WP506 (v1.1) July 10, 2020

Xilinx AI Engines and Their Applications

For compute-intensive applications like 5G cellular and machine learning DNN/CNN, Xilinx's new vector processor AI Engines are an array of VLIW SIMD high-performance processors that deliver up to 8X silicon compute density at 50% the power consumption of traditional programmable logic solutions.

ABSTRACT

This white paper explores the architecture, applications, and benefits of using Xilinx's new AI Engine for compute intensive applications like 5G cellular and machine learning DNN/CNN.

5G requires between five to 10 times higher compute density when compared with prior generations; AI Engines have been optimized for DSP, meeting both the throughput and compute requirements to deliver the high bandwidth and accelerated speed required for wireless connectivity.

The emergence of machine learning in many products, often as DNN/CNN networks, dramatically increases the compute-density requirements. AI Engines, which are optimized for linear algebra, provide the compute-density to meet these demands—while also reducing the power consumption by as much as 50% when compared to similar functions being performed in programmable logic.

AI Engines are programmed using a C/C++ paradigm familiar to many programmers. AI Engines are integrated with Xilinx's Adaptable And Scalar Engines to provide a highly flexibly and capable overall solution.

Xilinx's Rich Compute History

Xilinx products have a multi-decade history of implementation in computationally intense applications, starting with high-performance computing (HPC) and digital signal processing (DSP) implementations in the early 1990s. The Xilinx XC4000 series of FPGAs became the enabling technology for implementing digital front-end (DFE) solutions for both commercial and Aerospace & Defense wireless communications systems. These early adopters used LUTs and adders to implement the compute elements (such as multipliers) to build DSP functions, FIR filters, and FFTs.

As customers began selecting Xilinx devices to handle demanding new computational applications, specific compute-intensive elements were created, like the first “DSP slice,” developed with the Virtex®-II series of FPGAs in 2001. Riding Moore's Law, Xilinx has increased the number of LUTs from just 400 in the XC4000 FPGA to over 3.7 million LUTs and over 12,200 DSP slices in current devices—an increase in available resources of over 9,500 times. With this accelerating increase in compute resources, Xilinx products have been consistently able to provide the compute density and logic resources needed to keep pace with the burgeoning signal-processing marketplace.

Technology Advancements Driving Compute Density

Advancements in multiple technologies are driving the need for non-linear higher compute density. Data converters with sampling rates of gigahertz per second enable direct sampling of RF signals, simplifying the analog system but requiring a corresponding order of magnitude higher DSP compute density. Direct RF sampling is coupled with using multiple antennas, such as advanced radar systems with their tens of thousands of antennas.

The hype surrounding 5G wireless has been brewing for years, and the technology promises to change people’s lives by connecting everything in the environment to a network that is one hundred times faster than a cellular connection and ten times faster than the speediest home broadband service. Millimeter waves, massive MIMO, full duplex, beam-forming, and small cells are just a few of the technologies that enable ultrafast 5G networks. Speed and low latency are two major benefits of 5G that promise many new applications, from autonomous vehicles to virtual reality. These technologies drive compute density and memory requirements to an order of magnitude greater than 4G.

With 5G, new technologies such as massive MIMO, multiple antenna, and frequency bands, increase the complexity 100 times that of 4G. Increasing complexity directly drives the compute density, memory requirements, and RF data converters performance. See Figure 1.

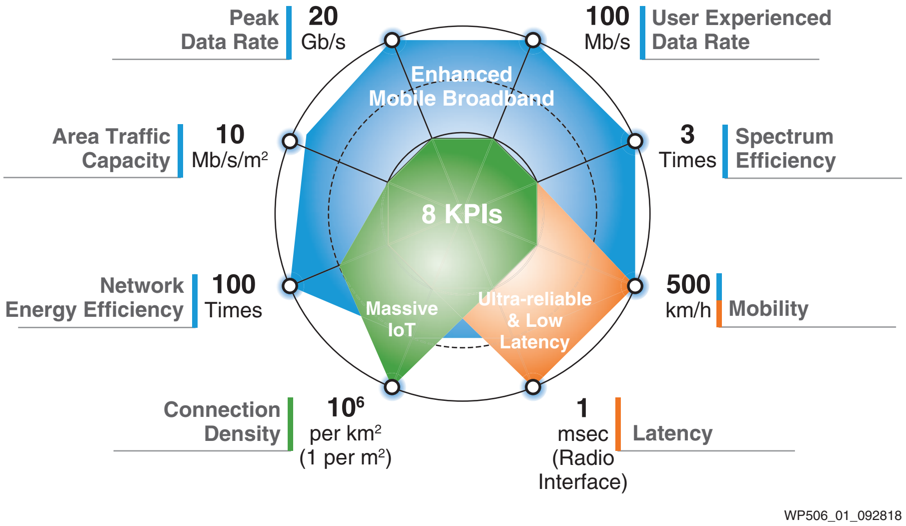


Figure 1: **5G Complexity vs. 4G**[1]

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

The Death of Moore's Law

In 1965, Intel co-founder Gordon Moore observed that the number of components in integrated circuits was doubling every 2 years. In 1965, this meant that 50 transistors per chip offered the lowest per-transistor cost; Moore predicted that by 1970, this would rise to 1,000 components per chip, and that the price per transistor would drop by greater than 90 percent. Moore later revised this to a doubling of resources every two years, which has held roughly true from 1975 until 2012.[(1)] Moore's Law projected that each next smaller process node would provide greater density, performance, and lower power, all at a lower cost. The observation was named "Moore's Law" and held true for roughly 50 years. The Moore's Law principle was a driving enabler for increasing IC density, performance, and affordability—a principle Xilinx has used to deliver devices that are increasingly capable at lower cost.

As IC process nodes reached 28nm and below, Moore's Law "broke"; devices built on smaller process nodes no longer provided a free ride for power, cost, and performance. A gap developed between the compute demand of 5G cellular systems and programmable logic compute density. The cost, power, and performance required by 5th-generation cellular were outstripping the programmable logic's ability to meet system-level goals.

Enter the AI Engine

Responding to the non-linear increase in demand by next-generation wireless and Machine Learning applications for higher compute density and lower power requirements, Xilinx began investigating innovative architecture, leading to the development of the AI Engine. The AI Engine along with Adaptable Engines (programmable logic) and Scalar Engines (processor subsystem) form a tightly integrated heterogeneous compute platform. AI Engines provide up to five times higher compute density for vector-based algorithms. Adaptable Engines provide flexible custom compute and data movement. Scalar Engines provide complex software support. See Figure 2.

[1] Wikipedia.org, "Moore's law," https://en.wikipedia.org/wiki/Moore%27s_law, retrieved Aug 2018.

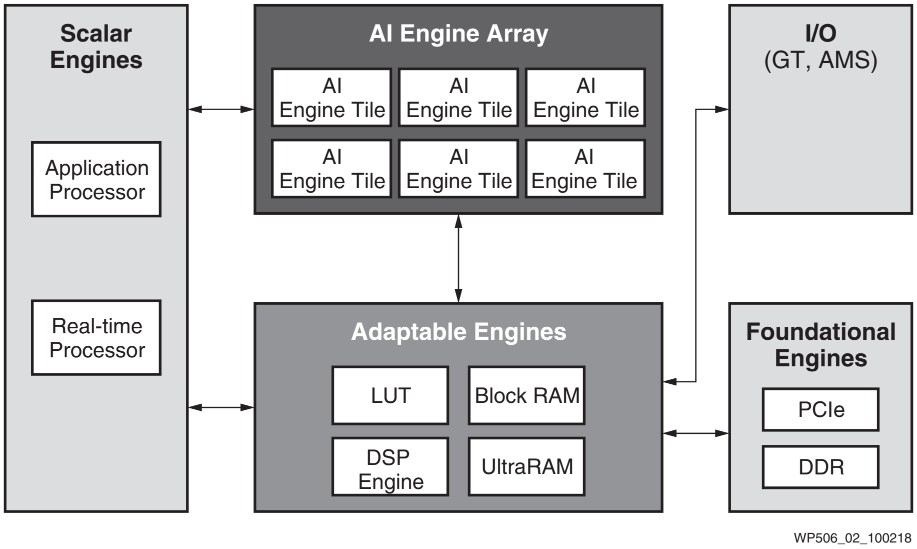


Figure 2: **Heterogeneous Compute**

Figure 3 illustrates the composition of AI Engine interface tiles into a 2D array.

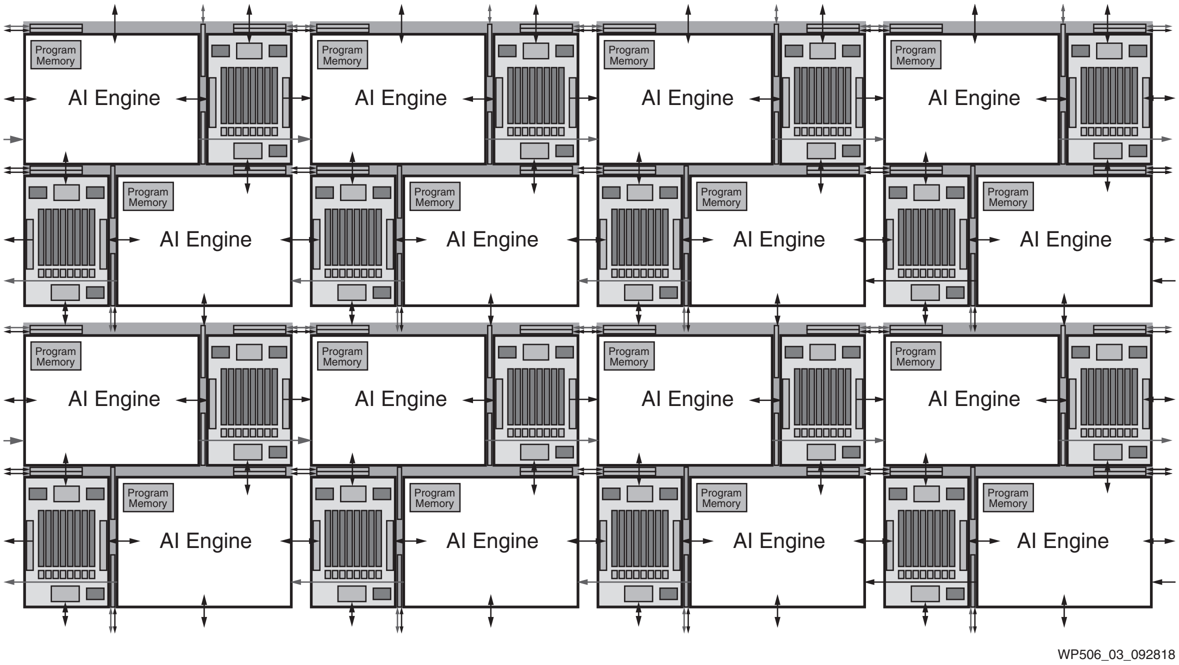


Figure 3: **AI Engine Array**

Each AI Engine tile includes vector processors for both fixed and floating-point operations, a scalar processor, dedicated program and data memory, dedicated AXI data movement channels, and support for DMA and locks. AI Engines are a single instruction multiple data (SIMD); and very long instruction word (VLIW), providing up to 6-way instruction parallelism, including two/three scalar operations, two vector load and one write operation, and one fixed or floating-point vector operation, every clock cycle.

Optimized for real-time DSP and AI/ML computation, the AI Engine array provides deterministic timing through a combination of dedicated data and instruction memories, DMA, locks, and software tools. Dedicated data and instructions memories are static, eliminating the inconsistencies that can come from cache misses and the associated fills.

AI Engine Goals and Objectives

AI Engine goals and objectives were derived from compute-intensive applications using DSP and AI/ML. Additional market requirements include greater developer productivity and higher levels of abstraction, which are driving the evolution of development tools. The AI Engine was developed to provide four primary benefits:

- Deliver three to eight times more compute capacity per silicon area versus PL implementation of compute-intensive applications
- Reduce compute-intensive power consumption by 50% versus the same functions implemented in PL
- Provide deterministic, high-performance, real-time DSP capabilities
- Dramatically improve the development environment and deliver greater designer productivity

AI Engine Tile Architecture Details

To truly get a sense of the tremendous capabilities of the AI Engine, it is essential to gain a general understanding of its architecture and capabilities. The AI Engine tile shown in Figure 4 provides a detailed accounting of the resources in each tile:

- Dedicated 16KB instruction memory and 32KB of RAM
- 32b RISC scalar processor
- 512b fixed-point and 512b floating-point vector processor with associated vector registers
- Synchronization handler
- Trace and debug

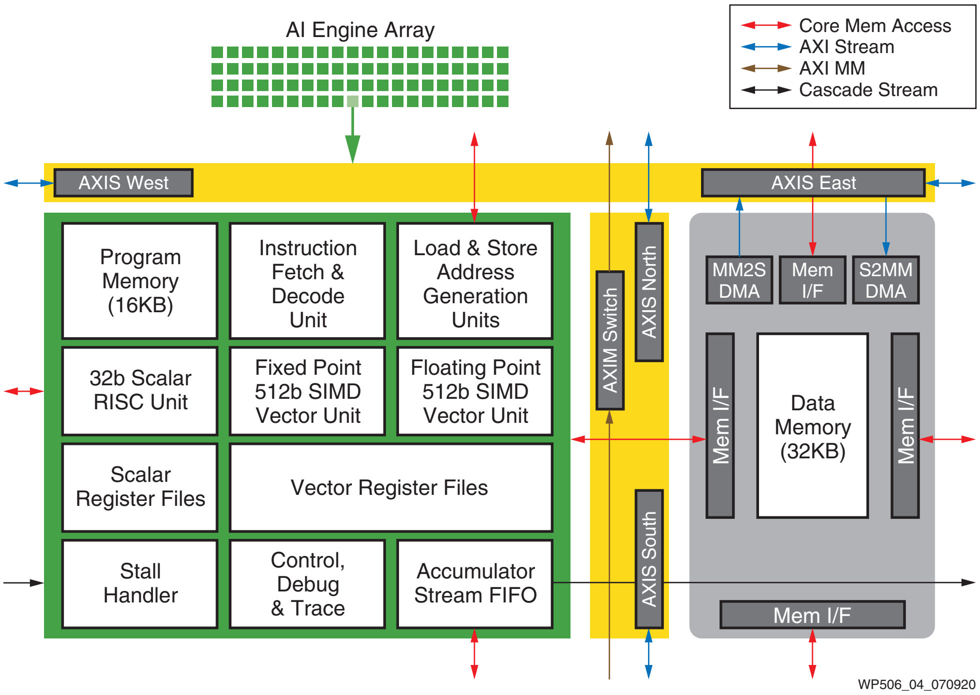


Figure 4: **Detail of AI Engine Tile**

An AI Engine with dedicated instruction and data memory is interconnected with other AI Engine tiles, using a combination of dedicated AXI bus routing and direct connection to neighboring AI Engine tiles. For data movement, dedicated DMA engines and locks connect directly to dedicated AXI bus connectivity, data movement, and synchronization.

Operand Precision Support

The vector processors are composed of both integer and floating-point units. Operands of 8-bit, 16-bit, 32-bit, and single-precision floating point (SPFP) are supported. For different operands, the operands-per-clock cycle changes, as detailed in Table 1.

Table 1: **AI Engine Vector Precision Support**

Operand A	Operand B	Output	Number of MACs/Clock
8b real	8b real	16b real	128
16b real	8b real	48b real	64
16b real	16b real	48b real	32
16b real	16b complex	48b complex	16
16b complex	16b complex	48b complex	8
16b real	32b real	48/80 real	16
16b real	32b complex	48/80 complex	8
16b complex	32b real	48/80 complex	8
16b complex	32b complex	48/80 complex	4
32b real	16b real	48/80 complex	16
32b real	16b complex	48/80 complex	8
32b complex	16b real	48/80 complex	8
32b complex	16b complex	48/80 complex	4
32b real	32b real	80b real	8
32b real	32b complex	80b complex	4
32b complex	32b real	80b complex	4
32b complex	32b complex	80b complex	2
32b SPFP	32b SPFP	32b SPFP	8

Instruction and Data Parallelism

Multiple levels of parallelism are achieved through instruction-level and data-level parallelism.

Instruction-level parallelism is shown in Figure 5. For each clock cycle, two scalar instructions, two vector reads, a single vector write, and a single vector instruction executed—6-way VLIW.

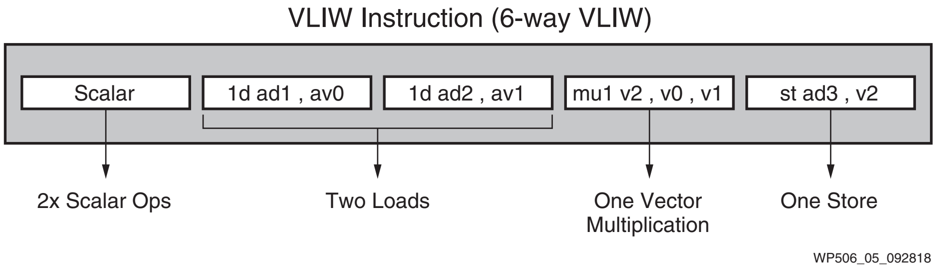


Figure 5: **AI Instruction Level Parallelism**

Data level parallelism is achieved via vector-level operations where multiple sets of data can be operated on a per-clock-cycle basis, as shown in Table 1.

Deterministic Performance and Connectivity

The AI Engine architecture was developed for real-time processing applications, which require deterministic performance. Two key architectural features ensure deterministic timing:

- Dedicated instruction and data memories
- Dedicated connectivity paired with DMA engines for scheduled data movement using connectivity between AI Engine tiles

Direct memory (DM) interfaces provide direct access between the AI Engine tile and its nearest neighbors, AI Engine tile data memory to the north, south, and west. This is typically used to move results to/from the vector processors while the overall processing chain produces and/or consumes data. Data memory is implemented to enable a “ping-pong” buffering scheme, which minimizes memory contention impacts on performance.

AXI-Stream and AXI-Memory Mapped Connectivity between AI Engine Tiles

The simplest form of AI Engine to AI Engine data movement is via the shared memory between direct neighboring AI Engine Tiles. However, when the tiles are further away, then the AI Engine tile needs to use the AXI-Streaming dataflow. AXI-Streaming connectivity is predefined and programmed by the AI Engine complier tools based on the data flow graph. These streaming interfaces can also be used to interface directly to the PL and the network on chip (NoC). See Figure 6.

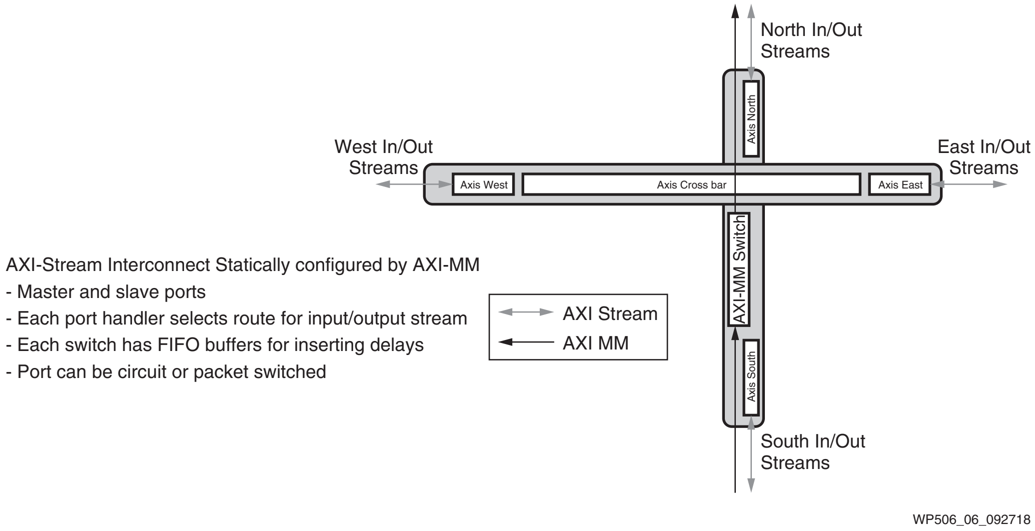

North In/Out
Streams
Axis North
West In/Out
Streams
Axis West
Axis Cross bar
Axis East
East In/Out
Streams
AXI-MM Switch
AXI-Stream Interconnect Statically configured by AXI-MM
- Master and slave ports
- Each port handler selects route for input/output stream
- Each switch has FIFO buffers for inserting delays
- Port can be circuit or packet switched
AXI Stream
AXI MM
Axis South
South In/Out
Streams
WP506_06_092718

Figure 6: **AI Engine Array AXI-MM and AXI-Stream Interconnect**

AI Engine and PL Connectivity

One of the Versal portfolio's highest value propositions is the ability to use the AI Engine array with programmable logic in the Adaptable Engine. The combination of resources provides great flexibility to implement functions in the optimal resource, AI Engine, Adaptable Engine, or Scalar Engine. Figure 7 illustrates the connectivity between the AI Engine array and the programmable logic, called the “AI Engine array interface.” AXI-Streaming connectivity exists on each side of the AI Engine array interface, and extends connectivity into the programmable logic and separately into the NoC.

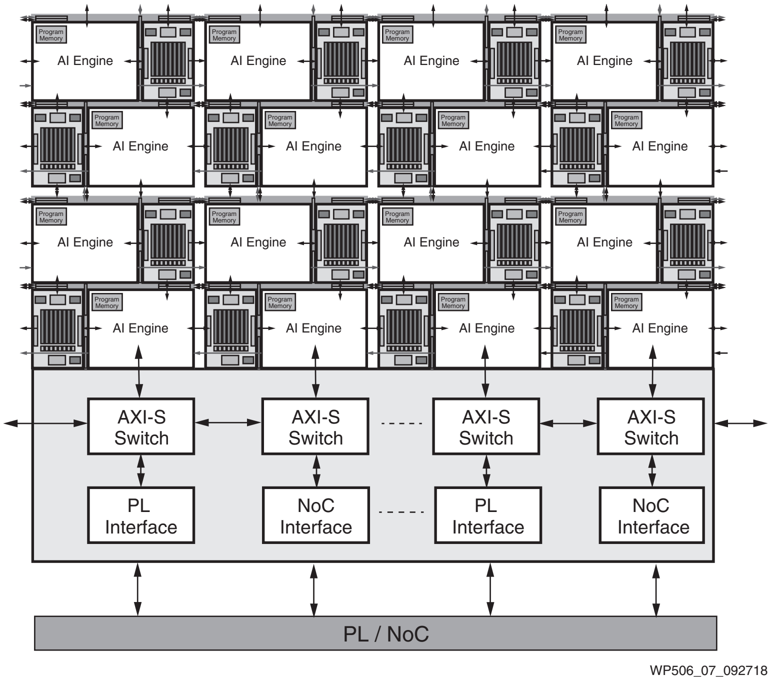


Figure 7: **AI Engine Array Interface**

AI Engine Control, Debug, and Trace

Control, debug, and trace functions are integrated into every AI Engine tile, providing visibility for debug and performance monitoring and optimization. Access to the debug capabilities is through the high-speed debug port introduced in the Versal portfolio.

Comparing AI Engine and Programmable Logic Implementations

Section AI Engine Goals and Objectives provides metrics required for assessing whether application and market demands are being met. The effectiveness of the architecture can be measured by implementing 4G and 5G cellular in both the PL and the AI Engine. A summary of the results shows that AI Engine-based solutions can provide:

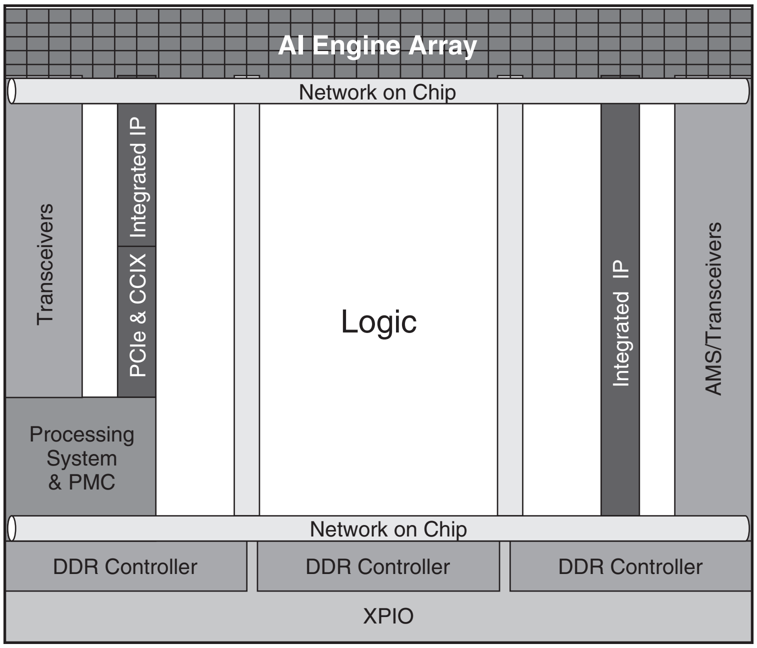
- Silicon area 3X–8X smaller compared to the same function implemented in PL on the same process node
- Power consumption about 50% that of PL implementation

For those functions that do not fit into a vector implementation, AI Engine efficiency is far less, and AI Engine is often not as good a fit. In these instances, the PL is a better solution. AI Engines and PL are intended to operate as compute peers, each handling functions that match their strengths. PL is excellent for data movement, bit-oriented functions, and non-vector-based computation; it can also implement custom accelerators for non-AI Engine supported operations. PL and AI Engine complement each other and form a stronger system-level solution. Programmable logic is still a highly valuable resource within most compute-intensive applications; the AI Engine/PL combination can provide flexibility, high compute performance, and high bandwidth data movement and storage.

Overview of Versal Portfolio with AI Engine Architecture

Versal devices include three types of programmable processors: Arm® processor subsystem (PS), programmable logic (PL), and AI Engines. Each provides different computation capabilities to meet different portions of the overall system. The Arm processor is typically used for control-plane applications, operating systems, communications interfaces, and lower level or complex computations. PL performs data manipulation and transport, non-vector-based computation, and interfacing. AI Engine is typically used for compute-intensive functions in vector implementations.

Figure 8 provides a high-level view of a Versal device with an AI Engine array located at the top of the device. Connectivity between the AI Engine array and PL is supported both directly and through the NoC.



WP506_08_092818

Figure 8: **Versal ACAP with AI Engine Architecture Overview**

AI Engine Development Environment

In recent years, Xilinx has placed a great deal of emphasis on the use of high-level languages (HLL) to help raise the level of abstraction for developing with Xilinx devices. The Versal architecture has three fundamentally different programmable elements: PL, PS, and AI Engine. All three can be programmed using C/C++.

AI Engine simulation can be functional or cycle accurate using an x86-based simulation environment. For system-level simulation, a System-C virtual platform is available that supports all three processing domains.

A key element in the development environment are the AI Engine libraries that support DSP and wireless functions, ML and AI, linear algebra, and matrix math. These libraries are optimized for efficiency and performance, enabling the developer to take full advantage of AI Engine capabilities.

AI Engine Applications

AI Engines have been optimized for compute-intensive applications, specifically digital signal processing (DSP) and some artificial intelligence (AI) technology such as machine learning (ML) and 5G Wireless applications.

Digital Signal Processing Using AI Engines

Radio Solutions Validation Suite

Real-time DSP is used extensively in wireless communications. Xilinx compared implementations of classic narrow-band and wide-band radio design principles, massive MIMO, and baseband and digital front-end concepts, validating the AI Engine architecture as being well-suited to building radio solutions.

Example: 100MHz 5-Channel LTE20 Wireless Solution

A 100MHz 5-channel LTE20 wireless was implemented in a portion of a Versal device. Five channels of 16b input data are streamed in at 30.72MSPS and processed in an 89-tap channel filter. The signals are then up-sampled by four using two stages of half-band filters (23 and 11 taps), resulting in a sample rate of 122.88MSPS.

The up-sampled stream is then mixed with a direct-digital synthesized (DDS) sine/cosine wave function and summed. Two additional half-band filters (47 and 27 taps) up-sample by a total of four to produce a 491.52MSPS input stream to a crest-factor reduction (CFR) function. A fractional rate change, five-up/four-down provided by a 41-tap filter, results in a 614.4MSPS input sample rate to a digital pre-distortion function (DPD).

A peak detector/scale find (PD/SF) circuit is implemented in PL; the output of the 491.52MSPS DUC and mixer stage comprises one of its inputs, while the CFR second stage provides its second input. The PD/SF circuit, implemented in PL, is resource-efficient; conversely, it is resource-*inefficient* if implemented in an AI Engine. This is a good illustration of an architectural decision taken to utilize the best resource for different functional blocks of the design. See Figure 9.

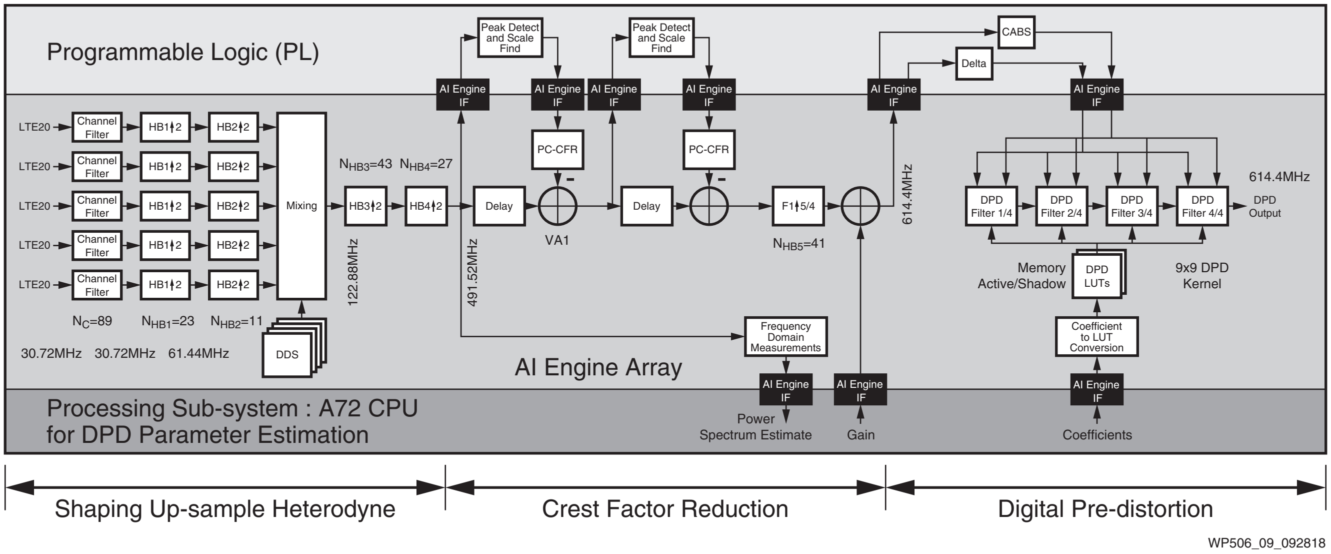


Figure 9: **Block Diagram: 100MHz 5-Channel LTE20 Wireless Solution with DSP**

The DPD function requires periodic recalculation of coefficients. A feedback path from the output of the transmit digital-to-analog converter (DAC) is sampled, using an analog-to-digital converter (ADC), and buffered. A buffered sample data set is passed to the PS, and used to calculate a new set of DPD coefficients ten times per second. New coefficient sets are written back into the DPD using the network-on-chip and AXI bus interconnect.

Machine Learning and AI Engines

In machine learning, a convolutional neural network (CNN) is a class of deep, feed-forward artificial neural networks most commonly applied to analyzing visual imagery. CNNs have become essential as computers are being used for everything from autonomous driving vehicles to video surveillance and Data Center analysis of images and video. CNNs provided the technological breakthrough that made the reliability of vision-image recognition accurate enough to be used to safely guide a vehicle.

CNN techniques are in their infancy, with new breakthroughs being announced nearly every week. The pace of innovation in this field is astounding and it means that new, previously unobtainable applications are likely to be enabled within the next few years.

However, the challenge with CNNs is the intense amount of computation required - commonly requiring multiple TeraOPS. AI Engines have been optimized to efficiently deliver this computational density cost effectively and power efficiently.

AI Engine CNN/DNN Overlay

Xilinx is developing a machine-learning inference engine built on the AI Engines and will be applied as an application overlay. Programmable logic is used to efficiently move and manage data. The AI Engine application overlay provides a defined structure for executing the compute and other operations needed to implement many of the popular CNN/DNN networks, such as ResNet, GoogLeNet, and AlexNet.

From a user's perspective, the overlay approach has many advantages, including the ability to be modified as newer network architectures emerge. The programmable combination of AI Engine and PL provides an efficient and very flexible platform that can grow and expand as the ML application space advances.

The AI Engine CNN/DNN overlay is used both in Data Center applications for accelerating ML network inferencing and in embedded systems. Integration is a simple matter of instantiating the solution into the user’s overall design. CNN/DNN networks are then developed using TensorFlow or Caffe and compiled into an executable program that runs on the AI Engine CNN/DNN overlay.

Summary

AI Engines represent a new class of high-performance compute. Integrated within a Versal-class device, the AI Engine can be optimally combined with PL and PS to implement high-complexity systems in a single Xilinx ACAP. Real-time systems require deterministic behavior, which the AI Engine delivers through a combination of architecture features such as dedicated data and programming memories, DMA and Locks, and compiler tools.

AI Engines deliver three to eight times better silicon area compute density when compared with traditional programmable logic DSP and ML implementations, while reducing power consumption by nominally 50%. A C/C++ programming paradigm raises the level of abstraction and promises to significantly increase the developer's productivity.

System performance scalability is provided through a family of devices, ranging from small devices with 30 AI Engines and 80K LUTs to devices with 400 AI Engines and nearly a million LUTs. Package footprint compatibility between these devices enables migration within the product family to meet varying performance and price targets.

For more information, go to:

WP505, *Versal: The First Adaptive Compute Acceleration Platform (ACAP)*

WP504, *Accelerating DNNs with Xilinx Alveo™ Accelerator Cards*

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
07/10/2020	1.1	Updated Figure 4.
10/03/2018	1.0.2	Editorial updates only.
10/02/2018	1.0.1	Editorial updates only.
10/02/2018	1.0	Initial Xilinx release.

Disclaimer

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 http://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 http://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.