Polling vs Interrupt

송수신 부분 링크 - 분석 - UART _ POLLING

▼ 개요 (Polling vs Interrupt)

기존 polling code의 아래 부분에서 1초가 걸리는 일(아래 참조)를 하려면 Receive를 할 수가 없게 됨 >> 왜냐면 계속 RXNE가 SET되었는지 확인해야하니까 >> 이거를 해결하기 위해서 인터럽트 방식이 사용됨

▼ 1초가 걸리는 일 (이해를 돕기위한 예시)

```
char temp_2 = 'A';

UART_HandleTypeDef *temp_uart2 = &huart2;

temp_uart2->Instance->DR = temp_2;

HAL_Delay(1000);
```

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 */

}

▼ Polling vs Interrupt의 이해하기 쉬운 예시

INTERRUPT의 포인트는 두가지 >> 완료 전달과 우선순위 설정.

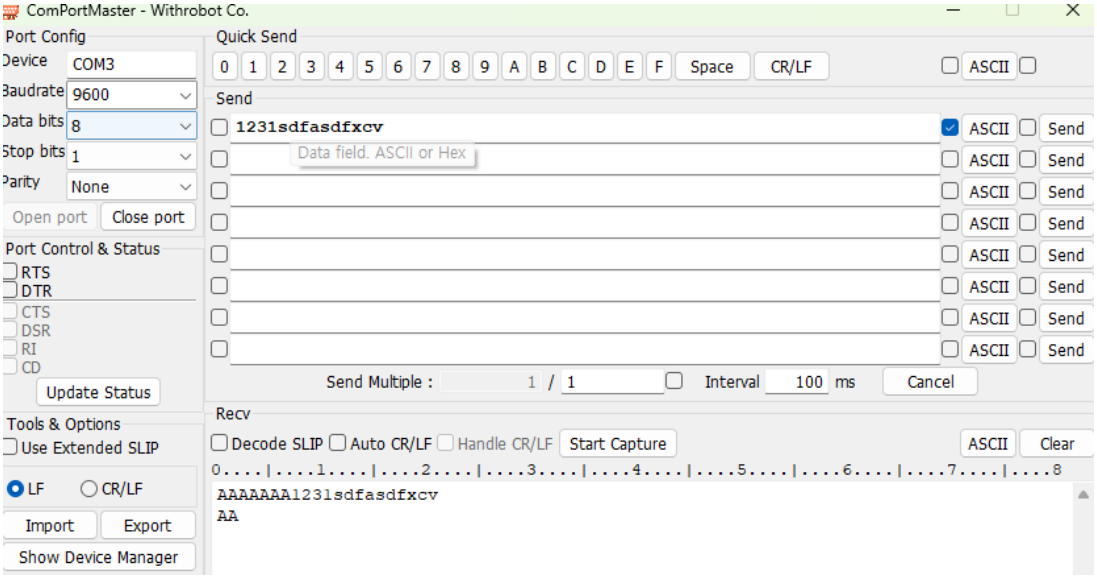
▼ Interrupt 실행 부분 진행 한 것 & 스터디

▼ Polling vs Interrupt (Plain)

▼ 개요

기본적으로 송수신 동작하기 위해서 Plain 부분에서 일부 수정한 부분이 있음 → ADDED 주석처리되어있음

동작확인 부분



▼ 코드 Main.c

```
/* 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 b
  * in the root directory of this software component
  * If no LICENSE file comes with this software, it
```

```
  *
  ******************************************************
  */
/* 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;

/* USER CODE BEGIN PV */
//Added 1028
volatile char Rx_Data;
// this volatill is super important cuz it is chaneg
//Added 1028
/* USER CODE END PV */

/* Private function prototypes -----------------------
```

```
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_NVIC_Init(void);
/* 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
  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();
  //Added 1028
  HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_Data, 1
  //ADDED 1028
  /* Initialize interrupts */
  MX_NVIC_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
      //Added 1028
      char temp_2 = 'A';

      UART_HandleTypeDef *temp_uart2 = &huart2;

      temp_uart2->Instance->DR = temp_2;

      HAL_Delay(1000);
      // 1sec delay
      //Added 1028
    /* 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 t
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORT
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICAL
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_O
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|R
                                |RCC_CLOCKTYPE_PCLK1|R
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_
  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_
  {
    Error_Handler();
  }
}

/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
```

```
  /* USART2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(USART2_IRQn);
}

/**
  * @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 */
//Added 1028
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *hua
{
    if (huart->Instance == USART2)
    {

        volatile char temp = Rx_Data;
        huart->Instance->DR = temp;
        HAL_UART_Receive_IT(huart, (uint8_t *)&Rx_Da
        // this has to be re-enabled
    }
}
//Added 1028
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of err
  * @retval None
  */
void Error_Handler(void)
```

```
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report t
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

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

▼ INTERRUPT EN으로 인한 코드 차이점

▼ Polling Code 대비 새로 생긴 부분

Main.C

▼ MX_NVIC_Init();

▼ HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_Data, 1);

▼ void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) 재정의 부분

stm32f1xx_it.c / h

▼ **void USART2_IRQHandler(void)** 정의 추가

```
// 추가된 부분
extern UART_HandleTypeDef huart2;

void USART2_IRQHandler(void)
{
  /* USER CODE BEGIN USART2_IRQn 0 */

  /* USER CODE END USART2_IRQn 0 */
  HAL_UART_IRQHandler(&huart2);
  /* USER CODE BEGIN USART2_IRQn 1 */

  /* USER CODE END USART2_IRQn 1 */
}
```

stm32f1xx_hal_msp.c

▼ USART2 interrupt DeInit 부분 추가

```
/* USART2 interrupt DeInit */

HAL_NVIC_DisableIRQ(USART2_IRQn);
```

▼ INTERRUPT 동작에 대한 소스코드 분석.

결국 MX_NVIC_Init() 으로 NVIC에서 USART2에 대한 Enable과 우선순위(선점,서브) 설정 후 HAL_UART_Receive_IT로 RXNEIE를 셋팅하고 RXNE가 셋되었을때(시작비트가 읽혔을때) `USARTx_IRQHandler` >> `HAL_UART_IRQHandler(&huartx)` >> `HAL_UART_RxCpltCallback` 순으로 실행 되는 거

▼ 메뉴얼

Received data ready to be read 인터럽트는 결국 RXNEIE && RXNE로 발생하는거임 >> RXNEIE가 1일떄 RXNE가 셋되면(시작비트가 감지되면) >> 인터럽트발생

- **UART 인터럽트 핸들러 실행**: `USARTx_IRQHandler` 함수가 호출됩니다. 이 함수는 `stm32f1xx_it.c` 파일에 정의되어 있으며, 모든 USART 인터럽트의 진입점입니다.

- **HAL_UART_IRQHandler 함수 호출**: `USARTx_IRQHandler` 내부에서 `HAL_UART_IRQHandler(&huartx);`가 호출됩니다. 여기서 `huartx`는 특정 UART 인스턴스(예: `USART2`일 경우 `huart2`)입니다.
- **HAL_UART_RxCpltCallback**: `HAL_UART_IRQHandler`는 인터럽트 플래그를 확인하여, 만약 `RXNE` 플래그가 설정된 상태라면, `HAL_UART_RxCpltCallback` 함수를 호출합니다. 사용자가 원하는 데이터 수신 처리를 이 콜백 함수에 작성할 수 있습니다.

Table 196. USART interrupt requests

Interrupt event	Event flag	Enable Control bit
Transmit data register empty	TXE	TXEIE
CTS flag	CTS	CTSIE
Transmission complete	TC	TCIE
Received data ready to be read	RXNE	RXNEIE
Overrun error detected	ORE	
Idle line detected	IDLE	IDLEIE
Parity error	PE	PEIE
Break flag	LBD	LBDIE
Noise flag, Overrun error and Framing error in multibuffer communication	NE or ORE or FE	EIE(1)

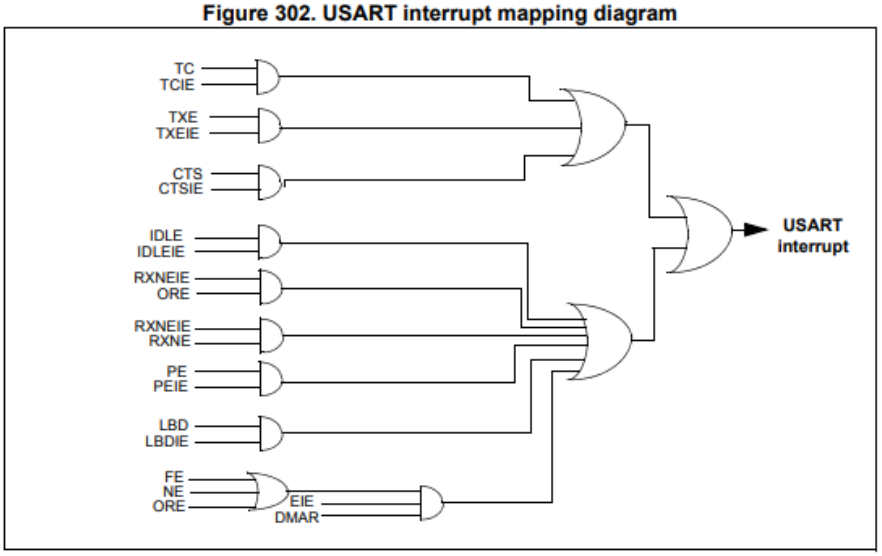
1. This bit is used only when data reception is performed by DMA.

The USART interrupt events are connected to the same interrupt vector (see *Figure 302*).

- During transmission: Transmission Complete, Clear to Send or Transmit Data register empty interrupt.
- While receiving: Idle Line detection, Overrun error, Receive Data register not empty, Parity error, LIN break detection, Noise Flag (only in multi buffer communication) and Framing Error (only in multi buffer communication).

These events generate an interrupt if the corresponding Enable Control Bit is set.

Figure 302. USART interrupt mapping diagram



*뒤에 직선은 AND 곡선은 OR임

▼ MX_NVIC_Init() **in main.c**;

`IRQn`은 **Interrupt Request Number**의 약자로 해당 인터럽트를 NVIC 레지스터에서 정해진 우선순위대로 Enable하고 셋팅하는 함수

```
/* USART2_IRQn interrupt configuration */
HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(USART2_IRQn);
```

▼ HAL_NVIC_SetPriority(USART2_IRQn, 0, 0); **in stm32f1xx_hal_cortex.c**

설명: 입력값 (선점,서브 우선순위)를 바탕으로 USART2_IRQn의 우선순위를 설정 >> **이건 CUBE MX에서 설정한 우선순위가 적용되어 있음**

▼ PreemptPriority(선점 우선순위) 와 SubPriority(서브 우선순위) >> 두가지 우선순위로 인터럽트 발생시킴 선점이동일하면 서브로 우선순위 설정.

2. `uint32_t PreemptPriority`

- `PreemptPriority`는 인터럽트의 "선점 우선순위(Preemption Priority)"를 지정합니다.
- 선점 우선순위가 낮을수록 높은 우선순위를 가지며, 높은 우선순위의 인터럽트는 현재 실행 중인 낮은 우선순위의 인터럽트를 선점하여 처리됩니다.
- 시스템이 여러 개의 인터럽트 우선순위를 갖고 있을 때, 이 우선순위를 기준으로 어떤 인터럽트가 다른 인터럽트를 선점할지 결정됩니다.
- 예를 들어, 선점 우선순위가 0인 인터럽트는 우선순위가 1인 인터럽트를 선점하여 실행됩니다.
- **참고**: 선점 우선순위가 동일하다면, `SubPriority`가 적용됩니다.

3. `uint32_t SubPriority`

- `SubPriority`는 인터럽트의 "서브 우선순위(Sub Priority)"를 지정합니다.

- 같은 선점 우선순위를 가진 인터럽트 간에 실행 순서를 정하기 위해 사용됩니다.
- 선점 우선순위가 동일한 두 인터럽트가 동시에 발생했을 때, 서브 우선순위가 낮은 쪽이 먼저 처리됩니다.
- 예를 들어, 선점 우선순위가 모두 1인 두 인터럽트 중 서브 우선순위가 0인 인터럽트가 서브 우선순위가 1인 인터럽트보다 먼저 실행됩니다.
- 서브 우선순위는 선점과 달리 중첩되지 않으며, 인터럽트 간의 실행 순서만 결정합니다.

우선순위 설정 예시

- 예를 들어, `HAL_NVIC_SetPriority(USART2_IRQn, 1, 0)`라고 설정할 경우:
  - `USART2_IRQn`에 해당하는 UART2 인터럽트의 선점 우선순위는 1, 서브 우선순위는 0으로 설정됩니다.
  - 선점 우선순위가 0인 다른 인터럽트가 발생하면 이 UART2 인터럽트는 선점당할 수 있으며, 다른 선점 우선순위 1인 인터럽트 중에서는 서브 우선순위가 높은 인터럽트보다 먼저 실행됩니다.

```
void HAL_NVIC_SetPriority(IRQn_Type IRQn, uint
{
  uint32_t prioritygroup = 0x00U;

  /* Check the parameters */
  assert_param(IS_NVIC_SUB_PRIORITY(SubPriorit
  // 해당 우선순위가 validate한지 검사
  assert_param(IS_NVIC_PREEMPTION_PRIORITY(Pre
  // 해당 우선순위가 validate한지 검사
  prioritygroup = NVIC_GetPriorityGrouping();
  // 선점 우선순위 그룹 return based on CORE 종류
  // 인터럽트의 "선점 우선순위(Preemption Priority)
  // 해당보드는 m3로 아래와 같은 설정이 들어감 in CMSI
  /*
  __STATIC_INLINE uint32_t __NVIC_GetPriorityG
{
```

```
  return ((uint32_t)((SCB->AIRCR & SCB_AIRCR_P
}

  */
  NVIC_SetPriority(IRQn, NVIC_EncodePriority(p
  // 반환값대로 NVIC 및 SCB의 레지스터 변경
}
```

▼ NVIC_SetPriority(IRQn, NVIC_EncodePriority(prioritygroup, PreemptPriority, SubPriority)) in core_cm3.h;

```
__STATIC_INLINE void __NVIC_SetPriority(IRQ
{
  if ((int32_t)(IRQn) >= 0)
  {
    NVIC->IP[((uint32_t)IRQn)]
  }
  else
  {
    SCB->SHP[(((uint32_t)IRQn) & 0xFUL)-4UL
  }
}
```

▼ HAL_NVIC_EnableIRQ(USART2_IRQn); **in stm32f1xx_hal_cortex.c**

```
void HAL_NVIC_EnableIRQ(IRQn_Type IRQn)
{
  /* Check the parameters */
  assert_param(IS_NVIC_DEVICE_IRQ(IRQn));
  //USART2_IRQn 이 유효한 값인지 확인

  /* Enable interrupt */
  NVIC_EnableIRQ(IRQn);
   //NVIC 레지스터 설정으로 해당 IRQn Enalbe
```

▼ NVIC_EnableIRQ(IRQn)

```
__STATIC_INLINE void __NVIC_EnableIRQ(IRQn_
{
  if ((int32_t)(IRQn) >= 0)
  {
    NVIC->ISER[(((uint32_t)IRQn) >> 5UL)] =
  }
}
```

▼ HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_Data, 1) **in main.c**

```
HAL_StatusTypeDef HAL_UART_Receive_IT(UART_Handle
{
  /* Check that a Rx process is not already ongoi
  if (huart->RxState == HAL_UART_STATE_READY)
  { // Rx의 상태확인
    if ((pData == NULL) || (Size == 0U))
    {
      return HAL_ERROR;
    }

    /* Set Reception type to Standard reception *
    huart->ReceptionType = HAL_UART_RECEPTION_STA

    return (UART_Start_Receive_IT(huart, pData, S
  }
  else
  {
    return HAL_BUSY;
  }
}
```

▼ UART_Start_Receive_IT(huart, pData, Size)

```
HAL_StatusTypeDef UART_Start_Receive_IT(UART_H
{
  huart->pRxBuffPtr = pData;
  huart->RxXferSize = Size;
```

```
  huart->RxXferCount = Size;
    // 데이터 전송할 버퍼와 size,count를 핸들러에 저장
  huart->ErrorCode = HAL_UART_ERROR_NONE;
  huart->RxState = HAL_UART_STATE_BUSY_RX;
    // rx 스테이터스 변경

  if (huart->Init.Parity != UART_PARITY_NONE)
  { // 여기는 parity_none이 아닐때 오류확인 interru
    /* Enable the UART Parity Error Interrupt
    __HAL_UART_ENABLE_IT(huart, UART_IT_PE);
  }
    // 오류 확인 및 RXNE 인터럽트 Eable.
  /* Enable the UART Error Interrupt: (Frame e
  __HAL_UART_ENABLE_IT(huart, UART_IT_ERR);
   //아래처럼 EIE를 1로 SET해주는거임.
   //#define UART_IT_ERR ((uint32_t)(UART_CR3_
  /* Enable the UART Data Register not empty I
  __HAL_UART_ENABLE_IT(huart, UART_IT_RXNE);
  // 아래처럼 RXNEIE를 1로 SET해주는거임.
  // #define UART_IT_RXNE ((uint32_t)(UART_CR1_

  return HAL_OK;
}
```

▼ 메뉴얼 : RXNEIE

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 in the USART_SR register

▼ 메뉴얼 : EIE

EIE: Error interrupt enable
Error Interrupt Enable Bit is required to enable interrupt generation in case of a framing
error, overrun error or noise error (FE=1 or ORE=1 or NE=1 in the USART_SR register) in
case of Multi Buffer Communication (DMAR=1 in the

USART_CR3 register).
0: Interrupt is inhibited
1: An interrupt is generated whenever DMAR=1 in the USART_CR3 register and FE=1 or ORE=1 or NE=1 in the USART_SR register.

▼ `USART2_IRQHandler` **in stm32f1xx_it.c** >> CUBE MX 셋팅으로 아래처럼 추가되어있고 단순히 핸들러를 외부 참조로 선언하고 HAL_UART_IRQHandler를 호출하는 부분.

```
extern UART_HandleTypeDef huart2;

void USART2_IRQHandler(void)
{
  /* USER CODE BEGIN USART2_IRQn 0 */

  /* USER CODE END USART2_IRQn 0 */
  HAL_UART_IRQHandler(&huart2);
  /* USER CODE BEGIN USART2_IRQn 1 */

  /* USER CODE END USART2_IRQn 1 */
}
```

▼ `HAL_UART_IRQHandler(&huart2)` **in stm32f1xx_hal_uart.c >>>> 제일 중요한거 핸들러에서 정해진 값대로 Recieve 및** `HAL_UART_RxCpltCallback` 호출

```
void HAL_UART_IRQHandler(UART_HandleTypeDef *huar
{
  uint32_t isrflags   = READ_REG(huart->Instance-
  uint32_t cr1its     = READ_REG(huart->Instance-
  uint32_t cr3its     = READ_REG(huart->Instance-
  uint32_t errorflags = 0x00U;
  uint32_t dmarequest = 0x00U;
    // 각 레지스터 설정값 및 Status 가져오는부분.

  /* If no error occurs */
  errorflags = (isrflags & (uint32_t)(USART_SR_PE
  if (errorflags == RESET)
  {
```

```
    /* UART in mode Receiver ---------------------
    if (((isrflags & USART_SR_RXNE) != RESET) &&
    {
      UART_Receive_IT(huart);
      return;
    }
  }
  // 사실 여기까지가 제일 중요한 부분임 PE || FE || ORE
  // 해당 함수 아래 토글 참조
  // 오류 정리 아래 토글 참조.
  /* If some errors occur */
  if ((errorflags != RESET) && (((cr3its & USART_
                                || ((cr1its & (US
  {
    /* UART parity error interrupt occurred -----
    //여기부턴 각 에러에 대해 핸들러 플래그 설정.
    if (((isrflags & USART_SR_PE) != RESET) && ((
    {
      huart->ErrorCode |= HAL_UART_ERROR_PE;
    }

    /* UART noise error interrupt occurred ------
    if (((isrflags & USART_SR_NE) != RESET) && ((
    {
      huart->ErrorCode |= HAL_UART_ERROR_NE;
    }

    /* UART frame error interrupt occurred ------
    if (((isrflags & USART_SR_FE) != RESET) && ((
    {
      huart->ErrorCode |= HAL_UART_ERROR_FE;
    }

    /* UART Over-Run interrupt occurred ---------
    if (((isrflags & USART_SR_ORE) != RESET) && (

    {
      huart->ErrorCode |= HAL_UART_ERROR_ORE;
```

```
    }

    /* Call UART Error Call back function if need
    if (huart->ErrorCode != HAL_UART_ERROR_NONE)
    {
      /* UART in mode Receiver ------------------
      if (((isrflags & USART_SR_RXNE) != RESET) &&
      {//그 후 동일하게 리시브 진행 후 에러 콜백 호출.
        UART_Receive_IT(huart);
      }

      /* If Overrun error occurs, or if any error
         consider error as blocking */
      dmarequest = HAL_IS_BIT_SET(huart->Instance
      if (((huart->ErrorCode & HAL_UART_ERROR_ORE
      {
        /* Blocking error : transfer is aborted
           Set the UART state ready to be able to
           Disable Rx Interrupts, and disable Rx
        UART_EndRxTransfer(huart);

        /* Disable the UART DMA Rx request if ena
        if (HAL_IS_BIT_SET(huart->Instance->CR3,
        {
          ATOMIC_CLEAR_BIT(huart->Instance->CR3,

          /* Abort the UART DMA Rx channel */
          if (huart->hdmarx != NULL)
          {
            /* Set the UART DMA Abort callback :
               will lead to call HAL_UART_ErrorCa
            huart->hdmarx->XferAbortCallback = UA
            if (HAL_DMA_Abort_IT(huart->hdmarx) !=
            {
              /* Call Directly XferAbortCallback
              huart->hdmarx->XferAbortCallback(hu
            }
          }
```

```
          else
          {
            /* Call user error callback */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
            /*Call registered error callback*/
            huart->ErrorCallback(huart);
#else
            /*Call legacy weak error callback*/
            HAL_UART_ErrorCallback(huart);
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
          }
        }
        else
        {
          /* Call user error callback */
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
          /*Call registered error callback*/
          huart->ErrorCallback(huart);
#else
          /*Call legacy weak error callback*/
          HAL_UART_ErrorCallback(huart);
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
        }
      }
      else
      {
        /* Non Blocking error : transfer could go
           Error is notified to user through user
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Call registered error callback*/
        huart->ErrorCallback(huart);
#else
        /*Call legacy weak error callback*/
        HAL_UART_ErrorCallback(huart);
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */

        huart->ErrorCode = HAL_UART_ERROR_NONE;
        // 핸들러 에러 초기화
```

```
      }
    }
    return;
  } /* End if some error occurs */

  /* Check current reception Mode :
     If Reception till IDLE event has been selecte
  if ((huart->ReceptionType == HAL_UART_RECEPTION_
      && ((isrflags & USART_SR_IDLE) != 0U)
      && ((cr1its & USART_SR_IDLE) != 0U))
  {
    __HAL_UART_CLEAR_IDLEFLAG(huart);

    /* Check if DMA mode is enabled in UART */
    if (HAL_IS_BIT_SET(huart->Instance->CR3, USART
    {
      /* DMA mode enabled */
      /* Check received length : If all expected d
         (DMA cplt callback will be called).
         Otherwise, if at least one data has alre
      uint16_t nb_remaining_rx_data = (uint16_t) _
      if ((nb_remaining_rx_data > 0U)
          && (nb_remaining_rx_data < huart->RxXfe
      {
        /* Reception is not complete */
        huart->RxXferCount = nb_remaining_rx_data

        /* In Normal mode, end DMA xfer and HAL UA
        if (huart->hdmarx->Init.Mode != DMA_CIRCU
        {
          /* Disable PE and ERR (Frame error, noi
          ATOMIC_CLEAR_BIT(huart->Instance->CR1, U
          ATOMIC_CLEAR_BIT(huart->Instance->CR3, U

          /* Disable the DMA transfer for the rec
             in the UART CR3 register */
          ATOMIC_CLEAR_BIT(huart->Instance->CR3, U
```

```
          /* At end of Rx process, restore huart-
          huart->RxState = HAL_UART_STATE_READY;
          huart->ReceptionType = HAL_UART_RECEPTI

          ATOMIC_CLEAR_BIT(huart->Instance->CR1,

          /* Last bytes received, so no need as t
          (void)HAL_DMA_Abort(huart->hdmarx);
        }

        /* Initialize type of RxEvent that corres
        In this case, Rx Event type is Idle Event
        huart->RxEventType = HAL_UART_RXEVENT_IDL

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Call registered Rx Event callback*/
        huart->RxEventCallback(huart, (huart->RxX
#else
        /*Call legacy weak Rx Event callback*/
        HAL_UARTEx_RxEventCallback(huart, (huart-
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
      }
      return;
    }
    else
    {
      /* DMA mode not enabled */
      /* Check received length : If all expected
         Otherwise, if at least one data has alre
      uint16_t nb_rx_data = huart->RxXferSize - h
      if ((huart->RxXferCount > 0U)
          && (nb_rx_data > 0U))
      {
        /* Disable the UART Parity Error Interrup
        ATOMIC_CLEAR_BIT(huart->Instance->CR1, (U

        /* Disable the UART Error Interrupt: (Fra
        ATOMIC_CLEAR_BIT(huart->Instance->CR3, USA
```

```
        /* Rx process is completed, restore huart
        huart->RxState = HAL_UART_STATE_READY;
        huart->ReceptionType = HAL_UART_RECEPTION_

        ATOMIC_CLEAR_BIT(huart->Instance->CR1, USA

        /* Initialize type of RxEvent that corres
           In this case, Rx Event type is Idle Ev
        huart->RxEventType = HAL_UART_RXEVENT_IDLE

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Call registered Rx complete callback*/
        huart->RxEventCallback(huart, nb_rx_data)
#else
        /*Call legacy weak Rx Event callback*/
        HAL_UARTEx_RxEventCallback(huart, nb_rx_da
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
      }
      return;
    }
  }

  /* UART in mode Transmitter ------------------
  if (((isrflags & USART_SR_TXE) != RESET) && ((c
  {
    UART_Transmit_IT(huart);
    return;
  }

  /* UART in mode Transmitter end --------------
  if (((isrflags & USART_SR_TC) != RESET) && ((cr
  {
    UART_EndTransmit_IT(huart);
    return;
  }
}
```

▼ UART_Receive_IT(huart) **in stm32f1xx_hal_uart.c >> 실질적인 리시브 과정. 및 IT Eable 해제**;

```
static HAL_StatusTypeDef UART_Receive_IT(UART_
{
  uint8_t  *pdata8bits;
  uint16_t *pdata16bits;

  /* Check that a Rx process is ongoing */
  if (huart->RxState == HAL_UART_STATE_BUSY_RX
  { // 이부분은 아까 HAL_UART_Receive_IT 진행할때 E
    if ((huart->Init.WordLength == UART_WORDLE
    { // 9B && None parity일때 16읽는부분 우린 아님
      pdata8bits  = NULL;
      pdata16bits = (uint16_t *) huart->pRxBuf
      *pdata16bits = (uint16_t)(huart->Instanc
      huart->pRxBuffPtr += 2U;
    }
    else
    { // 여기 실행됨.
      pdata8bits = (uint8_t *) huart->pRxBuffP
      pdata16bits  = NULL;

      if ((huart->Init.WordLength == UART_WORD
      {
        *pdata8bits = (uint8_t)(huart->Instanc
        // parity 없는 8bit read.
      }
      else
      {
        *pdata8bits = (uint8_t)(huart->Instanc
      }
      huart->pRxBuffPtr += 1U;
    }

    if (--huart->RxXferCount == 0U)
    { // 이제 저장된 버퍼 포인터는 올라가고 --huart->
      //USARTx_IRQHandler >> HAL_UART_IRQHandl
```

```
        // 아래는 만약 count가 0이라면 >> 한패킷을 전부
        // Interrupt들 Disable해줌.
        /* Disable the UART Data Register not em
        __HAL_UART_DISABLE_IT(huart, UART_IT_RXN

        /* Disable the UART Parity Error Interru
        __HAL_UART_DISABLE_IT(huart, UART_IT_PE)

        /* Disable the UART Error Interrupt: (Fr
        __HAL_UART_DISABLE_IT(huart, UART_IT_ERR

        /* Rx process is completed, restore huar
        huart->RxState = HAL_UART_STATE_READY;
              // UART STATE READY로 변경

        /* Initialize type of RxEvent to Transfe
        huart->RxEventType = HAL_UART_RXEVENT_TC
              // RxEventType  Transfer Complete
        /* Check current reception Mode :
           If Reception till IDLE event has been
        if (huart->ReceptionType == HAL_UART_REC
        { // TOIDLE TYPE인지 확인 아니라면 아래 두번째
          /* Set reception type to Standard */
          huart->ReceptionType = HAL_UART_RECEPT

          /* Disable IDLE interrupt */
          ATOMIC_CLEAR_BIT(huart->Instance->CR1,

          /* Check if IDLE flag is set */
          if (__HAL_UART_GET_FLAG(huart, UART_FL
          {
            /* Clear IDLE flag in ISR */
            __HAL_UART_CLEAR_IDLEFLAG(huart);
          }

#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Call registered Rx Event callback*/
        huart->RxEventCallback(huart, huart->R
```

```
#else
        /*Call legacy weak Rx Event callback*/
        HAL_UARTEx_RxEventCallback(huart, huar
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
      }
      else
      {
        /* Standard reception API called */

        // 여기로 이동해서 USE_HAL_UART_REGISTER_
#if (USE_HAL_UART_REGISTER_CALLBACKS == 1)
        /*Call registered Rx complete callback
        huart->RxCpltCallback(huart);
#else
        /*Call legacy weak Rx complete callbac
        HAL_UART_RxCpltCallback(huart);
#endif /* USE_HAL_UART_REGISTER_CALLBACKS */
      }

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

▼ 각 오류 정리

▼ 페리티 오류 `USART_SR_PE`

▼ 프레임 오류 `USART_SR_FE`

프레이밍 오류

프레이밍 오류는 다음과 같은 경우에 감지됩니다:

- 스톱 비트가 예상 시간에 수신되지 않거나, 비동기화 또는 과도한 노이즈가 발생한 경우.

프레이밍 오류가 감지되면:

- FE 비트는 하드웨어에 의해 설정됩니다.
- 유효하지 않은 데이터가 시프트 레지스터에서 USART_DR 레지스터로 전송됩니다.
- 단일 바이트 통신의 경우 인터럽트가 생성되지 않습니다. 그러나 이 비트는 RXNE 비트와 동시에 상승하며, RXNE 비트 자체는 인터럽트를 생성합니다. 다중 버퍼 통신의 경우 USART_CR3 레지스터의 EIE 비트가 설정되어 있으면 인터럽트가 발생합니다.
- FE 비트는 USART_SR 레지스터를 읽은 후 USART_DR 레지스터를 읽음으로써 리셋됩니다.

▼ 오버런 오류 `USART_SR_ORE`

오버런 오류

오버런 오류는 RXNE가 리셋되지 않은 상태에서 문자가 수신될 때 발생합니다. RXNE 비트가 클리어될 때까지는 시프트 레지스터에서 RDR 레지스터로 데이터가 전송될 수 없습니다.

RXNE 플래그는 매번 바이트가 수신될 때마다 설정됩니다. 오버런 오류는 다음 데이터가 수신될 때 RXNE 플래그가 설정되어 있거나 이전 DMA 요청이 처리되지 않았을 때 발생합니다. 오버런 오류가 발생하면:

- ORE 비트가 설정됩니다.
- RDR의 내용은 손실되지 않습니다. USART_DR을 읽으면 이전 데이터에 접근할 수 있습니다.
- 시프트 레지스터는 덮어씌워집니다. 그 이후에 오버런 중에 수신된 데이터는 손실됩니다.
- RXNEIE 비트가 설정되어 있거나 EIE 및 DMAR 비트가 모두 설정되어 있으면 인터럽트가 생성됩니다.
- ORE 비트는 USART_SR 레지스터를 읽은 후 USART_DR 레지스터를 읽으면 리셋됩니다.

참고: ORE 비트가 설정되면 적어도 1개의 데이터가 손실되었음을 나타냅니다. 두 가지 경우가 있습니다:

- RXNE=1인 경우, 마지막 유효 데이터가 수신 레지스터 RDR에 저장되어 읽을 수 있습니다.
- RXNE=0인 경우, 마지막 유효 데이터가 이미 읽혔음을 의미하므로 RDR에서 읽을 데이터가 없습니다. 이 경우는 마지막 유효 데이터가 RDR에서 읽히는 동시에 새로운(손실된) 데이터가 수신될 때 발생할 수 있습니다. 또한, USART_SR 레지스터 읽기 접근과 USART_DR 읽기 접근 사이에 새로운 데이터가 수신되는 동안 발생할 수 있습니다.

▼ 잡음 오류 `USART_SR_NE`

노이즈 오류

유효한 수신 데이터와 노이즈를 구별하기 위해 데이터 복구에 오버샘플링 기술이 사용됩니다(동기 모드 제외).

Figure 284. Data sampling for noise detection

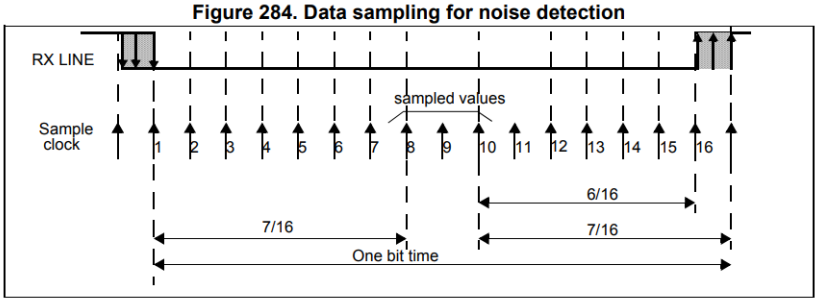


Table 191. Noise detection from sampled data

Sampled value	NE status	Received bit value	Data validity
000	0	0	Valid
001	1	0	Not Valid
010	1	0	Not Valid
011	1	1	Not Valid
100	1	0	Not Valid
101	1	1	Not Valid
110	1	1	Not Valid
111	0	1	Valid

그니까 샘플링 데이터가 저렇게 동일하지 않다면 NE SET되는거임.

프레임에서 노이즈가 감지되면:

- NE 비트는 RXNE 비트의 상승 엣지에서 설정됩니다.
- 유효하지 않은 데이터가 시프트 레지스터에서 USART_DR 레지스터로 전송됩니다.
- 단일 바이트 통신의 경우 인터럽트가 생성되지 않습니다. 그러나 이 비트는 RXNE 비트와 동시에 상승하며, RXNE 비트 자체는 인터럽트를 생성합니다. 다중 버퍼 통신의 경우 USART_CR3 레지스터의 EIE 비트가 설정되어 있으면 인터럽트가 발생합니다.
- NE 비트는 USART_SR 레지스터를 읽은 후 USART_DR 레지스터를 읽음으로써 리셋됩니다.

▼ void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) **in stm32f1xx_it.c**

사용자가 재정의(__weak로 되어있음.) 해서 사용

```
__weak void HAL_UART_RxCpltCallback(UART_HandleTy
{
  /* Prevent unused argument(s) compilation warni
  UNUSED(huart);
  /* NOTE: This function should not be modified, 
           the HAL_UART_RxCpltCallback could be i
   */
}
```

▼ Interrupt with Ling buffer

▼ 개요

수신 데이터 처리가 오래걸리는 경우 또는 수신 데이터가 큰경우 위에서 사용자 정의로 처리되는 HAL_UART_RxCpltCallback과 링버퍼 를 이용해서 보다 신뢰성 있는 수신이 가능

UART 수신에 링버퍼를 사용하는 이유 :

1. 일반적으로 통신 데이터는 다수의 바이트로 구성된 패킷 형태를 가진다.
2. 패킷의 구조는 정의된 프로토콜에 따라 다르나 완전한 패킷을 수신한 후에 처리해야 하는 점은 동일하다.
3. 패킷을 처리하는데 소요되는 시간은 패킷의 구조와 역할에 따라 다르다.
4. 링버퍼를 사용하지 않는 경우 UART 수신 인터럽트 함수에서 패킷을 처리하게 되는데 만약 패킷을 처리하는데 너무 많은 시간이 소요될 경우 시스템 성능을 저하시키는 원인이 되며 아래와 같은 문제가 발생할 수 있다.
   - 다른 인터럽트를 제 시간에 처리하지 못할 수 있다.(우선순위 설정에 따라 다르다.)
   - 패킷을 연속적으로 수신하는 경우 이전 패킷을 처리하는 동안 다음 패킷의 일부가 수신되지 않을 수 있다.
   - 이 때 다음 패킷은 완전한 형태를 지닐 수 없으므로 처리하지 못하고 버려지거나 잘못된 정보로 처리된다.
5. 링버퍼를 사용하면 패킷 수신과 패킷 처리 부분을 분리할 수 있으므로 위와 같은 문제를 방지할 수 있다.

▼ 링버퍼 사용 이유 추가 설명

HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_Data, 1);에서 맨뒤를 100으로 놓고 100바이트 문자열로 100씩 패킷 형태로 받으면 100바이트 받고 인터럽트가 걸리게 되지만 이 100바이트를 처리하는데 시간이 오래걸린다면 바로 뒤에 데이터들이 유실될 수 있다는거임. 이거 진짜 재밌고 신기하네. >> 그니까 HAL_UART_RxCpltCallback에서 100바이트를 처리할때 HAL_UART_Receive_IT를 다시 호출해주지 않았으니까 RXNEIE 가 0일꺼고 새로운 데이터가 들어와도 인터럽트가 걸리질않음.. 만약에 HAL_UART_Receive_IT를 제일 처음에 호출한다 하더라도 DR에서 긁어온 데이터가 덮어 씌워짐. 이외에도 다른 인터럽트들이 이 처리과정에서 누락될 수도 있음.

▼ 정리된 코드 Main.c 및 MyUART.c 설명

▼ 코드 Main.c

```
/* 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 b
  * in the root directory of this software component
  * If no LICENSE file comes with this software, it
  *
  ************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------
#include "main.h"

/* Private includes ----------------------------------
/* USER CODE BEGIN Includes */
/* added part 1016*/
#include <string.h>
#include "MyUART.h"
/* added part 1016*/
/* 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;
```

```
/* USER CODE BEGIN PV */
/* added part 1016 - extern volatile*/
extern volatile char Rx_Data;
/* added part 1016*/
/* USER CODE END PV */

/* Private function prototypes -----------------------
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_NVIC_Init(void);
/* 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

    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();
HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_Data,
/* Initialize interrupts */
MX_NVIC_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
int count=0;
char inputString[100];
memset(inputString, 0, sizeof(inputString));

sprintf(inputString, "Hello Cortex-M3\n");
UART_TxString(&huart2, (uint8_t *)inputString);

while (1)
{
    if (UART_RxCheck())
    {

        HAL_Delay(3);
        sprintf(inputString, "Count: %d ", count++
        UART_TxString(&huart2, (uint8_t *)inputStr
        UART_RxString((uint8_t *)inputString);
```

```
          UART_TxString(&huart2, (uint8_t *)inputStr
          //printf("count: %d\n", count++);
      }

      /* 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 t
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORT
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICAL
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_O
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|R
                              |RCC_CLOCKTYPE_PCLK1|R
```

```
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_
  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_
  {
    Error_Handler();
  }
}

/**
  * @brief NVIC Configuration.
  * @retval None
  */
static void MX_NVIC_Init(void)
{
  /* USART2_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(USART2_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(USART2_IRQn);
}

/**
  * @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 */
/**
 * @brief Retargets the C library printf function to
 * @param None
```

```
 * @retval None
 */
/* USER CODE END 4 */

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

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

▼ 코드 MyUART.c

```
#include "stm32f1xx_hal.h"
#include "MyUART.h"
#include <string.h>

extern UART_HandleTypeDef huart2;

#define LENGTH_RX_BUFFER    100

volatile char Rx_Data;
volatile uint8_t uart2Flag;
volatile unsigned char rx_buffer[LENGTH_RX_BUFFER];
volatile unsigned char rx_head=0, rx_tail=0;

//void HAL_UART_TxCpltCallback(UART_HandleTypeDef *h
//{
//  HAL_UART_Transmit(&huart2, (uint8_t *)&Rx_Data, 
//}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *hua
{
    if (huart->Instance == USART2)
    {
        if( (rx_head+1==rx_tail) || ((rx_head==LENGT
        {
            volatile char temp = Rx_Data; // rx_buff
        }
        else
        {
            rx_buffer[rx_head] = Rx_Data;
            rx_head = (rx_head==LENGTH_RX_BUFFER-1) 
        }
        HAL_UART_Receive_IT(&huart2, (uint8_t *)&Rx_
    }
}

int UART_RxCheck(void)
{
    return (rx_head != rx_tail) ? 1 : 0;
```

```
}

uint8_t UART_RxChar(void)
{
    unsigned char data;

    while( rx_head==rx_tail );  // 수신 문자가 없음

    data = rx_buffer[rx_tail];
    rx_tail = (rx_tail==LENGTH_RX_BUFFER-1) ? 0 : rx

    return data;
}

void UART_RxString(uint8_t *RxBuff)
{
    //while (UART_RxCheck())
    char getData=0;
    uint16_t rxCount=0;

    getData = UART_RxChar();
    while (!((getData == '\n') || (getData == '\r'))
    {
        *RxBuff = getData;
        RxBuff++;
        rxCount++;
        getData = UART_RxChar();

        if(rxCount >= LENGTH_RX_BUFFER-3) break;
    }
    *RxBuff = '\n';
    RxBuff++;
    *RxBuff = '\0'; // 문자열 끝 NULL 추가.
}

void UART_TxChar(UART_HandleTypeDef *huart, uint8_t
{
    while((__HAL_UART_GET_FLAG(huart, UART_FLAG_TXE)
```

```
    huart->Instance->DR = txData;
}

void UART_TxString(UART_HandleTypeDef *huart, uint8_
{
    while(*pData)
    {
        UART_TxChar(huart, *pData++);
    }
}
```