



Creating and Simulating AMD Vitis™ Model Composer Designs

Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

Create and Simulate an HLS Design

Create and Simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

Summary

What is AMD Vitis™ Model Composer?

AMD Vitis™ Model Composer **accelerates** development by offering a **productive** environment within MathWorks Simulink® for simulation, analysis, code generation, and hardware validation

AMD Vitis™ Model Composer

Model-based Design Tool



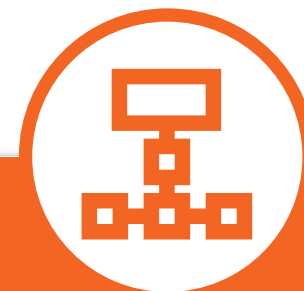
Enables rapid design exploration



Provides a library of performance-optimized HDL, HLS, and AI Engine blocks



Accelerates the path to production



Transforms your design through automatic optimizations

Design Flows Using AMD Vitis™ Model Composer

Algorithm Exploration

- Get a feel for likely design problems
- Estimate the performance and resource utilization in hardware

Implementing as Part of a Larger Design

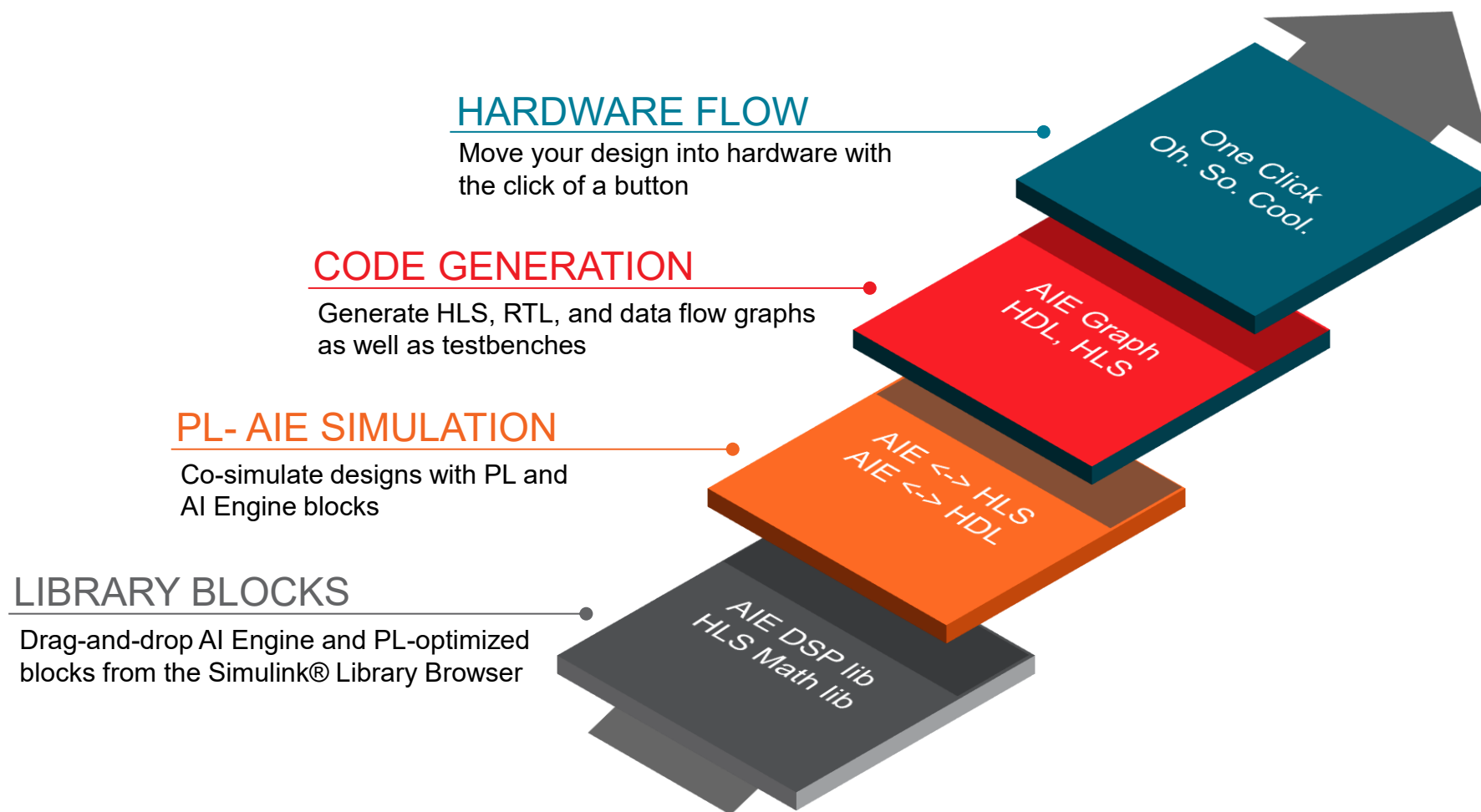
For sophisticated external interfaces:

- Implement parts of the design using Vitis™ Model Composer
- Implement other parts outside
- Combine the parts into a working whole

Implementing a Complete Design

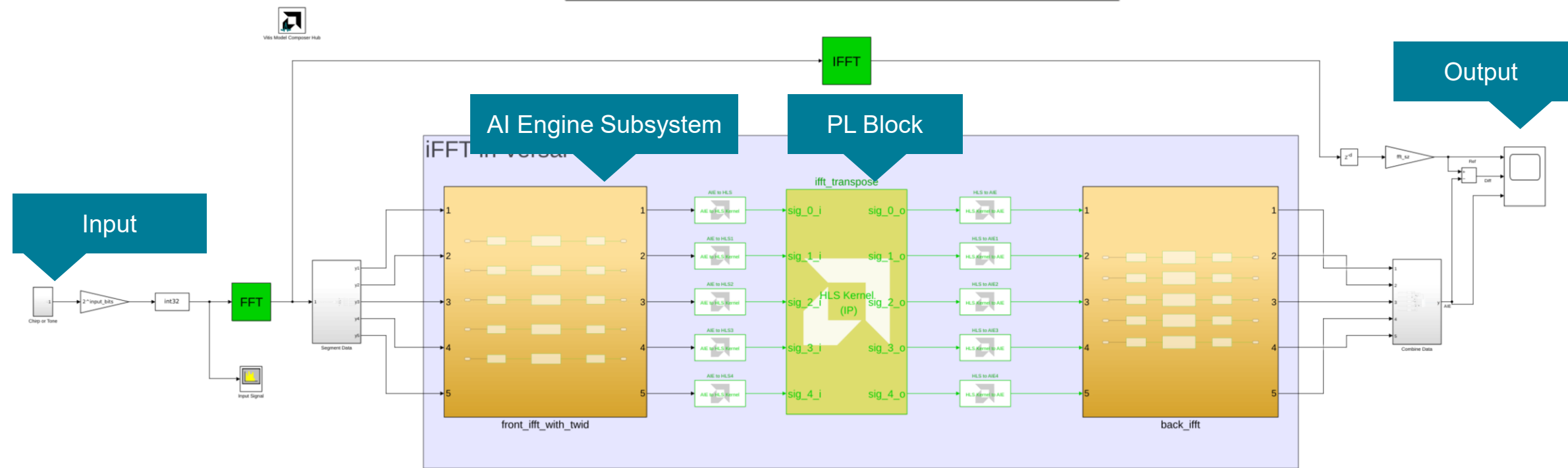
- Everything needed for a design is available inside
- Validate button instructs Model Composer to translate the design into HDL/HLS/AI Engine sources and write the files needed to process the design using downstream tools

AMD Vitis™ Model Composer in a Nutshell



Anatomy of a Design

64k iFFT using AI Engine and PL



Built on top of **MathWorks Simulink®**, AMD Vitis™ Model Composer enables the rapid design exploration of algorithms and accelerates the path to hardware

Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

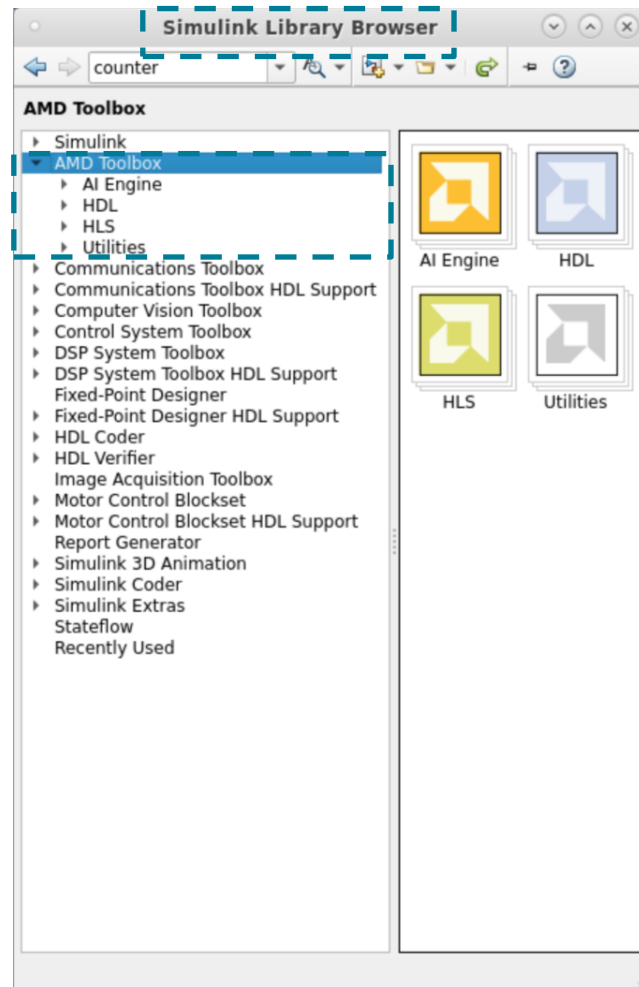
Create and Simulate an HLS Design

Create and Simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

Summary

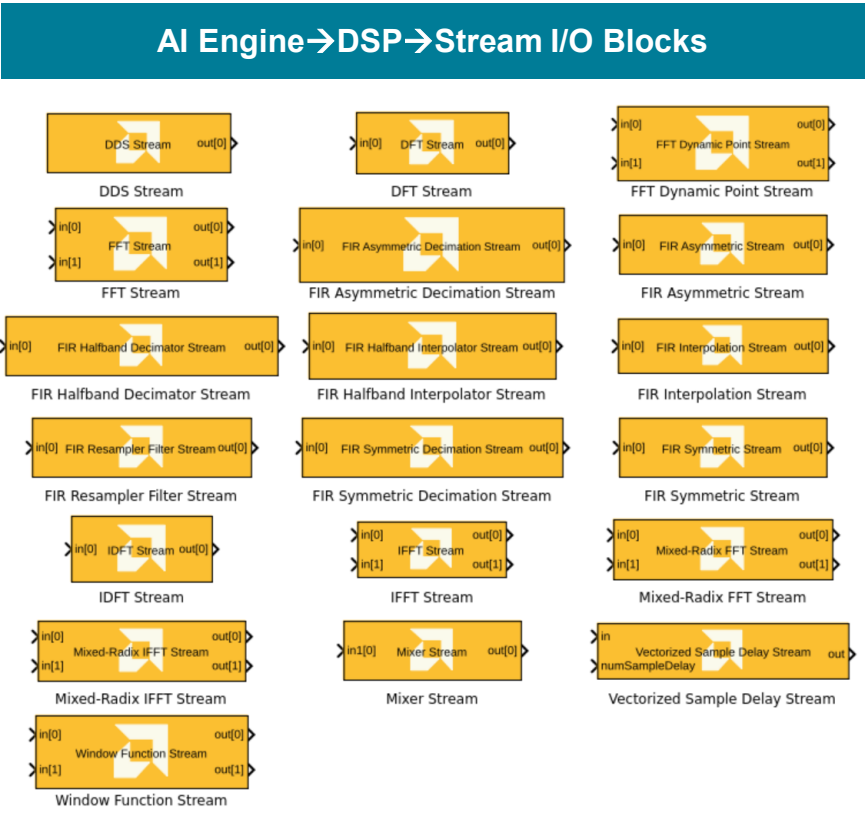
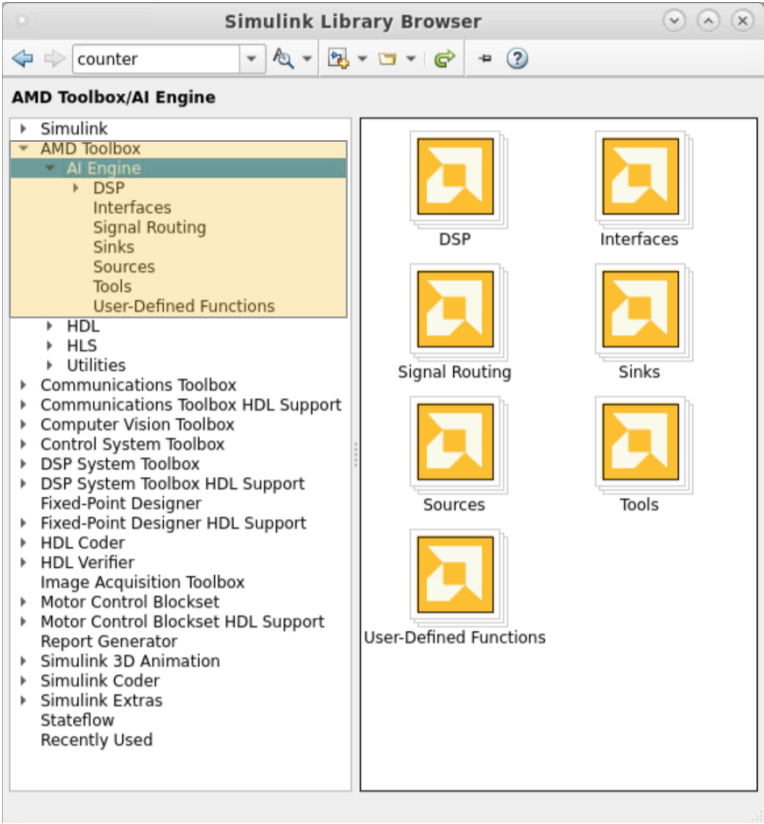
AMD Vitis™ Model Composer Library Blocks



Highly **optimized** blocks, targeting:

- AI Engines
- Programmable Logic

AI Engine Library Blocks

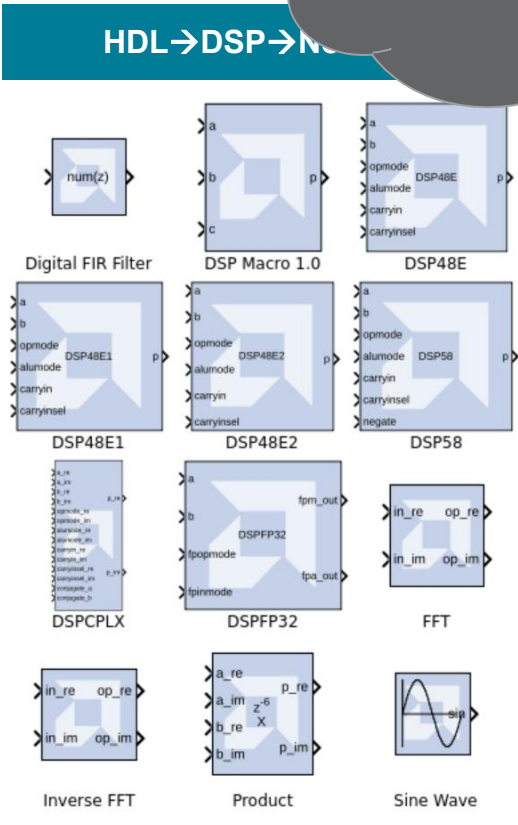
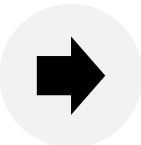
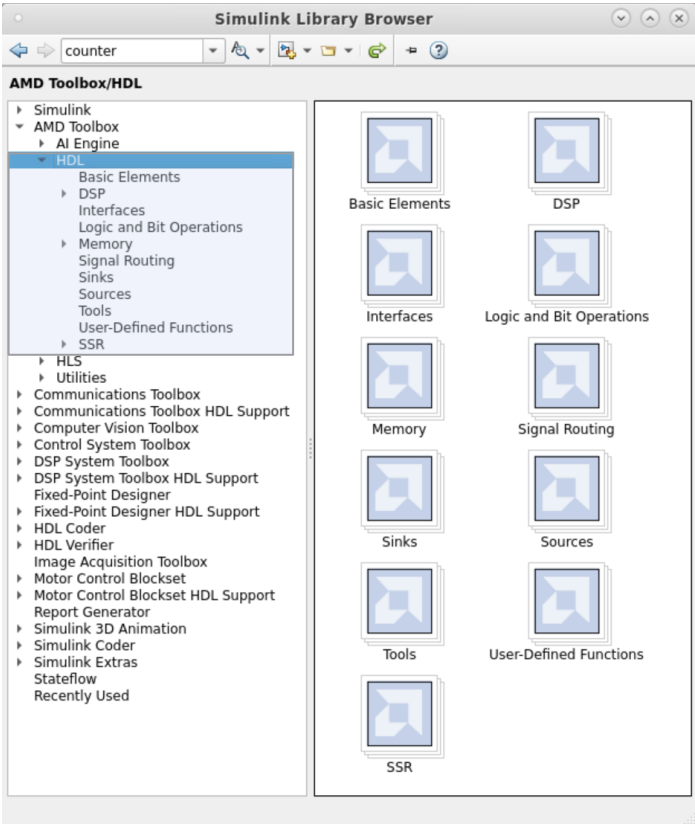


Bit accurate

AIE and AIE-ML devices

HDL Library Blocks

HDL library is from
**Xilinx System
Generator**

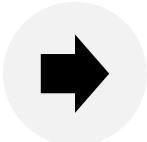
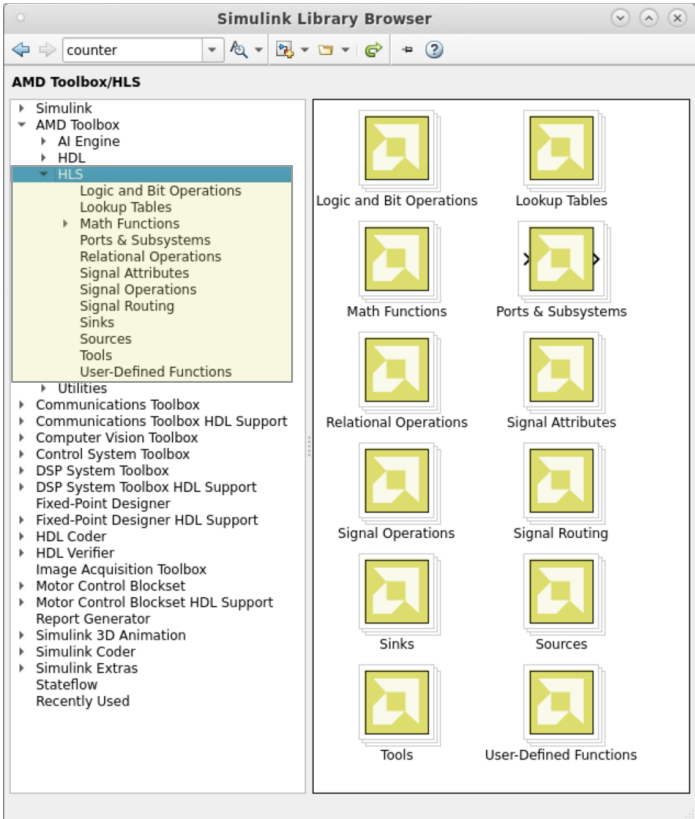


Cycle accurate and bit accurate

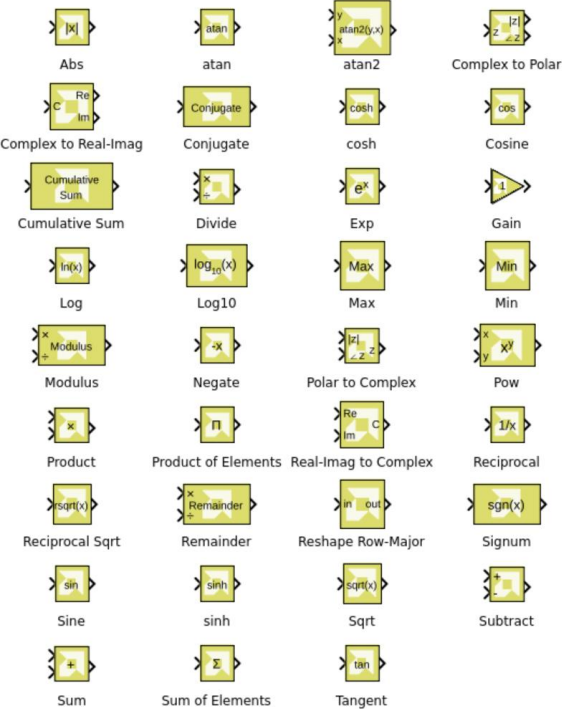
Uses C model for fast simulation

Generates RTL (Verilog/VHDL)

HLS Library Blocks



HLS→Math Functions→Math Operations

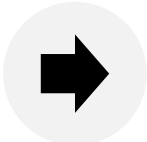
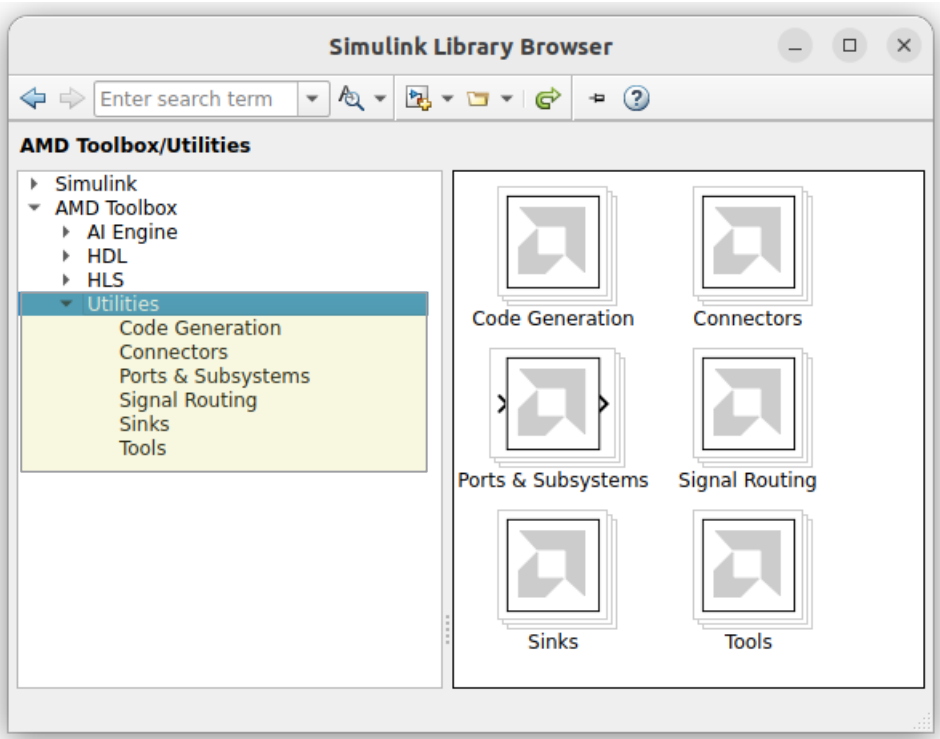


Bit accurate

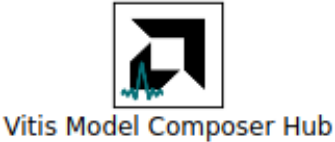
Uses C model for fast simulation

Generates C++ HLS code

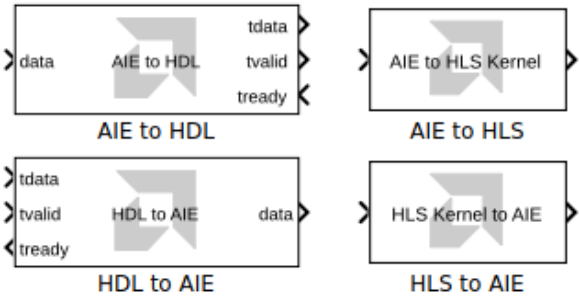
Utilities Library Blocks



Utilities→Code Generation



Utilities→Connectors



Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

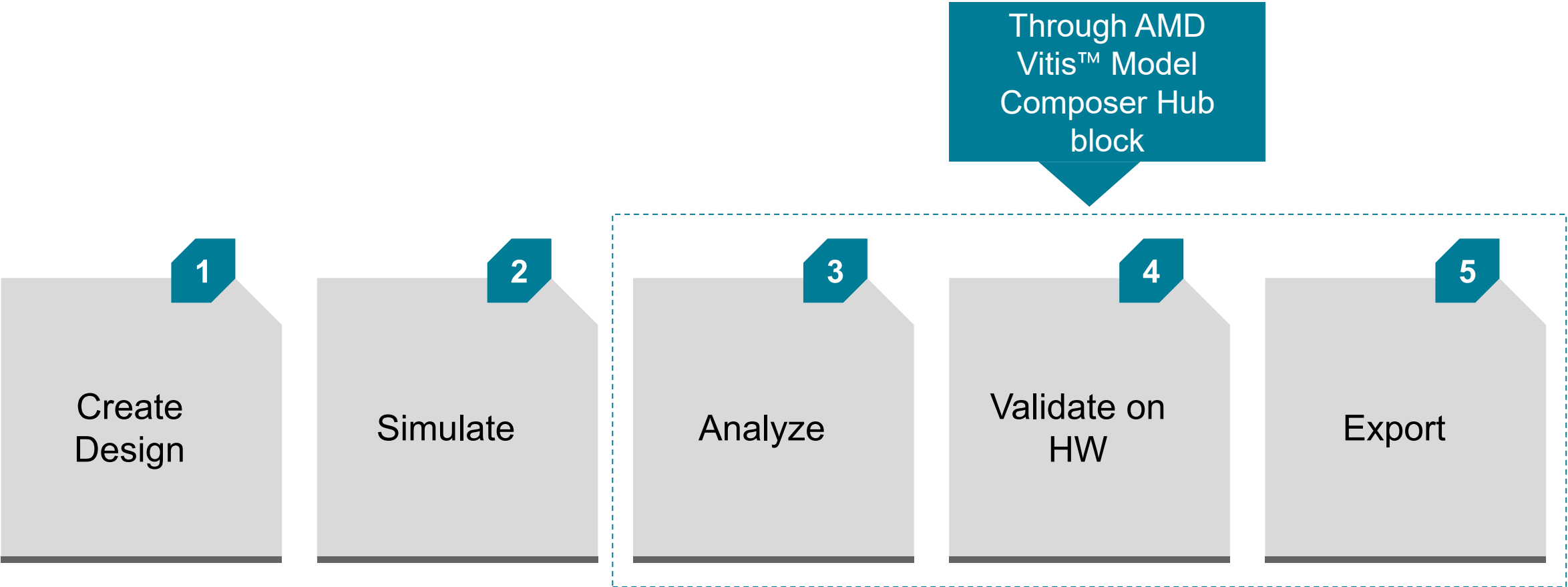
Create and Simulate an HLS Design

Create and Simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

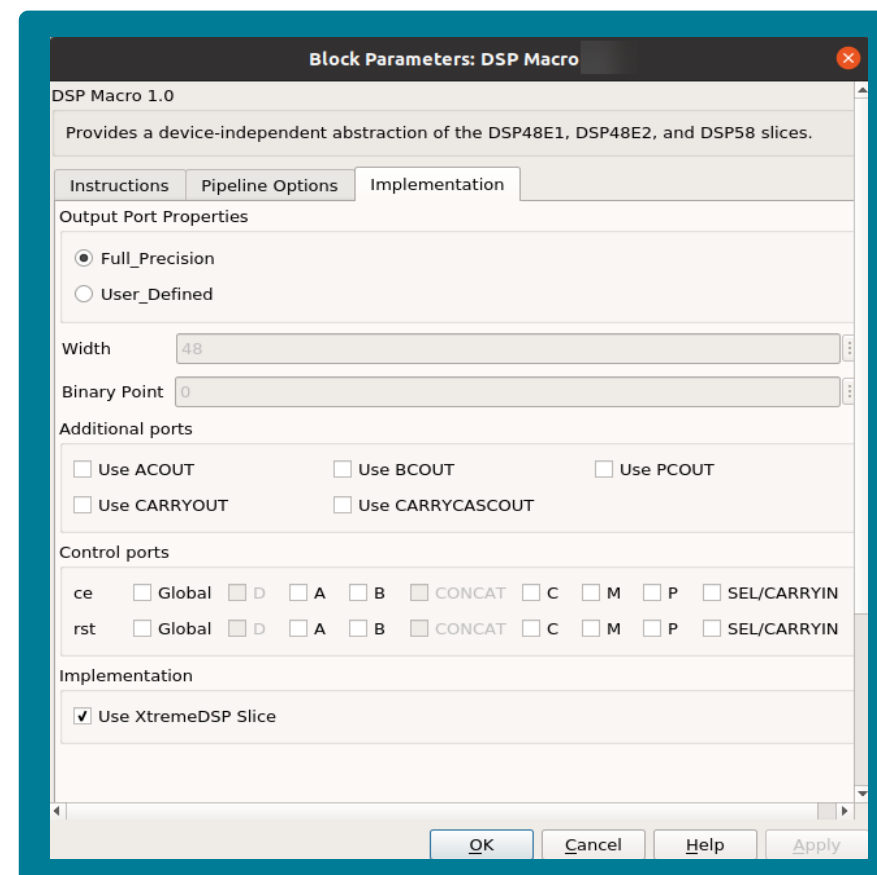
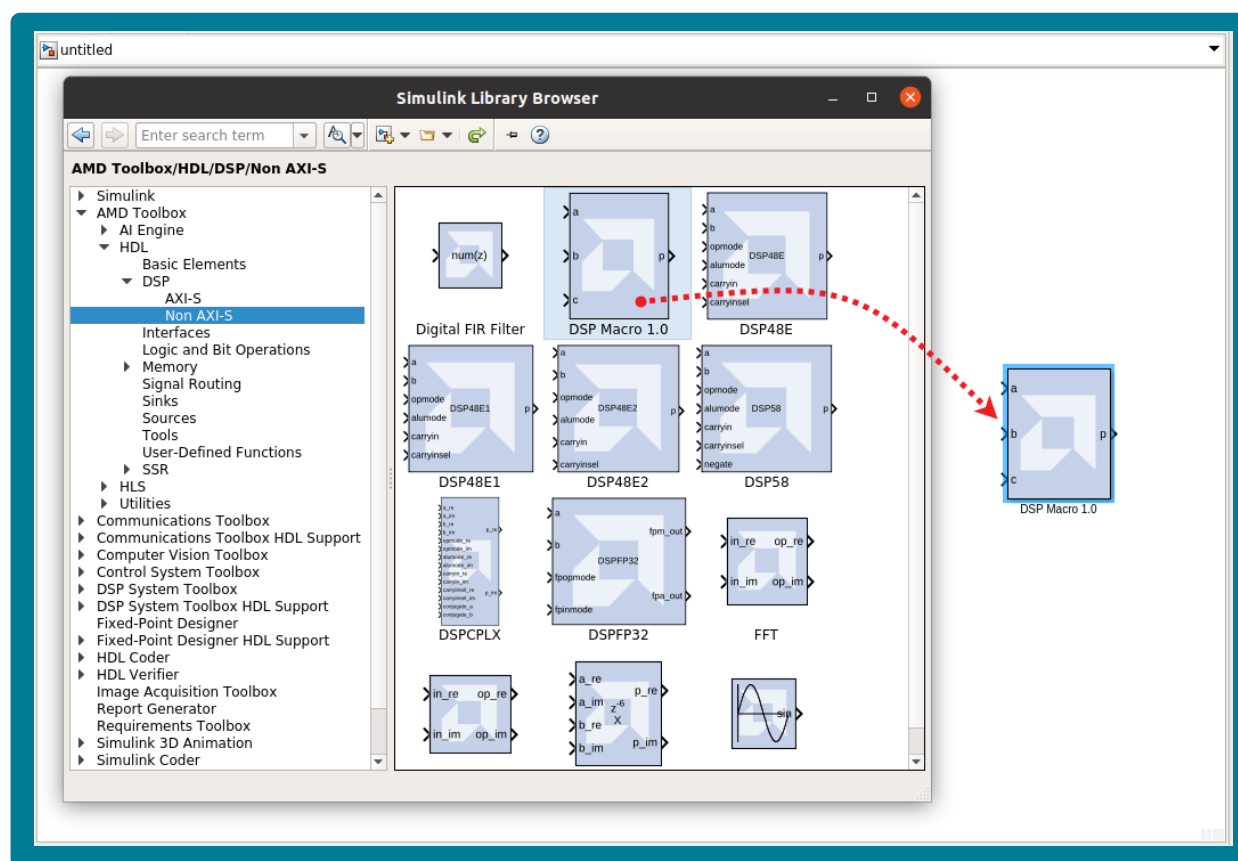
Summary

Homogeneous Design Flow – HDL Design



Creating an AMD Vitis™ Model Composer HDL Design

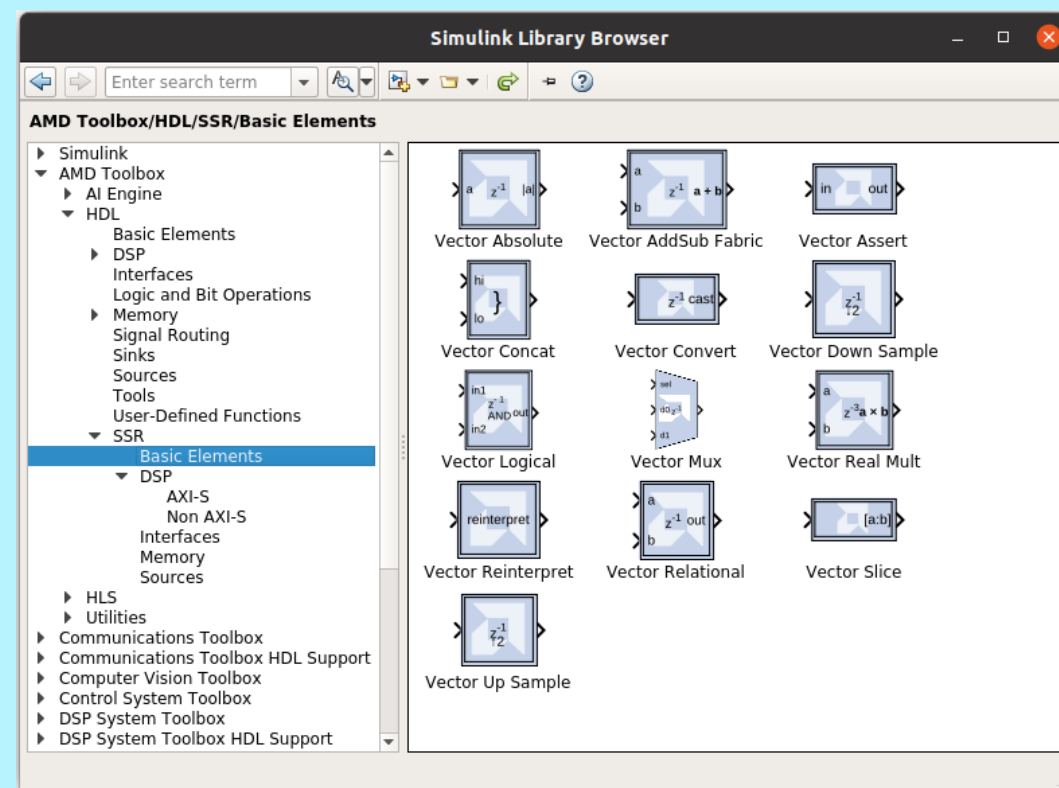
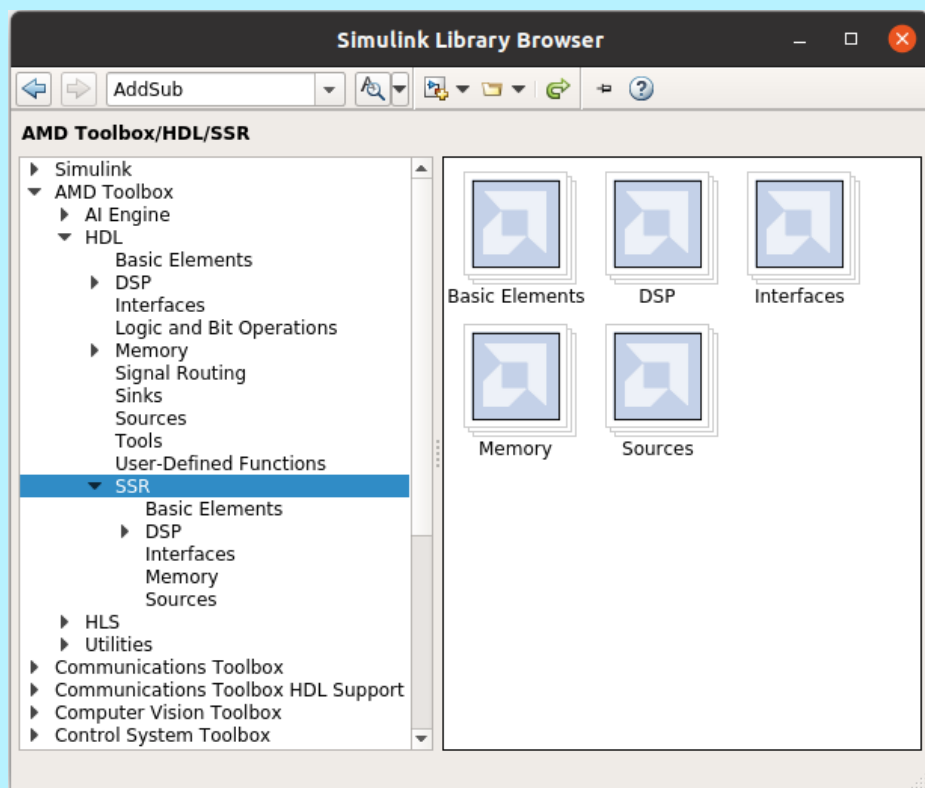
- Drag-and-drop blocks onto a new sheet to build a design
- Right-click a block to format the block or double click the block to configure
- Connect the blocks with signal lines



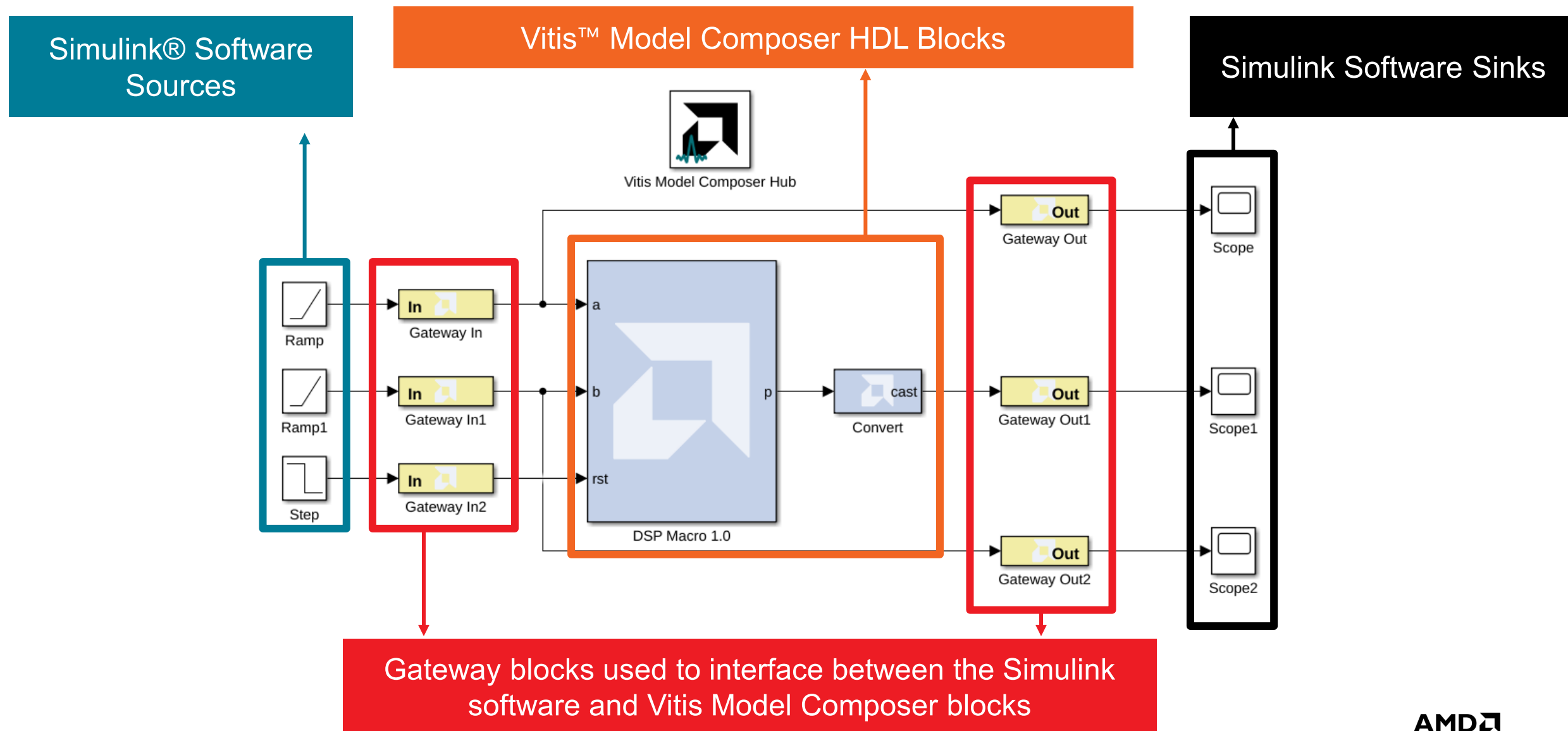
SSR Library

SSR: Parameter that determines how many parallel samples to accept for every clock cycle

Widely applicable to all AMD devices, especially
AMD Zynq™ UltraScale+™ RFSoc devices



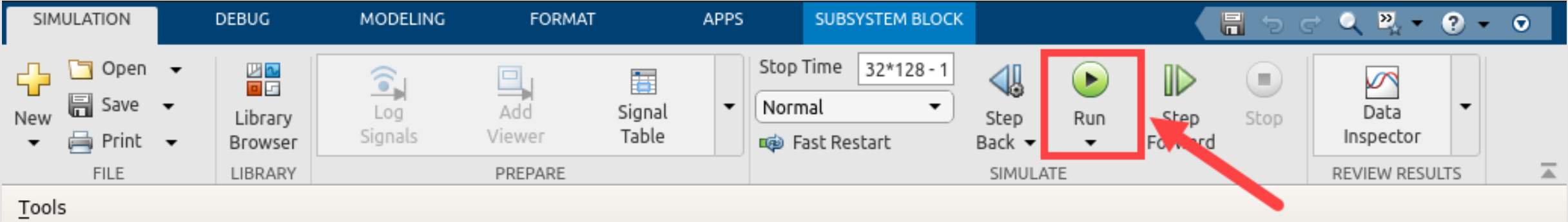
AMD Vitis™ Model Composer HDL Design - Example



Running Simulink Simulation

Compile and execute the design to produce the outputs

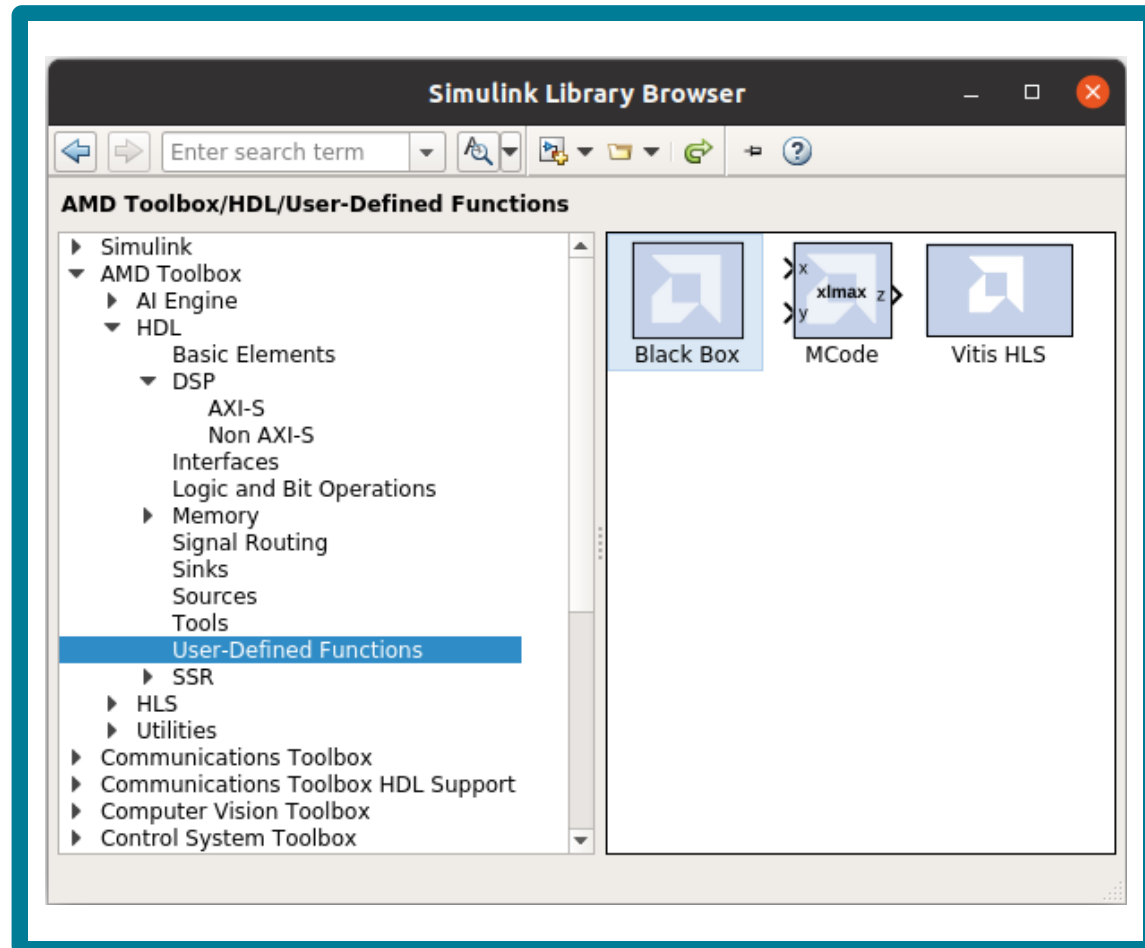
Define the inputs of the design using any Simulink® tool source blocks and analyze the output



Review results by connecting any of the Simulink tool sink blocks to appropriate points in the design

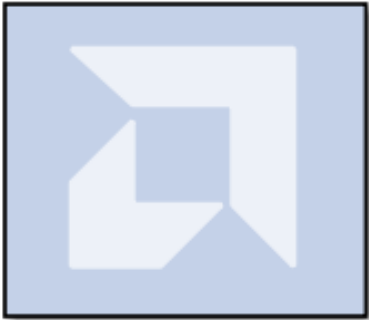
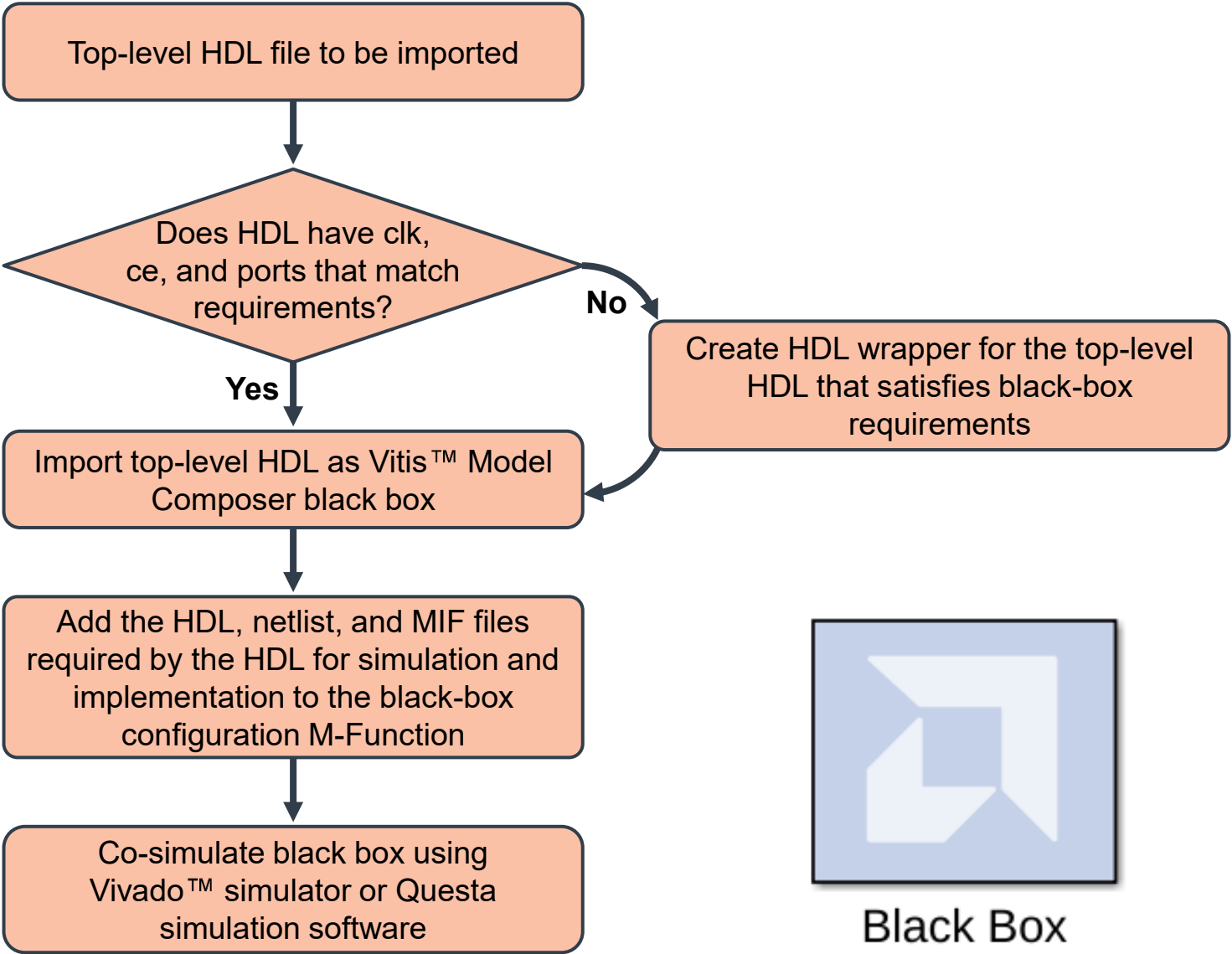
Importing an HDL Module Using the Black Box Block

Create and use your own HDL modules



- Allows HDL logic components in the HDL library
- Behaves like other Vitis™ Model Composer HDL blocks
- Ports can be connected to the rest of the design
- Can be configured to support either synchronous clock designs or multiple hardware clock designs

HDL Import Flow



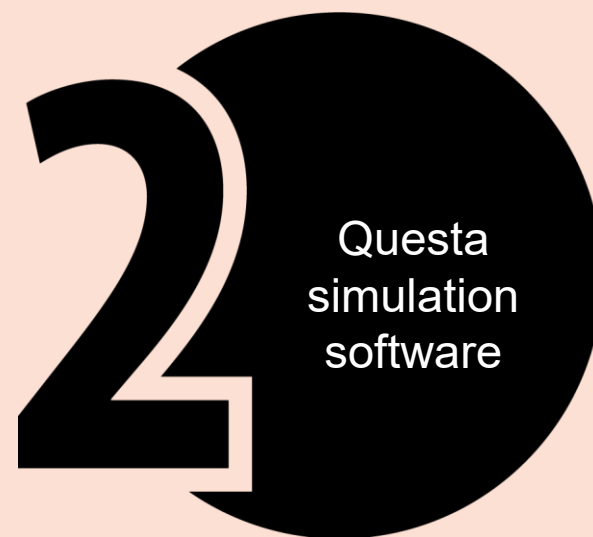
Black Box

HDL Co-Simulation

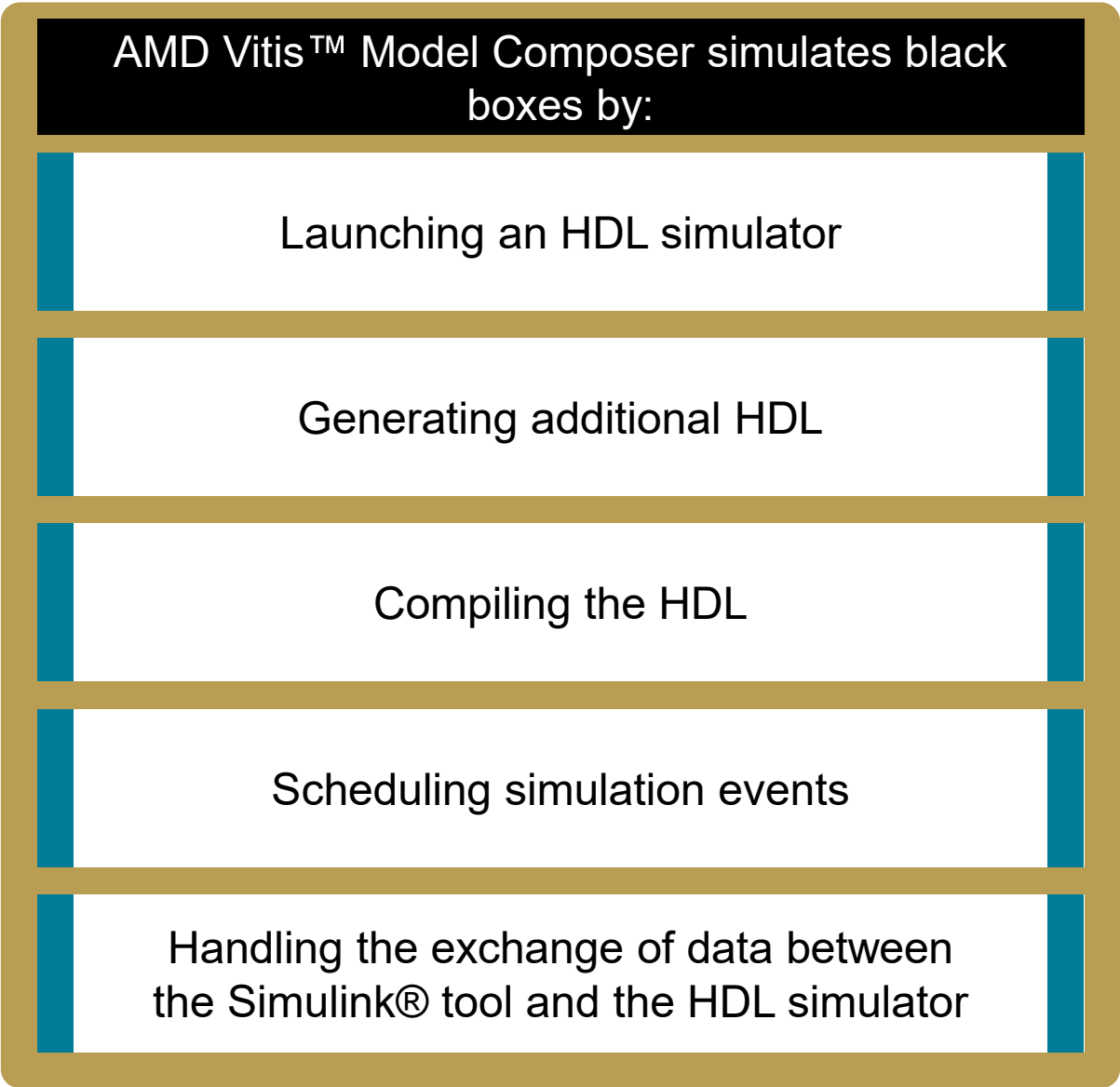
Why HDL Co-Simulation?

Simulink® software does not have the capability to perform HDL simulation
During HDL co-simulation, the black-box portion of the design is simulated by an HDL simulator

Supported Simulators



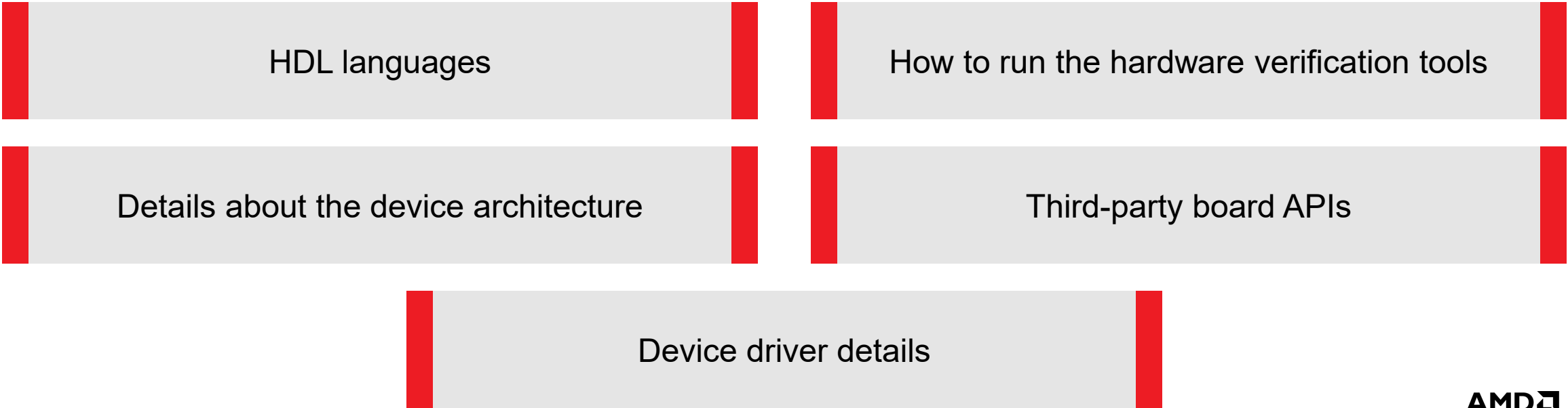
HDL Co-Simulation



Hardware Co-Simulation

- Accelerated simulation for HDL designs
- Allows a design running on the device to be directly executed in a Simulink® simulation
- Automatically creates a hardware simulation hub
- This hardware will co-simulate with the rest of the Simulink system

Perform hardware verification without knowing



Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

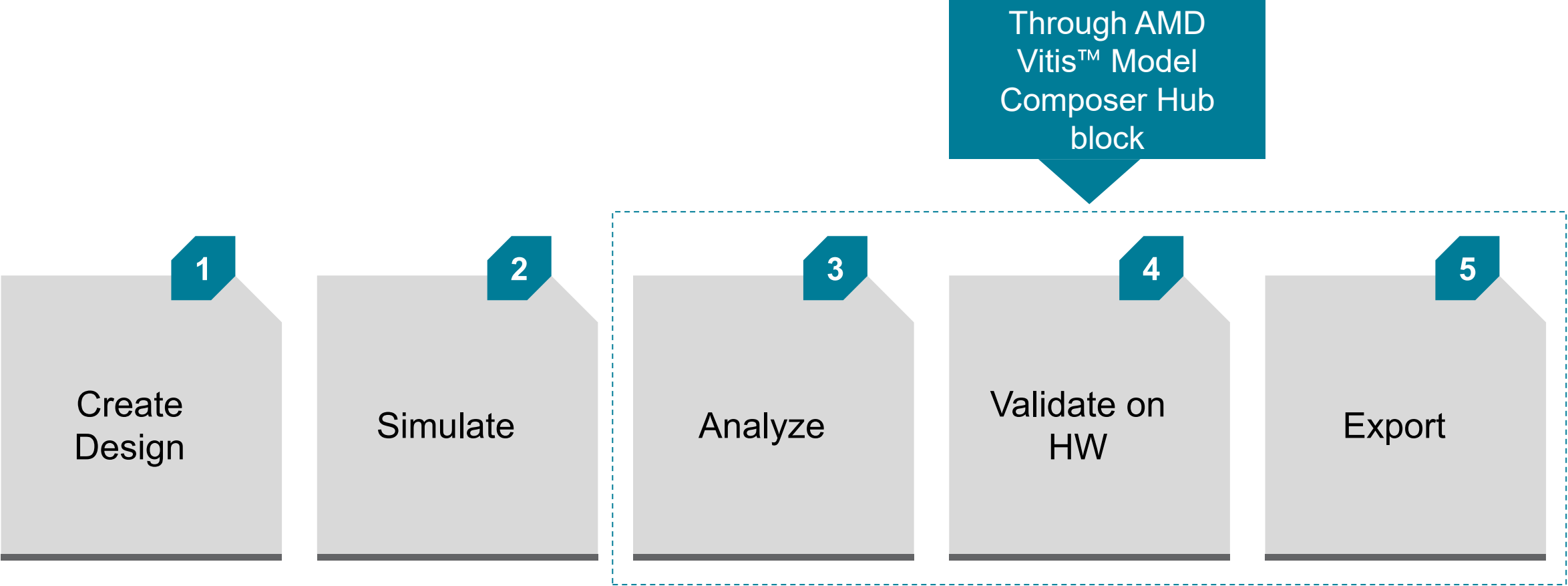
Create and simulate an HLS Design

Create and Simulate an AI Engine Design

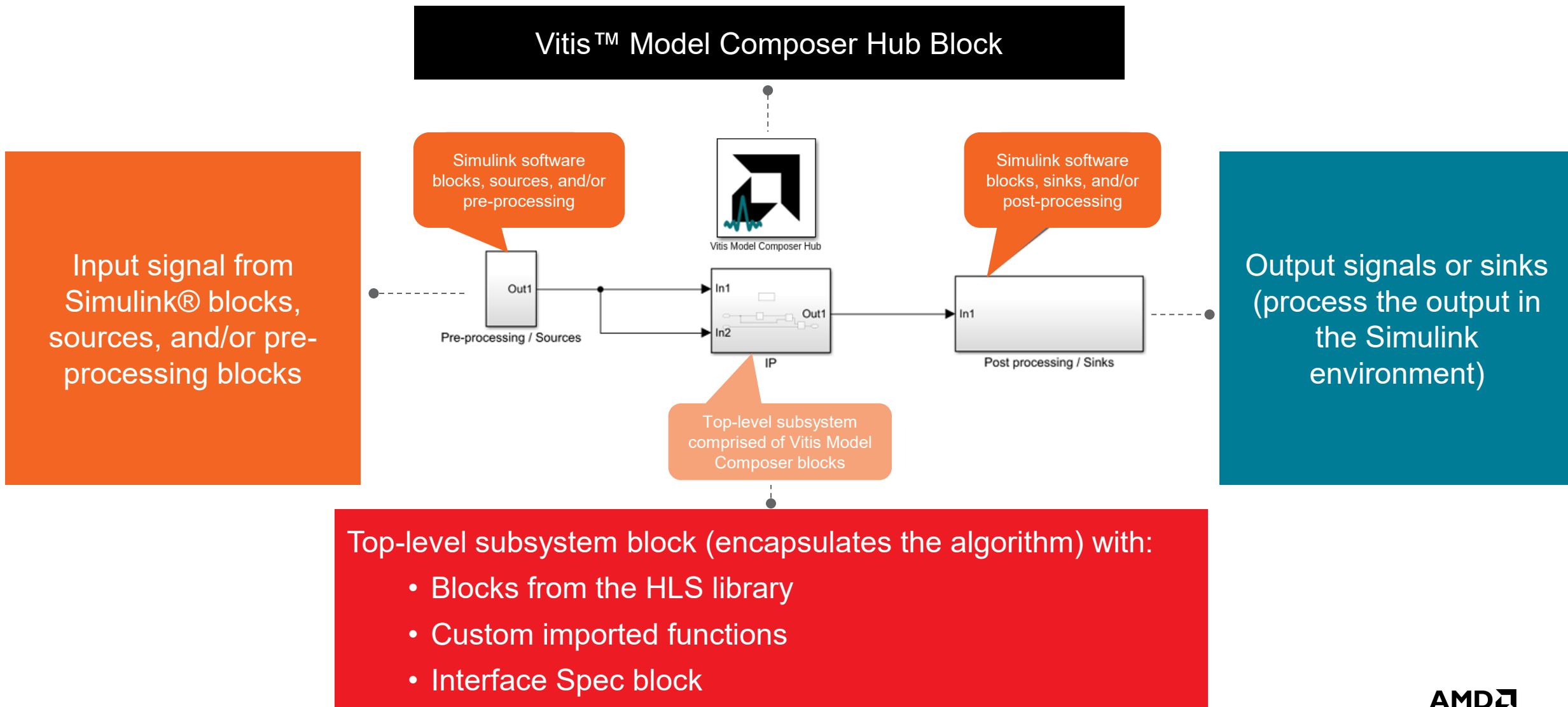
Create a Heterogeneous (AI Engine + PL) Design

Summary

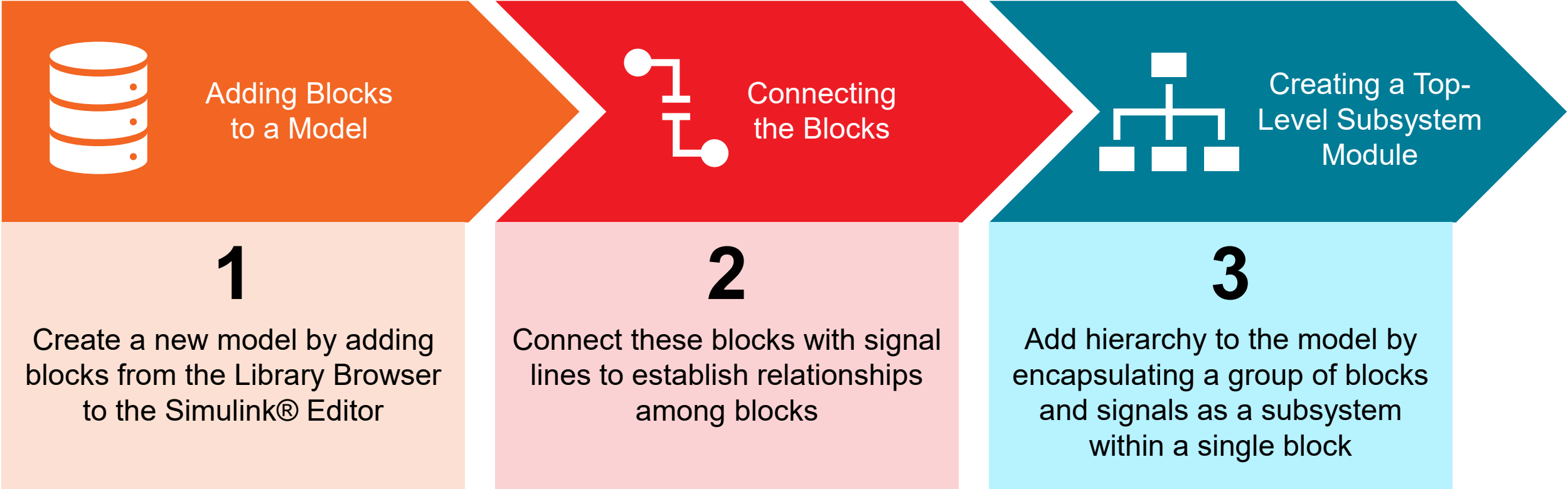
Homogeneous Design Flow – HLS Design



Elements of an AMD Vitis™ Model Composer Design with HLS Blocks



Creating an AMD Vitis™ Model Composer Design with HLS Blocks



Importing C/C++ Code as Custom Blocks



AMD Vitis™ Model Composer lets you import C or C++ code to create new blocks that can be added to a library



`xmcImportFunction:`

- Let's you specify the required source files and automatically creates an associated block that can be added into a model in the Simulink® environment



Requirements:

- Function source can be defined in a header file (.h) or in a C or C++ source file (.c, .cpp), but the header file must include the function signature
- Function arguments can be real or complex types of scalar, vectors, matrices, or fixed-point data types



Can change the source code without the need to re-import the block

Using the `xmcImportFunction` Command

```
xmcImportFunction('libName',{'funcNames'],'hdrFile',{'srcFiles'],'srchPaths'],'options')
```

libName

Specifies the name of the Vitis™ Model Composer HLS library that the new block is added to

funcNames

Specifies a list of one or more function names defined in the source or header files to import as a Vitis Model Composer block

hdrFile

Specifies a header file (.h) that contains the function declarations or definitions

srcFiles

Specifies a list of one or more source files to search for the function definitions

srchPaths

Specifies a list of one or more search paths for header and source files

Using the xmcImportFunction Command – Example

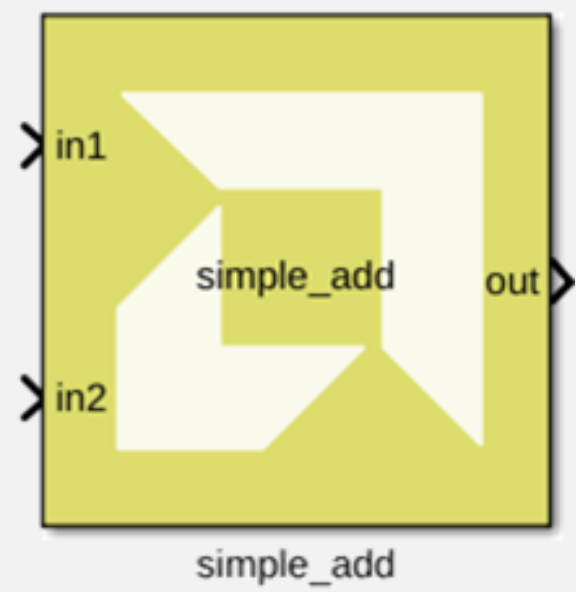
```
// simple.h
void simple_add(const double in1, const double in2, double *out)
{
    *out = in1 + in2;
}
```

To import the `simple_add` function as a block:

```
xmcImportFunction('SimpleLib',{'simple_add'},'simple.h',{},{})
```

- `SimpleLib` is the name of the Vitis™ Model Composer HLS library to add the block to
- `simple_add` is the function name to import
- `simple.h` is the header file to look in

Using the xmcImportFunction Command – Example



simple_add Block



Block Parameters: simple_add

Import Function

Function declaration

void simple_add(const double in1, const double in2, double * out);

Function General

Interfaces

Direction	Name	Type	Dimension
Input	in1	double	1
Input	in2	double	1
Output	out	double *	1

OK Cancel Help Apply

simple_add Block Parameters

Defining Blocks Using Function Templates

If you want to create a block that:



Accepts inputs of
different sizes



Supports different
data types



Accepts signals with different
fixed-point lengths and
fractional lengths

Function template lets you create a block that accepts:

- Variable signal size, data type, or data dimensions

Increase the re-usability of your block library

Defining Blocks Using Function Templates – Example

Defining Blocks Using Function Templates

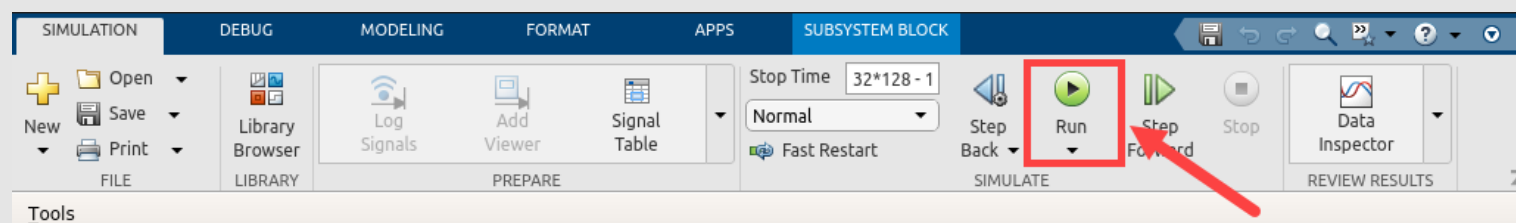
```
#include <stdint.h>
template <int ROWS, int COLS>
void simple_matrix_add(const int16_t in1[ROWS][COLS],
                      const int16_t in2[ROWS][COLS],
                      int16_t out[ROWS][COLS]) {
    for (int i = 0; i<ROWS; i++) {
        for (int j = 0; j<COLS; j++) {
            out[i][j] = in1[i][j] + in2[i][j];
        }
    }
}
```

```
xmcImportFunction('SimpleLib',{'simple_matrix_add'},...
'template_example.h',{},{},'unlock')
```

Running Simulink Simulation

Compiling and executing the design

Define the inputs of the design using any Simulink® tool source blocks and analyze the output



Results can be analyzed with MATLAB® scripts and visualization tools like the Simulation Data Inspector

Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

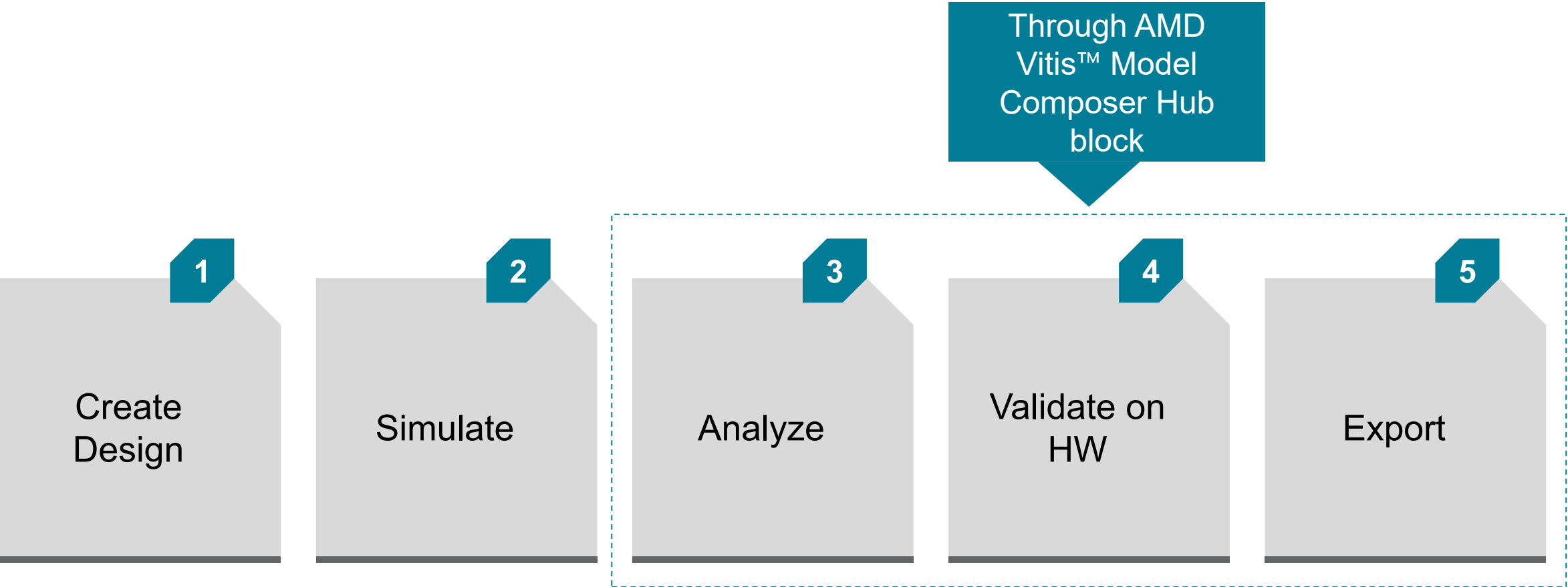
Create and Simulate an HLS Design

Create and simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

Summary

Homogeneous Design Flow – AI Engine Design



AI Engine DSP Library (DSPLib)

Library of commonly used DSP functions optimized for AI Engines

Different DSPLib functions as blocks are provided in the AMD Toolbox > AI Engine > DSP library

Synchronized DSP functions in Vitis_Libraries GitHub

Product Solutions Open Source Pricing

/ Vitis_Libraries

Public

<> Code

Issues 70

Pull requests 10

Discussions

Actions

main

14 branches

21 tags

create main branch from next branch

blas	Squashed 'blas' changes fr
codec	Squashed 'codec' changes
data_analytics	Squashed 'data_analytics' c
data_compression	Squashed 'data_compressio
data_mover	Squashed 'data_mover' cha
database	Squashed 'database' chang
dsp	Squashed 'dsp' changes fr
graph	Squashed 'graph' changes
hpc	Squashed 'hpc' changes fr

DDS

out[0]

DDS

in[0] DFT

out[0]

DFT

in[0] FFT

out[0]

FFT

in[0] FFT Dynamic Point

out[0]

FFT Dynamic Point

in[0] FIR Asymmetric

out[0]

FIR Asymmetric

in[0] FIR Asymmetric Decimation

out[0]

FIR Asymmetric Decimation

in[0] FIR Halfband

out[0]

FIR Halfband Decimator

in[0] FIR Halfband Interpolator

out[0]

FIR Halfband Interpolator

in[0] FIR Interpolation

out[0]

FIR Interpolation

in[0] FIR Resampler Filter

out[0]

FIR Resampler Filter

in[0] FIR Symmetric

out[0]

FIR Symmetric

in[0] FIR Symmetric Decimation

out[0]

FIR Symmetric Decimation

in[0] FIR TDM

out[0]

FIR TDM

in[0] IDFT

out[0]

IDFT

in[0] IFFT

out[0]

IFFT

in[0] Mixed-Radix FFT

out[0]

Mixed-Radix FFT

in[0] Mixed-Radix IFFT

out[0]

Mixed-Radix IFFT

in[0] Mixer

out[0]

Mixer

in[0] Vectorized Sample Delay

out[0]

Vectorized Sample Delay

in[0] Window Function

out[0]

Window Function

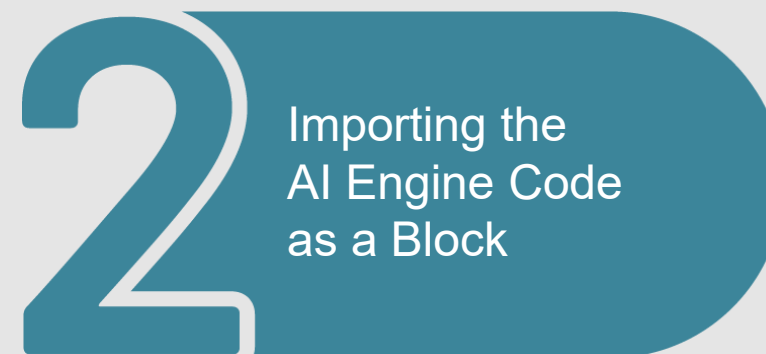
AI Engine > DSP > Buffer I/O Blocks

Creating an AI Engine Design Using AMD Vitis™ Model Composer

AI Engine kernels are functions that form the fundamental building blocks of the data flow graph

AMD Vitis™ Model Composer supports generating the AI Engine data flow graph by importing the AI Engine kernel or sub-graph

Steps for Creating AI Engine Design



Preparing the AI Engine Kernels

AI Engine Kernels

- Declared as C/C++ functions that return void and can use special data types for arguments
- Should be defined each in their own source file
- Source files should include all relevant header files to allow for independent compilation

Data-accessing Mechanisms

Buffer-based Access

- Kernels can process the data in blocks called buffers
- Requires synchronization of input/output buffers before entering the kernel
- No synchronization required within the kernel to read or write the individual elements of data

```
input_buffer<cint16> myInputBuffer;  
output_buffer<int32> myOutputBuffer;
```

Stream-based Access

- Kernels can access data streams in a sample-by-sample fashion
- Each access to these streams is synchronized
- Direct stream communication channel between one AI Engine and the adjacent AI Engine—called a cascade

```
input_stream<cint16> * myInputStream;  
output_stream<cint16> * myOutputStream;
```


Importing the AI Engine Code as a Block

AI Engine library blocks to import kernel functions and graphs

AI Engine Kernel

AI Engine Class Kernel

AI Engine Graph

Input: Kernel or a data flow sub-graph



AMD Vitis™ Model Composer: Generates a block with interfaces that match the function arguments of a kernel or a graph

Importing the AI Engine Code as a Block

AI Engine library blocks to import kernel functions and graphs

AI Engine Kernel

AI Engine Class Kernel

AI Engine Graph



Block Parameters: AIE Kernel

AIE Kernel

Import an AI Engine kernel or an AI Engine kernel template as a block.



General Constraints


Parameters


Kernel header file:  

Kernel function:

Kernel init function:

Kernel source file:  

Kernel search paths:  Add

Preprocessor options: 

Import

OK Cancel Help Apply

- Use an AI Engine Kernel block from the AI Engine library to import this kernel

AMD Vitis™ Model Composer supports:

- Importing both buffer-based and stream-based kernels
- Cascade stream connections between two AI Engine processors
- Importing AI Engine kernels with runtime parameters and function templates



Importing the AI Engine Code as a Block

AI Engine library blocks to import kernel functions and graphs

AI Engine Kernel

AI Engine Class Kernel

AI Engine Graph



Block Parameters: AIE Class Kernel

AIE Class Kernel

Import an AI Engine class kernel or an AI Engine class template kernel as a block.



General Constraints


Parameters


Kernel header file:  

Kernel class:

Kernel function:

Kernel source file:  

Kernel search paths:  Add

Preprocessor options: 

Import

OK Cancel Help Apply

- Use an AI Engine **Class** Kernel block from the AI Engine library to import the C++ kernel class to have constructor parameters for specifying parameter values

AMD Vitis™ Model Composer supports:

- Kernels with default constructors and parameterized constructors
- Importing the kernels with class templates using the AI Engine **Class** Kernel block using template specialization



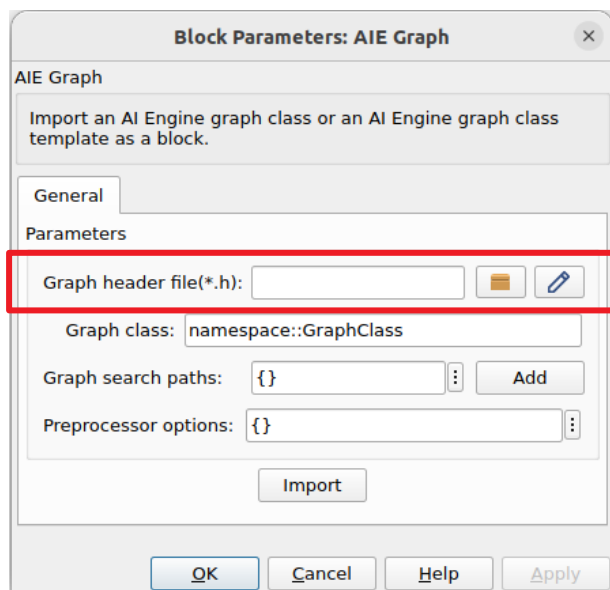
Importing the AI Engine Code as a Block

AI Engine library blocks to import kernel functions and graphs

AI Engine Kernel

AI Engine Class Kernel

AI Engine Graph



Using the header file (*.h)

- Graph is a connection of different compute kernel functions
- Graph code is imported as a block by selecting the AI Engine Graph block from the AI Engine library
- Connect the AI Engine Graph block and the AI Engine Kernel block to the simulate whole design in the Simulink® environment



Running Simulink Simulation

- After a high-level graphical design is created, simulate it interactively in the Simulink® environment
- Ensures the functional correctness of the design and displays the results

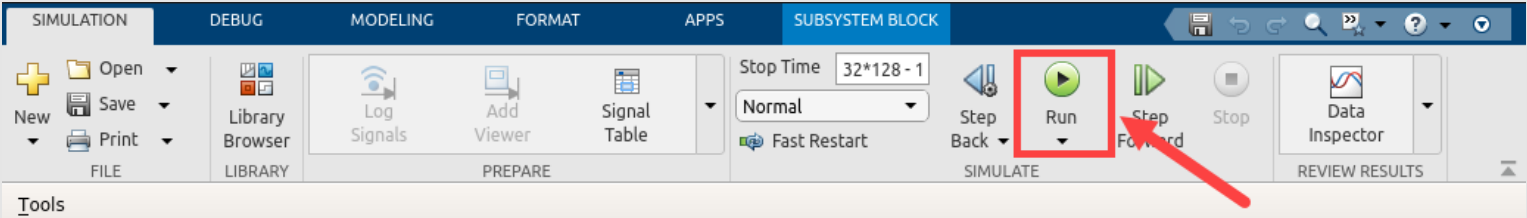
Compiling and executing the design

Simulink model defines input and output signals

AMD Vitis™ Model Composer provides two MATLAB® environment utilities to directly read/write data from/to the files:

xmcVitisRead

xmcVitisWrite



Progress window displays only when you are compiling a design for the first time; other times, it uses cached entry for faster simulation

Results can be reviewed by connecting any of the Simulink tool sink blocks to appropriate points in the design

Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

Create and Simulate an HLS Design

Create and Simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

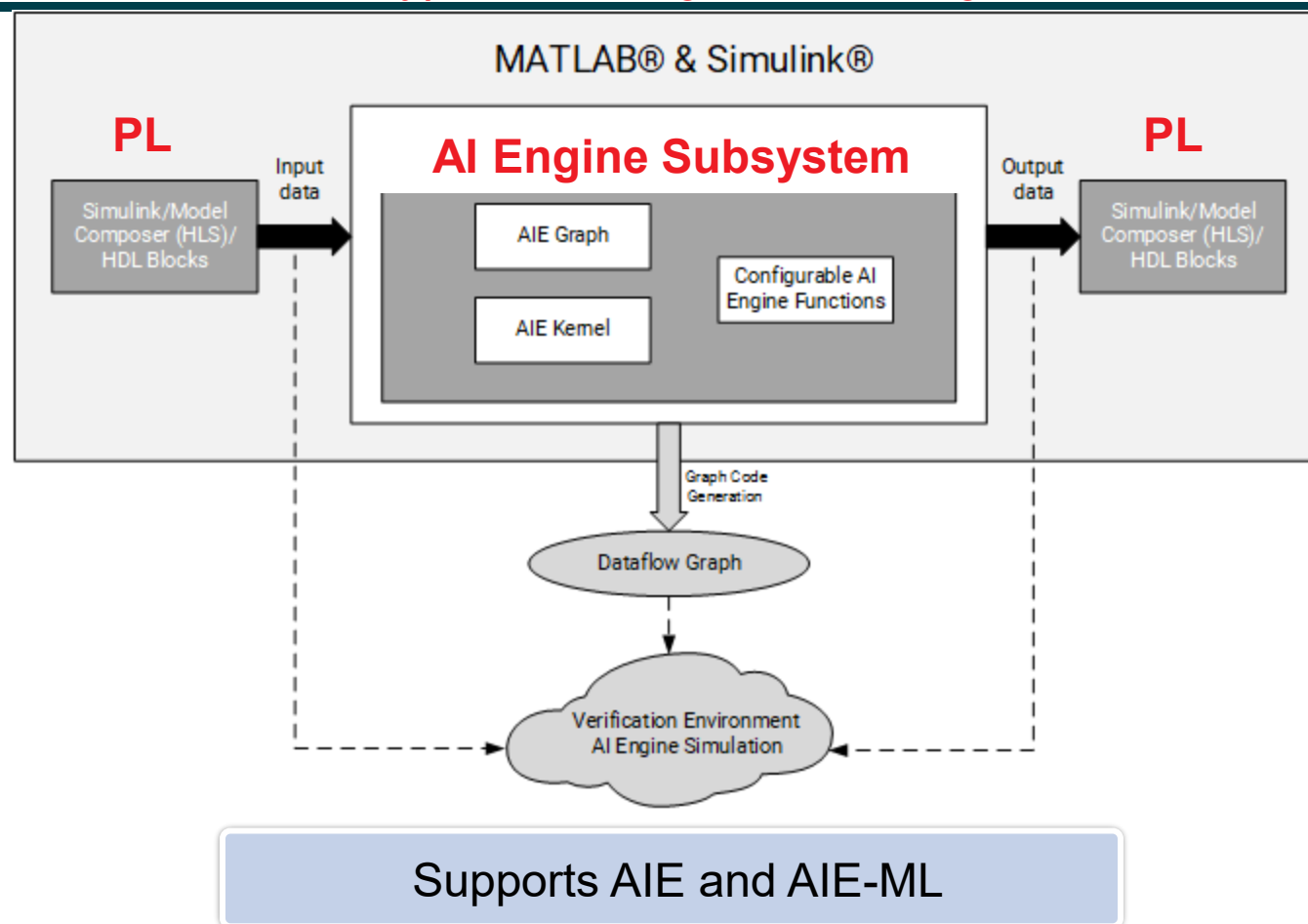
Summary

AMD Vitis™ Model Composer for AI Engine Development

Enables rapid simulation, exploration, and code generation of algorithms targeted for AI Engines from within the Simulink environment

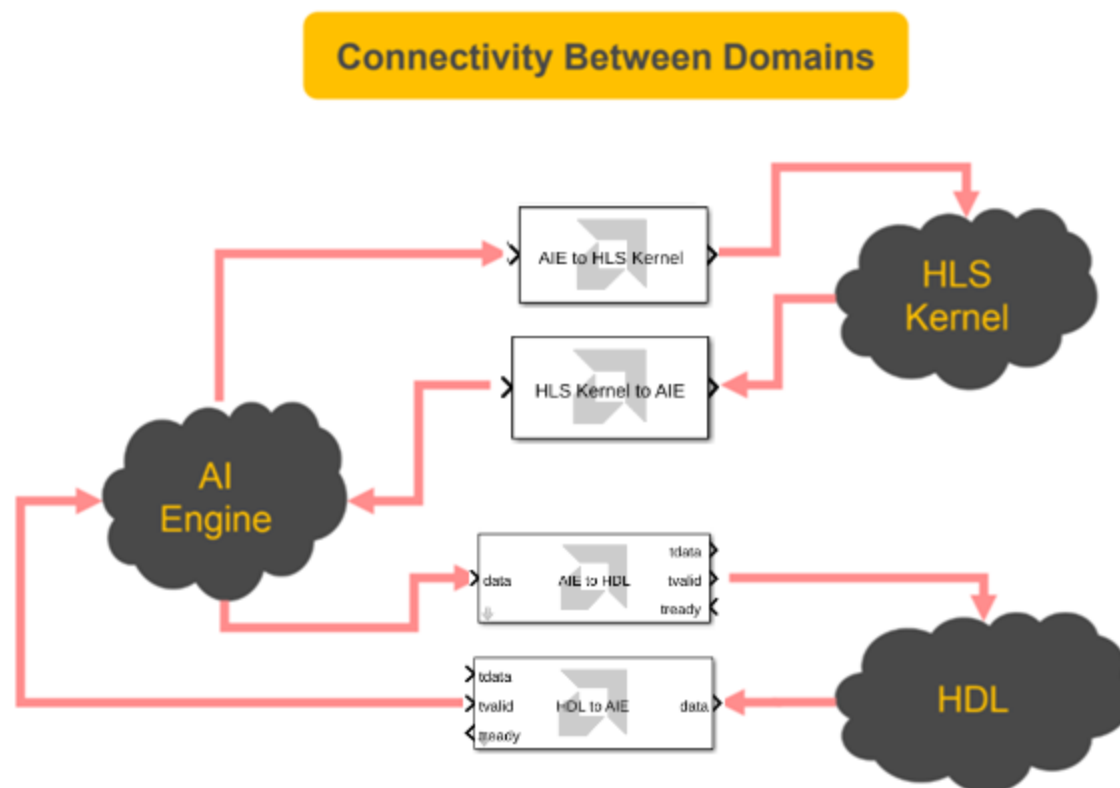
Typical Heterogeneous Design Flow

- Import AI Engine kernels and data flow graphs as blocks
- Control the behavior of the kernels and graphs
- Write PL kernels using RTL or HLS C/C++ functions
- Visualize the simulation results via the Simulink® software source and sink blocks



Connectivity Between Domains

AMD Vitis™ Model Composer takes advantage of the versatility of a heterogeneous system by interconnecting the various domains of the AMD Versal™ adaptive SoC, including the AI Engine, HDL, and HLS kernels



Connecting AI Engine and Non-AI Engine Blocks

AI Engine - Programmable Logic Integration

- AI Engine kernel imported into AMD Vitis™ Model Composer can be used as part of a larger Versal™ adaptive SoC system design
- Support for specifying kernels to run on the programmable logic (PL) region
- PL kernels can be written using RTL or HLS C/C++ functions
- Connection between AI Engine and PL block is routed through a physical channel interface tile
- Connecting an AI Engine kernel to an HLS PL kernel is allowed only if the data types and complexities of these ports match
- Interface blocks should be used to reconcile discrepancies

Interconnecting AI Engine and HDL Blocks

Interconnecting AI Engine and HLS Kernels

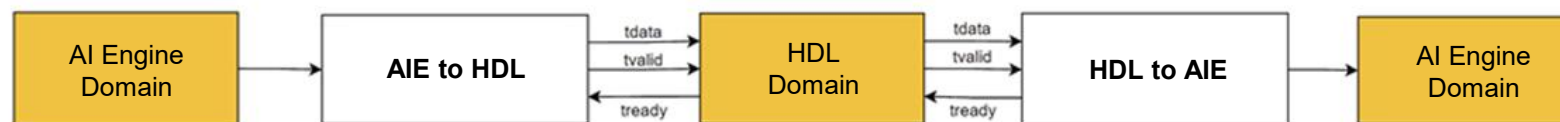
Interconnecting AI Engine and HDL Blocks

Helps to manage the sampling times across two domains and simulates a heterogeneous system with PL and AI Engines

Interface blocks are available in the Utilities library from **AMD Toolbox > Utilities > Connectors**

- AIE to HDL block: Connects AI Engine to HDL blocks using an AXI4-Stream-like interface
- HDL to AIE block: Connects HDL to AI Engine blocks using an AXI4-Stream-like interface

AIE to HDL and HDL to AIE blocks have tvalid and tready ports



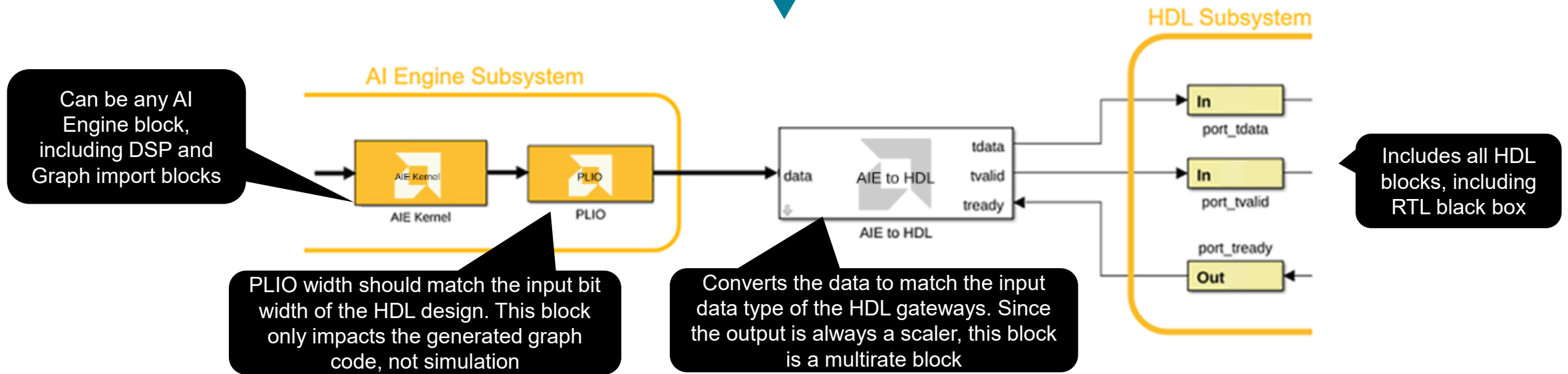
Connector Blocks

- Gateway from the AI Engine to the HDL domain can accept a vector input but generates a scalar output
- In the Simulink® environment, the HDL domain will run at a different rate than the AI Engine domain

AI Engine to HDL Block

Connects the output of an AI Engine block/subsystem with the input of an HDL block/subsystem

- Accepts variable-sized signals from AI Engine blocks along with the tready signal, which indicates whether the HDL domain can accept the data
- Bit width of the tdata output of the AI Engine to HDL signal is limited to 32, 64, or 128



Topology of Connections Between AI Engine and HDL

HDL to AIE Block

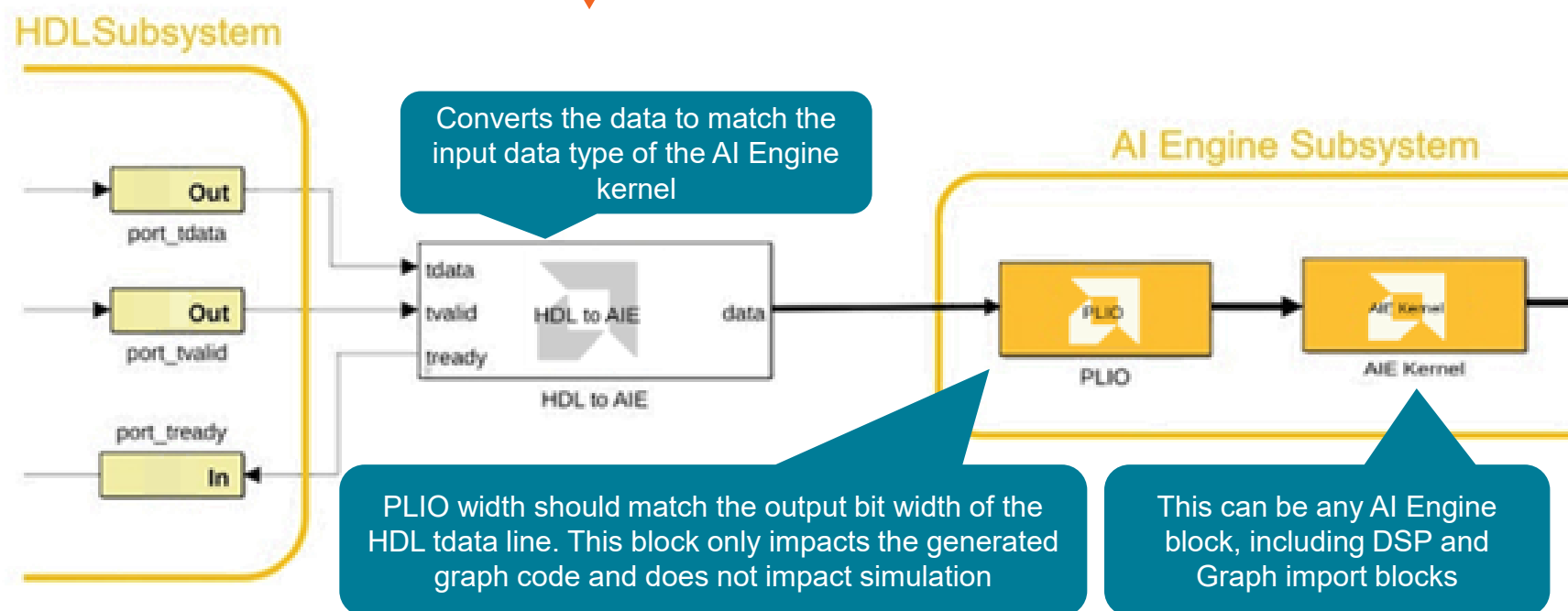
Connects the output of an HDL block/subsystem with the input of an AI Engine block/subsystem

- Accepts tdata, which is the primary input for the data, and the tvalid signal, which indicates the producer has valid data
- Bit width of the tdata output is limited to 32, 64, or 128, according to hardware functionality



Output from the HDL to AI Engine block is a variable-sized signal along with tready signal

Transfer takes place when both tvalid and tready are asserted



Topology of Connections Between HDL and AI Engine

Interconnecting AI Engine and HLS Kernel Blocks

AI Engine to HLS Kernel block: Connects output port of an AI Engine kernel and an input port of an HLS kernel

HLS Kernel to AI Engine block: Connects output port of an HLS kernel and an input port of an AI Engine kernel

Data reformatted to match the data type of the sink port

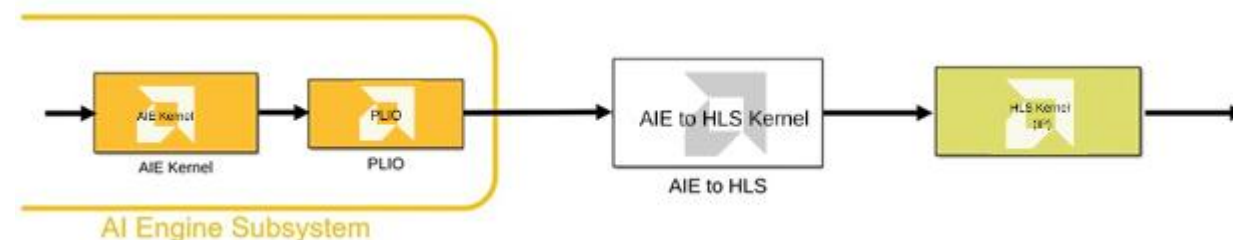
No data (information) is lost
Blocks are adjusting the data type and the number of samples

- Example: Interface block can reformat a signal carrying 64 INT8 values to a signal carrying 16 INT32 values
- Use of these blocks is not mandatory if the data types between the HLS kernel block and the AI Engine block match
- Available from the AMD Toolbox > Utilities > Connectors library

AI Engine to HLS Kernel Block

Reformats a signal driven by an AI Engine Kernel block or an AI Engine subsystem

- Double-click the block symbol to see the parameters of the AI Engine to HLS Kernel block
- Refer to the Graph block so that the resulting signal matches the data type and complexity required by the input of the HLS kernel block

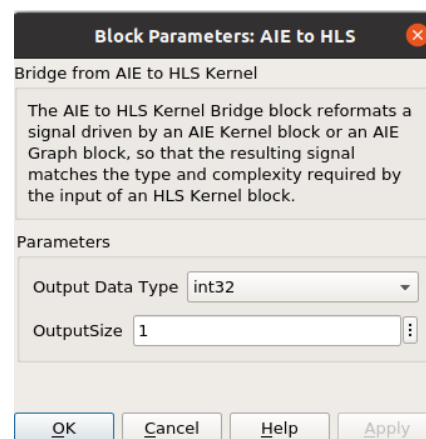


Topology of Connections Between AI Engine and HLS Kernel

Output Type

Possible values are:

- ap_axis<32>, ap_axis<64>, ap_axis<128>
- ap_axiu<32>, ap_axiu<64>, ap_axiu<128>
- ap_int<32>, ap_int<64>
- ap_uint<32>, ap_uint<64>
- int, long long, unsigned, unsigned long long



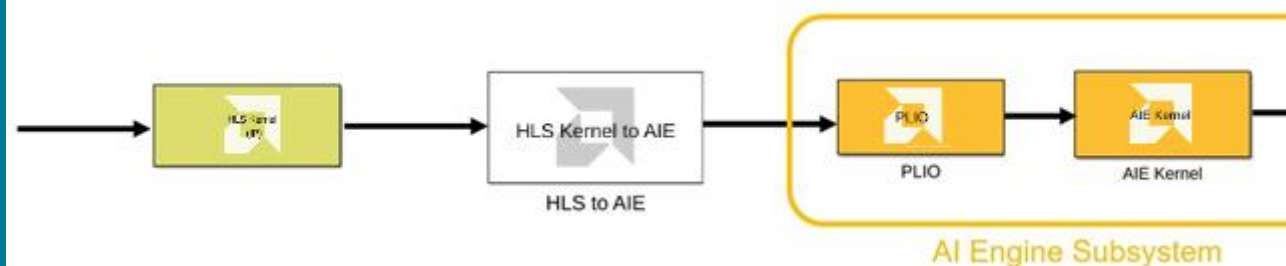
Output Size

- Output port is a variable-sized signal whose maximum size is specified by the Output Size parameter
- Default output size is 1

HLS Kernel to AI Engine Block

Reformats a signal driven by a port of an HLS Kernel block

- Resulting signal will match with the data type and complexity required by:
 - AI Engine kernel or
 - Input of an AI Engine Graph block
- Output port of this block is a variable-sized signal
- Double-click the HLS Kernel to AI Engine block symbol to see the parameters

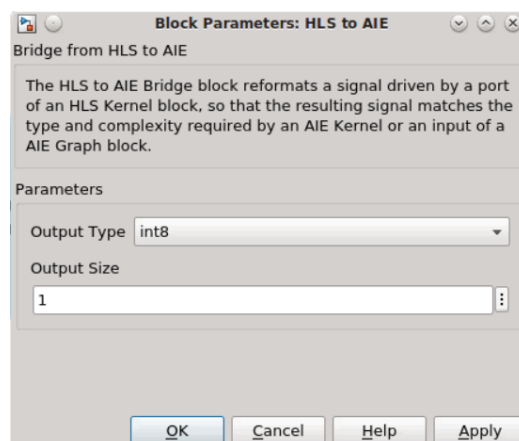


Topology of Connections Between HLS Kernel and AI Engine

Output Type

Possible values are:

- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- cint16, cint32
- sfix128, ufix128, float, cfloat



Output Size

- Output port is a variable-sized signal whose maximum size is specified by the Output Size parameter
- Default output size is 1

Agenda

Introduction

AMD Toolbox – Library Blocks

Create and Simulate an HDL Design

Create and Simulate an HLS Design

Create and Simulate an AI Engine Design

Create a Heterogeneous (AI Engine + PL) Design

Summary

Summary

01

AMD Vitis™ Model Composer is a model-based design tool that enables rapid design exploration within the MATLAB® MathWorks Simulink® environment

02

Vits Model Composer provides features such as:

- Analysis, debugging, and visualization
- Co-simulation of AI Engines and PL
- Code generation
- Validation of the design in hardware

03

Creating a Vitis Model Composer design consists of three steps:

- Adding blocks to a model
- Connecting the blocks
- Creating a top-level subsystem module

04

Run Simulink simulation for functional verification of the design

GENERAL DISCLOSURE AND ATTRIBUTION STATEMENT

The information contained herein is for informational purposes only and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18u.

©2025 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, UltraScale+, Versal, Vitis, Vivado, Zynq, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective owners. Certain AMD technologies may require third-party enablement or activation. Supported features may vary by operating system. Please confirm with the system manufacturer for specific features. No technology or product can be completely secure.

