Cortex Microcontroller Software Interface Standard (CMSIS)

Lecture 11

Microprocessor-based Systems (E155)

Prof. Josh Brake

Learning Objectives

By the end of this lecture you will be able to:

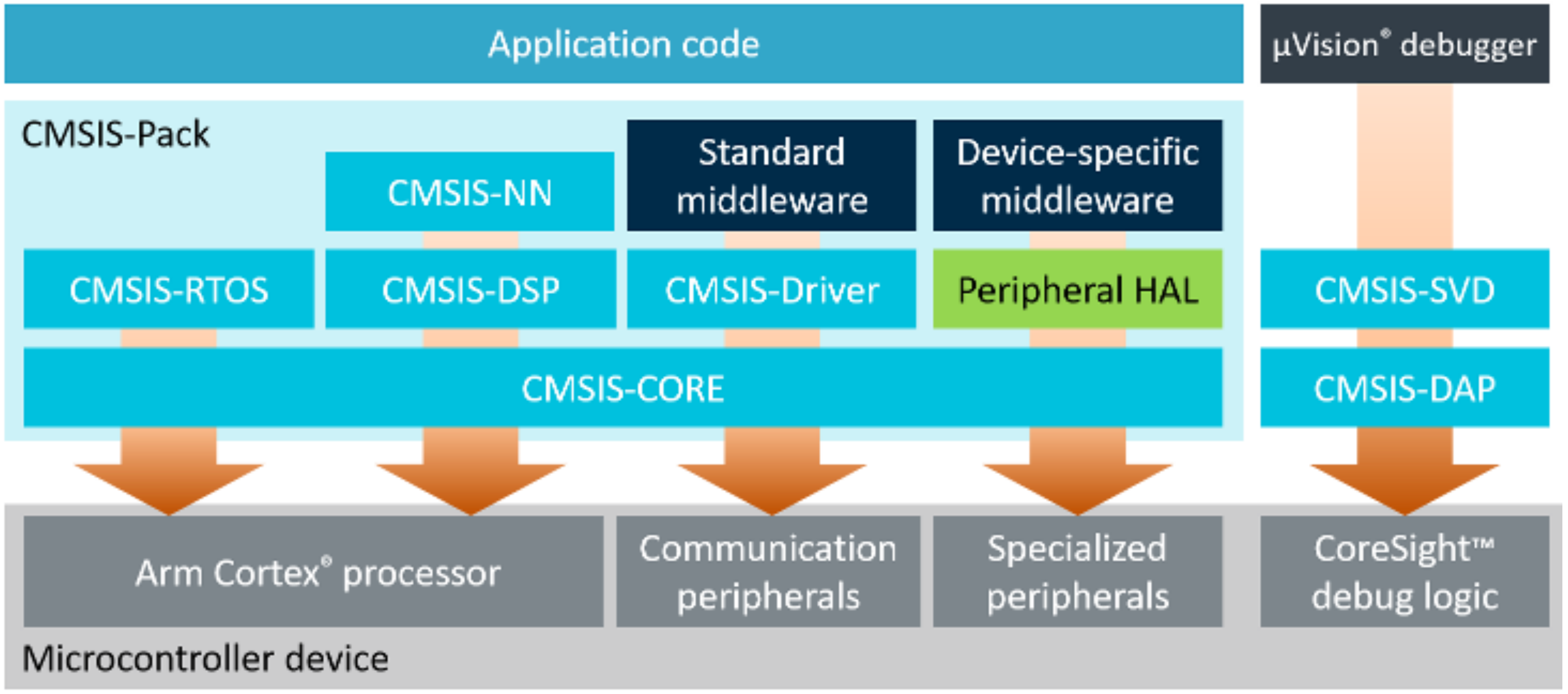
- Explain the core principles behind CMSIS
- Understand how to use the code provided by STM to build a project based on CMSIS

Outline

- CMSIS
    - What is it?
    - How to use it?
- Code Structures
    - Include guards
- PlatformIO Project Inspection
- Libraries – STM32Cube
    - Hardware Access Library (HAL)
    - Standard Peripheral Access Library

CMSIS Introduction

CMSIS Major Components



Goals of CMSIS

- Software reusability
- Software compatibility
- Easy to learn
- Toolchain independent
- Openness

CMSIS-Core Structure

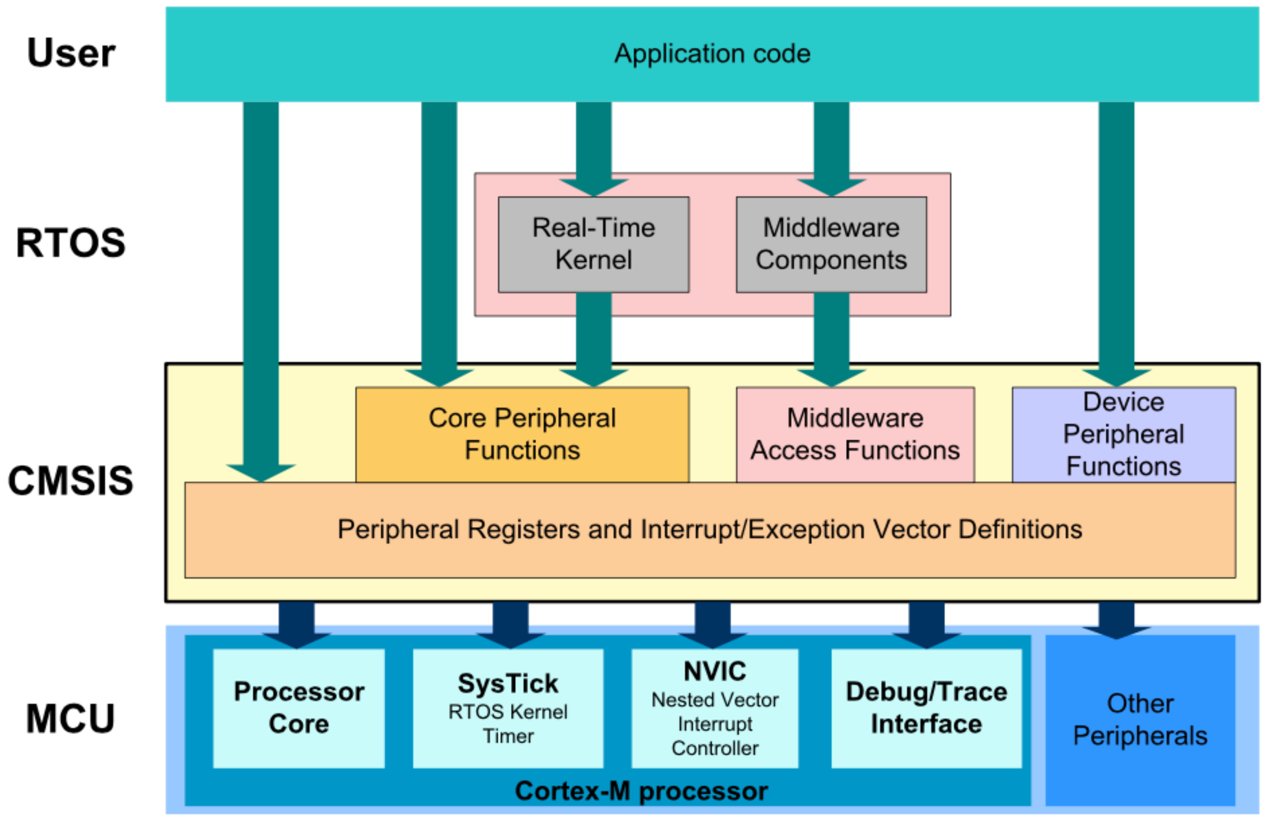
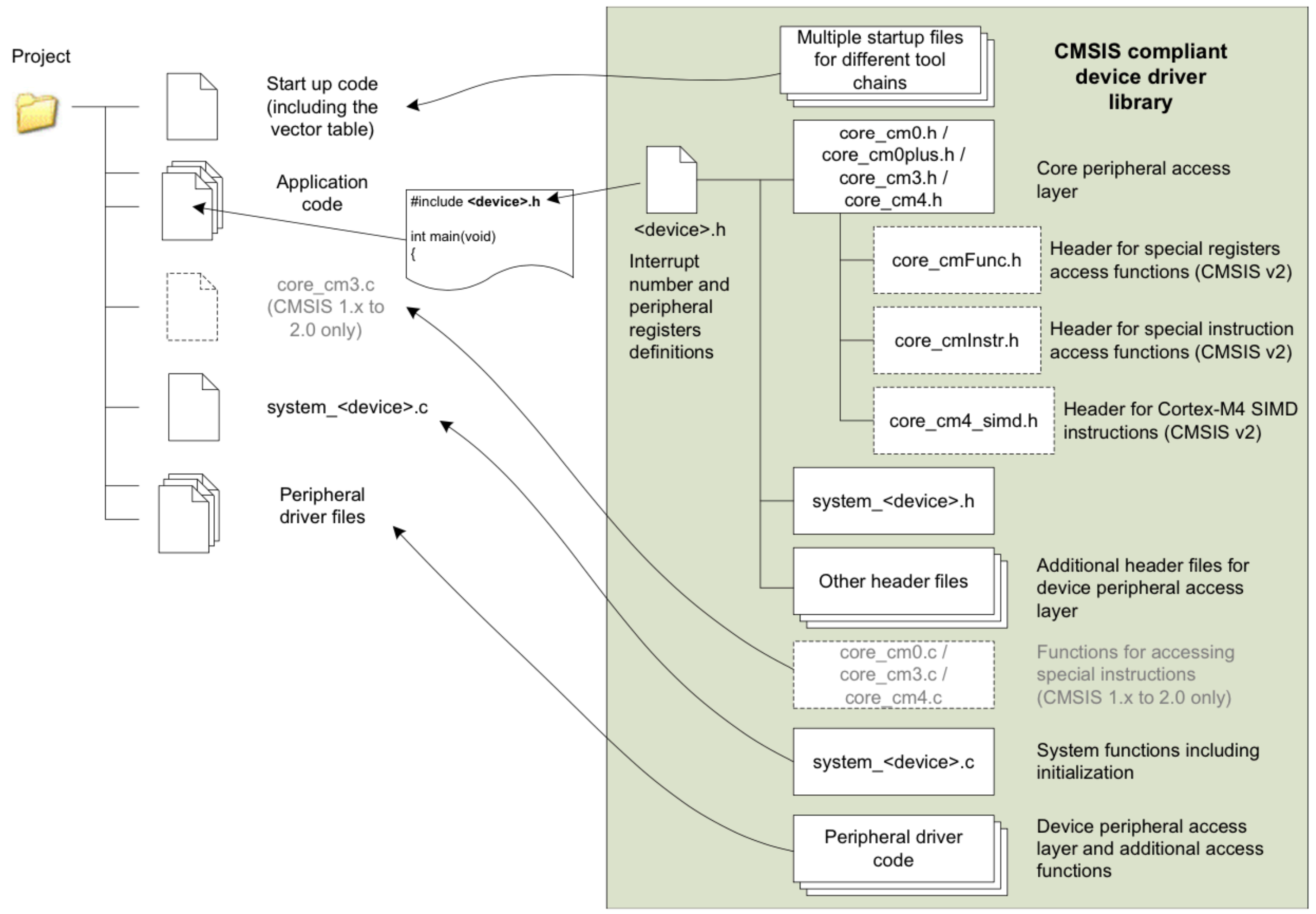

User
Application code
RTOS
Real-Time Kernel
Middleware Components
CMSIS
Core Peripheral Functions
Middleware Access Functions
Device Peripheral Functions
Peripheral Registers and Interrupt/Exception Vector Definitions
MCU
Processor Core
SysTick
RTOS Kernel Timer
NVIC
Nested Vector Interrupt Controller
Debug/Trace Interface
Other Peripherals
Cortex-M processor

FIGURE 2.13

CMSIS-Core structure

Using CMSIS-Core



Project
Start up code (including the vector table)
Application code
core_cm3.c (CMSIS 1.x to 2.0 only)
system_<device>.c
Peripheral driver files
#include <device>.h
int main(void)
{
<device>.h
Interrupt number and peripheral registers definitions
Multiple startup files for different tool chains
CMSIS compliant device driver library
core_cm0.h / core_cm0plus.h / core_cm3.h / core_cm4.h
Core peripheral access layer
core_cmFunc.h
Header for special registers access functions (CMSIS v2)
core_cmInstr.h
Header for special instruction access functions (CMSIS v2)
core_cm4_simd.h
Header for Cortex-M4 SIMD instructions (CMSIS v2)
system_<device>.h
Other header files
Additional header files for device peripheral access layer
core_cm0.c / core_cm3.c / core_cm4.c
Functions for accessing special instructions (CMSIS 1.x to 2.0 only)
system_<device>.c
System functions including initialization
Peripheral driver code
Device peripheral access layer and additional access functions

FIGURE 2.14

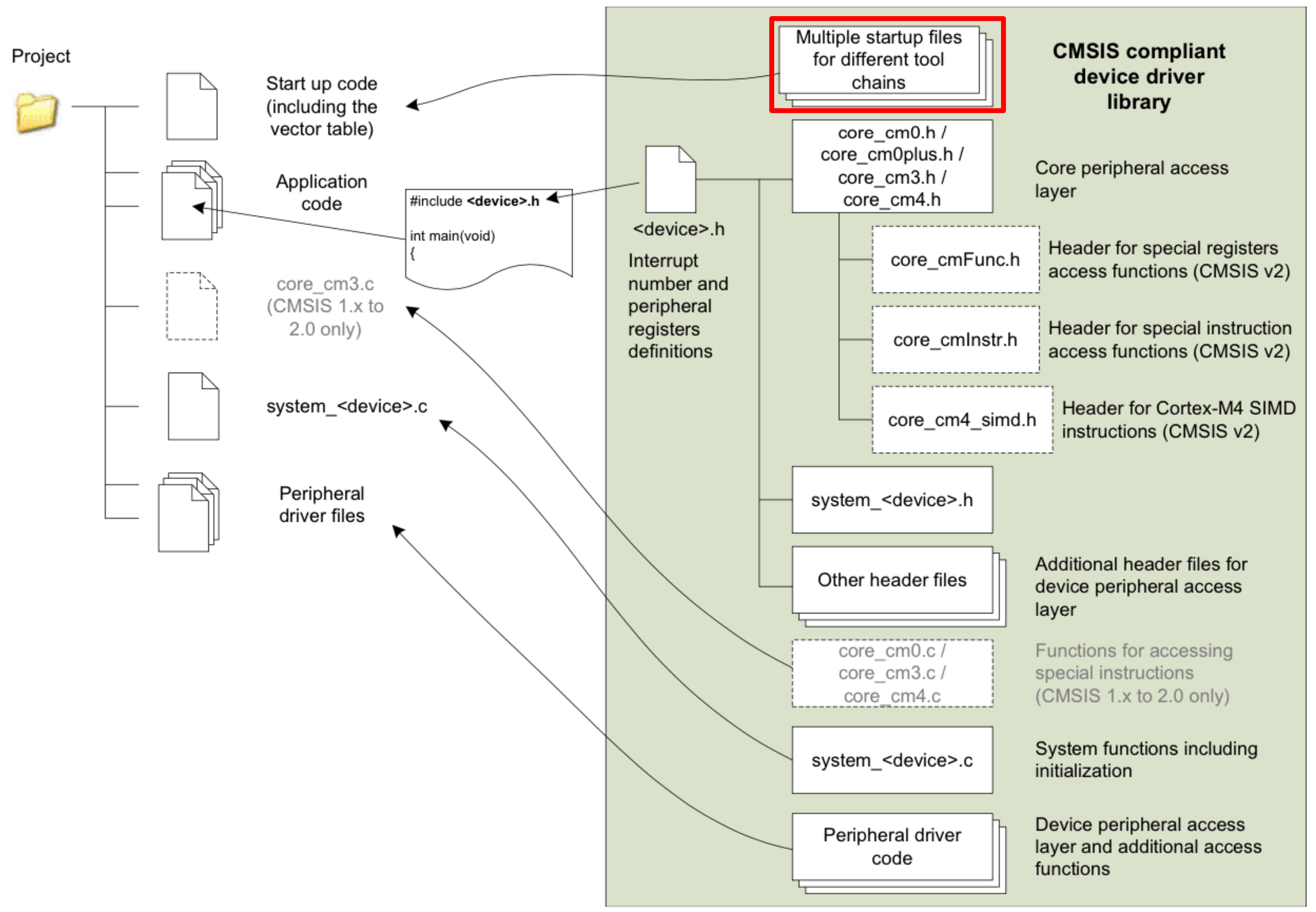
Using CMSIS-Core in a project

p. 52 *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

Files to Include in a project

- Startup Code (typically `startup_<device>.c/.s`)
- Application code (`main.c`)
- `<device>.h`
  - `core_cm4.h`
  - `system_<device>.h`
- `system_<device>.c`
- Peripheral Driver files (custom drivers you write or import)
  - For example, the drivers you are writing for lab.

Using CMSIS-Core: Startup Files



Project
Start up code (including the vector table)
Application code
#include <device>.h
int main(void)
{
core_cm3.c (CMSIS 1.x to 2.0 only)
system_<device>.c
Peripheral driver files
Multiple startup files for different tool chains
CMSIS compliant device driver library
core_cm0.h / core_cm0plus.h / core_cm3.h / core_cm4.h
Core peripheral access layer
<device>.h
Interrupt number and peripheral registers definitions
core_cmFunc.h
Header for special registers access functions (CMSIS v2)
core_cmInstr.h
Header for special instruction access functions (CMSIS v2)
core_cm4_simd.h
Header for Cortex-M4 SIMD instructions (CMSIS v2)
system_<device>.h
Other header files
Additional header files for device peripheral access layer
core_cm0.c / core_cm3.c / core_cm4.c
Functions for accessing special instructions (CMSIS 1.x to 2.0 only)
system_<device>.c
System functions including initialization
Peripheral driver code
Device peripheral access layer and additional access functions

FIGURE 2.14

Using CMSIS-Core in a project

p. 52 *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

`startup_<device>.c/.s`

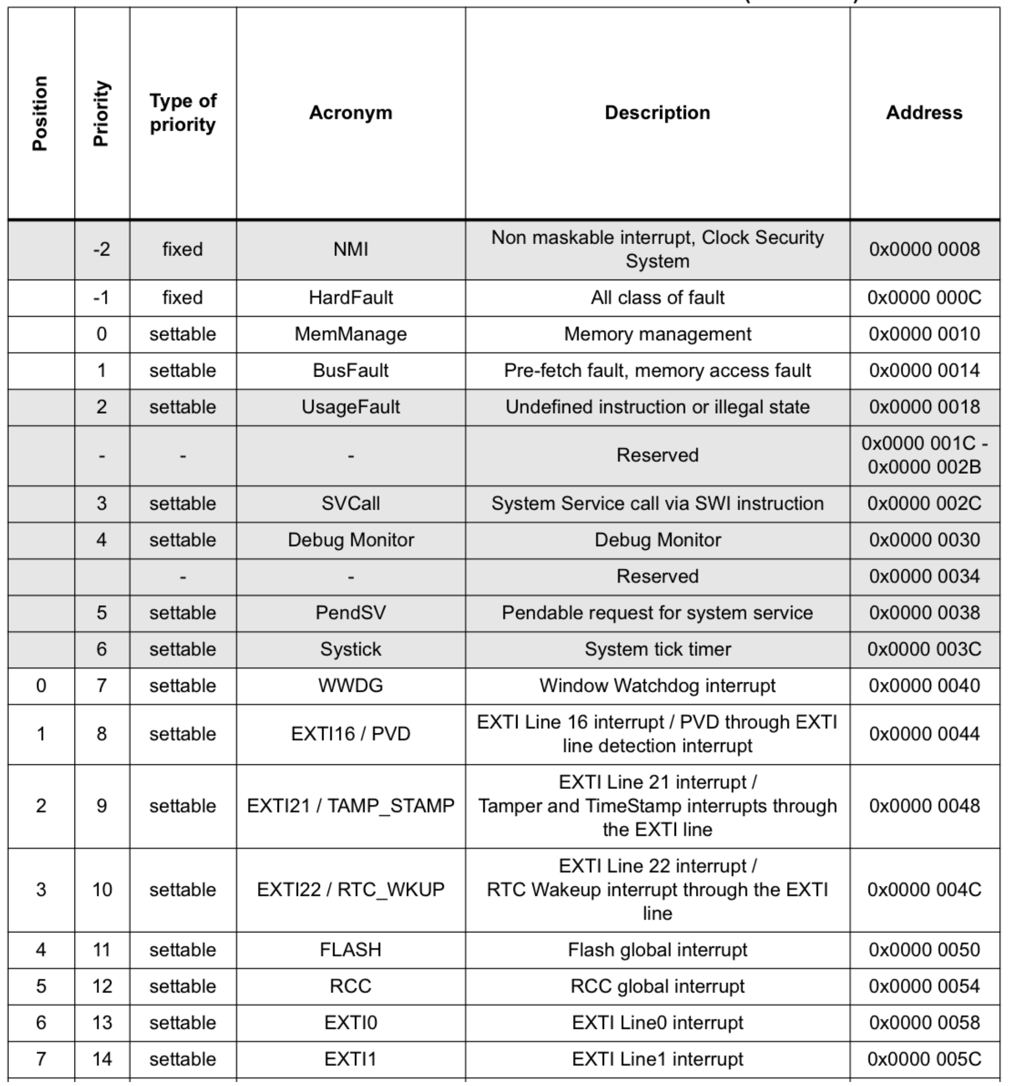
`startup_stm32f401xe.s`

- Set the initial SP
- Set the initial PC == Reset_Handler,
- Set the vector table entries with the exceptions ISR address
- Branches to main in the C library (which eventually calls main()).

startup_stm32f401xe.s

```
g_pfnVectors:
.word _estack
.word Reset_Handler
.word NMI_Handler
.word HardFault_Handler
.word MemManage_Handler
.word BusFault_Handler
.word UsageFault_Handler
.word 0
.word 0
.word 0
.word 0
.word SVC_Handler
.word DebugMon_Handler
.word 0
.word PendSV_Handler
.word SysTick_Handler
/* External Interrupts */
.word WWDG_IRQHandler /* Window WatchDog */
.word PVD_IRQHandler /* PVD through EXTI Line detection */
.word TAMP_STAMP_IRQHandler /* Tamper and TimeStamps through the EXTI line */
.word RTC_WKUP_IRQHandler /* RTC Wakeup through the EXTI line */
.word FLASH_IRQHandler /* FLASH */
.word RCC_IRQHandler /* RCC */
.word EXTI0_IRQHandler /* EXTI Line0 */
...
```

Table 38. Vector table for STM32F401xB/CSTM32F401xD/E (continued)



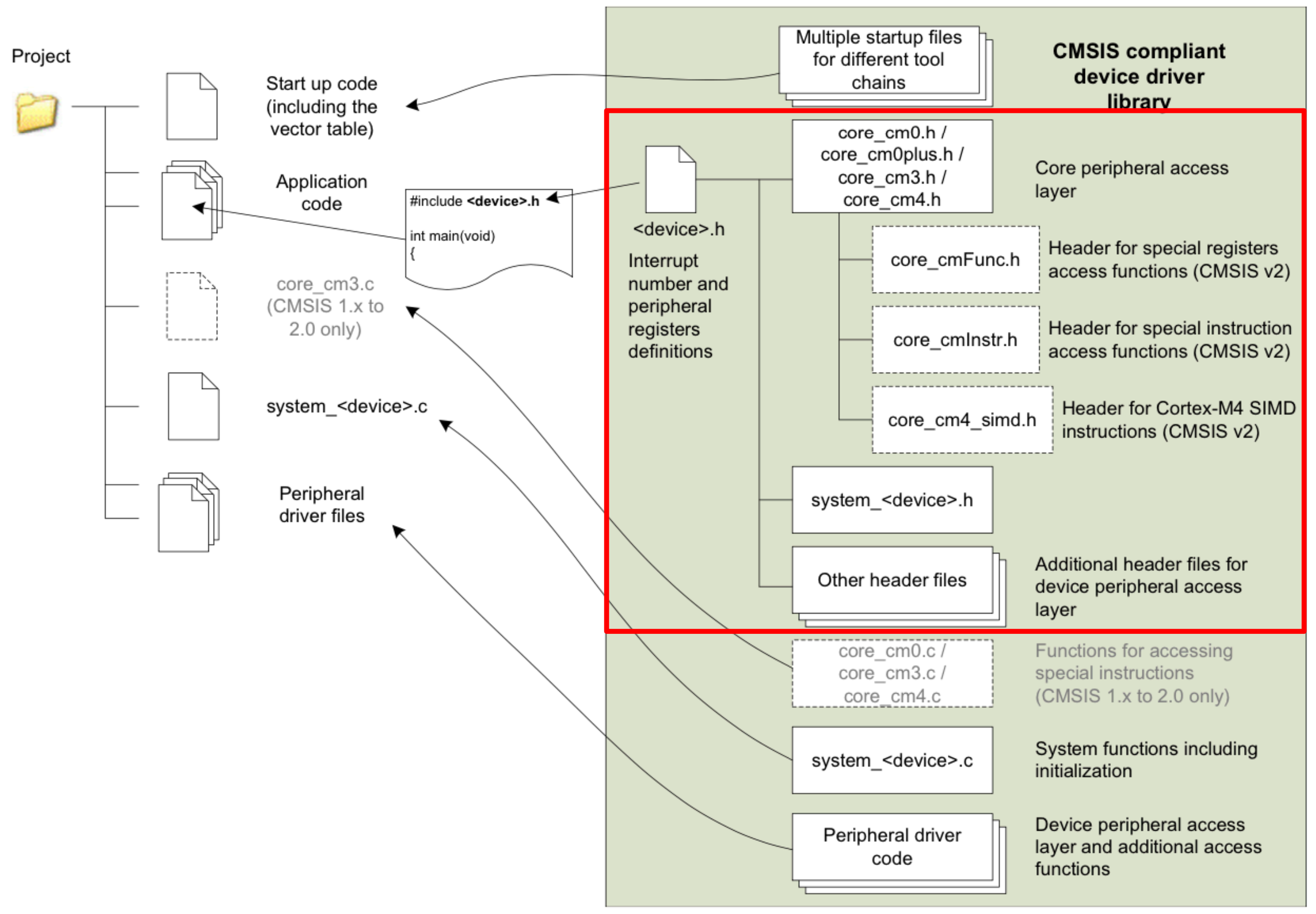
Position	Priority	Type of priority	Acronym	Description	Address
	-2	fixed	NMI	Non maskable interrupt, Clock Security System	0x0000 0008
	-1	fixed	HardFault	All class of fault	0x0000 000C
	0	settable	MemManage	Memory management	0x0000 0010
	1	settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
	2	settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
	-	-	-	Reserved	0x0000 001C - 0x0000 002B
	3	settable	SVCall	System Service call via SWI instruction	0x0000 002C
	4	settable	Debug Monitor	Debug Monitor	0x0000 0030
		-	-	Reserved	0x0000 0034
	5	settable	PendSV	Pendable request for system service	0x0000 0038
	6	settable	Systick	System tick timer	0x0000 003C
0	7	settable	WWDG	Window Watchdog interrupt	0x0000 0040
1	8	settable	EXTI16 / PVD	EXTI Line 16 interrupt / PVD through EXTI line detection interrupt	0x0000 0044
2	9	settable	EXTI21 / TAMP_STAMP	EXTI Line 21 interrupt / Tamper and TimeStamp interrupts through the EXTI line	0x0000 0048
3	10	settable	EXTI22 / RTC_WKUP	EXTI Line 22 interrupt / RTC Wakeup interrupt through the EXTI line	0x0000 004C
4	11	settable	FLASH	Flash global interrupt	0x0000 0050
5	12	settable	RCC	RCC global interrupt	0x0000 0054
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C

startup_stm32f401xe.s

```
/*******************************************************************************
*
* Provide weak aliases for each Exception handler to the Default_Handler.
* As they are weak aliases, any function with the same name will override
* this definition.
*
*******************************************************************************/
.weak NMI_Handler
.thumb_set NMI_Handler,Default_Handler
.weak HardFault_Handler
.thumb_set HardFault_Handler,Default_Handler
.weak MemManage_Handler
.thumb_set MemManage_Handler,Default_Handler
.weak BusFault_Handler
.thumb_set BusFault_Handler,Default_Handler

.weak UsageFault_Handler
.thumb_set UsageFault_Handler,Default_Handler
...
```

Using CMSIS-Core: Device Files



Project
Start up code (including the vector table)
Application code
#include <device>.h
int main(void)
{
core_cm3.c (CMSIS 1.x to 2.0 only)
system_<device>.c
Peripheral driver files
Multiple startup files for different tool chains
CMSIS compliant device driver library
core_cm0.h / core_cm0plus.h / core_cm3.h / core_cm4.h
Core peripheral access layer
<device>.h
Interrupt number and peripheral registers definitions
core_cmFunc.h
Header for special registers access functions (CMSIS v2)
core_cmInstr.h
Header for special instruction access functions (CMSIS v2)
core_cm4_simd.h
Header for Cortex-M4 SIMD instructions (CMSIS v2)
system_<device>.h
Other header files
Additional header files for device peripheral access layer
core_cm0.c / core_cm3.c / core_cm4.c
Functions for accessing special instructions (CMSIS 1.x to 2.0 only)
system_<device>.c
System functions including initialization
Peripheral driver code
Device peripheral access layer and additional access functions

FIGURE 2.14

Using CMSIS-Core in a project

p. 52 *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

`<device>.h`

- Data structures and the address mapping for all peripherals
- peripherals registers declarations and bits definition
- Macros to access peripheral’s registers hardware

stm32f401xe.h

Base Addresses

```
/** @addtogroup Peripheral_memory_map
* @{
*/
#define FLASH_BASE 0x08000000U /*!< FLASH(up to 1 MB) base address in the alias region */
#define SRAM1_BASE 0x20000000U /*!< SRAM1(96 KB) base address in the alias region */
#define PERIPH_BASE 0x40000000U /*!< Peripheral base address in the alias region */
#define SRAM1_BB_BASE 0x22000000U /*!< SRAM1(96 KB) base address in the bit-band region */
#define PERIPH_BB_BASE 0x42000000U /*!< Peripheral base address in the bit-band region */
#define BKPSRAM_BB_BASE 0x42480000U /*!< Backup SRAM(4 KB) base address in the bit-band region */
#define FLASH_END 0x0807FFFFU /*!< FLASH end address */
#define FLASH_OTP_BASE 0x1FFF7800U /*!< Base address of : (up to 528 Bytes) embedded FLASH OTP Area */
#define FLASH_OTP_END 0x1FFF7A0FU /*!< End address of : (up to 528 Bytes) embedded FLASH OTP Area */
```

SPI Register Mapping

```
typedef struct
{
__IO uint32_t CR1; /*!< SPI control register 1 (not used in I2S mode), Address offset: 0x00 */
__IO uint32_t CR2; /*!< SPI control register 2, Address offset: 0x04 */
__IO uint32_t SR; /*!< SPI status register, Address offset: 0x08 */
__IO uint32_t DR; /*!< SPI data register, Address offset: 0x0C */
__IO uint32_t CRCPR; /*!< SPI CRC polynomial register (not used in I2S mode), Address offset: 0x10 */
__IO uint32_t RXCRCR; /*!< SPI RX CRC register (not used in I2S mode), Address offset: 0x14 */
__IO uint32_t TXCRCR; /*!< SPI TX CRC register (not used in I2S mode), Address offset: 0x18 */
__IO uint32_t I2SCFGR; /*!< SPI_I2S configuration register, Address offset: 0x1C */
__IO uint32_t I2SPR; /*!< SPI_I2S prescaler register, Address offset: 0x20 */
} SPI_TypeDef;
```

stm32f401xe.h

Bit definitions

```
/****************** Bits definition for GPIO_MODER register *****************/
#define GPIO_MODER_MODE0_Pos (0U)
#define GPIO_MODER_MODE0_Msk (0x3U << GPIO_MODER_MODE0_Pos) /*!< 0x00000003 */
#define GPIO_MODER_MODE0 GPIO_MODER_MODE0_Msk
#define GPIO_MODER_MODE0_0 (0x1U << GPIO_MODER_MODE0_Pos) /*!< 0x00000001 */
#define GPIO_MODER_MODE0_1 (0x2U << GPIO_MODER_MODE0_Pos)


typedef struct {
__IO uint32_t MODER; /*!< GPIO port mode register, Address offset: 0x00 */
__IO uint32_t OTYPER; /*!< GPIO port output type register, Address offset: 0x04 */
__IO uint32_t OSPEEDR; /*!< GPIO port output speed register, Address offset: 0x08 */
__IO uint32_t PUPDR; /*!< GPIO port pull-up/pull-down register, Address offset: 0x0C */
__IO uint32_t IDR; /*!< GPIO port input data register, Address offset: 0x10 */
__IO uint32_t ODR; /*!< GPIO port output data register, Address offset: 0x14 */
__IO uint32_t BSRR; /*!< GPIO port bit set/reset register, Address offset: 0x18 */
__IO uint32_t LCKR; /*!< GPIO port configuration lock register, Address offset: 0x1C */
__IO uint32_t AFR[2]; /*!< GPIO alternate function registers, Address offset: 0x20-0x24 */
} GPIO_TypeDef;

#define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)
```

stm32f401xe.h

Using bit definitions

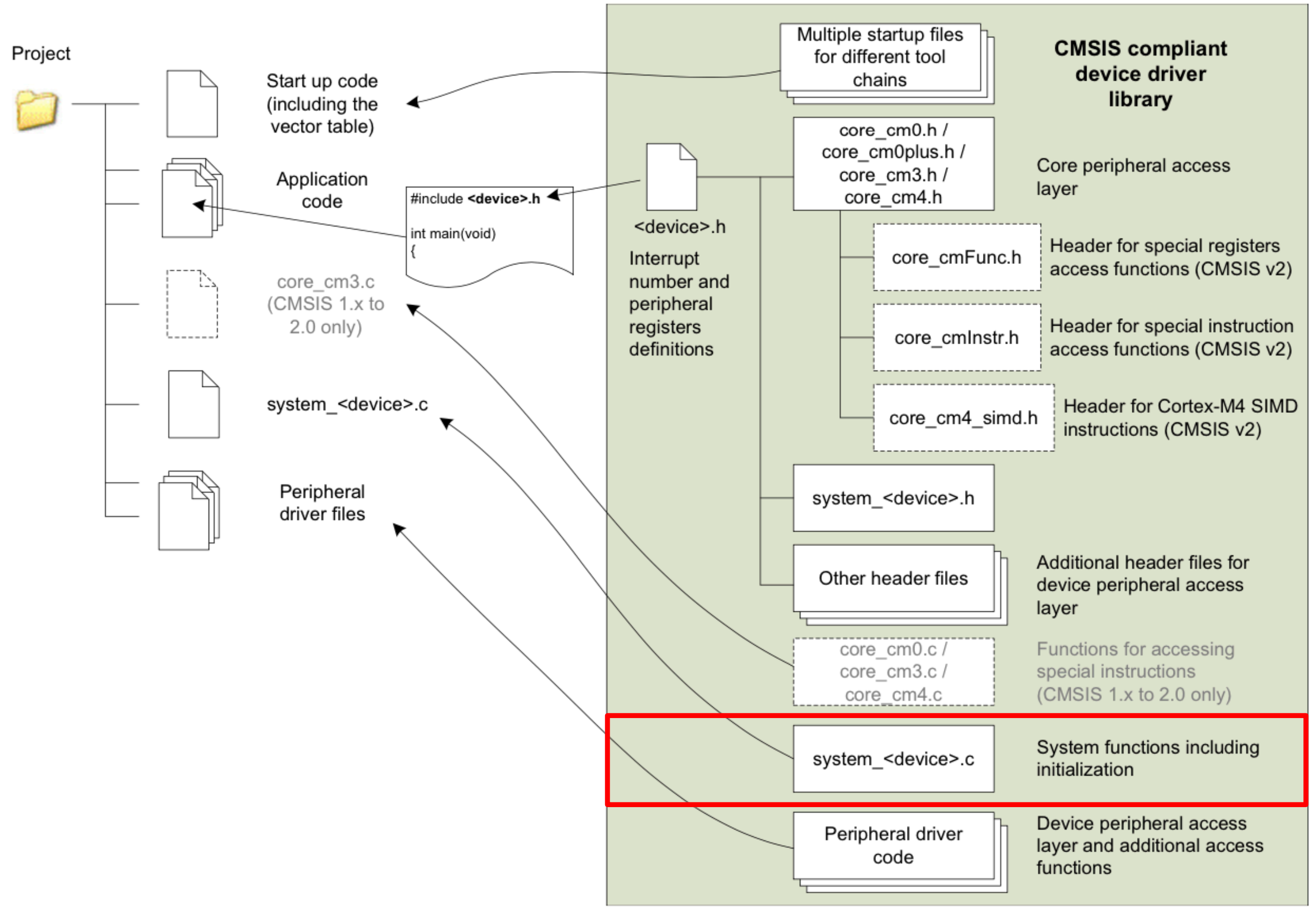
```
/****************** Bits definition for GPIO_MODER register *****************/
#define GPIO_MODER_MODE0_Pos (0U)
#define GPIO_MODER_MODE0_Msk (0x3U << GPIO_MODER_MODE0_Pos) /*!< 0x00000003 */
#define GPIO_MODER_MODE0 GPIO_MODER_MODE0_Msk
#define GPIO_MODER_MODE0_0 (0x1U << GPIO_MODER_MODE0_Pos) /*!< 0x00000001 */
#define GPIO_MODER_MODE0_1 (0x2U << GPIO_MODER_MODE0_Pos)

#define GPIOA ((GPIO_TypeDef *) GPIOA_BASE)
```

Example code to set PA0 to OUTPUT (0x1=0b01)

```
GPIOA->MODER &= ~(0b11 << GPIO_MODER_MODE0_Pos) // Clear bits
GPIOA->MODER |= (0b01 << GPIO_MODER_MODE0_Pos)  // Set bit 0
```

Using CMSIS-Core: System Source File



Project
Start up code (including the vector table)
Application code
#include <device>.h
int main(void)
{
core_cm3.c (CMSIS 1.x to 2.0 only)
system_<device>.c
Peripheral driver files
Multiple startup files for different tool chains
CMSIS compliant device driver library
core_cm0.h / core_cm0plus.h / core_cm3.h / core_cm4.h
Core peripheral access layer
<device>.h
Interrupt number and peripheral registers definitions
core_cmFunc.h
Header for special registers access functions (CMSIS v2)
core_cmInstr.h
Header for special instruction access functions (CMSIS v2)
core_cm4_simd.h
Header for Cortex-M4 SIMD instructions (CMSIS v2)
system_<device>.h
Other header files
Additional header files for device peripheral access layer
core_cm0.c / core_cm3.c / core_cm4.c
Functions for accessing special instructions (CMSIS 1.x to 2.0 only)
system_<device>.c
System functions including initialization
Peripheral driver code
Device peripheral access layer and additional access functions

FIGURE 2.14

Using CMSIS-Core in a project

p. 52 *The Definitive guide to ARM Cortex-M3 and Cortex-M4 Processors*

system_stm32f4xx.c

- SystemInit(): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup_stm32f4xx.s" file.
- SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.

Software Structures

Software Structures: Include Guards

- Prevents code from being included multiple times
- Remember that C preprocessor just copies and pastes headers into source code.
- We don't want to artificially inflate codebase.

Example

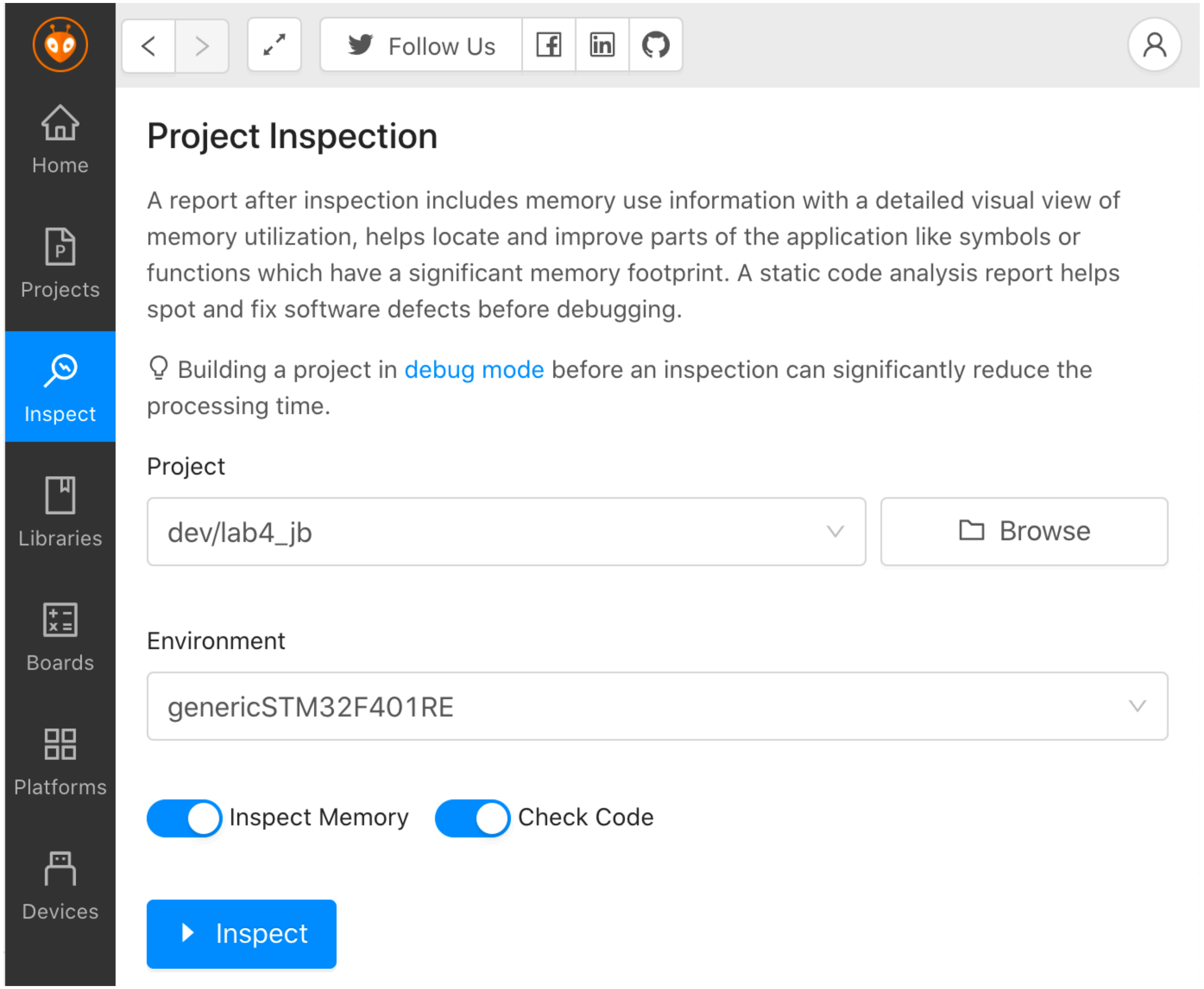
```
#ifndef __STM32F401xE_H
#define __STM32F401xE_H

//insert code here

#endif /* __STM32F401xE_H
*/
```

PlatformIO Project Inspection

PlatformIO Project Inspection



Project Inspection

A report after inspection includes memory use information with a detailed visual view of memory utilization, helps locate and improve parts of the application like symbols or functions which have a significant memory footprint. A static code analysis report helps spot and fix software defects before debugging.

Building a project in debug mode before an inspection can significantly reduce the processing time.

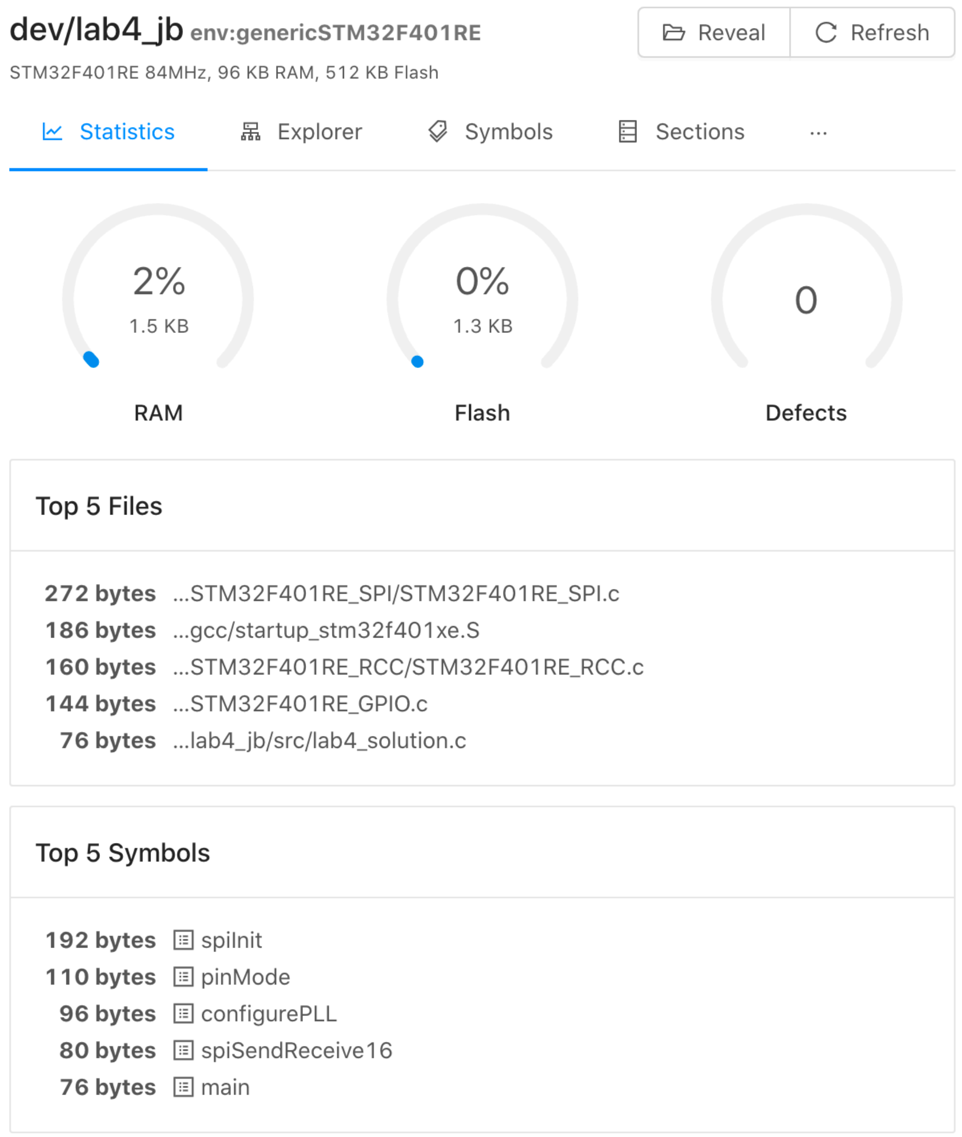
Project: dev/lab4_jb

Environment: genericSTM32F401RE

Inspect Memory: on

Check Code: on

PlatformIO Project Inspection



dev/lab4_jb env:genericSTM32F401RE

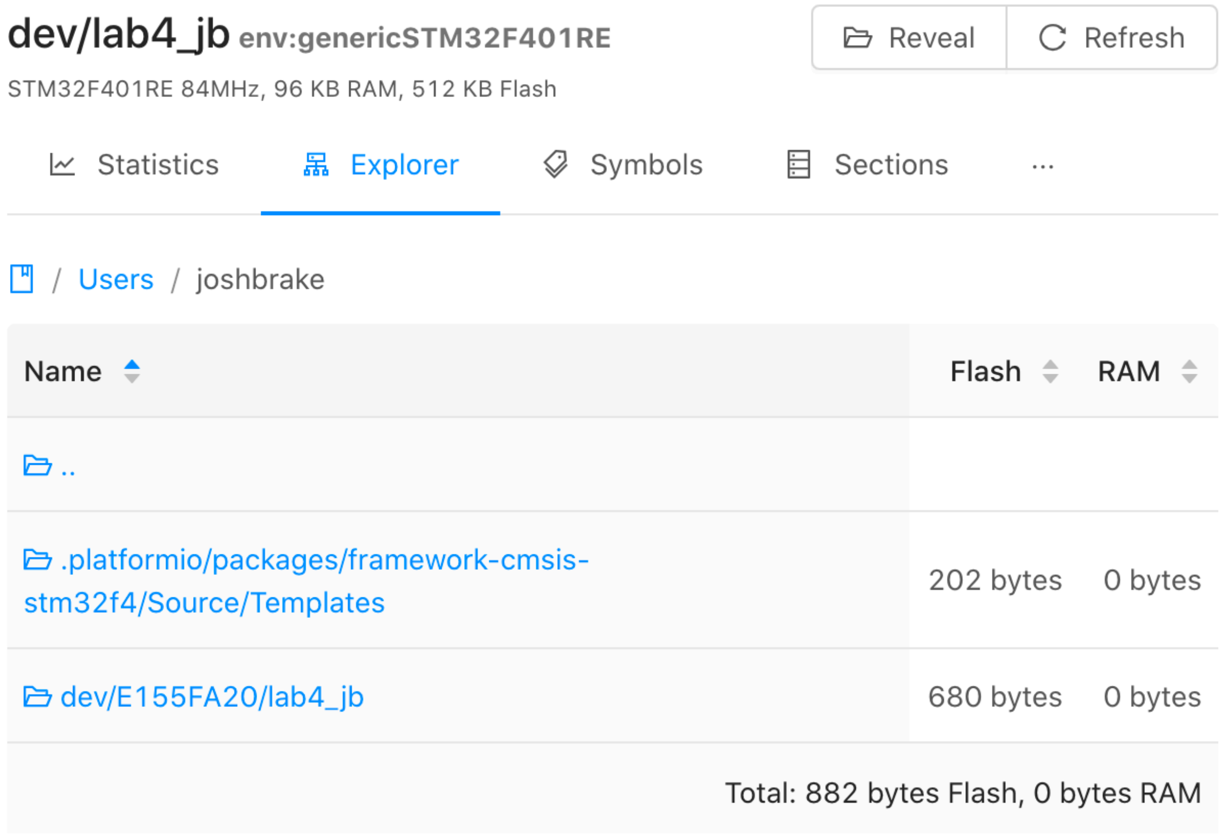
STM32F401RE 84MHz, 96 KB RAM, 512 KB Flash

Top 5 Files

272 bytes	...STM32F401RE_SPI/STM32F401RE_SPI.c
186 bytes	...gcc/startup_stm32f401xe.S
160 bytes	...STM32F401RE_RCC/STM32F401RE_RCC.c
144 bytes	...STM32F401RE_GPIO.c
76 bytes	...lab4_jb/src/lab4_solution.c

Top 5 Symbols

192 bytes	spiInit
110 bytes	pinMode
96 bytes	configurePLL
80 bytes	spiSendReceive16
76 bytes	main



dev/lab4_jb env:genericSTM32F401RE

STM32F401RE 84MHz, 96 KB RAM, 512 KB Flash

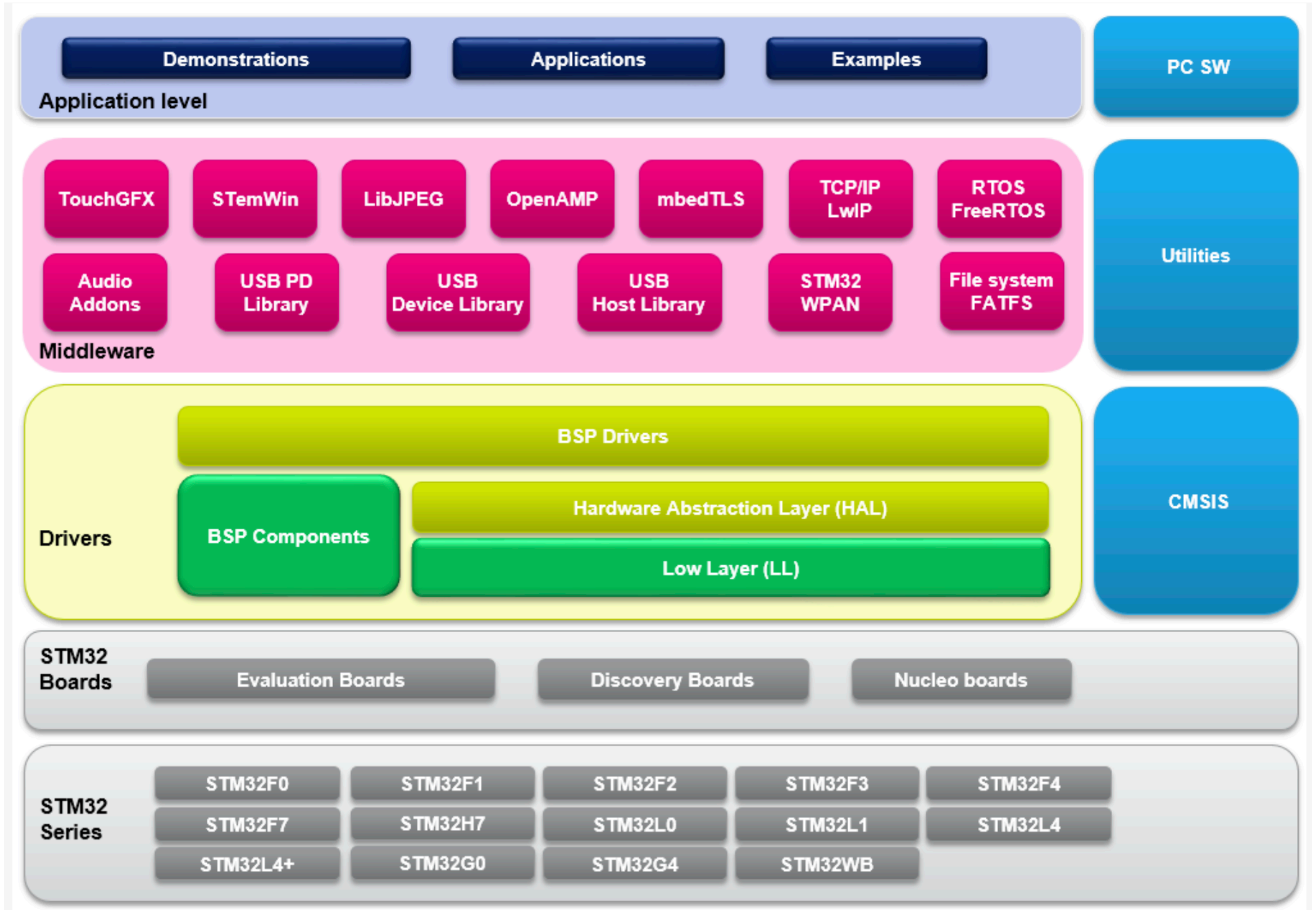
/ Users / joshbrake

Name	Flash	RAM
..		
.platformio/packages/framework-cmsis-stm32f4/Source/Templates	202 bytes	0 bytes
dev/E155FA20/lab4_jb	680 bytes	0 bytes

Total: 882 bytes Flash, 0 bytes RAM

STM32Cube Libraries

STM32Cube Libraries



STM32Cube Libraries

- Hardware Abstraction Layer (HAL) APIs/Drivers
  - Maximum portability
- Low-layer APIs (LL)
  - Maximum performance, closer to hardware and registers

https://github.com/STMicroelectronics/STM32CubeF4

stm32f4xx_hal_gpio.c

```c
/**
  * @brief  Toggles the specified GPIO pins.
  * @param  GPIOx Where x can be (A..K) to select the GPIO peripheral for STM32F429X device or
  *                      x can be (A..I) to select the GPIO peripheral for STM32F40XX and STM32F427X devices.
  * @param  GPIO_Pin Specifies the pins to be toggled.
  * @retval None
  */
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
{
  /* Check the parameters */
  assert_param(IS_GPIO_PIN(GPIO_Pin));

  if ((GPIOx->ODR & GPIO_Pin) == GPIO_Pin)
  {
    GPIOx->BSRR = (uint32_t)GPIO_Pin << GPIO_NUMBER;
  }
  else
  {
    GPIOx->BSRR = GPIO_Pin;
  }
}
```

Learning Objectives

By the end of this lecture you will be able to:

- Explain the core principles behind CMSIS
- Understand how to use the code provided by STM to build a project based on CMSIS

Up Next

- Wednesday: Interrupts

Lecture Feedback

- What is the most important thing you learned in class today?
- What point was most unclear from lecture today?

https://forms.gle/Ay6MkpZ6x3xsW2Eb8

µP

Feedback