


# Xilinx Zynq FPGA,TI DSP, MCU 기반의 프로그래밍 전문가 과정


강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – 정한별

hanbulkr@gmail.com

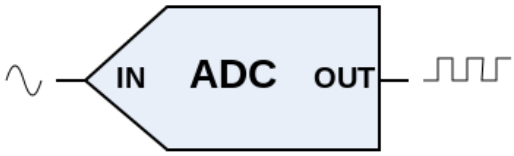
ADC (Analog Digital Converter) 프로그래밍

2018. 7. 16. 18:31
https://blog.naver.com/hanbulkr/221320232262

# 들어가기 전에...

## <기본 설명>

## ADC 는 무엇인가?




**아날로그-디지털 변환회로**는 **A/D 컨버터**(Analog-to-digital converter) 또는 간단하게 **ADC**라 하며, 아날로그 전기 신호를 디지털 전기 신호로 변환하는 전자 회로이다.

아날로그 신호는 저장이나 조작의 편리성이 디지털 신호보다 어렵기 때문에, 초기의 전자공학과는 달리 현재는 디지털화를 많이 한다.

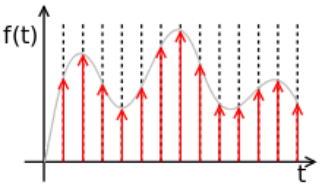
신호전송 시, 일반적으로 아날로그 신호를 디지털 신호를 변환되면 신호의 잡음등에 유리하다. 단지 변환 시 생기는 왜곡은 감수해야 한다. 따라서 아날로그를 디지털화하여 신호를 조작하고 다시 디지털-아날로그 변환회로(DAC)을 통해 아날로그로 변환한다.

예를 들어 초기의 전화기는 아날로그방식으로 신호를 전송하였다. 교환망이 진화하면서 음성신호를 PCM방식으로 디지털화 하고 이것을 전송하고 수신측에서 다시 아날로그로 디코딩 한다.

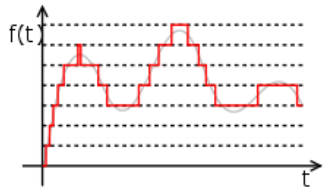
**<보충 설명>**
펄스 부호 변조(Pulse-code modulation, 줄여서 PCM)는 아날로그 신호의 디지털 표현으로, 신호 등급을 균일한 주기로 표본화한 다음 디지털 (이진) 코드로 양자화 처리된다.

샘플링(표본화) : 신호 처리에서 표본화(標本化) 또는 샘플링(sampling)은 연속 신호(유동적인 신호)를 이산 신호(수치화된 신호)로 감소시키는 것을 말한다. 이를테면 파동 (연속 시간 신호)을 일련의 표본(이산 시간 신호)으로 바꾸는 것을 들 수 있다.
표본은 시간 및 공간의 한 점의 값이나 값들의 모임을 가리킨다

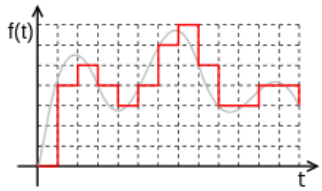
양자화 : 수학과 디지털 신호 처리에서 양자화는 유한 집합에 대량의 입력값을 매핑하는 과정을 말한다.(예를 들어 반올림값을 정밀 단위로) 좀더 구체적으로는 샘플링한 PAM신호(아날로그 데이터)를 이산적인 값(디지털 데이터)으로 바꾸어 표시하는 것을 말한다. -> 밑에 와 같은 양자화 과정을 거친다.



샘플링 된 신호




양자화된 신호




완성된 디지털 신호

**펄스 부호 변조 - 위키백과, 우리 모두의 백과사전**
펄스 부호 변조 위키백과, 우리 모두의 백과사전. ( PCM 에서 넘어옴) …
ko.wikipedia.org

**-> 마지막 요약**

**(아날로그 신호를 MCU가 받으려면 디지털로 받아야 처리가 쉽다.**
**그렇기 때문에 위의 설명들에 따른 과정을 거쳐야 한다.**
**하지만 ADC 라는 MCU 내부에 구현되어 있는 전용 PIN이 있기 때문에 핀에 아날로그 신호를 입력 주어 사용 하면 된다. )**

## <조도 센서 (입력) 회로 선택>

**1. 풀다운 저항 Vs 풀업 저항**

내 식견의 경우로 설명하자면... (틀린생각일 수도 있음)
- 풀업이나 풀다운으로 쓸 저항: 10K

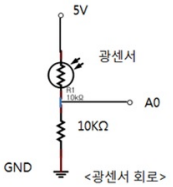
- 조도 센서의 **밝을 때**의 저항 : 1.7K
- 조도 센서의 **어두울 때**의 저항: 10K

풀다운 저항 으로 했을 때 전압 범위 : 4.275 ~ 2.5 V
풀업 저항 으로 했을 때 전압 범위 : 2.5V ~ 0.725V

**위의 상황으로 보아**
**풀 다운 저항으로 써야 더 입력에서 제대로 된 전압 범위에서 인지가 가능하다고 생각이 든다.**

- 광센서 회로 구성은 어떻게 할까요?

  - 아날로그 단자에 연결
  - 10KΩ 저항 사용



<광센서 회로>

※ A0단자에서 측정되는 전압

- 빛의 밝기에 따라 달라진다.
- 식 ) $\frac{10K\Omega}{\text{광센서 내부 저항} + 10K\Omega} \times 5V$
- 광센서의 내부 저항 크기가 외부 빛의 양에 따라 변하면, 위의 식 저항 비율에 따라 A0전압이 달라진다.

※ 10kΩ저항 용도

- 빛이 너무 밝아 광센서의 저항이 0이 되어도 과도한 전류가 흐르지 않도록 하고
- 외부 빛의 양에 따라 A0의 전압을 아날로그 신호로 읽기 위함 (0~5V까지 선형적으로 변화 )

## CORTEX -R5 (TMS570LC43XX)

## Analog To Digital Converter (ADC) Module - PDF

### 이 마이크로 컨트롤러는 최대 2 개의 ADC 모듈 인스턴스를 구현합니다.

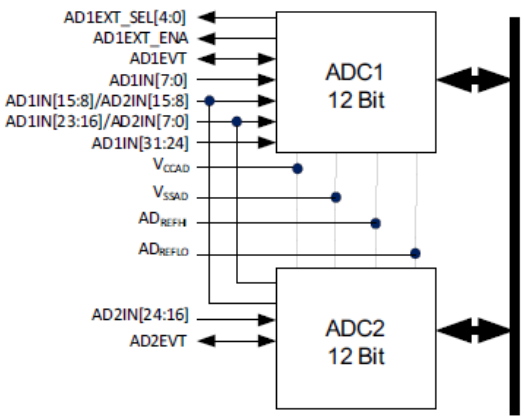
### ADC의 주요 특징 모듈은 다음과 같습니다.

• 선택 가능한 10 비트 또는 12 비트 해상도
• 연속 근사 - 레지스터 아키텍처
• 3 개의 변환 그룹 - 그룹 1, 그룹 2 및 이벤트 그룹
• 세 개의 변환 그룹을 모두 하드웨어 트리거되도록 구성 할 수 있습니다. group1과 group2는 또한 소프트웨어에 의해 촉발된다.
• 변환 결과는 64 워드 메모리 (SRAM) -이 64 단어는 세 개의 변환 그룹으로 나뉘며 소프트웨어별로 구성 할 수 있습니다 - 변환 결과 RAM에 대한 액세스는 패리티로 보호됩니다.
• 변환 결과 전송을위한 DMA 요청 생성을위한 유연한 옵션
• 선택 가능한 채널 변환 순서 - 채널 번호의 오름차순 순차 변환, 또는 - 향상된 채널 선택 모드로 사용자 정의 채널 변환 순서 • 향상된 채널 선택 모드는 ADC1에서만 사용할 수 있습니다.
• 단일 또는 연속 변환 모드
• 입력 채널 오류 감지를위한 내장 된 자체 테스트 로직 (전원에 대한 개방 / 단락 / 접지에 대한 단락)
• 오프셋 오류 보정을위한 내장 보정 로직 • 강화 된 파워 다운 모드
• 변환을 트리거하는 외부 이벤트 핀 (ADEVT) - ADEVT는 범용 I / O로 프로그래밍 가능 • 하드웨어 이벤트 8 개로 변환 실행

### 마이크로 컨트롤러에있는 12 비트 ADC 모듈의 두 인스턴스는 16 개의 아날로그 입력 채널을 공유합니다. 그만큼 연결은 그림 22-1에 나와 있습니다.

• ADC1은 32 채널을 지원합니다.
• ADC2는 25 개 채널을 지원하며 그 중 **16 개 채널이 ADC1과 공유**됩니다.
• 공유 채널에서 ADC1과 ADC2를 모두 사용하는 경우 샘플 윈도우는 동일해야합니다 샘플 윈도우가 서로 완전히 일치하거나 최소 2 개의 ADC와 **겹치지 않음** 하나의 ADC 샘플 윈도우의 끝과 다른 ADC의 샘플 윈도우의 시작 사이의 사이클 버퍼.
• 동작 전압과 동작 전압은 2 개의 ADC 코어간에 공유된다.

**Figure 22-1. Channel Assignments of Two ADC Cores**



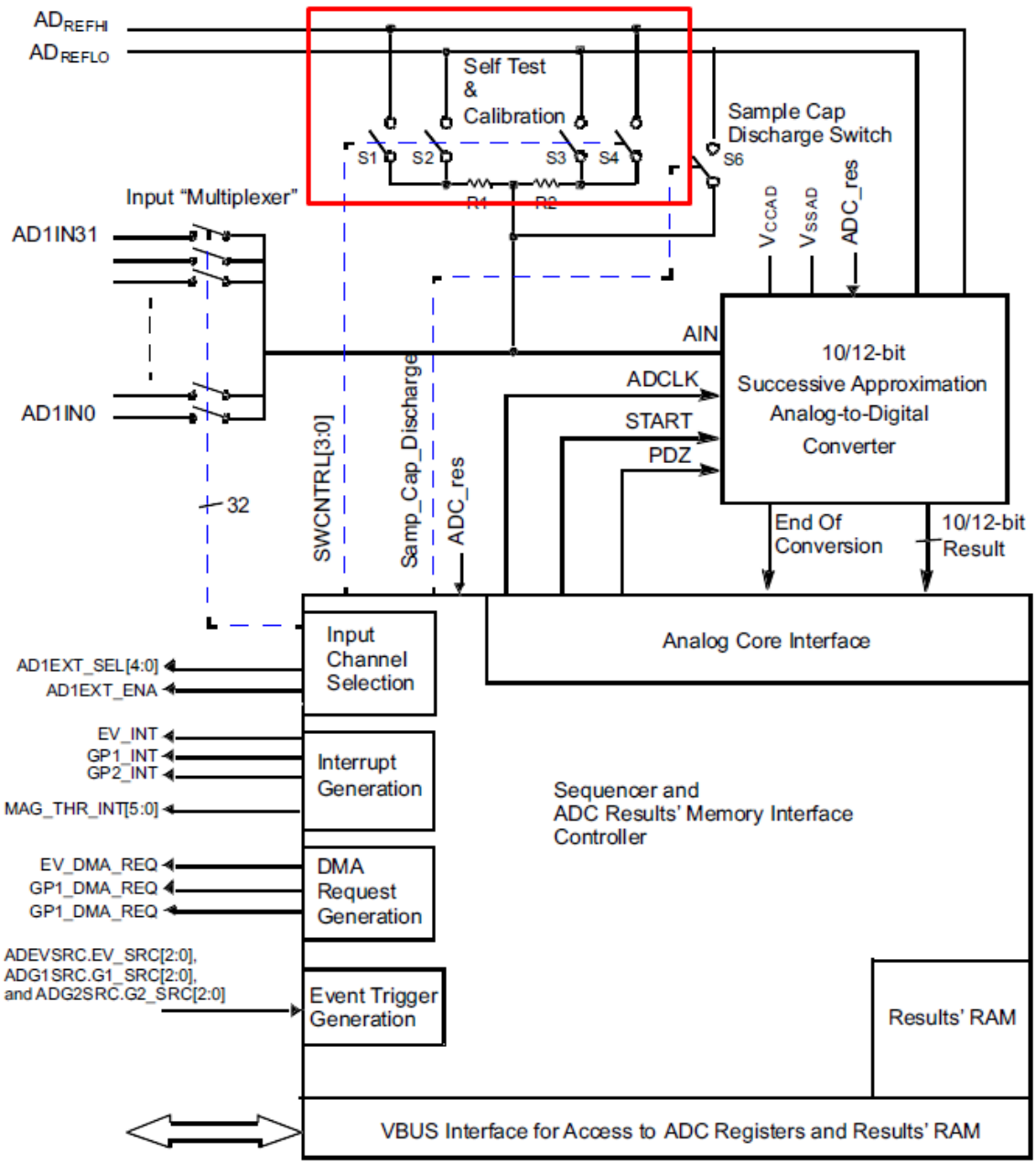

그림을 보면 2개의 핀(16개의 BIT) 가 공유되고 있다)

〈 ADC 내부 회로 Block Diagram 〉

### 22.1.1 Introduction

This section presents a brief functional description of the analog-to-digital converter (ADC) module. Figure 22-2 shows the components of the ADC module.

**Figure 22-2. ADC Block Diagram**




빨간 부분을 보면 어디서 켈리브레이션 하는지 알 수 있다.

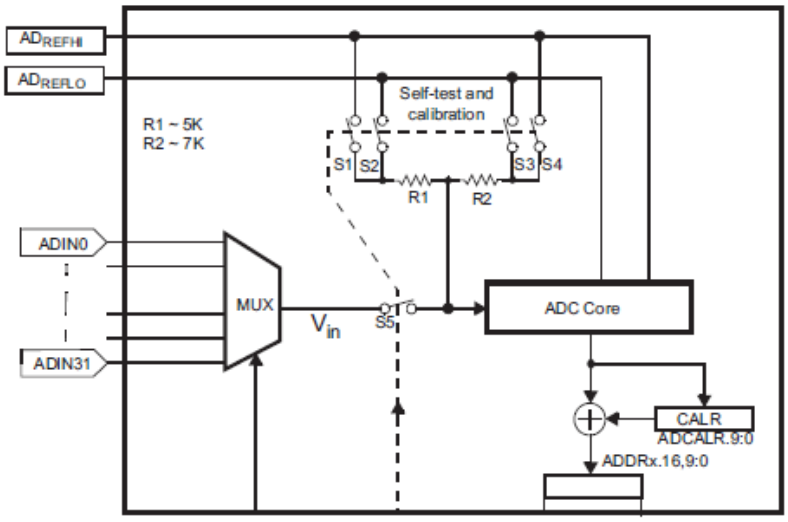
### 〈 Input Multiplexor 〉

- 입력 멀티플렉서 (MUX)는 선택된 입력 채널을 ADC 코어의 AIN 입력에 연결합니다.
- ADC1 모듈은 그림 22-2와 같이 최대 32 개의 입력을 지원합니다. ADC2 모듈은 최대 25 개의 입력을 지원합니다.
- 시퀀서는 변환 할 채널을 선택합니다. **향상된 채널 선택 모드**를 활성화하면 하나 이상의 아날로그 입력 채널을 외부 아날로그 스위치 또는 멀티플렉서의 출력에 연결할 수 있습니다.

### 〈 Self-Test and Calibration Cell 〉

**- ADC는 소프트웨어 알고리즘이 ADC 아날로그 입력에서 개방 / 단락을 감지 할 수있게하는 특정 하드웨어를 포함합니다. 또한 응용 프로그램이 ADC를 조정할 수 있도록합니다.**

**Figure 22-15. Self-Test and Calibration Logic**




In self-test mode, a test voltage defined by the HILO bit (ADCALCR.8) is provided to the ADC core input through a resistor (see Table 22-3). To change the test source, this bit can be toggled before any single conversion mode request. Changing this bit while a conversion is in progress *can* corrupt the results if the source switches during the acquisition period.

Note that the switch S5 shown in Figure 22-15 is only for the purpose of explaining the self-test sequence. There is no physical switch.

**Table 22-3. Self-Test Reference Voltages(1)**

| SELF_TEST | HILO | S1 | S2 | S3 | S4 | S5 | Reference Voltage |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | $AD_{REFLO}$ via R1 \|\| R2 connected to $V_{in}$ |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | $AD_{REFHI}$ via R1 \|\| R2 connected to $V_{in}$ |
| 0 | X | 0 | 0 | 0 | 0 | 1 | $V_{in}$ |

(1) Switches refer to Figure 22-15.

## 〈 Analog-to-Digital Converter Core 〉

ADC 코어는 결합 전압 스케일링, 전하 재분배 연속 근사 레지스터 (SAR) 기반의 아날로그 - 디지털 컨버터이다. 코어는 **10 비트 분해능**으로 작동하도록 구성할 수 있습니다. (기본값) 또는 **12 비트 해상도**. 이것은 시퀀서 로직에 의해 제어됩니다. 이 선택은 모든 변환은 ADC 모듈에 의해 수행됩니다. 일부 채널은 12 비트로 변환 할 수 없습니다. 해상도 및 일부는 10 비트 해상도입니다.

• **샘플링 기간 :**

- 시퀀서는 샘플링 기간의 시작을 알리기 위해 ADC 코어에 START 신호를 생성합니다.
- 아날로그 입력 신호는이 기간 동안 스위치드 커패시터 어레이로 직접 샘플링되며, 고유 한 **샘플 앤 홀드** (sample-and-hold) 기능을 제공합니다.
- **샘플링 기간**은 START 신호의 하강 에지 이후에 전체 ADCLK로 끝납니다.
- 시퀀서는 변환 **그룹의 구성**을 통해 **샘플링 기간을 제어** 할 수 있습니다. 샘플 시간 제어 레지스터 (ADEVSAMP, ADG1SAMP, ADG2SAMP). 이 레지스터는 START 신호가 하이로 유지되는 시간을 제어합니다.

• **전환 기간 :**

- **변환주기**는 START의 하강 에지 이후에 전체 ADCLK를 시작합니다.
- 변환 결과의 1 비트는 변환주기의 ADCLK의 각 상승 에지에서 출력되며, 최상위 비트부터 시작합니다.
- **변환주기**는 **12 비트 ADC의 경우 12 ADCLK 사이클**이며, **10 비트 ADC의 경우 10 ADCLK 사이클**입니다.
- ADC 코어는 **변환 기간이 끝날 때** 시퀀서에 EOC (End-Of-Conversion) 신호를 생성합니다. **이 시점에서 완전한 12 비트 또는 10 비트 변환 결과를 사용할 수 있습니다.**
- 시퀀서는 EOC가 하이로 구동되는 즉시 ADC 코어 변환 결과 출력을 캡처합니다.

**** 아날로그 변환 범위는 기준 전압 ADREFHI 및 ADREFLO에 의해 결정됩니다.**

**ADREFHI는 최고 기준 전압이며 변환 할 수있는 최대 아날로그 전압입니다**

$$DigitalResult = \frac{1024 \times (InputVoltage - AD_{REFLO})}{AD_{REFHI} - AD_{REFLO}} - 0.5$$

$$DigitalResult = \frac{4096 \times (InputVoltage - AD_{REFLO})}{(AD_{REFHI} - AD_{REFLO})} - 0.5$$

〈 Sequencer 〉

시퀀서는 입력 멀티플렉서, ADC 코어 및 ADC 코어를 포함하여 ADC의 동작을 조정합니다. 결과 메모리. 또한 시퀀서의 논리는 변환이 진행 중이거나 중지되거나 완료 될 때 상태 레지스터 플래그를 설정합니다. 시퀀서의 모든 기능은이 문서의 다음 섹션에서 자세히 설명합니다.

〈 Conversion Groups 〉

ADC 모듈은이 목적을 위해 3 개의 변환 그룹,
즉 Group1, Group2 및 이벤트 그룹. 으로 나뉘어 있다.

● Basic Operation( 기본 옵션 )

1. How to Select Between 12-bit and 10-bit Resolutions

ADC 동작 모드 제어 레지스터 (ADOPMODECR)의 10_12_BIT 필드는 ADC를 구성한다
10 비트 또는 12 비트 해상도 모드 일 때 :

- 10_12_BIT = 0이면 모듈은 10 비트 해상도 모드입니다. 이것은 기본 작동 모드입니다.
- 10_12_BIT = 1이면 모듈은 12 비트 해상도 모드입니다.

2. How to Set Up the ADCLK Speed

ADC 시퀀서는 ADC 코어에 대한 클럭 인 ADCLK를 생성한다. ADC 코어는 타이밍을 위해 ADCLK 신호를 사용한다. ADCLK는 입력 클록을 ADC 모듈 (VBUS P 인터페이스 클럭, VCLK)으로 분할하여 생성된다. ADC 클록 제어 레지스터 (ADCLOCKCR)의 5 비트 필드 (PS)는 VCLK를 1에서 32로 나누기 위해 사용된다. ADCLK 유효 주파수 범위는 디바이스 데이터 시트에 명시되어있다.

fADCLK = fVCLK / (PS + 1)
ADCLK의 최대 주파수는 장치 데이터 시트에 지정됩니다.

3. How to Set Up the Input Channel Acquisition Time

각 그룹의 신호 획득 시간은 ADG1SAMP [11 : 0], ADG2SAMP [11 : 0] 및 ADEVSAMP [11 : 0] 레지스터를 사용하여 개별적으로 구성 할 수 있습니다.
획득 시간은 ADCLK 사이클을 기준으로 지정되며 최소 2 ADCLK 사이클에서 최대 4098 ADCLK 사이클까지의 범위입니다.
예를 들어 Group1 수집 시간 인 tACQG1 = G1SAMP [11 : 0] + 2가 ADCLK 사이클에서 발생합니다.
최소 수집 시간은 장치 데이터 시트에 명시되어 있습니다. 이 시간은 변환되는 아날로그 입력 채널에 연결된 회로의 임피던스에 따라 달라집니다. Hercules ™ ARM® Safety MCU 용 ADC 소스 임피던스 Application Report (SPNA118)를 참조하십시오.

4. How to Select an Input Channel for Conversion

- 변환을위한 입력 채널을 선택하기 전에 먼저 ADC 모듈을 활성화해야합니다.
- ADC 작동 모드 제어 레지스터 (ADOPMODECR)에서 ADC_EN 비트를 설정하여 ADC 모듈을 활성화 할 수있다.
- 다중 입력 채널은 각 그룹에서 변환을 위해 선택 될 수 있습니다. 한 번에 하나의 입력 채널 만 변환됩니다.
- 변환 될 채널은 3 개 변환 그룹의 채널 선택 레지스터 중 하나 이상에서 구성됩니다.
- Group1에서 변환 될 채널은 Group1 채널 선택 레지스터 (ADG1SEL)에 구성되고 Group2에서 변환 될 채널은 Group2 채널 선택 레지스터 (ADG2SEL)에 구성되며 이벤트 그룹에서 변환 될 채널은 이벤트 그룹 채널 선택 레지스터 (ADEVSEL).
이 섹션의 설명은 고급 채널 선택 모드가 활성화되지 않은 경우를 나타냅니다.

향상된 채널 선택 모드의 입력 채널 선택은 22.2.2 절에 정의되어 있습니다.

5. How to Select Between Single Conversion Sequence or Continuous Conversions

각 그룹에는 자체 모드 제어 레지스터가 있습니다.
이 제어 레지스터의 MODE 필드는 애플리케이션이 **단일 변환 시퀀스** 또는 **연속 변환 모드** 중에서 선택할 수 있도록합니다.

**세 그룹 모두에 대해 연속 변환 모드 선택**
세 개의 변환 그룹을 모두 연속 변환 모드로 구성 할 수 없습니다. 응용 프로그램이 그룹 모드 제어 레지스터를 구성하여 세 그룹 모두에 대해 연속 변환 모드를 활성화하면 Group2가 자동으로 단일 변환 시퀀스 모드로 구성됩니다.

연속 변환 모드에서 변환이 진행중인 경우 그룹의 MODE 필드가 지워지면 해당 그룹은 단일 변환 순서 모드로 전환됩니다.

### 6. How to Start a Conversion

변환 그룹 Group1 및 Group2는 기본적으로 소프트웨어 트리거됩니다. 이 그룹의 변환은 각 채널 선택 레지스터에 원하는 채널을 쓰는 것만으로 시작할 수 있습니다.
예를 들어 Group1의 채널 0, 1, 2 및 3과 Group2의 채널 8, 9, 10 및 11을 변환하려면 응용 프로그램이 ADG1SEL에 0x0000000F를 입력하고 ADG2SEL에 0x00000F00을 씁니다. ADC 모듈은 먼저 트리거 된 그룹 인 Group1에 서비스를 제공함으로써 시작됩니다.
모든 그룹에 대한 전환은 채널 번호의 오름차순으로 수행됩니다. Group1의 경우 전환은 채널 0, 채널 1, 채널 2 순으로 순서대로 수행됩니다.
Group2 변환은 채널 8, 9, 10 및 11의 순서로 수행됩니다.

이벤트 그룹은 하드웨어로 트리거됩니다. ADC 모듈에 대해 정의 된 최대 8 개의 하드웨어 이벤트 트리거 소스가 있습니다. 이 8 가지 하드웨어 트리거 옵션의 전체 목록은 장치 데이터 시트를 확인하십시오.

사용되는 트리거 소스는 ADEVSRC 레지스터에서 구성해야합니다. Group1 및 Group2에 대해서도 유사한 레지스터가 존재하므로 이벤트 트리거되도록 구성 할 수 있습니다.
이벤트 트리거의 극성도 구성 가능하며, 하강 에지가 기본값입니다.

이 그룹에서 변환을 위해 하나 이상의 채널이 선택되고 정의 된 이벤트 트리거가 발생할 때 이벤트 그룹 변환이 시작됩니다.
*변환 그룹이 연속 변환 모드로 구성된 경우에는 한 번만 트리거*해야합니다. 해당 그룹에서 변환을 위해 선택된 모든 채널이 반복적으로 변환됩니다.

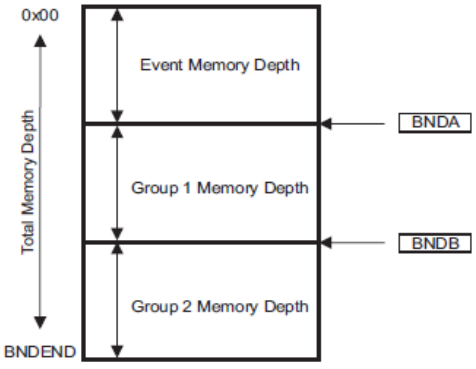
### 7. How to Know When the Group Conversion is Completed

각 전환 그룹에는 전환이 종료 된 시점을 나타내는 상태 플래그가 있습니다. ADEVSR, ADG1SR 및 ADG2SR을 참조하십시오. 이 비트는 그룹에 대한 변환 순서가 끝나면 설정됩니다. 그룹이 연속 변환으로 구성된 경우 이 비트는 항상 설정됩니다.

### 8. How Results are Stored in the Results' Memory

ADC는 각 그룹에 대해 하나의 영역 인 ADC 결과의 RAM에 3 개의 개별 메모리 영역에 변환 결과를 저장합니다. 각 메모리 영역은 하나의 변환 결과를 보유 할 수있는 버퍼의 스택입니다. 각 그룹에 할당 된 버퍼 수는 ADC 모듈 레지스터 ADBNDCR 및 ADBNDEND를 구성하여 프로그래밍합니다.
ADBNDCR에는 두 개의 9 비트 포인터 BNDA와 BNDB가 있습니다. BNDA, BNDB 및 BNDEND는 그림 22-3과 같이 사용 가능한 총 메모리를 세 개의 메모리 영역으로 분할하는 데 사용됩니다. BNDA와 BNDB는 모두 결과의 메모리 시작에서 참조되는 포인터입니다. BNDA는 이벤트 그룹 변환 결과에 할당 된 버퍼 수를 두 개의 버퍼 단위로 지정합니다. BNDB는 이벤트 그룹에 할당 된 버퍼 수와 Group1을 두 개의 버퍼 단위로 지정합니다. ADC 결과 메모리 구성에 대한 자세한 내용은 22.3.23 절을 참조하십시오.
ADBNDEND에는 사용 가능한 총 메모리를 구성하는 BNDEND라는 3 비트 필드가 들어 있습니다. *ADC 모듈은 최대 1024 개의 버퍼를 지원*할 수 있습니다. 이 소자는 두 ADC 모듈 모두를 위해 최대 64 개의 버퍼를 지원한다.

**Figure 22-3. FIFO Implementation**




- Number of buffers for Event Group = 2 × BNDA
- Number of buffers for Group1 = 2 × (BNDB – BNDA)
- Number of buffers for Group2 = Total number of buffers – 2 × BNDB

〈How to Read the Results from the Results' Memory〉
CPU는 다음 두 가지 방법 중 하나로 변환 결과를 읽을 수 있습니다.
1. 변환 결과 메모리를 FIFO 대기열로 사용합니다.
2. 변환 결과 메모리에 직접 액세스 합니다.

### 9. How to Stop a Conversion

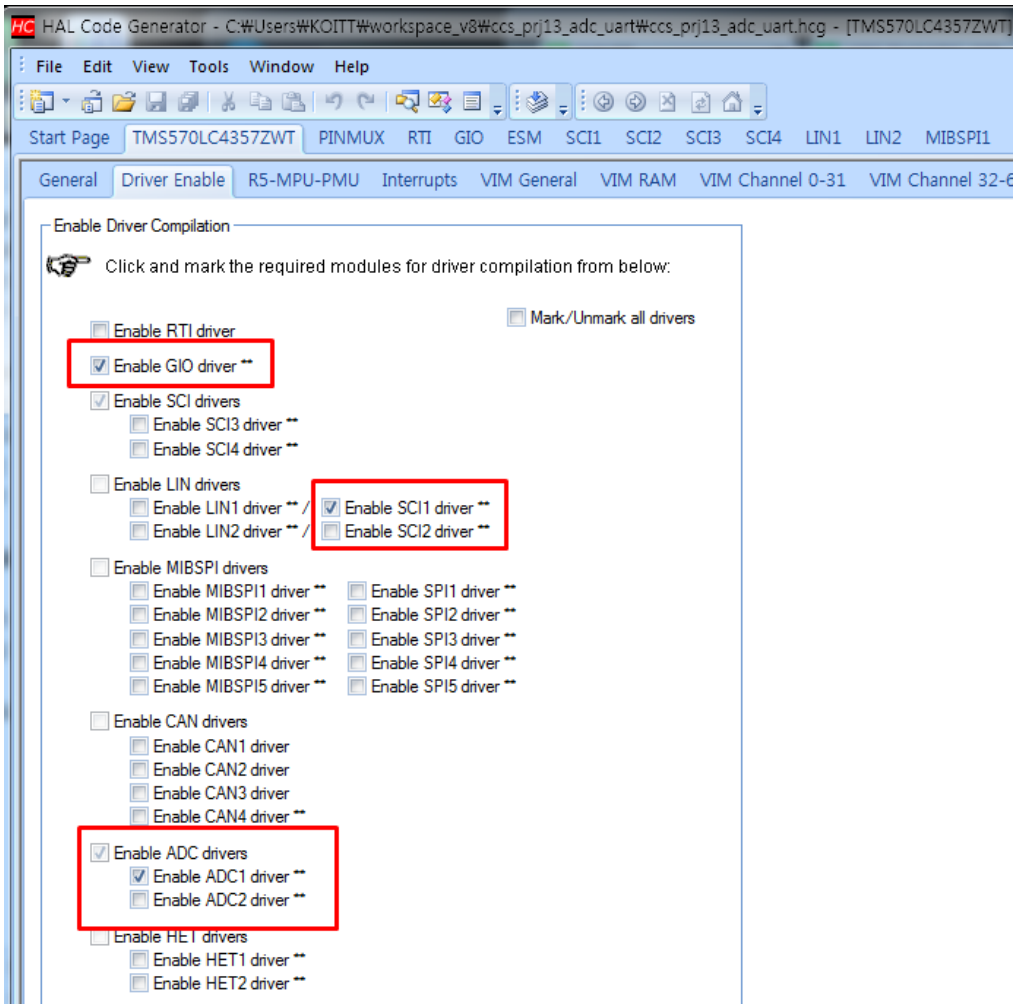
그룹의 *채널 선택 레지스터를 지우면* 그룹의 *변환을 중지* 할 수 있습니다.

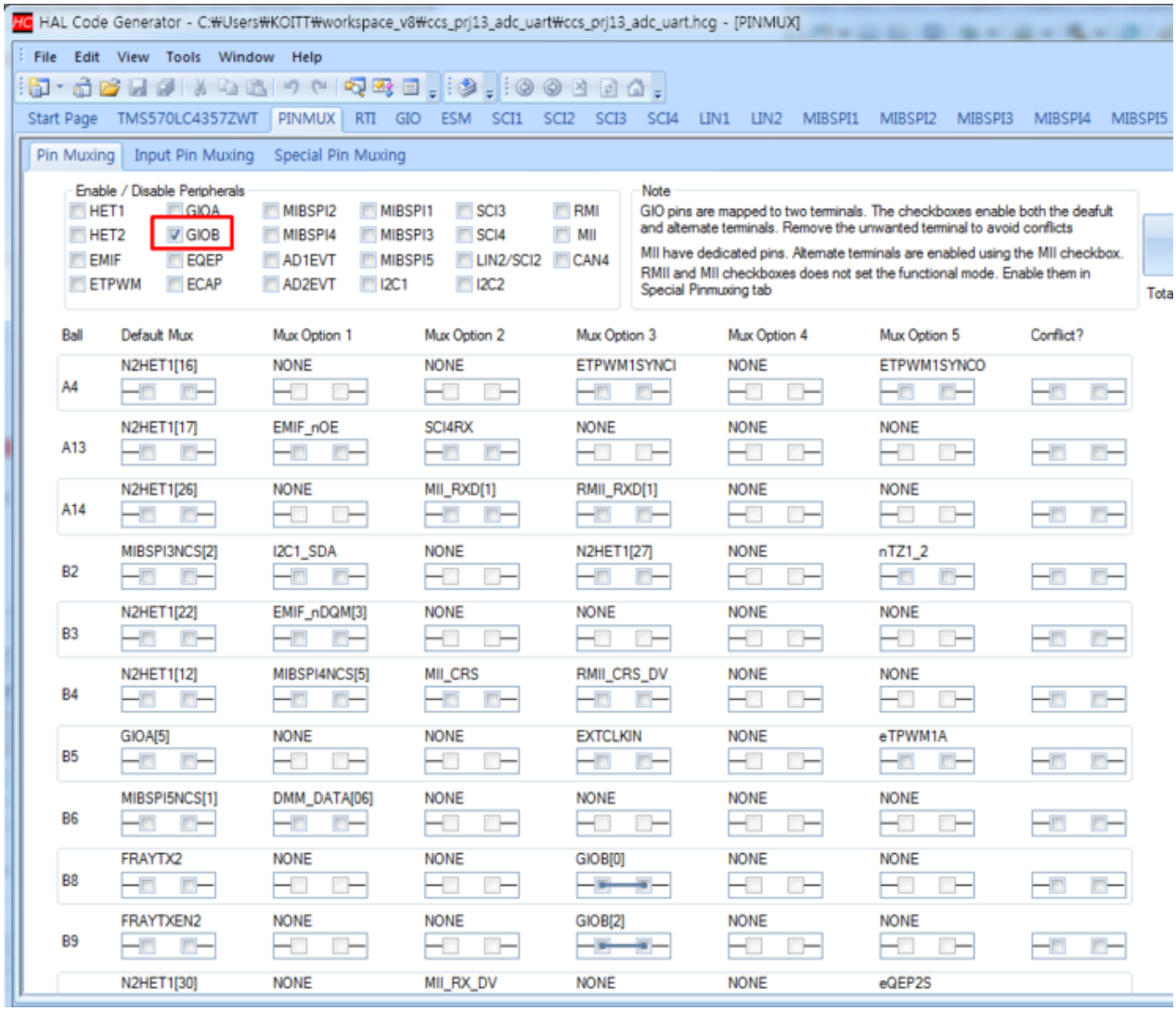
## 〈 조도센서, ADC _ UART 통신하기 〉

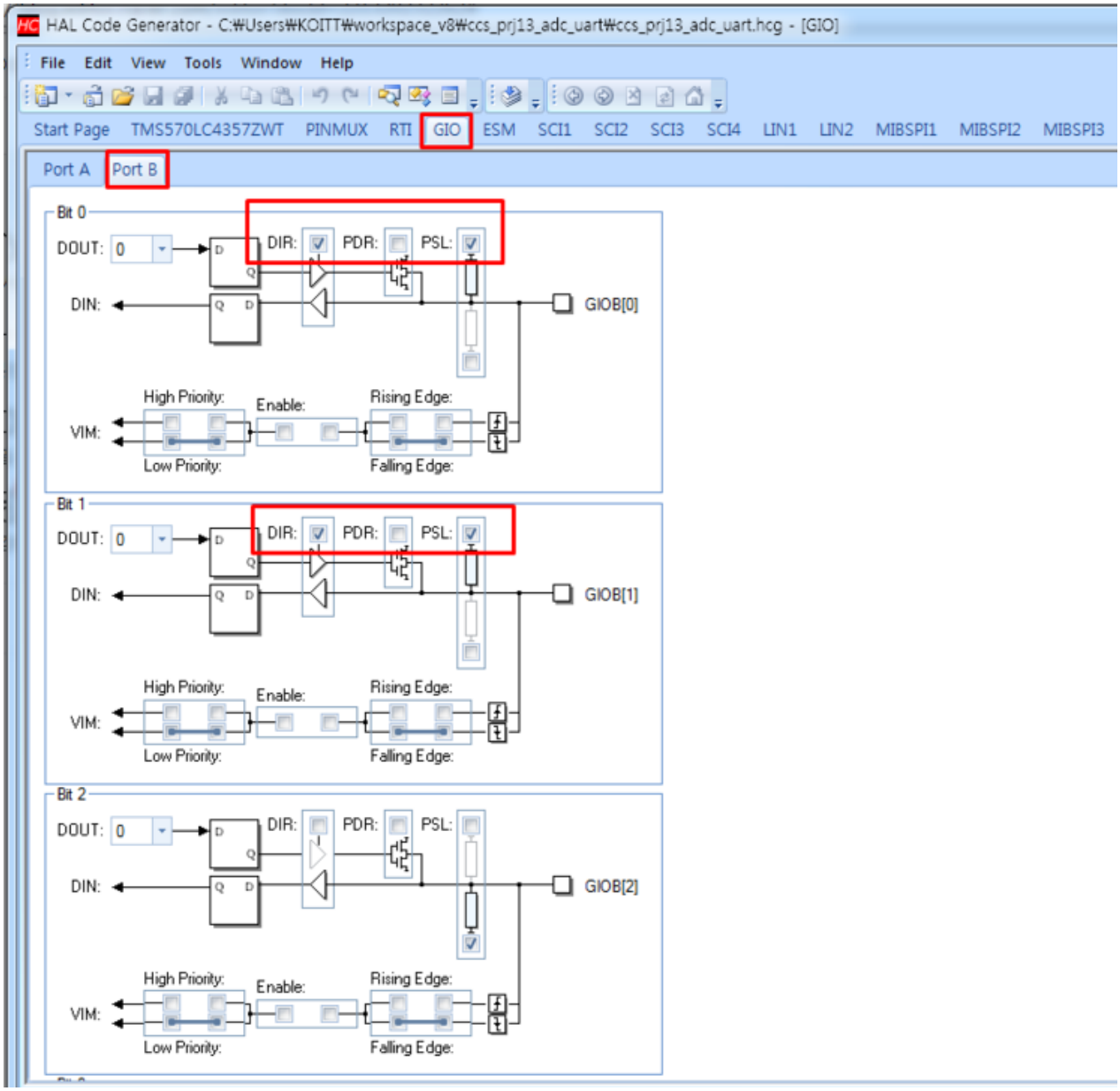
## 1) HALCoGen 설정.

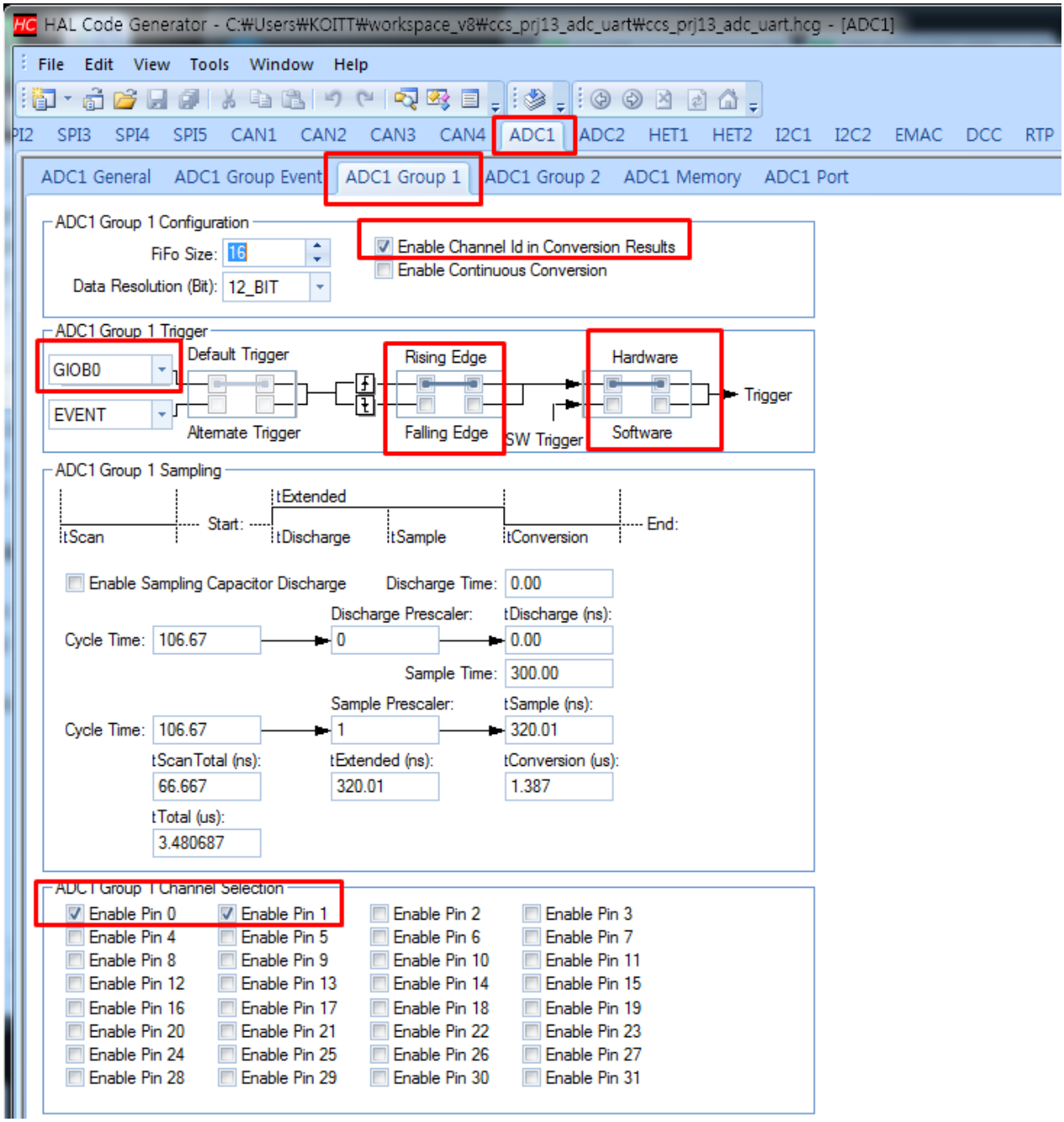
## - 일단 GIO, SCI , ADC 를 켜준다.

- GIO로 LED TRRIGER.
- ADC 값을 SCI로 출력.




HAL Code Generator - C:₩Users₩KOITT₩workspace_v8₩ccs_prj13_adc_uart₩ccs_prj13_adc_uart.hcg - [PINMUX]
File Edit View Tools Window Help
Start Page TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5
Pin Muxing Input Pin Muxing Special Pin Muxing
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
B6 MIBSPI5NCS[1] DMM_DATA[06] NONE NONE NONE NONE
B8 FRAYTX2 NONE NONE GIOB[0] NONE NONE
B9 FRAYTXEN2 NONE NONE GIOB[2] NONE NONE
N2HET1[30] NONE MII_RX_DV NONE NONE eQEP2S


HAL Code Generator - C:₩Users₩KOITT₩workspace_v8₩ccs_prj13_adc_uart₩ccs_prj13_adc_uart.hcg - [GIO]
File Edit View Tools Window Help
Start Page TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3
Port A Port B
Bit 0
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[0]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:
Bit 1
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[1]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:
Bit 2
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[2]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:




## 2) CCS 프로그래밍 하기

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include <include/HL_adc.h>
#include <include/HL_gio.h>
#include <include/HL_hal_stdtypes.h>
#include <include/HL_reg_sci.h>
#include <include/HL_sci.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


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

```
int idx = 0;
uint8 data_buffer[]= " ";

adcData_t adc_data[2];


/* USER CODE BEGIN (2) */
void rtiNo()
{

}


void delay(uint32 time)
{
    while (time--)
        ;
}

/* USER CODE END */

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

    uint8 msg[] = {0};

    gioInit();
    sciInit();

    // 할코젠에서 설정한 setting들을 설정해준다.
    adcInit();
    // adc 입력으로 들어오는 값을의 변화능 ㄹ시작해 준다. (ADC1의 adcGROUP1의 녀석이 가지고 있는 BIT의 컨버젼을 시작한다.)
    adcStartConversion(adcREG1, adcGROUP1);

    // 트리거를 해줄 포트 B를 0으로 기본설정 해준다.
    gioSetBit(gioPORTB, 0, 0);

    while(1)
    {
        // 할코젠에서 하이 엣지 트리거로 설정을 했기 때문에 직접 1이라는 값을 준다.
        // 포트 B의 0번 핀이 SET이 되면 트리거에 의해 입력 값이 들어와 컴버젼을 시작한다.
        gioSetBit(gioPORTB, 0, 1);

        // ADC의 컨버젼이 완료가 되면 트리거에서 8이라는 값을 리턴한다. 준비되어 있지 않으면 0 값을 리턴한다.
        while((adcIsConversionComplete(adcREG1, adcGROUP1)) == 0)
            ;
        // 준비가 완료 되면 ADC 값을 내가 만든 ADC 구조체인 adc_data 라는 박스로 가져온다.
        // -> 여기서 주의 할 점은 배열로 32개 까지 받아 쓸 수 있고
        //    배열의  index 값에 따라서 pin의 값이 들어간다.
        adcGetData(adcREG1, adcGROUP1, adc_data);

        // ADC 구조체인 adcData_t 속에는 ( value , id ) 가 들어 있다. value 값을 출력하면 변환된 값을 볼 수 있다.
        sprintf((char *)msg, "value0 = %d\r\n", adc_data[0].value);
```

```
            sciSend(sciREG1, strlen((const char *)msg), msg);
            sprintf((char *)msg, "value1 = %d\r\n\n", adc_data[1].value);
            sciSend(sciREG1, strlen((const char *)msg), msg);

            delay(5000000);
            // 마지막으로 중요한 트리거를 0으로 다운 시키는 부분이다.
            // 이 녀석을 해주면서 pin의 현재 값을 0으로 해주어야  다시 rising edge 값을 받을 준비를 할 수 있다.
            gioSetBit(gioPORTB, 0, 0);

        }
    /* USER CODE END */

        return 0;
    }


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

### 3) 코드 분석

### <ADC에서 쓰는 기본 함수들 모음>

```
/* ADC Interface Functions */

void adcInit(void);
void adcStartConversion(adcBASE_t *adc, uint32 group);
void adcStopConversion(adcBASE_t *adc, uint32 group);
void adcResetFiFo(adcBASE_t *adc, uint32 group);
uint32  adcGetData(adcBASE_t *adc, uint32 group, adcData_t *data);
uint32  adcIsFifoFull(adcBASE_t *adc, uint32 group);
uint32  adcIsConversionComplete(adcBASE_t *adc, uint32 group);
void adcEnableNotification(adcBASE_t *adc, uint32 group);
void adcDisableNotification(adcBASE_t *adc, uint32 group);
void adcCalibration(adcBASE_t *adc);
uint32 adcMidPointCalibration(adcBASE_t *adc);
void adcSetEVTPin(adcBASE_t *adc, uint32 value);
uint32 adcGetEVTPin(adcBASE_t *adc);


/** @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);
```

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

: 하드웨어 그룹의 변환을 시작하는 기능이다. ( 하드웨어적으로 들어오는 adc 값을 변환해 준다. )

```
/** @fn void adcStartConversion(adcBASE_t *adc, uint32 group)
*   @brief Starts an ADC conversion
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*
*   This function starts a conversion of the ADC hardware group.
*
*/
/* SourceId : ADC_SourceId_002 */
/* DesignId : ADC_DesignId_002 */
/* Requirements : HL_CONQ_ADC_SR3 */
void adcStartConversion(adcBASE_t *adc, uint32 group)
{
    uint32 index = (adc == adcREG1) ? 0U : 1U;

/* USER CODE BEGIN (7) */
/* USER CODE END */

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

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

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

/* USER CODE BEGIN (8) */
/* USER CODE END */
}
```

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

: 하드웨어 그룹의 변환을 중지하는기능이다.

```
/** @fn void adcStopConversion(adcBASE_t *adc, uint32 group)
*   @brief Stops an ADC conversion
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*
*   This function stops a conversion of the ADC hardware group.
*
*/
/* SourceId : ADC_SourceId_003 */
/* DesignId : ADC_DesignId_003 */
/* Requirements : HL_CONQ_ADC_SR4 */
void adcStopConversion(adcBASE_t *adc, uint32 group)
{
/* USER CODE BEGIN (10) */
/* USER CODE END */

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

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

/* USER CODE BEGIN (11) */
/* USER CODE END */
}
```

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

: 이 함수는 adc값이 변환 되어 온 값을 adc message 를 adc box에 저장 한다.

```
/** @fn uint32 adcGetData(adcBASE_t *adc, uint32 group, adcData_t * data)
*   @brief Gets converted a ADC values
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*   @param[out] data Pointer to store ADC converted data
*   @return The function will return the number of converted values copied into data buffer:
*
*   This function writes a ADC message into a ADC message box.
*
*/
/* SourceId : ADC_SourceId_005 */
/* DesignId : ADC_DesignId_005 */
/* Requirements : HL_CONQ_ADC_SR6 */
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;

/* USER CODE BEGIN (16) */
/* USER CODE END */

    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;

    /**   @note The function adcInit has to be called before this function can be used.\n
    *           The user is responsible to initialize the message box.
    */

/* USER CODE BEGIN (17) */
/* USER CODE END */

    return count;
}
```

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

: adc FIFO의 read 와 right 포인터를 리셋을 해준다.

```
/** @fn void adcResetFiFo(adcBASE_t *adc, uint32 group)
*   @brief Resets FiFo read and write pointer.
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*
*   This function resets the FiFo read and write pointers.
*
*/
/* SourceId : ADC_SourceId_004 */
/* DesignId : ADC_DesignId_004 */
/* Requirements : HL_CONQ_ADC_SR5 */
void adcResetFiFo(adcBASE_t *adc, uint32 group)
{
/* USER CODE BEGIN (13) */
/* USER CODE END */

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

    /**   @note The function adcInit has to be called before this function can be used.\n
    *           the conversion should be stopped before calling this function.
    */

/* USER CODE BEGIN (14) */
/* USER CODE END */
}
```

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

: FIFO 의 buffer 상태를 리턴해 준다.

3가지의 상태를 볼수 있다.

```
*   @return The function will return:
*       - 0: When FiFo buffer is not full
*       - 1: When FiFo buffer is full
*       - 3: When FiFo buffer overflow occurred
```

```
/** @fn uint32 adcIsFifoFull(adcBASE_t *adc, uint32 group)
*   @brief Checks if FiFo buffer is full
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*   @return The function will return:
*           - 0: When FiFo buffer is not full
*           - 1: When FiFo buffer is full
*           - 3: When FiFo buffer overflow occurred
*
*   This function checks FiFo buffer status.
*
*/
/* SourceId : ADC_SourceId_006 */
/* DesignId : ADC_DesignId_006 */
/* Requirements : HL_CONQ_ADC_SR7 */
uint32 adcIsFifoFull(adcBASE_t *adc, uint32 group)
{
    uint32 flags;

/* USER CODE BEGIN (19) */
/* USER CODE END */

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

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

/* USER CODE BEGIN (20) */
/* USER CODE END */

    return flags;
}
```

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

**: adc 의 변환이 완료되면 , 완료된 flag 값을 리턴해 준다.**

```
*  @return The function will return:
*        - 0: When is not finished
*        - 8: When conversion is complete
```

```
/** @fn uint32 adcIsConversionComplete(adcBASE_t *adc, uint32 group)
*   @brief Checks if Conversion is complete
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*   @return The function will return:
*           - 0: When is not finished
*           - 8: When conversion is complete
*
*   This function checks if conversion is complete.
*
*/
/* SourceId : ADC_SourceId_007 */
/* DesignId : ADC_DesignId_007 */
/* Requirements : HL_CONQ_ADC_SR8 */
uint32 adcIsConversionComplete(adcBASE_t *adc, uint32 group)
{
    uint32 flags;

/* USER CODE BEGIN (22) */
/* USER CODE END */

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

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

/* USER CODE BEGIN (23) */
/* USER CODE END */

    return flags;
}
```

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

: 이 함수는 보정 모드를 사용하여 오프셋 오류를 계산합니다.

```
/* USER CODE BEGIN (24) */
/* USER CODE END */

/** @fn void adcCalibration(adcBASE_t *adc)
*   @brief Computes offset error using Calibration mode
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   This function computes offset error using Calibration mode
*
*/
/* SourceId : ADC_SourceId_008 */
/* DesignId : ADC_DesignId_010 */
/* Requirements : HL_CONQ_ADC_SR11 */
void adcCalibration(adcBASE_t *adc)
{
/* USER CODE BEGIN (25) */
/* USER CODE END */

    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 */
        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->CALR = offset_error;

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

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

/* USER CODE BEGIN (26) */
/* USER CODE END */
}
```

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

: 이 함수는 중간 pointer 교정 모드를 사용하여 오프셋 오류를 계산합니다.

```
/** @fn void adcMidPointCalibration(adcBASE_t *adc)
*   @brief Computes offset error using Mid Point Calibration mode
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @return This function will return offset error using Mid Point Calibration mode
*
*   This function computes offset error using Mid Point Calibration mode
*
*/
/* SourceId : ADC_SourceId_009 */
/* DesignId : ADC_DesignId_011 */
/* Requirements : HL_CONQ_ADC_SR12 */
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);

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

/* USER CODE BEGIN (28) */
/* USER CODE END */
}
```

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

: 이 기능은 conversion의 notification 함수를 사용 가능하게합니다.

변환 완료를위한 단일 변환 모드와 FiFo 버퍼가 가득 차면 연속 변환 모드.

```
/** @fn void adcEnableNotification(adcBASE_t *adc, uint32 group)
*   @brief Enable notification
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*
*   This function will enable the notification of a conversion.
*   In single conversion mode for conversion complete and
*   in continuous conversion mode when the FiFo buffer is full.
*
*/
/* SourceId : ADC_SourceId_010 */
/* DesignId : ADC_DesignId_008 */
/* Requirements : HL_CONQ_ADC_SR9 */
void adcEnableNotification(adcBASE_t *adc, uint32 group)
{
    uint32 notif = (((uint32)(adc->GxMODECR[group]) & 2U) == 2U) ? 1U : 8U;

/* USER CODE BEGIN (30) */
/* USER CODE END */

    adc->GxINTENA[group] = notif;

    /**   @note The function adcInit has to be called before this function can be used.\n
    *           This function should be called before the conversion is started
    */

/* USER CODE BEGIN (31) */
/* USER CODE END */
}
```

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

: 이 기능은 conversion의 notification 함수를 사용 불가 하게 합니다.

```
/** @fn void adcDisableNotification(adcBASE_t *adc, uint32 group)
*   @brief Disable notification
*   @param[in] adc Pointer to ADC module:
*              - adcREG1: ADC1 module pointer
*              - adcREG2: ADC2 module pointer
*   @param[in] group Hardware group of ADC module:
*              - adcGROUP0: ADC event group
*              - adcGROUP1: ADC group 1
*              - adcGROUP2: ADC group 2
*
*   This function will disable the notification of a conversion.
*/
/* SourceId : ADC_SourceId_011 */
/* DesignId : ADC_DesignId_008 */
/* Requirements : HL_CONQ_ADC_SR9 */
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 */
}
```

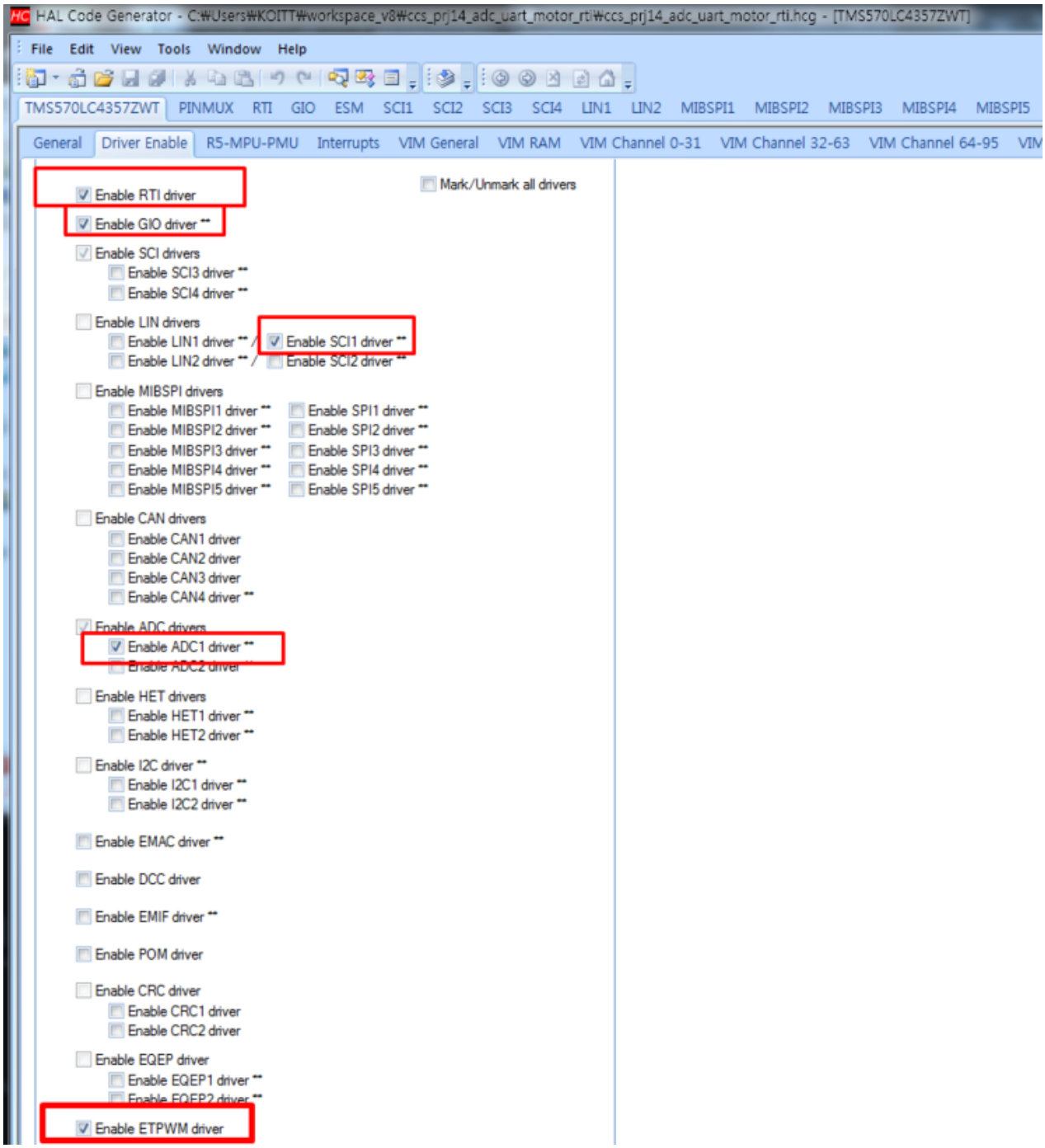
## 〈 조도센서, ADC _ UART 통신하며, ADC_MOTOR, 트리거는 RTI로 돌리기 〉

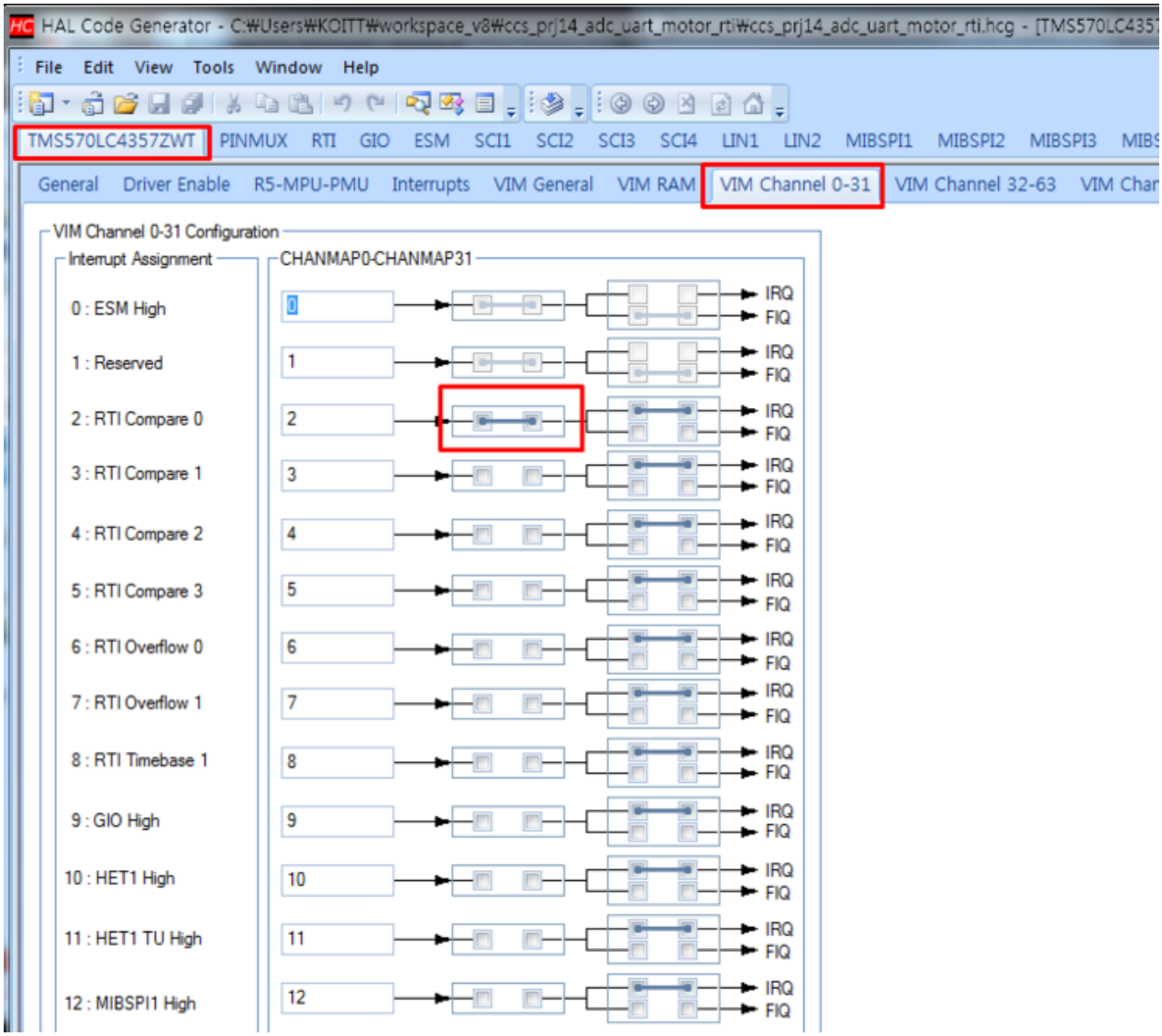
## 1) HALCoGen 설정.

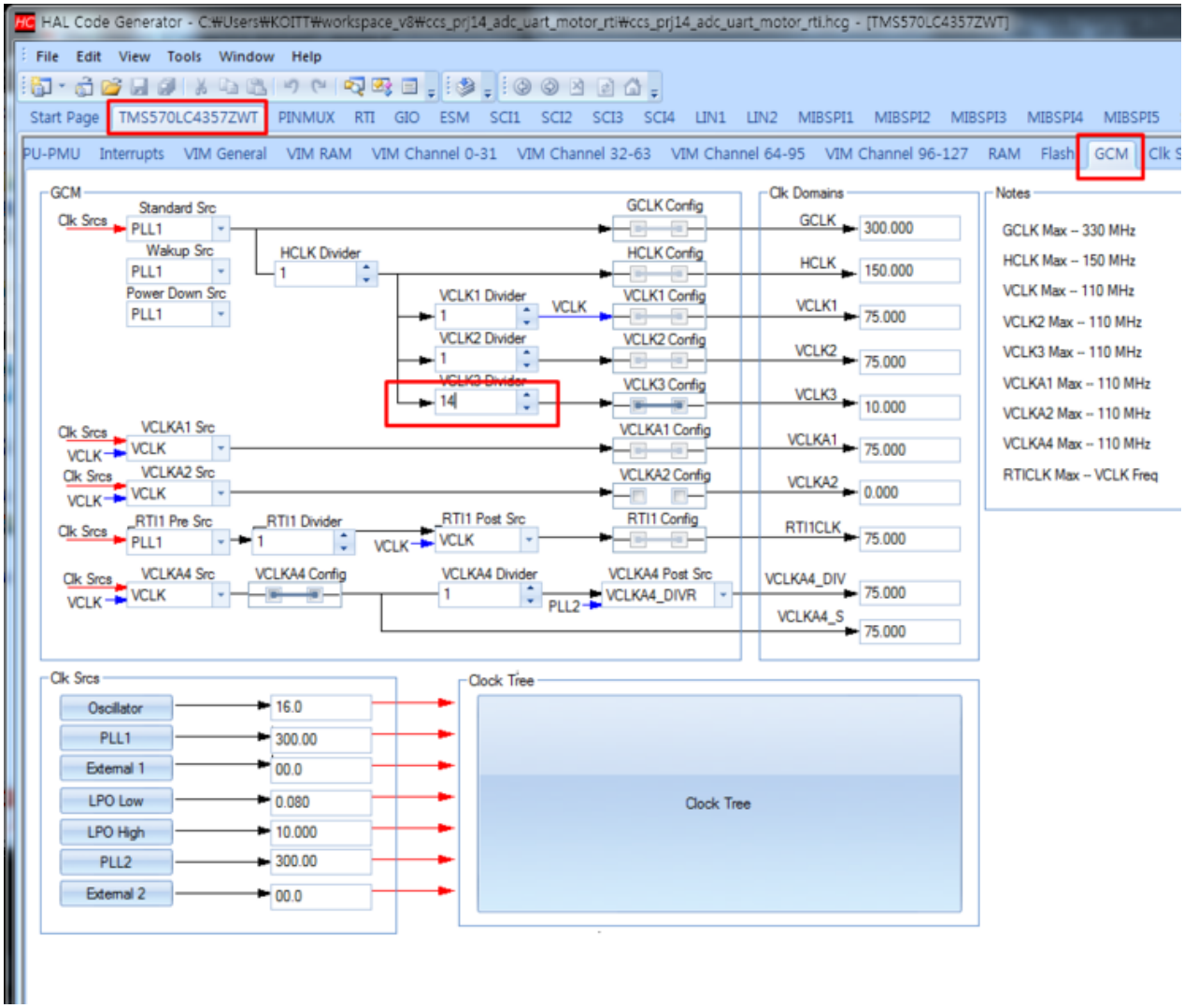
## - 일단 GIO, SCI, ADC, RTI 를 켜준다.

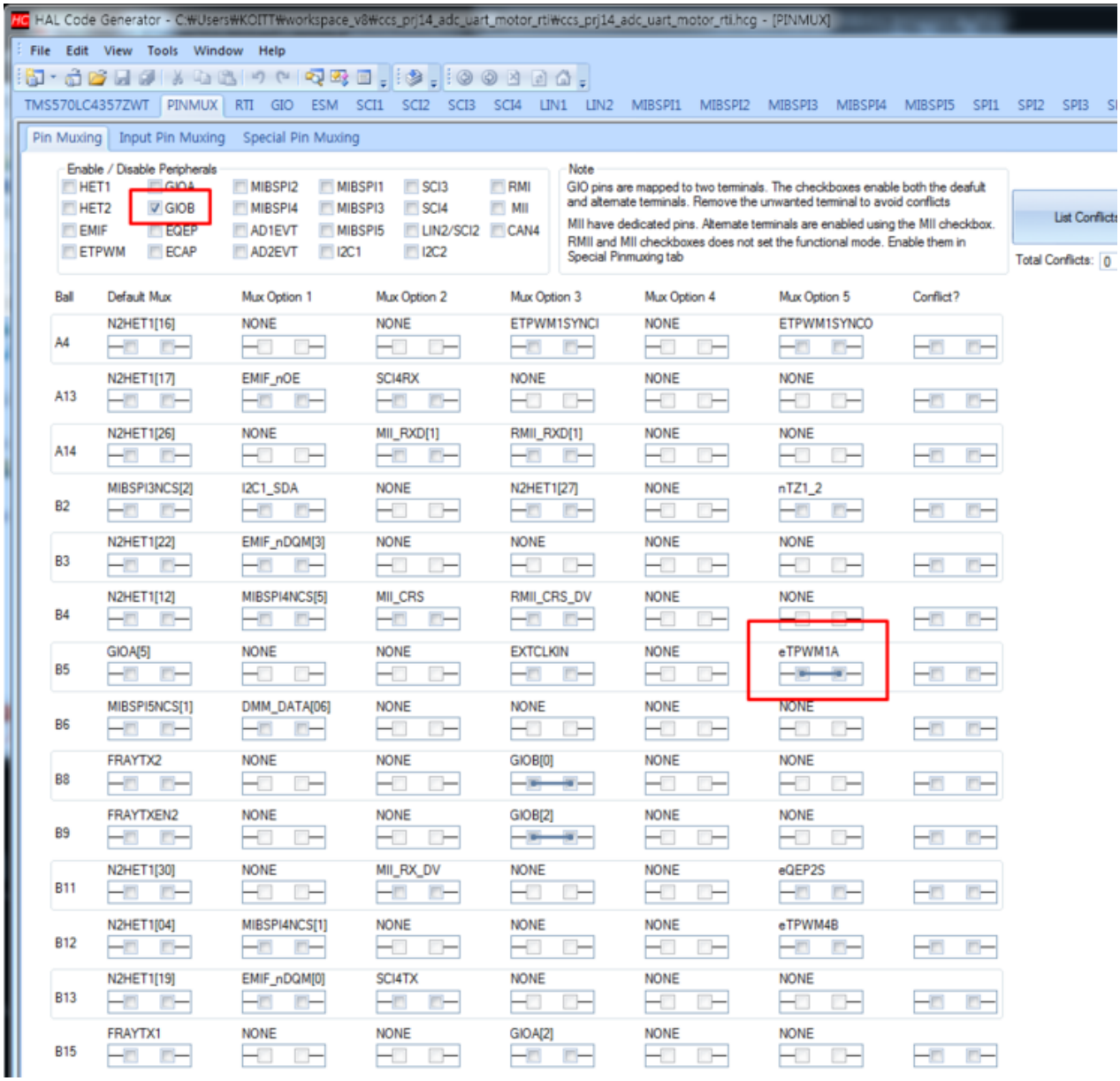
- GIO로 LED TRRIGER.
- ADC 값을 SCI로 출력.
- ADC 값을 이용해서 MOTOR 제어

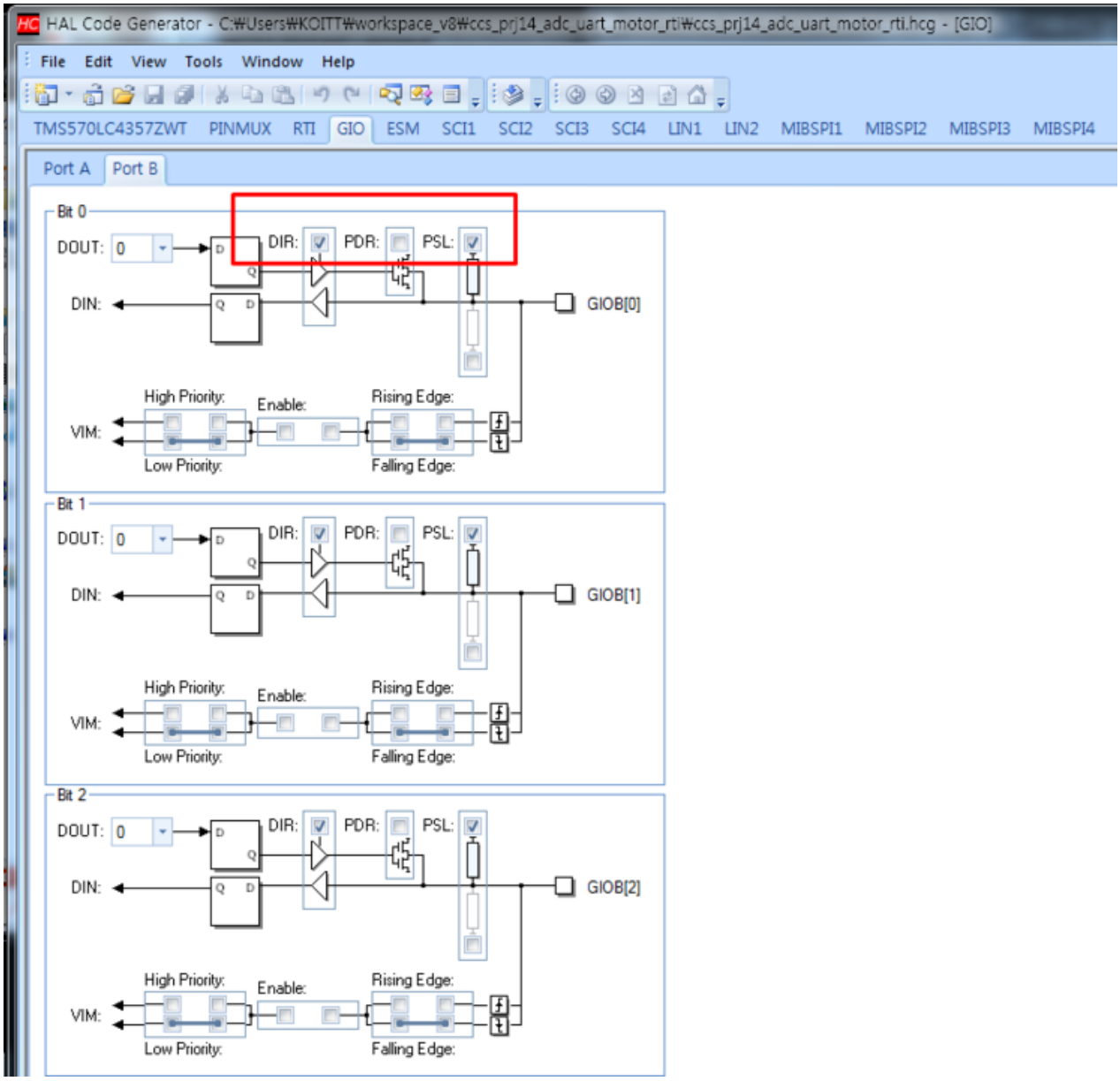
- RTI로 LED 트리거 돌리기.
- etpwm으로 servo mortor 돌리기.

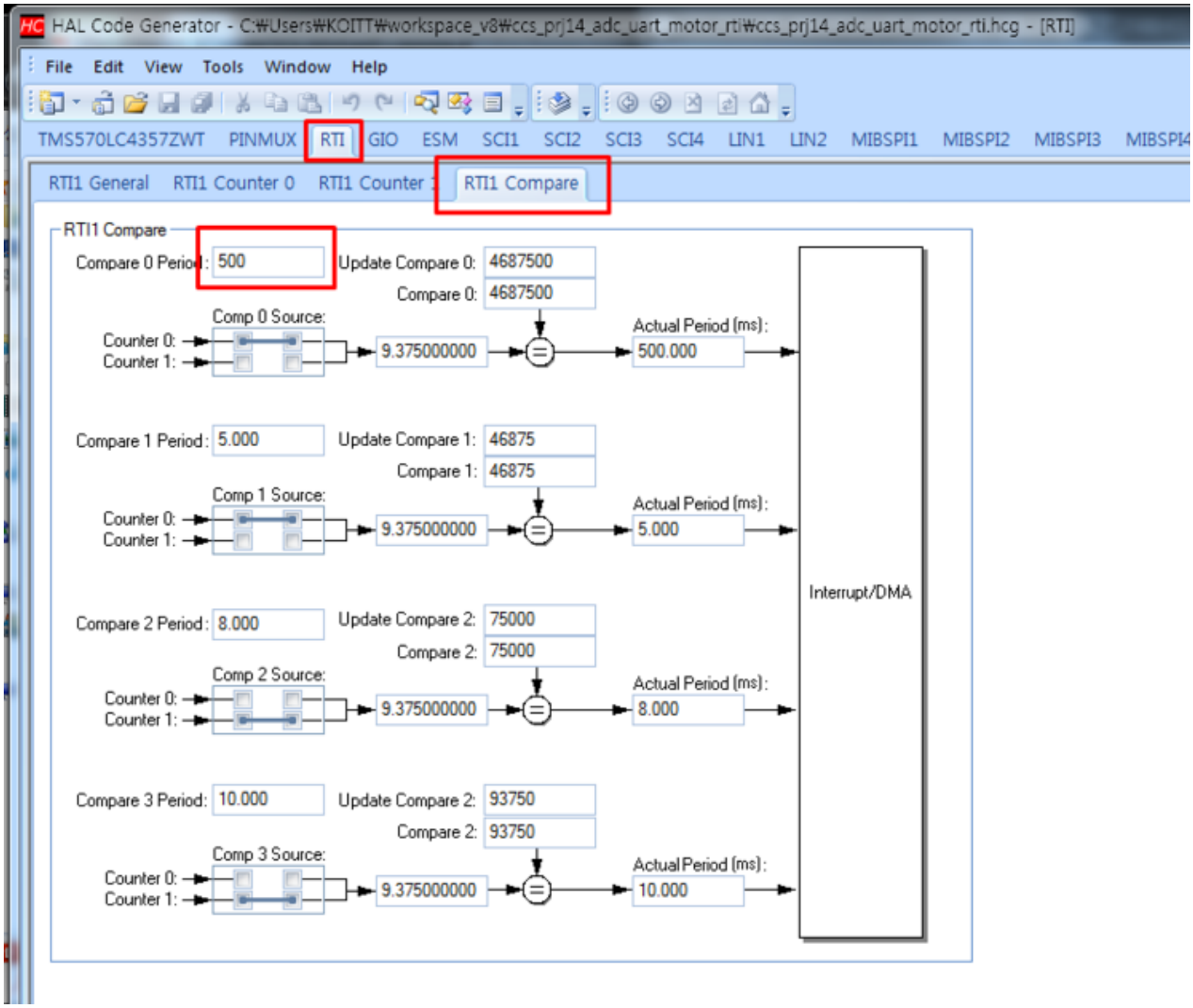


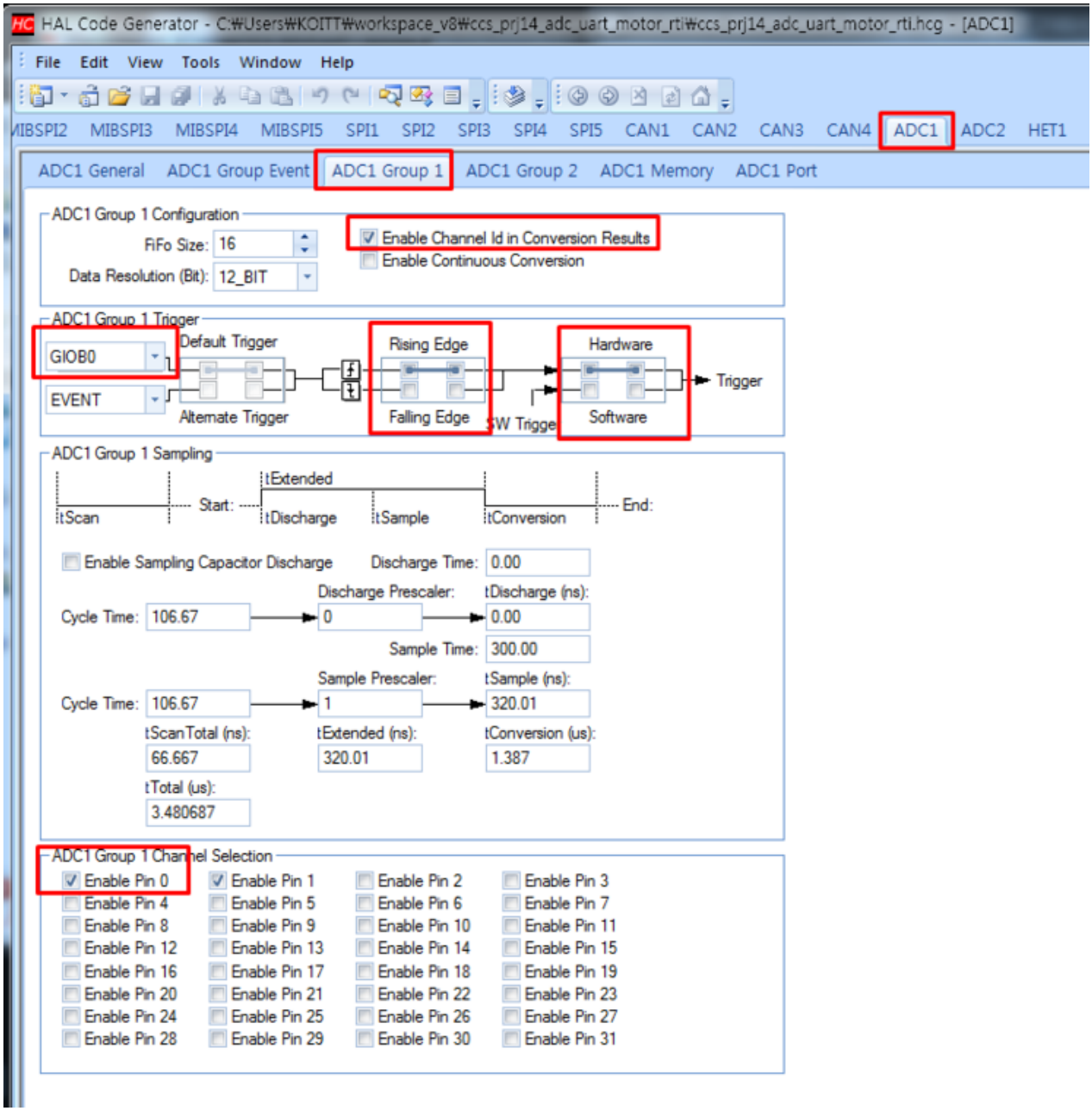

HAL Code Generator - C:₩Users₩KOITT₩workspace_v8₩ccs_prj14_adc_uart_motor_rti₩ccs_prj14_adc_uart_motor_rti.hcg - [TMS570LC435
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3
General Driver Enable R5-MPU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel 32-63
VIM Channel 0-31 Configuration
Interrupt Assignment
CHANMAP0-CHANMAP31
0 : ESM High
1 : Reserved
2 : RTI Compare 0
3 : RTI Compare 1
4 : RTI Compare 2
5 : RTI Compare 3
6 : RTI Overflow 0
7 : RTI Overflow 1
8 : RTI Timebase 1
9 : GIO High
10 : HET1 High
11 : HET1 TU High
12 : MIBSPI1 High
IRQ
FIQ

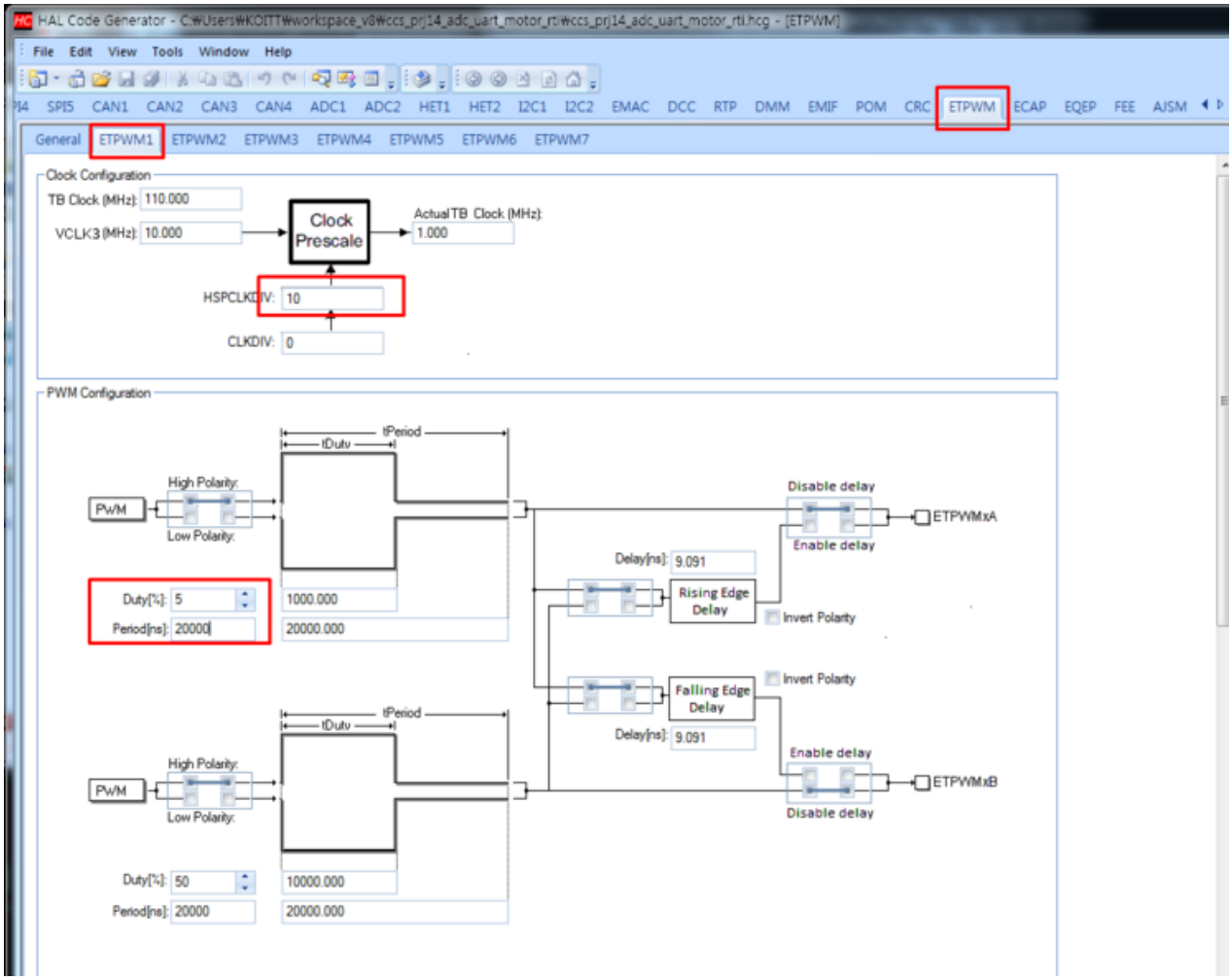

HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj14_adc_uart_motor_rti#ccs_prj14_adc_uart_motor_rti.hcg - [TMS570LC4357ZWT]
File Edit View Tools Window Help
Start Page
TMS570LC4357ZWT
PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5
PU-PMU Interrupts VIM General VIM RAM VIM Channel 0-31 VIM Channel 32-63 VIM Channel 64-95 VIM Channel 96-127 RAM Flash GCM
GCM
Clk Srcs
Standard Src
PLL1
Wakup Src
PLL1
Power Down Src
PLL1
HCLK Divider
1
GCLK Config
HCLK Config
VCLK1 Divider
1
VCLK
VCLK1 Config
VCLK2 Divider
1
VCLK2 Config
VCLK3 Divider
14
VCLK3 Config
VCLKA1 Src
VCLK
VCLKA1 Config
VCLKA2 Src
VCLK
VCLKA2 Config
RTI1 Pre Src
PLL1
RTI1 Divider
1
RTI1 Post Src
VCLK
RTI1 Config
VCLKA4 Src
VCLK
VCLKA4 Config
VCLKA4 Divider
1
VCLKA4 Post Src
VCLKA4_DIVR
PLL2
Clk Domains
GCLK 300.000
HCLK 150.000
VCLK1 75.000
VCLK2 75.000
VCLK3 10.000
VCLKA1 75.000
VCLKA2 0.000
RTI1CLK 75.000
VCLKA4_DIV 75.000
VCLKA4_S 75.000
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
VCLKA4 Max -- 110 MHz
RTICLK Max -- VCLK Freq
Clk Srcs
Oscillator 16.0
PLL1 300.00
External 1 00.0
LPO Low 0.080
LPO High 10.000
PLL2 300.00
External 2 00.0
Clock Tree
Clock Tree


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj14_adc_uart_motor_rti#ccs_prj14_adc_uart_motor_rti.hcg - [PINMUX]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3
Pin Muxing Input Pin Muxing Special Pin Muxing
Enable / Disable Peripherals
HET1 HET2 EMIF ETPWM
GIOA GIOB EQEP ECAP
MIBSPI2 MIBSPI4 AD1EVT AD2EVT
MIBSPI1 MIBSPI3 MIBSPI5 I2C1
SCI3 SCI4 LIN2/SCI2 I2C2
RMI MII CAN4
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
List Conflicts
Total Conflicts: 0
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
B6 MIBSPI5NCS[1] DMM_DATA[06] NONE NONE NONE NONE
B8 FRAYTX2 NONE NONE GIOB[0] NONE NONE
B9 FRAYTXEN2 NONE NONE GIOB[2] NONE NONE
B11 N2HET1[30] NONE MII_RX_DV NONE NONE eQEP2S
B12 N2HET1[04] MIBSPI4NCS[1] NONE NONE NONE eTPWM4B
B13 N2HET1[19] EMIF_nDQM[0] SCI4TX NONE NONE NONE
B15 FRAYTX1 NONE NONE GIOA[2] NONE NONE


HAL Code Generator - C:₩Users₩KOITT₩workspace_v8₩ccs_prj14_adc_uart_motor_rti₩ccs_prj14_adc_uart_motor_rti.hcg - [GIO]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4
Port A Port B
Bit 0
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[0]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:
Bit 1
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[1]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:
Bit 2
DOUT: 0
DIR:
PDR:
PSL:
DIN:
GIOB[2]
High Priority:
Enable:
Rising Edge:
VIM:
Low Priority:
Falling Edge:


HAL Code Generator - C:#Users#KOITT#workspace_v8#ccs_prj14_adc_uart_motor_rti#ccs_prj14_adc_uart_motor_rti.hcg - [RTI]
File Edit View Tools Window Help
TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4
RTI1 General RTI1 Counter 0 RTI1 Counter 1 RTI1 Compare
RTI1 Compare
Compare 0 Period: 500
Update Compare 0: 4687500
Compare 0: 4687500
Comp 0 Source:
Counter 0:
Counter 1:
9.375000000
Actual Period (ms): 500.000
Compare 1 Period: 5.000
Update Compare 1: 46875
Compare 1: 46875
Comp 1 Source:
Counter 0:
Counter 1:
9.375000000
Actual Period (ms): 5.000
Compare 2 Period: 8.000
Update Compare 2: 75000
Compare 2: 75000
Comp 2 Source:
Counter 0:
Counter 1:
9.375000000
Actual Period (ms): 8.000
Compare 3 Period: 10.000
Update Compare 2: 93750
Compare 2: 93750
Comp 3 Source:
Counter 0:
Counter 1:
9.375000000
Actual Period (ms): 10.000
Interrupt/DMA


HAL Code Generator - C:₩Users₩KOITT₩workspace_v8₩ccs_prj14_adc_uart_motor_rti₩ccs_prj14_adc_uart_motor_rti.hcg - [ADC1]
File Edit View Tools Window Help
MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4 SPI5 CAN1 CAN2 CAN3 CAN4 ADC1 ADC2 HET1
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
Software
Trigger
ADC1 Group 1 Sampling
tScan
Start:
tExtended
tDischarge
tSample
tConversion
End:
Enable Sampling Capacitor Discharge
Discharge Time: 0.00
Discharge Prescaler:
tDischarge (ns):
Cycle Time: 106.67
0
0.00
Sample Time: 300.00
Sample Prescaler:
tSample (ns):
Cycle Time: 106.67
1
320.01
tScanTotal (ns):
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
Enable Pin 4
Enable Pin 5
Enable Pin 6
Enable Pin 7
Enable Pin 8
Enable Pin 9
Enable Pin 10
Enable Pin 11
Enable Pin 12
Enable Pin 13
Enable Pin 14
Enable Pin 15
Enable Pin 16
Enable Pin 17
Enable Pin 18
Enable Pin 19
Enable Pin 20
Enable Pin 21
Enable Pin 22
Enable Pin 23
Enable Pin 24
Enable Pin 25
Enable Pin 26
Enable Pin 27
Enable Pin 28
Enable Pin 29
Enable Pin 30
Enable Pin 31




## 2) CCS 프로그래밍 하기

〈코드 1번〉 - 이렇게 해보니 단점이 있었다.

- 왜 인지 모르게 while문에 delay 외의 어떤 동작을 넣던 멈춘다.
- 원래 목적에 맞게는 잘 구동된다.

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include <HL_adc.h>
#include <HL_etpwm.h>
#include <HL_gio.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_adc.h>
#include <HL_reg_etpwm.h>
#include <HL_reg_gio.h>
#include <HL_reg_rti.h>
#include <HL_reg_sci.h>
#include <HL_rti.h>
#include <HL_sci.h>
#include <HL_sys_core.h>
#include <string.h>
#include <stdlib.h>
```

```
#include <stdio.h>


void check_msg();

int idx = 0;
uint8 data_buffer[]= " ";
uint8 msg[] = {0};
adcData_t adc_data[2];


/* USER CODE BEGIN (2) */
void delay(uint32 time)
{
    while (time--)
        ;
}

void rtiNotification(rtiBASE_t *rtiREG,uint32 notification)
{
    //rtiStopCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    if(notification == 1U){
        gioSetBit(gioPORTB, 0, 1);

        while((adcIsConversionComplete(adcREG1, adcGROUP1)) == 0)
            ;
        adcGetData(adcREG1, adcGROUP1, adc_data);
        delay(500);

        check_msg();
        sprintf((char *)msg, "value0 = %d\r\n\n", adc_data[0].value);
        sciSend(sciREG1, strlen((const char *)msg), msg);

        delay(500);

        gioSetBit(gioPORTB, 0, 0);
    }
    //rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
}

void check_msg()
{
    rtiStopCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    if(adc_data[0].value > 0 && adc_data[0].value < 300){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 22, "\r\n Very Darkness \r\n");

        etpwmSetCmpA(etpwmREG1,500);
    }

    else if(adc_data[0].value >= 300 && adc_data[0].value < 1000)
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 22, "\r\n little Darkness \r\n");
```

```
            etpwmSetCmpA(etpwmREG1,1400);
        }
        else if(adc_data[0].value >= 1000 && adc_data[0].value < 1800)
        {
            sciSend(sciREG1, 2, "\r\n");
            sciSend(sciREG1, 11, "\r\n good \r\n");

            etpwmSetCmpA(etpwmREG1,1200);
        }
        ////////////////////////////////////////////////////////////////////////
        // 2 수동으로 동작하는 부분 //
        else if(adc_data[0].value >= 1800 && adc_data[0].value < 2500)
        {
            sciSend(sciREG1, 2, "\r\n");
            sciSend(sciREG1, 20, "\r\n little bright \r\n");

            etpwmSetCmpA(etpwmREG1,1900);
        }
        else if(adc_data[0].value >= 2500)
        {
            sciSend(sciREG1, 2, "\r\n");
            sciSend(sciREG1, 18, "\r\n Very bright \r\n");

            etpwmSetCmpA(etpwmREG1,2500);
        }

       rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

    }

    /* USER CODE END */

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

        gioInit();
        sciInit();
        rtiInit();
        adcInit();
        etpwmInit();

        rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);

        // adc 입력으로 들어오는 값을의 변화능 ㄹ시작해 준다. (ADC1의 adcGROUP1의 녀석이 가지고 있는 BIT의 컨버젼을 시작한다.)
        adcStartConversion(adcREG1, adcGROUP1);
        rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
        etpwmStartTBCLK();


    //    _disable_IRQ_interrupt_();
        _enable_IRQ_interrupt_();

        // 트리거를 해줄 포트 B를 0으로 기본설정 해준다.
        gioSetBit(gioPORTB, 0, 0);
```

```
    while(1)
    {

    }


/* USER CODE END */

    return 0;
}


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

〈 코드 2번 〉 - 원인을 알 수 있었다.

- rti 안에서는 printf 함수 계열을 쓸 수 없다는 것을 알았다.

아마도 rti나 인터럽트에서 무언가 함수를 빠져 나가게 하는 요소 등으로 꼬이는 것 같다.

```
/* USER CODE BEGIN (0) */
/* USER CODE END */

/* Include Files */

#include <HL_adc.h>
#include <HL_etpwm.h>
#include <HL_gio.h>
#include <HL_hal_stdtypes.h>
#include <HL_reg_adc.h>
#include <HL_reg_etpwm.h>
#include <HL_reg_gio.h>
#include <HL_reg_rti.h>
#include <HL_reg_sci.h>
#include <HL_rti.h>
#include <HL_sci.h>
#include <HL_sys_core.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <stdarg.h>

void check_msg();

int idx = 0;
uint8 data_buffer[]= " ";
uint8 msg[] = {0};
adcData_t adc_data[2]={0};


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

```
void delay(uint32 time)
{
    while (time--)
        ;
}
void rtiNotification(rtiBASE_t *rtiREG,uint32 notification)
{
    //rtiStopCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    if(notification == 1U){
        gioSetBit(gioPORTB, 0, 1);

        while((adcIsConversionComplete(adcREG1, adcGROUP1)) == 0)
            ;
        adcGetData(adcREG1, adcGROUP1, adc_data);
        delay(500);

        check_msg();

        delay(500);

        gioSetBit(gioPORTB, 0, 0);
    }
    //rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
}

void check_msg()
{
//    rtiStopCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    if(adc_data[0].value > 0 && adc_data[0].value < 300){
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 22, "\r\n Very Darkness \r\n");

        etpwmSetCmpA(etpwmREG1,500);
    }

    else if(adc_data[0].value >= 300 && adc_data[0].value < 1000)
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 22, "\r\n little Darkness \r\n");

        etpwmSetCmpA(etpwmREG1,1400);
    }
    else if(adc_data[0].value >= 1000 && adc_data[0].value < 1800)
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 11, "\r\n good \r\n");

        etpwmSetCmpA(etpwmREG1,1200);
    }
    ////////////////////////////////////////////////////////////////////////////
    // 2 수동으로 동작하는 부분 //
    else if(adc_data[0].value >= 1800 && adc_data[0].value < 2500)
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 20, "\r\n little bright \r\n");
```

```
        etpwmSetCmpA(etpwmREG1,1900);
    }
    else if(adc_data[0].value >= 2500)
    {
        sciSend(sciREG1, 2, "\r\n");
        sciSend(sciREG1, 18, "\r\n Very bright \r\n");

        etpwmSetCmpA(etpwmREG1,2500);
    }

//   rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);

}

/* USER CODE END */

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

    gioInit();
    sciInit();
    rtiInit();
    adcInit();
    etpwmInit();

    rtiEnableNotification(rtiREG1, rtiNOTIFICATION_COMPARE0);

    // adc 입력으로 들어오는 값을의 변화능 ㄹ시작해 준다. (ADC1의 adcGROUP1의 녀석이 가지고 있는 BIT의 컨버젼을 시작한다.)
    adcStartConversion(adcREG1, adcGROUP1);
    rtiStartCounter(rtiREG1, rtiCOUNTER_BLOCK0);
    etpwmStartTBCLK();


//    _disable_IRQ_interrupt_();
    _enable_IRQ_interrupt_();

    // 트리거를 해줄 포트 B를 0으로 기본설정 해준다.
    gioSetBit(gioPORTB, 0, 0);
    while(1)
    {
        sprintf((char *)msg, "value0 = %d\r\n\n", adc_data[0].value);
        sciSend(sciREG1, strlen((const char *)msg), msg);
        delay(5000000);
    }


/* USER CODE END */

    return 0;
}
```

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