arm
APPROVED TRAINING PARTNER

Cortex-A55 MPCore Software Development

Course Description

Cortex-A55 MPCore software development is a 4 days ARM official course. The course goes into great depth and provides all necessary know-how to develop software for systems based on Cortex-A55 processors.

The course introduces the ARMv8-A architecture, instruction set, and the new model to handle interrupts and exceptions.

The course continues by covering the Cortex-A55 MPCore architecture based on DynamIQ technology, memory management unit, memory model, cache and branch prediction, cache coherency, processes synchronization, boot process, barriers, virtualization, Generic Interrupt Controller (GIC), System MMU (SMMU), power management, debug, security, RAS support, and DynamIQ Shared Unit (DSU).

At the end of the course the participant will receive a certificate from ARM.

Course Duration

4 days (5 with hands-on labs)

Goals

1. Become familiar with ARMv8-A Cortex-A55 architecture
2. Understand the main differences between ARMv7-A and ARMv8-A architectures
3. Understand the advantages of DynamIQ technology
4. Become familiar with ARMv8-A instruction set
5. Understand the ARMv8-A exception model
6. Be able to configure and use the ARMv8-A MMU
7. Be familiar with ARMv8-A memory model
8. Be familiar with ARMv8-A caches and branch prediction
9. Understand ARMv8-A cache coherency features and how to configure them
10. Be able to boot Cortex-A55 MPCore system
11. Implement synchronization processes using ARM primitives to build mutex/semaphore
12. Be able to add barriers instructions to control program flow order
13. Be able to program the GIC
14. Understand the use of System MMU
15. Become familiar with NEON coprocessor SIMD capabilities
16. Manage Cortex-A55 MPCore power modes
17. Be able to debug with invasive and non-invasive techniques
18. Become familiar with TrustZone infrastructure to build secured systems
19. Become familiar with Virtualization and its effect on the system
20. Embed AMP and SMP operating systems

Target Audience

Software engineers that would like developing software and BSP for platforms based on ARMv8-A Cortex-A55 MPCore processor.

Prerequisites

- ARMv7-A architecture
- Computer architecture background
- C and Assembler
- Experience in developing embedded systems

Course Material

- ARM official course book
- Labs handbook
- DS5 SDK

Agenda

Main Topics:

- Cortex-A55 Processor Overview
- Introduction to the ARMv8-A Architecture
- AArch64 A64 ISA Overview
- AArch64 Exception Handling
- ARMv8-A MMU
- ARMv8-A Memory Model
- ARMv8-A Caches and Branch Prediction
- ARMv8-A Cache Coherency
- Understanding Barriers
- Synchronization
- OS Support
- Booting a Cortex-A55 MPCore
- Programming the GIC
- Using the SMMU
- ARMv8-A Debug and Trace
- ARMv8-A Virtualization

- DynamIQ Shared Unit (DSU)
- DynamIQ RAS Support
- Cortex- A55 Power Management
- ARMv8-A Secure Environment using TrustZone

Day #1

❖ ARMv8-A Architecture Overview

- Architecture Versions
  - Development of the ARM architecture
  - What’s new in ARMv8-A?

- Privilege Levels
  - AArch64 privilege levels
  - AArch32 privilege levels
  - Moving between AARch32 and AArch64

- AArch64 Registers
  - Register banks
  - Other registers (XZR, WZR, X30, ELR_ELn)
  - Processor state
  - Procedure call standard
  - AArch64 and AArch32 register mappings
  - System control
  - System registers

- A64 Instruction Set
  - A64 overview

- Exception Model
  - AArch64 exceptions
  - Taking an exception

- Memory Model
  - Memory types
  - Data alignment
  - Virtual address space
  - Multiple virtual address spaces
  - Physical address spaces
  - MPCore configurations

DynamIQ A64 ISA Overview

- Instruction Sets
  - AArch32
  - AArch64

- Register Set
  - General purpose registers
  - Scalar FP and SIMD registers

- Load/Store Instructions
  - Load/store instructions overview
  - Register Load/store
  - Byte load examples
  - Load/store address
  - Addressing modes
  - Floating point loads and stores
  - PC relative load
  - Register pair load/store
  - Stack accesses

- Data Processing Instructions
  - Data processing overview
  - Shift/Rotate operations
  - Bit manipulation instructions
  - Signed or Zero extend
  - Multiply
  - Division
  - Conditional execution
  - Using the ALU flags
  - Floating point operations
  - Dot product (ARMv8.2)

- Program Flow Instructions
  - Branch instructions
  - Conditional branch

- System Control
  - System register access

- Advanced SIMD
  - SIMD operations
  - Vectors
  - Examples SIMD instructions

- Cryptographic Extensions
  - Cryptographic extensions overview

  - Using the cryptographic instructions
  - AES instructions

- Additional Instructions
  - Special load and store (LDNP/STNP, LDTR/STTR, LDAR/STLR, LDXR/STXR)
  - Prefetch memory
  - Exception generation and return
  - Breakpoints
  - Hints
  - Key differences from A32
  - Load/Store offset range
  - Immediate values logical operations
  - Load-Store non-temporal pair (LDNP/STNP)
  - Unprivileged Load/Store
  - Exclusive accesses (LDXR/STXR)
  - Load acquire – Store release (LDAR/STLR)
  - Bitfield Move/Extract (BFM/EXTR)
  - Data Move
  - Floating point compare/select

ARMv8-A AArch64 Exception Model

- The AArch64 Exception Model
  - Exception levels
  - AArch64 exceptions
  - Taking an exception
  - Exception routing
  - PSTATE and the SPSR
  - Changing execution state
  - Exception return address
  - Exception stacks
  - AArch32 register mapping
  - AArch64 vector table

- Interrupts
  - Interrupt handling
  - The Generic Interrupt Controller (GIC-400, GIC-500)
  - Interrupt example
  - Exception handler example
  - Nested exception example
  - Nested exception handler example

- Synchronous Exceptions
  - Synchronous exceptions overview
  - System calls
  - Handling synchronous exceptions

  - Exception Syndrome register

- SError Exceptions
  - SError exceptions overview

- Exceptions in EL2 and EL3
  - System calls to EL2/EL3
  - Routing exceptions to EL2/EL3
  - Routing exception example
  - Example system with EL3 - non secure
  - Example system with EL3 - secure

- **Cortex-A55 Processor Overview**

- Cortex-A55 Introduction
  - Cortex-A55 CPU
  - L1 cache overview
  - L2 cache overview
  - L1/L2 cache allocation (instruction fetch)
  - L1/L2 cache allocation (data access)
  - Core and cluster cache policies
  - System Control Register (SCTLR)
  - Memory system
  - AArch64 PRFM (prefetch memory) instructions
  - Non-temporal loads and stores
  - Writeback allocation hints
  - ECC and parity error detection and correction

Day #2

DynamIQ Memory Management

- Memory Management Quick Refresher
  - Why do we need memory management?
  - What is virtual addressing?
  - What is Memory Management Unit?
  - How a physical address is formed?
  - Multiple levels of translation table

- Stage 1 Translations at EL1/EL0
  - ARMv8-A translation tables
  - AArch64 translation tables
  - AArch64 table descriptor format
  - AArch64 tables with 4KB/16KB/64KB granules
  - Separate tables for application and kernel space
  - Translation control register
  - Setting the first level of lookup
  - Caching translation tables
  - Contiguous block entries

- Translations at EL2/EL3
  - Translation tables overview
  - Stage 2 translations (IPA -> PA)
  - Stage 1 translation EL2/3
  - Secure world translation tables

- TLB Maintenance
  - Translation table change example
  - AArch64 instructions (TLBI)

- ARMv8.2
  - ARMv8.2 VMSA changes
  - ARMv8.2-LVA
  - ARMv8.2-LPA

DynamIQ Memory Model

- Memory Model Quick Refresher
  - ARMv8-A memory model
  - How attributes are specified?
  - Hierarchical attributes

- Memory Types
  - ARMv8-A memory types overview
  - Normal memory

  - o Device memory
  - o Ordering of device accesses
  - o Specifying the type
  - o Stronger to weaker device memory

- ➢ Memory Attributes
  - o Cacheability
  - o Shareable
  - o Normal memory behavior guarantees
  - o Access permissions
  - o Executable
  - o Access flag
  - o Global/non-global translations
  - o ASIDs
  - o Reserved bits
  - o Physical address spaces
  - o Secure or non-secure (NS attribute)

- ➢ Alignment & Endianess
  - o Alignment
  - o Endianness in AArch64

- ➢ Tagged Pointers
  - o Tagged pointers (AArch54 only)

- ➢ ARMv8.2
  - o Hardware management of the Access flag and dirty state
  - o Common not private (CnP)
  - o Privileged Access Never (PAN)
  - o Hierarchical permissions disable
  - o Page based hardware attributes
  - o ARMv8.2-UAO
  - o ARMv8.2-LSMAOC

❖ **DynamIQ Caches & Branch Prediction**

- ➢ Caches in DynamIQ CPUs processors
  - o How is data stored in my cache?
  - o How are caches accessed?
  - o Level 1 and level 2 cache interaction
  - o Branch prediction

- ➢ Cache Attributes
  - o Cache policies
  - o Write-back and write-through
  - o Inner and outer
  - o Speculation and preloading

- Cache Maintenance Operations
  - Cache maintenance
  - PoU and PoC and cache maintenance
  - PoU and PoC compared
  - Persistent memory (AArch64 only)
  - AArch64 instructions
  - Maintenance broadcast
  - Maintenance broadcast – Aarch64

- Cache Discovery
  - Cache discovery code
  - Non-integrated caches
  - Cache disabled behavior

- **DynamIQ Cache Coherency**

- Introduction to Coherency
  - What is cache coherency?
  - DynamIQ cache coherency
  - Shareability in the translation tables
  - AMBA 4 ACE and AMBA 5 CHI
  - System coherency with GPUs and DMA

- MPCore Coherency
  - Coherency implementation details
  - MPCore Coherency management
  - Coherency logic
  - Cache coherency logic example

- Multi-Processor Systems Coherency
  - Multi-cluster coherency
  - Coherency example: Reads
  - Coherency example: Writes

Day #3

❖ DynamIQ Barriers

- Overview
  - Memory model
  - Why do I care about access order?
  - Barriers (DMB, DSB, ISB)

- Data Barriers
  - DMB vs DSB
  - DMB instruction example
  - DSB instruction example
  - Different observers
  - DMB and DSB qualifiers
  - Mailbox example
  - Mailboxes with interrupts
  - Speculation across barriers
  - Memory mapped peripherals
  - “One-Way” barriers

- Instruction Barriers
  - ISB instruction
  - ISB example
  - Translation table change example
  - Self-modifying code example

- DynamIQ Extensions
  - Limited Ordering Regions (LDLAR, STLLR)
  - LOR example
  - Release consistency weakening

- Compiler Barriers

❖ DynamIQ Synchronization

- Introduction to Synchronization
  - The Race for Atomicity
  - Critical Sections
  - Simple lock implementation
  - Atomicity in ARM DynamIQ processors

- Enforced Atomicity
  - Atomic memory accesses
  - Compare and swap
  - Atomic memory operations: mnemonics

- Atomic memory operations: the operations
- Swap
- Ordering requirements
- How are atomics implemented?
- Simple lock() implementation using atomics

- Measured Atomicity
  - Load/store exclusive operations
  - How does measured atomicity work?
  - Simple lock() implementation using exclusives
  - Multi-thread lock example

- Local and Global Exclusive Monitors
  - Where is the exclusive monitor?
  - Context switching
  - Granularity of exclusive monitor
  - Coherent lock example
  - WFE
  - Programs still have to be smart

❖ **DynamIQ RAS Support**

- RAS Introduction
  - What is RAS?
  - Spectrum of RAS implementations
  - ARMv8 RAS extension

- RAS Terminology
  - Nodes
  - RAS terminology
  - How errors move around the system
  - Error taxonomy

- Error Exceptions
  - FHI and ERI
  - Recording errors
  - Error exception example
  - How might software handle an error?

- ESB
  - Error Synchronization Barrier

- DynamIQ Memory Error Handling
  - Memory error handling
  - Correctable errors
  - Uncorrectable errors
  - Data poisoning

  - DSU interrupt outputs
  - Other error reporting
  - Error reporting examples

❖ **Software Guide to DynamIQ Shared Unit (DSU)**

- DSU Introduction
- CPU Bridges
- CPU Caches
- DSU Snoop Filter and L3
- L3 Cache Allocation
- DSU Memory Interfaces
- Debug & Trace
- Power Management

Day #4

❖ **DynamIQ Booting**

- Overview
  - Booting considerations

- Booting an ARM DynamIQ Processor in AArch64
  - Processor state at cold reset
  - Processor state at warm reset
  - Moving to lower exception levels
  - Saved program status register
  - Example: EL3 to EL2
  - What does the boot code need to handle?
  - CPU specific power-up sequences
  - Enabling floating point and SIMD
  - Enabling MMU and caches
  - Additional considerations

- Booting Multi-Core and Multi-Processor Systems
  - Multi-core processors
  - Multi-processor systems

- Real-World Booting
  - Simple boot sequences
  - Booting complex systems

- ARM Trusted Firmware
- Bootloader stages
- ARM Trusted Firmware architecture
- Using the ARM Trusted Firmware
- Key registers

❖ DynamIQ Power Management

- Overview
  - New power management features
  - Power domains
  - Software-visible power states
  - Power modes
  - Controlling power mode transitions

- Core Power Modes
  - Core power domain modes and transitions
  - Standby mode
  - Standby use cases and considerations
  - Core dynamic retention
  - CPU power down sequence
  - Core power down sequence differences
  - SIMD (NEON) dynamic retention

- Cluster Power Modes
  - Cluster power domain modes and transitions
  - Cluster power down register
  - Cluster shutdown
  - Enable/disable interconnector coherency
  - Cluster memory retention modes
  - Debug power management

- L3 Cache Power Modes
  - SCU-L3 RAM power domains
  - Dynamic resizing (through power down)
  - L3 partial power down
  - DynamIQ cluster memory retention modes
  - DSU L3 power domain modes and transitions

❖ DynamIQ Virtualization

- Virtualization Overview
  - What is virtualization
  - Type 1 and 2 hypervisors
  - DynamIQ virtualization features

- ARM Virtualization Support
  - ARMv8-A virtualization
  - Instruction/register trapping

- Memory Management
  - Second stage translation
  - Stage 2 memory management
  - Translation regimes
  - Stage 2 translation overhead
  - Virtual Machine ID (VMID)

- Exception Handling
  - Device interrupt routing
  - Virtualizing exceptions
  - Interrupt routing to EL2/hypervisor
  - Virtual exceptions
  - GICv2 & GICv3 – Virtual Interrupts
  - Virtual interrupt signaling (GIC)
  - Virtual interrupt signaling (internal)
  - Generic timer
  - Virtual count and timer
- Virtualization Host Extensions
  - VHE (AArch64 only)
  - EL2 system register access when E2H==1
  - DynamIQ virtual address spaces when E2H==1
  - Exception routing
  - Overhead without VHE
  - DynamIQ virtualization
  - Running host OS
  - Running host OS application
  - Running guest OS
  - Running guest OS application
  - Virtual count and timer when E2H==1

- **ARMv8-A Secure Environments**

- Why Do We Need a Secure Environment?
  - What are we protectin
  - What are we protecting it from?
  - How valuable is the thing we are protecting?
  - Example: Firmware/OS update

  - Example: Filesystem

- Software Stack
  - What would a TrustZone software stack look like?
  - Scheduling
  - Trusted boot
  - Example: ARM Trusted Firmware + OP-TEE

- System Architecture
  - Memory system support
  - What system resources do I need?
  - TBSA and TBBR
  - DRM example

❖ Software Engineer's Guide to System Fabric

- System Fabric Overview
- Interrupt Controller
  - Development of the GIC architecture
  - Interrupt types
  - Interrupt states
  - GICv2 architecture (GIC-400)
  - SPI routing in GICv1/v2
  - Interrupt security
  - Interrupt security example
  - GICv3 architecture (GIC-500, GIC-600)
  - Message based interrupts – new in GICv3
  - SPI routing in GICv3 – Affinity levels
  - Interrupt security
  - Interrupt security example
  - LPIs
  - What is ITS?
  - Differences between GICv2 & GICv3

- System MMU
  - What is a SMMU?
  - Development of the SMMU architecture
  - SMMU and multiple transaction streams
  - SMMUv1/v2 programming interface
  - SMMUv3 programming interface
  - Translation process
  - MMU-500

- TrustZone Address Space Controller
  - TZASC
  - Trusted video path

- Generic Timer
  - Generic timer overview
  - Programming (for EL1/0)
  - Virtual count and timer

- Heterogeneous Systems
  - Address space
  - Using a SMMU
  - Interrupts

- **DynamIQ Debug**

- Introduction to Debug
  - Types of debug: introduction
  - Invasive & non-invasive debug
  - Intrusive & non-intrusive debug

- Debug Facilities
  - External debug
  - Self-hosted debug
  - CoreSight
  - DAP block diagram

- Debug Features
  - Halting cores
  - What is a Cross Trigger Interface (CTI & CTM)
  - Cross trigger examples
  - Instruction and data transfer
  - Viewing memory via core
  - Viewing memory via MEM-AP
  - Debugger impact on performance
  - Instruction breakpoints (hardware)
  - Instruction breakpoints (software)
  - Watchpoints
  - Hardware single step
  - Debug reset and power down
  - Exception & reset catch
  - Performance monitoring hardware
  - Performance monitoring events
  - PC sampled-based profiling

- Trace
  - Trace overview
  - Instruction trace
  - Example system

- Debug Authentication

❖ ARMv8-A Advanced SIMD & Floating-Point

- Advanced SIMD & FP Introduction
  - What is NEON?
  - Why program for NEON?
  - Support in A64, A32, T32 ISA
  - Power considerations

- Programmer’s Model
  - NEON registers
  - Data sizes
  - Data types
  - Register and element size
  - Vectors and scalars
  - Specifying data types
  - Instruction “shape”
  - Long and narrow operations
  - Instruction “modifiers”
  - Floating point HP, SP and DP
  - Enabling Floating-point in software
  - NEON status registers

- NEON Software Support
  - How to use NEON?
  - What is project Ne10?
  - Why use project Ne10?
  - Automatic vectorizing
  - Tuning C/C++ code for vectorising
  - NEON vectorising example
  - Intrinsics functions