SWEST / ACRi 共同企画セッション

AIを加速するAI Engineのアーキテクチャ解説とプログラミング体験

ACRiルーム副室長 / ザイリンクス株式会社（AMD-Xilinx）

安藤 潤

アウトライン

- ACRiルームアカウント登録のお願い
- 自己紹介
- AI Engineアーキテクチャ解説
  - Versal Architecture AI Engine Technical Module
- ハンズオン

ACRi

ACRiルームアカウント登録のお願い

ACRiとは

- アダプティブコンピューティング研究推進体
  - https://www.acri.c.titech.ac.jp/
  - FPGAを盛り上げる会
  - 5大学、27社が参加（2022/7現在）
- 主な活動
  - ウェビナー、討論会、ハンズオン開催
    - https://acri.connpass.com/
    - https://www.youtube.com/channel/UCL15_5A9JKcVrmFUcMto6-Q
  - ACRiブログ
    - https://www.acri.c.titech.ac.jp/wordpress/
  - **ACRiルーム（無償FPGA利用環境）**
    - https://gw.acri.c.titech.ac.jp/wp/

ACRiルーム

- アカウント登録のお願い
  - ハンズオンではACRiルームが提供するサーバーを利用します
  - ハンズオンに参加される場合は、8/26までにアカウントを申請し、
    ご使用になるPCからリモートデスクトップ接続できることを
    事前に確認してください（ハンズオン当日のサーバーの予約は不要です）

  - アカウント申請方法
    - https://gw.acri.c.titech.ac.jp/wp/manual/apply-for-account
  - サーバーの利用方法
    - https://gw.acri.c.titech.ac.jp/wp/manual/how-to-reserve
  - Windows 10からPower Shellを使う場合
    - https://www.acri.c.titech.ac.jp/wordpress/archives/12855

自己紹介

自己紹介

- 組み込みと私
  - 液晶ディスプレイコントローラ向けのデジタルIP開発
    - アルゴ、ソフト、RTL、FPGA、ES
    - LPプロセス、案件たくさん
  - 8051ファームウェア
    - I2C、SPI
  - ザイリンクスに組み込みエンジニア（候補）として入社
    ⇒ 現在はデータセンター向け製品担当

AI Engineアーキテクチャ解説

ハンズオン

ハンズオンの流れ

1. ACRiルームのサーバーにリモートデスクトップ接続
   - as101~as105をご利用ください（予約なしで利用できます）

2. チュートリアルをコピーして起動

```
$ cd /scratch/$USER
$ cp -a /tools/repo/acri-room/aie-tutorial .
$ cd aie-tutorial
$ ./start.sh
```

3. ハードウェアエミュレーション（hw_emu）を選択して進めてください
   （実機で動作させる場合は後日as005でお試しください）

AMD
XILINX

Versal® Architecture AI Engine Technical Module

Florent Werbrouck
AI Engine & DSP Technical Marketing Engineer

Agenda

- Introduction and Motivation
- Architecture Overview
- AI Engine (AIE) vs AI Engine-Machine Learning (AIE-ML)
- Software and Tools
- System Partitioning for Versal® ACAP
- AI Engine Power & XPE
- Summary

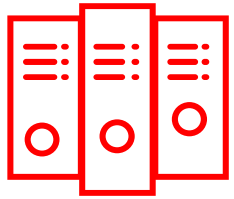
Introduction and Motivation

Motivation for AI Engine

AI IS EVERYWHERE

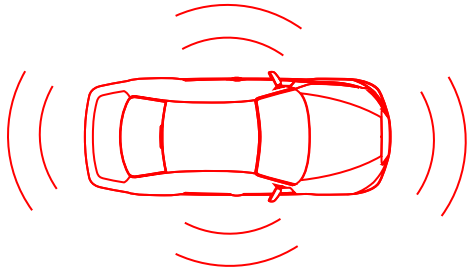
5G Wireless Radio

Data Center Workloads

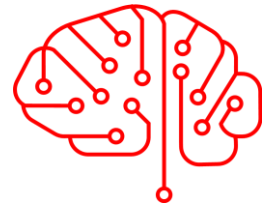


Smart Factories

ADAS & AD



Machine Learning



Technology Not Scaling

Applications More Demanding

Moore's Law / Dennard Scaling
Performance & Power Scaling
Traditional Single / Multi-core

Compute Density
Real Time Capability
Power Efficiency

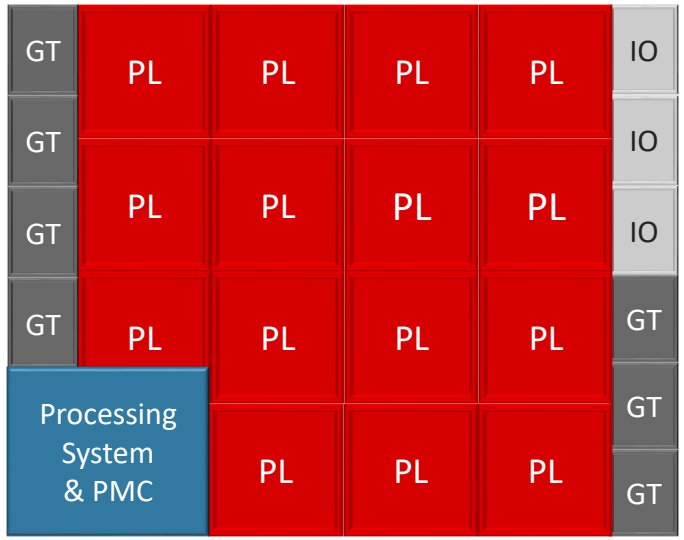
Dynamic Markets Require Adaptable Compute Acceleration

AMD XILINX

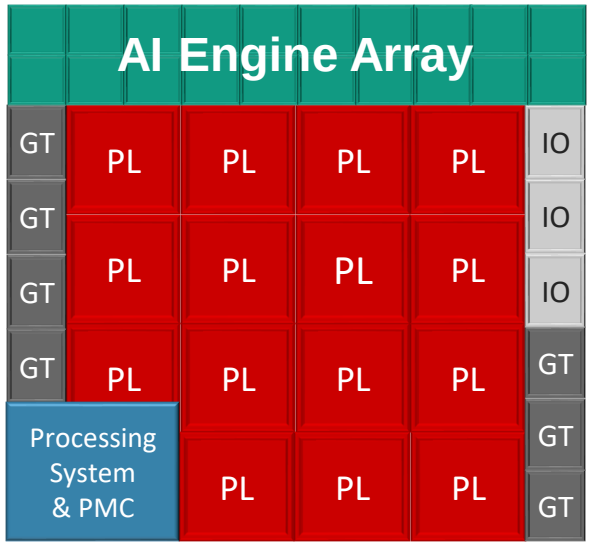
Conceptual Device with High Compute

- Applications need (1) cost reduction, (2) power reduction, (3) more compute, (4) more programmability
- How does AMD-Xilinx meet the demands of these evolving applications?

16nm Generation
(Zynq® UltraScale+™ MPSoC)



7nm Generation



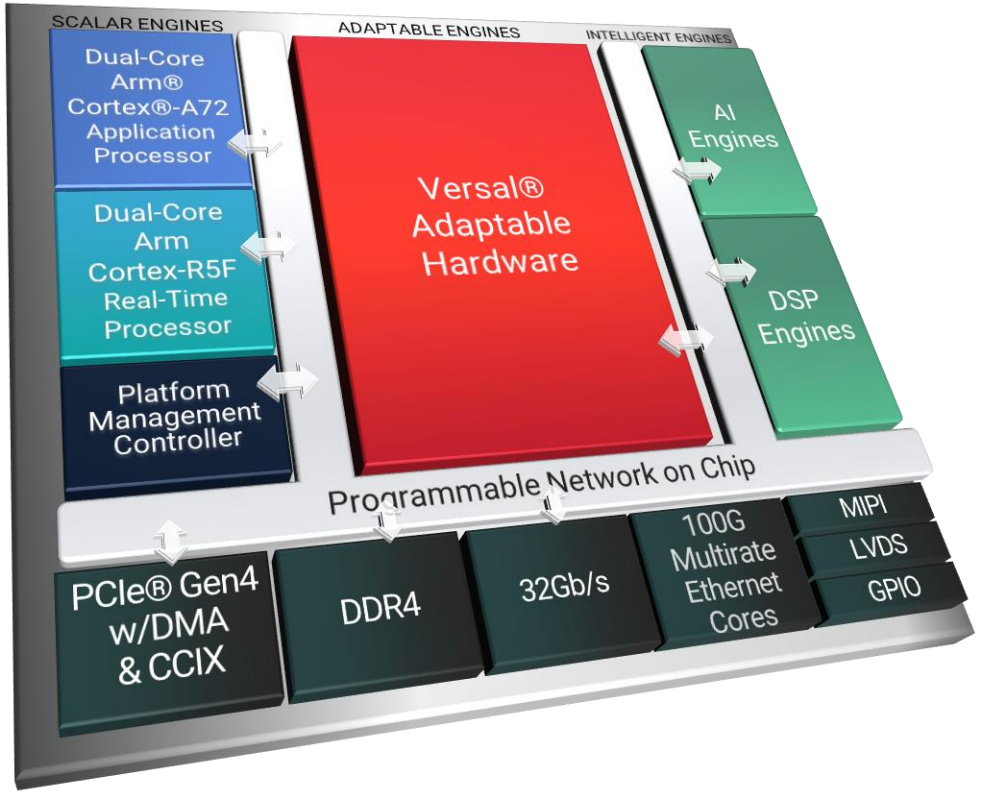
Goal: Increase Compute Density and Silicon Efficiency

AMD XILINX

Versal, Industry's First ACAP

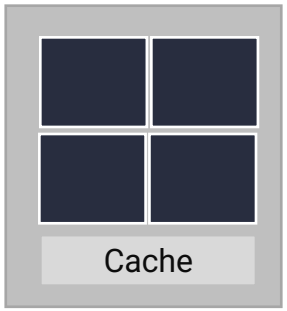
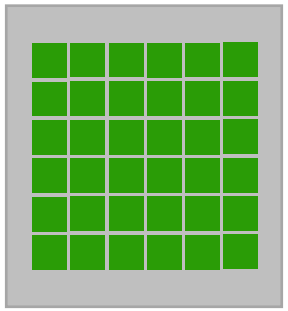
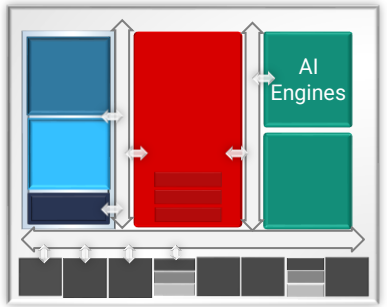
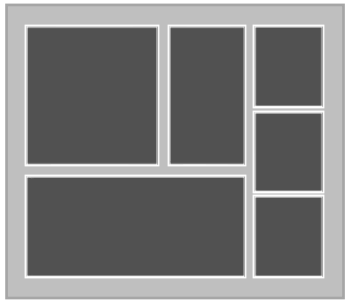
XILINX VERSAL®

- Heterogeneous Acceleration
- For Any Application
- For Any Developer



AMD XILINX

Delivering Adaptable Compute Acceleration

	CPU Single → Multi-Core	GPU (Parallel)	ACAP	ASIC
				
SW Programmable	✓	✓	✓	✓
HW Adaptable	–	–	✓	–
Workload Flexibility	✓	✓	✓	–
High Throughput and Low Latency	–	–	✓	✓
Device / Power Efficiency	–	–	✓	✓

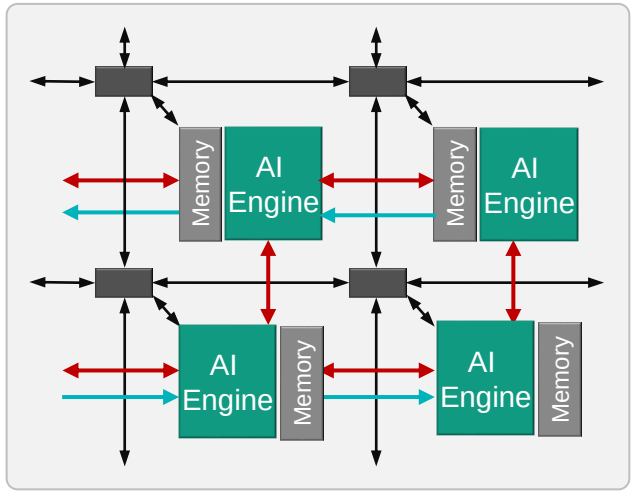
AMD
XILINX

Introducing the AI Engine

SW Programmable

Deterministic

Efficient



Target Applications

- 5G Wireless – Radio, Baseband
- Data Center - AI Inference
- ISM – Computer Vision, Genomics
- ADAS/AD – Embedded Vision
- A&D – Radar, Sonar

SIGNAL PROCESSING

ARTIFICAL INTELLIGENCE

CNN LSTM MLP

WIRELESS

COMPUTER VISION

Adaptable. Intelligent.

AMD XILINX

AI Engine

Hardened Compute, Memory, and Interconnect

Huge performance improvements vs. UltraScale+™ FPGAs

- Up to 8x compute density @ 40% lower power

1GHz+ VLIW / SIMD vector processors

- Versatile core for ML and other advanced DSP workloads

Massive array of interconnected cores

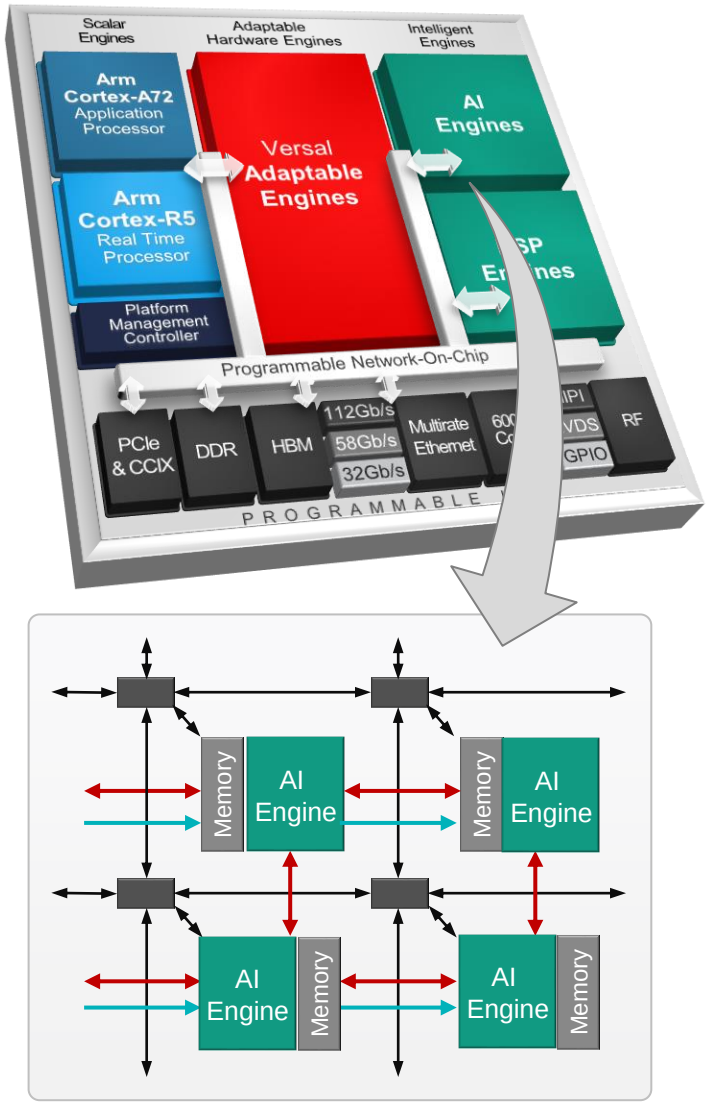
- Instantiate multiple tiles (10s to 100s) for scalable compute

Terabytes/sec of interface bandwidth to other engines

- Direct, massive throughput to adaptable HW engines
- Implement core application with AI for “Whole App Acceleration”

SW programmable for any developer

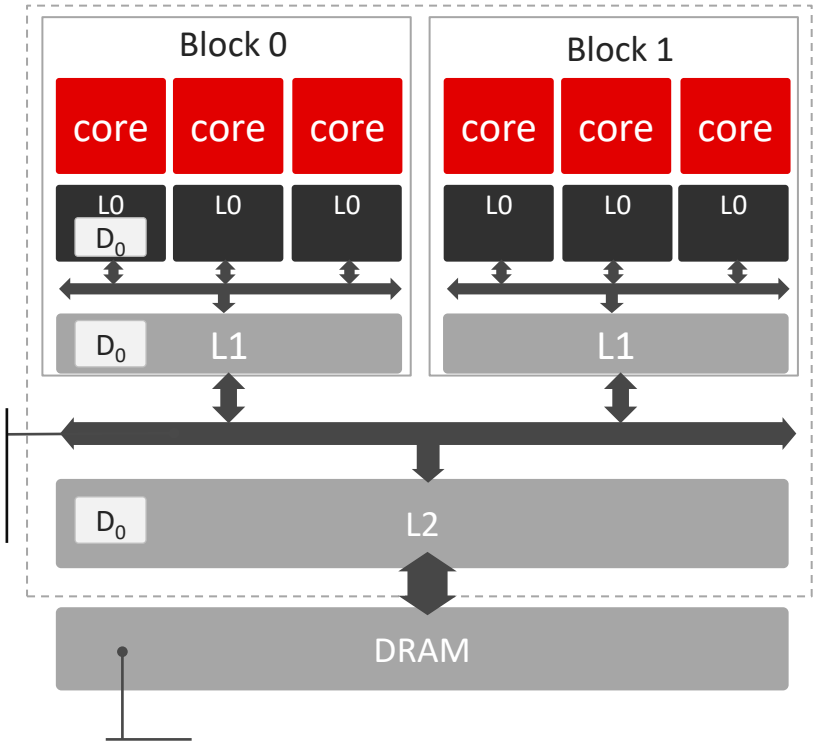
- C programmable, compile in minutes
- Library-based design for ML framework developers



AMD XILINX

AI Engine: AMD-Xilinx Reinvents Multi-Core Compute

Traditional Multi-core
(cache-based architecture)

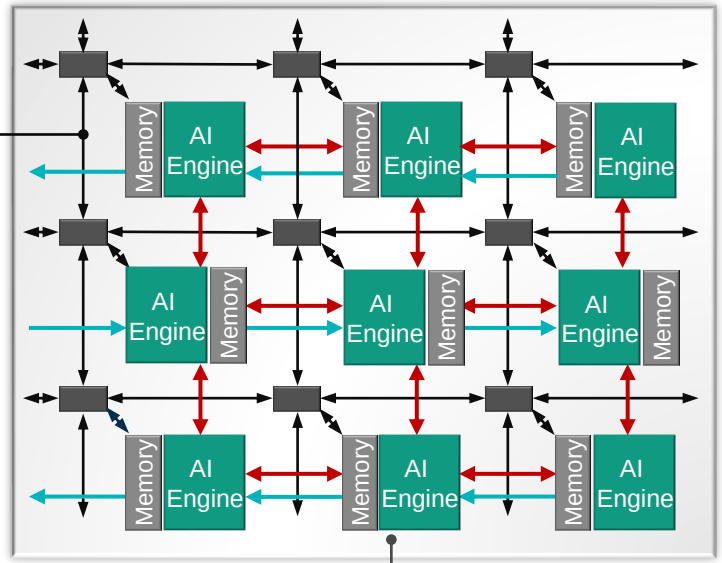


Fixed, shared Interconnect
- Blocking limits compute
- Timing not deterministic

Data Replicated
- Robs bandwidth
- Reduces capacity

AI Engine Array
(intelligent engine)

Dedicated Interconnect
- Non-blocking
- Deterministic



Local, Distributed Memory
- No cache misses
- Higher bandwidth
- Less capacity required

AMD XILINX

Software Programmable: Any Developer

1 Design

C/C++

Frameworks

mxnet TensorFlow Caffe

4G/5G/Radar Library | AI Library | Vision Library

2 Compile

AI Engine Compiler

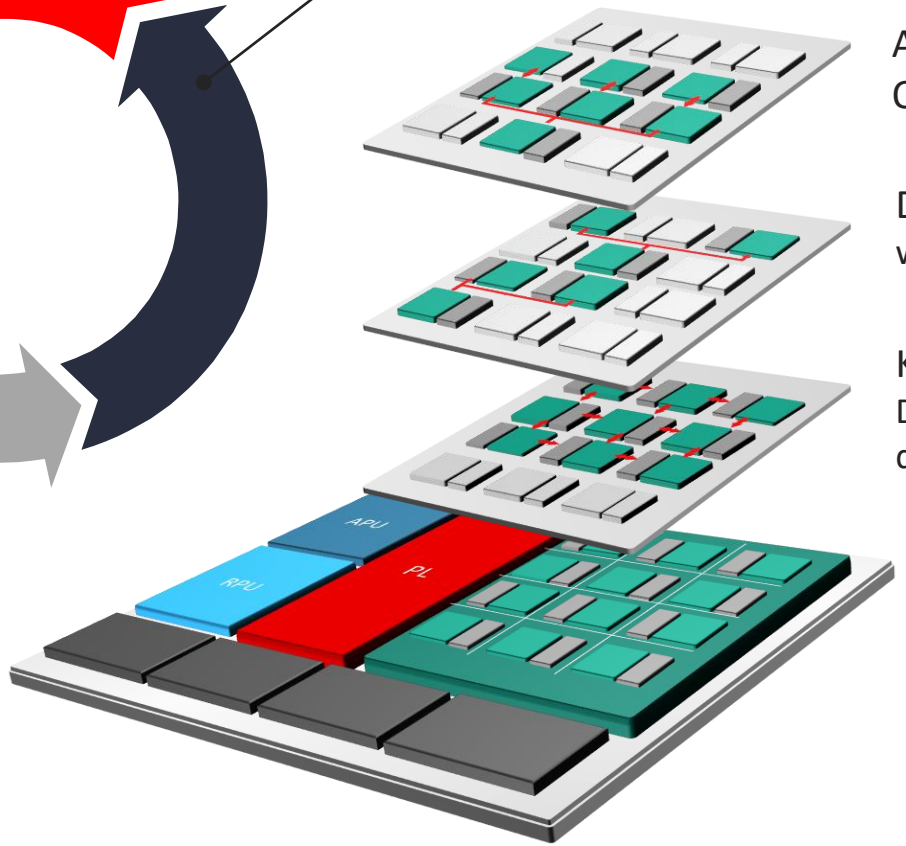
3 Run

Programming Abstraction Levels

Architecture
Overlay

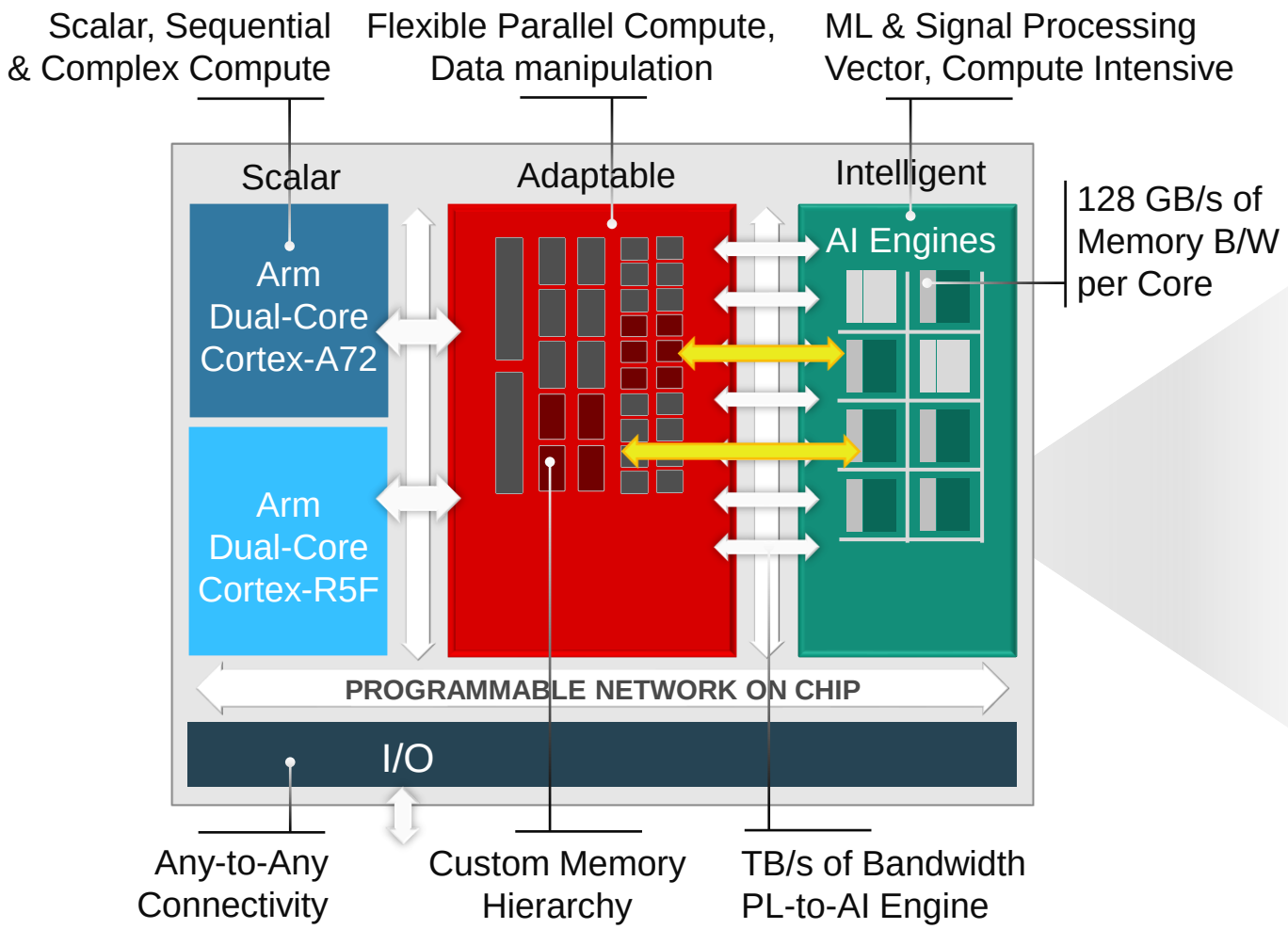
Data Flow
w/ Xilinx libraries

Kernel Program
Data Flow w/ user defined libraries



AMD XILINX

Hardware Adaptable: Accelerating the Whole Application



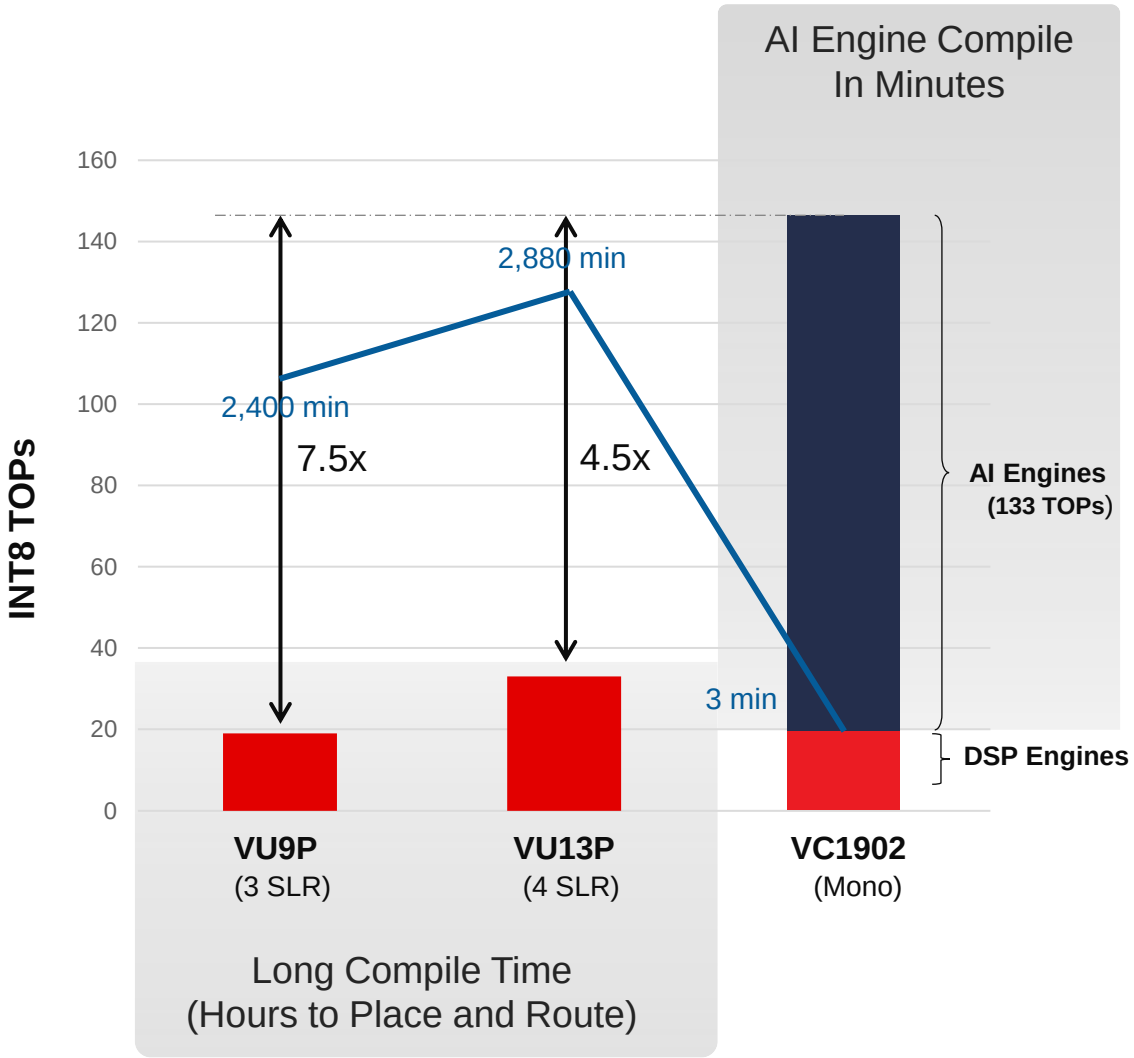
Heterogeneous Acceleration from Data Center to the Edge

- Video + AI
- Genomics + AI
- Risk Modeling + AI
- Database + AI
- Network IPS + AI
- Storage + AI

Delivering Deterministic Performance & Low Latency

AMD XILINX

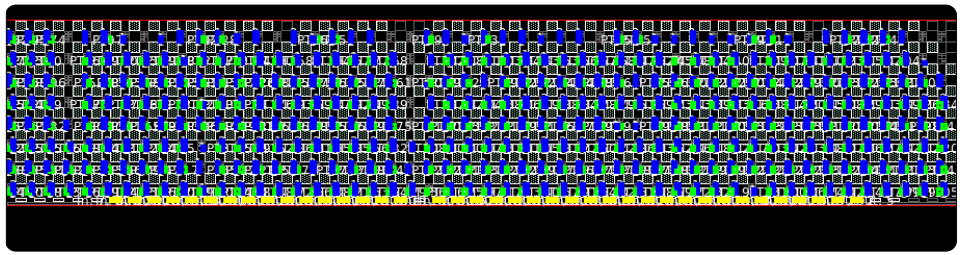
AI Engine Delivers Scalable Compute & Faster Compile



Example CNN Use-Case

- 343 AI Engine design (86% of VC1902)
- Ultrascale+™ P&R takes many hours
- AI Engine compiles in ~3 minutes

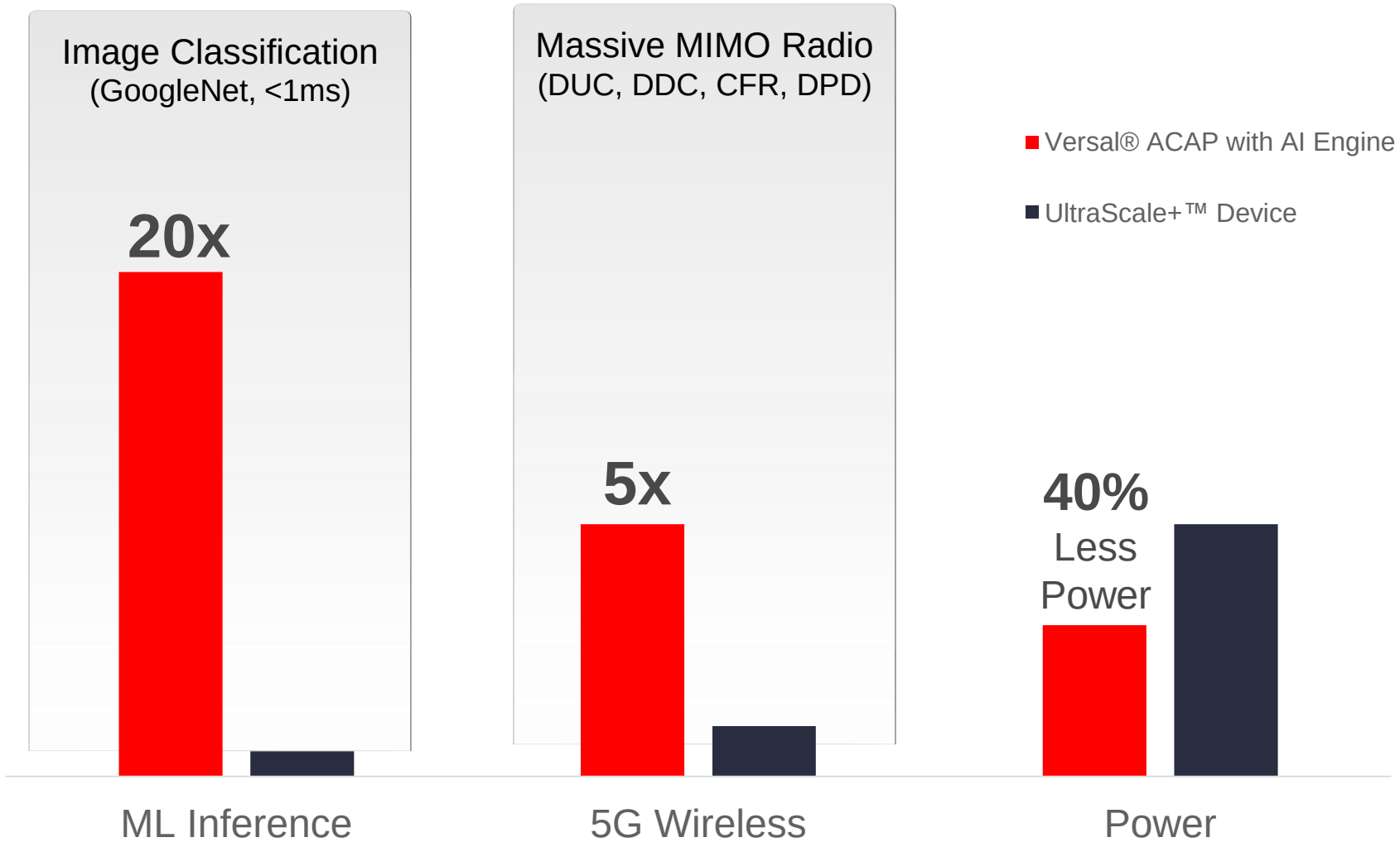
(Compile time for PL shell design not included)



343 AI Engines, 86% of VC1902

AMD XILINX

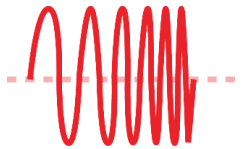
AI Engine Application Performance & Power Efficiency

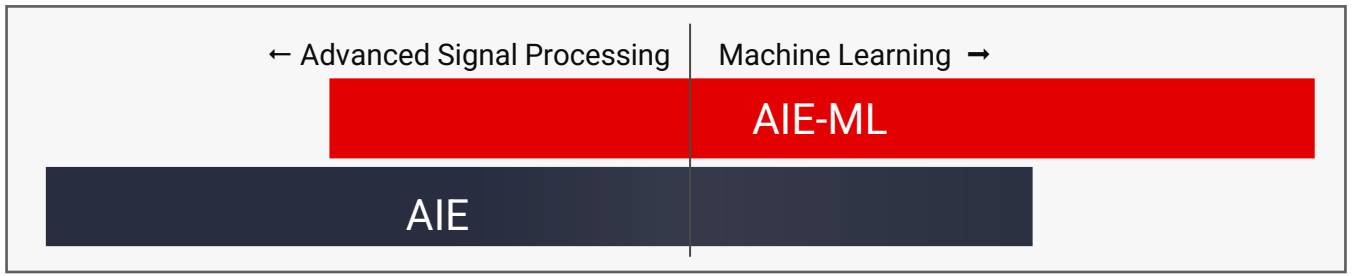


Architecture Overview

Multiple Versions Optimized for Specific Applications

BEAMFORMING,
RADAR PROCESSING,
HIGH PERF. COMPUTING



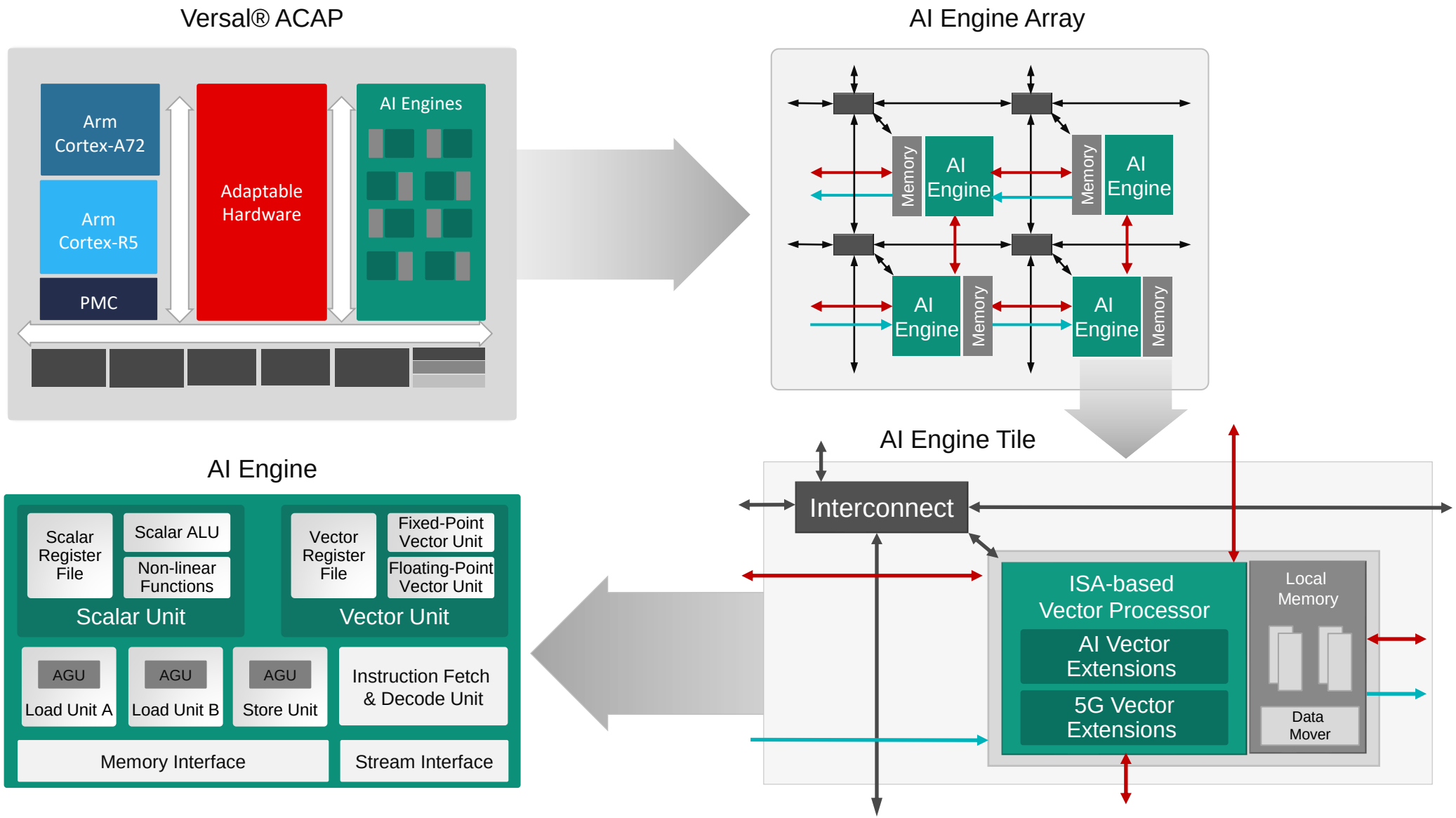


CNN, RNN, MLP

- AIE: Version Optimized signal processing AND ML
- AIE-ML: Optimized for ML inference applications
- In following slides, we refer to AI Engine as the technology (common to AIE and AIE-ML) and to AIE/AIE-ML as the versions of this technology

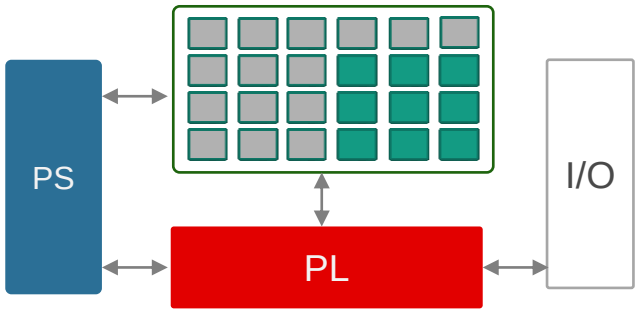
AMD XILINX

AI Engine: Terminology



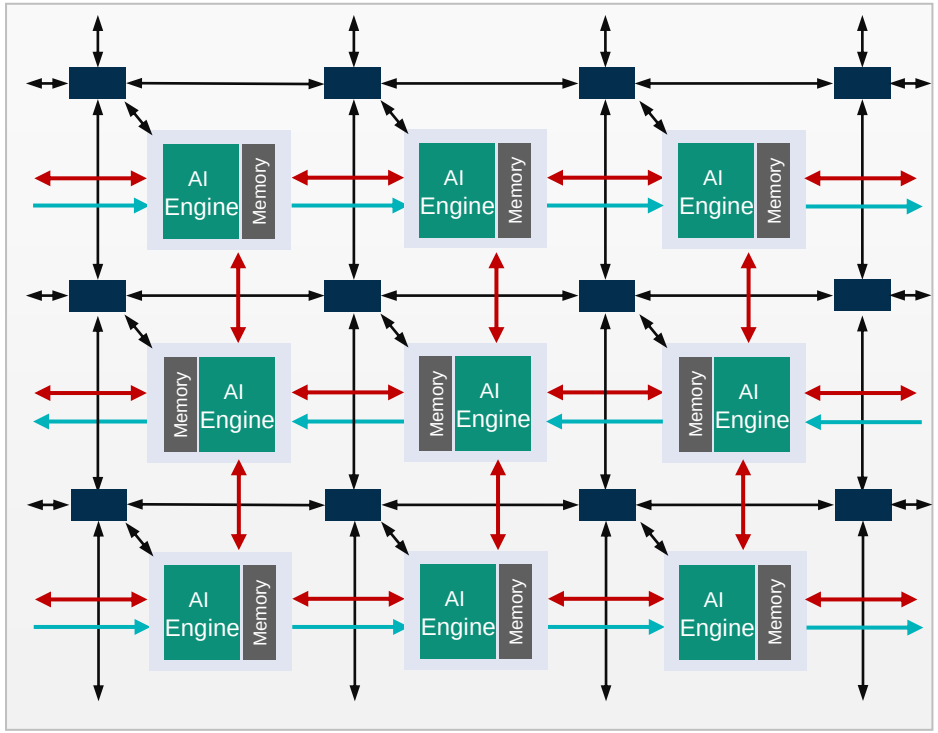
AMD XILINX

AI Engine: Array Architecture



Array of AI Engines Tiles

- Increase in compute, memory and communication bandwidth



Modular and scalable architecture

- More tiles = more compute
- Up to 400 per device
  - Versal AI Core VC1902 device

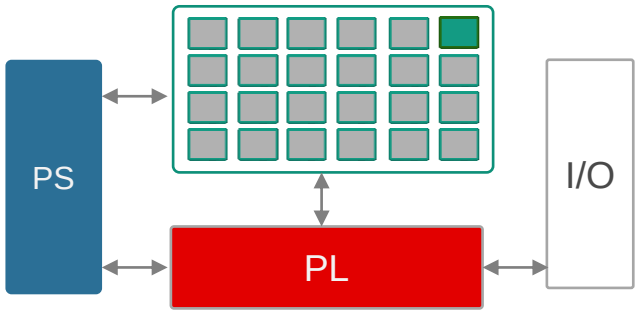
Distributed memory hierarchy

- Maximize memory bandwidth

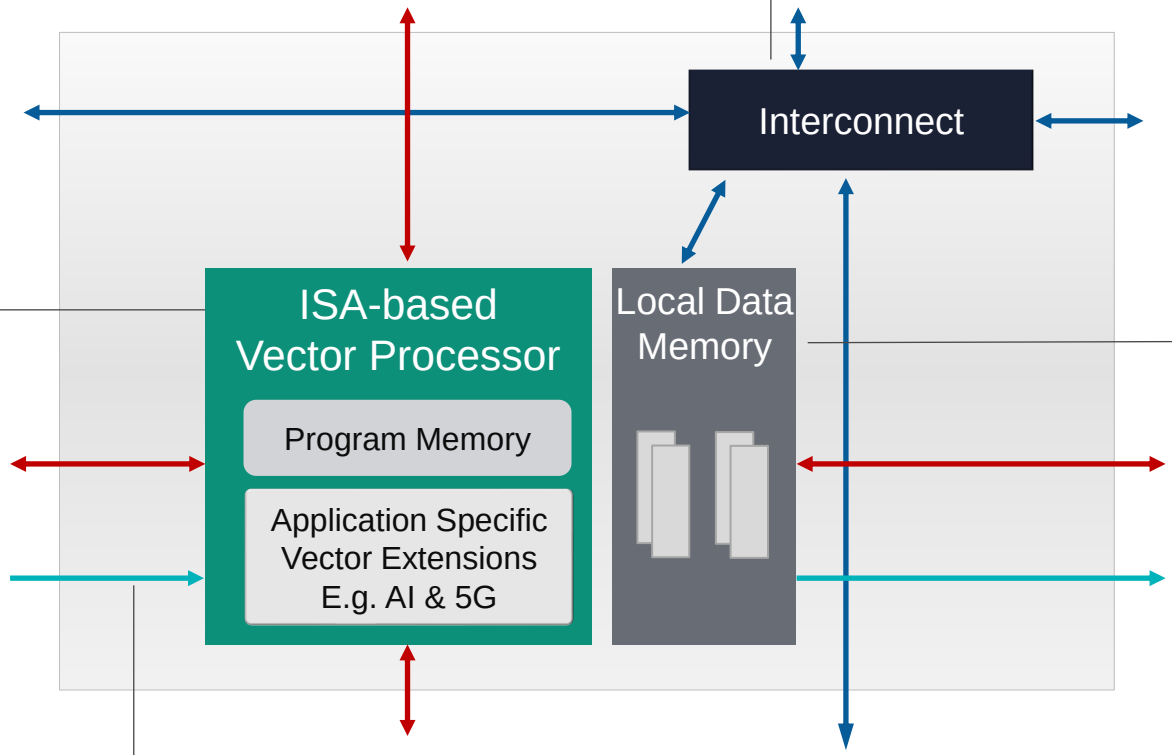
Deterministic Performance & Low Latency

AMD XILINX

AI Engine: Tile-Based Architecture



Non-Blocking AXI-Interconnect
Up to 200+ GB/s bandwidth per tile
Communicate between AI Engine Cores
Access to PL and memory in the PL



Memory Interface
Stream Interface
Cascade Interface

ISA-based
Vector Processor
Software Programmable
(e.g., C/C++)

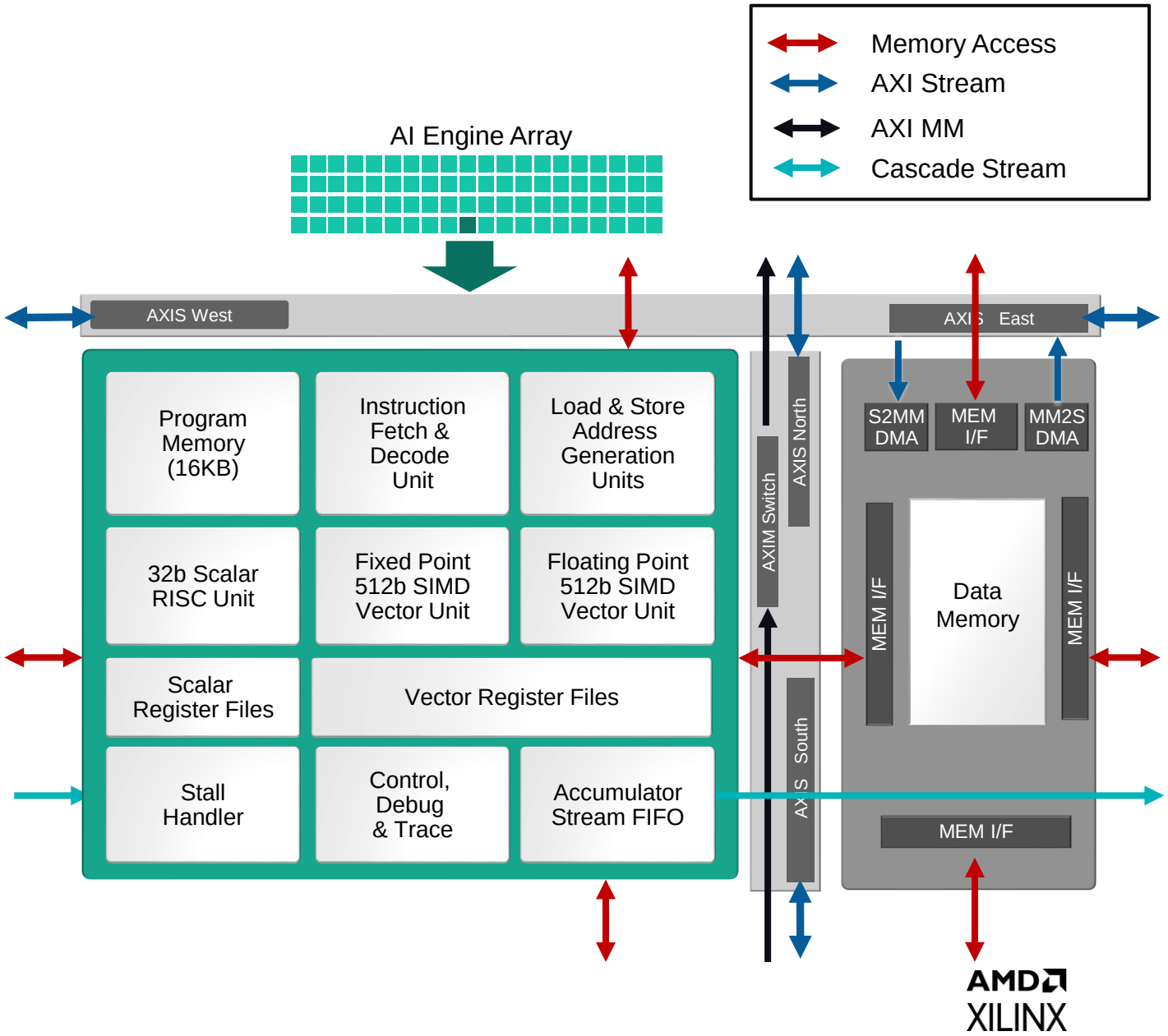
Local Memory
Multi-bank implementation
Shared across neighbor cores

Cascade Interface
Partial results to next core

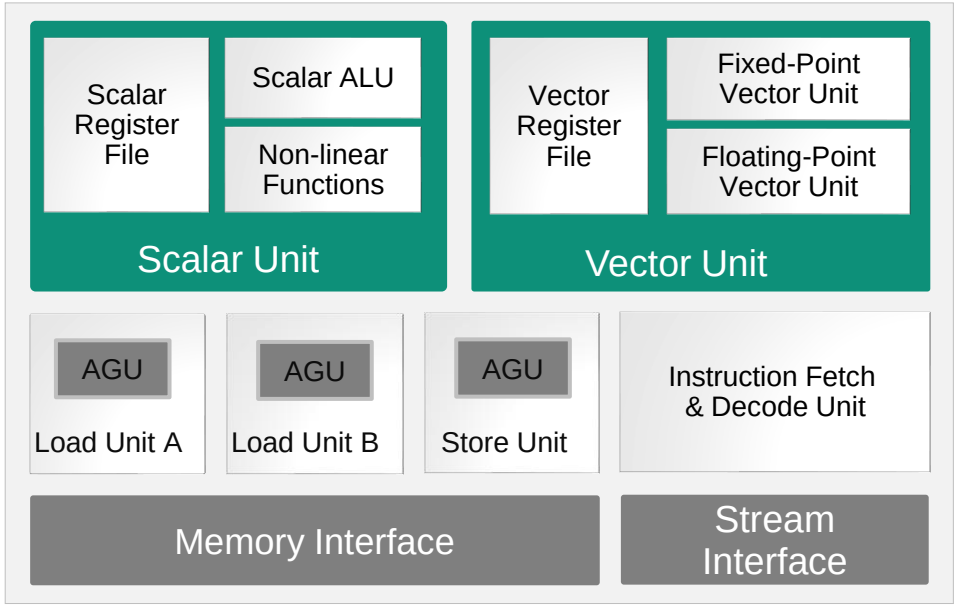
AMD
XILINX

AI Engine Tile

- AI Engine core
  - 512b SIMD vector units
    - Both fixed and floating point
    - 16KB program memory
  - 32b scalar RISC processor
  - 256-bit load/read (x2) and store/write units with individual AGUs
- Up to 256KB direct core memory access
  - 32KB local in AI Engine / 64KB local in AIE-ML
  - 32KB north, south, east or west in AI Engine / 64KB north, south and west in AIE-ML
- Streaming interconnects
  - AXI Memory Mapped (AXI-MM) switch
    - Configuration, control and debug
  - AXI-Stream crossbar switch
    - Routing N/S/E & W around the array
- Debug/Trace/Profile functionality



AI Engine: Highly Parallel Processor Core



Instruction Parallelism: VLIW

7 operations / clock cycle

- 2 Vector Loads / 1 Mult / 1 Store
- 2 Scalar Ops / Stream Access

Highly Parallel

Data Parallelism: SIMD

Multiple vector lanes

- 8 / 16 / 32-bit & SPFP operands (AIE)
- 4 / 8 / 16 / 32-bit & BFloat16 operands (AIE-ML)

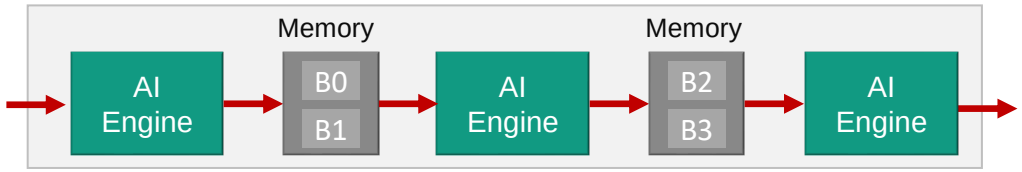
Up to 512 MACs / Clock Cycle per Core (INT 4 x INT 8) in AIE-ML

AMD XILINX

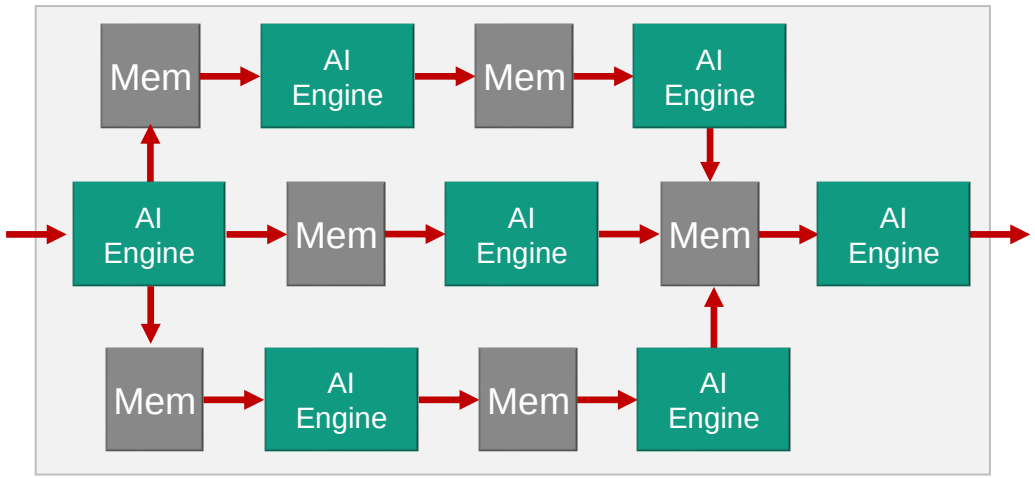
Data Movement Architecture

Memory Communication

Dataflow Pipeline



Dataflow Graph



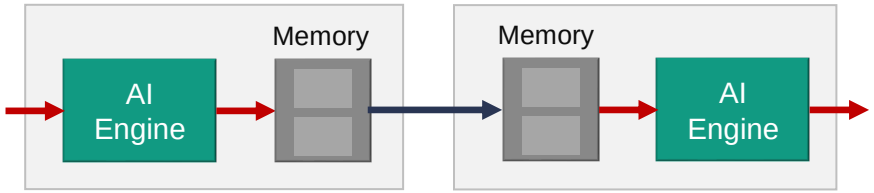
Memory Interface

Stream Interface

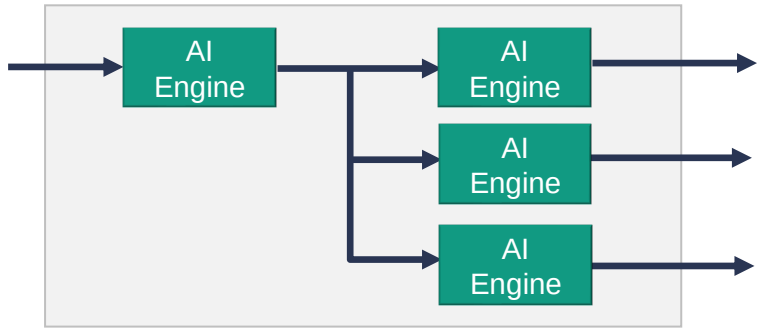
Cascade Interface

Streaming Communication

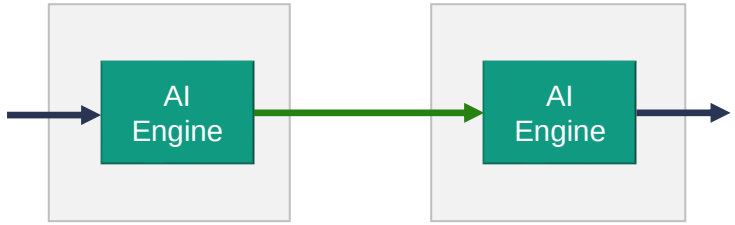
Non-Neighbor



Streaming Multicast



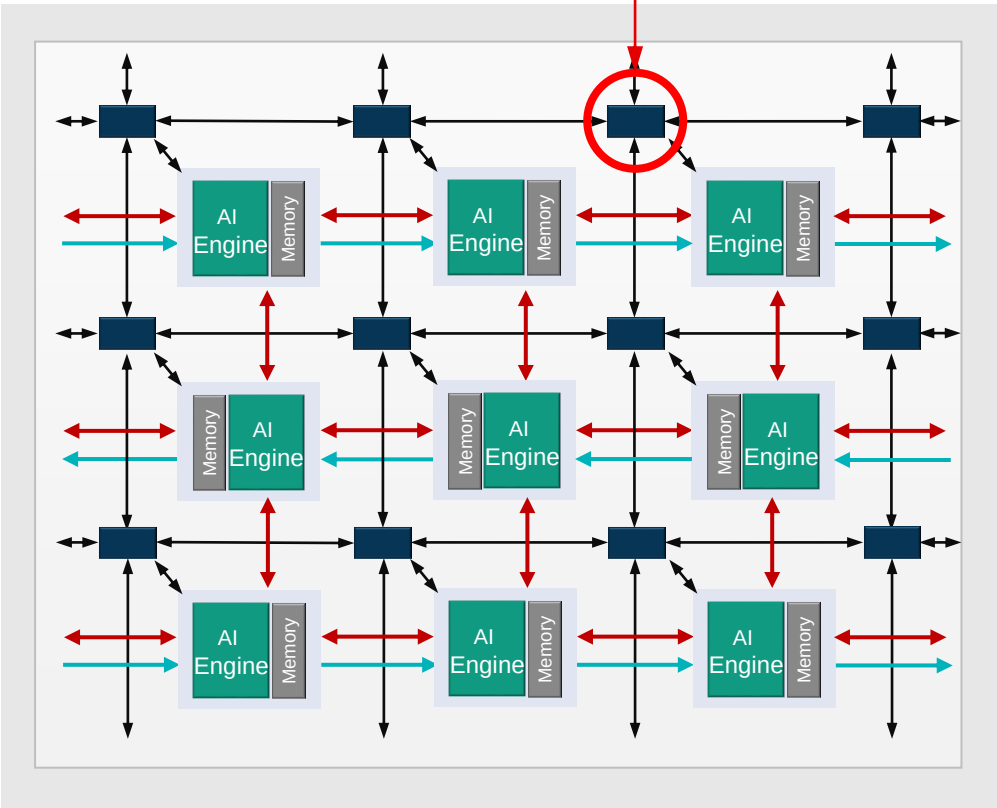
Cascade Streaming



AMD XILINX

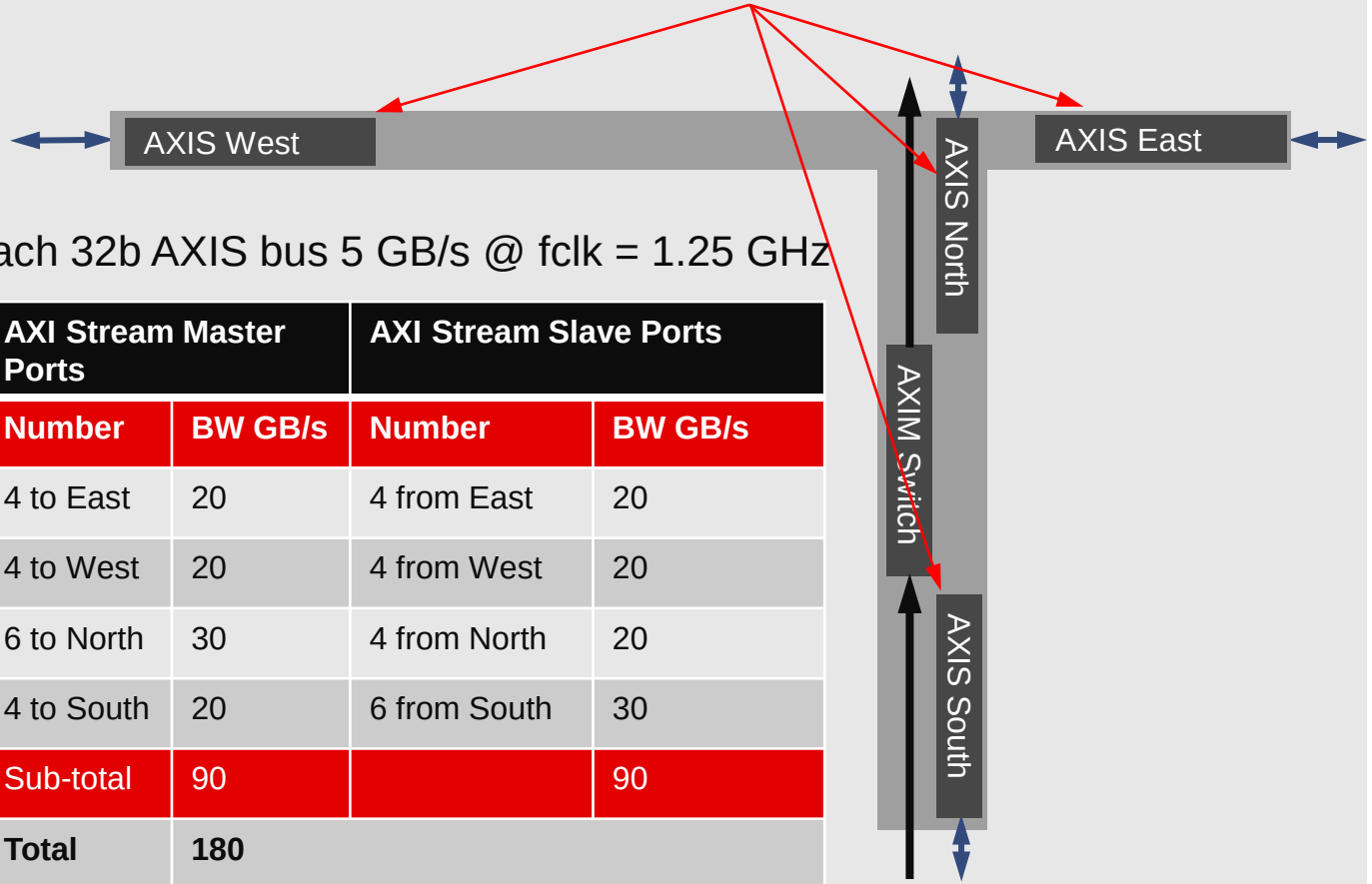
Tile Interconnect: AXIS

AI Engine Array



***Single* Tile Interconnect**

AXI stream switches



Each 32b AXIS bus 5 GB/s @ fclk = 1.25 GHz

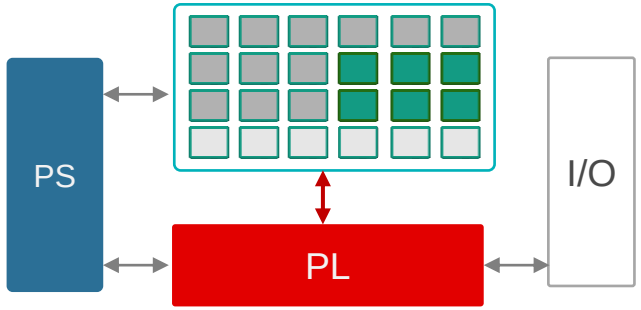
AXI Stream Master Ports		AXI Stream Slave Ports	
Number	**BW GB/s**	**Number**	**BW GB/s**
4 to East	20	4 from East	20
4 to West	20	4 from West	20
6 to North	30	4 from North	20
4 to South	20	6 from South	30
Sub-total	90		90
Total	**180**		

- Slowest speedgrade @ 1 GHz ➔ 144 GB/s total bandwidth per AI Engine tile

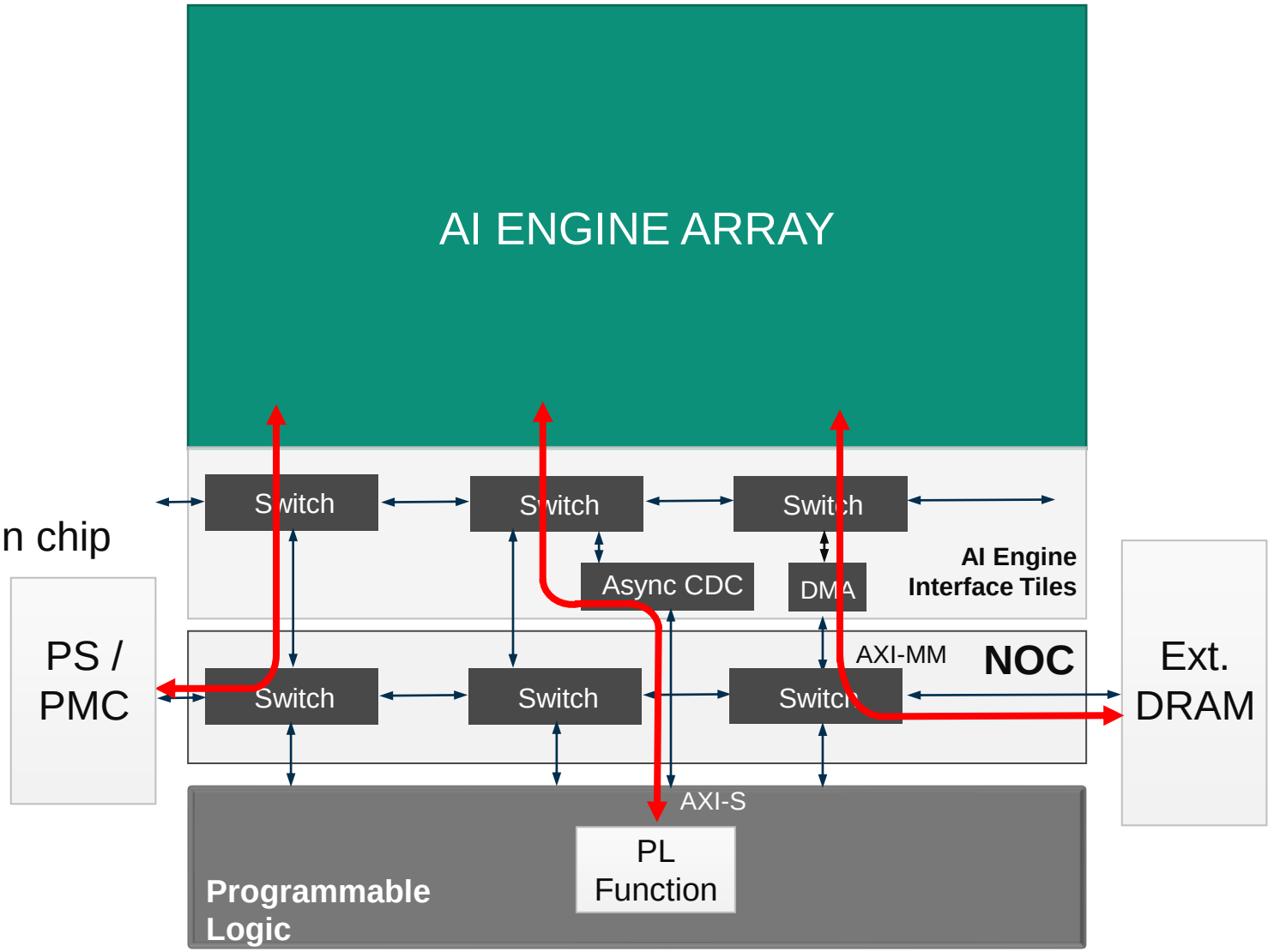
  Note: B/W is shared with adjacent AI Engine tiles; Traffic between non-adjacent tiles affects the available B/W

AMD XILINX

AI Engine Integration with Versal® ACAP

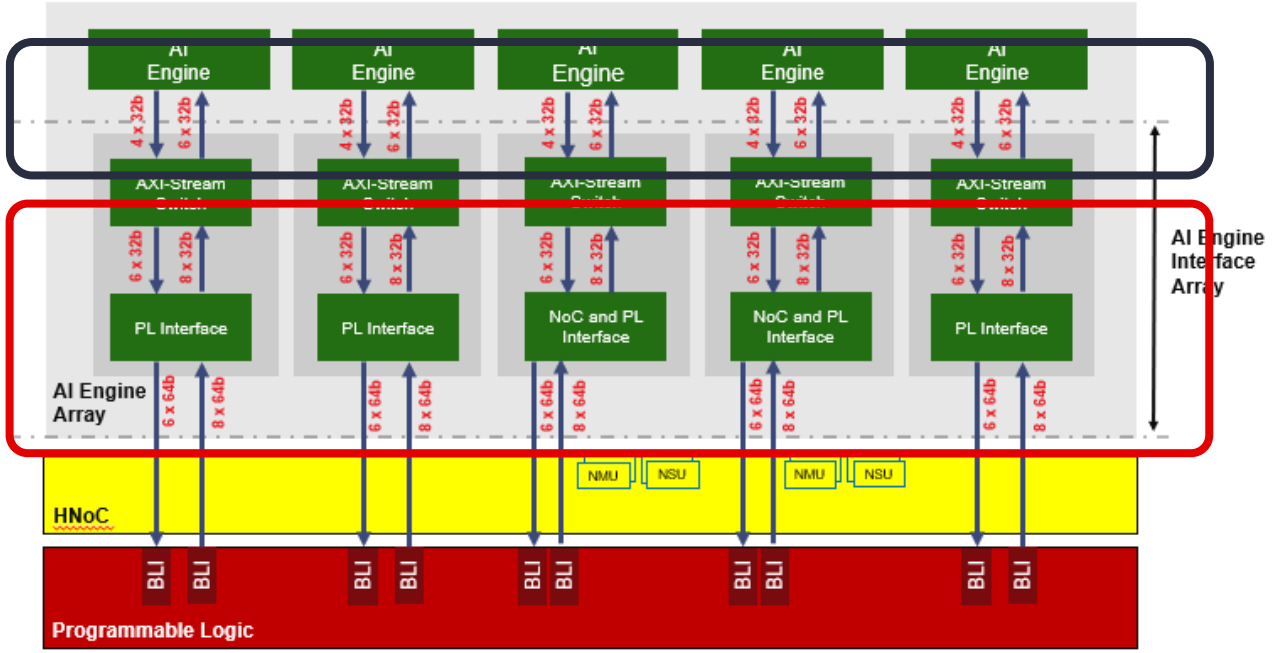


- TB/s of Interface Bandwidth
  - AI Engine to programmable logic
  - AI Engine to programmable network on chip
- Leveraging NoC connectivity
  - PS manages Config / Debug / Trace
  - AI Engine to DRAM (no PL req'd)



AI Engine to PL Interface

- AXI4-Stream switches in the PL interface tile directly communicate with the PL
  - Handle most of the data movement to/from the AI Engine array
  - Configurable bit widths (32b/64b/128b)



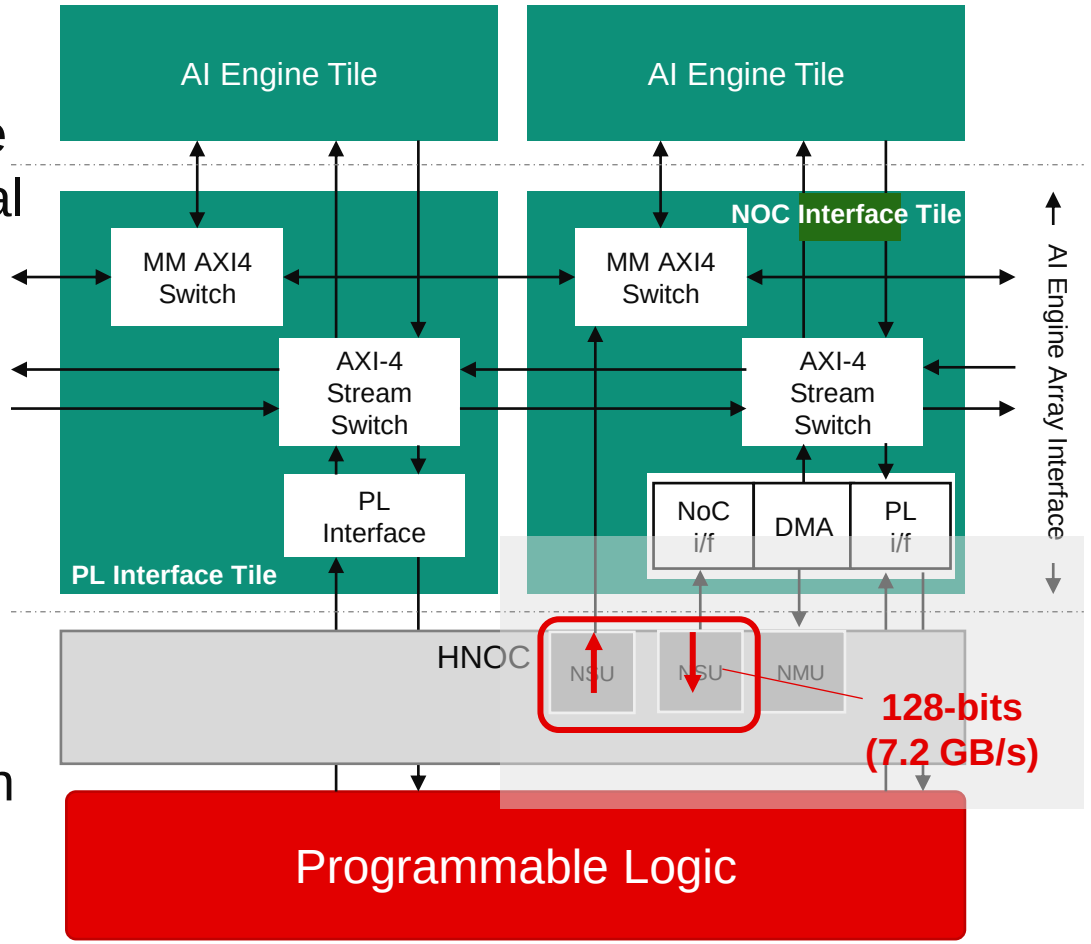
Communication PL ←→ AIE interface array (Some columns are not available)	Direction	#AXI Stream per Column	Bandwidth per Column	Bandwidth on VC1902
	→ North	8	32 GB/s	~1.3TB/s
	→ South	6	24 GB/s	~1TB/s

Within AI Engine Grid (All columns are available)	Direction	#AXI Stream per Interconnect	Bandwidth per Interconnect	Bandwidth on VC1902
	→ North	6	24 GB/s	1.2 TB/s
	→ South	4	16 GB/s	800 GB/s

Note: B/w calculation - 1 GHz AI Engine clock @ -1L speedgrade (0.7V), higher bandwidth is available with faster speed grade

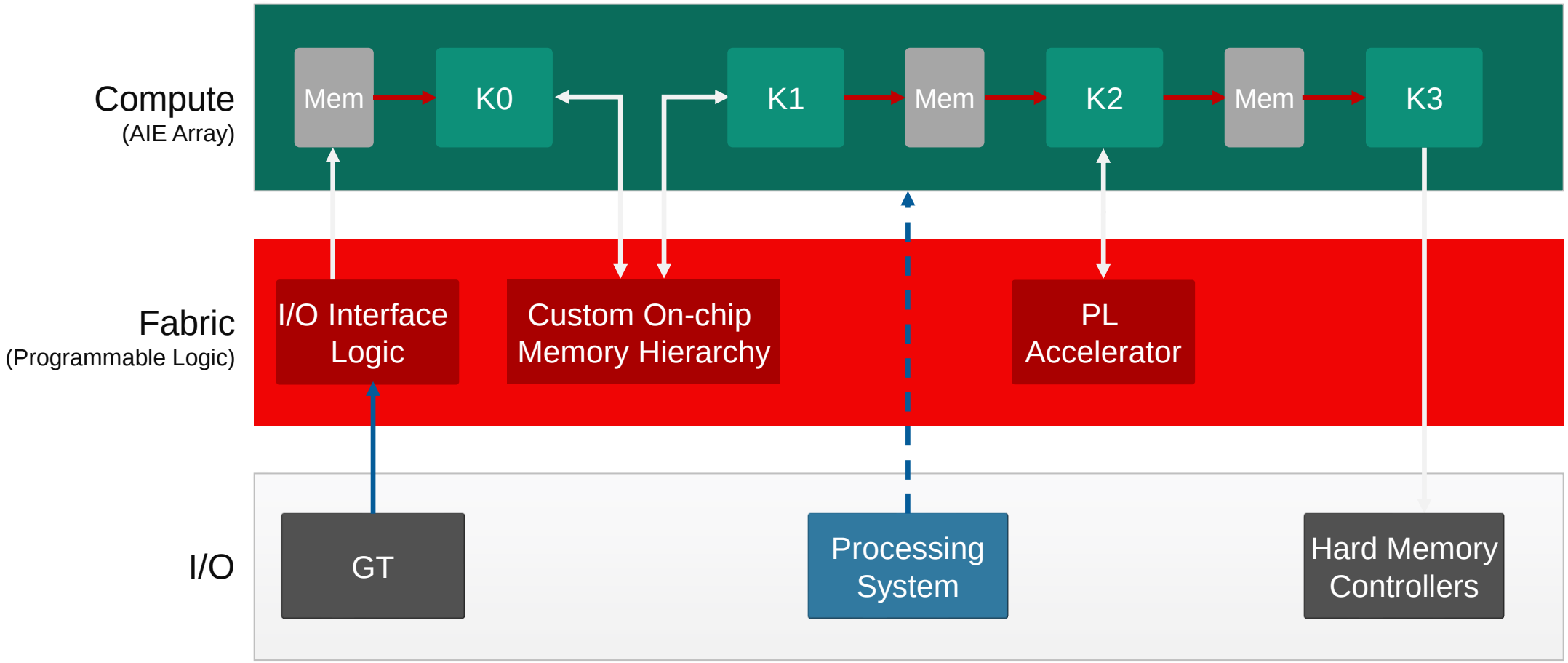
AI Engine to Programmable NoC Interface

- In addition to the AXI4-Stream, the NoC Interface tile provides a connection to the HNoC (horizontal NoC)
  - 4 streams from AI Engine to NoC
  - 4 streams NoC to AI Engine
- Each AI Engine to NoC can direct traffic between HNoC and the AXI4-Stream switch
  - Aggregation of 32-bit AI Engine stream into 128-bit NoC stream
- The number of horizontal and vertical channels in the device will limit b/w



Note: B/W calculation using -1L speedgrade (0.7V), higher bandwidth is available with faster speed grade

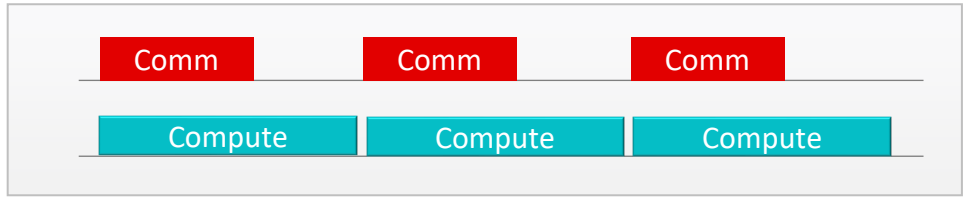
Data Movement Architecture: Device Integration



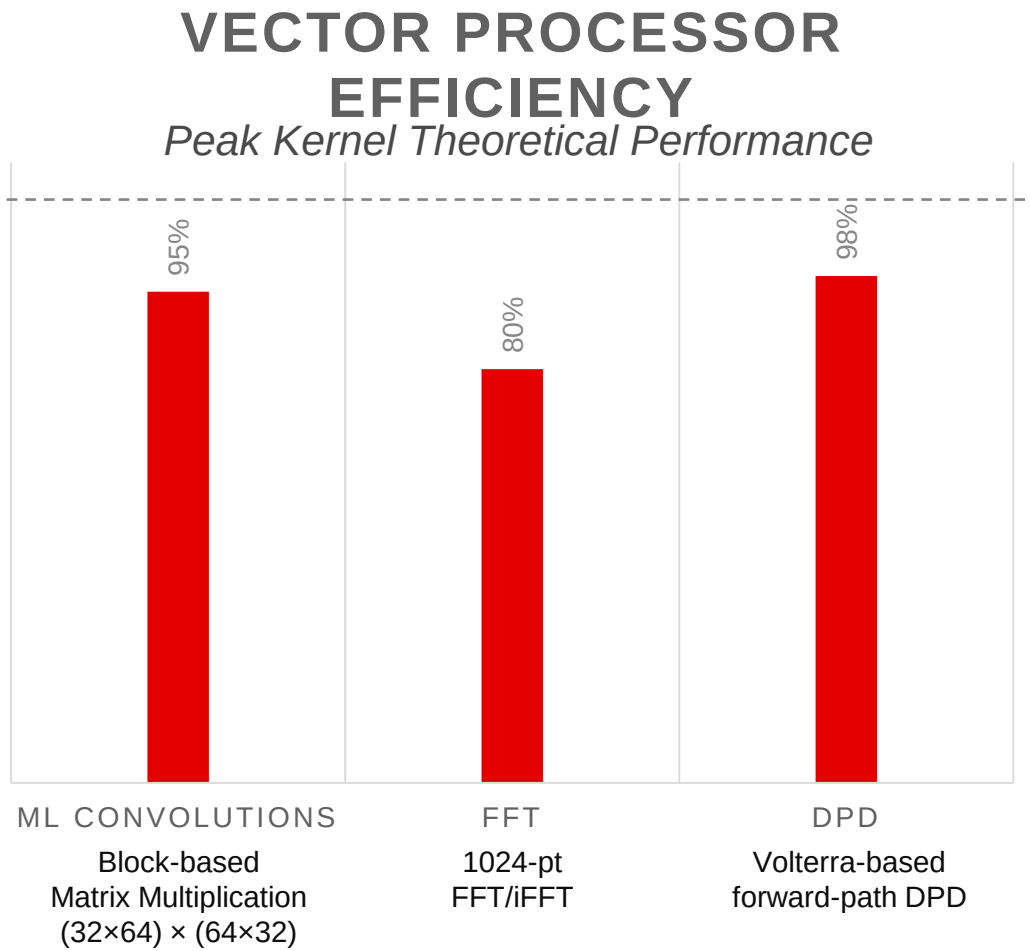
AMD XILINX

AI Engine Delivers High Compute Efficiency

- Adaptable, non-blocking interconnect
  - Flexible data movement architecture
  - Avoids interconnect “bottlenecks”
- Adaptable memory hierarchy
  - Local, distributed, shareable = extreme bandwidth
  - No cache misses or data replication
  - Extend to PL memory (BRAM, URAM)
- Transfer data while AI Engine Computes



Overlap Compute and Communication

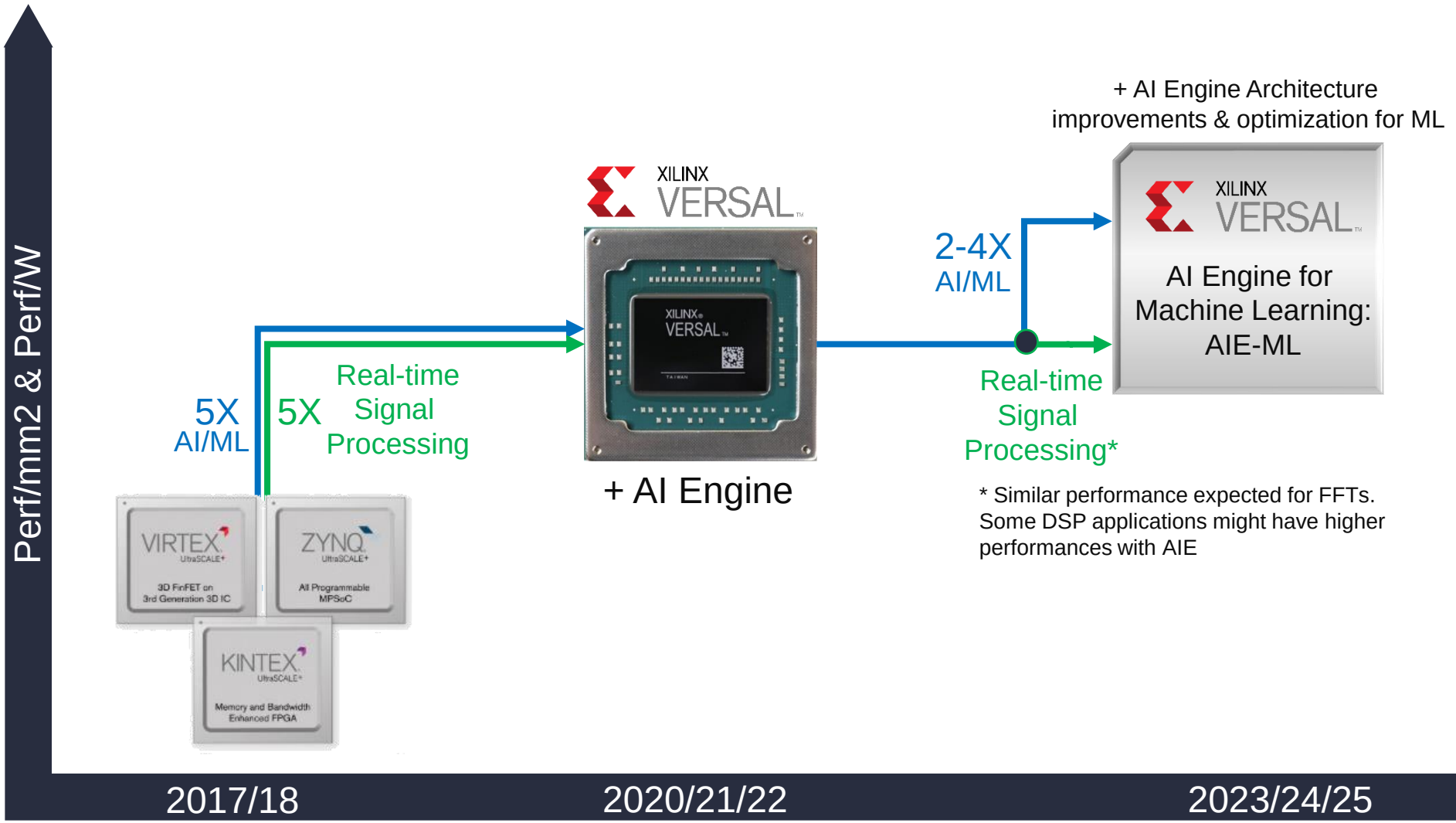


AMD XILINX

AIE vs AIE-ML

Key differences

Computed Density/Efficiency Progression



Versal Series with AI Engine

Versal® AI Core Series

		VC1352	VC1502	VC1702	VC1802	VC1902	VC2602	VC2802
Intelligent Engines	AI Engines Tiles	128	198	304	300	400	0	0
	AI Engine-ML Tiles	0	0	0	0	0	152	304
	AI Engine Data Memory (Mb)	32	50	76	75	100	76	152
	AIE-ML Shared Memory (Mb)	0	0	0	0	0	304	304
	DSP Engines	928	1,032	1,312	1,600	1,968	984	1,312
		AIE					AIE-ML	

Versal AI Edge Series

		VE2002	VE2102	VE2202	VE2302	VE1752	VE2602	VE2802
Intelligent Engines	AI Engine-ML Tiles	8	12	24	34	0	152	304
	AI Engine Tiles	0	0	0	0	304	0	0
	AIE/AIE-ML Data Memory (Mb)	4	6	12	17	76	76	152
	AIE-ML Shared Memory (Mb)	48	48	68	68	0	304	304
	DSP Engines	90	176	324	464	1,312	984	1,312
		AIE-ML				AIE	AIE-ML	

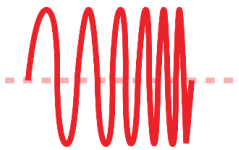
Versal Premium Series

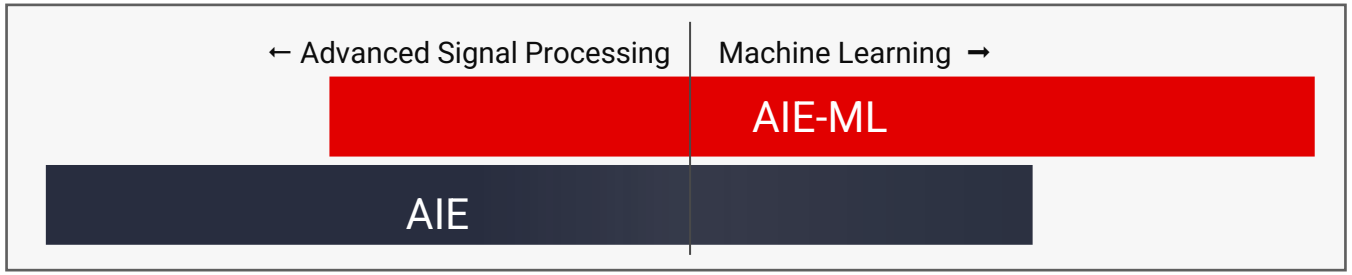
		VP1002	VP1052	VP1102	VP1202	VP1402	VP1502	VP2502	VP1552	VP1702	VP1802	VP2802
Intelligent Engines	DSP Engines	1,140	1,572	1,904	3,984	2,672	7,440	7,392	7,392	10,896	14,352	14,304
	AI Engines Tiles	-	-	-	-	-	-	472	-	-	-	472
	AI Engine Data Memory (Mb)	-	-	-	-	-	-	118	-	-	-	118
								AIE				AIE

AMD XILINX

Intelligent Engines Optimized for Any AI Application

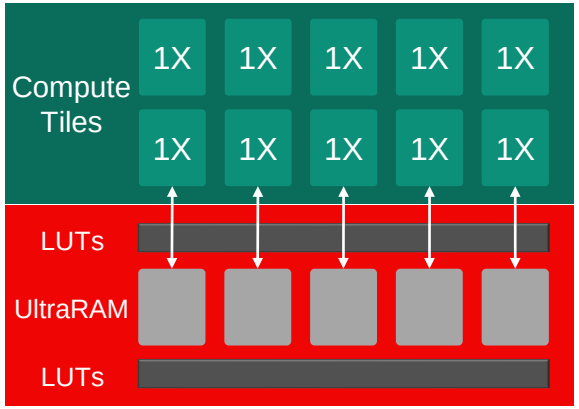
BEAMFORMING,
RADAR PROCESSING,
HIGH PERF. COMPUTING





CNN, RNN, MLP

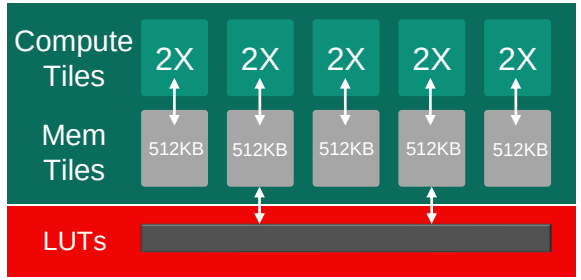
AIE Architecture



AIE	OPS / Tile	AIE-ML
256	INT4	1024
256	INT8	512
64	INT16	128
No native support	BFLOAT16	256
16	INT32	32*
16	FP32	51*

*Via software emulation (FP32 not IEEE-754 compliant)

AIE-ML Architecture



- Optimized for signal processing AND ML
- Flexibility for high performance DSP applications
- Native support for INT8, INT16, INT32, FP32

KB / Tile		
32	Data Memory	64
16	Program Memory	16

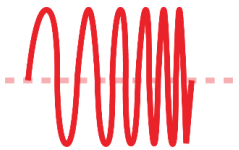
- Optimized for ML Inference Applications
- Maximum AI/ML compute with reduced footprint
- Native support for INT4, INT8, INT16, INT32, bfloat16
- Fine grained sparsity HW optimization
- Enhanced FFT & complex math support

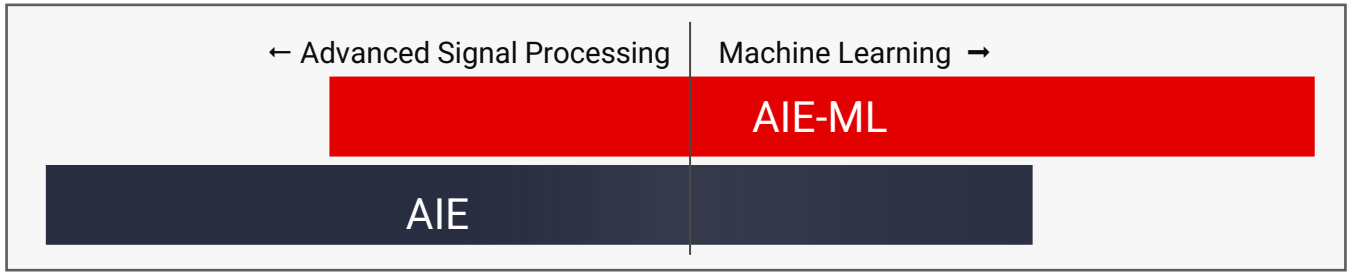
Versal® AIE-ML offers 2X AI Performance per Watt

AMD XILINX

Intelligent Engines Optimized for Any AI Application

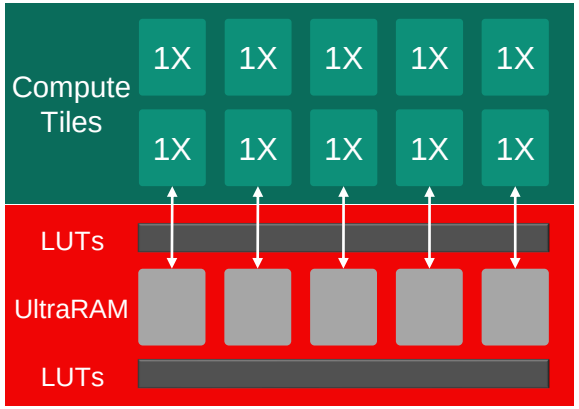
BEAMFORMING,
RADAR PROCESSING,
HIGH PERF. COMPUTING



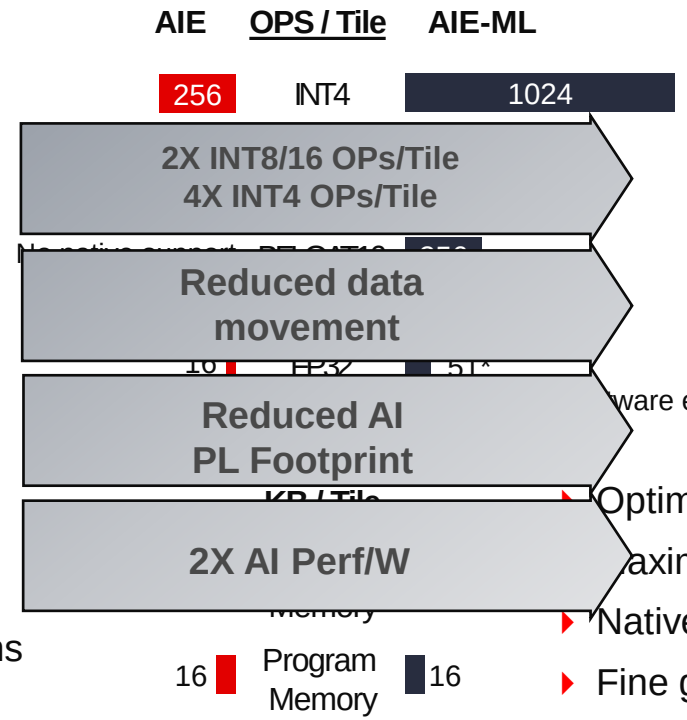


CNN, RNN, MLP

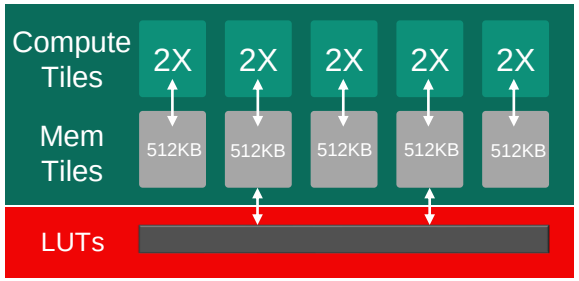
AIE Architecture



- Optimized for signal processing AND ML
- Flexibility for high performance DSP applications
- Native support for INT8, INT16, INT32, FP32



AIE-ML Architecture



ware emulation (FP32 not IEEE-754 compliant)

- Optimized for ML Inference Applications
- aximum AI/ML compute with reduced footprint
- Native support for INT4, INT8, INT16, INT32, bfloat16
- Fine grained sparsity HW optimization
- Enhanced FFT & complex math support

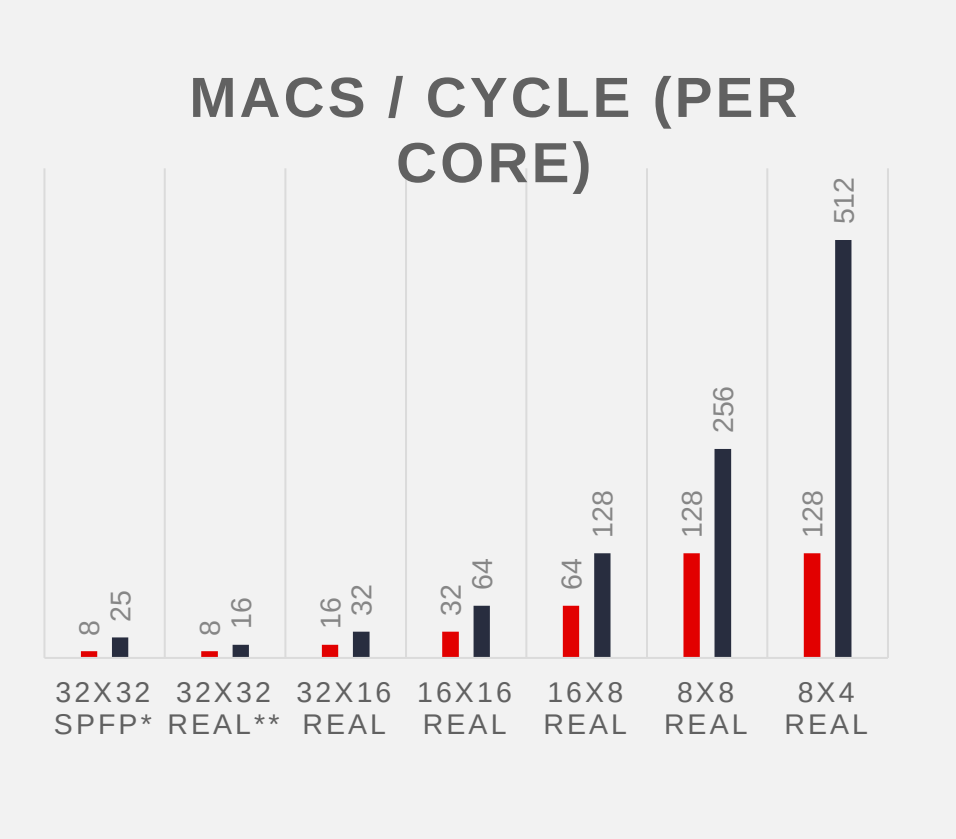
Versal® AIE-ML offers 2X AI Performance per Watt

AMD XILINX

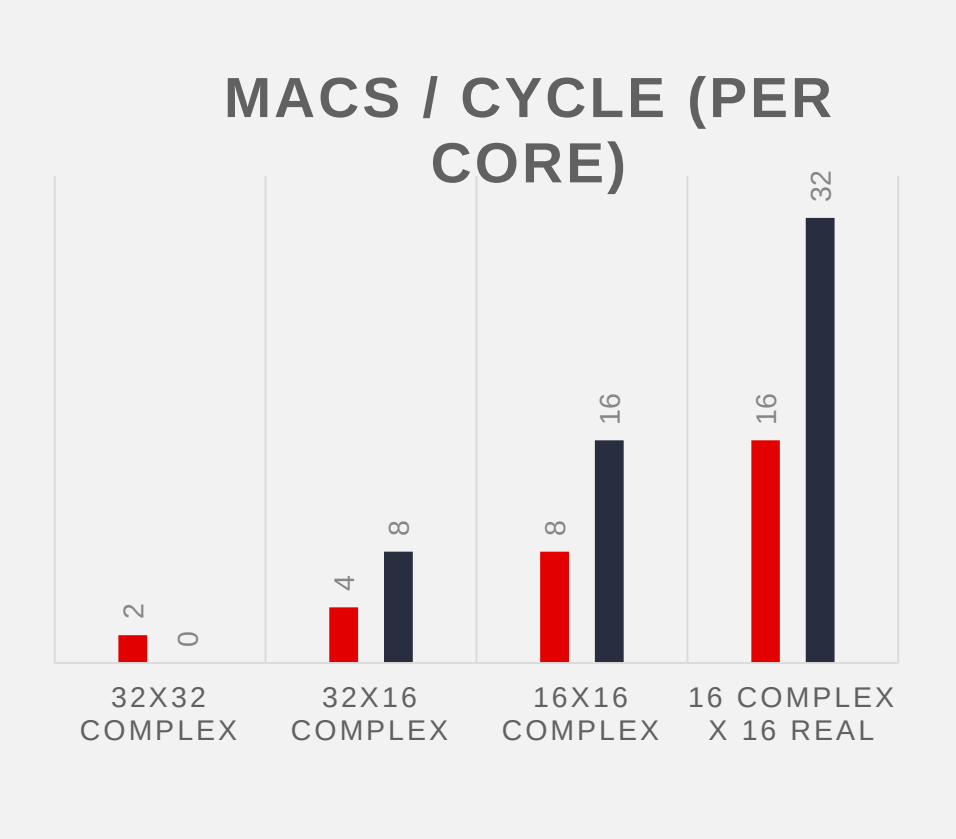
Multi-Precision Support

AIE
AIE-ML

Real Data Types



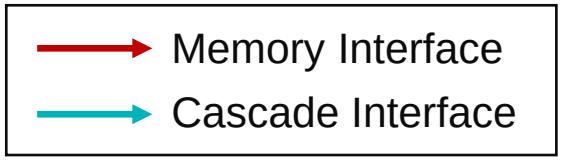
Complex Data Types



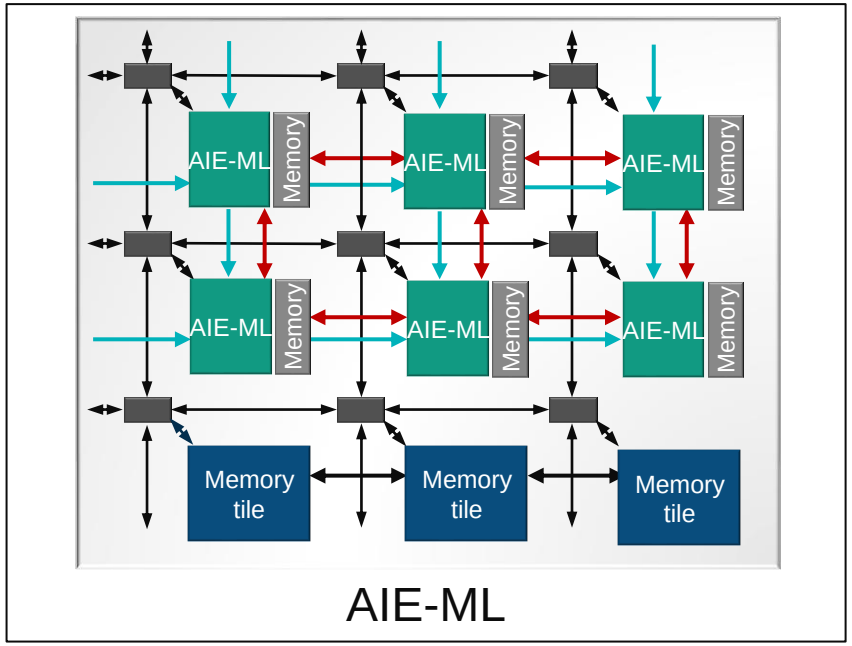
* 32 SPFP x 32 SPFP emulated using BFLOAT16 (following this paper). Not IEEE-754 compliant. Low accuracy mode
**32 Real X 32 Real is emulated

AMD XILINX

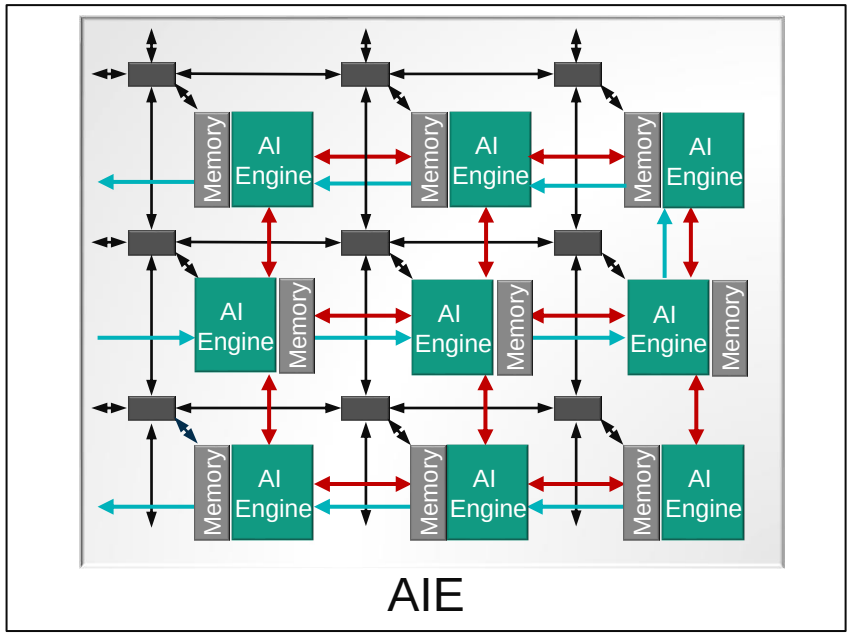
Array



- AIE-ML Array includes Memory tiles at the bottom of the array
- All lines of compute tiles are identical in AIE-ML
  - No snake shape of the cascade interface
    - always from left to right and top to bottom
  - Local memory is always East
  - Accessible neighbor memories are always North, South , and West
- Interfaces with PL and NoC stays the same
  - However, no direct access to the AIE-ML tiles
  - Interface through the AIE-ML memory tiles



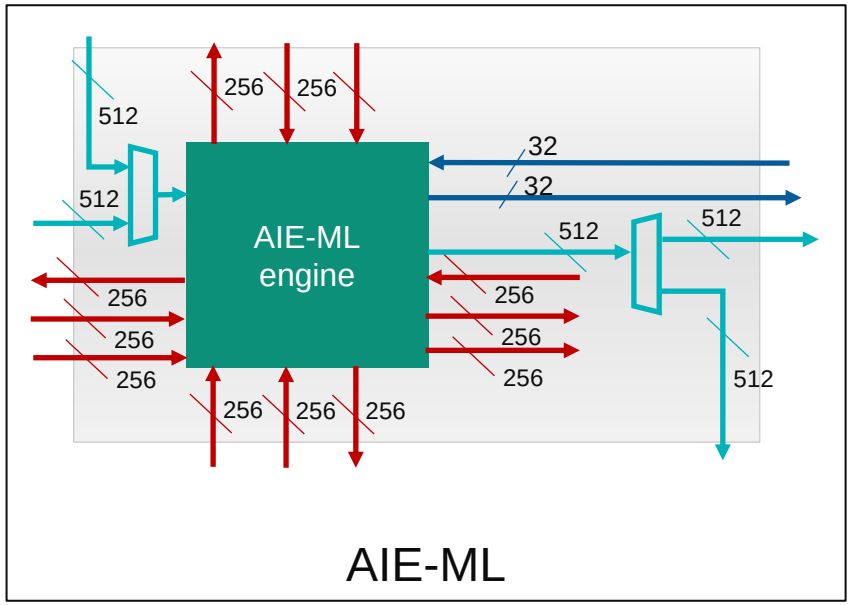
AIE-ML



AIE

AMD XILINX

Interfaces



- AIE-ML tile has 2 directions for the cascade input (North and West) and 2 direction for cascade output (South and East)
- Cascade interface width is increased in AIE-ML
  - to 512-bits in AIE-ML (16 lanes of 32-bit data or 8 lanes of 64-bit data)
  - 384-bits in AIE
- Direct AXI4-Stream interface
  - 2 x 32-bit input and 2 x 32-bit output in AIE
  - 1 x 32-bit input and 1 x 32-bit output in AIE-ML
- Stream FIFO only in AIE

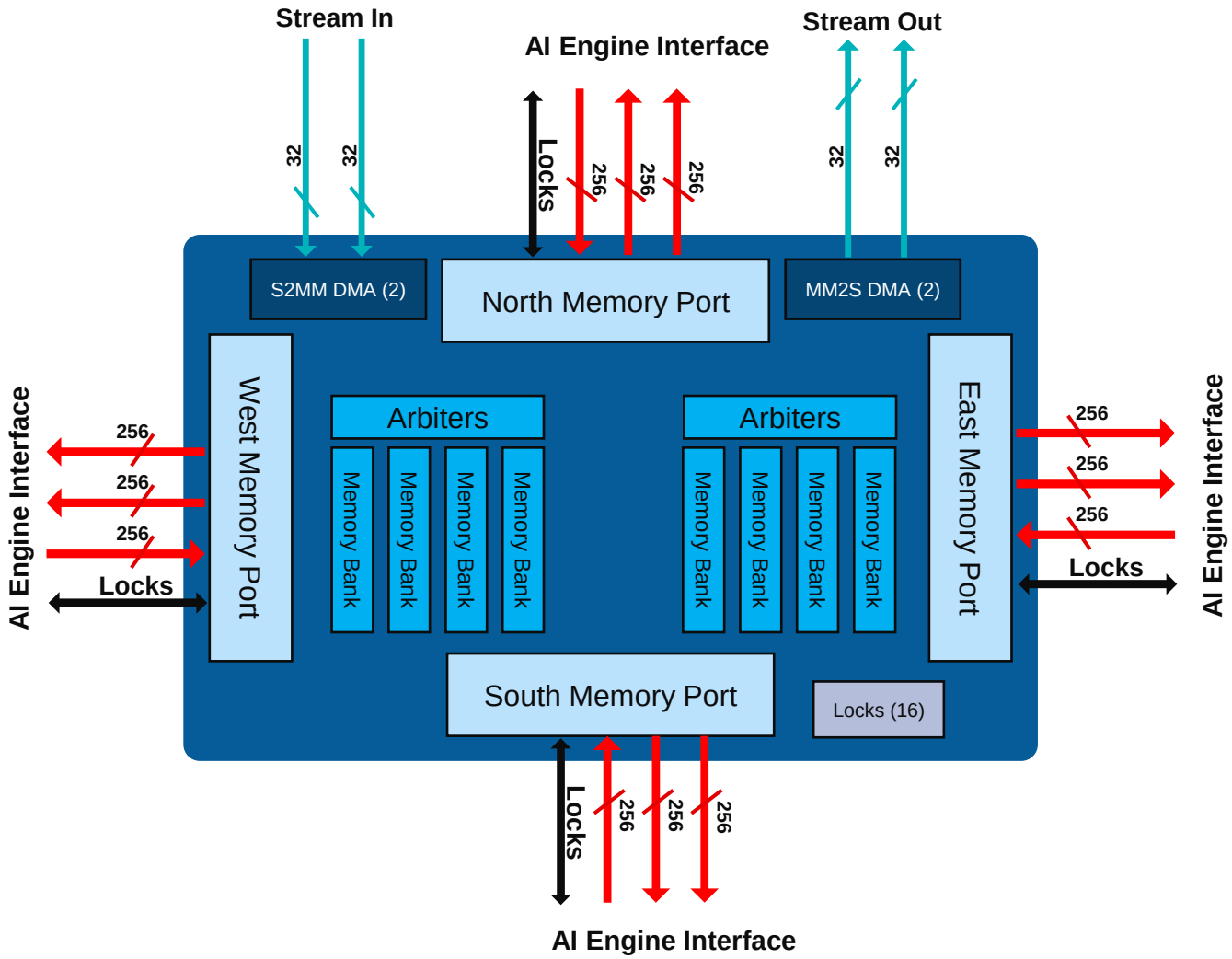
AMD XILINX

Core level

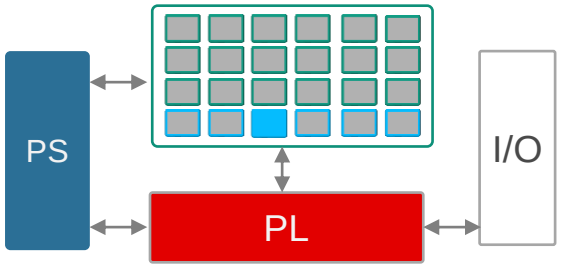
- Number of multipliers doubled in AIE-ML to 256 (int8 x int8) vs. 128 (int8 x int8) in AIE
- Added native support for INT4 and BFLOAT16 in AIE-ML
  - Removed native support for SPFP32
    - can be emulated using BFLOAT16 (not IEEE-754 compliant). Emulation following this paper
- Added support for 2:4 sparsity in AIE-ML
- Removed support for 5G specific extensions in AIE-ML
- Removed support for scalar non-linear functions (for example sqrt, Sin, Cos, and InvSqrt) and scalar floating point / integer conversions

Memory Module

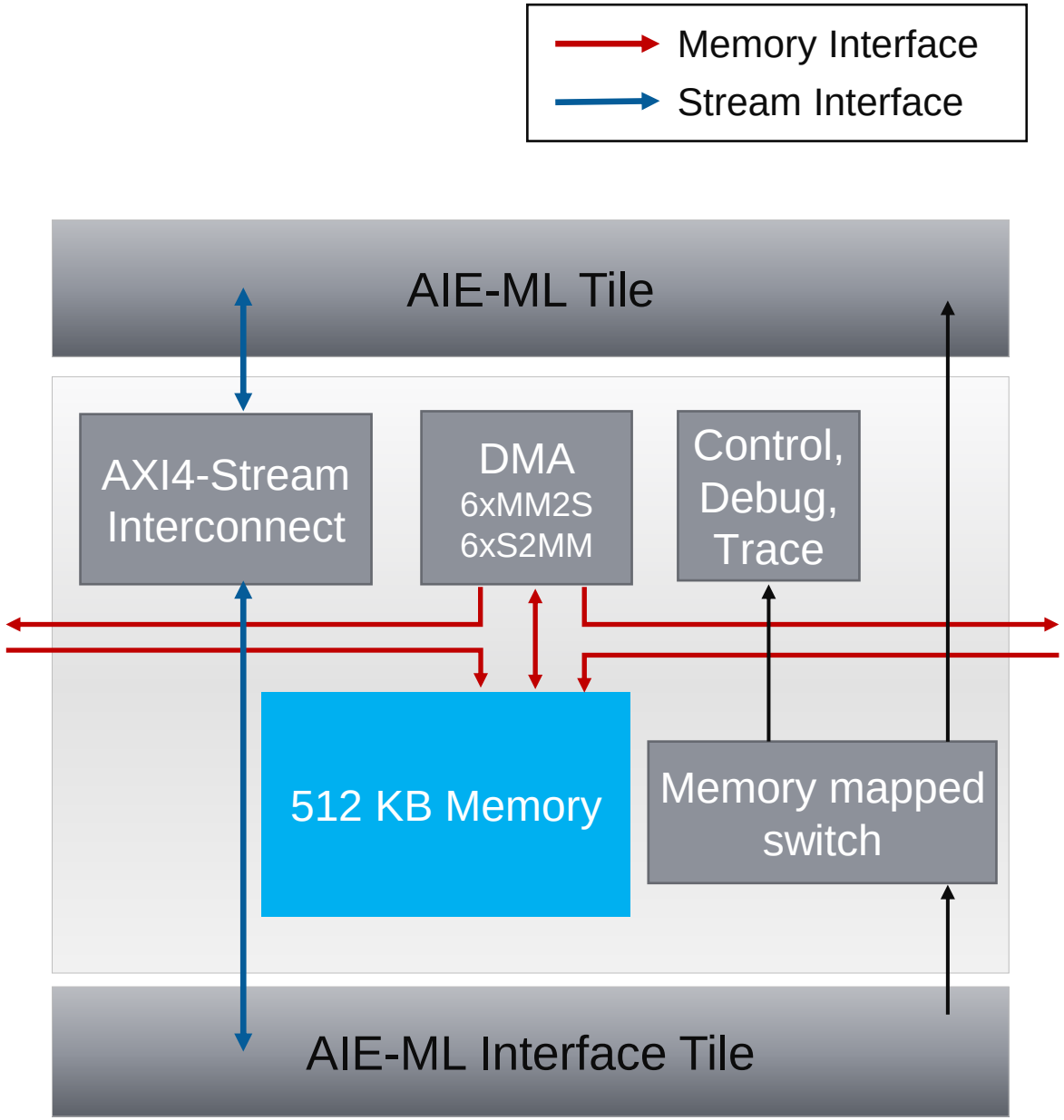
- Memory doubled in AIE-ML to 64KB
- DMA improved in AIE-ML
  - Improved address generation to support 3D addressing in AIE-ML
  - Supports S2MM finish on TLAST and out-of-order packets
  - Compression (MM2S) and decompression (S2MM)
- Semaphore locks in AIE-ML vs. binary locks in AIE
  - 6-bit unsigned lock state



AIE-ML Memory Tile

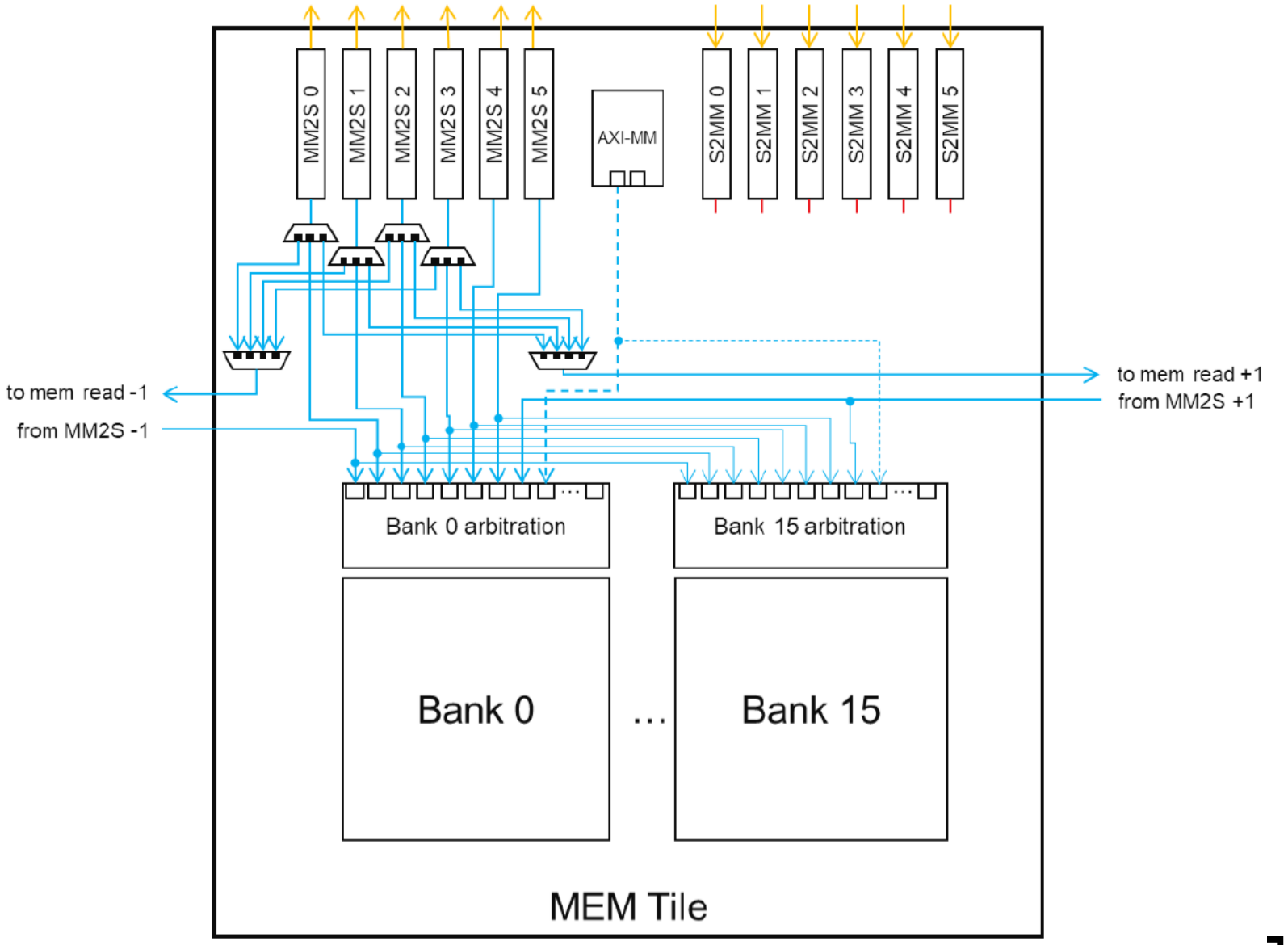


- AIE-ML Memory Tile (AIE-ML architecture only)
  - 512KB memory arranged into 16 banks
  - MM2S and S2MM with 6 channels each
    - 32-bit stream interfaces
    - 128-bit memory interfaces
    - Support 4D tensor address generation
    - Supports compression and decompression of the data
  - AXI4-Stream interconnect with North and South ports (no East and West)
- 1 or 2 rows at the bottom of the array



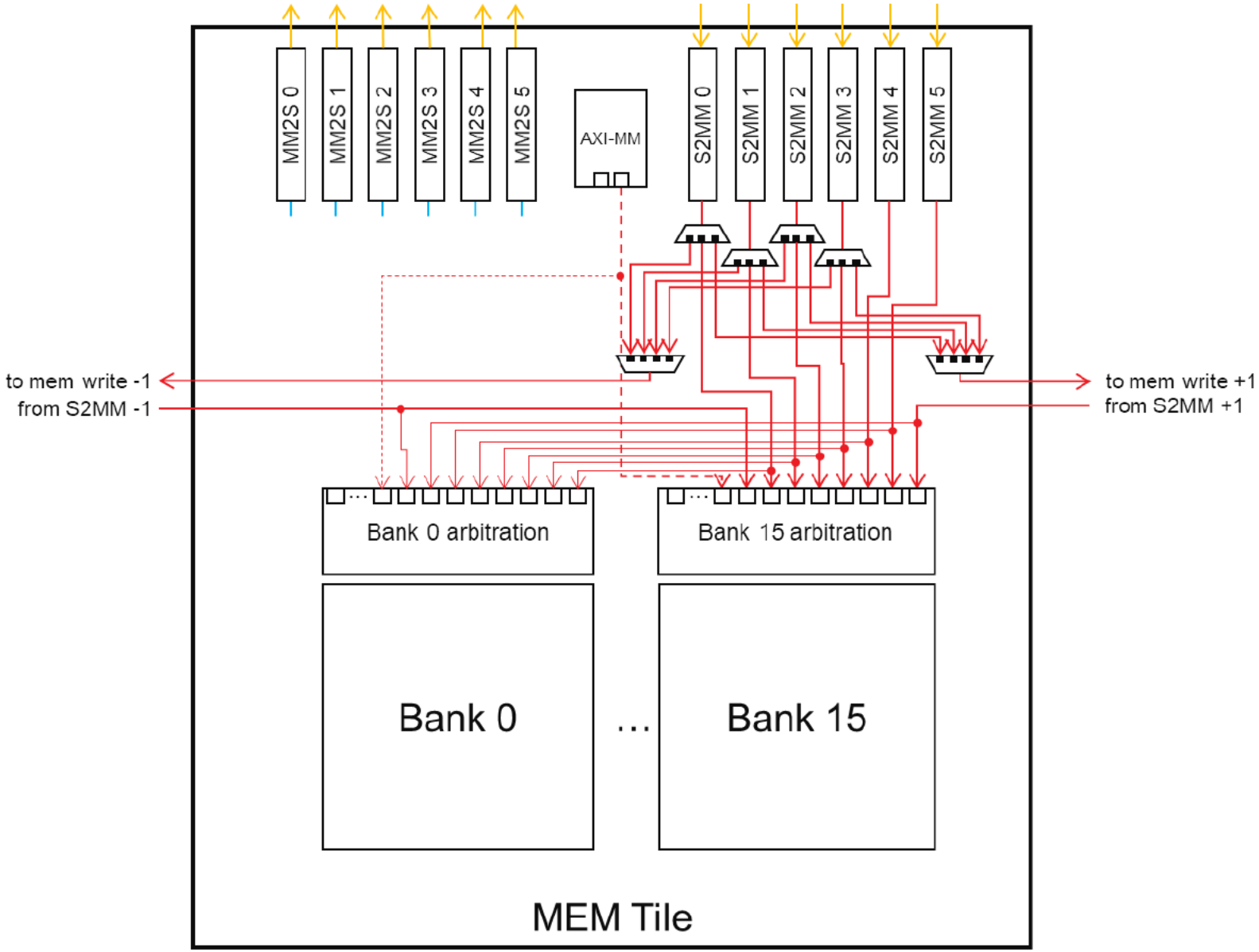
Memory Tile

- For Each Bank
  - 9 read interfaces
    - AXI-MM Read (inc. ctrl-pkts)
    - MM2S (MEM tile -1)
    - 6x MM2S channels [0-5]
    - MM2S (MEM tile +1)
  - 9 write interfaces
    - AXI-MM Write (inc. ctrl-pkts)
    - S2MM (MEM tile -1)
    - 6x S2MM channels [0-5]
    - S2MM (MEM tile +1)



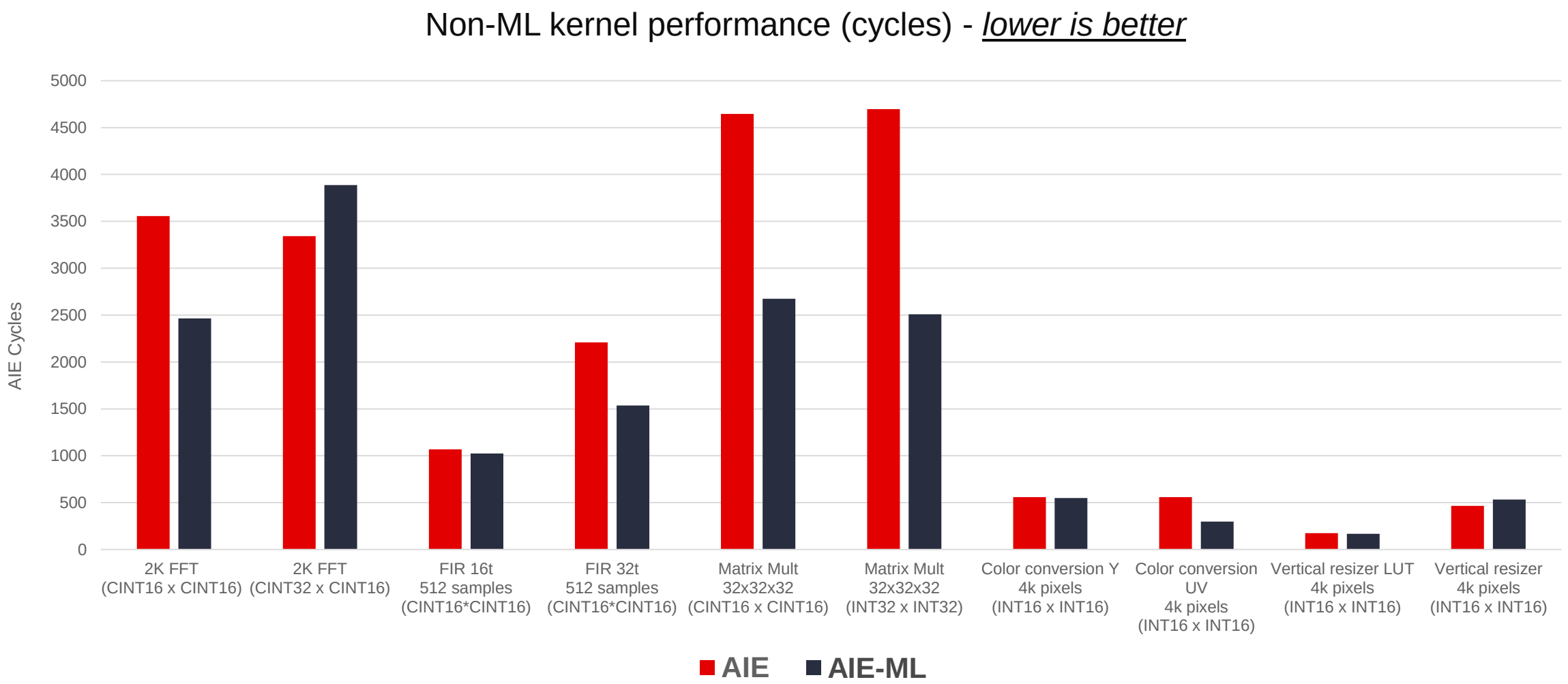
Memory Tile

- For Each Bank
  - 9 read interfaces
    - AXI-MM Read (inc. ctrl-pkts)
    - MM2S (MEM tile -1)
    - 6x MM2S channels [0-5]
    - MM2S (MEM tile +1)
  - 9 write interfaces
    - AXI-MM Write (inc. ctrl-pkts)
    - S2MM (MEM tile -1)
    - 6x S2MM channels [0-5]
    - S2MM (MEM tile +1)



AMD XILINX

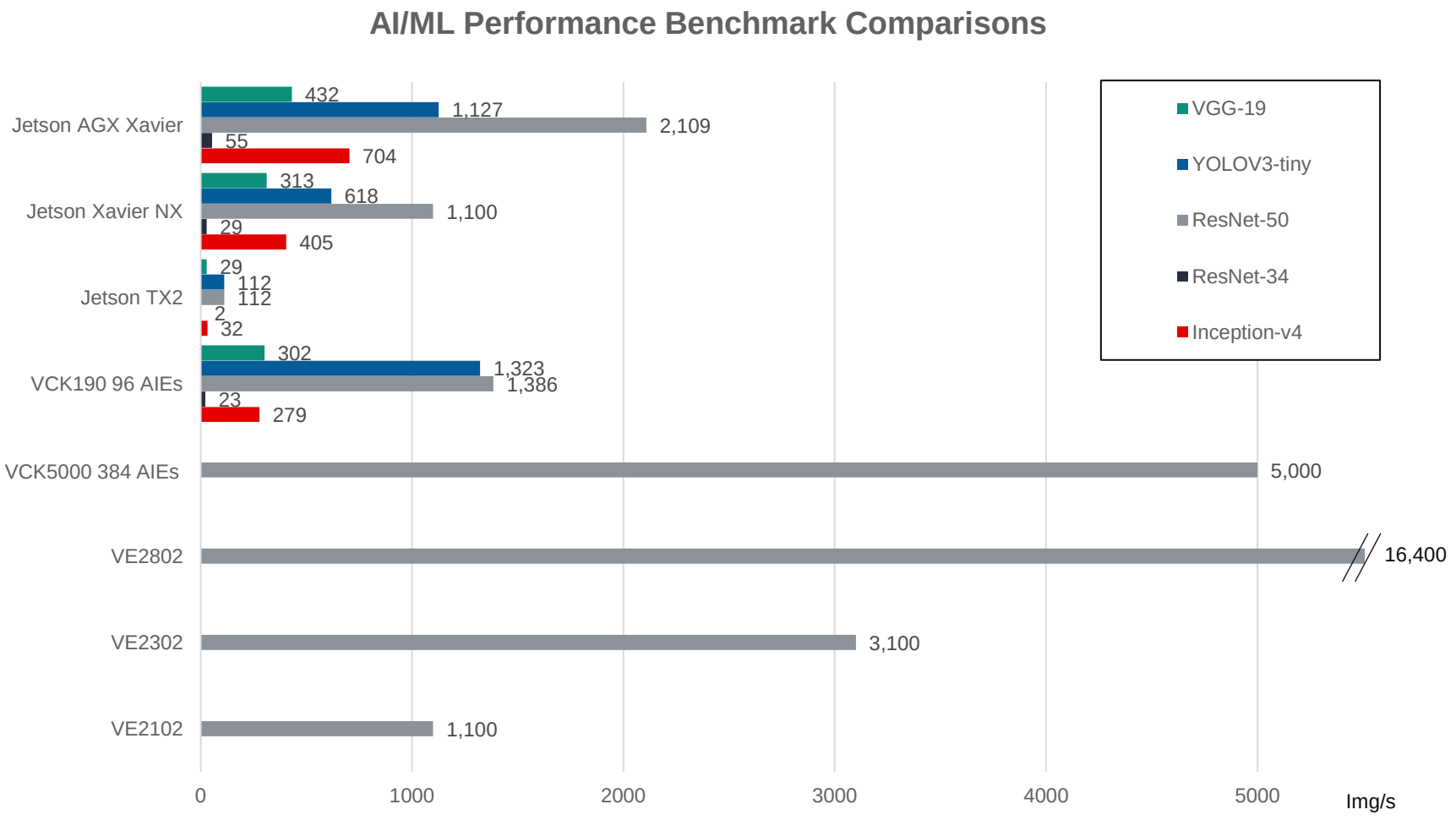
AIE / AIE-ML Kernel Performance using AIE APIs



AI Engine vs AI Engine-ML summary

	AI Engine	AI Engine-ML
Array structure	checkerboard	All lines identical
Cascade Interface	384-bits wide Horizontal Direction	512-bits wide Horizontal and Vertical Directions
Tile Stream Interfaces	2*32-bit In and 2*Out 32-bit out	1*32-bit In and 1*Out 32-bit out
Memory Load/Store (per cycle)	512/256 bits	512/256 bits
Int8 * Int8 Multipliers	128	256
Int4 * Int8 Multipliers	128	512
Format Native Support	Int8/16/32, Cint16/32, SPFP32	Int4/8/16, cint16, BFloat
Non-Linear Functions in scaler processor	Yes	No
Tile Local Memory	32KB	64KB
Tile Local Memory DMA	-	Support for 3D addressing modes S2MM finish on TLAST and out-of-order packets Compression/Decompression
Local Memory Locks	Boolean	Semaphore
Memory Tiles	No	Yes (512KB)

ML Performance Comparisons
Versal® Devices vs Nvidia Jetson Xavier and TX2



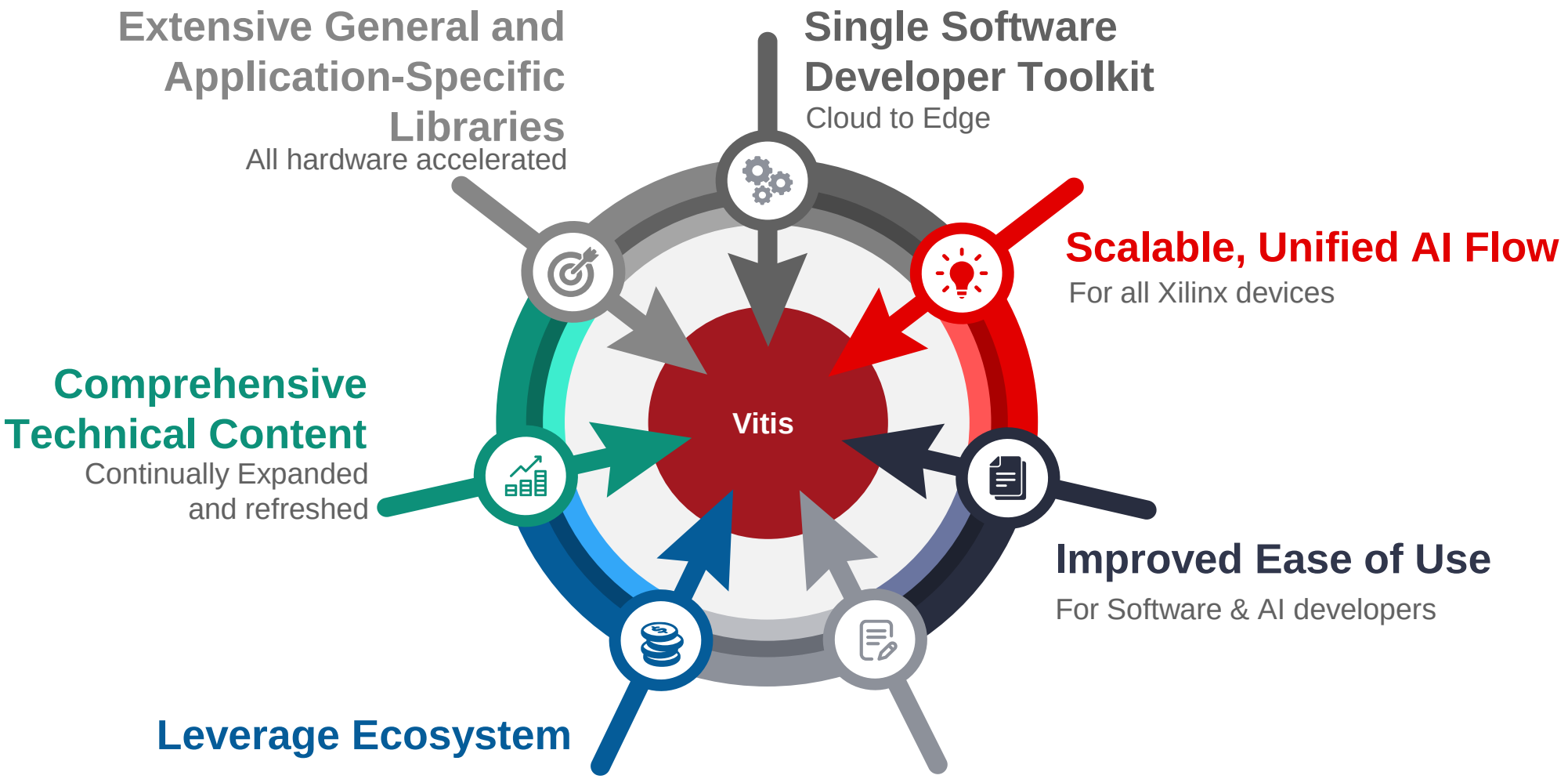
Notes:

1. Jetson benchmark data illustrates maximum performance results
   - The maximum throughput results were obtained without latency limitation and illustrate the maximum performance that can be achieved
   - https://developer.nvidia.com/embedded/jetson-benchmarks
2. Versal AI Core VC1902 device on VCK190 Eval Board with 96 AIEs measured with hardware
3. VCK5000 with 384 AIEs measured with hardware. Batch size=8, Latency=1.6ms
   - Nvidia T4: 3,785 img/s Latency = 2.1ms
4. VE2802, VE2302, VE2102 ResNet-50 performance simulated

Software and Tools

AI Engine Vitis™ Tool Flow

Vitis™ Unified Software Platform Guiding Principles



AMD XILINX

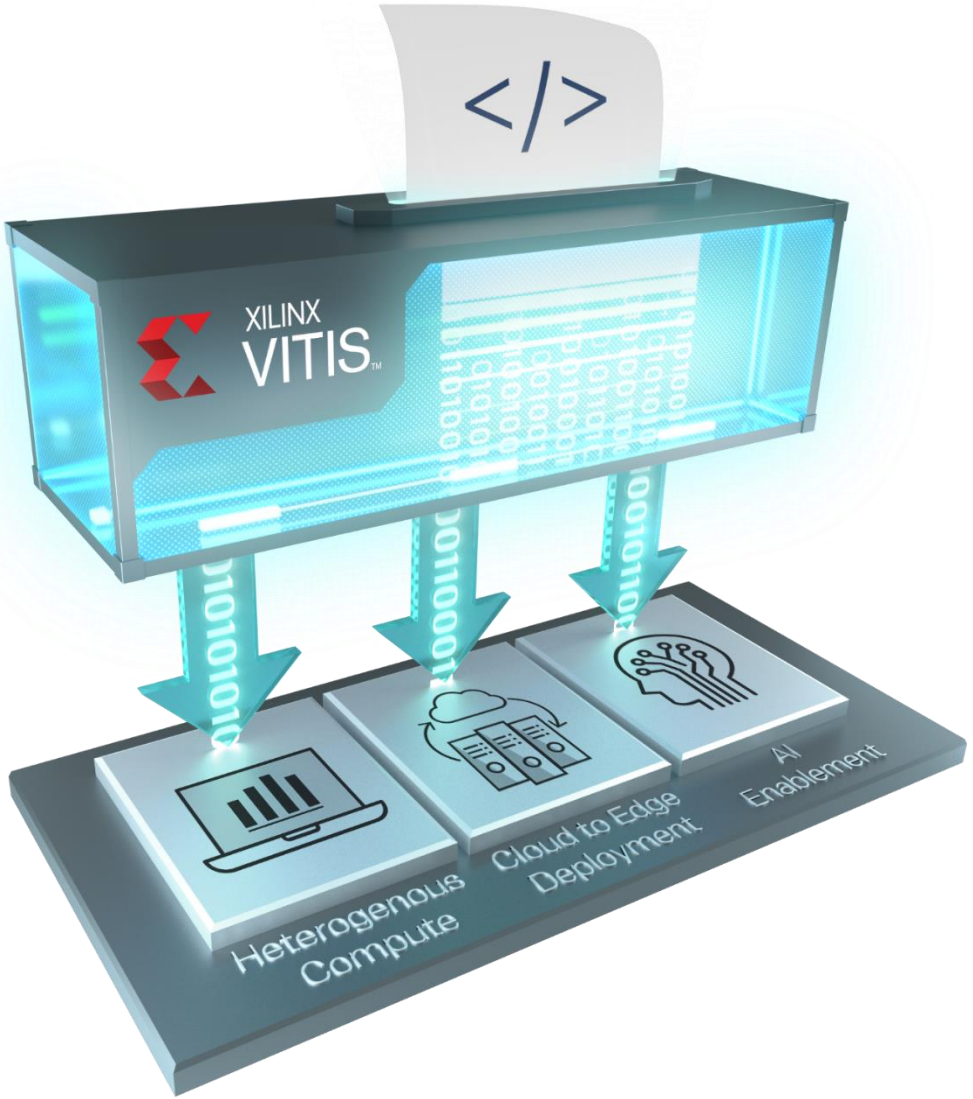
XILINX VITIS™

Unified Software Platform

Software & AI

Adaptive computing

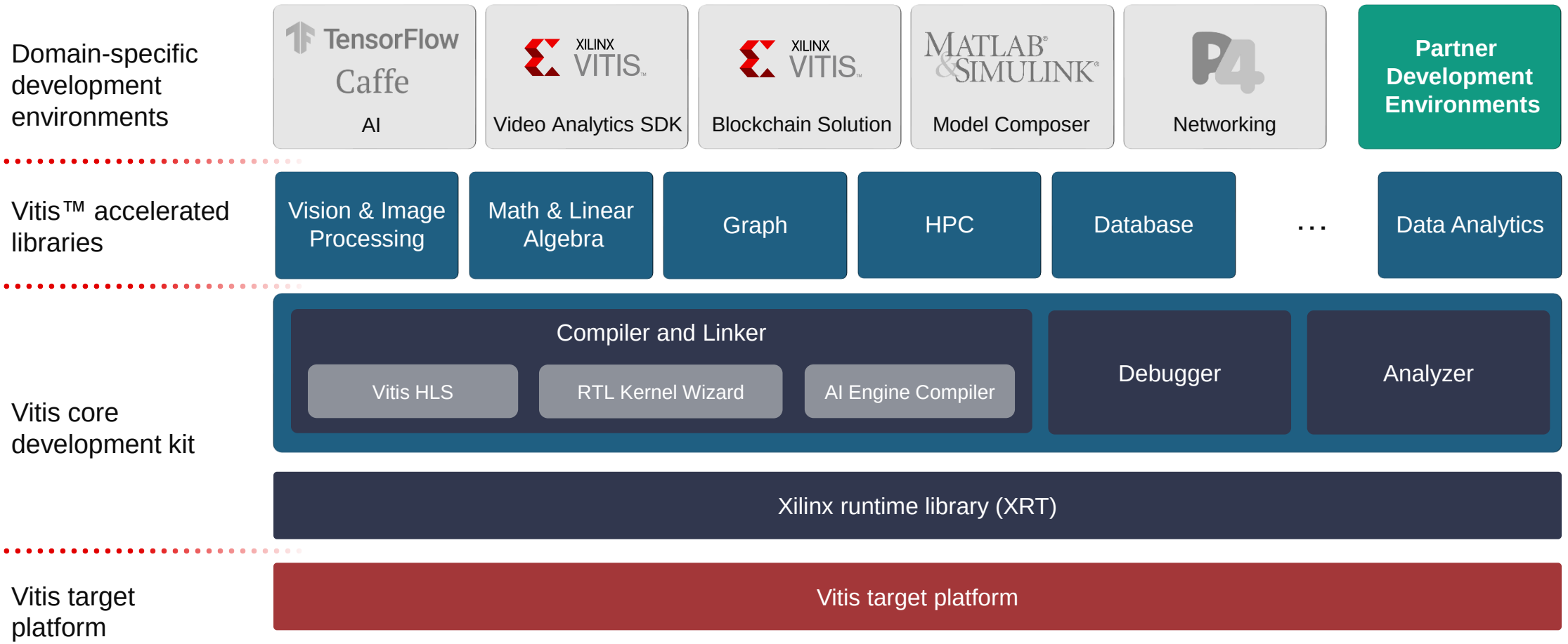
Edge to Cloud



AMD XILINX

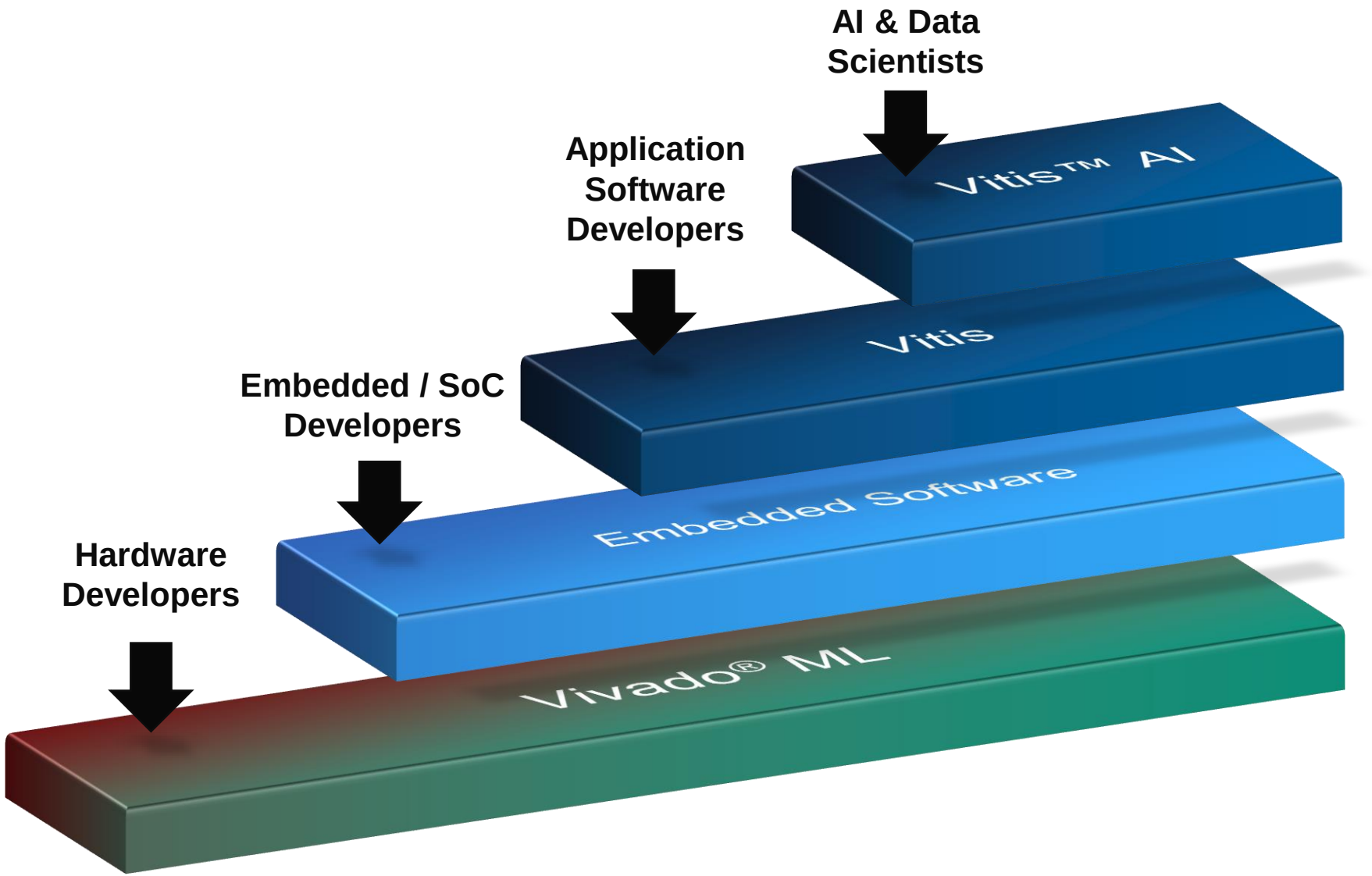
Vitis Unified Software Platform

For Heterogenous Compute, Edge to Cloud



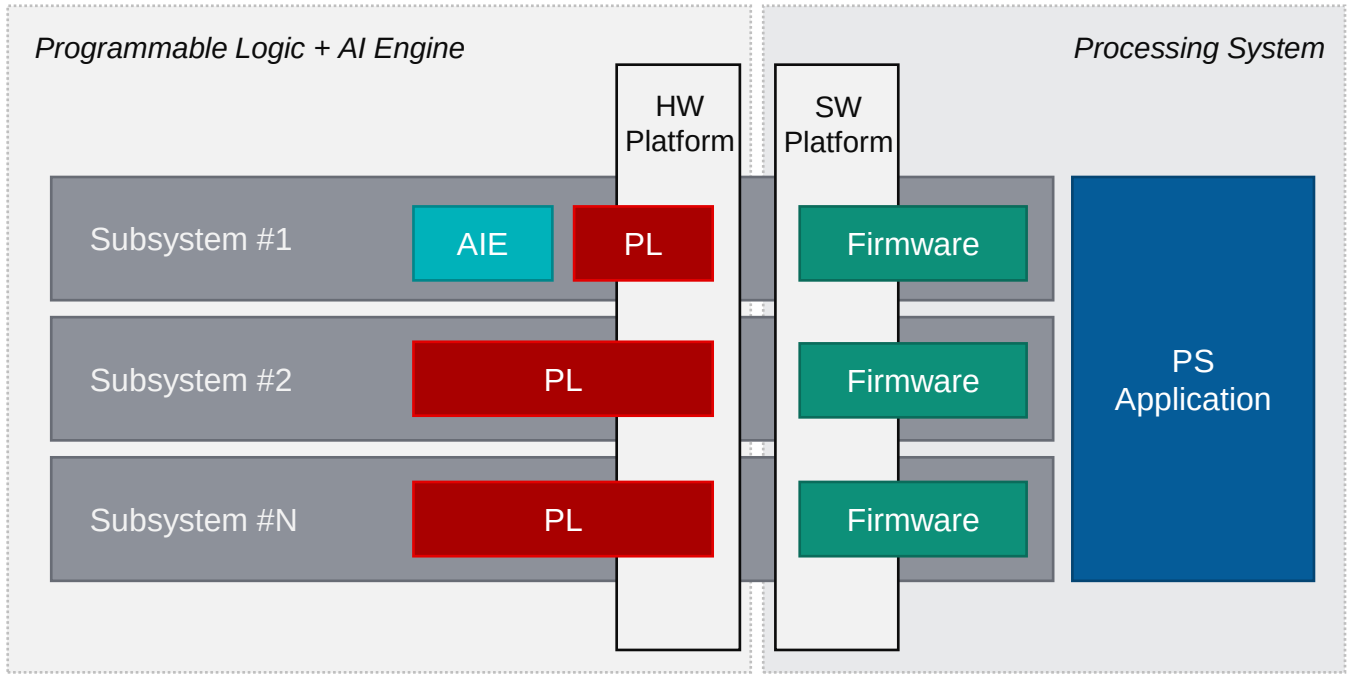
AMD XILINX

Development Platforms for ALL Developers



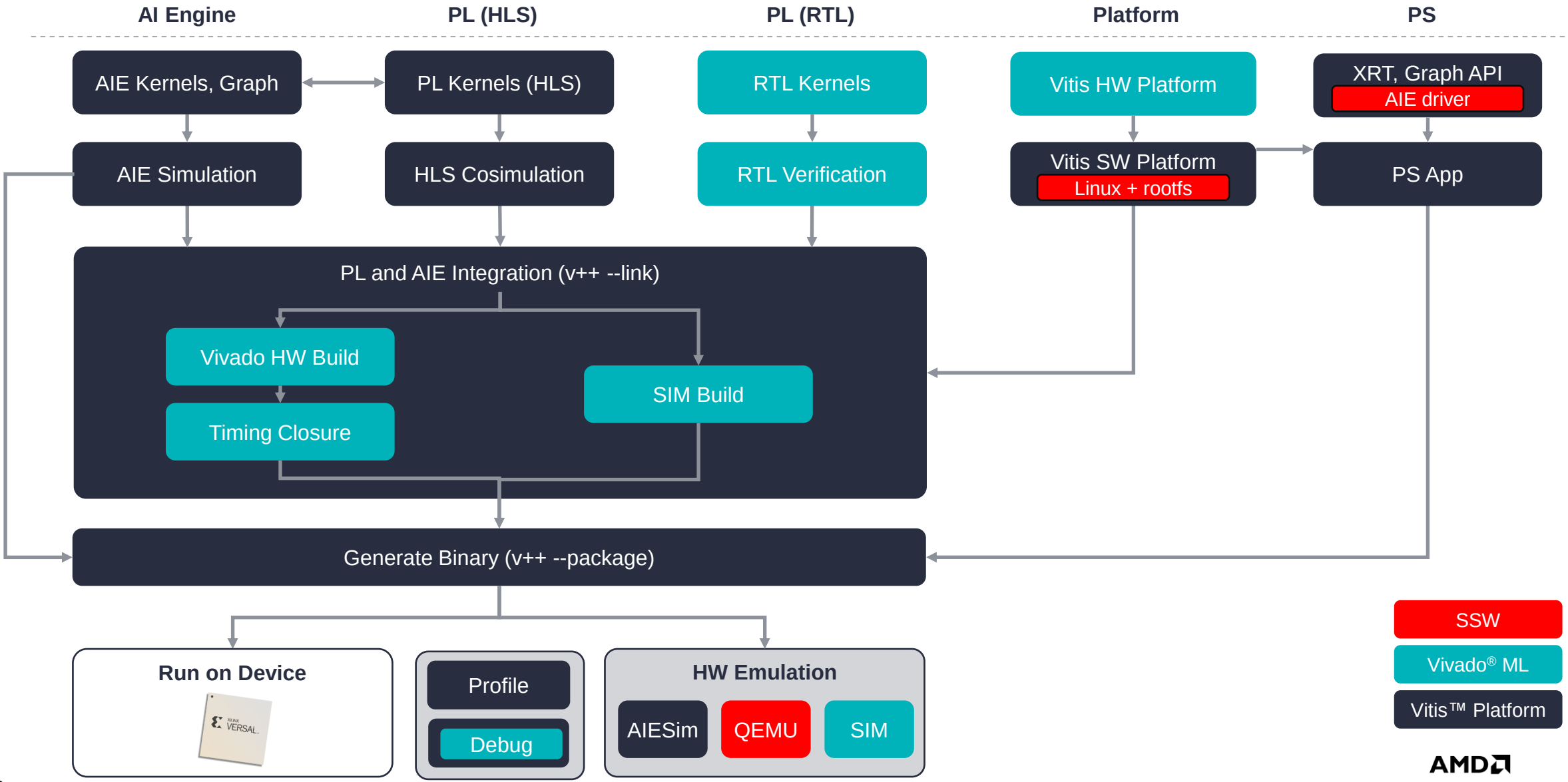
AMD XILINX

Vitis™ Platform Philosophy: Platforms and Subsystems



- Subsystems form the customer's differentiating logic: AI Engine and PL kernels, operating under the supervision of the processing system (PS)
- Versal platform provides essential infrastructure services (CIPS, programmable NoC, I/Os, OS, drivers…)
- Platform insulates developers from low-level details; lets them focus on application development (SW, PL, or AI Engine)

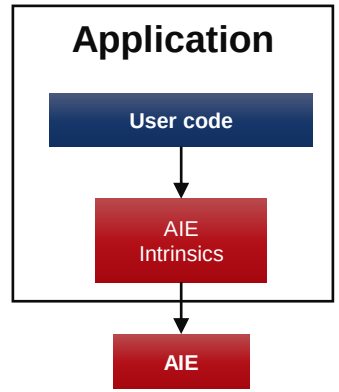
Vitis™ Development Flow for Versal® ACAP



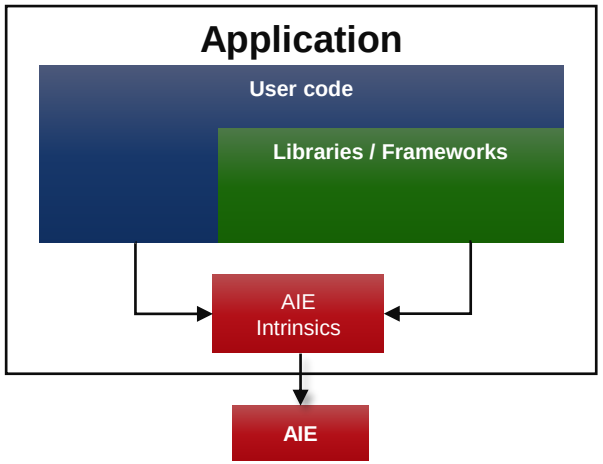
AI Engine API (High-Level Intrinsic APIs)

AI Engine Programming

- Kernel Development is Difficult
  - Long Development Times
  - Hard to Maintain
  - Difficult to optimize for performance



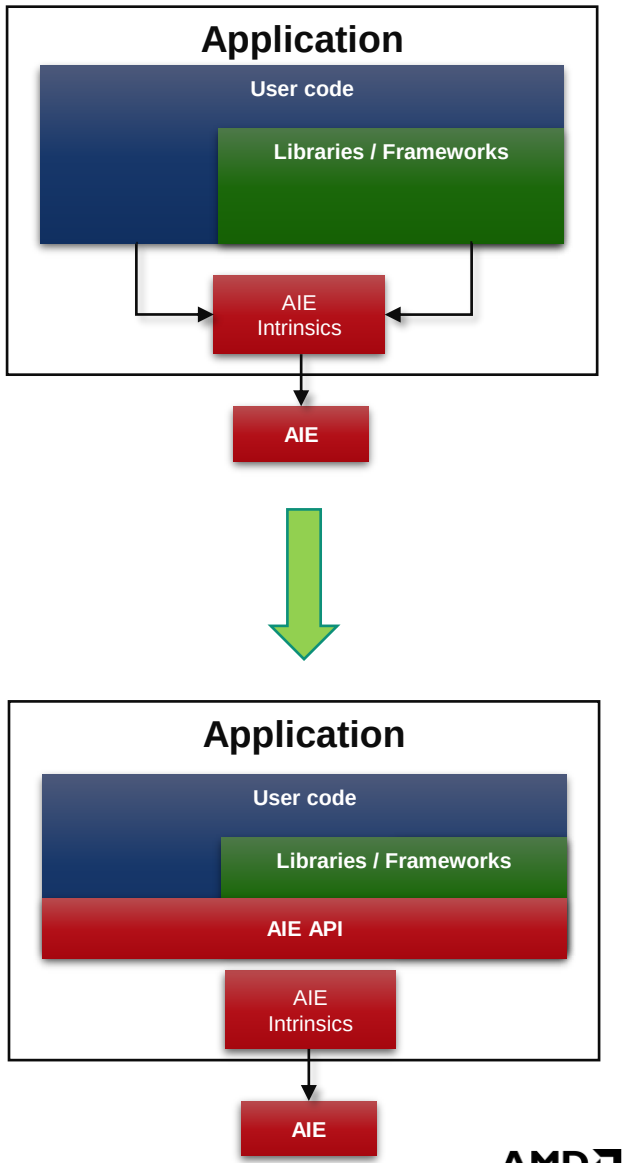
- Libraries / Frameworks
  - Specific Functionality
  - Do not address custom kernel development
  - Internal library development is still hard and costly



AMD
XILINX

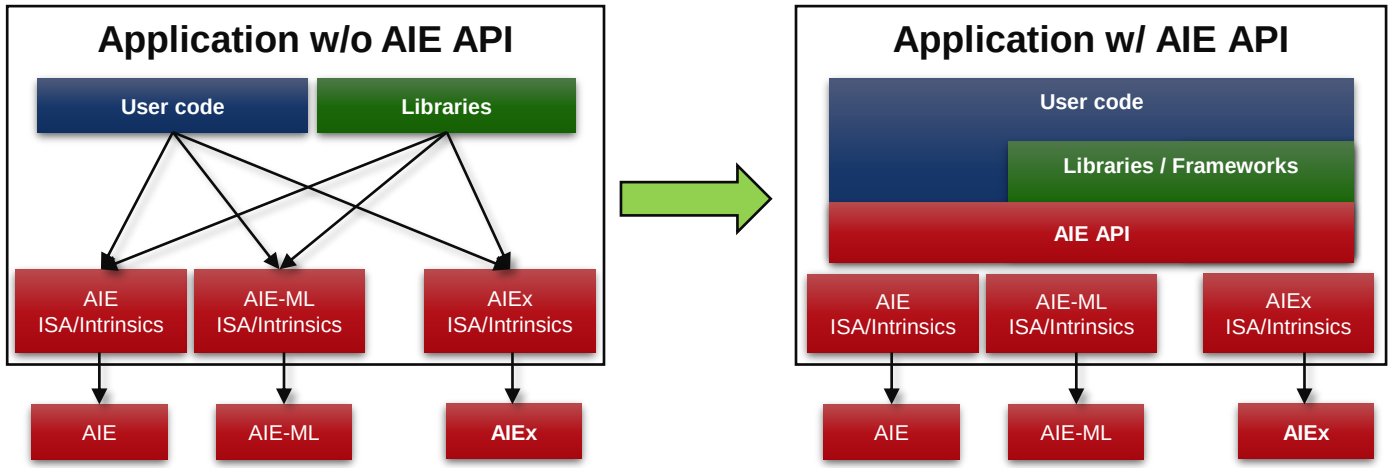
AI Engine API

- AI Engine API - Provides an easier to use programming model through higher-level abstraction that requires reduced knowledge of the underlying hardware to achieve optimized performance



AI Engine API - Portability

- Kernel code that is portable across AI Engine derivatives
  - No need to completely rewrite to take advantage of the latest features



AI Engine API – Reduced Development time

- Reduced Development time
  - Easier to read code
  - Easier to maintain code
  - Equivalent performance

Image Thresholding

Current (428 cycles)

```
void threshold_16bit_orig( const int16* restrict  img_in, int16* restrict img_out,
                           int image_width, int image_height)
{
    v16int16 * restrict ptr_img_buffer = (v16int16 *) img_in;
    v16int16 * restrict ptr_out_buffer = (v16int16 *) img_out;
    v16int16 * restrict ptr_in =(v16int16*) ptr_img_buffer;
    v32int16 * restrict ptr_out = (v32int16*)ptr_out_buffer;

    unsigned int  temp_val=0;

    v16int16  chess_storage(WR2) data_buf1;
    v16int16  chess_storage(WR3) data_buf2;

    v32int16 chess_storage(XA) constants;
    v32int16 chess_storage(XC) data_out;

    constants = upd_elem(constants, 0, 0);    // updating constant zero_val value
    constants = upd_elem(constants, 1, 100);  // updating constant threshold value

    for(int j = 0; j<(image_height * image_width); j+=32) // 16x samples per loop
        chess_prepare_for_pipelining
        chess_loop_range(14,)
    {
        data_buf1 = *(ptr_in++); // in:00++15 |________|________|________|
        data_buf2 = *(ptr_in++); // in:00...15|r1:16++31|________|________|___

        temp_val = lt32(constants, 1, 0, 0, 0x1111,
                        concat(data_buf1,data_buf2), 0, 0x06040200, 0x0e0c0a08, 0x3210);
        data_out = select32(temp_val, concat(constants, concat(data_buf1,data_buf2)),
                            0, 0, 0, 0x0000, 32, 0x06040200, 0x0e0c0a08, 0x3210);

        *(ptr_out++) = data_out;
    }
}
```

AIE API (424 cycles)

```
template <int N, typename T>
void threshold     ( const T * restrict  img_in, T * restrict img_out,
                     int image_width, int image_height)
{
    constexpr T      zero = 0;
    constexpr T threshold = 100;

    for (int j = 0; j< (image_height * image_width); j += N) // N samples per loop
        chess_prepare_for_pipelining
        chess_loop_range(14,)
    {
        aie::vector<T, N> data_buf = aie::load_v<N>(img_in); img_in += N;
        aie::vector<T, N> data_out;

        aie::mask<N> m = aie::lt(threshold, data_buf);

        data_out = aie::select(zero, data_buf, m);

        aie::store_v(img_out, data_out); img_out += N;
    }
}
```

```
int16 find_max(const int16 *data, unsigned count)
{
  const v32int16 *pData = (const v32int16 *)data;
  v32int16 vec = pData[0];
  for (unsigned i = 1; i < count / 32; ++i) {
      vec = max32(vec,       0, 0x06040200, 0x0E0C0A08, 0x3210,
                  pData[i], 32, 0x06040200, 0x0E0C0A08, 0x3210);
  }
  vec = max32(vec, 0, 0x06040200, 0x0, 0x3210, 16, 0x06040200, 0x0, 0x3210);
  vec = max32(vec, 0, 0x0200,     0x0, 0x3210,  8, 0x0200,     0x0, 0x3210);
  vec = max32(vec, 0, 0x00,       0x0, 0x3210,  4, 0x00,       0x0, 0x3210);
  vec = max32(vec, 0, 0x0,        0x0, 0x3210,  2, 0x0,        0x0, 0x10);
  vec = max32(vec, 0, 0x0,        0x0, 0x3210,  0, 0x0,        0x0, 0x1);
  return ext_elem(vec, 0);
}
```

```
template <typename T, unsigned N>
T find_max(const T *data, unsigned count)
{
    aie::vector<T, N> vec = aie::load_v<N>(data);

    for (unsignd i = N; i < count; i += N)
        vec = aie::max(vec, aie::load_v<N>(data + i));

    return aie::reduce_max(vec);
}
```

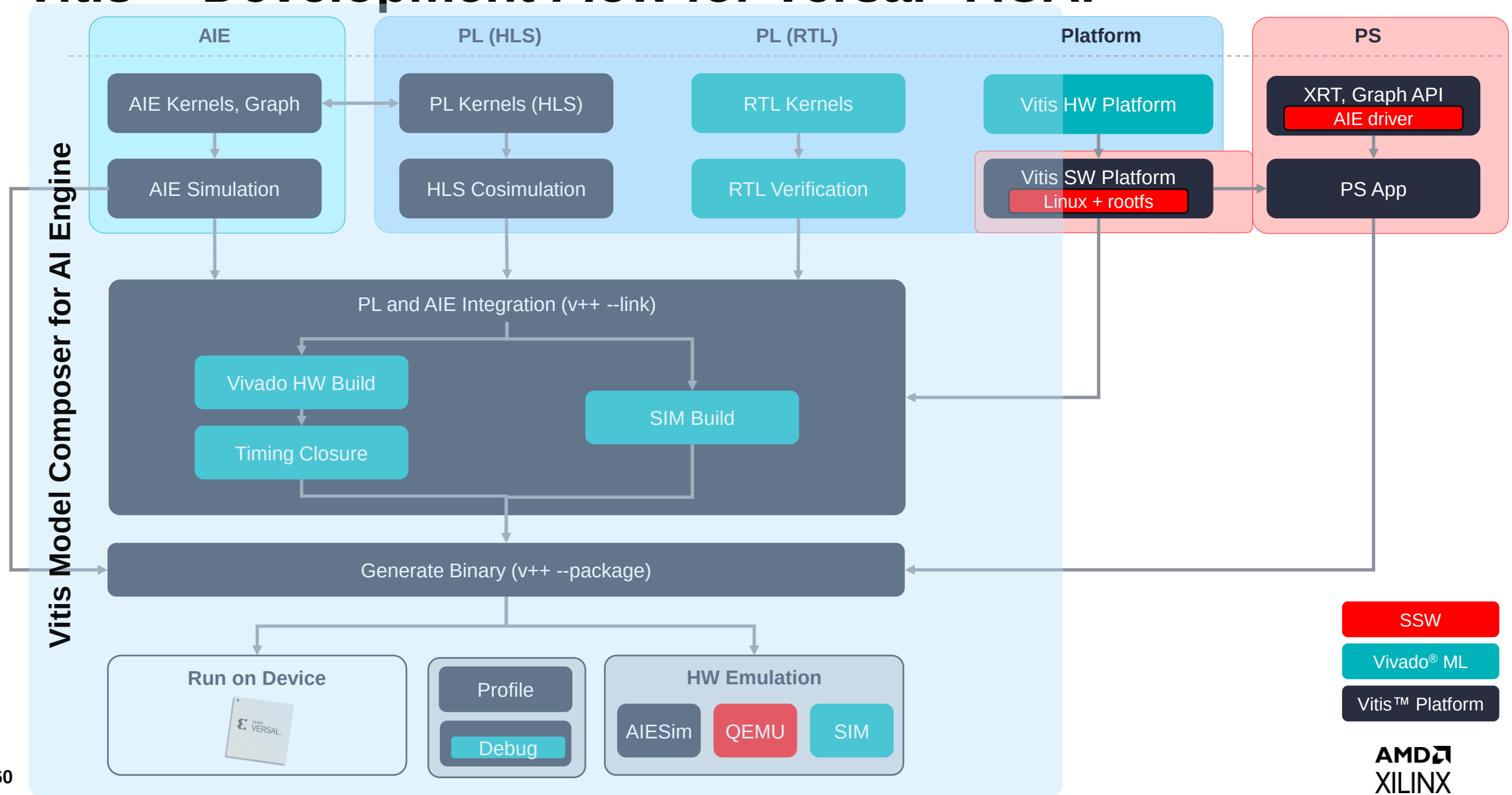
AI Engine API vs Intrinsics

- When to use AI Engine API vs Intrinsics

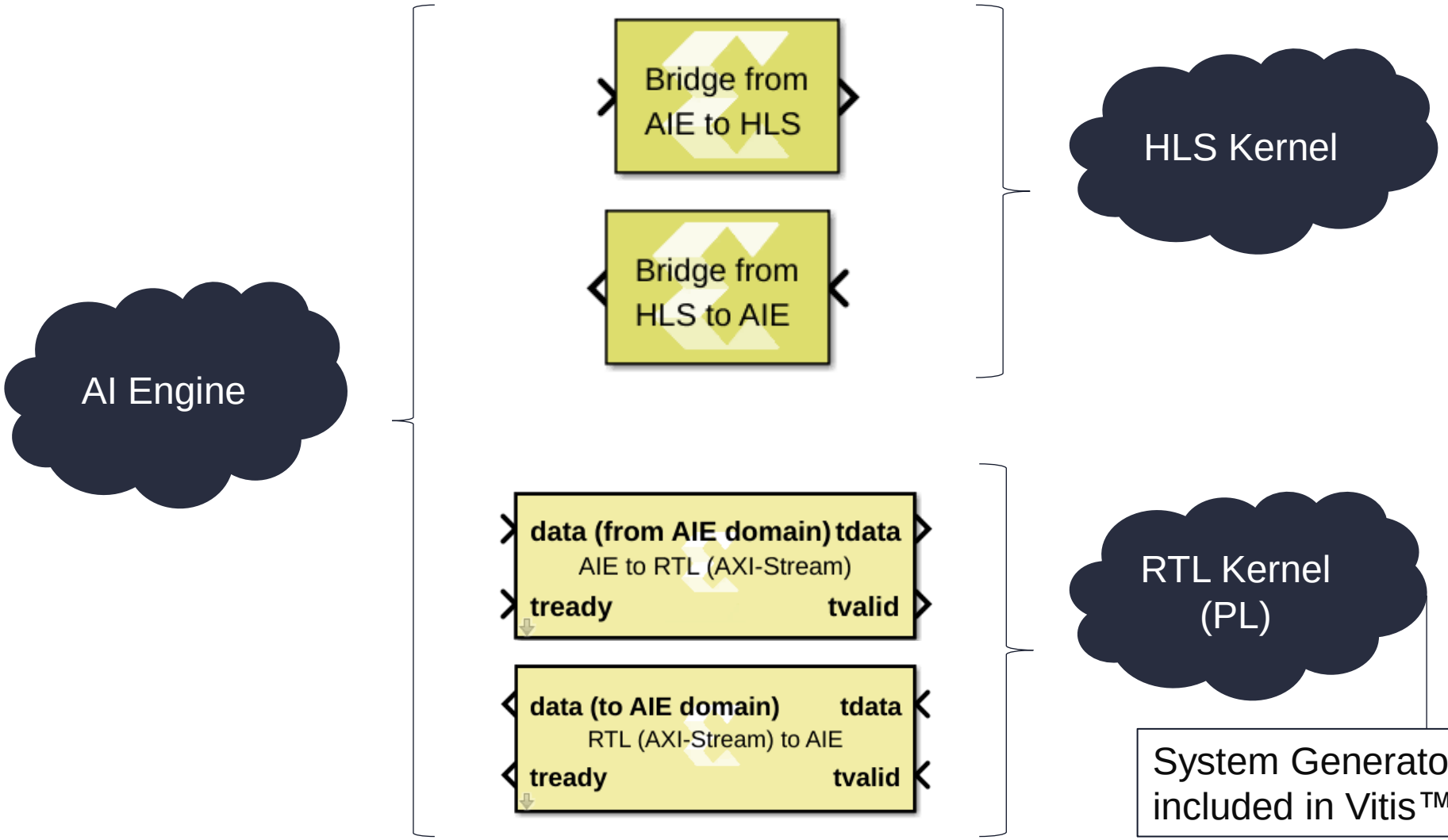
	AIE API	Intrinsics
Reduced Development Cycles	✓	
Maintainability	✓	
Portability	✓	
Performance	✓	✓
Direct Control		✓

Vitis™ Model Composer for AI Engine

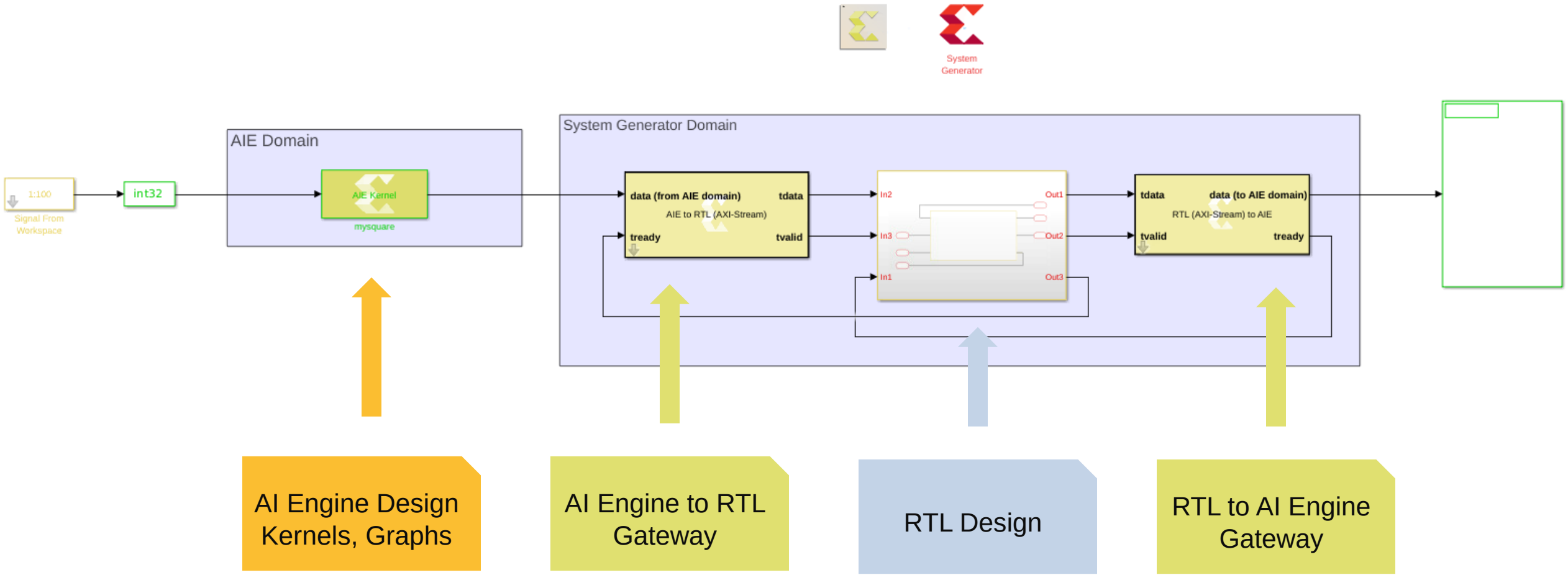
Vitis™ Development Flow for Versal® ACAP



Connectivity Between Domains



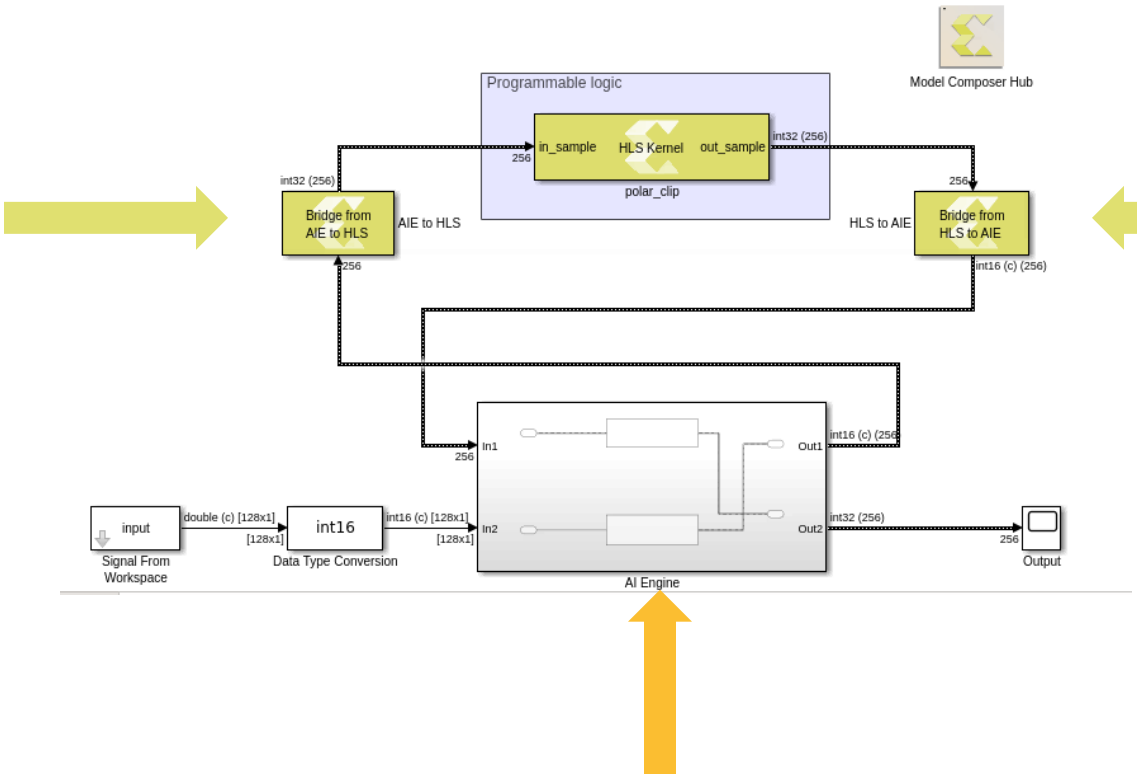
Mixed Domain Design: AI Engine + HDL (System Generator)



Mixed Domain Design: AI Engine + HLS

HLS Design

AI Engine to HLS Bridge



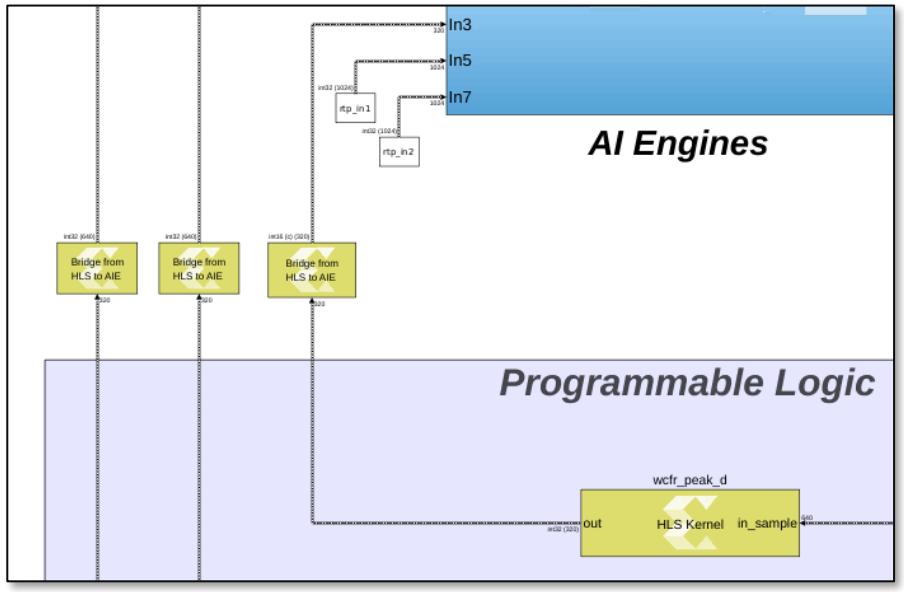
AI Engine to HLS Bridge

AI Engine Design Kernels, Graphs

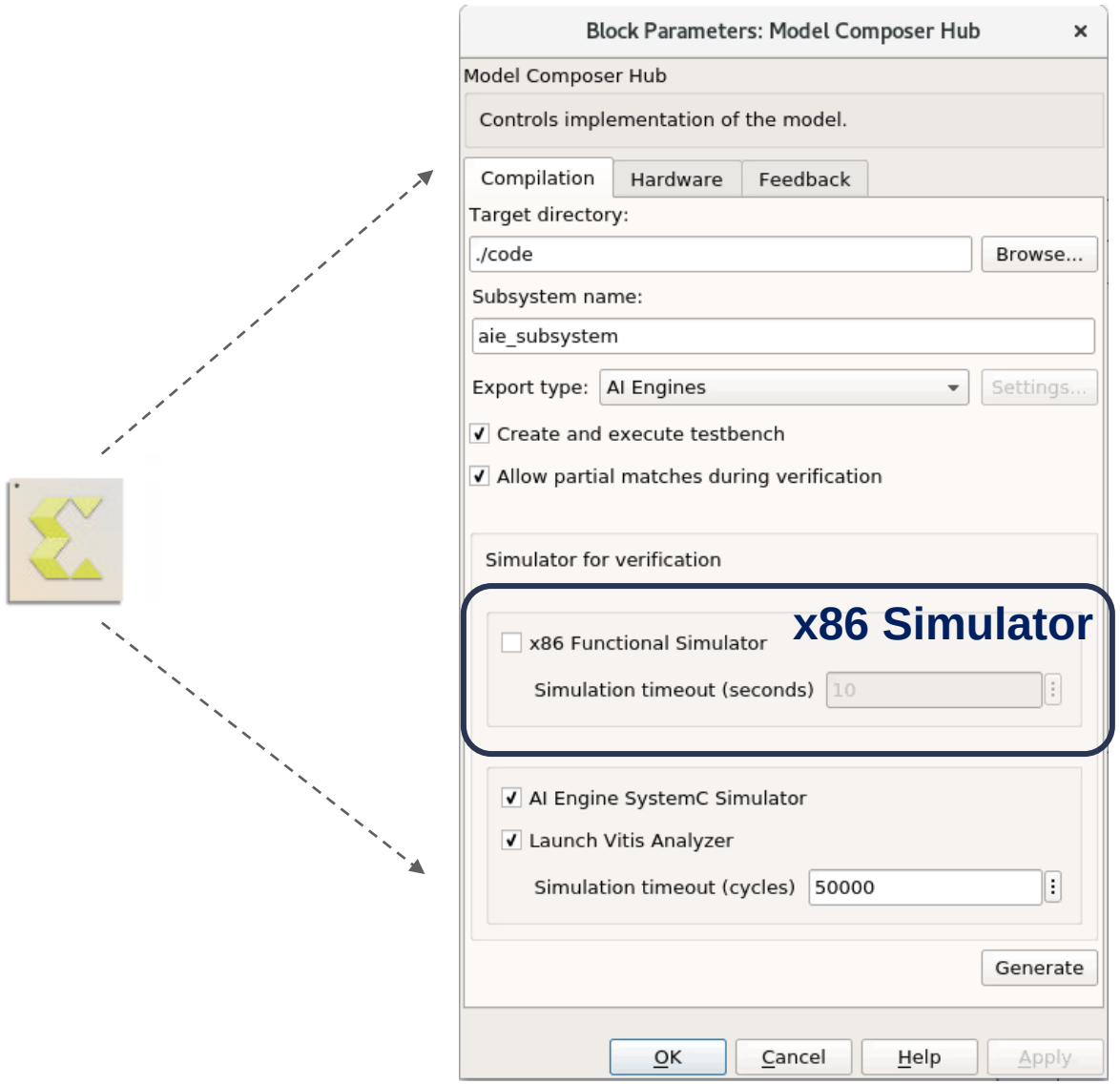
AMD XILINX

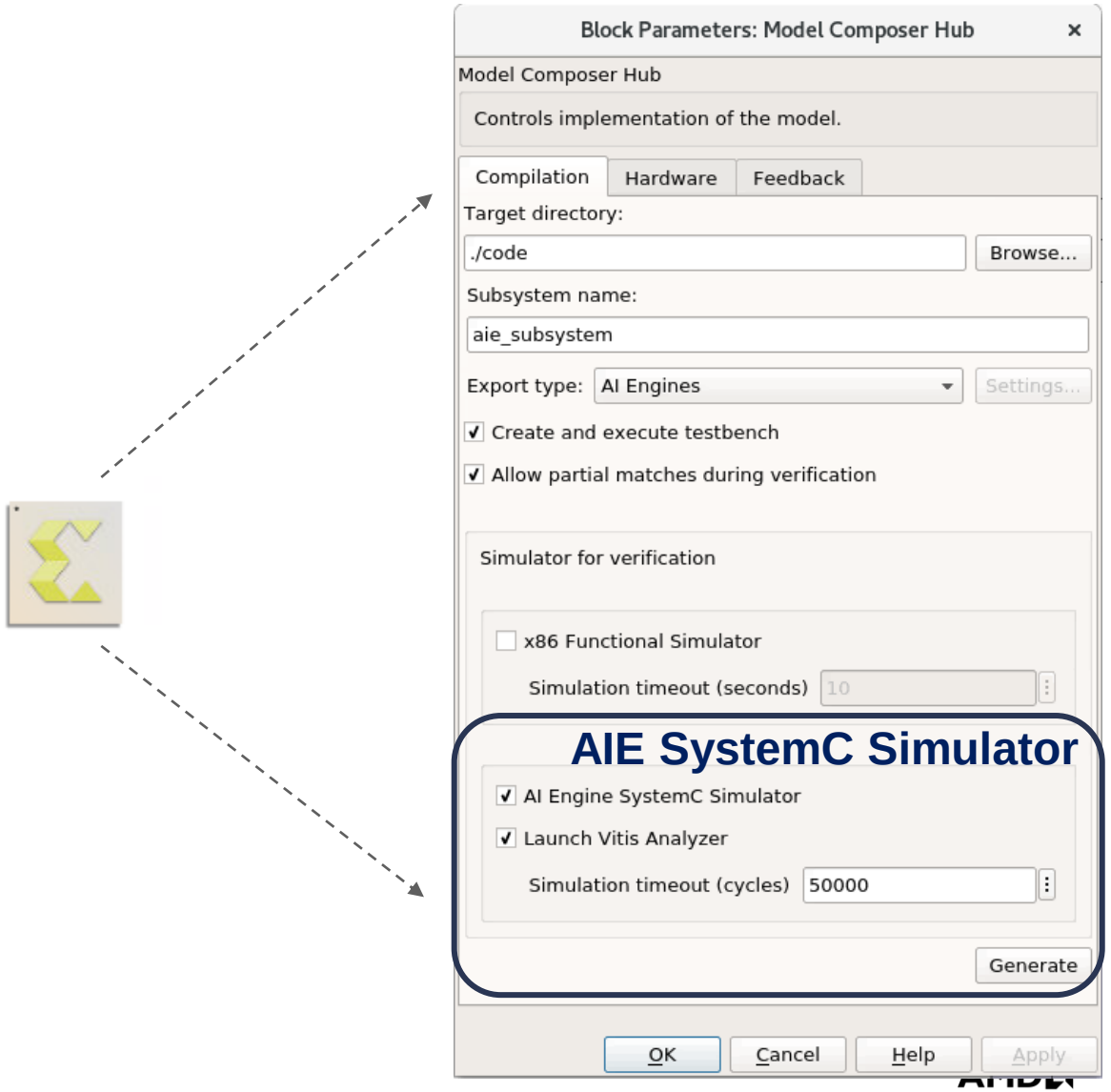
Model-Based Design – Simulation Support

Heterogeneous Functional Simulation:
AI Engine, HLS, and System Generator blocks all in the same design!

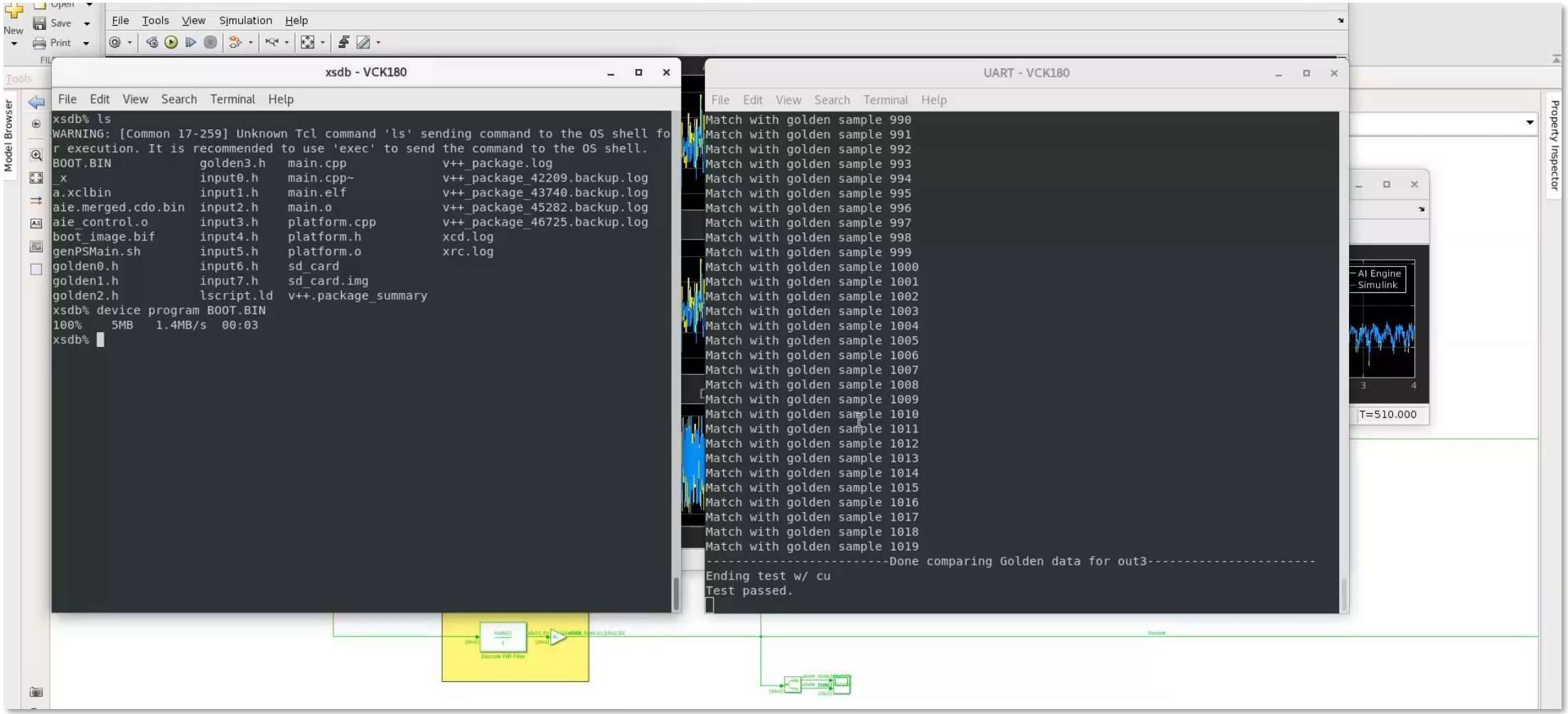


Model-Based Design – Simulation Support



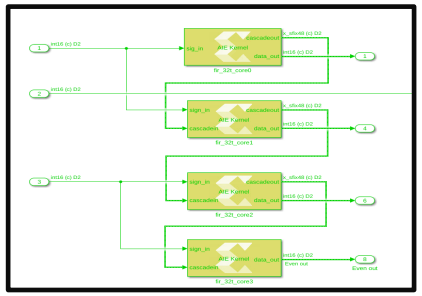


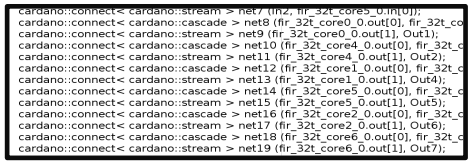
Hardware Co-Simulation



Code Generation

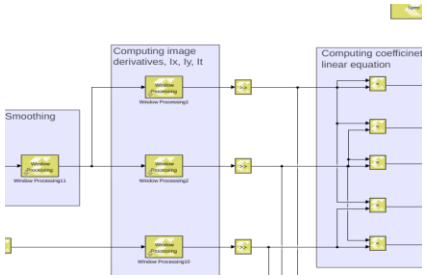
- Graph Code from AI Engine sub-system

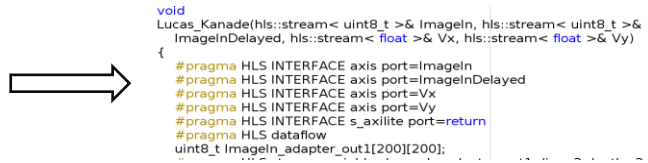




```
cardano::connect< cardano::stream > net7 (in2, fir_32t_core5_0.in[0]);
cardano::connect< cardano::cascade > net8 (fir_32t_core0_0.out[0], fir_32t_co
cardano::connect< cardano::stream > net9 (fir_32t_core0_0.out[1], Out1);
cardano::connect< cardano::cascade > net10 (fir_32t_core4_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net11 (fir_32t_core4_0.out[1], Out2);
cardano::connect< cardano::cascade > net12 (fir_32t_core1_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net13 (fir_32t_core1_0.out[1], Out4);
cardano::connect< cardano::cascade > net14 (fir_32t_core5_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net15 (fir_32t_core5_0.out[1], Out5);
cardano::connect< cardano::cascade > net16 (fir_32t_core2_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net17 (fir_32t_core2_0.out[1], Out6);
cardano::connect< cardano::cascade > net18 (fir_32t_core6_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net19 (fir_32t_core6_0.out[1], Out7);
```

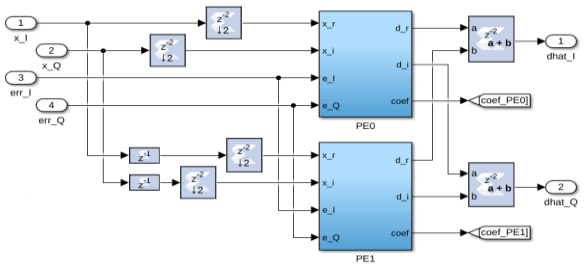
- HLS C++ code from HLS sub-system

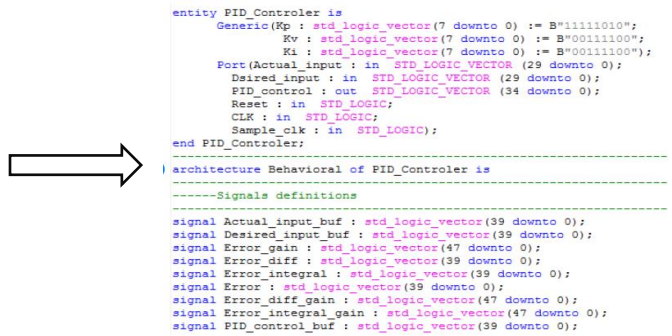




```
void
Lucas_Kanade(hls::stream< uint8_t >& ImageIn, hls::stream< uint8_t >&
   ImageInDelayed, hls::stream< float >& Vx, hls::stream< float >& Vy)
{
   #pragma HLS INTERFACE axis port=ImageIn
   #pragma HLS INTERFACE axis port=ImageInDelayed
   #pragma HLS INTERFACE axis port=Vx
   #pragma HLS INTERFACE axis port=Vy
   #pragma HLS INTERFACE s_axilite port=return
   #pragma HLS dataflow
   uint8_t ImageIn_adapter_out1[200][200];
```

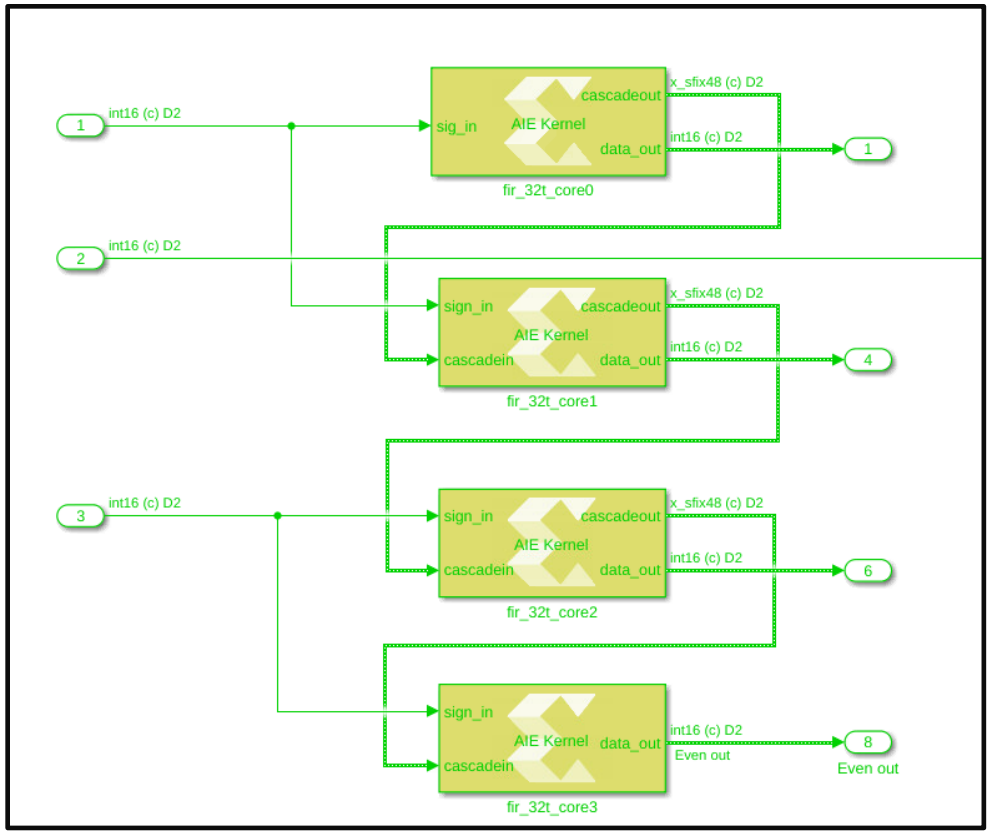
- RTL code from System Generator sub-system





```
entity PID_Controler is
      Generic(Kp : std_logic_vector(7 downto 0) := B"11111010";
              Kv : std_logic_vector(7 downto 0) := B"00111100";
              Ki : std_logic_vector(7 downto 0) := B"00111100");
      Port(Actual_input : in  STD_LOGIC_VECTOR (29 downto 0);
        Desired_input : in  STD_LOGIC_VECTOR (29 downto 0);
        PID_control : out  STD_LOGIC_VECTOR (34 downto 0);
        Reset : in  STD_LOGIC;
        CLK : in  STD_LOGIC;
        Sample_clk : in  STD_LOGIC);
end PID_Controler;
------------------------------------------------------------------------
architecture Behavioral of PID_Controler is
------------------------------------------------------------------------
------Signals definitions
------------------------------------------------------------------------
signal Actual_input_buf : std_logic_vector(39 downto 0);
signal Desired_input_buf : std_logic_vector(39 downto 0);
signal Error_gain : std_logic_vector(47 downto 0);
signal Error_diff : std_logic_vector(39 downto 0);
signal Error_integral : std_logic_vector(39 downto 0);
signal Error : std_logic_vector(39 downto 0);
signal Error_diff_gain : std_logic_vector(47 downto 0);
signal Error_integral_gain : std_logic_vector(47 downto 0);
signal PID_control_buf : std_logic_vector(39 downto 0);
```

Model Composer for AI Engine Development
Visualization (connectivity)



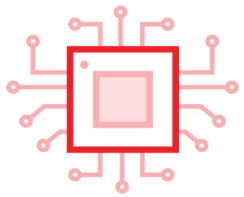
graph.h

```
cardano::connect< cardano::stream > net7 (In2, fir_32t_core5_0.in[0]);
cardano::connect< cardano::cascade > net8 (fir_32t_core0_0.out[0], fir_32t_co
cardano::connect< cardano::stream > net9 (fir_32t_core0_0.out[1], Out1);
cardano::connect< cardano::cascade > net10 (fir_32t_core4_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net11 (fir_32t_core4_0.out[1], Out2);
cardano::connect< cardano::cascade > net12 (fir_32t_core1_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net13 (fir_32t_core1_0.out[1], Out4);
cardano::connect< cardano::cascade > net14 (fir_32t_core5_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net15 (fir_32t_core5_0.out[1], Out5);
cardano::connect< cardano::cascade > net16 (fir_32t_core2_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net17 (fir_32t_core2_0.out[1], Out6);
cardano::connect< cardano::cascade > net18 (fir_32t_core6_0.out[0], fir_32t_c
cardano::connect< cardano::stream > net19 (fir_32t_core6_0.out[1], Out7);
```

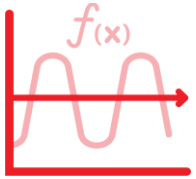
AI Engine Libraries

AI Engine Library Overview

- Enable short development cycles
  - Provide pre-packaged kernels & examples
  - Quick learning & adoption of AI Engine design
  - Remove kernel level coding for common functions
  - Portability across AI Engine architectures (AIE → AIE-ML)
  - Allow customers to focus on delivering proprietary algorithms
  - Align to Vitis™ acceleration library infrastructure

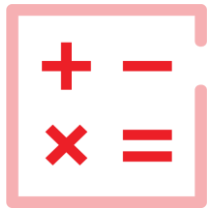


DSP



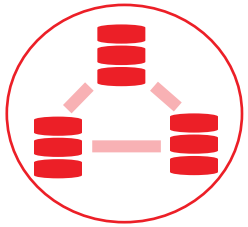
Math

Communications
(Restricted)



Linear Algebra

Vision & Image



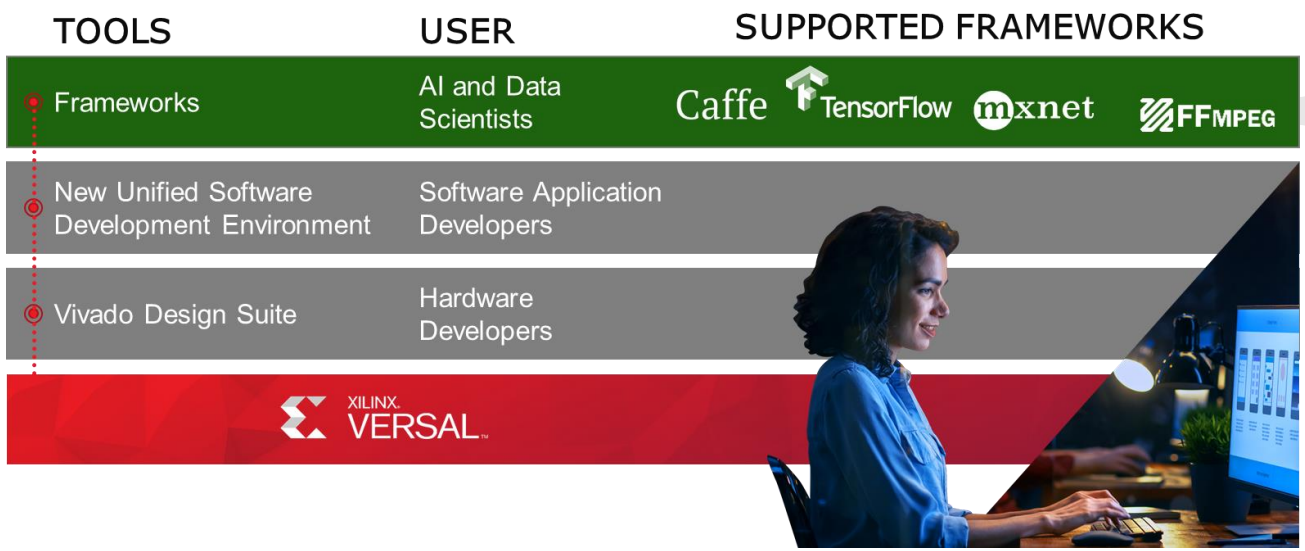
Data Movers

Vitis™ AI for Versal® ACAP

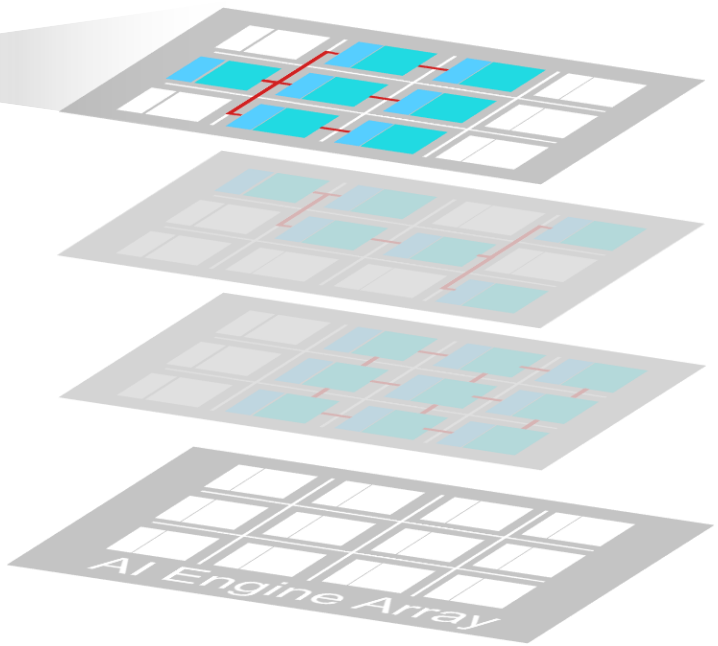
Machine Learning with AI Engine

AMD XILINX

Frameworks for Any Developer



Domain Specific Architecture
(e.g. AI Inference)



Architecture Overlay

Data Flow
w/ Xilinx libraries

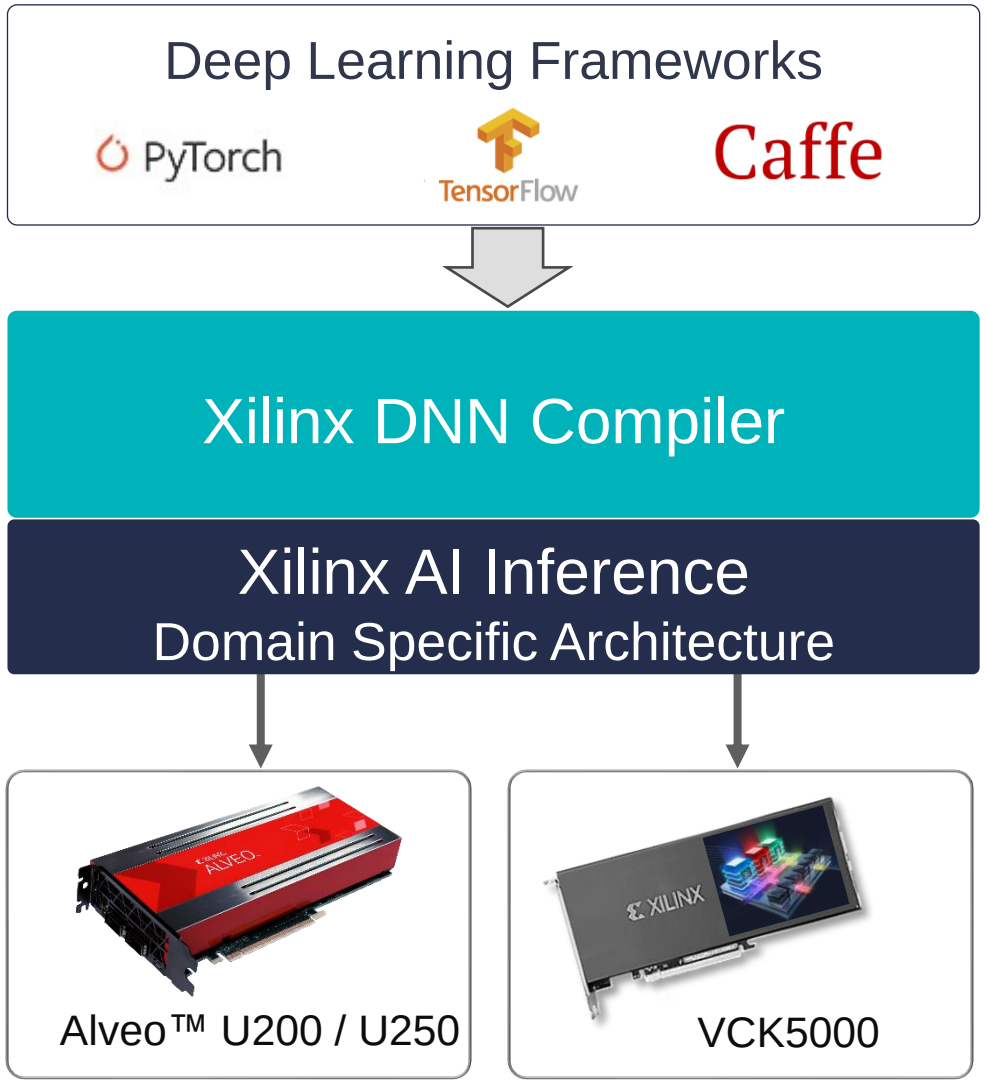
Kernel Program
Data Flow w/ user defined libraries

Target Domain Specific Architectures – No HW Design Experience Required

AMD XILINX

Accelerating AI Inference in the Data Center

1. User works in Framework of choice
   - Develop & train custom network
   - User provides trained model
2. Xilinx DNN Compiler implements network
   - Targets AI Inference Domain Specific Architecture
   - Quantize, merge layers, prune
   - Compile to AI Engines
3. Scalable across hardware targets
   - Start with Alveo today



For more information on Vitis AI

Versal® ACAP Design Partition
(AI Engine vs DSP and Programmable Logic)

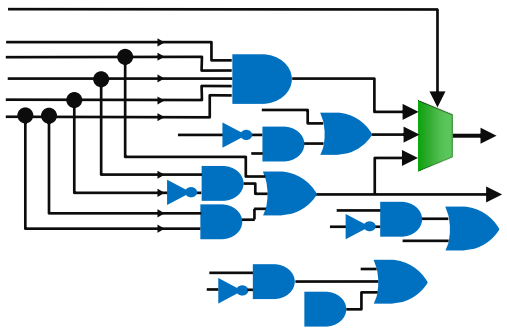
AMD XILINX

Intelligent Engines in Versal® ACAP → AI and DSP Engines

FPGA Fabric DSP

LUT and Memory

RTL Entry



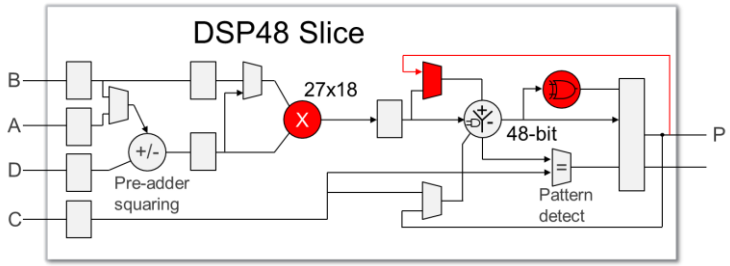
Increasing Capability

DSP48E2 Slice

Hardened MULT & ADDERS

ACC = ACC + (A × B)

RTL Entry

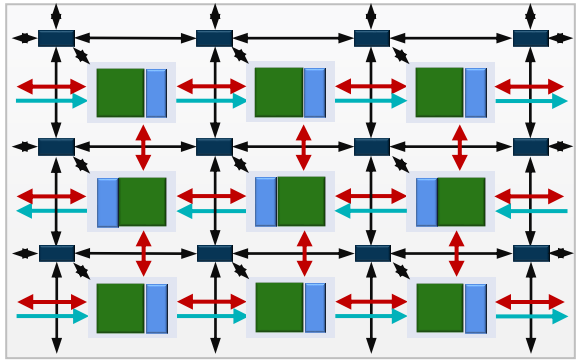


Increasing Capability

AI Engine(1) 2D Array

VLIW and SIMD Architecture

C/C++ Programmable

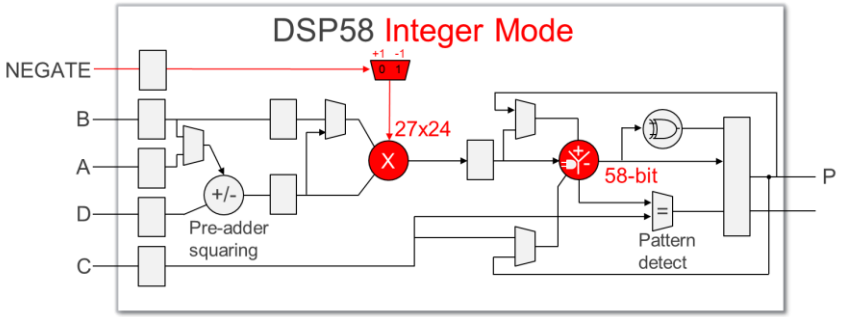


Increasing Capability

DSP Engine

Additional Features

RTL Entry



SPFP32 Native support

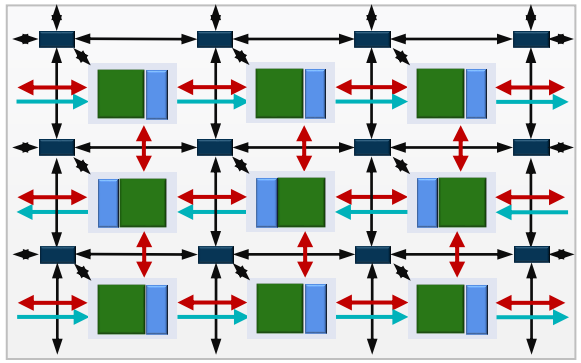
AMD XILINX

AI Engines and DSP Engines

AI Engine(1) 2D Array

VLIW and SIMD Architecture

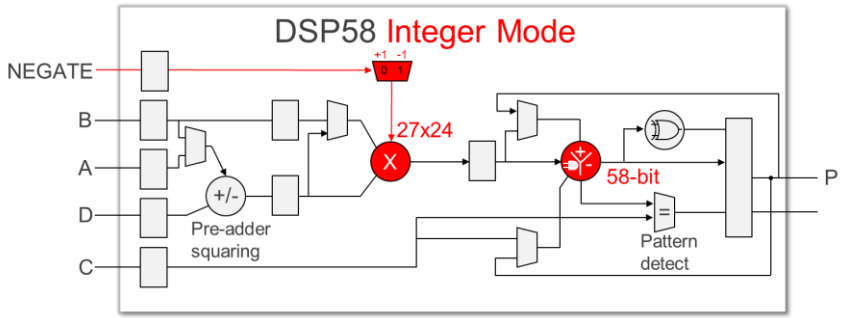
C/C++ Programmable



DSP Engine

Additional Features

RTL Entry



Why AI Engine?

- Massive compute performance
- S/W programmable (C/C++)
- Fast compile – increase productivity

Why DSP Engine?

- Existing RTL/HLS IP usage
- Additional features not available in AI Engine, e.g. 58-bit Logic Unit
- Pre/Post processing to/from AI Engine

Why both?

- Versal® ACAP accelerates the complete application
- AI Engine efficiency + PL flexibility (inc. DSP)

AMD XILINX

Designs with Versal® AI Core Device- DSP & PL v AI Engine

Key System Level Considerations

(Design Partition at the system level – not in isolation, i.e. not DSP vs AIE)

Compute

Compute/Func types
- Multiply
- Multiply & Accumulate
- FFT
- FIR etc

Data Widths
- AIE native 8/16/32 bit
- DSP & PL – variable

Vectorizable
- Can the dataflow be vectorize

Memory

Custom Mem Hierarchy
- URAMs & BRAMs
- LUTRAMs
- AIE Data Memory
  - 32 & 128KB
- Maybe external DDR

Type of Mem Access
- Random access
- Structured v Unstructured Data
- Data reuse or sharing

Throughput / Latency

Throughput or Latency
- AIE can provide both
- Most applications lean towards one or the other
- Decisions on partitioning likely to affect those parameters and need to be understood

Data Movement

Data movement key factor
- Data into and out of AI Engine array
- Data around the array (Cascade, AXI etc,)
- PL interface for Data
- NoC for trace & debug

Data Movement Libraries
- Must simplify the problem for users via utilities and libraries

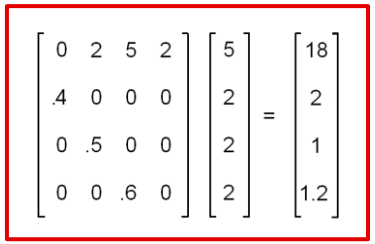
Power

AIE brings performance
- Perf/watt
- Perf/watt/mm^2

Managing these key system level considerations enables Versal AI Core meet expectations and deliver the promised perf/watt

AMD XILINX

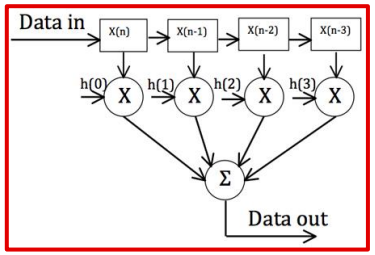
Representative AI Engine Algorithms



Linear Algebra

Matrix-Matrix Multiplication

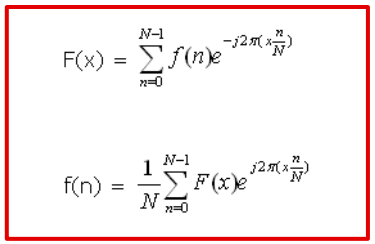
Matrix-Vector Multiplication



Convolution

FIR Filters

2-D Filters



Transforms

Fast Fourier Transforms (FFTs/iFFTs)

Discrete Cosine Transforms (DCT)

FIR Filter Micro Benchmarks

Filter 1:
- 32-tap real filter
- Symmetric
- 16-bit data & Coeffs
- 1 Gsps input sample rate

	AIE	DSP58
Processor/Operator Performance	32 GMACs @ 16-bits	0.5 GMACs @ 27x25 bits
Resource Usage 16 GMACs (16 bits real)	0.5 AI Engines	32 DSP58s

Filter 2:
- 8-tap real filter
- Non-symmetric
- 24-bit data & Coeffs
- 500 Msps input sample rate

	AIE	DSP58
Processor/Operator Performance	32 GMACs @ 16-bits 8 GMACs @ 32-bits	0.5 GMACs @ 27x25 bits
Resource Usage 4 GMACs (24 bits real)	0.5 AI Engines	8 DSP58s

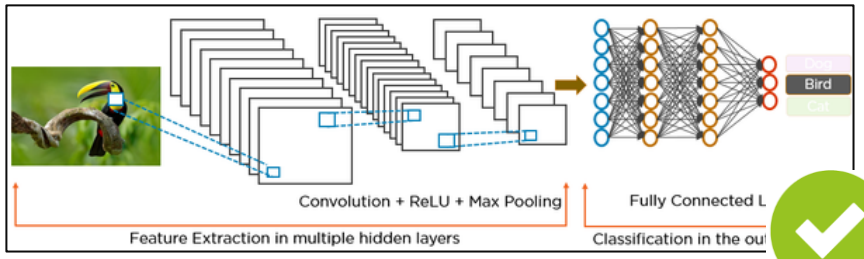
Filter 3:
- 32-tap complex filter
- 16-bit complex, data & Coeffs
- 1 Gsps input sample rate

	AIE	DSP58
Processor/Operator Performance	32 GMACs @ 16-bits (real) 8 GMACs @ 16-bits (complex)	2 DSPs in DSPCPLX slice 0.5 GMACs @ 16x16 bits
Resource Usage 32 GMACs (16 bits complex)	4 AI Engines	64 DSP58s

AMD XILINX

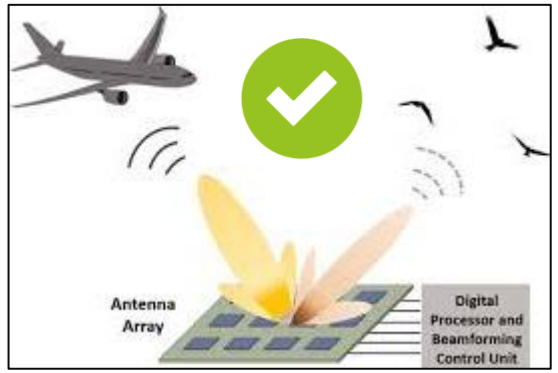
Representative Applications

AI Engine – Market / Application Diversity

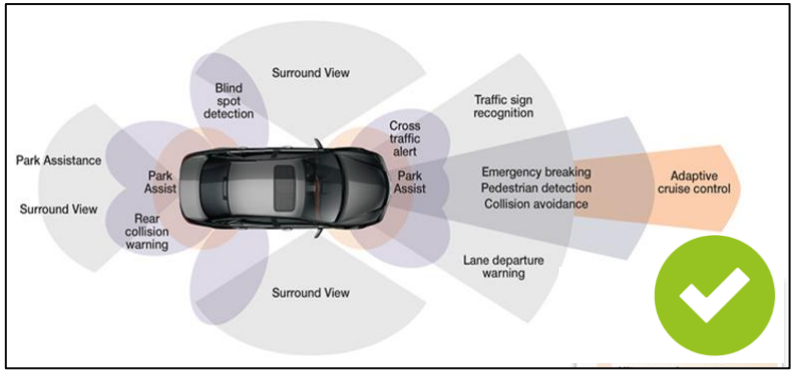


Data Center

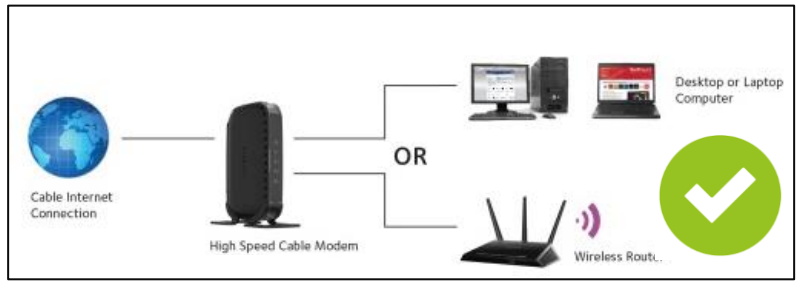
5G Wireless



Aerospace & Defence



ADAS/AD



Wired

Test & Measurement

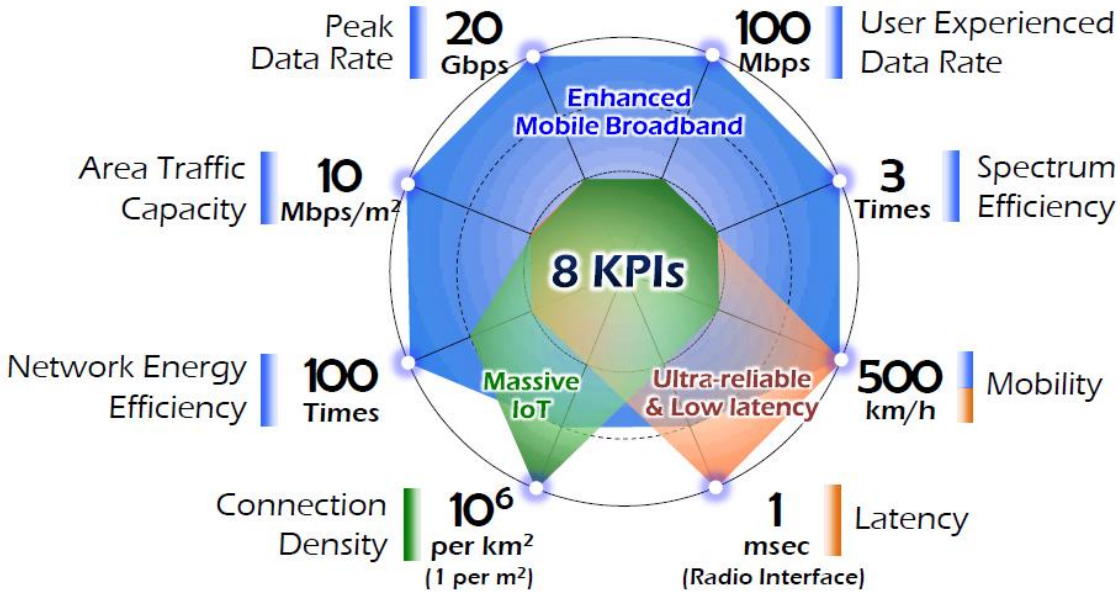
Fintech

AMD XILINX

Market Requirements and Trends: Wireless 5G

5G Complexity is 100X that of 4G

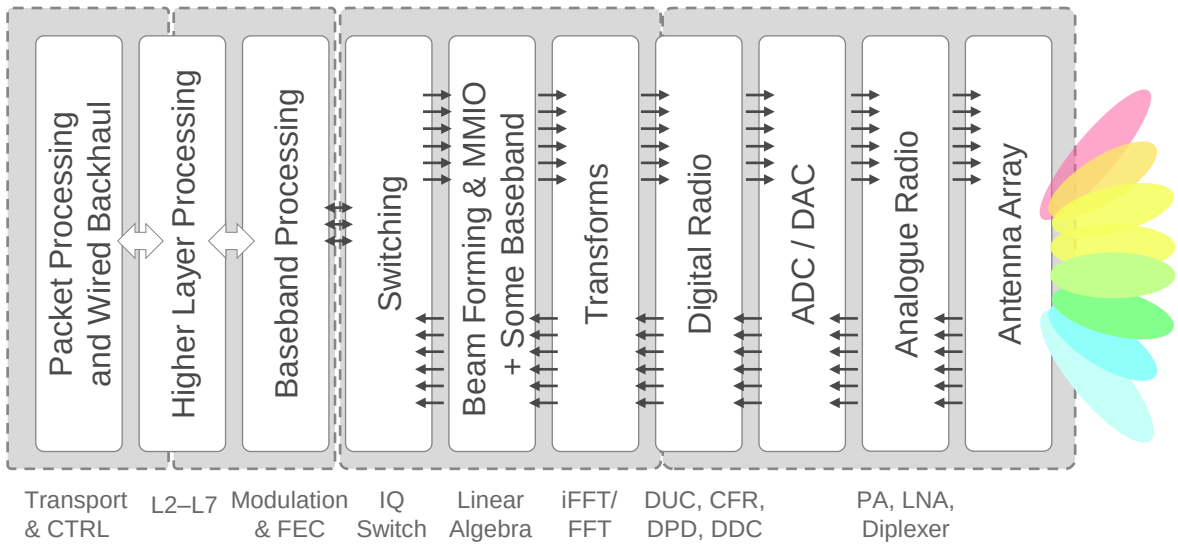
Still Evolving Standard



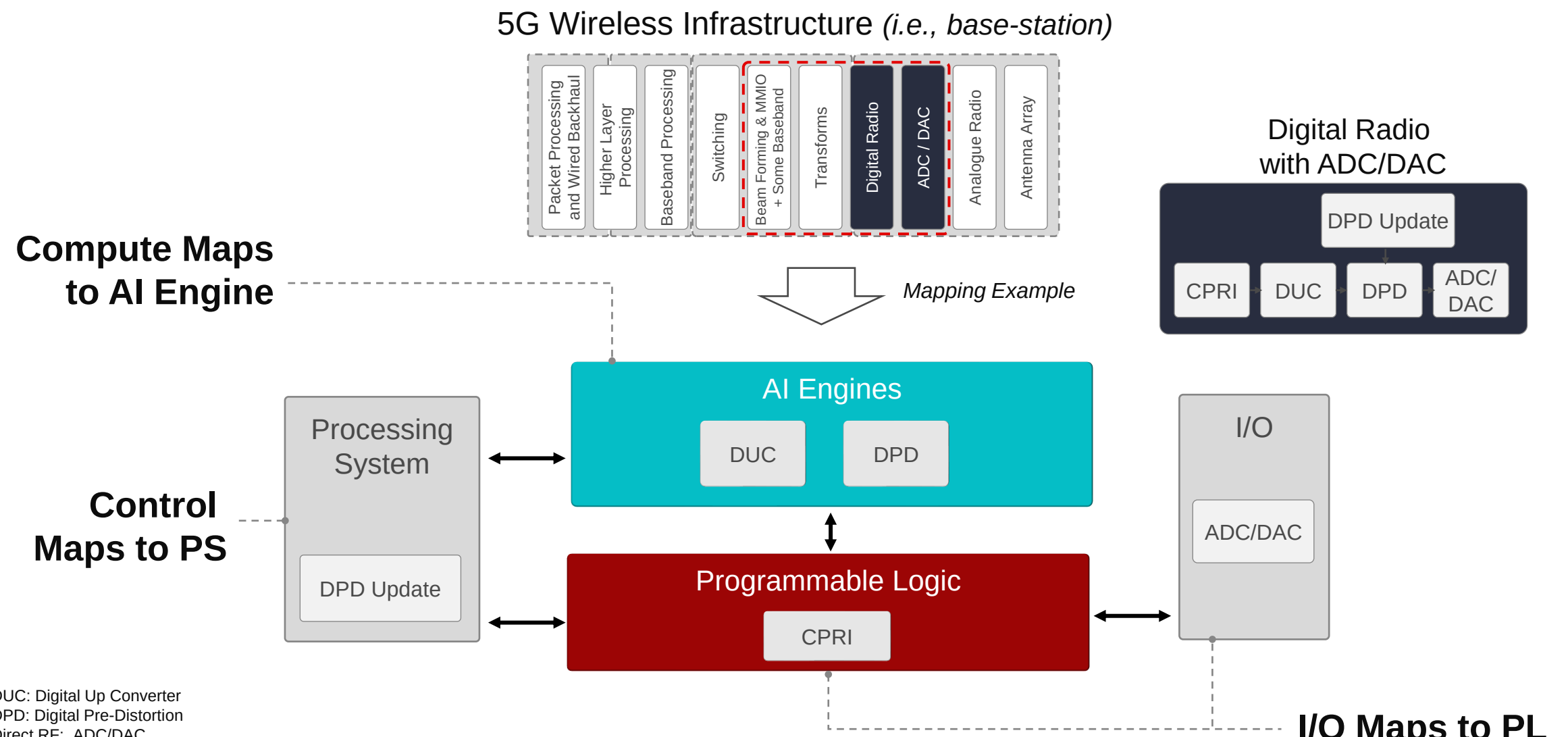
ETRI RWS-150029,
5G Vision and Enabling Technologies: ETRI Perspective 3GPP RAN Workshop
Phoenix, Dec. 2015
http://www.3gpp.org/ftp/tsg_ran/TSG_RAN/TSGR_70/Docs

New Technologies in 5G

- Massive MIMO
- Multiple antenna, frequency bands
- Changing functional partitioning



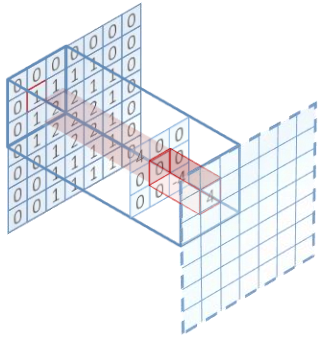
5G Wireless on Versal® ACAP



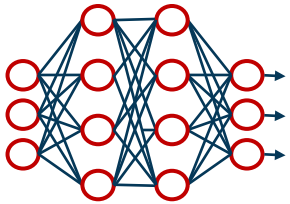
1: DUC: Digital Up Converter
2: DPD: Digital Pre-Distortion
3: Direct RF: ADC/DAC
4: CPRI: Common Public Radio Interface

AI Inference on Versal® ACAP

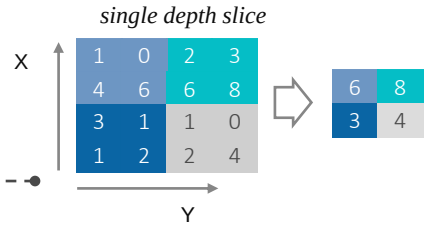
Convolutions



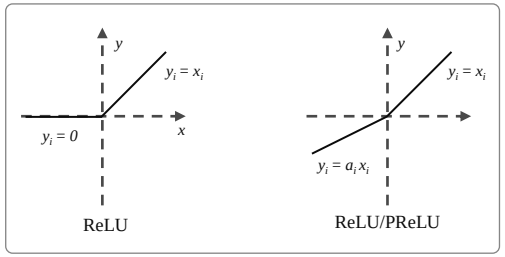
Fully Connected Layers



Pooling



Activations



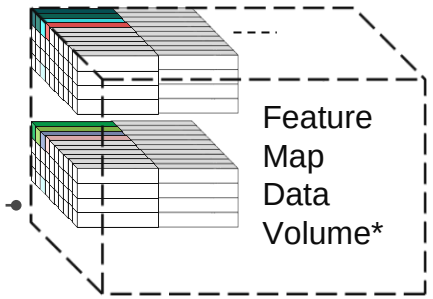
Processing System

AI Engines

Programmable Logic

I/O (GT, ADC/ DAC)

- Video
- Genomics
- Storage
- Database
- Network IPS
- Risk modeling



Custom Memory Hierarchy

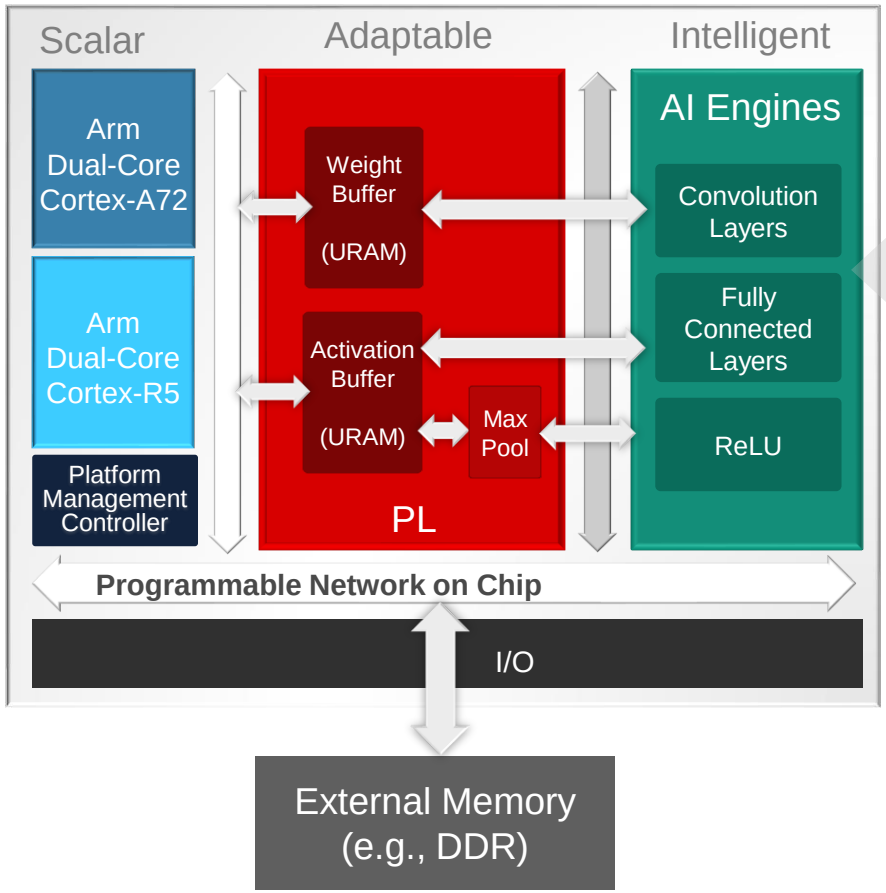
*Figure credit: https://en.wikipedia.org/wiki/Convolutional_neural_network

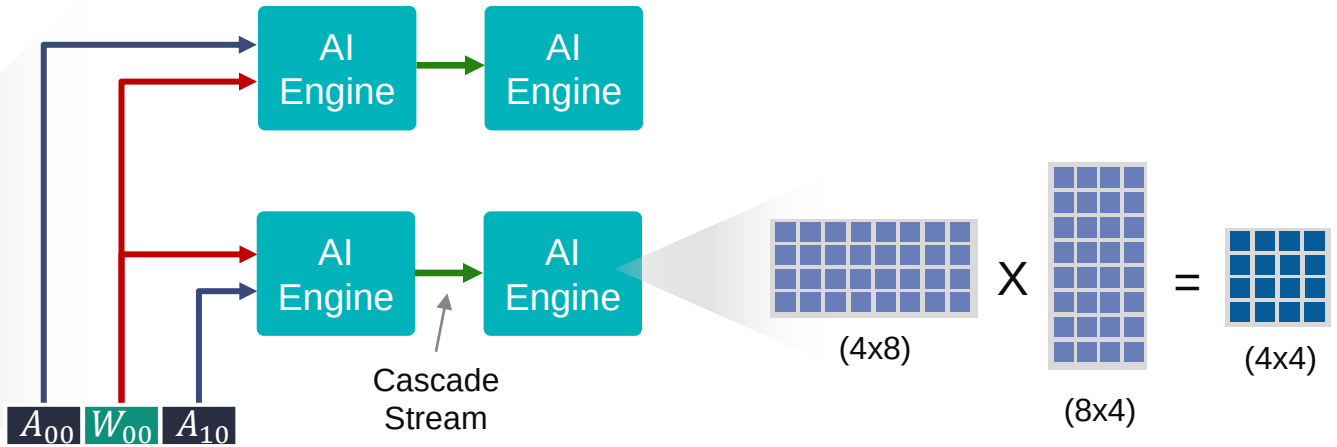
AMD XILINX

AI Inference Mapping on Versal® ACAP

A = Activations
W = Weights

$$\begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{bmatrix} = \begin{bmatrix} A_{00} \times \boldsymbol{W_{00}} + A_{01} \times W_{10} & \dots \\ A_{10} \times \boldsymbol{W_{00}} + A_{11} \times W_{10} & \dots \end{bmatrix}$$



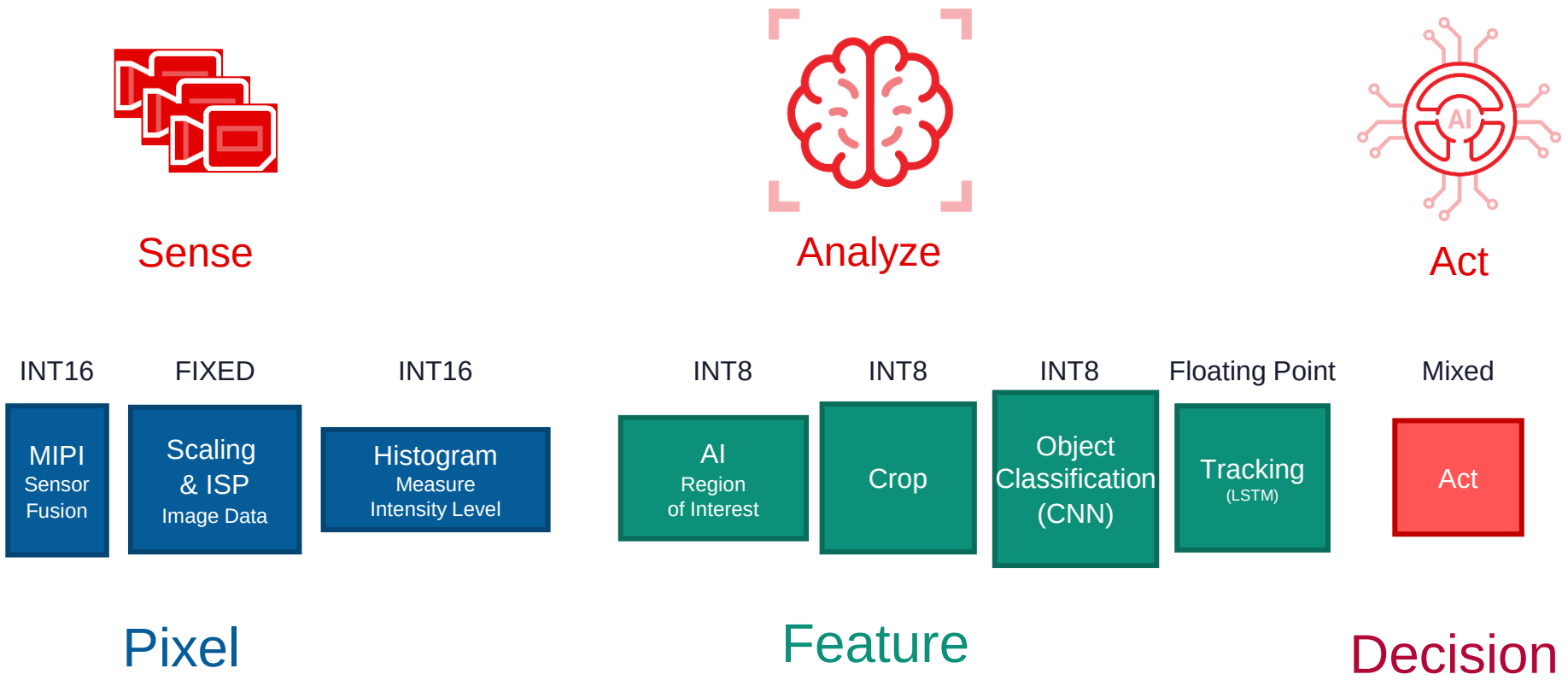


- Custom memory hierarchy
  - Buffer on-chip vs off-chip; Reduce latency and power
- Stream Multi-cast on AI interconnect
  - Weights and Activations
  - Read once: reduce memory bandwidth
- AI-optimized vector instructions (128 INT8 mults/cycle)

AMD XILINX

Partitioning Example: Vision Application

Example Mapping for Vision Application to Versal® AI Core Device

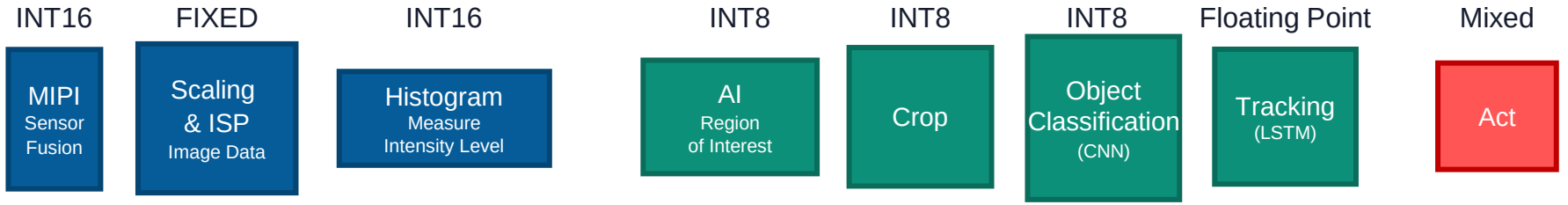


Custom Data-Flow, Custom Precision, Specific Compute

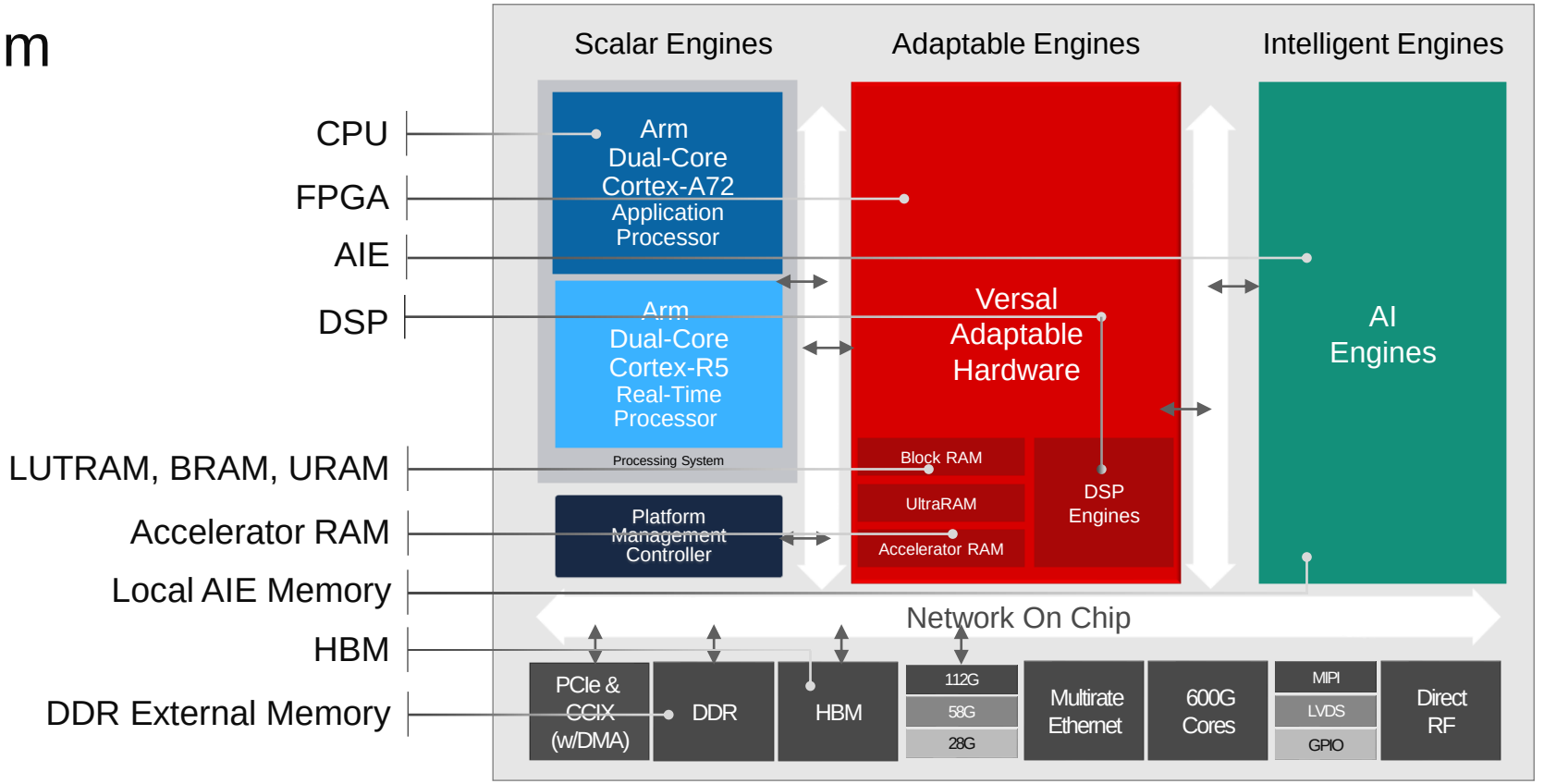
AMD XILINX

Match Application Components to Heterogenous ACAP

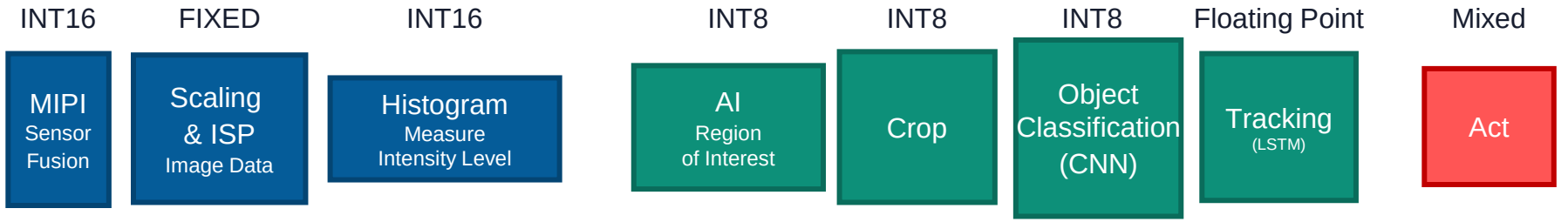
Key is to correctly map into each domain!



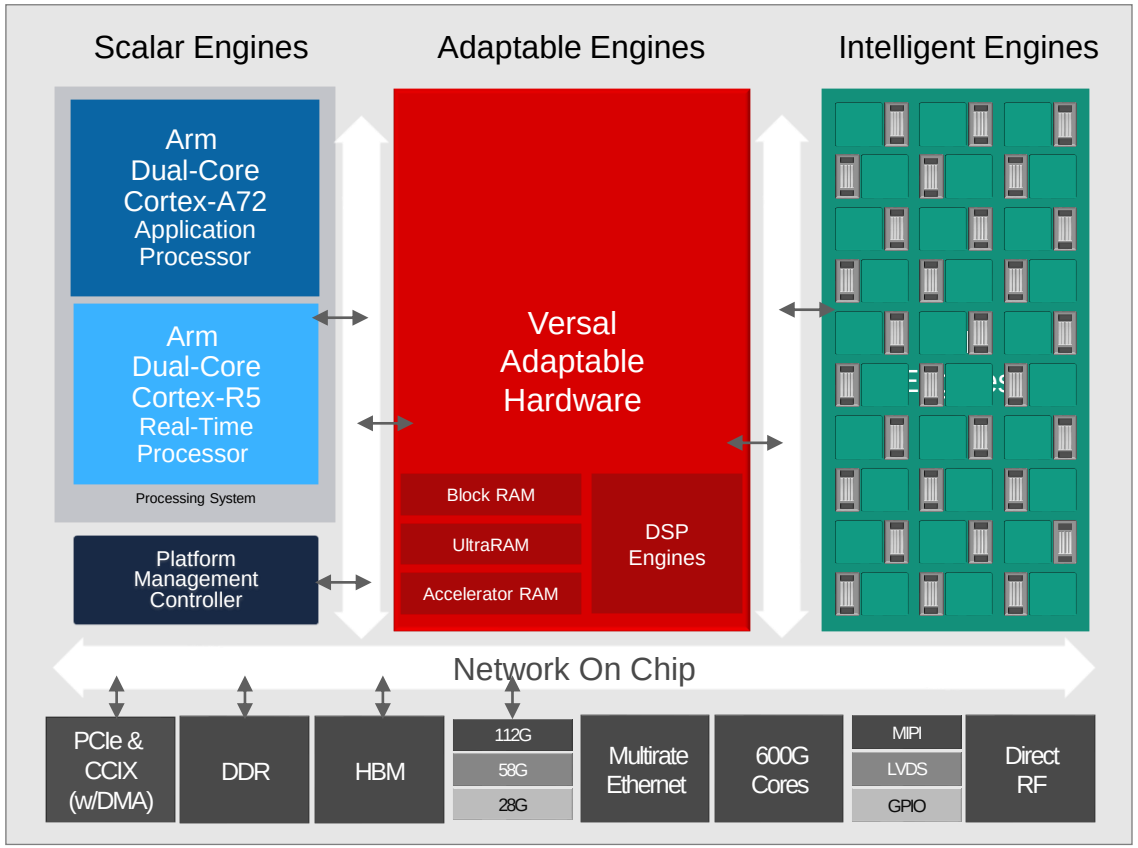
- Heterogenous system
  - Compute engines
  - Adaptable memory
  - Flexible interconnect



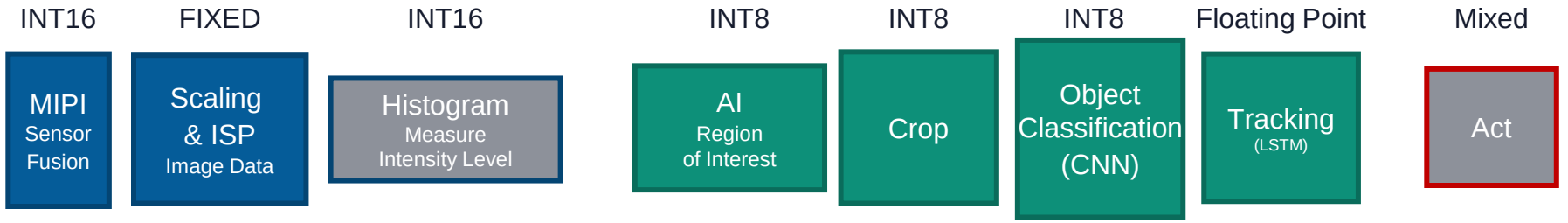
Match Application Components to Heterogenous ACAP



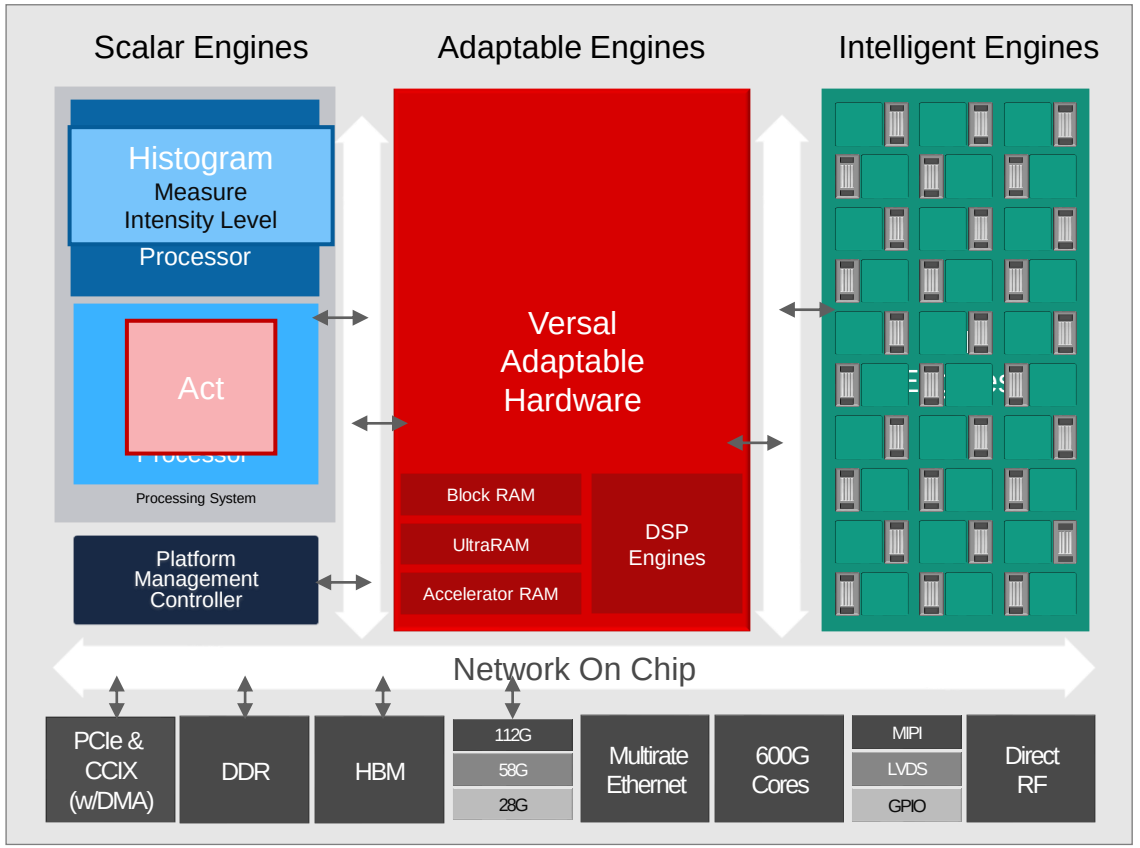
- **Scalar Engines**
  - SW programmable sequential processing
  - Complex algorithms and decision making
  - Initial configuration and real-time management of the system
  - Data transfers between DDR and AI Engine array
  - Runtime configuration



Match Application Components to Heterogenous ACAP

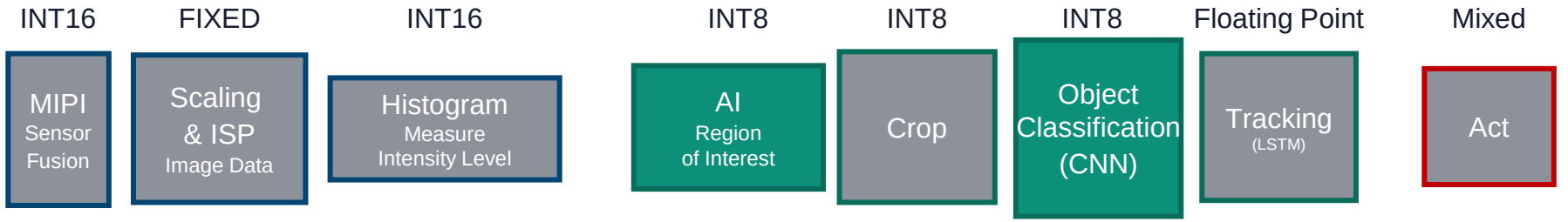


- **Adaptable (PL) & DSP Engines**
  - Flexible parallel compute, custom I/O and memory
  - Hardware acceleration of compute-intensive functions
  - Data processing that exploits heavy data parallelism
  - Deeply pipelined (video codecs),
  - Bit manipulations (AES/SHA)
  - Wide datapath
  - Various bitwidth compute

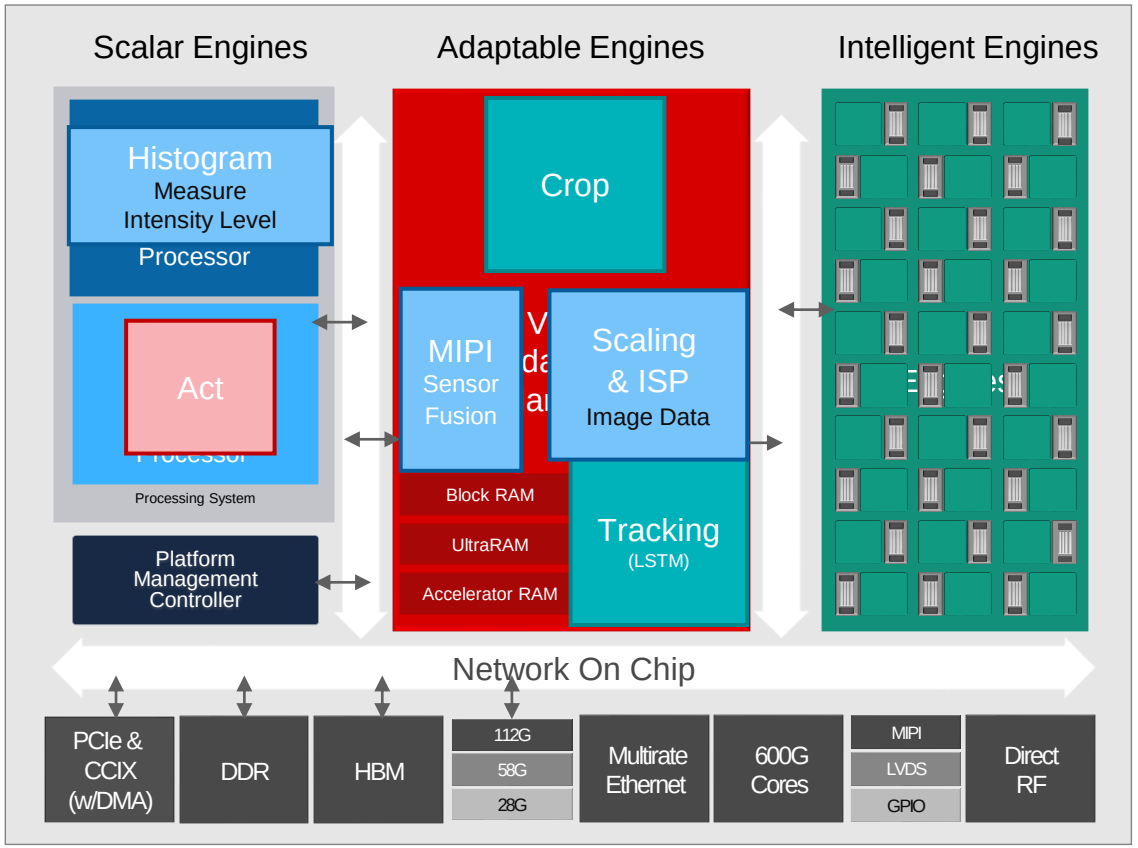


AMD XILINX

Match Application Components to Heterogenous ACAP



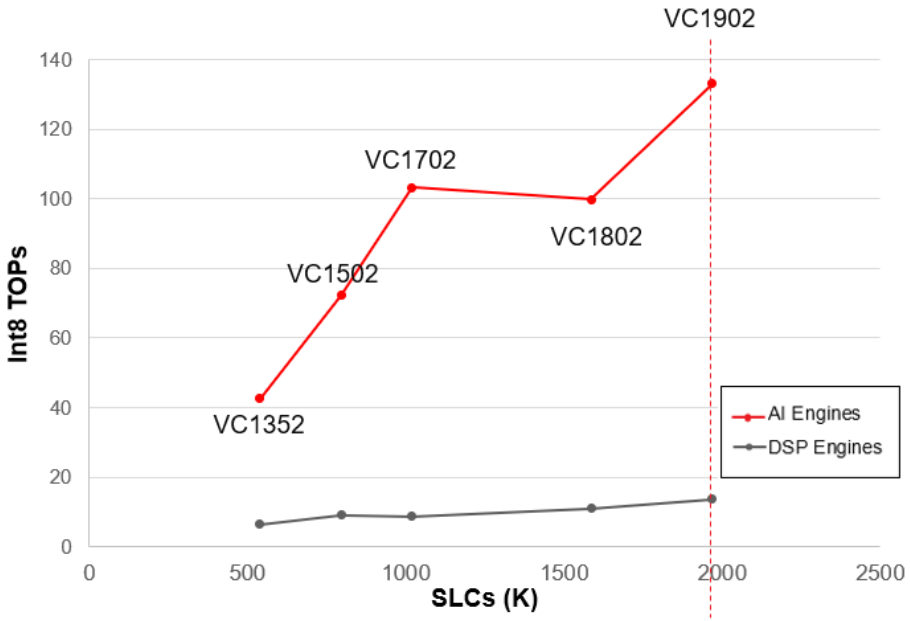
- **AI Engines**
  - Acceleration of vectorized compute intensive signal processing:
  - Linear Algebra
  - Convolution
  - Tensor operations
  - Transforms (FFT,DCT)
  - 8/16/32 bit integers
  - Single Precision Floating-Point
  - Real or Complex Operations

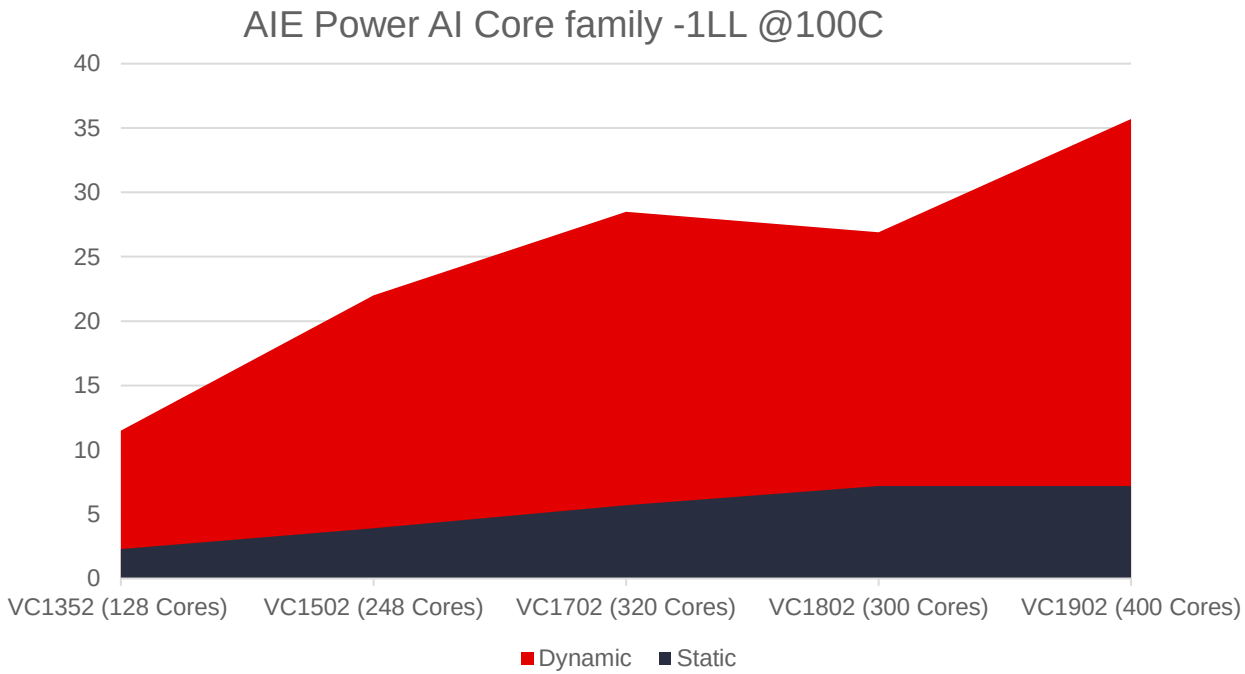


AI Engine Power & XPE

AI Engine Power

- Up to 40X Performance Increase for int8



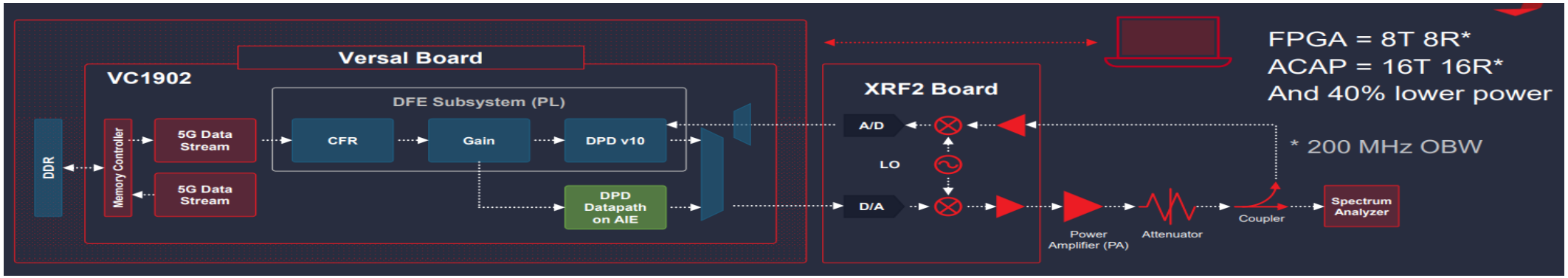


- Power Per AIE tile 80 to 110mW
  - Includes memory and interconnect
  - Typical Vector Load is 30% » 70% and Memory R/W rate 10% » 30%
- AI Engines Enable Massive Compute Gains!
  - With only minimal power increase

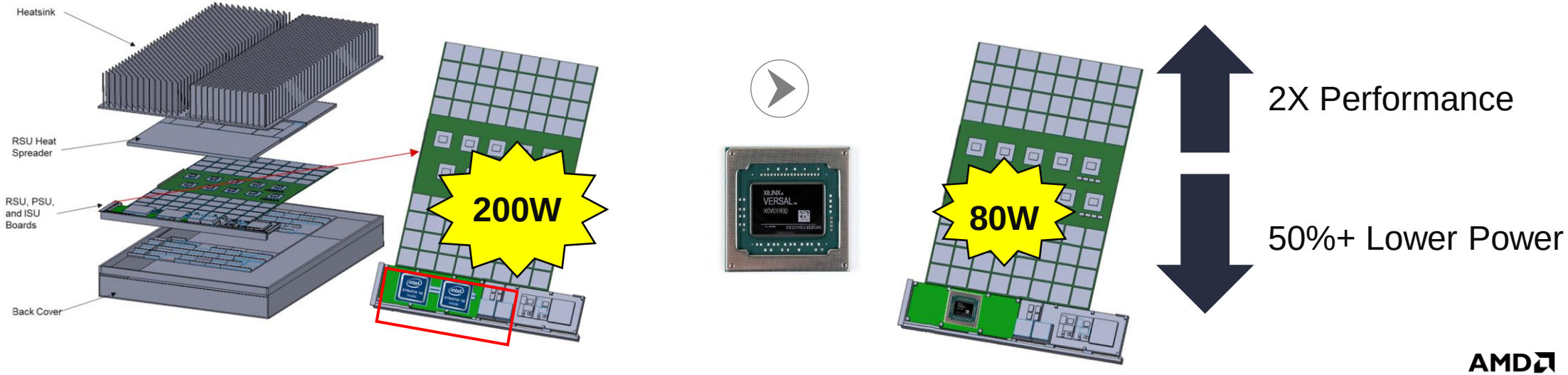
AMD XILINX

Versal ACAP with AI Engine → Power Reduction

- Virtex® UltraScale+™ FPGA 8T 8R 200Mhz -> Versal® ACAP 16T 16R 200Mhz
  - 2x Performance @40% lower power



- Beamformer Example



AMD XILINX

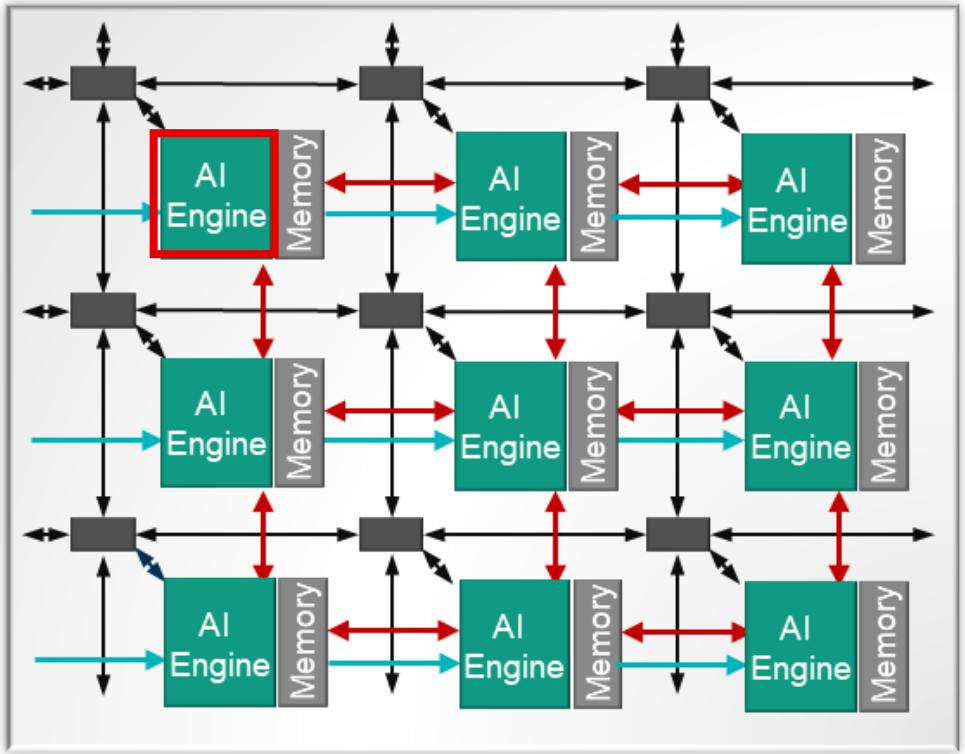
AIE Power Model

- AIE dynamic power is modeled as sum of several components
  - AI Engine Power (Core and Memory)
  - Interconnect Power
  - PL and programmable NoC Interface Power
  - Top Level Power
- Good correlation with silicon when using accurate design parameters
  - Internal correlation based on extracting design info from simulation

AI Engine Power: Engine

- AI Engine Power is a function of
  - Number of cores used in design
  - Kernel type
    - INT8, INT16, FP
  - Average “Vector Load”
    - Vector Load is the portion of time a core spends performing power-intensive vector operations
    - This is different from the runtime ratio used by AIE compiler, which is used for kernel mapping
    - Typical “Vector Load” in internal benchmarks is 30%-70%
    - For early estimation, the recommendation is to use the default of 50%

AI Engine Core			Data Memory		Interconn
Cores	Kernel Type	Vector Load	Memory Banks	Memory R/W Rate	AI Engine Tiles	Int
100	Int8	50%	800	20%	100	
	Int8	50%	0	20%	0	

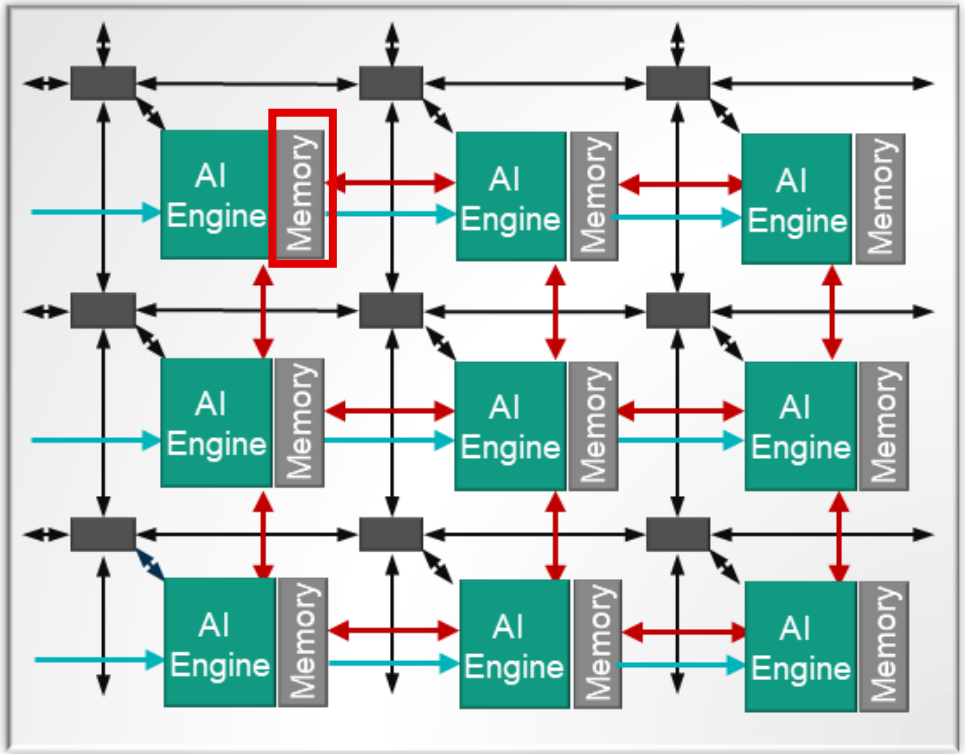


AMD XILINX

AI Engine Power: Local Memory

AI Engine Core			Data Memory		Interconn
Cores	Kernel Type	Vector Load	Memory Banks	Memory R/W Rate	AI Engine Tiles
100	Int8	50%	800	20%	100
	Int8	50%	0	20%	0

- AI Engine Memory power is a function of
  - Number of memory banks
    - Memory banks refers to physical memory banks
    - Each tile has 8 data memory banks
    - XPE defaults to using all available memory banks
  - Average “R/W rate”
    - This is the portion of time when memory bank is accessed with read or write
    - 100% means every bank is accessed every cycle
    - If a core reads from a bank every cycle, average R/W Rate is 12.5%
    - Typical Memory R/W Rate is 10%-30%
    - For early estimation, the recommendation is to use the default of 20%

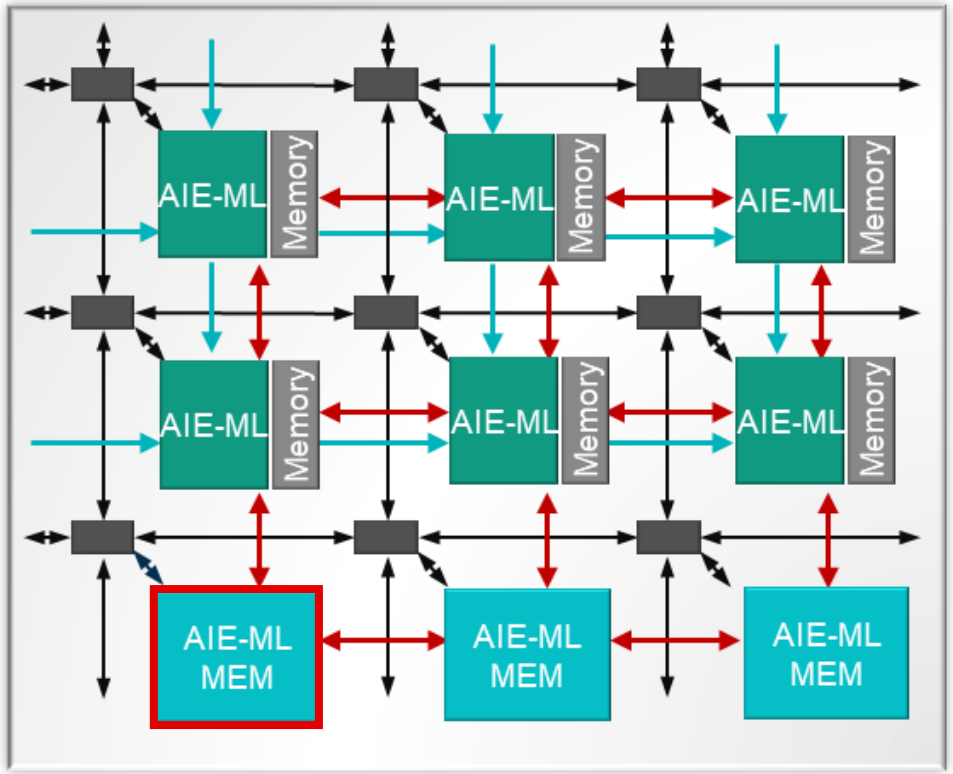


AMD XILINX

AI Engine Power: Memory Tiles

- For AIE-ML devices
- AI Engine Memory power is a function of
  - Number of memory banks
    - Memory banks refers to physical memory banks
    - Each memory tile has 16 data memory banks
  - Average “R/W rate”
    - This is the portion of time when memory bank is accessed with read or write
    - 100% means every bank is accessed every cycle
    - For early estimation, the recommendation is to use the default of 20%

pute Tile			Shared Memory Tile			
	Memory Banks	Memory R/W Rate	Memory Tiles	Memory Tile Memory Banks	Memory R/W Rate	AI Engin Comput Tiles
%	8	20%	4	8	20%	
%	0	20%	0		20%	
%	0	20%	0		20%	
%	0	20%	0		20%	
%	0	20%	0		20%	

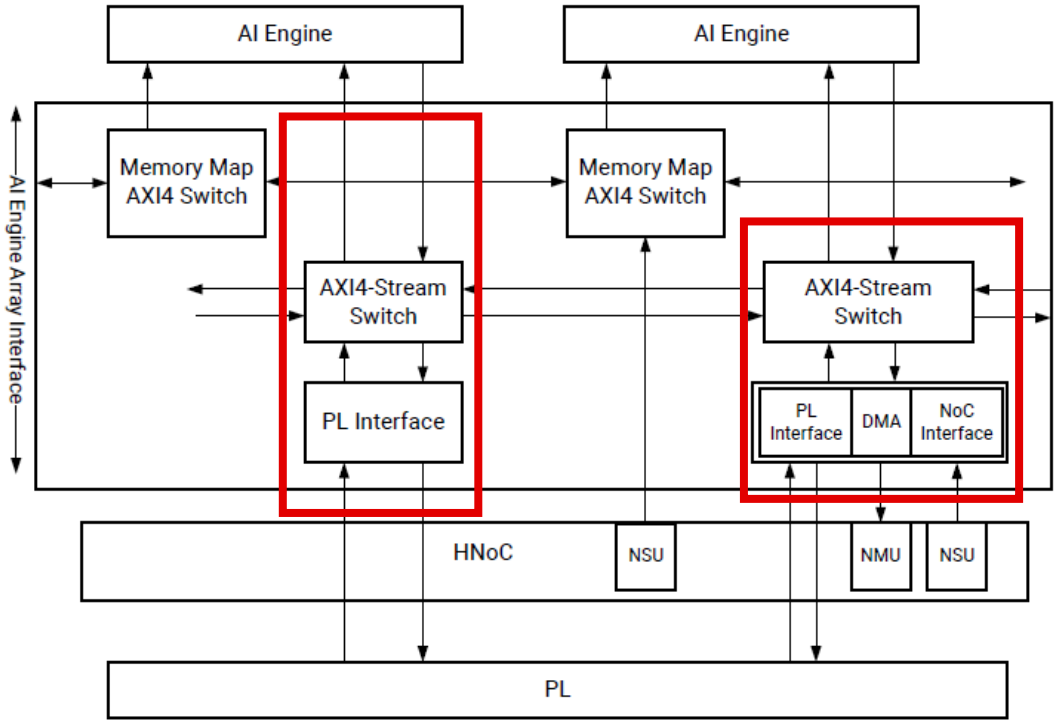


AMD XILINX

PL and Programmable NoC Interfaces

- Interface power is a function of number of interfaces to programmable logic (PL) or network on chip (NoC)
  - A PL interface is one AXI Stream
    - Power Model assumes 64b
    - Power error due to 32b or 128b is small
    - XPE defaults to 1 interface (14 Streams) per 20 Cores
  - A NoC interface is one NoC AXI interface
    - Power model does not distinguish between slave or master interface
    - Also does not distinguish between memory mapped or stream interface
  - Activity on the interface is assumes to be random data
    - Power error due to lower or higher activity is within overall margin

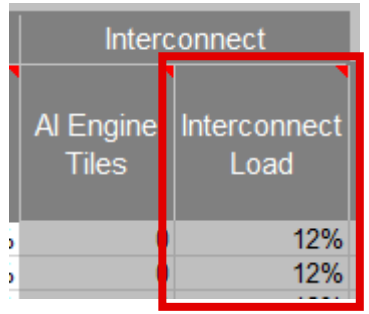
Interface Array Power		
Type	Number of Streams	Power
PL	0	0.000W
NoC	0	0.000W

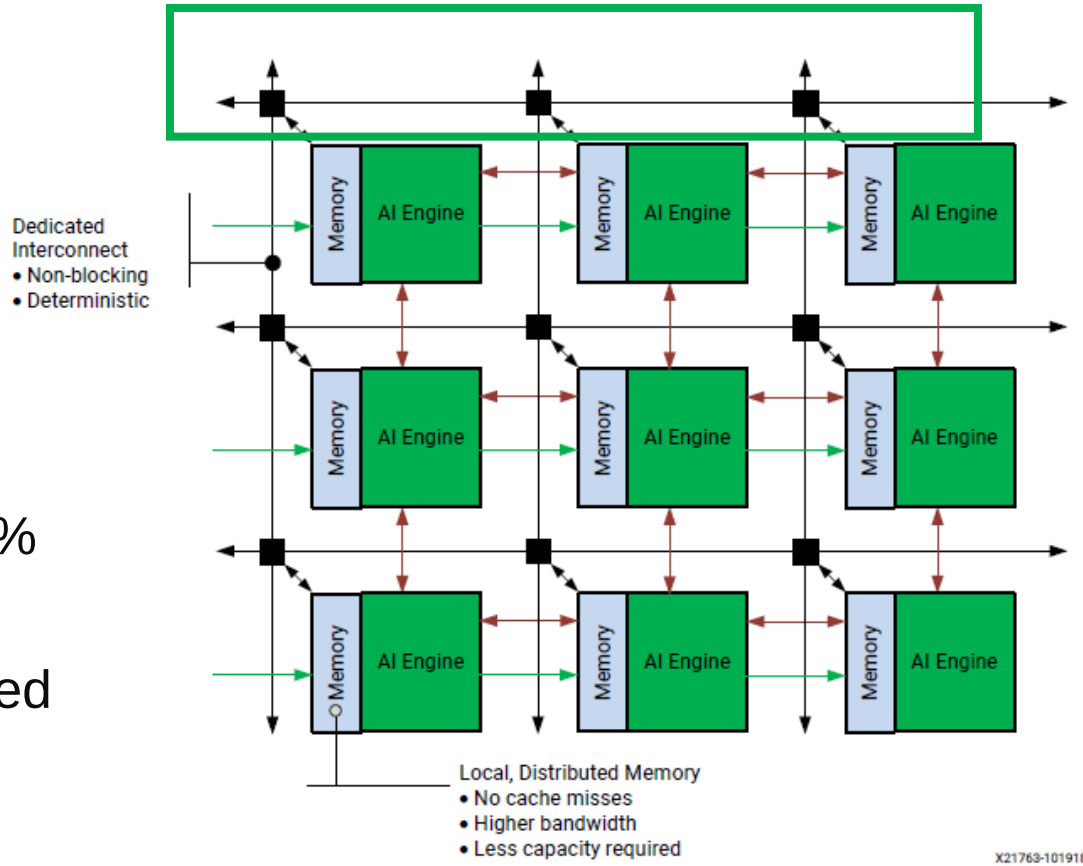


XILINX

Interconnect Power

- Interconnect Power captures power consumes in AXI Stream within the array
  - Total interconnect power is a function of Interconnect Load
    - This is the percentage of ports used by the application
  - Activity on streams is assumes to be random data
  - Typical application has Interconnect Load <20%
  - Value is generated by XPE and cannot be edited
    - Import flow (from AIE tools data) will use exact values





Top Level Power

- Top level power consists of components that are always on, including
  - PLL power
  - Shim clock power
    - Shim does not have coarse-grain clock gating, so regardless of application, it consumes some power
    - Shim clock power varies with device size (number of columns)

Top-level power in VC1902

0.448 – 0.089 = 0.359W (For a full AIE array <1mW per core)

Power		
V_{CCINT}	0.700V	0.419W
V_{CCAUX}	1.500V	0.029W
Total		0.448W

Utilization		
Core	1	0%
PL Stream	0	0%
NoC Stream	0	0%

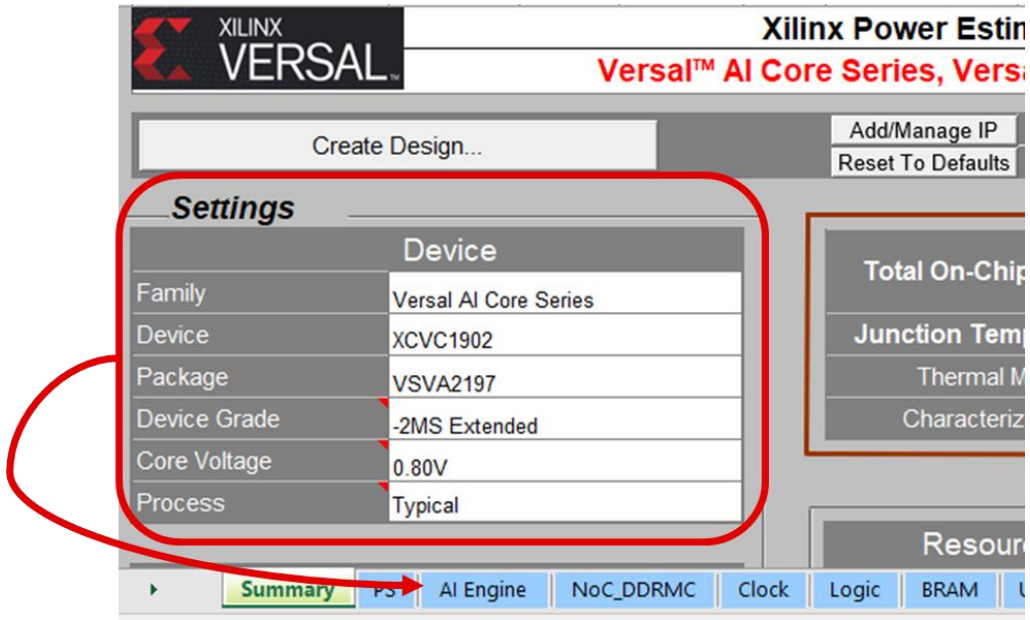
Dynamic Power	
AIE Core	0.068W
Memory	0.380W
Frequency (MHz)	1000.00

Interface Array Power		
Type	Number of Streams	Power
PL	0	0.000W
NoC	0	0.000W

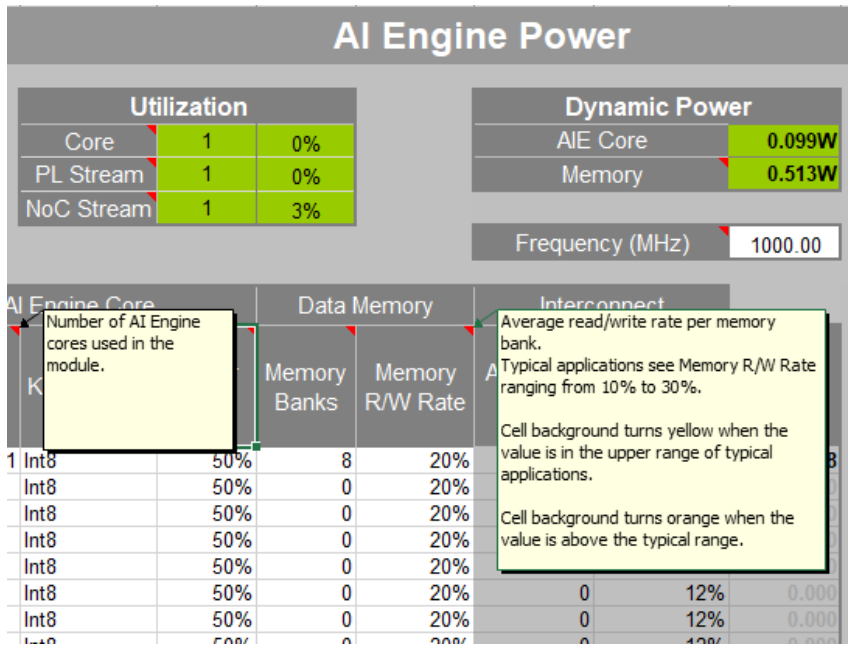
	AI Engine Core			Data Memory		Interconnect		
Module Name	Cores	Kernel Type	Vector Load	Memory Banks	Memory R/W Rate	AI Engine Tiles	Interconnect Load	V_{CCINT} (W)
	1	Int8	50%	8	20%	1	12%	0.089
		Int8	50%	0	20%	0	12%	0.000

AI Engine Power Analysis in XPE

To enable the AI Engine Tab, select a device which includes an AI Engine array (AIE or AIE-ML)



XPE includes tooltip to help users enter their design's information

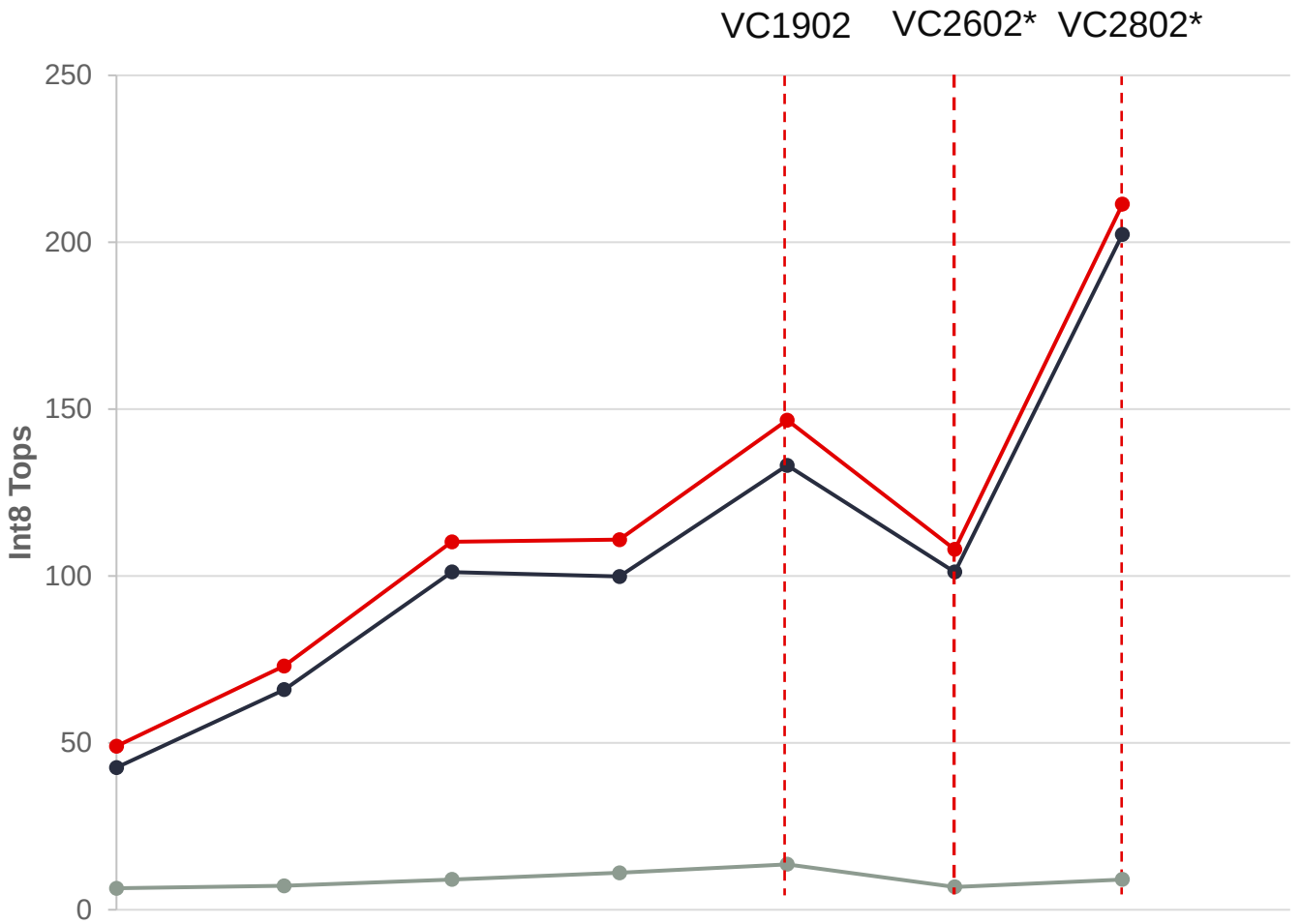


3 Different Flows

- Manual Flow
  - Users enter manually all the information in the AI Engine tab
- Complier Based Flow
  - Users import the .xpe file generated by aiecompiler in <*build directory*>/Work/reports
- Simulation Based Flow
  - Users need to run simulation with VCD file output enabled
  - VCD file can be converted to .xpe file using vcdanalyze command
    - `vcdanalyze --vcd <vcdfile> --xpe`
  - .xpe file can then be imported into XPE

Competitive

Versal® AI Core Series Peak Compute Capacity (int8)



*VC2602 and VC2802 have an AIE-ML array. Other devices of the series have an AIE array

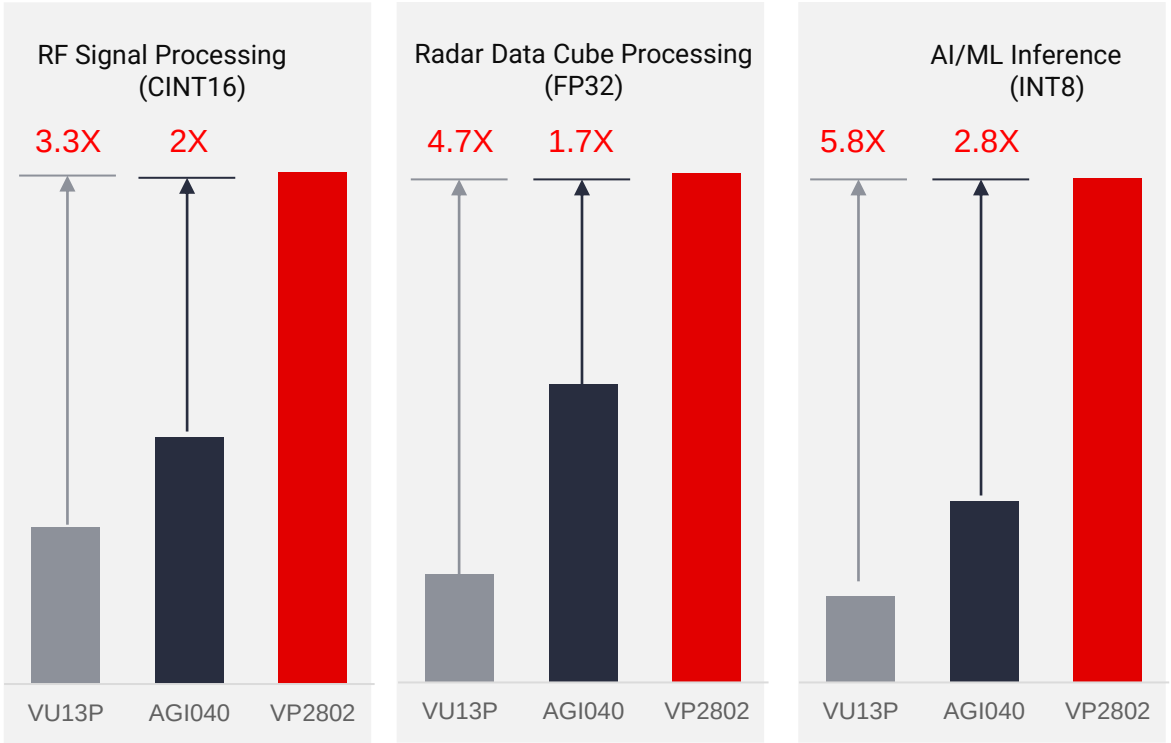
Notes:

1. AI Engine @ 1.3 GHz
   128 int8 MACs per AIE
   256 int8 MACs per AIE-ML
2. DSP Engine @ 1.15 GHz
   3 int8 MACs per DSP

https://www.xilinx.com/support/documentation/selection-guides/versal-ai-core-product-selection-guide.pdf

Versal® Premium Series vs. Agilex Compute Comparison

- **Peak compute advantage (DSP58 + AI Engine)**
  - INT8: 2.8X advantage
  - CINT16: 2X advantage
  - FP32: 1.7X advantage
- **Agilex increases DSP 50% per LE over Stratix 10**
  - No direct response to AI Engine
  - Addressing compute with more of same/similar
  - 9x9 multipliers return to action (not in Stratix 10)
- **Clearly optimizing for DC / ML with new:**
  - Int8 – ML inference optimization
  - FP16 / BFloat16 – ML training and inference EoU
- **A&D & Wireless markets also targeted:**
  - Retaining features like Complex MAC
  - Fmax reductions vs. Stratix 10 are evident



Remember: TOPs are only a first order metric for comparing relative capability. Application-level performance is what really matters and future benchmarks will be provided in these areas.

1: AMD-Xilinx results based on calculated compute capacity with DSP slices in Virtex UltraScale+ VU13P FPGA
2: Intel results based on calculated compute capacity with DSP blocks in Intel Agilex AGI040 FPGA and frequency using the fastest speed grade from Intel Agilex Device Data Sheet (DS-1060)
3: AMD-Xilinx results based on calculated compute capacity combined AI Engines and DSP Engines in the Versal Premium VP2802 ACAP

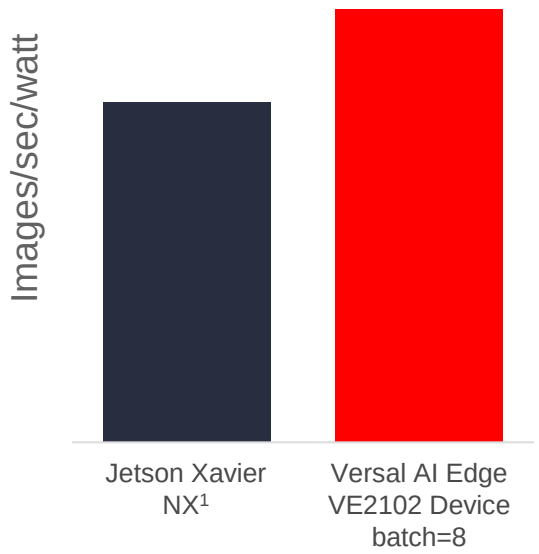
>> 106

AMD XILINX

Versal® AI Edge Series: Up to 4X Performance/Watt vs. GPUs

Intelligent Edge Sensor (10-15 Watts)

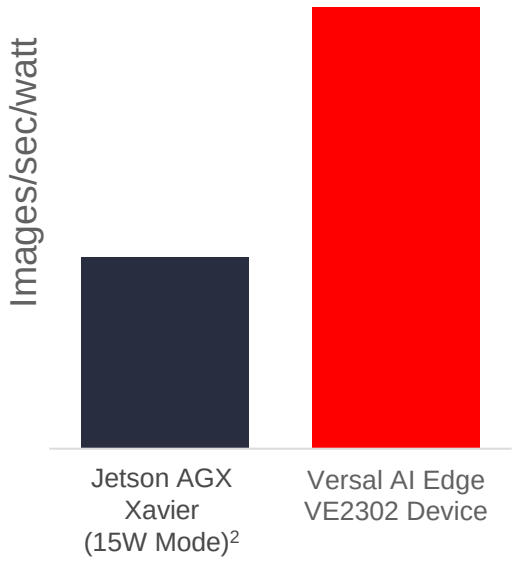
1.3X Performance/Watt



ResNet50 224x224

Autonomous System or Edge Aggregation (7-10 Watts)

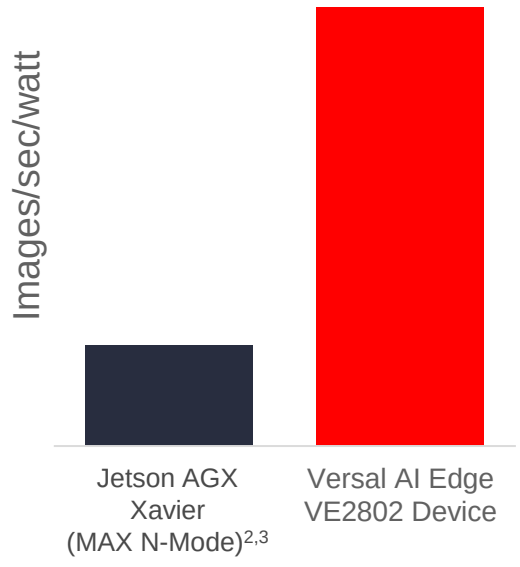
3.3X Performance/Watt



ResNet50 224x224, batch=1

CPU Accelerator (26-45 Watts)

4.2X Performance/Watt



ResNet50 224x224, batch=1

1: Jetson NX Xavier: https://developer.nvidia.com/embedded/jetson-benchmarks, batch size and number not published
2: Jetson AGX Xavier: https://developer.nvidia.com/embedded/jetson-agx-xavier-dl-inference-benchmarks,
3: Jetson AGX Xavier MAX N-Mode and VE2802 represent the highest performing device configuration in their respective portfolios
Jetson AGX Xavier device power estimated by subtracting published memory & IO power from total module power

AMD XILINX

Summary

AI Engine Architecture Summary

Compute Efficiency	8x increase in compute density At ~40% lower power
Heterogeneous Architecture	High-throughput, low-latency PL flexibility – accelerate the complete application Custom memory hierarchy
Multiple Applications	Data center: AI inference Comms: wireless 5G – radio (DFE), baseband, Wired – cable access ADAS/AD: embedded vision
Software Programmable	Frameworks; C/C++ SW compile, execute, debug Optimized S/W libraries

Thank You

Disclaimer and Attribution