Embedded System Course

Lecture 4: Cortex Microcontroller Software Interface Standard

Lesson Objectives

- *Understanding basis concepts about Common Microcontroller Software Interface Standard (*CMSIS).

Table of contents

Section 1:

WHAT IS SOFTWARE INTERFACE ?

What is software interface

A software interface is used to allow either two pieces of software to communicate with each other (software-software interface), or to allow software to communicate with a hardware device (software-hardware interface).

(wiki)

Section 2:

Common Microcontroller Software Interface Standard (CMSIS) overview.

Common Microcontroller Software Interface Standard

- *Definition*
- *CMSIS components*
- *CMSIS Structure*
- *Coding Rules*

Definition

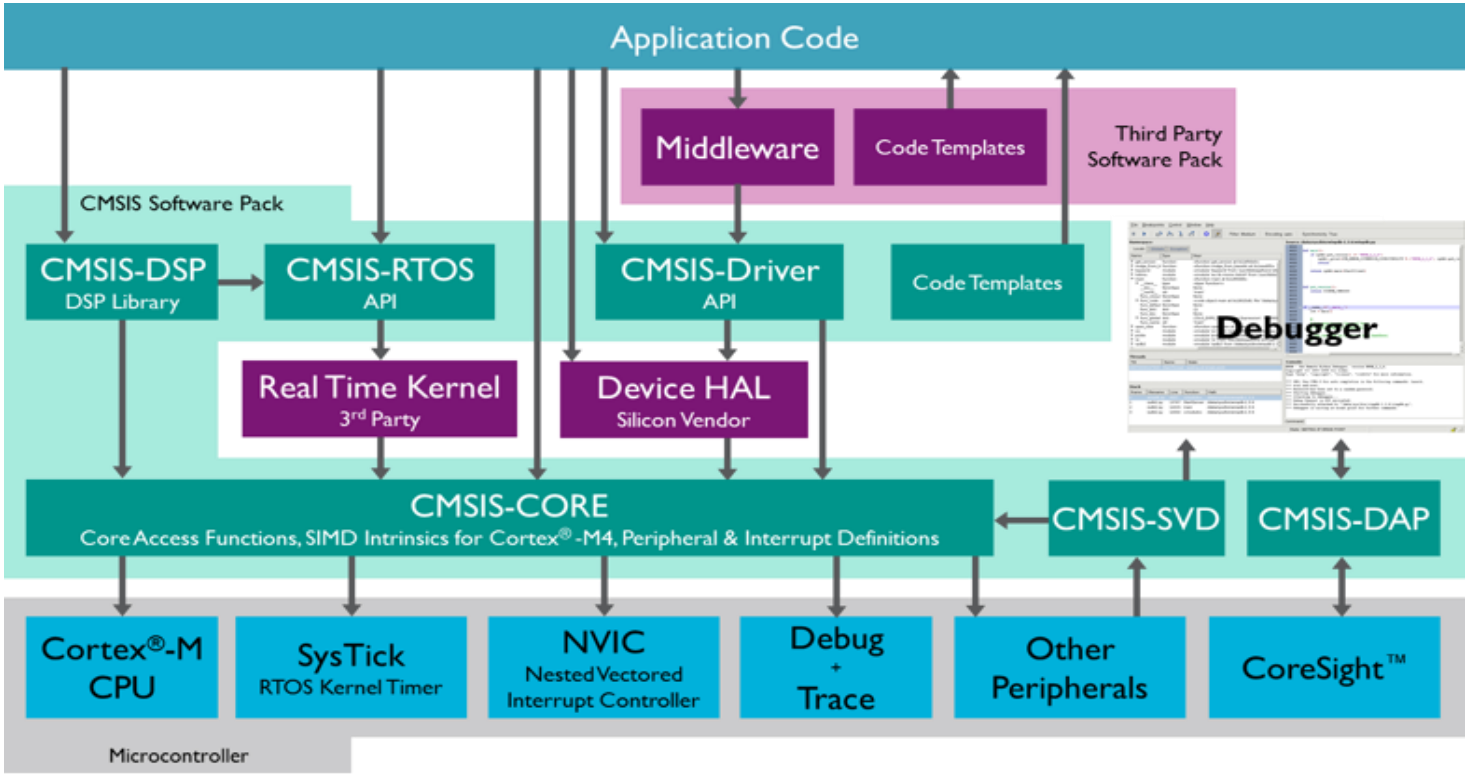
The **CMSIS** is a vendor-independent hardware abstraction layer for microcontrollers that are based on Arm® Cortex® processors. The CMSIS defines generic tool interfaces and enables consistent device support. It provides simple software interfaces to the processor and the peripherals, simplifying software re-use, reducing the learning curve for microcontroller developers, and reducing the time to market for new devices.

The CMSIS is defined in close cooperation with various silicon and software vendors and provides a common approach to interface to peripherals, real-time operating systems, and middleware components. The CMSIS is intended to enable the combination of software components from multiple middleware vendors.

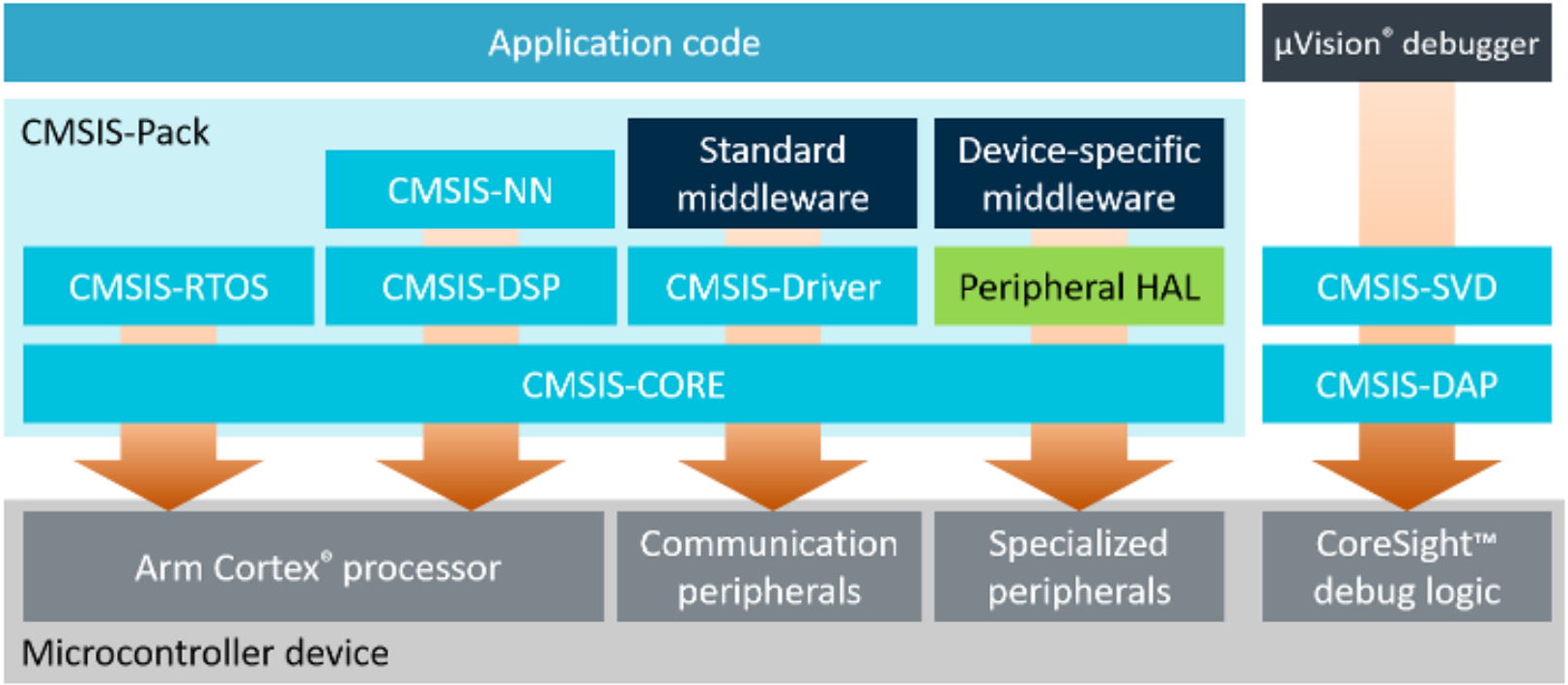
CMSIS components

CMSIS-...	Target Processors	Description
Core(M)	All Cortex-M, SecurCore	Standardized API for the Cortex-M processor core and peripherals. Includes intrinsic functions for Cortex-M4/M7/M33/M35P SIMD instructions.
Core(A)	Cortex-A5/A7/A9	Standardized API and basic run-time system for the Cortex-A5/A7/A9 processor core and peripherals.
Driver	All Cortex	Generic peripheral driver interfaces for middleware. Connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces.
DSP	All Cortex-M	DSP library collection with over 60 Functions for various data types: fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit). Implementations optimized for the SIMD instruction set are available for Cortex-M4/M7/M33/M35P.
NN	All Cortex-M	Collection of efficient neural network kernels developed to maximize the performance and minimize the memory footprint on Cortex-M processor cores.
RTOS v1	Cortex-M0/M0+/M3/M4/M7	Common API for real-time operating systems along with a reference implementation based on RTX. It enables software components that can work across multiple RTOS systems.
RTOS v2	All Cortex-M, Cortex-A5/A7/A9	Extends CMSIS-RTOS v1 with Armv8-M support, dynamic object creation, provisions for multi-core systems, binary compatible interface.
Pack	All Cortex-M, SecurCore, Cortex-A5/A7/A9	Describes a delivery mechanism for software components, device parameters, and evaluation board support. It simplifies software re-use and product life-cycle management (PLM).
Build	All Cortex-M, SecurCore, Cortex-A5/A7/A9	A set of tools, software frameworks, and work flows that improve productivity, for example with Continuous Integration (CI).
SVD	All Cortex-M, SecurCore	Peripheral description of a device that can be used to create peripheral awareness in debuggers or CMSIS-Core header files.
DAP	All Cortex	Firmware for a debug unit that interfaces to the CoreSight Debug Access Port.
Zone	All Cortex-M	Defines methods to describe system resources and to partition these resources into multiple projects and execution areas.

CMSIS Structure



CMSIS Structure



CMSIS Structure

CMSIS Structure

The benefits of the CMSIS are:

- CMSIS reduces the learning curve, development costs, and time-to-market. Developers can write software quicker through a variety of easy-to-use, standardized software interfaces.

- Consistent software interfaces improve the software portability and re-usability. Generic software libraries and interfaces provide consistent software framework.

- It provides interfaces for debug connectivity, debug peripheral views, software delivery, and device support to reduce time-to-market for new microcontroller deployment.

- Being a compiler independent layer, it allows to use the compiler of your choice. Thus, it is supported by mainstream compilers.

- It enhances program debugging with peripheral information for debuggers and ITM channels for printf-style output.

- CMSIS is delivered in CMSIS-Pack format which enables fast software delivery, simplifies updates, and enables consistent integration into development tools.

- CMSIS-Zone will simplify system resource and partitioning as it manages the configuration of multiple processors, memory areas, and peripherals.

Coding Rules

Coding Rules

The CMSIS uses the following essential coding rules and conventions:

- Compliant with ANSI C (C99) and C++ (C++03).
- Uses ANSI C standard data types defined in **<stdint.h>**.
- Variables and parameters have a complete data type.
- Expressions for *#define* constants are enclosed in parenthesis.
- Conforms to MISRA 2012 (but does not claim MISRA compliance). MISRA rule violations are documented.

In addition, the CMSIS recommends the following conventions for identifiers:

- **CAPITAL** names to identify Core Registers, Peripheral Registers, and CPU Instructions.
- **CamelCase** names to identify function names and interrupt functions.
- **Namespace_** prefixes avoid clashes with user identifiers and provide functional groups (i.e. for peripherals, RTOS, or DSP Library).

The CMSIS is documented within the source files with:

- Comments that use the C or C++ style.
- Doxygen compliant **function comments** that provide:
  - brief function overview.
  - detailed description of the function.
  - detailed parameter explanation.
  - detailed information about return values.

Doxygen comment example:

```
/**
 * @brief  Enable Interrupt in NVIC Interrupt Controller
 * @param  IRQn  interrupt number that specifies the interrupt
 * @return none.
 * Enable the specified interrupt in the NVIC Interrupt Controller.
 * Other settings of the interrupt such as priority are not affected.
 */
```

Section 3:

CMSIS-CORE (CORTEX-M)

Overview

CMSIS-Core (Cortex-M) implements the basic run-time system for a Cortex-M device and gives the user access to the processor core and the device peripherals. In detail it defines:

- Hardware Abstraction Layer (HAL) for Cortex-M processor registers with standardized definitions for the SysTick, NVIC, System Control Block registers, MPU registers, FPU registers, and core access functions.

- System exception names to interface to system exceptions without having compatibility issues.

- Methods to organize header files that makes it easy to learn new Cortex-M microcontroller products and improve software portability. This includes naming conventions for device-specific interrupts.

- Methods for system initialization to be used by each MCU vendor. For example, the standardized SystemInit() function is essential for configuring the clock system of the device.

- Intrinsic functions used to generate CPU instructions that are not supported by standard C functions.

- A variable to determine the system clock frequency which simplifies the setup the SysTick timer.

Using CMSIS in Embedded Applications

To use the CMSIS-Core (Cortex-M) the following files are added to the embedded application:

- *Startup File startup_<device>.c (formerly Startup File startup_<device>.s (deprecated)) with reset handler and exception vectors.*
- *System Configuration Files system_<device>.c and system_<device>.h with general device configuration (i.e. for clock and BUS setup).*
- *Device Header File <device.h> gives access to processor core and all peripherals.*

Startup File

- *The Startup File startup_<device>.c (or Startup File startup_<device>.s (deprecated)) is executed after reset and calls SystemInit. After the system initialization control is transferred to the C/C++ run-time library which performs initialization and calls the main function in the user code. In addition the Startup File startup_<device>.c (or Startup File startup_<device>.s (deprecated)) contains all exception and interrupt vectors and implements a default function for every interrupt. It may also contain stack and heap configurations for the user application.*

System Configuration Files

- *The System Configuration Files system_<device>.c and system_<device>.h performs the setup for the processor clock. The variable SystemCoreClock indicates the CPU clock speed. System and Clock Configuration describes the minimum feature set. In addition the file may contain functions for the memory BUS setup and clock re-configuration.*

The Device Header File

- *Peripheral Access provides a standardized register layout for all peripherals. Optionally functions for device-specific peripherals may be available.*
- *Interrupts and Exceptions (NVIC) can be accessed with standardized symbols and functions for the Nested Interrupt Vector Controller (NVIC) are provided.*
- *Intrinsic Functions for CPU Instructions allow to access special instructions, for example for activating sleep mode or the NOP instruction.*
- *Intrinsic Functions for SIMD Instructions [only Cortex-M4 and Cortex-M7] provide access to the DSP-oriented instructions.*
- *Systick Timer (SYSTICK) function to configure and start a periodic timer interrupt.*
- *Debug Access are functions that allow printf-style I/O via the CoreSight Debug Unit and ITM communication.*

CMSIS-Core Device Templates

The device configuration of the template files is described in detail on the following pages:

- **Startup File startup_<device>.c**
- **Startup File startup_<device>.s (deprecated)** (deprecated)
- **System Configuration Files system_<device>.c and system_<device>.h**
- **Device Header File <device.h>**

Q&A

Thank you