



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

altera™ solution
acceleration partner



Cortex-A76 & Cortex-A55 MPCore Software Development for Agilex 5 SoC

Course Description

This course is a unique crafted combination of content from Altera, Arm and HandsOn-Training to cover all of the essential know-how, highlighting the Agilex 5 SoC architecture and capabilities as well as the Cortex-A76 and Cortex-A55 MPCore and their configuration options.

The training is architected for embedded engineers and system designers, bridges the gap between hardware development in Quartus Prime and software implementation in Linux. You will move from high-level architecture to hands-on bitstream generation, multi-stage booting, and Yocto-based OS builds. Security and other advanced topics are also covered. The training has an extensive hands-on lab to experience with the theory topics.

The training starts by overview the Agilex 5 SoC FPGA architecture with an emphasis on the Hard Processor System (HPS), as well as the configuration and boot schemes with the first and second bootloaders along with Linux OS.

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-A76 and Cortex-A55 MPCore architecture based on DynamIQ technology, memory management unit, memory model, cache and



When innovation meets expertise...



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

branch prediction, cache coherency, processes synchronization, barriers, Generic Interrupt Controller (GIC), debug, and TrustZone, as well as other security features.

Each attendee receives an official certificate from Altera and from Arm (Exam must be passed).

Course Duration

5 days

Goals

- Understand the architecture for Altera SoC FPGAs
- Configure the features for the HPS IP in Platform Designer
- Become familiar with configuration and booting stages using Linux
- Become familiar with ARMv8-A instruction set
- Understand the ARMv8-A exception model
- Be able to configure and use the ARMv8-A MMU
- Be familiar with ARMv8-A memory model
- Be familiar with ARMv8-A caches and branch prediction
- Understand ARMv8-A cache coherency features and how to configure them
- Implement synchronization processes using ARM primitives to build mutex/semaphore
- Be able to add barriers instructions to control program flow order
- Be able to program the GIC
- Be able to debug with invasive and non-invasive techniques
- Become familiar with TrustZone infrastructure to build secured systems

Intended Users

Software engineers that would like developing software and BSP for Agilex 3 SoC platforms, based on ARMv8.2-A Cortex-A76 and Cortex-A55 MPCore processors.

Prerequisites

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



When innovation meets expertise...



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

Course Material

1. Course book
2. Labs handbook with lab files
3. Quartus Prime Pro
4. Intel® Simics® Simulator for Linux OS v25.1
5. VLP Session with Linux OS Ubuntu 20.04 environment



When innovation meets expertise...

Table of Contents

Day #1

❖ Introduction to Altera SoC FPGA Hardware

- Altera SoC FPGA families
- Applications of an FPGA in an Embedded system
- SoC FPGA key features and comparison
 - Stratix 10 and Agilex 7 SoC FPGAs
 - Agilex 5 SoC FPGAs
 - Agilex 3 SoC FPGAs
- Development roles
- How to best started on your design
- How to access software resources

❖ Hard Processor System (HPS)

- HPS at a high level
- Features and functionality of the HPS
- How do I use the features of the MPU?
- Arm cortex comparison
- Arm Cortex-A76 and Cortex-A55 features
- Arm DynamIQ Shared Unit (DSU) features
- Multi-core processing unit
- Memory Management Unit (MMU)
- Snoop control unit
- Data processing unit
- Hardware and software development
- Data movement in the HPS
- What is SMMU?
- Cache Coherency Unit (CCU)
- Generic Interrupt Controller (GIC) features
- Main interconnect
- SDRAM controllers
- PSS NOC & MPFE NOC



When innovation meets expertise...



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

- Bridges system integration in Agilex 5 SoC FPGA
- Design guidelines for FPGA/HPS connectivity
- Interface peripherals in the HPS
- MIPI D-PHY IP
- USB 3.1 controller features
- Using the peripherals
- Resets
- Peripherals in GHRD
- Quartus Prime device support
- Typical hardware design flow
- Open the design in Platform Designer
- The HPS component
- FPGA interface tab
- HPS to FPGA bridges
- DMA peripheral request
- Interrupts
- Input clocks
- PLL clocks
- Power and reset
- SoC SDRAM IP
- IO Delays tab
- Pin Mux and Peripherals Tab
- Advanced pin mux settings
- The HPS component in the system
- Connect components



When innovation meets expertise...

Day #2

❖ Configuration & Booting

- Why use a bootloader?
- Boot definitions
- The Secure Device Manager (SDM)
- HPS mailbox
- SDM & Booting
- HPS boot stages
- Configuration file generator
- Using QSPI for all boot storage
- Booting configuration
- First and second stage bootloaders
- Prepare the environment
- Commands for working with the toolchain
- Arm Trusted Firmware
- Booting flows supported
- Build Arm Trusted Firmware
- U-BOOT and SPL
- Obtaining U-Boot and making Config file
- Get source from repository
- Choose the appropriate defconfig file
- Compile U-Boot to obtain FSBL and SSBL
- Append FSBL to the FPGA sof file
- U-Boot user interface
- Single image boot
- Booting detailed walk-through, FPGA first
- What changes when booting HPS first?
- Possible locations of the SSBL

❖ The Linux OS & Building a Boot Image

- User space and kernel space
- Linux OS: "Everything is a file"
- Physical pieces of the Linux OS: the kernel



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

- Physical pieces of the Linux OS: the device tree
- Physical pieces of the Linux OS: the root filesystem
- Possible tasks involving Linux OS
- Short intro to drivers
- Building Linux OS
- The Yocto project
- Building with Yocto
- Yocto Pros and Cons
- Build recommendations
- Boot disk partition layout
- QSPI image + SD card layout

❖ **Security**

- Security states of the Arm Cortex-A76 and Cortex-A55
- Secure device manager
- Agilex 5 security features
- Implementing security features

- **Lab #1: Using the SoC FPGA in Quartus Prime Software**
- **Lab #2: Generating the First & Second Bootloaders**
- **Lab #3: Creating the Directories to Prepare a Boot Image**
- **Lab #4: Using Agilex 5 SoC with the Intel Simics Simulator**



When innovation meets expertise...

Day #3

- ❖ **ARMv8-A Architecture Overview**
 - 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

- ❖ **AArch64 A64 ISA**
 - 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**



When innovation meets expertise...

- 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

❖ **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-A76 & Cortex-A55 Processor in Details**

- Cortex-A76 CPU & 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

Lab #5: Handling simple and nested interrupts use cases

Lab #6: Handling synchronous exceptions

Lab #7: Configuring the GIC to optimize interrupt handling

Day #4

❖ Memory Management

- 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
- ❖ **Memory Model**
 - 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
 - Stronger to weaker device memory
- ❖ **Memory Attributes**
 - Cacheability
 - Shareable
 - Normal memory behavior guarantees
 - Access permissions
 - Executable
 - Access flag
 - Global/non-global translations
 - ASIDs
 - Reserved bits
 - Physical address spaces
 - Secure or non-secure (NS attribute)
- ❖ **Alignment & Endianness**
 - Alignment
 - Endianness in AArch64



When innovation meets expertise...

❖ **Tagged Pointers**

- Tagged pointers (AArch54 only)

❖ **ARMv8.2**

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

❖ **Caches & Branch Prediction**

- How is data stored in my cache?
- How are caches accessed?
- Level 1 and level 2 cache interaction
- Branch prediction

❖ **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
- Persistent memory (AArch64 only)
- AArch64 instructions
- Maintenance broadcast
- Maintenance broadcast – Aarch64

❖ **Other Cache Features**

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

❖ **Cache Coherency**

- What is cache coherency?
- 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

Lab #8: Configure the MMU with various attributes and memory types

Lab #9: Use cache instructions to optimize use cases

Lab #10: Experience with Cortex-A76 and Cortex-A55 Multi-core cache coherency

Day #5

❖ **Barriers**

- 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

❖ **Armv8.2-A Extensions**

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

❖ **Compiler Barriers**

❖ **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
- ❖ **ARMv8-A Secure Environments**
 - What are we protecting
 - 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
- ❖ **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)



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

- 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**

Lab #11: Use barriers instructions within a driver code

Lab #12: Use exclusive accesses to develop a mutex with priorities

Lab #13: Experience with a various debug features



When innovation meets expertise...