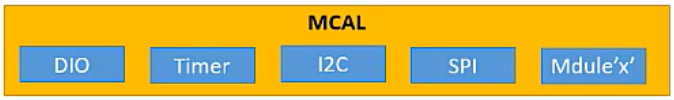
Layer Based Embedded Software Design

Modular Programming Concept

- First we need to know what's module
- What are types of files that module contains
- What's the content of each file

What is the modular programming ? Modular programming - Wikipedia

- ❑ Modular programming is a "software design technique" that emphasizes separating the functionality of a program into **independent**, **interchangeable modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.
- ❑ A module interface "Interface Header File" expresses the elements "SW Interfaces & Data Types" that are provided and required by the module.
- ❑ The elements defined in the interface are detectable by other modules.
- ❑ The implementation contains the "Working code" that corresponds to the elements declared in the interface.
- ❑ Modular programming is closely related to structured programming and object-oriented programming, all having the same goal of facilitating construction of large software programs and systems by decomposition into smaller pieces.
- ❑ Example : MCU Modules Device Driver



3

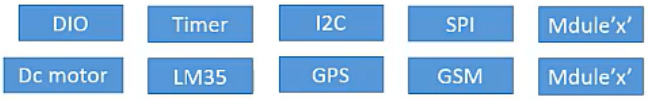
What is the modular programming ? Modular programming - Wikipedia

- Modular programming is a "software design technique" that emphasizes separating the functionality of a program into **independent, interchangeable modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.

- Software requirements analysis:
  - Group similar requirements for same task / action / activity in different groups.
  - Define software module for each requirement group.
  - Define APIs for each module based on the requirements.

- The implementation of each software module contains the "Working code" that corresponds to the elements declared in the requirements of that group.

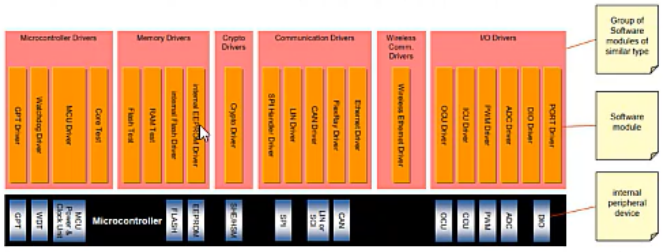
- Modular programming is closely related to structured programming and object-oriented programming, all having the same goal of facilitating construction of large software programs and systems by decomposition into smaller pieces.



What is the modular programming ? Modular programming - Wikipedia

- Each module provide a set of (APIs / Functions / Interfaces) that can be called by user software modules.
- Each module has one "Interface Header File" expresses the elements "SW Interfaces & Data Types" that are provided and required by the module.
- The elements defined in the interface are detectable by other modules.

- Group all software modules of similar type in a "Functional Group" ➔ *AUTOSAR MCAL Layer*
  - Ex. I/O Drivers – Memory Drivers - .... etc.
- Each module has:
  - A module interface "Interface Header File" expresses the elements "SW Interfaces & Data Types" that are provided and required by the module.
  - A module implementation contains the "Working code" that corresponds to the elements declared in the interface.



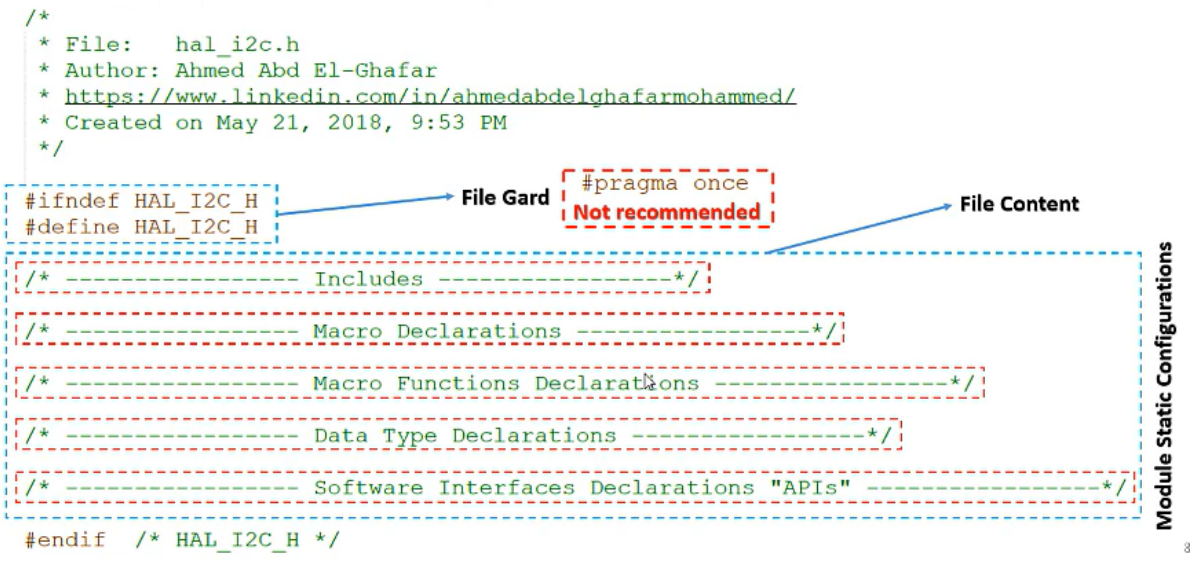
5

What is the modular programming ? Modular programming - Wikipedia

- ❑ Any software module contains at least these files:
  - ❑ Static Files
    - Contains the static code of this module (Static means not changed)
    - Static files at least consists from 2 files:
      - ✓ Static source file (Ex. HAL_GPIO.c / ECUAL_DC_Motor.c / .....)
      - ✓ Interface header file (Ex. HAL_GPIO.h / ECUAL_DC_Motor.h / .....)
  - ❑ Configuration Files
    - There are many types of configurations (Will be discussed later) and each type consists of 2 files.
      - Pre-Compile configurations (2 files)
      - Link Time configurations (2 files)
      - Post build time configurations (2 files)
  - ❑ Generators
    - Mainly using any scripting language (Python).
  - ❑ Make files
    - Facilitate the build of this files

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- ❑ **Header File Usage** (Header File Structure)



Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Include other header files needed for the module functionality)

```
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Includes -----------------*/
#include "pic18f4620.h"
#include "../mcal_std_types.h"
#include "../../MCAL_Layer/GPIO/hal_gpio.h"
#include "../../MCAL_Layer/Interrupt/mcal_internal_interrupt.h"


#endif  /* HAL_I2C_H */
```

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Any macros : #define)

```
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Macro Declarations -----------------*/
#define I2C_SLAVE_MODE_7BIT_ADDRESS                   0x06U
#define I2C_SLAVE_MODE_10BIT_ADDRESS                  0x07U
#define I2C_SLAVE_MODE_7BIT_ADDRESS_SS_INT_ENABLE     0x0EU
#define I2C_SLAVE_MODE_10BIT_ADDRESS_SS_INT_ENABLE    0x0FU
#define I2C_MASTER_MODE_DEFINED_CLOCK                 0x08U
#define I2C_MASTER_MODE_FIRMWARE_CONTROLLED           0x0BU

#endif  /* HAL_I2C_H */
```

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Any macro function : "Function like macro")

```
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Macro Functions Declarations -----------------*/
/* Slew Rate Enable/Disable */
#define I2C_SLEW_RATE_DISABLE_CFG()          (SSPSTATbits.SMP = 1)
#define I2C_SLEW_RATE_ENABLE_CFG()           (SSPSTATbits.SMP = 0)
/* SMBus Enable/Disable */
#define I2C_SMBus_DISABLE_CFG()              (SSPSTATbits.CKE = 0)
#define I2C_SMBus_ENABLE_CFG()               (SSPSTATbits.CKE = 1)

#endif  /* HAL_I2C_H */
```

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Any user defined data type: "Struct – Union - Enums" & Any Typedef)

```
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Data Type Declarations -----------------*/
typedef struct{
uint8 i2c_mode_cfg;              /* Master Synchronous Serial Port Mode Select */
uint8 i2c_slave_address;         /* I2C Slave Address */
uint8 i2c_mode : 1;              /* I2C : Master or Slave Mode */
uint8 i2c_slew_rate : 1;         /* Slew Rate Enable/Disable */
uint8 i2c_SMBus_control : 1;     /* SMBus Enable/Disable */
uint8 i2c_general_call : 1;      /* General Call Enable/Disable */
uint8 i2c_master_rec_mode : 1;   /* Master Receive Mode Enable/Disable */
uint8 i2c_reserved : 3;
#if MSSP_I2C_INTERRUPT_FEATURE_ENABLE==INTERRUPT_FEATURE_ENABLE
    interrupt_priority_cfg mssp_i2c_priority;
    interrupt_priority_cfg mssp_i2c_bc_priority;
#endif
}i2c_configs_t;

typedef struct{
    uint32 i2c_clock;    /* Master Clock Frequency */
    i2c_configs_t i2c_cfg; /* I2C Configurations */
#if MSSP_I2C_INTERRUPT_FEATURE_ENABLE==INTERRUPT_FEATURE_ENABLE
    void (*I2C_Report_Write_Collision)(void);    /* Write Collision Indicator */
    void (*I2C_DefaultInterruptHandler)(void);   /* Default Interrupt Handler */
    void (*I2C_Report_Receive_Overflow)(void);   /* Receive Overflow Indicator */
#endif
}mssp_i2c_t;

#endif  /* HAL_I2C_H */
```

```
typedef enum{
    ADC_0_TAD = 0,
    ADC_2_TAD,
    ADC_4_TAD,
    ADC_6_TAD,
    ADC_8_TAD,
    ADC_12_TAD,
    ADC_16_TAD,
    ADC_20_TAD
}adc_acquisition_time_t;
```

11

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Module Software Interfaces "APIs": used by other modules) ➔ **Non Static Functions**

```c
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Software Interfaces Declarations "APIs" -----------------*/
Std_ReturnType MSSP_I2C_Init(const mssp_i2c_t *i2c_obj);
Std_ReturnType MSSP_I2C_DeInit(const mssp_i2c_t *i2c_obj);
Std_ReturnType MSSP_I2C_Master_Send_Start(const mssp_i2c_t *i2c_obj);
Std_ReturnType MSSP_I2C_Master_Send_Repeated_Start(const mssp_i2c_t *i2c_obj);
Std_ReturnType MSSP_I2C_Master_Send_Stop(const mssp_i2c_t *i2c_obj);
Std_ReturnType MSSP_I2C_Master_Write_Blocking(const mssp_i2c_t *i2c_obj, uint8 i2c_data, uint8 *_ack);
Std_ReturnType MSSP_I2C_Master_Read_Blocking(const mssp_i2c_t *i2c_obj, uint8 ack, uint8 *i2c_data);
Std_ReturnType MSSP_I2C_Master_Write_NBlocking(const mssp_i2c_t *i2c_obj, uint8 i2c_data, uint8 *_ack);
Std_ReturnType MSSP_I2C_Master_Read_NBlocking(const mssp_i2c_t *i2c_obj, uint8 ack, uint8 *i2c_data);

#endif  /* HAL_I2C_H */
```

12

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Header File Usage** (Module "Data Sharing": Using extern)

```c
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */
uint8_t number;
```

Module Source File

```c
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ----------------- Module Data Sharing "Optional" -----------------*/
extern uint8_t number;

#endif  /* HAL_I2C_H */
```

Module Header File

13

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Good Practice Header Files Inclusion**

```c
/*
 * File:   hal_i2c.h
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#ifndef HAL_I2C_H
#define HAL_I2C_H

/* ------------------ Includes ------------------*/
#include "pic18f4620.h"
#include "../mcal_std_types.h"
#include "../../MCAL_Layer/GPIO/hal_gpio.h"
#include "../../MCAL_Layer/Interrupt/mcal_internal_interrupt.h"

#endif  /* HAL_I2C_H */
```

Module Header File

```c
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

#include "hal_i2c.h"
```

Module Source File

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Source File Usage** (Any Global variable definition (Static or Non-Static))
- **Source File Usage** (Any Function Definition (Software Interface or Helper Function))

```c
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */

/* ------------------ Global Variables Definition ------------------*/
static uint8_t number1 = 1;
uint16_t number2 = 2;

/* ------------------ APIs Definition or Helper Function Definition ------------------*/
Std_ReturnType MSSP_I2C_Init(const mssp_i2c_t *i2c_obj){
    /* Code */
}

Std_ReturnType MSSP_I2C_DeInit(const mssp_i2c_t *i2c_obj){
    /* Code */
}
```

16

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Source File Usage** (Interrupt Service Routine Definition)

```
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */
```

```
/* ----------------- ISRs Definition PIC Example -----------------*/
void __interrupt() InterruptManager(void){
    if((INTERRUPT_ENABLE == INTCONbits.INT0IE) && (INTERRUPT_OCCUR == INTCONbits.INT0IF)){
        INT0_ISR();
    }
}
```

```
/* ----------------- ISRs Definition AVR Example -----------------*/
/* External interrupt zero ISR */
ISR (INT0_vect)
{
    cnt_zero++;
}
```

17

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- **Source File Usage** (Any Helper Function Definition & Declaration "Prototype")

```
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */
```

```
/* ----------------- Helper Functions Declaration "Prototype" -----------------*/
static inline void I2C_Master_Mode_Clock_Configurations(const mssp_i2c_t *i2c_obj);
```

```
/* ----------------- APIs Definitions -----------------*/
Std_ReturnType MSSP_I2C_Init(const mssp_i2c_t *i2c_obj){
    I2C_Master_Mode_Clock_Configurations(i2c_obj);
}
```

```
/* ----------------- Helper Functions Definitions -----------------*/
static inline void I2C_Master_Mode_Clock_Configurations(const mssp_i2c_t *i2c_obj){

}
```

18

Two types of files used to construct each module : Source Files '.c' & Header Files '.h'

- ❑ **Source File Usage** (Module "Data Hiding": Using static)
- ❑ **Source File Usage** (Module Helper "Function Hiding": Using static)

```c
/*
 * File:   hal_i2c.c
 * Author: Ahmed Abd El-Ghafar
 * https://www.linkedin.com/in/ahmedabdelghafarmohammed/
 * Created on May 21, 2018, 9:53 PM
 */
/* ----------------- Includes -----------------*/

/* ----------------- Helper Functions Declaration "Prototype" -----------------*/
static inline void I2C_Master_Mode_Clock_Configurations(const mssp_i2c_t *i2c_obj);

/* ----------------- Global Variables Definitions -----------------*/
static uint8_t number = 33;

/* ----------------- APIs Definitions -----------------*/

/* ----------------- ISRs Definitions -----------------*/

/* ----------------- Helper Functions Definitions -----------------*/
static inline void I2C_Master_Mode_Clock_Configurations(const mssp_i2c_t *i2c_obj){
    number++;
}
```

19

Software Layered Architecture

- After know Modular programming conecpt let's know about Embedded Layers
- At least there are 3 Layers(**Application** layer, **ECUAL** Layer, **MCAL** Layer)
- **Application Layer :** Contians the **Algorithm** of the **App**
- **ECUAL :** Contians the Logic of **Interfacing** of the components **are used in the App**(Ex : LCD, LED, DC Motor, ……)
- **MCAL or HAL :** This layer is a composite of 2 layers(**HAL**, **LL**) and Contians the Logic of how thess components contact with **MCU**(Ex: which Pins are used(**GPIO**) to control Motors, **All MCU Peripherals** (Ex: **Timer**, **ADC**, **PWM**, **USART**, …….))
- So the calling will be at this sequence (**MCAL → ECUAL → APP**)

Embedded Software Design (Software Layered Architecture)

- Software layered architecture is an architectural pattern.
- The most common architecture pattern is the layered architecture pattern.
- Layered architecture pattern is "n-tiered pattern" where the components are organized in horizontal layers.
- This is the traditional method for designing most software and is meant to be self-independent.
  - This means that all the components are interconnected but do not depend on each other.
- The layered architecture pattern closely matches the traditional IT communication and organizational structures found in most companies.
- Components within the layered architecture pattern are organized into horizontal layers
  - Each layer performing a specific role within the application.
- The layered architecture pattern does not specify the number and types of layers that must exist in the pattern.

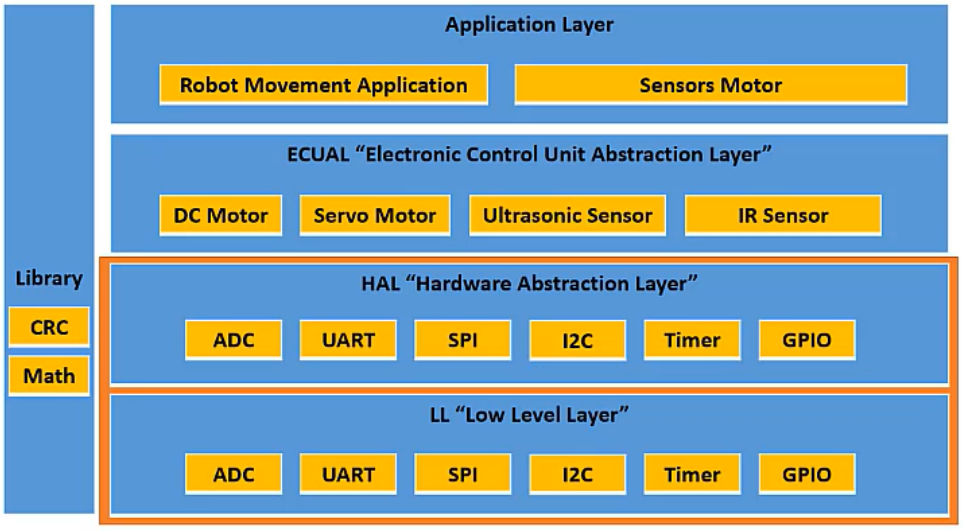
Embedded Software Design (Software Layered Architecture)

- Each layer of the layered architecture pattern has a specific role and responsibility within the application.
- Each layer in the architecture forms an abstraction around the work that needs to be done to satisfy a particular business request.
  - Example: The application layer doesn't need to know or worry how the data received by the MCU.
  - The application just needs the data to perform operation on it.
- At least a simple embedded application has 4 layers:
  1) Application Layer
  2) ECUAL "Electronic Control Unit Abstraction Layer"
  3) HAL "Hardware Abstraction Layer"
  4) LL (Low Level Layer) / MCAL "Microcontroller Abstraction Layer"

25

Embedded Software Design (Software Layered Architecture)

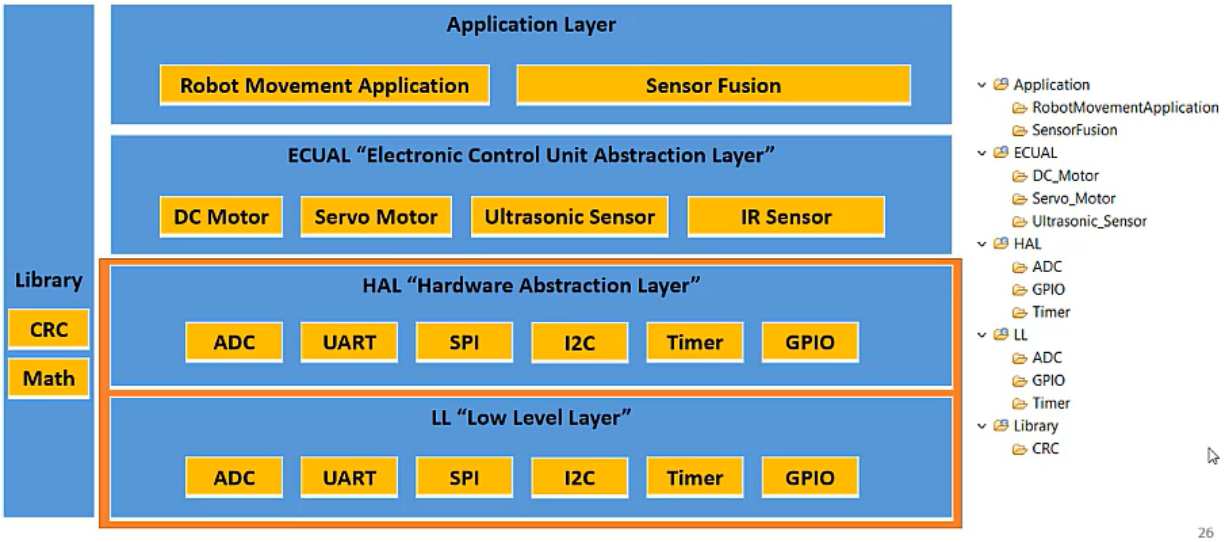
- ❑ At least a simple embedded application has 4 layers:



26

Embedded Software Design (Software Layered Architecture)

- ❑ At least a simple embedded application has 4 layers:

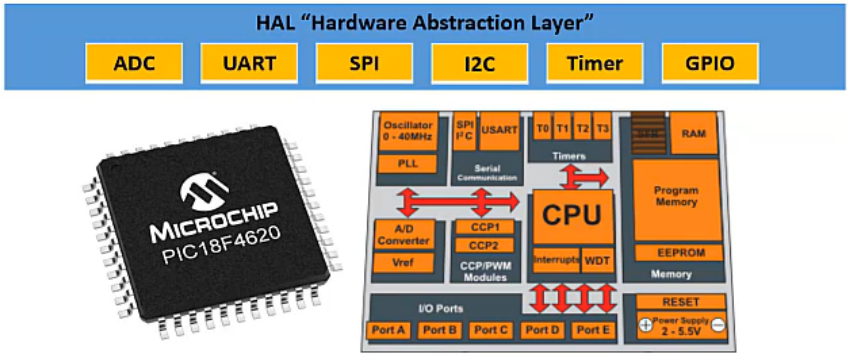


26

Embedded Software Design (Software Layered Architecture)

- One of the powerful features of the layered architecture pattern is the separation of concerns among components.

- Components within a specific layer deal only with logic that pertains to that layer.
  - In the MCAL each module/component is responsible to drive/operate a specific peripheral inside the MCU.
  - We are not expected any module/component not related to this rule.

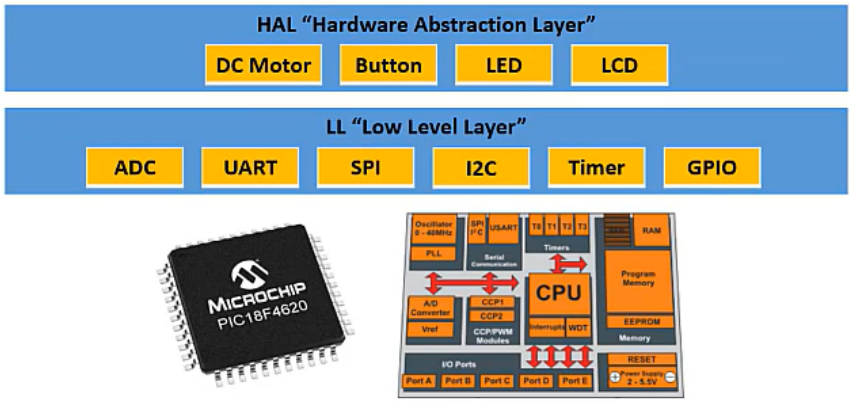


27

Embedded Software Design (Software Layered Architecture)

- This type of component classification makes it easy to:
  - Build effective roles and responsibility models into your architecture.

  - Makes it easy to develop, test, and maintain applications using this architecture pattern due to well-defined component interfaces and limited component scope.

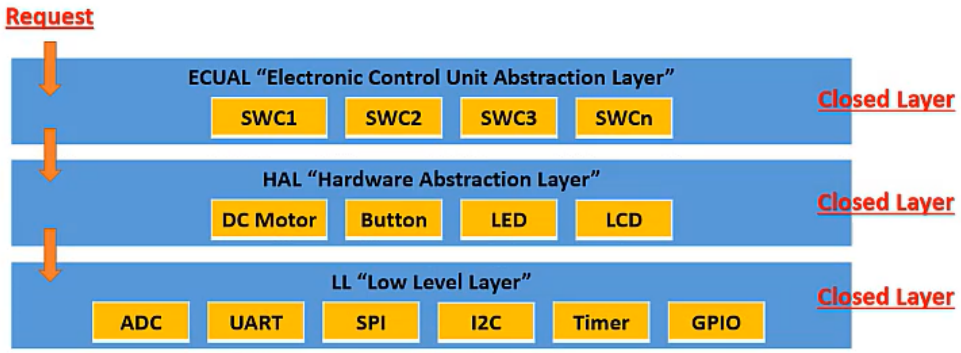


28

Embedded Software Design (Software Layered Architecture)

- ❑ In the figure blew each of the layers in the architecture is marked as being **closed**.
  - ✓ This is a very important concept in the layered architecture pattern.

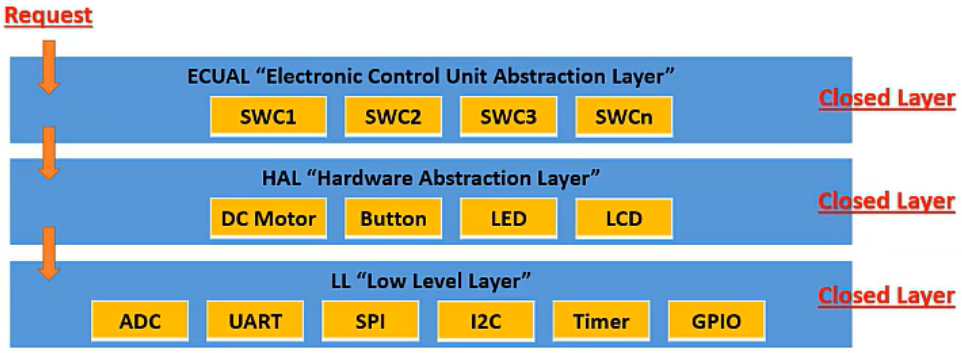
- o A closed layer means that as a request moves from layer to layer, it must go through the layer right below it to get to the next layer below that one.



Closed Layers and Request Access

Embedded Software Design (Software Layered Architecture)

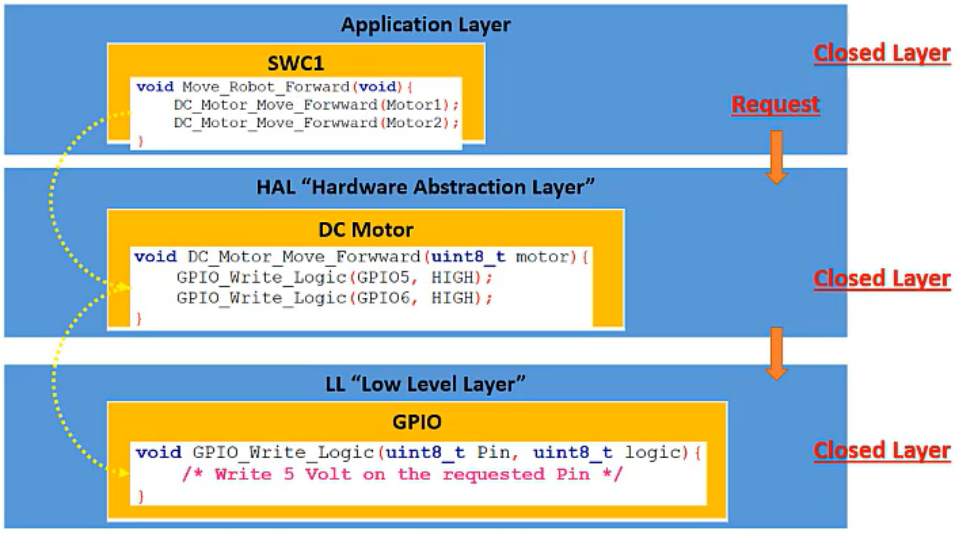
- ❑ Why the request path is from application to HAL then to LL, it will be faster if the request from application to LL directly without passing through the HAL ?
  - ✓ The answer to this question lies in a key concept known as "**layers of isolation**".
  - ✓ The layers of isolation concept means that changes made in one layer of the architecture generally don't impact or affect components in other layers.
  - ✓ The change is isolated to the components within that layer, and possibly another associated layer.



Closed Layers and Request Access

30

Embedded Software Design (Software Layered Architecture)



Closed Layers and Request Access

33

Embedded Software Design (Software Layered Architecture)

- ❑ At least a simple embedded application has 4 layers:
  1) LL (Low Level Layer):
     - This layer provides low-level APIs at register level, with better optimization but less portability.
     - They require a deep knowledge of the MCU and its peripherals.
     - The LL drivers are form a light-weight expert-oriented layer that is closer to the hardware than the HAL.
     - Features of the LL drivers:
       - ✓ A set of functions to initialize main features of a peripheral according to the parameters specified in data structures.
       - ✓ A function for de-initialization of a peripheral (registers restored to their default values).
       - ✓ A set of inline functions for direct and atomic register access.
     - Example of LL driver (GPIO):

```
#define LL_GPIO_WriteReg(__INSTANCE__, __REG__, __VALUE__) WRITE_REG(__INSTANCE__->__REG__, (__VALUE__))
#define LL_GPIO_ReadReg(__INSTANCE__, __REG__) READ_REG(__INSTANCE__->__REG__)

__STATIC_INLINE void LL_GPIO_SetPinMode(GPIO_TypeDef *GPIOx, uint32_t Pin, uint32_t Mode)
{
  register uint32_t *pReg = (uint32_t *)((uint32_t)((uint32_t)(&GPIOx->CRL) + (Pin >> 24)));
  MODIFY_REG(*pReg, ((GPIO_CRL_CNF0 | GPIO_CRL_MODE0) << (POSITION_VAL(Pin) * 4U)), (Mode << (POSITION_VAL(Pin) * 4U)));
}
```

```
LL_GPIO_DeInit(GPIO_TypeDef*) : ErrorStatus
LL_GPIO_Init(GPIO_TypeDef*, LL_GPIO_InitTypeDef*) : ErrorStatus
LL_GPIO_StructInit(LL_GPIO_InitTypeDef*) : void
```

36

Embedded Software Design (Software Layered Architecture)

- ❑ At least a simple embedded application has 4 layers:
  2) HAL "Hardware Abstraction Layer":
     - This layer provides the low-level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks).
     - It provides generic, multi-instance and function-oriented APIs which allow to offload the user application implementation by providing ready-to-use processes.
     - It provides APIs allowing to initialize and configure the peripheral, manage data transfer based on polling, interrupt or DMA process, and manage communication errors that may raise during communication.
     - **The HAL drivers APIs are split in two categories:**
       - **Generic APIs**
         - Which provide common and generic functions to all the MCUs Series to specific company.

           `<MCU_Name>_hal_GPIO.c`
           `<MCU_Name>_hal_GPIO.h`
       - **Extension APIs**
         - Which provide specific and customized functions for a specific MCU family or a specific part number.

           `<MCU_Name>_hal_GPIO_EX.c`
           `<MCU_Name>_hal_GPIO_EX.h`

37

Embedded Software Design (Software Layered Architecture)

- ❑ At least a simple embedded application has 4 layers:
  3) ECUAL "Electronic Control Unit Abstraction Layer"
     - Here we have the hardware drivers which interface the microcontroller to the outside world of sensors, IO units, displays, memories, and so on.
     - Anything on the ECU level that supports the functionality of the MCU in the embedded system shall go into this layer.
     - The application code doesn't talk directly to GPIO or ADC or whatever.
     - Example of ECUAL Layer for (LED)
       - Led_Init(const Led_t*) : Std_ReturnType
       - Led_DeInit(const Led_t*) : Std_ReturnType
       - Led_Turn_On(const Led_t*) : Std_ReturnType
       - Led_Turn_OFF(const Led_t*) : Std_ReturnType
     - Example of ECUAL Layer for (Push Button)
       - PushButton_Init(const PushButton_t*) : Std_ReturnType
       - PushButton_DeInit(const PushButton_t*) : Std_ReturnType
       - PushButton_ReadStatus(const PushButton_t*, PushButton_Status_t*) : Std_ReturnType

39