분석 - UART _ POLLING

▼ 전체적인 File 정의에 대한 내용

가장 중요한 흐름은 크게 세부분으로 나눌 수가있음 // 사용자 코드(Main) → HAL DRIVER(hal~~~~) → CMSIS

- `Main.c / Main.h`
  - 사용자 함수가 사용될 내용 내가 사용할 부분
- `stm32fxxx_hal_conf.h`:
  - CUBE MX에서 내가 누른 (configuration 한 부분) 정보들이 포함되어있는 부분임 Setting
  - 여기서 각 HAL 드라이버 파일 include 시키는 설정이 있음 ex) **#include** "stm32f1xx_hal_uart.h"
  - CallBack resister enable 항목 정의되어있음 → **#define USE_HAL_USART_REGISTER_CALLBACKS** 0U /* USART register callback disabled */
  - HAL 라이브러리의 전반적인 구성과 설정을 정의하는 헤더 파일
  - 각 하드웨어 모듈의 HAL 드라이버 활성화/비활성화, 클럭 설정, 디버깅 옵션 등을 제어
  - 시스템과 관련된 주요 설정들을 관리하여 MCU의 하드웨어 기능을 구성
- `stm32fxxx_hal_msp.c`: \
  - MSP - MCU Support package
  - 하드웨어 자원과(RCC, GPIO, NVIC and DMA) 연관된 로우 레벨에서의 초기화 함수임 여기에서는 GPIO

- **Drivers/CMSIS/ST/STM32F1xx/include/하위의 stm32f103xb.h 이거 진짜 중요:**
  - CMSIS는 ARM Cortex-M 마이크로컨트롤러에서 하드웨어를 제어하고 다양한 소프트웨어 기능을 사용할 수 있게 하는 표준 인터페이스이며 여기에 Resister들의 물리적 주소가 정의되어있음

  ▼ CMSIS 사용 예시

  stm32f103xb.h 여기에 정의가 되어있는거임 예를들어 CTS는 SR레지스터의 9 번째 << 으로 간 비트고 Msk는 마스킹하는거

  #define USART_SR_CTS_Pos (9U)

  #define USART_SR_CTS_Msk (0x1UL << USART_SR_CTS_Pos) /*!< 0x00000200 */

  #define USART_SR_CTS USART_SR_CTS_Msk /*!< CTS Flag */

  여기서 SR_CTS는 SR레지스터 전체에서 USART_SR_CTS 만 뽑아낼때 쓰는 경우로 USART_SR_CTS_Msk와 동일 값을 가짐 소름 그니까 어차피 접근할때 저거쓴다는거아냐 와 소름이다 예를들어 CTS는 9번째 << 비트니까 거기 1들어가있는 32비트짜리 값이고 이걸 SR 레지스터랑 & 때리면 SR_CTS만 나오는거지. 그리고 #define USART2_BASE (APB1PERIPH_BASE + 0x00004400UL) 같이 USART에 대한 BASE값도 정의가 되어있음 그리고 이걸 USART2라는 변수에 할당하는거임 #define USART2 ((USART_TypeDef *)USART2_BASE) 시작 주소랑 구조체 전체 크기만큼을

  이걸 Main.c의 Init부분 맨위 huart2.Instance = USART2; 여기서 가져오는거임 USART2 는 USART2 의 HW 주소

  이거를 이제 **stm32f1xx_hal_uart.h** 에서 아래와 같이 사용하는거임.

  #define UART_FLAG_CTS ((uint32_t)USART_SR_CTS)
  - **또한 UART의 대한 구조체 정의도 여기서 하고있음 (핸들러말고)**

  ▼ 내용

typedef struct

{

__IO uint32_t SR; /*!< USART Status register, Address offset: 0x00 */

__IO uint32_t DR; /*!< USART Data register, Address offset: 0x04 */

__IO uint32_t BRR; /*!< USART Baud rate register, Address offset: 0x08 */

__IO uint32_t CR1; /*!< USART Control register 1, Address offset: 0x0C */

__IO uint32_t CR2; /*!< USART Control register 2, Address offset: 0x10 */

__IO uint32_t CR3; /*!< USART Control register 3, Address offset: 0x14 */

__IO uint32_t GTPR; /*!< USART Guard time and prescaler register, Address offset: 0x18 */

} USART_TypeDef;

- `stm32f1xx_it.c,stm32f1xx_it.h` :
  - STMicroelectronics의 STM32 시리즈 마이크로컨트롤러에서 인터럽트 처리를 위한 소스 및 헤더 파일
  - HW적으로 UART 인터럽트가 발생하면 stm32f1xx_it.c의 **void USART2_IRQHandler(void)**가 **HAL_UART_IRQHandler를** 호출하고 실행되는거임.**(CUBE MX 에서 global interrupt enable 하면 있고 안하면 없음)** 이거의 정의는 `stm32f1xx_hal_uart.c` **에 나와있음 이거 중요한 함수니까 인터럽트할때 딥하게 파봐야함.**

- `stm32f1xx_hal_def.h`:
  - HAL 드라이버 파일들이 공통으로 사용하는 중요한 데이터 타입, 매크로, 상수 등을 정의

- `stm32fxxx_hal.c,stm32fxxx_hal.h`:
  - HAL 라이브러리 초기화 - HAL_Init() 정의되어있음.
  - 시스템 시간 관리 like Delay함수 정의되어있음
  - 에러 핸들링과 파워 모드 관리, 클럭 설정, NVIC 설정 등 마이크로컨트롤러의 기본적인 시스템 관리 기능

- `stm32fxxx_hal_uart.c,stm32fxxx_hal_uart.h` : 제일 중요한 소스코드 → 나머지 다여기에 있다고 보면 됨.

  ▼ **void USART2_IRQHandler(void)**

  ▼ HAL_UART_Init(&huart) , HAL_UART_DeInit(&huart);

  ▼ HAL_UART_Transmit(&huart, pData, Size, Timeout);

  ▼ HAL_UART_Receive(&huart, pData, Size, Timeout);

  ▼ void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart);

  ▼ void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);

  Below the list of most used macros in UART HAL driver.

  ▼ __HAL_UART_ENABLE: Enable the UART peripheral

  ▼ __HAL_UART_DISABLE: Disable the UART peripheral

  ▼ __HAL_UART_GET_FLAG : Check whether the specified UART flag is set or not

  ▼ __HAL_UART_CLEAR_FLAG : Clear the specified UART pending flag

  ▼ __HAL_UART_ENABLE_IT: Enable the specified UART interrupt

  ▼ __HAL_UART_DISABLE_IT: Disable the specified UART interrupt

  ▼ __HAL_UART_GET_IT_SOURCE: Check whether the specified UART interrupt has occurred or not

▼ (2) - **HAL_Init**(); >>>> **HAL_StatusTypeDef HAL_Init(void) in stm32f1xx_hal.c**

```
HAL_StatusTypeDef HAL_Init(void)
{
  /* Configure Flash prefetch */
#if (PREFETCH_ENABLE != 0)
// 조건부 컴파일: PREFETCH_ENABLE이 0이 아니면, 즉 프리페치 기능이 활성화되었으면 아래의 코드를 실행
#if defined(STM32F101x6) || defined(STM32F101xB) || defined(STM32F101xE) || defined(ST
    defined(STM32F102x6) || defined(STM32F102xB) || \
    defined(STM32F103x6) || defined(STM32F103xB) || defined(STM32F103xE) || defined(ST
    defined(STM32F105xC) || defined(STM32F107xC)
//조건부 컴파일: 특정 STM32F1 시리즈에서만 프리페치 기능이 사용 가능하도록 하여, 해당 조건에 맞는 마이크
  /* Prefetch buffer is not available on value line devices */
  __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
//함수 호출: 플래시 프리페치 버퍼를 활성화하는 매크로입니다. 플래시 메모리에서 데이터를 읽기 전에 CPU가
#endif
#endif /* PREFETCH_ENABLE */
//그니까 여기 위에까지 프리패치 기능이 있다면 그걸 써라 하는거고

  /* Set Interrupt Group Priority */
  HAL_NVIC_SetPriorityGrouping(NVIC_PRIORITYGROUP_4);
    // 밑에 함수 참조
    //VIC_PRIORITYGROUP_4는 4비트로 그룹 우선순위를 정의하며, 그룹별로 서브 우선순위 설정이 가능합니
  /* Use systick as time base source and configure 1ms tick (default clock after Reset
  HAL_InitTick(TICK_INT_PRIORITY);
    // 밑에 함수 참조
    //시스템 틱 타이머를 초기화합니다. 이 함수는 SysTick 인터럽트를 1밀리초 주기로 설정하여 시간 기반
    // 이 함수는 Systick 타이머를 1ms 단위로 설정하고, 해당 타이머 인터럽트의 우선순위를 설정하는 기능



  /* Init the low level hardware */
  HAL_MspInit();
    // RCC Setting
  /* Return function status */
  return HAL_OK;
}
```

▼ HAL_StatusTypeDef in **stm32f1xx_hal_def.h**

```
typedef enum
{
  HAL_OK       = 0x00U,
  HAL_ERROR    = 0x01U,
  HAL_BUSY     = 0x02U,
  HAL_TIMEOUT  = 0x03U
} HAL_StatusTypeDef;
```

▼ 중요함수 - void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup) **in stm32f1xx_hal_cortex.c**

여기서 NVIC_PRIORITYGROUP_4는 **stm32f1xx_hal_cortex.h에 정의되어있음**

▼ NVIC_PRIORITYGROUP_4 in **stm32f1xx_hal_cortex.h에 정의되어있음**

```
#define NVIC_PRIORITYGROUP_0         0x00000007U /*!< 0 bits for pre-emption pri
                                                  4 bits for subpriority */
#define NVIC_PRIORITYGROUP_1         0x00000006U /*!< 1 bits for pre-emption pri
```

```
                                                    3 bits for subpriority */
#define NVIC_PRIORITYGROUP_2         0x00000005U /*!< 2 bits for pre-emption pri
                                                    2 bits for subpriority */
#define NVIC_PRIORITYGROUP_3         0x00000004U /*!< 3 bits for pre-emption pri
                                                    1 bits for subpriority */
#define NVIC_PRIORITYGROUP_4         0x00000003U /*!< 4 bits for pre-emption pri
                                                    0 bits for subpriority */
```

```
void HAL_NVIC_SetPriorityGrouping(uint32_t PriorityGroup)
{
  /* Check the parameters */
  assert_param(IS_NVIC_PRIORITY_GROUP(PriorityGroup));
  // 여기서 유효성 확인.
  /* Set the PRIGROUP[10:8] bits according to the PriorityGroup parameter value */
  NVIC_SetPriorityGrouping(PriorityGroup);
  // 이거는  ARM Cortex-M3 코어의 SCB (System Control Block) 레지스터 중 하나인 AIRCR (Ap
  // 에서 PRIGROUP[10:8] 비트를 uint32_t PriorityGroup 값에 따라서 설정하는거 (4 bits for
  // NVIC_PRIORITYGROUP_4는 4비트로 그룹 우선순위를 정의하며, 그룹별로 서브 우선순위 설정이 가능
}
```

▼ NVIC_SetPriorityGrouping

▼ 중요함수 - __weak **HAL_StatusTypeDef HAL_InitTick(uint32_t** TickPriority) in **stm32f1xx_hal.c** // 그리고 여기 쓰인 **TICK_INT_PRIORITY** 는 **stm32f1xx_hal_conf.h**에 정의되어 있음 - **#define TICK_INT_PRIORITY** 15U /*!< tick interrupt priority (lowest by default) */

```
__weak HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
{
  /* Configure the SysTick to have interrupt in 1ms time basis*/
  if (HAL_SYSTICK_Config(SystemCoreClock / (1000U / uwTickFreq)) > 0U)
  {
    return HAL_ERROR;
  }

  /* Configure the SysTick IRQ priority */
  if (TickPriority < (1UL << __NVIC_PRIO_BITS))
  {
    HAL_NVIC_SetPriority(SysTick_IRQn, TickPriority, 0U);
    uwTickPrio = TickPriority;
  }
  else
  {
    return HAL_ERROR;
  }

  /* Return function status */
  return HAL_OK;
}
```

▼ 중요함수 - **void HAL_MspInit(void) in stm32f1xx_hal_msp.c -** `HAL_MspInit` 함수는 STM32의 초기화 과정에서 전원 관리와 핀 기능 설정을 포함한 기본 하드웨어 초기화 작업을 수행

```
void HAL_MspInit(void)
{
```

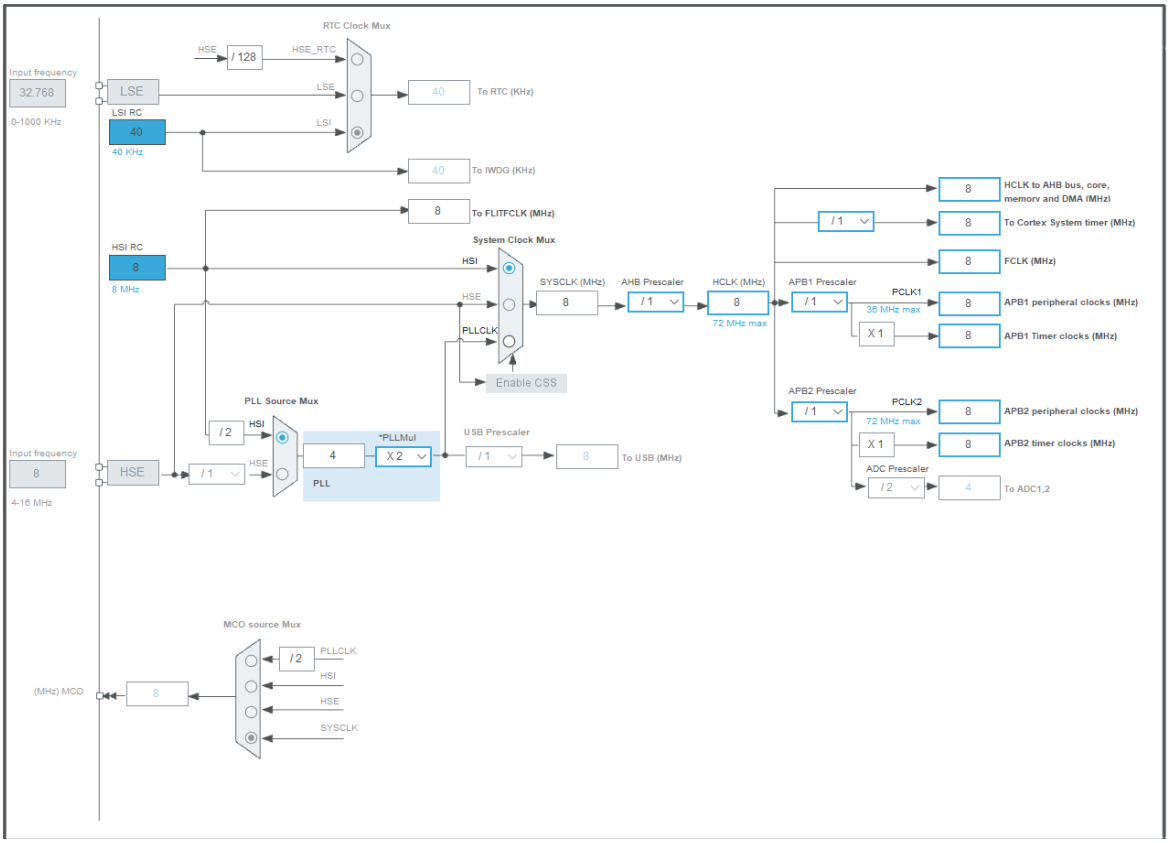
```
  /* USER CODE BEGIN MspInit 0 */

  /* USER CODE END MspInit 0 */

  __HAL_RCC_AFIO_CLK_ENABLE();
  //__HAL_RCC_AFIO_CLK_ENABLE(): AFIO (Alternate Function I/O) 클럭을 활성화합니다. AFI
  __HAL_RCC_PWR_CLK_ENABLE();
  //__HAL_RCC_PWR_CLK_ENABLE(): 전원 관리(PWR) 클럭을 활성화합니다. 이 함수는 STM32의 전원 관

  /* System interrupt init*/

  /** DISABLE: JTAG-DP Disabled and SW-DP Disabled
  */
  __HAL_AFIO_REMAP_SWJ_DISABLE();
  // __HAL_AFIO_REMAP_SWJ_DISABLE(): JTAG-DP와 SW-DP를 비활성화합니다. 이 함수를 호출하면
  /* USER CODE BEGIN MspInit 1 */

  /* USER CODE END MspInit 1 */
}
```

▼ (3) - **SystemClock_Config**() >>>> **void SystemClock_Config(void) in main.c**

▼ RCC 랑 CLOCK 간단 설명

https://velog.io/@pikamon/STM32-2

여기에 있는 RCC 설정을 아래와 같이 setting해주는거임



```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  // oscillator 설정 부분
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
   // RCC 설정 부분
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}
```

▼ (4) - **MX_GPIO_Init**(); >>> GPIOA에 CLK ENABLE >> HSI CLK (APBx CLK (PCLK))가 있어야 보레이트 설정할거아니

```
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
```

▼ (1),(5) - UART_HandleTypeDef huart2 , **MX_USART2_UART_Init**();

위의 (1),(2) 는 우리가 사용할 UART의 구조체를 사용하기 편하게 만든 핸들러 구조체를 선언하고 초기 정의 하는 부분임.

UART_HandleTypeDef 구조체는 아래와 같은 구조를 가지고 있으며 HAL 드라이버 하위 stm32f1xx_hal_uart.h에 정의되어 있음.

특히 if 하위의 부분 void(*~~~~) 부분은 call back 함수 enable 하는 부분으로 보임.

▼ 구조체 구조 - stm32f1xx_hal_uart.h

```
**typedef** **struct** **__UART_HandleTypeDef**

{

    **USART_TypeDef**                *Instance;        /*!< UART registers base ad

    **UART_InitTypeDef**             Init;             /*!< UART communication par

    **const** **uint8_t**                *pTxBuffPtr;      /*!< Pointer to UART Tx

    **uint16_t**                     TxXferSize;       /*!< UART Tx Transfer size

    __IO **uint16_t**                TxXferCount;      /*!< UART Tx Transfer Count

    **uint8_t**                      *pRxBuffPtr;      /*!< Pointer to UART Rx tra

    **uint16_t**                     RxXferSize;       /*!< UART Rx Transfer size

    __IO **uint16_t**                RxXferCount;      /*!< UART Rx Transfer Count

    __IO **HAL_UART_RxTypeTypeDef** ReceptionType;      /*!< Type of ongoing recept

    __IO **HAL_UART_RxEventTypeTypeDef** RxEventType;   /*!< Type of Rx Event

    **DMA_HandleTypeDef**            *hdmatx;          /*!< UART Tx DMA Handle par

    **DMA_HandleTypeDef**            *hdmarx;          /*!< UART Rx DMA Handle par

    **HAL_LockTypeDef**              Lock;             /*!< Locking object

    __IO **HAL_UART_StateTypeDef**    gState;           /*!< UART state information

    and also related to Tx operations.

    This parameter can be a value of @ref HAL_UART_StateTypeDef */

    __IO **HAL_UART_StateTypeDef**    RxState;          /*!< UART state information

    This parameter can be a value of @ref HAL_UART_StateTypeDef */

    __IO **uint32_t**                ErrorCode;        /*!< UART Error code

    **#if** (USE_HAL_UART_REGISTER_CALLBACKS == 1)

    **void** (* TxHalfCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* TxCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* RxHalfCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* RxCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* ErrorCallback)(**struct** __UART_HandleTypeDef *huart);
```

```
    **void** (* AbortCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* AbortTransmitCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* AbortReceiveCpltCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* WakeupCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* RxEventCallback)(**struct** __UART_HandleTypeDef *huart, uint16_t F

    **void** (* MspInitCallback)(**struct** __UART_HandleTypeDef *huart);

    **void** (* MspDeInitCallback)(**struct** __UART_HandleTypeDef *huart);

    **#endif**  /* USE_HAL_UART_REGISTER_CALLBACKS */

} **UART_HandleTypeDef**;
```

▼ Init 부분 - Main.c

static void MX_USART2_UART_Init(void)

{

(1-1) huart2.Instance = USART2;

▼ (1-1) huart2.Instance = USART2;

>> **USART2는 stm32f103xb.h에 정의되어 있는 주소로 주소값은 USART HW 위치값임 , 이 주소가 가르키는 값은 동일 소스에 정의된 USART 구조체임[SR,CR... ETC] 그니까 USART2→SR 하면 SR 레지스터 직접 접근인거임.**

보다 자세한 설명은 일단 여기서 USART2 주소값을 받아오는데 CMISIS(**stm32f103xb.h**)에서 **#define USART2 ((USART_TypeDef *)USART2_BASE)** 라고 정의가 되어있으며 이 USART_TypeDef는 아래와 같이 정의가 되어 있음 그러므로 아래 **구조체만큼의 공간을 가진 구조체를 USART2_BASE 즉 HW 주소를 시작점으로 할당**해주는거임.
#define USART2_BASE (APB1PERIPH_BASE + 0x00004400UL)

******* 아래 USART2 구조 (핸들러 구조 아님 주의할 것)**

typedef struct

{

__IO uint32_t SR; /*!< USART Status register, Address offset: 0x00 */

__IO uint32_t DR; /*!< USART Data register, Address offset: 0x04 */

__IO uint32_t BRR; /*!< USART Baud rate register, Address offset: 0x08 */

__IO uint32_t CR1; /*!< USART Control register 1, Address offset: 0x0C */

__IO uint32_t CR2; /*!< USART Control register 2, Address offset: 0x10 */

__IO uint32_t CR3; /*!< USART Control register 3, Address offset: 0x14 */

__IO uint32_t GTPR; /*!< USART Guard time and prescaler register, Address offset: 0x18 */

} USART_TypeDef;

여기 이후 부분은 huart2 핸들러의 의 init 구조체의 값을 넣어주는 부분으로 init 구조체를 먼저 알아 볼 필요가 있음.

▼ **UART_InitTypeDef**

typedef struct

{

uint32_t BaudRate; /*!< This member configures the UART communication baud rate.

The baud rate is computed using the following formula:

- IntegerDivider = ((PCLKx) / (16 * (huart→Init.BaudRate)))

- FractionalDivider = ((IntegerDivider - ((uint32_t) IntegerDivider)) * 16) + 0.5 */

uint32_t WordLength; /*!< Specifies the number of data bits transmitted or received in a frame.

This parameter can be a value of @ref UART_Word_Length */

uint32_t StopBits; /*!< Specifies the number of stop bits transmitted.

This parameter can be a value of @ref UART_Stop_Bits */

uint32_t Parity; /*!< Specifies the parity mode.

This parameter can be a value of @ref UART_Parity

@note When parity is enabled, the computed parity is inserted

at the MSB position of the transmitted data (9th bit when

the word length is set to 9 data bits; 8th bit when the

word length is set to 8 data bits). */

uint32_t Mode; /*!< Specifies whether the Receive or Transmit mode is enabled or disabled.

This parameter can be a value of @ref UART_Mode */

uint32_t HwFlowCtl; /*!< Specifies whether the hardware flow control mode is enabled or disabled.

This parameter can be a value of @ref UART_Hardware_Flow_Control */

uint32_t OverSampling; /*!< Specifies whether the Over sampling 8 is enabled or disabled, to achieve higher speed (up to fPCLK/8).

This parameter can be a value of @ref UART_Over_Sampling. This feature is only available

on STM32F100xx family, so OverSampling parameter should always be set to 16. */

} **UART_InitTypeDef**;

(1-2) huart2.Init.BaudRate = 9600;

→ Baudrate 설정

(1-3)huart2.Init.WordLength = UART_WORDLENGTH_8B;

→ Data bits 설정 8 비트

(1-4) huart2.Init.StopBits = UART_STOPBITS_1;

→ 스톱 비츠 1비트

(1-5) huart2.Init.Parity = UART_PARITY_NONE;

→ 패리티 None

(1-6) huart2.Init.Mode = UART_MODE_TX_RX;

→ init.Mode → UART 통신을 **양방향(송신 및 수신)** 모드로 설정 → 다른 옵션으로 송신만하거나 수신만 하거나 비활성화 할 수 있음 → 아래 HAL_UART_Init에서 레지스터 설정 들어갈 거임 이 모드에 따라서

(1-7) huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;

→ HWCONTROL은 DATA손실 방지를 위해(송신/수신측이 서로 약속하는거지) RTC와 CTS핀을 사용해서 (블록다이어그램 참조) 해당 pin이 low일때만(준비가 되어있을때) 송신/수신 하는 것
이거 NONE이라는거는 UART 하드웨어 흐름 제어(Hardware Flow Control)**를 사용하지 않겠다는 설정입니다. 즉,
하드웨어 레벨에서 송수신 데이터를 제어하는 추가적인 제어 핀(RTS, CTS)을 사용하지 않겠다는 뜻

(1-8) huart2.Init.OverSampling = UART_OVERSAMPLING_16;

→ UART 통신에서 **오버샘플링 비율**을 정의 → 오버샘플링이란 수신하는 데이터 비트의 정확성을 높이기 위해 신호를 더 자주 샘플링하는 과정을 말함

(높으면 왜곡이 줄어서 신뢰성이 향상됨) → 너무 높으면 여기에 드는 리소스로 인한 성능저하

if (**(1-9) HAL_UART_Init**(&huart2) != *HAL_OK*)

▼ **(1-9) HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef** *huart) in `stm32fxxx_hal_uart.c`

이건 기본적으로 위의 init.xxxx 에서 설정한 값들로 실제 레지스터에 값을 때려박는 부분임 **매우중요**

HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)

{

/* Check the UART handle allocation */

if (huart == NULL)

{

return *HAL_ERROR*;

}

/* Check the parameters */

parameter 유효성확인단계

if (huart->Init.HwFlowCtl != UART_HWCONTROL_NONE)

{

여기서 이제 HWCONTROL_NONE 그러니까 RTS, CTS 안쓰겠다고 체크가 되어있다면

/* The hardware flow control is available only for USART1, USART2 and USART3 */

assert_param(IS_UART_HWFLOW_INSTANCE(huart->Instance));

// 일단 assert_param은 아래와 같고 이게 유효한 값(True)라면 진행이 되고 아니면 디버깅 과정에서 Fail 됨.

```
#define assert_param(expr) ((expr) ? (void)0U : assert_failed((uint8_t *)__FI
LE__, __LINE__))
```

assert_param(IS_UART_HARDWARE_FLOW_CONTROL(huart->Init.HwFlowCtl));

// 동일 절차

}

else

{

assert_param(IS_UART_INSTANCE(huart->Instance));

// 동일 절차 단 instance만 HWCONTROL이 NONE이니까

}

assert_param(IS_UART_WORD_LENGTH(huart->Init.WordLength));

// 동일 절차 Init.WordLength 유효검사

#if defined(USART_CR1_OVER8)

assert_param(IS_UART_OVERSAMPLING(huart->Init.OverSampling));

//동일절차 근데 이 USART_CR1_OVER8은 어디에 정의되어있는거지?? 찾아보니까 이건 정의가 안되어있음 걍 없는 애인것같음 다른데서 쓰는거일 듯.

// 원래였으면 CMSIS에 있어야하는데 없음 그리고 애초에 16이 걍 0임 여기서 지원 안되는 부분이거같음 #define UART_OVERSAMPLING_16 0x00000000U

#endif /* USART_CR1_OVER8 */

if (huart→gState == *HAL_UART_STATE_RESET*)

▼ **huart→gState 는 HAL_UART_StateTypeDef 구조체로서 아래와 같이 열거 정의가 되어있음 in `stm32fxxx_hal_uart.h`. 예를들어 0 이면 *HAL_UART_STATE_RESET* 이고 *HAL_UART_STATE_RESET* 이면 0이라는거지 아래 정의 참조 근데 c언어 구조체에서는**
구조체가 전역 변수일 경우 모든 필드가 0으로 초기화되므로 기본적으로 HAL_UART_Init 함수 이전에는 huart->gState ==
HAL_UART_STATE_RESET 라는거임 그러니까 결국 초기화가 안되어있으면 아래 코드들 실행. << 이게 필요한 이유는 이제 완전 Low level 하드웨어 수정할거라서 LOCK 풀고 변경해야함. 안정성을 위해

typedef enum

{

HAL_UART_STATE_RESET = 0x00U, /*!< Peripheral is not yet Initialized

Value is allowed for gState and RxState */

HAL_UART_STATE_READY = 0x20U, /*!< Peripheral Initialized and ready for use

Value is allowed for gState and RxState */

HAL_UART_STATE_BUSY = 0x24U, /*!< an internal process is ongoing

Value is allowed for gState only */

HAL_UART_STATE_BUSY_TX = 0x21U, /*!< Data Transmission process is ongoing

Value is allowed for gState only */

HAL_UART_STATE_BUSY_RX = 0x22U, /*!< Data Reception process is ongoing

Value is allowed for RxState only */

HAL_UART_STATE_BUSY_TX_RX = 0x23U, /*!< Data Transmission and Reception process is ongoing

Not to be used for neither gState nor RxState.

Value is result of combination (Or) between gState and RxState values */

HAL_UART_STATE_TIMEOUT = 0xA0U, /*!< Timeout state

Value is allowed for gState only */

HAL_UART_STATE_ERROR = 0xE0U /*!< Error

Value is allowed for gState only */

} **HAL_UART_StateTypeDef**;

{

/* Allocate lock resource and initialize it */

huart→Lock = *HAL_UNLOCKED*;

// Lock 제거

// 파란배경 if

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)

▼ **USE_HAL_UART_REGISTER_CALLBACKS의 의미 >> 어쨋든 우리는 재정의해서 쓰니까 0으로 셋팅되어 있음.**

//USE_HAL_UART_REGISTER_CALLBACKS는 기본적으로 **stm32f1xx_hal_conf.h**에 0으로 정의되어있고 사용하려면 1로 바꿔줘야함

// 이게 의미하는거는 그냥 callback 함수들을 handler (huart)에 등록해서 쓸지말지임

아래처럼 **void HAL_UART_IRQHandler(UART_HandleTypeDef** *huart) 일부를 살펴보면 USE_HAL_UART_REGISTER_CALLBACKS == 1 때는 핸들러에 있는 콜백을 부르고 0일때는 weak 정의되어있는 기본 콜백을 부르는데 이게 0이면 weak로 정의되어있는 기본 콜백함수를 내가 메인에서 재정의 해서

쓰는거고 1일때는 `HAL_UART_RegisterCallback()` 함수로 `My_TxCpltCallback`이라는 사용자 정의 콜백을 등록하면, 이제 UART 전송 완료 시 HAL에서 제공하는 `HAL_UART_TxCpltCallback()` 대신 `My_TxCpltCallback()`이 호출됩니다.

▼ **void HAL_UART_IRQHandler(UART_HandleTypeDef** *huart) 일부 발췌 callback 부르는 부분

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)

/*Call registered Rx Event callback*/

huart→RxEventCallback(huart, (huart→RxXferSize - huart→RxXferCount));

#else

/*Call legacy weak Rx Event callback*/

HAL_UARTEx_RxEventCallback(huart, (huart→RxXferSize - huart→RxXferCount));

▼ **추가 설명 Detail by GPT**

`USE_HAL_UART_REGISTER_CALLBACKS`의 동작 방식을 이해하기 쉽게 예시를 통해 비교해 보겠습니다. 두 가지 상황을 다뤄볼 텐데, 하나는 `USE_HAL_UART_REGISTER_CALLBACKS == 0`일 때 기본적인 콜백 동작, 그리고 다른 하나는 `USE_HAL_UART_REGISTER_CALLBACKS == 1`일 때 사용자 정의 콜백을 사용하는 예입니다.

1. `USE_HAL_UART_REGISTER_CALLBACKS == 0` 일 때

이 경우는 기본적으로 HAL이 제공하는 **약한 정의(`__weak`)**의 콜백 함수들이 사용됩니다. 사용자가 별도의 설정이나 등록을 할 필요가 없으며, HAL 내부에서 자동으로 콜백 함수가 호출됩니다.

예시:

- UART 전송이 완료되었을 때 호출되는 콜백 함수 **`HAL_UART_TxCpltCallback`*을 사용해보겠습니다.

```
c
코드 복사
// HAL에서 제공하는 기본 콜백 함수 (약한 정의)
__weak void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    // 기본적으로는 아무 동작도 하지 않음
}
```

이 함수는 `__weak`로 선언되어 있기 때문에, 사용자가 별도의 정의를 하지 않으면 HAL에서 제공하는 기본 함수가 호출됩니다. 즉, **아무 작업도 하지 않음**으로 설정되어 있습니다.

이제 UART 전송이 완료되면, 인터럽트 핸들러는 자동으로 `HAL_UART_TxCpltCallback()`을 호출합니다:

```
c
코드 복사
void USART2_IRQHandler(void)
{
    // UART 인터럽트 발생 시 HAL의 핸들러를 호출
    HAL_UART_IRQHandler(&huart2);
}

// HAL의 내부에서 전송 완료 인터럽트가 발생하면 다음 콜백 호출
HAL_UART_TxCpltCallback(&huart2);  // 기본적으로는 약한 정의된 함수가 호출됨
```

이때 `USE_HAL_UART_REGISTER_CALLBACKS == 0`이므로, `HAL_UART_TxCpltCallback()`은 자동으로 호출되지만 아무 동작도 하지 않습니다. 만약 사용자가 이 기본 콜백을 변경하고 싶다면, 단순히 이 함수의 정의를 재정의하면 됩니다:

```
c
코드 복사
// 사용자가 원하는 방식으로 콜백 재정의
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
    printf("UART 전송 완료!\n");  // 전송 완료 시 콘솔에 메시지를 출력
}
```

이처럼 `USE_HAL_UART_REGISTER_CALLBACKS == 0`일 때는 HAL이 제공하는 기본 콜백 함수만 호출되고, 사용자는 `__weak`로 정의된 콜백을 **재정의**하는 방식으로 원하는 동작을 구현합니다.

---

2. `USE_HAL_UART_REGISTER_CALLBACKS == 1`일 때

이 경우는 사용자가 HAL에서 제공하는 기본 콜백 함수 외에도 자신만의 콜백 함수를 직접 **등록**할 수 있습니다. HAL은 사용자 정의 콜백을 등록할 수 있도록 **유연한 인터페이스**를 제공합니다.

예시:

`USE_HAL_UART_REGISTER_CALLBACKS == 1`로 설정된 경우, HAL에서 제공하는 `UART_InitCallbacksToDefault()` 함수는 기본 콜백 함수를 초기화합니다.

```
c
코드 복사
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
void UART_InitCallbacksToDefault(UART_HandleTypeDef *huart)
{
    huart->TxCpltCallback = HAL_UART_TxCpltCallback;  // 기본 콜백 설정
}
#endif
```

이 함수는 기본적으로 HAL의 약한 정의 콜백을 `huart` 핸들에 설정해 줍니다. 하지만 사용자는 나중에 `HAL_UART_RegisterCallback()` 함수를 이용해 **자신만의 콜백 함수**를 등록할 수 있습니다.

사용자 정의 콜백 등록 예시:

```
c
코드 복사
void My_TxCpltCallback(UART_HandleTypeDef *huart)
{
    printf("사용자 정의 콜백: UART 전송 완료!\n");
}

// 콜백 등록
HAL_UART_RegisterCallback(&huart2, HAL_UART_TX_COMPLETE_CB_ID, My_TxC
pltCallback);
```

`HAL_UART_RegisterCallback()` 함수로 `My_TxCpltCallback`이라는 사용자 정의 콜백을 등록하면, 이제 UART 전송 완료 시 HAL에서 제공하는 `HAL_UART_TxCpltCallback()` 대신 `My_TxCpltCallback()`이 호출됩니다.

동작 과정:

```
c
코드 복사
void USART2_IRQHandler(void)
```

```
{
    // UART 인터럽트 발생 시 HAL 핸들러 호출
    HAL_UART_IRQHandler(&huart2);
}

// HAL 내부에서 전송 완료 시 등록된 콜백 호출
if (huart2.TxCpltCallback != NULL)
{
    huart2.TxCpltCallback(&huart2);  // 사용자 정의 콜백이 호출됨
}
```

이 과정에서, **기본 콜백 함수 대신 사용자 정의 콜백 함수가 호출**됩니다. 이로써 사용자는 원하는 동작을 좀 더 유연하게 구현할 수 있습니다.

요약 비교

설정	콜백 방식	등록 방법	기본 콜백 호출 방식
`USE_HAL_UART_REGISTER_CALLBACKS == 0`	**기본 콜백** (`__weak`)	없음 (재정의 필요)	HAL이 제공하는 기본 콜백 함수가 자동 호출
`USE_HAL_UART_REGISTER_CALLBACKS == 1`	**사용자 정의 콜백** 가능	`HAL_UART_RegisterCallback()`으로 등록	사용자 정의 콜백을 등록한 경우 등록된 콜백이 호출

정리

- `USE_HAL_UART_REGISTER_CALLBACKS == 0`: 기본 제공 콜백 함수(`__weak`)가 자동으로 호출되며, 콜백을 수정하려면 해당 함수의 정의를 재정의해야 합니다.
- `USE_HAL_UART_REGISTER_CALLBACKS == 1`: 기본 콜백을 사용하면서도, 필요 시 `HAL_UART_RegisterCallback()`을 사용해 자신만의 콜백을 등록하고 호출할 수 있는 유연성을 제공합니다.

UART_InitCallbacksToDefault(huart);

▼ **UART_InitCallbacksToDefault(huart) >> 얘는 핸들러에 콜백들 추가시키는 부분임**

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)

void UART_InitCallbacksToDefault(UART_HandleTypeDef *huart)

{

핸들러에 각각의 callback함수들을 업로드

/* Init the UART Callback settings */

huart->TxHalfCpltCallback = HAL_UART_TxHalfCpltCallback; /* Legacy weak TxHalfCpltCallback */

huart->TxCpltCallback = HAL_UART_TxCpltCallback; /* Legacy weak TxCpltCallback */

huart->RxHalfCpltCallback = HAL_UART_RxHalfCpltCallback; /* Legacy weak RxHalfCpltCallback */

huart->RxCpltCallback = HAL_UART_RxCpltCallback; /* Legacy weak RxCpltCallback */

huart->ErrorCallback = HAL_UART_ErrorCallback; /* Legacy weak ErrorCallback */

```
        huart->AbortCpltCallback        = HAL_UART_AbortCpltCallback;        /* Legacy weak AbortCpltCallback        */
        huart->AbortTransmitCpltCallback = HAL_UART_AbortTransmitCpltCallback; /* Legacy weak AbortTransmitCpltCallback */
        huart->AbortReceiveCpltCallback  = HAL_UART_AbortReceiveCpltCallback;  /* Legacy weak AbortReceiveCpltCallback  */
        huart->RxEventCallback          = HAL_UARTEx_RxEventCallback;        /* Legacy weak RxEventCallback          */
        }
        #endif /* USE_HAL_UART_REGISTER_CALLBACKS */
if (huart->MspInitCallback == NULL)
{     //여기서도 만약에 USE_HAL_UART_REGISTER_CALLBACKS == 1 이면 콜백 추가해줘야하니까 넣은 부분임 신경 안써도될듯 밑에 빨간색 보샘
    huart->MspInitCallback = HAL_UART_MspInit;
}
/* Init the low level hardware */
huart->MspInitCallback(huart);

#else
/* Init the low level hardware : GPIO, CLOCK */
HAL_UART_MspInit(huart);
```

▼ 여기서 **HAL_UART_MspInit**(huart); 는 stm32f1xx_hal_msp.c 에 재정의 되어있음 GPIO랑 CLOCK init 해줌. CUBE setting에 따라서

```
    void HAL_UART_MspInit(UART_HandleTypeDef* huart)
    {
        GPIO_InitTypeDef GPIO_InitStruct = {0};
        if(huart->Instance==USART2)
        {
            /* USER CODE BEGIN USART2_MspInit 0 */
            /* USER CODE END USART2_MspInit 0 */
            /* Peripheral clock enable */
            __HAL_RCC_USART2_CLK_ENABLE();
            __HAL_RCC_GPIOA_CLK_ENABLE();
            /**USART2 GPIO Configuration
            PA2     ------> USART2_TX
            PA3     ------> USART2_RX
            */
            GPIO_InitStruct.Pin = GPIO_PIN_2;
            GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
            GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
            HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
            GPIO_InitStruct.Pin = GPIO_PIN_3;
            GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
```

```
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
        /* USER CODE BEGIN USART2_MspInit 1 */
        /* USER CODE END USART2_MspInit 1 */
      }
    }
  #endif /* (USE_HAL_UART_REGISTER_CALLBACKS) */
  }
```

// 이제 UART 레지스터 변경 들어갈꺼니까 State_busy로 변경해놓는다

huart→gState = *HAL_UART_STATE_BUSY*;

/* Disable the peripheral */

// UART Disable → 레지스터 변경할꺼니까

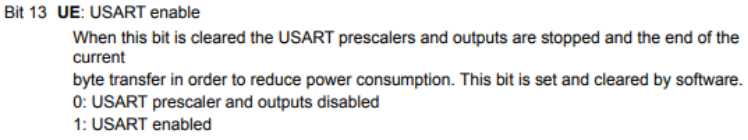
__HAL_UART_DISABLE(huart);

▼ **#define __HAL_UART_DISABLE(__HANDLE__) 함수 정의 내용**

함수와 정의된 Define값들

```
#define __HAL_UART_DISABLE(__HANDLE__)              ((__HANDLE__)->Instance->
// 아래 IT랑 헷갈리면 안됨.
#define __HAL_UART_DISABLE_IT(__HANDLE__, __INTERRUPT__)  ((((__INTERRUPT__)
                                                           (((__INTERRUPT__)
                                                           ((__HANDLE__)->Ins
```

여기서 CR1 전체랑 ~USART_CR1_UE(13번째로 정의) 랑 &= 하는거니까 UE에 0(CLEAR) 해서 UART 중지하는 거임.


Bit 13 **UE**: USART enable
When this bit is cleared the USART prescalers and outputs are stopped and the end of the current
byte transfer in order to reduce power consumption. This bit is set and cleared by software.
0: USART prescaler and outputs disabled
1: USART enabled

/* Set the UART Communication parameters */

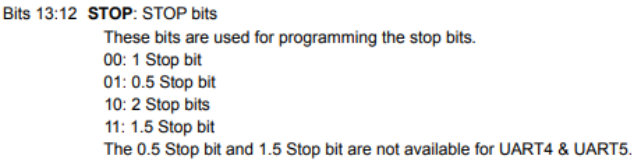
UART_SetConfig(huart);

▼ **UART_SetConfig(huart) → 실질적인 Resister setting하는 부분.**

```
static void UART_SetConfig(UART_HandleTypeDef *huart)
{
  uint32_t tmpreg;
  uint32_t pclk;

  /* Check the parameters */
  assert_param(IS_UART_BAUDRATE(huart->Init.BaudRate));
  assert_param(IS_UART_STOPBITS(huart->Init.StopBits));
  assert_param(IS_UART_PARITY(huart->Init.Parity));
  assert_param(IS_UART_MODE(huart->Init.Mode));
    //이거 했었지??? 유효성 검사하는 부분.

  /*-------------------------- USART CR2 Configuration -----------------------
  /* Configure the UART Stop Bits: Set STOP[13:12] bits
```

```
     according to huart->Init.StopBits value */
  MODIFY_REG(huart->Instance->CR2, USART_CR2_STOP, huart->Init.StopBits);
    // MODIFY_REG관련 아래 함수 설명 참조 stop 비트 레지스터에 00000들어감 어차피
  /*-------------------------- USART CR1 Configuration ----------------------
  /* Configure the UART Word Length, Parity and mode:
     Set the M bits according to huart->Init.WordLength value
     Set PCE and PS bits according to huart->Init.Parity value
     Set TE and RE bits according to huart->Init.Mode value
     Set OVER8 bit according to huart->Init.OverSampling value */

#if defined(USART_CR1_OVER8)
//이거 CR1_OVER8우리 안쓰니까 아래 else문실행임.
  tmpreg = (uint32_t)huart->Init.WordLength | huart->Init.Parity | huart->Ini
  MODIFY_REG(huart->Instance->CR1,
             (uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR
             tmpreg);
#else
  tmpreg = (uint32_t)huart->Init.WordLength | huart->Init.Parity | huart->Ini
  MODIFY_REG(huart->Instance->CR1,
             (uint32_t)(USART_CR1_M | USART_CR1_PCE | USART_CR1_PS | USART_CR
             tmpreg);
//여기서 이제 각 자리값 정해져있찌? 거기 마스킹 때려서 저기 tmpreg랑 or 때려서 레지스터 다
// 여기까지의 Resister 변화는
// __HAL_UART_DISABLE(huart) 여기서 CR1에 UE 0 넣어놨고 M에 0 들어갔고 (UART_WORDLE
// CR2에 Stop 비트 변경한거 MODIFY_REG(huart->Instance->CR2, USART_CR2_STOP, hua
// 그다음에 UART_WORDLENGTH_8B = 0 / UART_STOPBITS_1 = 0 이거 아까 바로 위에서 했고
// #define UART_MODE_TX_RX  ((uint32_t)(USART_CR1_TE | USART_CR1_RE)) >> 이뜻은
// 결국 여기까지 해서 TE , RE 각각 1들어가는게 끝임. (2번째 3번째 비트)

#endif /* USART_CR1_OVER8 */

  /*-------------------------- USART CR3 Configuration ----------------------
  /* Configure the UART HFC: Set CTSE and RTSE bits according to huart->Init.
  MODIFY_REG(huart->Instance->CR3, (USART_CR3_RTSE | USART_CR3_CTSE), huart->
    // 여기서 이제 CR3의 RTSE 랑 CTSE자리에 0들어감  #define UART_HWCONTROL_NONE

  if(huart->Instance == USART1)
  {
    pclk = HAL_RCC_GetPCLK2Freq();
  }
  else
  {
    pclk = HAL_RCC_GetPCLK1Freq();
    //아래 함수 정리 참조.
  }

  /*-------------------------- USART BRR Configuration ---------------------*
#if defined(USART_CR1_OVER8)
  if (huart->Init.OverSampling == UART_OVERSAMPLING_8)
  {
    huart->Instance->BRR = UART_BRR_SAMPLING8(pclk, huart->Init.BaudRate);
  }
  else
  {
    huart->Instance->BRR = UART_BRR_SAMPLING16(pclk, huart->Init.BaudRate);
```

```
  }
#else
      // 여기 들어가는거지 이제  아래 함수 정리 참조
  huart->Instance->BRR = UART_BRR_SAMPLING16(pclk, huart->Init.BaudRate);
#endif /* USART_CR1_OVER8 */
}
```


Bits 13:12 **STOP**: STOP bits
These bits are used for programming the stop bits.
00: 1 Stop bit
01: 0.5 Stop bit
10: 2 Stop bits
11: 1.5 Stop bit
The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

▼ **MODIFY_REG(REG, CLEARMASK, SETMASK) WRITE_REG((REG), (((READ_REG(REG)) & (~(CLEARMASK))) | (SETMASK))) in stm32f1xx.h**

예시로 만약에 MODIFY_REG(huart→Instance→CR2, USART_CR2_STOP, huart→Init.StopBits); 이렇게 호출이 된다고 한다면 CR2에

0x12345678이 들어있었다면 이걸 read_reg로 읽어와서 ~(CLEARMASK)랑 & 연산을 때리면 그 마스크 부분만 없어져 그니까 12:13 비트만 0이되는거지 그다음에 SETMASK랑 or을 때려서 SETMASK가 `0x00000001` (1비트 스톱 비트를 의미한다고 가정)이라면 0x12345671 이되는거임 그걸 CR2에 쓰는거

▼ **uint32_t HAL_RCC_GetPCLK1Freq(void) in stm32f1xx_hal_rcc.c >> Peripheral Clock 주파수 반환**

```
**uint32_t** **HAL_RCC_GetPCLK1Freq**(**void**)

{

/* Get HCLK source and Compute PCLK1 frequency ----------------------------

**return** (**HAL_RCC_GetHCLKFreq**() >> APBPrescTable[(RCC->CFGR & RCC_CF

}


HAL_RCC_GetHCLKFreq() 호출:

이 함수는 HCLK (High-speed Clock)의 주파수를 반환합니다. HCLK는 AHB 버스에서 사용되
APBPrescTable:

APBPrescTable은 APB1 버스의 분주기 값을 저장하는 테이블입니다. 분주기 값은 RCC 레지스
RCC_CFGR_PPRE1 값 추출**:

RCC->CFGR 레지스터에서 PPRE1 (APB1 분주기) 값을 가져옵니다. 이 값은 APB1 클럭이 HCL
PCLK1 계산:

HCLK 주파수를 APBPrescTable에 있는 값에 따라 나눕니다.
최종적으로 PCLK1 (Peripheral Clock 1) 주파수를 반환합니다.
```

▼ huart→Instance→BRR = UART_BRR_SAMPLING16(pclk, huart→Init.BaudRate);

USART2의 BRR 레지스터 = UART_BRR_SAMPLING16(pclk, huart→Init.BaudRate)

```
#define UART_BRR_SAMPLING16(_PCLK_, _BAUD_)            (((UART_DIVMANT_SAM
                                                         (UART_DIVFRAQ_SAM
                                                        (UART_DIVFRAQ_SAMP
mantissa를 계산해서 BRR 레지스터에 할당해줌 >> 이거 자세한건 필요없을듯?
```

/* In asynchronous mode, the following bits must be kept cleared:

// asynchronous니까 CR2 , CR3 관련 필요없는 비트들 클리어해줌

- LINEN and CLKEN bits in the USART_CR2 register,

- SCEN, HDSEL and IREN bits in the USART_CR3 register.*/

CLEAR_BIT(huart→Instance→CR2, (USART_CR2_LINEN | USART_CR2_CLKEN));

CLEAR_BIT(huart→Instance→CR3, (USART_CR3_SCEN | USART_CR3_HDSEL | USART_CR3_IREN));

/* Enable the peripheral */

__HAL_UART_ENABLE(huart);

// 아까 0으로 클리어했던 CR1 의 UE 다시 1로 set

// #define __HAL_UART_ENABLE(__HANDLE__) ((__HANDLE__)→Instance→CR1 |= USART_CR1_UE)

/* Initialize the UART state */

// 이제 아래 State setting이랑 error 처리 ,

huart→ErrorCode = HAL_UART_ERROR_NONE;

huart→gState = *HAL_UART_STATE_READY*;

huart→RxState = *HAL_UART_STATE_READY*;

huart→RxEventType = HAL_UART_RXEVENT_TC;

//#define HAL_UART_RXEVENT_TC (0x00000000U)

▼ **이 부분까지 해서 설정된 hadle type setting**

typedef struct __UART_HandleTypeDef

{

USART_TypeDef *Instance; >>> USART2(레지스터들 들어있는 구조체 시작주소)

UART_InitTypeDef Init; /*!< UART communication parameters */ >>>> main.c에서 한 init→xxxxx setting들 >> uart_init에서 레지스터 셋팅으로 사용됨

const uint8_t *pTxBuffPtr; /*!< Pointer to UART Tx transfer Buffer */ >>>> NULL

uint16_t TxXferSize; /*!< UART Tx Transfer size */ >>>> 0

__IO **uint16_t** TxXferCount; /*!< UART Tx Transfer Counter */ >>>>> 0

uint8_t *pRxBuffPtr; /*!< Pointer to UART Rx transfer Buffer */ >>>>> NULL

uint16_t RxXferSize; /*!< UART Rx Transfer size */ >>>>>> 0

__IO **uint16_t** RxXferCount; /*!< UART Rx Transfer Counter */ >>>>>0

__IO **HAL_UART_RxTypeTypeDef** ReceptionType; /*!< Type of ongoing reception */ >>>>> 0

▼ **HAL_UART_RxTypeTypeDef in stm32f1xx_hal_uart.h**

```
/**
  * @brief HAL UART Reception type definition
  * @note  HAL UART Reception type value aims to identify which type of Re
  *        This parameter can be a value of @ref UART_Reception_Type_Value
  *           HAL_UART_RECEPTION_STANDARD         = 0x00U,
```

```
  *           HAL_UART_RECEPTION_TOIDLE           = 0x01U,
  */
typedef uint32_t HAL_UART_RxTypeTypeDef;
```

__IO **HAL_UART_RxEventTypeTypeDef** RxEventType; /*!< Type of Rx Event */ >>>> **HAL_UART_RXEVENT_TC (0x00000000U)**

▼ **HAL_UART_RxEventTypeTypeDef in stm32f1xx_hal_uart.h**

```
/**
  * @brief HAL UART Rx Event type definition
  * @note  HAL UART Rx Event type value aims to identify which type of Eve
  *        leading to call of the RxEvent callback.
  *        This parameter can be a value of @ref UART_RxEvent_Type_Values
  *           HAL_UART_RXEVENT_TC                 = 0x00U,
  *           HAL_UART_RXEVENT_HT                 = 0x01U,
  *           HAL_UART_RXEVENT_IDLE               = 0x02U,
  */
typedef uint32_t HAL_UART_RxEventTypeTypeDef;
```

DMA_HandleTypeDef *hdmatx; /*!< UART Tx DMA Handle parameters */ >>>> NULL

DMA_HandleTypeDef *hdmarx; /*!< UART Rx DMA Handle parameters */ >>> NULL

HAL_LockTypeDef Lock; /*!< Locking object */ >>>>> *HAL_UNLOCKED >>> HAL_UNLOCKED = 0x00U*

▼ **HAL_LockTypeDef** in stm32f1xx_hal_def.h

```
typedef enum
{
  HAL_UNLOCKED = 0x00U,
  HAL_LOCKED   = 0x01U
} HAL_LockTypeDef;
```

__IO **HAL_UART_StateTypeDef** gState; /*!< UART state information related to global Handle management >>>> *HAL_UART_STATE_READY >>> *HAL_UART_STATE_READY* = 0x20U,

▼ **HAL_UART_StateTypeDef in stm32f1xx_hal_uart.h**

```
**typedef** **enum**

{

*HAL_UART_STATE_RESET*             = 0x00U,    /*!< Peripheral is not yet

Value is allowed for gState and RxState */

*HAL_UART_STATE_READY*             = 0x20U,    /*!< Peripheral Initialized

Value is allowed for gState and RxState */

*HAL_UART_STATE_BUSY*              = 0x24U,    /*!< an internal process is

Value is allowed for gState only */
```

```
*HAL_UART_STATE_BUSY_TX*           = 0x21U,    /*!< Data Transmission proc

Value is allowed for gState only */

*HAL_UART_STATE_BUSY_RX*           = 0x22U,    /*!< Data Reception process

Value is allowed for RxState only */

*HAL_UART_STATE_BUSY_TX_RX*        = 0x23U,    /*!< Data Transmission and

Not to be used for neither gState nor RxState.

Value is result of combination (Or) between gState and RxState values */

*HAL_UART_STATE_TIMEOUT*           = 0xA0U,    /*!< Timeout state

Value is allowed for gState only */

*HAL_UART_STATE_ERROR*             = 0xE0U     /*!< Error

Value is allowed for gState only */

} **HAL_UART_StateTypeDef**;
```

and also related to Tx operations.

This parameter can be a value of @ref HAL_UART_StateTypeDef */

__IO **HAL_UART_StateTypeDef** RxState; /*!< UART state information related to Rx operations. >>>> *HAL_UART_STATE_READY >>>* *HAL_UART_STATE_READY* = 0x20U,

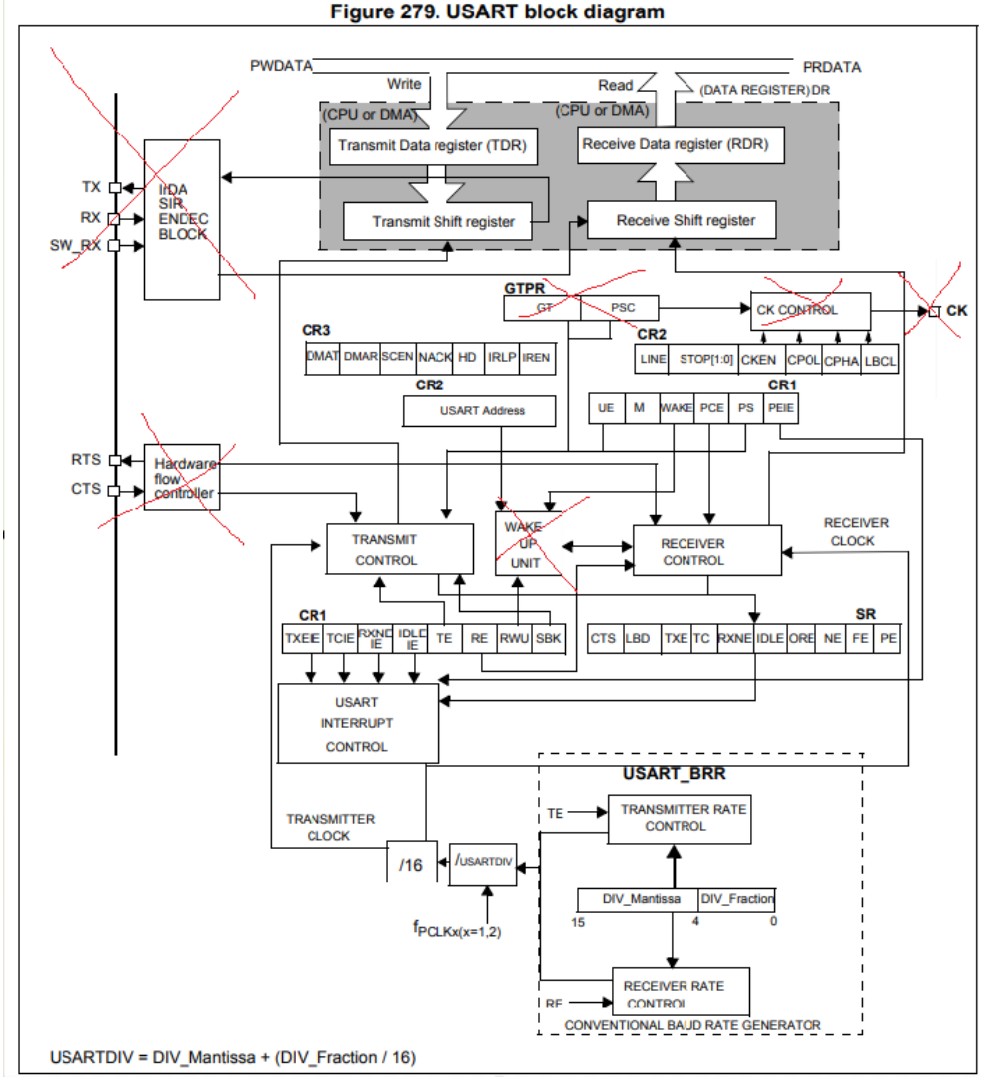
This parameter can be a value of @ref HAL_UART_StateTypeDef */

__IO **uint32_t** ErrorCode; /*!< UART Error code */ >>>HAL_UART_ERROR_NONE; >>>> **#define HAL_UART_ERROR_NONE** 0x00000000U /*!< No error */

▼ 이 부분까지 해서 설정된 usart2 레지스터 (Default 0) >>> 정리하자면 매뉴얼 블록다이어그램에서 이거뭐지? 하는건 안씀

쓰는부분

Figure 279. USART block diagram



CR1

UE >>> 0 >>> 0 >> 1

M >>>> 0

RE >>> 1

TE >>>> 1

PCE , PS >>> 0

CR2

Stop Bits: STOP[13:12] bits >>> 00

LINEN >>> 0 >>> LIN mode disabled

CLKEN >>> 0 >>> CK pin disabled

CR3

RTSE >>> 0 >>> RTS hardware flow control disabled

CTSE >>>> 0 >>> CTS hardware flow control disabled

SCEN >>>> 0 >>> Smartcard Mode disabled

HDSEL >>> 0 >>>Half duplex mode is not selected

IREN >>>> 0 >>> IrDA disabled

BRR

Mantisa setting based on baud rate <<< 우리가 설정해놓은.

return *HAL_OK*;

}

{

Error_Handler();

}

/* USER CODE BEGIN USART2_Init 2 */

/* USER CODE END USART2_Init 2 */

}

▼ (6) - **HAL_StatusTypeDef** RcvStat ;

이거는 **HAL_UART_Receive 가 return으로 status 보내주니까 그거 확인용**

▼ (7) - RcvStat = **HAL_UART_Receive**(&huart2, inputData, 1, 100) ; // receive data

▼ 코드

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint1
{
  uint8_t  *pdata8bits;
  uint16_t *pdata16bits;
  uint32_t tickstart = 0U;

  /* Check that a Rx process is not already ongoing */
  if (huart->RxState == HAL_UART_STATE_READY)
  {
    if ((pData == NULL) || (Size == 0U))
    {
      return  HAL_ERROR;
    }

    huart->ErrorCode = HAL_UART_ERROR_NONE;
    huart->RxState = HAL_UART_STATE_BUSY_RX;
    // RX Busy >> 처리중 flag 올려줌 rx state는 그거 알지 state 구조체임
    //*HAL_UART_STATE_BUSY_RX*           = 0x22U,    /*!< Data Reception process is
    huart->ReceptionType = HAL_UART_RECEPTION_STANDARD;
    // 여기서 IDLE모드는 저전력 모드에서 사용되며, 데이터 수신이 필요한 경우에만 활성화하여 전력을

    /* Init tickstart for timeout management */
    tickstart = HAL_GetTick();
        // HAL_GetTick 함수는 시스템의 현재 틱 카운터 값을 반환 >> 이게 있어야 Time out인지 아
    huart->RxXferSize = Size;
    huart->RxXferCount = Size;
        // RX 패킷 (한번에 받는 단위)Size를 넣어줌

    /* In case of 9bits/No Parity transfer, pRxData needs to be handled as a uint16
    if ((huart->Init.WordLength == UART_WORDLENGTH_9B) && (huart->Init.Parity == UA
    {
      pdata8bits  = NULL;
```

```
      pdata16bits = (uint16_t *) pData;
    }
    else
    { // 8BIT이니까 여기로 들어감.
      pdata8bits  = pData;
      pdata16bits = NULL;
    }

    /* Check the remain data to be received */
    while (huart->RxXferCount > 0U)
    { // 만약에 사이즈가 1보다 크다면 여러개 받을때 까지 while 돌린다는거
      if (UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_RXNE, RESET, tickstart, Time
      { // 아까 틱 카운터가지고 timeout인지 확인해야하니까 그거 확인. 근데 왜 이게 앞에있지???
          // 아래 토글 함수 설명
        //#define UART_FLAG_RXNE  ((uint32_t)USART_SR_RXNE) in stm32f1xx_hal_uart.h
        /*  SR의 RXNE 레지스터 설명 >> 쉽게말하면 읽을게 있다면 1이된다는거임 다읽으면 0되고 HW0
                이게 1되면 인터럽트가 생성되는거고.

            Bit 5 RXNE: 읽기 데이터 레지스터가 비어 있지 않음
                이 비트는 RDR 시프트 레지스터의 내용이 USART_DR 레지스터로 전송되면 하드웨어에 S
                만약 RXNEIE=1이 USART_CR1 레지스터에 설정되어 있으면 인터럽트가 생성됩니다.
                이 비트는 USART_DR 레지스터를 읽으면 클리어됩니다. 또한 RXNE 플래그는 0을 써서
                이 클리어 방식은 멀티버퍼 통신에만 권장됩니다.
                0: 데이터가 수신되지 않음1: 수신된 데이터가 읽을 준비가 됨*/

        /*  CR1의 RXNEIE 레지스터 설명
                Bit 5 RXNEIE: RXNE interrupt enable
                        This bit is set and cleared by software.
                        0: Interrupt is inhibited
                        1: A USART interrupt is generated whenever ORE=1 or RXNE=1
        */
        /*
                HAL setting 임 그니까 저 함수에서 ERROR가 리턴되면 타임아웃이라는거지.
                        typedef enum
                        {
                          HAL_OK       = 0x00U,
                          HAL_ERROR    = 0x01U,
                          HAL_BUSY     = 0x02U,
                          HAL_TIMEOUT  = 0x03U
                        } HAL_StatusTypeDef;

        */

        huart->RxState = HAL_UART_STATE_READY;

        return HAL_TIMEOUT;
      }
      if (pdata8bits == NULL)
      {
        *pdata16bits = (uint16_t)(huart->Instance->DR & 0x01FF);
        pdata16bits++;
      }
      else
      {
        if ((huart->Init.WordLength == UART_WORDLENGTH_9B) || ((huart->Init.WordLen
```

```
        {
          *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x00FF);
        }
        else
        {
          *pdata8bits = (uint8_t)(huart->Instance->DR & (uint8_t)0x007F);
        }
        pdata8bits++;
      }
      huart->RxXferCount--;
    }

    /* At end of Rx process, restore huart->RxState to Ready */
    huart->RxState = HAL_UART_STATE_READY;

    return HAL_OK;
  }
  else
  {
    return HAL_BUSY;
  }
}
```

▼ UART_WaitOnFlagUntilTimeout(huart, UART_FLAG_RXNE, RESET, tickstart, Timeout) **in stm32f1xx_hal_uart.c**

```
static HAL_StatusTypeDef UART_WaitOnFlagUntilTimeout(UART_HandleTypeDef *huart,
                                                     uint32_t Tickstart, uint32_
{
  /* Wait until flag is set */
  while ((__HAL_UART_GET_FLAG(huart, Flag) ? SET : RESET) == Status)
  {  // 이거는 SR의 RXNE 레지스터가 1되면(DR에 데이터가 들어오면) 나가지는거고
       // 계속 0 이면(받을 데이터가 없으면) 아래 계속 실행되는거임.
         // 그러니까 여기서 말하는 time아웃은 일정기간 송신측에서 데이터를 안보내는거임
         // 이 방식으로, UART 통신에서 데이터가 도착하지 않는 상황을 처리하여 시스템이 계속 무

    /* Check for the Timeout */
    if (Timeout != HAL_MAX_DELAY)
    { // #define HAL_MAX_DELAY      0xFFFFFFFFU in stm32f1xx_hal_def.h
      if (((HAL_GetTick() - Tickstart) > Timeout) || (Timeout == 0U))
      {  // 현재 틱 가져와서 tickstart랑 비교해서 timeout확인 만약 out이면 return timeou

        return HAL_TIMEOUT;
      }

      if ((READ_BIT(huart->Instance->CR1, USART_CR1_RE) != 0U) && (Flag != UART_
      { // 만약에 타임아웃이 아니고 RE가 1이고 (enable 되어있고) 내가 확인하고자 했던 flag가
        if (__HAL_UART_GET_FLAG(huart, UART_FLAG_ORE) == SET)
        { //#define UART_FLAG_ORE  ((uint32_t)USART_SR_ORE) >> ORE 1인지 확인 . >>
          /* Clear Overrun Error flag*/
          __HAL_UART_CLEAR_OREFLAG(huart);

          /* Blocking error : transfer is aborted
          Set the UART state ready to be able to start again the process,
          Disable Rx Interrupts if ongoing */
```

```
          UART_EndRxTransfer(huart);

          huart->ErrorCode = HAL_UART_ERROR_ORE;

          /* Process Unlocked */
          __HAL_UNLOCK(huart);

          return HAL_ERROR;
        }
      }
    }
  }
  return HAL_OK;
}
```

▼ __HAL_UART_GET_FLAG(huart, Flag) **in stm32f1xx_hal_uart.h**

#define __HAL_UART_GET_FLAG(__HANDLE__, __FLAG__) (((__HANDLE__)→Instance→SR & (__FLAG__)) == (__FLAG__))

▼ OVERRUN - SR→ORE: Overrun error

하드웨어적으로 RDR 레지스터에 오버런이 발생한 경우 발생

▼ **(8) - HAL_UART_Transmit**(&huart2, inputData, 1, 100) ; // send received data

▼ 코드

```
/* USeR CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
```

```
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart2;
//(1)

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
//(2)
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
```

```
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */
  HAL_StatusTypeDef RcvStat ;
  uint8_t buffer[10] = "Hello!\n" ;
  uint8_t inputData[20] ;

  HAL_UART_Transmit(&huart2, buffer, strlen(buffer), 100) ;  // send start data
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
      RcvStat = HAL_UART_Receive(&huart2, inputData, 1, 100) ;  // receive data
      if (RcvStat == HAL_OK) {  // receive check
          HAL_UART_Transmit(&huart2, inputData, 1, 100) ;  // send received data
      }
      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */

  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
```

```
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
/* USER CODE BEGIN MX_GPIO_Init_1 */
```

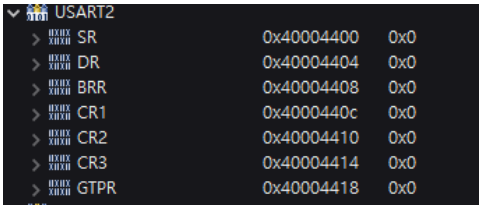
```
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

▼ 의문점 1 (거의 해결)USART_CR1_OVER8 이게 Define 되어있어야 oversampling 설정이 들어가는데 아무데도 이게 정의가 된 곳이없음.

이건 내가볼때 MCU종류마다 OVER8 쓸수있는 레지스터를 가진 MCU가 있는거고 내가 쓰는건 없는거 같음.

▼ 의문점 2 (미궁) UART_WaitOnFlagUntilTimeout 에서 왜 RXNE가 0일때(읽을 데이터가 없을때)만 overrun check를 하는 거지?

▼ 기억해야할점 중요 , 어차피 USART2의 모든 레지스터는 reset value default 값이 다 0임



▼ 기억해야할점 중요 , 어차피 USART의 Handler도 전역변수이기 때문에 0으로 초기화가 됨 포인터는 NULL **(gstate 는 0 즉 reset)**

전역변수로 선언된 구조체는 프로그램이 시작될 때 **자동으로 0으로 초기화**됩니다. 따라서 내부 구조체의 멤버들도 자동으로 초기화됩니다.

- 정수형(`int`, `short`, `long`) 멤버: `0`으로 초기화
- 실수형(`float`, `double`) 멤버: `0.0`으로 초기화
- 포인터형 멤버: `NULL`로 초기화
- 구조체 안에 또 다른 구조체가 있는 경우, 그 내부의 멤버들도 0으로 초기화됩니다.

▼ 느낀점(1) - HAL Library 는 아름답다.

예시 1: 동작과 의미별로 기가막히게 소스코드 나눠둔부분 >> main >> hal driver 중에서도 hal , uart_hal , 103xxrb 같이

예시 2 : 각 레지스터를 아래처럼 나눠서 이걸 flag에 matching시키고 get_flag나 modify 나 이런걸로 비트연산해서 원하는 작업을 하는거

#define USART_SR_RXNE_Pos (5U)

#define USART_SR_RXNE_Msk (0x1UL << USART_SR_RXNE_Pos) /*!< 0x00000020 */

#define USART_SR_RXNE USART_SR_RXNE_Msk /*!< Read Data Register Not Empty */

예시 3 :