임베디드 시스템 설계
EMBEDDED SYSTEM DESIGN

CHAPTER 05

HAL 드라이버

KYUNG HEE UNIVERSITY

Human-Robot Interaction Laboratory

5.1 STM32Cube 펌웨어

STM32Cube 펌웨어의 구성

- STM32F 시리즈의 MCU 구동을 위하여 ST사에서 제공하는 펌웨어
- 주변장치(입출력 포트, 타이머 등)의 제어 및 구동을 위한 다양한 함수와 구조체가 제공
- STM32F1 시리즈 : Cortex-M3 기반, 일반적인 용도로 많이 사용함
- 32비트인 Cortex-M MCU는 대부분 레지스터에 직접 값을 써 넣는 방법보다는 제조사에서 제공하는 주변장치 구동용 함수를 이용하여 MCU를 구동하는 방식을 사용
- STM32CubeMX : STM32Cube의 초기화 코드 자동 생성 소프트웨어

5.1 STM32Cube 펌웨어

STM32Cube 펌웨어의 구성

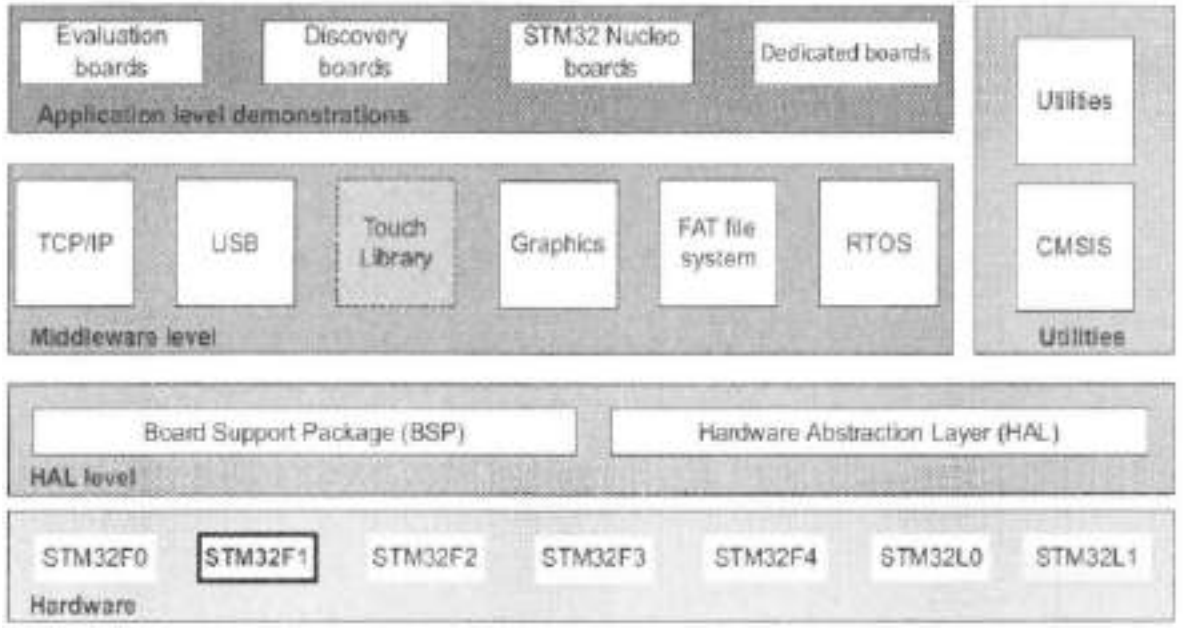
- **Application level demonstrations**
  ST사에서 제공하는 각종 Evaluation 보드, Discovery 보드, Nucleo 보드 등을 이용한 다양한 데모 프로그램 (소스코드 포함)

- **Middleware level**
  TCP/IP, USB, Touch Library, Graphics, FAT file system, RTOS를 지원. OS의 사용, 파일 관리, 인터넷 접 속, U8B 사용 등이 미들웨어를 사용하면 쉽게 구현

- **Utilities**
  ARM사의 CMSIS 지원을 위한 소스코드와 ST사에서 제공하는 CPU, Font, Log, Media 등의 유틸리티 등이 있음

- **HAL(Hardware Abstraction Layer) level**
  지원하는 MCU의 하드웨어 구동을 위한 소프트웨어로써 BSP(Board Support Package)와 HAL(Hardware Abstraction Layer) 이 있음. 해당 MCU 의 레지스터에 직접 값을 써넣거나 주변장치를 제어

- **Hardware**
  이 부분은 펌웨어가 아니며, 펌웨어가 구동되는 실제 하드웨어 (MCU)를 나타냄



STM32Cube 펌웨어의 구성

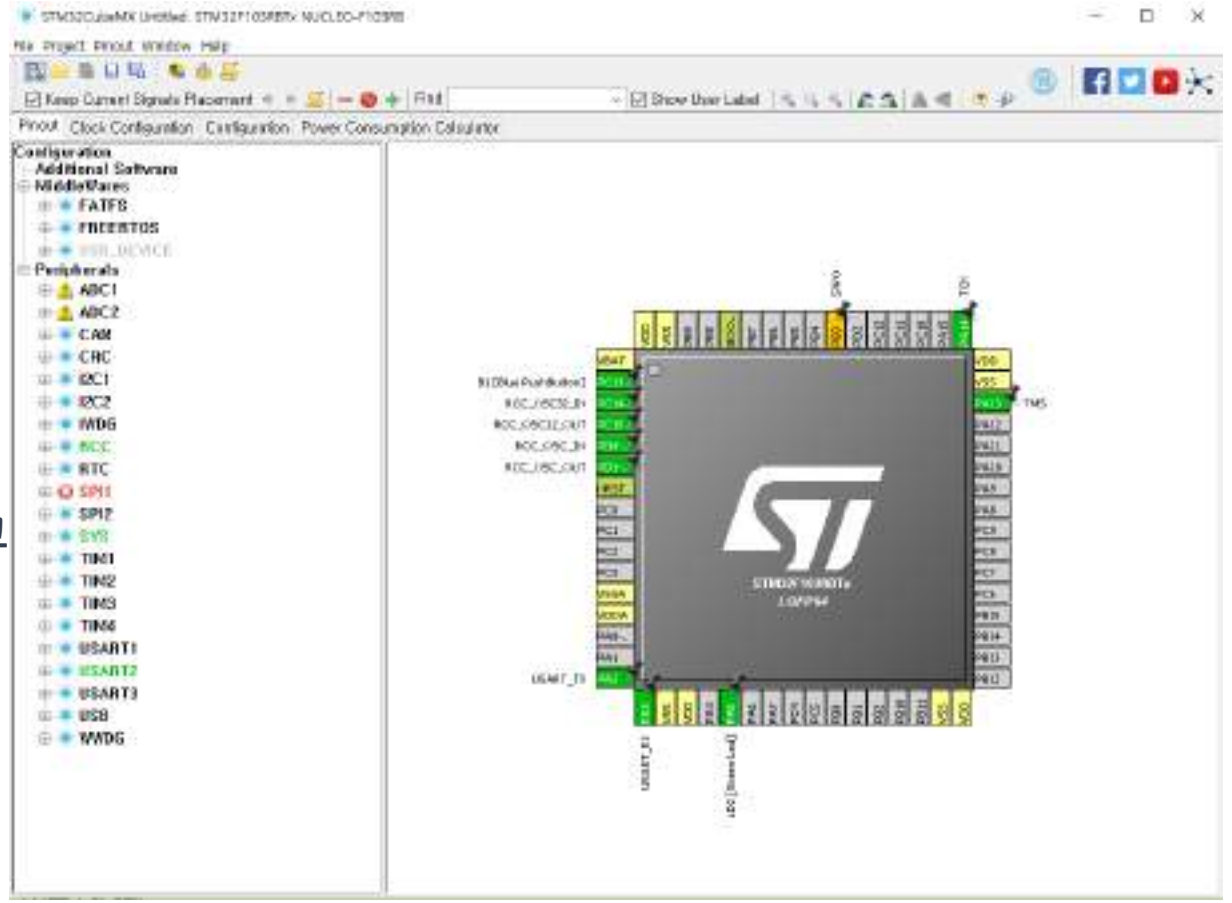
5.1 STM32Cube 펌웨어

STM32CubeMX와 CMSIS

- (1) STM32CubeMX

  ① STM32Cube 펌웨어에 포함된 소프트웨어

  ② STM32 MCU 의 동작조건 설정과 초기화를 위한 C 코드를 생성해주는 그래픽 기반의 소프트웨어



STM32CubeMX의 동작 화면의 예

5.1 STM32Cube 펌웨어

STM32CubeMX와 CMSIS

- (2) CMSIS(Cortex Microcontroller Software Interface Standard)

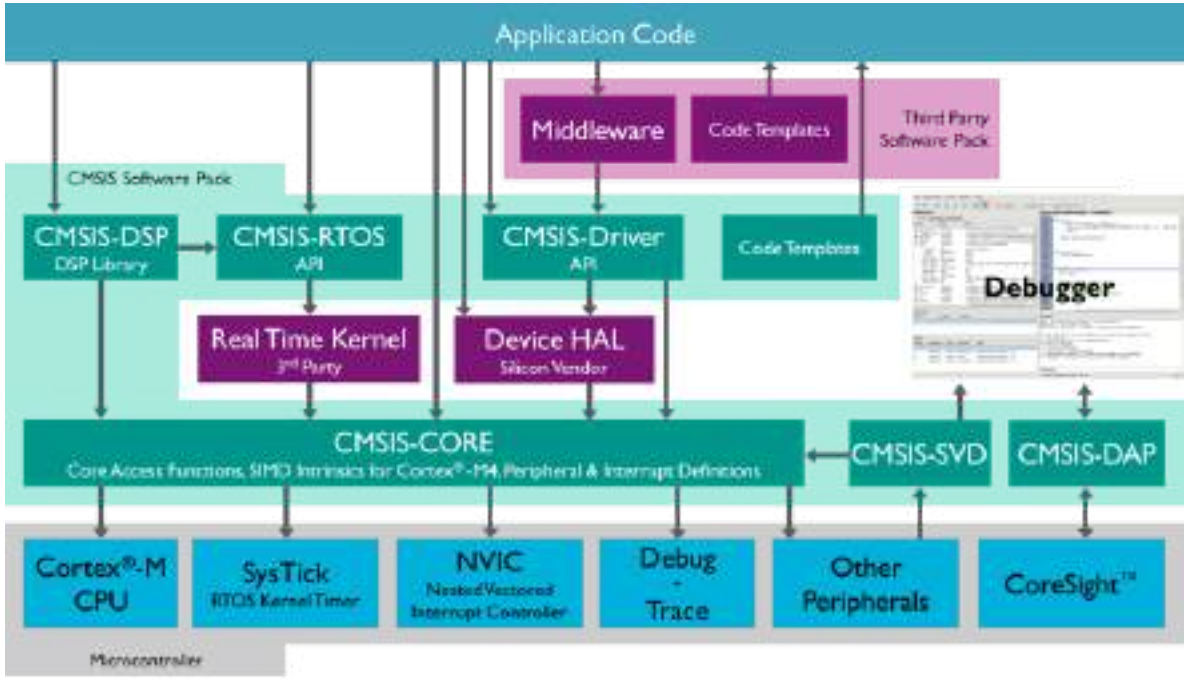
  ① ARM사에서 제안한 Cortex-M MCU의 소프트웨어 인터페이스에 대한 표준

  ② 어떤 제조사의 MCU를 사용하더라도 소프트웨어의 이식성, 재사용성을 높이기 위한 목적

  ③ CMSIS-DSP 라이브러리
  고정 소수점(fix-point)와 단정 밀도 부동 소수점 (single precision floating-point) 데이터 연산을 위한 DSP 라이브러리. 모든 Cortex-M 코어에서 사용 가능.

  ④ CMSIS-RTOS API
  RTOS(Real-Time operating system)을 위한 공통 API. 여러 RTOS로 포팅 가능한 표준 프로그래밍 인터페이스 제공.

  ⑤ CMSIS-Driver API
  일반적인 주변장치 드라이버 인터페이스를 정의한 API.



CMSIS의 구성

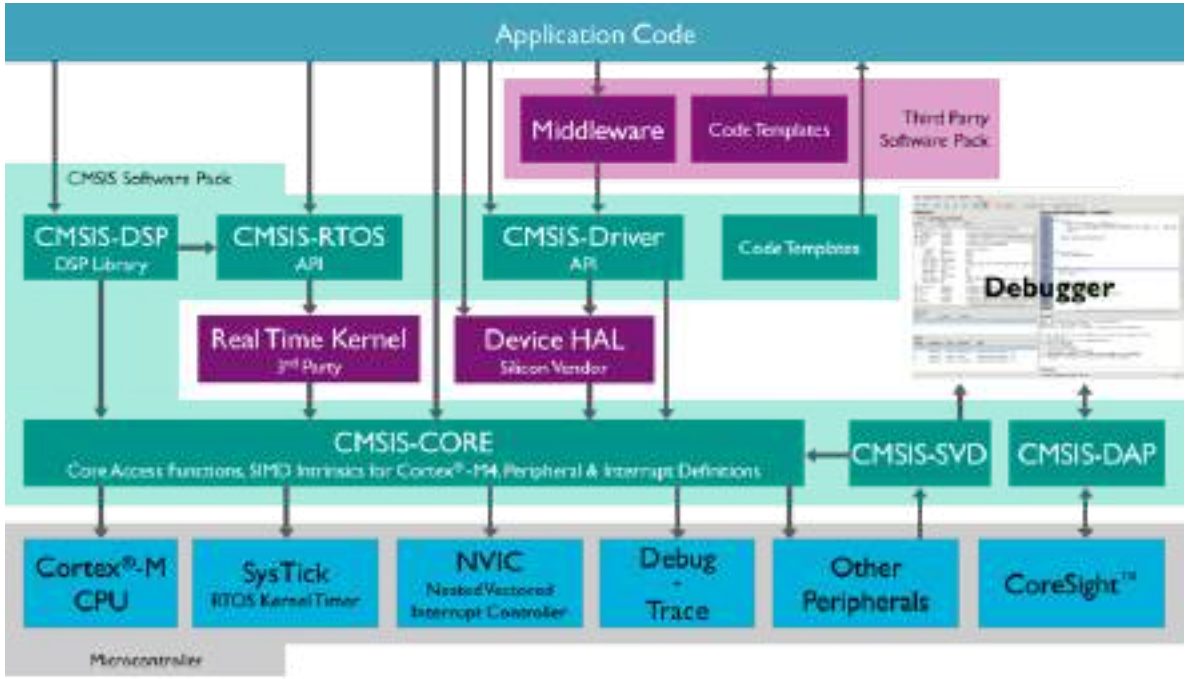
5.1 STM32Cube 펌웨어

STM32CubeMX와 CMSIS

⑥ **CMSIS-CORE**
- Cortex-M 프로세서 코어와 주변장치에 대한 API.
- Cortex-MO, Cortex-M3, Cortex-M4, Cortex-M7, SCOOO, 그리고 SC300에 대한 표준 인터페이스를 제공.
- Cortex-M4 와 Cortex-M7 의 SIMD 명령에 대한 intrinsic functions도 포함.

⑥ **CMSIS-SVD**
주변장치에 대한 SVD(System View Description) 파일. Device의 주변 장치들이 XML file로 기술 되어 있으며, Debugger 에서 주변 장치들을 인식하기 위해 사용.

⑦ **CMSIS-DAP**
Debug Access Port. CoreSight Debug Access Port에 연결되는 디버거를 위한 표준화된 펌웨어.



CMSIS의 구성

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL(Hardware Abstraction Layer) 드라이버

- MCU 의 주변장치를 제어하는 등의 동작을 위한 HAL level에서 있는 파일의 묶음.
- 여러 가지 주변장치의 설정을 위한 데이터 구조체와 주변장치의 구동을 위한 API 함수가 포함되어 있는 여러 개의 파일로 구성
- HAL 드라이버는 주변장치의 구동을 위하여 다음과 같은 종류의 데이터 구조체를 가짐.
  - ① 주변장치 핸들링용 구조체
  - ② 초기화 및 동작 조건 설정용 구조체
  - ③ 작업 수행용 구조체

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버용 데이터 구조체

- (1) 주변장치 핸들 구조체

① 주변장치/모듈의 설정, 레지스터 등과 관련된 사항을 다루기 위한 용도로 사용. 이 구조체는 다음과 같은 형태의 이름을 가짐.

```
PPP_HandleTypeDef  *handle
```

② 주변장치 핸들 구조체의 예
타이머 핸들 구조체인 TIM_HandleTypeDef는 stm32f1xx_haUim.h 에 다음과 같이 정의되어 있음.

```
/* @brief  TIM Time Base Handle Structure definition  */
typedef struct
{
  TIM_TypeDef              *Instance;   /*!< Register base address             */
  TIM_Base_InitTypeDef      Init;       /*!< TIM Time Base required parameters */
  HAL_TIM_ActiveChannel     Channel;    /*!< Active channel                    */
  DMA_HandleTypeDef        *hdma[7];    /*!< DMA Handlers array
                                         This array is accessed by a @ref TIM_DMA_Handle_index */
  HAL_LockTypeDef           Lock;       /*!< Locking object                    */
  __IO HAL_TIM_StateTypeDef  State;     /*!< TIM operation state               */
} TIM_HandleTypeDef;
```

③ 핸들 구조체를 사용하지 않는 주변 장치 : GPIO, SYSTICK, NVIC, PWR, RCC, FLASH

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버용 데이터 구조체

- **(2) 초기화 및 동작 조건 설정용 구조체**

  ① 주변장치의 초기화 및 동작 조건 설정을 위한 용도로 사용. 이 구조체는 다음과 같은 형태의 이름을 가짐.

```
PPP_InitTypeDef *handle
```

  ② 초기화 및 동작 조건 설정용 구조체의 예

  GPIO 의 초기화 및 동작조건 설정용 구조체인 GPIO_initTypeDef는 stm32f1xx_haL9pio.h에 다음과 같이 정의되어 있음.

```
/* GPIO Init structure definition  */
typedef struct
{
  uint32_t Pin;       /*!< Specifies the GPIO pins to be configured.
                           This parameter can be any value of @ref GPIO_pins_define */
  uint32_t Mode;      /*!< Specifies the operating mode for the selected pins.
                           This parameter can be a value of @ref GPIO_mode_define */
  uint32_t Pull;      /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
                           This parameter can be a value of @ref GPIO_pull_define */
  uint32_t Speed;     /*!< Specifies the speed for the selected pins.
                           This parameter can be a value of @ref GPIO_speed_define */
} GPIO_InitTypeDef;
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버용 데이터 구조체

- (3) 작업 수행용 구조체

  ① 특정한 작업을 수행하기 위해 API 함수 내에서 사용.

  ② 작업 수행용 구조체의 예

```
HAL_PPP_Process (PPP_HandleTypeDef* hadc, PPP_ProcessConfig* sConfig)
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL API의 이름 규칙

- **(1) 이름 규칙**

▼ 표 5-2-1 HAL API의 이름 규칙(naming rule)

	Generic	Family specific	Device specific
File names	stm32f1xx_hal_ppp (c/h)	stm32f1xx_hal_ppp_ex (c/h)	stm32f1xx_hal_ppp_ex (c/h)
Module name	HAL_PPP_MODULE		
Function name	HAL_PPP_Function HAL_PPP_FeatureFunction_MODE	HAL_PPP_Function HAL_PPPEx_FeatureFunction_MODE	HAL_PPPEx_Function HAL_PPPEx_FeatureFunction_MODE
Handle name	PPP_Handle Typedef	NA	NA
Init structure name	PPP_InitTypeDef	NA	PPP_InitTypeDef
Enum name	HAL_PPP_Structname Typedef	NA	NA

- **(2) 핸들(handle)이나 인스턴스 객체(instance object)를 사용하지 않는 주변 장치**
  GPIO, SYSTICK, GPIO, SYSTICK, NVIC, PWR, RCC, FLASH

- **(3) NVIC와 SYSTICK은 ARM Cortex 코어와 관련된 장치. 이 장치와 관련된 API는 “stm32f1xx_hal_cortex.c" 파일에 있음.**

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 인터럽트 핸들러 및 콜백(callback) 함수

- 주변장치 드라이버는 API 외에도 다음과 같은 인터럽트 핸들러 (interrupt handler)와 콜백함수(callback function)를 가짐.

- (1) 주변장치 인터럽트 핸들러

  ① 주변장치 인터럽트 핸들러
  ② 반드시 “stm32f1xxjt.c"에서 호출되어야 함.

- (2) 사용자 콜백 함수(User callback functions)

  ① 약한 속성(“ weak" attribute)을 가지는 빈 함수(empty function)
  ② 반드시 사용자가 직접 작성하는 소스.
  ③ 사용자 콜백 함수의 종류
  - HAL_PPP_MspInit() 와 HAL_PPP_MspDeInit()
    주변장치 초기화/초기화 해제
  - HAL_PPP_ProcessCpltCallback()
    프로세스 완료 시 호출되는 콜백 함수(Process complete callback). 주변 장치나 DMA 인터럽트 핸들러가 작업을 완료하였을 때 이 함수를 호출.
  - HAL_PPP_ErrorCallback()
    에러 발생시 호출되는 콜백 함수(Error callback). 주변장치 나 DMA 인터럽트 핸들러 에서 에러 가 발생한 경우 이 함수를 호출.

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL API의 종류

- **(1) 일반 API(Generic API)**

  ① 모든 STM32 디바이스에 공통적으로 적용되는 일반적인 함수.

  ② 초기화/초기화 해제 함수
  - HAL_PPP_Init() : 주변장치 초기화, 클럭, GPIO, DMA, 인터럽트 등의 low-level resources를 설정.
  - HAL_PPP_DeInit() : 주변장치를 디폴트 상태로 복귀. Low-level resources의 설정을 해제.

  ③ 입출력 함수
  - 주변장치의 입출력
  - HAL_PPP_Read(), HAL_PPP_Write(), HAL_PPP_ Transmit(), HAL_PP_Receive()

  ④ 제어 함수
  - 주변장치의 설정을 동적으로 변경하거나 다른 동작 모드로 전환할 때 사용.
  - HAL_PPP_Set(), HAL_PPP_Get()

  ⑤ 상태 및 에러 함수
  - 실행 도중에 주변장치의 상태와 데이터 의 상태를 알아보거나, 에러가 발생한 경우 에러의 종류를 알아보는데 사용.
  - HAL_PPP_GetState(), HAL_PPP_GetError()

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL API의 종류

- (2) HAL 확장(extension) API

① 특정 모델의 MCU에만 동작하는 함수 또는 특정한 기능을 하는 함수를 모아둔 것
② 특정 패밀리 관련 API
  - 특정 패밀리에 적용되는 API
  - ADC 에 대한 특정 패밀리 API 의 예

```
HAL_StatusTypeDef    HAL_ADCEx_Calibration_Start(ADC_HandleTypeDef*    hadc,    uint32_t
SingleDiff);
uint32_t HAL_ADCEx_Calibration_GetValue(ADC_HandleTypeDef* hadc, uint32_t SingleDiff);
```

③ 특정 디바이스 관련 API(Device part number specific API)
  - 특정 디바이스에 적용되는 API
  - ADC 에 대한 특정 디바이스 API 의 예

```
#if defined (STM32F101xG) || defined (STM32F103x6)|| defined (STM32F103xB) || defined
 (STM32F105xC) || defined (STM32F107xC) || defined (STM32F103xE) || defined (STM32F103xG)

/* ADC multimode */
HAL_StatusTypeDef HAL_ADCEx_MultiModeStart_DMA(ADC_HandleTypeDef *hadc, uint32_t *pData,
uint32_tLength);
HAL_StatusTypeDef HAL_ADCEx_MultiModeStop_DMA(ADC_HandleTypeDef *hadc);

#endif
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

시스템 주변장치 구동용 HAL API 함수

- **(1) 클럭(Clock)**

  시스템 클럭을 설정하기 위한 주요 함수

  ① HAL_RCC_OscConfig (RCC_OsclnitTypeDef *RCC_OsclnitStruct)

  여러 개의 클럭 소스(HSE, HSI, LSE, LSI, PLL)의 동작 조건을 설정하는 함수.

  ② HAL_RCC_ClockConfig (RCC_ClklnitTypeDef * RCC_ClklnitStruct, uint32_t FLatency)

  - 시스템의 클럭 소스 선택
  - AHB, APB1, APB2 클럭의 분주기 (divider) 설정
  - 플래시 메모리의 대기 상태 (wait states) 숫자 설정
  - HCLK 클럭 이 변경될 때 SysTick 설정 업데이트

  ③ HAL_RCC_Delnit() : 초기화를 해제하여 클럭 설정을 리셋 상태의 값으로 복귀

5.2 HAL(Hardware Abstraction Layer) 드라이버

시스템 주변장치 구동용 HAL API 함수

- **(2) GPIO(General Purpose I/O : 범용 입출력**

  ① GPIO를 통한 입출력용 HAL API

  HAL_GPIOJnit(), HAL_GPIO_Delnit(), HAL_GPIO_ReadPin(),
  HAL_GPIO_WritePin(), HAL_GPIO_TogglePin()

  ② 일반적인 입출력 이외에 외부 인터럽트(EXTI)의 입력용으로도 사용이 가능.

  ③ EXTI 모드에서 인터럽트 생성을 위해서는 다음의 설정이 필요.

  - 인터럽트 서비스 루틴인 stm32f1xxjt.c (또는 stm32f4xxjt.c)파일에서 HAL_GPIO_EXTI_IRQHandle() 함수를 호출.
  - 인터럽트 발생시 이에 대응하는 작업을 수행하는 소스코드를 가지고 있는 함수인 HAL GPIO EXTI Callback() 함수를 main.c 파일에 구현.

5.2 HAL(Hardware Abstraction Layer) 드라이버

시스템 주변장치 구동용 HAL API 함수

- **(3) NVIC와 SysTick 타이머**

  NVIC와 SysTick 타이머 구동을 위한 API 함수는 다음과 같으며, 이 함수들은 stm32f1xx_hal_cortex.c (또는 stm32f4xx_hal_cortex.c) 에 구현되어 있음.

  HAL_NVIC_SeIPriorily(), HAL_NVIC_SeIPriorilyGrouping(),
  HAL_NV|C_GetPriority(), HAL_NVIC_EnableIRQ(),
  HAL_NVIC_SyslemReset(), HAL_NVIC_GetPendingIRQ(),
  HAL_NVIC_SeIPendingIRQ(), HAL_NVIC_GetPriorityGrouping(),
  HAL_NVIC_DisableIRQ(), HAL_NVIC_ClearPendinglRQ(),
  HAL_NVIC_GetAclive(IRQn), HAL_SYSTICKjRQHandler(),
  HAL_SYSTICK_Config(), HAL_SYSTICK_CLKSourceConfig(),
  HAL_SYSTICK_Callback()

5.2 HAL(Hardware Abstraction Layer) 드라이버

시스템 주변장치 구동용 HAL API 함수

- **(4) 전원(PWR : Power)**

  ① PWR HAL 드라이버는 전원 관리를 위해서 사용.

  ② 모든 STM32 시리즈는 다음과 같은 PWR HAL 함수를 가짐.

  - PVD 설정, 활성화/비활성화 및 인터럽트 핸들링

    HAL_PWR_ConfigPVD(), HAL_PWR_EnablePVDO/HAL_PWR_DisablePVD(),
    HAL_PWR_PVDjRQHandler(), HAL_PWR_PVDCallback()

  - 웨이크업 핀 (Wakeup pin) 설정

    HAL_PWR_EnableWakeUpPinO/HAL_PWR_DisableWakeUpPin()

  - 저전력 (Low power) 모드 진입

    HAL_PWR_EnlerSLEEPMode(), HAL_PWR_EnlerSTOPMode(), HAL_PWR_EnterSTANDBYModeO

5.2 HAL(Hardware Abstraction Layer) 드라이버

시스템 주변장치 구동용 HAL API 함수

- (5) EXTI(EXTernal Interrupt : 외부 인터럽트)

  ① 독립된 주변장치가 아니라 주변장치가 사용하는 서비스 동작의 하나.

  ② 각 주변장치마다 각각의 EXTI 동작 설정과 EXTI 함수가 매크로로 구현되어 있음.

  ③ EXTI1 ~ EXTI16

  GPIO에 연결되어 있으며 GPIO 드라이버에 의해 관리. GPIOjnitTypeDef 구조체를 이용하면 각각의 1/0를 외부 인터럽트 또는 외부 이벤트로 사용하도록 설정이 가능.

- (6) DMA(Direct Memory Access)

  ① DMA HAL 드라이버를 이 용하면 주변장치를 DMA 채널 에 연결할 수 있음.

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

- **(1) HAL의 초기화**

① HAL의 초기화
HAL 코어 (stm32f1xx_hal.c / stm32f4xx_hal.c 파일에 구현)를 초기화하기 위하여 다음과 같은 함수들이 사용.

- HAL_Init()
  - 프로그램 이 시작될 때 반드시 호출되어야 함.
  - 데이터 명령어 캐시 (data/instruction cache)와 pre-fetch queue를 초기화.
  - Systick timer가 매 1ms마다(based on HSI clock) 인터럽트를 발생하도록 설정. 이 인터럽트는 가장 낮은 우선 순위(lowest priority).
  - 시스템 레벨의 초기화(Clock, GPIOs, DMA, interrupts)를 하기 위해서 HAL_Msplnit() user callback function을 호출. HAL_Msplnit()는 HAL 드라이버 내에서 약한 빈(weak empty) 함수로 정의.
- HAL Delnit()
  - 초기화 해제
  - 모든 주변장치를 리셋
  - HAL_MspDelnit() 함수를 호출. 이 함수는 시스템 레벨의 초기화 해제를 실행하는 사용자 콜백 함수.
- HAL_GetTick()
  - 현재 의 SysTick 카운터의 값을 읽어옴.
- HAL_Delay()
  - SysTick 타이 머 를 이용하여 시 간지연 (delay) 를 구현하는 함수.
  - 단위는 msec.

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

② 시스템 클럭의 초기화(System clock initialization)

- 시스템 클럭 설정은 사용자 작성 코드의 시작 부분에서.
- 사용자가 시용 조건에 맞도록 클럭 설정을 수정 하는 것도 가능.

```
void SystemClock_Config(void)
  {
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  /* Configure PLLs ------------------------------------------------*/
  /* PLL2 configuration: PLL2CLK = (HSE/HSEPrediv2Value)*PLL2MUL=(25/5)*8=40MHz */
  /* PREDIV1 configuration: PREDIV1CLK = PLL2CLK/HSEPrediv1Value = 40/5 = 8MHz */
  /* PLL configuration: PLLCLK = PREDIV1CLK * PLLMUL = 8 * 9 = 72MHz */
  /* Enable HSE Oscillator and activate PLL with HSE as source */

  oscinitstruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  oscinitstruct.HSEState = RCC_HSE_ON;
  oscinitstruct.HSEPrediv1Value = RCC_HSE_PREDIV_DIV5;
  oscinitstruct.Prediv1Source = RCC_PREDIV1_SOURCE_PLL2;
  oscinitstruct.PLL.PLLState = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  oscinitstruct.PLL.PLLMUL = RCC_PLL_MUL9;
  oscinitstruct.PLL2.PLL2State = RCC_PLL2_ON;
  oscinitstruct.PLL2.PLL2MUL = RCC_PLL2_MUL8;
  oscinitstruct.PLL2.HSEPrediv2Value = RCC_HSE_PREDIV2_DIV5;

  if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)
    { /* Initialization Error */
    while(1);
    }
  /* Select PLL as system clock source and configure the HCLK/PCLK1/PCLK2 clock
  dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK |
```

```
          RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
  clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)
    { /* Initialization Error */
    while(1);
    }
}
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

- **(2) HAL의 입출력 프로세스**
  HAL 함수의 데이터 입출력 모드 : 폴링(Polling), 인터럽트(interrupt), DMA(Direct Memory Access)

① 폴링 모드
- 폴링 모드에서 HAL 함수는 블로킹 (blocking) 모드에서 작업.
- 작업이 완료되면 처리 결과(또는 프로세스 상태 값)를 리턴.
- 함수에서 작업이 성공적으로 완료되면 HAL_OK 값을 리턴하고, 에러가 발생하면 에러의 상태 값을 리턴.
- HAL 함수 내에 서의 처리 지연, 무한 대 기 등을 방지하기 위해 타임아웃(timeout, msec) 값이 사용.
- 폴링 모드에서 작업 과정의 예

```
HAL_StatusTypeDef  HAL_PPP_Transmit ( PPP_HandleTypeDef * phandle,
                                      uint8_t pData,int16_tSize,uint32_tTimeout)
{
  if((pData == NULL) || (Size == 0)) {
    return HAL_ERROR;
  }
  (…)
  while (data processing is running) {
      if( timeout reached) {
      return HAL_TIMEOUT;
    }
  }
  (…)
  return HELIAC;
}
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

② 인터럽트 모드

- 인터럽트 모드에서 HAL 함수는 정해진 작업이 수행되면 인터럽트를 발생시켜 프로세스 상태 값을 리턴
- 작업 이 완료되면 미리 정해진 콜백 함수가 호출되어 실행
- 인터럽트 모드를 위해서 다음의 4개의 함수가 HAL 드라이버에 선언되어 있음.
  - ➢ HAL_PPP ProcessJT()
    인터럽트 모드에서 프로세스를 시작하는 함수
  - ➢ HAL_PPP JRQHandler()
    인터럽트 모드에서 프로세스를 시작하기 전에 stm32f1xx_it.c 파일에서 해당되는 핸들러가 코딩되어 있어야 함.
  - ➢ _weak HAL_PPP_ProcessCpltCallback()
    작업 완료시 호출되는 콜백 함수
  - ➢ _weak HAL_PPP_ProcessErrorCallback()
    작업시 에리가 빌생하면 호출되는 콜백 함수

[main.c]

```
UART_HandleTypeDef UartHandle;

int main(void)
{
        /* Set User Parameters */
        UartHandle.Init.BaudRate = 9600;
        UartHandle.Init.WordLength = UART_DATABITS_8;
        UartHandle.Init.StopBits = UART_STOPBITS_1;
        UartHandle.Init.Parity = UART_PARITY_NONE;
        UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        UartHandle.Init.Mode = UART_MODE_TX_RX;
        UartHandle.Init.Instance = USART1;
        HAL_UART_Init(&UartHandle);
        HAL_UART_SendIT(&UartHandle, TxBuffer, sizeof(TxBuffer));
        while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
        .. (여기에 원하는 작업을 하는 코드를 작성한다.) ..
}

void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
        .. (여기에 원하는 작업을 하는 코드를 작성한다.) ..
}
```

[stm32f1xx_it.c]

```
extern UART_HandleTypeDef UartHandle;

void USART1_IRQHandler(void)
{
        HAL_UART_IRQHandler(&UartHandle);
}
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

② DMA 모드

- 이 모드에서 HAL 함수는 DMA을 통해 작업을 시작
- 해당되는 DMA 인터럽터를 인에이블 시킨 후에 프로세스의 상태 값을 리턴
- 작업이 완료되면 미리 정해진 콜백 함수가 호출되어 실행
- DMA 모드를 위해서 다음의 4개의 함수가 HAL 드라이버에 선언되어 있음.
  - ➢ HAL_PPP_Process_DMA()
    DMA 모드에서 프로세스를 시작하는 함수
  - ➢ HAL_PPP_DMAJRQHandler()
    DAM 모드에서 프로세스를 시작하기 전에 stm32f1xx_it.c 파일에서 해당되는 핸들러가 코딩되어 있어야 함.
  - ➢ _weak HAL_PPP_ProcessCpltCallback()
    작업 완료시 호출되는 콜백 함수
  - ➢ _weak HAL_PPP_ErrorCpltCallback()
    작업시 에리가 빌생하면 호출되는 콜백 함수

[main.c]

```
UART_HandleTypeDef UartHandle;

int main(void)
{
        /* Set User Parameters */
        UartHandle.Init.BaudRate = 9600;
        UartHandle.Init.WordLength = UART_DATABITS_8;
        UartHandle.Init.StopBits = UART_STOPBITS_1;
        UartHandle.Init.Parity = UART_PARITY_NONE;
        UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
        UartHandle.Init.Mode = UART_MODE_TX_RX; UartHandle.Init.Instance = USART1;
        HAL_UART_Init(&UartHandle);
        HAL_UART_Send_DMA(&UartHandle, TxBuffer, sizeof(TxBuffer));
        while (1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *phuart)
{
        .. (여기에 원하는 작업을 하는 코드를 작성한다.) ..
}
void HAL_UART_TxErrorCallback(UART_HandleTypeDef *phuart)
{
        .. (여기에 원하는 작업을 하는 코드를 작성한다.) ..
}
```

[stm32f1xx_it.c]

```
extern UART_HandleTypeDef UartHandle;

void DMAx_IRQHandler(void)
{
        HAL_DMA_IRQHandler(&UartHandle.DMA_Handle_tx);
}
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

② DMA 모드

- 그리고 HAL_PPP Process_DMA() 함수 내에서 HAL_USART_TxCpltCallback()와 HAL_USART_ErrorCallback() 함수가 반드시 다음 코드와 같이 링크되어 있어야 함.

```
HAL_PPP_Process_DMA (PPP_HandleTypeDef *hppp, Params….)
{
        (…)
        hppp->DMA_Handle->XferCpltCallback = HAL_UART_TxCpltCallback ;
        hppp->DMA_Handle->XferErrorCallback = HAL_UART_ErrorCallback ;
        (…)
}
```

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

- (3) 타임아웃(Timeout)과 에러

① 타임아웃
- 블로킹 프로세스에서 작업 결과가 리턴되기까지 기다리는 최대 대기 시간을 지정하는 것
- 폴링모드에서 동작하는 API의 경우에 종종 사용
- 타임아웃을 사용하는 예

```
HAL_StatusTypeDef HAL_DMA_PollForTransfer (DMA_HandleTypeDef *hdma,
                                           uint32_t CompleteLevel, uint32_t Timeout)
```

- 어떤 경우에는 시스템 주변 장치나 내부 HAL 드라이버 프로세스에서 고정된 타임아웃(fixed timeout) 값이 사용되기도 함. 이 경우는 사용자 프로그램에서 이 값을 변경할 수 없으며, 이 값을 함수의 인자로 사용할 수도 없음.

② 에러 관리(Error management)
주변장치 의 프로세스 중에서 에러가 발생하면 HAL_PPP_Process() 함수는 발생한 에러에 해당되는 HAL_ERROR() 상태 값을 리턴
- 파라미터의 적절성
- 핸들(handle) 의 적절성
- 타임아웃 에러

5.2 HAL(Hardware Abstraction Layer) 드라이버

HAL 드라이버의 사용방법

③ 실행시 체크(Run-time checking)

- 모든 HAL 함수는 실행 시에 입력된 값이나 인자로 전달받은 값의 유효성 여부를 확인. 이 과정은 assert_param 매크로를 이용하여 이루어짐.
- 이 기능은 프로그램 실행 시에 오버헤드(overhead)로 작용하여 프로그램의 크기와 실행 속도를 떨어뜨리게 됨. 따라서 이 기능은 프로그램 개발 시에만 사용하고 최종의 실행판 소스코드에서는 사용하지 않는 것이 좋음.

```
/* Exported macro ------------------------------------------------------------*/
#ifdef USE_FULL_ASSERT
/**
* @brief The assert_param macro is used for function's parameters check.
* @param expr : If expr is false, it calls assert_failed function
* which reports the name of the source file and the source
* line number of the call that failed.
* If expr is true, it returns no value.
* @retval None */
#define assert_param(expr) ((expr)?(void)0:assert_failed((uint8_t *)__FILE__,
__LINE__))
/* Exported functions --------------------------------------*/
void assert_failed(uint8_t* file, uint32_t line);
#else
#define assert_param(expr)((void)0)
#endif /* USE_FULL_ASSERT */
```

5.3 HAL 시스템 드라이버 및 제어용 함수

초기화(Initialization) 및 초기화 해제(de-initialization)용 함수

- HAL_Init (void)

  플래시 메모리의 인터페이스 NVIC 할당 및 클럭 동작 설정 등을 초기화하는 함수

- HAL_DeInit (void)

  초기화를 해제하는 함수

- HAL_MspInit (void)

  MSP(MCU Specific Package)를 초기화하는 함수

- HAL_MspDeInit (void)

  MSP의 초기화를 해제하는 함수

- HALJnitTick (uint32_t TickPriority)

  SysTick 타이머 를 초기화는 함수.

5.3 HAL 시스템 드라이버 및 제어용 함수

HAL 제어(Control)용 함수

- HAL_IncTick (void)

  : 타임 베이스에 사용되는 전역 변수 “uwTick" 의 값을 증가시키는 함수

- HAL_GetTick (void)

  : 현재의 Tick 값을 얻어옴. 단위는 msec.

- HAL_Delay (_10 uint32_t Delay)

  : 블로킹된 시간 지연 (blocking delay)을 하는 함수. 단위는 msec.

- HAL_SuspendTick (void)

  : 타임 베이스를 소스로 하는 인터럽트(time base source interrupt) 발생을 재개시키는 함수

- HAL_GetHalVersion()

  : HAL API driver의 version을 읽어오는 함수

- HAL_GetDEVID(), HAL_GetREVID()

  : 디바이스의 식별자(identifier) / 리비전 식별자(revision identifier)를 읽어옴.

- HAL_DBGMCU_Enable(Disable)DBGSleepMode()

  : Sleep 모드에서 Debug 모듈을 Enable/Disable 시키는 함수

- HAL_DBGMCU_Enable(Disable)DBGStopMode()

  : HAL API driver의 verslon을 읽어오는 함수

- HAL_GetHalVersion()

  : STOP 모드에서 Debug 모듈을 Enable/Disable 시키는 함수

- HAL DBGMCU Enable(Disable)DBGStandbyMode()

  : STANDBY 모드에서 Debug 모듈을 Enable/Disable 시키는 함수

5.4 HAL RCC(Reset and Clock Control) 드라이버

RCC 설정용 구조체

- HAL RCC 드라이버 : 리셋, 클럭 등의 동작 조건을 설정하는 함수
- 클럭의 동작 조건을 설정하기 위해서는 RCC의 동작 조건 설정용 레지스터를 원하는 값으로 설정을 하여야 함.

RCC_ClkInitTypeDef

클럭의 초기설정용 구조체이며 stm32f1xx_hal_rcc.h(또는 stm32f4xx_hal_rcc.h)에 정의되어 있다.

[데이터 형]

- uint32_t ClockType
- uint32_t SYSCLKSource
- uint32_t AHBCLKDivider
- uint32_t APB1CLKDivider
- uint32_t APB2CLKDivider

[데이터 형의 설명]

- ClockType : 설정할 클럭. 이 파라미터는 RCC_System_Clock_Type 중의 값이어야 한다.
- SYSCLKSource : 시스템 클럭(SYSCLKS)으로 사용할 클럭 소스. 이 파라미터는 RCC_ System_Clock_Source 중의 값이어야 한다.
- AHBCLKDivider : AHB 클럭(HCLK)의 디바이더(divider). 이 파라미터는 RCC_AHB_ Clock_Source 중의 값이어야 한다. AHB 클럭(HCLK)은 시스템 클럭에 의해 동작된다.
- APB1CLKDivider : APB1 클럭(PCLK1)의 디바이더(divider). 이 파라미터는 RCC_ APB1_APB2_Clock_ Source 중의 값이어야 한다. APB1 클럭은 AHB 클럭에 의해 동작된다.
- APB2CLKDivider : APB2 클럭(PCLK2)의 디바이더(divider). 이 파라미터는 RCC_ APB1_APB2_Clock_ Source 중의 값이어야 한다. APB2 클럭은 AHB 클럭에 의해 동작된다.

RCC_PLLInitTypeDef

PLL의 초기설정용 구조체이며 stm32f1xx_hal_rcc.h (또는 stm32f4xx_hal_rcc.h)에 정의되어 있다.

[데이터 형]

- uint32_t PLLState
- uint32_t PLLSource
- uint32_t PLLMUL

[데이터 형의 설명]

- PLLState : The new state of the PLL의 새로운 state. 이 파라미터는 RCC_PLL_ Config 중의 값이어야 한다.
- PLLSource : PLL 진입 클럭 소스. 이 파라미터는 RCC_PLL_Clock_Source 중의 값이어야 한다.
- PLLMUL : PLL VCO input clock의 곱하기(Multiplication) 값.
  이 파라미터는 RCCEx_ PLL_Multiplication_ Factor 중의 값이어야 한다.

5.4 HAL RCC(Reset and Clock Control) 드라이버

RCC 구동용 HAL 함수

- (1) RCC의 특징

  - 리셋 후에 클럭과 관련된 동작은 다음과 같음.

    ① 리셋 후에 디바이스는 내부의 고속 오실레이터 (HSI : Internal High Speed oscillator, 8 MHz)로 동작. 그리고 플래시 프리패치(prefetch) 버퍼가 활성화되며 , 내부 SRAM, 플래시, JTAG를 제외한 모든 주변 장치는 OFF.
    ② AHB 버스(고속)와 APB 버스(저속)는 프리스케일리가 설정되지 않음. 그리고 이 버스와 매핑된 모든 주변장치들은 HSI 속도로 동작.
    ③ SRAM과 FLASH를 제외한 모든 주변장치의 클럭은 OFF.
    ④ 모든 GPIO는 입력 플로팅(input floating) 상태로 설정된다. 단 JTAG 관련 핀들은 디버깅 용으로 설정.

  - 디바이스가 리셋 후에 동작을 시작할 때 사용자 프로그램에서 다음을 설정

    ① 시스템 클럭을 구동할 클럭 소스를 설정(디폴트의 HSI 보다 더 빠른 동작을 원하는 경우)
    ② 시스템 클럭의 주파수와 플래시의 설정
    ③ AHB, APB 버스의 프리스케일러 (prescaler)의 설정
    ④ 사용할 주변장치의 클럭을 활성화
    ⑤ 시스템 클럭으로 구동되지 않는 주변장치 (12S , RTC, ADC, USB OTG FS) 에 대 한 클럭 소스를 설정

5.4 HAL RCC(Reset and Clock Control) 드라이버

RCC 구동용 HAL 함수

- **(2) 초기화(Initialization) 및 초기화 해제(de-initialization)용 함수**

HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

CPU와 AHB, APB 버스의 클럭의 동작조건을 설정해주는 함수

[파라미터]

- RCC_ClkInitStruct : 동작조건 설정을 위한 RCC_ClkInitTypeDef 구조체형 변수
- FLatency : FLASH 동작시간(Latency). 이 파라미터는 다음 중의 1개 값을 가진다.
  - FLASH_LATENCY_0 : FLASH 0 Latency cycle
  - FLASH_LATENCY_1 : FLASH 1 Latency cycle
  - FLASH_LATENCY_2 : FLASH 2 Latency cycle

HAL_RCC_ClockConfig (RCC_ClkInitTypeDef * RCC_ClkInitStruct, uint32_t FLatency)

CPU와 AHB, APB 버스의 클럭의 동작조건을 설정해주는 함수

[파라미터]

- RCC_ClkInitStruct : 동작조건 설정을 위한 RCC_ClkInitTypeDef 구조체형 변수
- FLatency : FLASH 동작시간(Latency). 이 파라미터는 다음 중의 1개 값을 가진다.
  - FLASH_LATENCY_0 : FLASH 0 Latency cycle
  - FLASH_LATENCY_1 : FLASH 1 Latency cycle
  - FLASH_LATENCY_2 : FLASH 2 Latency cycle

HAL_RCC_OscConfig (RCC_OscInitTypeDef * RCC_OscInitStruct)

RCC 동작조건 설정을 위한 RCC_OscInitTypeDef 구조체형 변수

HAL_RCC_DeInit()

RCC 클럭의 설정을 리셋후의 디폴트 상태로 복귀시키는 함수

5.4 HAL RCC(Reset and Clock Control) 드라이버

RCC 구동용 HAL 함수

- (3) 주변장치 제어용 함수

  RCC 의 주변장치 제어용 함수는 RC의 클럭 주파수를 제어하는역할을 하는 함수

  HAL_RCC_MCOConfig()
  HAL RCC EnableCSS()
  HAL_RCC_DisableCSS()
  HAL_RCC_GetSysClockFreq()
  HAL_RCC_GetHCLKFreq()
  HAL_RCC_GetPCLK1Freq()
  HAL_RCC_GetPCLK2Freq()
  HAL_RCC_GetOscConfig()
  HAL_RCC_GetClockConfig()
  HAL_RCC_NMUROHandler()
  HAL_RCC_CSSCallback()

5.5 HAL CORTEX 드라이버

CORTEX 구동용 HAL 함수

- 인터럽트(NVIC), SysTick 등의 동작조건을 설정 하는 함수
- (1) 초기화, 초기화 해제 함수(Initialization and de-initialization functions)

① HAL_NVIC_SetPriority()

인터럽트의 우선 순위(priority)를 설정

② HAL_NVIC_SetPriorityGrouping()

인터럽트의 우선 순위 그룹(priority grouping)을 설정

③ HAL_NVIC_EnableIRQ()

특정한 주변장치의 인터럽트를 활성화(인에이블)

① HAL_NVIC_DisableIRQ()

특정한 주변장치의 인터럽트를 비활성화

② HAL_NVIC_SystemReset()

MCU를 리셋하기 위해 시스템 리셋 요구(system reset request) 를 발생

③ HAL_SYSTICK_Config()

시스템 타이머 및 관련 인터럽트를 초기화하고 System Tick 타이머의 동작을 시작

5.5 HAL CORTEX 드라이버

CORTEX 구동용 HAL 함수

- **(1) 동작제어 함수(Peripheral Control functions)**

① HAL_NVIC_GetPriority()

인터럽트의 우선 순위를 읽어옴.

② HAL_NVIC_GetPriorityGrouping()

인터럽트의 우선 순위 그룹(priority grouping)을 읽어옴.

③ HAL_NVIC_SetPendingIRQ()

외부 인터럽트의 펜딩 비트(Pending bit)를 1로 설정

④ HAL_NVIC_GetPendingIRQ()

펜딩 인터럽트(Pending Interrupt) 을 읽어옴.

① HAL_NVIC_ClearPendingIRQ()

외부 인터럽트의 펜딩 비트(Pending bit)를 클리어

② HAL_NVIC_GetActive()

액티브 인터럽트(active interrupt)를 얻음.

③ HAL_SYSTICK_CLKSourceConfig()

SysTick 클럭 소스의 동작 조건을 설정

④ HAL_SYSTICK_IRQHandler()

SYSTICK 인터럽트를 처리하는 함수

⑤ HAL_SYSTICK_Callback()

SYSTICK 콜백 함수

5.5 HAL CORTEX 드라이버

주요 함수(Cortex 구동용 함수)의 상세 설명

HAL_NVIC_GetPendingIRQ (IRQn_Type IRQn)

펜딩 인터럽트(Pending Interrupt)을 읽어온다.

[파라미터]
- IRQn : 외부 인터럽트(EXTI : External interrupt)의 번호

[리턴 값]
- 0 : 인터럽트가 펜딩되어 있지 않음
- 1 : 인터럽트가 펜딩됨

HAL_NVIC_ClearPendingIRQ (IRQn_Type IRQn)

외부 인터럽트의 펜딩 비트(Pending bit)를 클리어한다.

[파라미터]
- IRQn : 외부 인터럽트(EXTI : External interrupt)의 번호

[리턴 값] 없음

HAL_NVIC_GetActive (IRQn_Type IRQn)

액티브 인터럽트(active interrupt)를 읽어온다.

[파라미터]
- IRQn : 외부 인터럽트(EXTI : External interrupt)의 번호

[리턴 값]
- 0 : 인터럽트가 펜딩되어 있지 않음
- 1 : 인터럽트가 펜딩됨

HAL_SYSTICK_CLKSourceConfig (uint32_t CLKSource)

SysTick 클럭 소스를 설정(선택)한다.

[파라미터]
- CLKSource : SysTick 클럭 소스를 지정한다. 이 파라미터는 다음 값 중의 하나를 가진다.
- SYSTICK_CLKSOURCE_HCLK_DIV8 : AHB clock을 8로 나눈 값(AHB clock/8)을 클럭 소스로 사용
- SYSTICK_CLKSOURCE_HCLK : AHB clock을 클럭 소스로 사용

[리턴 값] 없음

HAL_SYSTICK_IRQHandler (void)

SYSTICK 인터럽트를 처리하는 IRQ 함수

[리턴 값] 없음

HAL_SYSTICK_Callback (void)

SYSTICK 콜백 함수

[리턴 값] 없음