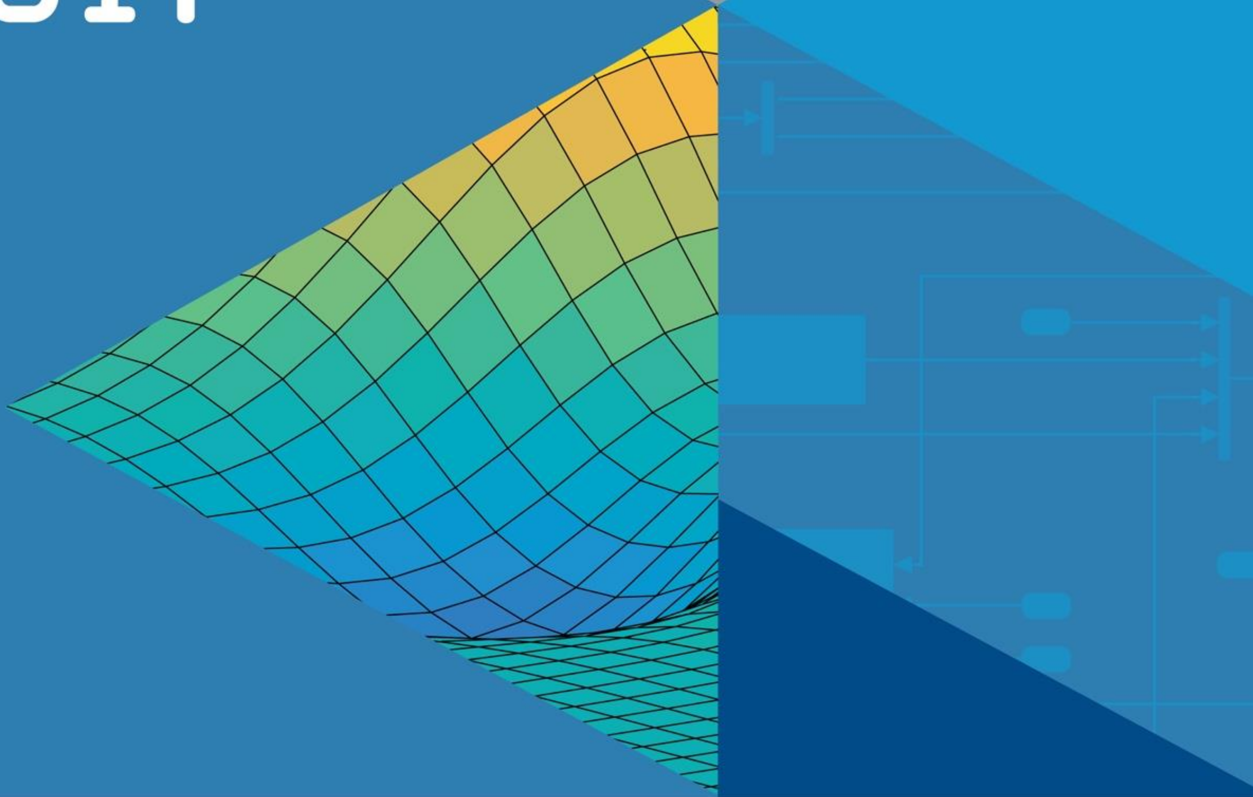
Accelerating FPGA/ASIC Design and Verification

MATLAB EXPO 2017

Tabrez Khan – Senior Application Engineer
Vidya Viswanathan – Application Engineer

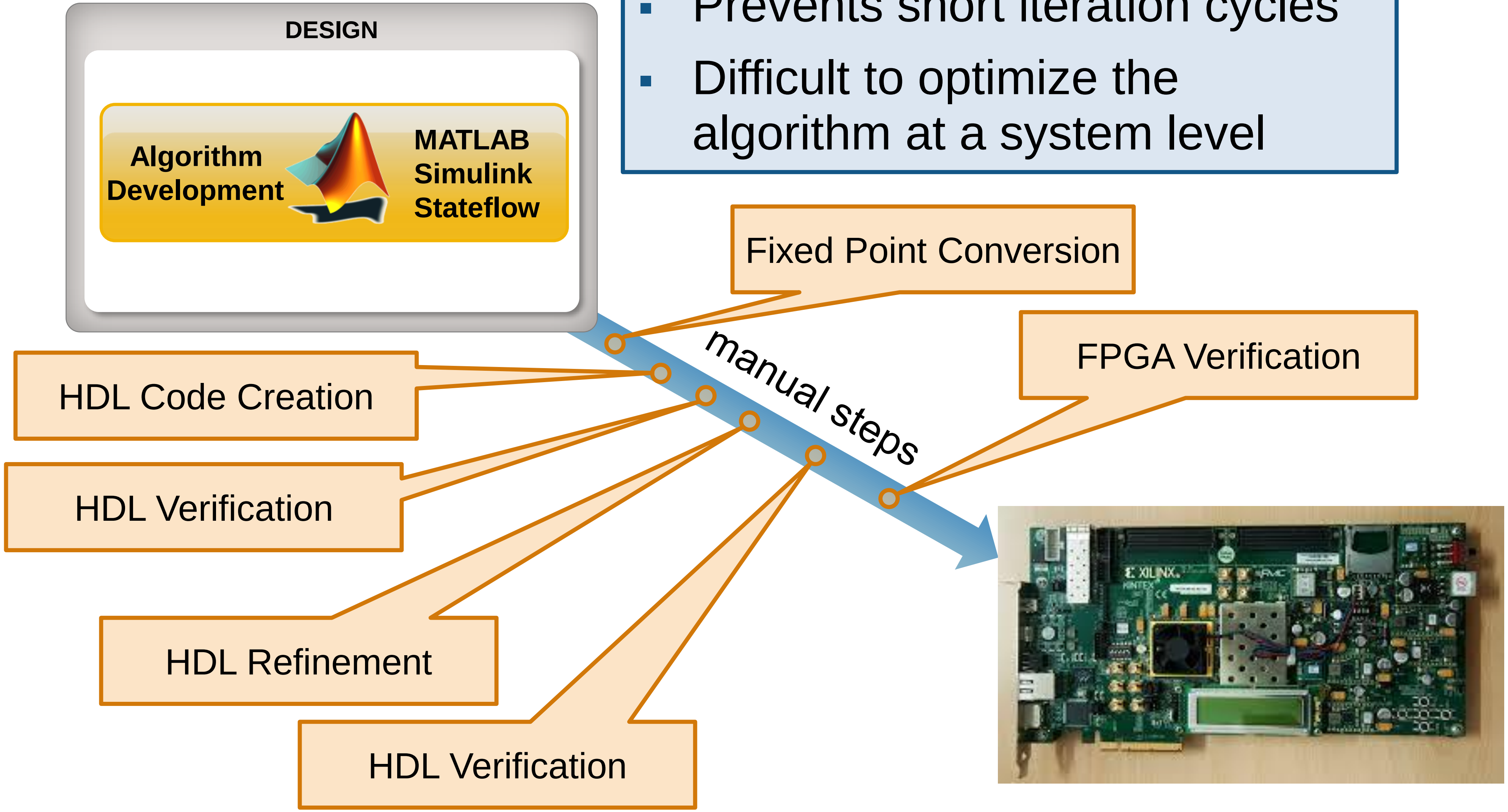


Agenda

- Challeges with Traditional Implementation workflow
- Model-Based Design for Implementation
- Generate VHDL® and Verilog® code from MATLAB, Simulink, and Stateflow®
- Optimize the generated RTL design for area and/or speed
- Develop system-level test benches in MATLAB and Simulink for RTL verification with EDA tools
- Automate verification with FPGA-in-the-Loop
- Summary & next steps

Traditional Implementation Workflow

- Long development cycles
- Prevents short iteration cycles
- Difficult to optimize the algorithm at a system level



DESIGN

Algorithm Development

MATLAB
Simulink
Stateflow

Fixed Point Conversion

HDL Code Creation

HDL Verification

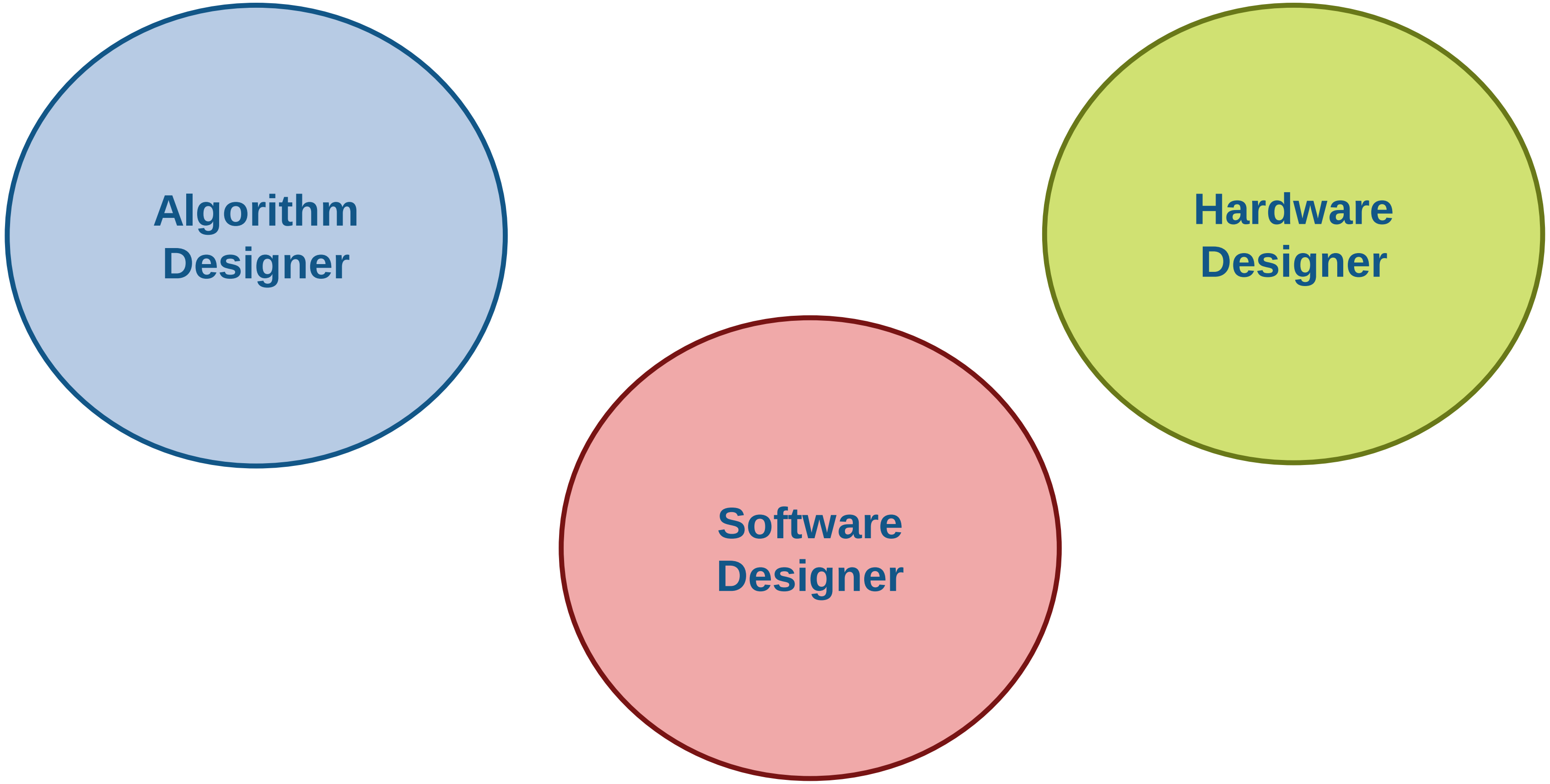
HDL Refinement

HDL Verification

FPGA Verification

manual steps

Separate Views of DSP Implementation

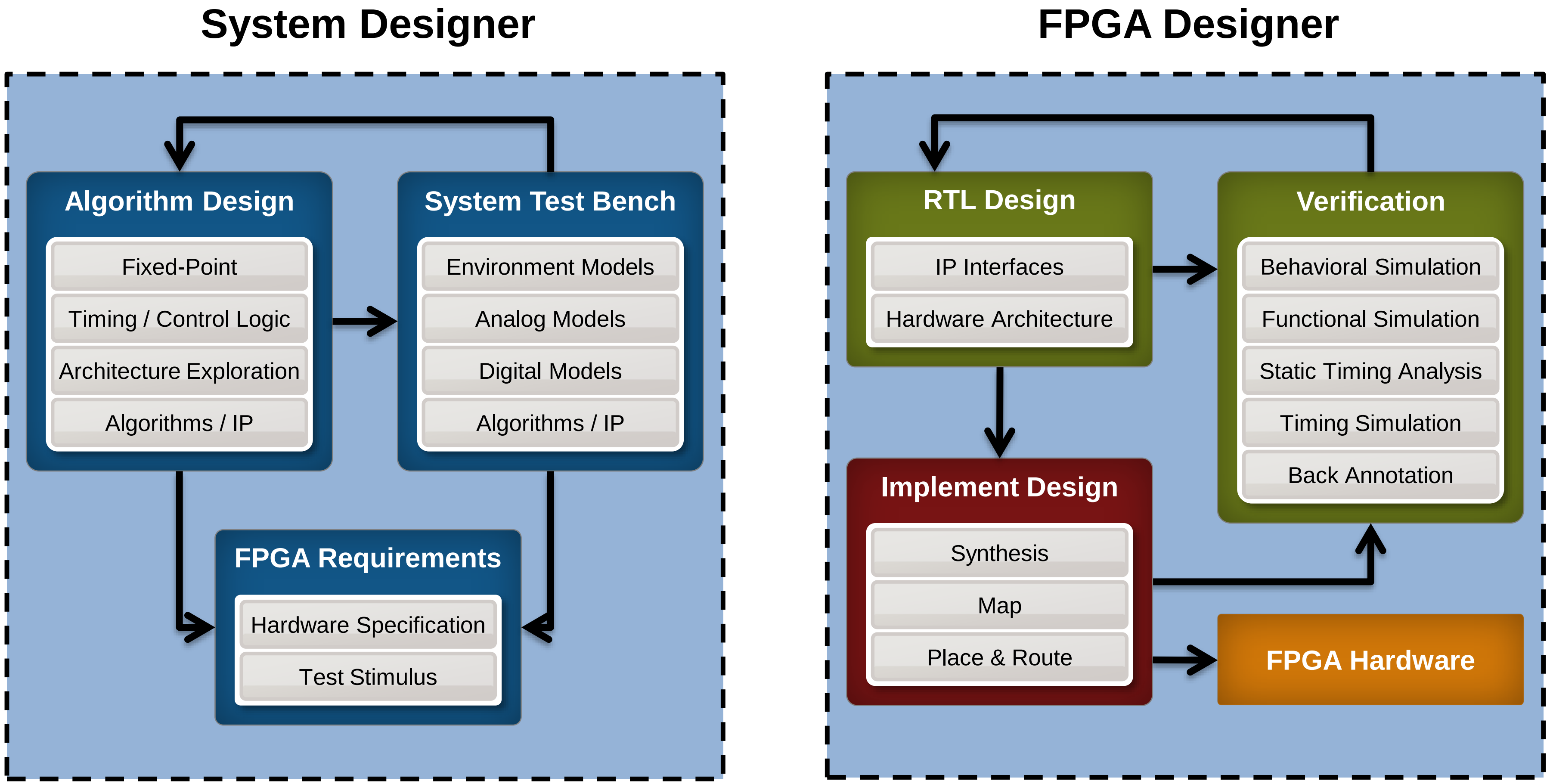


Algorithm Designer

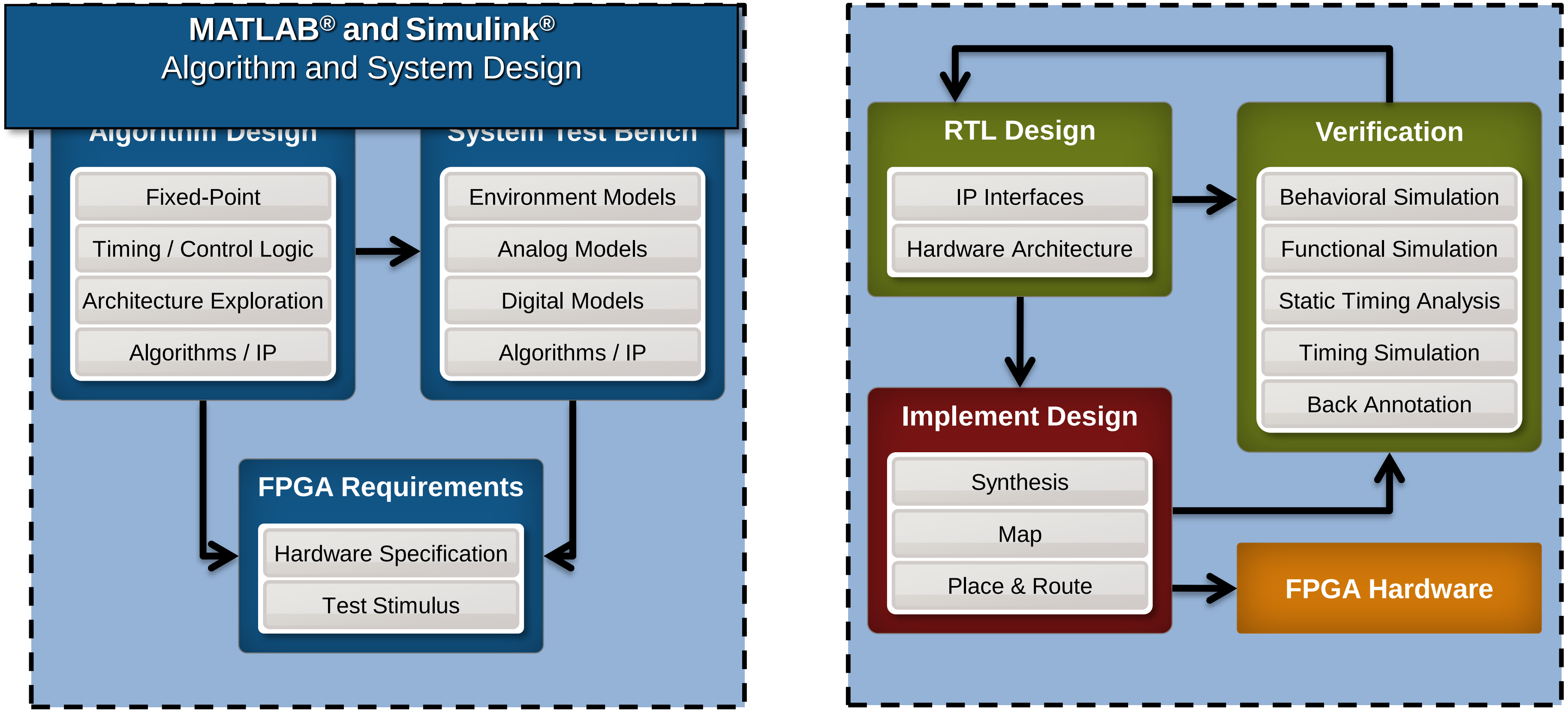
Hardware Designer

Software Designer

Separate Views of DSP Implementation



Model-Based Design for Implementation



MATLAB® and Simulink®
Algorithm and System Design

Algorithm Design
- Fixed-Point
- Timing / Control Logic
- Architecture Exploration
- Algorithms / IP

System Test Bench
- Environment Models
- Analog Models
- Digital Models
- Algorithms / IP

FPGA Requirements
- Hardware Specification
- Test Stimulus

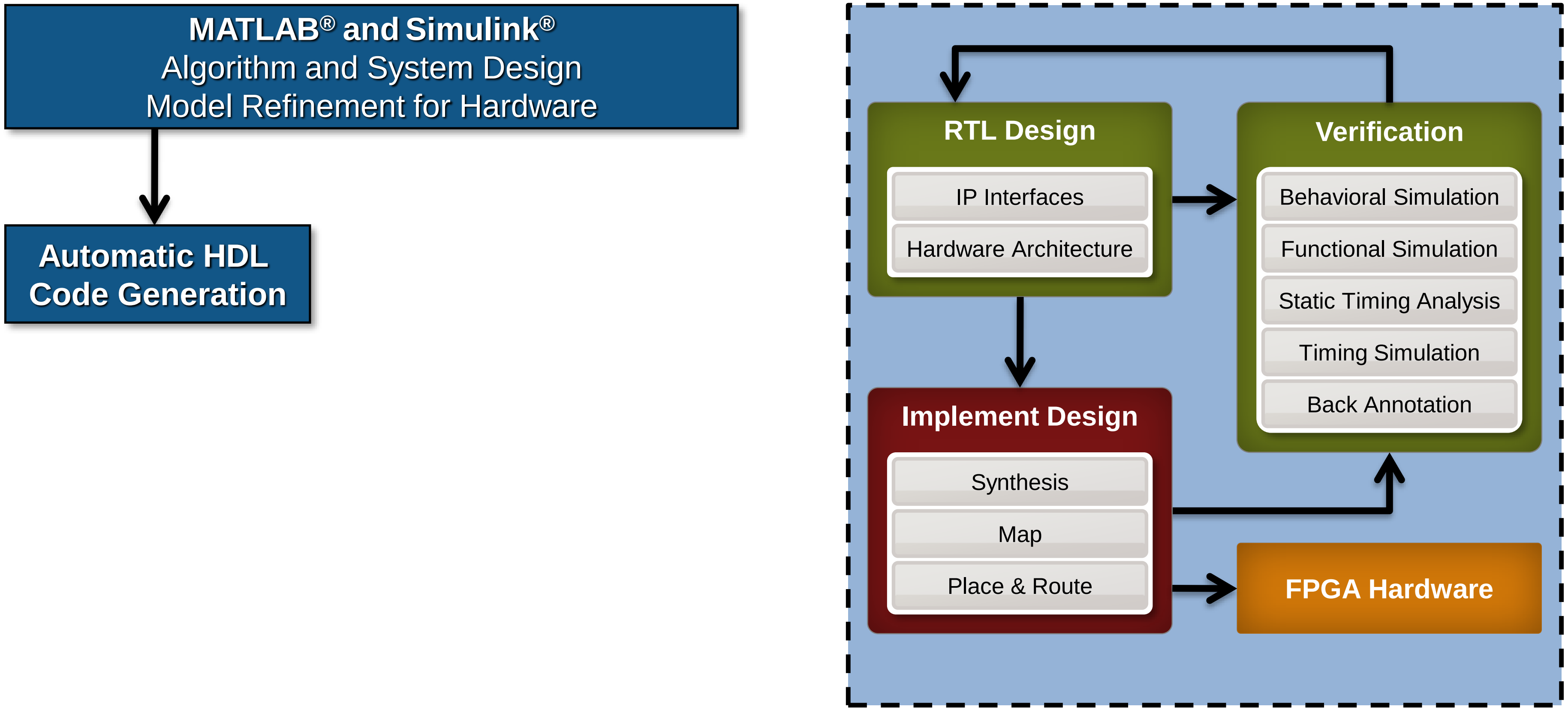
RTL Design
- IP Interfaces
- Hardware Architecture

Verification
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

FPGA Hardware

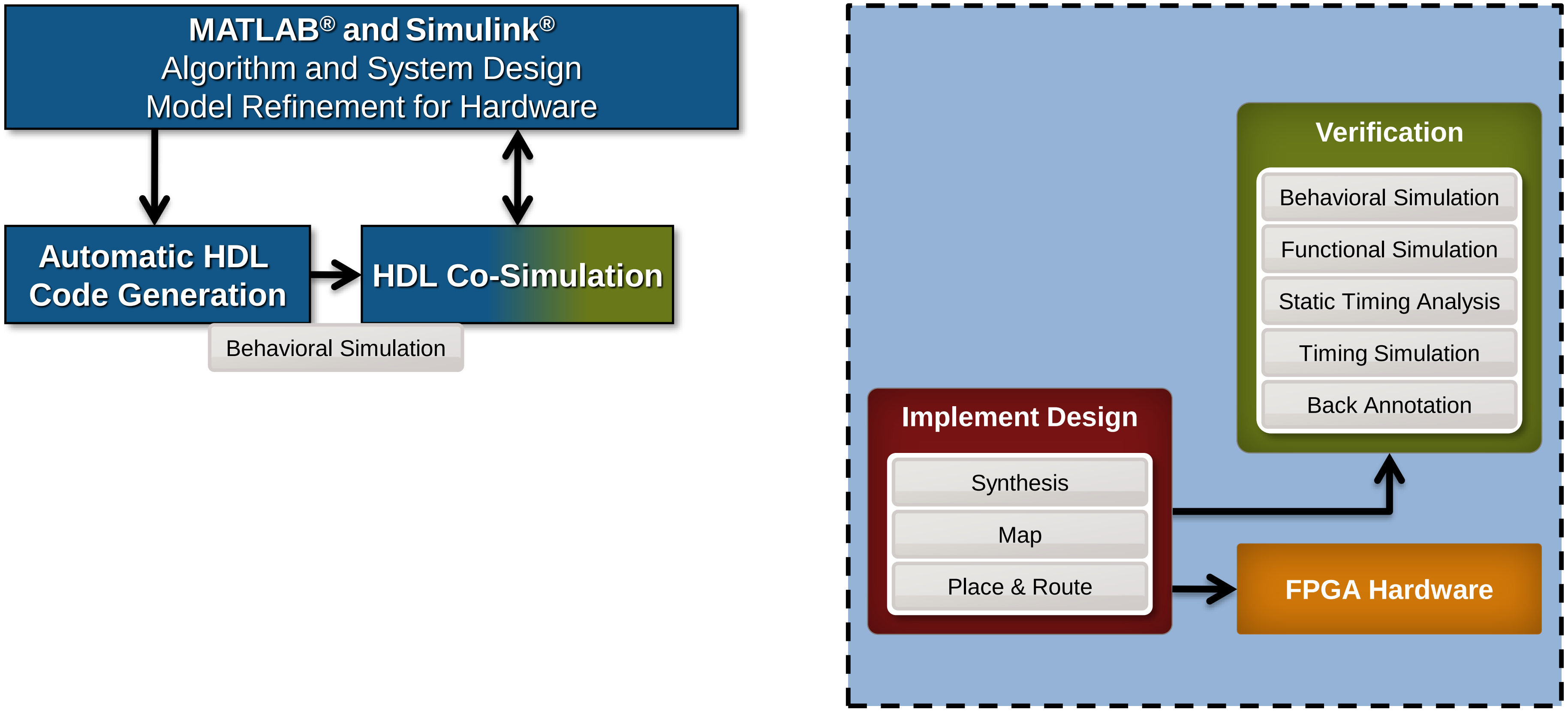
Model-Based Design for Implementation



MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

Model-Based Design for Implementation



MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

HDL Co-Simulation

Behavioral Simulation

Implement Design

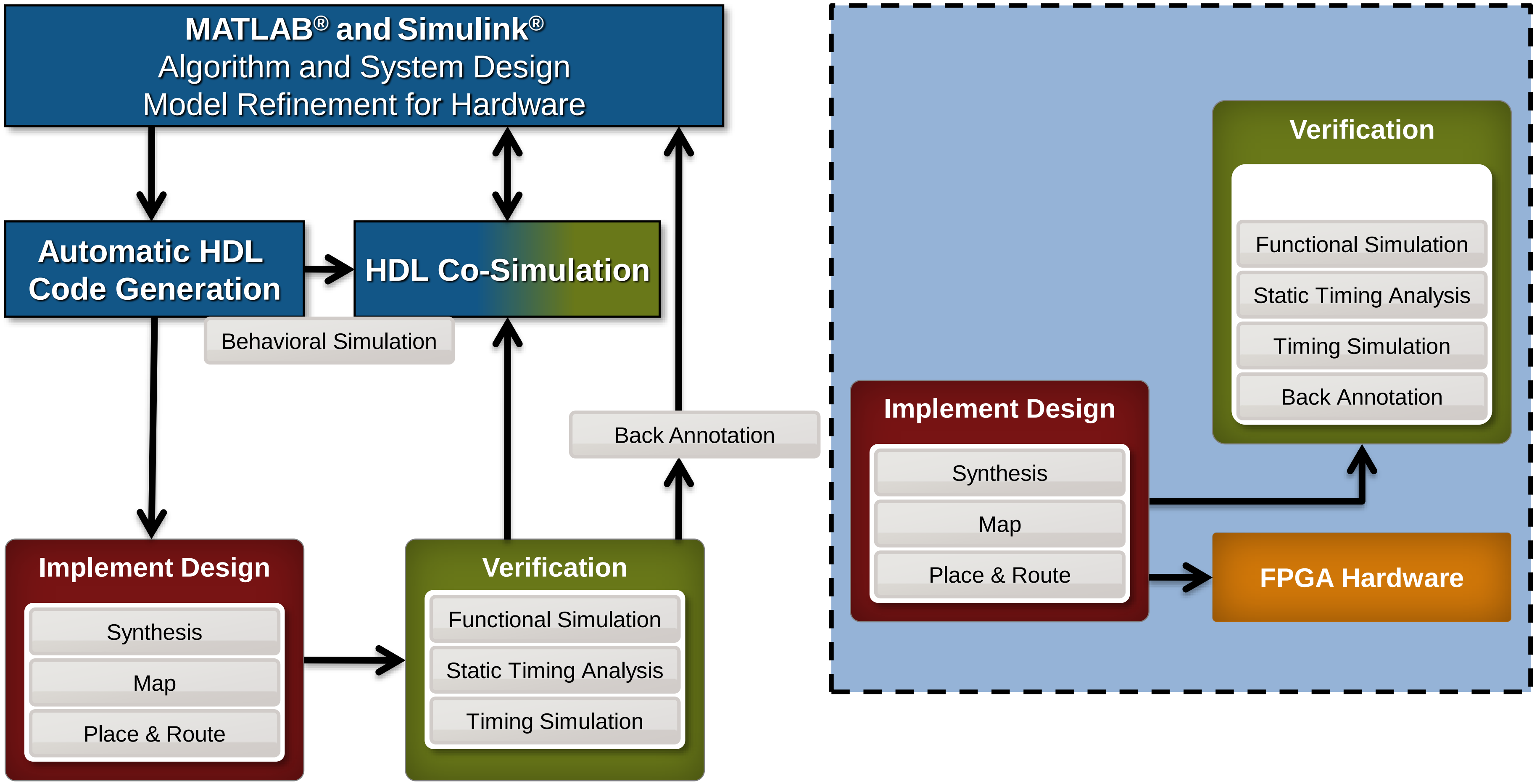
- Synthesis
- Map
- Place & Route

Verification

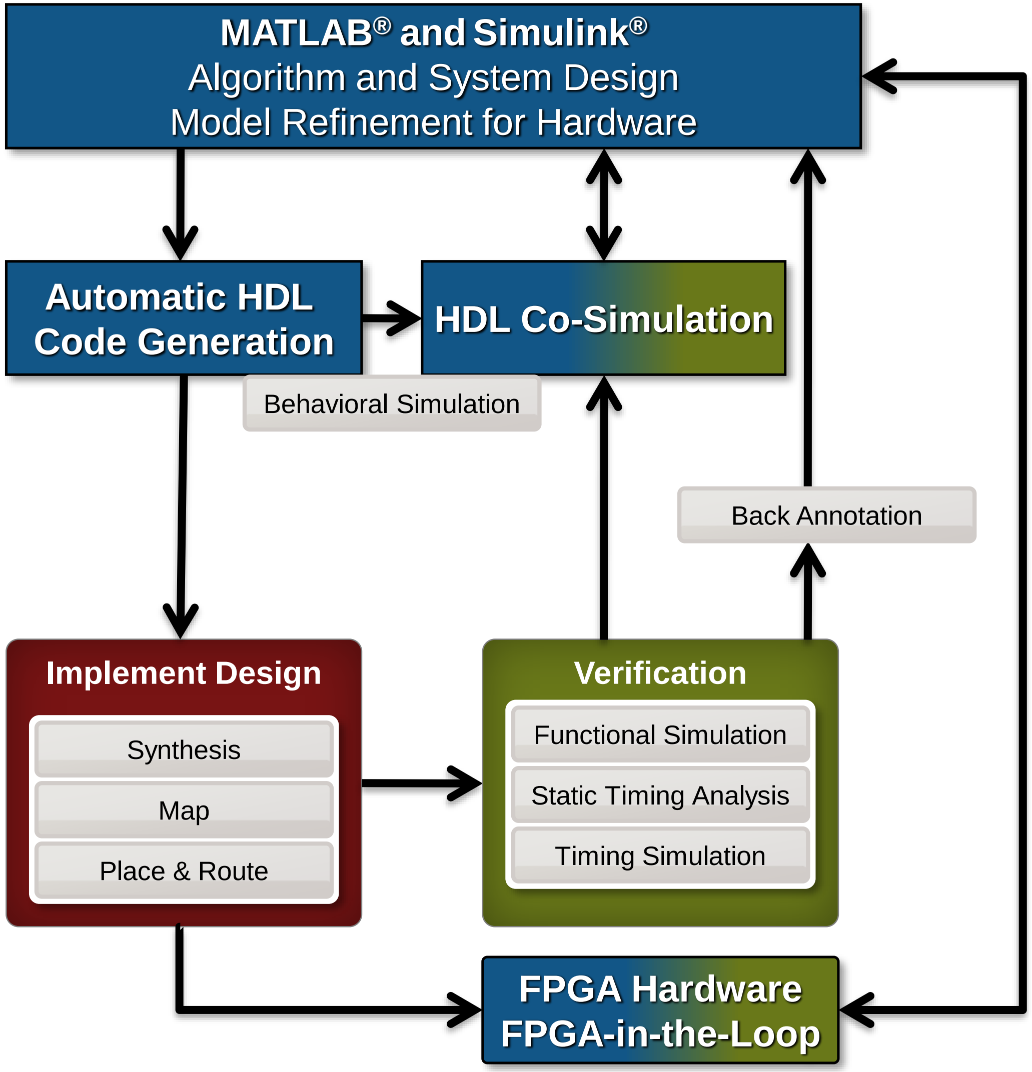
- Behavioral Simulation
- Functional Simulation
- Static Timing Analysis
- Timing Simulation
- Back Annotation

FPGA Hardware

Model-Based Design for Implementation



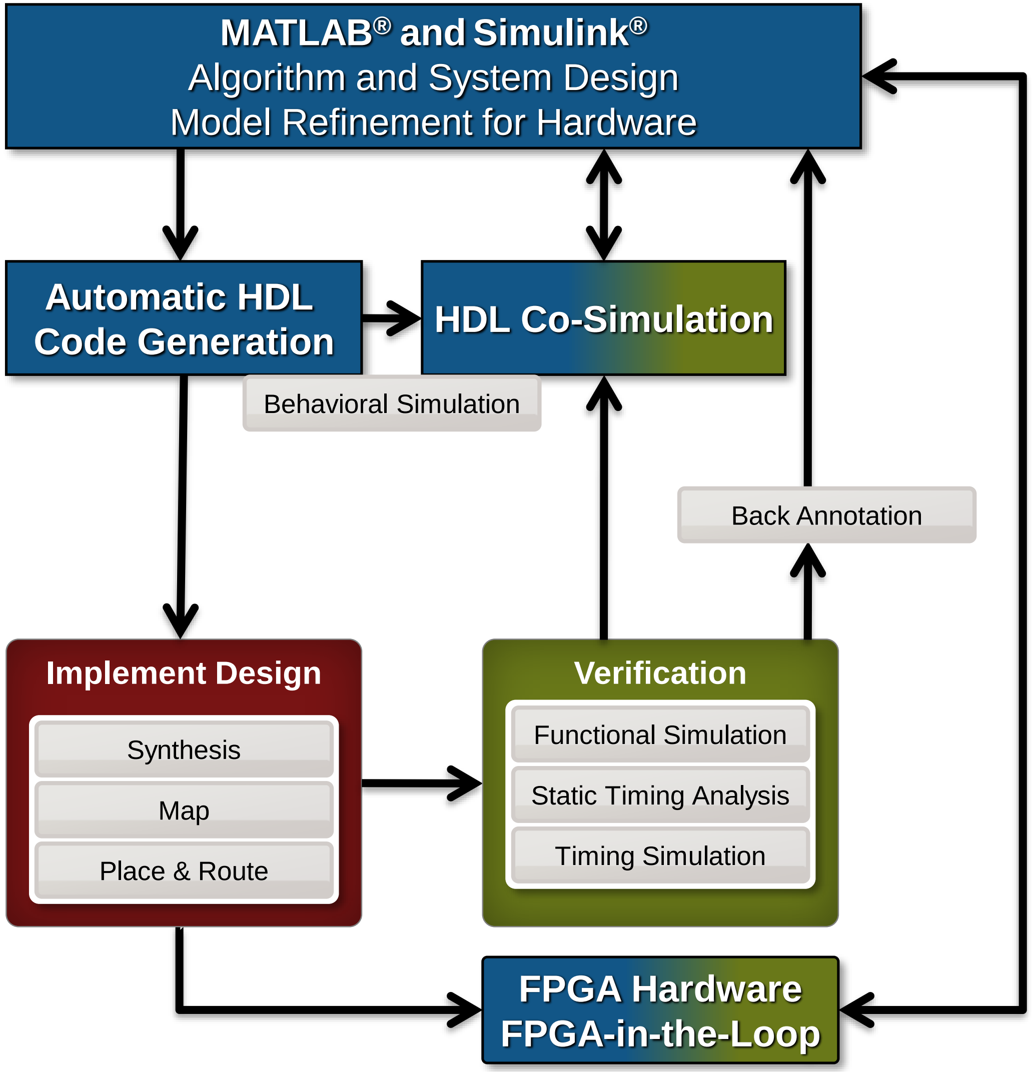
Model-Based Design for Implementation



FPGA Hardware

Model-Based Design for Implementation

Integrated Workflow

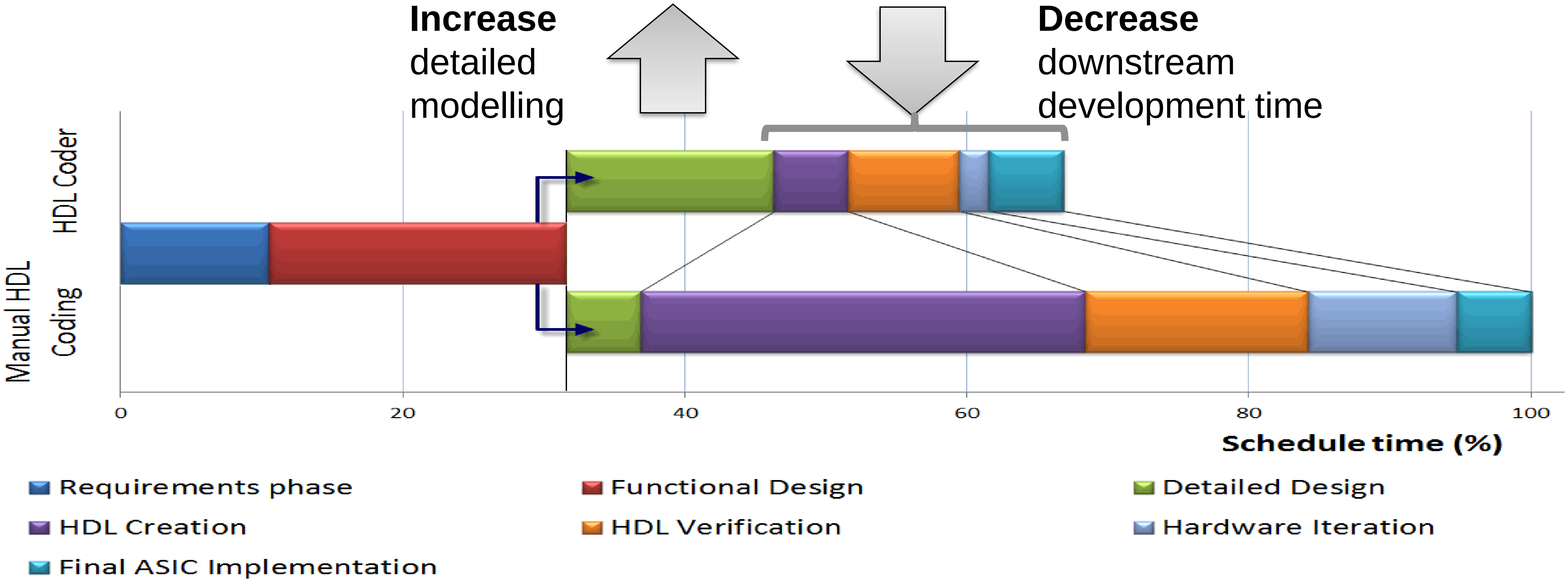


All steps from 1 single GUI

Why Model-Based Design: Achieving the Shift-Left

Reduce overall development time

- Reduced FPGA prototype development schedule
- Shorter design iteration cycle by 80%
- Improved product quality

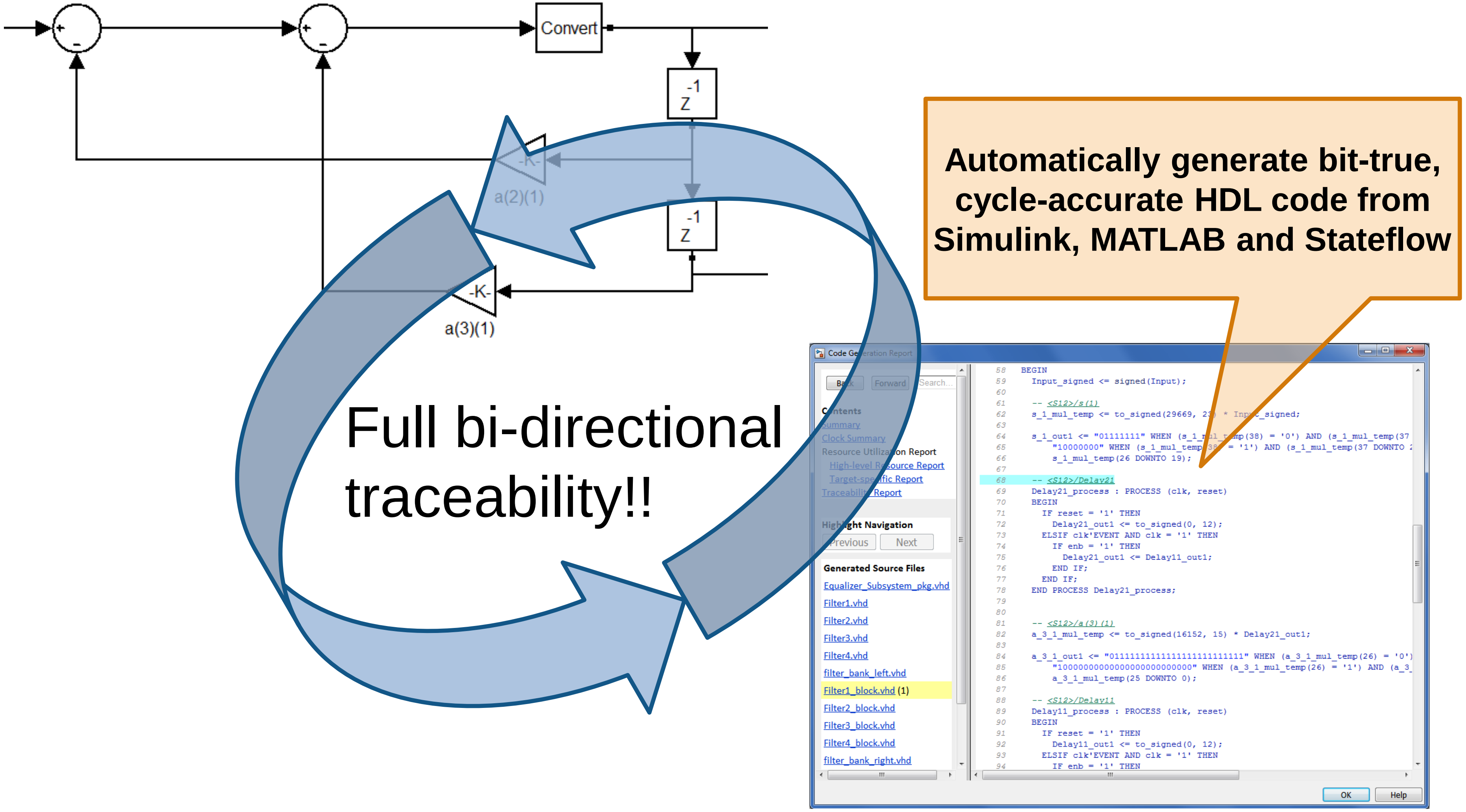


Increase detailed modelling

Decrease downstream development time

Automatic HDL Code Generation

HDL Coder

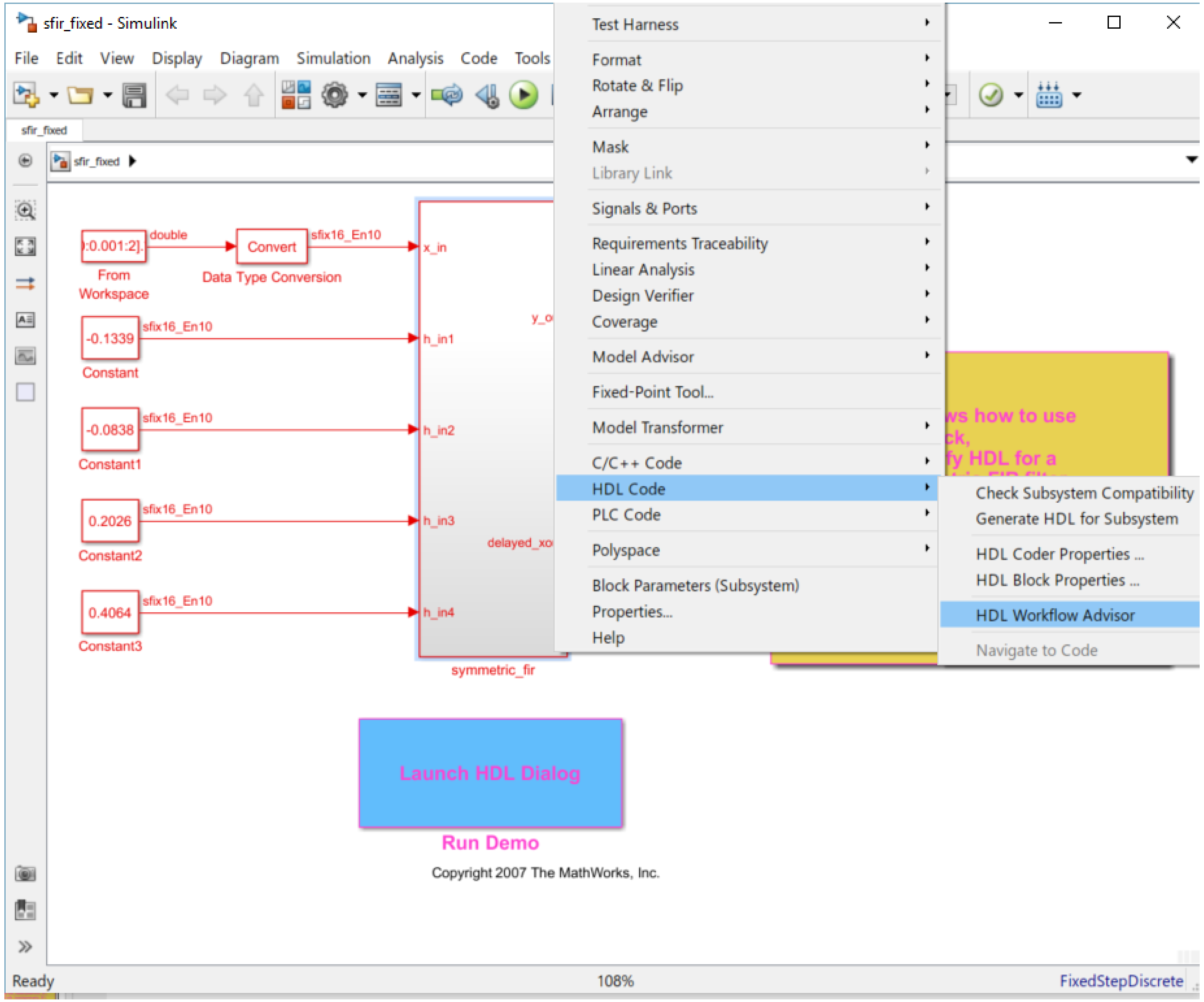


Full bi-directional traceability!!

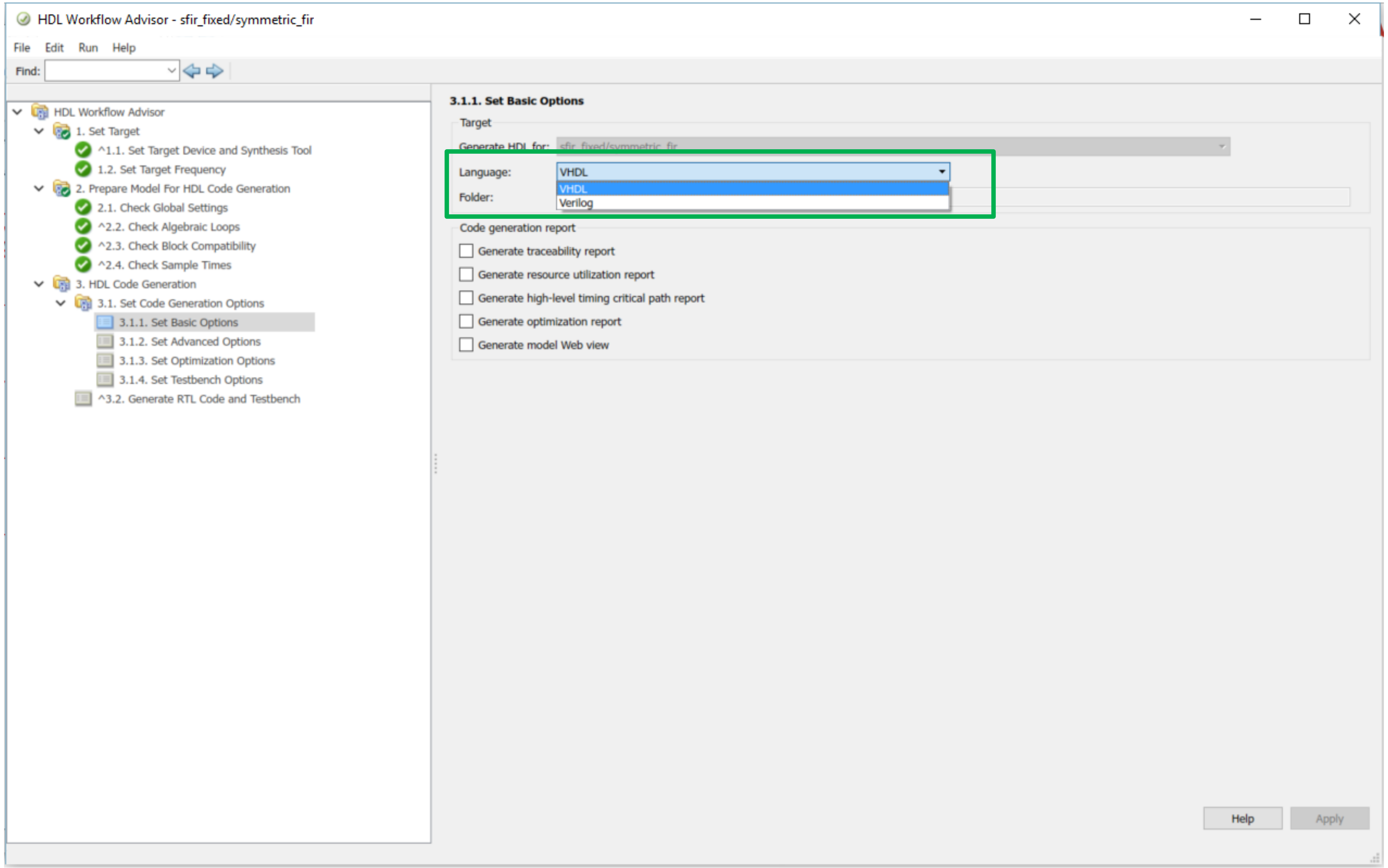
```
BEGIN
  Input_signed <= signed(Input);

  -- <S12>/s(1)
  s_1_mul_temp <= to_signed(29669, 23) * Input_signed;

  s_1_out1 <= "01111111" WHEN (s_1_mul_temp(38) = '0') AND (s_1_mul_temp(37
      "10000000" WHEN (s_1_mul_temp(38) = '1') AND (s_1_mul_temp(37 DOWNTO 2
      s_1_mul_temp(26 DOWNTO 19);

  -- <S12>/Delay21
  Delay21_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Delay21_out1 <= to_signed(0, 12);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        Delay21_out1 <= Delay11_out1;
      END IF;
    END IF;
  END PROCESS Delay21_process;


  -- <S12>/a(3)(1)
  a_3_1_mul_temp <= to_signed(16152, 15) * Delay21_out1;

  a_3_1_out1 <= "01111111111111111111111111" WHEN (a_3_1_mul_temp(26) = '0')
      "10000000000000000000000000" WHEN (a_3_1_mul_temp(26) = '1') AND (a_3_
      a_3_1_mul_temp(25 DOWNTO 0);

  -- <S12>/Delay11
  Delay11_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      Delay11_out1 <= to_signed(0, 12);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
```

Automatically generate bit-true, cycle-accurate HDL code from Simulink, MATLAB and Stateflow

HDL Code Generation Example



Generate Verilog or VHDL code



HDL Workflow Advisor - sfir_fixed/symmetric_fir

- HDL Workflow Advisor
  - 1. Set Target
    - ^1.1. Set Target Device and Synthesis Tool
    - 1.2. Set Target Frequency
  - 2. Prepare Model For HDL Code Generation
    - 2.1. Check Global Settings
    - ^2.2. Check Algebraic Loops
    - ^2.3. Check Block Compatibility
    - ^2.4. Check Sample Times
  - 3. HDL Code Generation
    - 3.1. Set Code Generation Options
      - 3.1.1. Set Basic Options
      - 3.1.2. Set Advanced Options
      - 3.1.3. Set Optimization Options
      - 3.1.4. Set Testbench Options
    - ^3.2. Generate RTL Code and Testbench

3.1.1. Set Basic Options

Target

Generate HDL for: sfir_fixed/symmetric_fir

Language: VHDL

VHDL
Verilog

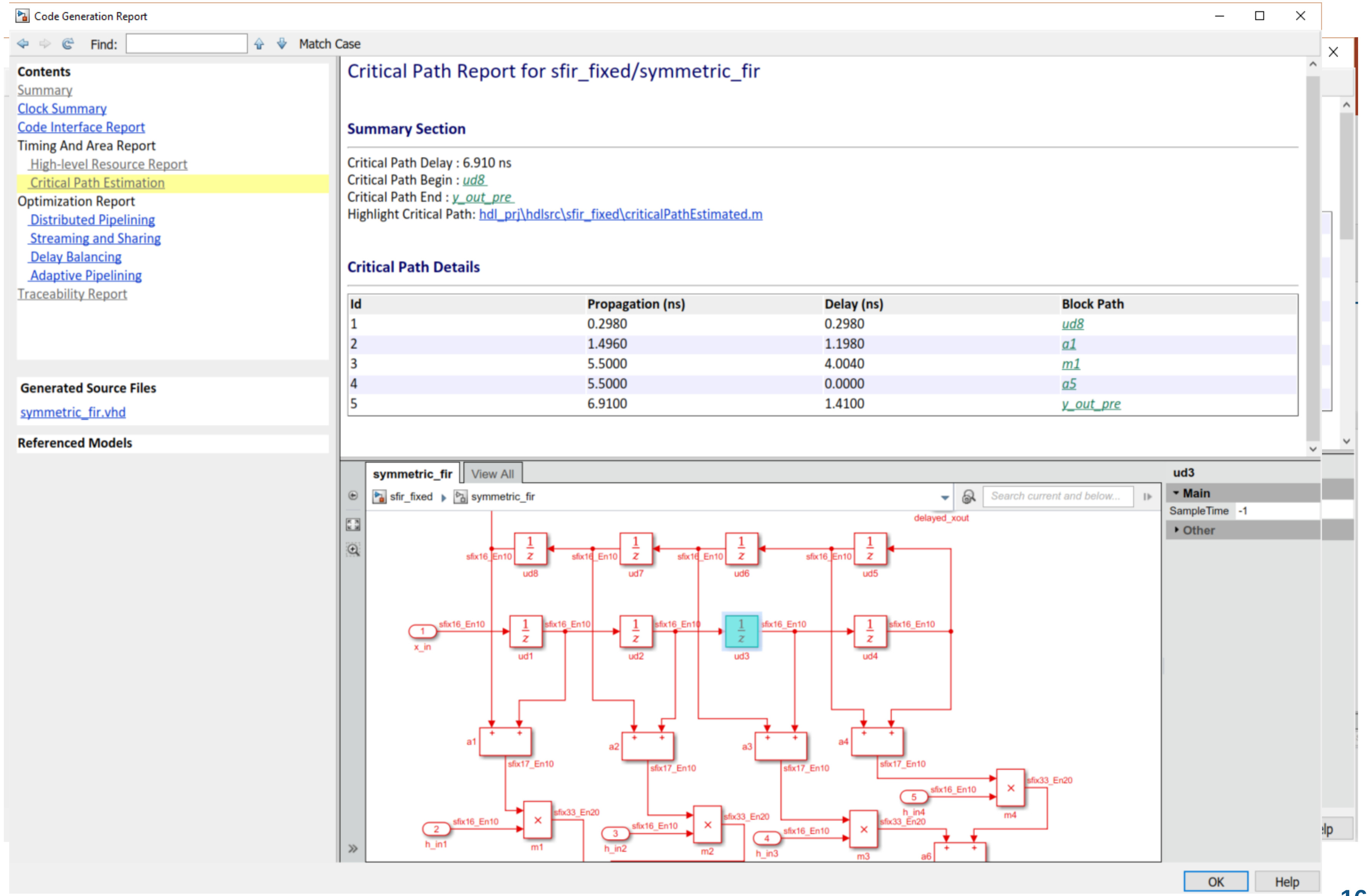
Folder:

Code generation report

- ☐ Generate traceability report
- ☐ Generate resource utilization report
- ☐ Generate high-level timing critical path report
- ☐ Generate optimization report
- ☐ Generate model Web view

Code Generation Report

- Traceability Report
- Resource Utilization Report
- Critical Path Estimation Report



Code Generation Report

Find: Match Case

Contents
- Summary
- Clock Summary
- Code Interface Report
- Timing And Area Report
  - High-level Resource Report
  - Critical Path Estimation
- Optimization Report
  - Distributed Pipelining
  - Streaming and Sharing
  - Delay Balancing
  - Adaptive Pipelining
- Traceability Report

Generated Source Files

symmetric_fir.vhd

Referenced Models

Critical Path Report for sfir_fixed/symmetric_fir

Summary Section

Critical Path Delay : 6.910 ns
Critical Path Begin : ud8
Critical Path End : y_out_pre
Highlight Critical Path: hdl_prj\hdlsrc\sfir_fixed\criticalPathEstimated.m

Critical Path Details

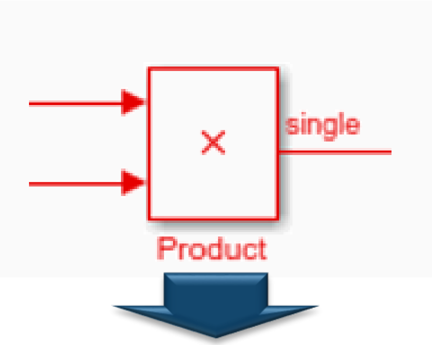
Id	Propagation (ns)	Delay (ns)	Block Path
1	0.2980	0.2980	ud8
2	1.4960	1.1980	a1
3	5.5000	4.0040	m1
4	5.5000	0.0000	a5
5	6.9100	1.4100	y_out_pre

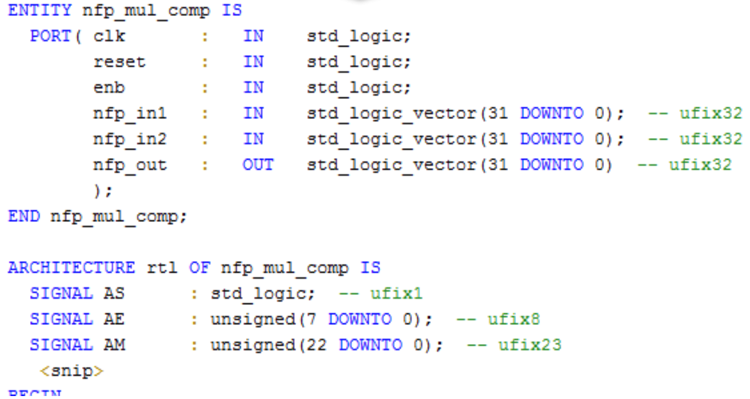
What’s new?

Native Floating-Point

Generate target-independent synthesizable RTL from single-precision floating-point models

- Good for:
  - Designs with high dynamic range calculations
  - Getting started prototyping FPGAs without having to perform fixed-point conversion
- Mix integer, fixed-point, and floating point operations to balance numerical accuracy versus hardware resource usage
- Over 130 Simulink blocks supported
- [Demo video]()

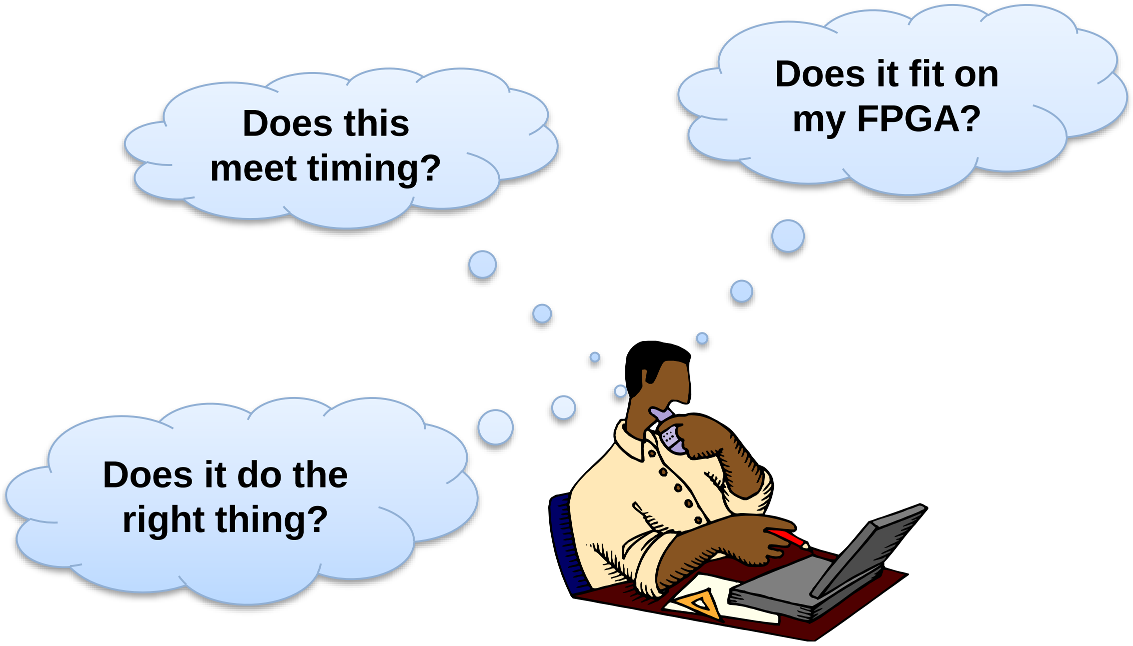




```
ENTITY nfp_mul_comp IS
  PORT( clk         :   IN    std_logic;
        reset       :   IN    std_logic;
        enb         :   IN    std_logic;
        nfp_in1     :   IN    std_logic_vector(31 DOWNTO 0);  -- ufix32
        nfp_in2     :   IN    std_logic_vector(31 DOWNTO 0);  -- ufix32
        nfp_out     :   OUT   std_logic_vector(31 DOWNTO 0)  -- ufix32
        );
END nfp_mul_comp;

ARCHITECTURE rtl OF nfp_mul_comp IS
  SIGNAL AS       : std_logic;  -- ufix1
  SIGNAL AE       : unsigned(7 DOWNTO 0);  -- ufix8
  SIGNAL AM       : unsigned(22 DOWNTO 0);  -- ufix23
   <snip>
BEGIN
```

```
» edit hdlcoderFocCurrentFloatScript
```

HDL Optimizations: What, How and Why?



Does this meet timing?

Does it fit on my FPGA?

Does it do the right thing?

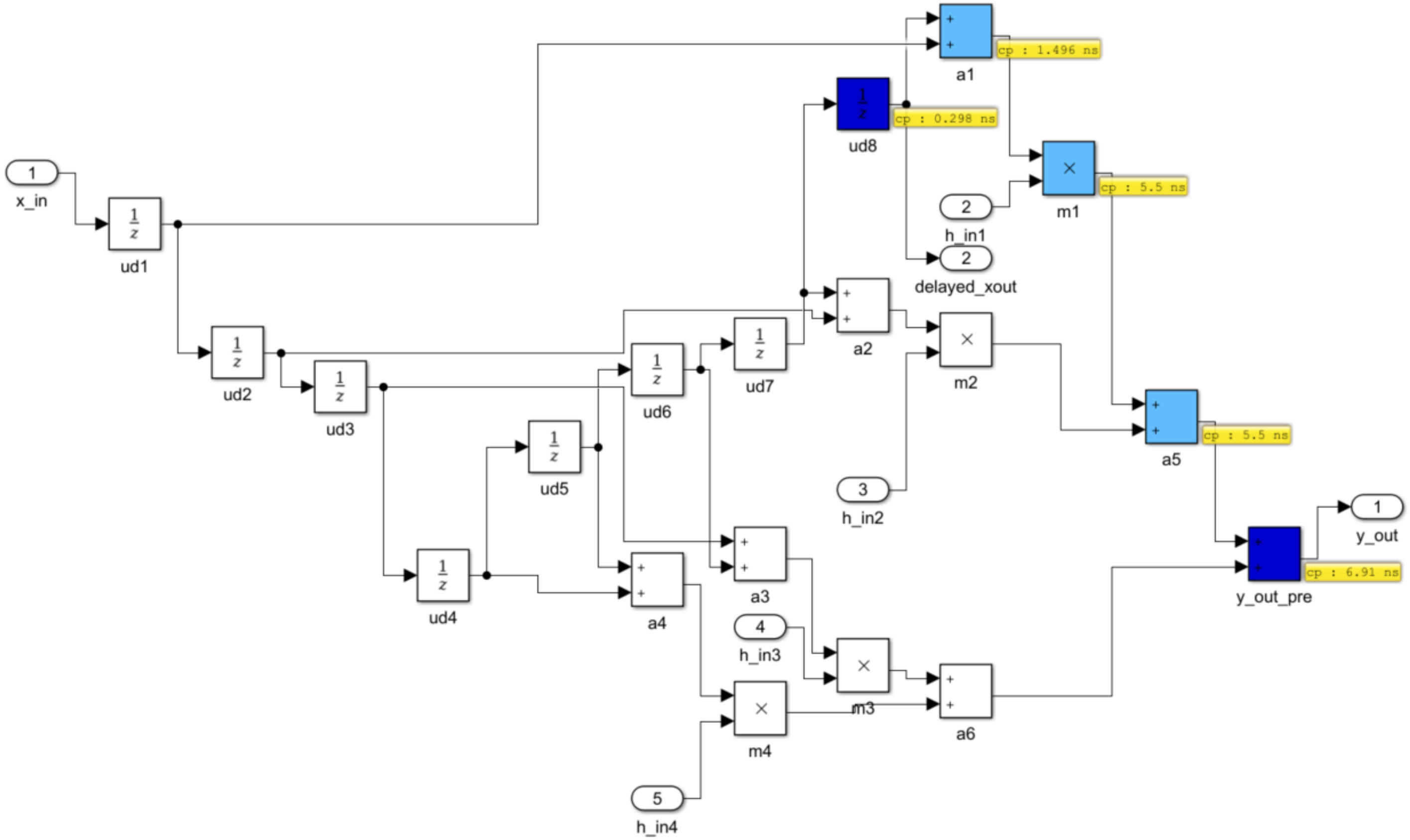
FPGA Engineer

The three golden questions:

1. Speed: Does it meet timing?
2. Area: Does it fit on my FPGA?
3. Validation: Does it do the right thing?

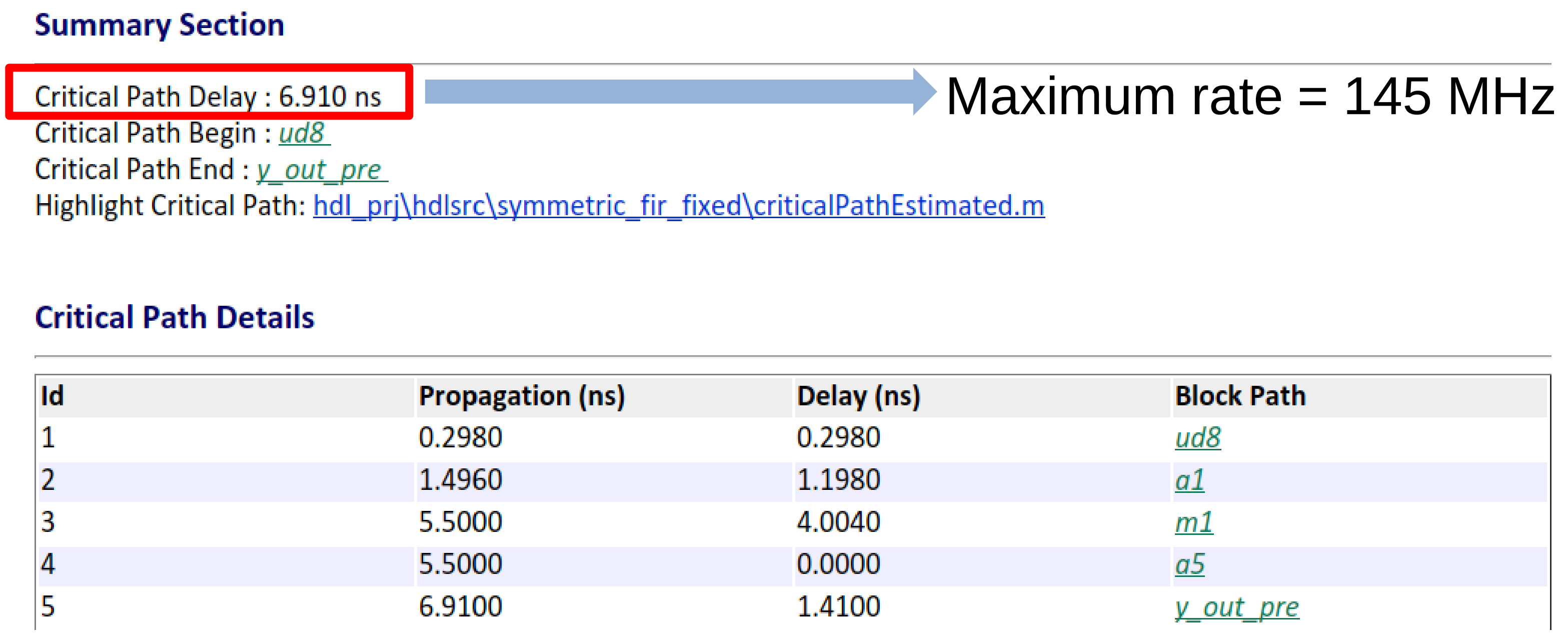
HDL optimizations assists the engineer in meeting these constraints

Critical Timing Path



- ✓ **Critical path highlighting**
- ✓ **Helps you identify speed bottlenecks**

Speed Optimization



Summary Section

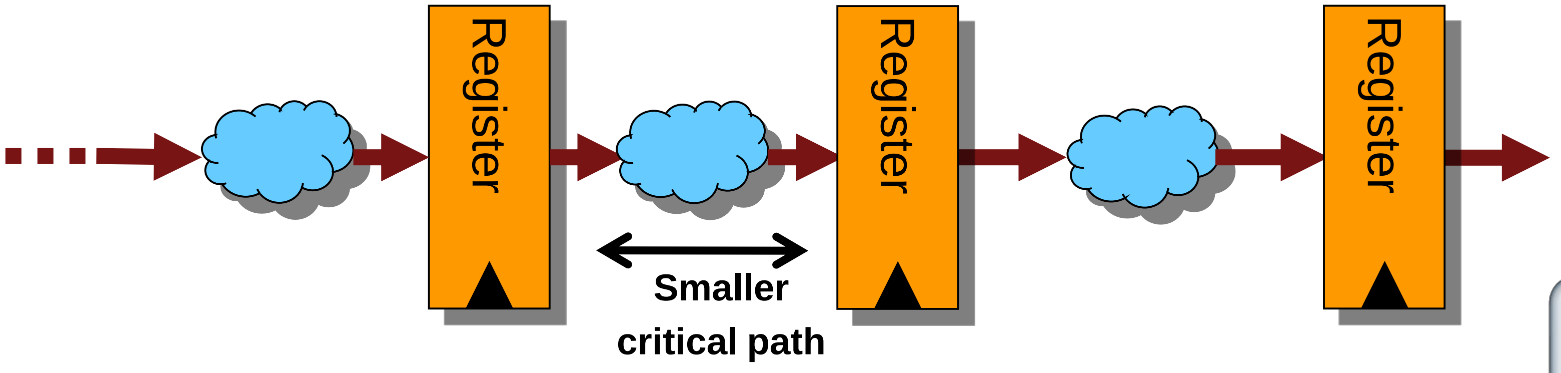
Critical Path Delay : 6.910 ns
Critical Path Begin : ud8
Critical Path End : y_out_pre
Highlight Critical Path: hdl_prj\hdlsrc\symmetric_fir_fixed\criticalPathEstimated.m

Maximum rate = 145 MHz

Critical Path Details

Id	Propagation (ns)	Delay (ns)	Block Path
1	0.2980	0.2980	ud8
2	1.4960	1.1980	a1
3	5.5000	4.0040	m1
4	5.5000	0.0000	a5
5	6.9100	1.4100	y_out_pre

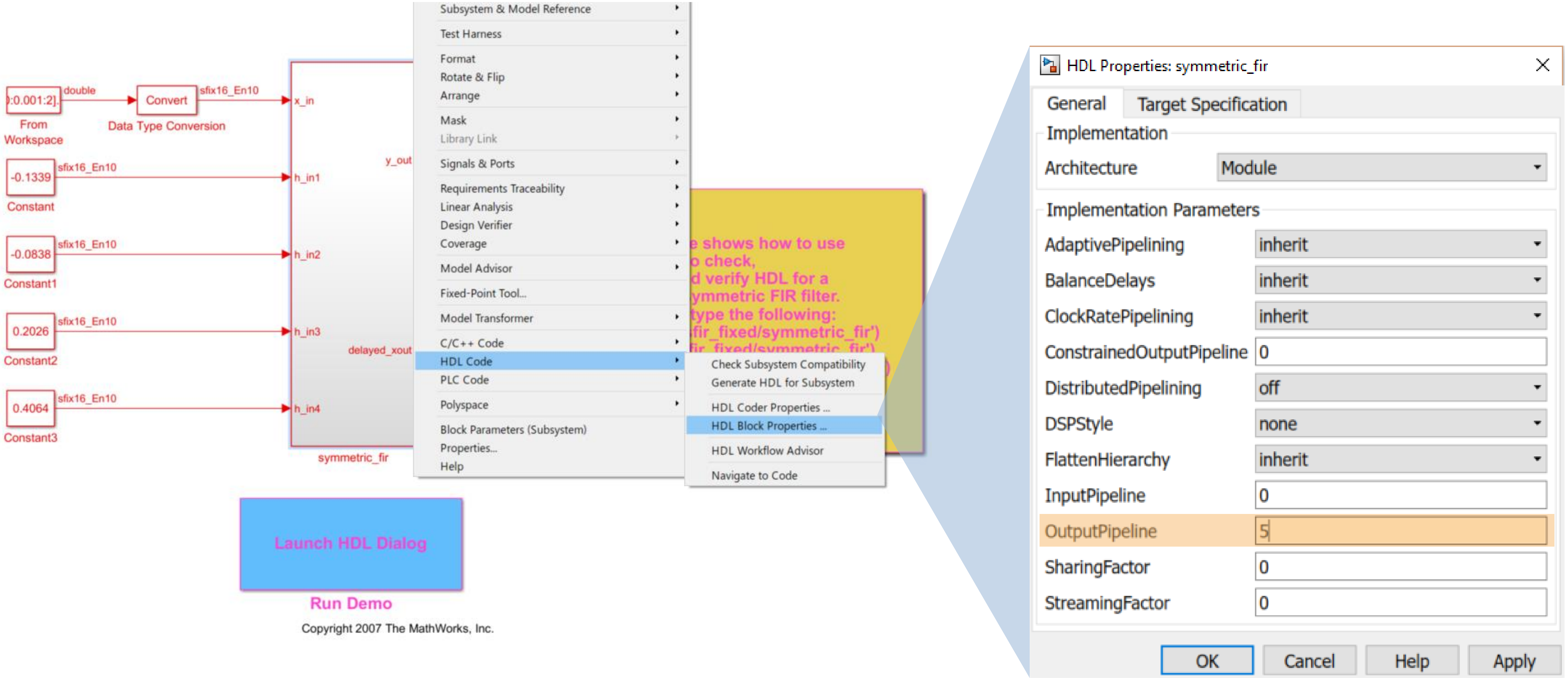
Is this the best rate that is achievable??



- ✓ **Automatic pipelining**
- ✓ **Helps you meet speed objectives**

Speed Optimization

Output Pipelining



e shows how to use
o check,
d verify HDL for a
ymmetric FIR filter.
type the following:
fir_fixed/symmetric_fir')
fir_fixed/symmetric_fir')

Implementation

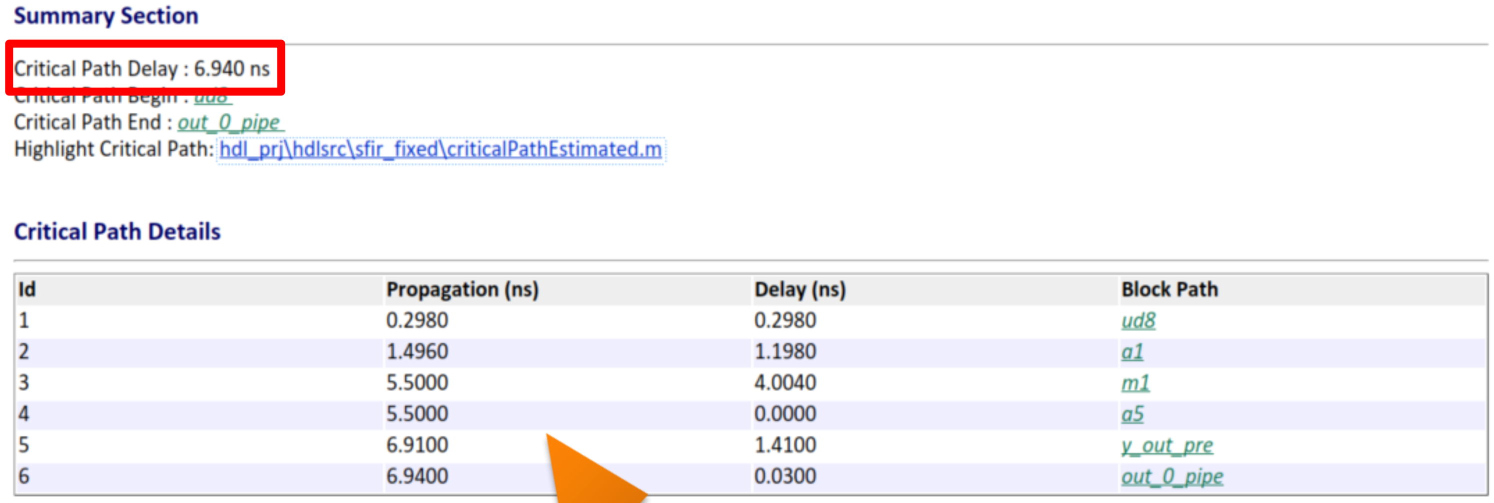
Architecture: Module

Implementation Parameters

AdaptivePipelining	inherit
BalanceDelays	inherit
ClockRatePipelining	inherit
ConstrainedOutputPipeline	0
DistributedPipelining	off
DSPStyle	none
FlattenHierarchy	inherit
InputPipeline	0
OutputPipeline	5
SharingFactor	0
StreamingFactor	0

Speed Optimization

Output Pipelining

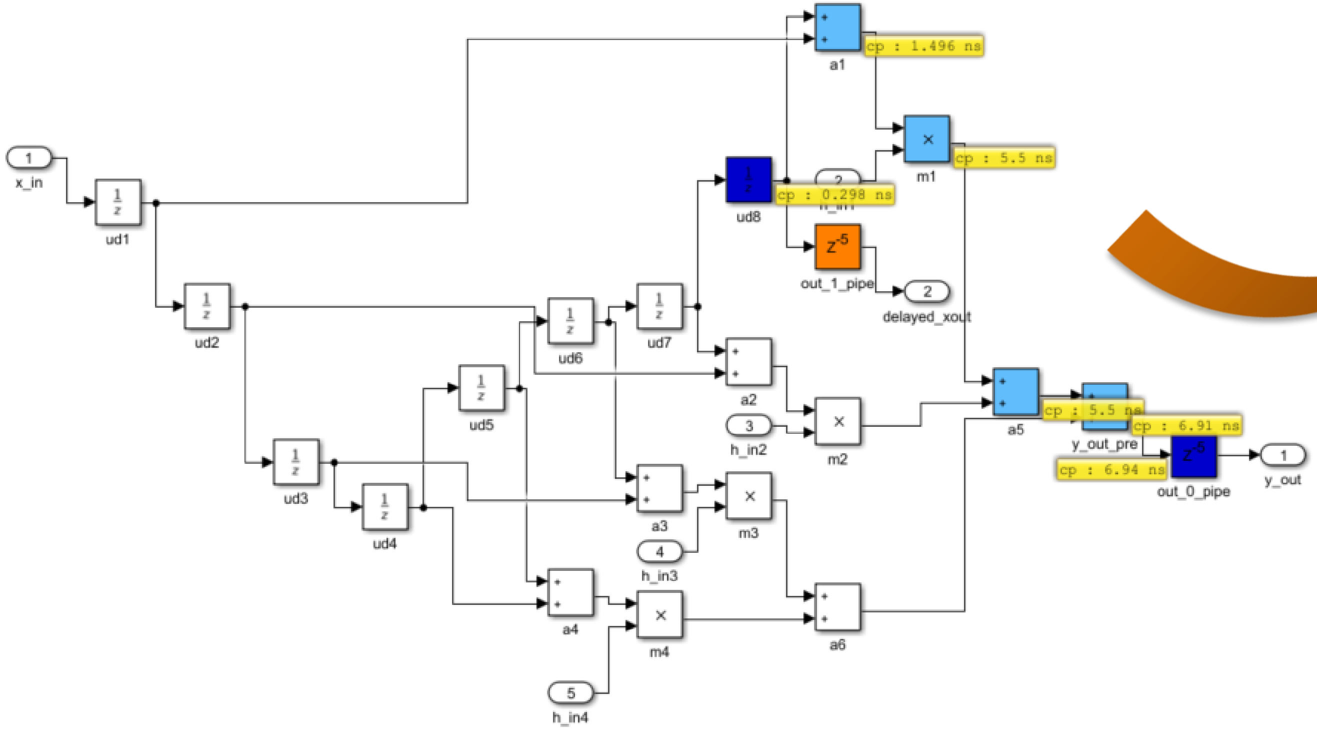


Summary Section

Critical Path Delay : 6.940 ns
Critical Path Begin : ud8
Critical Path End : out_0_pipe
Highlight Critical Path: hdl_prj\hdlsrc\sfir_fixed\criticalPathEstimated.m

Critical Path Details

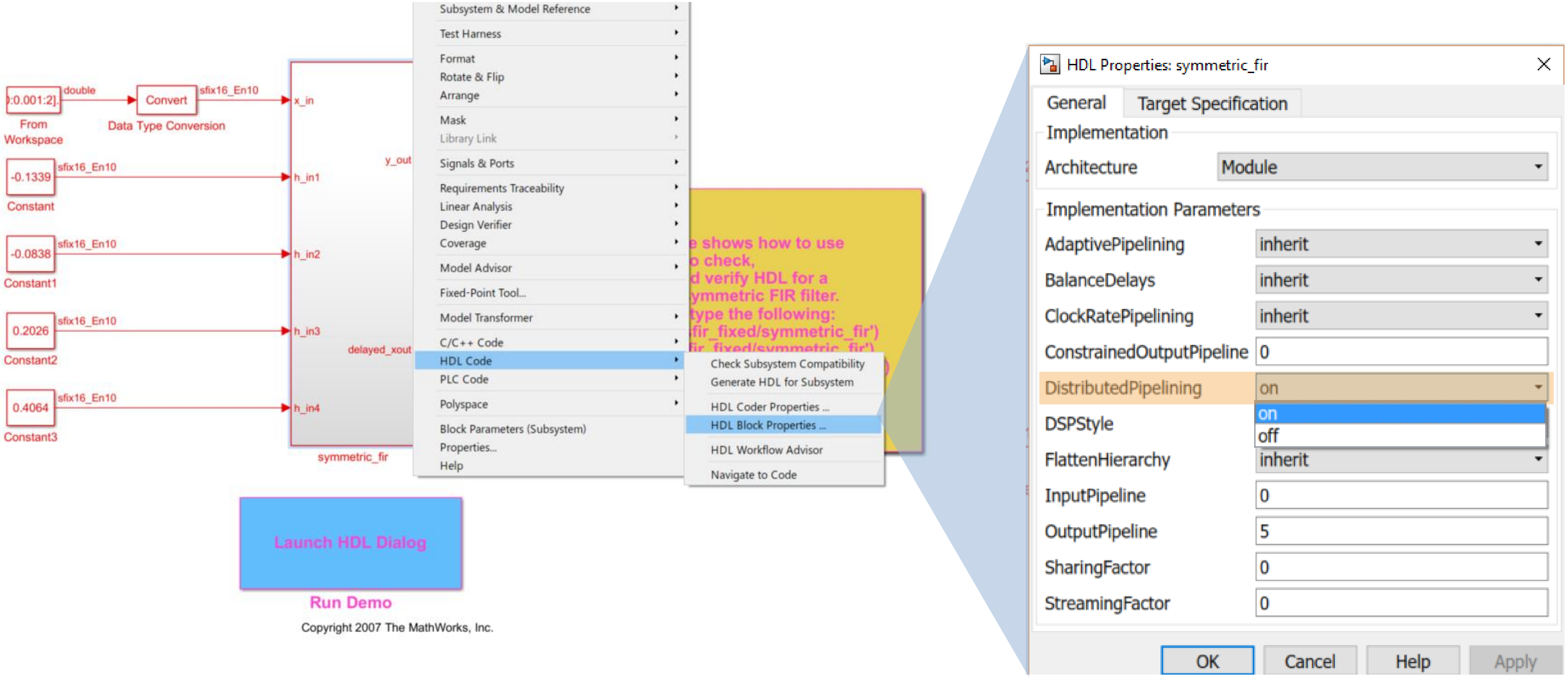
Id	Propagation (ns)	Delay (ns)	Block Path
1	0.2980	0.2980	ud8
2	1.4960	1.1980	a1
3	5.5000	4.0040	m1
4	5.5000	0.0000	a5
5	6.9100	1.4100	y_out_pre
6	6.9400	0.0300	out_0_pipe



Where do I place the pipeline registers??

Speed Optimization

Distributed Pipelining



Implementation

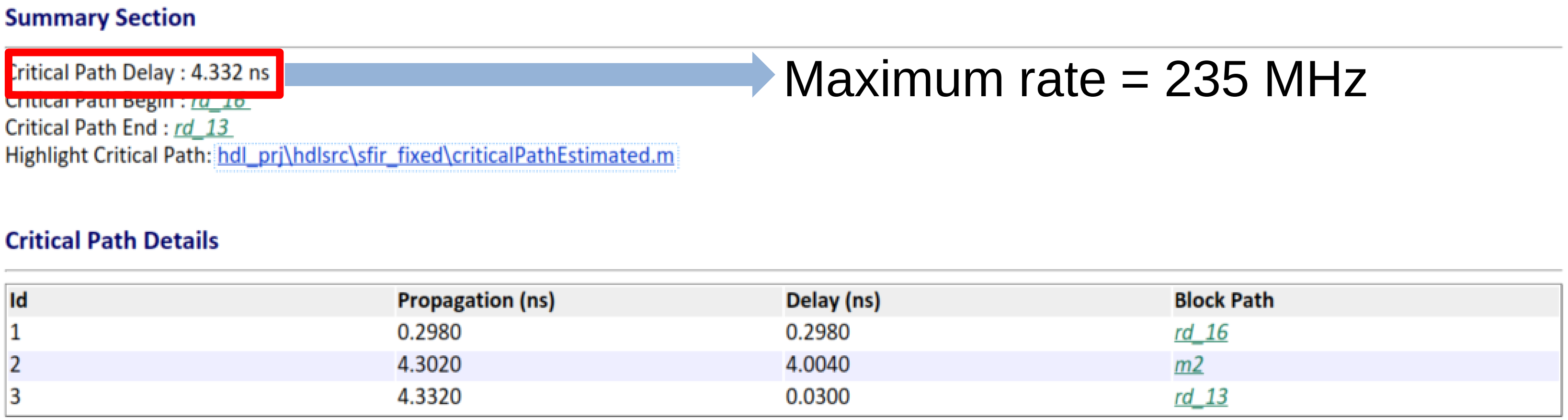
Architecture	Module

Implementation Parameters

AdaptivePipelining	inherit
BalanceDelays	inherit
ClockRatePipelining	inherit
ConstrainedOutputPipeline	0
DistributedPipelining	on
DSPStyle	
FlattenHierarchy	inherit
InputPipeline	0
OutputPipeline	5
SharingFactor	0
StreamingFactor	0

Speed Optimization

Distributed Pipelining



Summary Section

Critical Path Delay : 4.332 ns

Critical Path Begin : rd_16

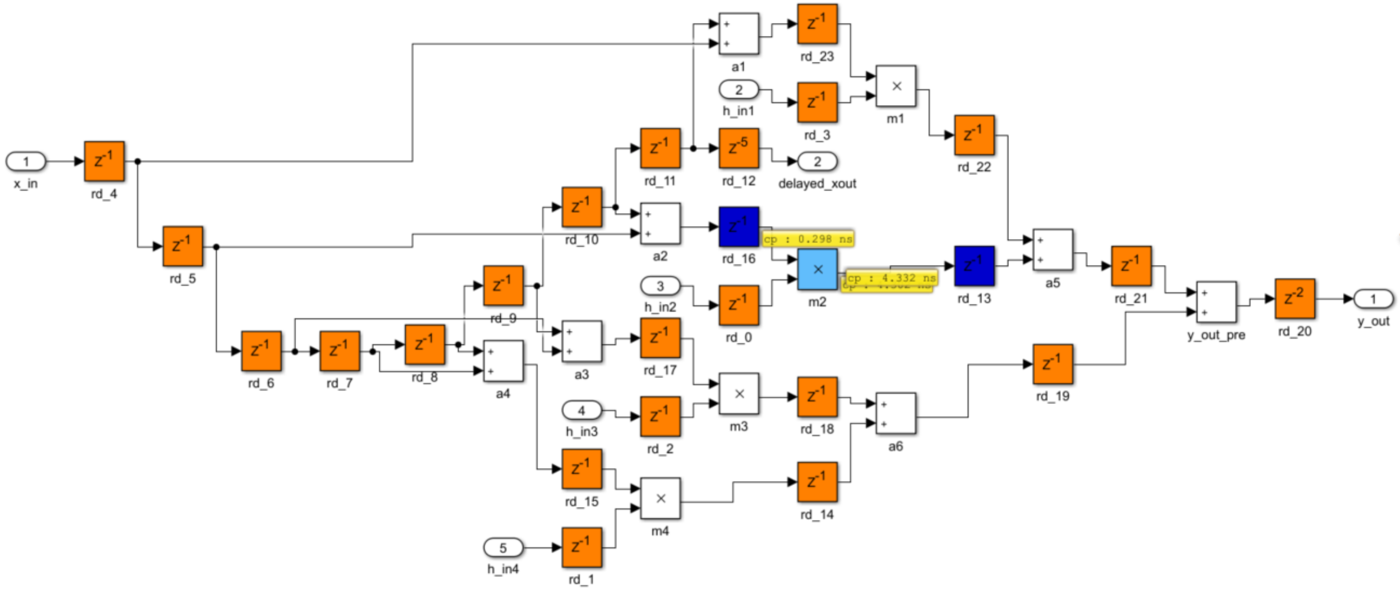
Critical Path End : rd_13

Highlight Critical Path: hdl_prj\hdlsrc\sfir_fixed\criticalPathEstimated.m

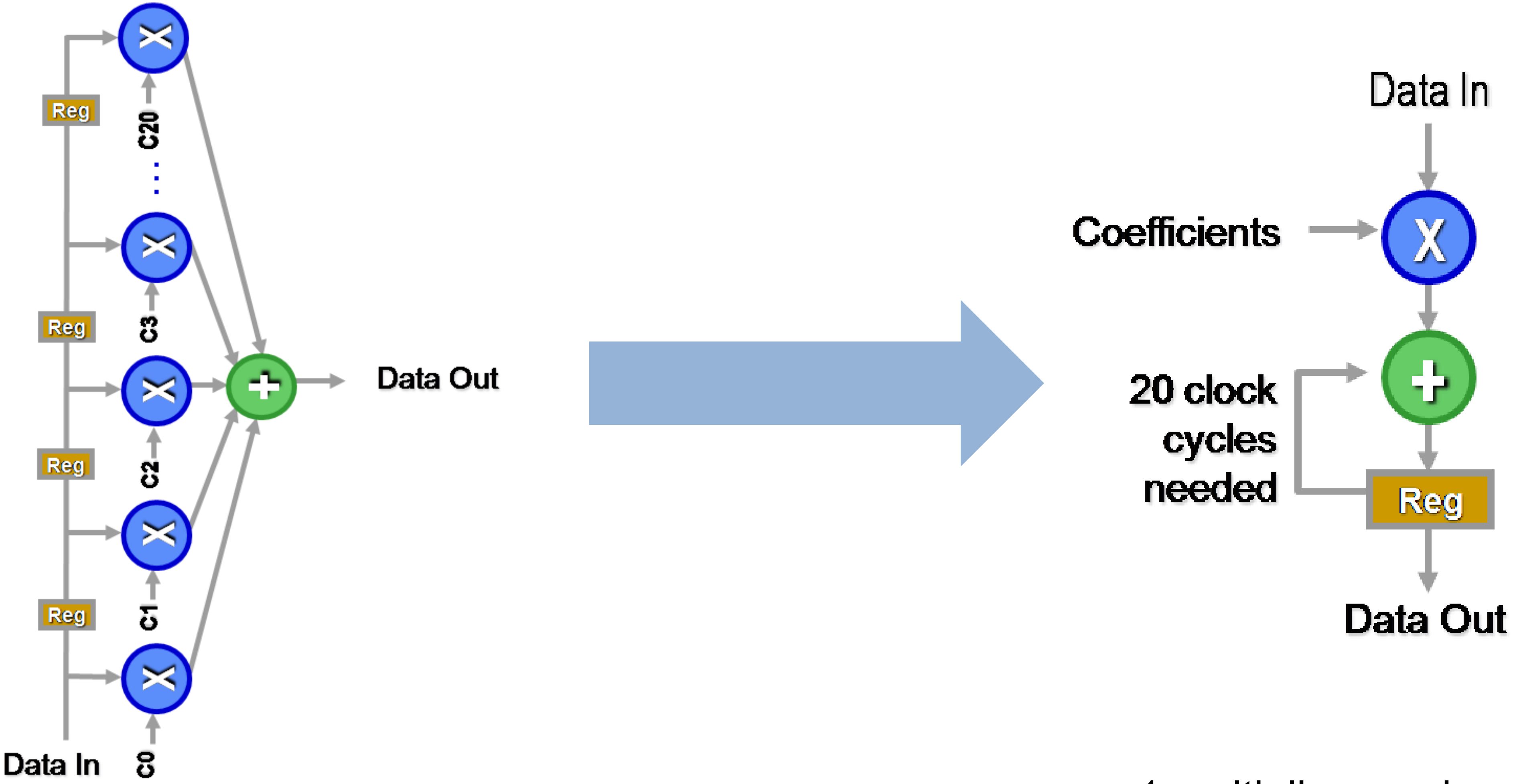
Critical Path Details

Id	Propagation (ns)	Delay (ns)	Block Path
1	0.2980	0.2980	rd_16
2	4.3020	4.0040	m2
3	4.3320	0.0300	rd_13

Maximum rate = 235 MHz



Area Optimization

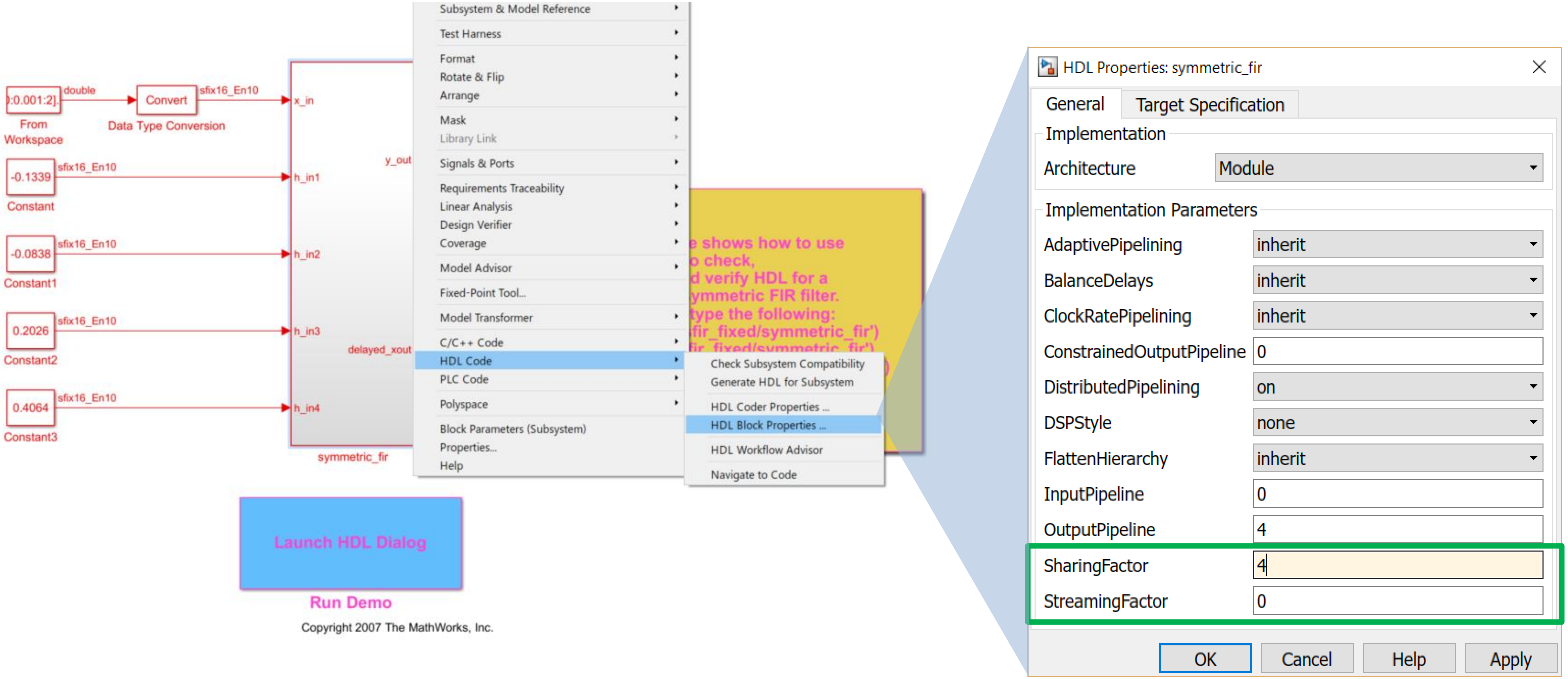


‘N’ (say 20) multipliers, each running at 1 clock cycle

1 multiplier running at ‘N’ (20) clock cycles

Area Optimization

Resource Sharing



HDL Properties: symmetric_fir

Implementation

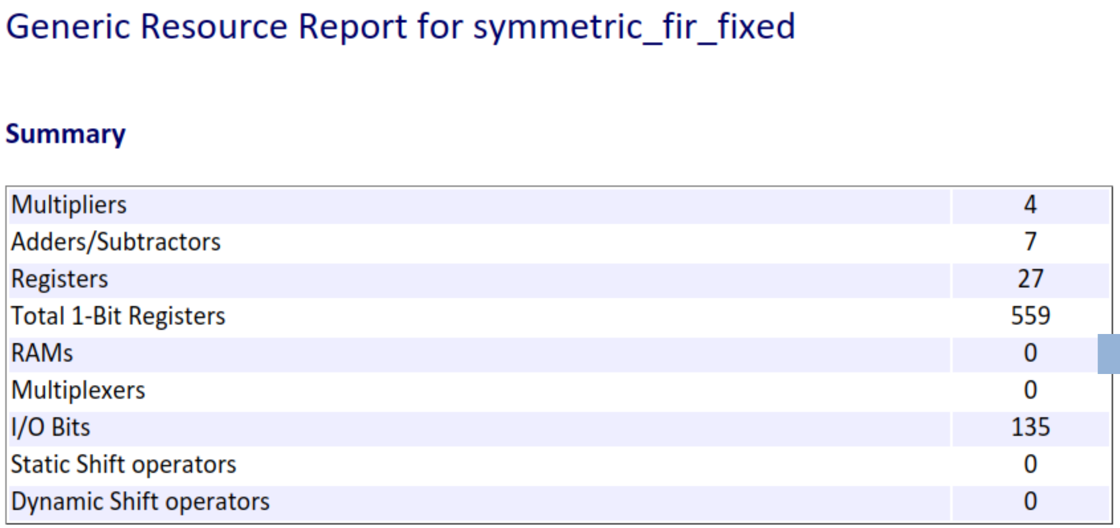
Architecture	Module

Implementation Parameters

AdaptivePipelining	inherit
BalanceDelays	inherit
ClockRatePipelining	inherit
ConstrainedOutputPipeline	0
DistributedPipelining	on
DSPStyle	none
FlattenHierarchy	inherit
InputPipeline	0
OutputPipeline	4
SharingFactor	4
StreamingFactor	0

Area Optimization

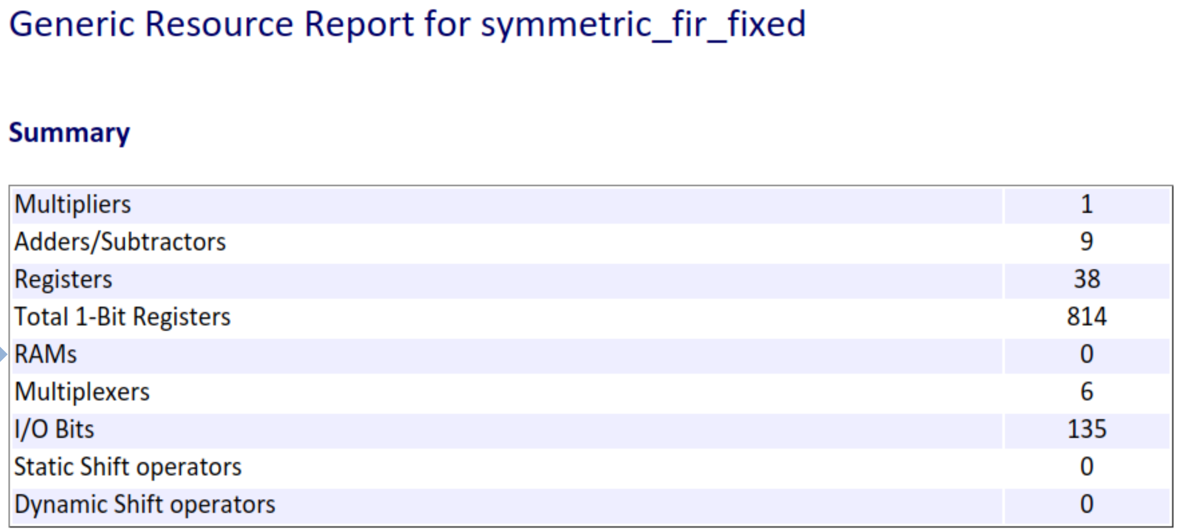
Resource Sharing



Generic Resource Report for symmetric_fir_fixed

Summary

Multipliers	4
Adders/Subtractors	7
Registers	27
Total 1-Bit Registers	559
RAMs	0
Multiplexers	0
I/O Bits	135
Static Shift operators	0
Dynamic Shift operators	0



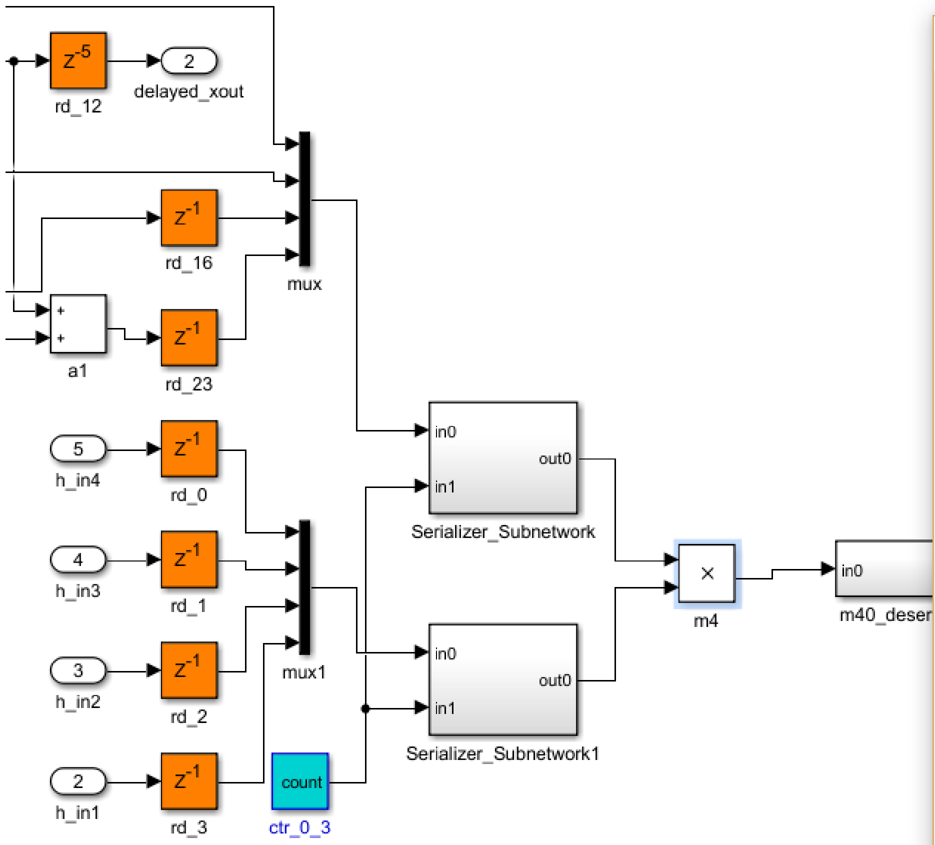
Generic Resource Report for symmetric_fir_fixed

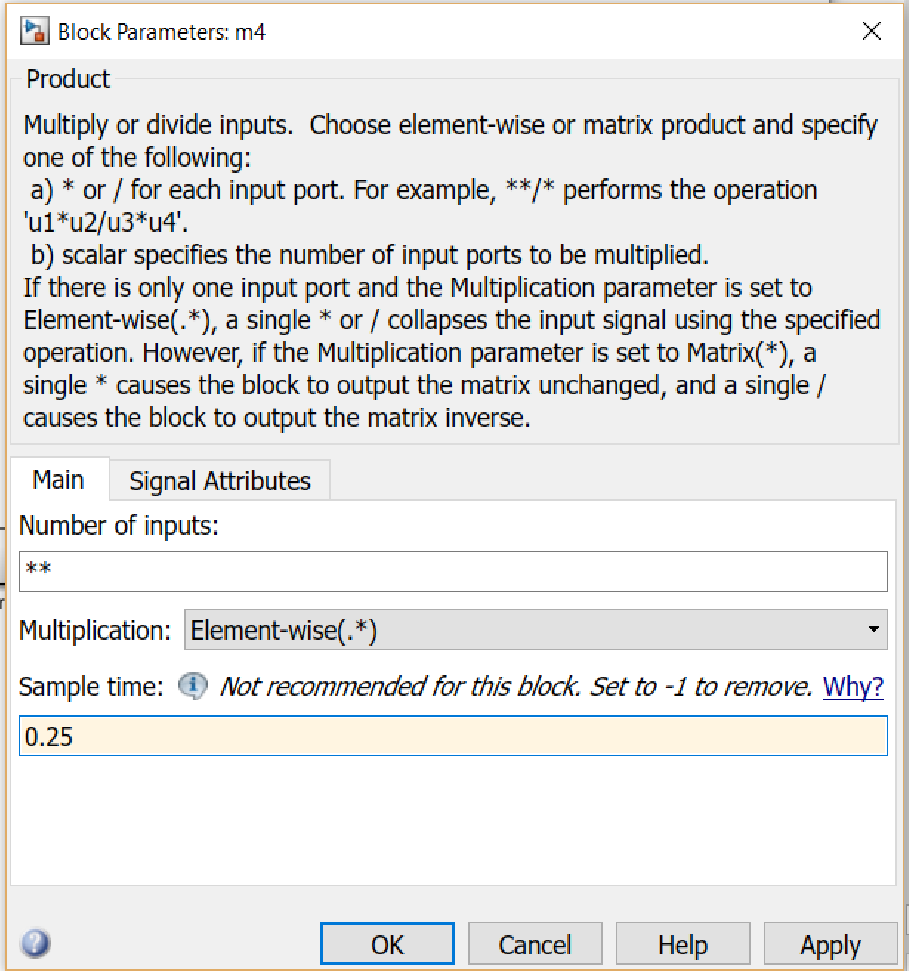
Summary

Multipliers	1
Adders/Subtractors	9
Registers	38
Total 1-Bit Registers	814
RAMs	0
Multiplexers	6
I/O Bits	135
Static Shift operators	0
Dynamic Shift operators	0

Area Optimization

Resource Sharing





Block Parameters: m4

Product

Multiply or divide inputs. Choose element-wise or matrix product and specify one of the following:
a) * or / for each input port. For example, **/* performs the operation 'u1*u2/u3*u4'.
b) scalar specifies the number of input ports to be multiplied.
If there is only one input port and the Multiplication parameter is set to Element-wise(.*), a single * or / collapses the input signal using the specified operation. However, if the Multiplication parameter is set to Matrix(*), a single * causes the block to output the matrix unchanged, and a single / causes the block to output the matrix inverse.

Main | Signal Attributes

Number of inputs: **

Multiplication: Element-wise(.*)

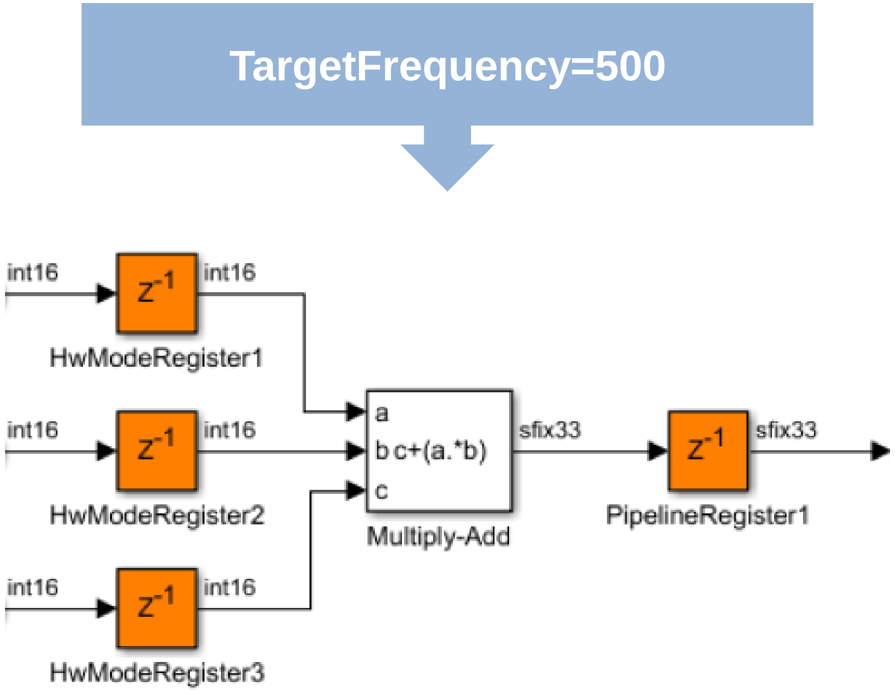
Sample time: *Not recommended for this block. Set to -1 to remove.* Why?

0.25

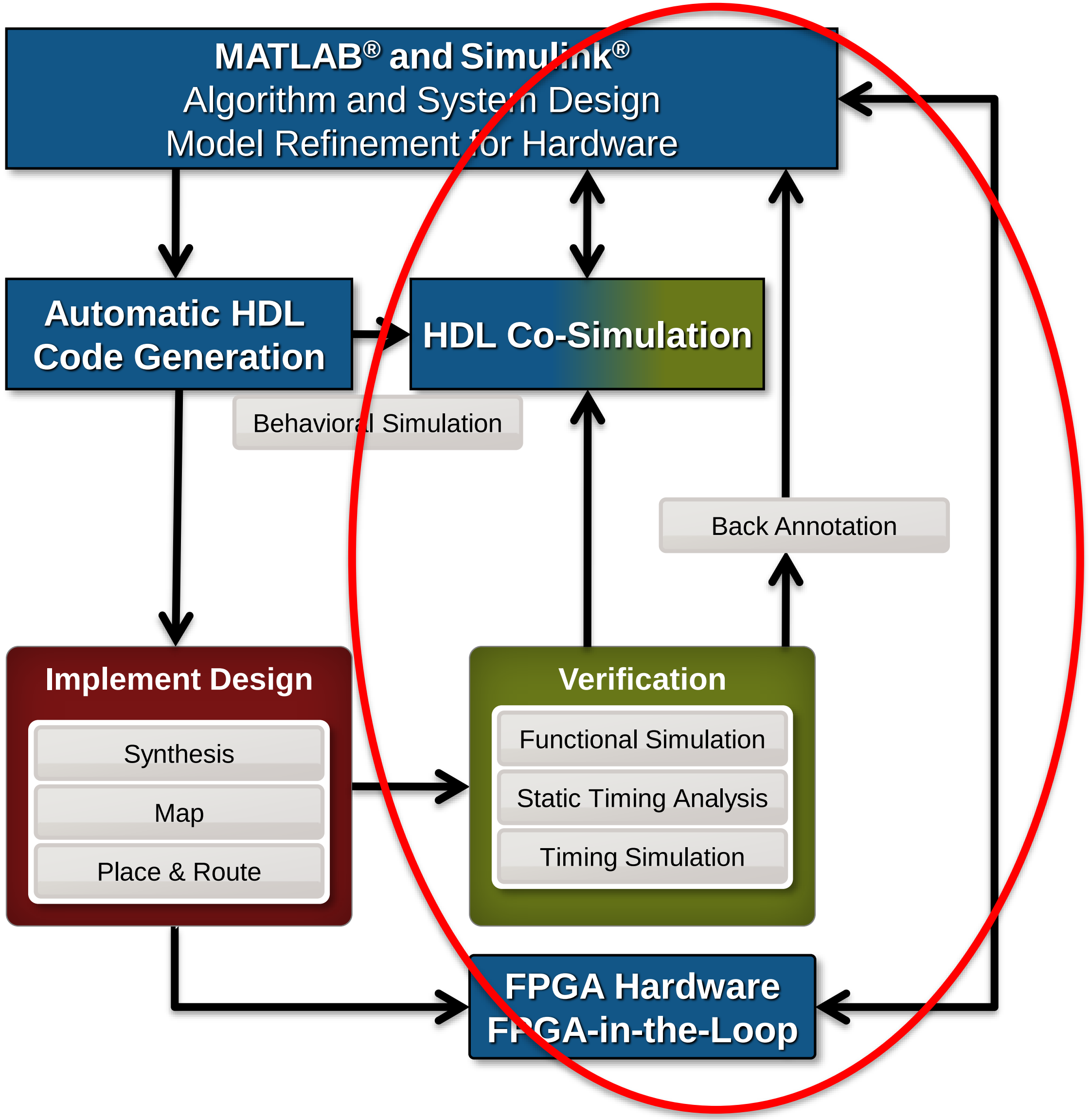
What's new?
Adaptive Pipelining

Specify synthesis tool and target clock frequency for automatic pipeline insertion and balancing

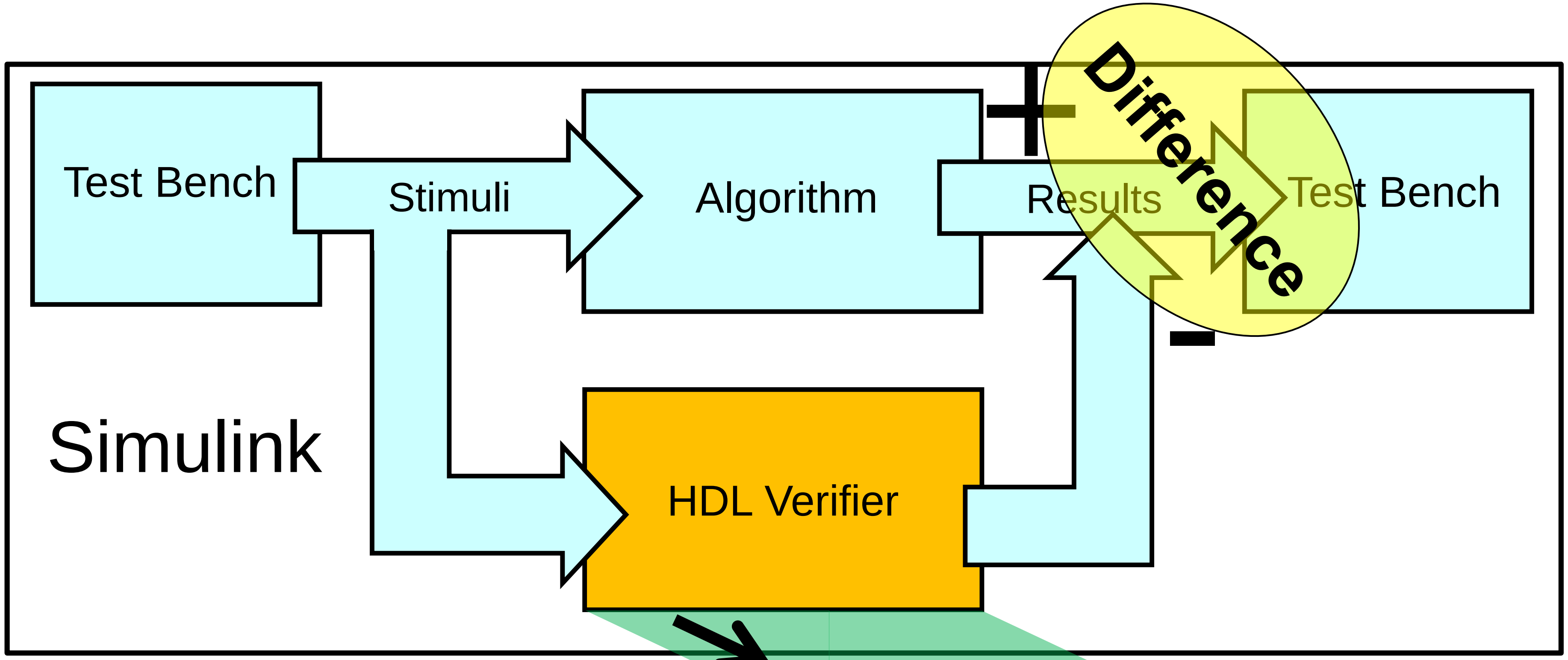
- Automatically inserts pipeline registers to meet target frequency
  - On by default
  - Adds pipeline registers on parallel paths to balance number of stages
- Good for:
  - Getting started prototyping FPGAs without worrying about manually inserting Delay blocks



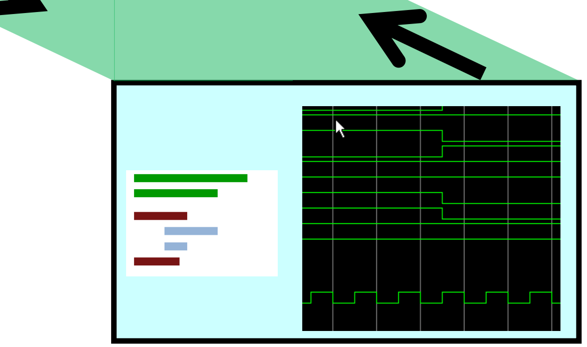
Integrated HDL Verification



Co-Simulation with HDL Simulator

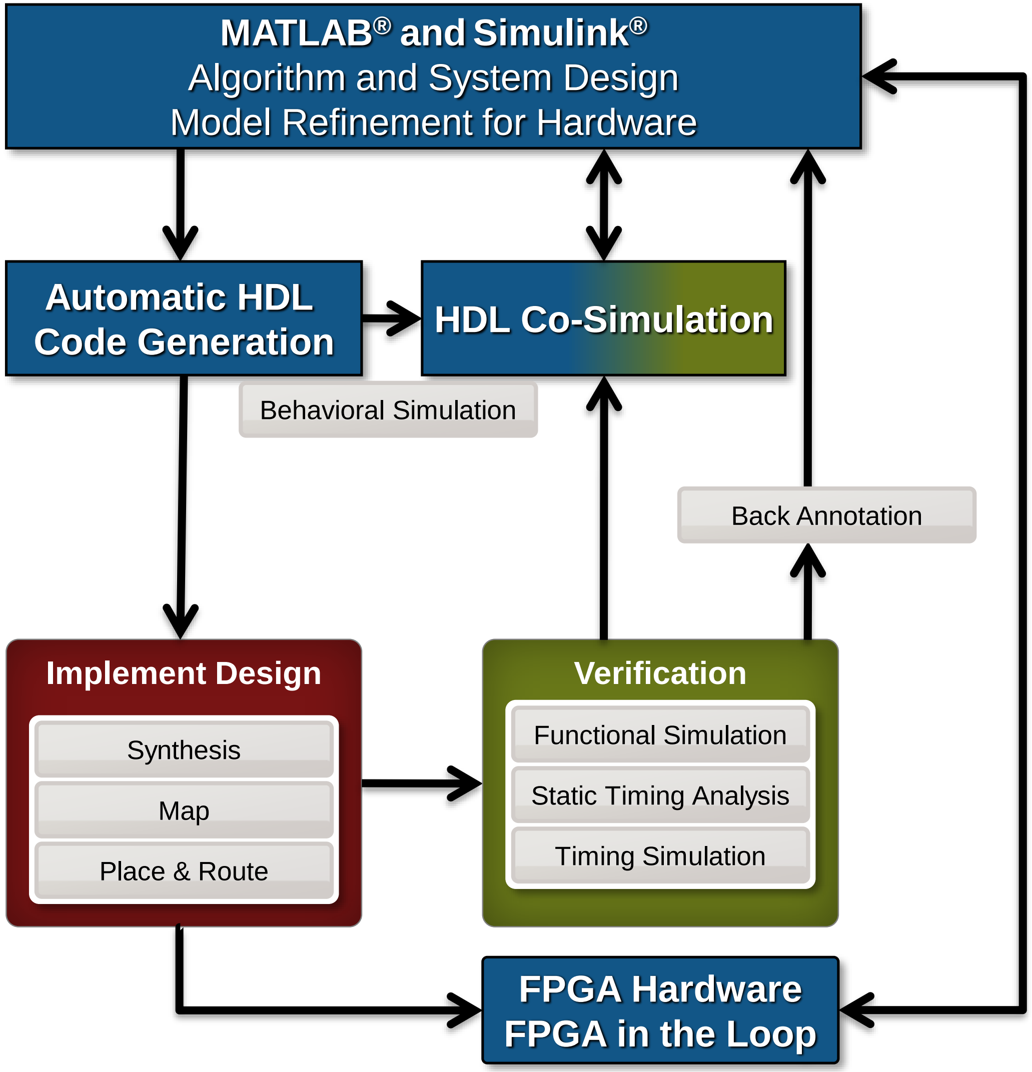


- Proof your HDL matches the MATLAB/Simulink specification
- Re-using MATLAB/Simulink testbench



HDL Simulator

Model-Based Design for Implementation



MATLAB® and Simulink®
Algorithm and System Design
Model Refinement for Hardware

Automatic HDL Code Generation

HDL Co-Simulation

Behavioral Simulation

Back Annotation

Implement Design
- Synthesis
- Map
- Place & Route

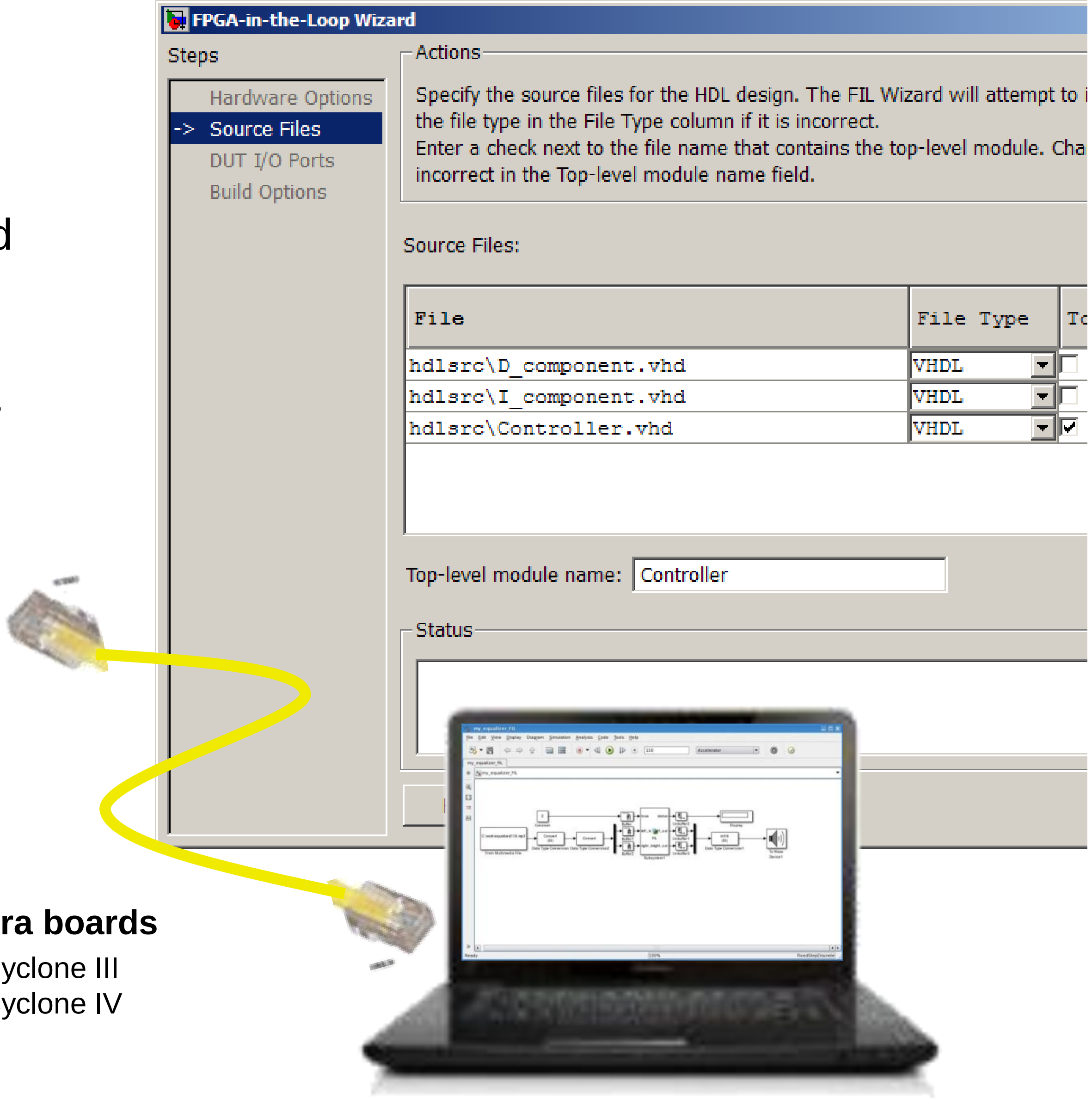
Verification
- Functional Simulation
- Static Timing Analysis
- Timing Simulation

FPGA Hardware
FPGA in the Loop

FPGA Hardware

FPGA-in-the-Loop (FIL) for any HDL code

- Part of HDL Verifier
- Easy to setup using FIL Wizard
- Fast simulation
  - HDL runs on FPGA
  - Gigabit Ethernet data transfer



FPGA-in-the-Loop Wizard

Steps

- Hardware Options
- -> Source Files
- DUT I/O Ports
- Build Options

Actions

Specify the source files for the HDL design. The FIL Wizard will attempt to i
the file type in the File Type column if it is incorrect.
Enter a check next to the file name that contains the top-level module. Cha
incorrect in the Top-level module name field.

Source Files:

File	File Type	To
hdlsrc\D_component.vhd	VHDL	□
hdlsrc\I_component.vhd	VHDL	□
hdlsrc\Controller.vhd	VHDL	☑

Top-level module name: Controller

Status

Supported Xilinx boards

KC 705
ML605
ML505
ML506
ML507
XUP Atlys
XUP-V5

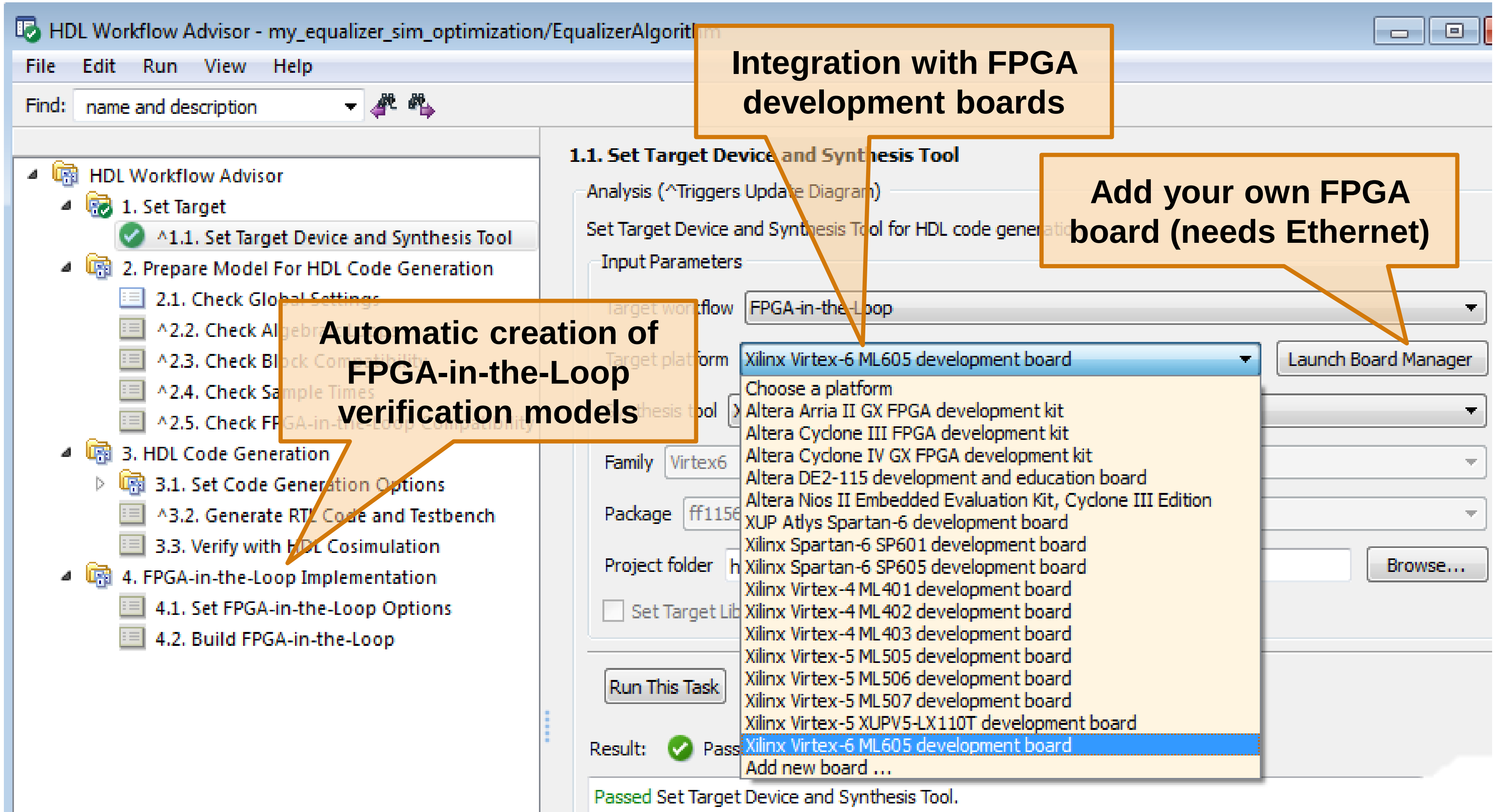
SP605
SP601
ML401
ML402
ML403

Supported Altera boards

Arria II
DE2-115

Cyclone III
Cyclone IV

Automation FPGA-in-the-Loop Verification



HDL Workflow Advisor - my_equalizer_sim_optimization/EqualizerAlgorithm

- HDL Workflow Advisor
  - 1. Set Target
    - ^1.1. Set Target Device and Synthesis Tool
  - 2. Prepare Model For HDL Code Generation
    - 2.1. Check Global Settings
    - ^2.2. Check Algebr
    - ^2.3. Check Block Compatibility
    - ^2.4. Check Sample Times
    - ^2.5. Check FPGA-in-the-Loop Compatibility
  - 3. HDL Code Generation
    - 3.1. Set Code Generation Options
    - ^3.2. Generate RTL Code and Testbench
    - 3.3. Verify with HDL Cosimulation
  - 4. FPGA-in-the-Loop Implementation
    - 4.1. Set FPGA-in-the-Loop Options
    - 4.2. Build FPGA-in-the-Loop

1.1. Set Target Device and Synthesis Tool

Analysis (^Triggers Update Diagram)

Set Target Device and Synthesis Tool for HDL code generati

Input Parameters

Target workflow: FPGA-in-the-Loop

Target platform: Xilinx Virtex-6 ML605 development board

Launch Board Manager

Synthesis tool: X

Family: Virtex6

Package: ff1156

Project folder: h

Browse...

☐ Set Target Lib

- Choose a platform
- Altera Arria II GX FPGA development kit
- Altera Cyclone III FPGA development kit
- Altera Cyclone IV GX FPGA development kit
- Altera DE2-115 development and education board
- Altera Nios II Embedded Evaluation Kit, Cyclone III Edition
- XUP Atlys Spartan-6 development board
- Xilinx Spartan-6 SP601 development board
- Xilinx Spartan-6 SP605 development board
- Xilinx Virtex-4 ML401 development board
- Xilinx Virtex-4 ML402 development board
- Xilinx Virtex-4 ML403 development board
- Xilinx Virtex-5 ML505 development board
- Xilinx Virtex-5 ML506 development board
- Xilinx Virtex-5 ML507 development board
- Xilinx Virtex-5 XUPV5-LX110T development board
- Xilinx Virtex-6 ML605 development board
- Add new board ...

Run This Task

Result: ✓ Pass

Passed Set Target Device and Synthesis Tool.

Integration with FPGA development boards

Add your own FPGA board (needs Ethernet)

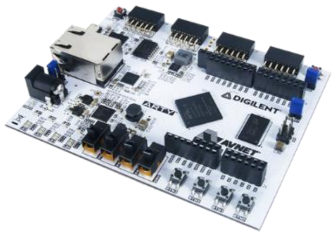
Automatic creation of FPGA-in-the-Loop verification models

New FPGA Families and Boards Supported by FIL

R2017a

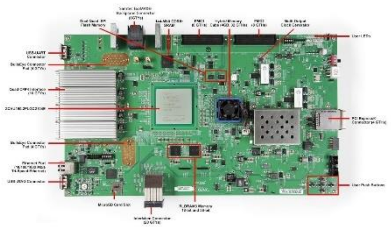
- FPGA Family
  - Virtex Ultrascale

- FPGA board
  - Artix-7 Arty (JTAG)



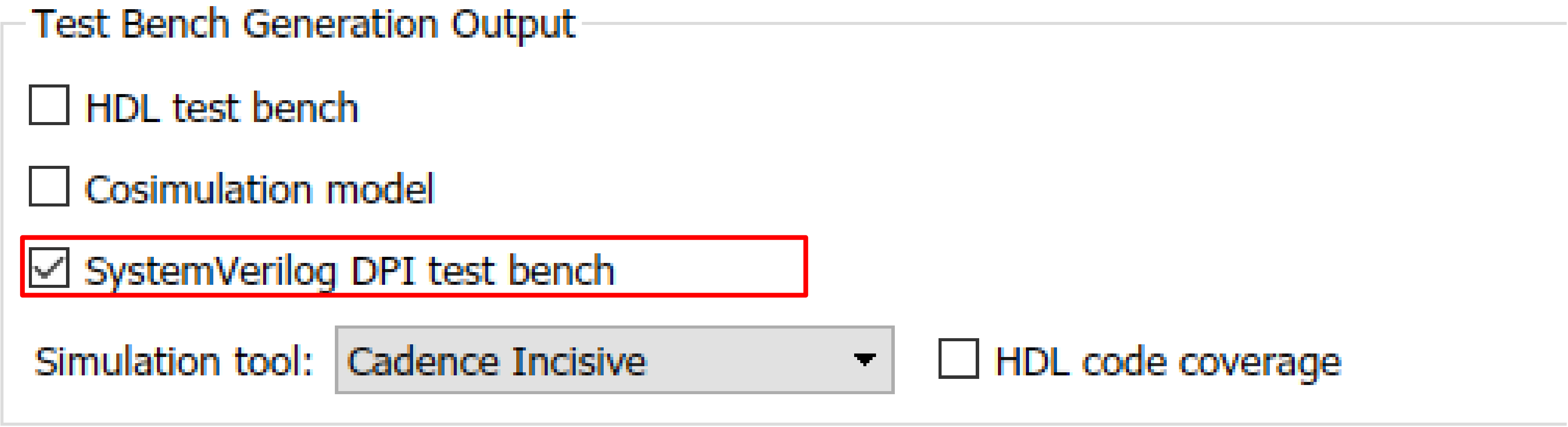
  - Virtex-7 VC709 (JTAG, PCIe)

  - Virtex Ultrascale VCU110 (JTAG)



SystemVerilog DPI Test Bench

- Previously only available via command-line interface
- Now it’s available in Config Param as well as HDL Workflow Advisor



Test Bench Generation Output

□ HDL test bench

□ Cosimulation model

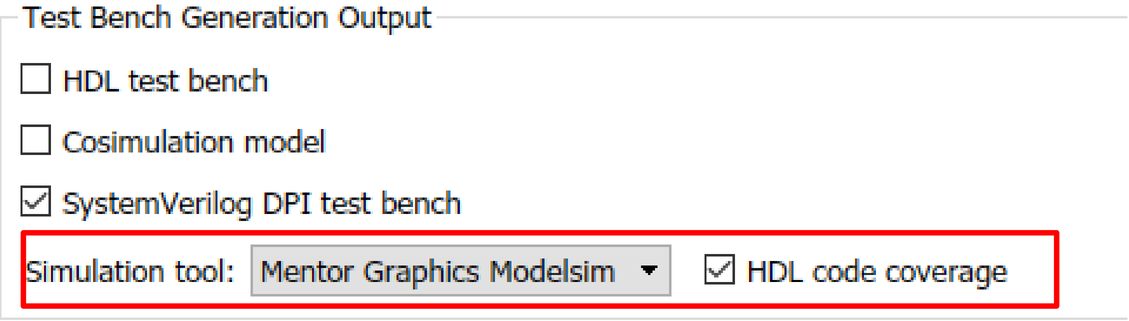
☑ SystemVerilog DPI test bench

Simulation tool: Cadence Incisive ▾ □ HDL code coverage

HDL Verifier: HDL Code Coverage

Activate HDL simulator code coverage in generated test benches

- Works for cosimulation, SystemVerilog DPI, or vector-based testbenches
- Supports Mentor Graphics Questa Sim and Cadence Incisive



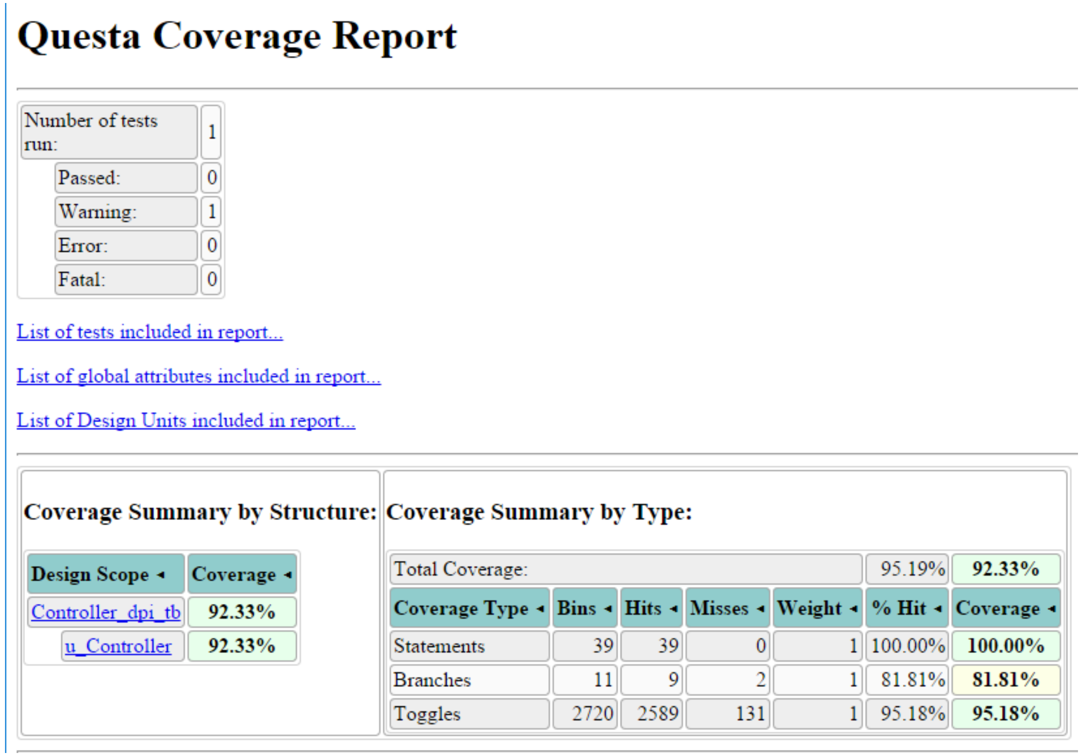
Test Bench Generation Output

☐ HDL test bench

☐ Cosimulation model

☑ SystemVerilog DPI test bench

Simulation tool: Mentor Graphics Modelsim ▾ ☑ HDL code coverage



Questa Coverage Report

Number of tests run:	1
Passed:	0
Warning:	1
Error:	0
Fatal:	0

List of tests included in report...

List of global attributes included in report...

List of Design Units included in report...

Coverage Summary by Structure:

Design Scope	Coverage
Controller_dpi_tb	92.33%
u_Controller	92.33%

Coverage Summary by Type:

Total Coverage:					95.19%	92.33%
Coverage Type	**Bins**	**Hits**	**Misses**	**Weight**	**% Hit**	**Coverage**
Statements	39	39	0	1	100.00%	100.00%
Branches	11	9	2	1	81.81%	81.81%
Toggles	2720	2589	131	1	95.18%	95.18%

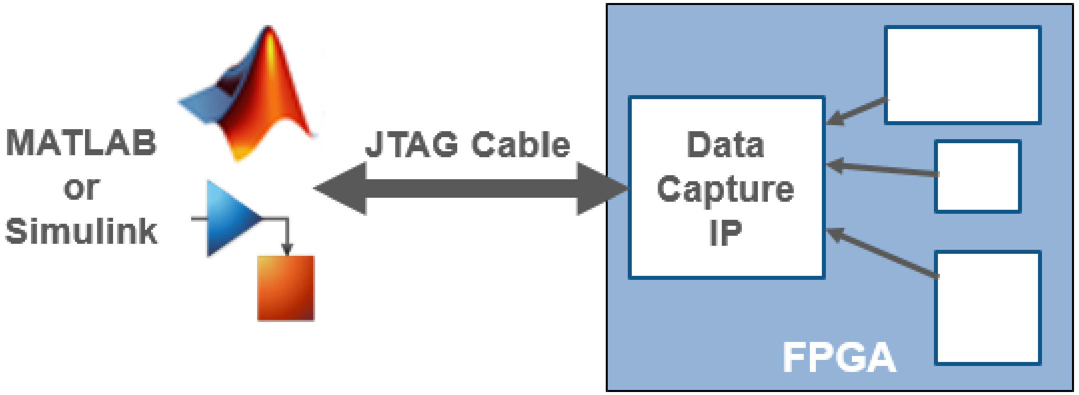
```
» makehdltb('sfir_fixed/symmetric_fir',...
» 'GenerateSVDPITestBench','ModelSim', ...
» 'HDLCodeCoverage', 'on', )
```

HDL Verifier: FPGA Data Capture

R2017a

Probe internal FPGA signals to analyze in MATLAB or Simulink

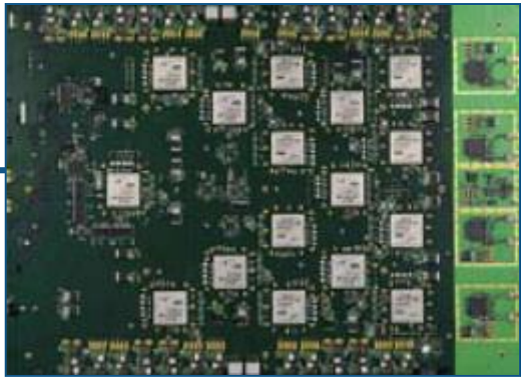
- Debug signals in a free-running FPGA directly in MATLAB or Simulink
- Generates a block to add into the VHDL/Verilog design going onto the FPGA
- Collects and visualizes the data in MATLAB or Simulink
- [Demo video]



Available as part of HDL Verifier Xilinx/Intel hardware support packages

```
» generateFPGADataCaptureIP
```

Harris Accelerates Verification of Signal Processing FPGAs



Harris FPGA-based system.

Challenge

Streamline a time-consuming manual process for testing signal processing FPGA implementation

Solution

Use HDL Verifier to verify the HDL design from within MATLAB

Results

- Functional verification time cut by more than 85%
- 100% of planned test cases completed
- Design implemented defect-free

> **"HDL Verifier enabled us to greatly reduce functional verification development time by providing a direct cosimulation interface between our MATLAB model and our logic simulator. As a result, we verified our design earlier, identified problems faster, completed more tests, and compressed our entire development cycle."**
>
> **Jason Plew**
> **Harris Corporation**

Link to user story

Lockheed Martin Develops Configurable, Space-Qualified Digital Channelizer Using MathWorks Tools

Artist's rendition of one of the satellites that will carry Lockheed Martin's digital channelizer.

Challenge

Design and implement a reconfigurable, space-qualified digital channelizer

Solution

Use Simulink to model and simulate the system, and HDL Verifier with Mentor Graphics ModelSim to verify the VHDL implementation

Results

- Verification time reduced by 90%
- Overall development time shortened by eight months
- Key algorithms reused, saving 50% of design effort on subsequent projects

> "With Simulink and HDL Verifier, simulation and verification are performed in one environment. As a result, we can test the design from end to end, improving quality and ensuring design accuracy and validity."
>
> Bradford Watson
> Lockheed Martin Space Systems

Link to user story

Summary

- **Respect project timeline**
  - Discover issues early through simulation
  - Fast code turnarounds allow better design trade-offs
- **Collaborate in multidisciplinary teams**
  - Use one Model for Design and Implementation
  - Seamlessly integrate version management
  - Graphically compare models
- **Create working code**
  - Analyze fixed-point impact before going to implementation
  - Auto-generate bug free code
  - Verify early through co-simulation with FPGA’s
- **Achieve required efficiency**
  - Optimize through advisors and automatic optimizations

Call To Action

Learn more with recorded webinars & videos

- Accelerate Design Space Exploration Using HDL Coder Optimizations
- HDL Implementation and Verification of a High-Performance FFT
- Using Custom Boards for FPGA-in-the-Loop Verification
- A Guided Workflow for Zynq Using MATLAB and Simulink
- HDL Verifier: FPGA Data Capture

MathWorks | *Training Services*

Generating HDL Code from Simulink

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

Programming Xilinx Zynq SoCs with MATLAB and Simulink

two-day course focuses on developing and configuring models in Simulink® and deploying on Xilinx® Zynq®-7000 All Programmable SoCs. For Simulink users who intend to generate, validate, and deploy embedded code and HDL code for software/hardware codesign using Embedded Coder® and HDL Coder™.
A ZedBoard™ is provided to each attendee for use throughout the course. The board is programmed during the class and is yours to keep after the training.

Topics include:

- Zynq platform overview and environment setup, introduction to Embedded Coder and HDL Coder
- IP core generation and deployment, Using AXI4 interface
- Processor-in-the-loop verification, data interface with real-time application
- Integrating device drivers, custom reference design

MathWorks® | Training Services

DSP for FPGAs

This three-day course will review DSP fundamentals from the perspective of implementation within the FPGA fabric. Particular emphasis will be given to highlighting the cost, with respect to both resources and performance, associated with the implementation of various DSP techniques and algorithms.

Topics include:

- Introduction to FPGA hardware and technology for DSP applications
- DSP fixed-point arithmetic
- Signal flow graph techniques
- HDL code generation for FPGAs
- Fast Fourier Transform (FFT) Implementation
- Design and implementation of FIR, IIR and CIC filters
- CORDIC algorithm
- Design and implementation of adaptive algorithms such as LMS and QR algorithm
- Techniques for synchronisation and digital communications timing recovery

MathWorks®

Accelerating the pace of engineering and science

Speaker Details

Email: tabrez.khan@mathworks.in

Vidya.viswanthan@mathworks.in

Contact MathWorks India

Products/Training Enquiry Booth

Call: 080-6632-6000

Email: info@mathworks.in

Your feedback is valued.

Please complete the feedback form provided to you.