การทดลองที่ 3 การใช้งาน UART

วัตถุประสงค์

1) เข้าใจการทำงานของ UART
2) สามารถเขียนโปรแกรมเพื่อรับส่งข้อความผ่านพอร์ต UART

1. UART / USART

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) เป็นพอร์ตสื่อสารซึ่งสามารถใช้งานได้ทั้งแบบ Asynchronous (UART) และ Synchronous (USART) ไมโครคอนโทรลเลอร์หมายเลข STM32F411 มีพอร์ต UART จำนวน 3 พอร์ต ได้แก่ UART1, UART2 และ UART6 แต่บอร์ด Nucleo-411RE ไม่มีไอซี MAX232 จึงไม่สามารถใช้งาน UART ผ่าน RS232 ได้ ทางผู้ผลิตได้เชื่อมต่อ UART2 เข้าไอซี ST-Link บนบอร์ดและเมื่อร่วมกับไดรเวอร์ ST-Link ที่เครื่องคอมพิวเตอร์จึงสามารถสร้าง Virtual Communication Port เพื่อสื่อสารข้อมูลระหว่างไมโครคอนโทรลเลอร์กับเครื่องคอมพิวเตอร์ผ่าน UART ได้

แต่ละขาของไอซีนั้นสามารถกำหนดหน้าที่การทำงานได้สูงสุด 3 รูปแบบ คือ

1) Main Function หรือ After Reset
2) Alternate Function
3) Additional Function

บอร์ด Nucleo411RE ได้เชื่อมต่อขา PA2 และ PA3 ซึ่งสามารถทำหน้าที่ UART2_Tx และ UART2_Rx ตามลำดับเข้ากับไอซี ST-Link เพื่อทำเป็นพอร์ต UART เสมือน สำหรับเชื่อมต่อกับเครื่องคอมพิวเตอร์ เมื่อไมโครคอนโทรลเลอร์เริ่มต้นการทำงานขา 16 และ 17 จะทำหน้าที่เป็น GPIO ได้แก่ PA2 และ PA3 ตามลำดับ หากต้องการเปลี่ยนให้ขา 16 และ 17 ทำหน้าที่เป็น UART2 จะต้องเขียนโปรแกรมเพื่อกำหนดให้ทั้งสองขาดังกล่าวทำหน้าที่ Alternate Function หากต้องการใช้ UART1 และ UART6 ของไมโครคอนโทรลเลอร์เป็นพอร์ตสื่อสารแบบ UART ด้วยมาตรฐาน RS232 ต้องต่อไอซีแปลงระดับแรงดันไฟฟ้าเสียก่อน ขาไมโครคอนโทรลเลอร์ที่สามารถทำหน้าที่ UART ได้แสดงดังตารางที่ 1.1

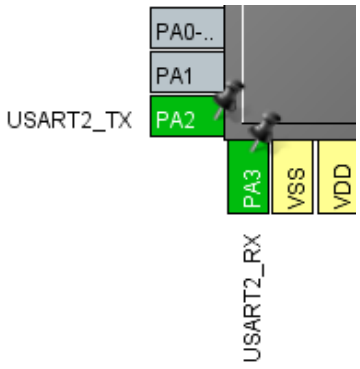
ตารางที่ 1.1 แสดงการทำงานของขาไมโครคอนโทรลเลอร์ที่เกี่ยวกับพอร์ต UART1 และ UART2

Pin NO.	Main Function (After Reset)	Alternate Functions	Additional Function
16	PA2	TIM2_CH3, TIM5_CH3, TIM9_CH1, I2S2_CKIN, **USART2_TX**, EVENTOUT	ADC1_2
17	PA3	TIM2_CH4, TIM5_CH4, TIM9_CH2, I2S2_MCK, **USART2_RX**, EVENTOUT	ADC1_3

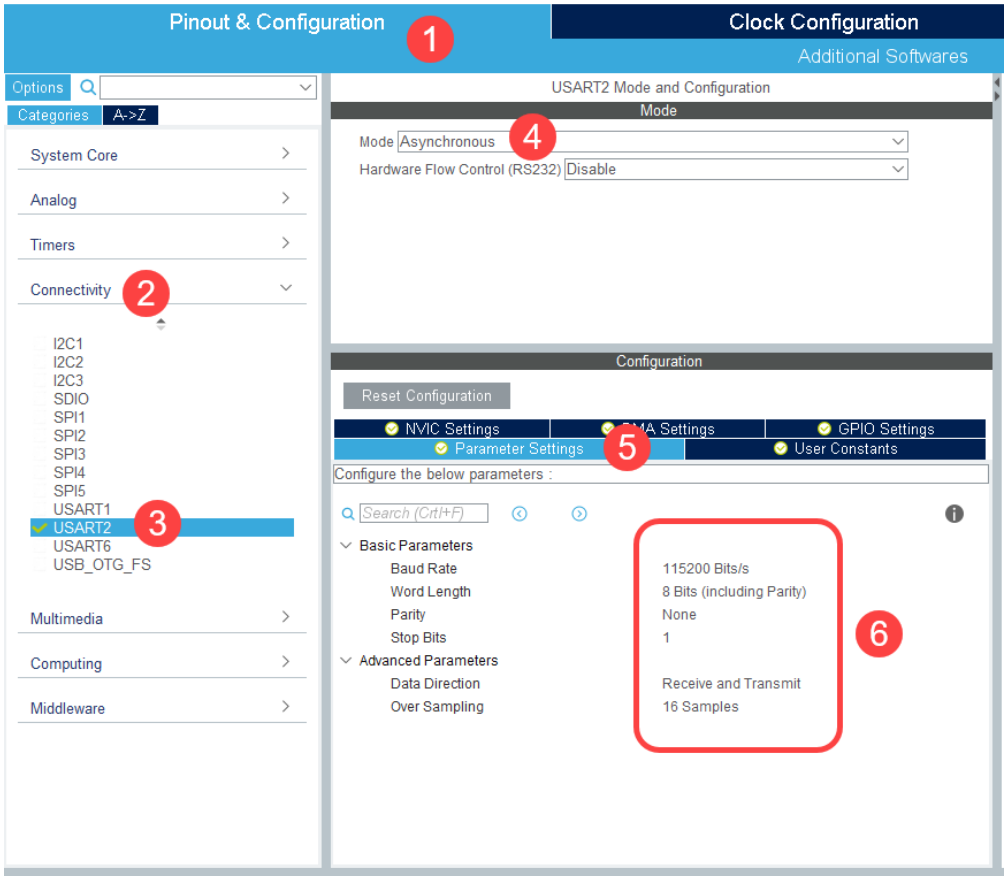
Pin NO.	Main Function (After Reset)	Alternate Functions	Additional Function
37	PC6	TIM3_CH1, I2S2_MCK, **USART6_TX**, SDIO_D6, EVENTOUT	-
38	PC7	TIM3_CH2, SPI2_SCK/I2S2_CK, I2S3_MCK, **USART6_RX**, SDIO_D7, EVENTOUT	-
42	PA9	TIM1_CH2, I2C3_SMBA, **USART1_TX**, USB_FS_VBUS, SDIO_D2, EVENTOUT	OTG_FS_VBUS
43	PA10	TIM1_CH3, SPI5_MOSI/I2S5_SD, **USART1_RX**, USB_FS_ID, EVENTOUT	-

2. การตั้งค่า UART

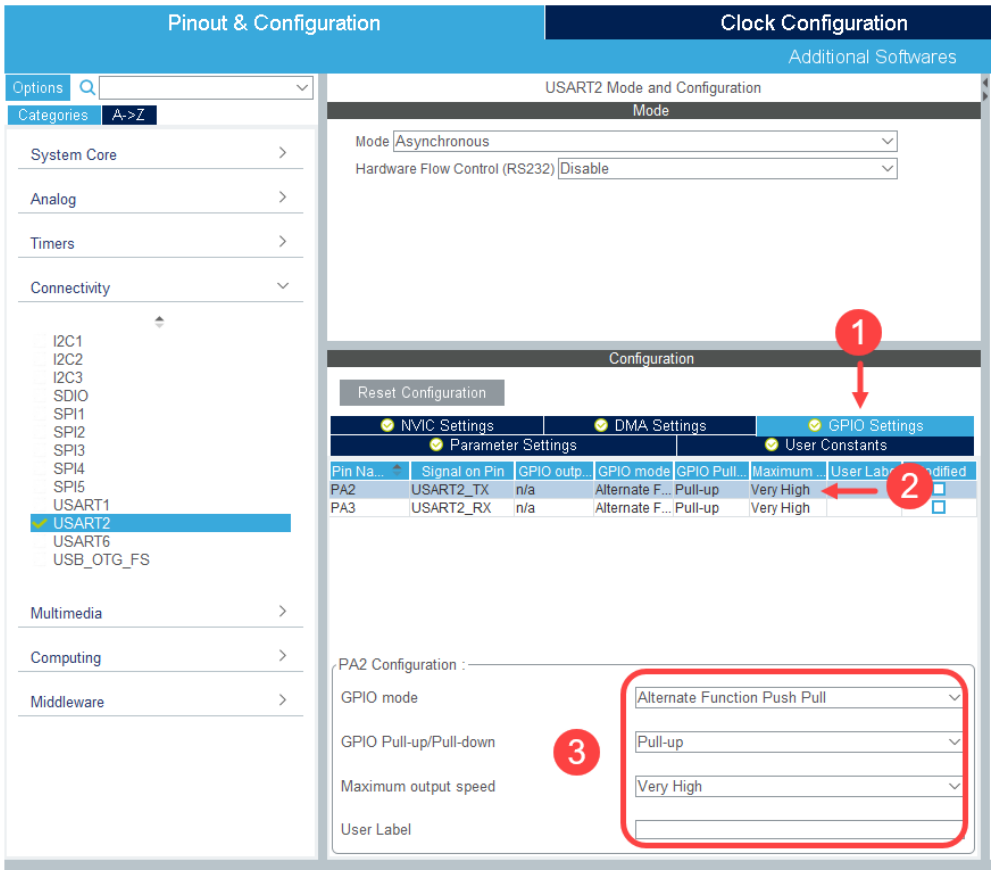
การจะใช้งานพอร์ต UART บนบอร์ดการทดลองผ่าน Virtual Com Port ต้องตั้งค่าขาของไมโครคอนโทรลเลอร์ที่เชื่อมต่ออยู่กับไอซี ST-Link ด้วยโปรแกรม STM32CubeMX ให้ถูกต้อง ดังรูปที่ 2.1 และ รูปที่ 2.2



รูปที่ 2.1 แสดงการตั้งค่าขา PA2 และ PA3 ให้ทำหน้าที่ UART2



(a)



(b)

รูปที่ 2.2 แสดงการตั้งค่าพอร์ต UART2

3. อธิบายการทำงาน

การเริ่มต้นใช้งาน UART ต้องกำหนดค่าต่างๆ ที่เกี่ยวข้องกับ UART เสียก่อน เช่น

- Baud rate (ความเร็วในการรับส่งข้อมูล) เช่น 115200, 57600 หรือ 38400 Bits/sec
- จำนวนบิตใน 1 เฟรมว่าจะเป็น 8 หรือ 9 บิต (รวม parity bit แล้ว)
- จำนวน Stop bit
- Parity ที่จะใช้ตรวจสอบความถูกต้องของการรับส่งข้อมูล ได้แก่ even/odd/no parity
- การเลือกว่าจะใช้โปรโตคอลเกี่ยวกับ hardware flow control หรือไม่
- โหมดการทำงานว่าจะให้รับหรือส่งข้อมูล หรือทั้งรับและส่ง เป็นต้น

จากนั้นทำการตั้งค่า Alternate Function ให้ขาไมโครคอนโทรลเลอร์จากที่ทำหน้าที่เป็น GPIO PA2 และ PA3 ให้ทำหน้าที่เป็น USART2_TX และ USART2_RX ตามลำดับ

การตั้งค่าการทำงาน UART จากโปรแกรม STM32CubeMX จะถูกกำหนดไว้ที่ไฟล์ 3 ไฟล์ ได้แก่

- `usart.c`
- `gpio.c`
- `main.c`

ไฟล์ `usart.c`

เป็นไฟล์ที่รวมการตั้งค่าขา GPIO ให้ทำหน้าที่ Alternate function และการตั้งค่า UART

`Global variables`

- จะทำการประกาศตัวแปร `huart2` เพื่อที่จะใช้เป็นตัวแทนของโมดูล UART2 ดังรูปที่ 3.1

```
UART_HandleTypeDef huart2;
```

- และเพื่อให้สามารถใช้ตัวแปร `huart2` นี้ในไฟล์อื่นๆ เช่น `main.c` ได้ จึงได้ทำการประกาศตัวแปรแบบ extern ไว้ในไฟล์ `usart.h`

```
extern UART_HandleTypeDef huart2;
```

ฟังก์ชัน `MX_USART2_UART_Init()`

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า UART2 บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับค่าที่กำหนดไว้ในโปรแกรม ดังรูปที่ 3.1
- เริ่มต้นด้วยการกำหนดให้ตัวแปร `huart2` เป็นตัวแทนของ UART2

```
huart2.Instance = USART2;
```

- แล้วทำการตั้งค่าการทำงานของ USART ดังนี้
  - Baud rate = 115,200 bits/second
  - ใช้ 8 บิตใน 1 เฟรม
  - ใช้ 1 stop bit
  - ไม่ใช้ Parity bit
  - กำหนดให้ทำงานทั้งรับและส่งข้อมูล
  - ไม่ใช้ Hardware Flow Control

- กำหนดให้ทำการ Oversampling 16 เท่า

```
huart2.Init.BaudRate      = 115200;
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;
HAL_UART_Init(&huart2);
```

```c
UART_HandleTypeDef huart2;

/* USART2 init function */

void MX_USART2_UART_Init(void)
{

  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  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();
  }

}
```

รูปที่ 3.1 ฟังก์ชันตั้งค่า UART2

ฟังก์ชัน HAL_UART_MspInit()

- เป็นฟังก์ชันที่ใช้ในการตั้งค่าขา GPIO ที่จะนำมาใช้เป็นขา TX และ RX ของ UART ดังรูปที่ 3.2
- เริ่มต้นการตั้งค่า UART2 ด้วยการจ่ายสัญญาณนาฬิกาให้กับโมดูล UART และ GPIOA

```
__HAL_RCC_USART2_CLK_ENABLE();
__HAL_RCC_GPIOA_CLK_ENABLE();
```

- กำหนดให้ PA2 และ PA3 ทำหน้าที่ UART โดยจะตั้งค่าให้เป็น alternate function แบบพุชพูลและ Pull up ที่ Very High Speed

```
GPIO_InitStruct.Pin       = GPIO_PIN_2|GPIO_PIN_3;
GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
GPIO_InitStruct.Pull      = GPIO_PULLUP;
GPIO_InitStruct.Speed     =  GPIO_SPEED_FREQ_VERY_HIGH;
GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```c
void HAL_UART_MspInit(UART_HandleTypeDef* uartHandle)
{

  GPIO_InitTypeDef GPIO_InitStruct = {0};
  if(uartHandle->Instance==USART2)
  {
  /* USER CODE BEGIN USART2_MspInit 0 */

  /* USER CODE END USART2_MspInit 0 */
    /* USART2 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_PIN_3;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
    GPIO_InitStruct.Alternate = GPIO_AF7_USART2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /* USER CODE BEGIN USART2_MspInit 1 */

  /* USER CODE END USART2_MspInit 1 */
  }
}
```

รูปที่ 3.2 ฟังก์ชัน `HAL_UART_MspInit()`สำหรับตั้งค่าให้ PA2 และ PA3 ทำหน้าที่ UART2

ไฟล์ `gpio.c`

ฟังก์ชัน `MX_GPIO_Init()`

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อจ่ายสัญญาณนาฬิกาให้กับ GPIO พอร์ต A ด้วยคำสั่ง

  `__HAL_GPIOA_CLK_ENABLE();`

ไฟล์ `main.c`

ฟังก์ชัน `main()`

- เริ่มต้นการทำงานด้วยฟังก์ชันเพื่อตั้งค่าโมดูลต่างๆ ได้แก่
  - `HAL_Init()`
  - `SystemClock_Config()`
  - `MX_GPIO_Init()`
  - `MX_USART2_UART_Init()`

4. การส่งและรับข้อมูลผ่าน UART

ก่อนการส่งหรือการรับข้อมูลผ่าน UART จำเป็นต้องตรวจสอบสถานะการทำงานของโมดูล UART เสียก่อนว่ามีสถานะที่พร้อมรับข้อมูลเพื่อส่งออก (send) หรือว่าพร้อมที่จะให้อ่านข้อมูลที่รับเข้ามาหรือไม่ (receive) ด้วยการตรวจสอบบางบิตในรีจิสเตอร์ของโมดูล (flag) เช่น ตรวจสอบแฟลก Transmission Control (TC) เพื่อตรวจสอบว่าการส่งข้อมูลก่อนหน้าดำเนินการเสร็จสิ้นหรือยัง ถ้ายังไม่เสร็จ UART ก็ไม่สามารถส่งข้อมูลใหม่ได้ หรือตรวจสอบแฟลก Read Data

Register Not Empty (RXNE) เพื่อดูว่าข้อมูลที่รับเข้ามาที่ละบิตได้ถูกเลื่อนบิต (shift) เข้ามาในบัฟเฟอร์ (buffer) จนครบแล้วหรือไม่ โดยมีมาโครที่เกี่ยวข้องในการรับส่งข้อมูลดังนี้

มาโคร `__HAL_UART_GET_FLAG (__HANDLE__, __FLAG__)`

- ใช้เพื่ออ่านค่าแฟลกต่างๆ ของ UART
- `__HANDLE__` : ระบุ UART ที่ต้องการ เช่น `&huart2` เป็นต้น
- `__FLAG__` : ระบุแฟลกที่ต้องการทราบค่า เช่น `UART_FLAG_RXNE` และ `UART_FLAG_TC` เป็นต้น

ฟังก์ชัน `HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)`

- ใช้เพื่อส่งข้อมูลที่ต้องการผ่านทาง UART
- `huart` : ระบุ UART ที่จะใช้ส่งข้อมูล เช่น `&huart2` เป็นต้น
- `pData` : คือ Pointer ที่ชี้ไปยังตำแหน่งเริ่มต้นของข้อมูลที่จะส่ง
- `Size` : ความยาวของข้อมูลที่จะส่งในหน่วย Byte
- `Timeout` : ระยะเวลาที่ฟังก์ชันสามารถใช้เพื่อส่งข้อมูลมีหน่วยเป็น millisecond หากไม่สามารถส่งข้อมูลเสร็จภายในระยะเวลาที่กำหนด ฟังก์ชันจะส่งค่ากลับเป็น `HAL_TIMEOUT`
- ฟังก์ชันจะส่งค่ากลับเป็นสถานะการทำงาน เช่น `HAL_OK` เมื่อส่งข้อมูลสำเร็จ หรือ `HAL_BUSY` ถ้าโมดูล UART ไม่พร้อมทำงาน

```
char str[] = “Hello, World!!\n\r”;

while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET){}
HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);

HAL_Delay(500);
```

รูปที่ 4.1 ตัวอย่างการส่งข้อมูลผ่าน UART

รูปที่ 4.1 แสดงตัวอย่างการส่งข้อมูลของตัวแปรข้อความ str ผ่าน UART ตัวแปร str ถูกปิดท้ายข้อความ “`Hello, World!!\n\r`” ด้วยตัวอักษรพิเศษ 2 ตัว ได้แก่ ตัวอักษร Line Feed หรือ ‘`\n`’ (รหัส ASCII 0xA) ใช้สำหรับขึ้นบรรทัดใหม่โดยเคอร์เซอร์จะอยู่ตำแหน่งเดียวกันกับบรรทัดบนและตัวอักษร Carriage Return หรือ ‘`\r`’ (รหัส ASCII 0xD) ใช้สำหรับเลื่อนเคอร์เซอร์ให้กลับสู่ตำแหน่งแรกของบรรทัดปัจจุบัน ก่อนการเรียกใช้ฟังก์ชันเพื่อส่งข้อมูลต้องรอให้ UART พร้อมรับข้อมูลใหม่ที่จะส่งออกไปด้วยการวนลูปรอจนกระทั่งแฟลก TC ถูกเซต

ฟังก์ชัน `HAL_StatusTypeDef HAL_UART_Receive (UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout)`

- ใช้เพื่ออ่านข้อมูลที่รับเข้ามาทาง UART
- `huart` : ระบุ UART ที่จะใช้ส่งข้อมูล เช่น `&huart2` เป็นต้น
- `pData` : คือ Pointer ที่ชี้ไปยังตำแหน่งที่อยู่ของตัวแปรที่ใช้รับข้อมูล
- `Size` : ระบุความยาวของข้อมูลที่จะรับในหน่วย Byte

- `Timeout` : ระยะเวลาที่ฟังก์ชันสามารถใช้เพื่ออ่านข้อมูลที่ได้รับเข้ามาจากบัฟเฟอร์มีหน่วยเป็น millisecond หากไม่สามารถทำงานเสร็จภายในระยะเวลาที่กำหนด ฟังก์ชันจะส่งค่ากลับเป็น `HAL_TIMEOUT`
- โดยตัวฟังก์ชันจะรีเทิร์นค่าออกเป็นสถานะการทำงาน เช่น `HAL_OK` หรือ `HAL_BUSY`

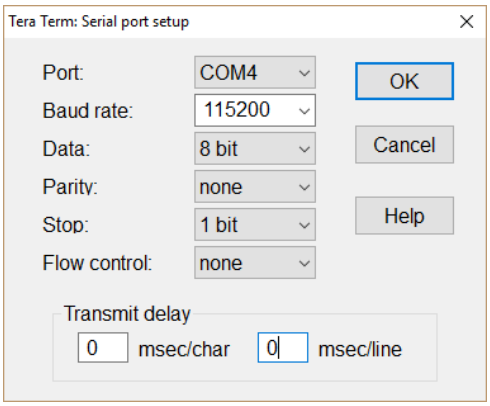
```
char ch1 = 'A';

while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_RXNE)== RESET){}
HAL_UART_Receive(&huart2, (uint8_t*)&ch1, 1, 1000);
```

รูปที่ 4.2 ตัวอย่างการรับข้อมูลผ่าน UART

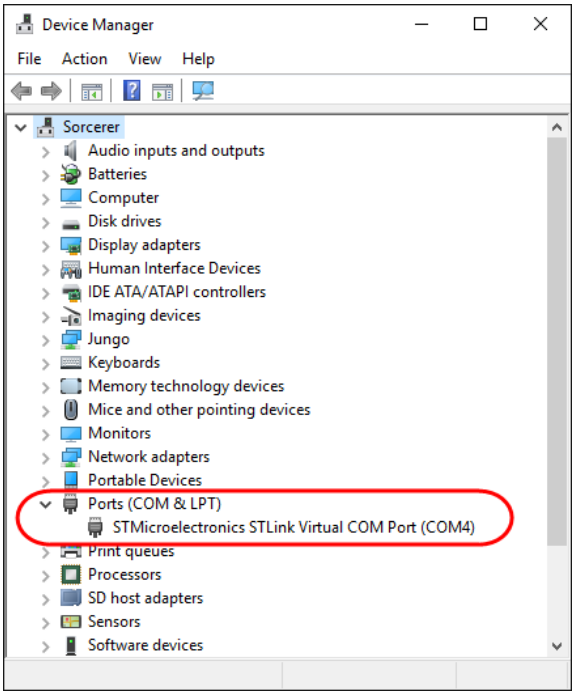
5. โปรแกรม Tera Term

โปรแกรม Tera Term ใช้สำหรับการรับส่งข้อมูลผ่านทางพอร์ตการสื่อสาร เช่น พอร์ตอนุกรม Serial Port (COM Port) ด้วยรหัส ASCII โดยเมื่อเชื่อมต่อเครื่องคอมพิวเตอร์เข้ากับไมโครคอนโทรลเลอร์ผ่านทางพอร์ตอนุกรมหรือพอร์ตอนุกรมเสมือนแล้ว หากผู้ใช้กดปุ่มใดๆ บนคีย์บอร์ดที่เครื่องคอมพิวเตอร์ในโปรแกรม Tera Term โปรแกรมจะส่งรหัส ASCII ของตัวอักษรที่ผู้ใช้กดออกไปทางพอร์ตอนุกรมส่งไปยังไมโครคอนโทรลเลอร์ และถ้าหากไมโครคอนโทรลเลอร์ส่งข้อมูลรหัส ASCII ผ่านทางพอร์ต UART ออกมา โปรแกรมนี้ก็จะรับข้อมูลแล้วแสดงผลตัวอักษรที่มีรหัส ASCII ตรงกันบนหน้าจอของโปรแกรม

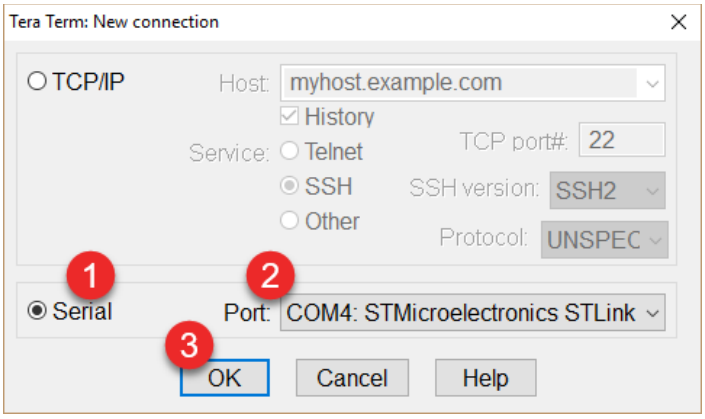
- ดาวน์โหลดและติดตั้งโปแกรม Tera Term ได้ที่ url https://ttssh2.osdn.jp/index.html.en
- เปิดโปรแกรมแล้วตั้งค่าพอร์ตอนุกรมโดยเลือกเมนู File -> Setup -> Serial Port แล้วตั้งค่าให้ตรงกันกับการตั้งค่าของ UART บนไมโครคอนโทรลเลอร์ ดังรูปที่ 5.1 โดยใช้หมายเลข Virtual COM port ตามที่ปรากฏใน Device Manager รูปที่ 5.2
- เชื่อมต่อเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์โดยเลือกเมนู File -> New connection แล้วเลือก COM port ที่ต้องการ ดังรูปที่ 5.3
- สามารถตัดการเชื่อมต่อ ที่เมนู File -> Disconnect
- เคลียร์หน้าจอที่เมนู Edit -> Clear screen



รูปที่ 5.1 การตั้งค่าโปรแกรม Tera Term



รูปที่ 5.2 แสดงหมายเลข COM Port ใน Device Manager



รูปที่ 5.3 การสร้างการเชื่อมต่อไปยังไมโครคอนโทรลเลอร์

6. การทดลอง

1. ให้เขียนโปรแกรมรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์และไมโครคอนโทรลเลอร์ผ่าน UART2 โดยใช้โปรแกรม STM32CubeMX สร้างไฟล์โปรเจ็คขึ้นมาแล้วตั้งค่าดังรูปที่ 2.1 และรูปที่ 2.2

จากนั้นให้เขียนโปรแกรมภายในลูป while ในฟังก์ชัน `main()` เพื่อส่งข้อความ **`“Hello World!!”`** ไปแสดงในโปรแกรม Tera Term ผ่านทางพอร์ต UART โดยใช้ฟังก์ชันส่งข้อมูลดังรูปที่ 4.1 โดยให้ include **`string.h`** เพิ่มเติมเข้ามาในไฟล์ **`main.c`** เพื่อให้สามารถใช้งานฟังก์ชัน **`strlen`** เพื่อหาความยาวของ string ได้ ดังรูปที่ 6.1

```
/* Includes ------------------------
#include "main.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------
/* USER CODE BEGIN Includes */
#include "string.h"
/* USER CODE END Includes */
```

รูปที่ 6.1 การสร้างการเชื่อมต่อไปยังไมโครคอนโทรลเลอร์

2. จงเขียนโปรแกรมบนไมโครคอนโทรลเลอร์เพื่อแสดงข้อความดังต่อไปนี้ในโปรแกรม Tera Term

`Input =>`

จากนั้นโปรแกรมจะ**<u>รอให้ผู้ใช้ป้อนตัวอักษร 1 ตัว</u>** เมื่อผู้ใช้ป้อนตัวอักษรใดให้ไมโครคอนโทรลเลอร์นำตัวอักษรที่ถูกป้อนนั้นกลับมาแสดงในโปรแกรม Tera Term จากนั้นให้แสดงข้อความเดิม (**`Input =>`**) ในบรรทัดไหม้ แล้วรอให้ผู้ใช้ป้อนตัวอักษรตัวต่อไป ให้โปแกรมทำงานแบบนี้ซ้ำไปเรื่อยๆ จนกระทั่งผู้ใช้กดปุ่ม ‘**q**’ ให้ไมโครคอนโทรลเลอร์แสดงข้อความ “**QUIT**” แล้วหยุดการทำงานไม่รับและไม่ส่งข้อมูลใดๆ อีก

```
Input =>a
Input =>b
Input =>1
Input =>q
QUIT
```

3. ต่อ LED สีแดงและสีเขียวอย่างละ 1 ดวง แบบ Pull up เข้ากับขา GPIO ของไมโครคอนโทรลเลอร์ตามต้องการ จากนั้นให้ไมโครคอนโทรลเลอร์แสดงข้อความเมนูผ่านทาง UART2 เพื่อไปแสดงในโปรแกรม Tera Term ดังรูปที่ 6.2

```
Display Blinking LED PRESS (r, g)
Display Group Members PRESS m
Quit PRESS q
        Input =>
```

รูปที่ 6.2 แสดงเมนูในโปรแกรม Hyper Terminal

จากนั้นรอให้ผู้ใช้ป้อนตัวอักษร 1 ตัว แสดงตัวอักษรที่ผู้ใช้ป้อนในโปรแกรม Tera Term แล้วให้โปรแกรมทำงานตามที่กำหนดในตารางที่ 6.1 หลังจากทำงานเสร็จสิ้นแล้ว ให้ไมโครคอนโทรลเลอร์รอรับคำสั่งถัดไปด้วยการแสดงเมนูเฉพาะบรรทัด `Input =>`

ตารางที่ 6.1 แสดงการทำงานของโปรแกรมสำหรับการทดลองข้อ 3

ปุ่มที่กด	การทำงานของ LED
r	ให้ LED **<u>สีแดง</u>**ที่เชื่อมต่อกับไมโครคอนโทรลเลอร์กระพริบ 3 ครั้ง โดยหน่วงเวลา 300 ms
g	ให้ LED **<u>สีเขียว</u>**ที่เชื่อมต่อกับไมโครคอนโทรลเลอร์กระพริบ 3 ครั้ง โดยหน่วงเวลา 300 ms
m	แสดงชื่อ และรหัสของสมาชิกในกลุ่ม โดยแสดงผลลัพธ์ 2-4 บรรทัด ดังตัวอย่าง 59xxxxxxxxx First1 Last1 59xxxxxxxxx First2 Last2
q	แสดงคำว่า `QUIT` แล้วให้ไมโครคอนโทรลเลอร์จบการทำงาน ไม่รับหรือส่งข้อมูลใดๆ อีก
ปุ่มอื่นๆ	แสดงคำว่า `Unknown Command` แล้วรอรับข้อมูลใหม่

ข้อที่ 1

```
char str[] = "Hello!, World\r\n";
while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
HAL_Delay(500);
```

ข้อที่ 2

```
/* USER CODE BEGIN WHILE */
int run = 1;
while(run){
	// Transmit
	char str[] = "\r\nInput =>";
	while(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_TC)==RESET);
	HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
	HAL_Delay(100);
	// Receive
	char ch1[2] = "";
	while(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_RXNE)== RESET);
	HAL_UART_Receive(&huart2, (uint8_t*) &ch1, 1, 1000);
	//Transmit data of receive
	while(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_TC)==RESET);
	HAL_UART_Transmit(&huart2, (uint8_t*) &ch1, strlen(ch1),1000);
	if(ch1[0] == 'q' || ch1[0] == 'Q') {
		char str[] = "\r\nQUIT\r\n";
		while(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_TC)==RESET);
		HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
		HAL_Delay(500);
		run = 0;
	}
	/* USER CODE END WHILE */
}
```

ข้อที่ 3

```
/* USER CODE BEGIN WHILE */
int run = 1;
char str[] = "Display Blinking LED PRESS (r, g)\r\nDisplay Group Members PRESS m\r\nQuit PRESS q";
HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
while(__HAL_UART_GET_FLAG(&huart2, UART_FLAG_TC)==RESET);
HAL_Delay(100);
while(run){
	//Transmit
	char str1[] = "\r\n\tInput =>";
	while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
	HAL_UART_Transmit(&huart2, (uint8_t*) str1, strlen(str1),1000);
	HAL_Delay(100);
	//Receive
	char ch1[2]="A";
	while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_RXNE)==RESET);
	HAL_UART_Receive(&huart2, (uint8_t*) &ch1, 1, 1000);
	//Transmit data of receive
	while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
	HAL_UART_Transmit(&huart2, (uint8_t*) &ch1, strlen(ch1),1000);
	if(ch1[0] == 'q' || ch1[0] == 'Q') {
		char str[] = "\r\nQUIT\r\n";
		while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
		HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
		HAL_Delay(500);
		run = 0;
	} else if(ch1[0] == 'r' || ch1[0] == 'R') {
		for(int i = 0 ; i < 6 ; i++) {
			i % 2 == 0 ?	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_SET):
					HAL_GPIO_WritePin(GPIOC,GPIO_PIN_0,GPIO_PIN_RESET);
			HAL_Delay(300);
		}
	} else if(ch1[0] == 'g' || ch1[0] == 'G') {
		for(int i = 0 ; i < 6 ; i++) {
			i % 2 == 0 ?	HAL_GPIO_WritePin(GPIOC,GPIO_PIN_1,GPIO_PIN_SET):
					HAL_GPIO_WritePin(GPIOC,GPIO_PIN_1,GPIO_PIN_RESET);
			HAL_Delay(300);
		}
	} else if(ch1[0] == 'm' || ch1[0] == 'M') {
		char str[] = "\r\n594020XXXX\r\nMr.XXXX XXXX\r\n594020YYYY\r\nMr.YYYY YYYY";
		while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
		HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
		HAL_Delay(100);
	} else {
		char str[] = "\r\nUnknow Command\r\n";
		while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET);
			HAL_UART_Transmit(&huart2, (uint8_t*) str, strlen(str),1000);
			HAL_Delay(100);
		}
	/* USER CODE END WHILE */
}
```

ใบตรวจการทดลองที่ 3

KU CSC Embedded System

วัน/เดือน/ปี _____ **กลุ่มที่** _____

1. รหัสนิสิต _____ ชื่อ-นามสกุล _____

2. รหัสนิสิต _____ ชื่อ-นามสกุล _____

3. รหัสนิสิต _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 2 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

การทดลองข้อ 3 ผู้ตรวจ _____ วันที่ตรวจ ☐ W ☐ W+1

คำถามท้ายการทดลอง

1. หากตัดคำสั่งต่อไปนี้ออกจากโค้ดของการทดลองข้อ 1 โปรแกรมจะสามารถทำงานได้สมบูรณ์เหมือนเดิมหรือไม่? เพราะเหตุใด?

```
while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_TC)==RESET){}
```

ทำงานได้สมบูรณ์เหมือนเดิม เพราะคำสั่งดังกล่าว เป็นคำสั่งรอว่าส่งข้อมูลสำเร็จแล้วหรือยัง โดยจะรอการตอบกลับจากฝั่งผู้รับ ก่อนที่จะให้บรรทัดถัดไปทำงานได้ ถ้าหากเอาโค้ดบรรทัดนี้ออก โปรแกรมก็ยังสามารถส่งข้อมูลไปได้เหมือนเดิม โดยไม่สนการตอบกลับของฝั่งผู้รับ

2. หากตัดคำสั่งต่อไปนี้ออกจากโค้ดรูปที่ 4.2 โปรแกรมจะสามารถทำงานได้สมบูรณ์เหมือนเดิมหรือไม่? เพราะเหตุใด?

```
while(__HAL_UART_GET_FLAG(&huart2,UART_FLAG_RXNE)== RESET){}
```

ทำงานไม่สมบูรณ์เหมือนเดิม เพราะคำสั่งดังกล่าว เป็นคำสั่งรอว่ามีการเขียนข้อมูลที่ Register นั้นหรือยังหรือรอให้ข้อมูลใน Register นั้นไม่เป็น Empty ก่อนที่จะให้บรรทัดถัดไปทำงานได้ ถ้าหากเอาโค้ดบรรทัดนี้ออก จะทำให้โปรแกรมนี้ไปอ่านค่า ที่เป็น Empty มาเก็บไว้ในตัวแปล ch1 อยู่ตลอดการลูปแม้ไม่ได้มีการส่งข้อมูลจากฝั่งผู้ส่งเลย