Course Description

This course provides all necessary theoretical and practical know-how to enhance performance with the Kinetis family.

The course provides an in-depth overview of the key points that a software developer has to take into his considerations while developing software for Cortex-M4.

The course also includes practical labs in which the software engineer can experience with the optimization process on Tower and Freedom evaluation boards.

The course starts with an overview of the key elements that can be optimized such as buses, SRAM frontdoor and backdoor, arbitration, cache coherency, FMC configuration, cross bar switch (AXBS), Cortex-M4 processor, coding style, interrupts and exception handling, DSP, compiler and linker.

The course continues with in-depth code examples for each element, provides useful tips for real programming situations, and deals with mixing C and Assembler languages.

The course also covers the CMSIS (Cortex Microcontroller Software Interface Standards) and CMSIS-DSP. In addition RTOS implications are also discussed.

At the end of this course, the participant will enhance his programming development skills, his understanding of the architecture features, and optimize the Kinetis SoC key elements.
Course Duration

3 days

Goals

1. Optimize Kinetis system performance
2. Optimize code for Cortex-M4
3. Use efficiently optimization options in the compiler
4. Define optimized interrupt and exception handling strategy
5. Understand on-chip memory cons and pros and best location for code and data for the highest performance
6. Configure FMC in the most optimal way including cache policy
7. Use CMSIS and CMSIS-DSP libraries
8. Mix C and assembly languages
9. Write an efficient DSP code
10. Be familiar with OS support features

Intended Users

Software engineers who would like to enhance their programming skills, in order to get the maximum performance from the Kinetis SoC devices.

Prerequisites

1. Kinetis architecture
2. Embedded software development experience
3. Cortex-M background is recommended
4. C language
5. ARM Thumb/Thumb2 instruction set – not mandatory but recommended

Course Material

1. Hardware: K64 Tower evaluation board, K64 Freedom evaluation board, Keil Ulink-Pro or Ulink-ME
2. Software: Keil MDK version 5
3. Course and labs books
Table of Contents

Day #1

- **Kinetis System Optimization**
  - Introduction to system optimization
  - Challenges with Kinetis system optimization
  - Kinetis-K architecture overview
  - Core Busses on Kinetis (I-Code, D-Code, System, PPB)
  - Kinetis bus performance
  - Memory map implications on user code
  - On-chip memories optimization
    - On-chip SRAM access time implications on critical code
    - SRAM backdoor port and its affect
    - SRAM utilization design considerations
    - Choosing the right SRAM arbitration scheme
    - Kinetis System cache modes configuration and regions
    - SRAM vs system cache performance
    - Cache coherency considerations and recommendations
    - Configure the Flash Memory Controller (FMC) for optimal performance
  - Crossbar switch optimization
    - Crossbar switch (AXBS) access time and arbitration
    - Configure AXBS arbitration mode during bursts for high performance
    - AXBS parking configuration for high performance between master-slave
  - Memory Protection Unit (MPU) optimization
    - MPU overview
    - Kinetis family MPU
    - MPU master & slave assignments
    - MPU block diagram
    - MPU feature set
    - Kinetis MPU programmers model
    - MPU control/error status register
    - Error address and detail registers
    - Region descriptor word registers
- Region descriptor alternate access control
- Access evaluation macro
- When an error is generated?
- MPU configuration code example
- Guidelines for setting up the MPU
- MPU important considerations
- OS support
- Program code partitioning
- Data memory and peripherals partitioning
- Malicious code
- Stack protection
- Overlapping region descriptor example
  - Kinetis system optimization summary & tips

**Architectural Optimization**
- Cortex-M4 memory system optimization
  - Introduction to Kinetis memory system
  - Kinetis-K memory system features overview
  - Memory map in details
  - Memory endianness
  - Data alignment and unaligned access support
  - Aliased regions
  - How bit-band operation works
  - When to use bit-band operations
  - Bit-band vs non bit-band example
  - Default memory access permissions
  - Memory access attributes
  - Exclusive accesses for synchronization
  - Spin lock example
  - Context switch support
  - Exclusive access in C
  - When to use Memory barriers
  - Memory barriers in C

**Hands-On Labs**
- Lab #1: Apply optimal configuration to Kinetis SoC (including MPU)
- Lab #2: Use atomic bit banding operations for shared resources and synchronization
Day #2

- **Architectural Optimization**
  - Exception and interrupts handling optimization
    - Exception types review
    - Kinetis exceptions types
    - Kinetis-K interrupt vector assignments
    - Interrupt management registers
    - CMSIS-CORE interrupts & exceptions
    - NVIC code example
    - Interrupt behavior after reset
    - Kinetis interrupt priority levels
    - Group priority & sub-priority
    - Why using priority group?
    - Interrupt attributes
    - Interrupt pending and activation behavior example
    - Interrupt behavior scenarios
    - Exception entrance
    - Exception handler execution & exception return
    - EXC_RETURN bit fields & valid values
    - Using NVIC registers for interrupt control
    - System control block registers for exception control
    - Execution priority
    - Priority boosting, registers and options
    - Useful CMSIS-CORE functions for priority boosting
    - Exception handlers in C
    - Stacking process
    - Exception stacking with MSP and PSP
    - Interrupt latency
    - What can increase interrupt latency?
    - Restarting multiple-cycle instructions
    - Tail chaining optimization
    - Late arrival optimization
    - Pop preemption optimization
    - Lazy stacking on Cortex-M4F
• **Cortex-M Software Optimization**
  - Compiler optimization
    - armcc compiler overview
    - source language modes of the compiler
    - Compiler optimization levels & debug view (-O0/-O1/-O2/-O3/-Os)
    - Selecting the target CPU or architecture at compile time
    - Idiom recognition
    - Tail-call optimization & tail recursion
    - Instruction scheduling
  - Writing efficient C code for Cortex-M processors
    - Loop termination
    - Loop unrolling
    - Volatile optimization
    - Inline functions cons and pros
    - Inline/static/pure/const functions
    - Multifile compilation
    - Types of data alignment
    - Local variables
    - Global & static variables
    - Global data in memory
    - Unaligned data access
    - Unaligned fields in structures
    - Marking whole structures as __packed vs individual elements packing
    - __packed struct vs #pragma packed struct
    - Conditional branch optimization

• **Using Embedded RTOS Considerations**
  - Cortex-M OS support
  - OS support features overview
  - MSP and PSP stacks (RTOS)
  - How to use PSP?
  - Context switch
  - SVC mechanism
  - SVC handler in assembly and C
  - PendSV exception
  - Context switch using PendSV
  - PendSV without OS

When innovation meets expertise...
• Hands-On Labs
  o Lab #1: Design interrupt and exception strategy
  o Lab #2: Optimize code for Kinetis SoC

Day #3

• Assembly and Mixed Language Projects
  o When do we need assembly code in projects?
  o Inline & embedded assembler support
  o Inline assembly syntax
  o Embedded assembly syntax
  o Access C/C++ compile-time constant expressions
  o Differences between C/C++ and embedded assembler
  o Application binary interfaces
  o Why do we care about ABI/AAPCS?
  o What should we take care of?
  o Writing assembly function with and without stack
  o Export, Import and Extern
  o Calling a C function from assembly
  o Calling an assembly function from C
  o Intrinsic functions
  o Performance benefits of compiler intrinsics
  o Introduction to CMSIS
  o CMSIS structure
  o CMSIS-CORE intrinsic functions
  o How do I use CMSIS-CORE?
  o CMSIS partners
  o Kinetis Software Development Kit (SDK)
  o Kinetis SDK and Processor Expert
  o Kinetis SDK device and toolchain support
  o Kinetis Design Studio (KDS)
  o KDS device support
• Cortex-M4 DSP Optimization
  o Advantages of DSP on a microcontroller
  o Developing optimal DSP code challenges
  o MAC operation on Cortex-M4 vs traditional DSP
  o Advantages of Cortex-M4
  o Cortex-M4 DSP registers & data types
  o SIMD data
  o Cortex-M4 MAC & SIMD instructions
  o Idiom and intrinsic functions
  o DSP general optimization tips & hints
  o FIR filter example
  o Writing optimized DSP code for Cortex-M4
  o CMSIS-DSP library overview

• Cortex-M4 Floating Point Optimization
  o Floating point precision formats (IEEE 754)
  o Kinetis Floating Point Unit (FPU) features
  o Flash-to-Zero mode behavior in ARM
  o Kinetis-K with FPU FAQ
  o Floating point registers
  o Enabling the FPU code example
  o FPU system registers
  o FPU instruction set
  o Lazy stacking and interrupts
  o When the long stack frame is used?
  o FPU lazy stacking FAQ
  o CMSIS support for FPU
  o Enabling FPU in Keil MDK-ARM
  o Using the FPU with intrinsic functions
  o Compiler support for floating-point computation & linkage
  o Floating point programming in C
  o Floating point exceptions
  o Performance considerations in floating-point MAC instructions

• Hands-On Labs
  o Lab #1: Use CMSIS libraries, intrinsic functions and assembly code
  o Lab #2: Write efficient DSP and using fixed and floating-point