Embedded Controller

Date: 2022-12-23

Author: SeungEung Hwang

Github: [repository link]

Program: C/C++

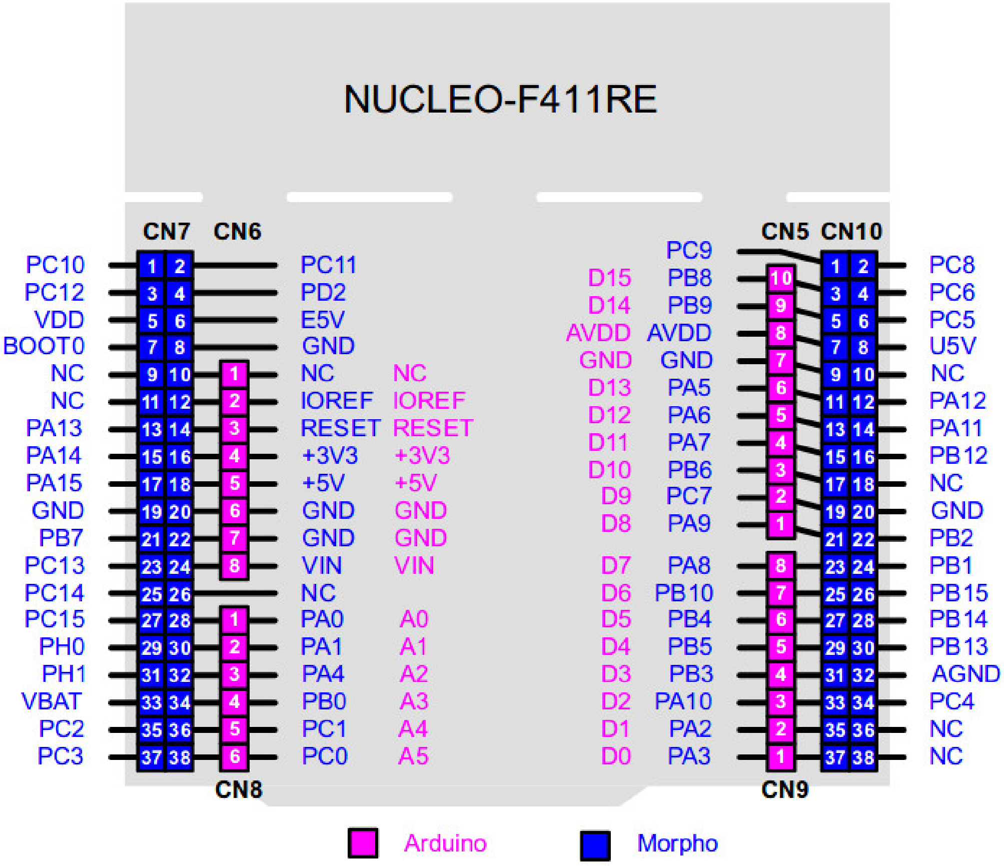
IDE/Compiler: Keil uVision 5

OS: WIn11

MCU: STM32F411RE (Nucleo-64)

Contents

STM 32 Circuit



Bitwise

a: array of register, **k**: bit number

- **write a (high) bit:** a|= (1 << k)
- **write two bits:** a |= (3 << k)
- **read one bit:** val = (a >> k) & 1
- **read 2 bits**: val = (a >> k) & 3   [0011]
- **clear bit:** a &= ~(1 << k)
- **Toggle:** a^= 1 << k (^= is XOR)
- **shift:** n time shift to right is divide by 2^n ex) pin >> 3 == pin / 8

RCC

RCC Register Table

RCC_CR

6.3.1 RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX81 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLI2S RDY	PLLI2S ON	PLLRDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
				r	rw	r	rw					rw	rw	r	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

RCC_PLLCFGR

6.3.2 RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

Access: no wait state, word, half-word and byte access.

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLLN / PLLM)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLLP$
- $f_{(USB\ OTG\ FS,\ SDIO)} = f_{(VCO\ clock)} / PLLQ$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserved	PLLSRC	Reserved				PLLP1	PLLP0
				rw	rw	rw	rw		rw					rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved	PLLN									PLLM5	PLLM4	PLLM3	PLLM2	PLLM1	PLLM0
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

RCC_CFGR

6.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: $0 \leq$ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
rw		rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PPRE2[2:0]			PPRE1[2:0]			Reserved		HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

RCC_AHB1ENR

6.3.9 RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									DMA2EN	DMA1EN	Reserved				
									rw	rw					

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved			CRCEN	Reserved				GPIOH EN	Reserved		GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
			rw					rw			rw	rw	rw	rw	rw

RCC_APB1ENR

6.3.11 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			PWR EN	Reserved				I2C3 EN	I2C2 EN	I2C1 EN	Reserved			USART2 EN	Reser-ved
			rw					rw	rw	rw				rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved							TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw								rw	rw	rw	rw

RCC_APB2ENR

6.3.12 RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved											SPI5EN	Reser-ved	TIM11 EN	TIM10 EN	TIM9 EN
											rw		rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reser-ved	SYSCF G EN	SPI4EN	SPI1 EN	SDIO EN	Reserved		ADC1 EN	Reserved		USART6 EN	USART1 EN	Reserved			TIM1 EN
	rw	rw	rw	rw			rw			rw	rw				rw

Internal Clock (HSI) for GPIO

1. **Enable HSI and choose as SYSCLK source**
   - Enable HSI: **(RCC->CR: HSION=1)**
   - Wait until HSI is stable and ready: **(RCC->CR: HSIRDY? 1)**
   - Choose the system clock switch : **(RCC->CFGR: SW = 00)**
   - Check if the selected source is correct: **(RCC->CFGR: SWS ? 00)**
2. Configure HSI (optional)
   - Calibration for RC oscillator: (RCC->CR: HSICAL, HSITRIM)
3. Configure APB/AHB Prescaler (optional)
   - Change Prescaler: RCC->CFGR: HPRE, PPRE
4. **Enable GPIOx clock(AHB1ENR)**
   - Enable (RCC_AHB1ENR) for PORTx

```
void RCC_HSI_init() {
    // Enable High Speed Internal Clock (HSI = 16 MHz)
    //RCC->CR |= ((uint32_t)RCC_CR_HSION);
    //  RCC->CR |= RCC_CR_HSION;
    RCC->CR |= 0x00000001U;

    // wait until HSI is ready
    //while ( (RCC->CR & (uint32_t) RCC_CR_HSIRDY) == 0 ) {;}
    while ( (RCC->CR & 0x00000002U) == 0 ) {;}

    // Select HSI as system clock source
    RCC->CFGR &= (uint32_t)(~RCC_CFGR_SW);          // not essential
    RCC->CFGR |= (uint32_t)RCC_CFGR_SW_HSI;         //00: HSI16 oscillator used
as system clock

    // Wait till HSI is used as system clock source
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != 0 );

        EC_SYSCLK=16000000;
}
```

External Clock (HSE) for GPIO

1. **Enable HSI and choose as SYSCLK source**
   - Enable HSE: **(RCC->CR: HSEON=1)**
   - Wait until HSE is stable and ready: **(RCC->CR: HSERDY? 1)**
   - Choose the system clock switch : **(RCC->CFGR: SW = 01)**
   - Check if the selected source is correct: **(RCC->CFGR: SWS ? 01)**
2. **Enable GPIOx clock(AHB1ENR)**
   - Enable (RCC_AHB1ENR) for PORTx

```
void RCC_HSE_init(){

    // HSE on
    RCC->CR |= RCC_CR_HSEON;
```

```
    // wait until HSE is ready
    while ( (RCC->CR & RCC_CR_HSERDY) == 0 ) {;}

    // Select HSI as system clock source
  RCC->CFGR &= (uint32_t)(~RCC_CFGR_SW);                                // not
essential
  RCC->CFGR |= (uint32_t)RCC_CFGR_SW_HSE;

        // Wait till HSE is used as system clock source
  while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS) != 1 );

    EC_SYSCLK=8000000;

}
```

PLL(HSI) for GPIO

1. **Enable either (HSI, or HSE) for PLL and Choose PLL for System Clock**
    - Enable HSI: **(RCC->CR : HSION=1)**
    - Wait until HSI is stable: **(RCC->CR : HSIRDY? 1)**
    - Choose PLL for system clock switch : **(RCC->CFGR : SW = 10)**
    - Check if PLL selection is correct: **(RCC->CFGR : SWS ? 10)**
2. **Select the clock source for PLL**
    - Select the PLL source(HSI or HSE): **(RCC->PLLCFGR : PLLSRC= 0 or 1)**
3. (Optional)Configure PLL parameters
    - Select (M/N/P): **(RCC->PLLCFGR : PLLM, PLLN, ...)** make 84MHz from 16MHz
4. **Enable PLL**
    - Enable main PLL: **(RCC->CR : PLLON=1), (RCC->CR : PLLRDY?0)**
5. (Optional)Configure APB/AHB Prescaler
    - Change Prescaler: (RCC->CFGR : HPRE, PPRE)
6. **Enable GPIOx clock(AHB1ENR)**
    - Enable (RCC_AHB1ENR) for PORTx

```
void RCC_PLL_init() {
    // To correctly read data from FLASH memory, the number of wait states
(LATENCY)
  // must be correctly programmed according to the frequency of the CPU clock
  // (HCLK) and the supply voltage of the device.
    FLASH->ACR &= ~FLASH_ACR_LATENCY;
    FLASH->ACR |=  FLASH_ACR_LATENCY_2WS;

    // Enable the Internal High Speed oscillator (HSI)
    RCC->CR |= RCC_CR_HSION;
    while((RCC->CR & RCC_CR_HSIRDY) == 0);

    // Disable PLL for configuration
    RCC->CR    &= ~RCC_CR_PLLON;

    // Select clock source to PLL
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;        // Set source for PLL: clear
bits
```

```
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSI; // Set source for PLL: 0 =HSI, 1 = HSE

    // Make PLL as 84 MHz
    // f(VCO clock) = f(PLL clock input) * (PLLN / PLLM) = 16MHz * 84/8 = 168 MHz
    // f(PLL_R) = f(VCO clock) / PLLP = 168MHz/2 = 84MHz
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 84U << 6;
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 8U ;
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLP;  // 00: PLLP = 2, 01: PLLP = 4, 10: PLLP = 6, 11: PLLP = 8

    // Enable PLL after configuration
    RCC->CR   |= RCC_CR_PLLON;
    while((RCC->CR & RCC_CR_PLLRDY)>>25 != 0);

    // Select PLL as system clock
    RCC->CFGR &= ~RCC_CFGR_SW;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    // Wait until System Clock has been selected
    while ((RCC->CFGR & RCC_CFGR_SWS) != 8UL);

    // The maximum frequency of the AHB and APB2 is 100MHz,
    // The maximum frequency of the APB1 is 50 MHz.
    RCC->CFGR &= ~RCC_CFGR_HPRE;        // AHB prescaler = 1; SYSCLK not divided (84MHz)
    RCC->CFGR &= ~RCC_CFGR_PPRE1;
    RCC->CFGR |=  RCC_CFGR_PPRE1_2;     // APB high-speed prescaler (APB1) = 2, HCLK divided by 2 (42MHz)
    RCC->CFGR &= ~RCC_CFGR_PPRE2;       // APB high-speed prescaler (APB2) = 1, HCLK not divided    (84MHz)

    EC_SYSCLK=84000000;
}
```

PLL(HSI) for GPIO

1. **Enable either (HSI, or HSE) for PLL and Choose PLL for System Clock**
   - Enable HSE: **(RCC->CR : HSEON=1)**
   - Wait until HSE is stable: **(RCC->CR : HSERDY? 1)**
   - Choose PLL for system clock switch : **(RCC->CFGR : SW = 10)**
   - Check if PLL selection is correct: **(RCC->CFGR : SWS ? 10)**
2. **Select the clock source for PLL**
   - Select the PLL source(HSE): **(RCC->PLLCFGR : PLLSRC= 0 or 1)**
3. (Optional)Configure PLL parameters
   - Select (M/N/P): **(RCC->PLLCFGR : PLLM, PLLN, ...)** make 84MHz from 8MHz
4. **Enable PLL**
   - Enable main PLL: **(RCC->CR : PLLON=1), (RCC->CR : PLLRDY?0)**
5. (Optional)Configure APB/AHB Prescaler
   - Change Prescaler: (RCC->CFGR : HPRE, PPRE)
6. **Enable GPIOx clock(AHB1ENR)**

- Enable (RCC_AHB1ENR) for PORTx

```
void RCC_PLL_HSE_init() {
    // To correctly read data from FLASH memory, the number of wait states
(LATENCY)
  // must be correctly programmed according to the frequency of the CPU clock
  // (HCLK) and the supply voltage of the device.
    FLASH->ACR &= ~FLASH_ACR_LATENCY;
    FLASH->ACR |=  FLASH_ACR_LATENCY_2WS;

    // Enable the Internal High Speed oscillator (HSI)
    RCC->CR |= RCC_CR_HSEON;
    while((RCC->CR & RCC_CR_HSERDY) == 0);

    // Disable PLL for configuration
    RCC->CR    &= ~RCC_CR_PLLON;

    // Select clock source to PLL
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLSRC;        // Set source for PLL: clear
bits
    RCC->PLLCFGR |= RCC_PLLCFGR_PLLSRC_HSE; // Set source for PLL: 0 =HSI, 1 =
HSE

    // Make PLL as 84 MHz
    // f(VCO clock) = f(PLL clock input) * (PLLN / PLLM) = 8MHz * 84/4 = 168 MHz
    // f(PLL_R) = f(VCO clock) / PLLP = 168MHz/2 = 84MHz
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLN) | 84U << 6;
    RCC->PLLCFGR = (RCC->PLLCFGR & ~RCC_PLLCFGR_PLLM) | 4U ;
    RCC->PLLCFGR &= ~RCC_PLLCFGR_PLLP;  // 00: PLLP = 2, 01: PLLP = 4, 10: PLLP
= 6, 11: PLLP = 8


    // Enable PLL after configuration
    RCC->CR   |= RCC_CR_PLLON;
    while((RCC->CR & RCC_CR_PLLRDY)>>25 != 0);

    // Select PLL as system clock
    RCC->CFGR &= ~RCC_CFGR_SW;
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    // Wait until System Clock has been selected
    while ((RCC->CFGR & RCC_CFGR_SWS) != 8UL);

    // The maximum frequency of the AHB and APB2 is 100MHz,
    // The maximum frequency of the APB1 is 50 MHz.
    RCC->CFGR &= ~RCC_CFGR_HPRE;        // AHB prescaler = 1; SYSCLK not divided
(84MHz)
    RCC->CFGR &= ~RCC_CFGR_PPRE1;
    RCC->CFGR |=  RCC_CFGR_PPRE1_2;     // APB high-speed prescaler (APB1) = 2,
HCLK divided by 2 (42MHz)
    RCC->CFGR &= ~RCC_CFGR_PPRE2;       // APB high-speed prescaler (APB2) = 1,
HCLK not divided    (84MHz)

    EC_SYSCLK=84000000;
}
```

GPIO digital I/O

GPIOx Register Table

GPIOx_MODER

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..E and H)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OTYPER

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..E and H)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_OSPEEDR

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..E and H)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_PUPDR

8.4.4 GPIO port pull-up/pull-down register (GPIOx_PUPDR) (x = A..E and H)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]		PUPDR14[1:0]		PUPDR13[1:0]		PUPDR12[1:0]		PUPDR11[1:0]		PUPDR10[1:0]		PUPDR9[1:0]		PUPDR8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]		PUPDR6[1:0]		PUPDR5[1:0]		PUPDR4[1:0]		PUPDR3[1:0]		PUPDR2[1:0]		PUPDR1[1:0]		PUPDR0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_IDR

8.4.5 GPIO port input data register (GPIOx_IDR) (x = A..E and H)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

GPIOx_ODR

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..E and H)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIOx_AFRL and GPIOx_AFRH

using at PWM

8.4.9 GPIO alternate function low register (GPIOx_AFRL) (x = A..E and H)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

8.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..E and H)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Process of GPIOx register initialization

- **output setting**
  - Enable Peripheral Clock (**AHB1ENR**)
  - Configure as Digital Output (**MODER**)
  - Configure pull-up/down resistors (**PUPDR**)
  - For Output: Configure Output Type (**OTYPE**)
  - For Output: Configure Output Speed (**OSPEEDR**)
- input setting
  - Enable Peripheral Clock (**AHB1ENR**)
  - Configure as Digital Output (**MODER**)
  - Configure pull-up/down resistors (**PUPDR**)
  - Input Data **(IDR)**

ecGPIO.c

void GPIO_init(GPIO_TypeDef *Port, int pin, int mode)

This function is initialize of GPIO. It has three parameters, GPIO Port, pin number and mode. After choose the port, it goes enable of each port. End of the **GPIO_init** is **GPIO_mode**.

```
void GPIO_init(GPIO_TypeDef *Port, int pin, int mode){
    if (Port == GPIOA)
        RCC_GPIOA_enable();

    if (Port == GPIOB)
        RCC_GPIOB_enable();

    if (Port == GPIOC)
        RCC_GPIOC_enable();
```

```
    if (Port == GPIOD)
        RCC_GPIOA_enable();

    if (Port == GPIOE)
        RCC_GPIOB_enable();

    GPIO_mode(Port, pin, mode);
}
```

void GPIO_mode(GPIO_TypeDef *Port, int pin, int mode)

GPIO has 4 state of mode. **Port -> MODER** consist 2-bits.

Input and Output are Digital signal, AlterFunc use at PWM signal and Analog

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

void GPIO_ospeed(GPIO_TypeDef *Port, int pin, int speed)

We can select the 4 types of speed at **GPIO Speed** . **GPIO_ospeed** consist 2-bits.

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

void GPIO_otype(GPIO_TypeDef *Port, int pin, int type)

Output push-pull is using internal power source and Output open drain is using external power source.

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

void GPIO_pupd(GPIO_TypeDef *Port, int pin, int pupd)

Pull up - pull down is prevent the "**floating**"

pull up circuit은 스위치를 누르지 않으면 5V로 인식 **HIGH**, 누르면 Input에서 **LOW**로 인식한다.

pull down circuit은 스위치를 누르지 않으면 GND인식 **LOW**, 누르면 **HIGH**로 인식

```
// 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, int pupd){
        Port->PUPDR  &=     ~(3UL << (pin * 2));
        Port->PUPDR  |=      (pupd   << (pin * 2));
}
```

int GPIO_read(GPIO_TypeDef *Port, int pin)

```
int GPIO_read(GPIO_TypeDef *Port, int pin){
    // 0 or 1만 읽기 위해서 사용하는 방법
    return ((Port -> IDR) >> pin) & 1UL;
}
```

void GPIO_write(GPIO_TypeDef *Port, int pin, int output)

```
void GPIO_write(GPIO_TypeDef *Port, int pin,  int output){
         if(output == 1)
                Port->ODR |= (1UL << pin);
            else
                Port->ODR &= ~(1UL << pin);
}
```

void GPIO_in_set(GPIO_TypeDef *Port, int pin, int pupd)

GPIO input setting

set port, pin number, pull up pull down

```
void GPIO_in_set(GPIO_TypeDef *Port, int pin, int pupd){
    GPIO_init(Port, pin, INPUT);
    GPIO_pupd(Port, pin, pupd);
}
```

void GPIO_out_set(GPIO_TypeDef *Port, int pin, int pupd, int speed, int type)

GPIO output setting

set port, pin number, pull up pull down, speed, pushpull

```
void GPIO_out_set(GPIO_TypeDef *Port, int pin, int pupd, int speed, int type){
    GPIO_init       (Port, pin, OUTPUT);
    GPIO_otype      (Port, pin, type);
    GPIO_pupd       (Port, pin, pupd);
    GPIO_ospeed     (Port, pin, speed);
    GPIO_write      (Port, pin, LOW);
}
```

External Interrupt

EXTI Register Table

EXTI_IMR

10.3.1 Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	Reserved		MR18	MR17	MR16
									rw	rw			rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

EXTI_EMR

10.3.2 Event mask register (EXTI_EMR)

Address offset: 0x04
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									MR22	MR21	Reserved		MR18	MR17	MR16
									rw	rw			rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

EXTI_RTSR

10.3.3 Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	Reserved		TR18	TR17	TR16
									rw	rw			rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

EXTI_FTSR

10.3.4 Falling trigger selection register (EXTI_FTSR)

Address offset: 0x0C
Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									TR22	TR21	Reserved		TR18	TR17	TR16
									rw	rw			rw	rw	rw
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

EXTI_PR

10.3.6 Pending register (EXTI_PR)

Address offset: 0x14
Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved									PR22	PR21	Reserved		PR18	PR17	PR16
									rc_w1	rc_w1			rc_w1	rc_w1	rc_w1
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

ecEXTI.c

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

1. Go to SYSCFG peripheral and clock enable
2. Select the port
3. Select the pin at **EXTI_CR**
4. Choose the Falling or Rising at **FTSR** or **RTSR**
5. Interrupt enable **IMR**
6. Array address allocation at EXTI_IRQn 0~4, 5~9 and 10 ~ 15
7. Priority of pending allocation
8. EXTI enable

```
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
    unsigned int EXTICR_port;
    // 미리 지정되어 있는 포트의 일반화 번호들이다
    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[BUTTON_PIN/4] &= ~SYSCFG_EXTICR4_EXTI13; //~15<<(4*
(pin%4))*/
    SYSCFG->EXTICR[Pin/4] &= ~( 15UL << (4*(Pin%4)) ) ;         // clear 4 bits
    SYSCFG->EXTICR[Pin/4] |= (EXTICR_port << (4*(Pin%4)) ) ;    // set 4 bits  무
슨 포트인지 일반화

    // 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  |= (1UL << Pin);     // not masked, 왜 not masked 였지?

    // NVIC(IRQ) Setting
    uint8_t EXTI_IRQn = 0;

    if (Pin < 5)     EXTI_IRQn = Pin + 6;                     // EXTI0이 6번 핀을 가
지고 있다. EXTI4 는 10번 핀에 할당되어 있다.
    else if (Pin < 10)  EXTI_IRQn = EXTI9_5_IRQn; // 5~9번핀일 때  EXTI9_5_IRQn ->
23번핀에 할당
    else            EXTI_IRQn = EXTI15_10_IRQn;               // 10 ~ 15번 핀
40번 할당

    NVIC_SetPriority(EXTI_IRQn, priority);  // NVIC sets the order of execution
according to prior
    NVIC_EnableIRQ(EXTI_IRQn);  // EXTI IRQ enable
}
```

void EXTI_enable(uint32_t pin)

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

void EXTI_disable(uint32_t pin)

```
void EXTI_disable(uint32_t pin) {
    EXTI->IMR &= ~(1UL << pin);     // masked (i.e., Interrupt disabled)
}
```

uint32_t is_pending_EXTI(uint32_t pin)

This function is return 1 or 0. EXTI로 지정한 pin 에서 signal이 들어오면 pending structure에서 비교를 하여 if 문 안의 조건을 실행할지 말지 결정한다

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

void clear_pending_EXTI(uint32_t pin)

You have to use this code after the Interrupt was began. If it does not use will gives huge error.

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

System Ticker

ecSysTick.c

systick is also interrupt, but it is down count system. And Systick's pending order is subordinate.

```
void SysTick_init(uint32_t msec){

    //  SysTick Control and Status Register
    // Disable SysTick IRQ and SysTick Counter
    SysTick_disable();

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

    // SysTick Reload Value Register
    SysTick->LOAD = (MCU_CLK_PLL / (1000)) * msec - 1;                  //
1ms, for HSI PLL = 84MHz.

    // SysTick Current Value Register
    SysTick_reset();

    // Enables SysTick exception request " 이거 이해가 잘 안간다"
    // 0 = Counting down to zero does not assert the 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_enable();

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

Down count delay set

```
void SysTick_Handler(void){
    SysTick_counter();
}

void SysTick_counter(void){
    TimeDelay--;
}

void Delay (uint32_t nTime){

    TimeDelay = nTime; //setup
    while(TimeDelay != 0);
}
```

void SysTick_reset(void)

set the **VAL** value goes to 0

```
    // SysTick -> VAL에서 VAL이 0이 되면 feedback loop에서 reload 값으로 다시 돌아간다
    // SysTick을 초기화시키는 방식 -> 다운 클락이기 때문에 가능한 방법이고
void SysTick_reset(void)  {
    SysTick->VAL = 0;           // if VAL is 0, VAL will update Reroad value
}
```

uint32_t SysTick_val(void)

check the current **VAL** value

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

void SysTick_enable(void)

```
void SysTick_enable(void){
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
}
```

void SysTick_disable(void)

```
void SysTick_disable(void){
    SysTick->CTRL = 0;
}
```

TIMER

ecTIM.c

void TIM_init(TIM_TypeDef* TIMx, uint32_t msec)

타이머를 초기 설정해주는 함수

- **TIMx:** Choose Timer number 1~4
- **sec**: Timer period in msec

```
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;
    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 ==TIM10) RCC->APB2ENR |= RCC_APB2ENR_TIM10EN;
    else if (TIMx ==TIM11) RCC->APB2ENR |= RCC_APB2ENR_TIM11EN;

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

// 3. CNT Direction
    // Uppercounter
    TIMx->CR1 &= ~(1 << 4);      // TIM_CR1_DIR_Msk
    // down counter
//  TIMx->CR1 |= (1 << 4);

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

void TIM_period_us(TIM_TypeDef *TIMx, uint32_t usec)

타이머 길이 설정

- **TIMx:** Choose Timer number 1~4
- **sec**: Timer period in usec

```
void TIM_period_us(TIM_TypeDef *TIMx, uint32_t usec){
    // Period usec = 1 to 1000
    // 1us(1MHz, ARR = 1) to 65msec (ARR = 0xFFFF)
    // 1e-6
    if(TIMx == TIM2 || TIMx == TIM5){

        uint16_t PSCval = 84;                    // 84/84000000       = 1e-6  1MHz
        uint32_t ARRval = usec;              // 1/1000000          = 1e-6 = 1us
= 1MHz
        TIMx->PSC = PSCval - 1;
```

```
        TIMx->ARR =(ARRval - 1);
    }else{

        uint16_t PSCval = 84;    // 84 000 00
        uint16_t ARRval = (uint16_t)usec;  // 84/100000

        TIMx->PSC = PSCval - 1;
        TIMx->ARR =(ARRval - 1);
    }
}
```

void TIM_period_ms(TIM_TypeDef* TIMx, uint32_t msec)

- **TIMx:** Choose Timer number 1~4
- **sec**: Timer period in msec

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

    // 0.1ms(10kHz, ARR = 1) to 6.5sec (ARR = 0xFFFF)

    if(TIMx == TIM2 || TIMx == TIM5){

        uint16_t PSCval = 840;          // 840/84000000         = 1e-5  100KHz
        uint32_t ARRval = 100;          // 100/100000       = 1e-3 = 1ms  1KHz

        TIMx->PSC = PSCval - 1;
        TIMx->ARR = (ARRval* msec - 1) ;
    }else {

        uint16_t PSCval = 840;          // 840/84000000         = 1e-5  100KHz
        uint16_t ARRval = 100;          // 100/100000       = 1e-3 = 1ms  1KHz

        TIMx->PSC = PSCval - 1;
        TIMx->ARR = (ARRval* msec - 1) ;
    }
    // 기본 설정 1ms에 msec를 곱해서 한주기에 걸리는 시간을 지정해줌
}
```

void TIM_INT_init(TIM_TypeDef* TIMx, uint32_t msec)

// Update Event Interrupt

- **TIMx:** Choose interrupt Timer number 1~4
- **sec**: Timer period in msec

```
void TIM_INT_init(TIM_TypeDef* TIMx, uint32_t msec){
// 1. Initialize Timer
    TIM_init(TIMx,msec);

// 2. Enable Update Interrupt
    TIM_INT_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;
    else if(TIMx == TIM3)   IRQn_reg = TIM3_IRQn;
    else if (TIMx ==TIM4)       IRQn_reg = TIM4_IRQn;
    else if (TIMx ==TIM5)   IRQn_reg = TIM5_IRQn;
    // 9 10 11을 위한 IRQ가 따로 없어서 일단 TIM1과 연결된 값을 불러왔다.
    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;
    // repeat for TIM3, TIM4, TIM5, TIM9, TIM10, TIM11

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

Interrupt 실행을 위한 함수들

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

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

// update interrupt flag
// pending
uint32_t is_UIF(TIM_TypeDef *TIMx){
    return ((TIMx->SR & TIM_SR_UIF) == TIM_SR_UIF);
}

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

}

void reset_TIMER(TIM_TypeDef* TIMx) {
    TIMx->CNT = 0;
}
```

PWM

ecPWM.c

void PWM_init(PWM_t *pwm, GPIO_TypeDef *port, int pin, int pupd, int speed, int type ,int dir, uint32_t msec)

- **pwm:** enter the pwm structure
- **pin**: pin number
- **pupd**: Pull-up Pull-down
- **speed**: Timer speed
- **type**: push pull, none

- **dir**: direction of timer counter
- **mesc**: Timer period

```
void PWM_init(PWM_t *pwm, GPIO_TypeDef *port, int pin, int pupd, int speed, int type ,int dir, uint32_t msec){
// 0. Match Output Port and Pin for TIMx
        pwm->port = port;
        pwm->pin  = pin;
        PWM_pinmap(pwm);
        TIM_TypeDef* TIMx = pwm->timer;
        int CHn = pwm->ch;
    //PWM_pinmap(pwm) 여기서 TIMx와 ch를 골라준다 내가 집어넣은 값고 맞게 타이머와 채널이 설정된다
    // AF1 at PA5 = TIM2_CH1 (p.150)  타이머마다 AF값이 다르다
    //------------------------------------------------------
    /*
            TIM Ch  Por pin
            1       1       A       8
            1       1N  A       7
            1       1N  B       13

            1       2       A       9
            1       2N  B       0
            1       2N  B       14

            1       3       A       10
            1       3N  B       1
            1       3N  B       15

            2       1       A       0
            2       1       A       5
            2       1       A       15

            2       2       A       1
            2       2       B       3

            2       3       B       10

            3       1       A       6
            3       1       B       4
            3       1       C       6

            3       2       B       5
            3       2       C       7

            3       3       C       8

            3       4       C       9

            4       1       B       6
            4       2       B       7
            4       3       B       8
            4       4       B       9
        ....
        이렇게 이미 다 지정이 되어 있다. 근데 그건
        PWM_pinmap(pwm);여기서 해준다
    */
```

```
    //---------------------------------------------------------


// 1. Initialize GPIO port and pin as AF
    GPIO_AF_set(port, pin, pupd, speed, type);

// 2. Configure GPIO AFR by Pin num.
    // pwm 채널에 따른 일반화 필요
    // AFR[0] == AFRL,   AFR[1] == AFRH 로 구분짓기 위해 배열을 사용했다. 4bit 16개가 
필요하기 때문에 64bit
    // pin 0 ~ 7까지 AFR[0], pin 8 ~ 15까지 AFR[1]
    // bit wise 할 때는 0~15번 pin이기 때문에 8개씩 나뉘고 0~7씩 bit연산을 하기 때문에 
%계산을 했다.

    // AF1: TIM1,2      AF2: TIM3~5         AF3: TIM9~11
    // bit 연산자는 다 되어있다

    uint8_t AFx = 0;
    if ((TIMx == TIM1) || (TIMx == TIM2)) { AFx = 1UL;}
    else if ((TIMx == TIM3) || (TIMx == TIM4) || (TIMx == TIM5)) { AFx = 2UL; }
    else if ((TIMx == TIM9) || (TIMx == TIM10) || (TIMx == TIM11)) { AFx = 3UL; 
}

    // 각 핀별로 AFR 배열로 들어가는 일반화를 만들자
    // AFR[0] for pin: 0~7      AFR[1] for pin:8~15
    // shift 하나 할 때 2^n으로 나누는 것

    // pin >> 3 == pin / 8
    port->AFR[pin >> 3]  &= ~(0xFUL << (4*(pin%8)));    // 4 bit clear AFRx
    port->AFR[pin >> 3]  |=     AFx     << (4*(pin%8));


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

// 3-2. Direction of Counter
        // Counting direction: 0 = up-counting, 1 = down-counting
        if(dir == UPPCOUNT){        TIMx->CR1 &= ~TIM_CR1_DIR_Msk;}
        else if(dir == DOWCOUNT){   TIMx->CR1 |= TIM_CR1_DIR_Msk;}

// 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_CCMR2_OC3PE;  // 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;
        TIMx->CCMR2 |= TIM_CCMR2_OC4M_1 | TIM_CCMR2_OC4M_2;
        TIMx->CCMR2 |= TIM_CCMR2_OC4PE;
        TIMx->CCR4 = ccVal;
        TIMx->CCER &= ~TIM_CCER_CC4P;
        TIMx->CCER |= TIM_CCER_CC4E;
    }

// 5. Enable Timer Counter
    if(TIMx == TIM1) {
        TIMx->BDTR |= TIM_BDTR_MOE; // Main output enable (MOE): 0 = Disable, 1
= Enable
    }

    TIMx->CR1 |= TIM_CR1_CEN;
    // Enable counter
}
```

void PWM period_ms,us(PWM_t *pwm, float pulse_width_us)

- **pwm:** enter the pwm structure
- **pulse_width_us**: pulse_width_usec

```
void PWM_period_ms(PWM_t *pwm, uint32_t msec){
    TIM_TypeDef *TIMx = pwm->timer;
    TIM_period_ms(TIMx, msec);
}
void PWM_period_us(PWM_t *pwm, uint32_t usec){
    TIM_TypeDef *TIMx = pwm->timer;
    TIM_period_us(TIMx, usec);
}
```

void PWM_pulsewidth_ms, us(PWM_t *pwm, float pulse_width_ms)

- **pwm:** enter the pwm stuructuer
- **pulse_width_us**: pulse_width_msec

```
void PWM_pulsewidth_ms(PWM_t *pwm, float pulse_width_ms){
    int CHn = pwm->ch;
    uint32_t fsys = 0;
    TIM_TypeDef *TIMx = pwm->timer;
    uint32_t psc = TIMx->PSC;

    // Check System CLK: PLL or HSI
    //PLL
    if((RCC->CFGR & (3<<0)) == 2)      { fsys = 84000; }  // for msec 84MHz/1000
    // HSI
    else if((RCC->CFGR & (3<<0)) == 0) { fsys = 16000; }

    float fclk = fsys / (psc + 1);                    // fclk=fsys/(psc+1);
    uint32_t ccval = pulse_width_ms * fclk - 1.0;       // pulse_width_ms *fclk
- 1;

    switch(CHn){
        case 1: pwm->timer->CCR1 = ccval; break;
        case 2: pwm->timer->CCR2 = ccval; break;
        case 3: pwm->timer->CCR3 = ccval; break;
        case 4: pwm->timer->CCR4 = ccval; break;

        default: break;
    }
}

void PWM_pulsewidth_us(PWM_t *pwm, float pulse_width_us){
    int CHn = pwm->ch;
    uint32_t fsys = 0;
    TIM_TypeDef *TIMx = pwm->timer;
    uint32_t psc = TIMx->PSC;

    // Check System CLK: PLL or HSI
    //PLL
    if((RCC->CFGR & (3<<0)) == 2)      { fsys = 84; }
    // HSI
    else if((RCC->CFGR & (3<<0)) == 0) { fsys = 16; }

    float fclk = fsys / (psc + 1);                    // fclk=fsys/(psc+1);
    uint32_t ccval = pulse_width_us * fclk - 1.0;       // pulse_width_ms *fclk
- 1;
```

```
    switch(CHn){
        case 1: pwm->timer->CCR1 = ccval; break;
        case 2: pwm->timer->CCR2 = ccval; break;
        case 3: pwm->timer->CCR3 = ccval; break;
        case 4: pwm->timer->CCR4 = ccval; break;

        default: break;
    }
}
```

void PWM_duty(PWM_t *pwm, float duty)

- **pwm:** enter the pwm structure
- **duty**: duty of pwm

```
void PWM_duty(PWM_t *pwm, float duty) { //  duty=0 to 1  한 주기 안에서 duty는 0에서 1사이다
    // ccval은 arr을 도달할 때 내가 지정한 값을 지나가면 그때 HIGH가 되게하는거다
    TIM_TypeDef *TIMx = pwm->timer;
    uint32_t arr = TIMx->ARR;

    float ccval = ((float)(arr + (uint32_t)1) ) * duty ;  // (ARR+1)*dutyRatio

    int CHn = pwm->ch;

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

void PWM_pinmap(PWM_t *pwm)

pin과 타이머에 따라 포트를 지정해줌

절대 건드리면 안됨!

```
void PWM_pinmap(PWM_t *pwm){
   GPIO_TypeDef *port = pwm->port;
   int pin = pwm->pin;

   if(port == GPIOA) {
      switch(pin){
         case 0 : pwm->timer = TIM2; pwm->ch = 1; break;
         case 1 : pwm->timer = TIM2; pwm->ch = 2; break;
         case 5 : pwm->timer = TIM2; pwm->ch = 1; break;
         case 6 : pwm->timer = TIM3; pwm->ch = 1; break;
         //case 7: PWM_pin->timer = TIM1; PWM_pin->ch = 1N; break;
```

```c
            case 8 : pwm->timer = TIM1; pwm->ch = 1; break;
            case 9 : pwm->timer = TIM1; pwm->ch = 2; break;
            case 10: pwm->timer = TIM1; pwm->ch = 3; break;
            case 15: pwm->timer = TIM2; pwm->ch = 1; break;
            default: break;
         }
      }
      else if(port == GPIOB) {
         switch(pin){
            //case 0: PWM_pin->timer = TIM1; PWM_pin->ch = 2N; break;
            //case 1: PWM_pin->timer = TIM1; PWM_pin->ch = 3N; break;
            case 3 : pwm->timer = TIM2; pwm->ch = 2; break;
            case 4 : pwm->timer = TIM3; pwm->ch = 1; break;
            case 5 : pwm->timer = TIM3; pwm->ch = 2; break;
            case 6 : pwm->timer = TIM4; pwm->ch = 1; break;
            case 7 : pwm->timer = TIM4; pwm->ch = 2; break;
            case 8 : pwm->timer = TIM4; pwm->ch = 3; break;
            case 9 : pwm->timer = TIM4; pwm->ch = 4; break;
            case 10: pwm->timer = TIM2; pwm->ch = 3; break;

            default: break;
         }
      }
      else if(port == GPIOC) {
         switch(pin){
            case 6 : pwm->timer = TIM3; pwm->ch = 1; break;
            case 7 : pwm->timer = TIM3; pwm->ch = 2; break;
            case 8 : pwm->timer = TIM3; pwm->ch = 3; break;
            case 9 : pwm->timer = TIM3; pwm->ch = 4; break;

            default: break;
         }
      }
        // TIM5 needs to be added, if used.
}
```

Input Caputre

ecIC.c

void ICAP_init(IC_t *ICx, GPIO_TypeDef *port, int pin)

- **ICx** :enter the IC structure
- **pin**: IC pin number

```c
void ICAP_init(IC_t *ICx, GPIO_TypeDef *port, int pin){
// 0. Match Input Capture Port and Pin for TIMx
    ICx->port = port;
    ICx->pin  = pin;
    ICAP_pinmap(ICx);   // Port, Pin --(mapping)--> TIMx, Channel

    TIM_TypeDef *TIMx = ICx->timer;
    int TIn = ICx->ch;
    int ICn = TIn;
```

```
    ICx->ICnum = ICn;   // (default) TIx=ICx

// GPIO configuration ------------------------------------------------------------
---
// 1. Initialize GPIO port and pin as AF
    GPIO_AF_set(port, pin, NOPUPD, DEFAULT, DEFAULT);

// 2. Configure GPIO AFR by Pin num.
    uint8_t AFx = 0;
    if ((TIMx == TIM1) || (TIMx == TIM2)) { AFx = 1UL;}
    else if ((TIMx == TIM3) || (TIMx == TIM4) || (TIMx == TIM5)) { AFx = 2UL; }
    else if ((TIMx == TIM9) || (TIMx == TIM10) || (TIMx == TIM11)) { AFx = 3UL;
}

    // 각 핀별로 AFR 배열로 들어가는 일반화를 만들자
    // AFR[0] for pin: 0~7      AFR[1] for pin:8~15
    // shift 하나 할 때 2^n으로 나누는 것
    // pin >> 3 == pin / 8
    port->AFR[pin >> 3]  &= ~(0xFUL << (4*(pin%8)));            // 4 bit clear
AFRx
    port->AFR[pin >> 3]  |=     AFx     << (4*(pin%8));




// TIMER configuration -----------------------------------------------------------
--------
// 1. Initialize Timer
    TIM_init(TIMx, 1);
// 2. Initialize Timer Interrpt
    TIM_INT_init(TIMx, 1);  // TIMx Interrupt initialize
// 3. 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)
// 4. 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)
    // Ch Default Setting

    TIMx->CCMR1 &=  ~TIM_CCMR1_CC1S;                // clear
    TIMx->CCMR1 |=  TIM_CCMR1_CC1S_0;       //01<<0   CC1S    TI1=IC1

    TIMx->CCMR1 &=  ~TIM_CCMR1_CC2S;                // clear
    TIMx->CCMR1 |=  TIM_CCMR1_CC2S_0;               //01<<8   CC2s    TI2=IC2

    TIMx->CCMR2 &=  ~TIM_CCMR2_CC3S;                // clear
    TIMx->CCMR2 |=  TIM_CCMR2_CC3S_0;       //01<<0   CC3s    TI3=IC3

    TIMx->CCMR2 &=  ~TIM_CCMR2_CC4S;
    TIMx->CCMR2 |=  TIM_CCMR2_CC4S_0;

// 2. Filter Duration (use default)
// 3. IC Prescaler (use default)
```

```
// 4. Activation Edge: CCyNP/CCyP
    TIMx->CCER &= ~(0xFUL << 4*(ICn-1));
    // CCy(Rising) for ICn 기본 세팅은 rising
// 5.   Enable CCy Capture, Capture/Compare interrupt
    TIMx->CCER |= 1 << (ICn-1);                  // CCn(ICn) Capture Enable

// 6.   Enable Interrupt of CC(CCyIE), Update (UIE)
    TIMx->DIER |= 1 << 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
}
```

void ICAP_setup(IC_t *ICx, int ICn, int edge_type)

- **ICx** :enter the IC structure
- **ICn**: IC number
- **edge_type**: falling, rising, both

```
void ICAP_setup(IC_t *ICx, int ICn, int edge_type){
    TIM_TypeDef *TIMx = ICx->timer; // TIMx
    int                 CHn     = ICx->ch;      // Timer Channel CHn
    ICx->ICnum = ICn;

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


// 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<<0
CC2S    Tx_Ch2=IC2
                    else TIMx->CCMR1 |= TIM_CCMR1_CC2S_1;     //10<<0    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<<0   CC4S    Tx_Ch4=IC4
                else TIMx->CCMR2 |= TIM_CCMR2_CC4S_1;   //10<<0   CC4S
Tx_Ch3=IC4
                break;
            default: break;
        }

// Configure Activation Edge direction
    TIMx->CCER &= ~(0xFUL << 4*(ICn-1));    // Clear CCnNP/CCnP bits for ICn
    switch(edge_type){
        case IC_RISE: TIMx->CCER |= (0b000 << (4*(ICn-1) + 1));  break;
//rising:  00
        case IC_FALL: TIMx->CCER |= (0b001 << (4*(ICn-1) + 1));  break;
//falling: 01
        case IC_BOTH: TIMx->CCER |= (0b101 << (4*(ICn-1) + 1));  break; //both:
   11
    }

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

Input capture 실행을 위한 함수들

```
// Time span for one counter step
void ICAP_counter_us(IC_t *ICx, int usec){
    TIM_TypeDef *TIMx = ICx->timer;
    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 & TIM_SR_UIF << ccNum) == TIM_SR_UIF << ccNum);
}

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

void ICAP_pinmap(IC_t *timer_pin)

절대 건드리지 마세요

```
void ICAP_pinmap(IC_t *timer_pin){
   GPIO_TypeDef *port = timer_pin->port;
   int pin = timer_pin->pin;

   if(port == GPIOA) {
```

```c
        switch(pin){
            case 0 : timer_pin->timer = TIM2; timer_pin->ch = 1; break;
            case 1 : timer_pin->timer = TIM2; timer_pin->ch = 2; break;
            case 5 : timer_pin->timer = TIM2; timer_pin->ch = 1; break;
            case 6 : timer_pin->timer = TIM3; timer_pin->ch = 1; break;
            //case 7: timer_pin->timer = TIM1; timer_pin->ch = 1N; break;
            case 8 : timer_pin->timer = TIM1; timer_pin->ch = 1; break;
            case 9 : timer_pin->timer = TIM1; timer_pin->ch = 2; break;
            case 10: timer_pin->timer = TIM1; timer_pin->ch = 3; break;
            case 15: timer_pin->timer = TIM2; timer_pin->ch = 1; break;
            default: break;
        }
    }
    else if(port == GPIOB) {
        switch(pin){
            //case 0: timer_pin->timer = TIM1; timer_pin->ch = 2N; break;
            //case 1: timer_pin->timer = TIM1; timer_pin->ch = 3N; break;
            case 3 : timer_pin->timer = TIM2; timer_pin->ch = 2; break;
            case 4 : timer_pin->timer = TIM3; timer_pin->ch = 1; break;
            case 5 : timer_pin->timer = TIM3; timer_pin->ch = 2; break;
            case 6 : timer_pin->timer = TIM4; timer_pin->ch = 1; break;
            case 7 : timer_pin->timer = TIM4; timer_pin->ch = 2; break;
            case 8 : timer_pin->timer = TIM4; timer_pin->ch = 3; break;
            case 9 : timer_pin->timer = TIM4; timer_pin->ch = 3; break;
            case 10: timer_pin->timer = TIM2; timer_pin->ch = 3; break;

            default: break;
        }
    }
    else if(port == GPIOC) {
        switch(pin){
            case 6 : timer_pin->timer = TIM3; timer_pin->ch = 1; break;
            case 7 : timer_pin->timer = TIM3; timer_pin->ch = 2; break;
            case 8 : timer_pin->timer = TIM3; timer_pin->ch = 3; break;
            case 9 : timer_pin->timer = TIM3; timer_pin->ch = 4; break;

            default: break;
        }
    }
}
```

Stepper Motor

ecStepper.c

preset for stepper motor

```c
// Stepper Motor function
uint32_t step_delay = 100;
uint32_t step_per_rev = 64*32;

// Stepper Motor variable
volatile Stepper_t myStepper;

//FULL stepping sequence  - FSM
```

```c
typedef struct {
    uint8_t out;
  uint32_t next[2];
} State_full_t;


State_full_t FSM_full[4] = {
//AA'BB'

 {0b1010,{S1,S3}},
 {0b0110,{S2,S0}},
 {0b0101,{S3,S1}},
 {0b1001,{S0,S2}}

};

//HALF stepping sequence
typedef struct {
    uint8_t out;
  uint32_t next[2];
} State_half_t;

State_half_t FSM_half[8] = {

 {0b1000,{S1,S7}},  // s0
 {0b1010,{S2,S0}},  // s1
 {0b0010,{S3,S1}},  // s2
 {0b0110,{S4,S2}},  // s3
 {0b0100,{S5,S3}},  // s4
 {0b0101,{S6,S4}},  // s5
 {0b0001,{S7,S5}},  // s6
 {0b1001,{S0,S6}}      // s7
};
```

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

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

//  GPIO Digital Out Initiation
    // A = 1
    myStepper.port1 = port1;
  myStepper.pin1  = pin1;
    // A' = 2
    myStepper.port2 = port2;
  myStepper.pin2  = pin2;
    // B = 3
    myStepper.port3 = port3;
  myStepper.pin3  = pin3;
```

```
    // B' = 4
    myStepper.port4 = port4;
  myStepper.pin4  = pin4;

    //  GPIO Digital Out Initiation
    // No pull-up Pull-down , Push-Pull, Fast
    // Port1,Pin1 ~ Port4,Pin4
    GPIO_out_set(myStepper.port1, myStepper.pin1, NOPUPD, FSPEED, PUSHPULL);
    GPIO_out_set(myStepper.port2, myStepper.pin2, NOPUPD, FSPEED, PUSHPULL);
    GPIO_out_set(myStepper.port3, myStepper.pin3, NOPUPD, FSPEED, PUSHPULL);
    GPIO_out_set(myStepper.port4, myStepper.pin4, NOPUPD, FSPEED, PUSHPULL);
}
```

void Stepper_setSpeed (long whatSpeed, int mode)

```
void Stepper_setSpeed (long whatSpeed, int mode){      // rpm

    if(mode == FULL){
        step_delay =    (60000*1000) /  (GEAR_RATIO * FULL_REV * whatSpeed); //
usec / (2048*rpm)
    }
    else if(mode == HALF){
        step_delay =    (60000*1000) /  (GEAR_RATIO * HALF_REV * whatSpeed); //
usec / (4096*rpm)
    }
        delay_ms(step_delay);
}
```

void Stepper_step(int steps, int dir, int mode, long rpm)

```
void Stepper_step(int steps, int dir, int mode, long rpm){
     uint32_t state = 0; // 여기서 시작 state를 0으로 시작함 --> S0에서 시작함
     myStepper._step_num = steps;

     for(; myStepper._step_num > 0; myStepper._step_num--){ // run for step size
                Stepper_setSpeed (rpm, mode);  // delay (step_delay);

            if (mode == FULL)
                        state = FSM_full[state].next[dir]; // state = next state
                else if (mode == HALF)
                        state = FSM_half[state].next[dir];// state = next state

                Stepper_pinOut(state, mode);
   }
}
```

void Stepper_pinOut (uint32_t state, int mode)

```
void Stepper_pinOut (uint32_t state, int mode){

    if (mode == FULL){          // FULL mode
        GPIO_write(myStepper.port1, myStepper.pin1, FSM_full[state].out >> 3 & 1);
        GPIO_write(myStepper.port2, myStepper.pin2, FSM_full[state].out >> 2 & 1);
        GPIO_write(myStepper.port3, myStepper.pin3, FSM_full[state].out >> 1 & 1);
        GPIO_write(myStepper.port4, myStepper.pin4, FSM_full[state].out >> 0 & 1);

    }
    else if (mode == HALF){     // HALF mode
        GPIO_write(myStepper.port1, myStepper.pin1, FSM_half[state].out >> 3 & 1);
        GPIO_write(myStepper.port2, myStepper.pin2, FSM_half[state].out >> 2 & 1);
        GPIO_write(myStepper.port3, myStepper.pin3, FSM_half[state].out >> 1 & 1);
        GPIO_write(myStepper.port4, myStepper.pin4, FSM_half[state].out >> 0 & 1);
    }
}
```

void Stepper_stop (void)

```
void Stepper_stop (void){

    myStepper._step_num = 0;
            // All pins(Port1~4, Pin1~4) set as DigitalOut '0'
        GPIO_write(myStepper.port1, myStepper.pin1, myStepper._step_num);
        GPIO_write(myStepper.port2, myStepper.pin2, myStepper._step_num);
        GPIO_write(myStepper.port3, myStepper.pin3, myStepper._step_num);
        GPIO_write(myStepper.port4, myStepper.pin4, myStepper._step_num);

}
```

UART

ecUART.c

preset for UART

```
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_getc(USART2);
  return rxByte;
}
```

void UART2_init(void)

```
void UART2_init(void){
    // 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)

```
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)

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

void USART_begin(USART_TypeDef* USARTx, GPIO_TypeDef* GPIO_TX, int pinTX, GPIO_TypeDef* GPIO_RX, int pinRX, int baud)

```
void USART_begin(USART_TypeDef* USARTx, GPIO_TypeDef* GPIO_TX, int pinTX,
GPIO_TypeDef* GPIO_RX, int pinRX, int baud){
//1. GPIO Pin for TX and RX
    // Enable GPIO peripheral clock
    // Alternative Function mode selection for Pin_y in GPIOx
    // No pull up, No pull down
    GPIO_AF_set(GPIO_TX, pinTX, NOPUPD, HSPEED, PUSHPULL);  // GPIO mode setting
: AF
    GPIO_AF_set(GPIO_RX, pinRX, NOPUPD, HSPEED, PUSHPULL);  // GPIO mode setting
: AF

    // Set Alternative Function Register for USARTx.
    // AF7 - USART1,2
    // AF8 - USART6
    if (USARTx == USART6){
        // USART_TX GPIO AFR
        GPIO_TX->AFR[pinTX >> 3]    &= ~(0xFUL << (4*(pinTX%8)));  // 4 bit
clear AFRx
        GPIO_TX->AFR[pinTX >> 3]    |=    0b1000  << (4*(pinTX%8));
        // USART_RX GPIO AFR
        GPIO_TX->AFR[pinRX >> 3]    &= ~(0xFUL << (4*(pinRX%8)));  // 4 bit
clear AFRx
        GPIO_TX->AFR[pinRX >> 3]    |=    0b1000  << (4*(pinRX%8));
    }
    else{   //USART1,USART2
        // USART_TX GPIO AFR
        GPIO_TX->AFR[pinTX >> 3]    &= ~(0xFUL << (4*(pinTX%8)));  // 4 bit
clear AFRx
        GPIO_TX->AFR[pinTX >> 3]    |=    0b0111  << (4*(pinTX%8));
        // USART_RX GPIO AFR
        GPIO_TX->AFR[pinRX >> 3]    &= ~(0xFUL << (4*(pinRX%8)));  // 4 bit
clear AFRx
        GPIO_TX->AFR[pinRX >> 3]    |=    0b0111  << (4*(pinRX%8));
    }
```

```c
//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_Pos;      // 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

    // Configure Baud-rate
    float Hz = 84000000;    // if(USARTx==USART1 || USARTx==USART6)
    if(USARTx == USART2) Hz = 42000000;

    float USARTDIV =  Hz / (float)(16 * baud);
    // 정수파트만 남긴다
    uint32_t MNT = (uint32_t)USARTDIV;
    // 소수 파트만 남기고 16 곱해서 16진수에 맞게 만든다 round로 반올림 해준다
    uint32_t FRC = round((USARTDIV - MNT) * 16);
    // if 소수점의 크기가 16진수를 넘어가면 정수로 반올림 해준다
    if (FRC > 15) {
        MNT += 1;
        FRC = 0;
    }

// 굳이 float로 해야하나? ㅇㅇ 9600을 제외하면 다른 것들은 정수로 떨어지지 않는다
    USARTx->BRR  &= ~USART_BRR_DIV_Fraction;        // clear Fraction
    USARTx->BRR  &= ~USART_BRR_DIV_Mantissa;        // clear Mantissa

    USARTx->BRR  |= (MNT << 4) | FRC;

    // Enable TX, RX, and USARTx
    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

// 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 USART_init(USART_TypeDef* USARTx, int baud)

```
void USART_init(USART_TypeDef* USARTx, int baud){
// **********************************************************
// Default Tx,Rx GPIO, pin configuration
// USART1 - TX: PB6,  RX: PB3  (default) // TX: PA9, RX: PA10
// USART2 - TX: PA2,  RX: PA3
// USART6 - TX: PA11, RX: PA12 (default) // TX: PC6, RX: PC7
// **********************************************************

// 1. GPIO Pin for TX and RX
    GPIO_TypeDef* GPIO_TX;
    GPIO_TypeDef* GPIO_RX;
    int pinTX = 0, pinRX =0;

    if (USARTx==USART1) {
        GPIO_TX = GPIOB;
        GPIO_RX = GPIOB;
        pinTX = 6;
        pinRX = 3;
    }
    if (USARTx==USART2) {
        GPIO_TX = GPIOA;
        GPIO_RX = GPIOA;
        pinTX = 2;
        pinRX = 3;
    }
    if (USARTx==USART6) {
        GPIO_TX = GPIOA;
        GPIO_RX = GPIOA;
        pinTX = 11;
        pinRX = 12;
    }
    // if for other USART input?

    USART_begin(USARTx, GPIO_TX, pinTX, GPIO_RX, pinRX, baud);
```

```
}
```

USART get and RX 통신

```
uint8_t USART_getc(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));
}

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

ADC

ADC TIM Table

Table 42. External trigger for regular channels

Source	Type	EXTSEL[3:0]
TIM1_CH1 event	Internal signal from on-chip timers	0000
TIM1_CH2 event		0001
TIM1_CH3 event		0010
TIM2_CH2 event		0011
TIM2_CH3 event		0100
TIM2_CH4 event		0101
TIM2_TRGO event		0110
TIM3_CH1 event		0111
TIM3_TRGO event		1000
TIM4_CH4 event		1001
TIM5_CH1 event		1010
TIM5_CH2 event		1011
TIM5_CH3 event		1100
Reserved		1101
Reserved		1110
EXTI line11	External pin	1111

INJ ADC TIM Table

Table 43. External trigger for injected channels

Source	Connection type	JEXTSEL[3:0]
TIM1_CH4 event	Internal signal from on-chip timers	0000
TIM1_TRGO event		0001
TIM2_CH1 event		0010
TIM2_TRGO event		0011
TIM3_CH2 event		0100
TIM3_CH4 event		0101
TIM4_CH1 event		0110
TIM4_CH2 event		0111
TIM4_CH3 event		1000
TIM4_TRGO event		1001
TIM5_CH4 event		1010
TIM5_TRGO event		1011
Reserved		1100
Reserved		1101
Reserved		1110
EXTI line15	External pin	1111

ADC pin map

ADC Pinout Map

ADC1

Channel	Port	Pin
0	A	0
1	A	1
2		
3		
4	A	4
5	A	5
6	A	6
7	A	7
8	B	0
9	B	1
10	C	0
11	C	1
12	C	2
13	C	3
14	C	4
15	C	5

ecADC.c

void ADC_init(GPIO_TypeDef *port, int pin, int trigmode)

- **port:** ADC port
- **pin**: Timer period in msec
- **trigmode**:

```
void ADC_init(GPIO_TypeDef *port, int pin, int trigmode){  //mode 0 : SW, 1 :
TRGO
// 0. Match Port and Pin for ADC channel
    int CHn = ADC_pinmap(port, pin);            // 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, NOPUPD);           // 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));         // clear bits
        ADC1->SMPR2 |= 4U << (3*CHn);                   // sampling time
conversion : 84
    }
    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;                 // default : Single conversion
mode

// 4. Single Channel or Scan mode
    // Configure the sequence length
    ADC1->SQR1 &= ADC_SQR1_L;                   // 0000: 1 conversion in the
regular channel conversion sequence

    // Configure the channel sequence
    ADC1->SQR3 &= ~ADC_SQR3_SQ1;                    // SQ1 clear bits
    ADC1->SQR3 |= (CHn & ADC_SQR3_SQ1);     // Choose the channel to convert
firstly

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

// 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


/* --------------------------------------------------------------------------
-------*/
```

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

    // TRGO Initialize : TIM3, 1msec, RISE edge
    if(trigmode == TRGO) ADC_TRGO(TIM3, 1, RISE);

}
```

void ADC_INJ_init(GPIO_TypeDef *port, int pin, int trigmode)

```
void ADC_INJ_init(GPIO_TypeDef *port, int pin, int trigmode){  //mode 0 : SW, 1
: TRGO
// 0. Match Port and Pin for ADC channel
    int CHn = (uint32_t)ADC_pinmap(port, pin);          // ADC Channel <-
>Port/Pin mapping

// GPIO configuration ------------------------------------------------------------
----------
// 1. Initialize GPIO port and pin as ANALOG, no pull up / pull down
    GPIO_init(port, (uint32_t)pin, ANALOG);                 // ANALOG = 3
    GPIO_pupd(port, (uint32_t)pin, NOPUPD);             // EC_NONE = 0

// ADC configuration    --------------------------------------------------------
-------------
// 1. Total time of conversion setting
    // Enable ADC pheripheral clock
    RCC->APB2ENR  |= 1UL<<8;        // Enable the clock of RCC_APB2ENR_ADC1EN,
ADC1ENABLE

    // Configure ADC clock pre-scaler
    ADC->CCR &= ~(3UL<<16);
//  ADC->CCR |= 3UL<<16;                // 11: PCLK2 divided by 8   (21MHz)

    // Configure ADC resolution
    ADC1->CR1 &= ~(3UL<<24);            // 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  &= ~(7U<<3*CHn);           // sampling time conversion : 84
//      ADC1->SMPR2  |= 4U<<3*CHn;              // sampling time conversion : 84
//  }
//  else{
//      ADC1->SMPR1  &= ~(7U<<3*(CHn%10));
//      ADC1->SMPR1  |= 4U<<3*(CHn%10);         // how about change one line
code?
//  }
//

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

```
// 3. Repetition: Single or Continuous conversion
    ADC1->CR2 &= ~(1UL<<1);                         // **default : Single
conversion mode

// 4. Single Channel or Scan mode
    //  - Single Channel: scan mode, right alignment
    ADC1->CR1 |= 1UL<<8;                                    // 1: Scan mode
enable, more than 2 chs
    ADC1->CR2 &= ~(1UL<<11);                            // 0: Right alignment,
only 1 ch

    // Configure the sequence length
    ADC1->JSQR &= ~(3UL<<20);                       // 0000: 1 conversion in the
injected channel conversion sequence

    // Configure the channel sequence
    ADC1->JSQR &= ~(0x1F<<0);                       //  SQ1 clear bits
    ADC1->JSQR |= (CHn & 0x1F);                 //  Choose the channel to
convert firstly
                                                                            //
CHn should be 1~4

// 5. Interrupt Enable

    // Enable JEOC(conversion) interrupt.
    ADC1->CR1 &= ~(1UL<<7);                 // Interrupt reset
    ADC1->CR1 |= 1UL<<7;                    // Interrupt enable

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

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

    // TRGO Initialize : TIM4, 1msec, RISE edge
    if(trigmode==TRGO) ADC_INJ_TRGO(TIM4, 1, RISE);

}
```

void ADC_TRGO(TIM_TypeDef* TIMx, int msec, int edge)

```
void ADC_TRGO(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


// HW Trigger configuration -----------------------------------------------------
--------

// 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 &= ~TIM_CR2_MMS;                        // reset MMS
  TIMx->CR2 |= TIM_CR2_MMS_2;                       //100: Compare - OC1REF signal
is used as trigger output (TRGO)

    // Output Compare Mode
  TIMx->CCMR1 &= ~TIM_CCMR1_OC1M;                       // OC1M : output compare 1
Mode
  TIMx->CCMR1 |= TIM_CCMR1_OC1M_1 | TIM_CCMR1_OC1M_2;  // OC1M = 110 for compare
1 Mode ch1

  // OC1 signal
  TIMx->CCER |= TIM_CCER_CC1E;          // CC1E Capture enabled
    TIMx->CCR1  = (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_INJ_TRGO(TIM_TypeDef* TIMx, int msec, int edge)

```
void ADC_INJ_TRGO(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 |= 1UL<<1;        // Discontinuous conversion mode(need clear
first)
    //ADC1->CR2 &= ~(1UL<<1);       // Discontinuous conversion mode(need clear
first)
    ADC1->CR2 |= 1UL<<10;           // Enable EOCS


    // HW Trigger configuration ---------------------------------------------------
------------

// 1. TIMx Trigger Output Config
    // Enable TIMx Clock
    TIM_init(TIMx, msec);
    TIMx->CR1 &= ~(1UL<<0);                     //counter disable(to change
setting)

    // Set PSC, ARR
  //TIM_period_ms(TIMx, msec);                  //(no need)

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

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

  // OC1 signal
  TIMx->CCER |= 1UL<<0;                     // CC1E Capture enabled
    TIMx->CCR1  = (TIMx->ARR)/2;                // duty ratio 50%

  // Enable TIMx
  TIMx->CR1 |= 1UL<<0;                              //counter enable

// 2. ADC HW Trigger Config.
    // Select Trigger Source
    ADC1->CR2 &= ~ADC_CR2_JEXTSEL;          // reset JEXTSEL


    if(TIMx==TIM1)          ADC1->CR2 |= 1UL<<16;   // TIM1 TRGO event
    else if(TIMx==TIM2) ADC1->CR2 |= 3UL<<16;   // TIM2 TRGO event
    else if(TIMx==TIM4) ADC1->CR2 |= 9UL<<16;   // TIM4 TRGO event
    else if(TIMx==TIM5) ADC1->CR2 |= 11UL<<16;  // TIM5 TRGO event

    //Select Trigger Polarity
    ADC1->CR2 &= ~ADC_CR2_JEXTEN;                               // reset JEXTEN,
default
    if(edge==RISE)          ADC1->CR2 |= 1UL<<20;       // trigger detection
rising edge
    else if(edge==FALL) ADC1->CR2 |= 2UL<<20;       // trigger detection falling
edge
```

```
    else if(edge==BOTH) ADC1->CR2 |= 3UL<<20;       // trigger detection both
edge

}
```

void ADC_continue(int contmode)

```
void ADC_continue(int contmode){
    if(contmode == CONT){
        // Repetition: Continuous conversion
        ADC1->CR2 |= ADC_CR2_CONT;                  // Enable Continuous
conversion mode
        ADC1->CR1 &= ~ADC_CR1_SCAN;                 // 0: Scan mode disable
    }
    else {
//if(contmode==SINGLE)
        // Repetition: Single conversion
        ADC1->CR2 &= ~ADC_CR2_CONT;             // Disable Continuous conversion
mode
        ADC1->CR1 |= ADC_CR1_SCAN;                  // 1: Scan mode enable
    }
}
```

void ADC_sequence(int length, int *seq)

```
void ADC_sequence(int length, int *seq){

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

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

void ADC_INJ_sequence(int length, int *seq)

```
void ADC_INJ_sequence(int length, int *seq){
    //length : how many use  // seq = channel that you want to use, ex) seq[2] =
{3, 5};
    ADC1->JSQR &= ~(3UL<<20);       // reset length of conversions in the
injected channel
    ADC1->JSQR |= (length-1)<<20;   // conversions in the injected channel
conversion sequence

    for(int i = 0; i<length; i++){

        // JL:00 JSQR4 only
        // JL:01 JSQR4, JSQR3
        // JL:10 JSQR4, JSQR3, JSQR2
        // JL:11 JSQR4, JSQR3, JSQR2, JSQR1
        ADC1->JSQR &= ~(0x1FUL<<((3-i)*5)); // Choose the channel to convert
sequence
        ADC1->JSQR |= seq[i] <<((3-i)*5);   // Choose the channel to convert
sequence
    }
}
```

void ADC_start(void)

```
void ADC_start(void){
    // Enable ADON, SW Trigger---------------------------------------------------
----------------------------
    ADC1->CR2 |= ADC_CR2_ADON;
    ADC1->CR2 |= ADC_CR2_SWSTART;
}
```

void ADC_INJ_start(void)

```
void ADC_INJ_start(void){
    // Enable ADON, SW Trigger---------------------------------------------------
----------------------------
    // Enable ADC
    ADC1->CR2 |= ADC_CR2_ADON;
    // ADC INJ Start (SW, once)
    ADC1->CR2 |= ADC_CR2_JSWSTART;
}
```

uint32_t is_ADC_EOC(void)

```
uint32_t is_ADC_EOC(void){
    return (ADC1->SR & ADC_SR_EOC) == ADC_SR_EOC;
}
```

uint32_t is_ADC_OVR(void)

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

void clear_ADC_OVR(void)

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

uint32_t is_ADC_JEOC(void)

```
uint32_t is_ADC_JEOC(void){
    return ADC1->SR & ADC_SR_JEOC;
}
```

void clear_ADC_JEOC(void)

```
void clear_ADC_JEOC(void){
    ADC1->SR &= ~ADC_SR_JEOC;
}
```

uint32_t ADC_read()

```
uint32_t ADC_read(){
    return ADC1->DR;
}
```

Function

ecFunc.c

void sevensegment_init(void)

GPIO output setting for 7segment initialization

```
void sevensegment_init(void) {
    // led a setup
```

```
    GPIO_out_set(GPIOA, PA8, NOPUPD, MSPEED, PUSHPULL);
    // led b setup
    GPIO_out_set(GPIOB, PB10, NOPUPD, MSPEED, PUSHPULL);
  // led c setup
    GPIO_out_set(GPIOA, PA7, NOPUPD, MSPEED, PUSHPULL);
    // led d setup
    GPIO_out_set(GPIOA, PA6, NOPUPD, MSPEED, PUSHPULL);
    // led e setup
    GPIO_out_set(GPIOA, PA5, NOPUPD, MSPEED, PUSHPULL);
    // led f setup
    GPIO_out_set(GPIOA, PA9, NOPUPD, MSPEED, PUSHPULL);
    // led g setup
    GPIO_out_set(GPIOC, PC7, NOPUPD, MSPEED, PUSHPULL);
    // led DP setup
    GPIO_out_set(GPIOB, PB6, NOPUPD, MSPEED, PUSHPULL);
}
```

void sevensegment_decoder(uint8_t flag)

seven_segment is second array. The array gives flag to GPIO_write.

```
void sevensegment_decoder(uint8_t flag) {
    int seven_segment[11][8] = {
            {LOW,   LOW,    LOW,    LOW,    LOW,    LOW,    HIGH,   HIGH},
 //zero
            {HIGH,  LOW,    LOW,    HIGH,   HIGH,   HIGH,   HIGH,   HIGH},
 //one
            {LOW,   LOW,    HIGH,   LOW,    LOW,    HIGH,   LOW,    HIGH},
 //two
            {LOW,   LOW,    LOW,    LOW,    HIGH,   HIGH,   LOW,    HIGH},
 //three
            {HIGH,  LOW,    LOW,    HIGH,   HIGH,   LOW,    LOW,    HIGH},
 //four
            {LOW,   HIGH,   LOW,    LOW,    HIGH,   LOW,    LOW,    HIGH},
 //five
            {LOW,   HIGH,   LOW,    LOW,    LOW,    LOW,    LOW,    HIGH},
 //six
            {LOW,   LOW,    LOW,    HIGH,   HIGH,   HIGH,   HIGH,   HIGH},
 //seven
            {LOW,   LOW,    LOW,    LOW,    LOW,    LOW,    LOW,    HIGH},
 //eight
            {LOW,   LOW,    LOW,    HIGH,   HIGH,   LOW,    LOW,    HIGH},
 //nine
            {HIGH,  HIGH,   HIGH,   HIGH,   HIGH,   HIGH,   HIGH,   LOW}
 //dot
    };

    GPIO_write(GPIOA, PA8, seven_segment[flag][0]);         // a
    GPIO_write(GPIOB, PB10, seven_segment[flag][1]);        // b
    GPIO_write(GPIOA, PA7, seven_segment[flag][2]);         // c
    GPIO_write(GPIOA, PA6, seven_segment[flag][3]);         // d
    GPIO_write(GPIOA, PA5, seven_segment[flag][4]);         // e
    GPIO_write(GPIOA, PA9, seven_segment[flag][5]);         // f
    GPIO_write(GPIOC, PC7, seven_segment[flag][6]);         // g
    GPIO_write(GPIOB, PB6, seven_segment[flag][7]);         // dp
```

```
}
```

void bitToggle(GPIO_TypeDef* Port, int pin)

Bit toggle use XOR

```
void bitToggle(GPIO_TypeDef* Port, int pin) {
    Port->ODR ^= (1 << pin);
}
```

void LED4_toggle(uint8_t flag)

```
void LED4_toggle(uint8_t flag){     // use truth table of 4 leds circuit
    int led4[4][4] = {
        {HIGH,      LOW,        LOW,        LOW},
        {LOW,       HIGH,       LOW,        LOW},
        {LOW,       LOW,        HIGH,       LOW},
        {LOW,       LOW,        LOW,        HIGH}
    };
        GPIO_write(GPIOA, PA5, led4[flag][0]);      // PA5
        GPIO_write(GPIOA, PA6, led4[flag][1]);      // PA6
        GPIO_write(GPIOA, PA7, led4[flag][2]);      // PA7
        GPIO_write(GPIOB, PB6, led4[flag][3]);      // PB6d
}
```

void bitToggle(GPIO_TypeDef* Port, int pin)

```
void bitToggle(GPIO_TypeDef* Port, int pin) {

    Port->ODR ^= (1 << pin);

}
```

void LED_4_init(void) for mid term

```
void LED_4_init(void){

    // led 0bit setup
    GPIO_out_set(GPIOA, PA0, NOPUPD, MSPEED, PUSHPULL);
    // led 1bit setup
    GPIO_out_set(GPIOA, PA1, NOPUPD, MSPEED, PUSHPULL);
  // led 2bit setup
    GPIO_out_set(GPIOB, PB0, NOPUPD, MSPEED, PUSHPULL);
    // led 3bit setup
    GPIO_out_set(GPIOC, PC1, NOPUPD, MSPEED, PUSHPULL);

    uint8_t flag_init = 0;
```

```
    LED_4_decoder(flag_init);
}
```

void LED_4_decoder(uint8_t flag) for mid term

```
{

    int led4[16][4] = {

        {0,0,0,0},      // 0
        {0,0,0,1},      // 1
        {0,0,1,0},      // 2
        {0,0,1,1},      // 3
        {0,1,0,0},      // 4
        {0,1,0,1},      // 5
        {0,1,1,0},      // 6
        {0,1,1,1},      // 7
        {1,0,0,0},      // 8
        {1,0,0,1},      // 9
        {1,0,1,0},      // 10
        {1,0,1,1},      // 11
        {1,1,0,0},      // 12
        {1,1,0,1},      // 13
        {1,1,1,0},      // 14
        {1,1,1,1}           // 15
    };

    GPIO_write(GPIOA, PA0, led4[flag][0]);      // a
    GPIO_write(GPIOA, PA1, led4[flag][1]);      // b
    GPIO_write(GPIOB, PB0, led4[flag][2]);      // c
    GPIO_write(GPIOC, PC1, led4[flag][3]);      // d

}
```

void LED_3_init(void) for mid term

```
void LED_3_init(void){

    // led 0bit setup
    GPIO_out_set(GPIOA, PA0, NOPUPD, MSPEED, PUSHPULL);
    // led 1bit setup
    GPIO_out_set(GPIOA, PA1, NOPUPD, MSPEED, PUSHPULL);
  // led 2bit setup
    GPIO_out_set(GPIOB, PB0, NOPUPD, MSPEED, PUSHPULL);

    uint8_t flag_init = 0;

    LED_3_decoder(flag_init);
}
```

void LED_3_decoder(uint8_t flag) for mid term

```
void LED_3_decoder(uint8_t flag){

    int led3[8][3] = {

        {0,0,0},        // 0
        {0,0,1},        // 1
        {0,1,0},        // 2
        {0,1,1},        // 3
        {1,0,0},        // 4
        {1,0,1},        // 5
        {1,1,0},        // 6
        {1,1,1},        // 7
    };

    GPIO_write(GPIOA, PA0, led3[flag][0]);      // a
    GPIO_write(GPIOA, PA1, led3[flag][1]);      // b
    GPIO_write(GPIOB, PB0, led3[flag][2]);      // c

}
```

Include header

ecInclude.h

```
/*----------------------------------------------------------------\
@ Embedded Controller edit by Seung-Eun Hwang
Author           : SeungEun Hwang
Created          : 09-13-2022
Modified         : 10-17-2022
Language/ver     : C++ in Keil uVision

Description      : ecInclude header file
/----------------------------------------------------------------*/

#ifndef __ECINCLUDE_H
#define __ECINCLUDE_H

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdint.h>


#include "stm32f4xx.h"
#include "stm32f411xe.h"
#include "ecRCC.h"
#include "ecGPIO.h"
#include "ecSysTick.h"
#include "ecFunc.h"
#include "ecEXTI.h"
#include "ecTIM.h"
#include "ecPWM.h"
```

```
#include "ecStepper.h"
#include "ecUART.h"
#include "ecIC.h"
#include "ecADC.h"
#include "ecRGBCS.h"


#define MCU_CLK_PLL 84000000
#define MCU_CLK_HSI 16000000
#define MCU_CLK_HSE 8000000



//#include "ecGPIO_API.h"



#ifdef __cplusplus

#endif /* __cplusplus */

#endif
```