



OREN HANDS ON TRAINING & DEVELOPMENT LTD.
20 YAIR ROZENBLUM ST. KFAR SABA 4464601 ISRAEL



Cortex-A53 MPCore & Cortex-R5 Software Development

Course Description

Cortex-A53 MPCore & Cortex-R5 software optimization is a 5 days ARM official course.

The course goes into great depth and provides all necessary know-how to develop software for systems based on Cortex-A53 & optimize code for Cortex-R5 processors and NEON.

The course starts with an overview of the 64-bit architecture, instruction set, and the new exception handling model.

The course continues by covering the Cortex-A53 MPCore architecture, memory management unit, memory model, cache and branch prediction, cache coherency, processes synchronization, barriers, Generic Interrupt Controller (GIC), NEON coprocessor, and debug.

The NEON programming & optimization is also included in this training and teach how to take advantage of NEON to optimize common signal processing functions used in Filtering, Sample Rate Conversion, and Audio and Video Codec applications.

Then the training covers the Cortex-R5 compiler & linker optimizations as well as how to write an efficient C code.



When innovation meets expertise...



OREN HANDS ON TRAINING & DEVELOPMENT LTD.
20 YAIR ROZENBLUM ST. KFAR SABA 4464601 ISRAEL

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

Course Duration

5 days

Goals

1. Become familiar with ARMv8-A Cortex-A53 architecture
2. Become familiar with ARMv8-A instruction set
3. Understand the ARMv8-A exception model
4. Be able to configure and use the ARMv8-A MMU
5. Be familiar with ARMv8-A memory model
6. Be familiar with ARMv8-A caches and branch prediction
7. Understand ARMv8-A cache coherency features and how they work
8. Implement synchronization processes using ARM primitives to build mutex/semaphore
9. Be able to add barriers instructions to control program flow order
10. Be able to program the GIC
11. Become familiar with the ARMv7-R and Cortex-R5 architecture
12. Be able to debug with invasive and non-invasive techniques
13. Optimize C code for Cortex-R5 and use compiler & linker features



When innovation meets expertise...



OREN HANDS ON TRAINING & DEVELOPMENT LTD.
20 YAIR ROZENBLUM ST. KFAR SABA 4464601 ISRAEL

Target Audience

Software engineers that would like developing software and BSP for platforms based on ARMv8-A Cortex-A53 MPCore and ARMv7-R Cortex-R5 processors for Xilinx US+ FPGAs.

Prerequisites

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

Course Material

- ARM official course book



When innovation meets expertise...

Agenda

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

❖ Cortex-A53 Processor Overview

➤ Cortex-A53 Introduction

- Cortex-A53 features
- Cortex-A53 hardware configuration options
- Software support
- Cortex-A53 pipeline
- PMU
- Debug and trace
- Generic timer architecture

➤ New features in Cortex-A53

- 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-A53 interrupt handling

❖ 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



When innovation meets expertise...

- 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

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 & Endianness**
 - 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**



When innovation meets expertise...

❖ 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 Generic Interrupt Controller Programming

➤ GIC Architecture

- GIC architecture overview
- Interrupt types (SGI, PPI, SPI, LPI)
- Interrupts IDs on Cortex-A53

➤ 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

Day #4

❖ **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)



When innovation meets expertise...

- Example trace system
- Hardware comparators
- Embedded cross trigger- CTI
- Debugger semi-hosting support

❖ **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 Secure Environments**

- TrustZone Overview
 - What is TrustZone?
 - Why do we need TrustZone?



When innovation meets expertise...

- 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

Day #5

❖ **Cortex-R5 C/C++ Compiler Hints & Tips**

- **Writing C for ARM**
 - Parameter passing
 - Parameter passing
 - Passing more than 4 parameters
 - Parameter alignment
 - Floating point linkage
 - HW and SW floating point linkage
 - Floating point linkage example



OREN HANDS ON TRAINING & DEVELOPMENT LTD.
20 YAIR ROZENBLUM ST. KFAR SABA 4464601 ISRAEL

- Alignment
 - Global data layout
 - Unaligned accesses
 - Packing and alignment of structures
 - Alignment of pointers
- Coding considerations
 - Size of local variables
 - Division
 - Base pointer optimization
 - Using “volatile”
- **C/C++ Compiler Hints & Tips**
 - Basic compilation
 - Language support
 - Variables types supported
 - Optimization levels
 - Selecting an architecture or processor
 - Compiler optimizations
 - Automatic optimizations
 - Tail-call optimization
 - Instruction scheduling
 - Idiom recognition
 - Inlining of functions
 - Loop transformation
 - Coding considerations
 - Register usage
 - Loop termination
 - Division operations
 - Reminders-Modulo arithmetic
 - C++ support
 - Local and global data issues
 - Variable types
 - Global data layout
 - Optimization of memcpy()



When innovation meets expertise...

❖ Linker & Libraries Hints & Tips

➤ Linker & Libraries Hints & Tips

- Linking basics
 - How does the linker know what to do?
 - Object file structure
 - Library structure
 - Scatter-loading
- System and user libraries
 - Libraries versus object files
 - Linker library searching
 - Creating and maintaining libraries
- Veneers and interworking
 - Dealing with branches
 - Linker generated veneers
 - Veneer types
 - Minimizing the number of veneers
- Linker optimizations and diagnostics
 - Unused section elimination
 - RW data compression
 - Small function inlining
 - Useful linker diagnostics
- ARM supplied libraries
 - ARM compiler standard libraries
 - Microlib
- Mixing C/C++ and Assembler
 - Register usage revisited
 - Mixing C and Assembly
 - Calling Assembly from C/C++
 - Intrinsics
 - Embedded Assembler
 - Inline Assembler
- Stack issues
 - Protecting and measuring stack usage
 - --callgraph example (Dhrystone)



- VFP
 - Floating point capabilities
 - Floating point linkage
- Advanced building facilities
 - Multifile compilation
 - Linker feedback
 - Linking for specific target
 - Debug issues & build time
 - BPABI & SysV

❖ Appendix

❖ ARMv7-R ISA Overview

➤ ARM Assembler File Syntax

➤ Load/Store Instructions

- Single/Double register data transfer
- Addressing memory
- Pre and post-indexed addressing
- Multiple register data transfer

➤ Data Processing Instructions

- Arithmetic, logical, move instructions
- Shift/Rotate operations
- The flexible second operand
- Instructions for loading constants
- Multiply/Divide
- Bit manipulation instructions
- Byte reversal

➤ Flow Control Instructions

- Branch instructions
- Interworking
- Compare and branch if Zero
- Condition codes and flags
- If-Then instruction
- Supervisor call instruction (SVC)

➤ Miscellaneous Instructions

- Coprocessor instructions



When innovation meets expertise...

- PSR access
- Breakpoint instruction (BKPT)
- Wait for interrupt instruction (WFI)
- NOP instruction
- Wait for event & send event instructions (WFE & SEV)

➤ **DSP Instructions**

- SIMD
- Saturated maths and CLZ
- Data packing/unpacking

❖ **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