AMD Versal™ AI Engines for DSP End-to-End Design Flow

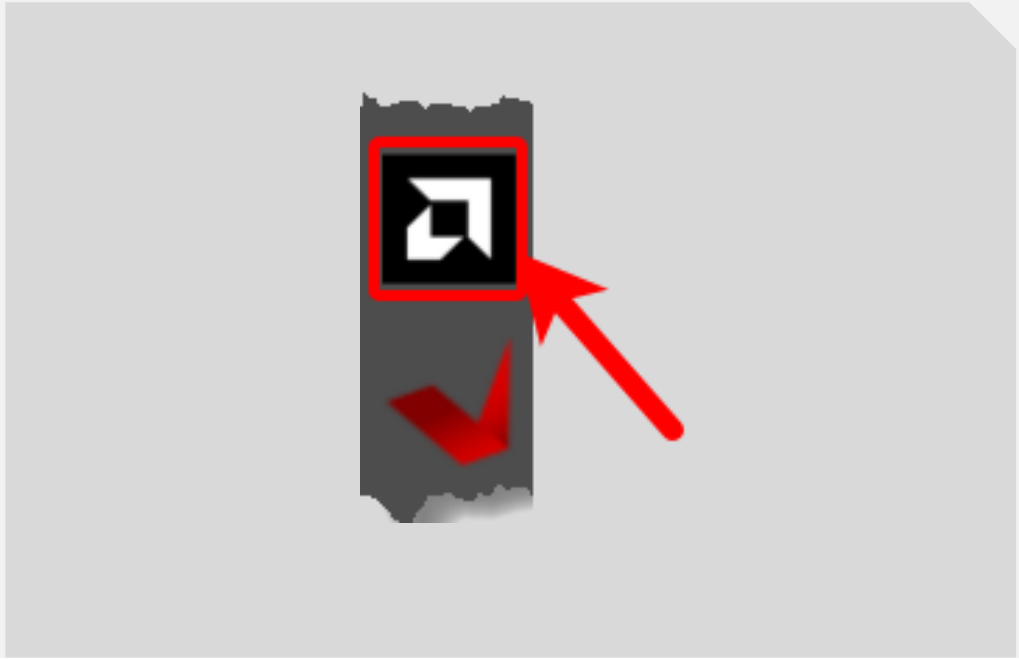
AMD
together we advance_

Agenda

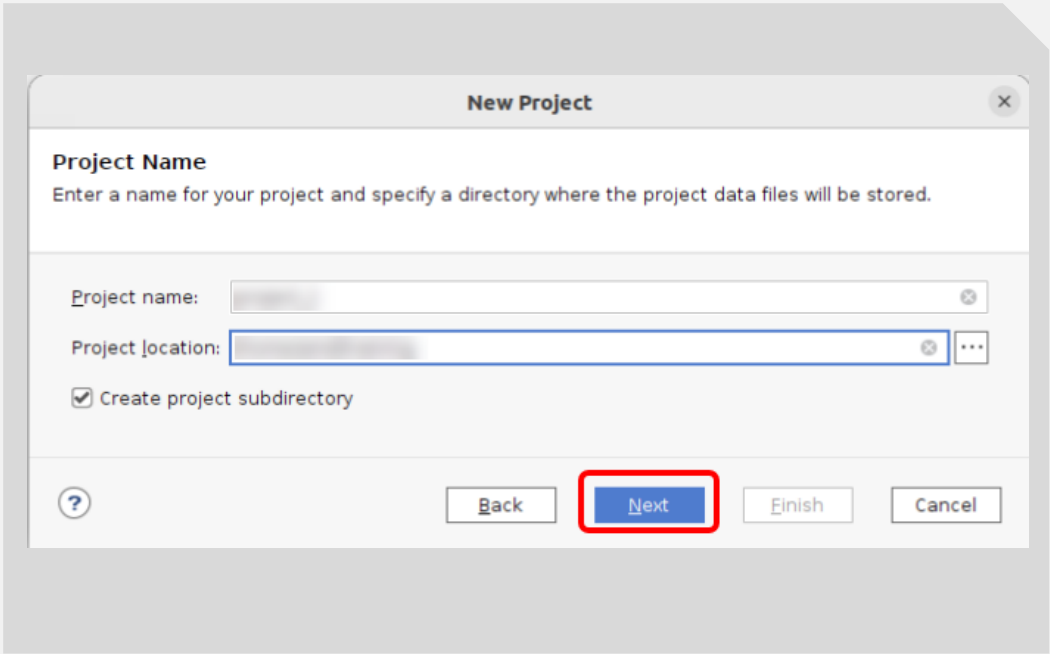
1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

Creating a Vivado Extensible Platform

- Click the Vivado Design Suite icon ( ) from the taskbar

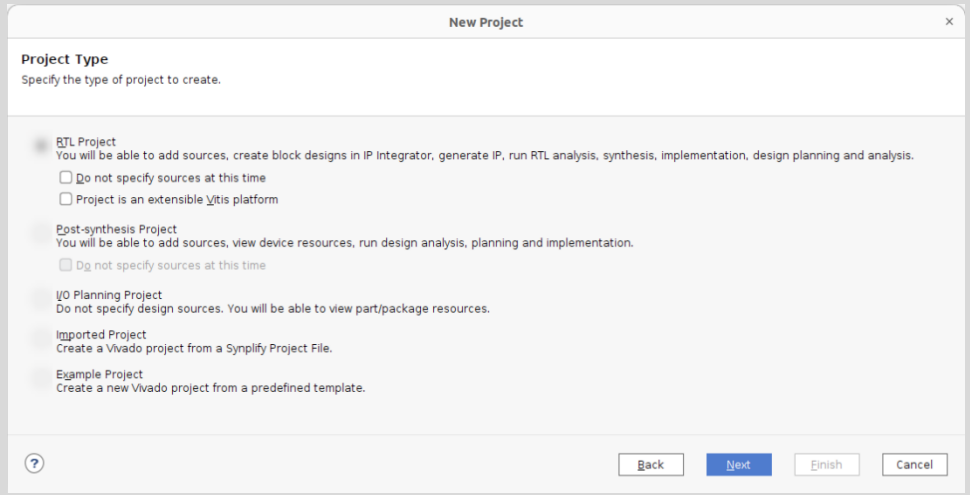


- Click **Create Project** and click Next on New Project window, enter project details and click on Next

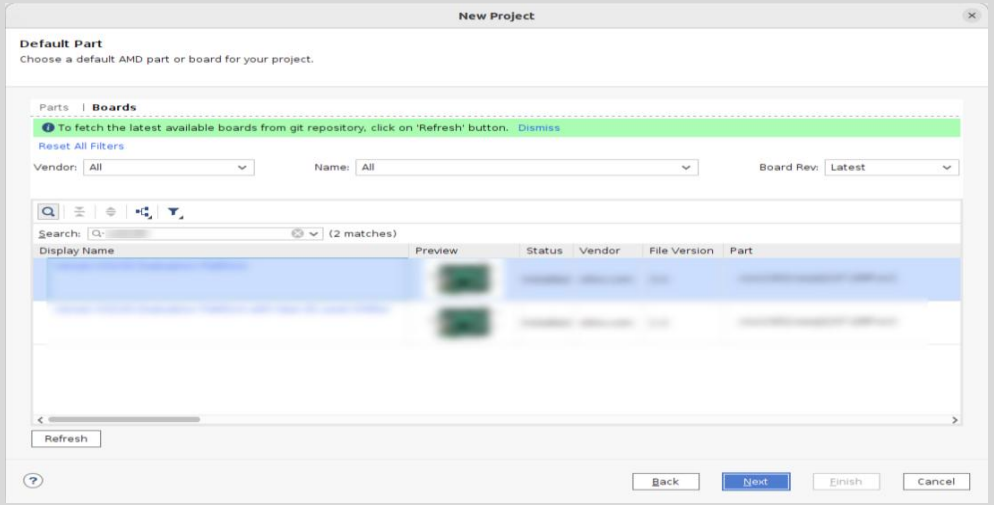


Creating a Vivado Extensible Platform

- Select the required **Project Type** and click on Next
- Now Add Sources and Constraints if required



- Choose a board required for the project and click Next
- Review the project summary and click on Finish

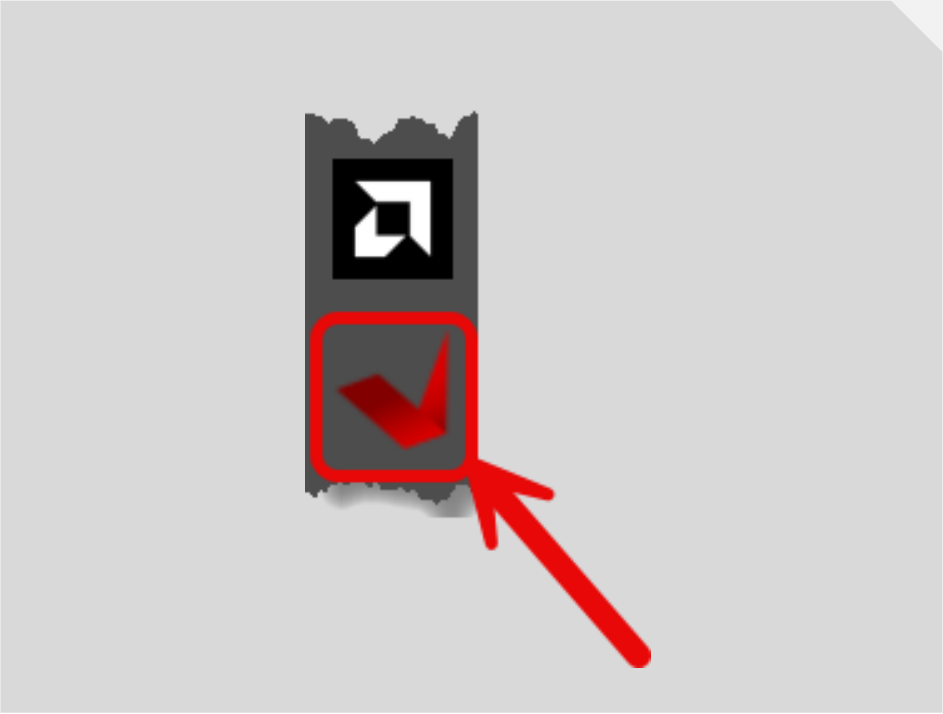


Agenda

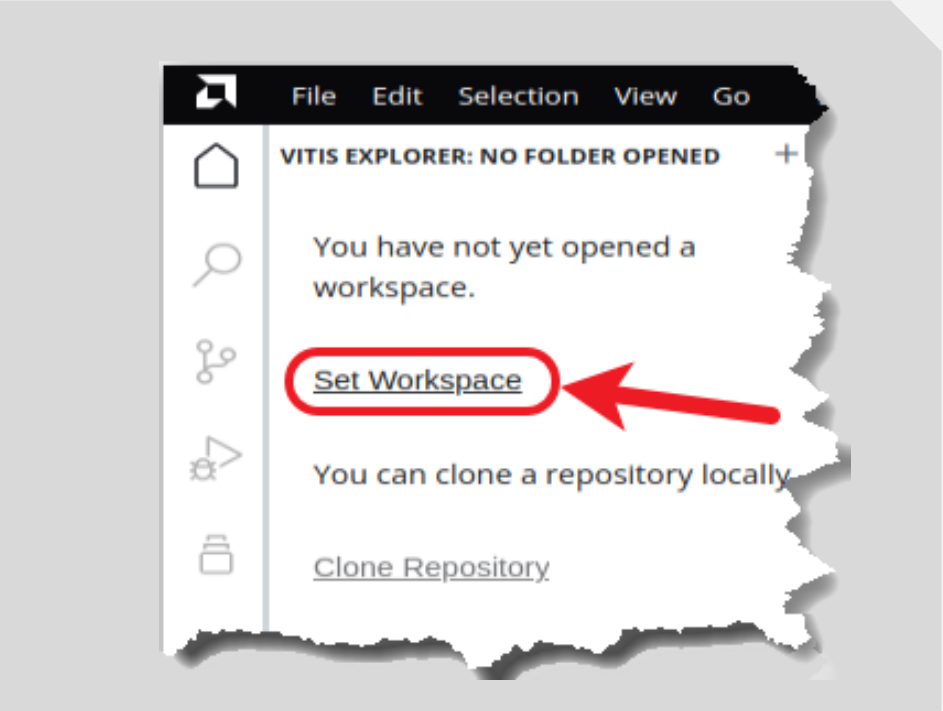
1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

Creating an AMD Vitis Unified IDE Project

- Click the Vitis™ Unifed IDE icon ( ) from the taskbar



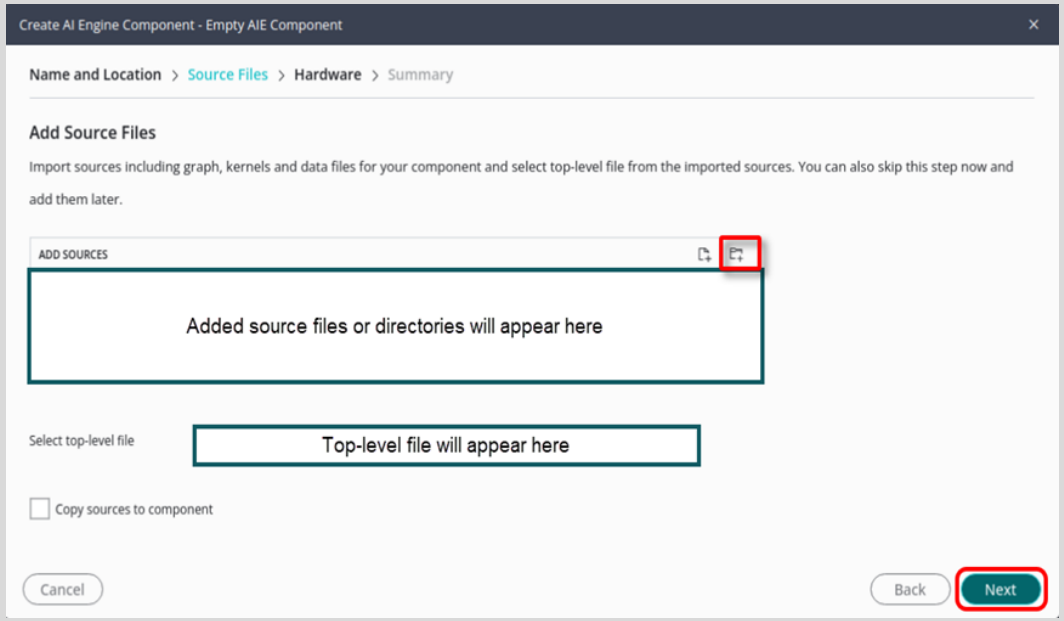
- Click **Set Workspace** to choose a workspace directory
- Click Open to proceed



Creating an AI Engine Component

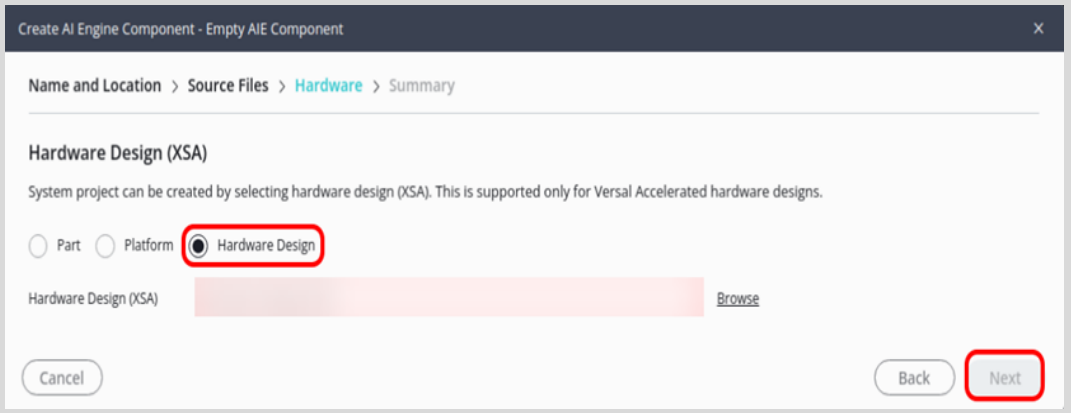
- File > New Component > AI Engine
- Component name (e.g., filter_design_acc)
- Choose Component location (default is workspace)

- Click Add Folders under Import Sources
- Select data and src directories

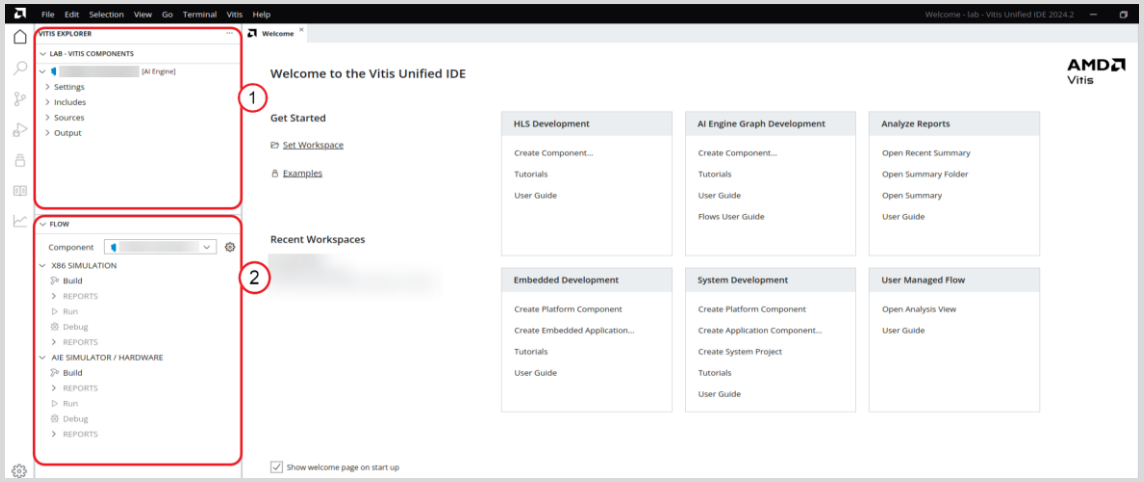


Creating an AI Engine Component

- Choose the required XSA
- Click Next



- Review Summary
- Click Finish
- Component appears in Components Window



Agenda

1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

Reviewing the Source Files

- Open graph code
  - Path: *[AI Engine] > Sources > src > fir1_graph.h*

- For the FIR filter, the following parameters are declared:
  - TT_DATA: Data type
  - TT_COEFF: Coefficient type
  - TP_FIR_LEN: Length of the filter (overall length including zeros)
  - TP_SHIFT: Shifting value operated just before sending the data to the output
  - TP_RND: Truncation, rounding, rounding up, etc.
  - TP_INPUT_WINDOW_VSIZE: Size of the window (buffer) in samples
  - TP_CASC_LEN: Number of cascaded kernels to use for the FIR
  - TP_DUAL_IP: Use dual inputs if set to 1
  - TP_USE_COEFF_RELOAD: No reload if set to 0
  - TP_NUM_OUTPUTS: Number of ports to broadcast the output
  - TP_API: Set for windows (I/O buffers) or stream data APIs
  - TP_SSR: Scale throughput using parallelization

```
#pragma once

#include <adf.h>
#include <fir_sr_sym_graph.hpp>
#include <fft_ifft_dit_1ch_graph.hpp>
#include "fir1_coeff.h"

using namespace adf;

using namespace xf::dsp::aie::fir::sr_sym;
using namespace xf::dsp::aie::fft::dit_1ch;

// ------------------------------------------------------------
// FIR Namespace
// ------------------------------------------------------------

namespace fir1 {
  static constexpr int  WIN_SIZE = 2048;
  typedef cint16                                TT_DATA;
  typedef int16                                 TT_COEFF;
  static constexpr int  TP_FIR_LEN            = [redacted]      // Total number of taps
  static constexpr int  TP_SHIFT              = [redacted]      // Depends on FXP properties
  static constexpr int  TP_RND                = [redacted]      // how to set this
  static constexpr int  TP_INPUT_WINDOW_VSIZE = [redacted];
  static constexpr int  TP_CASC_LEN           = [redacted]      // # of tiles in a cascade
  static constexpr int  TP_DUAL_IP            = [redacted]      // Only used in stream mode (TP_API = 1)
  static constexpr int  TP_USE_COEFF_RELOAD   = [redacted]      // Set to 1 to use coefficient reload feature
  static constexpr int  TP_NUM_OUTPUTS        = [redacted]      // Add optional 2nd output port for windows
  static constexpr int  TP_API                = [redacted]      // Set to 1 for stream mode (windows otherwise)
  static constexpr int  TP_SSR                = [redacted]      // Scale throughput using parallelization
};

// ------------------------------------------------------------
// FFT Namespace
// ------------------------------------------------------------

namespace fft1 {
  typedef cint16                                TT_TYPE;
  typedef cint16                                TT_TWIDDLE;
  static constexpr int  TP_POINT_SIZE         = [redacted];
  static constexpr int  TP_FFT_NIFFT          = [redacted]
  static constexpr int  TP_SHIFT              = [redacted]  // Excludes twiddle shift
  static constexpr int  TP_CASC_LEN           = [redacted]
  static constexpr int  TP_DYN_PT_SIZE        = [redacted]
  static constexpr int  TP_WINDOW_SIZE        = [redacted]
  static constexpr int  TP_API                = [redacted]  // Windows
  static constexpr int  TP_PARALLEL_POWER     = [redacted]
};
```

FIR filter parameters

FFT design parameters

Reviewing the Source Files

For the FFT, the following parameters are declared:

- TT_TYPE: Type of individual data samples input to and output from the transform function
- TT_TWIDDLE: Twiddle factors of the transform
- TP_POINT_SIZE: Number of samples processed by the FFT
- TP_FFT_NIFFT: To select the transform to perform (0 for IFFT and 1 for FFT)
- TP_SHIFT: Number of bits to shift accumulate down by before output
- TP_CASC_LEN: Number of bits to shift accumulate down by before output
- TP_DYN_PT_SIZE: Number of bits to shift accumulate down by before output
- TP_WINDOW_SIZE: Number of samples in the input window
- TP_API: Set for windows (I/O buffers) or stream data APIs
- TP_PARALLEL_POWER: Selects the parallelism factor as a power of 2. Values range from 0 to 4

```
#pragma once

#include <adf.h>
#include <fir_sr_sym_graph.hpp>
#include <fft_ifft_dit_1ch_graph.hpp>
#include "fir1_coeff.h"

using namespace adf;

using namespace xf::dsp::aie::fir::sr_sym;
using namespace xf::dsp::aie::fft::dit_1ch;

// ------------------------------------------------------------
// FIR Namespace
// ------------------------------------------------------------
namespace fir1 {
  static constexpr int  WIN_SIZE = 2048;
  typedef cint16                                TT_DATA;
  typedef int16                                 TT_COEFF;
  static constexpr int  TP_FIR_LEN           =      // Total number of taps
  static constexpr int  TP_SHIFT             =      // Depends on FXP properties
  static constexpr int  TP_RND               =      // how to set this
  static constexpr int  TP_INPUT_WINDOW_VSIZE =     ;
  static constexpr int  TP_CASC_LEN          =      // # of tiles in a cascade
  static constexpr int  TP_DUAL_IP           =      // Only used in stream mode (TP_API = 1)
  static constexpr int  TP_USE_COEFF_RELOAD  =      // Set to 1 to use coefficient reload feature
  static constexpr int  TP_NUM_OUTPUTS       =      // Add optional 2nd output port for windows
  static constexpr int  TP_API               =      // Set to 1 for stream mode (windows otherwise)
  static constexpr int  TP_SSR               =      // Scale throughput using parallelization
};

// ------------------------------------------------------------
// FFT Namespace
// ------------------------------------------------------------
namespace fft1 {
  typedef cint16                                TT_TYPE;
  typedef cint16                                TT_TWIDDLE;
  static constexpr int  TP_POINT_SIZE        =     ;
  static constexpr int  TP_FFT_NIFFT         =
  static constexpr int  TP_SHIFT             =    // Excludes twiddle 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               =    // Windows
  static constexpr int  TP_PARALLEL_POWER    =
};
```

FIR filter parameters

FFT design parameters

Reviewing the Source Files

- Review the other components
- Update the kernel connections in fir1_graph.h as per your design
  - filt_i.out [0] > fir_dut.in [0]
  - fir_dut.out [0] > fft_dut.in [0]
  - fir_dut.out [0] > filt_o.in [0]
  - fft_dut.out [0] > fft_o.in [0]

```
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  filt_i; // top level input
  output_plio filt_o; // top level output
  output_plio fft_o;  // fft output

  TT_FIR fir_dut; // for fir function
  TT_FFT fft_dut; // for fft function

  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<>(      );     //connect the top level input to fir
    connect<>(      );     //connect the fir output to fft input
    connect<>(      );     //connect the fir output to top level fir output
    connect<>(      );     //connect the fft output to top level fft output
  }
```

Kernel coefficient

FIR filter declaration

FFT declaration

Filter initialization with I/O data and graph connections

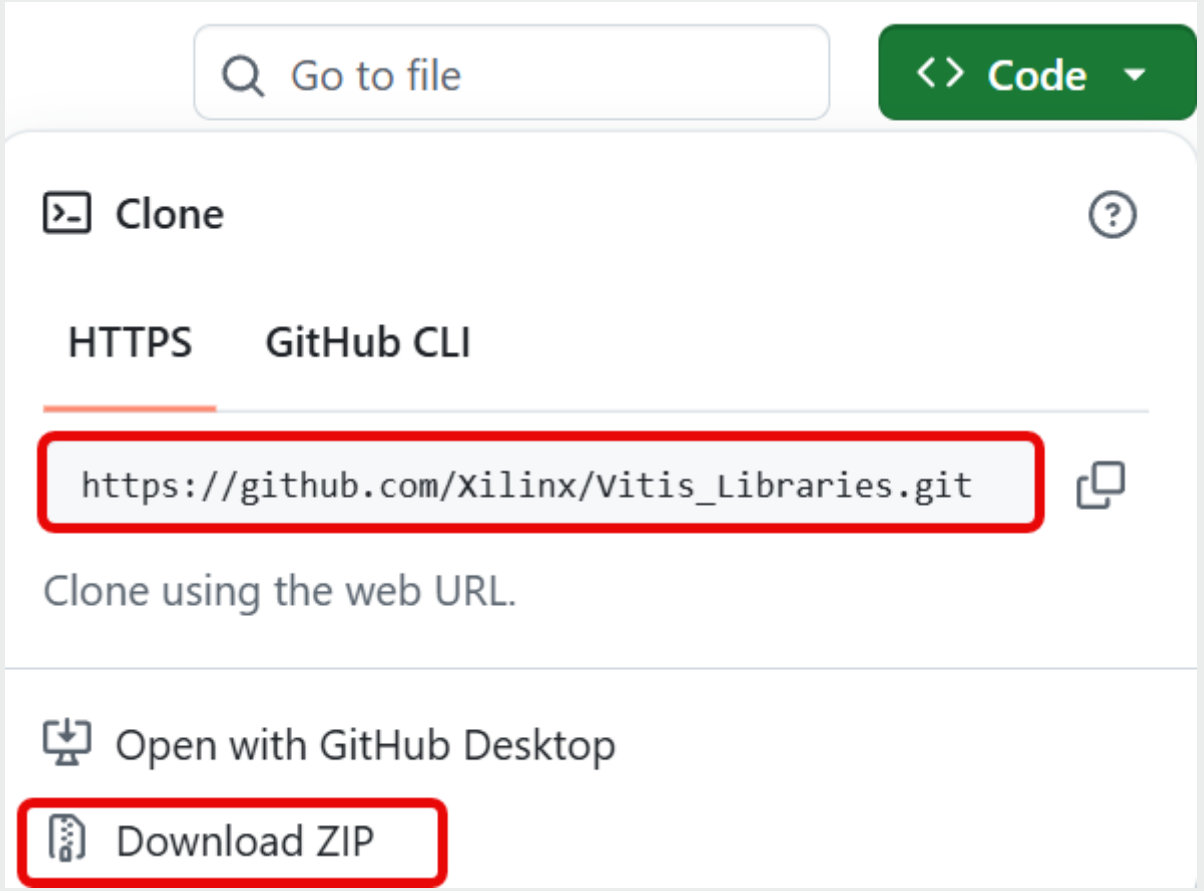
Agenda

1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

How to Download / Import Libraries from GitHub

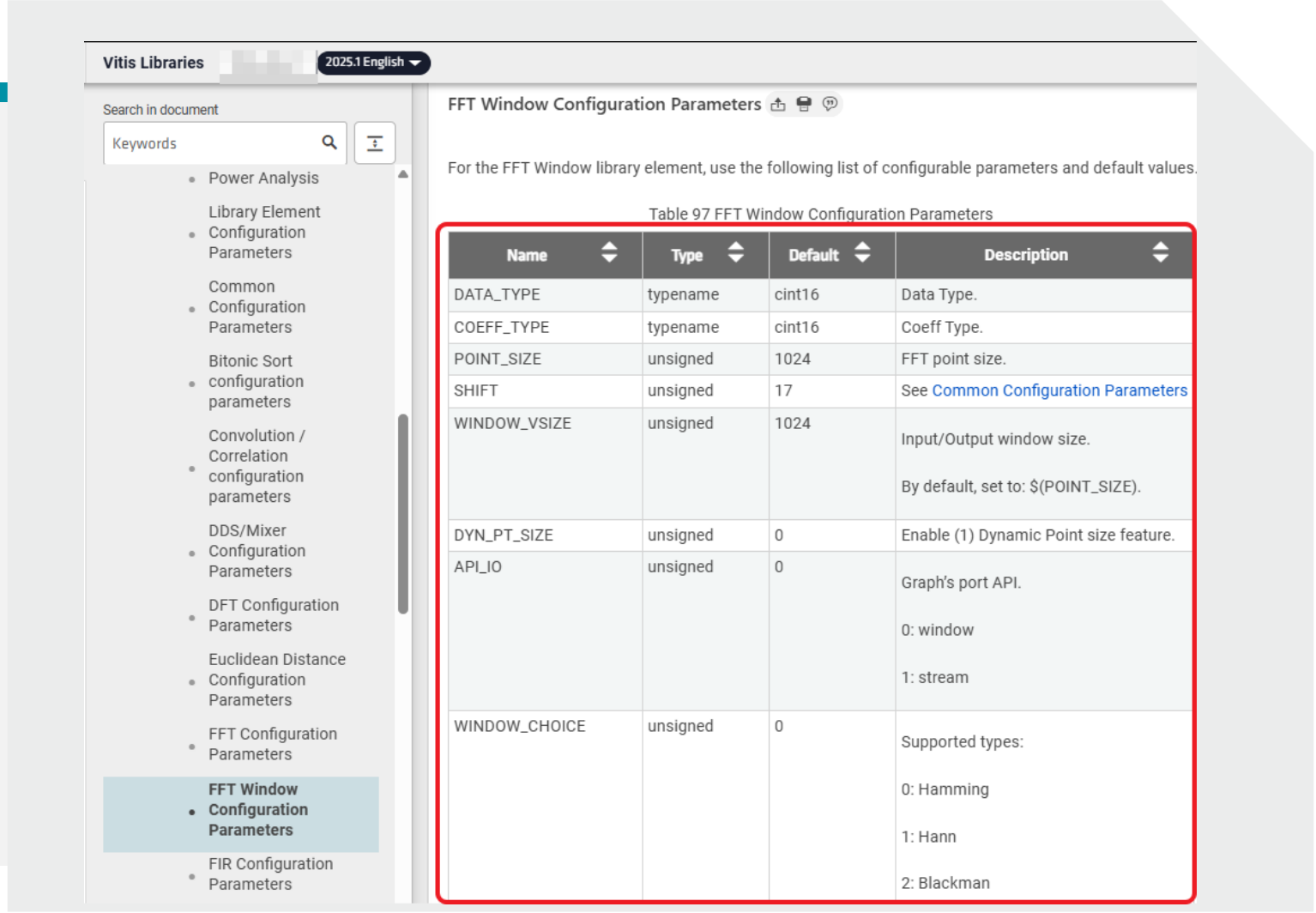
AMD Vitis™ DSP Library

https://github.com/Xilinx/Vitis_Libraries



Configuration Parameters

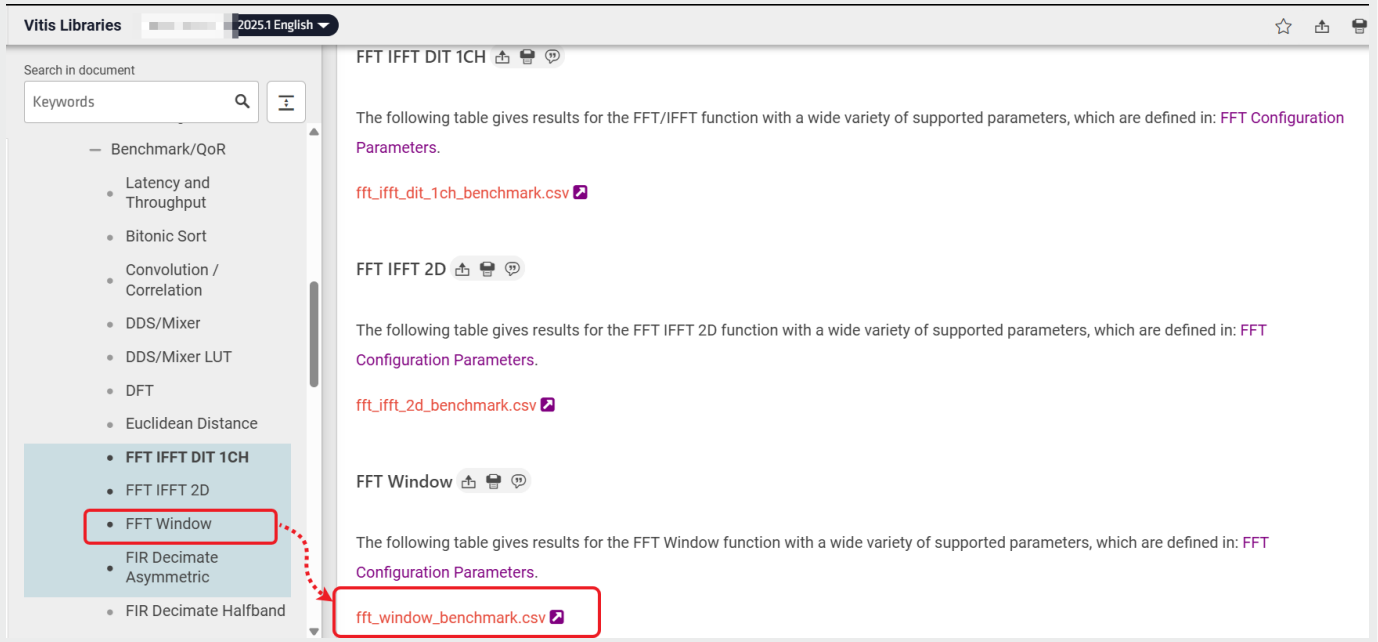
- Different set of parameters for different functions
- Available in AMD Vitis™ Libraries Configuration Parameters



Configuration Databases

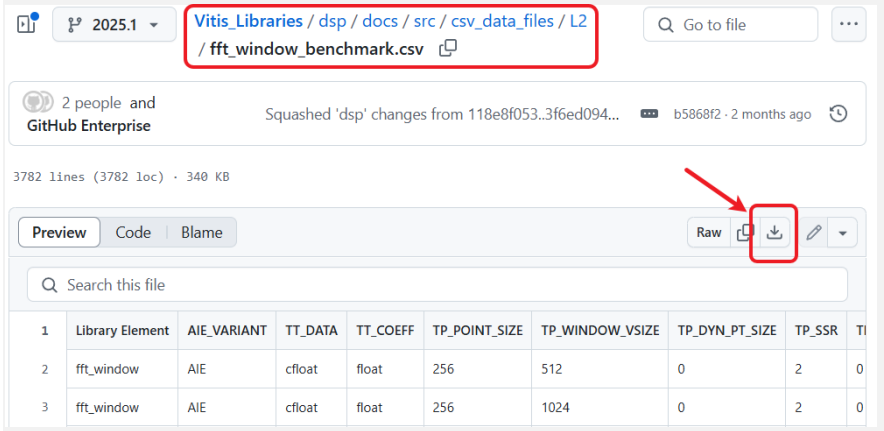
AMD Vitis™ DSP Library – Configuration Database

- Gives an early estimate of power, latency, throughput, and resource utilization
- Benchmarks are for different combination of Library parameters such as datatypes and AI Engine types
- Has ~3K to 5K test results
- Is accessed as a CSV file located at: Vitis_Libraries/dsp/docs/src/csv_data_files/L2 at 2025.1 · Xilinx/Vitis_Libraries · GitHub

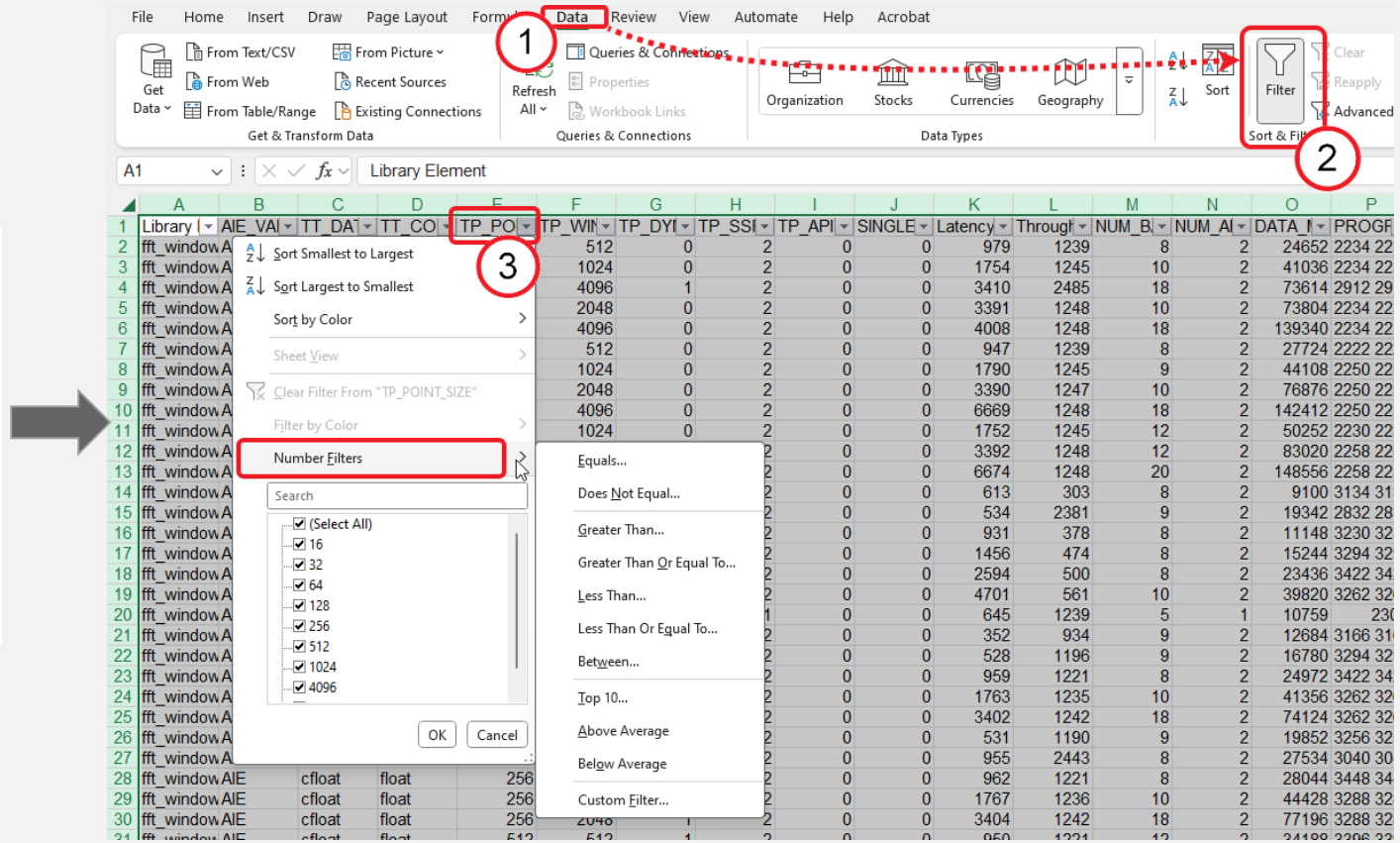


Filtering and Sorting CSV table

FFT Window Function – Filtering the values in the CSV table using the Excel sheet in Desktop



	Library Element	AIE_VARIANT	TT_DATA	TT_COEFF	TP_POINT_SIZE	TP_WINDOW_VSIZE	TP_DYN_PT_SIZE	TP_SSR	TI
2	fft_window	AIE	cfloat	float	256	512	0	2	0
3	fft_window	AIE	cfloat	float	256	1024	0	2	0

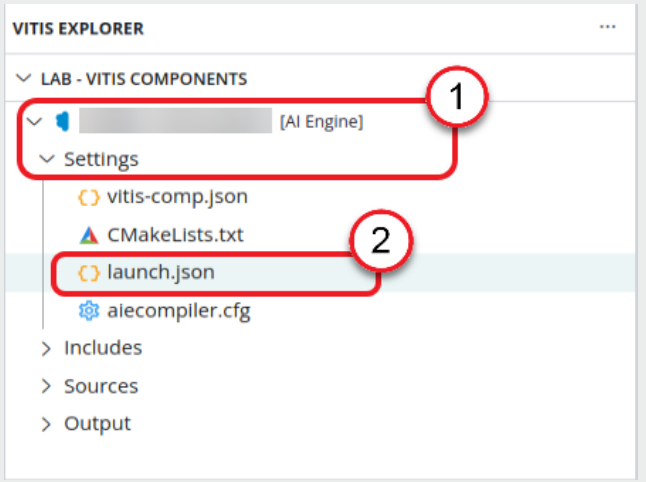


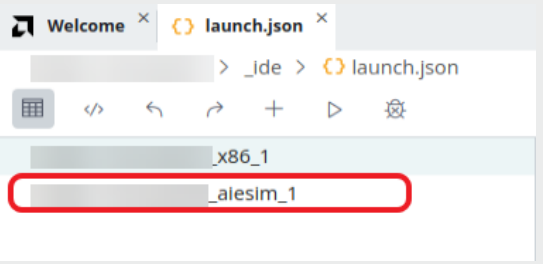
Agenda

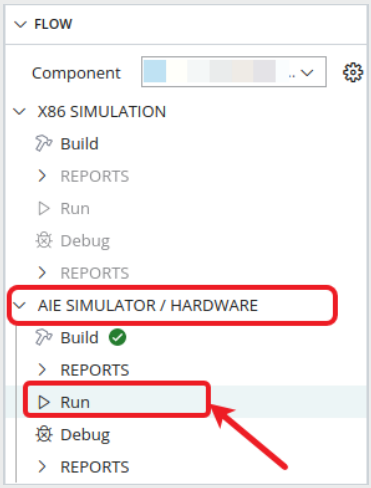
1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

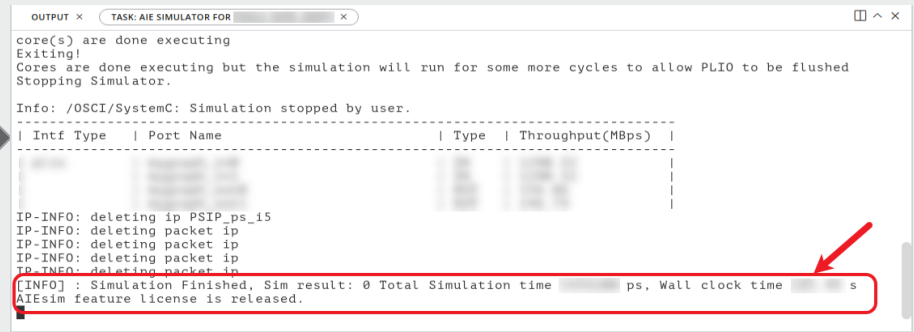
Building and Simulating the Project

- Click Build under AIE Simulator/Hardware to build the component
- Add Launch Configuration
  - Path: Settings > launch.json
  - Select filter_design_acc_aiesim_1 config
- Click Run under AIE Simulator/Hardware
- Monitor TASK window









Latency and Throughput Estimates

Average Throughput Computed

Displayed at End of AIE Simulation

VCD file generates latency and throughput estimates

```
aiesimulator –pkg-dir=./Work –dump-vcd foo –options-file=aiesim-options.txt
```

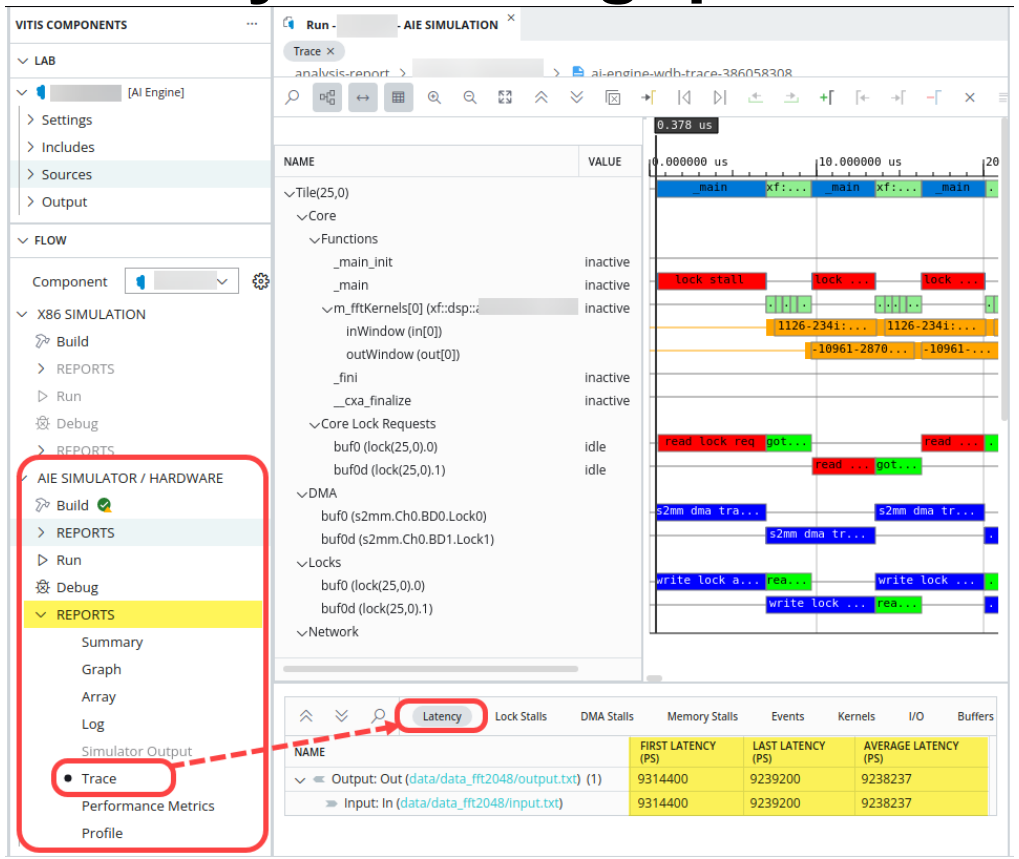
Continuous Latency Between specific input and output ports

Continuous Throughput At a specific input or output port

Vitis™ Unified IDE > AIE Simulation Reports > Trace > Opens Analysis View

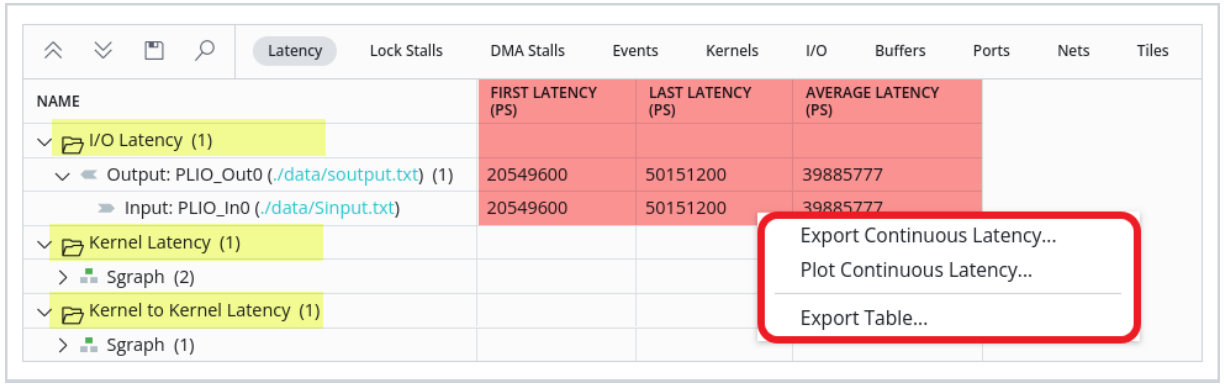
Command line: `vitis_analyzer aie/aiesimulator_output/default.aierun_summary`

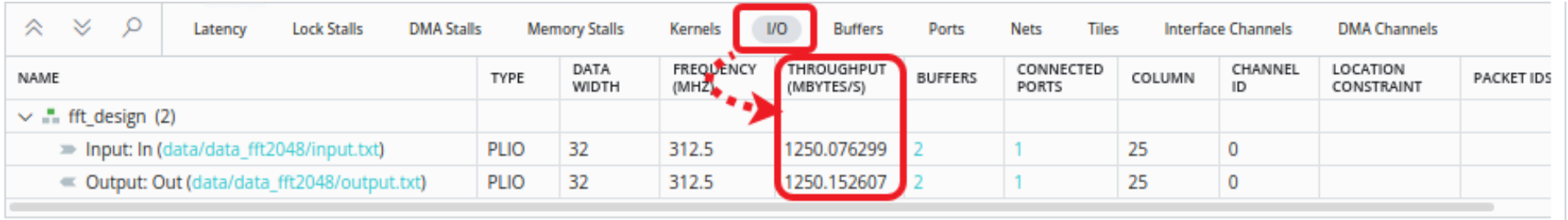
Latency & Throughput Table



First Latency	Last Latency	Average Latency
Latency time between first input to first output	Latency time between last input to last output	Difference between avg. output sample time and avg. input sample time

Plot or Export Continuous Latency and Throughput





Agenda

1. Creating a Vivado Extensible Platform
2. Creating an AI Engine Component in Vitis™ Unified IDE
3. Reviewing the Source Files
4. Configuring DSP Library Parameters Using .csv File
5. Running AI Engine Compiler and Simulator, Vitis Analyzer to Measure Latency and Throughput
6. Running Export to Vivado™ Tool Flow

AMD Vitis Export to Vivado Flow

Enables bi-directional hardware hand-offs between Vivado™ Design Suite and Vitis™ tools

`v++ -link --export_archive`

Create custom Vivado platform (Flat/BDC) using RTL, HLS, or IP catalog

Flow can be repeated for any number of design iterations

XSA file can be exported from Vivado IDE and passed back to Vitis tools

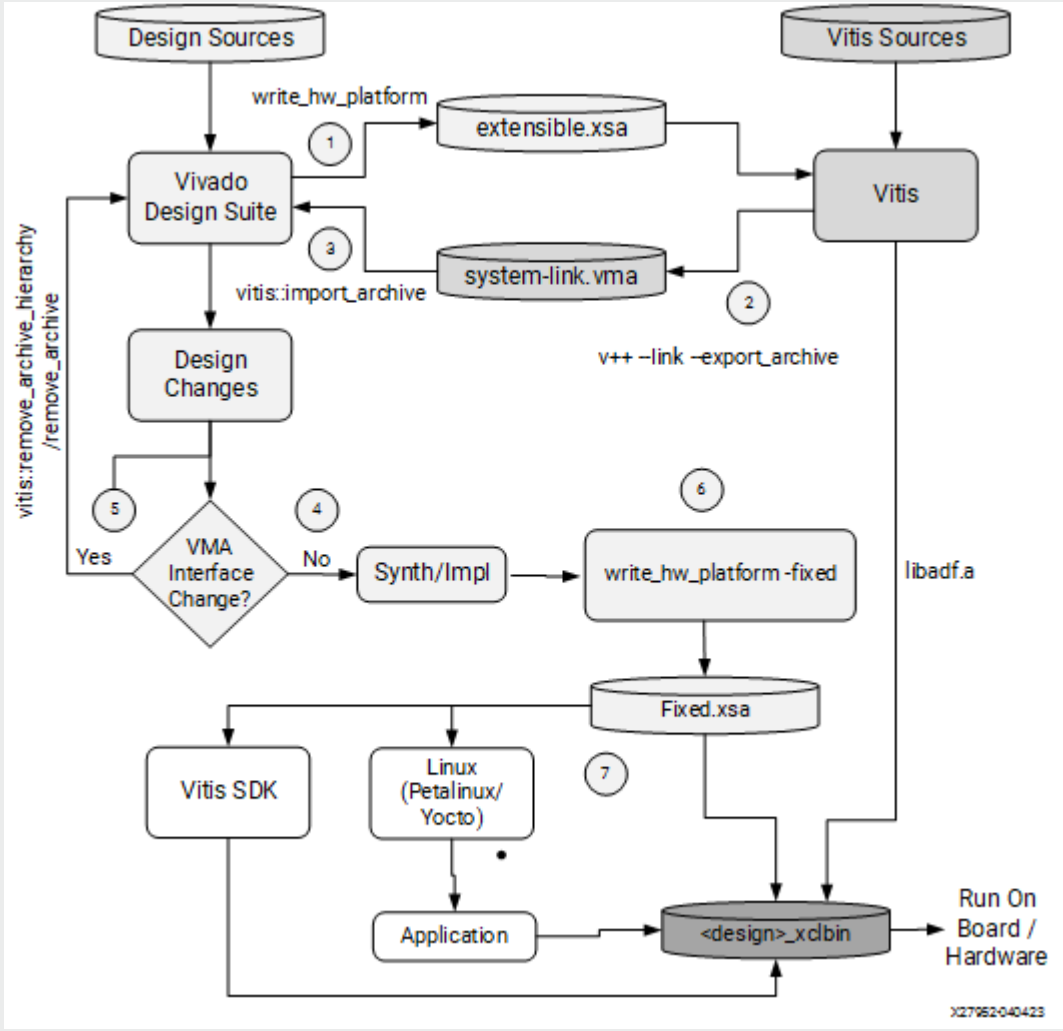
Updated VMA file can be reimported into Vivado IDE

v++ compiler operates on Vivado project that has been encapsulated in extensible XSA

Support for modifications to Vivado project that do not invalidate contract between imported design and XCLBIN

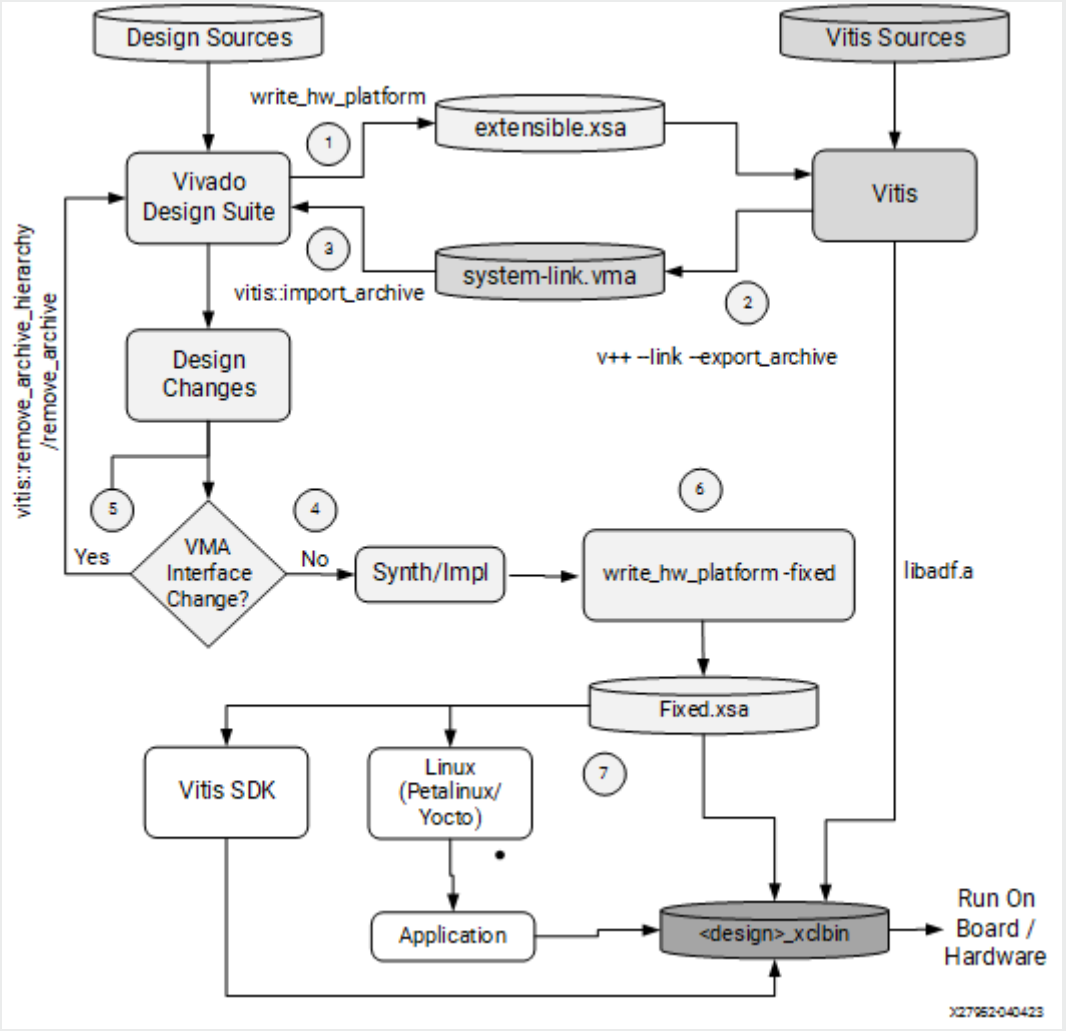
Vitis Export Flow Implementation

1. Import XSA in Vitis™ to compile and link:
   - •AI Engine graph (libadf.a)
   - •PL kernels (.xo)
   - •Update system.cfg, run Vitis linker
2. Export Vitis Metadata Archive (VMA)
   - **•v++ --link --export_archive --platform <xsa> --config system.cfg <xo> libadf.a -o <vma>.vma**



AMD Vitis Export Flow Implementation

3. Import VMA into Vivado™ tools:
   - **vitis::import_archive ./<vma>.vma**
   - Creates Vitis region block design (read-only Vitis hierarchy)
4. Modify design in Vivado as needed
   - For PL/AIE updates → remove VMA (**vitis::remove_archive**)
   - Re-export XSA and repeat Vitis™ tool flow if required
5. After implementation, generate fixed XSA:
   - **write_hw_platform -fixed ./<fixed_xsa>.xsa**
6. Use fixed XSA for:
   - Yocto™ / Vitis Embedded apps
   - Bare-metal or hardware validation
7. For emulation:
   - Generate sim-included XSA (include_sim_content)
8. Package and create deployable .xclbin:
   - **v++ --package -t <hw|hw_emu> --xsa <fixed_xsa>**



In this demo, we'll walk you through the complete end-to-end flow of developing a Vitis Subsystem, or VSS, and integrating it with a custom Vivado extensible platform — targeting the AMD Versal VCK190 device.

Watch the video on YouTube - AMD Versal™ AI Engines for DSP End-to-End Design Flow
https://youtu.be/J6NIZHCKqIg

Summary

1

Source AMD Vitis™ tools and configure Linux® sysroot

2

Use Makefiles to compile AI Engine, HLS, RTL, and validate functionality

3

Configure core blocks and export XSA for Vitis integration

4

Link components in Vitis, generate VMA, and import back into the Vivado™ for final design

General Disclaimer 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.


AMD