LAB: USART - LED, Bluetooth

Date: 2023-11-17

Author/Partner: Eunchan Kim / Heejun Lee

Demo Video: Problem 2

Problem 3

Introduction

In this lab, we will learn how to configure and use 'USART(Universal synchronous asynchronous receiver transmitter)' of MCU. Then, we will learn how to communicate between your PC and MCU and MCU to another MCU with wired serial communication.

- **Mission 1**: Control LED(LD2) of each other MCU
- **Mission 2**: Run DC motors with Bluetooth

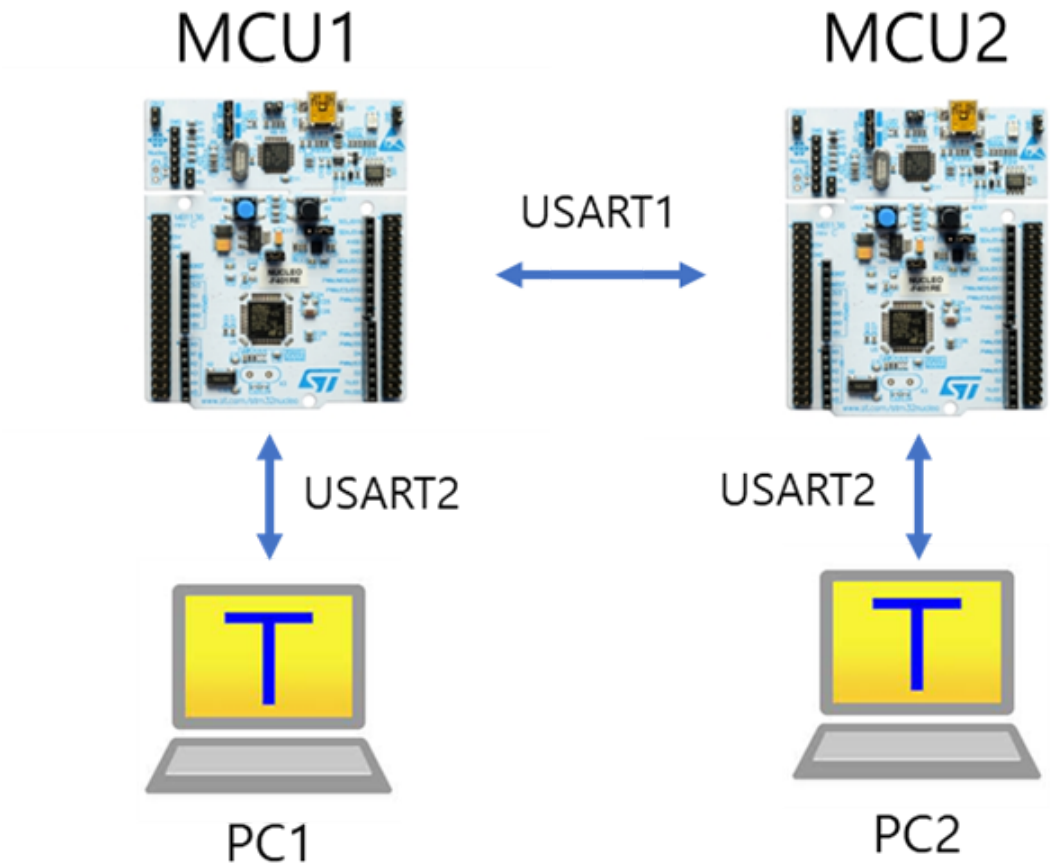


Figure1. Overview of MCU-PC communication

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - DC motor, DC motor driver(L9110s),
  - Bluetooth Module(HC-06),

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: EC HAL library

ecUART.c

A library has been completed according to the principle of USART communication. By using this, it is possible for the desired pins to be set as RX and TX, enabling the desired communication to be carried out.

```
#include "ecUART.h"
#include <math.h>

// ********************** DO NOT MODIFY HERE ***************************
//
// Implement a dummy __FILE struct, which is called with the FILE structure.
//#ifndef __stdio_h
struct __FILE {
    //int dummy;
        int handle;

};

FILE __stdout;
FILE __stdin;
//#endif

// Retarget printf() to USART2
int fputc(int ch, FILE *f) {
  uint8_t c;
  c = ch & 0x00FF;
  USART_write(USART2, (uint8_t *)&c, 1);
  return(ch);
}

// Retarget getchar()/scanf() to USART2
int fgetc(FILE *f) {
  uint8_t rxByte;
  rxByte = USART_read(USART2);
  return rxByte;
}


/*================ private functions ================*/
```

```c
void USART_write(USART_TypeDef * USARTx, uint8_t *buffer, uint32_t nBytes) {
    // TXE is set by hardware when the content of the TDR
    // register has been transferred into the shift register.
    int i;
    for (i = 0; i < nBytes; i++) {
        // wait until TXE (TX empty) bit is set
        while (!(USARTx->SR & USART_SR_TXE));
        // Writing USART_DR automatically clears the TXE flag
        USARTx->DR = buffer[i] & 0xFF;
        USART_delay(300);
    }
    // wait until TC bit is set
    while (!(USARTx->SR & USART_SR_TC));
    // TC bit clear
    USARTx->SR &= ~USART_SR_TC;

}

uint32_t is_USART_RXNE(USART_TypeDef * USARTx){
    return (USARTx->SR & USART_SR_RXNE);
}


uint8_t USART_read(USART_TypeDef * USARTx){
    // Wait until RXNE (RX not empty) bit is set by HW -->Read to read
    while ((USARTx->SR & USART_SR_RXNE) != USART_SR_RXNE);
    // Reading USART_DR automatically clears the RXNE flag
    return ((uint8_t)(USARTx->DR & 0xFF));
}

void USART_setting(USART_TypeDef* USARTx, GPIO_TypeDef* GPIO_TX, int pinTX,
GPIO_TypeDef* GPIO_RX, int pinRX, uint32_t baud){
//1. GPIO Pin for TX and RX
    // Enable GPIO peripheral clock
    // Alternative Function mode selection for Pin_y in GPIOx
    // AF, Push-Pull, No PUPD, High Speed
    GPIO_init(GPIO_TX, pinTX, EC_AF);
    GPIO_otype(GPIO_TX, pinTX, EC_PUSH_PULL);
    GPIO_pupd(GPIO_TX, pinTX, EC_NONE);
    GPIO_ospeed(GPIO_TX, pinTX, EC_HIGH);

    GPIO_init(GPIO_RX, pinRX, EC_AF);
    GPIO_otype(GPIO_RX, pinRX, EC_PUSH_PULL);
    GPIO_pupd(GPIO_RX, pinRX, EC_NONE);
    GPIO_ospeed(GPIO_RX, pinRX, EC_HIGH);

    // Set Alternative Function Register for USARTx.
    // AF7 - USART1,2
    // AF8 - USART6
    if (USARTx == USART6){
        // USART_TX GPIO AFR
        if (pinTX < 8) GPIO_TX->AFR[0] |= 8 << (4*pinTX);
        else                    GPIO_TX->AFR[1] |= 8 << (4*(pinTX-8));
        // USART_RX GPIO AFR
        if (pinRX < 8) GPIO_RX->AFR[0] |= 8 << (4*pinRX);
        else                    GPIO_RX->AFR[1] |= 8 << (4*(pinRX-8));
    }
    else{   //USART1,USART2
```

```
        // USART_TX GPIO AFR
        if (pinTX < 8) GPIO_TX->AFR[0] |= 7 << (4*pinTX);
        else                    GPIO_TX->AFR[1] |= 7 << (4*(pinTX-8));
        // USART_RX GPIO AFR
        if( pinRX < 8) GPIO_RX->AFR[0] |= 7 << (4*pinRX);
        else                    GPIO_RX->AFR[1] |= 7 << (4*(pinRX-8));
    }


//2. USARTx (x=2,1,6) configuration
    // Enable USART peripheral clock
    if (USARTx == USART1)
        RCC->APB2ENR |= RCC_APB2ENR_USART1EN;   // Enable USART 1 clock (APB2
clock: AHB clock = 84MHz)
    else if(USARTx == USART2)
        RCC->APB1ENR |= RCC_APB1ENR_USART2EN;   // Enable USART 2 clock (APB1
clock: AHB clock/2 = 42MHz)
    else
        RCC->APB2ENR |= RCC_APB2ENR_USART6EN;   // Enable USART 6 clock (APB2
clock: AHB clock = 84MHz)

    // Disable USARTx.
    USARTx->CR1  &= ~USART_CR1_UE;                      // USART disable

    // No Parity / 8-bit word length / Oversampling x16
    USARTx->CR1 |= 0 << 10;         // No parrity bit
    USARTx->CR1 |= 0 << 12;         // M: 0 = 8 data bits, 1 start bit
    USARTx->CR1 &= ~USART_CR1_OVER8;    // 0 = oversampling by 16 (to reduce RF
noise)
    // Configure Stop bit
    USARTx->CR2 |= 0 << 12;         // 1 stop bit

    // CSet Baudrate to 9600 using APB frequency (42MHz)
    // If oversampling by 16, Tx/Rx baud = f_CK / (16*USARTDIV),
    // If oversampling by 8,  Tx/Rx baud = f_CK / (8*USARTDIV)
    // USARTDIV = 42MHz/(16*9600) = 237.4375

    UART_baud(USARTx, baud);

    // Enable TX, RX, and USARTx
    USARTx->CR1 |= (USART_CR1_RE | USART_CR1_TE);       // Transmitter and
Receiver enable
    USARTx->CR1 |= USART_CR1_UE;                                        // USART
enable


// 3. Read USARTx Data (Interrupt)
    // Set the priority and enable interrupt
    USARTx->CR1 |= USART_CR1_RXNEIE;                // Received Data Ready to be
Read Interrupt
    if (USARTx == USART1){
        NVIC_SetPriority(USART1_IRQn, 1);           // Set Priority to 1
        NVIC_EnableIRQ(USART1_IRQn);                // Enable interrupt of
USART2 peripheral
    }
    else if (USARTx == USART2){
        NVIC_SetPriority(USART2_IRQn, 1);           // Set Priority to 1
```

```
        NVIC_EnableIRQ(USART2_IRQn);                // Enable interrupt of
USART2 peripheral
    }
    else {
// if(USARTx==USART6)
        NVIC_SetPriority(USART6_IRQn, 1);           // Set Priority to 1
        NVIC_EnableIRQ(USART6_IRQn);                // Enable interrupt of
USART2 peripheral
    }
    USARTx->CR1 |= USART_CR1_UE;                            // USART enable
}


void UART_baud(USART_TypeDef* USARTx, uint32_t baud){
    // Disable USARTx.
    USARTx->CR1  &= ~USART_CR1_UE;                      // USART disable
    USARTx->BRR = 0;

// Configure Baud-rate
    float fCK = 84000000;                                       //
if(USARTx==USART1 || USARTx==USART6), APB2
    if(USARTx == USART2) fCK =fCK/2;      // APB1

// Method 1
    float USARTDIV = (float) fCK/(16*baud);
    uint32_t mantissa = (uint32_t)USARTDIV;
    uint32_t fraction = round(USARTDIV-mantissa)*16;
    USARTx->BRR |= (mantissa << 4)|fraction;

    // Enable TX, RX, and USARTx
    USARTx->CR1 |= USART_CR1_UE;

}

void USART_delay(uint32_t us) {
   uint32_t time = 100*us/7;
   while(--time);
}


/*================ Use functions ================*/
void UART1_init(void){
    // ********************* USART 1 ************************
    // PA_9 = USART1_TX (default)   // PB_6  (option)
    // PA_10 = USART1_RX (default)  // PB_3 (option)
    // APB2
    // ******************************************************
    USART_setting(USART1, GPIOA, 9, GPIOA, 10, 9600);
}
void UART2_init(void){
    // ********************* USART 2 ************************
    // PA2 = USART2_TX
    // PA3 = USART2_RX
    // Alternate function(AF7), High Speed, Push pull, Pull up
    // ******************************************************
    USART_setting(USART2, GPIOA, 2, GPIOA, 3, 9600);
}
```

```
void UART1_baud(uint32_t baud){
    UART_baud(USART1, baud);
}
void UART2_baud(uint32_t baud){
    UART_baud(USART2, baud);
}

void USART1_write(uint8_t* buffer, uint32_t nBytes){
    USART_write(USART1, buffer, nBytes);
}

void USART2_write(uint8_t* buffer, uint32_t nBytes){
    USART_write(USART2, buffer, nBytes);
}
uint8_t USART1_read(void){
    return USART_read(USART1);
}

uint8_t USART2_read(void){
    return USART_read(USART2);
}

uint32_t is_USART1_RXNE(void){
    return is_USART_RXNE(USART1);
}
uint32_t is_USART2_RXNE(void){
    return is_USART_RXNE(USART2);
}
```

Problem 2: Communicate MCU1-MCU2 using RS-232

Procedure

1. Create a project using the new source file(ecUART.h, ecUART.c)
2. Connect each MCUs to each PC with **USART 2** via USB cable (ST-Link)

- MCU1-PC1, MCU2-PC2

3. Connect MCU1 to MCU2 with **USART 1**

- connect RX/TX pins externally as
    - MCU1_TX to MCU2_RXD
    - MCU1_RX - MCU2_TX

4. Send a message from PC_1 by typing keys on Teraterm. It should send that message from MCU_1 to MCU_2.
5. The received message by MCU_2 should be displayed on PC_2.
6. Turn other MCU's LED(LD2) On/OFF by sending text:

- "**L**" for Turn OFF
- "**H**" for Turn ON

Configuration

Type	Port - Pin	Configuration
System Clock		PLL 84MHz
USART2 : USB cable (ST-Link)		No Parity, 8-bit Data, 1-bit Stop bit, 38400 baud-rate
USART1 : MCU1 - MCU2	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit, 38400 baud-rate
Digital Out: LD2	PA5	

Code

LAB_USART_LED.c

The `setup()` function is utilized for initial settings. In this function, tasks such as the initialization of RCC and SysTick, initialization and baud rate setting of UART2 and USART1, and setting of pin 5 of GPIOA as output are carried out. The `USART2_IRQHandler()` function serves as the RX interrupt handler for USART2. Data is obtained from USART2 and stored in PC_Data, and then it is transmitted back to USART2 and USART1. Additionally, the data that has been read is outputted to the PC. The `USART1_IRQHandler()` function acts as the RX interrupt handler for USART1. Data is received from USART1 and stored in BT_Data, and it is outputted to the PC. If the data that has been read is 'L', pin 5 of GPIOA is set to LOW, if it is 'H', it is set to HIGH.

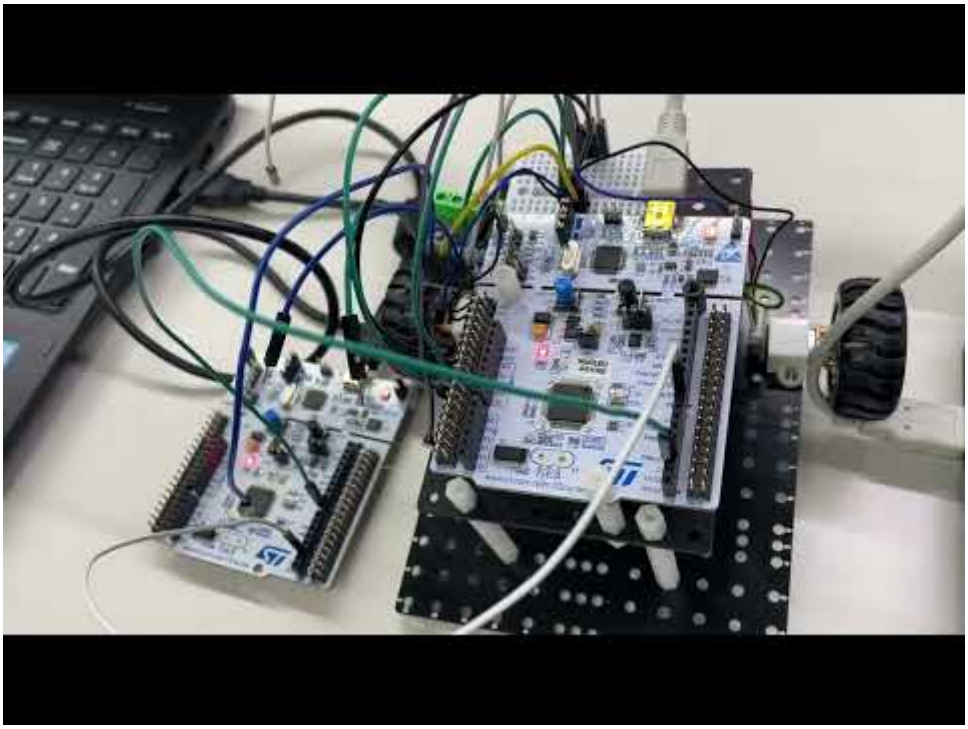
```
#include "stm32f4xx.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecUART.h"
#include "ecSysTick.h"


static volatile uint8_t PC_Data = 0;
static volatile uint8_t BT_Data = 0;


void setup(void){
  RCC_PLL_init();
    SysTick_init();

  // USB serial init
  UART2_init();
  UART2_baud(BAUD_38400);


    USART_setting(USART1,GPIOA,10,GPIOA,9,BAUD_38400);
    GPIO_init(GPIOA,5,OUTPUT);

}


void main(){
```

```
    setup();

        while(1){

    }
}


void USART2_IRQHandler(){                // USART2 RX Interrupt : Recommended
    if(is_USART2_RXNE()){
        PC_Data = USART2_read();         // RX from UART2 (PC)
        USART2_write(&PC_Data,1);        // TX to USART2  (PC)

        USART1_write(&PC_Data,1);        // TX to USART1  (BT)
        printf("MCU_1 sent : %c \r\n",PC_Data); // TX to USART2(PC)
    }
}

void USART1_IRQHandler(){                // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
            BT_Data = USART1_read();         // RX from UART1 (BT)

        printf("MCU_1 received : %c \r\n",BT_Data); // TX to USART2(PC)
        if(BT_Data == 'L'){
                GPIO_write(GPIOA,5,LOW);
        }
        else if(BT_Data == 'H'){
                GPIO_write(GPIOA,5,HIGH);
        }
    }
}
```

Result

As can be observed in the video, the LED on the other MCU is seen to turn on and off when 'H' or 'L' is input on one computer.

Demo video↓



Problem 3: Control DC Motor via Bluetooth

Hardware Setup

Connect the Bluetooth module with the DC motor driver by referring to the picture below.

Bluetooth

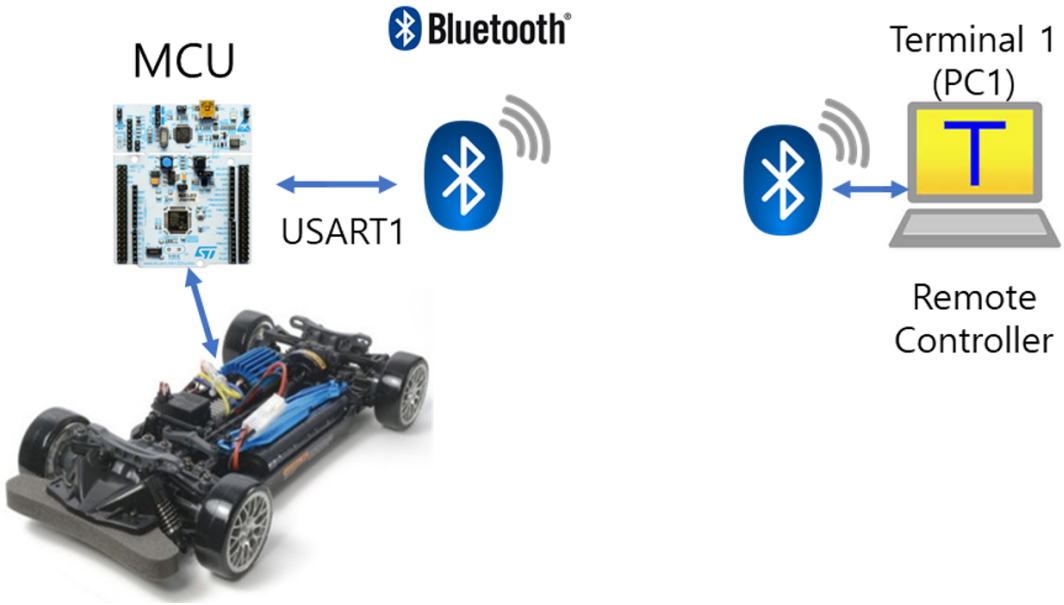
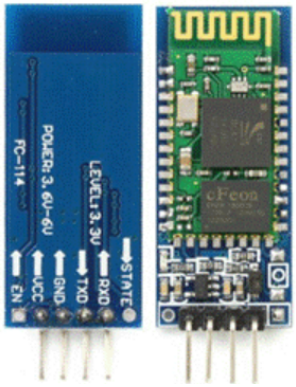


Figure2. Overview of MCU-DC Motor communication with Bluetooth Module



Bluetooth Module (HC-06)	STM32F411RE
RxD	PA_9(UART1_TX)
TxD	PA_10(UART1_RX)
GND	GND
VCC	5V

Figure3. Example of connecting to USART1

DC Motor Driver

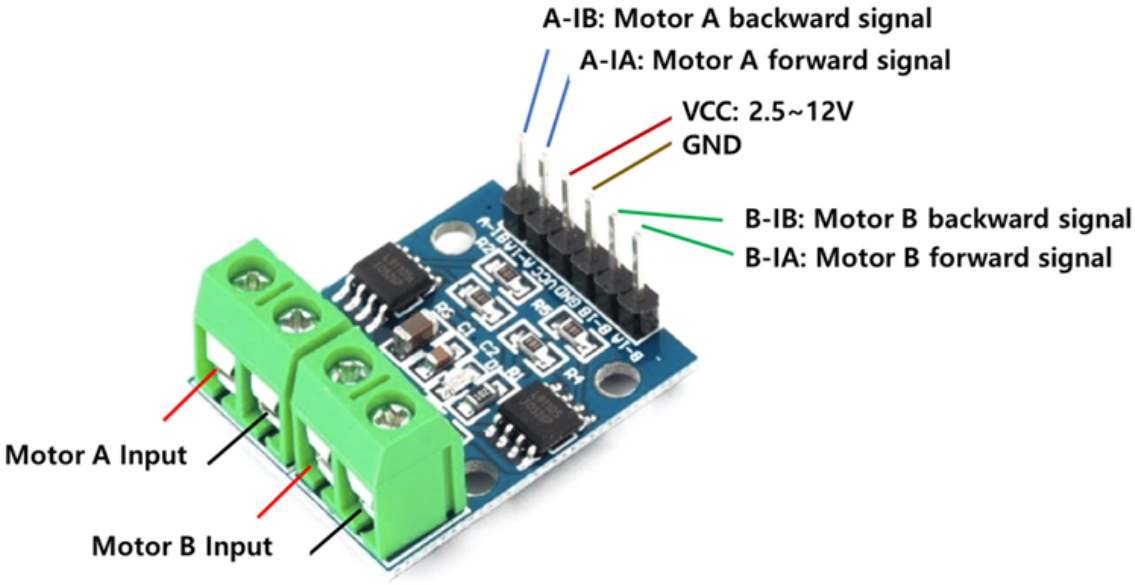


Figure4. DC motor driver Setup

Procedure

1. Connect the MCU to PC via Bluetooth. Use USART 1

- connect RX/TX pins as
  - MCU TXD - BLUE RXD
  - MCU RXD - BLUE TXD

2. Check the Bluetooth connection by turning MCU's LED(LD2) On/OFF by sending text of "**L0**" or "**L1**" from PC.
3. Run 2 DC motors(Left-wheel, Right-wheel) to steer.

- Turn Left: MotorA / MotorB = (50 / 80%) duty
- Turn Right: MotorA / MotorB = (80 / 50%) duty
- Go straight: MotorA / MotorB = (80 / 80 %) duty
- STOP: MotorA / MotorB = (0 / 0 %) duty

Left, right, straight, and stop were designated as 'A', 'D', 'W', and 'S', respectively.

Configuration

Type	Port - Pin	Configuration
System Clock		PLL 84MHz
USART1 : MCU - Bluetooth	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit, 9600 baud-rate
Digital Out: LD2	PA5	
PWM (Motor A)	TIM2-Ch1	PWM period (2kHz~10kHz)
PWM (Motor B)	TIM2-Ch2	

Code

LAB_USART_Bluetooth.c

The `void setup()` function initializes UART1 with a serial baud rate of 9600 and configures the LED pin and motor direction pins. It also initializes PWM. In the `void USART1_IRQHandler()` function, the code is structured to perform different actions based on the keys ('W', 'A', 'D', 'S') input from the keyboard by changing the PWM duty cycle of the left and right motors. Additionally, it includes actions to turn off the LED when 'L0' is received and turn on the LED when 'L1' is received.

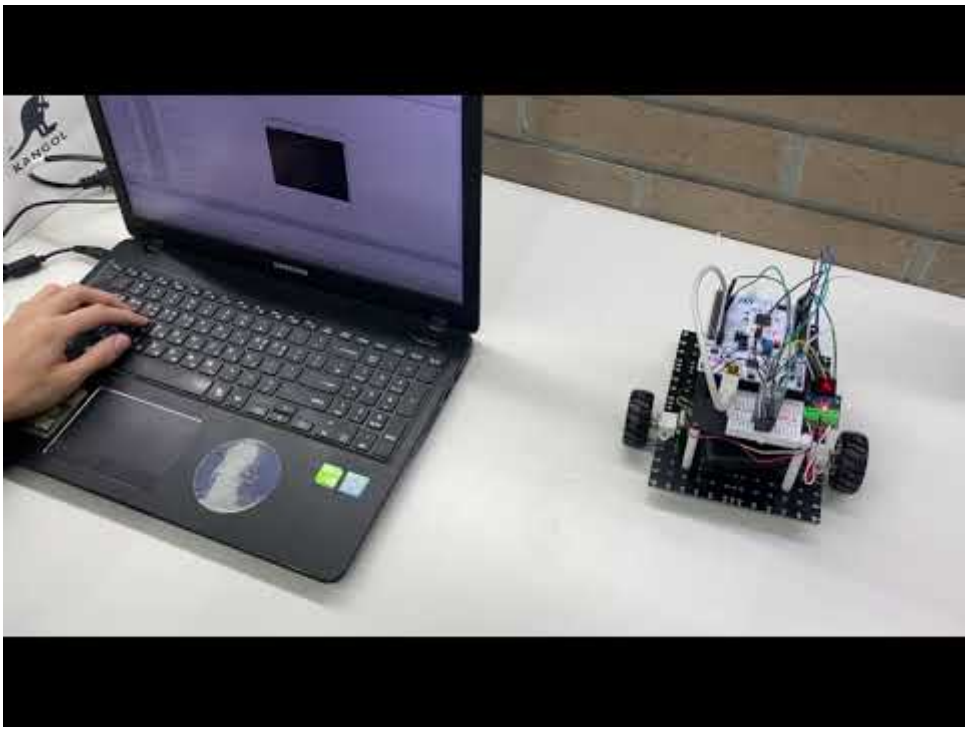
```
#include "stm32f4xx.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecUART.h"
#include "ecSysTick.h"
#include "ecPinNames.h"
#include "ecPWM.h"

static volatile uint8_t PC_Data = 0;
static volatile uint8_t BT_Data = 0;

float i=0;
float j=0;

void setup(void){
  RCC_PLL_init();

  // BT serial init
  UART1_init();
  UART1_baud(BAUD_9600);


    USART_setting(USART1, GPIOA, 9, GPIOA, 10, BAUD_9600);

    GPIO_init(GPIOA,5,OUTPUT);
    GPIO_init(GPIOC,2,OUTPUT);
    GPIO_init(GPIOC,3,OUTPUT);
```

```
    PWM_init(PA_0);
    PWM_period_us(PA_0, 200);
    PWM_init(PA_1);
    PWM_period_us(PA_1, 200);
}


void main(){
    setup();
  while(1){
        GPIO_write(GPIOC,2,LOW);
        GPIO_write(GPIOC,3,LOW);
        PWM_duty(PA_0, i);
        PWM_duty(PA_1, j);
        }
}

void USART1_IRQHandler(){               // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
    BT_Data = USART1_read();        // RX from UART1 (BT)
        USART1_write(&BT_Data,1);
        if(BT_Data=='W'){
            i=0.8;
            j=0.8;
      }
        else if(BT_Data=='A'){
            i=0.8;
            j=0.5;
        }
        else if(BT_Data=='D'){
            i=0.5;
            j=0.8;
        }
        else if(BT_Data=='S'){
            i=0;
            j=0;
        }
        else if (BT_Data == 'L') {
            BT_Data = USART_read(USART1);
            if (BT_Data == '0') {
                GPIO_write(GPIOA, 5, 0);
                USART_write(USART1, (uint8_t*)"L0", 5);
            }
            else if (BT_Data == '1') {
                GPIO_write(GPIOA, 5, 1);
                USART_write(USART1, (uint8_t*)"L1", 5);
            }
        }

    }
    USART_write(USART1, "\r\n", 1);
}
```

Result

As you can see from the image, you can check the LED behavior according to L0 or L1, and the left, right, straight, and stop are set to A, D, W, and S respectively, so it is confirmed that the operation is implemented.

Demo video↓



Reference

- STM32 Cortex®-M4 MCUs and MPUs programming manual Download Link
- STM32 Cortex®-M4 MCUs and MPUs reference manual Download Link