



Accelerating Signal Processing on AI Engines

***Using AMD Vitis™ DSP Libraries and
Vitis Model Composer***

**Chandra Sekar Balakrishnan
Mark Rollins**

Objectives

After completing this module, you will be able to:

01

Describe the Versal™ AI Engine DSP library

02

Identify various components in the AI Engine DSP library

03

Create your DSP application using the AI Engine DSP library

04

Develop designs using the following tool flows:

- Vitis™ AIE development
- Vitis Model Composer

Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - Design 1: FIR/FFT Design for $SSR < 1$
 - Design 2: FIR/FFT Design for $SSR > 1$
 - AI Engine Design Using the Vitis™ IDE
 - AI Engine Design Using Vitis Model Composer
 3. Summary

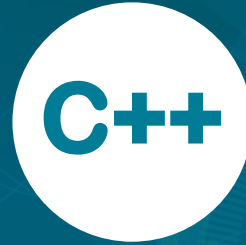
What Is the AI Engine DSP Library?



Configurable library of kernels that can be used to develop applications on Versal™ AI Engines



Open-source library for DSP applications



Kernels are coded using AI Engine APIs in C++ and give access to AI Engine vector processing capabilities



Kernels can be combined to construct graphs for developing complex designs



Example design is also provided with this library



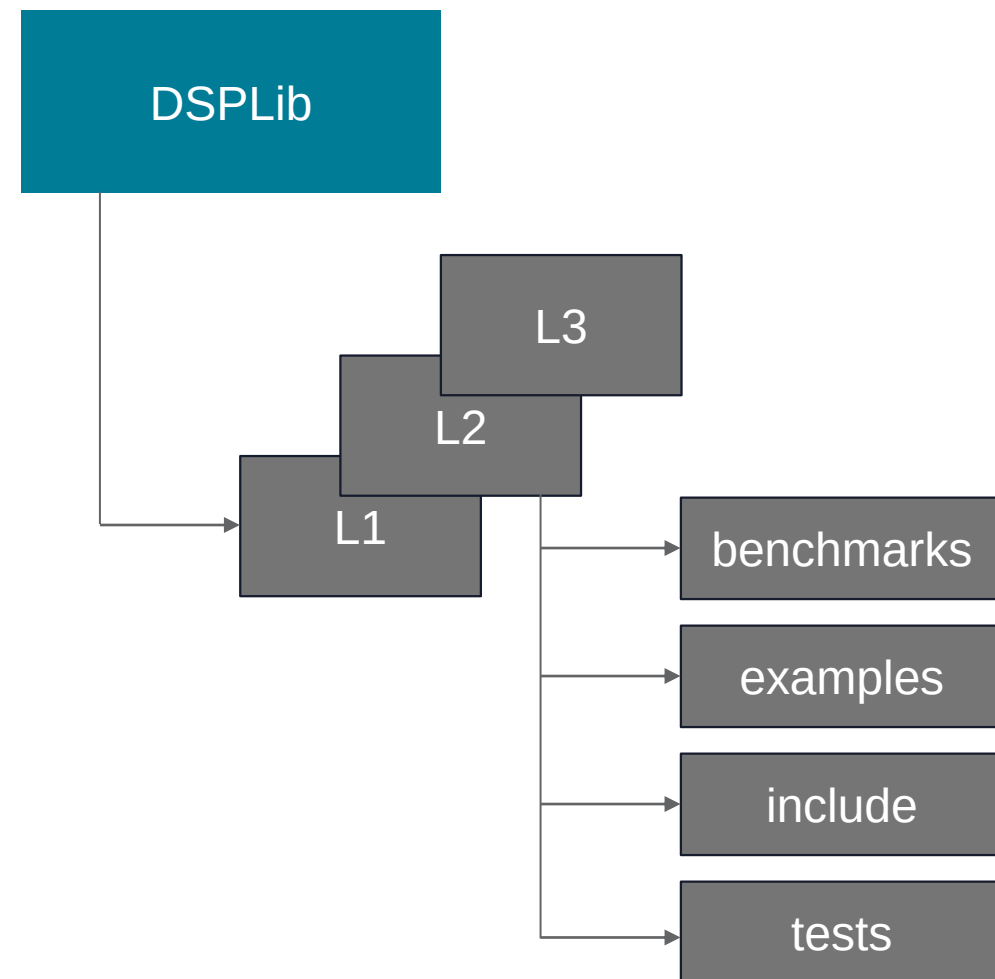
Use the library element's graph as the entry point

Parameterized Solution Functions

Directories L1, L2, and L3 correspond to AI Engine kernels, AI Engine graphs, and software APIs for each function

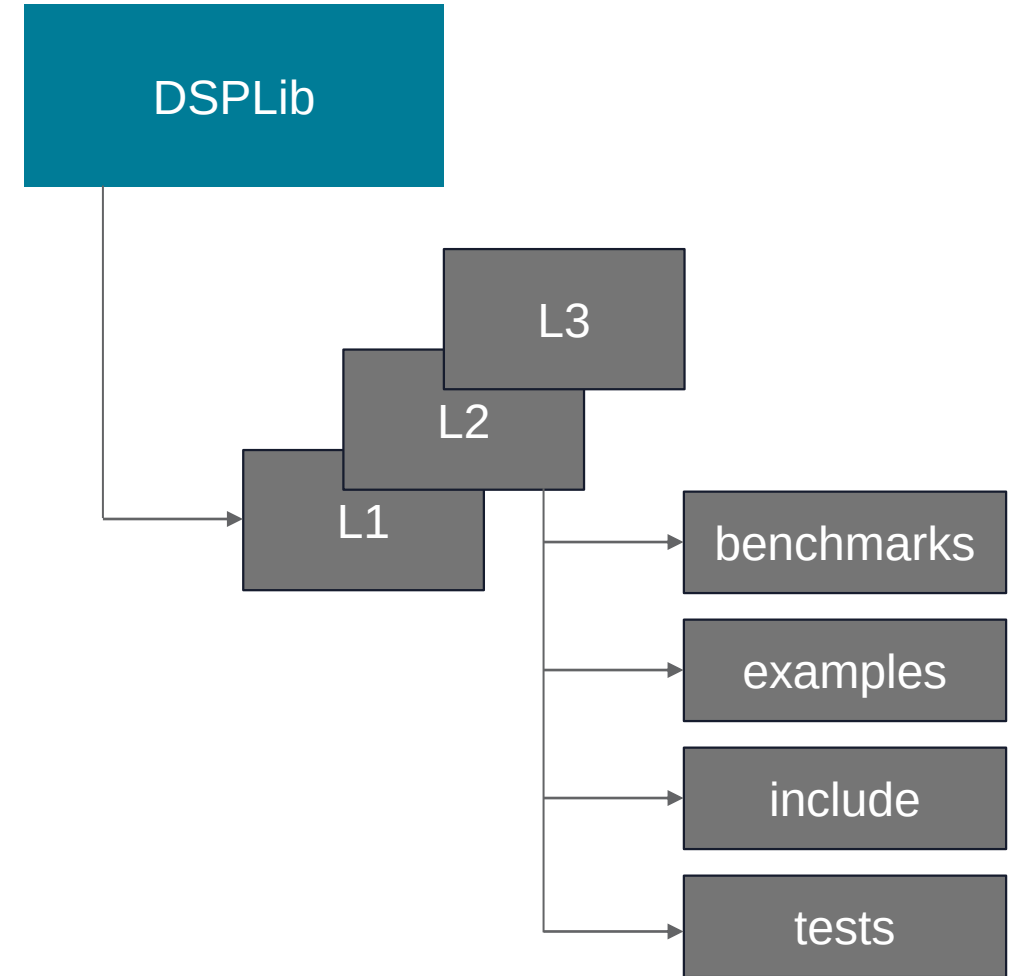
Inclusion of an L2 element rather than an L1 element is recommended

L3 directory not yet available



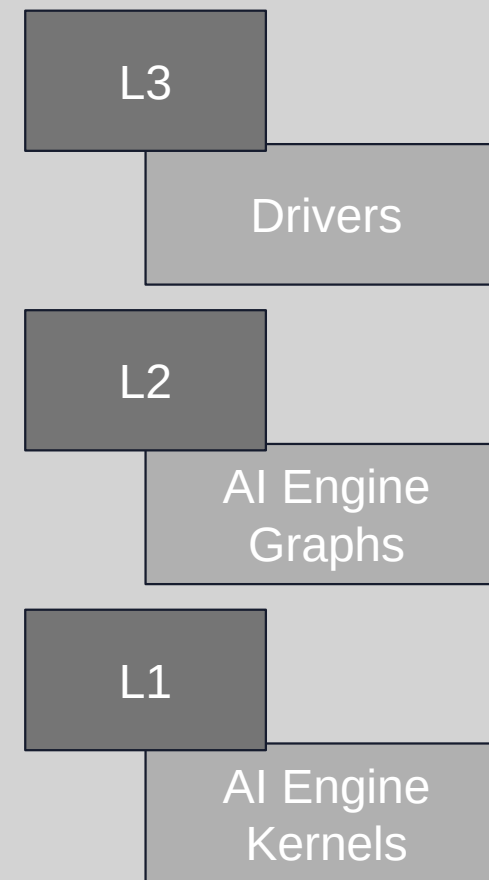
Parameterized Solution Functions

Description	Directory/Subdirectory
Graph class declarations and constants	L2/include/aie
Kernel class definitions, .cpp files, and corresponding .hpp files	L1/src/aie and L1/include/aie
Test bench for the library element	L2/tests/aie/<library_element>
Reference model graph classes	L2/tests/aie/common/inc
Reference model kernel classes	L1/tests/aie/inc/ and L1/tests/aie/src
Example wrapper designs	L2/examples



AI Engine DSP Library Installation

- Download library from GitHub
 - https://github.com/Xilinx/Vitis_Libraries/tree/master/dsp
- Set the environment variable
 - `export DSPLIB_ROOT=<vitis-libraries-install/dsp>`
- Use the following options in the aiecompiler command to provide the path
 - `-include=$DSPLIB_ROOT/L2/include/aie`
 - `-include=$DSPLIB_ROOT/L1/include/aie`
 - `-include=$DSPLIB_ROOT/L1/src/aie`



DSPLib Functions

DDS / Mixer

`xf::dsp::aie::mixer::dds_mixer::dds_mixer_graph`

FFT / iFFT

`xf::dsp::aie::fft::fft_ifft_dit_1ch_graph`

FFT Window

`xf::dsp::aie::fft::fft_window_graph`

Matrix Multiply

`xf::dsp::aie::blas::matrix_mult::matrix_mult_graph`

Widget API Cast

`xf::dsp::aie::widget::api_cast::widget_api_cast_graph`

Widget Real to Complex

`xf::dsp::aie::widget::real2complex::widget_real2complex_graph`

Filters

Filters in the Vitis™ DSP Library

FIR Filters

Single rate, asymmetrical

`dsplib::fir::sr_asym::fir_sr_asym_graph`

Single rate, symmetrical

`dsplib::fir::sr_sym::fir_sr_sym_graph`

Interpolation asymmetrical

`dsplib::fir::interpolate_asym::fir_interpolate_asym_graph`

Decimation, half band

`dsplib::fir::decimate_hb::fir_decimate_hb_graph`

Interpolation, half band

`dsplib::fir::interpolate_hb::fir_interpolate_hb_graph`

Decimation, asymmetric

`dsplib::fir::decimate_asym::fir_decimate_asym_graph`

Interpolation, fractional,
asymmetric

`dsplib::fir::interpolate_fract_asym::fir_interpolate_fract_asym_graph`

Decimation, symmetric

`dsplib::fir::decimate_sym::fir_decimate_sym_graph`

Supported Combinations of Data Types and Coefficient Types

Data Type		Int16	Cint16	Int32	Cint32	Float	Cfloat
Coefficient Type	Int16	✓ ✕	✓	✓	✓	✕	✕
	Cint16	✕	✓	✕	✓	✕	✕
	Int32	✕	✕	✓	✓	✕	✕
	Cint32	✕ ✕	✕	✕	✓	✕	✕
	Float	✕	✕	✕	✕	✓	✓
	Cfloat	✕	✕	✕	✕	✕	✓

✓ *Supported*

✕ *Complex coefficients are not supported for real-only data types*

✕ *Coefficient type of higher precision than data type will be supported in 2023.1_web*

✕ *Mix of float and integer types is not supported*

✕ *fir_interpolate_asym and fir_decimate_asym variants do not support int16/int16*

Parameters Supported by FIR Filters (1)

Parameter Name	Type	Description	Range
TP_FIR_LEN	Unsigned int	Number of taps	4 to 240
TP_RND	Unsigned int	Round mode	0 = truncate or floor 1 = ceiling (round up) 2 = positive infinity 3 = negative infinity 4 = symmetrical to infinity 5 = symmetrical to zero 6 = convergent to even 7 = convergent to odd
TP_SHIFT	Unsigned int	Number of bits to shift accumulation down by before output	0 to 61
TT_DATA	Typename	Data type	int16, cint16, int32, cint32, float, cfloat
TT_COEFF	Typename	Coefficient type	int16, cint16, int32, cint32, float, cfloat

Parameters Supported by FIR Filters (2)

Parameter Name	Type	Description	Range
TP_INPUT_WINDOW_VSIZE	Unsigned int	Number of samples in the input window	Must be a multiple of the number of lanes used (typically 4 or 8); no enforced range, but large windows will result in mapper errors due to excessive RAM use
TP_CASC_LEN	Unsigned int	Number of cascaded kernels to use for this FIR	1 to 9; defaults to 1 if not set
TP_DUAL_IP	Unsigned int	Use dual inputs (may increase throughput for symmetrical and halfband filters by avoiding load contention by using a second RAM bank for input)	Range 0 (single input), 1 (dual input); defaults to 0 if not set
TT_USE_COEFF_RELOAD	Unsigned int	Enable reloadable coefficient feature; additional 'coeff' RTP port will appear on the graph	0 (no reload), 1 (use reloads); defaults to 0 if not set
TT_NUM_OUTPUTS	Unsigned int	Number of fir output ports	>1

Parameters Supported by FIR Filters (2)

Parameter Name	Type	Description	Range
TP_API	Unsigned int	I/O interface port type	0 = Window 1 = Stream
TP_SSR	Unsigned int	Parallelism factor	Min = 1; default to 1; Max = limited by resource availability
TP_PARA_INTERP_POLY	Unsigned int	Number of interpolation polyphases that produce output data in parallel	Min = 1; Max= TP_INTERPOLATE_FACTOR Must be factor of interpolate factor
TT_PARA_DECI_POLY	Unsigned int	Number of decimation polyphases into which data is split into for parallel computation	Min = 1; Max= TP_DECIMATE_FACTOR Must be factor of decimation factor

Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - **MATLAB® Model of a FIR/FFT Design**
 - Design 1: FIR/FFT Design for $SSR < 1$
 - Design 2: FIR/FFT Design for $SSR > 1$
 - AI Engine Design Using the Vitis™ IDE
 - AI Engine Design Using Vitis Model Composer
 3. Summary

FIR/FFT Design – MATLAB® Model

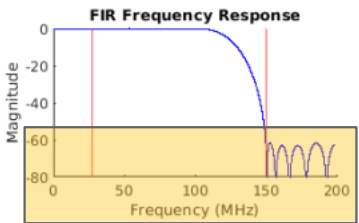
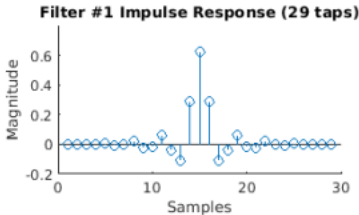
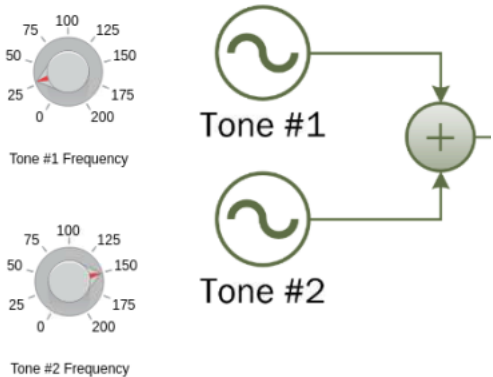
FIR: Suppress 1-tone from a 2-tone input signal

FFT: Produce spectrum plot

System Requirement

Parameters	Design 1	Design 2
SSR	SSR<1	SSR>1
Sampling Rate (Fs)	400 Msps	2000 Msps
Tone Amplitudes	0.2 & 0.4	
Tone Frequencies (MHz)	25 & 150	100 & 750
Taps	29 taps (-60 dBc)	

Vitis DSP Library Course: Session #2



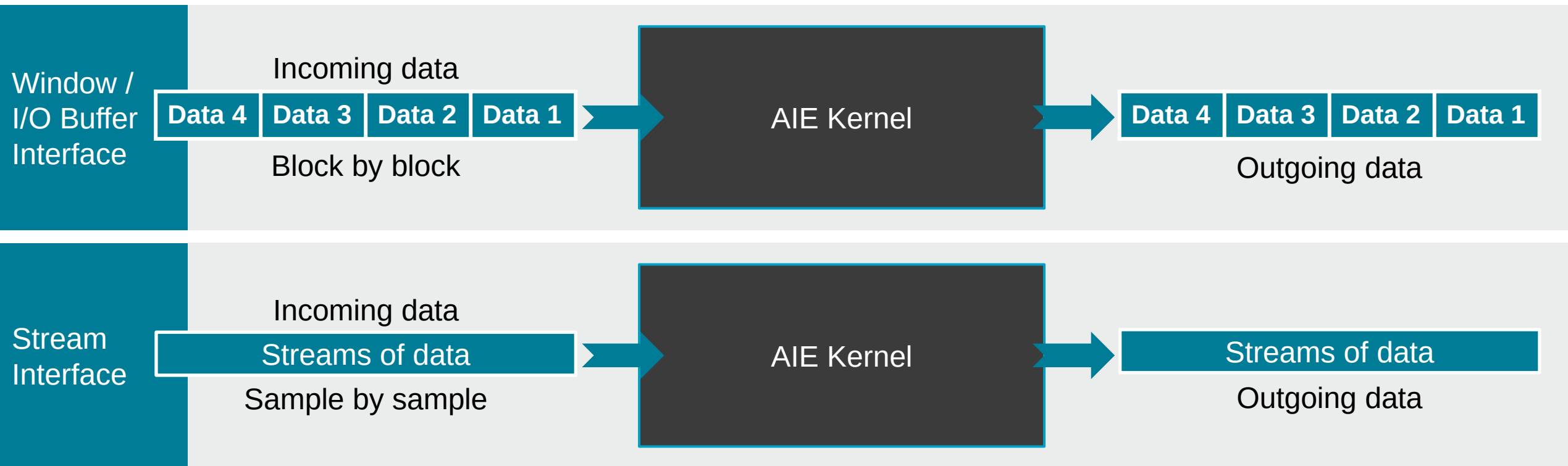
Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - **Design 1: FIR/FFT Design for SSR<1**
 - Design 2: FIR/FFT Design for SSR>1
 - AI Engine Design Using the Vitis™ IDE
 - AI Engine Design Using Vitis Model Composer
 3. Summary

Types of Interfaces for AI Engine Kernels

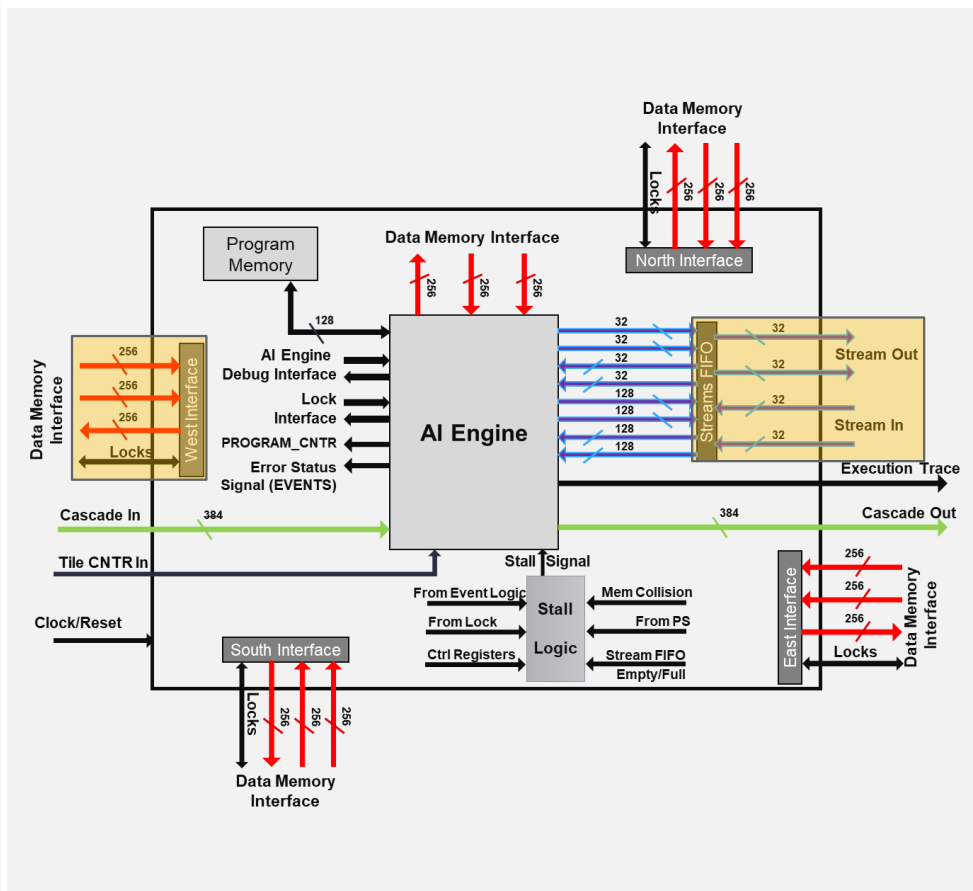


Two Types of Interfaces



Windows and I/O Buffers vs. Stream Interfaces

	Window Interface	Stream Interface
Latency	- Takes ~10-20 cycles for each window to get the data lock - Filling the data	No latency (very minimal)
Read & Write	Two 256-bit reads and one 256-bit write	Two 32-bit reads and one 32-bit write (parallel)
Sample Size	Window size – max. 32 KB	NA
Processing Data	Sequential or random	Sequential
Challenges	- Lock stall - Memory conflicts (ex: accessing the same bank by multiple AIEs)	Backpressure (stream stall)



Design 1: FIR/FFT Design Example Using Windows (SSR<1)

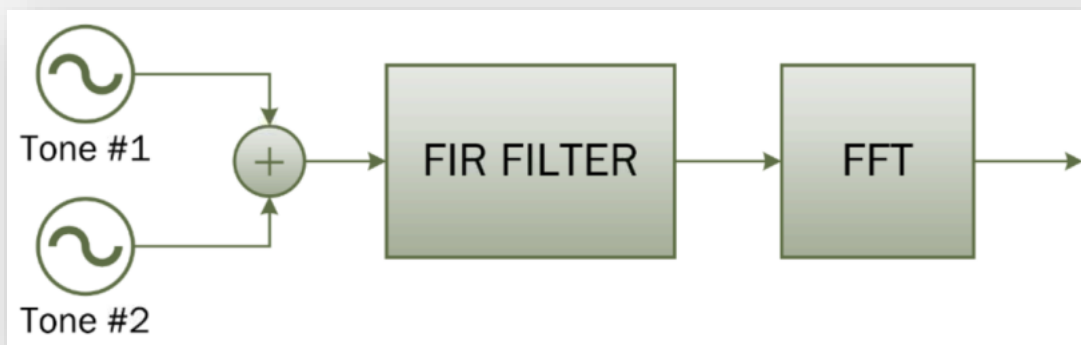
Step 1: Vitis™ DSP Libraries

Step 2: Configure the FIR Parameters

Step 3: Configure the FFT Parameters

Step 4: Passing the Parameters

Step 5: Connect the Kernels



Single rate, symmetrical

`xf::dsp::aie::fir::sr_sym::fir_sr_sym_graph`

Single channel,
decimation-in-time

`xf::dsp::aie::fft::dit_1ch`

Design 1: FIR/FFT Design Example Using Windows (SSR<1)

Step 1: Vitis™ DSP Libraries

Step 2: Configure the FIR Parameters

Step 3: Configure the FFT Parameters

Step 4: Passing the Parameters

Step 5: Connect the Kernels

Symmetrical Single Rate FIR filter

fir_sr_sym_graph

FIR Parameters

TP_FIR_LEN	29	Total number of taps
TP_SHIFT	15	Number of bits to shift accumulation down by before output
TP_RND	5	Symmetrical to zero
TP_INPUT_WINDOW_VSIZE	2048	Window size
TP_CASC_LEN	1	Number of cascaded kernels to use for this FIR
TP_DUAL_IP	0	Use dual inputs if set as 1
TP_USE_COEFF_RELOAD	0	No reload
TP_NUM_OUTPUTS	1	Number of ports to broadcast the output
TP_API	0	Set to windows mode
TP_SSR	1	Scale throughput using parallelization

```
namespace fir1 {
    static constexpr int WIN_SIZE = 2048;
    typedef cint16      TT_DATA;
    typedef int16       TT_COEFF;
    static constexpr int TP_FIR_LEN      = 29;
    static constexpr int TP_SHIFT       = 15;
    static constexpr int TP_RND         = 5;
    static constexpr int TP_INPUT_WINDOW_VSIZE = WIN_SIZE;
    static constexpr int TP_CASC_LEN    = 1;
    static constexpr int TP_DUAL_IP     = 0;
    static constexpr int TP_USE_COEFF_RELOAD = 0;
    static constexpr int TP_NUM_OUTPUTS = 1;
    static constexpr int TP_API         = 0;
    static constexpr int TP_SSR        = 1;
};
```

Design 1: FIR/FFT Design Example Using Windows (SSR<1)



FFT Parameters		
TP_POINT_SIZE	256	Number of samples in a frame to be processed
TP_FFT_NIFFT	1	Performs FFT (1)
TP_SHIFT	6	Excludes twiddle shift
TP_CASC_LEN	1	Number of kernels to use
TP_DYN_PT_SIZE	0	Not to use runtime point size determine
TP_WINDOW_SIZE	2048	Number of samples in the input window
TP_API	0	Set to windows mode
TP_PARALLEL_POWER	0	Selects the parallelism factor as a power of 2 - 0 to 4 (1 to 16 kernel -lanes of processing)

```
namespace fft1 {
  typedef cint16
  typedef cint16
  static constexpr int TP_POINT_SIZE
  static constexpr int TP_FFT_NIFFT
  static constexpr int TP_SHIFT
  static constexpr int TP_CASC_LEN
  static constexpr int TP_DYN_PT_SIZE
  static constexpr int TP_WINDOW_SIZE
  static constexpr int TP_API
  static constexpr int TP_PARALLEL_POWER
};

TT_TYPE;
TT_TWIDDLE;
= 256;
= 1;
= 6;
= 1;
= 0;
= fir1::WIN_SIZE;
= 0;
= 0;
```

Design 1: FIR/FFT Design Example Using Windows (SSR<1)

Step 1: Vitis™ DSP Libraries

Step 2: Configure the FIR Parameters

Step 3: Configure the FFT Parameters

Step 4: Passing the Parameters

Step 5: Connect the Kernels

FIR and FFT parameters based on the design requirement

```
class fir1_graph : public graph {
public:
private:
    // Filter taps:
    std::vector<fir1::TT_COEFF> m_taps = std::vector<fir1::TT_COEFF>(FIR1_COEFF);

    // Filter class:
    using TT_FIR = fir_sr_sym_graph<fir1::TT_DATA, fir1::TT_COEFF, fir1::TP_FIR_LEN, fir1::TP_SHIFT,
        fir1::TP_RND, fir1::TP_INPUT_WINDOW_VSIZE, fir1::TP_CASC_LEN,
        fir1::TP_DUAL_IP, fir1::TP_USE_COEFF_RELOAD, fir1::TP_NUM_OUTPUTS,
        fir1::TP_API, fir1::TP_SSR>;

    // FFT class:
    using TT_FFT = fft_ifft_dit_1ch_graph<fft1::TT_TYPE, fft1::TT_TWIDDLE, fft1::TP_POINT_SIZE,
        fft1::TP_FFT_NIFFT, fft1::TP_SHIFT, fft1::TP_CASC_LEN,
        fft1::TP_DYN_PT_SIZE, fft1::TP_WINDOW_SIZE, fft1::TP_API,
        fft1::TP_PARALLEL_POWER>;

public:
    input plio firt_i;
```

```
namespace fir1 {
    static constexpr int WIN_SIZE = 2048;
    typedef cint16 TT_DATA;
    typedef int16 TT_COEFF;
    static constexpr int TP_FIR_LEN = 29;
    static constexpr int TP_SHIFT = 15;
    static constexpr int TP_RND = 5;
    static constexpr int TP_INPUT_WINDOW_VSIZE = WIN_SIZE;
    static constexpr int TP_CASC_LEN = 1;
    static constexpr int TP_DUAL_IP = 0;
    static constexpr int TP_USE_COEFF_RELOAD = 0;
    static constexpr int TP_NUM_OUTPUTS = 1;
    static constexpr int TP_API = 0;
    static constexpr int TP_SSR = 1;
};
```

```
namespace fft1 {
    typedef cint16 TT_TYPE;
    typedef cint16 TT_TWIDDLE;
    static constexpr int TP_POINT_SIZE = 256;
    static constexpr int TP_FFT_NIFFT = 1;
    static constexpr int TP_SHIFT = 6;
    static constexpr int TP_CASC_LEN = 1;
    static constexpr int TP_DYN_PT_SIZE = 0;
    static constexpr int TP_WINDOW_SIZE = fir1::WIN_SIZE;
    static constexpr int TP_API = 0;
    static constexpr int TP_PARALLEL_POWER = 0;
};
```

Design 1: FIR/FFT Design Example Using Windows (SSR<1)

Step 1: Vitis™ DSP Libraries

Step 2: Configure the FIR Parameters

Step 3: Configure the FFT Parameters

Step 4: Passing the Parameters

Step 5: Connect the Kernels

```

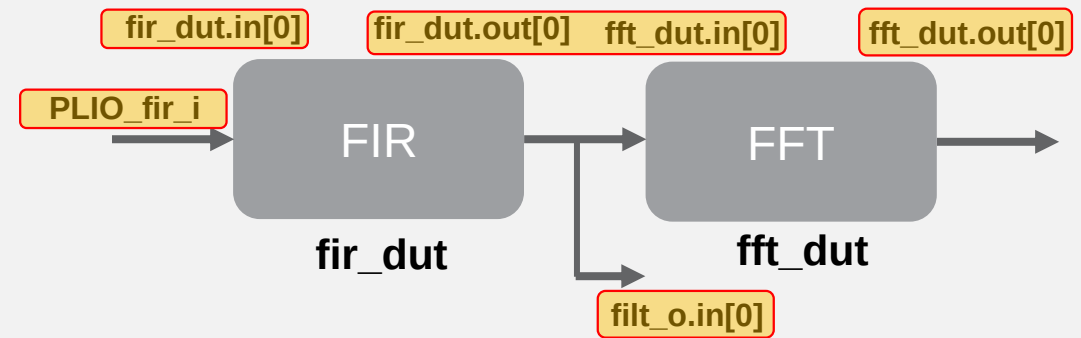
ffl::IP_PARALLEL_POWER>;

public:
  input_plio filt_i;
  output_plio filt_o;
  output_plio fft_o;

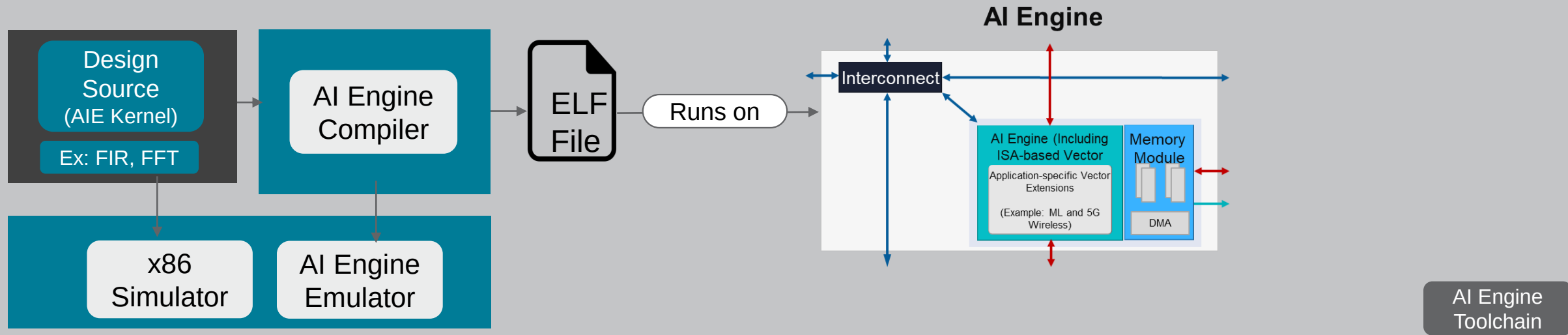
  TT_FIR fir_dut;
  TT_FFT fft_dut;

  fir1_graph(void) : fir_dut( m_taps ), fft_dut()
  {
    filt_i = input_plio::create("PLIO_fir_i", plio_64_bits, "data/sig_i.txt");
    filt_o = output_plio::create("PLIO_fir_o", plio_64_bits, "data/fir_o.txt");
    fft_o = output_plio::create("PLIO_fft_o", plio_64_bits, "data/fft_o.txt");
    connect<window<4*fir1::WIN_SIZE>>(<filt_i.out[0], fir_dut.in[0]>);
    connect<window<4*fir1::WIN_SIZE>>(<fir_dut.out[0], fft_dut.in[0]>);
    connect<window<4*fir1::WIN_SIZE>>(<fir_dut.out[0], filt_o.in[0]>);
    connect<window<4*fir1::WIN_SIZE>>(<fft_dut.out[0], fft_o.in[0]>);
  }
};

```



Development Flow – AI Engine Toolchain



Step 1

Compile the design

Step 2

Simulate the design (x86sim)

Step 3

Simulate the design (AIEsim)

Step 4

Analyze the performance

Makefile – Design Source, Platform, and DSPLib

Platform

```
PLATFORM_VER := xilinx vck190 base 202310 1
PLATFORM      := platform=${PLATFORM_REPO_PATHS}/${PLATFORM_VER}/${PLATFORM_VER}.xpfm
```

Design Source

```
PROJECT_PATH := $(shell pwd)
SIM_OUTPUT   := aiesimulator_output
LOG_FILE      := log
FIR1_APP      := fir1_app.cpp
AIE_OUTPUT    := libadf.a
SRC_PATH      := src
TARGET        := hw
WORK_AIESIM   := ${PROJECT_PATH}/build/hw/Work
WORK_X86SIM   := ${PROJECT_PATH}/build/x86sim/Work
```

DSPLib

```
DSPLIB_INCLUDE := --include=${DSPLIB_ROOT}/L2/include/aie \
                  --include=${DSPLIB_ROOT}/L1/include/aie \
                  --include=${DSPLIB_ROOT}/L1/src/aie
```

AI Engine
Compiler

x86
Simulator

AI Engine
Emulator

AI Engine
Toolchain

Makefile – AIE Compiler and Simulator

AIE Compiler

```
${AIE_OUTPUT}: ${SRC_PATH}/fir1_graph.h ${SRC_PATH}/${FIR1_APP}
```

```
ifeq ($(TARGET), x86sim)
    v++ -c --mode aie --config aie_config_x86sim.cfg ${DSPLIB_INCLUDE}
else ifeq ($(TARGET), hw)
    v++ -c --mode aie --config aie_config_hw.cfg ${DSPLIB_INCLUDE}
endif
```

X86 Simulator

```
x86sim:
```

```
    x86simulator --pkg-dir=${WORK_X86SIM} --dump |& tee -a ${LOG_FILE}
```

AIE Simulator

```
aiesim:
```

```
    aiesimulator --pkg-dir=${WORK_AIESIM} -o ${SIM_OUTPUT} |& tee -a ${LOG_FILE}
```

```
profile:
```

```
    aiesimulator --pkg-dir=${WORK_AIESIM} -o ${SIM_OUTPUT} --profile --online -wdb -ctf |& tee -a ${LOG_FILE}
```

```
throughput:
```

```
@../scripts/throughput.py aiesimulator_output/data/fir_o.txt --iscomplex
```

```
@../scripts/throughput.py aiesimulator_output/data/fft_o.txt --iscomplex
```

AI Engine
Compiler

x86
Simulator

AI Engine
Emulator

AI Engine
Toolchain

Compile the Design

Set the DSPLib path

`export DSPLIB_ROOT=/<PATH>/Vitis_Libraries/dsp`

Compile the design

`make aiecompile`

```
xsjapps54:/group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow $ make aiecompile
v++ -c --mode aie --config aie_config_hw.cfg --include=/group/techsup/cbalakr/dsp_lib/2022.2/Vitis_Libraries/dsp/L2/include/aie --include=/group/techsup/cbalakr/dsp_lib/2022.2/Vitis_Libraries/dsp/L1/include/aie --include=/group/techsup/cbalakr/dsp_lib/2022.2/Vitis_Libraries/dsp/L1/src/aie

***** v++ v2023.1 (64-bit)
**** SW Build 3860322 on 2023-05-04-06:32:48
** Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
** Copyright 2022-2023 Advanced Micro Devices, Inc. All Rights Reserved.

Running Dispatch Server on port: 33640
INFO: [v++ 60-1548] Creating build summary session with primary output /group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow/build/hw/Work/Work.aiecompiler_summary, at Sun Sep  3 20:53:33 2023
INFO: [v++ 60-895] Target platform: /proj/rdi/xbuilds/2023.1_released/internal_platforms/xilinx_vck190_base_202310_1/xilinx_vck190_base_202310_1.xpfm
INFO: [v++ 60-1578] This platform contains Xilinx Shell Archive '/proj/rdi/xbuilds/2023.1_released/internal_platforms/xilinx_vck190_base_202310_1/hw/hw_xsa'
INFO: [v++ 82-31] Launching aiecompiler: aiecompiler --config /group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow/build/hw/Work/aie_hw.cfg
aietools : /proj/gsd/vivado/Vitis/2023.1/aietools
```

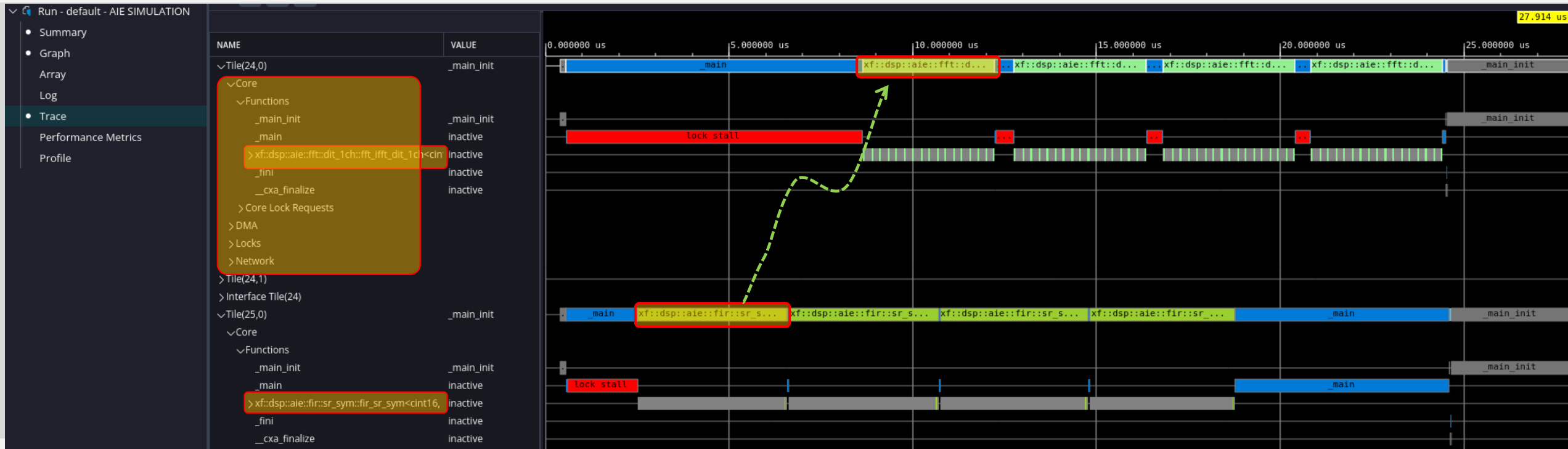
Simulate the Design



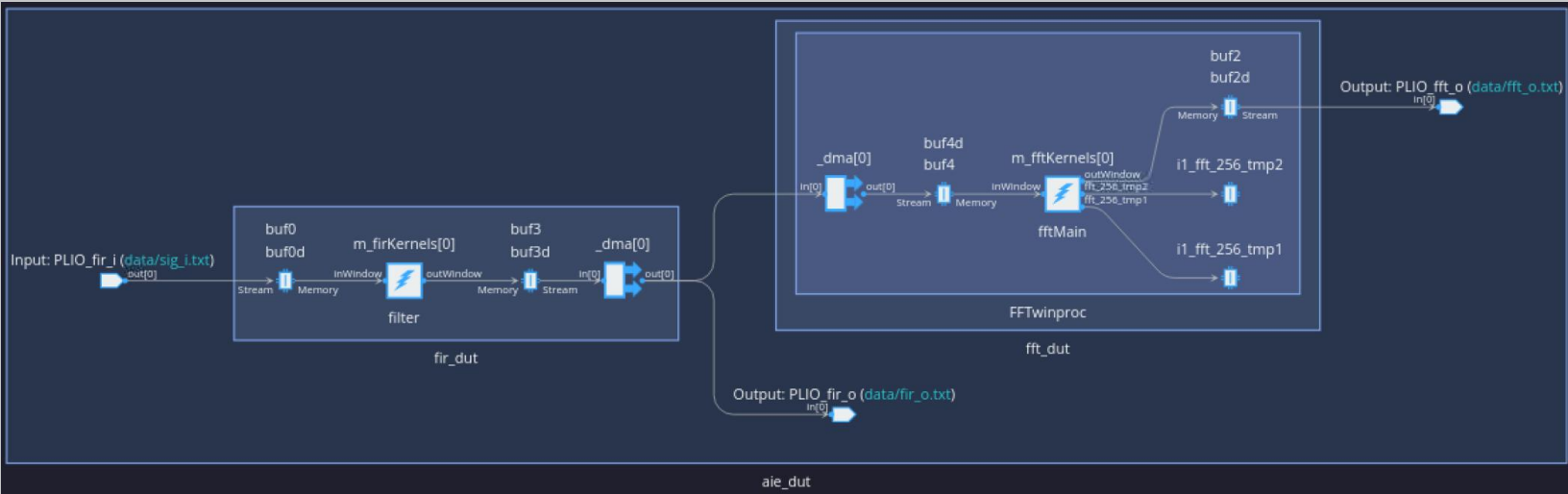
```
xsjapps54:/group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow $ make profile
aiesimulator --pkg-dir=/group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow/build/hw/Work -o aiesimulator
_output --profile --online -wdb -ctf |& tee -a log
Running AIE Simulator in Online mode...
AIEsim feature license is found.
INFO: Executing config: /group/techsup/cbalakr/github_cbalakr/my_projects/tsc_2023/ssr_lt1/makefile_flow/build/hw/Work/config/scsim_c
onfig.json
```

Analyze the Design – Trace View

Analyze the design

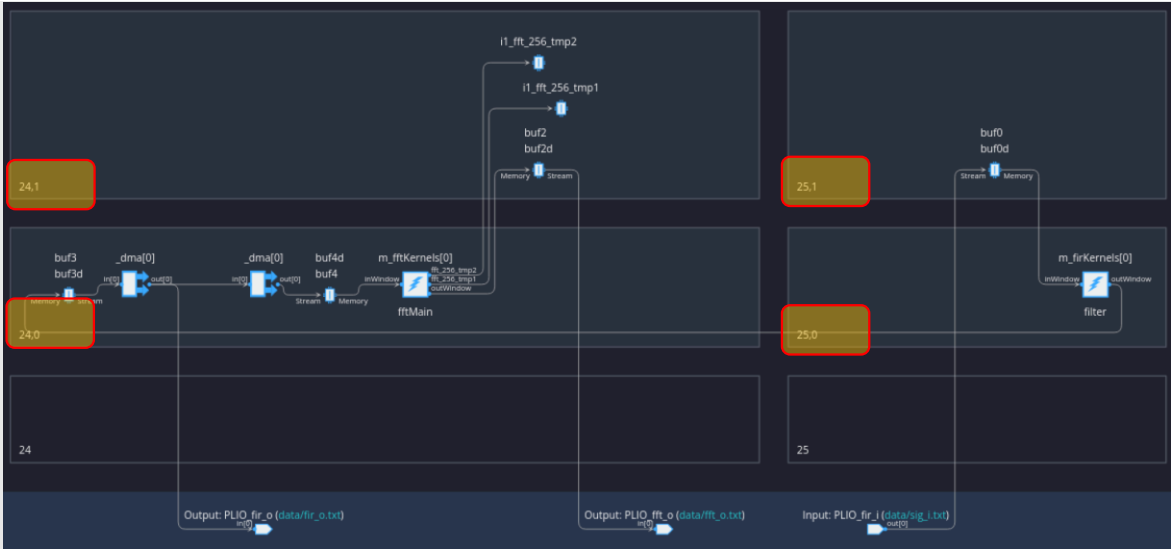
`vitis_analyzer aiesimulator_output/default.aierun_summary`

Analyze the Design – Graph View and Tile View



Graph View

Tile View



Verify the Performance

Verify the performance → make throughput

```
xsjapps59:/sales_exceed/session2 $ make throughput
```

FIR Raw Throughput: 574.96 (Msps)

FFT Raw Throughput: 574.96 (Msps)

Target → 400 Msps
Achieved → ~574 Msps

System Requirement		
Parameters	Design 1	Design 2
SSR	SSR < 1	SSR > 1
Sampling Rate (Fs)	400 Msps	2000 Msps
Tone Amplitudes	0.2 & 0.4	
Tone Frequencies (MHz)	25 & 150	100 & 750
Taps	29 taps (-60 dBc)	

Agenda

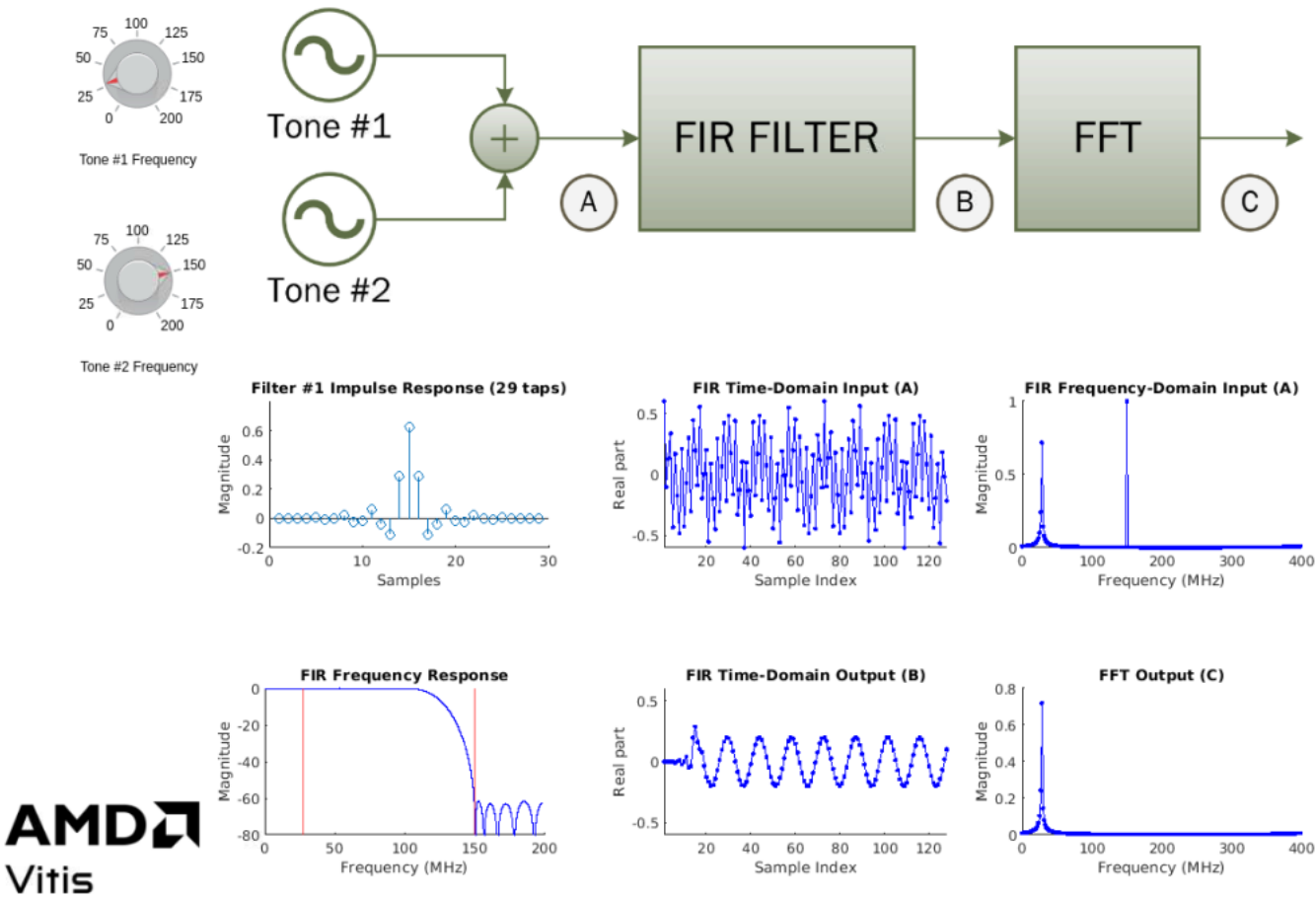
-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - Design 1: FIR/FFT Design for $SSR < 1$
 - **Design 2: FIR/FFT Design for $SSR > 1$**
 - AI Engine Design Using the Vitis™ IDE
 - AI Engine Design Using Vitis Model Composer
 3. Summary

Design 2: FIR/FFT Design Example Using Streams (SSR>1)

System Requirement

Parameters	Design 1	Design 2
SSR	SSR<1	SSR>1
Sampling Rate (Fs)	400 Msps	2000 Msps
Tone Amplitudes	0.2 & 0.4	
Tone Frequencies (MHz)	25 & 150	100 & 750
Taps	29 taps (-60 dBc)	

Vitis DSP Library Course: Session #2



Design 2: FIR/FFT Design Example Using Streams (SSR>1)



FIR Parameters			
Parameters	Design 1	Design 2	Description
TP_FIR_LEN	29		Total number of taps
TP_SHIFT	15		Number of bits to shift accumulation down by before output
TP_RND	5		Symmetrical to zero
TP_INPUT_WINDOW_VSIZE	2048		Window size
TP_CASC_LEN	1		Number of cascaded kernels to use for this FIR
TP_DUAL_IP	0		Use dual inputs if set as 1
TP_USE_COEFF_RELOAD	0		No reload
TP_NUM_OUTPUTS	1		Number of ports to broadcast the output
TP_API	0	1	Set to stream mode
TP_SSR	1	4	Scale throughput using parallelization

```
namespace fir2 {
    static constexpr int WIN_SIZE = 2048;
    typedef cint16      TT_DATA;
    typedef int16       TT_COEFF;
    static constexpr int TP_FIR_LEN      = 29;
    static constexpr int TP_SHIFT        = 15;
    static constexpr int TP_RND          = 5;
    static constexpr int TP_INPUT_WINDOW_VSIZE = WIN_SIZE;
    static constexpr int TP_CASC_LEN     = 1;
    static constexpr int TP_DUAL_IP      = 0;
    static constexpr int TP_USE_COEFF_RELOAD = 0;
    static constexpr int TP_NUM_OUTPUTS  = 1;
    static constexpr int TP_API          = 1;
    static constexpr int TP_SSR          = 4;
}
```

Design 2: FIR/FFT Design Example Using Streams (SSR>1)

Step 1: Vitis™ DSP Libraries

Step 2: Configure the FIR Parameters

Step 3: Configure the FFT Parameters

Step 4: Passing the Parameters

Step 5: Connect the Kernels

Single Channel DIT FFT

fft_ifft_dit_1ch_graph

FFT Parameters

Parameters	Design 1	Design 2	Description
TP_POINT_SIZE	256		Number of samples in a frame to be processed
TP_FFT_NIFFT	1		Performs FFT (1)
TP_SHIFT	6		Excludes twiddle shift
TP_CASC_LEN	1	3	Number of kernels to use
TP_DYN_PT_SIZE	0		Not to use runtime point size determine
TP_WINDOW_SIZE	2048		Number of samples in the input window
TP_API	0	1	Set to stream mode
TP_PARALLEL_POWER	0	1	Selects the parallelism factor as a power of 2 - 0 to 4 (1 to 16 kernel -lanes of processing)

```
namespace fft2 {
    static constexpr int WIN_SIZE = 2048;
    typedef cint16 TT_TYPE;
    typedef cint16 TT_TWIDDLE;
    static constexpr int TP_POINT_SIZE = 256;
    static constexpr int TP_FFT_NIFFT = 1;
    static constexpr int TP_SHIFT = 6;
    static constexpr int TP_CASC_LEN = 3;
    static constexpr int TP_DYN_PT_SIZE = 0;
    static constexpr int TP_WINDOW_SIZE = WIN_SIZE;
    static constexpr int TP_API = 1;
    static constexpr int TP_PARALLEL_POWER = 1;

    static constexpr int NPORTS_0 = 1 << (TP_PARALLEL_POWER+1);
};
```

Summary of Design 1 vs. Design 2

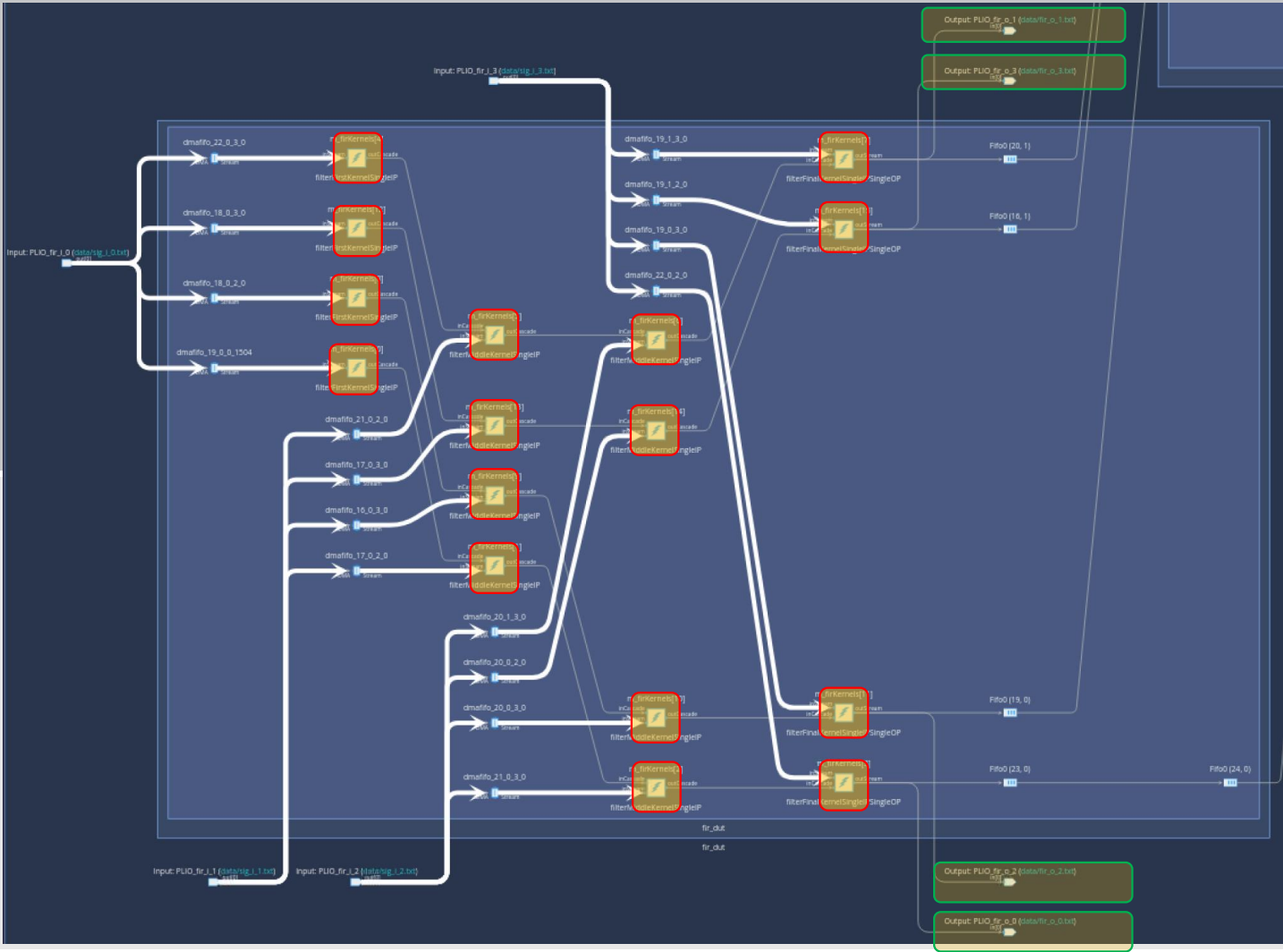
Design 1 (SSR<1)	Design 2 (SSR>1)
FIR Parameters: - TP_API = 0 - TP_SSR = 1	FIR Parameters: - TP_API = 1 - TP_SSR = 4
FFT Parameters: - TP_CASC_LEN = 1 - TP_API = 0 (window) - TP_PARALLEL_POWER = 0	FFT Parameters: - TP_CASC_LEN = 3 - TP_API = 1 (stream) - TP_PARALLEL_POWER = 1
<pre>fir1_graph(void) : fir_dut(m_taps), fft_dut() { filt_i = input_plio::create("PLIO_fir_i", plio_64_bits, "data/sig_i.txt"); filt_o = output_plio::create("PLIO_fir_o", plio_64_bits, "data/fir_o.txt"); fft_o = output_plio::create("PLIO_fft_o", plio_64_bits, "data/fft_o.txt"); connect<window>(*fir1::WIN_SIZE>>(filt_i.out[0], fir_dut.in[0]); connect<window>(*fir1::WIN_SIZE>>(fir_dut.out[0], fft_dut.in[0]); connect<window>(*fir1::WIN_SIZE>>(fir_dut.out[0], filt_o.in[0]); connect<window>(*fir1::WIN_SIZE>>(fft_dut.out[0], fft_o.in[0]); }</pre>	<pre> filt_i[ii] = input_plio::create(plio_fir_i, plio_64_bits, fname_fir_i); filt_o[ii] = output_plio::create(plio_fir_o, plio_64_bits, fname_fir_o); fft_o[ii] = output_plio::create(plio_fft_o, plio_64_bits, fname_fft_o); connect<stream>(filt_i[ii].out[0], fir_dut.filt_i[ii]); connect<stream>(fir_dut.filt_o[ii], filt_o[ii].in[0]); // Note: broad connect<stream>(fir_dut.filt_o[ii], fft_dut.fft_i[ii]); // Note: broad connect<stream>(fft_dut.fft_o[ii], fft_o[ii].in[0]); } }</pre>

Analyze the Design – Trace View

Analyze the design  vitis_analyzer aiesimulator_output/default.aierun_summary



Analyze the Design – Graph View



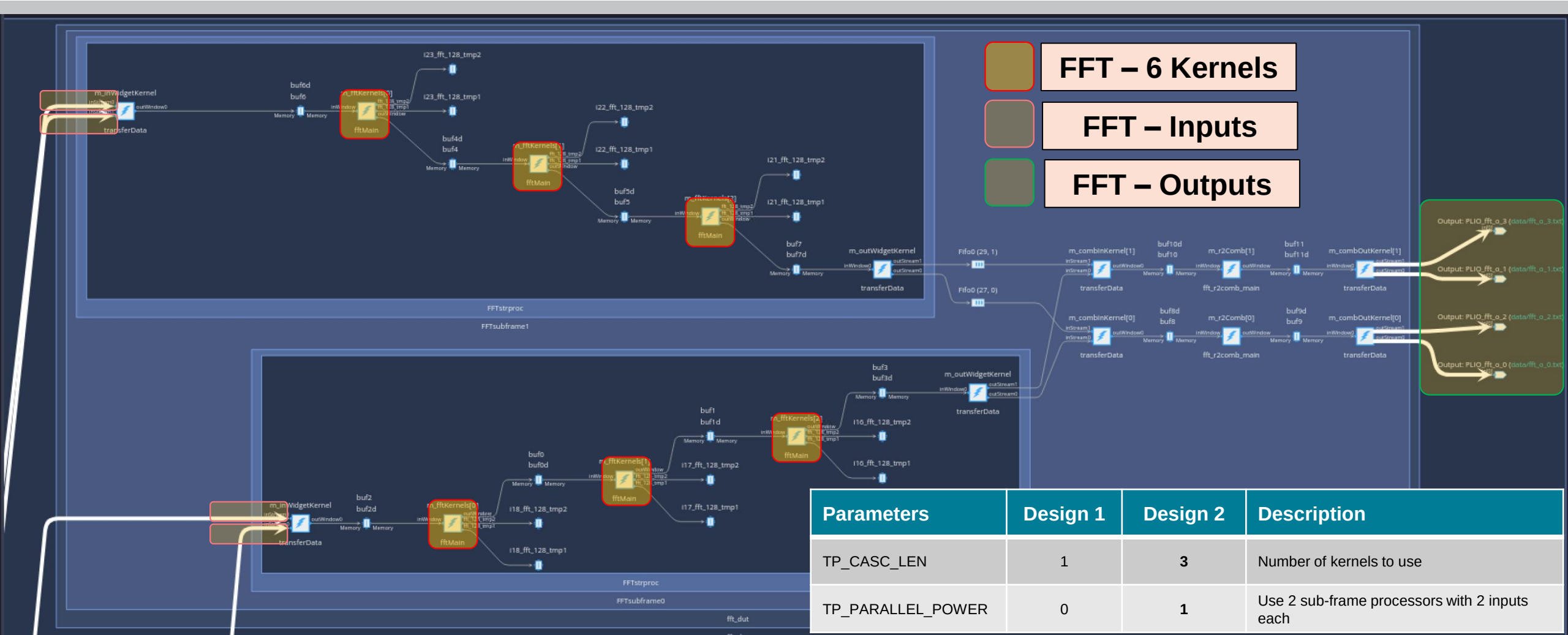
TP_SSR = 4
4 x 4 = 16 kernels

Parameters	Design 1	Design 2	Description
TP_SSR	1	4	Scale throughput

 **FIR – 16 Kernels**

 **FIR – Outputs**

Analyze the Design – Graph View – FFT Tiles



Verify the Performance

Verify the design

make throughput

```
xsjapps59:/sales_exceed/session3 $ make throughput_all2
```

FIR	Raw Throughput:	807.57	(Msps)
	Raw Throughput:	798.75	(Msps)
	Raw Throughput:	805.03	(Msps)
	Raw Throughput:	803.14	(Msps)
FFT	Raw Throughput:	705.23	(Msps)
	Raw Throughput:	704.75	(Msps)
	Raw Throughput:	705.23	(Msps)
	Raw Throughput:	704.75	(Msps)

Target → 2000 Msps

Achieved → ~2800 Msps

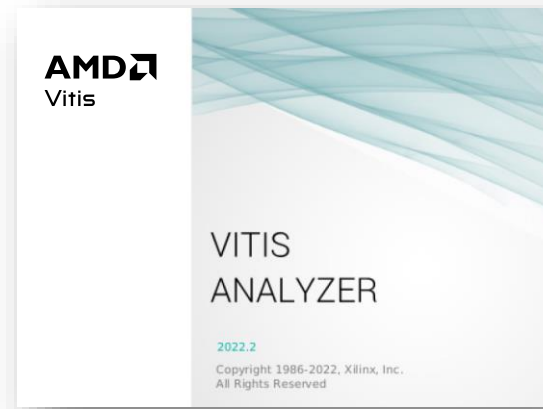
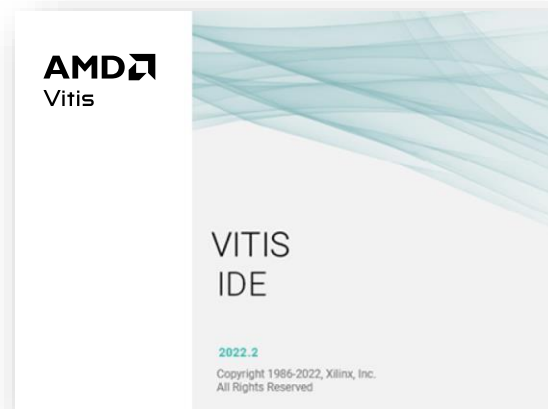
System Requirement		
Parameters	Design 1	Design 2
SSR	SSR < 1	SSR > 1
Sampling Rate (Fs)	400 Msps	2000 Msps
Tone Amplitudes	0.2 & 0.4	
Tone Frequencies (MHz)	25 & 150	100 & 750
Taps	29 taps (-60 dBc)	

Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - Design 1: FIR/FFT Design for $SSR < 1$
 - Design 2: FIR/FFT Design for $SSR > 1$
 - **AI Engine Design Using the Vitis™ IDE**
 - AI Engine Design Using Vitis Model Composer
 3. Summary

What Is the New AMD Vitis™ Unified IDE?

AMD Vitis™ Unified Software Platform – IDE Refresh (1)



**Vitis™ core
development kit**

AIE Compiler
HLS Compiler
GCC Compiler
HW Linker

Analyzers

Debuggers

AMD Vitis™ Unified Software Platform – IDE Refresh (2)

Vitis™ IDE refresh mostly updates the graphical user interface (GUI)



Core development tools and workflow don't change

AIE Compiler
HLS Compiler

GCC Compiler
HW Linker

Analyzers

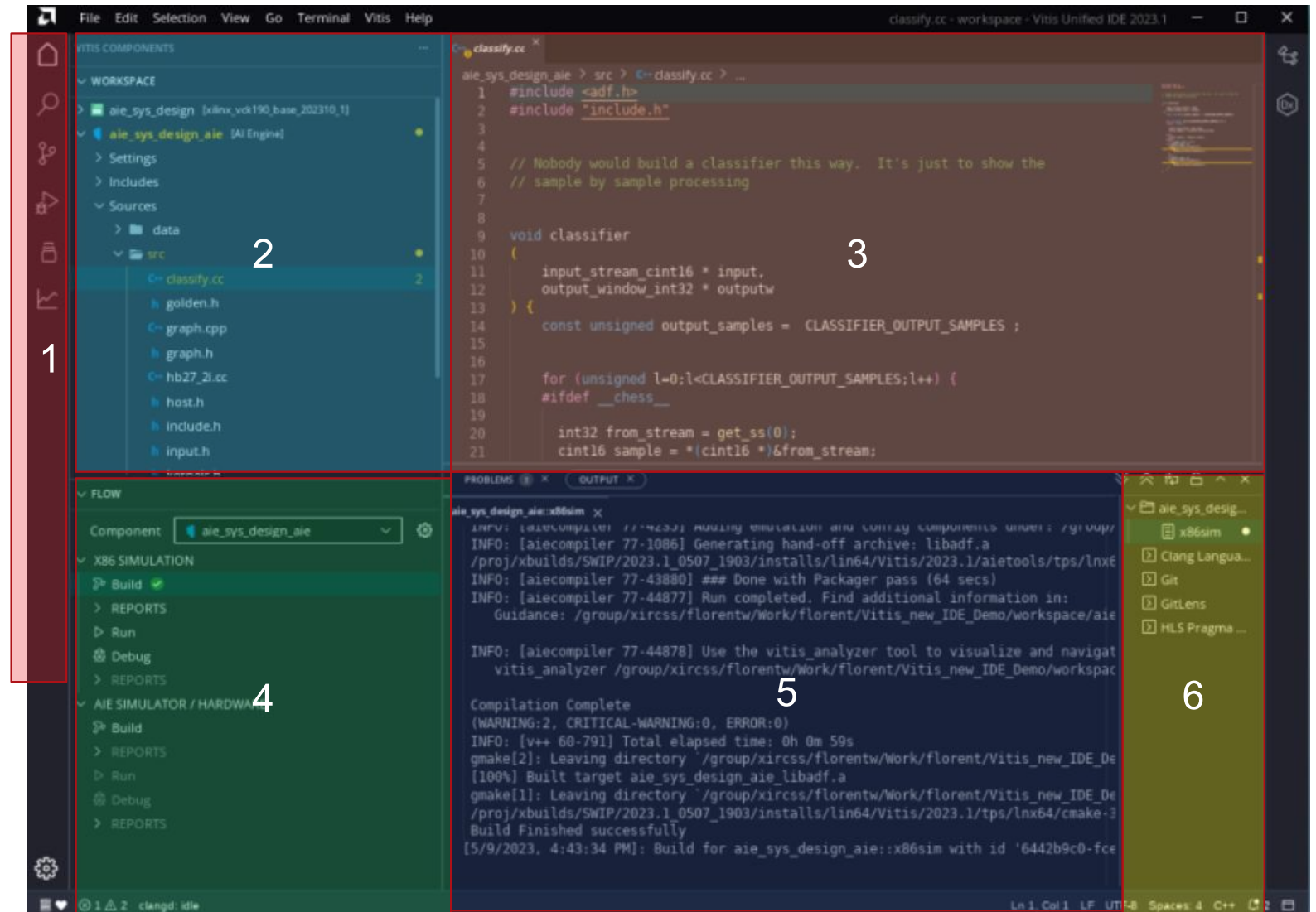
Debuggers

How Can I Use the New AMD Vitis™ Unified IDE?

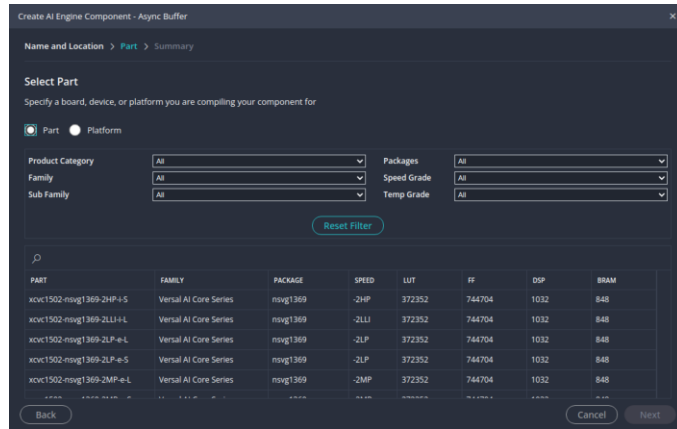
- Comes installed with 2023.1 as a preview along with the classic IDE
- Users just need to add the option --new to start the new IDE instead of classic
 - `vitis --workspace <workspace>` to start the classic IDE
 - `vitis --new -w <workspace>` to start the new IDE
- Note that workspaces are NOT interchangeable

Features of the New AMD Vitis™ IDE

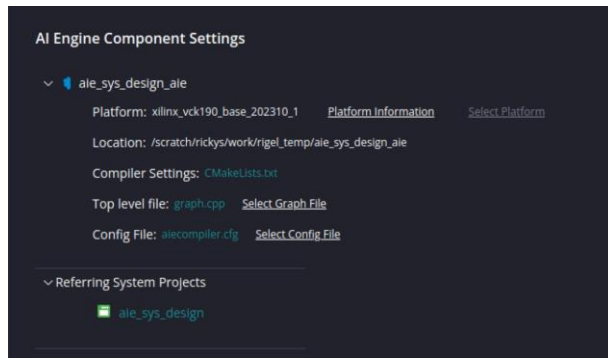
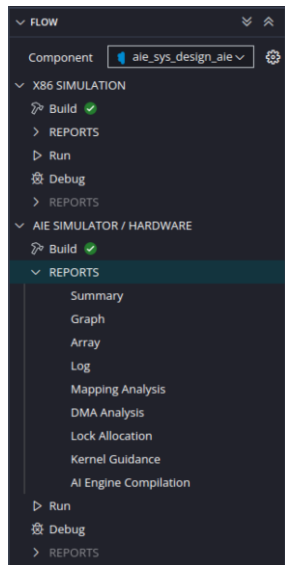
1. Toolbar
2. Component Explorer
3. Central Editor
4. Flow Navigator
5. Console, Output, Terminal
6. Index of Transcripts



AMD Vitis™ Unified IDE Enables Component Design (Bottom Up)

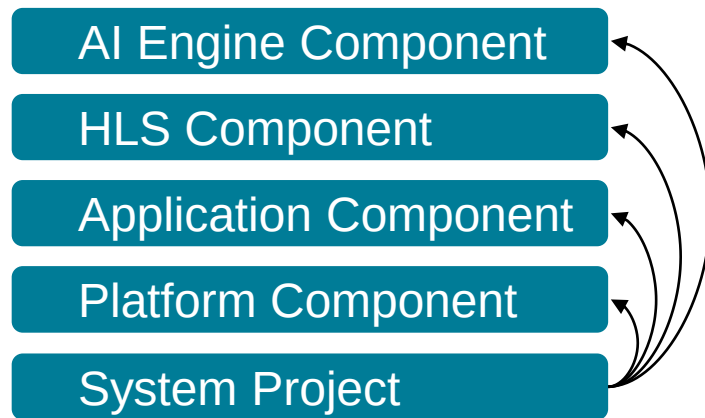


- AI Engine component can be created in isolation from the rest of the system
 - Create with platform or with device (part)
 - Run x86sim or aiesim
 - Debugging
 - Build the application targeting HW



- Component-specific Flow Navigator and Settings
 - Reports are close to build/run actions—easy to follow
- User can link the AI Engine component to a system later in the flow

Optional System Project



- System designer links AI Engine component, HLS component, and RTL kernels
 - Only needed during system integration and verification

Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

Step 4:
Add DSPLib

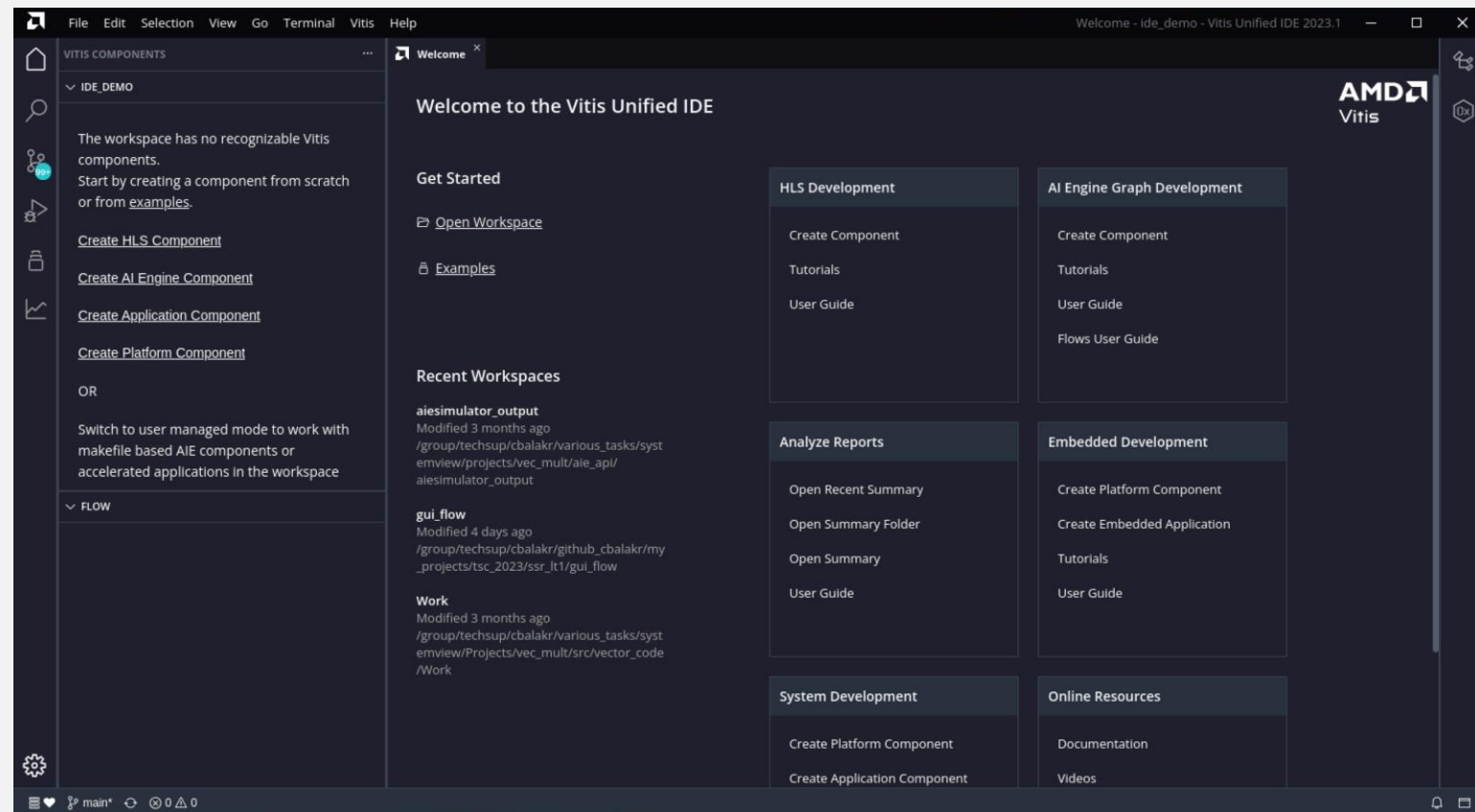
Step 5: X86SIM
Compile &
Simulate

Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

`vitis --new -w <PATH_TO_WORKSPACE>`

- Launch Vitis GUI
- Open workspace



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

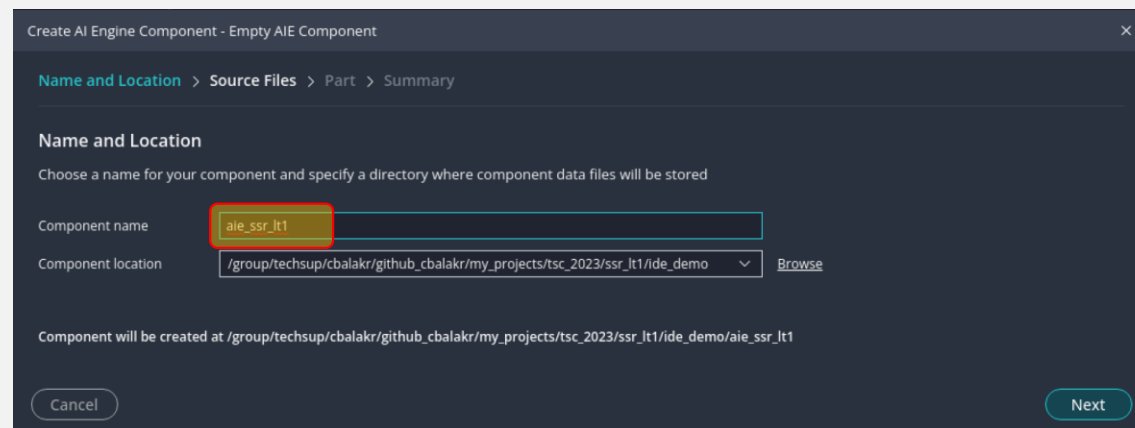
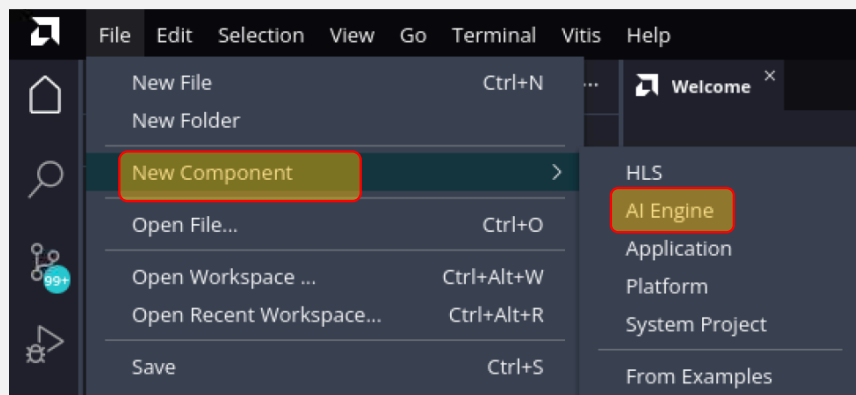
Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

- Create AIE component
- Enter the component name



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

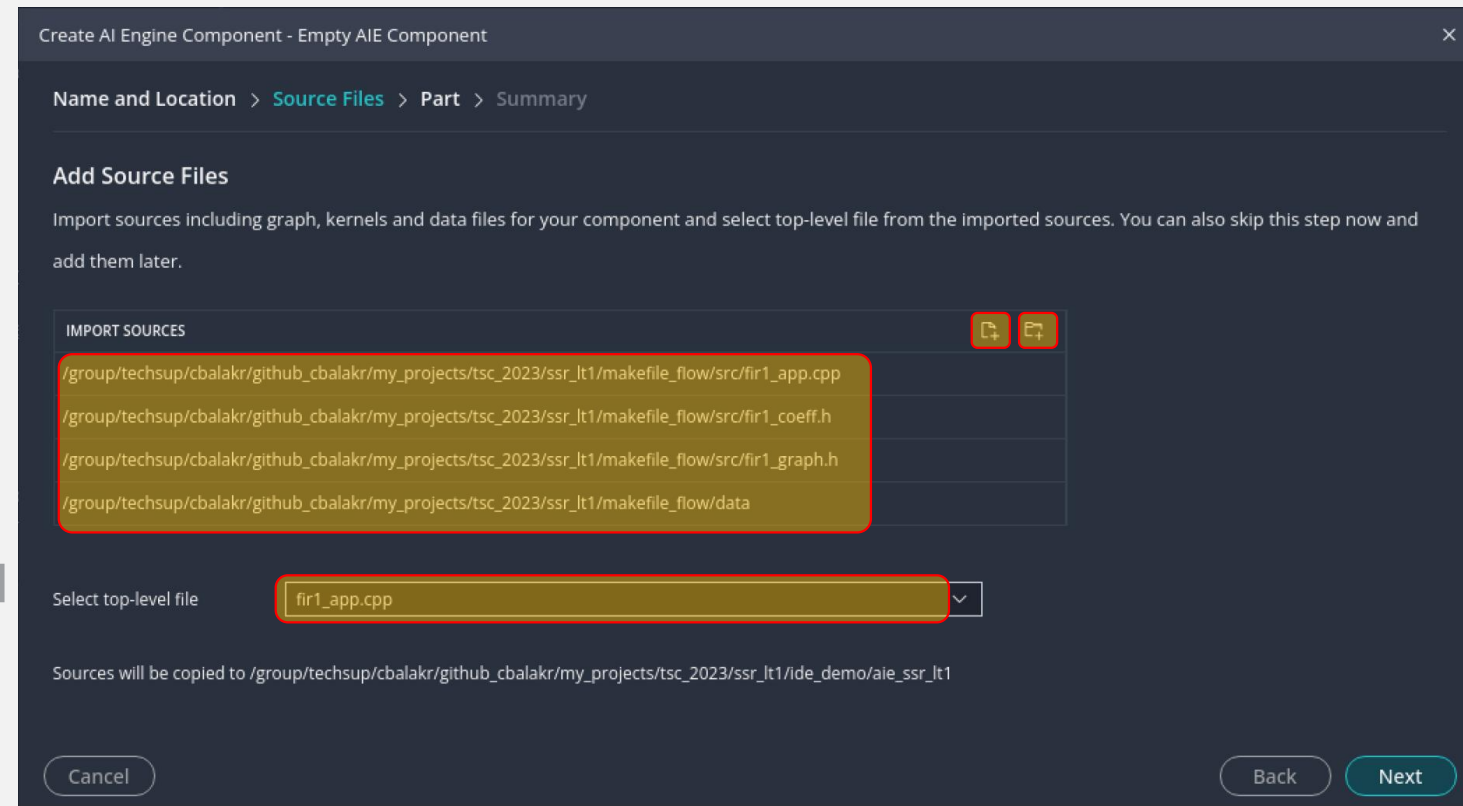
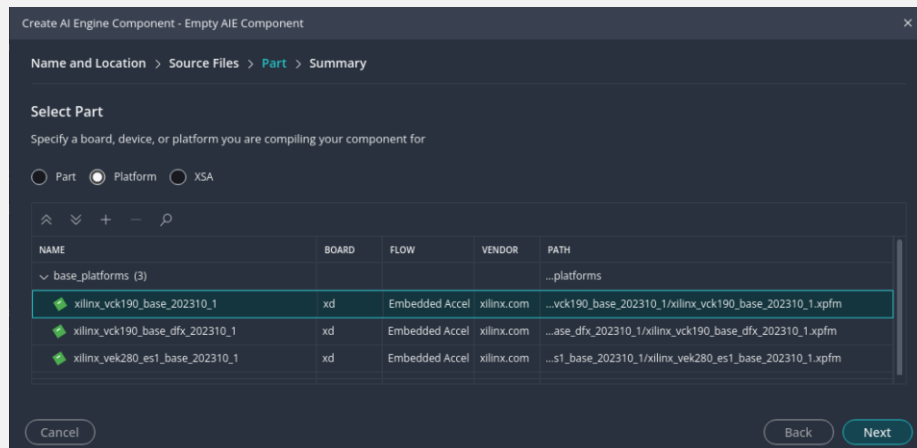
Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

- Add the files by clicking 
 - fir1_app.cpp, fir1_coeff.h, fir1_graph.h
- Add the folder by clicking 
 - data
- Select the platform



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

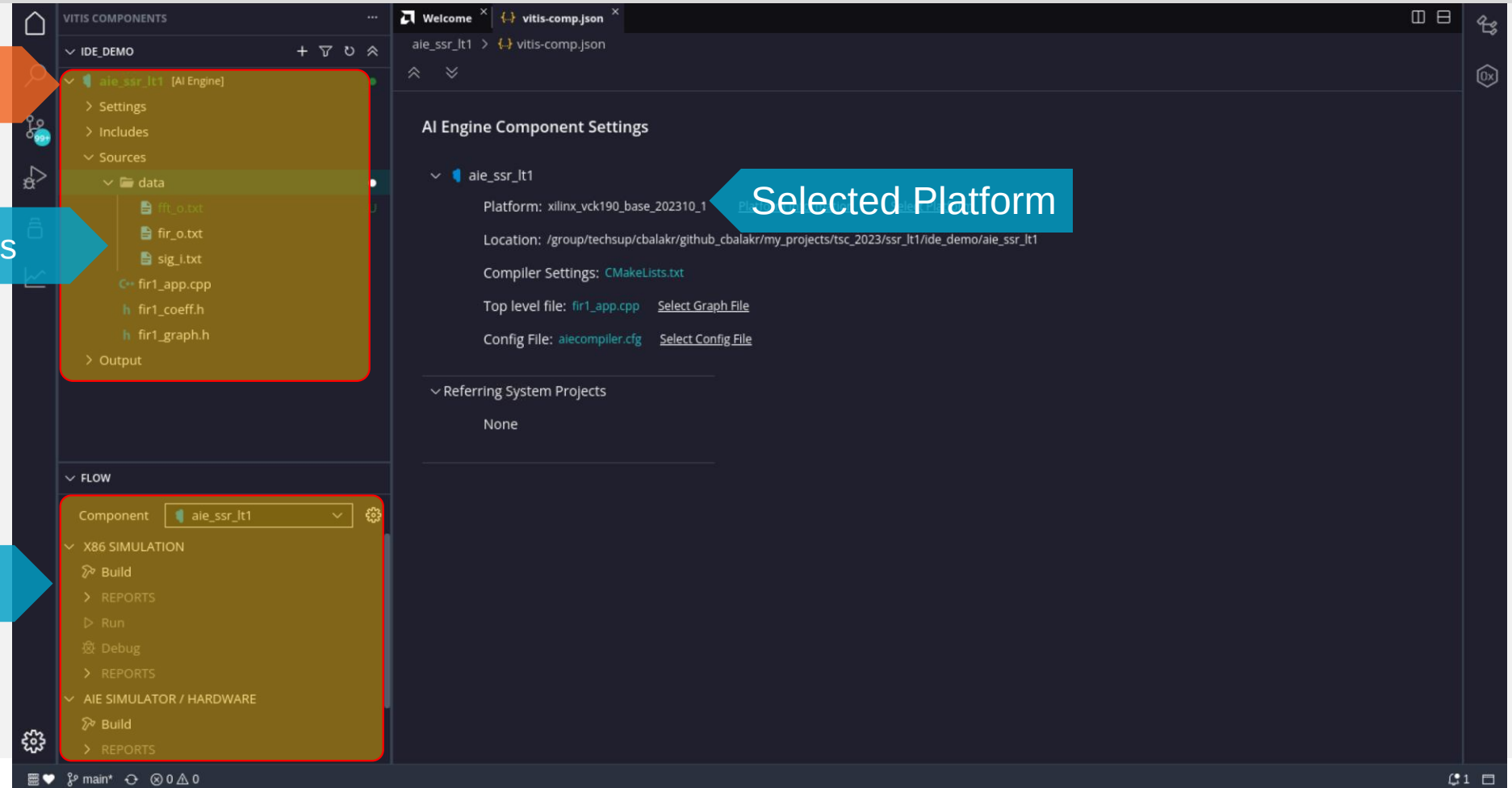
Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

AIE Component

Added Design Files

Flow Navigator



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

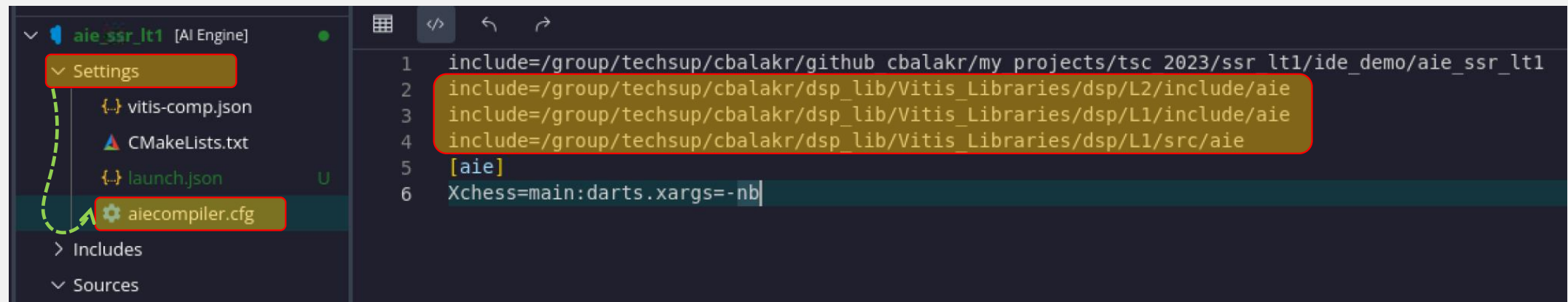
Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

- Expand *Settings* and open `aiecompiler.cfg` file
- Add the DSPLib path



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

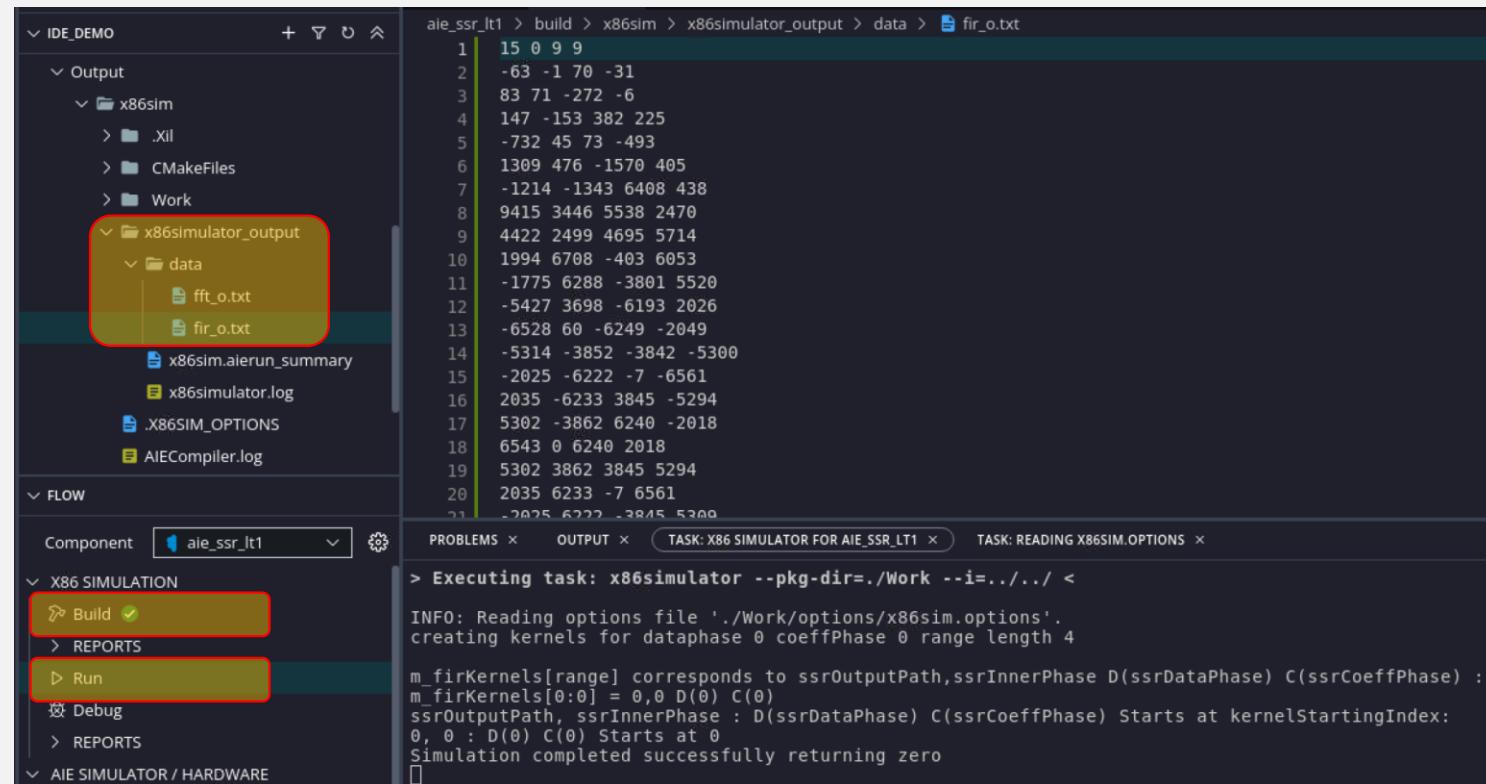
Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

Compile and verify
for x86 simulation

Verify the functionality

- Under X86 SIMULATION
 - Click **Build** to compile the project for x86 simulation
 - Click **Run** for X86 simulation
- Verify the output directory
 - x86simulator_output > data
 - Output files generated: fir_o.txt and fft_o.txt
 - Compare the results with the golden reference



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

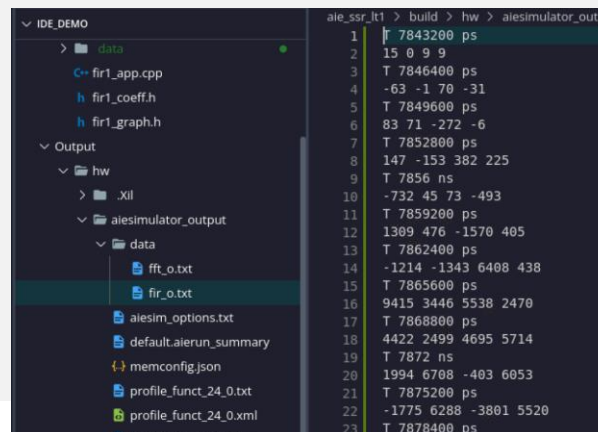
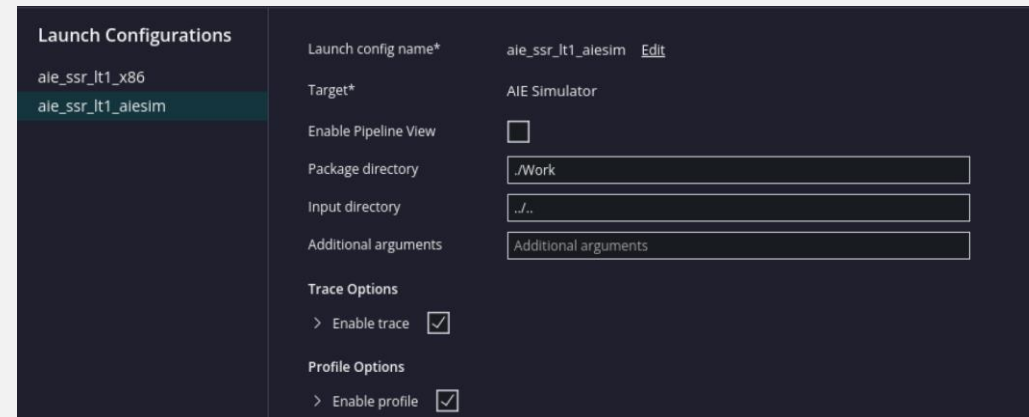
Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

Compile and verify
for AIE simulation

Emulate the design on SystemC model of AIE

- Under AIE SIMULATOR/HARDWARE
 - Click **Build** to compile the project
 - Configure the simulation
 - Enable trace and profile
 - Click **Run** for AIE simulation
- Verify the output directory
 - hw > aiesimulator_output > data
 - Output files generated: fir_o.txt and fft_o.txt
 - Compare the results with the golden reference



Vitis Unified IDE – AIE Development Flow

Step 1:
Launch Vitis™
IDE

Step 2:
Create AIE Comp

Step 3:
Add Design Files

Step 4:
Add DSPLib

Step 5: X86SIM
Compile &
Simulate

Step 6: AIESIM
Compile &
Simulate

Step 7:
Analyze the
Reports

IDE_DEMO

aie_ssr_lt1 [AI Engine]

Settings

vitis-comp.json

CMakeLists.txt

launch.json

alecompiler.cfg

Includes

Sources

data

fir1_app.cpp

fir1_coeff.h

fir1_graph.h

Output

FLOW

Component

aie_ssr_lt1

X86 SIMULATION

AIE SIMULATOR / HARDWARE

Build

REPORTS

Run

Debug

REPORTS

Summary

Graph

Array

Log

Simulator Output

Trace

Performance Metrics

Graph X

Subgraph View

Input: PLIO_fir_i (data/sig_i.txt)

buf0
buf0d

m_firKernels[0]

buf3
buf3d

_dma[0]

filter

fir_dut

buf4d
buf4

m_fftKernels[0]

fftMain

FFTwinproc

fft_dut

buf2
buf2d

Output: PLIO_fft_o (data/fft_o.txt)

Output: PLIO_fir_o (data/fir_o.txt)

aie_dut

Lock Stalls

Memory Stalls

Kernels

I/O

Buffers

Ports

Nets

Tiles

Interface Channels

NAME	STALLED TILE	STALLED KERNEL	STALLED PORT	START (PS)	DURATION (PS)	PC	RELATED STALLS	BUFFER	TYPE	LOCK HOLDER
LS_1	Tile [24,0]	_main	InWindow	589600	10410400	672		buf4	Read	
LS_2	Tile [25,0]	_main	InWindow	593600	3116800	624		buf0	Read	Input: PLIO_fir_i
LS_1953	Tile [24,0]	_main	InWindow	14630400	484800	672		buf4d	Read	out[0]
LS_3061	Tile [24,0]	_main	InWindow	18741600	437600	672		buf4	Read	out[0]
LS_3803	Tile [24,0]	_main	InWindow	22800800	391200	672		buf4d	Read	out[0]

Vitis Analyzer
Reports

AMD
together we advance_

56

Vitis Unified IDE – AIE Development Flow

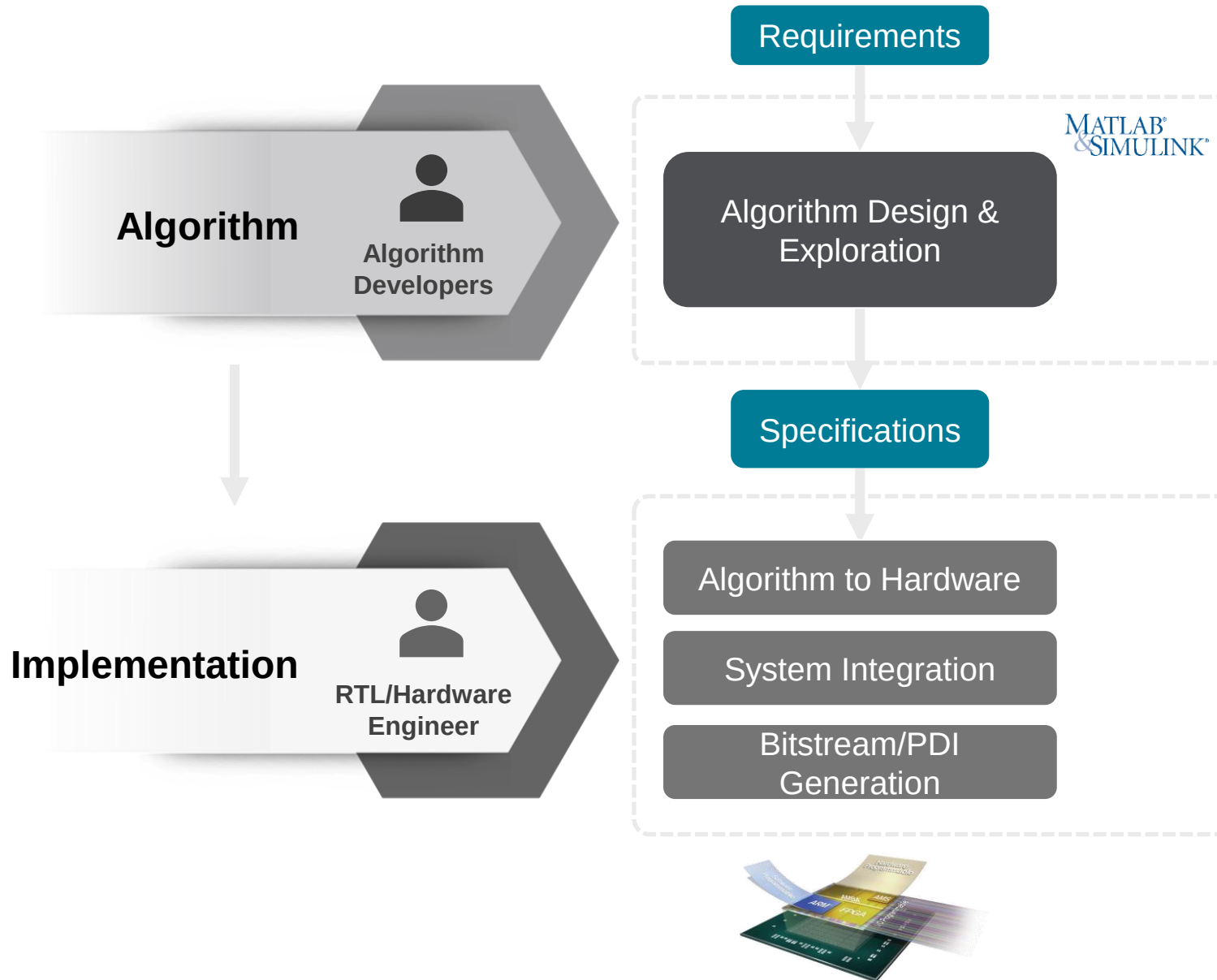
The screenshot displays the Vitis Unified IDE interface for AIE development. The left sidebar shows the project structure for 'IDE_DEMO', including settings, sources, and output. The central pane shows a hierarchical view of the design, with the 'Trace' option selected under the 'REPORTS' section. The right pane shows a timeline trace of the design execution, with various components and their states over time. A blue callout box points to the 'Trace' option in the 'REPORTS' section.

Vitis™ Analyzer - Trace

Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - Design 1: FIR/FFT Design for $SSR < 1$
 - Design 2: FIR/FFT Design for $SSR > 1$
 - AI Engine Design Using the Vitis™ IDE
 - **AI Engine Design Using Vitis Model Composer**
 3. Summary

Algorithm to Implementation: Challenges



Challenges

- Re-implement algorithm for efficient mapping to architecture
- Hardware-specific decisions to be made
- Simulation performance becomes a bottleneck
- Verifying equivalence between algorithm and hardware
- Difficult to debug issues with hardware implementation

Vitis™ Model Composer

Provides better integration from
algorithm to implementation



Vitis™ Model Composer – Toolbox



AI Engine Library



HDL Library
(previously part of
System Generator)

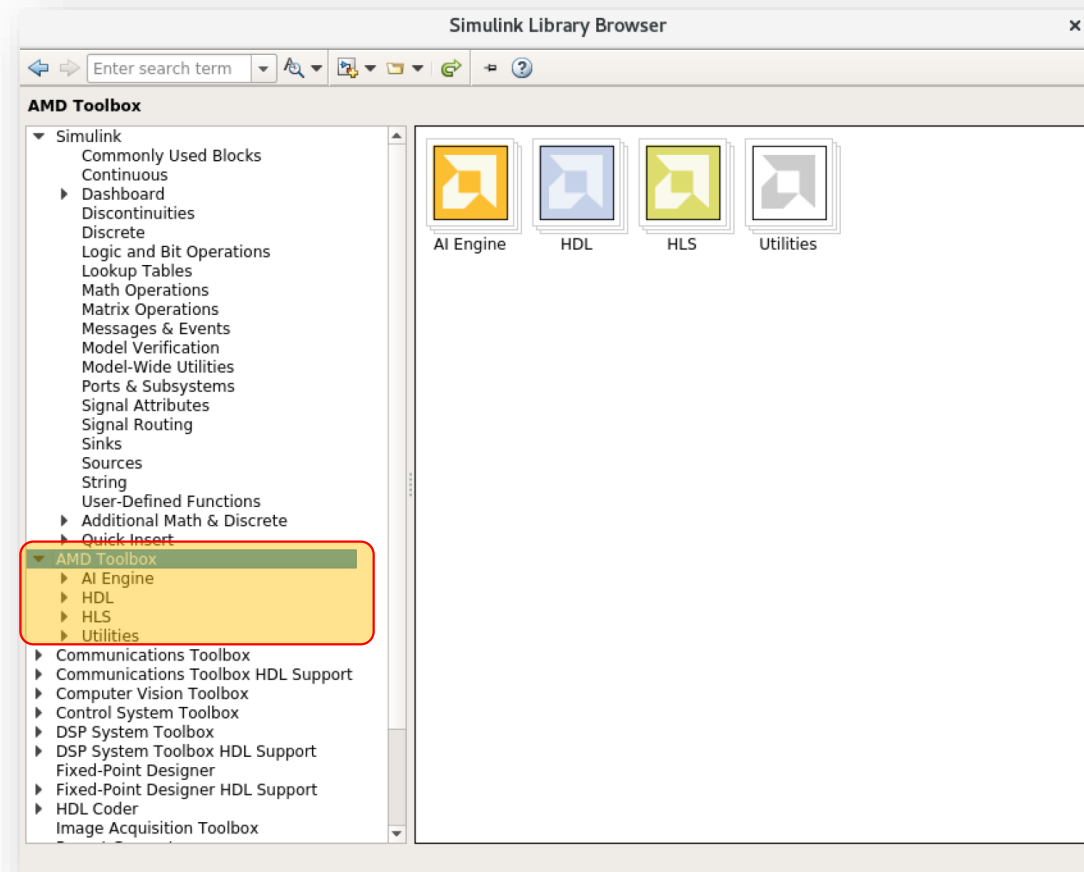


HLS Library

Highly optimized blocks
targeting:

- Programmable logic
- AIE Engines

HDL library contains all the library blocks previously
contained in the System Generator blockset



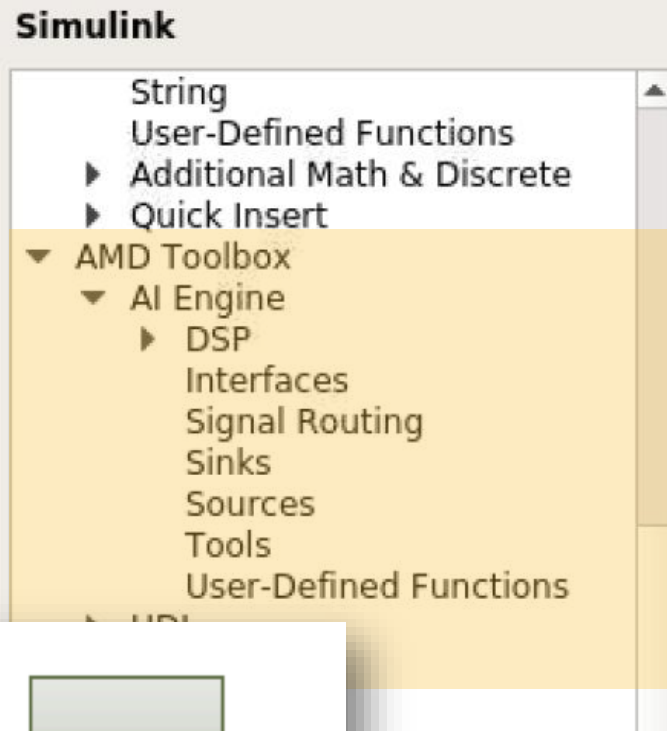
Design 1: FIR/FFT Design Example Using Windows (SSR<1) in Vitis Model Composer

DSP functions in sync with Vitis™ libraries

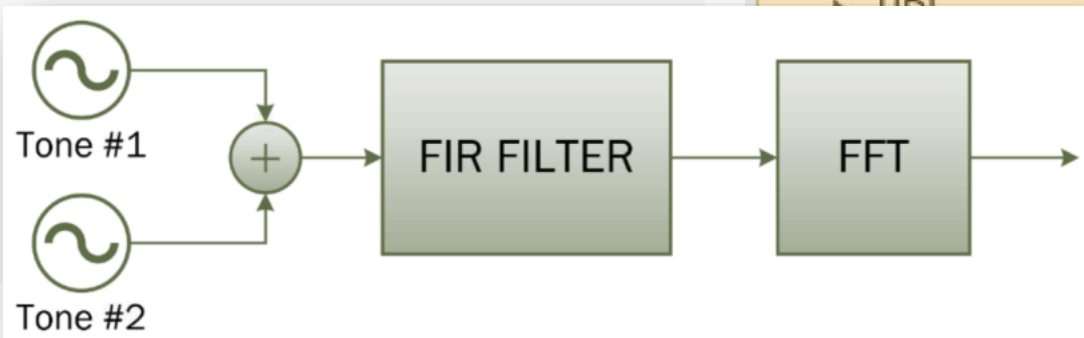
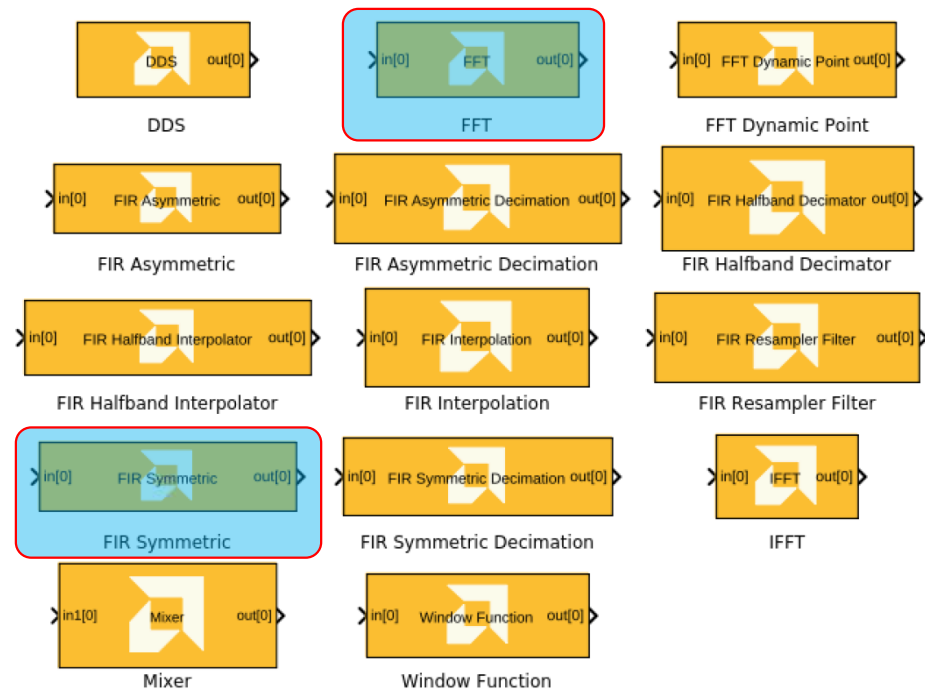
Add the blocks:

- FIR Symmetric
- FFT

Simulink Browser



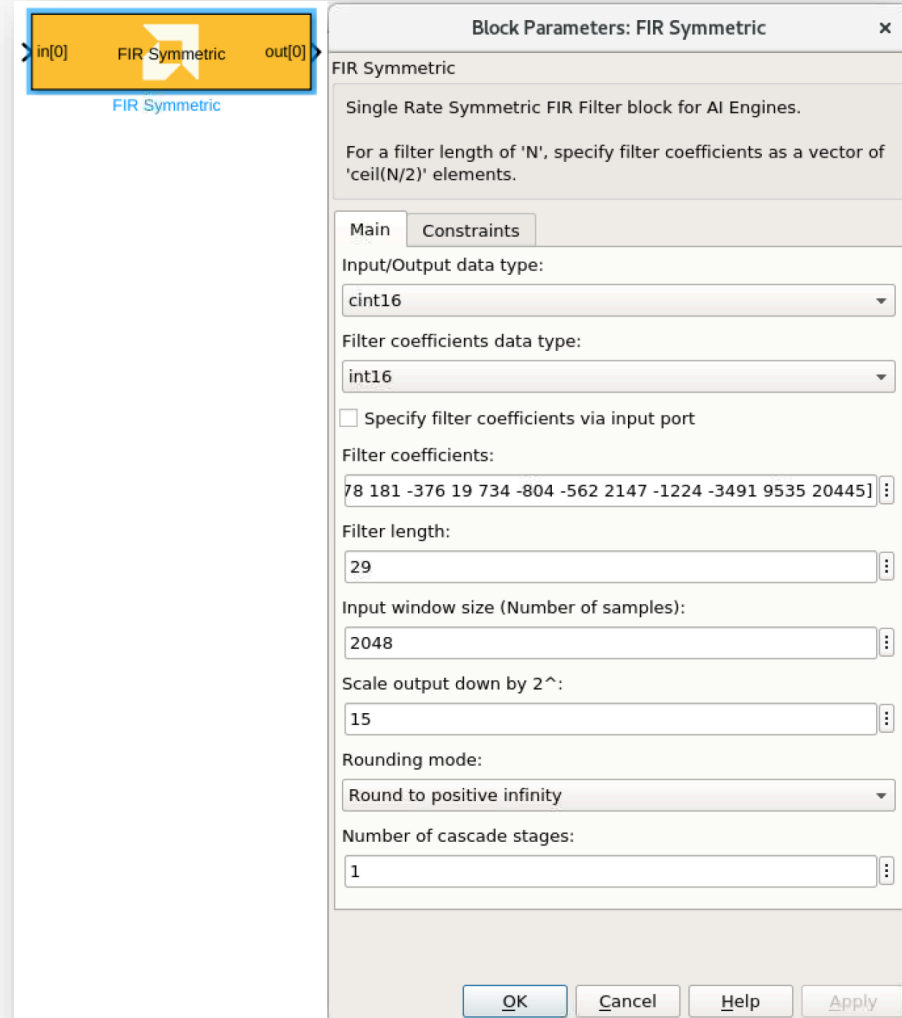
AI Engine > DSP > Buffer IO Blocks



Configure the Parameters – FIR Block

FIR Parameters

TP_FIR_LEN	29	Total number of taps
TP_SHIFT	15	Number of bits to shift accumulation down by before output
TP_RND	5	Symmetrical to zero
TP_INPUT_WINDOW_VSIZE	2048	Window size
TP_CASC_LEN	1	Number of cascaded kernels to use for this FIR
TP_DUAL_IP	0	Use dual inputs if set as 1
TP_USE_COEFF_RELOAD	0	No reload
TP_NUM_OUTPUTS	1	Number of ports to broadcast the output
TP_API	0	Set to windows mode
TP_SSR	1	Scale throughput using parallelization



Configure the Parameters – FFT Block

FFT Parameters		
TP_POINT_SIZE	256	Number of samples in a frame to be processed
TP_FFT_NIFFT	1	Performs FFT (1)
TP_SHIFT	6	Excludes twiddle shift
TP_CASC_LEN	1	Number of kernels to use
TP_DYN_PT_SIZE	0	Not to use runtime point size determine
TP_WINDOW_SIZE	2048	Number of samples in the input window
TP_API	0	Set to windows mode
TP_PARALLEL_POWER	0	Selects the parallelism factor as a power of 2 - 0 to 4 (1 to 16 kernel -lanes of processing)

in[0]

FFT

out[0]

FFT

Block Parameters: FFT

FFT

FFT block targeted for AI Engines.

Main

Constraints

Input/Output data type:

cint16

FFT Size:

256

Input window size (Number of Samples):

2048

Scale output down by 2[^]:

6

Number of cascade stages:

1

?

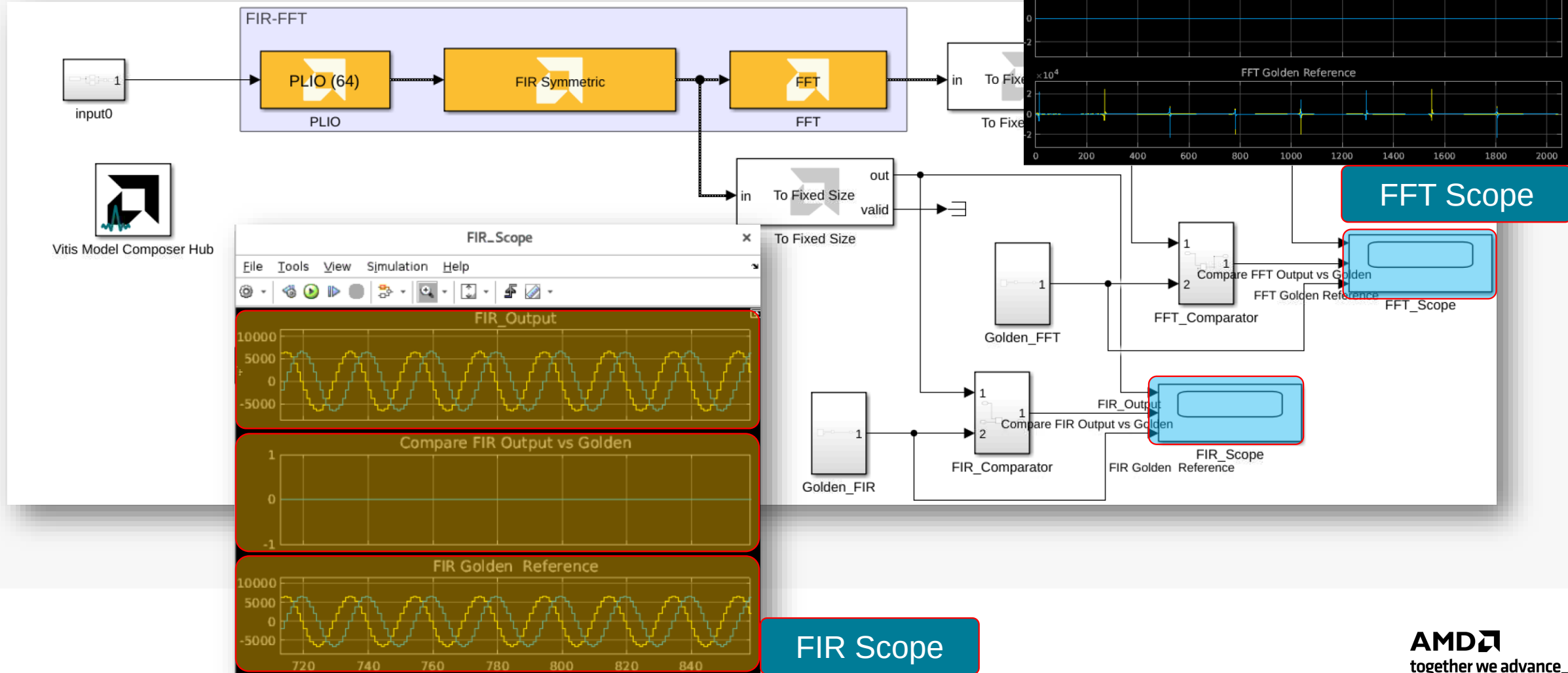
OK

Cancel

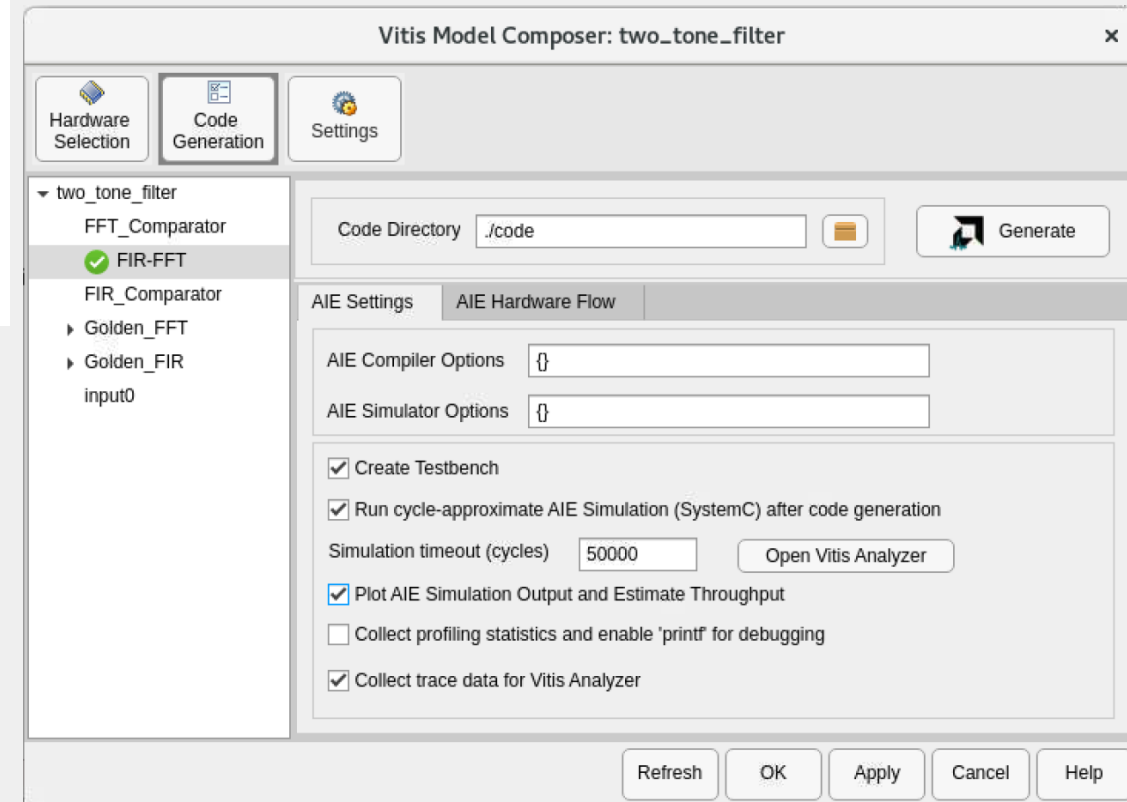
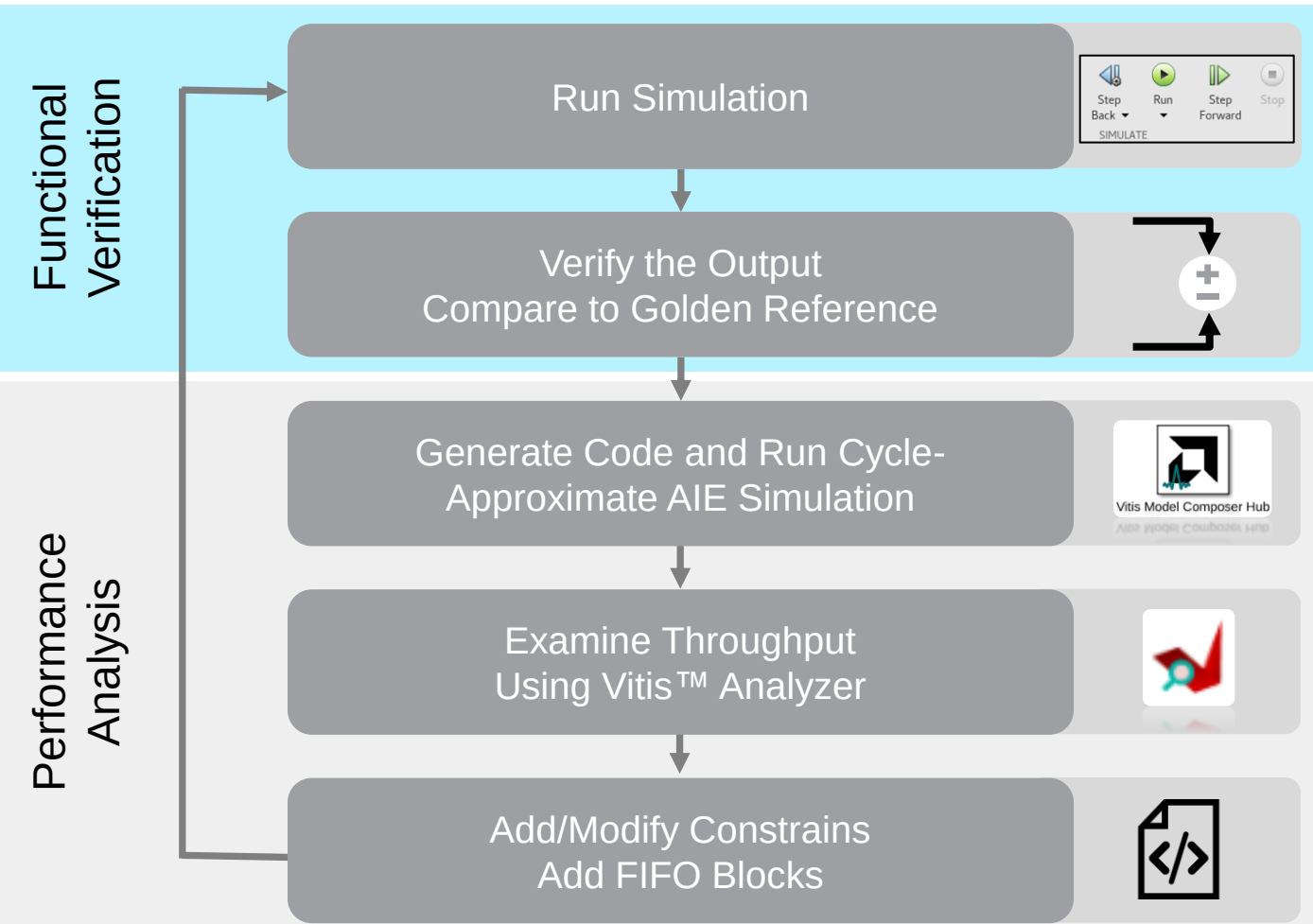
Help

Apply

AMD
together we advance_



AI Engine Kernel Development Flow Using Vitis Model Composer



Agenda

-
1. Overview of the Vitis™ DSP Library
 2. Demo Design Overview
 - MATLAB® Model of a FIR/FFT Design
 - Design 1: FIR/FFT Design for $SSR < 1$
 - Design 2: FIR/FFT Design for $SSR > 1$
 - AI Engine Design Using the Vitis™ IDE
 - AI Engine Design Using Vitis Model Composer
 3. **Summary**

Summary

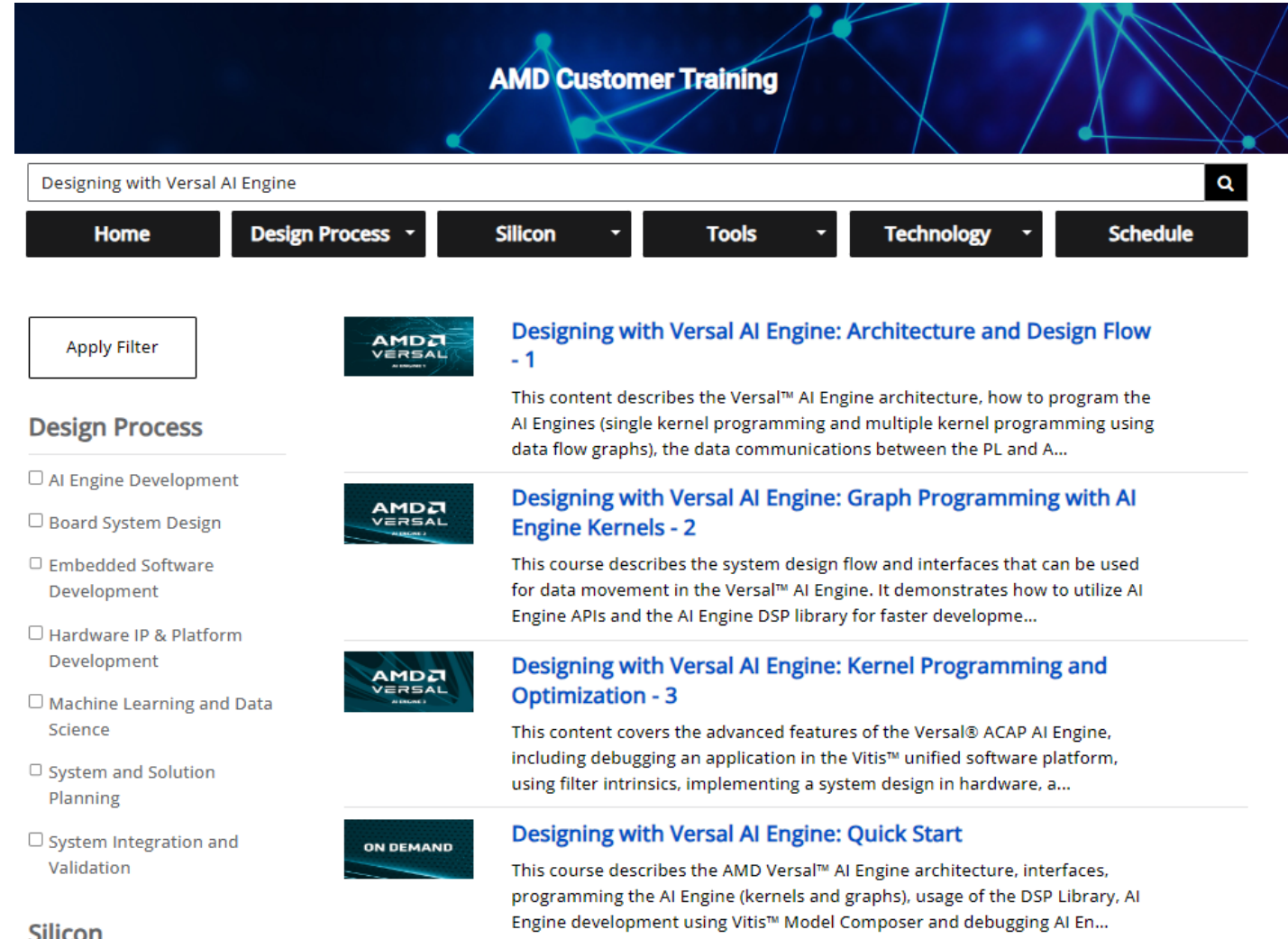
- DSPLib is a configurable library of kernels that can be used to develop applications on Versal™ adaptive SoC AI Engines
 - Distributed as free open-source code
- Download Vitis™ library from GitHub
 - https://github.com/Xilinx/Vitis_Libraries/tree/master/dsp
- Recommended entry point for all libraries is an L2 graph for your design
- Designing using FIRs/FFTs has been demonstrated using the tool flows:
 - Vitis IDE
 - Vitis Model Composer
 - Automatic code generation by the tool

Next Steps – Documentation

- Tool installation: Get the latest Vitis™ tool release
 - <https://www.xilinx.com/support/download.html>
- User guides
 - AI Engine Tools and Flows User Guide ([UG1076](#))
 - AI Engine Kernel and Graph Programming Guide ([UG1079](#))
 - Vitis Model Composer User Guide ([UG1483](#))
 - Supported MATLAB® tool versions and operating systems ([UG1483](#))
 - Vitis Libraries User Guide
 - https://docs.xilinx.com/r/en-US/Vitis_Libraries/dsp/index.html
 - AI Engine API User Guide
 - https://www.xilinx.com/htmldocs/xilinx2023_1/aiengine_api/aie_api/doc/index.html
- ...

Next Steps – Training

- Tutorials
 - Visit [GitHub](#) for more Vitis™ environment flow examples
 - https://github.com/Xilinx/Vitis-Tutorials/tree/2023.1/AI_Engine_Development
 - https://github.com/Xilinx/Vitis_Model_Composer
- On-Demand training:
 - <https://xilinxprod-catalog.netexam.com/Search?searchText=Designing+with+Versal+AI+Engine>



The screenshot shows the AMD Customer Training website. At the top is a dark blue header with the text "AMD Customer Training" and a network diagram. Below the header is a search bar containing the text "Designing with Versal AI Engine". Under the search bar is a navigation menu with buttons: Home, Design Process (with a dropdown arrow), Silicon (with a dropdown arrow), Tools (with a dropdown arrow), Technology (with a dropdown arrow), and Schedule. Below the navigation menu is a section titled "Design Process" with a list of checkboxes: AI Engine Development, Board System Design, Embedded Software Development, Hardware IP & Platform Development, Machine Learning and Data Science, System and Solution Planning, and System Integration and Validation. To the right of the "Design Process" list are four training modules, each with an AMD Versal logo icon and a title: "Designing with Versal AI Engine: Architecture and Design Flow - 1", "Designing with Versal AI Engine: Graph Programming with AI Engine Kernels - 2", "Designing with Versal AI Engine: Kernel Programming and Optimization - 3", and "Designing with Versal AI Engine: Quick Start". Each module has a brief description of its content.

Designing with Versal AI Engine: Architecture and Design Flow - 1

This content describes the Versal™ AI Engine architecture, how to program the AI Engines (single kernel programming and multiple kernel programming using data flow graphs), the data communications between the PL and A...

Designing with Versal AI Engine: Graph Programming with AI Engine Kernels - 2

This course describes the system design flow and interfaces that can be used for data movement in the Versal™ AI Engine. It demonstrates how to utilize AI Engine APIs and the AI Engine DSP library for faster developme...

Designing with Versal AI Engine: Kernel Programming and Optimization - 3

This content covers the advanced features of the Versal® ACAP AI Engine, including debugging an application in the Vitis™ unified software platform, using filter intrinsics, implementing a system design in hardware, a...

Designing with Versal AI Engine: Quick Start

This course describes the AMD Versal™ AI Engine architecture, interfaces, programming the AI Engine (kernels and graphs), usage of the DSP Library, AI Engine development using Vitis™ Model Composer and debugging AI En...

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, and combinations thereof are trademarks of Advanced Micro Devices, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

