# LAB: Input Capture - Ultrasonic

**Date:** 2023-11-06

**Author:** Han TaeGeon

**Demo Video:** [https://youtu.be/Y1fbwsmxXI8](https://youtu.be/Y1fbwsmxXI8)

# Introduction

In this lab, you are required to create a simple program that uses input capture mode to measure the distance using an ultrasonic distance sensor. The sensor also needs trigger pulses that can be generated by using the timer output.

You must submit

- LAB Report (*.pdf & *.md)
- 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:
  - HC-SR04
  - breadboard

### Software

- Keil uVision, CMSIS, EC_HAL library

# Problem 1: Create HAL library

## Create HAL library

Declare and Define the following functions in your library. You must update your header files located in the directory `EC \lib\`.

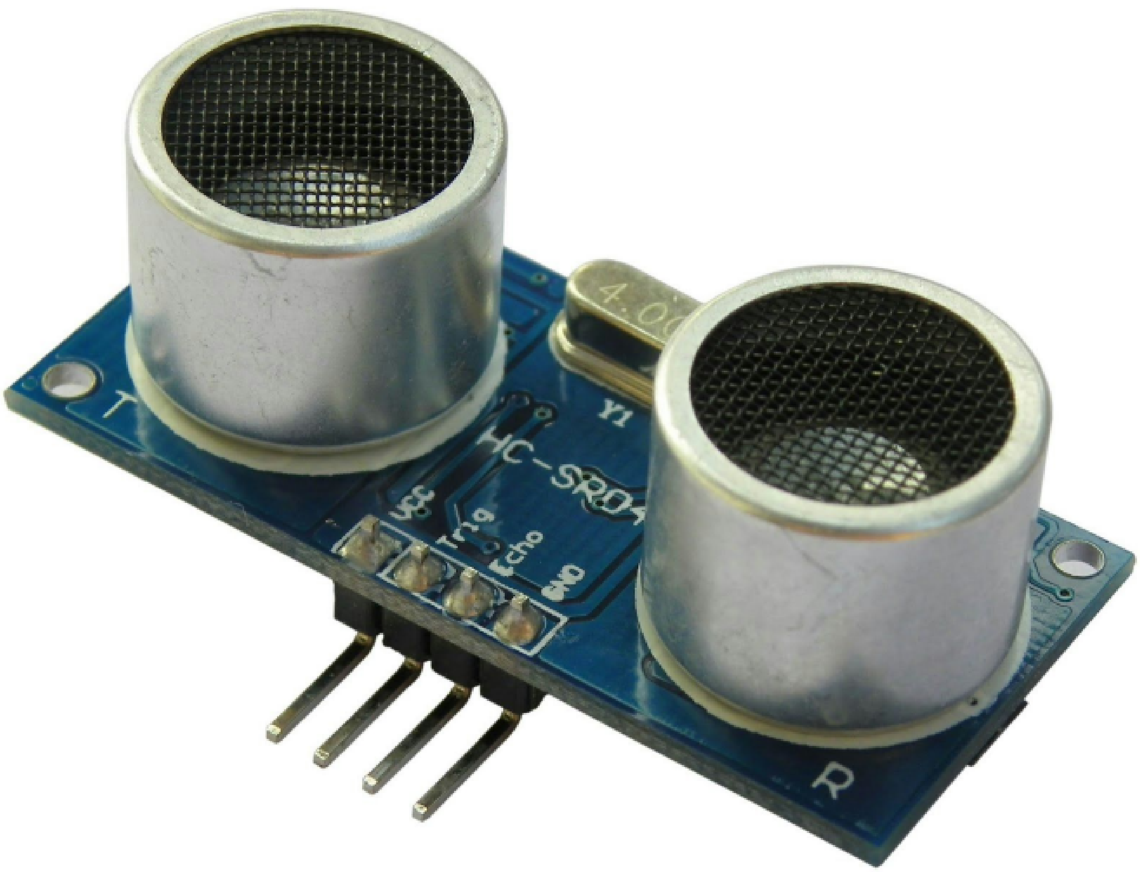
**ecTIM.h**

```
/* Input Capture*/
// ICn selection according to CHn
#define FIRST 1
#define SECOND 2
```

```c
// 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

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);
```

## Problem 2: Ultrasonic Distance Sensor (HC-SR04)

The HC-SR04 ultrasonic distance sensor. This economical sensor provides 2cm to 400cm of non-contact measurement functionality with a ranging accuracy that can reach up to 3mm. Each HC-SR04 module includes an ultrasonic transmitter, a receiver and a control circuit.




HC-SR04

**The HC-SR04 Ultrasonic Range Sensor Features:**

- Input Voltage: 5V
- Current Draw: 20mA (Max)

- Digital Output: 5V
- Digital Output: 0V (Low)
- Sensing Angle: 30° Cone
- Angle of Effect: 15° Cone
- Ultrasonic Frequency: 40kHz
- Range: 2cm - 400cm

## Procedure

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

- The project name is “**LAB_Timer_InputCaputre_Ultrasonic”.**
- Create a new source file named as “**LAB_Timer_InputCaputre_Ultrasonic.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**
- **ecTIM.h, ecTIM.c**
- **ecPWM.h, ecPWM.c**
- **ecSysTick.h, ecSysTick.c**
- **ecUART_simple.h, ecUART_simple.c**

3. Connect the HC-SR04 ultrasonic distance sensor to MCU pins(PA6 - trigger, PB6 - echo), VCC and GND

## Measurement of Distance

The program needs to

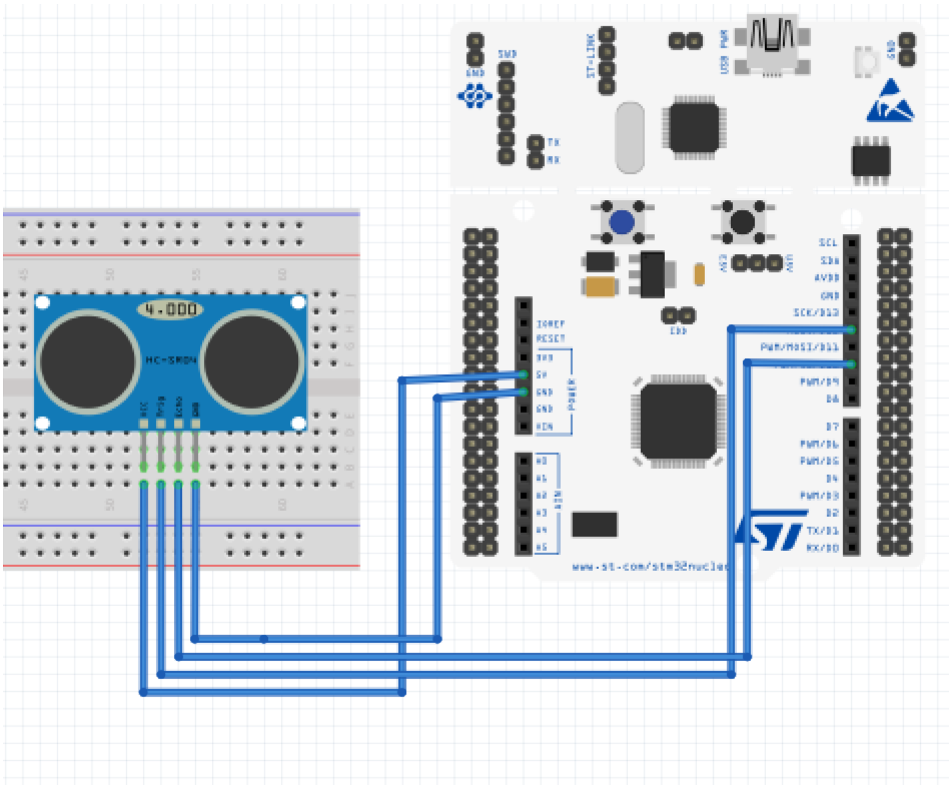
- Generate a trigger pulse as PWM to the sensor.
- Receive echo pulses from the ultrasonic sensor
- Measure the distance by calculating pulse-width of the echo pulse.
- Display measured distance in [cm] on serial monitor of Tera-Term for

  (a) 10mm (b) 50mm (c) 100mm

## Configuration

| System Clock | PWM | Input Capture |
|---|---|---|
| PLL (84MHz) | PA6 (TIM3_CH1) | PB6 (TIM4_CH1) |
| | AF, Push-Pull, No Pull-up Pull-down, Fast | AF, No Pull-up Pull-down |
| | PWM period: 50msec pulse width: 10usec | Counter Clock : 0.1MHz (10us) TI4 -> IC1 (rising edge) TI4 -> IC2 (falling edge) |

## Circuit Diagram



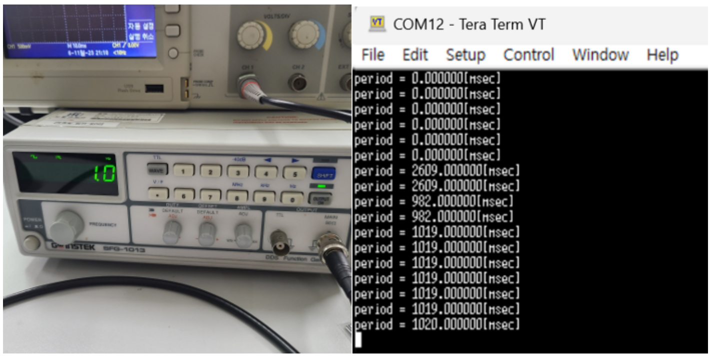
## Discussion

1. There can be an over-capture case, when a new capture interrupt occurs before reading the CCR value. When does it occur and how can you calculate the time span accurately between two captures?

   An over-capture case is usually capturing a new value before a previously captured value is read or processed. The timer captures from an external signal, but if the signal has a high frequency, the timer may not process data quickly. In this situation, over-capture occurs. In addition, if multiple events occur quickly in succession, the next event occurs before the timer processes all of one event, which also occurs in this case. In other words, it occurs because the processing speed is slower than the received event.

   When a quick successive event occurs, multiple capture interrupts are used to continue storing the event until over-caputre occurs. When over-capture occurs, the time span may be obtained by calculating a difference from the point at which the event first occurred.

2. In the tutorial, what is the accuracy when measuring the period of 1Hz square wave? Show your result.




$$accuracy = \frac{1019 - 1000}{1000} * 100 = 1.9\%$$

The error rate is about 2% when it is 1Hz square wave.

## Code

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

```
uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

#define TRIG PA_6
#define ECHO PB_6

void setup(void);
```

Calculates the distance in the Main function and outputs it as a Tera Term.

```
int main(void){

    setup();

    while(1){
        distance = (float) timeInterval * 340.0 / 2.0 / 10.0;    // [mm] -> [cm]
        printf("%f cm\r\n", distance);
        delay_ms(500);
    }
}
```

The below codes are the total time that is calculated using the Rising edge and Falling edge.

```
void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                     // Update interrupt
        uint32_t ovf_cnt = 0;                                  // overflow count
```

```
        clear_UIF(TIM4);                                // clear update
interrupt flag
    }

    if(is_CCIF(TIM4, 1)){                               // TIM4_Ch1 (IC1)
Capture Flag. Rising Edge Detect
        time1 = TIM4->CCR1;                                 // Capture TimeStart
        clear_CCIF(TIM4, 1);                  // clear capture/compare interrupt
flag
    }

    else if(is_CCIF(TIM4, 2)){                                  // TIM4_Ch2
(IC2) Capture Flag. Falling Edge Detect
        time2 = TIM4->CCR2;                                 // Capture TimeEnd
        timeInterval = (time2-time1+(TIM4->ARR+1)*ovf_cnt)*1e-2;    // (10us *
counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                         // overflow reset
        clear_CCIF(TIM4,2);                                // clear
capture/compare interrupt flag
    }
}
```

The codes below are set according to the conditions given in the question. Sets the values for PWM and Input Capture.

```
void setup(){

    RCC_PLL_init();
    SysTick_init();
    UART2_init();

// PWM configuration ---------------------------------------------------------------
---------
    PWM_init(TRIG);         // PA_6: Ultrasonic trig pulse
    PWM_period_us(TRIG, 50000);    // PWM of 50ms period. Use period_us()
    PWM_pulsewidth_us(TRIG, 10);   // PWM pulse width of 10us


// Input Capture configuration -----------------------------------------------------
---------------------
    ICAP_init(ECHO);        // PB_6 as input caputre
    ICAP_counter_us(ECHO, 10);      // ICAP counter step time as 10us
    ICAP_setup(ECHO, 1, IC_RISE);  // TIM4_CH1 as IC1 , rising edge detect
    ICAP_setup(ECHO, 2, IC_FALL);  // TIM4_CH2 as IC2 , falling edge detect

}
```

**Example Code**

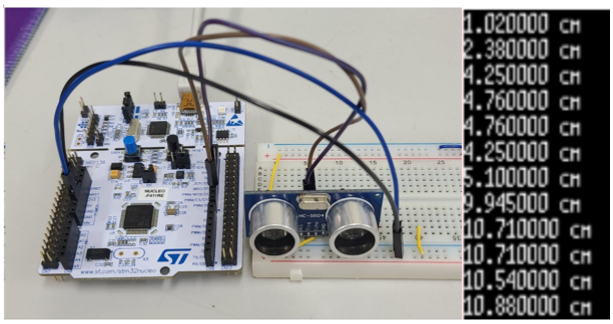
```
/**
******************************************************************************
* @author  SSSLAB
* @Mod       2023-10-31 by YKKIM
* @brief   Embedded Controller:  LAB - Timer Input Capture
```

```
*                                           - with Ultrasonic Distance Sensor
*
******************************************************************************
*/

#include "stm32f411xe.h"
#include "math.h"
#include "ecGPIO.h"
#include "ecRCC.h"
#include "ecTIM.h"
#include "ecPWM.h"
#include "ecUART_simple_student.h"
#include "ecSysTIck.h"

uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

#define TRIG PA_6
#define ECHO PB_6

void setup(void);

int main(void){

    setup();

    while(1){
        distance = (float) timeInterval * 340.0 / 2.0 / 10.0;   // [mm] -> [cm]
        printf("%f cm\r\n", distance);
        delay_ms(500);
    }
}

void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                          // Update interrupt
        __________                                                // overflow
count
        clear_UIF(TIM4);                                   // clear update
interrupt flag
    }
    if(is_CCIF(TIM4, 1)){                                  // TIM4_Ch1 (IC1)
Capture Flag. Rising Edge Detect
        time1 = __________;                                    // Capture TimeStart
        clear_CCIF(TIM4, 1);                 // clear capture/compare interrupt
flag
    }
    else if(__________){                                       // TIM4_Ch2 (IC2)
Capture Flag. Falling Edge Detect
        time2 = __________;                                    // Capture TimeEnd
        timeInterval = __________;  // (10us * counter pulse -> [msec] unit)
Total time of echo pulse
        ovf_cnt = 0;                         // overflow reset
        clear_CCIF(TIM4,2);                                  // clear
capture/compare interrupt flag
    }
```

```
}

void setup(){

    RCC_PLL_init();
    SysTick_init();
    UART2_init();

// PWM configuration -------------------------------------------------------------
---------
    __________;          // PA_6: Ultrasonic trig pulse
    PWM_period_us(TRIG, 50000);    // PWM of 50ms period. Use period_us()
    PWM_pulsewidth_us(TRIG, 10);   // PWM pulse width of 10us


// Input Capture configuration --------------------------------------------------
---------------------
    __________;      // PB_6 as input caputre
    ICAP_counter_us(ECHO, 10);       // ICAP counter step time as 10us
    ICAP_setup(ECHO, 1, IC_RISE);  // TIM4_CH1 as IC1 , rising edge detect
    __________;   // TIM4_CH2 as IC2 , falling edge detect

}
```

# Results

Images




Results

The ultrasonic distance sensor detects the distance between the sensor and the object. This is the principle that PWM is exported from the trigger and reflected when it meets an object and enters the echo. The ultrasonic distance sensor used this time can be measured to 400[cm]. In this LAB, 10[mm], 50[mm], and 100[mm] were measured, and the exact distance was confirmed through Terra Term.

### Demo Video

Link : [https://youtu.be/Y1fbwsmxXI8](https://youtu.be/Y1fbwsmxXI8)

# Reference

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

# Troubleshooting

I didn't think of units in the process of finding Timeinterval. The collect note only showed the formula without a unit, so I did it as it was, but it did not work normally. As it turned out, Timeinterval was in units of 10 micro. Therefore, the problem was solved by multiplying 10 and dividing 1000 to change it to 1[msec]. Of course, we can give it away by 100.

# Appendix

- ecGPIO.c

```c
#include "stm32f4xx.h"
#include "stm32f411xe.h"
#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);
}
```

- ecEXTI.c

```
#include "ecGPIO.h"
#include "ecSysTick.h"
```

```c
#include "ecEXTI.h"


void EXTI_init(GPIO_TypeDef *Port, int Pin, int trig_type,int priority){

    // SYSCFG peripheral clock enable
    RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN;

    // Connect External Line to the GPIO
    int EXTICR_port;
    if          (Port == GPIOA) EXTICR_port = 0;
    else if (Port == GPIOB) EXTICR_port = 1;
    else if (Port == GPIOC) EXTICR_port = 2;
    else if (Port == GPIOD) EXTICR_port = 3;
    else                                        EXTICR_port = 4;

    SYSCFG->EXTICR[Pin >> 2] &= ~(15 << 4*(Pin & 0x03));            // clear 4 bits
    SYSCFG->EXTICR[Pin >> 2] |= EXTICR_port << 4*(Pin & 0x03);           // connect port number

    // Configure Trigger edge
    if (trig_type == FALL) EXTI->FTSR |= 1UL << Pin;   // Falling trigger enable
    else if (trig_type == RISE) EXTI->RTSR |= 1UL << Pin;   // Rising trigger enable
    else if (trig_type == BOTH) {           // Both falling/rising trigger enable
        EXTI->RTSR |= 1UL << Pin;
        EXTI->FTSR |= 1UL << Pin;
    }

    // Configure Interrupt Mask (Interrupt enabled)
    EXTI->IMR  |= 1 << Pin;     // not masked


    // NVIC(IRQ) Setting
    int EXTI_IRQn = 0;

    if (Pin == 0) EXTI_IRQn = EXTI0_IRQn;
    else if (Pin == 1) EXTI_IRQn = EXTI1_IRQn;
    else if (Pin == 2) EXTI_IRQn = EXTI2_IRQn;
    else if (Pin == 3) EXTI_IRQn = EXTI3_IRQn;
    else if (Pin == 4) EXTI_IRQn = EXTI4_IRQn;
    else if (Pin > 4 && Pin < 10)   EXTI_IRQn = EXTI9_5_IRQn;
    else            EXTI_IRQn = EXTI15_10_IRQn;

    NVIC_SetPriority(EXTI_IRQn, priority);  // EXTI priority
    NVIC_EnableIRQ(EXTI_IRQn);  // EXTI IRQ enable
}


void EXTI_enable(uint32_t pin) {
    EXTI->IMR |= 1UL << pin;     // not masked (i.e., Interrupt enabled)
}
void EXTI_disable(uint32_t pin) {
    EXTI->IMR |= ~(1UL << pin);     // masked (i.e., Interrupt disabled)
}
```

```
uint32_t is_pending_EXTI(uint32_t pin){
    uint32_t EXTI_PRx = pin;        // check  EXTI pending
    return ((EXTI->PR & 1 << pin) == 1 << pin);
}


void clear_pending_EXTI(uint32_t pin){
    EXTI->PR |= 1 << pin;     // clear EXTI pending
}
```

- ecSysTick.c

```
#include "ecSysTick.h"

#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000

volatile uint32_t msTicks=0;

//EC_SYSTEM_CLK

void SysTick_init(void){
    //  SysTick Control and Status Register
    SysTick->CTRL = 0;                                          // Disable
SysTick IRQ and SysTick Counter

    // Select processor clock
    // 1 = processor clock;  0 = external clock
    SysTick->CTRL |= SysTick_CTRL_CLKSOURCE_Msk;

    // uint32_t MCU_CLK=EC_SYSTEM_CLK
    // SysTick Reload Value Register
    SysTick->LOAD = MCU_CLK_PLL / 1000 - 1;                    // 1ms, for
HSI PLL = 84MHz.

    // SysTick Current Value Register
    SysTick->VAL = 0;

    // Enables SysTick exception request
    // 1 = counting down to zero asserts the SysTick exception request
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    // Enable SysTick IRQ and SysTick Timer
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;

    NVIC_SetPriority(SysTick_IRQn, 16);     // Set Priority to 1
    NVIC_EnableIRQ(SysTick_IRQn);           // Enable interrupt in NVIC
}



void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(){
```

```
    msTicks++;
}


void delay_ms (uint32_t mesc){
  uint32_t curTicks;

  curTicks = msTicks;
  while ((msTicks - curTicks) < mesc);

    msTicks = 0;
}

//void delay_ms(uint32_t msec){
//  uint32_t now=SysTick_val();
//  if (msec>5000) msec=5000;
//  if (msec<1) msec=1;
//  while ((now - SysTick_val()) < msec);
//}


void SysTick_reset(void)
{
    // SysTick Current Value Register
    SysTick->VAL = 0;
}

uint32_t SysTick_val(void) {
    return SysTick->VAL;
}

//void SysTick_counter(){
//  msTicks++;
//  if(msTicks%1000 == 0) count++;
//}

void SysTick_enable(void) {
    NVIC_EnableIRQ(SysTick_IRQn);
}

void SysTick_disable(void) {
    NVIC_DisableIRQ(SysTick_IRQn);
}
```

- ecPinNames.c

```
#include "ecPinNames.h"

void ecPinmap(PinName_t pinName, GPIO_TypeDef **GPIOx, unsigned int *pin)
{

    unsigned int pinNum= pinName & (0x000F);
    *pin=pinNum;

    unsigned int portNum=(pinName>>4);
```

```
    if (portNum==0)
        *GPIOx=GPIOA;
    else if (portNum==1)
        *GPIOx=GPIOB;
    else if (portNum==2)
        *GPIOx=GPIOC;
    else if (portNum==3)
        *GPIOx=GPIOD;
    else if (portNum==7)
        *GPIOx=GPIOH;
    else
        *GPIOx=GPIOA;
}
```

- ecTIM.c

```
#include "gather.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_CC3S_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;
    else if (ICn == 2)
        capture_Value = TIMx->CCR2;
    else if (ICn == 2)
        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;
        }
    }
}
```

- 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);
}
```

```c
// 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);


// 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
```

```c
        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.
}
```

- ecUART_simple.c

```c
#include "ecUART_simple.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);
}



void UART2_init(){
    // Enable the clock of USART2
    RCC->APB1ENR |= RCC_APB1ENR_USART2EN;  // Enable USART 2 clock (APB1 clock: AHB clock / 2 = 42MHz)

    // Enable the peripheral clock of GPIO Port
    RCC->AHB1ENR |=   RCC_AHB1ENR_GPIOAEN;

    // ********************** USART 2 ***************************
    // PA2 = USART2_TX
    // PA3 = USART2_RX
    // Alternate function(AF7), High Speed, Push pull, Pull up
    // **********************************************************
    int TX_pin = 2;

    GPIOA->MODER   &= ~(0xF << (2*TX_pin)); // Clear bits
    GPIOA->MODER    |=   0xA << (2*TX_pin);  // Alternate Function(10)
    GPIOA->AFR[0]   |=   0x77<< (4*TX_pin);  // AF7 - USART2
    GPIOA->OSPEEDR  |=   0xF<<(2*TX_pin);        // High speed (11)
    GPIOA->PUPDR   &= ~(0xF<<(2*TX_pin));
    GPIOA->PUPDR    |=   0x5<<(2*TX_pin);    // Pull-up (01)
    GPIOA->OTYPER  &=  ~(0x3<<TX_pin) ;         // push-pull (0, reset)


    USART_TypeDef *USARTx = USART2;
    // No hardware flow control, 8 data bits, no parity, 1 start bit and 1 stop bit
    USARTx->CR1 &= ~USART_CR1_UE;           // Disable USART

    // Configure word length to 8 bit
    USARTx->CR1 &= ~USART_CR1_M;        // M: 0 = 8 data bits, 1 start bit
```

```
    USARTx->CR1 &= ~USART_CR1_PCE;          // No parrity bit
    USARTx->CR2 &= ~USART_CR2_STOP;         // 1 stop bit

    // Configure oversampling mode (to reduce RF noise)
    USARTx->CR1 &= ~USART_CR1_OVER8;    // 0 = oversampling by 16

    // 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

    //USARTx->BRR  = 42000000/ baud_rate;
    float Hz = 42000000;

    float USARTDIV = (float)Hz/16/9600;
    uint32_t MNT = (uint32_t)USARTDIV;
    uint32_t FRC = round((USARTDIV - MNT) * 16);
    if (FRC > 15) {
        MNT += 1;
        FRC = 0;
    }
    USARTx->BRR |= (MNT << 4) | FRC;

    USARTx->CR1 |= (USART_CR1_RE | USART_CR1_TE);       // Transmitter and
Receiver enable
    USARTx->CR3 |= USART_CR3_DMAT | USART_CR3_DMAR;
    USARTx->CR1 |= USART_CR1_UE;                                    //
USART enable

    USARTx->CR1 |= USART_CR1_RXNEIE;                    // Enable Read
Interrupt
    NVIC_SetPriority(USART2_IRQn, 1);       // Set Priority to 1
    NVIC_EnableIRQ(USART2_IRQn);                // Enable interrupt of
USART2 peripheral

}

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;

}

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

```
}
```

- gather.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 "math.h"
```