AMD
together we advance_

Designing with Versal AI Engine: Kernel Programming and Optimization - 3

AIE-KERNEL (v1.0)

Course Specification

Course Description

This course covers the advanced features of the AMD Versal™ adaptive SoC AI Engine, including kernel function development, optimizing an AI Engine kernel program, using AI Engine APIs and filter intrinsics, and debugging an application in the AMD Vitis™ Unified IDE.

The emphasis of this course is on:

- Reviewing the features of the Versal device AI Engine architecture
- Optimizing AI Engine kernels using compiler directives, programming style, profiling, and efficient movement of data
- Describing C++ kernel template functionality
- Identifying the different types of kernel instance states
- Programming FIR filters using AI Engine APIs
- Debugging AI Engine applications using the Vitis Unified IDE

What's New for 2025.2

- Added information on microcode basics, kernel optimization techniques via source code updating, and factors that impact performance
- All labs have been updated to the latest software versions

Level – VER 4

Course Details

- 2 days ILT or 3 sessions/19

Course Part Number – AIE-KERNEL

Who Should Attend? – Software and hardware developers, system architects, and anyone who needs to accelerate their software applications using AMD devices

Prerequisites

- Comfort with the C/C++ programming language
- Vitis software for application acceleration development flow
- *Designing with Versal AI Engine: Quick Start*
- *Designing with Versal AI Engine: Architecture and Design Flow - 1*
- *Designing with Versal AI Engine: Graph Programming with AI Engine Kernels - 2*

Software Tools

- Vitis Unified IDE 2025.2

Hardware

- Architecture: Versal adaptive SoCs

After completing this comprehensive training, you will have the necessary skills to:

- Utilize various AMD Versal AI Engine kernel optimization techniques, such as compiler directives, software pipelining, coding for performance, and core utilization
- Apply C/C++ coding guidelines for performance improvement, including function inlining, pointer restricting, profiling, and code shuffling
- Identify and implement the different types of kernel instance states using C++ kernel development
- Implement AI Engine kernels using AI Engine APIs for symmetric and non-symmetric FIRs
- Debug an AI Engine application for memory conflicts and stream deadlocks using the Analysis view and event traces in the Vitis Unified IDE

Course Outline

Day 1

- **AI Engine and Memory Module Architecture**
  Introduces the architecture of the AI Engine and describes the memory module architecture for the AI Engine. {Lecture}
- **Versal AI Engine Data Movement and Interfaces**
  Describes the data movement and memory access by the AI Engines in the AI Engine arrays. Also reviews the AI Engine interfaces that are available, including the lock, core debug, cascaded stream, and AXI-Stream interfaces. {Lecture}
- **Overview of AI Engine Kernel Optimization**
  Highlights the various AI Engine kernel optimization techniques, such as compiler directives, software pipelining, coding for performance, and core utilization. {Lecture}
- **AI Engine Kernel Optimization – Compiler Directives**
  Describes the usage of compiler directives for loop unrolling, loop flattening, and software pipelining to help improve the performance of AI Engine kernels. {Lecture}
- **AI Engine Kernel Optimization – Coding Style**
  Covers the coding guidelines for performance improvement, including function inlining, pointer restricting, and code shuffling. Also covers calculating AI Engine utilization for the kernels to help improve performance. The lab illustrates applying kernel optimization techniques such as the restrict keyword, custom pragmas, and code restructuring. {Lecture, Lab}
- **Advanced C++ Kernel Programming**
  Provides an overview of C++ kernel template functionality and the different types of states and kernel instance states using C++ classes. Also covered are kernel instance states with scalar parameters in a constructor as well as kernel instance states with array parameters in a constructor. {Lecture, Lab}

Day 2

- **Vector Data Types (Review)**
  Provides an AI Engine functional overview and identifies the supported vector data types and high-width registers for allowing single instruction, multiple data (SIMD) instructions. {Lecture}
- **AI Engine Symmetric and Asymmetric Filter Implementation**
  Describes AI Engine APIs for symmetric and asymmetric FIR implementation, such as aie::sliding_mul_sym_xy_ops operators. Also, provides an overview of the DSP library, which can help with creating filters more easily and faster. {Lecture, Lab}
- **Debugging AI Engine Applications – Event Trace**
  Describes the application simulation debugging methodology as well as debugging with event traces, such as AI Engine events, DMA events, lock events, and stream events. Also demonstrates how to visualize these events in the Vitis unified software platform. {Lecture}
- **Debugging AI Engine Applications – Use Cases**
  Reviews various use cases of problems that arise, such as memory conflicts and deadlock analysis. Also covers performance analysis (profiling) in hardware. {Lecture, Lab}

- **Introduction to the AI Engine DSP Library [Optional]**
  Provides an overview of the available DSP library, which enables faster development and comes with ready-to-use example designs that help with using the library and tools. {Lecture, Labs}

Appendix:

- **AI Engine Symmetric Filter Implementation Using Intrinsics**
  Describes advanced MAC intrinsic syntax, including the intrinsics for symmetric FIR implementation, such as mul4_sym and mac4_sym. Also provides guidelines for choosing the right fixed-point intrinsics for a FIR filter. {Lecture}
- **AI Engine Non-Symmetric Filter Implementation Using Intrinsics**
  Describes the intrinsics for non-symmetric FIR implementations, such as mul4_nc and mac4_nc. Also provides guidelines for choosing the right intrinsics for a FIR filter. {Lecture}
- **Floating-Point Operations Using Intrinsics**
  Reviews the floating-point operations fpmul, fpmac, and fpmsc as well as the fully configurable, floating-point intrinsics fpmac_conf. {Lecture}

.