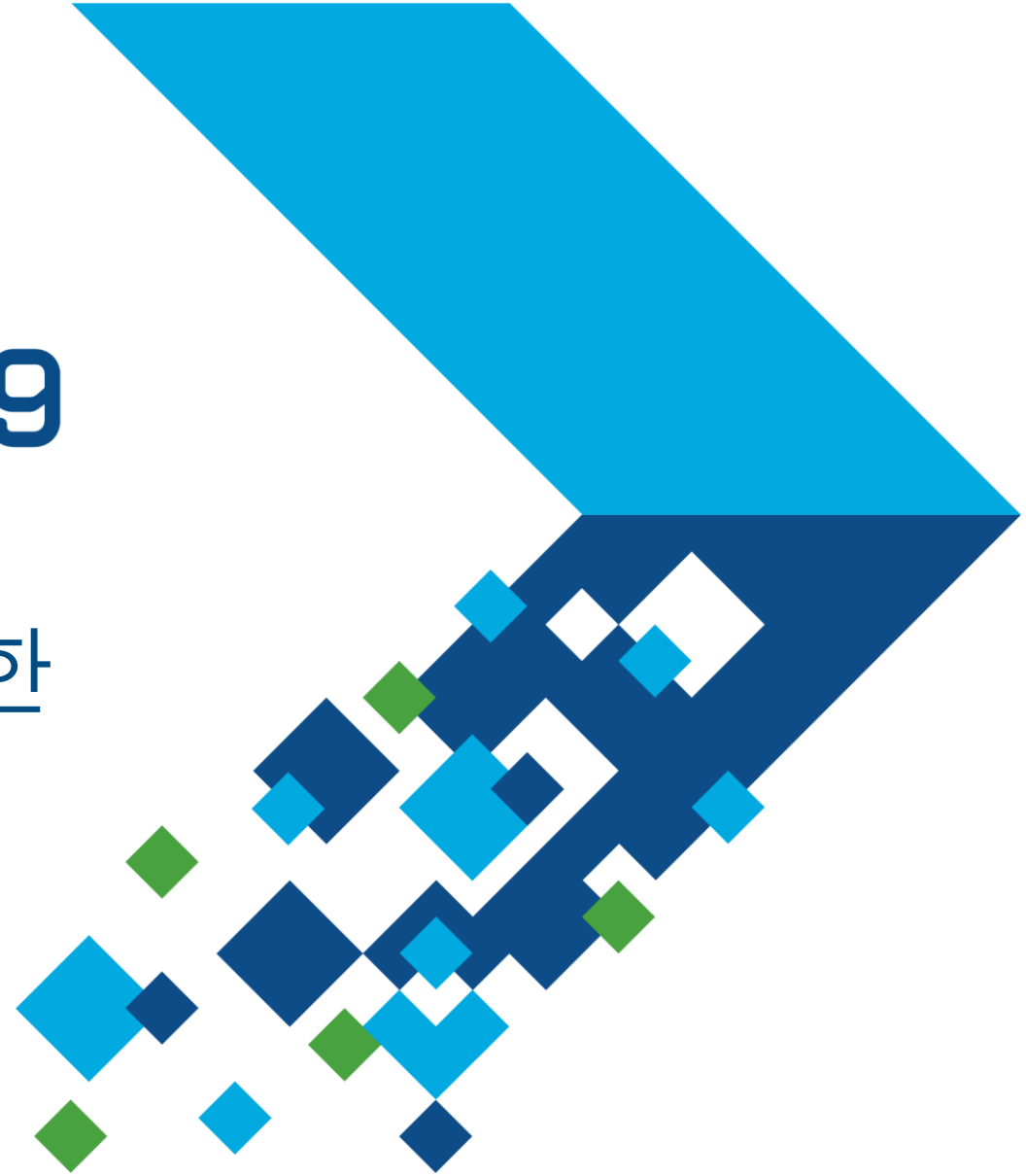
MATLAB EXPO 2019

FPGA, ASIC, SoC 개발을 위한 모델 기반 설계 도입 방안

정승혁



Agenda

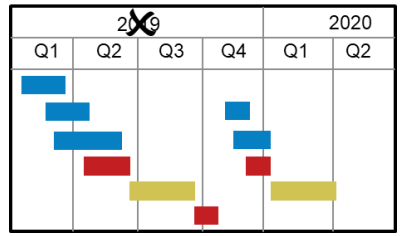
- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer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

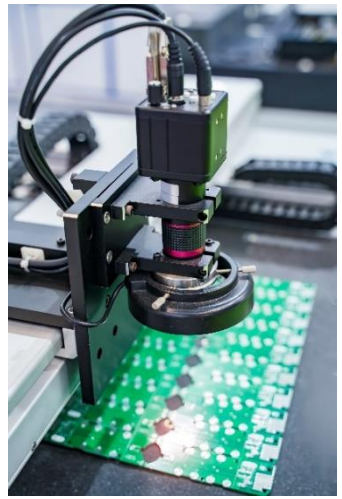
FPGA, ASIC, and SoC Development Projects

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**

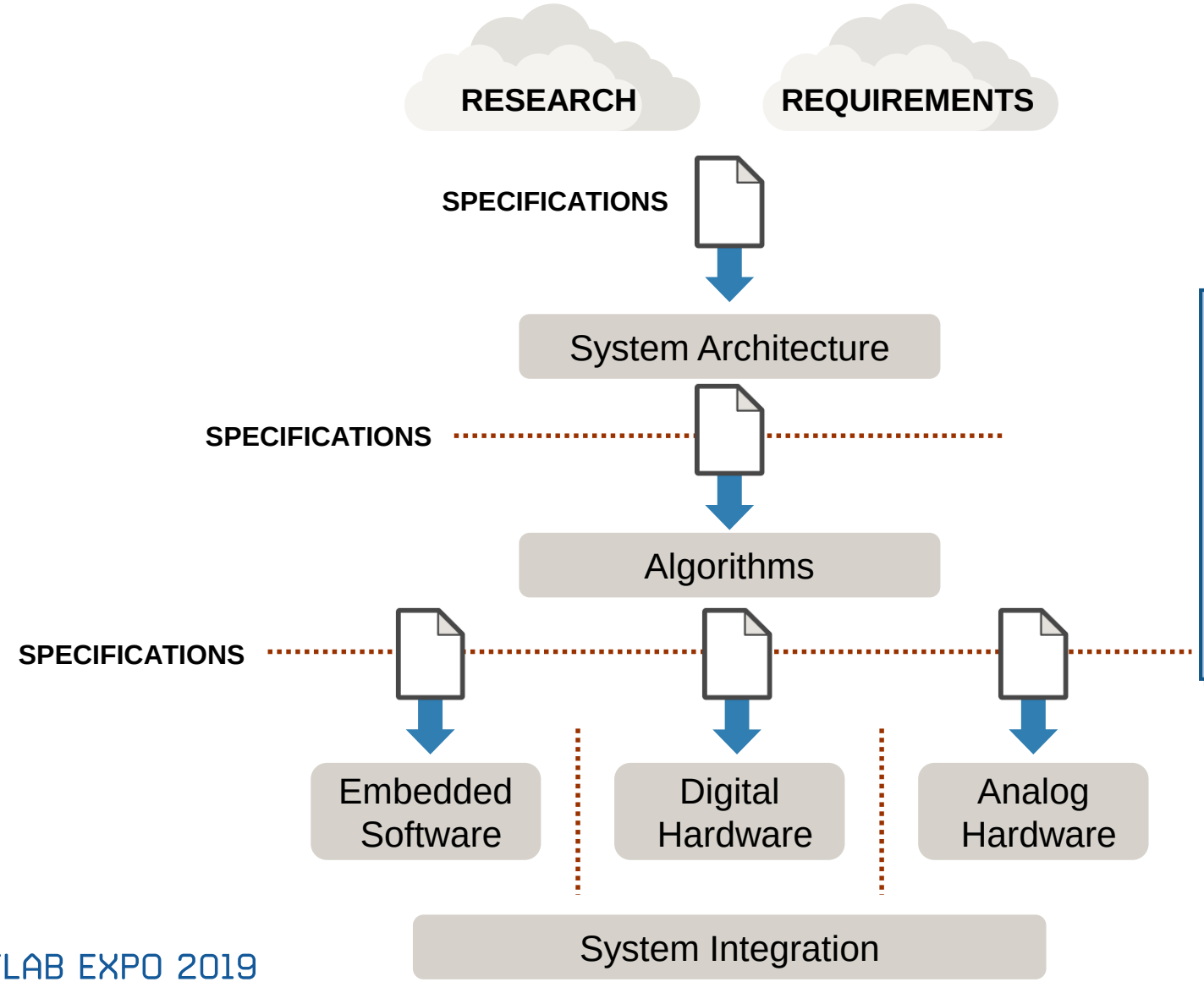




84% of FPGA projects have non-trivial **bugs escape into production**

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

Many Different Skill Sets Need to Collaborate

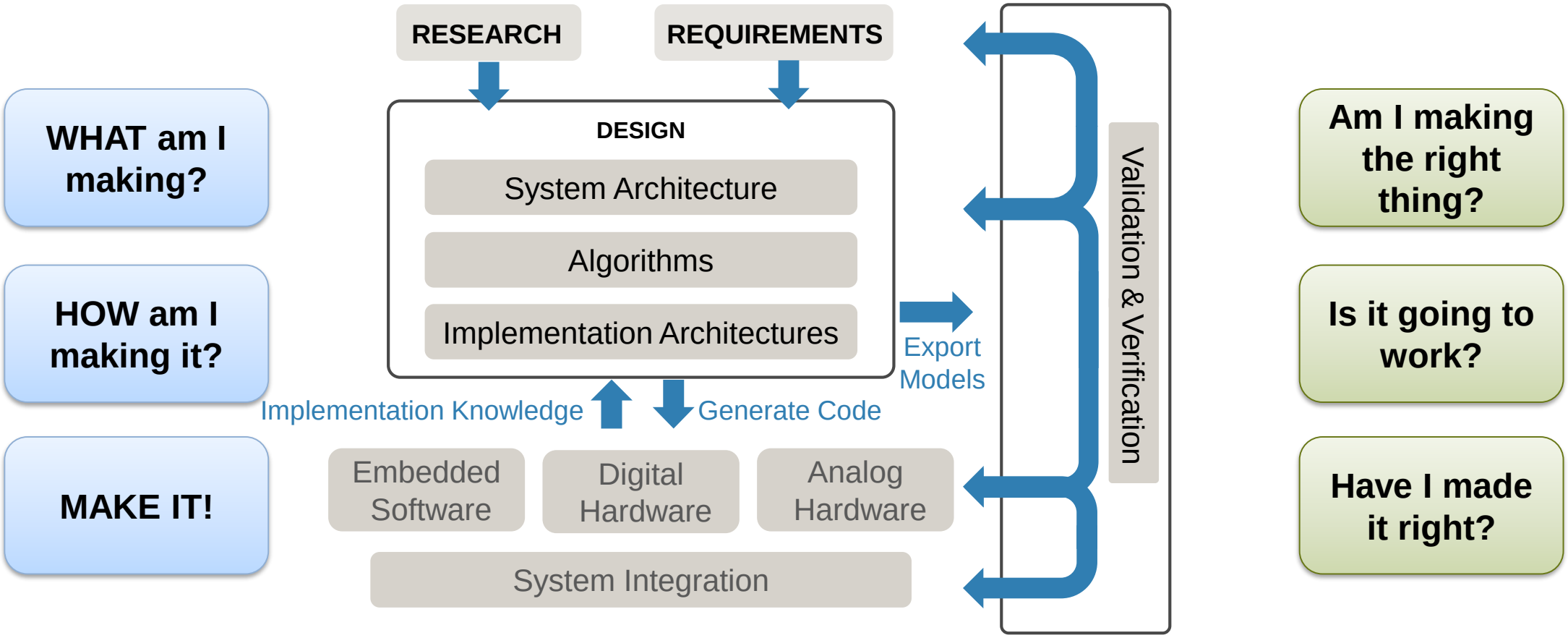


- Poor communication across teams
- Key decisions made in silos
- System-level issues found in late stages
- Hard to adapt to changing requirements

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

Hitachi
Noritaka Kosugi,
Kazuyuki Hori, and Yuji Ishida

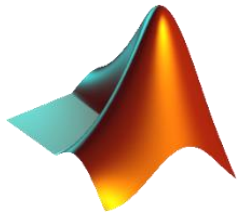
SoC Collaboration with Model-Based Design



Agenda

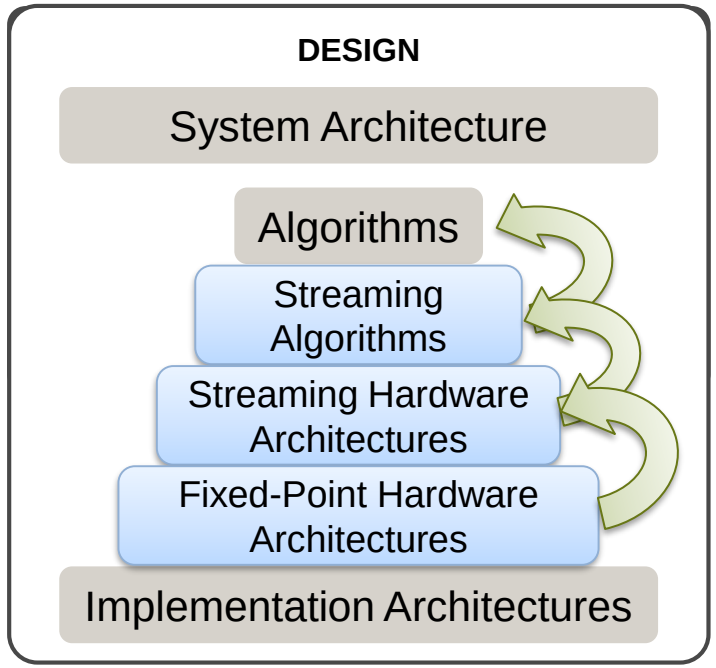
- Why Model-Based Design for FPGA, ASIC, or SoC?
- How to get started
  - General approach – collaborate to refine with implementation detail
  - Re-use work to help RTL verification
  - Hardware architecture
  - Fixed-point quantization
  - HDL code generation
  - Chip-level architecture
- Customer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

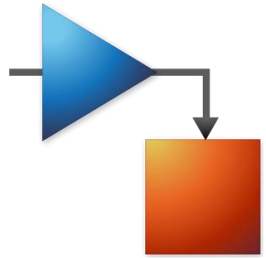
General Approach: Use the Strengths of MATLAB and Simulink



MATLAB

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





Simulink

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

Partition Hardware-Targeted Design, System Context, Testbench

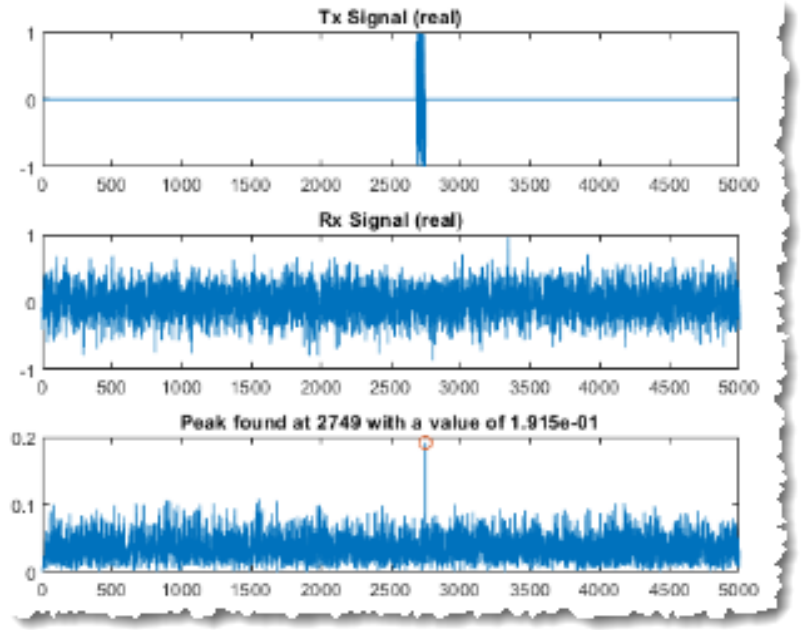
Algorithm Stimulus | **Hardware Algorithm** | **Software Algorithm** | **Analysis**

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));
```



[Example: HDL self-guided tutorial]

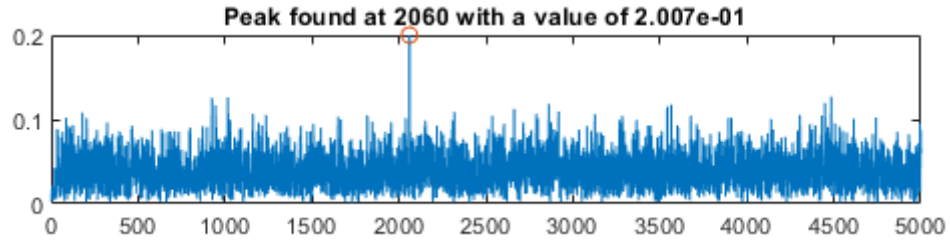
Streaming Algorithms: MATLAB or Simulink…or Both

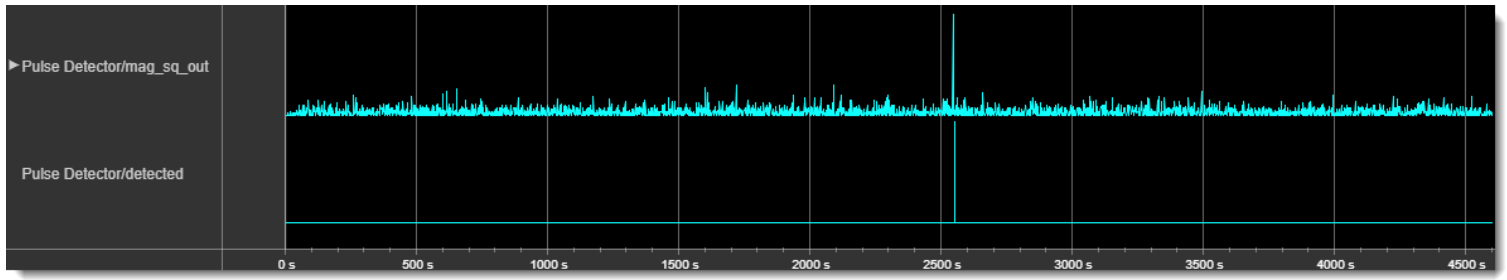
Hardware friendly implementation of peak finder

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:

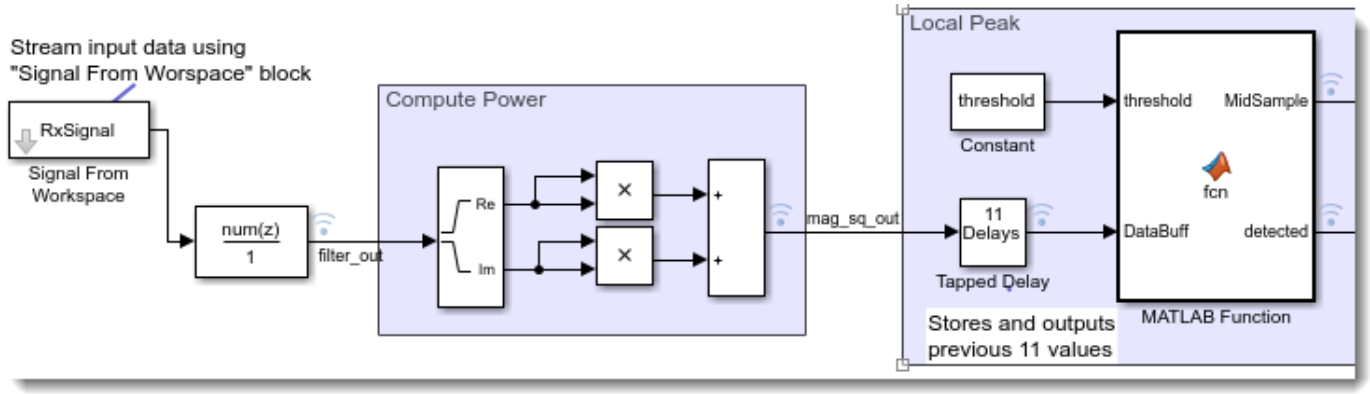
- The middle sample is the largest
- The middle sample is greater than a pre-defined threshold

```
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
```

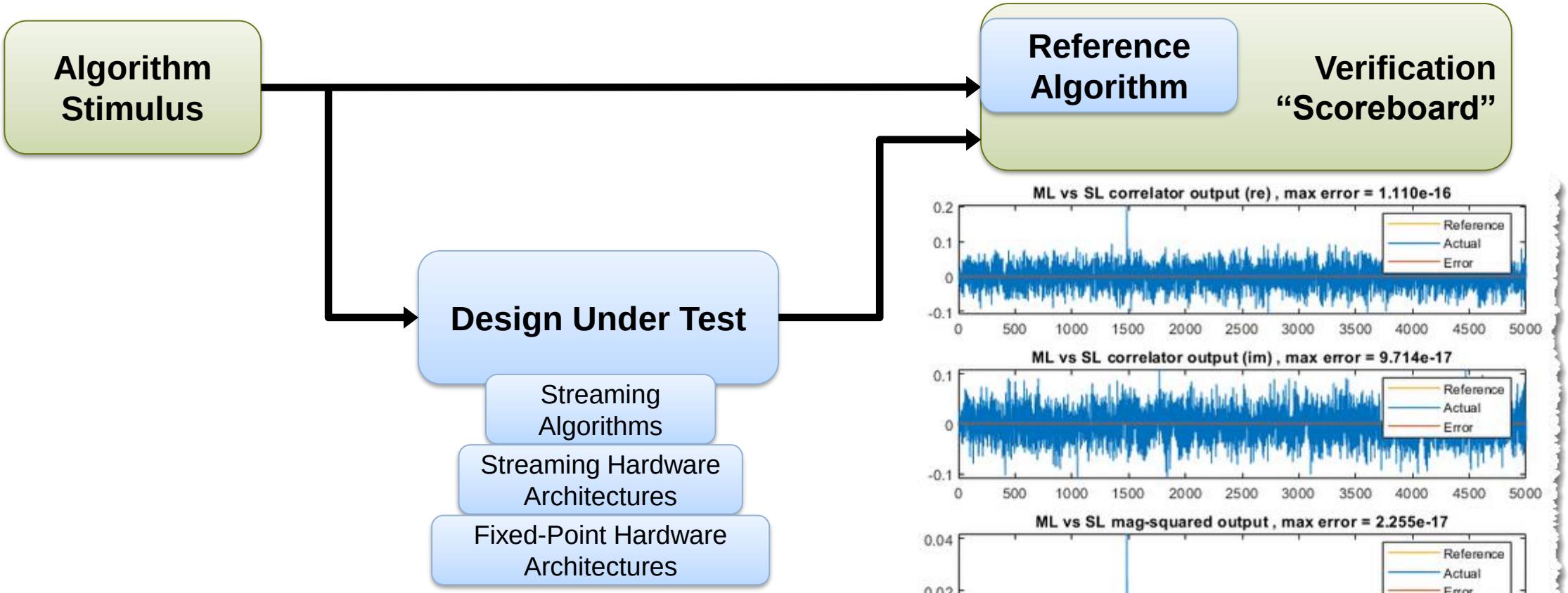




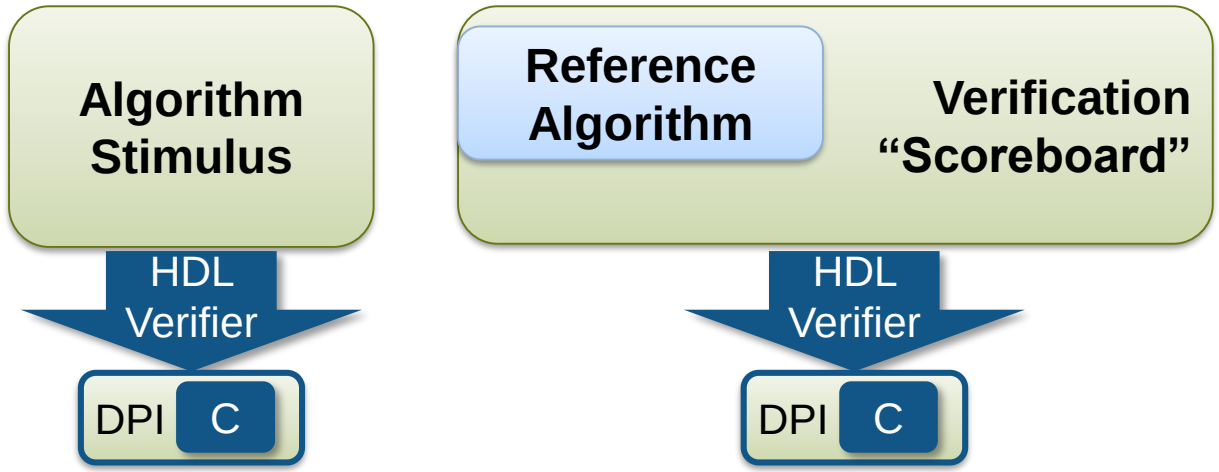

```
% Simulate model
sim('pulse_detector_v1')

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



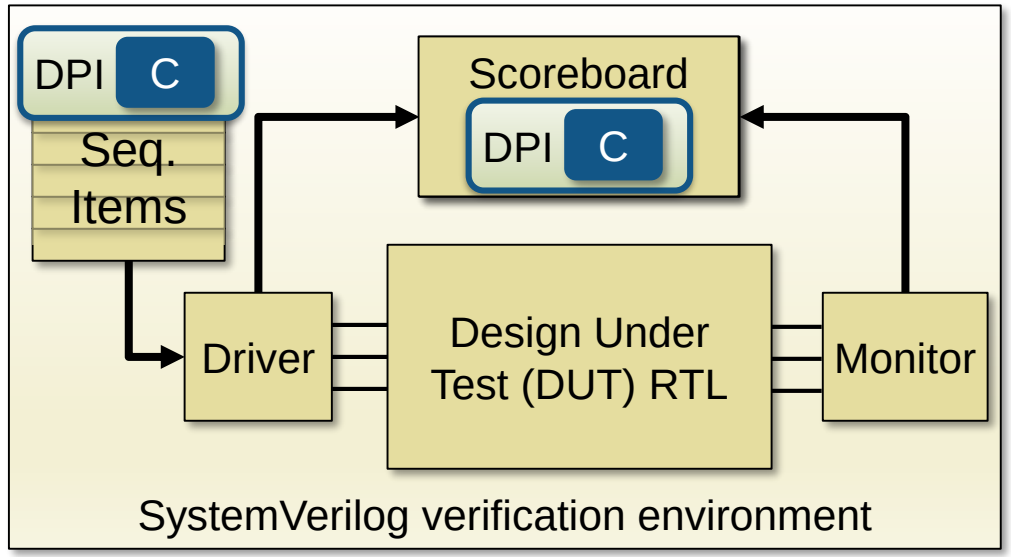
Refine Algorithm and Verify Against Golden Reference



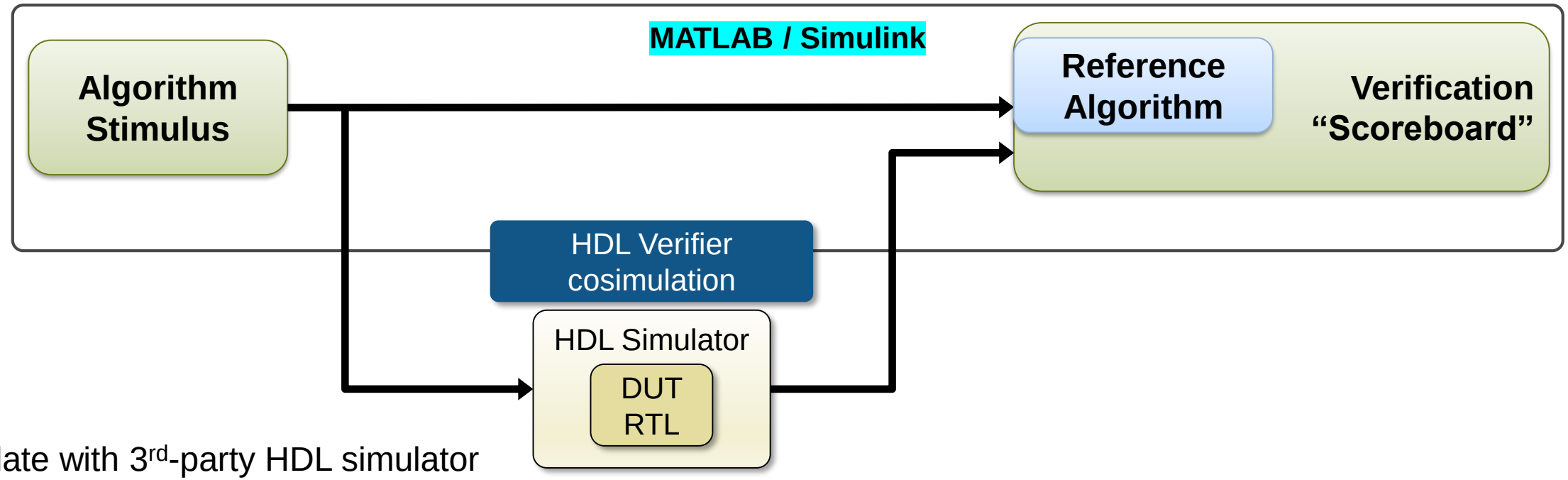
Generate SystemVerilog DPI Components for RTL Verification



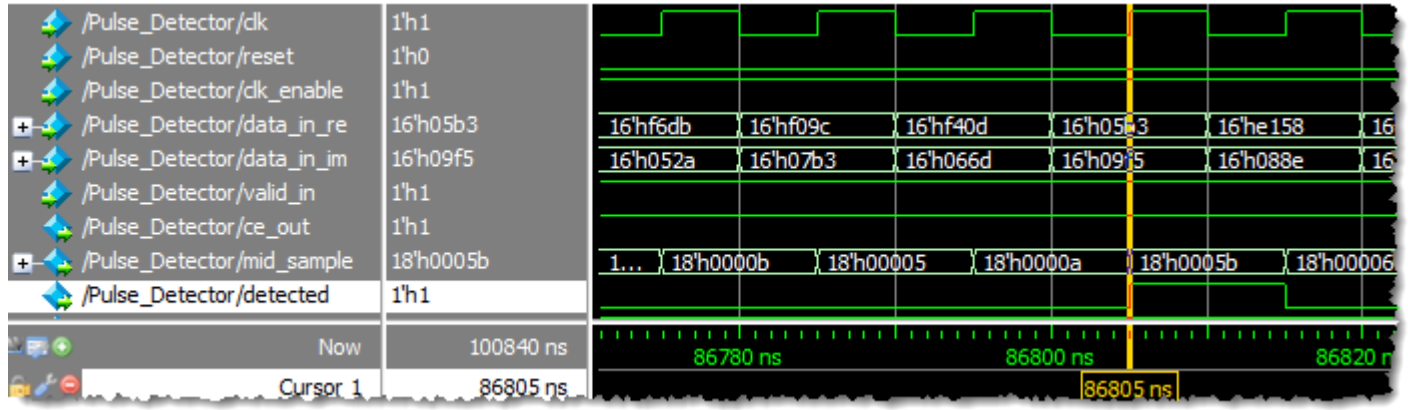
- Reuse MATLAB/Simulink models in verification
  - Scoreboard, stimulus, or models external to the RTL
    - Generate from frame-based or streaming algorithm
    - Floating-point or fixed-point
    - Individual components or entire testbench
  - Runs natively in SystemVerilog simulator
  - Eliminate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



What if there's a mismatch?



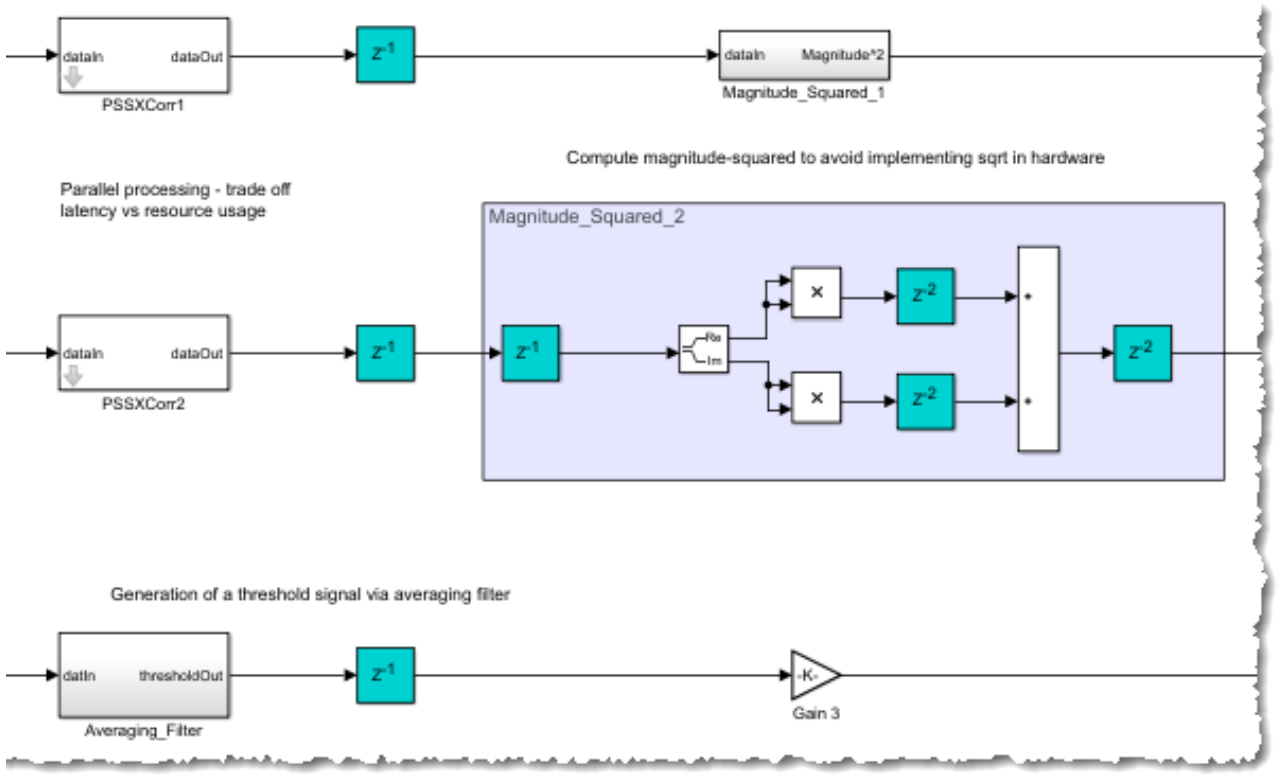
- Co-simulate with 3rd-party HDL simulator
  - Reuse MATLAB/Simulink test environment
  - Run HDL design in a supported simulator*
  - Generate co-simulation infrastructure and handshaking
  - Analyze both the design and test environment



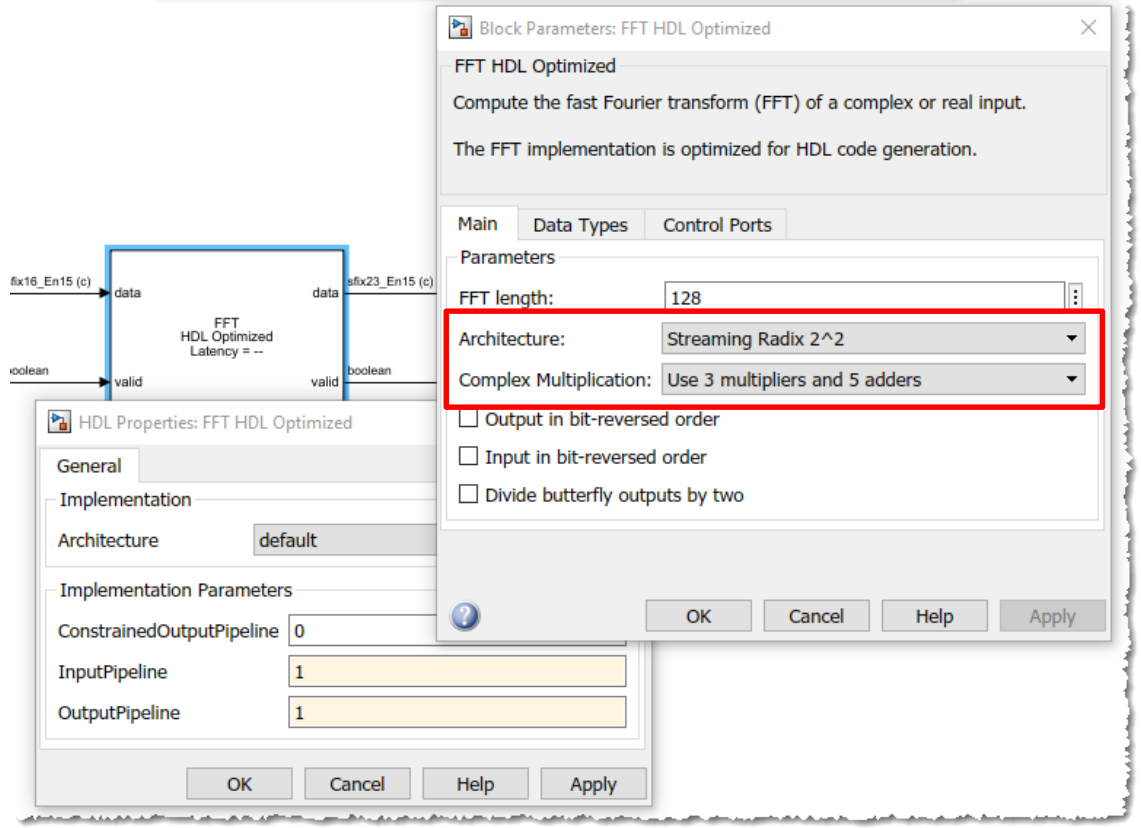
* Mentor Graphics® ModelSim® or Questa® Cadence® Incisive® or Xcelium™

Collaborate to Add Hardware Architecture

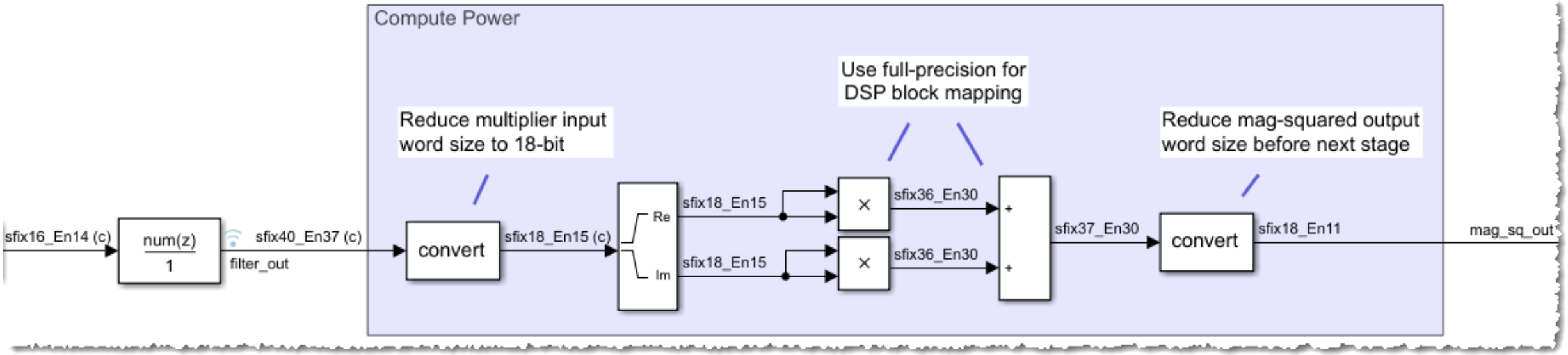
Optimize architecture design for hardware goals



Specify HDL implementation options



Fixed-Point Streaming Algorithms: Manual Approach



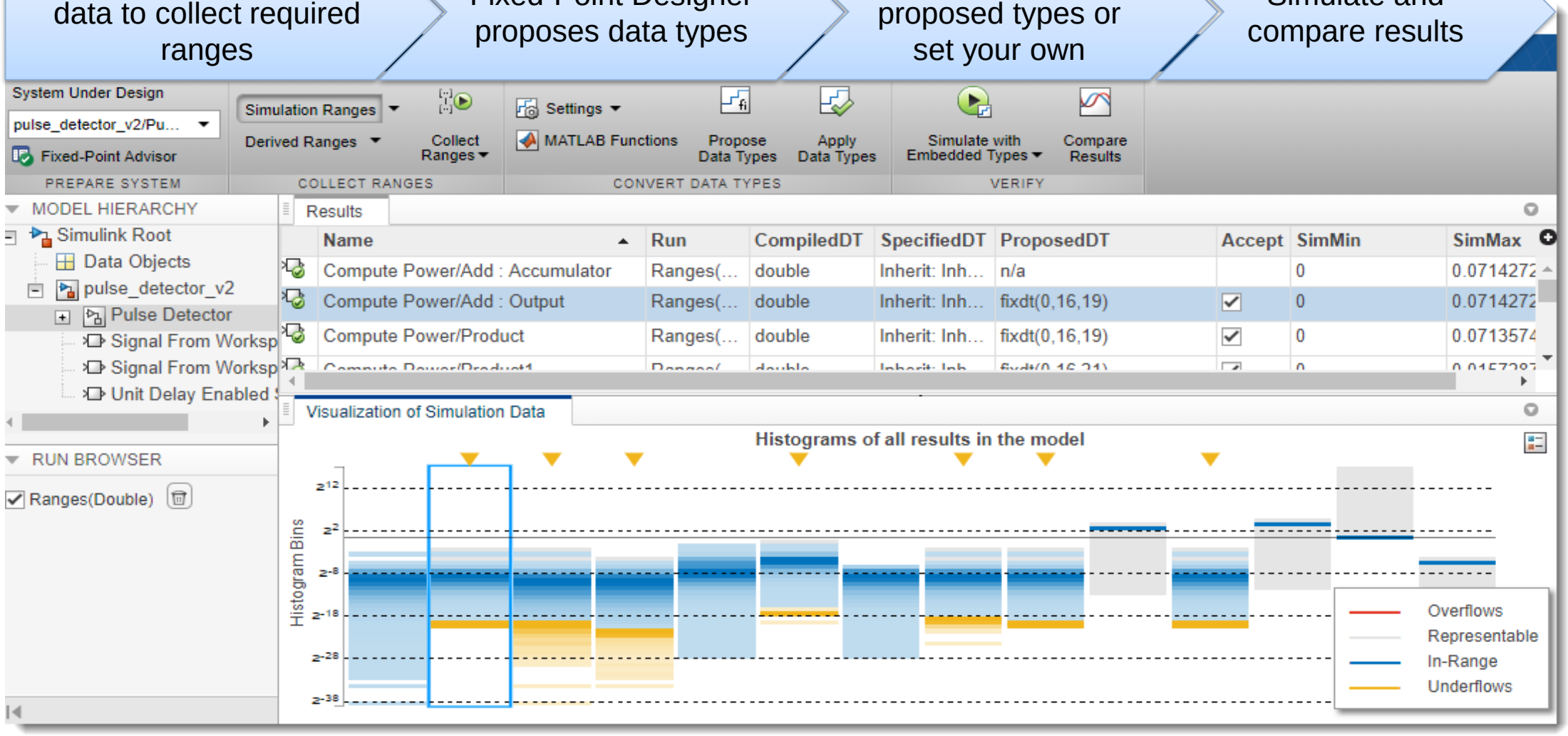
Fixed-Point Streaming Algorithms: Automated Approach

Simulate with representative data to collect required ranges

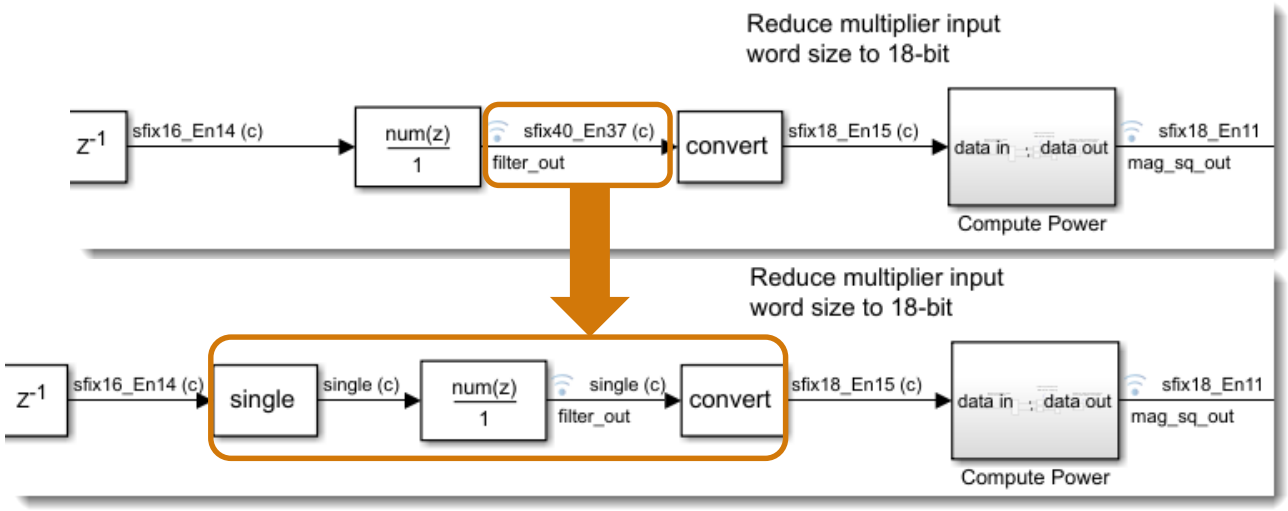
Fixed-Point Designer proposes data types

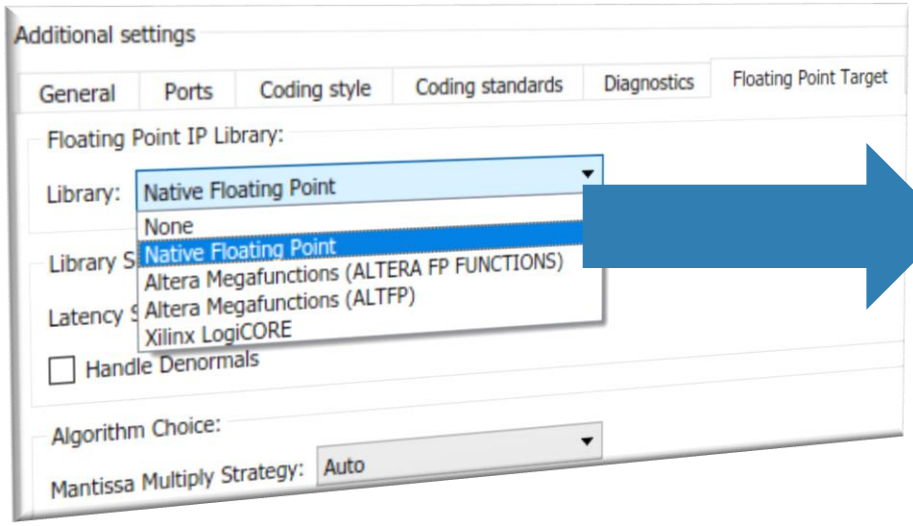
Choose to apply proposed types or set your own

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

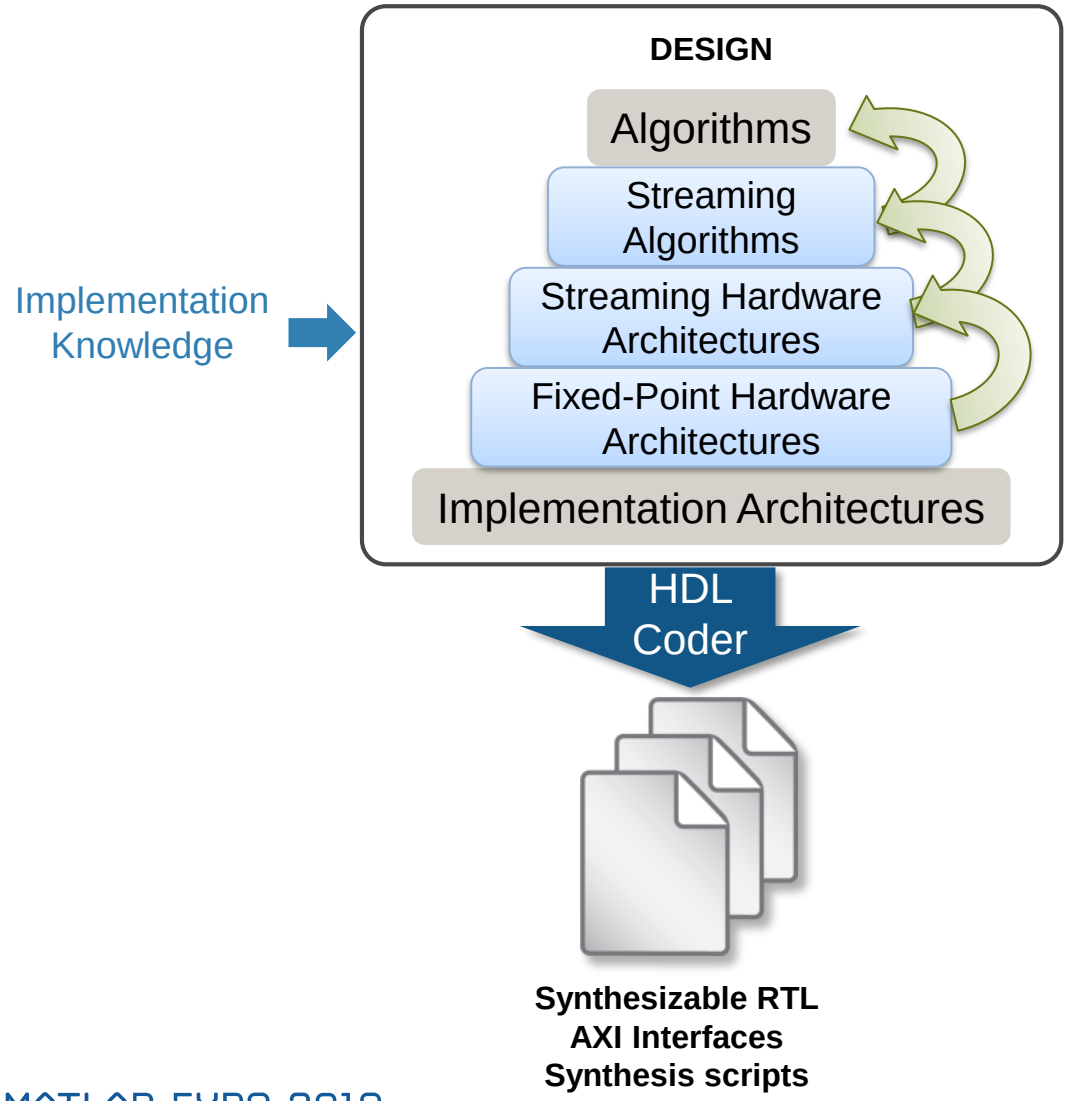
embeddedaward2017

```
SIGNAL nfp_out_1_im_93  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_im_94  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_im_95  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_im_96  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_im_97  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_im_98  : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL filter_out_re    : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL filter_out_im    : std_logic_vector(31 DOWNTO 0);  -- ufix32
SIGNAL nfp_out_1_re_99  : std_logic_vector(17 DOWNTO 0);  -- ufix18
SIGNAL nfp_out_1_im_99  : std_logic_vector(17 DOWNTO 0);  -- ufix18
SIGNAL mag_sq_out       : std_logic_vector(17 DOWNTO 0);  -- ufix18
SIGNAL MidSample        : std_logic_vector(17 DOWNTO 0);  -- ufix18
```

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

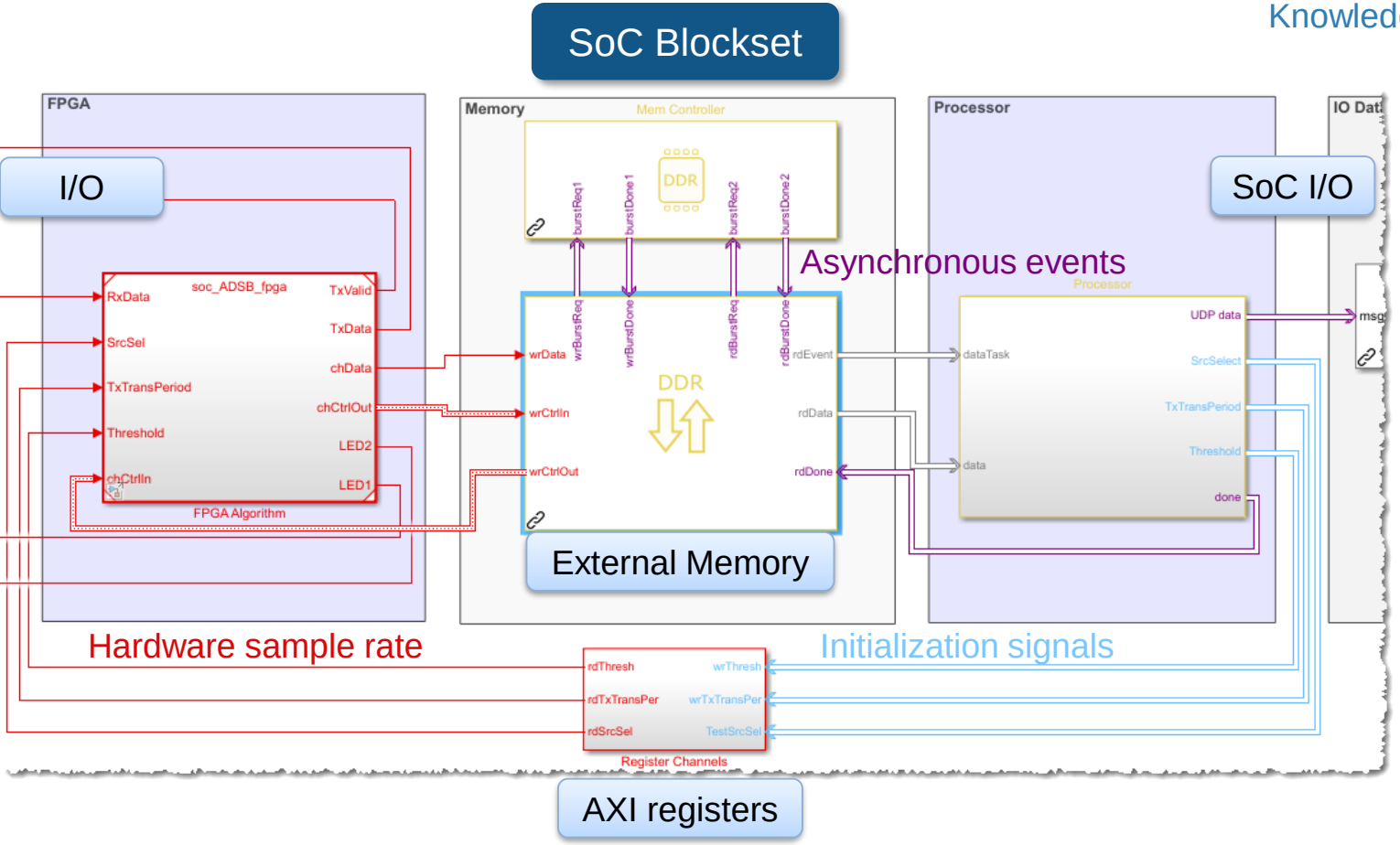
~2x more resources
~5x less development effort

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

Model and Simulate SoC Architecture



Implementation Knowledge

DESIGN

- System Architecture
- Algorithms
- Implementation Architectures

- 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

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 of Top-Down Model-Based Design at Huawei

Huawei is the largest networking and telecommunications equipment and services corporation in the world. It employs 176,000 people, 79,000 of whom work in Research and Development centers around the globe (based on 2015 data).

Kevin Law, director of algorithm architecture and design, provides an insider view of 5G development at Huawei.

Solution

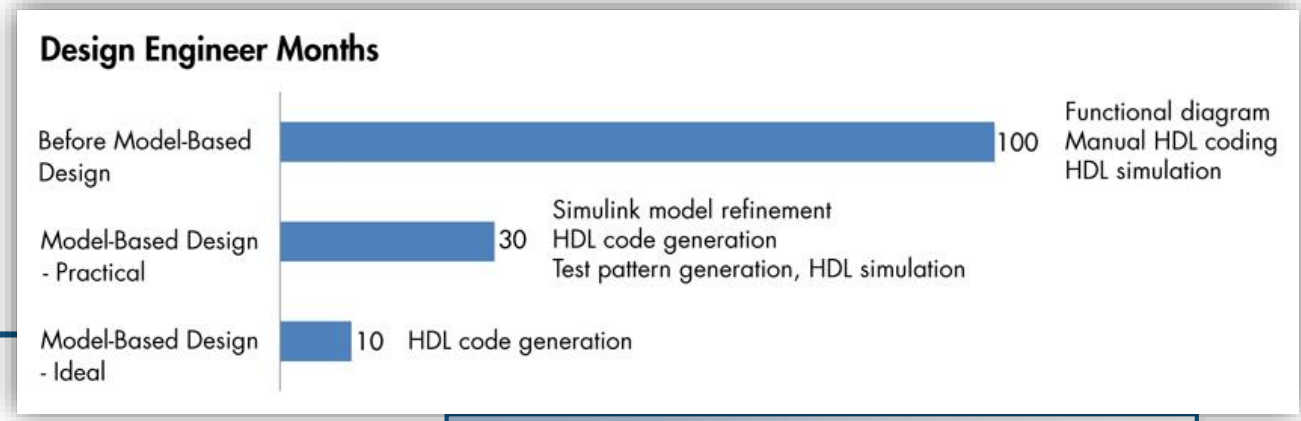
We have introduced Model-Based Design with MATLAB and Simulink and the Huawei R&D workflow. MATLAB and Simulink have a strong impact from product requirement import, modeling, and data analysis, all the way to link-level system simulation, algorithm selection, RTL code generation, testing, and prototyping, and even sample chip tape-out and testing. Model-Based Design can also **improve the overall end-to-end efficiency** quite a lot.

Results

- We generated code automatically from MATLAB, Simulink, and HDL Coder. Auto-generated code was slightly higher than hand-written coder in terms of hardware resources such as multipliers and memory, but relatively similar in timing. It satisfied our algorithm verification requirement. When we started using this approach for verification, we **reduced development time by 50%**. It is a huge gain that meets our manpower constraints and time-to-market requirement.

Link to full Q&A

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.

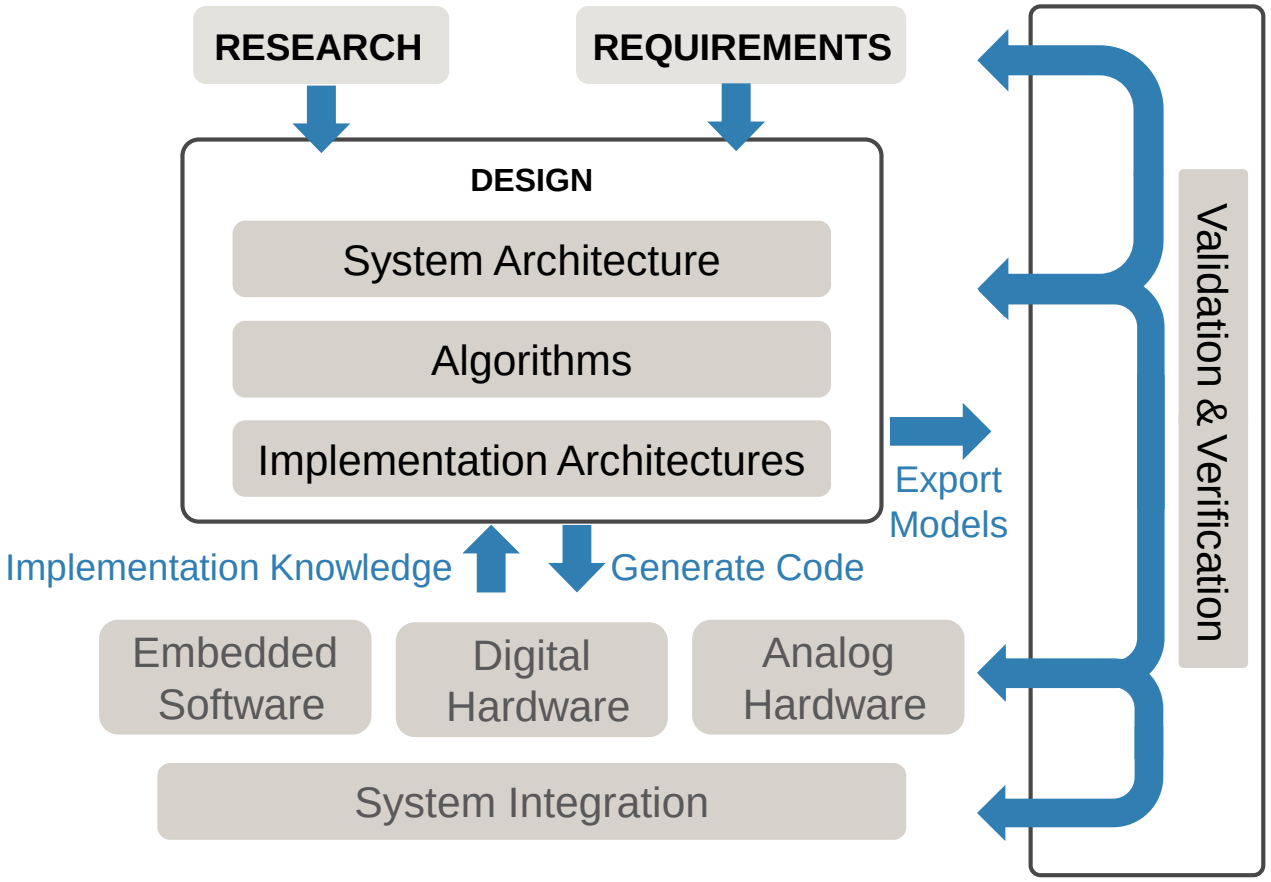
Results

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

Link to article

"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."

Getting Started Collaborating with Model-Based Design



- 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

- ❑ 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

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
  - Getting started guide for evaluating HDL Coder: HDL Coder Evaluation Reference Guide

Find out more:
SoC Blockset 소개

정승혁

MATLAB EXPO 2019

데모 부스와 상담부스로 질문 하시기 바랍니다.

감사합니다