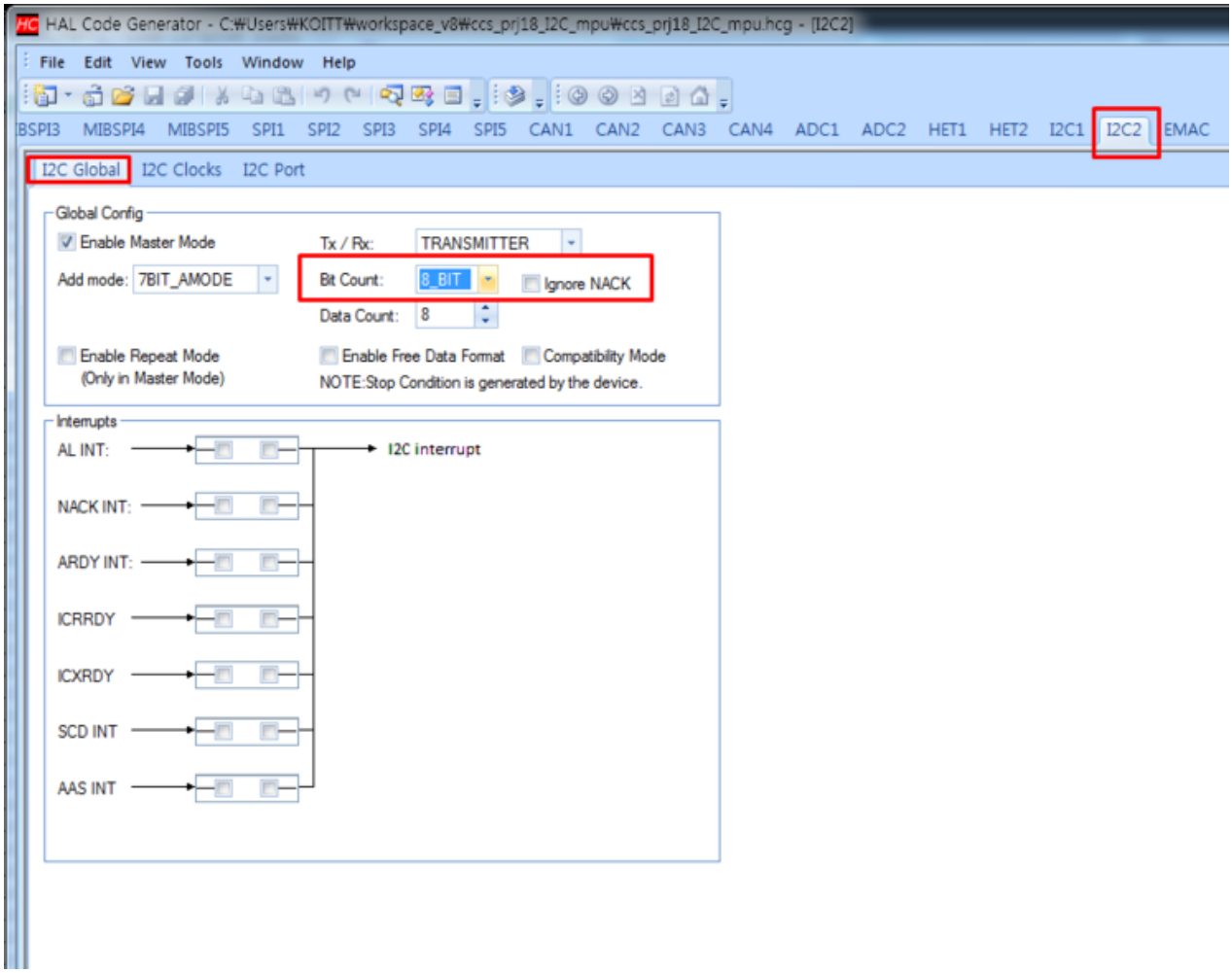

<code 1 _ 가속도센서에 따른 모터 제어>

```
#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);
    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_
            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;
        }
    }
#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++);
    }
}
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,빠져나
        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 않
                                                // 즉, 계속 통신하면서 무한 루프를 돈다는 것, stop condition이 설정되면 통신을
        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);
}
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);
}
```