ecLAB: Line Tracing RC Car

Date: 2023-11-23

Author/Partner: Han Tae Geon / Jang Ho Jin

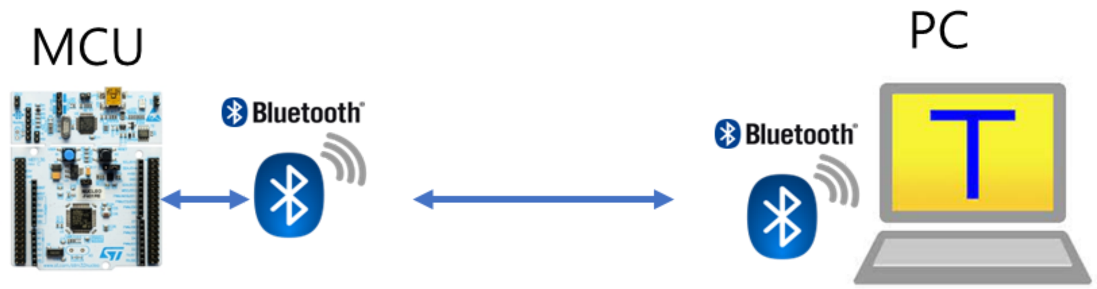
Github: https://github.com/hhangun/EC-taegeon-793

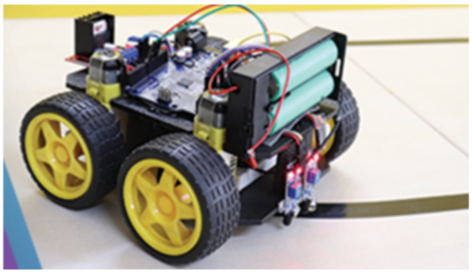
Demo Video: Manual Mode : https://youtu.be/icd1kjkOtf8

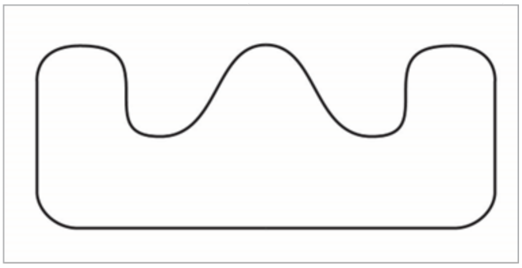
Automatic Mode : https://youtu.be/Pg68u4XeRaY

Introduction

Design an embedded system to control an RC car to drive on the racing track. The car is controlled either manually with wireless communication or automatically to drive around the track. When it sees an obstacle on the driving path, it should temporarily stop until the obstacle is out of the path.







Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others: Minimum
  - Bluetooth Module(HC-06)
  - DC motor x2, DC motor driver(L9110s)
  - IR Reflective Sensor (TCRT 5000) x2
  - HC-SR04
  - additional sensor/actuators are acceptable

Software

- Keil uVision, CMSIS, EC_HAL library

Preparation

Tutorials:

Complete the following tutorials:

1. TU: Managing library header files
2. TU: Custom initialization

Use `ecSTM32F411.h` and `void MCU_init(void)` in your project code.

LABS:

You should review previous labs for help

1. LAB: ADC IR Sensor
2. LAB: USART Bluetooth
3. LAB: Timer & PWM

Problem Definition

Design your RC car that has the following functions:

1. Line tracing on the given racing track
2. has 2 control modes: **Manual Mode** to **AUTO Mode**
3. stops temporally when it detects an object nearby on the driving path

On the PC, connected to MCU via bluetooth

- Print the car status every 1 sec such as “ (“ MOD: A DIR: F STR: 00 VEL: 00 ”)

Manual Mode

- Mode Change(MOD):
  - When 'M' or 'm' is pressed, it should enter **Manual Mode**
  - LD2 should be ON in Manual Mode
- Speed (VEL):
  - Increase or decrease speed each time you push the arrow key “UP” or “DOWN”, respectively.
  - You can choose the speed keys
  - Choose the speed level: V0 ~ V3
- Steer (STR):
  - Steering control with keyboard keys
  - Increase or decrease the steering angles each time you press the arrow key “RIGHT” or “LEFT”, respectively.
  - Steer angles with 3 levels for both sides: e.g: -3, -2, -1, 0, 1, 2, 3 // '-' angle is turning to left

- Driving Direction (DIR)
  - Driving direction is forward or backward by pressing the key “F” or “B”, respectively.
  - You can choose the control keys
- Emergency Stop
  - RC car must stop running when key “S” is pressed.

Automatic Mode

- Mode Change:
  - When 'A' or 'a' is pressed, it should enter **AUTO Mode**
- LD2 should blink at 1 second rate in AUTO Mode
- It should drive on the racing track continuously
- Stops temporally when it detects an object nearby on the driving path
- If the obstacle is removed, it should drive continuously

Procedure

1. Discuss with the teammate how to design an algorithm for this problem
2. In the report, you need to explain concisely how your system works with state tables/diagram or flow-chart.

   ● Listing all necessary states (states, input, output etc) to implement this design problem.

   ● Listing all necessary conditional FLAGS for programming.

   ● Showing the logic flow from the initialization

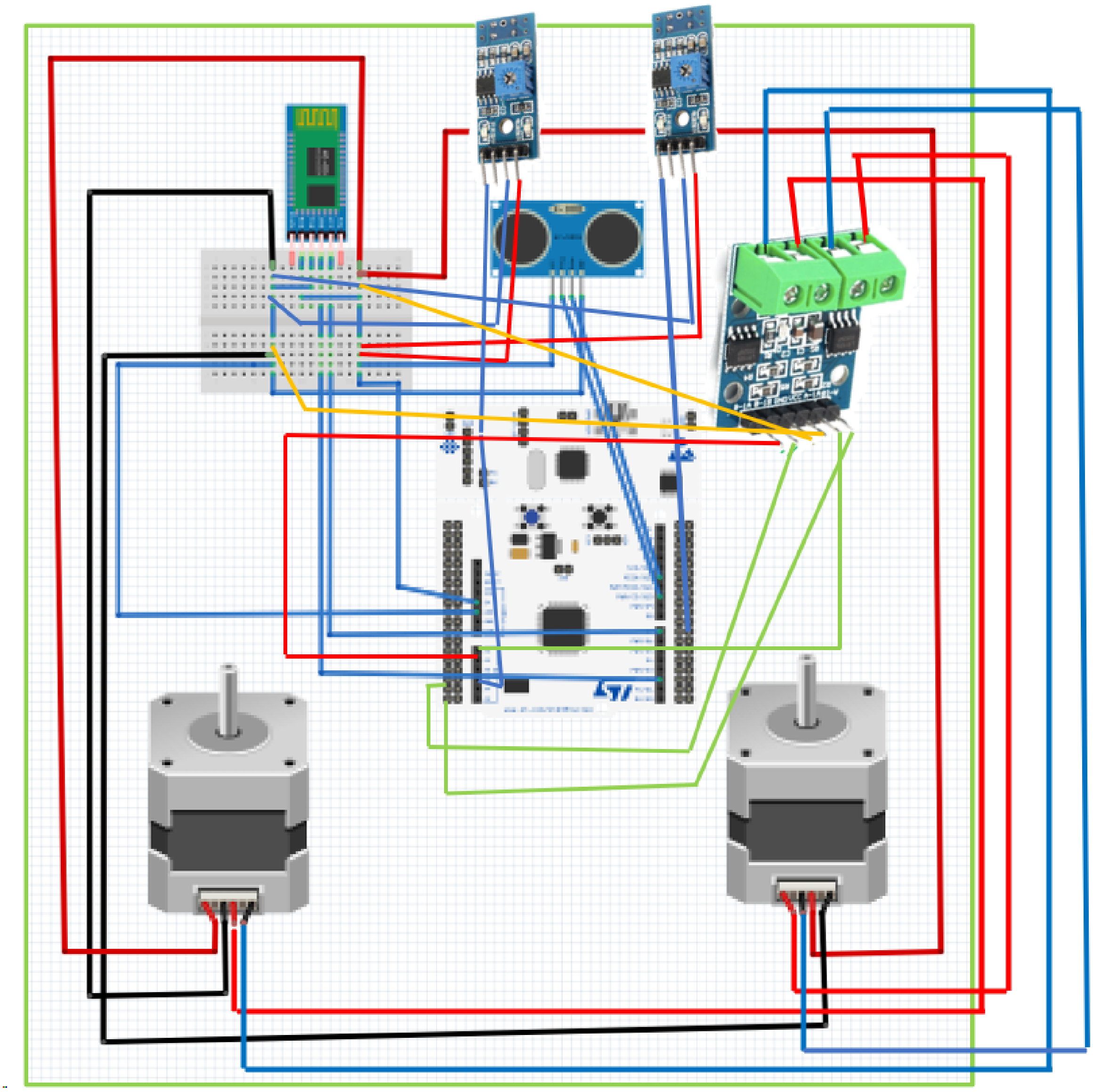
   and more
3. Select appropriate configurations for the design problem. Fill in the table.

Functions	Register	PORT_PIN	Configuration
System Clock	RCC		PLL 84MHz
delay_ms	SysTick		
Motor DIR	Digital Out		
	....		
LED2	Digital Out	PA5	
Ultrasonic	TIM4	PA6, PB6	NO Pull-up Pull-down, Push-Pull, Fast, AF
TIMER	TIMER1		
	TIMER2		
Timer Interrupt	...		1msec
ADC	ADC	PB_0, PB_1	ADC Clock Prescaler /8 12-bit resolution, right alignment Continuous Conversion mode Scan mode: Two channels in regular group External Trigger @ 1kHz Trigger Detection on Rising Edge
	....		
DC Motor Speed	PWM2	PA0, PA1	NO Pull-up Pull-down, Push-Pull, Fast, AF
ADC sampling trigger	PWM3		External Trigger @ 1kHz Trigger Detection on Rising Edge
RS-232 USB cable(ST-LINK)	USART2		No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate
Bluetooth	USART1	TXD: PA9 RXD: PA10	No Parity, 8-bit Data, 1-bit Stop bit 9600 baud-rate

4. Create a new project under the directory `\repos\EC\LAB\LAB_RCcar`

- The project name is “**LAB_RCcar”**
- You can share the same code with your teammate. But need to write the report individually

Circuit Diagram



Code

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

```
#include "ecSTM32F411.h"

#define MAX_BUF     10
#define END_CHAR     13
#define MOTOR_A    2
#define MOTOR_B    3
#define TRIG PA_6
#define ECHO PB_6


// 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 char mode = ' ';
static char Direction = ' ';     // direction control
```

```
static uint32_t count = 0;   // led count
static uint32_t cnt = 3;    // angle control
static int VER = 1;     // speed control
static int auto_state = 0;


static double dutyA = 0;      // PWM of Motor A
static double dutyB = 0;   // PWM of Motor B
static unsigned int dir = 1;

static volatile int bReceive = 0; // flag
static volatile int state = 3;  // Mode discrimination

// ADC
static PinName_t seqCHn[2] = {PB_0, PB_1};
static uint32_t value1, value2;
static int flag = 0;

// UltraSonic
static uint32_t ovf_cnt = 0;
static double distance = 0;
static float timeInterval = 0;
static float time1 = 0;
static float time2 = 0;

void setup(void);
void manual_mode();
void angle_go();
void angle_back();

// 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 msec PWM period
   PWM_period_us(PA_1, 200);   // 1 msec PWM period

    // Ultrasonic
    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

    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

   // 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);

      // TIM3
      TIM_UI_init(TIM3, 1);         // TIM3 Update-Event Interrupt every 1 msec
    TIM_UI_enable(TIM3);


    // ADC Init
    ADC_init(PB_0);   // priority 1
    ADC_init(PB_1);

    // ADC channel sequence setting
    ADC_sequence(seqCHn, 2);
}
```

Various functions are executed, and the state is printed each time they are executed.

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

    while(1){
            distance = (float) timeInterval * 340.0 / 2.0 / 10.0;     // [mm] ->
[cm]
            if(distance < 0) distance = distance*(-1);
        if (bReceive == 1){         // flag
            bReceive = 0;

            if(state == 0){
                manual_mode();
            }
                else if(state == 1){
                                // Erase distance error
                               if (distance >= 400) continue;
                }
                             PWM_duty(PA_0, dutyA);
                             PWM_duty(PA_1, dutyB);
        }
                if(state == 0){
                // Display values in Tera Term
                            USART_write(USART1,(uint8_t*)"MOD : ", 6);
                            USART_write(USART1, &mode, 1);
                            USART_write(USART1, (uint8_t*)" DIR : ", 7);
                            USART_write(USART1, &Direction, 1);
                            USART_write(USART1, (uint8_t*)" STR : ", 7);
                            char ssttrr[20];
                            int len = sprintf(ssttrr, "%d", cnt);
                            USART_write(USART1, (uint8_t*)ssttrr, len);
```

```
                    USART_write(USART1, &cnt, 1);
                    USART_write(USART1, (uint8_t*)" VER : ", 7);
                    char vveerr[20];
                    int len2 = sprintf(vveerr, "%d", VER);
                    USART_write(USART1, (uint8_t*)vveerr, len2);

                    USART_write(USART1, (uint8_t*)"\r\n", 2);
            }
            else if(state == 1){
                    // Display values in Tera Term
                    USART_write(USART1,(uint8_t*)"distance : ", 11);
                    char dist[20];
                    int d = sprintf(dist, "%f", distance);
                    USART_write(USART1, (uint8_t*)dist, d);

                    USART_write(USART1,(uint8_t*)" state : ", 9 );
                    char auto_st[30];
                    if(auto_state == 1 || auto_state == 4){
                        sprintf(auto_st, " state : GO straight");
                        USART_write(USART1, &auto_st, 20); }
                    else if(auto_state == 2){
                        sprintf(auto_st, " state : Right");
                        USART_write(USART1, &auto_st, 14); }
                    else if(auto_state == 3){
                        sprintf(auto_st, " state : Left");
                        USART_write(USART1, &auto_st, 13); }
                    else if(auto_state == 0){
                        sprintf(auto_st, " state : ");
                        USART_write(USART1, &auto_st, 9); }
                    else if(auto_state == 5){
                        sprintf(auto_st, " state : Stop");
                        USART_write(USART1, &auto_st, 13); }
                    USART_write(USART1, (uint8_t*)"\r\n", 2);
                }
                    delay_ms(1000);
        }
}
```

It is a TIM interrupt execution function and allows the LED to blink every second.

```
void TIM3_IRQHandler(void){

   if(is_UIF(TIM3)){         // Check UIF(update interrupt flag)
         // If modeA, LED blink at 1 sec
      if(BT_data == 'a' || BT_data == 'A'){
         count++;
         if (count <= 1000) GPIO_write(GPIOA, LED_PIN, 0);
            else if(count <= 2000)
               GPIO_write(GPIOA, LED_PIN, 1);
               else
                  count = 0;
            }
            else if(BT_data == 'm' || BT_data == 'M')
                GPIO_write(GPIOA, LED_PIN, 1);
      clear_UIF(TIM3);       // Clear UI flag by writing 0
   }
```

```
}
```

It is a TIM interrupt execution function and ultrasonic sensors are used to help measure the distance to the object.

```
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
   }
}
```

Get the data from the PC and show it on the Tera term.

```
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
   }
}
```

Control RC car movement by setting mode through communication function

```
void USART1_IRQHandler(){                    // USART2 RX Interrupt : Recommended
   if(is_USART1_RXNE()){
      USART_write(USART1, (uint8_t*)"Data = ", 7);
          BT_data = USART1_read();                    // RX from USART1
      USART_write(USART1, &BT_data, 1);    // TX to USART1

            USART_write(USART1, (uint8_t*)"\r\n", 2);

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

      else if(BT_data == 'm' || BT_data == 'M'){
            mode = 'M';
            dutyA = 1;    //stop
```

```
            dutyB = 1;
        state = 0;
         }
      else if(BT_data == 'a' || BT_data == 'A'){
            mode = 'A';
            Direction = 'F';
            dutyA = 0;
            dutyB = 0;
        state = 1;
         }
      bReceive = 1;    // flag = 1
      }
}
```

Control RC car for sensor values by controlling the IR sensor using the ADC function

```
void ADC_IRQHandler(void){
   if(is_ADC_OVR())
      clear_ADC_OVR();

   if(is_ADC_EOC()){      // after finishing sequence
      if (flag==0)
         value1 = ADC_read();
      else if (flag==1)
         value2 = ADC_read();

      flag =! flag;      // flag toggle
        }
         if(state == 1){
       if(value1 < 1000 && value2 < 1000){   // Go straight
                                    dutyA = 1;
                                    dutyB = 1;
                                    auto_state = 1;
                                }
                                else if(value1 < 1000 && value2 > 1000){   // Go
right
                                    dutyA = 0.8;
                                    dutyB = 1;
                                    auto_state = 2;
                                }
                                else if(value1 > 1000 && value2 < 1000){   // Go
left
                                    dutyA = 1;
                                    dutyB = 0.8;
                                    auto_state = 3;
                                }
                                else if(value1 > 1000 && value2 > 1000){
                                    dutyA = 1;
                                    dutyB = 1;
                                    auto_state = 4;
                                }
                                if(distance < 20){
                                    dutyA = 0;
                                    dutyB = 0;
                                    auto_state = 5;
                                }
                            PWM_duty(PA_0, dutyA);
```

```
                                PWM_duty(PA_1, dutyB);
    }
}
```

In the manual mode, the tasks to be performed were implemented as functions.

```
void manual_mode(){
        double sp = 0.1;
         if(BT_data == 'B') {
                Direction = 'B';    // for display in tera term
                dir = 1;
                                VER = 1;
                dutyA = 0.5;
                dutyB = 0.5;
            }
         else if(BT_data == 'F') {
                Direction = 'F';    // for display in tera term
                dir = 0;
                                VER = 1;
                dutyA = 0.5;
                dutyB = 0.5;
         }
               if(dir == 1){

               // Control a car speed
               switch(BT_data){
                  case 'W' :    // Go Right
                                        if(cnt>0){
                                        cnt--;
                                        angle_go();
                                        }
                                        else {}     break;
                  case 'Q' :        // Go Left
                                        if(cnt<6){
                                        cnt++;
                    angle_go();
                                        }
                                        else {}     break;
                  case 'S' :   // Stop
                     dutyA = 1;
                     dutyB = 1;   break;

                  default     : break;
               }
                  // Control Speed
                  if(BT_data == 'O'){   // speed down
                     dutyA += sp;
                     dutyB += sp;
                                         VER--;
                  }
                  else if(BT_data == 'P'){  // speed up
                     dutyA -= sp;
                     dutyB -= sp;
                                        VER++;
                  }
                                    if(VER < 0){    // speed limit
                                        VER = 0;
```

```
                                    dutyA -= sp;
                                    dutyB -= sp;
                                }
                                else if(VER > 3){   // speed limit
                                    VER = 3;
                                    dutyA += sp;
                                    dutyB += sp;
                                }
                }

                else if(dir == 0){
                   switch(BT_data){
                   case 'Q' :    // Go left
                     if(cnt<6){
                                    cnt++;
                                    angle_back();
                                    }
                                    else {}     break;
                   case 'W' :        // Go right
                                    if(cnt>0){
                                    cnt--;
                     angle_back();
                                    }
                                    else {}     break;
                   case 'S' :   // Stop
                      dutyA = 0;
                      dutyB = 0;    break;

                   default    : break;
                }
                   //Control speed
                   if(BT_data == 'P'){   // speed up
                      dutyA += sp;
                      dutyB += sp;
                                     VER++;
                   }
                   else if(BT_data == 'O'){  // speed down
                      dutyA -= sp;
                      dutyB -= sp;
                                    VER--;
                   }
                                if(VER < 0){    // speed limit
                                    VER = 0;
                                    dutyA += sp;
                                    dutyB += sp;
                                }
                                else if(VER > 3){   // speed limit
                                    VER = 3;
                                    dutyA -= sp;
                                    dutyB -= sp;
                                }
            }
         GPIO_write(GPIOC, MOTOR_A, dir);
         GPIO_write(GPIOC, MOTOR_B, dir);
}
```

It is a function that enables angle adjustment, one of the manual mode tasks.

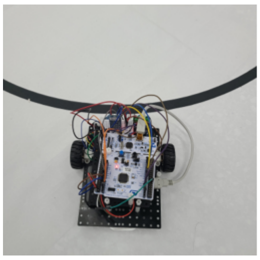
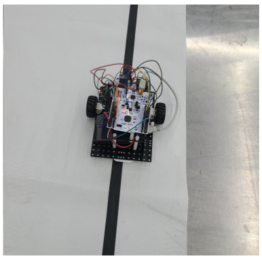
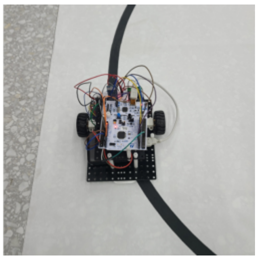
```
void angle_go(){
    if(cnt == 3){
    dutyA = 0.5;
    dutyB = 0.5;
    }
    else if(cnt == 2){
        dutyA = 0.6;
        dutyB = 0.3;
    }
    else if(cnt == 1){
        dutyA = 0.6;
        dutyB = 0.2;
    }
    else if(cnt <= 0){
        dutyA = 0.6;
        dutyB = 0.1;
    }
    else if(cnt == 4){
        dutyA = 0.3;
        dutyB = 0.6;
    }
    else if(cnt == 5){
        dutyA = 0.2;
        dutyB = 0.6;
    }
    else if(cnt <= 6){
        dutyA = 0.1;
        dutyB = 0.6;
    }
}

// Angle change with cnt value at
void angle_back(){
    if(cnt == 3){
    dutyA = 0.5;
    dutyB = 0.5;
    }
    else if(cnt == 2){
        dutyA = 0.4;
        dutyB = 0.7;
    }
    else if(cnt == 1){
        dutyA = 0.4;
        dutyB = 0.8;
    }
    else if(cnt <= 0){
        dutyA = 0.4;
        dutyB = 0.9;
    }
    else if(cnt == 4){
        dutyA = 0.7;
        dutyB = 0.4;
    }
    else if(cnt == 5){
        dutyA = 0.8;
        dutyB = 0.4;
    }
```

```
    else if(cnt <= 6){
        dutyA = 0.9;
        dutyB = 0.4;
    }
}
```

Results

Images

results

First, power is applied to the MCU board. Then, the MCU board that receives the data from the PC is controlled by Bluetooth. If a or A is inputted into the Tera term, the automatic mode is executed, and if m or B is inputted, the manual mode is executed.

In Automatic mode, the LED flashes every second. In addition, line tracing is performed using the IR sensor. The driving direction is determined by comparing the values of the two sensors. The RC car is set to stop when an object is detected in front of it while driving. The distance was set at 20 cm, and an ultrasonic sensor was used. When an object is detected and disappeared, it moves back to its original state. Of course, the LED blinks every second even when you're on the move. The distance and direction of the car are printed on the Tera term.

In Manual mode, the LED remains on. And using O and P, the speed was set to be adjusted from stage 0 to stage 3. It cannot be adjusted to less than 0 and cannot be adjusted to more than 3 levels. Next, it's angle adjustment. It was set to give the angle from 0 to 6 using Q and W. The initial state starts with 3 and presses Q, and the angle bends to the left and to the right when pressing W. If the number decreases or increases based on 3, the angle of movement becomes larger. Next, the direction of movement. Press F to move the RC car forward, and press B to move back. Finally, pressing S will stop the car under any circumstances. In all of these processes, the LED is always on, and values related to mode, direction, angle, and speed are printed on the Tera term.

Flow chart

a, A : Automatic mode

m, M : Manual mode

F : Direction is forward

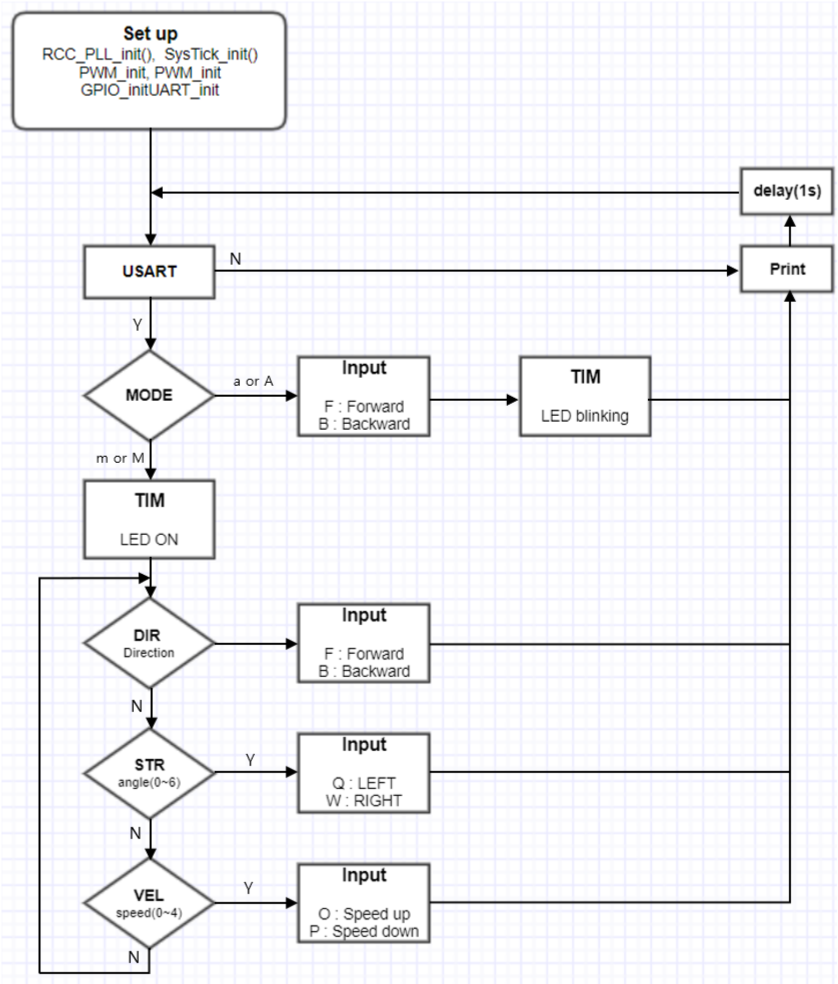
B : Direction is backward

Q : Go left

W : Go right

O : Speed up

P : Speed down



Demo Video

Manual Mode : https://youtu.be/icd1kjkOtf8

Automatic Mode : https://youtu.be/Pg68u4XeRaY

Reference

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

Troubleshooting

1. motor PWM duty ratio for different DIR

When, DIR=0 duty=0.8--> PWM 0.8 // pwm delivered to the actual motor

When, DIR=1 duty=0.8--> PWM 0.2 // pwm delivered to the actual motor

```
float targetPWM;  // pwm for motor input

float duty=abs(DIR-targetPWM); // duty with consideration of DIR=1 or 0
PWM_duty(PWM_PIN, duty);c
```

As the direction of the motor changes, the duty also changes in the opposite direction. This is expected to be an optimal design for motor control. Therefore, in this LAB, the duty ratio was adjusted according to the direction.

2. Print a string for BT (USART1)

Use `sprintf()`

```
\#define _CRT_SECURE_NO_WARNINGS    // sprintf 보안 경고로 인한 컴파일 에러 방지
\#include <stdio.h>      // sprintf 함수가 선언된 헤더 파일
char BT_string[20]=0;
int main()
{
•   sprintf(BT_string, "DIR:%d PWM: %0.2f\n", dir, duty);    // 문자, 정수, 실수를 문자열로 만듦
•   USART1_write(BT_string, 20);
•   // ...
}
```

https://dojang.io/mod/page/view.php?id=352 **

When the sprintf is used, it is easy to control various types of variable types within the string. In particular, it is useful because it can create a string by combining constant text and variable values. It is often used when variable content is included.

3. Motor does not run under duty 0.5

SOL) Configure motor PWM period as 1kHa

4. Check and give different Interrupt Priority

Check if you have different NVIC priority number for each IRQs

5. Ultrasoninc sensor does not measure properly when MCU is connected with motor driver

SOL) Give independent voltage source to motor driver. Giving DC power from MCU to motor driver is not recommended

Appendix

ecADC.h

```
#ifndef __MY_ADC_H
#define __MY_ADC_H

#include "stm32f411xe.h"
#include "ecSTM32F411.h"

// ADC trigmode
#define SW 0
#define TRGO 1

// ADC contmode
#define CONT 0
#define SINGLE 1

// Edge Type
#define RISE 1
#define FALL 2
#define BOTH 3

#define _DEFAULT 0

/////////////////////////////////////////////////////
// ADC default setting
/////////////////////////////////////////////////////

// ADC init
// Default:  one-channel mode, continuous conversion
// Default: HW trigger - TIM3 counter, 1msec
void ADC_init(PinName_t pinName);
void JADC_init(PinName_t pinName);


// Multi-Channel Scan Sequence
void ADC_sequence(PinName_t *seqCHn, int seqCHnums);
void JADC_sequence(PinName_t *seqCHn, int seqCHnums);

void ADC_start(void);
void JADC_start(void);

// flag for ADC interrupt
uint32_t is_ADC_EOC(void);
uint32_t is_ADC_OVR(void);
void clear_ADC_OVR(void);

// read ADC value
```

```
uint32_t ADC_read(void);


///////////////////////////////////////////////////////
// Advanced Setting
///////////////////////////////////////////////////////
// Conversion mode change: CONT, SINGLE / Operate both ADC,JADC
void ADC_conversion(int convMode);
void ADC_trigger(TIM_TypeDef* TIMx, int msec, int edge);

// JADC setting
void JADC_trigger(TIM_TypeDef* TIMx, int msec, int edge);


// Private Function
void ADC_pinmap(PinName_t pinName, uint32_t *chN);

#endif
```

ecADC.c

```
#include "stm32f411xe.h"
#include "ecSysTick.h"
#include "ecADC.h"
#include "ecGPIO.h"
#include "ecTIM.h"
#include <stdint.h>


/* ------------------------------------------------------------------------------
-------*/
//                                  ADC Configuration
//
/* ------------------------------------------------------------------------------
-------*/

void ADC_init(PinName_t pinName){  // trigmode 0 : SW, 1 : TRGO

// 0. Match Port and Pin for ADC channel
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    int chN;
    ADC_pinmap(pinName, &chN);          // ADC Channel <->Port/Pin mapping

// GPIO configuration -----------------------------------------------------------
----------
// 1. Initialize GPIO port and pin as ANALOG, no pull up / pull down
    GPIO_init(port, pin, ANALOG);               // ANALOG = 3
    GPIO_pupd(port, pin, EC_NONE);              // EC_NONE = 0

// ADC configuration    --------------------------------------------------------
-------------
// 1. Total time of conversion setting
    // Enable ADC pheripheral clock
    RCC->APB2ENR  |= RCC_APB2ENR_ADC1EN;        // Enable the clock of
RCC_APB2ENR_ADC1EN
```

```
    // Configure ADC clock pre-scaler
    ADC->CCR &= ~ADC_CCR_ADCPRE;                // 0000: PCLK2 divided by 2
(42MHz)

    // Configure ADC resolution
    ADC1->CR1 &= ~ADC_CR1_RES;                  // 00: 12-bit resolution
(15cycle+)

    // Configure channel sampling time of conversion.
    // Software is allowed to write these bits only when ADSTART=0 and JADSTART=0
   !!
    // ADC clock cycles @42MHz = 2us
    if(chN < 10) {
        ADC1->SMPR2 &= ~(7UL << (3* chN));          // clear bits
        ADC1->SMPR2 |= 4U << (3* chN);              // sampling time conversion
: 84
    }
    else{
        ADC1->SMPR1 &= ~(7UL << (3* (chN - 10)));
        ADC1->SMPR1 |= 4U << (3* (chN - 10));
    }

// 2. Regular / Injection Group
    //Regular: SQRx, Injection: JSQx

// 3. Repetition: Single scan or Continuous scan conversion
    ADC1->CR2 |= ADC_CR2_CONT;                  // default : Continuous
conversion mode

// 4. Single(one) Channel or Scan(multi-channel) mode
    // Configure the sequence length            // default: one-channel length
    ADC1->SQR1 &= ~ADC_SQR1_L;                  // 0000: one channel length in
the regular channel conversion sequence

    // Configure the multiple channel sampling sequence
    ADC1->SQR3 &= ~ADC_SQR3_SQ1;                // SQ1 clear
    ADC1->SQR3 |= (chN & ADC_SQR3_SQ1);         // Choose the first channelID to
sample

    // Default:  Single(one-channel) Channel mode
    ADC1->CR1 &= ~ADC_CR1_SCAN;                 // 0: One-channel mode

// 5. Interrupt Enable
    // Enable EOC(conversion) interrupt.
    ADC1->CR1 &= ~ADC_CR1_EOCIE;          // Interrupt reset
    ADC1->CR1 |=  ADC_CR1_EOCIE;           // Interrupt enable

    // Enable ADC_IRQn
    NVIC_SetPriority(ADC_IRQn, 2);              // Set Priority to 2
    NVIC_EnableIRQ(ADC_IRQn);                   // Enable interrupt form ACD1
peripheral


    // Hardware Trigger Configuration : TIM3, 1msec, RISE edge
    ADC_trigger(TIM3, 1, RISE);

    // Start ADC
```

```
    ADC_start();
}


void ADC_trigger(TIM_TypeDef* TIMx, int msec, int edge){
    // set timer
    int timer = 0;
    if(TIMx == TIM2) timer = 2;
    else if(TIMx == TIM3) timer = 3;

    // Single conversion mode (disable continuous conversion)
    ADC1->CR2 &= ~ADC_CR2_CONT;     // Discontinuous conversion mode
    ADC1->CR2 |= ADC_CR2_EOCS;          // Enable EOCS

    // Enable TIMx Clock as TRGO mode
// 1. TIMx Trigger Output Config
    // Enable TIMx Clock
    TIM_init(TIMx, msec);
    TIMx->CR1 &= ~TIM_CR1_CEN;                              //counter disable

    // Set PSC, ARR
  TIM_period_ms(TIMx, msec);

  // Master Mode Selection MMS[2:0]: Trigger output (TRGO)
  TIMx->CR2 &= ~TIM_CR2_MMS;                  // reset MMS
  TIMx->CR2 |= TIM_CR2_MMS_2 | TIM_CR2_MMS_1;           //100: Compare - OC1REF
signal is used as trigger output (TRGO)

    // Output Compare Mode
  TIMx->CCMR2 &= ~TIM_CCMR2_OC3M;                              // OC1M :
output compare 1 Mode
  TIMx->CCMR2 |= TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2; // OC1M = 110 for compare
1 Mode ch1

  // OC1 signal
  TIMx->CCER |= TIM_CCER_CC3E;  // CC1E Capture enabled
    TIMx->CCR3  = (TIMx->ARR)/2;  // duty ratio 50%

  // Enable TIMx
  TIMx->CR1 |= TIM_CR1_CEN;                              //counter enable

// 2. ADC HW Trigger Config.
    // Select Trigger Source
    ADC1->CR2 &= ~ADC_CR2_EXTSEL;           // reset EXTSEL
    ADC1->CR2 |= (timer*2 + 2) << 24;   // TIMx TRGO event (ADC : TIM2, TIM3
TRGO)

    //Select Trigger Polarity
    ADC1->CR2 &= ~ADC_CR2_EXTEN;        // reset EXTEN, default
    if(edge == RISE) ADC1->CR2 |= ADC_CR2_EXTEN_0;              // trigger
detection rising edge
    else if(edge == FALL) ADC1->CR2 |= ADC_CR2_EXTEN_1;     // trigger detection
falling edge
    else if(edge == BOTH) ADC1->CR2 |= ADC_CR2_EXTEN_Msk;   // trigger detection
both edge
}

void ADC_conversion(int convMode){
```

```
    if(convMode == CONT){
        // Repetition: Continuous conversion
        ADC1->CR2 |= ADC_CR2_CONT;              // Enable Continuous conversion
mode
    }
    else if(convMode == SINGLE){
        // Repetition: Single conversion
        ADC1->CR2 &= ~ADC_CR2_CONT;             // Disable Continuous conversion
mode
    }
}

void ADC_sequence(PinName_t *seqCHn, int seqCHnums){

    // Disable ADC
    ADC1->CR2 &= ~ADC_CR2_ADON;

    // Initialize ADC channels
    int chN[seqCHnums];

    // Change to Multi-Channel mode(scan mode)
    if (seqCHnums > 1)
        ADC1->CR1 |= ADC_CR1_SCAN;                  // 1: (multi-channel)Scan
mode

    // ADC channels mapping
    for(int k = 0; k < seqCHnums; k++)
        ADC_pinmap(seqCHn[k], &(chN[k]));

    ADC1->SQR1 &= ~ ADC_SQR1_L;                     // reset length of
conversions in the regular channel
    ADC1->SQR1 |= (seqCHnums - 1) << ADC_SQR1_L_Pos; // conversions in the
regular channel conversion sequence

    for(int i = 0; i < seqCHnums; i++){
        if (i < 6){
            ADC1->SQR3 &= ~(0x1F << i*5);                // SQn clear bits
            ADC1->SQR3 |= chN[i] << i*5;                 // Choose the channel to
convert sequence
        }
        else if (i <12){
            ADC1->SQR2 &= ~(0x1F << (i-6)*5);        // SQn clear bits
            ADC1->SQR2 |= chN[i] << (i-6)*5;         // Choose the channel to
convert sequence
        }
        else{
            ADC1->SQR1 &= ~(0x1F << (i-12)*5);   // SQn clear bits
            ADC1->SQR1 |= chN[i] << (i-12)*5;        // Choose the channel to
convert sequence
        }
    }

    // Start ADC
    ADC_start();
}


void ADC_start(void){
```

```
    // Enable ADON, SW Trigger---------------------------------------------------
----------------------------
    ADC1->CR2 |= ADC_CR2_ADON;
    ADC1->CR2 |= ADC_CR2_SWSTART;
}

// ADC value read
uint32_t ADC_read(void){
    return ADC1->DR;
}

// ADC interrupt flag
uint32_t is_ADC_EOC(void){
    return (ADC1->SR & ADC_SR_EOC) == ADC_SR_EOC;
}

// ADC overflow flag
uint32_t is_ADC_OVR(void){
    return (ADC1->SR & ADC_SR_OVR) == ADC_SR_OVR;
}

// ADC clear flag
void clear_ADC_OVR(void){
    ADC1->SR &= ~ADC_SR_OVR;
}


/* ---------------------------------------------------------------------------------
-------*/
//                                  JADC Configuration
//
/* ---------------------------------------------------------------------------------
-------*/

void JADC_init(PinName_t pinName){
// 0. Match Port and Pin for JADC channel
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    int chN;
    ADC_pinmap(pinName, &chN);          // ADC Channel <->Port/Pin mapping

// GPIO configuration ----------------------------------------------------------
----------
// 1. Initialize GPIO port and pin as ANALOG, no pull up / pull down
    GPIO_init(port, pin, ANALOG);               // ANALOG = 3
    GPIO_pupd(port, pin, EC_NONE);              // EC_NONE = 0

// ADC configuration    --------------------------------------------------------
-------------
// 1. Total time of conversion setting
    // Enable ADC pheripheral clock
    RCC->APB2ENR  |= RCC_APB2ENR_ADC1EN;    // Enable the clock of
RCC_APB2ENR_ADC1EN

    // Configure ADC clock pre-scaler
    ADC->CCR &= ~ADC_CCR_ADCPRE;                     // 0000: PCLK2 divided by 2
(42MHz)
```

```
    // Configure ADC resolution
    ADC1->CR1 &= ~ADC_CR1_RES;                  // 00: 12-bit resolution
(15cycle+)

    // Configure channel sampling time of conversion.
    // Software is allowed to write these bits only when ADSTART=0 and JADSTART=0
   !!
    // ADC clock cycles @42MHz = 2us
    if(chN < 10) {
        ADC1->SMPR2  &= ~(7 << (3* chN));
        ADC1->SMPR2  |= 4U << (3* chN);
    }
    else{
        ADC1->SMPR1  &= ~(7 << (3* (chN - 10)));
        ADC1->SMPR1  |= 4U << (3* (chN - 10));
    }
// 2. Regular / Injection Group
    //Regular: SQRx, Injection: JSQx

// 3. Repetition: Single or Continuous conversion
    ADC1->CR2 |= ADC_CR2_CONT;                  // Enable Continuous conversion
mode

// 4. Single Channel or Scan mode
    //  - Single Channel: scan mode, right alignment
    ADC1->CR1 |= ADC_CR1_SCAN;                      // 1: Scan mode enable
    ADC1->CR2 &= ~ADC_CR2_ALIGN;                // 0: Right alignment

    // Configure the sequence length
    ADC1->JSQR &= ~ADC_JSQR_JL;                     // 0000: 1 conversion in the
regular channel conversion sequence

    // Configure the channel sequence
    ADC1->JSQR &= ~ADC_JSQR_JSQ4;                   // SQ1 clear bits
    ADC1->JSQR |= (chN & ADC_JSQR_JSQ4);    // Choose the channel to convert
firstly

// 5. Interrupt Enable
    // Enable JEOC(conversion) interrupt.
    ADC1->CR1 &= ~ADC_CR1_JEOCIE;               // JEOC interrupt reset
    ADC1->CR1 |= ADC_CR1_JEOCIE;                    // JEOC interrupt enable

    // Enable ADC_IRQn
    NVIC_SetPriority(ADC_IRQn,1);               //NVIC interrupt setting
    NVIC_EnableIRQ(ADC_IRQn);                   //Enable NVIC

/* -------------------------------------------------------------------------------
-------*/
//                  HW TRIGGER MODE
/* -------------------------------------------------------------------------------
-------*/

    // TRGO Initialize : TIM3, 1msec, RISE edge
    JADC_trigger(TIM5, 1, RISE);
}

void JADC_trigger(TIM_TypeDef* TIMx, int msec, int edge){
```

```
    // set timer
    int timer = 0;
    if(TIMx==TIM1) timer=1;
    else if(TIMx==TIM2) timer=2;
    else if(TIMx==TIM4) timer=4;
    else if(TIMx==TIM5) timer=5;

    // Single conversion mode (disable continuous conversion)
    ADC1->CR2 &= ~ADC_CR2_CONT;     // Discontinuous conversion mode
    ADC1->CR2 |= ADC_CR2_EOCS;          // Enable EOCS

    // Enable TIMx Clock as TRGO mode
// 1. TIMx Trigger Output Config
    // Enable TIMx Clock
    TIM_init(TIMx, msec);
    TIMx->CR1 &= ~1;                            //counter disable

    // Set PSC, ARR
  TIM_period_ms(TIMx, msec);

  // Master Mode Selection MMS[2:0]: Trigger output (TRGO)
  TIMx->CR2 &=~(7<<4);                    // reset MMS
  TIMx->CR2 |= 4<<4;                      //100: Compare - OC1REF signal is used
as trigger output (TRGO)

    // Output Compare Mode
  TIMx->CCMR1 &= ~(7<<4);       // OC1M : output compare 1 Mode
  TIMx->CCMR1 |= 6<<4;          // OC1M = 110 for compare 1 Mode ch1

  // OC1 signal
  TIMx->CCER |=1;                     // CC1E Capture enabled
    TIMx->CCR1  = (TIMx->ARR)/2;  //msec*10 - 1;        // CCR set

  // Enable TIMx
  TIMx->CR1 |= 1;                           //counter enable

// 2. ADC HW Trigger Config.
    // Select Trigger Source
    ADC1->CR2 &= ~ADC_CR2_JEXTSEL;  // reset EXTSEL
    if(TIMx==TIM1) ADC1->CR2 |= timer<<16;                          // TIMx
TRGO event (JADC : TIM1, TIM2, TIM4, TIM5 TRGO)
    else if(TIMx==TIM2) ADC1->CR2 |= (timer+1)<<16;             // TIMx TRGO
event (JADC : TIM1, TIM2, TIM4, TIM5 TRGO)
    else ADC1->CR2 |= (timer*2+1)<<16;                                  //
TIMx TRGO event (JADC : TIM1, TIM2, TIM4, TIM5 TRGO)

    ADC1->CR2 &= ~ADC_CR2_JEXTEN;
// reset JEXTEN, default
    if(edge==RISE) ADC1->CR2 |= ADC_CR2_JEXTEN_0;                   // trigger
detection rising edge
    else if(edge==FALL) ADC1->CR2 |= ADC_CR2_JEXTEN_1;      // trigger detection
falling edge
    else if(edge==BOTH) ADC1->CR2 |= ADC_CR2_JEXTEN_Msk;    // trigger detection
both edge

}

void JADC_sequence(PinName_t *seqCHn, int seqCHnums){
```

```
    // Disable ADC
    ADC1->CR2 &= ~ADC_CR2_ADON;

    int chN[seqCHnums];

    // Chnage to Multi-Channel mode(scan mode)
    if (seqCHnums>1)
        ADC1->CR1 |= ADC_CR1_SCAN;                  // 1: (multi-channel)Scan
mode

    for(int k=0; k<seqCHnums; k++)                  // ADC Channel <->Port/Pin
mapping
        ADC_pinmap(seqCHn[k], &(chN[k]));

    ADC1->JSQR &= ~(0xF<<20);                   // reset length of conversions
in the regular channel
    ADC1->JSQR |= (seqCHnums-1)<<20;            // conversions in the regular
channel conversion sequence

    for(int i = 0; i<seqCHnums; i++){
        ADC1->JSQR &= ~(0x1F<<(5*i+15-(seqCHnums-1)*5));    // SQ1 clear bits
        ADC1->JSQR |= chN[i]<<(5*i+15-(seqCHnums-1)*5); // Choose the channel to
convert sequence
    }

    // Start ADC
    JADC_start();
}


// void JADC_sequence(int length, int *seq){

//  ADC1->JSQR &= ~(0xF<<20);                   // reset length of conversions
in the regular channel
//  ADC1->JSQR |= (length-1)<<20;           // conversions in the regular
channel conversion sequence

//  for(int i = 0; i<length; i++){
//      ADC1->JSQR &= ~(0x1F<<(5*i+15-(length-1)*5));   // SQ1 clear bits
//      ADC1->JSQR |= seq[i]<<(5*i+15-(length-1)*5);    // Choose the channel to
convert sequence
//  }
// }

void JADC_start(void){
    // Enable ADON, JSW Trigger--------------------------------------------------
-----------------------------
    ADC1->CR2 |= ADC_CR2_ADON;
    ADC1->CR2 |= ADC_CR2_JSWSTART;
}

uint32_t JADC_read(int chN){
    uint32_t outData=0;

    switch(chN){
        case 1: outData = ADC1->JDR1;
        case 2: outData = ADC1->JDR2;
        case 3: outData = ADC1->JDR3;
```

```c
        case 4: outData = ADC1->JDR4;
    }

    return outData;
}


uint32_t is_JADC_EOC(void){
    return (ADC1->SR & ADC_SR_JEOC) == ADC_SR_JEOC;
}

uint32_t is_JADC_OVR(void){
    return (ADC1->SR & ADC_SR_OVR) == ADC_SR_OVR;
}

void clear_JADC_OVR(void){
    ADC1->SR &= ~ADC_SR_OVR;
}


///////////////////////////////////////////////////
void ADC_pinmap(PinName_t pinName, uint32_t *chN){
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);

    if(port == GPIOA){
        switch(pin){
            case 0 : *chN = 0; break;
            case 1 : *chN = 1; break;
            case 4 : *chN = 4; break;
            case 5 : *chN = 5; break;
            case 6 : *chN = 6; break;
            case 7 : *chN = 7; break;
            default: break;
        }
    }
    else if(port == GPIOB){
        switch(pin){
            case 0 : *chN = 8; break;
            case 1 : *chN = 9; break;
            default: break;
        }
    }
    else if(port == GPIOC){
        switch(pin){
            case 0 : *chN = 10; break;
            case 1 : *chN = 11; break;
            case 2 : *chN = 12; break;
            case 3 : *chN = 13; break;
            case 4 : *chN = 14; break;
            case 5 : *chN = 15; break;
            default: break;
        }
    }
}
```

ecUART.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
```

ecUART.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;

};

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));
```

```c
        // 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);
}
```

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
```

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
```

```c
        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, 3);
}



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;
    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;
        }
    }
}
```

ecSysTick.h

```
#ifndef __EC_SYSTICK_H
#define __EC_SYSTICK_H

#include "stm32f4xx.h"
#include "ecRCC.h"
#include <stdint.h>

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

extern volatile uint32_t msTicks;
void SysTick_init(void);
void SysTick_Handler(void);
void SysTick_counter();
void delay_ms(uint32_t msec);
void SysTick_reset(void);
uint32_t SysTick_val(void);
void SysTick_enable(void);
void SysTick_disable(void);

#ifdef __cplusplus
}
#endif /* __cplusplus */

#endif
```

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;
```

```c
//  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);
}
```

ecSTM32F411.h

```c
#include "ecEXTI.h"
#include "ecGPIO.h"
#include "ecPinNames.h"
#include "ecPWM.h"
#include "ecRCC.h"
#include "ecSysTick.h"
#include "ecTIM.h"
#include "ecPWM.h"
#include "ecPinNames.h"
#include "ecStepper.h"
#include "ecUART.h"
#include "ecADC.h"

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