

TI DSP, MCU 및 Xilinx Zynq FPGA

프로그래밍 전문가 과정

2018-05-15 (54회차)

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

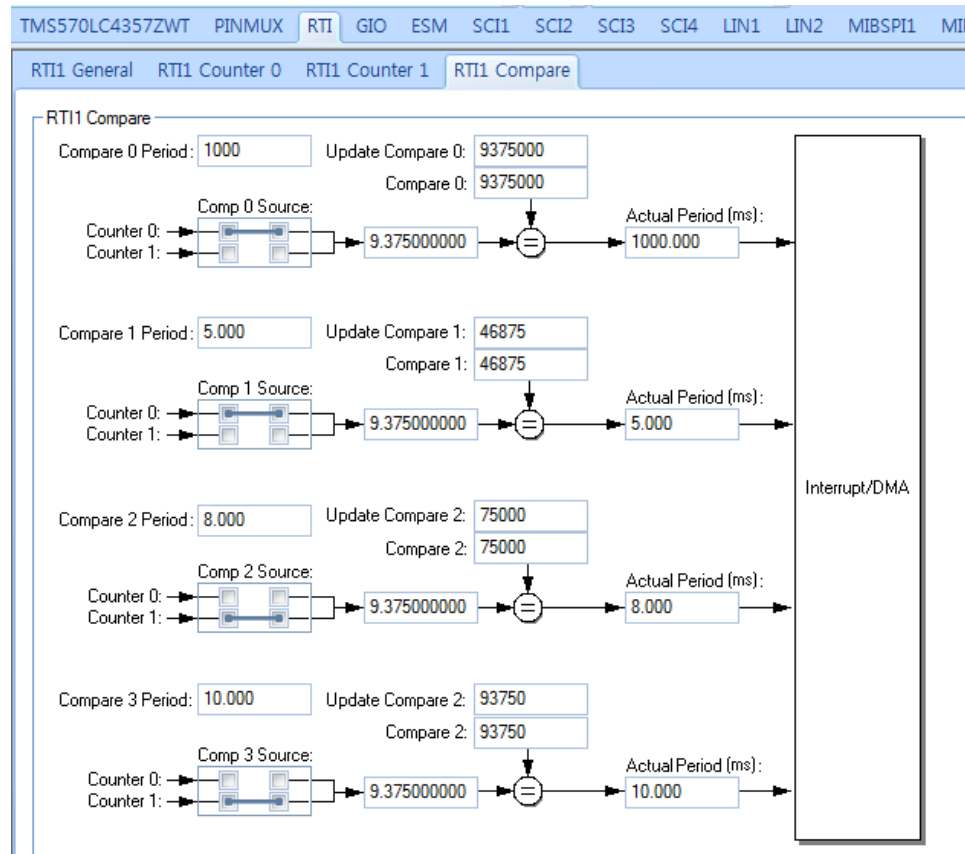
학생 - 정유경

ucong@naver.com

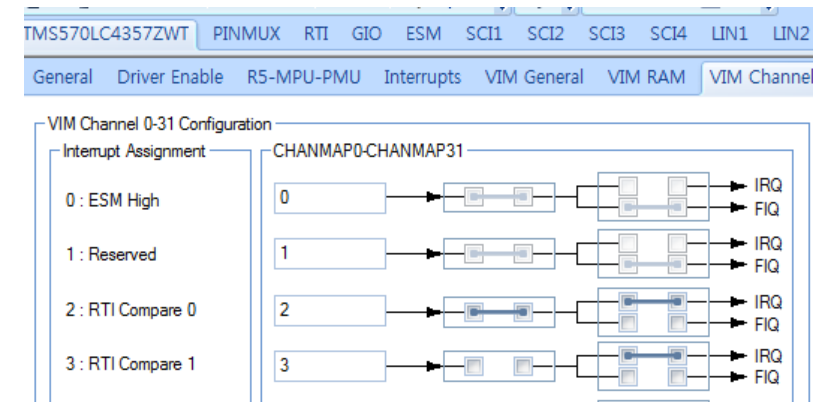
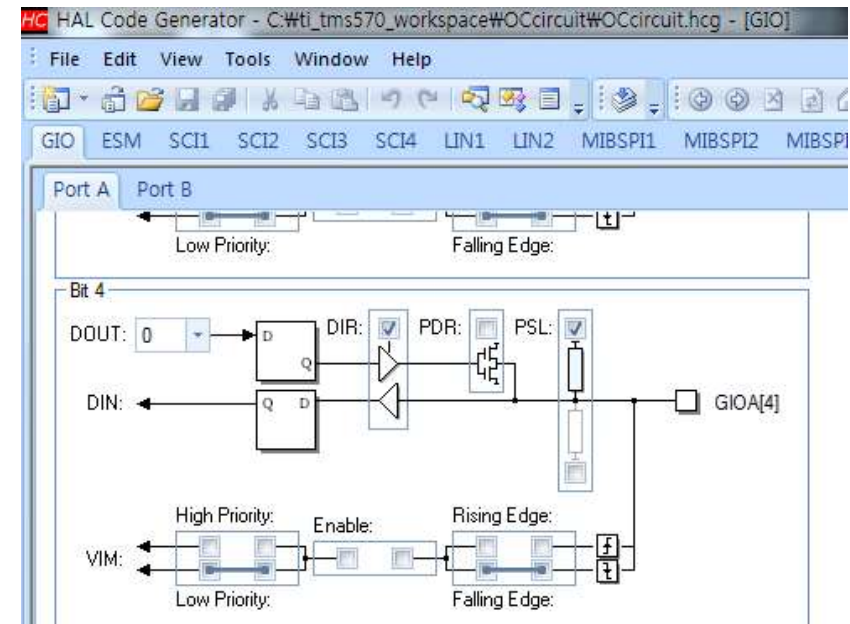
1. OC_circuit

Driver 설정: RTI, GIO

1초마다 온오프



설정을 마친 후 Generate Code



코드작성

(RTI_BLINK를 그대로 쓴다, Het Port GIO PORTA로 만든다)

```
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_sys_core.h"
#include "HL_mibspi.h"
#include "HL_esm.h"
#include "HL_rti.h"
#include "HL_gio.h"
#include "HL_het.h"
void main(void)
{
    rtiInit();
    gioInit();

    gioSetDirection(gioPORTA, 0x00000001);
    rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
    _enable_IRQ_interrupt();
    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

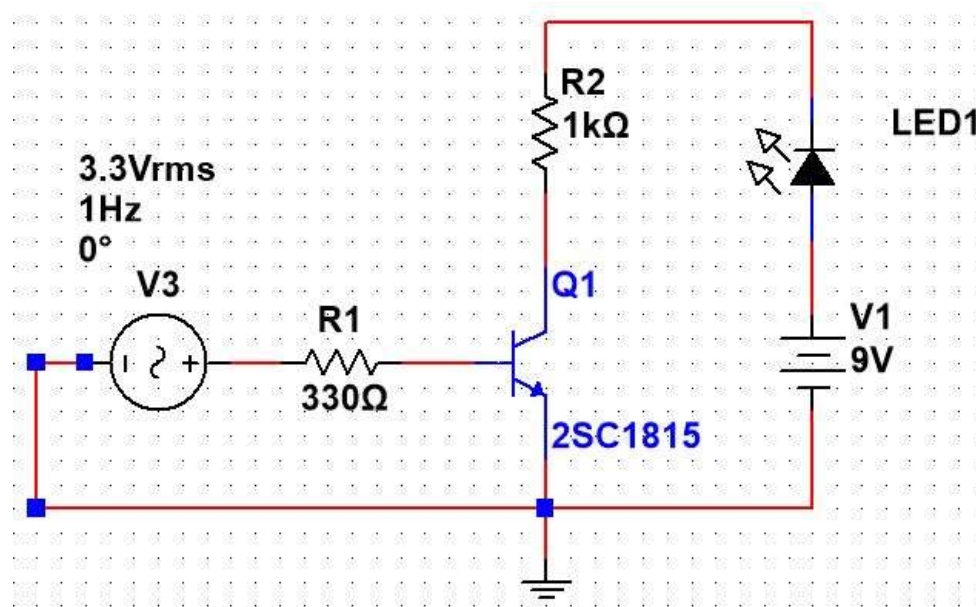
    srand(time(NULL));

    while(1);
}

void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
    gioSetPort(gioPORTA, gioGetPort(gioPORTA) ^ 0x00000001);
}
```

오픈컬렉터 회로 이해하기

Reference: <http://cafe.naver.com/hestit/2362>



선생님 팁

V3 에 보이는 교류 전압을 직류 전압으로 걸면 LED 는 항상 켜진다. 위와 같이 교류로 만든 상태에서 시뮬레이션을 해서 상태를 확인해보자.

LED 가 꺼졌다 켜졌다 하는 것을 볼 수 있을 것이다. V3 가 나오는 곳을 MCU 의 GPIO 핀이라고 생각하고

9V 가 걸리는 부분은 우리가 별도로 만든 전원 회로라고 생각하면 이해하기가 쉬워질 것이다.

오픈컬렉터를 사용하는 이유

1. 오픈컬렉터 없으면 항상 방전되어 소비전력이 높아진다. 하지만, 오픈 컬렉터 방식은 입력이 들어오는 그때에만 동작하게 할 수 있다.

즉, 센서가 감지하면 오픈 컬렉터로 구성하면 자동으로 원하는 시점에 트리거 해줄 수 있다.

2. 두개의 전원을 분리시킴, 보호회로의 역할

*. 출력(컬렉터)쪽 회로가 복잡할 때 TR의 전류가 부족하여 제대로 동작이 안될 수가 있다. 증폭을 더 시키거나 대책이 필요하다.

선생님 팁

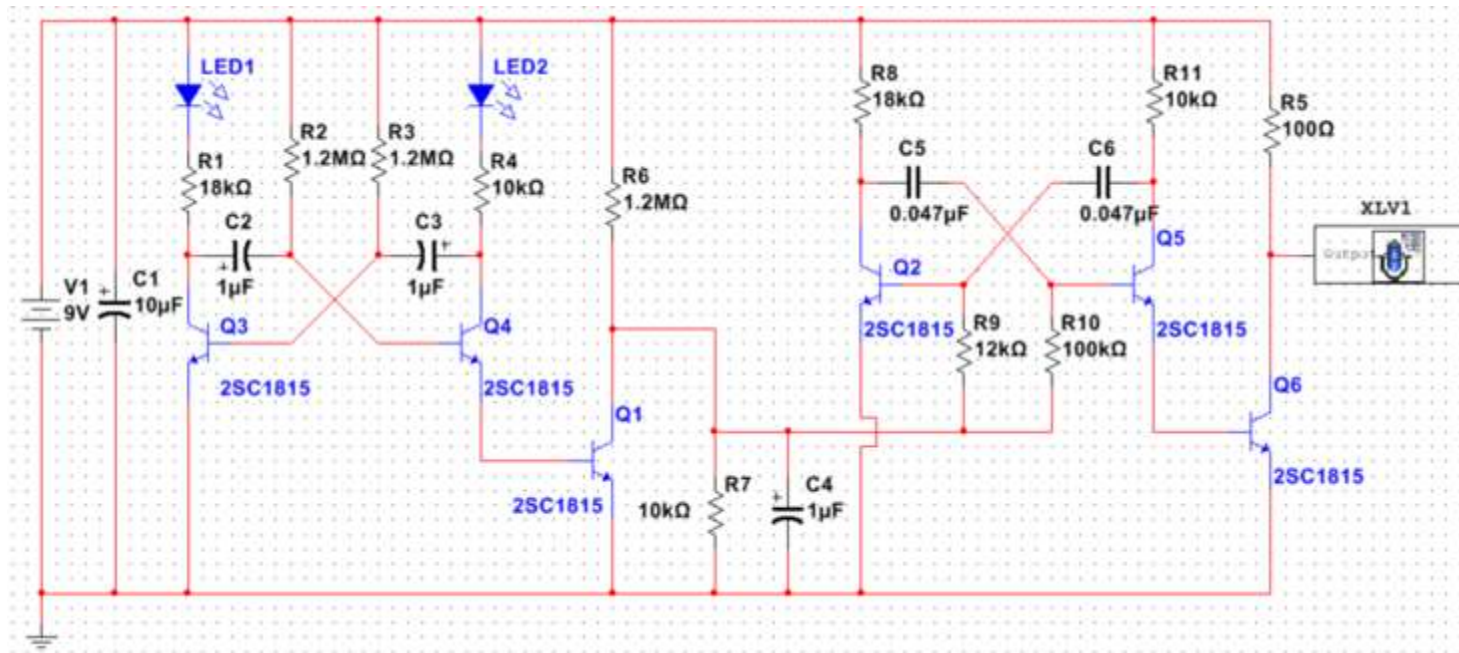
지금은 UART 로 트리거를 주고 있지만 특정 상황이 발생하면 알아서 동작하게 해주는 작업이 가능하다는 것이다.

대표적으로 현재 진행하는 프로젝트의 전투함에서 레이더(거리값을 가지고)가 적함을 발견하면 자동으로 트리거가 발생하여 GPIO 핀을 올리면 사이렌이 울리게 되는 것이다.

오픈컬렉터를 이용한 사이렌회로

Reference : <http://cafe.naver.com/hestit/2792>

사이렌 회로를 만들어보자(센서로 감지해서 거리가 가까우면 충돌한다는 사이렌을 울리는데 사용한다)



2. ADC_UART

General Driver Enable R5-MPU-PMU Interrupts VIM General VIM RAM VIM C

Enable Driver Compilation

Click and mark the required modules for driver compilation from below:

☐ Mark/Unmark all drivers

☐ Enable RTI driver

☒ Enable GPIO driver **

☒ Enable SCI drivers

☐ Enable SCI3 driver **

☐ Enable SCI4 driver **

☐ Enable LIN drivers

☐ Enable LIN1 driver ** / ☒ Enable SCI1 driver **

☐ Enable LIN2 driver ** / ☐ Enable SCI2 driver **

☐ Enable MIBSPI drivers

☐ Enable MIBSPI1 driver ** ☐ Enable SPI1 driver **

☐ Enable MIBSPI2 driver ** ☐ Enable SPI2 driver **

☐ Enable MIBSPI3 driver ** ☐ Enable SPI3 driver **

☐ Enable MIBSPI4 driver ** ☐ Enable SPI4 driver **

☐ Enable MIBSPI5 driver ** ☐ Enable SPI5 driver **

☐ Enable CAN drivers

☐ Enable CAN1 driver

☐ Enable CAN2 driver

☐ Enable CAN3 driver

☐ Enable CAN4 driver **

☒ Enable ADC drivers

☒ Enable ADC1 driver **

☐ Enable ADC2 driver **

☐ Enable HET drivers

☐ Enable HET1 driver **

☐ Enable HET2 driver **

사이클타임은 샘플링 주기

TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1

ADC1 General ADC1 Group Event ADC1 Group 1 ADC1 Group 2 ADC1 Memory

ADC1 Configuration

Cycle Time (ns): 100.00

Prescale: 7

Actual Cycle Time (ns): 106.67

VCLK1 (MHz): 75.0

☐ RAM Parity enable

TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2

ADC1 General ADC1 Group Event ADC1 Group 1 ADC1 Group 2 ADC1 Memory ADC1

ADC1 Group 1 Configuration

FiFo Size: 16

Data Resolution (Bit): 12_BIT

☒ Enable Channel Id in Conversion Results

☐ Enable Continuous Conversion

ADC1 Group 1 Trigger

Default Trigger: GIOB0

Alternate Trigger: EVENT

Rising Edge

Falling Edge

SW Trigger

Hardware

Software

Trigger

ADC1 Group 1 Sampling

Start: tScan tDischarge tSample tConversion End:

☐ Enable Sampling Capacitor Discharge

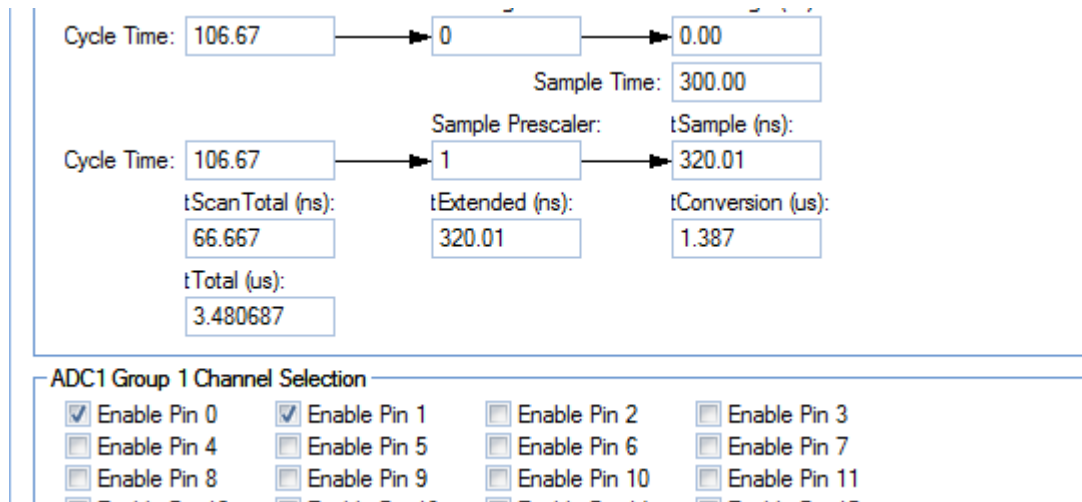
Discharge Time: 0.00

Discharge Prescaler: 0

tDischarge (ns): 0.00

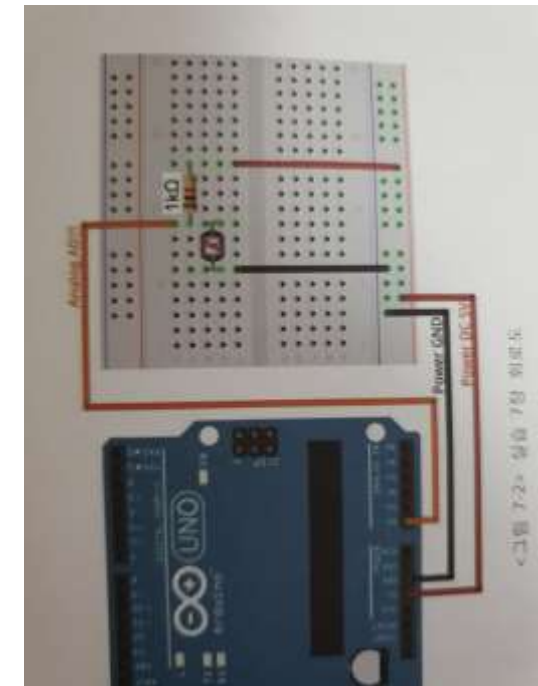
Cycle Time: 106.67

Sample Time: 300.00



32개 ADC 두개이므로 총 64개 쓸 수 있다

오른쪽 그림과 같이 회로를 구성한다



코드를 작성한다.

```
#include "HL_sys_common.h"
#include "HL_system.h"

#include "HL_sci.h"
#include "HL_esm.h"
#include "HL_adc.h"
#include "HL_gio.h"

#define TSIZE1 12
uint8 TEXT1[TSIZE1]={'\r', '\n', '|', '\t', 'C', 'H', '.', 'I', 'D', '-', '0', 'x'};
#define TSIZE2 9
uint8 TEXT2[TSIZE2]={'\t', 'V', 'A', 'L', 'U', 'E', '=', '0', 'x'};

adcData_t adc_data[2];

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 length);
void sciDisplayData(sciBASE_t *sci, uint8 *text, uint32 length);
void wait(uint32 time);

int main(void)
{
    uint32 ch_count =0;
    uint32 id =0;
    uint32 value = 0;

    gioInit();
    gioSetDirection(gioPORTB, 0xFF);

    sciInit();
    adcInit();
    adcStartConversion(adcREG1, adcGROUP1);

    while(1){
        gioSetBit(gioPORTB, 0, 1);
        while((adcIsConversionComplete(adcREG1, adcGROUP1))==0)
```

```

        ;
        ch_count = adcGetData(adcREG1, adcGROUP1, &adc_data[0]);
        id = adc_data[0].id;
        value = adc_data[0].value;
        gpioSetBit(gioPORTB, 0, 0);

        sciDisplayText(sciREG1, &TEXT1[0], TSIZE1);
        sciDisplayData(sciREG1, (uint8 *)&id, 4);
        sciDisplayText(sciREG1, &TEXT2[0], TSIZE2);
        sciDisplayData(sciREG1, (uint8 *)&value, 4);

        if(value > 0xEA0){
            gpioSetBit(gioPORTB, 4, 1);
        }
        else{
            gpioSetBit(gioPORTB, 4, 0);
        }

        wait(0xFFFF);
    }
}

void sciDisplayText(sciBASE_t *sci, uint8 *text, uint32 length)
{
    while(length--)
    {
        while((sciREG1->FLR & 0x4) == 4)
        ;
        sciSendByte(sciREG1, *text++);
    }
}

```

```

void sciDisplayData(sciBASE_t *sci, uint8 *text, uint32 length){

    uint8 txt =0;
    uint8 txt1 =0;

    #if ((__little_endian__ == 1) || (__LITTLE_ENDIAN__ == 1))

        text = text + (length -1);

    #endif

        while(length--){

    #if ((__little_endian__ == 1) || (__LITTLE_ENDIAN__ == 1))

        txt = *text--;

    #else

        txt = *text++;

    #endif

        txt1 = txt;
        txt &= ~(0xF0);
        txt1 &= ~(0x0F);
        txt1 = txt1>>4;

        if(txt <= 0x9){
            txt +=0x30;
        }

        else if(txt > 0x9 && txt < 0xF){
            txt +=0x37;
        }

        else{

```

```

        txt = 0x30;
    }

    if(txt1 <=0x9){
        txt1 += 0x30;
    }

    else if((txt1 > 0x9) && (txt1 <= 0xF)){
        txt1 += 0x37;
    }

    else{
        txt1 = 0x30;
    }

    while((sciREG1->FLR & 0x4)==4)
        ;

    sciSendByte(sciREG1, txt1);

    while((sciREG1->FLR & 0x4)==4)
        ;

    sciSendByte(sciREG1, txt);
    }

}

void wait(uint32 time){
    int i;
    for(i=0; i<time; i++);
}

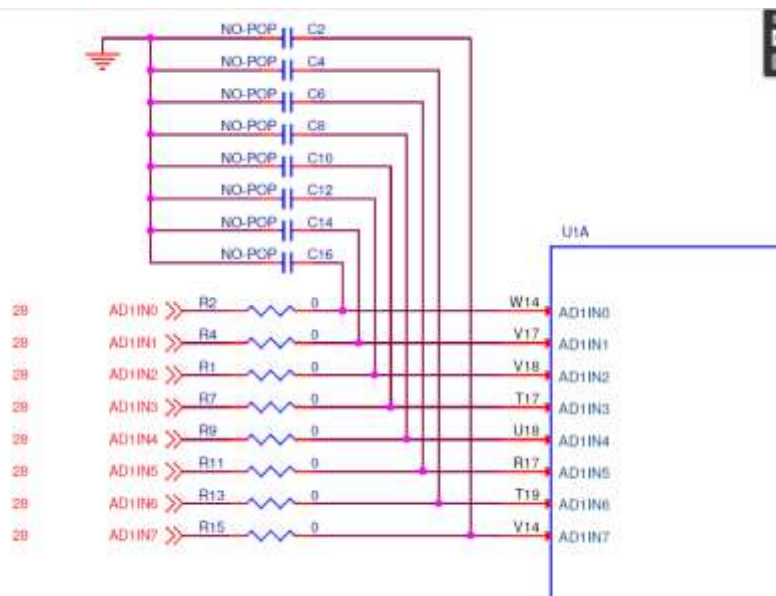
```

소프트웨어 필터 vs 하드웨어 필터

하드웨어 필터보다 소프트웨어 필터가 강력하다.

(필터: 사각파를 사인파로 만들 수 있다.

사각파는 여러 개의 고주파로 이루어져 있다. 여러 고주파들 중에서 특정 주파수의 파형을 필터링 해낼 수 있다.)

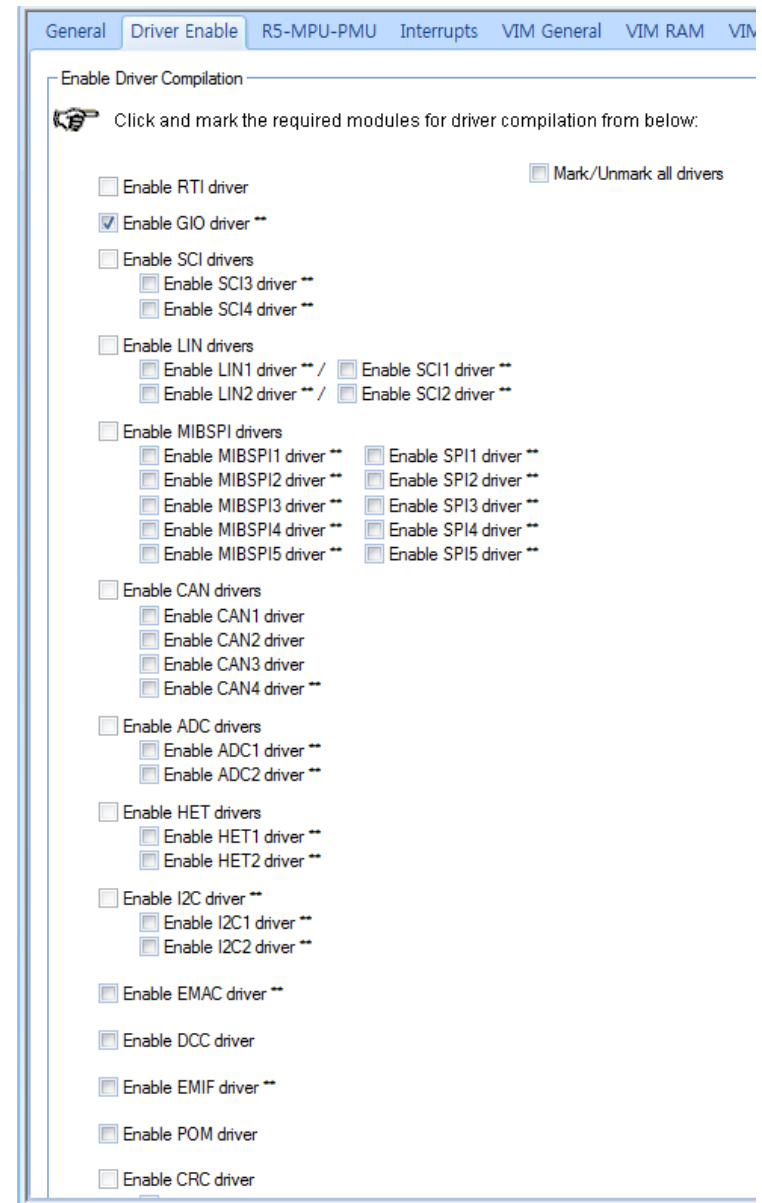
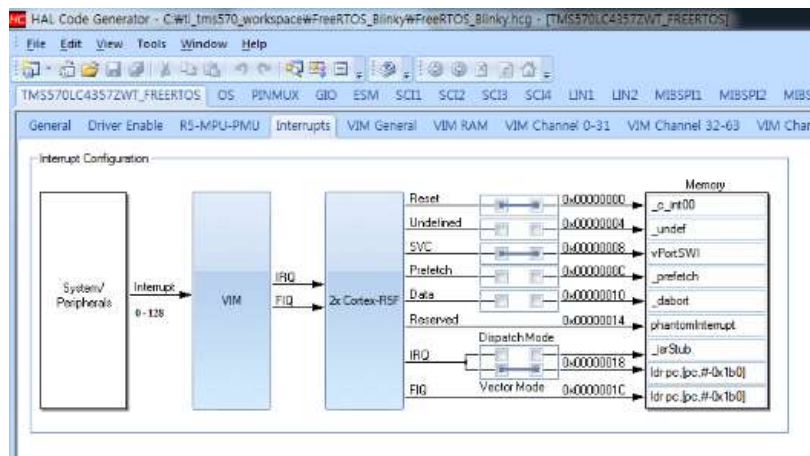
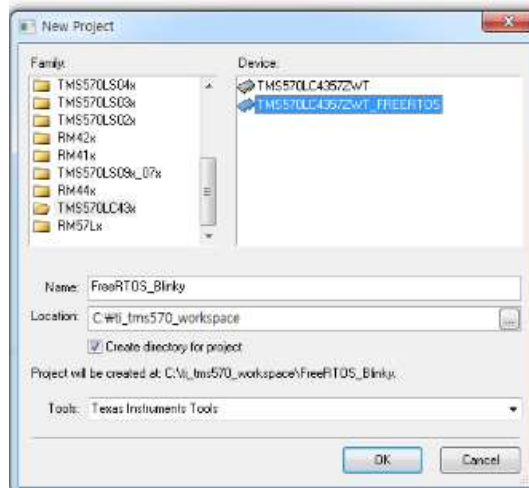


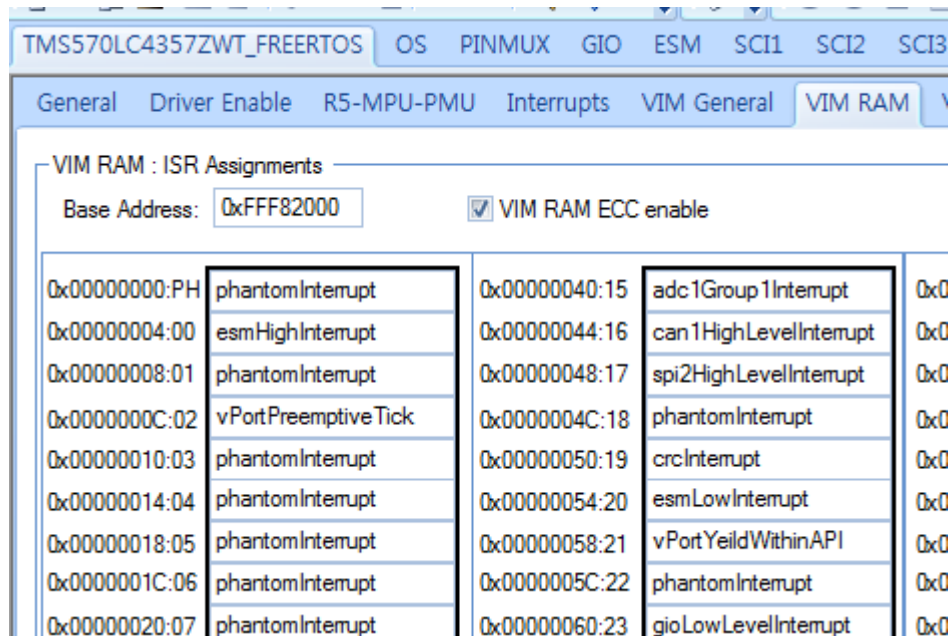
Schematics를 보면 RC회로를 이용한 하드웨어 필터를 사용하고 있음을 알 수있다.

아날로그 READ가 회로도에서 W14가 되어야함

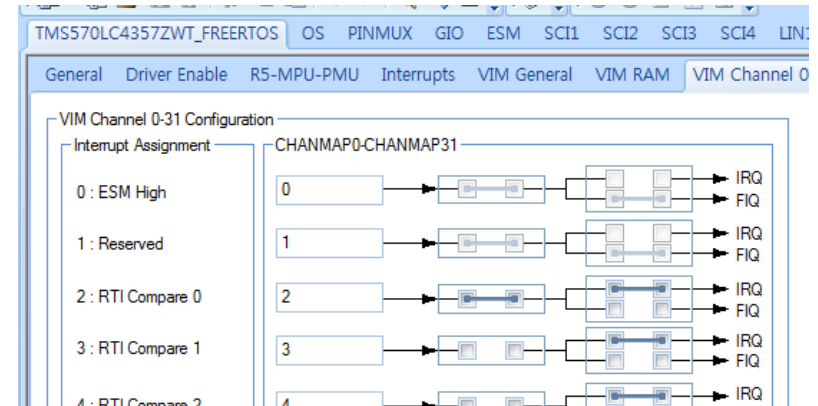
빛이 있으면 AD 꺼지도록 알고리즘을 구성한다.

3. FreeRTOS

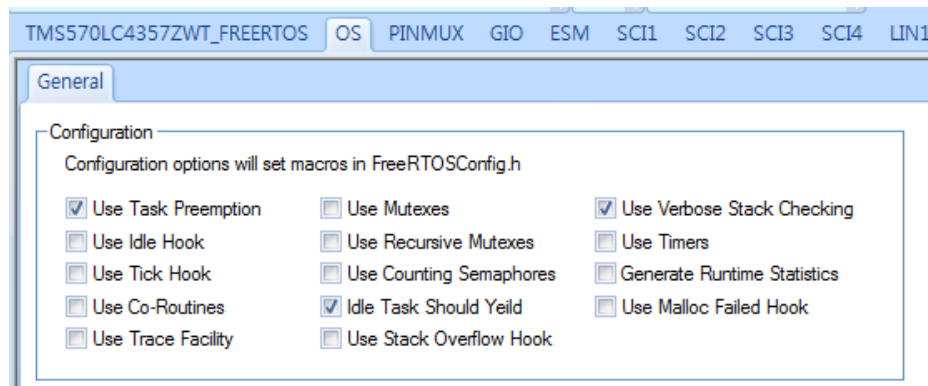




2번(선점형 틱을 발생)과 21번(급한 테스트에 양보한다)은 RTOS에서 사용하는 시스템 콜 인터럽트 벡터 테이블...?!



2번 21번 이 셋팅 되어있는지 확인



코드를 작성한다

```
/* USER CODE BEGIN (0) */
#include "FreeRTOS.h"
#include "os_task.h"
#include "HL_het.h"
#include "HL_gio.h"
#include "HL_sys_common.h"

xTaskHandle xTask1Handle;

void vTask1(void *pvParameters)
{
    for(;;)
    {
        gioSetBit(hetPORT1, 17, gioGetBit(hetPORT1,17)^1);
        vTaskDelay(100);
    }
}

/* USER CODE END */

/* Include Files */

/* USER CODE BEGIN (1) */
/* USER CODE END */

/** @fn void main(void)
 * @brief Application main function
 * @note This function is empty by default.
 *
 * This function is called after startup.
 * The user can use this function to implement the application.
 */
```



```

/* USER CODE BEGIN (2) */
/* USER CODE END */

void main(void)
{
/* USER CODE BEGIN (3) */

    gpioSetDirection(hetPORT1, 0xFFFFFFFF);

    if(xTaskCreate(vTask1, "Task1", configMINIMAL_STACK_SIZE, NULL,1,&xTask1Handle) != pdTRUE)
    {
        while(1);
    }
    /*Start Scheduler*/
    vTaskStartScheduler();











    /*Run forever*/
    while(1);

/* USER CODE END */
}

/* USER CODE BEGIN (4) */
/* USER CODE END */

```

우리가 분석해야 하는 코드들 (MCU하려면 이걸 반드시 해야한다)

- ▷  os_croutine.c
- ▷  os_event_groups.c
- ▷  os_heap.c
- ▷  os_list.c
- ▷  os_mpu_wrappers.c
- ▷  os_port.c
- ▷  os_portasm.asm
- ▷  os_queue.c
- ▷  os_tasks.c
- ▷  os_timer.c