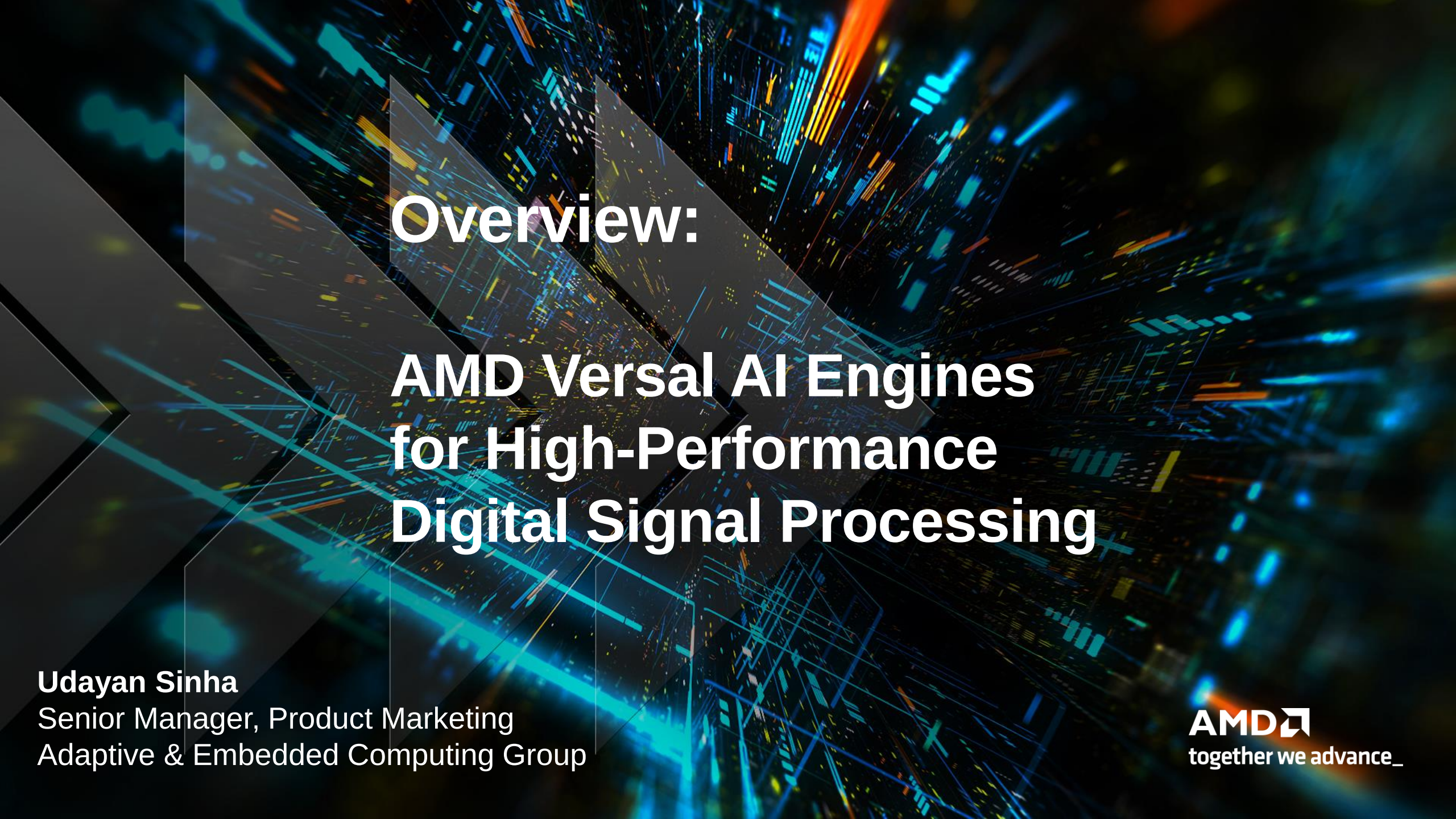




Overview:

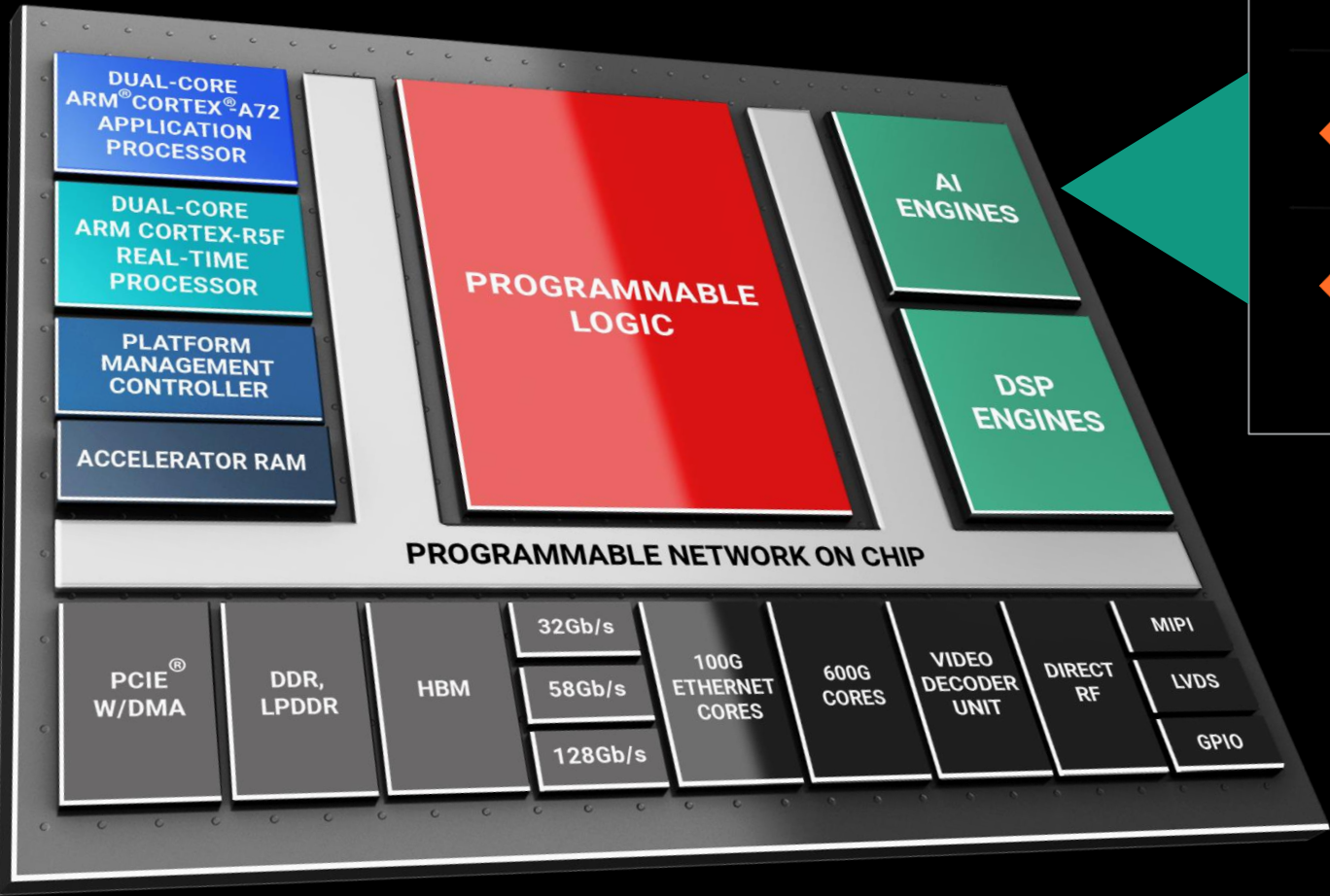
AMD Versal AI Engines for High-Performance Digital Signal Processing

Udayan Sinha

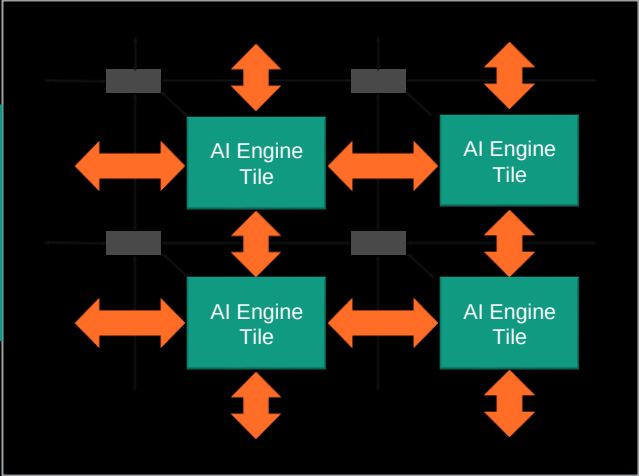
Senior Manager, Product Marketing
Adaptive & Embedded Computing Group

AMD 
together we advance_

AMD Versal™ Architecture Overview



Our focus for this presentation



AI Engine Applications

DSP



- FIR
- FFT
- Matrix Multiply

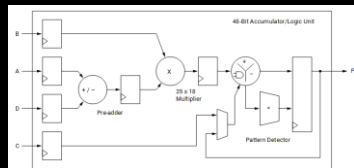
Machine Learning



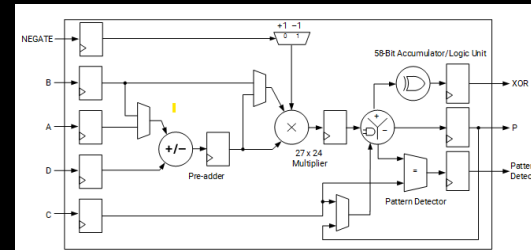
- CNN
- Gen AI
- Natural Language Processing

Why AI Engines for DSP?

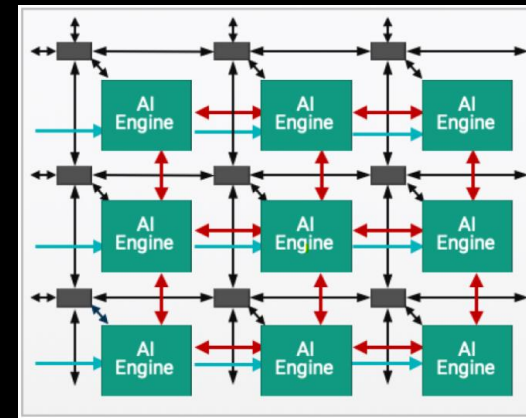
DSP48 Block



DSP58 Block



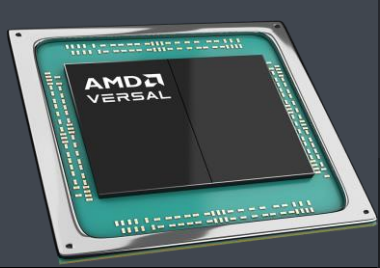
AI Engines



Significantly
increases overall
DSP compute
density

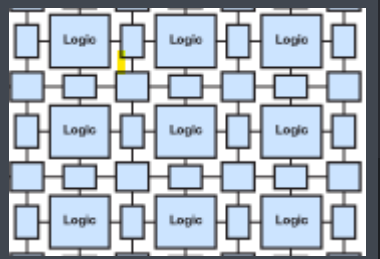
Increasing DSP Compute Requirements

Value Propositions for AMD AI Engine Technology



1. Increased Overall DSP Compute Density

More DSP compute compared to non-AI Engine devices



2. Programmable Logic (PL) Resource Savings

Significant LUT and FF saved vs. traditional PL



3. Dynamic Power Savings

Reduced dynamic power vs. traditional PL

Example Market Segments and Applications



Automotive & Industrial

Light Detection and Ranging (LiDAR)



Test & Measurement

Channel Emulation for
Wireless 5G/6G



Aerospace & Defense

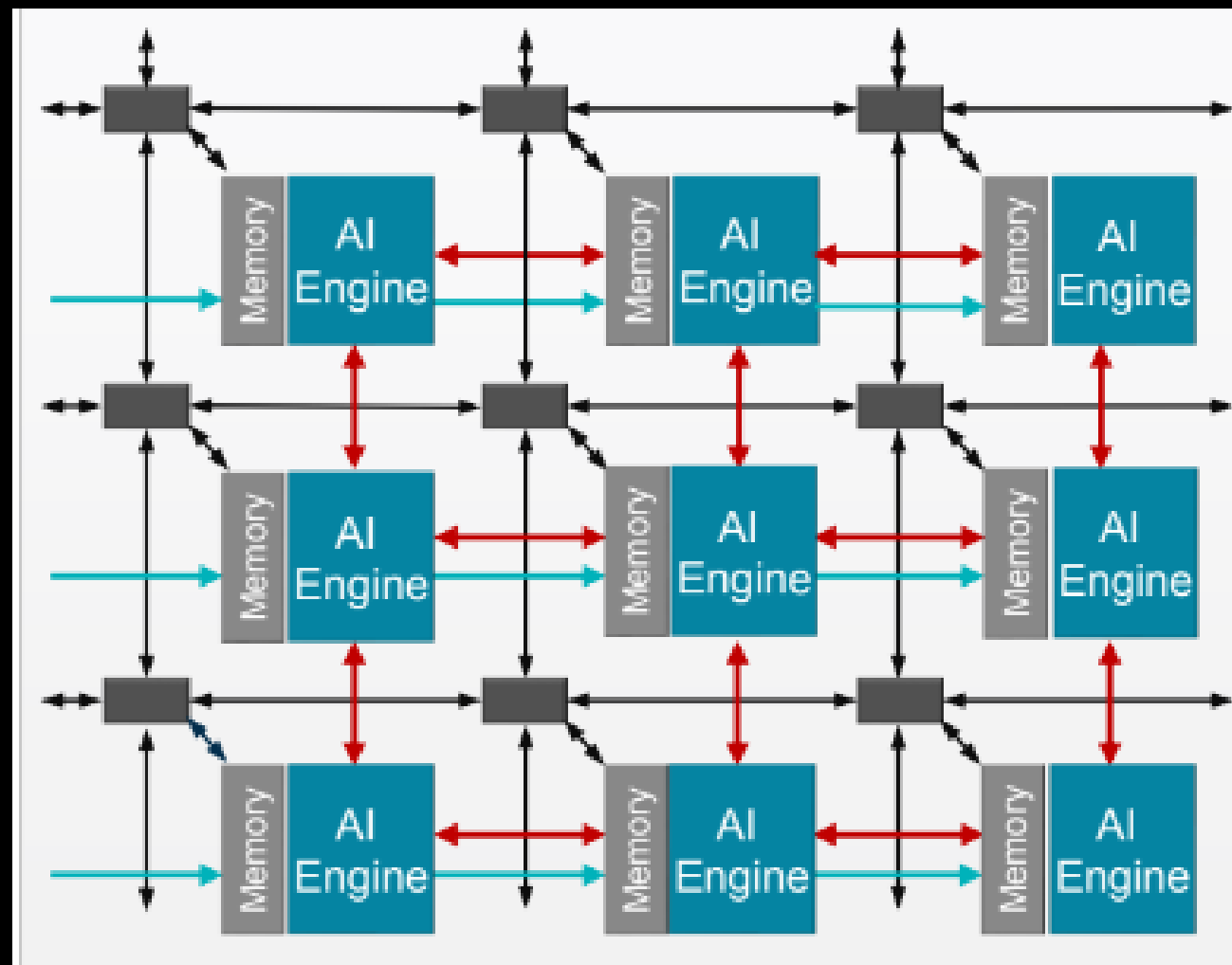
Next Gen Radar Systems

COMPONENTS OF THE AI ENGINE ARRAY

Certain Versal devices contain a two-dimensional array of AI Engines

Each AI Engine tile integrates a very long instruction word (VLIW) processor, integrated memory, and interconnects for streaming, configuration, and debug

The VLIW processor runs at over 1 GHz to implement high-performance DSP functions and machine learning algorithms



A Closer Look at the AI Engine

→

Memory Interface

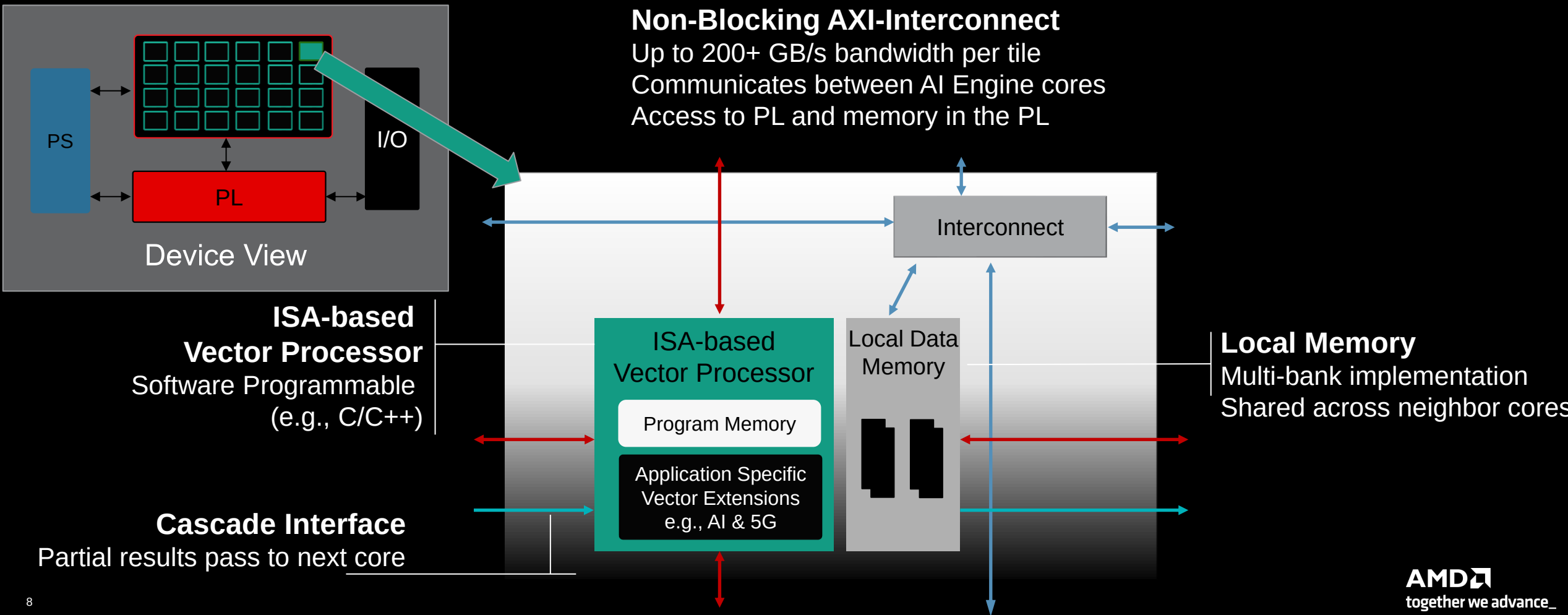
→

Stream Interface

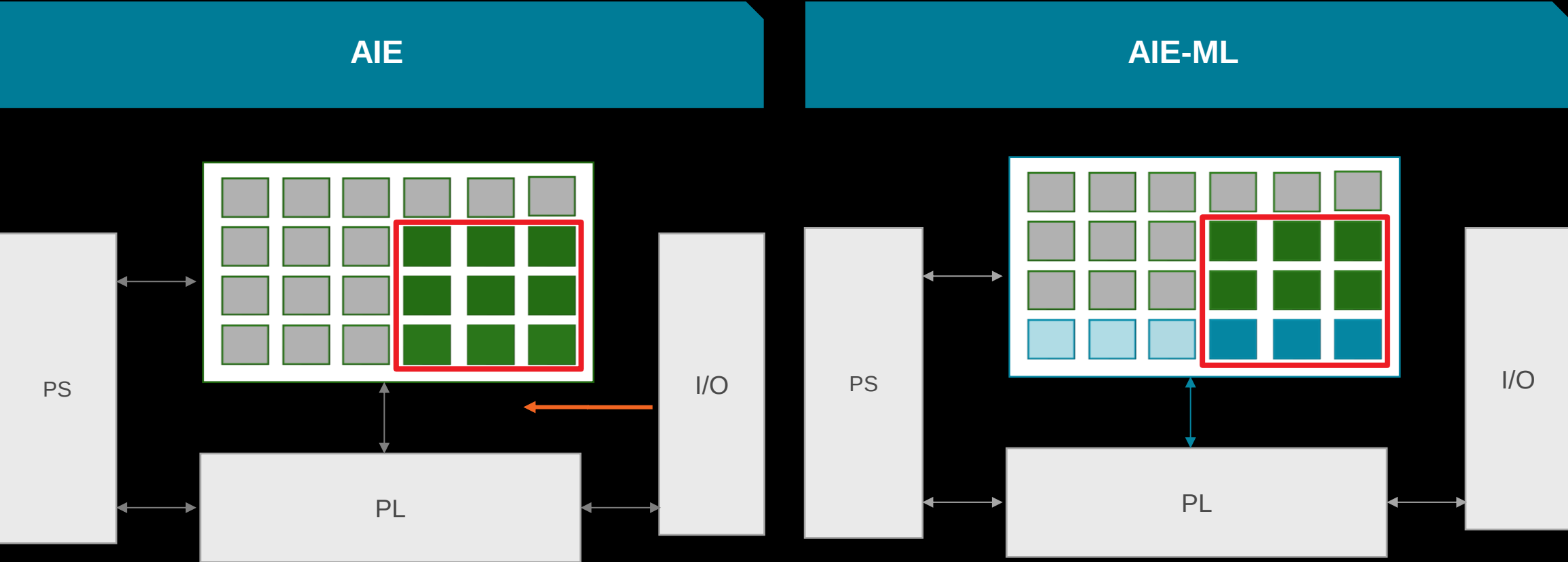
→

Cascade Interface

- The heart of the AI Engine is the **ISA-based Vector Processor**, a VLIW vector processor with AI and 5G specific vector extensions



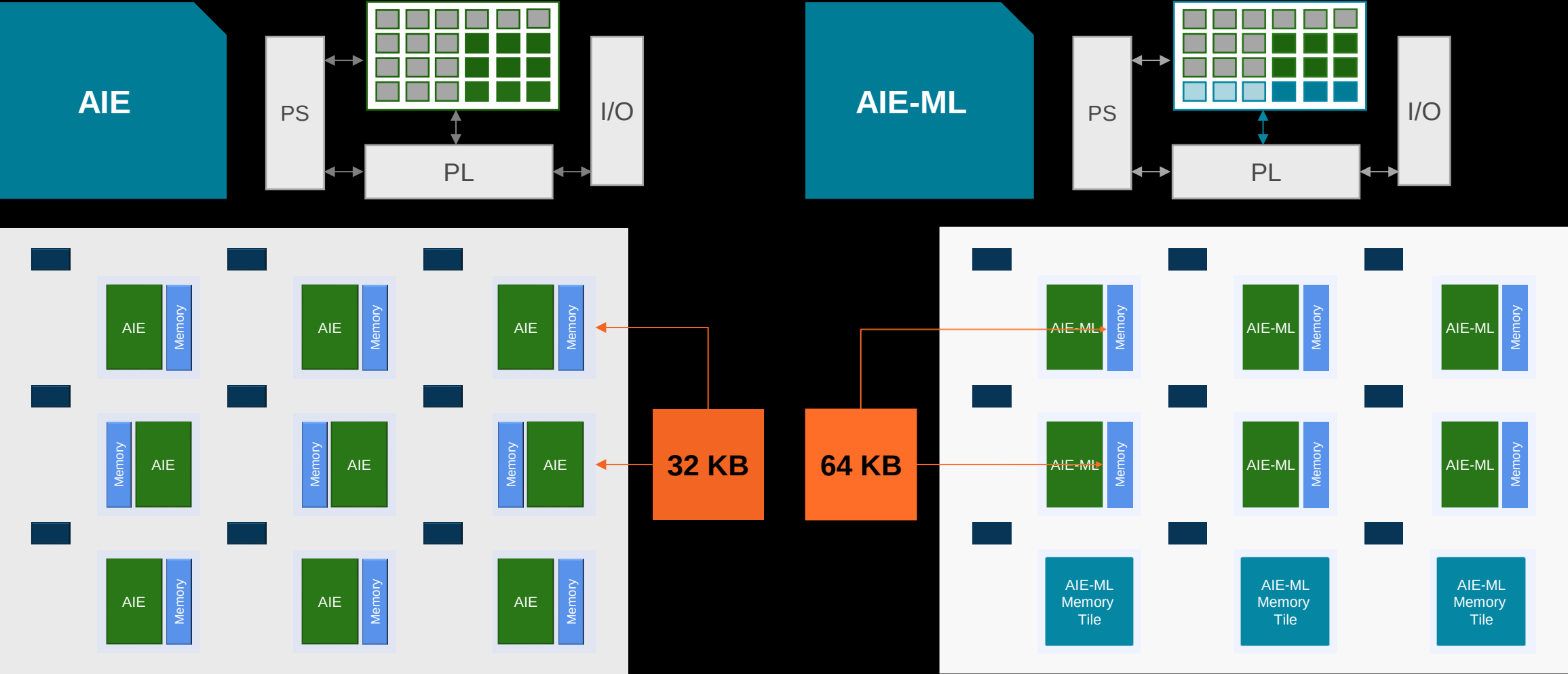
DIFFERENT AI ENGINE ARCHITECTURES: AIE AND AIE-ML



 Memory Tiles

DIFFERENT AI ENGINE ARCHITECTURES: AIE AND AIE-ML

MEMORY



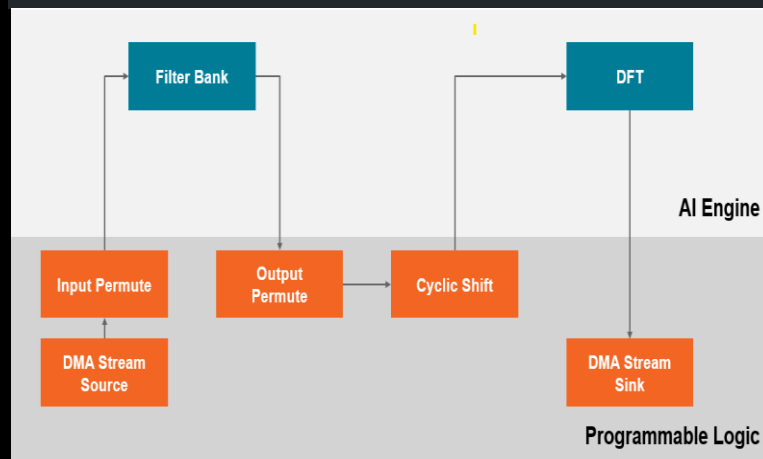
Top-Level Design Overview

How to program AI Engines for DSP

Think DSP/Vector Processor Methodology



Partition the DSP System



Code in C or Model-Based Flows



The AI Engine Development Team

Your team:



System Architect / Top-Level DSP Algorithm Specialist



RTL Designer

The AI Engine development flow is in the domain of PL and RTL developers—requiring:

- Programming of SIMD array of vector processors working in parallel
- Lower level optimizations (e.g., FIFO sizing)
- Data Flow Management via Graph C-Code (special C constructs)
- Memory Utilization Management
- Managing Interaction between AI Engine + PL
- Not equivalent to writing sequential C++ code

Design Rules of Thumb

What DSP designs are candidates for AI Engines?

Vectorizable Algorithms

FIR, FFT, Matrix Ops

+

High Throughput Requirements

GSPS vs. low MSPS

+

High Compute Complexity

High Point FFTs

High Tap Count FIR

Large Matrices

Design Flow

Map the system into functions that will be implemented in
AI Engine and in the PL
(Application Mapping & Design Partitioning)

Whiteboard
Mapping



Implement the functions in the AI Engine Tile Array: Use a
combination of library functions, AI Engine API,
and intrinsic coding

Software Tools



Instantiate and **interconnect** multiple AI Engine tiles – GRAPH
C-code

Software Tools



Compile & Simulate the AI Engine and
the AI Engine + PL design

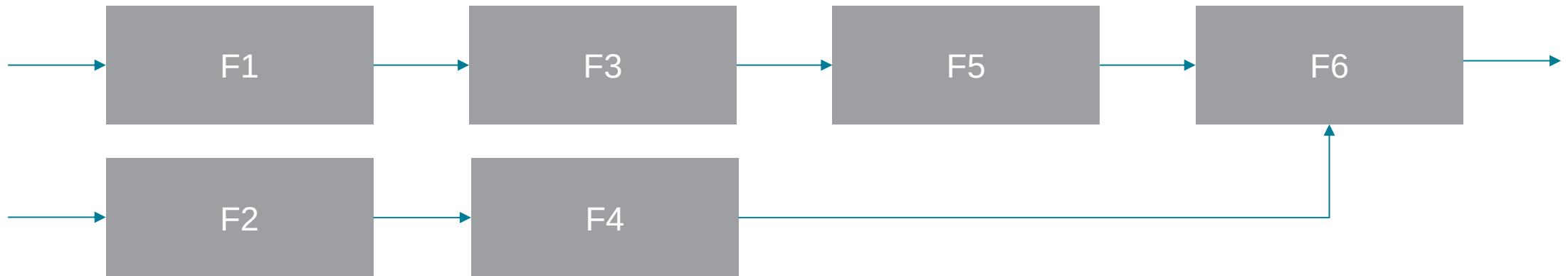
Software Tools

System Mapping Overview

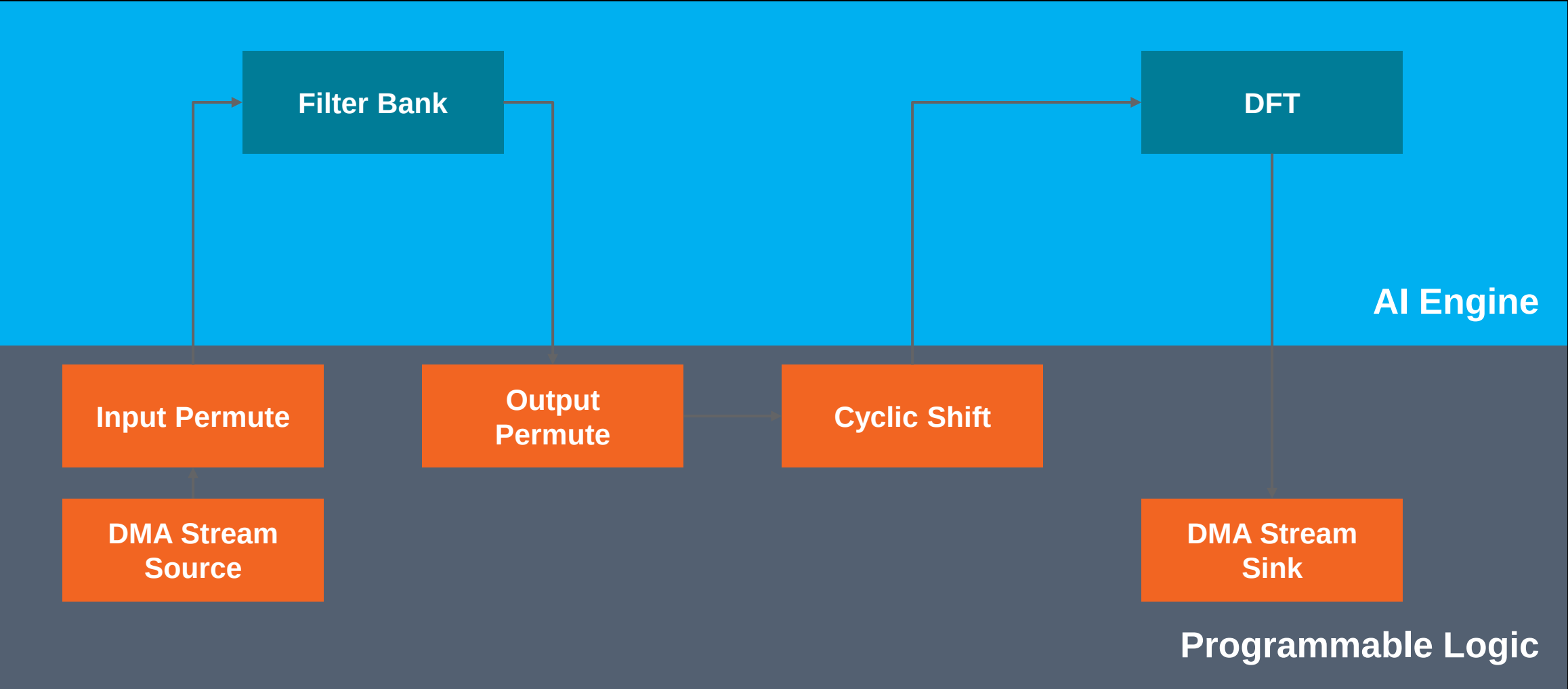
Top-Level DSP System



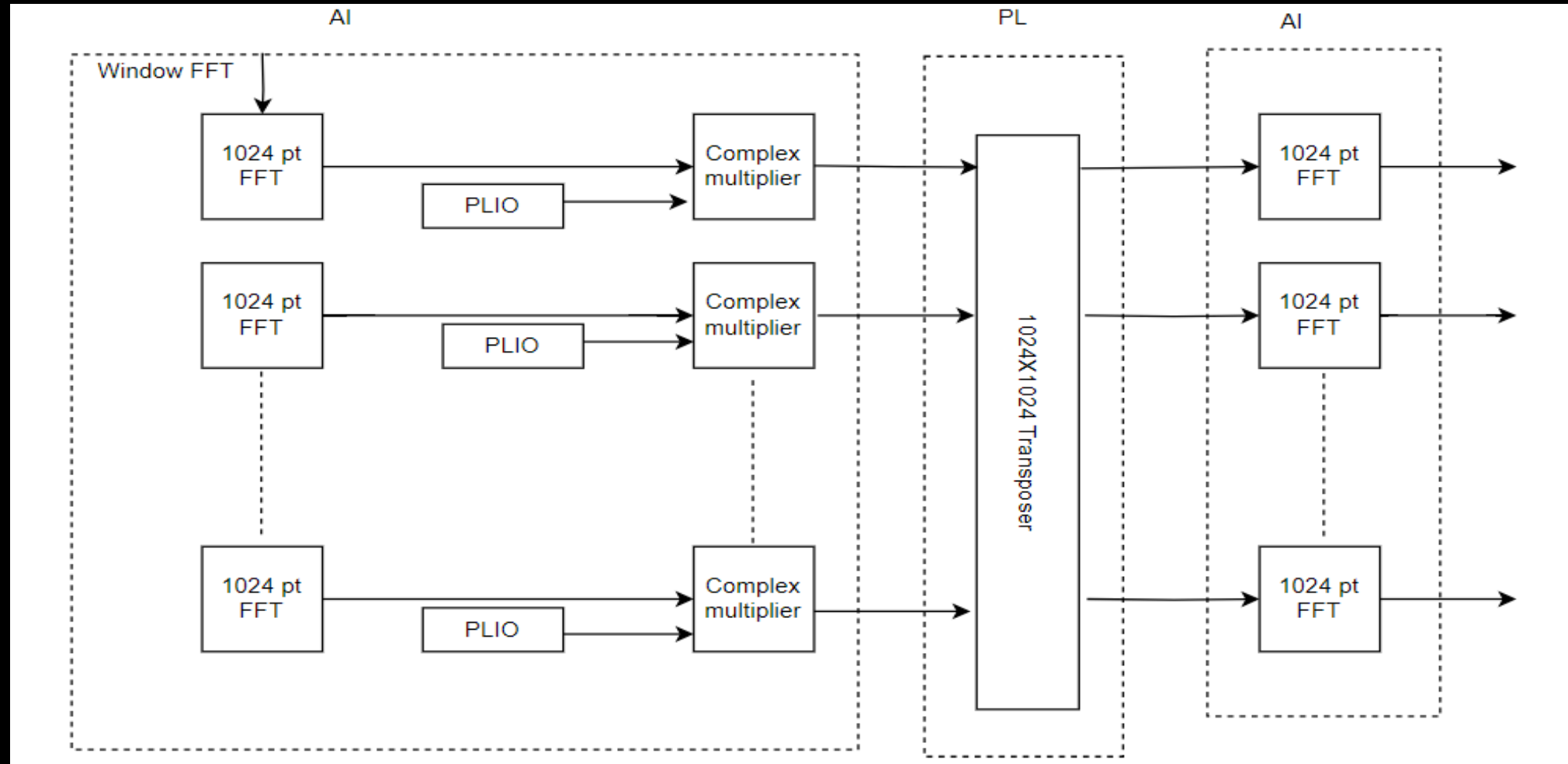
Map system into functions



System Mapping: Channelizer



System Mapping: 1 Million Point FFT



Compute Intensive

Not Compute Intensive

Compute Intensive

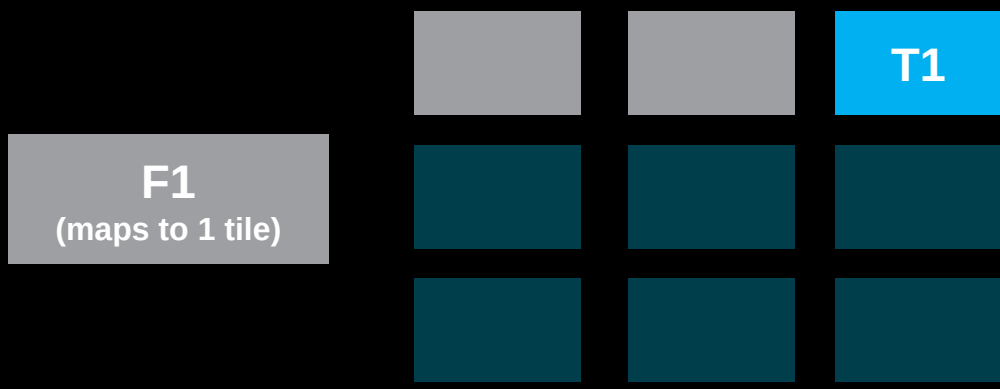
Mapping to AI Engines

Mapping the Functions into Tiles within the AI Engine Array

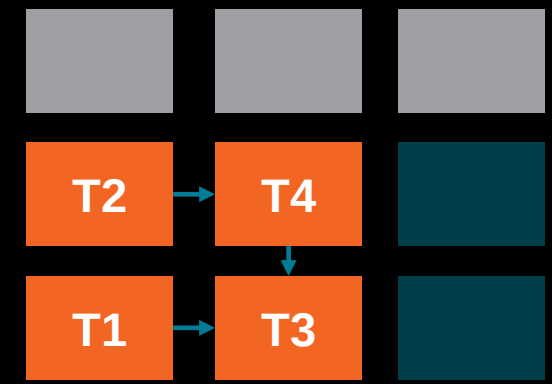
Considerations when choosing numbers of tiles for a particular function:

- 1.) Compute
- 2.) I/O Bandwidth
- 3.) Memory
- 4.) Storage

Note: Also possible to map multiple functions on a single AI Engine tile depending on runtime ratios.



F2
(maps to 4 tiles)



Increasing Function Complexity

Programming Methods

Option 1

AMD Vitis™ DSP Library Function Call

```
dsplib::fft::dit_1ch::fft_ifft_dit_1ch_g
```

- Easiest to use / Parameterizable
- Fastest Development

Option 2

Vectorized Programming with AI Engine APIs

```
aie::vector<int8_t, sizeTileA> A0 = aie::loa
aie::vector<int8_t, sizeTileA> A1 = aie::loa
```

- Customizable
- Faster Development

Option 3

Vectorized Programming with Intrinsics

```
acc0 = mac16(acc0, Bbuff, 0, 0x00000000, 8,
0x3120, Abuff0, 0, 0xCC884400, 2, 0x3210);
acc1 = mac16(acc1, Bbuff, 32, 0x00000000, 8,
0x3120, Abuff0, 0, 0xCC884400, 2, 0x3210);
```

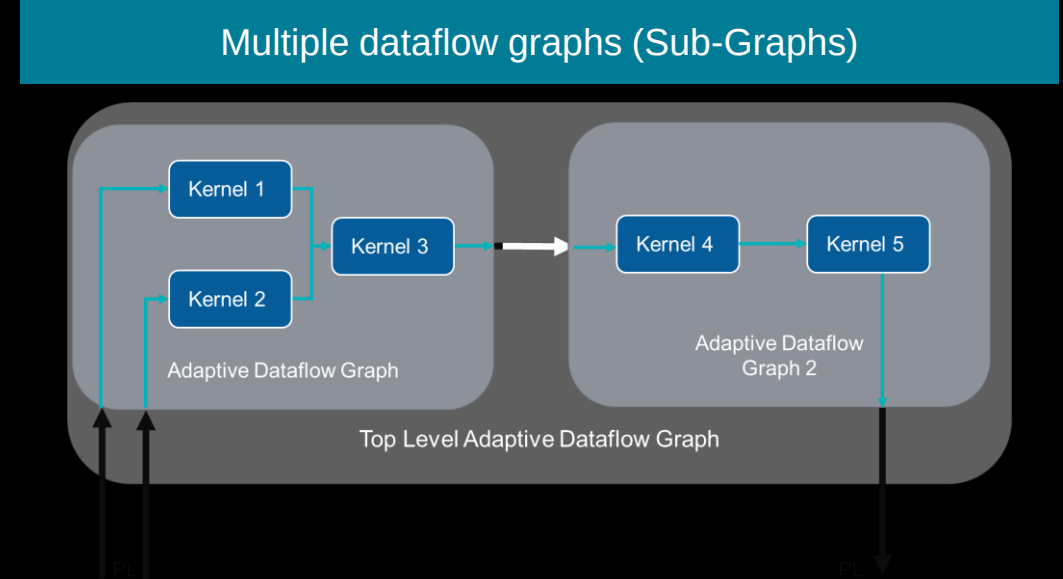
- Full, Low-Level Customization
- Longthiest Development

Graph Coding

- C++ code that describes a network with multiple AI Engine kernels connected by data streams or windows
- Can interact with the PL, global memory, and/or the host processor
- Compiled and executed using the AMD Vitis™ AIE compiler (free license as part of AMD Vitis tool)

```
myGraph() {
    kernel1 = kernel::create(fir_16taps_symm);
    kernel2 = kernel::create(fir_23taps_symm);
    connect< window<128> > net0 (in, kernel1.in[2]);
    connect< window<128> > net1 (kernel1.out[0], kernel2.in[0]);
    connect< window<128> > net2 (kernel2.out[0], out);
    connect< parameter>      (NSamples, async(kernel1.in[0]));
    connect< parameter>      (Coefficients, async(kernel1.in[1]));

    source(kernel1) = "kernels/Kernel_1.cc";
    source(kernel2) = "kernels/Kernel_2.cc";
    runtime<ratio>(kernel1) = 0.1;
    runtime<ratio>(kernel2) = 0.1;
}
```



AI Engine Simulation

```
myGraph() {  
  
  kernel1 = kernel::create(fir_16taps_symm);  
  kernel2 = kernel::create(fir_23taps_symm);  
  connect< window<128> > net0 (in, kernel1.in[2]);  
  connect< window<128> > net1 (kernel1.out[0], kernel2.in[0]);  
  connect< window<128> > net2 (kernel2.out[0], out);  
  connect<parameter>      (NSamples, async(kernel1.in[0]));  
  connect<parameter>      (Coefficients, async(kernel1.in[1]));  
  
  source(kernel1) = "kernels/Kernel_1.cc";  
  source(kernel2) = "kernels/Kernel_2.cc";  
  runtime<ratio>(kernel1) = 0.1;  
  runtime<ratio>(kernel2) = 0.1;  
}
```



AMD Vitis™
AI Engine Compiler



X86 Simulator

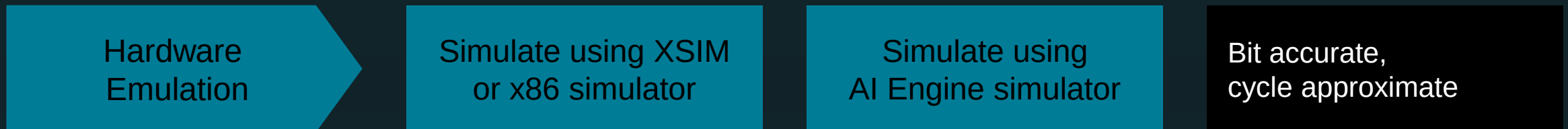
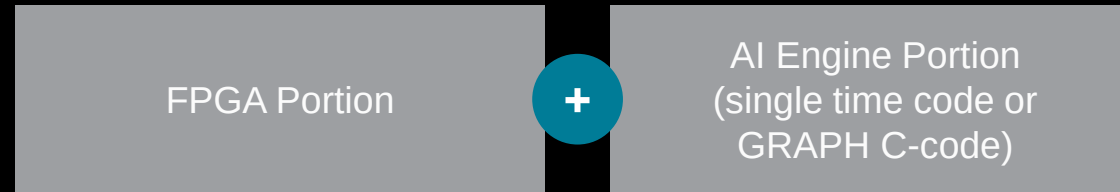
Fast - Functional Simulation



AI Engine Simulator

*Bit Accurate – Cycle Approximate
Simulation*

Co-Simulation: AI Engine + PL

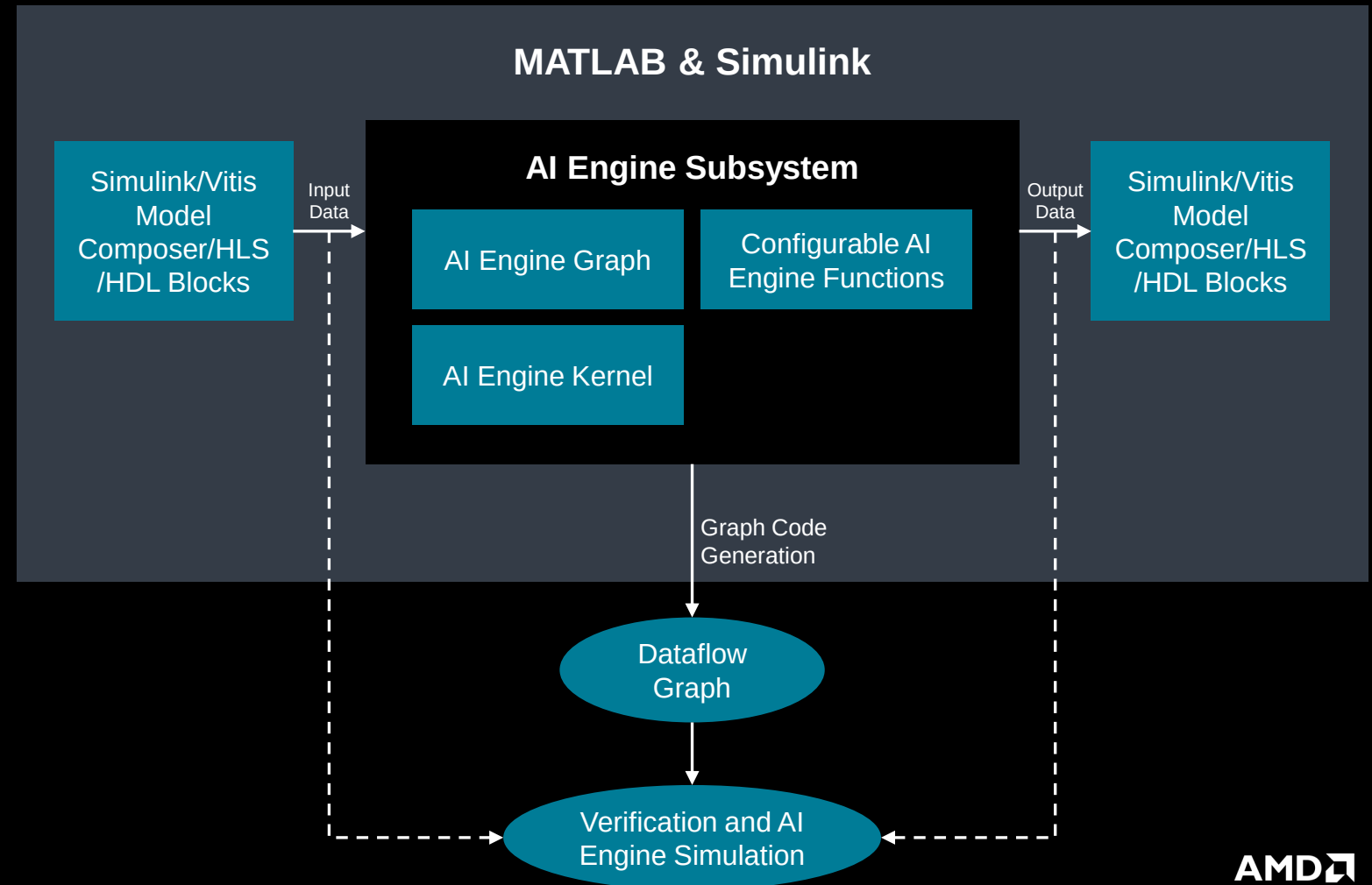


Model-Based Flow: Vitis Model Composer

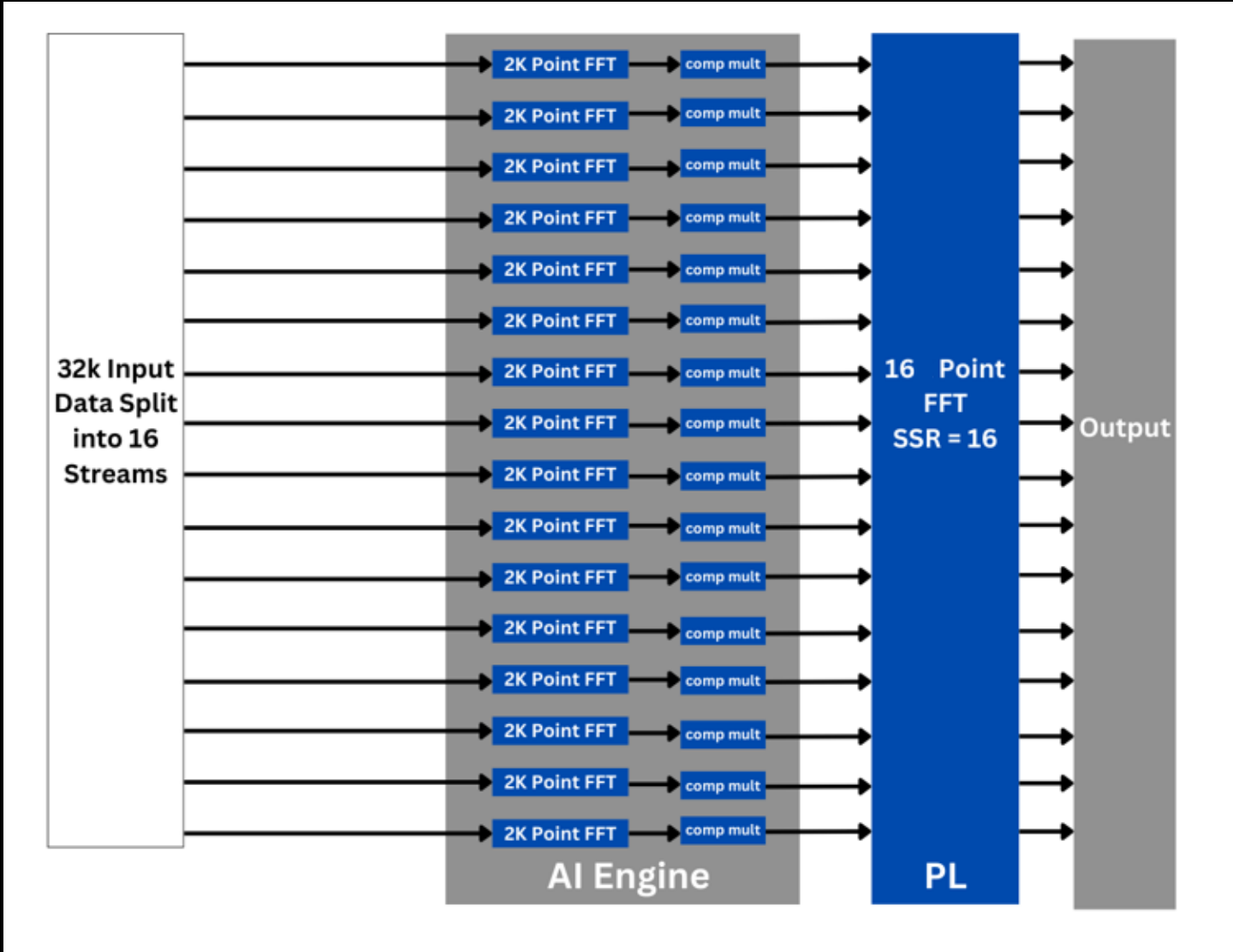
AMD Vitis™ Model Composer is a model-based design tool that enables rapid design exploration within the MathWorks MATLAB® and Simulink® environment.

Advantages for AI Engine-DSP Design Flow:

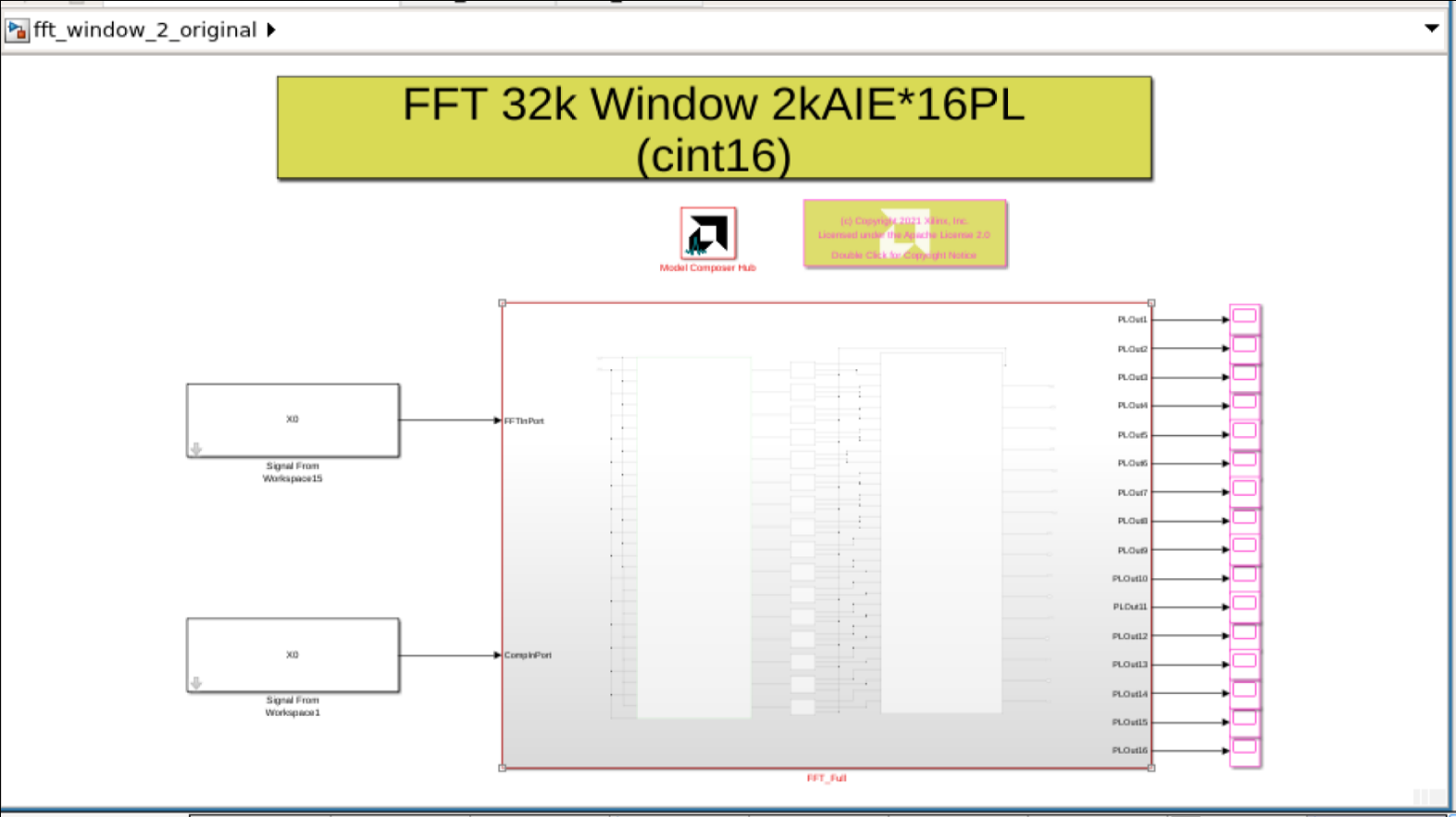
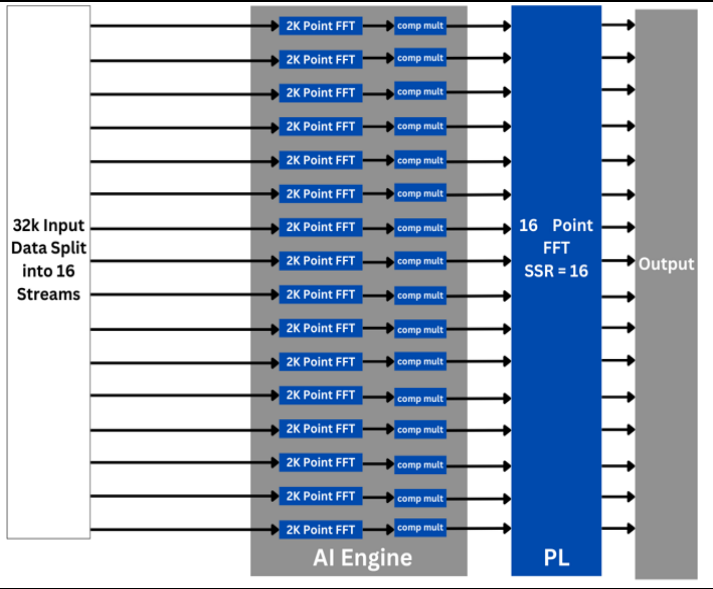
- **Automatically generate Graph C code for AI Engines**
- **Import HDL blocks in AI Engine designs**
- **Heterogeneous simulation with robust test bench capabilities from Simulink**



Model Composer: Design Example



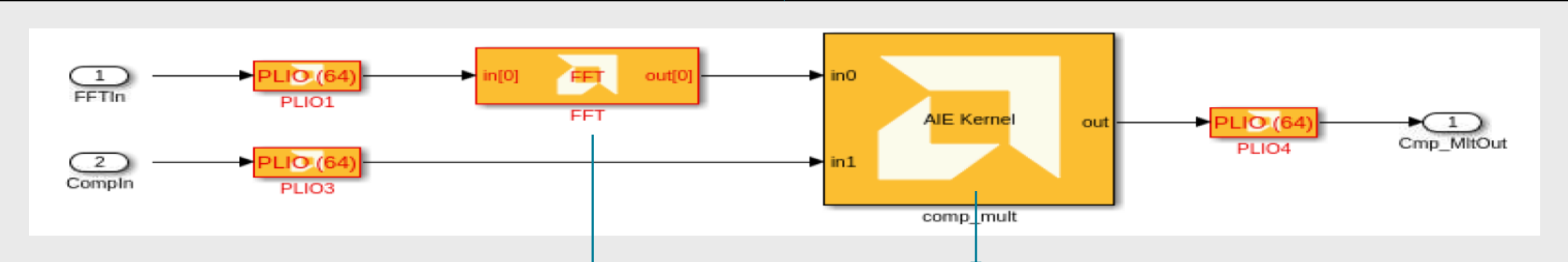
Model Composer: Top-Level Tools View



Model Composer: FFT Kernels in AI Engine



Vitis™ Model Composer

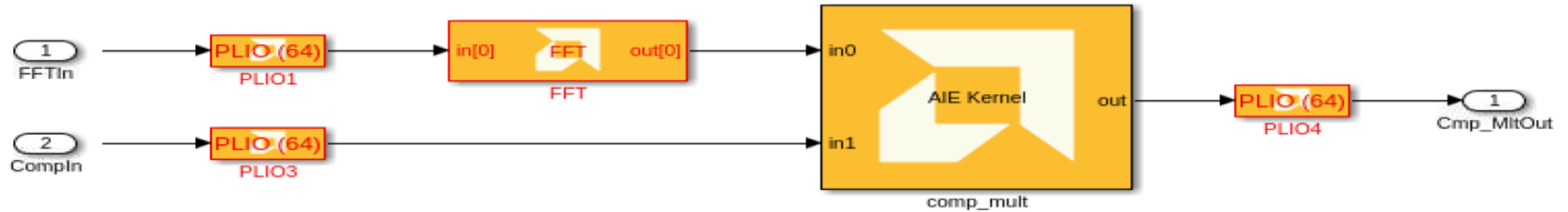


AMD Vitis™ DSP
Library Function Block

```
Editor: proj0001/Workspace/Dhananjay/Copy of DJ_256_FFT_cint16/complex_mult/ad...
adder.cc
1  #include <ad.h>
2  #include "aie_api/aie.hpp"
3  #include "aie_api/aie_adf.hpp"
4  #include "aie_api/util.h"
5  #include "adder.h"
6
7  #define ACC_SHIFT 0
8  void aie_adder(input_window_cint16 * in0, input_window_cint16 * in1, output_window_cint16
9  //runtime iteration(comp_mult.getkernel()) = 0.0;
10
11
12  cint16 *restrict pA1=(cint16 *)in0->ptr;
13  cint16 *restrict pB1=(cint16 *)in1->ptr;
14  cint16 *restrict pC1=(cint16 *)out->ptr;
15  for(unsigned i=0; i<128; i++)
16  chess_prepare_for_pipelining
17  {
18      aie::vector<cint16, 16> mul1 = aie::load_v<16>(pA1);pA1 += 16;
19      aie::vector<cint16, 16> mul2 = aie::load_v<16>(pB1);pB1 += 16;
20      //
21      // vhwacc48 acc;
22      aie::accum<acc48, 16> res = aie::mul(mul1,mul2);
23      auto res = aie::mul(mul1,mul2);
24
25
26      aie::store_v(pC1, res.to_vector<cint16>(15)); pC1 += 16;
27  }
28
29
```

AMD Vitis™ AI Engine API

Model Composer: Generating Graph Code



Generated Graph C-Code

```

#include "FFT_AIE.h"

// instantiate ADF graph
FFT_AIE mygraph;

// initialize and run the dataflow graph
#ifdef __AIESIM__ || defined(__X86SIM__)
int main(void) {
    mygraph.init();
    mygraph.run();
    mygraph.end();
    return 0;
}
#endif
  
```

Get Started with GitHub: AIE DSP Library Overview

Overview of Functions

FIR

- Interpolator / Decimator
- Halfband
- Symmetric / Asymmetric
- P/Q Resampler

TRANSFORMS

- Stockham FFT / IFFT
- DFT
- Mixed-Radix (3/5)
- FFT 32-bit Twiddles

BLAS

- GEMM
- GEMV

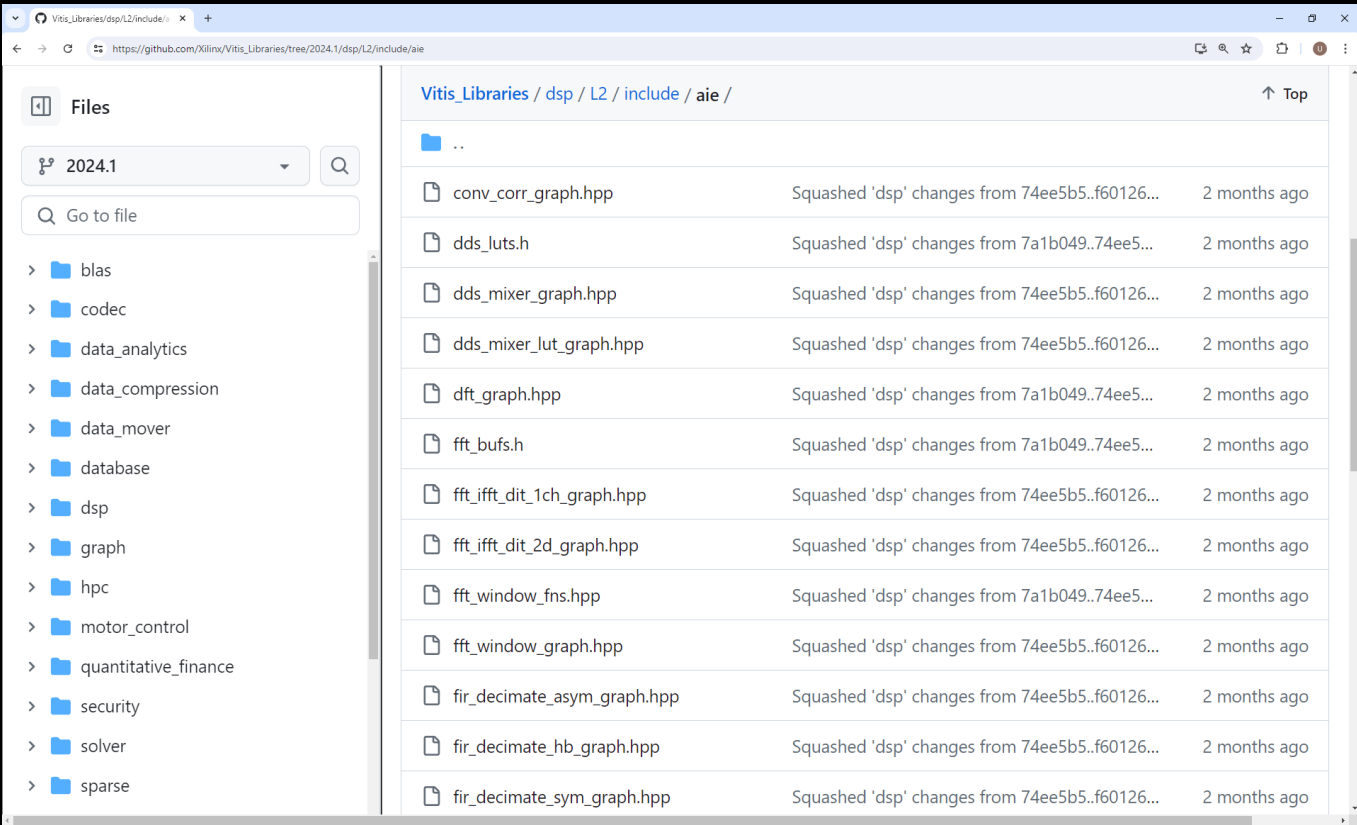
SOLVER
 $Ax = b$

- QRD
- Cholesky
- Matrix Pseudo Inverse
- Forward/Backward Subst.

VECTORIZED
COMPUTE

- Euclidean Distance
- Sorting
- Function Approximation
- Correlation

AMD GitHub – AIE DSP Libraries
www.github.com/Xilinx/Vitis_Libraries





Disclaimer & Attribution

Timelines, roadmaps, and/or product release dates shown in these slides are plans only and subject to change.

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. 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.

© Copyright 2024 Advanced Micro Devices, Inc. All rights reserved. AMD, the AMD Arrow logo, Alveo, EPYC, Instinct, Radeon, ROCm, Ryzen, Vivado, Virtex, Vitis, and other designated brands included herein are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies. PCIe, and PCI Express are trademarks of PCI-SIG and used under license. PID1978700