임베디드시스템설계
EMBEDDED SYSTEM DESIGN

CHAPTER 09

STM32F AD 변환기를 이용한 가변 저항값 읽어오기

KYUNG HEE UNIVERSITY

Human-Robot Interaction Laboratory

9.1 AD 변환기의 구조 및 기능

AD 변환기의 특징

- Nucleo-F103 확장보드에 사용되는 STM32F103RB는 2개의 AD 변환기를 내장함
- 12비트 변환기이므로 표현할 수 있는 값의 범위 : 0x000~0xFFF (0~4096, 참고로 아두이노는 0~1023)
- 변환 종료 시 또는 아날로그 워치독 이벤트 발생 시에 인터럽트가 발생함
- 단일(single) 또는 연속(continuous) 변환 모드
- 스캔 (scan)모드에서는 채널 0부터 n 까지 자동 변환이 가능 (최대 16채널)
- AD 변환 시간
  - STM32F103xx performance line : 1μs(56MHz 동작 시), 1.17μs(72MHz 동작 시)
- ADC 공급 전압 요구사항 : Full speed 상황에 서 2.4V ~ 3.6V, Slow speed 상황에 서 1.8V 이하
- ADC input 범위 : Vref- (Vssa 에 연결됨)에서 Vref+ (Vdda 에 연결됨 또는 외부 전압 사항)

KYUNG HEE UNIVERSITY

9.1 AD 변환기의 구조 및 기능

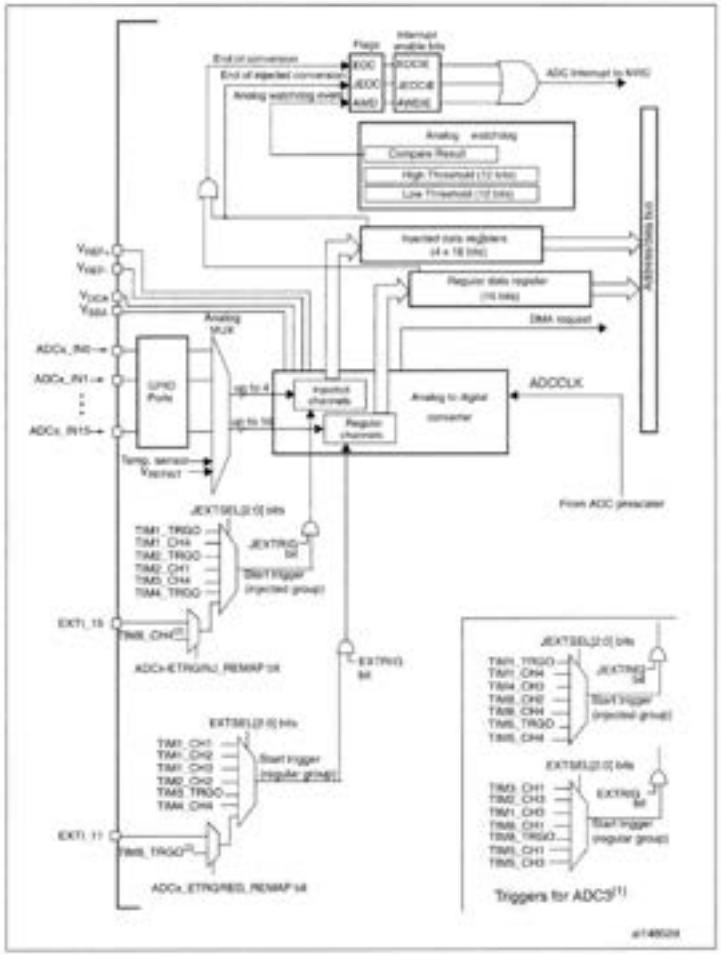
AD 변환기의 구성

- 중앙에 위치한 'Analog to Digital Converter' 가 외부의 아날로그 입력을 디지털로 변환해 주는 장치임
- AD 변환기의 내부에는 2개 채널(Regular channel, Injected channel) 의 변환 모듈이 있음
- 변환된 데이터는 2개의 데이터 레지스터(Regular data register, Injected data register)에 저장되어 어드레스/데이터 버스를 통해 MCU의 내부로 전달됨
- 외부의 아날로그 입력은 ADCxIN0 ~ ADCxIN15 의 16개 핀을 통해 입력됨
- 외부 인터럽트 입력 EXTI_11과 EXTI_15는 AD 변환을 시작하는 트리거 신호의 입력 핀으로 사용됨

KYUNG HEE UNIVERSITY

9.1 AD 변환기의 구조 및 기능

AD 변환기의 구성



AD 변환기의 구성도

Name	Signal type	Remarks
V_{REF+}	Input, analog reference positive	The higher/positive reference voltage for the ADC, $2.4V \leq V_{REF+} \leq V_{DDA}$
V_{DDA}[1]	Input, analog supply	Analog power supply equal to V_{DD} and $2.4V \leq V_{DDA} \leq 3.6V$
V_{REF-}	Input, analog reference negative	The lower/negative reference voltage for the ADC, $V_{REF-} = V_{SSA}$
V_{SSA}[1]	Input, analog supply ground	Ground for analog power supply equal to V_{SS}
ADCx_IN[15:0]	Analog signals	16 analog channels

AD 변환기의 외부 입력 핀

KYUNG HEE UNIVERSITY

9.1 AD 변환기의 구조 및 기능

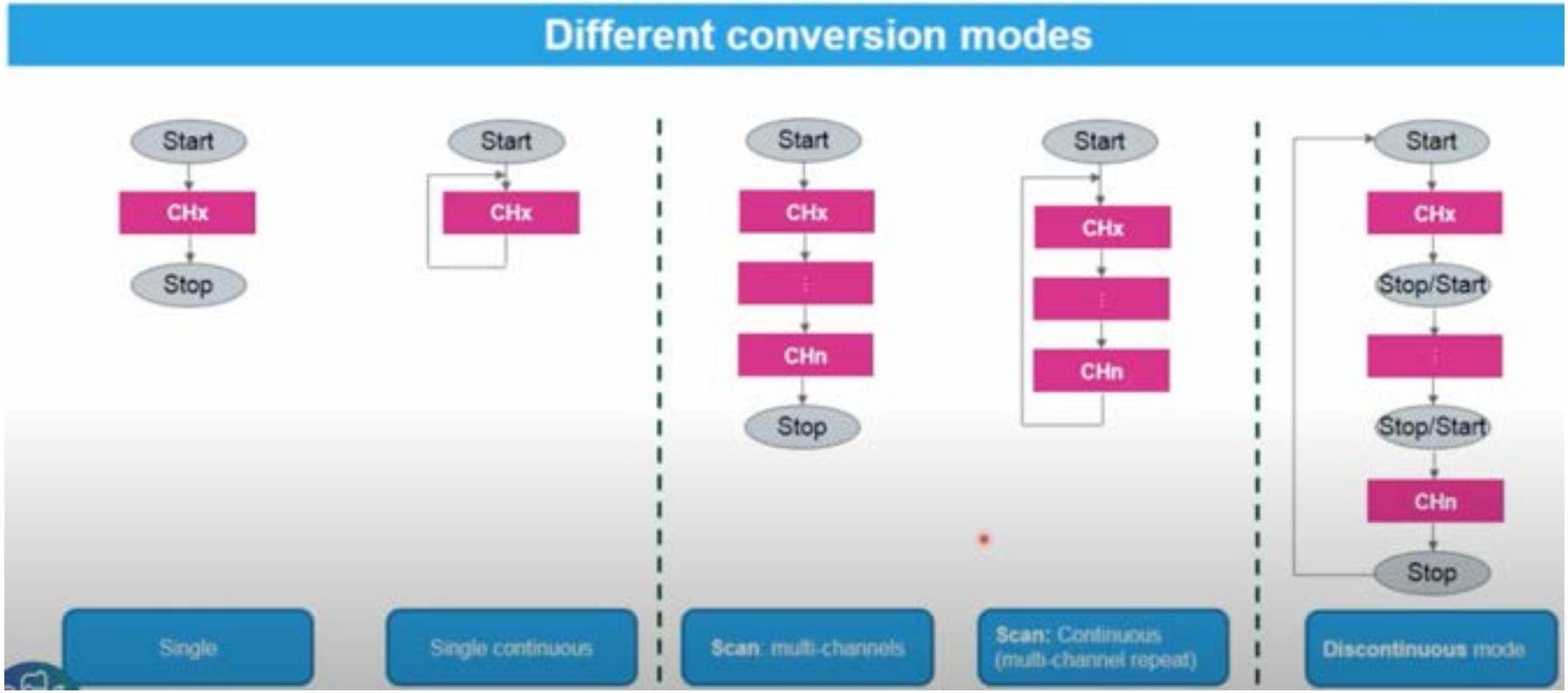
채널의 변환 순서

- 외부입력을 받는 167개 채널의 변환순서를 지정하기 위한 방법으로는 다음과 같은 2가지 방법이 있음
  - 레귤러 그룹(Regular group)을 이용 : ADC내에는 1개의 16비트 레귤러 데이터 레지스터가 있는데 이를 이용하면 채널의 변환순서를 16개까지 지정할 수 있음. 주로 ADC를 이 방법으로 많이 사용함

- 인젝티드 그룹(Injected group)을 이용 : ADC 내에는 4개의 16 비트 인젝티드 데이터 레지스터가 있는데 이를 이용하면 채널의 변환순서를 4개까지 지정할 수 있음. 인젝티드 그룹은 레귤러 그룹보다는 우선순위가 높음

9.1 AD 변환기의 구조 및 기능

채널의 변환 순서



9.1 AD 변환기의 구조 및 기능

주요 기능

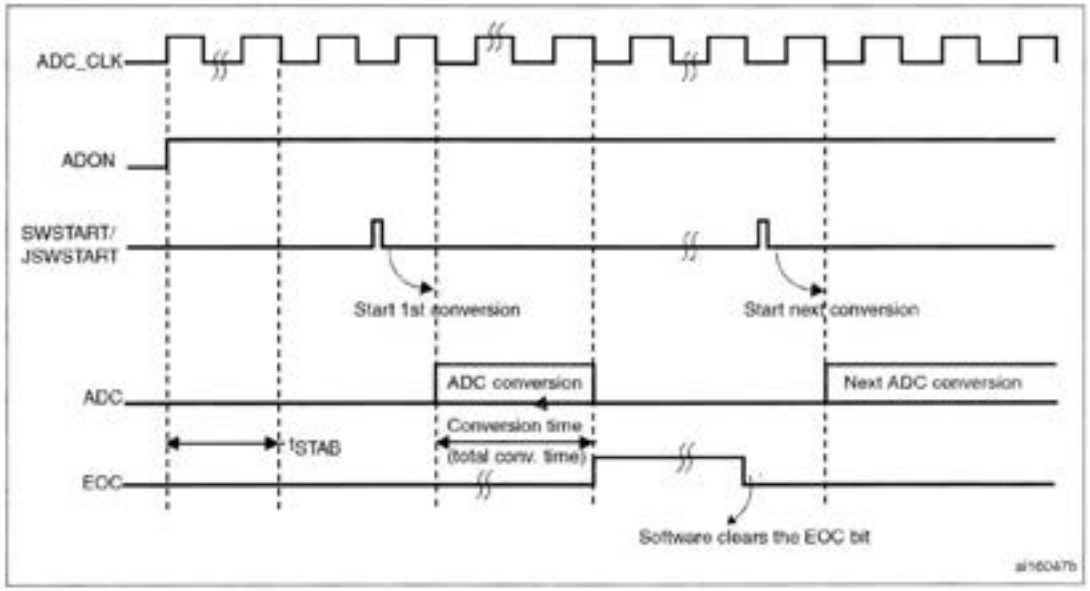
- 단일(Single) 변환 모드:
  - 단일 변환 모드에 서 AD 변환기는 변환을 1 번만 수행
  - ADC_CR2 레지스터의 CONT 비트를 0으로 두면 단일 변환 모드로 설정됨
  - ADC_CR2 레지스터의 ADON 비트가 설정되거나(레귤러 그룹의 경우) 또는 외부 트리거 입력이 인가되면(인젝티드 및 레귤러 그룹의 경우) 변환이 시작됨

- 연속(Continous) 변환 모드:
  - 연속 변환 모드에서 AD 변환기는 하나의 변환이 완료되면 바로 다음의 변환을 시작함
  - ADC_CR2 레지스터의 CONT 비트를 1 로 두면 연속 변환 모드로 설정됨
  - ADC_CR2 레지스터의 ADON 비트가 설정 되거나 또는 외부 트리거 입력이 인가되면 변환이 시작됨

9.1 AD 변환기의 구조 및 기능

동작 타이밍 선도

- AD 변환기는 전원 공급 후 제대로 동작하기 위해서는 t_{STAB} 만큼의 안정화(stabilization) 시간이 필요함
- 변환 시작 신호가 주어진 후 14클럭 사이클이 지나면 변환이 완료되어 변환 값이 16 비트의 ADC 데이터 레지스터에 저장되고 EOC(End of Completion) 플래그가 설정됨



AD 변환기의 동작 타이밍 선도

KYUNG HEE UNIVERSITY

9.1 AD 변환기의 구조 및 기능

주요 기능

- **스캔(Scan) 모드:**
  - 스캔 모드는 채널 그룹 내의 모든 채널들을 순서대로 스캔하여 변환하는 모드임
  - ADC_CR1 레지스터의 SCAN 비트를 1로 두면 스캔 모드로 설정됨. 이 모드에서는 채널 그룹(인젝티드 그룹 또는 레귤러 그룹)에 포함된 모든 채널들이 미리 정해진 순서를 따라 변환됨
  - 연속 변환 모드인 경우는 스캔 모드의 동작이 연속적으로 일어나게 됨

- **비연속 모드:**
  - 비연속 모드는 채널 그룹 내의 일부 채널을 미리 정해진 순서대로 변환하는 모드임
  - 외부 트리거 신호가 입력되면 n 개 (n < =8) 채널의 변환만 이루어짐
  - 다시 외부 트리거 신호가 입력되면 다음의 n 개의 변환이 이루어지며, n 개가 되지 않더라도 정해진 변환 순서의 끝이 되면 그대로 종료됨

  ex) (변환의 예) n = 3 이고, 채널의 변환 순서 = 0, 1, 2, 3, 6, 7, 9, 10 일 경우
  - 1 번째 트리거 입력 : 채널 0, 1, 2가 변환됨
  - 2 번째 트리거 입력 : 채널 3, 6, 7 이 변환됨
  - 3 번째 트리거 입력 : 채널 9, 10 이 변환되고 변환 종료(EOC) 이벤트 발생
  - 4번째 트리거 입력 : 다시 채널 0, 1, 2가 변환됨
  - 이후는 통일한 방식의 동작이 반복됨

9.1 AD 변환기의 구조 및 기능

주요 기능

- 데이터 정렬:
  - AD 변환기의 데이터 레지스터에 데이터 를 저장하기 위한 정렬 방식에는 오른쪽 정렬(Right aligned) 과 왼쪽 정렬 (Left aligned) 방식의 두 가지 방법이 있음
  - AD 변환기는 12비트이므로 오른쪽 정렬의 경우, 인젝티드 그룹은 최상위 비트 4개 가 SEXT라는 값으로 설정됨
  - SEXT는 변환 결과가 음수인가 양수인가에 대한 부호를 나타내는 값임
  - 레귤러 그룹의 경우는 최상위 비트 4개 가 모두 0 으로 설정됨
-
  - 왼쪽 정렬의 경우, 인젝티드 그룹은 최상위 비트 1개가 SEXT로 설정되며 최하위 비트 3 개 는 0으로 설정됨. 레귤러 그룹의 경우는 최하위 비트 4개가 모두 0으로 설정된다.

데이터 레지스터의 정렬 방식

KYUNG HEE UNIVERSITY

9.1 AD 변환기의 구조 및 기능

주요 기능

- **외부 트리거에 의한 변환 시작:**
  - ADC 컨트롤 레지스터 2(ADC_CR2) 의 EXTSEL[2:0] 비트를 설정을 하면 레귤러 채널에 대하여 외부 트리거 신호에 의한 변환 시작이 가능함

- **ADC 인터럽트:**
  - AD 변환이 완료된 경우 인터럽트가 발생함
  - ADC1 과 ADC2 의 인터럽트는 동일한 인터럽트 벡터를 가지며 ADC3은 이와는 다른 인터럽트 벡터를 가짐

9.2 AD 변환기 관련 HAL 드라이버

ADC 설정용 구조체의 종류

- ADC_InitTypeDef
  - ADC 의 초기 설정을 위한 구조체
- ADC_ChannelConfTypeDef
  - ADC 의 채널 설정을 위한 구조체
- ADC_AnalogWDGConfTypeDef
  - ADC 의 아날로그 워치독(analog watchdog) 설정을 위한 구조체
- ADC_HandleTypeDef
  - ADC 의 핸들 설정을 위한 구조제

9.2 AD 변환기 관련 HAL 드라이버

주요 구조체의 상세 설명

- 다음은 ADC 설정용 구조체 중에서 중요한 구조체에 대한 상세 설명을 나타냄
  - 나머지 구조체에 대한 자세한 설명은 ST사의 [HAL 드라이버 User Manual]을 참조

ADC_InitTypeDef

ADC의 초기 설정을 위한 구조체이며 stm32f1xx_hal_adc.h에 정의되어 있다.

[데이터 형]

- uint32_t DataAlign
- uint32_t ScanConvMode
- uint32_t ContinuousConvMode
- uint32_t NbrOfConversion
- uint32_t DiscontinuousConvMode
- uint32_t NbrOfDiscConversion
- uint32_t ExternalTrigConv

[데이터 형의 설명]

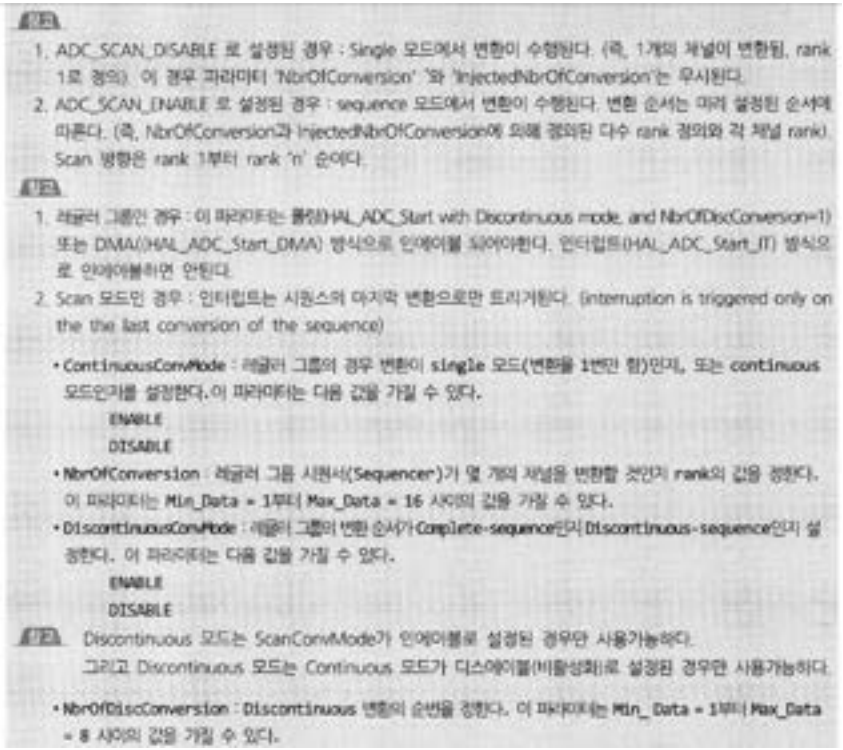
- DataAlign : ADC의 데이터 정렬 방법(right 또는 left)을 설정한다. 이 파라미터는 다음 값을 가질 수 있다.
  - ADC_DATAALIGN_RIGHT
  - ADC_DATAALIGN_LEFT
- ScanConvMode : 레귤러와 인젝티드 그룹의 시퀀서(Sequencer)들을 설정한다. 이 파라미터는 다음 값을 가질 수 있다.
  - ADC_SCAN_DISABLE
  - ADC_SCAN_ENABLE

KYUNG HEE UNIVERSITY

9.2 AD 변환기 관련 HAL 드라이버

주요 구조체의 상세 설명

- 다음은 ADC 설정용 구조체 중에서 중요한 구조체에 대한 상세 설명을 나타냄
  - 나머지 구조체에 대한 자세한 설명은 ST사의 [HAL 드라이버 User Manual]을 참조



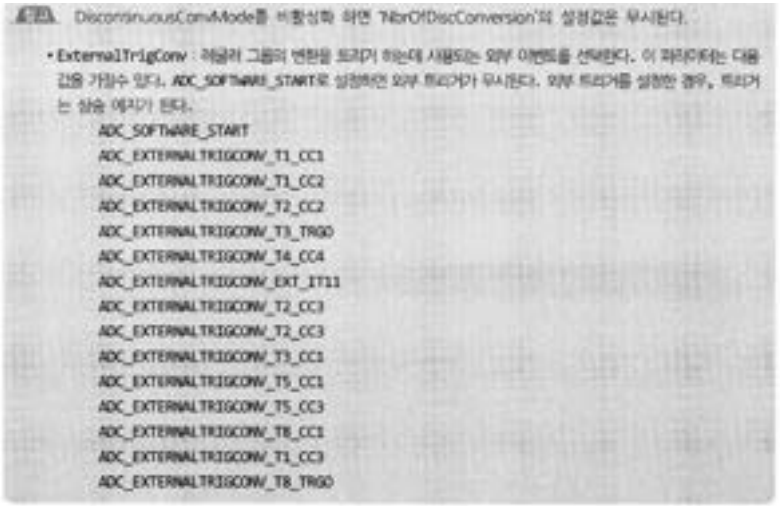
1. ADC_SCAN_DISABLE 로 설정된 경우 : Single 모드에서 변환이 수행된다. (즉, 1개의 채널이 변환됨, rank 1로 정의). 이 경우 파라미터 'NbrOfConversion' '와 'InjectedNbrOfConversion'는 무시된다.
2. ADC_SCAN_ENABLE 로 설정된 경우 : sequence 모드에서 변환이 수행된다. 변환 순서는 미리 설정된 순서에 따른다. (즉, NbrOfConversion과 InjectedNbrOfConversion에 의해 정의된 다수 rank 정의와 각 채널 rank). Scan 방향은 rank 1부터 rank 'n' 순이다.

1. 레귤러 그룹인 경우 : 이 파라미터는 폴링(HAL_ADC_Start with Discontinuous mode, and NbrOfDiscConversion=1) 또는 DMA(HAL_ADC_Start_DMA) 방식으로 인에이블 되어야한다. 인터럽트(HAL_ADC_Start_IT) 방식으로 인에이블하면 안된다.
2. Scan 모드인 경우 : 인터럽트는 시퀀스의 마지막 변환으로만 트리거된다. (interruption is triggered only on the the last conversion of the sequence)

- **ContinuousConvMode** : 레귤러 그룹의 경우 변환이 single 모드(변환을 1번만 함)인지, 또는 continuous 모드인지를 설정한다.이 파라미터는 다음 값을 가질 수 있다.
  - ENABLE
  - DISABLE
- **NbrOfConversion** : 레귤러 그룹 시퀀서(Sequencer)가 몇 개의 채널을 변환할 것인지 rank의 값을 정한다. 이 파라미터는 Min_Data = 1부터 Max_Data = 16 사이의 값을 가질 수 있다.
- **DiscontinuousConvMode** : 레귤러 그룹의 변환 순서가 Complete-sequence인지 Discontinuous-sequence인지 설정한다. 이 파라미터는 다음 값을 가질 수 있다.
  - ENABLE
  - DISABLE

Discontinuous 모드는 ScanConvMode가 인에이블로 설정된 경우만 사용가능하다.
그리고 Discontinuous 모드는 Continuous 모드가 디스에이블(비활성화)로 설정된 경우만 사용가능하다.

- **NbrOfDiscConversion** : Discontinuous 변환의 순번을 정한다. 이 파라미터는 Min_Data = 1부터 Max_Data = 8 사이의 값을 가질 수 있다.



DiscontinuousConvMode를 비활성화 하면 'NbrOfDiscConversion'의 설정값은 무시된다.

- **ExternalTrigConv** : 레귤러 그룹의 변환을 트리거 하는데 사용되는 외부 이벤트를 선택한다. 이 파라미터는 다음 값을 가질수 있다. ADC_SOFTWARE_START로 설정하면 외부 트리거가 무시된다. 외부 트리거를 설정한 경우, 트리거는 상승 에지이 된다.
  - ADC_SOFTWARE_START
  - ADC_EXTERNALTRIGCONV_T1_CC1
  - ADC_EXTERNALTRIGCONV_T1_CC2
  - ADC_EXTERNALTRIGCONV_T2_CC2
  - ADC_EXTERNALTRIGCONV_T3_TRGO
  - ADC_EXTERNALTRIGCONV_T4_CC4
  - ADC_EXTERNALTRIGCONV_EXT_IT11
  - ADC_EXTERNALTRIGCONV_T2_CC3
  - ADC_EXTERNALTRIGCONV_T2_CC3
  - ADC_EXTERNALTRIGCONV_T3_CC1
  - ADC_EXTERNALTRIGCONV_T5_CC1
  - ADC_EXTERNALTRIGCONV_T5_CC3
  - ADC_EXTERNALTRIGCONV_T8_CC1
  - ADC_EXTERNALTRIGCONV_T1_CC3
  - ADC_EXTERNALTRIGCONV_T8_TRGO

9.2 AD 변환기 관련 HAL 드라이버

주요 구조체의 상세 설명

- 다음은 ADC 설정용 구조체 중에서 중요한 구조체에 대한 상세 설명을 나타냄
  - 나머지 구조체에 대한 자세한 설명은 ST사의 [HAL 드라이버 User Manual]을 참조

ADC_ChannelConfTypeDef

ADC의 채널 설정을 위한 구조체이며 stm32f1xx_hal_adc.h에서 정의되어 있다.

[데이터 형]

- uint32_t Channel
- uint32_t Rank
- uint32_t SamplingTime

[데이터 형의 설명]

- Channel : 레귤러 그룹의 경우 설정을 할 채널을 정한다. 이 파라미터는 다음 값을 가질수 있다.

  ADC_CHANNEL_0 ADC_CHANNEL_1 ADC_CHANNEL_2
  ADC_CHANNEL_3 ADC_CHANNEL_4 ADC_CHANNEL_5
  ADC_CHANNEL_6 ADC_CHANNEL_7 ADC_CHANNEL_8
  ADC_CHANNEL_9 ADC_CHANNEL_10 ADC_CHANNEL_11
  ADC_CHANNEL_12 ADC_CHANNEL_13 ADC_CHANNEL_14
  ADC_CHANNEL_15 ADC_CHANNEL_16 ADC_CHANNEL_17
  ADC_CHANNEL_TEMPSENSOR ADC_CHANNEL_VREFINT

- Rank : 레귤러 그룹의 시퀀서의 랭크 순서(채널의 변환 순서)를 설정한다. 이 파라미터는 다음 값을 가질 수 있다.

  ADC_REGULAR_RANK_1 ADC_REGULAR_RANK_2 ADC_REGULAR_RANK_3
  ADC_REGULAR_RANK_4 ADC_REGULAR_RANK_5 ADC_REGULAR_RANK_6
  ADC_REGULAR_RANK_7 ADC_REGULAR_RANK_8 ADC_REGULAR_RANK_9
  ADC_REGULAR_RANK_10 ADC_REGULAR_RANK_11 ADC_REGULAR_RANK_12
  ADC_REGULAR_RANK_13 ADC_REGULAR_RANK_14 ADC_REGULAR_RANK_15
  ADC_REGULAR_RANK_16

참고 어떤 채널의 변환순서를 변경하고 싶은 경우는 그 채널의 변환순서를 새로운 값으로 설정하면 된다.
(이 경우 이전의 설정값에 overwrite됨)

- SamplingTime : 샘플링 시간을 설정한다. 이 파라미터는 다음 값을 가질 수 있다.

  ADC_SAMPLETIME_1CYCLE_5 : Sampling time 1.5 ADC clock cycle
  ADC_SAMPLETIME_7CYCLES_5 : Sampling time 7.5 ADC clock cycles
  ADC_SAMPLETIME_13CYCLES_5 : Sampling time 13.5 ADC clock cycles
  ADC_SAMPLETIME_28CYCLES_5 : Sampling time 28.5 ADC clock cycles
  ADC_SAMPLETIME_41CYCLES_5 : Sampling time 41.5 ADC clock cycles
  ADC_SAMPLETIME_55CYCLES_5 : Sampling time 55.5 ADC clock cycles
  ADC_SAMPLETIME_71CYCLES_5 : Sampling time 71.5 ADC clock cycles
  ADC_SAMPLETIME_239CYCLES_5 : Sampling time 239.5 ADC clock cycles

9.2 AD 변환기 관련 HAL 드라이버

주요 구조체의 상세 설명

- 다음은 ADC 설정용 구조체 중에서 중요한 구조체에 대한 상세 설명을 나타냄
  - 나머지 구조체에 대한 자세한 설명은 ST사의 [HAL 드라이버 User Manual]을 참조

ADC_HandleTypeDef

ADC의 핸들 설정을 위한 구조체이며 stm32f1xx_hal_adc.h에 정의되어 있다.

[데이터 형]

• ADC_TypeDef *	Instance
• ADC_InitTypeDef	Init
• __IO uint32_t	NbrOfConversionRank
• DMA_HandleTypeDef *	DMA_Handle
• HAL_LockTypeDef	Lock
• __IO HAL_ADC_StateTypeDef	State
• __IO uint32_t	ErrorCode

[데이터 형의 설명]

• Instance	: ADC 이름 (예 : ADC 1)
• Init	: ADC의 초기 설정을 위한 ADC_Init Type Def 구조체
• NbrOfConversionRank	: ADC conversion rank counter
• DMA_Handle	: DMA Handler 포인터
• Lock	: ADC locking object
• State	: ADC의 동작상태
• ErrorCode	: ADC 에러 코드

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 HAL 함수의 종류

- (1) 초기화(Initialization) 및 초기화 해제(de-initialization)용 함수
  - HAL_ADCInit(), HAL_ADC_DeInit()
    - ADC 의 초기화, 초기화 해제
  - HAL_ADC_MspInit(), HAL_ADC_MspDeInit()
    - ADC Msp 의 초기화, 초기화 해제
- (2) 입출력용 함수
  - HAL_ADC_Start()
    - ADC 의 레귤러 그룹의 동작을 시작
  - HAL_ADC_Stop()
    - ADC 의 레귤러 그룹의 동작을 정지
  - HAL_ADC_PollForConversion()
    - ADC의 레귤러 그룹의 변환이 완료될 때까지 기다림
- • HAL_ADC_Start_IT()
    - ADC 의 레귤러 그룹의 동작을 시작하고 관련 인터럽트를 인에이블시킴
- HAL_ADC_Stop_IT()
    - ADC 의 레귤러 그룹의 동작을 정지하고 관련 인터럽트를 정지 시킴
- HAL_ADC_Start_DMA()
    - ADC 의 레귤러 그룹의 동작을 시작하고 관련 DMA 를 사용함
- HAL_ADC_Stop_DMA()
    - ADC 의 레귤러 그룹의 동작을 정지하고 관련 DMA 를 중지함
- HAL_ADC_GetValue()
    - 레귤러 그룹의 ADC 변환 결과 값을 반환
- HAL_ADC_IRQHandler()
    - ADC 관련 인터럽트의 핸들러

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 HAL 함수의 종류

- (3) 주변장치 제어 함수
  - HAL_ADC_ConfigChannel()
    - ADC 의 레귤러 그룹의 채 널의 동작 조건을 설정
  - HAL_ADC_AnalogWDGConfig()
    - Analog watchdog 의 동작 조건을 설정
- (4) 주변장치 상태 함수
  - HAL_ADC_GetState()
    - ADC 가 변환중인지 , 변환이 완료되었는지 등의 상태값을 읽어옴
  - HAL_ADC_GetError()
    - ADC 의 에러 코드를 읽어옴
- (5) ADCEx 관련 함수
  - HAL_ADCEx_Calibration_Start()
    - ADC 의 지동 Calibration을 실행함. 이 함수는 ADC 가 비 활성화된 상태에서 시용 해야함.
    - HAL_ADC_Start() 함수의 실행 이전이나, HAL_ADC_Stop() 함수의 실행 이후에 사용해야함
- (6) 콜백 함수(Callback function)
  - HAL ADC ErrorCallback()
  - HAL_ADC_LevelOutOfWindowCallback() (analog watchdog 콜백)
  - HAL_ADC_ConvCpltCallback()
  - HAL_ADC_ConvHalfCpltCallback()
  - HAL_ADCExJnjectedConvCpltCallback()

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 함수

- **(1) 초기화(Initialization) 및 초기화 해제(de-initialization)용 함수**

HAL_ADC_Init (ADC_HandleTypeDef * hadc)

파라미터 hadc의 설정값에 따라 ADC와 Regular 그룹을 초기화한다.

[파라미터]
- hadc : ADC handle

[반환 값] HAL status

HAL_ADC_DeInit (ADC_HandleTypeDef * hadc)

ADC MSP와 ADC 주변장치를 해제한다.

[파라미터]
- hadc : ADC handle

[반환 값] HAL status

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 함수

- **(2) 입출력용 함수**

HAL_ADC_Start (ADC_HandleTypeDef * hadc)

ADC를 활성화하고 레귤러 그룹의 변환을 시작한다.

[파라미터]
- hadc : ADC handle

[반환 값] HAL status

HAL_ADC_Stop (ADC_HandleTypeDef * hadc)

ADC를 비활성화하고 변환을 정지한다.

[파라미터]
- hadc : ADC handle

[반환 값] HAL status

HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

레귤러 그룹의 변환이 완료될 때까지 기다린다.

[파라미터]
- hadc : ADC handle
- Timeout : 타임 아웃 시간(단위는 msec)

[반환 값] HAL status

HAL_ADC_Start_IT (ADC_HandleTypeDef * hadc)

ADC를 활성화하고 레귤러 그룹의 변환을 시작한다. 그리고 관련 인터럽트를 인에이블시킨다.

[파라미터]
- hadc : ADC handle

[반환 값] 없음

HAL_ADC_Stop_IT (ADC_HandleTypeDef * hadc)

ADC를 비활성화하고 레귤러 그룹의 변환을 정지한다. 그리고 관련 인터럽트를 디스에이블시킨다.

[파라미터]
- hadc : ADC handle

[반환 값] 없음

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 함수

- **(2) 입출력용 함수**

HAL_ADC_Start_DMA (ADC_HandleTypeDef * hadc, uint32_t* pData, uint32_t Length)

ADC를 활성화하고 레귤러 그룹의 변환을 시작한다. 그리고 DMA를 인에이블하고 변환 결과를 DMA를 통해 전달한다.

[파라미터]
- hadc : ADC handle
- pData : 해당 데이터
- Length : 데이터 크기

[반환 값] 없음

HAL_ADC_Stop_DMA (ADC_HandleTypeDef * hadc)

ADC를 비활성화하고 레귤러 그룹의 변환을 정지한다. 그리고 DMA를 디스에이블한다.

[파라미터]
- hadc : ADC handle

[반환 값] 없음

HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)

레귤러 그룹의 ADC 변환 결과 값을 읽어온다.

[파라미터]
- hadc : ADC handle

[반환 값] ADC 변환 결과 값

HAL_ADC_IRQHandler (ADC_HandleTypeDef * hadc)

ADC 인터럽트의 처리를 위한 인터럽트 핸들러 함수

[파라미터]
- hadc : ADC handle

[반환 값] 없음

HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef * hadc)

DC가 Non blocking 모드로 동작할 경우 변환이 완료되면 호출되는 callback 함수

[파라미터]
- hadc : ADC handle

[반환 값] 없음

9.2 AD 변환기 관련 HAL 드라이버

ADC 구동용 함수

- **(3) 주변장치 제어 함수**

HAL_ADC_ConfigChannel (ADC_HandleTypeDef * hadc, ADC_ChannelConfTypeDef * sConfig)

파라미터 sConfig에 설정된 값으로 ADC 채널의 동작조건을 설정한다.

[파라미터]
- hadc : ADC handle
- sConfig : 해당히 그룹을 위한 ADC 채널 구조체

[반환 값] HAL status

- **(4) 주변장치 상태 함수**

HAL_ADC_GetState (ADC_HandleTypeDef * hadc)

ADC의 동작상태 값을 반환한다.

[파라미터]
- hadc : ADC handle

[반환 값] HAL state

HAL_ADC_GetError (ADC_HandleTypeDef * hadc)

ADC의 에러 상태를 반환한다.

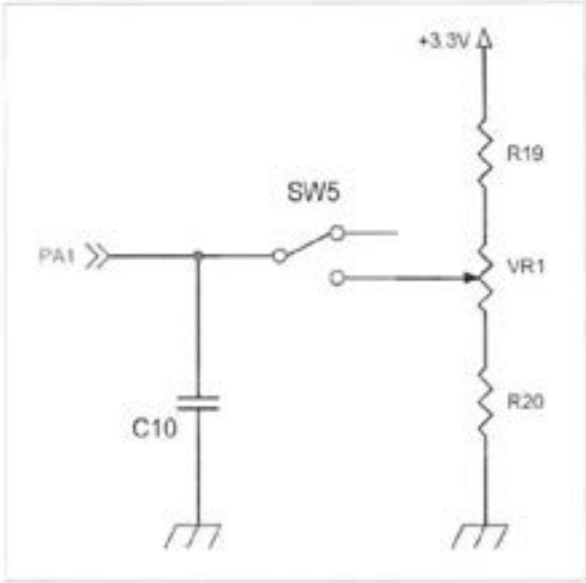
[파라미터]
- hadc : ADC handle

[반환 값] ADC 에러 코드

9.3 ADC 응용 예제

ADC 예제 관련 회로도

- Nucleo-F103 확장보드 :
  가변저항(VR 1)은 포트 A의 1번 핀(PA1)에 연결됨
- PA1(ADC123_IN1)는 ADC1, ADC2, ADC3의 채널 1의 입력핀으로도 사용 가능
- PC0(ADC12_IN10)는 ADC1, ADC2의 채널 10의 입력핀으로도 사용 가능
- 가변저항(VR1)을 회전시키면 핀에 입력되는 전압이 0V~ 3.3V의 범위 내에서 변하게 됨
- 가변 저항을 조절하면 ADC 로 입력되는 전압의 값을 변화시킬 수 있음



Nucleo-F103 확장보드

9.3 ADC 응용 예제

ADC 예제1 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

- 실습보드 내의 기변 저항(VR1)을 조절하여 ADC 1로 입력되는 전압의 값을 변화시킴
- ADC1의 출력값이 변하게 되며, 이에 따라 LED1 ~ 8의 On/Off 주기를 변화시킴
- 가변저항(VR1)을 움직이면 LED1~8 의 On/Off 주기가 바뀌게 됨

- 이 예제는 인터럽트 방식이 아니라 폴링 (poliing) 방식으로 ADC1 을 동작시킴
- 사용자 프로그램 내에서 HAL_ADC_Start() 함수를 이용하여 ADC1 을 동작시키면(즉, AD 변환을 시작하면) , 이 AD 변환이 완료될 때까지 프로그램은 다른 동작을 수행하지 않고 대기함

- 이 대기하는 동작은 HAL_ADC_PollForConversion() 함수를 이용하여 이루어짐

9.3 ADC 응용 예제

ADC 예제1 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

- (1) Nucleo-F103 확장보드를 이용하는 경우는 다음과 같이 소스 코드를 작성함
  - 폴더 명 : [Examples_Nucleo F103] - [ADC] - [ADC 1f103] - [MDK- ARM]
  - 위의 폴더에 있는 [Project.uvprojx] 파일을 더블클릭하여 실행함

- (2) 파일이 열리면 [Example/User] 폴더 내의 [main.c] 파일을 더블 클릭하여 연 후에 아래와 같이 소스 코드를 작성함

[main.c]

```
#include "main.h"
#include "Nucleo_F103.h"              // Nucleo-F103 확장보드용 헤더 파일
//#include "Nucleo_F429.h"            // Nucleo-F429 확장보드용 헤더 파일

// -- <1> AdcHandler, Adc_sConfig 변수를 외부정의 변수로 선언
extern ADC_HandleTypeDef              AdcHandler;
extern ADC_ChannelConfTypeDef         Adc_sConfig;
int adc_value;
int delay_time = 0;

// ------------------------------------------------------------------------ //
```

KYUNG HEE UNIVERSITY

9.3 ADC 응용 예제

ADC 예제1 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

[main.c]

```
#include "main.h"
#include "Nucleo_F103.h"              // Nucleo-F103 확장보드용 헤더 파일
//#include "Nucleo_F429.h"            // Nucleo-F429 확장보드용 헤더 파일

// -- <1> AdcHandler, Adc_sConfig 변수를 외부정의 변수로 선언
extern ADC_HandleTypeDef              AdcHandler;
extern ADC_ChannelConfTypeDef         Adc_sConfig;
int adc_value;
int delay_time = 0;

// ------------------------------------------------------------------------------ //
```

```
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    LED_Config();

    // -- <2> ADC의 초기설정용 함수를 호출
    ADC1_Polling_Config();
    // -- <3> 폴링으로 ADC값 입력받기
    while (1)
    {
                // -- <4> 설정된 Adc_sConfig와 AdcHandler를 이용하여 ADC를 초기화 함
                HAL_ADC_ConfigChannel(&AdcHandler, &Adc_sConfig);

                // -- <5> ADC를 동작시킴
                HAL_ADC_Start(&AdcHandler);

                // -- <6> 변환이 완료될 때까지 대기
                HAL_ADC_PollForConversion(&AdcHandler, 10);

                // -- <7> ADC 변환 결과 값을 저장
                adc_value = HAL_ADC_GetValue(&AdcHandler)* 6000;

                // -- <8> 변환 결과 값을 지연함수로 구현하여 LED On/Off 제어
                HAL_GPIO_WritePin (GPIOExt, GPIO_PIN_All, GPIO_PIN_SET);
                for(delay_time=0; delay_time<adc_value; delay_time++);
                HAL_GPIO_WritePin (GPIOExt, GPIO_PIN_All, GPIO_PIN_RESET);

                // -- <9> for문을 이용하여 delay시간을 0~1,000,000까지 구현
                for(delay_time=0; delay_time<1000000; delay_time++);
    }
}
```

9.3 ADC 응용 예제

ADC 예제1 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

[main.c]

<1> ADC의 초기화를 위하여 구조체 ADC_HandleTypeDef 형의 변수 AdcHandler와 ADC_ChannelConfTypeDef형의 변수 Adc_sConfig를 외부정의 변수로 선언한다. 이 변수들은 <Nucleo_F103.c> / <Nucleo_F429.c>에 선언되어 있다.

<2> ADC1_Polling_Config() 함수 : ADC1을 Polling 방식으로 동작하도록 설정해주는 함수이다. 이 함수는 각각 <Nucleo_F103.c> / <Nucleo_F429.c>에 정의되어 있다. [14.3절. 예제 소스코드 추가 설명 : Nucleo_F103.c 및 Nucleo_F429.c]을 참고하기 바란다.

** 이 함수에는 ADC의 초기 동작을 설정하는 소스코드가 있다. 이 소스코드는 ADC의 이해를 위해서 매우 중요한 코드이므로 반드시 찾아서 살펴보기 바란다.

<3> while(1) { } 부분은 항상 참(True)이다. 그러므로 이 부분은 무한 루프로 동작하며 폴링 방식으로 ADC 변환을 한다.

<4> HAL_ADC_ConfigChannel() 함수 : Adc_sConfig에 설정된 값으로 ADC 채널의 동작조건을 설정한다.

<5> HAL_ADC_Start() 함수 : ADC를 활성화하고 변환을 시작한다.

<6> HAL_ADC_PollForConversion() 함수 : 변환이 완료될 때까지 대기한다. 최대 대기시간(timeout 시간)은 10msec이다.

<7> HAL_ADC_GetValue() : 변환 결과 값을 받아 변수 adc_value에 저장한다.

<8> 변환 결과 값(adc_value) 만큼 LED를 On한 후에 Off한다.

<9> 시간지연을 위한 for 문이다.

9.3 ADC 응용 예제

ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

- 실습보드 내의 가변 저항(VR1)을 조절하여 ADC로 입력되는 전압의 값을 변화시키면 LED 의 On/Off 주기가 변경됨
- ADC 의 출력 값이 작아지면 주기가 빨라지고 반대의 경우는 느려짐

- 이 예제는 ADC 변환이 완료되면 발생하는 인터럽트를 이용함
- 따라서 인터럽트 핸들러루틴에서 LED를 On/Off하는 프로그램을 작성함
- ADC는 ADC1을 사용하며 연속 변환 모드로 설정하고, 변환 채널은 1개를 사용함

9.3 ADC 응용 예제

ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

- **(1) Nucleo-F103 확장보드를 이용하는 경우는 다음과 같이 소스 코드를 작성함**
  - 폴더 명 : [Examples_Nucleo F103] - [ADC] - [ADC 2J103] - [MDK- ARM]
  - 위의 폴더 에 있는 [Project. Uvprojx] 파일을 더블클럭하여 실행함

- **(2) 파일이 열리면 [Example/User] 폴더 내의 [main.c] 파일을 더블클릭하여 연 후에 아래와 같이 소스 코드를 작성함**

9.3 ADC 응용 예제

ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

[main.c]

```
#include "main.h"
#include "Nucleo_F103.h"              // Nucleo-F103 확장보드용 헤더 파일
//#include "Nucleo_F429.h"            // Nucleo-F429 확장보드용 헤더 파일

// -- <1> AdcHandler, Adc_sConfig 변수를 외부정의 변수로 선언
extern ADC_HandleTypeDef              AdcHandler;
extern ADC_ChannelConfTypeDef         Adc_sConfig;

int adc_value;
int delay_time = 0;
// ----------------------------------------------------------------------------//

int main(void)
{
   HAL_Init();
   SystemClock_Config();
   LED_Config();

   // -- <2> ADC의 Interrupt용 초기설정 함수를 호출
   ADC1_Interrupt_Config();

   // -- <3> 무한루프를 돌며 계속 기다림
   while (1) { }
}

// ----------------------------------------------------------------------------//
// -- <4> ADC 인터럽트 Callback 함수
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *AdcHandler)
{
   // ADC 변환 결과 값을 저장
   adc_value = HAL_ADC_GetValue(AdcHandler)* 6000;
   // 변환 결과 값을 지연함수로 구현하여 LED On/Off 제어
   HAL_GPIO_WritePin (GPIOExt, GPIO_PIN_All, GPIO_PIN_SET);
   for(delay_time=0; delay_time<adc_value; delay_time++);
   HAL_GPIO_WritePin (GPIOExt, GPIO_PIN_All, GPIO_PIN_RESET);

   // for문을 이용하여 delay시간을 0~1,000,000까지 구현
   for(delay_time=0;delay_time<1000000;delay_time++);
}
```

KYUNG HEE UNIVERSITY

9.3 ADC 응용 예제

ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

- **(1) 인터럽트 핸들러 함수 작성을 위해서 [Example/User 1 폴더 내의 [stm32f1xxjt.cl 파일을 더블클럭하여 연 후에 아래와 같이 소스 코드를 작성함**

[stm32f1xx_it.c]

```
#include "main.h"
#include "stm32f1xx_it.h"    //인터럽트 사용에 필요한 헤더파일

// -- <4> main.c에서 정의한 AdcHandler 변수를 외부정의변수로 선언
extern ADC_HandleTypeDef    AdcHandler;
```

```
// ------------------------------------------------------------------------//

void SysTick_Handler(void)
{
   HAL_IncTick();
}

// ------------------------------------------------------------------------//
// -- <5> 인터럽트가 발생하면 이를 처리하는 Handler 함수
void ADC1_IRQHandler(void)
{
   HAL_ADC_IRQHandler(&AdcHandler);
}
```

KYUNG HEE UNIVERSITY

9.3 ADC 응용 예제

ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

[main.c]

<1> ADC의 초기화를 위하여 구조체 ADC_HandleTypeDef 형의 변수 AdcHandler와 ADC_ChannelConfTypeDef 형의 변수 Adc_sConfig를 외부변수로 선언한다. 이 변수들은 <Nucleo_F103.c> / <Nucleo_F429.c>에 선언되어 있다.

<2> ADC1_Interrupt_Config() 함수 : ADC1의 초기설정용 함수를 Interrupt 방식으로 동작하도록 설정해주는 함수이다. 이 함수는 각각 <Nucleo_F103.c> / <Nucleo_F429.c>에 정의되어 있다. [14.3절. 예제 소스코드 추가 설명 : Nucleo_F103.c 및 Nucleo_F429.c]을 참고하기 바란다.

** 이 함수에는 ADC의 초기 동작을 설정하는 소스코드가 있다. 이 소스코드는 ADC의 이해를 위해서 매우 중요한 코드이므로 반드시 찾아서 살펴보기 바란다.

<3> while(1) { } 부분은 항상 참(True)이다. 그러므로 이 부분은 무한 루프로 동작하며 아무런 일도 하지 않는다. ADC 변환이 완료되면 그 결과는 아래의 <4>번 함수에서 처리해 준다.

<4> HAL_ADC_ConvCpltCallback( ) 함수는 인터럽트가 발생하면 호출되는 Callback 함수이다. 따라서 인터럽트 발생 시에 처리하고 싶은 내용을 이 함수 내에 코딩하면 된다.

[stm32f1xx_it.c]

<4> : 'main.c' 에서 정의한 AdcHandler 변수를 외부정의 변수로 선언하였다. 이 변수는 ADC1_IRQHandler( ) 함수에서 사용한다.

<5> : ADC1_IRQHandler( ) 함수는 ADC 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

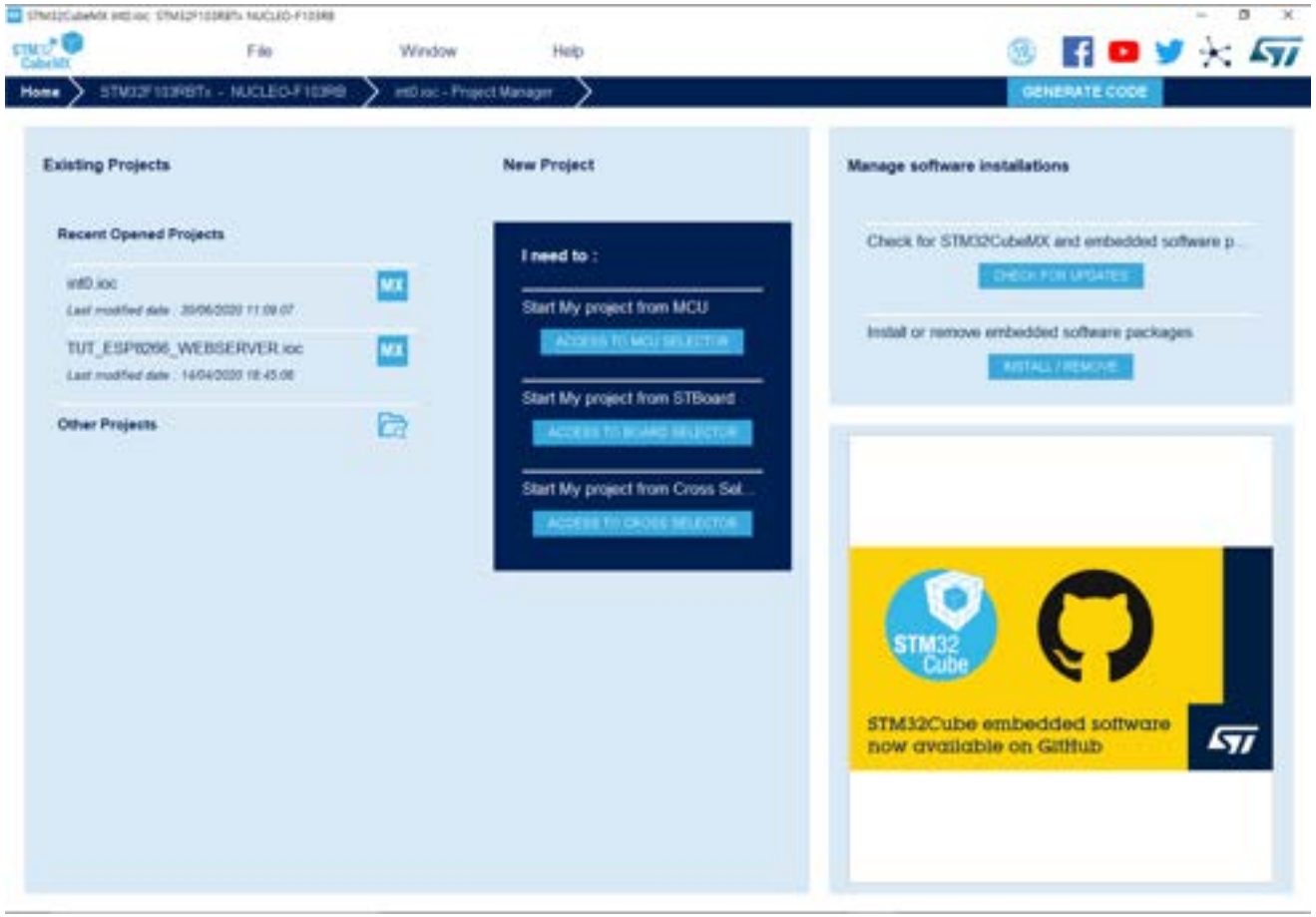
KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 구현하기

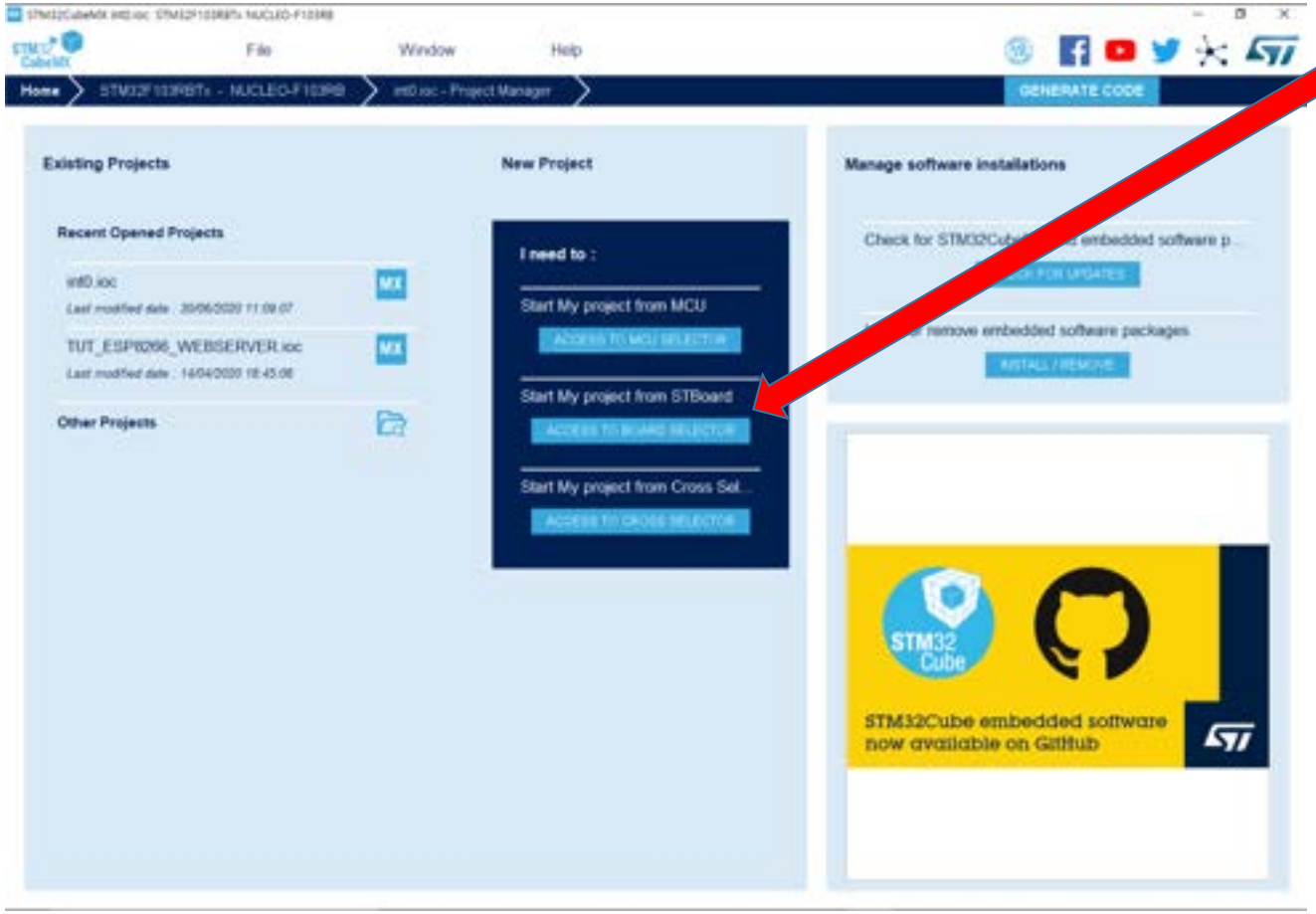
CubeMX 실행

초기화면



9.4 ADC CubeMX과제

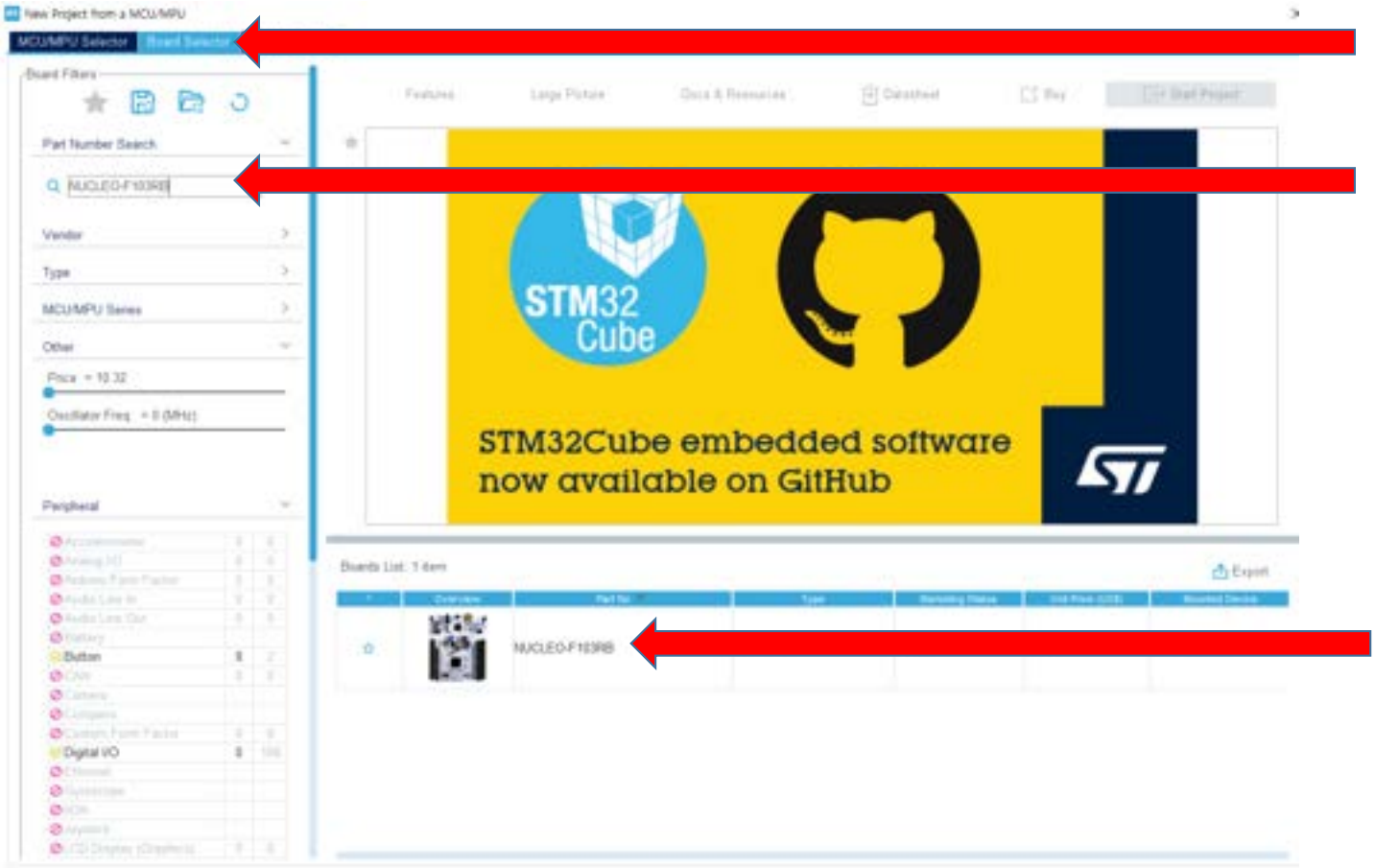
CubeMX로 예제 9.1 구현하기



보드 선택

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 구현하기



보드 선택

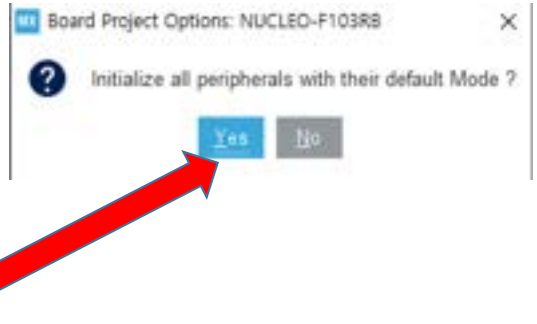
NUCLEO-F103RB 선택

NUCLEO-F103RB 더블클릭

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

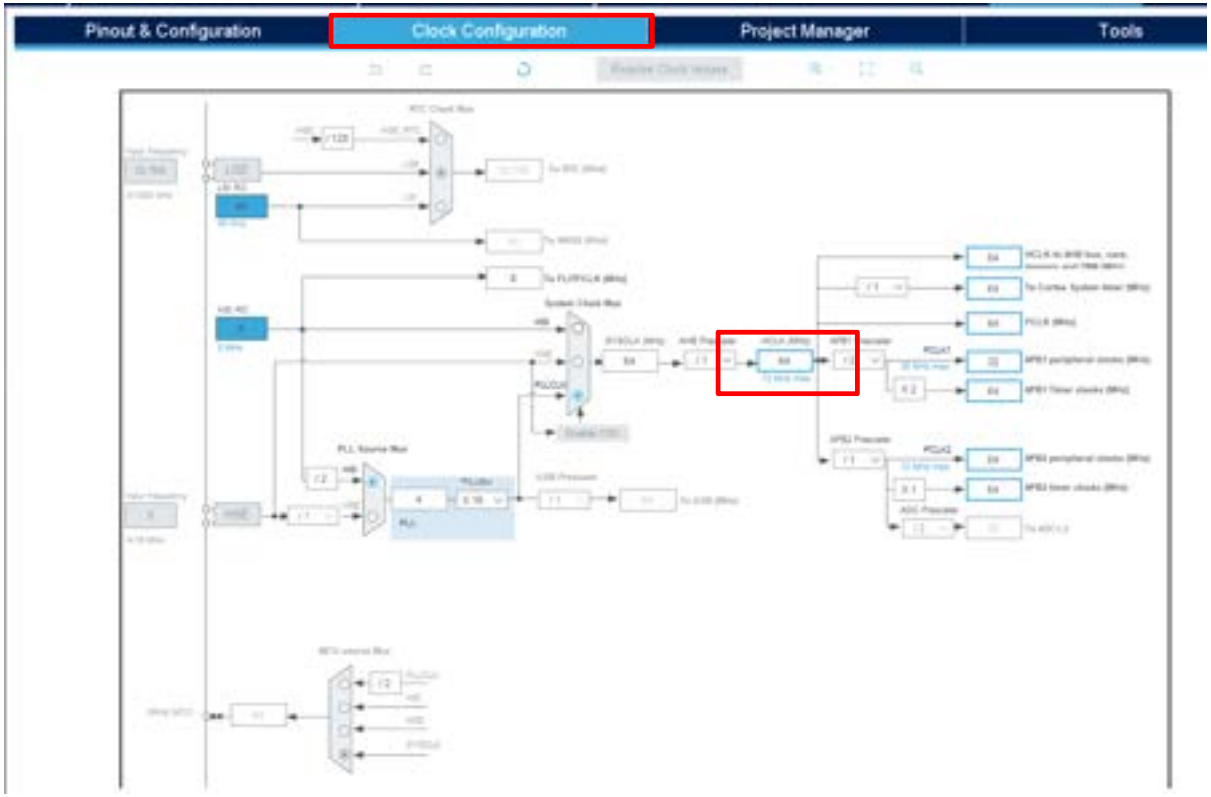
CubeMX로 예제 9.1 구현하기



Yes 선택하면 오른쪽과 같이 칩이 보임

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

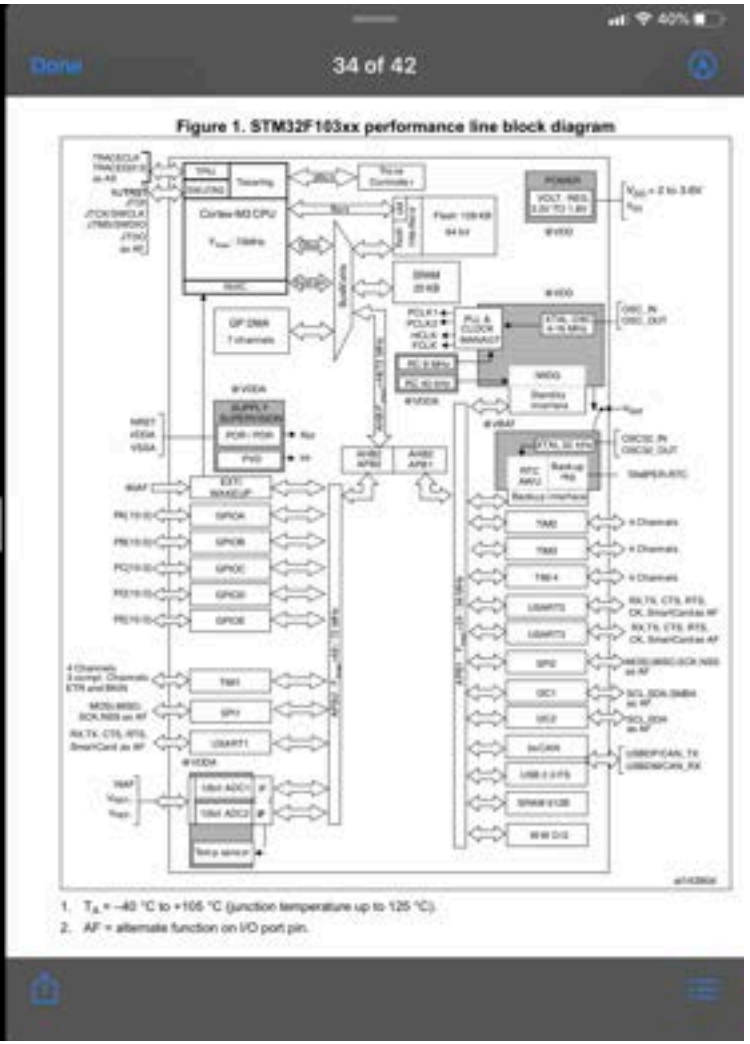


Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

```
 *          PLLMUL                  = 16
 *          Flash Latency(WS)       = 2
 */
// ------------------------------------------------------------------------- //

void  SystemClock_Config(void)
{
          RCC_ClkInitTypeDef clkinitstruct = {0};
          RCC_OscInitTypeDef oscinitstruct = {0};

          /* Configure PLL ------------------------------------------------------*/
          /* PLL configuration: PLLCLK = (HSI / 2) * PLLMUL = (8 / 2) * 16 = 64 MHz */
          /* PREDIV1 configuration: PREDIV1CLK = PLLCLK / HSEPredivValue = 64 / 1 = 64
MHz */
          /* Enable HSI and activate PLL with HSI_DIV2 as source */
          oscinitstruct.OscillatorType  = RCC_OSCILLATORTYPE_HSI;
          oscinitstruct.HSEState        = RCC_HSE_OFF;
          oscinitstruct.LSEState        = RCC_LSE_OFF;
          oscinitstruct.HSIState        = RCC_HSI_ON;
          oscinitstruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
          oscinitstruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
          oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
          oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_HSI_DIV2;
          oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;

          if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)  {
                    /* Initialization Error */
                    while(1);
          }

          /* Select PLL as system clock source and configure the HCLK, PCLK1 and
PCLK2
              clocks dividers */
          clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
          clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
          clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
          clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
          clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;

          if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)  {
                    /* Initialization Error */
                    while(1);
          }
}

// ------------------------------------------------------------------------- //
//  --  <18>  Clock 설정시 에러가 발생하면 처리해주는 함수
/**
```



KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

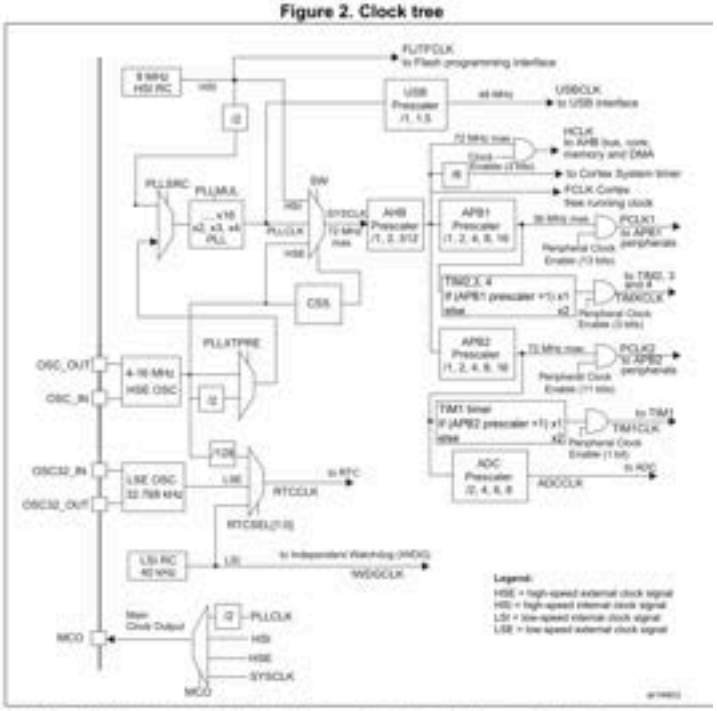
8:41 PM Mon Oct 28

Done 2 of 4

```
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  /* Configure PLL ------------------------------------------------------*/
  /* PLL configuration: PLLCLK = (HSI / 2) * PLLMUL = (8 / 2) * 16 = 64 MHz */
  /* PREDIV1 configuration: PREDIV1CLK = PLLCLK / HSEPredivValue = 64 / 1 = 64
MHz */
  /* Enable HSI and activate PLL with HSi_DIV2 as source */
  oscinitstruct.OscillatorType  = RCC_OSCILLATORTYPE_HSI;
  oscinitstruct.HSEState        = RCC_HSE_OFF;
  oscinitstruct.LSEState        = RCC_LSE_OFF;
  oscinitstruct.HSIState        = RCC_HSI_ON;
  oscinitstruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  oscinitstruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
  oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_HSI_DIV2;
  oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;

  if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)  {
      /* Initialization Error */
      while(1);
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and
PCLK2
      clocks dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
  clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)  {
      /* Initialization Error */
      while(1);
  }
}
```

Done 34 of 42

Figure 2. Clock tree

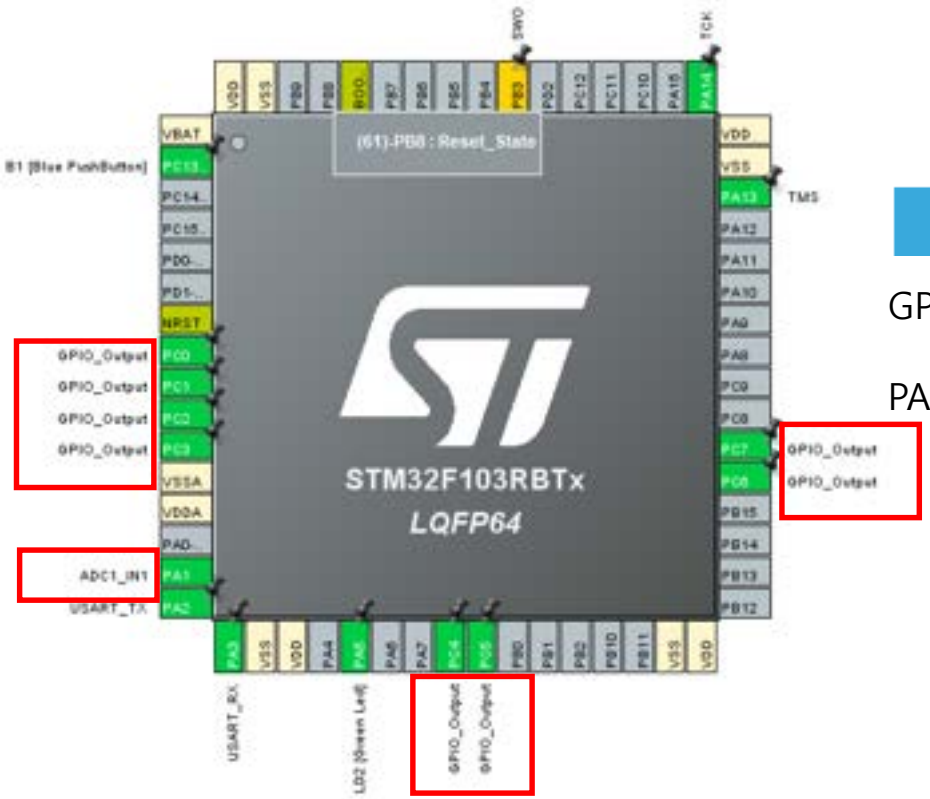


1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
2. For the USB function to be available, both HSE and PLL must be enabled, with USBCLK running at 48 MHz.
3. To have an ADC conversion time of 1 µs, APB2 must be at 14 MHz, 28 MHz or 56 MHz.

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)



Pinout & Configuration 탭에서

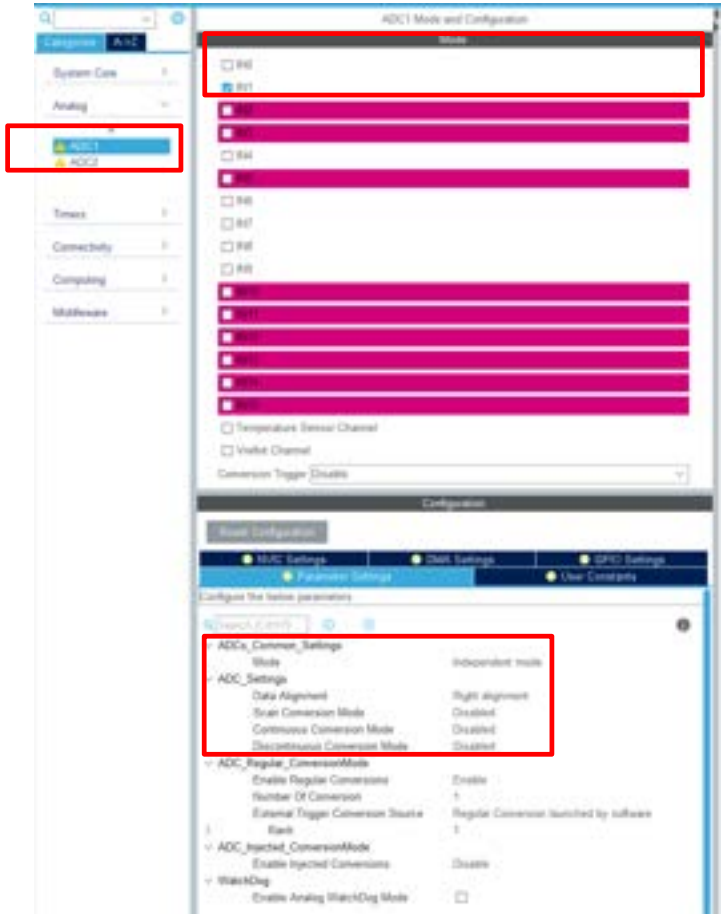
GPIO 출력핀 설정 : PC0~7에 LED 연결됨

PA1을 ADC1_IN1으로 설정함

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)



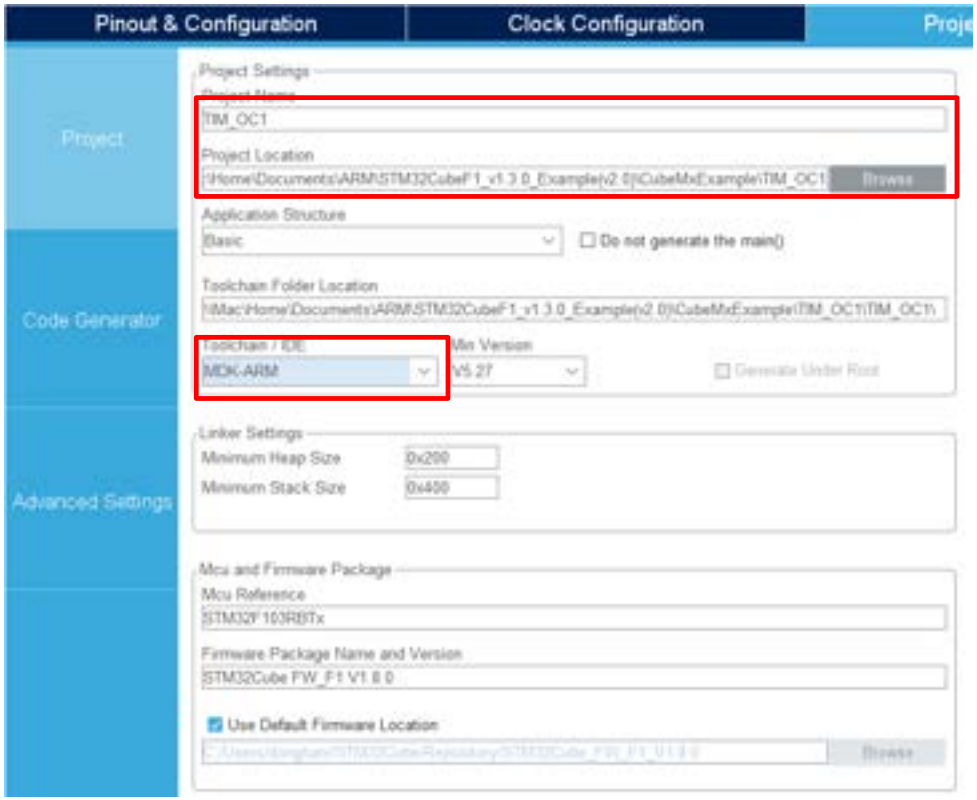
ADC1 중 IN1을 선택

Configuration에서 Parameter Settings 탭은 기본적으로 Scan, Continuous, Discontinuous Conversion Mode가 Disabled되어 있으므로 그대로 둠

KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

CubeMX로 예제 9.1 ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)



Project Manager 탭을 누르면, 왼쪽의 그림처럼 project 생성 단계가 되며, project name과 location을 정해줌.

Toolchain / IDE는 꼭 MDK-ARM을 선택

마지막으로 GENERATE CODE 탭을 누르면, 코드가 생성되면서 MDK uVision이 실행됨

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 예제 9.1 ADC 출력 값에 따른 LED의 On/Off 주기 변경(폴링 방식)

```
/* Private user code ----------------
/* USER CODE BEGIN 0 */
uint32_t  adcValue = 0;
int delay_time = 0;
/* USER CODE END 0 */
```

→ main() 함수 앞 쪽에 변수 선언

```
105  while (1)
106  {
107    /* USER CODE END WHILE */
108
109    /* USER CODE BEGIN 3 */
110    HAL_ADC_Start(&hadc1);
111
112    if(HAL_ADC_PollForConversion(&hadc1, 5) == HAL_OK)
113    {
114      adcValue = HAL_ADC_GetValue(&hadc1) * 6000;
115    }
116
117    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_SET);
118    for(delay_time = 0; delay_time < adcValue; delay_time++);
119    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_RESET);
120
121    HAL_Delay(1000);
```

→ ADC1의 ADC를 시작
→ ADC 변환이 끝날 때를 5ms 동안 기다리고 나서 변환이 끝나면, 값을 받아옴. 그리고 6000을 곱해서 adcValue에 저장

→ 최종 코드(교과서의 코드와 CubeMX의 코드)를 비교해보세요

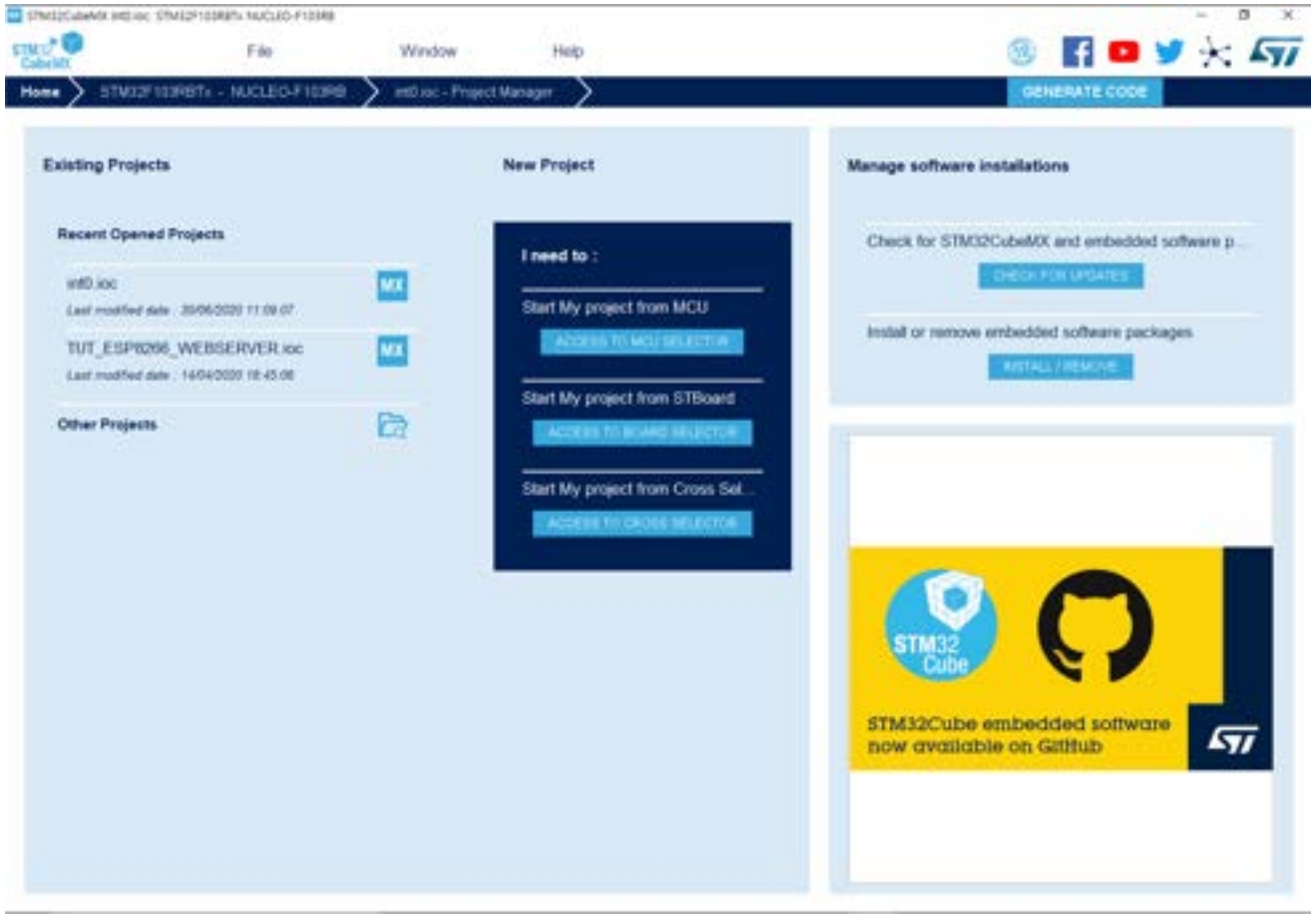
KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 예제 9.2 구현하기

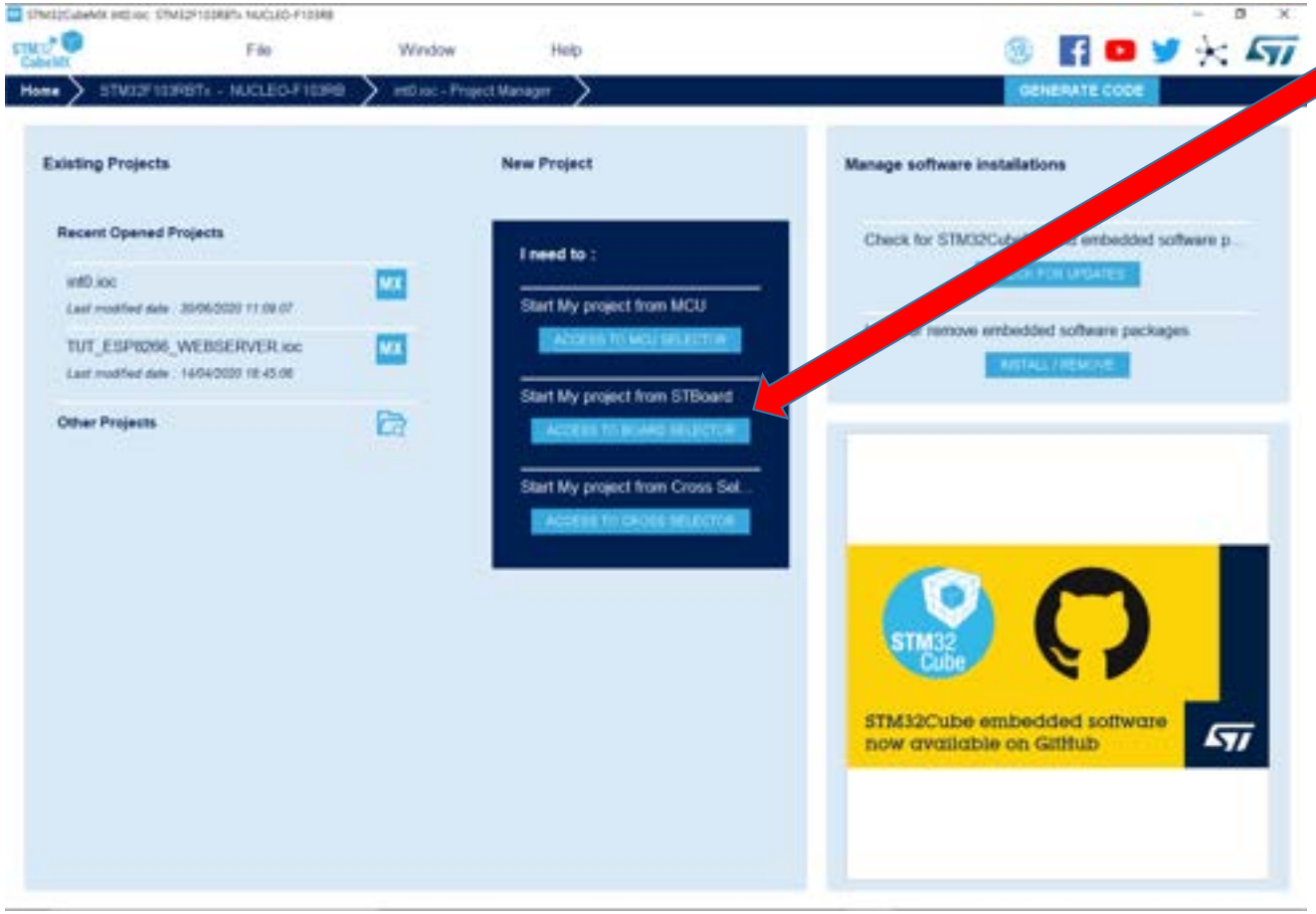
CubeMX 실행

초기화면



9.4 ADC CubeMX과제

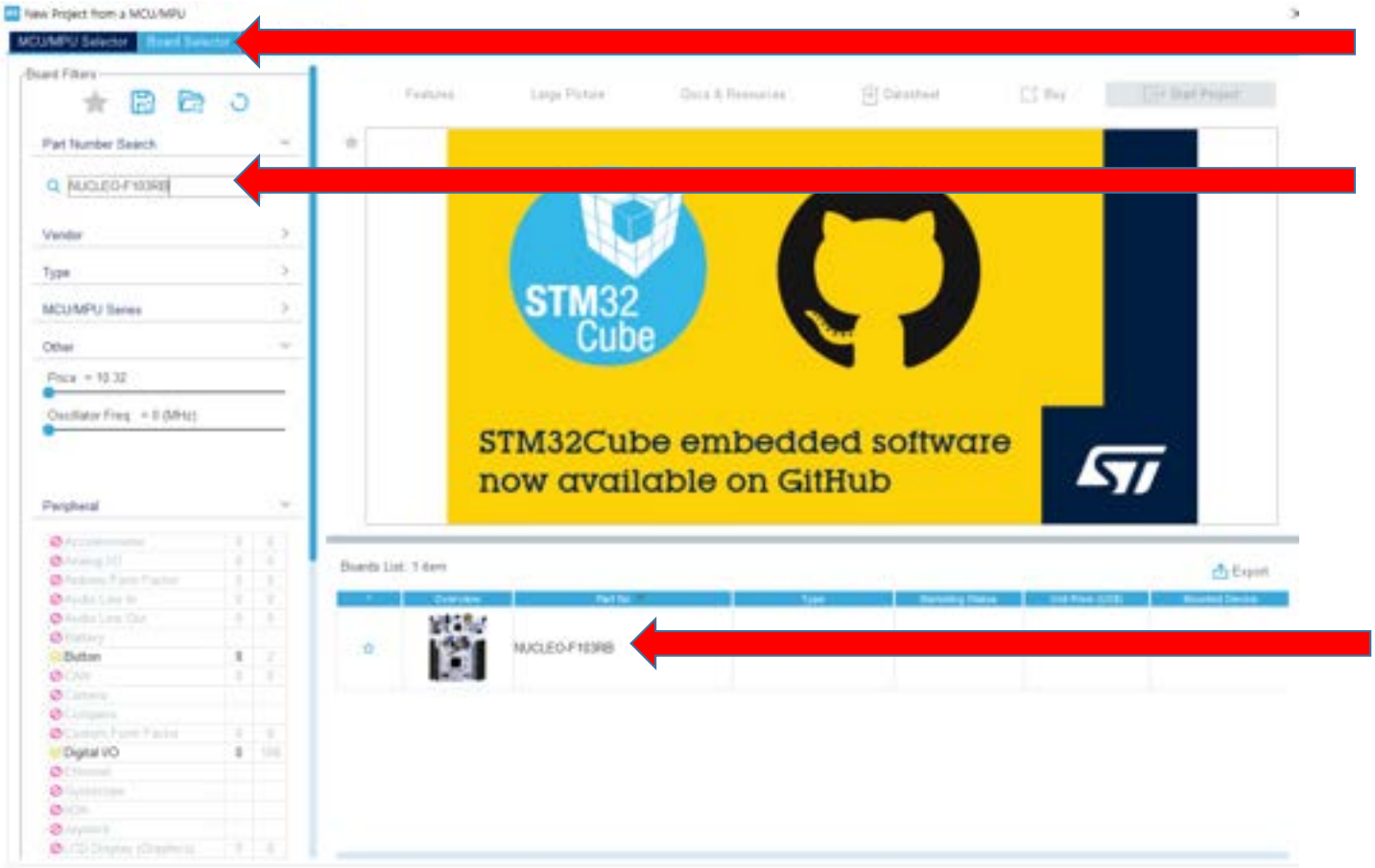
CubeMX로 예제 9.2 구현하기



보드 선택

9.4 ADC CubeMX과제

CubeMX로 예제 9.2 구현하기



보드 선택

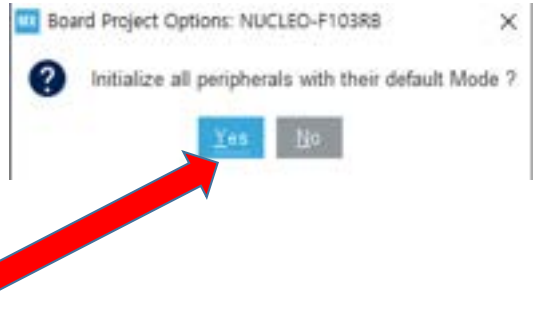
NUCLEO-F103RB 선택

NUCLEO-F103RB 더블클릭

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

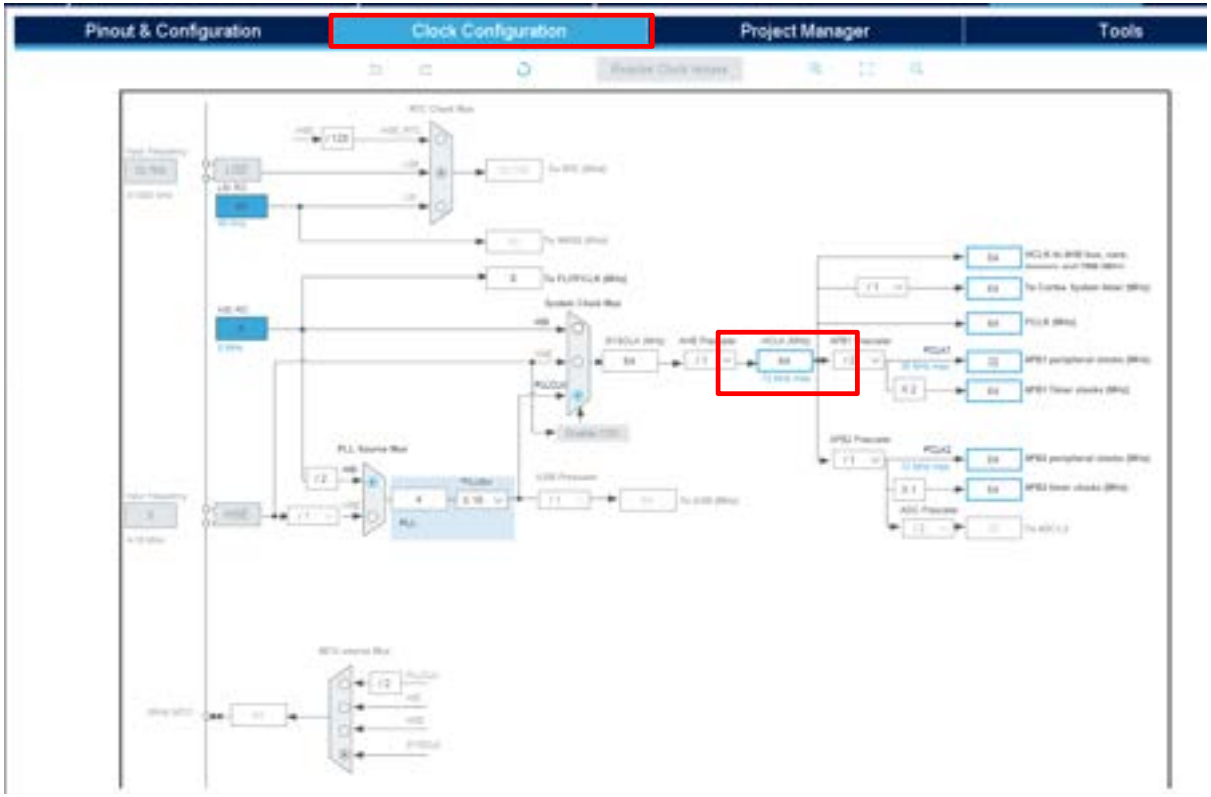
CubeMX로 예제 9.2 구현하기



Yes 선택하면 오른쪽과 같이 칩이 보임

9.4 ADC CubeMX과제

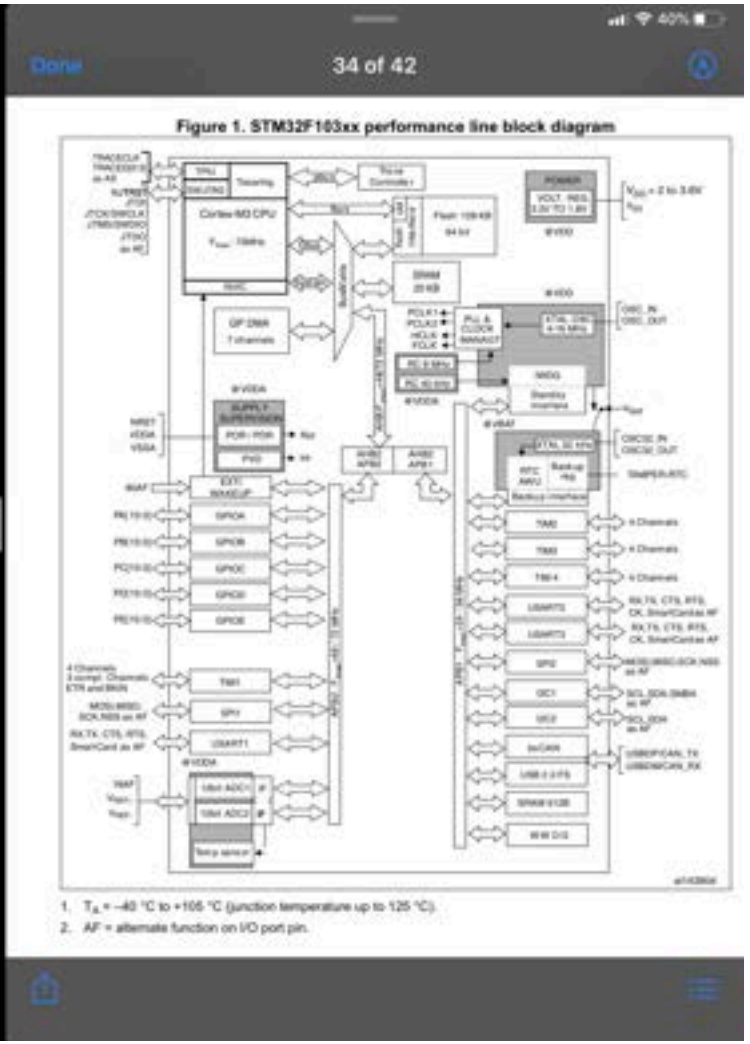
CubeMX로 ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

9.4 ADC CubeMX과제

```
 *          PLLMUL                  = 16
 *          Flash Latency(WS)       = 2
 */
// ------------------------------------------------------------------------- //

void  SystemClock_Config(void)
{
          RCC_ClkInitTypeDef clkinitstruct = {0};
          RCC_OscInitTypeDef oscinitstruct = {0};

          /* Configure PLL ------------------------------------------------------*/
          /* PLL configuration: PLLCLK = (HSI / 2) * PLLMUL = (8 / 2) * 16 = 64 MHz */
          /* PREDIV1 configuration: PREDIV1CLK = PLLCLK / HSEPredivValue = 64 / 1 = 64
MHz */
          /* Enable HSI and activate PLL with HSI_DIV2 as source */
          oscinitstruct.OscillatorType  = RCC_OSCILLATORTYPE_HSI;
          oscinitstruct.HSEState        = RCC_HSE_OFF;
          oscinitstruct.LSEState        = RCC_LSE_OFF;
          oscinitstruct.HSIState        = RCC_HSI_ON;
          oscinitstruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
          oscinitstruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
          oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
          oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_HSI_DIV2;
          oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;

          if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)  {
                    /* Initialization Error */
                    while(1);
          }

          /* Select PLL as system clock source and configure the HCLK, PCLK1 and
PCLK2
              clocks dividers */
          clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
          clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
          clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
          clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
          clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;

          if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)  {
                    /* Initialization Error */
                    while(1);
          }
}

// ------------------------------------------------------------------------- //
//  --  <18>  Clock 설정시 에러가 발생하면 처리해주는 함수
/**
```



KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

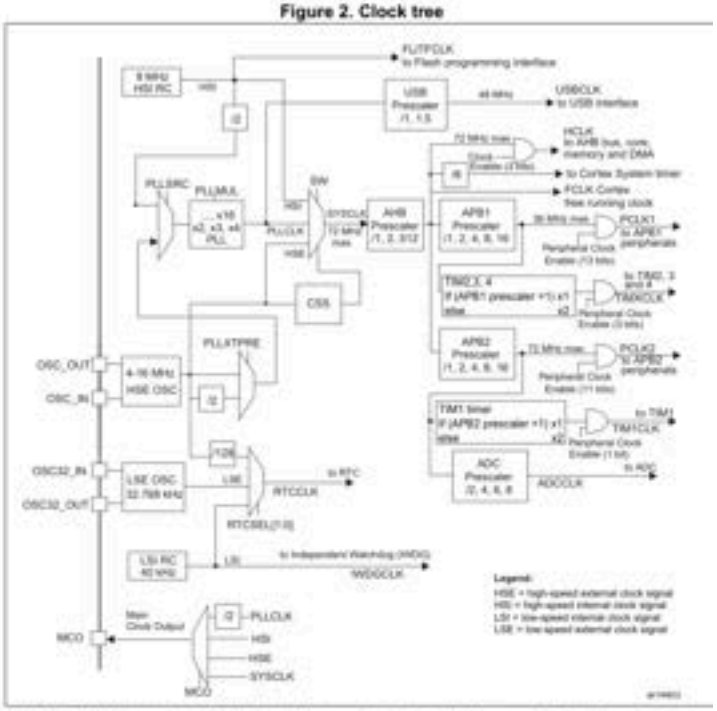
8:41 PM Mon Oct 28

Done 2 of 4

```
void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef clkinitstruct = {0};
  RCC_OscInitTypeDef oscinitstruct = {0};

  /* Configure PLL ------------------------------------------------------*/
  /* PLL configuration: PLLCLK = (HSI / 2) * PLLMUL = (8 / 2) * 16 = 64 MHz */
  /* PREDIV1 configuration: PREDIV1CLK = PLLCLK / HSEPredivValue = 64 / 1 = 64
MHz */
  /* Enable HSI and activate PLL with HSi_DIV2 as source */
  oscinitstruct.OscillatorType  = RCC_OSCILLATORTYPE_HSI;
  oscinitstruct.HSEState        = RCC_HSE_OFF;
  oscinitstruct.LSEState        = RCC_LSE_OFF;
  oscinitstruct.HSIState        = RCC_HSI_ON;
  oscinitstruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  oscinitstruct.HSEPredivValue    = RCC_HSE_PREDIV_DIV1;
  oscinitstruct.PLL.PLLState    = RCC_PLL_ON;
  oscinitstruct.PLL.PLLSource   = RCC_PLLSOURCE_HSI_DIV2;
  oscinitstruct.PLL.PLLMUL      = RCC_PLL_MUL16;

  if (HAL_RCC_OscConfig(&oscinitstruct)!= HAL_OK)  {
      /* Initialization Error */
      while(1);
  }

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and
PCLK2
      clocks dividers */
  clkinitstruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK
| RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  clkinitstruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  clkinitstruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  clkinitstruct.APB2CLKDivider = RCC_HCLK_DIV1;
  clkinitstruct.APB1CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&clkinitstruct, FLASH_LATENCY_2)!= HAL_OK)  {
      /* Initialization Error */
      while(1);
  }
}
```

Done 34 of 42

Figure 2. Clock tree

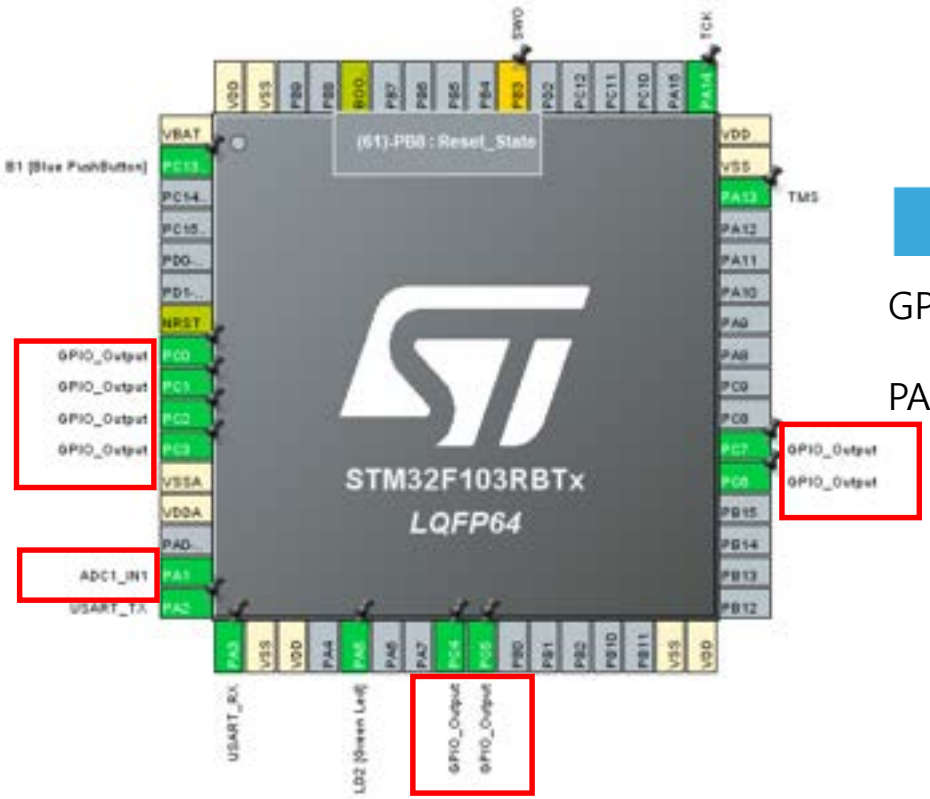


1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
2. For the USB function to be available, both HSE and PLL must be enabled, with USBCLK running at 48 MHz.
3. To have an ADC conversion time of 1 µs, APB2 must be at 14 MHz, 28 MHz or 56 MHz.

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)



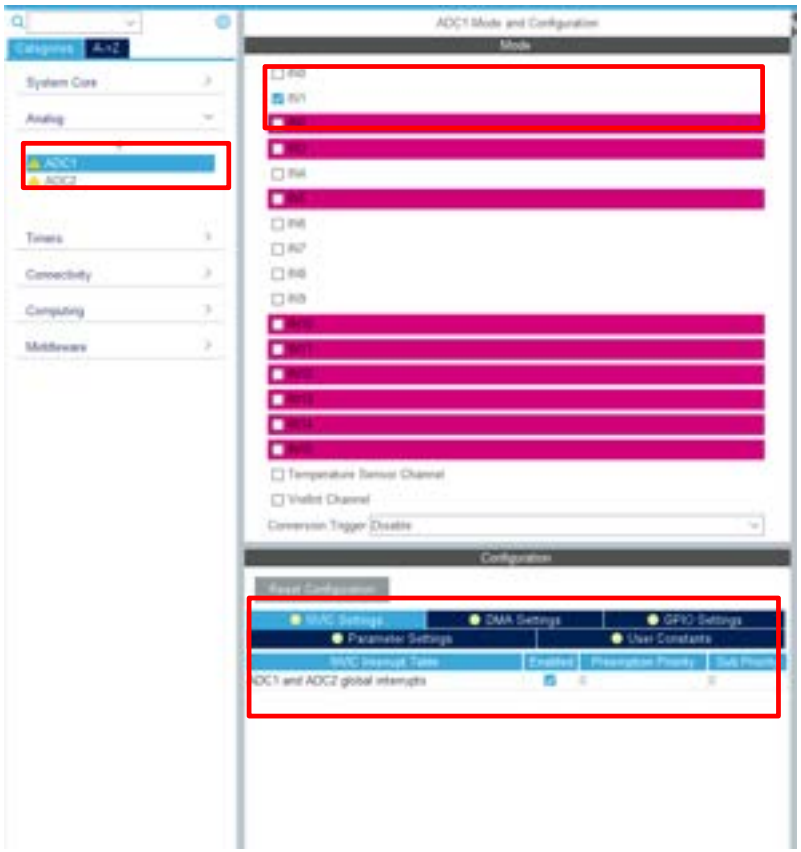
Pinout & Configuration 탭에서

GPIO 출력핀 설정 : PC0~7에 LED 연결됨

PA1을 ADC1_IN1으로 설정함

9.4 ADC CubeMX과제

CubeMX로 ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)



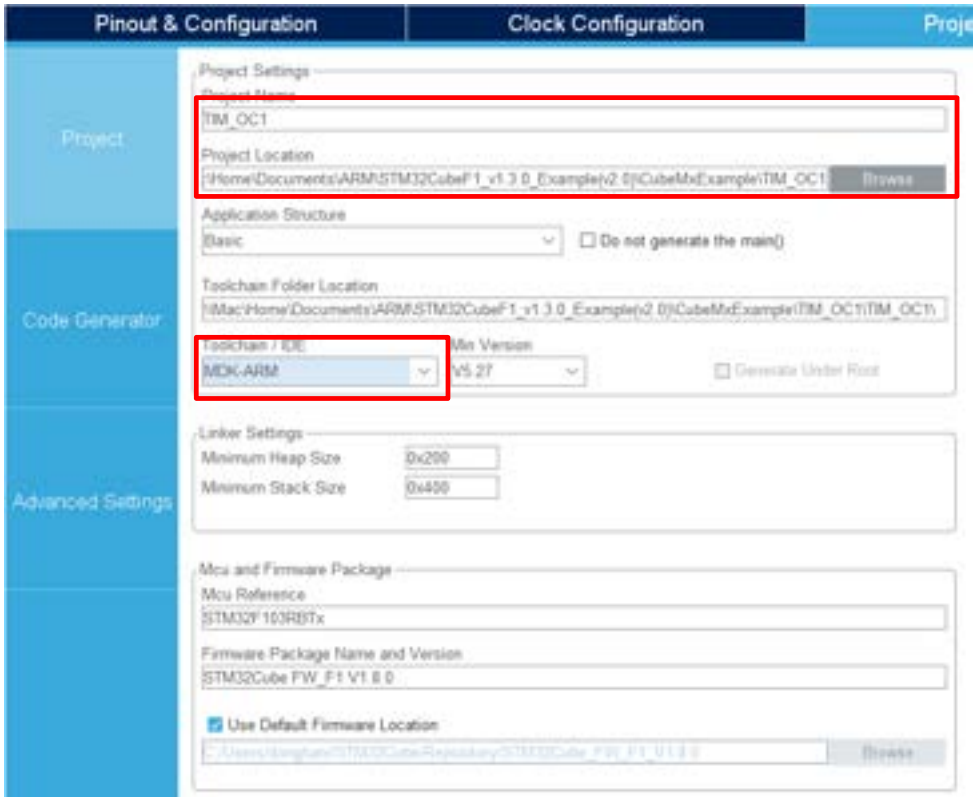
ADC1 중 IN1을 선택

Configuration에서 Parameter Settings 탭은 기본적으로 Scan, Continuous, Discontinuous Conversion Mode가 Disabled되어 있으므로 그대로 둠
NVIC 탭에서 ADC1 and ADC2 global interrupts를 Enabled 함

KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

CubeMX로 ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)



Project Manager 탭을 누르면, 왼쪽의 그림처럼 project 생성 단계가 되며, project name과 location을 정해줌.

Toolchain / IDE는 꼭 MDK-ARM을 선택

마지막으로 GENERATE CODE 탭을 누르면, 코드가 생성되면서 MDK uVision이 실행됨

KYUNG HEE UNIVERSITY

9.4 ADC CubeMX과제

CubeMX로 ADC 예제2 : ADC 출력 값에 따른 LED의 On/Off 주기 변경(인터럽트 방식)

```
60  void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
61  {
62      adcValue = HAL_ADC_GetValue(&hadc1) * 6000;
63      HAL_ADC_Start_IT(&hadc1);
64  }
```

→ main() 함수 앞 쪽에 ADC 변환 완료 콜백함수 선언하고 인터럽트 걸리면 해야 할 일 정의

```
117   while (1)
118   {
119     /* USER CODE END WHILE */
120
121     /* USER CODE BEGIN 3 */
122
123     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 |  GPIO_PIN_1 |  GPIO_PIN_2 |  GPIO_PIN_3 |  GPIO_PIN_4 |
124                       GPIO_PIN_5 |  GPIO_PIN_6 |  GPIO_PIN_7, GPIO_PIN_SET);
125     for(delay_time = 0; delay_time < adcValue; delay_time++);
126     HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 |  GPIO_PIN_1 |  GPIO_PIN_2 |  GPIO_PIN_3 |  GPIO_PIN_4 |
127                       GPIO_PIN_5 |  GPIO_PIN_6 |  GPIO_PIN_7, GPIO_PIN_RESET);
128
129
130     HAL_Delay(1000);
131
132   }
```

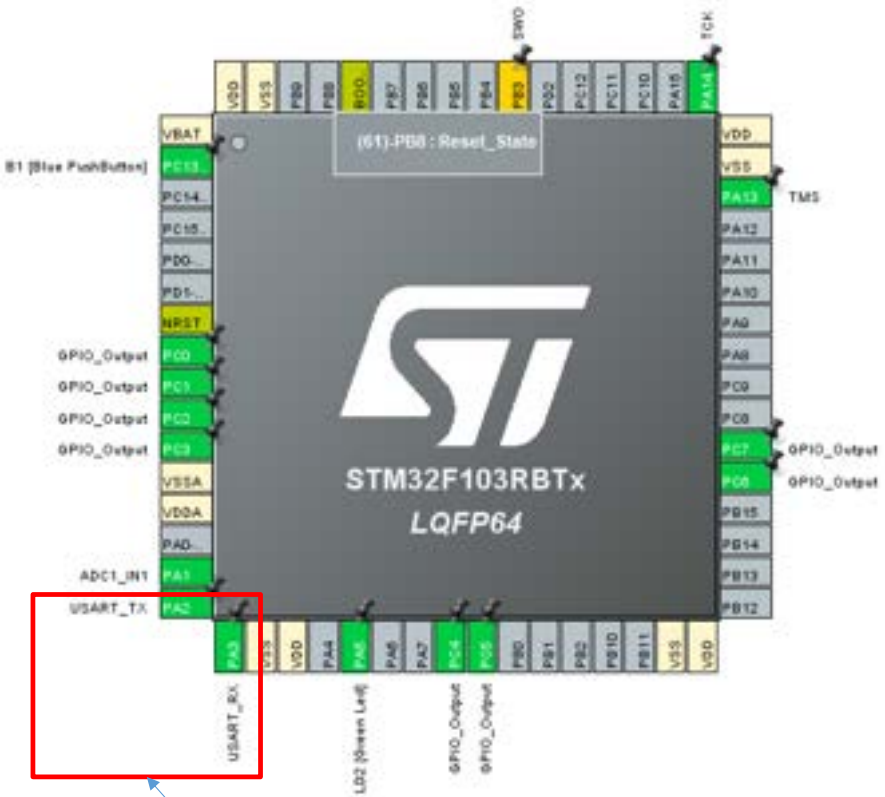
→ 예제 9.1과 비교해보세요.

→ 최종 코드(교과서의 코드와 CubeMX의 코드)를 비교해보세요

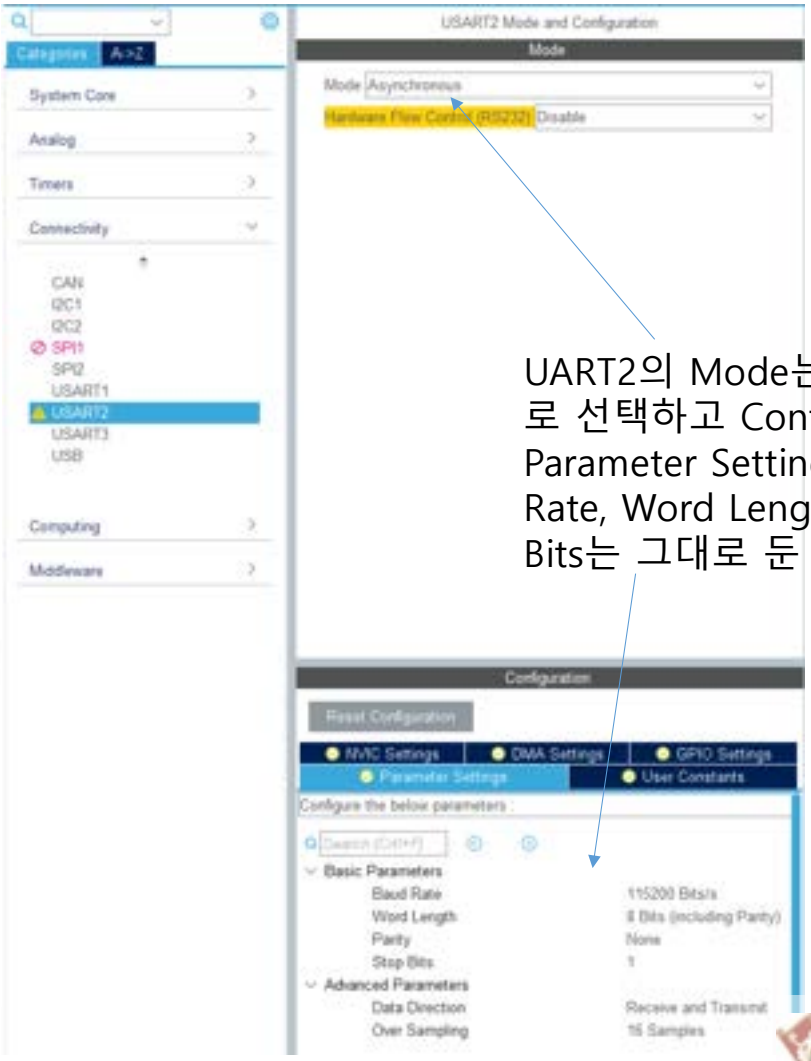
KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

중요: printf 함수를 사용해서 debugging 하는 방법



UART2를 사용함



UART2의 Mode는 Asynchronous로 선택하고 Configuration의 Parameter Settings 탭의 Baud Rate, Word Length, Parity, Stop Bits는 그대로 둔 상태로 둠

추가. ADC CubeMX과제

중요: printf 함수를 사용해서 debugging 하는 방법

GENERATE CODE 버튼을 누르면 기존 코딩에 추가됨

```
24  /* Private includes ----------------------------------------------------------*/
25  /* USER CODE BEGIN Includes */
26  #include <stdio.h>
27  /* USER CODE END Includes */
```

→ #Include 부분에 stdio.h 추가

```
59  /* USER CODE BEGIN PFP */
60  int fputc(int ch, FILE *f)
61  {
62    HAL_UART_Transmit(&huart2, (uint8_t *)&ch, 1, 0xFFFF);
63
64    return ch;
65  }
```

→main() 함수 앞에 fputc 추가. UART2를 선택했으므로 &huart2로 한 것임

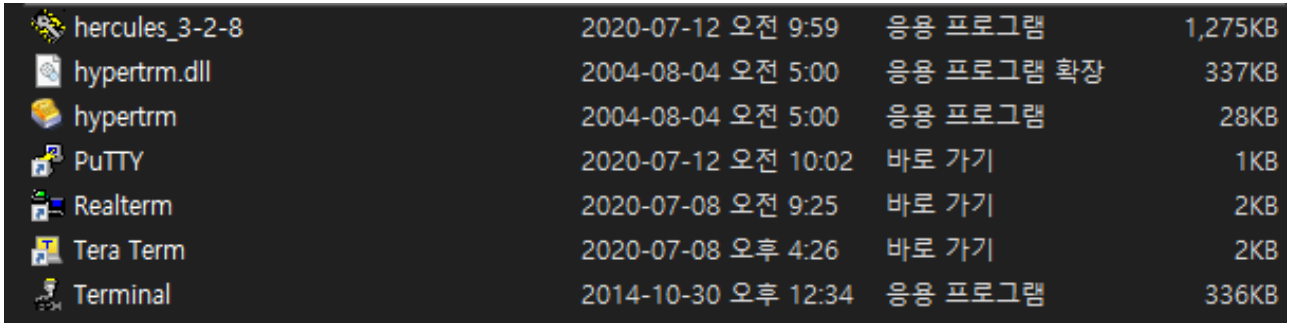
```
129     printf("adcValue = %d\r\n", adcValue);
```

→ while(1) 안에 보고 싶은 변수 값을 printf 하라고 하면 UART 통신을 통해서 컴퓨터로 전송됨

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

중요: printf 함수를 사용해서 debugging 하는 방법

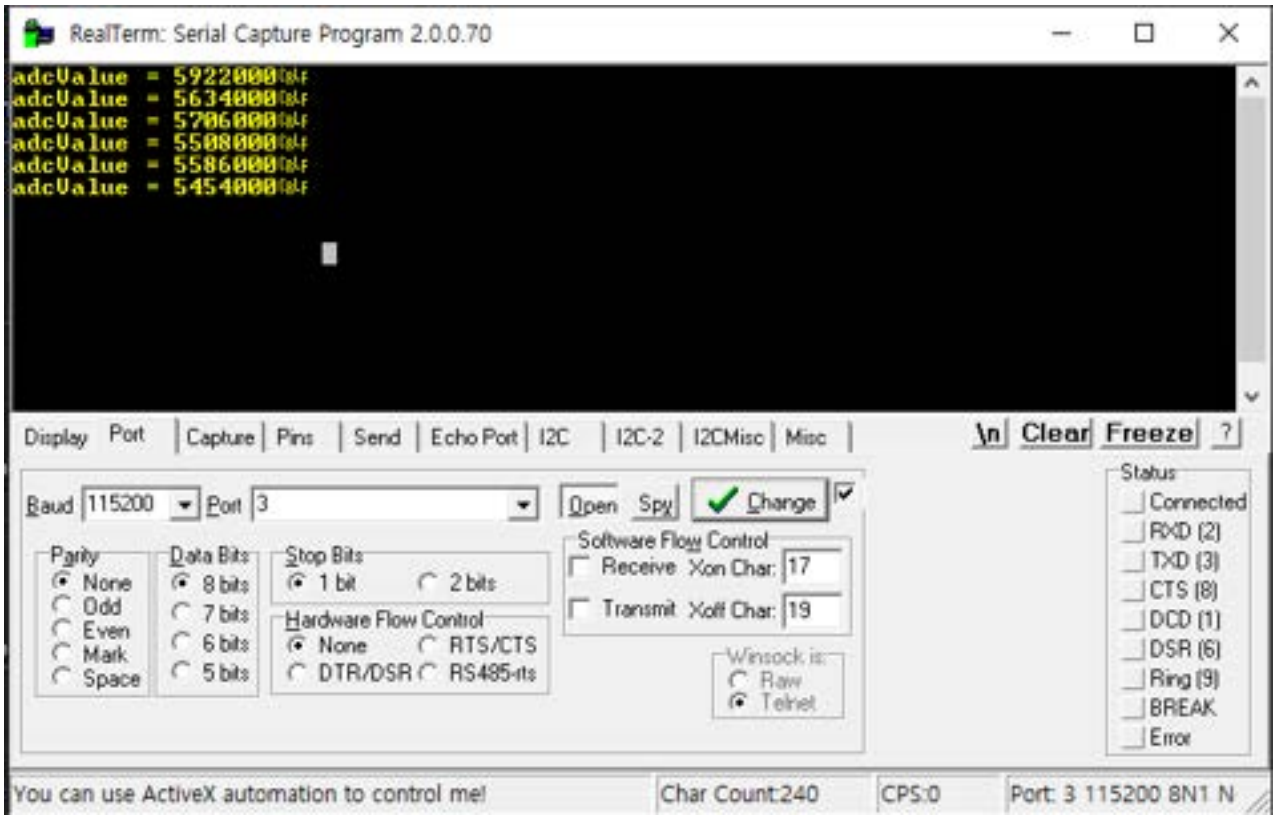


hercules_3-2-8	2020-07-12 오전 9:59	응용 프로그램	1,275KB
hypertrm.dll	2004-08-04 오전 5:00	응용 프로그램 확장	337KB
hypertrm	2004-08-04 오전 5:00	응용 프로그램	28KB
PuTTY	2020-07-12 오전 10:02	바로 가기	1KB
Realterm	2020-07-08 오전 9:25	바로 가기	2KB
Tera Term	2020-07-08 오후 4:26	바로 가기	2KB
Terminal	2014-10-30 오후 12:34	응용 프로그램	336KB

→ 인터넷에서 아무 시리얼 모니터 프로그램 다운로드함. 바이러스 주의
→ Baud, Port, Parity, Stop Bits 등을 CubeMX 설정과 동일하게 해줌

추가. ADC CubeMX과제

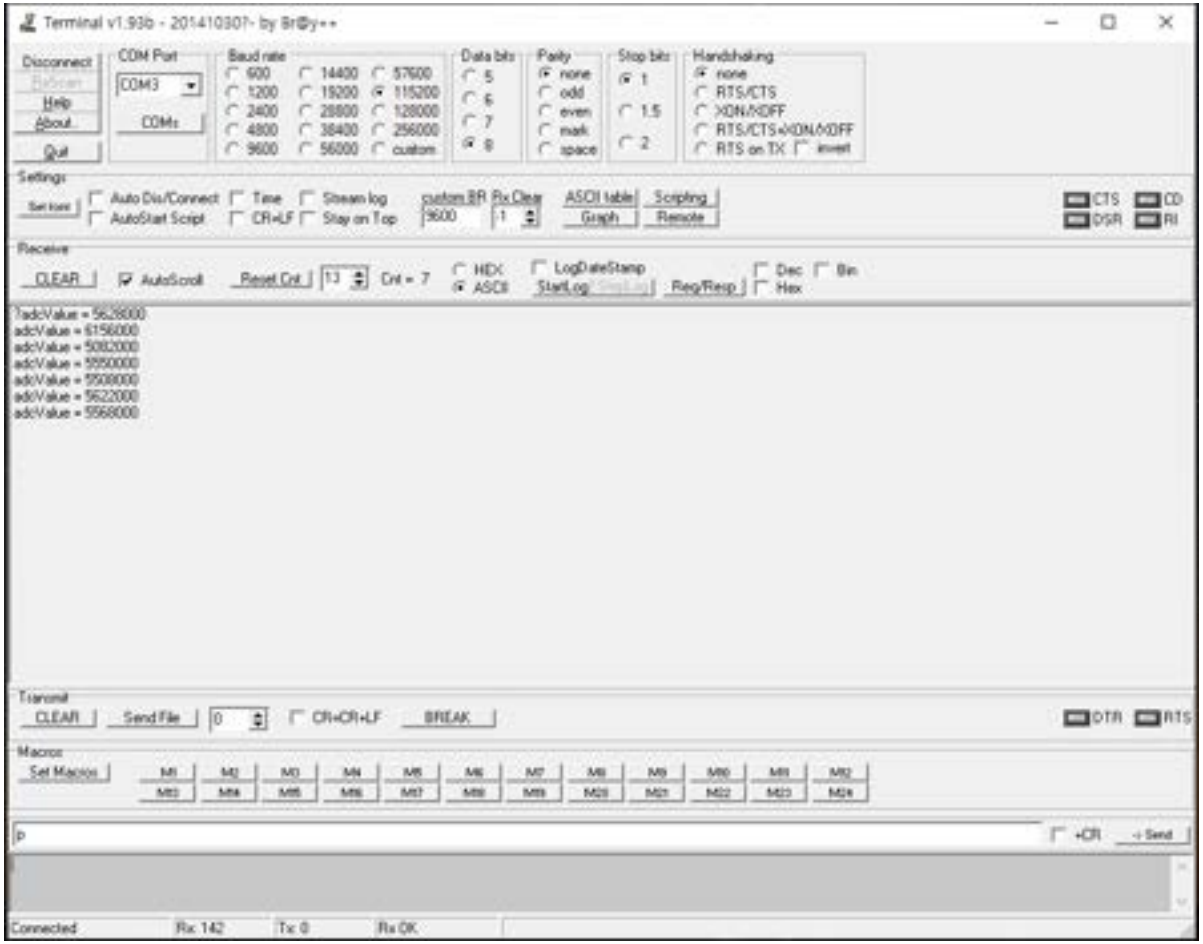
중요: printf 함수를 사용해서 debugging 하는 방법



KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

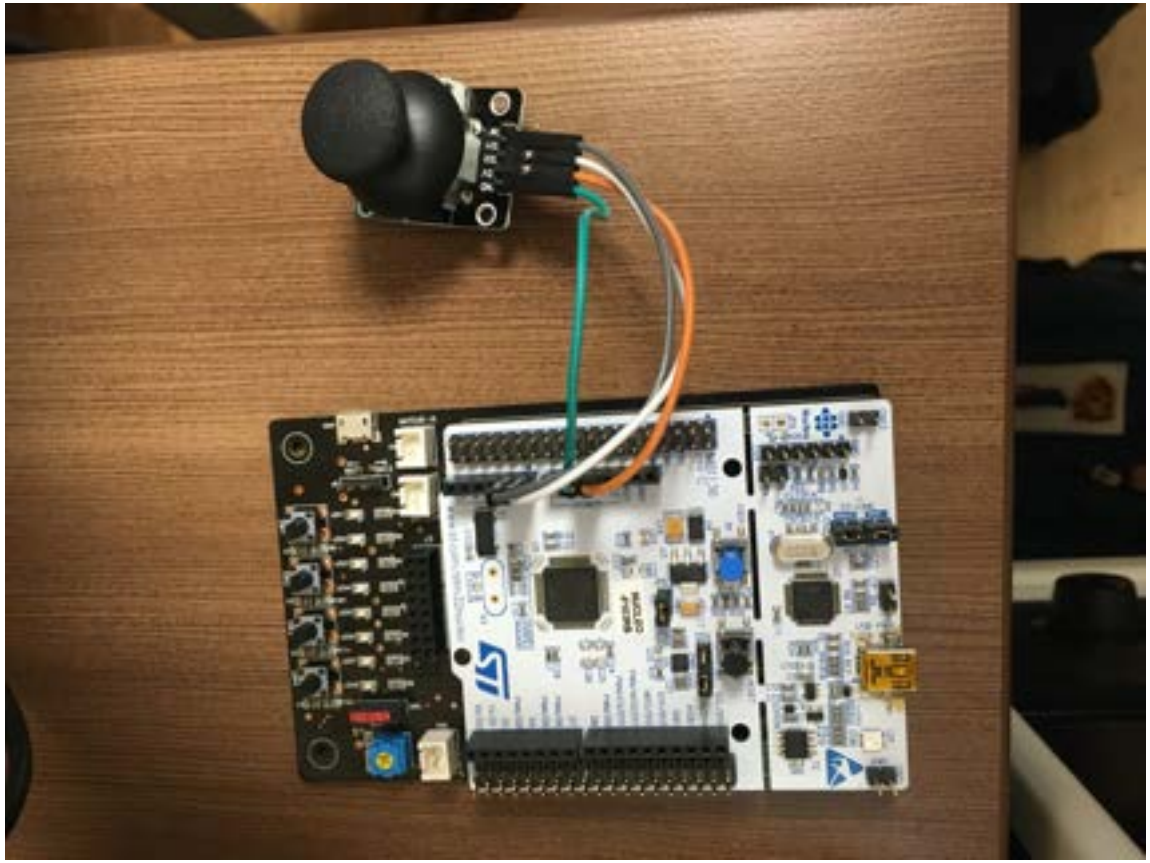
중요: printf 함수를 사용해서 debugging 하는 방법



KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

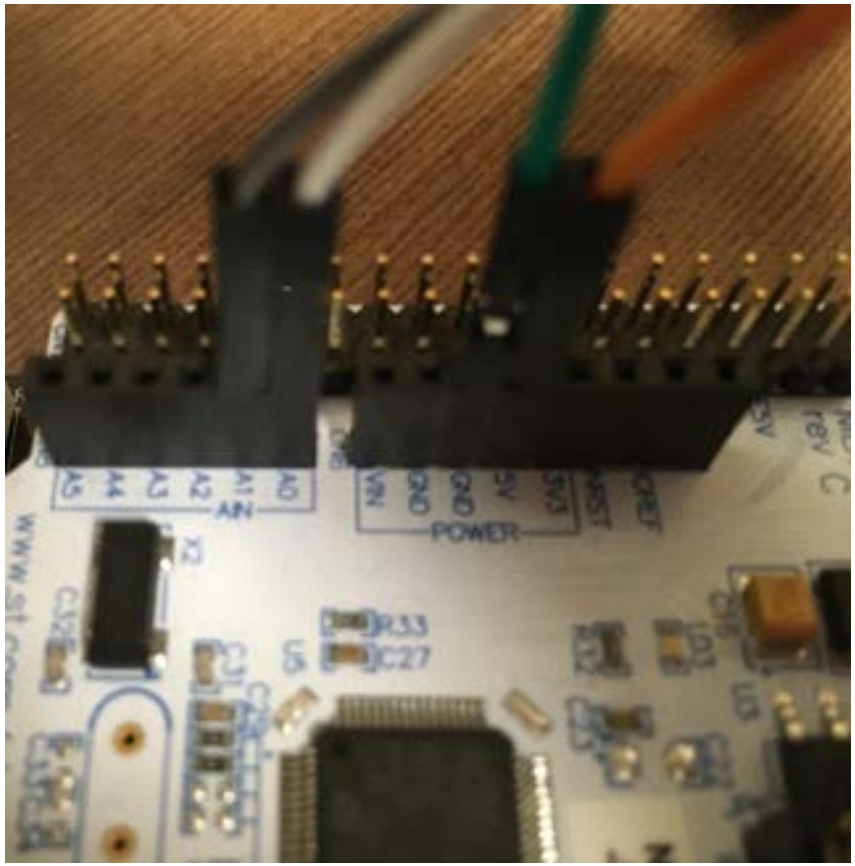


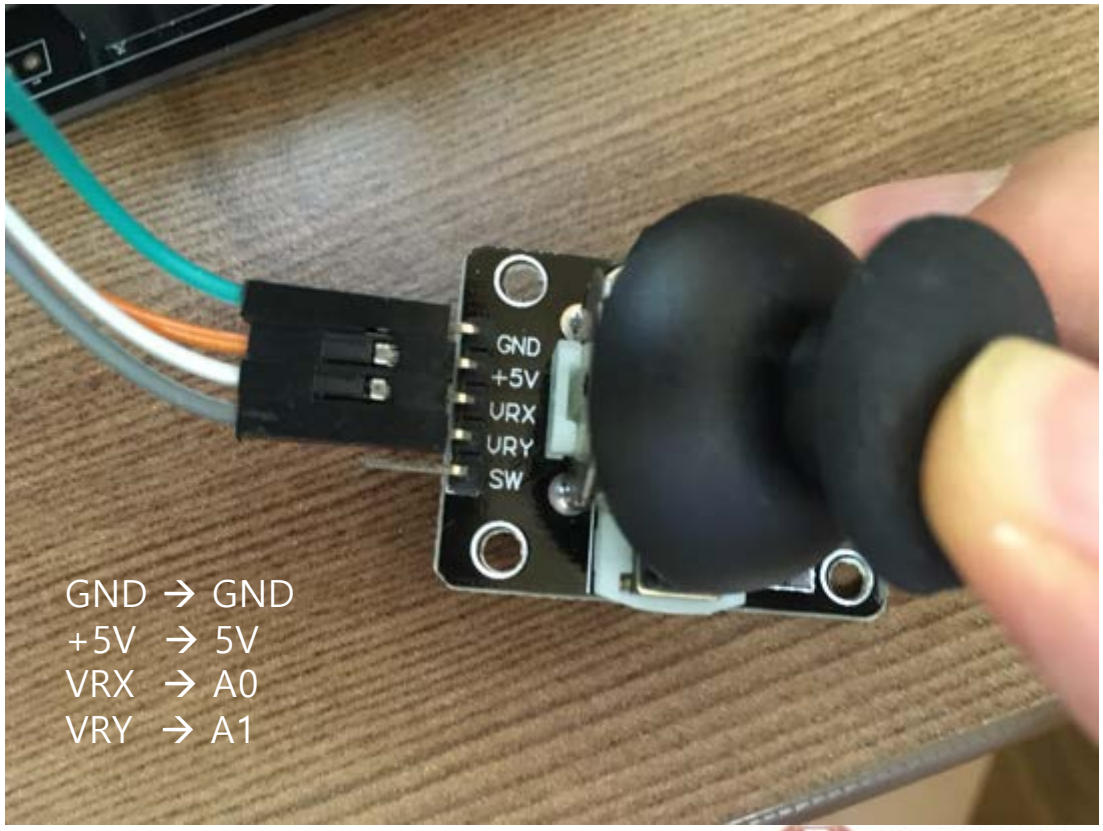
- XBOX나 PS에서 사용하는 아날라고 조이스틱임

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

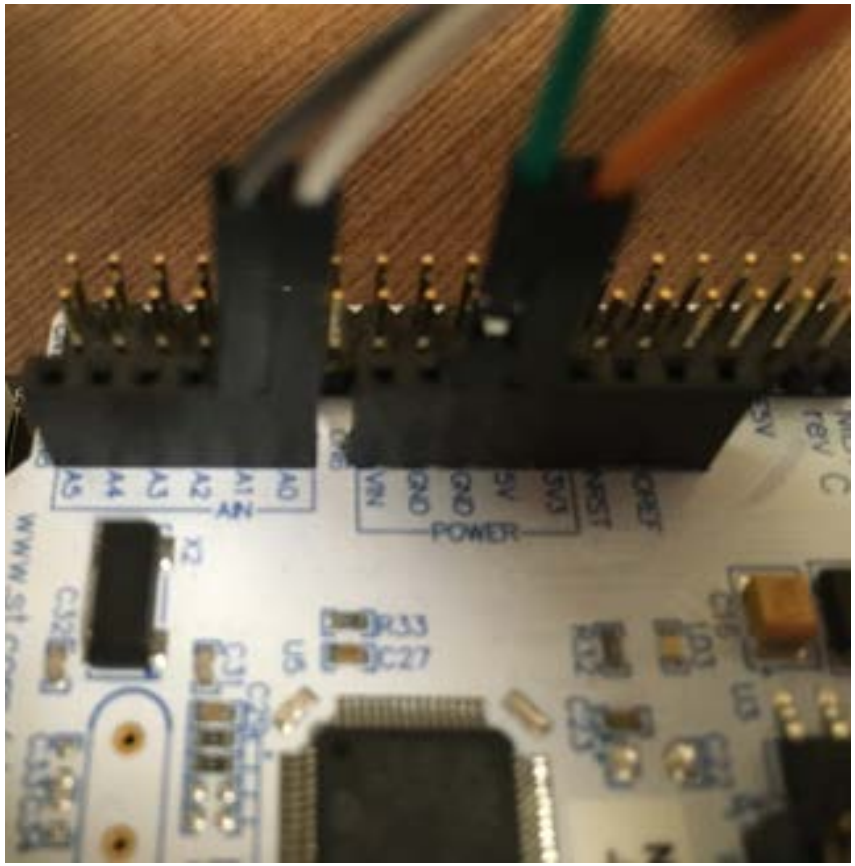


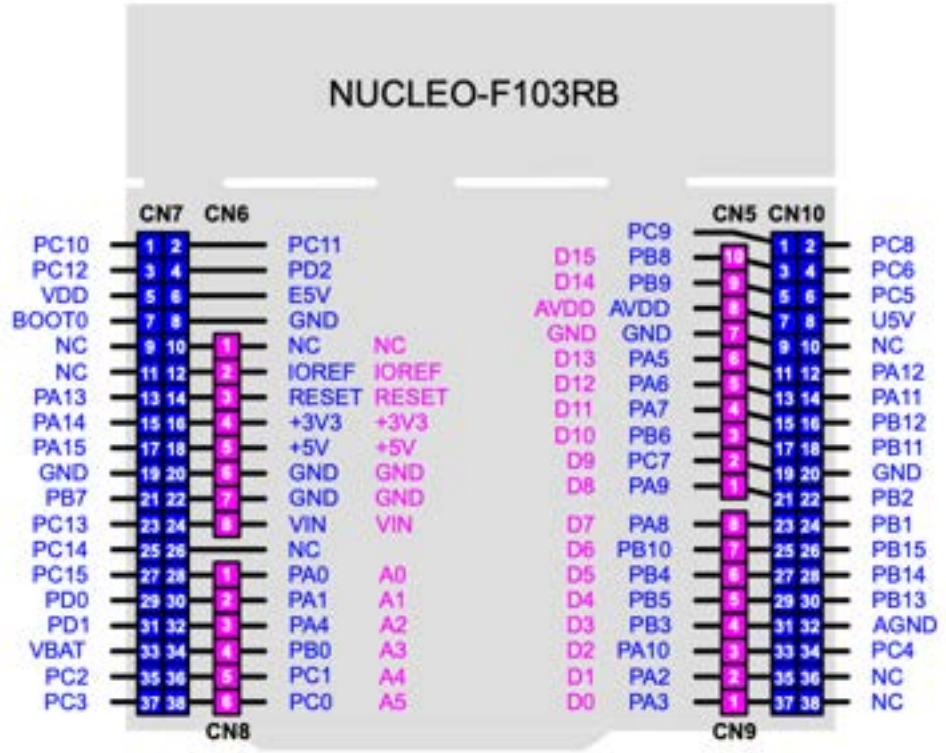


KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제



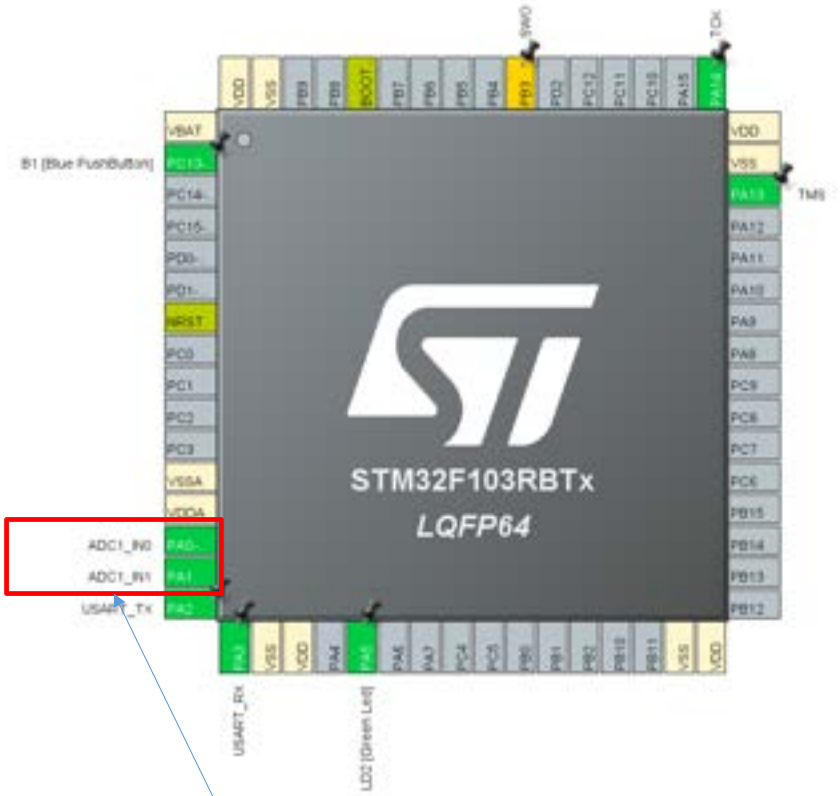


조이스틱→커넥터 → ARM 핀
GND → GND → GND
+5V → +5V → +5V
VRX → A0 → PA0
VRY → A1 → PA1

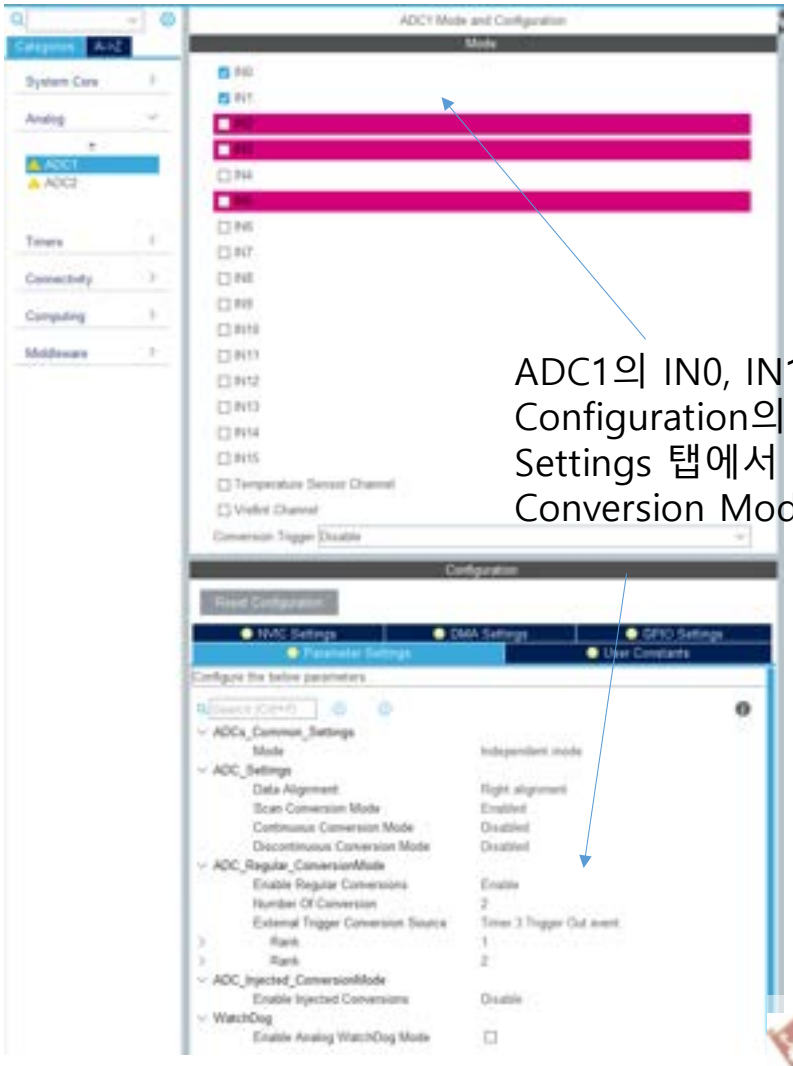
KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제



ADC1의 IN0, IN1을 사용함



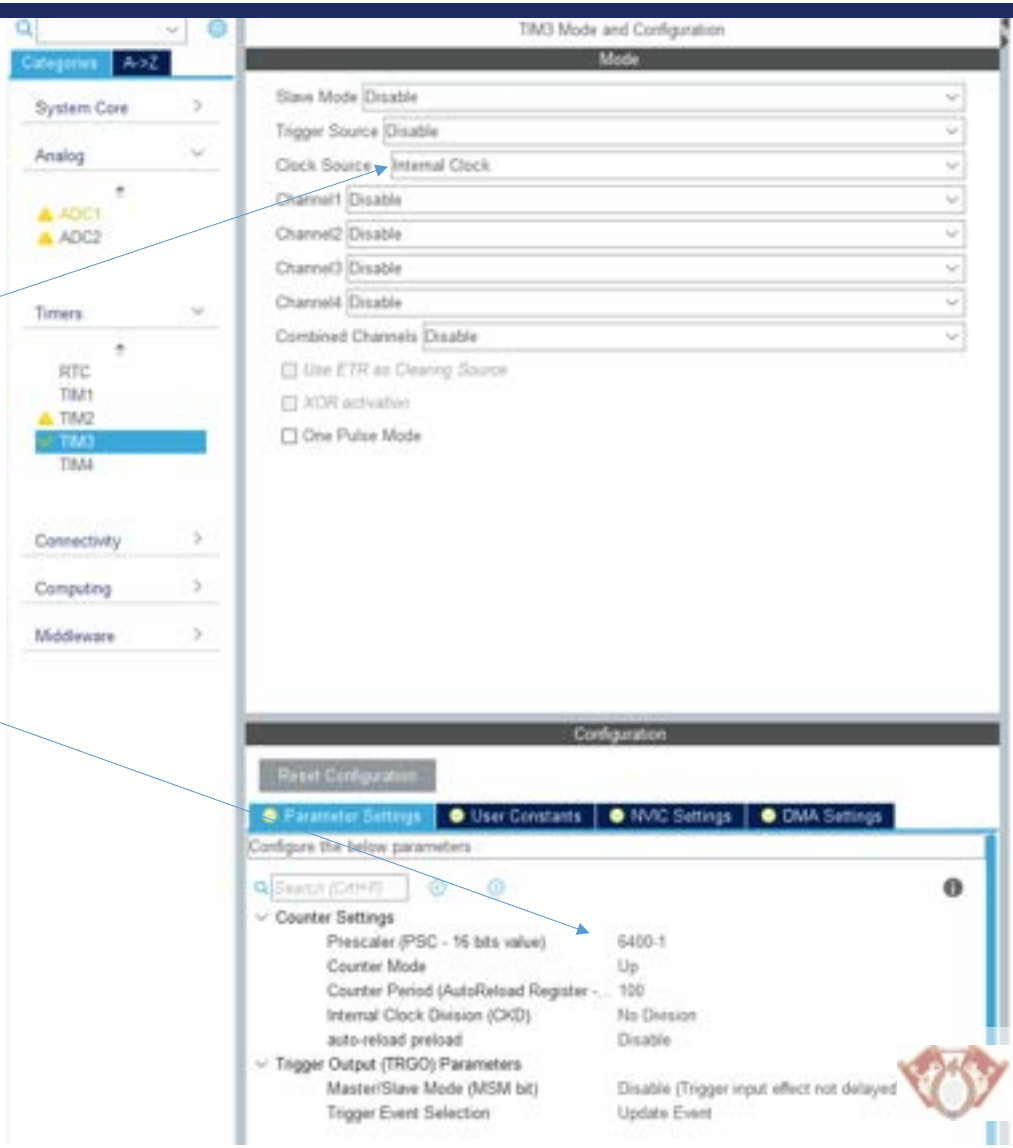
ADC1의 IN0, IN1을 선택하고 Configuration의 Parameter Settings 탭에서 Scan Conversion Mode를 Enabled

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

TIM3를 Internal Clock을 source 로 해서 설정함. 속도는 64MHz/6400/100 = 100Hz 로 설정함

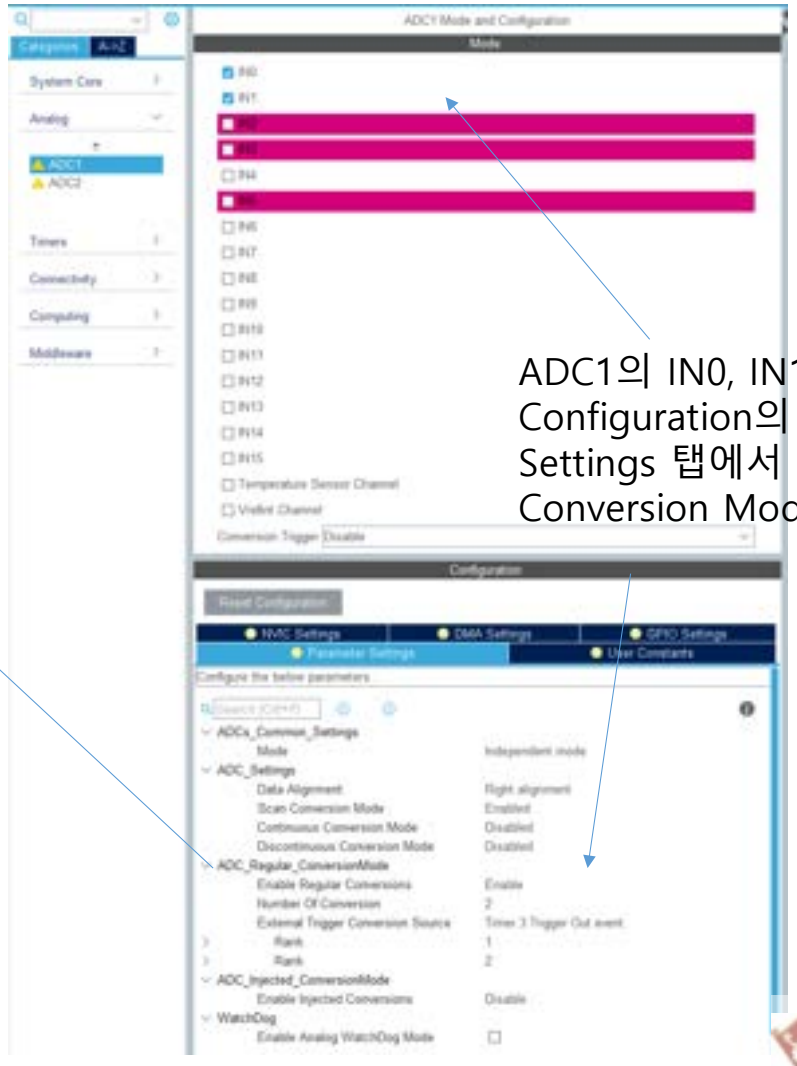


추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable
Number Of Conversion	2
External Trigger Conversion Source	Timer 3 Trigger Out event
Rank	1
Channel	Channel 0
Sampling Time	13.5 Cycles
Rank	2
Channel	Channel 1
Sampling Time	13.5 Cycles

- Number of Conversion을 2로 하면 Rank가 2개가 생길 것임. Rank 1을 Channel 0로, Rank2를 Channel 1로 설정함
- 그리고, External Trigger Conversion Source를 앞장에서 설정한 Timer 3 Trigger Out event로 설정함
- Sampling Time은 적당히 큰 값 선택



ADC1의 IN0, IN1을 선택하고 Configuration의 Parameter Settings 탭에서 Scan Conversion Mode를 Enabled

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

GENERATE CODE 버튼을 누르면 기존 코딩에 추가됨

```
82  /* Private user code ---------
83  /* USER CODE BEGIN 0 */
84  uint32_t  adcVal[2];
85  /* USER CODE END 0 */
```

→ IN0, IN1에서 값이 들어오므로, 배열 adcVal[2]를 선언

```
119   /* USER CODE BEGIN 2 */
120   //start the timer 3
121   HAL_TIM_Base_Start(&htim3);
122
123   //start ADC DMA
124   HAL_ADC_Start_DMA(&hadc1, adcVal, 2);
125   /* USER CODE END 2 */
126
127   /* Infinite loop */
128   /* USER CODE BEGIN WHILE */
129   while (1)
130   {
131     /* USER CODE END WHILE */
132
133     /* USER CODE BEGIN 3 */
134     printf("X axis = %d, Y axis = %d\n\r", adcVal[0], adcVal[1]);
135     HAL_Delay(500);
136   }
137   /* USER CODE END 3 */
```

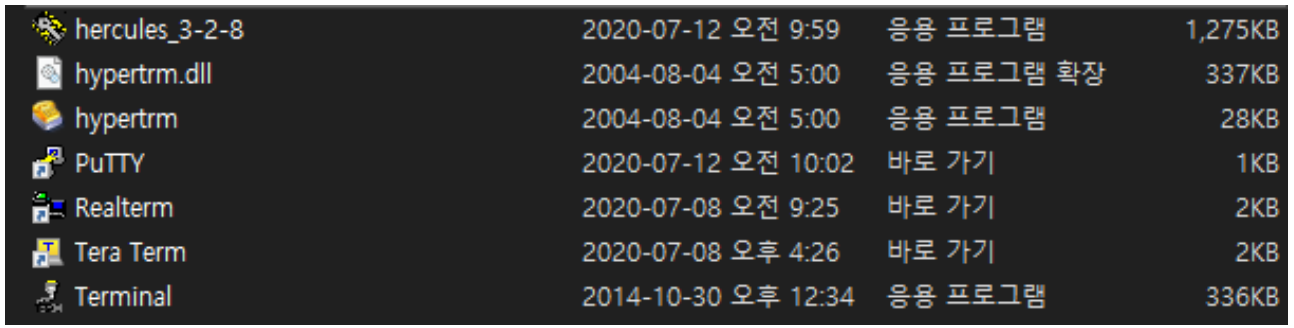
→ TIM3을 시작하고, ADC_DMA도 시작함
→ HAL_ADC_Start_DMA의 인자들 이해해야 함

→ 앞의 예제처럼 while(1) 안에 보고 싶은 변수 값을 printf 하라고 하면 UART 통신을 통해서 컴퓨터로 전송됨

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

Multi-channel ADC-DMA 예제

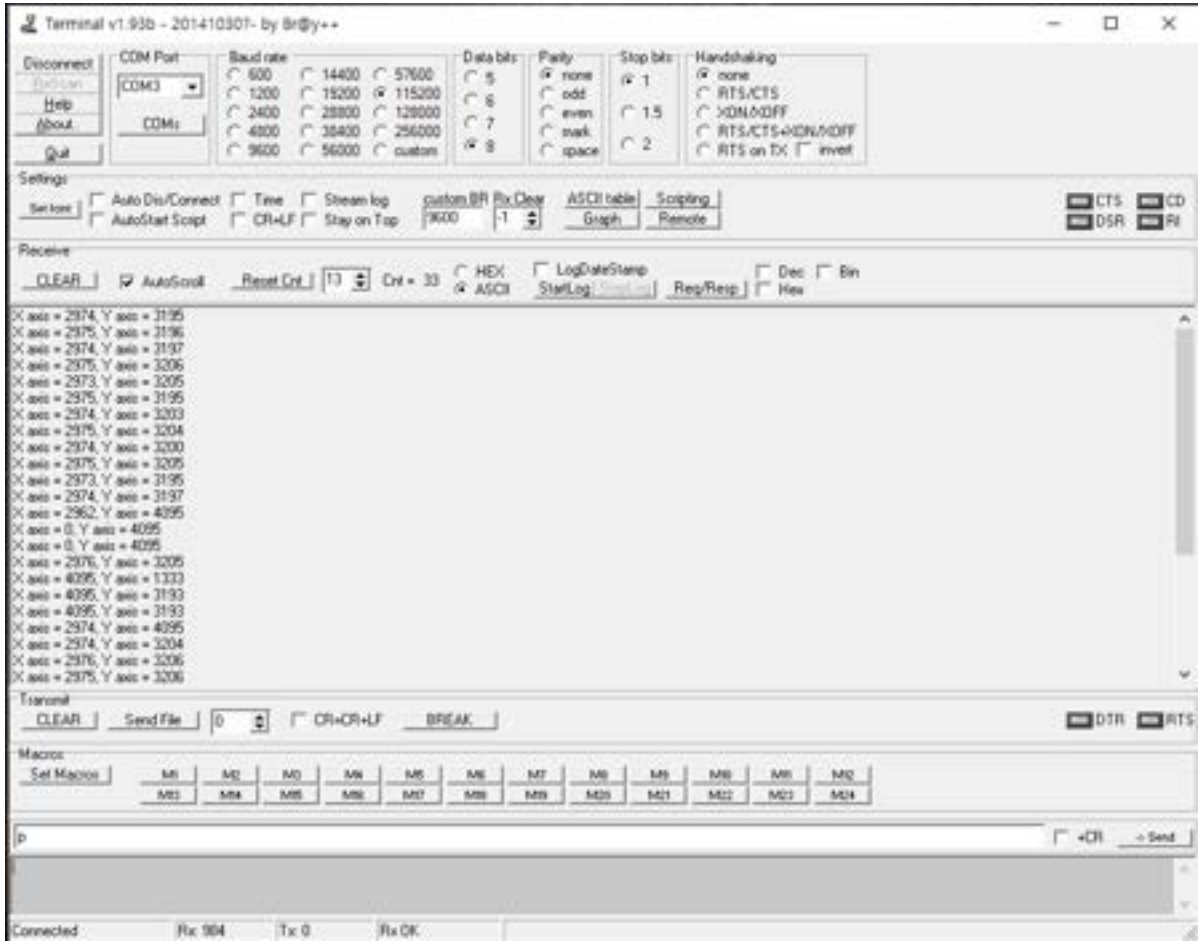


→ 인터넷에서 아무 시리얼 모니터 프로그램 다운로드함. 바이러스 주의
→ Baud, Port, Parity, Stop Bits 등을 CubeMX 설정과 동일하게 해줌

KYUNG HEE UNIVERSITY

추가. ADC CubeMX과제

중요: printf 함수를 사용해서 debugging 하는 방법



KYUNG HEE UNIVERSITY