Software Libraries: CMSIS and mbed SDK

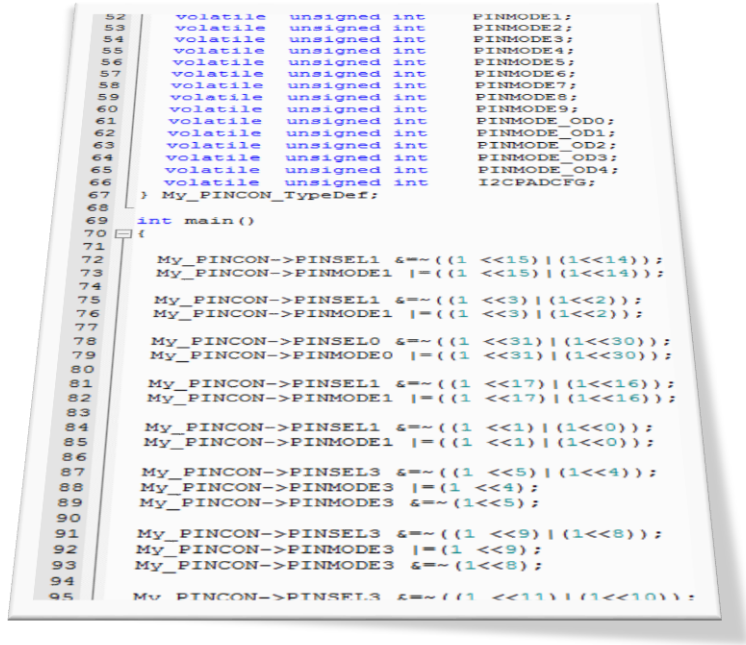
Module Syllabus

- Overview of Software Libraries
- Cortex Microcontroller Software Interface Standard (CMSIS)
  - What is CMSIS
  - What is Standardized in CMSIS
  - Benefits of CMSIS
  - CMSIS Functions
- mbed Software Development Kit (SDK)
  - What is mbed SDK
  - Features of mbed SDK
  - mbed SDK Library Structure
  - Example of mbed SDK Library Layers

Overview of Software Libraries

- In the previous modules, we have learned how to control a GPIO peripheral (connected to digital IOs such as LEDs) at a low-level. However, there are a number of disadvantages when programing in low-level, such as:

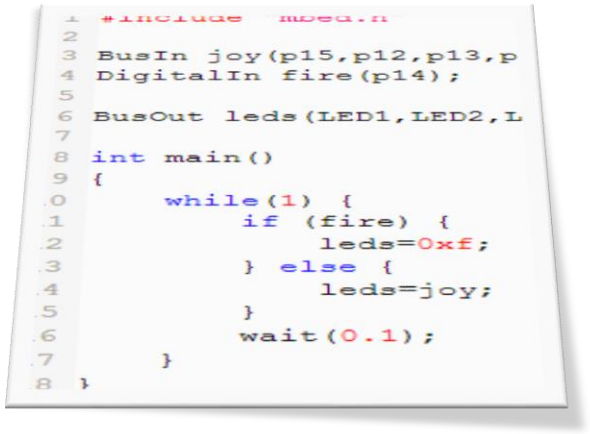
  - Low productivity
  - Less portable from one device to another device
  - Resulting code is more difficult for others to read, reuse, and maintain
  - Can result in inefficient code as high level optimization are missed (low code density)



ARM

Overview of Software Libraries

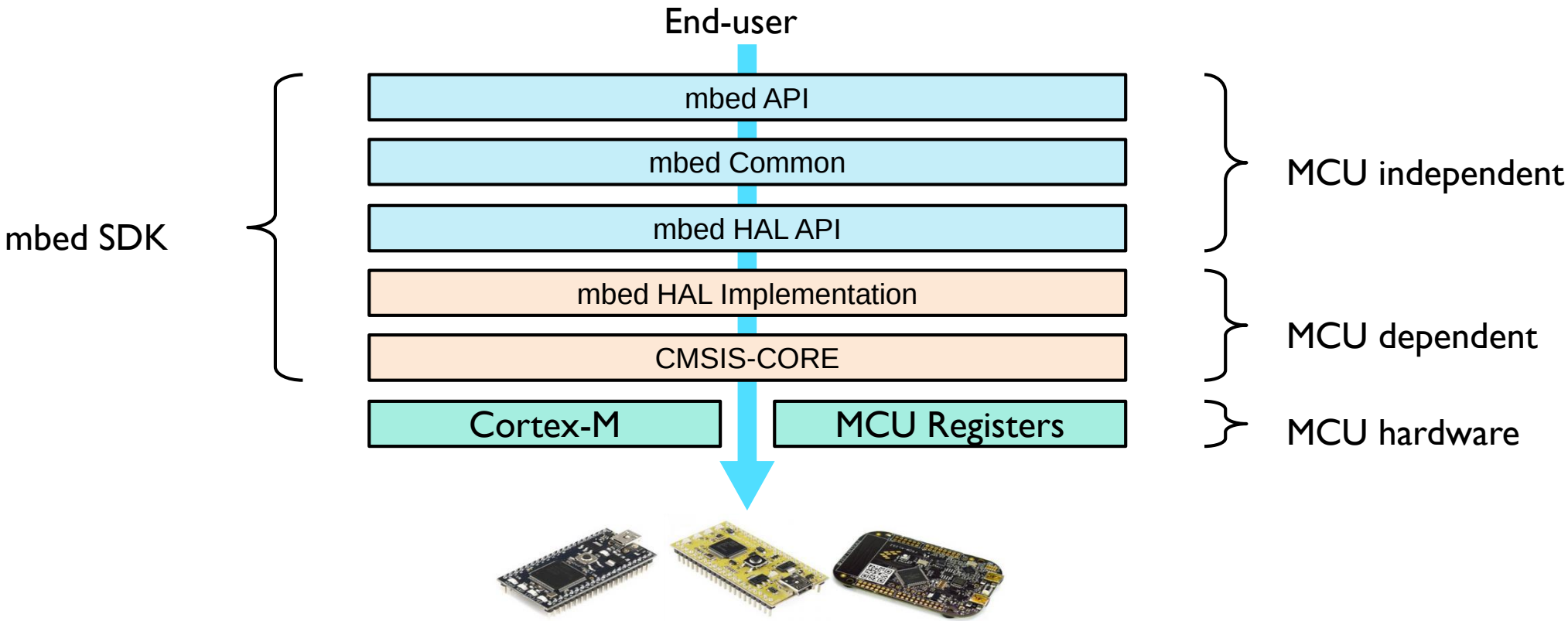
- With support from software libraries or Application Programming Interfaces (APIs), we could ease our application development in a variety of ways to achieve :
  - Higher productivity (less development time)
  - Portability across devices
  - Resulting code is easier code to read, reuse and maintain by others
  - More efficient code (code density and performance)



ARM

ARM CMSIS and mbed SDK

- In this module, we will introduce two libraries/APIs:
  - CMSIS-CORE – Cortex Microcontroller Software Interface Standard
  - mbed SDK – mbed Software Development Kit



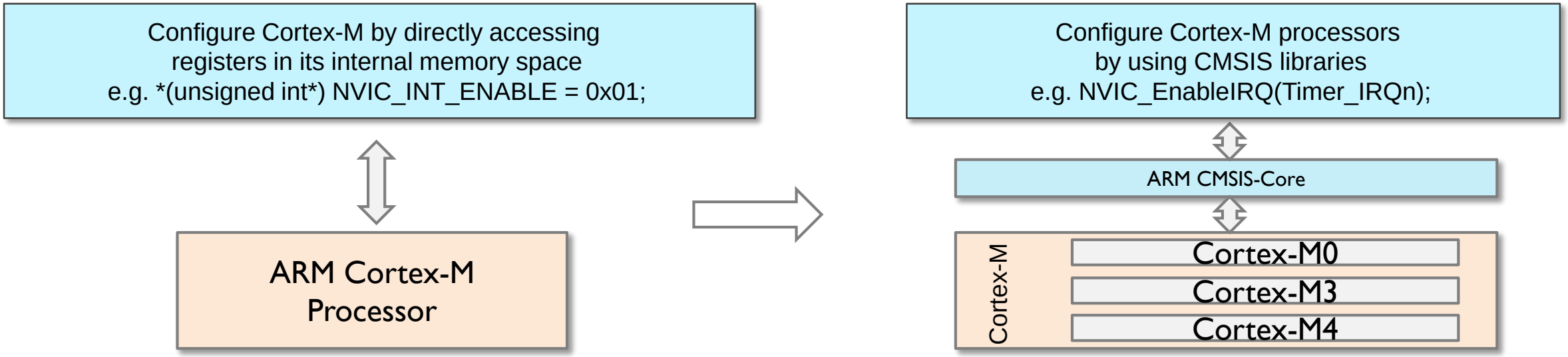
ARM

CMSIS – Cortex Microcontroller Software Interface Standard

ARM

What is CMSIS

- CMSIS – Cortex Microcontroller Software Interface Standard
  - CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series
  - CMSIS provides a standardized software interface, such as library functions which help you control the processor more easily, e.g. configuring the Nested Vectored Interrupt Controller (NVIC)
  - Main reason is to improve software portability across different Cortex-M processors and Cortex-M based microcontrollers



ARM

What is Standardized in CMSIS

- Standardized functions to access NVIC, System Control Block (SCB), and System Tick timer (SysTick), for example:
  - Enables an interrupt or exception: NVIC_EnableIRQ (IRQn_Type IRQn)
  - Sets pending status of interrupt: void NVIC_SetPendingIRQ (IRQn_Type IRQn)
- Standardized access of special registers, for example:
  - Read PRIMASK register: uint32_t __get_PRIMASK (void)
  - Set CONTROL register: void __set_CONTROL (uint32_t value)
- Standardized functions to access special instructions, e.g.
  - REV: uint32_t __REV(uint32_t int value)
  - NOP: void __NOP(void)
- Standardized names of system initialization functions, e.g.
  - System initialization: void SystemInit(void)

ARM

Benefits of CMSIS

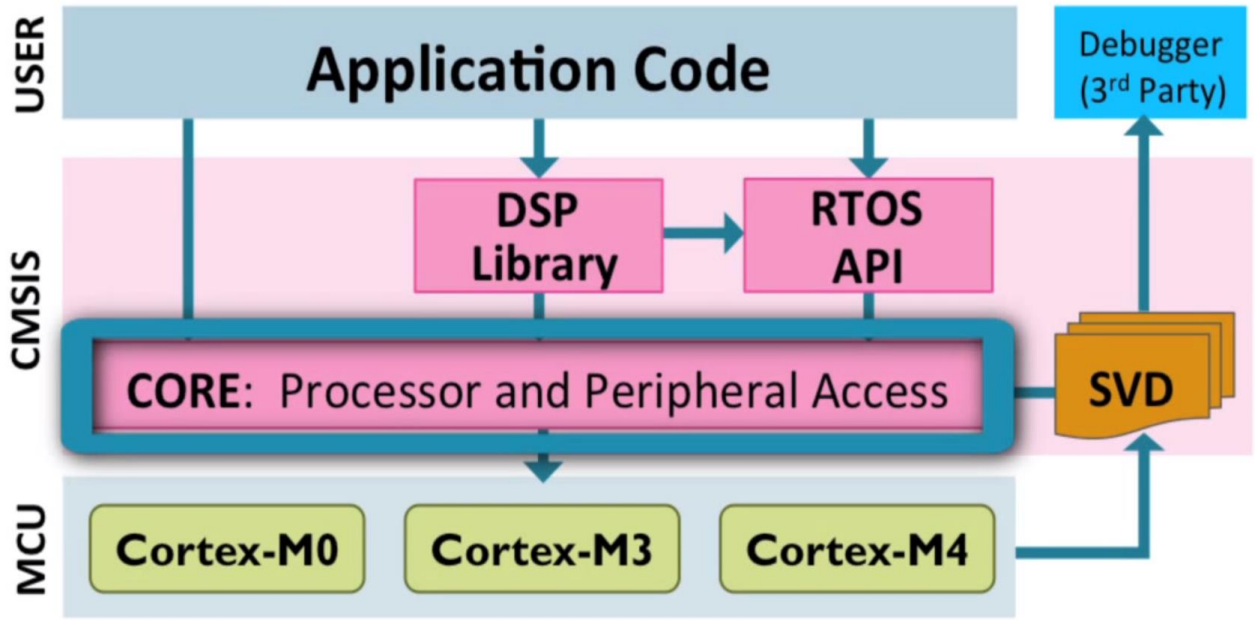
- Easier to port application code from one Cortex-M based microcontroller to another Cortex-M based microcontroller
- Less effort to reuse the same code between different Cortex-M based microcontrollers

- Better compatibility when integrating third-party software components, since all the third-party components such as applications, embedded OS, middleware etc., can share the same standard CMSIS interface
- Better code density and smaller memory footprint, since the codes in CMSIS have been optimised and tested

ARM

CMSIS Components

- The CMSIS consists of the following components:
  - CMSIS-CORE
  - CMSIS-DSP, CMSIS-RTOS API and CMSIS-SVD
  - In this module, we will focus on using CMSIS-CORE



ARM

CMSIS Components

- CMSIS-CORE
  - Provides an interface to Cortex-M0, Cortex-M3, Cortex-M4, SecureCore™ SC000 and SC300 processors, and peripheral registers
- CMSIS-DSP
  - DSP library with over 60 functions in fixed-point (fractional q7, q15, q31) and single precision floating-point (32-bit) implementation
- CMSIS-RTOS API
  - Standardized programming interface for real-time operating systems, for thread control, resource, and time management
- CMSIS-SVD:
  - System View Description XML files that contain the programmer's view of a complete microcontroller system including peripherals

ARM

CMSIS Functions: Access NVIC

CMSIS function	Description
void NVIC_EnableIRQ (IRQn_Type IRQn)	Enables an interrupt or exception.
void NVIC_DisableIRQ (IRQn_Type IRQn)	Disables an interrupt or exception.
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	Sets the pending status of interrupt or exception to 1.
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	Clears the pending status of interrupt or exception to 0.
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	Reads the pending status of interrupt or exception. This function returns non-zero value if the pending status is set to 1.
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	Sets the priority of an interrupt or exception with configurable priority level to 1.
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	Reads the priority of an interrupt or exception with configurable priority level. This function return the current priority level.

CMSIS Functions: Access Special Registers

Special register	Access	CMSIS function
PRIMASK	Read	*uint32_t __get_PRIMASK (void)*
	Write	*void __set_PRIMASK (uint32_t value)*
CONTROL	Read	*uint32_t __get_CONTROL (void)*
	Write	*void __set_CONTROL (uint32_t value)*
MSP	Read	*uint32_t __get_MSP (void)*
	Write	*void __set_MSP (uint32_t TopOfMainStack)*
PSP	Read	*uint32_t __get_PSP (void)*
	Write	*void __set_PSP (uint32_t TopOfProcStack)*

CMSIS Functions: Execute Special Instructions

Instruction	CMSIS intrinsic function
CPSIE i	*void __enable_irq(void)*
CPSID i	*void __disable_irq(void)*
ISB	*void __ISB(void)*
DSB	*void __DSB(void)*
DMB	*void __DMB(void)*
NOP	*void __NOP(void)*
REV	*uint32_t __REV(uint32_t int value)*
REV16	*uint32_t __REV16(uint32_t int value)*
REVSH	*uint32_t __REVSH(uint32_t int value)*
SEV	*void __SEV(void)*
WFE	*void __WFE(void)*
WFI	*void __WFI(void)*

CMSIS Functions: Access System

CMSIS function	Description
void NVIC_SystemReset(void)	Initiate a system reset request
uint32_t SysTick_Config(uint32_t ticks)	Initialize and start the SysTick counter and its interrupt
void SystemInit (void)	Initialize the system
void SystemCoreClockUpdate(void)	Update the SystemCoreClock variable

MBED SOFTWARE DEVELOPMENT KIT (SDK)

ARM

What is mbed SDK

- mbed platform includes:
  - **A standards-based C/C++ Software Development Kit (SDK)**
  - A microcontroller Hardware Development Kit (HDK) and supported development boards
  - Integrated Development Environment (IDE), including an online compiler and online developer collaboration tools

- mbed Software Development Kit (SDK) includes:
  - Software libraries
    - Official C/C++ software libraries
      - Start-up code, peripheral drivers, networking, RTOS and runtime environment
    - Customer-developed libraries and codes
      - Cookbook of hundreds of reusable peripheral and module libraries have been built on top of the SDK, which can be used to build your projects faster
  - Software tools, such as build tools, test and debug scripts

ARM

Features of mbed SDK

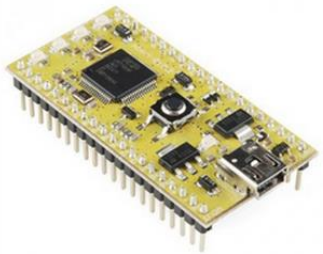
- Legal – are you allowed to use mbed SDK?
  - mbed SDK is licensed under the permissive Apache 2.0 licence
  - All codes can be used in both commercial and personal projects with confidence
- Supported Toolchains and IDEs
  - GCC ARM: GNU Tools for ARM Embedded Processors
  - ARMCC (standard library and MicroLib): uVision
  - IAR: IAR Embedded Workbench
  - GCC code_red: Red Suite
  - GCC CodeSourcery: Sourcery CodeBench

ARM

Features of mbed SDK

- Compatible with different hardware platforms
  - NXP series MCUs, including:
    - Cortex-M0: LPC11U24, LPC1114, LPC11C24, LPC4330
    - Cortex-M0+: LPC810, LPC812
    - Cortex-M3: LPC1768, LPC1347
    - Cortex-M4: LPC4088, LPC4330
    - ARM7TDMI-S: LPC2368
  - Freescale series MCUs including:
    - Cortex-M0+: KL25Z , KL05Z
  - STMicroelectronics series MCUs including:
    - Cortex-M4: STM32F407

mbed LPC1768



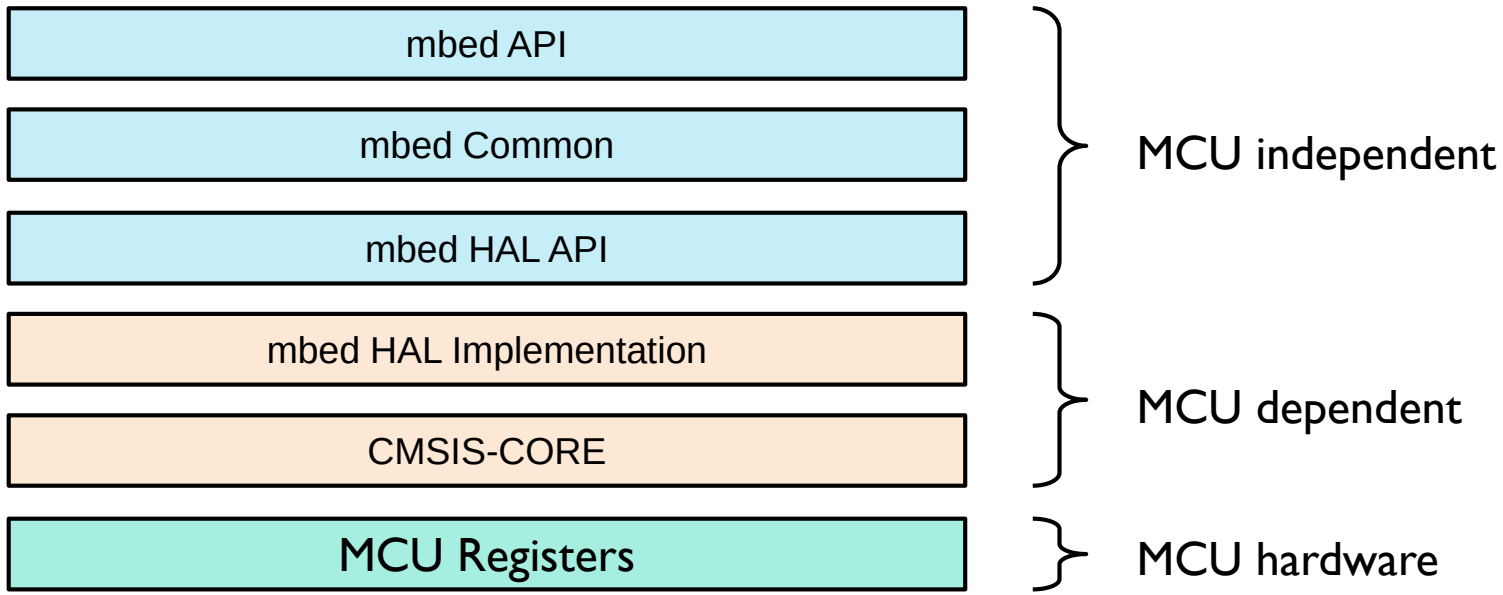
mbed LPC11U24

Freescale KL25Z

ARM

mbed SDK Library Structure

- The mbed SDK library provides abstractions for:
  - MCU independent layer – mbed API, mbed common, and mbed HAL API
  - MCU dependent layer – mbed HAL Implementation and CMSIS-CORE
  - MCU hardware – MCU Registers



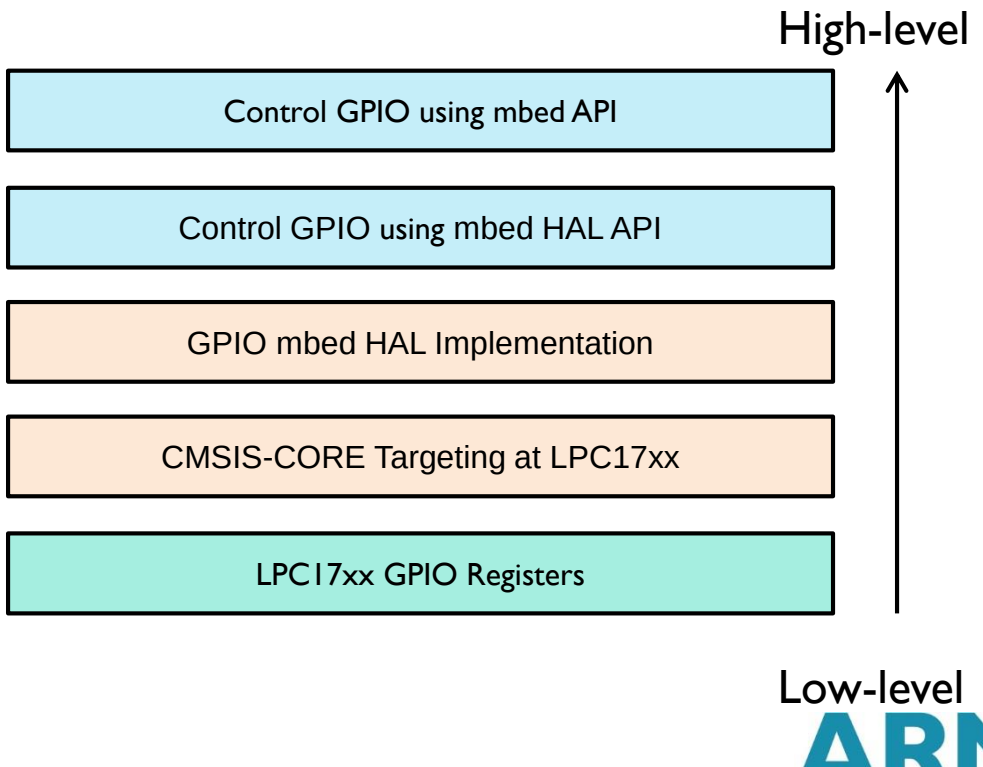
ARM

Example of mbed SDK Library Layers

- To understand how the mbed SDK library is structured and layered, we will use the GPIO example that we have been familiar with in the previous modules

- We will explain it using a LED blinky example, which can be programmed at different levels, using libraries from the lowest layer to the highest layer:

  - MCU register layer
    - Blinky example by poking registers
  - CMSIS-CORE
    - Blinky example using CMSIS-CORE structures
  - mbed HAL API
    - Blinky example using mbed HAL API functions
  - mbed API
    - Blinky example using mbed API functions



ARM

MCU Register Example

- Similar as we did in the previous labs, the GPIO peripheral can be directly accessed by writing/ reading specific memory addresses
  - Assign a pointer to the address of each register
  - Registers can be read/ written to using the pointer

```
1  #include "mbed.h"
2
3  // Reuse initialization code from the mbed library
4  DigitalOut led1(LED1); // P1_18
5
6  int main() {
7      unsigned int mask_pin18 = 1 << 18;
8
9      volatile unsigned int *port1_set = (unsigned int *)0x2009C038;
10     volatile unsigned int *port1_clr = (unsigned int *)0x2009C03C;
11
12     while (true) {
13         *port1_set |= mask_pin18;
14         wait(0.5);
15
16         *port1_clr |= mask_pin18;
17         wait(0.5);
18     }
19 }
```

Directly access MCU registers

CMSIS-CORE Example

- Alternatively, we also have tried defining a structure for the peripheral and use the structure to ease our design
- Similar to what we did, the CMSIS-CORE library defines a structure for accessing the peripherals, for instance:

```
996 /** GPIO - Register Layout Typedef */
997 typedef struct {
998   __IO uint32_t PDOR;                              /**< Port Data Output Register, offset: 0x0 */
999   __O  uint32_t PSOR;                              /**< Port Set Output Register, offset: 0x4 */
1000  __O  uint32_t PCOR;                              /**< Port Clear Output Register, offset: 0x8 */
1001  __O  uint32_t PTOR;                              /**< Port Toggle Output Register, offset: 0xC */
1002  __I  uint32_t PDIR;                              /**< Port Data Input Register, offset: 0x10 */
1003  __IO uint32_t PDDR;                              /**< Port Data Direction Register, offset: 0x14 */
1004 } GPIO_Type;
1005
1006 /* ----------------------------------------------------------------------------
1007    -- GPIO Register Masks
1008    ---------------------------------------------------------------------------- */
1009
1010 /**
1011  * @addtogroup GPIO_Register_Masks GPIO Register Masks
1012  * @{
1013  */
1014
1015 /* PDOR Bit Fields */
1016 #define GPIO_PDOR_PDO_MASK                       0xFFFFFFFFu
1017 #define GPIO_PDOR_PDO_SHIFT                      0
1018 #define GPIO_PDOR_PDO(x)                         (((uint32_t)(((uint32_t)(x))<<GPIO_PDOR_PDO_SHIFT))&GPIO_PDOR_PDO_MAS
1019 /* PSOR Bit Fields */
1020 #define GPIO_PSOR_PTSO_MASK                      0xFFFFFFFFu
1021 #define GPIO_PSOR_PTSO_SHIFT                     0
1022 #define GPIO_PSOR_PTSO(x)                        (((uint32_t)(((uint32_t)(x))<<GPIO_PSOR_PTSO_SHIFT))&GPIO_PSOR_PTSO_M
1023 /* PCOR Bit Fields */
1024 #define GPIO_PCOR_PTCO_MASK                      0xFFFFFFFFu
1025 #define GPIO_PCOR_PTCO_SHIFT                     0
1026 #define GPIO_PCOR_PTCO(x)                        (((uint32_t)(((uint32_t)(x))<<GPIO_PCOR_PTCO_SHIFT))&GPIO_PCOR_PTCO_M
1027 /* PTOR Bit Fields */
```

Data structure provided by CMSIS-CORE

ARM

CMSIS-CORE Example

- The mbed library also provides certain additions to the CMSIS-CORE layer, such as:
  - Startup file for each of the supported Toolchains
  - Linker file and support functions to define the Memory Map
  - Functions to set and get Interrupt Service Routines (ISR) addresses from the Nested Vectored Interrupt Controller (NVIC) and to program Vector Table Offset Register (VTOR)

```
1  #include "mbed.h"
2
3  // Reuse initialization code from the mbed library
4  DigitalOut led1(LED1); // P1_18
5
6  int main() {
7      unsigned int mask_pin18 = 1 << 18;
8
9      while (true) {
10         LPC_GPIO1->FIOSET |= mask_pin18;
11         wait(0.5);
12
13         LPC_GPIO1->FIOCLR |= mask_pin18;
14         wait(0.5);
15     }
16 }
```

Blink an LED using the data structure provided by CMSIS-CORE

ARM

HAL API Example

- The Hardware Abstraction Layer (HAL) Application Programming Interface (API) is a library layer that provides easy-to-use functions which are hardware platform independent
- For example, the GPIO HAL API defines the following functions:

```
typedef struct gpio_s gpio_t;

void gpio_init (gpio_t *obj, PinName pin, PinDirection direction);

void gpio_mode (gpio_t *obj, PinMode mode);
void gpio_dir  (gpio_t *obj, PinDirection direction);

void gpio_write(gpio_t *obj, int value);
int  gpio_read (gpio_t *obj);
```

GPIO functions defined in HAL API

- The HAL API is an internal interface
  - Used to help porting the mbed library to a new target and is subject to change
  - If you want to "future proof" your application avoid using this API and use the mbed API instead

mbed API Example

- The mbed API provides the actual user-friendly object-oriented API to the final user
  - More friendly functions/ APIs
  - Object oriented API (using C++)
  - Top-level API used by the majority of the programs developed on the mbed platform
  - Define basic operators to provide intuitive casting to primitive types and assignments
  - A digital IO class is defined as shown in the code clip

```
class DigitalInOut {

public:
    DigitalInOut(PinName pin) {
        gpio_init(&gpio, pin, PIN_INPUT);
    }

    void write(int value) {
        gpio_write(&gpio, value);
    }

    int read() {
        return gpio_read(&gpio);
    }

    void output() {
        gpio_dir(&gpio, PIN_OUTPUT);
    }

    void input() {
        gpio_dir(&gpio, PIN_INPUT);
    }

    void mode(PinMode pull) {
        gpio_mode(&gpio, pull);
    }

    DigitalInOut& operator= (int value) {
        write(value);
        return *this;
    }

    DigitalInOut& operator= (DigitalInOut& rhs) {
        write(rhs.read());
        return *this;
    }
```


ARM

mbed API Example

- With the support of the mbed API, the same blinky example can be programed in a much simpler and ,ore intuitive way:

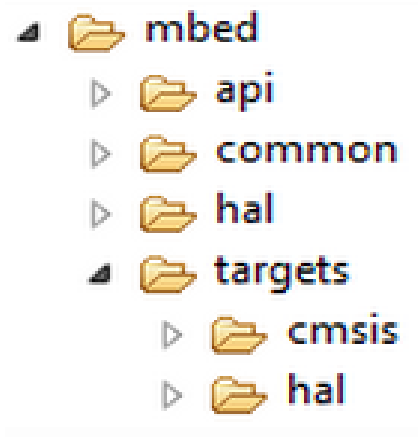
```
1  #include "mbed.h"
2
3  DigitalOut led1(LED1);
4
5  int main() {
6      while (true) {
7          led1 = 1;
8          wait(0.5);
9
10         led1 = 0;
11         wait(0.5);
12     }
13 }
```

- Note that the mbed API is programed using the object-oriented language C++, which originated from C language with object oriented features such as classes

- Deep knowledge of C++ is not necessary to use the mbed API. However, there is a large number of tutorials and books on C++ programming that you can avail of to learn C++

mbed SDK Library Directory Structure

- Three target independent directories:
  - `mbed/api`
  - The headers defining the actual mbed library API
  - `mbed/common`
  - mbed common sources
  - `mbed/hal`
  - The HAL API to be implemented for every target
- Two target dependent directories:
  - `mbed/targets/hal`
  - The HAL implementations
  - `mbed/targets/cmsis`
  - CMSIS-CORE sources



- The directory structure of the sources constituting the mbed library is published on the official mbed github repository at:
  - https://github.com/mbedmicro/mbed/tree/master/libraries

ARM

Useful Resources

- CMSIS webpage
  - http://www.arm.com/products/processors/cortex-m/cortex-microcontroller-software-interface-standard.php
- Explore the mbed platform
  - http://mbed.org/explore/
- Official mbed github repository
  - https://github.com/mbedmicro/mbed/tree/master/libraries
- mbed SDK porting
  - http://mbed.org/handbook/mbed-SDK-porting
- mbed library internals
  - http://mbed.org/handbook/mbed-library-internals
- "The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors" by Joseph Yiu, ISBN 13: 9780124080829 ISBN 10: 0124080820, 13 December 2013

ARM