

7.Communication

DR.SOMSIN THONGKRAIRAT

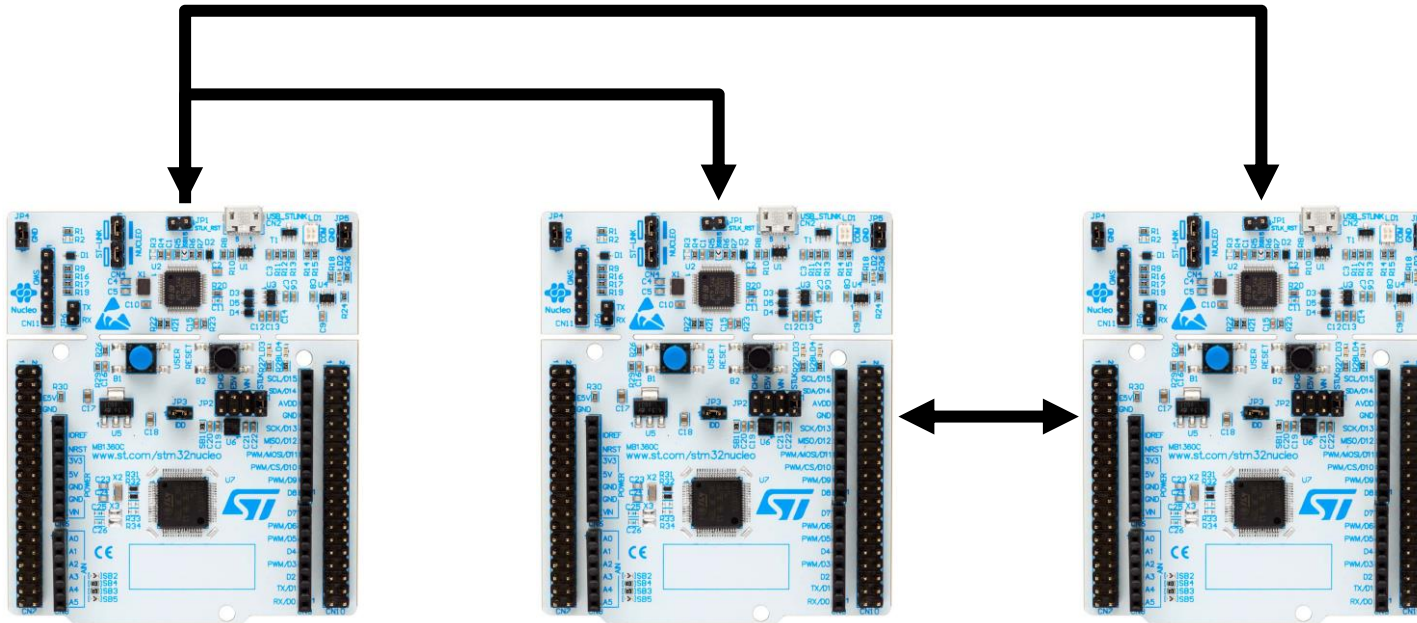


life.augmented



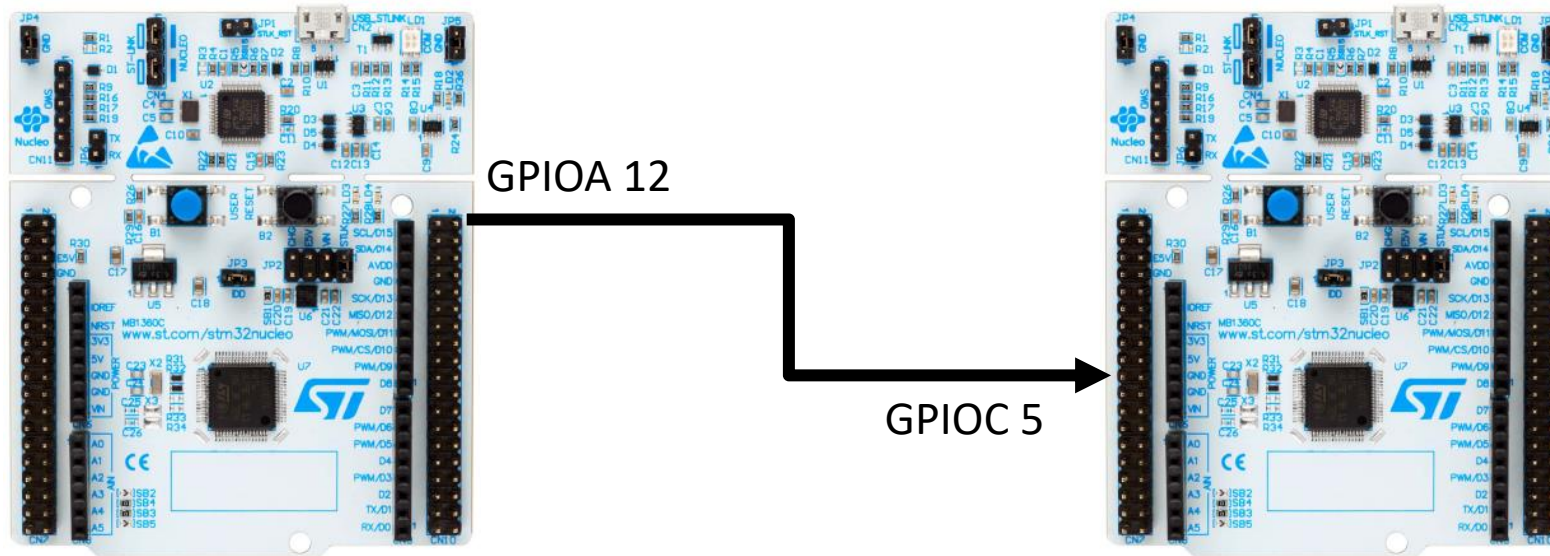
Connectivity

When two or more MCU are in the same system.
Communication are need!



Simplest communication

Send 1 bit or 1 state at a time (GPIO)



Type of communication

- Categorize by direction

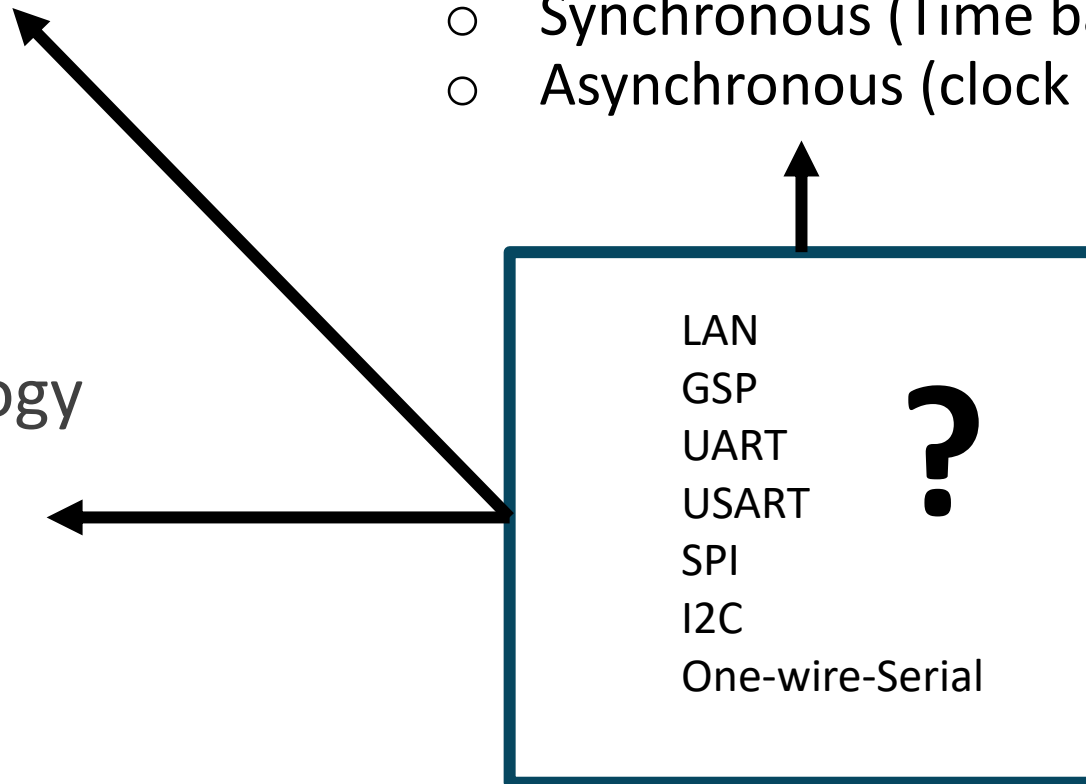
- Simplex
- Half duplex
- Full duplex

- Categorize by Clock

- Synchronous (Time base)
- Asynchronous (clock base)

- Categorize by topology

- Master-slave
- One-to-one
- broadcast

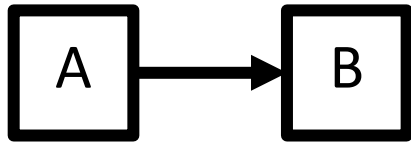


Categorize by direction

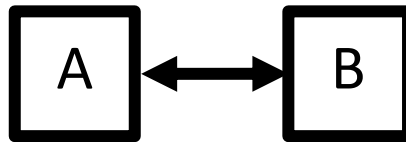
Simplex -> ส่งข้อมูลทางเดียว ไม่สามารถส่งข้อมูลกลับได้

Full duplex -> ส่งข้อไปกลับพร้อมกันได้

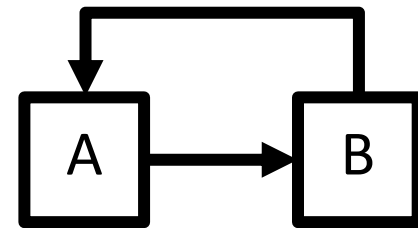
Half duplex -> ส่งข้อไปกลับ แต่ทำพร้อมกันไม่ได้



Simplex



Half-Duplex



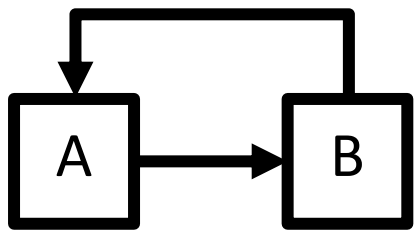
Full-Duplex

Categorize by topology

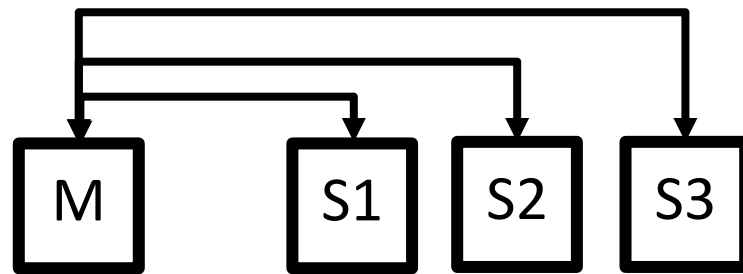
One-to-one -> only 2 MCU in network

Master-slave -> Master device control communication BUS

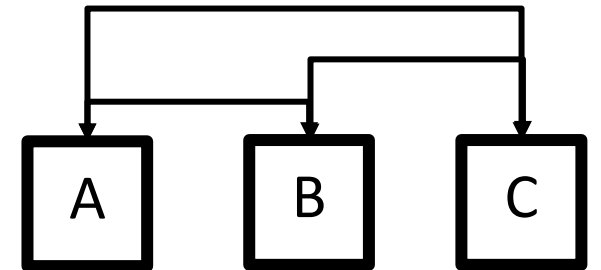
BUS -> every device do same behavior



One to One



Master slave

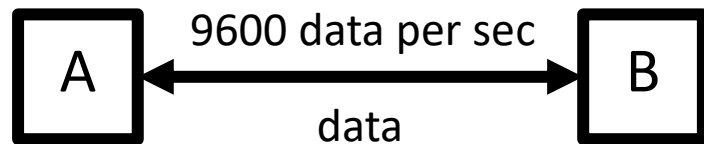


BUS

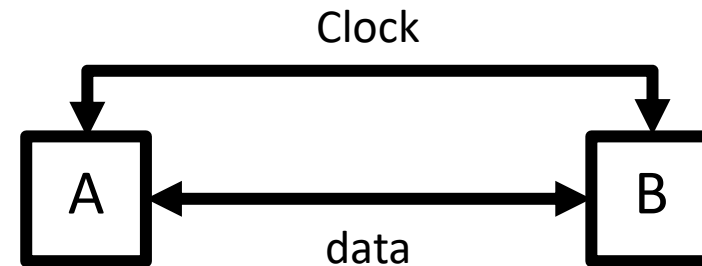
Categorize by Clock

Synchronous (Time base) -> speed define by time (pre-define)

Asynchronous (clock base) -> speed define by clock signal (data send with clock signal)



Asynchronous



Synchronous

STM availability

Serial -> UART -> Asynchronous , FULL-Duplex

SPI -> USART -> Synchronous , Master slave, Half-Duplex

I2C -> Two Wire -> Synchronous , Master slave , Full-Duplex

CAN -> CAN interface -> Asynchronous , BUS, Half-Duplex

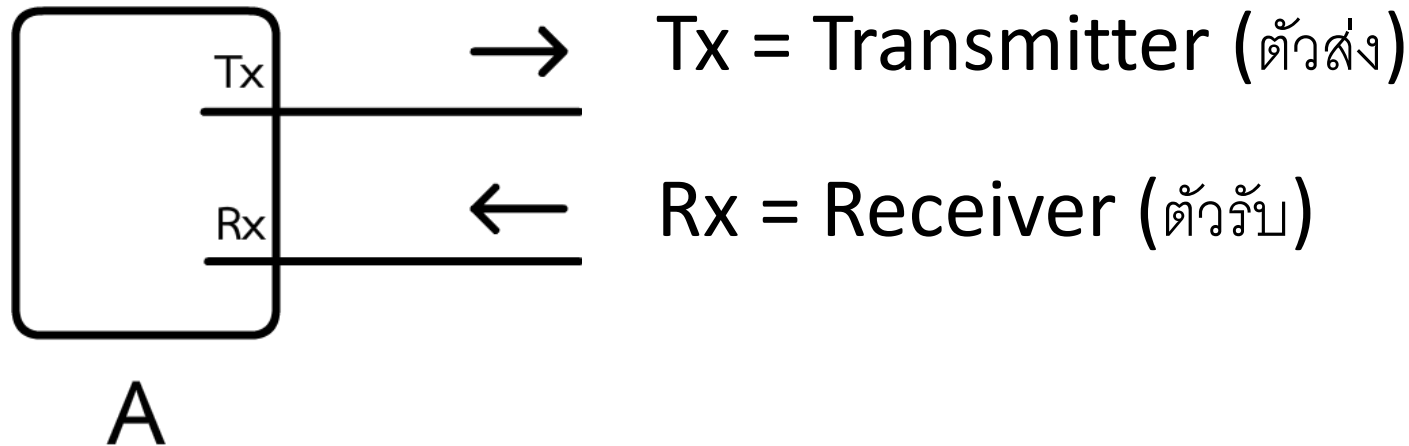
UCPD -> Asynchronized

ETH -> LAN

Serial communication

send serial data through media directory

'H' -> 0b01001000 (MSB)



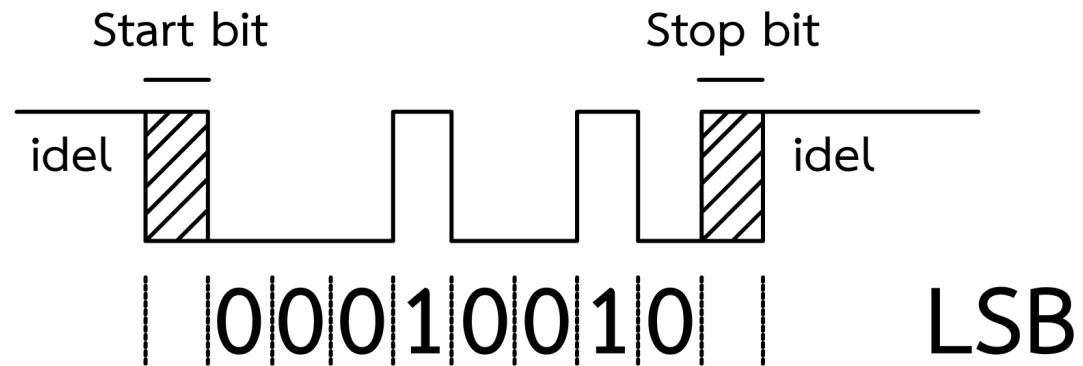
Serial communication data

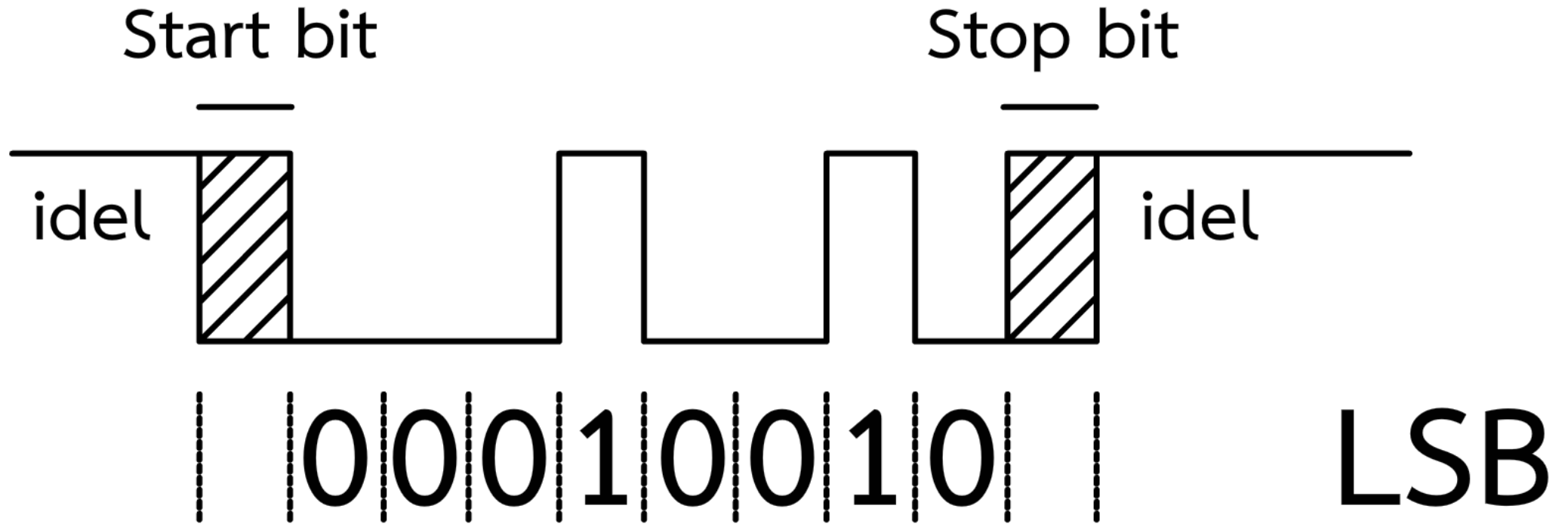
LSB (less significant bit first) format (Default)

Start bit -> start signal to receive incoming data (logic 0 by default)

Stop bit -> mark end of 1 data (e.g., end of byte) (logic 1 by default)

Parity bit -> check sum (transmit error checking)



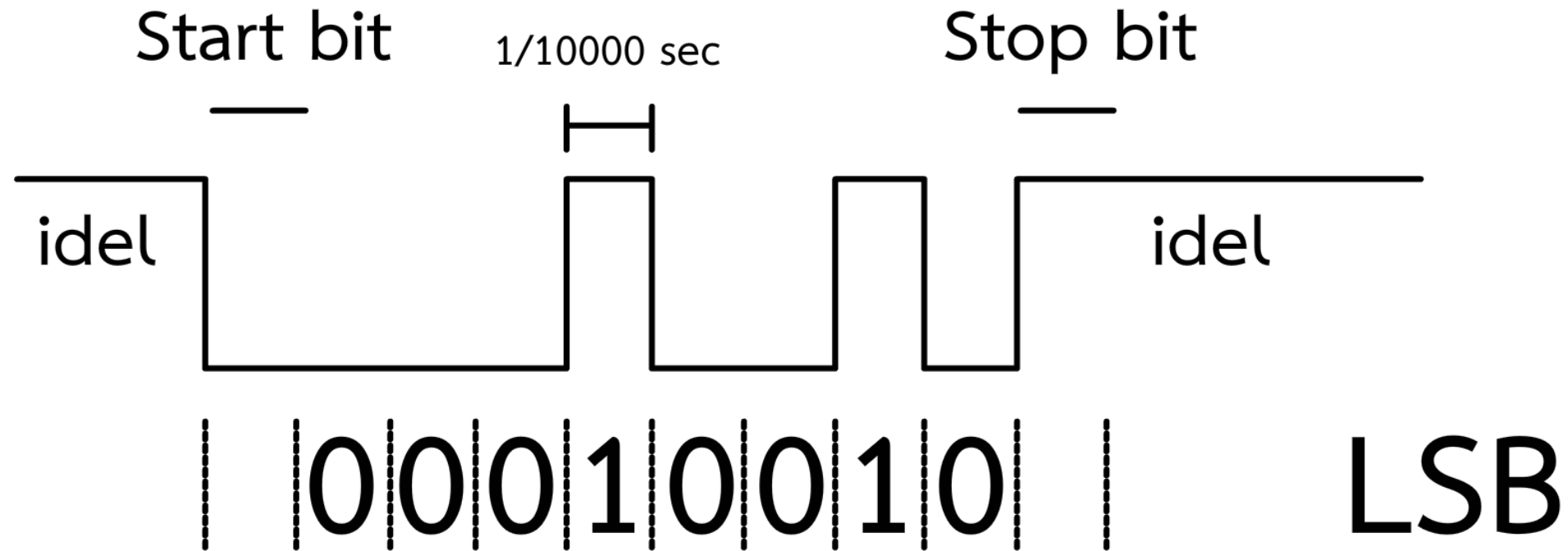


'H' -> 0b01001000 (MSB)

Serial communication

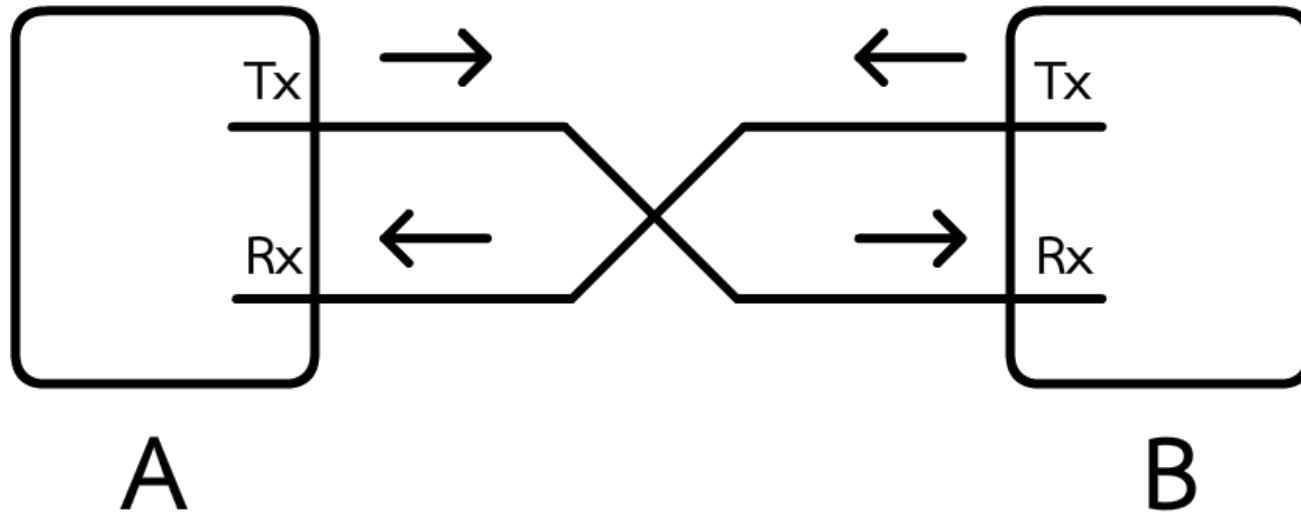
'H' -> 0b01001000 (MSB)

Baud rate = 10000 bps, Start bit = 1 baud, Stop bit = 1 baud



UART

Full duplex , One-to-one , Asynchronous

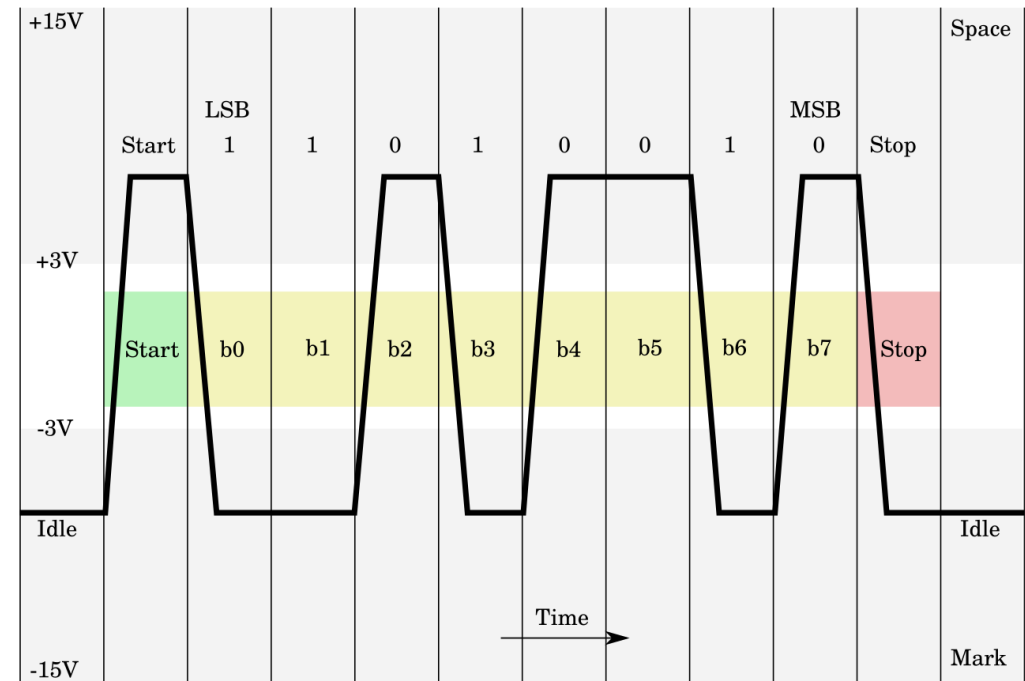


Rx connect to Tx
Tx connect to Rx
Same baud rate

Serial (UART) variant

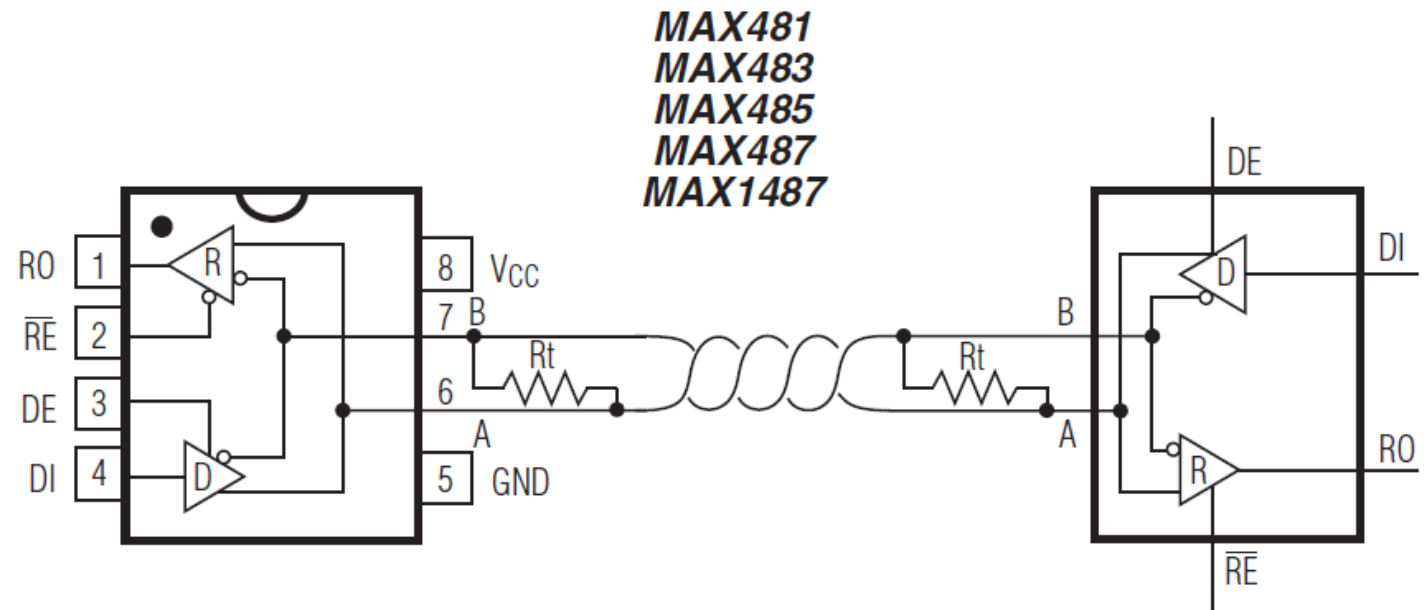
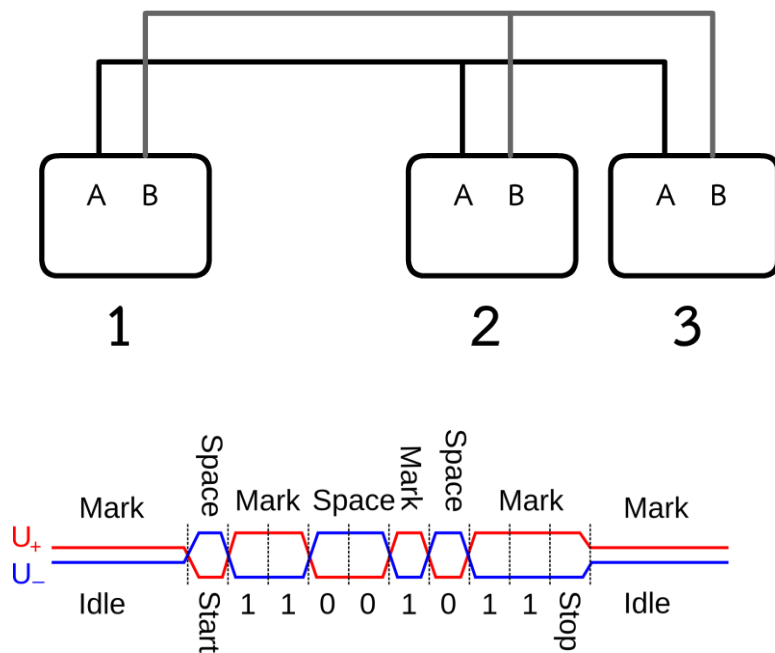
RS232 -> convert logic level to positive and negative (-5V -> 1, 5V -> 0) or (-15V -> 1, 15V -> 0)

Improve robustness (longer and faster)



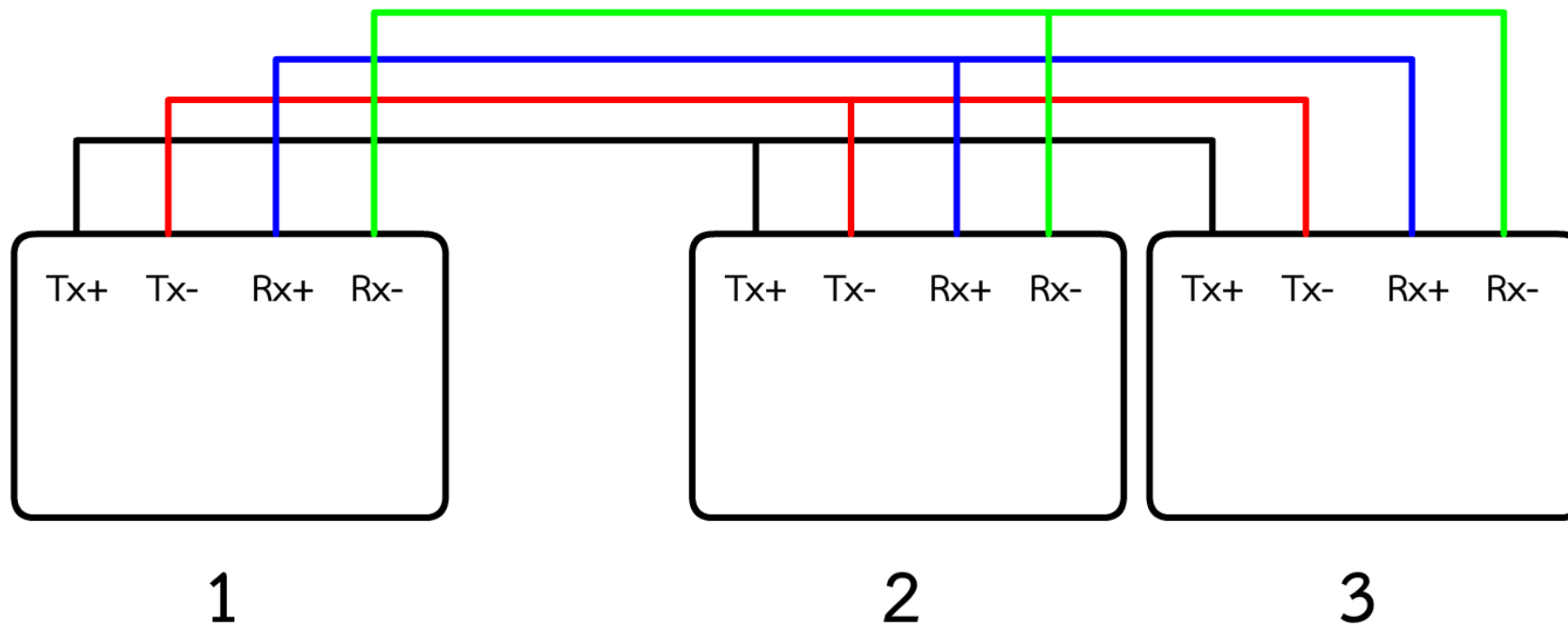
Serial (UART) variant

RS485-> using differential logic () but Half duplex , BUS



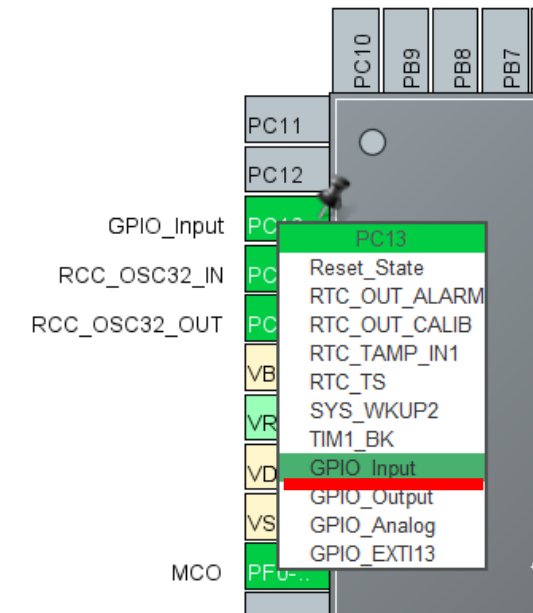
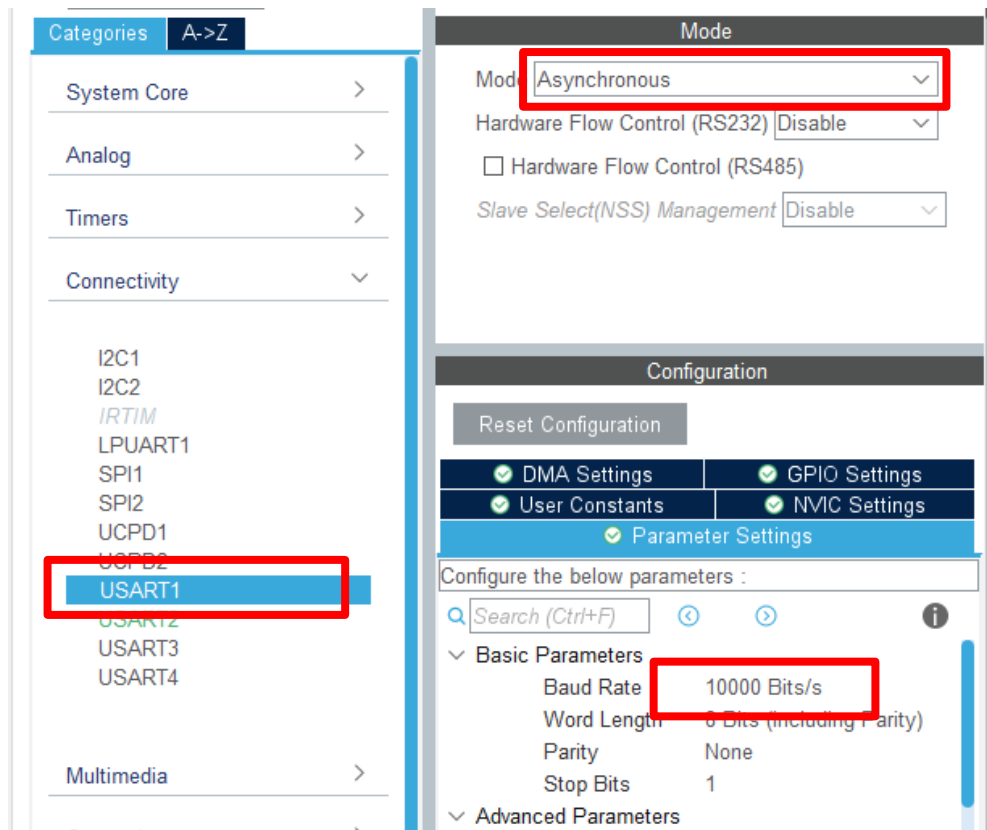
Serial (UART) variant

RS422-> Full duplex version of RS485 (add 1 pair of wire)



Communication programming sender

Sender is easy because device can send when every they want! (LAB1)



Set PC 13 to Input
Setup USART 1 in Asynchronous mode
Set baud rate to 10000 Bps (8bit)

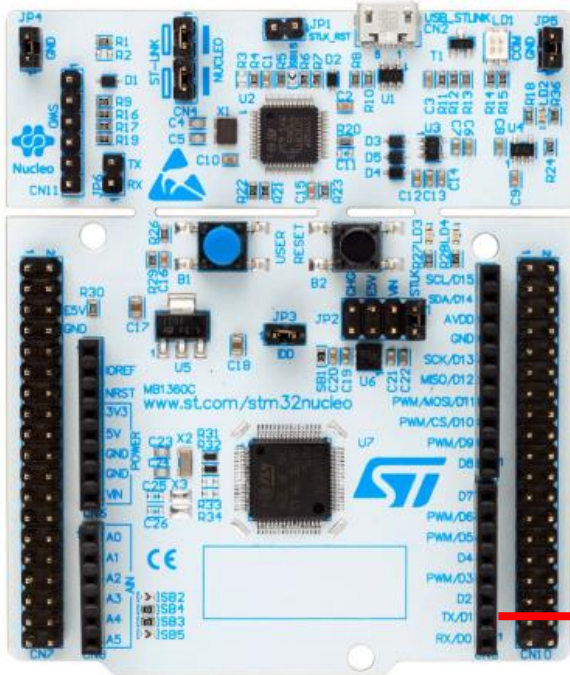
LAB1

```
99  /* Infinite loop */
100 /* USER CODE BEGIN WHILE */
101 while (1)
102 {
103     /* USER CODE END WHILE */
104
105     /* USER CODE BEGIN 3 */
106     unsigned char data[10] = "A"; // = 65 DEC , 0X41 , 0b01000001
107     while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
108         HAL_UART_Transmit(&huart1, data, 1, 1000);
109         data[0]++;
110         HAL_Delay(3000);
111     }
112     HAL_Delay(1000);
113 }
114 /* USER CODE END 3 */
115 }
116
```

```
/* USER CODE BEGIN 3 */
unsigned char data[10] = "A"; // = 65 DEC , 0X41 , 0b01000001
while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
    HAL_UART_Transmit(&huart1, data, 1, 1000);
    data[0]++;
    HAL_Delay(3000);
}
HAL_Delay(1000);
}
/* USER CODE END 3 */
```

Send data when button pressed and increase data 1 when hold button

Seeing the data



Measure data using Oscilloscope
USART1 TX -> PC4 (D1 Arduino)

Then hold button

PC4 (D1 Arduino)

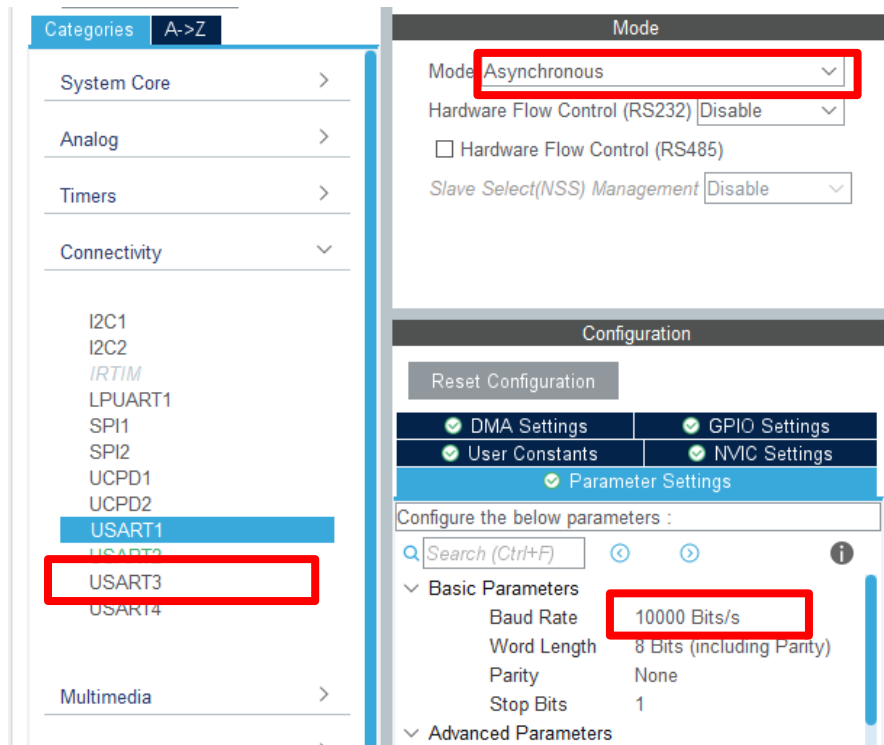
Oscilloscope

*Don't forget to connect GND

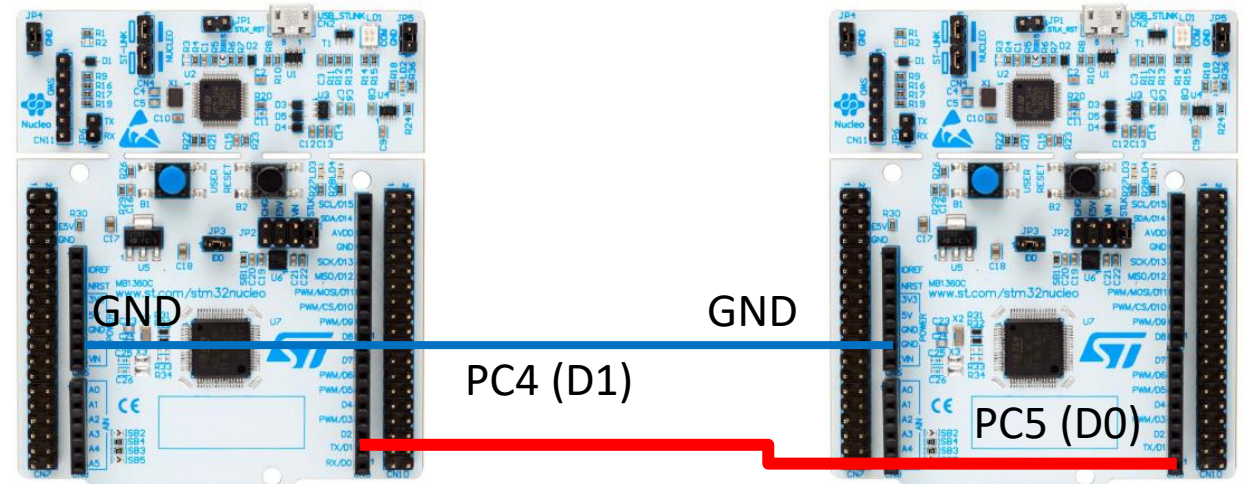
Serial Receiver

Sender is easy because device can send when every they want! (LAB1)

Receive is more complex because ,We don't know when the data will come. (LAB2)



Setup USART 1 in Asynchronous mode
Set baud rate to 10000 Bps (8bit)



Sender

Receiver

Lab 2 coding

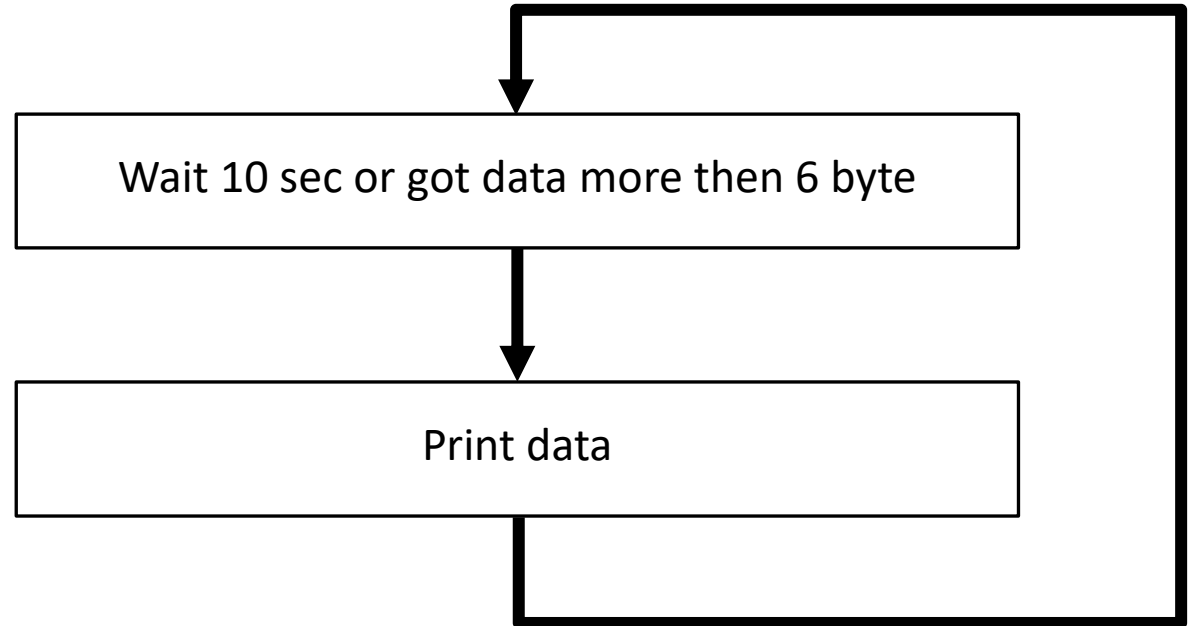
```
97  /* USER CODE END 2 */
98
99  /* Infinite loop */
100 /* USER CODE BEGIN WHILE */
101 while (1)
102 {
103     /* USER CODE END WHILE */
104
105     /* USER CODE BEGIN 3 */
106     unsigned char data[20] = "XXXXXX";
107     HAL_UART_Receive(&huart1, data, 6, 10000); // blocking receiving
108     HAL_UART_Transmit(&huart2, data, 6, 1000); // print out to console
109     HAL_UART_Transmit(&huart2, "\r\n", 2, 1000); // ending line
110     HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
111     HAL_Delay(1000);
112 }
113 /* USER CODE END 3 */
114 }
```

```
/* USER CODE BEGIN 3 */
unsigned char data[20] = "XXXXXX";
HAL_UART_Receive(&huart1, data, 6, 10000); // blocking receiving
HAL_UART_Transmit(&huart2, data, 6, 1000); // print out to console
HAL_UART_Transmit(&huart2, "\r\n", 2, 1000); // ending line
HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
HAL_Delay(1000);
}
/* USER CODE END 3 */
```

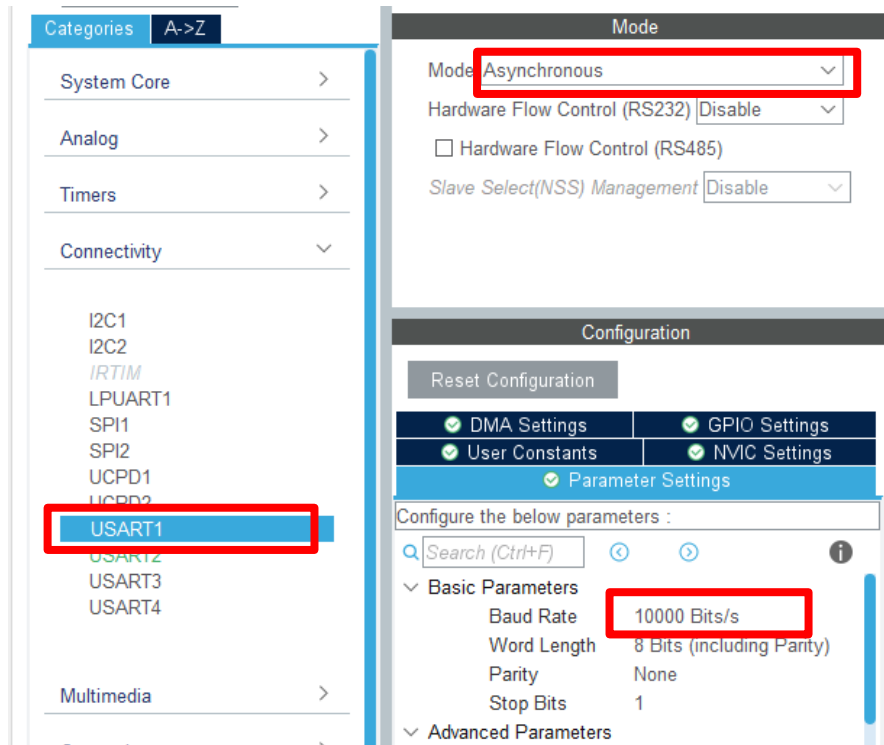
LAB2 blocking mode

```
XXXXXXXX
AXXXXX
BCDXXX
EFGHXX
IJKLXX
XXXXXXXX
ABCDXX
XXXXXXXX
XXXXXXXX
```

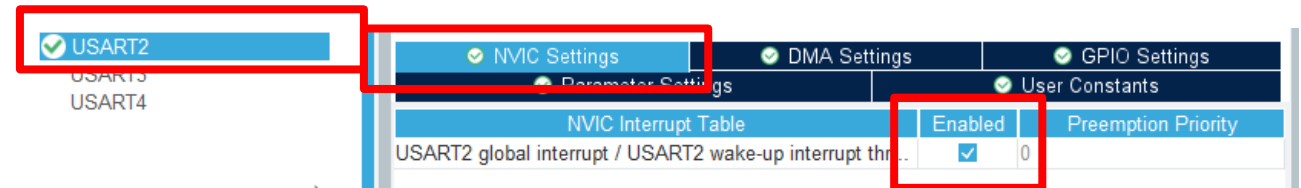
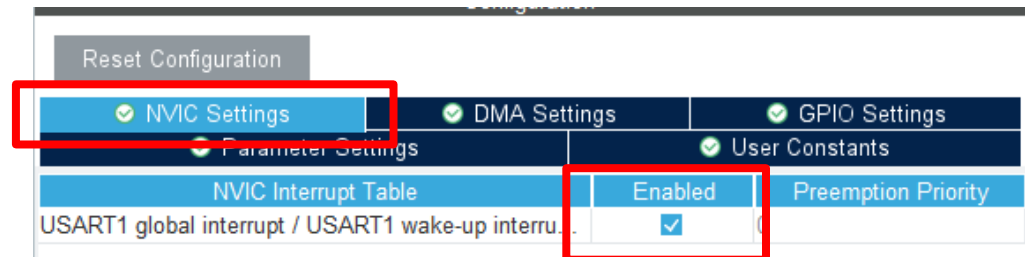
Wait 10 sec every display



LAB 3 interrupt mode



Setup USART 1 in Asynchronous mode
Set baud rate to 10000 Bps (8bit)
Enable interrupt (both USART1 AND USART2)



LAB 3 interrupt mode

```
59 /* Private user code -----
60 /* USER CODE BEGIN 0 */
61 unsigned char data[10] = "XXXXXX\r\n";
62 /* USER CODE END 0 */
63
64 /**
99  /* Infinite loop */
100 /* USER CODE BEGIN WHILE */
101 while (1)
102 {
103     /* USER CODE END WHILE */
104
105     /* USER CODE BEGIN 3 */
106     HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
107     HAL_Delay(1000);
108 }
109 /* USER CODE END 3 */
110 }
111
278 /* USER CODE BEGIN 4 */
279 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
280 {
281     if(huart == &huart1){
282         HAL_UART_Transmit_IT(&huart2, data, 8);
283         HAL_UART_Receive_IT(&huart1, data, 6);
284     }
285 }
286 /* USER CODE END 4 */
287
```

```
/* USER CODE BEGIN 0 */
unsigned char data[10] = "XXXXXX\r\n";
/* USER CODE END 0 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
    HAL_Delay(1000);
}
/* USER CODE END 3 */

/* USER CODE BEGIN 4 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    if(huart == &huart1){
        HAL_UART_Transmit_IT(&huart2, data, 8);
        HAL_UART_Receive_IT(&huart1, data, 6);
    }
}
/* USER CODE END 4 */
```

LAB 3 interrupt mode

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
}

```

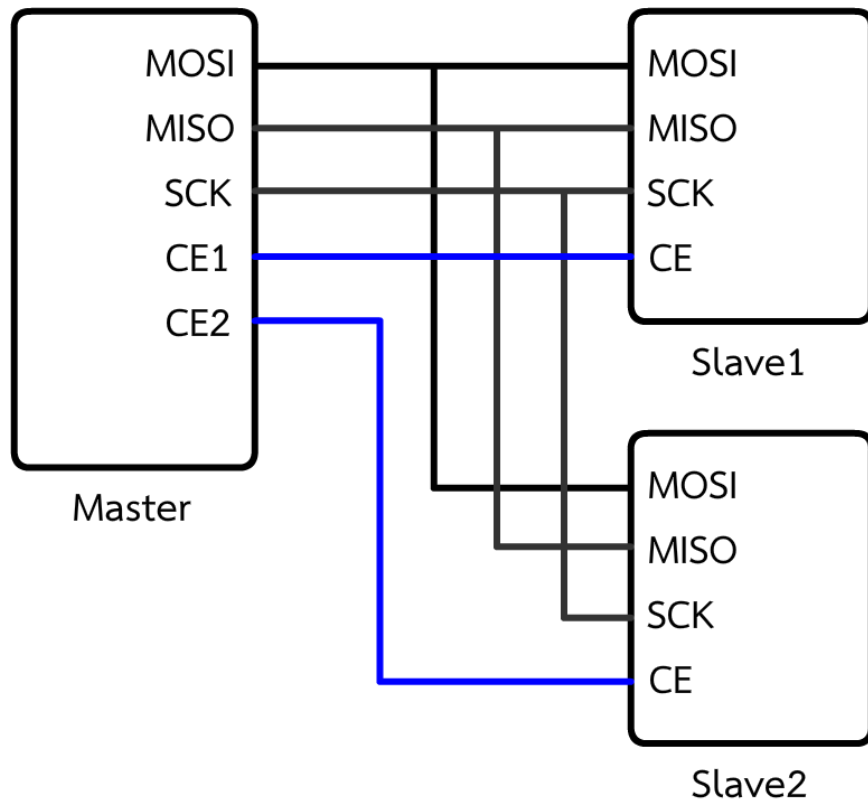
Called when received all byte

Non - blocking

FGHIJK
LMNOPQ
ABCDE
FGHIJK
|

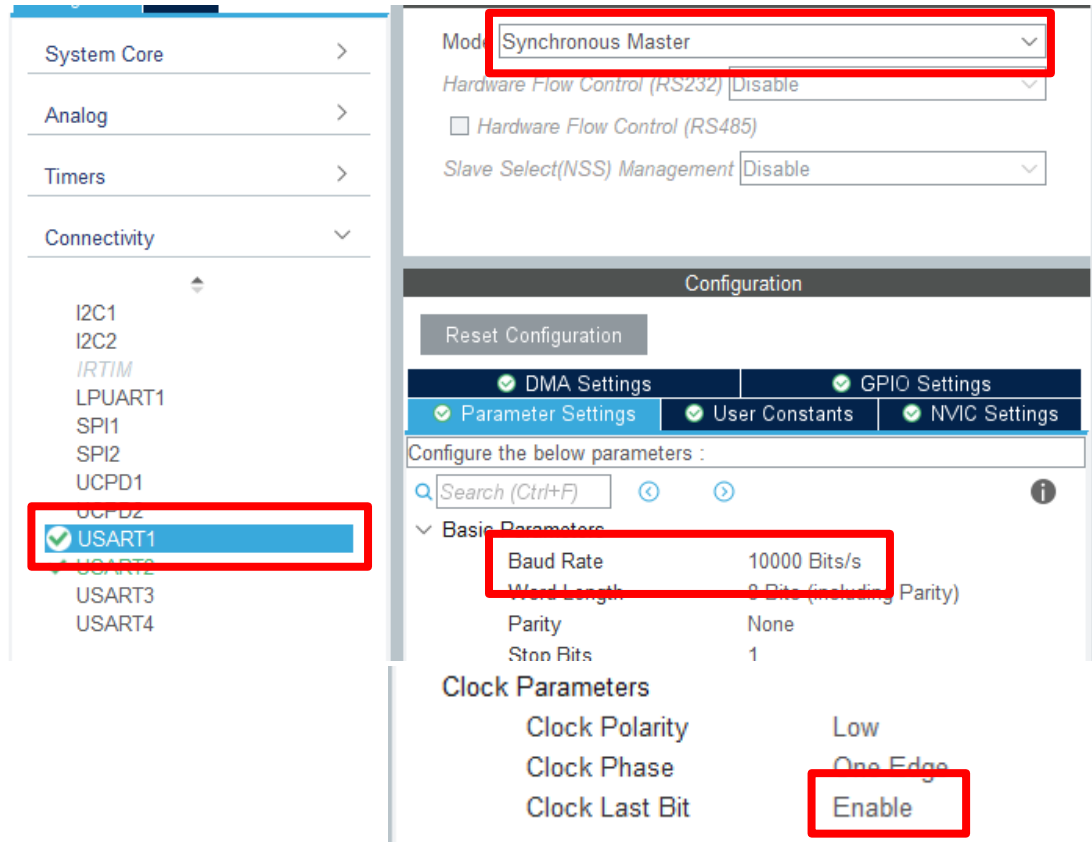
SPI

Serial but Synchronous and Master – Slave topology.

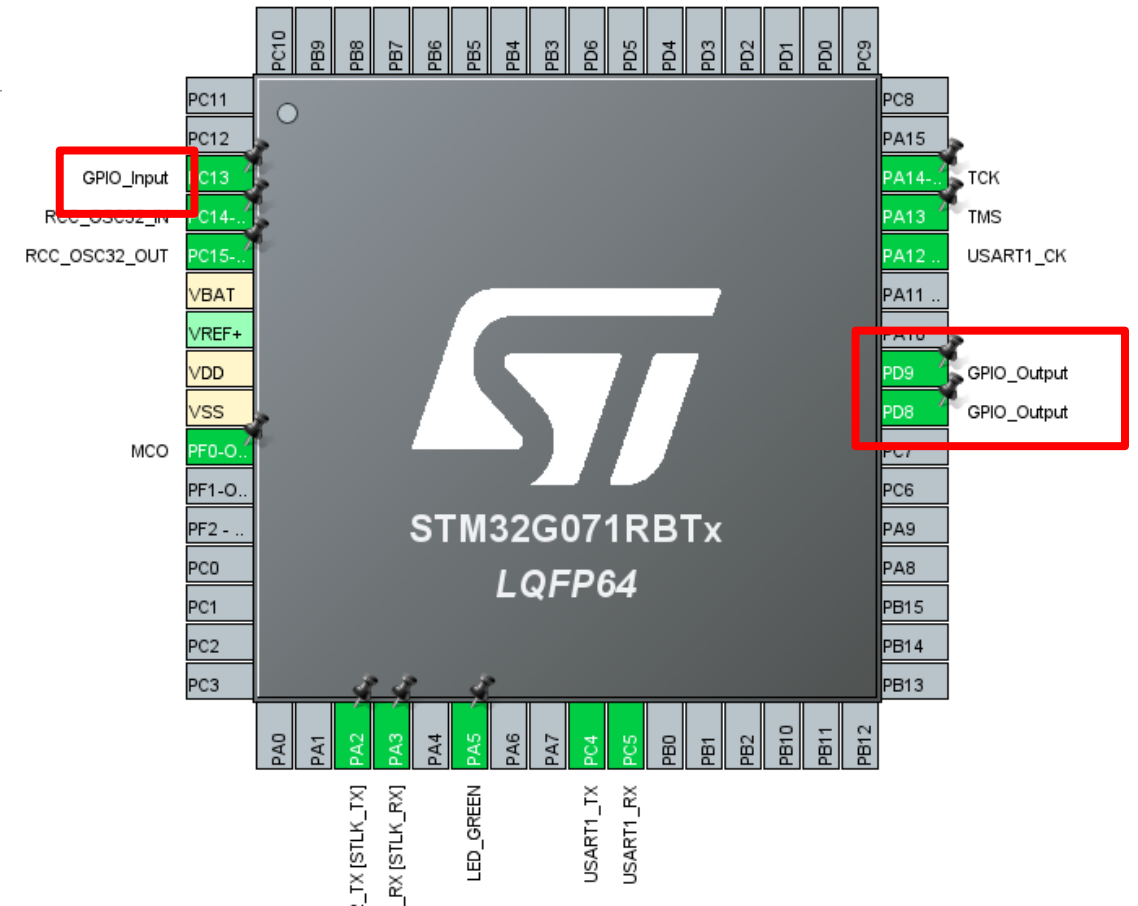


MOSI -> Master out SLAVE in
MISO -> MASTER in SLAVE out
SCK -> Clock signal from MASTER
CE -> chip select (select chip to communicate)

LAB4 SPI (MASTER)



Set USART1 to Synchronous Master (10000 bps) , Enable Clock Last bit



Set PD9,PD8 to GPIO output ,PC13 to input

LAB4 SPI (MASTER)

```
95  /* USER CODE BEGIN 2 */
96  unsigned char data[10] = "A";
97  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);
98  HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
99  /* USER CODE END 2 */
100
103 while (1)
104 {
105     /* USER CODE END WHILE */
106
107     /* USER CODE BEGIN 3 */
108     if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
109         HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 0);
110         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 1);
111         HAL_UART_Transmit(&husart1, data, 1, 1000);
112         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);
113     }
114     else{
115         HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 1);
116         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 1);
117         HAL_UART_Transmit(&husart1, data, 1, 1000);
118         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
119     }
120     HAL_Delay(500);
121     data[0]++;
122     if(data[0] > 'Z'){
123         data[0] = 'A';
124     }
125 }
126 /* USER CODE END 3 */
127 }
```

```
/* USER CODE BEGIN 2 */
unsigned char data[10] = "A";
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);
HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 0);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 1);
        HAL_UART_Transmit(&husart1, data, 1, 1000);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_8, 0);
    }
    else{
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 1);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 1);
        HAL_UART_Transmit(&husart1, data, 1, 1000);
        HAL_GPIO_WritePin(GPIOD, GPIO_PIN_9, 0);
    }
    HAL_Delay(500);
    data[0]++;
    if(data[0] > 'Z'){
        data[0] = 'A';
    }
}
/* USER CODE END 3 */
```

LAB5 SPI (Slave)

The screenshot shows the STM32CubeMX configuration interface. On the left, the 'Connectivity' category is expanded, and 'SPI1' is selected. The main panel shows the 'Mode' dropdown set to 'Full-Duplex Slave'. Below this, the 'Configuration' section is visible, with tabs for 'Parameter Settings', 'User Constants', and 'NVIC Settings'. The 'Parameter Settings' tab is active, showing a search bar and a list of parameters. The 'Basic Parameters' section is expanded, showing 'Frame Format' set to 'Motorola', 'Data Size' set to '8 Bits', and 'First Bit' set to 'LSB First'. The 'Clock Parameters' section is also expanded, showing 'Clock Polarity (CPOL)' set to 'Low' and 'Clock Phase (CPHA)' set to '1 Edge'. The 'Advanced Parameters' section is partially visible, showing 'CRC Calculation' set to 'Disabled' and 'NSS Signal Type' set to 'Input Hardware'.

DMA Settings		GPIO Settings	
Parameter Settings	User Constants	NVIC Settings	
NVIC Interrupt Table		Enabled	Preemption Priority
SPI1 global interrupt		<input checked="" type="checkbox"/>	0

DMA Settings		GPIO Settings	
Parameter Settings	User Constants	NVIC Settings	
NVIC Interrupt Table		Ena...	Preemption
USART2 global interrupt / USART2 wake-up interrupt...		<input checked="" type="checkbox"/>	0

Set SPI1 to Full-Duplex Slave
Set 8 bit LSB
Enable SPI1 and UART2 Interrupt

*not necessary to assign baud rate why?

LAB5 SPI (Slave)

```
60 /* Private user code -----
61 /* USER CODE BEGIN 0 */
62 unsigned char data[10] = "a\r\n";
63 /* USER CODE END 0 */
64
65 /**
66  * @brief The application entry point.
```

```
95 MX_SPI1_Init();
96 /* USER CODE BEGIN 2 */
97 HAL_SPI_Receive_IT(&hspi1, data, 1);
98 /* USER CODE END 2 */
99
100 /* Infinite loop */
```

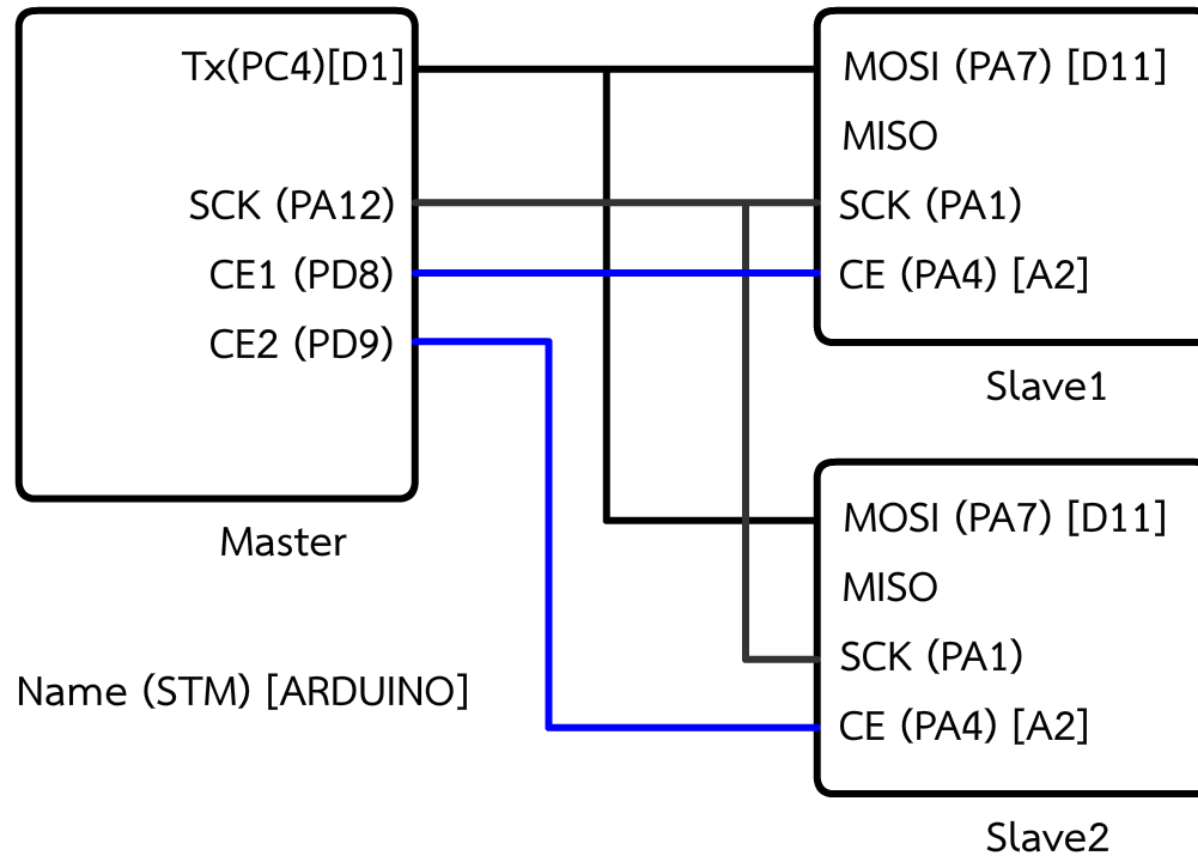
```
268 /* USER CODE BEGIN 4 */
269 void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi)
270 {
271     if(hspi == &hspi1){
272         HAL_UART_Transmit_IT(&huart2, data, 3);
273         HAL_SPI_Receive_IT(&hspi1, data, 1);
274     }
275 }
276 /* USER CODE END 4 */
277
```

```
/* USER CODE BEGIN 0 */
unsigned char data[10] = "a\r\n";
/* USER CODE END 0 */
```

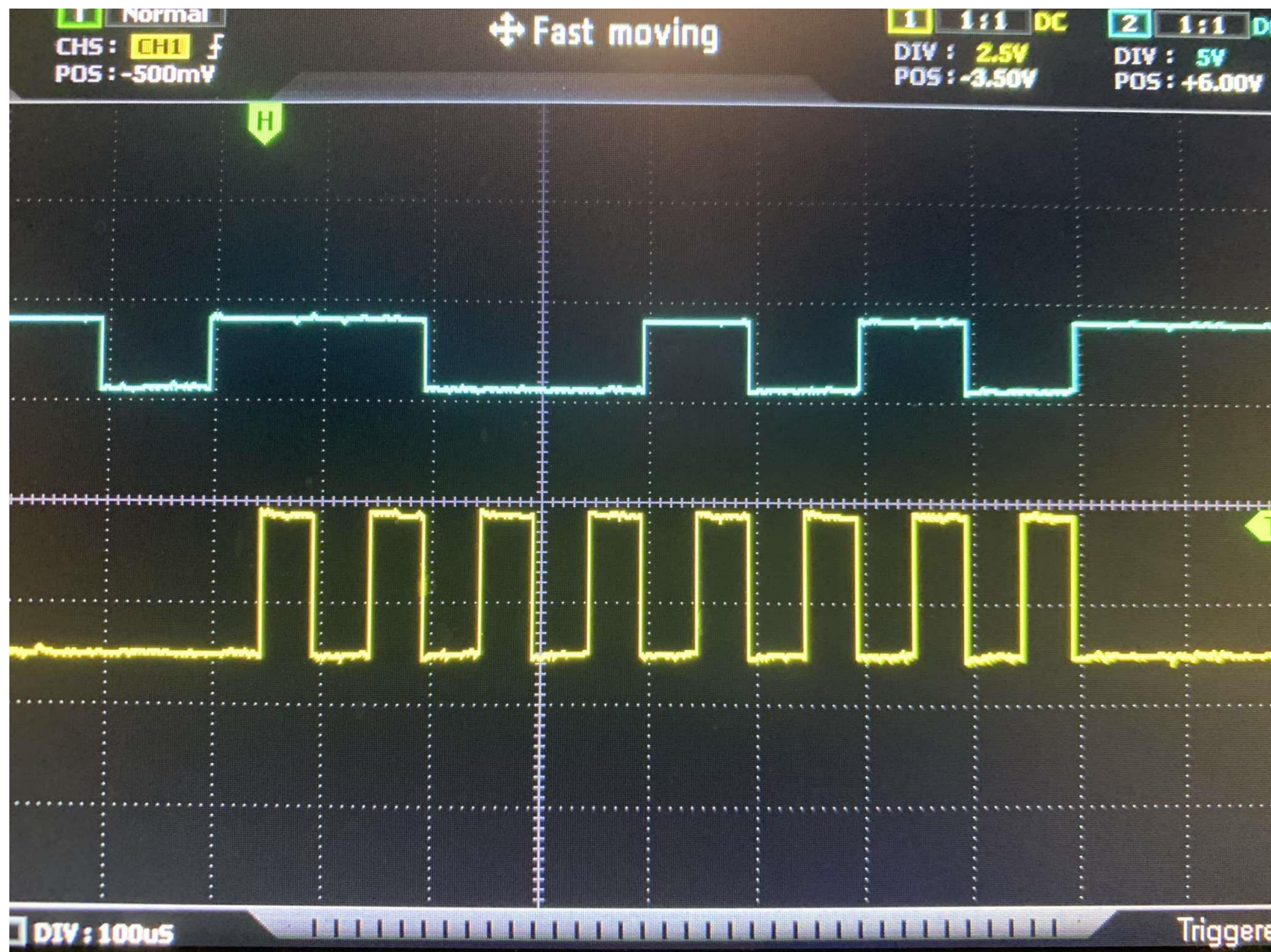
```
/* USER CODE BEGIN 2 */
HAL_SPI_Receive_IT(&hspi1, data, 1);
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN 4 */
void HAL_SPI_RxCpltCallback(SPI_HandleTypeDef *hspi)
{
    if(hspi == &hspi1){
        HAL_UART_Transmit_IT(&huart2, data, 3);
        HAL_SPI_Receive_IT(&hspi1, data, 1);
    }
}
/* USER CODE END 4 */
```


LAB 4 & 5 Topology

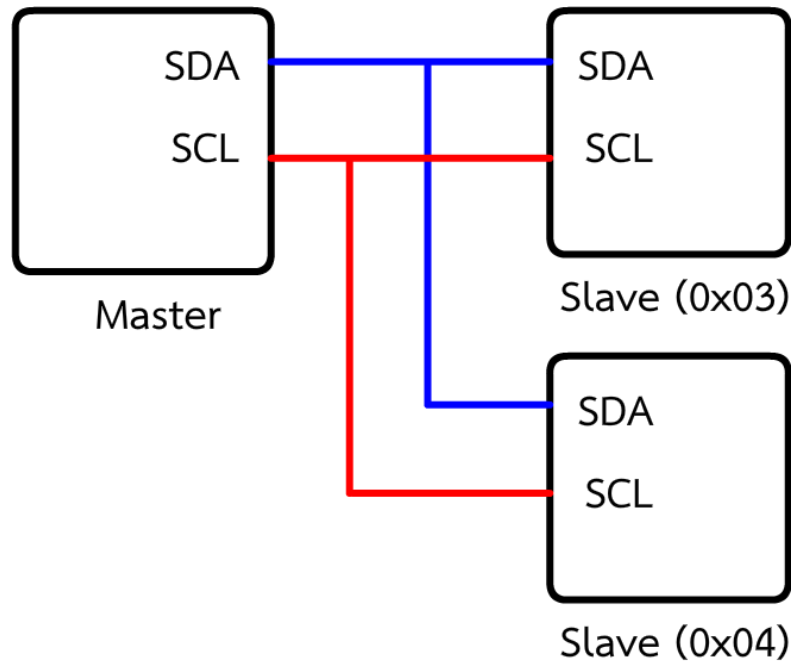


Baud rate is defined by master



I2C

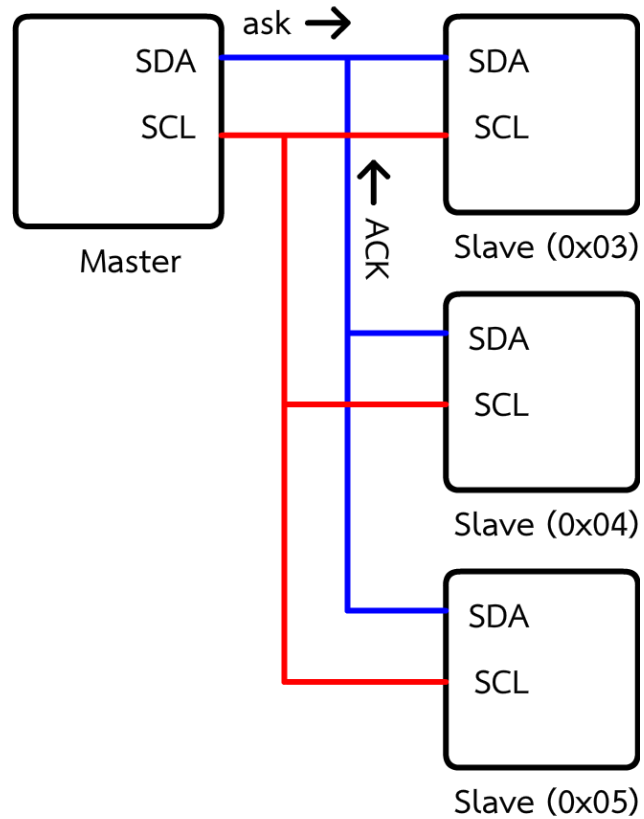
SPI but use address in communication to select chip (use only 2 wire)



SDA -> DATA
SCL -> CLOCK

Address was assigned to every device except master
Select slave by address

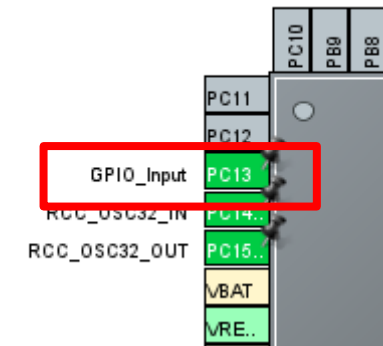
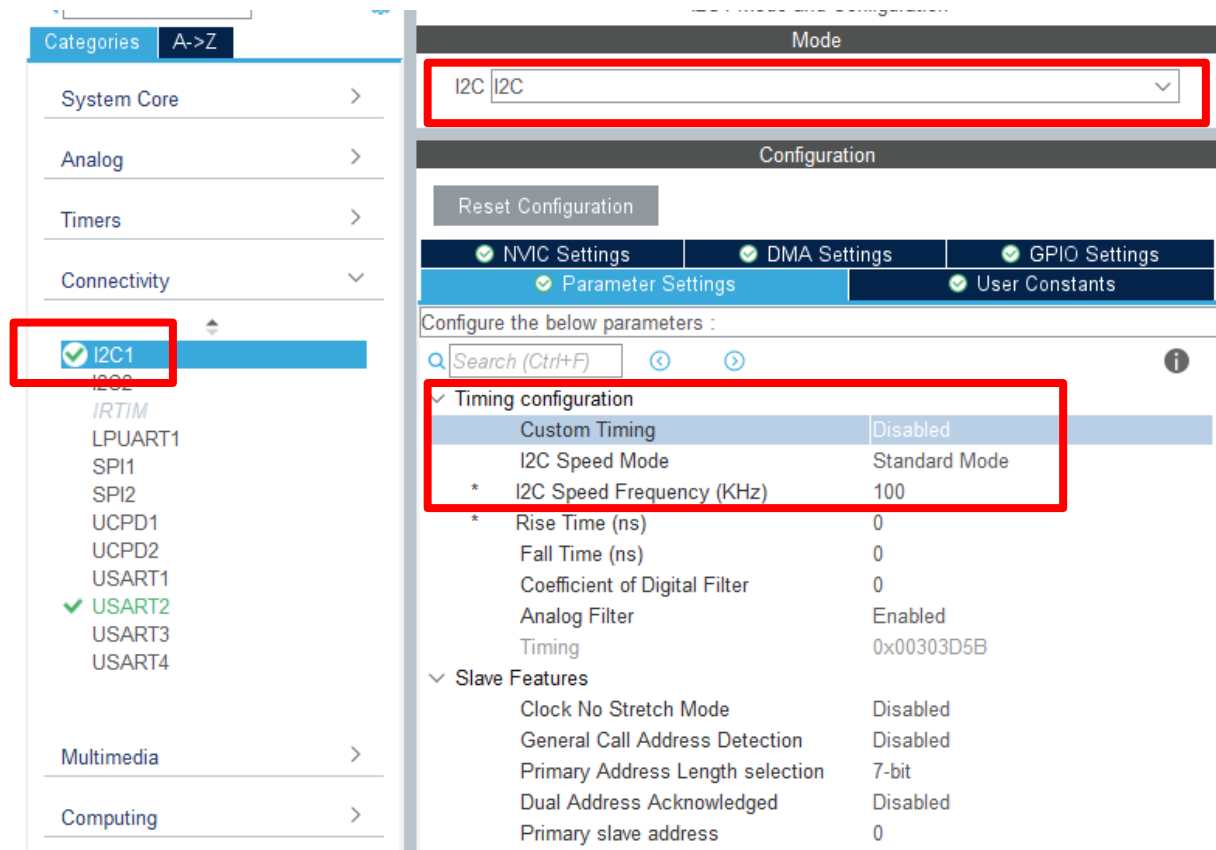
I2C



Connect start by master ask for device xx
If there are device address xx , device will
send ACK back to BUS

ACK = acknowledgement = สัญญาณตอบกลับ
NACK = negative-acknowledgement = สัญญาณตอบกลับแบบลบ

LAB 6 I2C (MASTER)



Enable I2C set standard frequency 100000 Hz
Set address to 0 (MASTER)
Set PC13 to Input

LAB 6 I2C (MASTER)

Set pullup PA9(SCL) , PA10(SDA)

The screenshot shows the STM32CubeMX configuration interface. On the left, the 'System Core' tree has 'GPIO' highlighted with a red box. On the right, the 'Configuration' tab is active, showing a table of pin configurations. The 'GPIO P...' column for PA9 and PA10 is highlighted with a red box. Below the table, the 'PA9 Configuration' section shows the 'GPIO Pull-up/Pull-down' dropdown set to 'Pull-up', which is also highlighted with a red box.

Pin...	Signal...	GPIO...	GPIO...	GPIO P...	Maxim...	Fast...	User L...	Modified
PA9	I2C1_...	n/a	Altern...	Pull-up	Low	n/a		✓
PA10	I2C1_...	n/a	Altern...	Pull-up	Low	n/a		✓

PA9 Configuration :

GPIO mode: Alternate Function Open Drain

GPIO Pull-up/Pull-down: Pull-up

Maximum output speed: Low

User Label:

LAB 6 I2C (MASTER)

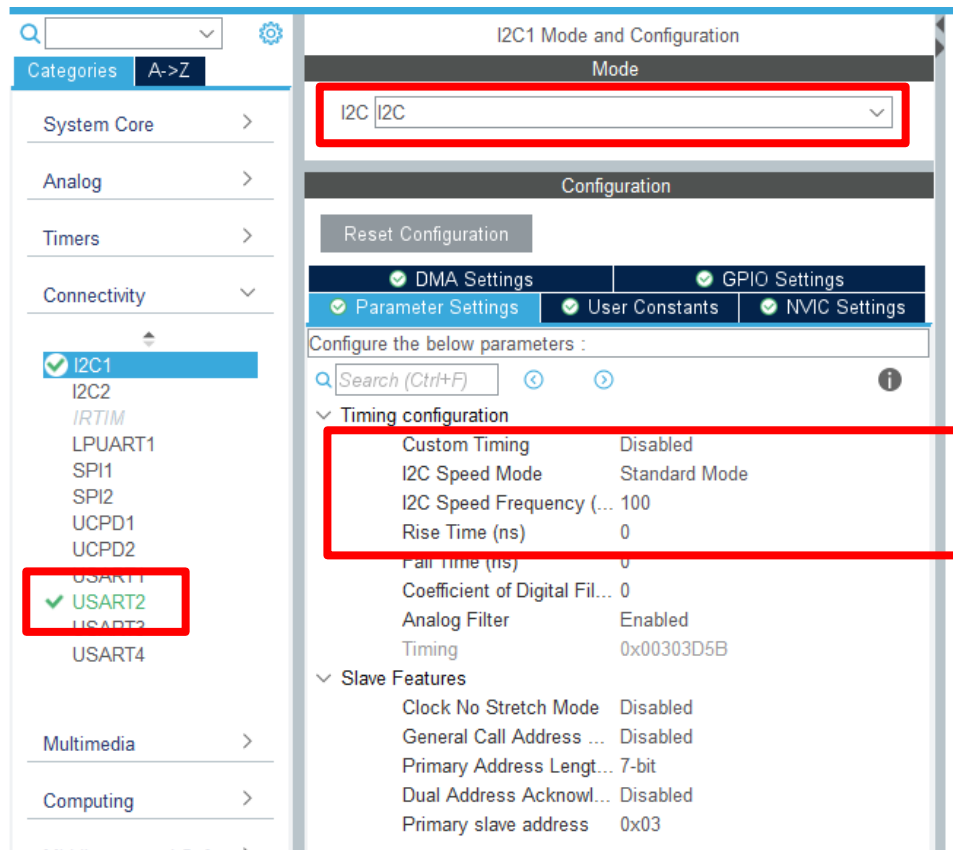
```
97  /* USER CODE BEGIN 2 */
98  unsigned char data[10] = "haruhi";
99  /* USER CODE END 2 */
100
101  /* Infinite loop */
102  /* USER CODE BEGIN WHILE */
103  while (1)
104  {
105      /* USER CODE END WHILE */
106
107      /* USER CODE BEGIN 3 */
108      if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
109          HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 0);
110          HAL_I2C_Master_Transmit(&hi2c1, 0x03 << 1, data, 6, 1000);
111      }
112      else{
113          HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 1);
114          HAL_I2C_Master_Transmit(&hi2c1, 0x04 << 1, data, 6, 1000);
115      }
116      HAL_Delay(2000);
117      data[0]++;
118      if(data[0] > 'z'){
119          data[0] = 'a';
120      }
121  }
122  /* USER CODE END 3 */
```

```
/* USER CODE BEGIN 2 */
unsigned char data[10] = "haruhi";
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    if(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == 0){
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 0);
        HAL_I2C_Master_Transmit(&hi2c1, 0x03 << 1, data, 6, 1000);
    }
    else{
        HAL_GPIO_WritePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin, 1);
        HAL_I2C_Master_Transmit(&hi2c1, 0x04 << 1, data, 6, 1000);
    }
    HAL_Delay(2000);
    data[0]++;
    if(data[0] > 'z'){
        data[0] = 'a';
    }
}
/* USER CODE END 3 */
```

LAB 7 I2C Slave



✔ NVIC Settings	✔ DMA Settings	✔ GPIO Settings
✔ Parameter Settings	✔ User Constants	
NVIC Interrupt Table	Enabled	Preemption Priority
I2C1 event global interrupt / I2C1 wake-up inter...	✔	0

✔ NVIC Settings	✔ DMA Settings	✔ GPIO Settings
✔ Parameter Settings	✔ User Constants	
NVIC Interrupt Table		Enab... Preemption Pr...
USART2 global interrupt / USART2 wake-up interrupt throug...		<input checked="" type="checkbox"/> 0

Enable I2C set standard frequency 100000 Hz
Set address to 0x03 and 0x04
Enable I2C1 and UART2 interrupt

LAB 7 I2C Slave

Set pullup PA9(SCL) , PA10(SDA)

The screenshot shows the STM32CubeMX configuration interface. On the left, the 'System Core' tree has 'GPIO' highlighted with a red box. On the right, the 'Configuration' tab is active, showing a table of pin configurations. The table has columns: Pin, Signal, GPIO, GPIO Mode, GPIO Pull-up/Pull-down, Maximum output speed, Fast mode, User Label, and Modified. The rows for PA9 and PA10 are highlighted, with PA9's 'GPIO Pull-up/Pull-down' set to 'Pull-up' (indicated by a red box). Below the table, the 'PA9 Configuration' section shows 'GPIO mode' set to 'Alternate Function Open Drain' and 'GPIO Pull-up/Pull-down' set to 'Pull-up' (indicated by a red box).

Pin	Signal	GPIO	GPIO Mode	GPIO Pull-up/Pull-down	Maximum output speed	Fast mode	User Label	Modified
PA9	I2C1_...	n/a	Alternate...	Pull-up	Low	n/a		✓
PA10	I2C1_...	n/a	Alternate...	Pull-up	Low	n/a		✓

PA9 Configuration :

GPIO mode: Alternate Function Open Drain

GPIO Pull-up/Pull-down: Pull-up

Maximum output speed: Low

User Label:

LAB 7 I2C Slave

```
/* USER CODE BEGIN 0 */
unsigned char data[10] = "A\r\n";
/* USER CODE END 0 */
```

```
/* USER CODE BEGIN 2 */
HAL_I2C_EnableListen_IT(&hi2c1);
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN 4 */
void HAL_I2C_ListenCpltCallback (I2C_HandleTypeDef *hi2c)
{
    HAL_I2C_EnableListen_IT(hi2c);
}

extern void HAL_I2C_AddrCallback(I2C_HandleTypeDef *hi2c, uint8_t TransferDirection, uint16_t AddrMatchCode)
{
    if(TransferDirection == I2C_DIRECTION_TRANSMIT) // if the master wants to transmit the data
    {
        HAL_I2C_Slave_Sequential_Receive_IT(hi2c, data, 6, I2C_FIRST_AND_LAST_FRAME);
    }
    else // master requesting the data is not supported yet
    {
        Error_Handler();
    }
}

void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c)
{
    if(hi2c == &hi2c1){
        HAL_UART_Transmit_IT(&huart2, data, 6);
    }
}

void HAL_I2C_ErrorCallback(I2C_HandleTypeDef *hi2c)
{
    HAL_I2C_EnableListen_IT(hi2c);
}

/* USER CODE END 4 */
```

I2C topology

