Versal ACAP AI Engine

Architecture Manual

AM009 (v1.1) April 22, 2021

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
4/22/2021 Version 1.1	
General	Added links to *Versal ACAP AI Engine Register Reference* (AM015)
Memory Error Handling	Clarified how the performance counter during memory-mapped AXI4 access works.
Arithmetic Logic Unit, Scalar Functions, and Data Type Conversions	Added implementation notes on managing the `Float2fix` conversion for overflow exceptions.
7/16/2020 Version 1.0	
Initial release.	N/A

Table of Contents

Revision History ... **2**

Chapter 1: Overview ... **5**

Introduction to Versal ACAP ... 5
AI Engine Array Features ... 7
AI Engine Array Overview ... 8
AI Engine Array Hierarchy ... 9
AI Engine Applications ... 10
Performance ... 11
Memory Error Handling ... 11

Chapter 2: AI Engine Tile Architecture ... **13**

Memory Mapped AXI4 Interconnect ... 16
AXI4-Stream Interconnect ... 16
AI Engine Tile Program Memory ... 18
AI Engine Interfaces ... 18
AI Engine Memory Module ... 21
AI Engine Data Movement Architecture ... 22
AI Engine Debug ... 26
AI Engine Trace and Profiling ... 27
AI Engine Events ... 31

Chapter 3: AI Engine Array Interface Architecture ... **34**

AI Engine Array Interface ... 36
Features of the AI Engine Array Interface ... 39
Array Interface Memory-Mapped AXI4 Interconnect ... 40
Array Interface AXI4-Stream Interconnect ... 40
AI Engine to Programmable Logic Interface ... 40
AI Engine to NoC Interface ... 41
Interrupt Handling ... 41

Chapter 4: AI Engine Architecture ... **44**

Functional Overview ... 44

Register Files....47
Instruction Fetch and Decode Unit....49
Load and Store Unit....49
Scalar Unit....50
Vector Unit....53
Register Move Functionality....59

Chapter 5: AI Engine Configuration and Boot....60
AI Engine Array Configuration....60
AI Engine Boot Sequence....60
AI Engine Array Reconfiguration....61

Appendix A: Additional Resources and Legal Notices....63
Xilinx Resources....63
Documentation Navigator and Design Hubs....63
References....63
Please Read: Important Legal Notices....64

Overview

Introduction to Versal ACAP

Versal™ adaptive compute acceleration platforms (ACAPs) combine Scalar Engines, Adaptable Engines, and Intelligent Engines with leading-edge memory and interfacing technologies to deliver powerful heterogeneous acceleration for any application. Most importantly, Versal ACAP hardware and software are targeted for programming and optimization by data scientists and software and hardware developers. Versal ACAPs are enabled by a host of tools, software, libraries, IP, middleware, and frameworks to enable all industry-standard design flows.

Built on the TSMC 7 nm FinFET process technology, 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 includes six series of devices 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 combines different engine types with a wealth of connectivity and communication capability and a network on chip (NoC) to enable seamless memory-mapped access to the full height and width of the device. Intelligent Engines are SIMD VLIW AI Engines for adaptive inference and advanced signal processing compute, and DSP Engines for fixed point, floating point, and complex MAC operations. Adaptable Engines are a combination of programmable logic blocks and memory, architected for high-compute density. Scalar Engines, including Arm® Cortex®-A72 and Cortex-R5F processors, allow for intensive compute tasks.

The Versal AI Core series delivers breakthrough AI inference acceleration with AI Engines that deliver over 100x greater compute performance than current server-class of CPUs. 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 is the foundation and the mid-range of the Versal platform, 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 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 wired communications, data center, test & measurement, and other applications. Versal Premium series ACAPs include 112G PAM4 transceivers and integrated blocks for 600G Ethernet, 600G Interlaken, PCI Express® Gen5, and high-speed cryptography.

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

Navigating Content by Design Process

Xilinx® documentation is organized around a set of standard design processes to help you find relevant content for your current development task. All Versal™ ACAP design process Design Hubs can be found on the Xilinx.com website. 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 AI Engine architecture and includes:
    - AI Engine Array Overview
    - AI Engine Array Hierarchy
    - Performance
  - Chapter 2: AI Engine Tile Architecture describes the interaction between the memory module and the interconnect and between the AI Engine and the memory module.
  - Chapter 3: AI Engine Array Interface Architecture is a high-level view of the AI Engine array interface to the PL and NoC.
  - Chapter 4: AI Engine Architecture describes the processor functional unit and register files.
  - Chapter 5: AI Engine Configuration and Boot describes configuring the AI Engine 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: AI Engine Tile Architecture
- Chapter 4: AI Engine Architecture

AI Engine Array Features

Some Versal ACAPs include the AI Engine array that consists of an array of AI Engine tiles and the AI Engine array interface consisting of the network on chip (NoC) and programmable logic (PL) tiles. The following lists the features of each.

AI Engine Tile Features

- A separate building block, integrated into the silicon, outside the programmable logic (PL)
- One AI Engine incorporates a high-performance very-long instruction word (VLIW) single-instruction multiple-data (SIMD) vector processor optimized for many applications including signal processing and machine learning applications among others
- Eight banks of single-port data memory for a total of 32 KB
- Streaming interconnect for deterministic throughput, high-speed data flow between AI Engines and/or the programmable logic in the Versal device
- Direct memory access (DMA) in the AI Engine 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 AI Engine tile
- Hardware synchronization primitives (for example, locks) provide synchronization of the AI Engine, between the AI Engine and the tile DMA, and between the AI Engine and an external master (through the memory-mapped AXI4 interface)
- Debug, trace, and profile functionality

AI Engine Array Interface to NoC and PL Resources

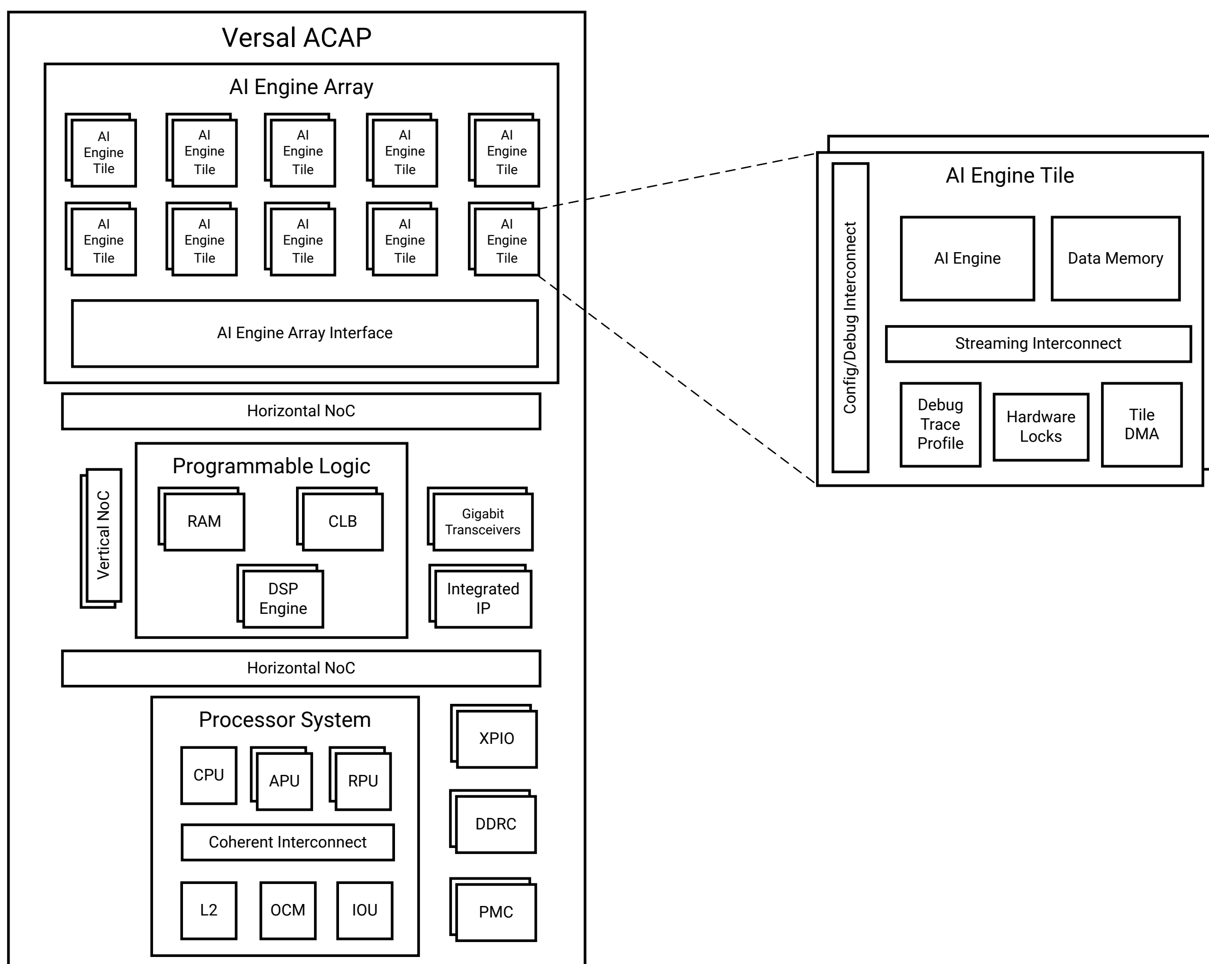
- Direct memory access (DMA) in the AI Engine array interface NoC tile manages incoming and outgoing memory-mapped and streams traffic into and out of the AI Engine array
- Configuration and control interconnect functionality (through the memory-mapped AXI4 interface)
- Streaming interconnect that leverages the AI Engine tile streaming interconnect functionality
- AI Engine to programmable logic (PL) interface that provides asynchronous clock-domain crossing between the AI Engine clock and the PL clock
- AI Engine to NoC interface logic to the NoC master unit (NMU) and NoC slave unit (NSU) components

- Hardware synchronization primitives (for example, locks) leverage features from the AI Engine tile locks module
- Debug, trace, and profile functionality that leverage all the features from the AI Engine tile

AI Engine Array Overview

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

Figure 1: **Versal Device Top-Level Block Diagram**

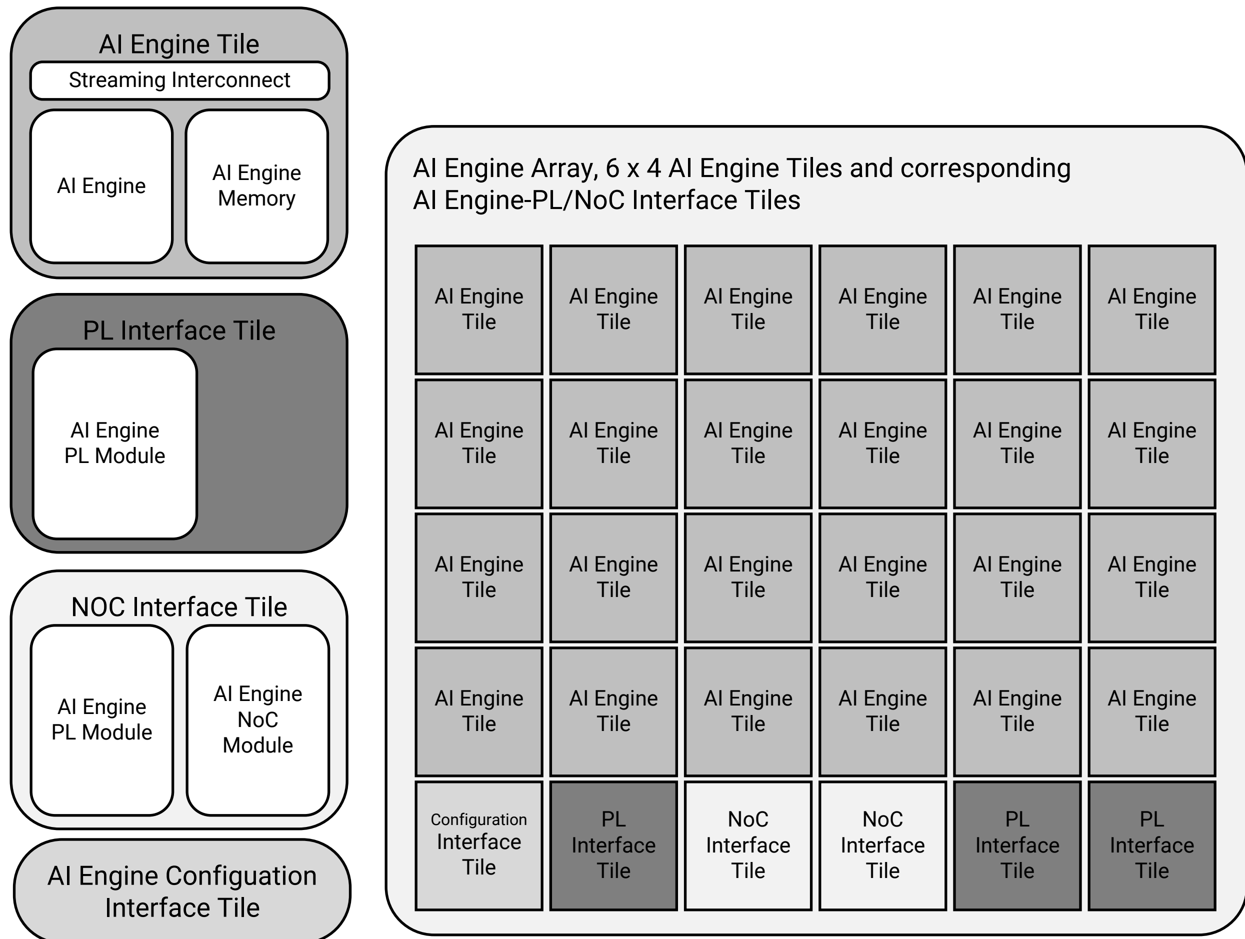


The AI Engine array is the top-level hierarchy of the AI Engine architecture. It integrates a two-dimensional array of AI Engine tiles. Each AI Engine tile integrates a very-long instruction word (VLIW) processor, integrated memory, and interconnects for streaming, configuration, and debug. The AI Engine array interface enables the AI Engine to communicate with the rest of the Versal device through the NoC or directly to the PL. The AI Engine array also interfaces to the processing system (PS) and platform management controller (PMC) through the NoC.

AI Engine Array Hierarchy

The AI Engine array is made up of AI Engine tiles and AI Engine array interface tiles (the last row of the array). The types of interface tiles include the AI Engine to PL and AI Engine to NoC interface tiles. There is also exactly one configuration interface tile in each AI Engine array that contains a PLL for AI Engine clock generation and other global control functions. The following figure shows a conceptual view of the complete tile hierarchy associated with the AI Engine array. See Chapter 2: AI Engine Tile Architecture and Chapter 3: AI Engine Array Interface Architecture for detailed descriptions of the various tiles.

Figure 2: **Hierarchy of Tiles in a AI Engine Array**



AI Engine Applications

The non-linear increase in demand in next generation wireless, machine learning, and other compute intensive applications lead to the development of the Versal™ ACAP AI Engine. The AI Engine, 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 ACAP. The AI Engine 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 ACAP AI Engines are optimized for various compute-intensive applications, for example, advanced radio systems supporting all classical radio functionality plus wideband/multiband capability, 5G wireless communications (remove the need of vector-DSP-based ASICs), and machine learning inference acceleration in data center applications by enabling deterministic latency and low neural network latency with acceptable performance.

Performance

The AI Engine array has a single clock domain for all the tiles and array interface blocks. The performance target of the AI Engine array for the -1L speed grade devices is 1 GHz with V_{CCINT} at 0.70V. In addition, the AI Engine array has clocks for interfacing to other blocks. The following table summarizes the various clocks in the AI Engine array and their performance targets.

Table 1: **AI Engine Interfacing Clock Domains**

Clock	Target for -1L	Source	Relation to AI Engine Clock
AI Engine array clock	1 GHz	AI Engine PLL	N/A
NoC clock	800 MHz	NoC clocking	Asynchronous, CDC within NoC
PL clocks	500 MHz	PL clocking	Asynchronous, CDC within AI Engine array interface
NPI clock	300 MHz	NPI clocking	Asynchronous

Memory Error Handling

Memory Error Detection and Correction

Each AI Engine has 32 KB of data memory and 16 KB of program memory. For devices with many AI Engine tiles, protection against soft errors are both required and provided. 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 AI Engine DMA or memory-mapped AXI4.

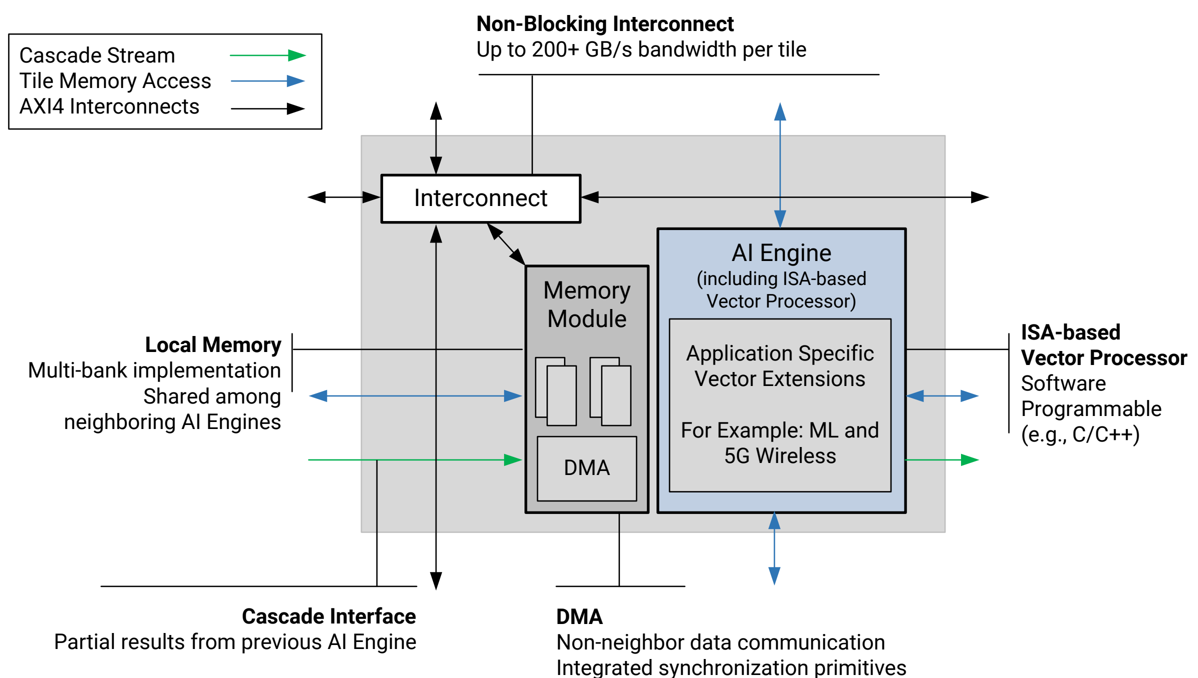
When the memory-mapped AXI4 access reads or writes to AI Engine data memory, two requests are sent to the memory module. On an ECC/parity event, the event might be counted twice in the AI Engine performance counter. There is duplicate memory access but no impact on functionality. Refer to Chapter 2: AI Engine 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.

AI Engine Tile Architecture

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

Figure 3: **AI Engine Tile Block Diagram**



The AI Engine tile consists of the following high-level modules:

- Tile interconnect
- AI Engine
- AI Engine 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 AI Engine memory module has 32 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 AI Engine 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

AI Engine. The AI Engine has a scalar processor, a vector processor, three address generators, and 16 KB of program memory. It also has a cascade stream access for forwarding accumulator output to the next AI Engine tile. The AI Engine is described in more detail in Chapter 4: AI Engine Architecture. Both the AI Engine and the AI Engine memory module have control, debug, and trace units. Some of these units are described later in this chapter:

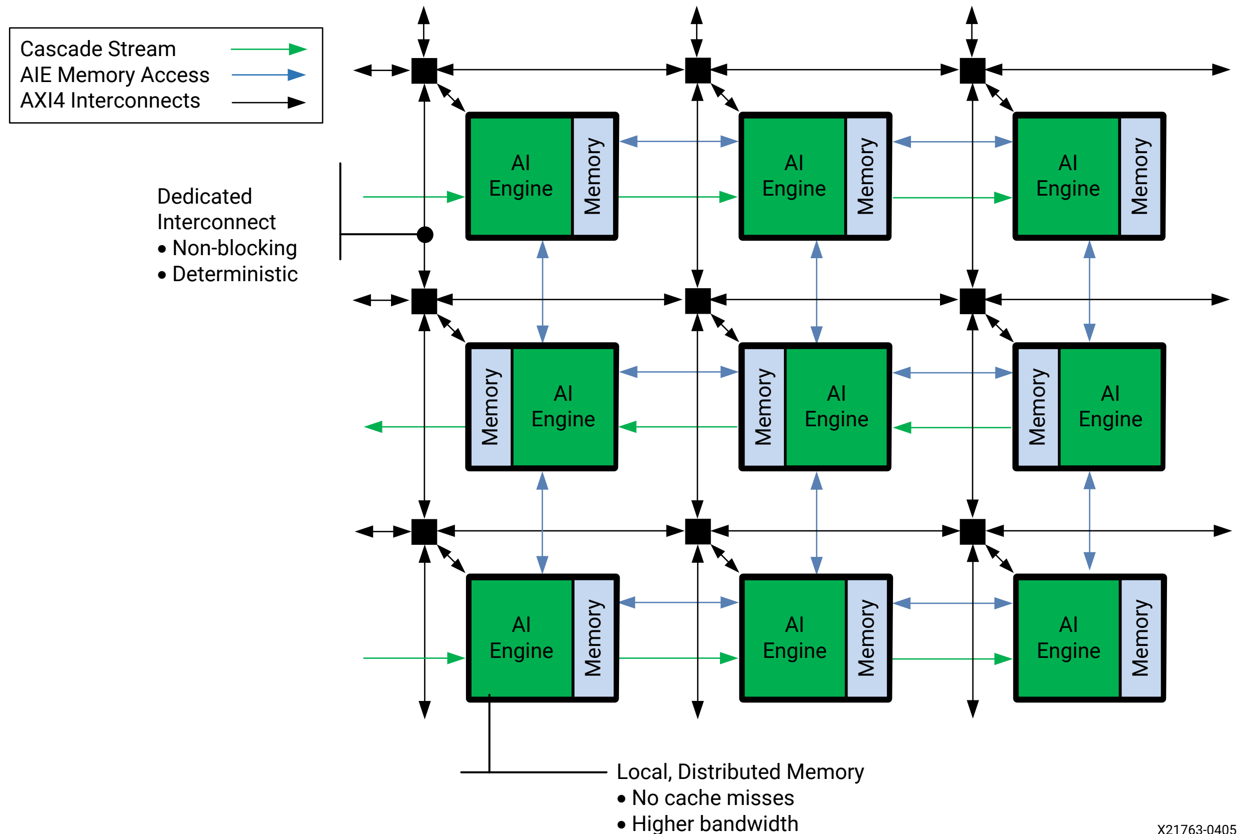
- Control and status registers
- Events, event broadcast, and event actions
- Performance counters for profiling and timers

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

- Its own
- The module on the north
- The module on the south
- The module on the east or west depending on the row and the relative placement of AI Engine and memory module

The AI Engines on the edges of the array have access to one or two fewer memory modules, following a checkerboard pattern.

Figure 4: **AI Engine Array**



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

The cascade streams travel from tile to tile in horizontal manner from the bottom row to the top. As a cascade stream reaches the edge at one end, it is connected to the input of the tile above it. Therefore, the flow changes direction on alternate rows (west to east on one row, and east to west on another). The cascading continues until it reaches one end of the top row at which point the stream ends with no further connection. Because of the change in direction, the relative placement of the AI Engine and memory module in a tile is reversed from one row to the other.

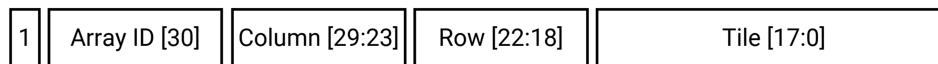
Memory Mapped AXI4 Interconnect

Each AI Engine 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 AI Engine tile. The memory-mapped AXI4 interconnect inside the AI Engine array can be driven from outside of the AI Engine array by any AXI4 master that can connect to the network on chip (NoC). All internal resources in an AI Engine tile including memory, and all registers in an AI Engine and AI Engine memory module, are mapped onto a memory-mapped AXI4 interface.

Each AI Engine tile has a memory-mapped AXI4 switch that will accept 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 AI Engine tile. The lower 18 bits represent the tile address range of `0x00000` to `0x3FFFF`, followed by five bits that represent the row location and seven bits that represent the column location.

Figure 5: **AI Engine Memory-Mapped AXI4 Interface Addresses**



The AI Engine internal memory-mapped AXI4 interconnect is a subset of the full memory-mapped AXI4 protocol, with the following limitations.

- No write data before write address
- Only one `WSTRB` signal for the write data
- Only burst of one to four, 32-bit words
- 32-bit fixed size

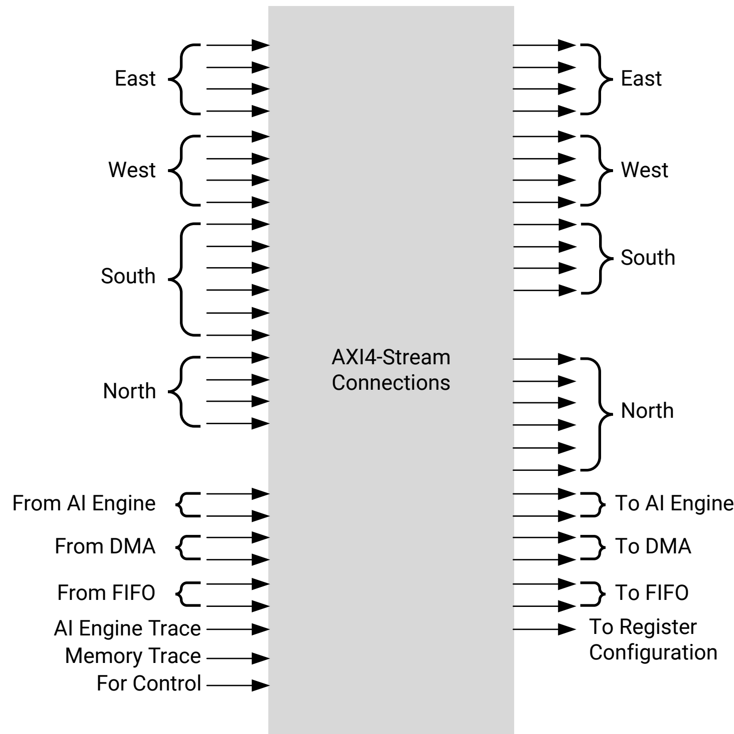
AXI4-Stream Interconnect

Each AI Engine 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 following figure shows an AXI4-Stream interconnect. The building blocks of the AXI4-Stream interconnect are listed as follows.

- Port handlers

- FIFOs
- Arbiters
- Stream switch configuration registers

Figure 6: **AXI4-Stream Switch High-level Block Diagram**



Each port has a port handler that selects the route for the input/output stream. Each switch has two FIFO buffers (16-deep, 32 bit data + 1 bit TLAST wide) that can be chained together and used for adding buffering to a stream. Each switch has six programmable arbiters for packet switching.

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. Packet-switched streams 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 will cause performance degradation.

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

Table 2: **AI Engine 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	AI Engine (1 GHz)	4	24
To South/From North	4	32	AI Engine (1 GHz)	4	16
To West/From East	4	32	AI Engine (1 GHz)	4	16
To East/From West	4	32	AI Engine (1 GHz)	4	16

AI Engine Tile Program Memory

The AI Engine 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
- AI Engine interface

An external master can read or write to the program memory using the memory-mapped AXI4 interface. The AI Engine has 128-bit wide interfaces to the program memory to fetch instructions. The AI Engine can read from, but not write to, the program memory. To access the program memory simultaneously from the memory-mapped AXI4 and AI Engine, 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.

AI Engine Interfaces

The AI Engine has multiple interfaces. The following block diagram shows the interfaces.

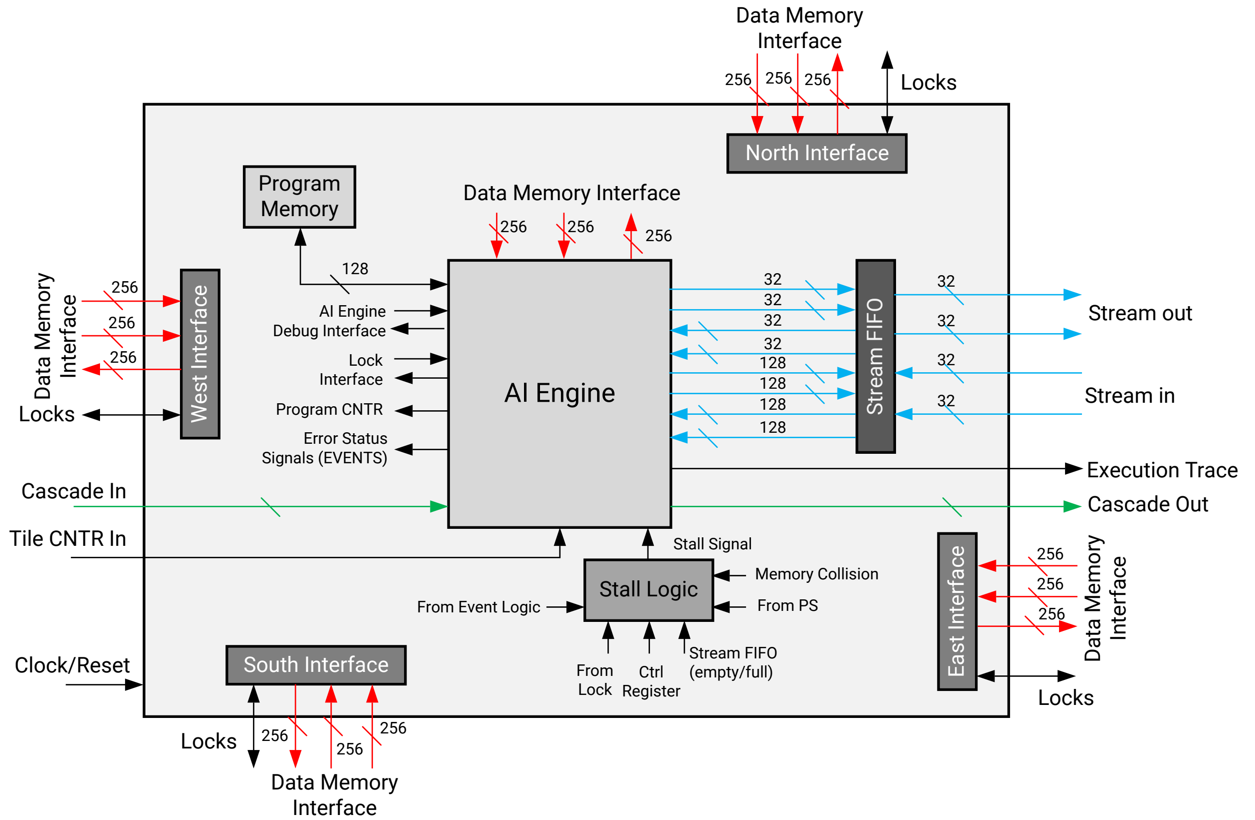
- **Data Memory Interface:** The AI Engine can access data memory modules on all four directions. They are accessed as one contiguous memory. The AI Engine has two 256-bit wide load units and one 256-bit wide store unit. From the AI Engines perspective, the throughput of each of the loads (two) and store (one) is 256 bits per clock cycle.

- **Program Memory Interface:** This 128-bit wide interface is used by the AI Engine to access the program memory. A new instruction can be fetched every clock cycle.

- **Direct AXI4-Stream Interface:** The AI Engine has two 32-bit input AXI4-Stream interfaces and two 32-bit output AXI4-Stream interfaces. Each stream is connected to a FIFO both on the input and output side, allowing the AI Engine to have a 4 word (128-bit) access per 4 cycles, or a 1 word (32-bit) access per cycle on a stream.

- **Cascade Stream Interface:** The 384-bit accumulator data from one AI Engine can be forwarded to another by using these cascade streams to form a chain. There is a small, two-deep, 384-bit wide FIFO on both the input and output streams that allow storing up to four values between AI Engines.
- **Debug Interface:** This interface is able to read or write all AI Engine registers over the memory-mapped AXI4 interface.
- **Hardware Synchronization (Locks) Interface:** This interface allows synchronization between two AI Engines or between an AI Engine and DMA. The AI Engine can access the lock modules in all four directions.
- **Stall Handling:** An AI Engine 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.
- **AI Engine 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 AI Engine generated packet-based execution trace can be sent over the AXI4-Stream.

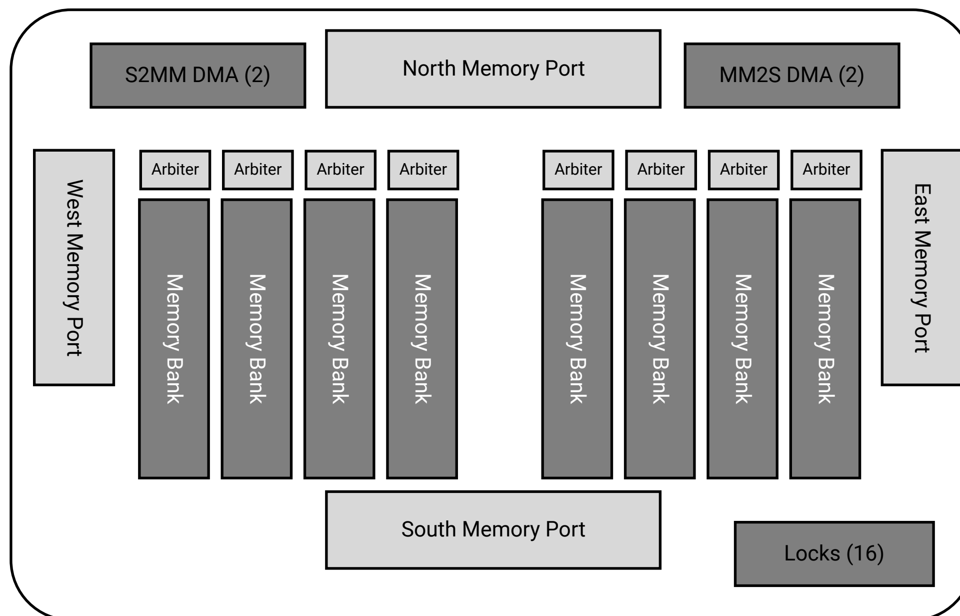
Figure 7: **AI Engine Interfaces**



AI Engine Memory Module

The AI Engine 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 8: **AI Engine Memory Module**



- **Memory Banks:** The AI Engine memory modules consist of eight memory banks, where each memory bank is a 256 word x 128-bit single-port memory. 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 will be allowed to access the memory. The other requesters are stalled for one cycle and the hardware will retry 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 AI Engine 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 AI Engine and DMA or any external memory-mapped AXI4 master (outside of the AI Engine 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.
- **Lock Module:** The AI Engine memory module contains a lock module to achieve synchronization amongst the AI Engines, tile DMA, and an external memory-mapped AXI4 interface master (for example, the processor system (PS)). There are 16 hardware locks with a binary data value for each AI Engine memory module lock unit. Each lock has an arbitrator for simultaneous request management. The lock module handles lock requests from the AI Engines in all four directions, the local DMA controller, and memory-mapped AXI4.

AI Engine Data Movement Architecture

This section describes examples of the data communications within the AI Engine array and between the AI Engine tile and programmable logic (PL).

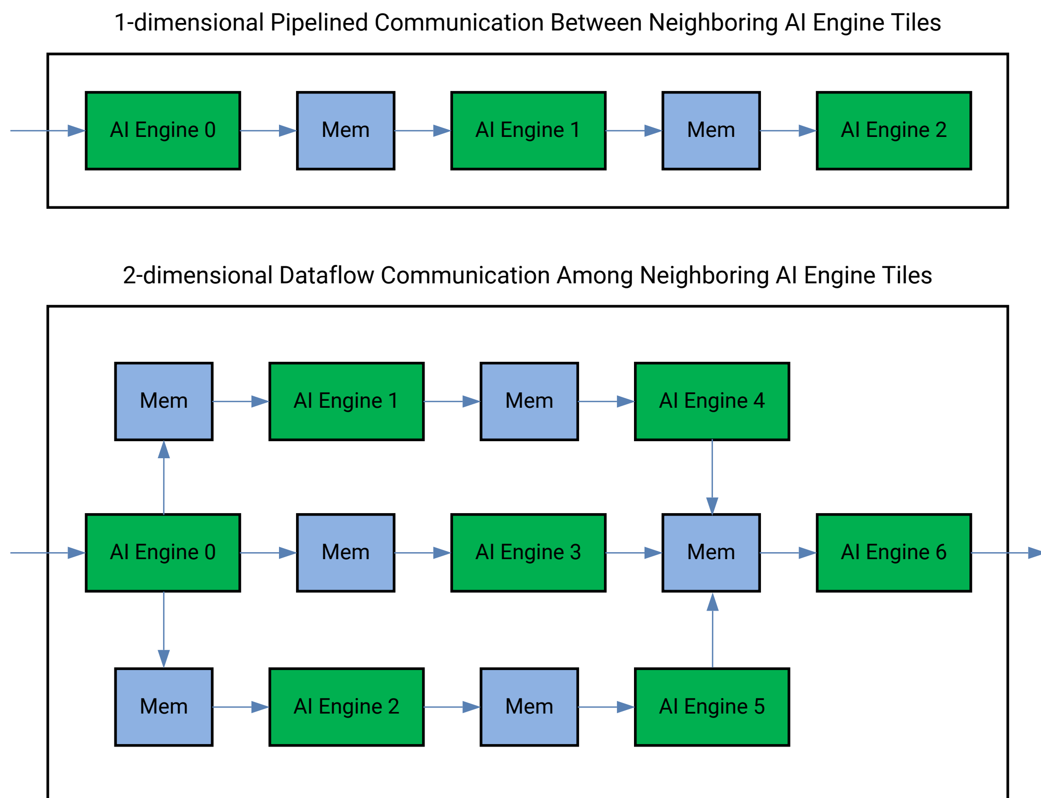
AI Engine to AI Engine Data Communication via Shared Memory

AI Engine to AI Engine Data Communication via Shared Memory

In the case where multiple kernels fit in a single AI Engine, 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 AI Engine, 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 AI Engines 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 the data communication between the AI Engine tiles. They are a logical representation of the AI Engine tiles and shared memory modules.

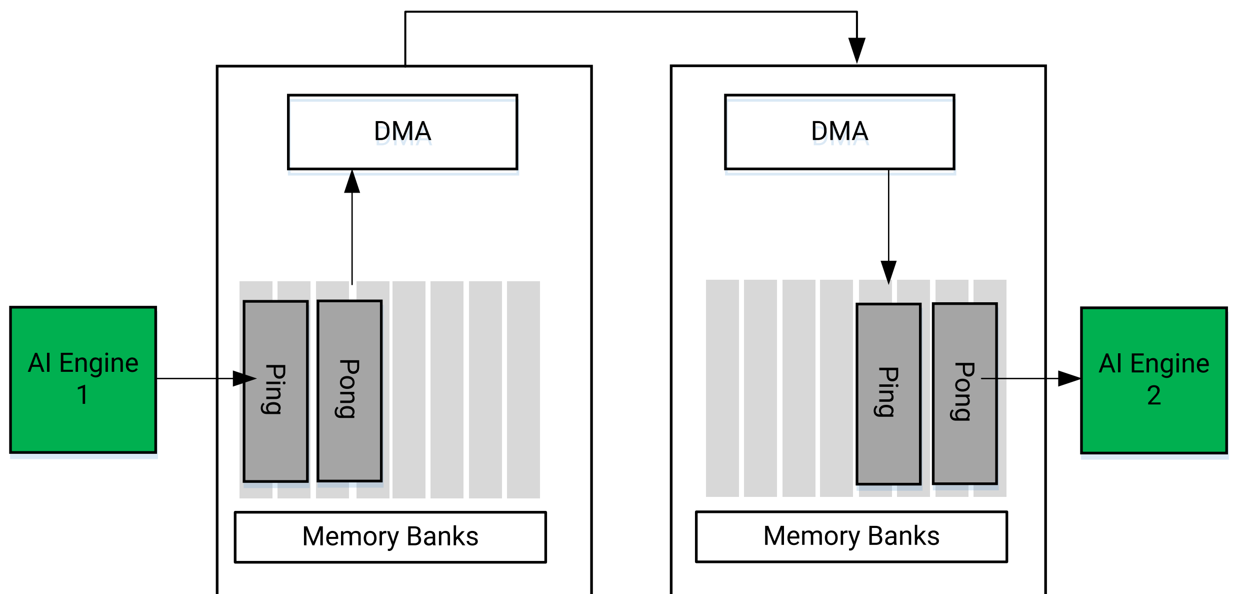
Figure 9: **AI Engine to AI Engine Data Communication via Shared Memory**



AI Engine Tile to AI Engine Tile Data Communication via Memory and DMA

The communication described in the previous section is inside an AI Engine tile or between two neighboring AI Engine tiles. For non-neighboring AI Engine tiles, a similar communication can be established using the DMA in the memory module associated with each AI Engine 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 AI Engine to AI Engine Data Communication via Shared Memory section. The main differences are increased communication latency and memory resources.

Figure 10: **Data Communication Between Two Non-neighboring AI Engine Tiles**

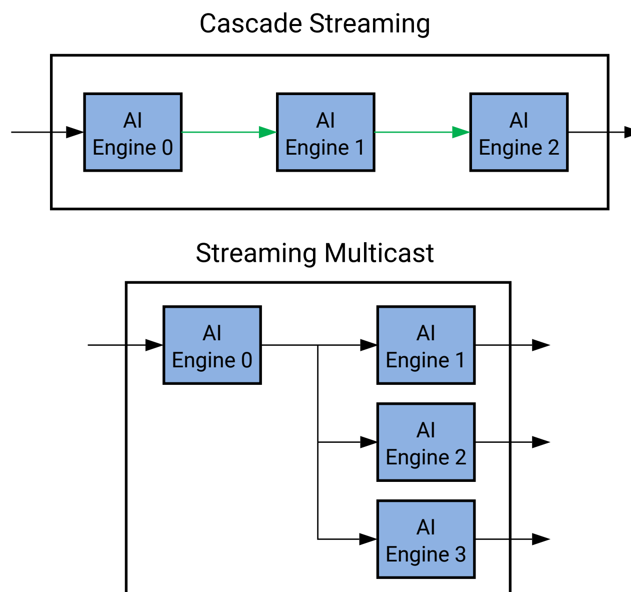


AI Engine Tile to AI Engine Tile Data Communication via AXI4-Stream Interconnect

AI Engines 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 AI Engine to another through the streaming interface in a serial fashion, or the same information can be sent to an arbitrary number of AI Engine 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 will stop until all receivers are ready again.

Figure 11: **AI Engine to AI Engine Data Communication via AXI4-Stream Interconnect**

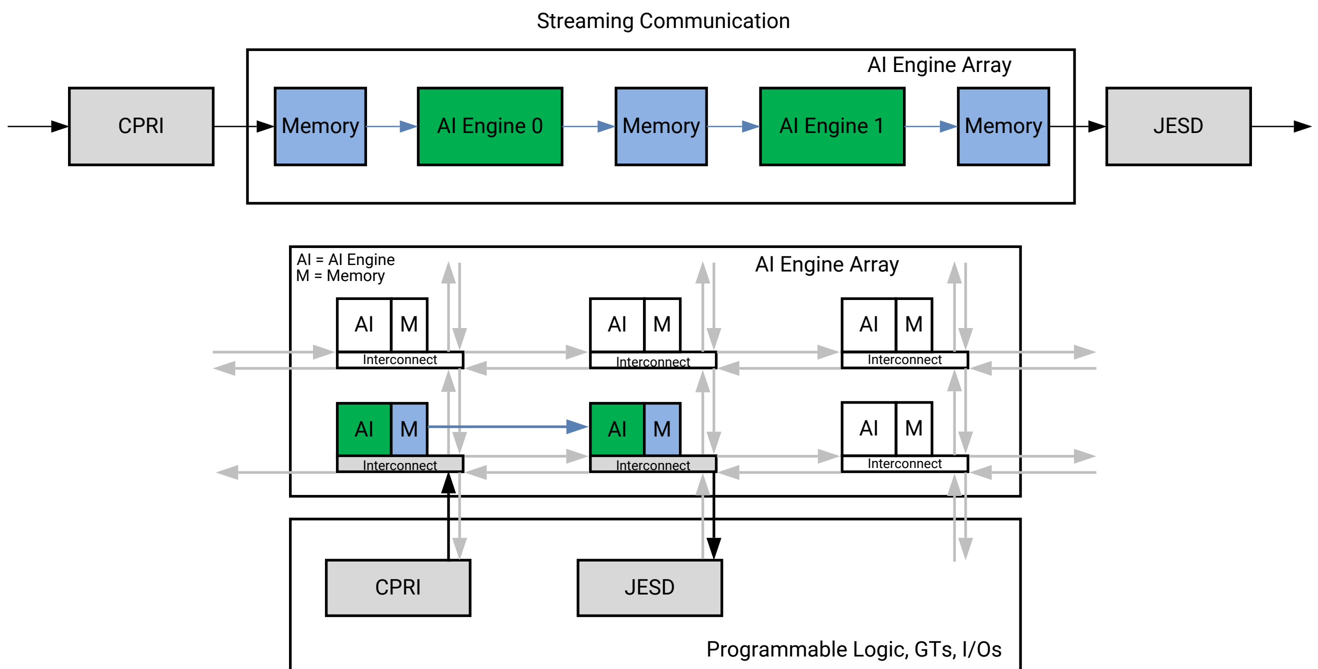


AI Engine to PL Data Communication via Shared Memory

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 AI Engine stream. The AI Engine stream is then routed to the AI Engine 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 (use case) between the common public radio interface (CPRI™) and the JESD® in the PL. The AI Engine and PL can communicate using DMA in the AI Engine tile. The DMA moves the stream into a memory block neighboring the consuming AI Engine. The first diagram represents the logical view and the second diagram represents the physical view.

Figure 12: **AI Engine to PL Data Communication via Shared Memory**



AI Engine Debug

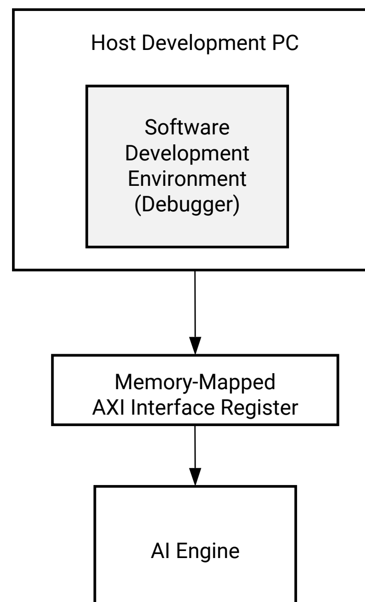
Debugging the AI Engine uses the memory-mapped AXI4 interface. All the major components in the AI Engine array are memory mapped.

- Program memories

- Data memories
- AI Engine registers
- DMA registers
- Lock module registers
- Stream switch registers
- AI Engine 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 AI Engine 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 13: **Overview of the AI Engine Debug Interface**



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

AI Engine Trace and Profiling

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

- AI Engine modes
  - Event-time
  - Event-PC
  - Execution-trace
- AI Engine memory module mode
  - Event-time
- AI Engine PL module mode
  - Event-time

The trace is output from the unit through the AXI4-Stream as an AI Engine 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 AI Engine destination it can be routed to, including AI Engine local data memory through the AI Engine tile DMA, external DDR memory through the AI Engine array interface DMA, and block RAM or URAM through the AI Engine 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 AI Engine 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 AI Engine together with the current value of the eight watched events. The frames are collected in an output buffer into an AI Engine packet-switched stream packet.

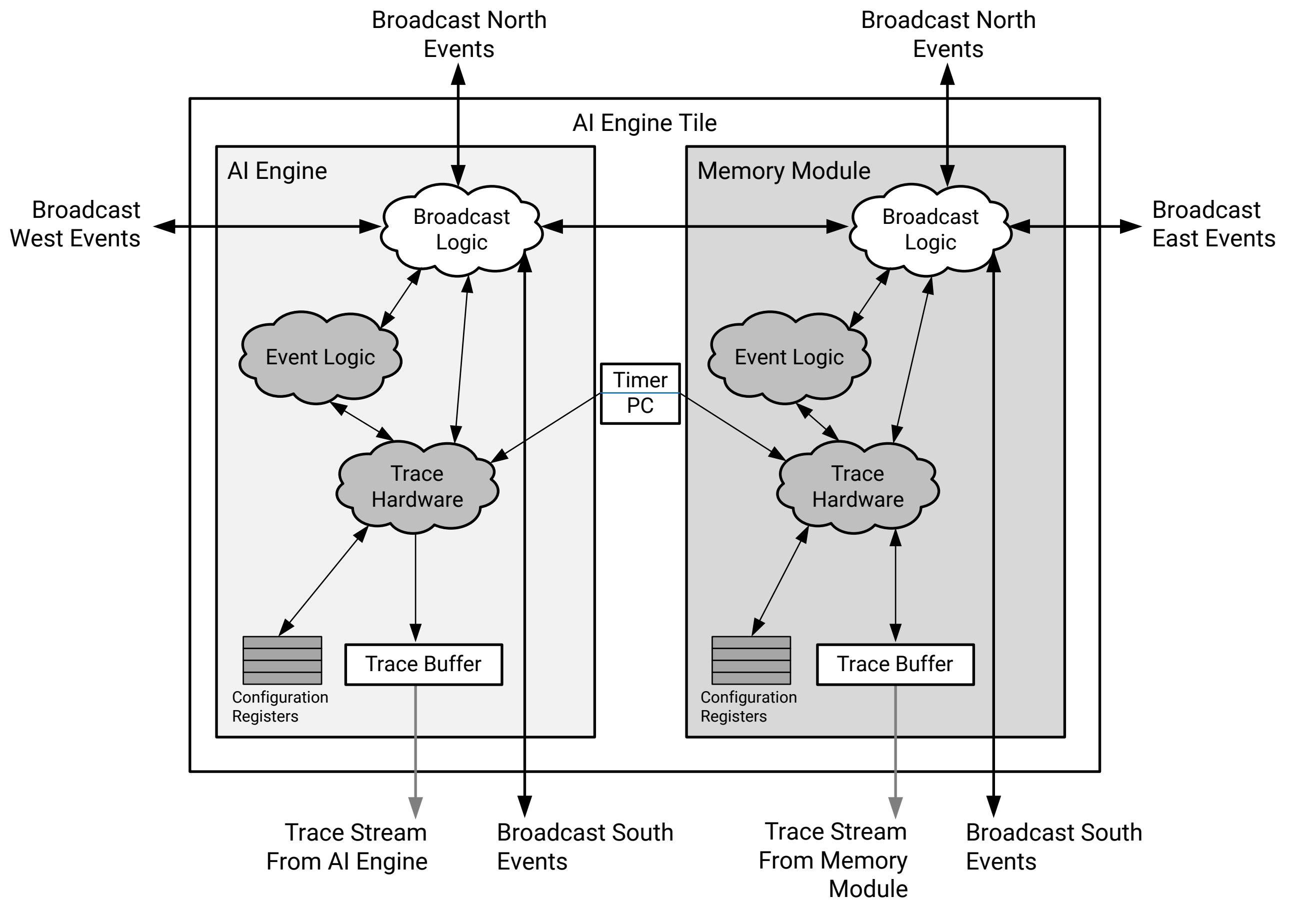
The trace unit in the AI Engine can operate in execution-trace mode. In real time, the unit will send, via the AXI4-Stream, a minimum set of information to allow an offline debugger to reconstruct the program execution flow. This assumes the offline debugger has access to the ELF. The information includes:

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

The AI Engine 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 AI Engine 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 14: **Logical View of AI Engine Trace Hardware**

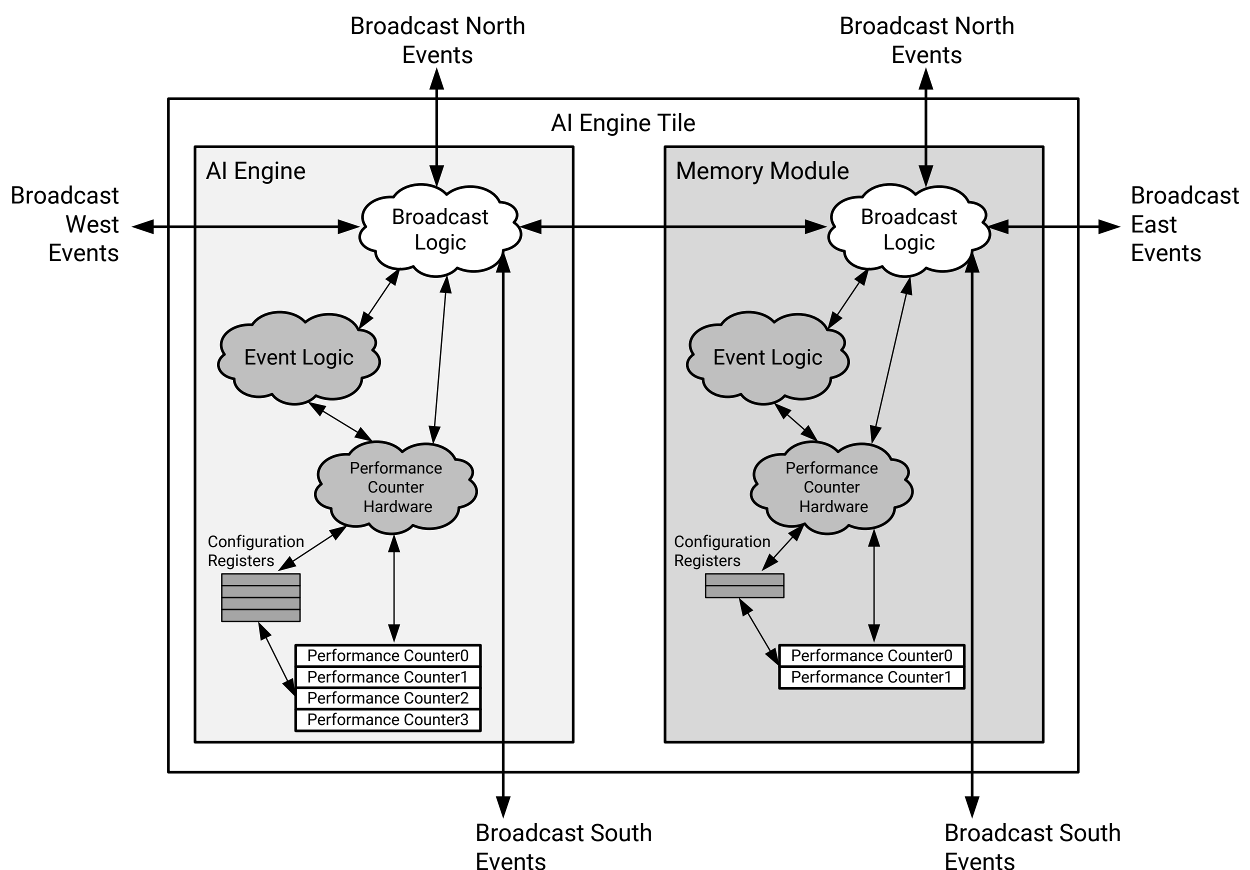


To control the trace stream for an event trace, there is a 32-bit `trace_control0/1` register to start and stop the trace. There are also the `trace_event0/1` registers to program the internal event number to be added to the trace. See the *Versal ACAP AI Engine Register Reference* (AM015) for specific register information.

Profiling (Performance Counters)

The AI Engine array has performance counters that can be used for profiling. The AI Engine 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 AI Engine tile. The performance control registers and performance counter registers are described in the *Versal ACAP AI Engine Register Reference* (AM015).

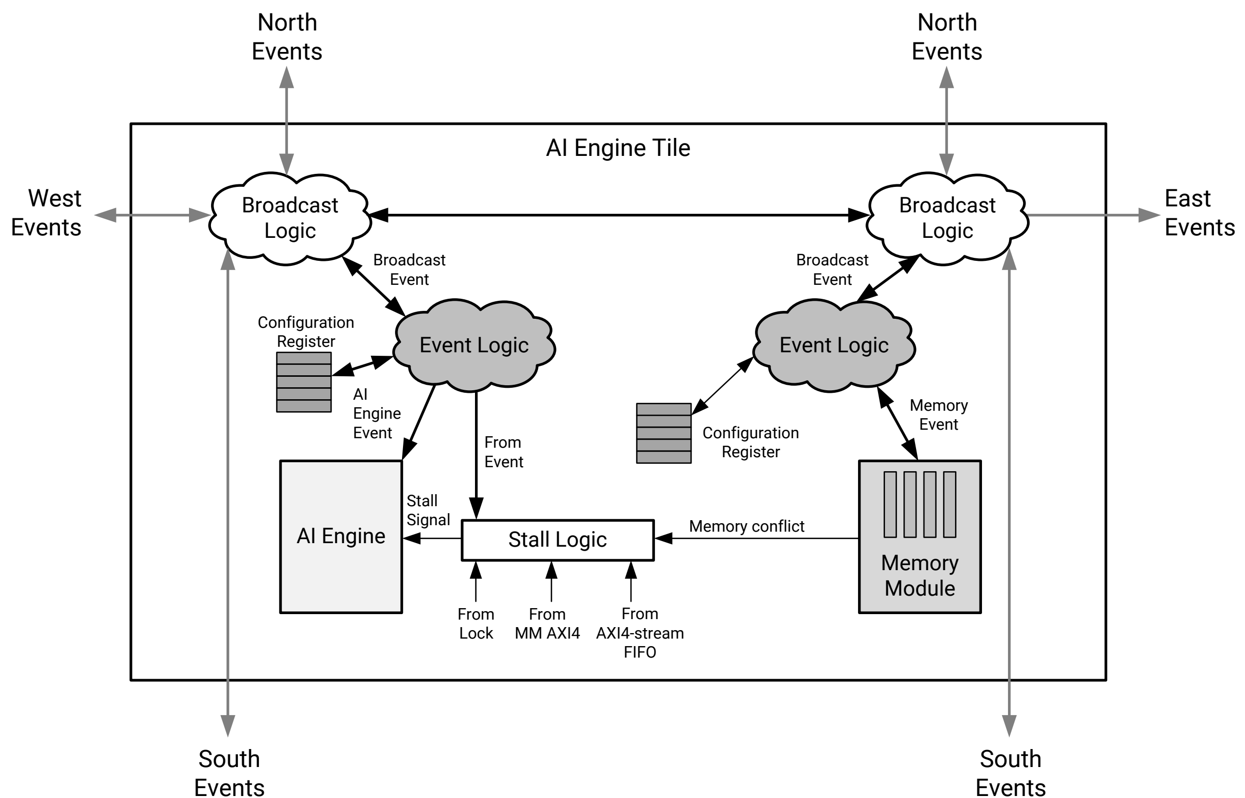
Figure 15: **Logical View of AI Engine Profiling**



AI Engine Events

The AI Engine 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 AI Engine 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 is event logic hardware that only interacts with the AI Engine and memory modules. Event actions can be configured to perform a task whenever a specific event occurs. There are two separate sets of configuration registers for event hardware. Also, there is separate broadcast logic to send event signals to neighboring modules.

Figure 16: **Events in an AI Engine 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 AI Engine
- Debug-halt, resume, or single-step of an AI Engine

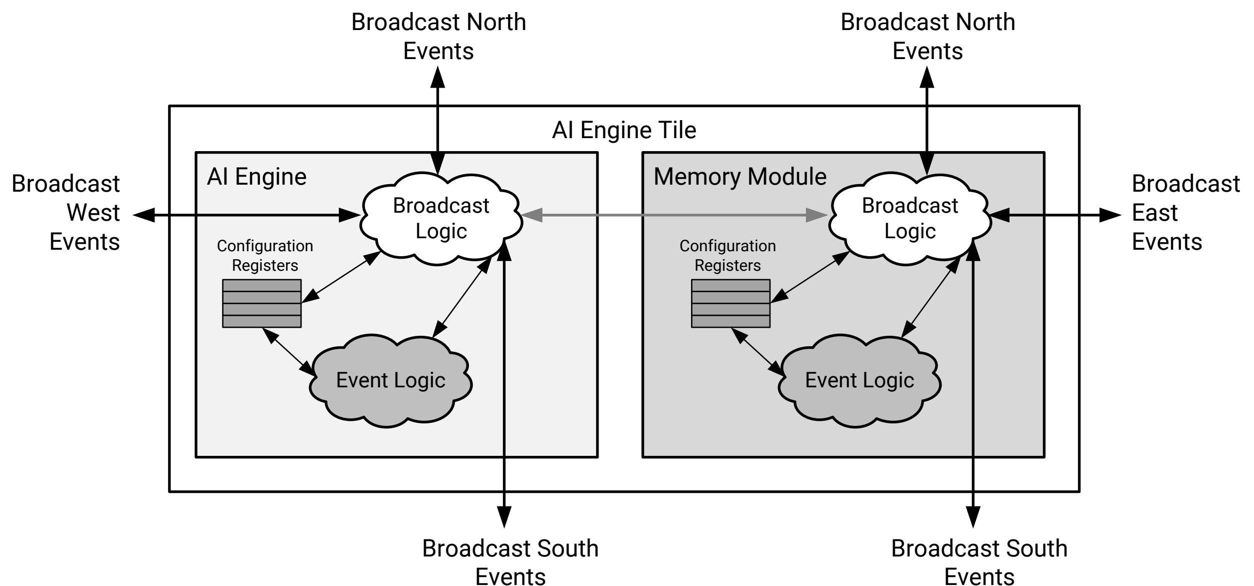
- Error halt of an AI Engine
- Resynchronize timer
- Start and stop performance counters
- Start and stop trace streams
- Generate broadcast events
- Drive combo events
- ECC scrubbing event

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. The full list of the event action configuration registers for the AI Engine and memory modules are found in the *Versal ACAP AI Engine Register Reference* (AM015).

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 AI Engine tile. The units in the broadcast logic in the AI Engine 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 AI Engine tile.

Figure 17: **AI Engine 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

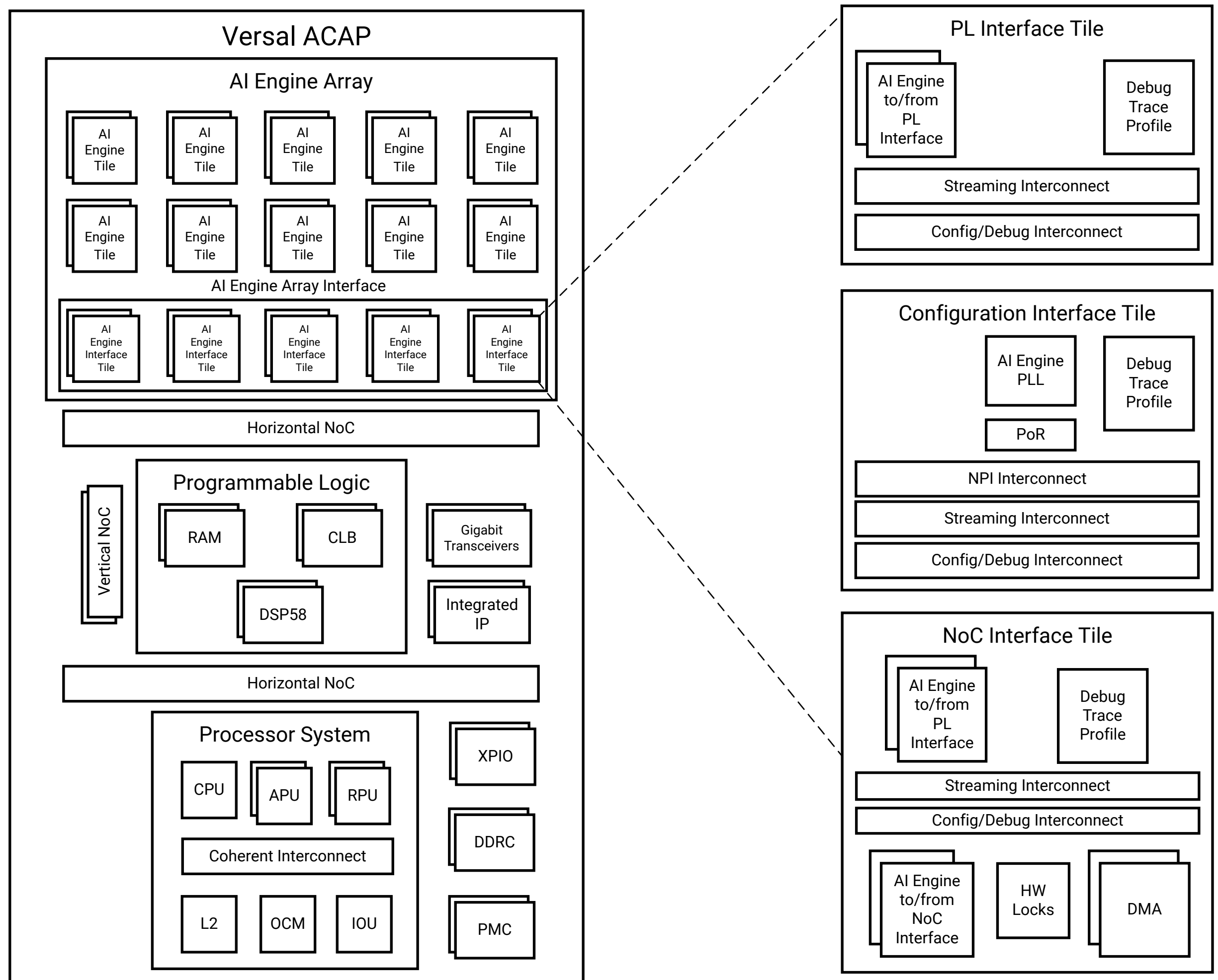
TIP: *The AI Engine module east broadcast event interface is internally connected to the memory module west broadcast event interface and does not go out of the AI Engine tile. In the AI Engine module, there are 16 broadcast events each in the north, south, and west directions. In the memory module, there are 16 broadcast events each in the north, south, and east directions. With respect to broadcast events and from the programmer's perspective, there are no differences in the interface between odd and even rows.*

Chapter 3

AI Engine Array Interface Architecture

The AI Engine is arranged in a 2D array as shown in the following figure. The AI Engine array interface provides the necessary functionality to interface with the rest of the device. The AI Engine array interface has three types of AI Engine interface tiles. There is a one-to-one correspondence of interface tiles for every column of the AI Engine 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 AI Engine tile column. The AI Engine 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 AI Engine array interface in the AI Engine array.

Figure 18: **AI Engine Array Interface Hierarchy**



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

- AI Engine PL interface tile
  - PL module includes:
    - AXI4-Stream switch
    - Memory-mapped AXI4 switch
    - AI Engine to PL stream interface
    - Control, debug, and trace unit
- AI Engine configuration interface tile (exactly one instance per AI Engine array)
  - PLL for AI Engine clock generation
  - Power-on-reset (POR) unit
  - Interrupt generation unit
  - Dynamic function exchange (DFx) logic

- NoC peripheral interconnect (NPI) unit
- AI Engine array global registers that control global features such as PLL/clock control, secure/non-secure behavior, interrupt controllers, global reset control, and DFx logic

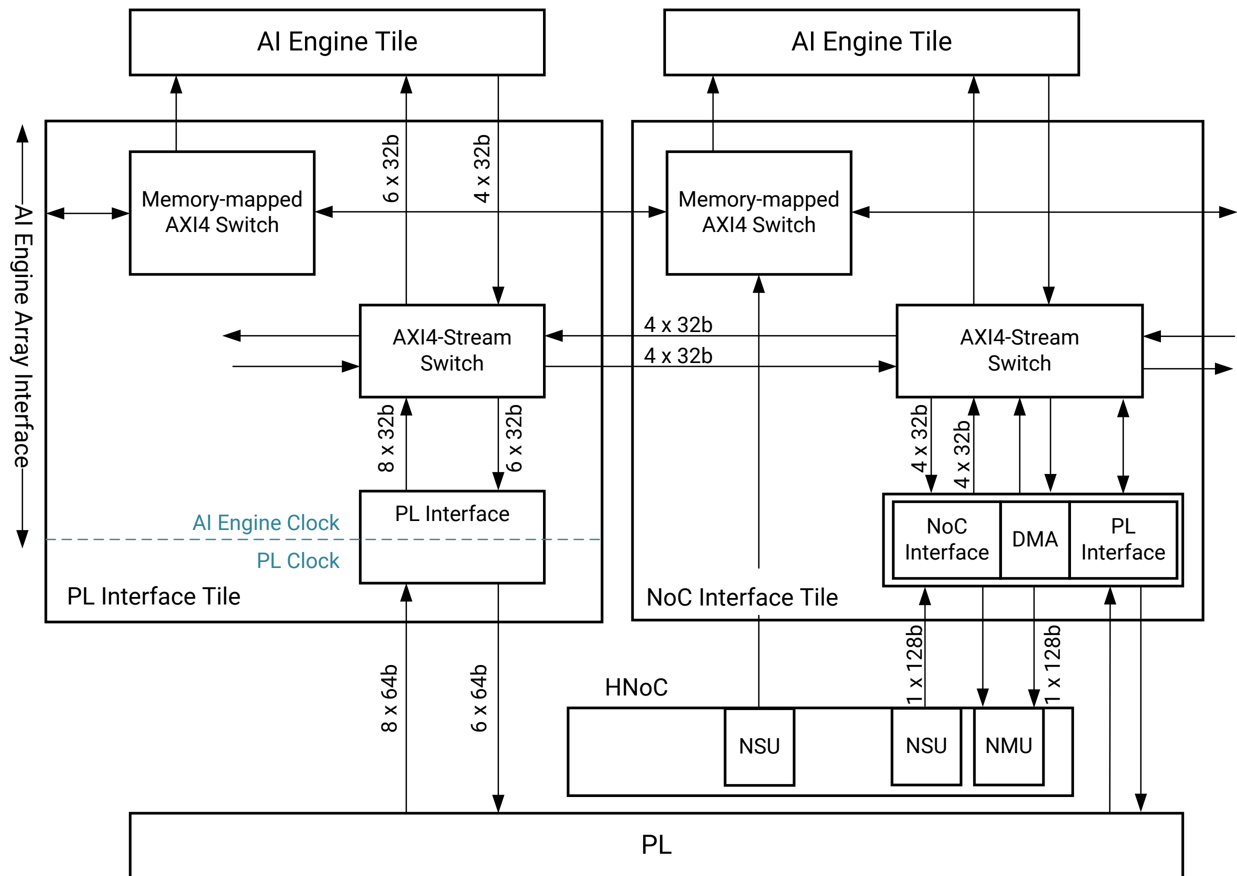
- AI Engine NoC interface tile
  - PL module (see previous description)
  - NoC module with interfaces to NMU and NSU includes:
    - Bi-directional NoC streaming interface
    - Array interface DMA

AI Engine Array Interface

The AI Engine 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 AI Engine 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 AI Engine 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 AI Engine array.*

Figure 19: **AI Engine Array Interface Topology**



***Note*:** The AI Engine 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 AI Engine and the NoC.

The types of interfaces to the PL and NoC are:

- Memory-mapped AXI4 interface: the communication channel is from the NSU to the AI Engine as a slave
- AXI4-Stream interconnect has three types of interfaces:
  - 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 AI Engine array interface tiles manage the two high performance interfaces:

- AI Engine to PL
- AI Engine to NoC

The following tables summarize the bandwidth performance of the AI Engine array interface with the PL, the NoC, and the AI Engine tile. The bandwidth performances are specified per each AI Engine column for the -1L speed grade devices. There is a reduction in the number of connections per column between the PL to AI Engine interface and the AXI4-Stream switch to the AI Engine 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).

Table 3: **AI Engine 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 AI Engine array interface	8	64	PL (500 MHz)	4	32
AI Engine array interface to PL	6	64	PL (500 MHz)	4	24
AI Engine array interface to AXI4-Stream switch	8	32	AI Engine (1 GHz)	4	32
AXI4-Stream switch to AI Engine array interface	6	32	AI Engine (1 GHz)	4	24
Horizontal interface between AXI4-Stream switches[1]	4	32	AI Engine (1 GHz)	4	16

Notes:

1. 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 4: **AI Engine Array Interface to NoC Interface Bandwidth Performance**

Connection Type	Number of Connections	Data Width (bits)	Clock Domain	Bandwidth per Connection (GB/s)	Aggregate Bandwidth (GB/s)
AI Engine to NoC (NoC side)	1	128	NoC Interface (1 GHz)	16	16
AI Engine to NoC (AI Engine side)	4	32	AI Engine (1 GHz)	4	16
NoC to AI Engine (NoC side)	1	128	NoC Interface (1 GHz)	16	16
NoC to AI Engine (AI Engine side)	4	32	AI Engine (1 GHz)	4	16

Table 5: **AI Engine Array Interface to AI Engine 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 AI Engine tile	6	32	AI Engine (1 GHz)	4	24
AI Engine tile to AXI4-Stream switch	4	32	AI Engine (1 GHz)	4	16

The following sections contain additional AI Engine array interface descriptions. The AI Engine tiles are described in the Chapter 2: AI Engine Tile Architecture chapter.

Features of the AI Engine Array Interface

- **Memory Mapped AXI4 Interconnect:** Provides functionality to transfer the incoming memory-mapped AXI4 requests from the NoC to inside the AI Engine array.
- **AXI4-Stream Interconnect:** Leverages the AI Engine tile streaming interconnect functionality.
- **AI Engine to PL Interface:** The AI Engine PL modules directly communicate with the PL. Asynchronous FIFOs are provided to handle clock domain crossing.
- **AI Engine to NoC Interface:** The AI Engine to NoC module handles the conversion of 128-bit NoC streams into 32-bit AI Engine 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 AI Engine.
- **Hardware Locks:** Leverages the corresponding unit in the AI Engine tile and is accessible from the AI Engine 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.
- **Debug, Trace, and Profile:** Leverages all the features from the AI Engine tile for local event debugging, tracing, and profiling.

Array Interface Memory-Mapped AXI4 Interconnect

The main task of the AI Engine memory-mapped AXI4 interconnect is to allow external access to internal AI Engine 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 AI Engine array. The memory-mapped AXI4 interfaces are all interconnected across the AI Engine 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 AI Engine 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.

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 AXI4-Stream Interconnect

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

AI Engine to Programmable Logic Interface

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

- 24 GB/s from each AI Engine column to PL
- 32 GB/s from PL to each AI Engine column

In the VC1902 device, there are 50 columns of AI Engine tiles and AI Engine array interface tiles, however, only 39 array interface tiles are available to the PL interface. Therefore, the aggregate bandwidth for PL interface is approximately:

- 1.0 TB/s from AI Engine to PL
- 1.3 TB/s from PL to AI Engine

All bandwidth calculations assume a nominal 1 GHz AI Engine clock for the -1L speed grade devices at V_{CCINT} = 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).

AI Engine to NoC Interface

The AI Engine 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 AI Engine, there are four streams from the AI Engine to the NoC, and four streams from the NoC to the AI Engine. From a bandwidth perspective each AI Engine 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 AI Engine array. This section gives an introduction to the types of interrupts from the AI Engine array.

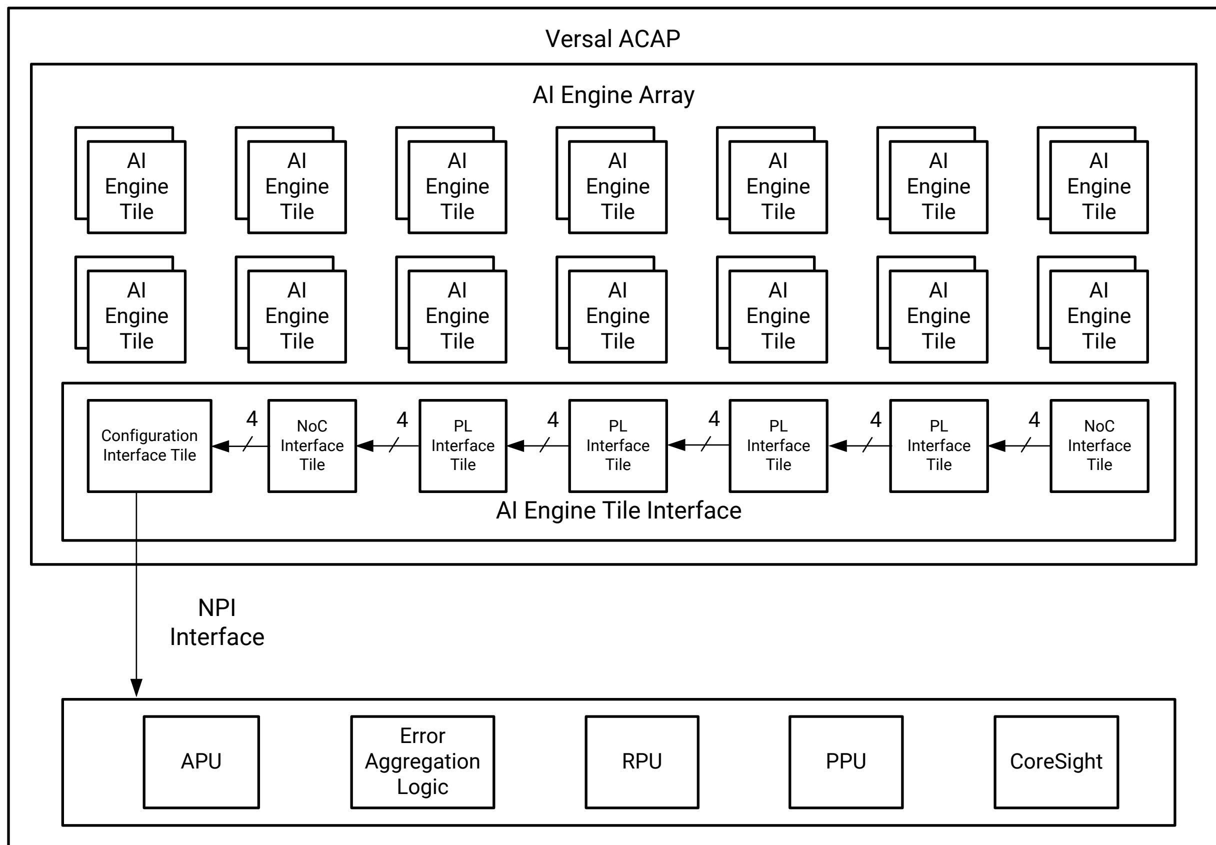
The AI Engine array generates four interrupts that can be routed from the AI Engine array to the PMC, application processing unit (APU), and real-time processing unit (RPU). The overall hierarchy for interrupt generation from AI Engine array is as follows:

- Events get triggered from any of the AI Engine tiles or AI Engine 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 AI Engine array interface.

- These four interrupt lines are eventually connected to the AI Engine configuration interface tile.

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

Figure 20: **Connecting Interrupts from the AI Engine 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. The structure and address of the NPI registers are described in the *Versal ACAP AI Engine Register Reference* (AM015).

XILINX

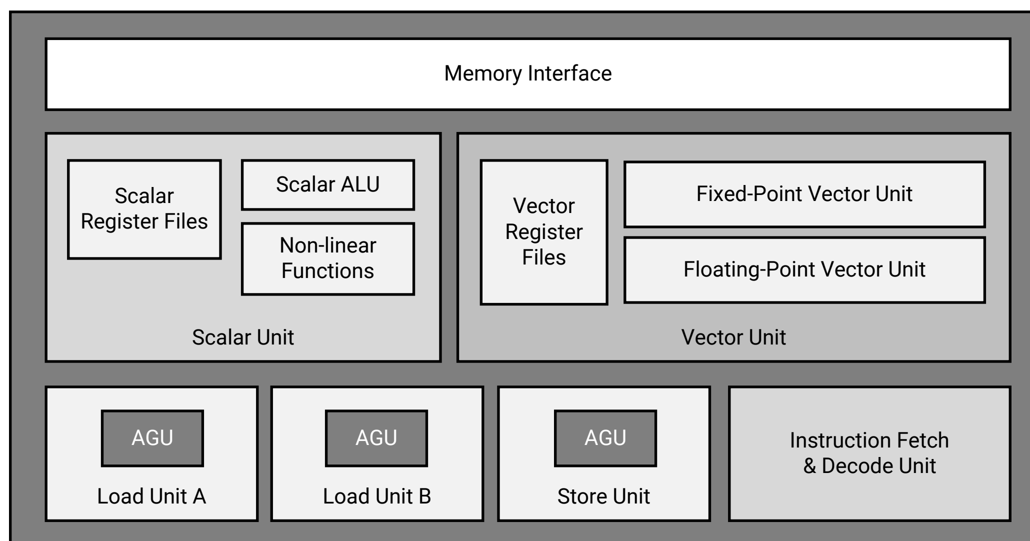
Chapter 4

AI Engine Architecture

Functional Overview

The AI Engine 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 AI Engine has a memory interface, a scalar unit, a vector unit, two load units, one store unit, and an instruction fetch and decode unit.

Figure 21: **AI Engine**



The features of the AI Engine include:

- 32-bit scalar RISC processor
  - General purpose pointer and configuration register files
  - Supports non-linear functions (for example: sqrt, Sin/Cos, and InvSqrt)
  - A scalar ALU, including 32 x 32-bit scalar multiplier
  - Supports conversion of the data type between scalar fixed point and scalar floating point

- Three address generator units (AGU)
  - Support for multiple addressing modes: Fixed, indirect, post-incremental, or cyclic
  - Supports Fast Fourier Transform (FFT) address generation
  - Two AGUs dedicated for two load units
  - One AGU dedicated for the store unit
- Vector fixed-point/integer unit
  - Concurrent operations on multiple vector lanes
  - Accommodate multiple precision for complex and real operand (see Table 6)

Table 6: **Supported Precision Width of the Vector Data Path**

X Operand	Z Operand	Output	Number of MACs
8 real	8 real	48 real	128
16 real	8 real	48 real	64
16 real	16 real	48 real	32
16 real	16 complex	48 complex	16
16 complex	16 real	48 complex	16
16 complex	16 complex	48 complex	8
16 real	32 real	48/80 real	16
16 real	32 complex	48/80 complex	8
16 complex	32 real	48/80 complex	8
16 complex	32 complex	48/80 complex	4
32 real	16 real	48/80 real	16
32 real	16 complex	48/80 complex	8
32 complex	16 real	48/80 complex	8
32 complex	16 complex	48/80 complex	4
32 real	32 real	80 real	8
32 real	32 complex	80 complex	4
32 complex	32 real	80 complex	4
32 complex	32 complex	80 complex	2
32 SPFP	32 SPFP	32 SPFP	8

  - Can be configured to perform eight complex 16-bit multiplications
  - Full permute unit with 32-bit granularity
  - Shift, round, and saturate with multiple rounding and saturation modes
  - Two-step post adding along with 768-bit intermediate results

    - The X operand is 1024 bits wide and the Z operand is 256 bits wide. In terms of component use, consider the first row in Table 6. The multiplier operands come from the same 1024-bit and 256-bit input registers, but some values are broadcast to multiple multipliers. There are 128 8-bit single multipliers and the results are post-added and accumulated into 16 or 8 accumulator lanes of 48 bits each.
- Single-precision floating-point (SPFP) vector unit
    - Use same permute as a fixed-point vector unit
    - Concurrent operation of multiple vector lanes
    - Eight single-precision multiplier–accumulators (MACs) per cycle
- 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 or 32-bit/16-bit/8-bit scalar register mode. 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 seven operations can be issued in parallel using one VLIW word
- Direct stream interface
    - Two input streams and two output streams
    - Each stream can be configured to be either 32-bit or 128-bit wide
    - One cascade stream in, one cascade stream out (384-bit)
- Interface to the following modules
    - Lock module
    - Stall module
    - Debug and trace module
- Event interface is a 16-bit wide output interface from the AI Engine

Register Files

The AI Engine 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 7: **Scalar Registers**

Syntax	Number of bits	Description
r0..r15	32 bits	General-purpose registers
m0..m7	20 bits	Modifier registers
p0..p7	20 bits	Pointer registers
cl0..cl7	32 bits	Configuration registers
ch0..ch7		
c0..c7	64 bits	

Special Registers

Table 8: **Special Registers**

Syntax	Number of bits	Description
cb0..cb7	20 bits	Circular buffer start address
cs0..cs7	20 bits	Circular buffer size
wcs0..wcs3	40 bits	Wide circular buffer size
s0..s7	8 bits	Shift control
sp	20 bits	Stack pointer
lr	20 bits	Link register
pc	20 bits	Program counter
fc	20 bits	Fetch counter
mc0..mc1	32 bits	Status register
md0..md1	32 bits	Mode control register
ls	20 bits	Loop start
le	20 bits	Loop end
lc	32 bits	Loop count
lci	32 bits	Loop count (PCU)
S	8 bits	Shift control

Vector Registers

Vector registers are high-width registers to allow SIMD instructions. The underlying basic hardware registers are 128-bit wide, prefixed with the letter V. Two V registers can be grouped to form a 256-bit register prefixed with W. WR, WC, and WD registers are grouped in pairs to form 512-bit registers (XA, XB, XC, and XD). XA and XB form the 1024-bit wide YA registers. For all the registers except YD, the order is LSB from the top of the table to MSB at the bottom of the table. For YD, the LSBs are from the XD, and the MSBs are from the XB, that is:

YD = VDL0::VDH0::VDL1::VDH1::VRL2::VRH2::VRL3::VRH3

Table 9: **Vector Registers**

128-bit	256-bit	512-bit	1024-bit	
vrl0	wr0	xa	ya	N/A
vrh0				
vrl1	wr1			
vrh1				
vrl2	wr2	xb		yd (MSBs)
vrh2				
vrl3	wr3			
vrh3				
vcl0	wc0	xc	N/A	N/A
vch0				
vcl1	wc1			
vch1				
vdl0	wd0	xd	N/A	yd (LSBs)
vdh0				
vdl1	wd1			
vdh1				

Accumulator Registers

Accumulator registers are used to store the results of the vector data path. They are 384-bit wide which can be viewed as 8 vector lanes of 48-bit each. The idea is to have 32-bit multiplication results and accumulate over those results without bit overflows. The 16 *guard bits* allow up to 2^{16} accumulations. The accumulator registers are prefixed with the letters AM. Two of them are aliased to form a 768-bit register that is prefixed with BM.

***Note*:** There are two modes of operation. In the first mode, the multiplication results are post-added into 8 accumulators using 16 post additions before the accumulation. In the second mode, the multiplication results are post-added into 16 accumulators using 8 post additions before the accumulation.

Table 10: **Accumulator Registers**

384-bit	768-bit
aml0	bm0
amh0	
aml1	bm1
amh1	
aml2	bm2
amh2	
aml3	bm3
amh3	

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 AI Engine. The program memory size on the AI Engine is 16 KB, which allows storing 1024 instructions of 128-bit each.

The AI Engine instructions are 128-bits wide 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.

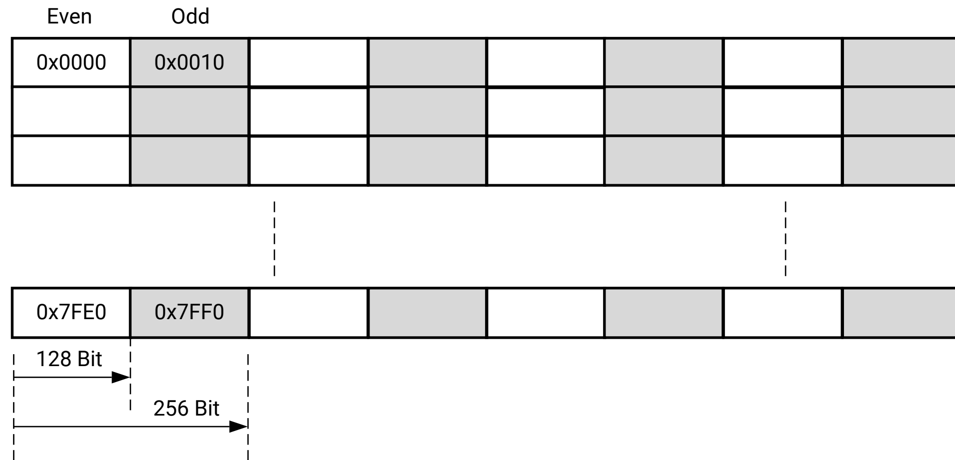
Load and Store Unit

The AI Engine 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 32 KB. The AI Engine accesses four 32 KB data memory blocks to create a 128 KB unit. These four memory blocks are located on each side of the AI Engine and are divided and interleaved as odd and even banks (see the following figure).

Figure 22: **Interleaving in Data Memory (32 KB per Block)**

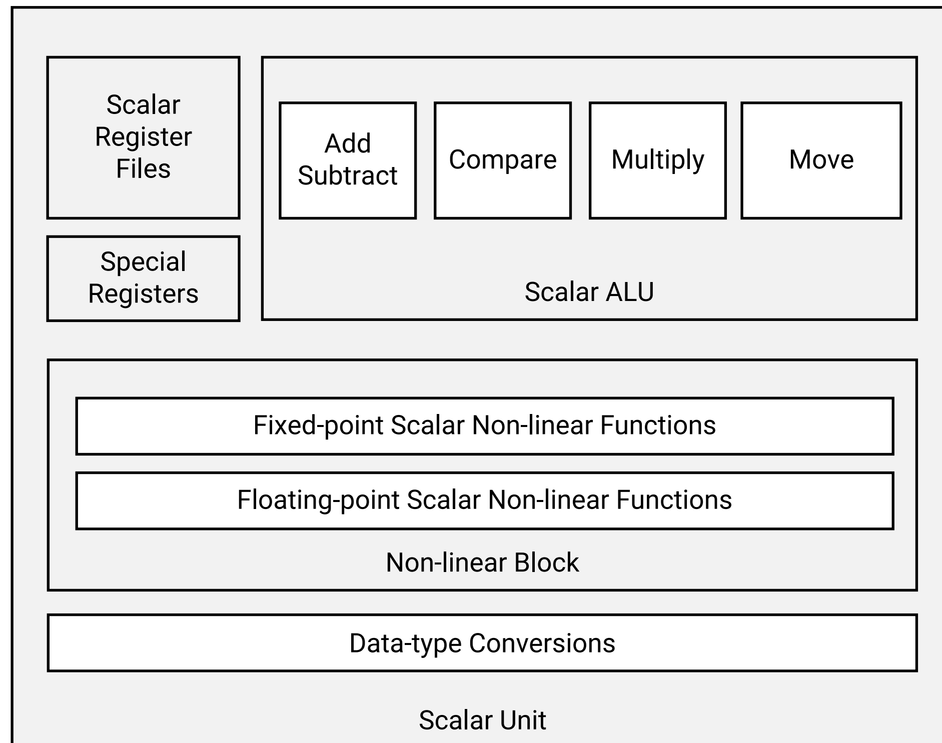


In a logical representation the 128 KB memory can be viewed as one contiguous 128 KB block or four 32 KB blocks, and each block can be divided into odd and even banks. The memory can also be viewed as eight 16 KB banks (four odd and four even). The AGU generates addresses for data memory access that span from `0x0000` to `0x1FFFF` (128 KB).

Scalar Unit

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

Figure 23: **AI Engine Scalar Unit**



The scalar unit contains the following functional blocks.

- Register files and special registers
- Arithmetic and logical unit (ALU)
- Non-linear functions – fixed-point and floating-point precision
- Data type conversions

Integer add, subtract, compare, and shift functions are one-cycle operations. The integer multiplication operation has a three-cycle latency. Non-linear functions take one or four cycles to produce scalar results. The throughput of the aforementioned operations is one cycle.

Arithmetic Logic Unit, Scalar Functions, and Data Type Conversions

The arithmetic logic unit (ALU) in the AI Engine 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 three 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 are two types of scalar elementary functions in the AI Engine: fixed-point and floating-point. The following describes each function.

- Fixed-point non-linear functions
  - Sine and cosine:
    - Input is from the upper 20 bits of a 32-bit input
    - Output is a concatenated word with the upper 16-bit sine and the lower 16-bit cosine
    - The operations have a four cycle latency
  - Absolute value (ABS): Invert the input number and add one. The operation has a one cycle latency
  - Count leading zeroes (CLZ): Count leading zeroes in a 32-bit input. The operation has a one cycle latency
  - Minimum/maximum (lesser than (LG)/greater than (GT)): Two inputs are compared to find the minimum or maximum. The operation has a one cycle latency
  - Square Root, Inverse Square Root, and Inverse: These operations are implemented with floating point precision. For fixed point implementation the input needs to be first converted to floating-point precision and then passed as input to these non-linear operations. Also, the output is in a floating-point format that needs to be converted back to fixed-point integer format. The operations have a four cycle latency
- Floating-point non-linear functions
  - Square root: Both input and output are single precision floating point numbers operating on R register file. The operation has a four cycle latency
  - Inverse square root: Both input and output are single precision floating point numbers operating on R register file. The operation has a four cycle latency
  - Inverse: Both input and output are single precision floating point numbers operating on R register file. The operation has a four cycle latency
  - Absolute value (ABS), the operation has a one cycle latency
  - Minimum/maximum, the operation has a one cycle latency

The AI Engine scalar unit supports data-type conversion operations to convert input data from a fixed-point to a floating-point and from a floating-point to a fixed-point. The `fix2float` operation and `float2fix` operation provide support for variable decimal point where input is a 32-bit value. Along with the input, the decimal point position is also taken as another input. The operations scale the value up or down, if required. Both operations have a one cycle latency.

The AI Engine floating-point is not completely compliant with the IEEE standards and there are restrictions on a set of functionality. The exceptions are outlined in this section.

- When the `float2fix` function is called with a very large positive or negative number and the additional exponent increment is greater than zero, the instruction returns 0 instead of the correct saturation value of either $2^{31}-1$ or -2^{31}.

  The `float2fix` function takes two input parameters:

  - n: The floating point input value to be converted.
  - sft: A 6-bit signed number representing the number of fractional bits in fixed-point representation (from −32 to 31).

  Consider the two scenarios:

  - If $n*2^{sft} > 2^{129}$, the output should return `0x7FFFFFFF`. Instead, it returns `0x00000000`.
  - If $n*2^{sft} < -2^{129}$, the output should return `0X80000000`. Instead, it returns `0x00000000`.

  In general, you should ensure that the `n` floating-point input value stays in the bug-free range of $-2^{(129-sft)} < n < 2^{(129-sft)}$ for sft > 0.

  Two implementations are introduced to provide a workaround:

  - `float2fix_safe`: This is the default mode if you specify `float2fix` without any option. The implementation returns the correct value for any range, but is slower.
  - `float2fix_fast`: This implementation returns the correct value only in the bug-free range and you need to ensure the range is valid. To choose the `floatfix_fast` implementation, you need to add the preprocessor FLOAT2FIX_FAST to the project file.

- A fixed-point value has a legal range of -2^{31} to $2^{32}-1$. When the `float2fix` function returns a value of -2^{31}, the value is within range but an overflow exception is incorrectly set. There is no workaround for this overflow exception.

Vector Unit

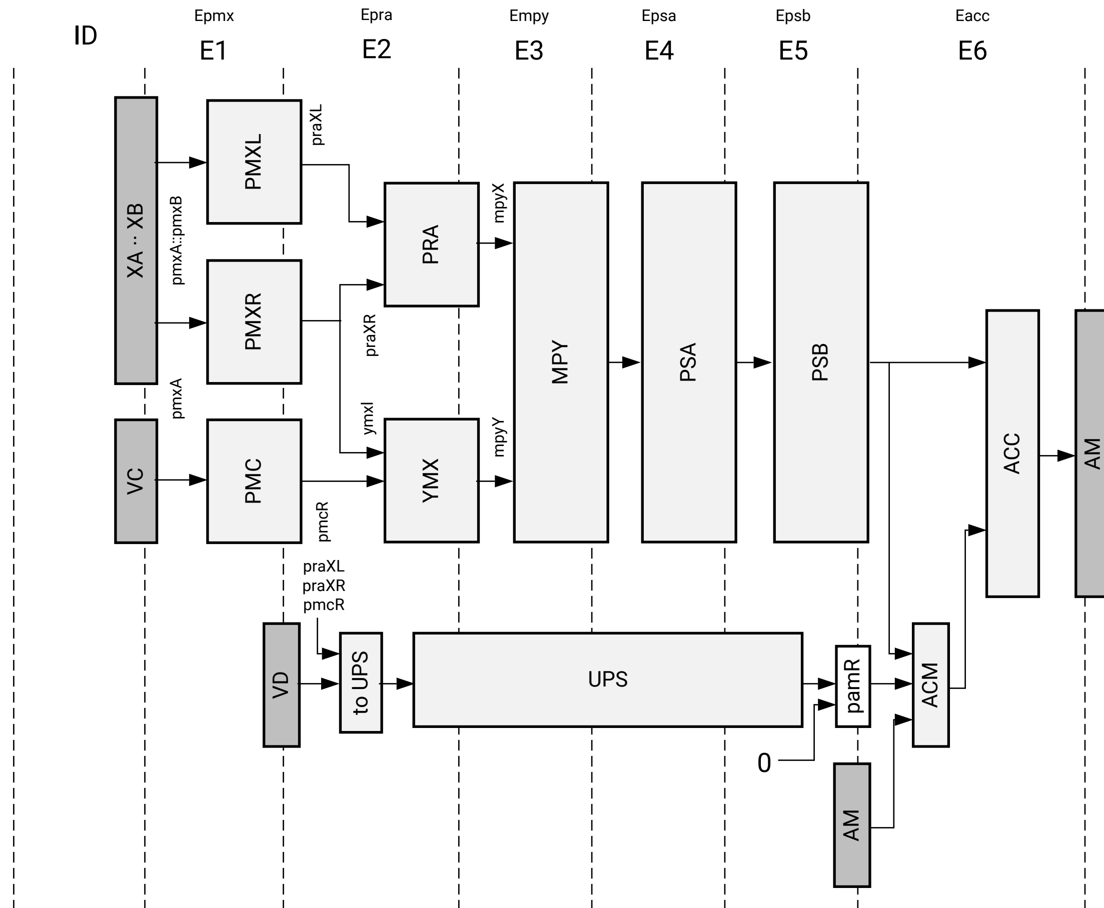
Fixed Point Vector Unit

The fixed-point vector unit contains three separate and largely independent data paths.

- **Multiply Accumulator (MAC) Path:** The main multiplication path reads values from vector registers, permutes them in a user controllable manner, performs optional pre-adding, multiplies them, and after some post-adding, accumulates them to the previous value of the accumulator register.
- **Upshift Path:** The path runs in parallel to the MAC path. It reads data from the permute units in the MAC path or from the vector register, left-shifts, and feeds it to the accumulator registers.
- **Shift-round Saturate (SRS) Path:** This path reads from the accumulator registers and stores to the vector registers or the data memory. It is needed because the accumulators are 48 or 80 bits wide per lane and the vector registers and the data memory have 8, 16, 32, or 64-bit power-of-two widths. Therefore, the data needs to be right shifted on a lane-by-lane basis.The SRS unit uses the saturation and rounding control register MD with its fields Q and R to influence its behavior, and the status register MC to provide information back to the environment. The shift control register S determines the shift amount. The unit supports various rounding modes based on the value of the field R in register MD. If R is set to 0, the value is truncated on the LSB side. If R is set to 1, a ceiling behavior is achieved, which means that there is no actual rounding. For R = 2 to 7, the modes are PosInf, NegInf, SymInf, SymZero, ConvEven, and ConvOdd (respectively).

The following figure is the pipeline diagram of the main multiplication and upshift path. After the instruction decode stage (ID), the six execute stages are numbered E1 to E6. The dark gray boxes, which always cross two stages, are registers. The light gray boxes, that can span multiple stages, are the functional units. The white box represents hardware registers that are internal to the processor description. Between all boxes there are arrow connectors. They are nML *transitories*, which are pure non-storing wires. In addition to the elements shown in the diagram, there are multiplexers that realize different connectivity depending on the instruction that is executed. The *to UPS* unit implies a multiplexer that selects among the three permute units and the VD register. There is an internal unit that reads the inputs and pre-adds two values before outputting the data to the UPS unit.

Figure 24: **Pipeline Diagram of AI Engine Fixed-point Vector Unit Multiplication and Upshift Paths**



The following table shows the functional units in the main multiplication path.

Table 11: **Functional Units in the Multiplication Path**

Functional Unit	Description
Permute Units	
PMXL	Permutes the data from the vector registers for the left input of the pre-adder PRA.
PMXR	Permutes the data from the vector registers for the right input of the pre-adder PRA or alternatively for the input of the YMX unit.
PMC	Permutes the data from the vector registers for the input of the YMX unit.
Pre-adder Units	
PRA	Pre-add or pre-subtract the PMXL and PMXR outputs to form the first multiplier argument. Additionally, some restricted permute takes place to compensate for the 32-bit granularity of the PMXL/PMXR units.
YMX	Special operations, such as inserting the constant 1 to multiply the pre-adder result with a 1, or sign extending the individual lanes. Additionally, some restricted permute takes place to compensate for the 32-bit granularity of the PMXR unit.

Table 11: **Functional Units in the Multiplication Path** *(cont'd)*

Functional Unit	Description
Multiplier Unit	
MPY	Multiplies the PRA and YMX outputs.
Post-adder Units	
PSA	First post-adding stage that reduces the 32-MPY output lanes to 16 lanes.
PSB	Second post-adding stage that further reduces the lanes to 8. Alternatively, it forwards the inputs to the output.
Accumulator	
ACM	Multiplexes the data that is to be added to the post-adder output. Can be the old accumulator value, the output of the upshift path, or the cascade stream input.
ACC	Adds or subtracts the ACM and PSB outputs.

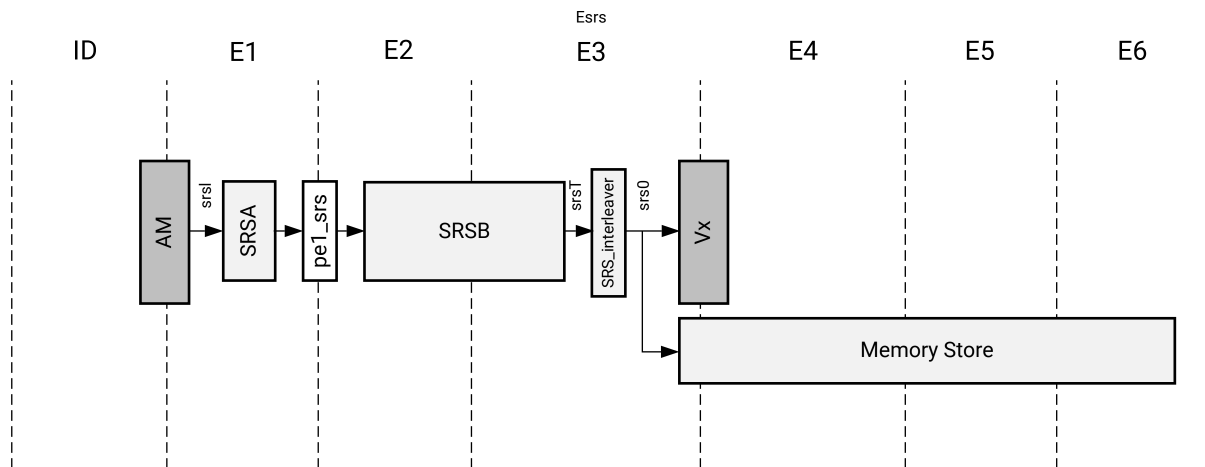
The following table shows the functional units in the upshift path.

Table 12: **Upshift Path**

Upshift Units	Description
to UPS	Reads vector register and selects only certain lanes.
UPS	Perform the actual upshifting and output to the ACM unit in the main data path.

The following figure is a pipeline diagram of the shift-round-saturate path. An accumulator register is read, the shift-round-saturate operation occurs, and the output is either written into any vector register or to the data memory. The value is stored in memory in the E3 stage and arrives in memory in the E6 stage.

Figure 25: **Pipeline Diagram of AI Engine Shift-Round-Saturate Data Path**



The following table shows the functional units in the shift-round-saturate path.

Table 13: **Shift-Round-Saturate Path**

Shift-Round-Saturate Units	Description
SRSA	Performs the combination of the two parts of an 80-bit accumulator. It bypasses the data when 48-bit accumulators are to be shifted. Works on eight 48-bit lanes or four 80-bit lanes in parallel. The functionality is split into a low and high part to perform the same operation in parallel.
SRSB	Perform the actual shifting of the lanes. The functionality is split into a low and high part to perform the same operation in parallel.
SRS Interleaver	Interleaves the outputs of the SRSB high and low units when accumulator interleaving is required (controlled by the MSB of the shift amount register S).

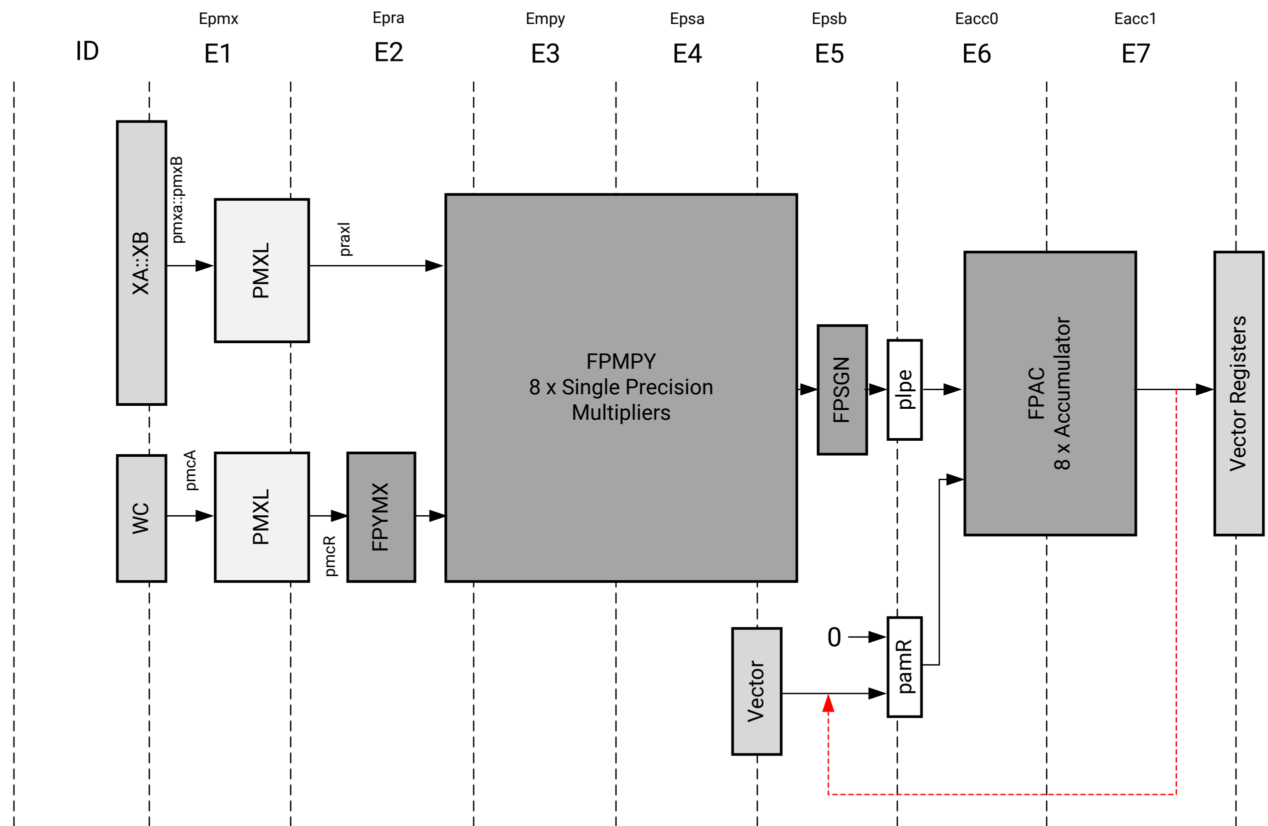
Floating Point Vector Unit

The AI Engine provides eight lanes of single-precision floating-point multiplication and accumulation. The unit reuses the vector register files and permute network of the fixed-point data path. In general, only one vector instruction per cycle can be done in fixed point or floating point.

The following figure shows the pipeline diagram of the single precision floating-point data flow. Compared to the fixed-point vector unit, only the PMXL and PMC units are used (the PMXR unit is removed). FPYMX is in the style of YMX and the results from FPYMX and PMXL are forwarded to a single-precision multiplier unit (FPMPY) that can compute eight products in parallel. The operation in FPMPY has a three-cycle latency and a one-cycle throughput. Next, there is an FPSGN unit that allows sign negation of the results on a per-lane basis.

After the FPSGN unit there is a two-stage accumulator unit called FPACC. It accumulates the multiplication results with values from various sources, such as zeroes or values directly from another vector register. However, it is not possible to add lanes within the same vector directly. The accumulator does not support subtraction as it is handled by the FPSGN unit.

Figure 26: **Pipeline Diagram of the AI Engine Floating-point Vector Unit Single-precision Floating-point Data Path**



The AI Engine supports several vector elementary function for the floating-point format. These functions include a vector comparison, minimum, and maximum. The floating-point data path supports a vector fixed-point to single precision floating-point conversion as well as a reverse operation of a floating-point to fixed-point conversion, but only at a lower performance through the scalar unit. In that situation, extract elements extracted from the vector, perform the scalar conversion, and push the results back into a vector. When implemented in an efficiently pipelined loop, close to one sample per cycle conversion performance can be achieved.

Some features are not supported by the AI Engine floating-point data path.

- Double-precision operations
- Half-precision operations
- Custom floating-point formats, for example 2-bit exponent, and 14-bit mantissa (E2:M14)
- Pre-adding before multiplication
- Post-adding between multiplication and accumulator
- Increased precision between multiplier and accumulator

- Denormalized and subnormal floating-point numbers

Register Move Functionality

The register move capabilities of the AI Engine 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, C, and special registers
  - Move immediate values to R, M, P, C, 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 384-bit accumulator (AM) register to another AM-register in one cycle
- Vector to accumulator; there are two possibilities:
  - Upshift 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`
- Accumulator to vector: Shift-round saturate datapath moves the accumulator to a vector register
- Accumulator to cascade stream and cascade to accumulator: Cascade stream connects the AI Engines in the array in a chain and allows the AI Engines to transfer an accumulator register (384-bit) from one to the next. A small two-deep 384-bit wide FIFO on both the input and output streams allows storing up to four values in the FIFOs between the AI Engines
- Scalar to vector: Moves a scalar value from an R-register to a vector register
- Vector to scalar: Extracts an arbitrary 8-bit, 16-bit, or 32-bit value from a 128-bit or 256-bit vector register and writes results into a scalar R-register

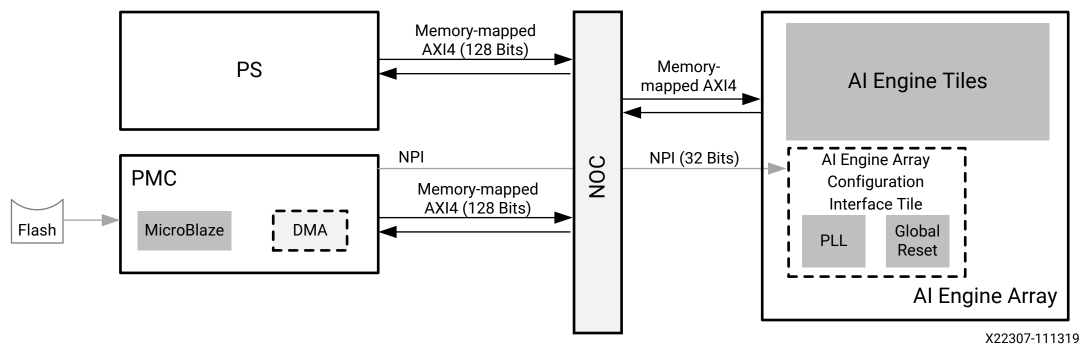
Chapter 5

AI Engine Configuration and Boot

AI Engine Array Configuration

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

Figure 27: **AI Engine Array Configuration using NoC and NPI**



Any memory-mapped AXI4 master can configure any memory-mapped AXI4 register in the AI Engine 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.

AI Engine Boot Sequence

This section describes the steps involved in the boot process for the AI Engine array.

1. Power-on and power-on-reset (POR) deassertion: Power is turned on for all modules related to the AI Engine 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 AI Engine boot sequence is initiated. After the array power is turned on, the PMC can deassert a POR signal in the AI Engine array.
2. AI Engine array configuration using NPI: After power-on, the PMC uses the NPI interface to program the different global registers in the AI Engine array (for example, the PLL configuration registers). The AI Engine configuration image that is required over the NPI for AI Engine 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 ACAP Technical Reference Manual* (AM011).
4. Release reset: Once the PLL is locked, software can program a register to deassert the global reset signal for the AI Engine array.
5. AI Engine array programming: The AI Engine array interface needs to be configured over the memory-mapped AXI4 from the NoC interface. This includes all AXI4 stream switches, memory-mapped AXI4 switches, array interface DMAs, event, and trace configuration registers.

AI Engine Array Reconfiguration

The AI Engine configuration process writes a programmable device image (PDI) produced by the bootgen tool into AI Engine configuration registers. The AI Engine configuration is done over memory-mapped AXI4 via the NoC. Any master on the NoC can configure the AI Engine array. For more information on generating a PDI with the bootgen tool, refer to *Versal ACAP AI Engine Programming Environment User Guide* (UG1076).

The AI Engine 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 AI Engine 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.

Two scenarios are described for AI Engine array reconfiguration:

- **Complete reconfiguration:** The global reset is asserted for the AI Engine array and the entire array is reconfigured by downloading a new configuration image.

- **Partial reconfiguration:** Some of the AI Engine tiles in the array are reconfigured while the rest of the tiles continue to run kernels. Reconfiguration occurs without affecting already running kernels in the AI Engine array.

The PMC and PS are responsible for initializing the AI Engine array. The following table summarizes the reset controls available for the global AI Engine array.

Table 14: **Categories of AI Engine Resets**

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

The combination of column reset and array interface tile reset (refer to AI Engine Array Hierarchy) enables a partial reconfiguration use case where a sub-array that comprises AI Engine 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).

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

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

Xilinx 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.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

References

These documents provide supplemental material useful with this guide:

1. Versal ACAP data sheets:
   - *Versal Architecture and Product Data Sheet: Overview* (DS950)
2. *Versal ACAP AI Engine Register Reference* (AM015)
3. *Versal ACAP AI Engine Programming Environment User Guide* (UG1076)

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

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.

This document contains preliminary information and is subject to change without notice. Information provided herein relates to products and/or services not yet available for sale, and provided solely for information purposes and are not intended, or to be construed, as an offer for sale or an attempted commercialization of the products and/or services referred to herein.

Copyright

© Copyright 2020–2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. All other trademarks are the property of their respective owners.