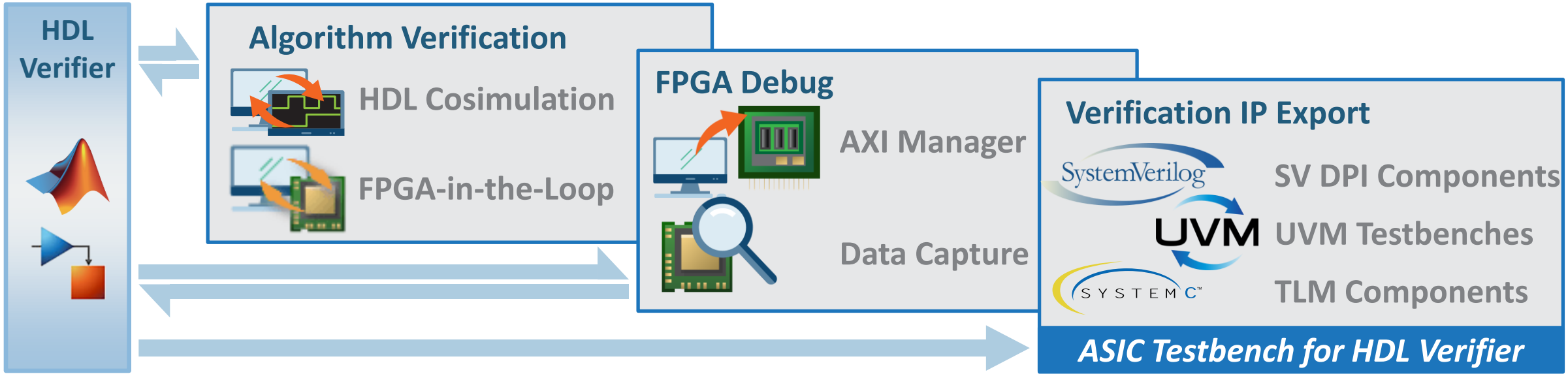
HDL Verifier Self-Guided Tutorial
R2022b

MathWorks Application Engineering

HDL Verifier

Test and verify Verilog and VHDL using HDL simulators and FPGA boards



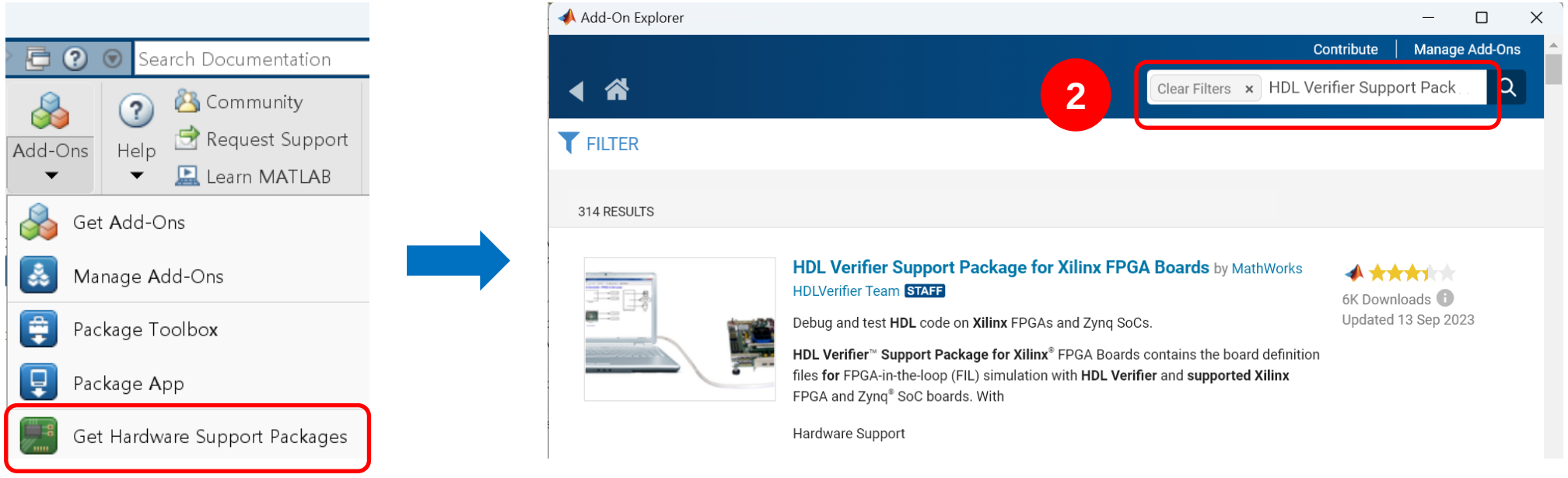
This self-guided tutorial covers the HDL Cosimulation, FPGA-in-the-Loop, and Data Capture features of HDL Verifier™
For other features, see the documentation here.

Hardware Support Package Installation

Before you can use the features in the HDL Verifier Support Package for Xilinx® FPGA Boards *, you must establish communication between the host computer and the hardware board.

Step1) On the MATLAB® **Home** tab, in the **Environment** section, click **Add-Ons > Get Hardware Support Packages**

Step2) In the right pane of the Add-On Explorer, find the HDL Verifier Support Package for AMD® FPGA Boards



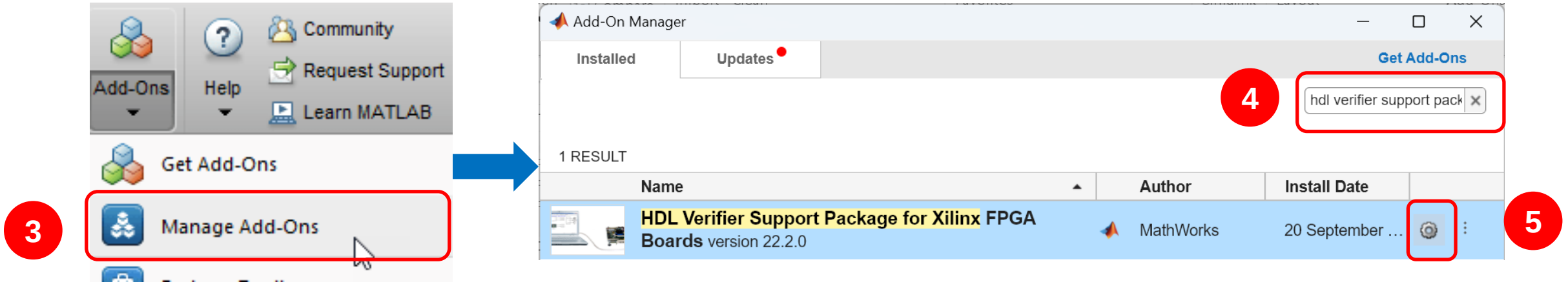
***** Note: in R2024b, this support package was renamed as **HDL Verifier Support Package for AMD® FPGA and SoC Devices.**

Guided Hardware Setup

Step3) After successful installation of the hardware support package, click **Add-Ons > Manage Add-Ons**

Step4) Find the installed hardware support package

Step5) Click the **Gear** icon to complete the hardware setup process



For more information, refer to the below link
https://www.mathworks.com/help/hdlverifier/ug/guided-hardware-setup.html

Prerequisite

Required MathWorks® Products

- MATLAB® (R2022b)
- Simulink®
- Fixed-Point Designer™
- MATLAB Coder™
- HDL Coder™
- HDL Verifier™
- Signal Processing Toolbox™
- DSP System Toolbox™
- DSP HDL Toolbox™

Background knowledge

- MATLAB and Simulink Onramp tutorials
- HDL Coder Self-Guided Tutorial
- Basic knowledge of HDL coding (VHDL or Verilog)

Required Tools

- MATLAB R2022b and other toolboxes (see Required MathWorks Products)
- HDL Verifier Support Package for Xilinx FPGA Boards (for FPGA-in-the-Loop and Data Capture)
- Supported EDA Tools (we will use Siemens® Questa™ or ModelSim™ for this demo)

Required Hardware

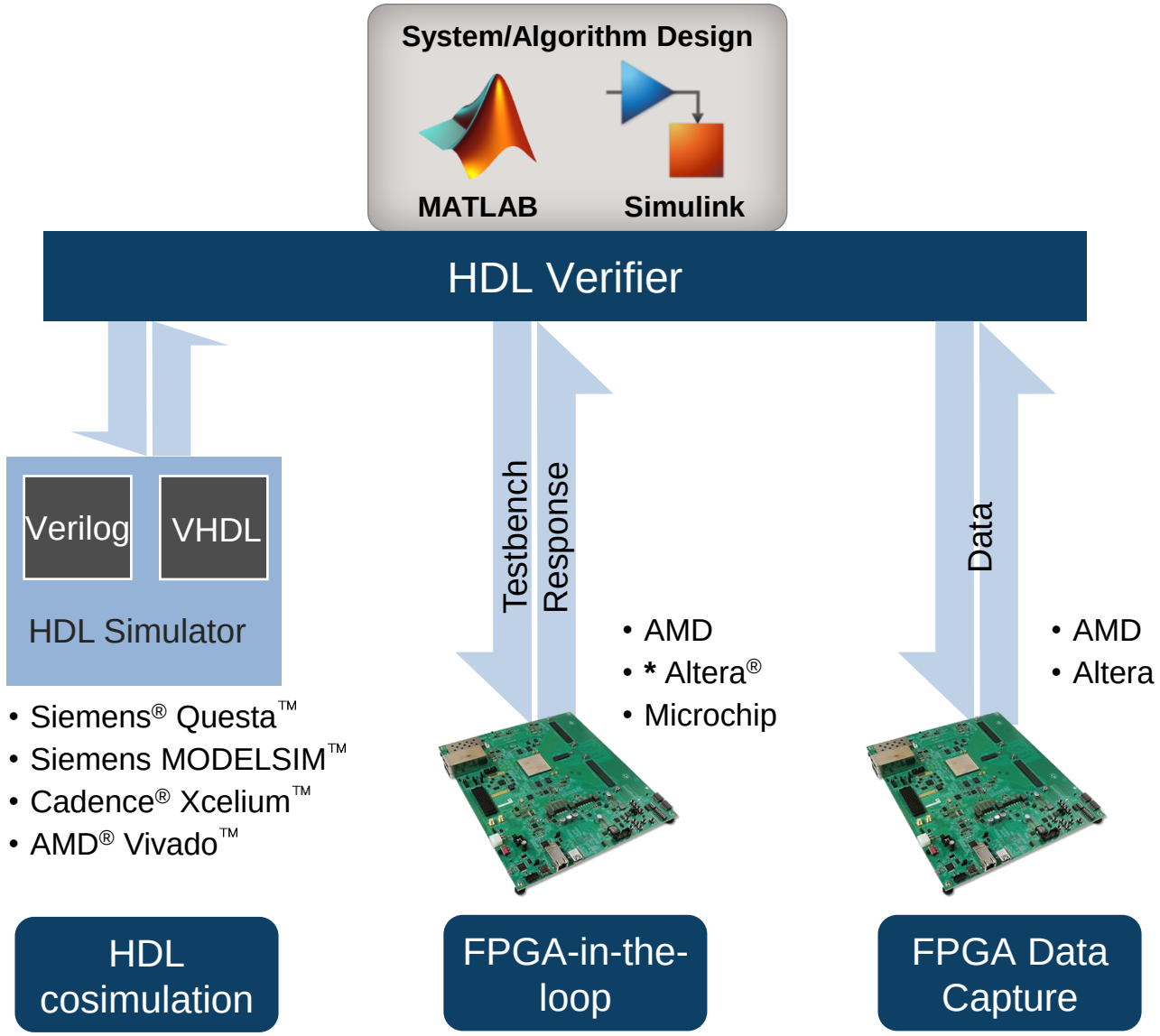
- ZedBoard™ (For other hardware, you can refer to Supported FPGA Devices for FPGA Verification)

Demo files

- Locate the example files in the folder: /HDLV_Tutorial

Agenda

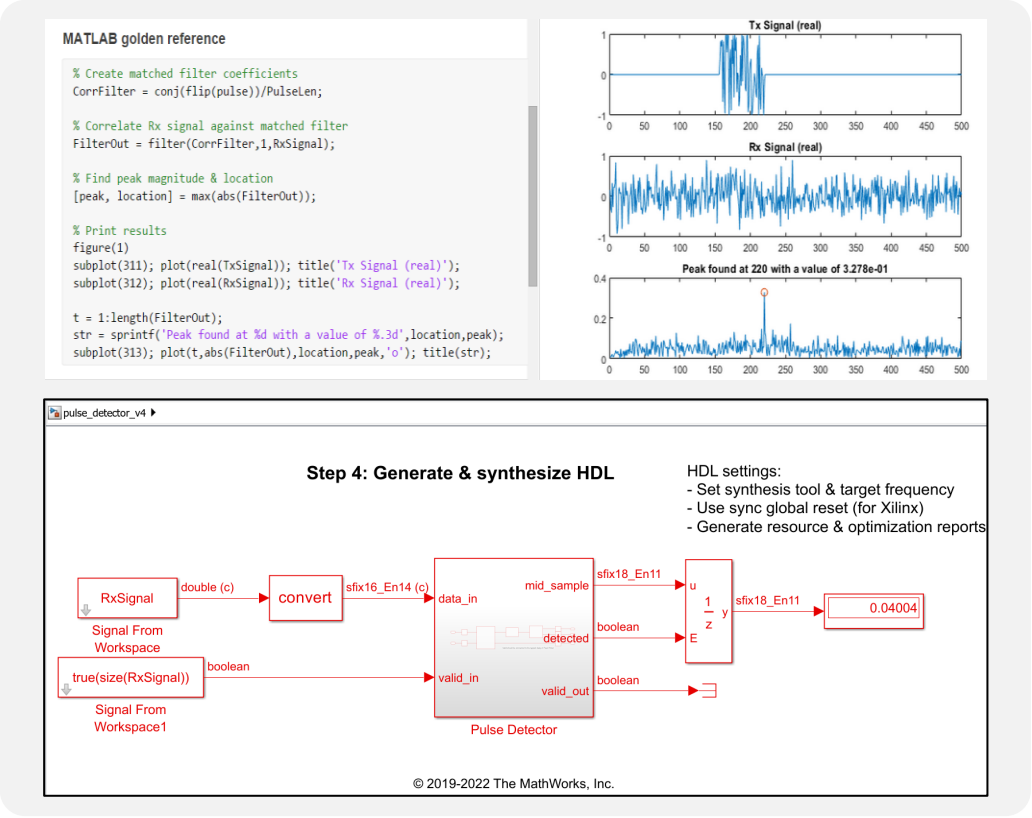
- Overview
- HDL Cosimulation
- FPGA-in-the-Loop
- FPGA Data Capture
- Additional Resources
- Conclusion

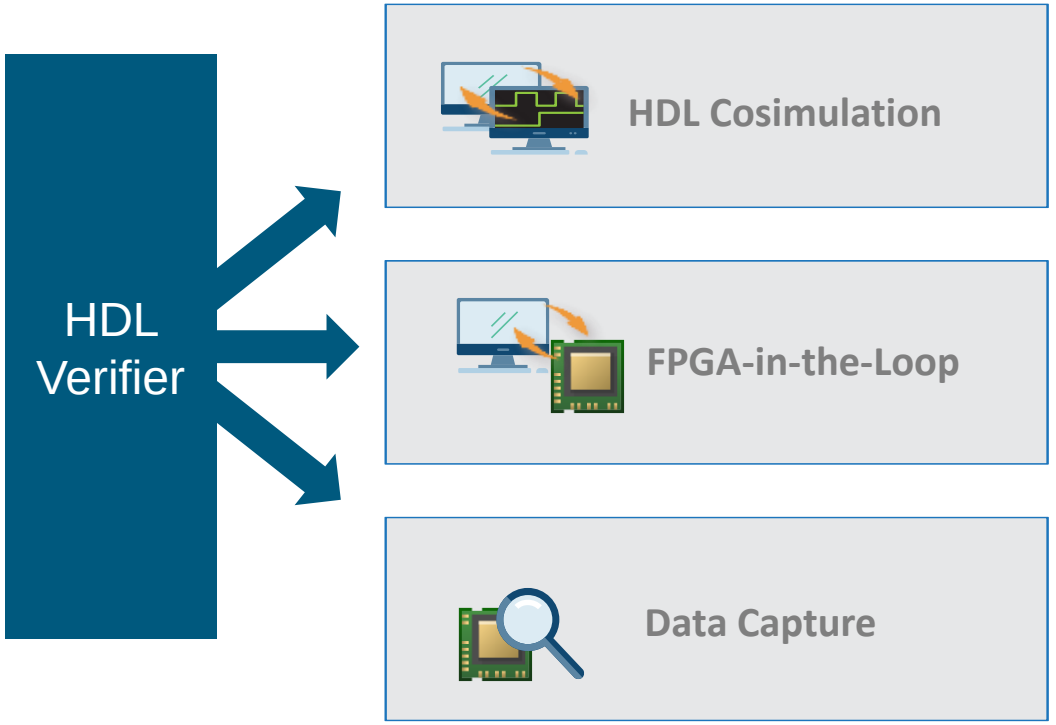


* Note: Altera became independent of Intel in 2025. All references to Intel products in this tutorial refer to Altera products.

Example Overview

- In this HDL Verifier Self-Guided Tutorial, you will learn about the different ways to verify the examples used in the HDL Coder Self-Guided Tutorial. It introduces the ability to verify automatically generated HDL code as well as HDL code that you write yourself.

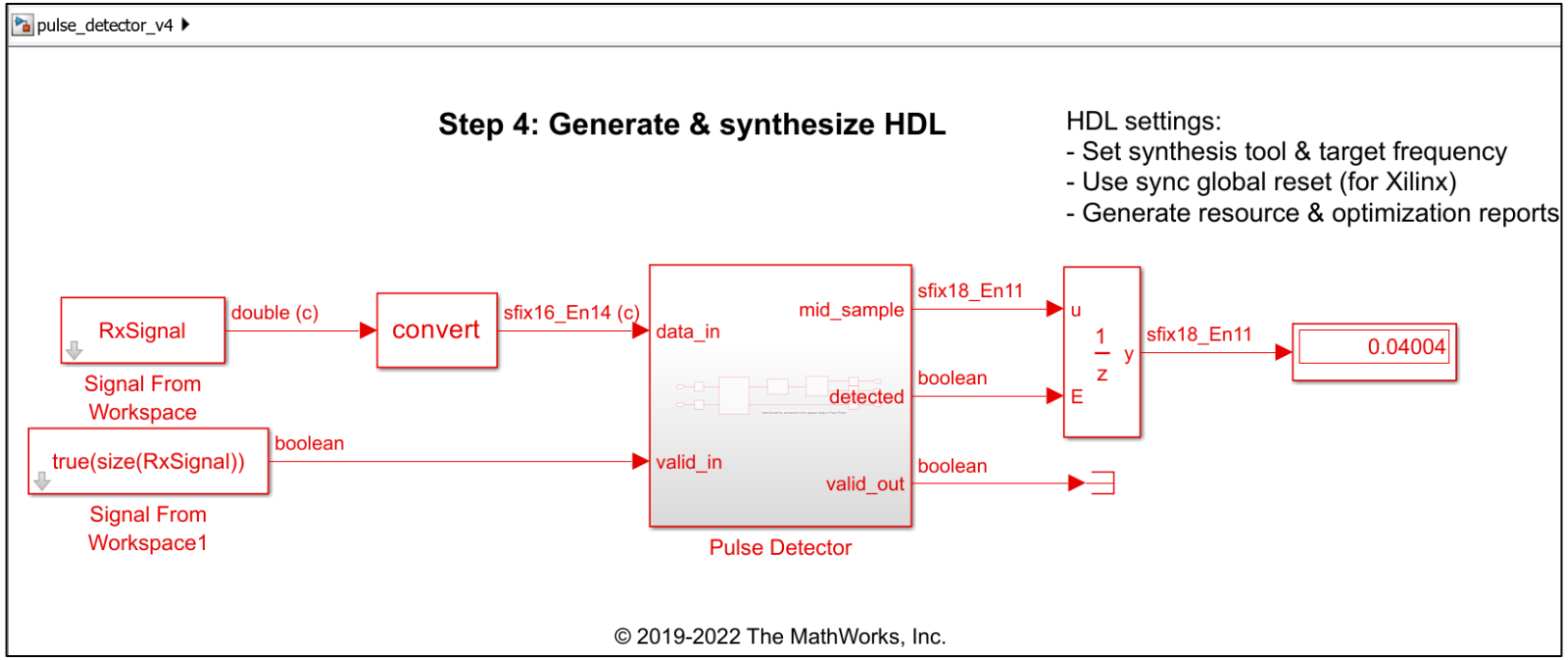


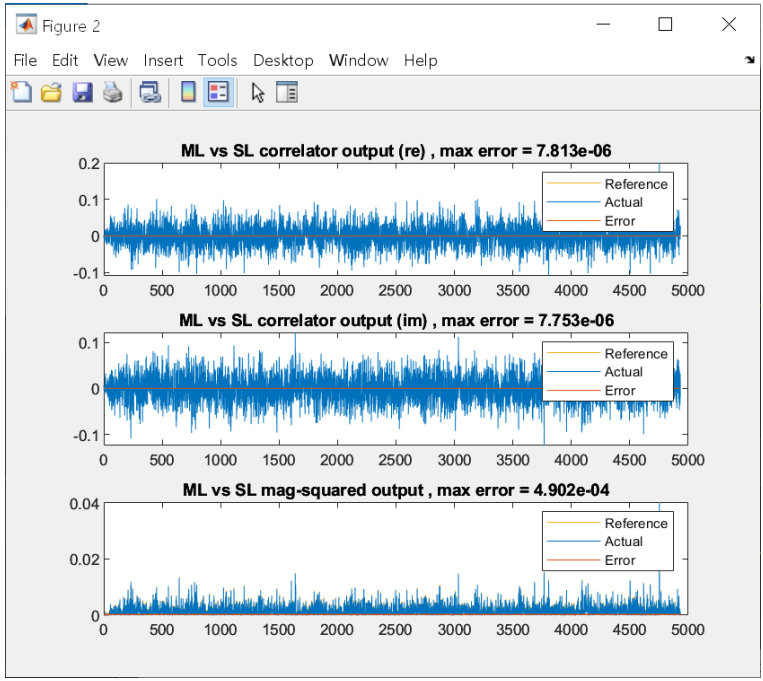


Example Setup

1. Open **pulse_detector_v4.slx** in solution folder
2. Run **pulse_detector_v4_tb.mlx** to simulate the model
3. Run hdlsetuptoolpath in MATLAB Command Window

   >> hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath','C:\Xilinx\Vivado\2020.2\bin\vivado.bat');

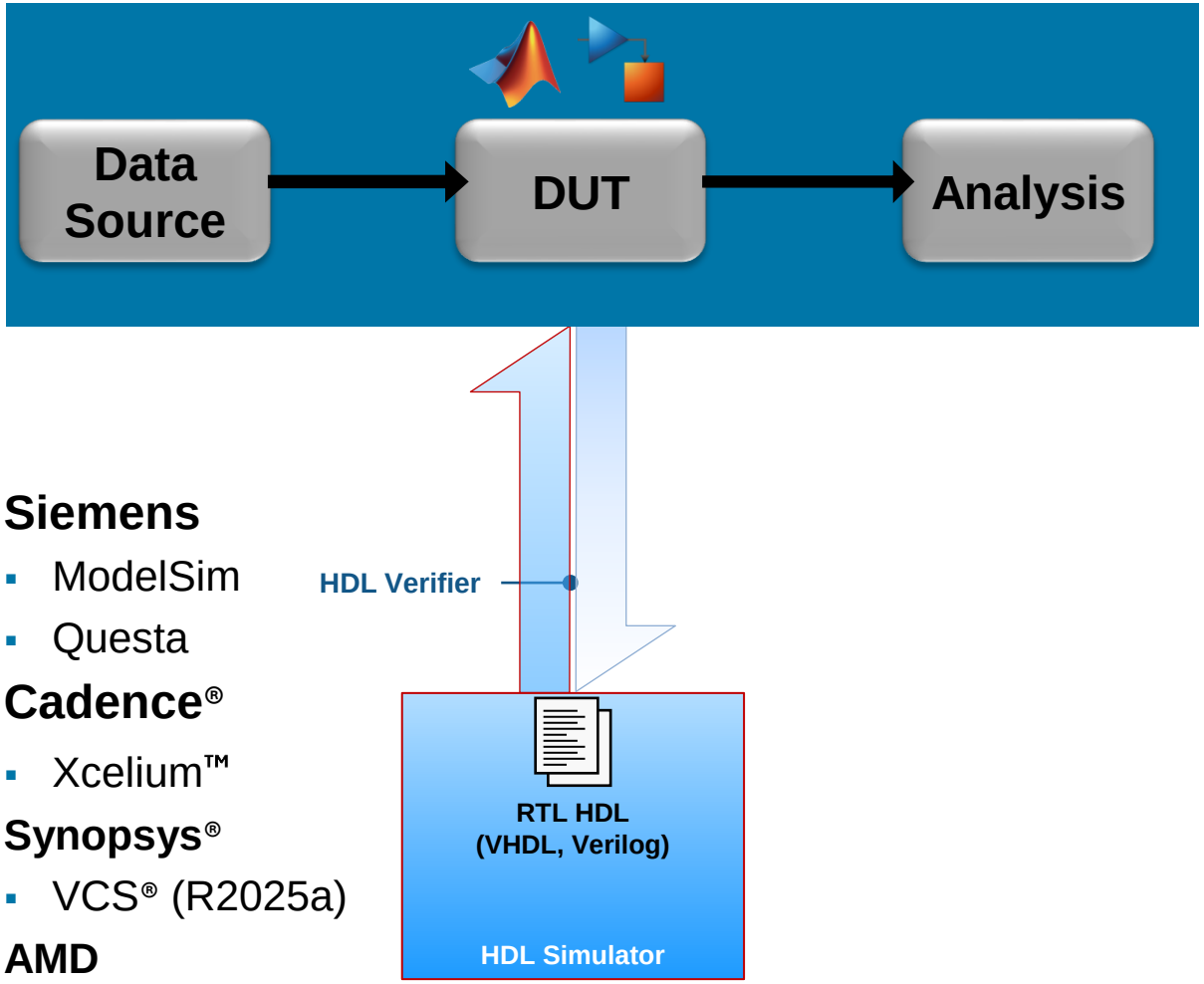




HDL Cosimulation

HDL Cosimulation

Connecting MATLAB/Simulink with HDL Simulators



- Use existing MATLAB / Simulink testbenches with HDL code execution in HDL simulator
- Verify handwritten HDL code, auto-generated code, or any combination
- Offers debugging and analysis in HDL simulator or MATLAB / Simulink
- Evaluate test benches for code coverage

Siemens
- ModelSim
- Questa

Cadence®
- Xcelium™

Synopsys®
- VCS® (R2025a)

AMD
- Vivado

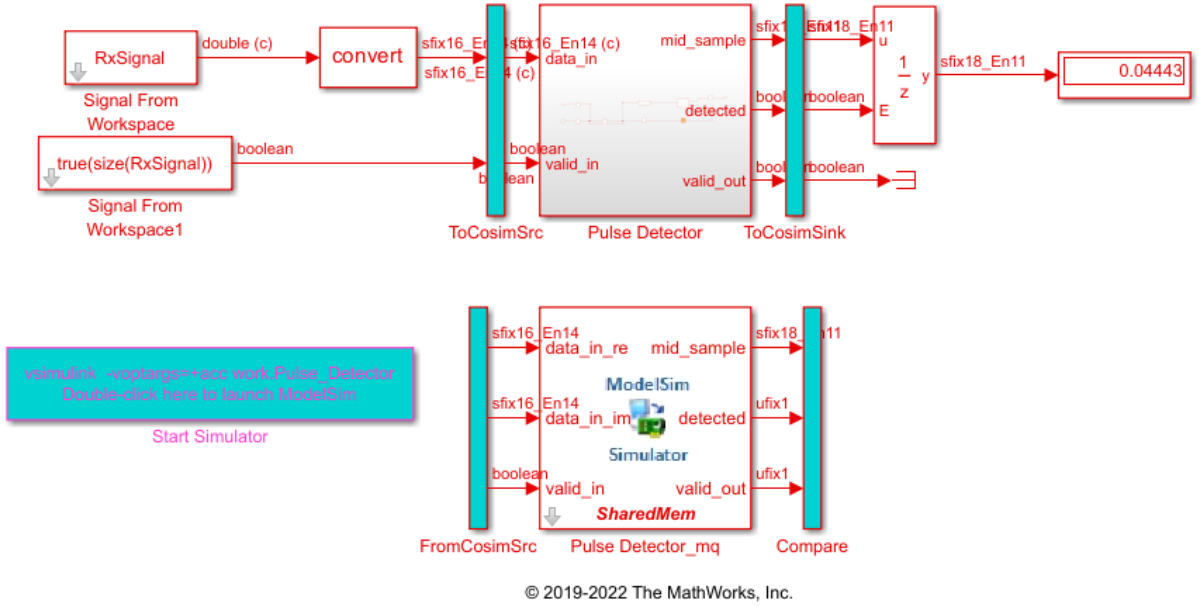
Listing of supported EDA tools

Demo 1: HDL Cosimulation with generated HDL Code

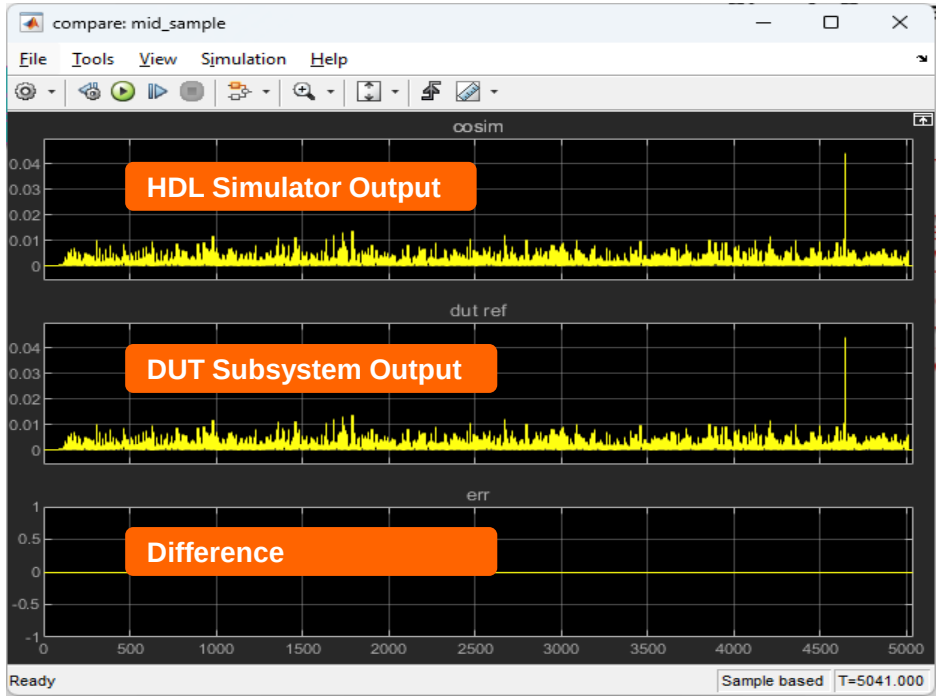
Demo 1 – Introduction!

In this demo, you will:

- Build **Cosimulation model** for generated HDL code
- Run and check the equivalence between original model and the HDL Simulator result



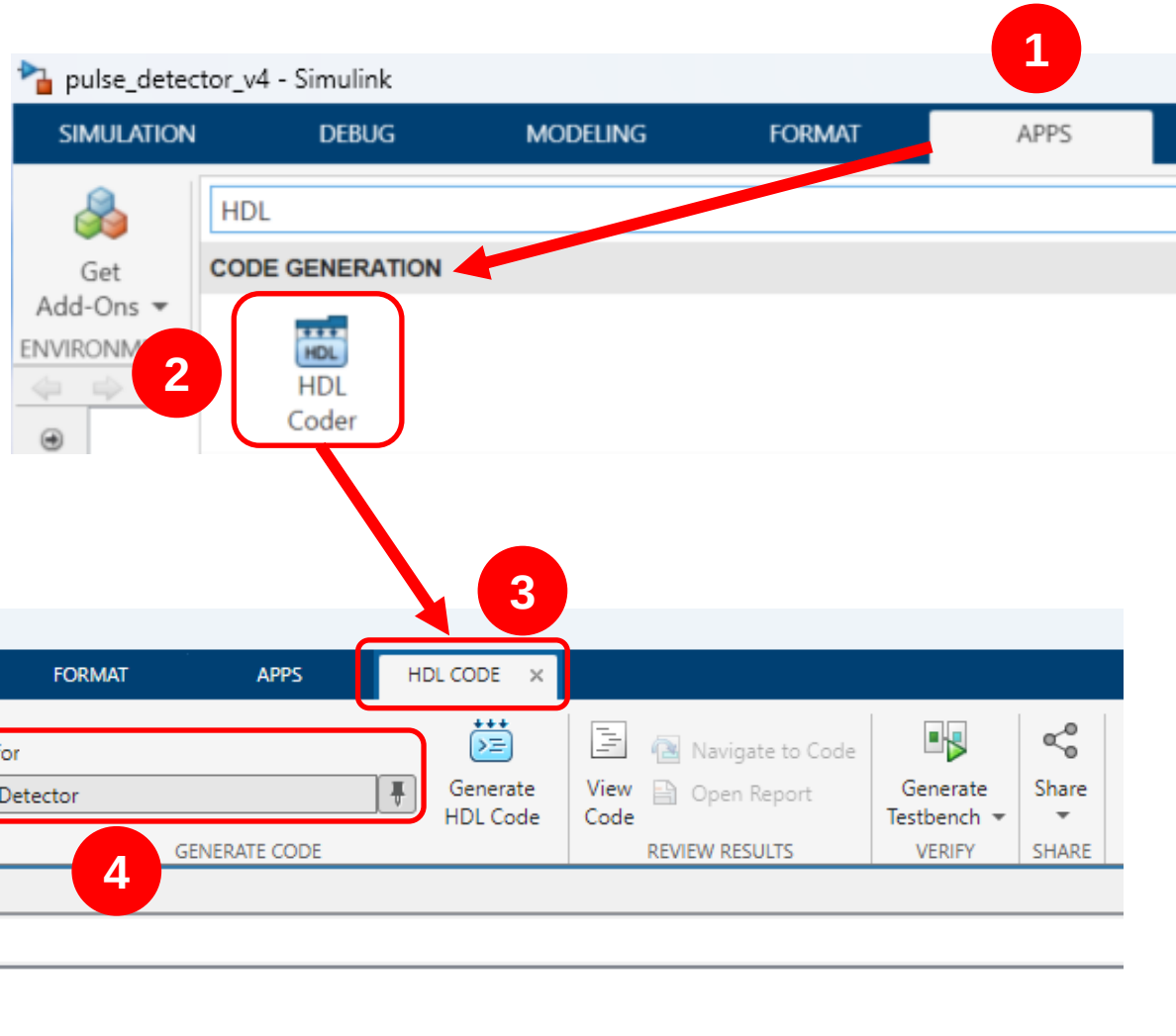
Cosimulation Model



HDL Cosimulation Result

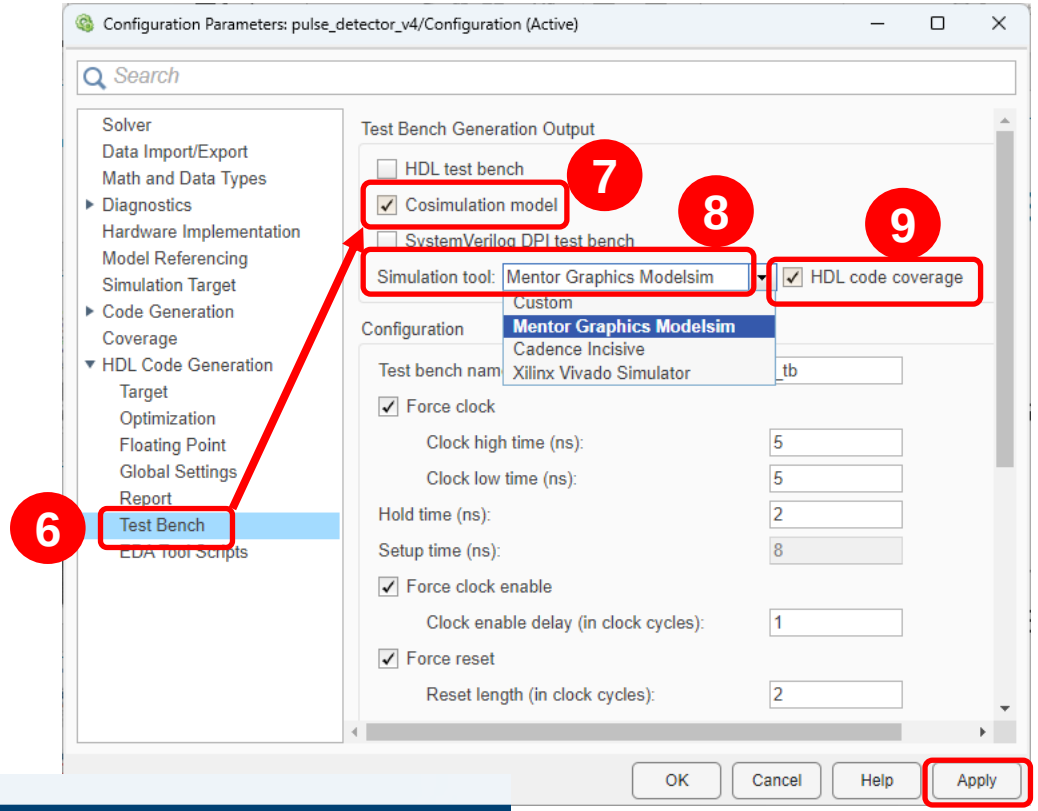
Demo 1: HDL Cosimulation with Generated HDL Code

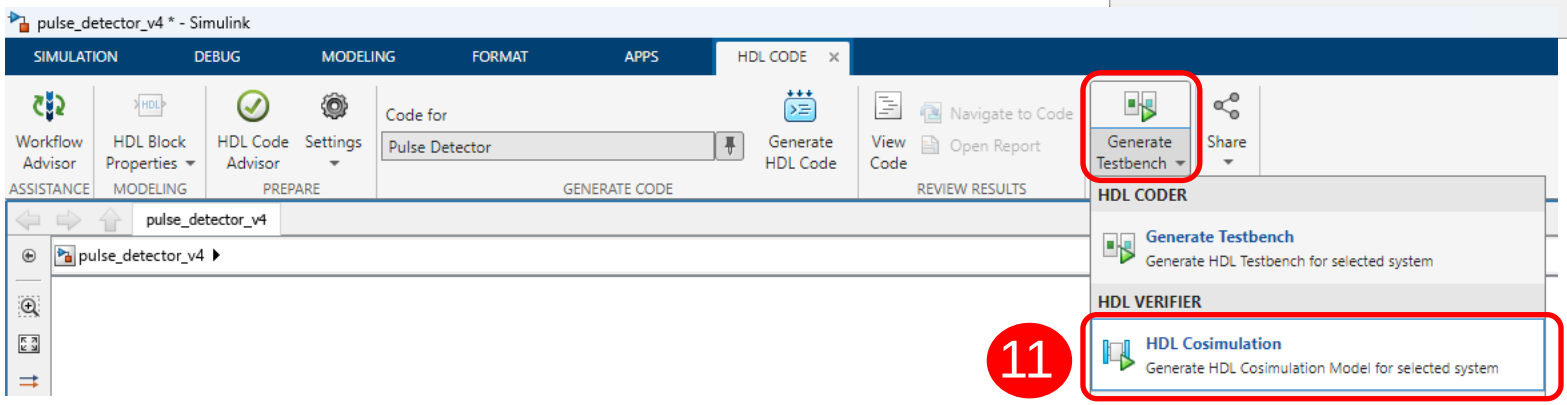
1. In the model **pulse_detector_v4.slx,** click **APPS**
2. Select **HDL Coder** under **CODE GENERATION** in the APP Gallery
3. **HDL CODE** toolstrip will appear
4. Set the Pulse Detector subsystem as DUT (HDL Code Toolstrip > unpin > click on subsystem > pin)
5. Click on **Settings** to choose HDL Simulator tool



Demo1: HDL Cosimulation with Generated HDL Code

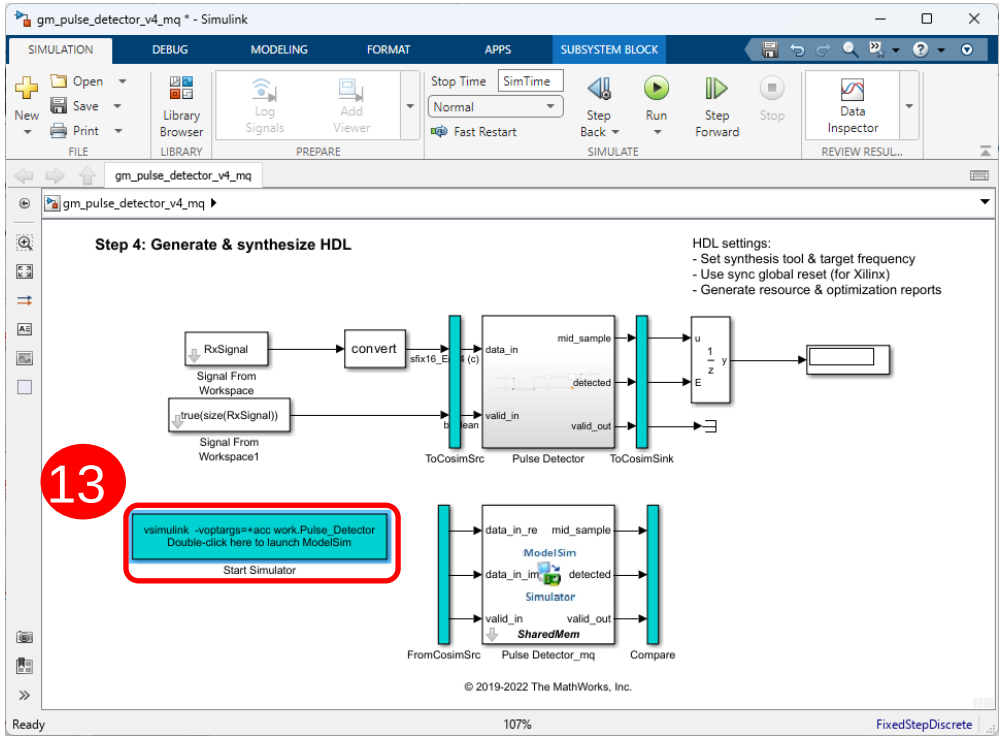
6. Click on **Test Bench** in the Configuration window
7. Check **Cosimulation model** under Test Bench Generation Output section
8. Select **Simulation tool** as ‘Mentor Graphics Modelsim’ from the drop-down menu for co-simulating with Questa or MODELSIM Simulators
9. Check **HDL code coverage** option to display code coverage results in Simulator post Cosimulation
10. Click **Apply** and **Close** the Configuration window
11. In HDL CODE toolstrip, Click drop-down of **Generate Testbench,** Select **HDL Cosimulation** to generate Cosimulation model and the required EDA scripts





Demo1: HDL Cosimulation with generated HDL Code

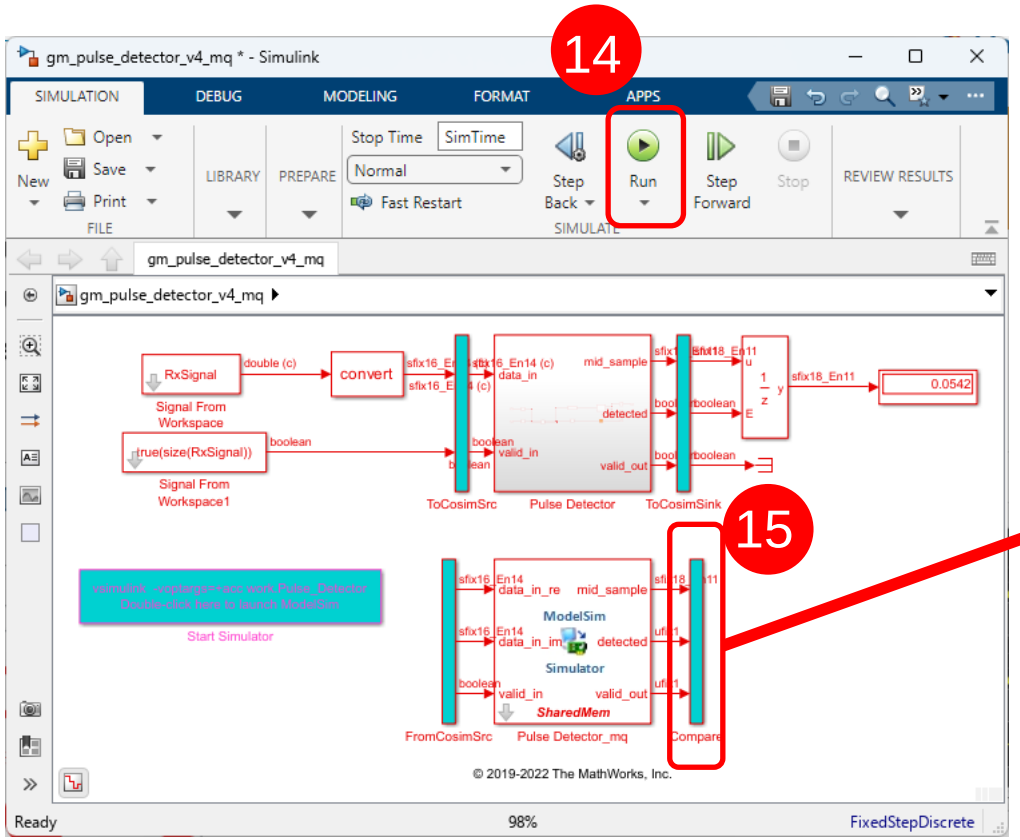
12. A new Simulink model '**gm_pulse_detector_v4_mq**' gets created which has original model along with Cosimulation block
13. Double-click on **Start Simulator** which then opens up the Questa simulator, compiles generated HDL code, creates waveform window and keeps it Ready for Cosimulation

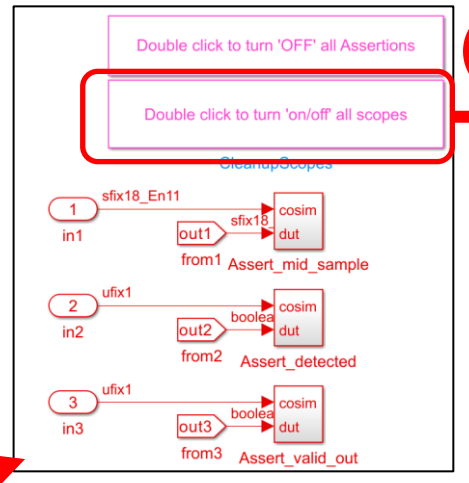


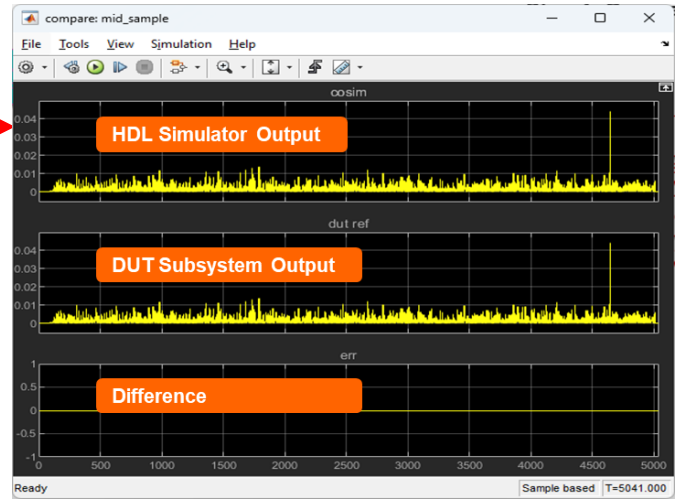
Troubleshoot Guide

Demo1: HDL Cosimulation with Generated HDL Code

14. Click **Run**  button to run the simulation
15. Click **Compare** block to see the equivalence check module
16. Double Click the button below to turn on all scopes







Demo 2: HDL Cosimulation with Handwritten HDL Code

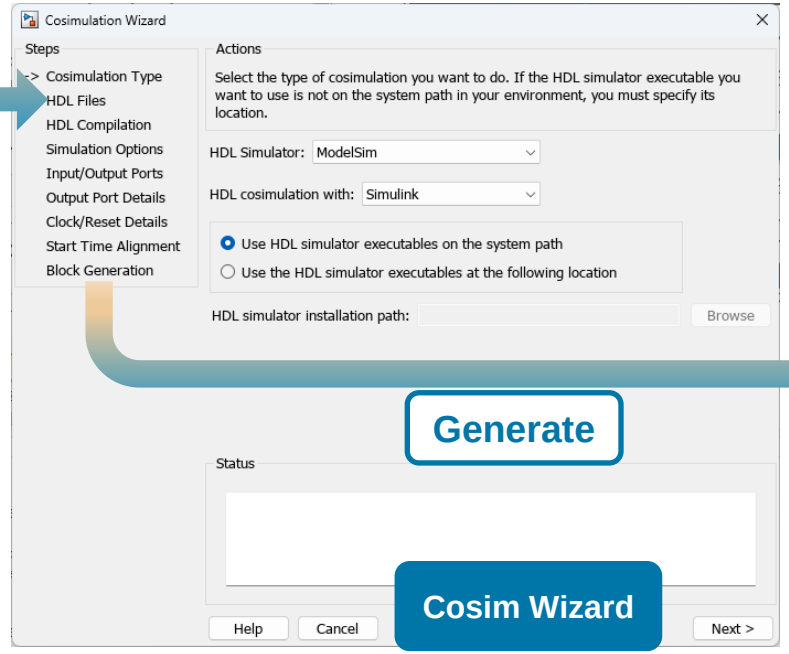
Demo 2: Introduction!

- Bring your handwritten HDL code as a Cosimulation block into Simulink using CosimWizard
- Use Simulink as the testbench to simulate your handwritten HDL code

handwritten HDL Code

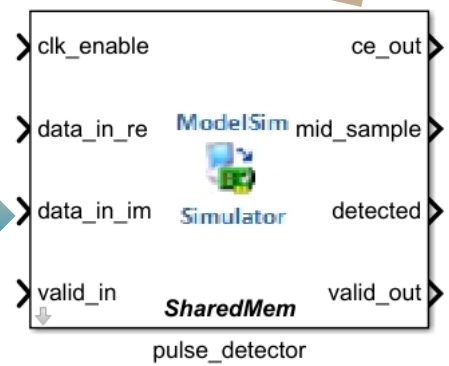
```
ENTITY Pulse_Detector IS
  PORT( clk           :   IN    std_logic;
        reset         :   IN    std_logic;
        clk_enable    :   IN    std_logic;
        data_in_re    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        data_in_im    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        valid_in      :   IN    std_logic;
        ce_out        :   OUT   std_logic;
        mid_sample    :   OUT   std_logic_vector(17 DOWNTO 0);  -- sfix18_En11
        detected      :   OUT   std_logic;
        valid_out     :   OUT   std_logic
        );
END Pulse_Detector;
```

Import



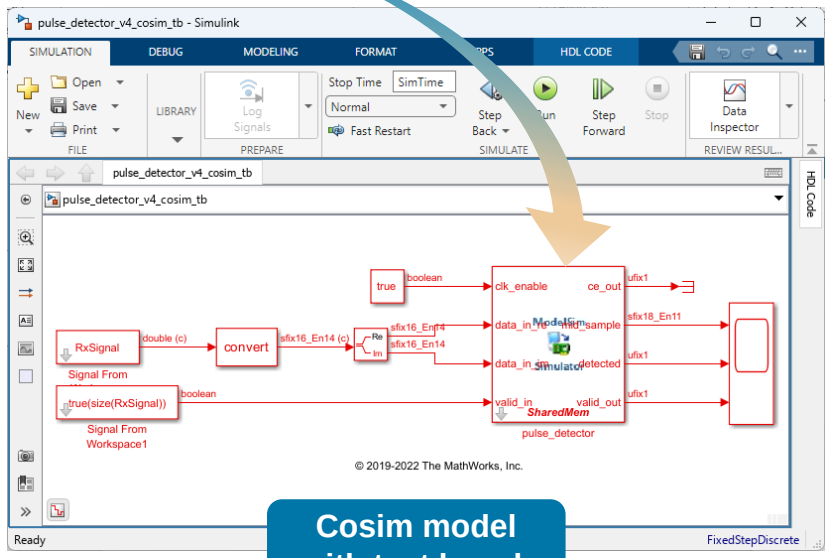
Generate

Cosim Wizard



Cosim Block

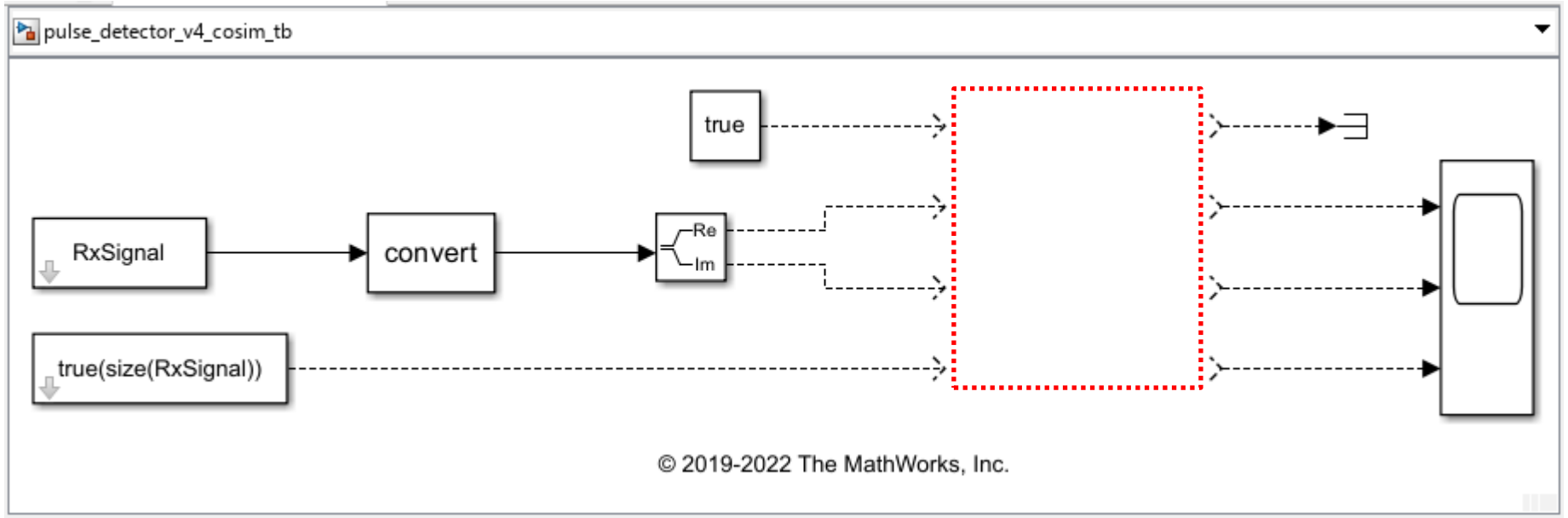
Test with Cosim Block



Cosim model with test bench

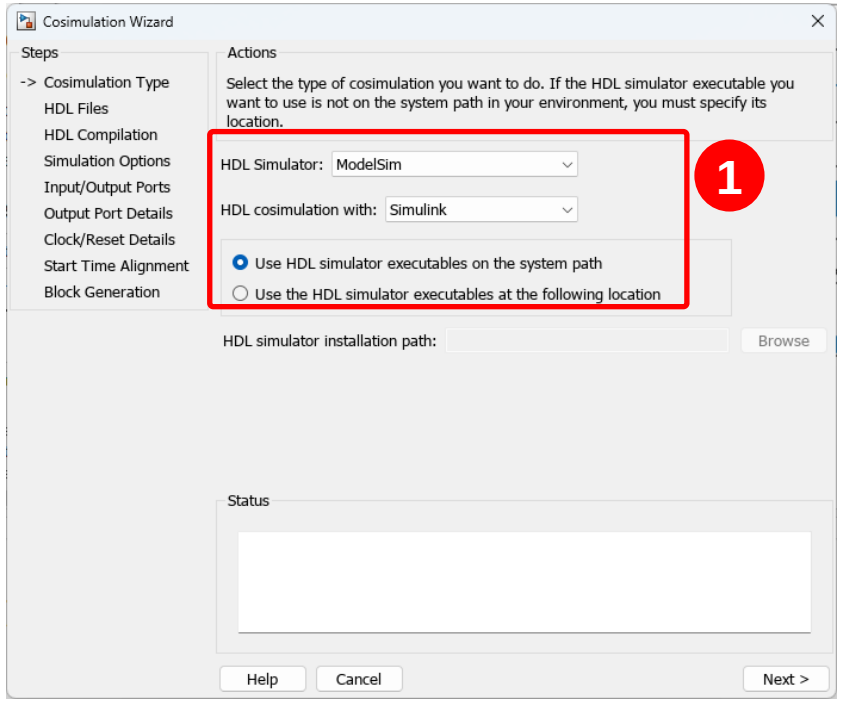
Initial Setup

1. Open **pulse_detector_v4.slx** model and run **pulse_detector_v4_tb.mlx** to load variables to workspace
2. Open **pulse_detector_v4_cosim_tb.slx** model in CosimFiles. This is the testbench provided.
3. HDL files are available in <*CosimFiles/CosimHDLFiles*> folder. A Cosim block is created for these files and is integrated into the test bench.



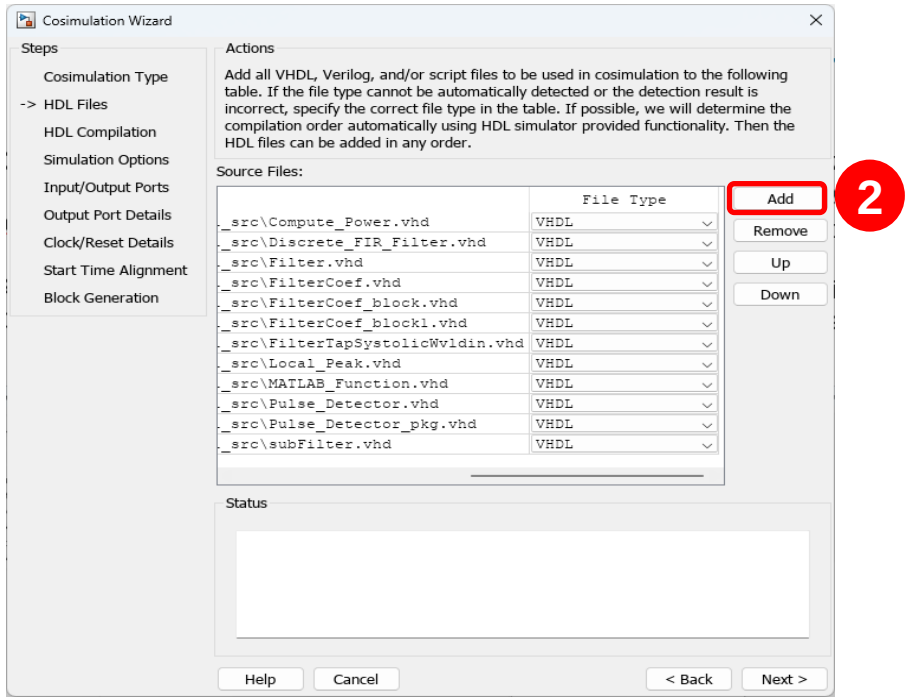
Demo 2: HDL Cosimulation with handwritten HDL Code

1. Launch **Cosimulation Wizard** by executing the following command in MATLAB command window. ***>> cosimWizard***
   In the **cosimWizard**, set each parameters as below and click **Next**
   - HDL Simulator: ModelSim
   - HDL Cosimulation with: Simulink
   - Use HDL simulator executables on the system path



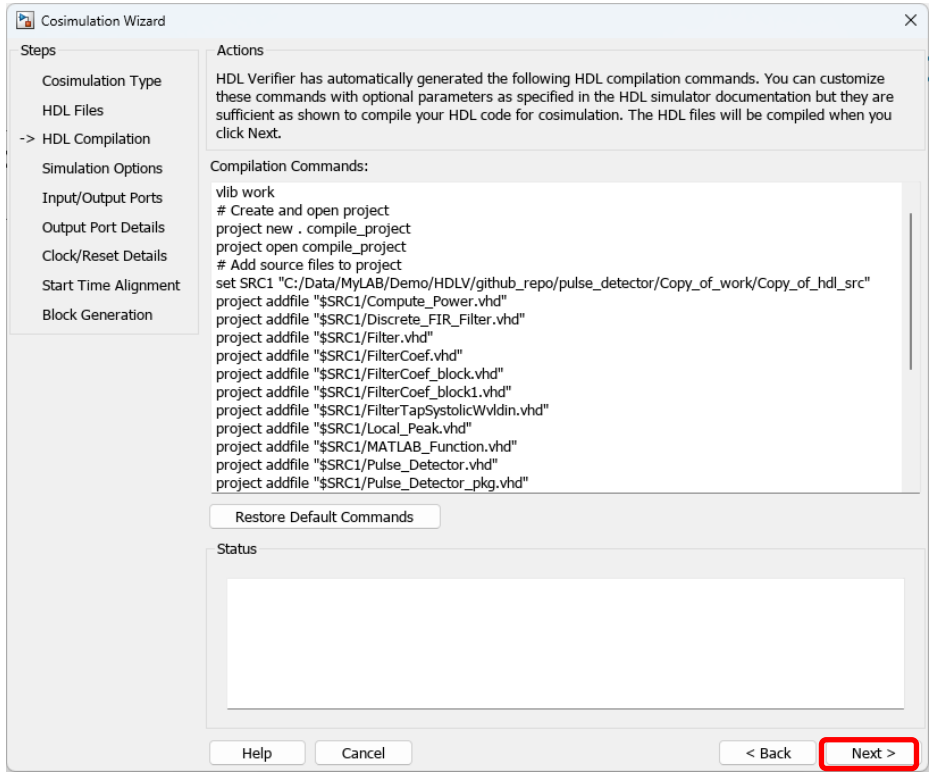
2. Click **Add** to specify the source files for the HDL design

-The source HDL files for the current demo are available in the *<CosimFiles/CosimHDLFiles>* folder

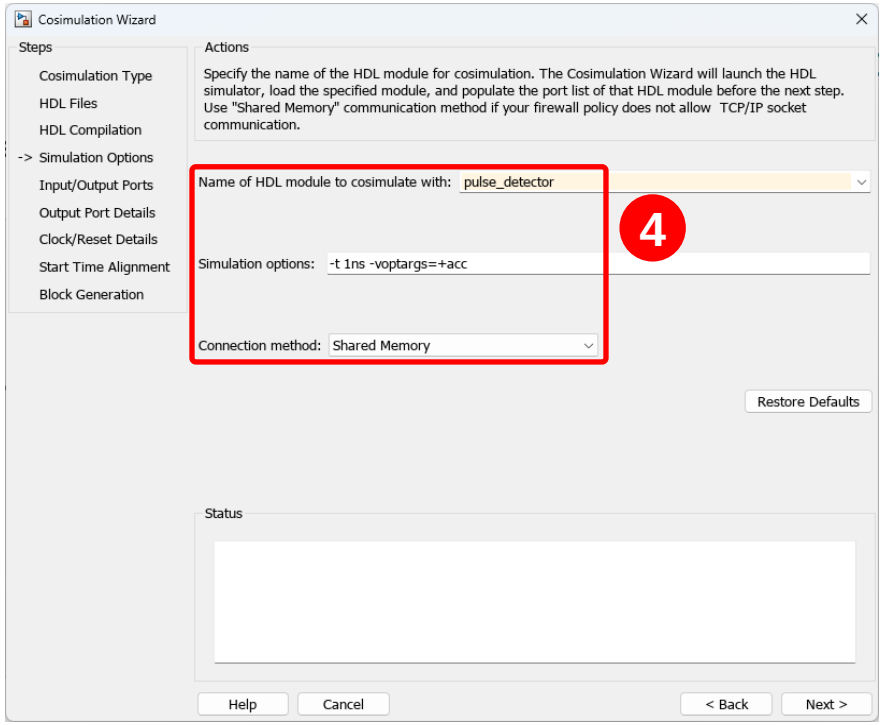


Demo 2: HDL Cosimulation with handwritten HDL Code

3. Click **Next** to compile the HDL design using automatically generated compilation commands. The command window shows the status of compilation.



4. In Simulation Options step, set the parameters as below and click **Next** to load Simulation and HDL Verifier Libraries
   - Name of HDL module to cosimulate with: pulse_detector (This is the top module for Cosimulation, select from drop down list)
   - Connection method: Shared Memory (when HDL Simulator is on the same computer as MATLAB)

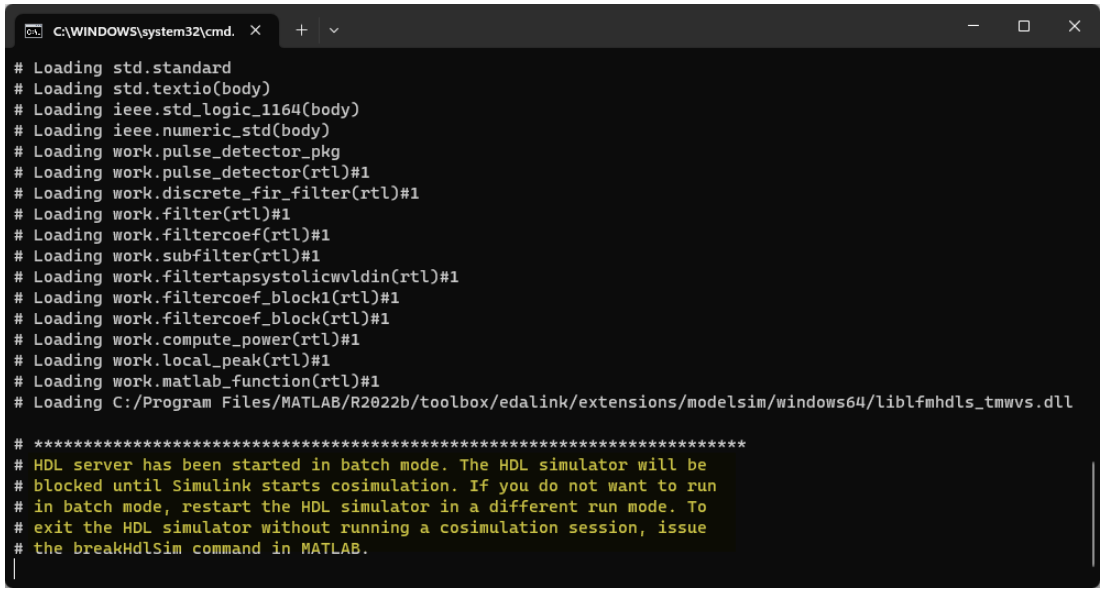


Demo 2: HDL Cosimulation with handwritten HDL Code

5. After clicking on **Next** in the Simulation Step, a terminal window opens for ‘Loading Simulation and HDL Verifier Libraries’. Wait till the terminal window shows message as in the picture.

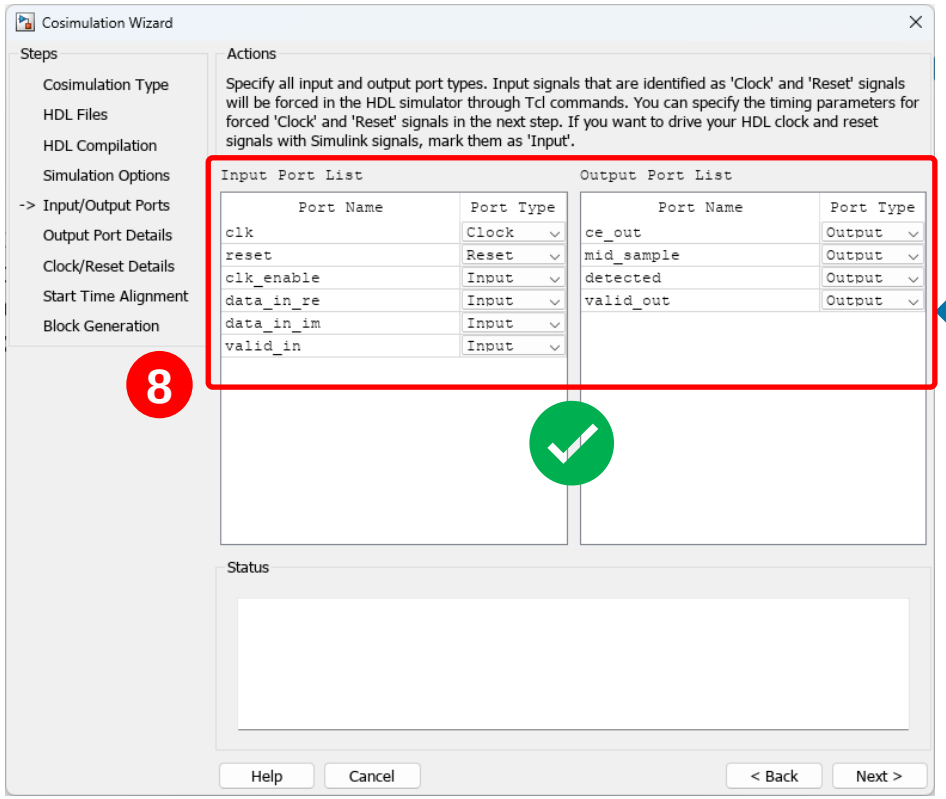
6. At this stage, HDL Simulator has been invoked by MATLAB in batch mode and blocked until Cosimulation starts.

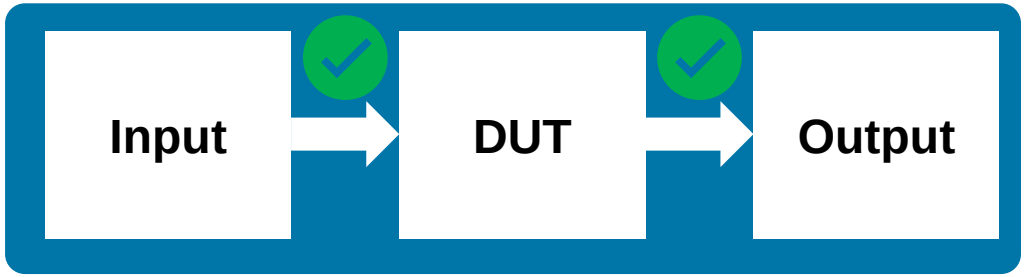
7. To run the HDL Simulator in GUI mode, execute the following command in MATLAB command window to release it from batch mode: ***>> breakHdlSim***



Demo 2: HDL Cosimulation with handwritten HDL Code

8. Review the DUT Input and Output ports of Top-level File and check if is assigned as intended. If not, change the Port Type accordingly.
Click **Next**.





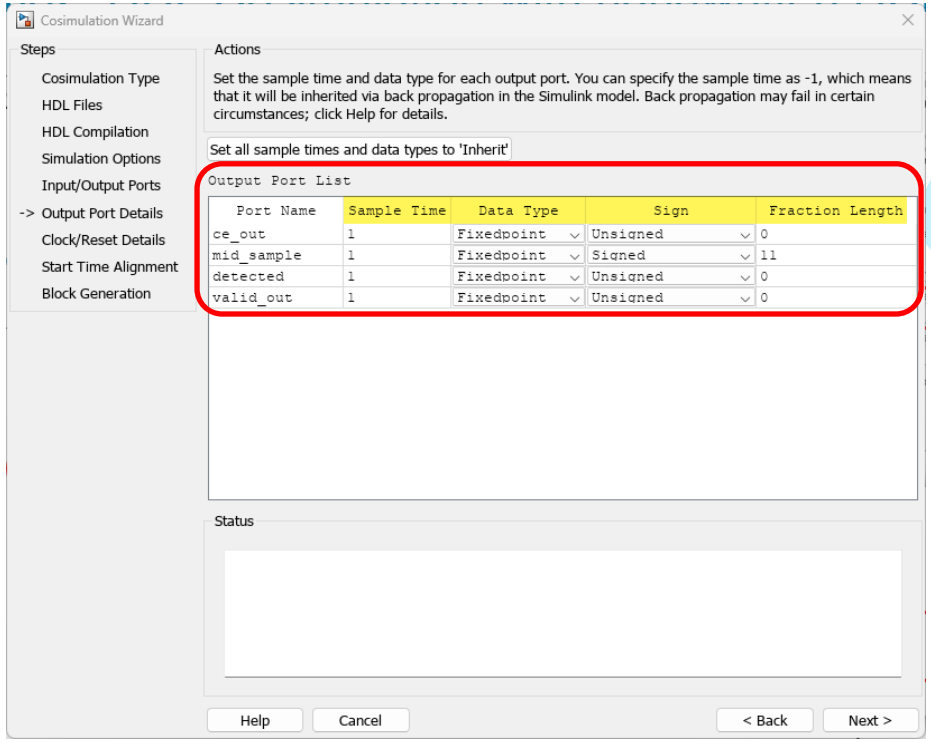

```
ENTITY Pulse_Detector IS
  PORT( clk           :   IN    std_logic;
        reset         :   IN    std_logic;
        clk_enable    :   IN    std_logic;
        data_in_re    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        data_in_im    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        valid_in      :   IN    std_logic;
        ce_out        :   OUT   std_logic;
        mid_sample    :   OUT   std_logic_vector(17 DOWNTO 0);  -- sfix18_En11
        detected      :   OUT   std_logic;
        valid_out     :   OUT   std_logic
        );
END Pulse_Detector;
```

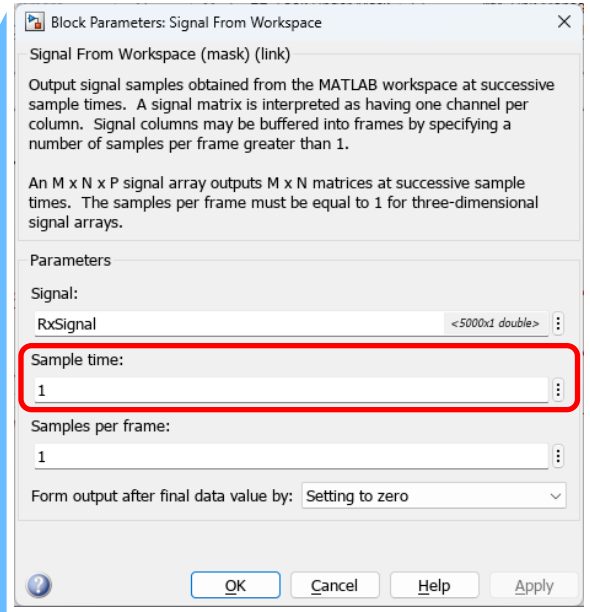
<Top module File: Pulse_Detector.vhd>

Demo 2: HDL Cosimulation with handwritten HDL Code

9. Review the Sample Time and Data Types of the Output Ports and set them accordingly.
   - The Sample Time of the output ports is set based on the sample time in the test bench model. In this case, it is 1 s as set in the ‘Signal From Workspace’ block.
   - The Data Types of the output ports can be matched against the top HDL module file

9



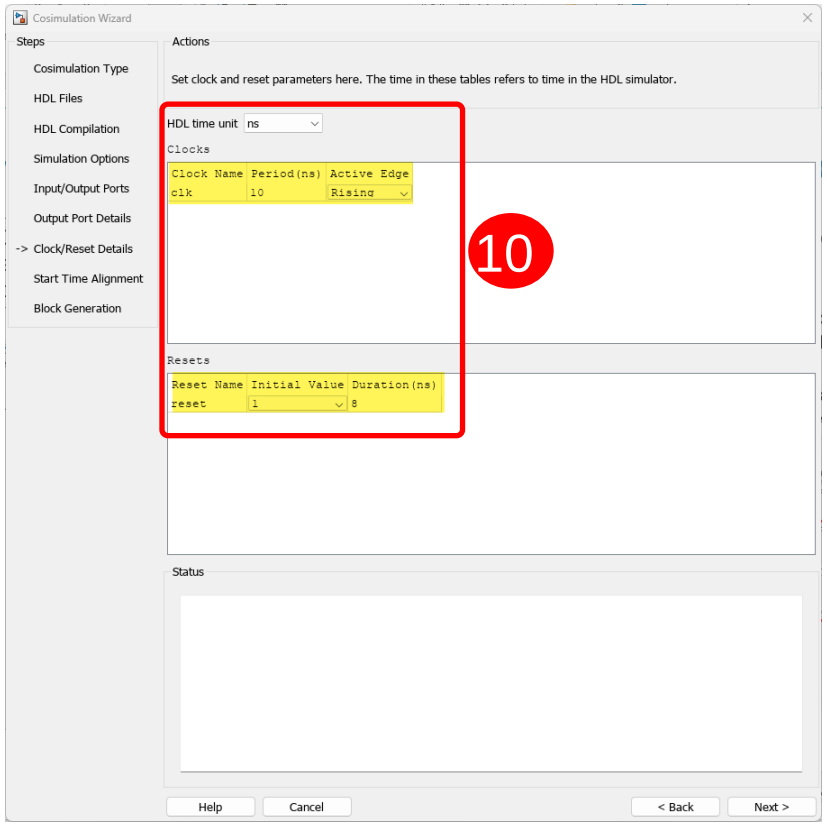



```
ENTITY Pulse_Detector IS
  PORT( clk           :   IN    std_logic;
        reset         :   IN    std_logic;
        clk_enable    :   IN    std_logic;
        data_in_re    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        data_in_im    :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        valid_in      :   IN    std_logic;
        ce_out        :   OUT   std_logic;
        mid_sample    :   OUT   std_logic_vector(17 DOWNTO 0);  -- sfix18_En11
        detected      :   OUT   std_logic;
        valid_out     :   OUT   std_logic
        );
END Pulse_Detector;
```

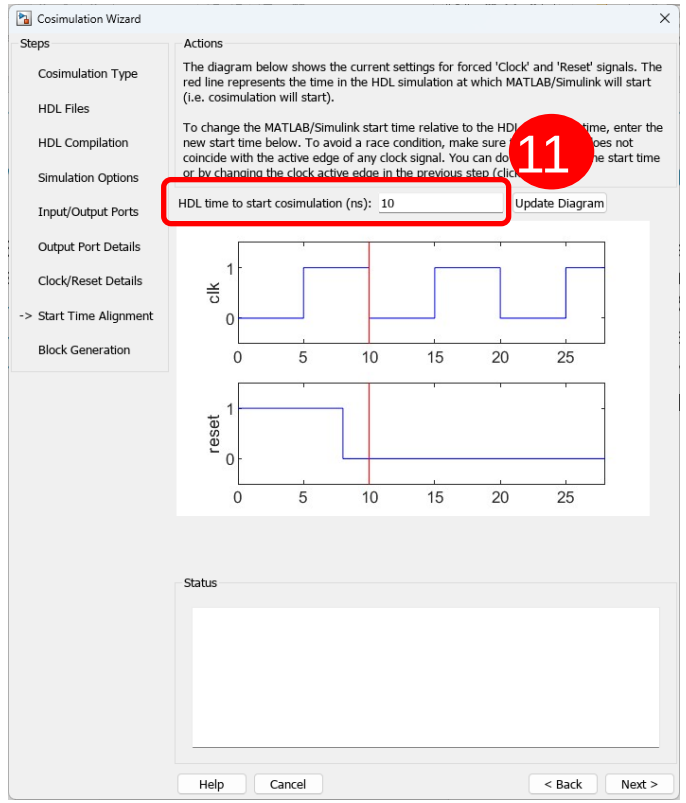
<Top module File: Pulse_Detector.vhd>

Demo 2: HDL Cosimulation with handwritten HDL Code

10. Set HDL time unit, Clock and Reset parameters as shown below and click **Next.**
    - HDL time unit: ns
    - Clock – Name: clk; Period: 10; Active Edge: Rising
    - Reset – Name: reset; Initial Value: 1; Duration: 8

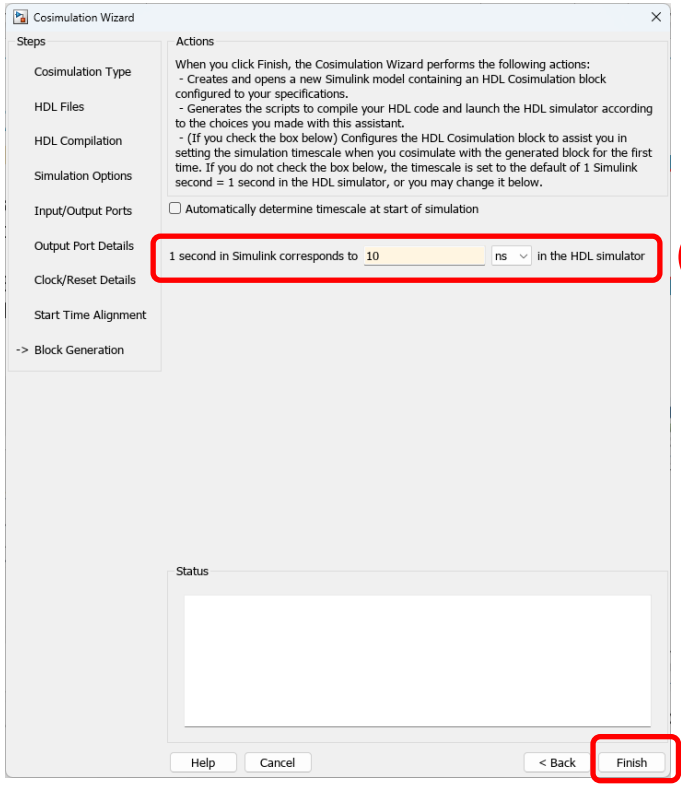


11. In Start Time Alignment step, choose when to start the Cosimulation and click **Next**. This is the point from when Simulink and HDL Simulator starts exchanging the data
    - Set the HDL time to start Cosimulation to 10 ns
    - The window also shows the timing diagram representing Clock, Reset and Cosim start timings

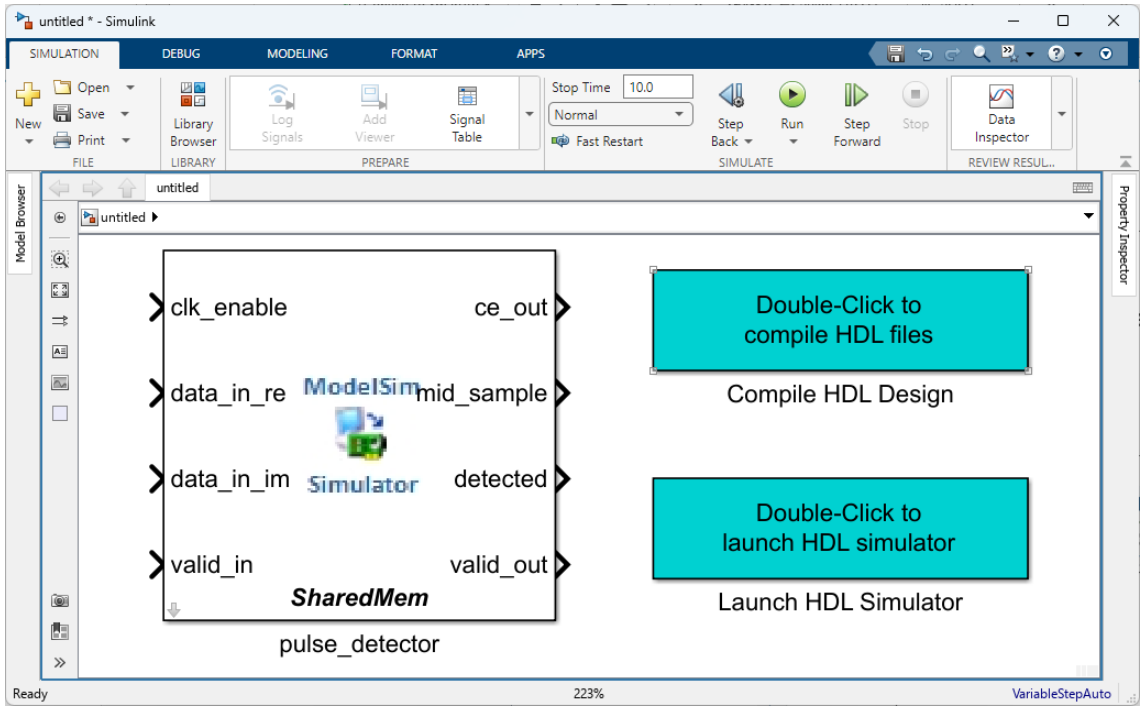


Demo 2: HDL Cosimulation with handwritten HDL Code

12. In the Block Generation step, define the relation between Simulink step time and HDL Clock configured. This ensures proper exchange of data.
    - The clock period configured at step 10 is 10 ns
    - The highest sample time in the Simulink model is 1 s
    - So, Timescale is 1 s (Simulink) = 10 ns (HDL simulator)



13. Click Finish and this generates a new Simulink model with Cosimulation block and Callback Functions to compile HDL files and start the HDL Simulator.

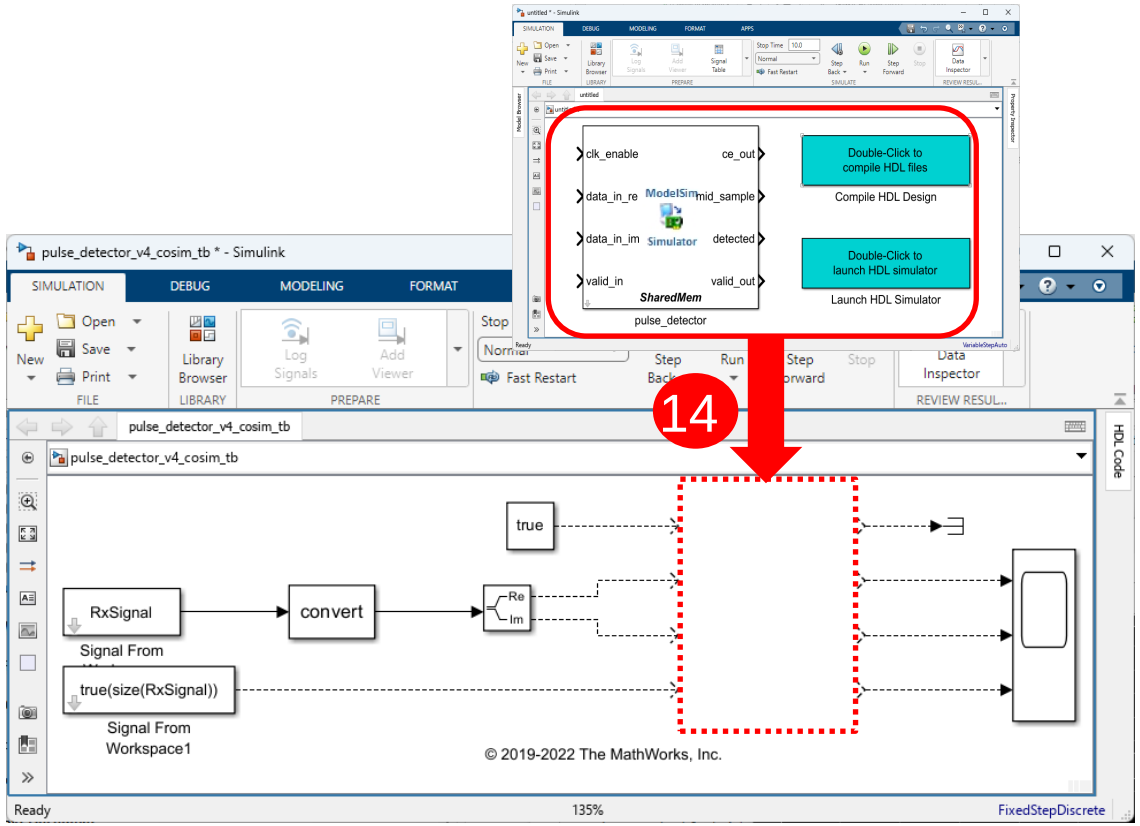


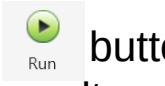
<Cosimulation Block and Callback Functions>

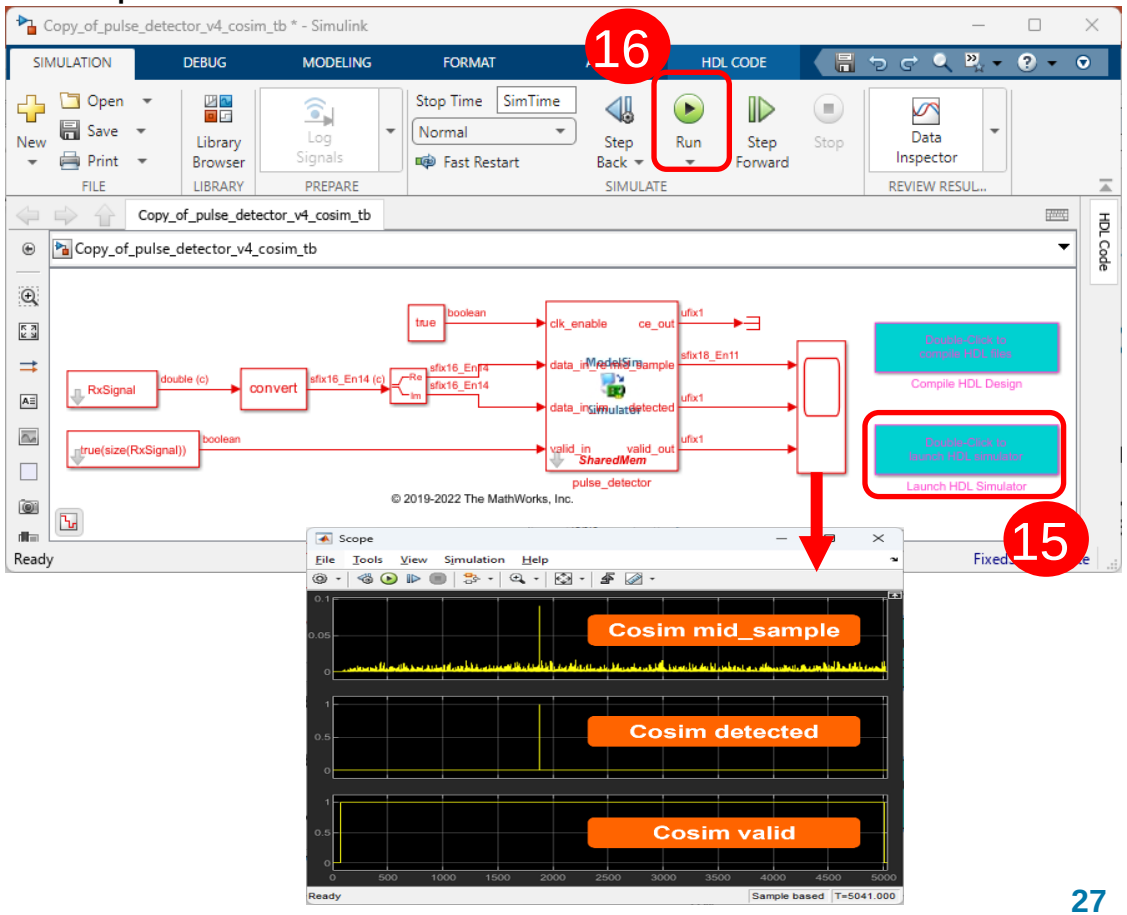
For more details, check timescale documentation

Demo 2: HDL Cosimulation with handwritten HDL Code

14. Copy all the contents from generated model to provided **pulse_detector_v4_cosim_tb.slx** and connect input/output ports with matched signals as shown in the picture.



15. Double-Click on Launch HDL Simulator to open the simulator in GUI mode
16. Click Run  button to run the Cosimulation and check scope for results



Conclusion: HDL Cosimulation with HDL Verifier

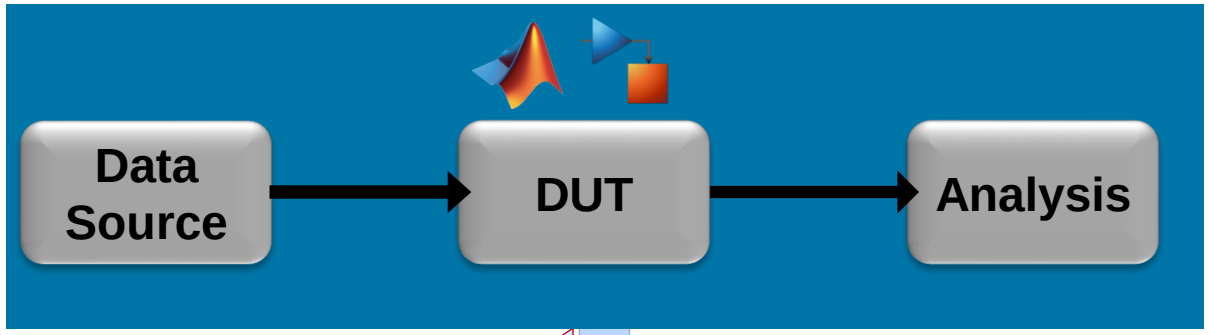
HDL Cosimulation creates a communication link between the HDL simulator and MATLAB or Simulink. Such a link enables you to:

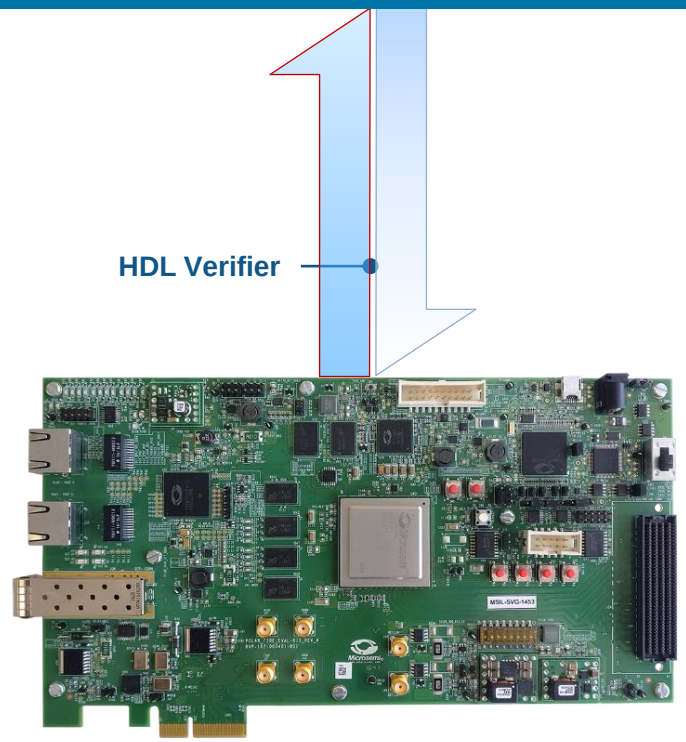
- Verify MATLAB code or a Simlulink model directly against the HDL implementation.
- Create test signals and testbenches for HDL code.
- Use a behavioral model as a reference in an HDL simulation.
- Use analysis and visualization features for real-time insight into an HDL implementation.
- Include existing HDL code within a Simulink model or MATLAB code

FPGA-in-the-Loop

FPGA-in-the-Loop (FIL)

Benefits of FIL

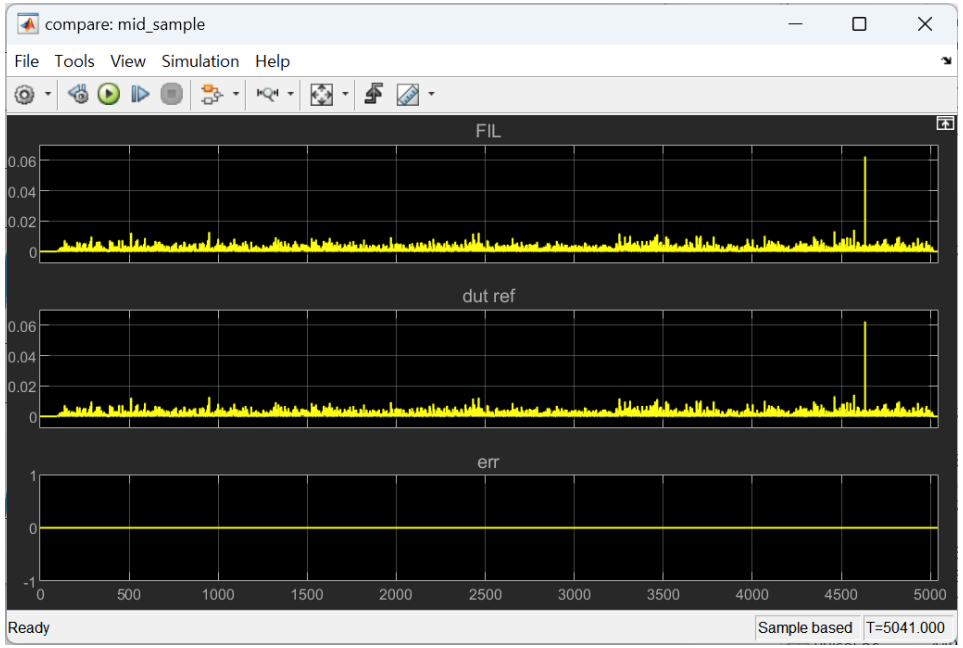




Boards with devices from:

- AMD
- Altera
- Microchip

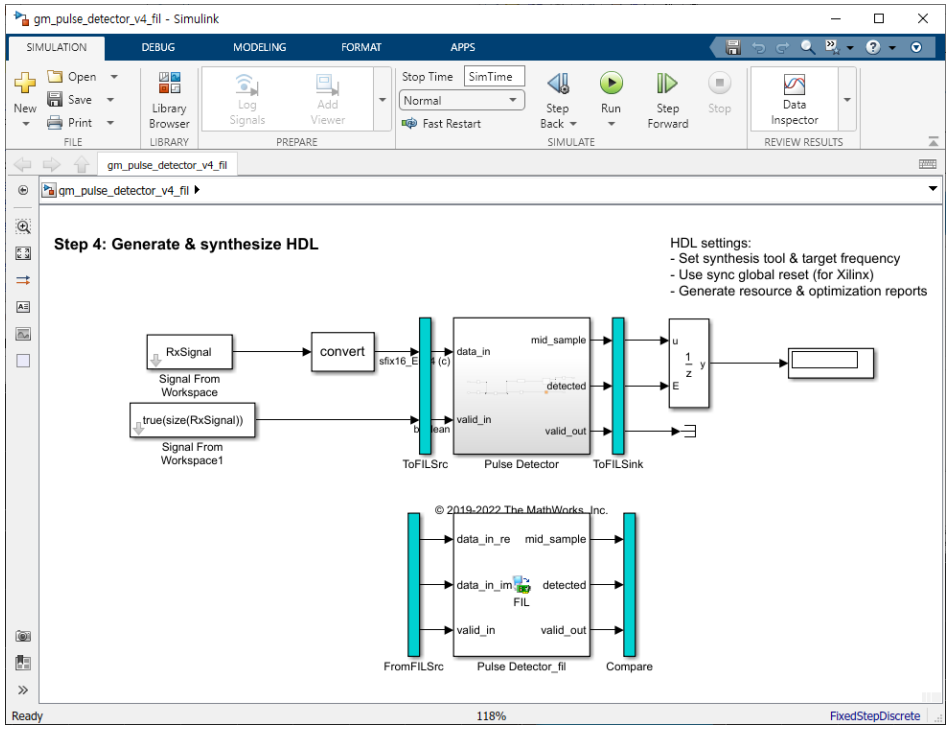
- FIL simulation with FPGA development board:
  - Reuse of existing MATLAB and Simulink testbenches
  - HDL code execution on FPGA
  - Flexible HDL sources (Generated or handwritten)
  - Automated generation of FIL infrastructure (Ethernet, JTAG, PCIe)



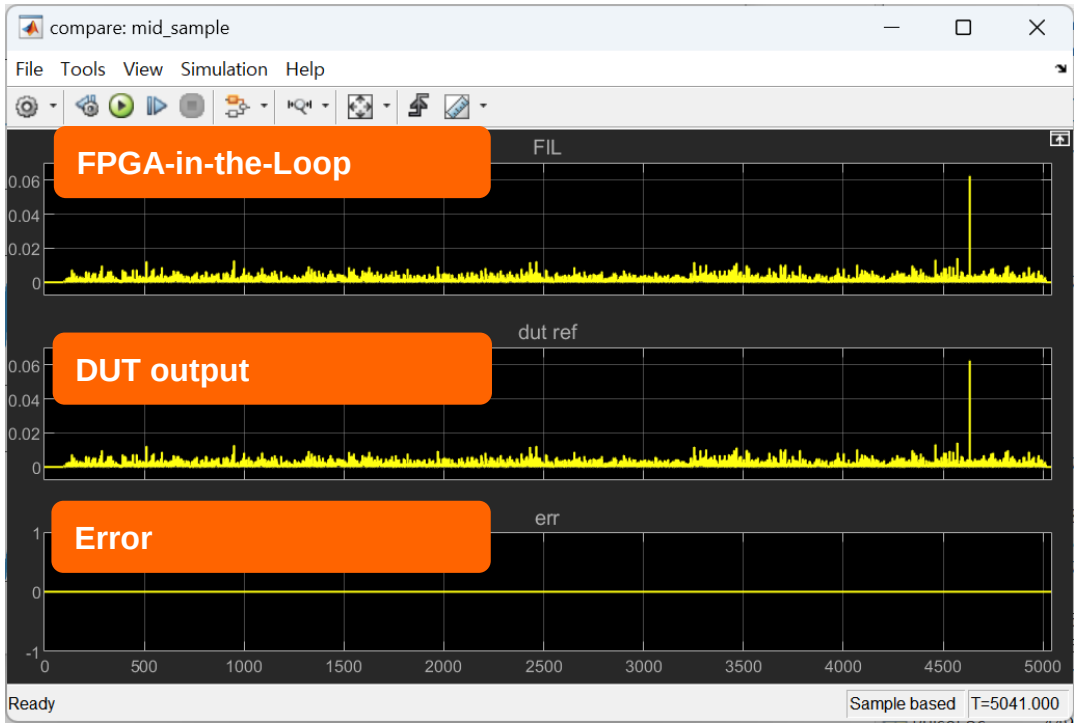
FPGA-in-the-Loop (FIL)

In this step, you will:

- Generate **FPGA-in-the-Loop** model from Simulink
- Check the results of FIL test and simulation are equal



FIL Model

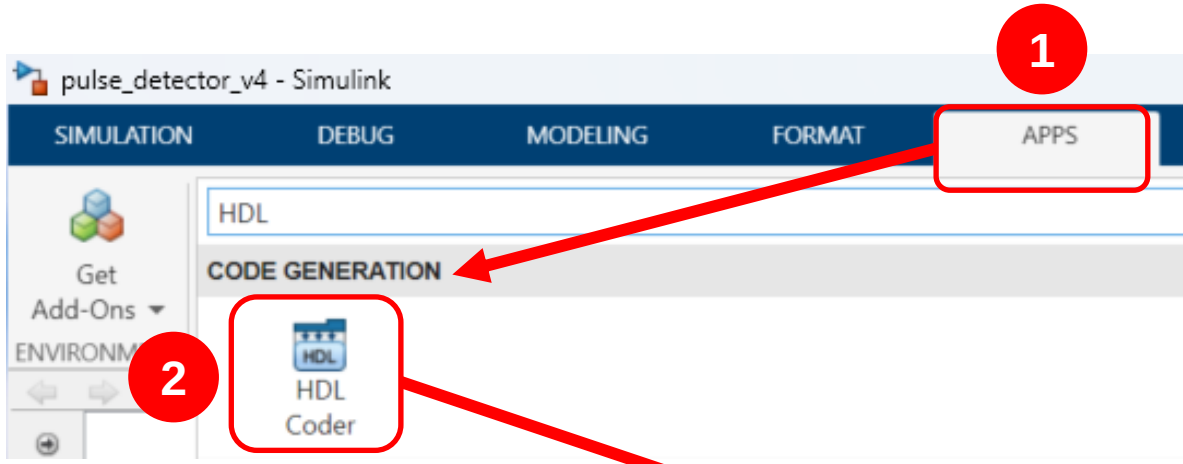


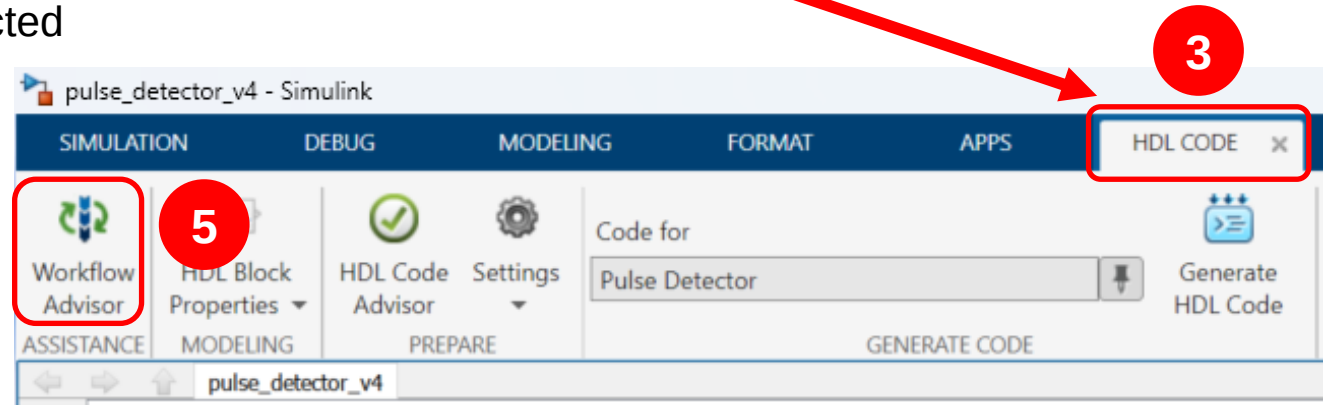
FIL Test Result

Demo1: FPGA-in-the-Loop Workflow

Verification of generated HDL Code

1. In the model **pulse_detector_v4.slx,** click **APPS**
2. Select **HDL Coder** under **CODE GENERATION** in the APP Gallery
3. HDL CODE Toolstrip will appear
4. Run hdlsetuptoolpath in MATLAB Command window to build FPGA-in-the-Loop
5. Click **Workflow Advisor** button to launch **HDL Workflow Advisor** to complete automated workflows for selected FPGA target devices





4

```
Command Window
>> hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...
'C:\Xilinx\Vivado\2020.2\bin\vivado.bat');
Prepending following Xilinx Vivado path(s) to the system path:
C:\Xilinx\Vivado\2020.2\bin
```

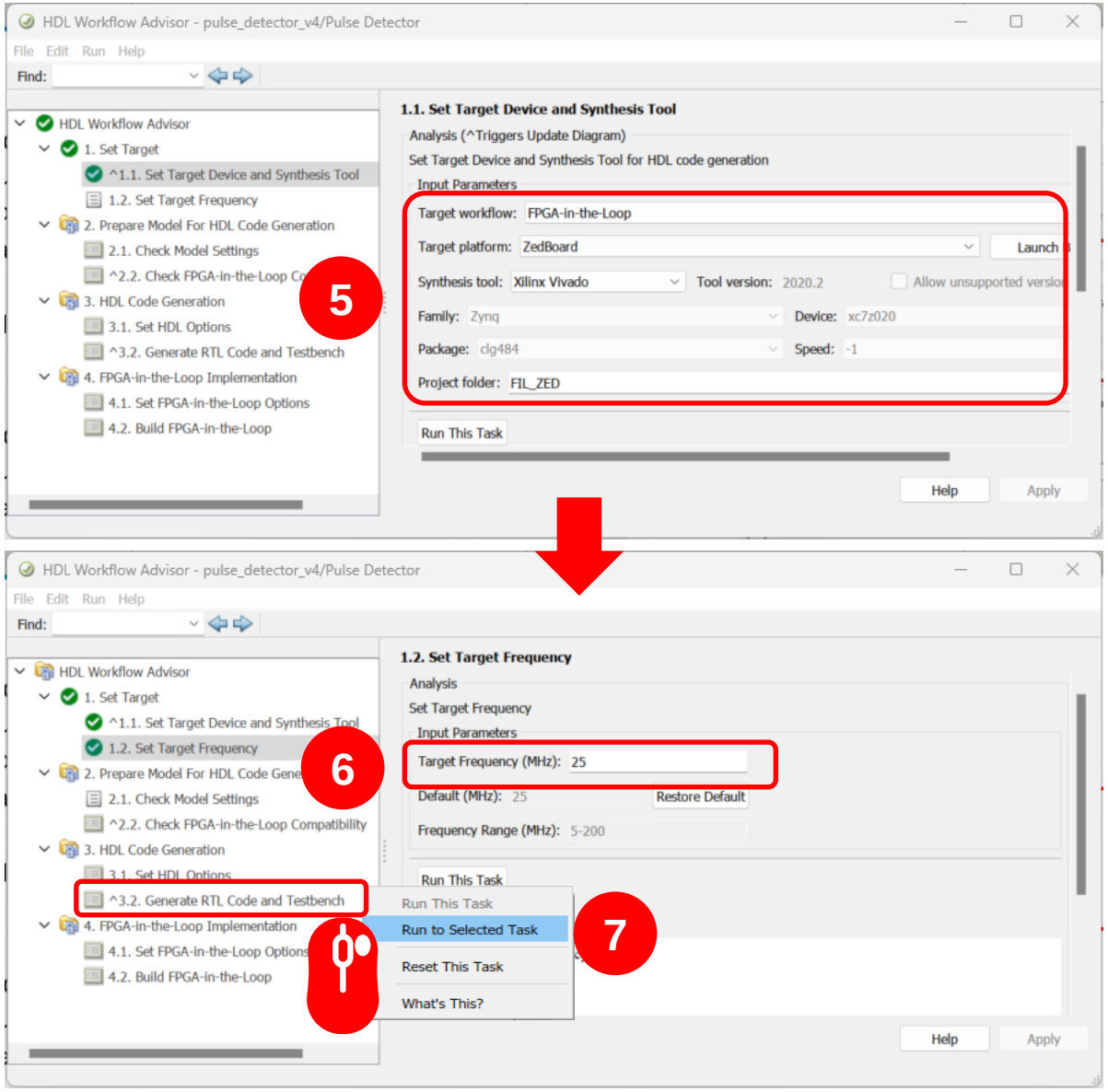
Demo 1: FPGA-in-the-Loop Workflow

Verification of generated HDL Code

6. In Step 1.1 of **HDL Workflow Advisor**, set as below and click **Run This Task** button in each step
   - Target Workflow: FPGA-in-the-Loop
   - Target Platform: ZedBoard
   - Synthesis tool: Xilinx Vivado
   - Project folder: Specify the name you want to save the artifacts

7. In Step 1.2 of **HDL Workflow Advisor**, set as below and click **Run This Task** button in each step
   - Target Frequency(MHz): 25

8. Run through steps 2 to 3 (right-click on Step 3.2 > **Run to Selected Task**)

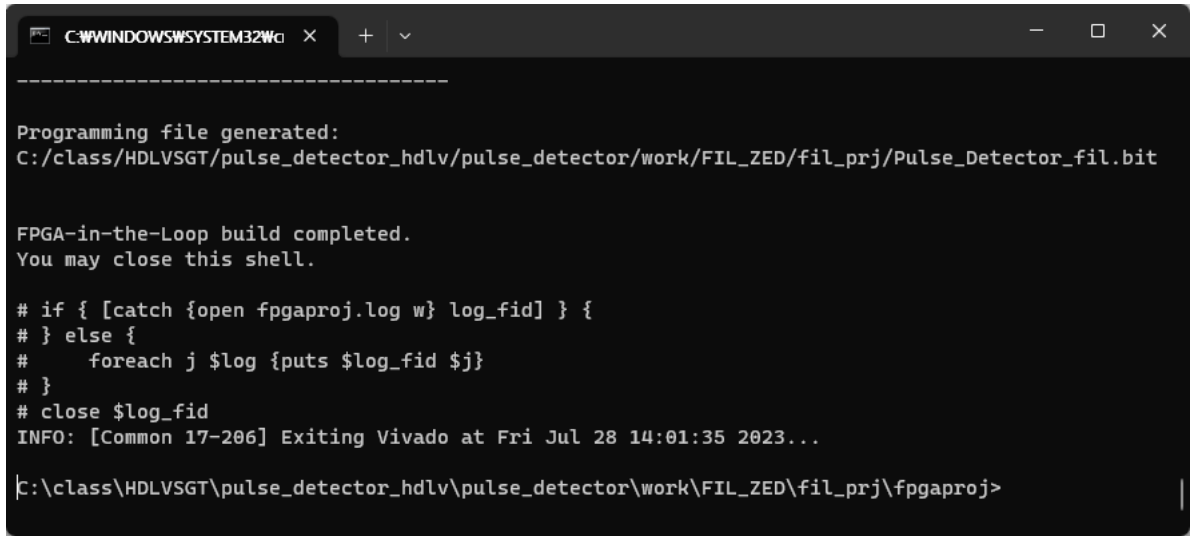


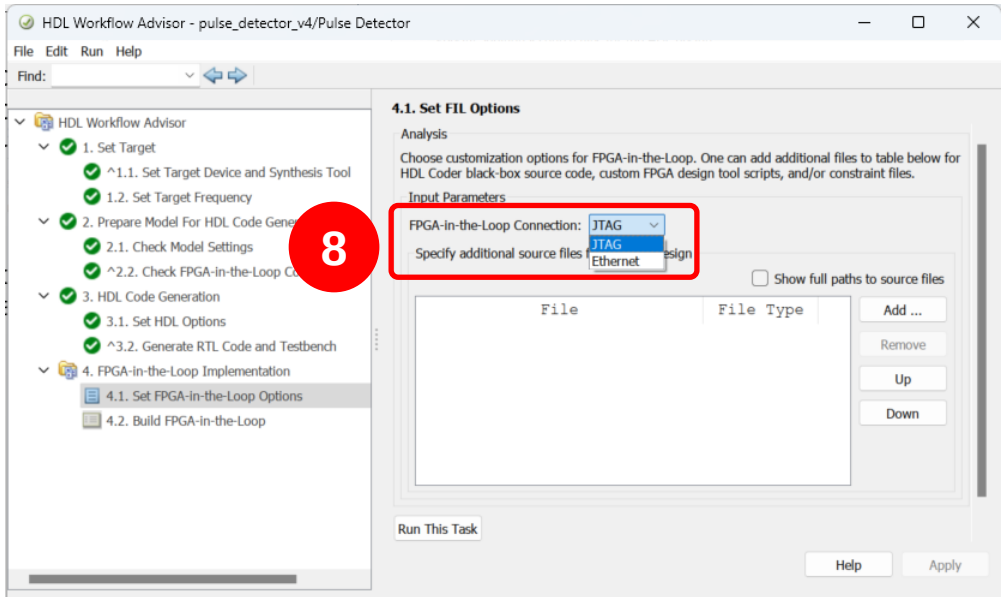
Demo 1: FPGA-in-the-Loop Workflow

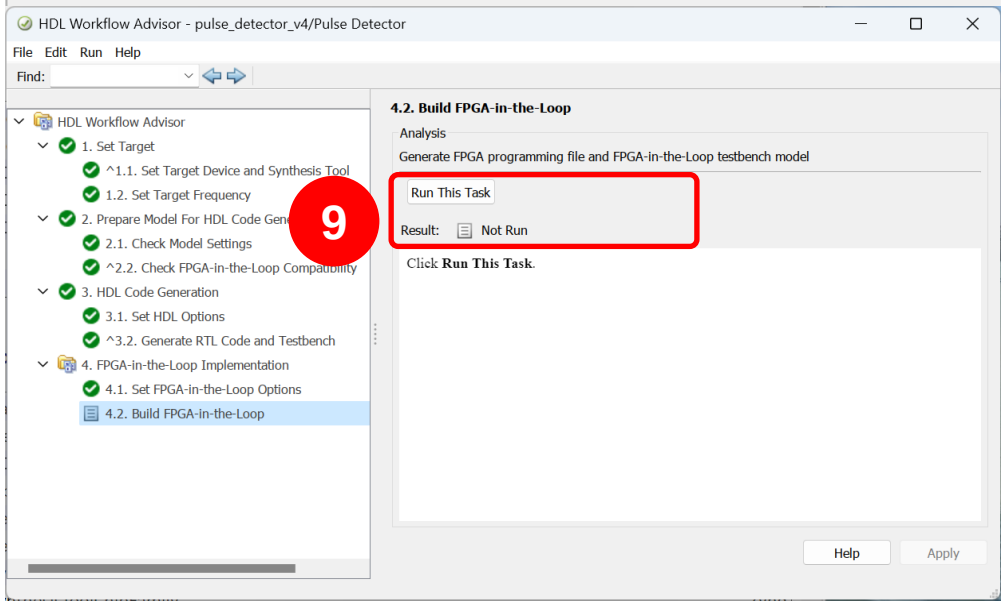
Verification of generated HDL Code

9. In step 4.1, Select FIL Connection as JTAG and click **Run This Task** (Ethernet provides more bandwidth than JTAG, but we choose JTAG for simple HW configuration in this demo)

10. In step 4.2, **Run This Task** to build FPGA-in-the-Loop. You can observe the FIL build process on the Vivado Console window. (The build process takes 20-30 minutes.)

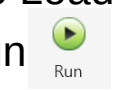


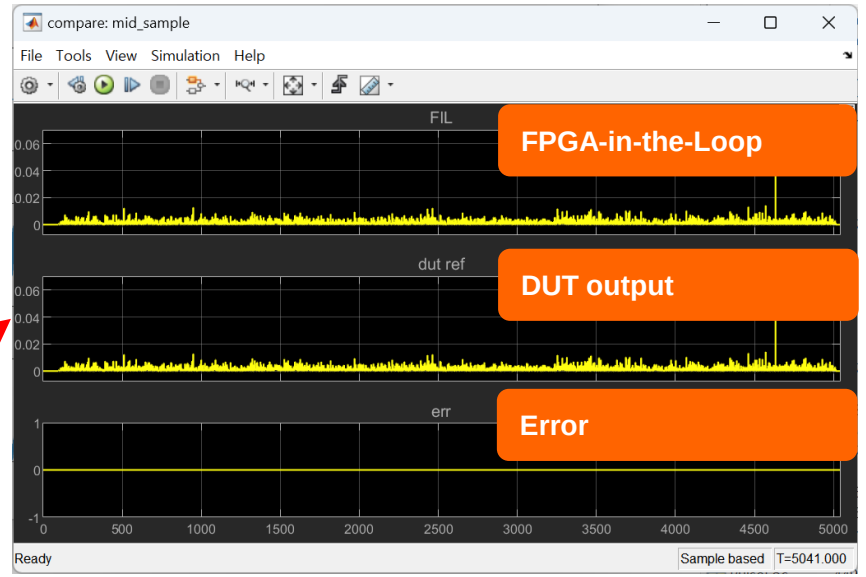


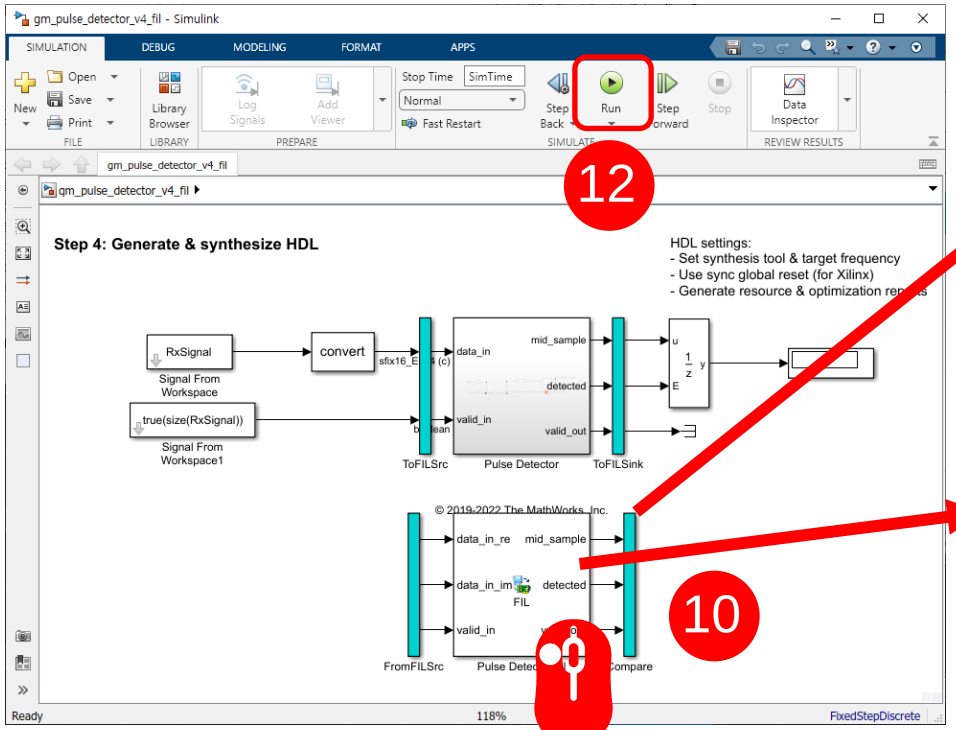


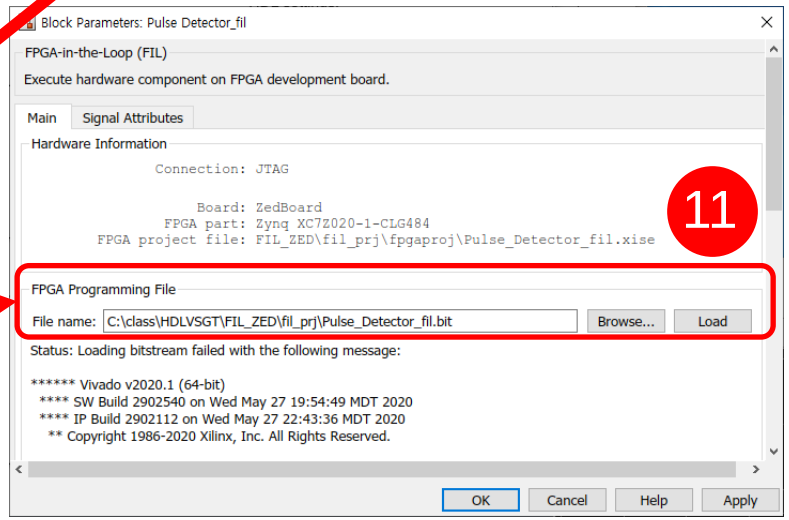
Demo 1: FPGA-in-the-Loop Workflow

Verification of generated HDL Code

11. In the generated FIL model (gm_pulse_detector_v4_fil.slx), double-click the FIL block to configure the FPGA Programming file.
12. Click the Load button to program FPGA using the selected file.
13. Click Run  button to run the FIL test
14. Click the Compare block in the FIL model to see the result



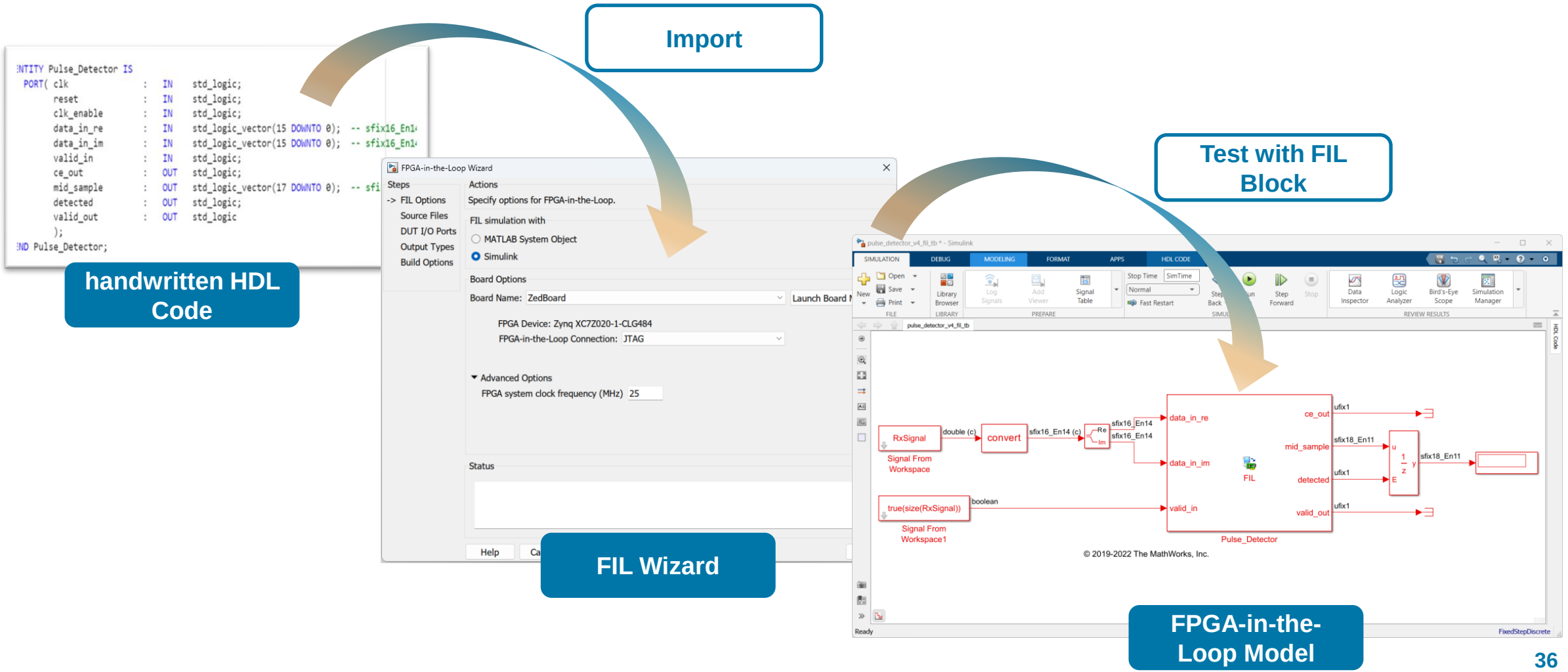




Demo 2: FPGA-in-the-Loop Workflow for Handwritten HDL Code

Verification of handwritten HDL Code

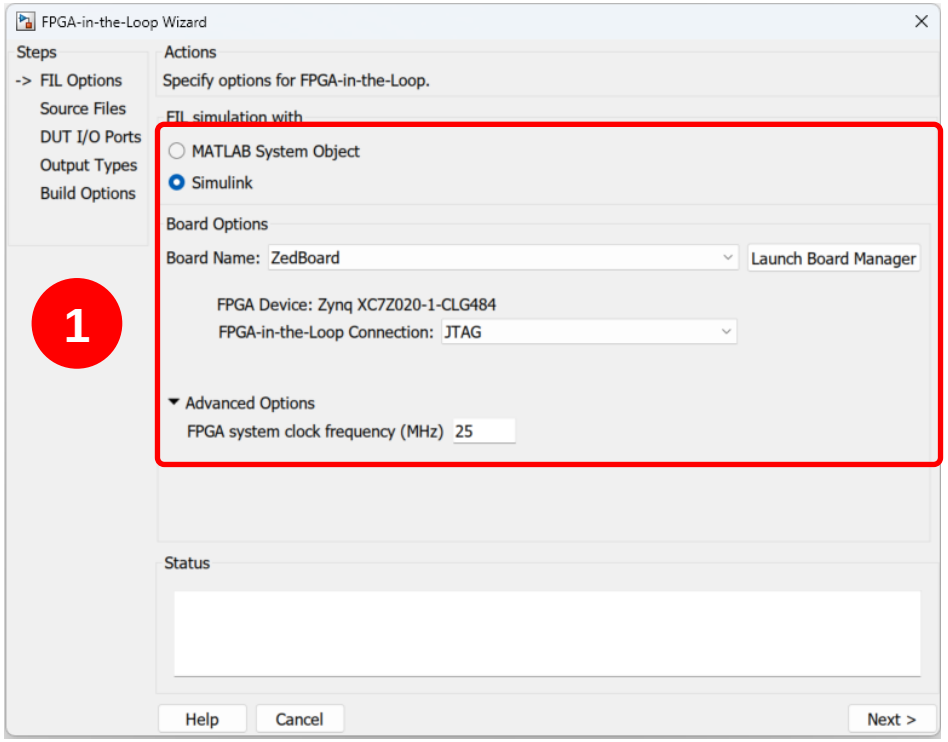
- Bring your handwritten HDL Code using FIL Wizard to run the coexecution between FPGA and MATLAB/Simulink



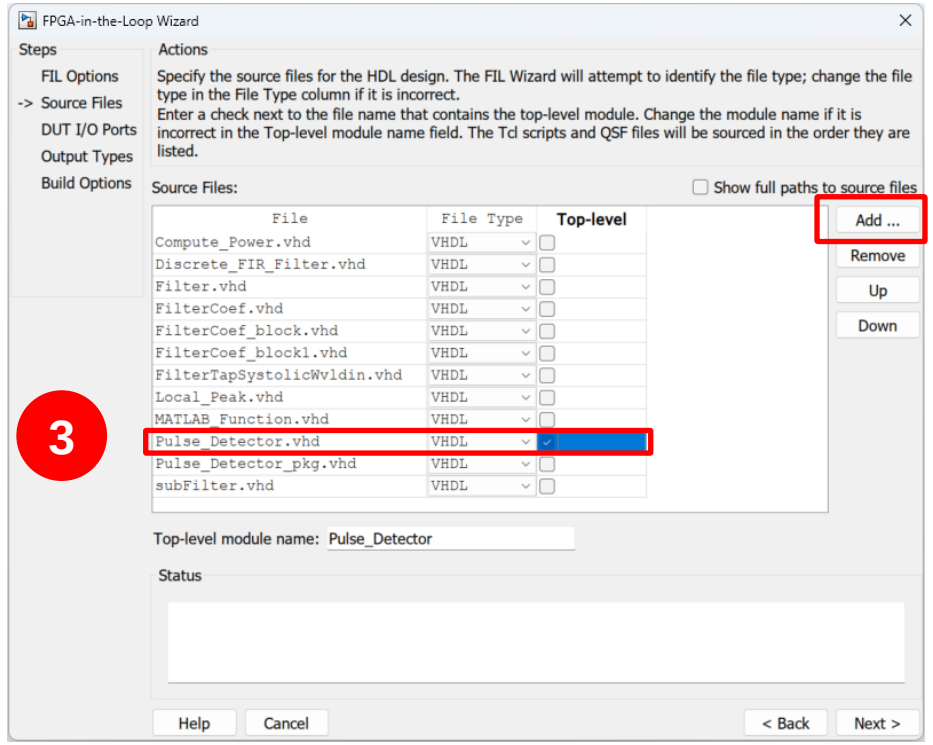
Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

1. Launch **filWizard** by executing the command below in MATLAB command window. ***>> filWizard***
   In the **filWizard**, set each parameters as below and click **Next**
   - FIL simulation with: Simulink
   - Board Name: ZedBoard
   - FPGA-in-the-Loop Connection: JTAG
   - FPGA system clock frequency(MHz): 25



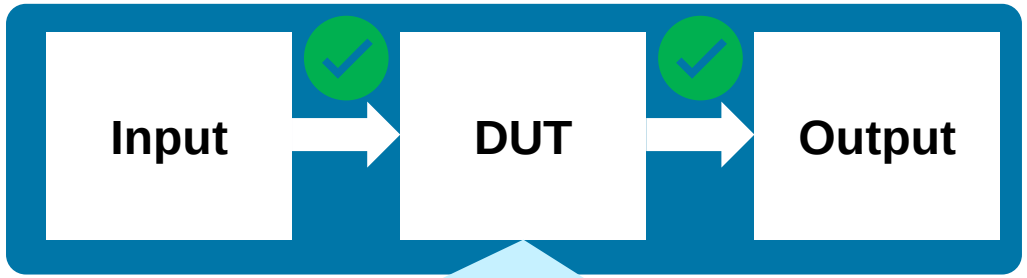
2. Click **Add** to Specify the source files for the HDL Design
3. Enable checkbox of **Top-level** File out of the source files and Click **Next.**
   In this demo, *Pulse_Detector.vhd* is the Top-level File and other files are under that level.

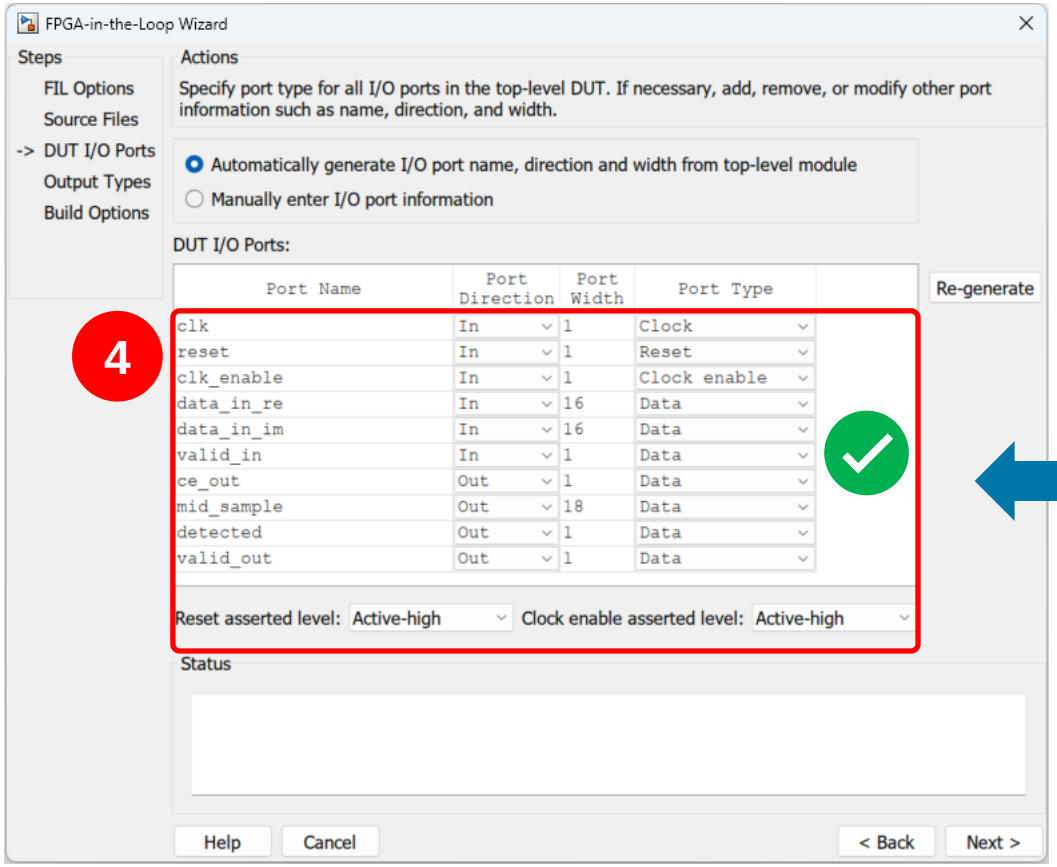


Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

4. Review the DUT Input and Output ports of Top-level File and check if is assigned as intended. If not, change the Port Type or Manually enter I/O port information.
Click **Next.**





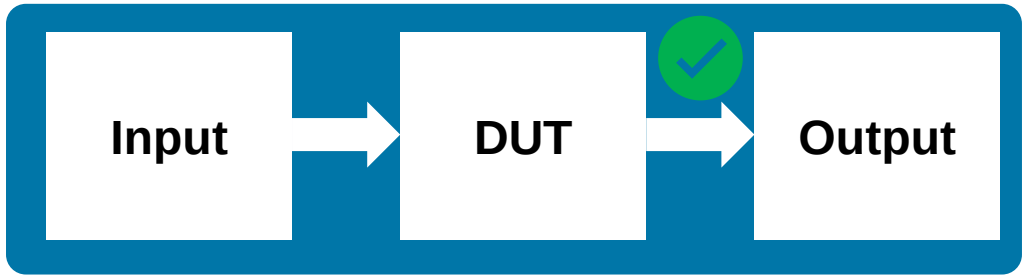

```
ENTITY Pulse_Detector IS
  PORT( clk             :   IN    std_logic;
        reset           :   IN    std_logic;
        clk_enable      :   IN    std_logic;
        data_in_re      :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        data_in_im      :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        valid_in        :   IN    std_logic;
        ce_out          :   OUT   std_logic;
        mid_sample      :   OUT   std_logic_vector(17 DOWNTO 0);  -- sfix18_En11
        detected        :   OUT   std_logic;
        valid_out       :   OUT   std_logic
        );
END Pulse_Detector;
```

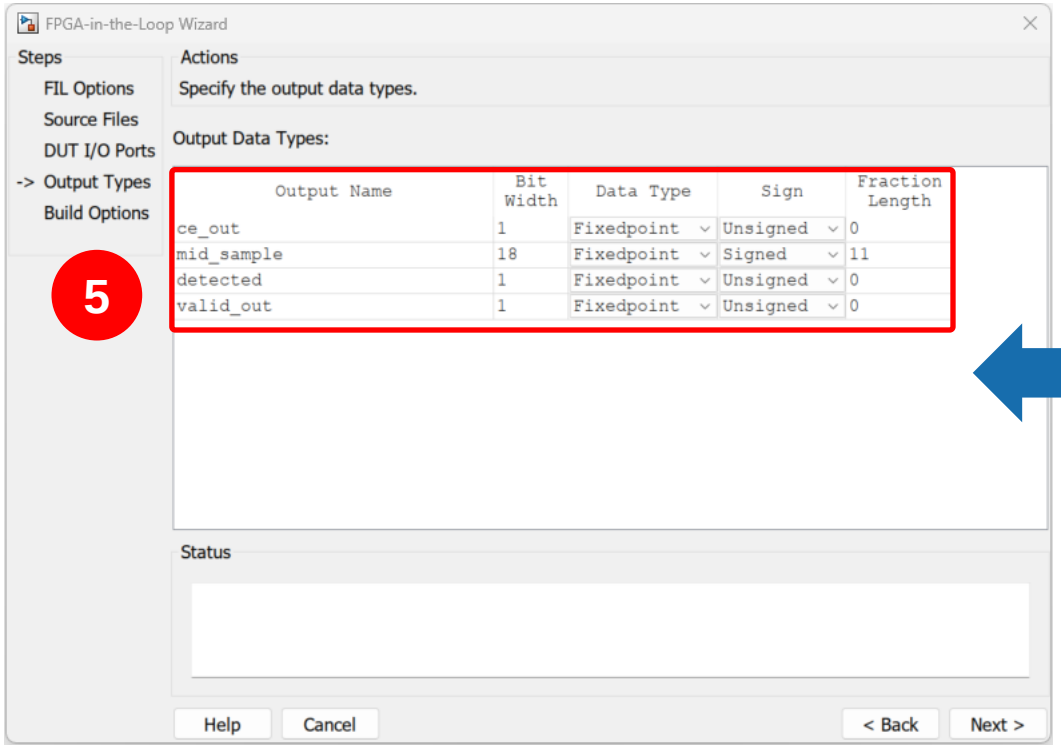
<Top-level File: Pulse_Detector.vhd>

Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

5. Review the Output Data Types of Top-level File and set properly matches in the original HDL files. Click **Next.** This step is very important because the Simulink Testbench will assume the output data type of FIL Block based on this configuration.





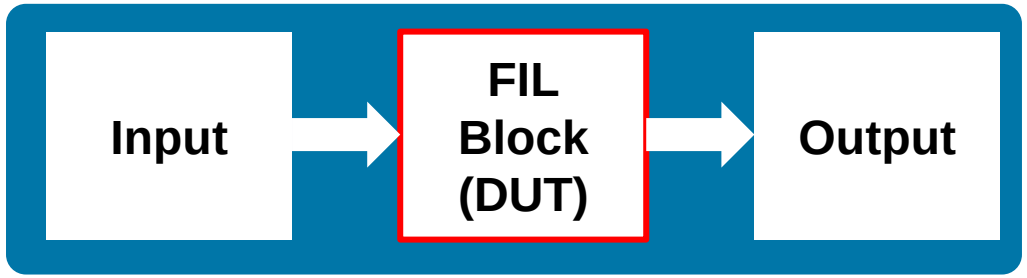

```
ENTITY Pulse_Detector IS
  PORT( clk              :   IN    std_logic;
        reset            :   IN    std_logic;
        clk_enable       :   IN    std_logic;
        data_in_re       :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        data_in_im       :   IN    std_logic_vector(15 DOWNTO 0);  -- sfix16_En14
        valid_in         :   IN    std_logic;
        ce_out           :   OUT   std_logic;
        mid_sample       :   OUT   std_logic_vector(17 DOWNTO 0);  -- sfix18_En11
        detected         :   OUT   std_logic;
        valid_out        :   OUT   std_logic
        );
END Pulse_Detector;
```

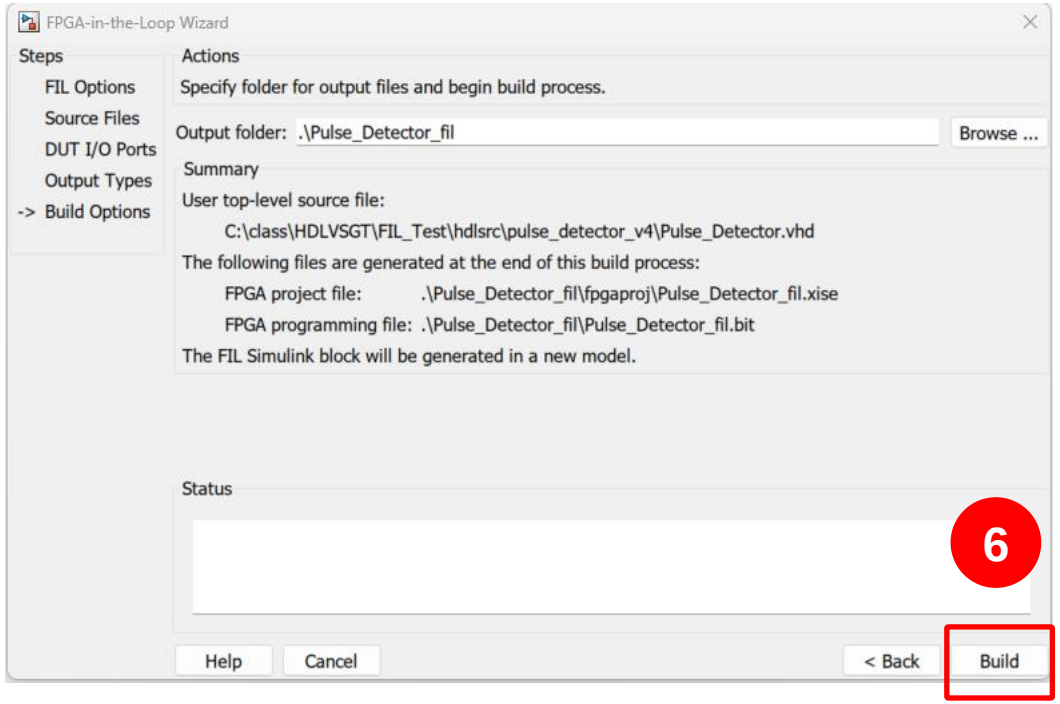
<Top-level File: Pulse_Detector.vhd>

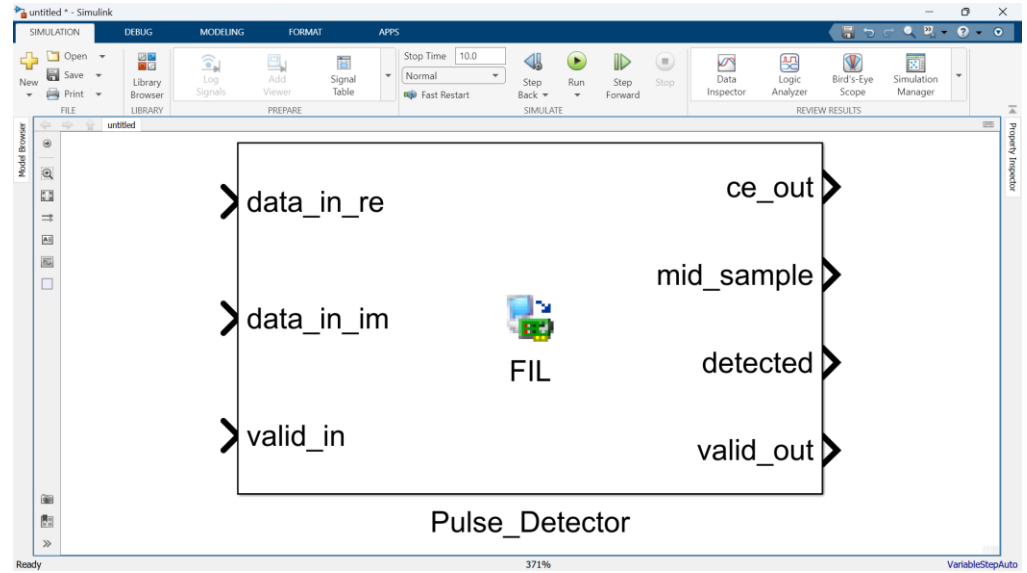
Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

6. The Build process generates a FIL block from your HDL code and saves the output to the folder you specify. Click the **Build** button to run the build process and generate the FIL block.
   (You can see the build process through the FPGA Synthesis Tool Console).





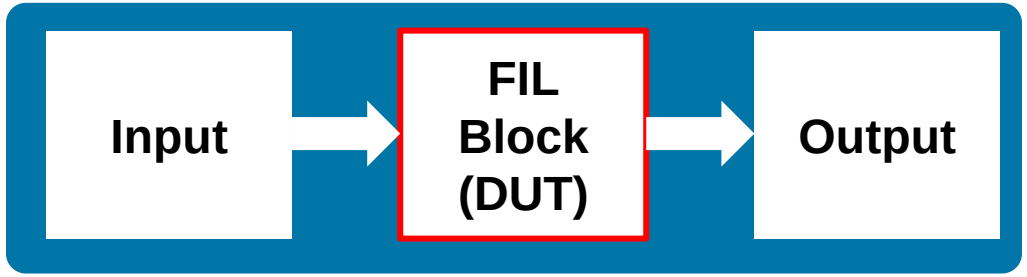


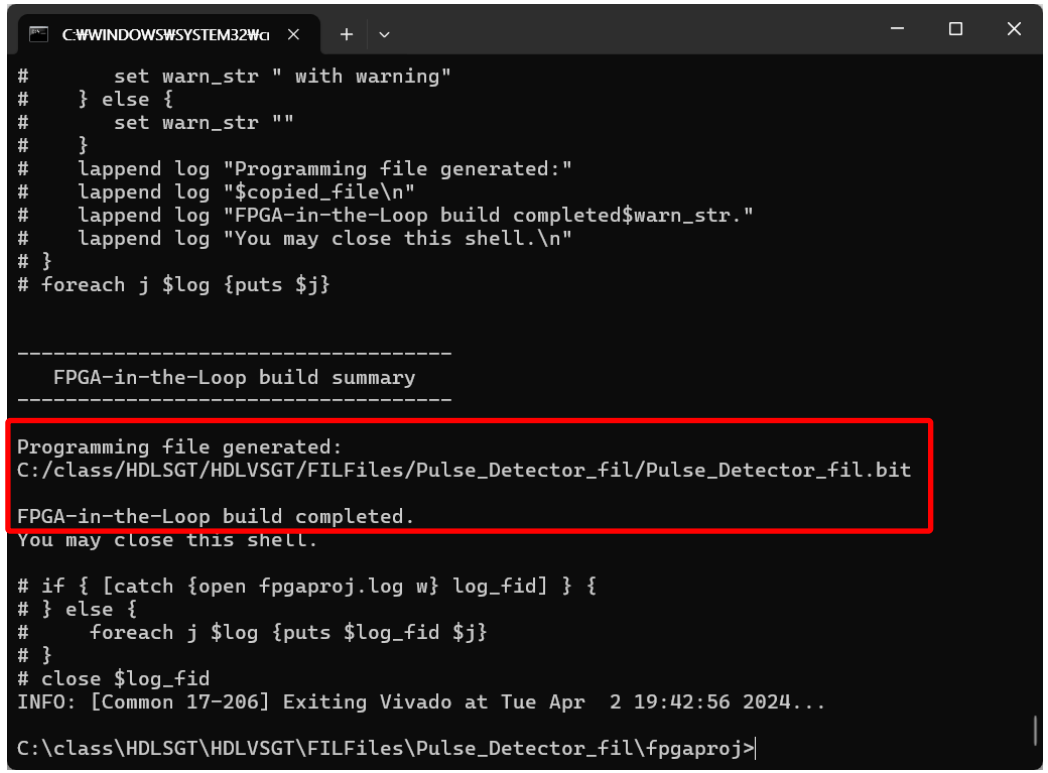
<FPGA-in-the-Loop Block>

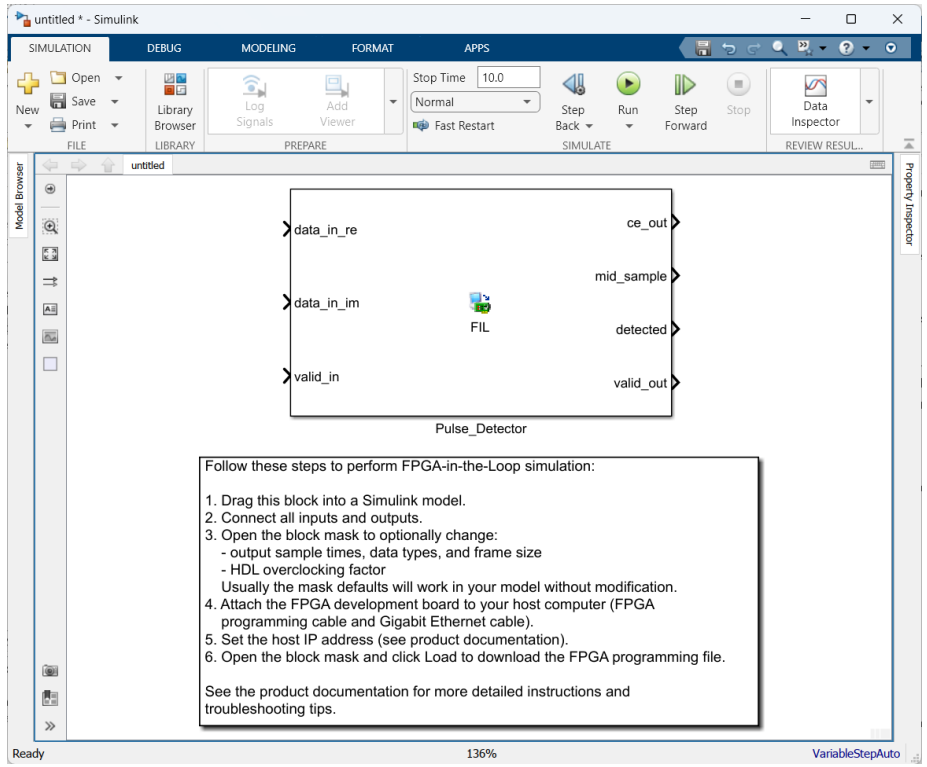
Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

After pressing the Build button and waiting 2-3 minutes, the FPGA Synthesis Tool Console will pop up. Here you can see the build process of the FPGA-in-the Loop.





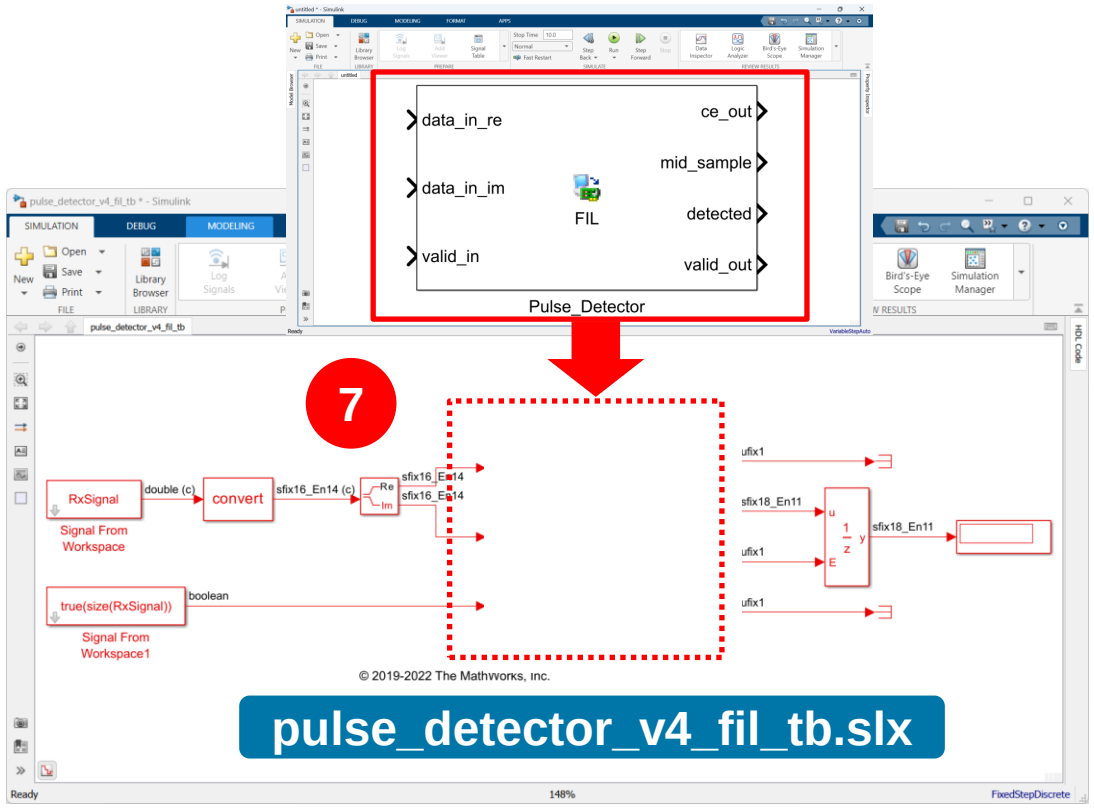


<FPGA-in-the-Loop Block>

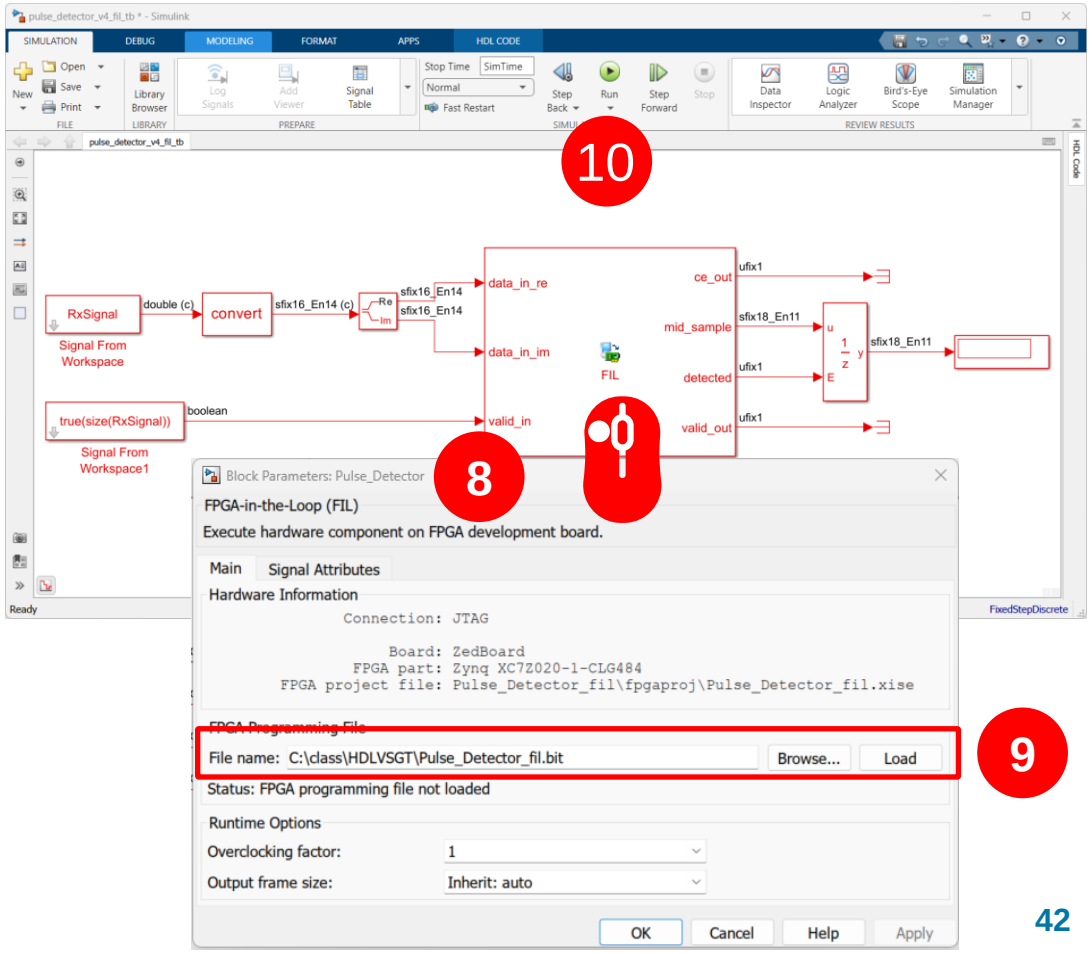
Demo 2 : FPGA-in-the-Loop Workflow

Verification of handwritten HDL code

7. Copy the FIL block to the prebuilt model (**pulse_detector_v4_fil_tb.slx)** to complete the demo and connect input/output ports with matched signals. The prebuilt model is under the ./HDLV_Tutorial/FILFiles/.
8. Double-click the FIL block to configure the FPGA Programming file.



9. Click the Load button to program FPGA using the selected file.
10. Click Run  button to run the FIL test

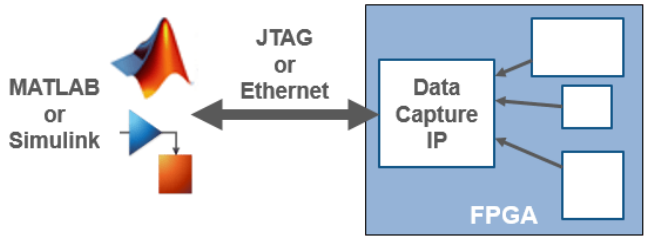


Conclusion: FPGA-in-the-Loop with HDL Verifier

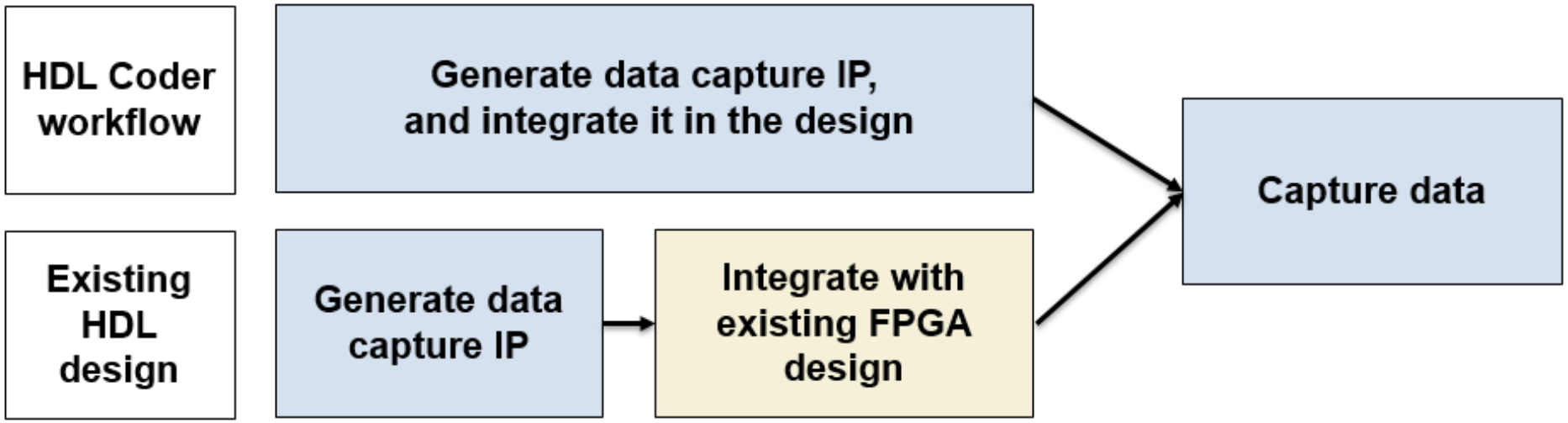
- FPGA-in-the-Loop
  - Enables FPGA verification within Simulink or MATLAB environment
  - Bridges the gap between high-level simulation and hardware implementation
- Benefits
  - Hardware acceleration for verification: Relatively fast compared to cosimulation because it uses physical communication between the target board and MATLAB
  - Reduced Development Time: By testing algorithms on actual hardware without manual coding
  - Increased Confidence: Through early detection of design flaws by running algorithms on real FPGA hardware

FPGA Data Capture

FPGA Data Capture (FDC)



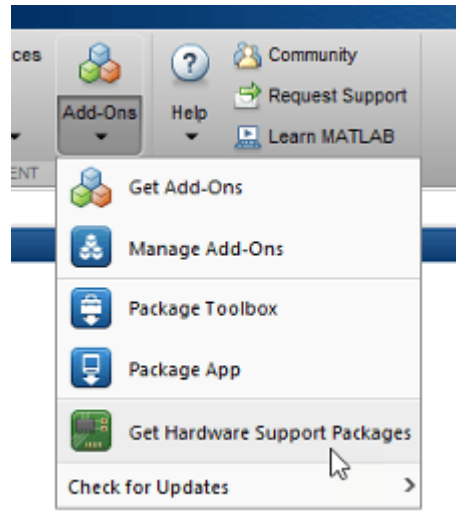
- Integrate Data Capture IP in your design.
- Debug the design running on FPGA by capturing Signal Data and observe it in MATLAB/Simulink
- Workflows
  - **Generated HDL** : Use HDL Workflow Advisor tool to generate the data capture IP and integrate it into your FPGA design
  - **Existing HDL**: Use HDL Verifier to generate the Data Capture IP. Then manually integrate the generated IP into FPGA design



FPGA Data Capture (FDC)

- **Requirement**
  - Xilinx Vivado™ Design Suite, with a supported version listed in the HDL Language Support and Supported Third-Party Tools and Hardware
  - Set Xilinx tool path in MATLAB as per your machine setup ***hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath', … 'C:\Xilinx\Vivado\2020.2\bin\vivado.bat');***

- **Setup**
  - Connect the Ethernet and the USB JTAG Cables to the ZedBoard
  - On the MATLAB Home tab, in the Environment section, select Add-Ons
    - Get Hardware Support Packages.
    - Download and install HDL Coder Support Package for Xilinx Zynq™ Platform
  - Follow steps for guided SD Card setup (one time only).



Demo1 - FDC Workflow using HDL workflow advisor

In this demo, you will use HDL Workflow advisor to generate complete design.
Download folder from following link and unzip it. Change directory to unzipped folder.

You have following options

Use artifacts from prebuilt folder

a) Add prebuilt and scripts folder to MATLAB path. A prebuilt bit file is provided.
b) Run MATLAB script **program_board** to load the bitfile on to the FPGA.
c) To test the design on the board, use provided test script **pulse_detector_ipcore_hw_demo.mlx**
d) For demo of FPGA Data Capture, type **launchDataCaptureApp.** Select capture data. Run the live script.
e) Logic Analyzer will display capture signals.

Follow steps in this tutorial to generate your own bitfile

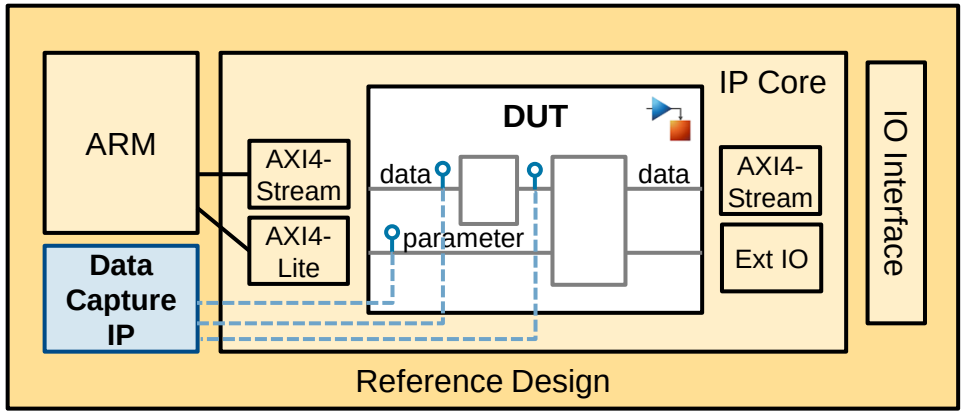
Demo1 - FDC Workflow using HDL Workflow Advisor

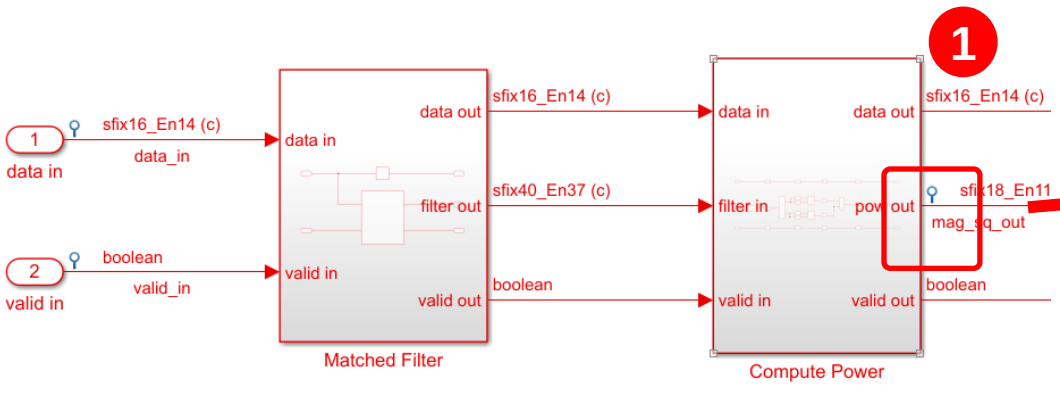
In this demo, you will use HDL Workflow advisor to generate complete design.

- In setup.m, update Xilinx tool path and run setup.m

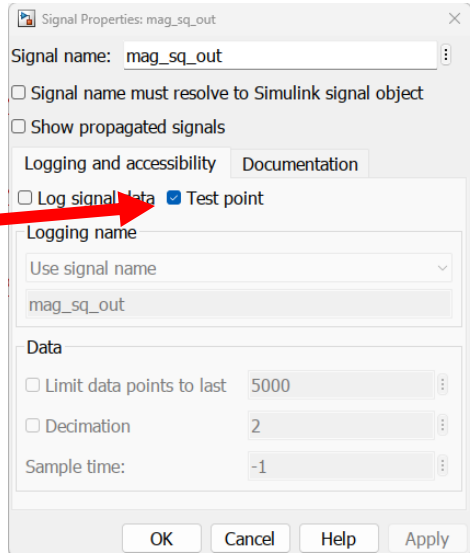
Steps

1. setup.m will open model pulse_detector_ipcore.slx.
   - Go to subsystem **Pulse Detector IP -> Pulse Detector**
   - Insert Test points for signals that you want to observe/debug
   - In the Simulink Editor, right-click the signal, and select **Properties**. Then, select **Test Point**.



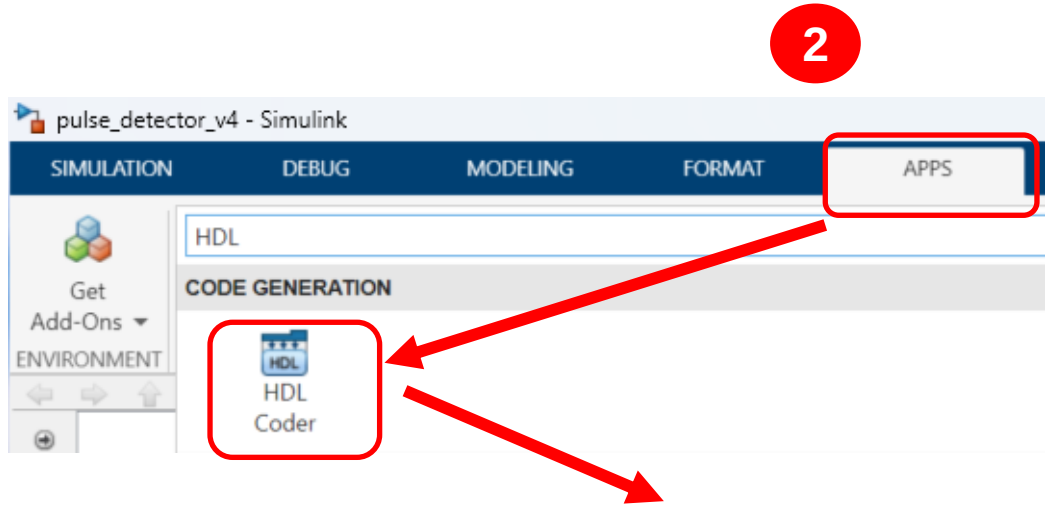


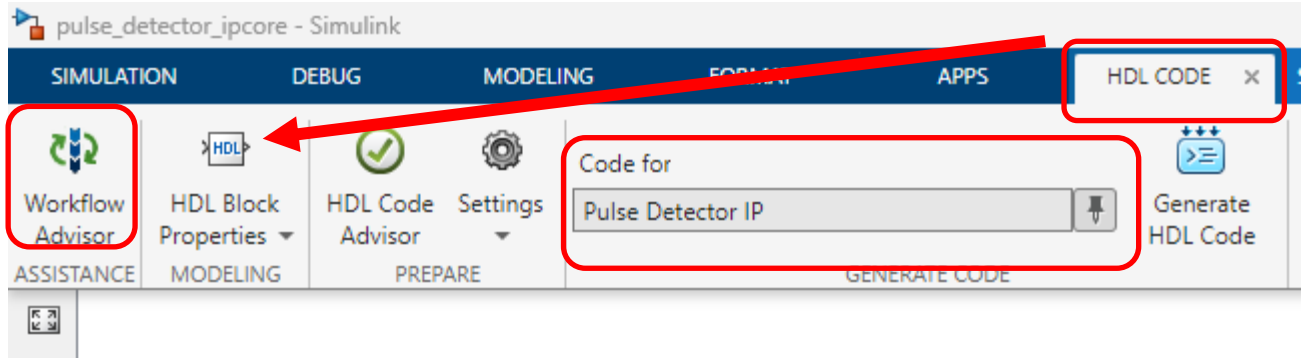
Pusle Detector



Demo1 - FDC Workflow using HDL Workflow Advisor

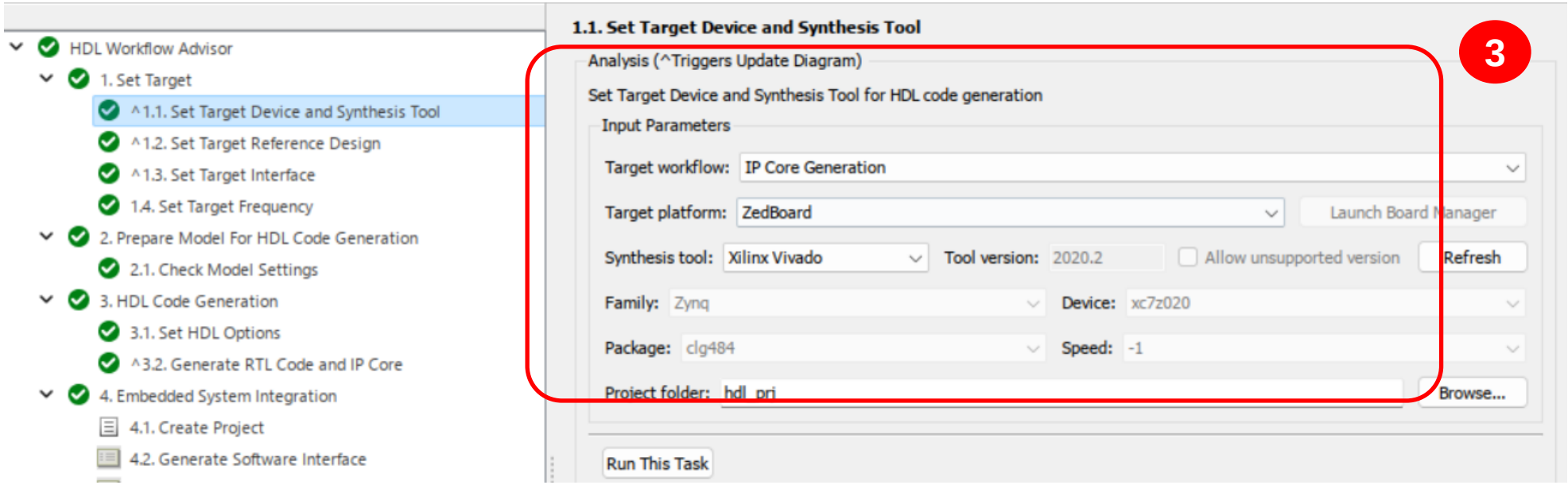
2. Follow steps listed below to launch HDL Workflow Advisor
   - In the model**,** click **APPS**
   - Search for **HDL Coder and select it**
   - HDL CODE Toolstrip will appear
   - **Select "Pulse Detector IP" in model and Pin it**
   - Click **Workflow Advisor** to launch HDL Workflow Advisor





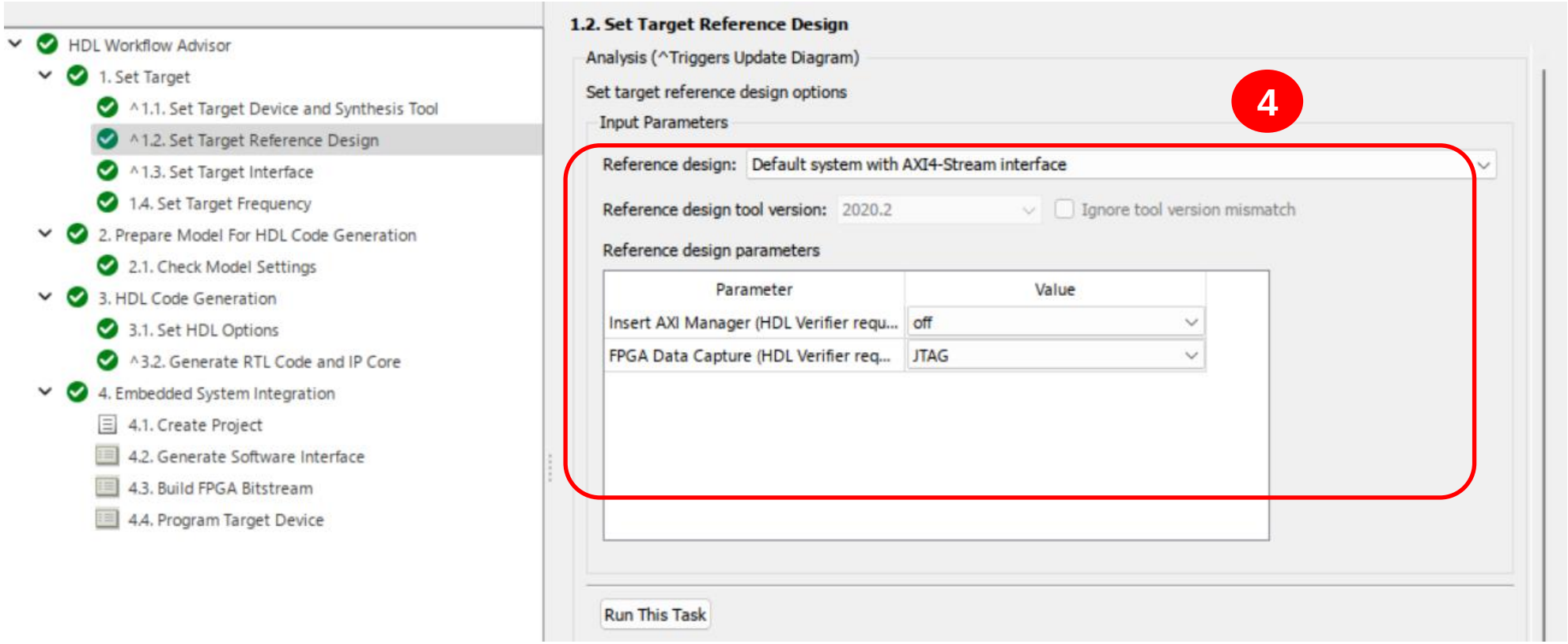
Demo1 - FDC Workflow using HDL workflow advisor

3. In Step 1.1 of **HDL Workflow Advisor**, set as below and click **Run This Task** button in each step
   - Target Workflow: IP Core Generation
   - Target Platform: ZedBoard
   - Synthesis tool: Xilinx Vivado
   - Project folder: Specify the name you want to save the artifacts



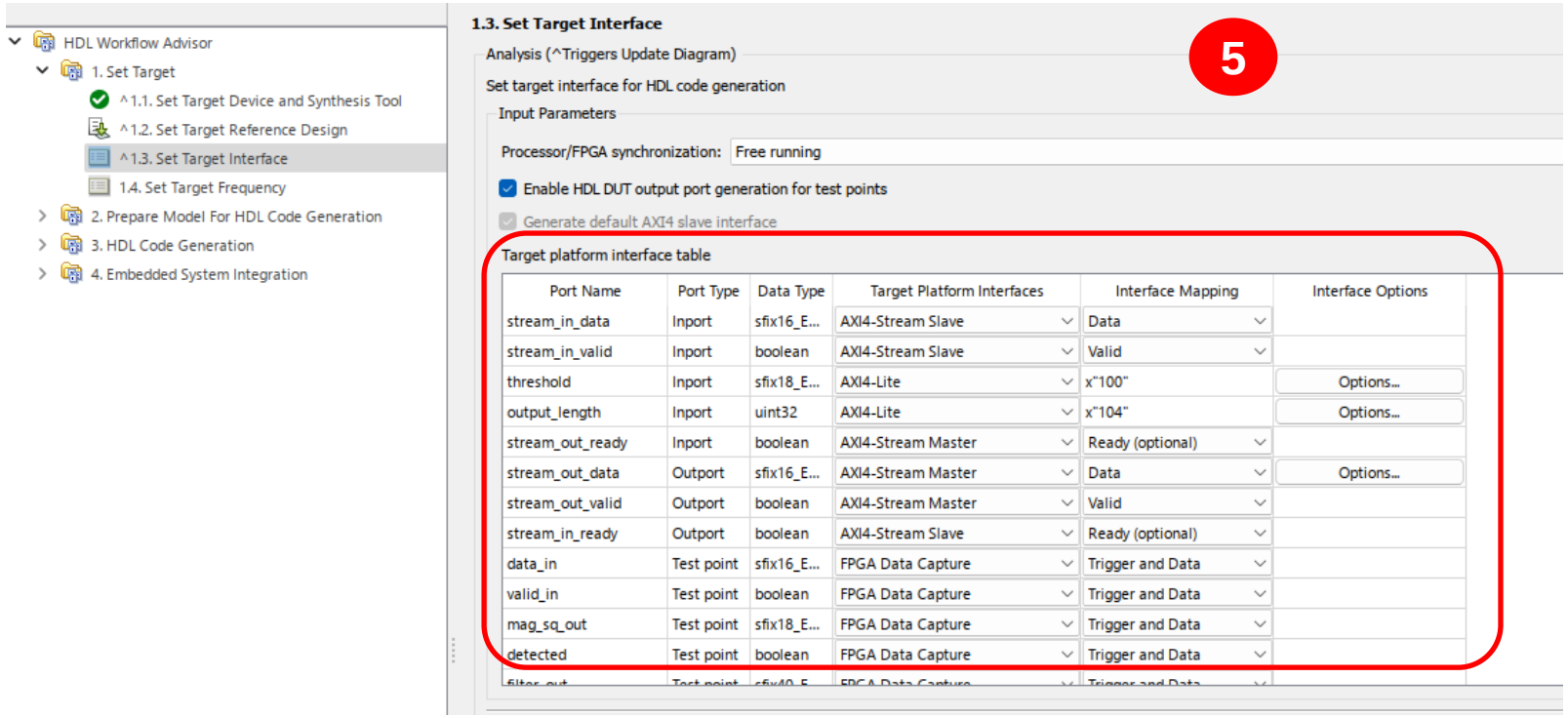
Demo1 - FDC Workflow using HDL workflow advisor

4. In Step 1.2 of **HDL Workflow Advisor**, set as below and click **Run This Task** button



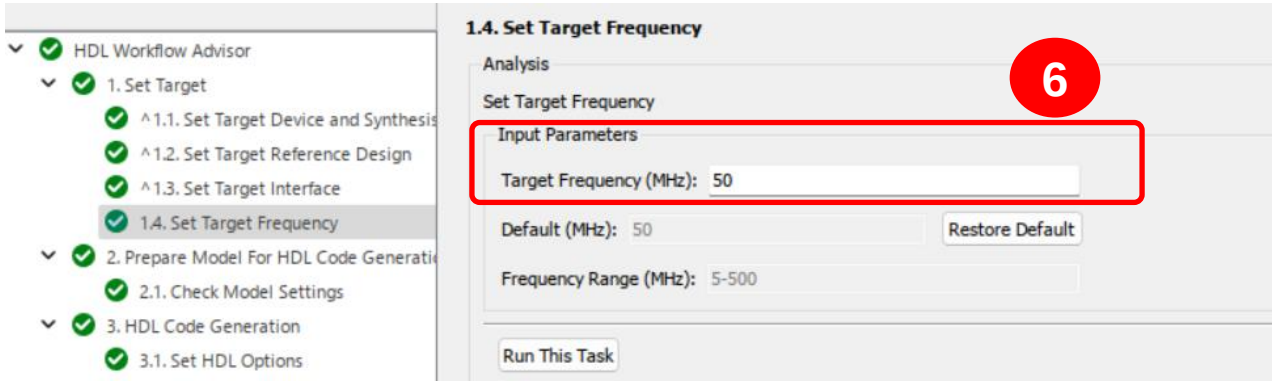
IP Core gen workflow with FPGA Data Capture IP

5. In Step 1.3 of **HDL Workflow Advisor**, set as below and click **Run This Task** button
   - Map Signals of the design to Target Bus interface
     - Input Signals, stream_in_data and stream_in_valid are mapped to AXI4-Stream Slave
     - Output Signals, stream_out_data and stream_out_valid are mapped to AXI4-Stream Master
     - Threshold as tunable parameter is mapped to AXI4-lite register interface with address offset of hex(100)
   - Enable HDL DUT output port generation for test ports, it will populate the test points as output ports
   - Map Test point signals to FPGA Data capture Interface and choose option as Data only or Trigger and Data.

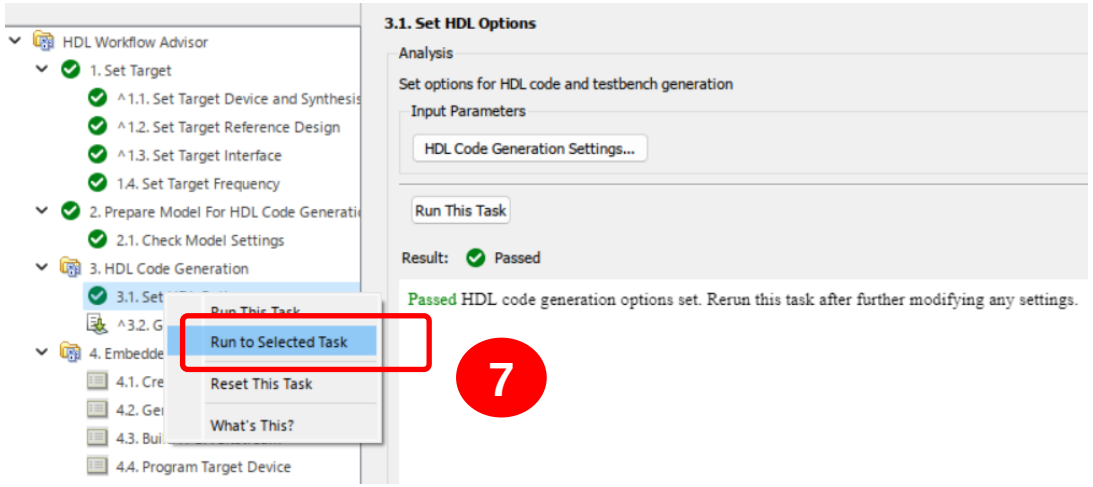


Demo1 - FDC Workflow using HDL workflow advisor

6. In Step 1.4 of **HDL Workflow Advisor**, set as below and click **Run This Task** button
   - Set Target Frequency to 50 Mhz

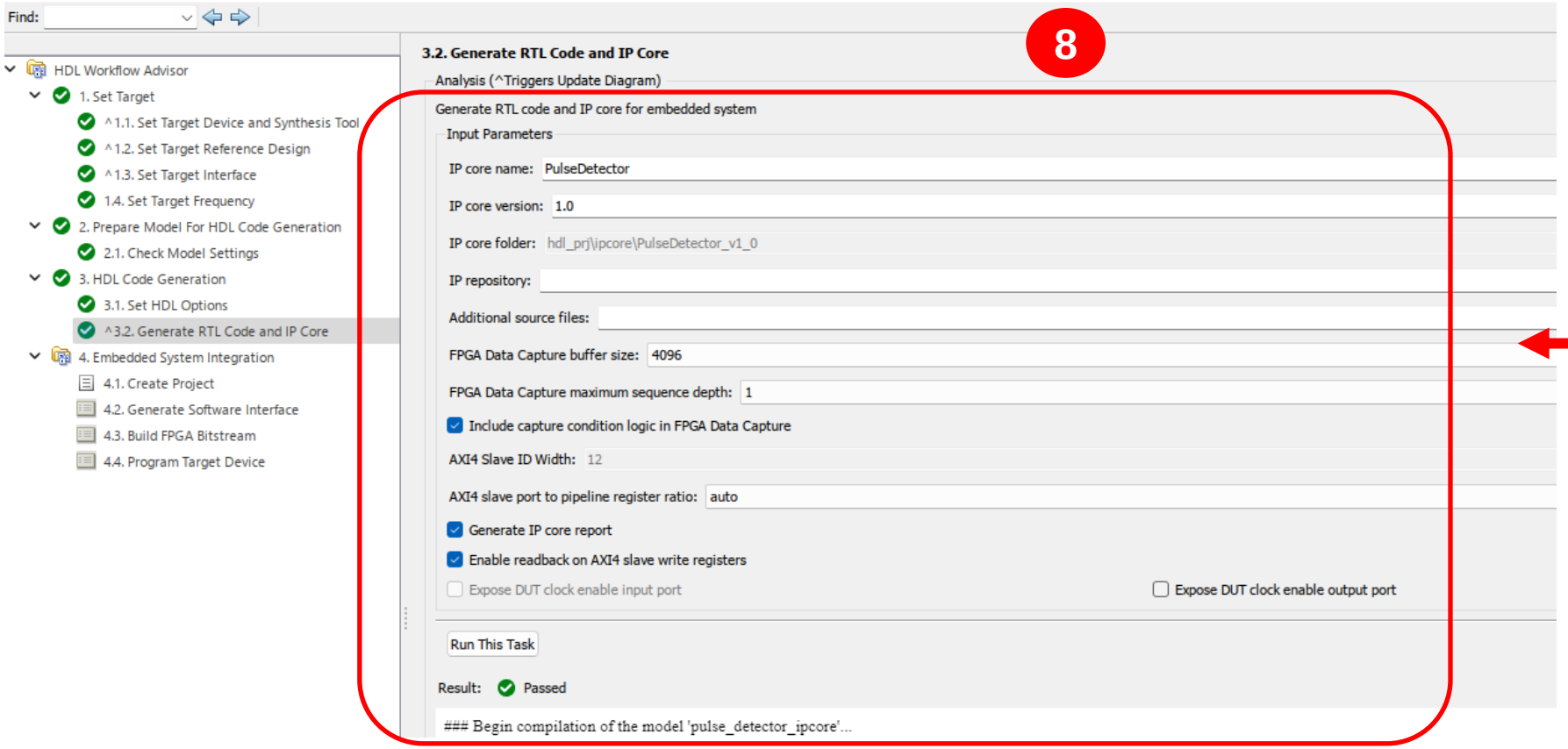


7. Go to Step 3.1 of **HDL Workflow Advisor**, and right click and Select **Run to Selected Task**



IP Core gen workflow with FPGA Data Capture IP

8. In Step 3.2 of **HDL Workflow Advisor**, set as below and click **Run This Task** button.
   Ensure that there is no error and review the HDL code generation Report



- Data capture Buffer size
- Include Capture logic
- Enable Readback on AXI4 Slave write registers

IP Core gen workflow with FPGA Data Capture IP

- Review HDL Code generation Report (IP Core Generation Report)

Contents

Summary
Clock Summary
Code Interface Report
Timing And Area Report
  High-level Resource Report
Optimization Report
  Distributed Pipelining
  Streaming and Sharing
  Delay Balancing
  Adaptive Pipelining
  Hierarchy Flattening
  Target Code Generation
  Code Reuse
IP Core Generation Report

Referenced Models

Target Interface Configuration

You chose the following target interface configuration for pulse_detector_ipcore :

Processor/FPGA synchronization mode: **Free running**

Target platform interface table:

Port Name	Port Type	Data Type	Target Platform Interfaces	Interface Mapping	Interface Options
stream_in_data	Inport	sfix16_En14	AXI4-Stream Slave	Data	
stream_in_valid	Inport	boolean	AXI4-Stream Slave	Valid	
threshold	Inport	sfix18_En11	AXI4-Lite	x"100"	
output_length	Inport	uint32	AXI4-Lite	x"104"	
stream_out_ready	Inport	boolean	AXI4-Stream Master	Ready (optional)	
stream_out_data	Outport	sfix16_En14	AXI4-Stream Master	Data	
stream_out_valid	Outport	boolean	AXI4-Stream Master	Valid	
stream_in_ready	Outport	boolean	AXI4-Stream Slave	Ready (optional)	
detected	Outport	boolean	FPGA Data Capture	Trigger and Data	
data_in	Test point	sfix16_En14	FPGA Data Capture	Trigger and Data	
valid_in	Test point	boolean	FPGA Data Capture	Trigger and Data	
mag_sq_out	Test point	sfix18_En11	FPGA Data Capture	Trigger and Data	

Register Address Mapping

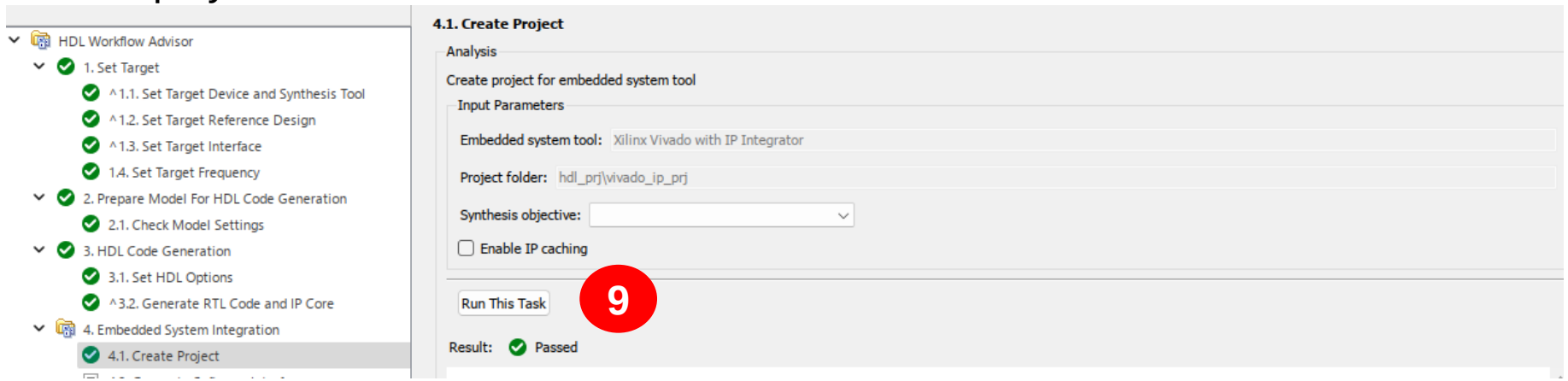
The following AXI4-Lite bus accessible registers were generated for this IP core:

Register Name	Address Offset	Description
IPCore_Reset	0x0	write 0x1 to bit 0 to reset IP core
IPCore_Enable	0x4	enabled (by default) when bit 0 is 0x1
IPCore_PacketSize_AXI4_Stream_Master	0x8	Packet size for AXI4-Stream Master interface, the default value is 1024. The TLAST output signal of the AXI4-Stream Master interface is generated based on the packet size.
IPCore_Timestamp	0xC	contains unique IP timestamp (yymmddHHMM): 2405291005
threshold_Data	0x100	data register for Inport threshold
output_length_Data	0x104	data register for Inport output_length

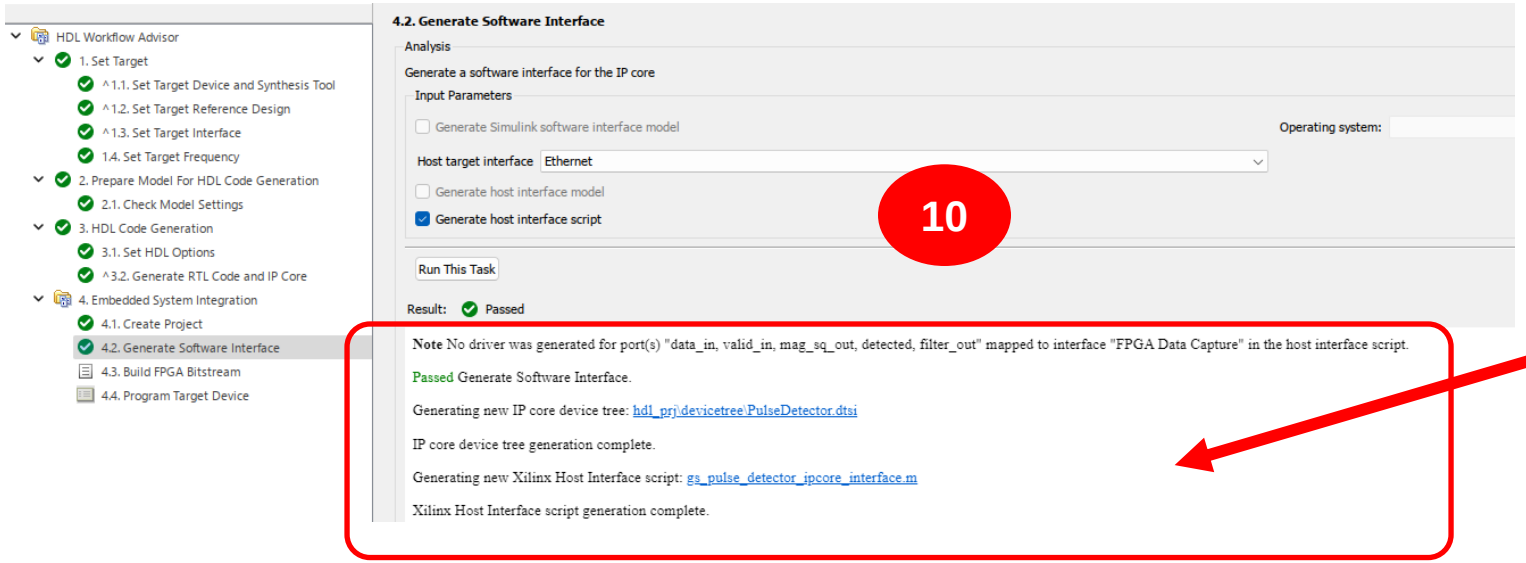
Following are the AXI4 slave Base address and Master address space specified in the reference design: **Default system with AXI4-Stream interface.**

IP Core gen workflow with FPGA Data Capture IP

9. In Step 4.1 of **HDL Workflow Advisor**, press **Run This Task** button. At end of this task, Result should display as Passed



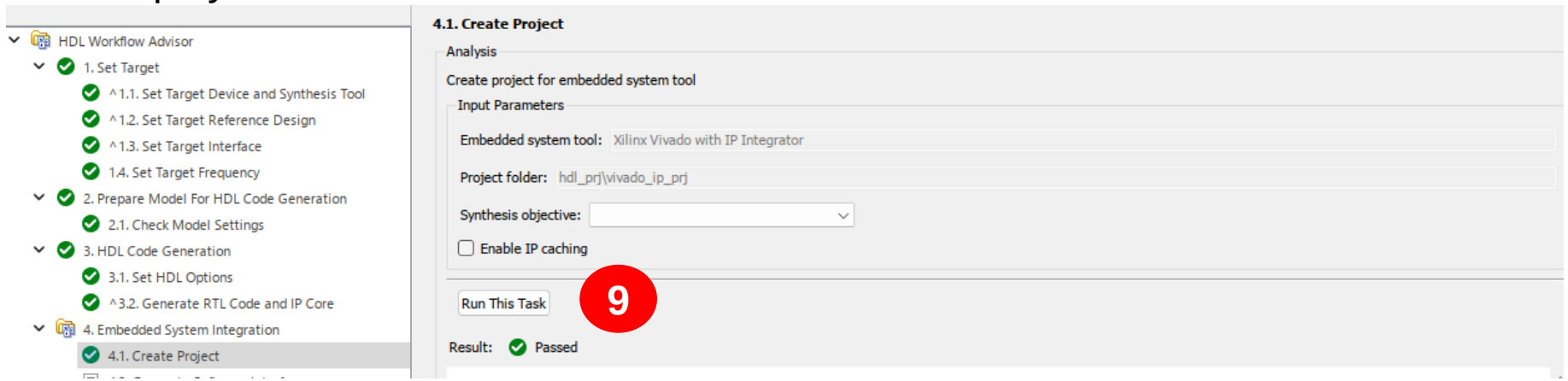
10. In Step 4.2 of HDL Workflow Advisor, set as below and click **Run This Task** button



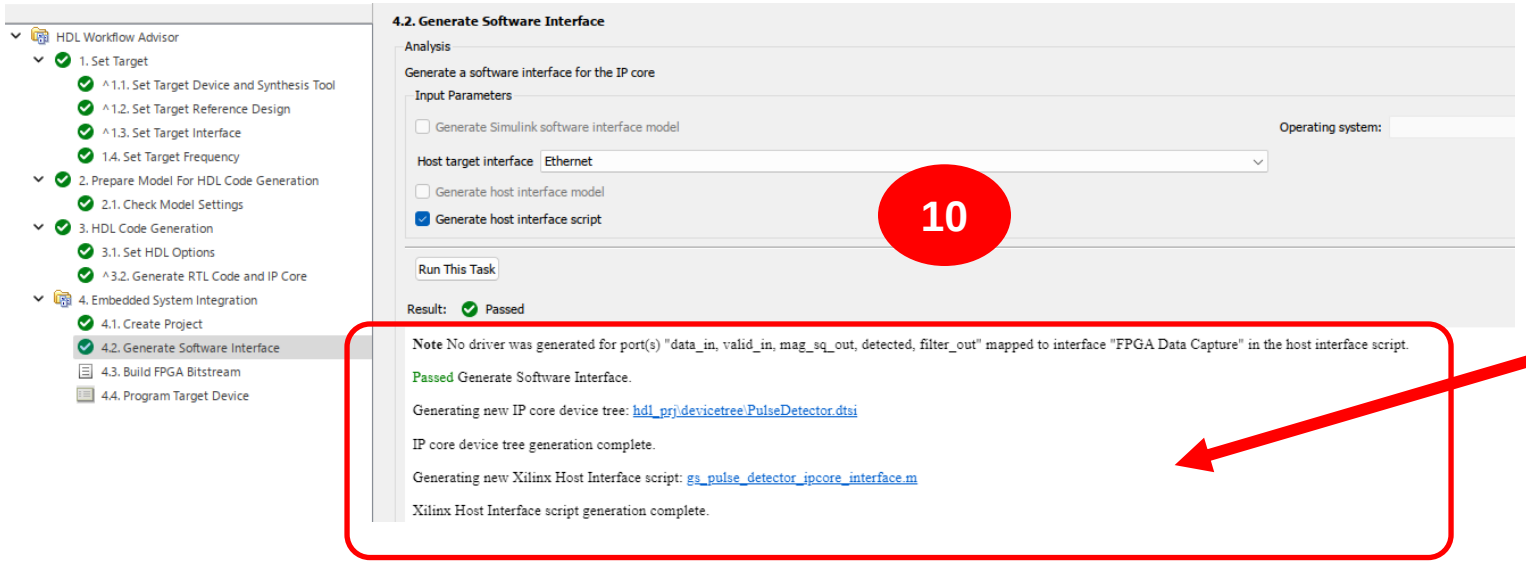
Generated
- IP Core Device Tree
- Xilinx Host Interface script

IP Core gen workflow with FPGA Data Capture IP

9. In Step 4.1 of **HDL Workflow Advisor**, press **Run This Task** button. At end of this task, Result should display as Passed



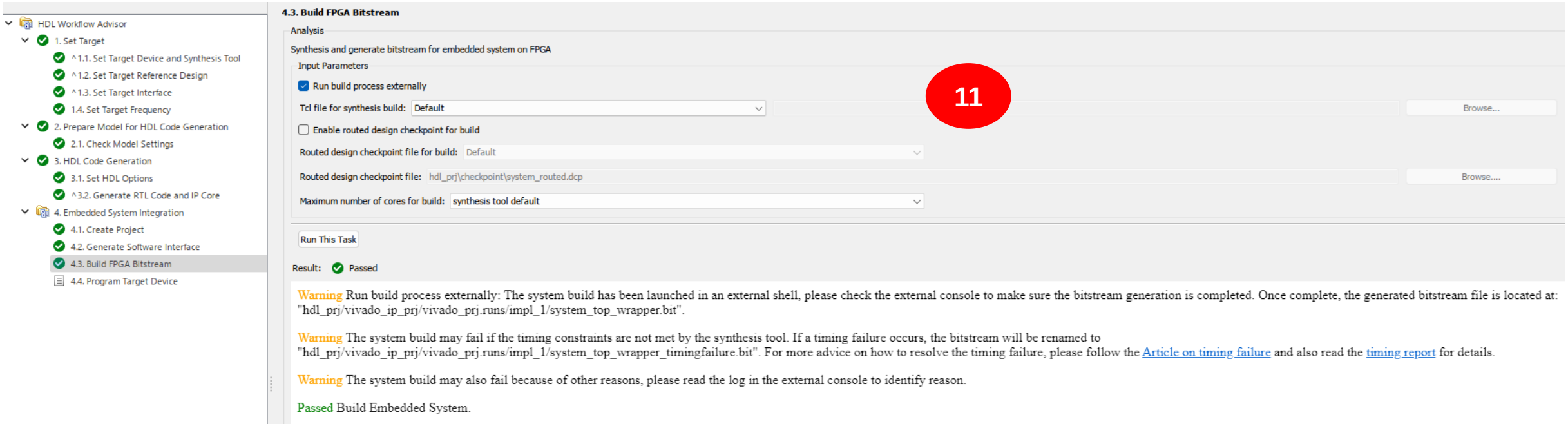
10. In Step 4.2 of HDL Workflow Advisor, set as below and click **Run This Task** button



Generated
- IP Core Device Tree
- Xilinx Host Interface script

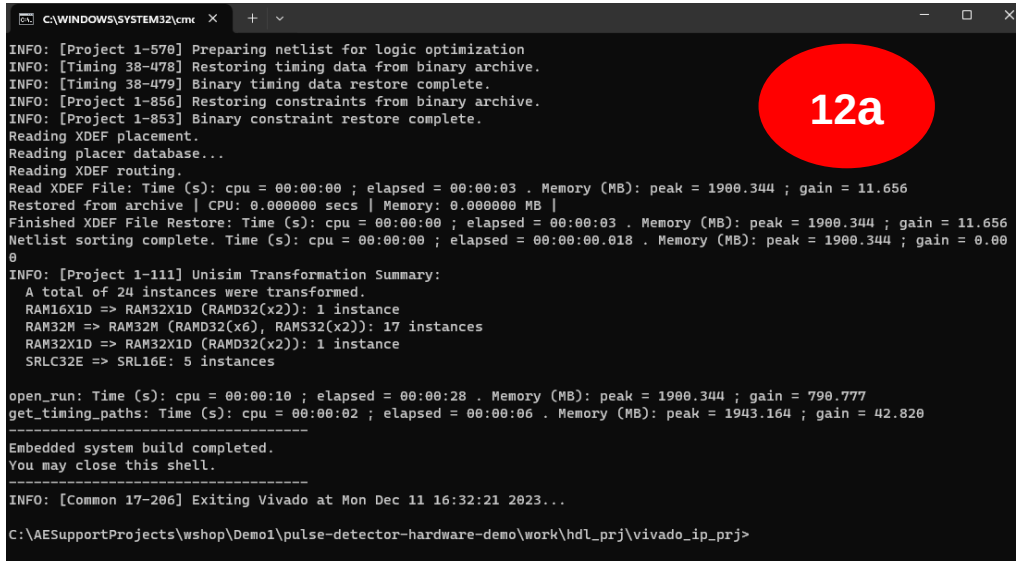
IP Core gen workflow with FPGA Data Capture IP

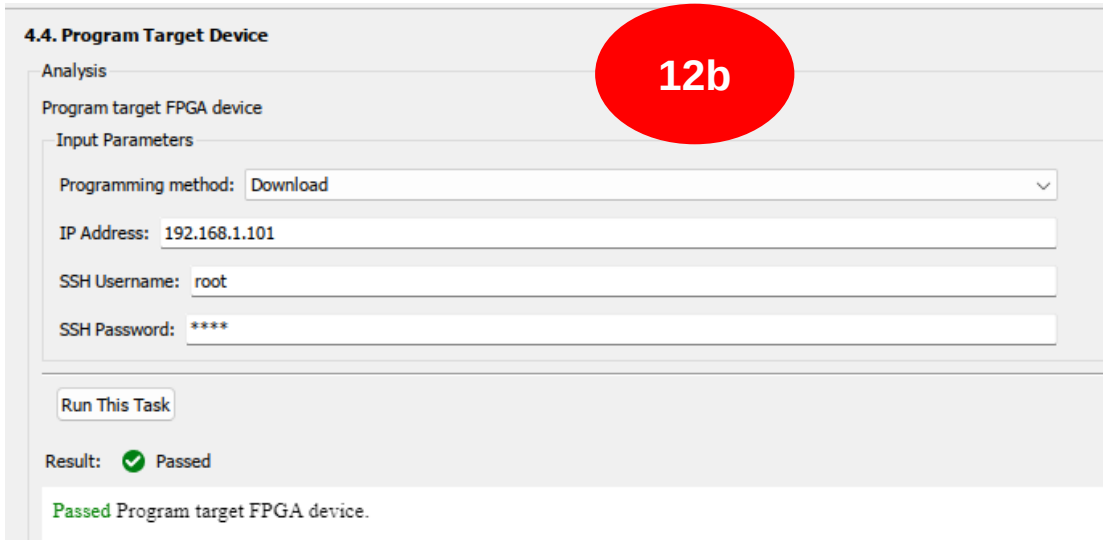
11. In Step 4.3 of HDL Workflow Advisor, set as below and click **Run This Task** button to generate FPGA Bitstream.



IP Core gen workflow with FPGA Data Capture IP

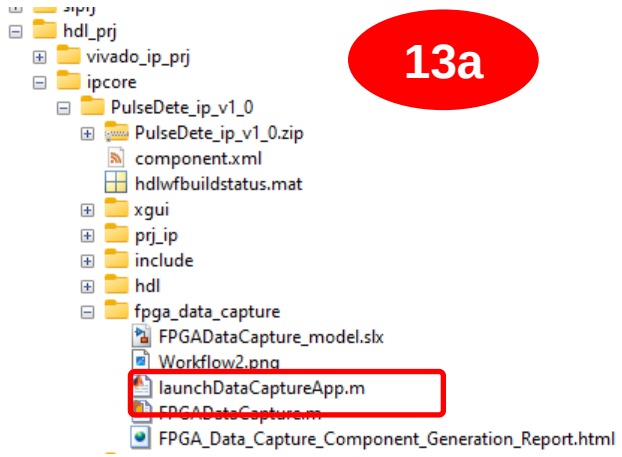
12. Build Process is launched externally
    a) Check Bitstream is generated successfully.
    b) Connect ZedBoard through Ethernet cable to your PC. Run Step 4.4, Program Target device.
    c) You can also export a script from HDL workflow advisor to do all these steps through command line.
       Please see scripts/hdlworkflow.m





Demo1 - FDC Workflow using HDL Workflow Advisor

13. Edit the generated launchDataCaptureApp.m
    a) The launchDataCaptureApp is created under work folder as shown in 13a **Add the generated hdl_prj** folder to the MATLAB path.
    b) At MATLAB command prompt type "edit launchDataCaptureApp".
    This will open the script launchDataCaptureApp.m
    a) Add line 6 to enable [nonblocking Capture Mode]



13b

```
1  % Automatically generated script to laun FPGA Data Captu
2  % Instantiate FPGA Data Capture System o t
3  if ~exist('fpgadc_obj','var') || ~isa(fpgadc_obj,'FPGAData
4      fpgadc_obj = FPGADataCapture;
5  end
6  fpgadc_obj.CaptureMode = 'nonblocking';
7  fpgadc_obj.launchApp;
8
```

Demo1 - FDC Workflow using HDL Workflow Advisor

14. Review the provided test script

    a) pulse_detector_ipcore_hw_demo.mlx

Testing HDL Pulse Detector on Hardware

Configure Parameters

```
pulse_detector_init;

% Board IP address
IPAddress = '192.168.1.101';

% Length of test signal
testSignalLength = 1500;

% Test signal noise scale
noiseScale = 1e-2;

% Number of samples to output after pulse detection
outputLength = 250;

% Detection threshold
threshold = 0.8;

% Location of pulse in test signal
PulseLoc = 500;

% Generate test signal
testSignal = generate_test_signal(testSignalLength,noiseScale,PulseLoc);
```

14a

Set up FPGA I/O

```
hw = xilinxsoc(IPAddress,'root','root');
hFPGA = fpga(hw);

WriteFrameLength = testSignalLength;
ReadFrameLength = outputLength;

pulse_detector_ipcore_hw_setup(hFPGA,WriteFrameLength,ReadFrameLength);
```

Write to registers, send/receive test vectors

```
writePort(hFPGA, "threshold", threshold);
writePort(hFPGA, "output_length", outputLength);

writePort(hFPGA, "stream_in_data", testSignal);
[captureData, captureValid] = readPort(hFPGA, "stream_out_data");
```

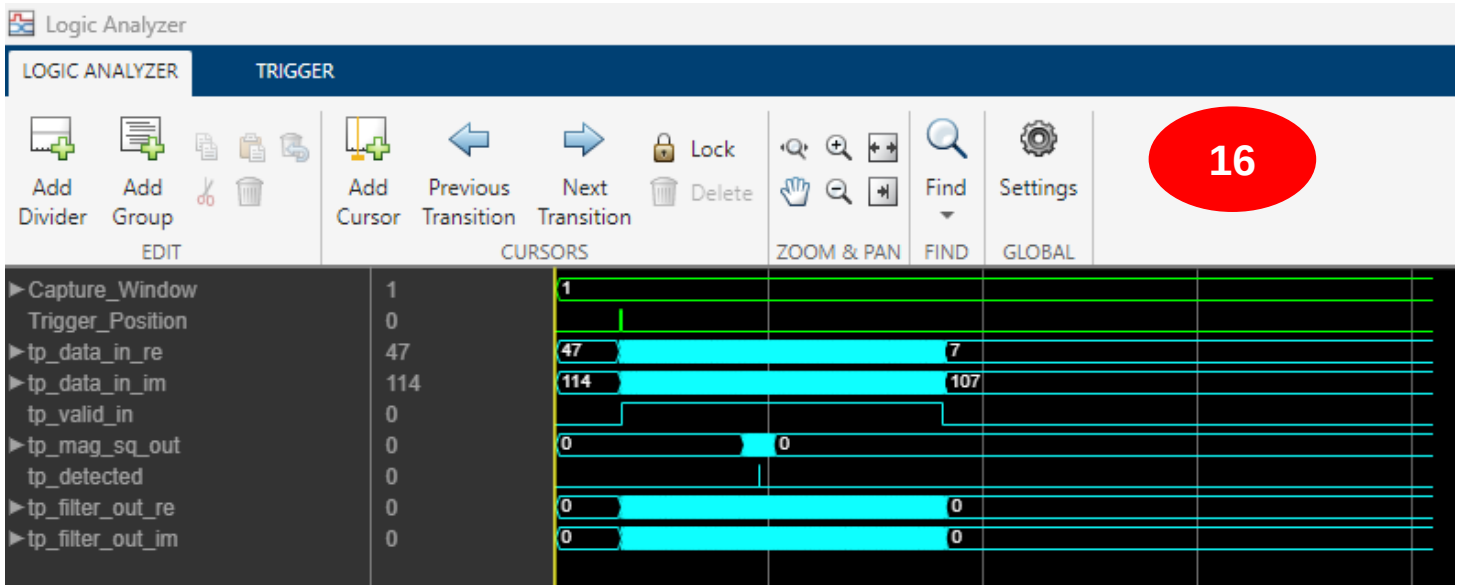
Display results

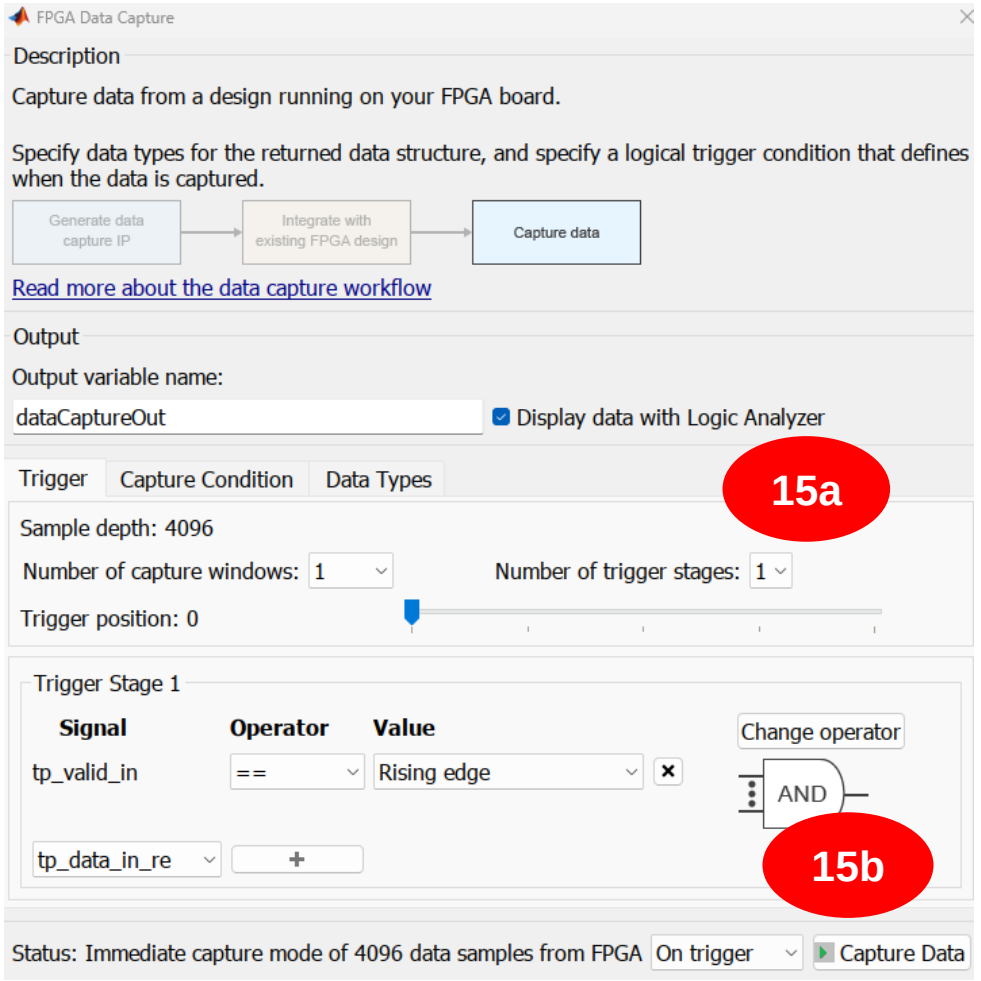
```
if captureValid
    figure(100); clf;
```

Demo1 - FDC Workflow using HDL Workflow Advisor

15. Launch FPGA data capture app by typing **launchDataCaptureApp** in MATLAB command Window
    a) Set trigger to tp_valid_in rising edge
    b) Click on Capture Data

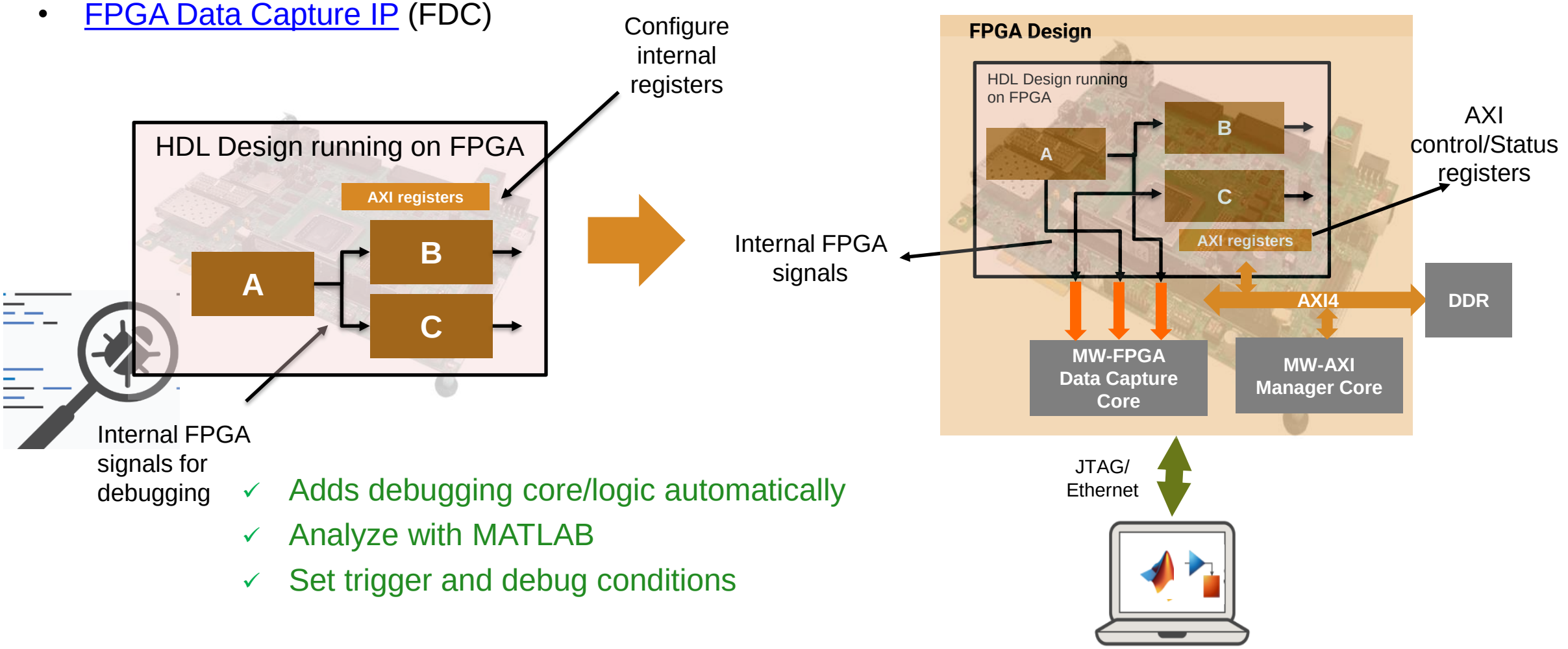
16. Run the test script in 14a. The Logic Analyzer window will automatically open to display the captured Test-point signals.





Dynamic Configure and Debug with AXI Manager and Data Capture

- AXI Manager IP (AXIM)
- FPGA Data Capture IP (FDC)



- ✓ Adds debugging core/logic automatically
- ✓ Analyze with MATLAB
- ✓ Set trigger and debug conditions

Demo2 – Use AXI Manager (AXIM) and Data Capture IP with RTL Design

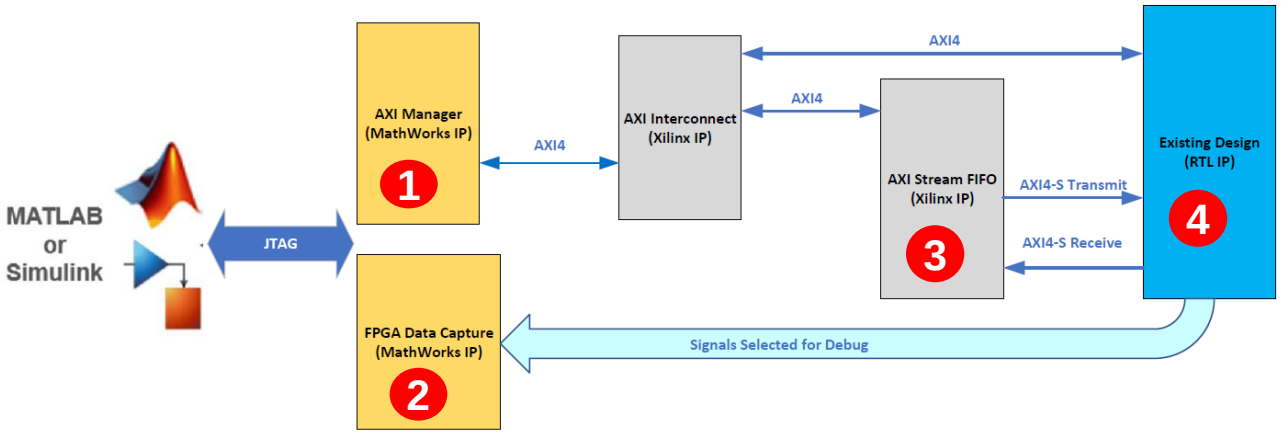


Fig – Data flow between MATLAB and Design on FPGA

Workflow

1. AXI Manager IP from MathWorks is instantiated in the block design to provide Stimulus. The IP also responds to read and write commands from MATLAB or Simulink, over JTAG, PCI Express® (PCIe), or Ethernet cable.
2. Data Capture IP from MathWorks is instantiated in the block design to capture and observe signals while the design is running on FPGA
3. AXI Stream FIFO (AXI-SF), an AMD IP is used for converting memory mapped to AXI stream transactions and vice-versa between AXI Manager and existing RTL design.
4. The Pulse detector IP is used as an example of existing RTL design

AMD Vivado IP integrator is used to build the block design.

Note - AXIM IP is added to Vivado project using setupAXIManagerForVivado(projectname)

See link on how to add AXIM path to Vivado project

Demo2 – Use AXI Manager and Data Capture IP with RTL Design

Workflow (Contd.)

A test program is written in a MATLAB live script that uses AXI Manager to

- Configure registers in Pulse detector and AXI-SF.
- Write data to the transmit FIFO in AXI-SF, which is then sent to AXIS input of Pulse detector
- Pulse detector AXIS output is collected in receive FIFO in AXI-SF which is then read using AXI manager.
- Snapshot from Test Program is pasted below

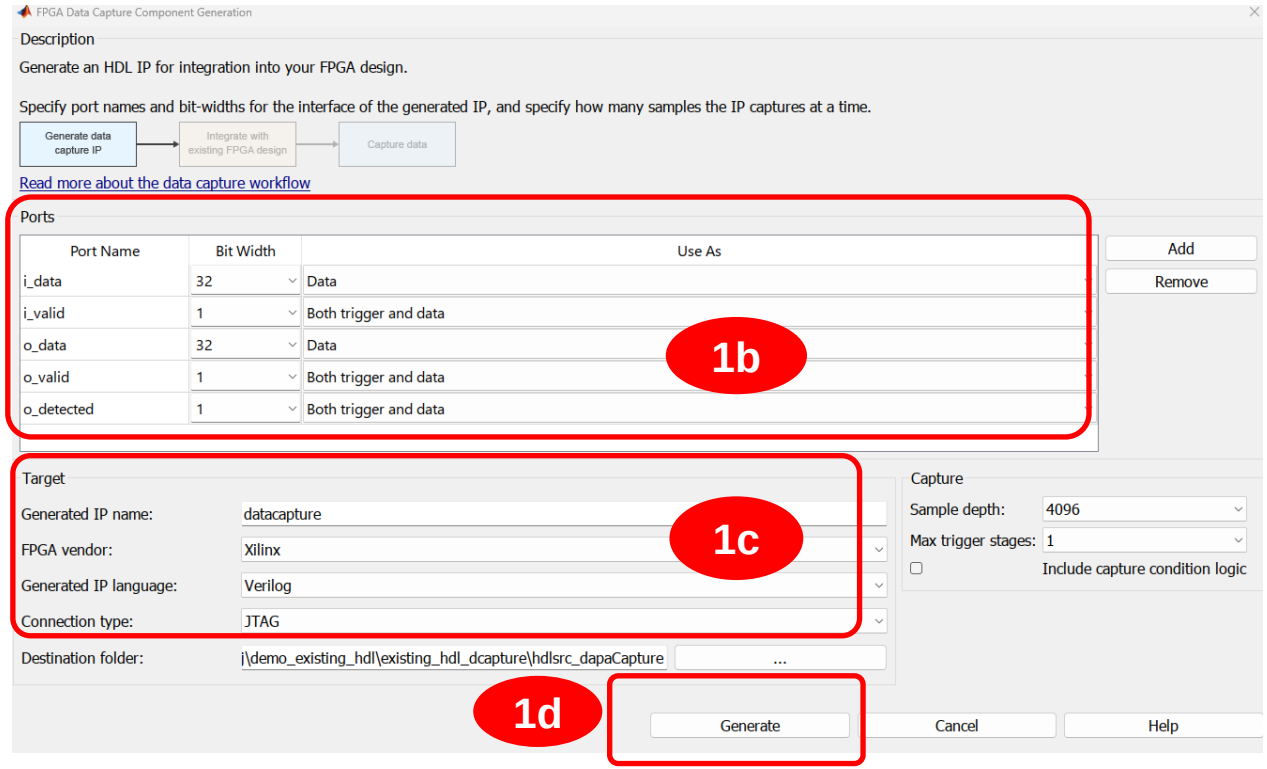
```
% Define Register Addresses here
REG_PD_BASE = 0x43c00000;
REG_PD_TH = REG_PD_BASE + 0x00100 ; % Pulsedetect Threshold register
REG_PD_OLEN = REG_PD_BASE + 0x00104 ; % Pulsedetect Output Length
% AXIS Stream FIFO
REG_AXISF_Base = 0x43C10000;
REG_AXSIF_IS = REG_AXISF_Base;
REG_AXISF_TDFR = REG_AXISF_Base + 0x000008; %Transmit Data FIFO Reset (TDFR) Write
REG_AXISF_TDFV = REG_AXISF_Base + 0x00000C; %Transmit Data FIFO Vacancy (TDFV) Read
REG_AXISF_TDFD = REG_AXISF_Base + 0x000010; %Transmit Data FIFO 32-bit Wide Data Write Port
REG_AXISF_TLR  = REG_AXISF_Base + 0x000014; %Transmit Length Register (TLR)
REG_AXISF_RDFR = REG_AXISF_Base + 0x000018; %Receive Data FIFO reset (RDFR)
REG_AXISF_RDFO  = REG_AXISF_Base + 0x00001c; %Receive Data FIFO Occupancy(RDFO)
REG_AXISF_RDFD  = REG_AXISF_Base + 0x000020; %Receive Data FIFO 32-bit Wide Data Read Port
REG_AXISF_RLR   = REG_AXISF_Base + 0x000024; %Receive Length Register
REG_AXISF_SRR   = REG_AXISF_Base + 0x000028; %AXI4-Stream Reset (SRR)
```

```
%Simple test block
 h = aximanager('Xilinx');
 writememory(h,REG_PD_TH,fi(0.2,0,18,11));
 rdata = readmemory(h,REG_PD_TH,1,'OutputDataType',numerictype(0,18,11));
 disp(rdata);
 release(h);
```

Demo2 – Use AXI Manager and Data Capture IP with RTL design

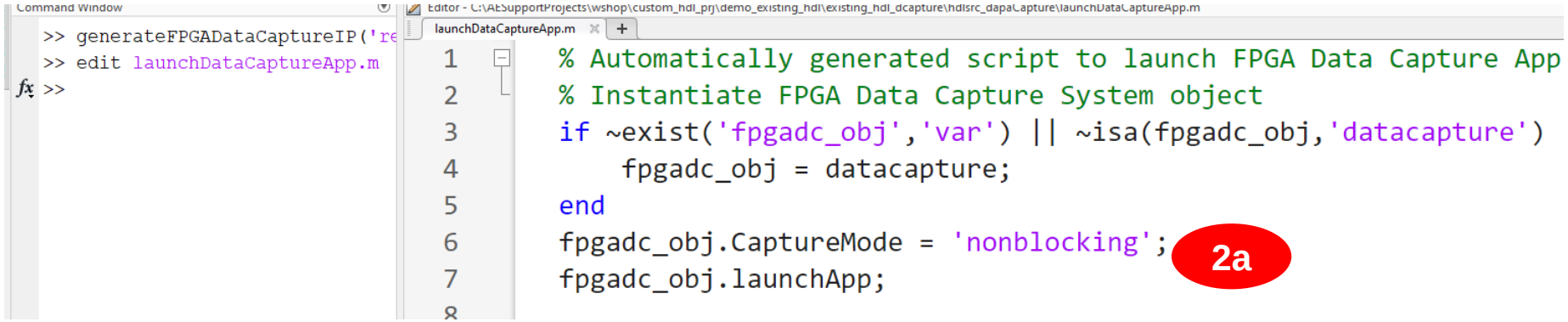
Steps

1. In this step, we will demonstrate, how to generate Data Capture IP for Integration into your FPGA design
   a. At the MATLAB command prompt, type **generateFPGADataCaptureIP**
   b. Add Port Name, Bit Width and Trigger/Data Information in Ports Table for Signals that you want to be captured for debug. Here we are capturing Input and Output Data/Valid to/from Pulse Detector IP.
   c. Add connection type (JTAG for this demo) , Destination folder where the Verilog IP code will be generated
   d. Press Generate button. **Add the generated folder to MATLAB path**
   e. You can include the generated IP folder in AMD Vivado IP integrator

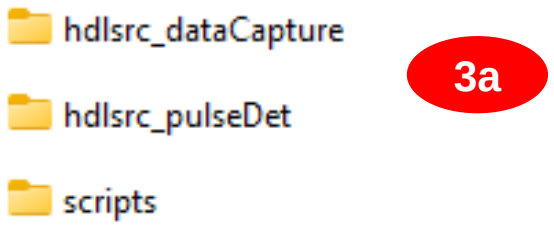


Demo2 – Include FPGA Data Capture IP in Existing HDL Design

2. At MATLAB Command prompt type, **edit launchDataCaptureApp.m**
   a. Add line 6 to enable nonblocking Capture Mode
   b. For more information on Nonblocking vs Blocking mode, please go through this Link



3. For this demo, You can download **fdc_existing_hdl** to your area and unzip it
   a. It consists of subfolders with RTL for FPGA Data Capture IP, Pulse detect and FPGA project tcl scripts



Demo2 – Use AXI Manager and Data Capture IP with RTL Design

4. In MATLAB change directory to **fdc_existing_hdl**
   - Add **scripts and hdlsrc_dataCapture** folders to the MATLAB path,

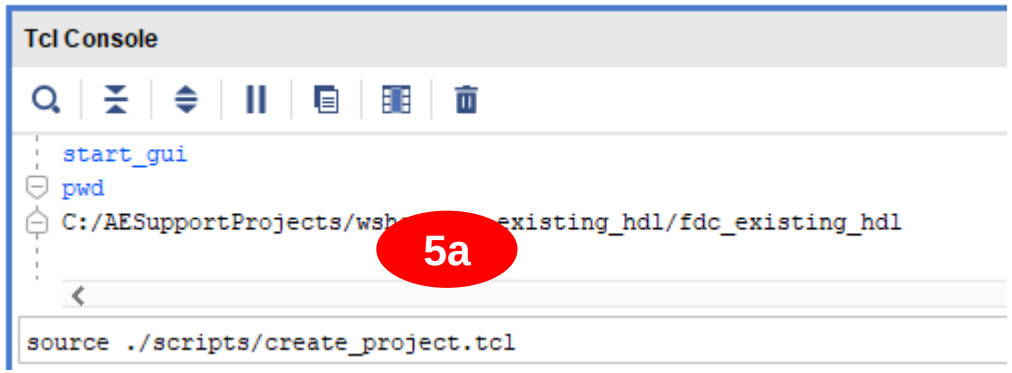
     >> addpath('scripts','hdlsrc_dataCapture')
   - In **script/setup. tcl ,** set **matlab_path** variable value to your MATLAB SupportPackage root folder

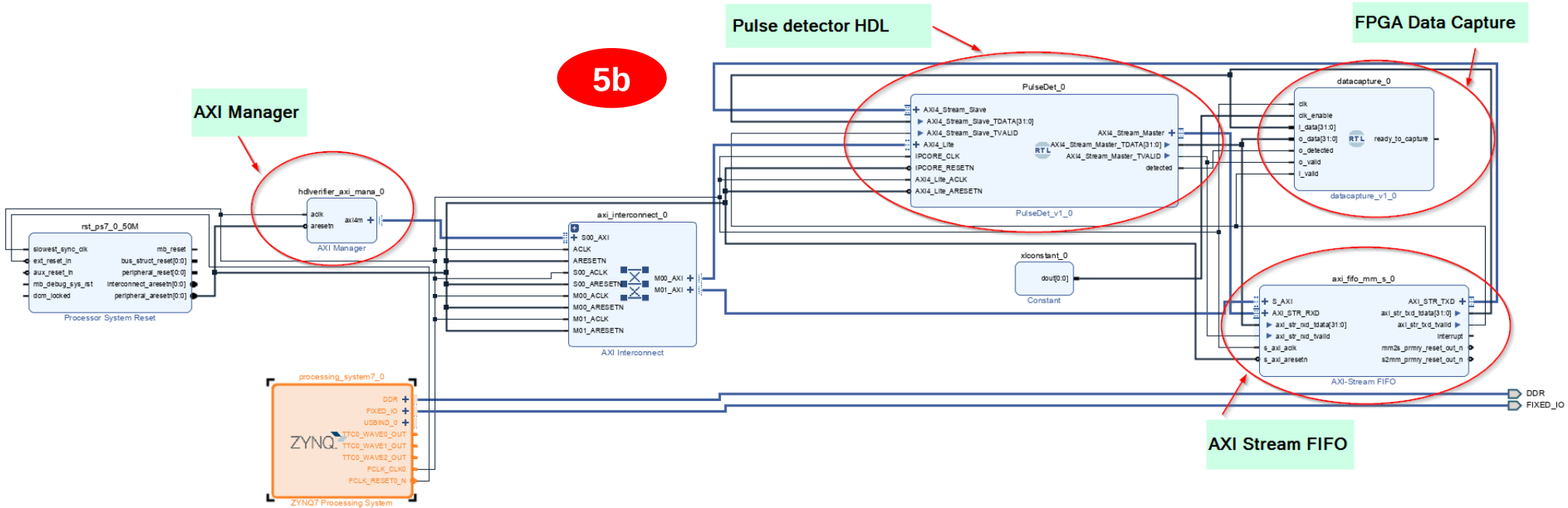
     Note - Read instructions in setup.tcl
   - Check that the FPGA board is powered-on.
   - Connect Micro-USB cable to the board to enable programming over JTAG.
   - Choose one of the following options
     - **Run runme.m.**
       - This script will launch Vivado in batch mode, generate Bitfile and program the FPGA.
       - Skip steps 5-8 and go to step 9 for debugging using Data Capture App.
     - **Run program_soln.m**
       - This script programs FPGA with prebuilt Bit file in solution folder.
       - Skip steps 5-8 and go to step 9 for debugging using Data Capture App
     - **Run Steps 5-8 using tcl scripts in Vivado**
       - Create Vivado® project, generate and program bitfile.

Demo2 – Use AXI Manager and Data Capture IP with RTL Design

5. In MATLAB type !vivado& and this will launch Vivado
   a. In Vivado, source ./scripts/create_project.tcl
      - It will create the Vivado Project and block design.
   b. Review the created Block design
   c. FDC & AXI manager have already been added to the design
      - See link on how to add AXIM to path of Vivado® project
      - ***setupAXIManagerforVivado(projectName)***

        adds the AXI Manage IP folder to the path of the Vivado® project, projectName

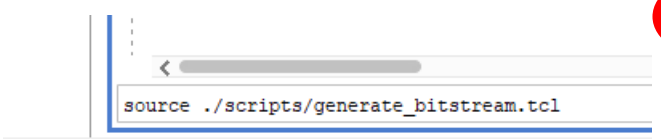




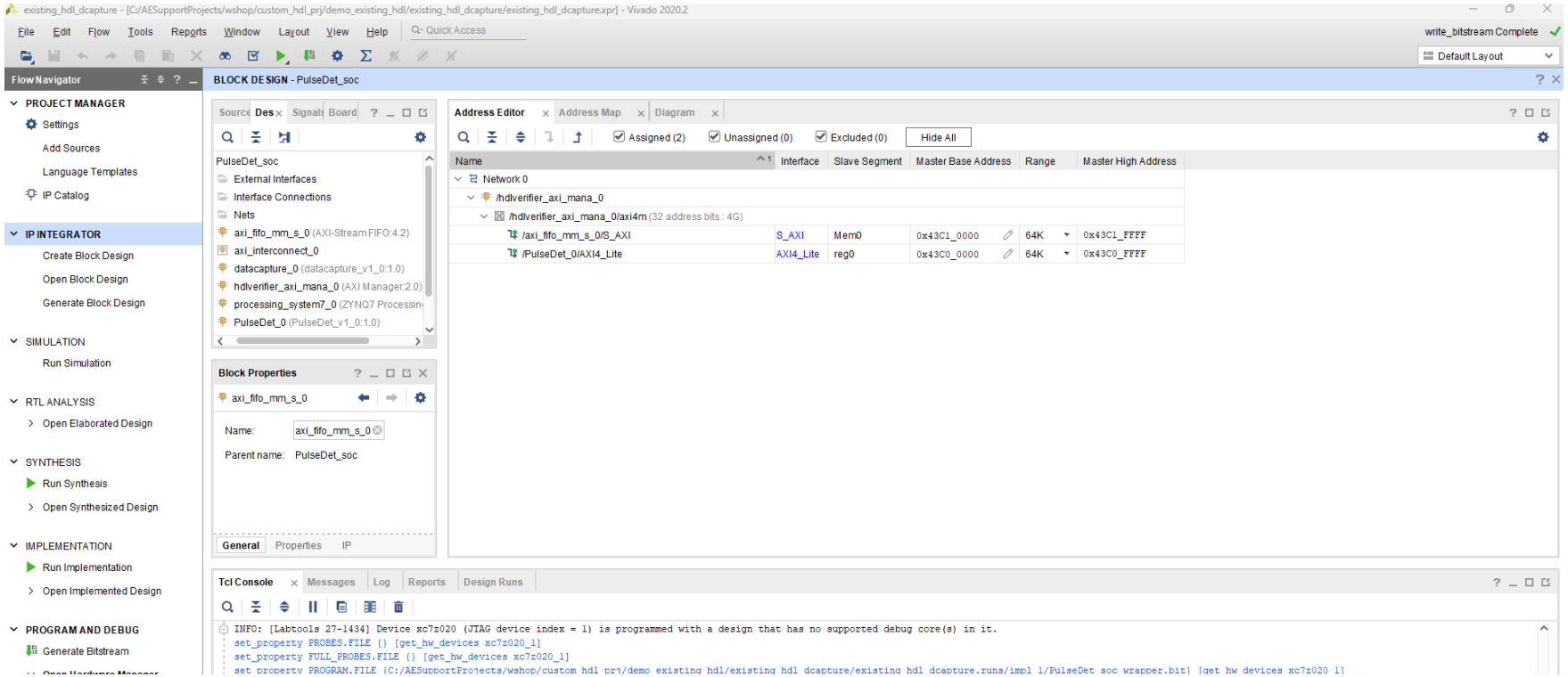
Demo2 – Use AXI Manager and Data Capture IP with RTL Design

6. Run synthesis, implementation and generate bitstream
   a. source ./scripts/generate_bitstream.tcl
      This script will run synthesis, implementation and generate the bitstream

   b. Check bitstream is generated successfully



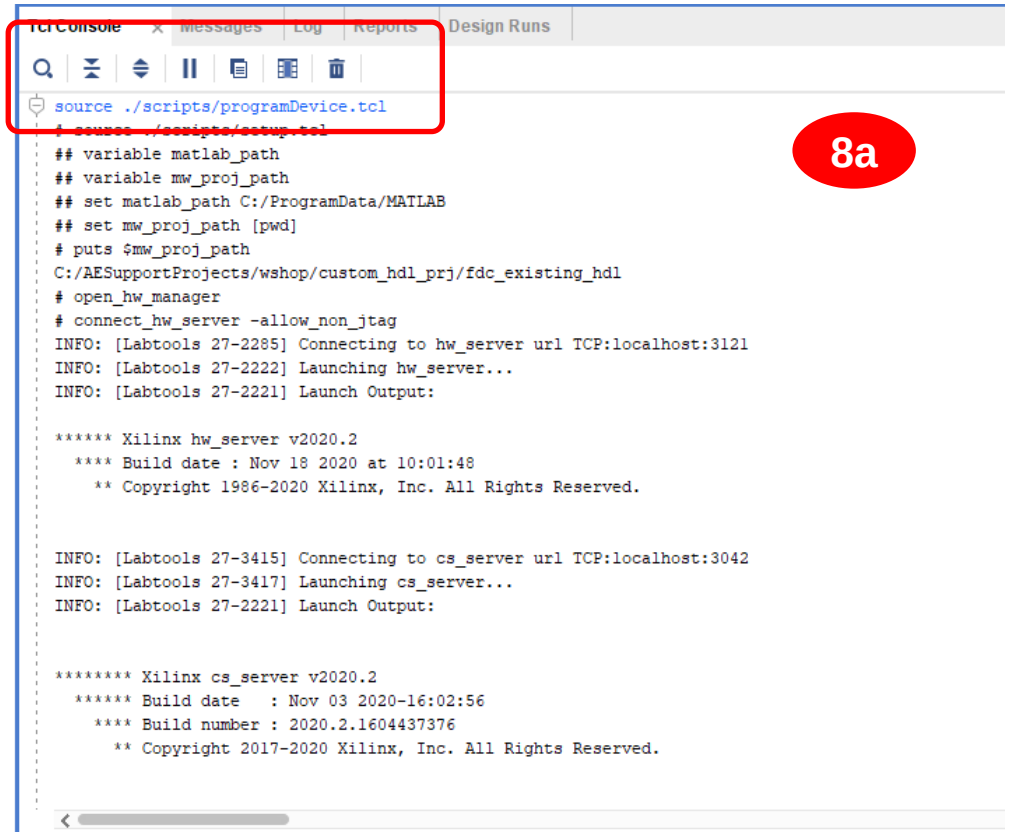
6a



6b

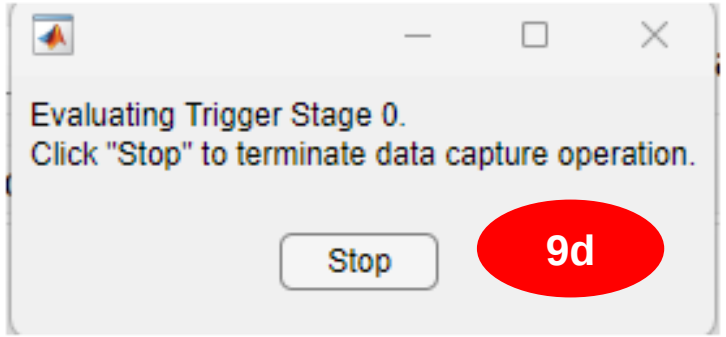
Demo2 – Use AXI Manager and Data Capture IP with RTL Design

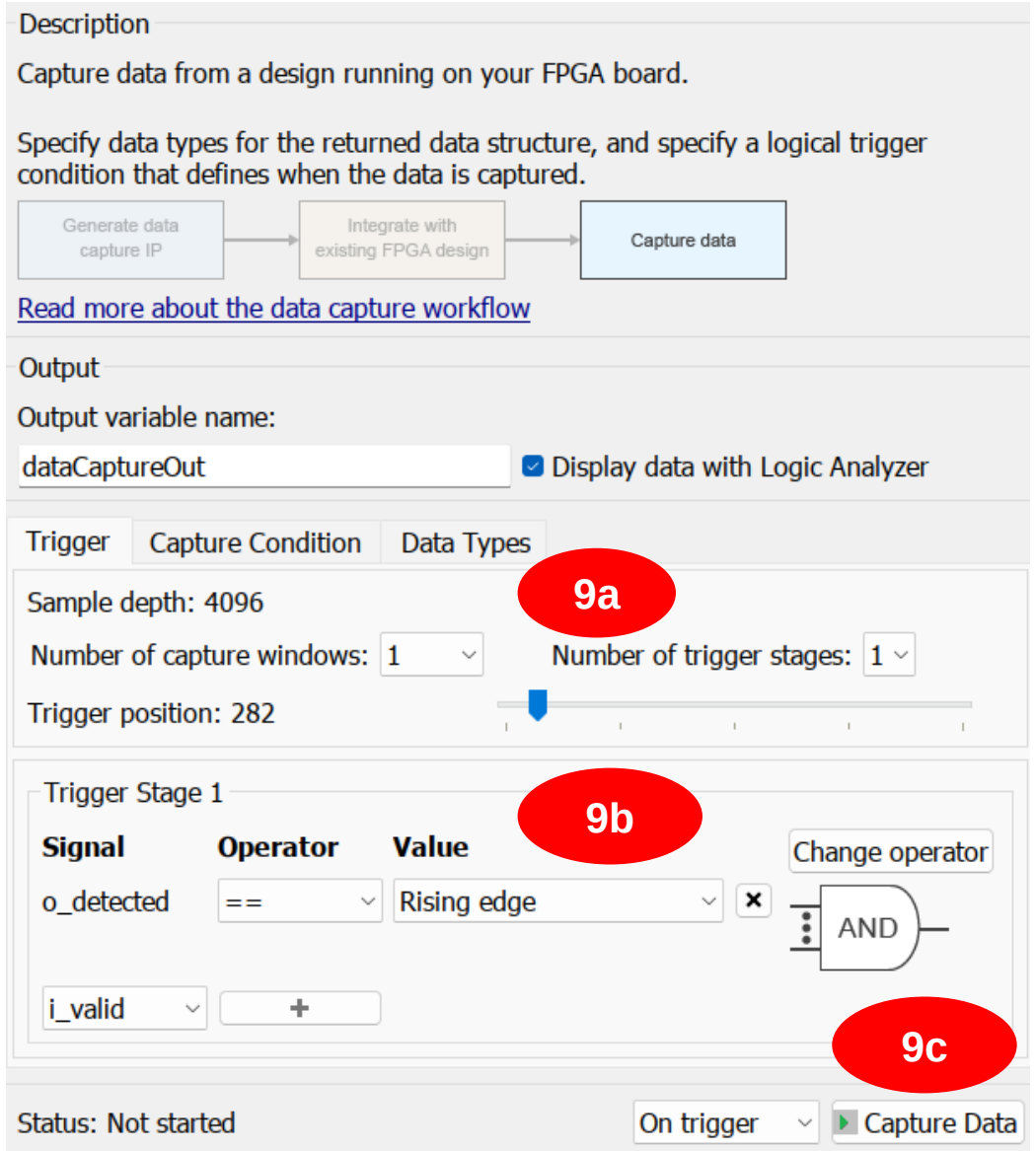
7. Check that USB cable is connected between ZedBoard and PC, and the board is powered up.
8. Program Device using generated Bit stream. This needs to be done every time you power cycle the board.
   a. source ./scripts/programDevice.tcl which will program the device. Check there are no errors.



Demo2 – Use AXI Manager and Data Capture IP with RTL Design

9. In MATLAB type **launchDataCaptureApp**
   a) Once App is open, Set Trigger position
   b) Set Trigger condition on **o_detected** rising edge
   c) Select On trigger and Press Capture Data
   d) This will launch a small window waiting for trigger condition to occur.





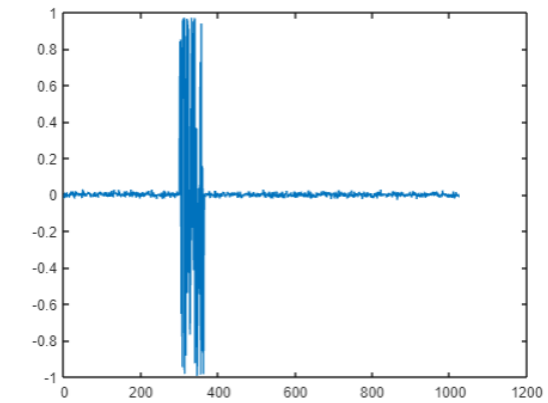
Demo2 – Use AXI Manager and Data Capture IP with RTL Design

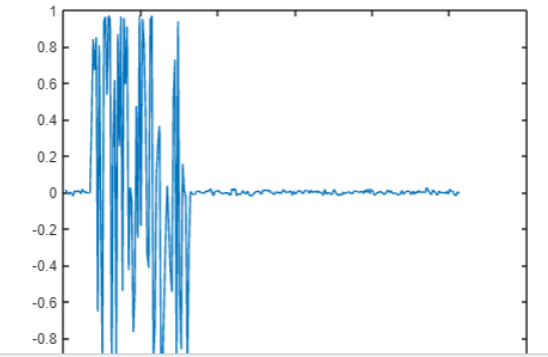
10. Open the provided test script scripts/test_custom_hdl.mlx. Run the script.
    - This will send data to FPGA over JTAG
    - AXI Manager is used to configure registers and send input stimulus to Pulse detect IP and get the output.

```
h = aximanager('Xilinx');

%===============
% PULSE DETECT
% Write to Threshold Register
writememory(h,REG_PD_TH,fi(0.2,0,18,11));
% Write to output length Register
writememory(h,REG_PD_OLEN,out_len);

%===============
% AXIS FIFO

% Reset AXI4 Stream
writememory(h,REG_AXISF_SRR,0xa5);

%Reset the Transmit FIFO
writememory(h,REG_AXISF_TDFR,0xa5);
writememory(h,REG_AXISF_TDFR,0x0);

%Reset the RX FIFO
writememory(h,REG_AXISF_RDFR,0xa5);
writememory(h,REG_AXISF_RDFR,0x0);

% Clear Interrupt Status
writememory(h,REG_AXSIF_IS,0xFFFFFFFF);

% Write Pulse to FIFO
writememory(h,REG_AXISF_TDFD,fi_testSignal,'BurstType','Fixed');


%Write Pulse Length to FIFO
writememory(h,REG_AXISF_TLR,testSignalLength*4); % Number of bytes

% Read Pulse length
rdata = readmemory(h,REG_AXISF_RLR,1);
fprintf("\n Number of Bytes in Receive FIFO = %d \n",int32(rdata));

% Get the Received Pulse
pulse_rx = readmemory(h, REG_AXISF_RDFD,out_len,'BurstType','Fixed','OutputDataType',numerictype(1,16,14));

plot(real(testSignal))
plot(pulse_rx);
```

```
release(h);
```

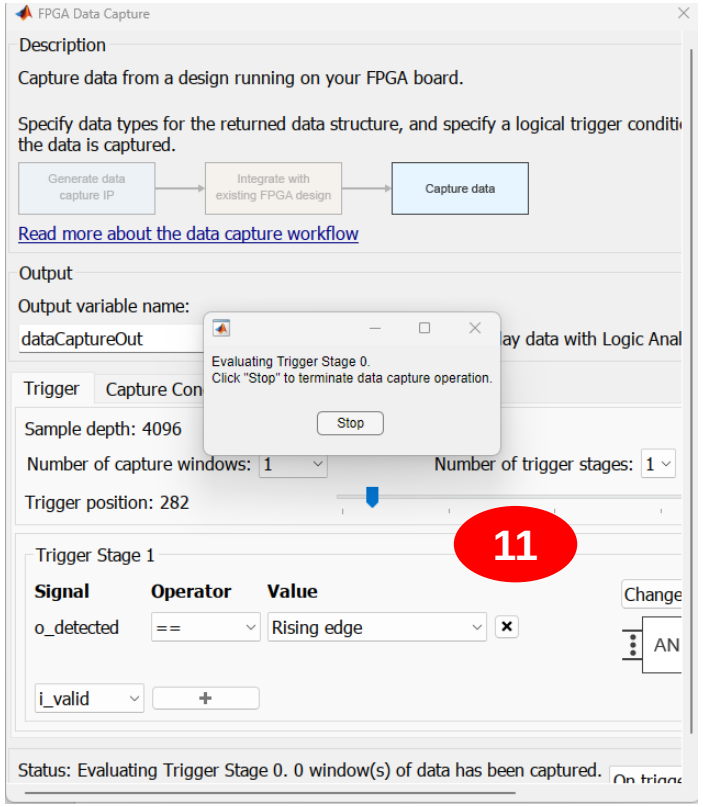
```
Number of Bytes in Receive FIFO = 1024
```

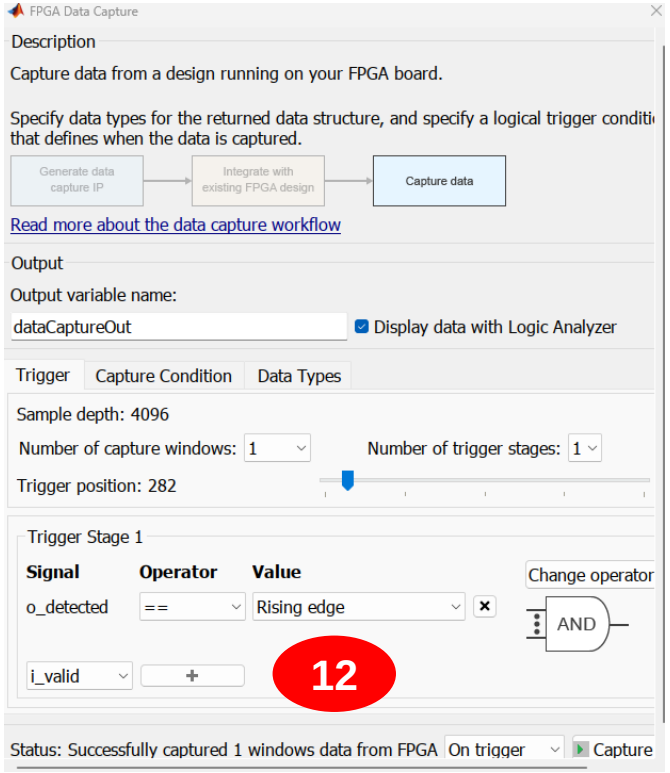


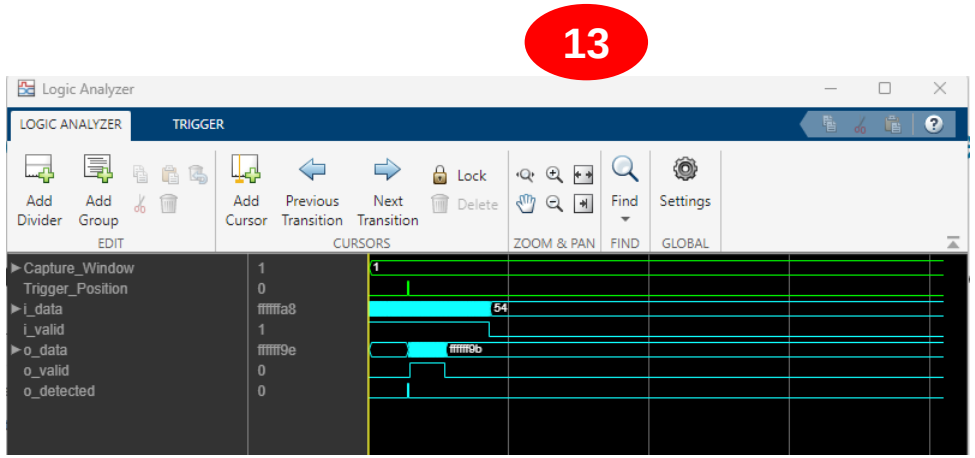


Demo2 – Use AXI Manager and Data Capture IP with RTL Design

11. Running the test script sets the trigger condition that FPGA Data capture App is waiting on.
12. The app shows successful Data capture from FPGA based on Trigger condition (o_detected rising edge)
13. Logic Analyzer displays the captured Data

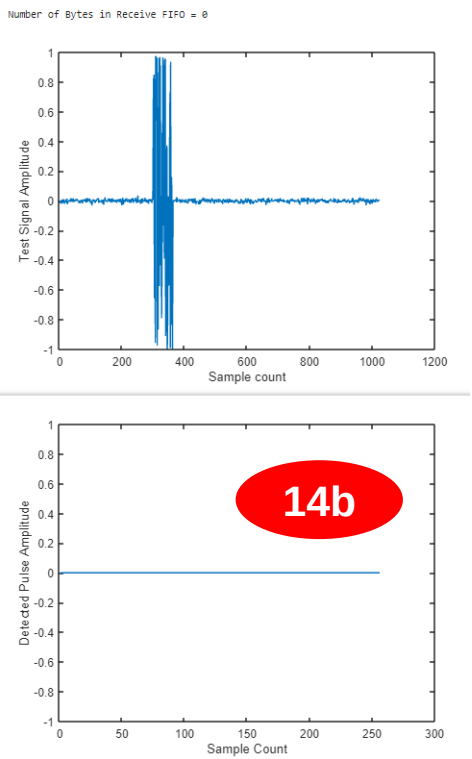


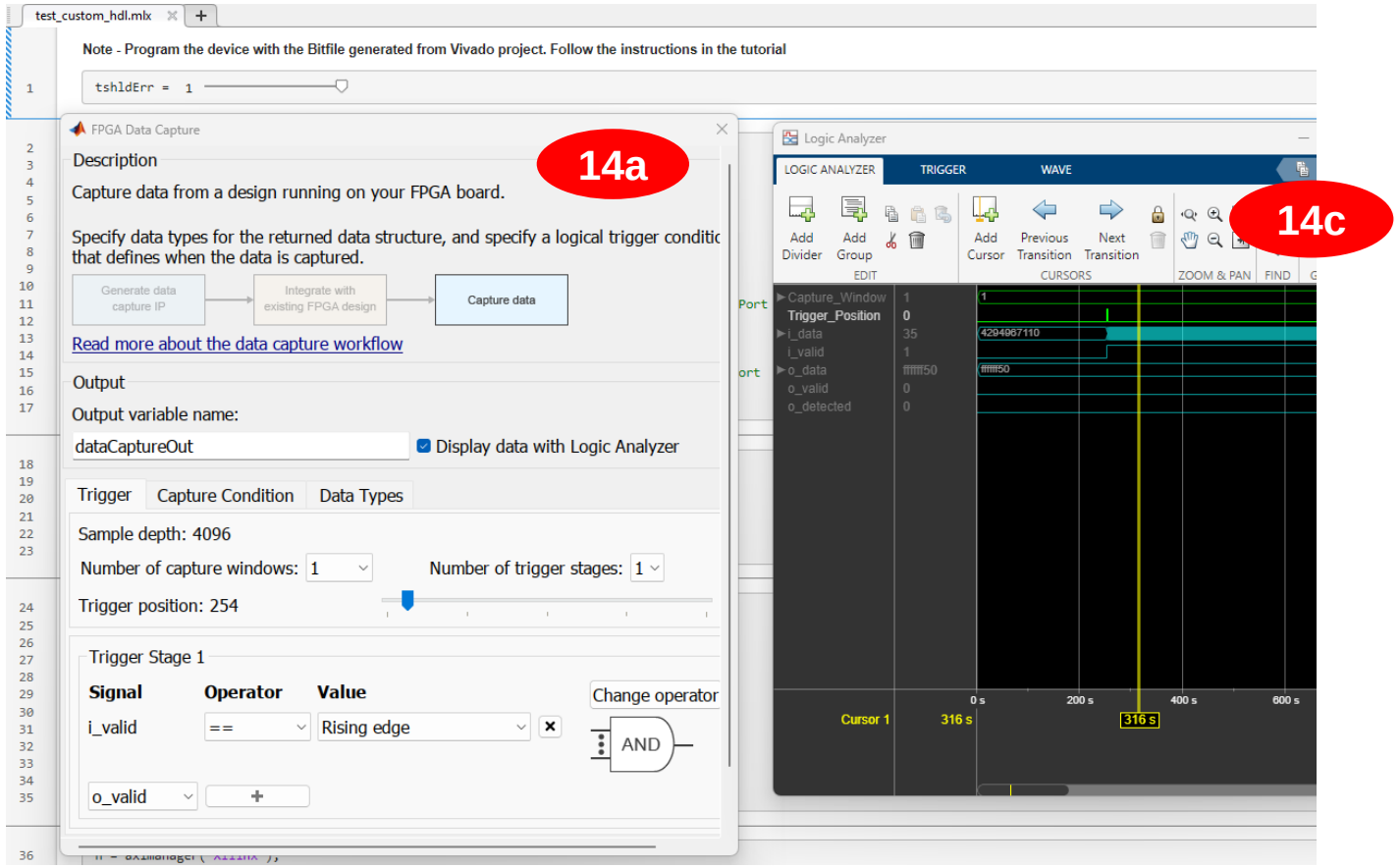




Demo2 - Debug invalid Threshold programming value.

14. Error Scenario
    a. Set the numeric slider tshldErr =1 in test script. In FPGA data capture app, change the trigger to i_valid rising edge, and run capture data.
    b. Run the test script. The pulse is not detected as the threshold value is programmed too high.
    c. Using Logic analyzer, you can view that o_detected is not asserted





Comparing FPGA-in-the-Loop and FPGA Data Capture

	FPGA-in-the-Loop	FPGA Data Capture
Execution speed	Does not run at speed - clock rate limited by host/board communication	Design runs at-speed
Connectivity	JTAG / Ethernet / PCI-Express	JTAG / Ethernet
Inputs to DUT	Inputs to board must originate in MATLAB / Simulink	Board inputs allowed
Signal visibility	Outputs of DUT only	Any signal specified for capture
Data availability	Output data streamed immediately to MATLAB / Simulink	Probed data returned to MATLAB or Simulink after triggering
Data resolution	Data rate limited by host/board communication	Data may be saved up to clock rates

HDL Verifier Support Packages for AMD, Altera and Microchip Boards

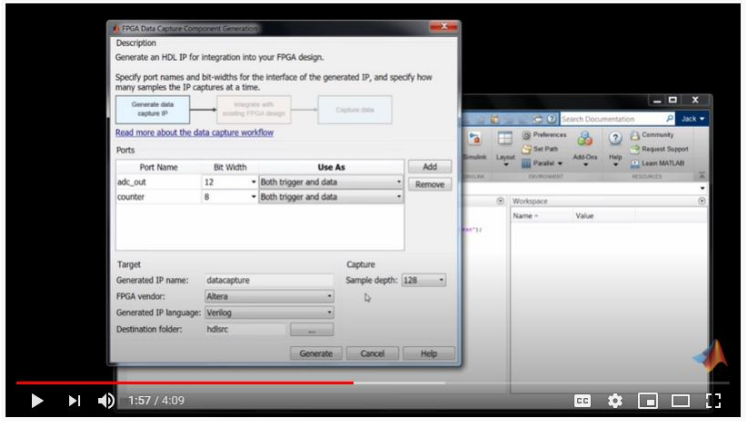
- Data Capture & AXI Master over JTAG supports:
  - Same hardware as FPGA-in-the-Loop
- AXI Master over Ethernet supports:
  - AMD ZC706, ZedBoard, AMD KC705, Arrow MAX®10 DECA
- AXI Master over PCIe supports:
  - AMD KCU116, Altera Arria® 10 GX

Requires:

- HDL Verifier

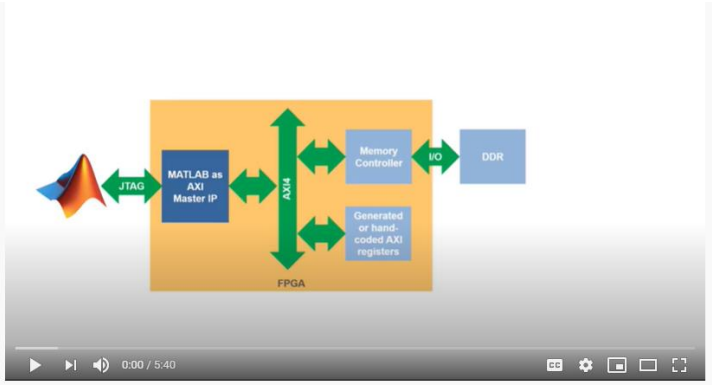
Optional:

- HDL Coder



HDL Verifier: FPGA Data Capture

https://www.youtube.com/watch?v=eHebToNdLpU

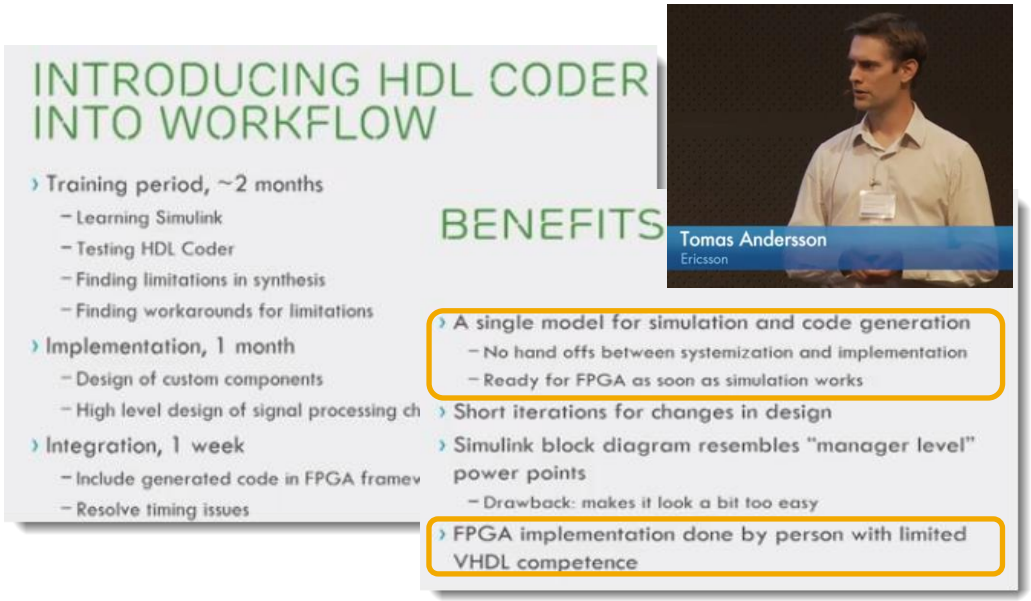


MATLAB as AXI Master with Xilinx FPGA and Zynq SoC Boards

https://www.youtube.com/watch?v=U-RaLUFkodI

Resources to Get Started and Speed Adoption

- Getting started:
  - MATLAB Onramp
  - Simulink Onramp
  - HDL pulse detector self-guided tutorial and videos



- Proof-of-concept guided evaluations
  - Free support via weekly WebEx meetings using custom sample designs

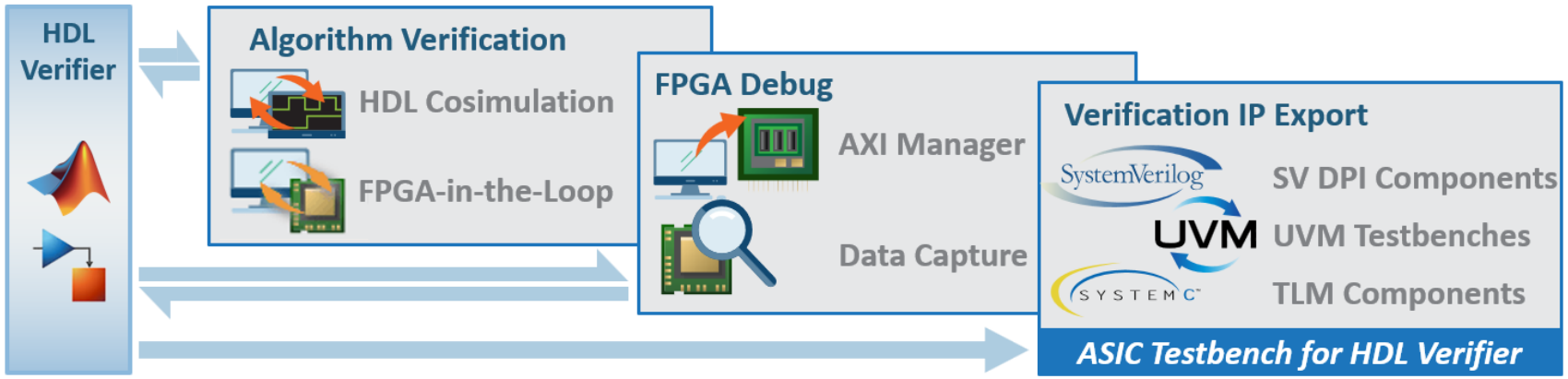
- Training & consulting services
  - HDL code generation, FPGA signal processing & Zynq™ programming training courses
  - Consulting service on deep technical coaching, custom design / hardware and more

Conclusion

This tutorial covered three features of HDL Verifier used with generated or hand-coded HDL

- **HDL Cosimulation** works in conjunction with supported HDL Simulator to provide a convenient and efficient way to verify automatically generated or handwritten HDL code.
- **FPGA-in-the-Loop** provides a convenient and efficient way to verify automatically generated or hand-coded HDL on a target board.
- **FPGA Data Capture** enables you to observe your design's signals while it runs on an FPGA.

Learn about AXI Manager, SystemVerilog DPI-C, and UVM component generation capabilities available in HDL Verifier here.



Trademark Notices

- *MathWorks®, MATLAB® and Simulink® are registered trademarks of The MathWorks, Inc. DSP HDL Toolbox ™, DSP System Toolbox ™, Fixed-Point Designer ™, HDL Coder ™, HDL Verifier ™, MATLAB Coder ™, Signal Processing Toolbox ™, and the L-shaped membrane logo are trademarks of The MathWorks, Inc.*
- *Siemens® is a registered trademark of Siemens Trademark GmbH & Co. KG. Questa ™ and MODELSIM ™ are trademarks of Siemens Industry Software, Inc.*
- *Cadence® is a registered trademark and Xcelium is a trademark of Cadence Design Systems, Inc.*
- *Synopsys® and VCS® are registered trademarks of Synopsys, Inc.*
- *AMD® and Xilinx® are registered trademarks of Advanced Micro Devices, Inc. AMD Vivado ™ is a trademark of Advanced Micro Devices, Inc.*
- *ZedBoard ™ is a trademark of Avnet, Inc.*
- *Altera®, MAX ®, and Arria are registered trademarks of Altera Corporation.*
- *All other trademarks herein are the property of their respective owners.*