Chapter 4

CMSIS and Cortex-M4 CMSIS-DSP Programming

Introduction

In this chapter we overview the Cortex Microcontroller Interface standard (CMSIS) and move on to focus on efficient C programming for DSP.

CMSIS Overview

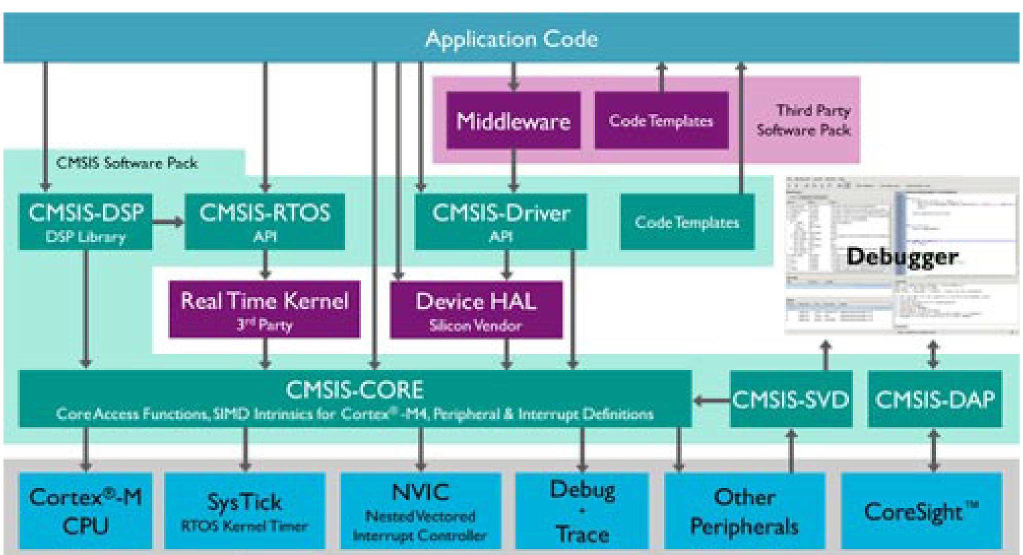
- CMSIS was created to portability and reusability across the M-series variants (M0 — M7) and development toolchains
- The CMSIS consists of the following components[1]:

- **CMSIS-CORE**: API for the Cortex-M processor core and peripherals. It provides at standardized interface for Cortex-M0, Cortex-M3, Cortex-M4, SC000, and SC300. Included are also SIMD intrinsic functions for Cortex-M4 SIMD instructions.
- **CMSIS-Driver**: defines generic peripheral driver interfaces for middleware making it reusable across supported devices. The API is RTOS independent and connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces.

1. http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php

- **CMSIS-DSP**: DSP Library Collection with over 60 Functions for various data types: fix-point (fractional q7, q15, q31) and single precision floating-point (32-bit). The library is available for Cortex-M0, Cortex-M3, and Cortex-M4. The Cortex-M4 implementation is optimized for the SIMD instruction set.
- **CMSIS-RTOS API**: Common API for Real-Time operating systems. It provides a standardized programming interface that is portable to many RTOS and enables therefore software templates, middleware, libraries, and other components that can work across supported the RTOS systems.
- **CMSIS-Pack**: describes with a XML based package description (PDSC) file the user and device relevant parts of a file collection (called software pack) that includes source, header, and library files, documentation, Flash programming algorithms, source code templates, and example projects. Development tools and web infrastructures use the PDSC file to extract device parameters, software components, and evaluation board configurations.
- **CMSIS-SVD**: System View Description for Peripherals. Describes the peripherals of a device in an XML file and can be used to create peripheral awareness in debuggers or header files with peripheral register and interrupt definitions.
- **CMSIS-DAP**: Debug Access Port. Standardized firmware for a Debug Unit that connects to the CoreSight Debug Access Port. CMSIS-DAP is distributed as separate package and well suited for integration on evaluation boards. This component is provided as separate download.

CMSIS v4



CMSIS Foundations

- Besides providing the interfaces listed above, the CMSIS provides/encourages overarching C coding rules
- In particular *MISRA C* (Motor Industry Software Reliability Association) is endorsed
  - The original MISRA standard was created in 1998 as guidelines for programming C in vehicle electronics
- A major impact for our purposes is C *type defs* to insure that the ANSI types are properly represented for a given compiler, e.g. CMSIS includes `stdint.h` which provides:

Standard ANSI C Type	MISRA C Type
`signed char`	`int8_t`
`signed short`	`int16_t`
`signed int`	`int32_t`
`signed __int64`	`int64_t`
`unsigned char`	`uint8_t`
`unsigned short`	`uint16_t`
`unsigned int`	`uint32_t`
`unsigned __int64`	`uint64_t`

- When using CMSIS-DSP and in particular floating point math (think Cortex-M4 and M7), more types are added via `arm_math.h`

MISRA /ANSI	MISRA C *like*	Description
`int8_t`	`q7_t`	8-bit fractional data type in 1.7 format.
`int16_t`	`q15_t`	16-bit fractional data type in 1.15 format.
`int32_t`	`q31_t`	32-bit fractional data type in 1.31 format.
`int64_t`	`q63_t`	64-bit fractional data type in 1.63 format.
`float`	`float32_t`	32-bit floating-point type definition.
`double`	`float64_t`	64-bit floating-point type definition.

- **Note**: To include `arm_math.h` in a project requires that you begin the includes section of a code module with

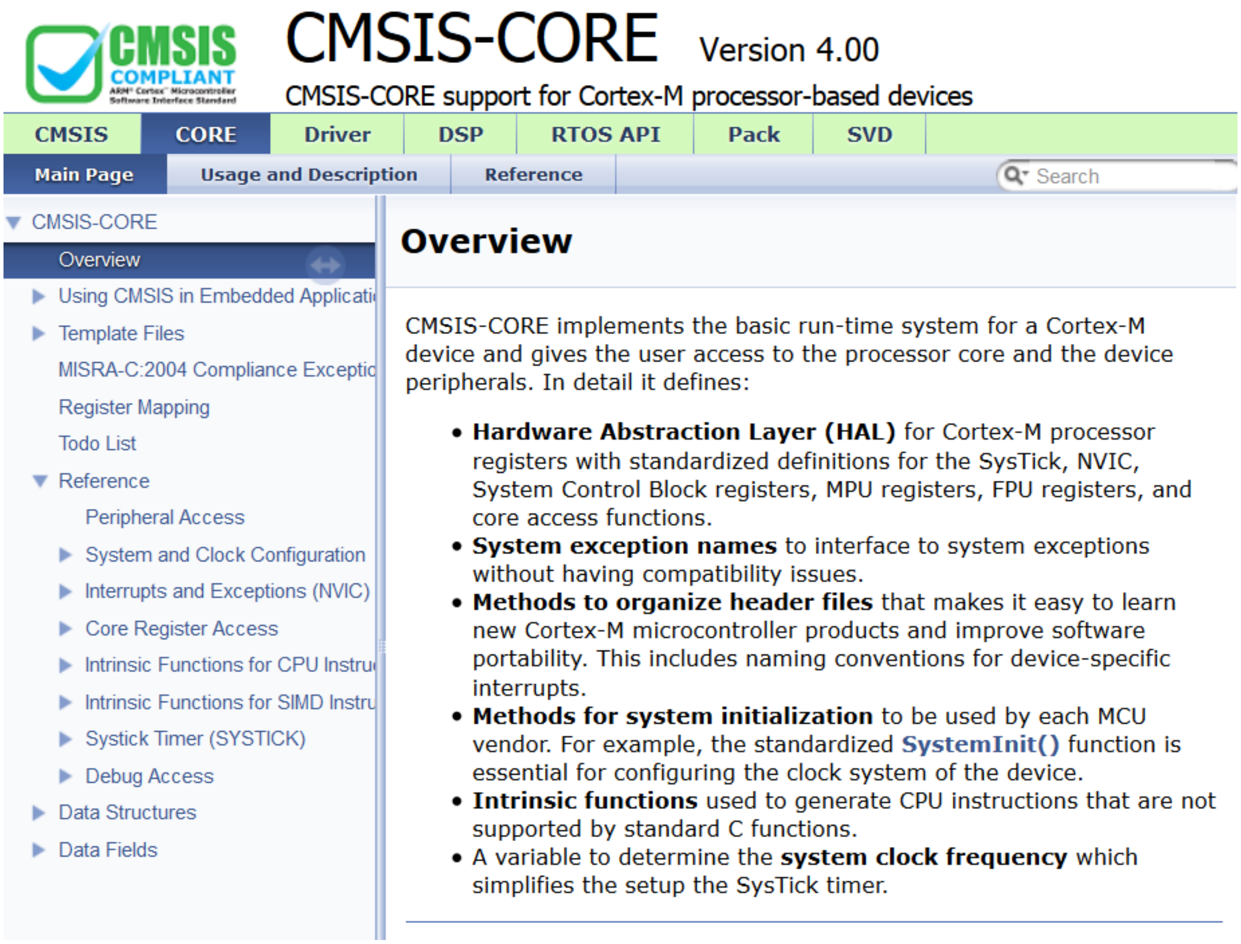
```
#define ARM_MATH_CM4
```

- In most projects we work with include the header

```
#include "stm32_wm5102_init.h"
//or on the LPC4088
#include "audio.h"
```

which takes of this

CMSIS-Core[1]



CMSIS COMPLIANT

CMSIS-CORE Version 4.00

CMSIS-CORE support for Cortex-M processor-based devices

CMSIS | CORE | Driver | DSP | RTOS API | Pack | SVD

Main Page | Usage and Description | Reference

Search

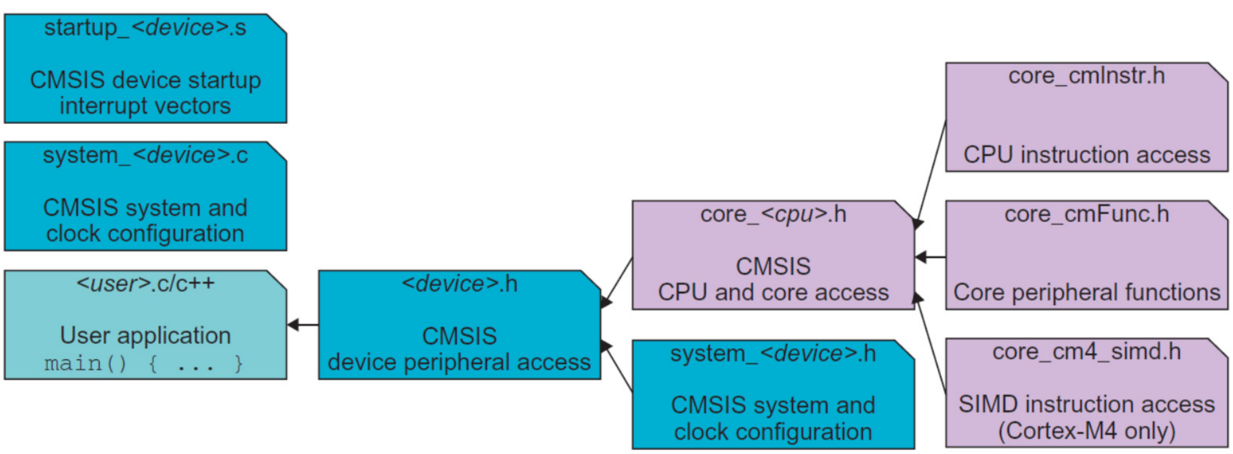
- CMSIS-CORE
  - Overview
  - Using CMSIS in Embedded Applicati
  - Template Files
  - MISRA-C:2004 Compliance Exceptic
  - Register Mapping
  - Todo List
  - Reference
    - Peripheral Access
    - System and Clock Configuration
    - Interrupts and Exceptions (NVIC)
    - Core Register Access
    - Intrinsic Functions for CPU Instru
    - Intrinsic Functions for SIMD Instru
    - Systick Timer (SYSTICK)
    - Debug Access
  - Data Structures
  - Data Fields

Overview

CMSIS-CORE 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:

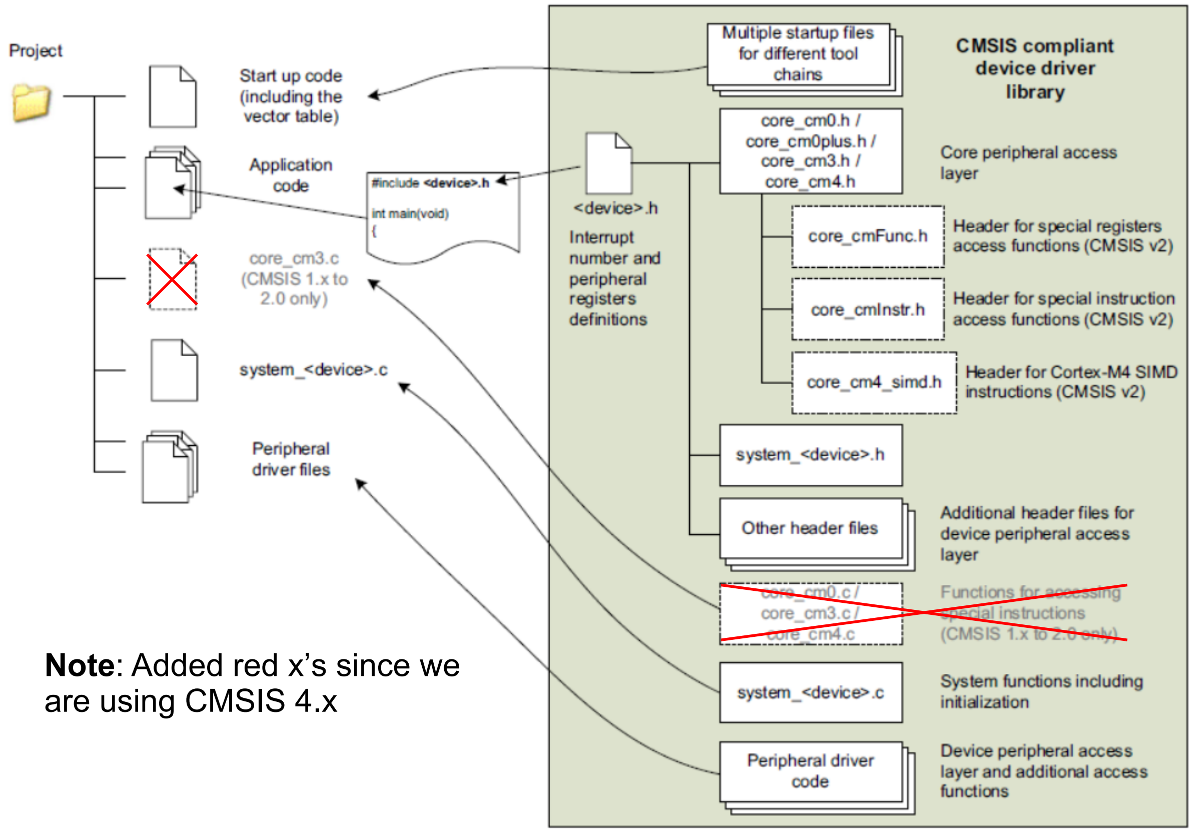
- **Hardware Abstraction Layer (HAL)** for Cortex-M processor registers with standardized definitions for the SysTick, NVIC, System Control Block registers, MPU registers, FPU registers, and core access functions.
- **System exception names** to interface to system exceptions without having compatibility 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.

- Files brought into a CMSIS core project:

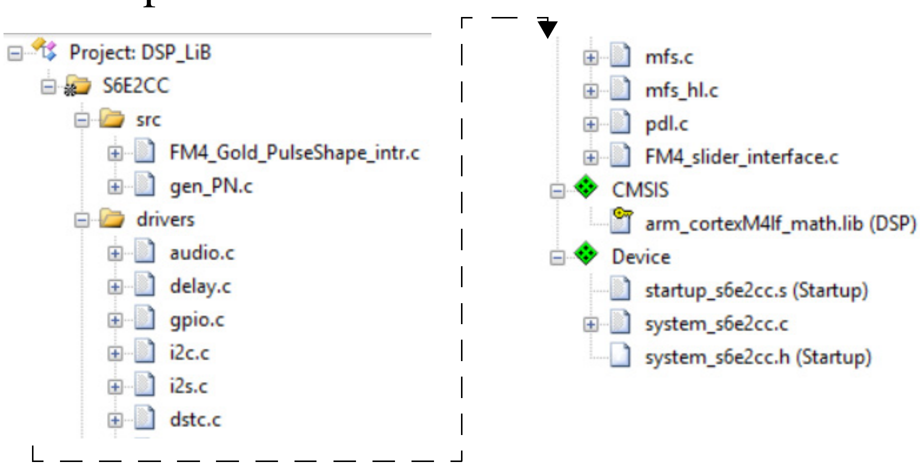


1. http://www.keil.com/pack/doc/CMSIS/Core/html/index.html

- In a generic project setting, Liu[1], depicts it a shown below:

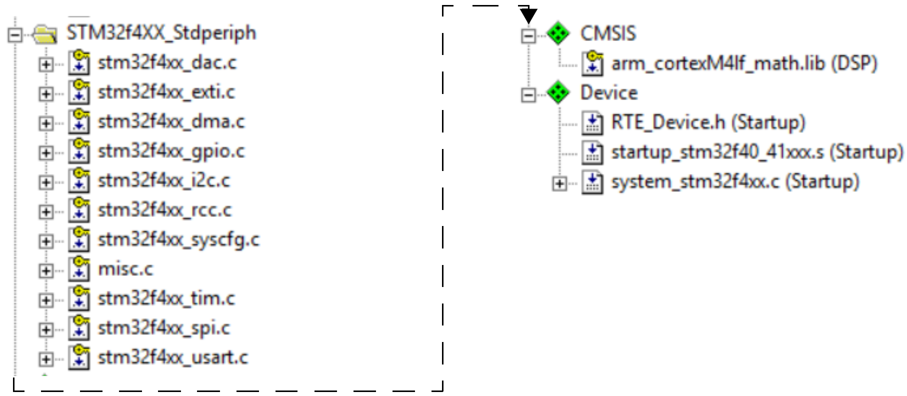


- Cypress FM4 projects we have been working with thus far, take the specific form shown below:



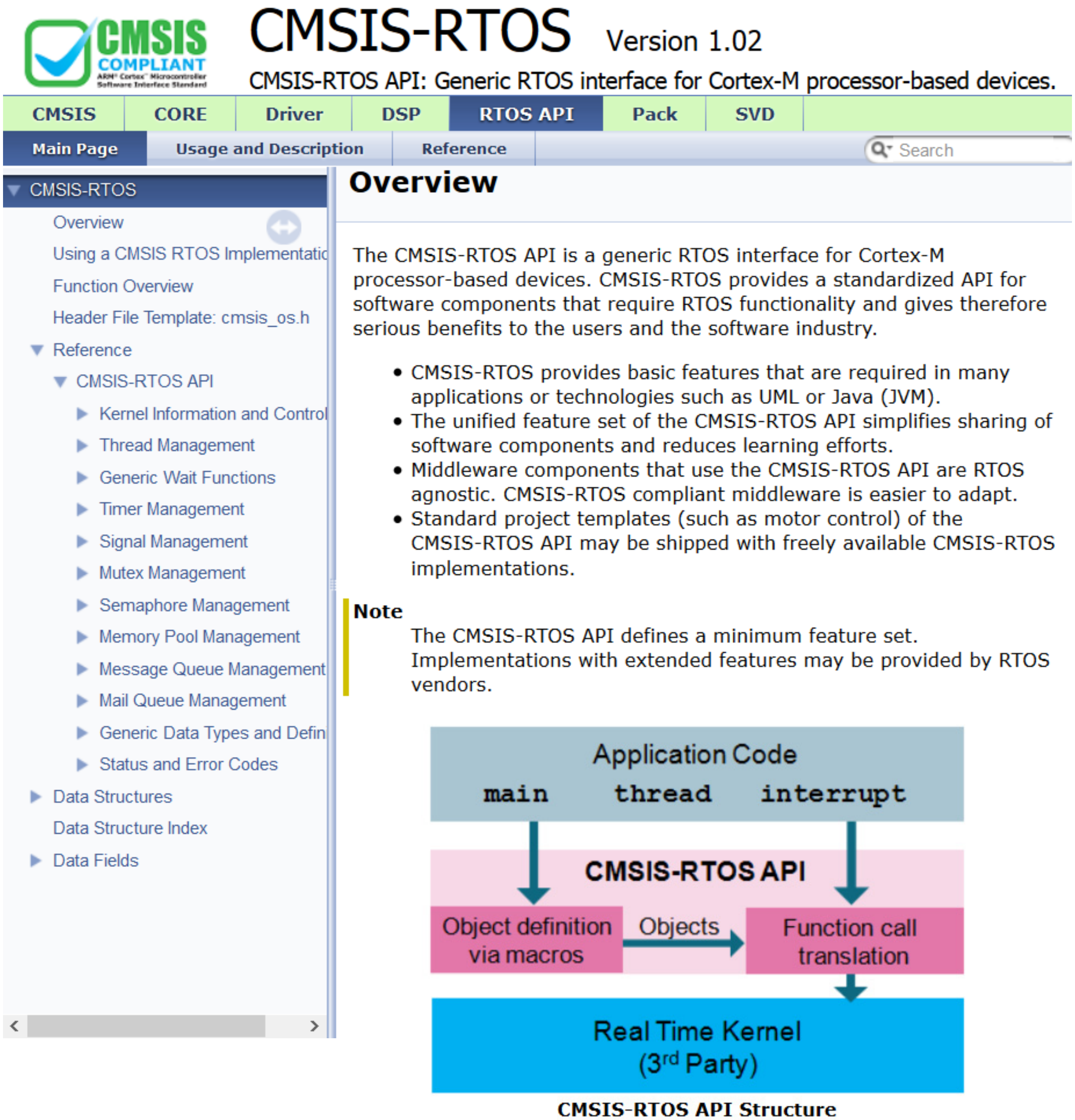
1. J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, Third Edition, Newnes, 2014.

- The STM32F4 projects take the specific form shown below:



-

CMSIS-RTOS[1]

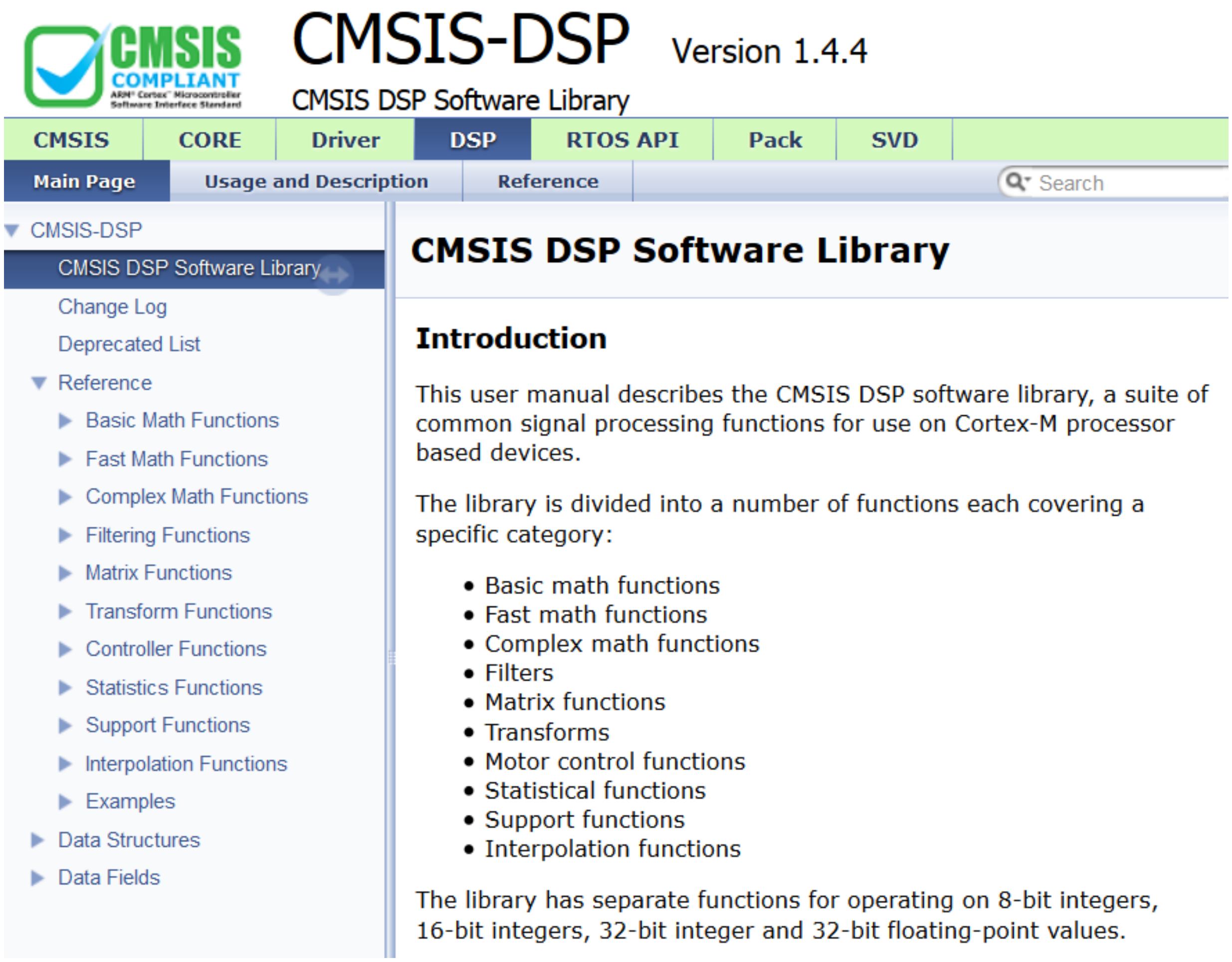


- This is an interesting topic for future study, perhaps in the final project

---

1. http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html

CMSIS-DSP[1]



CMSIS COMPLIANT

CMSIS-DSP Version 1.4.4

CMSIS DSP Software Library

CMSIS | CORE | Driver | DSP | RTOS API | Pack | SVD

Main Page | Usage and Description | Reference | Search

- CMSIS-DSP
  - CMSIS DSP Software Library
  - Change Log
  - Deprecated List
  - Reference
    - Basic Math Functions
    - Fast Math Functions
    - Complex Math Functions
    - Filtering Functions
    - Matrix Functions
    - Transform Functions
    - Controller Functions
    - Statistics Functions
    - Support Functions
    - Interpolation Functions
    - Examples
  - Data Structures
  - Data Fields

CMSIS DSP Software Library

Introduction

This user manual describes the CMSIS DSP software library, a suite of common signal processing functions for use on Cortex-M processor based devices.

The library is divided into a number of functions each covering a specific category:

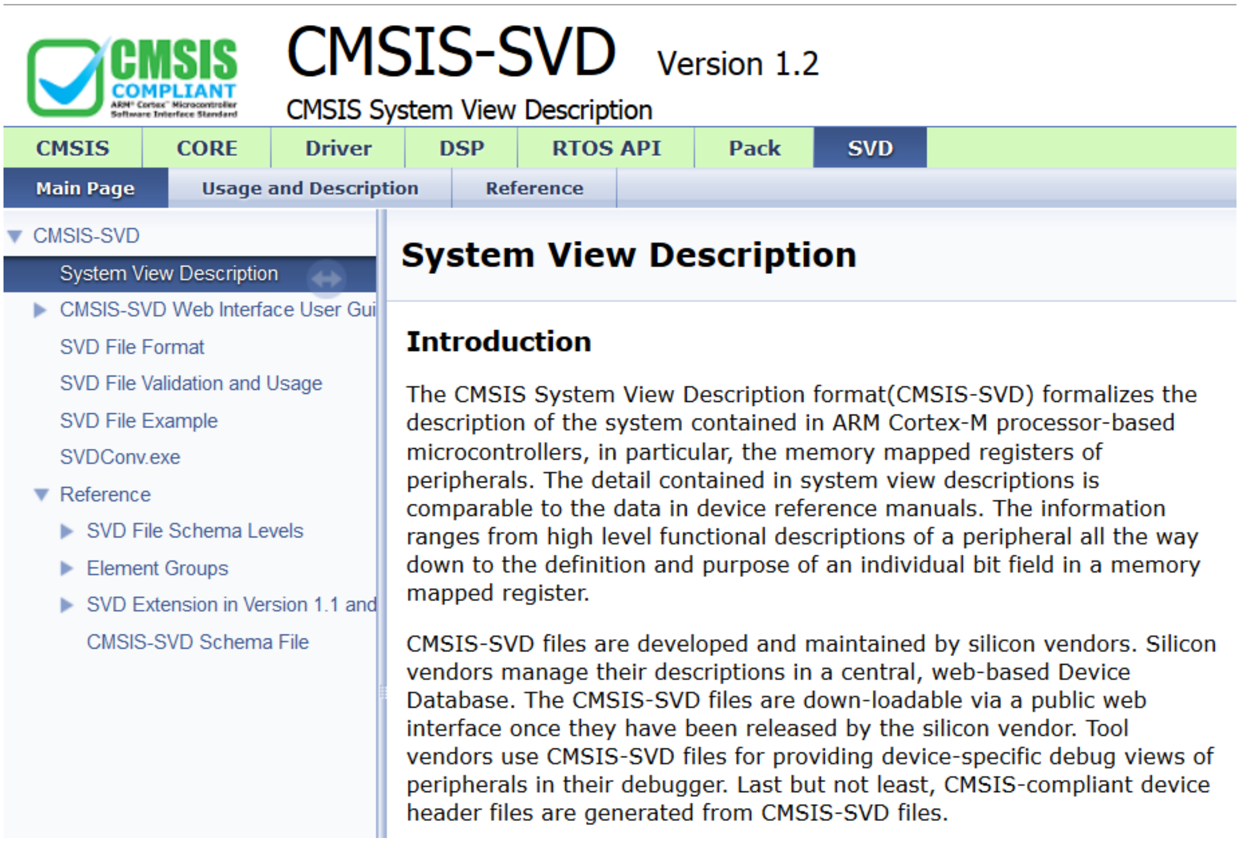
- Basic math functions
- Fast math functions
- Complex math functions
- Filters
- Matrix functions
- Transforms
- Motor control functions
- Statistical functions
- Support functions
- Interpolation functions

The library has separate functions for operating on 8-bit integers, 16-bit integers, 32-bit integer and 32-bit floating-point values.

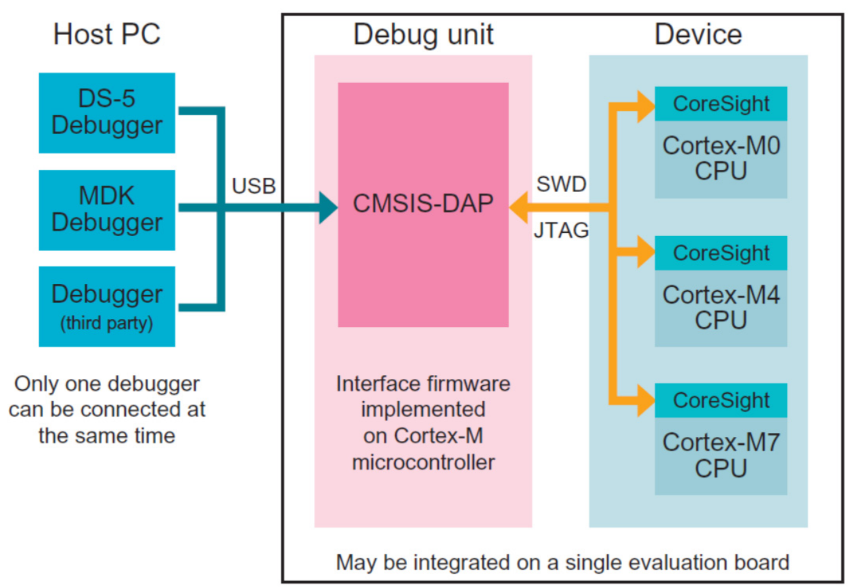
- Very powerful and convenient for implementing core DSP algorithms across Cortex-M processors
- In particular notes chapters that deal with digital filters (FIR and IIR) and the FFT we will explore this library
- Assignment #2 begins the exploration by considering matrix functions and FIR filtering using the `float32_t` data type

1. https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html

CMSIS-SVD[1] and CMSIS-DAP



- Fully realized with advanced debugging hardware



1. http://www.keil.com/pack/doc/CMSIS/SVD/html/index.html

Efficient C Coding of DSP on the M4[1]

- Efficient C-coding of DSP is possible using pure C coding augmented with
  - Intrinsic functions
  - Idiom recognition patterns (for some compilers i.e. MDK)

  and of course the use of CMSIS-DSP itself
- The use of intrinsic functions is the traditional way of getting instant access to assembly functions
- With idiom recognition patterns you get to write real C-code, but given a compatible compiler, the pattern is automatically converted to a special instruction or sequence of instructions
  - The advantage is that the code is more portable, as the C code will run under any compiler, while the intrinsic will not

DSP Programming Overview

- In Chapter 21 of Yiu there is a good discussion about using the Cortex-M4 for DSP
- The starting point is why consider DSP on a microcontroller
  - DSP applications are well known for requiring math intensive algorithms, so what advantage can the Cortex-M4 bring to the table?

1. Chapters 21 and 22 of J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, Third Edition, Newnes, 2014.

  - Dedicated digital signal processors such as those of Texas Instruments and Analog devices are very powerful, but microcontrollers are very good at peripheral interfacing
  - Certain types of connected devices need some of both and by traditional standards would require two processors, microcontroller and DSP
  - The Cortex-M4 (and M7), with its DSP extensions, can serve both purposes, which leads to lower power consumption, ease of integration, and lower cost overall
- To make all this happen efficiently, requires the use of:
  - Efficient C-code
  - Occasional use of intrinsics
  - Occasional use of idiom recognition patterns
  - Use of CMSIS-DSP

Intrinsics for DSP

- Assembly is the *go-to* for fast DSP code, but...
- Traditionally real-time DSP programming in C also makes use of intrinsic functions
- All intrinsic function begin with a double underscore (__)
- Appendix E.5 of the Yiu book lists most all of the intrinsics available for the M4
  - Table E.7 lists CMSIS-Core intrinsic functions for DSP related operations in the Cortex-M4 processor

CMSIS Functions Available for Cortex-M4	
uint32_t	**__SADD8** (uint32_t val1, uint32_t val2) GE setting quad 8-bit signed addition.
uint32_t	**__QADD8** (uint32_t val1, uint32_t val2) Q setting quad 8-bit saturating addition.
uint32_t	**__SHADD8** (uint32_t val1, uint32_t val2) Quad 8-bit signed addition with halved results.
uint32_t	**__UADD8** (uint32_t val1, uint32_t val2) GE setting quad 8-bit unsigned addition.
uint32_t	**__UQADD8** (uint32_t val1, uint32_t val2) Quad 8-bit unsigned saturating addition.
uint32_t	**__UHADD8** (uint32_t val1, uint32_t val2) Quad 8-bit unsigned addition with halved results.
uint32_t	**__SSUB8** (uint32_t val1, uint32_t val2) GE setting quad 8-bit signed subtraction.
uint32_t	**__QSUB8** (uint32_t val1, uint32_t val2) Q setting quad 8-bit saturating subtract.
uint32_t	**__SHSUB8** (uint32_t val1, uint32_t val2) Quad 8-bit signed subtraction with halved results.
uint32_t	**__USUB8** (uint32_t val1, uint32_t val2) GE setting quad 8-bit unsigned subtract.
uint32_t	**__UQSUB8** (uint32_t val1, uint32_t val2) Quad 8-bit unsigned saturating subtraction.
uint32_t	**__UHSUB8** (uint32_t val1, uint32_t val2) Quad 8-bit unsigned subtraction with halved results.
uint32_t	**__SADD16** (uint32_t val1, uint32_t val2) GE setting dual 16-bit signed addition.

More to follow...

Idiom Patterns for DSP

- See Yiu Chapter 21

Dot Product Example

- See Yiu Chapter 21

IIR Example: The Biquad Section

- See Yiu Chapter 21

Useful Resources

-

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0403c/index.html