TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)

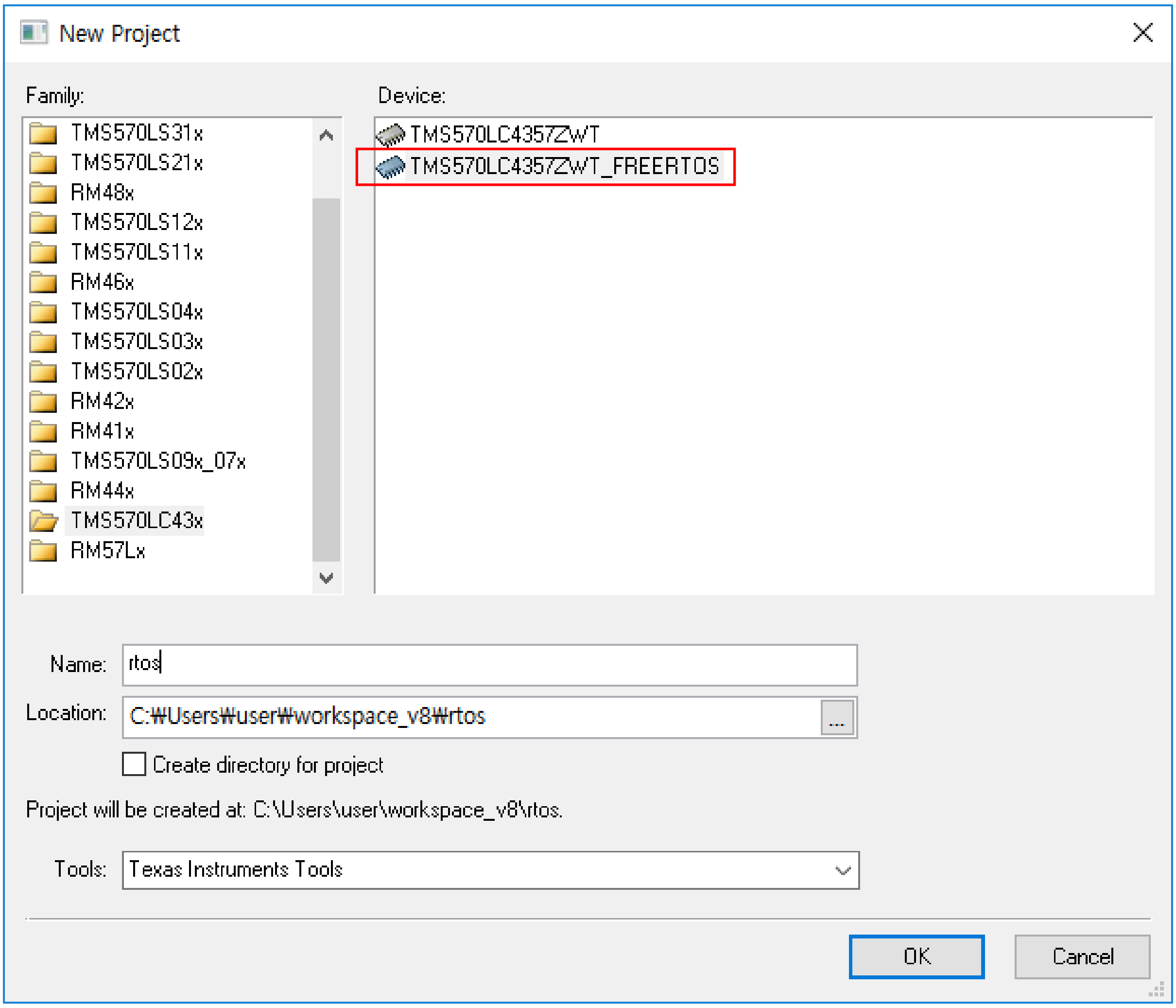
gcccompil3r@gmail.com

학생 – 문한나

mhn97@naver.com

목차

HGC 설정



New Project

Family:

- TMS570LS31x
- TMS570LS21x
- RM48x
- TMS570LS12x
- TMS570LS11x
- RM46x
- TMS570LS04x
- TMS570LS03x
- TMS570LS02x
- RM42x
- RM41x
- TMS570LS09x_07x
- RM44x
- TMS570LC43x
- RM57Lx

Device:

- TMS570LC4357ZWT
- TMS570LC4357ZWT_FREERTOS

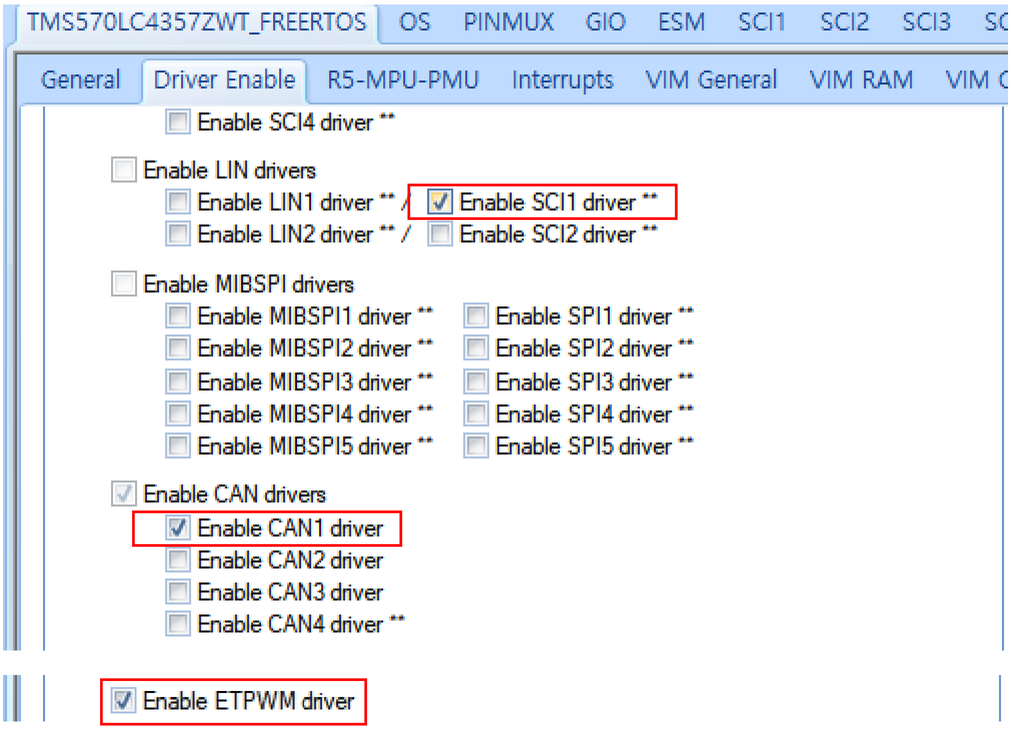
Name: rtos

Location: C:₩Users₩user₩workspace_v8₩rtos

□ Create directory for project

Project will be created at: C:\Users\user\workspace_v8\rtos.

Tools: Texas Instruments Tools



☐ 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

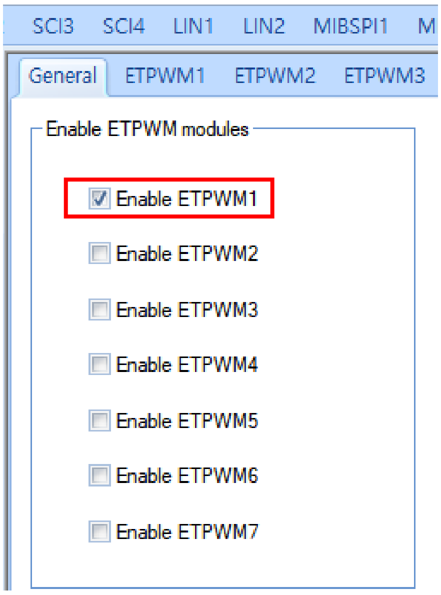
☑ Enable CAN1 driver

☐ Enable CAN2 driver

☐ Enable CAN3 driver

☐ Enable CAN4 driver **

☑ Enable ETPWM driver



Enable ETPWM modules

☑ Enable ETPWM1

☐ Enable ETPWM2

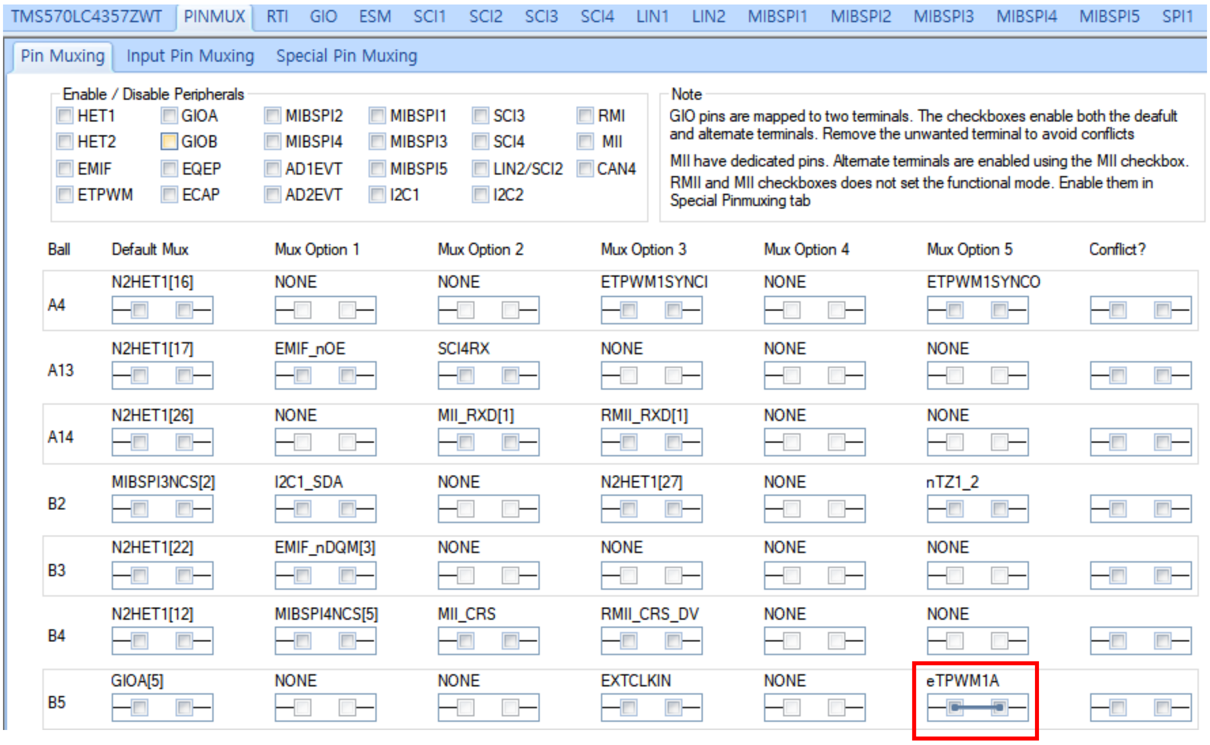
☐ Enable ETPWM3

☐ Enable ETPWM4

☐ Enable ETPWM5

☐ Enable ETPWM6

☐ Enable ETPWM7



Enable / Disable Peripherals

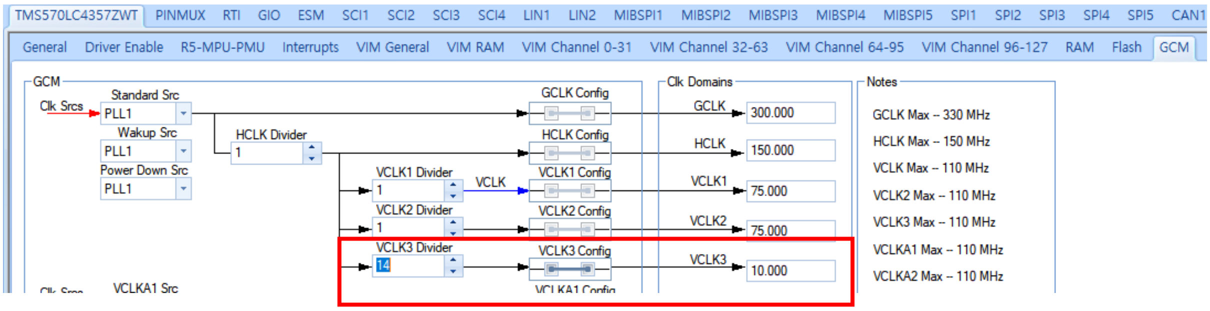
□ HET1	□ GIOA	□ MIBSPI2	□ MIBSPI1	□ SCI3	□ RMI
□ HET2	□ GIOB	□ MIBSPI4	□ MIBSPI3	□ SCI4	□ MII
□ EMIF	□ EQEP	□ AD1EVT	□ MIBSPI5	□ LIN2/SCI2	□ CAN4
□ ETPWM	□ ECAP	□ AD2EVT	□ I2C1	□ I2C2	

Note

GIO pins are mapped to two terminals. The checkboxes enable both the deafult and alternate terminals. Remove the unwanted terminal to avoid conflicts

MII have dedicated pins. Alternate terminals are enabled using the MII checkbox. RMII and MII checkboxes does not set the functional mode. Enable them in Special Pinmuxing tab

Ball	Default Mux	Mux Option 1	Mux Option 2	Mux Option 3	Mux Option 4	Mux Option 5	Conflict?
A4	N2HET1[16]	NONE	NONE	ETPWM1SYNCI	NONE	ETPWM1SYNCO	
A13	N2HET1[17]	EMIF_nOE	SCI4RX	NONE	NONE	NONE	
A14	N2HET1[26]	NONE	MII_RXD[1]	RMII_RXD[1]	NONE	NONE	
B2	MIBSPI3NCS[2]	I2C1_SDA	NONE	N2HET1[27]	NONE	nTZ1_2	
B3	N2HET1[22]	EMIF_nDQM[3]	NONE	NONE	NONE	NONE	
B4	N2HET1[12]	MIBSPI4NCS[5]	MII_CRS	RMII_CRS_DV	NONE	NONE	
B5	GIOA[5]	NONE	NONE	EXTCLKIN	NONE	eTPWM1A	



Standard Src: PLL1

Wakup Src: PLL1

Power Down Src: PLL1

HCLK Divider: 1

VCLK1 Divider: 1

VCLK2 Divider: 1

VCLK3 Divider: 14

Clk Domains

GCLK: 300.000

HCLK: 150.000

VCLK1: 75.000

VCLK2: 75.000

VCLK3: 10.000

Notes

GCLK Max -- 330 MHz

HCLK Max -- 150 MHz

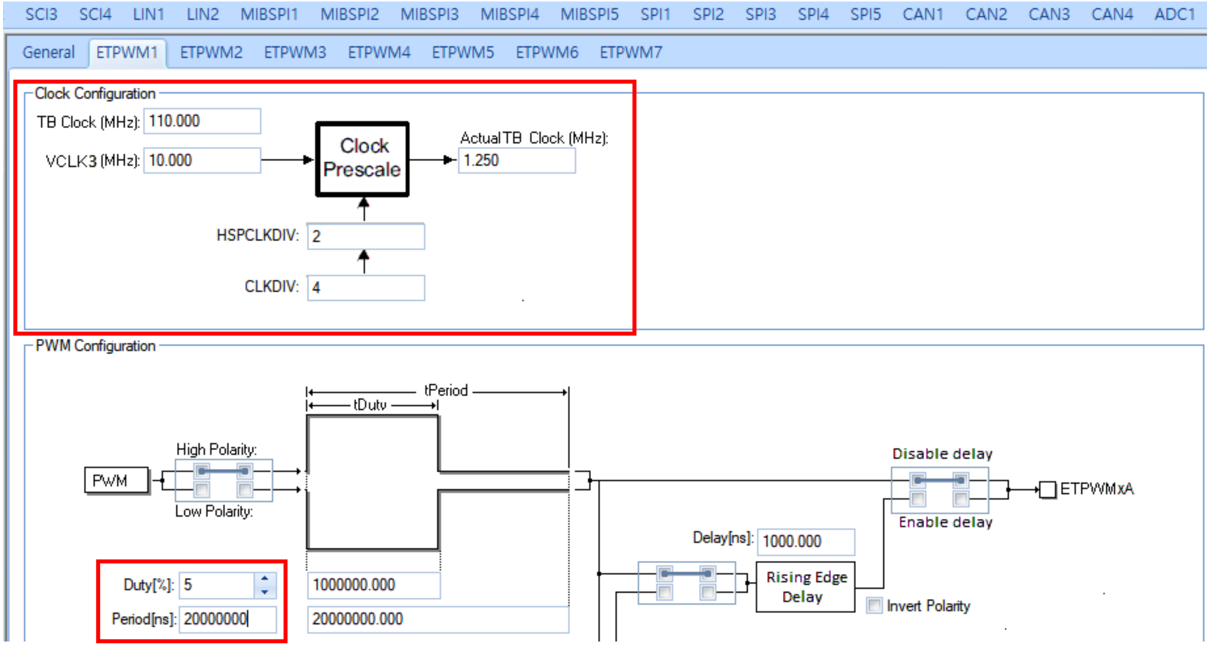
VCLK Max -- 110 MHz

VCLK2 Max -- 110 MHz

VCLK3 Max -- 110 MHz

VCLKA1 Max -- 110 MHz

VCLKA2 Max -- 110 MHz


SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4 SPI5 CAN1 CAN2 CAN3 CAN4 ADC1
General ETPWM1 ETPWM2 ETPWM3 ETPWM4 ETPWM5 ETPWM6 ETPWM7
Clock Prescale
PWM Configuration
tPeriod
tDuty
High Polarity:
PWM
Low Polarity:
Disable delay
Enable delay
ETPWMxA
Rising Edge Delay

Clock Configuration

TB Clock (MHz): 110.000

VCLK3 (MHz): 10.000

Actual TB Clock (MHz): 1.250

HSPCLKDIV: 2

CLKDIV: 4

Duty[%]: 5

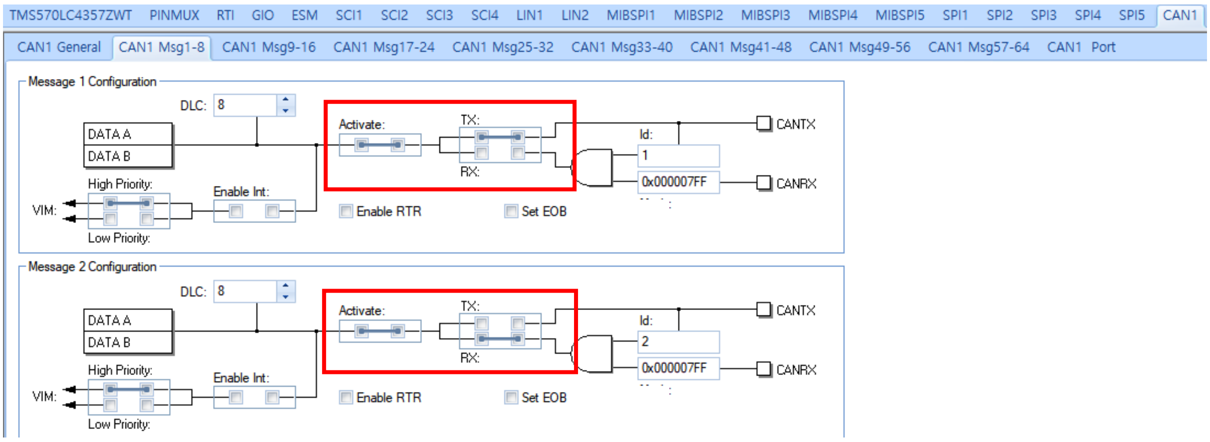
Period[ns]: 20000000

1000000.000

20000000.000

Delay[ns]: 1000.000

□ Invert Polarity


TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4 SPI5 CAN1
CAN1 General CAN1 Msg1-8 CAN1 Msg9-16 CAN1 Msg17-24 CAN1 Msg25-32 CAN1 Msg33-40 CAN1 Msg41-48 CAN1 Msg49-56 CAN1 Msg57-64 CAN1 Port
DATA A
DATA B
High Priority:
VIM:
Low Priority:
Enable Int:
Activate:
TX:
RX:
CANTX
CANRX

Message 1 Configuration

DLC: 8

Id: 1

0x000007FF

□ Enable RTR

□ Set EOB

Message 2 Configuration

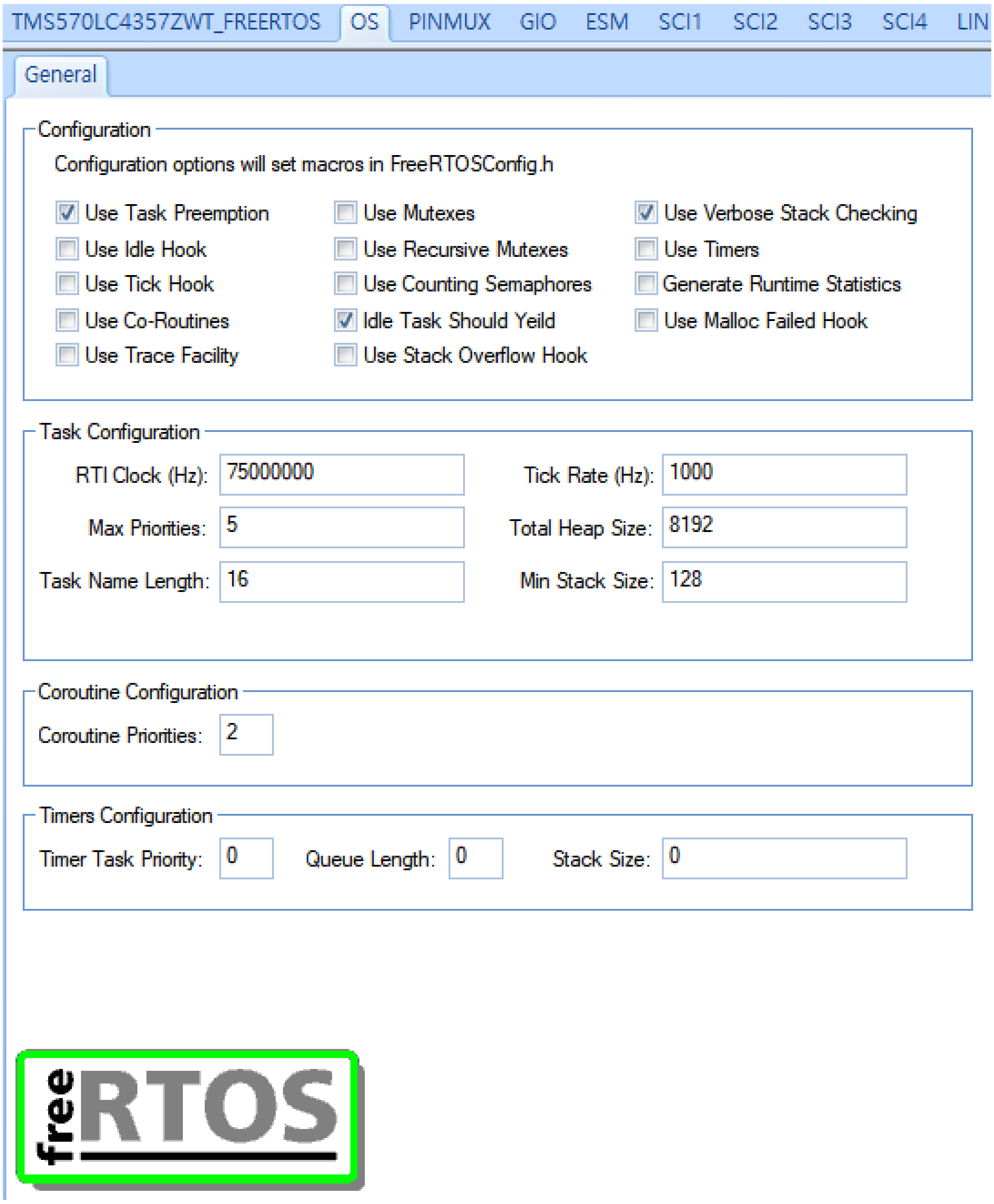
DLC: 8

Id: 2

0x000007FF

□ Enable RTR

□ Set EOB



Configuration

Configuration options will set macros in FreeRTOSConfig.h

☑ Use Task Preemption
☐ Use Idle Hook
☐ Use Tick Hook
☐ Use Co-Routines
☐ Use Trace Facility

☐ Use Mutexes
☐ Use Recursive Mutexes
☐ Use Counting Semaphores
☑ Idle Task Should Yeild
☐ Use Stack Overflow Hook

☑ Use Verbose Stack Checking
☐ Use Timers
☐ Generate Runtime Statistics
☐ Use Malloc Failed 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

ccs 코드

```c
#include <FreeRTOS.h>
#include <FreeRTOSConfig.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_sci.h>
#include <HL_sci.h>
#include <os_mpu_wrappers.h>
#include <os_projdefs.h>
#include <os_semphr.h>
#include <os_task.h>
#include "string.h"
#include "stdio.h"
#include "stdlib.h"
#include "HL_can.h"
#include "HL_sys_common.h"
#include "HL_system.h"
#include "HL_esm.h"
#include "HL_sys_core.h"
#include "HL_etpwm.h"

xTaskHandle xTask1Handle;
xTaskHandle xTask2Handle;
xTaskHandle xTask3Handle;

QueueHandle_t mutex = NULL;

long check = 0;
char flag = 0;
uint8 rx_data[8] = { 0 };

void vTask1(void* pvParameters);
void vTask2(void* pvParameters);
void vTask3(void* pvParameters);
void send_data(sciBASE_t* sci, uint8* msg, int length);

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

int main(void)
{
    sciInit();
    canInit();
    etpwmInit();

    etpwmStartTBCLK();

    delay(1000000);

    canEnableErrorNotification(canREG1);

    vSemaphoreCreateBinary(mutex) // 리턴값으로 생성이 성공하면 NULL이 아닌값을 리턴한다.

    if(xTaskCreate(vTask1, "Task1" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask1Handle) !=
pdTRUE){
        while(1)
            ;
    }
    if(xTaskCreate(vTask2, "Task2" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask2Handle) !=
pdTRUE){
        while(1)
            ;
    }
    if(xTaskCreate(vTask3, "Task3" , configMINIMAL_STACK_SIZE, NULL, 1, &xTask3Handle) !=
pdTRUE){
            while(1)
                ;
    }

    vTaskStartScheduler();

    while(1)
        ;
}
void vTask1(void *pbParameters)
{
    uint8 str[32] = {'s','t','r','a','i','g','h','t','\r','\n'};
    uint8 rig[32] = {'r','i','g','h','t','\r','\n'};
    uint8 lef[32] = {'l','e','f','t','\r','\n'};
    uint8 msg[32] = {'e','r','r','o','r','\r','\n'};

    while(1){

        if(flag == 2){ //실행 순서를 정해주는 flag
            if(xSemaphoreTake(mutex, ( TickType_t ) 10) == pdTRUE){ //키를 받아왔을 경우,
실행코드 입력.

                if(*rx_data == 0){
                    send_data(sciREG1, str, strlen(str));
                }else if(*rx_data == 1){
                    send_data(sciREG1, rig, strlen(rig));
                }else if(*rx_data == 2){
                    send_data(sciREG1, lef, strlen(lef));
                 }else
                     send_data(sciREG1,msg,strlen(msg));

                xSemaphoreGive(mutex);
                flag = 0; //다음에 실행 될 태스크를 지정할 수 있다.
                vTaskDelay(10); // 10틱을 기다려서 내가 take 하기로한 시간을 초과해 주어 다른
태스크에 키를 전달할 수 있도록 한다. (딜레이 없을 경우 키 반납하자마자 바로 take함)
                }

            else{ //키를 받아오지 못하는 경우에 실행 코드를 작성
                }
        }
        else{// flag가 다른 경우의 작동이 필요한 경우 사용한다.
        }
    }
}

void vTask2(void *pbParameters)
{

    uint8 msg[32] = {'e','r','r','o','r','\r','\n'};

    while(1){
          if(flag == 1){ //실행 순서를 정해주는 flag
             if(xSemaphoreTake(mutex, ( TickType_t ) 10) == pdTRUE){ //키를 받아왔을 경우, 실행코드 입력.

                 if(rx_data[0] == 1){ //우회전
                     etpwmREG1->CMPA = 1875 + (20 * rx_data[1]);
                     sciSendByte(sciREG1, rx_data[1]);
                 }else if(rx_data[0] == 2){ //좌회전
                     etpwmREG1->CMPA = 1875 - (20 * rx_data[1]);
                     sciSendByte(sciREG1, rx_data[1]);
                    // send_data(sciREG1, rx_data[1], strlen(rx_data[1]));
                 }else if(rx_data[0] == 0){ //직진
                     etpwmREG1->CMPA = 1875;
                 }else
                     send_data(sciREG1,msg,strlen(msg));

                 xSemaphoreGive(mutex);
                 flag = 2; //다음에 실행 될 태스크를 지정할 수 있다.
                 vTaskDelay(10); // 10틱을 기다려서 내가 take 하기로한 시간을 초과해 주어 다른 태스크에 키를 전달할 수 있도록 한다. (딜레이 없을 경우 키 반납하자마자 바로 take함)
                 }

             else{ //키를 받아오지 못하는 경우에 실행 코드를 작성
                 }
          }
          else{//  flag가 다른 경우의 작동이 필요한 경우 사용한다.
          }
      }
}

void vTask3(void *pbParameters)
{
    while(1){

          if(flag == 0){ //실행 순서를 정해주는 flag

             if(xSemaphoreTake(mutex, ( TickType_t ) 10) == pdTRUE){ //키를 받아왔을 경우, 실행코드 입력.
                 if (canIsRxMessageArrived(canREG1, canMESSAGE_BOX2)){
                            canGetData(canREG1, canMESSAGE_BOX2, rx_data);
                            //send_data(sciREG1, rx_data, strlen(rx_data));
                            //sciSendByte(sciREG1,'\r');
                            //sciSendByte(sciREG1,'\n');
                            xSemaphoreGive(mutex);
                            flag = 1; //다음에 실행 될 태스크를 지정할 수 있다.
                            vTaskDelay(10); // 10틱을 기다려서 내가 take 하기로한 시간을 초과해 주어 다른 태스크에 키를 전달할 수 있도록 한다. (딜레이 없을 경우 키 반납하자마자 바로 take함)

                 }
             }
             else{ //키를 받아오지 못하는 경우에 실행 코드를 작성
                 //flag = 1;
                 xSemaphoreGive(mutex);
                 vTaskDelay(10);
                 }
          }
          else{// flag가 다른 경우의 작동이 필요한 경우 사용한다.
          }
       }
}


void send_data(sciBASE_t* sci, uint8* msg, int length)
{
    int i;
    for(i=0;i<length;i++)
        sciSendByte(sci,msg[i]);
}
```

ADC 함수 정리

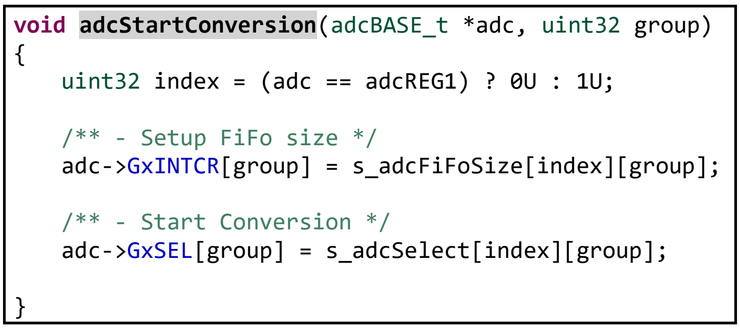
```
void adcInit(void);
void adcStartConversion(adcBASE_t *adc, uint32 group); //디지털 신호로 변환을 시작해주는 함수
void adcStopConversion(adcBASE_t *adc, uint32 group); //디지털 신호로의 변환을 끝내는 함수
void adcResetFiFo(adcBASE_t *adc, uint32 group); // FiFo의 읽기 및 쓰기 포인터를 재설정하는 함수
uint32  adcGetData(adcBASE_t *adc, uint32 group, adcData_t *data); //변환된 데이터를 가져오는 함수
uint32  adcIsFifoFull(adcBASE_t *adc, uint32 group); // FiFo 버퍼 상태를 확인하는 함수
uint32  adcIsConversionComplete(adcBASE_t *adc, uint32 group); //변환이 완료되었는지 확인하는 함수
void adcEnableNotification(adcBASE_t *adc, uint32 group); // Notification 함수를 활성화시키는 함수
void adcDisableNotification(adcBASE_t *adc, uint32 group); // Notification 함수를 비활성화시키는 함수
void adcCalibration(adcBASE_t *adc); // ADC 모듈 교정 함수
uint32 adcMidPointCalibration(adcBASE_t *adc); //ADC 모듈 교정 함수(중간 지점 사용)
void adcSetEVTPin(adcBASE_t *adc, uint32 value); // 출력 핀으로 구성된 ADC EVT 핀을 설정하는 함수
uint32 adcGetEVTPin(adcBASE_t *adc);// ADC EVT 핀의 값을 반환하는 함수

void adc1GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type); // 초기 값 또는 현재 값을 복사하는 함수
void adc2GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type); // 초기 값 또는 현재 값을 복사하는 함수

/** @fn void adcNotification(adcBASE_t *adc, uint32 group)
*   @brief Group notification
*   @param[in] adc Pointer to ADC node:
*             - adcREG1: ADC1 module pointer
*             - adcREG2: ADC2 module pointer
*   @param[in] group number of ADC node:
*             - adcGROUP0: ADC event group
*             - adcGROUP1: ADC group 1
*             - adcGROUP2: ADC group 2
*
*   @note This function has to be provide by the user.
*/
void adcNotification(adcBASE_t *adc, uint32 group); // Notification 함수 활성화 시 동작함수
void adcInit(void)
{
    /** @b Initialize @b ADC1: */

    /** - Reset ADC module */
    adcREG1->RSTCR = 1U; //모든 모듈 재설정

    adcREG1->RSTCR = 0U; //재설정 상태 해제

    /** - Enable 12-BIT ADC  */
    adcREG1->OPMODECR |= 0x80000000U; // ADC 코어 및 디지털 로직이 12 비트로 구성(HCG에서 설정)

    /** - Setup prescaler */
    adcREG1->CLOCKCR = 7U; /*ADC 코어 클록 (ADCLK)에 대한 prescaler 값 정의
                            (t C(ADCLK) = t C(VCLK) × (PS[4:0] + 1)) */

    /** - Setup memory boundaries */
    adcREG1->BNDCR  = (uint32)((uint32)8U << 16U) | (8U + 8U); // ADC 결과 메모리 구성

    adcREG1->BNDEND = (adcREG1->BNDEND & 0xFFFF0000U) | (2U); // ADC 결과 메모리 크기 구성

    /** - Setup event group conversion mode
    *     - Setup data format
    *     - Enable/Disable channel id in conversion result
    *     - Enable/Disable continuous conversion
    */
    adcREG1->GxMODECR[0U] = (uint32)ADC_12_BIT //이벤트 그룹 변환 모드 설정
                          | (uint32)0x00000000U
                          | (uint32)0x00000000U;

    /** - Setup event group hardware trigger
     *     - Setup hardware trigger edge
     *     - Setup hardware trigger source
     */
    adcREG1->EVSRC = (uint32)0x00000000U // 이벤트 그룹 하드웨어 트리거 설정
                   | (uint32)ADC1_EVENT;

    /** - Setup event group sample window */
    adcREG1->EVSAMP = 1U; /* ADC 이벤트 그룹 샘플링 시간 구성 (SW = EV_ACQ + 2 in terms of
                            ADCLK cycles)*/

    /** - Setup event group sample discharge
    *     - Setup discharge prescaler
    *     - Enable/Disable discharge
    */
    adcREG1->EVSAMPDISEN = (uint32)((uint32)0U << 8U) // ADC 이벤트 그룹 샘플 캡 방전 제어
                         | (uint32)0x00000000U;

    /** - Setup group 1 conversion mode
    *     - Setup data format
    *     - Enable/Disable channel id in conversion result
    *     - Enable/Disable continuous conversion
    */
    adcREG1->GxMODECR[1U] = (uint32)ADC_12_BIT // group 1 변환 모드 설정
                          | (uint32)0x00000000U
                          | (uint32)0x00000008U
                          | (uint32)0x00000000U;
    /** - Setup group 1 hardware trigger
     *     - Setup hardware trigger edge
     *     - Setup hardware trigger source
     */
    adcREG1->G1SRC = (uint32)0x00000008U /* group 1 하드웨어 트리거 설정. 선택한 소스에서
                                             low->high 전환하면 Group1 변환이 트리거됨 */

                   | (uint32)ADC1_GIOB0;

    /** - Setup group 1 sample window */
    adcREG1->G1SAMP = 1U; /* group 1샘플링 시간 구성
                           (SW = EV_ACQ + 2 in terms of ADCLK cycles)*/

    /** - Setup group 1 sample discharge
    *     - Setup discharge prescaler
    *     - Enable/Disable discharge
    */
    adcREG1->G1SAMPDISEN = (uint32)((uint32)0U << 8U) // ADC 이벤트 그룹 샘플 캡 방전 제어
                         | (uint32)0x00000000U;

    /** - Setup group 2 conversion mode
     *     - Setup data format
     *     - Enable/Disable channel id in conversion result
     *     - Enable/Disable continuous conversion
     */
    adcREG1->GxMODECR[2U] = (uint32)ADC_12_BIT //group 2 변환 모드 설정
                          | (uint32)0x00000000U
                          | (uint32)0x00000000U
                          | (uint32)0x00000000U;

    /** - Setup group 2 hardware trigger
        *     - Setup hardware trigger edge
    *     - Setup hardware trigger source
        */
    adcREG1->G2SRC = (uint32)0x00000000U //group 2 하드웨어 트리거 설정.
                   | (uint32)ADC1_EVENT;

    /** - Setup group 2 sample window */
    adcREG1->G2SAMP = 1U; /* group 2 샘플링 시간 구성
                           (SW = EV_ACQ + 2 in terms of ADCLK cycles)*/

    /** - Setup group 2 sample discharge
    *     - Setup discharge prescaler
    *     - Enable/Disable discharge
    */
    adcREG1->G2SAMPDISEN = (uint32)((uint32)0U << 8U) // ADC 이벤트 그룹 샘플 캡 방전 제어
                         | (uint32)0x00000000U;

/* ADC EVT 핀 설정 */

    /** - ADC1 EVT pin output value */
    adcREG1->EVTOUT = 0U;

    /** - ADC1 EVT pin direction */
    adcREG1->EVTDIR = 0U;

    /** - ADC1 EVT pin open drain enable */
        adcREG1->EVTPDR = 0U;
    /** - ADC1 EVT pin pullup / pulldown selection */
        adcREG1->EVTPSEL = 1U;

    /** - ADC1 EVT pin pullup / pulldown enable*/
        adcREG1->EVTDIS = 0U;

        /** - Enable ADC module */
    adcREG1->OPMODECR |= 0x80140001U; /* 10000000000101000000000000000001
                                        12bit 로직 사용
                                       측정모드 비활성화
                                        ADC 변환 진행 가능 */

    /** - Wait for buffer initialization complete */
    /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Hardware status bit read check" */
    while (((adcREG1->BNDEND & 0xFFFF0000U) >> 16U ) != 0U)
    {
        } /* Wait */

    /** - Setup parity */
    adcREG1->PARCR = 0x00000005U; // Enable parity checking

}
```

```
void adcStartConversion(adcBASE_t *adc, uint32 group);
```

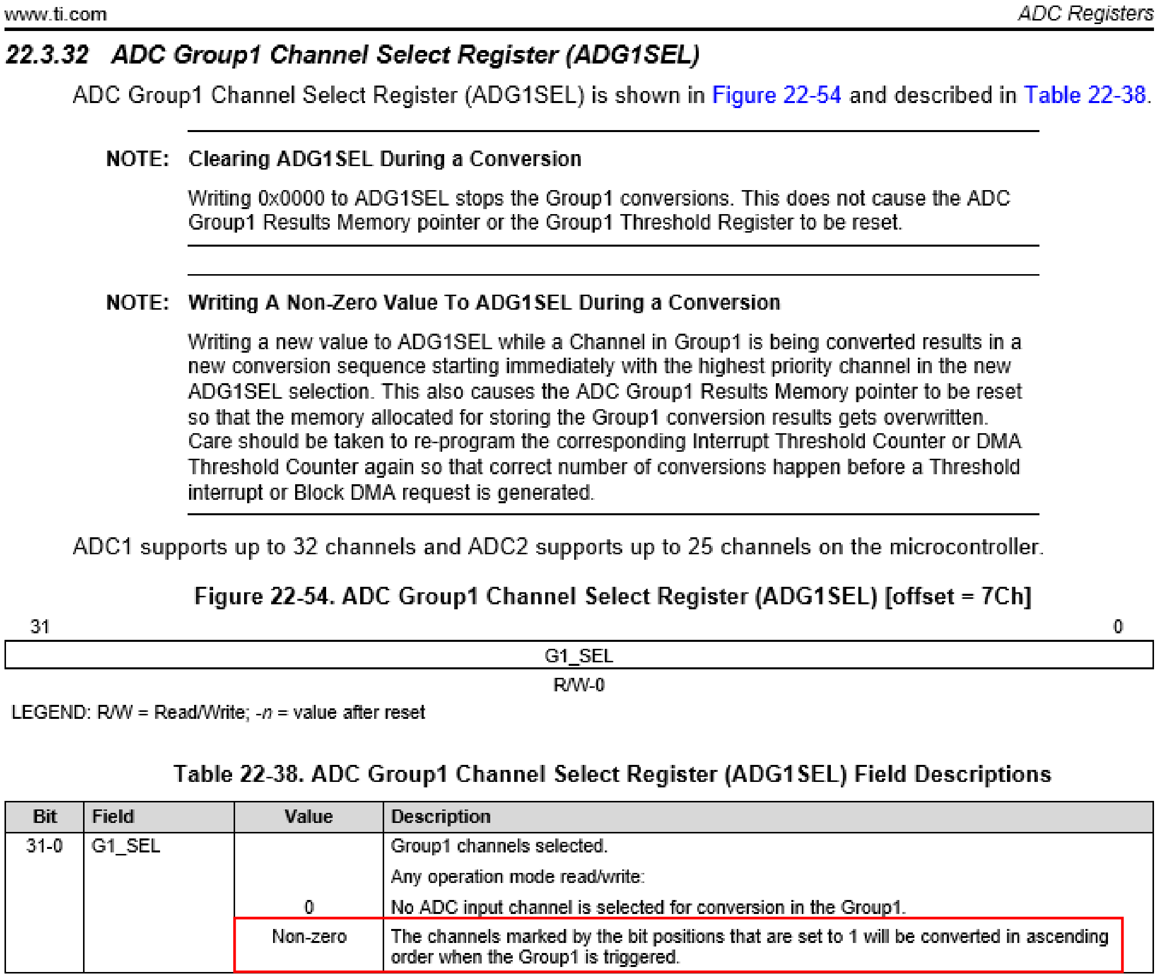
디지털 신호로 변환을 시작해주는 함수

```
adcBASE_t *adc : ADC모듈 번호
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

```
uint32 group : ADC그룹 번호
- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2
```



```
void adcStartConversion(adcBASE_t *adc, uint32 group)
{
    uint32 index = (adc == adcREG1) ? 0U : 1U;

    /** - Setup FiFo size */
    adc->GxINTCR[group] = s_adcFiFoSize[index][group];

    /** - Start Conversion */
    adc->GxSEL[group] = s_adcSelect[index][group];

}
```



www.ti.com ADC Registers

22.3.32 ADC Group1 Channel Select Register (ADG1SEL)

ADC Group1 Channel Select Register (ADG1SEL) is shown in Figure 22-54 and described in Table 22-38.

NOTE: Clearing ADG1SEL During a Conversion

Writing 0x0000 to ADG1SEL stops the Group1 conversions. This does not cause the ADC Group1 Results Memory pointer or the Group1 Threshold Register to be reset.

NOTE: Writing A Non-Zero Value To ADG1SEL During a Conversion

Writing a new value to ADG1SEL while a Channel in Group1 is being converted results in a new conversion sequence starting immediately with the highest priority channel in the new ADG1SEL selection. This also causes the ADC Group1 Results Memory pointer to be reset so that the memory allocated for storing the Group1 conversion results gets overwritten. Care should be taken to re-program the corresponding Interrupt Threshold Counter or DMA Threshold Counter again so that correct number of conversions happen before a Threshold interrupt or Block DMA request is generated.

ADC1 supports up to 32 channels and ADC2 supports up to 25 channels on the microcontroller.

Figure 22-54. ADC Group1 Channel Select Register (ADG1SEL) [offset = 7Ch]

31	0
G1_SEL	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 22-38. ADC Group1 Channel Select Register (ADG1SEL) Field Descriptions

Bit	Field	Value	Description
31-0	G1_SEL		Group1 channels selected. Any operation mode read/write:
		0	No ADC input channel is selected for conversion in the Group1.
		Non-zero	The channels marked by the bit positions that are set to 1 will be converted in ascending order when the Group1 is triggered.

```
void adcStopConversion(adcBASE_t *adc, uint32 group);
```

디지털 신호로의 변환을 끝내는 함수

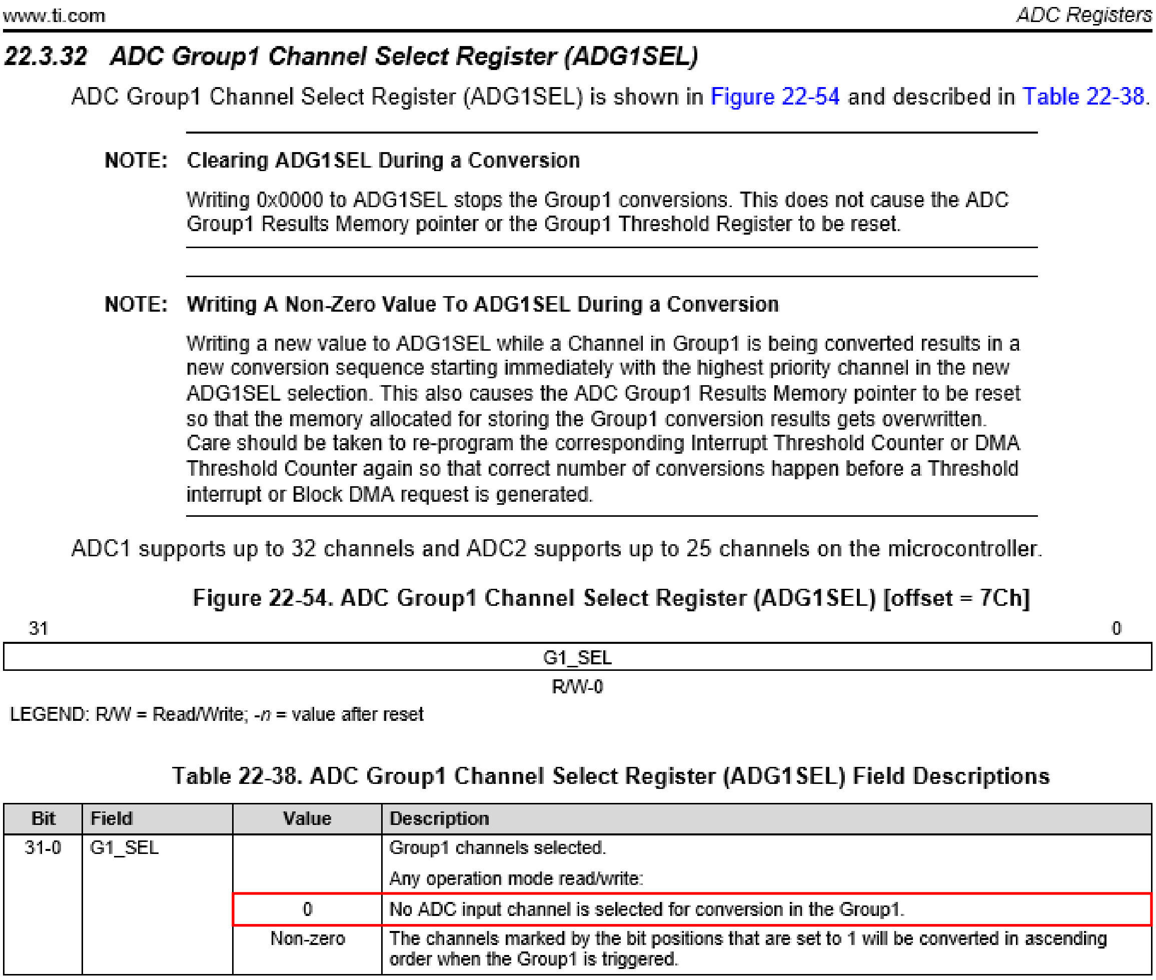
adcBASE_t *adc : ADC모듈 번호

```
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

uint32 group : ADC그룹 번호

```
- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2
```

```
void adcStopConversion(adcBASE_t *adc, uint32 group)
{

    /** - Stop Conversion */
    adc->GxSEL[group] = 0U;

}
```



www.ti.com ADC Registers

22.3.32 ADC Group1 Channel Select Register (ADG1SEL)

ADC Group1 Channel Select Register (ADG1SEL) is shown in Figure 22-54 and described in Table 22-38.

NOTE: Clearing ADG1SEL During a Conversion

Writing 0x0000 to ADG1SEL stops the Group1 conversions. This does not cause the ADC Group1 Results Memory pointer or the Group1 Threshold Register to be reset.

NOTE: Writing A Non-Zero Value To ADG1SEL During a Conversion

Writing a new value to ADG1SEL while a Channel in Group1 is being converted results in a new conversion sequence starting immediately with the highest priority channel in the new ADG1SEL selection. This also causes the ADC Group1 Results Memory pointer to be reset so that the memory allocated for storing the Group1 conversion results gets overwritten. Care should be taken to re-program the corresponding Interrupt Threshold Counter or DMA Threshold Counter again so that correct number of conversions happen before a Threshold interrupt or Block DMA request is generated.

ADC1 supports up to 32 channels and ADC2 supports up to 25 channels on the microcontroller.

Figure 22-54. ADC Group1 Channel Select Register (ADG1SEL) [offset = 7Ch]

31	0
G1_SEL	
R/W-0	

LEGEND: R/W = Read/Write; -*n* = value after reset

Table 22-38. ADC Group1 Channel Select Register (ADG1SEL) Field Descriptions

Bit	Field	Value	Description
31-0	G1_SEL		Group1 channels selected. Any operation mode read/write:
		0	No ADC input channel is selected for conversion in the Group1.
		Non-zero	The channels marked by the bit positions that are set to 1 will be converted in ascending order when the Group1 is triggered.

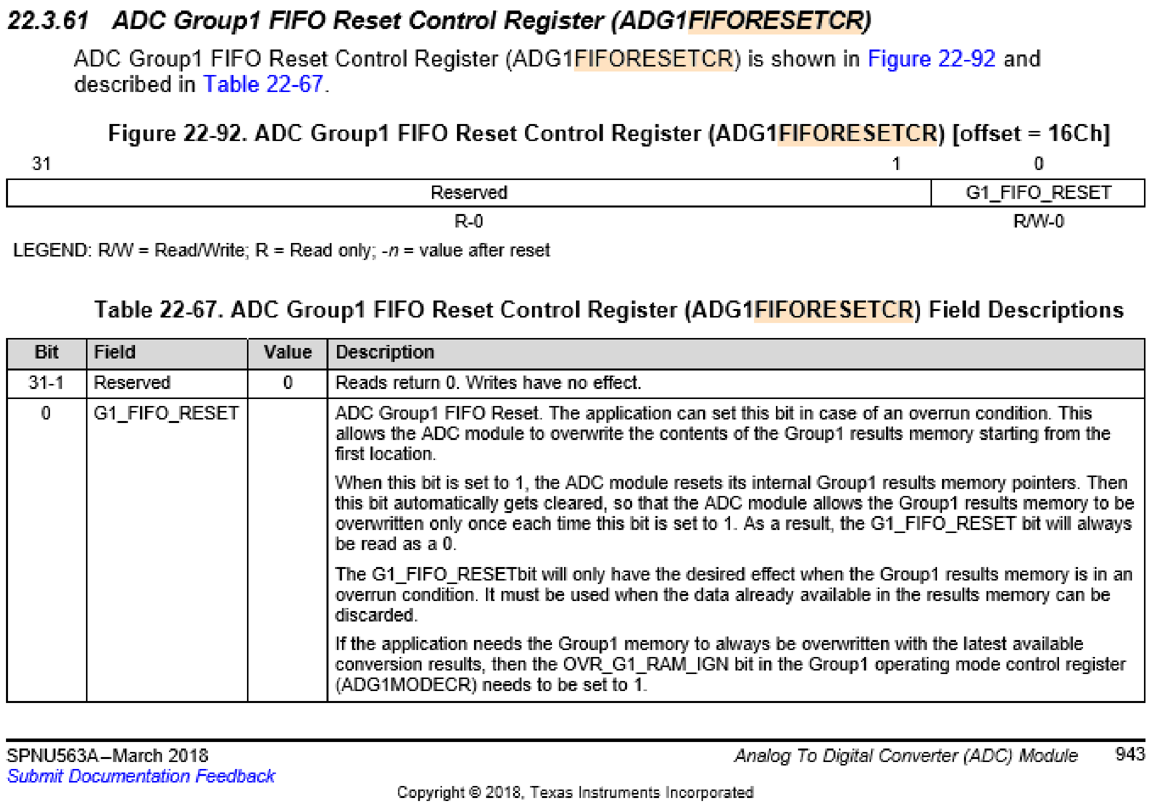
```
void adcResetFiFo(adcBASE_t *adc, uint32 group);
```

FiFo의 읽기 및 쓰기 포인터를 재설정하는 함수

```
adcBASE_t *adc : ADC모듈 번호
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

```
uint32 group : ADC그룹 번호
- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2
```

```
void adcResetFiFo(adcBASE_t *adc, uint32 group)
{

    /** - Reset FiFo */
    adc->GxFIFORESETCR[group] = 1U;

}
```



22.3.61 ADC Group1 FIFO Reset Control Register (ADG1FIFORESETCR)

ADC Group1 FIFO Reset Control Register (ADG1FIFORESETCR) is shown in Figure 22-92 and described in Table 22-67.

Figure 22-92. ADC Group1 FIFO Reset Control Register (ADG1FIFORESETCR) [offset = 16Ch]

31 … 1	0
Reserved	G1_FIFO_RESET
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22-67. ADC Group1 FIFO Reset Control Register (ADG1FIFORESETCR) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reads return 0. Writes have no effect.
0	G1_FIFO_RESET		ADC Group1 FIFO Reset. The application can set this bit in case of an overrun condition. This allows the ADC module to overwrite the contents of the Group1 results memory starting from the first location. When this bit is set to 1, the ADC module resets its internal Group1 results memory pointers. Then this bit automatically gets cleared, so that the ADC module allows the Group1 results memory to be overwritten only once each time this bit is set to 1. As a result, the G1_FIFO_RESET bit will always be read as a 0. The G1_FIFO_RESETbit will only have the desired effect when the Group1 results memory is in an overrun condition. It must be used when the data already available in the results memory can be discarded. If the application needs the Group1 memory to always be overwritten with the latest available conversion results, then the OVR_G1_RAM_IGN bit in the Group1 operating mode control register (ADG1MODECR) needs to be set to 1.

SPNU563A–March 2018 *Analog To Digital Converter (ADC) Module* 943
Submit Documentation Feedback

Copyright © 2018, Texas Instruments Incorporated

이 비트가 1로 설정되면 ADC 모듈은 Group1의 메모리 포인터를 재설정합니다.

재설정 후 비트는 자동으로 지워지므로 ADC 모듈이 Group1의 메모리를 덮어쓰는 것은 한 번씩만 가능합니다.

결과적으로 G1_FIFO_RESET 비트는 항상 0으로 읽혀집니다.

```
uint32  adcGetData(adcBASE_t *adc, uint32 group, adcData_t *data);
```

변환된 데이터를 얻어오는 함수. 변환한 데이터는 ADC 메시지 박스에 기록됨

리턴값으로 변환된 값의 수를 가짐

adcBASE_t *adc : ADC모듈 번호

```
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

uint32 group : ADC그룹 번호

```
- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2
```

adcData_t *data : ADC 변환 데이터를 저장하는 포인터

채널 id와 값을 가지고 있음

```
typedef struct adcData
{
    uint32 id;     /**< Channel/Pin Id       */
    uint16 value;  /**< Conversion data value */
} adcData_t;
```

```
uint32 adcGetData(adcBASE_t *adc, uint32 group, adcData_t * data)
{
    uint32  i;
    uint32  buf;
    uint32  mode;
    uint32  index = (adc == adcREG1) ? 0U : 1U;

         uint32  intcr_reg = adc->GxINTCR[group];
    uint32  count = (intcr_reg >= 256U) ? s_adcFiFoSize[index][group] :
(s_adcFiFoSize[index][group] - (uint32)(intcr_reg & 0xFFU));
    adcData_t *ptr = data;

    mode = (adc->OPMODECR & ADC_12_BIT_MODE);

    if(mode == ADC_12_BIT_MODE)
      {
        /** -  Get conversion data and channel/pin id */
        for (i = 0U; i < count; i++)
        {
          buf        = adc->GxBUF[group].BUF0;
                    /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters
are only allowed in this driver" */
          ptr->value = (uint16)(buf & 0xFFFU);
          ptr->id    = (uint32)((buf >> 16U) & 0x1FU);
          /*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" */
                    ptr++;
        }
      }
      else
      {
        /** -  Get conversion data and channel/pin id */
        for (i = 0U; i < count; i++)
```

```
       {
         buf        = adc->GxBUF[group].BUF0;
                  /*SAFETYMCUSW 45 D MR:21.1 <APPROVED> "Valid non NULL input parameters
are only allowed in this driver" */
         ptr->value = (uint16)(buf & 0x3FFU);
         ptr->id    = (uint32)((buf >> 10U) & 0x1FU);
         /*SAFETYMCUSW 567 S MR:17.1,17.4 <APPROVED> "Pointer increment needed" */
                  ptr++;
       }
     }

    adc->GxINTFLG[group] = 9U;

    return count;
}
```

```
uint32 adcIsFifoFull(adcBASE_t *adc, uint32 group);
```

FiFo 버퍼 상태를 확인하는 함수

반환값

0 : FiFo 버퍼가 가득 차 있지 않을 때

1 : FiFo 버퍼가 꽉 찬 경우

3 : FiFo 버퍼 오버 플로우가 발생했을 때

adcBASE_t *adc : ADC모듈 번호

```
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

uint32 group : ADC그룹 번호

```
- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2
```

```
uint32 adcIsFifoFull(adcBASE_t *adc, uint32 group)
{
    uint32 flags;

    /** - Read FiFo flags */
    flags = adc->GxINTFLG[group] & 3U;

    return flags;
}
```

uint32 **adcIsConversionComplete**(adcBASE_t *adc, uint32 group)

변환이 완료되었는지 확인하는 함수

변환 성공시 리턴값 flags에 1리턴, 실패시 0 리턴

adcBASE_t *adc : ADC모듈 번호

adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer

uint32 group : ADC그룹 번호

- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2

```
uint32 adcIsConversionComplete(adcBASE_t *adc, uint32 group)
{
    uint32 flags;

    /** - Read conversion flags */
    flags = adc->GxINTFLG[group] & 8U;

    return flags;
}
```

22.3.15 ADC Group1 Interrupt Flag Register (ADG1INTFLG)

ADC Group1 Interrupt Flag Register (ADG1INTFLG) is shown in Figure 22-37 and described in Table 22-21.

Figure 22-37. ADC Group1 Interrupt Flag Register (ADG1INTFLG) [offset = 38h]

31 … 8
Reserved
R-0

7 … 4	3	2	1	0
Reserved	G1_END	G1_MEM_ EMPTY	G1_MEM_ OVERRUN	G1_THR_ INT_FLG
R-0	R/W1C-0	R-1	R-0	R/W1C-0

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to clear; -*n* = value after reset

Table 22-21. ADC Group1 Interrupt Flag Register (ADG1INTFLG) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3	G1_END		Group1 Conversion End. This bit will be set only if the Group1 conversions are configured to be in the single-conversion mode. Any operation mode read:
		0	All the channels selected for conversion in the Group1 have not yet been converted.
		1	All the channels selected for conversion in the Group1 have been converted. A Group1 conversion end interrupt is generated, if enabled, when this bit gets set.
			This bit can be cleared by any one of the following ways: • By writing a 1 to this bit • By writing a 1 to the Group1 status register (ADG1SR) bit 0 (G1_END) • By reading one conversion result from the Group1 results' memory in the "read from FIFO" mode • By writing a new set of channels to the Group1 channel select register
2	G1_MEM_EMPTY		Group1 Results Memory Empty. This is a read-only bit; writes have no effect. It is not a source of an interrupt from the ADC module. Any operation mode read:
		0	The Group1 results memory is not empty.
		1	The Group1 results memory is empty.
1	G1_MEM_OVERRUN		Group1 Memory Overrun. This is a read-only bit; writes have no effect. Any operation mode read:
		0	Group1 results memory has not overrun.
		1	Group1 results memory has overrun.
0	G1_THR_INT_FLG		Group1 Threshold Interrupt Flag. Any operation mode read:
		0	The number of conversions completed for the Group1 is smaller than the threshold programmed in the Group1 interrupt threshold register.
		1	The number of conversions completed for the Group1 is equal to or greater than the threshold programmed in the Group1 interrupt threshold register. This bit can be cleared by writing a 1; writing a 0 has no effect.

```
void adcEnableNotification(adcBASE_t *adc, uint32 group);
```

Notification 함수를 활성화시키는 함수

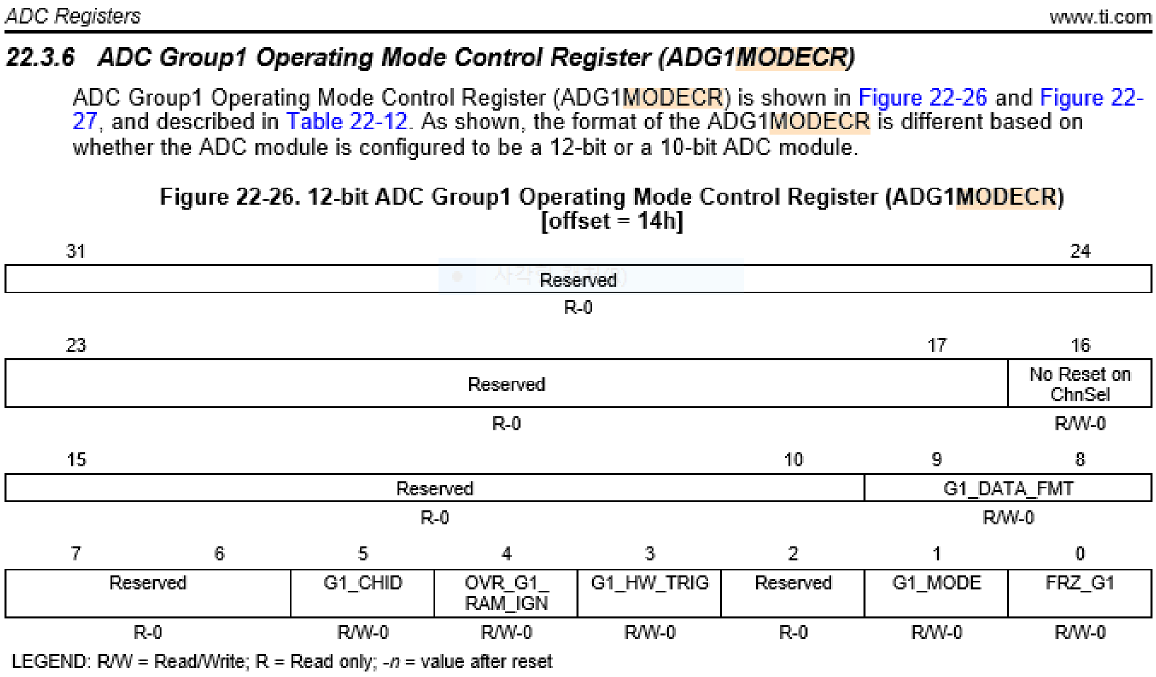
adcBASE_t *adc : ADC모듈 번호

adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer

uint32 group : ADC그룹 번호

- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2

```
void adcEnableNotification(adcBASE_t *adc, uint32 group)
{
    uint32 notif = (((uint32)(adc->GxMODECR[group]) & 2U) == 2U) ? 1U : 8U;

    adc->GxINTENA[group] = notif;

}
```



ADC Registers www.ti.com

22.3.6 ADC Group1 Operating Mode Control Register (ADG1MODECR)

ADC Group1 Operating Mode Control Register (ADG1MODECR) is shown in Figure 22-26 and Figure 22-27, and described in Table 22-12. As shown, the format of the ADG1MODECR is different based on whether the ADC module is configured to be a 12-bit or a 10-bit ADC module.

Figure 22-26. 12-bit ADC Group1 Operating Mode Control Register (ADG1MODECR) [offset = 14h]

31							24
Reserved							
R-0							

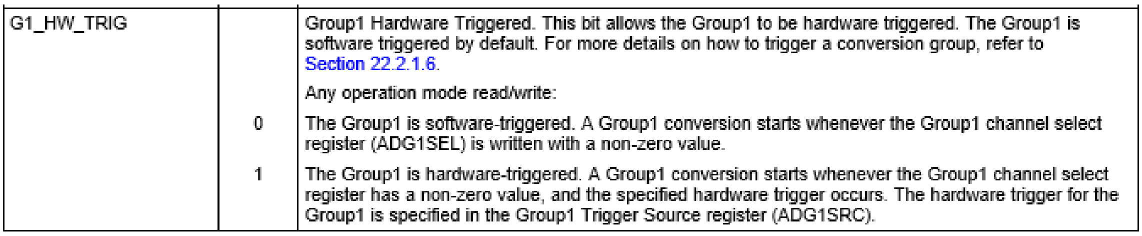
23						17	16
Reserved							No Reset on ChnSel
R-0							R/W-0

15					10	9	8
Reserved						G1_DATA_FMT	
R-0						R/W-0	

7	6	5	4	3	2	1	0
Reserved		G1_CHID	OVR_G1_RAM_IGN	G1_HW_TRIG	Reserved	G1_MODE	FRZ_G1
R-0		R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

8U : G1_HW_TRIG

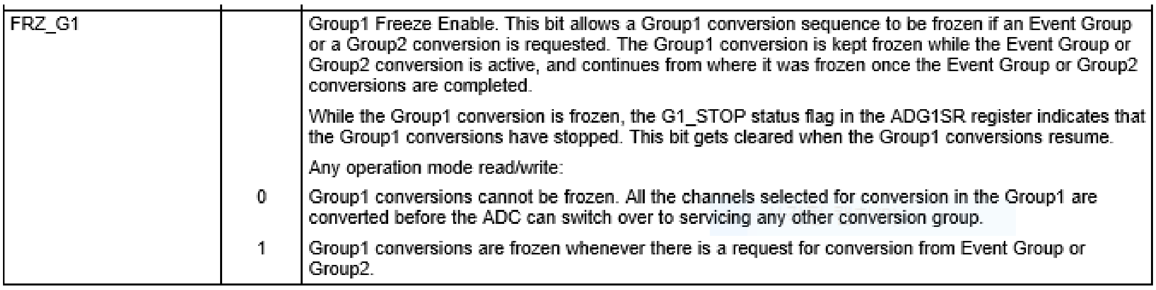


G1_HW_TRIG		Group1 Hardware Triggered. This bit allows the Group1 to be hardware triggered. The Group1 is software triggered by default. For more details on how to trigger a conversion group, refer to Section 22.2.1.6. Any operation mode read/write:
	0	The Group1 is software-triggered. A Group1 conversion starts whenever the Group1 channel select register (ADG1SEL) is written with a non-zero value.
	1	The Group1 is hardware-triggered. A Group1 conversion starts whenever the Group1 channel select register has a non-zero value, and the specified hardware trigger occurs. The hardware trigger for the Group1 is specified in the Group1 Trigger Source register (ADG1SRC).

이 비트를 사용하면 Group1을 하드웨어로 트리거 할 수 있습니다.

Group1은 하드웨어에 의해 트리거 됩니다. 그룹 1 채널 선택 레지스터가 0이 아닌 값을 가지며 지정된 하드웨어 트리거가 발생할 때마다 그룹 1 변환이 시작됩니다. Group1의 하드웨어 트리거는 Group1 Trigger Source 레지스터 (ADG1SRC)에 지정됩니다.

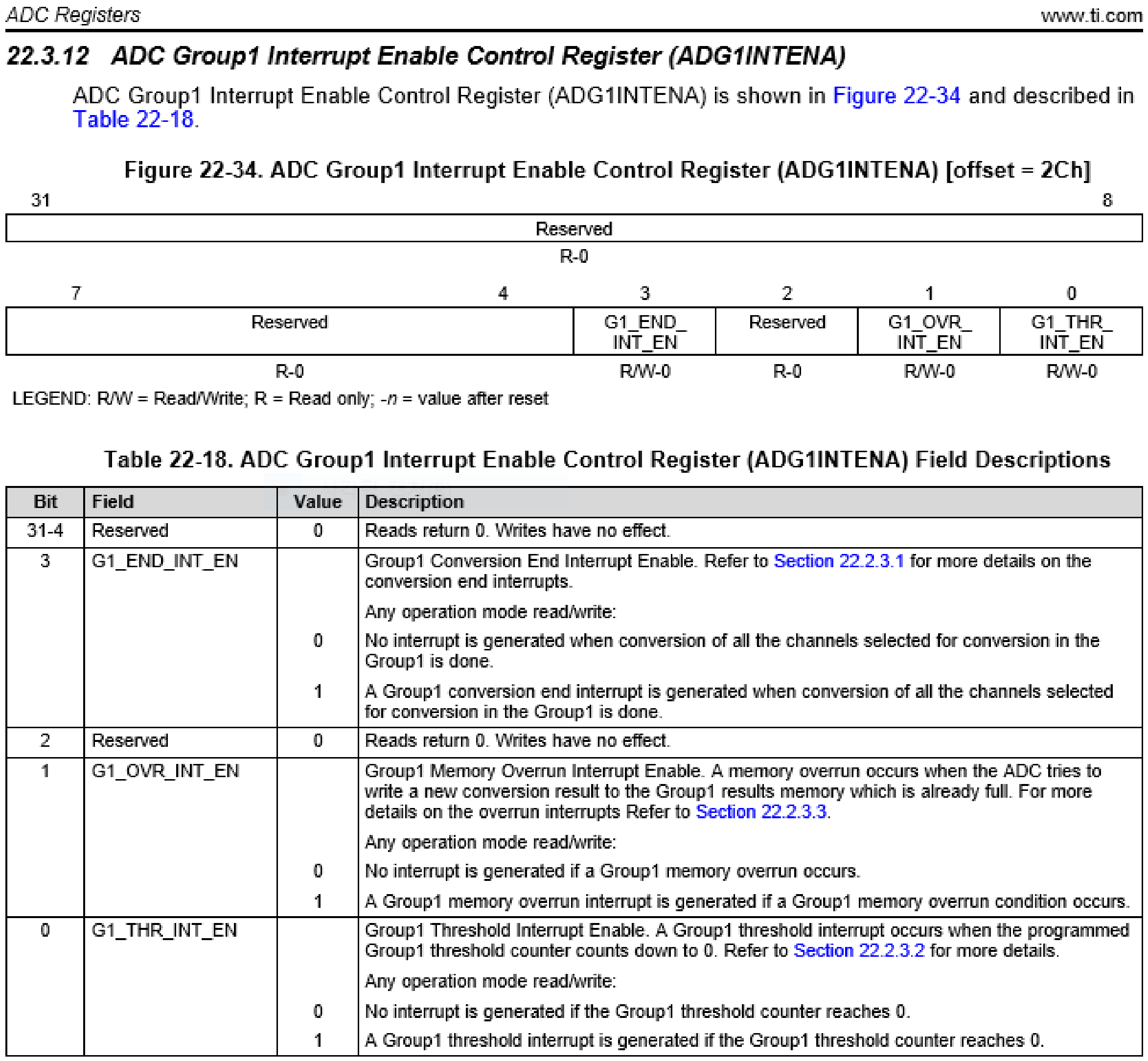
1U : FRZ_G1



FRZ_G1		Group1 Freeze Enable. This bit allows a Group1 conversion sequence to be frozen if an Event Group or a Group2 conversion is requested. The Group1 conversion is kept frozen while the Event Group or Group2 conversion is active, and continues from where it was frozen once the Event Group or Group2 conversions are completed. While the Group1 conversion is frozen, the G1_STOP status flag in the ADG1SR register indicates that the Group1 conversions have stopped. This bit gets cleared when the Group1 conversions resume. Any operation mode read/write:
	0	Group1 conversions cannot be frozen. All the channels selected for conversion in the Group1 are converted before the ADC can switch over to servicing any other conversion group.
	1	Group1 conversions are frozen whenever there is a request for conversion from Event Group or Group2.

이 비트를 사용하면 Group1 변환 시퀀스를 고정할 수 있습니다.
1 Group1 변환은 Event Group 또는 Group2에서 변환 요청이 있을 때마다 고정됩니다.

ADG1MODECR의 형식은 ADC 모듈의 구성여부가 12 비트인지 10비트인지에 따라 다르다.



ADC Registers www.ti.com

22.3.12 ADC Group1 Interrupt Enable Control Register (ADG1INTENA)

ADC Group1 Interrupt Enable Control Register (ADG1INTENA) is shown in Figure 22-34 and described in Table 22-18.

Figure 22-34. ADC Group1 Interrupt Enable Control Register (ADG1INTENA) [offset = 2Ch]

31	8
Reserved	
R-0	

7	4	3	2	1	0
Reserved		G1_END_INT_EN	Reserved	G1_OVR_INT_EN	G1_THR_INT_EN
R-0		R/W-0	R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22-18. ADC Group1 Interrupt Enable Control Register (ADG1INTENA) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3	G1_END_INT_EN		Group1 Conversion End Interrupt Enable. Refer to Section 22.2.3.1 for more details on the conversion end interrupts.
			Any operation mode read/write:
		0	No interrupt is generated when conversion of all the channels selected for conversion in the Group1 is done.
		1	A Group1 conversion end interrupt is generated when conversion of all the channels selected for conversion in the Group1 is done.
2	Reserved	0	Reads return 0. Writes have no effect.
1	G1_OVR_INT_EN		Group1 Memory Overrun Interrupt Enable. A memory overrun occurs when the ADC tries to write a new conversion result to the Group1 results memory which is already full. For more details on the overrun interrupts Refer to Section 22.2.3.3.
			Any operation mode read/write:
		0	No interrupt is generated if a Group1 memory overrun occurs.
		1	A Group1 memory overrun interrupt is generated if a Group1 memory overrun condition occurs.
0	G1_THR_INT_EN		Group1 Threshold Interrupt Enable. A Group1 threshold interrupt occurs when the programmed Group1 threshold counter counts down to 0. Refer to Section 22.2.3.2 for more details.
			Any operation mode read/write:
		0	No interrupt is generated if the Group1 threshold counter reaches 0.
		1	A Group1 threshold interrupt is generated if the Group1 threshold counter reaches 0.

Notif == 8 : 3번비트 1로 세팅

Group1의 모든 채널 변환이 완료되면 변환 종료 인터럽트가 발생합니다.

Notif == 1 : 0번비트 1로 세팅

Group1 임계 값 카운터가 0에 도달하면 인터럽트가 생성됩니다.

```
void adcDisableNotification(adcBASE_t *adc, uint32 group)
{

    adc->GxINTENA[group] = 0U;

}
```

Notification 함수를 비활성화시키는 함수

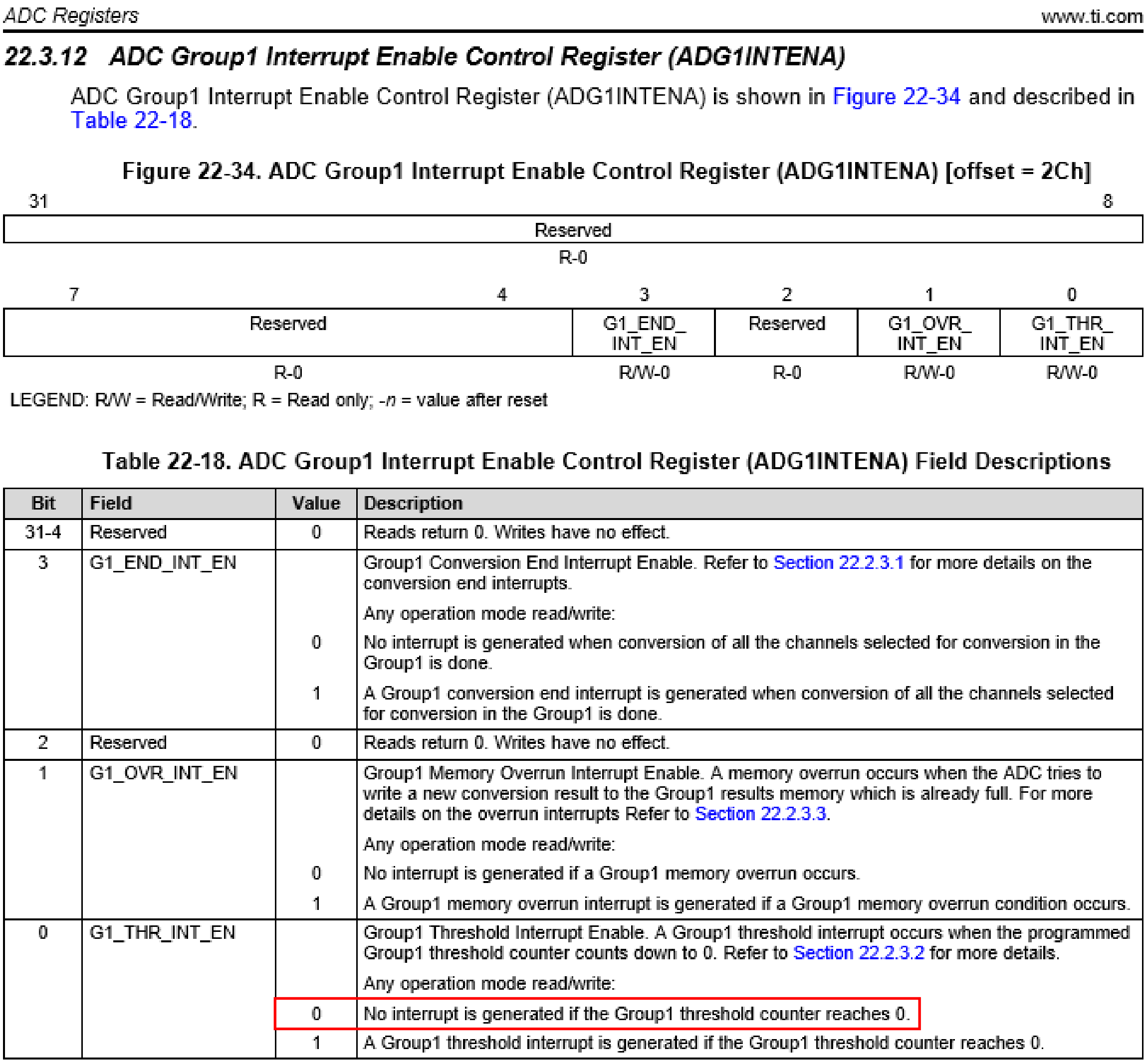
adcBASE_t *adc : ADC모듈 번호

adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer

uint32 group : ADC그룹 번호

- adcGROUP0: ADC event group
- adcGROUP1: ADC group 1
- adcGROUP2: ADC group 2

```
void adcDisableNotification(adcBASE_t *adc, uint32 group)
{
/* USER CODE BEGIN (33) */
/* USER CODE END */

    adc->GxINTENA[group] = 0U;

    /**   @note The function adcInit has to be called before this function can be used. */

/* USER CODE BEGIN (34) */
/* USER CODE END */
}
```



ADC Registers www.ti.com

22.3.12 ADC Group1 Interrupt Enable Control Register (ADG1INTENA)

ADC Group1 Interrupt Enable Control Register (ADG1INTENA) is shown in Figure 22-34 and described in Table 22-18.

Figure 22-34. ADC Group1 Interrupt Enable Control Register (ADG1INTENA) [offset = 2Ch]

31	8
Reserved	
R-0	

7	4	3	2	1	0
Reserved		G1_END_ INT_EN	Reserved	G1_OVR_ INT_EN	G1_THR_ INT_EN
R-0		R/W-0	R-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22-18. ADC Group1 Interrupt Enable Control Register (ADG1INTENA) Field Descriptions

Bit	Field	Value	Description
31-4	Reserved	0	Reads return 0. Writes have no effect.
3	G1_END_INT_EN		Group1 Conversion End Interrupt Enable. Refer to Section 22.2.3.1 for more details on the conversion end interrupts.
			Any operation mode read/write:
		0	No interrupt is generated when conversion of all the channels selected for conversion in the Group1 is done.
		1	A Group1 conversion end interrupt is generated when conversion of all the channels selected for conversion in the Group1 is done.
2	Reserved	0	Reads return 0. Writes have no effect.
1	G1_OVR_INT_EN		Group1 Memory Overrun Interrupt Enable. A memory overrun occurs when the ADC tries to write a new conversion result to the Group1 results memory which is already full. For more details on the overrun interrupts Refer to Section 22.2.3.3.
			Any operation mode read/write:
		0	No interrupt is generated if a Group1 memory overrun occurs.
		1	A Group1 memory overrun interrupt is generated if a Group1 memory overrun condition occurs.
0	G1_THR_INT_EN		Group1 Threshold Interrupt Enable. A Group1 threshold interrupt occurs when the programmed Group1 threshold counter counts down to 0. Refer to Section 22.2.3.2 for more details.
			Any operation mode read/write:
		0	No interrupt is generated if the Group1 threshold counter reaches 0.
		1	A Group1 threshold interrupt is generated if the Group1 threshold counter reaches 0.

Group1 임계 값 카운터가 0에 도달하면 인터럽트가 불가능합니다

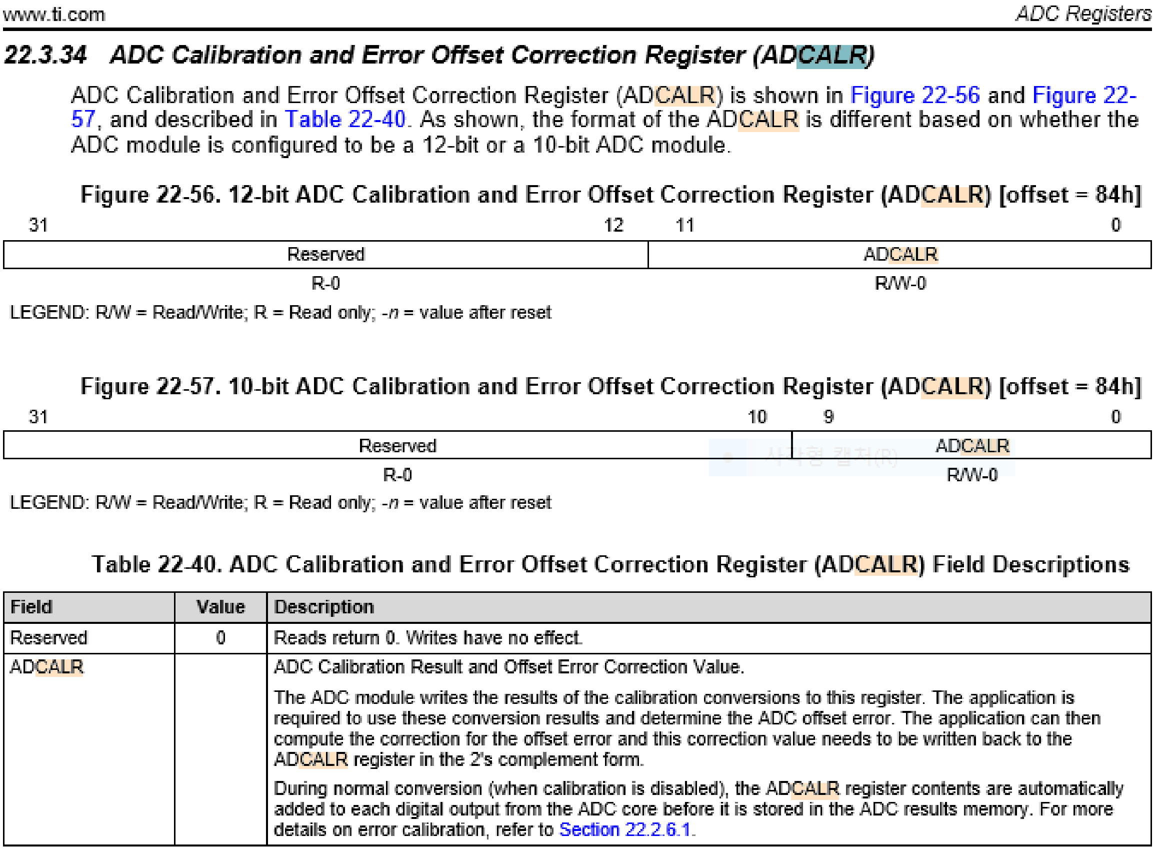
```
void adcCalibration(adcBASE_t *adc)
{
```

ADC 모듈 교정 함수

보정 모드를 사용하여 오프셋 오류를 계산하며, 교정변환 결과는 ADCALR register에 기록된다

```
adcBASE_t *adc : ADC모듈 번호
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

```
void adcCalibration(adcBASE_t *adc)
{
        uint32 conv_val[5U]={0U,0U,0U,0U,0U};
        uint32 loop_index=0U;
        uint32 offset_error=0U;
        uint32 backup_mode;

        /** - Backup Mode before Calibration  */
        backup_mode = adc->OPMODECR;

        /** - Enable 12-BIT ADC  */
        adc->OPMODECR |= 0x80000000U;

        /* Disable all channels for conversion */
        adc->GxSEL[0U]=0x00U;
        adc->GxSEL[1U]=0x00U;
        adc->GxSEL[2U]=0x00U;

        for(loop_index=0U;loop_index<4U;loop_index++)
        {
                /* Disable Self Test and Calibration mode */

        /*
        응용 프로그램은 자체 테스트 모드가 비활성화 될 때마다 (SELF_TEST = 0) 교정 시퀀스를
        활성화할 수 있습니다.
        */

                adc->CALCR=0x0U;

                switch(loop_index)
                {
                        case 0U :       /* Test 1 : Bride En = 0 , HiLo =0 */
                                        adc->CALCR=0x0U;
                                        break;

                        case 1U :       /* Test 1 : Bride En = 0 , HiLo =1 */
                                        adc->CALCR=0x0100U;
                                        break;

                        case 2U :       /* Test 1 : Bride En = 1 , HiLo =0 */
                                        adc->CALCR=0x0200U;
                                        break;

                        case 3U :       /* Test 1 : Bride En = 1 , HiLo =1 */
                                        adc->CALCR=0x0300U;
                                        break;
                        default :
                                    break;
                }
                /* Enable Calibration mode */
                adc->CALCR|=0x1U;

                /* Start calibration conversion */
                adc->CALCR|=0x00010000U;

                /* Wait for calibration conversion to complete */
                /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Hardware status bit read check" */
                while((adc->CALCR & 0x00010000U)==0x00010000U)
            {
            } /* Wait */

                /* Read converted value */
                conv_val[loop_index]= adc->CALR;
        }

        /* Disable Self Test and Calibration mode */
        adc->CALCR=0x0U;

        /* Compute the Offset error correction value */
        conv_val[4U]=conv_val[0U]+ conv_val[1U] + conv_val[2U] + conv_val[3U];

        conv_val[4U]=(conv_val[4U]/4U);

        offset_error=conv_val[4U]-0x7FFU;

        /*Write the offset error to the Calibration register */
        /* Load 2;s complement of the computed value to ADCALR register */
        offset_error=~offset_error;
        offset_error=offset_error & 0xFFFU;
        offset_error=offset_error+1U;

        /*
            ADC 모듈은 교정 변환 결과를 이 레지스터에 씁니다.
        */

        adc->CALR = offset_error;

  /** - Restore Mode after Calibration  */
        adc->OPMODECR = backup_mode;

}
```



www.ti.com ADC Registers

22.3.34 ADC Calibration and Error Offset Correction Register (ADCALR)

ADC Calibration and Error Offset Correction Register (ADCALR) is shown in Figure 22-56 and Figure 22-57, and described in Table 22-40. As shown, the format of the ADCALR is different based on whether the ADC module is configured to be a 12-bit or a 10-bit ADC module.

Figure 22-56. 12-bit ADC Calibration and Error Offset Correction Register (ADCALR) [offset = 84h]

31 12	11 0
Reserved	ADCALR
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Figure 22-57. 10-bit ADC Calibration and Error Offset Correction Register (ADCALR) [offset = 84h]

31 10	9 0
Reserved	ADCALR
R-0	R/W-0

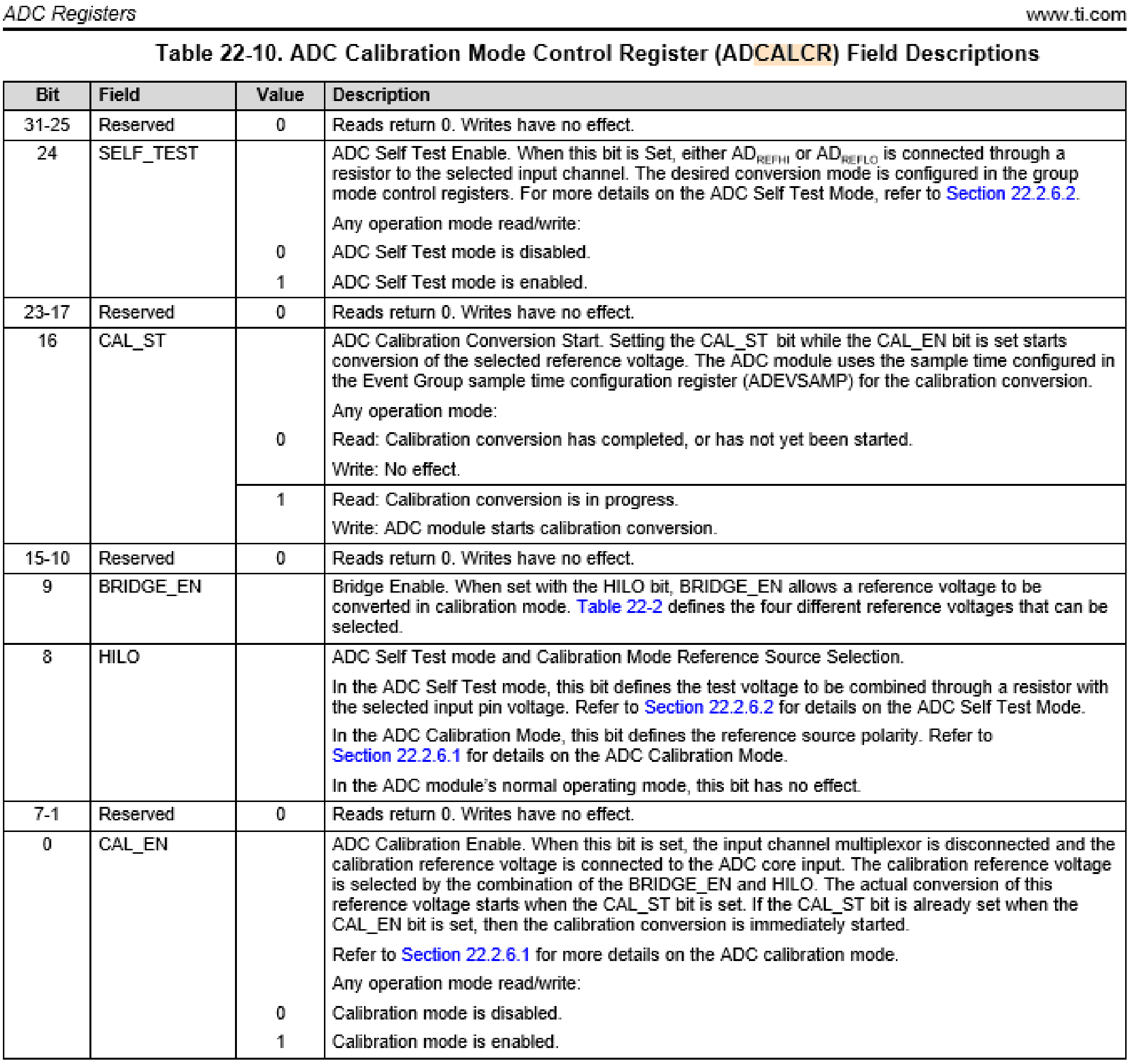
LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22-40. ADC Calibration and Error Offset Correction Register (ADCALR) Field Descriptions

Field	Value	Description
Reserved	0	Reads return 0. Writes have no effect.
ADCALR		ADC Calibration Result and Offset Error Correction Value. The ADC module writes the results of the calibration conversions to this register. The application is required to use these conversion results and determine the ADC offset error. The application can then compute the correction for the offset error and this correction value needs to be written back to the ADCALR register in the 2's complement form. During normal conversion (when calibration is disabled), the ADCALR register contents are automatically added to each digital output from the ADC core before it is stored in the ADC results memory. For more details on error calibration, refer to Section 22.2.6.1.

ADC 모듈은 교정 변환 결과를 이 레지스터에 씁니다. 애플리케이션은 이러한 변환 결과를 사용하고 ADC 오프셋 오류를 결정해야합니다. 그런 다음 애플리케이션은 오프셋 오류에 대한 보정을 계산할 수 있으며 이 보정 값을 2의 보수 형식으로 ADCALR 레지스터에 다시 써야합니다.

정상 변환 중 (보정이 비활성화 된 경우) ADCALR 레지스터 내용은 ADC 결과 메모리에 저장되기 전에 ADC 코어의 각 디지털 출력에 자동으로 추가됩니다.



ADC Registers www.ti.com

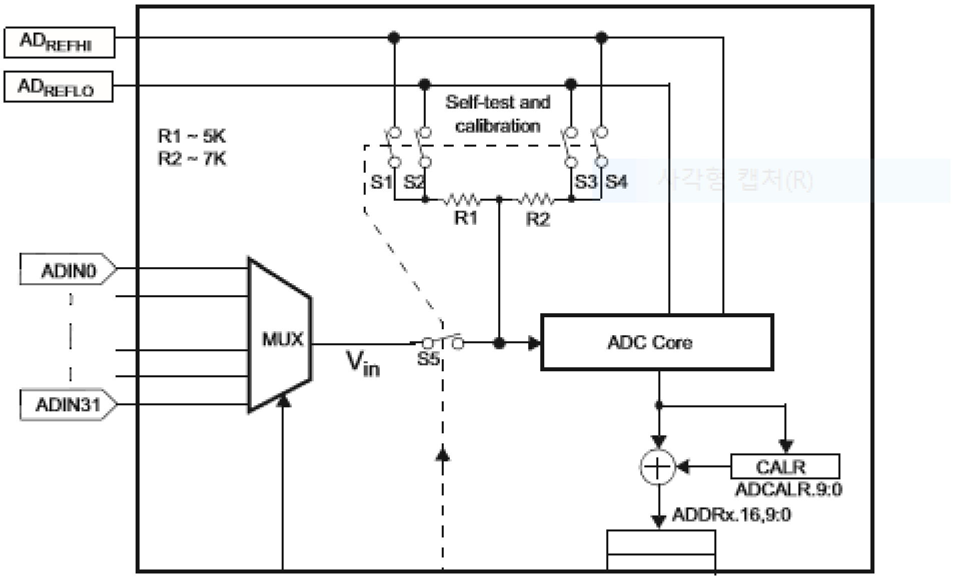
Table 22-10. ADC Calibration Mode Control Register (ADCALCR) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reads return 0. Writes have no effect.
24	SELF_TEST		ADC Self Test Enable. When this bit is Set, either AD_{REFHI} or AD_{REFLO} is connected through a resistor to the selected input channel. The desired conversion mode is configured in the group mode control registers. For more details on the ADC Self Test Mode, refer to Section 22.2.6.2. Any operation mode read/write:
		0	ADC Self Test mode is disabled.
		1	ADC Self Test mode is enabled.
23-17	Reserved	0	Reads return 0. Writes have no effect.
16	CAL_ST		ADC Calibration Conversion Start. Setting the CAL_ST bit while the CAL_EN bit is set starts conversion of the selected reference voltage. The ADC module uses the sample time configured in the Event Group sample time configuration register (ADEVSAMP) for the calibration conversion. Any operation mode:
		0	Read: Calibration conversion has completed, or has not yet been started. Write: No effect.
		1	Read: Calibration conversion is in progress. Write: ADC module starts calibration conversion.
15-10	Reserved	0	Reads return 0. Writes have no effect.
9	BRIDGE_EN		Bridge Enable. When set with the HILO bit, BRIDGE_EN allows a reference voltage to be converted in calibration mode. Table 22-2 defines the four different reference voltages that can be selected.
8	HILO		ADC Self Test mode and Calibration Mode Reference Source Selection. In the ADC Self Test mode, this bit defines the test voltage to be combined through a resistor with the selected input pin voltage. Refer to Section 22.2.6.2 for details on the ADC Self Test Mode. In the ADC Calibration Mode, this bit defines the reference source polarity. Refer to Section 22.2.6.1 for details on the ADC Calibration Mode. In the ADC module's normal operating mode, this bit has no effect.
7-1	Reserved	0	Reads return 0. Writes have no effect.
0	CAL_EN		ADC Calibration Enable. When this bit is set, the input channel multiplexor is disconnected and the calibration reference voltage is connected to the ADC core input. The calibration reference voltage is selected by the combination of the BRIDGE_EN and HILO. The actual conversion of this reference voltage starts when the CAL_ST bit is set. If the CAL_ST bit is already set when the CAL_EN bit is set, then the calibration conversion is immediately started. Refer to Section 22.2.6.1 for more details on the ADC calibration mode. Any operation mode read/write:
		0	Calibration mode is disabled.
		1	Calibration mode is enabled.

보정 모드는 CAL_EN 비트 (ADCALCR.0)를 설정하여 활성화됩니다. 응용 프로그램은 보정 모드가 활성화 되어있을 때 변환 그룹이 서비스되고 있는지 확인해야합니다.

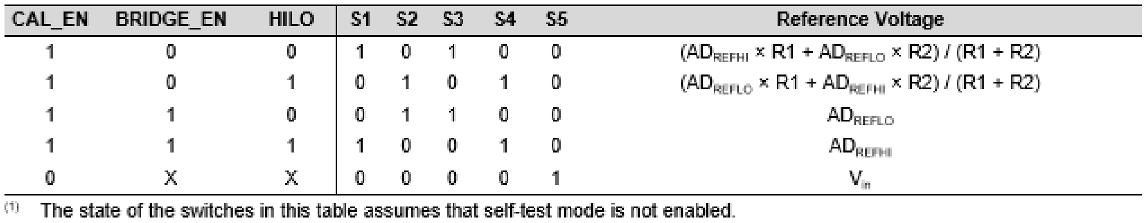
입력 멀티플렉서는 비활성화 되고 기준 전압 만이 ADC 코어 입력에 연결된다. 그림 22-13의 스위치 S5가 열립니다. 또한 변환으로 발생한 디지털 결과는 ADC 코어에서 보정 및 오프셋 오류 보정 레지스터 ADCALR로 출력됩니다. ADC 결과의 메모리는 보정 변환의 영향을 받지 않습니다. 보정 모드가 비활성화되면 ADC는 정상적인 변환을 위해 구성될 수 있습니다.

Figure 22-13. Self-Test and Calibration Logic



ADC 교정을 위한 측정 횟수와 측정 소스는 애플리케이션에 따라 다릅니다. 측정할 각 교정 소스에 대해 CAL_ST 비트를 설정해야 합니다. 보정 모드가 활성화 되어있는 동안 사용 가능한 모든 교정 소스는 BRIDGE_EN 및 HILO 비트에 따라 변환될 수 있습니다 (표 22-2 참조).

Table 22-2. Calibration Reference Voltages[1]



CAL_EN	BRIDGE_EN	HILO	S1	S2	S3	S4	S5	Reference Voltage
1	0	0	1	0	1	0	0	$(AD_{REFHI} \times R1 + AD_{REFLO} \times R2) / (R1 + R2)$
1	0	1	0	1	0	1	0	$(AD_{REFLO} \times R1 + AD_{REFHI} \times R2) / (R1 + R2)$
1	1	0	0	1	1	0	0	AD_{REFLO}
1	1	1	1	0	0	1	0	AD_{REFHI}
0	X	X	0	0	0	0	1	V_{in}

[1] The state of the switches in this table assumes that self-test mode is not enabled.

조정 값의 디지털 결과는 각 참조 변환 후에 ADCALR에서 응용 프로그램에 의해 읽혀 져야 수정 값을 계산하여 ADCALR에 다시 기록할 수 있습니다.

애플리케이션에 필요한 교정 데이터가 있으면 옵셋 오류 보정 값을 계산하여 보정 및 보정 레지스터 ADCALR에로드해야합니다. CAL_EN 비트가 해제된 후 정상적인 변환 모드가 다시 시작되고 정지된 지점부터 계속되지만 자체 교정 데이터가 추가됩니다.

정상 모드에서 자체 교정 시스템은 각 그룹의 FIFO에 기록되기 전에 ADCALR에 저장된 보정 값을 각 디지털 결과에 추가합니다.

기본 교정 루틴

1. CAL_EN (ADCALCR.0)을 통해 교정을 활성화

2. BRIDGE_EN과 HILO를 통해 전압 소스를 선택 (ADCALCR.9 : 8).

3. CAL_ST (ADCALCR.16)로 변환을 시작

4. CAL_ST가 0이 될 때까지 대기

5. ADCALR에서 결과를 가져와 메모리에 저장

6. 원하는 기준 전압에 대해 교정 변환 데이터가 수집될 때까지 2 단계로 반복합니다.

7. 메모리에 저장된 보정 데이터를 사용하여 오류 보정 값을 계산합니다.

8. ADCALR 레지스터에 계산된 오류 보정 값의 2를 더합니다.

9. 교정 모드를 비활성화

이 시점에서 ADC는 정상 작동하도록 구성할 수 있으며 ADCALR에로드 된 오류 수정 값으로 각 디지털 결과를 수정

```
uint32 adcMidPointCalibration(adcBASE_t *adc);
```

ADC 모듈 교정 함수

중간 지점 교정 모드를 사용하여 오프셋 오류를 계산하며, 교정변환 결과는 ADCALR register에 기록된다.

adcPointCalibration 함수보다 정확성이 더 뛰어나다.

```
adcBASE_t *adc : ADC모듈 번호
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

```
uint32 adcMidPointCalibration(adcBASE_t *adc)
{
/* USER CODE BEGIN (27) */
/* USER CODE END */

        uint32 conv_val[3U]={0U,0U,0U};
        uint32 loop_index=0U;
        uint32 offset_error=0U;
        uint32 backup_mode;

        /** - Backup Mode before Calibration  */
        backup_mode = adc->OPMODECR;

        /** - Enable 12-BIT ADC  */
        adc->OPMODECR |= 0x80000000U;

        /* Disable all channels for conversion */
        adc->GxSEL[0U]=0x00U;
        adc->GxSEL[1U]=0x00U;
        adc->GxSEL[2U]=0x00U;

        for(loop_index=0U;loop_index<2U;loop_index++)
        {
                /* Disable Self Test and Calibration mode */
                adc->CALCR=0x0U;

                switch(loop_index)
                {
                        case 0U :       /* Test 1 : Bride En = 0 , HiLo =0 */
                                        adc->CALCR=0x0U;
                                        break;

                        case 1U :       /* Test 1 : Bride En = 0 , HiLo =1 */
                                        adc->CALCR=0x0100U;
                                        break;

                        default :
                                   break;

                }

                /* Enable Calibration mode */
                adc->CALCR|=0x1U;

                /* Start calibration conversion */
                adc->CALCR|=0x00010000U;

                /* Wait for calibration conversion to complete */
                /*SAFETYMCUSW 28 D MR:NA <APPROVED> "Hardware status bit read check" */
                while((adc->CALCR & 0x00010000U)==0x00010000U)
            {
            } /* Wait */

                /* Read converted value */
                conv_val[loop_index]= adc->CALR;
        }

        /* Disable Self Test and Calibration mode */
        adc->CALCR=0x0U;

        /* Compute the Offset error correction value */
        conv_val[2U]=(conv_val[0U])+ (conv_val[1U]);

        conv_val[2U]=(conv_val[2U]/2U);

        offset_error=conv_val[2U]-0x7FFU;

        /* Write the offset error to the Calibration register          */
        /* Load 2's complement of the computed value to ADCALR register */
        offset_error=~offset_error;
        offset_error=offset_error+1U;
        offset_error=offset_error & 0xFFFU;

        adc->CALR = offset_error;

  /** - Restore Mode after Calibration  */
        adc->OPMODECR = backup_mode;

        return(offset_error);

}
```

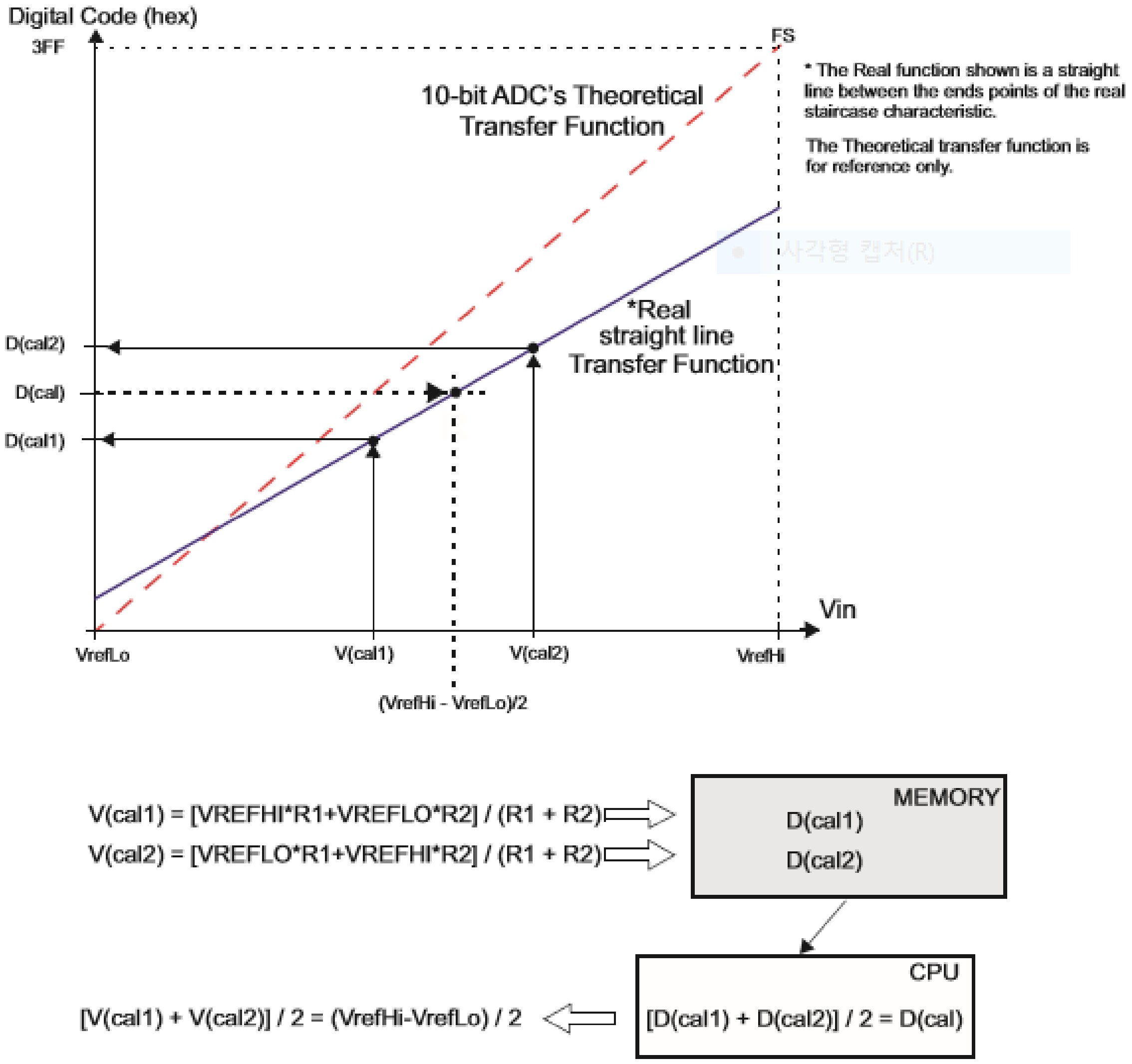
스위칭 기준 전압 장치는 중간 지점 전압의 차동 측정을 지원하도록 설계되어 있어 정확성이 보장되므로 교정 효율성이 보장됩니다. (차동 중간 점 교정은 소프트웨어로 제어)

1. 애플리케이션 프로그램은 전압 VrefHi를 R1에, VrefLo를 R2에 연결하고 (BRIDGE_EN = 0, HILO = 0), 입력 전압 V (cal1)의 변환을 시작한다. 디지털 결과 D (cal1)를 메모리에 저장한다.

2. 응용 프로그램은 전압 VrefHi를 R2로 변경하고 VrefLo를 R1 (BRIDGE_EN = 0, HILO = 1)로 변환하고 이 새 입력 전압 V (cal2)를 변환한 다음 발행된 디지털 결과 D (cal2)를 다시 메모리에 저장합니다.

3. 실제 중간 점의 실제 값은 이 두 결과의 평균을 계산하여 구합니다. [D (cal1) + D (cal2)] / 2;

Figure 22-14. Mid-point Value Calculation


Digital Code (hex)
3FF
FS
10-bit ADC's Theoretical Transfer Function
* The Real function shown is a straight line between the ends points of the real staircase characteristic.
The Theoretical transfer function is for reference only.
사각형 캡처(R)
*Real straight line Transfer Function
D(cal2)
D(cal)
D(cal1)
Vin
VrefLo
V(cal1)
V(cal2)
VrefHi
(VrefHi - VrefLo)/2
V(cal1) = [VREFHI*R1+VREFLO*R2] / (R1 + R2)
V(cal2) = [VREFLO*R1+VREFHI*R2] / (R1 + R2)
MEMORY
D(cal1)
D(cal2)
CPU
[D(cal1) + D(cal2)] / 2 = D(cal)
[V(cal1) + V(cal2)] / 2 = (VrefHi-VrefLo) / 2

(추가예정)

```
void adcSetEVTPin(adcBASE_t *adc, uint32 value);
```

출력 핀으로 구성된 ADC EVT 핀을 설정하는 함수

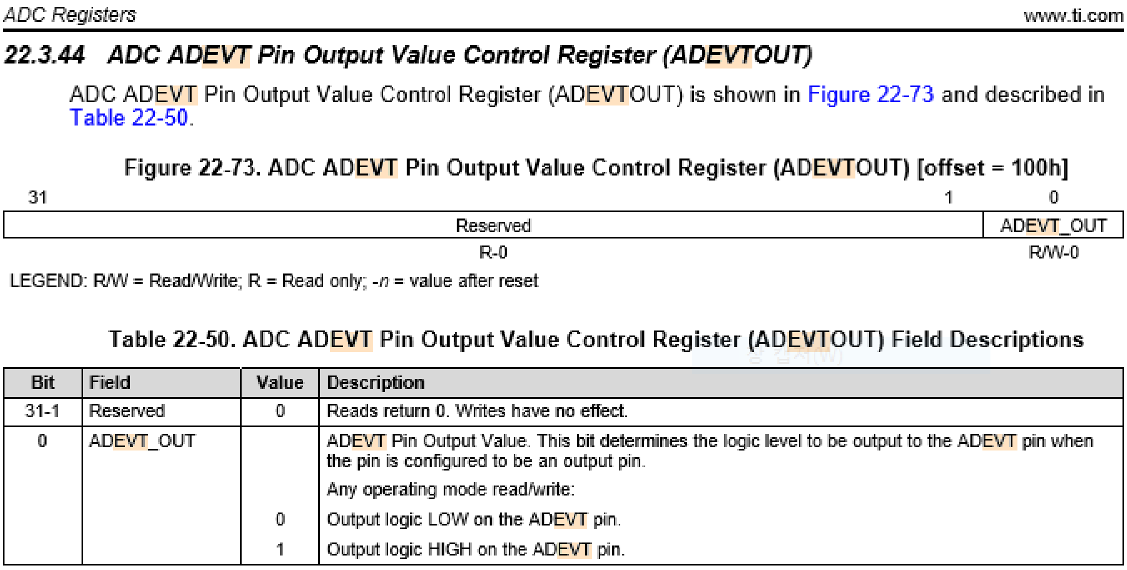
(External event pin(ADEVT) : 외부 이벤트 핀)

adcBASE_t *adc : ADC모듈 번호

adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer

uint32 value : 0 혹은 1.

```
void adcSetEVTPin(adcBASE_t *adc, uint32 value)
{

    adc->EVTOUT = value;

}
```



ADC Registers   www.ti.com

22.3.44 ADC ADEVT Pin Output Value Control Register (ADEVTOUT)

ADC ADEVT Pin Output Value Control Register (ADEVTOUT) is shown in Figure 22-73 and described in Table 22-50.

Figure 22-73. ADC ADEVT Pin Output Value Control Register (ADEVTOUT) [offset = 100h]

31–1	0
Reserved	ADEVT_OUT
R-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 22-50. ADC ADEVT Pin Output Value Control Register (ADEVTOUT) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reads return 0. Writes have no effect.
0	ADEVT_OUT		ADEVT Pin Output Value. This bit determines the logic level to be output to the ADEVT pin when the pin is configured to be an output pin. Any operating mode read/write:
		0	Output logic LOW on the ADEVT pin.
		1	Output logic HIGH on the ADEVT pin.

```
uint32 adcGetEVTPin(adcBASE_t *adc);
```

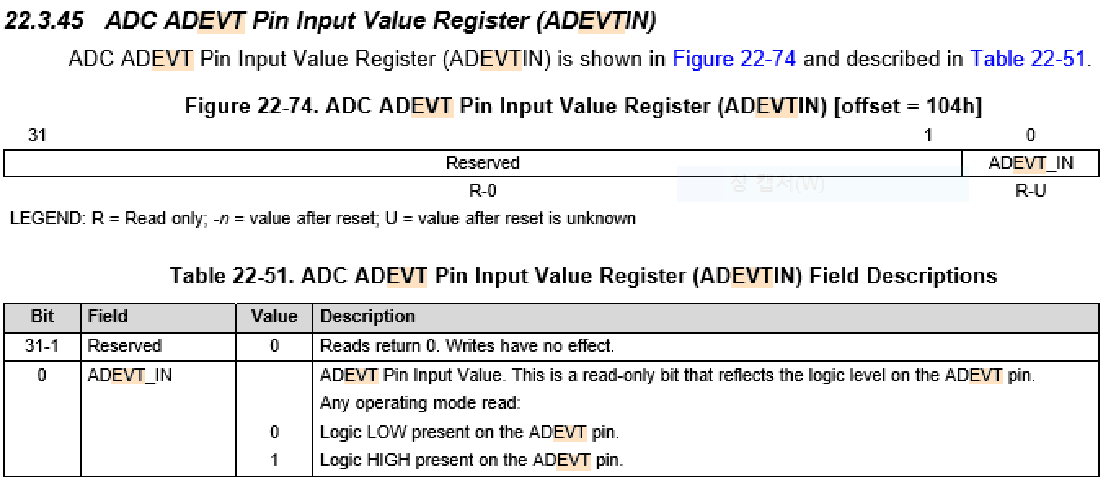
ADC EVT 핀의 값을 반환하는 함수

(External event pin(ADEVT) : 외부 이벤트 핀)

adcBASE_t *adc : ADC모듈 번호

```
adcREG1: ADC1 module pointer
adcREG2: ADC2 module pointer
```

```
uint32 adcGetEVTPin(adcBASE_t *adc)
{
    return adc->EVTIN;
}
```



22.3.45 ADC ADEVT Pin Input Value Register (ADEVTIN)

ADC ADEVT Pin Input Value Register (ADEVTIN) is shown in Figure 22-74 and described in Table 22-51.

Figure 22-74. ADC ADEVT Pin Input Value Register (ADEVTIN) [offset = 104h]

31 … 1	0
Reserved	ADEVT_IN
R-0	R-U

LEGEND: R = Read only; -*n* = value after reset; U = value after reset is unknown

Table 22-51. ADC ADEVT Pin Input Value Register (ADEVTIN) Field Descriptions

Bit	Field	Value	Description
31-1	Reserved	0	Reads return 0. Writes have no effect.
0	ADEVT_IN		ADEVT Pin Input Value. This is a read-only bit that reflects the logic level on the ADEVT pin. Any operating mode read:
		0	Logic LOW present on the ADEVT pin.
		1	Logic HIGH present on the ADEVT pin.

`void adc1GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type);`

초기 값 또는 현재 값을 복사하는 함수

adc_config_reg_t *config_reg : 초기 또는 현재의 구조체에 대한 포인터

config_value_type_t type : 설정 레지스터의 초기 값 또는 현재 값을 저장할 필요가 있는지 여부

```
void adc1GetConfigValue(adc_config_reg_t *config_reg, config_value_type_t type)
{
        if (type == InitialValue)
        {
       config_reg->CONFIG_OPMODECR = ADC1_OPMODECR_CONFIGVALUE;
       config_reg->CONFIG_CLOCKCR = ADC1_CLOCKCR_CONFIGVALUE;
       config_reg->CONFIG_GxMODECR[0U] = ADC1_G0MODECR_CONFIGVALUE;
       config_reg->CONFIG_GxMODECR[1U] = ADC1_G1MODECR_CONFIGVALUE;
       config_reg->CONFIG_GxMODECR[2U] = ADC1_G2MODECR_CONFIGVALUE;
       config_reg->CONFIG_G0SRC = ADC1_G0SRC_CONFIGVALUE;
       config_reg->CONFIG_G1SRC = ADC1_G1SRC_CONFIGVALUE;
       config_reg->CONFIG_G2SRC = ADC1_G2SRC_CONFIGVALUE;
       config_reg->CONFIG_BNDCR = ADC1_BNDCR_CONFIGVALUE;
       config_reg->CONFIG_BNDEND = ADC1_BNDEND_CONFIGVALUE;
       config_reg->CONFIG_G0SAMP = ADC1_G0SAMP_CONFIGVALUE;
       config_reg->CONFIG_G1SAMP = ADC1_G1SAMP_CONFIGVALUE;
       config_reg->CONFIG_G2SAMP = ADC1_G2SAMP_CONFIGVALUE;
       config_reg->CONFIG_G0SAMPDISEN = ADC1_G0SAMPDISEN_CONFIGVALUE;
       config_reg->CONFIG_G1SAMPDISEN = ADC1_G1SAMPDISEN_CONFIGVALUE;
       config_reg->CONFIG_G2SAMPDISEN = ADC1_G2SAMPDISEN_CONFIGVALUE;
       config_reg->CONFIG_PARCR = ADC1_PARCR_CONFIGVALUE;
        }
        else
        {
       config_reg->CONFIG_OPMODECR = adcREG1->OPMODECR;
       config_reg->CONFIG_CLOCKCR = adcREG1->CLOCKCR;
       config_reg->CONFIG_GxMODECR[0U] = adcREG1->GxMODECR[0U];
       config_reg->CONFIG_GxMODECR[1U] = adcREG1->GxMODECR[1U];
       config_reg->CONFIG_GxMODECR[2U] = adcREG1->GxMODECR[2U];
       config_reg->CONFIG_G0SRC = adcREG1->EVSRC;
       config_reg->CONFIG_G1SRC = adcREG1->G1SRC;
       config_reg->CONFIG_G2SRC = adcREG1->G2SRC;
       config_reg->CONFIG_BNDCR = adcREG1->BNDCR;
       config_reg->CONFIG_BNDEND = adcREG1->BNDEND;
       config_reg->CONFIG_G0SAMP = adcREG1->EVSAMP;
       config_reg->CONFIG_G1SAMP = adcREG1->G1SAMP;
       config_reg->CONFIG_G2SAMP = adcREG1->G2SAMP;
       config_reg->CONFIG_G0SAMPDISEN = adcREG1->EVSAMPDISEN;
       config_reg->CONFIG_G1SAMPDISEN = adcREG1->G1SAMPDISEN;
       config_reg->CONFIG_G2SAMPDISEN = adcREG1->G2SAMPDISEN;
       config_reg->CONFIG_PARCR = adcREG1->PARCR;
        }
}
```