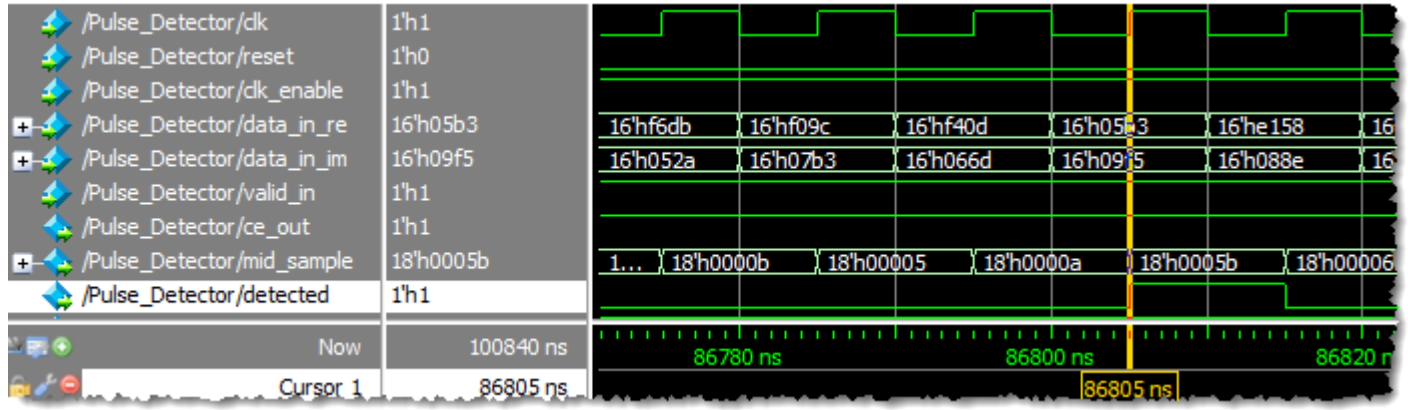
- Reuse MATLAB/Simulink models in verification
  - Scoreboard, stimulus, or models external to the RTL
    - Generate from frame-based or streaming algorithm
    - Floating-point or fixed-point
    - Individual components or entire testbench
  - Runs natively in SystemVerilog simulator
  - Eliminate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



What if there's a mismatch?



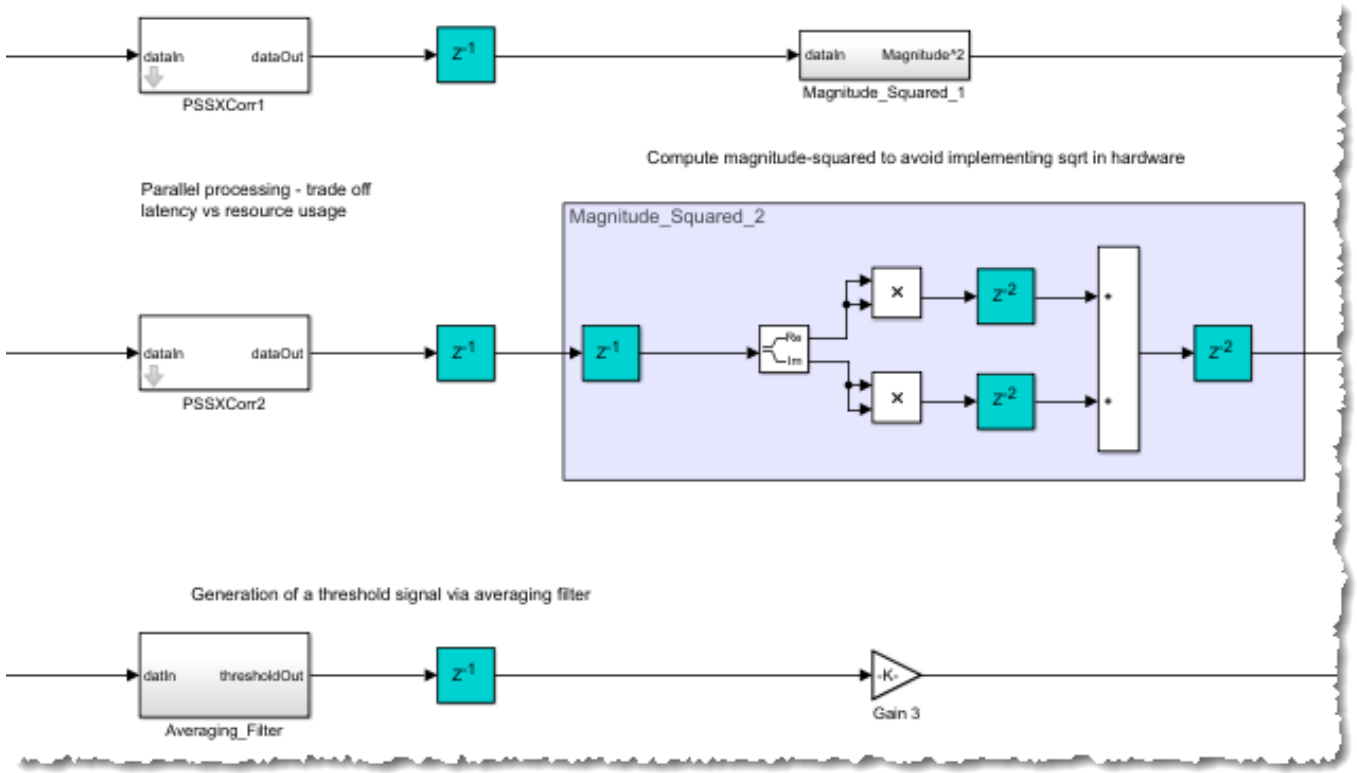
- Co-simulate with 3rd-party HDL simulator
  - Reuse MATLAB/Simulink test environment
  - Run HDL design in a supported simulator*
  - Generate co-simulation infrastructure and handshaking
  - Analyze both the design and test environment



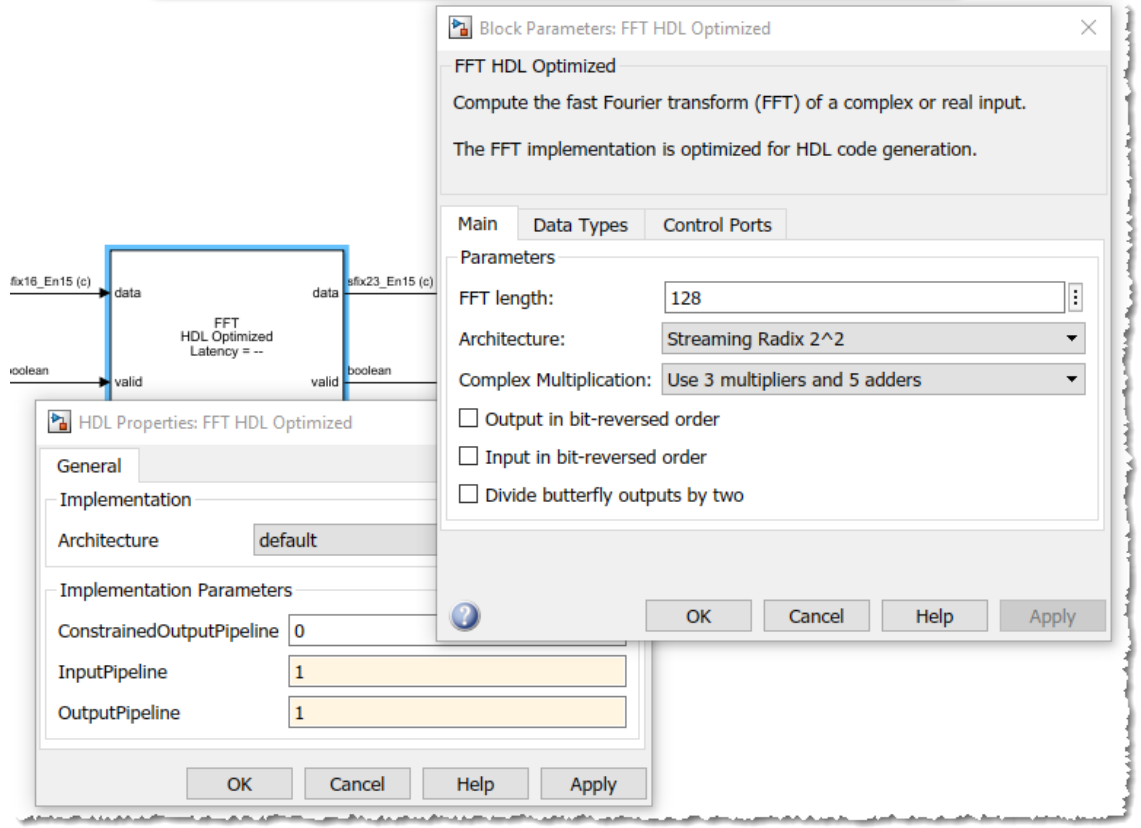
* Mentor Graphics® ModelSim® or Questa®
Cadence® Incisive® or Xcelium™

Collaborate to Add Hardware Architecture

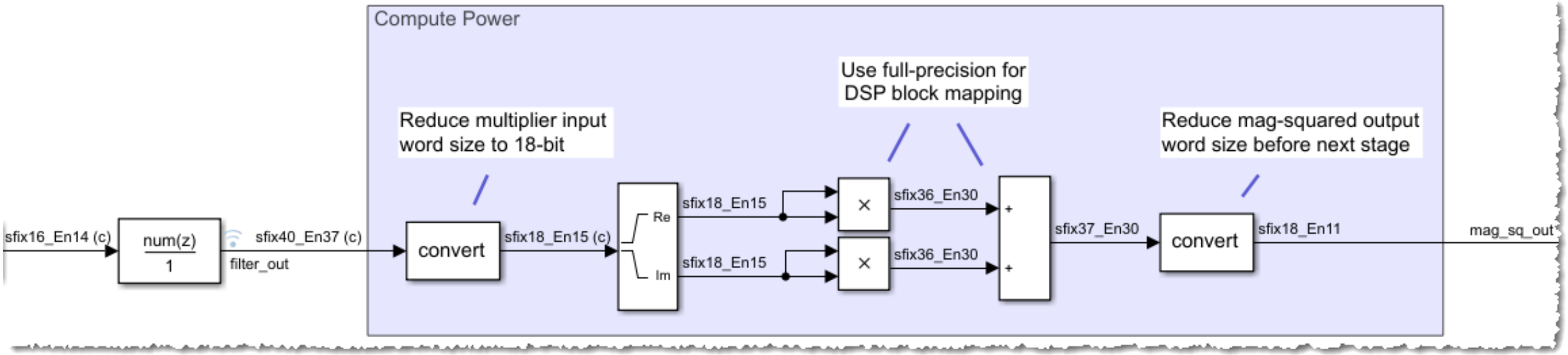
Optimize architecture design for hardware goals



Specify HDL implementation options

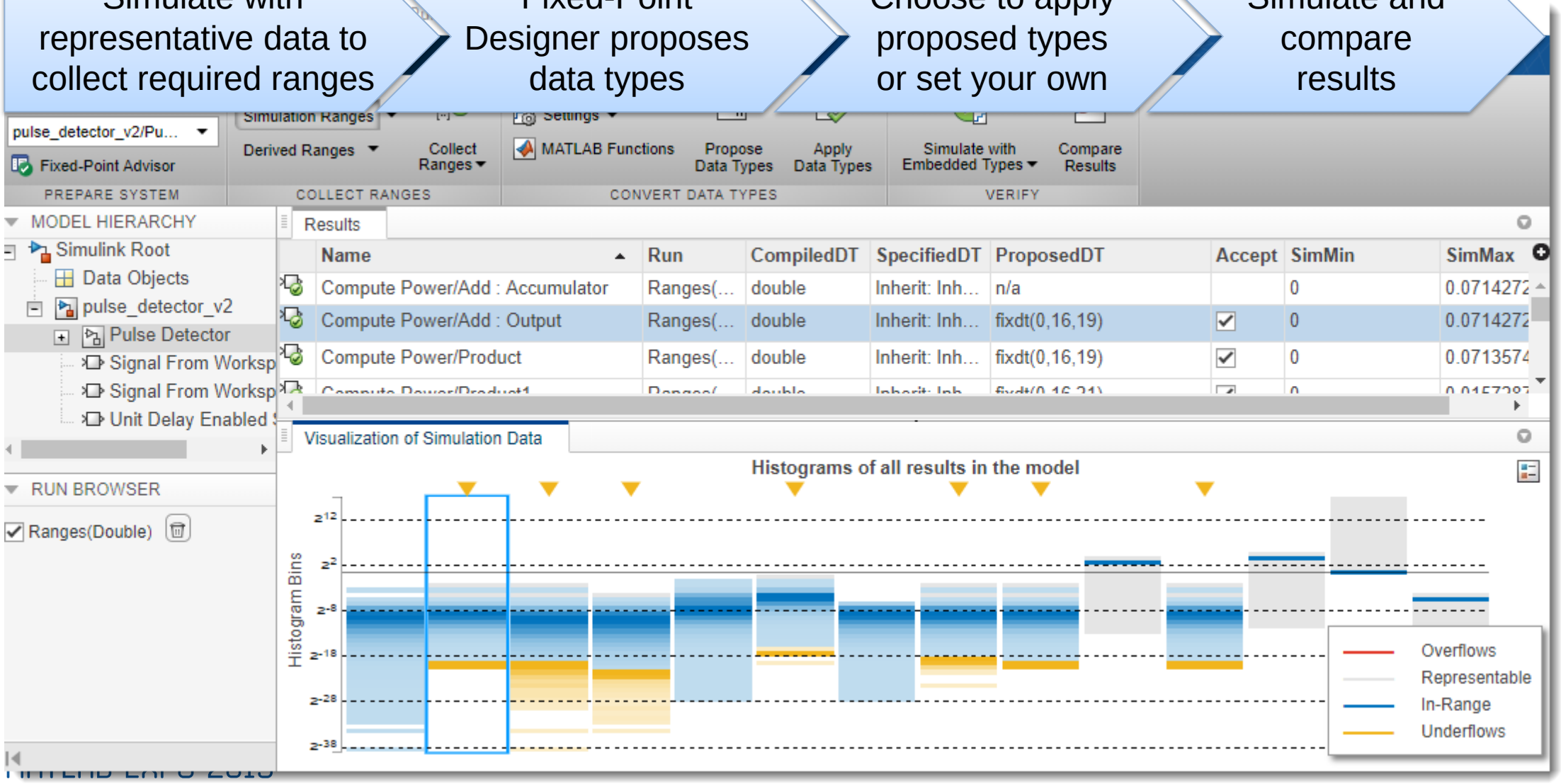


Fixed-Point Streaming Algorithms: Manual Approach

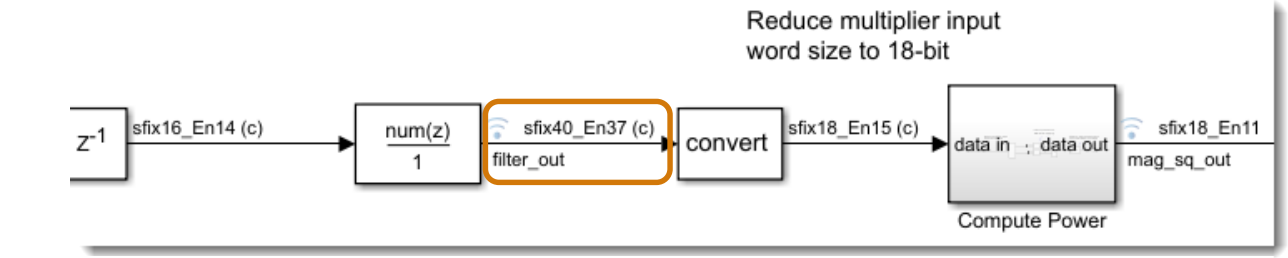


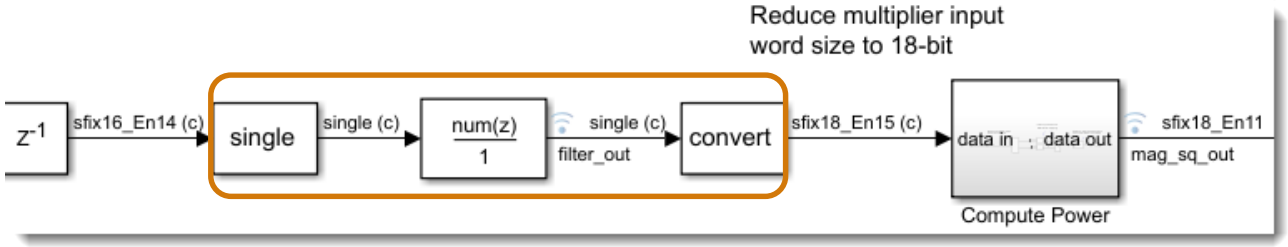
Fixed-Point Streaming Algorithms: Automated Approach

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



Generating Native Floating Point Hardware

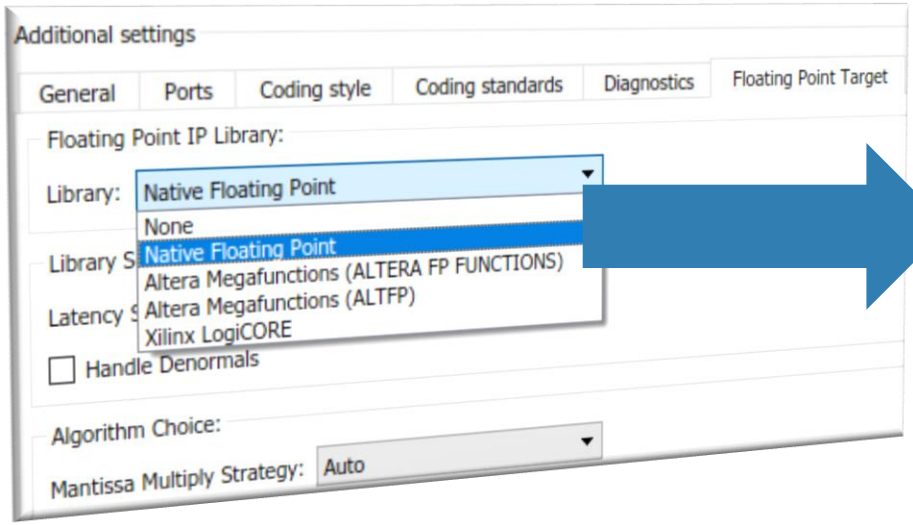


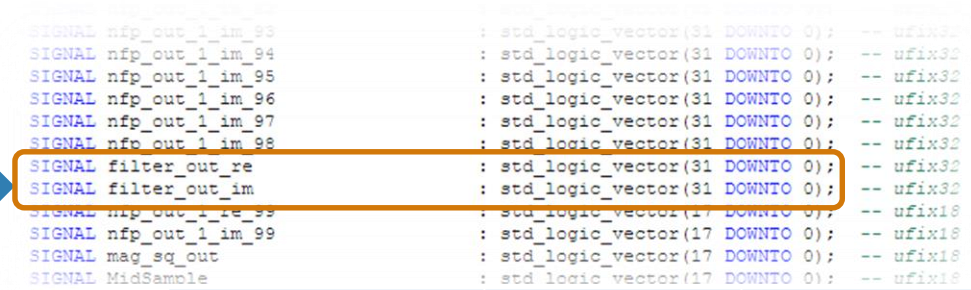


HDL Coder Native Floating Point

- Extensive math and trigonometric operator support
- Optimal implementations without sacrificing numerical accuracy
- Mix floating- and fixed-point operations
- Generate target-independent HDL

embeddedaward2017

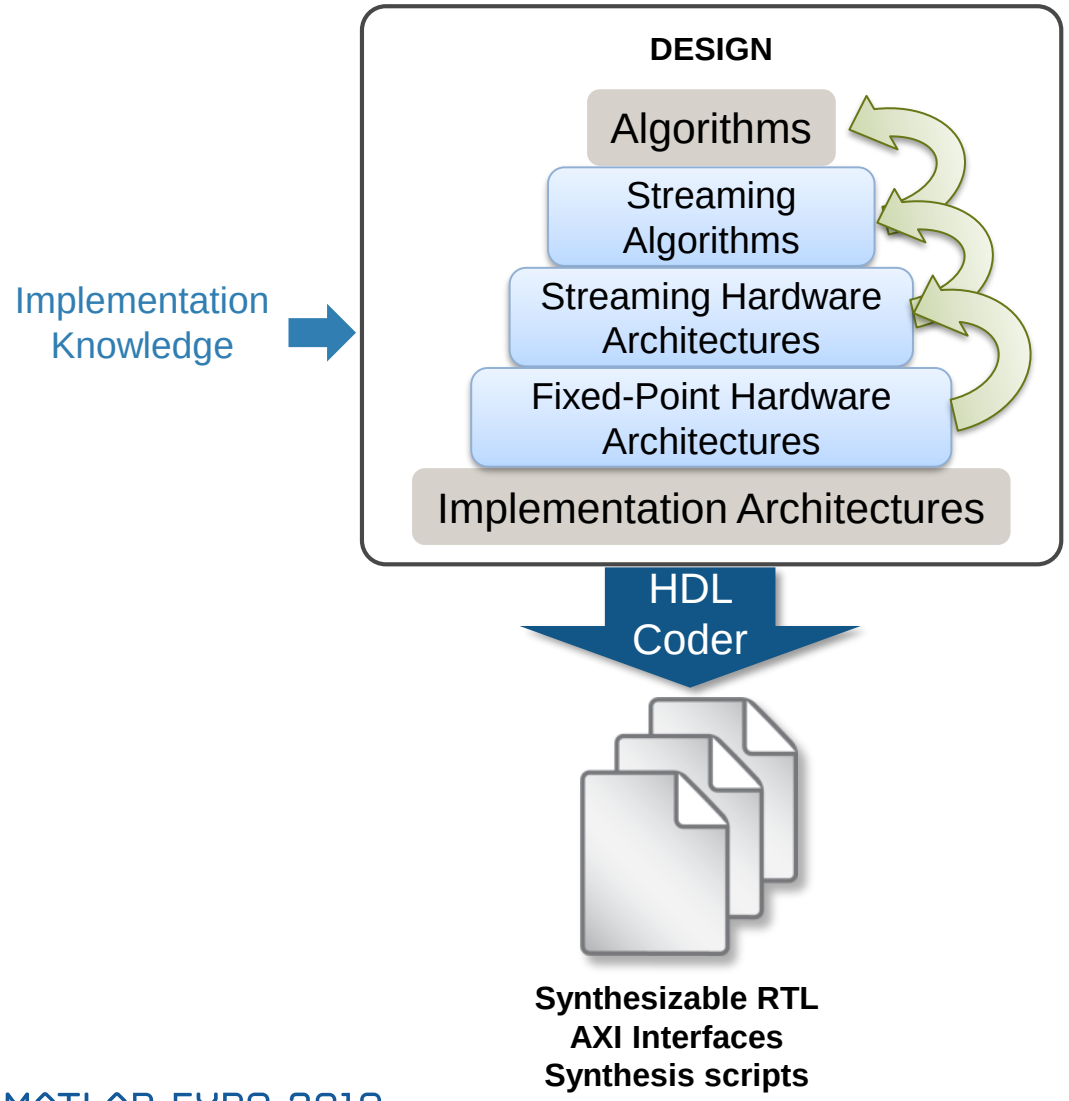




	Fixed point	Floating point
LUTs	10k	25k
DSP slices	50	100
Development time	~1 week	~1 day

~2x more resources
~5x less development effort

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

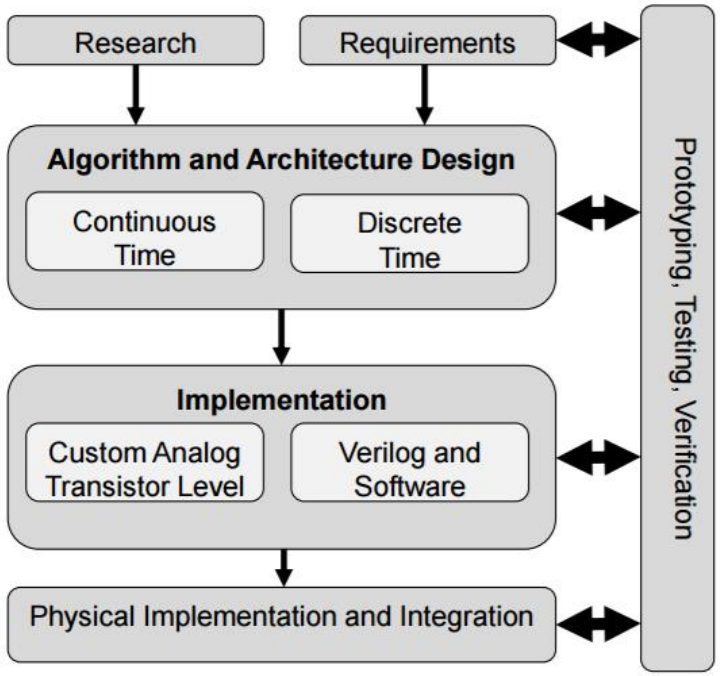
- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer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

Results at Allegro Microsystems

Allegro MicroSystems, LLC

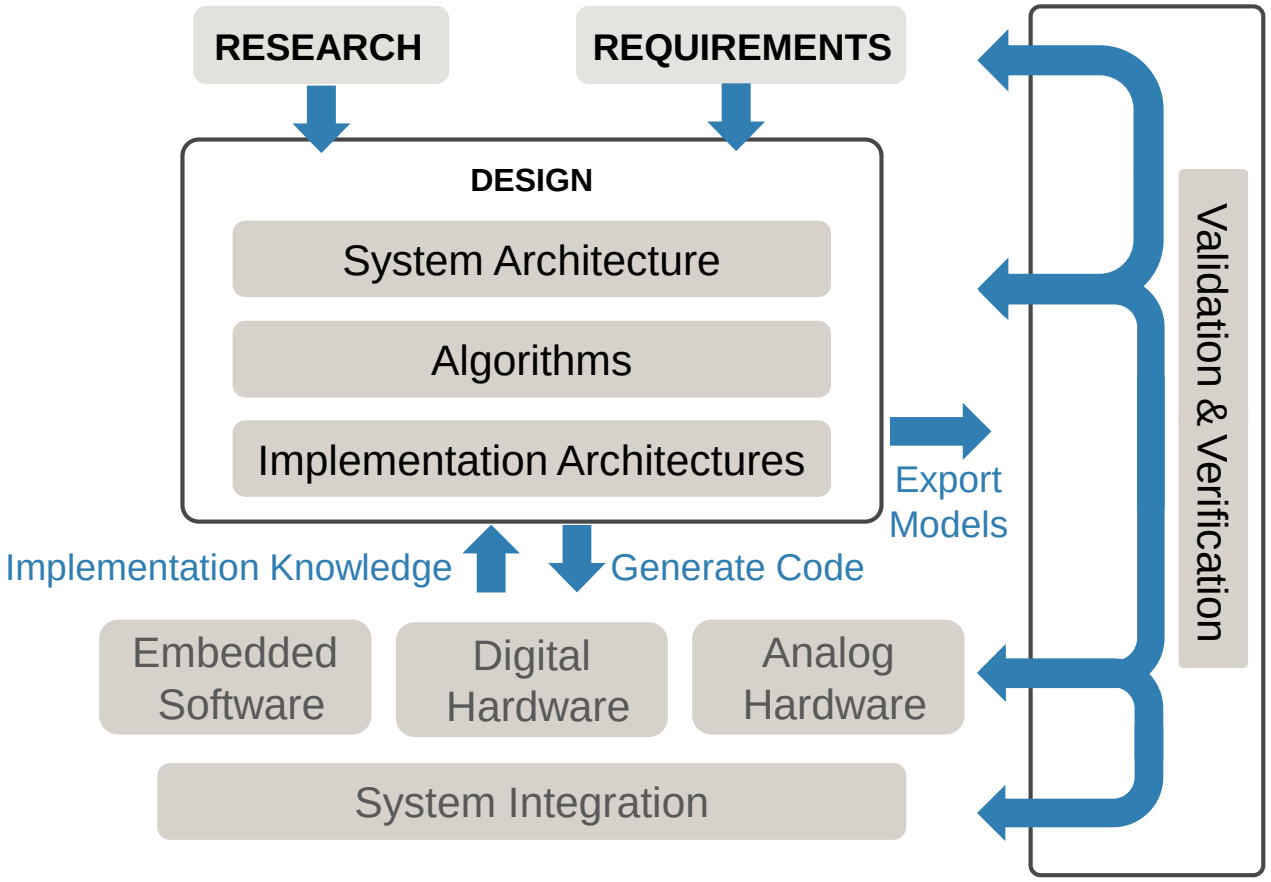
Allegro Confidential Information

The Enlightenment: Model Based Design



- **Architecture and Algorithm Design Evolve into Executable Specifications**
- **Front load testing and verification**
- **Development is "parallelized"**
- **Continuous Equivalency Testing is utilized**
- **.... And of course auto-generated production code**

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

Learn More

- Next steps to get started:
  - 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
  - https://www.mathworks.com/solutions/fpga-asic-soc-development/asic.html
- Technology showcase here at MATLAB EXPO
- MathWorks Advisory Board (MAB)
- Pilots and Consulting services to help you get on-board
- Contact your local sales representative for hands-on workshops