Embedded Software Learning Center

EMBEDDED IS EVERYWHERE.

HandsOn Training

arm APPROVED TRAINING PARTNER

Cortex-M55 Software Development

Course Description

Cortex-M55 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-M55 processor.

The course covers the Cortex-M55 architecture (Armv8.1-M), processor core, programmer's model, instruction set, CMSIS, exception handling, memory model, caches management, memory protection unit (MPU), MVE, synchronization, efficient C programming, compiler optimizations, linker optimizations, debug & trace, floating point and DSP instructions, security extension and safety features.

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

Course Duration

4 days

Goals

1. Become familiar with ARMv8.1-M architecture
2. Become familiar with Cortex-M55 architecture
3. Become familiar with ARMv8-M instruction set
4. Be able to handle interrupts and various exceptions
5. Be able to configure and use the MPU
6. Understand the memory model in v8-M architecture
7. Manage caches
8. Write an efficient C code for Cortex-M processor
9. Be able to debug your design
10. Become familiar with DSP and FP instructions
11. Optimize software for Cortex-M microcontrollers with the compiler and linker
12. Design a secured system with TrustZone for ARMv8-M
13. Use the MVE engine to enhance performance with vectorization

Target Audience

Software engineers that would like developing software and Firmware for platforms based on Cortex-M55 microcontroller.

Prerequisites

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

Course Material

- ARM official course book
- Labs handbook

Agenda

Main Topics:

- Cortex-M55 Overview
- ARMv8-M Mainline Programmer's Model
- Cortex-M55 Processor Core
- CMSIS Overview
- ARMv8-M Mainline Assembly Programming
- ARMv8-M Mainline Exception Handling
- ARMv8-M Mainline Memory Model
- ArmV8-M Mainline Memory Protection
- ARMv8-M Synchronization
- ARMv8-M Mainline Compiler Hints & Tips
- ARMv8-M Mainline Linker Hints & Tips
- ARMv8-M Embedded Software Development
- ARMv8-M Mainline Debug & Trace
- ARMv8-M Mainline DSP Extension
- ARMv8-M Mainline Floating-Point Extension
- ARMV8-M Mainline Security Extension
- ARMv8.1-M MVE (M-Profile Vector Extension)

Day #1

- **Cortex-M55 Overview**
  - Cortex-M55 processor block diagram
  - Architectural features and programmer's model
  - Processor core
  - ARMv8-M programmer's model register view
  - Modes of operation and execution
  - Memory map
  - Bus interfaces
  - Coprocessor interface
  - Reliability, availability, and serviceability (RAS) extension
  - Interrupts and exceptions
  - Memory protection
  - Security attribution
  - Security gating
  - Power management
  - Low power features
  - System timer
  - Extension processing unit (EPU)

- Debug
- Trace
- Configuration: synthesis and fusible
- RTL configuration
- Integration example

➢ **ARMv8-M Mainline Programmer's Model**
- ARMv8-M profile overview
- Data types
- Core registers
- Modes, privilege and stacks
- Exceptions
- Instruction set overview

➢ **ARMv8.1-M Overview**
- ARMv8.1-M extensions
- What does Cortex-M55 implement?
- New loops
- New security features
- RAS and FuSa features
- New debug features
- Half-precision floating point

➢ **Cortex-M55 L1 Sub-Systems**
- Tightly Coupled Memory (TCM) introduction
- TCM interfaces
- TCM configuration
- Control registers
- TCM transactions
- Accessing TCM via S-AHB
- Booting from TCM
- TCM protocol
- TCM timing
- Instruction & Data cache
- Cache maintenance
- Automatic cache invalidation
- Cache coherency
- Direct cache access registers
- System cache support

➢ **CMSIS Overview**
- Beneficial for the ARM Cortex-M ecosystem
- CMSIS partners
- CMSIS structure
- CMSIS bundle and documentation
- CMSIS-Core
- CMSIS-DSP

- CMSIS-Driver
- CMSIS-RTOS
- CMSIS-SVD
- CMSIS-Pack
- CMSIS-DAP

Day #2

- **ARMv8-M Mainline Assembly Programming**
  - Why do you need to know assembler?
  - Instruction set basics
  - Unified Assembler Language (UAL)
  - Condition codes and flags
  - Thumb instruction encoding choice
  - Data processing instructions
  - Load/Store instructions
  - Flow control
  - Miscellaneous instructions

- **ARMv8-M Mainline Exception Handling**
  - Exception architecture overview
  - Micro-coded interrupt mechanism
  - Interrupt overheads
  - Security extension – TrustZone for ARMv8-M
  - Exceptions model
  - Writing the vector table and interrupts handlers in C/C++ or Assembly
  - Internal exceptions and RTOS support
  - Fault exceptions

- **ARMv8-M Mainline Memory Model**
  - Introduction to ARMv8-M memory model
  - Memory address space and memory segments
  - Memory types and attributes
  - Endianness
  - Barriers

- **ARMV8-M Mainline Memory Protection**
  - Motivation: memory protection
  - Memory protection & security attribution
  - Default memory map
  - Memory Protection Unit (MPU)
  - Memory regions overview
  - Memory protection regions
  - Memory protection unit specification

- MPU registers
- Configuring the MPU
- Region programming
- MemManage faults

Day #3

- **ARMV8-M Synchronization**
- The need for atomicity
- The race for atomicity
- Critical sections
- Effective atomicity
- LDREX, STREX and CLREX instructions
- Example of lock() and unlock() functions
- Programs still have to be smart
- Example: multi-thread Mutex
- Non-coherent multiprocessor
- Memory attributes
- Configuring sharable memory
- Context switching
- Exclusives Reservation Granule (ERG)
- Example: Multiprocessor Mutex
- Weakly ordered memory and mutual exclusion
- Ordering with DMB
- Exclusive access with LDAEX/STLEX

- **ARMv8-M Mainline Compiler Hints & Tips**
- Compiler support for ARMv8-M
- Language and procedure call standards
- Compiler optimizations
    - Optimization levels
    - Selecting a target
    - Automatic optimizations
    - Using volatile to limit compiler optimizations
    - Tail-call optimization
    - Instruction scheduling
    - Idiom recognition
    - Inlining of functions
    - Loop transformation
    - Branch target optimization
    - Link time optimization
- Coding considerations
    - Loop termination
    - Division by compile-time constants

        - Modulo arithmetic
        - Floating point
    - Mixing C/C++ and assembler
        - Inline assembler
        - Intrinsics, libraries and extensions
        - CMSIS
    - Local and global data issues
        - Variable types
        - Size of local variables
        - Global RW/ZI data
        - Global data layout
        - Unaligned accesses
        - Packing of structures
        - Alignment of structures
        - Alignment of pointers
        - Optimization of memcpy()
        - Base pointer optimization

- **ARMv8-M Mainline Linker Hints & Tips**
    - Linking basics
    - System and user libraries
    - Veneers
    - Stack issues
    - Linker optimizations and diagnostics
    - ARM supplied libraries

- **MVE Introduction**
    - What is MVE?
    - Benefits of MVE
    - Vector Extension operation
    - Vector register file
    - VPR register
    - Lanes
    - Beats
    - Vector chaining
    - Exception state
    - Loop-tail predication
    - VPT predication
    - Interleaving and de-interleaving load and store
    - Vector gather load and scatter store
    - How to use MVE

Day #4

- **ARMv8-M Mainline Debug**
  - Introduction to Debug
  - Debug modes and security
  - Debug events and reset
  - Flash patch and breakpoint unit (FPB)
  - Data watchpoint and trace unit (DWT)
  - Instrumentation trace Macrocell (ITM)
  - Micro Trace Buffer (MTB)
  - Embedded Trace Macrocell (ETM)
  - Trace Port Interface Unit (TPIU)

- **ARMv8-M Mainline DSP Extension**
  - Extensions overview
  - DSP extension overview
  - SIMD instructions
  - Saturating arithmetic
  - SIMD multiplies
  - SIMD comparisons
  - ARMv8-M DSP instruction set

- **ARMv8-M Mainline Floating-Point Extension**
  - Floating point extension overview
  - Registers
  - Enabling the FPU
  - Floating point instructions
  - Exceptions
  - Basic versus extended frame
  - Lazy context save
  - Interaction with the security extension
  - VLSTM and VLLDM
  - Floating point conversion instructions

- **ARMv8-M Mainline Security Extension**
  - Introduction to TrustZone for ARMv8-M
  - Secure and non-secure states
  - Calling between security states
  - General purpose register banking
  - Special purpose register banking
  - Memory security
  - Secure memory rules
  - Memory security determination
  - Memory protection unit
  - Secure view of SCS
  - Non-secure view of SCS

- SAU registers
- Boot security map
- Runtime security map
- SAU region configuration
- Enabling the SAU
- Configuring the SAU with CMSIS
- Branching between secure and non-secure states
- Function calls using branch instructions
- ARM C Language Extensions (ACLE)
- Calling non-secure code from secure code
- Calling secure code from non-secure code
- Creating an import library in ARM compiler 6
- Using the import library
- Secure gateway veneers
- NSC veneers in ARM compiler 6
- TT instruction
- Security state changes using software
- Interrupts and exceptions
- Exception priorities overview
- System handler priority
- Secure exception prioritization
- Configuring the NVIC
- EXC_RETURN
- Taking an exception
- Secure -> non-secure exceptions
- Chaining secure and non-secure exceptions
- Stack frame layout
- Register values after context stacking
- Integrity signature
- Stack frame layout with floating point extension