LAB: USART - LED, Bluetooth

Date: 2023-11-16

Author/Partner: Han TaeGeon / Jang HoJin

Demo Video:

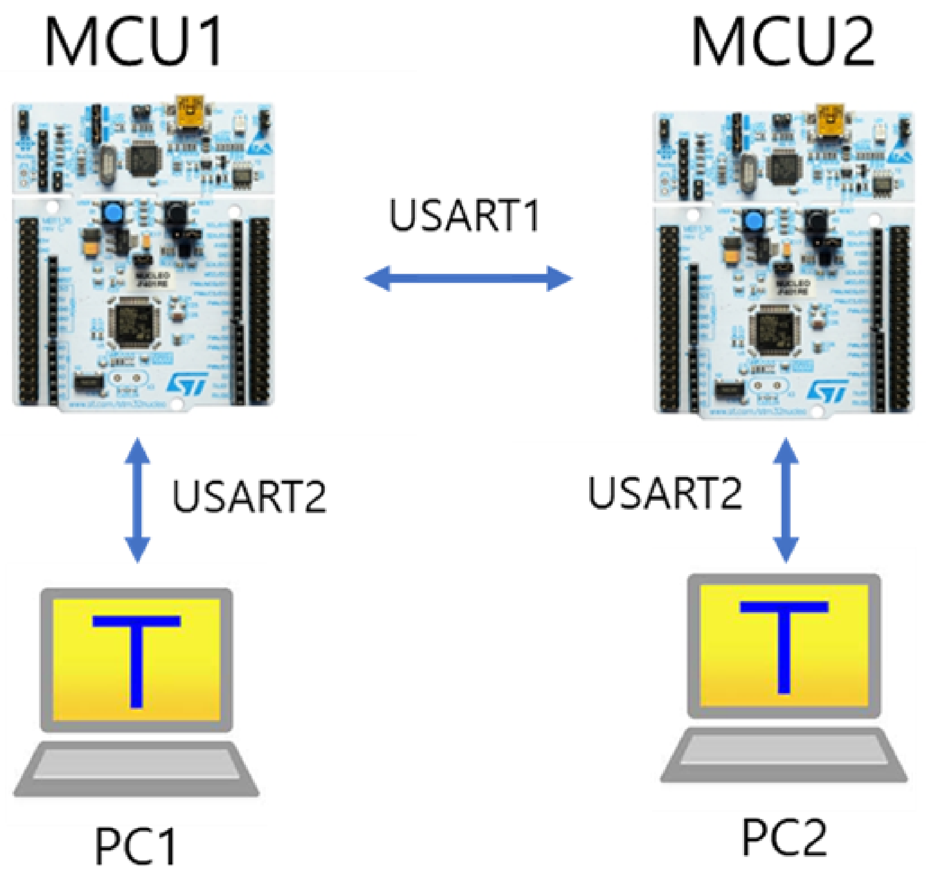
Problem2: https://youtu.be/pQhTp_7-oOQ

Problem3: https://youtu.be/JINbZKxEJf0

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



You must submit

- LAB Report (*.md & *.pdf)
- Zip source files(main*.c, ecRCC.h, ecGPIO.h, ecSysTick.c etc...).

    - Only the source files. Do not submit project files

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

Using HAL library

Download ecUART_student.c ecUART_student.h

Change the file names as

- ecUART.c, ecUART.h

Fill in empty spaces in the code.

You must update your header files located in the directory `EC \lib\`.

ecUSART.h

```
#ifndef __EC_USART_H
#define __EC_USART_H

#include <stdio.h>
//#include "stm32f411xe.h"
//#include "ecGPIO.h"
//#include "ecRCC.h"
#include "ecSTM32F411.h"
#include "string.h"

#define POL 0
#define INT 1

// You can modify this
#define BAUD_9600   9600
#define BAUD_19200  19200
#define BAUD_38400      38400
#define BAUD_57600  57600
#define BAUD_115200     115200
#define BAUD_921600     921600


// ********************** USART 2 (USB) ***************************
// PA_2 = USART2_TX
```

```
// PA_3 = USART2_RX
// Alternate function(AF7), High Speed, Push pull, Pull up
// APB1
// **********************************************************

// ********************** USART 1 ***************************
// PA_9 = USART1_TX (default)   // PB_6  (option)
// PA_10 = USART1_RX (default)  // PB_3 (option)
// APB2
// **********************************************************

// ********************** USART 6 ***************************
// PA_11 = USART6_TX (default)  // PC_6  (option)
// PA_12 = USART6_RX (default)  // PC_7 (option)
// APB2
// **********************************************************

// Configuration UART 1, 2 using default pins
void UART1_init(void);
void UART2_init(void);
void UART1_baud(uint32_t baud);
void UART2_baud(uint32_t baud);

// USART write & read
void USART1_write(uint8_t* buffer, uint32_t nBytes);
void USART2_write(uint8_t* buffer, uint32_t nBytes);
uint8_t USART1_read(void);
uint8_t USART2_read(void);

// RX Inturrupt Flag USART1,2
uint32_t is_USART1_RXNE(void);
uint32_t is_USART2_RXNE(void);

// private functions
void USART_write(USART_TypeDef* USARTx, uint8_t* buffer, uint32_t nBytes);
void USART_init(USART_TypeDef* USARTx, uint32_t baud);
void UART_baud(USART_TypeDef* USARTx, uint32_t baud);
uint32_t is_USART_RXNE(USART_TypeDef * USARTx);
uint8_t USART_read(USART_TypeDef * USARTx);
void USART_setting(USART_TypeDef* USARTx, GPIO_TypeDef* GPIO_TX, int pinTX,
GPIO_TypeDef* GPIO_RX, int pinRX, uint32_t baud);
void USART_delay(uint32_t us);

#endif
```

ecUSART.c

```
#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;
```

```c
};

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 ================*/
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, 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, 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 &= ~USART_CR1_PCE;          // No parrity bit
    USARTx->CR1 &= ~USART_CR1_M;        // 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 &= ~USART_CR2_STOP;         // 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);
}
```

Example Code

Example 1

```
#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;
uint8_t PC_string[]="Loop:\r\n";

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

    // USART2: USB serial init
    UART2_init();
    UART2_baud(BAUD_9600);

    // USART1: BT serial init
    UART1_init();
    UART1_baud(BAUD_9600);
}

int main(void){
    setup();
    printf("MCU Initialized\r\n");
    while(1){
        // USART Receive: Use Interrupt only
        // USART Transmit:  Interrupt or Polling
        USART2_write(PC_string, 7);
        delay_ms(2000);
    }
}

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)    Echo of
keyboard typing
    }
}


void USART1_IRQHandler(){                // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
        BT_Data = USART1_read();         // RX from UART1 (BT)
        printf("RX: %c \r\n",BT_Data); // TX to USART2(PC)
    }
}
```

Example 2

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

#define MAX_BUF     10
#define END_CHAR    13

static volatile uint8_t buffer[MAX_BUF]={0, };
static volatile uint8_t PC_string[MAX_BUF]={0, };
static volatile uint8_t PC_data = 0;

static volatile int idx = 0;
static volatile int bReceive =0;

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

    // USART2: USB serial init
    UART2_init();
    UART2_baud(BAUD_9600);

    // USART1: BT serial init
    UART1_init();
    UART1_baud(BAUD_9600);
}

int main(void){
    setup();
    printf("MCU Initialized\r\n");

    while(1){
        if (bReceive == 1){
            printf("PC_string: %s\r\n", PC_string);
            bReceive = 0;
        }
    }
}

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)    Echo of
keyboard typing

        // Creates a String from serial character receive
        if(PC_data != END_CHAR && (idx < MAX_BUF)){
            buffer[idx] = PC_data;
            idx++;
        }
        else if (PC_data== END_CHAR) {
            bReceive = 1;
            // reset PC_string;
            memset(PC_string, 0, sizeof(char) * MAX_BUF);
            // copy to PC_string;
```

```
            memcpy(PC_string, buffer, sizeof(char) * idx);
            // reset buffer
            memset(buffer, 0, sizeof(char) * MAX_BUF);
            idx = 0;
        }
        else{               //  if(idx >= MAX_BUF)
            idx = 0;
            // reset PC_string;
            memset(PC_string, 0, sizeof(char) * MAX_BUF);
            // reset buffer
            memset(buffer, 0, sizeof(char) * MAX_BUF);  // reset buffer
            printf("ERROR : Too long string\r\n");
        }
    }
}
```

General USART Setup

```
// General Setting
void setup(){
  RCC_PLL_init();

  // BT serial : specific RX/TX pins
  USART_setting(USART1, GPIOA,9,GPIOA,10, BAUD_9600);   // PA9 - RXD , PA10 -
TXD
}
```

Problem 2: Communicate MCU1-MCU2 using RS-232

Procedure

1. Create a new project under the directory `\repos\EC\LAB\LAB_USART_LED`

- The project name is “**LAB_USART_LED”.**
- Create a new source files named as “**LAB_USART_LED.c”**

2. Include your updated library in `\repos\EC\lib\` to your project.

- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecUART.h, ecUART.c**
- and other necessary header files

3. Connect each MCUs to each PC with **USART 2** via USB cable (ST-Link)

- MCU1-PC1, MCU2-PC2

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

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

> Connect the same GND for MCU1-MCU2

5. Send a message from PC_1 by typing keys on Teraterm. It should send that message from MCU_1 to MCU_2.
6. The received message by MCU_2 should be displayed on PC_2.
7. 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

The codes below are set according to the conditions given in the question.

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

#define MAX_BUF     10
#define END_CHAR    13

static volatile uint8_t buffer[MAX_BUF]={0, };
static volatile uint8_t PC_string[MAX_BUF]={0, };
static volatile uint8_t PC_data = 0;
static volatile uint8_t BT_data = 0;


static volatile int idx = 0;
static volatile int bReceive =0;

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

    // USART2: USB serial init
    UART2_init();
    UART2_baud(BAUD_9600);

    // USART1: BT serial init
```

```
    UART1_init();
    UART1_baud(BAUD_9600);
}
```

Outputs a string in the main statement.

```
int main(void){
    setup();
    printf("MCU Initialized\r\n");

    while(1){
        if (bReceive == 1){
            printf("PC_string: %s\r\n", PC_string);
            bReceive = 0;
        }
    }
}
```

It covers the values of USART2.

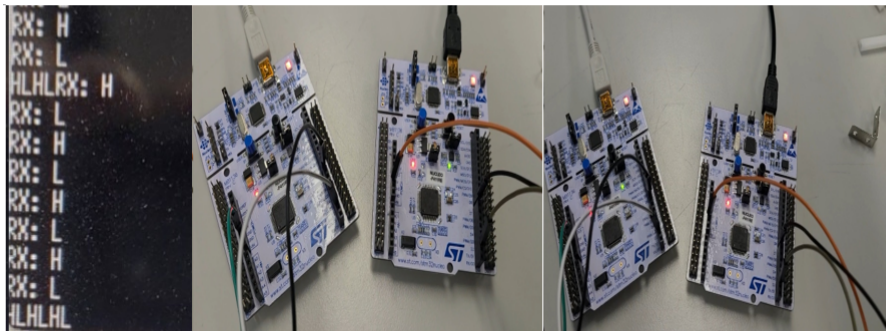
```
void USART2_IRQHandler(){                    // USART2 RX Interrupt : Recommended
    if(is_USART2_RXNE()){
        PC_data = USART2_read();             // RX from UART2 (PC)

        USART2_write(&PC_data,1);            // TX to USART2  (PC)    Echo of
keyboard typing
        USART1_write(&PC_data,1);            // TX to USART1  (BT)

    }
}
```

Read the values in USART1 and adjust the LED ON/OFF according to the conditional statement.

```
void USART1_IRQHandler(){                    // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
        GPIO_init(GPIOA, LED_PIN, OUTPUT);
        if(BT_data == 'L')  GPIO_write(GPIOA, LED_PIN, 1);
        else if(BT_data == 'H') GPIO_write(GPIOA, LED_PIN, 0);
        BT_data = USART1_read();

        printf("RX: %c \r\n",BT_data); // TX to USART2(PC)
    }
}
```

Result

- Experiment images



- Results

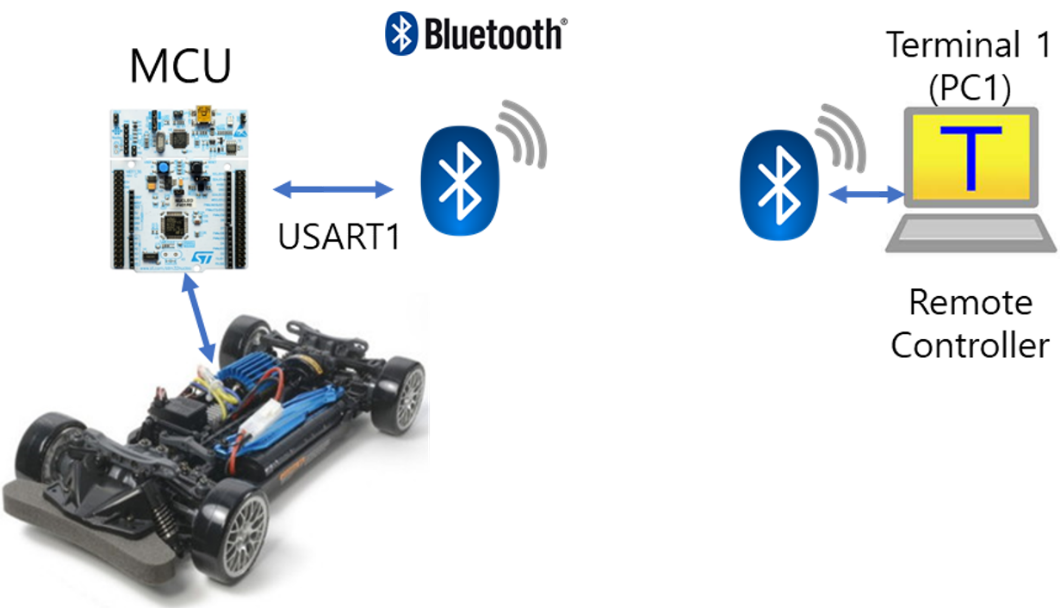
  Turned each other's LEDs on and off using the same code as the partner. If I write 'L' on Tera Term, the partner's LED will be off. When I write 'H' on Tera Term, the partner's LED turns on. At this time, my LED does not change at all, and only the partner's LED responds. Conversely, when a partner writes an 'H' or 'L' on the Tera Term, my LED turns on/off. If you type a letter other than 'L' or 'H', nothing changes.

Demo Video

- Link : https://youtu.be/pQhTp_7-oOQ

Problem 3: Control DC Motor via Bluetooth

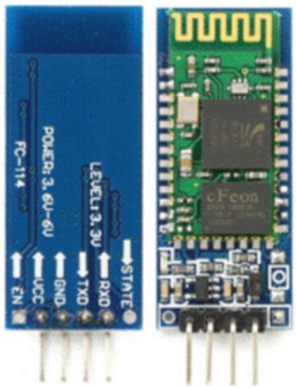
Bluetooth



image

Search for the bluetooth module specification sheet (HC-06) and study the pin configurations. The default PIN number is 1234.

Example of connecting to USART1



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

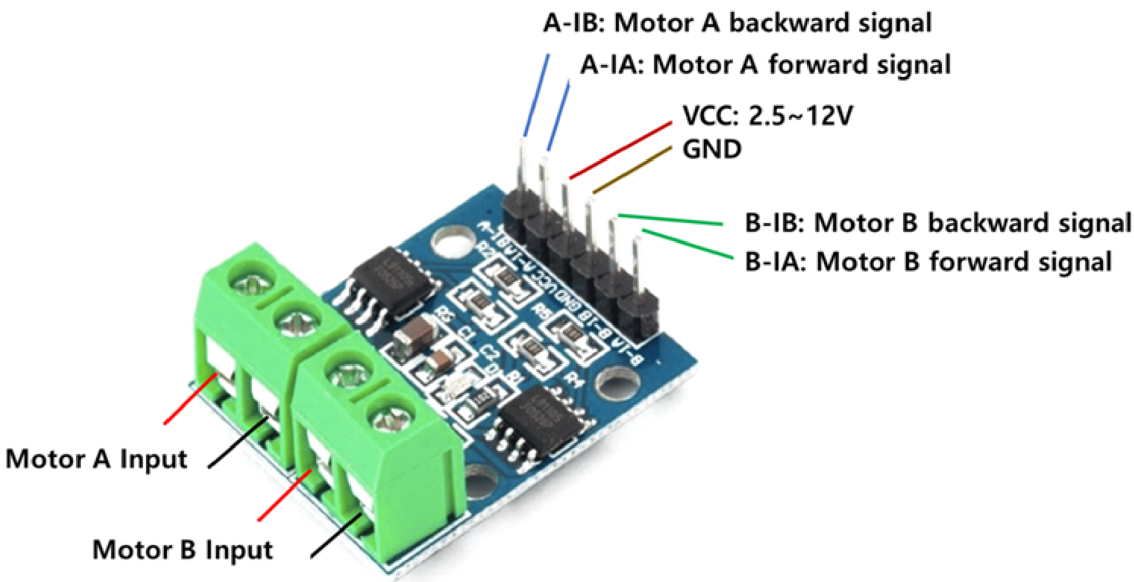
image

DC Motor Driver

Connect DC motor driver(L9110s) module pins to MCU as shown below.

DO NOT use MCU's VCC to motor driver. You should use external voltage source.

- A- IA: PWM pin (0~100% duty) for Motor A
- A- IB: Direction Pin (Digital Out H or L) for Motor B



image

Procedure

1. Create a new project under the directory `\repos\EC\LAB\LAB_USART_Bluetooth

- The project name is "**LAB_USART_Bluetooth".**
- Create a new source files named as "**LAB_USART_Bluetooth.c"**

> You MUST write your name on the source file inside the comment section.

2. Include your updated library in `\repos\EC\lib\` to your project.

- **ecGPIO.h, ecGPIO.c**
- **ecRCC.h, ecRCC.c**
- **ecUART.h, ecUART.c**
- **ecTIM.h, ecTIM.c**

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

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

4. Check the Bluetooth connection by turning MCU's LED(LD2) On/OFF by sending text of "**L0**" or "**L1**" from PC.
5. 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

You may use the key inputs as your preference for Left, Right, Straight.

- Ex) 'L', 'R', 'U' 'S'

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

Your code goes here: [ADD Code LINK such as github]

The codes below set up what is needed to solve the problem.

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

#define MAX_BUF     10
#define END_CHAR    13
#define MOTOR_A 2
#define MOTOR_B 3

static volatile uint8_t buffer[MAX_BUF]={0, };
static volatile uint8_t PC_string[MAX_BUF]={0, };
static volatile uint8_t PC_data = 0;
```

```
static volatile uint8_t BT_data = 0;


static float dutyA = 0;     // PWM of Motor A
static float dutyB = 0;   // PWM of Motor B

static volatile int bReceive =0; // flag

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

    // PWM init
    PWM_init(PA_0);
    PWM_init(PA_1);

    PWM_period_us(PA_0, 200);   // 1 usec PWM period
    PWM_period_us(PA_1, 200);   // 1 usec PWM period

    // GPIO
    GPIO_init(GPIOA, LED_PIN, OUTPUT);  // LED PIN
    GPIO_init(GPIOC, MOTOR_A, OUTPUT);  // motorA direction
    GPIO_init(GPIOC, MOTOR_B, OUTPUT);  // motorB direction
    mcu_init(GPIOA, LED_PIN);
    mcu_init(GPIOC, MOTOR_A);
    mcu_init(GPIOC, MOTOR_B);

    // USART2: USB serial init
    UART2_init();
    UART2_baud(BAUD_9600);

    // USART1: BT serial init
    UART1_init();
    UART1_baud(BAUD_9600);
    USART_setting(USART1,GPIOA, 9, GPIOA, 10, 9600);
}
```

It controls the movement of the car and LED within the main function according to the pwm duty.

```
int main(void){
    setup();
    printf("MCU Initialized\r\n");

    while(1){
        if (bReceive == 1){         // flag
            bReceive = 0;

                // LED ON/OFF
                if(BT_data == 'A')  GPIO_write(GPIOA, LED_PIN, 0);
                else if(BT_data == 'H') GPIO_write(GPIOA, LED_PIN, 1);

                    // Control a car direction and speed
                    switch(BT_data){
                        case 'L' :  // Go left
                            GPIO_write(GPIOC, MOTOR_A, 1);
                            GPIO_write(GPIOC, MOTOR_B, 1);
```

```
                            dutyA = 0.5;
                            dutyB = 0.2;
                        break;
                        case 'R' :  // Go right
                            GPIO_write(GPIOC, MOTOR_A, 1);
                            GPIO_write(GPIOC, MOTOR_B, 1);
                            dutyA = 0.2;
                            dutyB = 0.5;
                        break;
                        case 'U' :  // Go straight
                            GPIO_write(GPIOC, MOTOR_A, 1);
                            GPIO_write(GPIOC, MOTOR_B, 1);
                            dutyA = 0.2;
                            dutyB = 0.2;
                        break;
                        case 'S' :  // Stop
                            GPIO_write(GPIOC, MOTOR_A, 1);
                            GPIO_write(GPIOC, MOTOR_B, 1);
                            dutyA = 1;
                            dutyB = 1;
                        break;

                        case 'B' :  // Go back
                            GPIO_write(GPIOC, MOTOR_A, 0);
                            GPIO_write(GPIOC, MOTOR_B, 0);
                            dutyA = 0.8;
                            dutyB = 0.8;
                        break;
                        case 'N' :  // Go back left
                            GPIO_write(GPIOC, MOTOR_A, 0);
                            GPIO_write(GPIOC, MOTOR_B, 0);
                            dutyA = 0.8;
                            dutyB = 0.5;
                        break;
                        case 'V' :  // Go back right
                            GPIO_write(GPIOC, MOTOR_A, 0);
                            GPIO_write(GPIOC, MOTOR_B, 0);
                            dutyA = 0.5;
                            dutyB = 0.8;
                        break;
                        default  : break;
            }
        }
        PWM_duty(PA_0, dutyA);
        PWM_duty(PA_1, dutyB);
    }
}
```

Check USART2 to see if the value is correct.

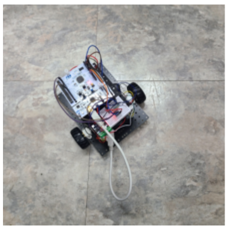
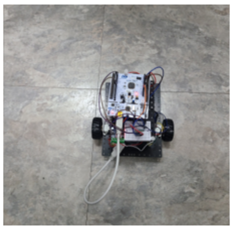
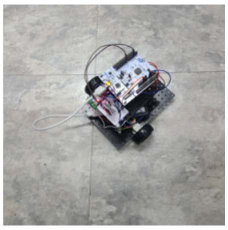
```
void USART2_IRQHandler(){                    // USART2 RX Interrupt : Recommended
    if(is_USART2_RXNE()){
        PC_data = USART2_read();             // RX from UART2 (PC)

        USART2_write(&PC_data,1);            // TX to USART2  (PC)    Echo of
keyboard typing
        //USART1_write(&PC_data,1);          // TX to USART1  (BT)
    }
}
```

Bluetooth data is transmitted.

```
void USART1_IRQHandler(){                    // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){

        BT_data = USART1_read();                         // RX from USART1
        USART_write(USART1, &BT_data, 1);    // TX to USART1

        if(BT_data == END_CHAR)      // Press enter key
            USART_write(USART1, "\r\n", 2);

        bReceive = 1;    // flag = 1
        }
}
```

Result

Experiment images

Results

If you press 'H', the LED turns on. If you press 'A', the LED turns off. The problem says 'L0' and 'L1', but 'L' moves to the left, so the character has been changed so that it does not overlap. If you press 'U', the car moves forward, press 'S', and stop the car. Press 'R' to move to the right.

It's not shown in the problem, but it also added a backward function. If you press 'B', the car moves backward, if you press 'V', it moves back to the left, and if you press 'N', it moves back to the right. Every time press the keyboard, the value appears in TerraTerm.

Demo Video

Link : https://youtu.be/JINbZKxEJf0

Reference

```
Young-Keun Kim (2023). https://ykkim.gitbook.io/ec/
```

Troubleshooting

Although defined in the problem, USART1 is used between MCU boards and USART2 is used between PC and MCU boards. Also, between Bluetooth and MCU boards is USART1. These were confusing, so I wrote the wrong code in the USART function. In particular, TX and RX are important in communication. Data must be transmitted from the TX and received from the RX.There was an error because this principle was not applied.

Appendix

- ecPWM.c

```
#include "stm32f4xx.h"
#include "ecPWM.h"
#include "math.h"
#include "ecPinNames.h"

/* PWM Configuration using PinName_t Structure */

/* PWM initialization */
// Default: 84MHz PLL, 1MHz CK_CNT, 50% duty ratio, 1msec period
void PWM_init(PinName_t pinName){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Initialize GPIO port and pin as AF
    GPIO_init(port, pin, AF);  // AF=2
    GPIO_otype(port, pin, EC_PUSH_PULL);
    GPIO_pupd(port, pin, EC_NONE);
    GPIO_ospeed(port, pin, EC_MEDIUM);



// 2. Configure GPIO AFR by Pin num.
```

```
    //  AFR[0] for pin: 0~7,     AFR[1] for pin 8~15
    //  AFR=1 for TIM1,TIM2 AFR=2 for TIM3 etc
         if(pin >= 0 && pin <8){
             if(TIMx == TIM1 || TIMx == TIM2)       port->AFR[0] |= 1 <<
(4*pin);
             else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)      port-
>AFR[0] |= 2 << (4*pin);
             else if(TIMx == TIM9 || TIMx == TIM10 || TIMx == TIM11)    port-
>AFR[0] |= 3 << (4*pin);
         }
         else if(pin >= 8 && pin <=15){
             if(TIMx == TIM1 || TIMx == TIM2)       port->AFR[1] |= 1 << (pin-
8);
             else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)      port-
>AFR[1] |= 2 << (pin-8);
             else if(TIMx == TIM9 || TIMx == TIM10 || TIMx == TIM11)    port-
>AFR[1] |= 3 << (pin-8);
         }


// 3. Initialize Timer
    TIM_init(TIMx, 1);  // with default msec=1msec value.
    TIMx->CR1 &= ~TIM_CR1_CEN;

// 3-2. Direction of Counter
    TIMx->CR1 &= ~TIM_CR1_DIR;                          // Counting direction: 0
= up-counting, 1 = down-counting


// 4. Configure Timer Output mode as PWM
    uint32_t ccVal = TIMx->ARR/2;  // default value  CC=ARR/2
    if(chN == 1){
        TIMx->CCMR1 &= ~TIM_CCMR1_OC1M;                     // Clear ouput
compare mode bits for channel 1
        TIMx->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2; // OC1M = 110 for
PWM Mode 1 output on ch1. #define TIM_CCMR1_OC1M_1          (0x2UL <<
TIM_CCMR1_OC1M_Pos)
        TIMx->CCMR1 |= TIM_CCMR1_OC1PE;                     // Output 1 preload
enable (make CCR1 value changable)
        TIMx->CCR1  = ccVal;
// Output Compare Register for channel 1 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC1P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC1E;
// Enable output for ch1
    }
    else if(chN == 2){
        TIMx->CCMR1 &= ~TIM_CCMR1_OC2M;                     // Clear ouput
compare mode bits for channel 2
        TIMx->CCMR1 |= TIM_CCMR1_OC2M_1 | TIM_CCMR1_OC2M_2; // OC1M = 110 for
PWM Mode 1 output on ch2
        TIMx->CCMR1 |= TIM_CCMR1_OC2PE;                     // Output 1 preload
enable (make CCR2 value changable)
        TIMx->CCR2  = ccVal;
// Output Compare Register for channel 2 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC2P;                       // select output
polarity: active high
```

```
        TIMx->CCER  |= TIM_CCER_CC2E;
// Enable output for ch2
    }
    else if(chN == 3){
        TIMx->CCMR2 &= ~TIM_CCMR2_OC3M;                     // Clear ouput
compare mode bits for channel 3
        TIMx->CCMR2 |= TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2; // OC1M = 110 for
PWM Mode 1 output on ch3
        TIMx->CCMR2 |= TIM_CCMR1_OC1PE;                     // Output 1 preload
enable (make CCR3 value changable)
        TIMx->CCR3  = ccVal;
// Output Compare Register for channel 3 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC3P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC3E;
// Enable output for ch3
    }
    else if(chN == 4){
        TIMx->CCMR2 &= ~TIM_CCMR2_OC4M;                     // Clear ouput
compare mode bits for channel 4
        TIMx->CCMR2 |= TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2; // OC1M = 110 for
PWM Mode 1 output on ch4
        TIMx->CCMR2 |= TIM_CCMR1_OC2PE;                     // Output 1 preload
enable (make CCR3 value changable)
        TIMx->CCR4  = ccVal;
// Output Compare Register for channel 4 (default duty ratio = 50%)
        TIMx->CCER &= ~TIM_CCER_CC4P;                       // select output
polarity: active high
        TIMx->CCER  |= TIM_CCER_CC4E;
// Enable output for ch4
    }


// 5. Enable Timer Counter
    // For TIM1 ONLY
    if(TIMx == TIM1) TIMx->BDTR |= TIM_BDTR_MOE;                    // Main
output enable (MOE): 0 = Disable, 1 = Enable
    // Enable timers
    TIMx->CR1  |= TIM_CR1_CEN;
// Enable counter

}

/* PWM PERIOD SETUP */
// allowable range for msec:  1~2,000
void PWM_period_ms(PinName_t pinName,  uint32_t msec){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Set Counter Period in msec
    TIM_period_ms(TIMx, msec);
```

```
}


// allowable range for msec:  1~2,000
void PWM_period(PinName_t pinName,  uint32_t msec){
    PWM_period_ms(pinName,  msec);
}


// allowable range for usec:  1~1,000
void PWM_period_us(PinName_t pinName,  uint32_t usec){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Set Counter Period in usec
    TIM_period_us(TIMx, usec);  //YOUR CODE GOES HERE

}

/* DUTY RATIO SETUP */
// High Pulse width in msec
void PWM_pulsewidth(PinName_t pinName, uint32_t pulse_width_ms){
// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);


// 1. Declaration System Frequency and Prescaler
    uint32_t fsys = 0;
    uint32_t psc = TIMx->PSC;


// 2. Check System CLK: PLL or HSI
    if((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)        fsys = 84000;
 // for msec 84MHz/1000 [msec]
    else if((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI) fsys = 16000;


// 3. Configure prescaler PSC
    float fclk = fsys/(psc+1);                              //
fclk=fsys/(psc+1);
    uint32_t value = pulse_width_ms *fclk - 1;      // pulse_width_ms *fclk - 1;

    switch(chN){
        case 1: TIMx->CCR1 = value; break;
        case 2: TIMx->CCR2 = value; break;
```

```
        case 3: TIMx->CCR3 = value; break;
        case 4: TIMx->CCR4 = value; break;
        default: break;
    }
}

// High Pulse width in msec
void PWM_pulsewidth_ms(PinName_t pinName, uint32_t pulse_width_ms){
    PWM_pulsewidth(pinName, pulse_width_ms);
}

// High Pulse width in usec
void PWM_pulsewidth_us(PinName_t pinName, uint32_t pulse_width_us){
// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);

// 1. Declaration system frequency and prescaler
    uint32_t fsys = 0;
    uint32_t psc = TIMx->PSC;


// 2. Check System CLK: PLL or HSI
    if((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)        fsys = 84;  //
for msec 84MHz/1000000 [usec]
    else if((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI) fsys = 16;


// 3. Configure prescaler PSC
    float fclk = fclk=fsys/(psc+1);                             //
fclk=fsys/(psc+1);
    uint32_t value = pulse_width_us *fclk - 1;      // pulse_width_us *fclk - 1;

    switch(chN){
        case 1: TIMx->CCR1 = value; break;
        case 2: TIMx->CCR2 = value; break;
        case 3: TIMx->CCR3 = value; break;
        case 4: TIMx->CCR4 = value; break;
        default: break;
    }
}

// Dutry Ratio from 0 to 1
void PWM_duty(PinName_t pinName, float duty){

// 0. Match TIMx from  Port and Pin
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int chN;
    PWM_pinmap(pinName, &TIMx, &chN);
```

```c
// 1. Configure prescaler PSC
    float value = TIMx->ARR + 1;                                    //
(ARR+1)*dutyRatio + 1
        value = value*duty - 1;

    if(chN == 1)      { TIMx->CCR1 = value; }          //set channel
        else if(chN == 2)      { TIMx->CCR2 = value; }
        else if(chN == 3)      { TIMx->CCR3 = value; }
        else if(chN == 4)      { TIMx->CCR4 = value; }

}

// DO NOT MODIFY HERE
void PWM_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN)
{
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);


  if(port == GPIOA) {
      switch(pin){
         case 0 : *TIMx = TIM2; *chN    = 1; break;
         case 1 : *TIMx = TIM2; *chN = 2; break;
         case 5 : *TIMx = TIM2; *chN = 1; break;
         case 6 : *TIMx = TIM3; *chN = 1; break;
         //case 7: TIMx = TIM1; *chN = 1N; break;
         case 8 : *TIMx = TIM1; *chN = 1; break;
         case 9 : *TIMx = TIM1; *chN = 2; break;
         case 10: *TIMx = TIM1; *chN = 3; break;
         case 15: *TIMx = TIM2; *chN = 1; break;
         default: break;
      }
   }
   else if(port == GPIOB) {
      switch(pin){
         //case 0: TIMx = TIM1; *chN = 2N; break;
         //case 1: TIMx = TIM1; *chN = 3N; break;
         case 3 : *TIMx = TIM2; *chN = 2; break;
         case 4 : *TIMx = TIM3; *chN = 1; break;
         case 5 : *TIMx = TIM3; *chN = 2; break;
         case 6 : *TIMx = TIM4; *chN = 1; break;
         case 7 : *TIMx = TIM4; *chN = 2; break;
         case 8 : *TIMx = TIM4; *chN = 3; break;
         case 9 : *TIMx = TIM4; *chN = 4; break;
         case 10: *TIMx = TIM2; *chN = 3; break;
         default: break;
      }
   }
   else if(port == GPIOC) {
      switch(pin){
         case 6 : *TIMx = TIM3; *chN = 1; break;
         case 7 : *TIMx = TIM3; *chN = 2; break;
         case 8 : *TIMx = TIM3; *chN = 3; break;
         case 9 : *TIMx = TIM3; *chN = 4; break;
         default: break;
      }
   }
```

```
    // TIM5 needs to be added, if used.
}
```

ecPWM.h

```
#ifndef __EC_PWM_H
#define __EC_PWM_H


#include "stm32f411xe.h"
#include "ecGPIO.h"
#include "ecTIM.h"

#include "ecPinNames.h"


#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */


/* PWM Configuration using PinName_t Structure */

/* PWM initialization */
// Default: 84MHz PLL, 1MHz CK_CNT, 50% duty ratio, 1msec period
void PWM_init(PinName_t pinName);
void PWM_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN);


/* PWM PERIOD SETUP */
// allowable range for msec:  1~2,000
void PWM_period(PinName_t pinName,  uint32_t msec);
void PWM_period_ms(PinName_t pinName,  uint32_t msec);  // same as PWM_period()
// allowable range for usec:  1~1,000
void PWM_period_us(PinName_t pinName, uint32_t usec);


/* DUTY RATIO SETUP */
// High Pulse width in msec
void PWM_pulsewidth(PinName_t pinName, uint32_t pulse_width_ms);
void PWM_pulsewidth_ms(PinName_t pinName, uint32_t pulse_width_ms);  // same as
void PWM_pulsewidth
void PWM_pulsewidth_us(PinName_t pinName, uint32_t pulse_width_us);
// Duty ratio 0~1.0
void PWM_duty(PinName_t pinName, float duty);

// dutycycle = CCR/(ARR+1);
// PWM_Period = (1 + ARR)*CLK_Period
// CountPeriod = (1+ARR)/f_CK_CNT
// f_cl_cnt = f_CL_PSC/(PSC+1)

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

- ecGPIO.c

```
#include "ecGPIO.h"

void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode){
    // mode  : Input(0), Output(1), AlterFunc(2), Analog(3)
    if (Port == GPIOA)
        RCC_GPIOA_enable();
    if (Port == GPIOC)
        RCC_GPIOC_enable();
    if (Port == GPIOB)
        RCC_GPIOB_enable();
    if (Port == GPIOD)
        RCC_GPIOD_enable();

    // Make it for GPIOB, GPIOD..GPIOH

    // You can also make a more general function of
    // void RCC_GPIO_enable(GPIO_TypeDef *Port);

    GPIO_mode(Port, pin, mode);

}


// GPIO Mode          : Input(00), Output(01), AlterFunc(10), Analog(11)
void GPIO_mode(GPIO_TypeDef *Port, int pin, unsigned int mode){
   Port->MODER &= ~(3UL<<(2*pin));
   Port->MODER |= mode<<(2*pin);
}


// GPIO Speed          : Low speed (00), Medium speed (01), Fast speed (10), High speed (11)
void GPIO_ospeed(GPIO_TypeDef *Port, int pin, unsigned int speed){
    Port->OSPEEDR &= ~(3UL<<(2*pin));
    Port->OSPEEDR |= speed<<(2*pin);
}

// GPIO Output Type: Output push-pull (0, reset), Output open drain (1)
void GPIO_otype(GPIO_TypeDef *Port, int pin, unsigned int type){
    Port->OTYPER &= ~(1UL)<< pin;
    Port->OTYPER |= (type<< pin);

}

// GPIO Push-Pull    : No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)
void GPIO_pupd(GPIO_TypeDef *Port, int pin, unsigned int pupd){
    Port->PUPDR &= ~(3UL<<2*pin);
    Port->PUPDR |= pupd<<(2*pin);   //write
}

int GPIO_read(GPIO_TypeDef *Port, int pin){
    unsigned int BVal = (Port->IDR)>>pin & (1);

    return BVal;
```

```
}

void GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output){
    Port->ODR &= ~(1<<pin);
    Port->ODR |= (Output << pin);
}


void sevensegment_init(void){
        // Calls RCC_GPIO_enable()
        GPIO_init(GPIOA, LED_PA5, OUTPUT);
        GPIO_init(GPIOA, LED_PA6, OUTPUT);
        GPIO_init(GPIOA, LED_PA7, OUTPUT);
        GPIO_init(GPIOB, LED_PB6, OUTPUT);
        GPIO_init(GPIOC, LED_PC7, OUTPUT);
        GPIO_init(GPIOA, LED_PA9, OUTPUT);
        GPIO_init(GPIOA, LED_PA8, OUTPUT);
        GPIO_init(GPIOB, LED_PB10, OUTPUT);
}


void sevensegment_decoder(uint8_t  num){
      //pins are sorted from upper left corner of the display to the lower right
corner
    //the display has a common cathode
    //the display actally has 8 led's, the last one is a dot
        unsigned int led[8]=
{LED_PB9,LED_PA6,LED_PA7,LED_PB6,LED_PC7,LED_PA9,LED_PA8,LED_PB10};

        //each led that has to light up gets a 1, every other led gets a 0
        //its in order of the DigitalOut Pins above
        unsigned int number[11][8]={
                                                {0,0,0,0,0,0,1,1},    //zero
                                                {1,0,0,1,1,1,1,1},    //one
                                                {0,0,1,0,0,1,0,1},    //two
                                                {0,0,0,0,1,1,0,1},    //three
                                                {1,0,0,1,1,0,0,1},    //four
                                                {0,1,0,0,1,0,0,1},    //five
                                                {0,1,0,0,0,0,0,1},    //six
                                                {0,0,0,1,1,0,1,1},    //seven
                                                {0,0,0,0,0,0,0,1},    //eight
                                                {0,0,0,0,1,0,0,1},    //nine
                                              {0,0,1,1,0,0,0,1},        //P
        };
            //all led's off
        for(int i = 0; i<8;i++){led[i] = 0;}

            //display shows the number in this case 6
        for (int i=0; i<8; i++){led[i] = number[num][i];}         //the digit
after "number" is displayed

        GPIO_write(GPIOB, LED_PB9,  led[0]);
        GPIO_write(GPIOA, LED_PA6,  led[1]);
        GPIO_write(GPIOA, LED_PA7,  led[2]);
        GPIO_write(GPIOB, LED_PB6,  led[3]);
        GPIO_write(GPIOC, LED_PC7,  led[4]);
        GPIO_write(GPIOA, LED_PA9,  led[5]);
        GPIO_write(GPIOA, LED_PA8,  led[6]);
```

```
        GPIO_write(GPIOB, LED_PB10, led[7]);
}


void sevensegment_display_init(void){
            // Calls RCC_GPIO_enable()
        GPIO_init(GPIOA, LED_PA7, OUTPUT);
        GPIO_init(GPIOB, LED_PB6, OUTPUT);
        GPIO_init(GPIOC, LED_PC7, OUTPUT);
        GPIO_init(GPIOA, LED_PA9, OUTPUT);

}

void sevensegment_display(uint8_t  num){
    //pins are sorted from upper left corner of the display to the lower right
corner
    //the display has a common cathode
    //the display actally has 8 led's, the last one is a dot
        unsigned int led[4]={LED_PA7,LED_PB6,LED_PC7,LED_PA9}; // A,B,C,D

        //each led that has to light up gets a 1, every other led gets a 0
        //its in order of the DigitalOut Pins above
        unsigned int number[10][4]={
                                                {0,0,0,0},      //zero
                                                {0,0,0,1},    //one
                                                {0,0,1,0},    //two
                                                {0,0,1,1},    //three
                                                {0,1,0,0},    //four
                                                {0,1,0,1},    //five
                                                {0,1,1,0},    //six
                                                {0,1,1,1},    //seven
                                                {1,0,0,0},    //eight
                                                {1,0,0,1},    //nine
        };
            //all led's off
        for(int i = 0; i<4;i++){led[i] = 0;}

            //display shows the number in this case 6
        for (int i=0; i<4; i++){led[i] = number[num][i];}         //the digit
after "number" is displayed

        GPIO_write(GPIOA, LED_PA7,  led[3]);
        GPIO_write(GPIOB, LED_PB6,  led[2]);
        GPIO_write(GPIOC, LED_PC7,  led[1]);
        GPIO_write(GPIOA, LED_PA9,  led[0]);

}

void LED_UP(uint8_t  num) {
    unsigned int led[4] = {LED_A0, LED_A1, LED_B0, LED_C1};
    unsigned int number[16][4]={
                                                {0,0,0,0},    //zero
                                                {0,0,0,1},    //one
                                                {0,0,1,0},    //two
                                                {0,0,1,1},    //three
                                                {0,1,0,0},    //four
                                                {0,1,0,1},    //five
                                                {0,1,1,0},    //six
```

```
                                            {0,1,1,1},    //seven
                                            {1,0,0,0},    //eight
                                            {1,0,0,1},    //nine
                                            {1,0,1,0},    //ten
                                            {1,0,1,1},    //eleven
                                            {1,1,0,0},    //twelve
                                            {1,1,0,1},    //thirteen
                                            {1,1,1,0},    //fourteen
                                            {1,1,1,1},      //fifteen
                                        };

                                        for(int i = 0; i<4;i++){led[i] = 0;}
                                        for (int i=0; i<4; i++){led[i] =
number[num][i];}

        GPIO_write(GPIOA, LED_A0, led[0]);
        GPIO_write(GPIOA, LED_A1, led[1]);
        GPIO_write(GPIOB, LED_B0, led[2]);
        GPIO_write(GPIOC, LED_C1, led[3]);
}


void LED_toggle(){
    int led_state = GPIO_read(GPIOA, LED_PIN);
    int time = 0;
    while(time < 1000)
        time++;

    GPIO_write(GPIOA, LED_PIN, !led_state);
}

void mcu_init(GPIO_TypeDef *Port, int pin){
    GPIO_pupd(Port, pin, EC_NONE);
    GPIO_otype(Port, pin, EC_PUSH_PULL);
    GPIO_ospeed(Port, pin, EC_FAST);
}
```

- ecGPIO.h

```
#ifndef __ECGPIO_H
#define __ECGPIO_H


#include "ecRCC.h"


#define INPUT  0x00
#define OUTPUT 0x01
#define AF     0x02
#define ANALOG 0x03

#define HIGH 1
#define LOW  0

#define EC_NONE 0
#define  EC_PU 1
#define EC_PD 2
```

```c
#define EC_PUSH_PULL 0
#define EC_OPEN_DRAIN 1

#define EC_LOW 0
#define EC_MEDIUM 1
#define EC_FAST 2
#define EC_HIGH 3

#define LED_A0 0
#define LED_A1 1
#define LED_B0 0
#define LED_C1 1
#define LED_PB9     9
#define LED_PA5     5
#define LED_PA6     6
#define LED_PA7     7
#define LED_PB6     6
#define LED_PC7     7
#define LED_PA9     9
#define LED_PA8     8
#define LED_PB10    10
#define BUTTON_PIN 13
#define LED_PIN 5

#define Direction_PIN 2
#define PWM_PIN PA_0

#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */

void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode);
void GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output);
int  GPIO_read(GPIO_TypeDef *Port, int pin);
void GPIO_mode(GPIO_TypeDef* Port, int pin, unsigned  int mode);
void GPIO_ospeed(GPIO_TypeDef* Port, int pin, unsigned int speed);
void GPIO_otype(GPIO_TypeDef* Port, int pin, unsigned int type);
void GPIO_pupd(GPIO_TypeDef* Port, int pin, unsigned int pupd);

void sevensegment_init(void);
void sevensegment_decoder(uint8_t  num);

void sevensegment_display_init(void);
void sevensegment_display(uint8_t  num);
void LED_UP(uint8_t  num);

void LED_toggle();

void mcu_init(GPIO_TypeDef* Port, int pin);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

- ecTIM.c

```
#include "ecTIM.h"

/* Timer Configuration */

void TIM_init(TIM_TypeDef* TIMx, uint32_t msec) {

    // 1. Enable Timer CLOCK
    if (TIMx == TIM1) RCC->APB2ENR |= RCC_APB2ENR_TIM1EN;
    else if (TIMx == TIM2) RCC->APB1ENR |= RCC_APB1ENR_TIM2EN;
    else if (TIMx == TIM3) RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    // repeat for TIM4, TIM5, TIM9, TIM11
  else if (TIMx == TIM4) RCC->APB1ENR |= RCC_APB1ENR_TIM4EN;
    else if (TIMx == TIM5) RCC->APB1ENR |= RCC_APB1ENR_TIM5EN;
    else if (TIMx == TIM9) RCC->APB2ENR |= RCC_APB2ENR_TIM9EN;
    else if (TIMx == TIM11) RCC->APB2ENR |= RCC_APB2ENR_TIM11EN;


// 2. Set CNT period
    TIM_period_ms(TIMx, msec);


    // 3. CNT Direction
    TIMx->CR1 &= ~(1 << 4);                  // Upcounter

    // 4. Enable Timer Counter
    TIMx->CR1 |= TIM_CR1_CEN;
}


//  Q. Which combination of PSC and ARR for msec unit?
//  Q. What are the possible range (in sec ?)
void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec) {
    // Period usec = 1 to 1000

    // 1us(1MHz, ARR=1) to 65msec (ARR=0xFFFF)
    uint16_t PSCval;
    uint32_t Sys_CLK;

    if ((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)
        Sys_CLK = 84000000;

    else if ((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI)
        Sys_CLK = 16000000;


    if (TIMx == TIM2 || TIMx == TIM5) {
        uint32_t ARRval;

        PSCval = Sys_CLK / 1000000;                         // 84 or 16
--> f_cnt = 1MHz
        ARRval = Sys_CLK / PSCval / 1000000 * usec;     // 1MHz*usec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
    else {
```

```
        uint16_t ARRval;

        PSCval = Sys_CLK / 1000000;                                    // 84 or
16 --> f_cnt = 1MHz
        ARRval = Sys_CLK / PSCval / 1000000 * usec;     // 1MHz*usec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
}


void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec) {
    // Period msec = 1 to 6000

    // 0.1ms(10kHz, ARR = 1) to 6.5sec (ARR = 0xFFFF)
    // uint16_t PSCval = 8400;
    // uint16_t ARRval = ______________;            // 84MHz/1000ms
    //
    // TIMx->PSC = PSCval - 1;
    // TIMx->ARR = ARRval;
    uint16_t PSCval;
    uint32_t Sys_CLK;

    if ((RCC->CFGR & RCC_CFGR_SW_PLL) == RCC_CFGR_SW_PLL)
        Sys_CLK = 84000000;

    else if ((RCC->CFGR & RCC_CFGR_SW_HSI) == RCC_CFGR_SW_HSI)
        Sys_CLK = 16000000;


    if (TIMx == TIM2 || TIMx == TIM5) {
        uint32_t ARRval;

        PSCval = Sys_CLK / 100000;                                 // 840 or
160   --> f_cnt = 100kHz
        ARRval = Sys_CLK / PSCval / 1000 * msec;        // 100kHz*msec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
    else {
        uint16_t ARRval;

        PSCval = Sys_CLK / 10000;                                  // 8400 or
1600 --> f_cnt = 10kHz
        ARRval = Sys_CLK / PSCval / 1000 * msec;        // 10kHz*msec
        TIMx->PSC = PSCval - 1;
        TIMx->ARR = ARRval - 1;
    }
}

// msec = 1 to 6000
void TIM_period(TIM_TypeDef* TIMx, uint32_t msec){
    TIM_period_ms(TIMx, msec);
}

// Update Event Interrupt
void TIM_UI_init(TIM_TypeDef* TIMx, uint32_t msec) {
    // 1. Initialize Timer
```

```
    TIM_init(TIMx, msec);

    // 2. Enable Update Interrupt
    TIM_UI_enable(TIMx);

    // 3. NVIC Setting
    uint32_t IRQn_reg = 0;
    if (TIMx == TIM1)         IRQn_reg = TIM1_UP_TIM10_IRQn;
    else if (TIMx == TIM2)    IRQn_reg = TIM2_IRQn;
    // repeat for TIM3, TIM4, TIM5, TIM9, TIM10, TIM11
  else if (TIMx == TIM3)    IRQn_reg = TIM3_IRQn;
    else if (TIMx == TIM4)    IRQn_reg = TIM4_IRQn;
    else if (TIMx == TIM5)    IRQn_reg = TIM5_IRQn;
    else if (TIMx == TIM9)    IRQn_reg = TIM1_BRK_TIM9_IRQn;
    else if (TIMx == TIM10)   IRQn_reg = TIM1_UP_TIM10_IRQn;
    else if (TIMx == TIM11)   IRQn_reg = TIM1_TRG_COM_TIM11_IRQn;

    NVIC_EnableIRQ(IRQn_reg);
    NVIC_SetPriority(IRQn_reg, 2);
}



void TIM_UI_enable(TIM_TypeDef* TIMx) {
    TIMx->DIER |= TIM_DIER_UIE;         // Enable Timer Update Interrupt
}

void TIM_UI_disable(TIM_TypeDef* TIMx) {
    TIMx->DIER &= ~TIM_DIER_UIE;                // Disable Timer Update
Interrupt
}

uint32_t is_UIF(TIM_TypeDef* TIMx) {
    return TIMx->SR & TIM_SR_UIF;
}

void clear_UIF(TIM_TypeDef* TIMx) {
    TIMx->SR &= ~TIM_SR_UIF;
}


/* -------- Timer Input Capture -------- */

void ICAP_init(PinName_t pinName){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int TIn;

    ICAP_pinmap(pinName, &TIMx, &TIn);
    int ICn = TIn;                                                  // (default)
TIx=ICx

// GPIO configuration -----------------------------------------------------------
----------
// 1. Initialize GPIO port and pin as AF
```

```
    GPIO_init(port, pin, AF);                              // AF=2
    GPIO_ospeed(port, pin, EC_HIGH);                           // speed VHIGH=3

// 2. Configure GPIO AFR by Pin num.
    if(TIMx == TIM1 || TIMx == TIM2)
port->AFR[pin >> 3] |= 0x01 << (4*(pin % 8)); // TIM1~2
    else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)  port->AFR[pin >> 3]
|= 0x02 << (4*(pin % 8)); // TIM3~5
    else if(TIMx == TIM9 || TIMx == TIM10|| TIMx == TIM11) port->AFR[pin >> 3]
|= 0x03 << (4*(pin % 8)); // TIM9~11


// TIMER configuration -----------------------------------------------------------
-----------
// 1. Initialize Timer Interrpt
    TIM_UI_init(TIMx, 1);                              // TIMx Interrupt initialize

// 2. Modify ARR Maxium for 1MHz
    TIMx->PSC = 84-1;                                              // Timer counter
clock: 1MHz(1us)  for PLL
    TIMx->ARR = 0xFFFF;                                            // Set auto
reload register to maximum (count up to 65535)

// 3. Disable Counter during configuration
    TIMx->CR1 &= ~TIM_CR1_CEN;                                 // Disable Counter
during configuration



// Input Capture configuration ---------------------------------------------------
-------------------
// 1. Select Timer channel(TIx) for Input Capture channel(ICx)
    // Default Setting
    TIMx->CCMR1 &=  ~TIM_CCMR1_CC1S;
    TIMx->CCMR1 &=  ~TIM_CCMR1_CC2S;
    TIMx->CCMR2 &=  ~TIM_CCMR2_CC3S;
    TIMx->CCMR2 &=  ~TIM_CCMR2_CC4S;
    TIMx->CCMR1 |=  TIM_CCMR1_CC1S_0;      //01<<0   CC1S    TI1=IC1
    TIMx->CCMR1 |=  TIM_CCMR1_CC2S_0;                      //01<<8   CC2s
TI2=IC2
    TIMx->CCMR2 |=  TIM_CCMR2_CC3S_0;                      //01<<0   CC3s
TI3=IC3
    TIMx->CCMR2 |=  TIM_CCMR2_CC4S_0;                          //01<<8   CC4s
TI4=IC4


// 2. Filter Duration (use default)

// 3. IC Prescaler (use default)

// 4. Activation Edge: CCyNP/CCyP
    TIMx->CCER &= ~(0b1010 << 4*(ICn-1));                                      //
CCy(Rising) for ICn,  ~(1<<1)


// 5.   Enable CCy Capture, Capture/Compare interrupt
    TIMx->CCER |= 1 << (4*(ICn-1));                    // CCn(ICn) Capture Enable
```

```
// 6.   Enable Interrupt of CC(CCyIE), Update (UIE)
    TIMx->DIER |= ICn;                      // Capture/Compare Interrupt Enable
for ICn
    TIMx->DIER |= TIM_DIER_UIE;                         // Update Interrupt
enable

// 7.   Enable Counter
    TIMx->CR1    |= TIM_CR1_CEN;                            // Counter enable
}


// Configure Selecting TIx-ICy and Edge Type
void ICAP_setup(PinName_t pinName, int ICn, int edge_type){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int CHn;
    ICAP_pinmap(pinName, &TIMx, &CHn);

// 1. Disable  CC. Disable CCInterrupt for ICn.
    TIMx->CCER &= ~(1 << (4*(ICn - 1)));
// Capture Enable
    TIMx->DIER &= ~(1 << ICn);
// CCn Interrupt enabled

    // setting on timers by channer. ex) ch1 -> 1or2~
// 2. Configure  IC number(user selected) with given IC pin(TIMx_CHn)
    switch(ICn){
            case 1:
                    TIMx->CCMR1 &= ~TIM_CCMR1_CC1S;
//reset   CC1S
                    if (ICn==CHn) TIMx->CCMR1 |=    TIM_CCMR1_CC1S_0;
//01<<0   CC1S    Tx_Ch1=IC1
                    else TIMx->CCMR1 |=     TIM_CCMR1_CC1S_1;
//10<<0   CC1S    Tx_Ch2=IC1
                    break;
            case 2:
                    TIMx->CCMR1 &= ~TIM_CCMR1_CC2S;
//reset   CC2S
                    if (ICn==CHn) TIMx->CCMR1 |=    TIM_CCMR1_CC2S_0;
//01<<8   CC2S    Tx_Ch2=IC2
                    else TIMx->CCMR1 |=     TIM_CCMR1_CC2S_1;
//10<<8   CC2S    Tx_Ch1=IC2
                    break;
            case 3:
                    TIMx->CCMR2 &= ~TIM_CCMR2_CC3S;
//reset   CC3S
                    if (ICn==CHn) TIMx->CCMR2 |= TIM_CCMR2_CC3S_0;      //01<<0
  CC3S    Tx_Ch3=IC3
                    else TIMx->CCMR2 |= TIM_CCMR2_CC3S_1;
//10<<0   CC3S    Tx_Ch4=IC3
                    break;
            case 4:
                    TIMx->CCMR2 &= ~TIM_CCMR2_CC4S;
//reset   CC4S
```

```
        if (ICn==CHn) TIMx->CCMR2 |= TIM_CCMR2_CC4S_0;      //01<<8
  CC4S    Tx_Ch4=IC4
        else TIMx->CCMR2 |= TIM_CCMR2_CC4S_1;
//10<<8   CC4S    Tx_Ch3=IC4
        break;
      default: break;
    }


// 3. Configure Activation Edge direction
    TIMx->CCER  &= ~(0b1010 << 4*(ICn - 1));                // Clear
CCnNP/CCnP bits
    switch(edge_type){
        case IC_RISE: TIMx->CCER &= ~(0b1010 << 4*(ICn - 1)); break;
//rising:  00
        case IC_FALL: TIMx->CCER |= 0b0010 << 4*(ICn - 1);   break; //falling:
01
        case IC_BOTH: TIMx->CCER |= 0b1010 << 4*(ICn - 1);   break; //both:
11
    }

// 4. Enable CC. Enable CC Interrupt.
    TIMx->CCER |= 1 << (4*(ICn - 1));                                       //
Capture Enable
    TIMx->DIER |= 1 << ICn;
// CCn Interrupt enabled
}

// Time span for one counter step
void ICAP_counter_us(PinName_t pinName, int usec){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int CHn;
    ICAP_pinmap(pinName, &TIMx, &CHn);

// 1. Configuration Timer Prescaler and ARR
    TIMx->PSC = 84*usec-1;                      // Timer counter clock: 1us *
usec
    TIMx->ARR = 0xFFFF;                               // Set auto reload
register to maximum (count up to 65535)
}

uint32_t is_CCIF(TIM_TypeDef *TIMx, uint32_t ccNum){
    return (TIMx->SR & (0x1UL << ccNum)) != 0;
}

void clear_CCIF(TIM_TypeDef *TIMx, uint32_t ccNum){
    TIMx->SR &= ~(1 << ccNum);
}

uint32_t ICAP_capture(TIM_TypeDef* TIMx, uint32_t ICn){
    uint32_t capture_Value;

    if (ICn == 1)
        capture_Value = TIMx->CCR1;
```

```c
    else if (ICn == 2)
        capture_Value = TIMx->CCR2;
    else if (ICn == 3)
        capture_Value = TIMx->CCR3;
    else
        capture_Value = TIMx->CCR4;

    return capture_Value;
}

//DO NOT MODIFY THIS
void ICAP_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN){
     GPIO_TypeDef *port;
     unsigned int pin;
     ecPinmap(pinName, &port, &pin);

   if(port == GPIOA) {
      switch(pin){
         case 0 : *TIMx = TIM2; *chN = 1; break;
         case 1 : *TIMx = TIM2; *chN = 2; break;
         case 5 : *TIMx = TIM2; *chN = 1; break;
         case 6 : *TIMx = TIM3; *chN = 1; break;
         //case 7: *TIMx = TIM1; *chN = 1N; break;
         case 8 : *TIMx = TIM1; *chN = 1; break;
         case 9 : *TIMx = TIM1; *chN = 2; break;
         case 10: *TIMx = TIM1; *chN = 3; break;
         case 15: *TIMx = TIM2; *chN = 1; break;
         default: break;
      }
   }
   else if(port == GPIOB) {
      switch(pin){
         //case 0: *TIMx = TIM1; *chN = 2N; break;
         //case 1: *TIMx = TIM1; *chN = 3N; break;
         case 3 : *TIMx = TIM2; *chN = 2; break;
         case 4 : *TIMx = TIM3; *chN = 1; break;
         case 5 : *TIMx = TIM3; *chN = 2; break;
         case 6 : *TIMx = TIM4; *chN = 1; break;
         case 7 : *TIMx = TIM4; *chN = 2; break;
         case 8 : *TIMx = TIM4; *chN = 3; break;
         case 9 : *TIMx = TIM4; *chN = 3; break;
         case 10: *TIMx = TIM2; *chN = 3; break;

         default: break;
      }
   }
   else if(port == GPIOC) {
      switch(pin){
         case 6 : *TIMx = TIM3; *chN = 1; break;
         case 7 : *TIMx = TIM3; *chN = 2; break;
         case 8 : *TIMx = TIM3; *chN = 3; break;
         case 9 : *TIMx = TIM3; *chN = 4; break;

         default: break;
      }
   }
}
```

- ecTIM.h

```
#ifndef __EC_TIM_H
#define __EC_TIM_H
#include "stm32f411xe.h"
#include "ecSTM32F411.h"

#ifdef __cplusplus
 extern "C" {
#endif /* __cplusplus */


// ICn selection according to CHn
#define FIRST 1
#define SECOND 2

// Edge Type
#define IC_RISE 0
#define IC_FALL 1
#define IC_BOTH 2

// IC Number
#define IC_1 1
#define IC_2 2
#define IC_3 3
#define IC_4 4


/* Timer Configuration */
void TIM_init(TIM_TypeDef *TIMx, uint32_t msec);
void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec);
void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec);
void TIM_period(TIM_TypeDef* TIMx, uint32_t msec);

void TIM_UI_init(TIM_TypeDef* TIMx, uint32_t msec);
void TIM_UI_enable(TIM_TypeDef* TIMx);
void TIM_UI_disable(TIM_TypeDef* TIMx);

uint32_t is_UIF(TIM_TypeDef *TIMx);
void clear_UIF(TIM_TypeDef *TIMx);

/* Input Capture*/
void ICAP_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN);
void ICAP_init(PinName_t pinName);
void ICAP_setup(PinName_t pinName, int ICn, int edge_type);
void ICAP_counter_us(PinName_t pinName, int usec);
uint32_t ICAP_capture(TIM_TypeDef* TIMx, uint32_t ICn);

uint32_t is_CCIF(TIM_TypeDef *TIMx, uint32_t CCnum);  // CCnum= 1~4
void clear_CCIF(TIM_TypeDef *TIMx, uint32_t CCnum);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

- ecSTM32F411.h

```
#include "ecEXTI.h"
#include "ecGPIO.h"
#include "ecPinNames.h"
#include "ecPWM.h"
#include "ecRCC.h"
#include "ecSysTick.h"
#include "ecTIM.h"
//#include "ecUART_simple.h"
#include "ecPWM.h"
#include "ecPinNames.h"
#include "ecStepper.h"
#include "ecUART.h"

#include "stm32f411xe.h"
#include "stm32f4xx.h"
#include "math.h"
```