Versal Adaptive SoC AIE-ML

Architecture Manual

AM020 (v1.4) June 16, 2025

AMD

Table of Contents

Overview

Introduction to Versal Adaptive SoCs

AMD Versal™ adaptive SoCs combine programmable logic (PL), processing system (PS), and AI Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. The hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. A host of tools, software, libraries, IP, middleware, and frameworks enable Versal adaptive SoCs to support all industry-standard design flows.

The Versal portfolio is the first platform to combine software programmability and domain-specific hardware acceleration with the adaptability necessary to meet today's rapid pace of innovation. The portfolio is uniquely architected to deliver scalability and AI inference capabilities for a host of applications across different markets—from cloud—to networking—to wireless communications—to edge computing and endpoints.

The Versal architecture has a wealth of connectivity and communication capability and a programmable network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. AI Engines are SIMD VLIW vector processors for adaptive inference and advanced signal processing compute. The PL combines configurable logic blocks, memory, and DSP Engines architected for high-compute density. The PS includes application and real-time processors from Arm® for intensive compute tasks.

The Versal AI Edge Series Gen 2 delivers end-to-end acceleration for AI-driven embedded systems—all in a single device built on a foundation of enhanced safety and security. Combining world-class programmable logic with a new high-performance processing system of integrated Arm CPUs and next-generation AI Engines, these devices enable all three phases of compute in embedded AI applications: preprocessing, AI inference, and postprocessing.

The Versal AI Edge Series focuses on AI performance per watt for real-time systems in automated drive, predictive factory and healthcare systems, multi-mission payloads in aerospace & defense, and a breadth of other applications. More than just AI, the Versal AI Edge Series accelerates the whole application from sensor to AI to real-time control, all with the highest levels of safety and security to meet critical standards such as ISO 26262 and IEC 61508.

The Versal AI Core Series delivers breakthrough AI inference acceleration and compute performance. This series is designed for a breadth of applications, including cloud for dynamic workloads and network for massive bandwidth, all while delivering advanced safety and security features. AI and data scientists, as well as software and hardware developers, can all take advantage of the high-compute density to accelerate the performance of any application.

The Versal Prime Series Gen 2 combines world-class programmable logic from AMD with a new high-performance processing system of integrated Arm CPUs—offering significantly greater scalar compute than existing Versal or Zynq™ adaptive SoCs. This powerful combination of flexible, real-time sensor processing and the ability to handle complex embedded computing workloads allows designers to maximize system performance while avoiding the overhead of a multi-chip solution.

The Versal Prime Series is the foundation and the mid-range of the Versal portfolio, serving the broadest range of uses across multiple markets. These applications include 100G to 200G networking equipment, network and storage acceleration in the data center, communications test equipment, broadcast, and aerospace & defense. The series integrates mainstream 58G transceivers and optimized I/O and DDR connectivity, achieving low-latency acceleration and performance across diverse workloads.

The Versal Premium Series Gen 2 offers new levels of memory and data bandwidth with CXL® 3.1, PCIe® Gen6, and DDR5/LPDDR5X interfacing capabilities, tailored to fit the application requirements of tomorrow's data center, communications, test & measurement, and aerospace & defense data-intensive applications. As a heterogeneous compute platform, the series is engineered to help users reach high levels of acceleration for a wide range of compute-intensive workloads by providing high compute density, custom memory hierarchy, and DSP Engine resources.

The Versal Premium Series provides breakthrough heterogeneous integration, very high-performance compute, connectivity, and security in an adaptable platform with a minimized power and area footprint. The series is designed to exceed the demands of high-bandwidth, compute-intensive applications in communications, data center, test & measurement, and other applications. The Versal Premium Series includes 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

The Versal HBM Series enables the convergence of fast memory, adaptable compute, and secure connectivity in a single platform. The series is architected to keep up with the higher memory needs of the most compute intensive, memory-bound applications, providing adaptable acceleration for data center, communications, test & measurement, and aerospace & defense applications. Versal HBM adaptive SoCs integrate the most advanced HBM2e DRAM, providing high memory bandwidth and capacity within a single device.

The Versal architecture documentation suite is available at: https://www.amd.com/versal.

Navigating Content by Design Process

AMD Adaptive Computing documentation is organized around a set of standard design processes to help you find relevant content for your current development task. You can access the AMD Versal™ adaptive SoC design processes on the Design Hubs page. You can also use the Design Flow Assistant to better understand the design flows and find content that is specific to your intended design needs. This document covers the following design processes:

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:
  - Chapter 1: Overview provides an overview of the AIE-ML architecture and includes:
    - AIE-ML Array Overview
    - AIE-ML Array Hierarchy
    - Performance
  - Chapter 2: AIE-ML Tile Architecture describes the interaction between the memory module and the interconnect and between the AIE-ML and the memory module.
  - Chapter 3: AIE-ML Array Interface Architecture is a high-level view of the AIE-ML array interface to the PL and NoC.
  - Chapter 4: AIE-ML Architecture describes the processor functional unit and register files.
  - Chapter 5: AIE-ML Memory Tile Architecture describes the features and functionality of the AIE-ML memory tile, which is an additional functional unit in the AIE-ML architecture.
  - Chapter 6: AIE-ML Configuration and Boot describes configuring the AIE-ML v2 array from the processing system during boot and reconfiguration.
- **AI Engine Development:** Creating the AI Engine graph and kernels, library use, simulation debugging and profiling, and algorithm development. Also includes the integration of the PL and AI Engine kernels. Topics in this document that apply to this design process include:
  - Chapter 2: AIE-ML Tile Architecture
  - Chapter 4: AIE-ML Architecture
  - Chapter 5: AIE-ML Memory Tile Architecture

Motivation to AIE-ML

The non-linear increase in demand in machine learning and other compute intensive applications leads to the development of the AMD Versal™ adaptive SoC AI Engine-Machine Learning (AIE-ML). The AIE-ML, the dual-core Arm® Cortex®-A72 and Cortex-R5F processor (PS), and the next generation programmable logic (PL) are all tied together with a high-bandwidth NoC to form a new architecture in adaptive SoC. The AIE-ML and PL are intended to complement each other to handle functions that match their strengths. With the custom memory hierarchy, multi-cast stream capability on AI interconnect and AI-optimized vector instructions support, the Versal adaptive SoC AIE-MLs are optimized for various compute-intensive applications, for example machine learning inference acceleration in data center applications by enabling deterministic latency and low neural network latency with high performance per watt.

AIE-ML Array Features

AMD developed multiple iterations of AI Engines. This architecture manual details the specifics of the AIE-ML.

Some Versal adaptive SoCs include the AIE-ML that consists of an array of AIE-ML tiles, AIE-ML memory tiles, and the AIE-ML array interface consisting of the network on chip (NoC) and programmable logic (PL) tiles. The following lists the features of each. A pictorial view of the array organization is shown in Figure 1: Versal Device (with AIE-ML) Top-Level Block Diagram.

AIE-ML Tile Features

- A separate building block, integrated into the silicon, outside the programmable logic (PL).
- One AIE-ML incorporates a high-performance very-long instruction word (VLIW) single-instruction multiple-data (SIMD) vector processor optimized for many applications including machine learning applications.
- From a hardware perspective, data memory is 64 KB organized as eight banks of 8 KB. From a programmer's perspective, every two banks are interleaved to form one bank, that is, a total of four banks of 16 KB each.
- Streaming interconnect for deterministic throughput, high-speed data flow between AIE-ML tiles and/or the programmable logic in the Versal device.
- Direct memory access (DMA) in the AIE-ML tile moves data from incoming stream(s) to local memory and from local memory to outgoing stream(s).
- Configuration interconnect (through memory-mapped AXI4 interface) with a shared, transaction-based switched interconnect for access from external masters to internal AIE-ML tile.

- Hardware synchronization primitives provide synchronization of the AIE-ML, between the AIE-ML and the tile DMA, and between the AIE-ML and an external master (through the memory-mapped AXI4 interface).
- Debug, trace, and profile functionality.
- The AIE-ML tile has additional granularity on clock gating and reset. Clock gating and reset of the AIE-ML tile can be done via the memory-mapped AXI4 register inside the tile. In AIE-ML the memory-mapped AXI4, clock gating, and reset registers are moved into an always-on domain to give modular control to core, stream-switch and memory module in the tile. A similar arrangement also applies to the memory tile, a functional unit in the AIE-ML that is introduced in the following section.

AIE-ML Memory Tile Features

- A tile containing 512 KB of high-density, high-bandwidth memory to reduce the use of PL resources in machine learning (ML) applications.
- The memory tile DMA has the same channel features as the AIE-ML tile with the exception that the memory tile DMA also supports 4-D addressing modes. See Chapter 5: AIE-ML Memory Tile Architecture for a more detailed description.
- AXI4-Stream interconnect is the same as AIE-ML tile except the number of ports and connectivity are different.
- Memory-mapped AXI4 configuration is the same as the AIE-ML tile.

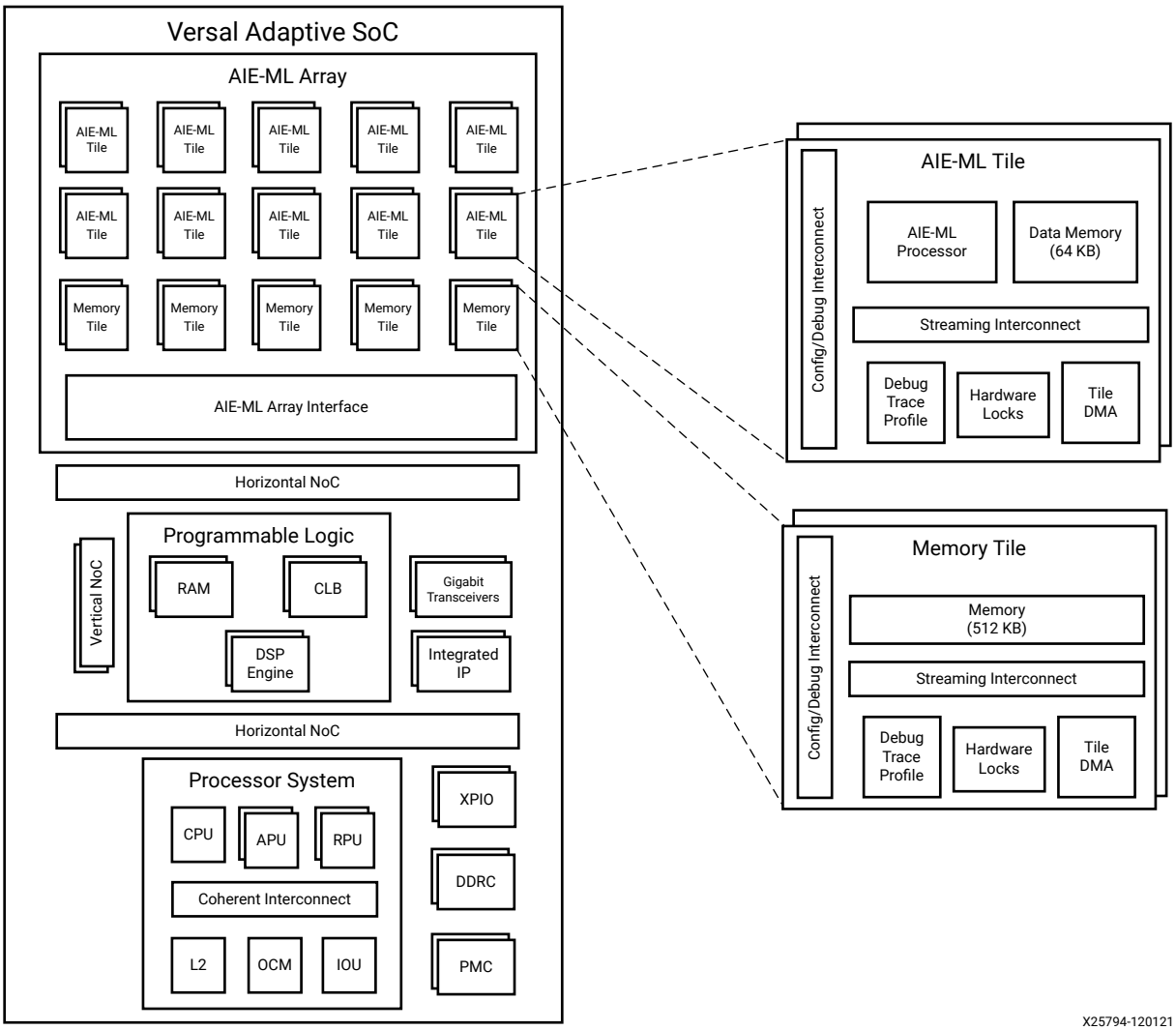
AIE-ML Array Interface to NoC and PL Resources

- Direct memory access (DMA) in the AIE-ML NoC interface tile manages incoming and outgoing memory-mapped and streams traffic into and out of the AIE-ML array. The interface tile is described in Chapter 3: AIE-ML Array Interface Architecture.
- Configuration and control interconnect functionality (through the memory-mapped AXI4 interface)
- Streaming interconnect that leverages the AIE-ML tile streaming interconnect functionality.
- AIE-ML to programmable logic (PL) interface that provides asynchronous clock-domain crossing between the AIE-ML clock and the PL clock.
- AIE-ML to NoC interface logic to the NoC master unit (NMU) and NoC slave unit (NSU) components.
- Hardware synchronization primitives leverage features from the AIE-ML tile locks module.
- Debug, trace, and profile functionality that leverage all the features from the AIE-ML tile.
- For a list of changes from the AI Engine (AIE) array interface, see Chapter 3: AIE-ML Array Interface Architecture.

AIE-ML Array Overview

The following figure shows the high-level block diagram of a Versal adaptive compute acceleration platforms (adaptive SoCs) with an AIE-ML array in it. The device consists of the processor system (PS), programmable logic (PL), and the AIE-ML array.

Figure 1: **Versal Device (with AIE-ML) Top-Level Block Diagram**



The AIE-ML array is the top-level hierarchy of the AIE-ML architecture. It integrates a two-dimensional array of AIE-ML tiles. Each AIE-ML tile integrates a very-long instruction word (VLIW) processor, integrated memory, and interconnects for streaming, configuration, and debug. The AIE-ML array introduced a separate functional block, the memory tile, that is used to significantly reduce PL resources (LUTs and UltraRAMs) for ML applications. The memory tile

has 512 KB data memory, 12 DMA channels (eight can access neighboring memory tiles) and stream interfaces. Depending on the device there can be one or two rows of memory tiles. The AIE-ML array interface enables the AIE-ML array to communicate with the rest of the Versal device through the NoC or directly to the PL. The AIE-ML array also interfaces to the processing system (PS) and platform management controller (PMC) through the NoC.

AMD Versal™ adaptive SoC devices that integrate AIE-ML tiles have access to the following types of memory:

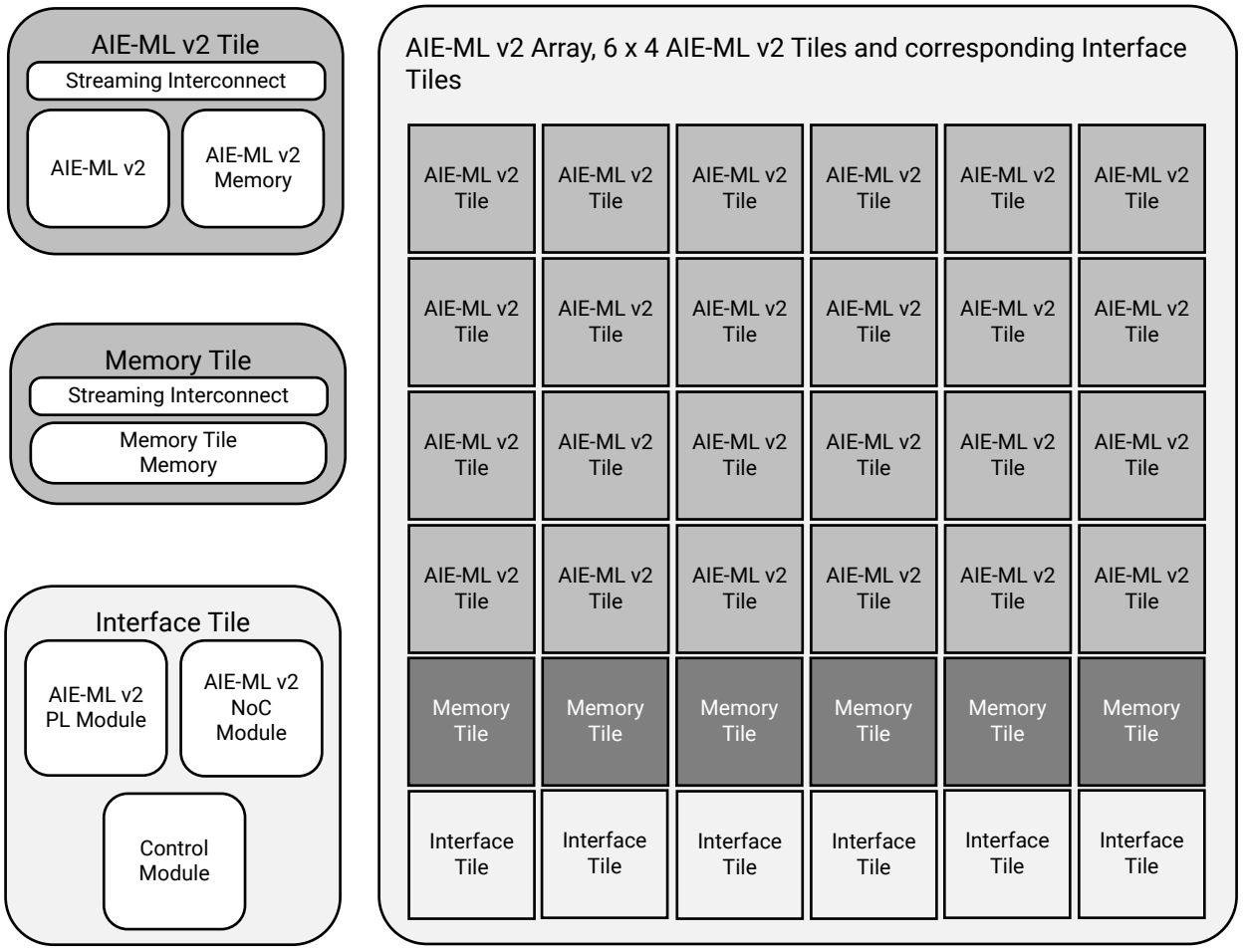
- External DDR memory (via NoC)
- On-chip PL memory resources (UltraRAM/block RAM)
- On-chip shared memory in AIE-ML memory tiles
- On-chip local data memory in AIE-ML tiles

Depending on the use case, the data and weights move through the memory hierarchy in different ways.

AIE-ML Array Hierarchy

The AIE-ML array is made up of AIE-ML tiles, one or two rows of AIE-ML memory tiles, and AIE-ML array interface-tiles (the last row of the array). The following figure shows a conceptual view of the complete tile hierarchy associated with the AIE-ML array. See Chapter 2: AIE-ML Tile Architecture, Chapter 3: AIE-ML Array Interface Architecture, and Chapter 4: AIE-ML Architecture for detailed descriptions of the various tiles.

Figure 2: **Hierarchy of Tiles in an AIE-ML Array**



Performance

The AIE-ML array has a single clock domain for all the tiles and array interface blocks. The performance target of the AIE-ML array for the -1L speed grade devices is 1 GHz with V_{CCINT} at 0.70V. In addition, the AIE-ML array has clocks for interfacing to other blocks. The following table summarizes the various clocks in the AIE-ML array and their performance targets. For more information, see *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957) or *Versal AI Edge Series Data Sheet: DC and AC Switching Characteristics* (DS958).

Table 1: **AIE-ML Interfacing Clock Domains**

Clock	Target for -1L	Source	Relation to AIE-ML Clock
AIE-ML array clock	1 GHz	AIE-ML PLL	N/A

Table 1: **AIE-ML Interfacing Clock Domains** *(cont'd)*

Clock	Target for -1L	Source	Relation to AIE-ML Clock
NoC clock	960 MHz	NoC clocking	Asynchronous, clock domain crossing (CDC) within the NoC
PL clocks	500 MHz	PL clocking	Asynchronous, CDC within AIE-ML array interface
NPI clock	300 MHz	NPI clocking	Asynchronous

Clocking Structure

Each AIE-ML interface tile has a column clock gate control register that controls the clock to all the memory tiles and AIE-ML tiles in the same column. The register does not affect the clock of the AIE-ML interface tile. When all the memory tiles and AIE-ML tiles in a column are unused, disabling its clock through this control register reduces the power consumption of the AIE-ML array.

Memory Error Handling

Each AIE-ML has 64 KB of data memory and 16 KB of program memory. Due to the large amount of memory in the AIE-ML tiles, protection is provided against soft errors. The 128-bit word in the program memory is protected with two 8-bit ECC (one for each 64-bit). The 8-bit ECC can detect 2-bit errors and detect/correct a 1-bit error within the 64-bit word. The two 64-bit data and two 8-bit ECC fields are each interleaved within its own pair (distance of two) to create larger bit separation.

There are eight memory banks in each data memory module. The first two memory banks have 7-bit ECC protection for each of the four 32-bit fields. The 7-bit ECC can detect 2-bit errors and detect/correct a 1-bit error. The last six memory banks have even parity bit protection for each 32 bits in a 128-bit word. The four 32-bit fields are interleaved with a distance of four.

Error injection is supported for both program and data memory. Errors can be introduced into program memory over memory-mapped AXI4. Similarly, errors can be injected into data memory banks over AIE-ML DMA or memory-mapped AXI4.

When the memory-mapped AXI4 access reads or writes to AIE-ML data memory, two requests are sent to the memory module. On an ECC/parity event, the event might be counted twice in the AIE-ML performance counter. There is duplicate memory access but no impact on functionality. Refer to Chapter 2: AIE-ML Tile Architecture for more information on events and performance counters.

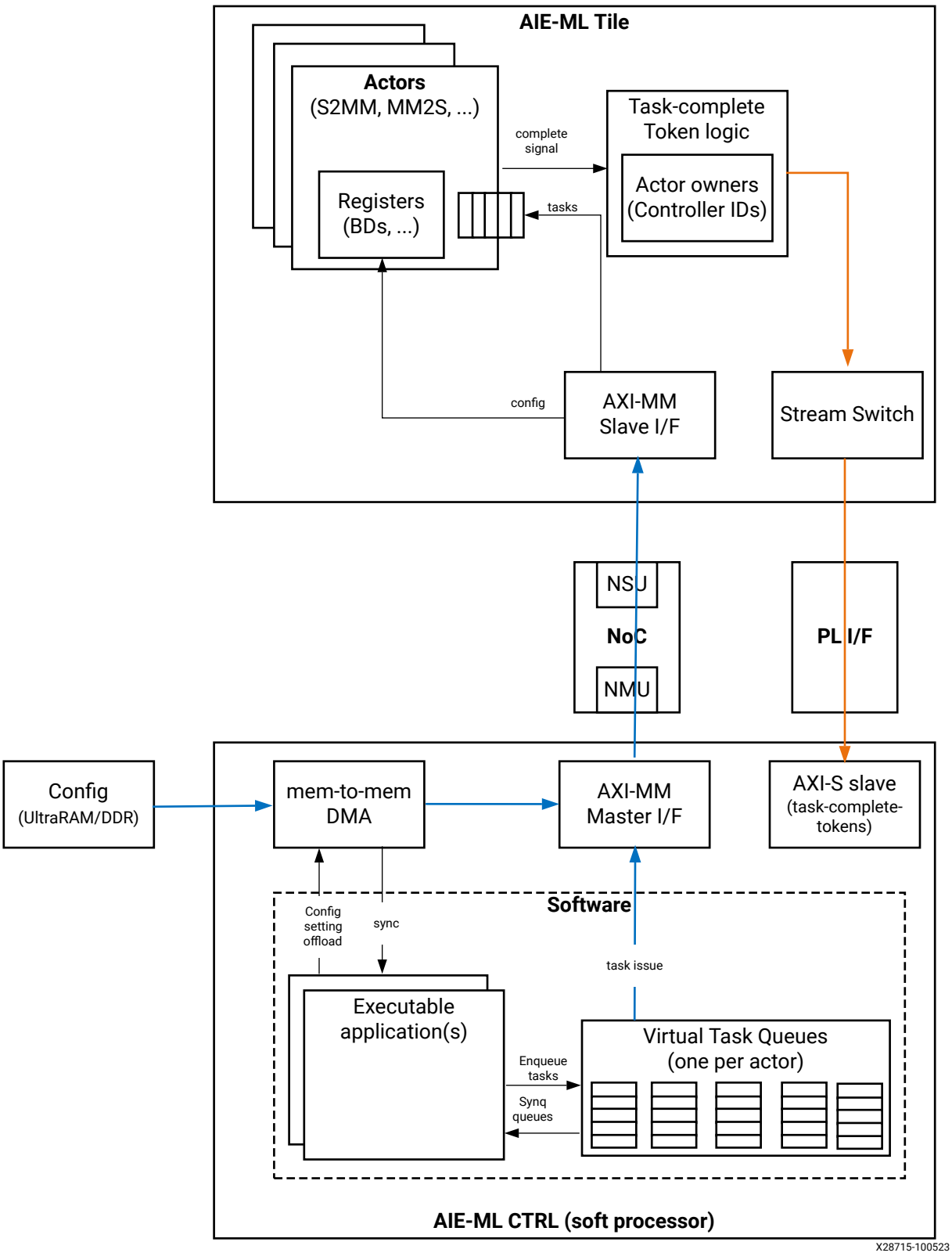
Internal memory errors (correctable and uncorrectable) create internal events that use the normal debug, trace, and profiling mechanism to report error conditions. They can also be used to raise an interrupt to the PMC/PS.

Task-Completion-Tokens

In the context of task-completion tokens, a task refers to a unit of work that is pushed into the input task queue of an actor (such as an AIE-ML tile, AIE-ML memory, and AIE-ML array interface) through the memory mapped interface. The AIE-ML tile, AIE-ML memory, and AIE-ML array interfaces are referred to as actors, each equipped with an input task queue. A task can be placed in the queue through the AXI4 interface, managed by a controller. The actor processes the task until it is completed before proceeding to the next one. Once all tasks in the queue are finished, the actor enters an idle state, awaiting the insertion of a new task.

Upon the completion of a task, the actor triggers the tile task completion unit to send a task completion token to the stream switch through the control-packet response port. In AIE-ML, these task completion tokens are exposed at the AXIS PLIO interface located at the perimeter of the AI Engine array and are intended to be used through PL.

Figure 3: **Task-Complete-Tokens**



The figure shows the controller implemented in soft logic in the PL, however, a valid design methodology is to have the front end of the application run on the PS and only a minimum back-end IP on the PL to service the token streams. The task-complete-token consists of a single stream word. This token is routed back to the issuing controller through the standard stream-switch network. There is an eight-bit controller ID register that determines the ownership of each actor. The lower five bits of the controller ID register are used as the stream packet ID, and the upper three bits are available for further routing in the PL.

Sparsity

Many neural networks generate zero activations between layers due to the use of the `ReLU` function, which simply clamps the negative values to zero. So, these arbitrary zeros are introduced by the `ReLU` function in the input and output activations. These zeroes are not transmitted over the AXI4-Streams because of decompression/compression logic added to the AIE-ML tile and the AIE-ML Mem DMAs. The zero-valued data weights can also be compressed offline – moving from external memory in a compressed state and decompressed by the S2MM channel of the tile DMA.

Moreover, AIE-ML cores support on-the-fly decompression during data loading by inserting zero weights before performing convolutions. This feature ensures that only non-zero weights need to be stored in local tile memories.

Compression of activations in AXI4-Streams and on-the-fly decompression of weights during core loads are optional features. The compression/decompression in AIE-ML memory and AIE-ML tile DMA is controlled by a dedicated bit in the BD. The two primary use cases are shown in the following figure.

Figure 4: **Compression and Decompression in DMAs**

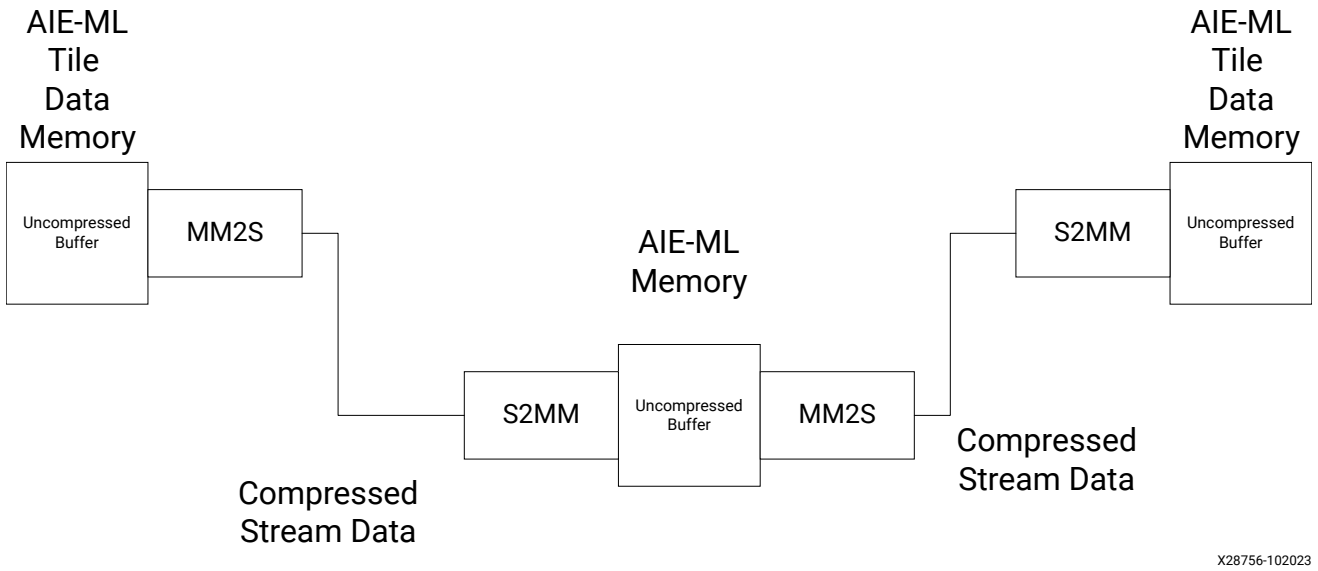
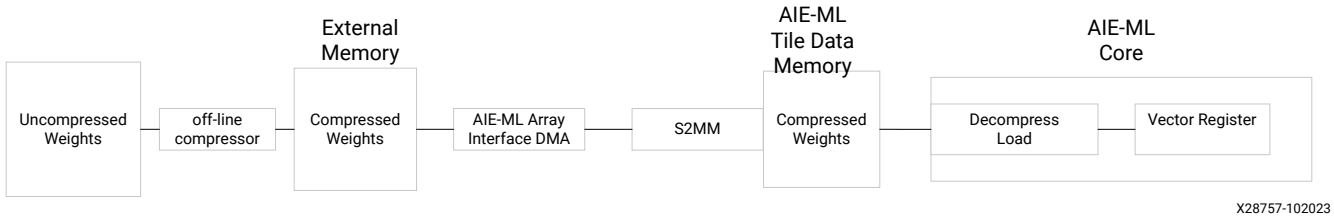
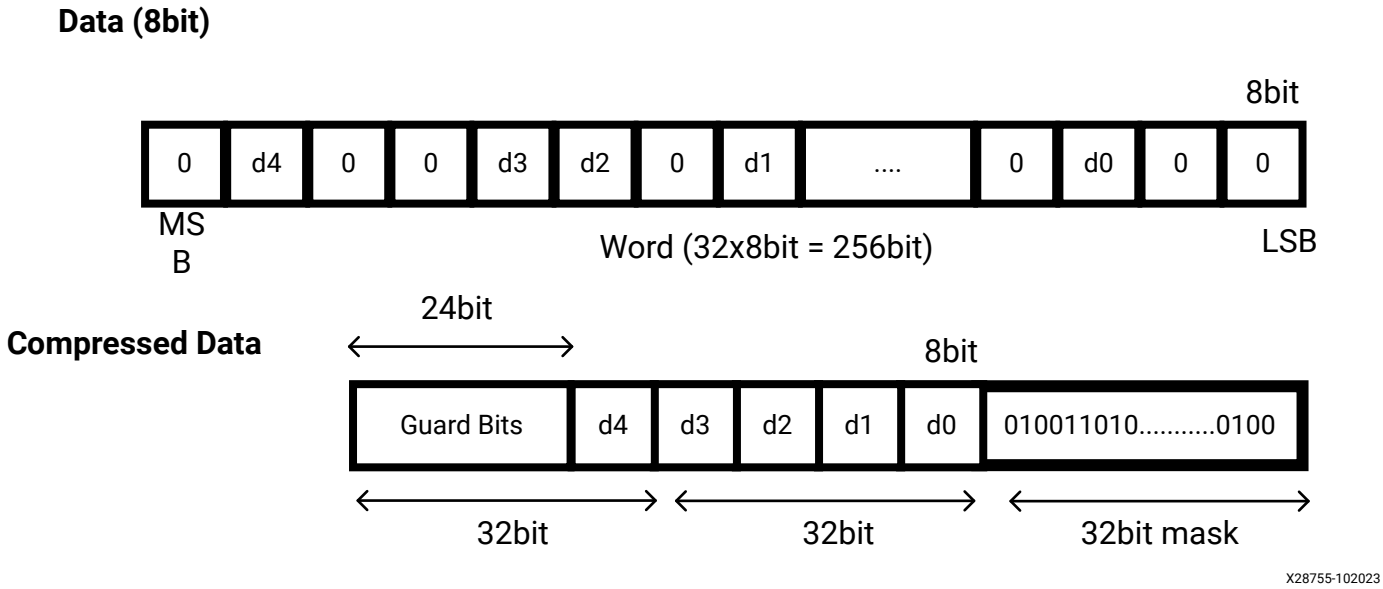


Figure 5: **Offline Compression and Decompression in Core Load Interface**



The following figure illustrates a compression algorithm designed to eliminate zeros in both weights and activations. This algorithm works with 8-bit data samples and employs a 32-bit mask to encode zero and non-zero bytes within a 256-bit word. Zero bytes are represented as 0 in the bit mask, while non-zero bytes are represented as 1. Zero-valued bytes are omitted from the compressed data. Guard bits are inserted as necessary to ensure 32-bit alignment for the subsequent mask. This compression process is consistently applied to all 256-bit data words.

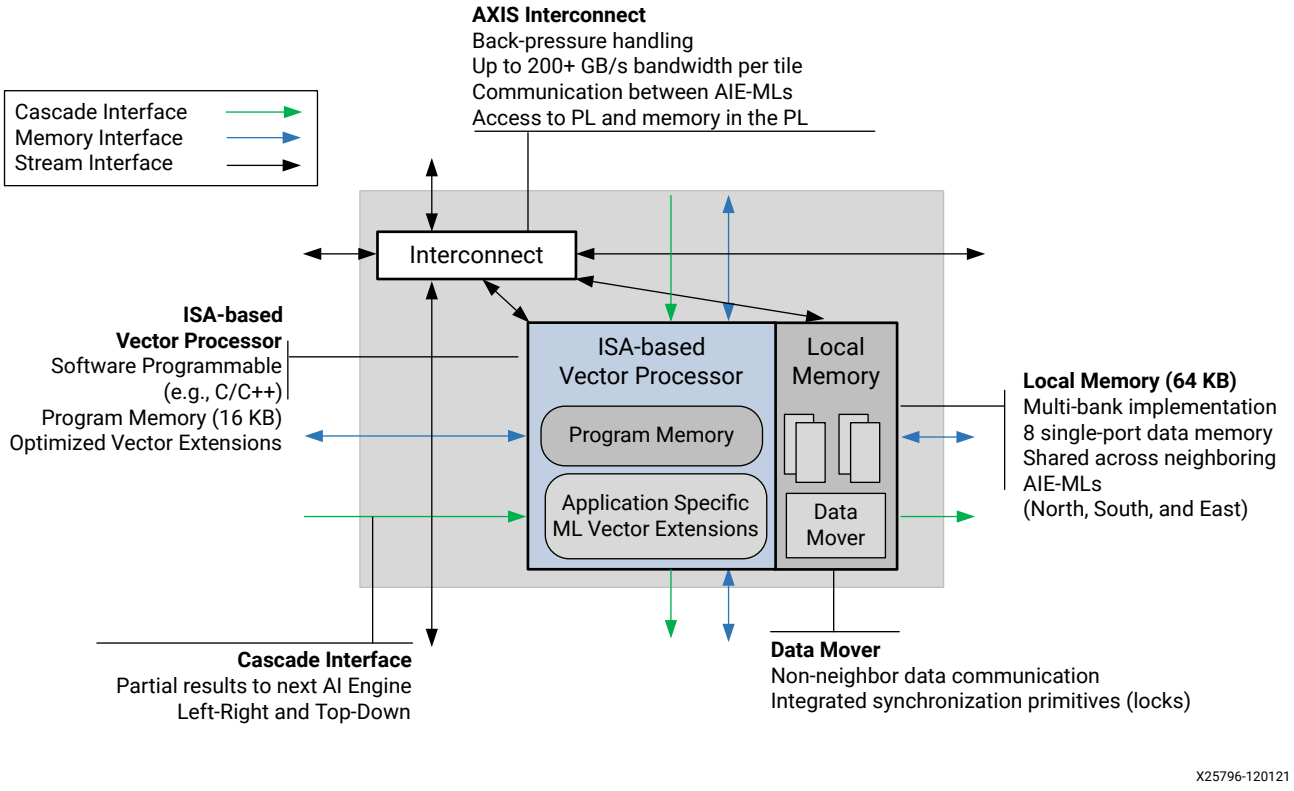
Figure 6: **Compression of 8-bit Samples**



AIE-ML Tile Architecture

The top-level block diagram of the AIE-ML tile architecture, key building blocks, and connectivity for the AIE-ML tile are shown in the following figure.

Figure 7: **AIE-ML Tile Block Diagram**



The AIE-ML tile consists of the following high-level modules:

- Tile interconnect
- AIE-ML
- AIE-ML memory module

The tile interconnect module handles AXI4-Stream and memory mapped AXI4 input/output traffic. The memory-mapped AXI4 and AXI4-Stream interconnect is further described in the following sections. The AIE-ML memory module has 64 KB of data memory divided into eight memory banks, a memory interface, DMA, and locks. There is a DMA in both incoming and outgoing directions and there is a Locks block within each memory module. The AIE-ML can

access memory modules in all four directions as one contiguous block of memory. The memory interface maps memory accesses in the right direction based on the address generated from the AIE-ML. The AIE-ML has a scalar datapath, a vector datapath, three address generators, and 16 KB of program memory. The program and data memory of AIE-ML tile initializes to zero at boot and resets. It also has a cascade stream access for forwarding accumulator output to the next AIE-ML tile. The AIE-ML is described in more detail in Chapter 4: AIE-ML Architecture. Both the AIE-ML and the AIE-ML memory module have control, debug, and trace units. Some of these units are described later in this chapter:

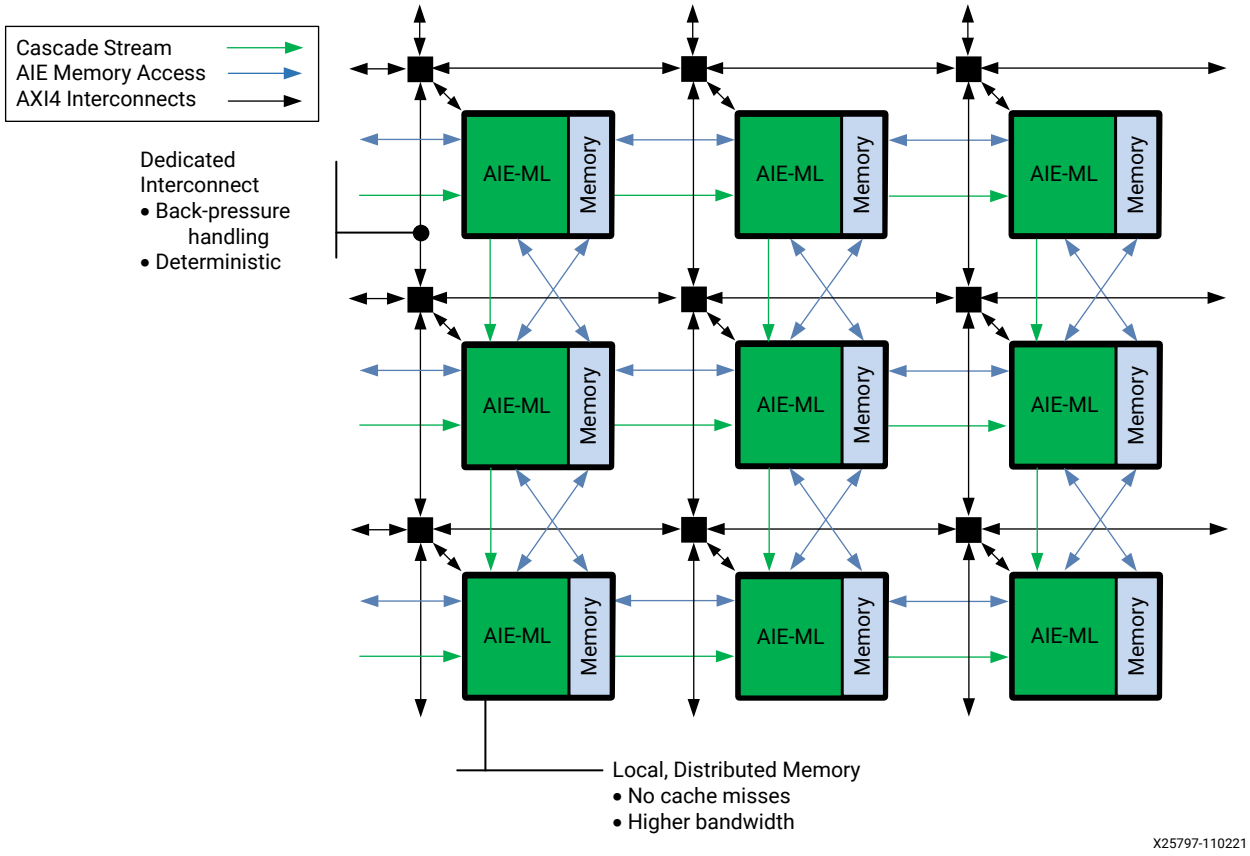
- Control and status registers
- Events, event broadcast, and event actions
- Performance counters for profiling and timers

The following figure shows the AIE-ML array with the AIE-ML tiles and the dedicated interconnect units arrayed together. Sharing data with local memory between neighboring AIE-MLs is the main mechanism for data movement within the AIE-ML array. Each AIE-ML can access up to four memory modules:

- Its own
- The module on the north
- The module on the south
- The module on the west

The AIE-ML on the edges of the array have access to one or two fewer memory modules.

Figure 8: **AIE-ML Array**



Together with the flexible and dedicated interconnects, the AIE-ML array provides deterministic performance, low latency, and high bandwidth. The modular and scalable architecture allows more compute power as more tiles are added to the array.

The AIE-ML has both horizontal and vertical cascade connections, directed from north to south and from west to east. The cascade start points and end points are tied off at the array edges.

Memory-mapped AXI4 Interconnect

Each AIE-ML tile contains a memory-mapped AXI4 interconnect for use by external blocks to write to or read from any of the registers or memories in the AIE-ML tile. The memory-mapped AXI4 interconnect inside the AIE-ML array can be driven from outside of the AIE-ML array by any AXI4 master that can connect to the network on chip (NoC). All internal resources in an AIE-ML tile including memory, and all registers in an AIE-ML and AIE-ML memory module, are mapped onto a memory-mapped AXI4 interface.

Each AIE-ML tile has a memory-mapped AXI4 switch that accepts all memory-mapped AXI4 accesses from the south direction. If the address is for the tile, access occurs. Otherwise, the access is passed to the next tile in the north direction.

The following figure shows the addressing scheme of memory-mapped AXI4 in the AIE-ML tile. The increase in the tile address space is to accommodate the memory tile (512 KB) and also the increase in tile data memory from 32 KB to 64 KB. The lower 20 bits represent the tile address range, followed by five bits that represent the row location and seven bits that represent the column location.

Figure 9: **AIE-ML Memory-Mapped AXI4 Tile Addresses**

Column	Row [22:18]	AIE-ML Tile/Memory Tile/Array Interface Tile
[31:25] (7b)	[24:20] (5b)	[19:0] (20b)

X25888-101921

AXI4-Stream Interconnect

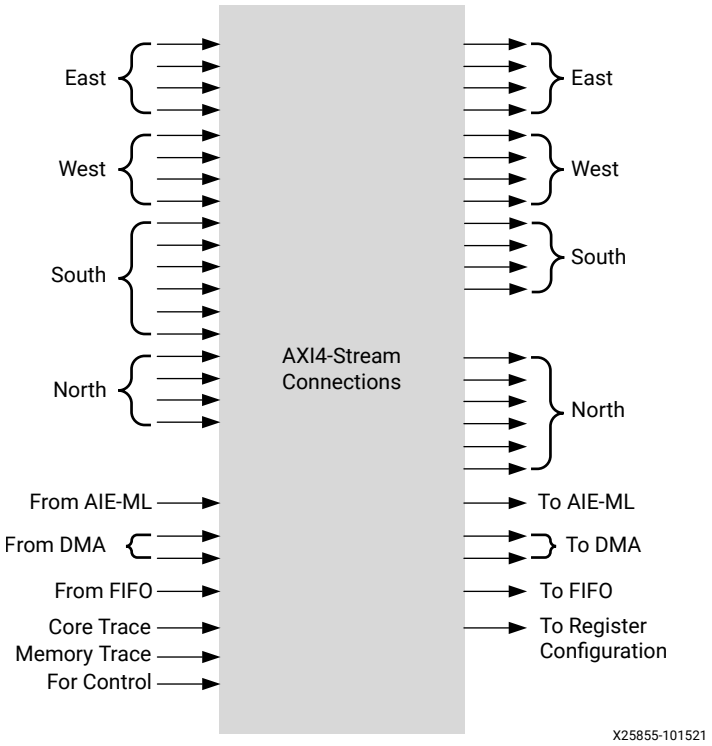
Each AIE-ML tile has an AXI4-Stream interconnect (alternatively called a stream switch) that is a fully programmable, 32-bit, AXI4-Stream crossbar, and is statically configured through the memory-mapped AXI4 interconnect. It handles backpressure and is capable of the full bandwidth on the AXI4-Stream. The following figure is a high-level block diagram of the AXI4-Stream switch. The switch has master ports (data flowing from the switch) and slave ports (data flowing to the switch). The building blocks of the AXI4-Stream interconnect are as follows.

- Port handlers
- FIFOs
- Arbiters
- Stream switch configuration registers

The following lists some of the features of the AXI4-Stream interconnect:

- AIE-ML features 1-to-1 loopback, where only ports with the same ID are connected to each other
- There are 25 slave ports and 23 master ports
- The switch has one FIFO that is 16-deep and 34 bit (32 bit + 1 bit parity + 1 bit TLAST)

Figure 10: **AXI4-Stream Switch High-level Block Diagram**



In AIE-ML, the ports are divided into external and local ports. External ports are South, West, North, and East. The local ports are AIE-ML, DMA, FIFO, and trace. The features of the ports are as follows:

- External ports are 2-cycle latency and a 4-deep FIFO
- Local slave ports are 2-cycle latency and a 4-deep FIFO
- Local master ports have one register slice with 1-cycle latency and a 2-deep FIFO

Therefore, the latency and buffering crossing the switch are (excluding packet switch arbitration overhead):

- Local slave to local master: 3-cycle latency and 6-deep FIFO
- Local slave to external master: 4-cycle latency and 8-deep FIFO
- External slave to local master; 3-cycle latency and 6-deep FIFO
- External to external: 4-cycle latency and 8-deep FIFO

Each stream port can be configured for either circuit-switched or packet-switched streams (never at the same time) using a packet-switching bit in the configuration register. A circuit-switched stream is a one-to-many streams. This means that it has exactly one source port and an arbitrary number of destination ports. All data entering the stream at the source is streamed to all destinations. A packet-switched stream can share ports (and therefore, physical wires) with other logical streams. Because there is a potential for resource contention with other packet-switched streams, they do not provide deterministic latency. The latency for the word transmitted in a circuit-switched stream is deterministic; if the bandwidth is limited, the built-in backpressure causes performance degradation.

A packet-switched stream is identified by a 5-bit ID which has to be unique amongst all streams it shares ports with. The stream ID also identifies the destination of the packet. A destination can be an arbitrary number of master ports and packet-switched streams make it possible to realize all combinations of single/multiple master/slave ports in any given stream.

A packet-switched packet has:

- **Packet header:** Routing and control information for the packet
- **Data:** Actual data in the packet
- **TLAST:** Last word in the packet must have TLAST asserted to mark the end of packet

The packet header is shown here:

Table 2: **Packet Header**

Odd Parity	3'b000	Source Column	Source Row	1'b0	Packet Type	7'b0000000	Stream ID
[31]	[30:28]	[27:21]	[20:16]	[15]	[14:12]	[11:5]	[4:0]

The following table summarizes the AXI4-Stream tile interconnect bandwidth for the -1L speed grade devices.

Table 3: **AIE-ML AXI4-Stream Tile Interconnect Bandwidth**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
To North/From South	6	32	AIE-ML (1 GHz)	4	24
To South/From North	4	32	AIE-ML (1 GHz)	4	16
To West/From East	4	32	AIE-ML (1 GHz)	4	16
To East/From West	4	32	AIE-ML (1 GHz)	4	16

AIE-ML Tile Program Memory

The AIE-ML has a local 16 KB of program memory that can be used to store VLIW instructions. There are two interfaces to the program memory:

- Memory-mapped AXI4 interface
- AIE-ML interface

An external master can read or write to the program memory using the memory-mapped AXI4 interface. The AIE-ML has 128-bit wide interfaces to the program memory to fetch instructions. The AIE-ML can read from, but not write to, the program memory. To access the program memory simultaneously from the memory-mapped AXI4 and AIE-ML, divide the memory into multiple banks and access mutually exclusive parts of the program memory. Arbitration logic is needed to avoid conflicts between accesses and to assign priority when accesses are to the same bank.

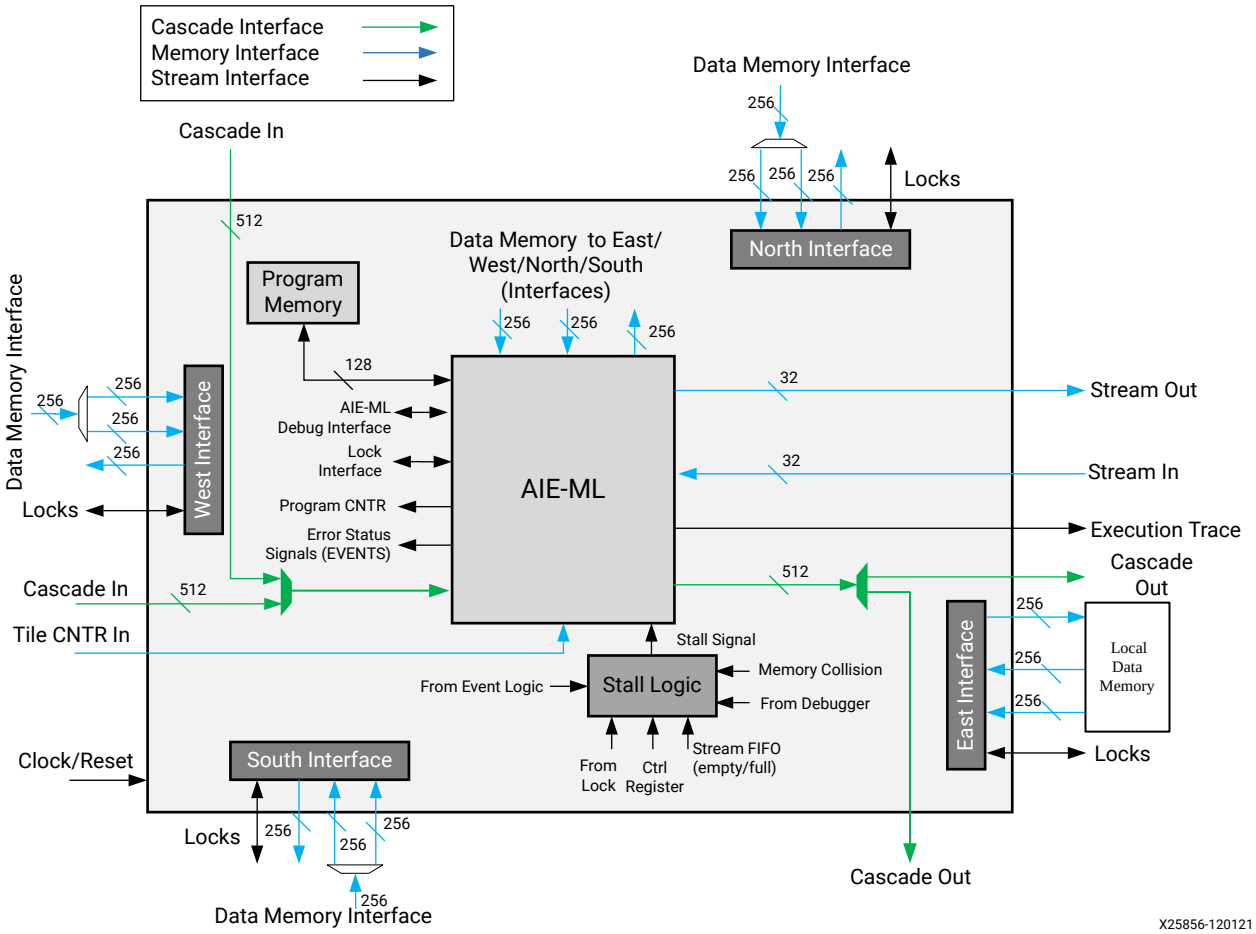
AIE-ML Interfaces

The AIE-ML has multiple interfaces. The following block diagram shows the interfaces.

- **Data Memory Interface:** The AIE-ML can access data memory modules on all four directions. They are accessed as one contiguous memory. The AIE-ML has two 256-bit wide load units and one 256-bit wide store unit. From the AIE-MLs perspective, the throughput of each of the loads (two) and store (one) is 256 bits per clock cycle.
- **Program Memory Interface:** This 128-bit wide interface is used by the AIE-ML to access the program memory. A new instruction can be fetched every clock cycle.
- **Direct AXI4-Stream Interface:** The AIE-ML has one 32-bit input AXI4-Stream interfaces and one 32-bit output AXI4-Stream interfaces. There are no 128-bit stream interfaces and no FIFO connected to either input or output of the stream.
- **Cascade Stream Interface:** The 512-bit accumulator data from one AIE-ML can be forwarded to another by using these cascade streams to form a chain. There is a small, two-deep, 512-bit wide FIFO on both the input and output streams that allow storing up to four values between AIE-MLs. In addition to the horizontal cascade, there is additional vertical cascade interface and the direction is controlled by the configuration memory-mapped AXI4 register.
- **Debug Interface:** This interface is able to read or write all AIE-ML registers over the memory-mapped AXI4 interface.
- **Hardware Synchronization (Locks) Interface:** This interface allows synchronization between two AIE-MLs or between an AIE-ML and DMA. The AIE-ML can access the lock modules in all four directions. There is also added support for semaphore locks.

- **Stall Handling:** An AIE-ML can be stalled due to multiple reasons and from different sources. Examples include: external memory-mapped AXI4 master (for example, PS), lock modules, empty or full AXI4-Stream interfaces, data memory collisions, and event actions from the event unit.
- **AIE-ML Event Interface:** This 16-bit wide EVENT interface can be used to set different events.
- **Tile Timer:** The input interface to read the 64-bit timer value inside the tile.
- **Execution Trace Interface:** A 32-bit wide interface where the AIE-ML generated packet-based execution trace can be sent over the AXI4-Stream.

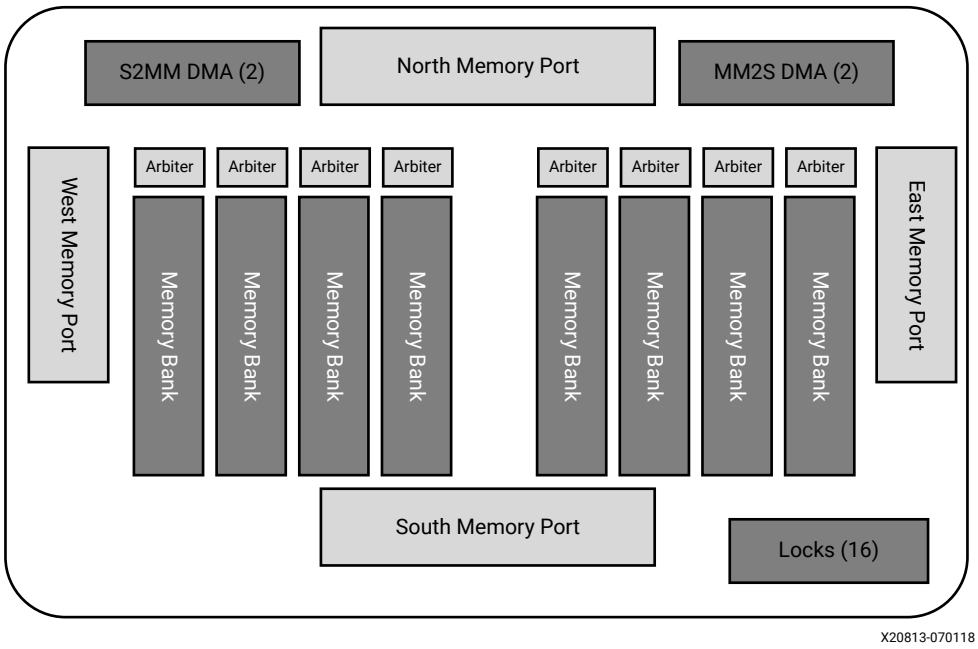
Figure 11: **AIE-ML Interfaces**



AIE-ML Memory Module

The AIE-ML memory module (shown in the following figure) contains eight memory banks, two input streams to memory map (S2MM) DMA, two memory-map to output DMA streams (MM2S), and a hardware synchronization module (locks). For each of the four directions (south, west, north, and east), there are separate ports for even and odd ports, and three address generators, two loads, and one store.

Figure 12: **AIE-ML Memory Module**



- **Memory Banks:** The AIE-ML data memory is 64 KB, organized as eight memory banks, where each memory bank is a 512 word x 128-bit single-port memory. From a programmer's perspective, every two banks are interleaved to form one bank, that is, a total of four banks of 16 KB. Each memory bank has a write enable for each 32-bit word. Banks [0-1] have ECC protection and banks [2-7] have parity check. Bank [0] starts at address 0 of the memory module. ECC protection is a 1-bit error detector/corrector and 2-bit error detector per 32-bit word.
- **Memory Arbitration:** Each memory bank has its own arbitrator to arbitrate between all requesters. The memory bank arbitration is round-robin to avoid starving any requester. It handles a new request every clock cycle. When there are multiple requests in the same cycle to the same memory bank, only one request per cycle is allowed to access the memory. The other requesters are stalled for one cycle and the hardware retries the memory request in the next cycle.

- **Tile DMA Controller:** The tile DMA has two incoming and two outgoing streams to the stream switches in the AIE-ML tile. The tile DMA controller is divided into two separate modules, S2MM to store stream data to memory (32-bit data) and MM2S to write the contents of the memory to a stream (32-bit data). Each DMA transfer is defined by a DMA buffer descriptor and the DMA controller has access to the 16 buffer descriptors. These buffer descriptors can also be accessed using a memory-mapped AXI4 interconnect for configuration. Each buffer descriptor contains all information needed for a DMA transfer and can point to the next DMA transfer for the DMA controller to continue with after the current DMA transfer is complete. The DMA controller also has access to the 16 locks that are the synchronization mechanism used between the AIE-ML and DMA or any external memory-mapped AXI4 master (outside of the AIE-ML array) and the DMA. Each buffer descriptor can be associated with locks. This is part of the configuration of any buffer descriptor using memory-mapped AXI4 interconnect. The DMA controller has the support for the following features:
  - Support 4D tensor address generation (including iteration-offset)
  - Supports task queues and task complete tokens
  - Supports S2MM finish on TLAST and out-of-order packets
  - Adds decompression to the two S2MM channels
  - Adds compression to the two MM2S channels
  - Supports task queue and task-complete-tokens (see Task-Completion-Tokens for more information)
- **Lock Module:** The AIE-ML memory module contains a lock module to achieve synchronization amongst the AIE-MLs, tile DMA, and an external memory-mapped AXI4 interface master (for example, the processor system (PS)). The AIE-ML features 16 semaphore locks. The semaphore lock has a larger state and no acquired bit; each lock state is 6-bit unsigned. The lock module handles lock requests from the AIE-MLs in all four directions, the local DMA controller, and memory-mapped AXI4.

Data Movement

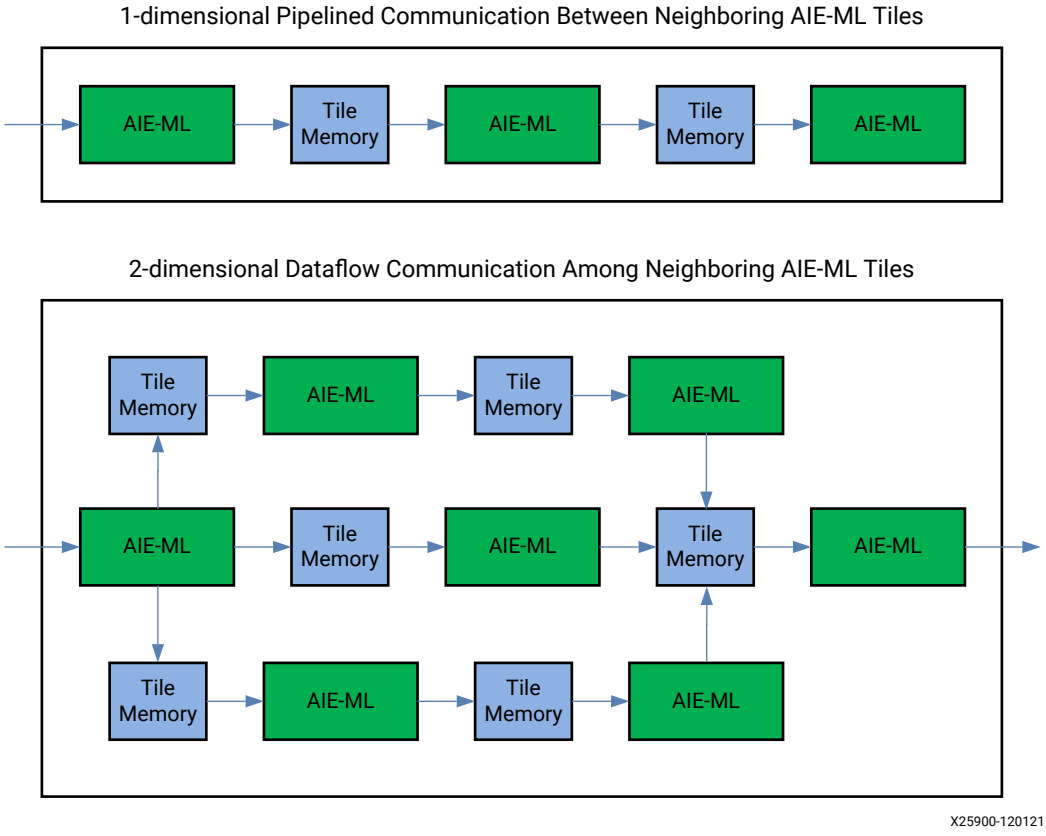
This section describes examples of the data communications within the AIE-ML array and between the AIE-ML tile and the PL, using various combinations of shared memory, AXI4-Stream interconnect, and the AI Engine tile DMA.

AIE-ML to AIE-ML Data Communication via Local Memory

In the case where multiple kernels fit in a single AIE-ML, communications between two consecutive kernels can be established using a common buffer in the shared memory. For cases where the kernels are in separate but neighboring AIE-ML, the communication is through the shared memory module. The processing of data movement can be through a simple pipeline or multiple parallel pipe stages (see the following figure). Communication between the two AIE-MLs can use ping and pong buffers (not shown in the figure) on separate memory banks to avoid access conflicts. The synchronization is done through locks. DMA and AXI4-Stream interconnect are not needed for this type of communication.

The following figures show examples of the data communication between the AIE-ML tiles. They are a logical representation of the AIE-ML tiles and shared memory modules.

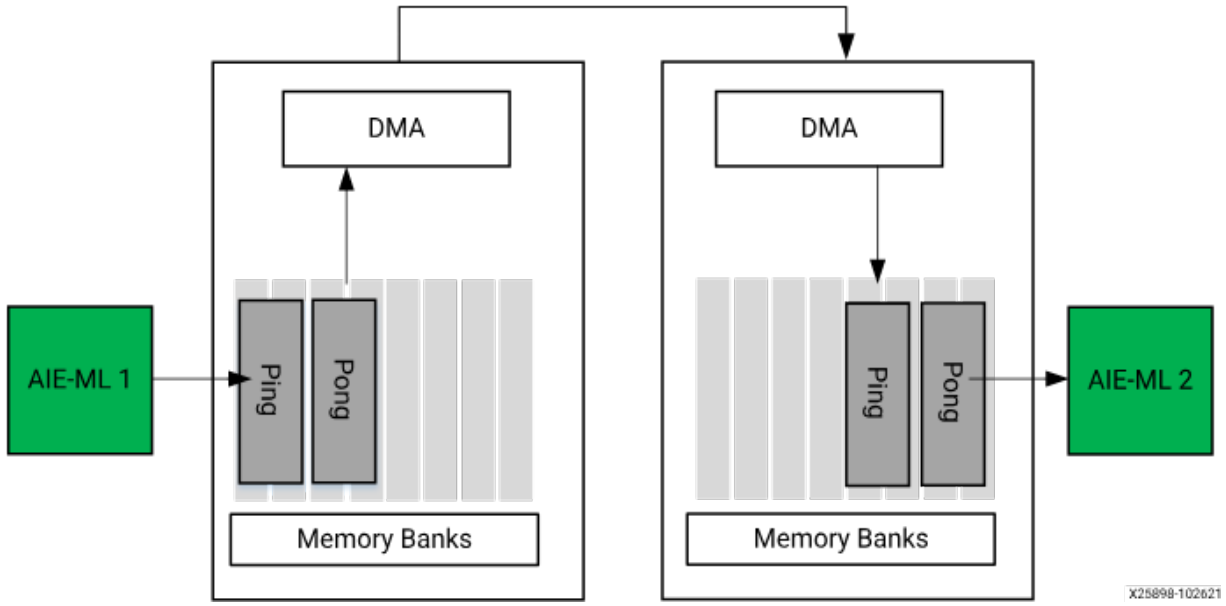
Figure 13: **Example of AIE-ML to AIE-ML Data Communication via Shared Memory**



AIE-ML Tile to AIE-ML Tile Data Communication via Memory and DMA

The communication described in the previous section is inside an AIE-ML tile or between two neighboring AIE-ML tiles. For non-neighboring AIE-ML tiles, a similar communication can be established using the DMA in the memory module associated with each AIE-ML tile, as shown in the following figure. The synchronization of the ping-pong buffers in each memory module is carried out by the locks in a similar manner to the AIE-ML to AIE-ML Data Communication via Local Memory section. In comparison to the neighboring tile communication, the main differences in this mode are increased communication latency and memory resources.

Figure 14: **Data Communication Between Two Non-neighboring AIE-ML Tiles**



AIE-ML Tile to AIE-ML Tile Data Communication via AXI4-Stream Interconnect

AIE-MLs can directly communicate through the AXI4-Stream interconnect without any DMA and memory interaction. As shown in the following figure, data can be sent from one AIE-ML to another through the streaming interface in a serial fashion, or the same information can be sent to an arbitrary number of AIE-ML tiles using a multicast communication approach. The streams can go in north/south and east/west directions. In all the streaming cases there are built-in hand-shake and backpressure mechanisms.

***Note*:** In a multicast communication approach, if one of the receivers is not ready, the whole broadcast stops until all receivers are ready again.

Figure 15: **AIE-ML to AIE-ML Data Communication via AXI4-Stream Interconnect**

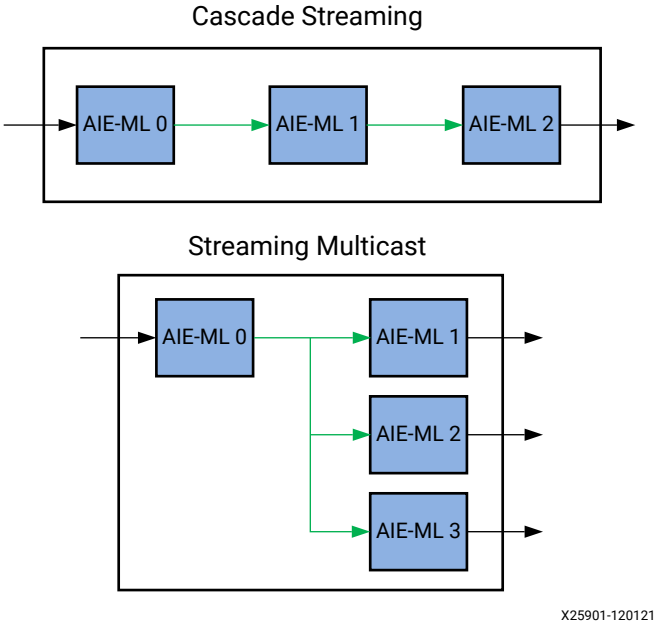


Figure 16: **Packet-Switched Split 1:N**

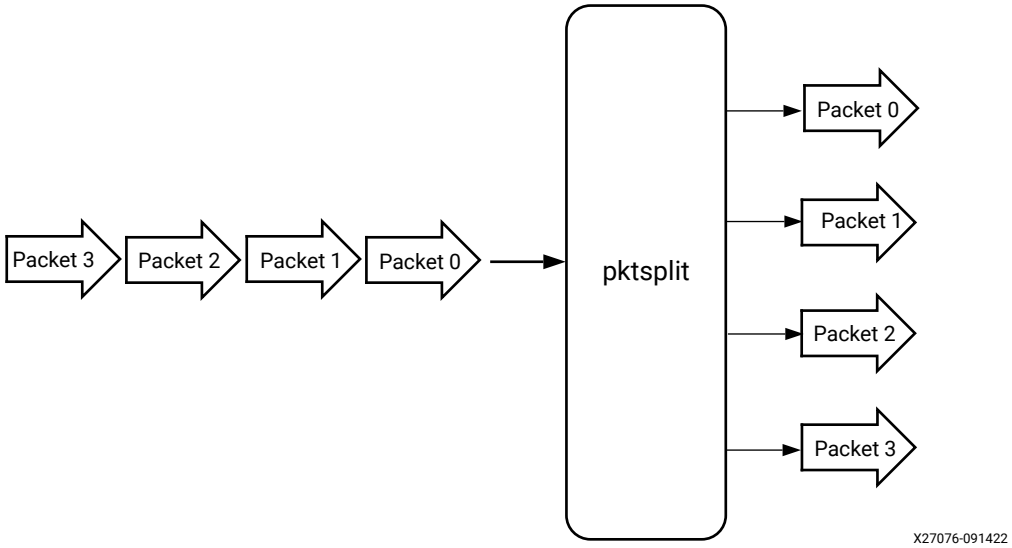
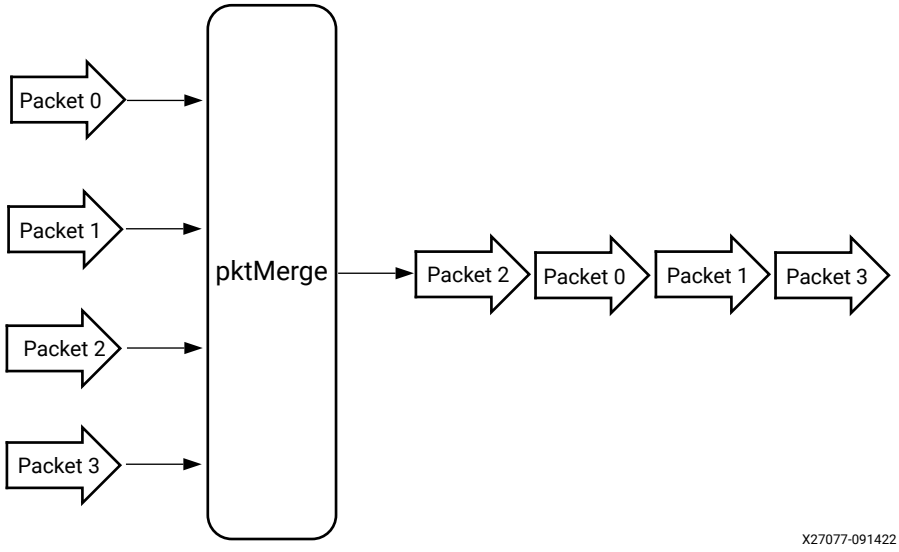


Figure 17: **Packet-Switched Merge N:1**



AIE-ML to PL Data Communication via Shared Memory

In the generic case, the PL block consumes data via the stream interface. It then generates a data stream and forwards it to the array interface, where inside there is a FIFO that receives the PL stream and converts it into an AIE-ML stream. The AIE-ML stream is then routed to the AIE-ML destination function. Depending on whether the communication is block-based or stream-based, DMA, and ping-pong buffers could be involved.

The following figure shows an example. The AIE-ML and PL can communicate using DMA in the AIE-ML tile. The DMA moves the stream into a memory block neighboring the consuming AIE-ML. The first diagram represents the logical view and the second diagram represents the physical view.

Figure 18: **Logical View of AIE-ML to PL Data Communication via Shared Memory**

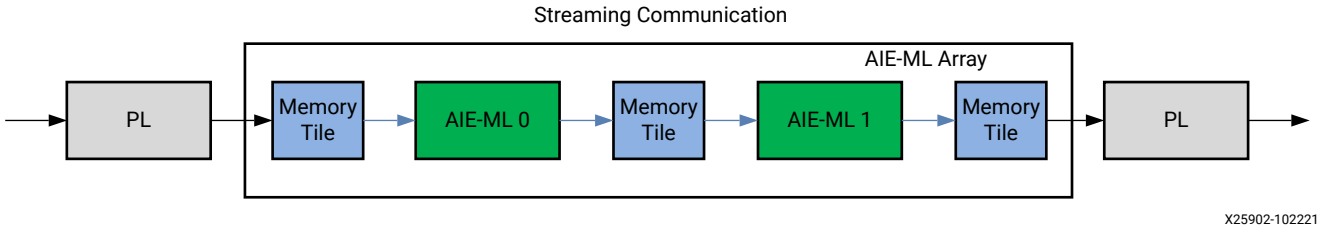
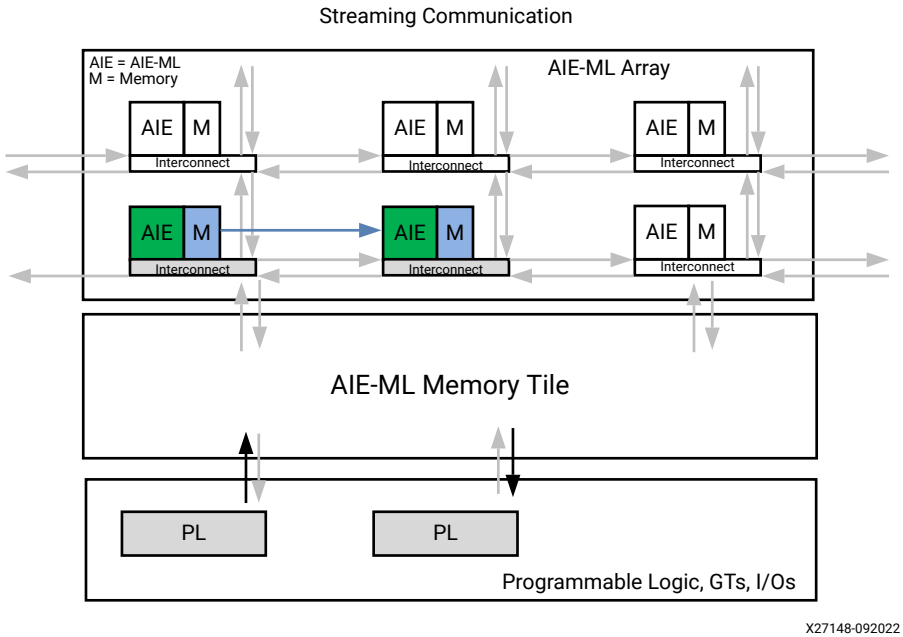


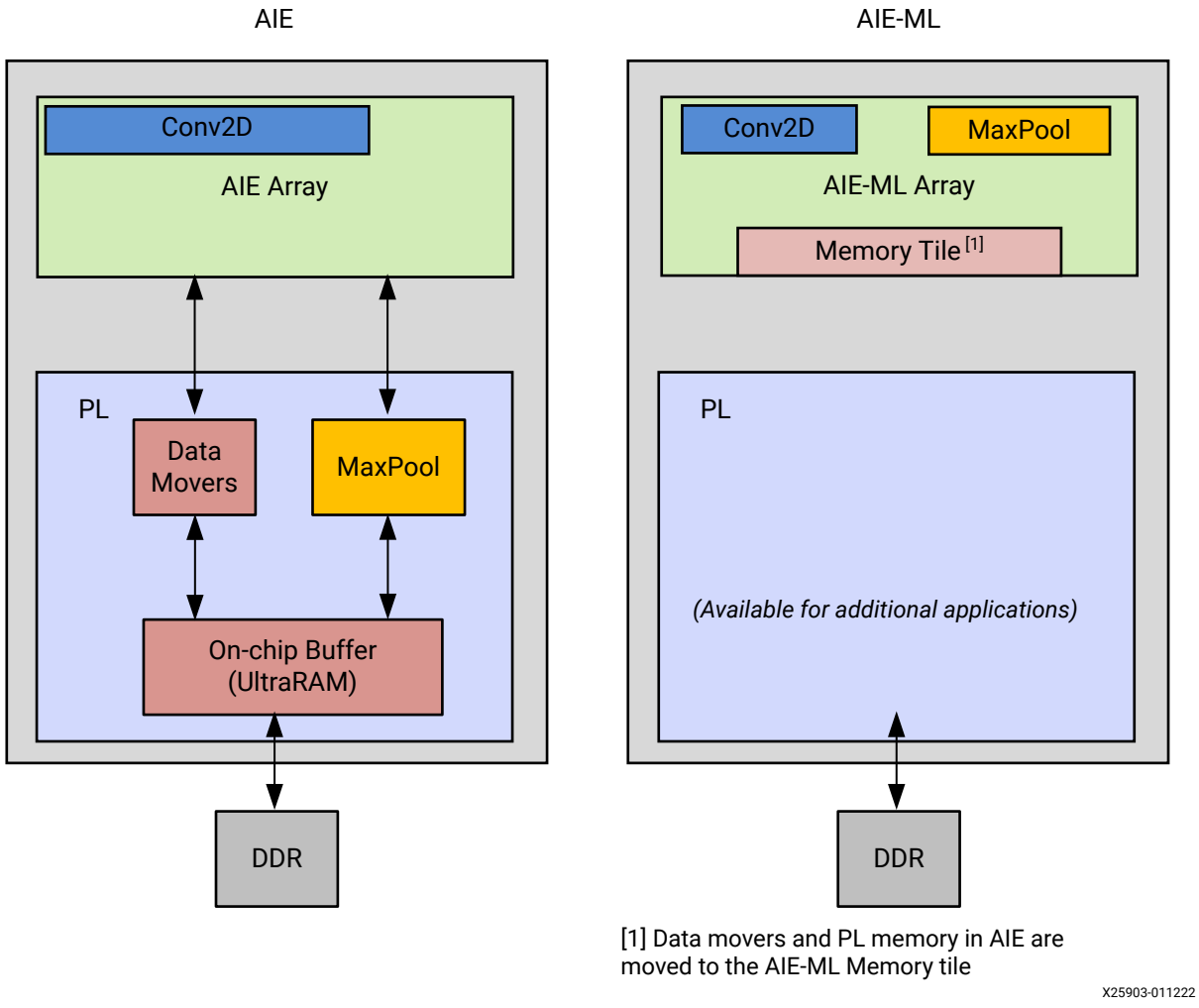
Figure 19: **Physical View of AIE-ML to PL Data Communication via Shared Memory**



AIE-ML Data Movement with Memory Tiles

The AIE-ML Memory tile is introduced in the AIE-ML architecture to significantly increase the on-chip memory inside the AIE-ML array. This new functional unit reduced the utilization of PL resources including block RAMs and UltraRAMs in ML applications. The following figure illustrates the general concept:

Figure 20: **AIE-ML Data Movement and Use of Memory Tile**



Depending on the characteristics of the ML applications and bandwidth requirement, the AIE-ML data movement architecture supports different dataflow mappings where either the activations and/or the weights are stored in the AIE-ML memory tiles. The following figure shows examples of data alternative mappings supported by the AIE-ML architecture.

Figure 21: **Dataflow Mappings**

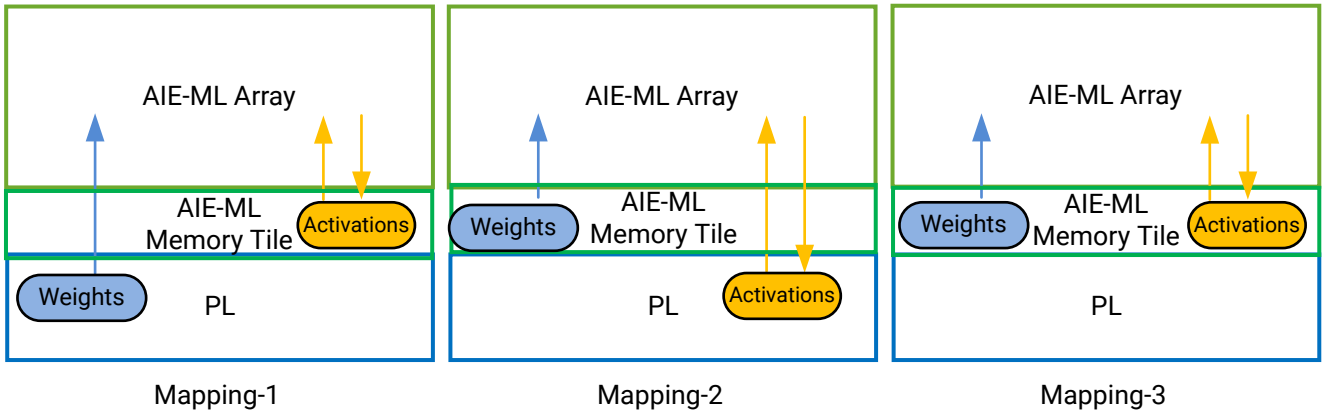


Figure 22: **Data From External Memory to AIE-ML Array**

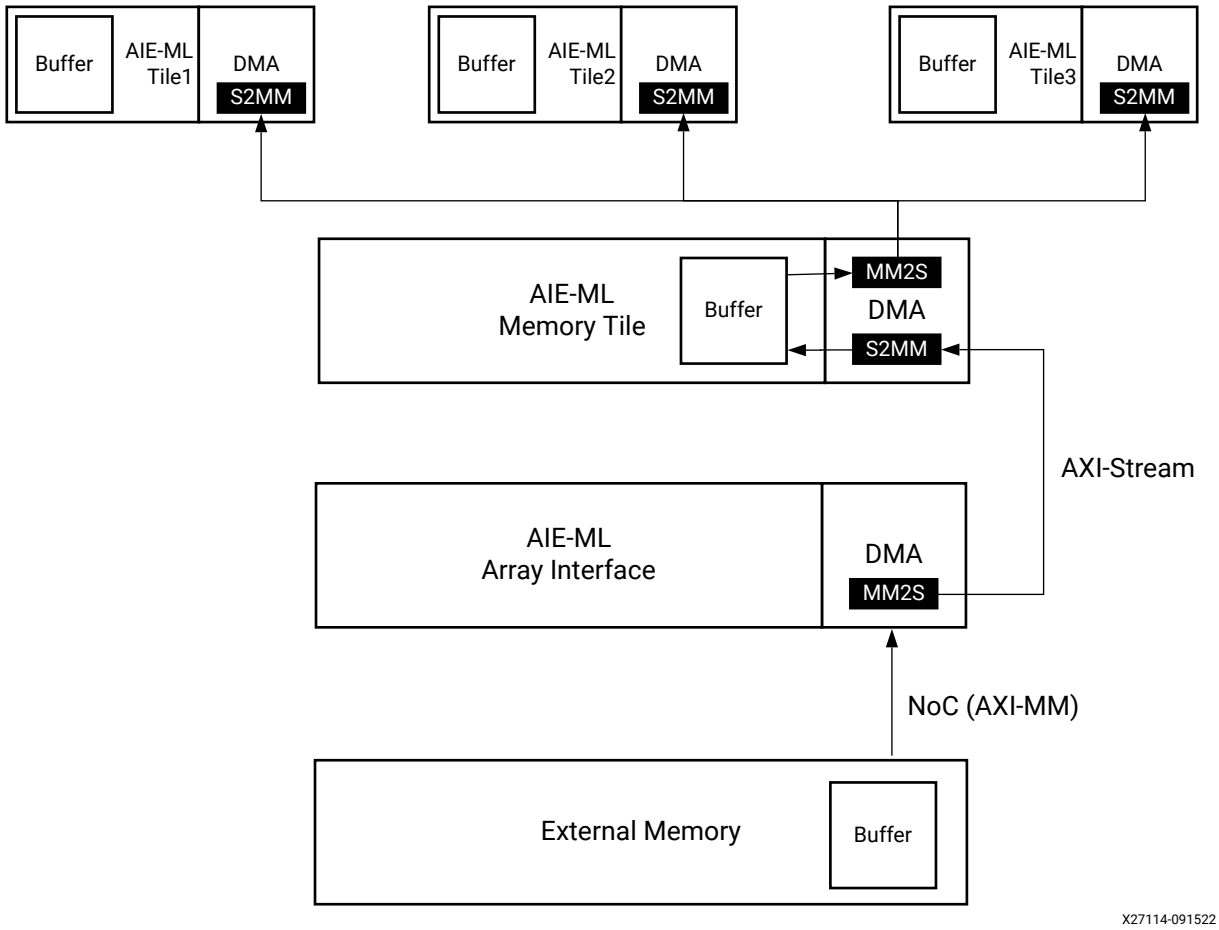
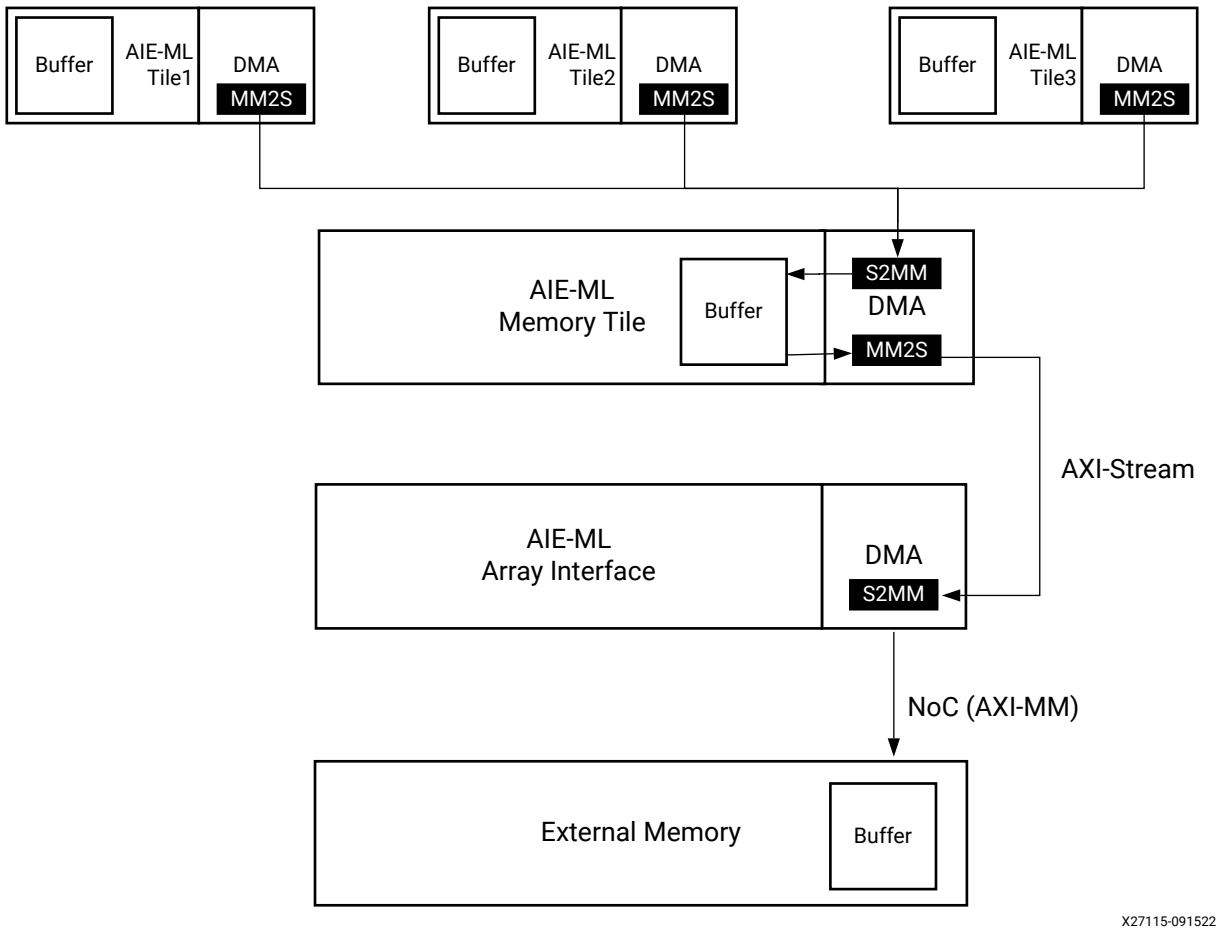


Figure 23: **Data Copy from AIE-ML Array to External Memory**



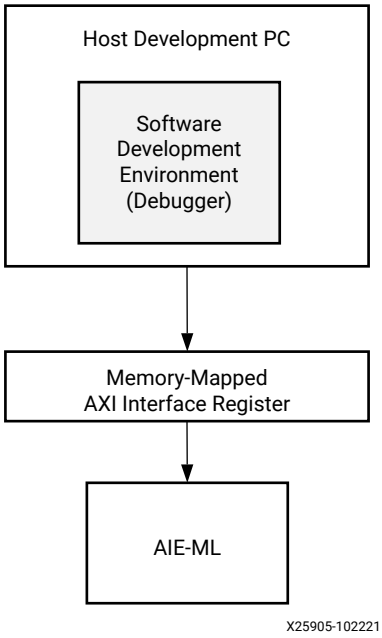
AIE-ML Debug

Debugging the AIE-ML uses the memory-mapped AXI4 interface. All the major components in the AIE-ML array are memory mapped.

- Program memories
- Data memories
- AIE-ML registers
- DMA registers
- Lock module registers
- Stream switch registers
- AIE-ML break points registers
- Events and performance counters registers

These memory-mapped registers can be read and/or written from any master that can produce memory-mapped AXI4 interface requests (PS, PL, and PMC). These requests come through the NoC to the AIE-ML array interface, and then to the target tile in the array. The following figure shows a typical debugging setup involving a software development environment running on a host development system combined with its integrated debugger.

Figure 24: **Overview of the AIE-ML Debug Interface**



The debugger connects to the platform management controller (PMC) on an AIE-ML enabled Versal device either using a JTAG connection or the AMD high-speed debug port (HSDP) connection.

AIE-ML Trace and Profiling

The AIE-ML tile has provisions for trace and profiling. It also has configuration registers that control the trace and profiling hardware.

Trace

There are two trace streams coming out of each AIE-ML tile. One stream from the AIE-ML and the other from the memory module. Both these streams are connected to the tile stream switch. There is a trace unit in each AIE-ML module and memory module in an AIE-ML tile, and an AIE-ML programmable logic (PL) module in an AIE-ML PL interface tile (see types of array interface tiles). The units can operate in the following modes:

- AIE-ML modes
  - Event-time
  - Event-PC
  - Execution-trace
- AIE-ML memory module mode
  - Event-time
- AIE-ML PL module mode
  - Event-time

The trace is output from the unit through the AXI4-Stream as an AIE-ML packet-switched stream packet. The packet size is 8x32 bits, including one word of header and seven words of data. The information contained in the packet header is used by the array AXI4-Stream switches to route the packet to any AIE-ML destination it can be routed to, including AIE-ML local data memory through the AIE-ML tile DMA, external DDR memory through the AIE-ML array interface DMA, and block RAM or UltraRAM through the AIE-ML to PL AXI4-Stream.

The event-time mode tracks up to eight independent numbered events on a per-cycle basis. A trace frame is created to record state changes in the tracked events. The frames are collected in an output buffer into an AIE-ML packet-switched stream packet. Multiple frames can be packed into one 32-bit stream word but they cannot cross a 32-bit boundary (filler frames are used for 32-bit alignment).

In the event-PC mode, a trace frame is created each cycle where any one or more of the eight watched events are asserted. The trace frame records the current program counter (PC) value of the AIE-ML together with the current value of the eight watched events. The frames are collected in an output buffer into an AIE-ML packet-switched stream packet.

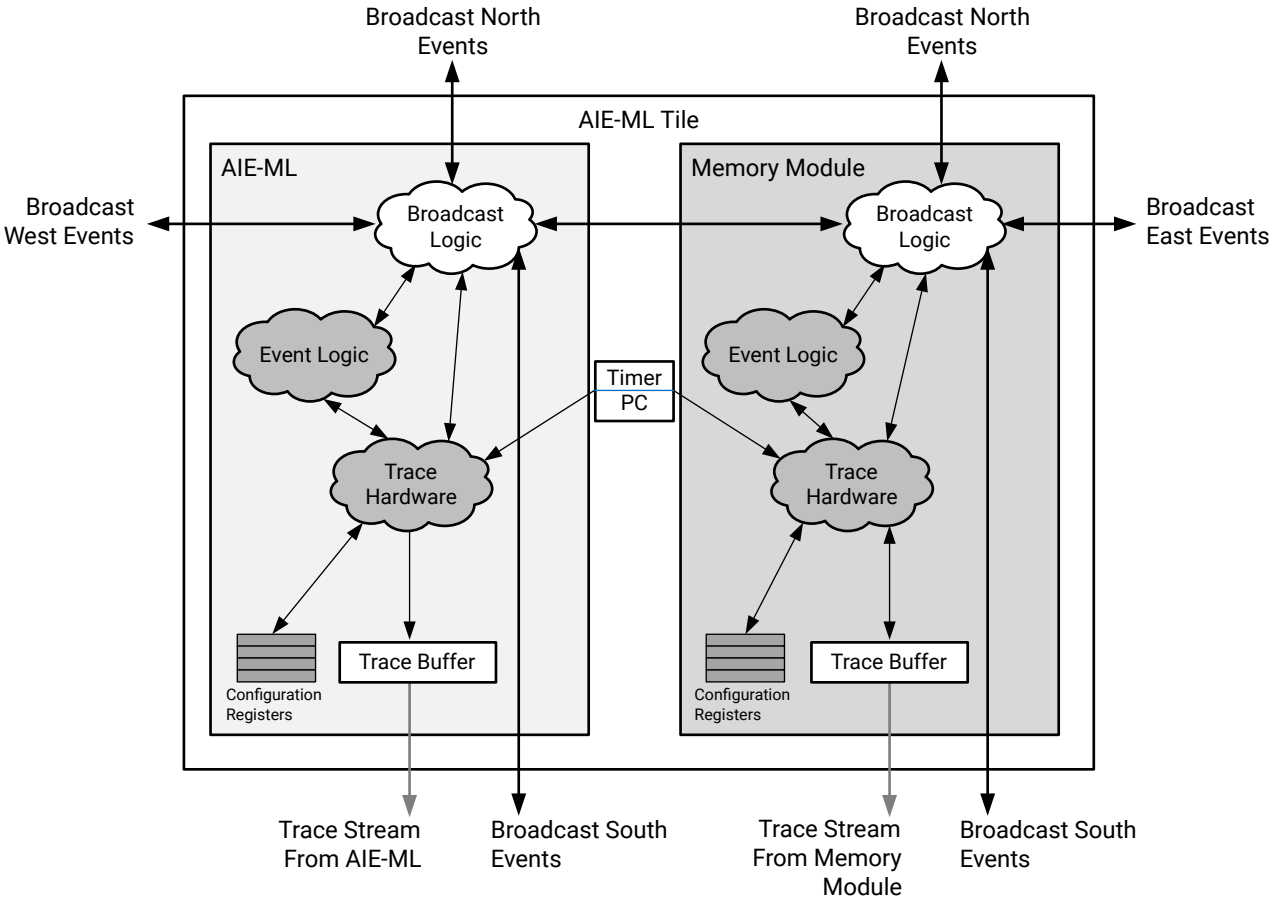
The trace unit in the AIE-ML can operate in execution-trace mode. In real time, the unit will send, via the AXI4-Stream, a minimum set of information to allow an offline debugger to reconstruct the program execution flow. This assumes the offline debugger has access to the ELF. The information includes:

- Conditional and unconditional direct branches
- All indirect branches
- Zero-overhead-loop LC

The AIE-ML generates the packet-based execution trace, which can be sent over the 32-bit wide execution trace interface. The following figure shows the logical view of trace hardware in the AIE-ML tile. The two trace streams out of the tile are connected internally to the event logic, configuration registers, broadcast events, and trace buffers.

***Note*:** The different operating modes between the two modules are not shown.

Figure 25: **Logical View of AIE-ML Trace Hardware**

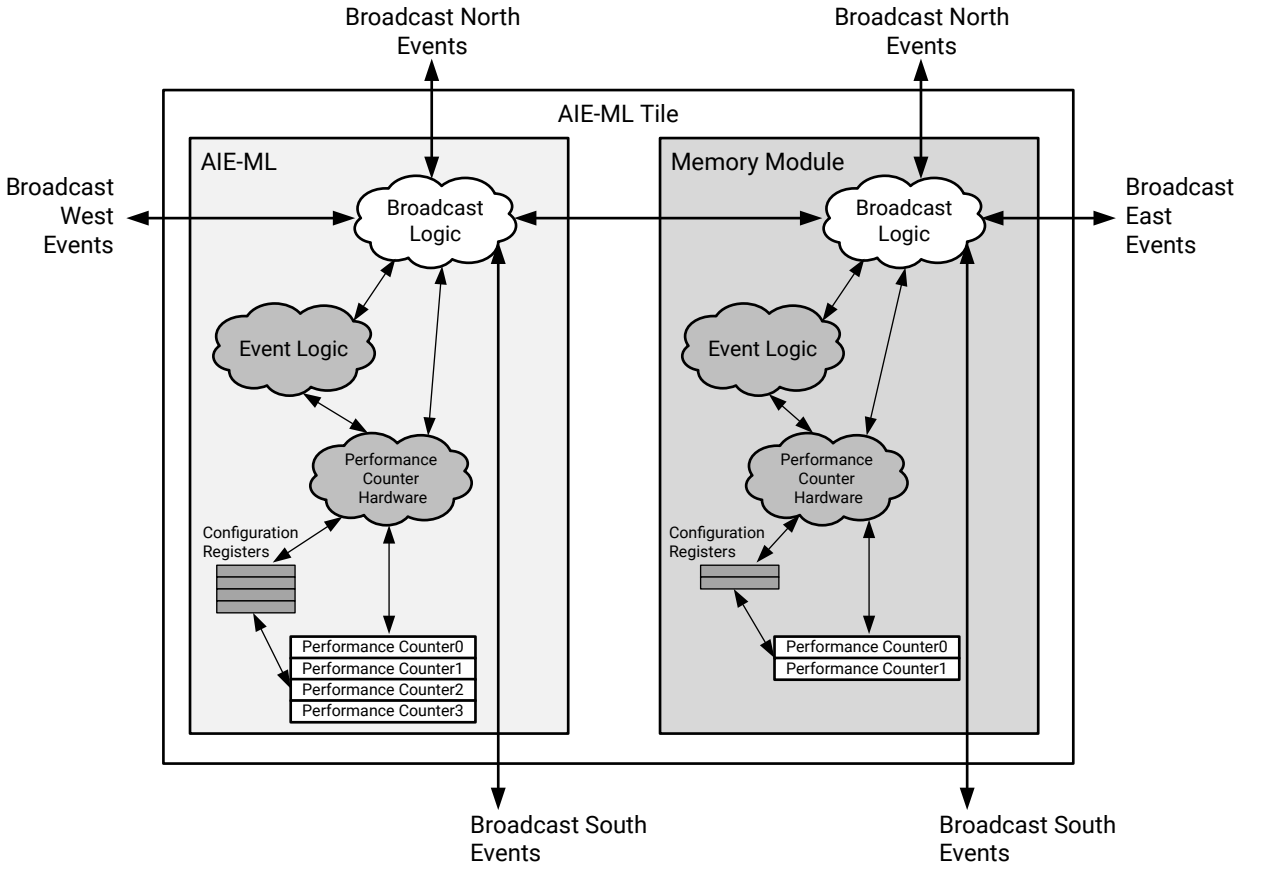


To control the trace stream for an event trace, there is a 32-bit `trace_control0/1` register to start and stop the trace. There are also the `trace_event0/1` registers to program the internal event number to be added to the trace.

Profiling (Performance Counters)

The AIE-ML array has performance counters that can be used for profiling. The AIE-ML has four performance counters that can be configured to count any of the internal events. It will either count the occurrence of the events or the number of clock cycles between two defined events. The memory module and the PL modules in the PL and NoC array interface tiles each have two performance counters that can be configured to perform similar functions. The following figure shows a high-level logical view of the profiling hardware in the AIE-ML tile.

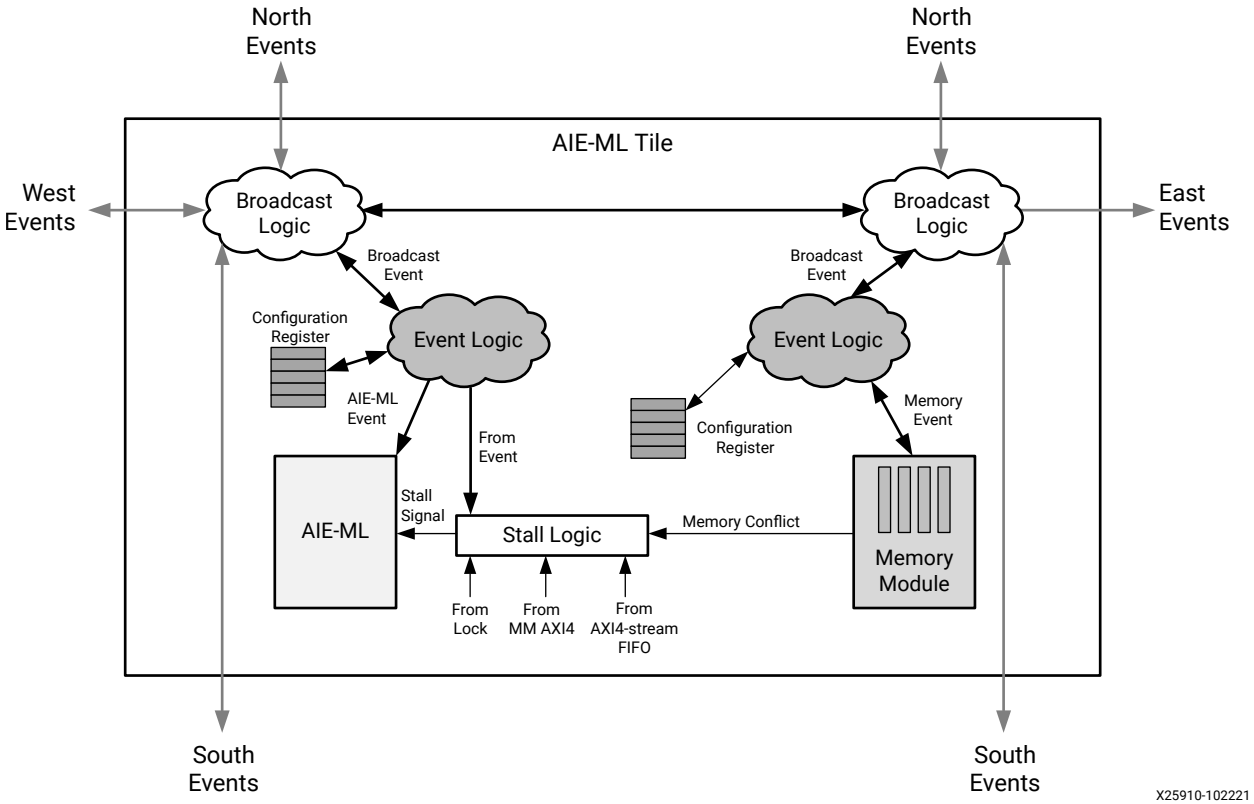
Figure 26: **Logical View of AIE-ML Profiling**



AIE-ML Events

The AIE-ML and memory modules each have an event logic unit. Each unit has a defined set of local events. The following diagram shows the high-level logical view of events in the AIE-ML tile. The event logic needs to be configured with a set of action registers that can be programmed over the memory-mapped AXI4 interface. There are separate sets of registers associated with event logic for the AIE-ML and memory modules. Event actions can be configured to perform a task whenever a specific event occurs. Also, there is separate broadcast logic to send event signals to neighboring modules.

Figure 27: **Events in an AIE-ML Tile**



Event Actions

An event itself does not have an action, but events can be used to create an action. Event broadcast and event trace can be configured to monitor the event. Examples of event actions include:

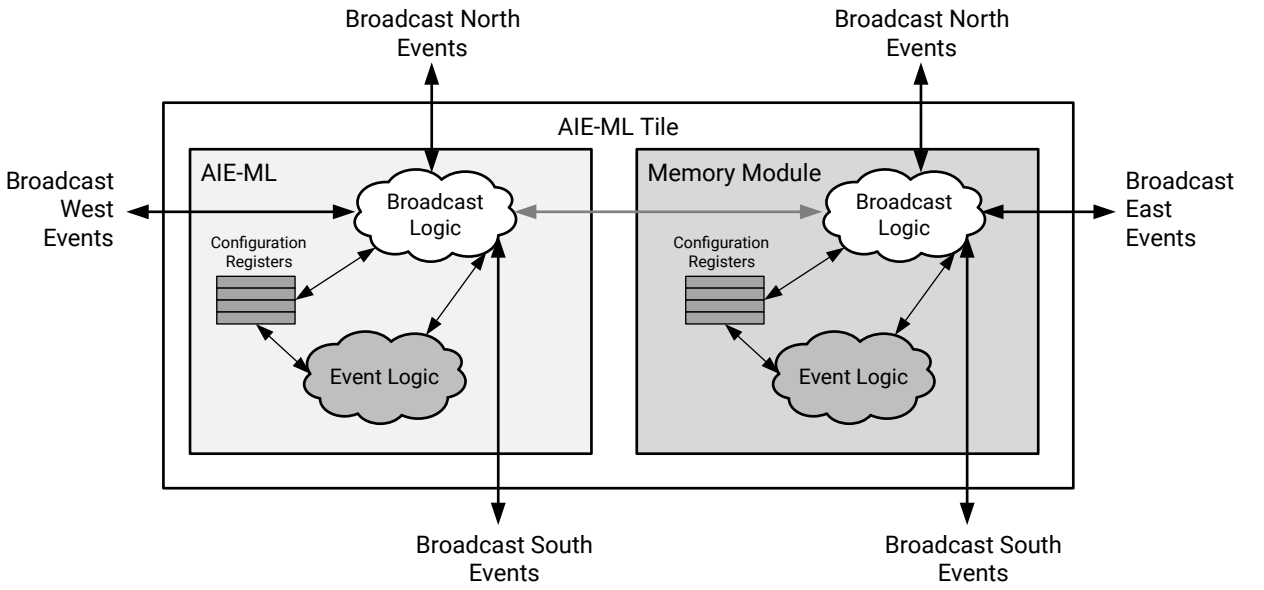
- Enable, disable, or reset of an AIE-ML
- Debug-halt, resume, or single-step of an AIE-ML
- Error halt of an AIE-ML
- Resynchronize timer
- Start and stop performance counters
- Start and stop trace streams
- Generate broadcast events
- Drive combo events
- ECC scrubbing event

For each of these event actions there are associated registers where a 7-bit event number is set and is used to configure the action to trigger on a given event.

Event Broadcast

Broadcast events are both the events and the event actions because they are triggered when a configured event is asserted. The following figure shows the logical view of the broadcast logic inside the AIE-ML tile. The units in the broadcast logic in the AIE-ML and memory modules receive input from and send out signals in all four directions. The broadcast logic is connected to the event logic, which generates all the events. There are configuration registers to select the event sent over, and mask registers to block any event from going out of the AIE-ML tile.

Figure 28: **AIE-ML Broadcast Events**



Each module has an internal register that determines the broadcast event signal to broadcast in the other directions. To avoid broadcast loops, the incoming event signals are ORed with the internal events to drive the outgoing event signals according the following list:

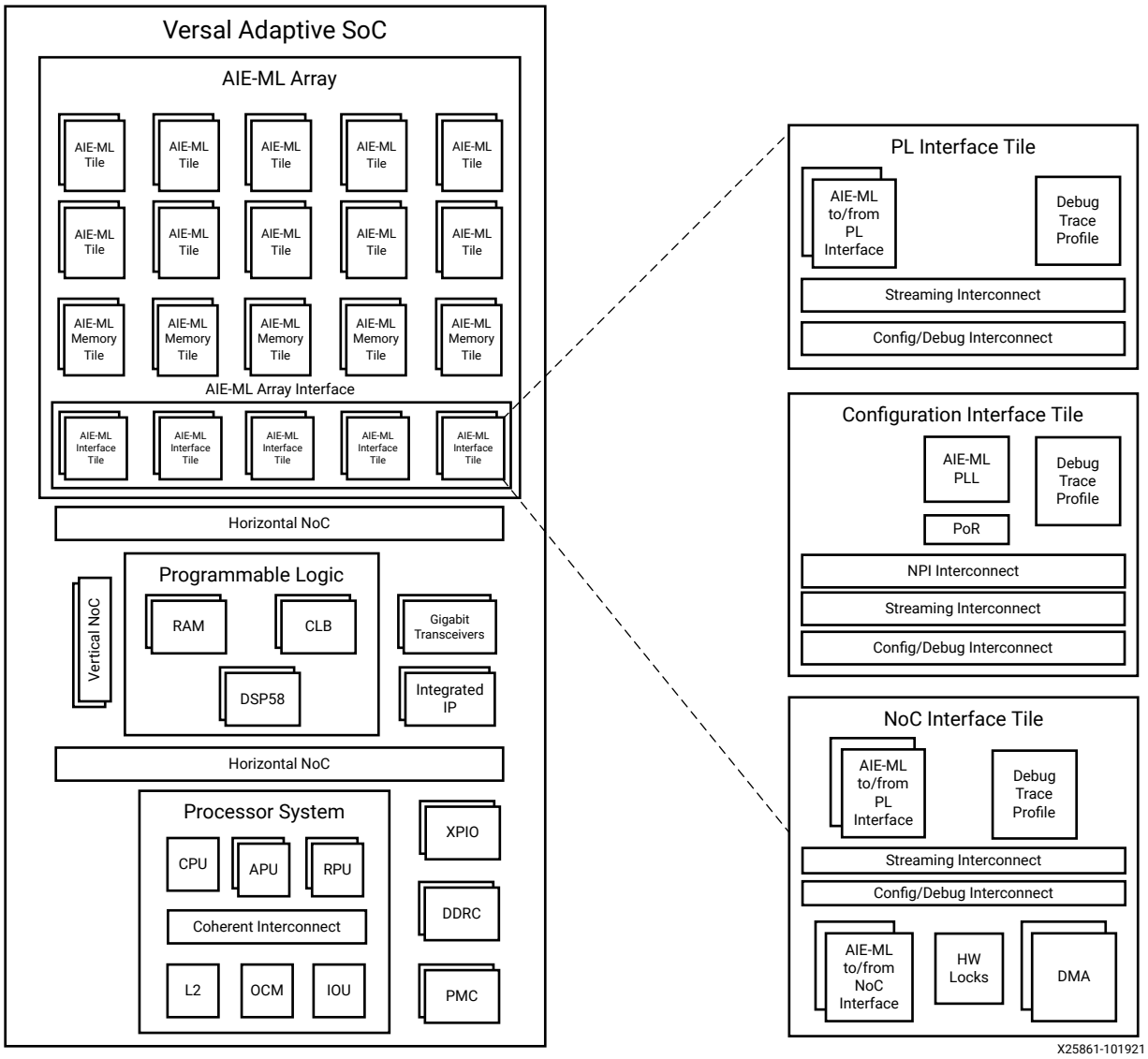
- Internal, east, north, south → west
- Internal, west, north, south → east
- Internal, south → north
- Internal, north → south

TIP: *The AIE-ML module east broadcast event interface is internally connected to the memory module west broadcast event interface and does not go out of the AIE-ML tile. In the AIE-ML module, there are 16 broadcast events each in the north, south, and west directions. In the memory module, there are 16 broadcast events each in the north, south, and east directions.*

AIE-ML Array Interface Architecture

The AIE-ML is arranged in a 2D array as shown in the following figure. The AIE-ML array interface provides the necessary functionality to interface with the rest of the device. The AIE-ML array interface has three types of AIE-ML interface tiles. There is a one-to-one correspondence of interface tiles for every column of the AIE-ML array. The interface tiles form a row and move memory-mapped AXI4 and AXI4-Stream data horizontally (left and right) and also vertically up a AIE-ML tile column. The AIE-ML interface tiles are based on a modular architecture, but the final composition is device specific. Refer to the following figure for the internal hierarchy of the AIE-ML array interface in the AIE-ML array.

Figure 29: **AIE-ML Array Interface Hierarchy**



The types of array interface tiles and the modules within them are described in this section.

- AIE-ML PL interface tile
  - PL module includes:
    - AXI4-Stream switch
    - Memory-mapped AXI4 switch
    - AIE-ML to PL stream interface
    - Control, debug, and trace unit
- AIE-ML configuration interface tile (exactly one instance per AIE-ML array)
  - PLL for AIE-ML clock generation
  - Power-on-reset (POR) unit

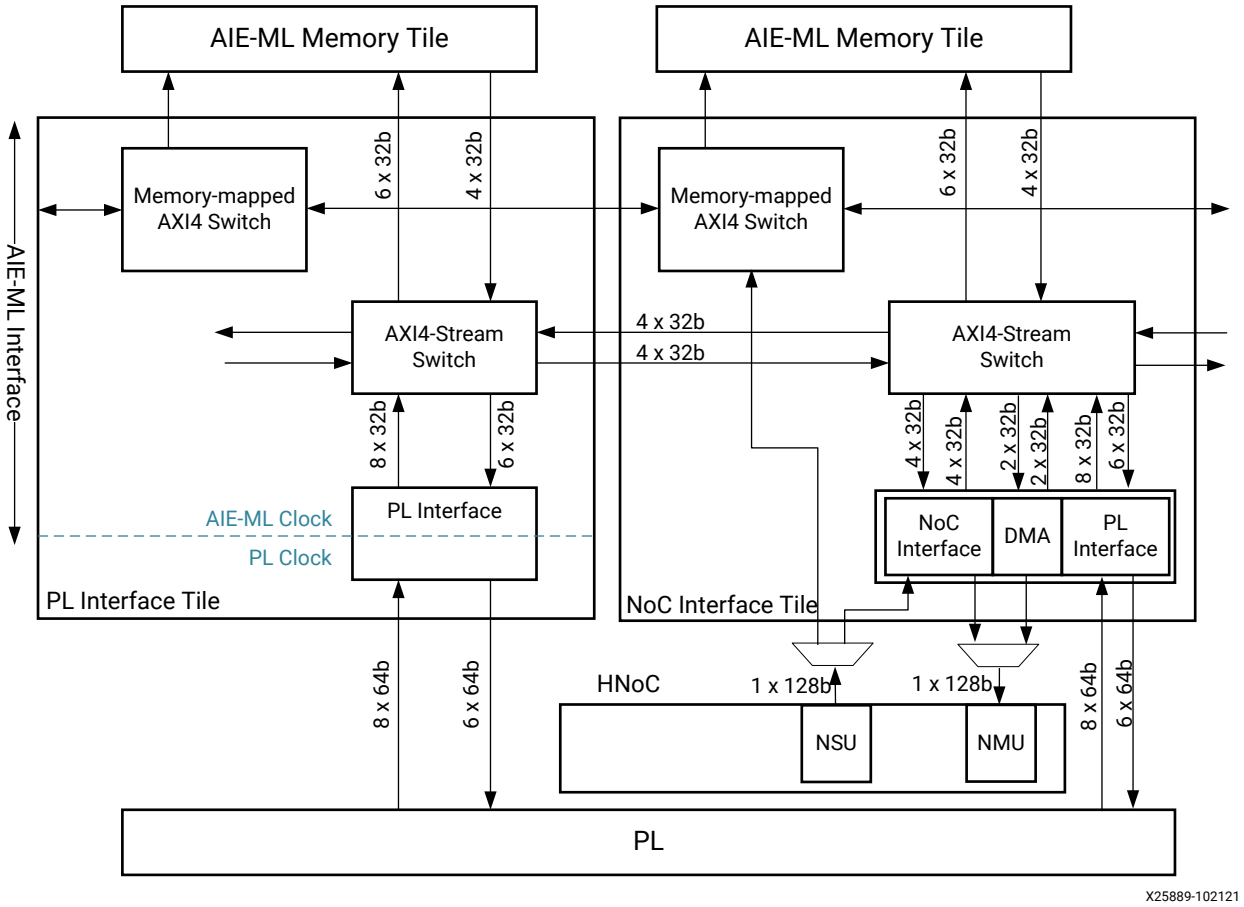
    - Interrupt generation unit
    - Dynamic function exchange (DFx) logic
    - NoC peripheral interconnect (NPI) unit
    - AIE-ML array global registers that control global features such as PLL/clock control, secure/non-secure behavior, interrupt controllers, global reset control, and DFx logic
- AIE-ML NoC interface tile
    - PL module (see previous description)
    - NoC module with interfaces to NMU and NSU includes:
        - Bi-directional NoC streaming interface
        - Array interface DMA

AIE-ML Array Interface

The AIE-ML array interface consists of PL and NoC interface tiles. There is also one configuration interface tile per device. The following figure shows the array interface connectivity that the AIE-ML array uses to communicate with other blocks in the Versal architecture. Also specified are the number of streams in the AXI4-Stream interconnect interfacing with the PL, NoC, or AIE-ML tiles, and between the AXI4-Stream switches.

TIP: *The exact number of PL and NoC interface tiles is device specific. The Versal Architecture and Product Data Sheet: Overview (DS950) lists the size of the AIE-ML array.*

Figure 30: **AIE-ML Array Interface Topology**



***Note*:** The AIE-ML F_{MAX} is 1 GHz for the -1L speed grade devices. The PL clock should be set at half that speed to 500 MHz. There is also a clock domain crossing at the NoC interface tile between the clocks for the AIE-ML and the NoC.

***Note*:** In both the NMU and NSU, AXI4 and NoC streams are mutually exclusive. Each of the two interfaces has a configuration register that selects between AXI4 and NoC stream. When NoC streams are enabled, they can be connected to up to four AXI4-Streams to/from the AIE4-Stream Switch. The outgoing NoC stream to the NMU is time-multiplexed across the incoming streams. Traffic from each stream is tagged with a separate TDEST set via configuration registers. Similarly, the incoming NoC stream from the NSU is demultiplexed across up to four streams to the AIE4-Stream Switch based on TDEST.

The types of interfaces to the PL and NoC are:

- Memory-mapped AXI4 interface: the communication channel is from the NSU to the AIE-ML as a slave
- AXI4-Stream interconnect has four types of interfaces:
  - Connection to stream switches in other tiles
  - Bi-directional connection to the PL streaming interface

- Connection to the array interface DMA that generates traffic into the NoC using a memory-mapped AXI4 interface
- Direct connection to the NoC streaming interfaces (NSU and NMU)

The AIE-ML array interface tiles manage the two high performance interfaces:

- AIE-ML to PL
- AIE-ML to NoC

The following tables summarize the bandwidth performance of the AIE-ML array interface with the PL, the NoC, and the AIE-ML tile. The bandwidth performances are specified per each AIE-ML column for the -1L speed grade devices. There is a reduction in the number of connections per column between the PL to AIE-ML interface and the AXI4-Stream switch to the AIE-ML tile. This is to support the horizontally connected stream switches that provide additional horizontal routing capability. The total bandwidth for the various devices across speed grades can be found in the *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957) or *Versal AI Edge Series Data Sheet: DC and AC Switching Characteristics* (DS958).

Table 4: **AIE-ML Array Interface to PL Interface Bandwidth Performance**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
PL to AIE-ML array interface	8	64[1]	PL (500 MHz)	4	32
AIE-ML array interface to PL	6	64	PL (500 MHz)	4	24
AIE-ML array interface to AXI4-Stream switch	8	32	AIE-ML (1 GHz)	4	32
AXI4-Stream switch to AIE-ML array interface	6	32	AIE-ML (1 GHz)	4	24
Horizontal interface between AXI4-Stream switches[2]	4	32	AIE-ML (1 GHz)	4	16

Notes:

1. All streams to and from the PL are 64 bits on the PL side. The streams can be converted to 32-bit wide, but only one 32-bit word is valid out of the 64-bit wide stream. Two 64-bit wide streams can be combined to form a 128-bit wide stream, but the number of connections are halved.
2. The aggregate bandwidth shown is in the east/west direction. There are two sets of connections going in and out of each AXI4-Stream switch.

Table 5: **AIE-ML Array Interface to NoC AXI4-Stream Interface Bandwidth Performance**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
AIE-ML to NoC (NoC side)	1	128	NoC Interface (960 MHz)[1]	16	16
AIE-ML to NoC (AIE-ML side)	4	32	AIE-ML (1 GHz)	4	16
NoC to AIE-ML (NoC side)	1	128	NoC Interface (960 MHz)[1]	16	16
NoC to AIE-ML (AIE-ML side)	4	32	AIE-ML (1 GHz)	4	16

Notes:

1. The frequency is based off of a –1 speed grade.

Table 6: **AIE-ML Array Interface to AIE-ML Tile Bandwidth Performance**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
AXI4-Stream switch to AIE-ML tile	6	32	AIE-ML (1 GHz)	4	24
AIE-ML tile to AXI4-Stream switch	4	32	AIE-ML (1 GHz)	4	16

The following sections contain additional AIE-ML array interface descriptions. The AIE-ML tiles are described in Chapter 2: AIE-ML Tile Architecture.

Features of the AIE-ML Array Interface

- **Memory Mapped AXI4 Interconnect:** Provides functionality to transfer the incoming memory-mapped AXI4 requests from the NoC to inside the AIE-ML array.
- **AXI4 Master: Interface-DMA:** Memory mapped access to the rest of the device via the NoC, including external memory.
- **AXI4-Stream Interconnect:** Leverages the AIE-ML tile streaming interconnect functionality.
- **AIE-ML to PL Interface:** The AIE-ML PL modules directly communicate with the PL. Asynchronous FIFOs are provided to handle clock domain crossing.

- **AIE-ML to NoC Interface:** The AIE-ML to NoC module handles the conversion of 128-bit NoC streams into 32-bit AIE-ML streams (and vice versa). It provides the interface logic to the NoC components (NMU and NSU). Level shifting is performed because the NMU and NSU are in a different power domain from the AIE-ML.
- **Hardware Locks:** Leverages the corresponding unit in the AIE-ML tile and is accessible from the AIE-ML array interface or an external memory-mapped AXI4 master, the module is used to synchronize the array interface to DMA transfer to/from external memory. The lock module has 16 semaphore locks and the lock state is 6-bit unsigned.
- **Debug, Trace, and Profile:** Leverages all the features from the AIE-ML tile for local event debugging, tracing, and profiling.

Array Interface Memory-Mapped AXI4 Slave Interconnect

The main task of the AIE-ML memory-mapped AXI4 interconnect is to allow external access to internal AIE-ML tile resources such as memories and registers for configuration and debug. It is not designed to carry the bulk of the data movement to and from the AIE-ML array. The memory-mapped AXI4 interfaces are all interconnected across the AIE-ML array interface row. This enables the memory-mapped AXI4 interconnects in the array interface tiles to move incoming memory-mapped signals to the correct column horizontally and then forward them vertically to the memory-mapped AXI4 interconnect in the bottom AIE-ML tile of that column through a switch.

Each memory-mapped AXI4 interface is a 32-bit address with 32-bit data. The maximum memory-mapped AXI4 bandwidth is designed to be 1.5 GB/s. The memory-mapped AXI4 interface supports 1 MB address space per tile.

To feed the memory-mapped AXI4 interface, the NoC module contains a memory-mapped AXI4 bridge that accepts memory-mapped AXI4 transfers from the NoC NSU interface, and acts as a memory-mapped AXI4 master to the internal memory-mapped AXI4 interface switch.

Array Interface DMA Memory-Mapped AXI4 Master Interface

The AIE-ML array interface DMA provides direct access to external memory. The DMA is an AXI4 master, capable of issuing read and write requests to the NoC NMU interface, and hence to any AXI4 slave on the Versal device provided the NoC configuration provides the path. The DMA supports a 32-bit aligned start address. Each DMA channel generates addresses based on the base address in the buffer descriptor that stores the incremental address offset between BD calls and avoids the need to reconfigure a BD for subsequent buffer transfers.

The DMA is composed of four independent channels, two MM2S (read from external memory), and two S2MM (write to external memory). Each channel can sustain 4 bytes per cycle (4 Gb/s at 1 GHz) throughput, giving a total of up to 8 Gb/s read and 8 Gb/s write in parallel per interface tile.

MM2S Channels (two in total) :

- 32-bit stream master interface per channel
- 128-bit AXI4 master read interface, shared between two channels
- 4D tensor address generation (including iteration-offset)
- Access shared lock module (local to interface tile)
- Support task queue and task-complete-tokens; queue depth is four tasks per channel (see Task-Completion-Tokens for more information)

S2MM Channels (two in total)

- 32-bit stream slave interface per channel
- 128-bit AXI4-MM master write interface, shared between two channels
- 4D tensor address generation (including iteration-offset)
- Access shared lock module (local to interface tile)
- Support task queue and task-complete-tokens; queue depth is four tasks per channel (see Task-Completion-Tokens for more information)
- Support out-of-order packet transfer, finish-on-TLAST enabling compressed spill and restore of intermediate results to external memory

Buffer descriptors (BD):

- 16 shared BDs

The interface DMA, together with tile and memory tile DMAs, and the streaming interconnect supports the following data-flows (non-exhaustive list).

- Buffer copy from external-memory to memory tile
- Buffer copy from external-memory to AIE-ML tile data memory
- Buffer copy from memory tile to external-memory
- Buffer copy from AIE-ML tile data memory to external-memory

Array Interface AXI4-Stream Interconnect

The main task of the AIE-ML AXI4-Stream switch is to carry deterministic throughput and high-speed circuit or packet data-flow between AIE-MLs and the programmable logic or NoC. Therefore, it is designed to carry the bulk of the data movement to/from the AIE-ML array. The AXI4-Stream switches in the bottom row of AIE-ML tiles interface directly to another row of AXI4-Stream interconnected switches in the AIE-ML array interface. The stream switch has one stream FIFO.

AIE-ML to Programmable Logic Interface

AXI4-Stream switches in the AIE-ML to PL tiles can directly communicate with the programmable logic using the AXI4-Stream interface. There are six streams from AIE-ML to PL and eight streams from PL to each AIE-ML column. From a bandwidth perspective, each AXI4-Stream interface can support the following.

- 24 GB/s from each AIE-ML column to PL
- 32 GB/s from PL to each AIE-ML column

Each stream has a 64-bit interface but can be configured to operate as a 32-bit or 64-bit stream, or two physical streams can be configured to operate as a single 128-bit stream. Streams from the PL support a subset of the AXI4-Stream protocol:

- When TLAST = 0, TKEEP is ignored and assumed to be all ones
- When TLAST = 1, TKEEP is only supported at 32-bit granularity and valid words must be contiguous:
  - 32-bit streams: TKEEP must be 0xF
  - 64-bit streams: TKEEP can be 0x0F or 0xFF
  - 128-bit streams: TKEEP can be 0x000F, 0x00FF, 0x0FFF, 0xFFFF

In the VC2802 device, there are 38 columns of AIE-ML tiles, AIE-ML memory tiles, and AIE-ML interface tiles. However, only 28 array interface tiles are available to the PL interface. Therefore, the aggregate bandwidth for PL interface is approximately:

- 670 GB/s from AI Engine to PL
- 900 GB/s from PL to AI Engine

All bandwidth calculations assume a nominal 1 GHz AI Engine clock for the -1L speed grade devices at VCCINT = 0.70V. The number of array interface tiles available to the PL interface and total bandwidth of the AI Engine to PL interface for other devices and across different speed grades is specified in *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957).

AIE-ML to NoC Interface AXI-Stream

The AIE-ML to NoC interface tile, in addition to the AXI4-Stream interface capability, also contains paths to connect to the horizontal NoC (HNoC). Looking from the AIE-ML, there are four streams from the AIE-ML to the NoC, and four streams from the NoC to the AIE-ML. From a bandwidth perspective each AIE-ML to NoC interface tile can direct traffic between the HNoC and the AXI4-Stream switch.

TIP: *The actual total bandwidth can be limited by the number of horizontal and vertical channels available in the device and also the bandwidth limitation of the NoC.*

Interrupt Handling

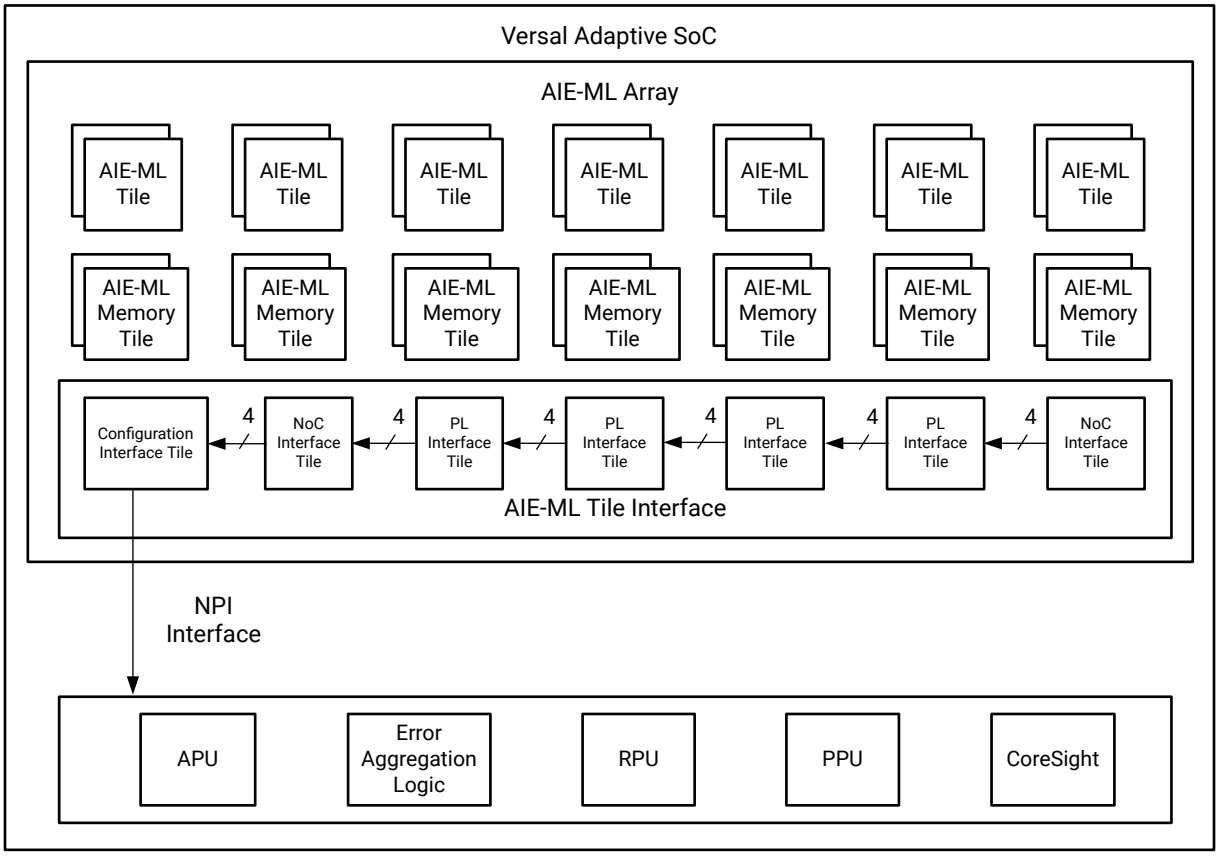
It is possible to setup interrupts to the processor system (PS) and the platform management controller (PMC) triggered by events inside the AIE-ML array. This section gives an introduction to the types of interrupts from the AIE-ML array.

The AIE-ML array generates four interrupts that can be routed from the AIE-ML array to the PMC, application processing unit (APU), and real-time processing unit (RPU). The overall hierarchy for interrupt generation from AIE-ML array is as follows:

- Events get triggered from any of the AIE-ML tiles or AIE-ML interface tiles.
- Each column has first-level interrupt handlers that can capture the trigger/event generated and forward it to the second-level interrupt handler. Second-level interrupt handlers are only available in NoC interface tiles.
- A second-level interrupt handler can drive any one of the four interrupt lines in a AIE-ML array interface.
- These four interrupt lines are eventually connected to the AIE-ML configuration interface tile.

The following figure is a high-level block diagram showing the connections of the NPI interrupts from the AIE-ML array to other blocks in the Versal device. The diagram does not show the actual layout/placement of the array interface tiles and the AIE-ML tiles.

Figure 31: **Connecting Interrupts from the AIE-ML Array to Other Functional Blocks**



In the previous figure, the four interrupts are generated from a NoC interface tile. They pass through the PL interface tile and reach a configuration interface tile. Internal errors (such as PLL lock loss) are then ORed with the four incoming interrupts and the resulting four interrupts are connected directly to the NPI interrupt signals on the NPI interface, which is a 32-bit wide memory-mapped AXI4 bus.

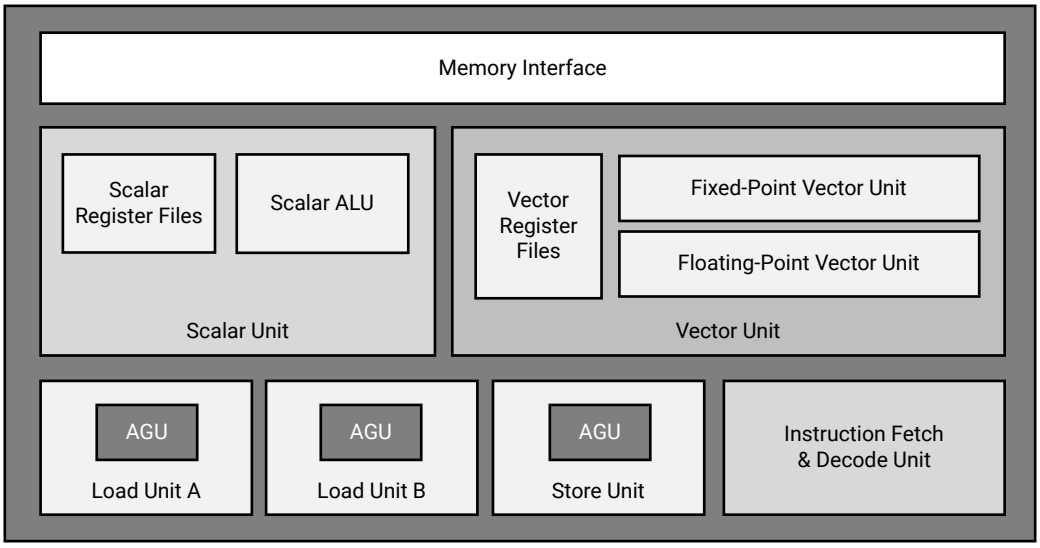
At the device level, the four NPI interrupts are assigned 4 to 7. There are three groups of NPI registers (IMR0...IMR3, IER0...IER3, and IDR0...IDR3). Each of the pairs (IMR, IER, and IDR) can be used to configure the four NPI interrupts. IMR registers are read only, and IER and IDR registers are write only. Only the registers corresponding to NPI interrupt 4 can be programmed. For NPI interrupts 5, 6, and 7, the three sets of registers have no effect and the three interrupts cannot be masked by programming the NPI register.

AIE-ML Architecture

Functional Overview

The AIE-ML is a highly-optimized processor featuring single-instruction multiple-data (SIMD) and very-long instruction word (VLIW) processor that supports both fixed-point and floating-point precision. As shown in the following figure, the AIE-ML has a memory interface, a scalar unit, a vector unit, two load units, one store unit, and an instruction fetch and decode unit.

Figure 32: **AIE-ML**



The features of the AIE-ML include:

- Instruction-based VLIW SIMD processor
- 32-bit scalar RISC processor
  - Scalar register files and special registers
  - 32 x 32-bit multiplier (signed and unsigned)
  - 32-bit add/subtract
  - ALU operations like shifts, compares, and logical operations

    - No floating point unit: Supported through emulation
- Three address generator units (AGU)
    - Two 256-bit load and one 256-bit store units with aligned addresses
    - Supports 2D/3D addressing modes for ML functionality
    - On-the-fly decompression during loading of sparse weights. See Sparsity for more information.
    - One AGU dedicated for the store unit
- Vector fixed-point/integer unit
    - Supports FFT processing and sparsity for ML inference applications, including cint32 x cint16 multiplication (data in cint32 and twiddle factor in cint16), control support for complex and conjugation, new permute mode, and shuffle mode. See Sparsity for more information.
    - Accommodate multiple precision for complex and real operand. See Table 7: Supported Precision Width of the Vector Data Path for more information.

Table 7: **Supported Precision Width of the Vector Data Path**

Precision 1	Precision 2	Number of Accumulator Lanes	Bits per Accumulator Lane	Number of MACs
int 8	int 8	32	32	256
int 16	int 8	32	32	128
int 16	int 8	16	64	128
int 16	int 16	32	32	64
int 16	int 16	16	64	64
int 32[1]	int 16	16	64	32
cint 16	cint 16	8	64	16
cint 32	cint 16	8	64	8
bfloat 16[3]	bfloat 16	16	SPFP 32[2]	128

Notes:

1. int32 x int32 can be emulated. The operation should have half the performance of int32 x int16 and there should be 16 multiplications per cycle.
2. Single precision floating point (SPFP) per the IEEE standard.
3. float32 x float32 can be emulated. Emulation deviates from the IEEE-754 standard. See Answer Record 34376 for more information.
4. The number of MACs is for matrix multiply operations. The throughput of element wise multiplications matches the number of accumulator lanes.
5. int8 x int4 can be emulated. The operation has the same performance as int8 x int8.

    - The multiplier|multiplicand can be signed or unsigned. The accumulator is always signed.
    - The accumulation can be performed in two operation modes, with either 32 lanes of 32 bits or 16 lanes of 64 bits.
    - The total number of multipliers and the number of accumulation lanes determine the depth of the post-adding.

    - In terms of component use, consider the first row in Table 7: Supported Precision Width of the Vector Data Path. Depending on whether or not sparsity is used, the multiplier inputs can be 1024 x 512 or 512 x 512 bits. The number of int8 multipliers is 256. The accumulation is on 32 lanes of 32 bits. See Sparsity for more information.
- Single-precision floating-point (SPFP) vector unit:
    - Supports 128 bfloat 16 MAC operations with FP32 accumulation by reusing the integer multipliers and post adders along with additional blocks for floating point exponent compute and mantissa shifting and normalization.
    - Concurrent operation of multiple vector lanes.
    - Supports multiplying bfloat16 numbers (16-bit vector lanes) and accumulating in SPFP (32-bit register lanes). Only 16 accumulator lanes are used in this mode.
- The vector unit also supports integer arithmetic on 8, 16, and 32 bit operands, and bitwise AND, OR, and NEGATE.
- Balanced pipeline:
    - Different pipeline on each functional unit (eight stages maximum).
    - Load and store units manage the 5-cycle latency of data memory.
- Three data memory ports:
    - Two load ports and one store port
    - Each port operates in 256-bit/128-bit vector register mode. Scalar accesses (32-bit/16-bit/8-bit) are supported by only one load port and one store port. The 8-bit and 16-bit stores are implemented as read-modify-write instructions.
    - Concurrent operation of all three ports
    - A bank conflict on any port stalls the entire data path
- Very-long instruction word (VLIW) function:
    - Concurrent issuing of operation to all functional units
    - Support for multiple instruction formats and variable length instructions
    - Up to six operations can be issued in parallel using one VLIW word
- Direct stream interface:
    - One input stream and one output stream
    - Each stream is 32-bits wide
    - Vertical in addition to horizontal cascade stream in and stream out in 512 bits
- Interface to the following modules:
    - Lock module
    - Stall module
    - Debug and trace module
- Event interface is a 16-bit wide output interface from the AIE-ML.
- Processor bus interface:
    - The AIE-ML architecture is a processor that allows the AIE-ML to perform direct read/write access to local tile memory mapped registers.

Register Files

The AIE-ML has several types of registers. Some of the registers are used in different functional units. This section describes the various types of registers.

Scalar Registers

Scalar registers include configuration registers. See the following table for register descriptions.

Table 8: **Scalar Registers**

Syntax	Number of Bits	Description
r0..r31	32 bits	General-purpose registers
m0..m7	20 bits	Modifier registers
p0..p7	20 bits	Pointer registers

Special Registers

Table 9: **Special Registers**

Syntax	Number of Bits	Description
dn0..dn7	20 bits	AGU dimension size register
dj0..dj7	20 bits	AGU dimension stride (jump) register
dc0..dc7	20 bits	AGU dimension count register
s0..s3	6 bits	Shift control
sp	20 bits	Stack pointer
lr	20 bits	Link register
pc	20 bits	Program counter
fc	20 bits	Fetch counter
	32 bits	Status register[1]
	32 bits	Mode control register[1]
ls	20 bits	Loop start
le	20 bits	Loop end
lc	32 bits	Loop count
lci	32 bits	Loop count (PCU)

Notes:

1. The status and control registers are each separate registers of a small number of bits each. Only through the debug interface are the various individual registers accessed together as one 32-bit wide SR and CR register.

Vector Registers

Vector registers are wide to allow SIMD instructions and to be used as operand storage. These registers are prefixed with a *W*. There are 24 x 256-bit registers: wln and whn, n-0..11. Two W registers can be grouped to form a 512-bit register prefixed with an X. Two X registers then can be grouped to form a 1,024-bit register with the prefix Y and Y2 ... Y5 are aliased for X4 ... X11.

Table 10: **AIE-ML Vector Registers**

256-bit	512-bit	1024-bit
wl0	x0	
wh0		
wl1	x1	
wh1		
wl2	x2	
wh2		
wl3	x3	
wh3		
wl4	x4	y2
wh4		
wl5	x5	
wh5		
wl6	x6	y3
wh6		
wl7	x7	
wh7		
wl8	x8	y4
wh8		
wl9	x9	
wh9		
wl10	x10	y5
wh10		
wl11	x11	
wh11		

Mask Registers

In addition to the vector registers, there are 4 x 128-bit mask registers (Q0 to Q3) used for sparsity. See Sparsity for more information.

Accumulator Registers

Accumulator registers are used to store the results of the vector data path. 256 bit wide, they can be viewed as eight lanes of 32-bit data or four lanes of 64-bit data. The accumulator registers are prefixed with `am`. Two of them are aliased to form a 512-bit register prefixed with `bm`, and two `bm` can be aliased to form a 1024-bit register prefixed with `cm`.

Table 11: **Accumulator Registers**

256-bit	512-bit	1024-bit
amll0	bml0	cm0
amlh0		
amhl1	bmh0	
amhh1		
...	...	...
...		
...	...	
...		
amll8	bml8	cm8
amlh8		
amhl8	bmh8	
amhh8		

Instruction Fetch and Decode Unit

The instruction fetch and decode unit sends out the current program counter (PC) register value as an address to the program memory. The program memory returns the fetched 128-bit wide instruction value. The instruction value is then decoded, and all control signals are forwarded to the functional units of the AIE-ML. The program memory size on the AIE-ML is 16 KB, which allows storing 1024 instructions of 128-bit each.

The AIE-ML instruction size ranges from 16 to 128 bits and support multiple instruction formats and variable length instructions to reduce the program memory size. In most cases, the full 128 bits are needed when using all VLIW slots. However, for many instructions in the outer loops, main program, control code, or occasionally the pre- and post-ambles of the inner loop, the shorter format instructions are sufficient, and can be used to store the more compressed instructions with a small instruction buffer.

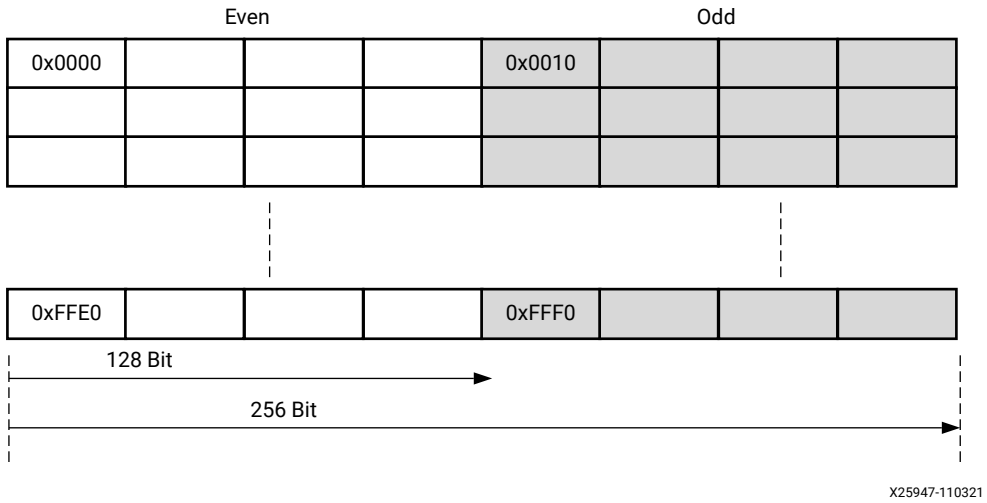
Load and Store Unit

The AIE-ML has two load units and one store unit for accessing data memory. Data is loaded or stored in data memory.

Each of the load or store units has an address generation unit (AGU). AGUA and AGUB are the load units and the store unit is AGUS. Each AGU has a 20-bit input from the P-register file and a 20-bit input from the M-register file (refer to the pointer registers and the modifier registers in Register Files). The AGU has a one cycle latency.

An individual data memory block is 64 KB. The AIE-ML accesses four 64 KB data memory blocks to create a 256 KB unit. These four memory blocks are located on each side of the AIE-ML and are divided and interleaved as odd and even banks (see the following figure).

Figure 33: **Interleaving in Data Memory (64 KB per Block)**



In a logical representation the 256 KB memory can be viewed as one contiguous 256 KB block or four 64 KB blocks, and each block can be divided into odd and even banks. The memory can also be viewed as eight 32 KB banks (four odd and four even). The AGU generates addresses for data memory access that span from `0x0000` to `0x3FFFF` (256 KB).

One of the load units supports online decompression of activations/weights.

Scalar Unit

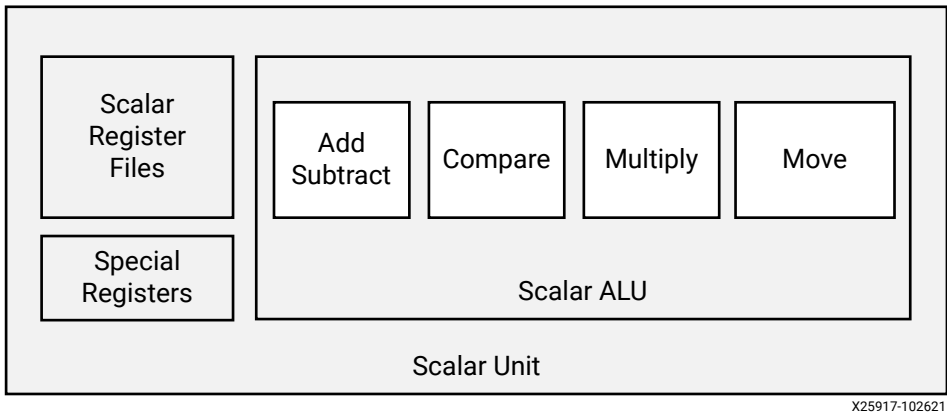
The following figure shows a block diagram of the scalar unit, including the scalar register files and scalar functional units.

The scalar unit contains the following functional blocks.

- Register files and special registers
- Arithmetic and logical unit (ALU)

Integer add, subtract, compare, and shift functions are one-cycle operations. The integer multiplication operation has a two-cycle latency.

Figure 34: **AIE-ML Scalar Unit**



Arithmetic Logic Unit and Scalar Functions

The arithmetic logic unit (ALU) in the AIE-ML manages the following operations. In all cases the issue rate is one instruction per cycle.

- Integer addition and subtraction: 32 bits. The operation has a one cycle latency.
- Bit-wise logical operation on 32-bit integer numbers (BAND, BOR, BXOR). The operation has a one cycle latency.
- Integer multiplication: 32 x 32 bit with output result of 32 bits stored in the R register file. The operation has a two cycle latency.
- Shift operation: Both left and right shift are supported. A positive shift amount is used for left shift and a negative shift amount is used for right shift. The shift amount is passed through a general purpose register. A one bit operand to the shift operation indicates whether a positive or negative shift is required. The operation has a one-cycle latency.

There is no floating point unit in the scalar unit. The floating point operations are supported through emulation. In general, it is preferred to perform add and multiply in the vector unit.

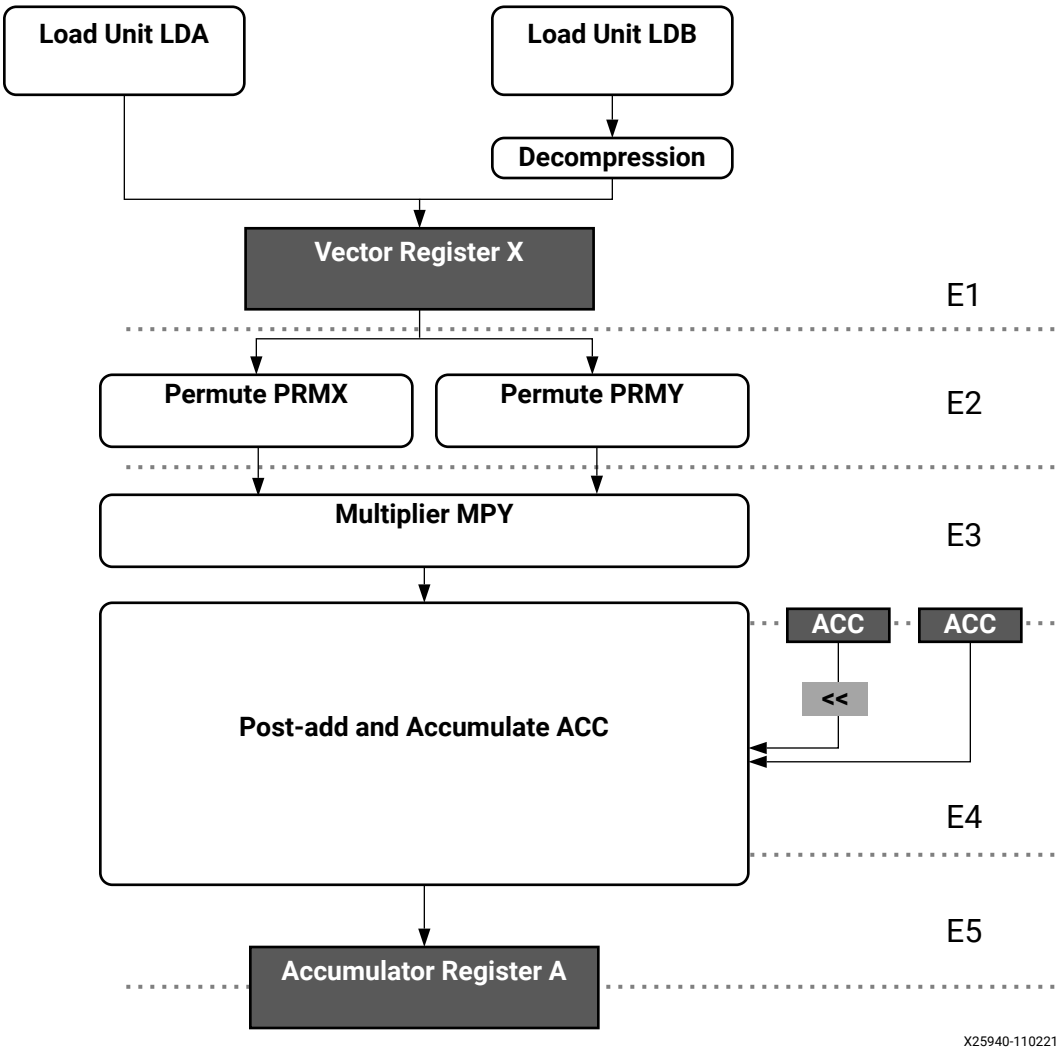
Vector Unit

Fixed-Point Vector Unit

AIE-ML Fixed-Point Vector Unit

The following is a block diagram of the fixed-point vector data path. The datapath is split into five pipeline stages.

Figure 35: **Pipeline Diagram of AIE-ML Fixed-point Vector Unit Multiplication Paths**



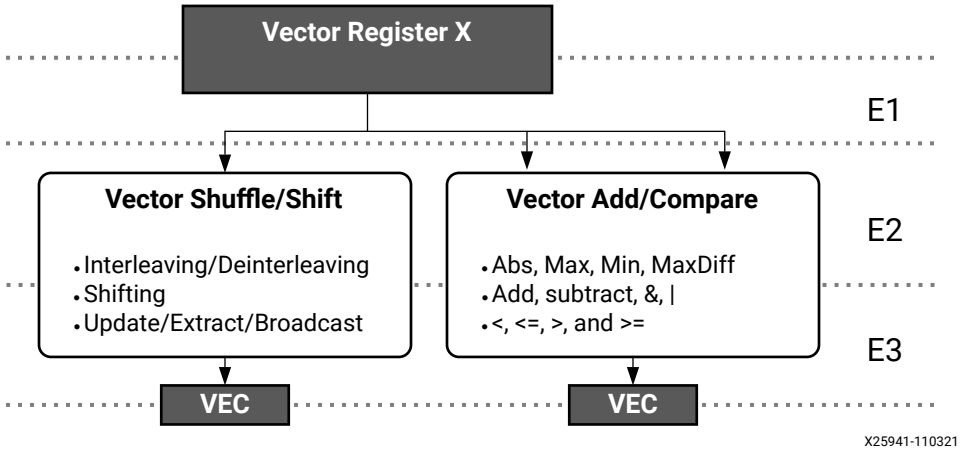
The features of the units in the datapath are as follow:

- The multiplier unit is fed by the output of the permute blocks. The vector adder is in a separate functional unit together with a vector shuffle and shift datapath.

- There are two permute units PRMX and PRMY that handle a set of permutes of X vector registers.
- In addition to the permute and multiplier, there are two additional vector units: shuffle/shift and add/compare. The input comes directly from two vector registers and the results are stored back in the vector registers. The supported bit-width modes are (both signed and unsigned):
  - 16 lanes of 32-bit
  - 32 lanes of 16-bit
  - 64 lanes of 8-bit

  The unit supports lane-by-lane control whether addition or subtraction is performed.

Figure 36: **Pipeline Diagram of AIE-ML Fixed-point Vector Unit Shuffle/Shift and Adder Paths**



The previous image shows that in addition to the vector adder, there is a vector shuffle and shift datapath. The vector shift unit takes one or two 512-bit vector registers as an input and produces one 512-bit output vector. It supports the following modes:

- Standard right shift with 8-bit granularity
- Shift and push in scalar value either at the left or right-hand side. An 8, 16, or 32-bit lane can be shifted into the LSB lane of a 512-bit vector register, and all existing values are shifted one lane up. The value of MSB lane is dropped.

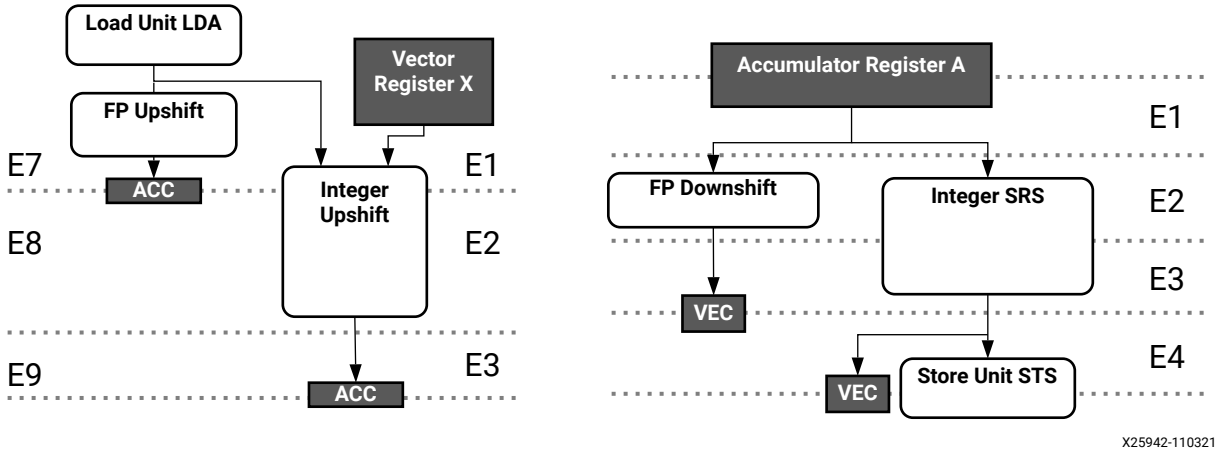
The shuffle unit allows different modes to transform the input vectors. It supports the following features:

- Interleaving and deinterleaving of values at 8-bit, 16-bit, and 32-bit
- Extraction of upper and lower half of the transformed input.

Fixed-Point SRS and UPS Conversions

A block diagram of the units is shown in the following figure.

Figure 37: **Fixed-point SRS and UPS Datapath**



The SRS unit reads an accumulator register, performs the conversion, and restores the result either back to the vector register or directly to memory. The UPS unit reads a vector register directly from memory or a register and stores the result into an accumulator register. The supported modes include:

- 32 lanes of 8-bit to/from 32-bit conversion
- 32 lanes of 16-bit to/from 32-bit conversion
- 16 lanes of 16-bit to/from 64-bit conversion
- 16 lanes of 32-bit to/from 64-bit conversion

A floating-point conversion mode is also supported. It converts bfloat16 to single precision or vice versa. The modes supported are:

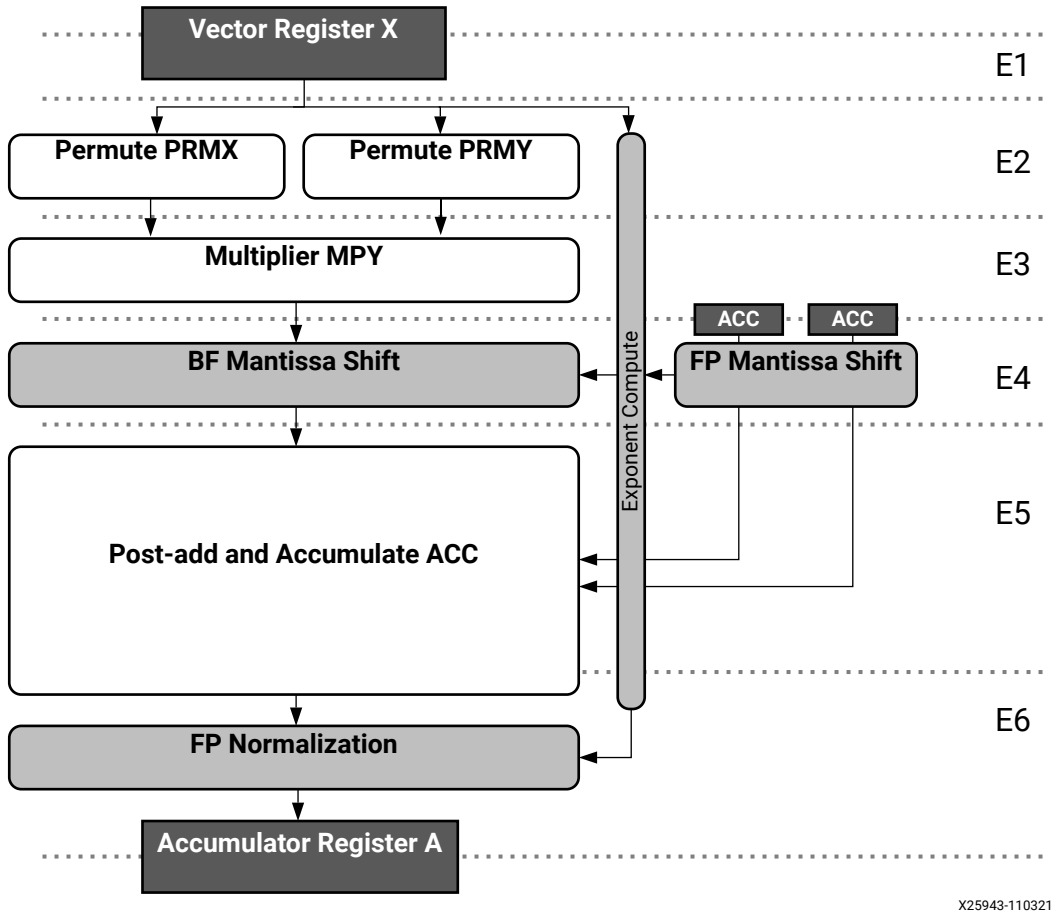
- 16 lanes of fp32 accumulators to bfloat16 vector registers
- 16 lanes of bfloat16 vector registers to fp32 accumulators

In addition (not shown in the figure), the unit also supports floating-point to integer conversion: 16 lanes of bfloat16 vector registers to 32-bit signed registers.

Floating-Point Vector Unit

The following figure shows the block diagram of the bfloat16 (BF) vector datapath. It shares the 8 x 8 multipliers with half of the integer datapath along with additional blocks for floating point exponent compute and mantissa shifting and normalization.

Figure 38: **Bfloat16 Vector Datapath**



To reduce the accumulation feedback loop, multiple accumulator registers are used to allow back-to-back floating-point MAC instructions.

The BF and FP mantissa shift unit shifts down each of the 128 multiplier lanes and the 2 x 16 accumulator lanes. The accumulator unit supports addition/subtract/negate of accumulator registers in a single-precision FP32 format. All floating-point additions are done in one go, by aligning all mantissas to the one with the largest exponent and with 23 bits of fractional bits. The FP normalization unit handles the cases where the mantissa coming from the post-adder is negative and if the mantissa is outside the acceptable range.

The AIE-ML supports several vector element-wise functions for the bfloat16 format. These functions include a vector comparison, minimum, and maximum. They operate in an element-wise fashion comparing two vectors. The separate fixed-point vector add/compare unit is extended to handle the floating-point elementary function.

The floating-point unit can issue events that correspond to standard floating-point exceptions and the status registers keep track of the events. There are eight exception bits per floating-point functional unit. The exceptions are (from bit 0 to 7): zero, infinity, tiny (underflow), huge (overflow), inexact, huge integer, and divide-by-zero. Of the eight exceptions, tiny, huge, invalid, and divide-by-zero can be converted into an event that can be broadcast to the AIE-ML array interface and then sent to the PS/PMC as an interrupt.

Denormalized numbers are not supported by the AIE-ML floating-point data path.

Register Move Functionality

The register move capabilities of the AIE-ML are covered in this section (refer to the Register Files section for a description of the naming of register types.

- Scalar to scalar:
  - Move scalar values between R, M, P, and special registers.
  - Move immediate values to R, M, P, and special registers.
  - Move a scalar value to/from an AXI4-Stream.
- Vector to vector: Move one 128-bit V-register to an arbitrary V-register in one cycle. It also applies to the 256-bit W-register and the 512-bit X-register. However, vector sizes must be the same in all cases.
- Accumulator to accumulator: Move one 512-bit accumulator (AM) register to another AM-register in one cycle. There is also register BM to BM accumulator register move (1024 bits).
- Vector to accumulator: there are three possibilities:
  - Up shift path takes 16 or 32-bit vector values and writes into an accumulator.
  - Use the normal multiplication datapath and multiply each value by a constant value of 1.
  - Move between BM and X registers.
- Accumulator to vector: Shift-round saturate datapath moves the accumulator to a vector register. There is also a direct register move from accumulator to vector register.
- Accumulator to cascade stream and cascade to accumulator: Cascade stream connects the AIE-MLs in the array in a chain and allows the AIE-MLs to transfer an accumulator register (512-bit) from one to the next. A small two-deep 512-bit wide FIFO on both the input and output streams allows storing up to four values in the FIFOs between the AIE-MLs.
- Scalar to vector: Moves a scalar value from an R-register to a vector register. Different from AIE where most operations were on the 128-bit granularity except for shift element operation, only operations on 512-bit registers are allowed in AIE-MLs.

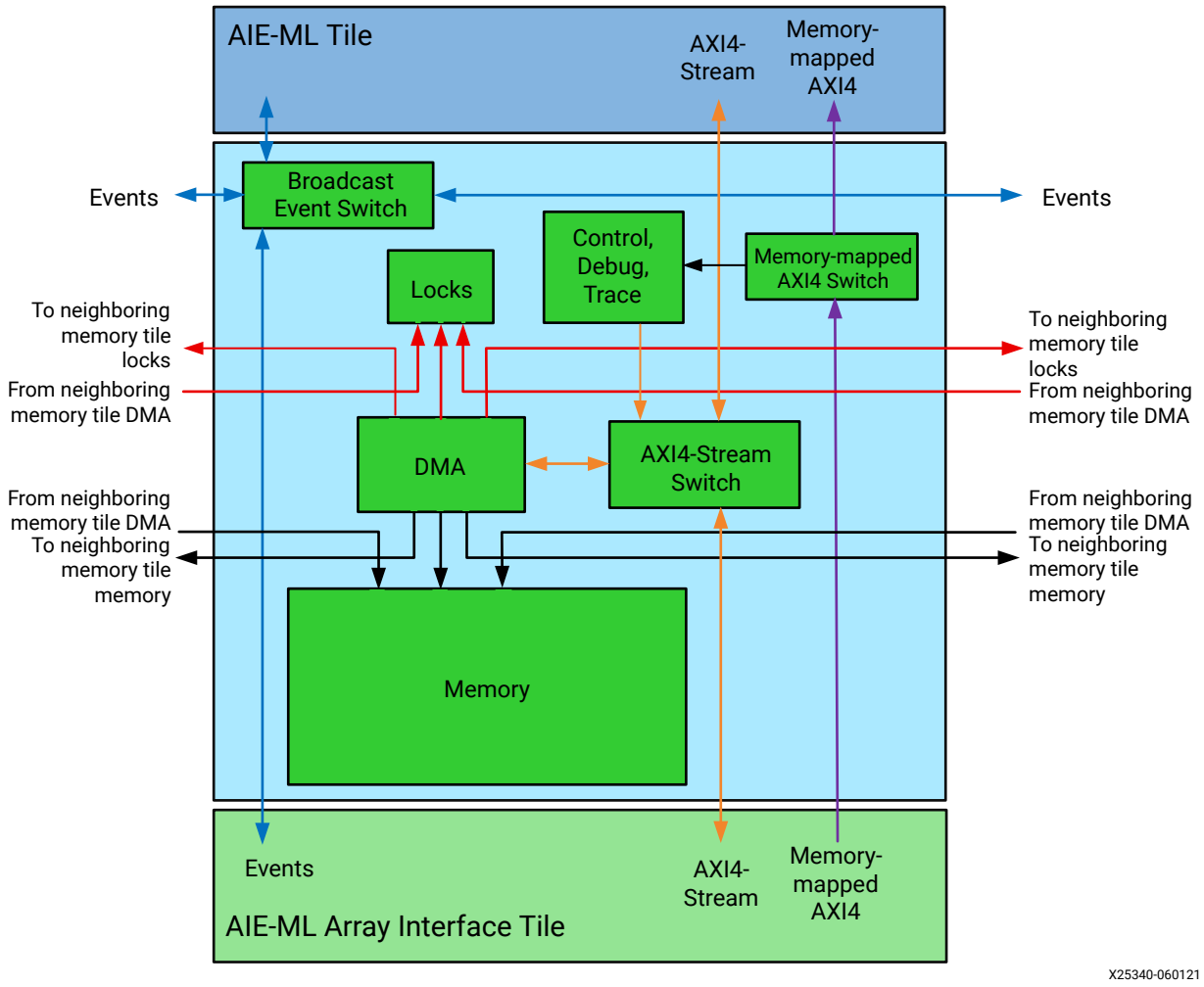
- Vector to scalar: Extracts an arbitrary 8, 16, or 32-bit value from a 512-bit vector register and writes results into a scalar R-register.

AIE-ML Memory Tile Architecture

AIE-ML Memory Tile Overview and Features

The AIE-ML memory tile is introduced in the AIE-ML architecture to significantly increase the on-chip memory inside the AIE-ML array. The memory tile reduces the utilization of PL resources (LUTs, block RAMs and UltraRAMs) in ML applications. It is similar to the AIE-ML tile but without the AIE-ML processor and program memory. The AIE-ML memory tile contains high-density (512 KB) and high bandwidth memory, and an integrated DMA to access local memory and neighboring memories. The AIE-ML memory tile only has vertical streaming interfaces (no cascade or horizontal). A subset of DMA channels can directly access memory in the nearest neighboring memory tiles to the East and West. The following figure shows the AIE-ML memory tile architecture.

Figure 39: **AIE-ML Memory Tile Architecture**



The memory tile has the following functional blocks. They are either the same or similar to the equivalent blocks in the AIE-ML tile:

- Memory
- DMA
- Locks
- AXI4-Stream switch
- Memory-mapped AXI4 switch
- Control, debug, and trace
- Events and event broadcast

The following is a list of AIE-ML memory tile features:

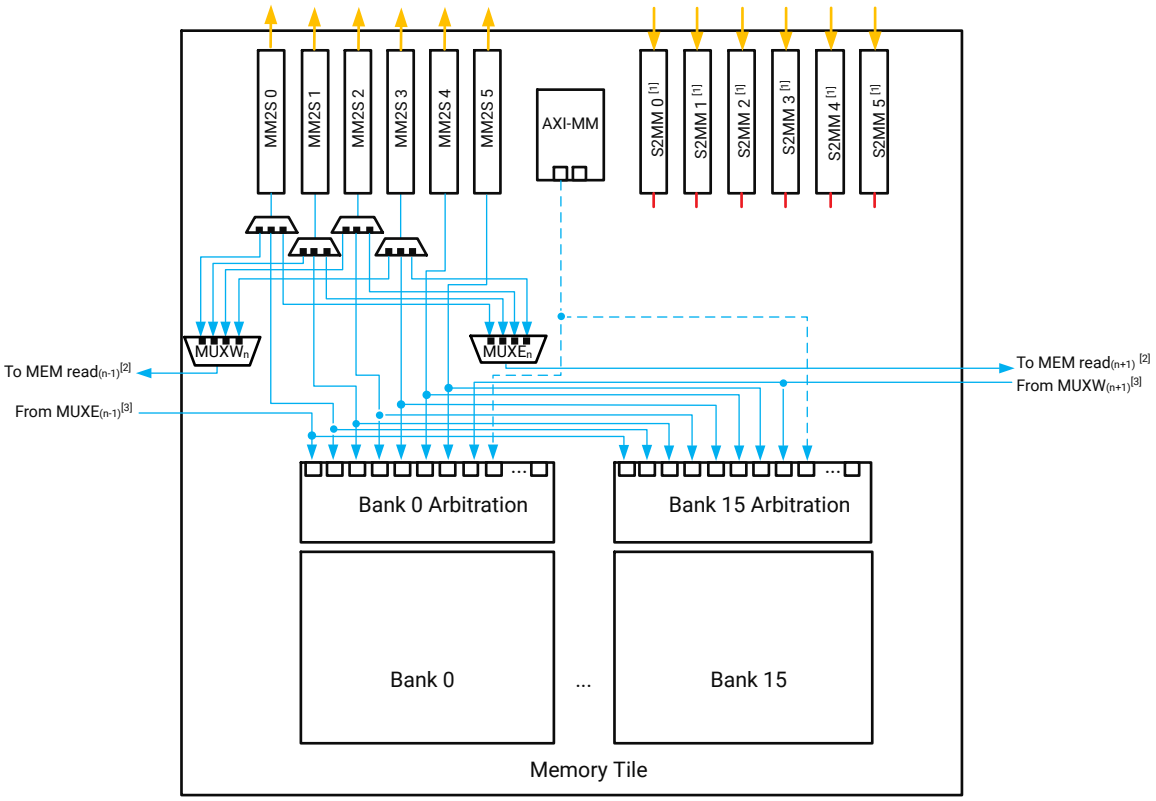
- Memory
  - 512 KB memory arranged into 16 banks (each 128-bit wide and 2k words deep), ECC protected
  - The memory banks in the AIE-ML memory tile initializes to zero at boot and reset
  - Supports up to 30 GB/s read and 30 GB/s write in parallel per memory tile
- DMA
  - Memory to stream DMA (MM2S) with six channels
    - 6 x 32-bit stream interfaces
    - 6 x 128-bit memory interfaces
    - 5D tensor address generation (including iteration-offset)
    - Support inserting zero padding into stream data and compression
    - Access memory and locks in east/west neighboring tiles (channels 0–3)
    - Support task queue and task-complete-tokens; queue depth is four tasks per channel (see Task-Completion-Tokens for more information)
  - Stream to memory DMA (S2MM) with six channels
    - 6x32-bit stream interfaces
    - 6x128-bit memory interfaces
    - 5D tensor address generation (including iteration-offset)
    - Support out-of-order packet transfer, finish-on-TLAST, and decompression
    - Access memory and locks in east/west neighboring tiles (channel 0-3)
    - Support task queue and task-complete-tokens; queue depth is four tasks per channel (see Task-Completion-Tokens for more information)
  - Buffer descriptors (BD)
    - 48 shared BDs
    - Each channel can access 24 BDs and each BD can be accessed by six channels
  - Stream Switch
    - Share the same design as AIE-ML tile. 17 master and 18 slave ports
    - North and South ports but no east and west streams
    - Trace and control ports
  - Lock Module
    - Accessible from neighboring AIE-ML memory tile DMA channels; there are 64 semaphore locks and each lock state is 6-bit unsigned
  - Additional control and status registers
    - Events, event actions, event broadcast, combo events
    - Task-complete-tokens logic (see Task-Completion-Tokens for more information)
  - Configuration/debug interconnect (memory-mapped AXI4)
    - 1 MB address space per tile
    - Write bandwidth improvement and stream control-packet support
  - Debug and Trace
    - Similar to that in AIE-ML tile

- Event trace stream; 4x performance counters and 64-bit tile timer

AIE-ML Memory Tile Memory

Each AIE-ML memory tile has 512 KB of memory as 16 banks of 32 KB. Each bank is 128 bits wide and 2k words deep. Each bank allows one read or one write every cycle and can be accessed by nine read interfaces and nine write interfaces. Each interface is 128-bit. The following block diagrams show the memory tile read and write interfaces.

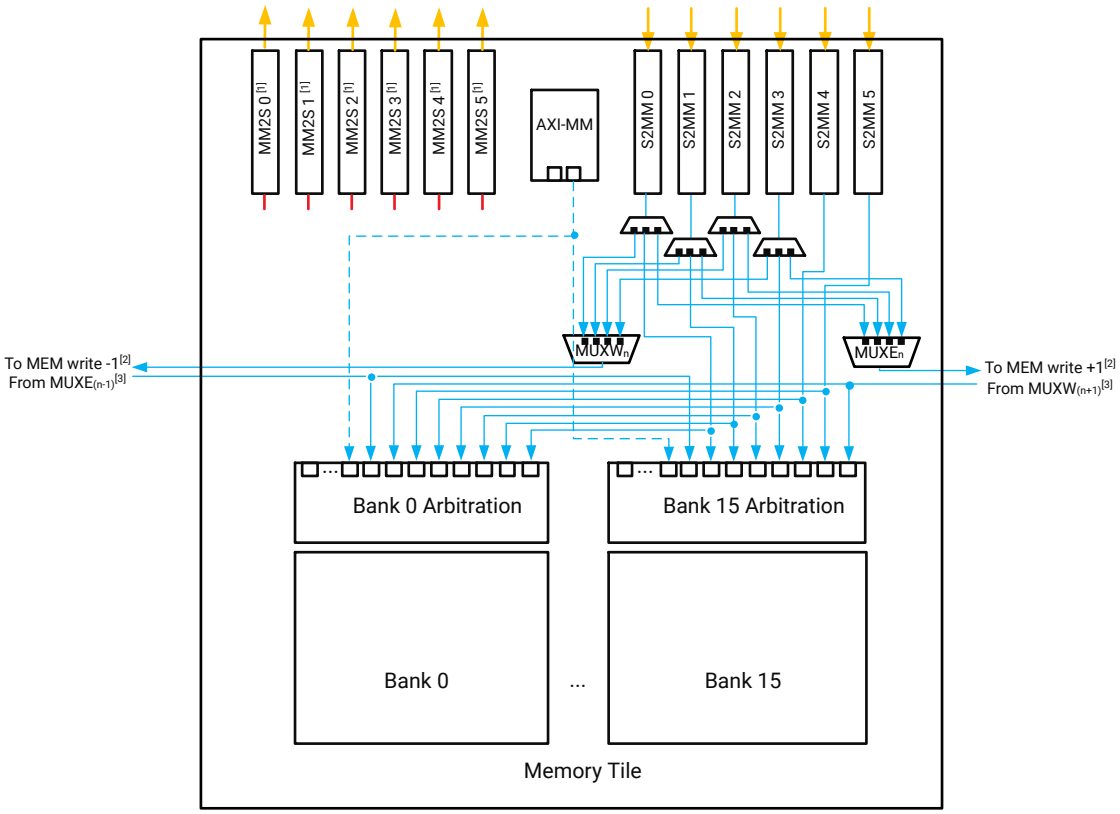
Figure 40: **Memory Tile Memory Read Interfaces**



[1] The S2MM interfaces exist for the memory tile, but are not used in the memory read operation.
[2] MEM read$_{(n-1)}$ and MEM read$_{(n+1)}$ correspond to the neighboring tile memory banks on the east and west.
[3] MUXE$_{(n-1)}$ refers to the multiplexer of the tile on the east and MUXW$_{(n+1)}$ refers to the multiplexer of the tile on the west.

X25865-101921

Figure 41: **Memory Tile Memory Write Interfaces**



[1] The MM2S interfaces exist for the memory tile, but are not used in the memory write operation.
[2] MEM write -1 and MEM write +1 correspond to the neighboring tiles on the east and west.
[3] $MUXE_{(n-1)}$ refers to the multiplexer of the tile on the east and $MUXW_{(n+1)}$ refers to the multiplexer of the tile on the west.

X25880-101921

The interfaces also have the following features:

- **Read Interfaces:** Memory-mapped AXI4 write including control packets for memory tile –1 and memory tile +1, and six MM2S channels [0-5]
- **Write Interfaces:** Memory-mapped AXI4 read including control packets for memory tile –1 and memory tile +1, and six S2MM channels [0-5]

DMA S2MM channels (0 – 3) and MM2S channels (0 – 3) can access the local memory banks and the memory banks of the memory tile to the east and the west.

The memory in memory tile supports bank interleaving. Interleaving is done at a 128-bit granularity, such that sequential 128-bit accesses map to different banks and wrap around after the 16 banks, every 256B.

The memory tile banks have ECC protection and ECC scrubbing similar to the AIE-ML data memory.

AIE-ML Memory Tile DMA

The list of features in the AIE-ML memory tile DMA is covered in the AIE-ML Memory Tile Overview and Features section. The memory tile DMA is similar to the AI Engine (AIE) tile DMA with a few enhancements:

- Supports 5D tensor address generation (including iteration-offset)
- Allows out-of-order buffer descriptor (BD) processing based on incoming packet header information
- Supports compression and decompression

The memory tile DMA has 12 independent channels, six S2MM and six MM2S. Each channel has an input task queue. It can load a BD, generate address, access memory over a shared interface, and read or write to and from its stream port. Each channel can also trigger the issuing of a task-complete-token upon completing a task. The AIE-ML memory tile DMA supports address generation as described in the Data Movement section. The memory tile DMA supports up to four dimensions (K=4).

Of the six S2MM and MM2S channels, DMA S2MM channels 0-3 and MM2S channels 0-3 can access the memory banks in the tile to the west and east, in addition to the local memory banks. These same channels can also access lock modules in tile to the east and west. Both MM2S and S2MM channels 4-5 can only access local memory banks and local lock modules.

All 12 channels use the same address scheme and lock indexes, as shown in the following table.

Table 12: **Address and Lock Ranges for Memory Tile DMAs**

	Address Ranges	Lock Indexes	Description
West	0x0_0000 – 0x7_FFFF	0–63	Channels 0–3 only
Local	0x8_0000 – 0xF_FFFF	64–127	
East	0x10_0000 – 0x17_FFFF	128–191	Channels 0–3 only

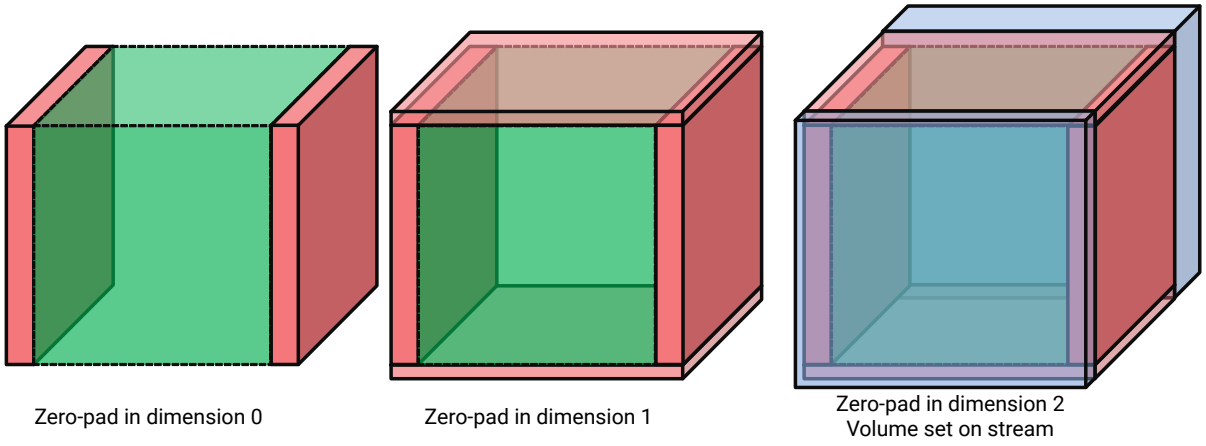
With this addressing scheme, it is possible to configure the hardware where the address and lock requests could be out of range for a specific DMA channel. This condition can result in the DMA channel stalling requiring a channel reset to proceed.

The memory tile MM2S channels support zero-padding insertion. This feature satisfies two application requirements:

- Algorithmic padding: To recreate surrounding data on the edge of valid data.
- Granularity padding: An optimized kernel can operate on 16 channels while a layer can have 24 channels, the MM2S channel pads the channel dimension up to 32 channels.

The zero-padding insertion is linked to 4D address generation. For the lower three dimensions, there is a field for padding before and after that dimension. The following figure illustrates zero-padding in three dimensions. The padding on a dimension is added on top of the wrap for that dimension.

Figure 42: **Zero Padding in 3D**



AIE-ML Memory Tile Locks Module and Stream Switch

The memory tile semaphore locks are identical to those in the AIE-ML tile. The memory tile supports up to 64 semaphore locks each identified by a 6-bit unsigned lock state. All the locks are accessible from local DMA channels as well as S2MM and MM2S channels from both east and west neighboring memory tiles. Locks are also accessible through memory-mapped AXI4.

The memory tile stream switch is similar to the tile stream switch with 17 master ports and 18 slave ports. The differences from the tile stream switch are:

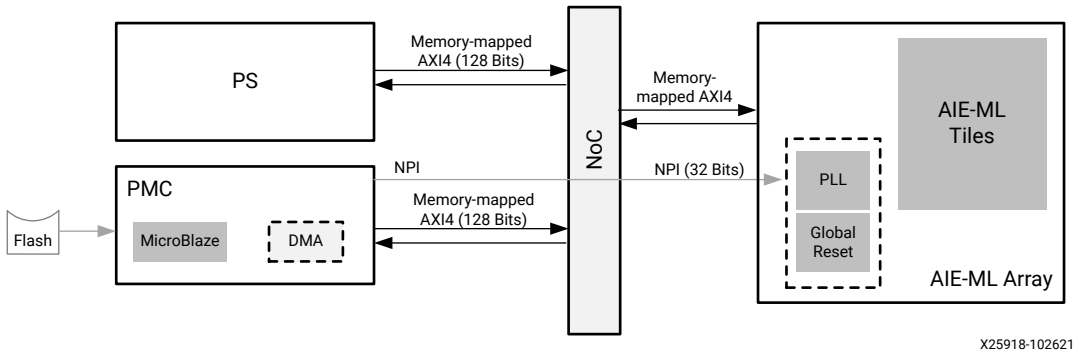
- No east or west stream ports
- No stream FIFO

AIE-ML Configuration and Boot

AIE-ML Array Configuration

There are two top-level scenarios in the AIE-ML array configuration: AIE-ML array configuration from power-up and AIE-ML array partial reconfiguration. The following figure shows a high-level view of the AIE-ML array and configuration interface along with the registers to the PS and the platform management controller (PMC) through the NoC.

Figure 43: **AIE-ML Array Configuration using NoC and NPI**



Any memory-mapped AXI4 master can configure any memory-mapped AXI4 register in the AIE-ML array using the NoC (for example, the PS and PMC). The global registers (including PLL configuration, global reset, and security bits) in the array configuration interface tile can be programmed using the NPI interface because the global registers are mapped onto the NPI address space.

AIE-ML Boot Sequence

This section describes the steps involved in the boot process for the AIE-ML array.

- The column clock enable value is disabled by default.

- Memory zeroization hardware logic is added that applies to each of the tile program memory, tile data memory and memory tile data memory. For each of the memories there is a 1-bit memory-mapped AXI4 register that is set to `1` when zeroization starts. When the process is completed, the internal hardware sets this bit to `0`.

1. Power-on and power-on-reset (POR) deassertion: Power is turned on for all modules related to the AIE-ML array, including the PLL. After power-on, the PLL runs at a default speed. The platform management controller (PMC) and NoC need to be up and running before the AIE-ML boot sequence is initiated. After the array power is turned on, the PMC can deassert a POR signal in the AIE-ML array.
2. AIE-ML array configuration using NPI: After power-on, the PMC uses the NPI interface to program the different global registers in the AIE-ML array (for example, the PLL configuration registers). The AIE-ML configuration image that is required over the NPI for AIE-ML array initialization comes from a flash device.
3. Enable PLL: Once the PLL registers are configured (after POR), the PLL-enable bit can be enabled to turn on the PLL. The PLL then settles on the programmed frequency and asserts the **LOCK** signal. The source of the PLL input (ref_clk) is from hsm_ref_clk and is generated in the control interfaces and processing system (CIPS).
    - The generation and distribution of the clock is described in the *PMC and PS Clocks* chapter of the *Versal Adaptive SoC Technical Reference Manual* (AM011).
4. Column clock and column reset assertion/de-assertion: Once the PLL is locked, all column clocks are enabled by writing a `1` to a memory-mapped AXI4 register bit. All column resets are then asserted writing a `1` to a memory-mapped AXI4 register bit. After waiting for a number of cycles, all column resets are de-asserted by writing a `0` to the same register bit.
5. The array is partitioned into one or more independent partitions, with an integer number of AI Engine (AIE) columns per partition. Isolation is enabled by default in all tiles, therefore must be disabled on the internal edges of each partition.
6. AIE-ML array programming: The AIE-ML array interface needs to be configured over the memory-mapped AXI4 from the NoC interface. This includes all program memories, AXI4 stream switches, DMAs, event, and trace configuration registers.

AIE-ML Array Reconfiguration

The AIE-ML configuration process writes a programmable device image (PDI) produced by the bootgen tool into AIE-ML configuration registers. The AIE-ML configuration is done over memory-mapped AXI4 via the NoC. Any master on the NoC can configure the AIE-ML array. For more information on generating a PDI with the bootgen tool, refer to *Bootgen User Guide* (UG1283).

The AIE-ML array can be reconfigured at any time. The application drives the reconfiguration. Safe reconfiguration requires:

- Ensuring that reconfiguration is not occurring during ongoing traffic.
- Disabling the AIE-ML to PL interface prior to reconfiguration.
- Draining all data in the sub-region before it is reconfigured to prevent side-effects from remnant data from a previous configuration.

Complete reconfiguration is currently used to reconfigure the AIE-ML array. The global reset is asserted for the AIE-ML array and the entire array is reconfigured by downloading a new configuration image.

The PMC and PS are responsible for initializing the AIE-ML array. The following table summarizes the reset controls available for the global AIE-ML array.

Table 13: **Categories of AIE-ML Resets**

Type	Trigger	Scope
Internal power-on-reset	Part of boot sequence	AIE-ML array
System reset	NPI input	AIE-ML array
INITSTATE reset	PCSR bit	AIE-ML array
Array soft reset	Software register write over NPI	AIE-ML array
AIE-ML tile column reset	Memory-mapped AIE-ML register bit in the array interface tile	AIE-ML tile column
AIE-ML array interface reset	From NPI register	AIE-ML array interface tile

The combination of column reset and array interface tile reset (refer to AIE-ML Array Hierarchy) enables a partial reconfiguration use case where a sub-array that comprises AIE-ML tiles and array interface tiles can be reset and reprogrammed without disturbing adjacent sub-arrays. The specifics of handling the array splitting and adding isolation depend on the type of use case (multi-user/tenancy or single-user/tenancy multiple-tasks).

Key Differences between AI Engine and AIE-ML

AI Engines offered in some AMD Versal™ adaptive SoCs are present in different versions optimized for different markets. The initial version, AI Engine (AIE), is optimized for DSP and communication applications, while the AI Engine-Machine Learning (AIE-ML) introduces a version optimized for machine learning. In this section, the main differences between AIE and AIE-ML are described, including:

- Increased throughput for ML/AI inference workloads.
- Optimized ML/AI application precision. For example, added bfloat16.
- Increased on-chip memory capacity and bandwidth (two times the data memory in each AIE-ML tile and the addition of AIE-ML memory tiles per column in the AIE-ML array).
- Increased multiplier performance.
- Focus on power efficiency (increase TOPs/W).
- Improved hardware for synchronization and reconfiguration.

The differences between the AIE and AIE-ML blocks include the following:

- Removed:
  - Native support for INT32. Multiplication of 32-bit numbers are not directly supported but are emulated via decomposition into multiple multiplications of 16 x 16 bit. Also supports cint32 x cint16 multiplication to optimize FFT performance.
  - Native FP32 (supported through emulation using bfloat16).
- Added:
  - Double INT8/16 compute per tile vs AIE
  - Bfloat16
  - Local memory tiles

TIP: *To understand the features in the first version of AI Engine, refer to the Versal Adaptive SoC AI Engine Architecture Manual (AM009).*

AIE-ML Array Architecture

This section compares differences in the arrays. For more information, see Chapter 3: AIE-ML Array Interface Architecture and AI Engine Array Interface Architecture in the *Versal Adaptive SoC AI Engine Architecture Manual* (AM009). The following provides a summary of the key features of AIE-ML that are similar to AIE:

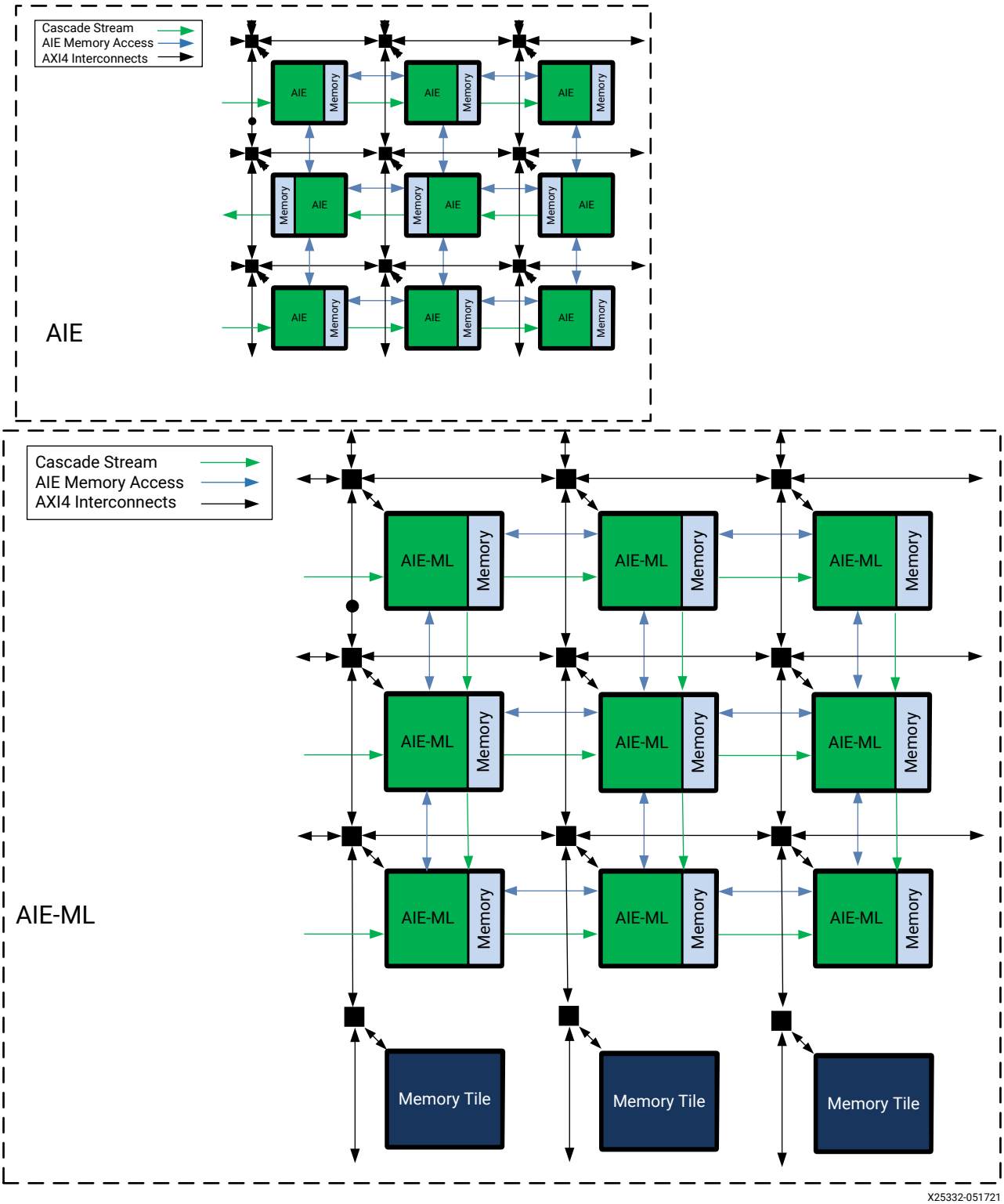
- Same process, voltage, frequency, clock, and power distribution
- Same array topology (one VLIW SIMD processor per AIE-ML tile)
- Each AIE-ML tile has eight integrated banks of data memory shared with three neighboring tiles.
- Each AIE-ML tile has two DMA channels in each direction
- AIE-ML tile to tile stream interconnect has same bandwidth as AIE
- Same PL and NoC interface
- Same debug/trace functionality

The following provides a summary of the key features of AIE-ML that are different or enhanced from AIE:

- At the tile level, the compute/memory is doubled. A processor bus is added to allow the AIE-ML perform direct read/write accesses to local tile memory mapped registers.
- Enhanced DMA are added to the AIE-ML tiles, AIE-ML memory tiles, and AIE-ML array interface tiles that include 3D address generation for tiles/array interface tiles and 4D address generation for memory tiles, out-of-order packets, and Finish-on-TLast in S2MM. Supports Compression and decompression (tiles and memory tiles) are supported to better handle sparse weights and activations in CNN and RNN application. See Sparsity for more information.
- Addition of AIE-ML memory tiles (maximum of two rows) to significantly reduce programmable logic (PL) resources (LUTs and UltraRAMs) utilization. There is 512 KB of memory per memory tile with ECC and 12 DMA channels (6 MM2S and 6 S2MM).
- Increased memory capacity due to the doubling of the data memory in AIE-ML tiles and the addition of AIE-ML memory tiles.
- Increase in power efficiency (TOPs/W).
- Improved stream switch functionality including source to destination parity check and deterministic merge.
- Improved reconfiguration and synchronization support.
- Grid array architecture to support vertical (from top to bottom) and horizontal (from left to right) 512-bit cascade, versus 384-bit horizontal cascade only.

The following figures show the change from checkerboard architecture in AIE to grid architecture in AIE-ML. Of note, in AIE-ML the tile rows are all in the same direction. The cascade connections are only from north to south and from west to east.

Figure 44: **AIE to AIE-ML Array Configuration**



AIE-ML Tile Architecture

The AIE-ML tile architecture leverages the functionality and performance requirements from the AI Engine tile architecture. The following provides an overview of the changes made to the AIE-ML tile architecture:

- AIE-ML (see AIE-ML Processor for more information)
- Data memory:
  - Data memory is increased from 32 KB to 64 KB organized as eight banks of 8 KB from a hardware perspective. From a programmer's perspective, every two banks are interleaved to form one bank, that is, a total of four banks of 16 KB. The AIE-ML tile has access to the four nearest memory modules in the cardinal directions: north, south, east (local data memory in the tile itself) and west.
  - Added memory zero-init
- DMA:
  - Features an improved address generation to support 3D addressing modes and iteration-state offset
  - Adds task queues and task-complete-tokens (see Task-Completion-Tokens for more information)
  - Supports S2MM Finish on TLAST and out-of-order packets
  - Added decompression to two S2MM channels
  - Added compression to two MM2S channels
  - Memory-mapped AXI4 interface: improved read and write bandwidth
- Lock module: 16 semaphore locks and each lock state is 6-bit unsigned versus 16 locks with binary data value in the AI Engine.

AIE-ML Processor

Similar to AIE, the AI Engine processor in AIE-ML consists of a scalar 32-bit data path, a SIMD vector data path, two load units, and a store unit, and is optimized for ML applications.

The following provides a list of AIE-ML processor features:

- Instruction-based VLIW SIMD processor with new instructions
- Same 16 KB program memory as in AIE
- Vector unit supports 256 (8b x 8b) and 512 (4b x 8b) MAC operations

- Vector unit supports 128 bloat16 MAC operations with FP32 accumulation
- Vector unit supports structure sparsity and FFT processing for ML inference applications, including cint32 x cint16 multiplication (data in cint32 and twiddle factor is cint16), control support for complex and conjugation, new permute mode, and shuffle mode. See Sparsity for more information.
- A new processor bus that allows the processor to access memory mapped registers in the local AIE-ML tile
- The complex circular addressing modes are dropped and replaced by a 3D addressing mode
- On-the-fly decompression during loading of sparse weights. See Sparsity for more information.

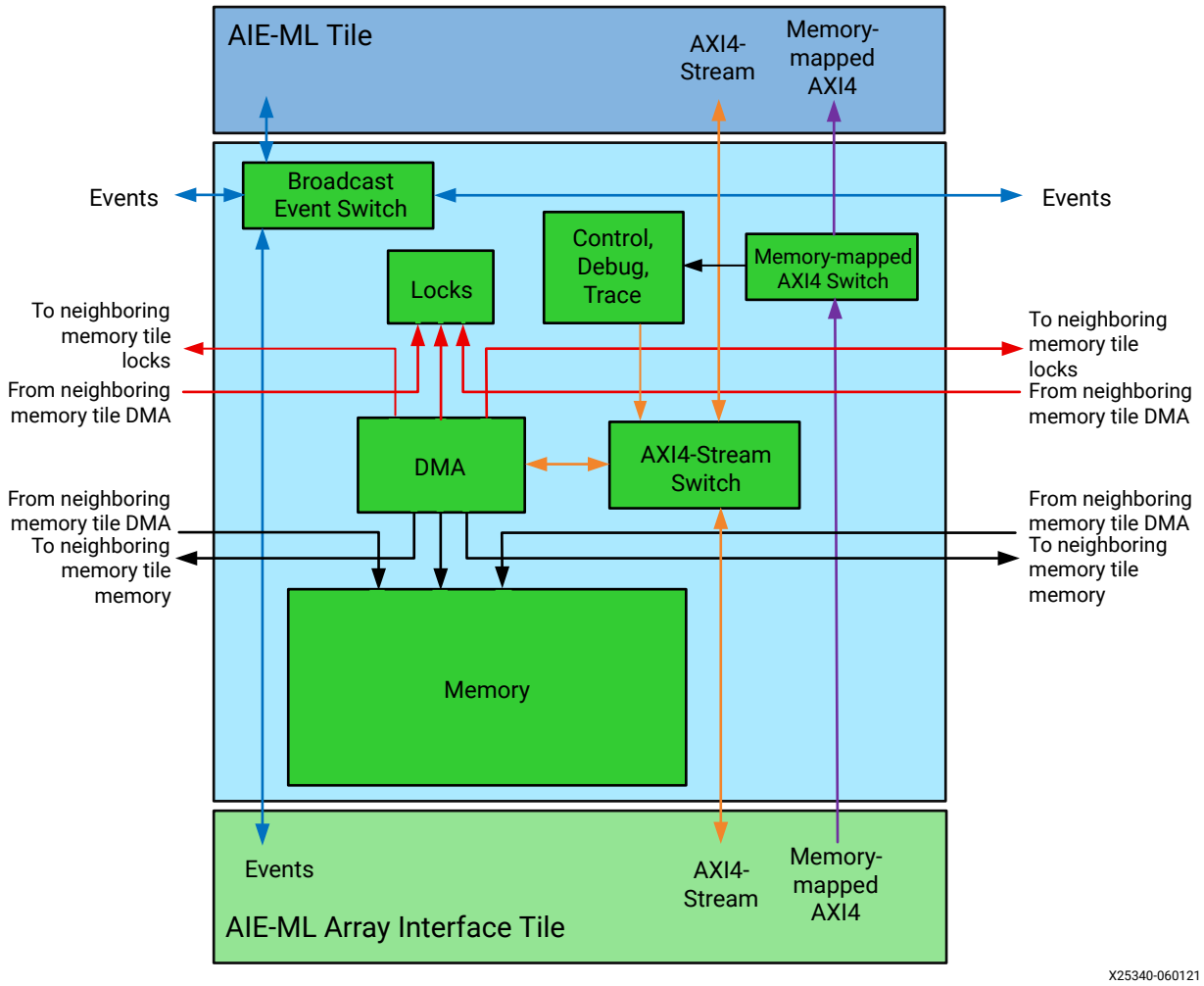
The AIE-ML processor removes some advanced DSP functionality used in the AIE processor including:

- 32-bit floating-point vector data path is not directly supported but can be emulated via decomposition into multiple multiplications of 16 x 16-bit
- Scalar non-linear functions, including sin/cos, sqrt, inverse sqrt and inverse
- Scalar floating point/integer conversions
- Complex circular addressing and FFT addressing modes. However, some level of FFT and complex support is provided; see the AIE-ML processor features.
- Limited support 128-bit load/store
- Non-aligned memory access
- Support for some complex data-types; some level of complex support is provided, see the AIE-ML processor features
- Native support for 32 × 32 multiplication but can be emulated using 16-bit integer operands
- Removal of non-blocking 128-bit stream interfaces and stream FIFOs
- Control streams and packet header generations

AIE-ML Memory Tile

Memory tiles are added in AIE-ML, not in AIE. See Chapter 2: AIE-ML Tile Architecture for more information. The following figure shows the AIE-ML memory tile architecture.

Figure 45: **AIE-ML Memory Tile Architecture**



The following is a high-level list of AIE-ML memory tile features:

- 512 KB memory arranged in 16 banks, ECC protected
- Supports up to 30 GB/s read and 30 GB/s write in parallel per memory tile
- DMA channels can directly access memory in the nearest neighbor memory tiles to the east and west
- Memory to stream DMA (MM2S) with six channels and support for 4D tensor address generation, zero-padding insertion, and compression. Accesses memory and locks in east and west neighboring tiles. Supports task queue and task-complete-tokens.
- Stream to memory DMA (S2MM) with six channels and support for 4D tensor address generation, out-of-order packet transfer, finish-on-TLAST, and decompression. Accesses memory and locks in east and west neighboring tiles. Supports task queue and task-complete-tokens.

- Stream switch in the AIE-ML memory tile shares the same design as the AIE-ML tile. There are 17 master ports and 18 slave ports, but no east or west streams.
- Locks module is accessible from neighboring AIE-ML memory tile DMA channels; there are 64 semaphore locks and each lock state is 6-bit unsigned.
- Additional control and status registers
- Configuration/debug interconnect with a 1 MB memory-mapped AXI4 address space per tile
- Debug and trace similar to that in the AIE-ML tile.

AIE-ML Array Interface

Similar to the Array interface in AIE, the AIE-ML array interface provides the necessary functionality to interface with the rest of the device. The array interface is made up of PL and NoC interface tiles, and there is one configuration interface tile per device. The following is a list of changes from the AIE array interface. The AIE-ML array interface has:

- AIE-ML array interface DMA (read and write to external memory)
  - Supports 32-bit aligned start addresses
  - 3D address generation and iteration-state offset that supports, with a single buffer descriptor (BD) configuration, an incremental offset to be added to the base address with each subsequent transfer. Also supports 32-bit aligned addresses to external memory.
  - Task queue and task-complete-tokens
  - Support for S2MM out-of-order and Finish-on-TLAST features (enabling compressed spill and restore of intermediate results to external memory)
  - Task queues and task complete tokens
- A lock design with 16 semaphore locks and 6-bit unsigned lock state
- One stream FIFO (stream switch). This is a reduction from two in the AIE array interface.
- Additional control and status registers for new features
- Memory-mapped AXI4 interface for improved read and write bandwidth

Software Programmability

Versal AI Engines are software programmable. You write C/C++ functions/kernels using intrinsic code targeting the VLIW ISA processors in a software programmable environment.

RECOMMENDED: *Use the AMD provided AI Engine library functions where appropriate, and write custom AI Engine kernels for the functions not covered by existing optimized library elements.*

AIE and AIE-ML have a different set of instructions and intrinsics. Kernels written with intrinsics are not portable across the versions of AI Engine. For portability between the versions of AI Engines, users should use the AI Engine API (see *AI Engine Kernel and Graph Programming Guide* (UG1079)) or the AI Engine libraries (see AI Engine Documentation on the AMD website).

Summary

Table 14: **Summary of the Key Differences between AIE and AIE-ML**

	AIE	AIE-ML[1]
Array structure	Checkerboard	All lines identical
Cascade interface	384-bits wide Horizontal direction	512-bits wide Horizontal and vertical directions
Tile stream interface	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
Advanced DSP functionality	Yes	No
INT4 operations/tile	256	512
INT8 operations/tile	256	512
INT16 operations/tile	64	128
INT32 operations/tile	16	32[4]
Bfloat16 float operations/tile	–	256
FP32 float operations/tile	16	4[2][3]
Data memory/tile	32 KB	64 KB
Program memory/tile	16 KB	16 KB
Memory tiles	–	512 KB
Programmable logic (PL) to AIE array bandwidth	1X	1X
Tile local memory DMA	–	Support for • 3D addressing modes • S2MM finish on `TLAST` and out-of-order packets • Compression/ decompression

Table 14: **Summary of the Key Differences between AIE and AIE-ML** *(cont'd)*

	AIE	AIE-ML[1]
Local memory locks	Boolean	Semaphore

Notes:

1. In cases without sparsity and for ML applications. See Sparsity for more information.
2. Actually INT8 x INT4.
3. Emulation mode: The specified value gives 16b accuracy for the mantissa. The number is reduced for higher accuracy on the mantissa.
4. int32 × int32 can be emulated.

Appendix B

Additional Resources and Legal Notices

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help → Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

Support Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Support.

References

These documents provide supplemental material useful with this guide:

1. *Versal Adaptive SoC AI Engine Architecture Manual* (AM009)
2. *Versal Adaptive SoC Technical Reference Manual* (AM011)
3. *Versal Architecture and Product Data Sheet: Overview* (DS950)
4. *Versal AI Core Series Data Sheet: DC and AC Switching Characteristics* (DS957)
5. *Versal AI Edge Series Data Sheet: DC and AC Switching Characteristics* (DS958)
6. *AI Engine Tools and Flows User Guide* (UG1076)
7. *AI Engine Kernel and Graph Programming Guide* (UG1079)
8. *Bootgen User Guide* (UG1283)
9. Answer Record 34376
10. AI Engine Technology

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
06/16/2025 Version 1.4	
Introduction to Versal Adaptive SoCs	Updated the section.
AIE-ML to Programmable Logic Interface	Updated the section.
Clocking Structure	Added the section.
AIE-ML Array Interface	Updated the figure and added a note.
Data Movement	Updated the section.
AIE-ML Interfaces	Updated the figure.
AIE-ML Memory Tile Memory	Updated the paragraphs.
05/15/2024 Version 1.3	
AIE-ML Interfaces	Updated the figure.

Section	Revision Summary
Memory-mapped AXI4 Interconnect	Removed restrictions.
AIE-ML Array Reconfiguration	Removed the Partial reconfiguration bullet.
11/10/2023 Version 1.2	
Task-Completion-Tokens	Added section.
Sparsity	Added section.
02/15/2023 Version 1.1	
Functional Overview	Changed bullet under table from "Each of the two multipliers can be signed or unsigned." to "The multiplier multiplicand can be signed or unsigned."
Chapter 2: AIE-ML Tile Architecture	Added text regarding the memory tile being initialized to zero at boot and reset.
AIE-ML Memory Tile Overview and Features	Added a bullet to the second list regarding the memory tile being initialized to zero at boot and reset.
AIE-ML Memory Tile Memory	Updated both figures and added two notes.
Appendix B: Additional Resources and Legal Notices	Added section.
09/28/2022 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright