유자차

Unbelievable 자율주행 자동차

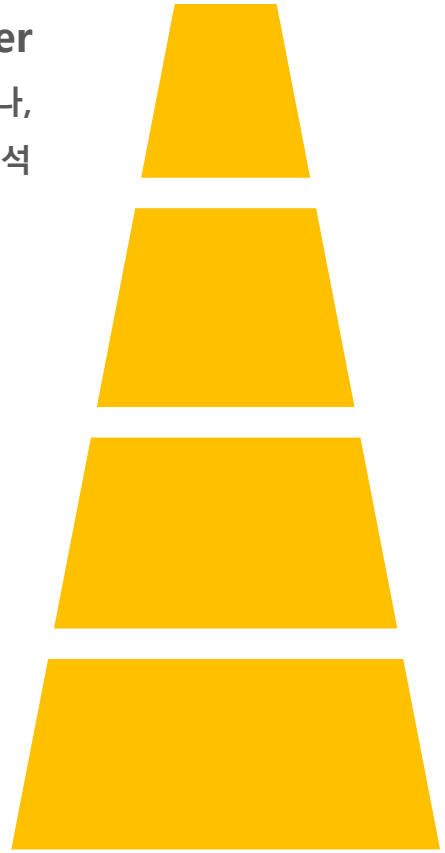
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

Member

장성환, 정한별, 문한나,
정상용, 하성용, 이우석

Details

I2C 통신을이용한 mpu6050 프로그래밍 하기
기존에 있던 소스에서 에러가 확률을 적게 수정



Contents

1. I2C 란 무엇인가?
2. I2C 의 특징은?
3. I2C 의 통신 Protocol은?
4. MPU6050이란?
5. MPU6050 배선도 및 코딩하기

정한별 발표

지금 부터 I2C에 대해서 설명하겠습니다.

1. I2C 란 무엇인가?
2. I2C 의 특징은?
3. I2C 의 통신 Protocol은?
4. MPU6050이란?
5. MPU6050 배선도 및 코딩하기

I2C란 무엇인가?

I2C Bus 는 동일한 회로 기판 상에 상주하는 컴포넌트 사이의 통신을 쉽게 할 수 있도록 80년대 초반 Philips Semiconductors (현재 NXP) 에 의해 디자인 되었다. 이 bus 는 Inter-IC 나 I2C-bus 라고 불린다.

I2C-bus 는 여러가지 control architecture 로 사용되어진다.

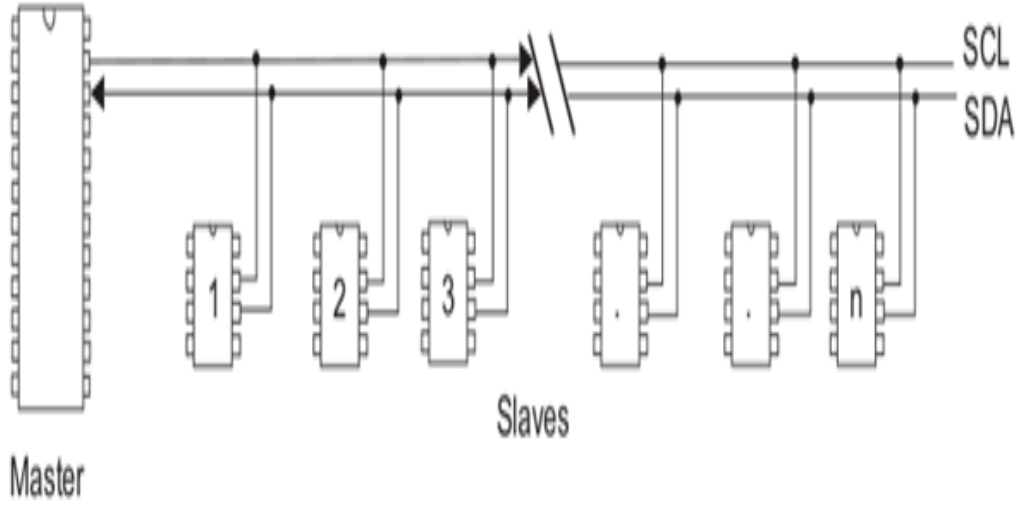
- -System Management Bus (SMBus)
- -Power Management Bus (PMBus)
- -Intelligent Platform Management Interface (IPMI)
- -Display Data Channel (DDC)
- -Advanced Telecom Computing Architecture (ATCA)

직렬 컴퓨터 버스이며 마더보드, 임베디드 시스템, 휴대전화 등에 저속의 주변 기기를 연결하기 위해 사용한다.

마이크로컨트롤러에 메모리나 I/O 소자들을 인터페이스하는 것은 물론 어드레스 버스와 데이터 버스를 사용하는 병렬 방식이 기본이지만, TV 나 VCR과 같은 가전제품에서는 이렇게 할 경우 많은 버스선이나 인터페이스 소자들 때문에 PCB의 면적을 많이 차지 해서 사용 하게 되었다.

제품의 가격을 낮추거나 소형화하는데 좋으며 필립스에서 **2선식의 간편한 양방향 직렬통신용 버스**를 제안 하면서 사용하게 되었다.

Figure 31-1. Multiple I2C Modules Connection Diagram



I2C의 특징은?

MCU, LCD, LED Driver, EEPROM, A/D, D/A Converter, Remote I/O Port, RAM, real-time clock,
radio and video systems 의 Digital Turning, signal processing, temperature sensors, smart cards 등 많은 곳에 포함되어 있다. 회로 단순화 및 효율적인 제어를 위해 개발 되었다.

a. 2개의 Bus Line 만 필요하다. (SDA, SCL)

b. Bus 에 연결된 장치들은 마스터/슬레이브 관계로 되어 있고 고유주소를 가진다.

c. 두개의 마스터가 동시에 데이터를 전송할 경우 데이터 손실을 방지하기 위하여 충돌을 검출하고 중재하는 멀티 마스터 방식이다.

d. 통신속도는 아래와 같다.
Bidirectional bus :
- Standard mode : 100Kbit/s
- Fast mode : 400Kbit/s
- Fast mode + : 1Mbit/s
- High Speed mode : 3.4Mbit/s

Unidirectional bus :
- Ultra Fast mode : 5Mbit/s

e. Bus Line 에 Spike (50ns) 는 Input Filter 에 의해 걸러진다.
불안정해진다.)

f. 동일한 버스에 접속 될 수 있는 IC 의 개수는 Bus 의 최대 Capacitance (400pF)에 의해 제한된다.

g.송신기 또는 수신기 인 것 외에도 I2C 버스에 연결된 장치 데이터 전송을 수행 할 때 마스터 또는 슬레이브로 간주 될 수도 있다.

h.전송 중에 마스터에 의해 어드레싱 된 디바이스는 슬레이브로 간주된다.

i. SPI 통신과 비교하자면 통신속도 면에서는 **SPI가 빠르며**, **연결**되어야 하는 **선의 개수**는 **I2C 경우 2개(SPI는 4개)**이므로 유리하다. 두 통신 모드 UART와는 다르게 **1:N 통신**이 가능하다. 즉, 다수의 슬레이브를 연결할 수 있다. 이는 위의 주소 지정모드를 이용해서 **슬레이브가 가지고 있는 주소로 접근**함으로써 가능한 것이다.

j. 통신이 개시되었을 때 SDA(Serial Data)는 데이터를 주고 받는 역할을 하고 SCL(Serial Clock)은 동기화를 위한 CLOCK 역할을 합니다.

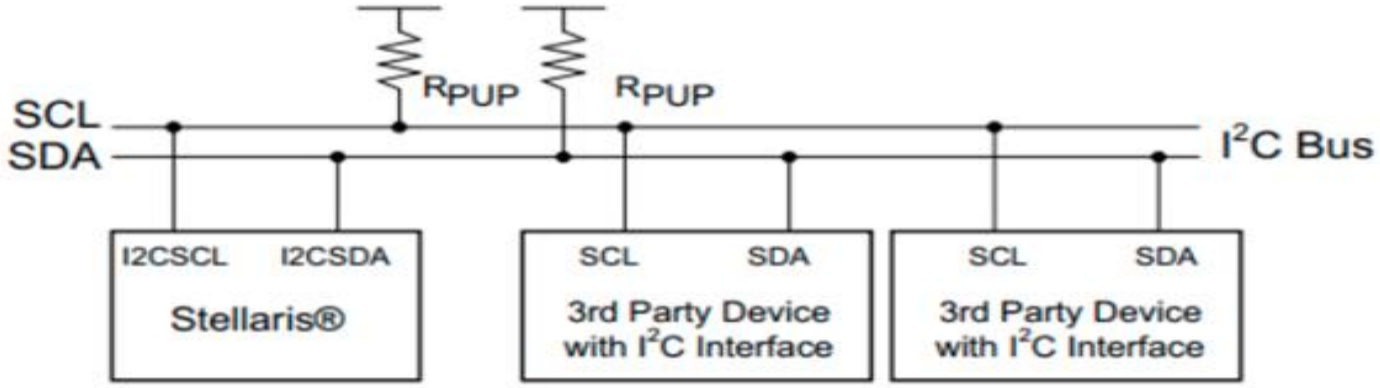
k.풀업 저항이 연결된 직렬 데이터(SDA)와 직렬 클럭(SCL)이라는 두 개의 양 방향 오픈 컬렉터 라인을 사용(즉, 두 개 다 풀업 저항을 사용)한다.

l. 10K 옴으로 풀업 연결하여 데이터 송수신이 되지 않을 때는 항상 HIGH 상태로 유지해주는 것이 좋다. 플로팅이 되는 경우를 방지하기 위해서 이다.(플로팅 상태는 잡음에 매우 취약해 시스템이

I2C의 특징은?

<풀업저항을 걸어주는 경우>

Figure 15-2. I²C Bus Configuration



I2C의 통신 Protocol은?

3.1. SDA and SCL logic levels

다양한 Device 들이 I2C Bus 에 연결 되기 때문에 SDA 와 SCL 의 Logic Level 은 고정되어 있지 않다.

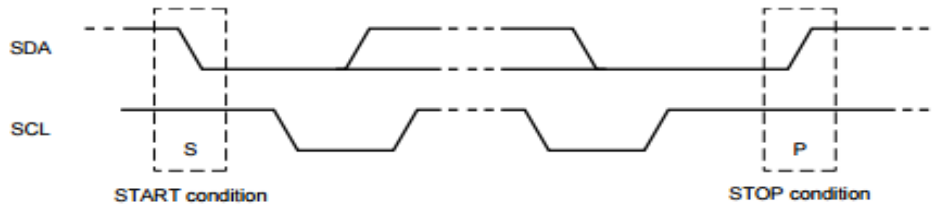
SDA 와 SCL 의 **Logic Level** 은 **Vdd 의 70% 이상일때 High,**
Vdd 의 30% 이하일때 Low 로 결정된다.

3.2. Data 의 유효성

a. SDA Line 의 Data 는 **SCL Line 이 High 인 동안 안정적이어야 한다.**
b. SCL Line 의 Clock 이 **Low 일때 Data Line 은 High/Low 로 상태를 변경 할 수 있다.**

3.3. START and STOP conditions

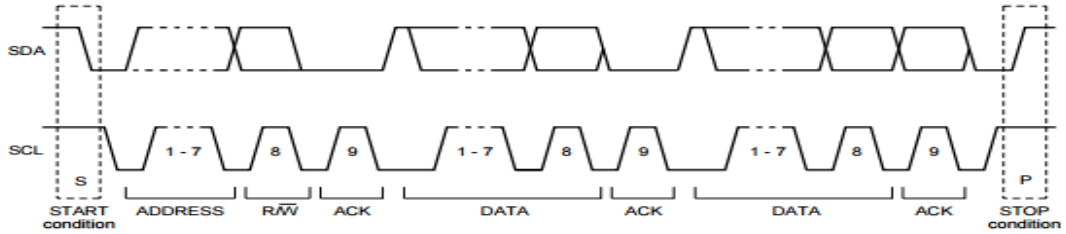
a. 모든 처리는 **START(S) 로 시작**하고 **STOP(P) 으로 종료**된다.
b. **START 조건**의 정의는 **SCL Line 이 High** 일때 **SDA Line 이 Low** 상태가 될때 이다.
c. **STOP 조건**의 정의는 **SCL Line 이 High** 일때 **SDA Line 이 High** 상태가 될때 이다.



d. **START 및 STOP 조건**은 항상 마스터에 의해 생성된다.
e. Bus 는 **START 조건 후에 Busy 상태**로 간주 된다.
f. Bus 는 **STOP 조건 후**에 일정 시간(Buf Time : **0.5us ~ 4.7us**) 이 지나면 **Free 상태**로 간주된다.
g. **Repeat START 조건** 일 경우 Bus 는 **Busy 상태**로 유지 된다.
h. **START 와 Repeat START** 의 조건은 **기능적으로 동일**하다.

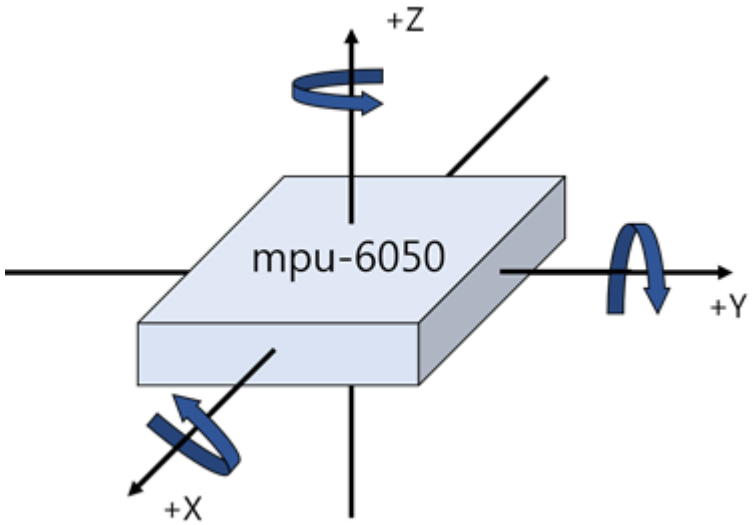
3.8. The slave address and R/W bit

a. 데이터 전송은 아래 그림과 같은 형식을 따른다. b. START 상태 후에, **슬레이브 주소가 전송**된다.
c. **Address bit 인 7번째 비트 다음 1bit** 에 **데이터 방향** 비트 (R/W) 를 나타낸다. (**'0' WRITE, '1' READ)**
d. 데이터 전송은 항상 **마스터에서 생성 된 STOP 신호에 의해 종료**된다.
e. Master 가 다른 Slave 와 **계속 통신하고자하는 경우,** STOP 조건을 발생시키지 않고 **Repeat START 조건을 생성**한다.



6축 기울기 센서 - MPU6050

MPU6050 이란?

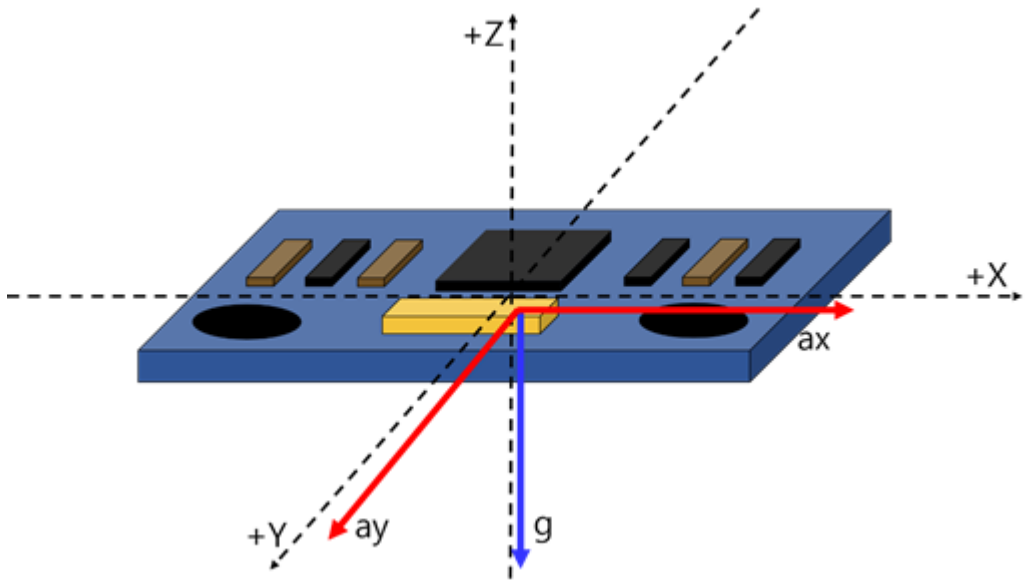


기울기 센서인 MPU-6050센서는 6축 자이로 가속도 센서라고 불리는 센서입니다.

여기서 6축의 뜻은 6자유도(DOF)를 의미하며 **가속도3축 + 자이로 2축 +온도1축**을 줄여서 6축 기울기 센서라고 부릅니다.

MPU6050 원리

그림

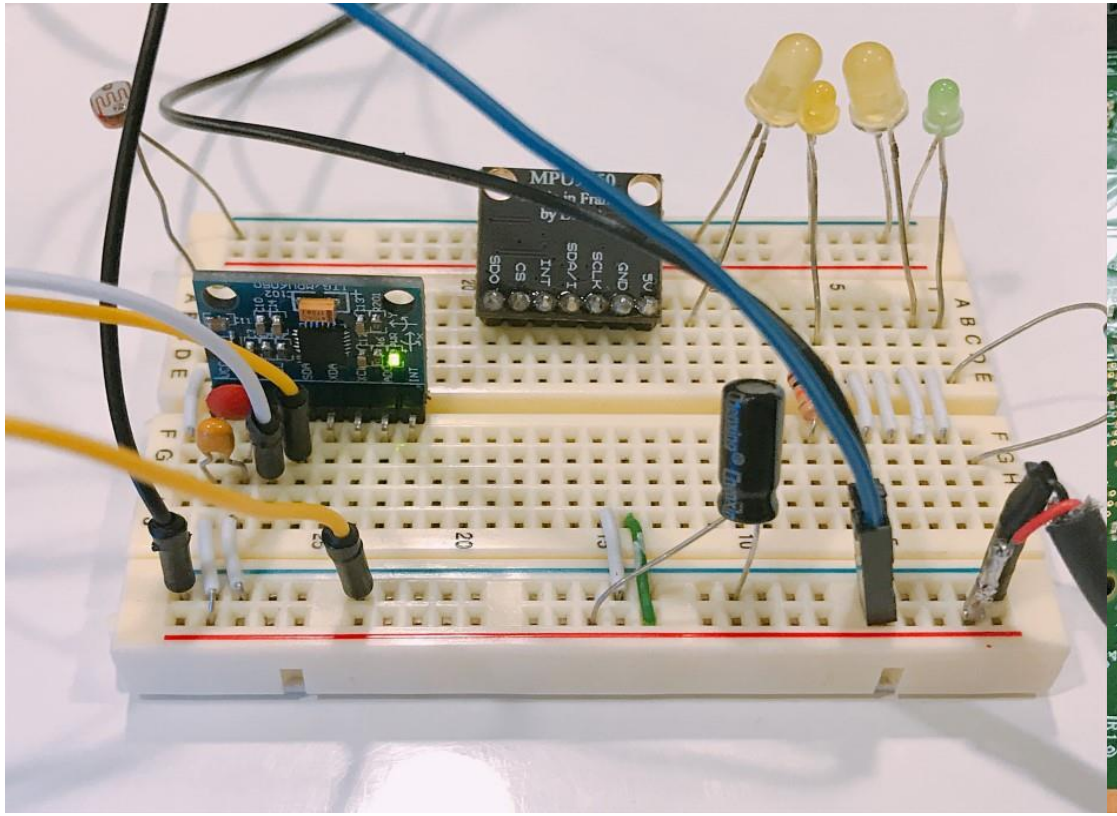


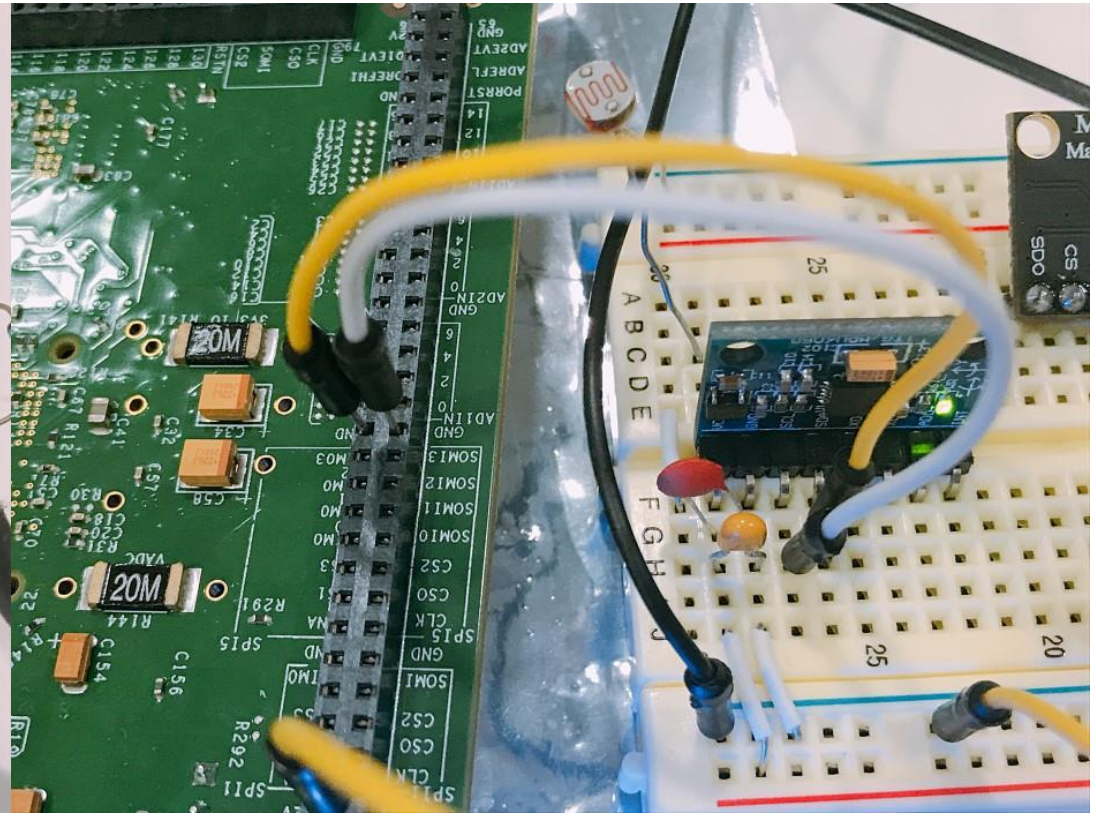
이해하기

가속도는 시간에 대한 속도 변화의 비율을 뜻 합니다. **가속도 센서**는 **가속도 자체를 측정하는 것이 아닌 중력가속도를 이용하여 가속도를 측정**하게 됩니다.
중력 가속도가 3축으로 얼만큼의 영향을 주었는가를 측정하여 센서의 기울어진 정도를 확인할 수 있습니다.
가속도센서의 민감도는 매우 높아 정적인 상태에서만 정확한 기울기를 측정할 수 있다는 한계를 가지고 있습니다.

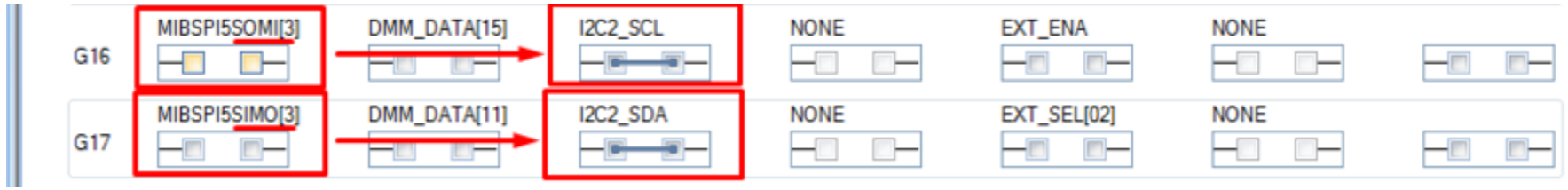
자이로 센서는 각속도를 측정 할 수 있는 센서로서 3축의 물리 량을 측정하여 센서의 움직임을 감지하고 기울기를 측정할 수 있습니다. **가속도 센서의 비해 비교적 안정적**인 값이 출력되지만 **각도를 계산하는 과정에서 사용되는 적분에 의해 시간이 지날수록 누적된 오차가 발생**하게 됩니다. 따라서 가속도 센서와 자이로 센서의 장점을 적절히 혼합하여 센서를 사용하는 것이 중요합니다.

MPU6050 배선





MPU6050 배선



위와 같이 연결하면 된다. 왜 somi pin, simo pin에 연결하나요 라고 한다면 같은 pin mux에 물려 있기 때문에 그래 보이는 것이다.

somi 는 scl
simo 는 sda

생각을 하면 이렇다 spi에서도 데이터를 보내고 받는 녀석이 있다고 생각한다.
mpu6050이 마스터라고 **mcu가 슬레이브**라고 생각하자.
mpu6050에서 보내는 data가 있을 것이고 그것은 말그대로 master out, slave input이 된다.

그거 하나만 기억하고 연결한다.

그래서 simo3 -> sda에 연결하면 된다.

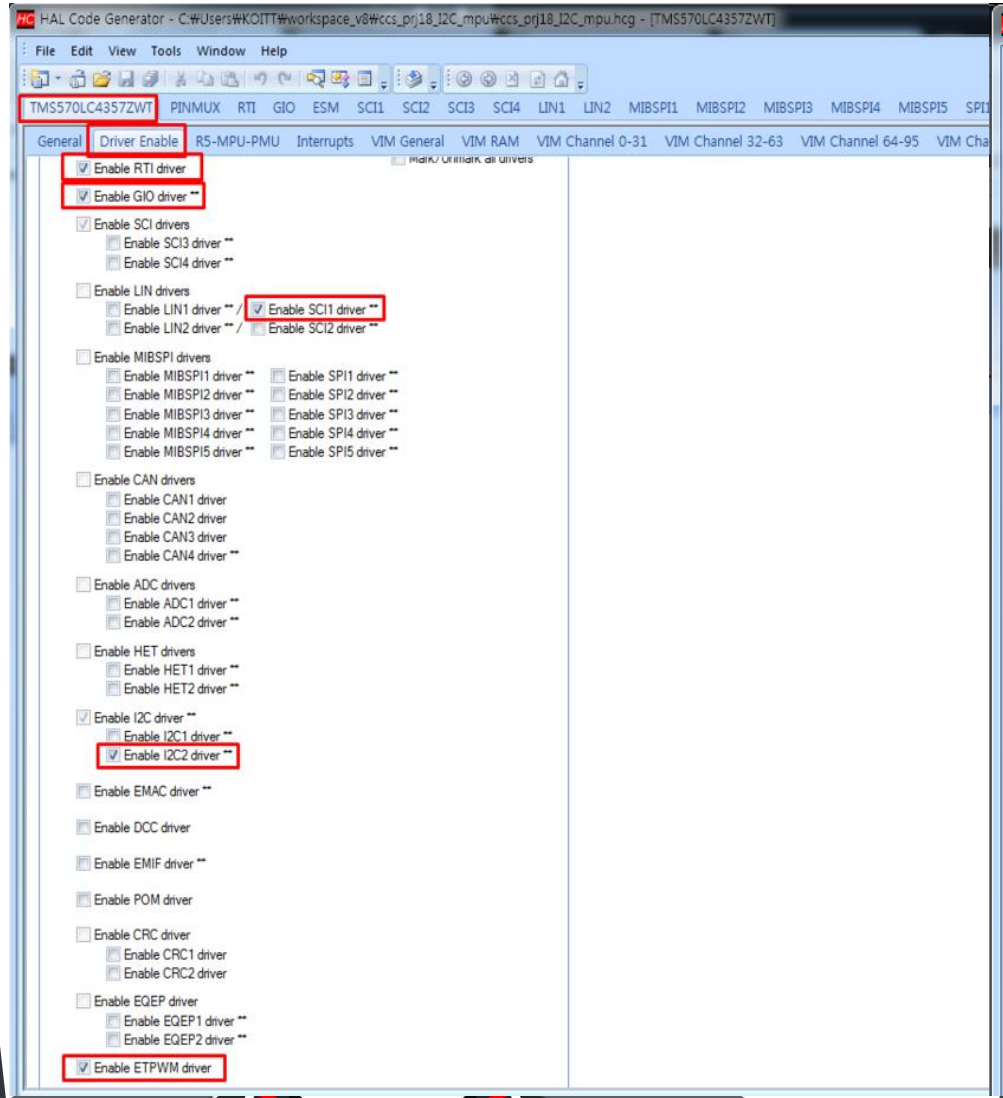
여기서 잠깐 SPI와 I2C 차이 설명

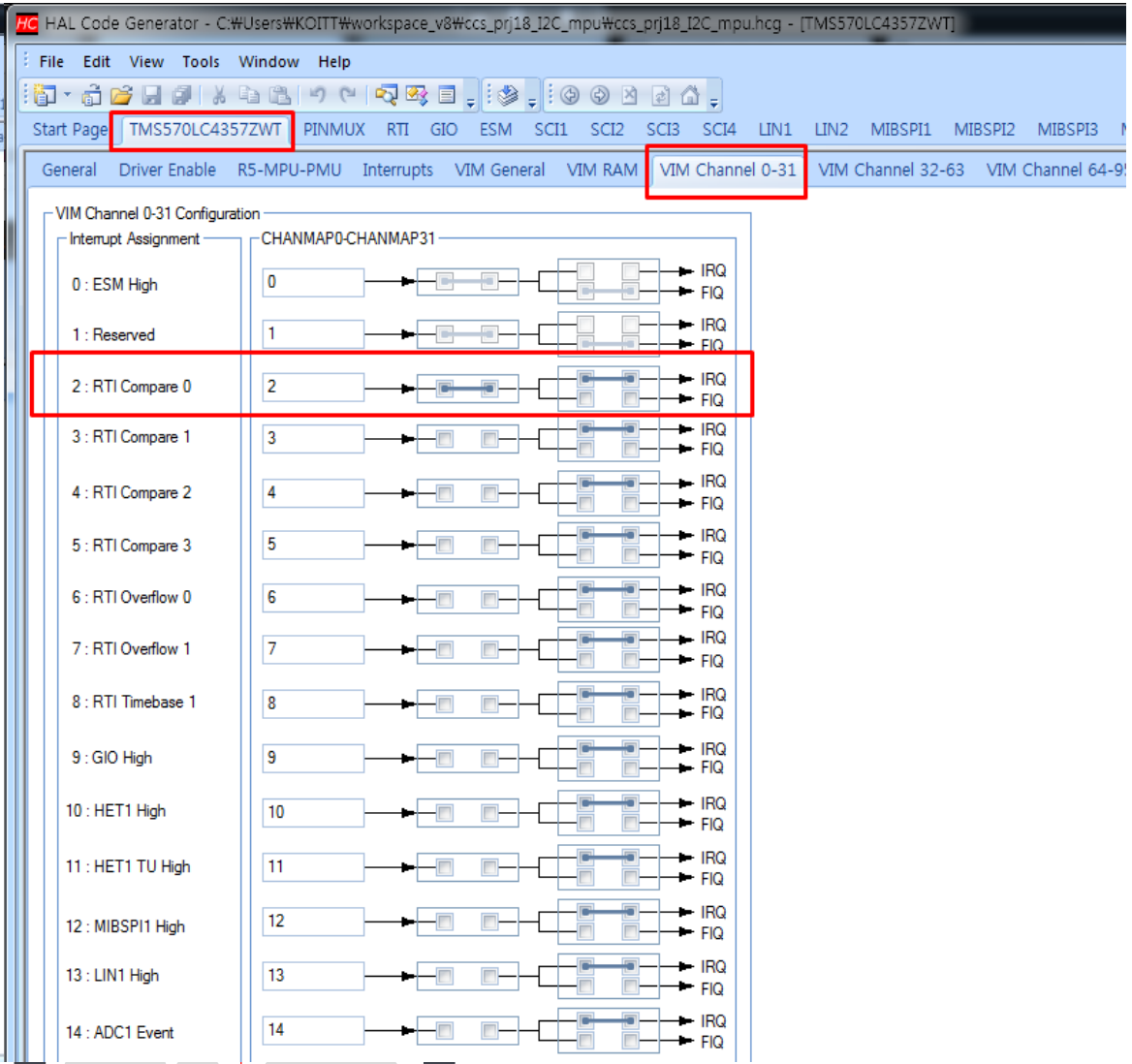
SPI: 동기식 직렬 통신 규격으로 4개의 선으로 이루어진 직렬버스라고도 하는데 하나의 마스터에서 여러개의 slave가 붙을수 있는 형식으로 되어 있고 이렇게 여러개 붙어있는 slave를 select하는 형식으로 통신이 이루어 지게 된다.full duplex방식이다.

<SPI 경우>
SCLK(serial clock_output from master) : 마스터로부터 클럭 출력됨
MOSI or SIMO(master output slave input): 마스터 출력, 슬레이브 입력 (마스터로부터의 출력)
MISO or SOMI(master input slave output): 마스터 입력, 슬레이브 출력 (슬레이브로부터의 출력)
SS(Slave select-active low): 만약 slave가 n개가 있다면 / ss1~ssn 까지를 두어 사용할 slave select만 low로 하여 사용.

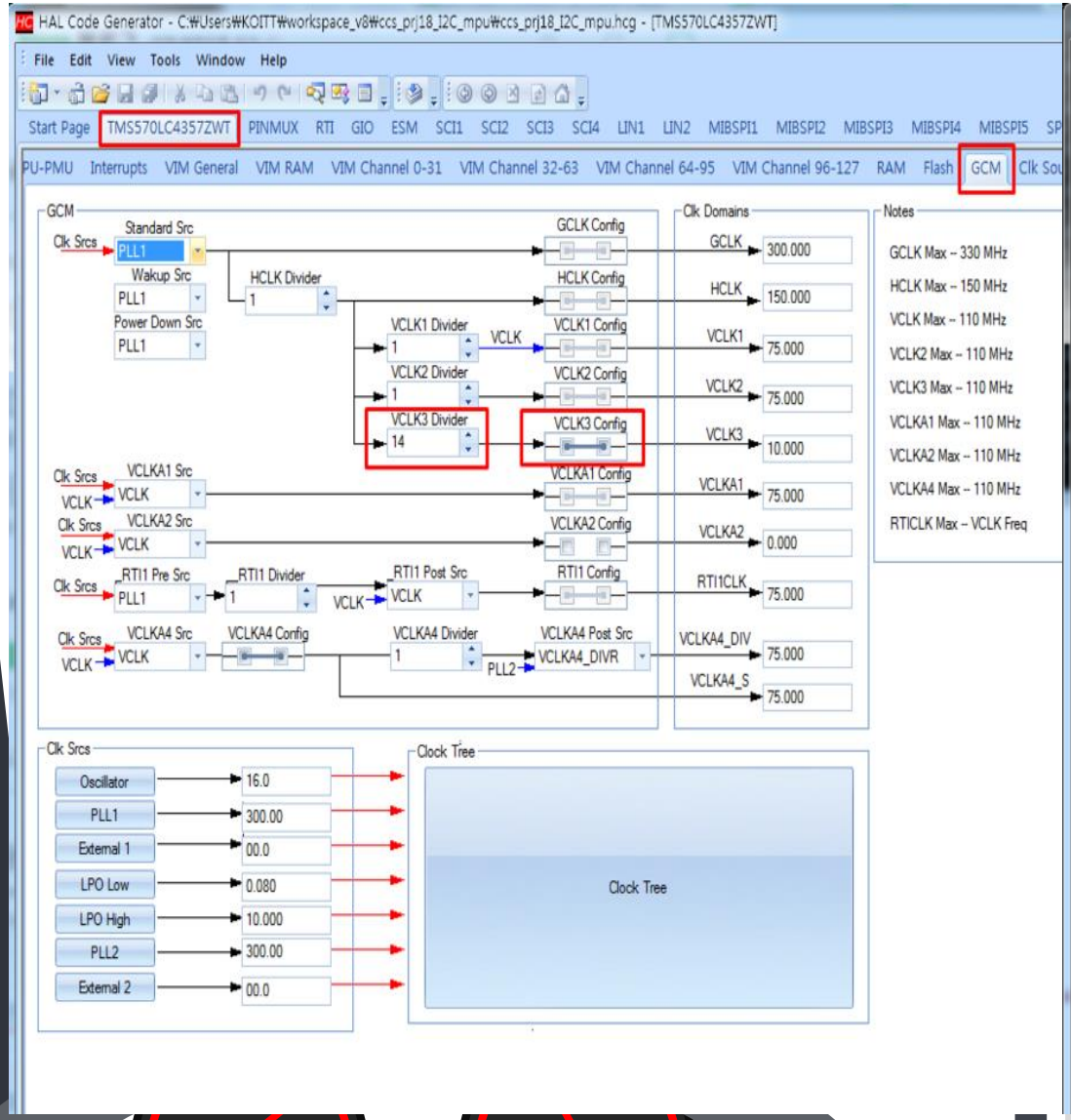
<I2C 경우>
SDA(Serial data) : 데이터 전송
SCL(Serial Clock : 클럭 전송

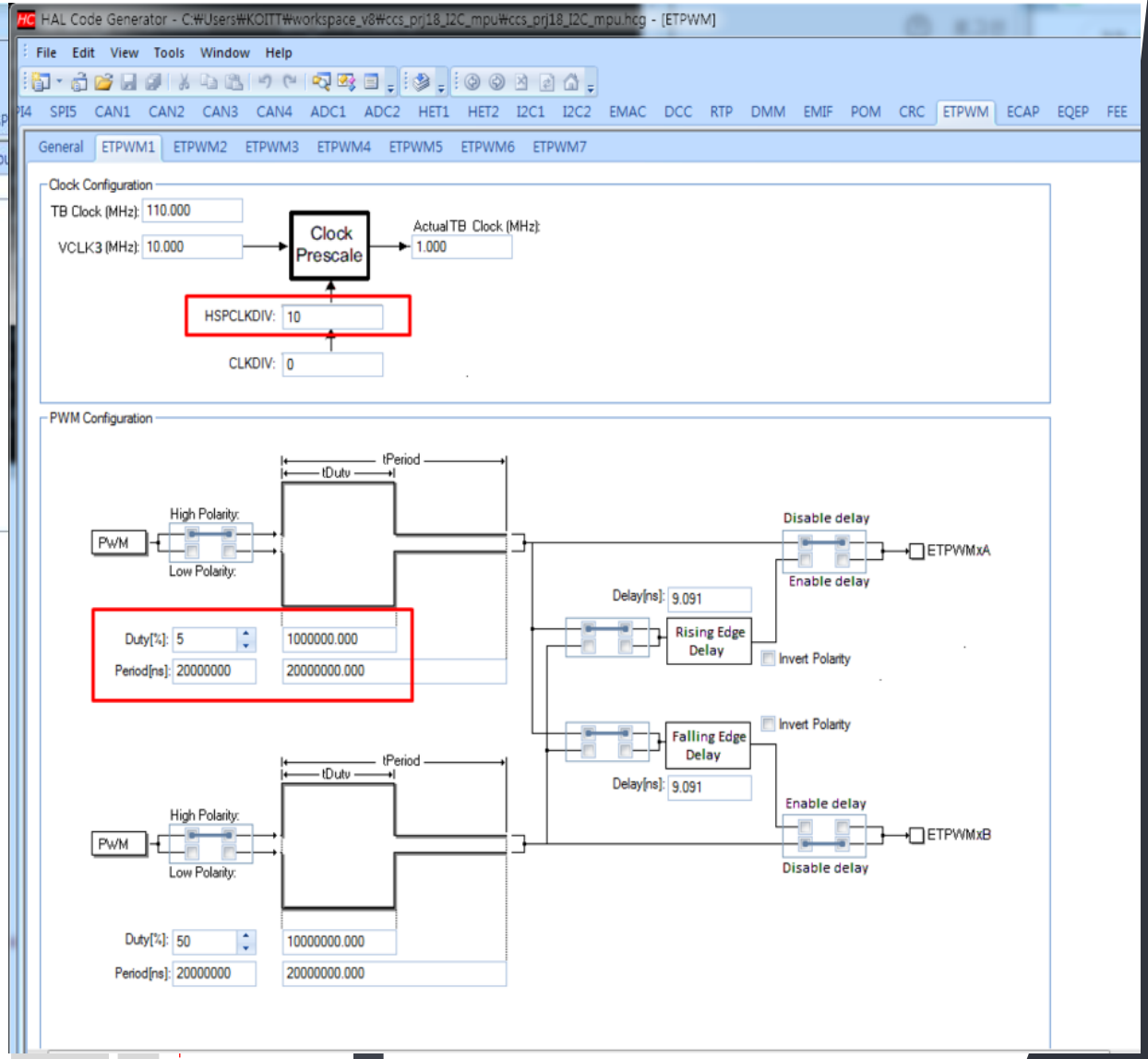
Halcogen 설정하기.



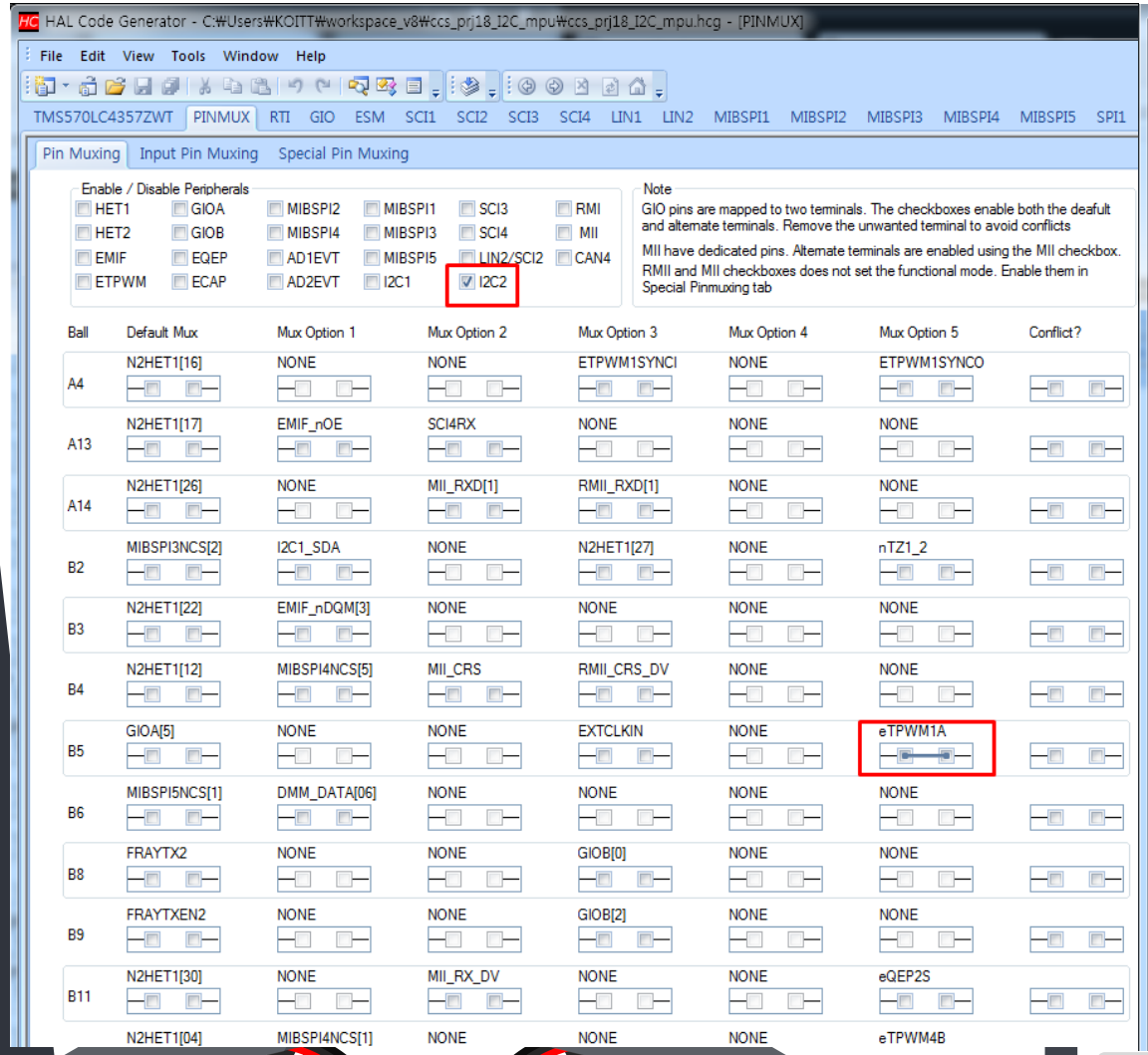


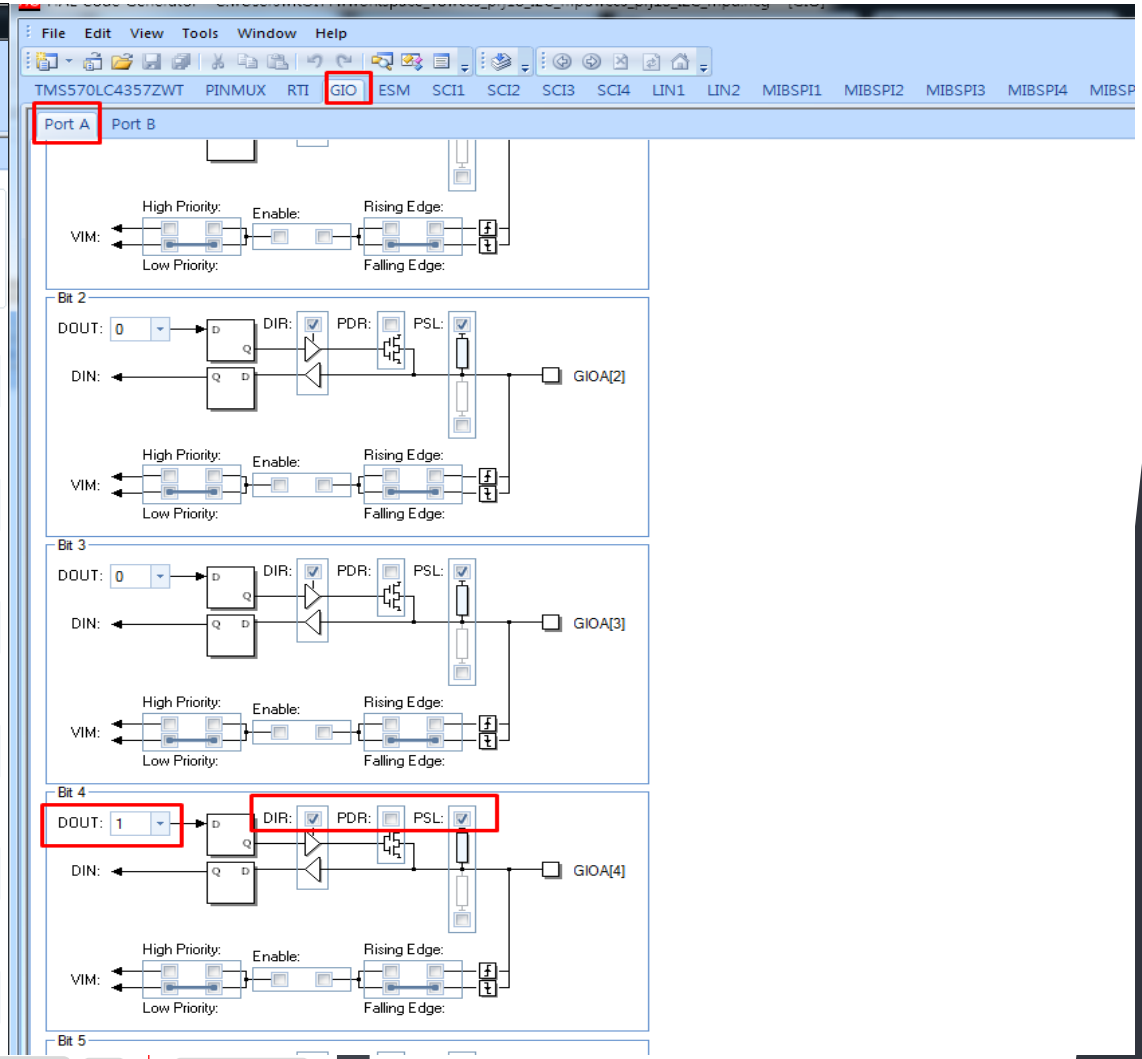
Halcogen 설정하기.



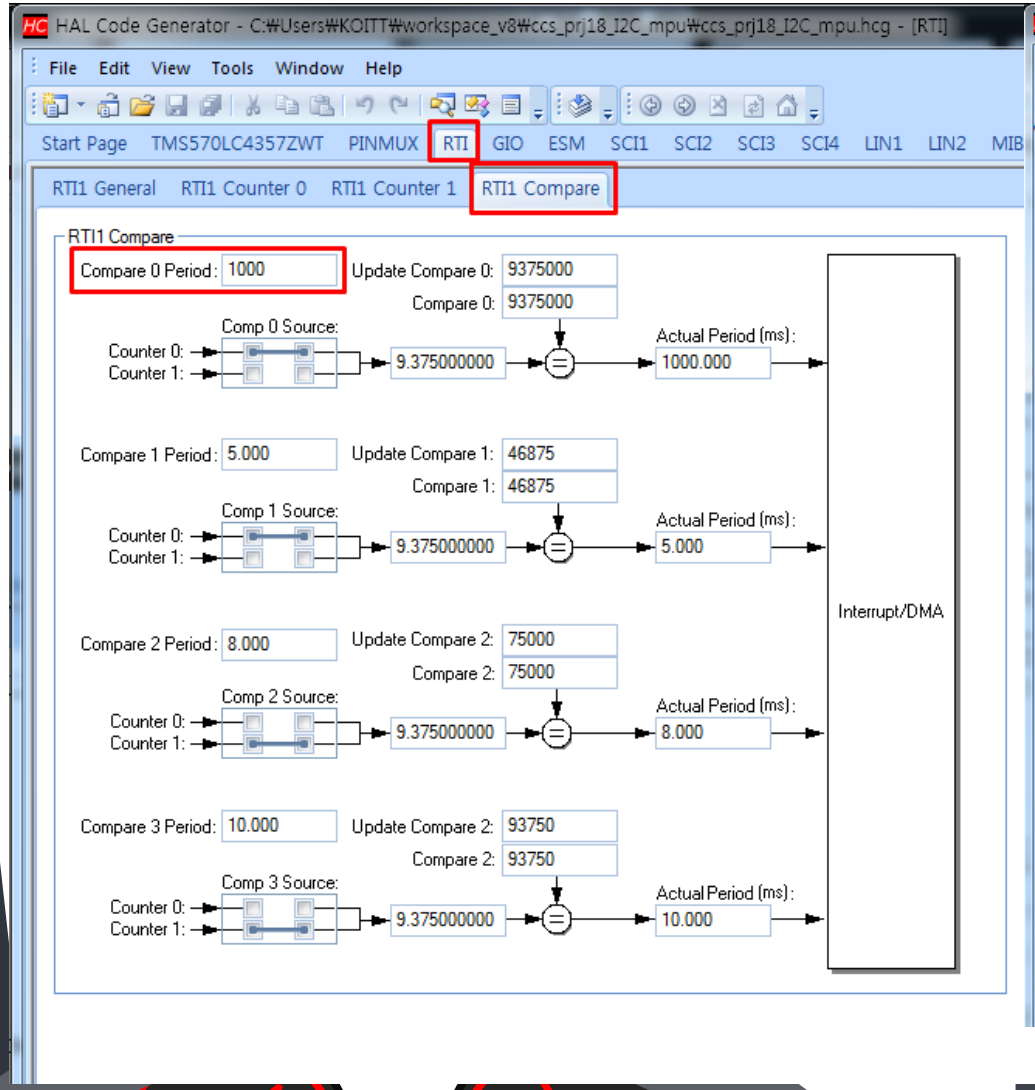


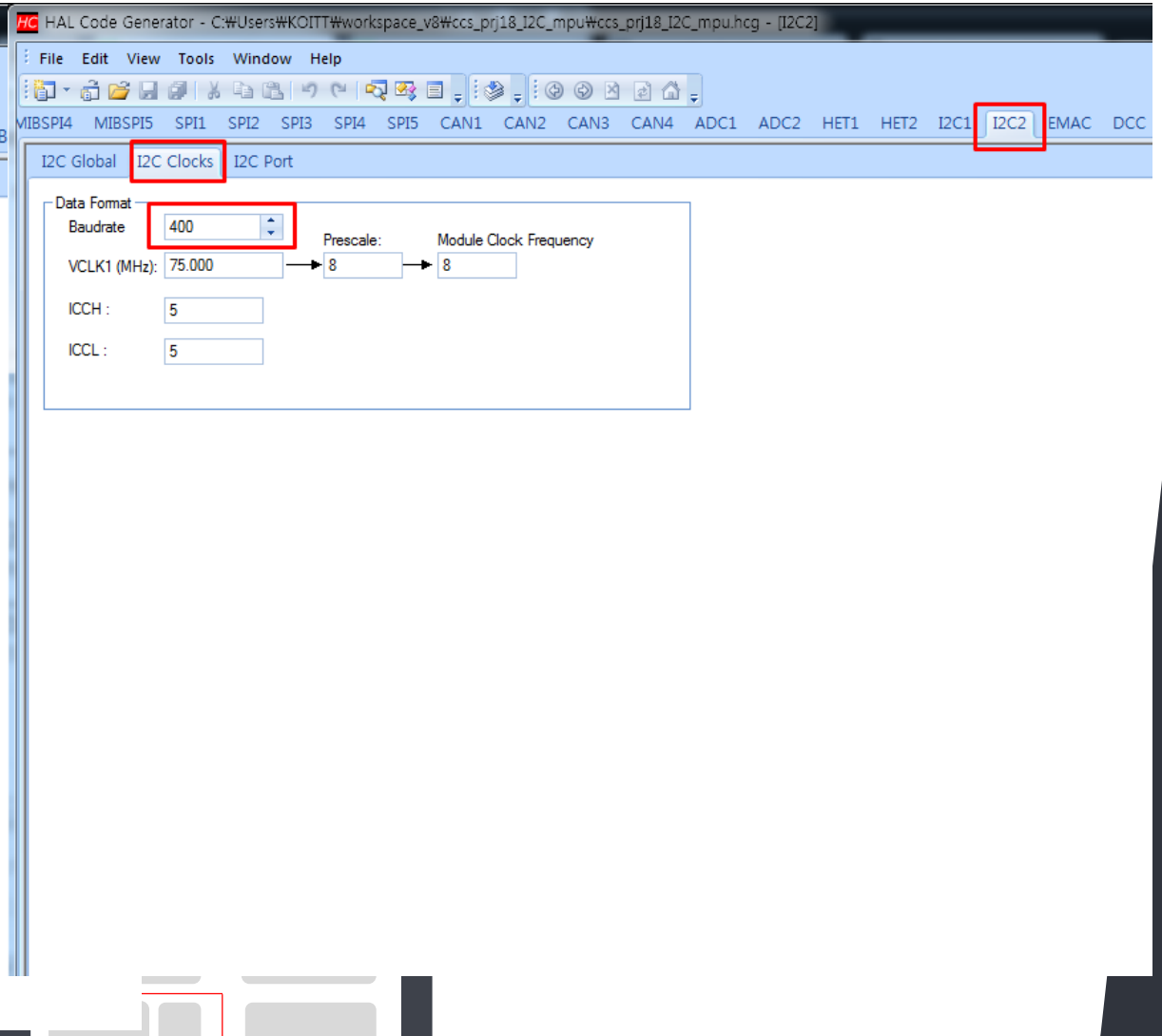
Halcogen 설정하기.



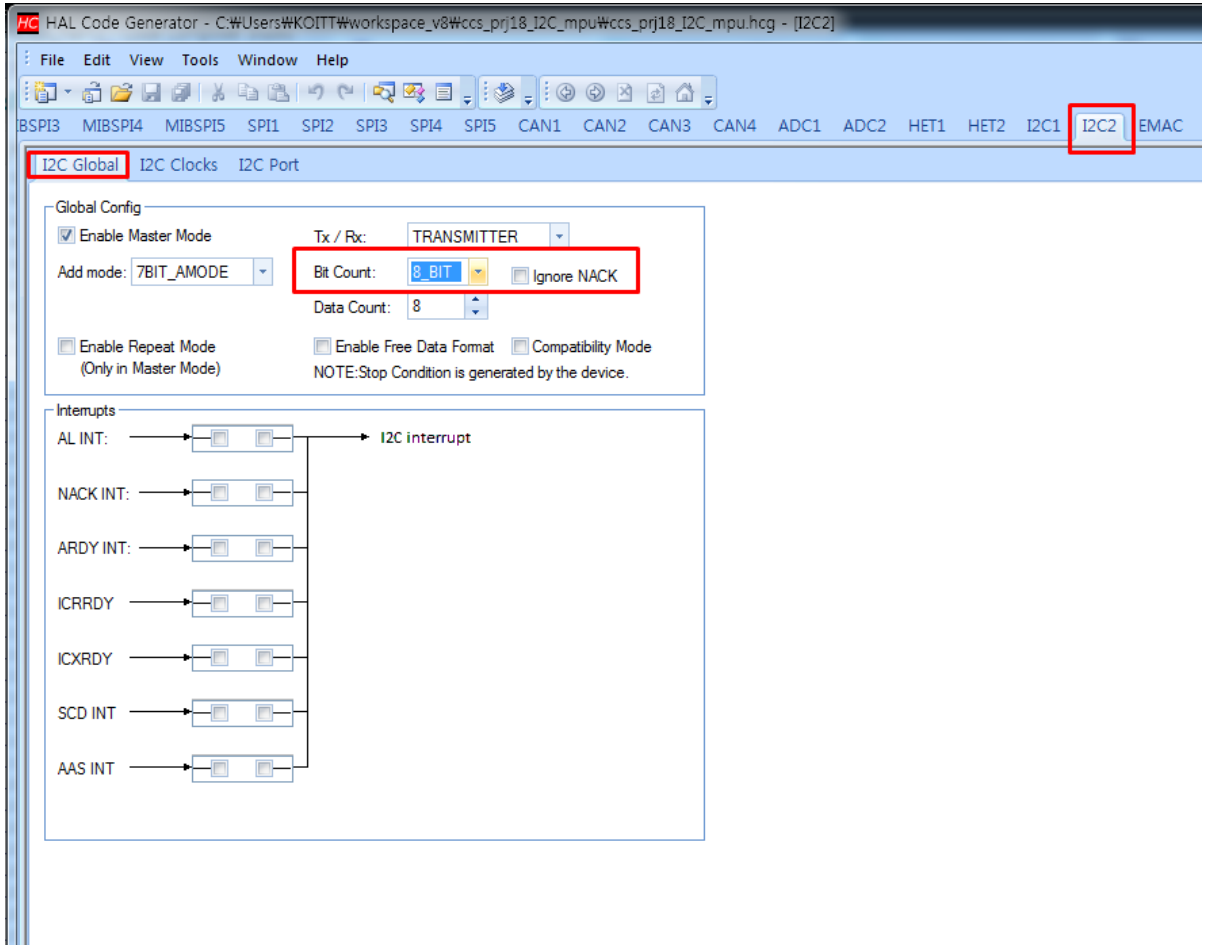


Halcogen 설정하기.





Halcogen 설정하기.



CCS 코딩하기 <가속도센서에 따른 모터 제어>

```c
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_etpwm.h"
#include "HL_sci.h"
#include "HL_gio.h"
#include "HL_i2c.h"
#include "HL_rti.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define UART sciREG1
#define MPU6050_ADDR 0x68

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len);
void pwmSet(void);
void wait(uint32 delay);
void MPU6050_enable(void);
void MPU6050_acc_config(void);
void disp_set(char *);
uint32 rx_data = 0;
uint32 tmp = 0;
uint32 value = 0;
char count_restart = 0;
volatile char g_acc_xyz[6];
volatile int g_acc_flag;
#define IDX 6
uint32 duty_arr[IDX] = { 1000, 1200, 1400, 1600, 1800, 2000 };  // 모터의 듀티를 조정할 함수.
int main(void) {
    char num_buf[256] = { 0 };
    char txt_buf[256] = { 0 };
    unsigned int buf_len;
    volatile int i;
    signed short acc_x, acc_y, acc_z;
    double real_acc_x, real_acc_y, real_acc_z;
    sciInit();  // sci의 기본을 setting해주는 함수.
    // disp_set: 글자를 출력해준다. sprintf를 최대한 쓰지 않아야 한다.
    disp_set("SCI Configuration Success!!\n\r\0");

    gioInit();  // gio의 기본 설정이 들어 있는 함수 이다. (할코젠에서 설정한 것이 들어있다.)
    disp_set("GIO Init Success!!\n\r\0");

    i2cInit();  // i2c의 기본 설정이 들어 있는 함수 이다. ( 할코젠에서 설장한 것이 들어있다.)
    wait(10000000);

    MPU6050_enable(); // mpu6050 을 연결하게 설정해주는 함수이다.
    disp_set("MPU6050 Enable Success!!\n\r\0");

    MPU6050_acc_config();
    disp_set("MPU6050 Accelerometer Configure Success!!\n\r\0");

    rtiInit();      // rti의 기본 설정이 들어 있는 함수 이다.
    rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);   //
    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    disp_set("RTI Init Success!!\n\r\0");

    etpwmInit(); // etPWM의 기본적인 설정이 들어 있는 함수 이다.
    disp_set("ETPWM Configuration Success!!\n\r\0");

    etpwmStartTBCLK(); // etPWM의 설정값에 맞게 파형을 만들어 시작하게 해준다.
    disp_set("ETPWM Start Success!!\n\r\0");

    _enable_IRQ_interrupt_();   // 인터럽트를 인에블 해주는 함수이다.
    disp_set("Interrupt enable Success!!\n\r\0");
    wait(1000000);
```

CCS 코딩하기 <가속도센서에 따른 모터 제어>

```
    for (;;) {
        if (g_acc_flag) {
            acc_x = acc_y = acc_z = 0;
            real_acc_x = real_acc_y = real_acc_z = 0.0;
            acc_x = g_acc_xyz[0];
            acc_x = acc_x << 8;
            acc_x |= g_acc_xyz[1];
            real_acc_x = ((double) acc_x) / 2048.0;
//          sciDisplayText(sciREG1, (uint8 *) "acc_x = %2.5lf \n\r\0", "acc_x = %2.5lf \n\r\0");

            acc_y = g_acc_xyz[2];
            acc_y = acc_y << 8;
            acc_y |= g_acc_xyz[3];
            real_acc_y = ((double) acc_y) / 2048.0;

            acc_z = g_acc_xyz[4];
            acc_z = acc_z << 8;
            acc_z |= g_acc_xyz[5];
            real_acc_z = ((double) acc_z) / 2048.0;

            sprintf(txt_buf, "acc_x = %2.5lf\tacc_y = %2.5lf\tacc_z = %2.5lf\n\r\0", real_acc_x, real_acc_y, real_acc_z);
            buf_len = strlen(txt_buf);
            sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);

            if (real_acc_x > 0) {
                rx_data = 1;
                //sprintf(txt_buf, "rx = %d\n\r\0", rx_data);
                //buf_len = strlen(txt_buf);
                //sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
                etpwmREG1->CMPA = duty_arr[rx_data];
                disp_set("acc_x == +\n\r\0");
                //sprintf(txt_buf, "PWM Duty = %d\n\r\0", value);
                //buf_len = strlen(txt_buf);
                //sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
            }
            else if (real_acc_x <= 0) {
                rx_data = 4;
                //sprintf(txt_buf, "rx = %d\n\r\0", rx_data);
                //buf_len = strlen(txt_buf);
                //sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
//              pwmSet();
                etpwmREG1->CMPA = duty_arr[rx_data];
                disp_set("acc_x == -\n\r\0");
                //sprintf(txt_buf, "PWM Duty = %d\n\r\0", value);
                //buf_len = strlen(txt_buf);
                //sciDisplayText(sciREG1, (uint8 *) txt_buf, buf_len);
            }
//
            g_acc_flag = 0;
        }
    }
```

CCS 코딩하기 <가속도센서에 따른 모터 제어>

```
#if 0
    for(;;)
    {
        tmp = sciReceiveByte(UART);
        rx_data = tmp - 48;
        sprintf(txt_buf, "rx = %d\n\r\0", rx_data);
        buf_len = strlen(txt_buf);
        sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
        pwmSet();
        sprintf(txt_buf, "PWM Duty = %d\n\r\0", value);
        buf_len = strlen(txt_buf);
        sciDisplayText(sciREG1, (uint8 *)txt_buf, buf_len);
    }
#endif
    return 0;
}
void pwmSet(void) {
    value = duty_arr[rx_data];
    etpwmSetCmpA(etpwmREG1, value);
    wait(10000);
}
void wait(uint32 delay) {
    int i;
    for (i = 0; i < delay; i++)
        ;
}
void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 len) {
    while (len--) {
        while ((UART->FLR & 0x4) == 4)
            ;
        sciSendByte(UART, *text++);
    }
}
```

CCS 코딩하기 <가속도센서에 따른 모터 제어>

```
void MPU6050_enable(void) {
    volatile unsigned int cnt = 2;
    unsigned char data[2] = { 0x00U, 0x00U };
    unsigned char slave_word_address = 0x6bU;    // slave_word_address = 0x6b일 때 write 모드이다.
    restart1_1: restart1_2:

    i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR);       // MPU6050 address =    0x68 일 때 reed 모드 이다.
                                                 // i2c 버스에서 통신할 슬레이브 장치의 주소를 지정해준다.
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);   // 송수신 모드 설정, 전송모드로 설정.
    i2cSetCount(i2cREG2, cnt + 1);               // cnt+1 만큼 count 하고 stop condition 을 생성하여 중지한다.
    i2cSetMode(i2cREG2, I2C_MASTER);             // 마스터 슬레이브 모드 설정, i2cREG2 레지스터를 마스터 모드로 설정한다.
    i2cSetStop(i2cREG2);                         // stop condition을 생성해서 통신을 정지 하는 함수.
    i2cSetStart(i2cREG2);                        // start condition 을 생성해서 통신을 시작하는 함수.
    i2cSendByte(i2cREG2, slave_word_address);    // i2cReceive 데이터 블록을 단위로 전송하는 함수.
    i2cSend(i2cREG2, cnt, data);                 // i2cReceive 데이터 블록을 전송하는 함수. ack 받기 위해 보냄.
    wait(100000);
    while (i2cIsBusBusy(i2cREG2) == true)        // bus가 사용 중일 때 참으로 무한루프를 돌게 된다. bus가 사용중이 아닐 때,빠져나온다.
        if ((count_restart++) == 30)
            goto restart1_1;
    count_restart = 0;
    while (i2cIsStopDetected(i2cREG2) == 0)      // stop condition 을 확인하는 것으로 SCD가 없다면 0으로 stop을 송수신 하지 않는다.
                                                 // 즉, 계속 통신하면서 무한 루프를 돈다는 것, stop condition이 설정되면 통신을 중단하고 빠져나온
        if ((count_restart++) == 30)
            goto restart1_2;
    count_restart = 0;
    wait(100000);
    i2cClearSCD(i2cREG2);                        // 위에서 set된 SCD플래그를 클리어 해주는 함수. 즉 stop 클리어.
    wait(100000);
}
void MPU6050_acc_config(void) {
    volatile unsigned int cnt = 1;
    unsigned char data[1] = { 0x18U };
    unsigned char slave_word_address = 0x1cU;
    restart2_1: restart2_2: i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR); // i2c 버스에서 통신할 슬레이브 장치의 주소를 지정해준다.
    i2cSetDirection(i2cREG2, I2C_TRANSMITTER);   // 송수신 모드 설정, 전송모드로 설정.
    i2cSetCount(i2cREG2, cnt + 1);               // cnt+1 만큼 count 하고 stop condition 을 생성하여 중지한다.
    i2cSetMode(i2cREG2, I2C_MASTER);             // 마스터 슬레이브 모드 설정, i2cREG2 레지스터를 마스터 모드로 설정한다.
    i2cSetStop(i2cREG2);                         // stop condition을 생성해서 통신을 정지 하는 함수.
    i2cSetStart(i2cREG2);                        // start condition 을 생성해서 통신을 시작하는 함수.
    i2cSendByte(i2cREG2, slave_word_address);    // i2cReceive 데이터 블록을 단위로 전송하는 함수. write 모드로 설정/
    i2cSend(i2cREG2, cnt, data);                 // i2cReceive 데이터 블록을 전송하는 함수.
    while (i2cIsBusBusy(i2cREG2) == true)        // 처음 셋팅중 안될경우를 생각해서 goto 문을 사용한 오류 처리를 하였다.
        if ((count_restart++) == 30)             // 동작중인 경우 계속 돌고 한가해 지면 빠져나온다.
            goto restart2_1;
    count_restart = 0;
    while (i2cIsStopDetected(i2cREG2) == 0)      // 통신이 스탑이 감지되면 빠져나오는 함수 이다.
        if ((count_restart++) == 30)
            goto restart2_2;
    count_restart = 0;
    i2cClearSCD(i2cREG2);                        // SCD 플래그를 클리어 하고 마무리 한다.
    wait(1000000);
}
```

CCS 코딩하기 <가속도센서에 따른 모터 제어>

```
void rtiNotification(rtiBASE_t *rtiREG, uint32 notification) {
    if (notification == 1U) {
        unsigned char slave_word_address = 0x3B;

        i2cSetSlaveAdd(i2cREG2, MPU6050_ADDR);
        i2cSetDirection(i2cREG2, I2C_TRANSMITTER);
        i2cSetCount(i2cREG2, 1);
        i2cSetMode(i2cREG2, I2C_MASTER);
        i2cSetStop(i2cREG2);
        i2cSetStart(i2cREG2);
        i2cSendByte(i2cREG2, slave_word_address);
        while (i2cIsBusBusy(i2cREG2) == true)
            ;
        while (i2cIsStopDetected(i2cREG2) == 0)
            ;
        i2cClearSCD(i2cREG2);
        i2cSetDirection(i2cREG2, I2C_RECEIVER);      // 송수신 모드 설정, 수신모드로 설정.
        i2cSetCount(i2cREG2, 6);                     // count를 6까지 해준다. 그리고 stop된다.
        i2cSetMode(i2cREG2, I2C_MASTER);             // 모드를 다시 마스터 모드로 셋팅한다.
        i2cSetStart(i2cREG2);                        // 전송을 시작한다.
        i2cReceive(i2cREG2, 6, (unsigned char *) g_acc_xyz);    // 데이터 버퍼에(g_acc_xyz)수신 값을 6바이트 수신한다.
        i2cSetStop(i2cREG2);                         // 전송을 중지 한다.
        while (i2cIsBusBusy(i2cREG2) == true)
            ;
        while (i2cIsStopDetected(i2cREG2) == 0)
            ;
        i2cClearSCD(i2cREG2);
        g_acc_flag = 1;
    }
}
void disp_set(char *str) {
    unsigned int buf_len;
    buf_len = strlen(str);
    sciDisplayText(sciREG1, (uint8 *) str, buf_len);
//  wait(1000);
}
```

유자차

들어주셔서 감사합니다.