EC API Documentation

Github

Embedded Controller - STM32F411 Driver Library

Written by: Gyeonheal An

Program: C/C++

IDE/Compiler: Keil uVision 5

OS: WIn10/11

MCU: STM32F411RE, Nucleo-64

I. GPIO.h

```
#include "GPIO.h"
```

i. GPIO_init()

Enables GPIO port and pins and initializes mode [this function contains function GPIO_mode()]

```
void GPIO_init(GPIO_TypeDef *Port, int pin, unsigned int mode);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use
- **mode:** Input(00), Output(01), Alternate Function(10), Analog(11)

Example code

```
void setup(){
    GPIO_init(GPIOA, 1, OUTPUT);
}
```

ii. GPIO_mode()

Initializes GPIO mode

```
void GPIO_mode(GPIO_TypeDef* Port, int pin, unsigned int mode);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use
- **mode:** Input(00), Output(01), Alternate Function(10), Analog(11)

Example code

```
void setup(){
    GPIO_mode(GPIOA, 1, OUTPUT);
}
```

iii. GPIO_ospeed()

Initializes GPIO speed

```
void GPIO_ospeed(GPIO_TypeDef *Port, int pin, unsigned int speed){
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

- **speed:** Low speed (00), Medium speed (01), Fast speed (10), High speed (11)

Example code

```
void setup(){
    GPIO_ospeed(GPIOC, 1, EC_LOW);
}
```

iv. GPIO_otype()

Initializes GPIO output type

```
void GPIO_otype(GPIO_TypeDef* Port, int pin, unsigned int type);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

- **type:** Output push-pull (0, reset), Output open drain (1)

Example code

```
void setup(){
    GPIO_otype(GPIOC, 1, EC_PUSH_PULL);
}
```

v. GPIO_pupd()

Initializes GPIO pin connection type (pull up: Vcc, pull down: GND)

```
void GPIO_pupd(GPIO_TypeDef *Port, int pin, unsigned int pupd);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

- **pupd:** No pull-up, pull-down (00), Pull-up (01), Pull-down (10), Reserved (11)

Example code

```
void setup(){
    GPIO_pupd(GPIOC, 1, EC_NONE);
}
```

vi. GPIO_write()

It changes Port->ODR resister to active fuction

```
void GPIO_write(GPIO_TypeDef *Port, int pin, unsigned int Output);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

- **Output:** 0 or 1 (Low, High)

Example code

```
GPIO_write(GPIOC, 1, 1);
```

vii. GPIO_read()

It reads Port->IDR resister value

```
int  GPIO_read(GPIO_TypeDef *Port, int pin);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

Example code

```
if(GPIO_read(GPIOC, BUTTON_PIN) == 0){
    count++;
}
```

II. EXTI.h

i. EXTI_init()

It initializes external interrupt function

```
void EXTI_init(GPIO_TypeDef *Port, int pin, int trig, int priority);
```

Parameters

- **Port:** GPIOA~GPIOD
- **pin:** select pin for use

- **trig:** FALL, RISE, BOTH
- **priority:** Determine the priority of tasks to be executed first. (0~15)

Example code

```
EXTI_init(GPIOC, BUTTON_PIN, FALL, 0);
```

ii. EXTI_enable()

It enables EXTI function

```
void EXTI_enable(uint32_t pin);
```

Parameters

- **pin:** select pin for use

Example code

```
EXTI_enable(BUTTON_PIN);
```

iii. EXTI_disable()

It disables EXTI function

```
void EXTI_disable(uint32_t pin);
```

Parameters

- **pin:** select pin for use

Example code

```
EXTI_disable(BUTTON_PIN);
```

iv. is_pending_EXTI()

Check whether it is on pending or not.

```
is_pending_EXTI(uint32_t pin)
```

Parameters

- **pin:** select pin for use

v. clear_pending_EXTI()

It clears pending of pin

```
void clear_pending_EXTI(uint32_t pin);
```

Parameters

- **pin:** select pin for use

Example code

```
void EXTI15_10_IRQHandler(void) {
    if (is_pending_EXTI(BUTTON_PIN) == 1) {
        while(1){
            delay++;
            if(delay > 3000000) break;
        }
        clear_pending_EXTI(BUTTON_PIN);
        delay = 0;
    }
}
```

III. TIM.h

```
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_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);
```

i. TIM_init()

```
void TIM_init(TIM_TypeDef *TIMx, uint32_t msec);
```

it initializes timer setting.

1. Enables Timer clock
2. Set count period
3. Set count direction Upcounter/Downcounter
4. Enables timer counter

Parameters

- **TIMx:** select timer for use
- **msec:** counting period

ii. TIM_period_us()

```
void TIM_period_us(TIM_TypeDef* TIMx, uint32_t usec);
```

It changes PSC, ARR value to use micro second unit.

Parameters

- **TIMx:** select timer for use
- **usec:** counting period

iii. TIM_period_ms()

```
void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec);
```

It changes PSC, ARR value to use millisecond unit.

Parameters

- **TIMx:** select timer for use
- **msec:** counting period

iv. TIM_UI_init()

```
void TIM_UI_init(TIM_TypeDef* TIMx, uint32_t msec);
```

It initializes and enables timer update interrupt.

1. Initializes Timer
2. Enables Update Interrupt
3. NVIC setting
4. Set priority

Parameters

- **TIMx:** select timer for use
- **usec:** counting period

v. TIM_UI_enable()

```
void TIM_UI_enable(TIM_TypeDef* TIMx);
```

It enables Timer Update Interrupt.

Parameters

- **TIMx:** select timer for use

vi. TIM_UI_disable()

```
void TIM_UI_disable(TIM_TypeDef* TIMx);
```

It disables Timer Update Interrupt.

Parameters

- **TIMx:** select timer for use

vii. is UIF() & clear_UIF()

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

It check whether timer update interrupt has occurred or not.

It clears pending of timer.

Parameters

- **TIMx:** select timer for use

Example code of TIM.h

```
void setup(void){
    RCC_PLL_init();                 // System Clock = 84MHz
    TIM_UI_init(TIM2, 1);               // TIM2 Update-Event Interrupt every 1 msec
}

void TIM2_IRQHandler(void){
    if(is_UIF(TIM2)){               // Check UIF(update interrupt flag)
        _count++;
        if (_count > 1000) {
            LED_toggle();           // LED toggle every 1 sec
            _count = 0;
        }
        clear_UIF(TIM2);            // Clear UI flag by writing 0
    }
}
```

IV. PWM.h

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

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

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

i. PWM_init()

```
void PWM_init(PinName_t pinName);
```

It initializes GPIO setup and Timer

1. Matches TIMx from port and pin
2. Initializes GPIO port and pin as Alternative Function

```
GPIO_init(port, pin, AF);
GPIO_ospeed(port, pin, EC_FAST);
GPIO_otype(port, pin, EC_PUSH_PULL);
GPIO_pupd(port, pin, EC_PU);
```

3. Configure GPIO AFR by pin num.
4. Initializes Timer
5. Set Timer Direction
6. Configure Timer Output mode as PWM
7. Enable Timer Counter and Timer

Parameters

- **pinName:** select pins for use

ii. PWM_pinmap()

```
void PWM_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN);
```

Find TIM and Channel through pin (STM32F411 board)

Parameters

- **pinName:** select pins for use
- **TIMx:** Timer we use
- **chN:** Channel we use

iii. PWM_period() & PWM_period_ms() & PWM_period_us()

```
void PWM_period(PinName_t pinName,  uint32_t msec);
void PWM_period_ms(PinName_t pinName,  uint32_t msec);
void PWM_period_us(PinName_t pinName, uint32_t usec);
```

It sets PWM period by milliseconds or micro seconds.

1. Match TIMx from Port and Pin
2. Set Counter Period in msec or usec

Parameters

- **pinName:** select pins for use
- **msec or usec:** counting period

iv. PWM_pulsewidth() & PWM_pulsewidth_ms() & PWM_pulsewidth_us()

```
void PWM_pulsewidth(PinName_t pinName, uint32_t pulse_width_ms);
void PWM_pulsewidth_ms(PinName_t pinName, uint32_t pulse_width_ms);
void PWM_pulsewidth_us(PinName_t pinName, uint32_t pulse_width_us);
```

It regulates pulsewidth by milliseconds and micro seconds

1. Match TIMx from port and pin
2. Declare system frequency and prescaler
3. Check system CLK: PLL or HSI

4. Configure PSC

Parameters

- **pinName:** select pins for use
- **pulse_width_ms, pulse_width_us:** set millisecond or microsecond unit

v. PWM_duty

```
void PWM_duty(PinName_t pinName, float duty);
```

Set the duty ratio and generate a PWM signal based on it.

Parameters

- **pinName:** select pins for use
- **PWM_duty:** 0~1 duty ration to generate

Example code of PWM.h

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

    // PWM of 20 msec:  TIM2_CH1 (PA_5 AFmode)
    GPIO_init(GPIOA, PWM_PIN, AF);
    PWM_init(PWM_PIN);
    PWM_period(PWM_PIN, 20);   // 20 msec PWM period
}

int main(void) {
    // Initialization --------------------------------------------------
    setup();

    // Infinite Loop ---------------------------------------------------
    while(1){
        for (int i=0; i<5; i++) {
            PWM_duty(PWM_PIN, (float)0.2*i);
            delay_ms(1000);
        }
    }
}
```

V. ICAP.h

```
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);
```

i. ICAP_pinmap()

```
void ICAP_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN);
```

It finds TIMx and channel by the pin

Parameters

- **pinName:** select pins for use
- **TIMx:** Timer we use
- **chN:** Channel we use

ii. ICAP_init()

```
void ICAP_init(PinName_t pinName);
```

It initializes Input Capture

1. Match Input Capture Port and Pin for TIMx

2. Initialize GPIO port and pin as AF

3. Configure GPIO AFR by Pin num.

4. Initializes Timer configuration
5. Initializes Input capture configuration
6. Activation Edge: CCyNP/CCyP
7. Enable CCy Capture, Capture/Compare interrupt
8. Enable Interrupt of CC(CCyIE), Update (UIE)
9. Enable Counter

Parameters

- **pinName:** select pins for use

iii. ICAP_setup()

```
void ICAP_setup(PinName_t pinName, int ICn, int edge_type);
```

It configures selecting TIx-ICy and Edge Type

1. Match Input Capture Port and Pin for TIMx
2. Disable CC. Disable CCInterrupt for ICn.
3. Configure IC number(user selected) with given IC pin(TIMx_CHn)
4. Configure Activation Edge direction
5. Enable CC. Enable CC Interrupt.

Parameters

- **pinName:** select pins for use
- **ICn:** select input capture channel
- **edge_type**: select type of edge (IC_RISE(0), IC_FALL(1), IC_BOTH(2))

iv. ICAP_counter_us

```
void ICAP_counter_us(PinName_t pinName, int usec);
```

It sets Time span for one counter step

1. Match Input Capture Port and Pin for TIMx
2. Configuration Timer Prescaler and ARR

Parameters

- **pinName:** select pins for use
- **usec:** select input capture channel

v. is_CCIF() & clear_CCIR()

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

It checks whether capture flag is on or not

It clears pending of input capture.

Parameters

- **TIMx:** Timer we use
- **ccNum:** channel number we use

vi. ICAP_capture()

```
uint32_t ICAP_capture(TIM_TypeDef* TIMx, uint32_t ICn);
```

it reads Input captre value

Parameters

- **TIMx:** Timer we use
- **ICn:** select input capture channel

Example code of ICAP.h

```
void setup(){
// 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
}

void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                     // Update interrupt
        ovf_cnt++;                                                    // 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) * 0.01;  //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                        // overflow reset
        clear_CCIF(TIM4,2);                               // clear
capture/compare interrupt flag
    }
}
```

VI. Stepper.h

```
void Stepper_init(GPIO_TypeDef* port1, int pin1, GPIO_TypeDef* port2, int pin2,
GPIO_TypeDef* port3, int pin3, GPIO_TypeDef* port4, int pin4);
void Stepper_setSpeed(long whatSpeed);
void Stepper_step(uint32_t steps, uint32_t direction, uint32_t mode);
void Stepper_stop(void);
```

i. Stepper_init()

```
void Stepper_init(GPIO_TypeDef* port1, int pin1, GPIO_TypeDef* port2, int pin2,
GPIO_TypeDef* port3, int pin3, GPIO_TypeDef* port4, int pin4);
```

It initializes GPIO port of stepper motor

Parameters

- **port1:** first GPIO port
- **pin1:** pin A
- **port2:** second GPIO port
- **pin2:** pin B
- **port3:** third GPIO port
- **pin3:** pin A not
- **port4:** fourth GPIO port
- **pin4:** pin B not

ii. Stepper_setSpeed()

```
void Stepper_setSpeed(long whatSpeed);
```

It sets step delay. Convert rpm to [msec] delay

```
step_delay =  1000 * 60 / whatSpeed  / step_per_rev;
```

Parameters

- **whatspeed:** rpm [rev/min]

iii. Stepper_step()

```
void Stepper_step(uint32_t steps, uint32_t direction, uint32_t mode);
```

It sets output step, direction and mode

Parameters

- **steps:** select steps for active
- **direction:** stepper motor direction
- **mode:** select accuracy of stepper motor (HALF , FULL)

iv. Stepper_stop()

```
void Stepper_stop(void);
```

It stops stepper motor

Example code of Stepper.h

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

    Stepper_init(GPIOB, A, GPIOB, B, GPIOB, NA, GPIOB, NB);
    Stepper_setSpeed(RPM);
}

int main(){
    setup();
    Stepper_step(2048*1, 1, HALF);
    while(1){
    }
}
```

VII. ADC.h

i. ADC_init()

```
void ADC_init(PinName_t pinName);
```

It initializes Analog Digital Converter

1. Match Port and Pin for ADC channel

2. Initializes GPIO pin (ANALOG, EC_NONE)
3. Enables ADC pheripheral clock, pre-caler, ADC resolution
4. Configures channel smapling time of conversion
5. Set repetition (single/continuous)
6. Interrupt Enable

Parameters

- **pinName:** select pins for use

ii. ADC_sequence()

```
void ADC_sequence(PinName_t *seqCHn, int seqCHnums);
```

It sets ADC channel sequence.

1. Disable ADC
2. Initialize ADC channels
3. Change to Multi-Channel mode(scan mode)
4. ADC channels mapping
5. Start ADC

Parameters

- **seqCHn:** select channel for use
- **seqCHnums:** number of sequence channel

iii. ADC_start()

```
void ADC_start(void);
```

It starts ADC

iv. is_ADC_EOC()

```
uint32_t is_ADC_EOC(void);
```

This is ADC interrupt flag

v. is_ADC_OVR() & clear_ADC_OVR()

```
uint32_t is_ADC_OVR(void);
void clear_ADC_OVR(void);
```

It checks whether ADC overflow flag on or not and clears pending

vi. ADC_read()

```
uint32_t ADC_read(void);
```

It reads ADC value

vii. ADC_trigger()

```
void ADC_trigger(TIM_TypeDef* TIMx, int msec, int edge);
```

It sets trigger conditions

1. set timer
2. Enable TIMx Clock as TRGO mode
3. ADC HW Trigger Config.

4. Select Trigger Polarity

Parameters

- **TIMx:** Timer we use
- **msec:** counting period
- **edge:** RISE, FALL, BOTH

viii. ADC_pinmap()

```
void ADC_pinmap(PinName_t pinName, uint32_t *chN);
```

Find channel number by the pin

Parameters

- **pinName:** select pins for use
- **chN:** channel number we found

Example code of ADC.h

```
void setup(void)
{
    RCC_PLL_init();                         // System Clock = 84MHz
    UART2_init();                     // UART2 Init
    SysTick_init();                     // SysTick Init

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

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

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
    }
}
```

VIII. UART.h

```
// 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);
```

i. UARTx_init()

```
void UART1_init(void);
void UART2_init(void);
```

It contains USART_setting() function to initialize UARTx setting

ii. UARTx_baud()

```
void UART1_baud(uint32_t baud);
void UART2_baud(uint32_t baud);
```

It setup baud rate of UARTx

Parameters

- **baud:** 9600~

iii. UARTx_read()

```
uint8_t USART1_read(void);
uint8_t USART2_read(void);
```

It reads UARTx value

iv. UARTx_write()

```
void USART1_write(uint8_t* buffer, uint32_t nBytes);
void USART2_write(uint8_t* buffer, uint32_t nBytes);
```

It displays UARTx value on PC screen

Parameters

- **buffer:** data what we are going to write
- **nBytes**: size of buffer

v. is_USARTx_RXNE()

```
uint32_t is_USART1_RXNE(void);
uint32_t is_USART2_RXNE(void);
```

It checks if USARTx value has income or not.

vi. USART_setting()

```
void USART_setting(USART_TypeDef* USARTx, GPIO_TypeDef* GPIO_TX, int pinTX,
GPIO_TypeDef* GPIO_RX, int pinRX, uint32_t baud);
```

It initializes USART setting

1. GPIO Pin for TX and RX
2. USARTx (x=2,1,6) configuration
    - Enable USART peripheral clock
    - Disable USARTx.
    - No Parity / 8-bit word length / Oversampling x16
    - Configure Stop bit
    - Enable TX, RX, and USARTx
3. Read USARTx Data (Interrupt)
    - Set the priority and enable interrupt

vii. USART_delay()

```
void USART_delay(uint32_t us);
```

It makes delay of USART

Parameters

- **us:** time = 100*us/7

Example code of USART.h

```
void setup(void){
    RCC_PLL_init();

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

    // BT serial init
    UART1_init();
    UART1_baud(BAUD_38400);

    //LED setup
    GPIO_init(GPIOA, LED_PIN, OUTPUT);    // calls RCC_GPIOA_enable()
}
```

```
void USART1_IRQHandler(){               // USART2 RX Interrupt : Recommended
    if(is_USART1_RXNE()){
            PC_Data = USART1_read();        // RX from UART1 (BT)
            if(PC_Data == 'L'){
                GPIO_write(GPIOA, LED_PIN, 0);
            }
            else if(PC_Data == 'H'){
                GPIO_write(GPIOA, LED_PIN, 1);
            }
      printf("MCU_1 received : %c \r\n",PC_Data); // TX to USART2(PC)
    }
}

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

IX. RCC.h

```
void RCC_HSI_init(void);
void RCC_PLL_init(void);
void RCC_GPIOA_enable(void);
void RCC_GPIOB_enable(void);
void RCC_GPIOC_enable(void);
void RCC_GPIOD_enable(void);
```

i. RCC_HSI_init()

```
void RCC_HSI_init(void);
```

It sets clock time as HSI unit (16MHz)

ii. RCC_PLL_init()

```
void RCC_PLL_init(void);
```

It sets clock time as PLL unit (84MHz)

iii. RCC_GPIOx_enable()

```
void RCC_GPIOA_enable(void);
void RCC_GPIOB_enable(void);
void RCC_GPIOC_enable(void);
void RCC_GPIOD_enable(void);
```

It is RCC Peripheral Clock Enable Register

It's about GPIOA to GPIOD

X. ecSTM32F411.h

It contains all of header files of embedded controller

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

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