Polyphase Channelizer Design on AMD Versal™ AIE-ML using Vitis™ Libraries

Faisal El-Shabani – Technical Marketing Manager | AI Engines and DSP

AMD
together we advance_

Agenda

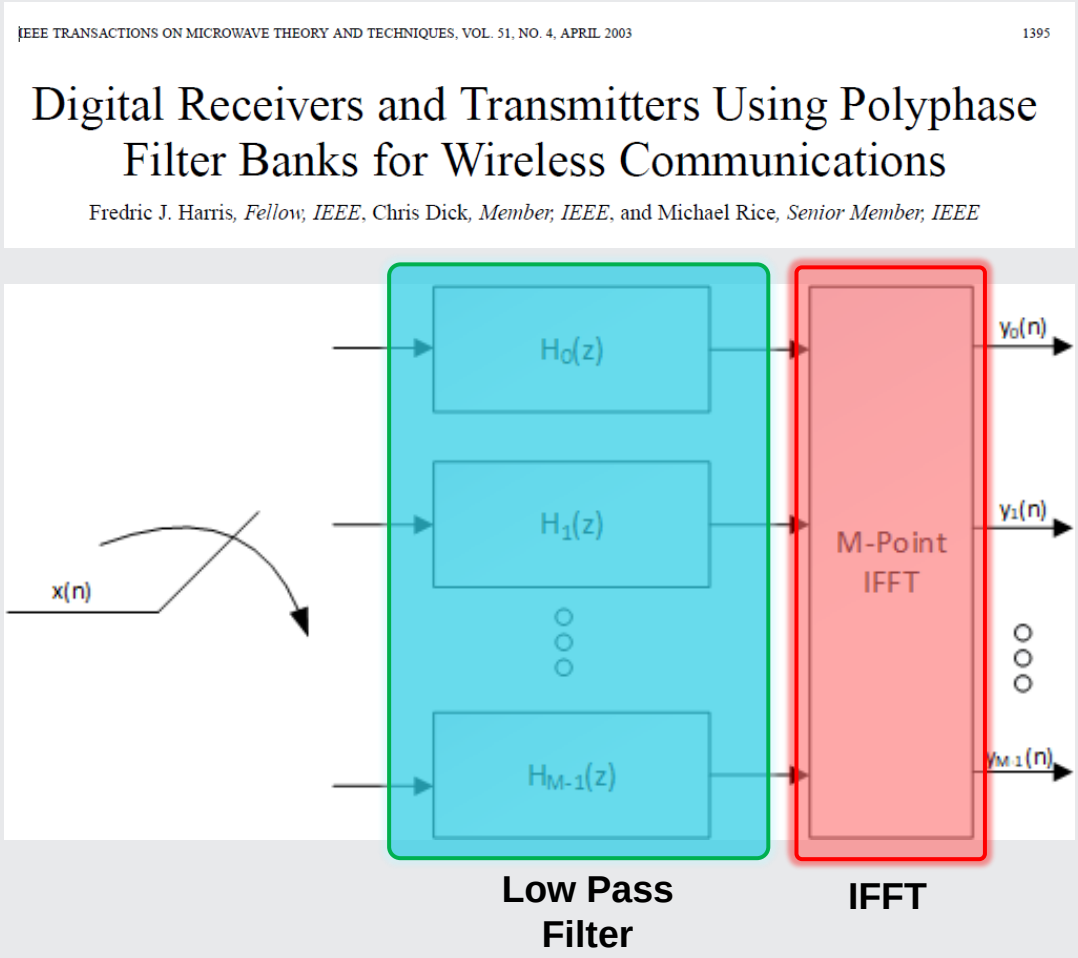
1. What is a Polyphase Channelizer?
2. New AMD Vitis™ Libraries IPs for Building Channelizers!
3. Example Channelizer with 4,096 Channels @ 2 GSPS on AIE-ML Using the New IPs
4. Current Availability of newly introduced IPs and Design Example

What is a Polyphase Channelizer?

AMD Vitis™ Libraries has Two New IPs for Building Channelizers

Polyphase Channelizer on AMD AIE-ML

System Requirement	
Parameters	**Value**
Sampling Rate (Fs)	2000 MSPS
# of Channels (M)	4096
# of Taps per Channel (K)	36
Input Datatype	cint16
Output Datatype	cint32
Filterbank Coefficient Type	int32



AMD Vitis Libraries

Has two new IPs

To build **Channelizer**

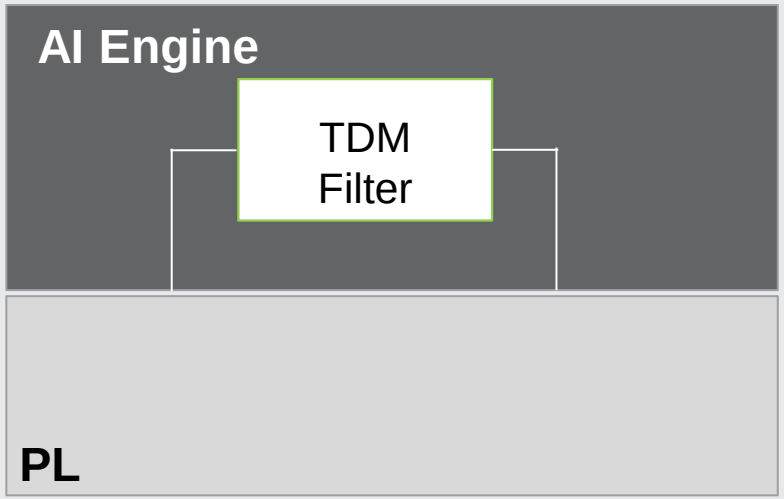
- Time-Division Multiplexing (TDM) FIR filter
- 2D FFT/IFFT

AMD Vitis™ Libraries

AMD Vitis™ Libraries comes in various formats

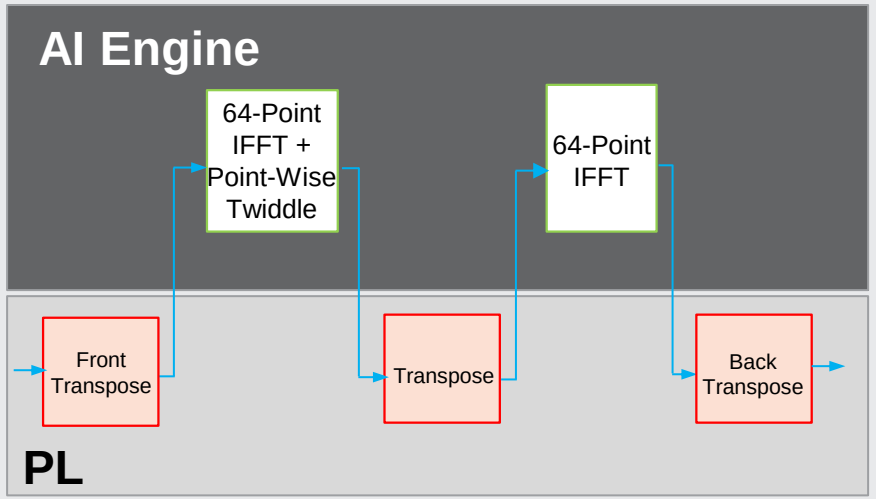
AMD Versal™ AI Engine Only

- Time-Division Multiplexing (TDM) FIR filter



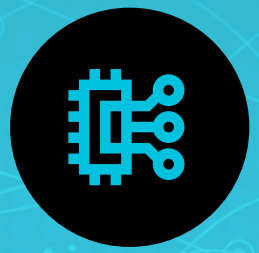
Vitis Subsystem (VSS) [AI Engine + PL]

- 2D FFT/IFFT



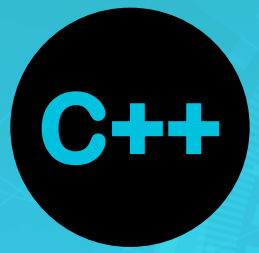
New AMD Vitis™ Libraries IPs for Building Channelizers

What is the AMD Versal™ AI Engine DSP Library?

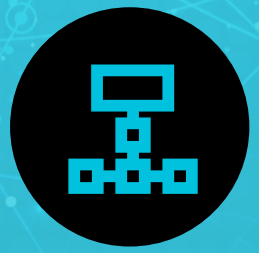


Configurable library of kernels that can be used to develop applications on AMD Versal AI Engines

Open-source library for DSP applications



Kernels are coded using Versal AI Engine APIs in C++ to 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

IP is instantiated & configured using its L2 graph

Scope of AMD Versal™ AI Engine DSP/Solver Libraries

Five Algorithm Categories

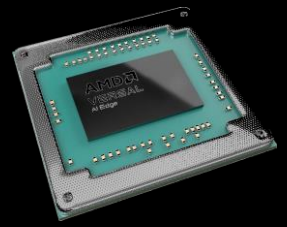
- FIR — DSP Filters
- TRANSFORMS — Various Transforms
- BLAS — Basic Linear Algebra
- SOLVER Ax = b — Solving Linear Systems
- VECTORIZED COMPUTE — SIMD Compute Kernels

+

Three AI Engine Architecture Variants

AIE

AIE-ML



AIE-ML v2

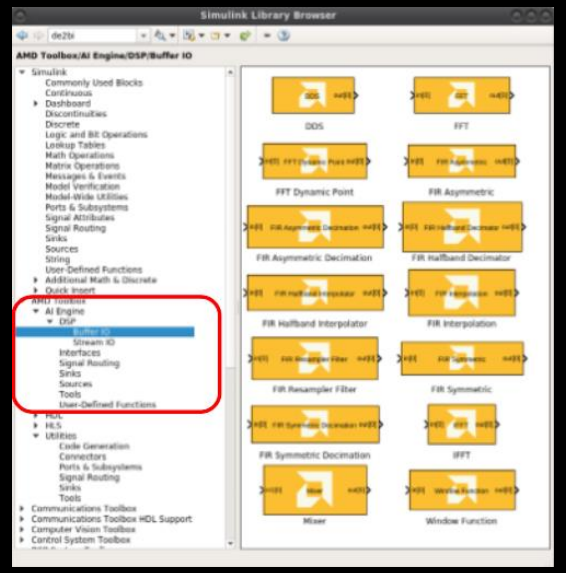
+

Support Several Data Types

- int16
- int32
- cint16
- cint32
- float
- cfloat
- bfloat16

+

AMD Vitis™ Model Composer Blockset



=

AI Engine DSP/ Solver Libraries

TDM FIR Parameters

Graph Entry	xf::dsp::aie::fir::tdm::fir_tdm_graph		
Data Type	Data Type	TT_DATA	int16,cint16,int32,cint32,float,cfloat
	Coefficient Type	TT_COEFF	int16,cint16,int32,cint32,float,cfloat
	Output Data Type	TT_OUT_DATA	Optional parameter to specify a different datatype for output compared to input
Taps		TP_FIR_LEN	Number of taps in the filter
Shift, Rounding & Saturate Selection		TP_SHIFT	Must be in the range of 0 to 61
		TP_RND	Selection of rounding method
		TP_SAT	Selection of saturation method
TDM Channels		TP_TDM_CHANNELS	Number of TDM channel processed by the FIR
Window Size		TP_INPUT_WINDOW_VSIZE	Number of samples processed in a single graph iteration. Must be a multiple of TDM channels
Broadcast		TP_NUM_OUTPUTS	Number of ports to broadcast the output, currently restricted to 1
Stream Input		TP_DUAL_IP	Allows 2 stream inputs to be used, increasing available bandwidth, currently restricted to 1
Parallelization		TP_SSR	Number of interleaved parallel I/O paths
		TP_CASC_LEN	Divide requested TP_FIR_LEN among cascaded kernels

2D FFT/IFFT Parameters

	Graph Entry		`xf::dsp::aie::fft::vss_1d::vss_fft_ifft_1d_graph`	
AMD Versal™ AI Engine only parameters	Data Type	Data Type	`TT_DATA`	`cint32, cfloat`
		Twiddle Data Type	`TT_TWIDDLE`	`cint16,cint32,cfloat`
	FFT or IFFT		`TP_FFT_NIFFT`	Transform to perform FFT (0) or IFFT (1)
	Interface Type		`TP_API`	Select window (0) or stream (1) interfaces
	Shift, Rounding, Saturation & Scaling Selection		`TP_SHIFT`	Must be in the range of 0 to 61
			`TP_RND`	Selection of rounding method
			`TP_SAT`	Selection of saturation method
			`TP_TWIDDLE_MODE`	Magnitude of integer twiddle
AI Engine/PL parameters	Transform Point Size		`TP_POINT_SIZE`	Number of samples in the transform. This must be 2^N where N is an integer in the range * 4 to 16 inclusive
	Parallelization (Sub-frame Processors)		`TP_SSR`	Describe N where 2^N is the number of subframe processors, to achieve higher throughput

Building a Polyphase Channelizer using the New IPs 4K Channels @ 2GSPS, Targeting AMD Versal™ AIE-ML

Overview of System Partitioning Methodology

Requirements Gathering

- Analyze algorithm characteristics
- Analyze system requirements
- Analyze hardware requirements

Solution Synthesis

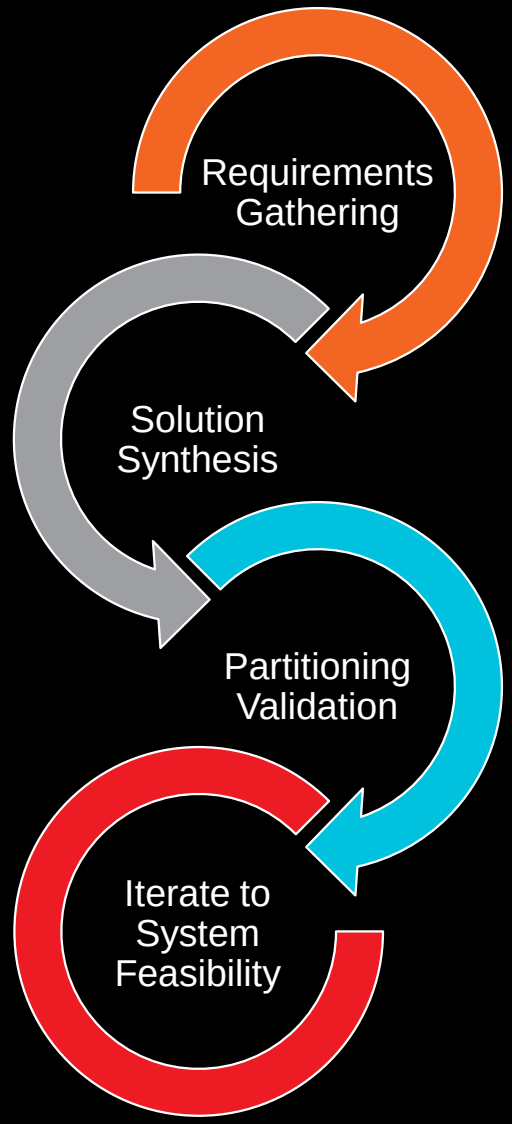
- Assess suitability of AMD Versal™ AI Engine processing
- Partition algorithms to AI Engine, PL, PS
- Identify system data flow

Partitioning Validation

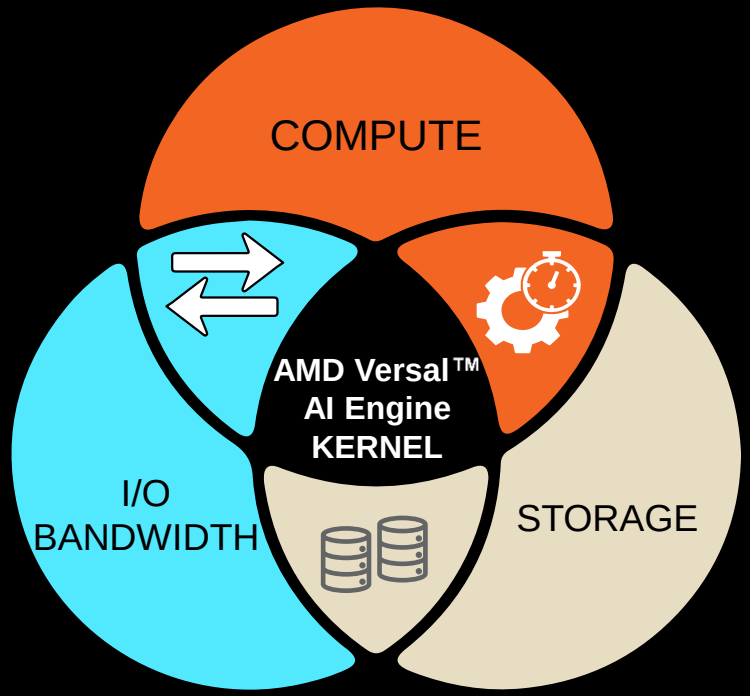
- Algorithm vectorization for AI Engine subsystems
- Prototyping of AI Engine subsystems
- Prototyping of PL subsystems
- Power analysis using PDM
- Traffic generator analysis

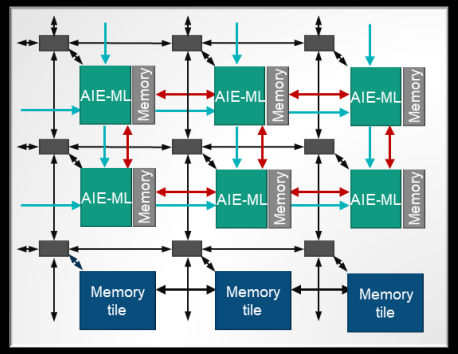
Iterate to System Feasibility

- Device selection & speed grade confirmed
- Initial definition of required custom platform
- Data flow template model of AI Engine subsystems



System Mapping Objectives





AMD Versal AIE-ML

Understanding the number of AMD Versal™ AI Engine tiles required based on compute

- Data types
- Sampling rate
- Algorithm

Refer: MACs based on data types

Memory requirements such as local memory and non-neighboring tile memory

- Buffer size
- Data types
- RTPs
- Stack size
- DMA FIFOs
- Memory tile (AIE-ML)

Understanding the number of ports required for the design

- Number of PLIO ports
- PL clock
- Buffer/stream interface

System mapping helps to achieve your design goal

System Partitioning: Filterbank – Compute

System Requirement	
Parameters	**Value**
Sampling Rate (Fs)	2,000 MSPS
# of Channels (M)	4096
# of Taps per Channel (K)	36
Input Datatype	`cint16`
Output Datatype	`cint32`
Filterbank Coefficient Type	`int32`

Precision 1	Precision 2	Number of Accumulator Lanes	Bits per Accumulator Lane	Number of MACs
`int32`	`cint16`	8	64	**8**

Compute

Compute capability based on data type	8 MACs/cycle
Tile compute capacity	8 x 1.25 GHz = 10 GMAC/sec
Per channel compute	36 x 2G/4096 = 0.0176 GMAC/sec
Compute need for all channels	4096 x 0.0176 = 72 GMAC/sec
# Tiles based on compute requirement	8

System Partitioning: Filterbank – Storage and Bandwidth

System Requirement	
Parameters	**Value**
Sampling Rate (Fs)	2000 MSPS
# of Channels (M)	4096
# of Taps per Channel (K)	36
Input Datatype	`cint16`
Output Datatype	`cint32`
Filterbank Coefficient Type	`int32`

Storage

AMD Versal™ AIE-ML Local Tile Memory		64 KB
Coefficient Size	Coefficients Type	int32 taps @ 4B = 36 x 4 = 144 bytes
	Total Coefficients	4096 x 144 = 589824 = 576 KB
Data Size	Data Type	cint16 @ 4B = 35 x 4 = 140 bytes
	State History	4096 x 140 = 573440 = 560 KB
Total Memory Size (Coeff + Data)		576 + 560 = 1136 KB
# Tiles Based on Storage Requirement		1136 KB => 1136/64 = ~17 -> 32

Bandwidth

Device Capability of One Port	32 bits @ 1250 MHz
# Input Ports (cint16 @ 2 GSPS)	Two ports
# Output Ports (cint32 @ 2GSPS)	Four ports

System Partitioning: Filterbank - Summary

Compute

# Tiles Based on Compute Requirement	8

8 Tiles

Storage

Total Memory Size (Coeff + Data)	1152 + 560 = 1712 KB
# Tiles Based on Storage Requirement	1712 KB = 1712/64 = ~17 -> 32 tiles

32 Tiles

Bandwidth

# Input Ports (cint16 @ 2 GSPS)	Two ports
# Output Ports (cint32 @ 2GSPS)	Four ports

2 Input Ports

4 Output Ports

How to Configure TDM FIR to Achieve System Partitioning Targets?

```
class firbank_graph : public graph {
public:
  typedef cint16                    TT_DATA;
  typedef cint32                    TT_OUT_DATA;
  typedef int32                     TT_COEFF;
  static constexpr unsigned         TP_FIR_LEN            = 36;
  static constexpr unsigned         TP_TDM_CHANNELS       = 4096;
  static constexpr unsigned         TP_SHIFT              = 15;
  static constexpr unsigned         TP_RND                = 12;
  static constexpr unsigned         TP_NUM_OUTPUTS        = 1;
  static constexpr unsigned         TP_DUAL_IP            = 0;
  static constexpr unsigned         TP_SAT                = 1;
  static constexpr unsigned         TP_SSR                = 32;
  static constexpr unsigned         TP_INPUT_WINDOW_VSIZE = 4096;
  static constexpr unsigned         TP_CASC_LEN           = 1;

  using TT_FIR = xf::dsp::aie::fir::tdm::fir_tdm_graph<TT_DATA,TT_COEFF,TP_FIR_LEN,TP_SHIFT,TP_RND,TP_INPUT_WINDOW_VSIZE,
                                                       TP_TDM_CHANNELS,TP_NUM_OUTPUTS,TP_DUAL_IP,TP_SSR,TP_SAT,TP_CASC_LEN,TT_OUT_DATA>;

  TT_FIR tdmfir;

  std::array< port< input>,TP_SSR> sig_i;
  std::array<port< output>,TP_SSR> sig_o;

  // Constructor:
  firbank_graph( std::vector<typename firbank_graph::TT_COEFF> TAPS_INIT_0 ) : tdmfir(TAPS_INIT_0)
  {
    for (unsigned ii=0; ii < TP_SSR; ii++) {
      connect<>( sig_i[ii], tdmfir.in[ii] );
      connect<>( tdmfir.out[ii], sig_o[ii] );
    }
  }
};
```

Algorithm Requirements

From System Mapping Analysis

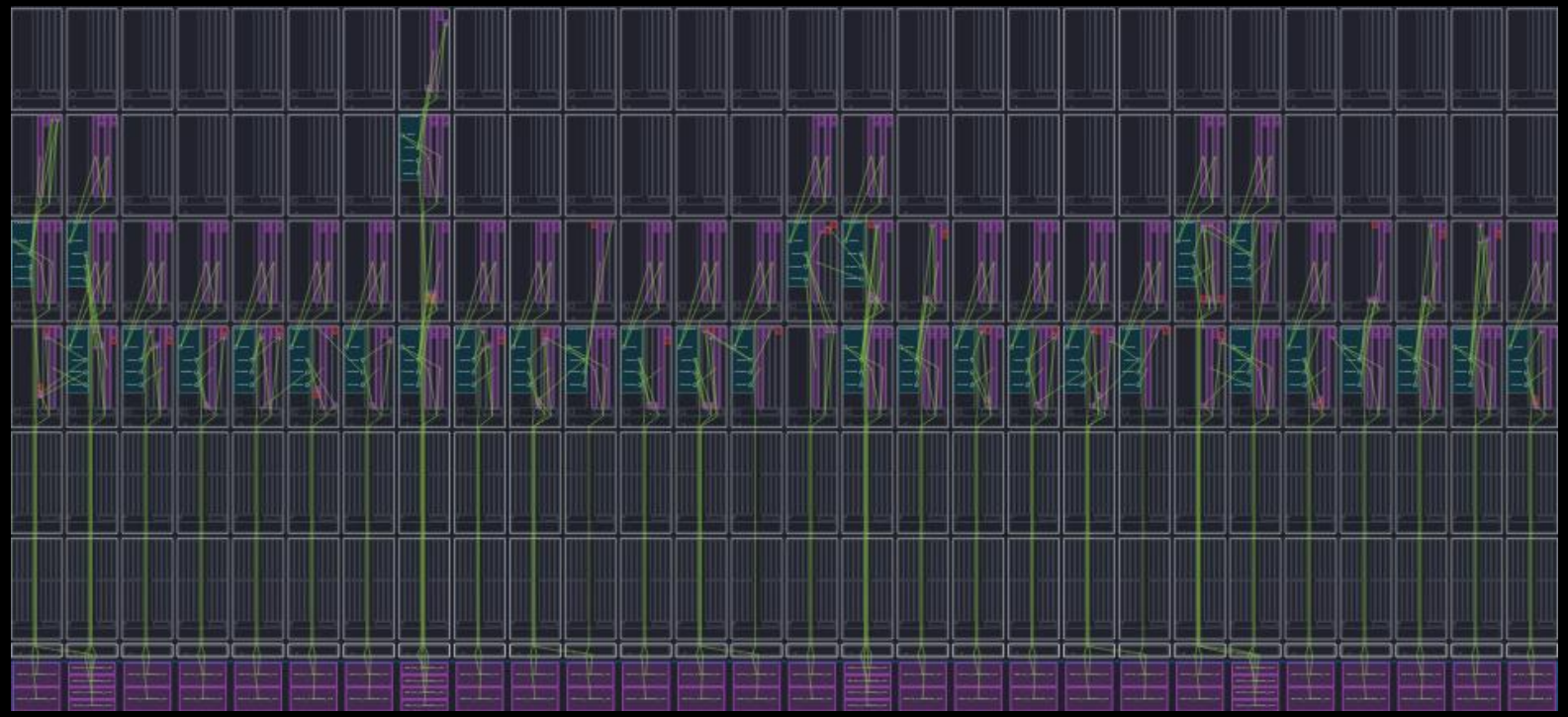
Storage-Bound

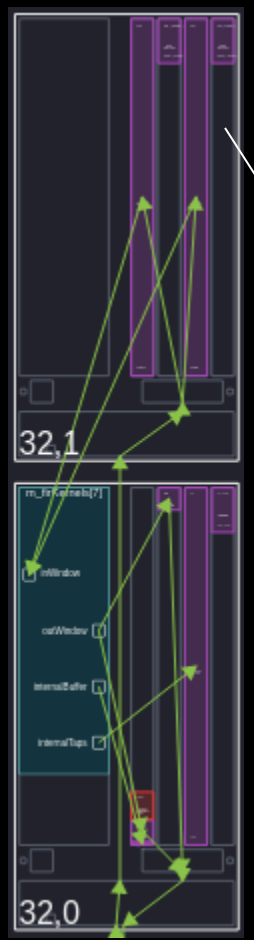
32 Tiles

Filterbank Prototyping – Resource Usage

Resource Usage	System Mapping Analysis (Expected)	Actual
# tiles	32	64

Double the expected. Why?





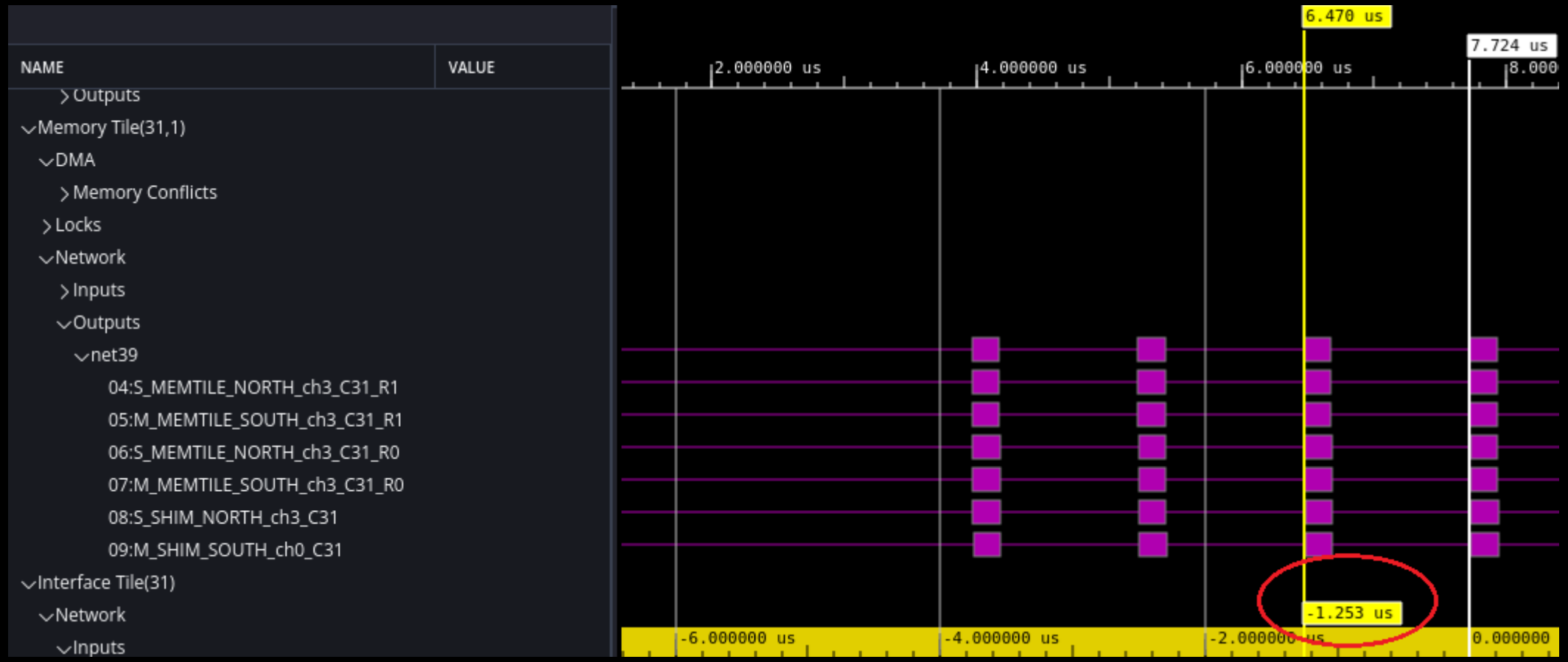
Key Finding:

- State history of TDM FIR stored with input window (double-buffered)

Filterbank Prototyping – Throughput

Plenty of margin. Can we trade-off storage for throughput?

Throughput = 4,096/1.253 = 3,270 MSPS >> 2,000 MSPS



Filterbank Optimization – Trade-Off Throughput for Storage

```
single_buffer(dut.tdmfir.m_firKernels[ii+0].in[0]);
location<kernel>    (dut.tdmfir.m_firKernels[ii])                =       tile(start_index+xoff,0);
location<stack>     (dut.tdmfir.m_firKernels[ii])                =       bank(start_index+xoff,0,3);
location<parameter>(dut.tdmfir.m_firKernels[ii].param[0])        =       bank(start_index+xoff,0,3);
location<parameter>(dut.tdmfir.m_firKernels[ii].param[1])        = address(start_index+xoff,0,0x4C00);
location<buffer>    (dut.tdmfir.m_firKernels[ii].in[0])          =       bank(start_index+xoff,0,0);
location<buffer>    (dut.tdmfir.m_firKernels[ii].out[0])         = {     bank(start_index+xoff,0,1),
                                                                         bank(start_index+xoff,0,3) };
```

- Use single_buffer on input connection
- Place each tile's storage locally
- This will come at the expense of throughput

	System Mapping Analysis	Baseline Design	Optimized Design
# Tiles	32	64	32

Throughput = 2,200 MSPS

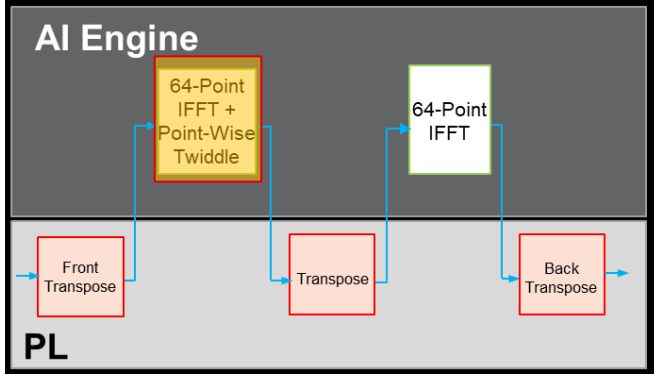
Prototyping 2D IFFT IP to Assess Required # Instances

```
class ifft4096_2d_graph : public graph {
public:
  typedef cint32              TT_DATA;
  typedef cint16              TT_TWIDDLE;
  static constexpr unsigned TP_POINT_SIZE = 4096;
  static constexpr unsigned TP_FFT_NIFFT = 0;
  static constexpr unsigned TP_SHIFT = 0;
  static constexpr unsigned TP_API = 0;
  static constexpr unsigned TP_SSR = 1;
  static constexpr unsigned TP_RND = 12;
  static constexpr unsigned TP_SAT = 1;
  static constexpr unsigned TP_TWIDDLE_MODE = 0;

  std::array< port< input>,TP_SSR>  front_i;
  std::array< port< input>,TP_SSR>  back_i;
  std::array<port< output>,TP_SSR>  front_o;
  std::array<port< output>,TP_SSR>  back_o;

  using TT_2D_IFFT = xf::dsp::aie::fft::vss_1d::vss_fft_ifft_1d_graph<TT_DATA,TT_TWIDDLE,TP_POINT_SIZE,TP_FFT_NIFFT,
                                                                      TP_SHIFT,TP_API,TP_SSR,TP_RND,TP_SAT,TP_TWIDDLE_MODE>;

  TT_2D_IFFT ifft4096_2d;

  // Constructor:
  ifft4096_2d_graph( void )
  {
    for ( unsigned ff=0; ff < TP_SSR; ff++) {
      connect<>( front_i[ff],                ifft4096_2d.front_i[ff] );
      connect<>( ifft4096_2d.front_o[ff],      front_o[ff] );
      connect<>( back_i[ff],                 ifft4096_2d.back_i[ff] );
      connect<>( ifft4096_2d.back_o[ff],       back_o[ff] );
      location<kernel>(ifft4096_2d.m_fftTwRotKernels[ff]) = location<kernel>(ifft4096_2d.frontFFTGraph[ff].FFTwinproc.m_fftKernels[0]);
    }
  }
};
```

Algorithm requirements

Benchmark and measure performance

Group front IFFT and point-wise twiddle multiplication into single tile



Prototyping 2D IFFT to Assess Required # Instances

Using SSR=1, we achieve

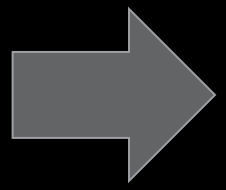
Front IFFT + Point-Wise Twiddle	
Required	2,000 MSPS
Achieved	466 MSPS

Requires 5 instances to meet requirement

Back IFFT	
Required	2,000 MSPS
Achieved	539 MSPS

Requires 4 instances to meet requirement

IP is restricted to have a common # instances for front and back via SSR parameter



SSR=5 should achieve our throughput requirement

Configure 2D IFFT IP to Achieve 2,000 MSPS

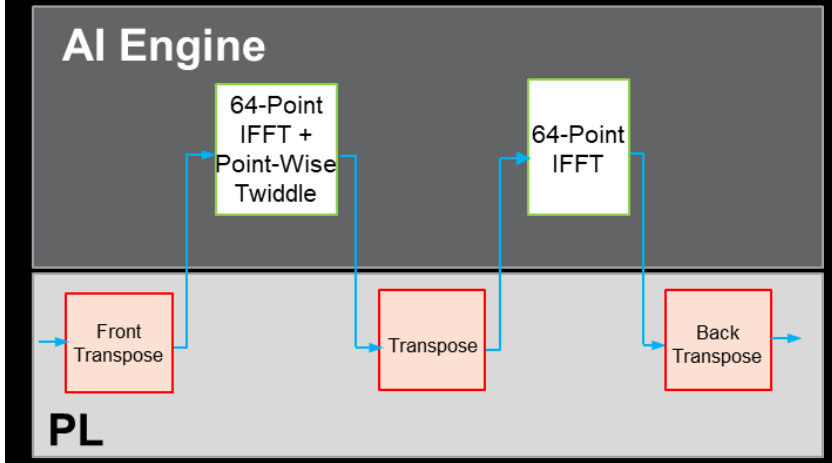
Achieve throughput:

- **SSR=5** should be enough to meet throughput

```
class ifft4096_2d_graph : public graph {
public:
  typedef cint32            TT_DATA;
  typedef cint16            TT_TWIDDLE;
  static constexpr unsigned TP_POINT_SIZE = 4096;
  static constexpr unsigned TP_FFT_NIFFT = 0;
  static constexpr unsigned TP_SHIFT = 0;
  static constexpr unsigned TP_API = 0;
  static constexpr unsigned TP_SSR = 8;
  static constexpr unsigned TP_RND = 12;
  static constexpr unsigned TP_SAT = 1;
  static constexpr unsigned TP_TWIDDLE_MODE = 0;

  std::array< port< input>,TP_SSR>  front_i;
  std::array< port< input>,TP_SSR>  back_i;
  std::array<port< output>,TP_SSR>  front_o;
  std::array<port< output>,TP_SSR>  back_o;

  using TT_2D_IFFT = xf::dsp::aie::fft::vss_1d::vss_fft_ifft_1d_graph<TT_DATA,TT_TWIDDLE,
                                                                      TP_POINT_SIZE,TP_FFT_NIFFT,
                                                                      TP_SHIFT,TP_API,TP_SSR,TP_RND,
                                                                      TP_SAT,TP_TWIDDLE_MODE>;
```

Configured to meet 2000 MSPS

- Using **SSR=8** simplifies the mapping of Filterbank to IFFT
  - No state machine is required



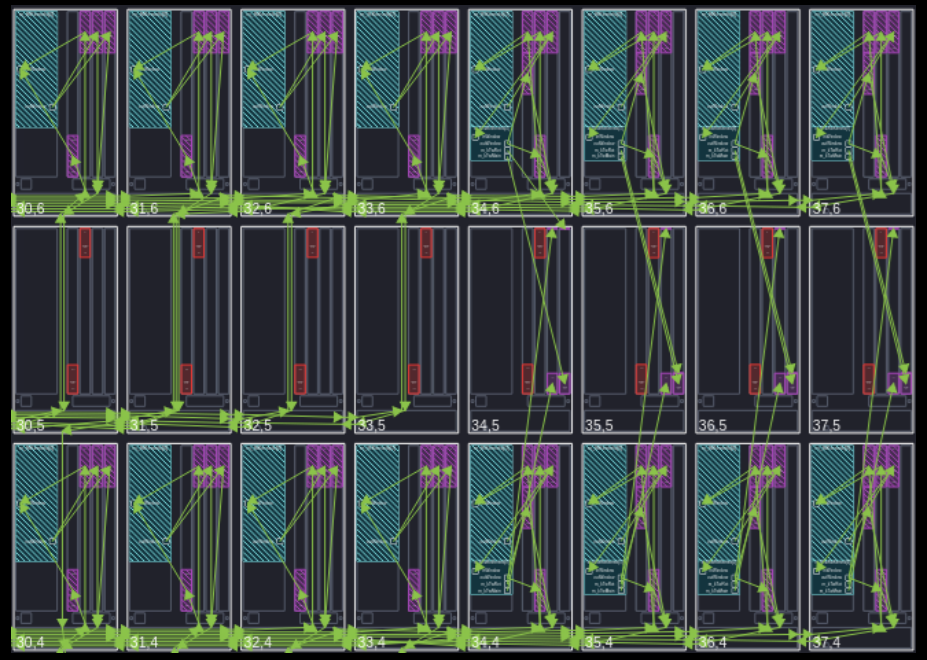
Architecting a High-Throughput IFFT – Implementation with SSR=8

Front IFFT Throughput = 4,096/1.164 = 3,519 MSPS

Back IFFT Throughput = 4,096/.973 = 4,210 MSPS

AI Engine Resource Utilization

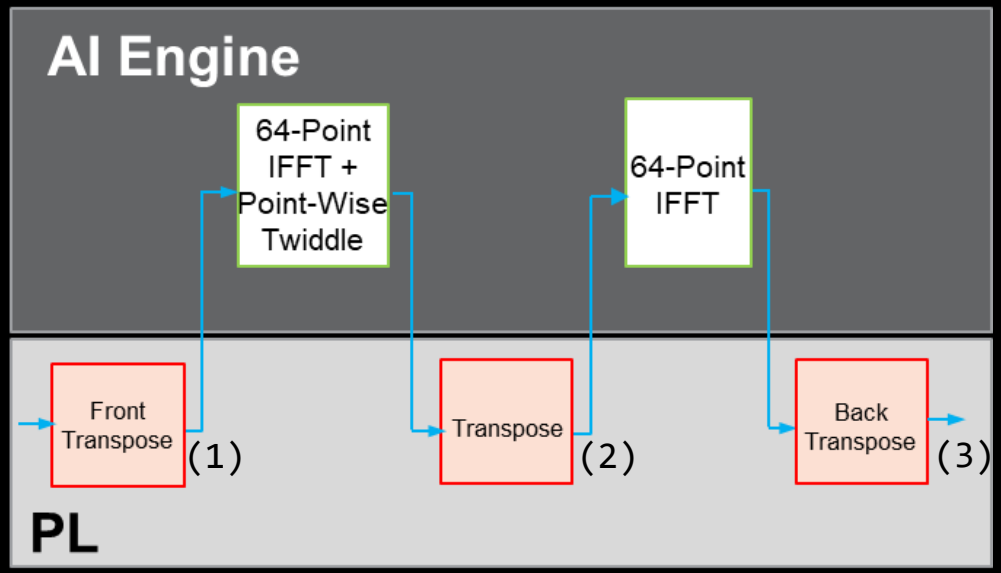
Tiles used for AI Engine Kernels:	16 of 304 (5.26 %)
Tiles used for Buffers:	24 of 304 (7.89 %)
Memory Tiles used for Shared Buffers:	0 of 76 (0.00 %)
Tiles used for Stream Interconnect:	139 of 381 (36.48 %)
Interface Channels used for ADF Input/Output:	32 (PLIO: 32)
Interface Channels used for Trace data:	0



Heterogeneous Compute: Using PL for Sample-Oriented Processing

Configure 2D IFFT HLS IP

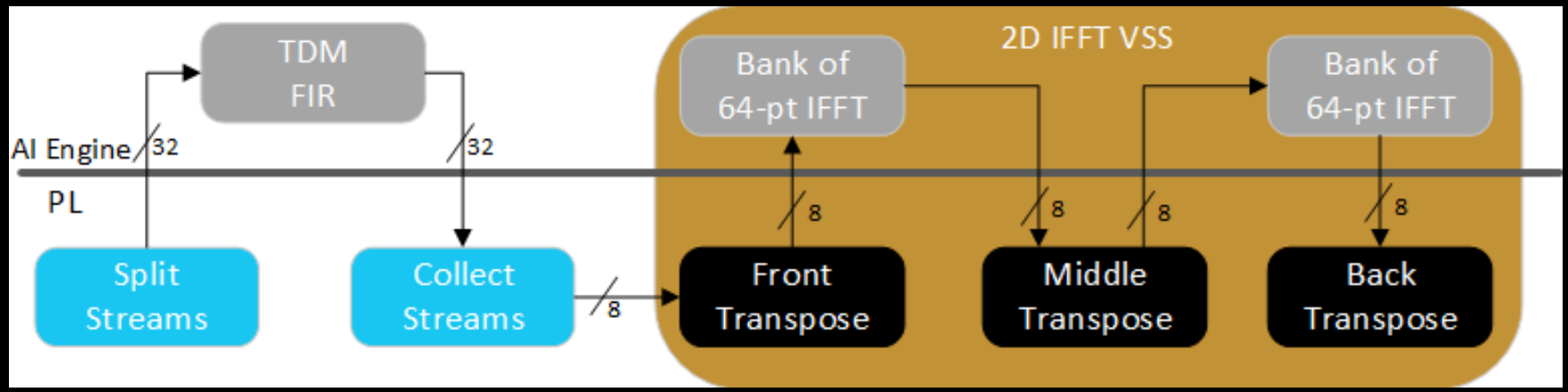
- SSR = 8 (matches AI Engine configuration)
- POINT_SIZE = 4096 (matches AI Engine configuration)
- `<dsp_lib>/L1/src/hw/ifft_front_transpose (1)`
- `<dsp_lib>/L1/src/hw/ifft_transpose (2)`
- `<dsp_lib>/L1/src/hw/ifft_back_transpose (3)`



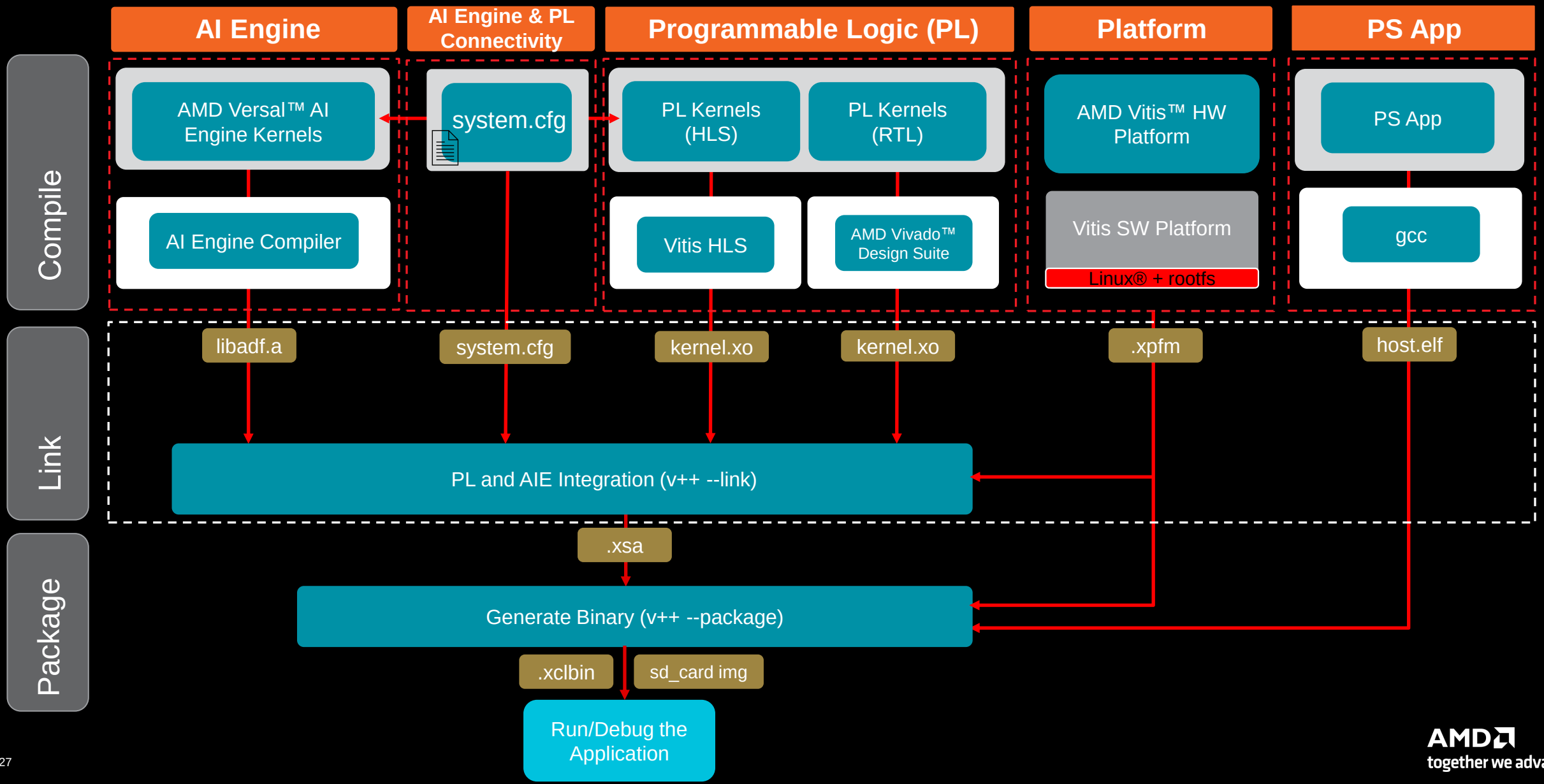
Channelizer – Complete System Diagram

Design Summary

- TDM FIR uses 32 AMD Versal™ AI Engine tiles with 32 I/O streams
- 2D IFFT implemented uses a mixture of AIE and PL resources, with 8 I/O streams
- Split/collect streams implemented to manage connectivity between IP blocks

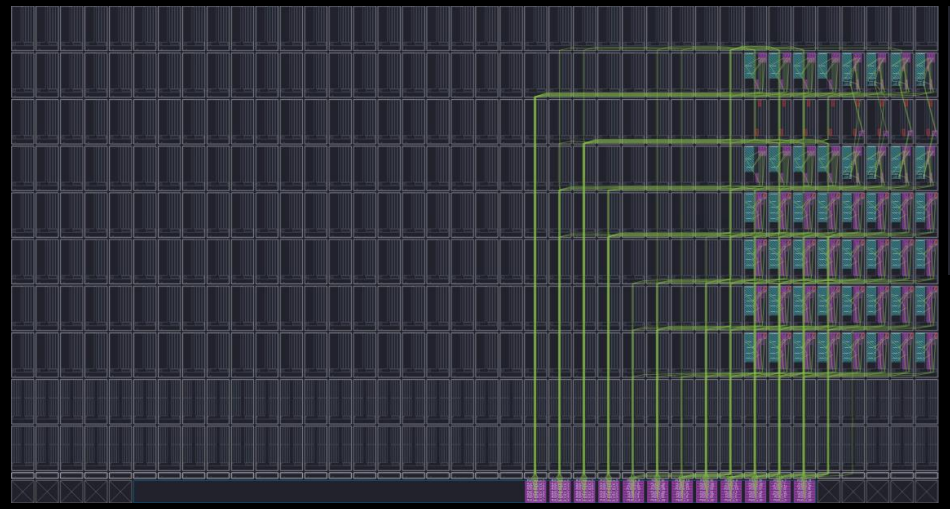


Channelizer Integration through Vitis™ Tool Flow for Versal™ Devices



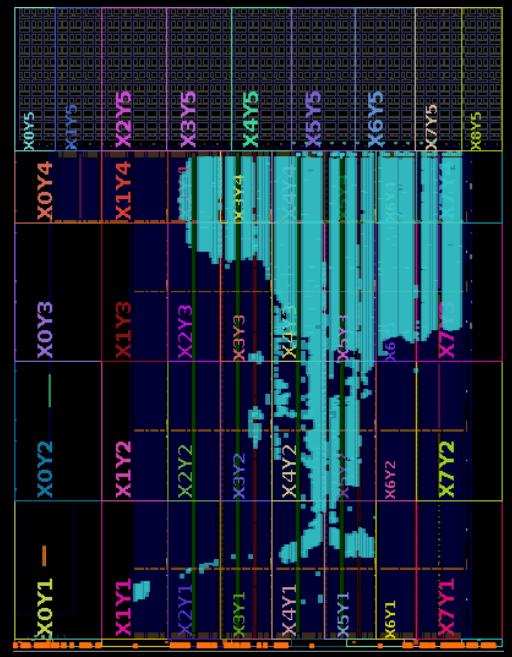
Channelizer 4K Channels @ 2 GSPS on AIE-ML – Implementation

AI Engine Implementation



AI Engine Resource Utilization

Tiles used for AI Engine Kernels:	48 of 304 (15.79 %)
Tiles used for Buffers:	56 of 304 (18.42 %)
Memory Tiles used for Shared Buffers:	0 of 76 (0.00 %)
Tiles used for Stream Interconnect:	173 of 381 (45.41 %)
Interface Channels used for ADF Input/Output:	96 (PLIO: 96)
Interface Channels used for Trace data:	0



PL Implementation

Resource	Utilization	Available	Utilization %
LUT	66265	520704	12.73
FF	140636	1041408	13.50
BRAM	143.50	600	23.92
URAM	16	264	6.06
IO	336	530	63.40
MMCM	2	9	22.22

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.000 ns	Worst Hold Slack (WHS):	0.002 ns	Worst Pulse Width Slack (WPWS):	0.016 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	454869	Total Number of Endpoints:	453745	Total Number of Endpoints:	164592

All user specified timing constraints are met.

Simulating the Complete Design in hw_emu

- Design is functionally correct
- Achieves target throughput

```
ss: 16360  Gld: 33432 40  Act: 33423 44  Err: 9 4
ss: 16361  Gld: -10932 9781  Act: -10936 9787  Err: 4 6
ss: 16362  Gld: 29444 12737  Act: 29439 12749  Err: 5 12
ss: 16363  Gld: 15054 -2322  Act: 15051 -2331  Err: 3 9
ss: 16364  Gld: 2330 16325  Act: 2338 16329  Err: 8 4
ss: 16365  Gld: 216 1721  Act: 221 1711  Err: 5 10
ss: 16366  Gld: -3708 18999  Act: -3708 18992  Err: 0 7
ss: 16367  Gld: -10300 8012  Act: -10306 8014  Err: 6 2
ss: 16368  Gld: -6638 5957  Act: -6638 5953  Err: 0 4
ss: 16369  Gld: 4170 -674  Act: 4168 -664  Err: 2 10
ss: 16370  Gld: 8899 1897  Act: 8914 1899  Err: 15 2
ss: 16371  Gld: 9511 18087  Act: 9510 18085  Err: 1 2
ss: 16372  Gld: 12576 -5810  Act: 12576 -5807  Err: 0 3
ss: 16373  Gld: 28733 -1254  Act: 28727 -1259  Err: 6 5
ss: 16374  Gld: 16177 -20661  Act: 16174 -20656  Err: 3 5
ss: 16375  Gld: -7603 -1098  Act: -7610 -1097  Err: 7 1
ss: 16376  Gld: 4252 19009  Act: 4252 19015  Err: 0 6
ss: 16377  Gld: 20743 14211  Act: 20739 14216  Err: 4 5
ss: 16378  Gld: 18148 18240  Act: 18151 18239  Err: 3 1
ss: 16379  Gld: -8065 6295  Act: -8071 6292  Err: 6 3
ss: 16380  Gld: 17064 -10287  Act: 17056 -10290  Err: 8 3
ss: 16381  Gld: 2061 17956  Act: 2073 17968  Err: 12 12
ss: 16382  Gld: 10270 34560  Act: 10266 34561  Err: 4 1
ss: 16383  Gld: -12794 5457  Act: -12788 5453  Err: 6 4
Level: 16384
Max Error: 25
============================
Cycle count: 17104
Approx Throughput: 19158.1 MB/sec
Approx Throughput: 2394.76 Msps
============================

--- PASSED ---
```

Channelizer 4K Channels @ 2 GSPS on AMD Versal™ AIE-ML – Hardware Setup

Build the Project – Generate PDI

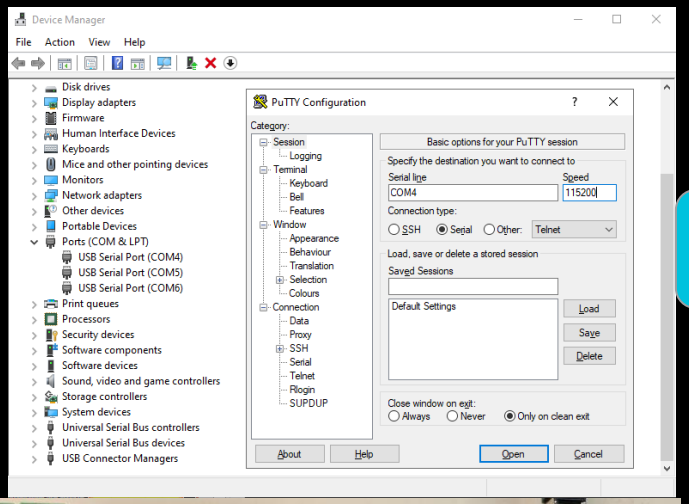
```
% make clean all TARGET=hw
```

Flash Program

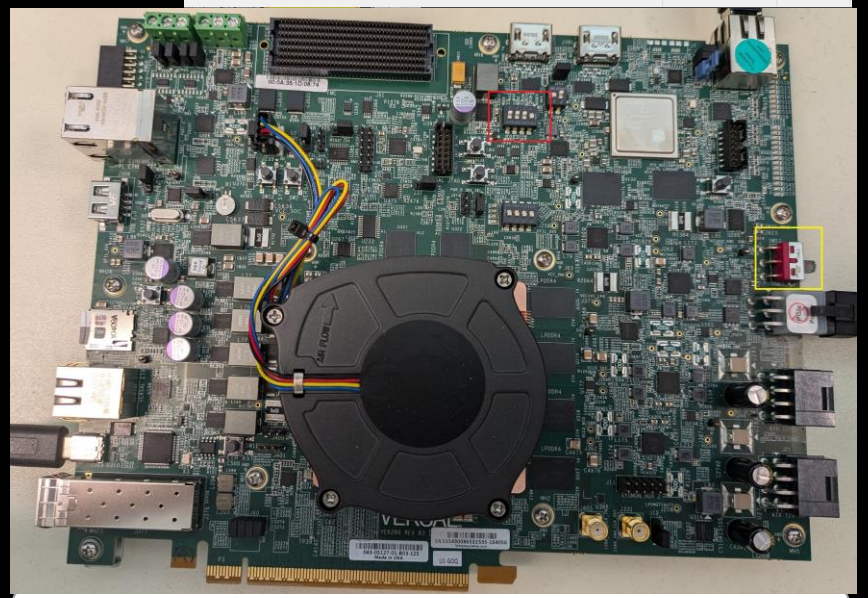
SD Card Image: channelizer/package/sd_card.img

Use PuTTY: Connect to VEK280 board using USB serial port
Use 115200

AMD Versal™ AI Edge Series VEK280 Evaluation Kit Board Settings:
Set the switches on the board to boot using sd_card as shown in red, then flip the switch shown in yellow to turn on the board



PuTTY



VEK280

Channelizer 4K Channels @ 2 GSPS on AMD Versal™ AIE-ML – Hardware run

VEK280 Board Power ON and run the design

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

$ cd /run/media/mmcblk0p1/
$ ./embedded_exec.sh a.xclbin
```

- This shall execute APIs existing in host.cpp
  - Measure throughput
  - Confirms completion of execution on hardware with a “PASSED” message

```
ss: 16360  Gld: 33432 40  Act: 33423 44  Err: 9 4
ss: 16361  Gld: -10932 9781  Act: -10936 9787  Err: 4 6
ss: 16362  Gld: 29444 12737  Act: 29439 12749  Err: 5 12
ss: 16363  Gld: 15054 -2322  Act: 15051 -2331  Err: 3 9
ss: 16364  Gld: 2330 16325  Act: 2338 16329  Err: 8 4
ss: 16365  Gld: 216 1721  Act: 221 1711  Err: 5 10
ss: 16366  Gld: -3708 18999  Act: -3708 18992  Err: 0 7
ss: 16367  Gld: -10300 8012  Act: -10306 8014  Err: 6 2
ss: 16368  Gld: -6638 5957  Act: -6638 5953  Err: 0 4
ss: 16369  Gld: 4170 -674  Act: 4168 -664  Err: 2 10
ss: 16370  Gld: 8899 1897  Act: 8914 1899  Err: 15 2
ss: 16371  Gld: 9511 18087  Act: 9510 18085  Err: 1 2
ss: 16372  Gld: 12576 -5810  Act: 12576 -5807  Err: 0 3
ss: 16373  Gld: 28733 -1254  Act: 28727 -1259  Err: 6 5
ss: 16374  Gld: 16177 -20661  Act: 16174 -20656  Err: 3 5
ss: 16375  Gld: -7603 -1098  Act: -7610 -1097  Err: 7 1
ss: 16376  Gld: 4252 19009  Act: 4252 19015  Err: 0 6
ss: 16377  Gld: 20743 14211  Act: 20739 14216  Err: 4 5
ss: 16378  Gld: 18148 18240  Act: 18151 18239  Err: 3 1
ss: 16379  Gld: -8065 6295  Act: -8071 6292  Err: 6 3
ss: 16380  Gld: 17064 -10287  Act: 17056 -10290  Err: 8 3
ss: 16381  Gld: 2061 17956  Act: 2073 17968  Err: 12 12
ss: 16382  Gld: 10270 34560  Act: 10266 34561  Err: 4 1
ss: 16383  Gld: -12794 5457  Act: -12788 5453  Err: 6 4
Level: 16384
Max Error: 25
============================
Cycle count: 16912
Approx Throughput: 19375.6 MB/sec
Approx Throughput: 2421.95 Msps
============================

--- PASSED ---
```

How Do I Get Access to New AMD Vitis™ Library IPs for Building Channelizers?

- TDM FIR General Availability (GA) in 2024.2 (November)
- 2D FFT/IFFT Early Access (EA) in 2024.2 (November)
- Example design will be uploaded as a GitHub tutorial in 2024.2

Summary

1. Polyphase channelizers are important designs in several signal processing market segments
2. AMD Vitis™ Libraries now deliver new IPs for rapid construction & delivery of high-performance channelizers
   - Parameterized to support a broad scope of system scenarios
   - Target AMD Versal™ AIE or AIE-ML devices
   - Delivered & packaged as new “AIE + PL” Vitis subsystems (VSS)
3. Example 4K channel @ 2 GSPS design on AMD Versal™ AIE-ML
4. Discussion on current availability of newly introduced IP blocks and design example

DISCLAIMER AND ATTRIBUTIONS


AMD