

# **TI DSP, MCU 및 Xilinx Zynq FPGA**

## **프로그래밍 전문가 과정**

**강사 - Innova Lee(이상훈)**  
**[gcccompil3r@gmail.com](mailto:gcccompil3r@gmail.com)**  
**학생 - 하성용**  
**[accept0108@naver.com](mailto:accept0108@naver.com)**

54 일차

//Open Collector

ccs 실행

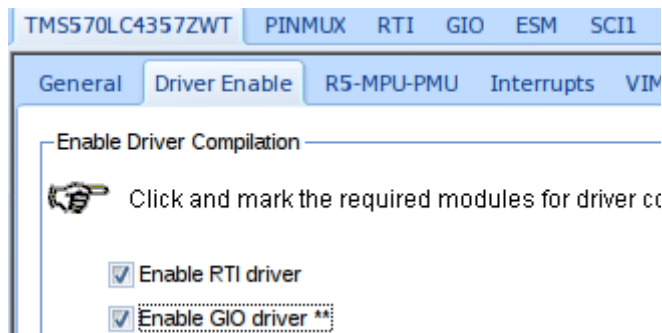
oc\_circuit

"\${workspace\_loc}/\${ProjName}/include}"

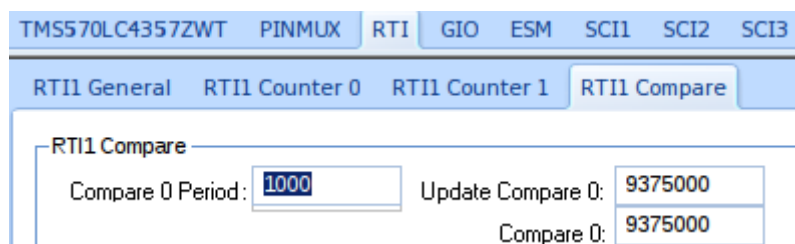
hal 실행

oc\_circuit

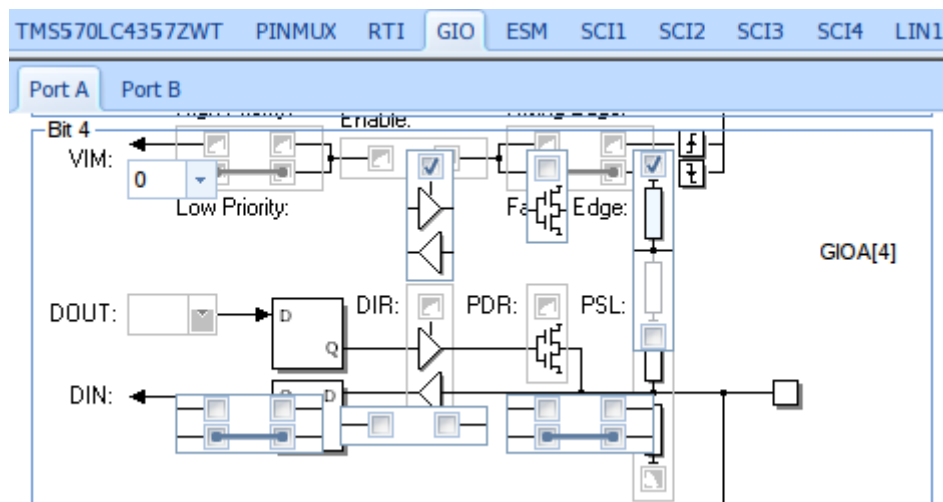
/home/yong/workspace\_v8/OC\_circuit



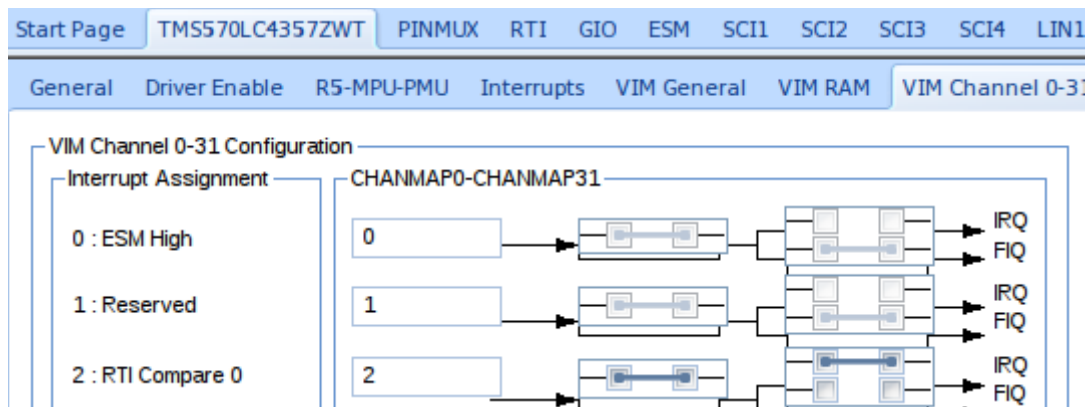
//드라이버



//RTI



//Bit4의 DIR, PSL 활성화



VIM// 2 번

CCS – OC\_circuit 소스

```
#include "HL_sys_common.h"
#include "HL_gio.h"
#include "HL_rti.h"

int main(void)
{
    gioInit();
    rtiInit();

    gioSetDirection(gioPORTA, 0xffffffff);
    rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);
    gioSetPort(gioPORTA, 0xffffffff);

    _enable_IRQ_interrupt_();

    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

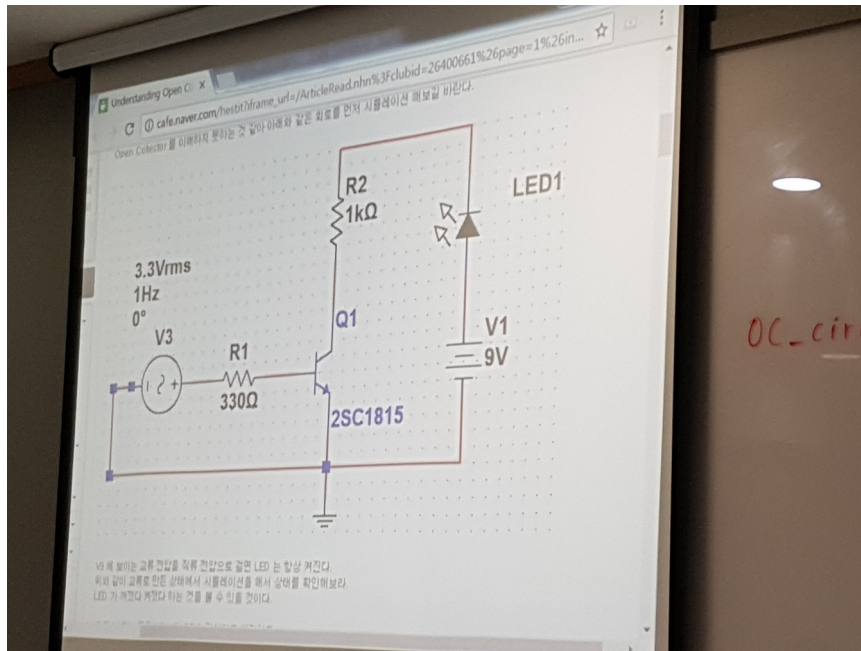
    for(;;)
        ;

    return 0;
}

void rtiNotification(rtiBASE_t *rtiREG, uint32 notification)
{
    gioToggleBit(gioPORTA, 4);
}
```

// 회로 구성

Open Collector(OC)를 이해하기 위한 회로



// V9 에 보이는 교류 전압을 직류 전압으로 걸면 LED 는 항상 켜진다

// LED 가 꺼졌다 켜졌다 하는 것을 볼 수 있다

이론설명

MCU 는 3.3V 가 나온다

3.3V 가 330Ω 으로 나뉘면 전류가 10mA(milli Ampere, 밀리암페어)

계산 : 330 분에 3.3 = 3300 분에 33 은 100 분의 1

약 10mA 가 들어가고

LED(컬렉터)쪽에는

계산 : 1000 분에 9

약 9mA

R1 쪽에 전류가 흐르면 2SC1815(트랜지스터)가 열림

건전지 꺼져있으면 안열림

즉, 스위치(switch) 역할을 할수있음

그리고 어떤 기능을 하는 회로가 R2~LED 쪽에있을때

OC 가 없으면 언제나 동작하고있어 방전이 된다

즉, 특정상황에 트리거를 해줄수있음 (V3)

자동차를 예로들면 unlock signal 을 날려주면 mcu 가 부딪힌다, 속도줄여라 등의 명령을 자동으로 해줄수있다

Open Collector(OC)없이 자동으로 해줄수있는게없으며

R1 을 지배하는게 소프트웨어이다

하드웨어를 지배하는건 정해진동작뿐이 못하지만 소프트웨어가 제어하는건 우리가 원하는 상황에서 제어할수있음 이게 OC 를 쓰는이유이다

또 다른이유로는 기본적으로 V3 에서 3.3V 가 나오는데

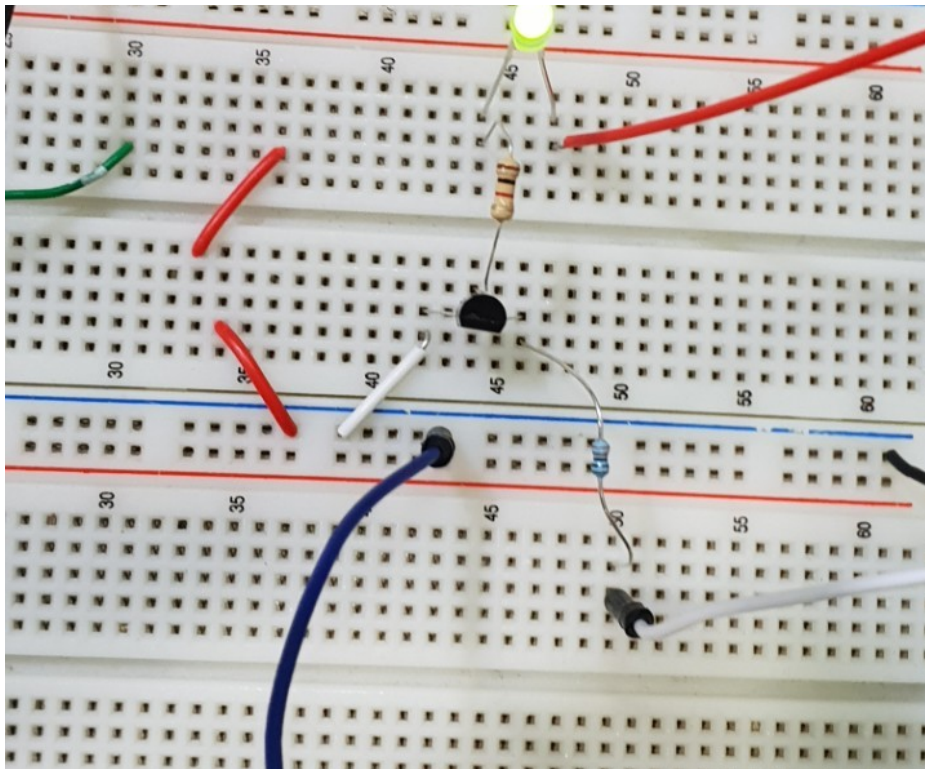
9V 바로 넣으면 타버린다 일종의 보호역할도 하고있음

위쪽회로가 복잡해지면 소비하는 전류가 커지고

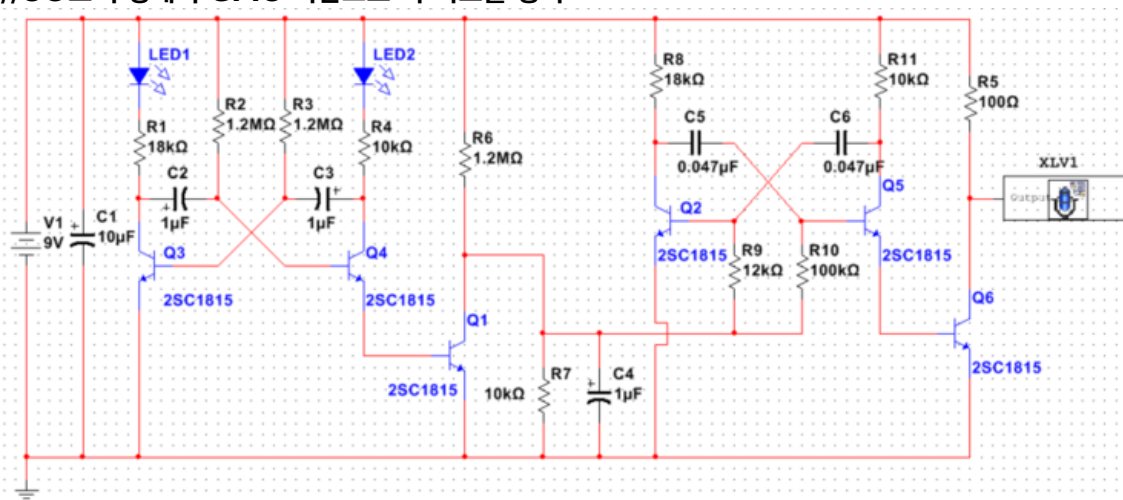
mcu 는 3.3V 고정

전류가 부족해지면 회로가 제대로 동작을 못한다

증폭을 시켜야하는 이유



//OC로 구성해서 GPIO 기반으로 이 회로를 동작



1 $\mu$ F - 이진 콘덴서  
0.047 $\mu$ F - 세라믹 콘덴서

(소자가 없어 사이렌 보류)

영상처리, 레이더등 충격이 생길거같은때 경보음 울리기등 하는거에 필요함

ex) 차량에 라이트  
좋은차는 어두워진다 → 자동으로 라이트켜짐  
비오면 → 자동 와이퍼  
센서가 필요  
그 센서를 제어하는게 ADC\_UART

//ADC\_UART

ccs\_실행

/home/yong/workspace\_v8/ADC\_UART

hal 실행

TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1

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

Enable Driver Compilation

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

☐ Enable RTI driver ☐ Mark/Unmark all drivers

☒ Enable GIO 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 \*\*

//드라이버

MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4 SPI5 CAN1 CAN2 CAN3 CAN4 ADC1

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

ADC1 Configuration

Cycle Time (ns): 100.00

VCLK1 (MHz): 75.0

Prescale: 7

Actual Cycle Time (ns): 106.67

☐ RAM Parity enable

// ADC - 12 비트

MIBSPI4
MIBSPI5
SPI1
SPI2
SPI3
SPI4
SPI5
CAN1
CAN2
CAN3
CAN4
ADC1

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

### 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

GIOB0
EVENT

Default Trigger
Alternate Trigger

Rising Edge
Falling Edge

Hardware
SW Trigger
Software

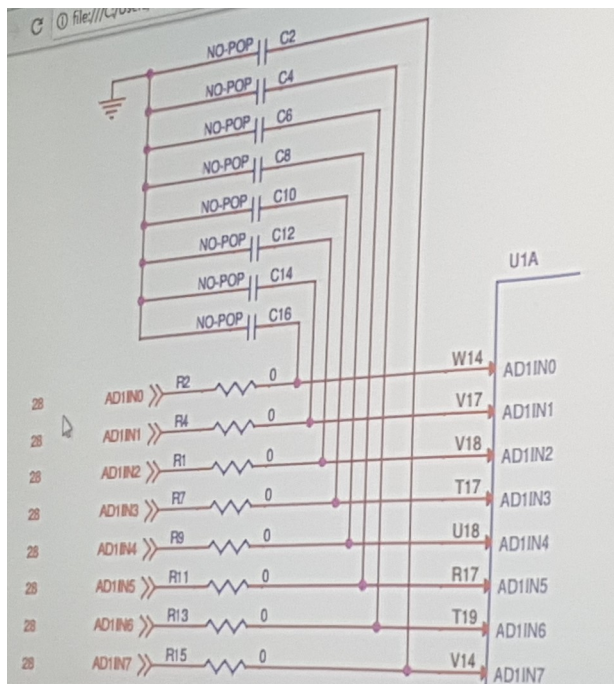
### ADC1 Group 1 Sampling

	tScan	Start:	tDischarge	Sample	tConversion	End:
<input type="checkbox"/> Enable Sampling Capacitor Discharge						
Cycle Time:	106.67		Discharge Time: 0.00			
	100.00		Discharge Prescaler: 0	Sample Time: 300.00		
Cycle Time:	106.67		Sample Prescaler: 1	tSample (ns): 320.01		
tScanTotal	66.667		tExtended (ns): 320.01	tConversion (us): 1.387		
tTotal (us):	3.480687					

### ADC1 Group 1 Channel Selection

☒ Enable Pin 0
☒ Enable Pin 1
☐ Enable Pin 2
☐ Enable Pin 3

//ADC 그룹설정





AD1IN0 에 세라믹콘덴서가 물려있음  
디지털필터가 아니기때문에 잡음은있지만 다른것보다 훨씬 안정적  
소프트웨어필터가 하드웨어필터보다 뛰어남, 고주파들 걸러내고 잡음제거  
대신 어려움, 정확도가 무지높음

---

CCS

ADC\_UART → 5V 필요

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

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

#define TSIZE1 12
uint8 TEXT1[TSIZE1] = {'\r', '\n', '|', 't', 'C', 'H', '-', '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, 1);

    sciInit();

    adcInit();
    adcStartConversion(adcREG1, adcGROUP1);

    for(;;){
        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;

        gioSetBit(gioPORTB, 0, 0);

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



```

        wait(0xFFFFF);
    }
    return 0;
}

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;

    while(length--){
        txt = *text++;

        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){
    while(time){
        time--;
    }
}

```

//조도가 없어지면 LED 가 켜지는 소스 (make by sh)

```

#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;

        gioSetBit(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){
            gioSetBit(gioPORTB, 4, 1);
        }
        else{
            gioSetBit(gioPORTB, 4, 0);
        }
    }
    /*
    id = adc_data[1].id;
    value = adc_data[1].value;
    */
}
59

```

```

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

}

}

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);
        }
    }

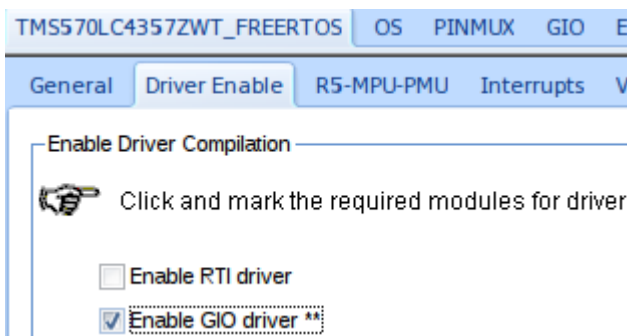
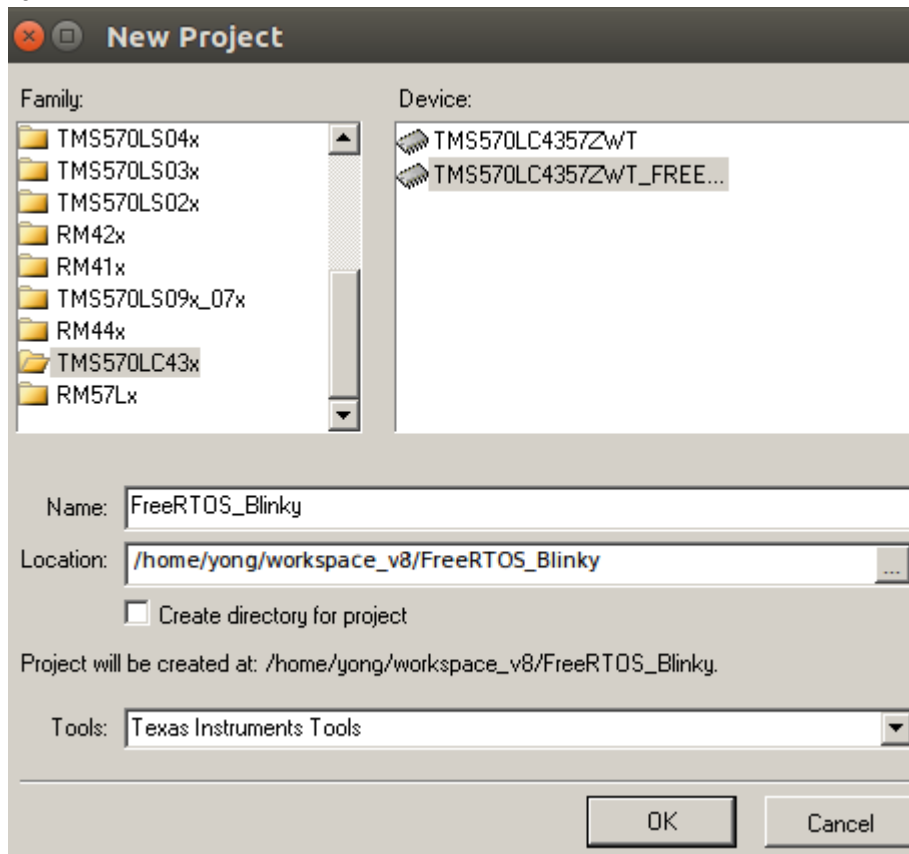
    void wait(uint32 time){
        int i;
        for(i=0; i<time; i++);
    }
}

```

```

//FreeRTOS_Blinky
hal

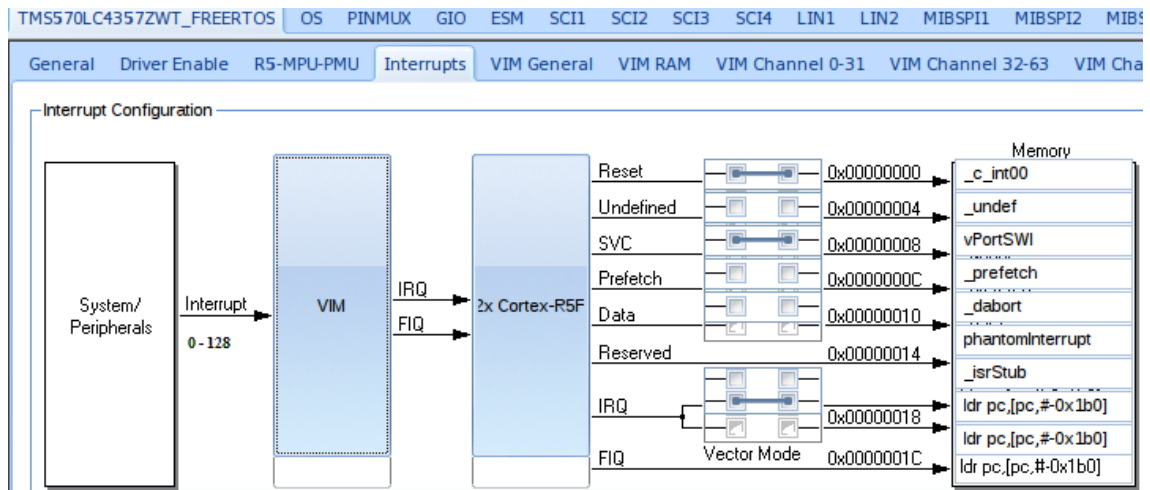
```



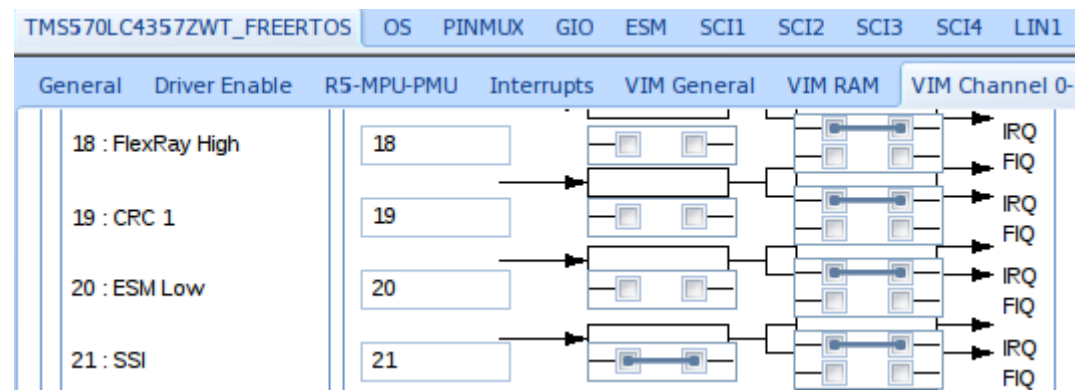
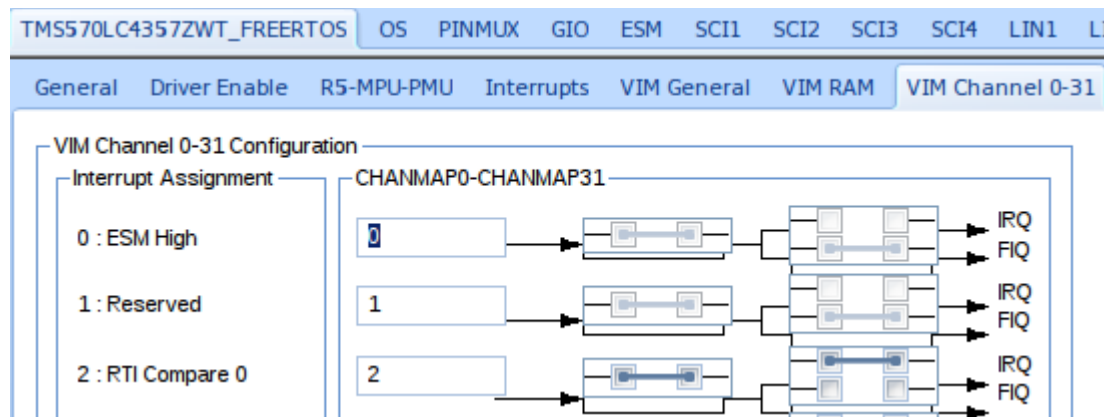
```

//드라이버

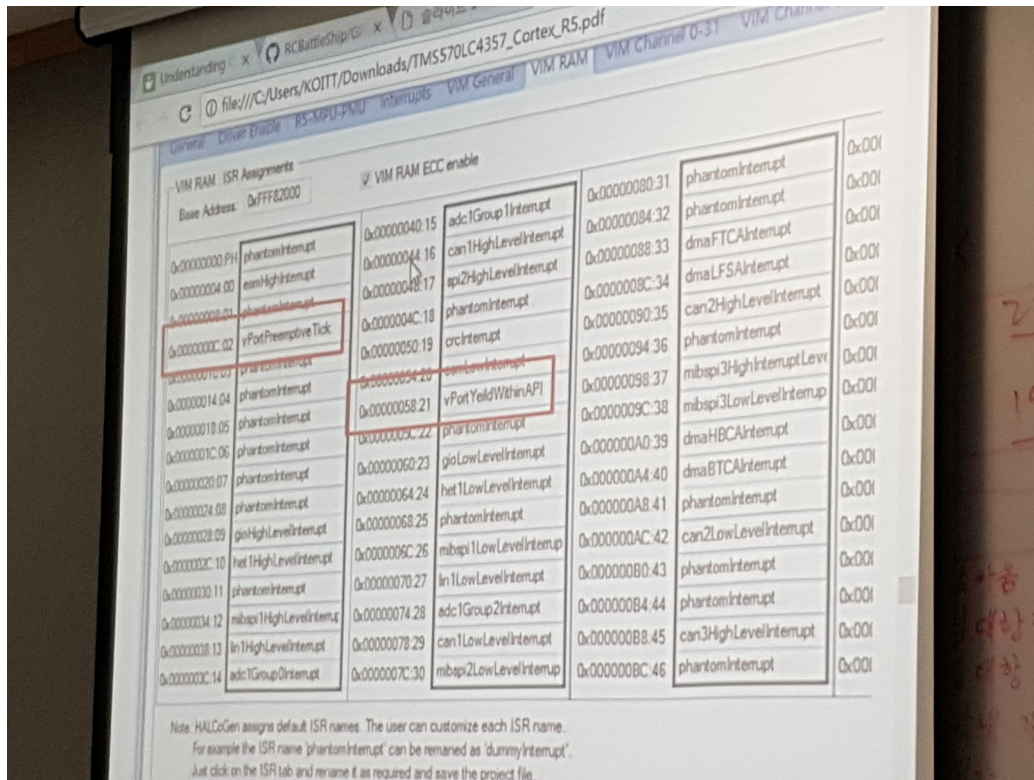
```



//Interrupts



//VIM Channel 0-31



//VIM RAM 2 번하고 21 번이 같아야함  
선정형 틱 발형  
나보다 높은사람이 왔으니 양보를하겠다.  
그런이유로 2 번과 21 번이 세팅되었음

TMS570LC4357ZWT\_FREERTOS OS PINMUX GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 L

General

Configuration

Configuration options will set macros in

☒ Use Task Preemption ☐ Use Mutexes ☒ Use Verbose Stack Checking

☐ Use Idle Hook ☐ Use Recursive Mutexes ☐ Use Timers

☐ Use Tick Hook ☐ Use Counting Semaphores ☐ Generate Runtime Statistics

☐ Use Co-Routines ☒ Idle Task Should Yield ☐ Use Malloc Failed Hook

☐ Use Trace Facility ☐ Use Stack Overflow Hook

Task Configuration

RTI Clock (Hz): 75000000 Tick Rate (Hz): 1000

Max Priorities: 5 Total Heap Size: 8192

Task Name Length: 16 Min Stack Size: 128

Coroutine Configuration

Coroutine Priorities: 2

Timers Configuration

Timer Task Priority: 0 Queue Length: 0 Stack Size: 0

//OS

Use Verbose Stack Checking //스택이 깨지는지 안깨지는지 체크

→ Generate Code

ccs - 코드 입력하고 - 실행해서 het 이 동작하면됨

```
#include "HL_sys_common.h"
#include "FreeRTOS.h"
#include "os_task.h"

#include "HL_het.h"
#include "HL_gio.h"

xTaskHandle xTask1Handle;

void vTask1(void *pvParameters)
{
    for(;;)
    {
        /*Taggle HET[1] with timer tick */
        gioSetBit(hetPORT1, 17, gioGetBit(hetPORT1, 17)^1);
        vTaskDelay(100);
    }
}

void main(void)
{
    /*Set high end timer GIO port hetPort pin direction to all output */
    gioSetDirection(hetPORT1, 0xFFFFFFFF);

    /*Create Task 1 → rtos 에 있는 테스크를 생성하는 api
    리눅스 커널의 포크랑 같음











    Task1 이라는애가 새로생김
    Task1 을 구동시키는 소스가 vTask1
    configMINIMAL → 일단은 테스크라 스택이 필요해서 최소한의 스택사이즈 Define
    해서 잡아놓고있는것
    FreeRTOS manual 옵션값 → MCU 하는사람들이 하나더 해야하는거
    지정하는거 자기가 원하는대로 제어할수있게 입력 */

    if(xTaskCreate(vTask1, "Task1", configMINIMAL_STACK_SIZE, NULL, 1,
    &xTask1Handle) != pdTRUE)
    {
        /*Task could not be created */
        while(1);
    }

    /*Start Scheduler */
    vTaskStartScheduler();

    /* Run forever */
    while(1);
}
```



-  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

-----

os\_port- 스케줄링하는 용도로써

os\_portasm.asm -

portSAVE\_CONTEXT //컨텍스트 스위칭 → 현재 컨텍스트를 저장하는 함수

portRESTORE\_CONTEXT .macro

```

        LDR      R0, pxCurTCBConst
        LDR      R0, [R0]

        ; task stack MPU region
        mov      r4, #12                ; Task Stack Region
        add      r12, r0, #4            ; point to regions in TCB
        ldmia    r12!, {r1-r3}
        mcr      p15, #0, r4, c6, c2, #0 ; Select region
        mcr      p15, #0, r1, c6, c1, #0 ; Base Address
        mcr      p15, #0, r3, c6, c1, #4 ; Access Attributes
        mcr      p15, #0, r2, c6, c1, #2 ; Size and Enable

        ldr      r5, portMax_MPU_Region
        mov      r4, #13
        ; dynamic MPU per task

```

컨텍스트 저장하는데 코프로세스 15 번이 쓰임

동작이 다다름

; Start the first task by restoring its context.

```

        .def      vPortStartFirstTask
        .asmfunc

```

```

vPortStartFirstTask
    cps #0x13
    portRESTORE_CONTEXT
    .endasmfunc

```

vPortStartFirstTask //메인태스크

```

/*-----*/
; Yield to another task.

```

```

        .def      vPortYieldProcessor
        .asmfunc

```

```

swiPortYield
    ; Restore stack and LR before calling vPortYieldProcessor
    ldmfd    sp!, {r11,r12,lr}

```

```

vPortYieldProcessor
    ; Within an IRQ ISR the link register has an offset from the true return
    ; address. SWI doesn't do this. Add the offset manually so the ISR
    ; return code can be used.
    ADD     LR, LR, #4

```

yield//스택을 보관하고 스택과 에러를 보관하고 동작하고있다가  
우선순위 높은애가 오면 양보한다  
자기가 저장해놓았던걸 빼놓고 저기넘어가서 작업

swiPortYield//소프트웨어

```
; Select the next task to execute. */  
BL      vTaskSwitchContext
```

vTaskSwitchContext //텍스트 전환

```
;/*-----*/  
; Preemptive Tick
```

```
        .def vPortPreemptiveTick  
        .asmfunc  
vPortPreemptiveTick  
        ; Save the context of the current task.  
        portSAVE_CONTEXT
```

Tick // 인터럽트 2 번

os\_task.c -  
xTaskCreate//FreeRTOS 에 있는 task\_struct

pxStack//태스크로 스택이 필요

pvPortMalloc//스택할당

tasktoReadyList // 런큐  
리스트에 우선순위 높은걸 먼저하는걸 도와줌

vTaskDelete // vTask 를 종료시키는 함수

Linux 는 모든것을 다올려야해서 복잡한데 비해  
FreeRTOS 는 os 만 올리면되서 구동시키는 함수들이 훨씬 단순함