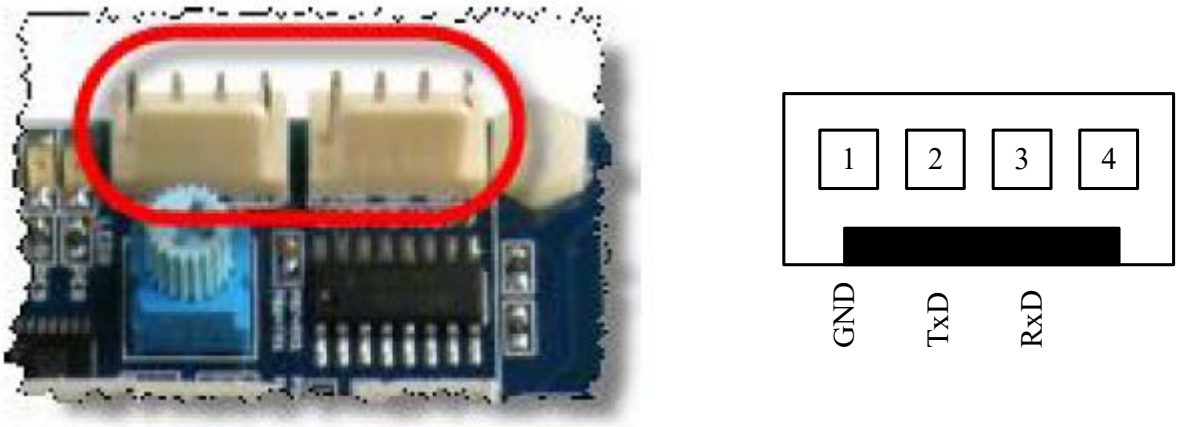
<u>การทดลองที่ 3</u> การใช้งาน UART

วัตถุประสงค์

1) เข้าใจการทำงานของ UART
2) สามารถเขียนโปรแกรมเพื่อรับส่งข้อความผ่านพอร์ต UART

1. UART / USART

USART (Universal Synchronous/Asynchronous Receiver/Transmitter) เป็นพอร์ตสื่อสารซึ่งสามารถใช้งานได้ทั้งแบบ Asynchronous (UART) และ Synchronous (USART) ไมโครคอนโทรลเลอร์หมายเลข STM32F107 มีพอร์ต UART จำนวน 5 พอร์ต แต่บอร์ด ET-STM32F ARM KIT ได้สร้างคอนเนคเตอร์ไว้ให้เพียง 2 พอร์ต ได้แก่ UART1 และ UART2 การดาวน์โหลดโปรแกรมลงบอร์ด (In System Programming) ใช้พอร์ต UART2 โดย UART ทั้งสองพอร์ตทำงานด้วยสัญญาณ RS232 เนื่องจากได้เชื่อมต่อวงจรแปลงระดับสัญญาณ MAX3232 เรียบร้อยแล้ว



รูปที่ 1.1 แสดงพอร์ต UART ทั้งสองพอร์ตและตำแหน่งของขาสัญญาณ

แต่ละขาของไอซีนั้นสามารถกำหนดหน้าที่การทำงานได้สูงสุด 3 รูปแบบ คือ

1) Main Function หรือ After Reset
2) Default Alternate Function
3) Remap Alternate Function

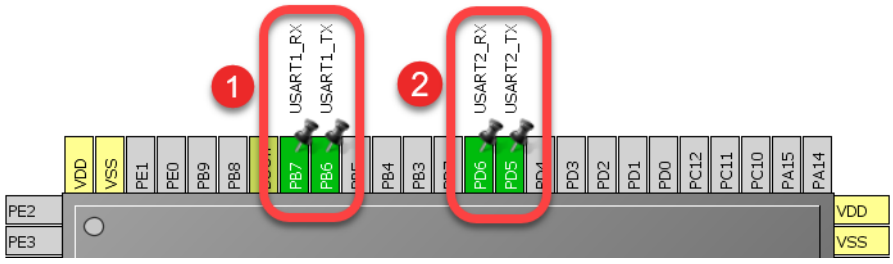
บอร์ด ET-STM32F ARM KIT ได้เชื่อมต่อขา PB6 และ PB7 เข้ากับคอนเนคเตอร์ของพอร์ต UART1 บนบอร์ดที่ขา Tx และ Rx ตามลำดับ เมื่อไมโครคอนโทรลเลอร์เริ่มต้นการทำงานขา 92 และ 93 จะทำหน้าที่เป็น GPIO ได้แก่ PB6 และ PB7 ตามลำดับ หากต้องการเปลี่ยนให้ขา 92 และ 93 ทำหน้าที่เป็น UART1 จะต้องเขียนโปรแกรมเพื่อกำหนดให้ขา 92 และ 93 ทำหน้าที่ Alternate Function แบบ Remap ส่วนพอร์ต UART2 บนบอร์ดเชื่อมต่อกับไมโครคอนโทรเลอร์ที่ขาหมายเลข 86 และ 87 สรุปการเชื่อมต่อขาของไอซีไมโครคอนโทรลเลอร์กับพอร์ต UART บนบอร์ดการทดลองได้ดังตารางที่ 1.1

ตารางที่ 1.1 แสดงการทำงานของขาไมโครคอนโทรลเลอร์ที่เกี่ยวกับพอร์ต UART1 และ UART2

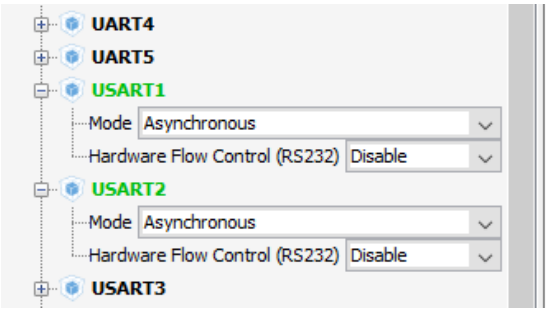
Pin NO.	Main Function (After Reset)	Alternate Function	
		Default	Remap
92	PB6	I2C1_SCL / TIM4_CH1	**USART1_TX** / CAN2_TX
93	PB7	I2C1_SDA / TIM4_CH2	**USART1_RX**
86	PD5	-	**USART2_TX**
87	PD6	-	**USART2_RX**
68	PA9	**USART1_TX** / TIM1_CH2 / OTG_FS_VBUS	-
69	PA10	**USART1_RX** / TIM1_CH3 / OTG_FS_ID	-
25	PA2	**USART2_TX** / TIM5_CH3 / ADC12_IN2 / TIM2_CH3 (7) / ETH_MII_MDIO / ETH_RMII_MDIO	-
26	PA3	**USART2_RX** / TIM5_CH4 / ADC12_IN3 / TIM2_CH4 / ETH_MII_COL	-

2. การตั้งค่า UART

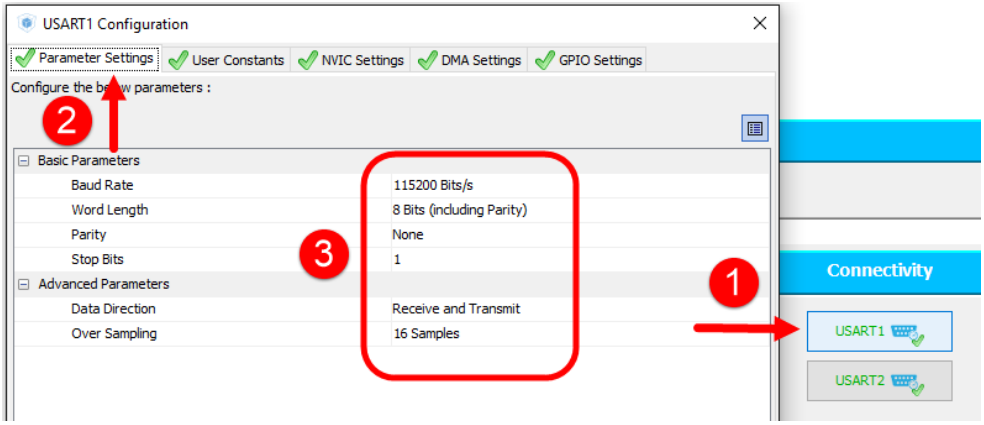
การจะใช้งานพอร์ต UART บนบอร์ดการทดลองต้องตั้งค่าขาของไมโครคอนโทรลเลอร์ที่เชื่อมต่ออยู่กับพอร์ตบนบอร์ดด้วยโปรแกรม STM32CubeMX ให้ถูกต้อง ดังรูปที่ 2.2 - รูปที่ 2.4



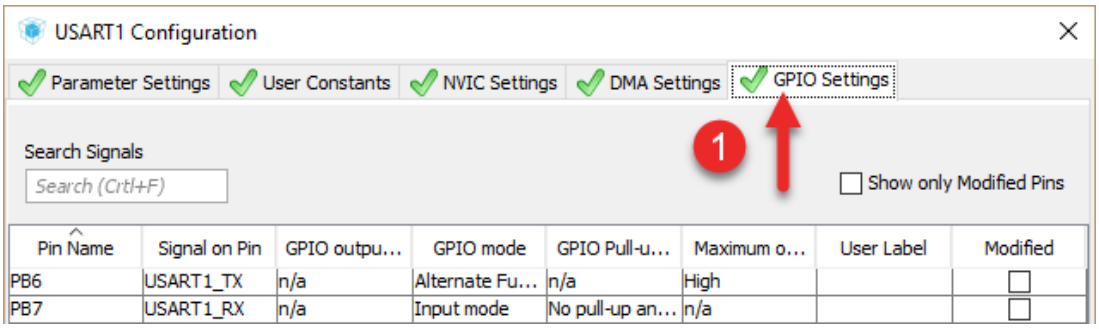
รูปที่ 2.1 แสดงการตั้งค่าพอร์ต UART1 และ UART2 ที่ขาของไมโครคอนโทรลเลอร์



รูปที่ 2.2 แสดงการตั้งค่าพอร์ต UART1 และ UART2 ที่แท็บ PINOUT ของโปรแกรม STM32CubeMX

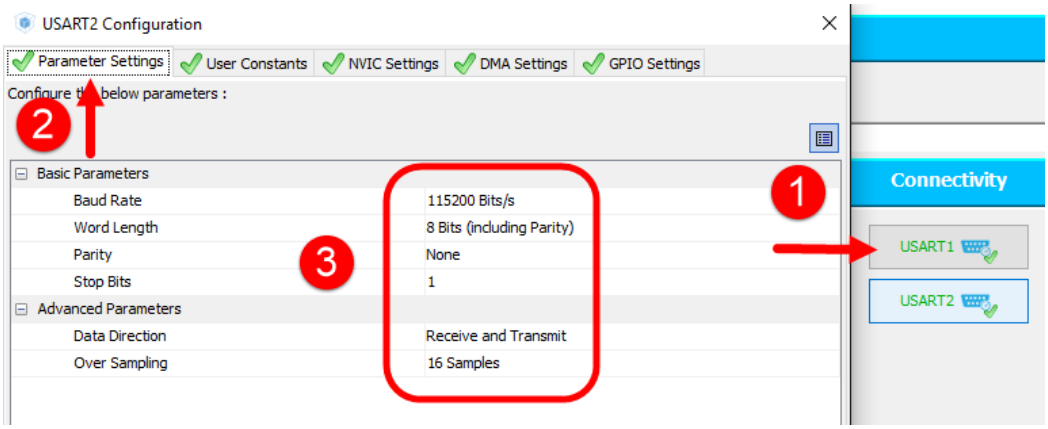


(a)

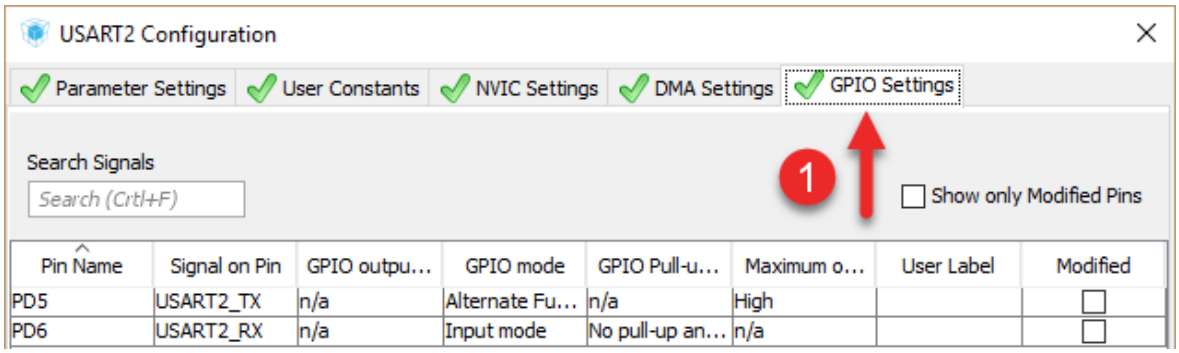


(b)

รูปที่ 2.3 แสดงการตั้งค่าพอร์ต UART1



(a)



(b)

รูปที่ 2.4 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

จากนั้นทำการ remap ให้ขาไมโครคอนโทรลเลอร์จากที่ทำหน้าที่เป็น GPIO PB6, PB7, PD5 และ PD6 ให้ทำหน้าที่เป็น USART1_TX, USART1_RX, USART2_TX และ USART2_RX ตามลำดับ

การตั้งค่าการทำงาน UART จากโปรแกรม STM32CubeMX จะถูกกำหนดไว้ที่ไฟล์ 2 ไฟล์ ได้แก่

- `stm32f1xx_hal_msp.c`
- `main.c`

ไฟล์ **`stm32f1xx_hal_msp.c`**

เป็นไฟล์ที่รวมการตั้งค่าขา GPIO ต่างๆ

ฟังก์ชัน `HAL_UART_MspInit()`

- เป็นฟังก์ชันที่ใช้ในการตั้งค่าขา GPIO ที่จะนำมาใช้เป็นขา TX และ RX ของ USART โดยภายในฟังก์ชันจะแยกเป็น 2 ส่วน ได้แก่การตั้งค่าของ UART1 และ UART2 ดังรูปที่ 3.1
- เริ่มต้นการตั้งค่า UART1 ด้วยการจ่ายสัญญาณนาฬิกาให้กับโมดูล

```
__HAL_RCC_USART1_CLK_ENABLE();
```

- กำหนดให้ PB6 เป็น TX โดยจะตั้งค่าให้เป็นเอาต์พุตพุชพูลแบบ alternate function ที่ Speed 50 MHz

```
GPIO_InitStruct.Pin       = GPIO_PIN_6;
GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_HIGH;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

- กำหนดให้ PB7 เป็น RX โดยจะตั้งค่าให้เป็นอินพุตแบบ Floating

```
GPIO_InitStruct.Pin       = GPIO_PIN_7;
GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull      = GPIO_NOPULL;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

- กำหนดขา PB7 เป็นแบบ remap alternate function เพื่อทำหน้าที่เป็น UART1 ด้วยคำสั่ง

```
__HAL_AFIO_REMAP_USART1_ENABLE();
```

- สำหรับการตั้งค่า UART2 สามารถทำได้เช่นเดียวกับ UART1 ด้วยเริ่มต้นจ่ายสัญญาณนาฬิกาให้กับโมดูล

```
__HAL_RCC_USART2_CLK_ENABLE();
```

- จากนั้นตั้งค่าให้ PD5 และ PD6 ทำหน้าที่เป็น TX และ RX ของ UART2 ตามลำดับ

```
GPIO_InitStruct.Pin       = GPIO_PIN_5;
GPIO_InitStruct.Mode      = GPIO_MODE_AF_PP;
GPIO_InitStruct.Speed     = GPIO_SPEED_FREQ_HIGH;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

GPIO_InitStruct.Pin       = GPIO_PIN_6;
GPIO_InitStruct.Mode      = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull      = GPIO_NOPULL;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

__HAL_AFIO_REMAP_USART2_ENABLE();
```

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{

  GPIO_InitTypeDef GPIO_InitStruct;
  if(huart->Instance==USART1)
  {
  /* USER CODE BEGIN USART1_MspInit 0 */

  /* USER CODE END USART1_MspInit 0 */
    /* Peripheral clock enable */
    __HAL_RCC_USART1_CLK_ENABLE();

    /**USART1 GPIO Configuration
    PB6     ------> USART1_TX
    PB7     ------> USART1_RX
    */
    GPIO_InitStruct.Pin = GPIO_PIN_6;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = GPIO_PIN_7;
    GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

    __HAL_AFIO_REMAP_USART1_ENABLE();

  /* USER CODE BEGIN USART1_MspInit 1 */

  /* USER CODE END USART1_MspInit 1 */
  }
```

(a)

```
else if(huart->Instance==USART2)
{
/* USER CODE BEGIN USART2_MspInit 0 */

/* USER CODE END USART2_MspInit 0 */
  /* Peripheral clock enable */
  __HAL_RCC_USART2_CLK_ENABLE();

  /**USART2 GPIO Configuration
  PD5     ------> USART2_TX
  PD6     ------> USART2_RX
  */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  __HAL_AFIO_REMAP_USART2_ENABLE();

/* USER CODE BEGIN USART2_MspInit 1 */

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

(b)

รูปที่ 3.1 ฟังก์ชัน `HAL_UART_MspInit()`
(a) ส่วนที่ตั้งค่าให้กับ UART1
(b) ส่วนที่ตั้งค่าให้กับ UART2

ไฟล์ `main.c`

`Global variables`

- จะทำการประกาศตัวแปร `huart1` และ `huart2` เพื่อที่จะใช้เป็นตัวแทนของโมดูล UART1 และ UART2 ตามลำดับ

```
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
```

ฟังก์ชัน MX_GPIO_Init()

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อจ่ายสัญญาณนาฬิกาให้กับ GPIO พอร์ต B และพอร์ต D ด้วยคำสั่ง

```
__HAL_RCC_GPIOD_CLK_ENABLE();
__HAL_RCC_GPIOB_CLK_ENABLE();
```

ฟังก์ชัน MX_USART1_UART_Init()

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า UART1 บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับค่าที่กำหนดไว้ในโปรแกรม ดังรูปที่ 3.2 (a)
- เริ่มต้นด้วยการกำหนดให้ `huart1` เป็นตัวแทนของ UART1

```
huart1.Instance =USART1;
```

- แล้วทำการตั้งค่าการทำงานของ USART ดังนี้
  - Baud rate = 115200 bits/second
  - ใช้ 8 บิตใน 1 เฟรม
  - ใช้ 1 stop bit
  - ไม่ใช้ Parity bit
  - กำหนดให้ทำงานทั้งรับและส่งข้อมูล
  - ไม่ใช้ Hardware Flow Control
  - กำหนดให้ทำการ Oversampling 16 เท่า

```
huart1.Init.BaudRate        = 115200;
huart1.Init.WordLength      = UART_WORDLENGTH_8B;
huart1.Init.StopBits        = UART_STOPBITS_1;
huart1.Init.Parity          = UART_PARITY_NONE;
huart1.Init.Mode            = UART_MODE_TX_RX;
huart1.Init.HwFlowCtl       = UART_HWCONTROL_NONE;
huart1.Init.OverSampling    = UART_OVERSAMPLING_16;
if (HAL_UART_Init(&huart1) != HAL_OK)
{
_Error_Handler(__FILE__, __LINE__);
}
```

```c
/* USART1 init function */
static void MX_USART1_UART_Init(void)
{

  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    _Error_Handler(__FILE__, __LINE__);
  }

}
```

(a)

```c
/* USART2 init function */
static 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(__FILE__, __LINE__);
  }

}
```

(b)

รูปที่ 3.2 ฟังก์ชันตั้งค่า UART (a) UART1 และ (b) UART2

ฟังก์ชัน `MX_USART2_UART_Init()`

- เป็นฟังก์ชันที่โปรแกรม STM32CubeMX สร้างขึ้นมา เพื่อตั้งค่า USART2 บนไมโครคอนโทรลเลอร์ให้สอดคล้องกับค่าที่กำหนดไว้ในโปรแกรม ดังรูปที่ 3.2 (b) โดยมีรายละเอียดเช่นเดียวกับการตั้งค่า UART1

ฟังก์ชัน `main()`

- เริ่มต้นการทางานด้วยฟังก์ชันเพื่อตั้งค่าโมดูลต่างๆ ได้แก่
    - `HAL_Init()`
    - `SystemClock_Config()`
    - `MX_GPIO_Init()`
    - `MX_USART1_UART_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 โดยมีฟังก์ชันที่เกี่ยวข้องในการรับส่งข้อมูลดังนี้

ฟังก์ชัน `__HAL_UART_GET_FLAG (__HANDLE__, __FLAG__)`

- ใช้เพื่ออ่านค่าแฟลกต่างๆ ของ UART
- `__HANDLE__` : ระบุ UART ที่ต้องการ เช่น `&huart1` 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 `&huart1` 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(&huart1,UART_FLAG_TC)==RESET){}
HAL_UART_Transmit(&huart1, (uint8_t*) str, strlen(str),1000);
```

รูปที่ 4.1 ตัวอย่างการส่งข้อมูลผ่าน UART

รูปที่ 4.1 แสดงตัวอย่างการส่งข้อมูลของตัวแปรข้อความ str ผ่าน UART ตัวแปร str ถูกปิดท้ายข้อความ **"Hello, World!!"** ด้วยตัวอักษรพิเศษ 2 ตัว ได้แก่ ตัวอักษร Line Feed หรือ **'\n'** ใช้สำหรับขึ้นบรรทัดใหม่โดยเคอร์เซอร์จะอยู่ตำแหน่งเดียวกันกับบรรทัดบนและตัวอักษร Carriage Return หรือ **'\r'** 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 `&huart1` เป็นต้น
- `pData` : คือ Pointer ที่ชี้ไปยังตำแหน่งที่อยู่ของตัวแปรที่ใช้รับข้อมูล
- `Size` : ระบุความยาวของข้อมูลที่จะรับในหน่วย Byte
- `Timeout` : ระยะเวลาที่ฟังก์ชันสามารถใช้เพื่ออ่านข้อมูลที่ได้รับเข้ามาจากบัฟเฟอร์มีหน่วยเป็น millisecond หากไม่สามารถทำงานเสร็จภายในระยะเวลาที่กำหนด ฟังก์ชันจะส่งค่ากลับเป็น `HAL_TIMEOUT`
- โดยตัวฟังก์ชันจะรีเทิร์นค่าออกเป็นสถานะการทำงาน

```
char str;

while(__HAL_UART_GET_FLAG(&huart1,UART_FLAG_RXNE)==RESET){}
HAL_UART_Receive(&huart1, (uint8_t*)&str, 1, 1000);
```

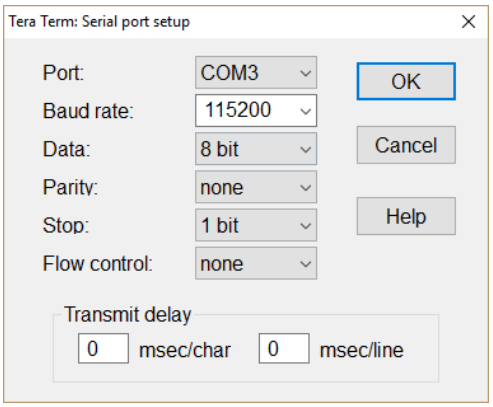
รูปที่ 4.2 ตัวอย่างการรับข้อมูลผ่าน UART

5. โปรแกรม Tera Term

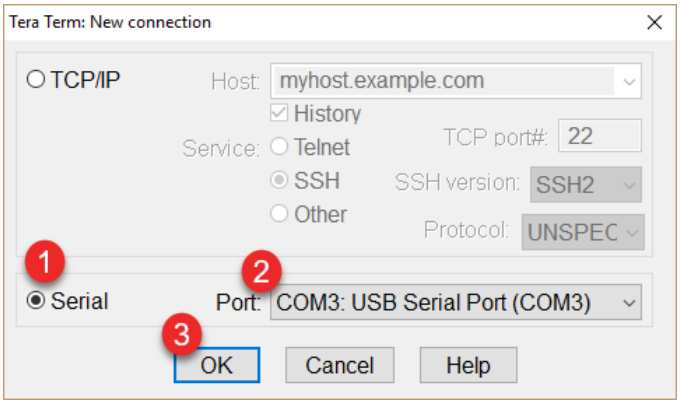
โปรแกรม Tera Term ใช้สำหรับการรับส่งข้อมูลผ่านทางพอร์ตการสื่อสาร เช่น พอร์ตอนุกรม Serial Port (COM Port) ด้วยรหัส ASCII โดยเมื่อเชื่อมต่อเครื่องคอมพิวเตอร์เข้ากับไมโครคอนโทรลเลอร์ผ่านทางพอร์ตอนุกรมแล้ว หากผู้ใช้กดปุ่มใดๆ บนคีย์บอร์ดที่เครื่องคอมพิวเตอร์ในโปรแกรม Tera Term โปรแกรมจะส่งรหัส ASCII ของตัวอักษรที่ผู้ใช้กดออกไปทางพอร์ตอนุกรมส่งไปยังไมโครคอนโทรลเลอร์ 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 COM port ตามที่ปรากฏใน Device Manager
- เชื่อมต่อเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์โดยเลือกเมนู File -> New connection แล้วเลือก COM port ที่ต้องการ ดังรูปที่ 5.2
- สามารถตัดการเชื่อมต่อ ที่เมนู File -> Disconnect
- เคลียร์หน้าจอที่เมนู Edit -> Clear screen



รูปที่ 5.1 การตั้งค่าโปรแกรม Tera Term



รูปที่ 5.2 การสร้างการเชื่อมต่อไปยังไมโครคอนโทรลเลอร์

เนื่องจากโปรแกรม Demo Flash Loader และโปรแกรม Tera Term ใช้ COM port ของเครื่องคอมพิวเตอร์พอร์ตเดียวกัน ดังนั้นหากต้องการใช้งานโปรแกรมใดก็ให้ปิดอีกโปรแกรมก่อนเสมอ การแก้ปัญหากรณีที่โปรแกรมหนึ่งไม่คืนสิทธิ์การใช้งาน COM port จนทำให้อีกโปรแกรมไม่สามารถใช้งานได้ ให้ทดลองถอดสาย USB ออกแล้วเสียบใหม่ หากยังไม่สามารถแก้ปัญหาได้ให้รีสตาร์ทเครื่องคอมพิวเตอร์

6. การทดลอง

1. การทดลองการทำงานแบบ **Null Modem** เพื่อทดสอบตัวโปรแกรมและสายสัญญาณที่ใช้รับส่งข้อมูลว่าอยู่ในสภาพปกติหรือไม่ โดยให้เปิดโปรแกรม Tera Term ขึ้นมาแล้วสร้าง New Connection จากนั้นนำสายสัญญาณอนุกรม (USB to RS232) เชื่อมต่อเข้าที่พอร์ต USB ของเครื่องคอมพิวเตอร์ ส่วนปลายอีกด้ายเชื่อมต่อเข้ากับสายดาวน์โหลด สำหรับปลายสายดาวน์โหลดอีกด้านให้ปล่อยไว้**ไม่ต้อง**เสียบเข้าที่บอร์ด แล้วทดลองกดปุ่มใดๆ บนคีย์บอร์ดในโปรแกรมแล้วสังเกตผล

จากนั้นให้เชื่อมขา Tx กับ Rx ที่ปลายสายดาวน์โหลดเข้าด้วยกันด้วยสายไฟ แล้วทดลองกดปุ่มบนคีย์บอร์ดเช่นเดิม สังเกตผลลัพธ์ที่เกิด พร้อมอธิบายการทำงาน

---

2. ให้เขียนโปรแกรมรับส่งข้อมูลระหว่างไมโครคอนโทรลเลอร์และเครื่องคอมพิวเตอร์**ผ่าน UART2** โดยเริ่มต้นให้ไมโครคอนโทรลเลอร์แสดงเมนูในโปรแกรม Tera Term ดังรูปที่ 6.1

```
Display Animated LED PRESS (u, d, l, r c)
Display Group Members PRESS m
Quit PRESS q
        Input =>
```

รูปที่ 6.1 แสดงเมนูในโปรแกรม Hyper Terminal

จากนั้นรอให้ผู้ใช้ป้อนตัวอักษร 1 ตัว แสดงตัวอักษรที่ผู้ใช้ป้อนให้ปรากฏบนโปรแกรม Tera Term แล้วให้โปรแกรมทำงานตามที่กำหนดในตารางที่ 6.1 หลังจากทำงานเสร็จสิ้นแล้ว ให้**ไมโครคอนโทรลเลอร์รอรับคำสั่งถัดไปด้วยการแสดงเมนูเฉพาะบรรทัด** **`Input =>`**

3. ให้เปลี่ยนพอร์ตที่ใช้ในการทดลองข้อ 2 เป็น UART1

ตารางที่ 6.1 แสดงการทำงานของโปรแกรมสำหรับการทดลองข้อ 2

ปุ่มที่กด	การทำงานของ LED
u	เมื่อโยก Joy Switch Up ให้ LED ติดค้างทีละดวงต่อการโยก 1 ครั้ง ถ้าหากไม่มี LED ดวงใดติดอยู่เลยให้เริ่มต้นที่ LED7 ไปยัง LED0 แล้ววนไป LED7 อีกครั้ง **เริ่มต้น** Joy Switch UP -> LED7 ติดค้าง -> Joy Switch UP -> LED7 ดับ และ LED6 ติดค้าง -> Joy Switch UP -> LED6 ดับ และ LED5 ติดค้าง . . . LED1 ดับ และ LED0 ติดค้าง -> Joy Switch UP -> LED0 ดับ และ LED7 ติดค้าง แต่ถ้ามี LED ติดค้างอยู่แล้วให้เริ่มต้นที่ LED ดวงนั้น
d	เหมือนกับการกดปุ่ม u แต่มีทิศทางตรงกันข้าม (LED0 ไปยัง LED7)
l	เมื่อโยกสวิตช์ 1 ครั้ง LED จะไล่ติดดับทีละ 1 คู่ ทั้งหมด 4 ครั้ง เริ่มต้นจากคู่นอกสุด เข้ามายังคู่ในสุด โดย LED7 ติดคู่กับ LED0, LED6 ติดคู่กับ LED1, LED5 ติดคู่กับ LED2 และ LED4 ติดคู่กับ LED3 โดยหน่วงเวลาตามความเหมาะสมไม่เร็วหรือช้าจนเกินไป
r	เหมือนกับการกดปุ่ม l แต่เริ่มต้นติดจากคู่ในออกไปยังคู่นอก
c	เมื่อกดให้ LED7 LED5 LED3 LED1 ติดค้างชั่วครู่แล้วดับ จากนั้นให้ LED อีก 4 ดวงที่เหลือติดค้างชั่วครู่แล้วดับ
m	แสดงชื่อ และรหัสของสมาชิกในกลุ่ม โดยแสดงผลลัพธ์ 4 บรรทัด
q	แสดงคำว่า Quit แล้วให้ไมโครคอนโทรลเลอร์จบการทำงาน ไม่รับหรือส่งข้อมูลใดๆ อีก
ปุ่มอื่นๆ	แสดงคำว่า Unknown Command แล้วรอรับข้อมูลใหม่

7. การทดลองพิเศษ (เลือกทำข้อ 1 ข้อ)

1. จงสร้างโปรแกรมเครื่องคิดเลขโดยรับอินพุตจากเครื่องคอมพิวเตอร์ผ่านทาง UART เพื่อนำไปประมวลผลที่ไมโครคอนโทรลเลอร์

- อินพุตจะแยกเป็น 2 ส่วน ได้แก่ เครื่องหมายที่ต้องการคำนวณ (+, -, *, /) และข้อมูลตัวเลข
- กำหนดให้ข้อมูลอินพุตเป็นตัวเลขจำนวนเต็มบวกขนาดไม่เกิน 8 บิต
- กำหนดให้ข้อมูลเอาต์พุตเป็นตัวเลขจำนวนเต็มบวกขนาดไม่เกิน 16 บิต
- แสดงตัวอย่างการแสดงผลดังรูปที่ 7.1

```
Calculator:
 Select Operation PRESS (+, -, *, /)
 Quit PRESS q
        First Operand :10↵
   Operation  :+↵
        Second Operand : 20↵
   Result => 30

   First Operand  :
```

รูปที่ 7.1 ตัวอย่างการทำงานของโปรแกรมเครื่องคิดเลข

2. จงสร้างโปรแกรมสนทนาระหว่างเครื่องคอมพิวเตอร์ 2 เครื่องผ่านไมโครคอนโทรลเลอร์ทางพอร์ต UART ทั้งสองพอร์ต โดยการสนทนาจะมีเงื่อนไขดังนี้

- เป็นการสนทนาสลับกันระหว่างเครื่องคอมพิวเตอร์ที่เชื่อมต่อ UART1 กับเครื่องคอมพิวเตอร์ที่เชื่อมต่อ UART2 กำหนดให้เครื่องคอมพิวเตอร์ที่เชื่อมต่อ UART1 เป็นฝ่ายเริ่มส่งข้อความก่อนเสมอ
- การเริ่มสนทนาในแต่ละฝั่งจะต้องทำการตั้งชื่อผู้สนทนาและชื่อนั้นจะต้องไปปรากฏในเครื่องคอมพิวเตอร์ที่เป็นคู่สนทนาด้วย
- การจบการสนทนาจะทำได้จากฝั่งที่ได้สิทธิ์เป็นผู้ส่งข้อความในขณะนั้นเท่านั้น โดยจะต้องมีข้อความแจ้งทั้งสองฝั่งว่าการสนทนาจบแล้ว แล้วทำการหยุดรับอินพุตใดๆ จากผู้ใช้ทั้งสองฝั่ง
- แสดงตัวอย่างการแสดงผลดังรูปที่ 7.2

```
Man from U.A.R.T.1!
Quit PRESS q
        Name: Mr.One↵
        Mr.Two is ready
        Mr.One => Hi there!↵
        Mr.Two :Hi!!
        Mr.One => q↵
Conversation is ended.
```

```
Man from U.A.R.T.2!
Quit PRESS q
        Mr.One is ready
        Name: Mr.Two↵
        Mr.One :Hi there!
        Mr.Two => Hi!!↵
Conversation is ended.
```

รูปที่ 7.2 ตัวอย่างการทำงานของโปรแกรมสนทนา

ใบตรวจการทดลองที่ 3

วัน/เดือน/ปี _____ ☐ **Sec 1** ☐ **Sec 2 กลุ่มที่** _____

1. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

2. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

3. รหัสนักศึกษา _____ ชื่อ-นามสกุล _____

ลายเซ็นผู้ตรวจ

การทดลองข้อ 1 & 3 ผู้ตรวจ _____ สัปดาห์ที่ตรวจ _____

คำถามท้ายการทดลอง

1. หากต้องการใช้งานโมดูล UART3 เพิ่มเติม ต้องใช้ขาใดสำหรับการส่งและรับข้อมูล และกำหนดการทำงาน Alternate Function แบบ default หรือ remap

---
