



AI Engine Test Harness & Versal™ System Integration

Faisal El-Shabani – Technical Marketing Manager | DSP and AIE
Florent Werbrouck – Technical Marketing Manager | DSP and AIE

Objectives

During today's session, we will:

01

Describe the
Versal™ AI
Engine
development flow

02

Introduce AI
Engine Test
Harness

03

Demonstrate the
AI Engine Test
Harness using a
design example

04

Demonstrate
Versal™ system
integration using
the Vitis™ tool
flow

Agenda

-
1. **AI Engine Development Flow**
 2. AI Engine Test Harness
 3. Demo – Polyphase Channelizer on Test Harness
 4. Demo – Versal™ System Integration using the Vitis™ Tool Flow

AI Engine Development Flow

Recommended flow:

1

AI Engine Coding

1. AI Engine Kernels 2. AI Engine Graphs

2

Functional Verification

x86 Simulation

3

Cycle-Approximate Verification (SystemC Model)

AIE Simulation

4

System-Level Emulation (PS+PL+AIE)

Hardware Emulation

5

Hardware Verification

Hardware Run

AI Engine Development Flow

Recommended flow:

- Challenges in Step 3
 - AIE simulation can become slow for complex graphs with large datasets
 - Limited verification scope
 - Design effort to move from Step 3 to Step 5 is high

1

AI Engine Coding

1. AI Engine Kernels 2. AI Engine Graphs

2

Functional Verification

x86 Simulation

3

Cycle-Approximate Verification (SystemC Model) AIE Simulation

4

System-Level Emulation (PS+PL+AIE) Hardware Emulation

5

Hardware Verification Hardware Run

AI Engine Development Flow

Recommended flow:

- Solution:
 - AI Engine Test Harness
 - Transition seamlessly from AIE simulation to hardware

1

AI Engine Coding

1. AI Engine Kernels 2. AI Engine Graphs

2

Functional Verification

x86 Simulation

3

Cycle-Approximate Verification (SystemC Model)

AIE Simulation

4

***AI Design Only* in Hardware**

AIE Test Harness

5

System-Level Emulation (PS+PL+AIE)

Hardware Emulation

6

Hardware Verification

Hardware Run

Agenda

-
1. AI Engine Development Flow
 - 2. AI Engine Test Harness**
 3. Demo – Polyphase Channelizer on Test Harness
 4. Demo – Versal™ System Integration using the Vitis™ Tool Flow

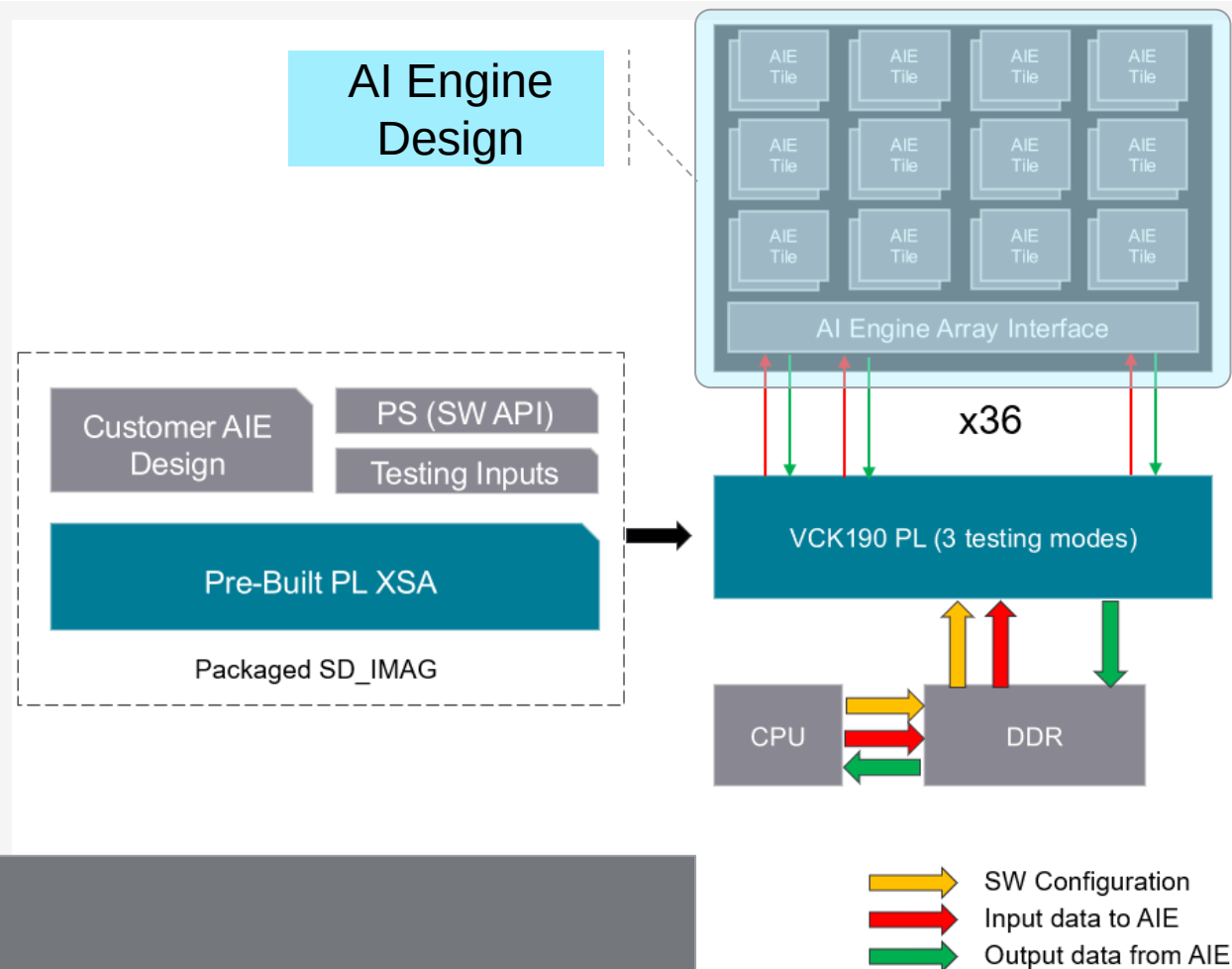
What is the AI Engine Test Harness?

AI Engine designers can do quick build and on-board tests with VCK190

- Focus only on AI Engine design development
- Verify design on board using Test Harness
- Test Harness speeds up your development
 - Precompiled data movers in programmable logic (PL) deliver stimulus and collect results from AIE array
 - Software APIs on processor subsystem (PS) initialize device and run tests
 - Packages design into boot image for on-board tests

Save the most time-consuming part

- Fast compilation and runtime of designs



Features and Requirements

Features

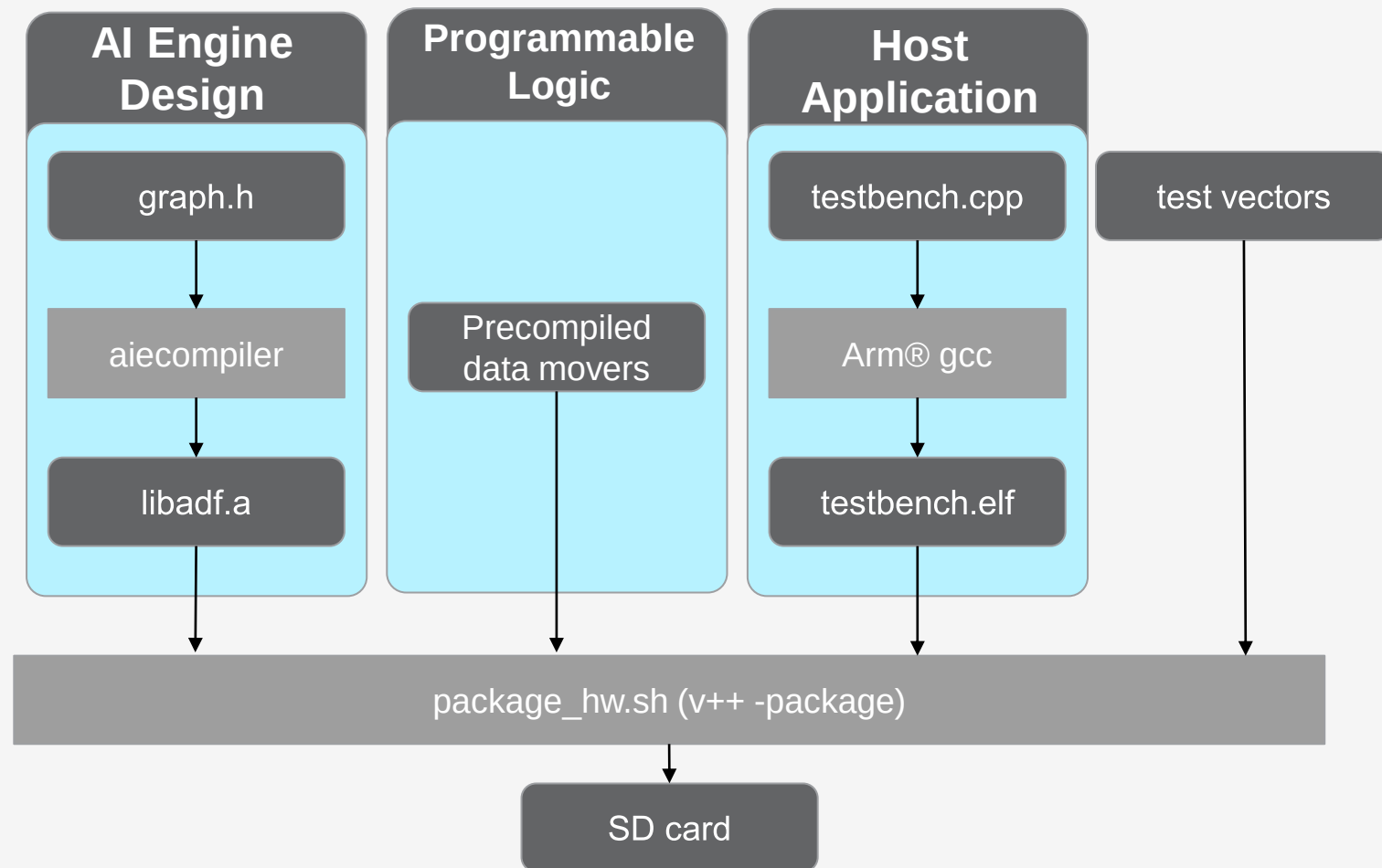
- 72-channel DMA engine (36 Tx, 36 Rx, each running at 5GB/s)
- Performance counters to measure throughput
- Verify correctness against golden I/O files
- Fast build time using precompiled .xsa

Current Requirements

- Data size $\leq 128\text{KB}$ and must be a multiple of 16 bytes
- Test Harness is targeted for the VCK190 board
- AIE graphs must use PLIO interfaces (*GMIO interfaces are not supported*)

Build Flow for Using the AI Engine Test Harness

- graph.h
 - Original AIE graph with minor instrumentations
- Precompiled data movers
 - Moves data from DDR memory to AI Engine graph and back
- testbench.cpp
 - Program using Test Harness APIs to drive the HW tests
- Test vectors
 - Test vectors can be loaded from files or dynamically generated by testbench.cpp



Agenda

-
1. AI Engine Development Flow
 2. AI Engine Test Harness
 - 3. Demo – Polyphase Channelizer on Test Harness**
 4. Demo – Versal™ System Integration using the Vitis™ Tool Flow

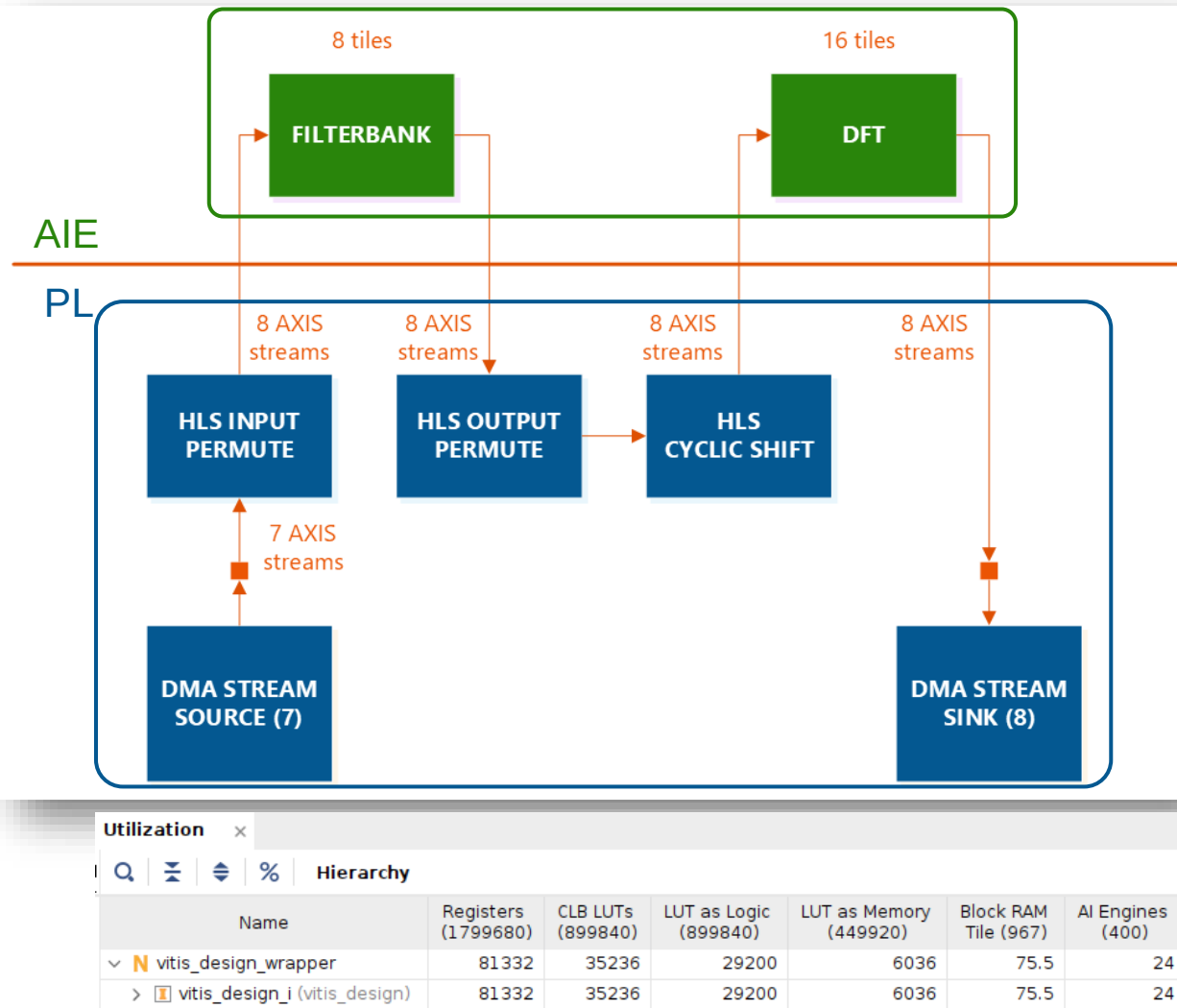
Background – Polyphase Channelizer Design

System Requirement

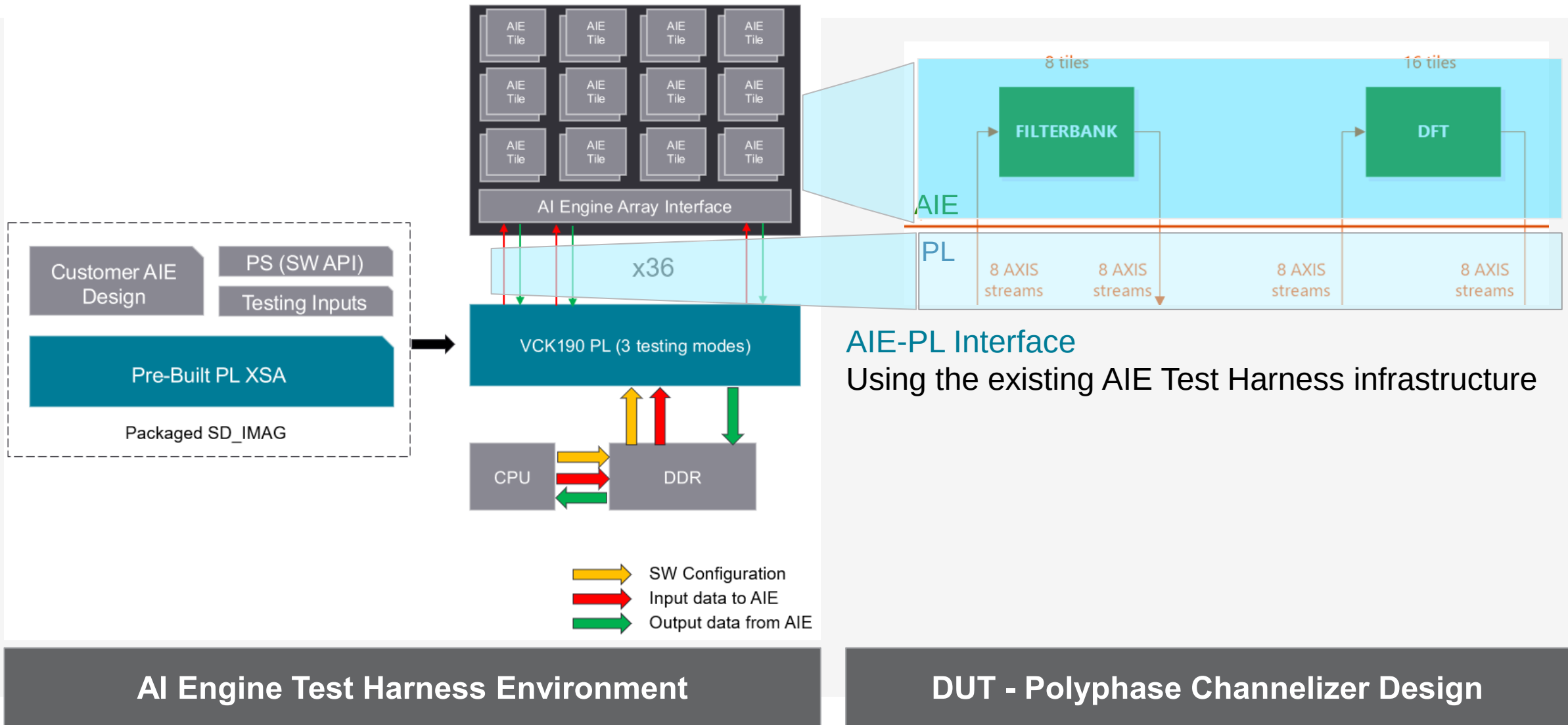
Polyphase Channelizer

Sampling Rate (Fs)	10 Gsps, AIE clock (Fclk) = 1.25 GHz
SSR	8
FILTERBANK	128-tap, 16 channels, 8 AI Engine tiles
DFT	16 AI Engine tiles

HLS blocks responsible for data buffering to support required operations, as well as transferring data from memory to design and back. These are mapped to programmable logic (PL).



Polyphase Channelizer on Test Harness



AI Engine Test Harness Environment

DUT - Polyphase Channelizer Design

Polyphase Channelizer on Test Harness

User Changes	HLS-Based Test Harness Design on PL	AXI Streams Between DDR and AI Engine	36 PLIO Input 36 PLIO Output
	Integrate User Graph Design	Include Test Harness Header Use from Available PLIOs (Data Movers)	

Polyphase Channelizer Design

Filterbank Kernel	8 PLIO Input	8 PLIO Output
DFT Kernel	8 PLIO Input	8 PLIO Output

Polyphase Channelizer (DUT)	m16_ssr8_graph.h
-----------------------------	------------------

```
class m16_ssr8_graph : public graph {
public:
    std::array<input_plio, 8> dft_i;
    std::array<output_plio, 8> dft_o;
    std::array<input_plio, 8> fbank_i;
    std::array<output_plio, 8> fbank_o;

    // Position filterbank to the right side of DFT:
    m16_ssr8_dft_graph dut_dft;
    m16_ssr8_filterbank_graph dut_fbank;

    // Constructor:
    m16_ssr8_graph(void) : dut_dft(), dut_fbank() {
        // Connections:
        for (int ii = 0; ii < 8; ii++) {
            // Note: There are no filenames on these ::create() routines because we either connect these up to
            // the actual PL HLS blocks in 'system.cfg' or External Traffic Generators for debug
            std::string plio_fbank_i = vck190_test_harness::in_names[16 + ii * 2];
            std::string plio_fbank_o = vck190_test_harness::out_names[16 + ii * 2 + 1];
            std::string plio_dft_i = vck190_test_harness::in_names[ii * 2];
            std::string plio_dft_o = vck190_test_harness::out_names[ii * 2 + 1];

            std::string fname_fbank_i = "data/fir_i_" + std::to_string(ii) + ".txt";
            std::string fname_fbank_o = "data/fir_o_" + std::to_string(ii) + ".txt";
            std::string fname_dft_i = "data/dft_i_" + std::to_string(ii) + ".txt";
            std::string fname_dft_o = "data/dft_o_" + std::to_string(ii) + ".txt";
            fbank_i[ii] = input_plio::create(plio_fbank_i, plio_128_bits, fname_fbank_i);
            fbank_o[ii] = output_plio::create(plio_fbank_o, plio_128_bits, fname_fbank_o);
            dft_i[ii] = input_plio::create(plio_dft_i, plio_128_bits, fname_dft_i);
            dft_o[ii] = output_plio::create(plio_dft_o, plio_128_bits, fname_dft_o);

            connect<stream>(fbank_i[ii].out[0], dut_fbank.sig_i[ii]);
            connect<stream>(dut_fbank.sig_o[ii], fbank_o[ii].in[0]);
            connect<stream>(dft_i[ii].out[0], dut_dft.sig_i[ii]);
            connect<stream>(dut_dft.sig_o[ii], dft_o[ii].in[0]);
        }
    }
};
```

Polyphase Channelizer on Test Harness

Polyphase Channelizer Design

HLS-Based Test
Harness Design on PL

AXI Streams Between
DDR and AI Engine

36 PLIO Input
36 PLIO Output

Filterbank Kernel
DFT Kernel

8 PLIO Input | 8 PLIO Output
8 PLIO Input | 8 PLIO Output

User Changes

Integrate User Graph
Design

Include Test Harness Header
Use from Available PLIOs (Data Movers)

Polyphase
Channelizer (DUT)

m16_ssr8_graph.h

(Software API
Host Code)

To Run the Design on
Board

- Initialize device and load xclbin
- Set data to feed AIE and fetch from AIE
- Run user graph

Polyphase
Channelizer
(Host Code)

host.cpp

```
read_data_from_file(in_dft_0_path, in_dft_0);
read_data_from_file(in_dft_1_path, in_dft_1);
read_data_from_file(in_dft_2_path, in_dft_2);
read_data_from_file(in_dft_3_path, in_dft_3);
read_data_from_file(in_dft_4_path, in_dft_4);
read_data_from_file(in_dft_5_path, in_dft_5);
read_data_from_file(in_dft_6_path, in_dft_6);
read_data_from_file(in_dft_7_path, in_dft_7);
read_data_from_file(out_dft_0_path, golden_dft_0);
read_data_from_file(out_dft_1_path, golden_dft_1);
read_data_from_file(out_dft_2_path, golden_dft_2);
read_data_from_file(out_dft_3_path, golden_dft_3);
read_data_from_file(out_dft_4_path, golden_dft_4);
read_data_from_file(out_dft_5_path, golden_dft_5);
read_data_from_file(out_dft_6_path, golden_dft_6);
read_data_from_file(out_dft_7_path, golden_dft_7);
```

```
read_data_from_file(out_dft_5_path, golden_dft_5);
read_data_from_file(out_dft_6_path, golden_dft_6);
read_data_from_file(out_dft_7_path, golden_dft_7);
read_data_from_file(in_fir_0_path, in_fir_0);
read_data_from_file(in_fir_1_path, in_fir_1);
read_data_from_file(in_fir_2_path, in_fir_2);
read_data_from_file(in_fir_3_path, in_fir_3);
read_data_from_file(in_fir_4_path, in_fir_4);
read_data_from_file(in_fir_5_path, in_fir_5);
read_data_from_file(in_fir_6_path, in_fir_6);
read_data_from_file(in_fir_7_path, in_fir_7);
read_data_from_file(out_fir_0_path, golden_fir_0);
read_data_from_file(out_fir_1_path, golden_fir_1);
read_data_from_file(out_fir_2_path, golden_fir_2);
```

```
// run test with test harness
test_harness_mgr mgr(0, argv[1], {"aie_dut"});
std::vector<test_harness_args> args;
args.push_back({channel_index(PLIO_01_TO_AIE), in_sz, 1, 0, (char*)in_dft_0.data()});
args.push_back({channel_index(PLIO_03_TO_AIE), in_sz, 1, 0, (char*)in_dft_1.data()});
args.push_back({channel_index(PLIO_05_TO_AIE), in_sz, 1, 0, (char*)in_dft_2.data()});
args.push_back({channel_index(PLIO_07_TO_AIE), in_sz, 1, 0, (char*)in_dft_3.data()});
args.push_back({channel_index(PLIO_09_TO_AIE), in_sz, 1, 0, (char*)in_dft_4.data()});
args.push_back({channel_index(PLIO_11_TO_AIE), in_sz, 1, 0, (char*)in_dft_5.data()});
args.push_back({channel_index(PLIO_13_TO_AIE), in_sz, 1, 0, (char*)in_dft_6.data()});
args.push_back({channel_index(PLIO_15_TO_AIE), in_sz, 1, 0, (char*)in_dft_7.data()});
args.push_back({channel_index(PLIO_17_TO_AIE), in_sz, 1, 0, (char*)in_fir_0.data()});
args.push_back({channel_index(PLIO_19_TO_AIE), in_sz, 1, 0, (char*)in_fir_1.data()});
args.push_back({channel_index(PLIO_21_TO_AIE), in_sz, 1, 0, (char*)in_fir_2.data()});
```

How to Setup the Test Harness Environment?



AI Test Harness Environment

<https://github.com/Xilinx/AI-Engine-Test-Harness>



Setup the Environment

Tool requirement: Vitis™ 2023.1 software

Platform: Set up your PLATFORM_REPO_PATHS environment variable to VCK190 platform



Example Design – Polyphase Channelizer

```
% source setup.sh
% cd /examples/channelizer/
```

AI-Engine-Test-Harness Public		Watch 2
main	2 branches	1 tag
Go to file	Add file	Code
leol-xilinx Update top readme for 36x PLIO		885c942 on Aug 9 88 commits
bin	Update 36x PLIO XSA to Xilinx public download	2 months ago
cfg	Design update for 36x PLIO	2 months ago
docs	Fix typo in step-by-step example	last month
examples	Fix pl-freq & cleanup issues in Makefile of each example	2 months ago
include	Design update for 36x PLIO	2 months ago
src	Re-triggering Jenkins regression for polarchip	2 months ago
test_harness	Explicit clock freq @ 312.5MHz for v++ link	2 months ago
Jenkinsfile	Fix reg 0502 (#46)	5 months ago
LICENSE	Update MIT license to MIT X11 license (#16)	5 months ago
README.md	Update top readme for 36x PLIO	last month
setup.sh	GCC LIB setup notice to setup.sh (#28)	5 months ago

Polyphase Channelizer on Test Harness – Hardware Setup



Build the Project – Generate PDI

```
% make cleanall run TARGET=hw
```

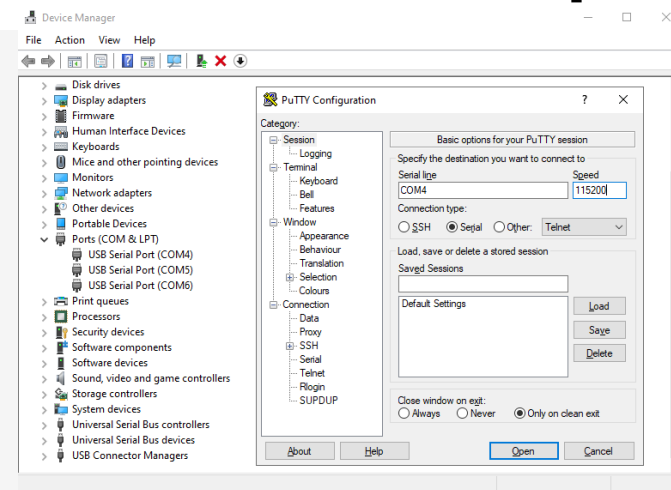


Flash Program

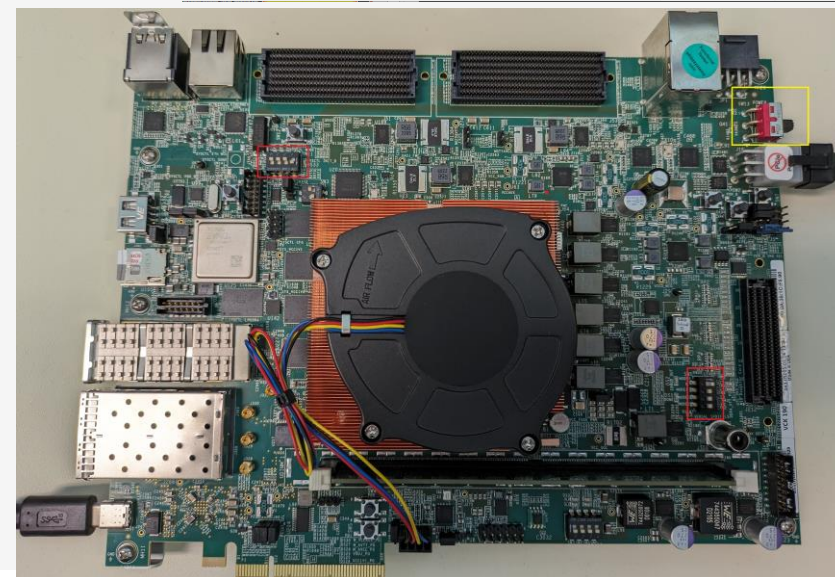
SD card image: /examples/channelizer/package_hw/sd_card.img

Use PuTTY: Connect to VCK190 board using USB serial port
Use 115200

VCK190 board settings:
Set the switches on the board to boot using sd_card as shown in red, then flip switch shown in yellow to turn on the board



PuTTY



VCK190

Polyphase Channelizer on Test Harness – Hardware Validation



VCK190 Board Power ON and Run Design

```
$ username: petalinux
$ password: petalinux
$ sudo su - root
Enter Password as petalinux

$ cd /run/media/mmcblk0p1/
$ source ./run_script.sh
```

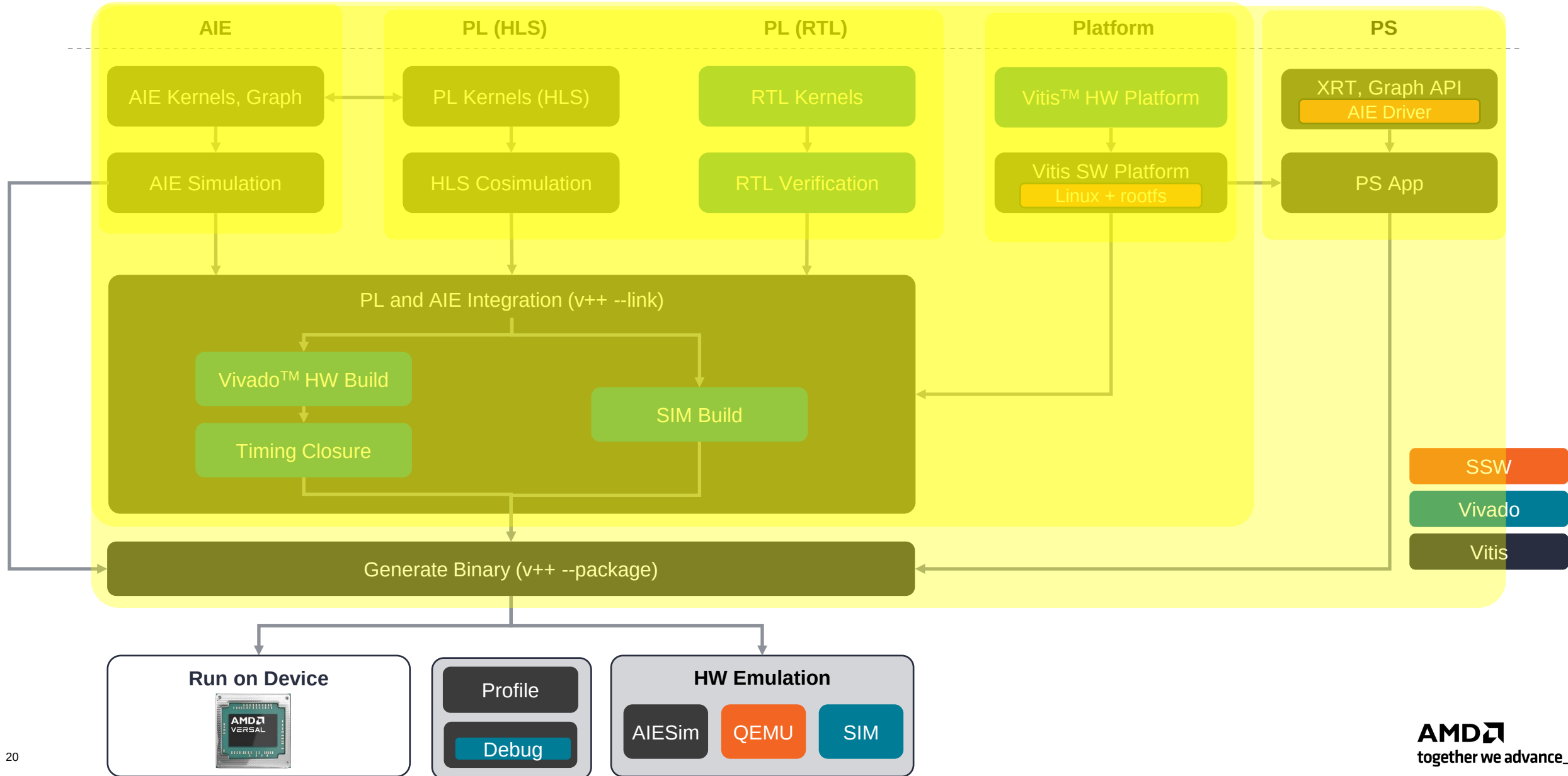
- This shall execute APIs existing in host.cpp
 - Counts the number of cycles to pump I/O data through PLIOs
 - Confirms completion of execution on hardware with a “TEST PASSED” message

```
PLIO_1_TO_AIE starts from cycle[0], ends at cycle[4804].
PLIO_3_TO_AIE starts from cycle[0], ends at cycle[4805].
PLIO_5_TO_AIE starts from cycle[0], ends at cycle[4810].
PLIO_7_TO_AIE starts from cycle[0], ends at cycle[4813].
PLIO_9_TO_AIE starts from cycle[0], ends at cycle[4817].
PLIO_11_TO_AIE starts from cycle[0], ends at cycle[4814].
PLIO_13_TO_AIE starts from cycle[0], ends at cycle[4824].
PLIO_15_TO_AIE starts from cycle[0], ends at cycle[4825].
PLIO_17_TO_AIE starts from cycle[0], ends at cycle[4548].
PLIO_19_TO_AIE starts from cycle[0], ends at cycle[4282].
PLIO_21_TO_AIE starts from cycle[0], ends at cycle[4282].
PLIO_23_TO_AIE starts from cycle[0], ends at cycle[4282].
PLIO_25_TO_AIE starts from cycle[0], ends at cycle[4283].
PLIO_27_TO_AIE starts from cycle[0], ends at cycle[4283].
PLIO_29_TO_AIE starts from cycle[0], ends at cycle[4283].
PLIO_31_TO_AIE starts from cycle[0], ends at cycle[4282].
PLIO_2_FROM_AIE starts from cycle[0], ends at cycle[4933].
PLIO_4_FROM_AIE starts from cycle[0], ends at cycle[4932].
PLIO_6_FROM_AIE starts from cycle[0], ends at cycle[4906].
PLIO_8_FROM_AIE starts from cycle[0], ends at cycle[4905].
PLIO_10_FROM_AIE starts from cycle[0], ends at cycle[4893].
PLIO_12_FROM_AIE starts from cycle[0], ends at cycle[4888].
PLIO_14_FROM_AIE starts from cycle[0], ends at cycle[4913].
PLIO_16_FROM_AIE starts from cycle[0], ends at cycle[4911].
PLIO_18_FROM_AIE starts from cycle[0], ends at cycle[4577].
PLIO_20_FROM_AIE starts from cycle[0], ends at cycle[4311].
PLIO_22_FROM_AIE starts from cycle[0], ends at cycle[4312].
PLIO_24_FROM_AIE starts from cycle[0], ends at cycle[4310].
PLIO_26_FROM_AIE starts from cycle[0], ends at cycle[4309].
PLIO_28_FROM_AIE starts from cycle[0], ends at cycle[4309].
PLIO_30_FROM_AIE starts from cycle[0], ends at cycle[4309].
PLIO_32_FROM_AIE starts from cycle[0], ends at cycle[4309].
TEST PASSED
INFO: TEST PASSED, RC=0
INFO: Embedded host run completed.
```

Agenda

-
1. AI Engine Development Flow
 2. AI Engine Test Harness
 3. Demo – Polyphase Channelizer on Test Harness
 4. **Demo – Versal™ System Integration using the Vitis™ Tool Flow**

Vitis™ Tool Flow for Versal™ Devices



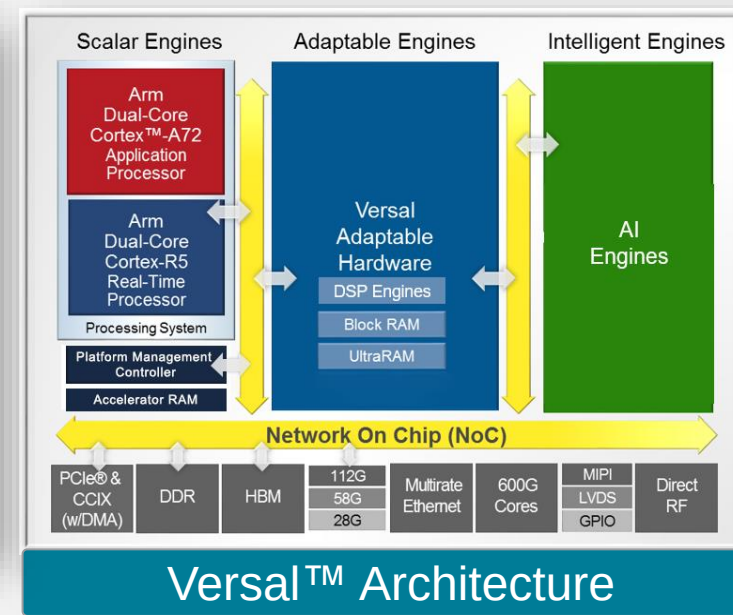
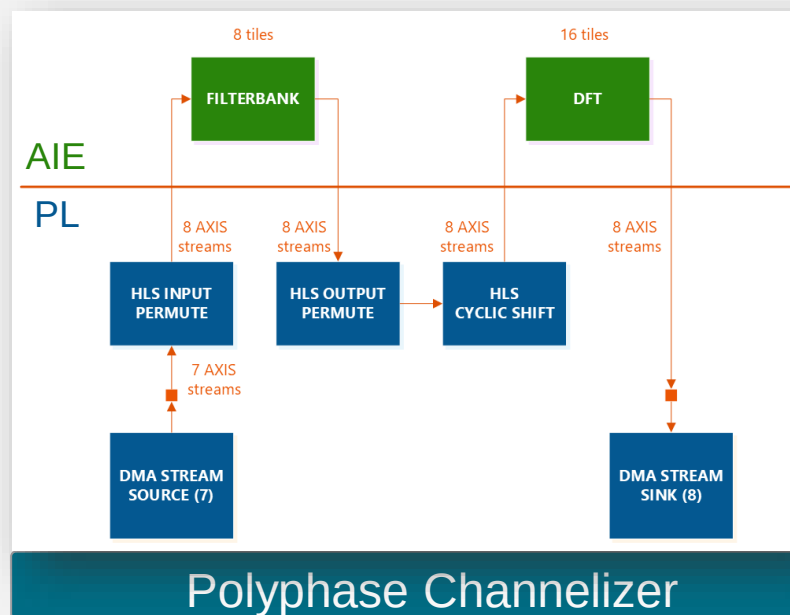
Versal™ System Integration – Vitis™ Tool Flow

AI Engine Design

PL Design

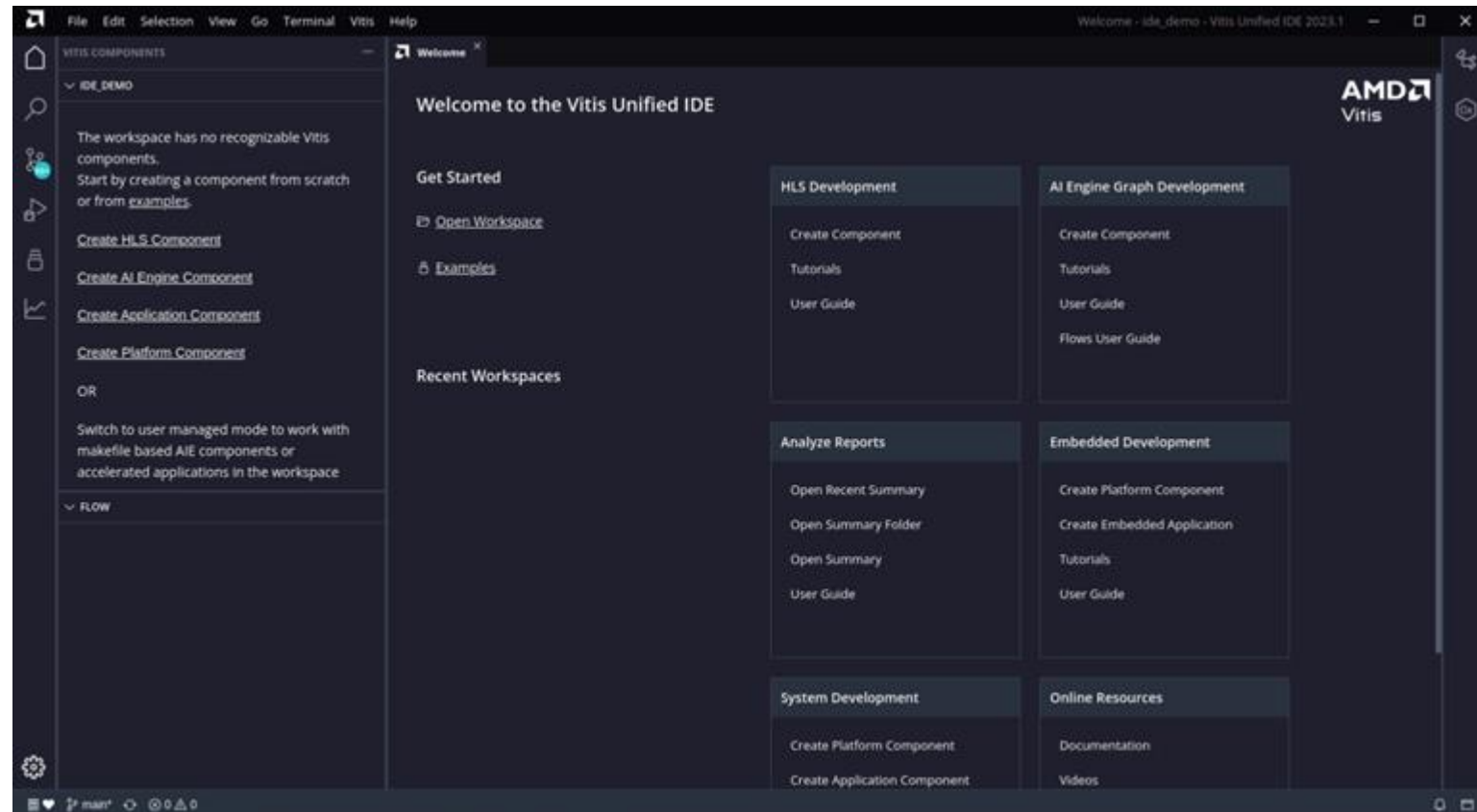
Application (PS – Host Code)

System Project



Vitis™ Unified IDE – Getting Started

- Install Vitis™ 2023.1 software
 - Clone <https://github.com/Xilinx/Vitis-Tutorials>
 - Launch Vitis GUI
- `vitis --new -w <PATH_TO_WORKSPACE>`
- Design files shall be sourced from
Vitis-Tutorials/AI_Engine_Development/
Design_Tutorials/04-Polyphase-Channelizer



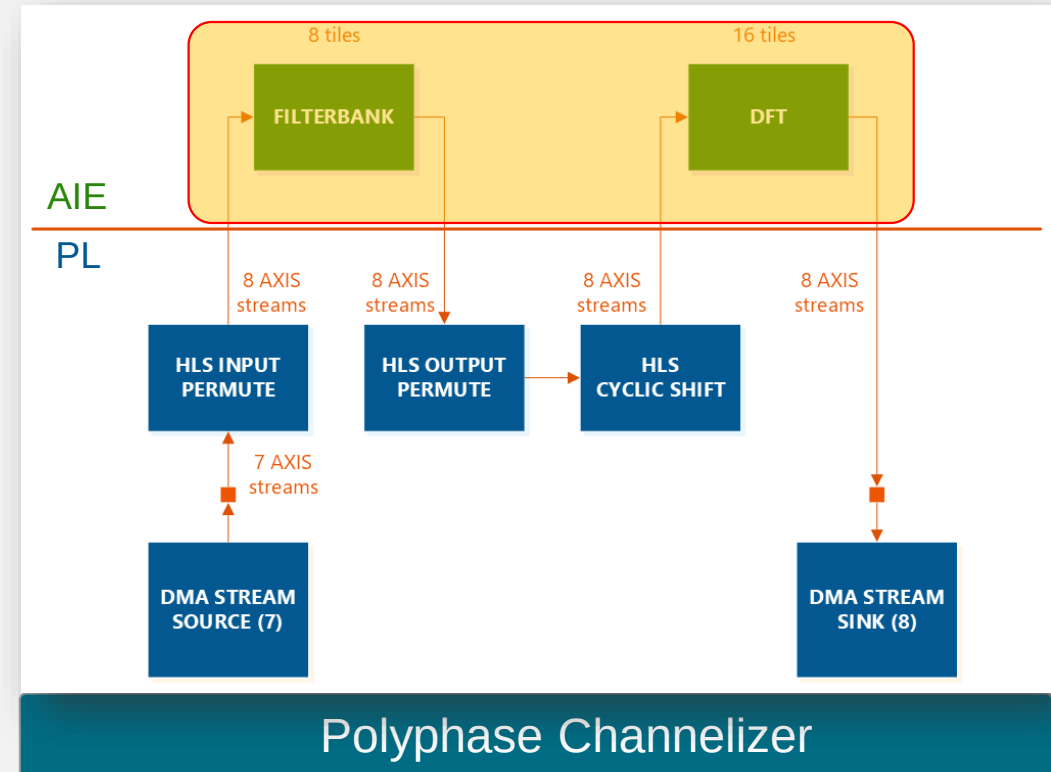
Versal™ System Integration – Vitis™ Tool Flow

AI Engine Design

PL Design

Application (PS – Host Code)

System Project



Vitis™ Unified IDE – AIE Development Flow

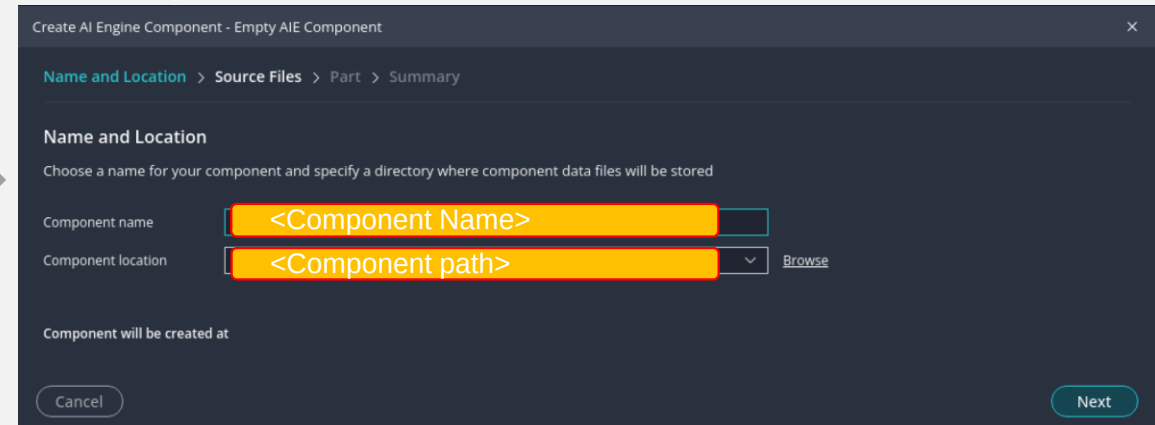
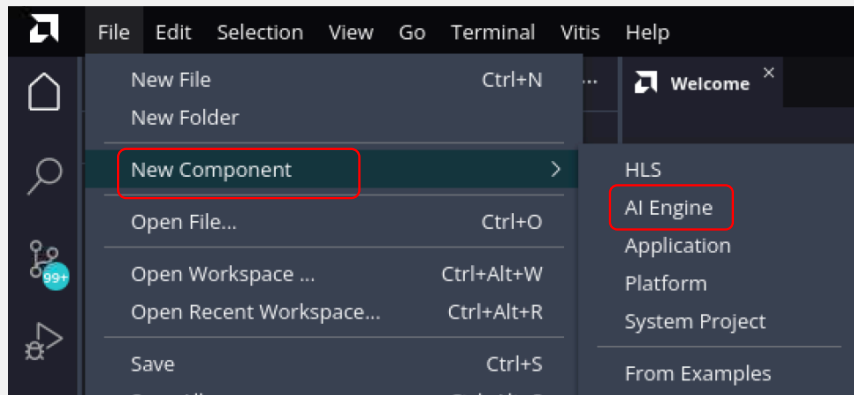
Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine Design

Step 4:
Analyze the Reports

- Create AIE component
- Enter the component name



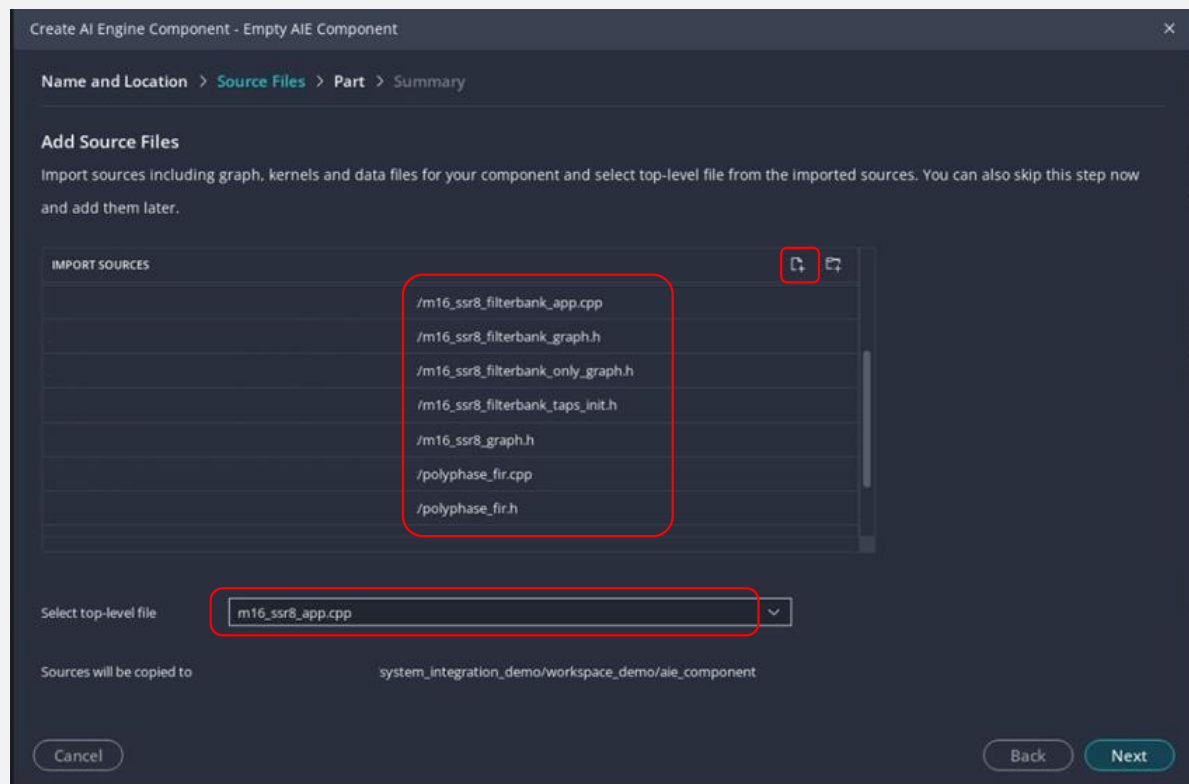
Vitis™ Unified IDE – AIE Development Flow

Step 1:
Create AIE Comp

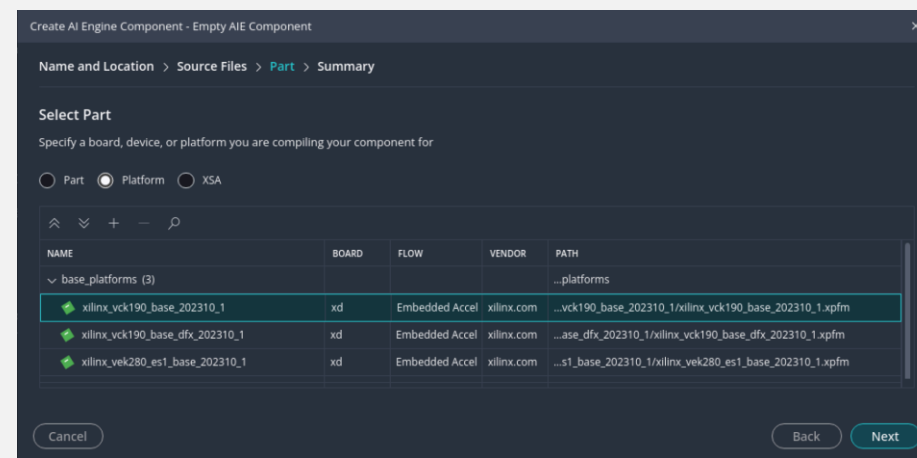
Step 2:
Add Design Files

Step 3: Build AI
Engine Design

Step 4:
Analyze the Reports



- Add the files by clicking 
- Polyphase Channelizer design files
- Select the platform



Vitis™ Unified IDE – AIE Development Flow

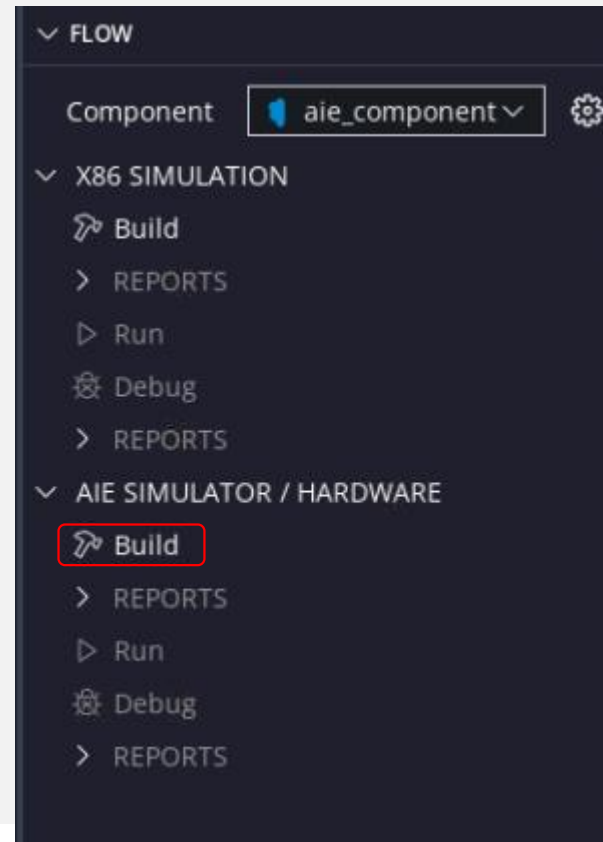
Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine Design

Step 4:
Analyze the Reports

- Under AIE SIMULATOR/HARDWARE
 - Click **Build** to compile the graph



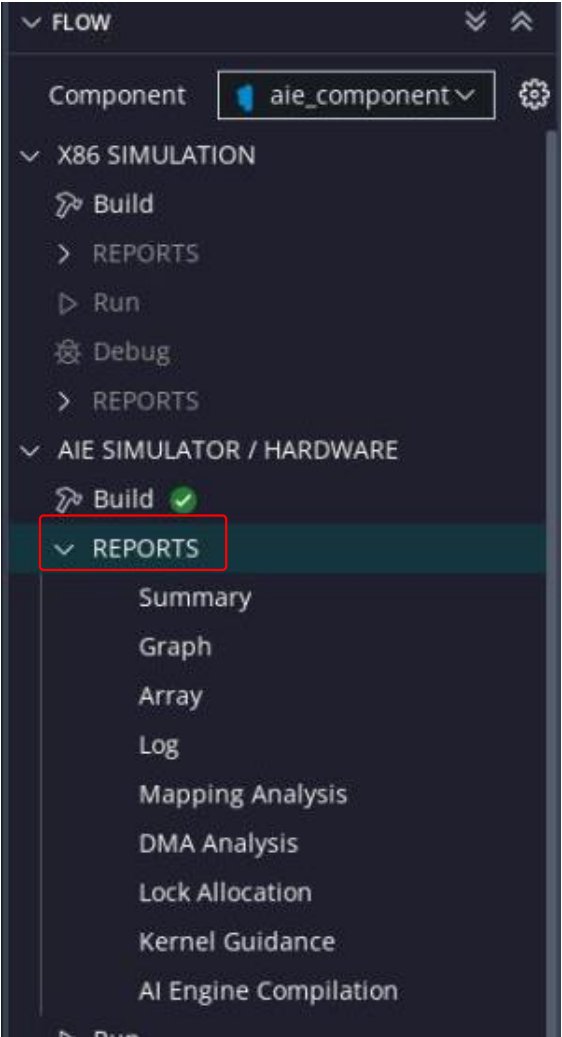
Vitis™ Unified IDE – AIE Development Flow

Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine Design

Step 4:
Analyze the Reports



Vitis™ Unified IDE – AIE Development Flow

Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine design

Step 4:
Analyze the Reports

VITIS COMPONENTS

04-POLYPHASE-CHANNELIZER

ale_component [AI Engine]

> Settings

> Includes

> Sources

> Output

FLOW

Component

ale_component

X86 SIMULATION

Build

REPORTS

Run

Debug

REPORTS

AIE SIMULATOR / HARDWARE

Build

REPORTS

Summary

Graph

Array

Log

Build - ale_component - HARDWARE

Summary

AI Engine Build | ale_component (HARDWARE)

Completed

Version: Vitis AI Engine Compiler Release 2023.1. SW Build 3855003 on 2023-05-04-23:53:06

Log Started: September 13, 2023 08:10:57 Completed: September 13, 2023 08:14:42 Elapsed: 04m 46s

Device: VC1902 AI Engine Frequency: 1250 MHz

File: /Vitis-Tutorials/AI_Engine_Development/Design_Tutorials/04-Polyphase-Channelizer/ale_component/build/hw/Work/m16_ssr8_app.aiecompile_summary

AI Engine Compilation

Completed 24 of 24

AI Engine Resource Utilization

Tiles used for AI Engine Kernels: 24 of 400 (6.00 %)

Tiles used for Buffers: 22 of 400 (5.50 %)

Tiles used for Stream Interconnect: 32 of 450 (7.11 %)

DMA FIFO Buffers: 0

Interface Channels used for ADF Input/Output: 32 (PLIO: 32)

Interface Channels used for Trace data: 0

AI Engine Kernels

NAME	INSTANCES
dft_1xN_middle<clnt16, clnt16, cacc48, 8192>::run_...	8
polyphase_fir<7, 8192>::run_i	1
polyphase_fir<0, 8192>::run_i	1
polyphase_fir<1, 8192>::run_i	1
polyphase_fir<2, 8192>::run_i	1

Vitis™ Unified IDE – AIE Development Flow

Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine design

Step 4:
Analyze the Reports

VITIS COMPONENTS

WORKSPACE_DEMO

aie_component [AI Engine]

Settings

Includes

Sources

Output

app_component [Application]

cyclic_shift [HLS]

dma_sink [HLS]

FLOW

Component

aie_component

X86 SIMULATION

Build

REPORTS

Run

Debug

REPORTS

AIE SIMULATOR / HARDWARE

Build

REPORTS

Summary

Graph

Array

Log

Mapping Analysis

DMA Analysis

Lock Allocation

Kernel Guidance

AI Engine Compilation

Run

Debug

component

vitis-comp.json dma_sink

vitis-comp.json dma_source

vitis-comp.json cyclic_shift

vitis-comp.json input_permute

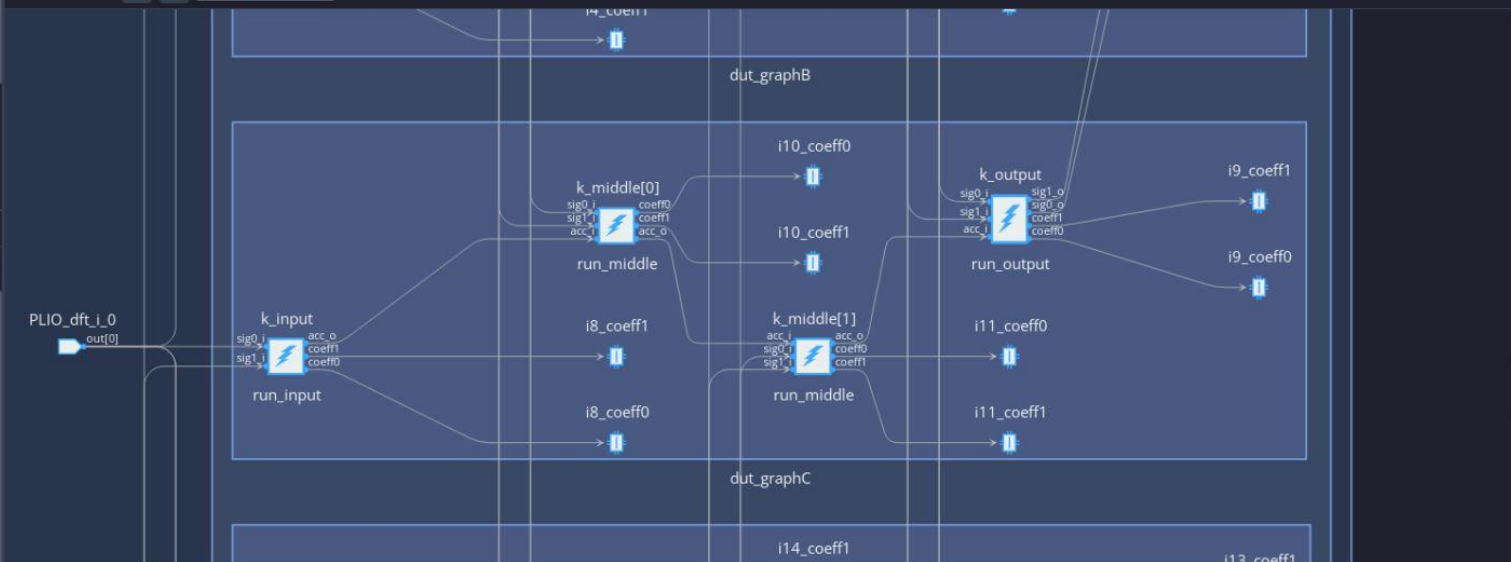
vitis-sys.json

Build - aie_component - HARDWARE

vitis-coi

Graph

Subgraph View



Kernels

I/O

Buffers

Ports

Nets

Tiles

Interface Channels

GRAPH INSTANCE	ID	AI ENGINE KERNEL	SOURCE	COLUMN	ROW	SCHEDULE	RUNTIME RATIO	GRAPH SOURCE
▼ aie_dut (2)								
▼ dut_fbanks (8)								
▼ dut_graph1 (1)								
k_fir	i17	polyphase_fir<1, 8192>::run_1	polyphase_fir.cpp	22	1	0	0.899	polyphase_fir_graph.h:33:13
▼ dut_graph0 (1)								
k_fir	i16	polyphase_fir<0, 8192>::run_1	polyphase_fir.cpp	22	0	0	0.899	polyphase_fir_graph.h:33:13

Vitis™ Unified IDE – AIE Development Flow

Step 1:
Create AIE Comp

Step 2:
Add Design Files

Step 3: Build AI
Engine design

Step 4:
Analyze the Reports

The screenshot displays the Vitis Unified IDE interface. The top menu bar includes File, Edit, Selection, View, Go, Terminal, Vitis, and Help. The left sidebar shows the project structure under 'VITIS COMPONENTS' and 'FLOW'. The 'FLOW' section is expanded, showing 'Component' as 'aie_component'. Under 'AIE SIMULATOR / HARDWARE', the 'Build' status is green, and the 'REPORTS' section is expanded, with 'Array' highlighted. The central canvas shows a hardware design on a grid, with two clusters of components highlighted in green and yellow. The bottom panel displays a table of kernels.

GRAPH INSTANCE	ID	AI ENGINE KERNEL	SOURCE	COLUMN	ROW	SCHEDULE	RUNTIME RATIO	GRAPH SOURCE
▼ aie_dut (2)								
▼ dut_fbanks (8)								
▼ dut_graph1 (1)								
k_fir	117	polyphase_fir<1, 8192>::run_1	polyphase_fir.cpp	22	1	0	0.899	polyphase_fir_graph.h:33:13
▼ dut_graph0 (1)								
k_fir	116	polyphase_fir<0, 8192>::run_1	polyphase_fir.cpp	22	0	0	0.899	polyphase_fir_graph.h:33:13

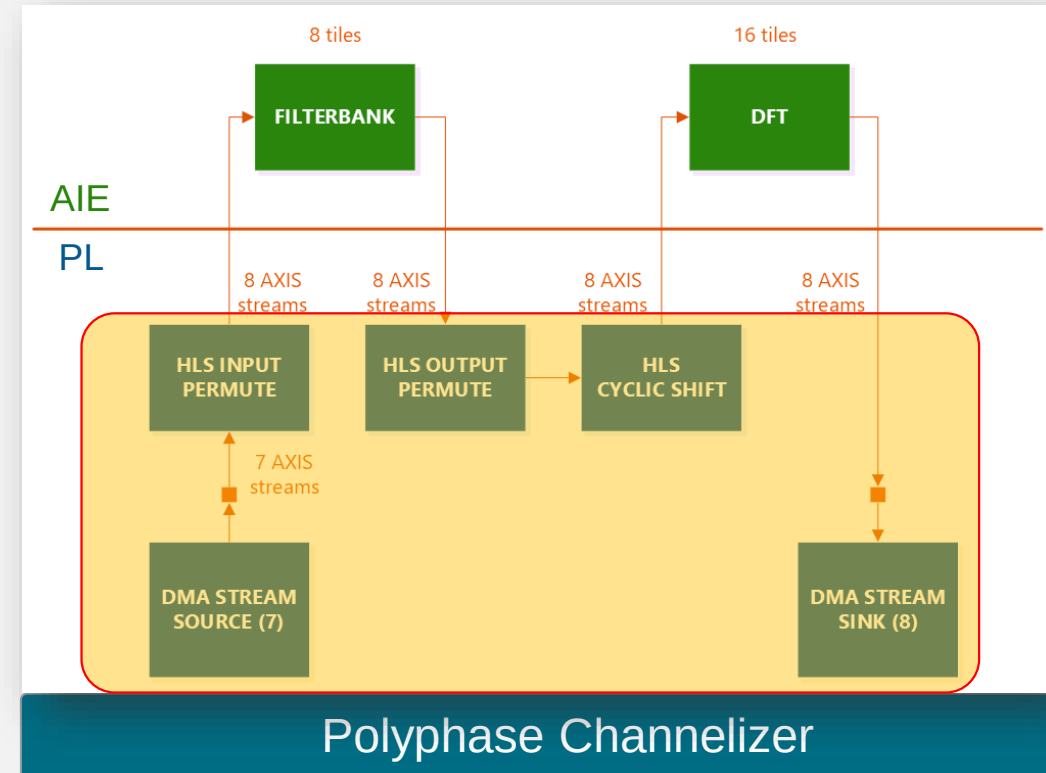
Versal™ System Integration – Vitis™ Tool Flow

AI Engine Design

PL Design

Application (PS – Host Code)

System Project



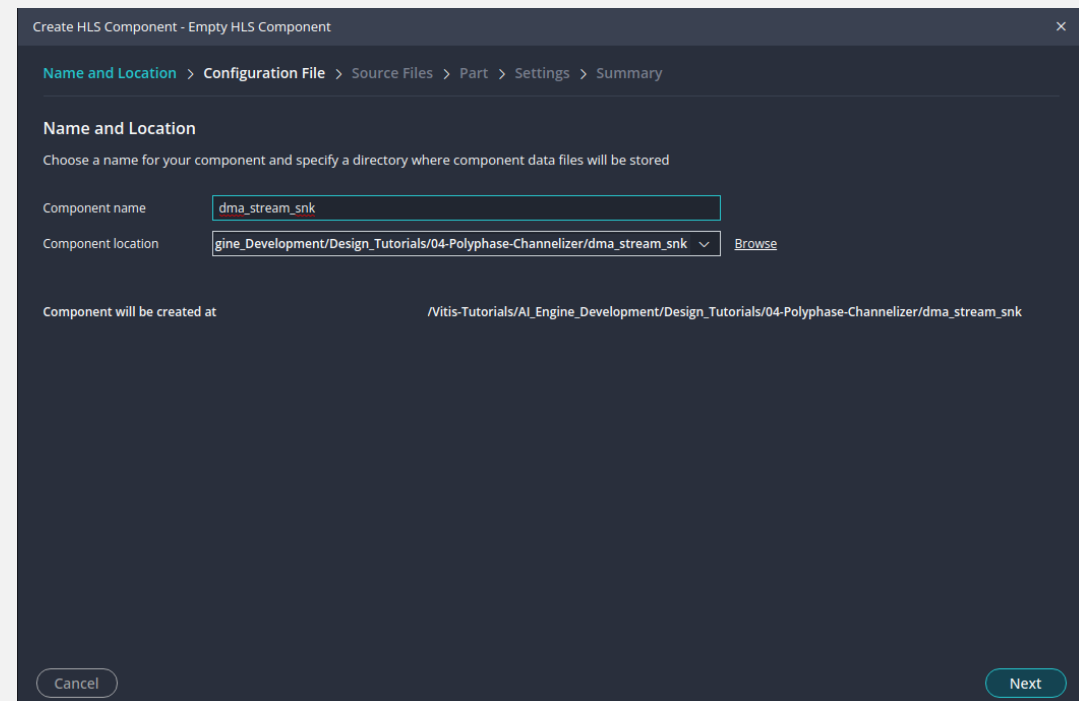
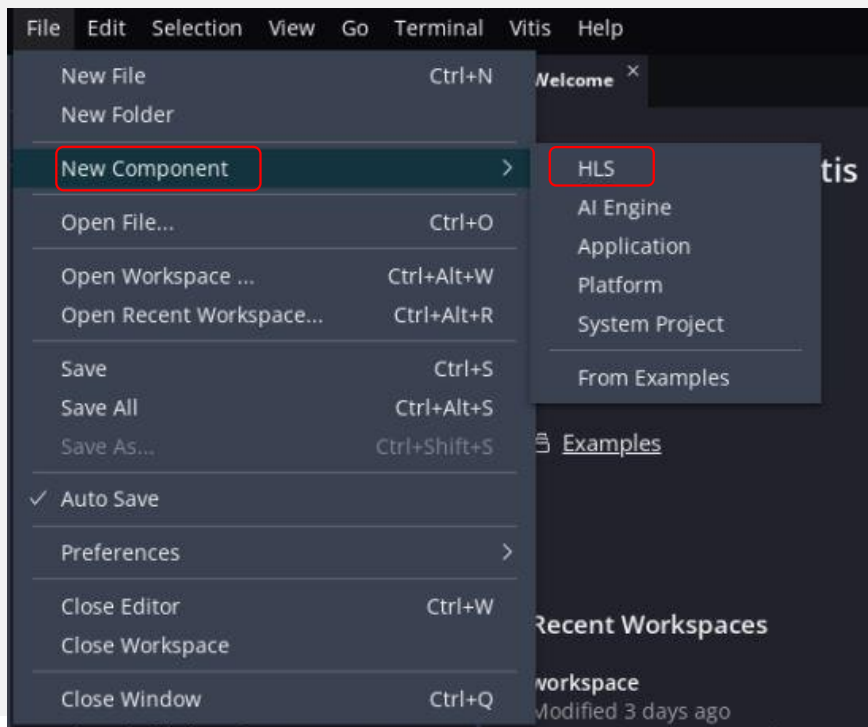
Vitis™ Unified IDE – HLS Development Flow

Step 1:
Create HLS Comp

Step 2:
Add Design Files

Step 3:
Build the Comp

- Create HLS component
- Enter the component name



Vitis™ Unified IDE – HLS Development Flow

Step 1:
Create HLS Comp

Step 2:
Add Design Files

Step 3:
Build the Comp

Add the design files | Select the platform | Select the target flow

Create HLS Component - Empty HLS Component

Name and Location > Configuration File > Source Files > Part > Settings > Summary

Add Source Files

Specify design, test bench files and flags for your component. You can also skip this step now and add sources later

DESIGN FILES

FILE(S)	CFLAGS	CSIMFLAGS
hls/dma_stream_snk/dma_stream_snk.cpp		
hls/dma_stream_snk/dma_stream_snk.h		

Top Function: dma_stream_snk_wrapper Browse

TEST BENCH FILES

FILE/FOLDER(S)	CFLAGS

Cancel Back Next

Create HLS Component - Empty HLS Component

Name and Location > Configuration File > Source Files > Part > Settings > Summary

Select Part

Specify a board, device, or platform you are compiling your component for

☐ Part ☒ Platform

NAME	BOARD	FLOW	VENDOR	PATH
base_platforms (7)				
xilinx_vck190_base_dfx_202310_1	xd	Embedded Accel	xilinx.com	..._0507_1903/installs/lin64/Vitis/2023.1/base_platforms
xilinx_zcu102_base_202310_1	xd	Embedded Accel	xilinx.com	...ase_dfx_202310_1/xilinx_vck190_base_dfx_202310_1.xpfm
xilinx_zcu102_base_dfx_202310_1	xd	Embedded Accel	xilinx.com	...zcu102_base_202310_1/xilinx_zcu102_base_202310_1.xpfm
xilinx_vmk180_base_202310_1	xd	Embedded Accel	xilinx.com	...ase_dfx_202310_1/xilinx_zcu102_base_dfx_202310_1.xpfm
xilinx_vck190_base_202310_1	xd	Embedded Accel	xilinx.com	
xilinx_zcu104_base_202310_1	xd	Embedded Accel	xilinx.com	
xilinx_vak280_est_base_202310_1	xd	Embedded Accel	xilinx.com	

Create HLS Component - Empty HLS Component

Name and Location > Configuration File > Source Files > Part > Settings > Summary

Edit Settings

Specify your flow target, clock frequency, and output file names

TARGET FLOW

☐ Vivado IP Flow

☒ Vitis Kernel Flow

☒ Run Additional Packing Step to create .xo output

CLOCK

Period: Period

Uncertainty: Uncertainty

Cancel Back Next

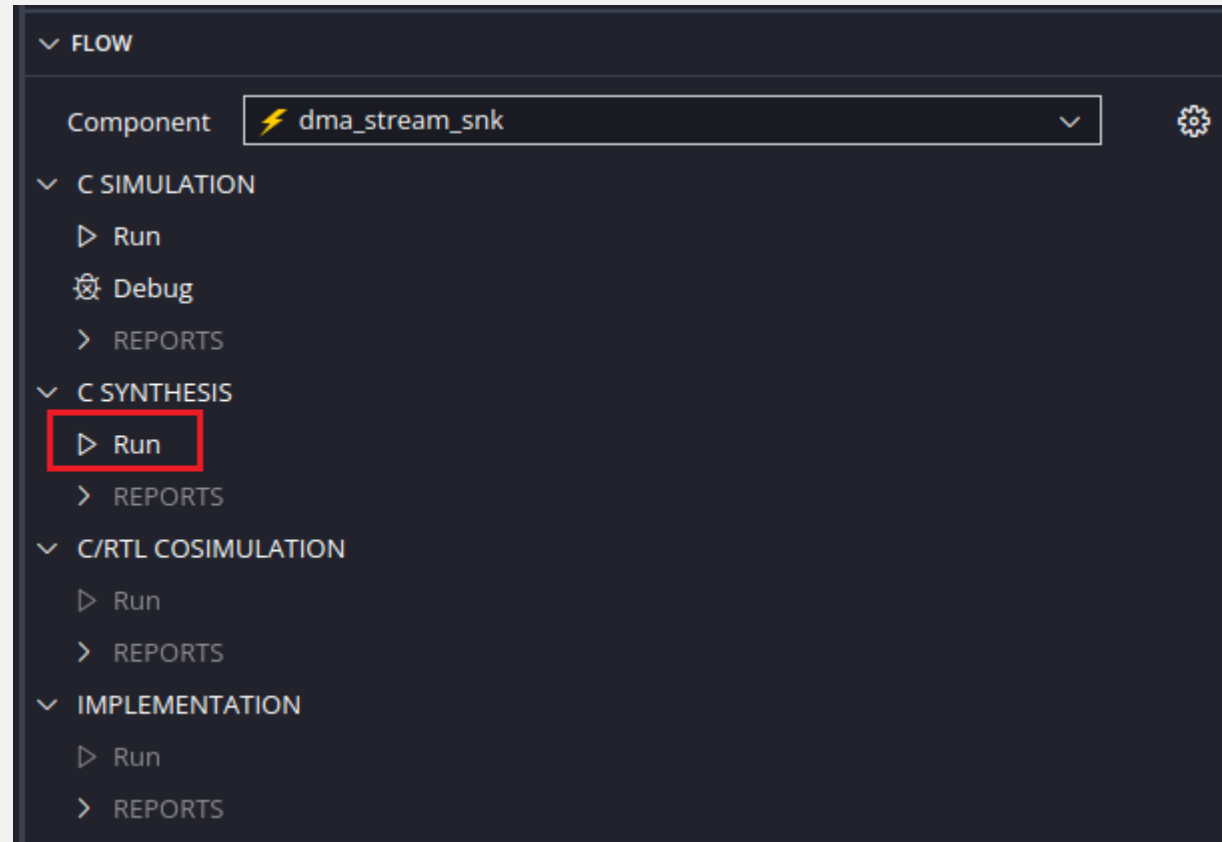
Vitis™ Unified IDE – HLS Development Flow

Step 1:
Create HLS Comp

Step 2:
Add Design Files

Step 3:
Build the Comp

- Build the HLS component



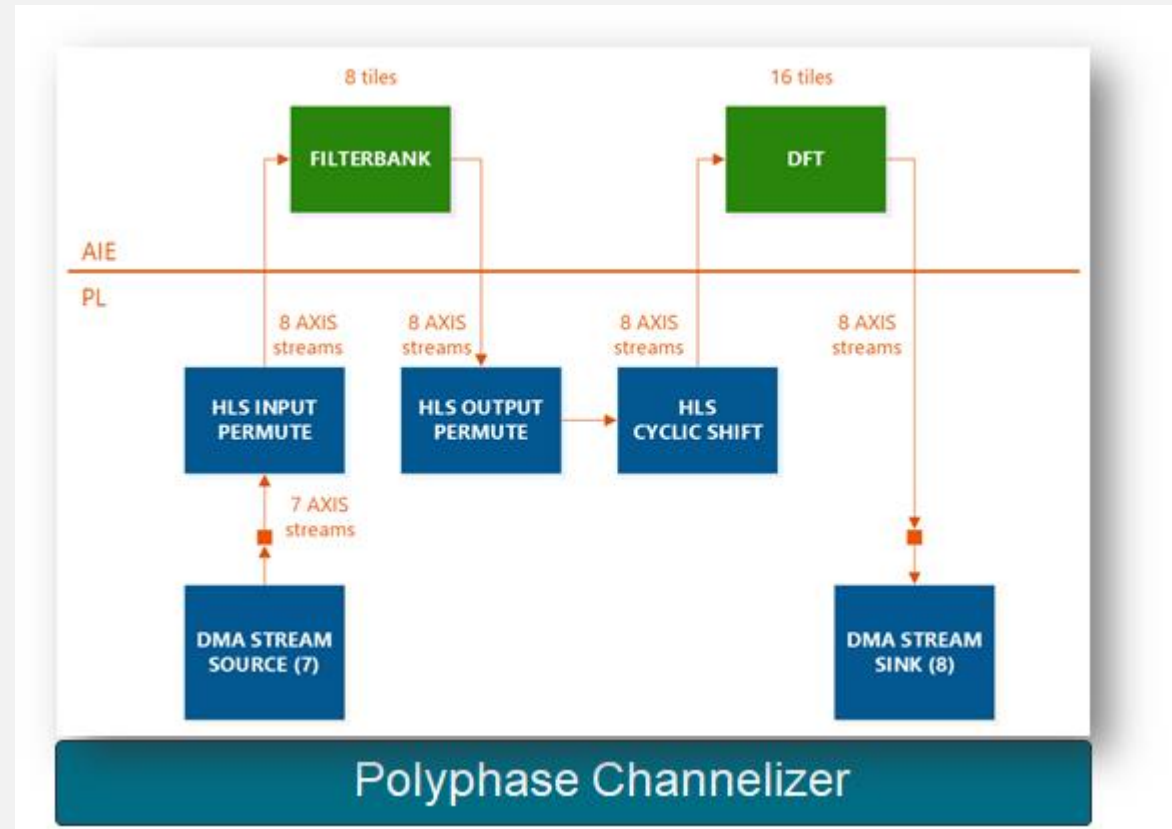
Versal™ System Integration – Vitis™ Tool Flow

AI Engine Design

PL Design

Application (PS – Host Code)

System Project



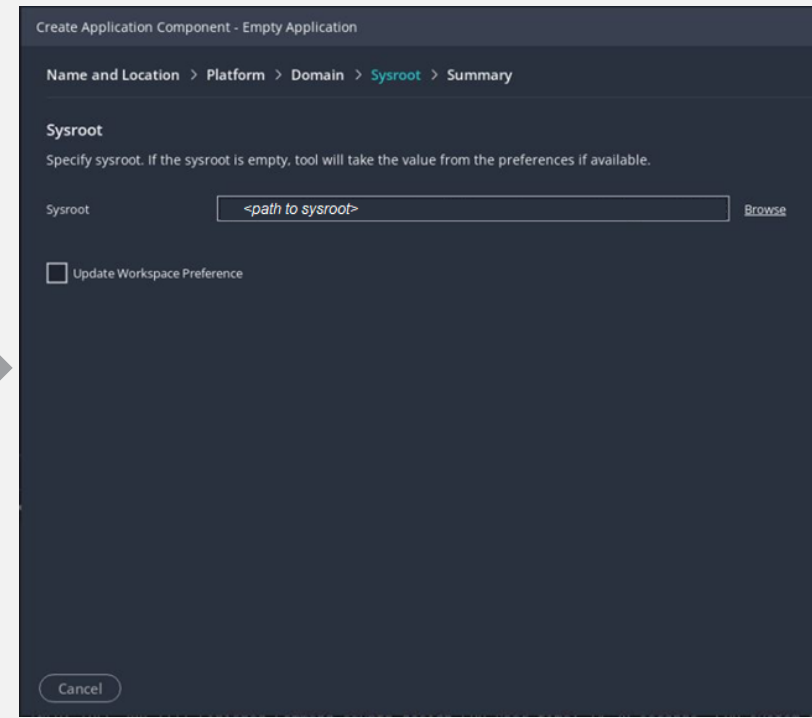
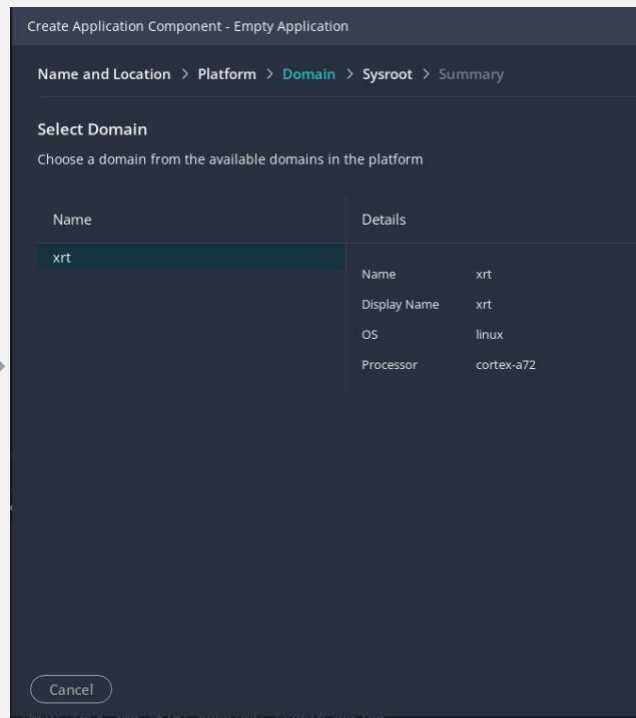
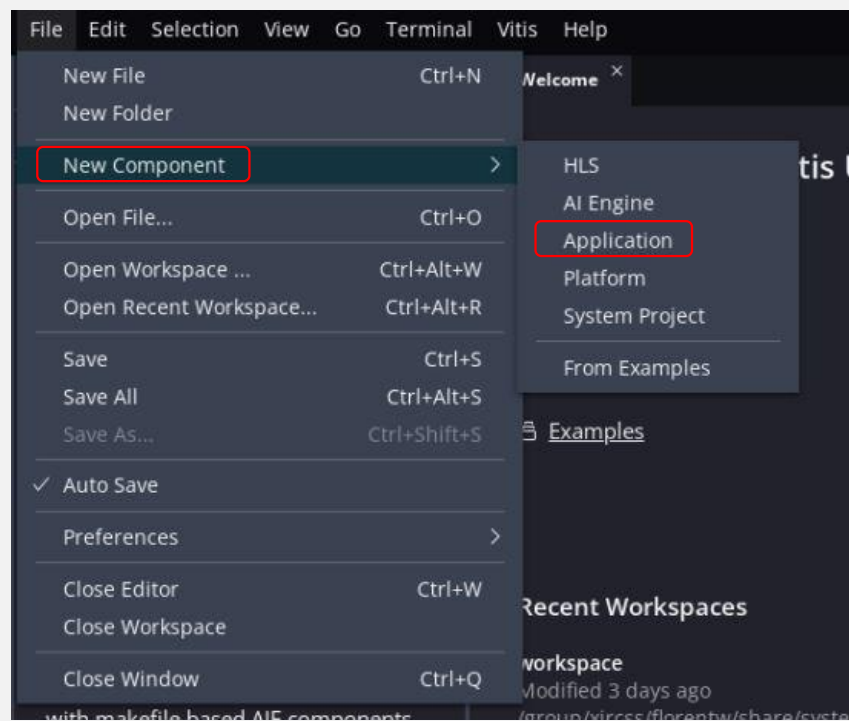
Vitis™ Unified IDE – Application Development Flow

Step 1:
Create PS Component

Step 2:
Add Design Files

Step 3:
Include AIE Kernel Path

Create PS comp | Select the platform | Select domain | Select sysroot



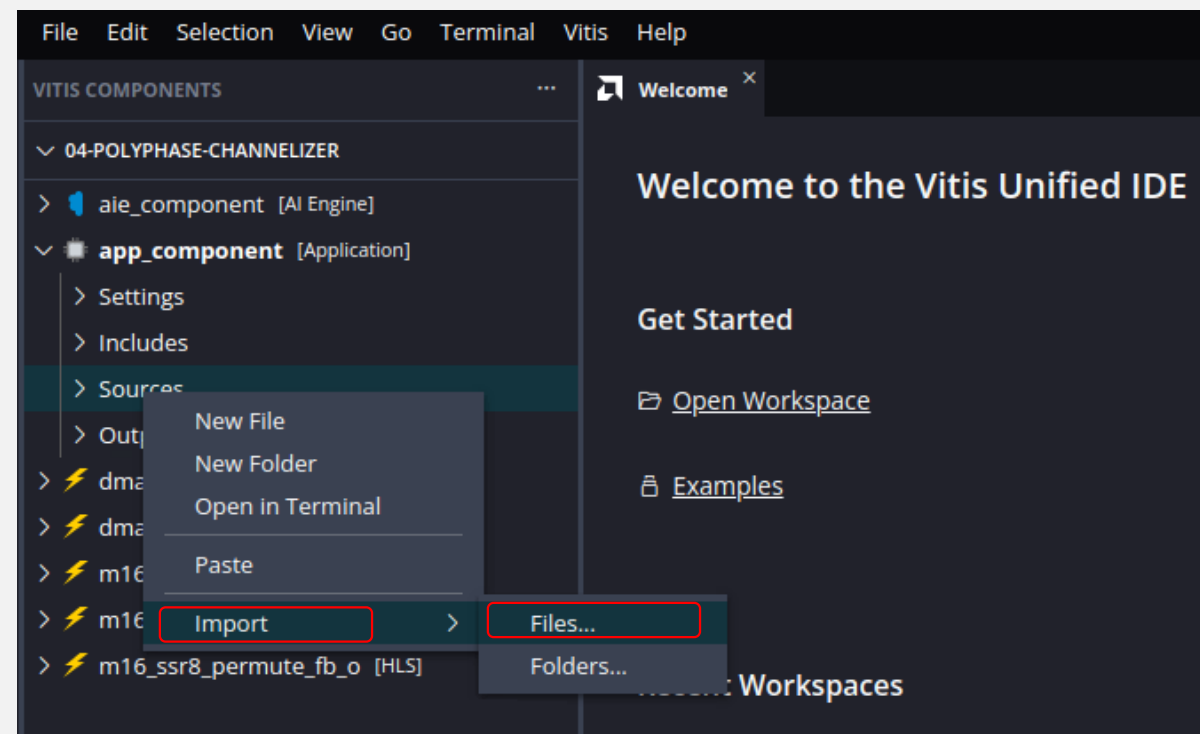
Vitis™ Unified IDE – Application Development Flow

Step 1:
Create PS Component

Step 2:
Add Design Files

Step 3:
Include AIE Kernel Path

- Import application file
 - host.cpp, found in ps_apps/hw_emu
- Import aie_control_xrt.cpp
 - Generated after compiling aie_component, found in aie_component/build/hw/Work/ps/c_rts



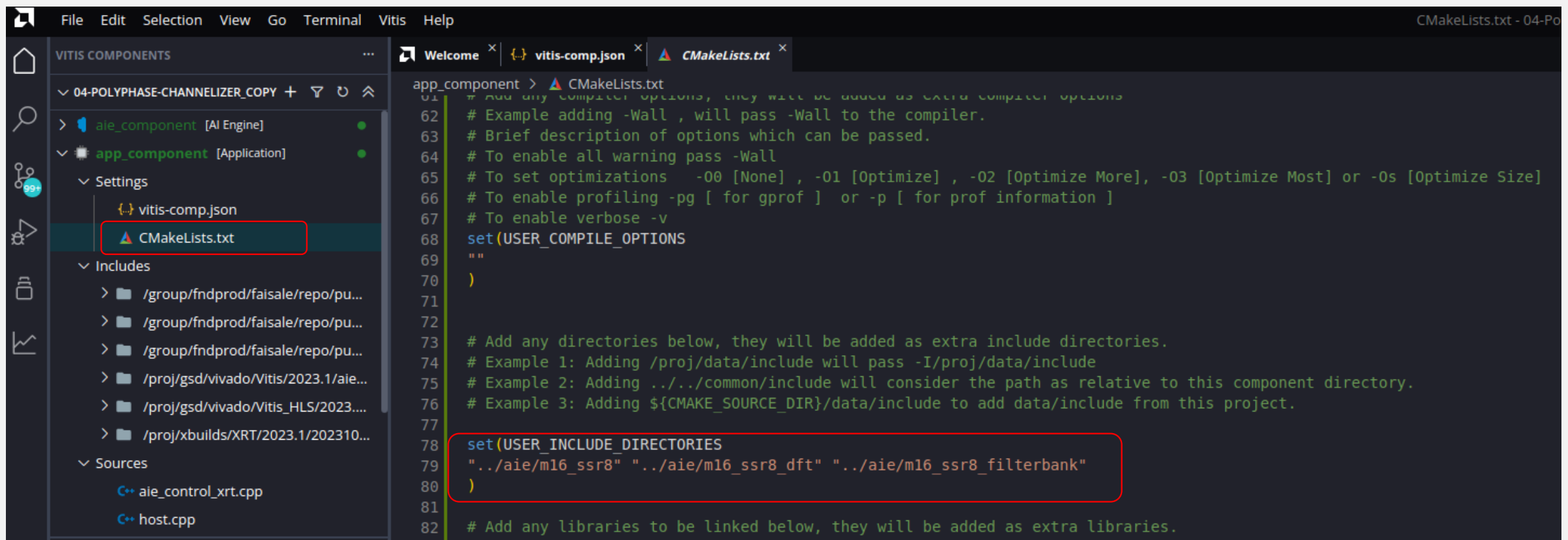
Vitis™ Unified IDE – Application Development Flow

Step 1:
Create PS Component

Step 2:
Add Design Files

Step 3:
Include AIE Kernel Path

- Include AIE kernel path



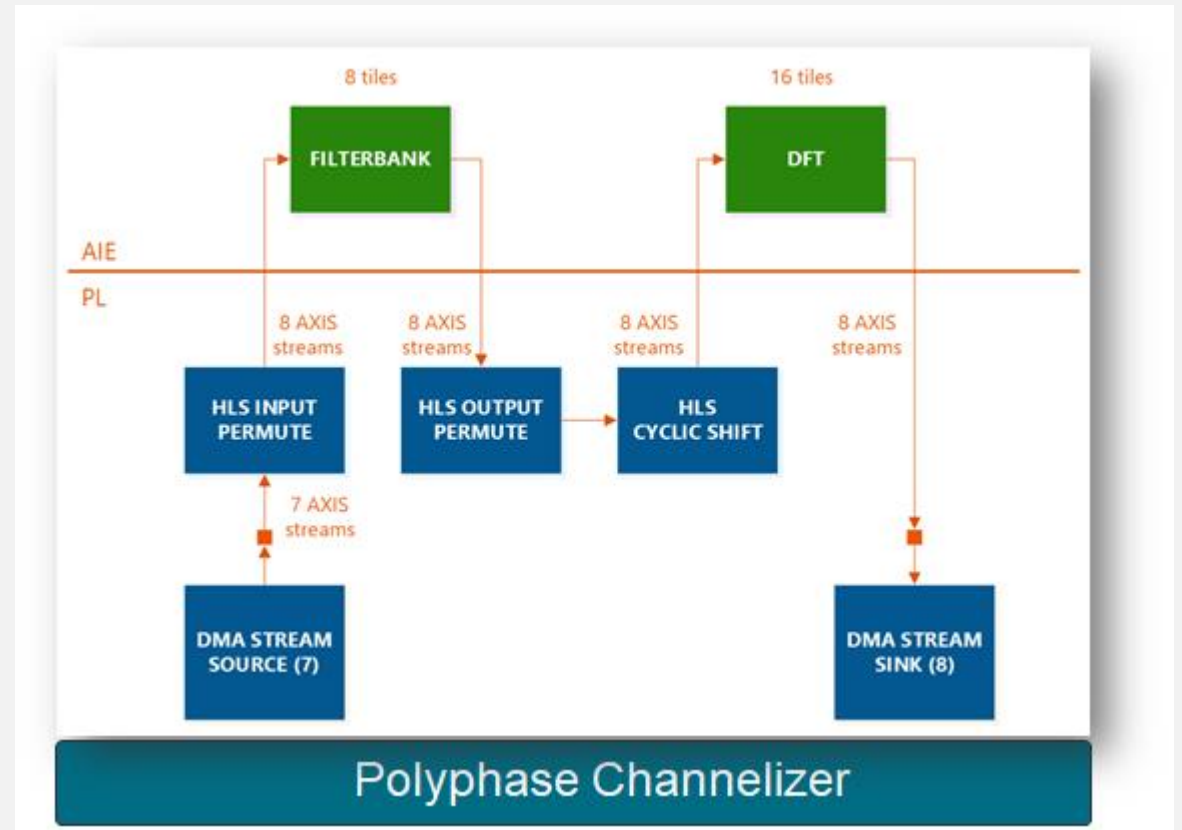
Versal™ System Integration – Vitis™ Tool Flow

AI Engine Design

PL Design

Application (PS – Host Code)

System Project



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

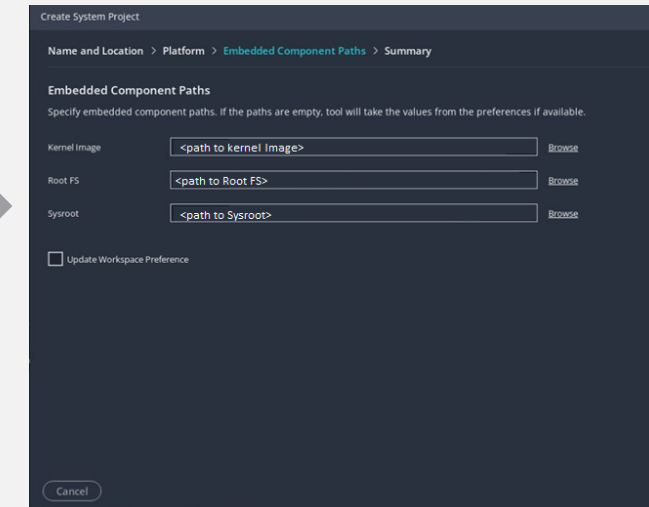
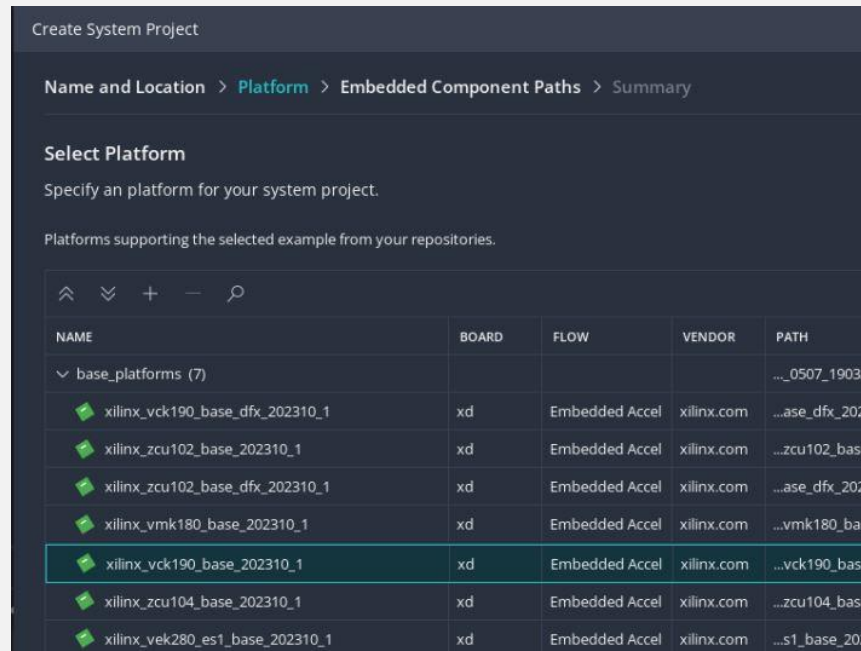
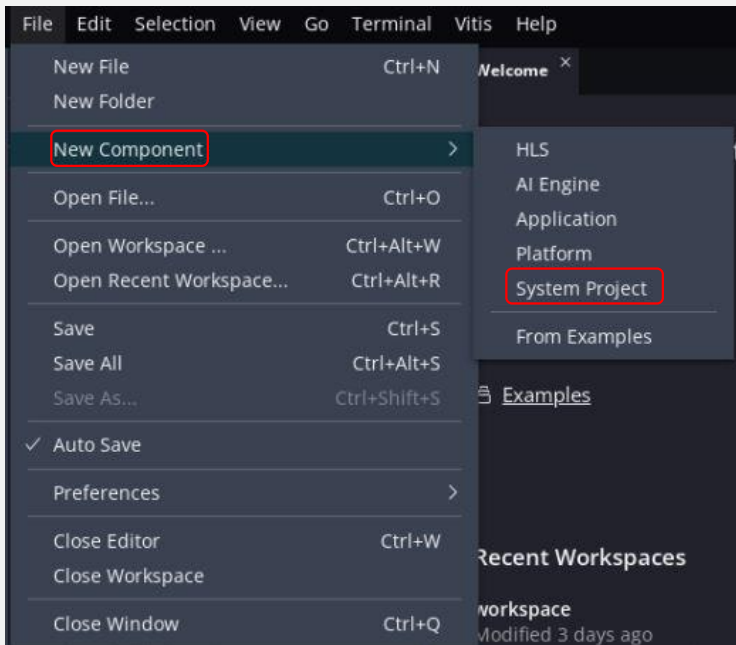
Create system project

Select the platform

Select sysroot

- Kernel image, rootfs, and sysroot can be generated from PetaLinux or another Linux build

- To download prebuilt images for VCK190 base platform, refer to <https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/embedded-platforms.html>



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

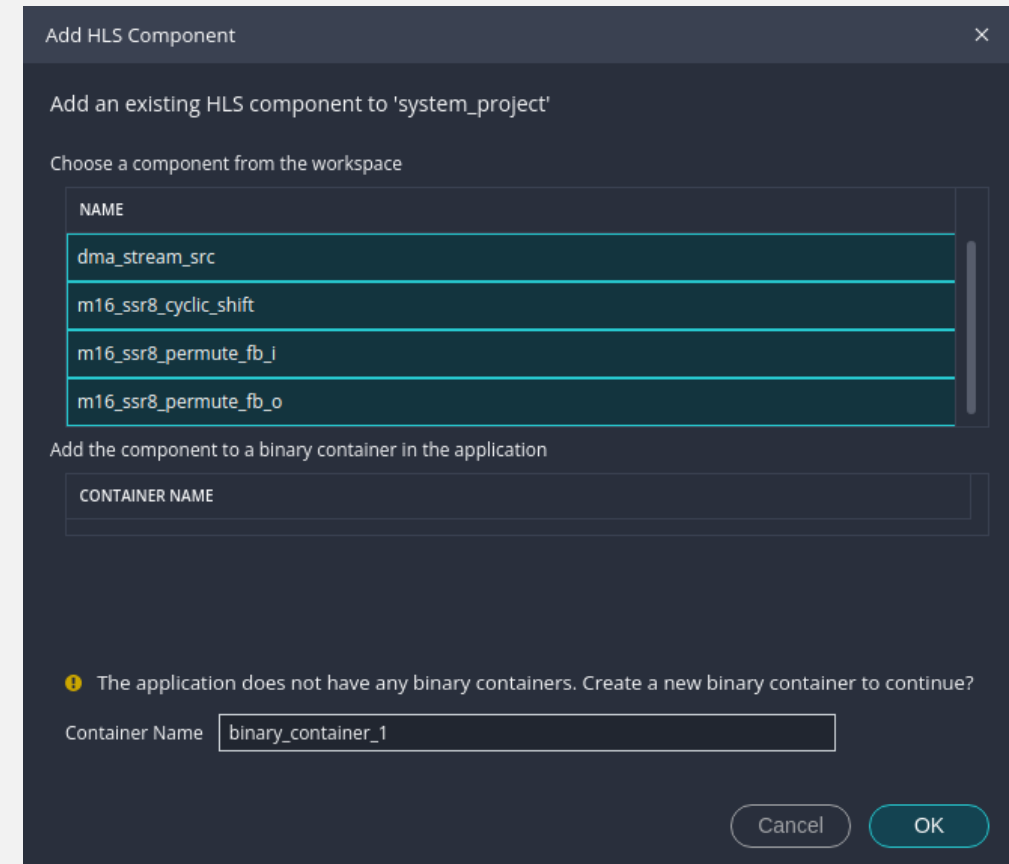
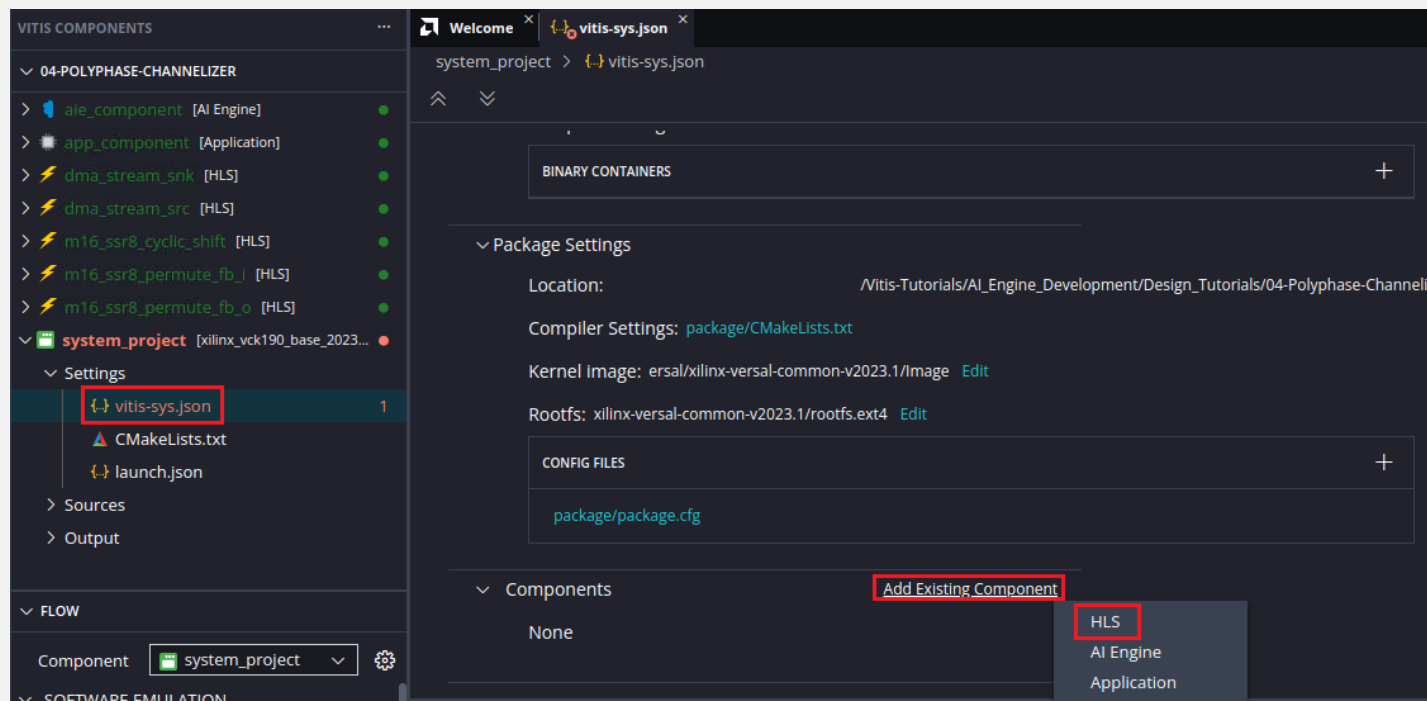
Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

HLS component



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

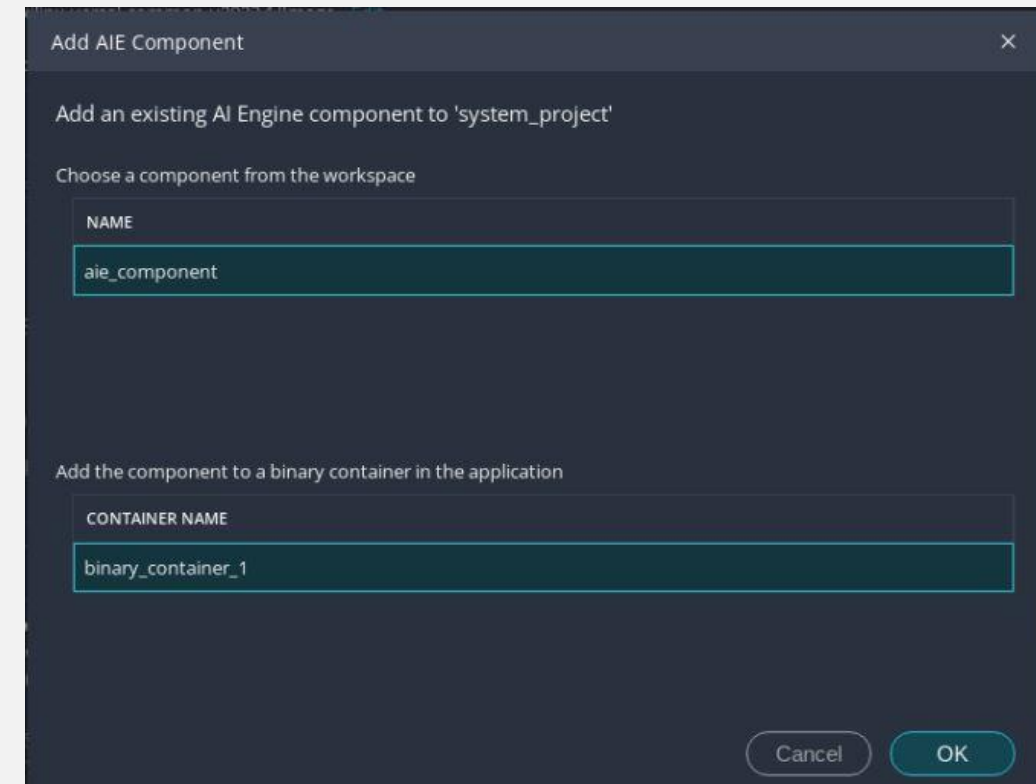
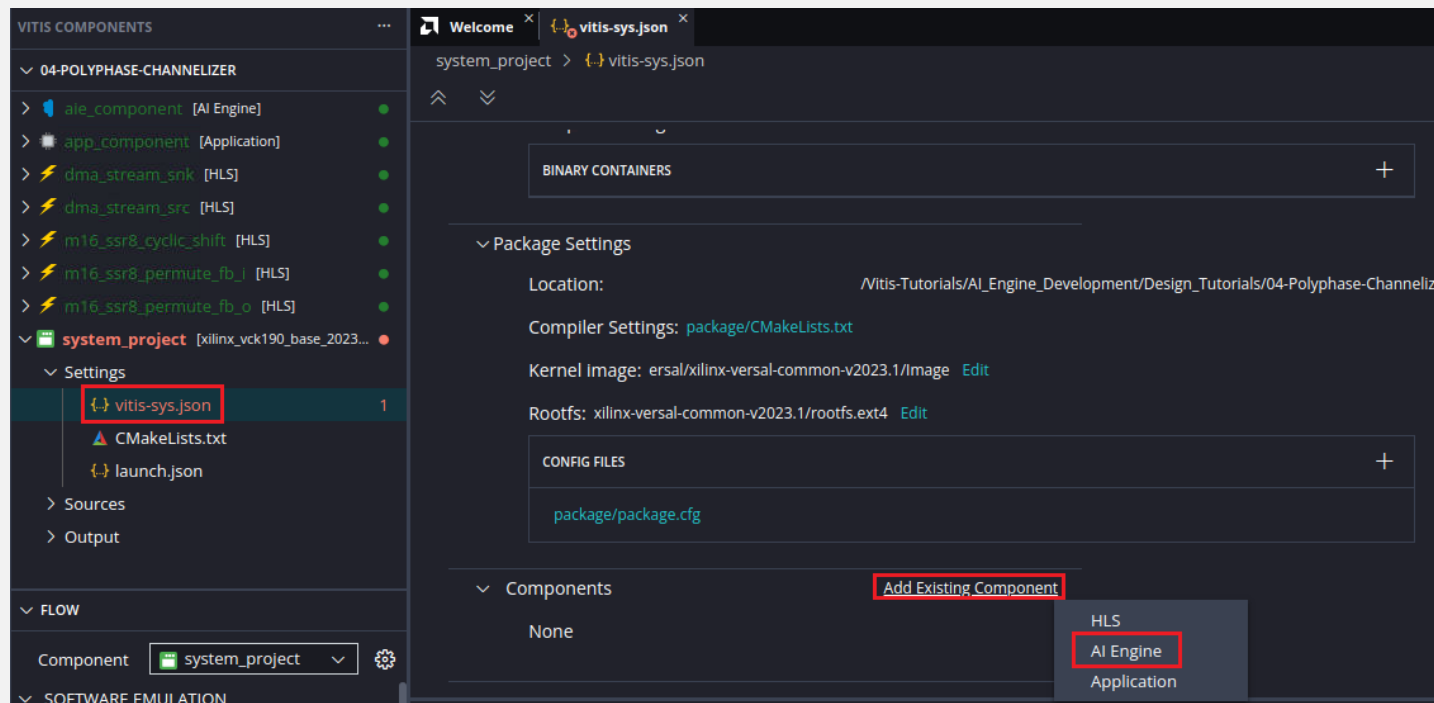
Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

HLS component

AIE component



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

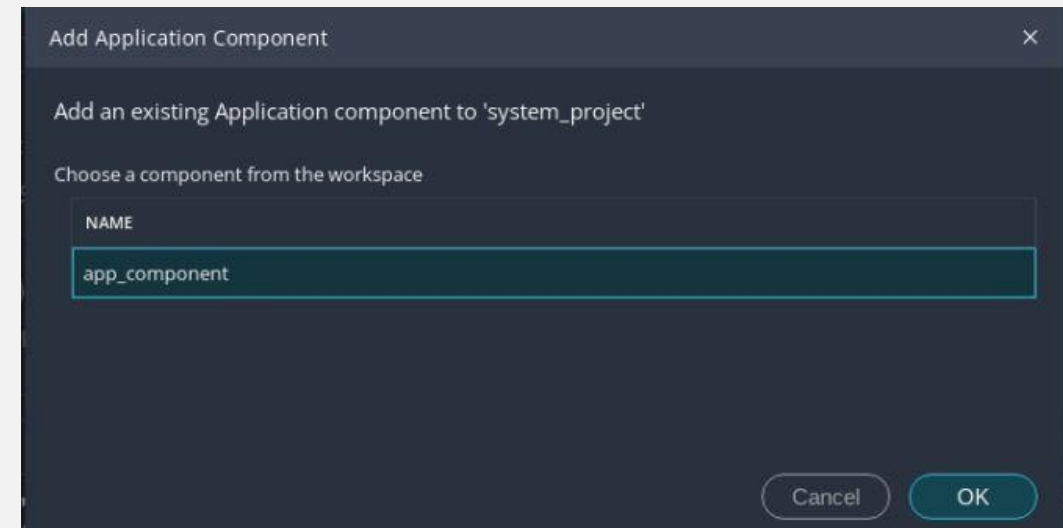
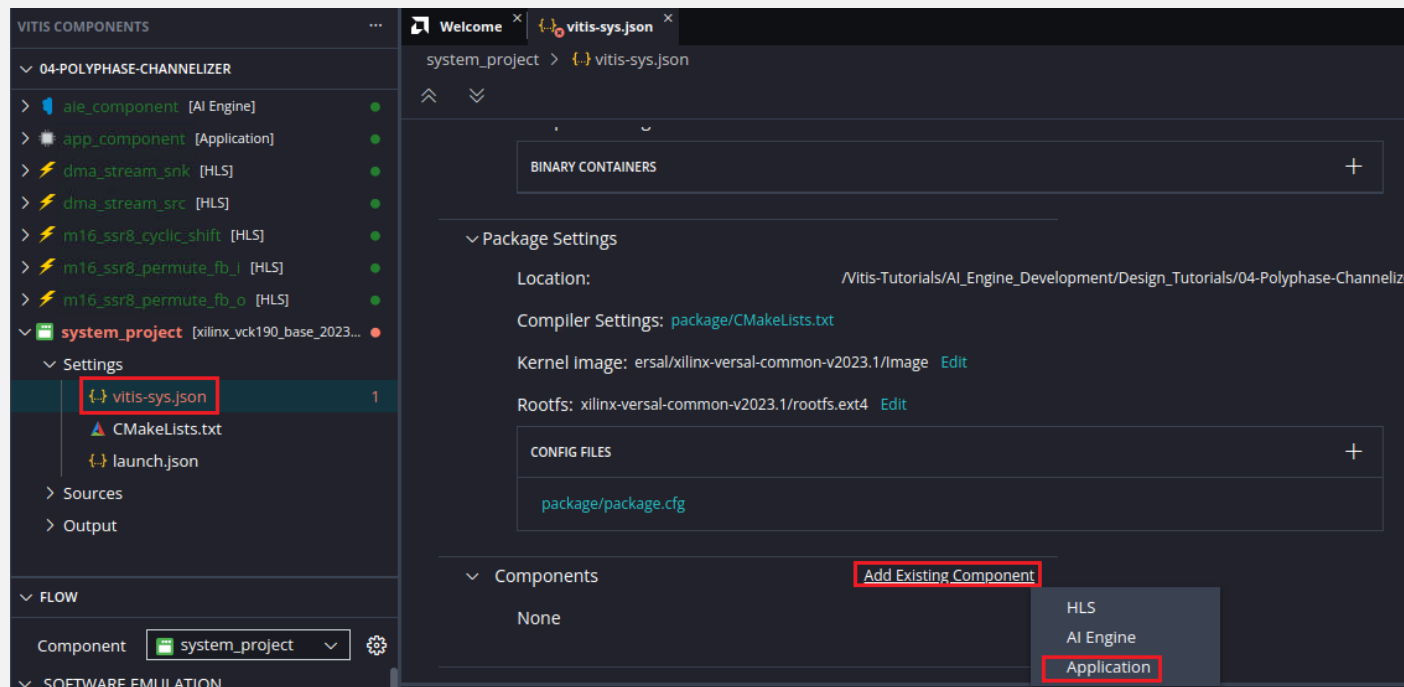
Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

HLS component

AIE component

PS component



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

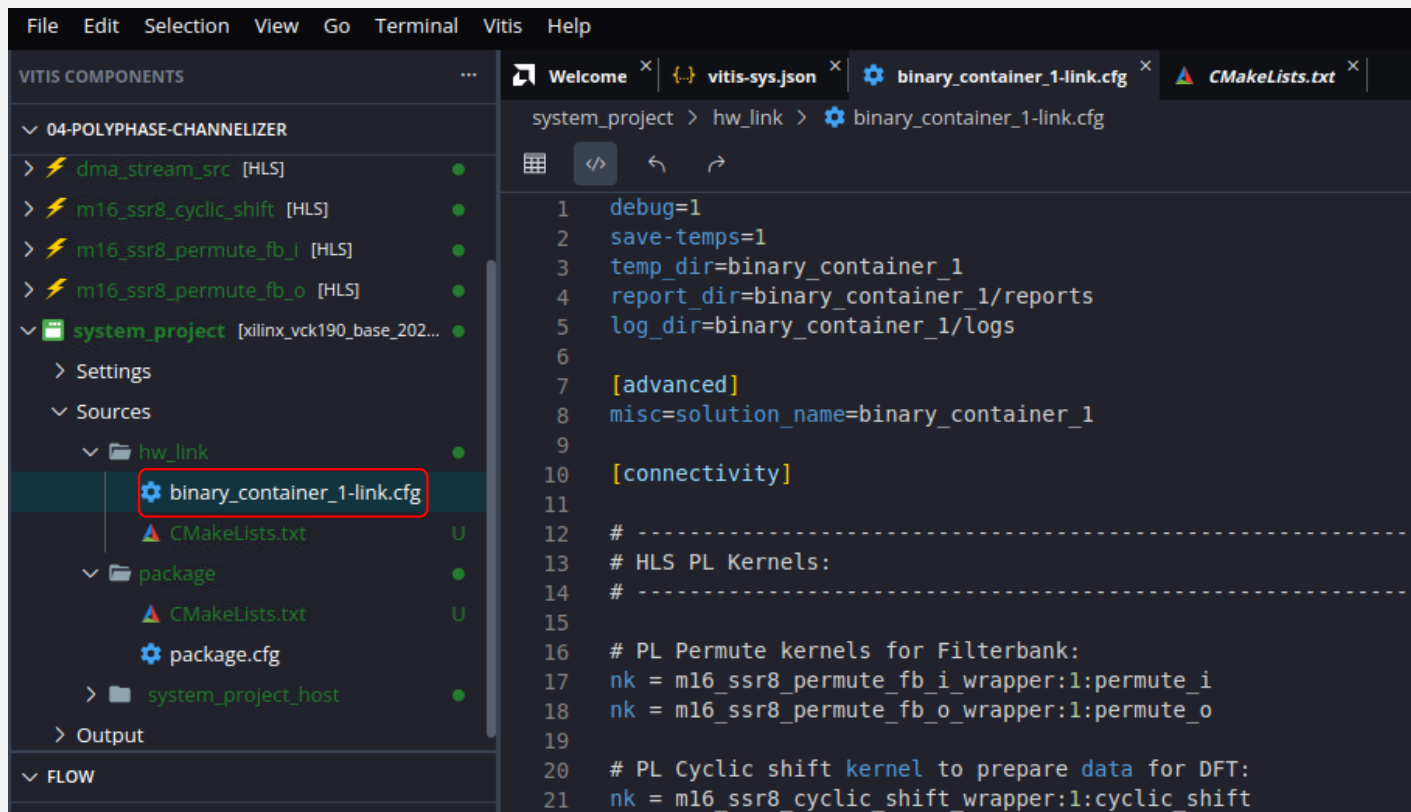
Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System



- Edit hw_link binary_container_1-link.cfg
- Connect PL kernels to PL kernels
- Connect PL kernels to AIE kernels
- Use system.cfg in vitis/final as reference
- For more information, refer to <https://docs.xilinx.com/r/en-US/ug1393-vitis-application-acceleration/connectivity-Options>

Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

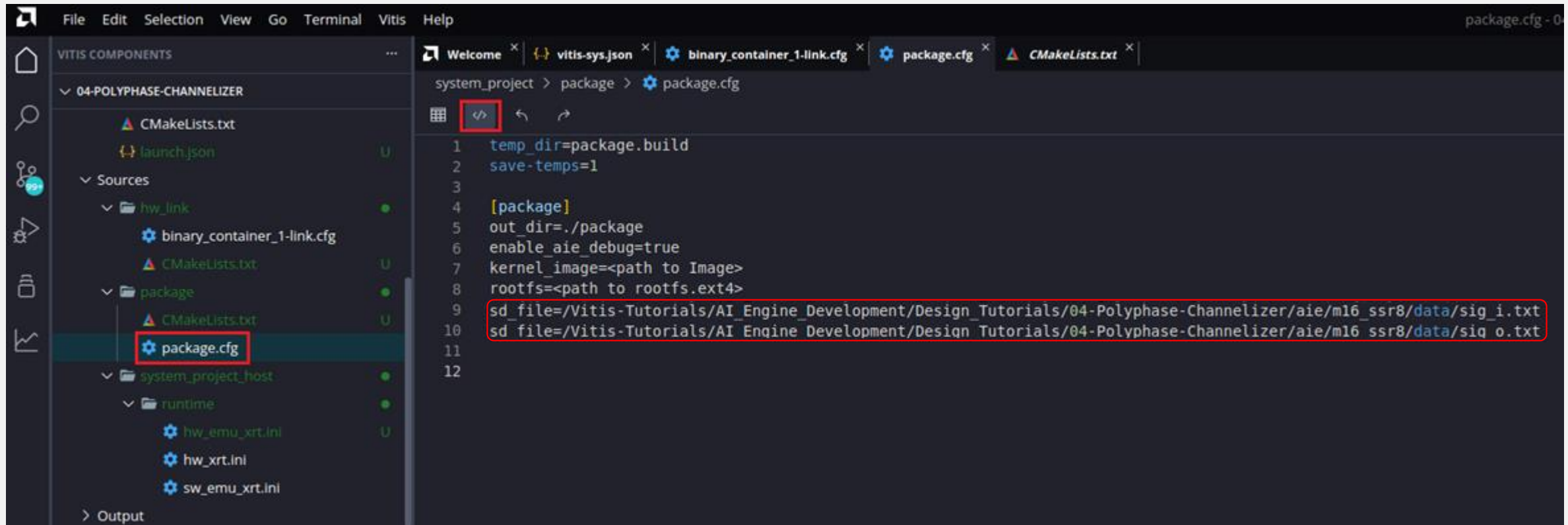
Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

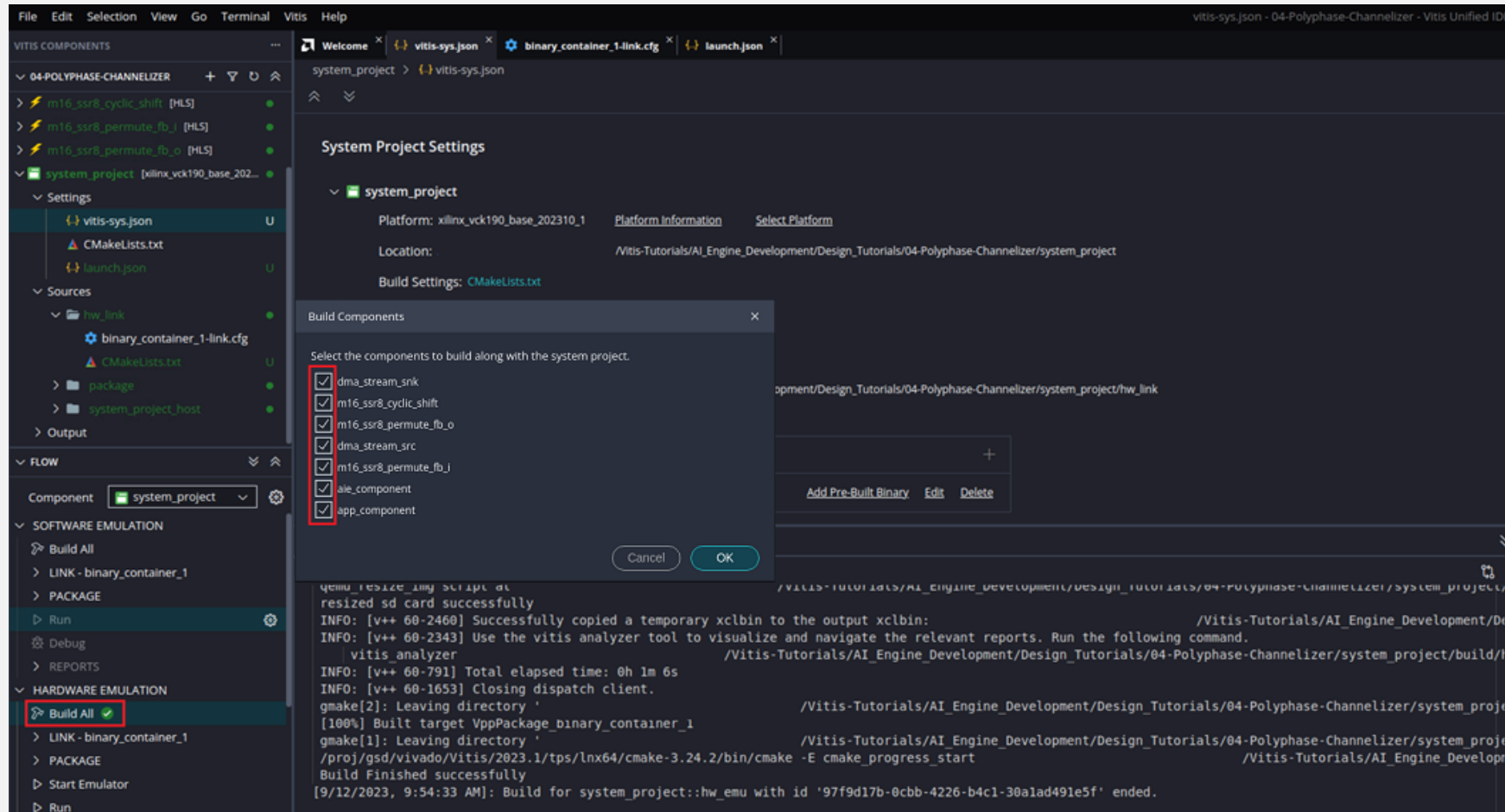
Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

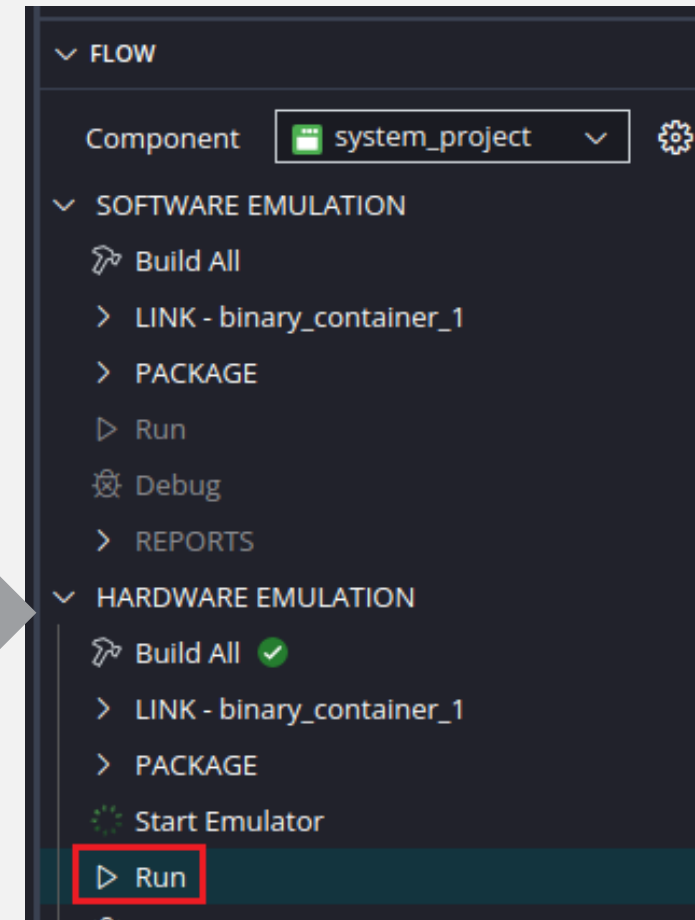
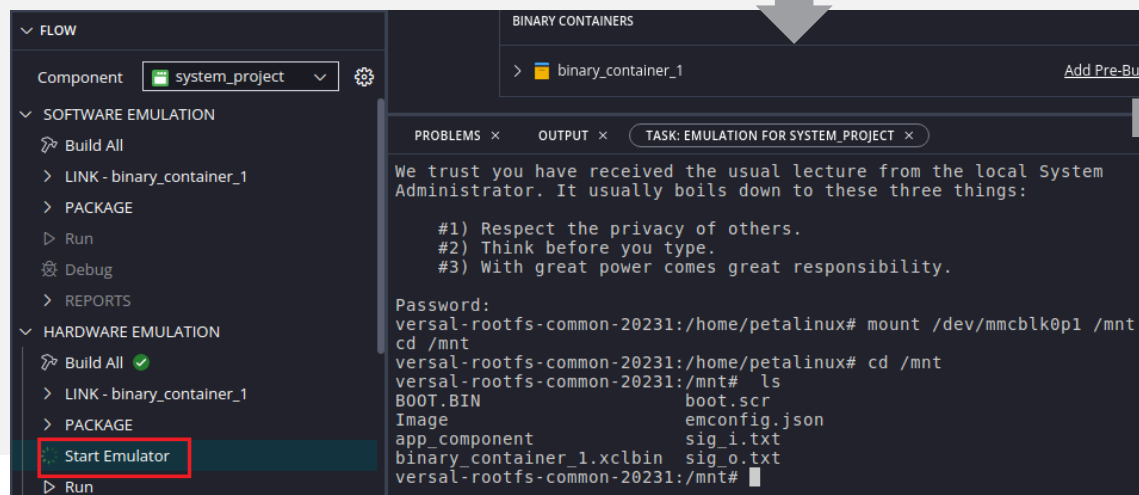
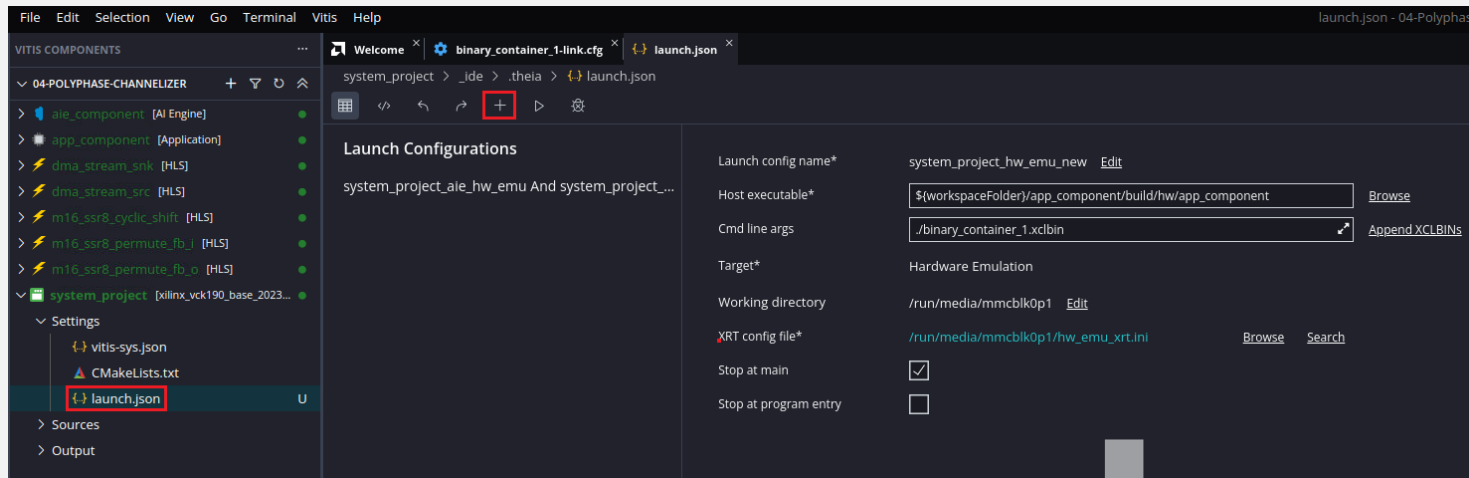
Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

FLOW

Component

system_project

SOFTWARE EMULATION

Build All

LINK - binary_container_1

PACKAGE

Run

Debug

REPORTS

HARDWARE EMULATION

Build All

LINK - binary_container_1

PACKAGE

Start Emulator

Run

PROBLEMS

TASK: EMULATION FOR SYSTEM_PROJECT

DEBUG CONSOLE

16375	:Gold	2	-1	:Actual	2	-1
16376	:Gold	2	-1	:Actual	2	-1
16377	:Gold	-3	2	:Actual	-3	2
16378	:Gold	2	1	:Actual	2	1
16379	:Gold	-7475	-1841	:Actual	-7475	-1841
16380	:Gold	-1	-8	:Actual	-1	-8
16381	:Gold	-4	-4	:Actual	-4	-4
16382	:Gold	-7	5	:Actual	-7	5
16383	:Gold	5	7	:Actual	5	7

--- PASSED ---

Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

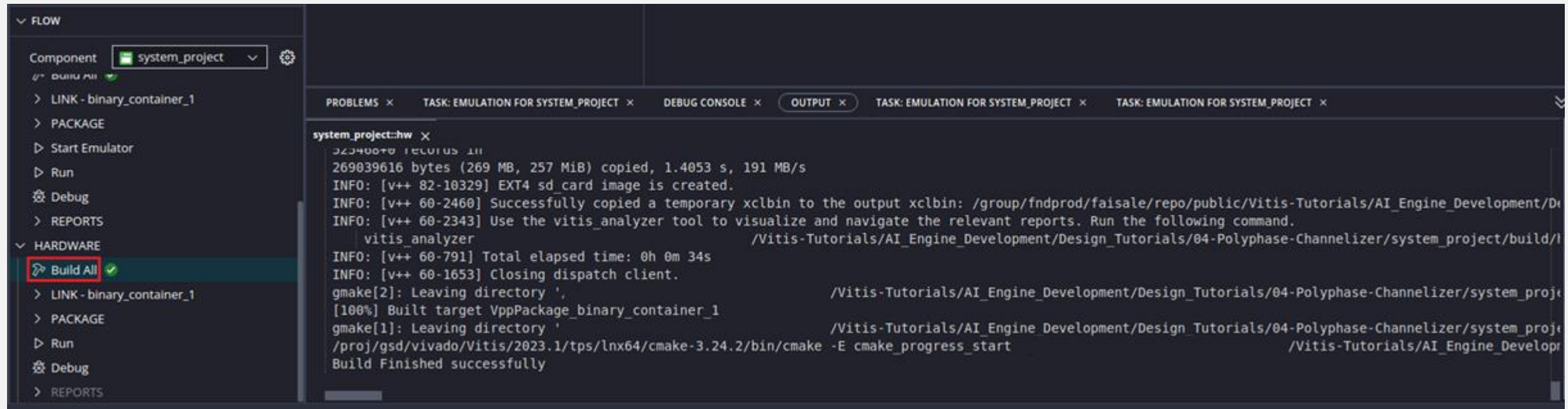
Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System



Vitis™ Unified IDE – System Project

Step 1:
Create System
Project

Step 2:
Link System Comp

Step 3:
AIE & PL Conn.

Step 4:
Package Config

Step 5: HW EMU
Build & Run the
System

Step 6: HW
Build & Run the
System

- Flash sd_card.img into compatible SD card using [Balena Etcher](#) tool
- Insert SD card into board and power up. After boot up sequence is completed, log in using username petalinux and password petalinux
- sudo su - root and password petalinux
- cd /run/media/mmcblk0p1/
- ./app_component
binary_container_1.xclbin

```

Initializing ADF API...
PASSED: auto my_device = xrt::device(0)
XAIEFAL: INFO: Resource group Avail is created.
XAIEFAL: INFO: Resource group Static is created.
XAIEFAL: INFO: Resource group Generic is created.
PASSED: auto xclbin_uuid = my_device.load_xclbin(a.xclbin)
PASSED: auto my_graph = xrt::graph(my_device, xclbin_uuid, "aie_dut")
PASSED: my_graph.reset()
PASSED: my_graph.run( NUM_ITER=2 )
PASSED: auto dma_src = xrt::kernel(my_device, xclbin_uuid, "dma_stream_src_wrapper:{dma_src}")
PASSED: auto dma_snk = xrt::kernel(my_device, xclbin_uuid, "dma_stream_snk_wrapper:{dma_snk}")
PASSED: xrt::run dma_src_run = xrt::run(dma_src)
PASSED: xrt::run dma_snk_run = xrt::run(dma_snk)
PASSED: dma_src_bo = xrt::bo(my_device, DDR_BUFSIZE_I_BYTES, dma_src.group_id(0) (=2))
PASSED: auto dma_src_bo_mapped = dma_src_bo.map<TT_DATA*>()
PASSED: dma_snk_bo = xrt::bo(my_device, DDR_BUFSIZE_O_BYTES, dma_snk.group_id(0) (=2))
PASSED: auto dma_snk_bo_mapped = dma_snk_bo.map<TT_DATA*>()
PASSED: Successfully read input file sig_i.txt
PASSED: dma_src_run.set_arg( 0, dma_src_bo )
PASSED: dma_src_run.set_arg( 1, LOOP_CNT_I=8 )
PASSED: dma_snk_run.set_arg( 0, dma_snk_bo )
PASSED: dma_snk_run.set_arg( 1, LOOP_SEL=0 )
PASSED: dma_snk_run.set_arg( 2, LOOP_CNT_O=4 )
PASSED: dma_snk_run.set_arg( 3, DFT_PERM=1 )
PASSED: dma_src_run.start()
PASSED: dma_snk_run.start()
INFO: Waiting for kernels to end...
PASSED: dma_snk_run.wait()
PASSED: dma_snk_bo.sync(XCL_BO_SYNC_BO_FROM_DEVICE)
0 :Gold 0 0 :Actual 0 0
1 :Gold 0 0 :Actual 0 0
2 :Gold 0 0 :Actual 0 0
3 :Gold 0 0 :Actual 0 0
4 :Gold 0 0 :Actual 0 0
5 :Gold 0 0 :Actual 0 0
.
.
.
.
.
16380 :Gold -1 -8 :Actual -1 -8
16381 :Gold -4 -4 :Actual -4 -4
16382 :Gold -7 5 :Actual -7 5
16383 :Gold 5 7 :Actual 5 7
--- PASSED ---

```

Summary

-
1. Discussed AI Engine development flow
 2. Introduced AI Engine Test Harness to rapidly take AI Engine graph from simulation to hardware
 3. Discussed Versal™ system integration using Vitis™ tool flow
 4. Demonstrate various flows using design example – Polyphase Channelizer

Additional Resources to Get Started

- AI Engine Test Harness
 - GitHub: <https://github.com/Xilinx/AI-Engine-Test-Harness>
 - Documentation: <https://xilinx.github.io/AI-Engine-Test-Harness>
- Vitis™ Tutorials GitHub:
 - <https://github.com/Xilinx/Vitis-Tutorials>

DISCLAIMER AND ATTRIBUTIONS

DISCLAIMER

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's products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale. GD-18

©2023 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Versal, Vitis, Vivado, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

