MATLAB EXPO

FPGA开发流程中的HDL代码生成和验证

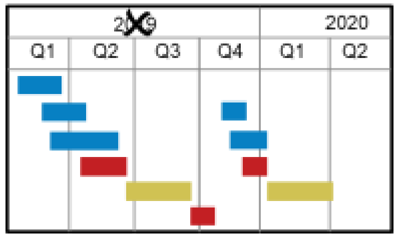
赵恒，MathWorks中国项目工程师

MathWorks

FPGA、ASIC和SoC开发现状

67%的ASIC/FPGA项目**落后于进度**

超过**50%**的项目时间用于**验证**



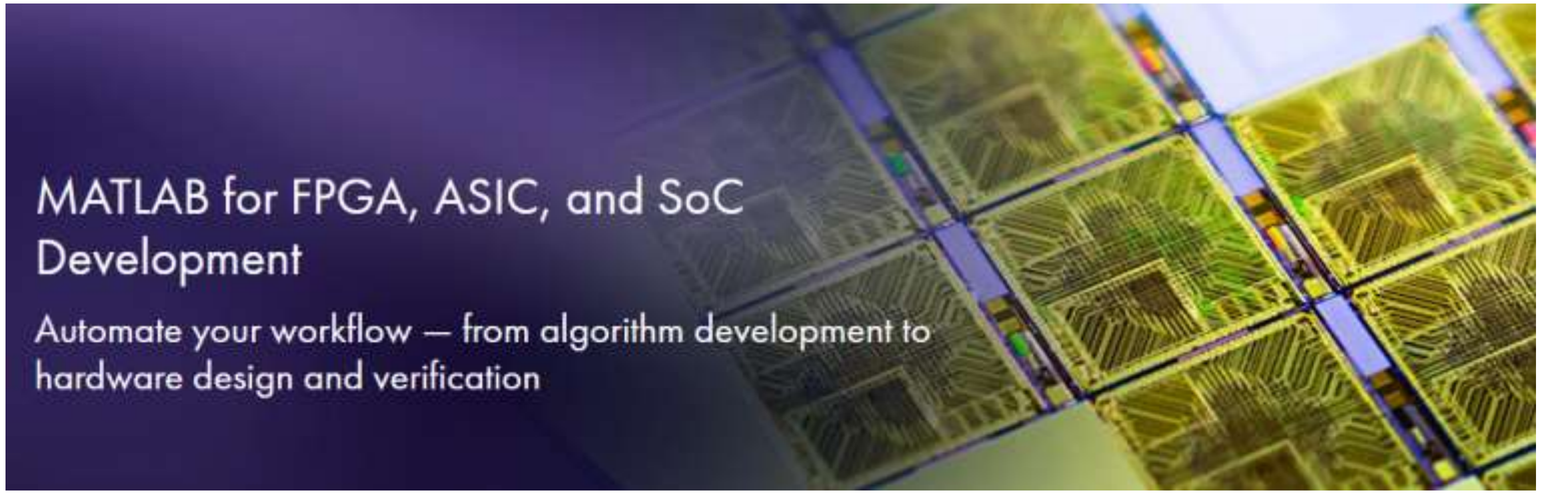
75%的ASIC项目投片完成后发现问题

84%的FPGA项目交付后发现bug

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

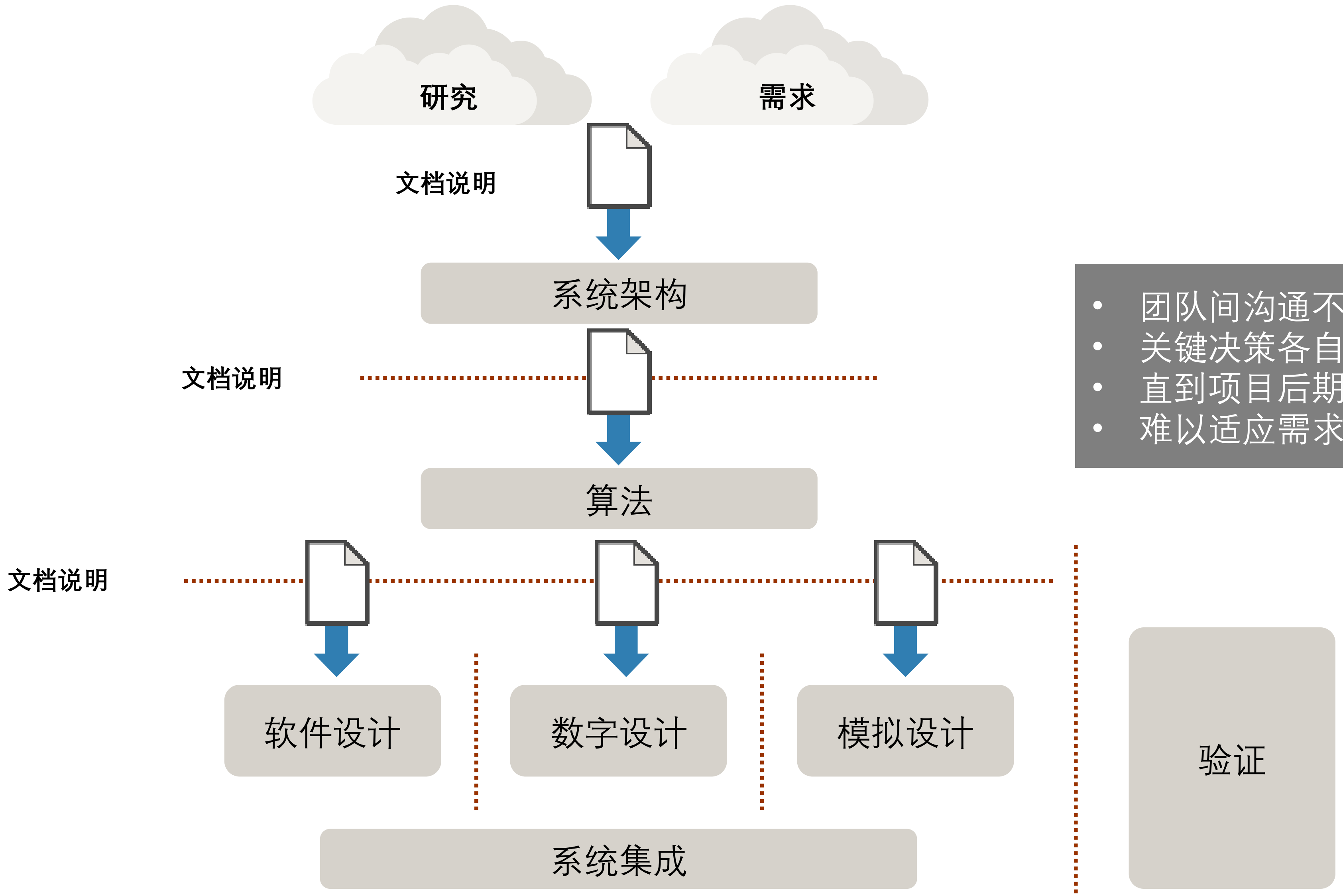
Agenda

- **基于模型的FPGA开发**
- HDL代码生成
- FPGA验证


MATLAB for FPGA, ASIC, and SoC Development

Automate your workflow — from algorithm development to hardware design and verification

芯片设计需要多方面知识



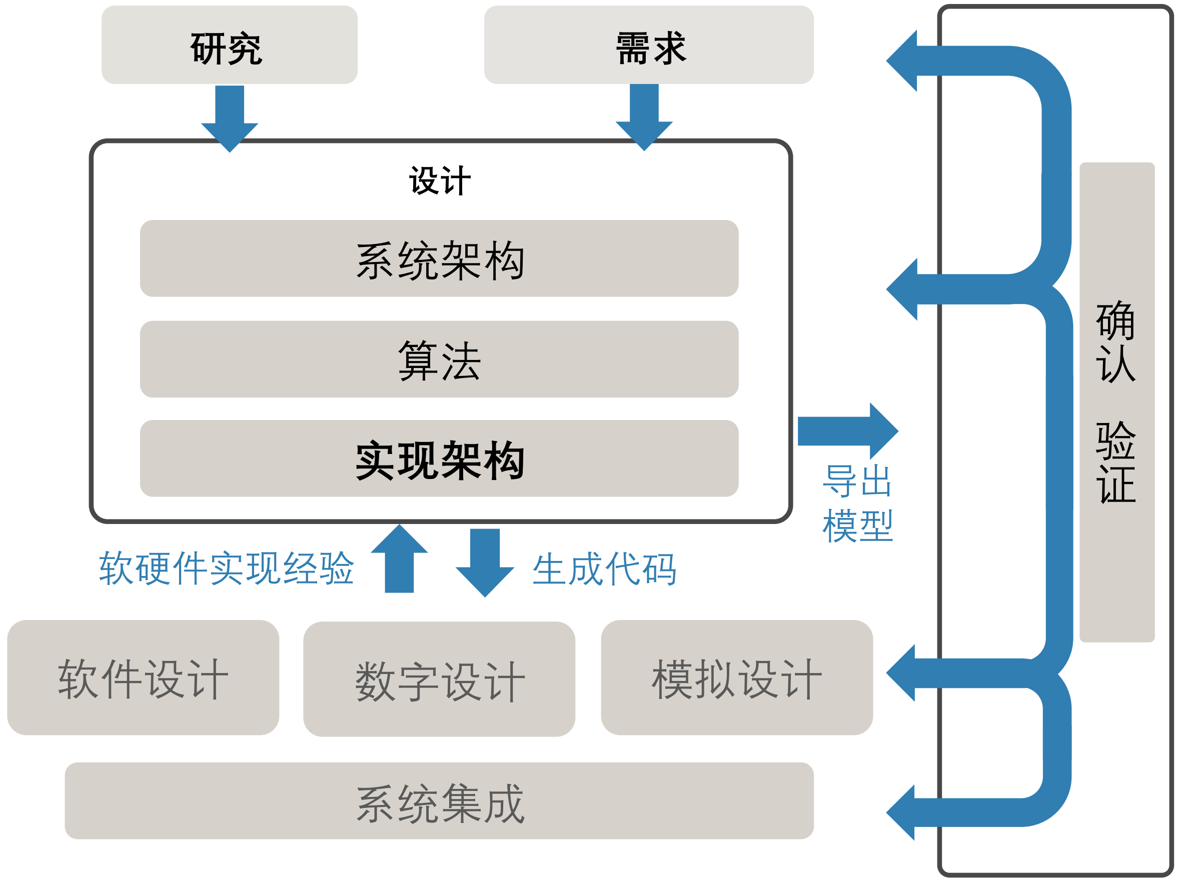
- 团队间沟通不畅
- 关键决策各自为政
- 直到项目后期才发现系统级问题
- 难以适应需求的不断变更

基于模型的FPGA/SoC设计

WHAT am I making?

HOW am I making it?

MAKE IT!

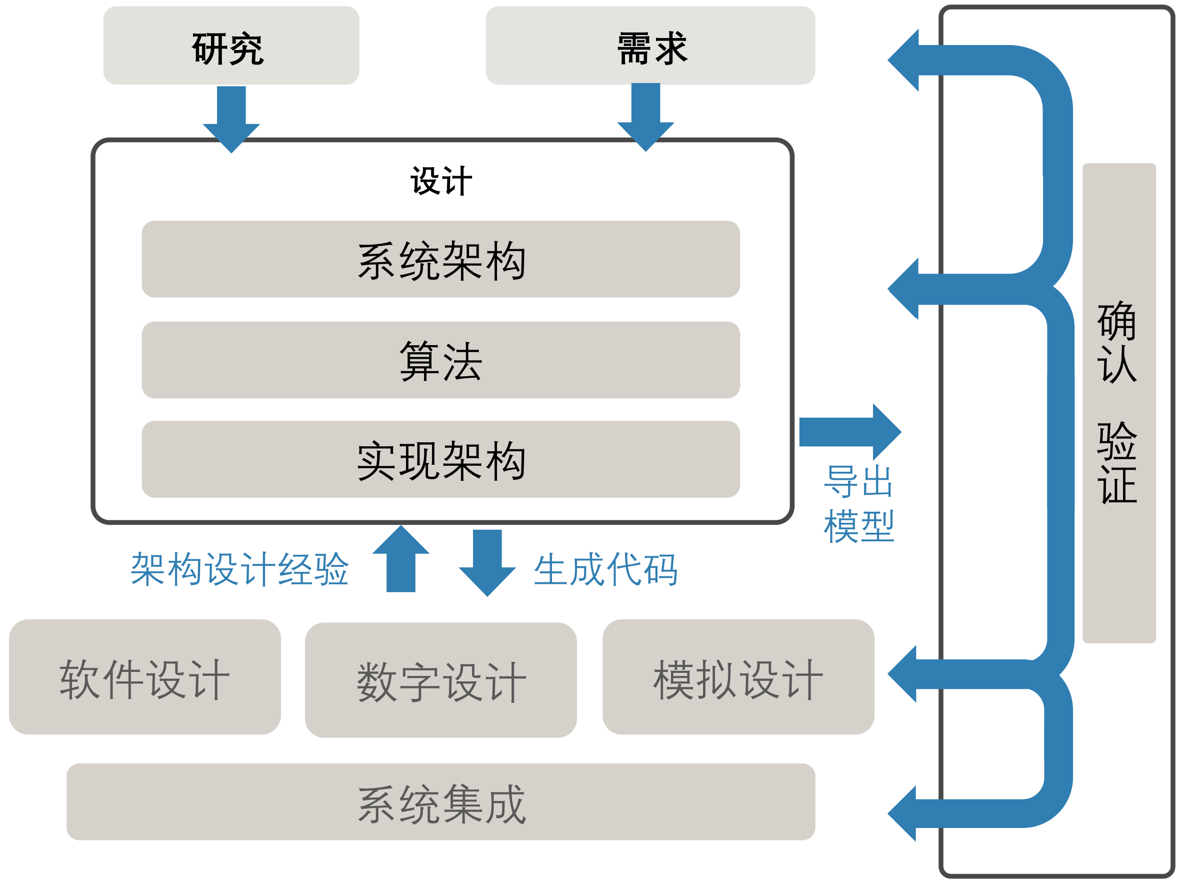


Am I making the right thing?

Is it going to work?

Have I made it right?

基于模型的FPGA/SoC设计



- ❑ 算法实现上的优化
- ❑ 优化实现前后的对比
- ❑ 生成验证模型
- ❑ 增加硬件实现上的细节并生成优化的RTL
- ❑ 对SoC架构设计的仿真

- ➢ 避免沟通不畅带来的影响
- ➢ 做出关键决策之前综合考虑多种因素
- ➢ 在子系统实现之前确定并解决系统级问题
- ➢ 灵活的适应不断变化的需求

Agenda

- 基于模型的FPGA开发
- **HDL代码生成**
- FPGA验证

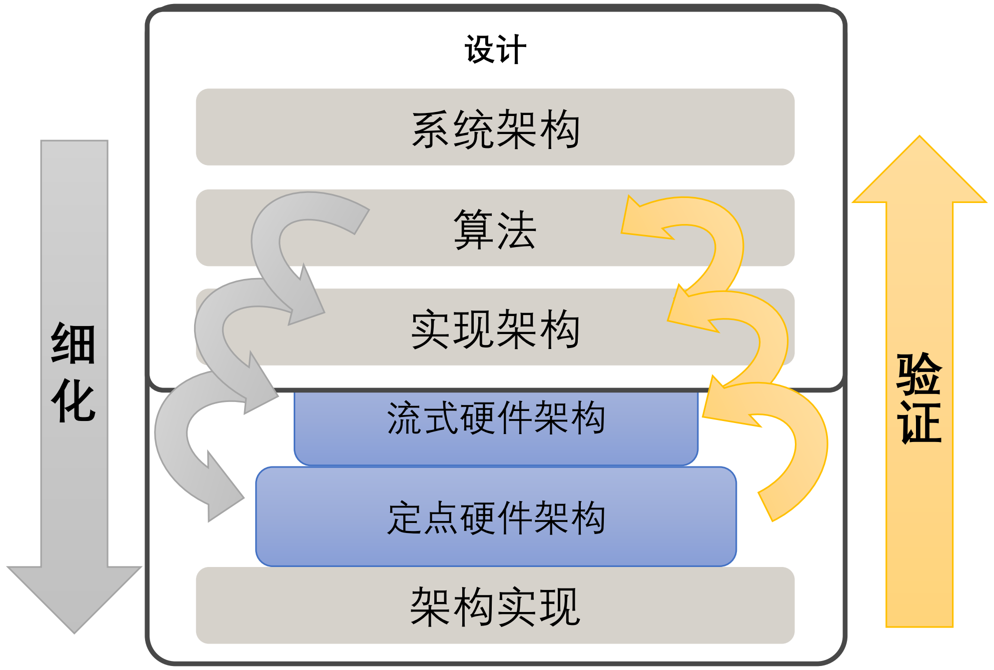
HDL Coder

Generate VHDL and Verilog code for FPGA and ASIC designs

结合使用MATLAB、Simulink和Stateflow

MATLAB

- ✓ 数据处理
- ✓ 算法描述
- ✓ 控制逻辑
- ✓ 数据可视化



Simulink

- ✓ 并行架构
- ✓ 时序
- ✓ 数据类型继承
- ✓ 混合信号建模

Stateflow

- ✓ 复杂控制逻辑/状态机

对系统进行拆分

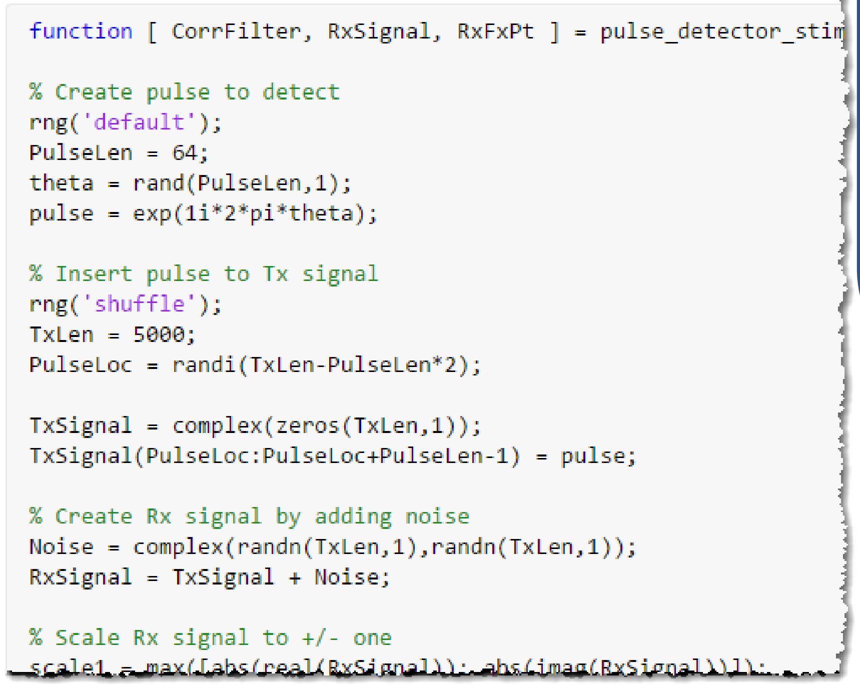
输入激励

算法的硬件实现部分

算法的软件实现部分

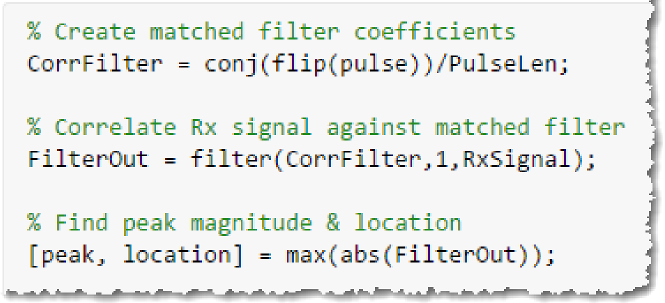
结果分析

Create input stimulus

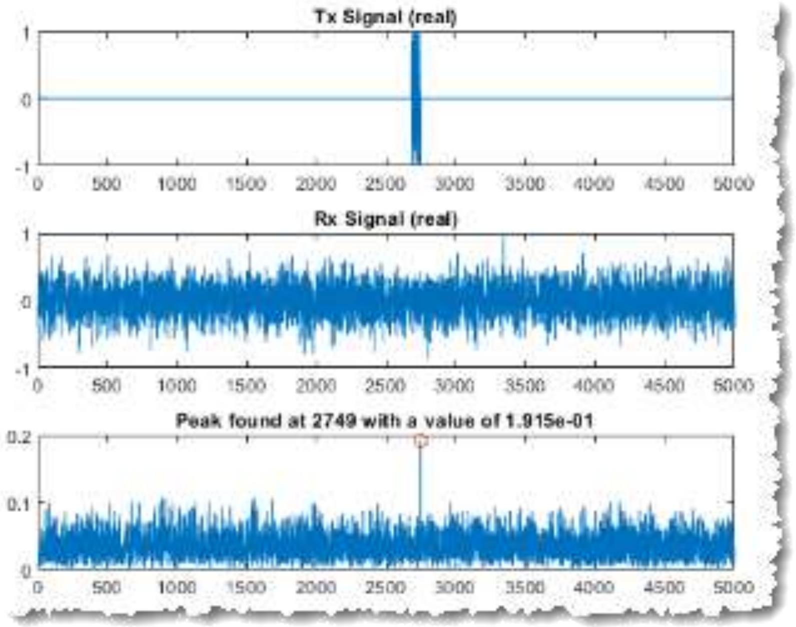


```
function [ CorrFilter, RxSignal, RxFxPt ] = pulse_detector_stim

% 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))]);
```

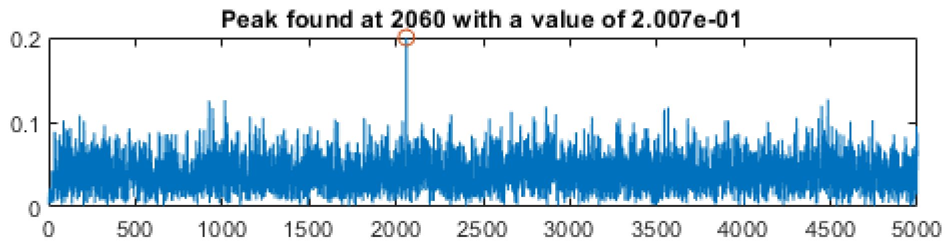
MATLAB golden reference



```
% 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));
```



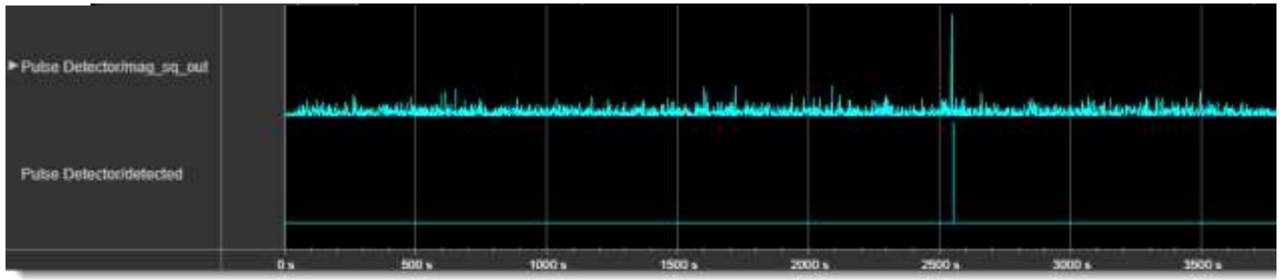
流式输入输出



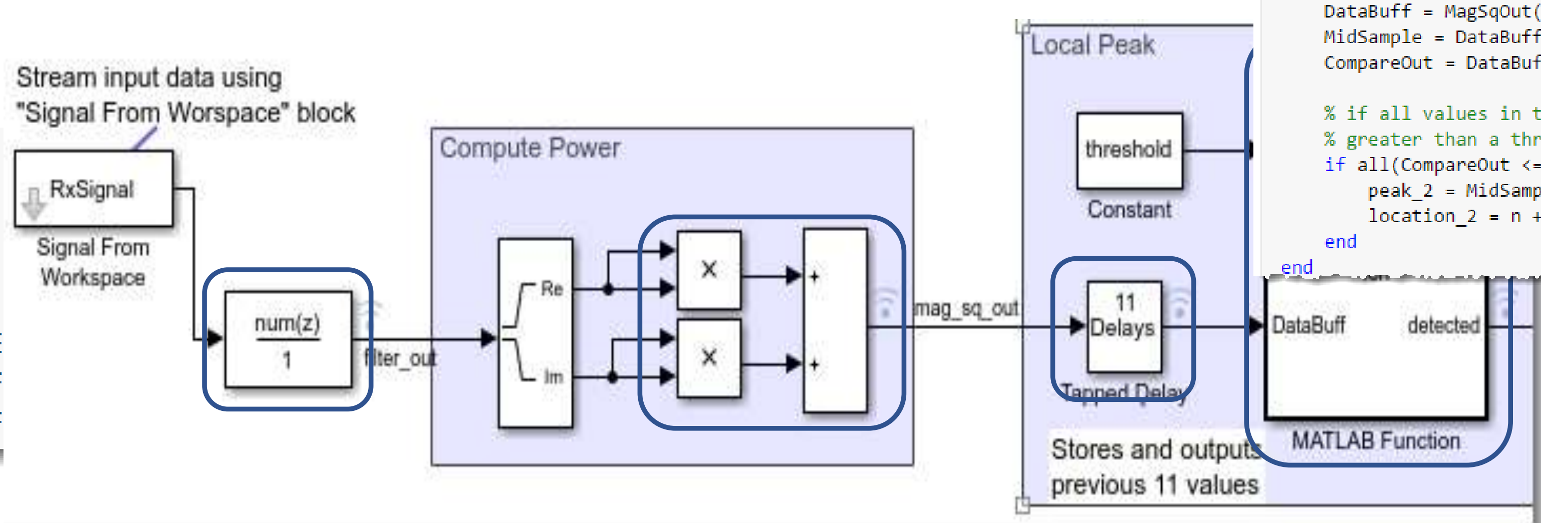
Hardware friendly implementation of peak finder

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

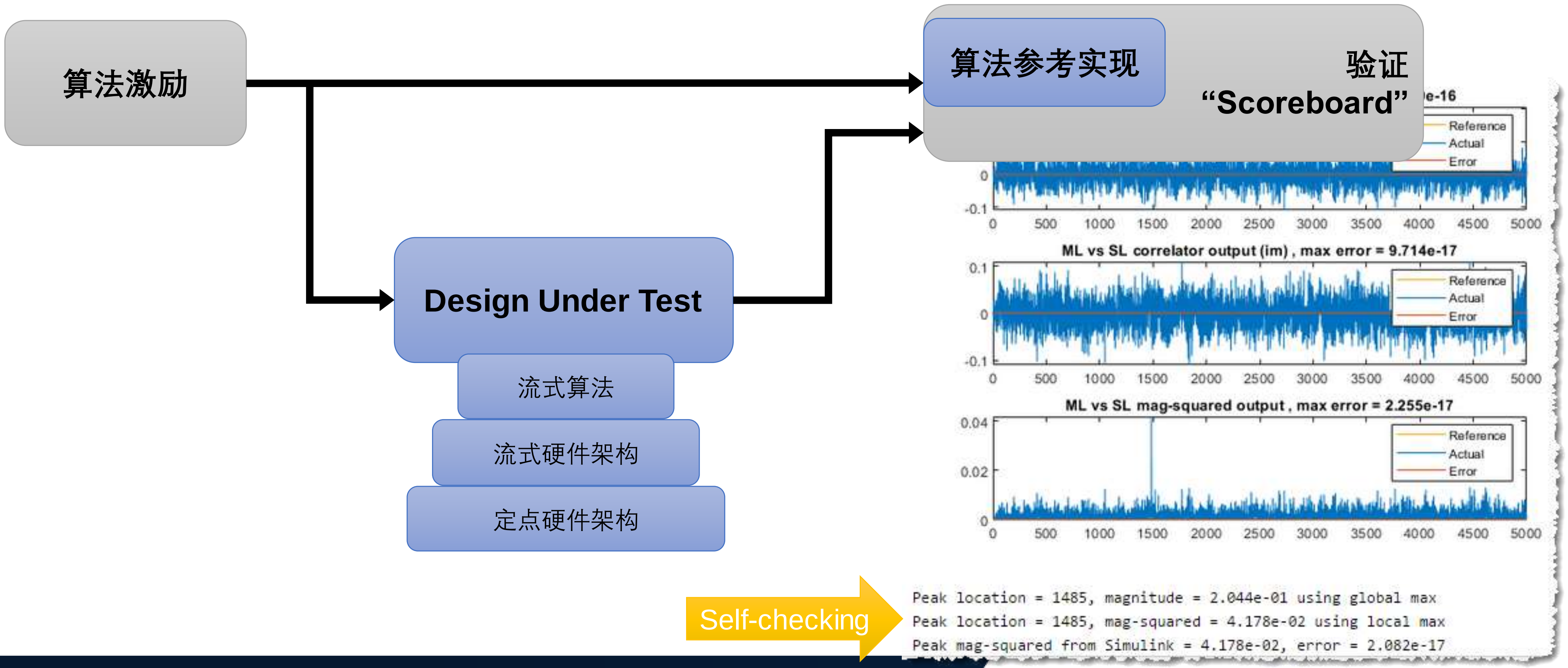


```
WindowLen = 11;
MidIdx = ceil(WindowLen/2);
threshold = 0.03;

% 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
```

算法细化并对照参考实现进行验证

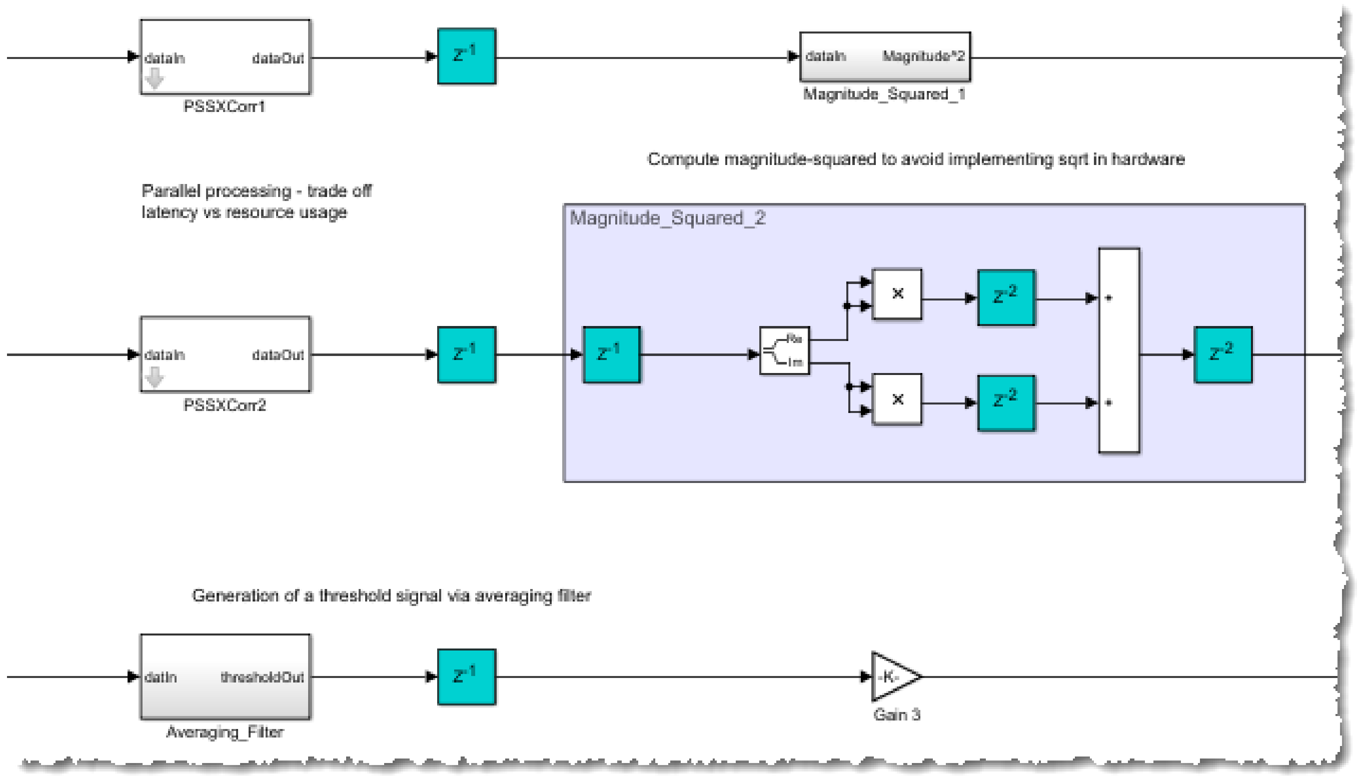


```
Peak location = 1485, magnitude = 2.044e-01 using global max
Peak location = 1485, mag-squared = 4.178e-02 using local max
Peak mag-squared from Simulink = 4.178e-02, error = 2.082e-17
```

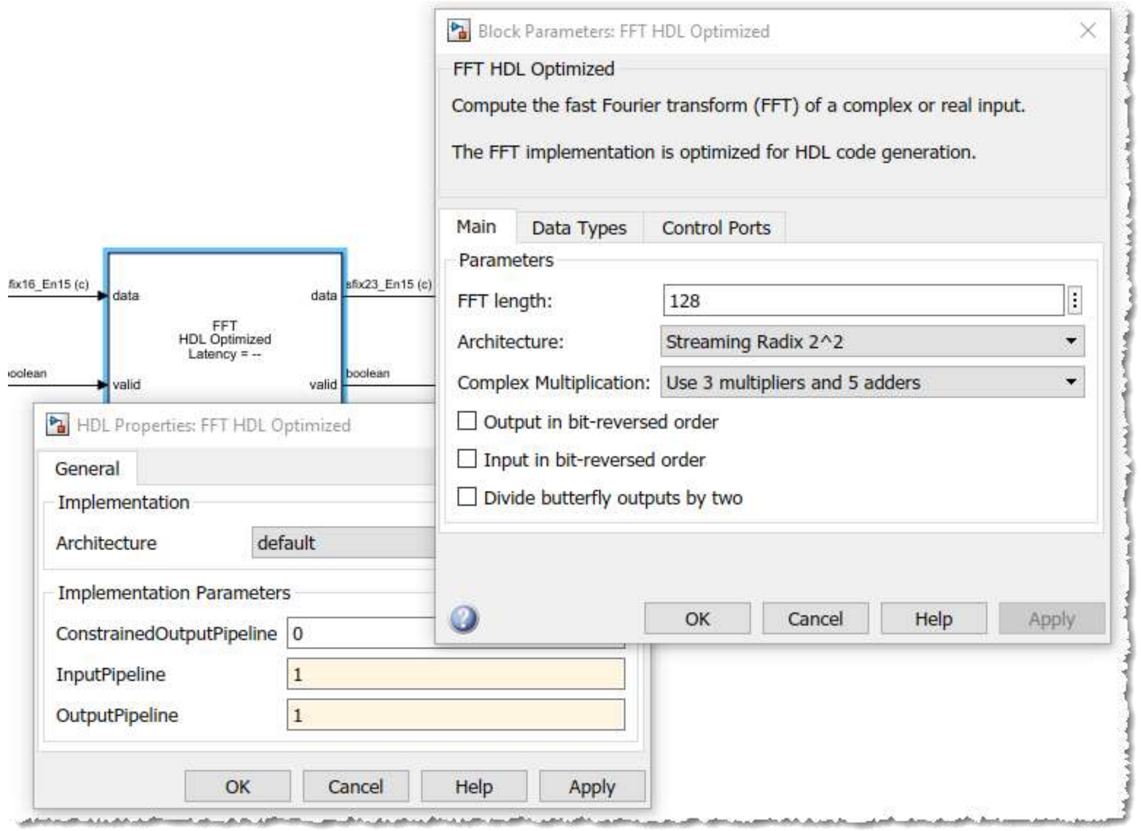
FPGA架构实现上的细化

为达到FPGA设计指标
优化架构设计

HDL实现上的不同选项



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
PSSXCorr2
z^-2
Generation of a threshold signal via averaging filter
datIn
thresholdOut
Averaging_Filter
-K-
Gain 3



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

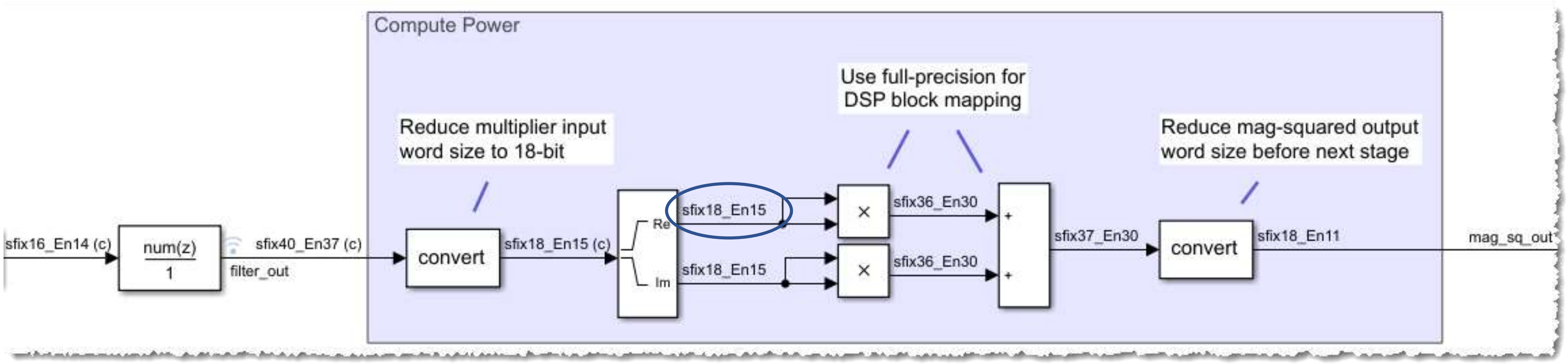
ConstrainedOutputPipeline: 0

InputPipeline: 1

OutputPipeline: 1

OK | Cancel | Help | Apply

手工定点化


Compute Power
Reduce multiplier input word size to 18-bit
Use full-precision for DSP block mapping
Reduce mag-squared output word size before next stage
sfix16_En14 (c)
num(z)
1
sfix40_En37 (c)
filter_out
convert
sfix18_En15 (c)
Re
Im
sfix18_En15
sfix18_En15
×
×
sfix36_En30
sfix36_En30
+
+
sfix37_En30
convert
sfix18_En11
mag_sq_out

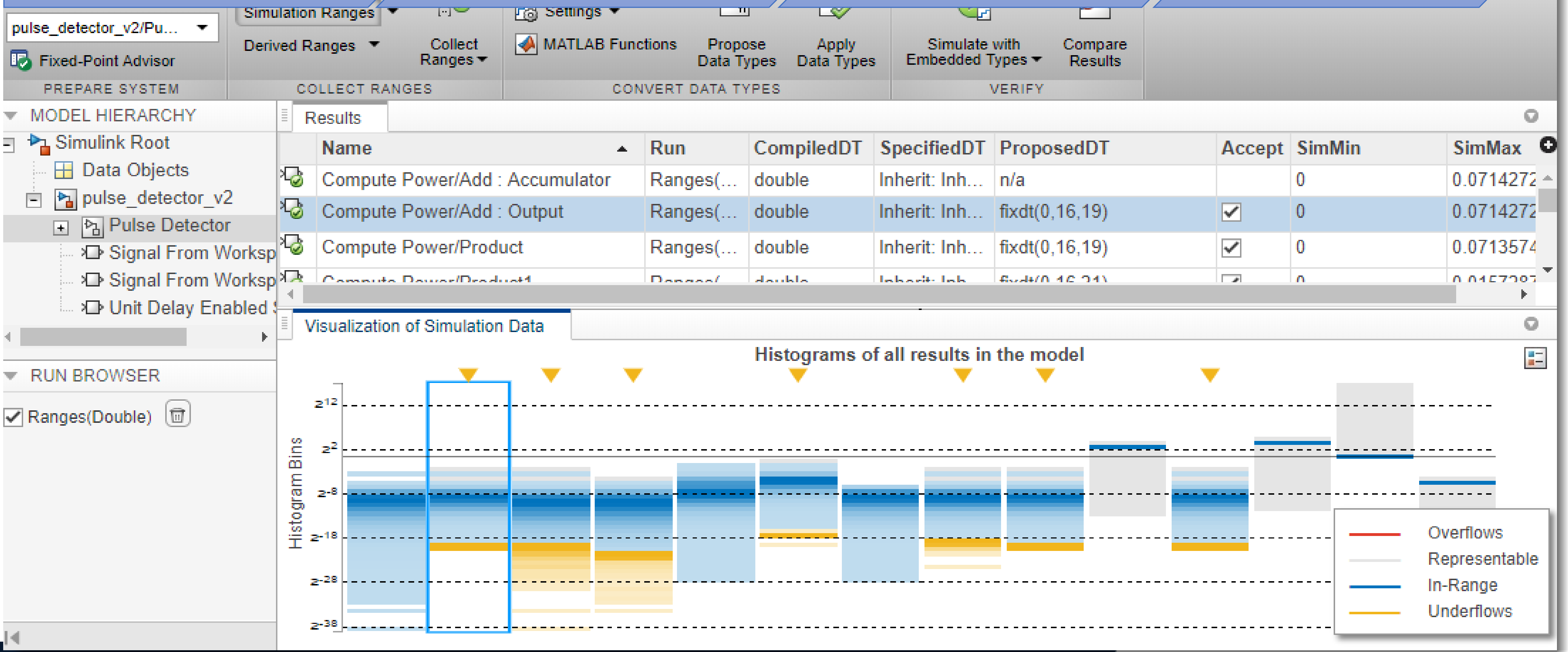
自动定点化：Fixed-Point Designer

仿真得出数据取值范围

Fixed-Point Designer 给出建议的数据类型

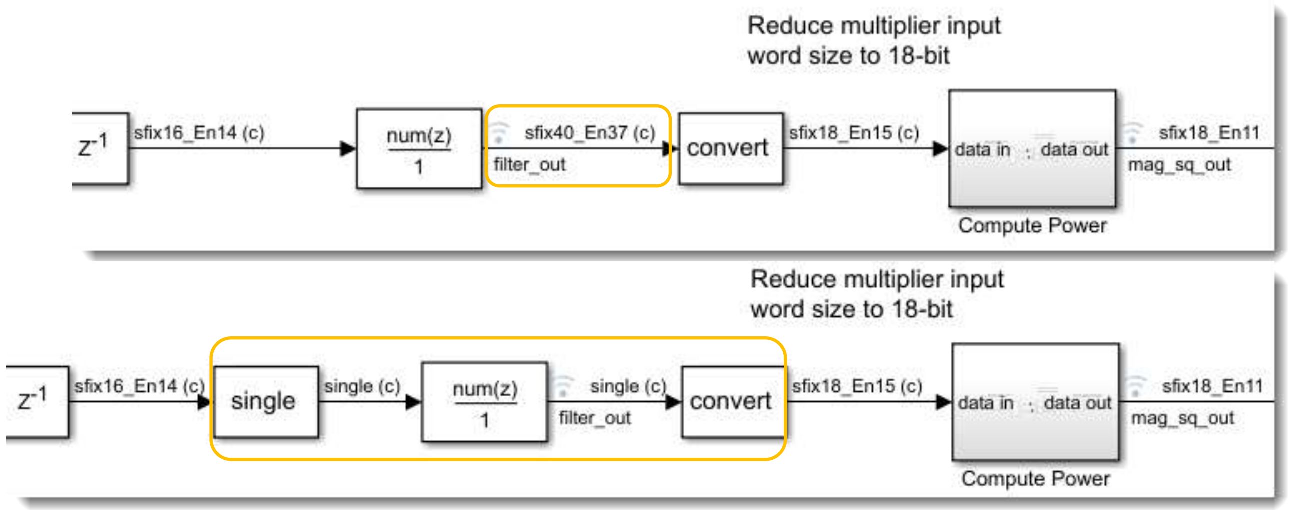
用户可接受建议的数据类型或自行指定数据类型

对定点化后的模型进行仿真并对比结果



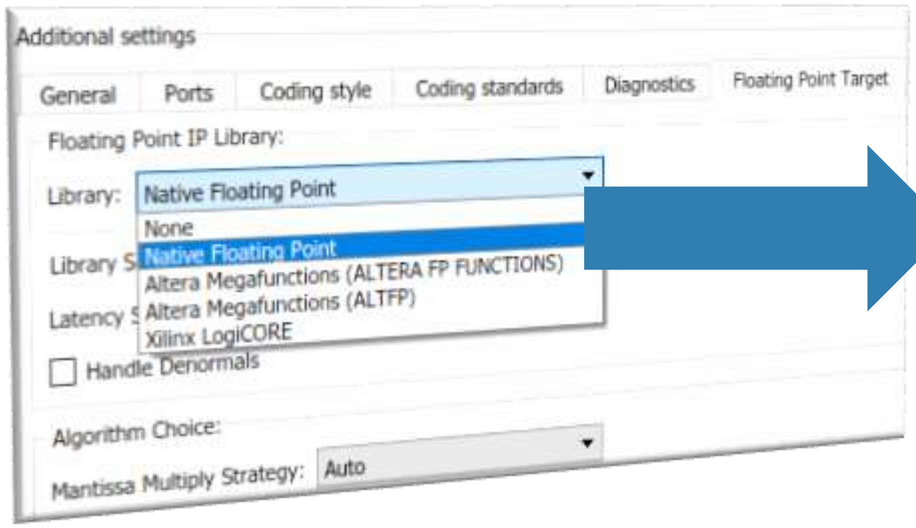
Name	Run	CompiledDT	SpecifiedDT	ProposedDT	Accept	SimMin	SimMax
Compute Power/Add : Accumulator	Ranges(...	double	Inherit: Inh...	n/a		0	0.0714272
Compute Power/Add : Output	Ranges(...	double	Inherit: Inh...	fixdt(0,16,19)	✓	0	0.0714272
Compute Power/Product	Ranges(...	double	Inherit: Inh...	fixdt(0,16,19)	✓	0	0.0713574
Compute Power/Product1	Ranges(...	double	Inherit: Inh...	fixdt(0,16,21)	✓	0	0.0157287

支持生成浮点代码



HDL Coder Native Floating Point

- 支持常见的数学和三角函数
- 优化实现并尽可能保证数值精度
- 浮点和定点运算可混合使用
- 生成通用的HDL代码，不依赖于特定器件


Additional settings

General | Ports | Coding style | Coding standards | Diagnostics | Floating Point Target

Floating Point IP Library:

Library: Native Floating Point
- None
- Native Floating Point
- Altera Megafunctions (ALTERA FP FUNCTIONS)
- Altera Megafunctions (ALTFP)
- Xilinx LogiCORE

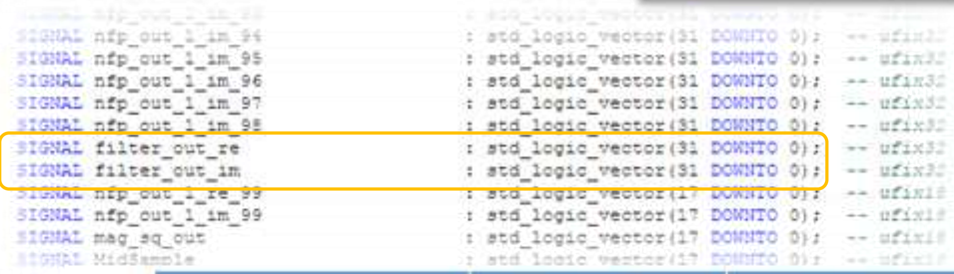
Library S

Latency S

☐ Handle Denormals

Algorithm Choice:

Mantissa Multiply Strategy: Auto


```
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
```

embeddedaward2017

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

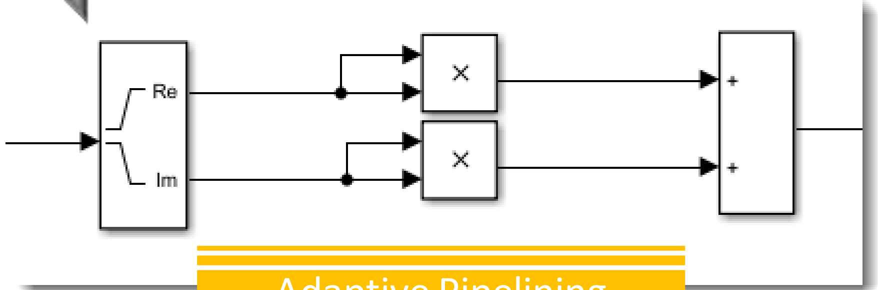
~2x more resources
~5x less development effort

DEMCON

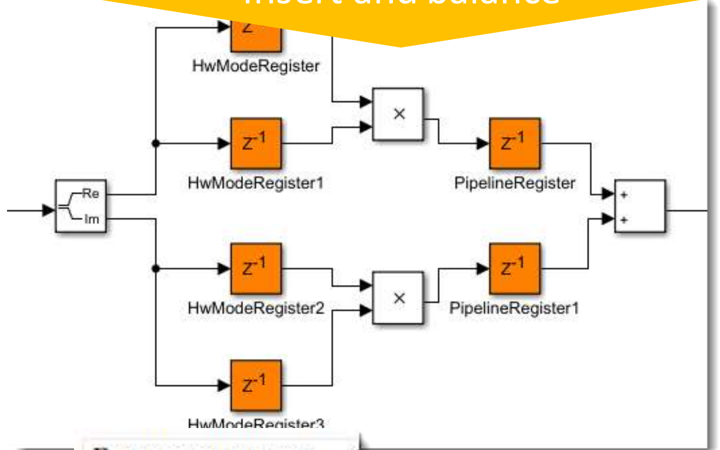
时序优化

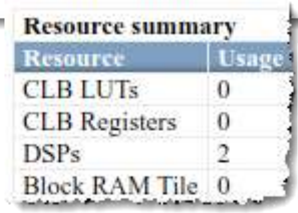
Automation

Control



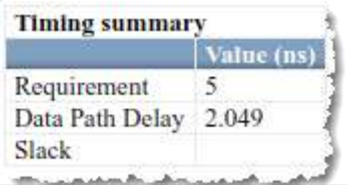
Adaptive Pipelining
Insert and balance





Resource summary

Resource	Usage
CLB LUTs	0
CLB Registers	0
DSPs	2
Block RAM Tile	0



Timing summary

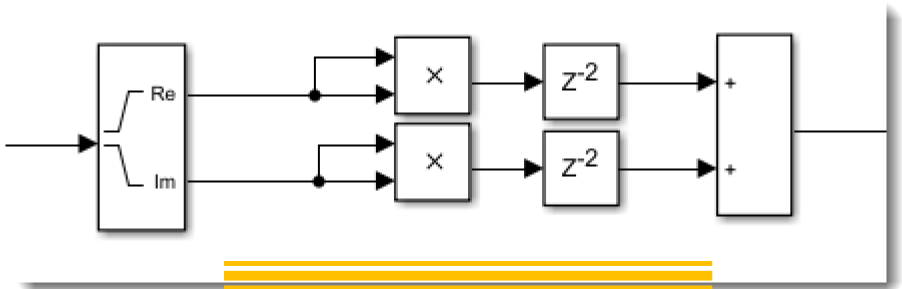
	Value (ns)
Requirement	5
Data Path Delay	2.049
Slack	

硬件架构设计

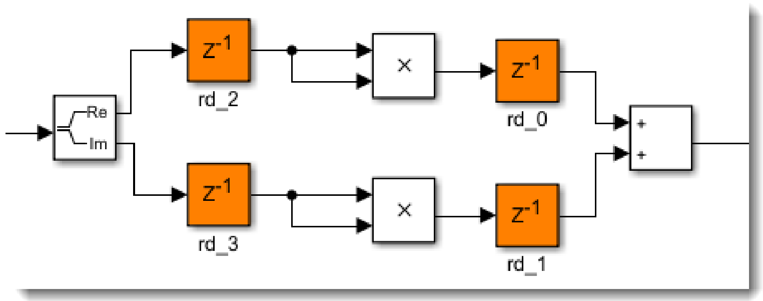
目标频率 =
200 MHz

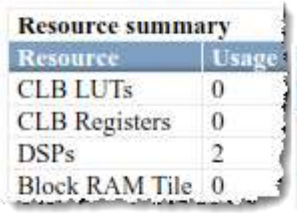
生成优化模型

综合与实现



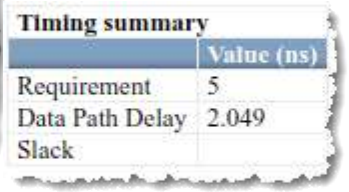
Distributed Pipelining
Balance only





Resource summary

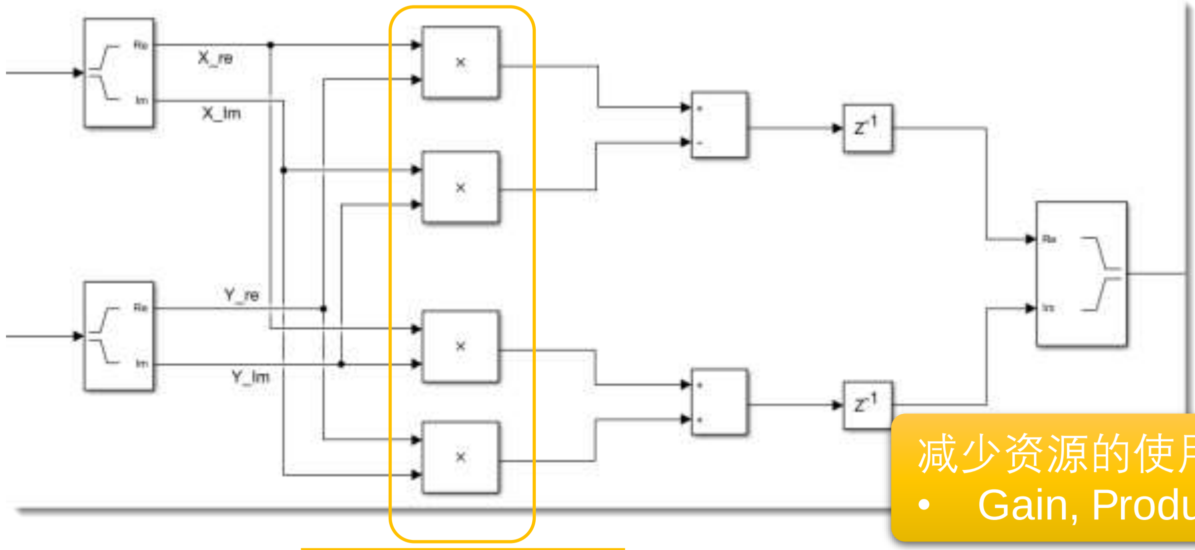
Resource	Usage
CLB LUTs	0
CLB Registers	0
DSPs	2
Block RAM Tile	0



Timing summary

	Value (ns)
Requirement	5
Data Path Delay	2.049
Slack	

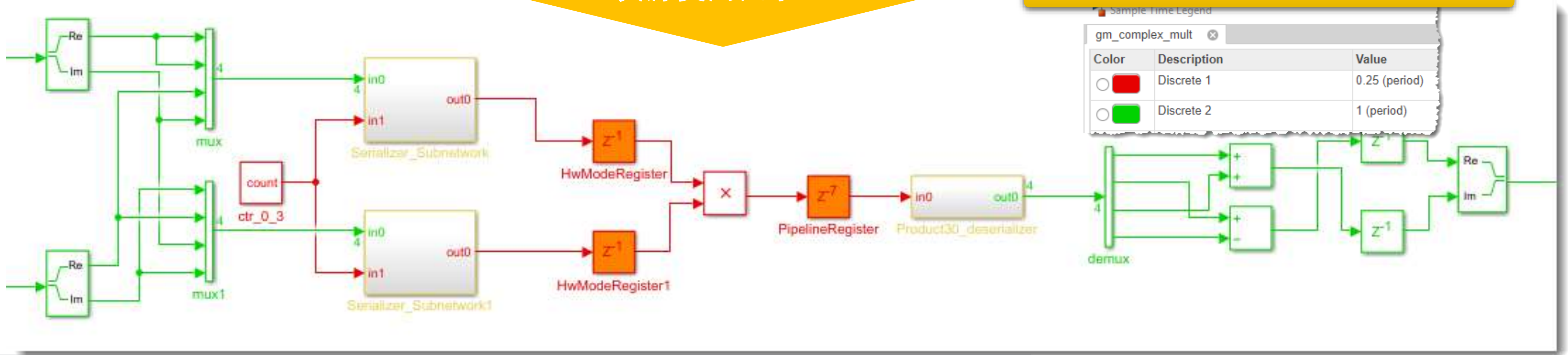
面积优化



资源复用因子=4

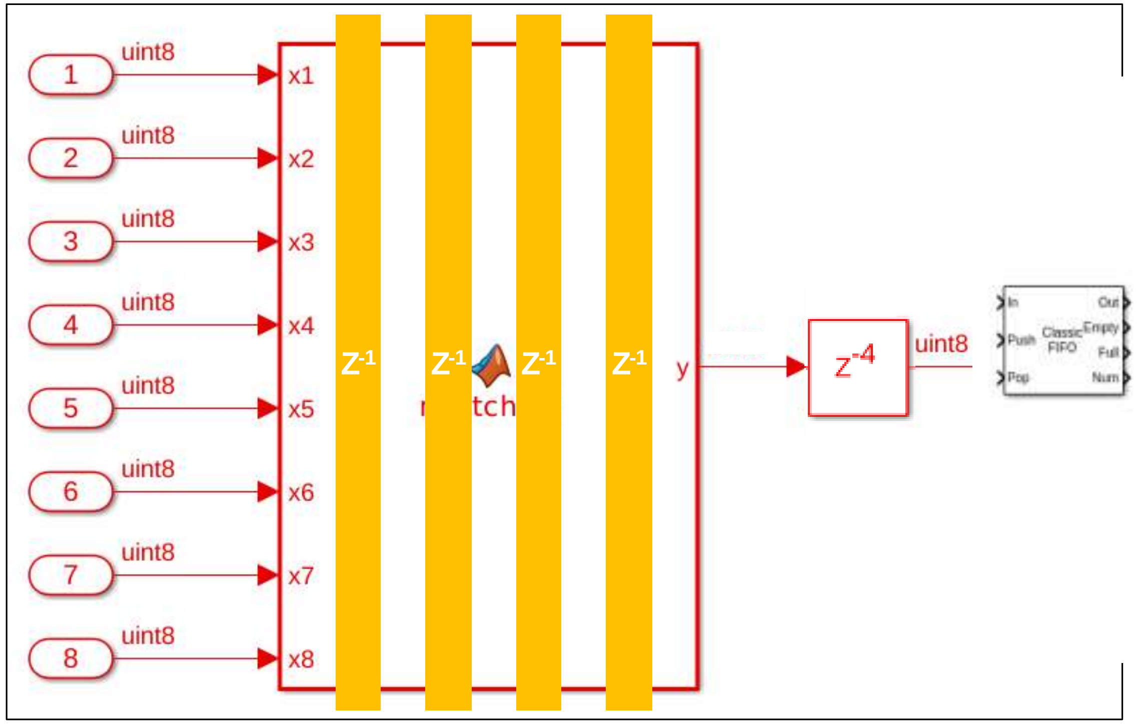
减少资源的使用，例如
- Gain, Product, Multiply-Add

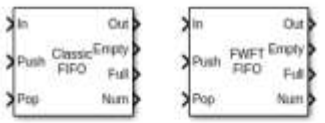
自动添加分时复用逻辑



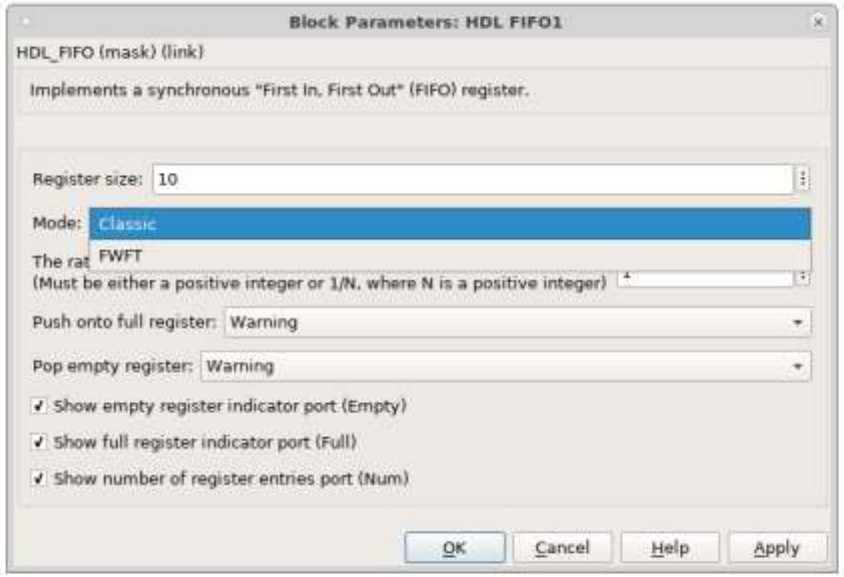
Color	Description	Value
🟥	Discrete 1	0.25 (period)
🟩	Discrete 2	1 (period)

HDL Coder 新特性举例





UltraRAM
(100s of megabits)



Block Parameters: HDL FIFO1

HDL_FIFO (mask) (link)

Implements a synchronous "First In, First Out" (FIFO) register.

Register size: 10

Mode: Classic

FWFT

The rat

(Must be either a positive integer or 1/N, where N is a positive integer) 1

Push onto full register: Warning

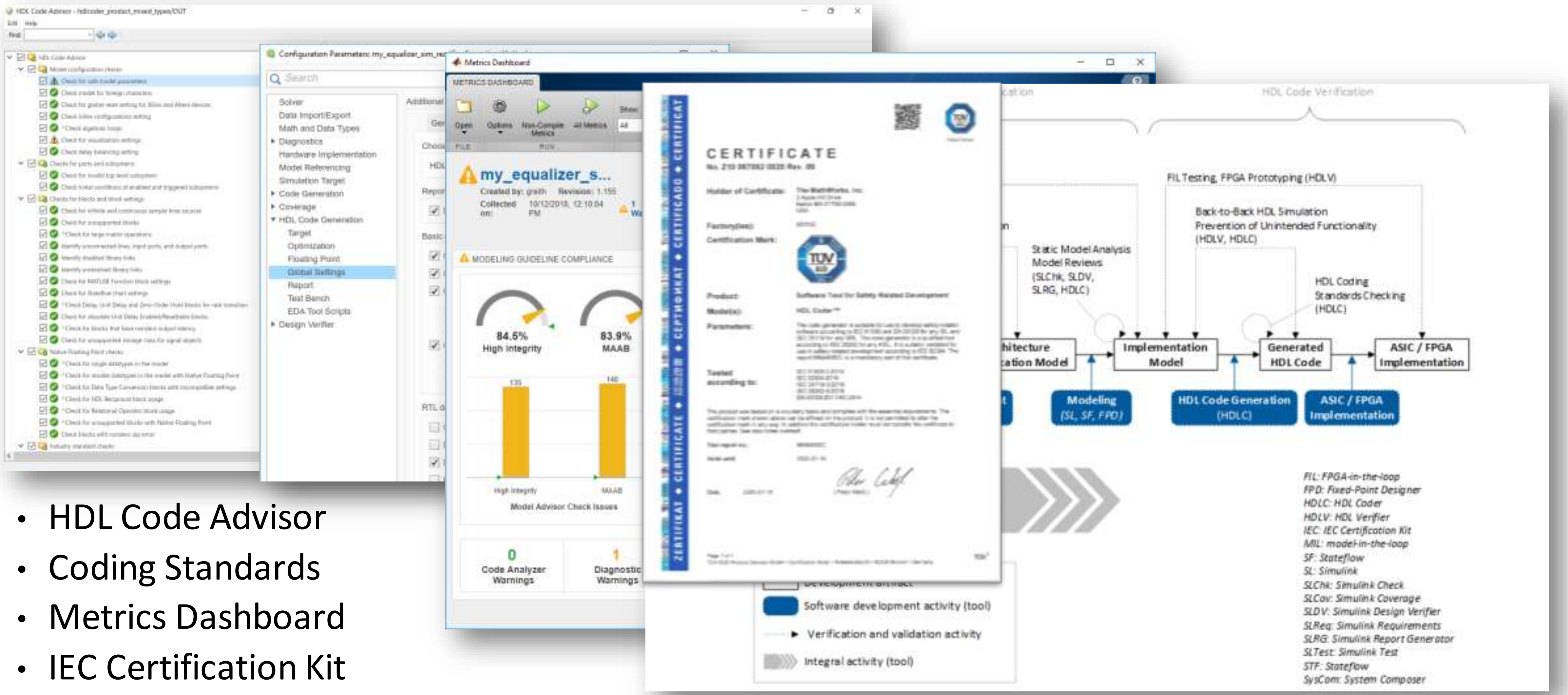
Pop empty register: Warning

✓ Show empty register indicator port (Empty)

✓ Show full register indicator port (Full)

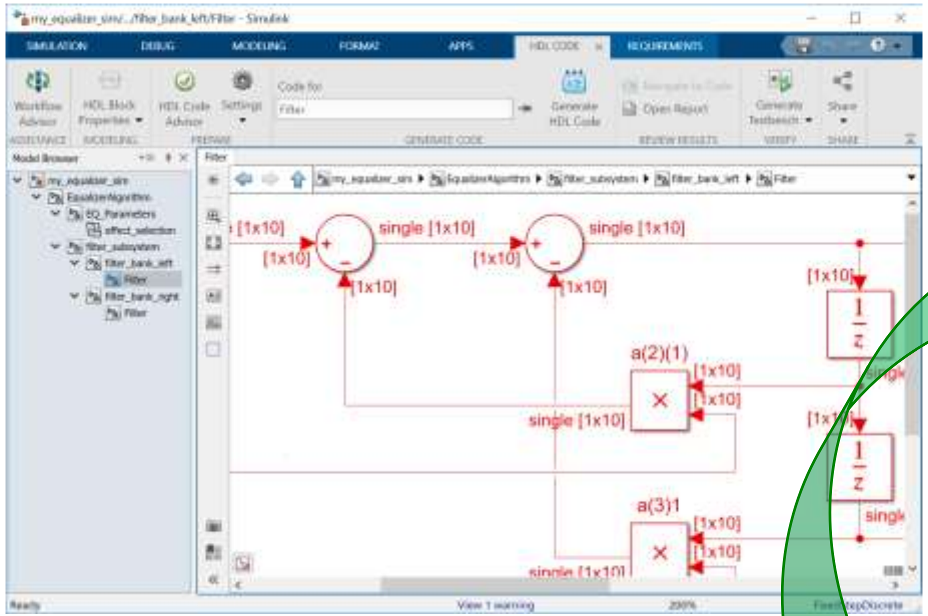
✓ Show number of register entries port (Num)

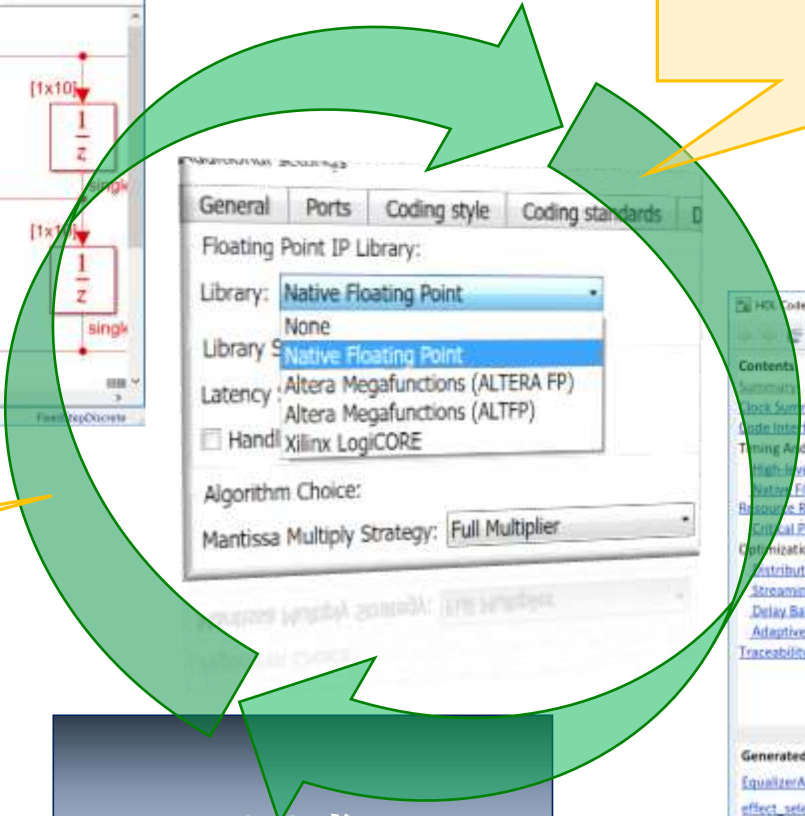
模型自动检查、HDL代码标准和认证



- HDL Code Advisor
- Coding Standards
- Metrics Dashboard
- IEC Certification Kit

HDL代码生成和追溯





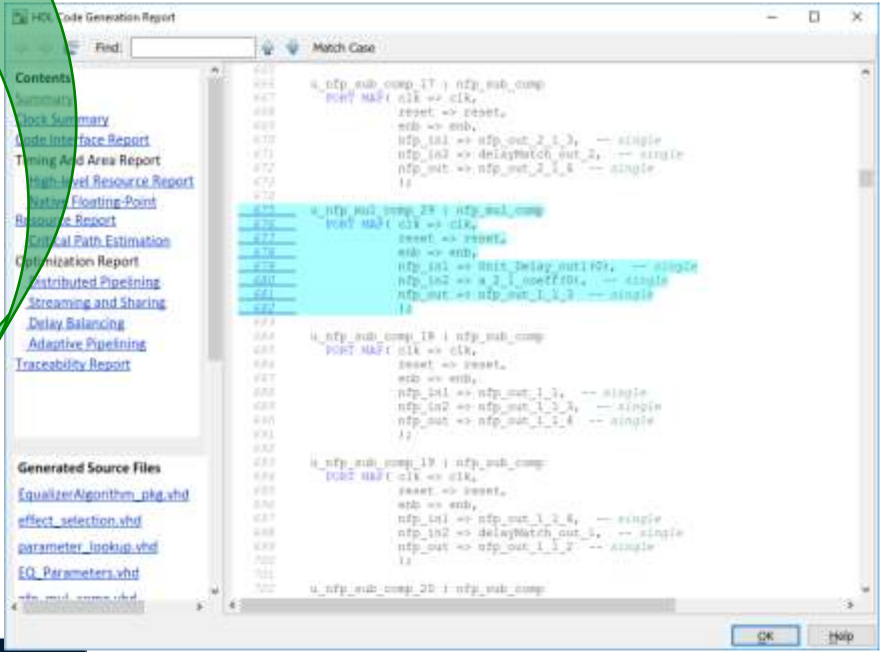
Library: Native Floating Point

Mantissa Multiply Strategy: Full Multiplier

HDL代码生成

双向追溯

需求

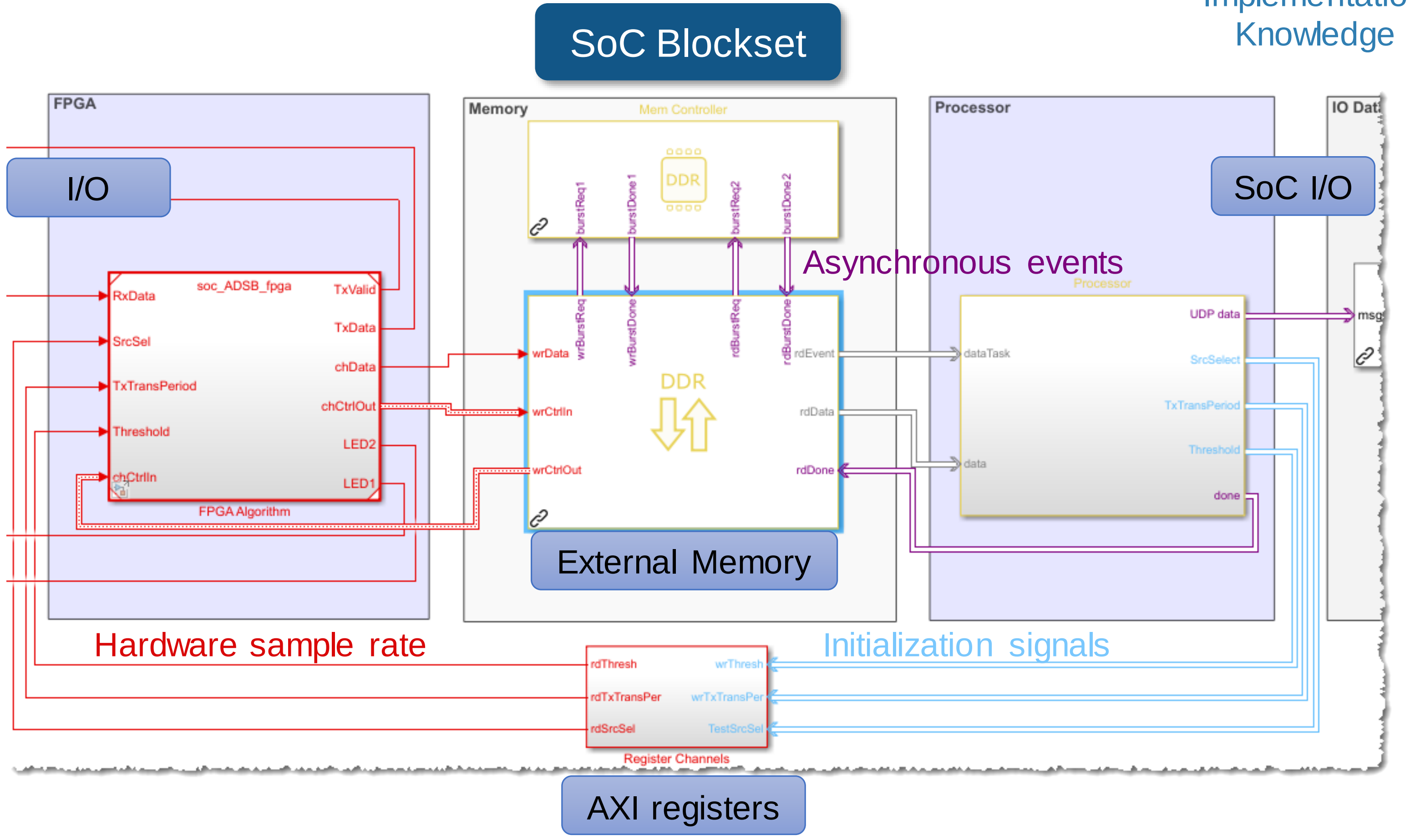


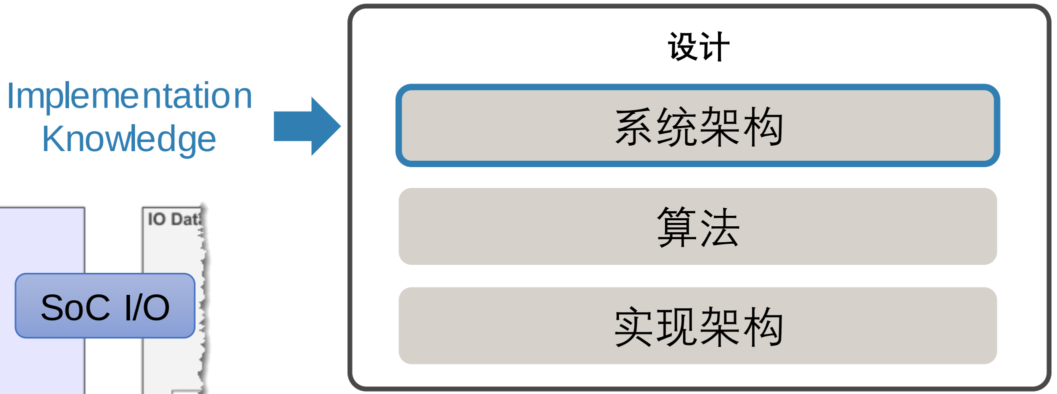
```
    u_nfp_sub_comp_17 : nfp_sub_comp
      PORT MAP( clk => clk,
                reset => reset,
                enb => enb,
                nfp_in1 => nfp_out_2_1_3,  -- single
                nfp_in2 => delayMatch_out_2,  -- single
                nfp_out => nfp_out_2_1_4  -- single
                );

    u_nfp_mul_comp_29 : nfp_mul_comp
      PORT MAP( clk => clk,
                reset => reset,
                enb => enb,
                nfp_in1 => Unit_Delay_out1(0),  -- single
                nfp_in2 => a_2_1_coeff(0),  -- single
                nfp_out => nfp_out_1_1_3  -- single
                );

    u_nfp_sub_comp_18 : nfp_sub_comp
      PORT MAP( clk => clk,
                reset => reset,
                enb => enb,
                nfp_in1 => nfp_out_1_1,  -- single
                nfp_in2 => nfp_out_1_1_3,  -- single
                nfp_out => nfp_out_1_1_4  -- single
                );

    u_nfp_sub_comp_19 : nfp_sub_comp
      PORT MAP( clk => clk,
                reset => reset,
                enb => enb,
                nfp_in1 => nfp_out_1_2_6,  -- single
                nfp_in2 => delayMatch_out_1,  -- single
                nfp_out => nfp_out_1_1_2  -- single
                );

    u_nfp_sub_comp_20 : nfp_sub_comp
```

MATLAB EXPO

MathWorks®

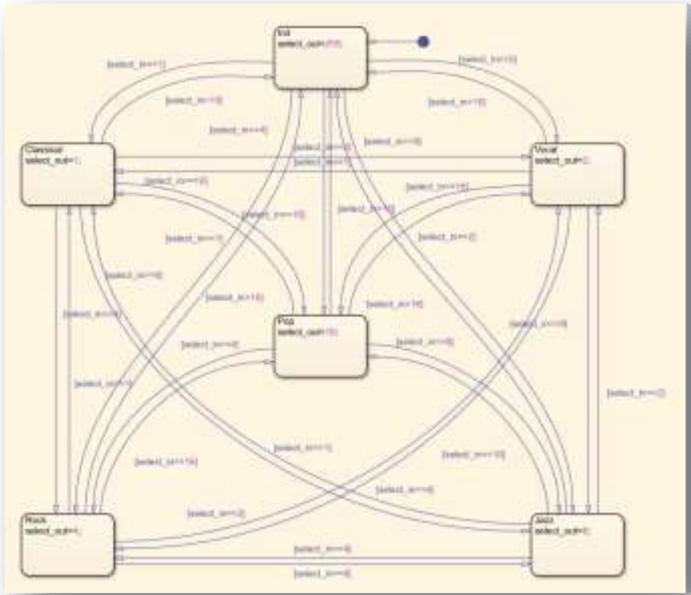
SoC应用架构建模与仿真：SoC Blockset

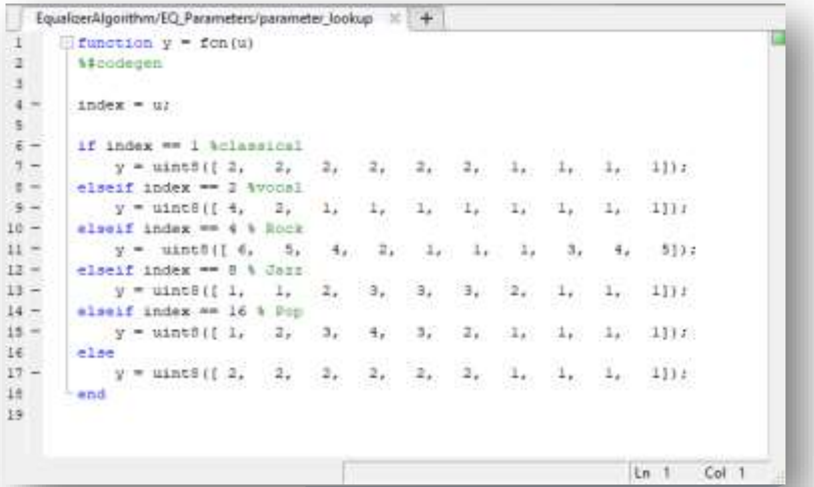




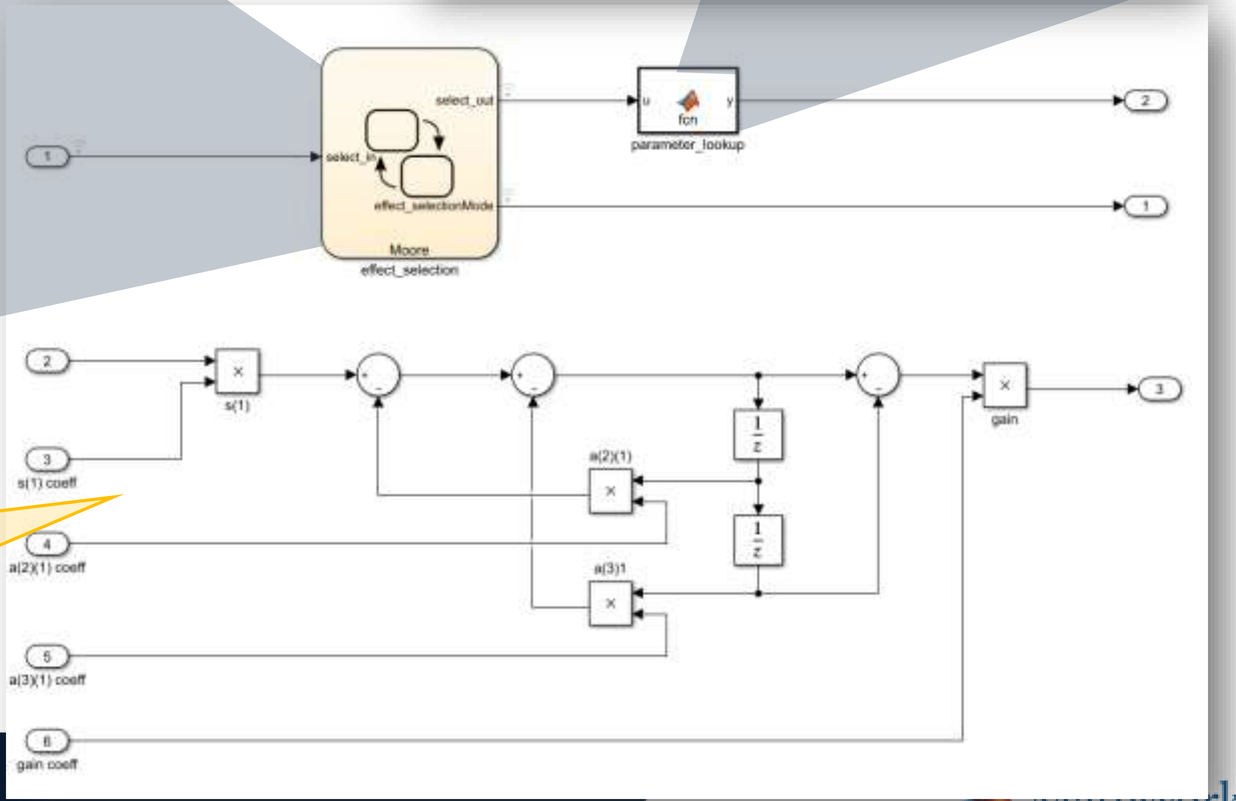
- 对组件行为和延迟的仿真
  - ✓ 算法、存储、内部和外部连接
  - ✓ 调度和操作系统
  - ✓ 实时流式I/O数据
- 软件性能和硬件利用率
- 获得对算法调整的依据

支持生成HDL代码的Simulink模块





```
function y = fcn(u)
%#codegen

index = u;

if index == 1 %classical
    y = uint8([ 2,   2,   2,   2,   2,   2,   1,   1,   1,   1]);
elseif index == 2 %vocal
    y = uint8([ 4,   2,   1,   1,   1,   1,   1,   1,   1,   1]);
elseif index == 4 % Rock
    y =  uint8([ 6,   5,   4,   2,   1,   1,   1,   3,   4,   5]);
elseif index == 8 % Jazz
    y = uint8([ 1,   1,   2,   3,   3,   3,   2,   1,   1,   1]);
elseif index == 16 % Pop
    y = uint8([ 1,   2,   3,   4,   3,   2,   1,   1,   1,   1]);
else
    y = uint8([ 2,   2,   2,   2,   2,   2,   1,   1,   1,   1]);
end
```

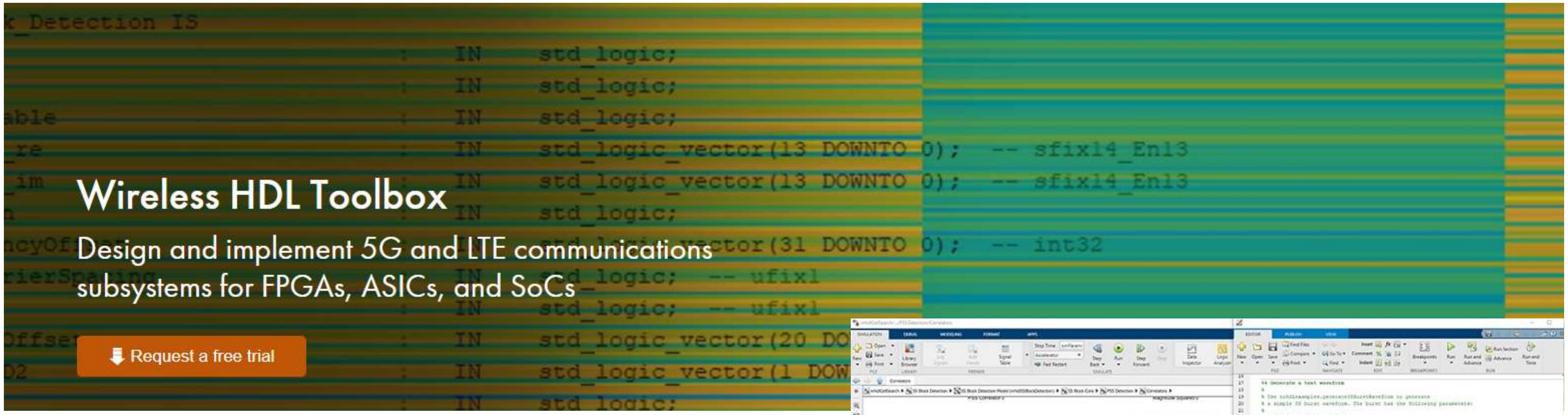


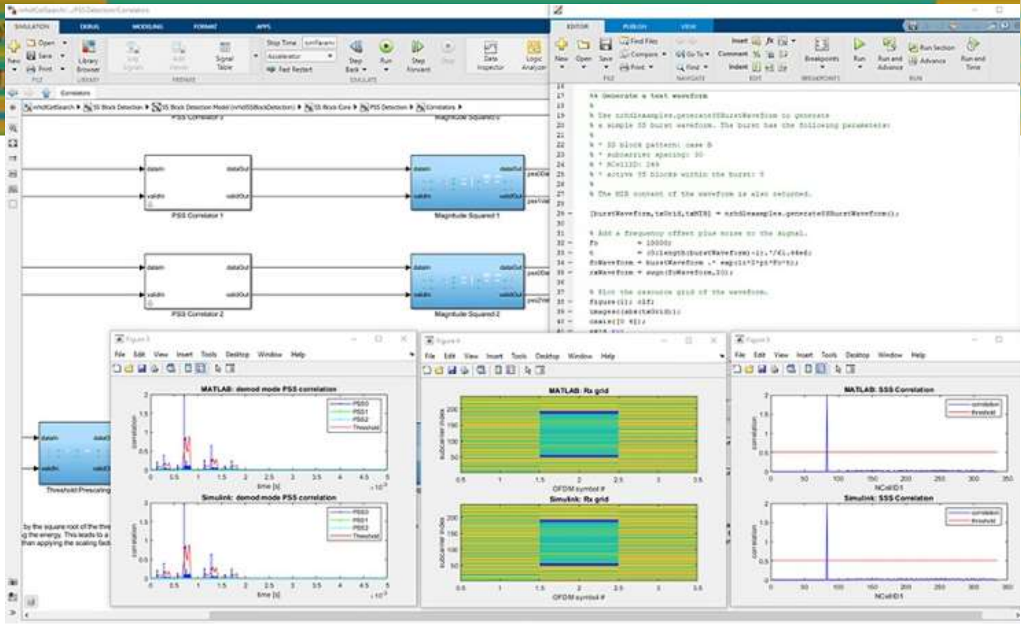
针对HDL优化实现的模块

MATLAB EXPO

MathWorks®

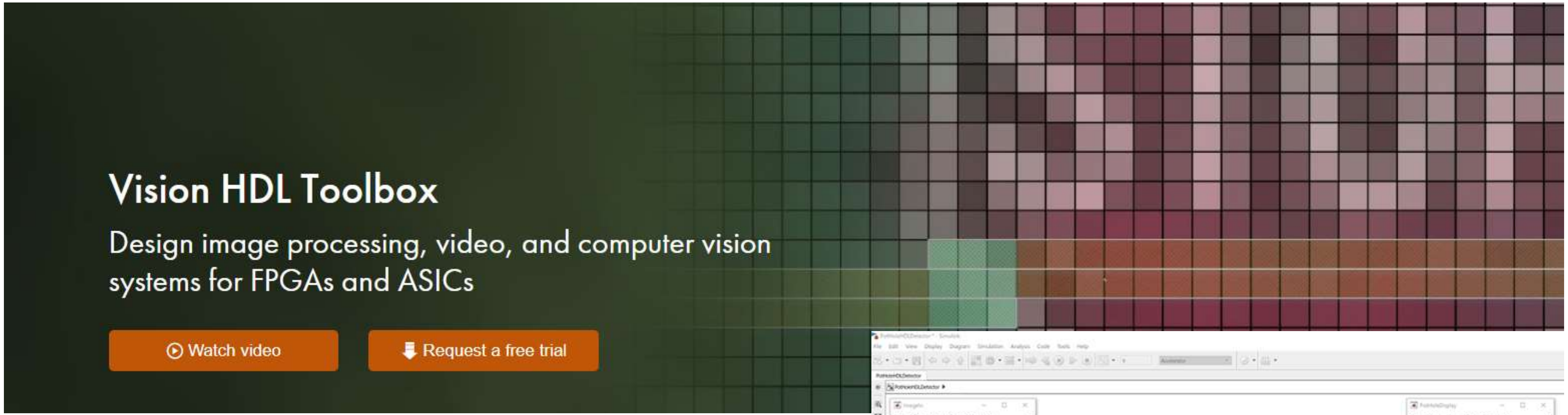
Wireless HDL Toolbox


Wireless HDL Toolbox
Design and implement 5G and LTE communications subsystems for FPGAs, ASICs, and SoCs
Request a free trial

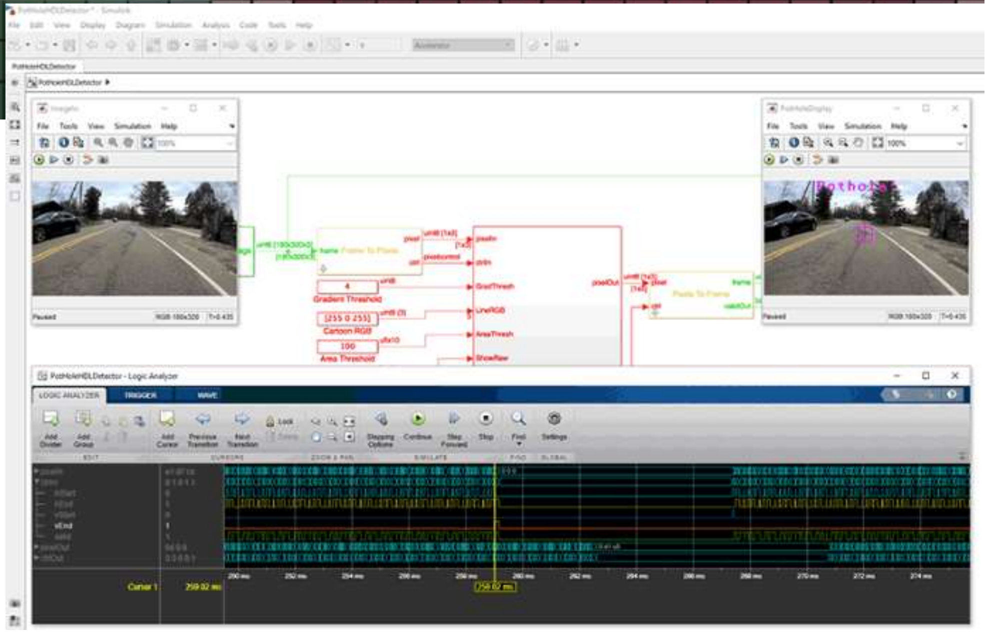

MATLAB: demod mode PSS correlation
Simulink: demod mode PSS correlation
MATLAB: Rx grid
Simulink: Rx grid
MATLAB: SSS Correlation
Simulink: SSS Correlation

- 5G、LTE 和 Wireless IP 模块
- 使用 5G 或 LTE golden reference进行验证
- FPGA、ASIC及SoC部署
- 硬件子系统参考应用提升设计效率
- 在 FPGA 上部署 5G NR 无线通信：一套完整的 MATLAB 与 Simulink 工作流程

Vision HDL Toolbox


Vision HDL Toolbox

Design image processing, video, and computer vision systems for FPGAs and ASICs



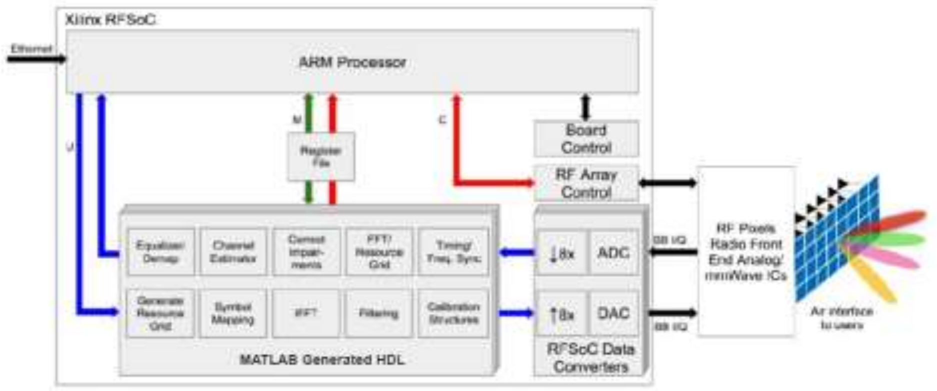
- 基于帧的算法验证
- 流式硬件实现
- FPGA、ASIC及SoC部署
- 高效的图像处理IP
- Vision Processing for FPGA

  https://ww2.mathworks.cn/videos/series/vision-processing-for-fpga.html

RF Pixels使用HDL Coder加速算法实现并在Zynq RFSoC上快速部署

- 挑战
  - ✓ 对包含了专用射频电子和毫米波器件的无线电前端设计进行验证
- 方案
  - ✓ 使用MATLAB和Simulink实现数字基带，并将其部署到Zynq RFSoC开发板上进行无线测试
- 结果
  - ✓ 极大节省了工程实现上的工作量
  - ✓ 设计迭代时间从几周减少到几天

https://ww2.mathworks.cn/company/newsletters/articles/verifying-millimeter-wave-rf-electronics-on-a-zynq-rfsoc-based-digital-baseband.html



"By adapting the LTE golden reference model from Wireless HDL Toolbox and deploying it to a Zynq UltraScale+ RFSoC board using HDL Coder, we saved us at least a year of engineering effort—and this approach enabled me to complete the implementation myself, without having to hire an additional digital engineer."

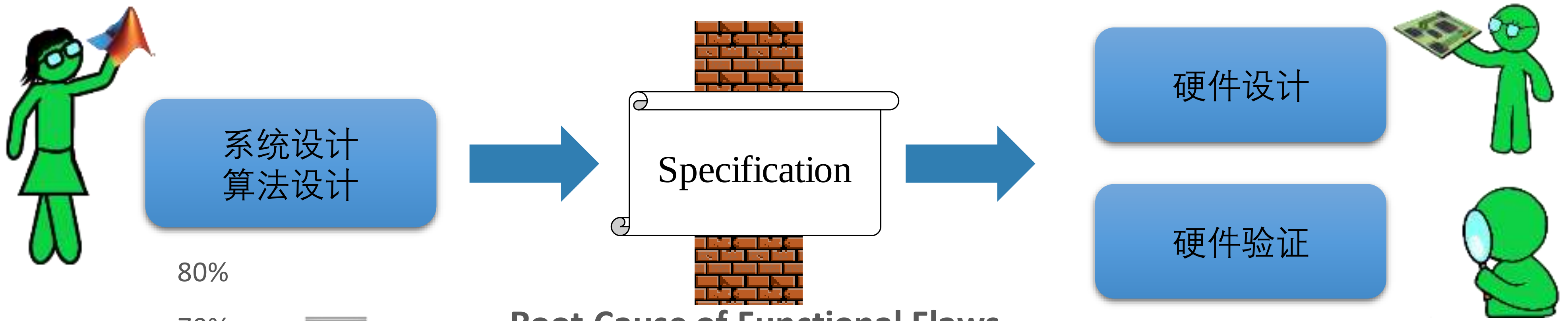
- Matthew Weiner, RF Pixels

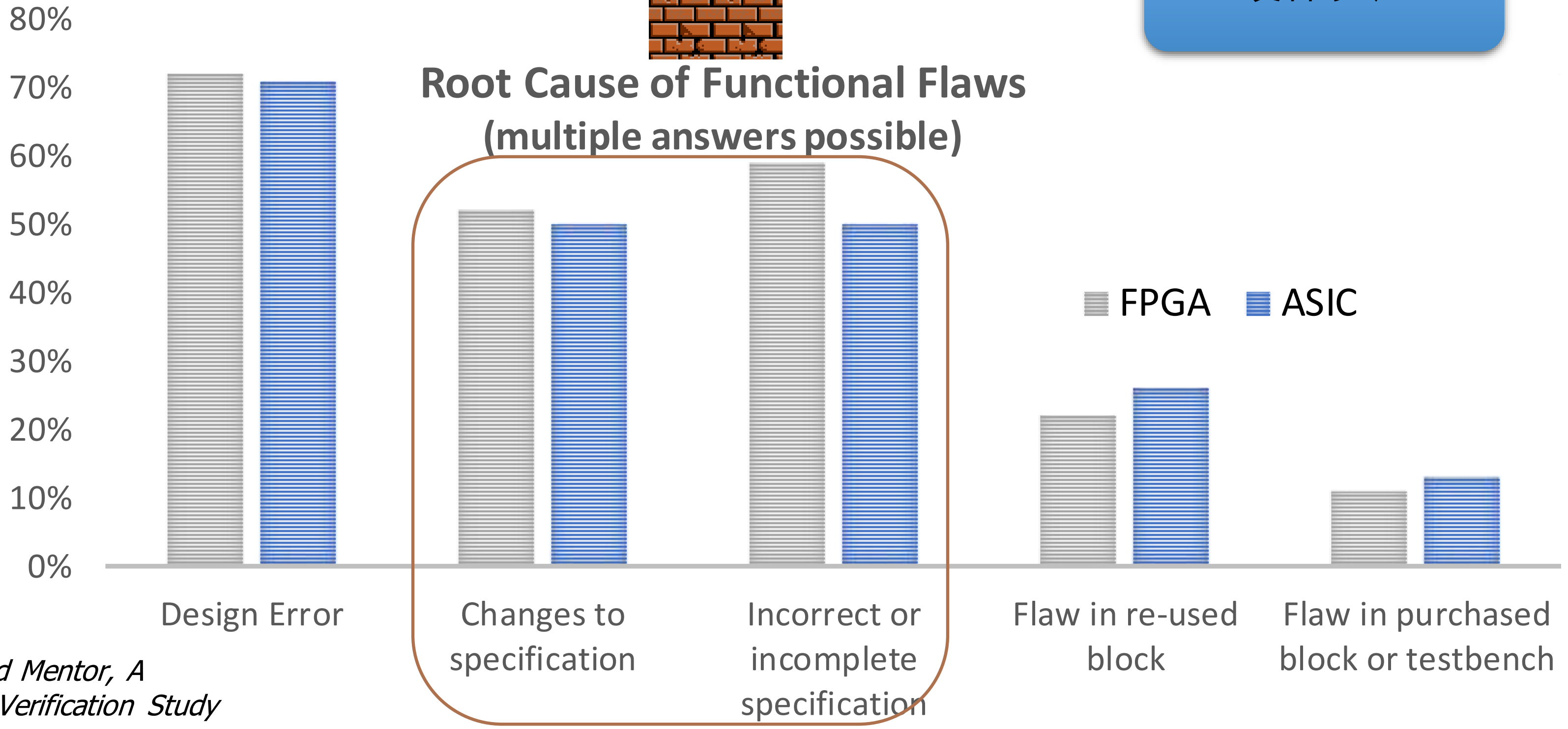
Agenda

- 基于模型的FPGA开发
- HDL代码生成
- **FPGA验证**

HDL Verifier
Test and verify Verilog and VHDL using HDL simulators and FPGA boards
Watch video
Request a free trial

造成硬件设计功能缺陷的原因



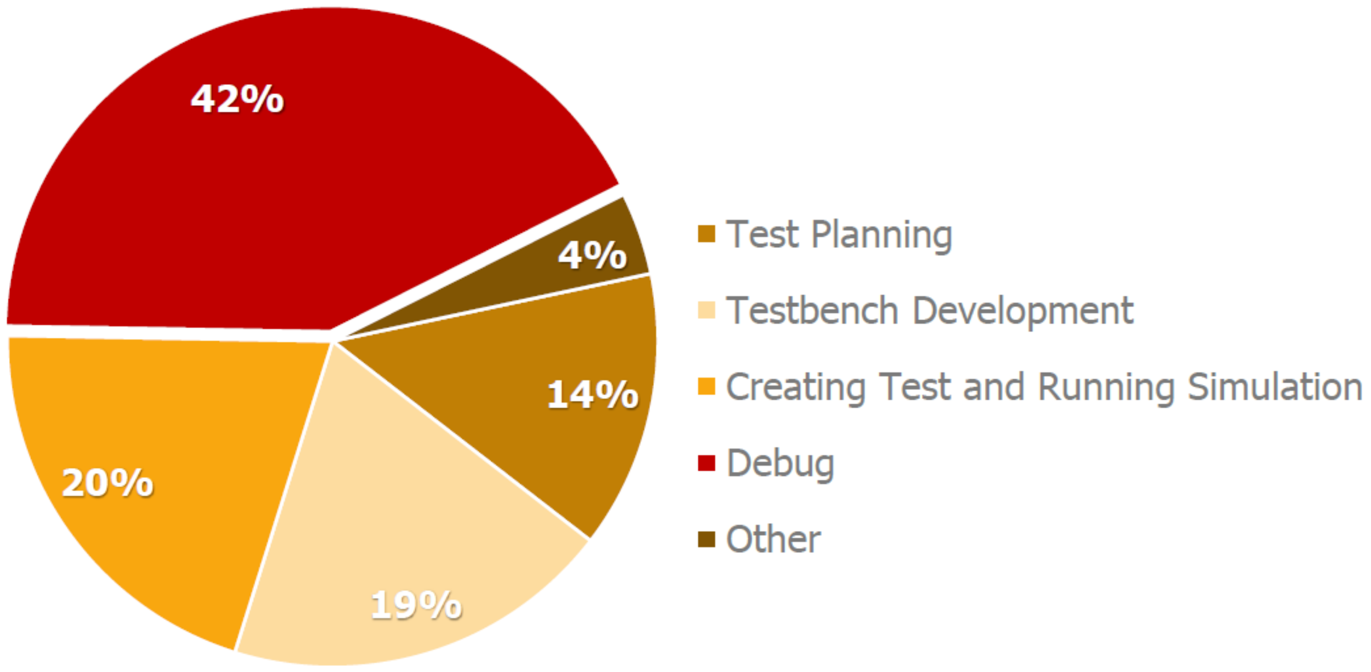


Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study

FPGA验证工程师的时间分配

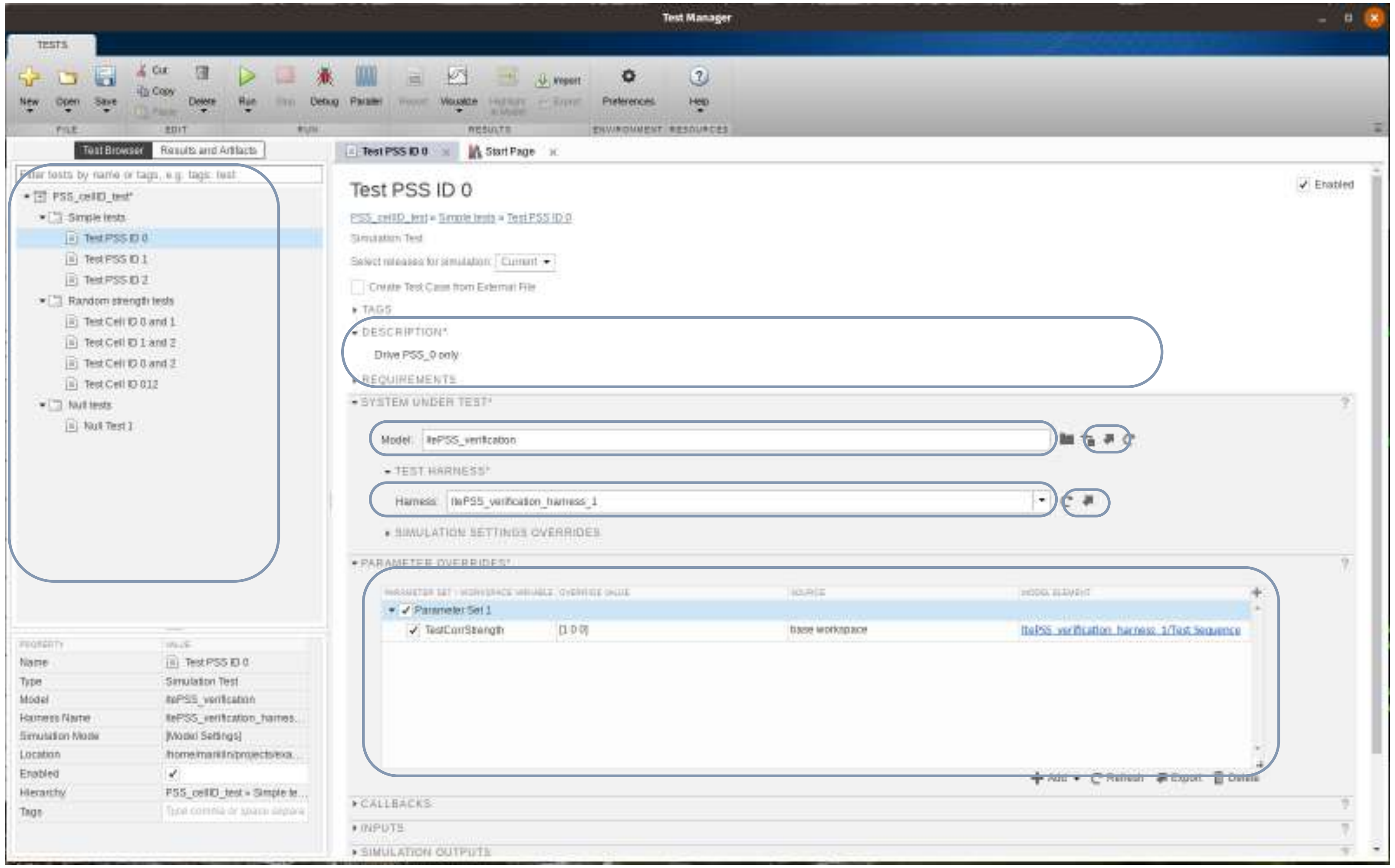
FPGA: Where Verification Engineers Spend Their Time

<u>Project Time Spent in Verification:</u>
2012: Average 44%
2014: Average 46%
2016: Average 49%
2018: Average 50%



Source: Wilson Research Group and Mentor, A Siemens Business, 2018 Functional Verification Study

Simulink Test



Filter tests by name or tags, e.g. tags: test

- PSS_cellID_test*
  - Simple tests
    - Test PSS ID 0
    - Test PSS ID 1
    - Test PSS ID 2
  - Random strength tests
    - Test Cell ID 0 and 1
    - Test Cell ID 1 and 2
    - Test Cell ID 0 and 2
    - Test Cell ID 012
  - Null tests
    - Null Test 1

PROPERTY	VALUE
Name	Test PSS ID 0
Type	Simulation Test
Model	ltePSS_verification
Harness Name	ltePSS_verification_harnes...
Simulation Mode	[Model Settings]
Location	/home/marklin/projects/exa...
Enabled	✓
Hierarchy	PSS_cellID_test » Simple te...
Tags	Type comma or space separa

Test PSS ID 0

✓ Enabled

PSS_cellID_test » Simple tests » Test PSS ID 0

Simulation Test

Select releases for simulation: Current

☐ Create Test Case from External File

▸ TAGS

▾ DESCRIPTION*

Drive PSS_0 only

▸ REQUIREMENTS

▾ SYSTEM UNDER TEST*

Model: ltePSS_verification

▾ TEST HARNESS*

Harness: ltePSS_verification_harness_1

▸ SIMULATION SETTINGS OVERRIDES

▾ PARAMETER OVERRIDES*

PARAMETER SET / WORKSPACE VARIABLE	OVERRIDE VALUE	SOURCE	MODEL ELEMENT
✓ Parameter Set 1			
✓ TestCorrStrength	[1 0 0]	base workspace	ltePSS_verification_harness_1/Test Sequence

Add ▾ Refresh Export Delete

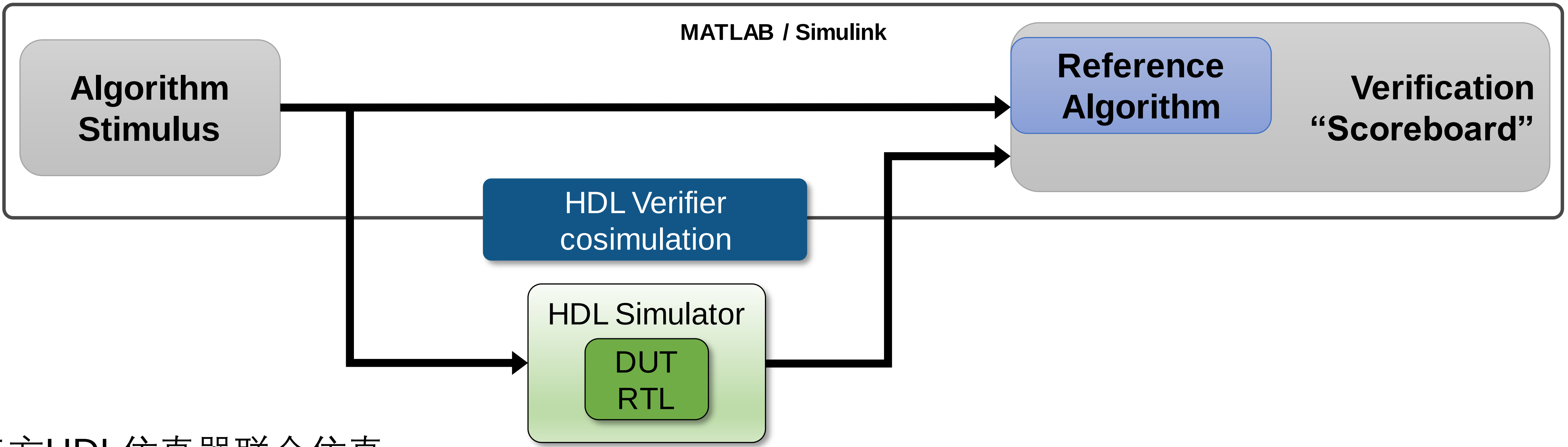
▸ CALLBACKS

▸ INPUTS

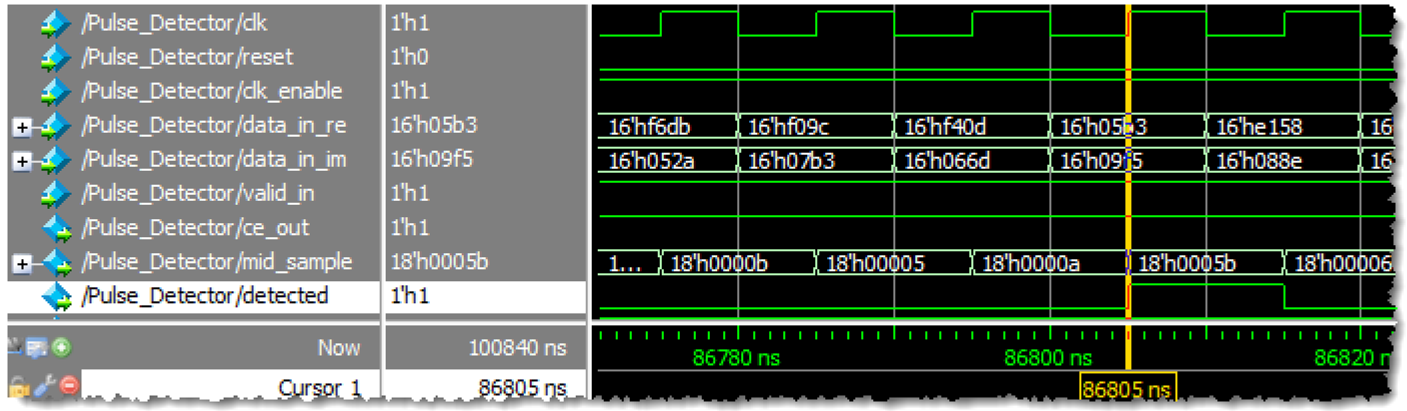
▸ SIMULATION OUTPUTS

- 全部测试概览
- 测试描述
- 链接到Simulink模型
- 链接到test harness
- 指向测试计划文档
- 测试参数

HDL Cosimulation



- 与第三方HDL仿真器联合仿真
  - ✓ 复用已有的 MATLAB/Simulink 测试环境
  - ✓ 在支持的仿真器中运行HDL代码
  - ✓ 生成联合仿真所需的接口和交互信号
  - ✓ 对设计和测试环境的分析
- 使用 HDL 协同仿真验证 Viterbi 解码器

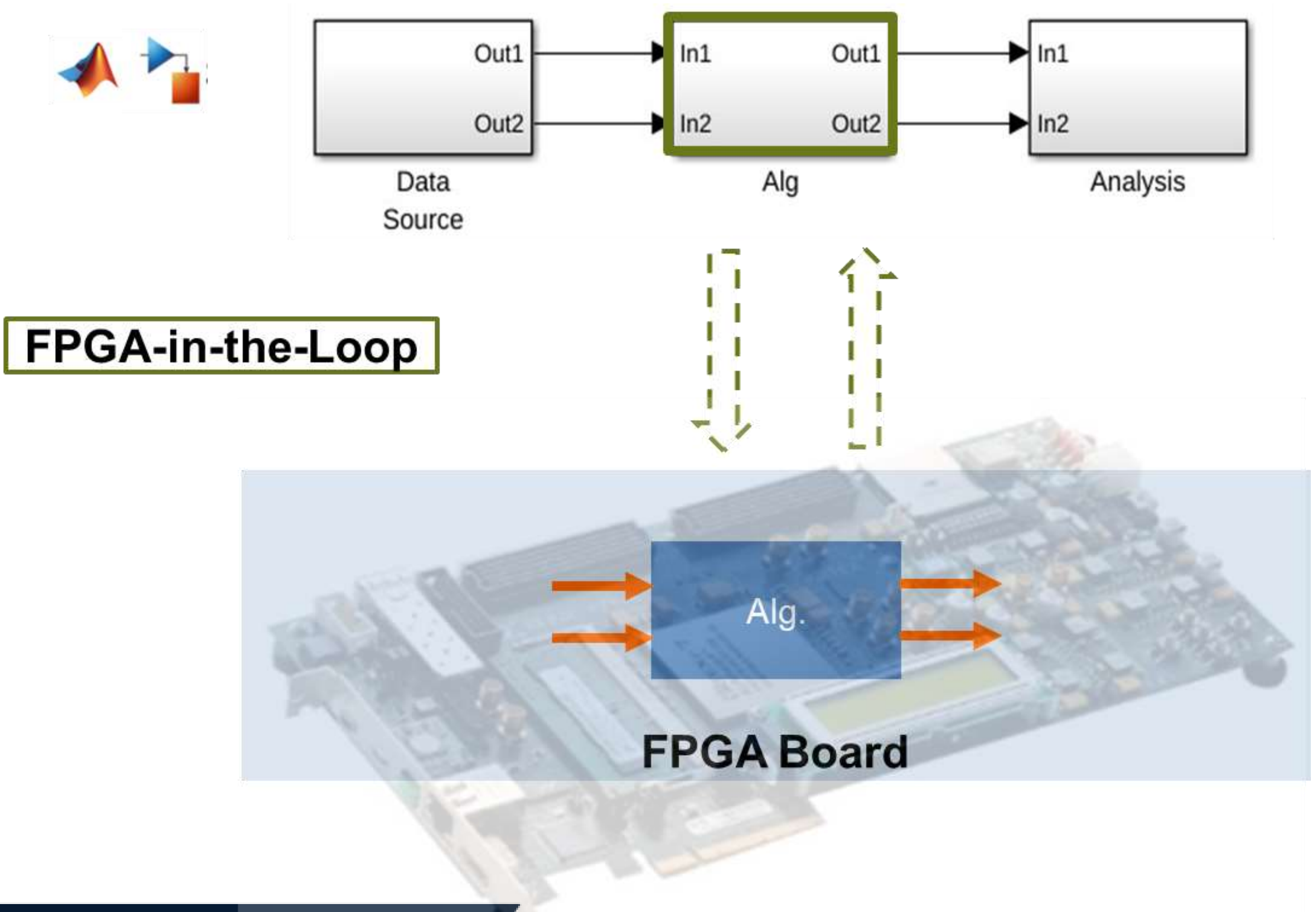


```
/Pulse_Detector/clk           1'h1
/Pulse_Detector/reset         1'h0
/Pulse_Detector/clk_enable    1'h1
/Pulse_Detector/data_in_re    16'h05b3    16'hf6db  16'hf09c  16'hf40d  16'h05b3  16'he158  16
/Pulse_Detector/data_in_im    16'h09f5    16'h052a  16'h07b3  16'h066d  16'h09f5  16'h088e  16
/Pulse_Detector/valid_in      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 ns
Cursor 1                      86805 ns    86805 ns
```

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

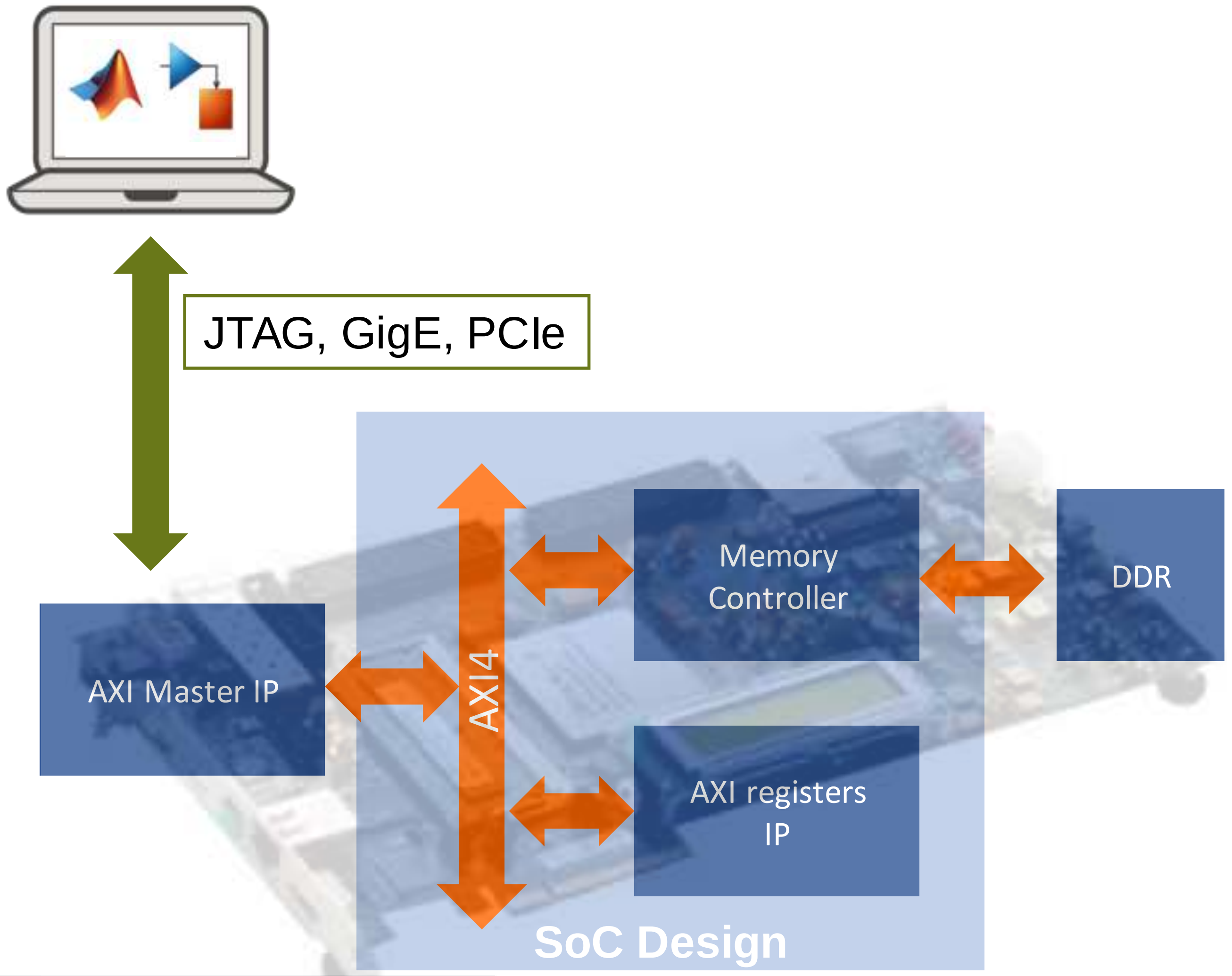
FPGA-in-the-Loop (FIL)

- 连接FPGA开发板进行在环仿真
  - ✓ 复用现有的ML/SL testbench
  - ✓ HDL生成网表后下载到FPGA中运行
  - ✓ 支持生成的代码和手写代码
  - ✓ 自动生成在环仿真所需的Ethernet/JTAG/PCIe接口
- [使用 FPGA 在环加速通信系统仿真]



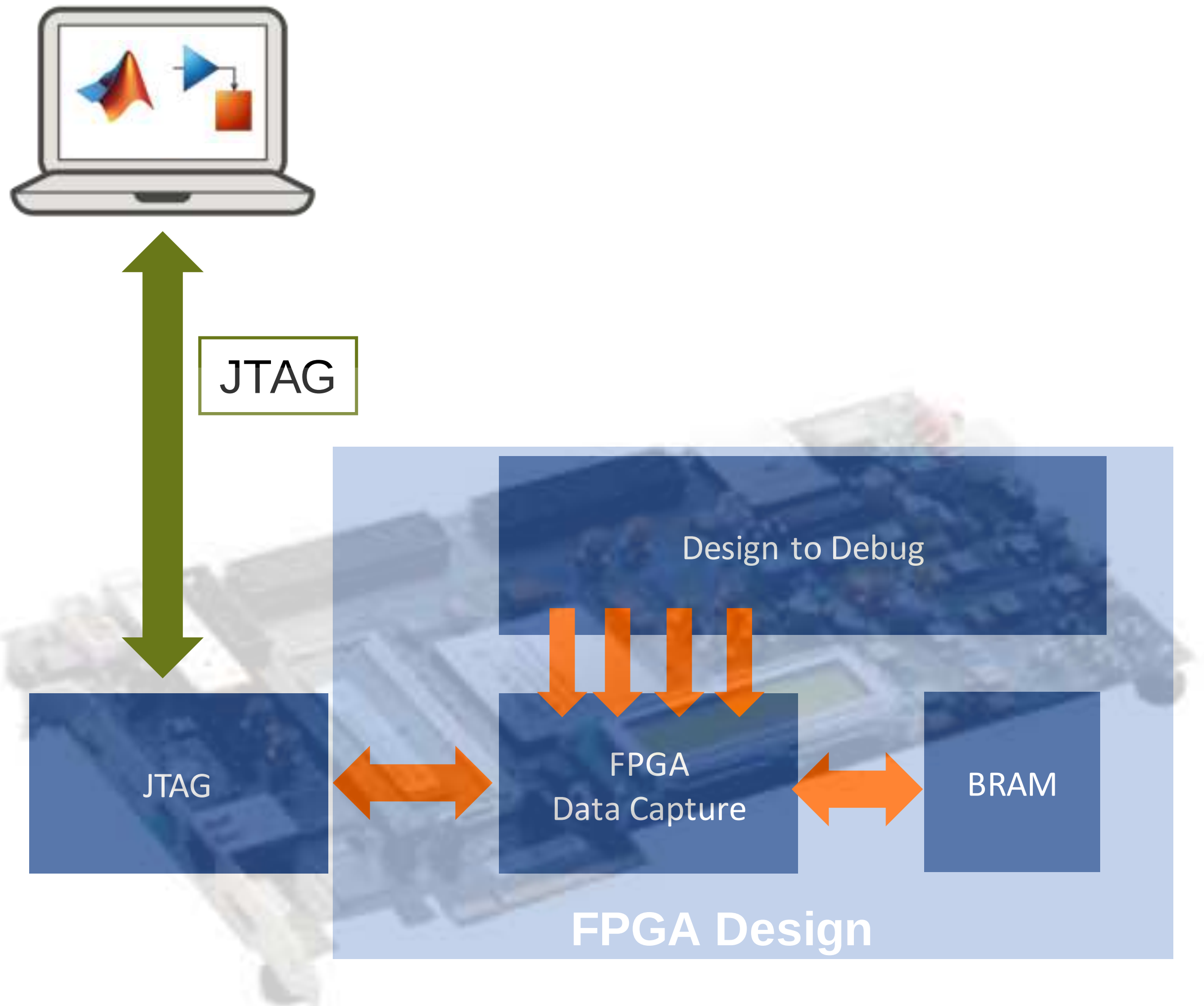
快速调试： AXI Master

- 连接MATLAB与FPGA/SoC硬件，进行数据分析和参数整定
- 将数据预加载到外部存储器， 以便与IP构成一个完整的系统使用
- 与MathWorks HDL workflows直接集成



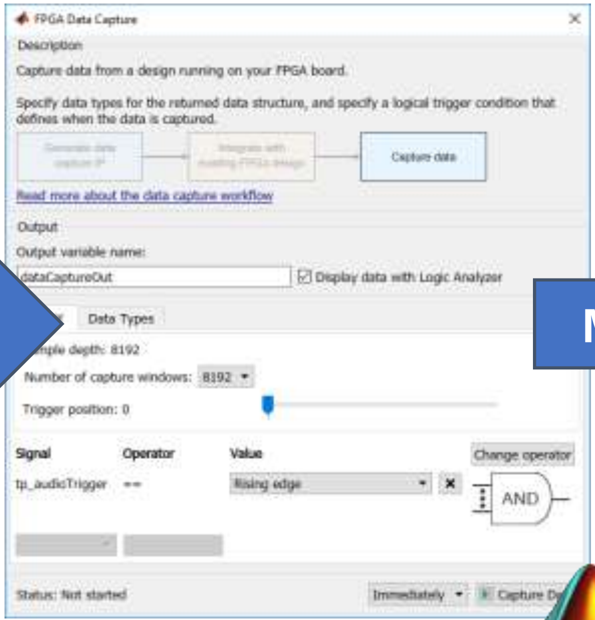
快速调试：FPGA Data Capture

- 便于访问内部FPGA信号
- 自动分析大型数据集
- 使用记录的FPGA数据通过仿真再现真实场景
- 与MathWorks HDL workflows直接集成



连接MATLAB调试

JTAG


FPGA Data Capture

Description

Capture data from a design running on your FPGA board.

Specify data types for the returned data structure, and specify a logical trigger condition that defines when the data is captured.

Read more about the data capture workflow

Output

Output variable name: dataCaptureOut

☑ Display data with Logic Analyzer

Data Types

Sample depth: 8192

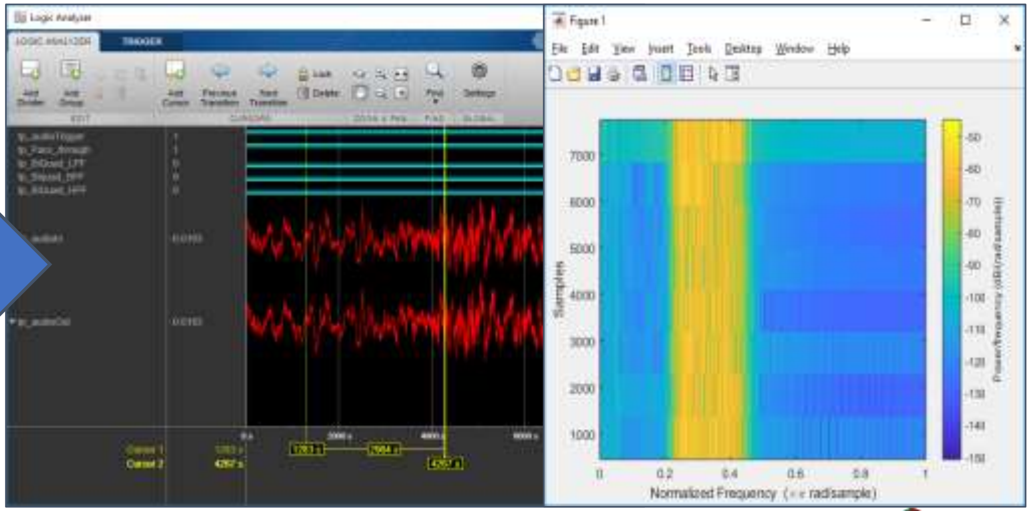
Number of capture windows: 8192

Trigger position: 0

Signal	Operator	Value
tp_audioTrigger	==	Rising edge

Status: Not started

MATLAB



Logic Analyzer
Figure 1
File Edit View Insert Tools Desktop Window Help
Samples
Normalized Frequency (×π rad/sample)
Power/frequency (dB/rad/sample)

SystemVerilog Direct Programming Interface (DPI)

SystemVerilog code

```
module testbench;
    int in1, in2, result;
    chandle dpic_h;

    import "DPI-C" function void
      DPI_checker_model(
        input chandle objhandle,
        input int in1,
        input int in2,
        output int result
      );

    myDesign DUT1 (in1, in2, result);

    initial
    begin
        in1=6; in2=7;
        DPI_checker_model (dpic_h, in1, in2,result);
        $display("in1=%d in2=%d result=%d", in1, in2, result);
    end

endmodule
```

Declare a C function using "import" keyword

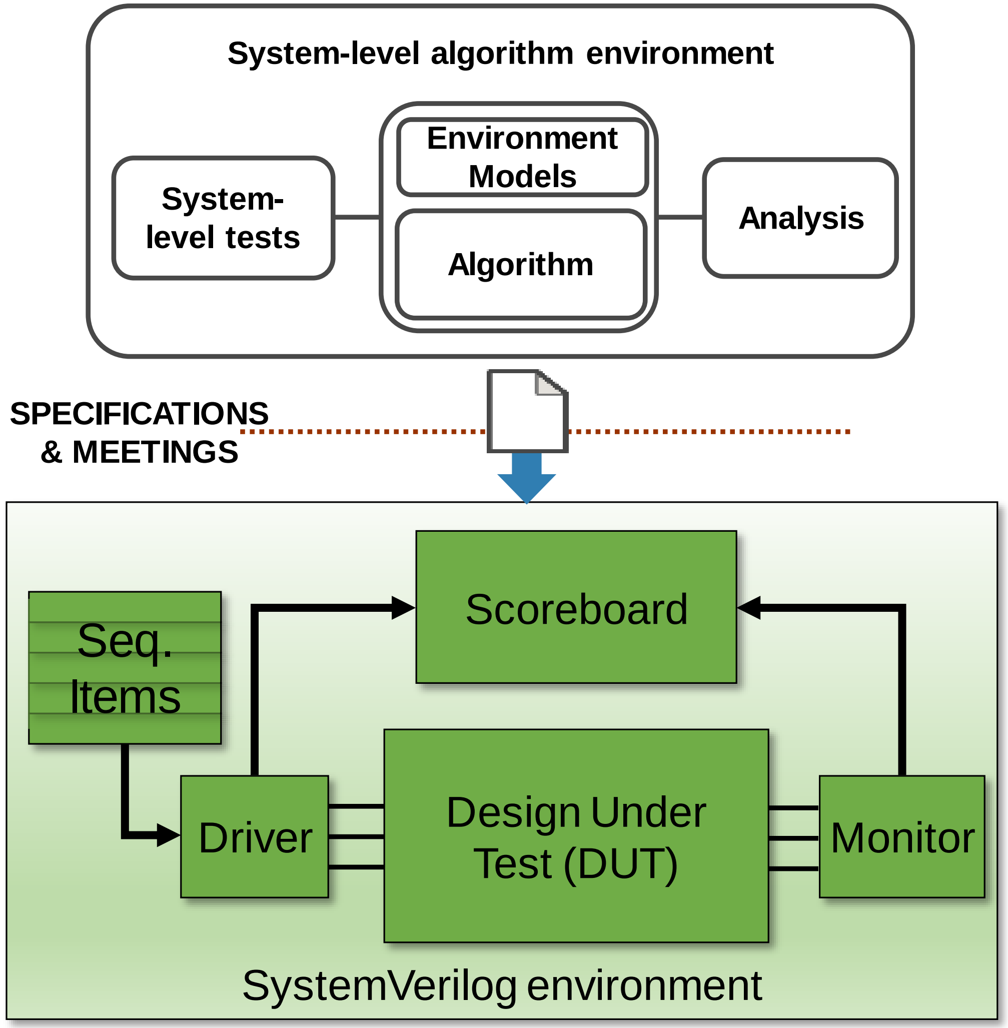
Call it just like a SystemVerilog function

Enables easy integration of C and SystemVerilog

验证环境的创建

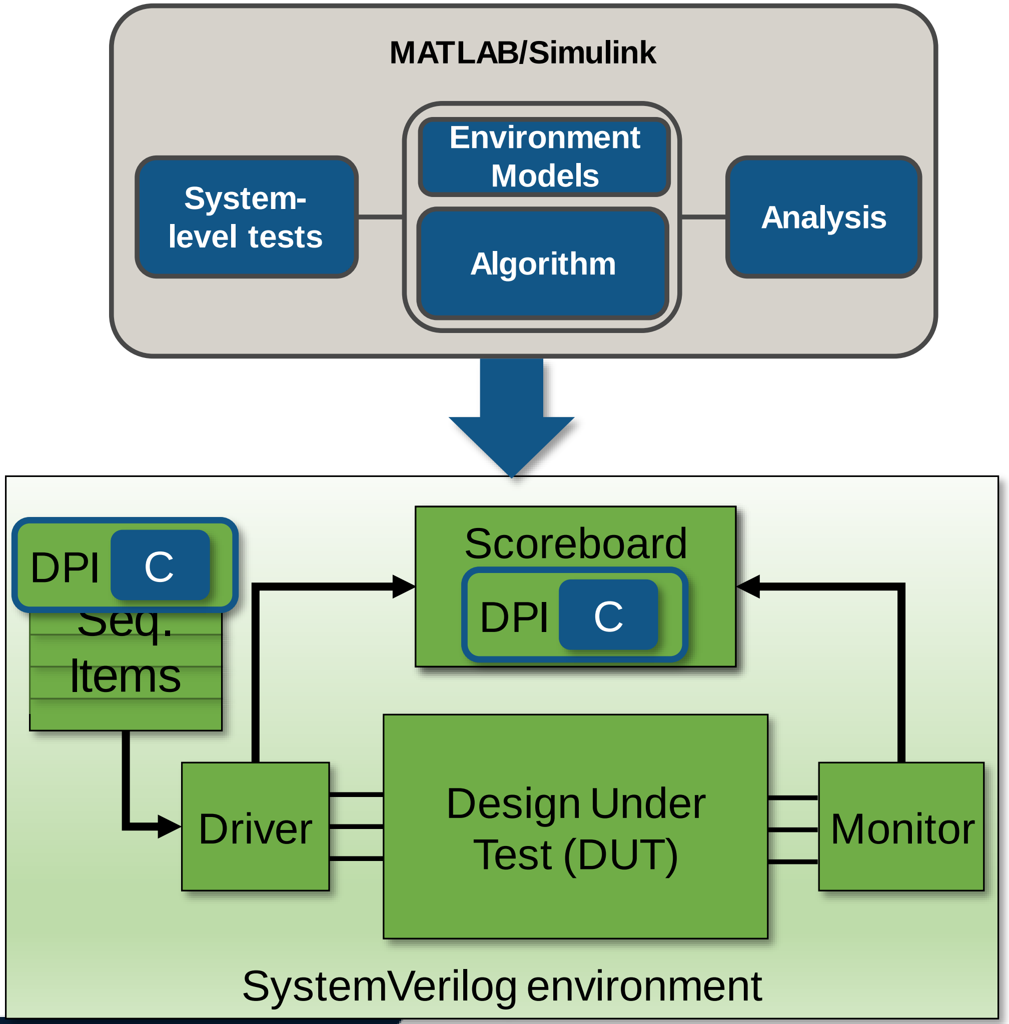
- 在MATLAB或Simulink中验证的系统和算法
  - ✓ 算法和模型
  - ✓ 环境模型
  - ✓ 系统级的激励
- 编写说明文档，交给验证人员
- 验证人员阅读并理解文档，然后重新创建：
  - ✓ checker模型
  - ✓ DUT以外的模型
  - ✓ 激励

能否复用?

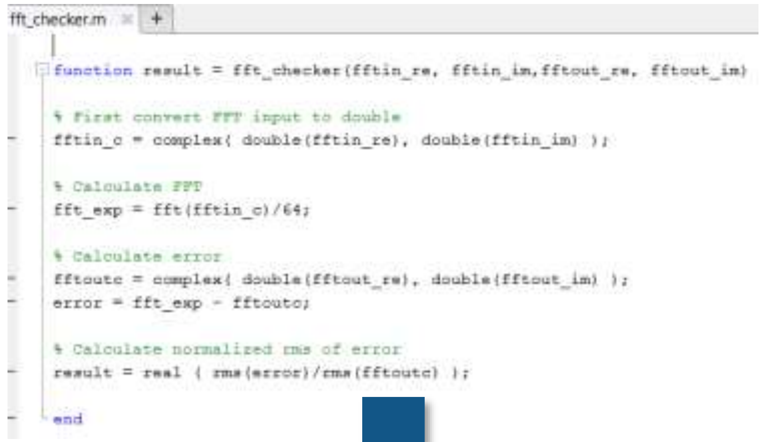


HDL Verifier 可生成 SystemVerilog DPI 组件

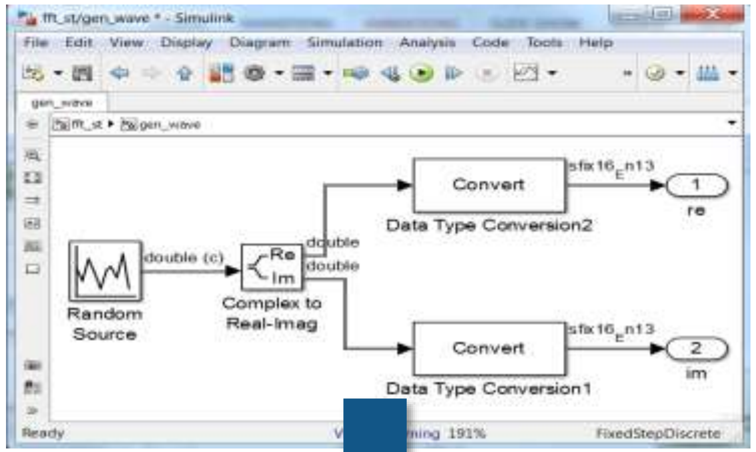
- 在验证中复用MATLAB/Simulink模型
  - ✓ 节省模型/测试用例开发时间
  - ✓ 需求变更时易于更新
  - ✓ 支持主流SystemVerilog仿真器
- 可自动化
  - ✓ 生成代码和SystemVerilog接口
  - ✓ 生成并自动运行makefile以生成共享库
- 支持任何可以生成C代码的模块
  - ✓ 数字设计和模拟设计
  - ✓ 支持的模块多



从MATLAB函数或Simulink模型生成DPI组件


fft_checker.m

```
function result = fft_checker(fftin_re, fftin_im,fftout_re, fftout_im)

% First convert FFT input to double
fftin_c = complex( double(fftin_re), double(fftin_im) );

% Calculate FFT
fft_exp = fft(fftin_c)/64;

% Calculate error
fftoutc = complex( double(fftout_re), double(fftout_im) );
error = fft_exp - fftoutc;

% Calculate normalized rms of error
result = real ( rms(error)/rms(fftoutc) );

end
```



fft_checker Component

- C Source and Header files
- DPI-C wrapper Source and Heade
- gcc and VC makefiles
- SystemVerilog Module

Component Testbench

- SystemVerilog Testbench
- Reference IO Data
- DO files to Run testbench

gen_wave component

- C Source and Header files
- DPI-C wrapper Source and Header
- gcc and VC makefiles
- SystemVerilog Module

Component Testbench

- SystemVerilog Testbench
- Reference IO Data
- DO files to Run testbench

集成到 SystemVerilog UVM 环境

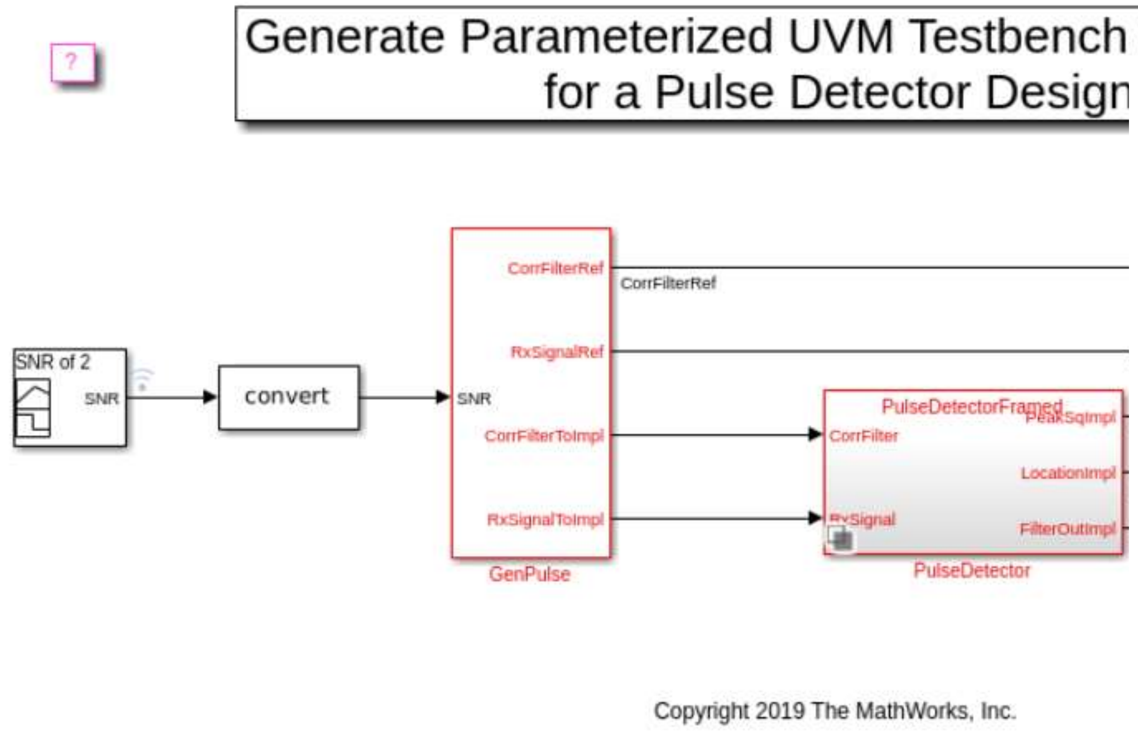
```
import "DPI-C" function chandle DPI_gen_wave0_initialize(
      chandle existhandle);

import "DPI-C" function void DPI_gen_wave0_output(
      input chandle objhandle,
      output shortint re [64],
      output shortint im [64]);
```

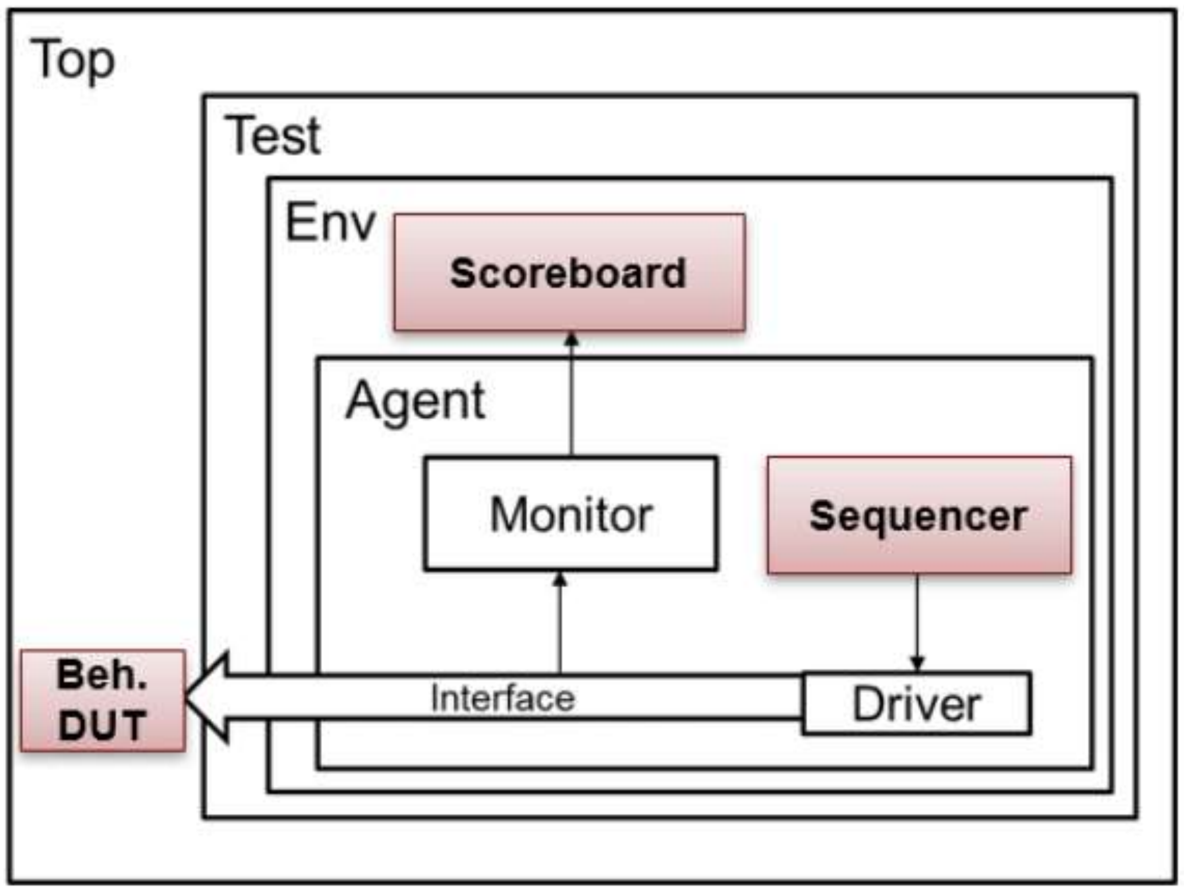
Use in test sequence

```
import "DPI-C" function chandle DPI_fft_checker_initialize(
      chandle existhandle);

import "DPI-C" function void DPI_fft_checker(
      input chandle objhandle,input shortint fftin_re [64],
      input shortint fftin_im [64],
      input shortint fftout_re [64],
      input shortint fftout_im [64],
      output real result);
```

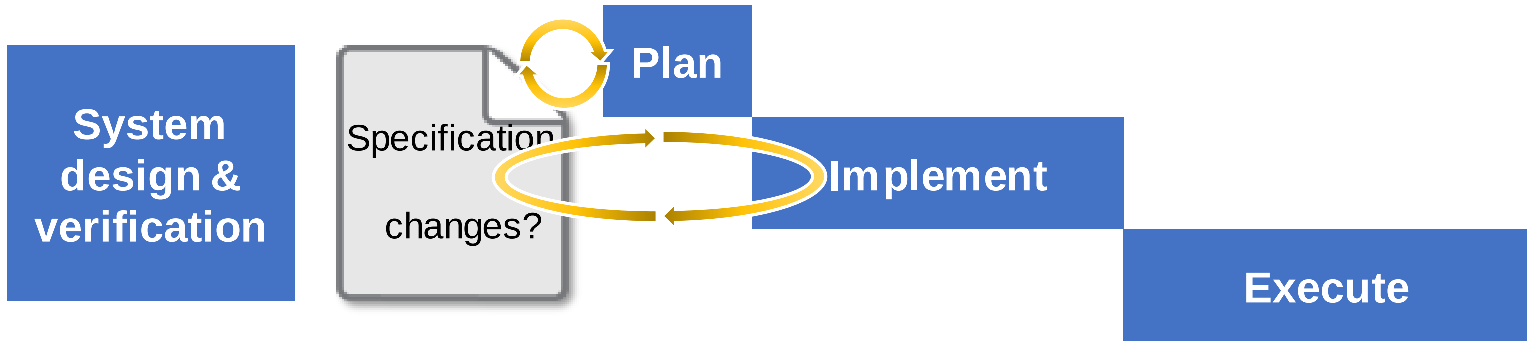
Use in scoreboard

集成到 SystemVerilog UVM 环境： uvmbuild

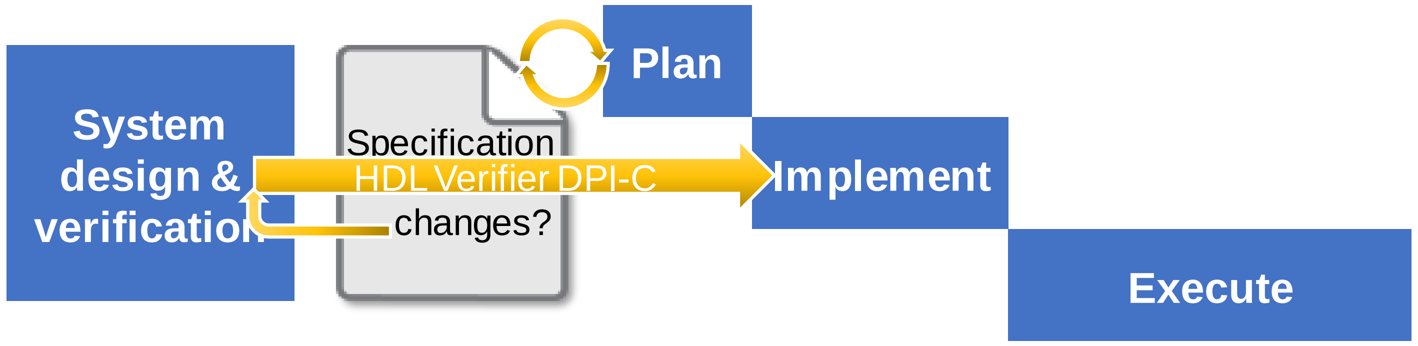




HDL Verifier带来验证过程的改变



- 手写验证模型
  - 需要准确理解spec
  - 调试工作量大
- Spec发生变更:
  - 手工更新模型
  - 难以准确反映系统设计



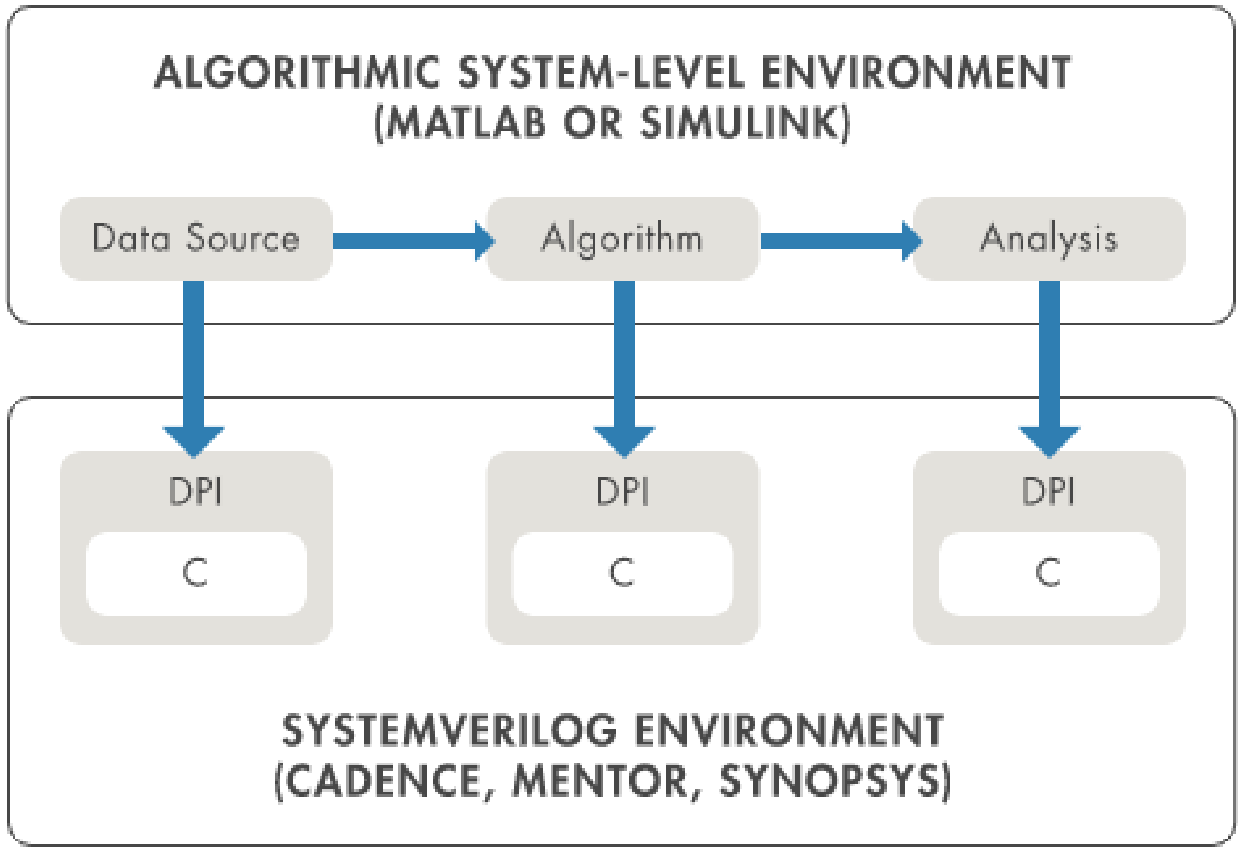
- 从原始模型自动生成验证模型
  - 原始模型就是spec
  - 无需额外工作
- Spec发生变更:
  - 对算法更新并确认
  - 重新生成即可

HDL Verifier对FPGA验证的支持

Cosimulation

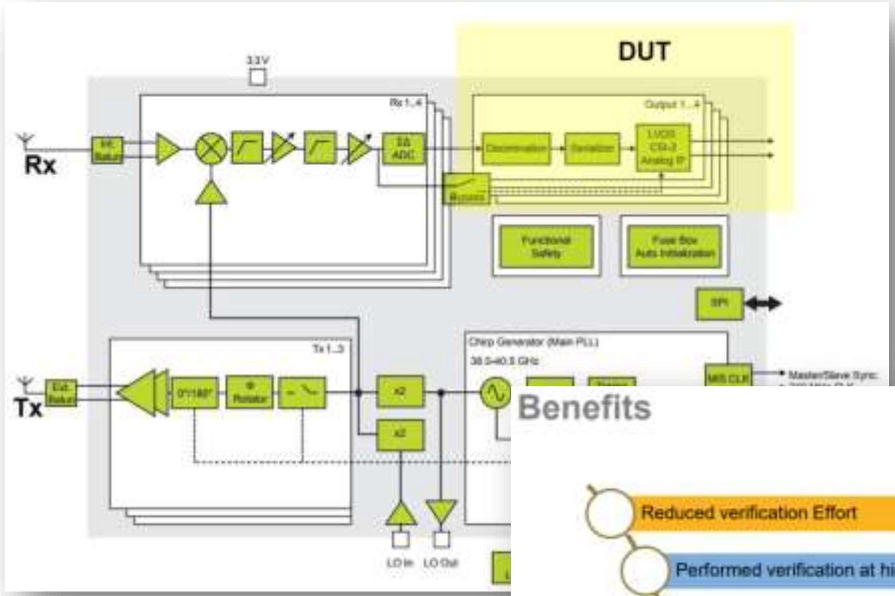
SystemVerilog DPI

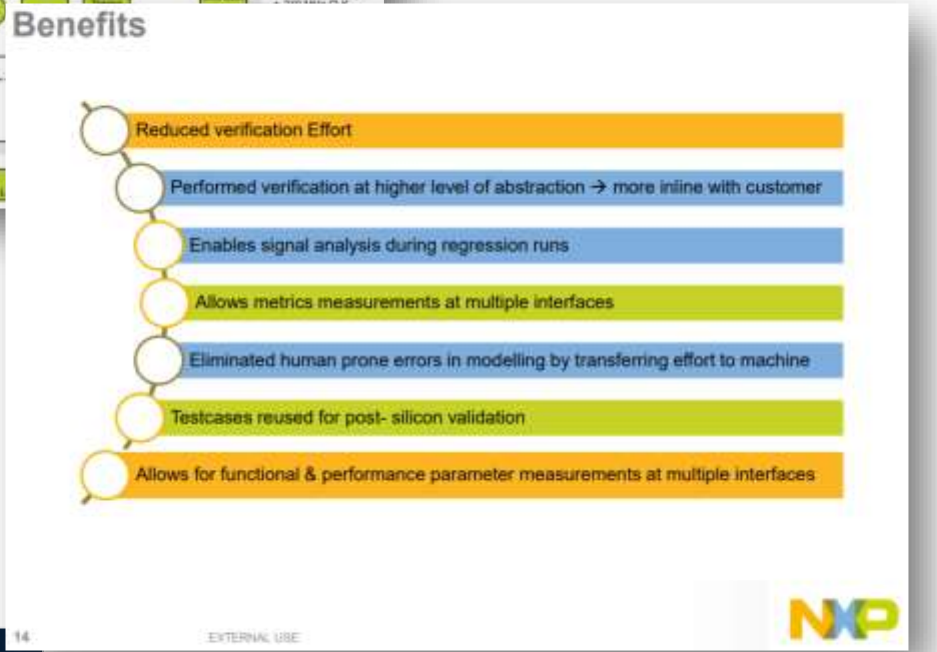
FPGA-in-the-Loop


ALGORITHMIC SYSTEM-LEVEL ENVIRONMENT
(MATLAB OR SIMULINK)
Data Source
Algorithm
Analysis
DPI
C
DPI
C
DPI
C
SYSTEMVERILOG ENVIRONMENT
(CADENCE, MENTOR, SYNOPSYS)

NXP 使用 SystemVerilog DPI 提升验证效率

- 应用：汽车雷达
- 挑战
  - 包括数字信号处理和多种运行模式的复杂混合信号设计
  - 设计变更后需要修改UVM组件
- 解决方案
  - 在MATLAB中编写SNR和PSD函数，生成用于UVM的SystemVerilog DPI组件




Benefits

- Reduced verification Effort
- Performed verification at higher level of abstraction → more inline with customer
- Enables signal analysis during regression runs
- Allows metrics measurements at multiple interfaces
- Eliminated human prone errors in modelling by transferring effort to machine
- Testcases reused for post- silicon validation
- Allows for functional & performance parameter measurements at multiple interfaces

14 EXTERNAL USE

MathWorks® 产品 解决方案 学术 支持 社区 活动 获取 MATLAB

HDL Code Generation and Verification

搜索 MathWorks.com

概述 快速入门

Simulink 用于 HDL 代码生成和验证

无需编写 HDL 代码即可探索、实现和验证 FPGA、SoC 或 ASIC 设计。

进行顶层设计和探索，之后直接通过 MATLAB® 或 Simulink® 生成并验证 HDL，以用于 FPGA、ASIC 或片上系统 (SoC) 原型或生产项目。

“与传统的设计流程相比，采用基于

谢 谢