Cortex-A35 MPCore Software Development

Course Description

Cortex-A35 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-A35 processors.

The course starts with a quick review of the ARMv7-A architecture, then introduce the new 64-bit architecture, instruction set, and the new exception model to handle interrupts and exceptions.

The course continues by covering the Cortex-A35 MPCore architecture, memory management unit, memory model, cache and branch prediction, cache coherency, processes synchronization, boot process, barriers, virtualization, Generic Interrupt Controller (GIC), NEON coprocessor, power management, debug, security and OS support.

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-A35 architecture
2. Understand the main differences between ARMv7-A and ARMv8-A architectures
3. Become familiar with ARMv8-A instruction set
4. Understand the ARMv8-A exception model
5. Be able to configure and use the ARMv8-A MMU
6. Be familiar with ARMv8-A memory model
7. Be familiar with ARMv8-A caches and branch prediction
8. Understand ARMv8-A cache coherency features and how to configure them
9. Be able to boot Cortex-A35 MPCore system
10. Implement synchronization processes using ARM primitives to build mutex/semaphore
11. Be able to add barriers instructions to control program flow order
12. Be able to program the GIC
13. Become familiar with NEON coprocessor SIMD capabilities
14. Manage Cortex-A35 MPCore power modes
15. Be able to debug with invasive and non-invasive techniques
16. Become familiar with TrustZone infrastructure to build secured systems
17. Become familiar with Virtualization and its effect on the system
18. Embed AMP and SMP operating systems

Target Audience

Software engineers that would like developing software and BSP for platforms based on ARMv8-A Cortex-A35 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:

- ARMv7-A Review
- Cortex-A35 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-A35 MPCore
- Programming the GIC
- Neon Overview
- ARMv8-A Debug and Trace
- ARMv8-A Virtualization
- Cortex- A53 Power Management
- ARMv8-A Secure Environment using TrustZone

Day #1

❖ ARMv7-A Architecture Review

- Architecture Versions
  - Development of the ARM architecture
  - ARM Cortex processors (A/R/M)

- Registers and Instruction Sets
  - Data sizes and instruction sets
  - The ARM register set
  - Program status register
  - ARM, Thumb, Thumb2 instruction sets
  - AAPCS standard
  - Endianness

- Exception Model
  - Processor modes
  - Banking of registers
  - Taking an exception
  - Vector table

- Memory Model
  - Memory model review
  - Memory types
  - Data alignment

- Coprocessors
  - Coprocessors and their role
  - Example CP15 registers
  - Performance Monitoring Unit (PMU)

- Architecture Extensions
  - Floating point and NEON
  - Large Physical Address Extensions (LPAE)
  - TrustZone
  - Virtualization

❖ 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

- ARMv8-A Software Support
  - Linux
  - Filesystems
  - ARM foundation model
  - Open source tools support
  - ARM DS5

❖ **ARMv8-A AArch64 ISA Overview**

- Instruction Sets
  - AArch32
  - AArch64

- Register Set
  - General purpose registers
  - Register banks

- 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
  - Conditional operations
  - Floating point operations
  - Floating point conversions

- 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-A35 Processor Overview**

- Cortex-A35 Introduction
  - Cortex-A35 features
  - Cortex-A35 hardware configuration options
  - Software support
  - Cortex-A35 pipeline
  - PMU
  - Debug and trace
  - Generic timer architecture

- New features in Cortex-A35
  - New features since ARMv7-A CPUs
  - AArch64 privilege model
  - ARMv8-A advanced SIMD and FP
  - MMU support
  - AMBA 5 CHI bus architecture
  - ARMv8 debug
  - Cortex-A35 interrupt handling

Day #2

❖ **ARMv8-A 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-A 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
- Device memory
- Ordering of device accesses
- Specifying the type

➢ Memory Attributes
- Cacheability
- Shareable
- Access permissions
- Executable
- Access flag
- Secure or non-secure (NS attribute)

➢ Alignment & Endianess
- Alignment
- Endianness in AArch64

❖ **ARMv8-A Caches & Branch Prediction**

➢ Caches in Cortex-A series processors
- How is data stored in my cache?
- How are caches accessed?
- Level 1 and level 2 cache interaction
- Branch prediction resources

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

- Cache Maintenance Operations
  - Cache maintenance
  - PoU and PoC and cache maintenance
  - PoU and PoC compared
  - AArch64 instructions
  - Maintenance broadcast
  - Maintenance broadcast – Aarch64

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

❖ **ARMv8-A Cache Coherency**

- Introduction to Coherency
  - What is cache coherency?
  - Software and hardware coherency
  - Evolution of ARM coherency support
  - Cortex-A family coherency
  - ACE system level coherency
  - System coherency with GPUs and DMA
  - Shareability in the translation tables
  - Shareability and software
  - Example: big.LITTLE implementation

- 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
  - Barriers
  - Barriers and ACE

Day #3

❖ Barriers in ARMv8-A

- 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
  - Mail box example
  - Speculation across barriers
  - Memory mapped peripherals
  - “One-Way” barriers

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

- Compiler Barriers

❖ ARMv8-A Synchronization

- Introduction to Synchronization
  - The Race for Atomicity
  - Critical Sections
  - Simple lock implementation
- Synchronization in ARMv8-A
  - Synchronization instructions
  - Exclusive monitor
  - AArch64 example lock/unlock
  - Multi-thread lock example

- Local and Global Exclusive Monitors
  - Where is the exclusive monitor?
  - Context switching
  - Granularity of exclusive monitor

- Coherent lock example
- Other considerations
- AArch64 Linux example
- AArch32 Linux example

❖ **ARMv8-A OS Support Features**

➢ Context Switching
- Split translation tables
- Global/non-global translations
- Address Space Identifiers (ASIDs)
- Thread registers
- ASID with short descriptor format

➢ Modifying Translation Tables
- Translation table change example
- Caching translation tables
- Access flag
- Reserved bits
- Tagged pointers

➢ Privilege Escalation Protections
- Executable
- Unprivileged loads and stores

➢ Timers
- System timer in Cortex-A35 MPCore
- System timer registers (for EL1/0)

➢ Linux Examples
- Linux table update example
- Linux access flag example

❖ **Cortex-A35 Booting**

➢ Overview
- Booting considerations

➢ Booting a Cortex-A35 Processor in AArch64
- Booting and the processor state
- AArch64 boot sequence mapping
- AArch32 boot sequence mapping
- What happens at reset in AArch64?
- Entering lower exception levels

➢ Processor Setup
- Processor setup after reset

- Vector tables and stacks
- Floating point and advanced SIMD
- Caches and MMU
- MP booting process
- How to identify the CPU
- Entering lower exception levels procedure

Day #4

❖ **ARMv8-A Generic Interrupt Controller Programming**

➢ GIC Architecture
- GIC architecture overview
- Interrupt types (SGI, PPI, SPI, LPI)
- Interrupts IDs on Cortex-A35

➢ Distributor and CPU Interfaces
- Register interfaces
- Distributor interface
- CPU interface
- Programming guidelines

➢ How to Enable and Configure Interrupts
- Enabling the IC
- Interrupt configuration

➢ How to Handle Interrupts
- Interrupt states
- Taking an interrupt
- Which CPU services an SPI?
- Priority mask register
- Interrupt priority registers
- Interrupt enable registers
- Interrupt configuration
- Running priority
- Group priority and Pre-emption
- Binary point register
- Nesting interrupts
- Lock down support

➢ How to Send Software Interrupts
- SGI capability
- Sending a SGI

- Receiving a SGI

- Security Extensions
  - Group 0 and Group 1
  - Acknowledging interrupts
  - Priority and banking

❖ **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

❖ **ARMv8-A Debug**

- Debug Overview
  - Debug infrastructure
  - How do I access the debug logic?
  - Types of debug (invasive, non-invasive)
  - Hosted vs self-hosted
  - Viewing memory
  - Debugger impact on caches

- Instruction breakpoint types
- Breakpoint comparison
- Exception catch (vector catch)
- Performance monitoring hardware
- Events and filtering
- Trace introduction
- Trace sinks (ETB, TPIU)
- Example trace system
- Hardware comparators
- Embedded cross trigger- CTI
- Debugger semi-hosting support

❖ **ARMv8-A Virtualization**

➢ Virtualization Overview
- What is virtualization
- Why virtualization?
- Type 1 and 2 hypervisors

➢ 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 – Virtual Interrupts
- Virtual interrupt signaling (GIC)
- Virtual interrupt signaling (internal)

➢ Introduction to SMMU
- Why do we need a SMMU?
- What is a SMMU?
- MMU-40x overview
- MMU-500 overview

❖ **Power Management for Cortex-A Processors**

- Power Overview
  - Power consumption
  - Example power contributions
  - Power reduction techniques
  - Example power domains
  - Additional power modes/interconnect
  - Energy cost

- Processor Power Modes
  - ARM processor power modes
  - Processor standby mode
  - Standby use cases and considerations
  - Using WFE to enter standby
  - Processor power down
  - Power down example
  - Barriers and power down modes
  - Entering power modes
  - Point of no return

- Multiprocessor and System Power Modes
  - Multiprocessor power modes
  - L2 cache power considerations
  - Power down mode examples
  - Preparing the L2 cache for power down
  - SoC and system power down

❖ **ARMv8-A Secure Environments**

- TrustZone Overview
  - What is TrustZone?
  - Why do we need TrustZone?
  - What kind of attacks are there?
  - What does it add?
  - TrustZone is not…

- Software Stack
  - Software stack
  - How the worlds communicate?
  - Secure monitor
  - Trusted boot
  - Multi-core topology
  - Scheduling of secure world

- Memory System
  - Physical address space

- Caches and TLBs
- Example system

- Debug
  - Debug configuration
  - Debug authentication

- TBSA
  - What is TBSA?
  - TBBR functionality