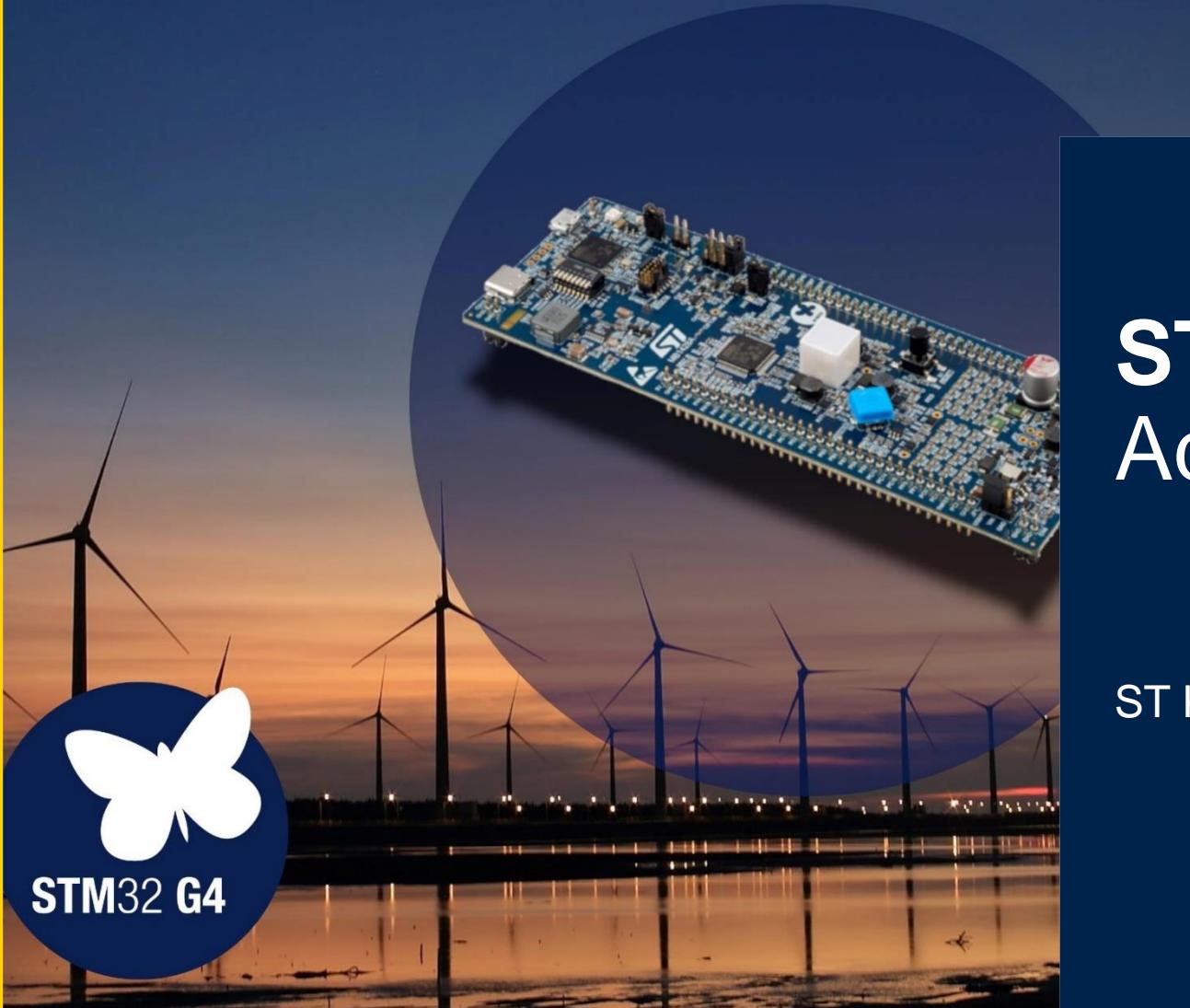


Discover digital power



STM32 Technical Training Advanced Course

ST Korea MMS Team

Agenda

- 1 10:00 – 10:20** Preparations
- 2 10:30 – 11:50** Hands-On Session: RCC, PWR
- **Lunch**

- 3 13:00 – 13:50** Hands-On Session: DMA
- 4 14:00 – 14:50** Hands-On Session: SPI
- 5 15:00 – 15:50** Hands-On Session: I2C
- 6 16:00 – 16:50** Hands-On Session: ADC

Preparations

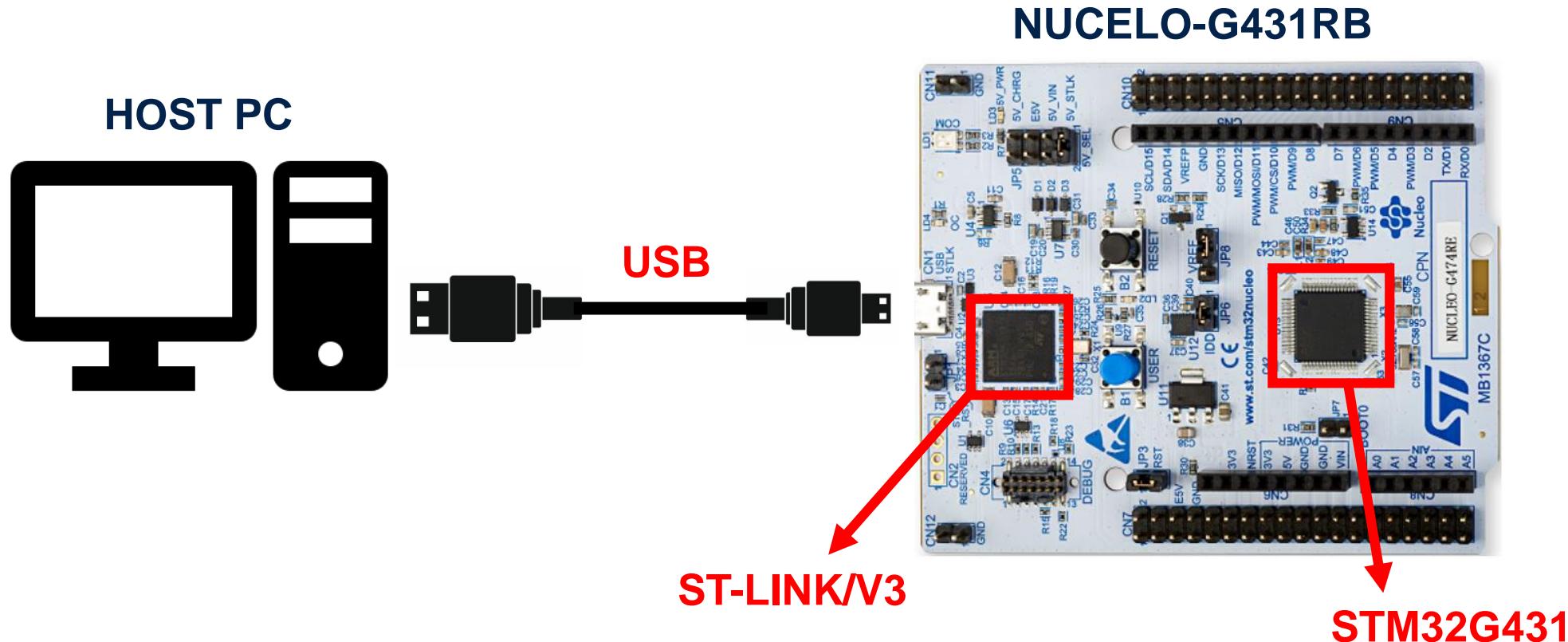


Software and Document Check

- STM32CubeIDE (STM32CubeG4 Package Update) 설치
 - URL: [STM32CubeIDE](#)
 - URL: [STM32CubeG4](#)
- STM32CubeProgrammer (또는 STM32 ST-Link Utility) 설치
 - URL: [STM32CubeProg](#)
- Reference Manual for STM32G431 MCU
 - URL: [RM0440](#)
- Datasheet for STM32G431 MCU
 - URL: [DS12589](#)
- NUCLEO/X-NUCLEO Board User Manual
 - URL: [UM2505](#) / [UM2665](#)

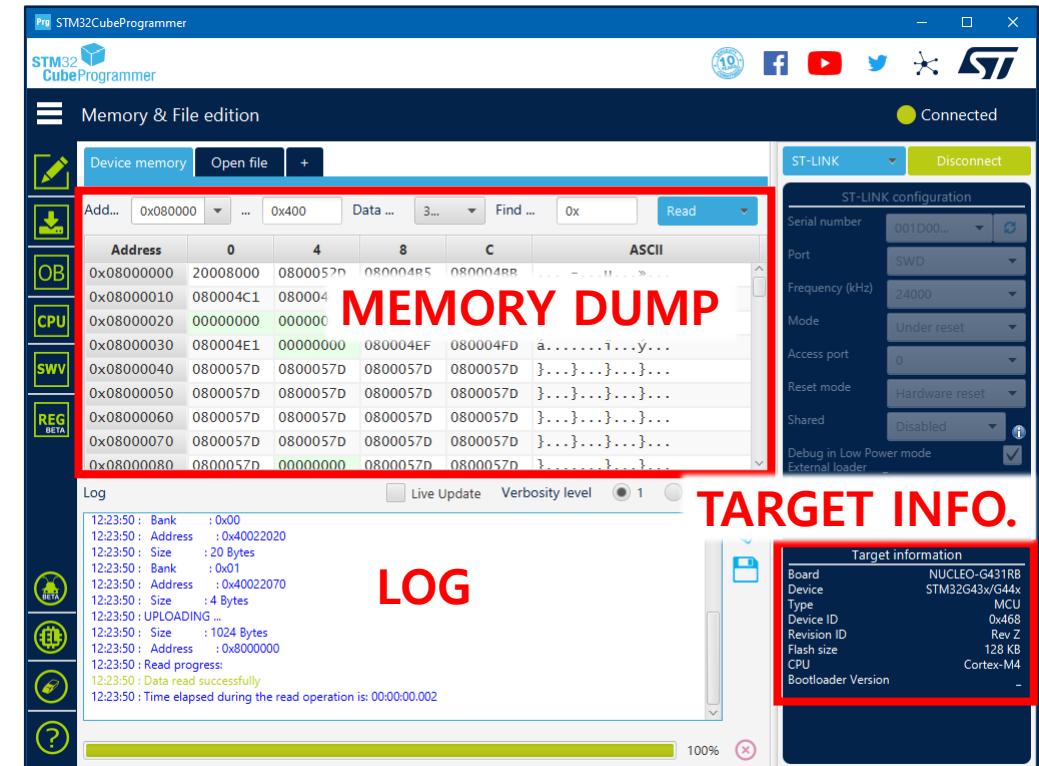
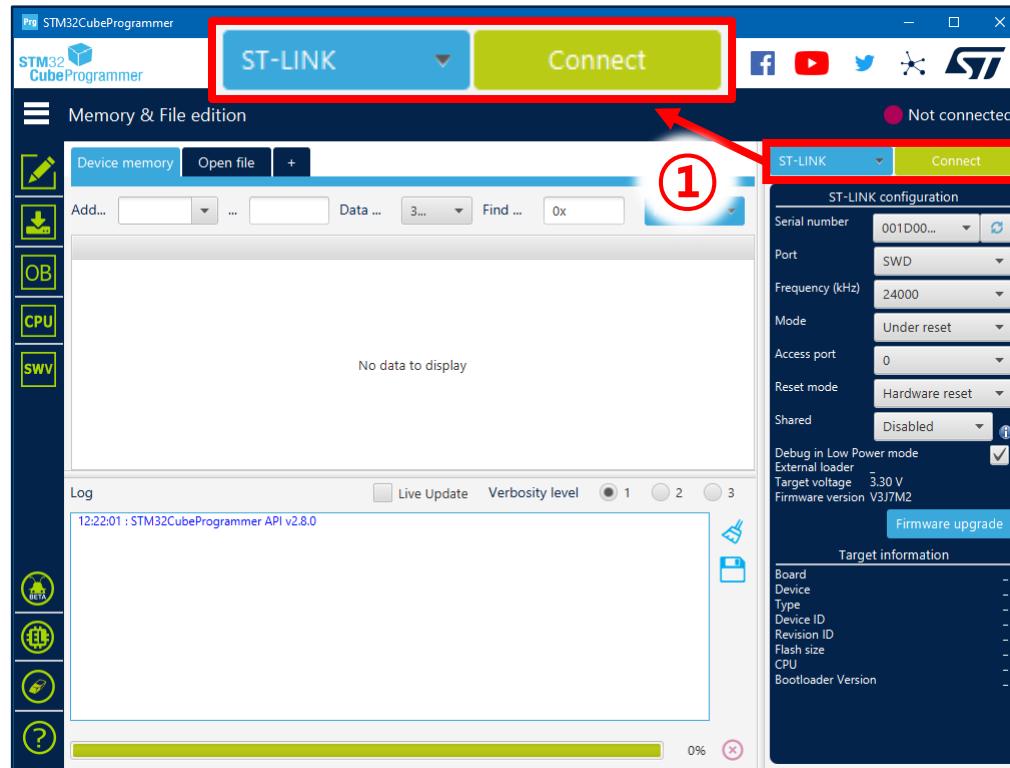
Board Check

- Host(PC)와 STM32G4 NUCELO Board를 **USB**로 연결
 - NUCELO board는 ST-LINK/V3 디버거를 포함



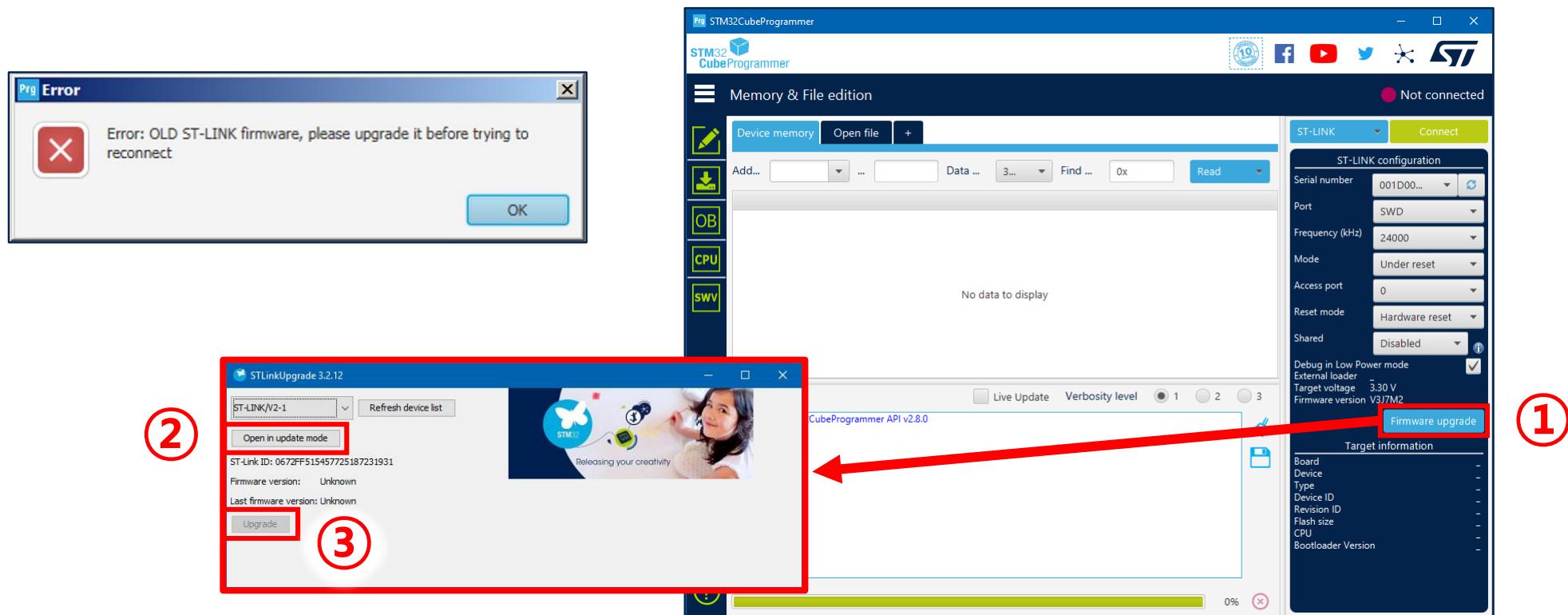
Board Check

- STM32CubeProgrammer()를 실행하여 Target 보드 정상 연결 확인
 - ST-LINK 선택, Connect 클릭
 - 디폴트로 선택되는 주소인 0x08000000 (플래시메모리)의 내용이 읽혀지는지 확인



Board Check

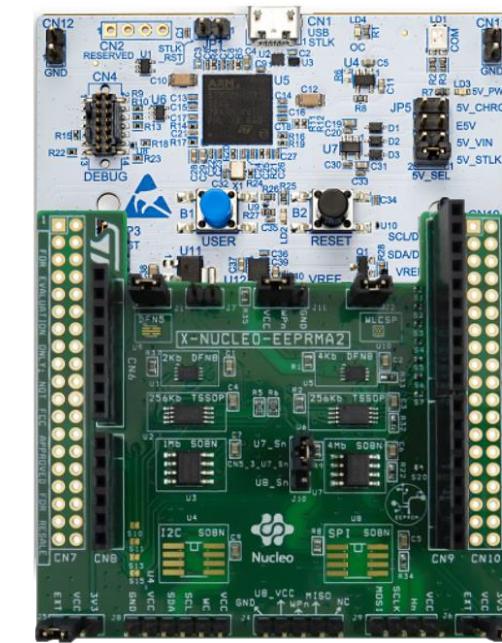
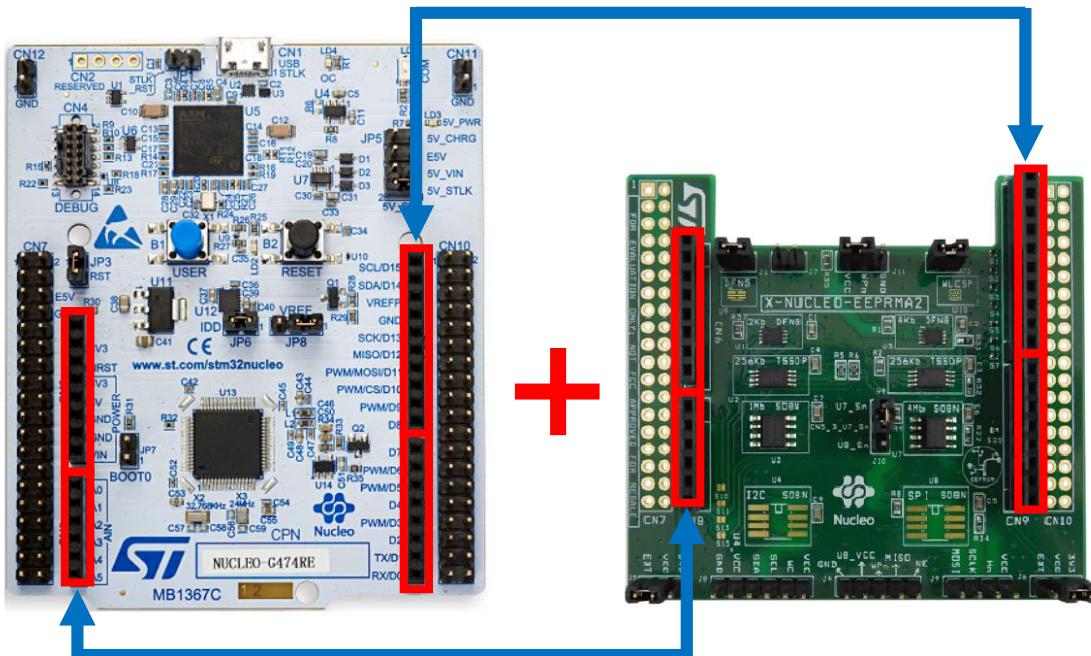
- 에러 메세지
 - 다음과 같이 ST-LINK 펌웨어를 업데이트 하라는 에러 메시지가 표시되는 경우, “**Firmware upgrade**” 을 눌러서 ST-LINK/V3 자체의 펌웨어를 업그레이드 한다.



X-NUCLEO 연결

- X-NUCLEO board (EEPRMA2)를 NUCLEO board와 결합한다.
 - **Arduino UNO3 커넥터**를 기준으로 결합. (* 물리적인 연결은 보드 전원을 제거한 상태로 수행)

Arduino Conn. (Right)



Arduino Conn. (Left)

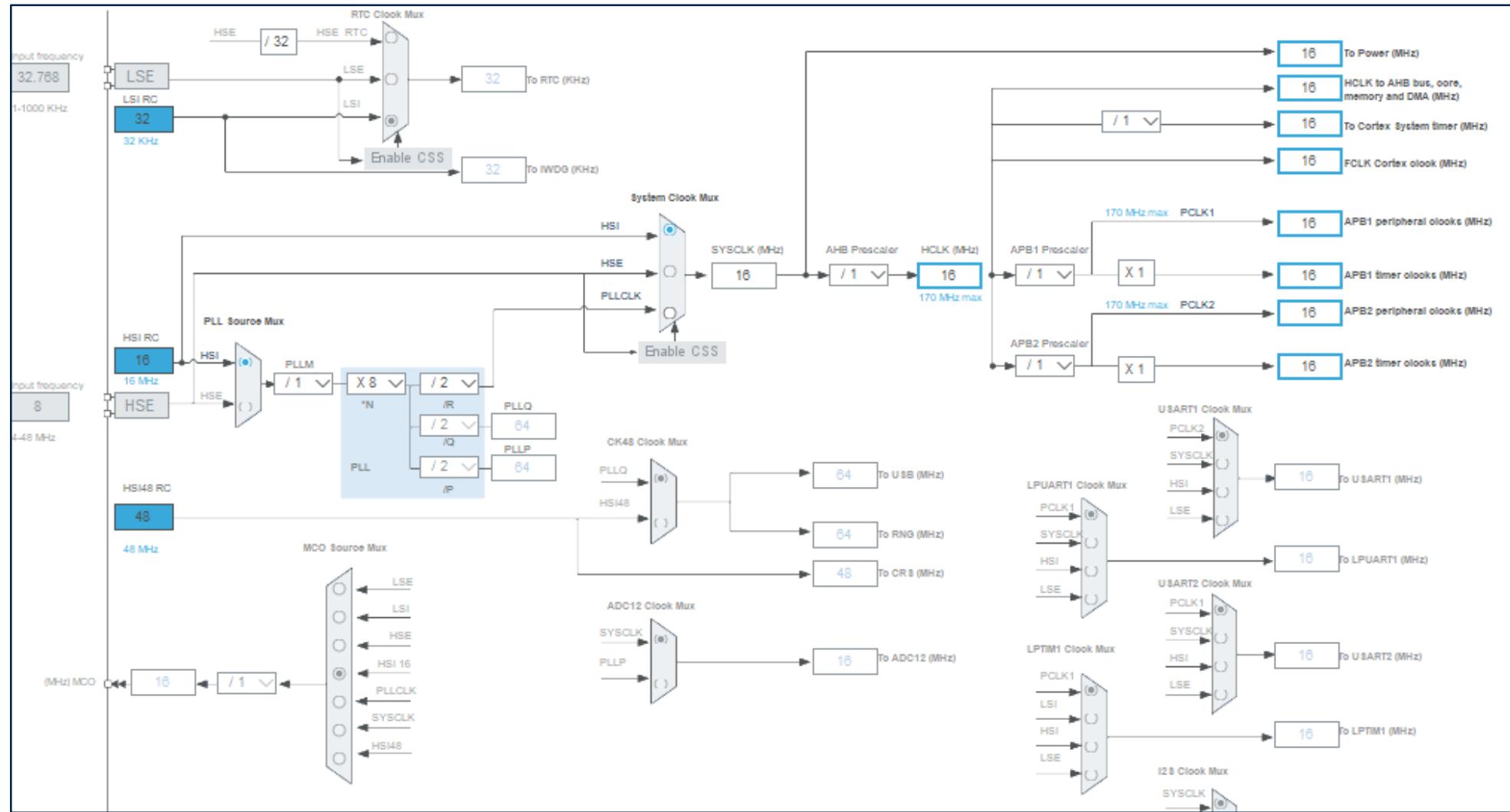
RCC (Reset and Clock Controller)



RCC (Reset and Clock Controller)

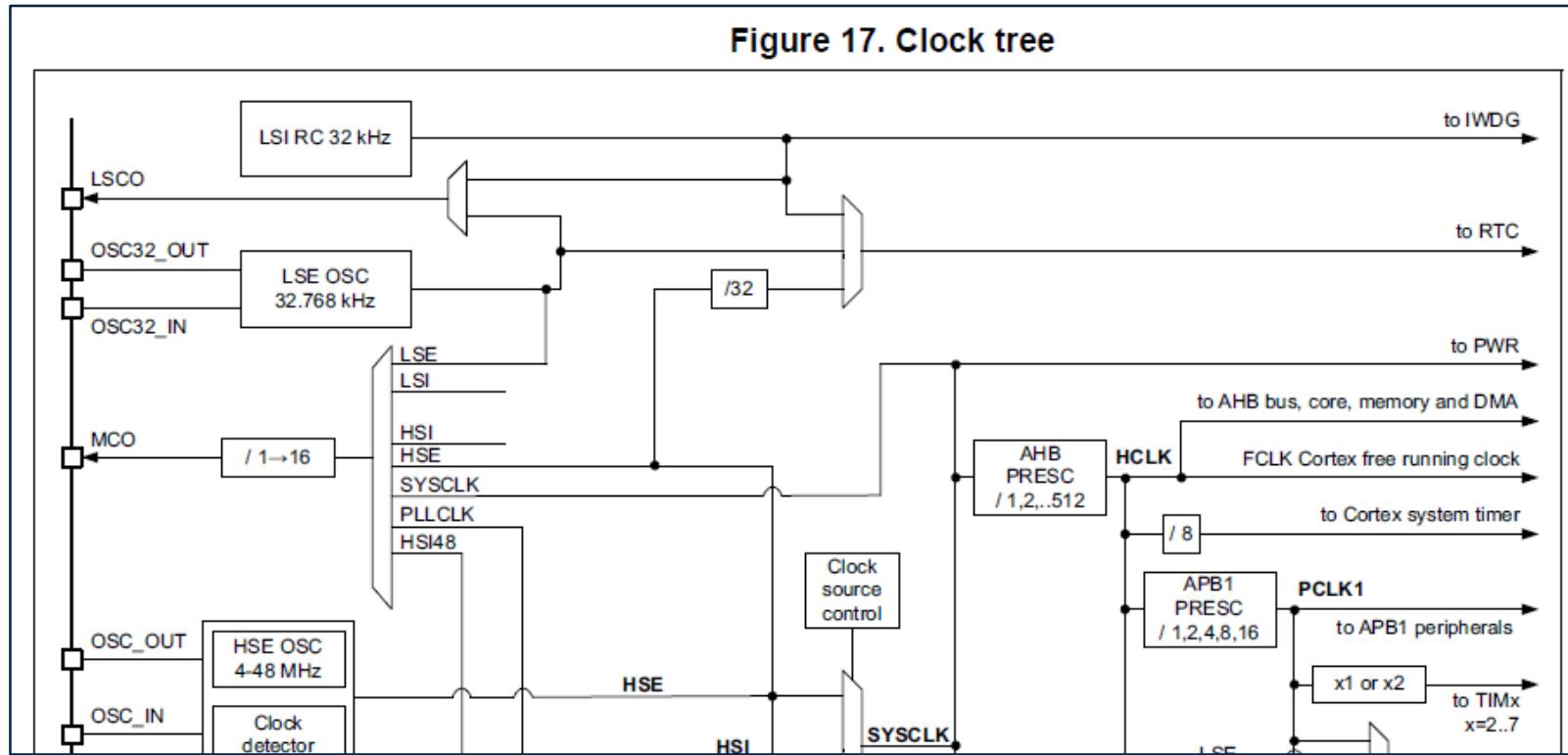
- RCC 는 system 과 peripheral 의 클럭을 관리한다
 - 3 개의 내부 클럭 소스 (HSI16, HSI48, LSI)
 - 2 개의 외부 클럭 소스 (HSE, LSE)
 - 1개의 내부 PLL (PLL)
 - Peripheral 들로 입력되는 클럭 소스 선택이 다양하다
- RCC 는 system 과 peripheral 의 리셋을 관리한다
 - System 리셋
 - Power 리셋
 - RTC domain 리셋
 - 레지스터의 리셋 컨트롤 비트를 통해서 peripheral 들을 개별적으로 리셋 시킬 수 있다

Clock tree in CubeIDE



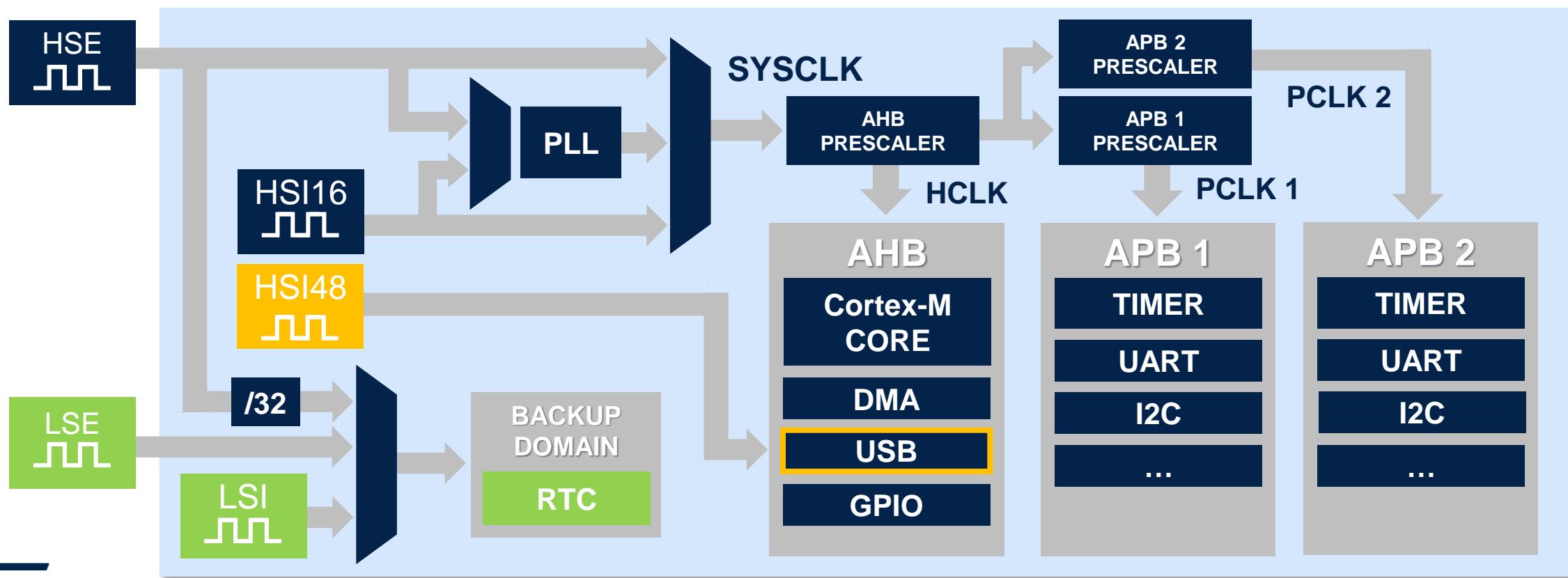
Clock tree in Reference manual

- Clock Tree @ Reference Manual (RM0440)
 - Device(MCU)의 정확한 Clock 정보를 제공.
 - Figure 17. Clock tree (for STM32G4 series devices) @ 7.2 Clocks



Clock tree in diagram

- STM32G431 MCU Clock tree 구조
 - 리셋 디폴트로 시스템 클럭 (SYSCLK) 의 소스는 HSI16 (16 MHz)이 선택된다



System clock

- **SYSCLK (System Clock)**

- HSI16 / HSE / PLL 출력 중에 하나를 SYSCLK 으로 선택해서 사용한다
- 내부 voltage regulator (LDO) 출력 설정에 따른 최대 클럭 속도는 아래와 같이 제한된다
 - 단계에 따라 dynamic voltage scaling이 적용되어 전력 효율을 높이게 된다

Voltage range	V_{CORE}	HSI16	HSE	PLL
Range 1 Boost mode	1.28 V	16 MHz	48 MHz	170 MHz
Range 2 Normal mode	1.20 V	16 MHz	48 MHz	150 MHz
Range 2	1.00 V	16 MHz	26 MHz	26 MHz

High-Speed Internal (HSI16/48) clock

- HSI16 내부 RC 클럭 특징
 - 16 MHz로 주파수가 고정이다.
 - HSE 대비 주파수 정확도는 낮지만 대신 소모 전류가 낮은 이점이 있다
 - HSE보다 startup time 이 빠른 편이다 (저전력모드에서 wakeup 이후)
 - I2C, U(S)ART, LPUART 와 같이 일부 패턴 (순서)로 wakeup 을 시키는 이벤트가 있는 peripheral 은 HSI 클럭을 STOP 모드 진입해 있는 동안에도 패턴 (순서) 감지용으로 enable 시킬 수 있다
- HSI48 내부 RC 클럭 특징
 - 48 MHz로 주파수가 고정이다
 - USB 및 RNG 전용의 기준 주파수 공급 용도로 사용한다
 - 기본적인 오차는 큰 편이지만, CRS(Clock recovery system)과 연동하여 오차를 줄일 수 있다
 - Reference Manual (RM0440), 8. Clock recovery system 참고
 - USB 기능에 사용하는 경우, 온도에 따른 오차가 큰 편으로 CRS를 사용하는 경우에도 대략 -15~65도 범위를 넘어서는 경우에는 오차 보정을 보장받기 어렵다. 이런 경우 HSE의 사용을 고려해야 한다

High-Speed External (HSE) clock

- HSE 외부 클럭 특징
 - Crystal / resonator 를 외부에 연결하는 경우, 4 ~ 48 MHz 까지 입력 가능
 - Bypass clock (구형파 입력)을 외부에 연결하는 경우, 최대 48 MHz 까지 입력 가능
 - HSI16보다 주파수 정확도가 높지만 소모전류가 높다
 - USB 와 같이 높은 PPM 사양 (full speed 기준 +-0.25%, 2500 ppm) 을 요구하며 HSI48+CRS 조합으로 대응이 불가능한 (*온도) 어플리케이션에서는 HSE를 사용해야 한다
- Clock Security System (CSS)
 - HSE 클럭 신호가 들어오지 않는 failure 를 감지하면 다음과 같은 동작을 자동으로 수행한다
 - NMI (Non-maskable interrupt) 발생
 - TIM[1, 8, 15, 16, 17, 20]의 break input 소스 발생 (모터 어플리케이션 등에서 안전을 위한 처리)
 - RCC_CFGR -> STOPWUCK 설정에 따라 HSI16 클럭으로 시스템 클럭을 자동 전환하고 어플리케이션을 계속 수행한다

Low-Speed Internal (LSI) clock

- LSI 내부 RC 클럭 특징
 - 약 32 KHz 로 고정된 내부 클럭
 - 소모전류가 LSE 보다 낮지만 주파수 정확도가 낮다
 - 주파수 정확도는 꽤 낮은 편, 데이터 시트(5.3.8 Internal clock source characteristics@DS12589)를 참고
 - RTC, IWDG, LPTIM, peripheral 의 클럭 소스 중의 하나로만 사용될 수 있다
 - SHUTDOWN, VBAT 모드를 제외한 모든 저전력 모드에서 사용할 수 있다

Low-Speed External (LSE) clock

- LSE 외부 클럭 특징

- Bypass clock 모드와 crystal / resonator 모드의 클럭 입력을 지원한다
- 어플리케이션에서 높은 정확도의 RTC 시간 유지 등이 필요한 경우 LSE 가 필요하다
- RTC, LPTIM, U(S)ART peripheral 의 클럭 소스중의 하나로만 사용할 수 있다
- VBAT, SHUTDOWN, STANDBY 등 모든 저전력 모드에서 사용할 수 있다
- RCC_BDCR 레지스터의 LSEDRV 비트를 통해서 drive strength 를 아래 4가지 중 하나를 선택할 수 있다

Mode	Maximum critical crystal gm ($\mu\text{A/V}$)	Consumption (nA)
Ultra-low power	0.5	250
Medium-low driving	0.75	315
Medium-high driving	1.7	500
High driving	2.7	630

Clock sources

- 정확한 내용은 데이터시트의 electrical characteristics 참조

RESET

CLK Source	Frequency	Conso	Precision (0-85°C)	Settling time
HSI16	16 MHz	155 µA	±1 %	3.8 µs
HSI48	48 MHz	340 µA	±4.5 % (trim: ±1 %)	2.5 µs
HSE (external)	4 ~ 48 MHz	~ 440 µA (8MHz, 10pF)	±0.01 % (100ppm)	2 ms
PLL	26 ~ 170 MHz	~ 520 µA (@344MHz VCO)	N/A	15 µs
LSI	typ. 32 kHz	0.11 µA	±10 %	125 µs
LSE (external)	typ. 32.768 kHz	~ 0.25 µA	±0.002 % (20ppm)	~2 s

LSE usually woken-up only once after power-on

- System reset
 - VDD 입력 전원은 유지된 상태에서 아래 원인에 의해 리셋이 발생한다
 - NRST 핀 리셋
 - WWDG 리셋
 - IWDG 리셋
 - 소프트웨어 리셋 (through NVIC)
 - Low power mode security 리셋
 - Option byte loader 리셋
 - BOR 리셋
 - RCC_CSR 레지스터의 리셋 소스 플래그로 리셋 발생 원인을 알 수 있다
 - 위의 리셋 발생시 NRST 핀 리셋도 항상 함께 발생한다
 - 일부 RCC 레지스터, PWR 레지스터, RTC domain 을 제외하고 모든 레지스터를 리셋 한다

- Power reset

- 아래의 경우 중 하나가 발생하면 power reset 을 발생하고 backup domain 을 제외한 모든 레지스터를 리셋 한다
 - VDD 입력 전원이 BOR (PDR) 이하로 내려간 경우 (VBAT 전원은 유지)
 - RTC 도메인을 제외한 모든 레지스터 리셋
 - STANDBY 모드에서 wakeup 하는 경우
 - V_{CORE} 도메인만 리셋
 - SHUTDOWN 모드에서 wakeup 하는 경우
 - RTC 도메인을 제외한 모든 레지스터 리셋

- RTC domain reset

- 아래의 경우 중 하나가 발생하면 RTC domain 을 포함한 모든 레지스터를 리셋 한다
 - 소프트웨어에서 명시적으로 RCC_BDCR 레지스터의 BDRST 비트로 리셋을 수행한 경우
 - VDD 입력 전원과 VBAT 전원이 모두 BOR (PDR) 이하로 내려간 경우

Power supply supervisor

- STM32G431은 BOR 이 POR, PDR 기능도 함께 한다
 - BOR 은 SHUTDOWN 모드를 제외하고 항상 enable
 - 공장 초기 설정은 BOR 레벨 0이며, Option byte 의 BOR_LEV[2:0] 으로 변경 가능 (5종류)
 - 6.2 Power supply supervisor @RM0440
 - 5.3.3 Embedded reset and power control block characteristics @DS12589

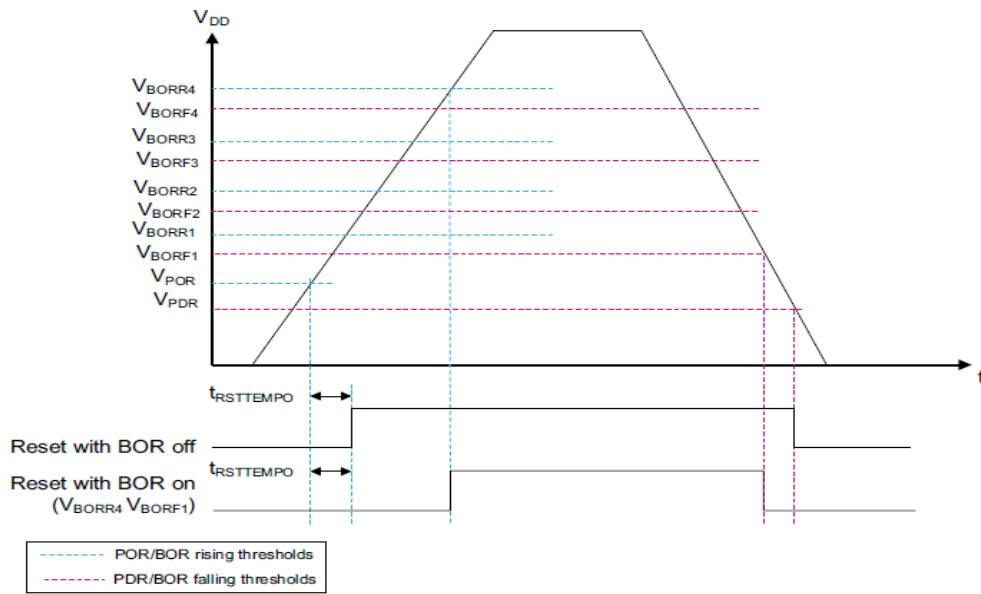


Table 19. Embedded reset and power control block characteristics

Symbol	Parameter	Conditions ⁽¹⁾	Min	Typ	Max	Unit
$t_{RSTTEMPO}^{(2)}$	Reset temporization after BOR0 is detected	V_{DD} rising	-	250	400	μs
$V_{BOR0}^{(2)}$	Rising edge	1.62	1.66	1.7		V
	Falling edge	1.6	1.64	1.69		V
V_{BOR1}	Rising edge	2.06	2.1	2.14		V
	Falling edge	1.96	2	2.04		V
V_{BOR2}	Rising edge	2.26	2.31	2.35		V
	Falling edge	2.16	2.20	2.24		V
V_{BOR3}	Rising edge	2.56	2.61	2.66		V
	Falling edge	2.47	2.52	2.57		V
V_{BOR4}	Rising edge	2.85	2.90	2.95		V
	Falling edge	2.76	2.81	2.86		V

PWR (Power Control)



PWR (Power Control)

- 특징

- 7 개 종류의 저전력 모드 지원
 - I/O 로 wakeup 이 가능한 30 nA 저전력 모드 (SHUTDOWN 모드)
 - 32 KB RAM 내용이 유지되는 350 nA 저전력 모드 (STADNBY + 32 KB RAM 모드)
 - 많은 종류의 peripheral 로 wakeup 가능
- Run 모드에서 최소 100 uA / MHz 의 소모전류
- 배터리로 유지 가능한 RTC 와 backup 레지스터
- 세분화된 독립 전원 핀

Overview

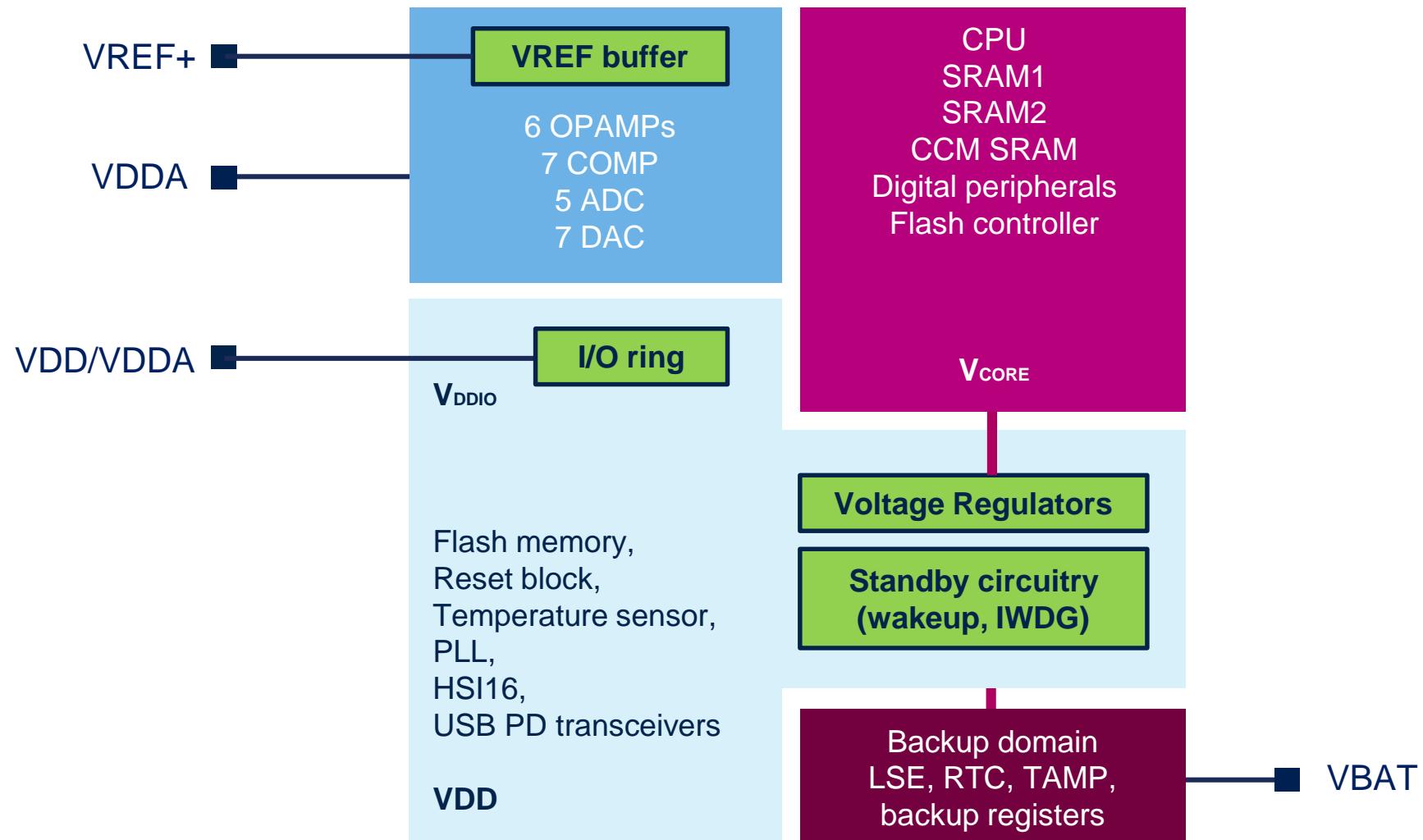
Wake-up time	RUN (Range1 boost) at 170 MHz from Flash	173 μ A / MHz
	RUN (Range1 normal) at 150 MHz from Flash	163 μ A / MHz
	RUN (Range2) at 26 MHz	128 μ A / MHz
	LPRUN at 2 MHz, running from SRAM1	183 μ A / MHz
	SLEEP at 26 MHz	37 μ A / MHz
	STOP 0 (RTC disabled)	190 μ A
	STOP 1 (RTC disabled)	80 μ A
	STANDBY (SRAM2 retained)	435 nA
	STANDBY (RTC,IWDG disabled)	130 nA
	SHUTDOWN	43 nA
	VBAT (RTC disabled)	7 nA

Typ @ VDD = 1.8 V @ 25 °C
Oscillator: HSE bypass for
LPRUN, LPSLEEP

Application benefits

- High performance
→ CoreMark score = 3.42/MHz
- Outstanding power efficiency

Power Schemes



- Voltage regulators
 - MR 또는 LPR로부터 V_{CORE} 전원을 만든다
 - Main Regulator (MR)
 - Range 1 : 코어와 peripheral 을 최대 속도 170 MHz 로 동작 가능
 - Range 2 : 코어와 peripheral 을 최대 속도 150 MHz 로 제한 한다
 - RUN, SLEEP, STOP0 모드에서 사용됨
 - Low Power Regulator (LPR)
 - 코어 속도는 2 MHz 로 제한되며 PCLK 가 아닌 다른 클럭 소스를 사용할 수 있는 peripheral 은 HSI16 MHz 를 peripheral clock 으로 선택할 수 있다
 - LP RUN, LP SLEEP, STOP1, STANDBY with SRAM2 retention 모드에서 사용된다

Sleep and Low-power Sleep modes

- Cortex 코어 클럭 OFF, 일부 peripheral 클럭 ON/OFF 선택 가능
 - RCC_AHB1SMENR, RCC_APB1SMENR1, etc.,
- Sleep 모드: MR (Main regulator) ON 상태
- Low power sleep 모드: LPR (Lower power regulator) ON 상태
 - HSI48은 Low-power sleep 모드에서 OFF
- 모든 I/O 와 peripheral 이벤트로 Sleep 모드에서 wakeup 가능
- **WFI** (Wait For Interrupt) 또는 **WFE** (Wait For Event) 명령어로 Sleep 모드 진입
 - **Sleep Now**: WFI / WFE 명령어가 수행되는 즉시 진입
 - **Sleep on Exit**: 가장 우선순위가 낮은 ISR의 수행이 종료되고 나서 진입
 - Sleep 모드에 진입하기 전에 스택 POP을 하지 않고 다음 번 wakeup 인터럽트 발생 시 스택 PUSH를 하지 않음으로 시간을 절약
 - Cortex-M4 System Control Register [SLEEPONEXIT]

Stop modes

- Cortex 코어 클럭 OFF
- PLL / HSI16 / HSI48 / HSE 클럭 OFF
- LSE / LSI 클럭 ON/OFF 선택 가능
- 일부 peripheral 클럭 ON/OFF 선택 가능
- Stop0 모드는 MR 이 ON 상태, Stop1 모드는 LPR 이 ON 상태
- SRAM1, SRAM2, CCM SRAM 및 모든 peripheral 레지스터 값과 I/O 상태 유지
- I2C, U(S)ART, LPUART 로 wakeup 을 시켜야 되는 경우 HSI16 클럭 ON 가능
- Sleep 모드에 비해 wakeup 소스는 적지만 모든 I/O 로 wakeup 가능하며 I2C, U(S)ART, LPTIM 등의 peripheral 의 이벤트로 wakeup 가능

Stop modes comparison

	STOP0	STOP1
Consumption (STM32G474)	25°C, 1.8V 155 µA	46 µA when RTC is disabled
Wakeup time to 16 MHz	6 µs in Flash memory initially powered down 3 µs in RAM	9.8 µs in Flash memory initially powered down 6.9 µs in RAM
Wakeup clock	HSI16 at 16 MHz	
Regulator	Main regulator	Low power regulator
Peripherals	RTC, I/Os, BOR, PVD, PWM, UCPD, DACs, OPAMPs, COMPs, IWDG 1 LP TIMER 1 LP UART (Start, address match or byte reception) 5 U(S)ART (Start, address match or byte reception) 4 I2C (address match)	

- Cortex 코어 클럭 OFF
- PLL / HSI16 / HSI48 / HSE 클럭 OFF
- LSE / LSI 클럭 ON/OFF 선택 가능
- BOR, RTC, IWDG 를 제외한 모든 peripheral 클럭 OFF
- MR 과 LPR 모두 OFF, 모든 레지스터와 SRAM 값 소실 (backup register 제외)
 - PWR_CR3 의 RRS 비트를 세팅하면 LPR 을 On 시켜서 SRAM2 값을 유지 가능
- I/O 는 analog float 상태로 변경
 - Standby 모드 동안 일부 I/O 에 pull-up, pull-down 설정 가능 (PWR_PUCRx / PWR_PDCRx registers (x = A,B,...H))
- Wakeup 핀 I/O 5개와 BOR, RTC, IWDG 로만 wakeup 가능
- BOR 동작으로 인해 VDD 가 BOR 이하로 내려가면 VBAT 전환 가능
- Wakeup 이후 코드는 다시 처음부터 수행

- Standby 모드와 동일하고 차이점은
 - BOR /PDR 동작을 하지 않으므로 VBAT 전환을 지원하지 않는다
 - LSE 만 ON 가능 하고 LSI 는 항상 OFF 이므로 IWDG 를 지원하지 않는다
 - Shutdown 모드에서 wakeup 되면 POR 리셋이 발생한다
- 128-byte 의 backup registers 값 유지 가능
- Wakeup 핀 I/O 5개와 RTC 로만 wakeup 가능

VBAT backup domain

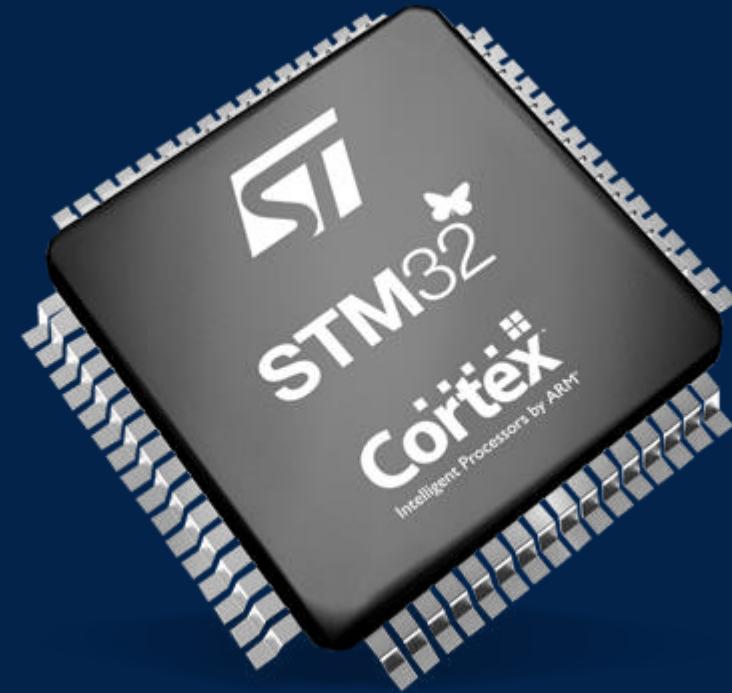
- Backup domain 포함 내용
 - RTC (LSE로 동작), 3개의 tamper 핀
 - 128 bytes backup 레지스터
 - RCC_BDCR 레지스터
- VDD 전압에 따라서 backup domain 의 전원 소스가 내부적으로 스위칭
 - VDD 전압 > BOR 전압 이면 backup domain 전원 = VDD 핀
 - VDD 전압 < BOR 전압 이면 backup domain 전원 = VBAT 핀
- VBAT 전압은 내부 ADC 입력 채널로 연결이 되어 있어서 모니터링 가능 ($V_{BAT}/3$)
- VBAT 충전(charging) 가능

Low-power modes summary

Mode	Regulator	CPU	Flash	SRAM	Clocks	Peripherals
Run	MR Range 1	Yes	ON ¹	ON	Any	All
	MR Range 2					All except USB, RNG
LPRun	LPR	Yes	ON ¹	ON	Any except PLL	All except USB, RNG
Sleep	MR Range 1	No	ON ¹	ON ²	Any	All, Any interrupt or event
	MR Range 2					
LPSleep	LPR	No	ON ¹	ON ²	Any except PLL	All, Any interrupt or event
Stop 0	MR	No	OFF	ON	LSE/LSI	Reset pin, all I/Os BOR, PVD, PVM, RTC, IWDG, COMPx, DACx, OPAMPx, USARTx, LPUART, I2Cx, LPTIM1, USB, UCPD
Stop 1	LPR					
Standby	LPR	DOWN	OFF	SRAM2 ON	LSE/LSI	Reset pin, 5 WKUPx pins BOR, RTC, IWDG
	OFF			DOWN		
Shutdown	OFF	DOWN	OFF	DOWN	LSE	Reset pin, 5 WKUPx pins RTC

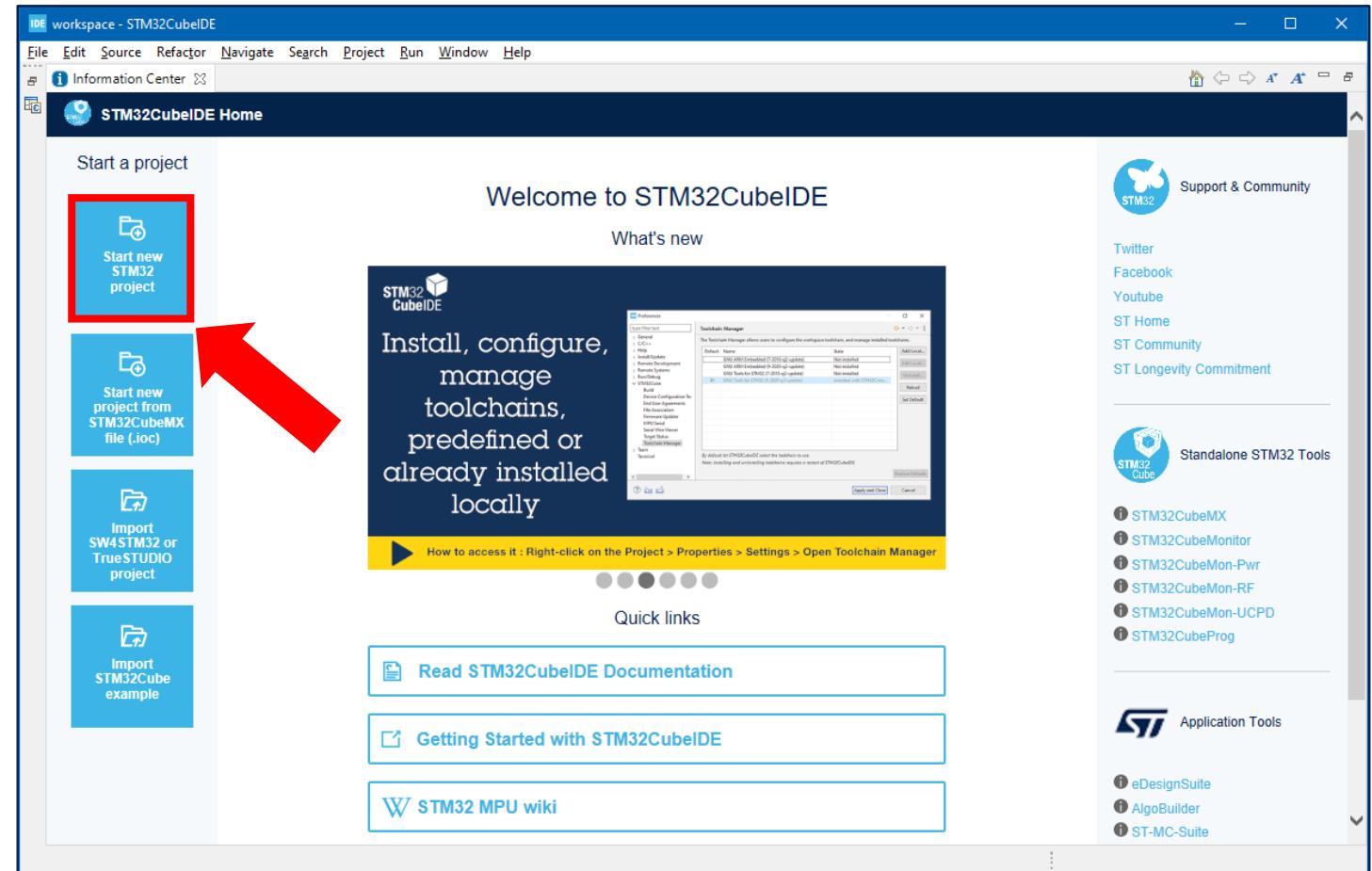
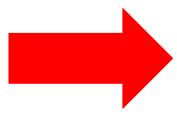
1. Can be put in power-down and clock can be gated off
2. SRAM1 and SRAM2 can be gated off independently

Hands-On Session: RCC & PWR



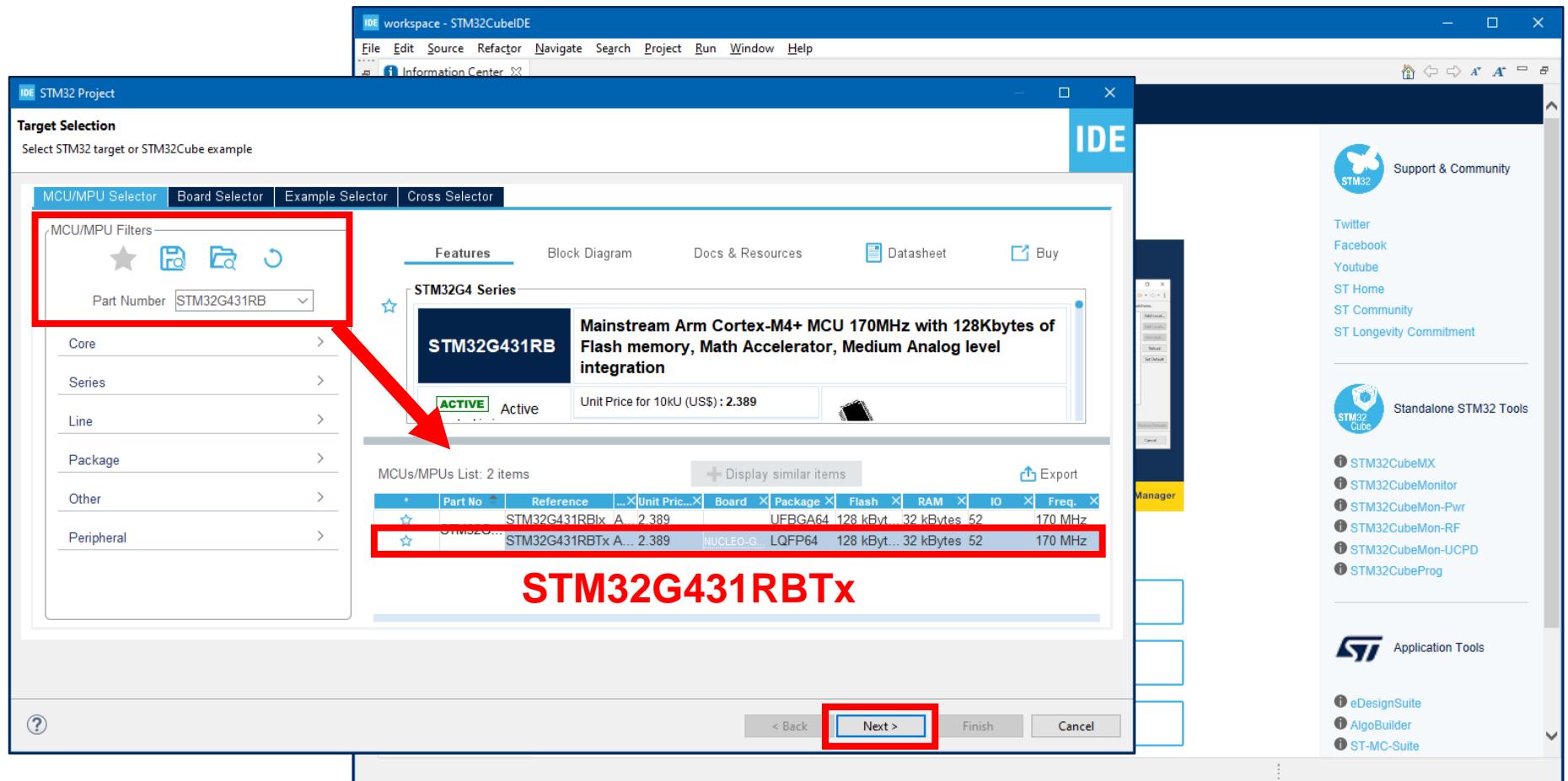
Run STM32CubeIDE

- Run STM32CubeIDE
- Start new STM32 project



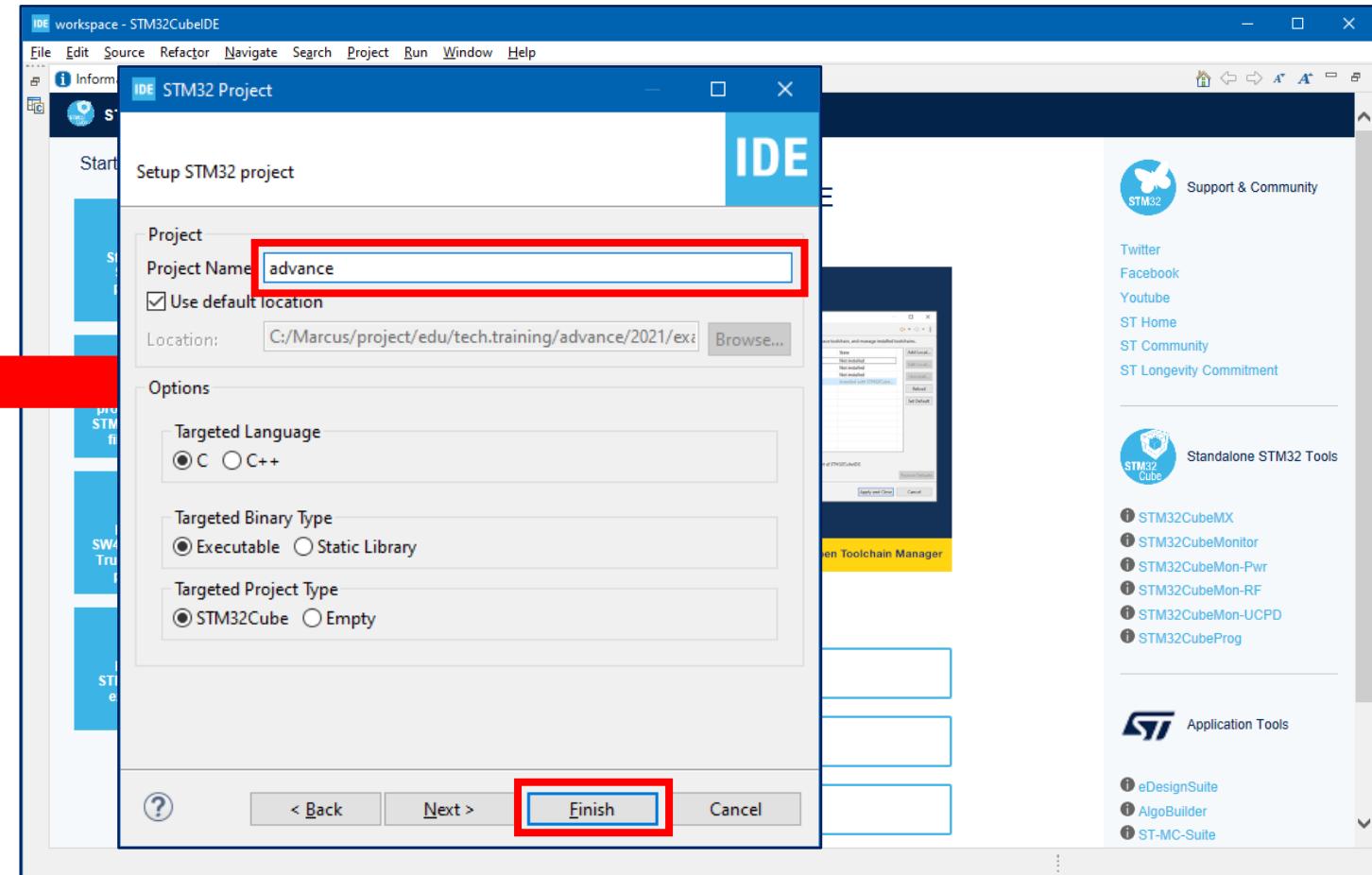
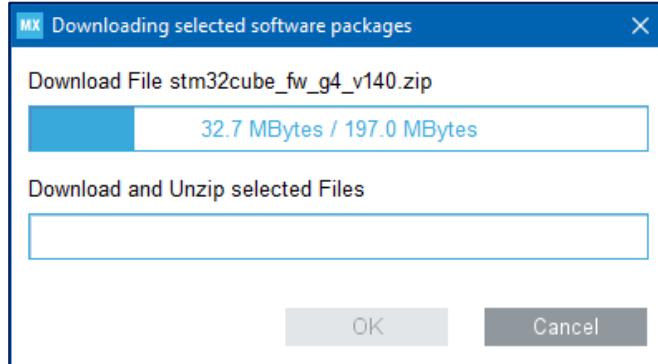
Target Selection

- Part Number Search: **STM32G431RB**
- Click [Next]



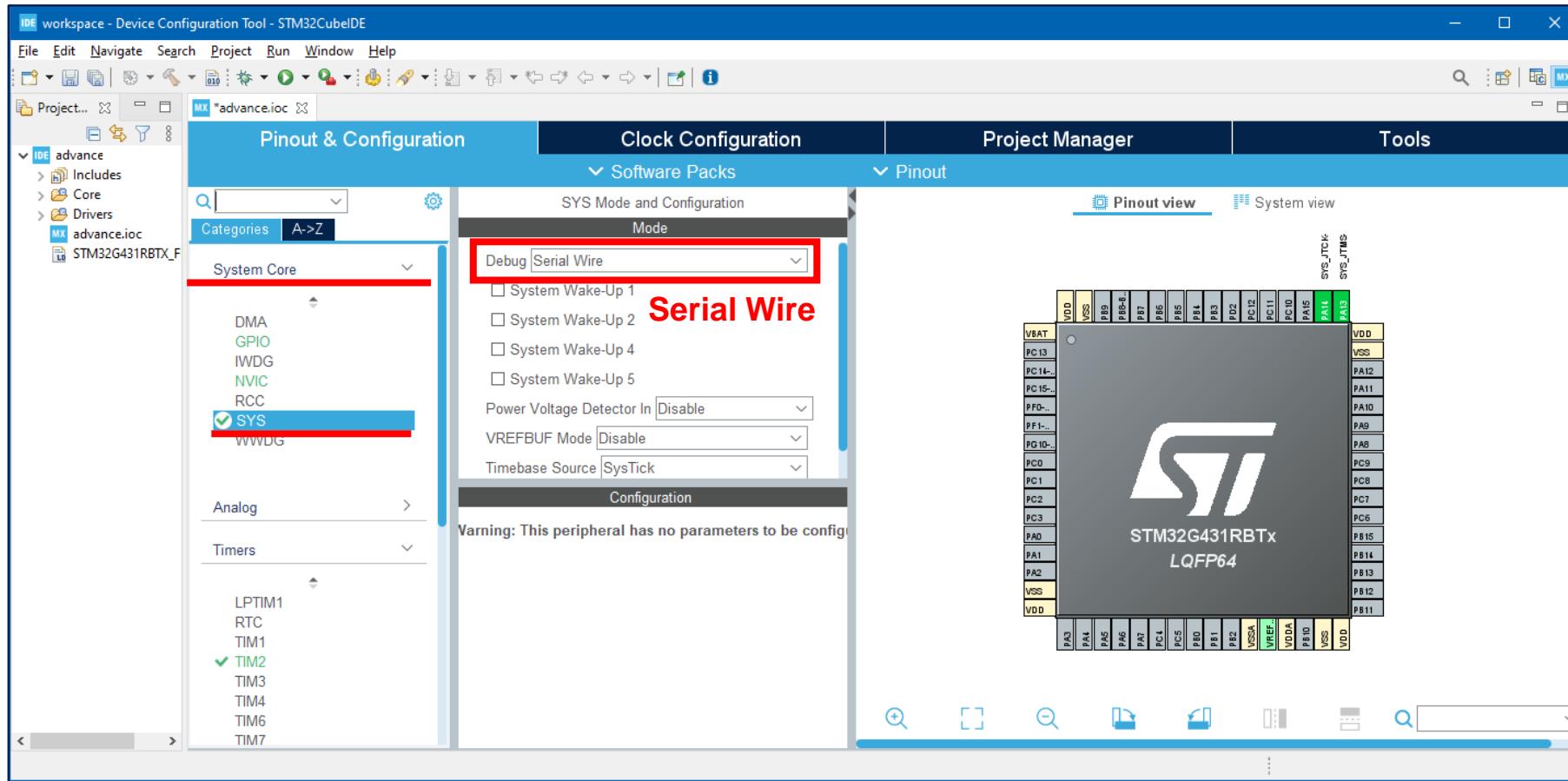
Setup STM32 project

- Project Name: 프로젝트명 입력
- Click [Finish]



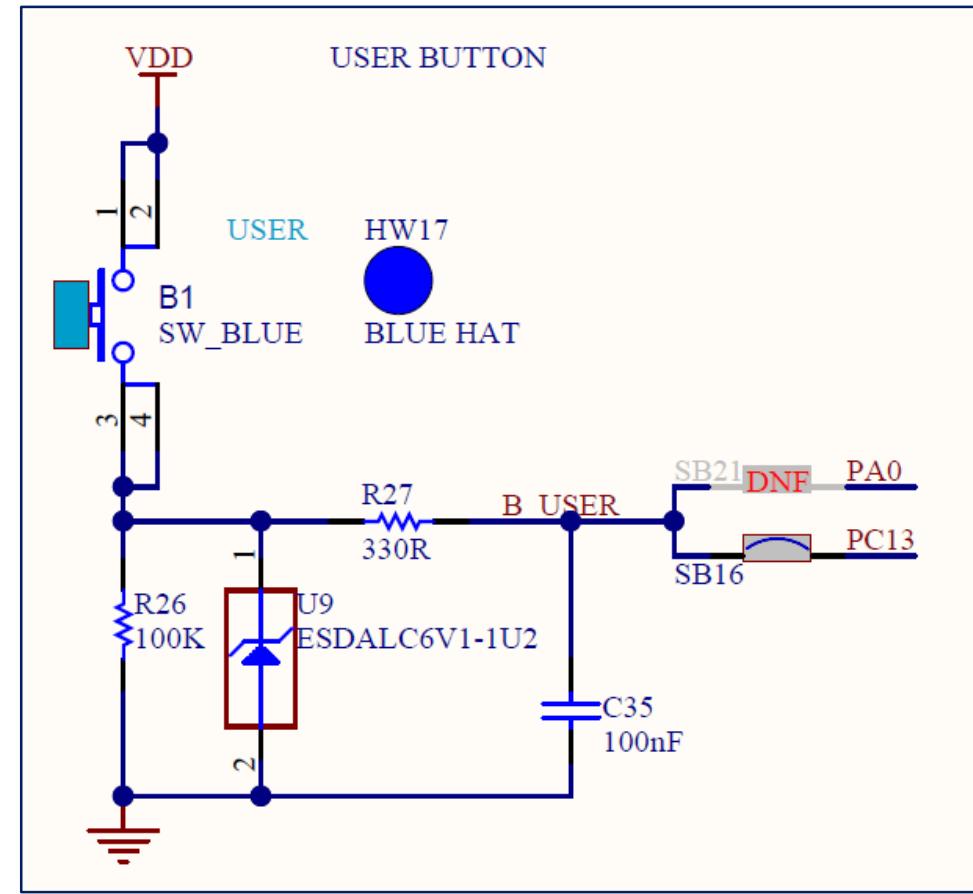
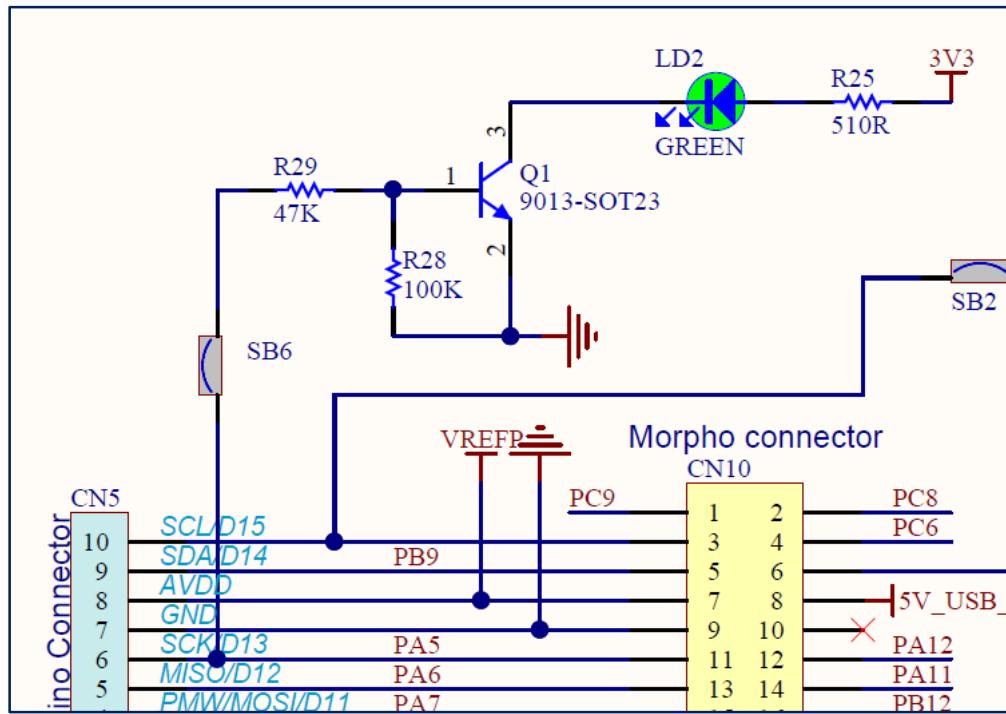
Configuration

- Pinout & Configuration
 - System Core > SYS > *Debug: Serial Wire*



Configuration

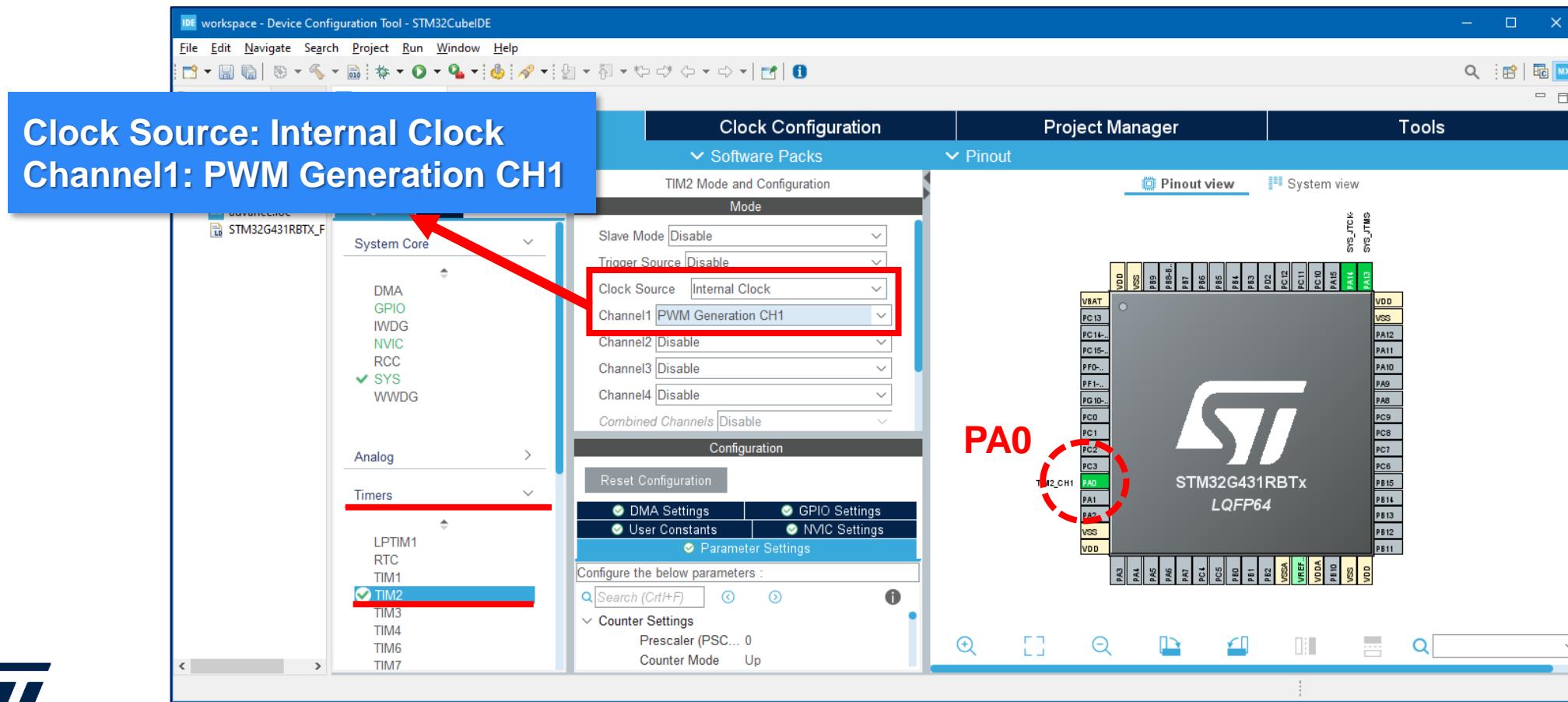
- LED Control: **PA5 (TIM2 CH1)**
- Button Control: **PC13**



Configuration

- Pinout & Configuration

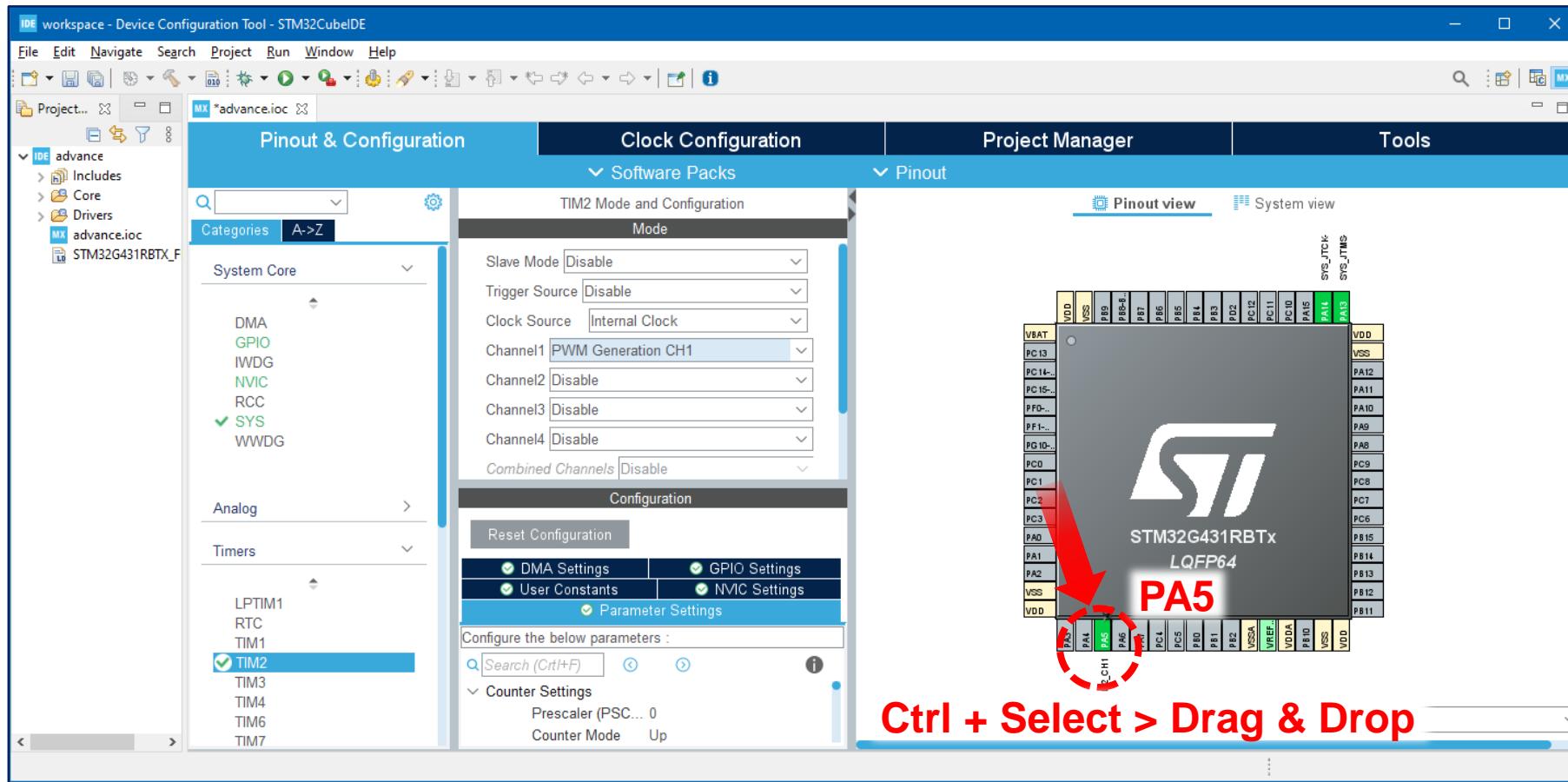
- Timers > TIM2 > *Internal Clock, PWM Generation CH1 (Channel1)*



Configuration

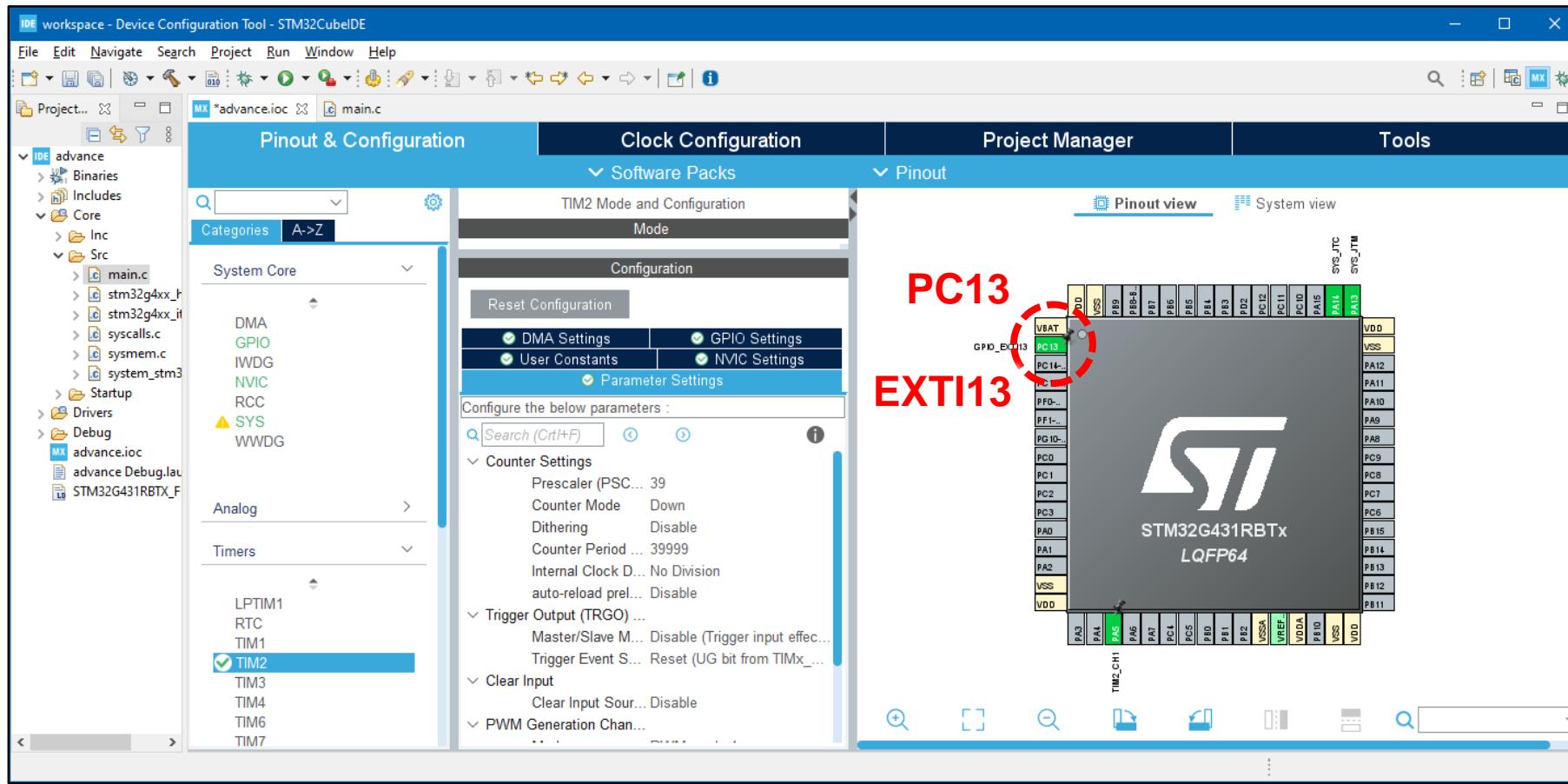
- Pinout & Configuration

- Timers > TIM2 > *Internal Clock, PWM Generation CH1 (Channel1)*



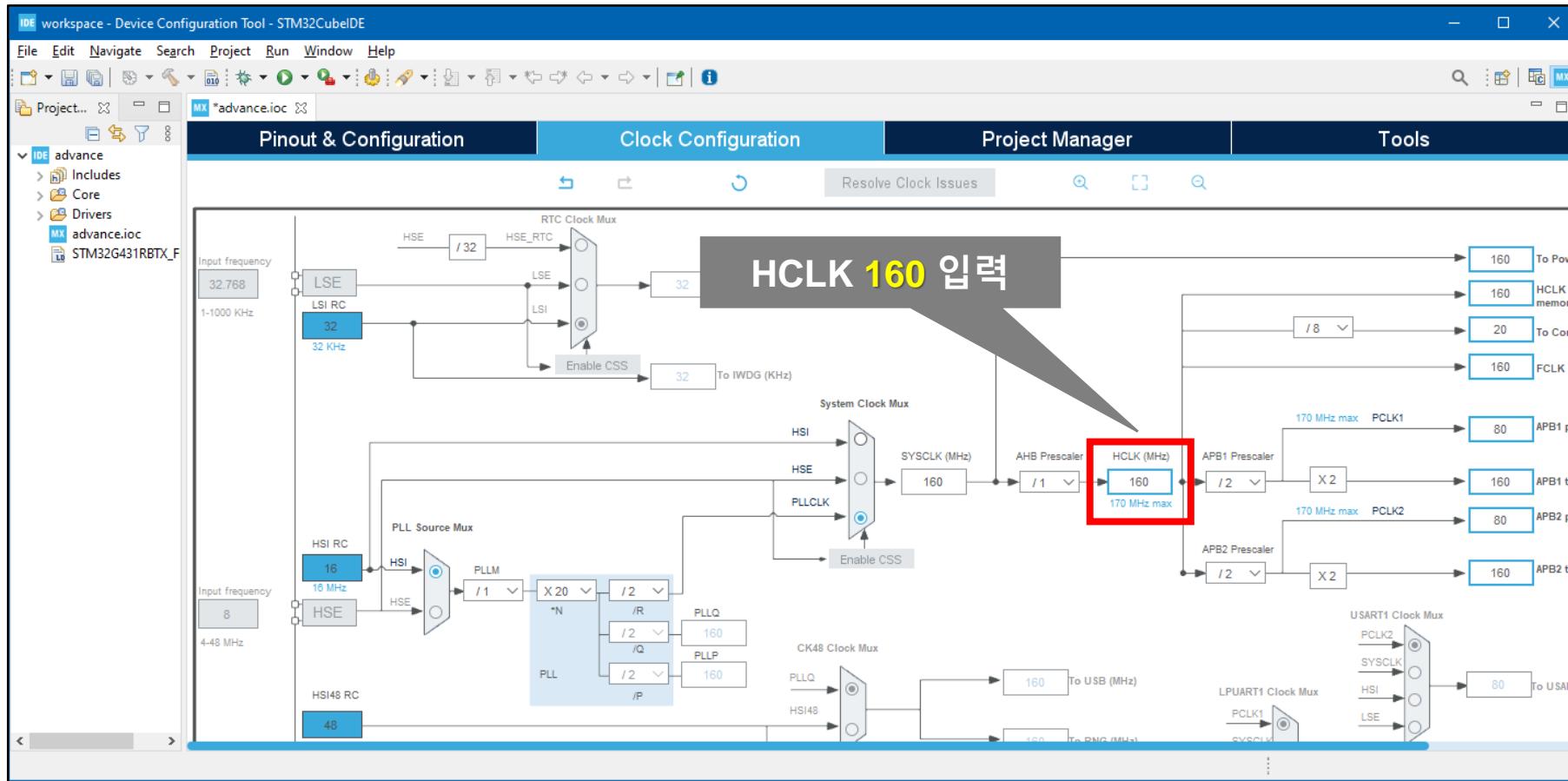
Configuration

- Pinout & Configuration
 - PC13 > EXTI13



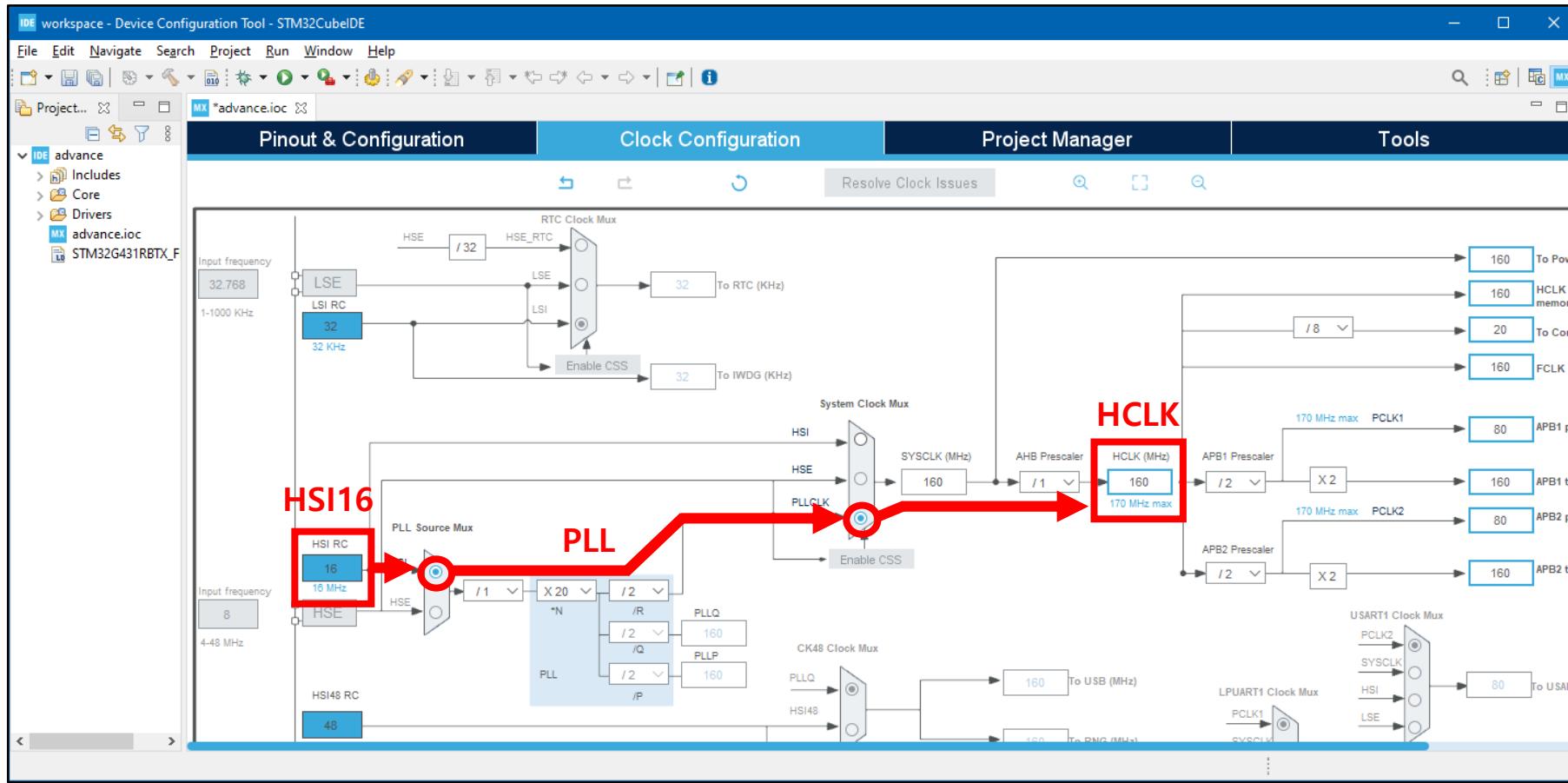
Clock Configuration

- SYSCLK 및 HCLK을 160MHz 설정 (PLL 사용)
- [OK] Click



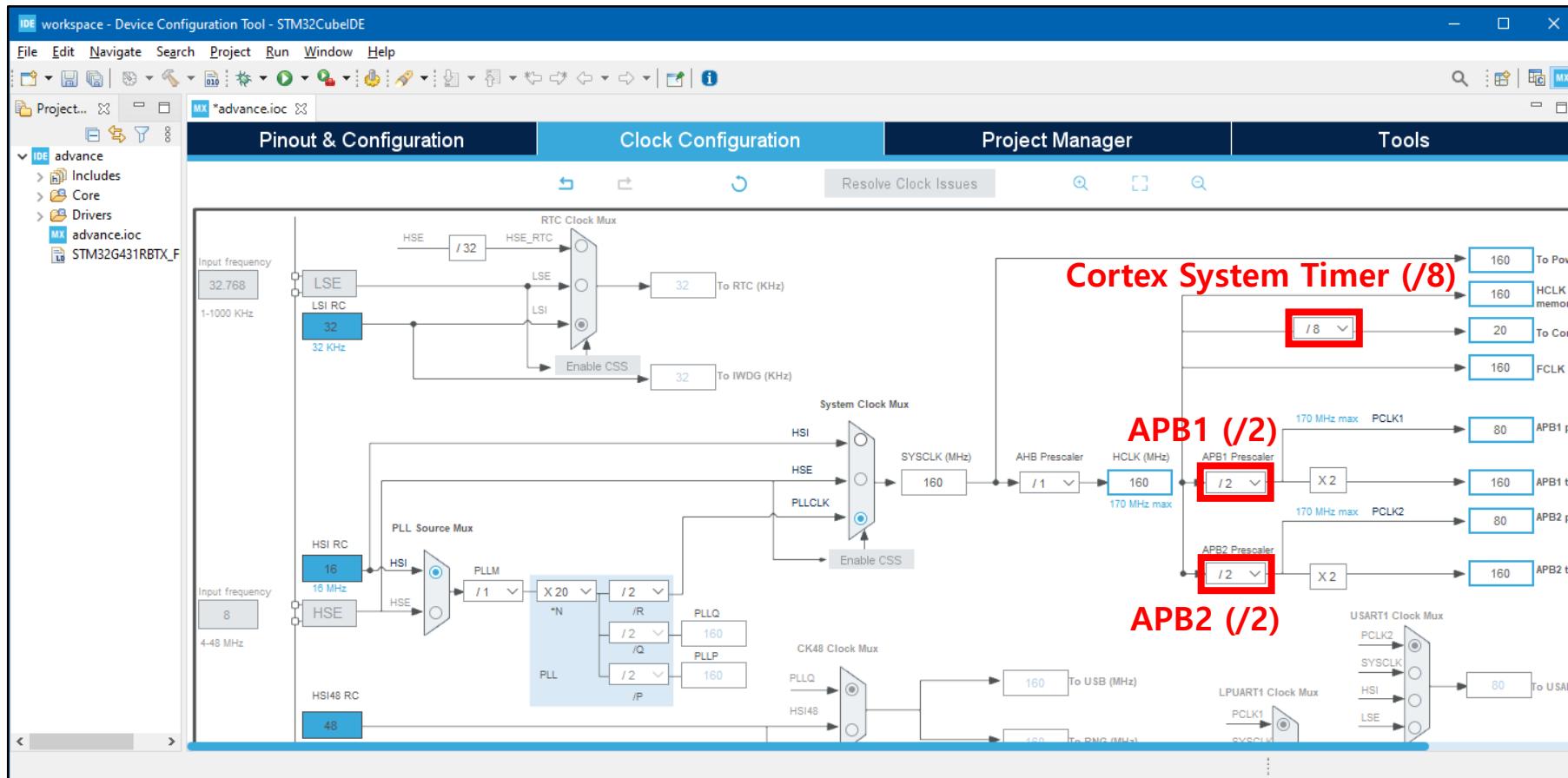
Clock Configuration

- SYSCLK 및 HCLK을 160MHz 설정 (PLL 사용)



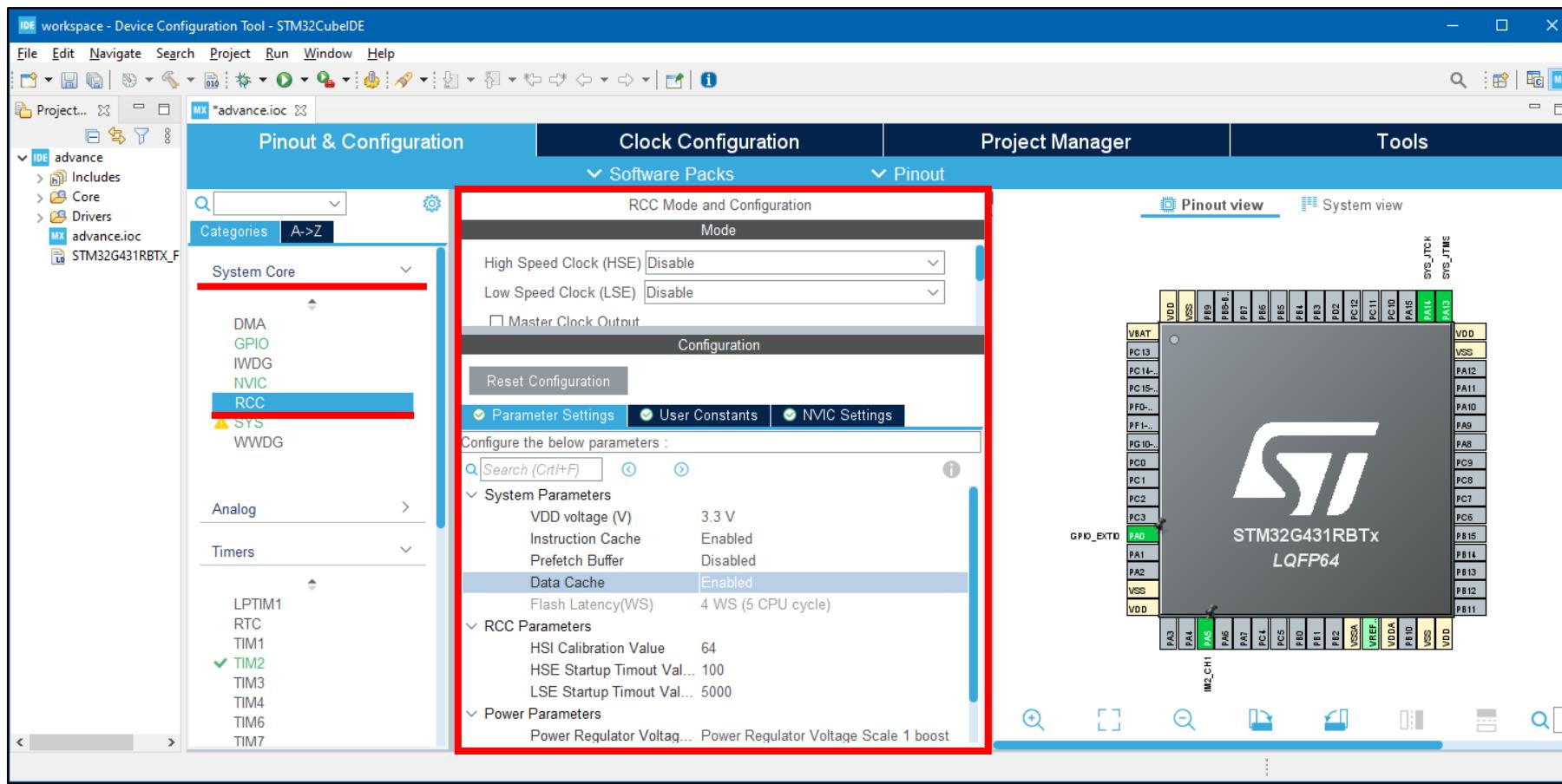
Clock Configuration

- Prescaler configuration



Clock Configuration

- STM32 MCU System, Middleware 및 Peripheral의 상세 설정.
 - System Core > RCC (Reset and Clock Control)

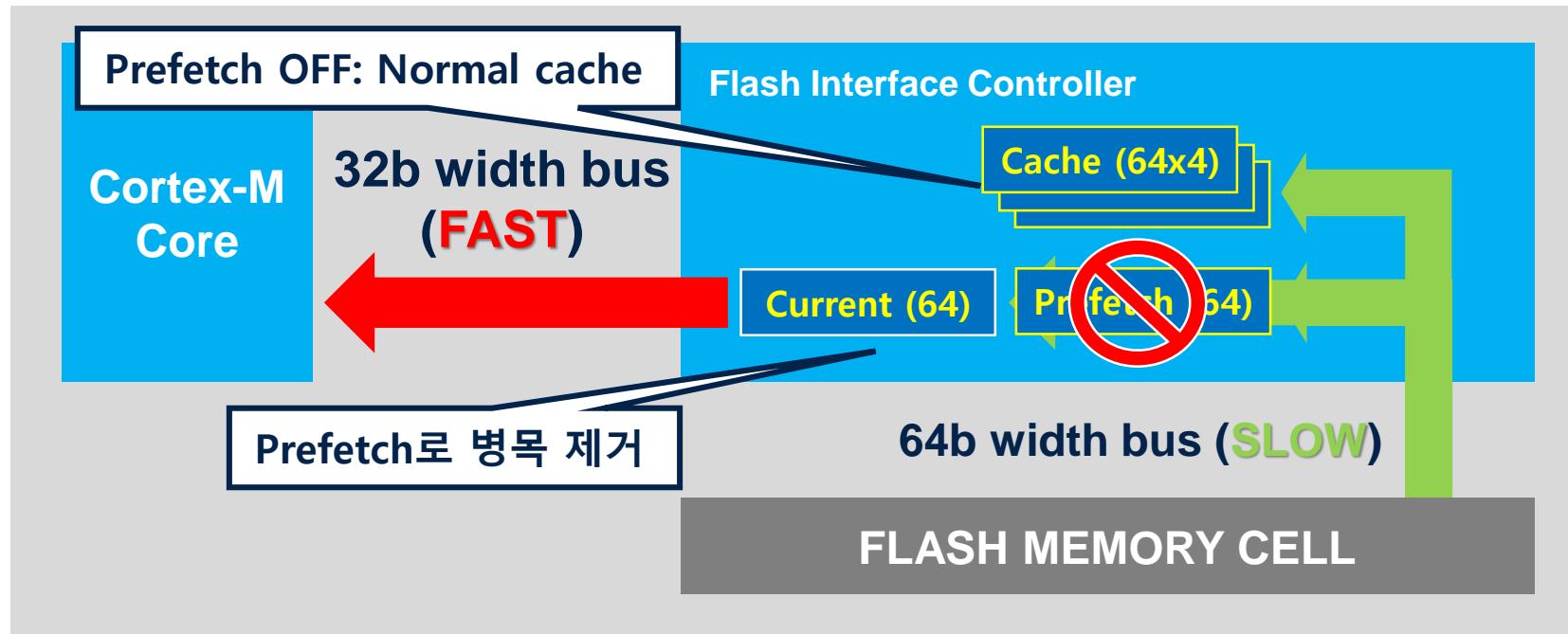


RCC Configuration

- RCC (Reset and Clock Control)
 - System Feature
 - VDD voltage (V) : 5.3 Operating Conditions @Datasheet (DS12589)
 - Internal Flash Memory Feature
 - ART accelerator : Flash memory와 Core간 병목 제거를 위한 cache system이며 3.3.4 Adaptive real-time accelerator @ Reference Manual (RM0440) 참조
 - Instruction prefetch / Instruction & Data cache
 - Flash Latency (Wait-State) : Bus access 속도와 Flash memory 의 동기화를 위한 대기 시간이며 3.3.3 Read access latency @ Ref. Manual (RM0440) 참조
 - Clock Feature
 - HSI Calibration : 부팅 시점에 xCAL(공정 값) 및 xTRIM(사용자 옵셋 값)으로 해당 clock 보정, HSI16TRIM의 기본값은 64: 7.4.2 RCC_ICSCR @ Ref. Manual (RM0440) 참조

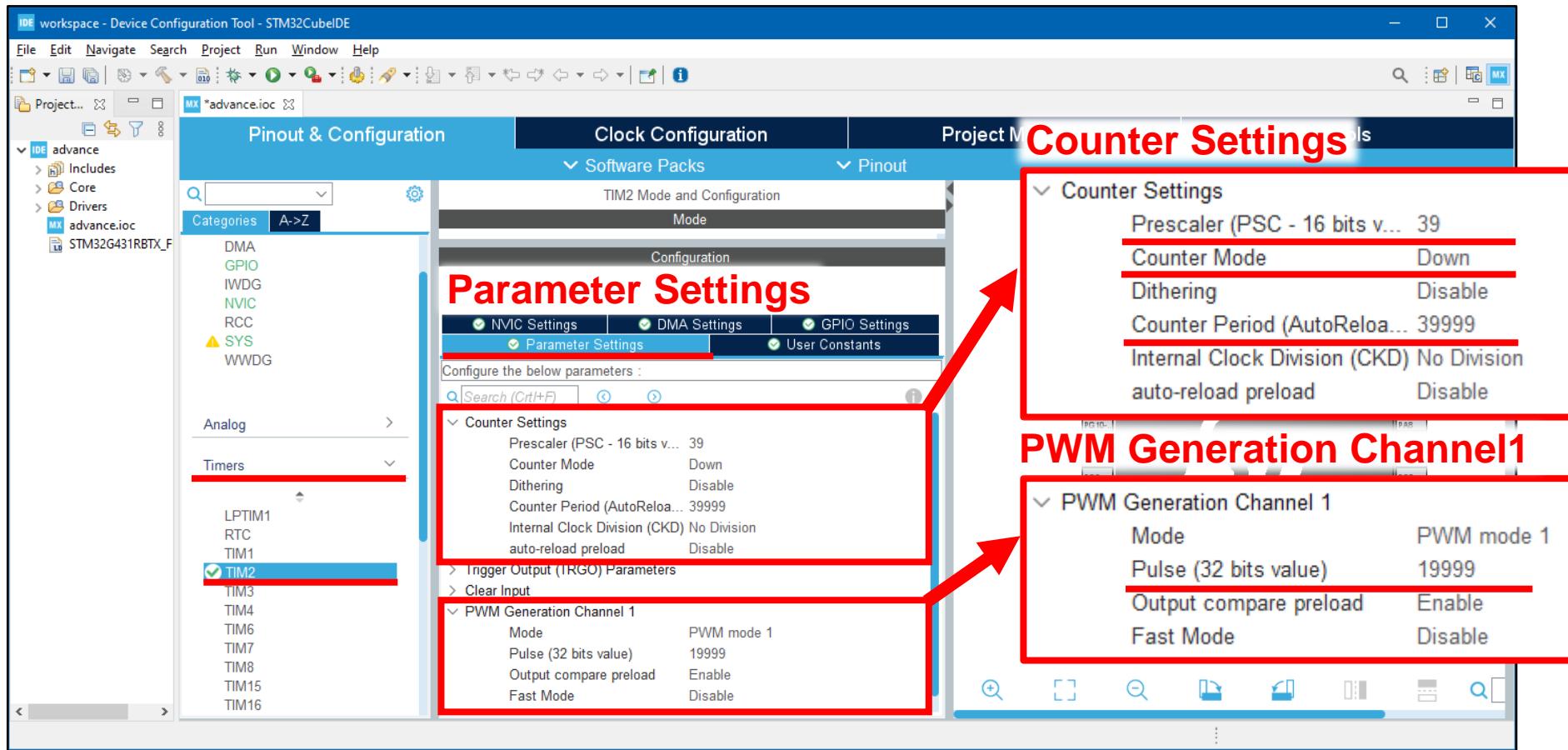
RCC Configuration

- STM32G4 ART Accelerator
 - Instruction cache(32-line x 256(64x4) bit) / Data cache(8-line x 256(64x4)bit)
 - Instruction Prefetch buffer
 - 3.3.4 ART Accelerator™ @ Ref. Manual (RM0440)
※ Default: Prefetch OFF and I/D cache ON



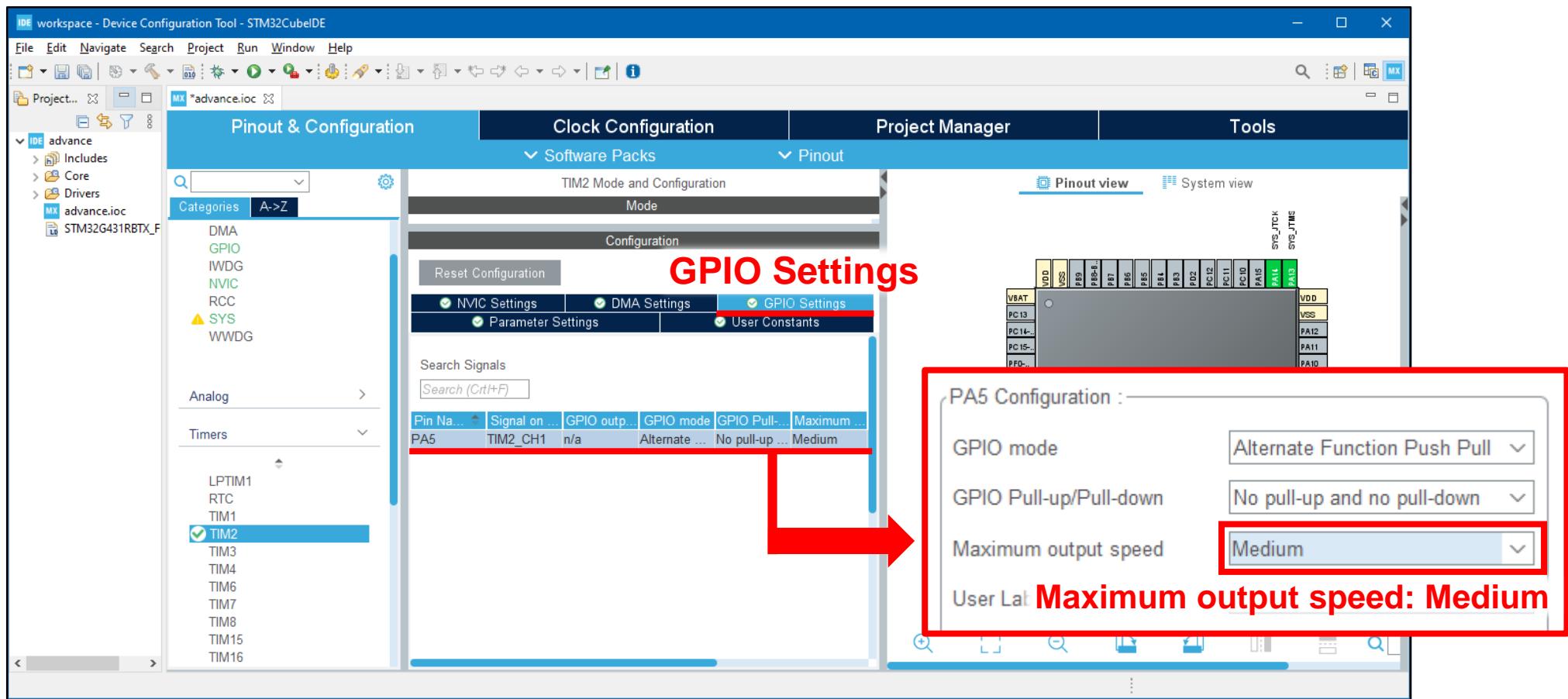
Timer Configuration

- Timers > TIM2 > Configuration: Parameter Setting
 - Prescaler: 39, Counter Mode: Down ,Counter Period: 39999 ,Pulse: 19999



Timer Configuration

- Timers > TIM2 > Configuration: GPIO Settings
 - Maximum output speed: Medium



Timer Configuration

- TIM2 설정
 - 10msec PWM 주기 계산 (1초의 100개의 PWM 파형 출력)

1) Timer 공급 클럭: 4MHz (1 clock 주기: 250ns)

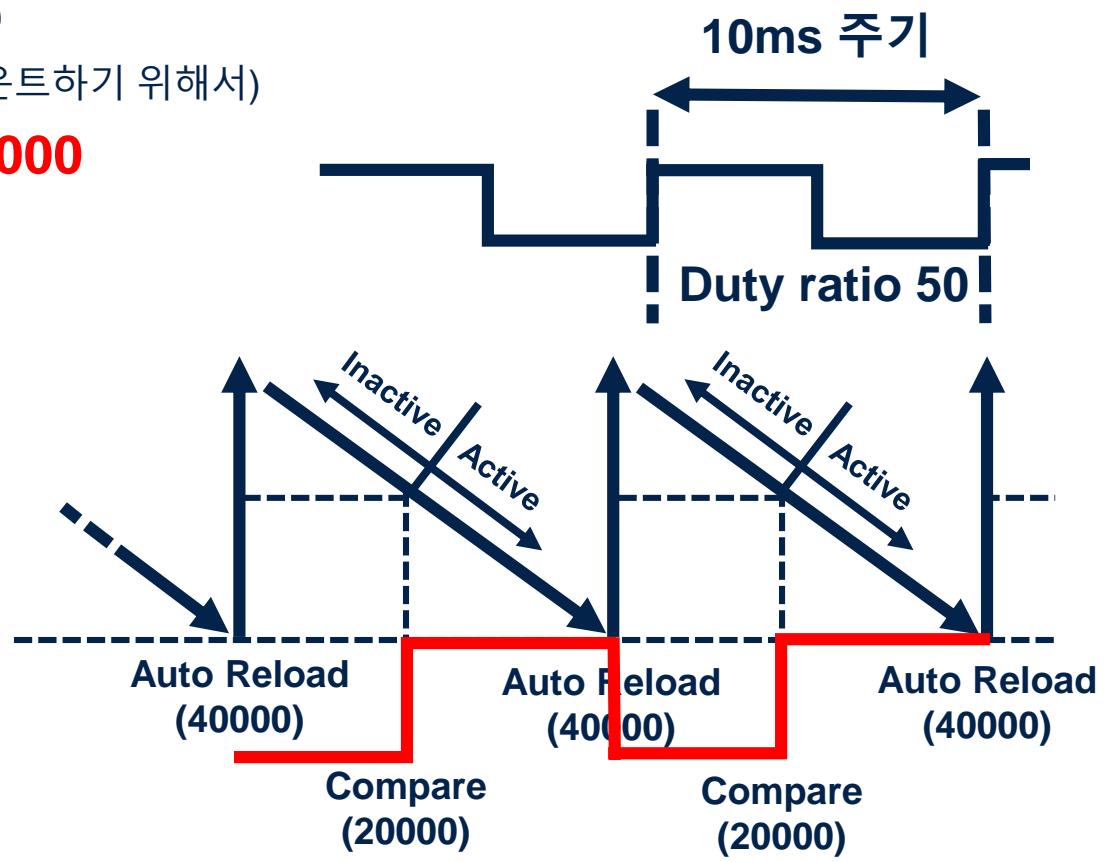
※ Prescaler로 160MHz를 40분주 (16bit timer로 10ms 범위 카운트하기 위해서)

2) 10msec당 clock 개수: $10,000,000 / 250 = 40000$

3) 기본 duty 비를 50으로 하여 **20000**

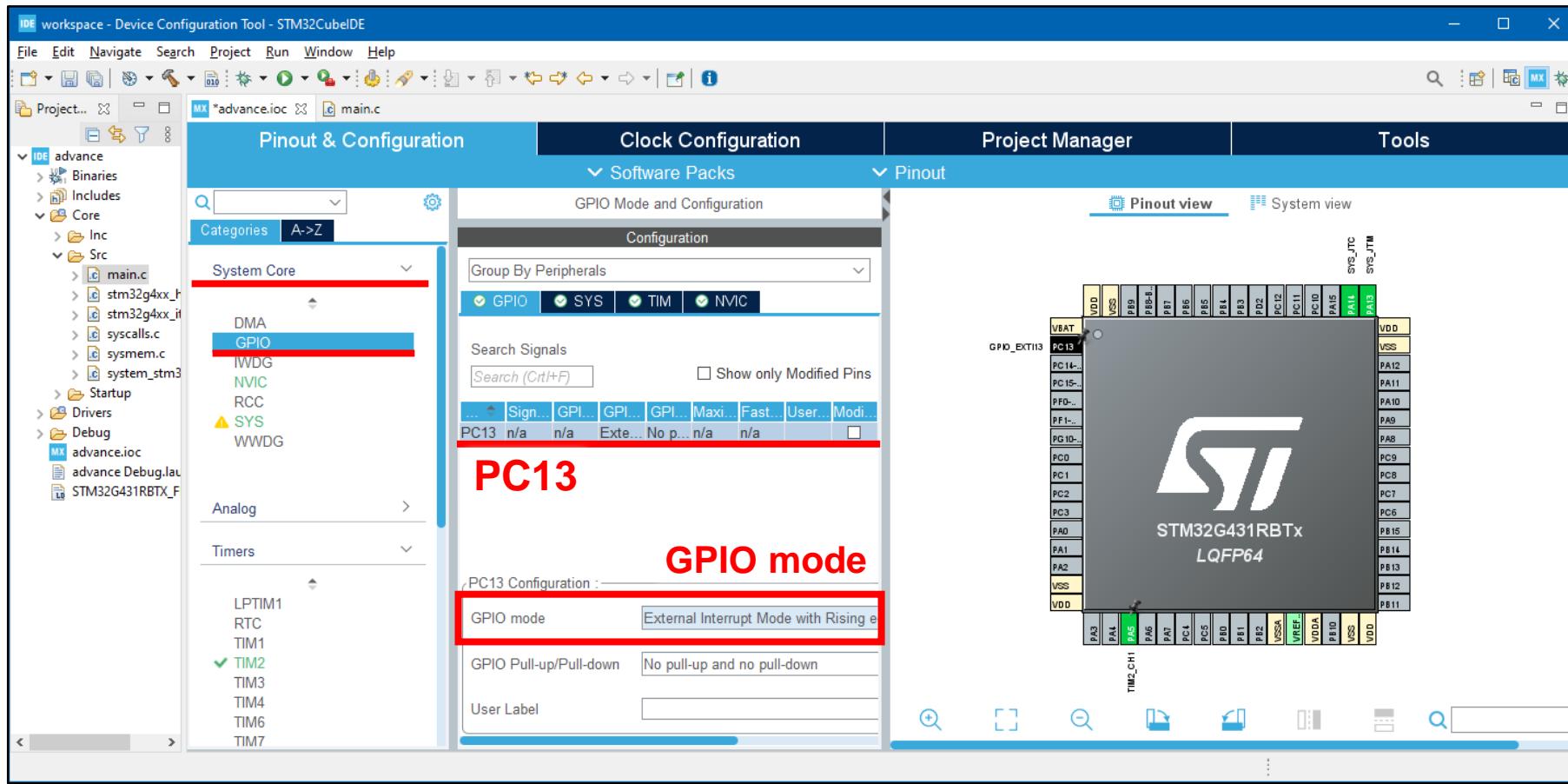
- PWM 출력

- Down counter
- PWM mode 1: (In downcounting)
 - CNT > CCR : inactive (low)
 - CNT <= CCR: active (high)



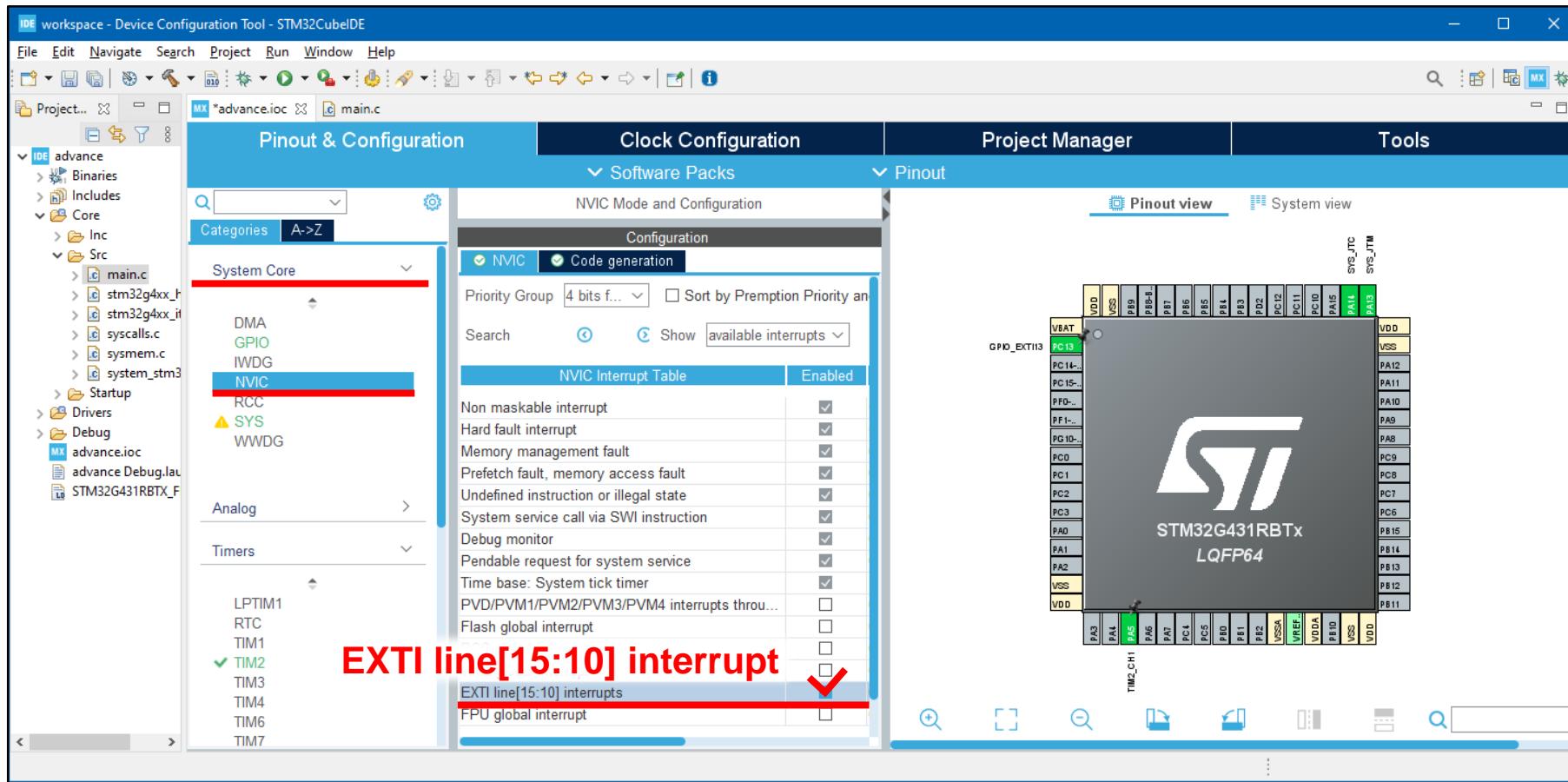
EXTI Configuration

- System Core > GPIO
 - PC13 > GPIO mode: External interrupt Mode with Rising edge trigger detection



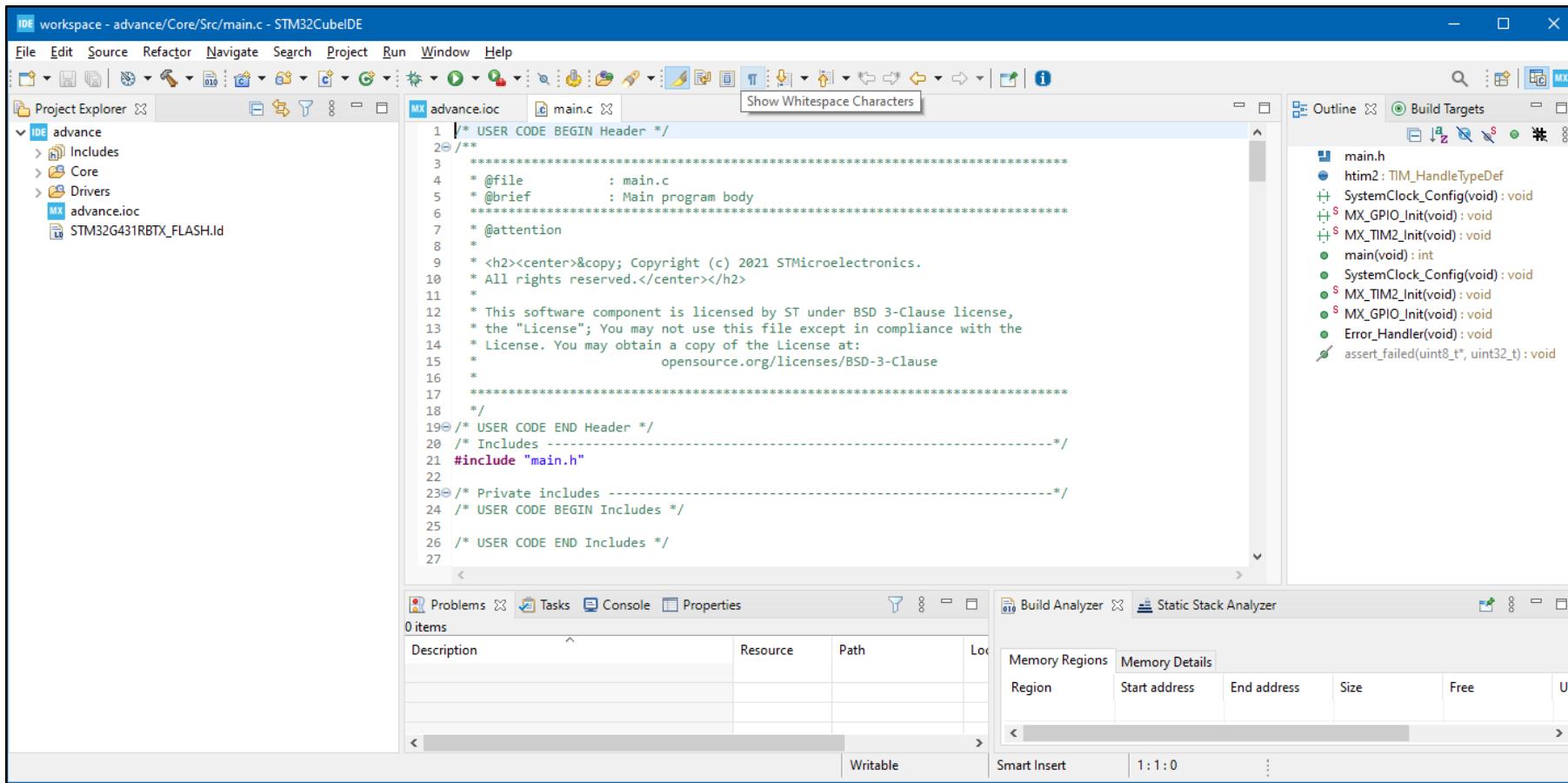
EXTI Configuration

- System Core > NVIC
 - **EXTI line[15:10] interrupt: Enabled 체크**



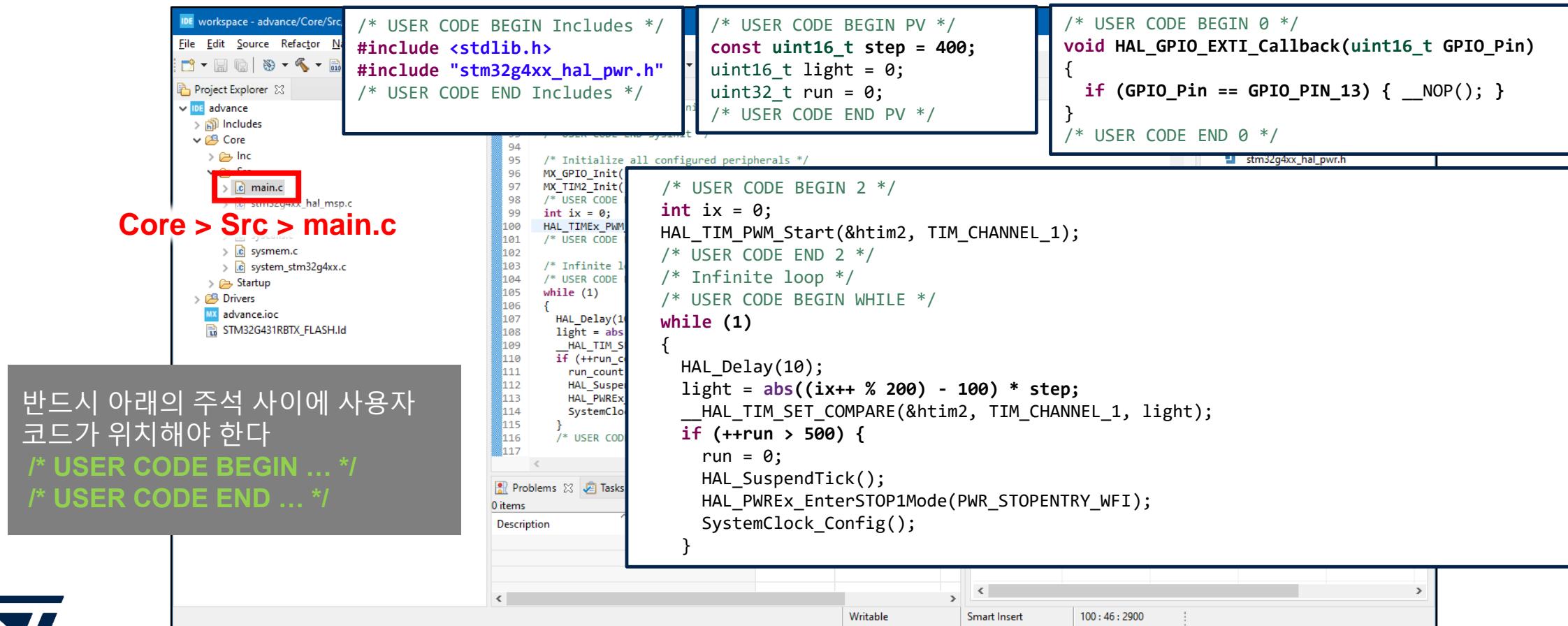
Generate Source Code

- Project > Generate Code or Save
 - 생성된 STM32CubeIDE Project



Generate Source Code

- TIM Compare Register 값을 2초 동안 10ms 단위(PWM 주기)로 제어
 - PWM duty 조절을 통해 LED 밝기를 제어 (Duty 값은 삼각파 공식으로 생성)



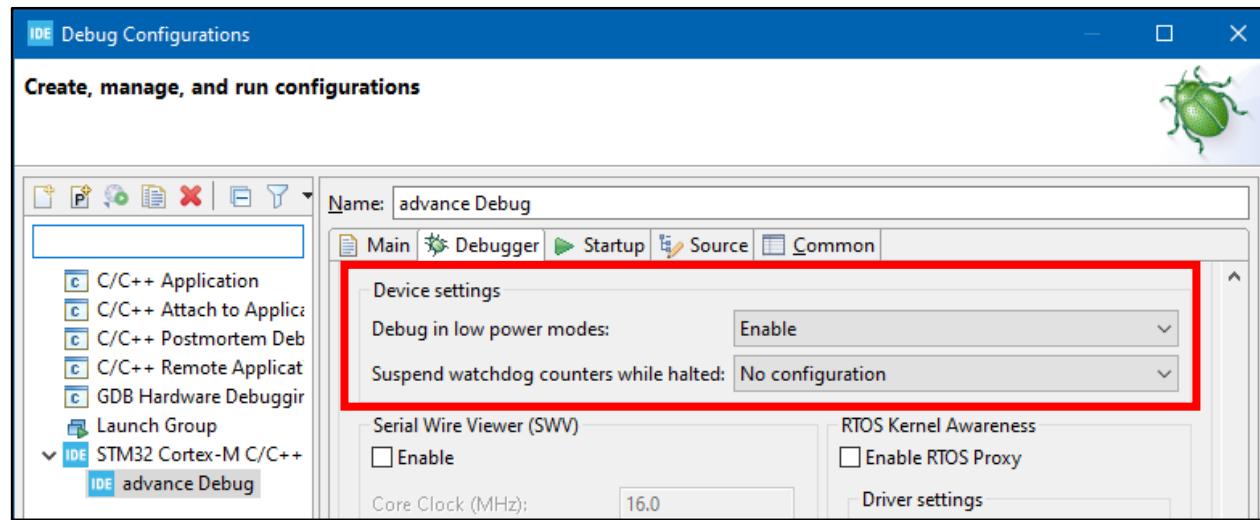
Build and Target Download

- Build (**Ctrl + B**) 
- Target(Discovery board)을 Host(PC)와 USB로 연결
- Start/Stop Debug Session (**F11**) 
- Run (**F8**) 
- Terminate (**Ctrl + F2**) 

- 1) Target board의 LD2(**Green LED**)가 점멸 되는지 확인
- 2) 약 5초 후에 점멸을 멈추고 STOP 모드에 들어가는지 확인
- 3) 사용자 버튼을 눌렀을 때 다시 LD2 (**Green LED**) 가 점멸을 시작하는지 확인

Appendix: 저전력 모드 진입

- Sleep, Stop, Standby, Shutdown 등 저전력 모드로 진입할 때 유의사항
 - HAL_PWREx_EnterSLEEPxxx, HAL_PWREx_EnterSTOPxxx 등이 호출되는 시점에 인터럽트(이벤트) pending 플래그가 세팅 되어 있으면 Sleep, Stop 모드 등에 진입하지 못하고 wakeup 된다
 - 저전력 모드에서도 디버그 장비 연결을 유지하기 위해 HAL_DBGMCU_EnableDBGSleepMode(), HAL_DBGMCU_EnableDBGStopMode() 와 같은 함수 호출하는 코드를 넣을 수 있다
 - Debug configuration 의 Debug in low power mode 를 enable 하는 것과 동일한 기능



Appendix: 저전력 모드 진입

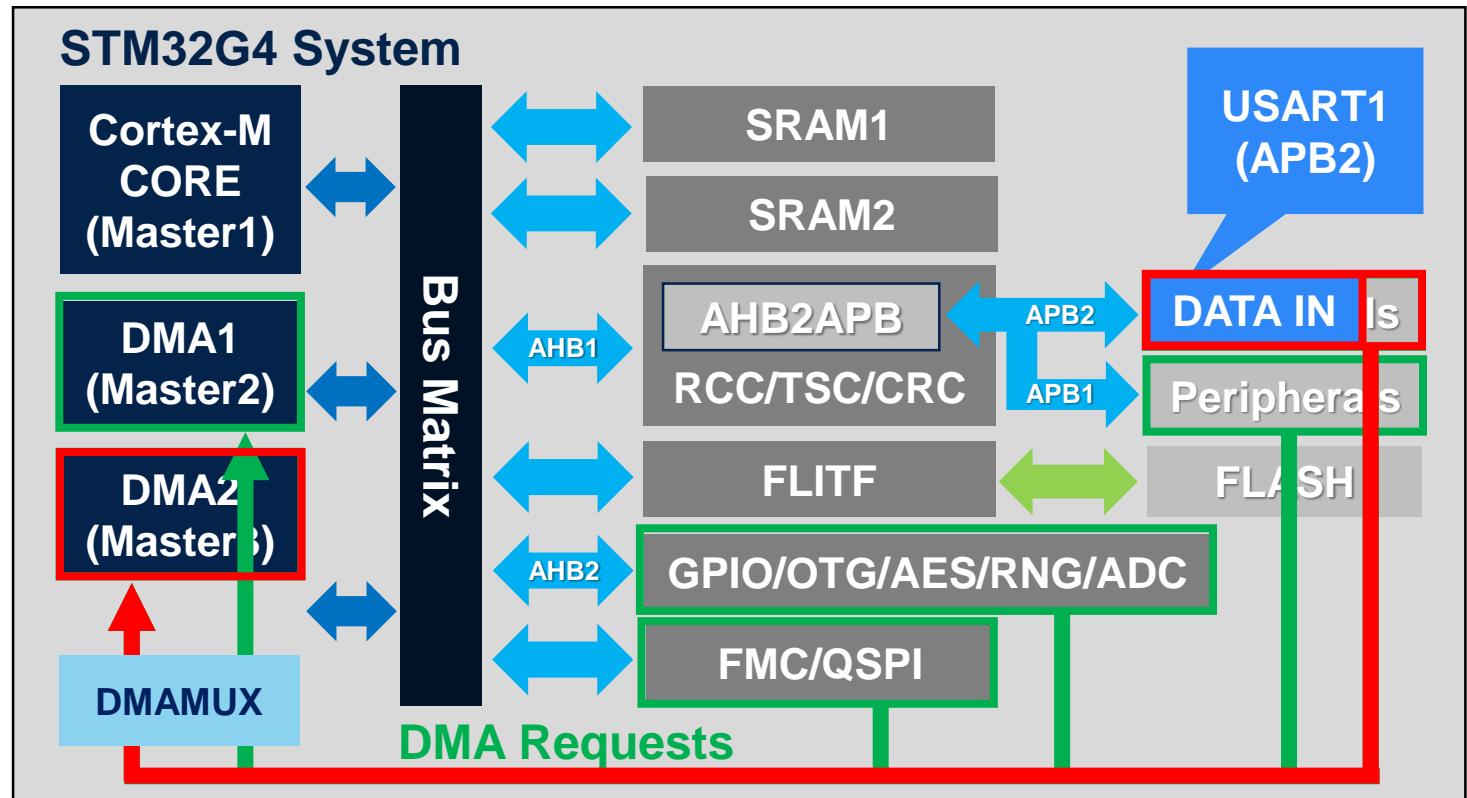
- Debug in low power mode 가 enable 되어 있거나 HAL_DBGMCU_Enablexxx 가 호출된 경우 Cortex 코어의 클럭이 켜지고 코어 내부 타이머인 Systick 인터럽트도 켜진 채로 유지된다
 - 디버거 연결은 유지하는 대신 소모 전류가 증가
 - 디버거 연결은 유지하는 대신 저전력 모드에 들어갈때 Systick 인터럽트 플래그가 pending 상태가 되어 있어서 저전력 모드에 진입할 수 없는 경우 발생하므로 진입전 HAL_SuspendTick(); 호출 필요

DMA (Direct Memory Access)



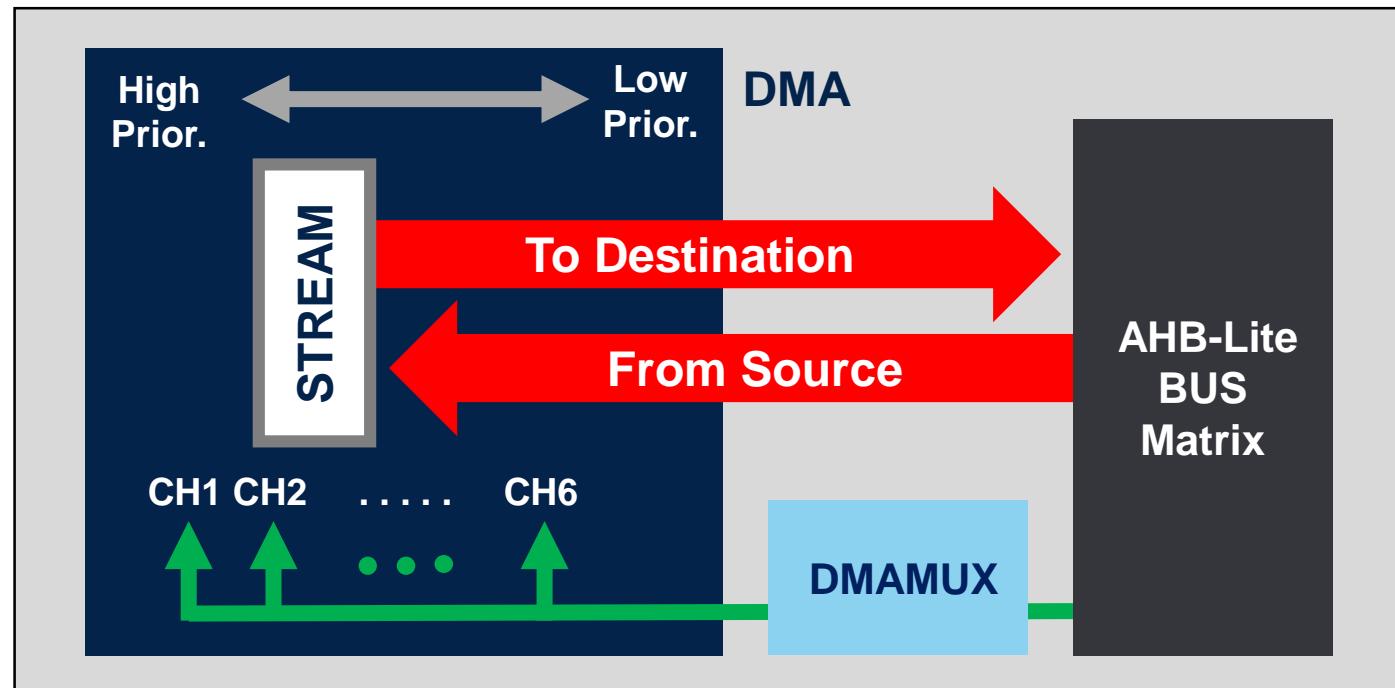
DMA Feature

- DMA (Direct Memory Access)
 - CPU에 대해 독립적으로 동작하는 bus slave간 데이터 전송 기능을 가지는 장치
 - Request(Trigger)를 통해 **@Source**의 데이터를 **@Destination**로 지정된 회수(**NDT: Number of Data to Transfer**)만큼 전송

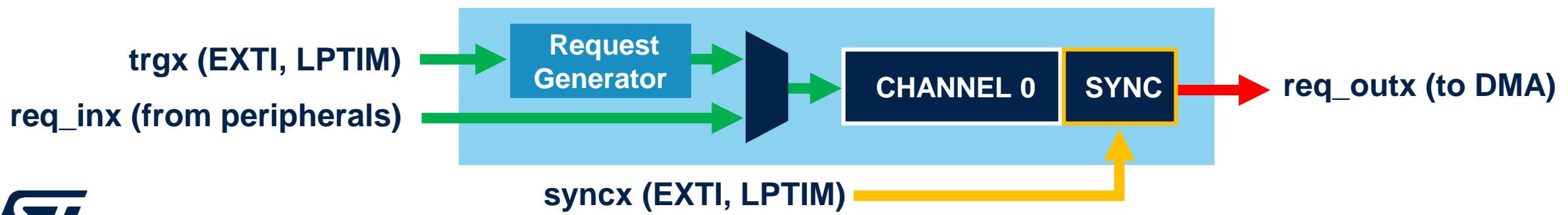


- DMA 구조

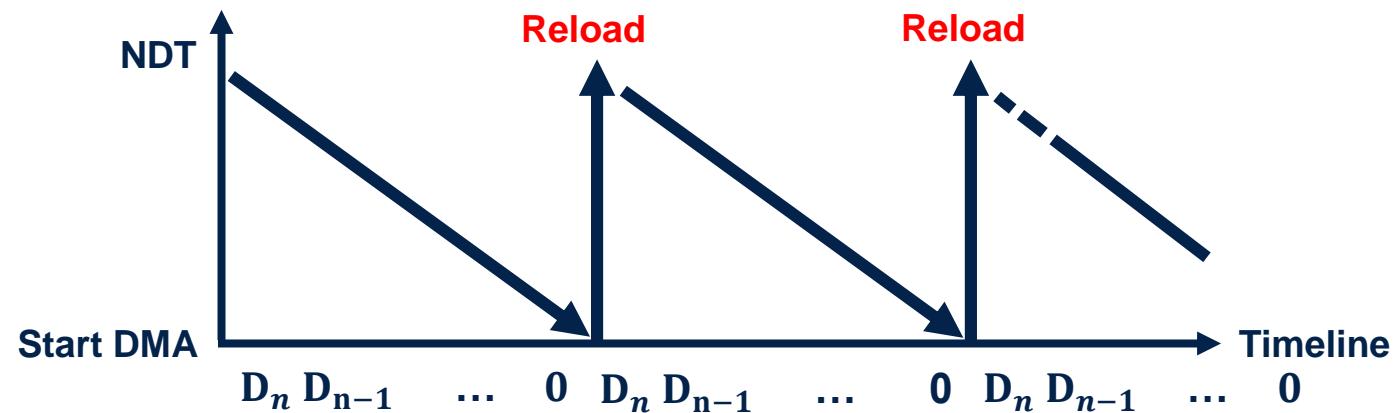
- 6개의 Channel은 DMAMUX를 통해 Request Source (Bus slave)와 연결될 수 있다.
 - 13.3.2 DMAMUX mapping @ Ref. Manual (RM0440)
- DMA Data Stream은 우선순위에 따른 Channel Request에 맞게 Transaction을 수행.
 - Channel 소프트웨어 우선순위는 프로그램 가능 (4단계)
 - 소프트웨어 우선순위가 동일할 때는 번호가 낮은 채널이 우선순위가 높다



- 12-channel의 프로그램 가능한 DMA request를 설정 가능
 - DMAMUX CH[0:5]는 DMA1 CH [1:6]에 연결, DMAMUX CH[6:11]는 DMA2 CH[1:6]에 연결
- Channel 별로 최대 128개의 DMA request 입력 지원
 - DMA request generator[0:3]는 EXTIx 및 LPTIM1_OUT 입력 소스
 - 나머지는 peripheral로부터의 입력 소스
- Channel의 출력은 DMAx channel에 연결 가능
 - event 출력, feedback 가능
 - 출력 동기화(synchronization) 기능: EXTIx 및 LPTIM1_OUT 입력 소스



- DMA Operation
 - **Direction:** Memory To/From Peripheral / Memory To Memory
 - **DMA Mode:** Normal Mode / **Circular Mode:** DMA 전송 횟수를 나타내는 NDT 값이 0이 될 경우 최초 설정 값으로 자동으로 reload 되고 source 주소와 destination 주소도 increment 설정이 된 경우 최초 주소 값으로 복귀된다
 - **DMA Interrupt source**
 - TC (Transmission complete)
 - HT (Half-transfer complete)

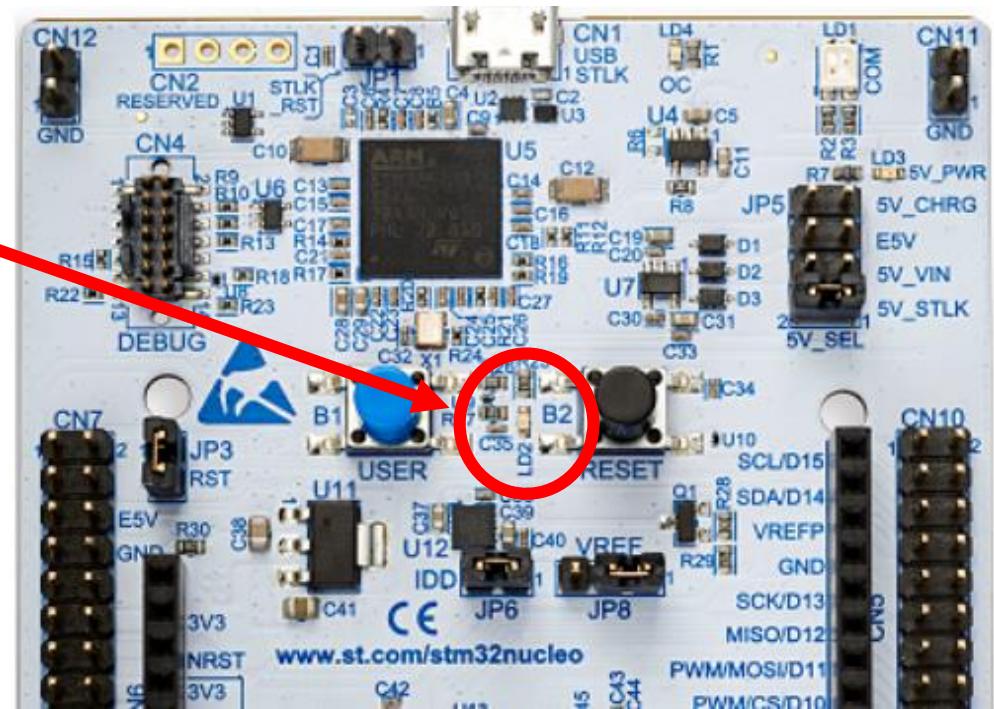
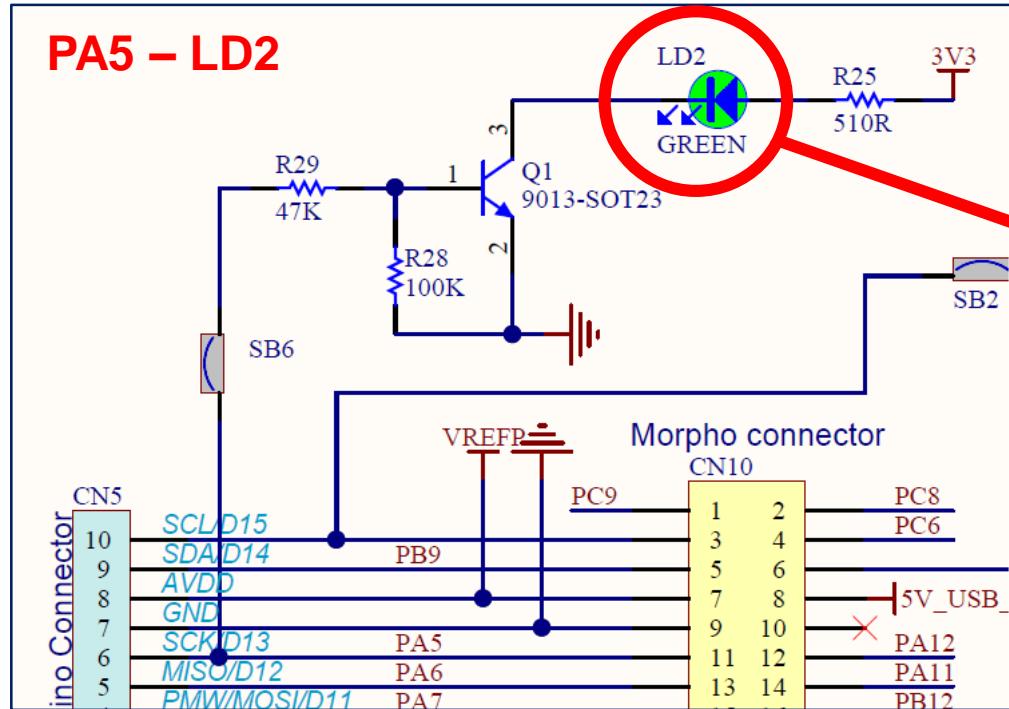


※ NDT register는 down counting 동작하며 0가 되면 전송이 중지

LED Dimming Project

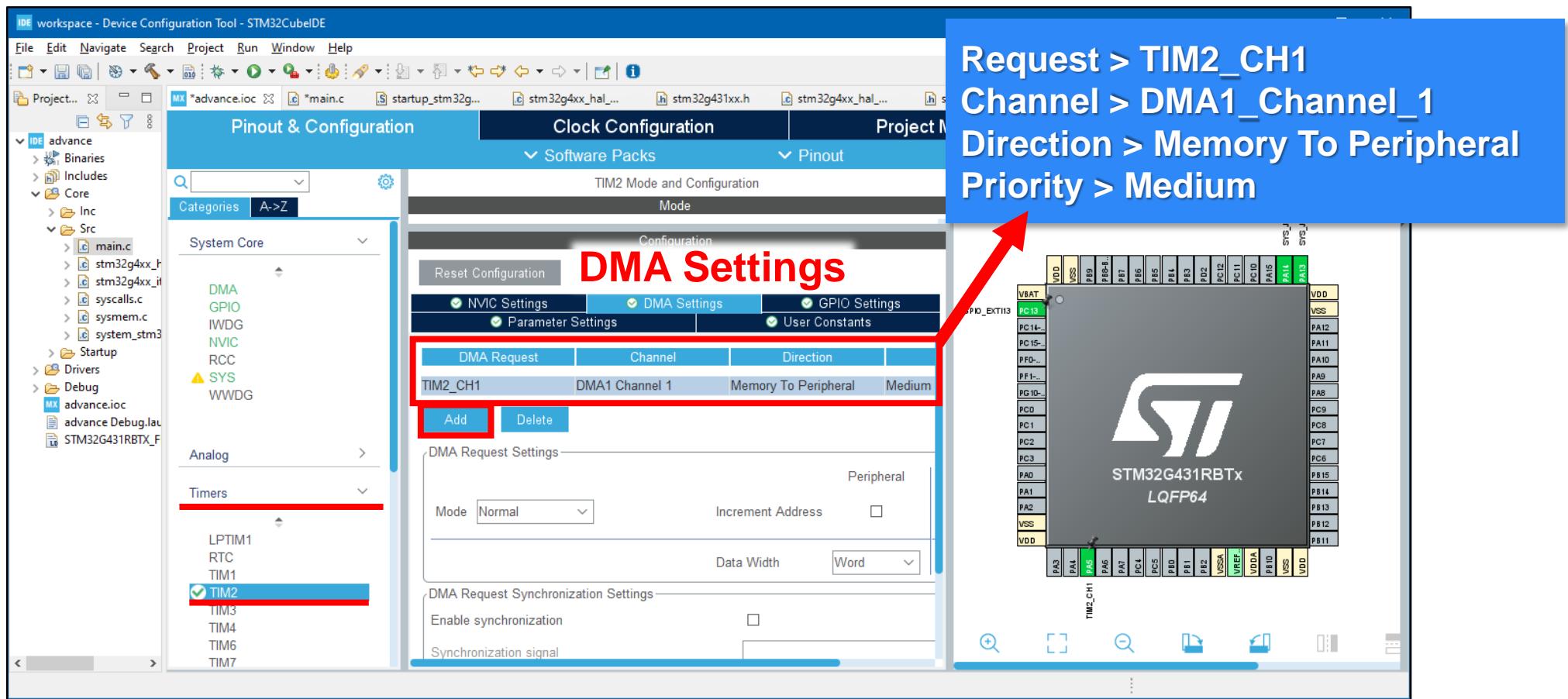
- LED (LD2: Green LED) Dimming Project를 DMA 기반 코드로 변경하여 동일 기능을 구현 (Timer2 CH1 PWM (w/ DMA))

LD2 (Green LED) Dimming 제어



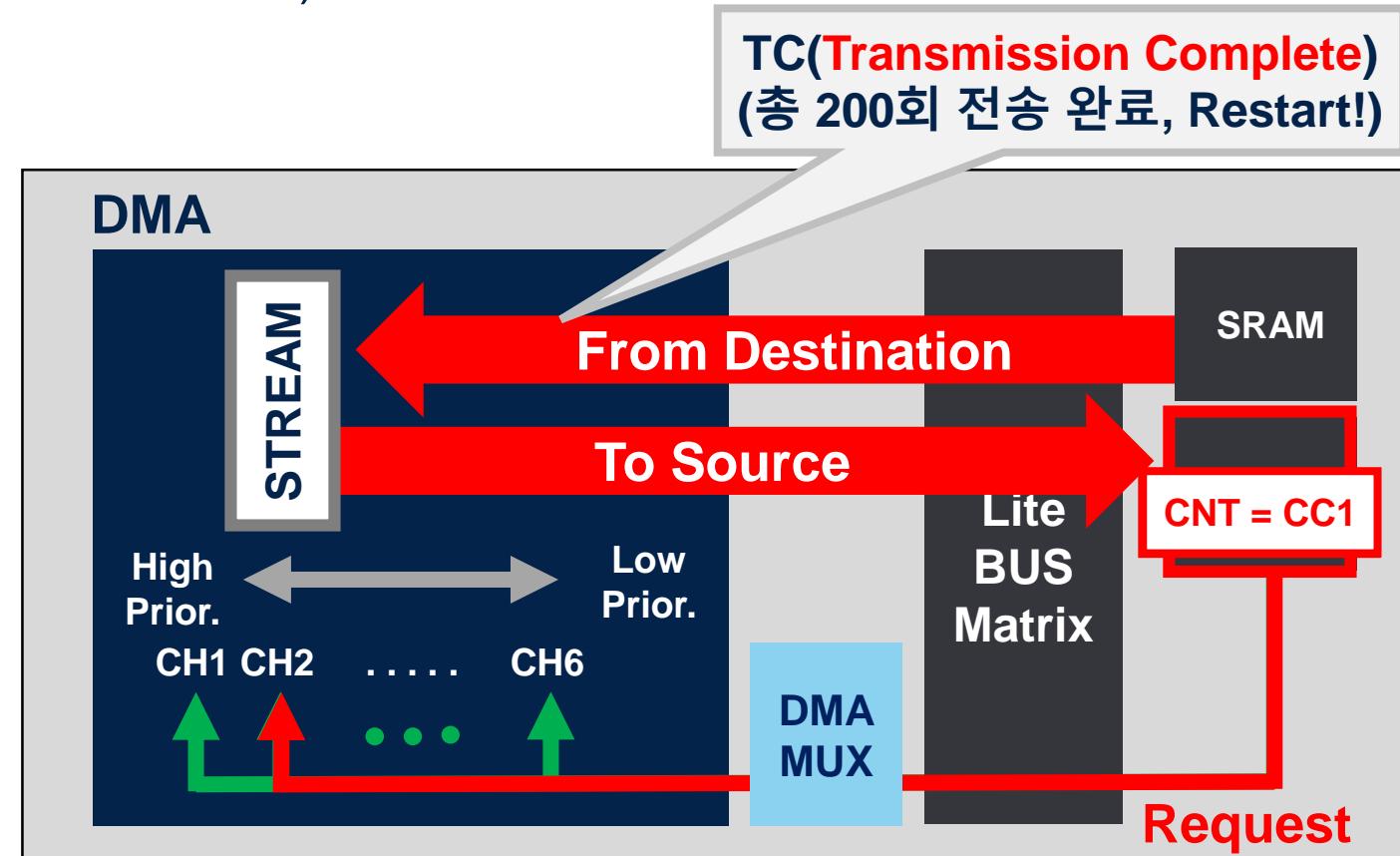
Configuration

- Timers > TIM2 > Configuration: DMA Settings
 - **TIM2_CH1, DMA1_Channel_1, Memory To Peripheral, Medium**



LED Dimming Code (Principle)

- DMA 설정 정보
 - DMA1 / Channel1 (Normal operation mode)
 - Memory To Peripheral
 - Medium Priority
 - Half word unit transfer



LED Dimming Code

- 기존 코드를 DMA 사용 형태로 변경 (**main() Infinite loop 내의 내용은 삭제**)

```
/* USER CODE BEGIN PV */  
const uint16_t step = 400;  
uint32_t light[200] = {0, };  
/* USER CODE END PV */
```

```
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)  
{  
    if (htim == &htim2)  
        HAL_TIM_PWM_Start_DMA(htim, TIM_CHANNEL_1, (uint32_t *)light, 200);  
}
```

```
/* USER CODE BEGIN 2 */  
for (int ix = 0; ix < 200; ++ix)  
    light[ix] = abs(ix - 100) * step;  
HAL_TIM_PWM_Start_DMA(&htim2, TIM_CHANNEL_1, (uint32_t *)light, 200);  
/* USER CODE END 2 */  
  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    /* USER CODE END WHILE */  
  
    /* USER CODE BEGIN 3 */  
}  
/* USER CODE END 3 */
```

Build and Target Download

- Build (**Ctrl + B**)
- Target(Discovery board)을 Host(PC)와 USB로 연결
- Start/Stop Debug Session (**F11**)
- Run (**F8**)
- Terminate (**Ctrl + F2**)

Target board의 LD2(**Green LED**)가 2초 주기로
서서히 밝아지고 서서히 어두워지는 형태로 점멸되는지 확인

Appendix: DMA ISR

- HAL Interrupt and User Callback Procedure

User code: **stm32g4xx_it.c** (low-level vector)

```
/**  
 * @brief This function handles DMA1 channel2  
 */  
void DMA1_Channel2_IRQHandler(void)  
{  
    /* USER CODE BEGIN DMA1_Channel2_IRQHandler_0 */  
  
    /* USER CODE END DMA1_Channel2_IRQHandler_0 */  
    HAL_DMA_IRQHandler(&hdma_tim1_ch1);  
    /* USER CODE BEGIN DMA1_Channel2_IRQHandler_1 */  
  
    /* USER CODE END DMA1_Channel2_IRQHandler_1 */  
}
```

```
void TIM_DMADelayPulseCplt(DMA_HandleTypeDef *hdma)  
{  
    TIM_HandleTypeDef* htim = (TIM_HandleTypeDef*)((DMA_HandleTypeDef*)htim);  
    ...  
  
    HAL_TIM_PWM_PulseFinishedCallback(htim);  
  
    htim->Channel = HAL_TIM_ACTIVE_CHANNEL_C1;  
}
```

HAL code: **stm32g4xx_hal_tim.c**

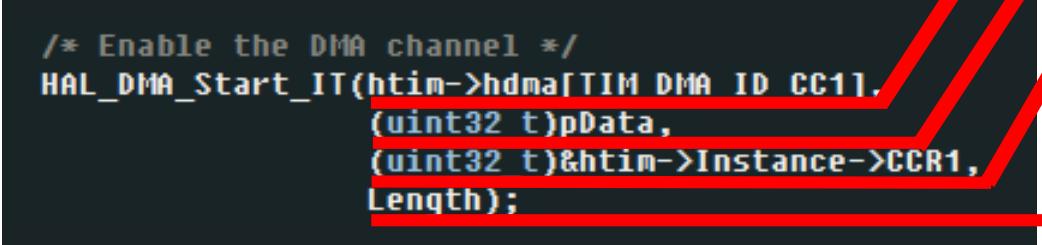
HAL code: **stm32g4xx_hal_dma.c**

```
void HAL_DMA_IRQHandler(DMA_HandleTypeDef *hdma)  
{  
    ...  
    /* Transfer Complete Interrupt management *****/  
    else if ((RESET != (flag_it & (DMA_FLAG_TC1 << hdma->  
    {  
        ...  
        if(hdma->XferCpltCallback != NULL)  
        {  
            /* Transfer complete callback */  
            hdma->XferCpltCallback(hdma);  
        }  
    }  
}
```

```
/* USER CODE BEGIN 0 */  
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)  
{  
    /* Prevent unused argument(s) compilation warning */  
    UNUSED(htim);  
}
```

User code

Appendix: DMA Configuration Code

- DMA Callback Allocation
 - DMA handle (hdma)는 XferCpltCallback 및 XferHalfCpltCallback 등의 전송 완료 callback 정보를 가진다.
 - DMA 기능을 제공하는 peripheral HAL driver API는 DMA에 대한 callback을 자체 설정한다. 따라서 **HAL DMA 설정 함수(HAL_TIMEx_PWMN_Start_DMA)** body 구현에서 callback에 대한 설정 정보를 확인해야 한다.
- DMA destination 및 source의 정보 확인
 - DMA 설정 함수: **HAL_DMA_Start_IT**

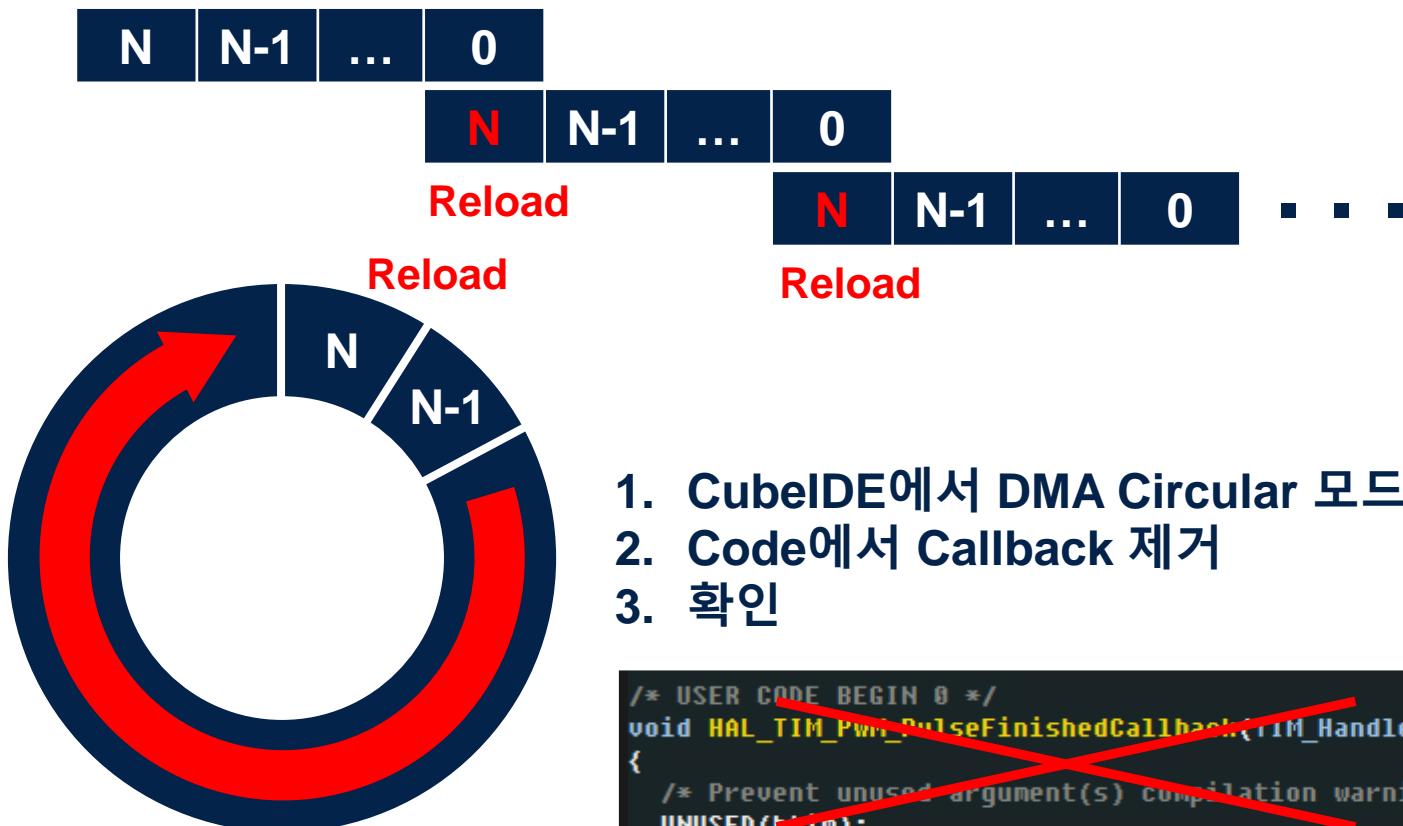
```
/* Enable the DMA channel */
HAL_DMA_Start_IT(htim->hdma[TIM DMA_ID_CC1],
                 (uint32_t)pData,
                 (uint32_t)&htim->Instance->CCR1,
                 Length);
```

 - Red arrows point to the following parameters:
 - @How(When) (TIM CC1) points to the first parameter: htim->hdma[TIM DMA_ID_CC1]
 - @Source (Memory) points to the second parameter: (uint32_t)pData
 - @Destination (CCR1) points to the third parameter: (uint32_t)&htim->Instance->CCR1
 - A red arrow points to the fourth parameter: Length (to NDT) points to the last parameter: Length;
- DMA 설정 함수 호출도 HAL DMA 설정 함수(**HAL_TIMEx_PWMN_Start_DMA**) body 구현에 포함되어 있다.

Appendix: DMA Circular Mode

- Circular Mode

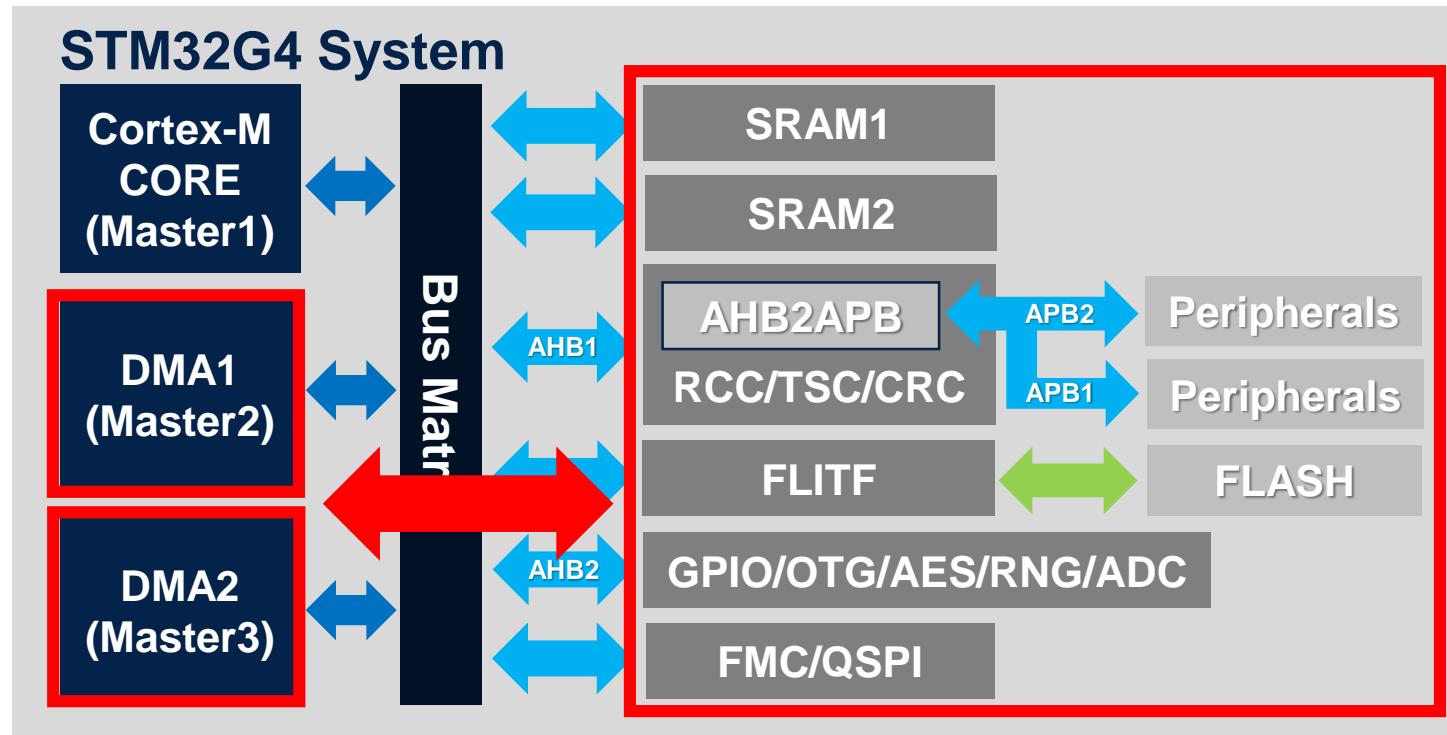
- NDT가 down-count되어 0이 되었을 때(TC) 최초 NDT 값과 최초 주소를 다시 로드하여 down-count 지속함으로서 circular 형태로 DMA transaction이 수행되는 모드.



Appendix: DMA for Low power mode

- Low power mode

- DMA는 bus master이므로 slave와 함께 독립적으로 운용 가능하며 이에 따라 상황에 따라 상대적으로 전력소비가 큰 Core를 OFF하여 전력 효율을 높일 수 있음
- Core는 모든 Interrupt 또는 Event에 의해 Wake-up 가능



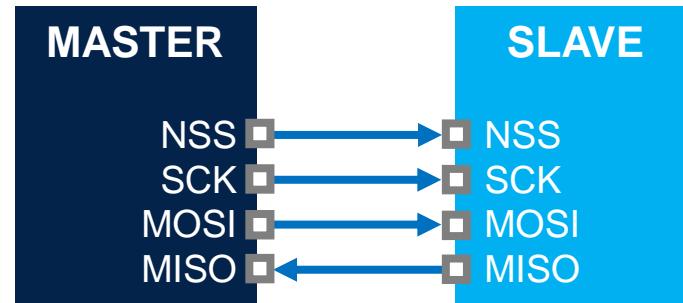
SPI (Serial peripheral interface)



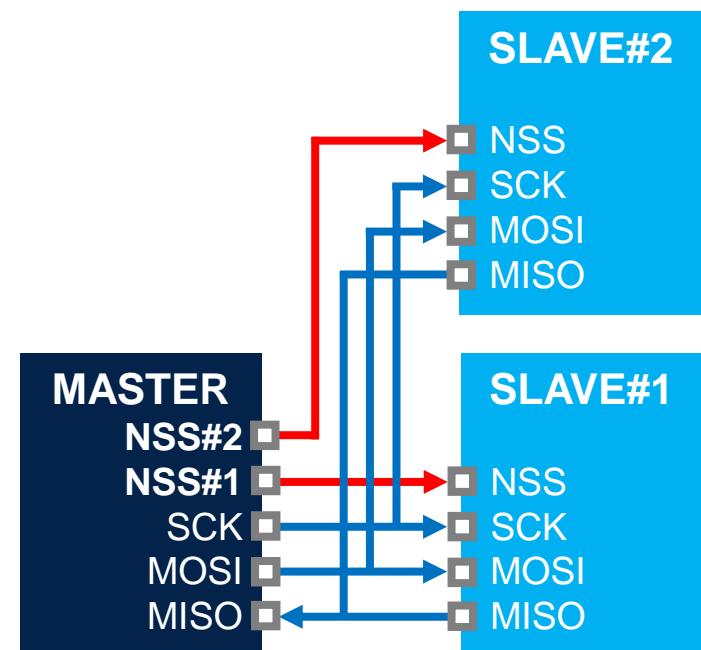
What is SPI

- SPI (Serial Peripheral Interface)

- Full-duplex 통신이 가능한 동기화 방식의 4핀 (데이터 2핀) 직렬 통신 규격
 - 병렬 통신 extension: Quad SPI, Octo SPI
- 신호선 구성
 - NSS : Slave Select 신호 (Low active chip select)
 - SCLK : Clock 신호
 - MOSI : Master Out Slave In 데이터 신호
 - MISO : Master In Slave Out 데이터 신호
- NSS, SCLK 은 Master가 출력



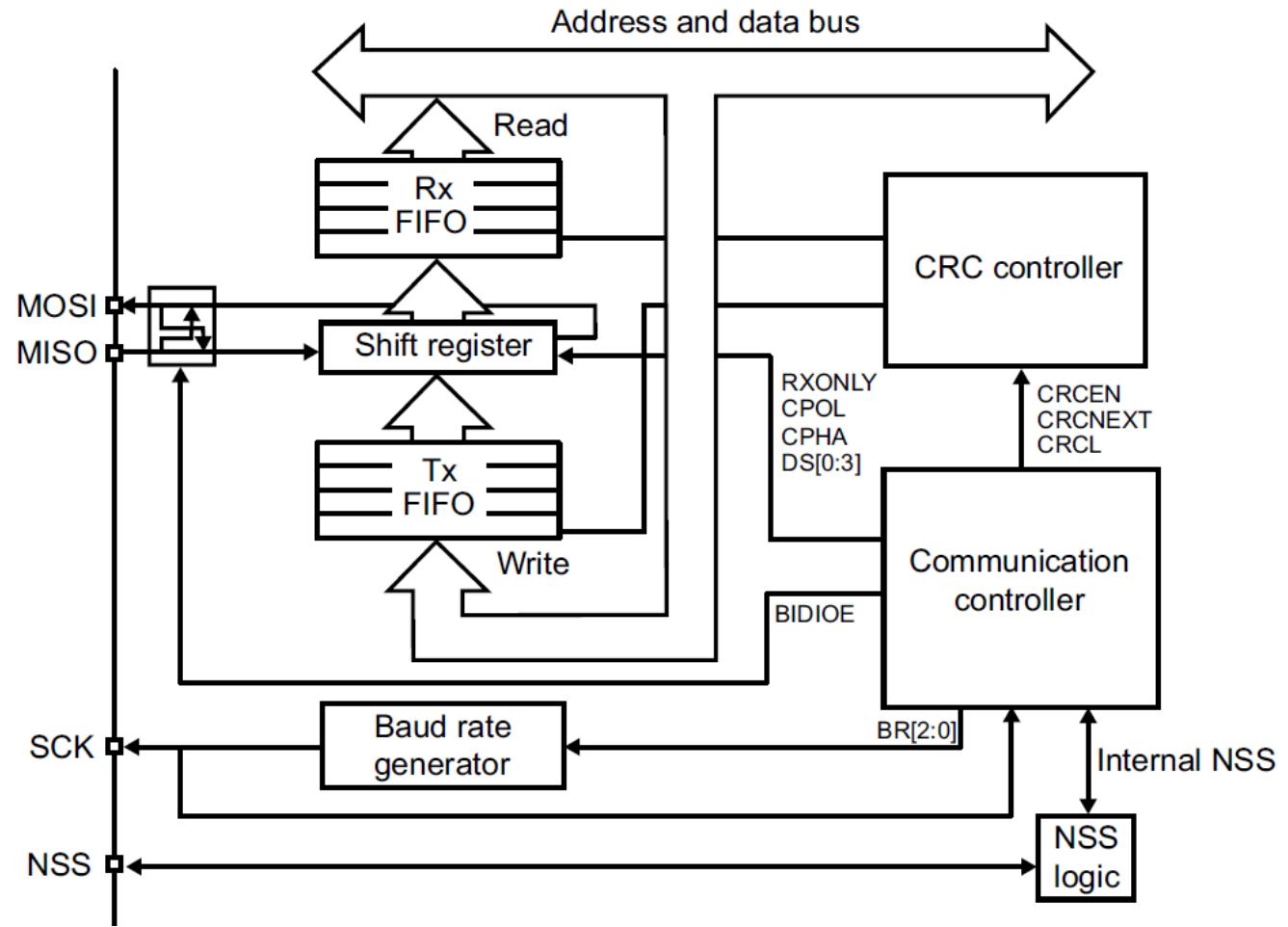
1 Master – 1 Slave



1 Master – Multi Slave

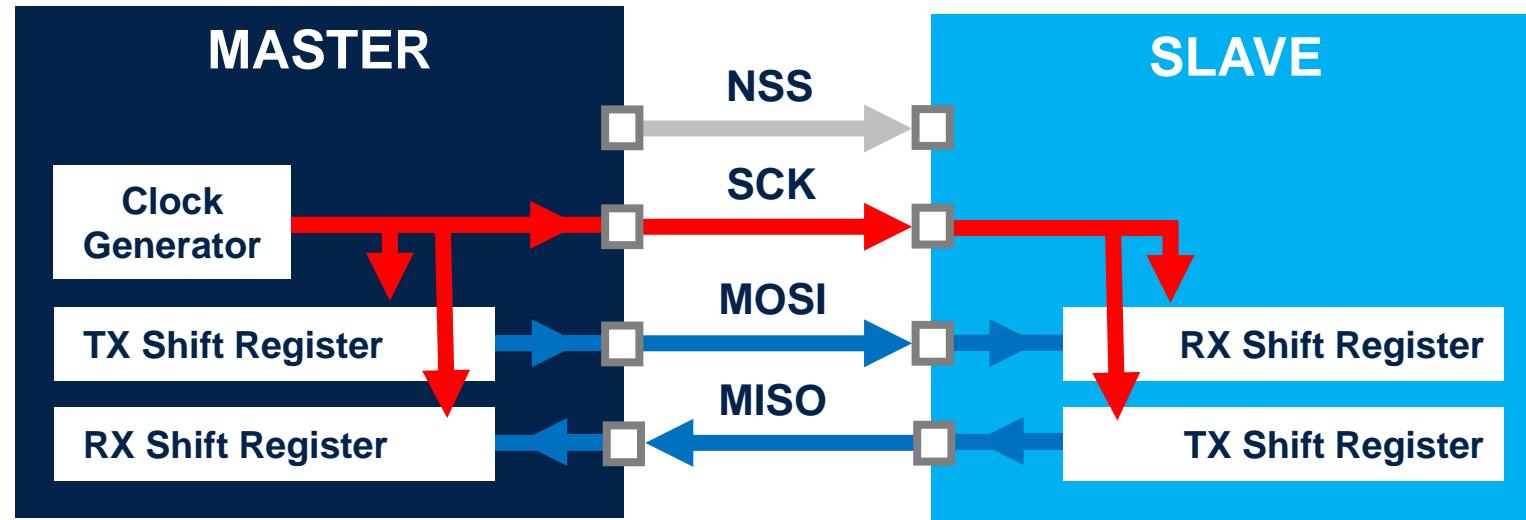
- Operating Modes
 - Master 또는 Slave로 설정 가능
 - Full-duplex, Simplex (Receive only, Transmit only), Half-duplex 로 설정 가능
 - Motorola (기본) 와 TI 표준 지원
 - Master 와 Slave 모드에서 Hardware 또는 Software로 NSS 관리
- 특징
 - 최대 $f_{PCLK}/2$ 의 SCLK 속도 지원 (/2,/4,/6,/8,/16,/32,/64,/128,/256 분주)
 - 최소 2-Wire 인터페이스 (SCLK, MOSI/MISO) 로 동작 가능
 - CPOL (Clock polarity), CPHA (Clock phase) 설정 변경 가능
 - SPI 내장 32bit TX 와 32 bit RX FIFO 지원
 - 하드웨어 CRC 지원

Block Diagram



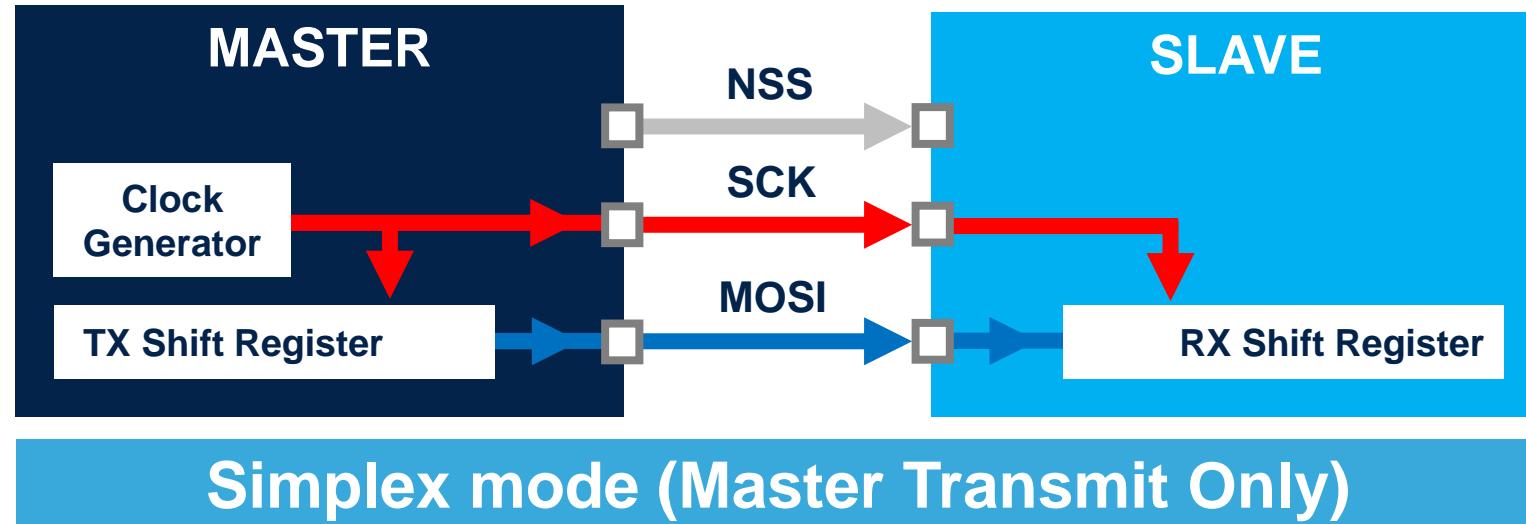
Interconnection of SPI nodes

- Full-Duplex mode (BIDIMODE=0)
 - 동일 클럭(SCK)에 Master와 Slave가 MOSI/MISO 핀으로 동시에 data 를 송수신.
 - Hardware NSS 핀 (또는 Software NSS) 으로 선택된 Slave 만 송수신에 참여한다



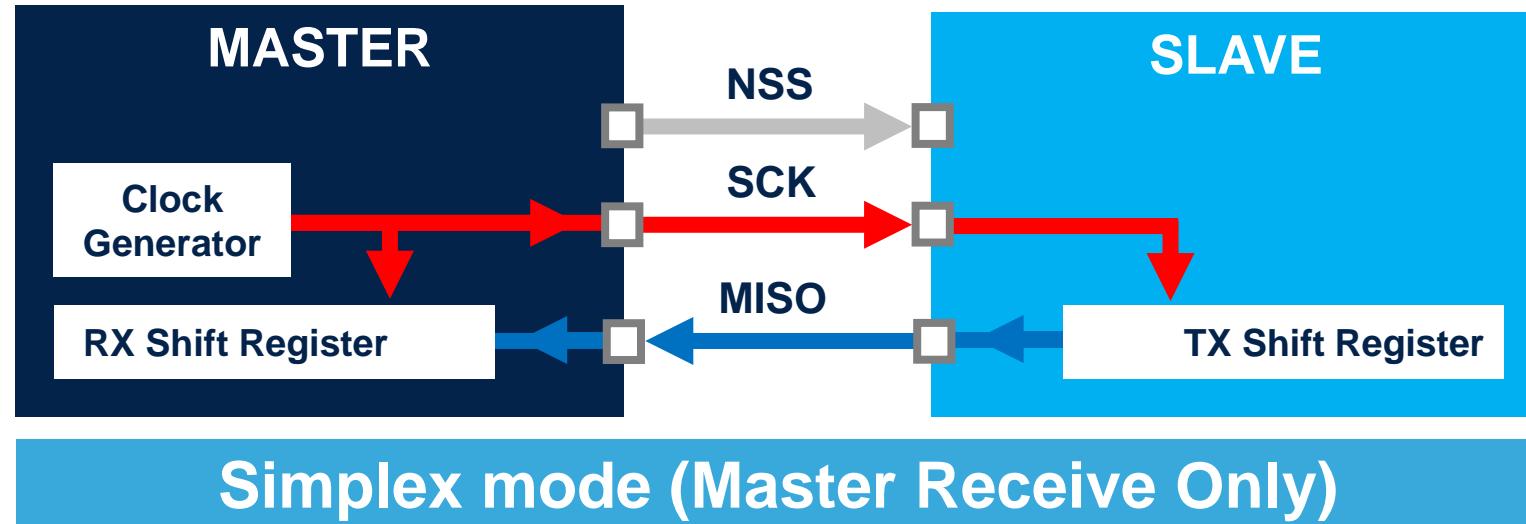
Interconnection of SPI nodes

- Simplex mode
 - 한 node는 transmitter로 다른 node는 오직 receiver로 동작
 - Transmit only ($\text{RXONLY}=0$) 또는 Receive only ($\text{RXONLY}=1$)
 - Hardware NSS 핀 (또는 Software NSS)으로 선택된 Slave 만 송수신에 참여한다



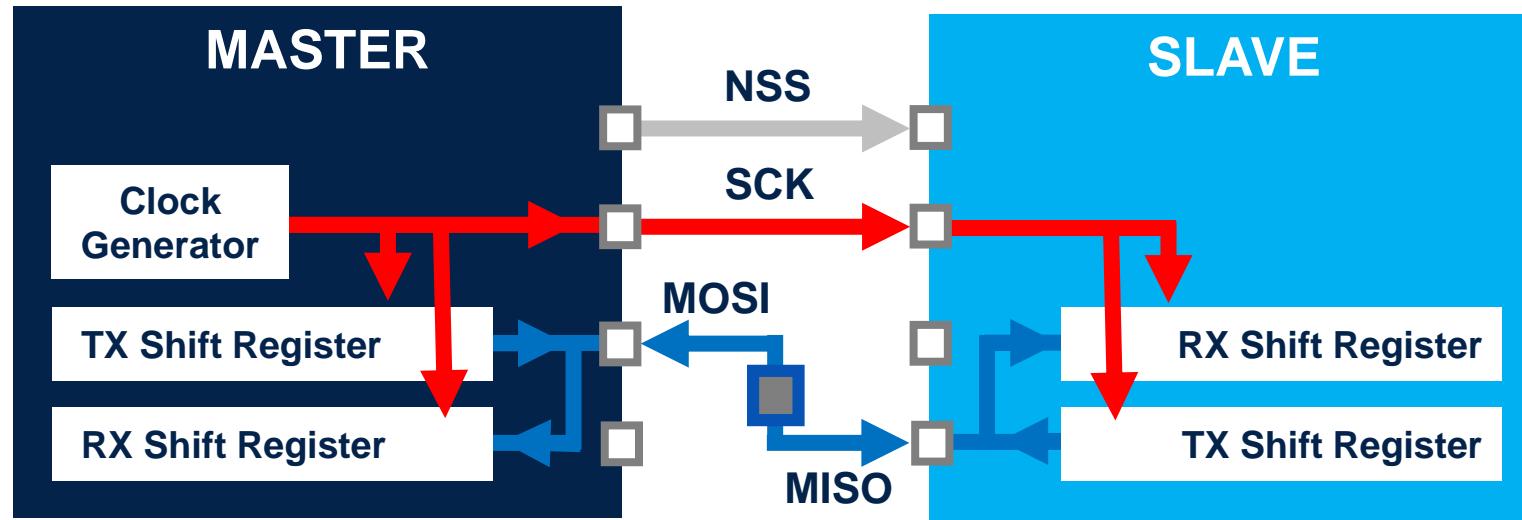
Interconnection of SPI nodes

- Simplex mode
 - 한 node는 transmitter로 다른 node는 오직 receiver로 동작
 - Transmit only (RXONLY=0) 또는 Receive only (RXONLY=1)
 - Hardware NSS 핀 (또는 Software NSS)으로 선택된 Slave 만 송수신에 참여한다



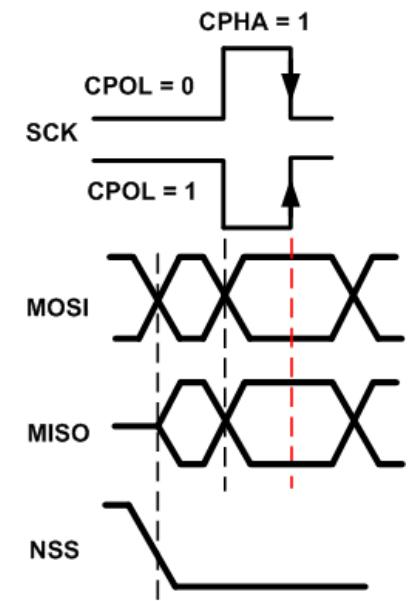
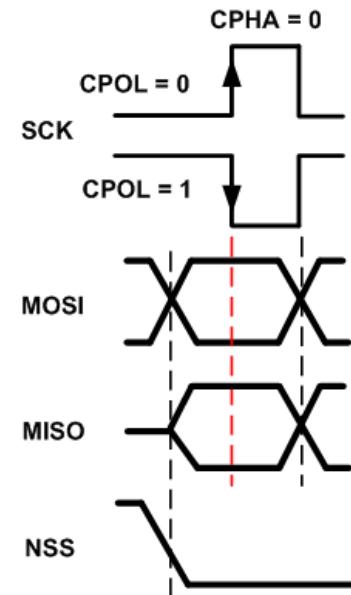
Interconnection of SPI nodes

- Half-duplex mode (BIDIMODE=1)
 - Master와 slave가 data 송신과 수신을 하나의 data frame 단위로 교대로 동작
 - Data 전송을 위해 단일 데이터 line 을 사용
 - Master 와 slave에서 입출력 전환이 동시에 변경되지 않을 수 있으므로 일시적인 short circuit 으로부터 핀 보호를 위해 일반적으로 data line에 저항을 추가한다
 - Hardware NSS 핀 (또는 Software NSS) 으로 선택된 Slave 만 송수신에 참여한다



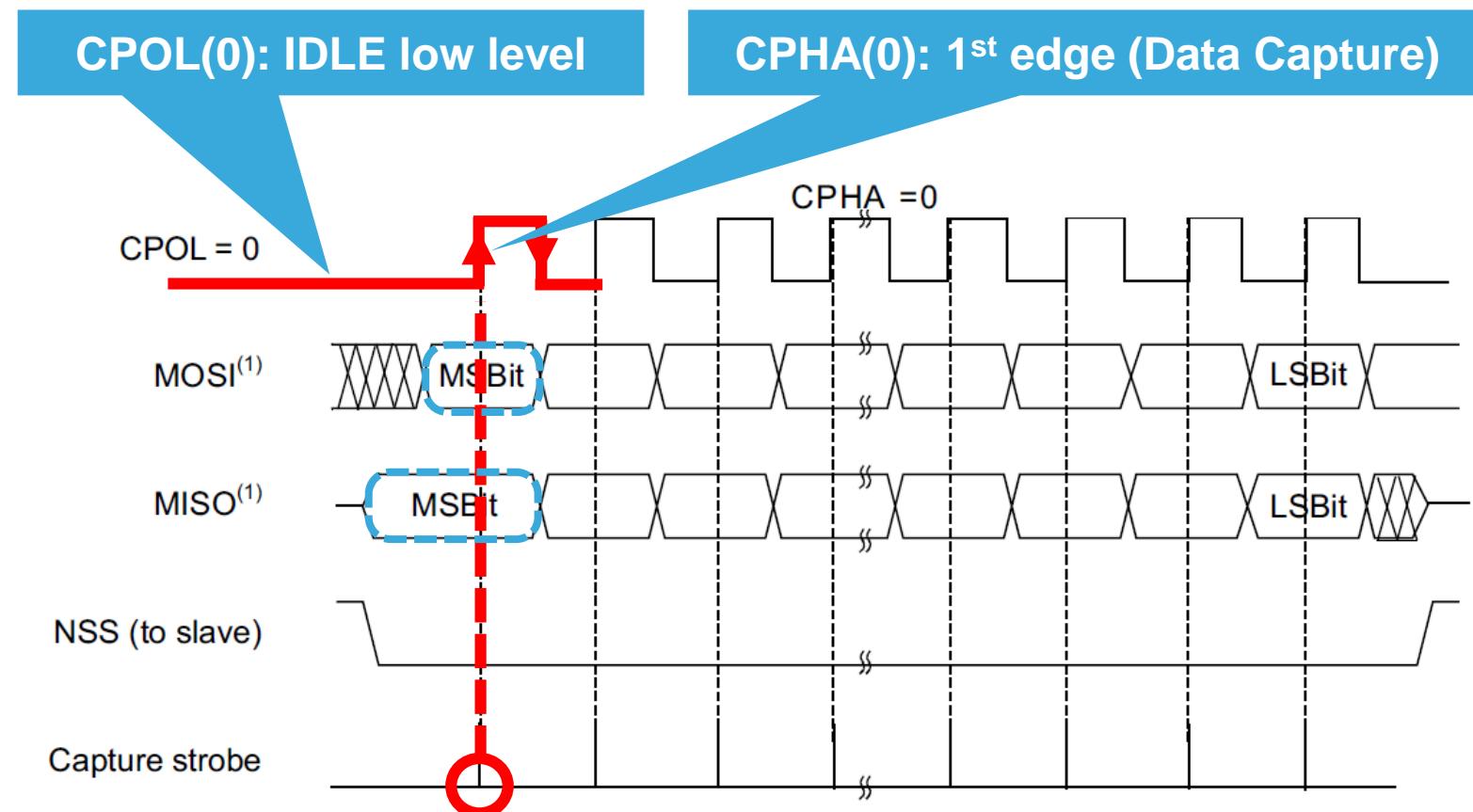
Data Frame Format

- 데이터 사이즈 및 전송 순서
 - 4-bit에서 16-bit 까지 설정 가능, MSB 또는 LSB first로 전송 순서 설정 가능
- Clock 설정
 - CPOL (Clock Polarity): Idle 상태의 clock level을 정의 (low or high)
 - CPHA (Clock Phase): Data capture 시점을 정의
 - 첫번째(odd: leading) 또는 두번째(even: trailing) edge



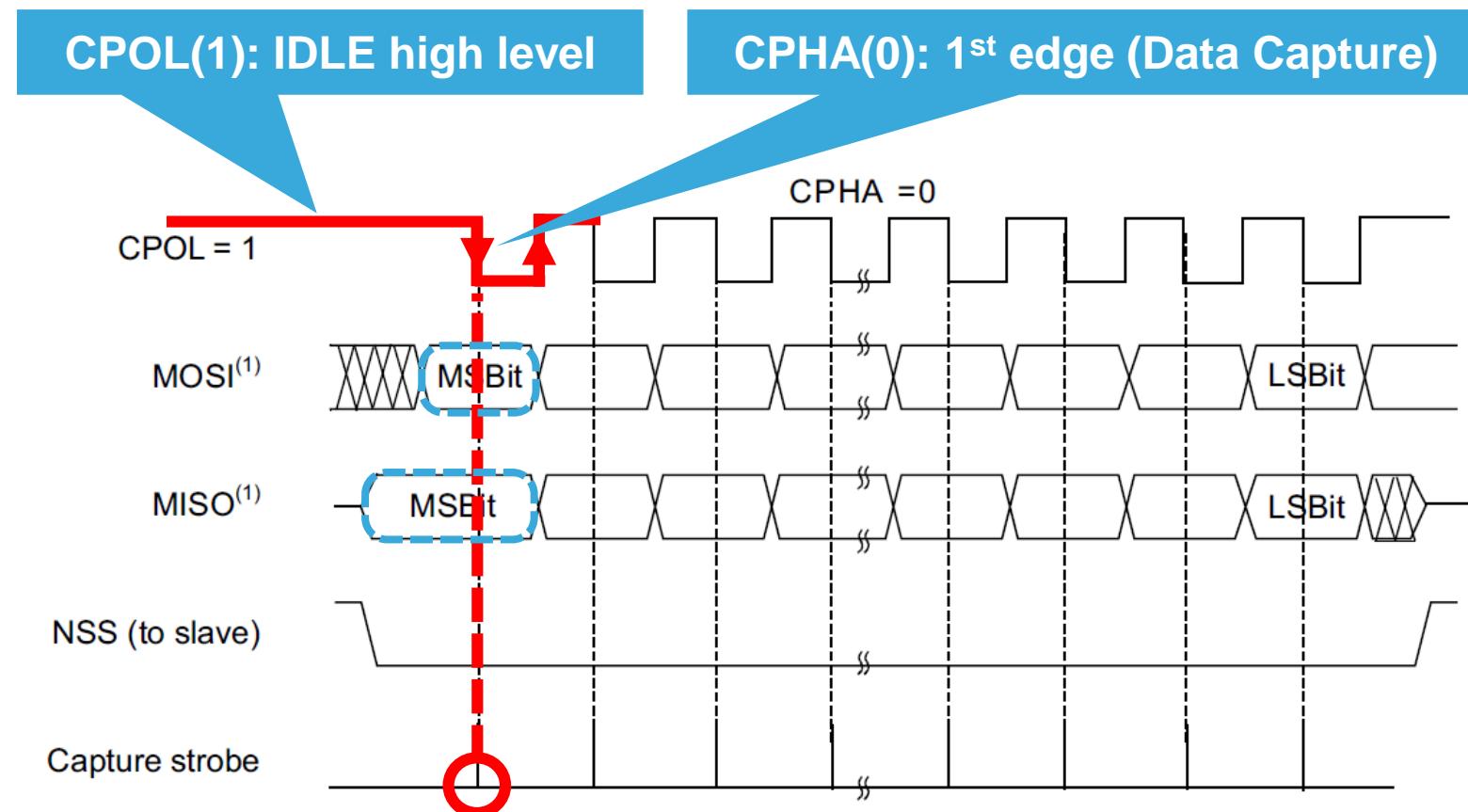
Data Frame Format

- Clock Phase 및 Polarity 제어 예 (CPHA = 0, CPOL = 0)



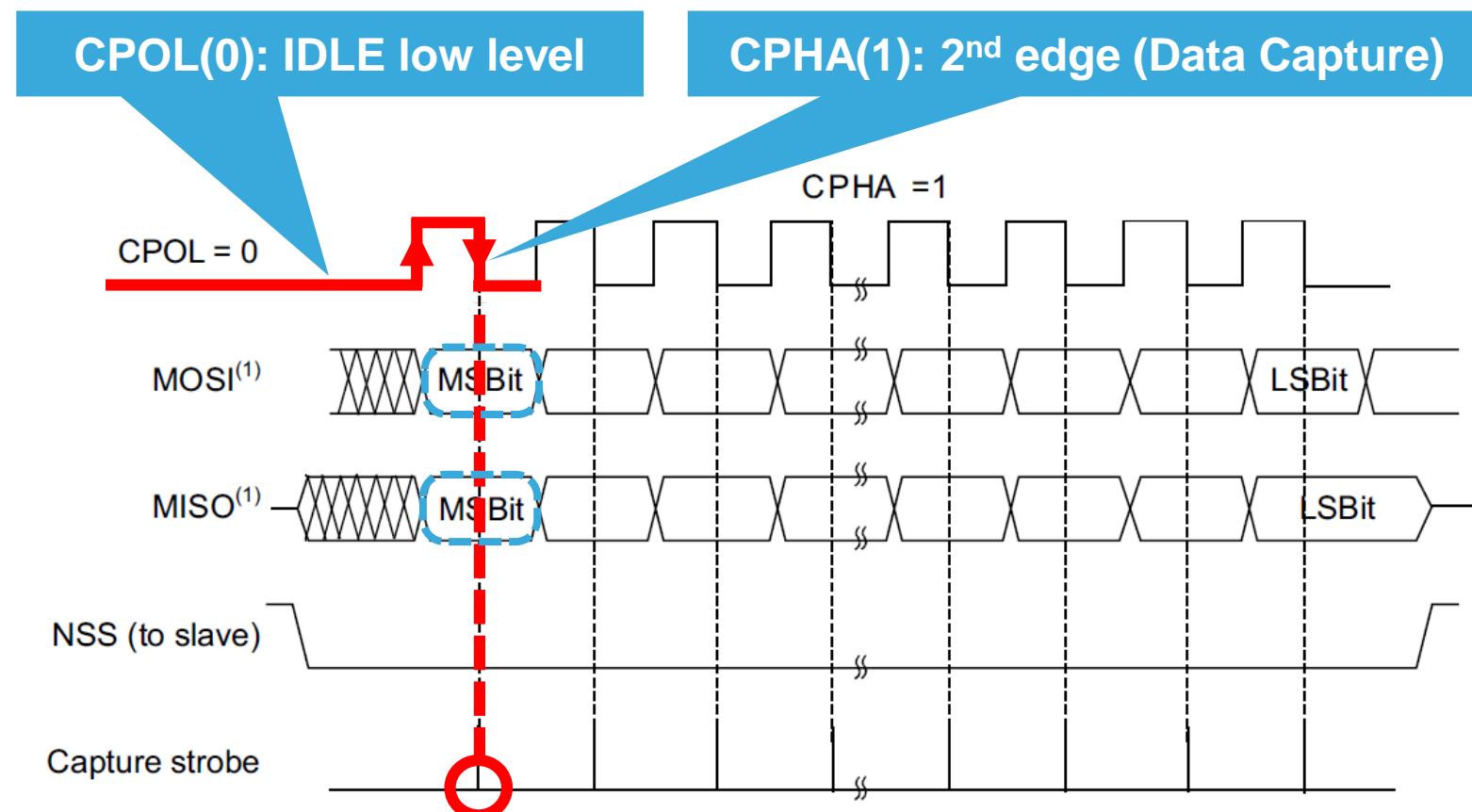
Data Frame Format

- Clock Phase 및 Polarity 제어 예 (CPHA = 0, CPOL = 1)



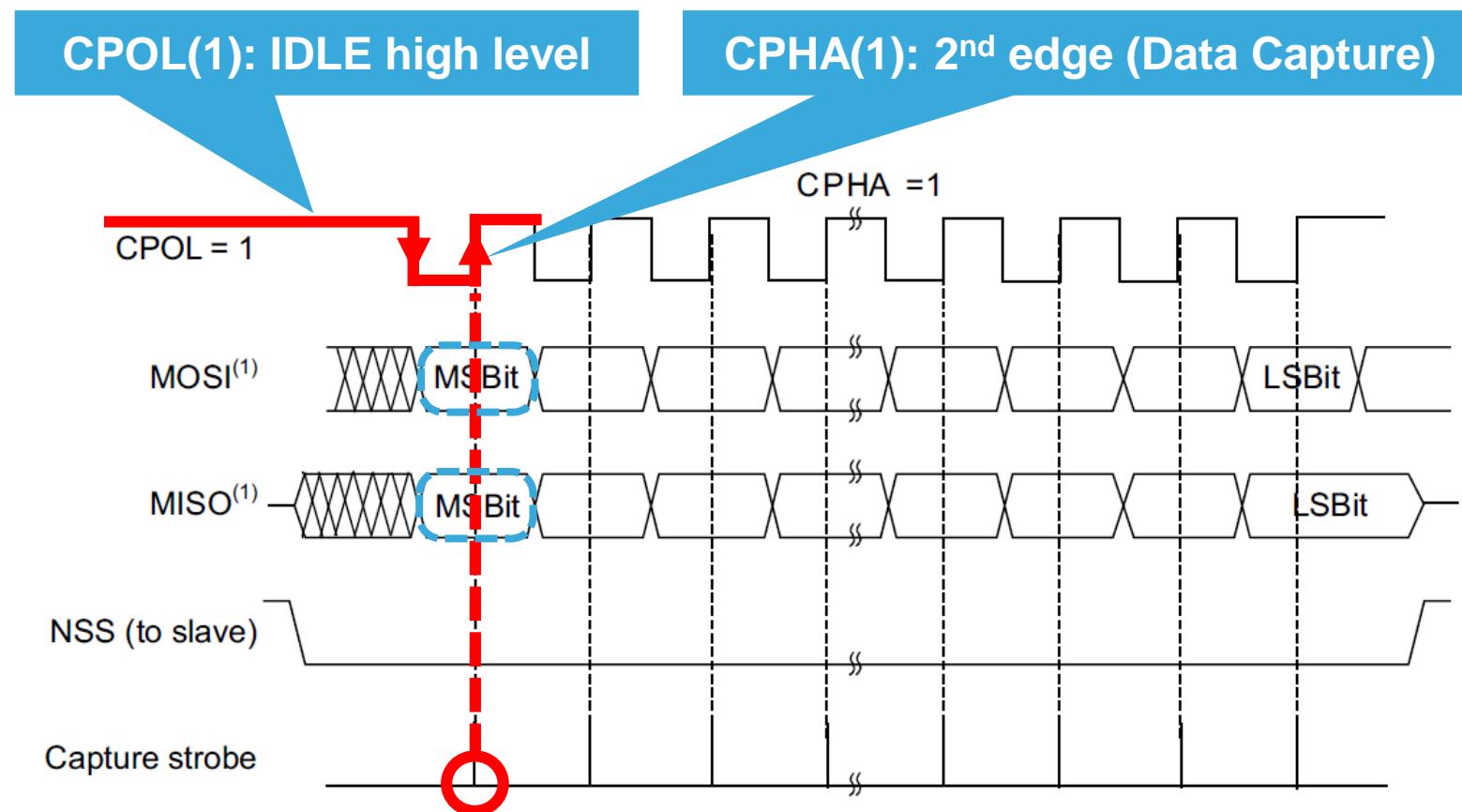
Data Frame Format

- Clock Phase 및 Polarity 제어 예 (CPHA = 1, CPOL = 0)



Data Frame Format

- Clock Phase 및 Polarity 제어 예 (CPHA = 1, CPOL = 1)



NSS (Slave Select) modes

- NSS output
 - Hardware 또는 Software management (SSM)
 - Master mode
 - Software management의 경우 GPIO를 하나 할당해서 SPI 출력 전후에 Toggle하도록 코딩한다
 - Hardware management의 경우 NSS 전용핀이 SPI 입출력 시점에 맞춰 자동으로 Toggle된다
- NSS input
 - Hardware 또는 Software management (SSM)
 - Slave mode
 - Software management의 경우 송수신이 필요할 때 SSI를 1로 세팅한다
 - Hardware management의 경우 NSS 전용핀의 상태에 따라 자동으로 송수신을 한다
 - Master mode
 - Multi-master로 동작하려는 경우 hardware management를 선택하고 입력 (SSOE = 0)으로 설정해서 상대방의 master 동작을 감지한다

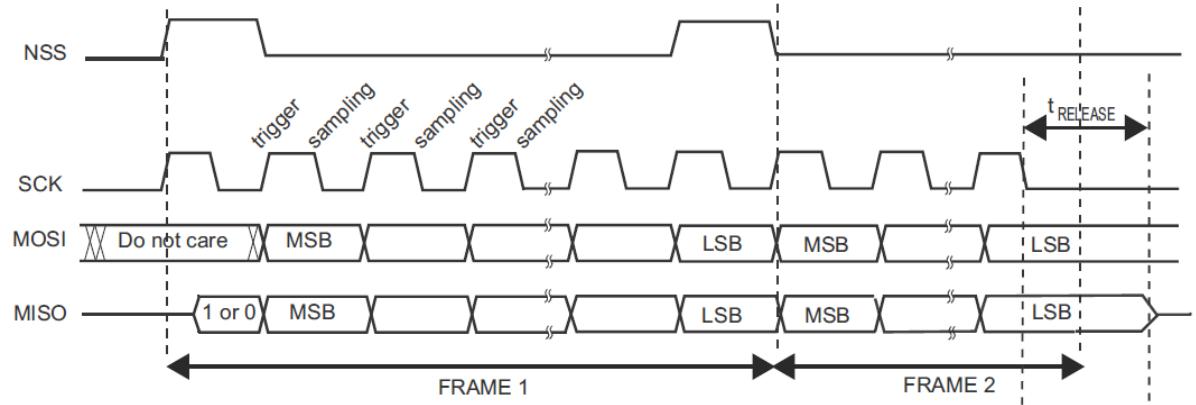
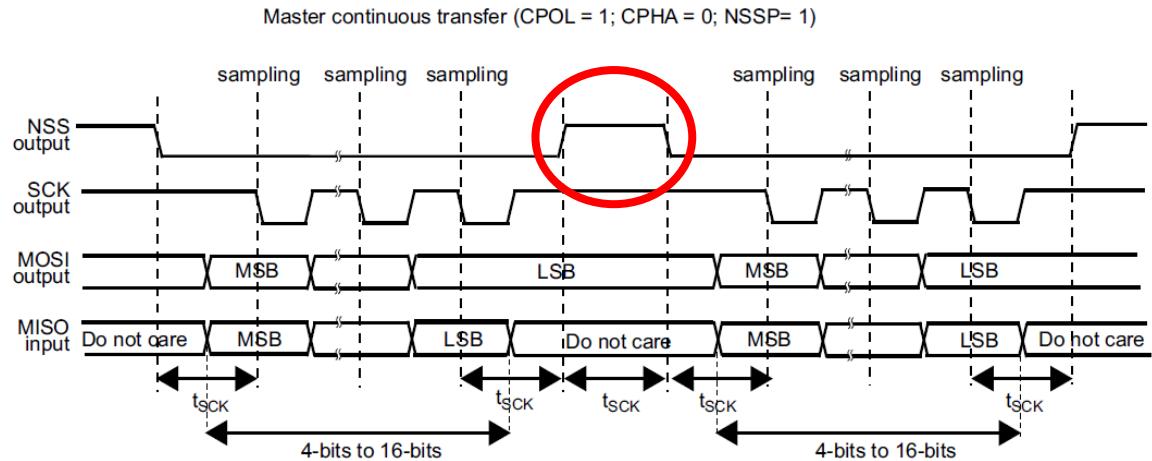
Specific NSS modes

- **NSS Pulse mode**

- Motorola mode 만 지원
 - TI mode 에서는 지원하지 않음
- CPHA = 0 만 지원
 - CPHA = 1 에서는 지원하지 않음
 - CPOL = 0 또는 1 은 don't care
- SPIx_CR2->NSSP = 1
- 연속된 데이터 전송 중에 NSS 를 계속 low로 잡고 있지 않고 중간에 high -> low 동작한다.

- **TI mode**

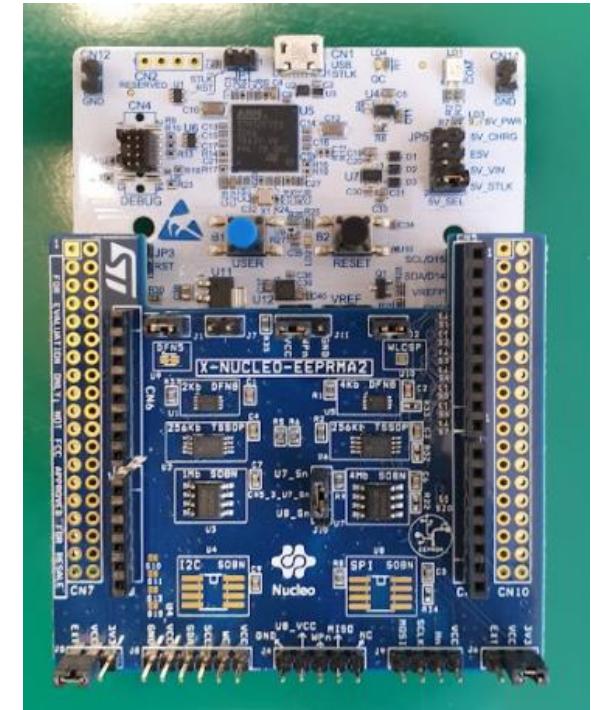
- Master and Slave support
- Fixed CPOL and CPHA setting
- Hi-Z slave MISO automatic control



Hands-On Session: SPI

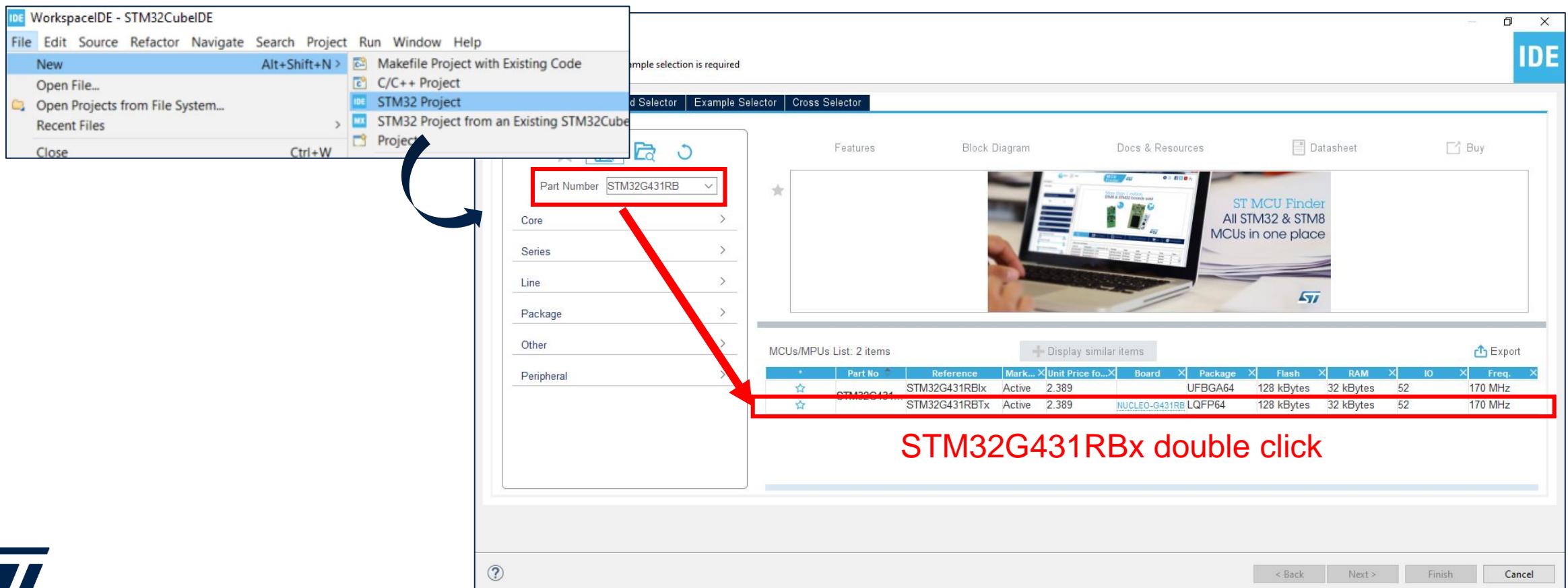


- NUCLEO-G431RB 보드와 X-NUCLEO-EEPRMA2 와 SPI 통신
- STM32CubeIDE를 사용하여 'SPI1' 설정 후 Initialization code를 생성
- X-NUCLEO-EEPRMA2에 탑재된 EEPROM을 쓰고 읽음
- 데이터가 올바르게 써져 있는지 비교



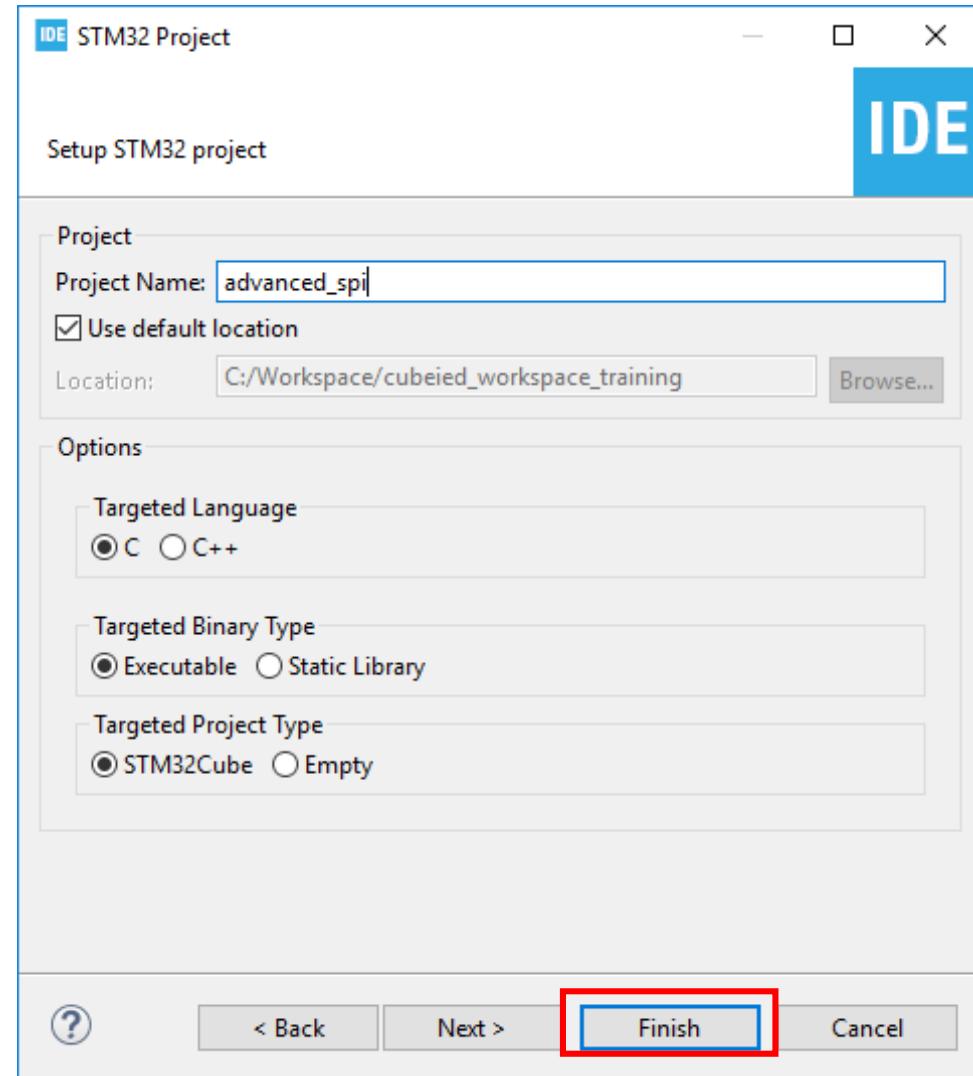
Create New Project

- File > New > STM32 Project
 - Part Number Search: **STM32G431RB**



Create New Project

- Project Name : 프로젝트 명칭 입력
- Click [Finish]



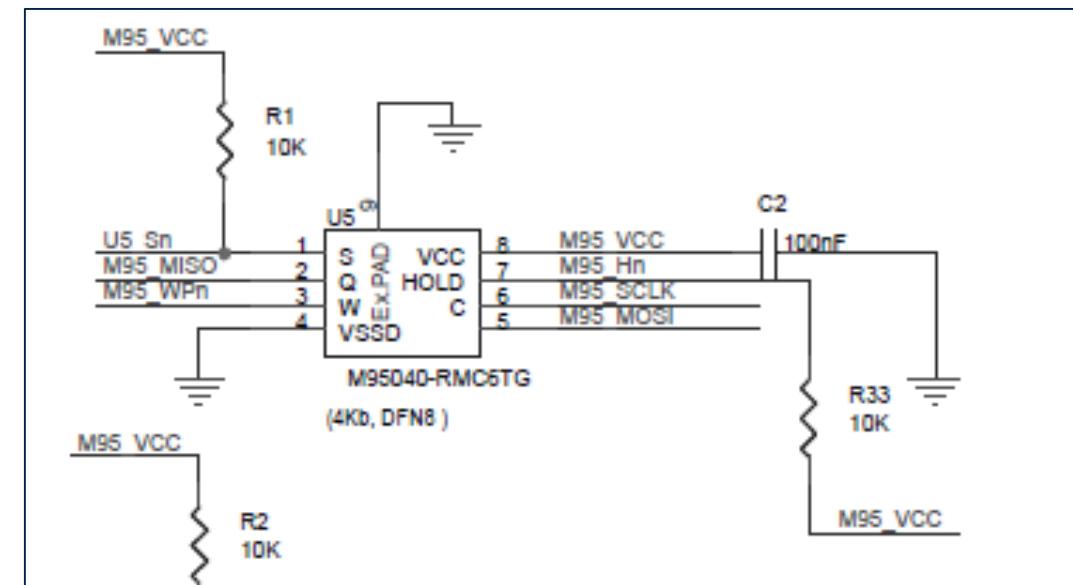
- Connect to the X-NUCLEO-EEPRM A2

- M95040-RMC6TG

- M95_SCLK: SPI1_SCK (CN5 – 6pin)
- M95_MISO: SPI1_MISO (CN5 – 5pin)
- M95_MOSI: SPI1_MOSI (CN5 – 4pin)
- U5_Sn: SPI1_NSS (CN9 – 3pin) (* Software NSS Control)

Table 2. Signal names

Signal name	Function
C	Serial Clock
D	Serial Data input
Q	Serial Data output
S	Chip Select
W	Write Protect
HOLD	Hold
V _{CC}	Supply voltage
V _{ss}	Ground



Configuration

- Pinout & Configuration

- Connectivity > SPI1 > Mode: Full-Duplex Master, Hardware NSS Signal: Disable
- Pin: SCK(PA5), MISO(PA6), MOSI(PA7), NSS(PA10: GPIO Output)

The screenshot shows the STM32CubeMX software interface with the following highlights:

- SPI1 Configuration:** The "Pinout & Configuration" tab is selected. In the "Connectivity" section, "SPI1" is selected. The "Mode" dropdown is set to "Full-Duplex Master" and "Hardware NSS Signal" is set to "Disable". A blue box labeled "Mode: Full-Duplex Master Hardware NSS Signal: Disable" is overlaid on this area.
- Pinout Labels:** A blue box labeled "SCK(PD1), MISO(PD3), MOSI(PD4), NSS(PD7)" is overlaid on the SPI pins (PD1, PD3, PD4, PD7).
- Pinout Diagram:** The right side shows the STM32G431RBTx QFP64 package with pins labeled. Two pins are circled in red:
 - Pin PA10 is circled in red and labeled "PA10, SPI1_CS".
 - Pin PA7 is circled in red and labeled "PA7, SPI1_MOSI".A blue box labeled "Enter User Label PA10, SPI1_CS 설정" is overlaid on the PA10 pin area.

SPI modes

The device can be driven by a microcontroller with its SPI peripheral running in either of the following modes:

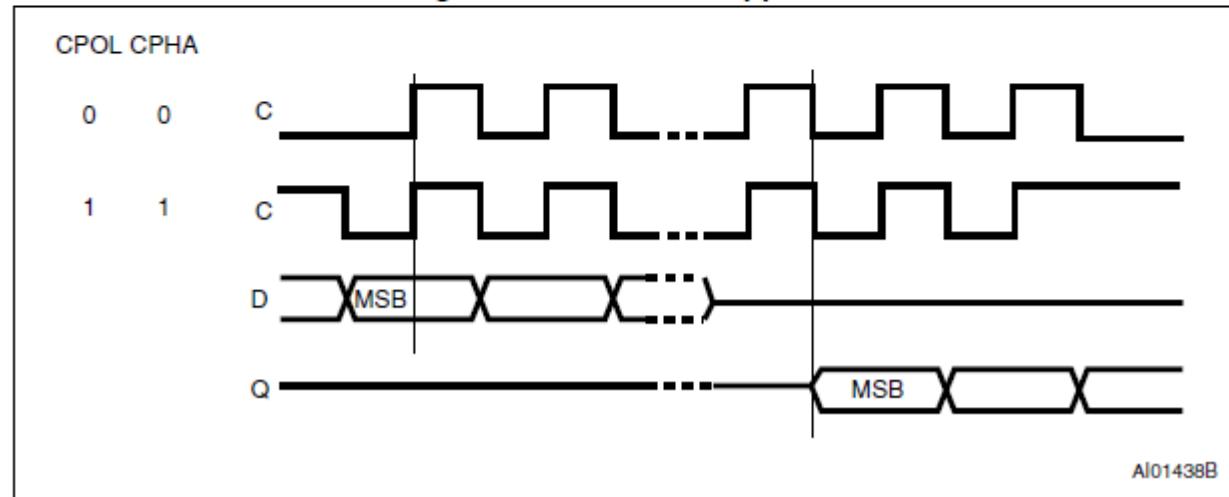
- CPOL=0, CPHA=0
- CPOL=1, CPHA=1

For these two modes, input data is latched in on the rising edge of Serial Clock (C), and output data is available from the falling edge of Serial Clock (C).

The difference between the two modes, as shown in [Figure 4: SPI modes supported](#), is the clock polarity when the bus master is in Stand-by mode and not transferring data:

- C remains at 0 for (CPOL=0, CPHA=0)
- C remains at 1 for (CPOL=1, CPHA=1)

Figure 4. SPI modes supported

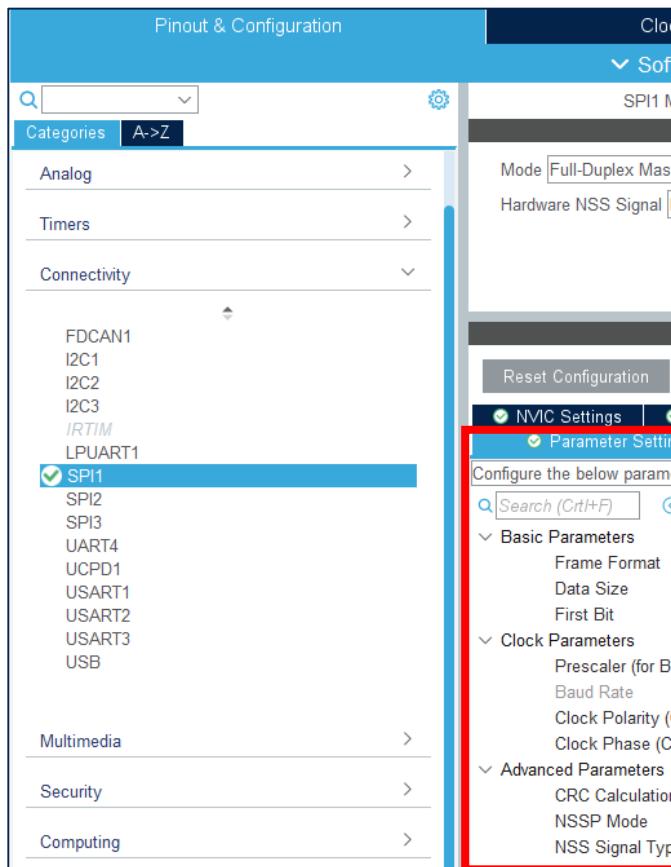


Configuration

- Pinout & Configuration

- Connectivity > SPI1 > Configuration > Parameter Setting

- Data Size: 8 Bits, Prescaler: 16



Frame Format: Motorola
Data Size: 8 Bits
First Bit: MSB First

Prescaler: 16
CPOL: Low
CPHA: 1 Edge

CRC Calculation: Disable
NNSS Mode: Disable
NSS Signal Type: Software

Configuration

- Pinout & Configuration

- Connectivity > SPI1 > Configuration > GPIO Settings
 - (PA5/PA6/PA7) Maximum output speed: High

I/O Maximum output speed
> High

Pin N...	Signal on...	GPIO out...	GPIO mode	GPIO Pul...	Maximu...	Fast Mode	User Label	Modified
PA5	SPI1_SCK	n/a	Alternate...	No pull-u...	High	n/a		<input checked="" type="checkbox"/>
PA6	SPI1_MI...	n/a	Alternate...	No pull-u...	High	n/a		<input checked="" type="checkbox"/>
PA7	SPI1_M...	n/a	Alternate...	No pull-u...	High	n/a		<input checked="" type="checkbox"/>

PA5#PA6#PA7 Configuration :

GPIO mode

GPIO Pull-up/Pull-down

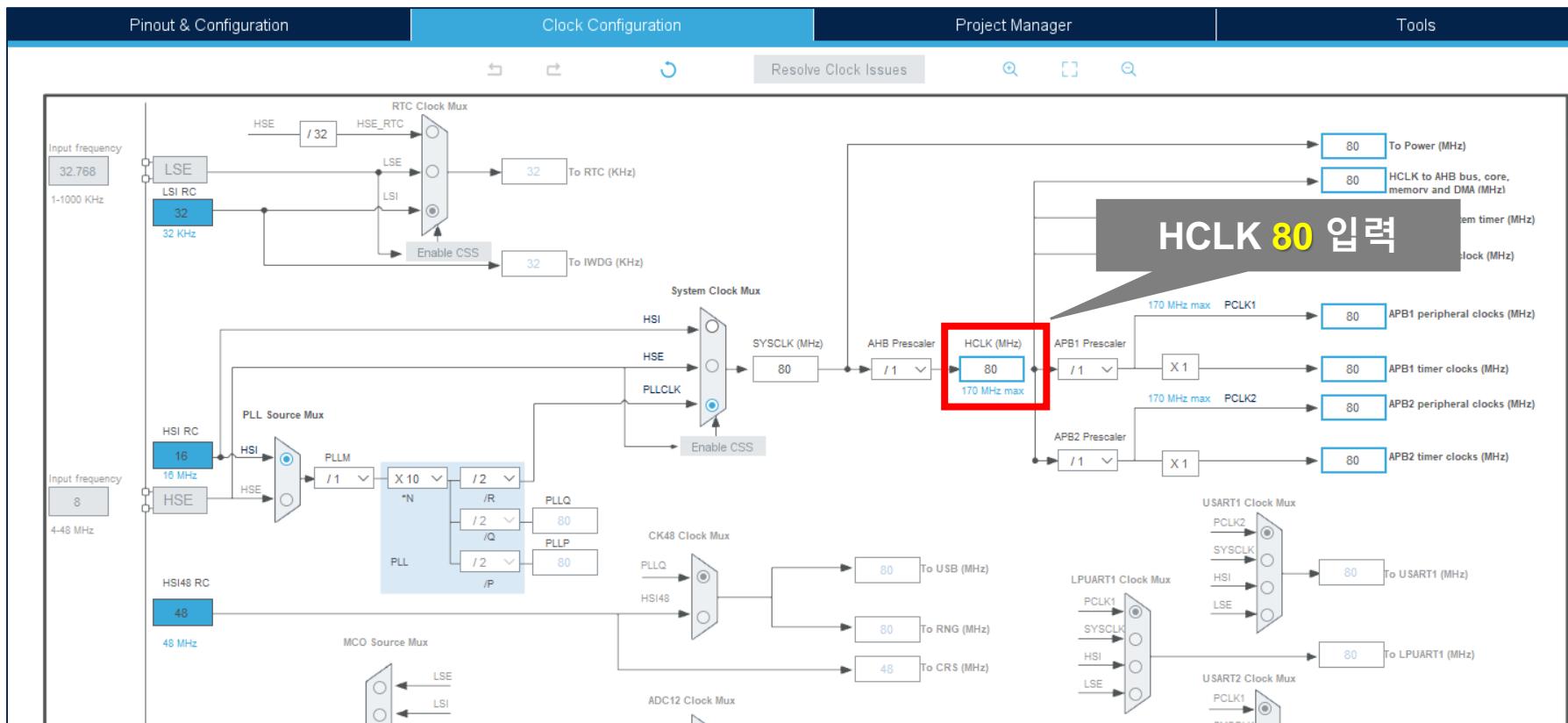
Maximum output speed

High

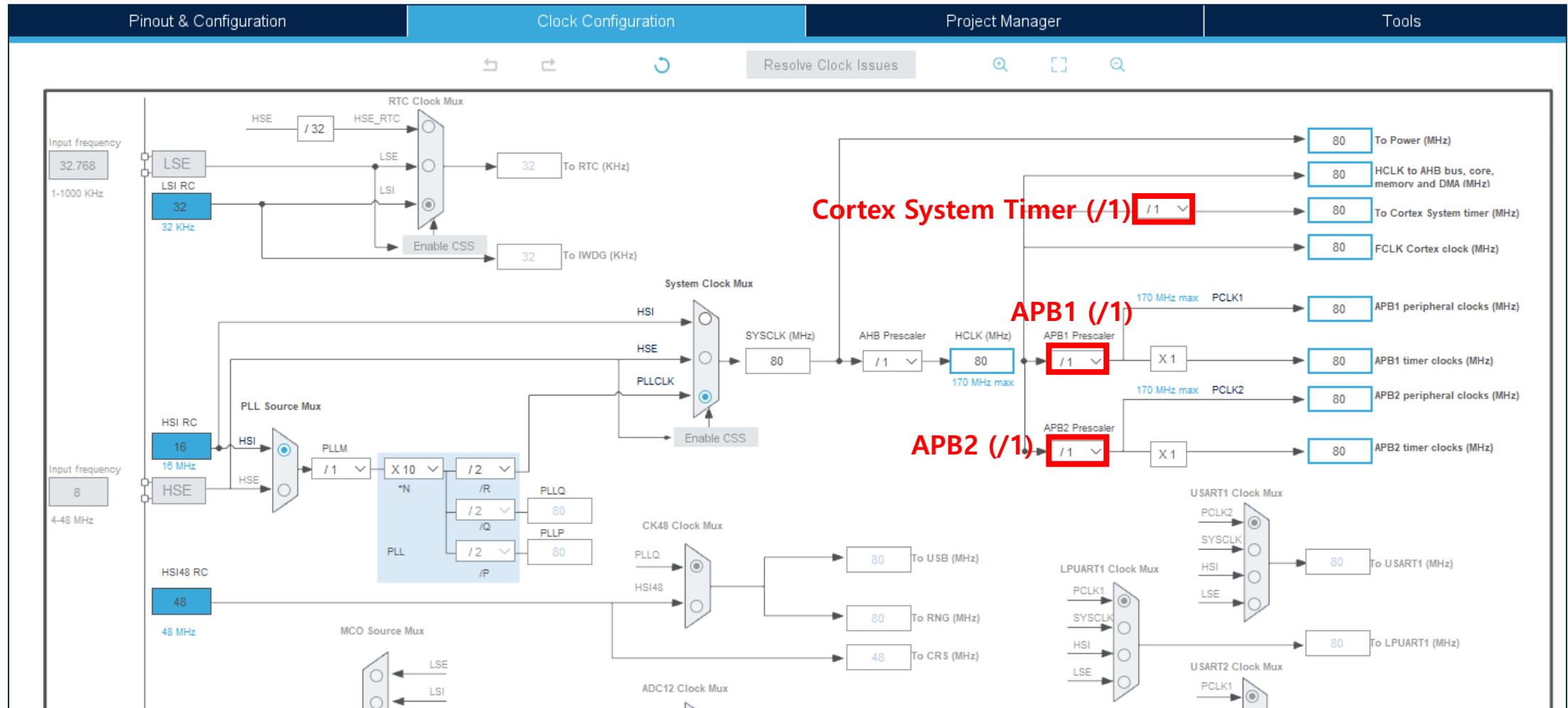
STM32G431RBTx
LQFP64

Clock Configuration

- SYSCLK 및 HCLK을 80MHz 설정 (PLL 사용)
 - Click [OK]

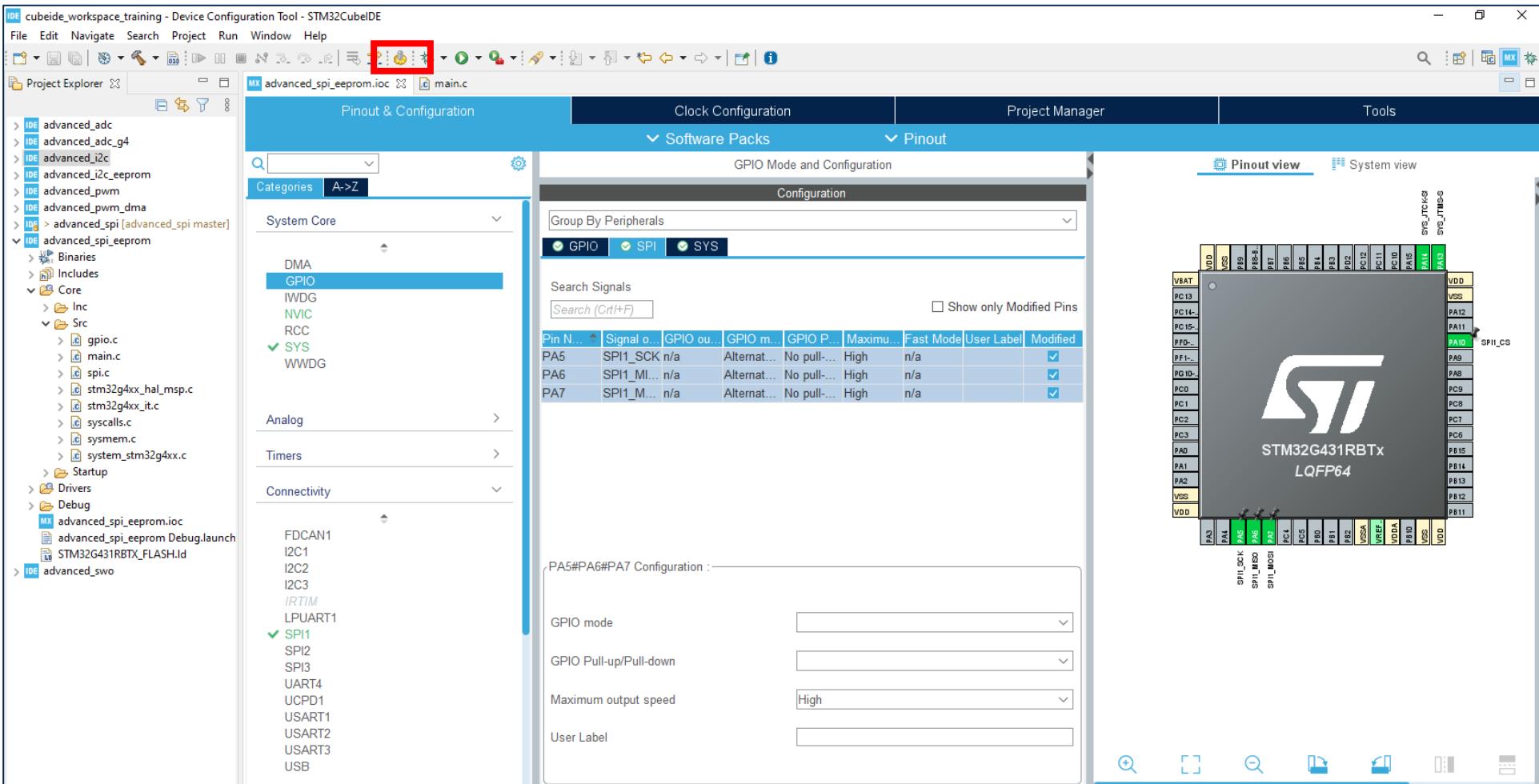


Clock Configuration



Generate Source Code

- Project > **Generate Code(Alt + K)**



Code description

- EEPROM Instruction

Table 4. Instruction set

Instruction	Description	Instruction format
WREN	Write Enable	0000 X110 ⁽¹⁾
WRDI	Write Disable	0000 X100 ⁽¹⁾
RDSR	Read Status Register	0000 X101 ⁽¹⁾
WRSR	Write Status Register	0000 X001 ⁽¹⁾
READ	Read from Memory Array	0000 A ₈ 011 ⁽²⁾
WRITE	Write to Memory Array	0000 A ₈ 010 ⁽²⁾
RDID ⁽³⁾	Read Identification Page	1000 0011
WRID ⁽³⁾	Write Identification Page	1000 0010
RDLS ⁽³⁾	Reads the Identification Page lock status.	1000 0011
LID ⁽³⁾	Locks the Identification page in read-only mode.	1000 0010

1. X = Don't Care.
2. For M95040, A8 = 1 for the upper half of the memory array and 0 for the lower half, while for M95010 and M95020, A8 is Don't Care.
3. Available only for the M95040-DF device.

Code description

- EEPROM Instruction defines

```
/* Private define -----*/
/* USER CODE BEGIN PD */
#define EEPROM_WREN 0x06 /* Write Enable */
#define EEPROM_WRDI 0x04 /* Write Disable */
#define EEPROM_RDSR 0x05 /* Read Status Register */
#define EEPROM_WRSR 0x01 /* Write Status Register */
#define EEPROM_READ 0x03 /* Read from Memory Array */
#define EEPROM_WRITE 0x02 /* Write to Memory Array */

#define EEPROM_WIP_FLAG 0x01 /* Write In Progress (WIP) flag */

#define EEPROM_PAGESIZE 16 /* Page size*/
/* USER CODE END PD */
```

Code description

- Buffer declare

```
/* Private variables -----*/  
  
/* USER CODE BEGIN PV */  
#define COUNTOF(__BUFFER__)    (sizeof(__BUFFER__) / sizeof(*(__BUFFER__)))  
#define TXBUFFERSIZE          (COUNTOF(aTxBuffer) - 1)  
/* Size of Reception buffer */  
#define RXBUFFERSIZE          TXBUFFERSIZE  
  
uint8_t aTxBuffer[] = /* SPI HAL API EEPROM driver example: \  
                         This firmware provides a basic example of how to use the SPI HAL API based and\  
                         an associate SPI EEPROM driver to communicate with M95040-RMC6TG EEPROM */;  
uint8_t aRxBuffer[RXBUFFERSIZE];  
  
/* USER CODE END PV */
```

Code description

- Function prototypes

```
/* Private function prototypes -----*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
static uint8_t EEPROM_Read_Register(uint8_t inst);
static int EEPROM_Read(uint32_t address, uint8_t *buff, int size);
static int EEPROM_Write(uint32_t address, uint8_t *buff, int size);
static uint16_t Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength);
/* USER CODE END PFP */
```

Code description

- Global 변수

```
/* Private user code -----*/  
/* USER CODE BEGIN 0 */  
uint16_t Memory_Address;  
int Remaining_Bytes;  
/* USER CODE END 0 */
```

Code description

- EEPROM_Read_Register

```
/* USER CODE BEGIN 4 */
static uint8_t EEPROM_Read_Register(uint8_t inst)
{
    uint8_t instruction = 0xff;
    uint8_t result = 0xff;

    instruction = inst;

    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);

    if(HAL_SPI_Transmit(&hspi1, (uint8_t *)&instruction, 1, 100) != HAL_OK)
    {
        return -1;
    }

    if(HAL_SPI_Receive(&hspi1, (uint8_t *)&result, 1, 100) != HAL_OK)
    {
        return -1;
    }
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    return result;
}
```

Code description

- EEPROM_Read

```
static int EEPROM_Read(uint32_t address, uint8_t *buff, int size)
{
    uint8_t instruction[2];

    instruction[0] = (uint8_t)((address >> 8) << 3) | (EEPROM_READ));
    instruction[1] = (uint8_t)((address) & (0x000000ff));

    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);

    if(HAL_SPI_Transmit(&hspi1, (uint8_t *)instruction, 2, 200) != HAL_OK)
    {
        return -1;
    }

    if(HAL_SPI_Receive(&hspi1, (uint8_t *)buff, size, (100 * size)) != HAL_OK)
    {
        return -1;
    }
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    return 0;
}
```

Code description

- EEPROM_Write

```
static int EEPROM_Write(uint32_t address, uint8_t *buff, int size)
{
    uint8_t instruction[2];

    /* Initialize Remaining Bytes Value to TX Buffer Size */
    Remaining_Bytes = size;
    /* Initialize Memory address to 0 since EEPROM write will start from address 0 */
    Memory_Address = address;

    while(Remaining_Bytes > 0)
    {
        /* write enable */
        HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);

        instruction[0] = EEPROM_WREN;

        if(HAL_SPI_Transmit(&hspi1, (uint8_t *)instruction, 1, 200) != HAL_OK)
        {
            Error_Handler();
        }
        HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

        /* write data */
        instruction[0] = (uint8_t)((Memory_Address >> 8) << 3) | (EEPROM_WRITE));
        instruction[1] = (uint8_t)((Memory_Address) & (0x000000ff));

        HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
        if(HAL_SPI_Transmit(&hspi1, (uint8_t *)instruction, 2, 200) != HAL_OK)
        {
            return -1;
        }
    }

    if(HAL_SPI_Transmit(&hspi1, (uint8_t *)(buff + Memory_Address), EEPROM_PAGESIZE, EEPROM_PAGESIZE*100) != HAL_OK)
    {
        Error_Handler();
    }
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    /* wait until finish */
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
    while((EEPROM_Read_Register(EEPROM_RDSR)&EEPROM_WIP_FLAG) == EEPROM_WIP_FLAG)
    {
        HAL_Delay(1);
    }
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    /* write disable */
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_RESET);
    instruction[0] = EEPROM_WRDI;

    if(HAL_SPI_Transmit(&hspi1, (uint8_t *)instruction, 1, 200) != HAL_OK)
    {
        return -1;
    }
    HAL_GPIO_WritePin(SPI1_CS_GPIO_Port, SPI1_CS_Pin, GPIO_PIN_SET);

    Remaining_Bytes -= EEPROM_PAGESIZE;
    Memory_Address += EEPROM_PAGESIZE;
}

return 0;
}
```

Code description

- **Buffercmp**

```
static uint16_t Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength)
{
    while (BufferLength--)
    {
        if ((*pBuffer1) != *pBuffer2)
        {
            return BufferLength;
        }
        pBuffer1++;
        pBuffer2++;
    }

    return 0;
}
/* USER CODE END 4 */
```

Code description

- User code

```
/* USER CODE BEGIN 2 */
if((EEPROM_Read_Register(EEPROM_RDSR)&EEPROM_WIP_FLAG) == EEPROM_WIP_FLAG)
{
    Error_Handler();
}

if(EEPROM_Write(0x0000, aTxBuffer, TXBUFFERSIZE) != 0x00)
{
    Error_Handler();
}

if(EEPROM_Read(0x0000, aRxBuffer, RXBUFFERSIZE) != 0x00)
{
    Error_Handler();
}

if(Buffercmp((uint8_t*)aTxBuffer, (uint8_t*)aRxBuffer, RXBUFFERSIZE))
{
    /* Processing Error */
    Error_Handler();
}
/* USER CODE END 2 */
```

Debugging

- Debugging start 후 aRxBuffer에 데이터 확인

이곳에 걸려있지 않으면
데이터 쓰기/읽기 정상으로 간주

Expression	Type	Value
aRxBuffer	uint8_t [238]	0x2000011c <aRxBuffer>
[0..99]	uint8_t [100]	0x2000011c <aRxBuffer>
(0: aRxBuffer[0])	uint8_t	47 'I'
(0: aRxBuffer[1])	uint8_t	42 'A'
(0: aRxBuffer[2])	uint8_t	32 ' '
(0: aRxBuffer[3])	uint8_t	83 'S'
(0: aRxBuffer[4])	uint8_t	80 'P'
(0: aRxBuffer[5])	uint8_t	73 'I'
(0: aRxBuffer[6])	uint8_t	32 ' '
(0: aRxBuffer[7])	uint8_t	72 'H'
(0: aRxBuffer[8])	uint8_t	65 'A'
(0: aRxBuffer[9])	uint8_t	76 'L'
(0: aRxBuffer[10])	uint8_t	32 ' '
(0: aRxBuffer[11])	uint8_t	65 'A'
(0: aRxBuffer[12])	uint8_t	80 'P'
(0: aRxBuffer[13])	uint8_t	73 'I'
(0: aRxBuffer[14])	uint8_t	32 ' '
(0: aRxBuffer[15])	uint8_t	69 'E'
(0: aRxBuffer[16])	uint8_t	69 'E'
(0: aRxBuffer[17])	uint8_t	80 'P'
(0: aRxBuffer[18])	uint8_t	82 'R'
(0: aRxBuffer[19])	uint8_t	79 'O'
(0: aRxBuffer[20])	uint8_t	77 'M'
(0: aRxBuffer[21])	uint8_t	32 ' '
(0: aRxBuffer[22])	uint8_t	100 'd'
(0: aRxBuffer[23])	uint8_t	114 'r'
(0: aRxBuffer[24])	uint8_t	105 'i'
(0: aRxBuffer[25])	uint8_t	118 'v'
(0: aRxBuffer[26])	uint8_t	101 'e'
(0: aRxBuffer[27])	uint8_t	114 'r'
(0: aRxBuffer[28])	uint8_t	32 ' '
(0: aRxBuffer[29])	uint8_t	101 'e'

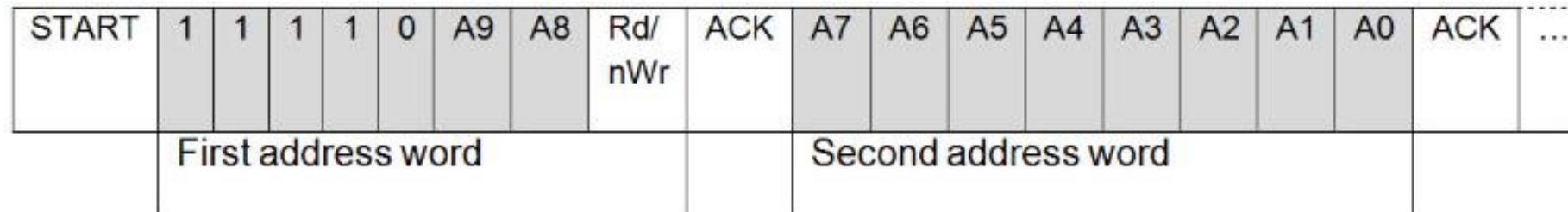
I2C (Inter-Integrated Circuit)



- I²C (Inter Integrated Circuit)

- Half-duplex 동기화 방식의 2핀 직렬 통신 규격
- 신호선 구성
 - SCL: Clock 신호
 - SDA : 양방향 데이터 신호
- 하나 이상의 Master 와 하나 이상의 Slave 가 연결되는 구조
- SPI 에서 사용하는 하드웨어 chip select 핀이 없는 대신 SDA 첫번째 데이터로 전달되는 7bit 또는 10bit 의 Slave 고유주소로 각 장치를 addressing 한다

10 bit address

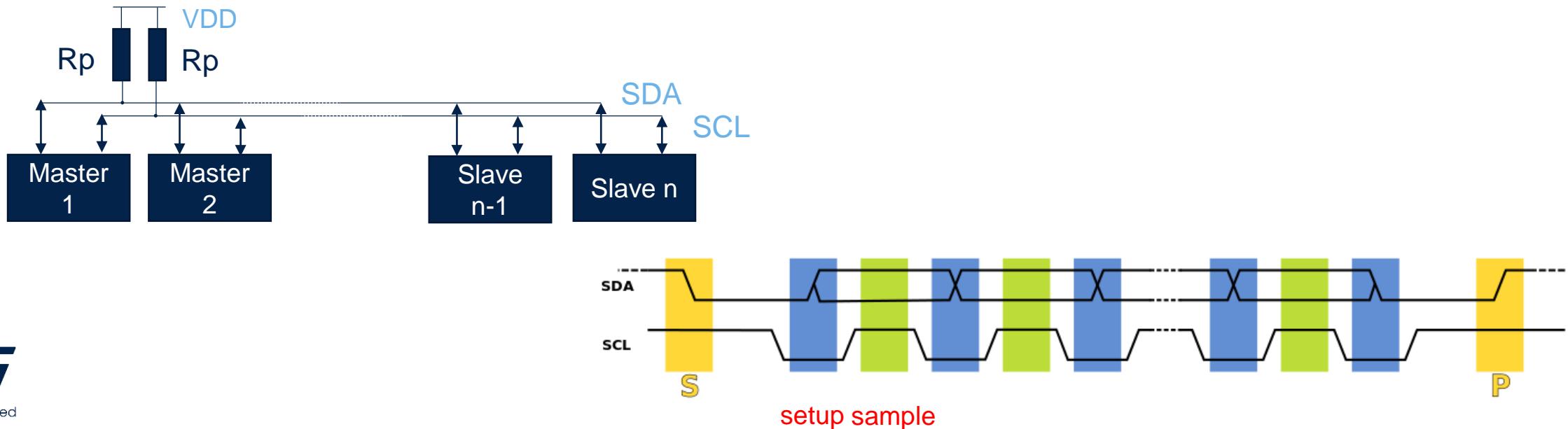


What is I2C

- SCL 은 항상 Master 가 출력하지만 Clock stretch 를 할 때는 Slave 도 SCL 을 low 로 출력할 수는 있다. SDA 는 Master 와 Slave 가 교대로 출력한다.
- 최대 통신 속도
 - Standard mode : 100 Kbit/s
 - Fast mode : 400 Kbit/s
 - Fast mode plus : 1 Mbit/s
 - High Speed mode : 3.4 Mbit/s
 - Ultra Fast mode : 5 Mbit/s (Write only)

- SDA와 SCL 신호

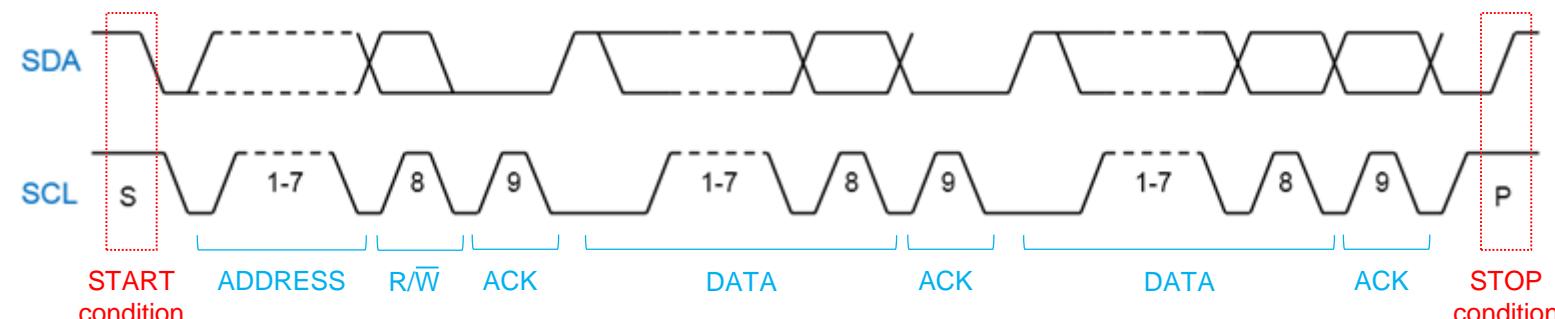
- SDA 와 SCL 모두 open drain 출력 형태를 사용하며 typical 3.3V 전원에 2.2K~4.7K 옴의 풀업 저항을 연결한다
- 버스 길이, 커넥터 등으로 인해 capacitance 가 증가하면 SDA 와 SCL 라인의 rising/falling edge 속도가 느려지고 원하는 통신속도가 나오지 않을 수 있다.
- 풀업 전압 70% 이상에서 logic high, 30% 이하에서 logic low로 인식된다.
- SCL 이 low 일 때 SDA 가 setup 되고 SCL 이 high 일 때 SDA 가 sample 된다.



I2C Bus Protocol

- I2C Protocol Features

- Data 송수신은 Master 가 생성한 Start 신호 이후에 시작하고 Stop 신호로 종료한다
 - **Start (S)** : SCL이 High 상태일때 SDA가 High에서 Low로 변하는 패턴
 - **Stop (P)** : SCL이 High 상태일때 SDA가 Low에서 High로 변하는 패턴
 - **Repeated Start (Sr)** : Stop 이전에 발생하는 추가 Start 패턴
- Data 는 8 bit (1 byte) 단위로 Most Significant Bit (MSB) 가 먼저 전송된다
- 연속으로 전송할 수 있는 byte 수에 제한이 없다
- Start 신호 이후 첫번째 Data 는 Slave address (7bit 또는 10bit)이며 다음 추가 1 bit 의 Data 로 Slave 에 쓰기(0) 요청인지 읽기(1) 요청인지 알려준다
- 각 Data 의 9번째 Clock 의 1 bit Data 는 수신 ACK(0)/NACK(1) 용도로 사용된다
- I2C 버스를 Stop 으로 끊지 않고 연속으로 데이터 송수신 해야 되는 경우 Repeated Start 를 전송 후 다시 Slave address 전송을 시작한다



I2C Bus Protocol

- From master to slave
- From slave to master
- Depends on R/W

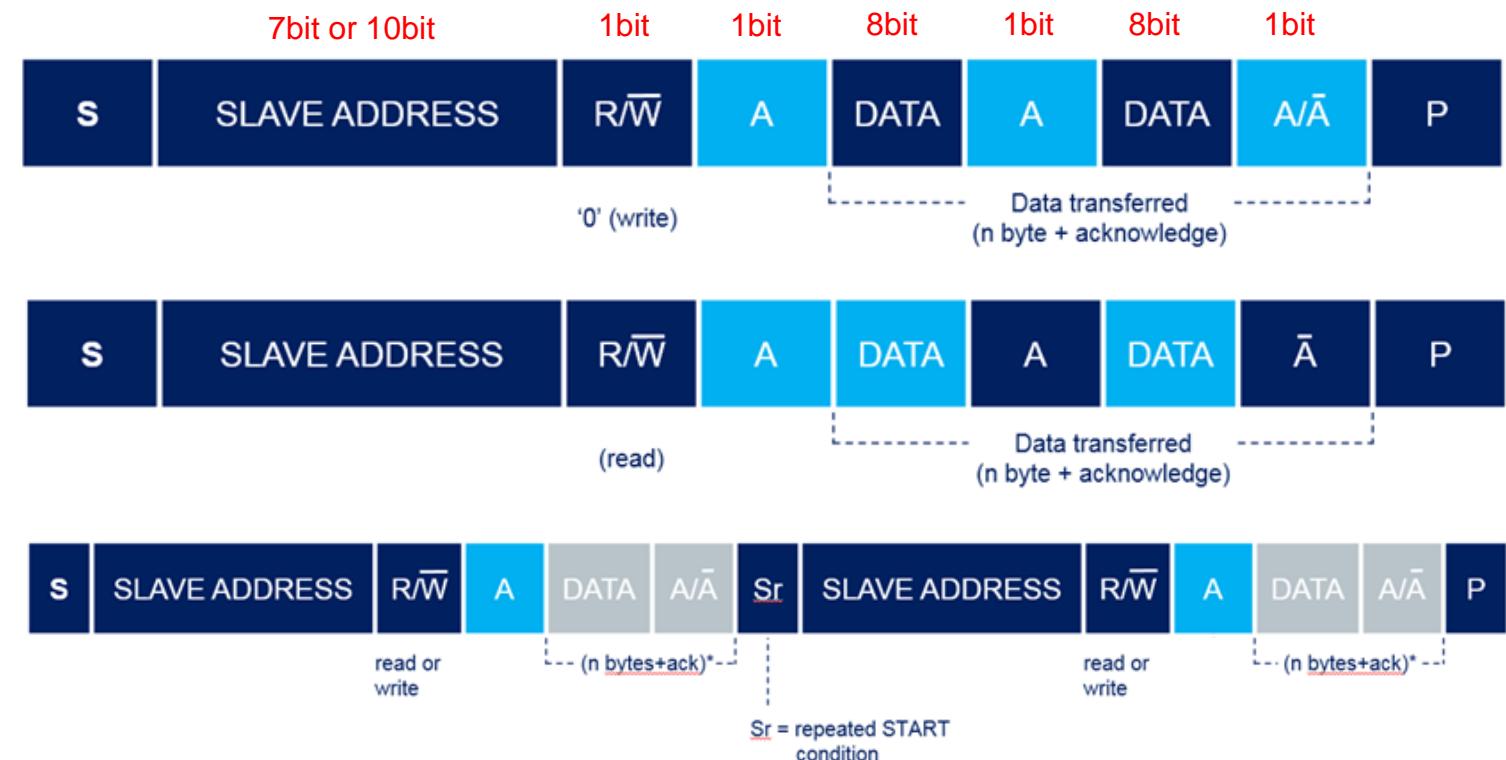
A = acknowledge (SDA Low)

\bar{A} = Not acknowledge (SDA High)

S = START condition

P = STOP condition

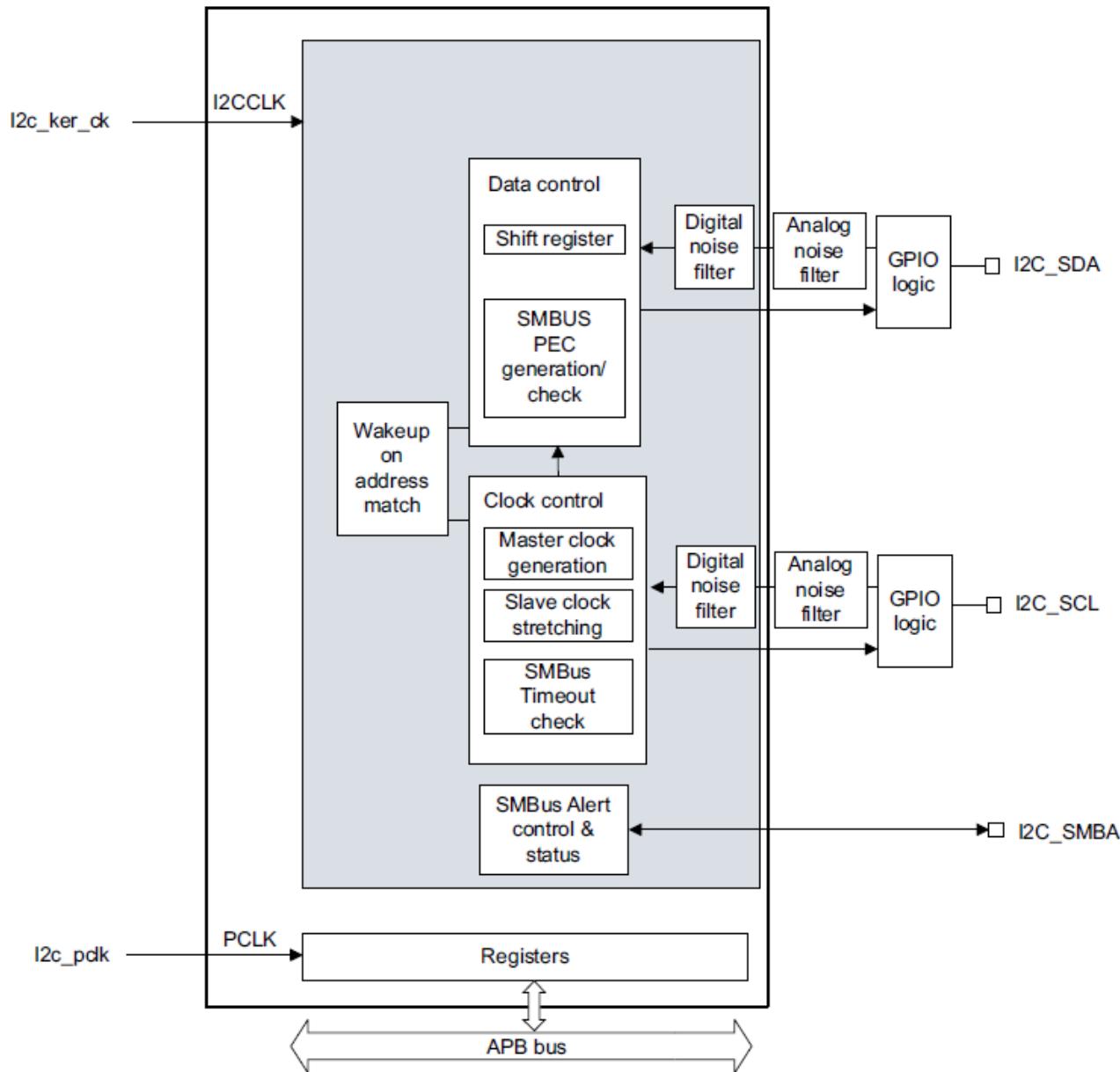
S_r = Repeated START condition



- 특징

- Standard mode, Fast mode, FM+ mode (20mA output drive) 지원
- (Multi) Master 또는 Slave로 동작 가능
- Program 가능한 setup time과 hold time 지원
- Program 가능한 analog와 digital noise filter 지원
- 7-bit 와 10-bit address mode 지원 및 7-bit dual addressing 지원
- Slave mode에서 clock stretching 지원 선택 가능 (Master mode 항상 enable)
- Stop mode에서 address match에 의한 wakeup 가능
- HSI, SYSCLK, PCLK에서 I2CCLK 클럭 소스 선택 가능
- SMBus 3.0 호환
- PMBus 1.3 호환

Block diagram



SDA and SCL noise filter

- Analog Noise Filter

- Pulse width 가 50 ns 보다 작은 noise spike 만 필터링 할 수 있으며 90 ns 보다 큰 noise 는 필터링 할 수 없다

Table 82. I2C analog filter characteristics⁽¹⁾

Symbol	Parameter	Min	Max	Unit
t_{AF}	Maximum pulse width of spikes that are suppressed by the analog filter	50 ⁽²⁾	90 ⁽³⁾	ns

1. Guaranteed by design.
2. Spikes with widths below $t_{AF(min)}$ are filtered.
3. Spikes with widths above $t_{AF(max)}$ are not filtered

- 온도, 전압, 제품 편차 등에 의해 필터링 pulse width 편차가 발생할 수 있다
- 디폴트 enable이며 소프트웨어 설정으로 enable, disable 을 변경 할 수 있다
- STOP 모드에서 I2C로 wakeup 할 때에도 사용 할 수 있다

SDA and SCL noise filter

- **Digital Noise Filter**

- 0에서 15 사이로 설정한 I2CCLK 클럭 이하의 noise 만 필터링 할 수 있다
- 디폴트 disable이며 소프트웨어 설정으로 enable, disable 을 변경 할 수 있다
- Analog filter 가 enable 되어 있으면 여기에 필터링 시간이 추가 된다
- 온도, 전압, 제품 편차 등에 의해 필터링 pulse width 편차가 발생하지 않는다
- STOP 모드에서 사용할 수 없다

- **NBYTES**

- Master 또는 Slave 모드에서 송수신 시작하기 전에 송수신할 전체 byte 수를 NBYTES [7:0]에 설정하면 송수신이 완료 되었을 때 Stop, ACK/NACK 와 같은 하드웨어 처리가 자동으로 된다
- 8 bit 최대크기인 255 byte 이상을 송수신 해야 되는 경우 RELOAD를 enable 한다

- **RELOAD**

- 255 byte 이상을 송수신하는 경우 또는 Slave byte control (SBC=1) 기능으로 수신 byte에 ACK/NACK 응답을 선택하고 싶은 경우 RELOAD를 enable 한다
- **NBYTES > 255**
 - 설정한 NBYTES 전송이 완료되면 TCR (Transfer Complete Reload) 플래그가 설정된다 (TCIE=1인 경우 TCR 인터럽트 발생)
 - NBYTES에 0이 아닌 새로운 값을 쓰면 TCR 플래그는 clear 되고 TCR 플래그가 clear 되기 전까지는 SCL 신호가 low로 유지된다
 - 255보다 작은 최종 byte 수를 NBYTES에 썼다면 RELOAD를 disable 한다
- **SBC = 1**
 - 1 byte 단위로 ACK/NACK를 하려면 NBYTES를 1로 설정하고 RELOAD를 enable 한다
 - 1 byte 수신이 완료되면 TCR 플래그가 설정되고 9번째 clock에 clock stretch가 시작된다
 - I2C_RXDR를 읽어서 ACK/NACK 설정을 한 후 NBYTES에 0이 아닌 새로운 값을 쓰면 TCR 플래그가 clear 되고 clock stretch는 종료된다

Automatic end mode

- Automatic end mode
 - NBYTES 로 설정한 개수 만큼 송수신이 완료되면 자동으로 Stop 신호가 출력된다
 - Master 모드에서만 enable (AUTOEND = 1) 할 수 있으며 Slave 또는 RELOAD 가 설정되어 있으면 AUTOEND 설정은 무시된다
- Software end mode
 - NBYTES 로 설정한 개수 만큼의 송수신이 완료되면 TC (Transfer Complete) 플래그가 설정되지만 (TCIE=1 인 경우 TC 인터럽트 발생) Stop 신호를 출력하지는 않는다
 - Start (Repeated Start) 또는 Stop 신호를 generation 하면 TC 플래그가 clear 되며 TC 플래그가 clear 되기 전까지는 SCL 신호가 low 로 유지된다
 - Start 이후에 Repeated Start 생성이 필요한 경우 주로 사용된다

STM32G4 I2C Registers

- I2Cx_CR1, I2Cx_CR2
 - I2C Control Register
 - SMBus 설정, Clock Stretch enable 여부, Noise Filter enable 여부 및 값 설정, NBYTES[7:0] 송수신 byte 수 설정, AUTOEND 설정 등
 - Error, Transmit Complete, Stop detection, NACK received, Address match, RX/TX 인터럽트들 enable 여부를 설정한다
 - Start, Stop, NACK 신호를 generation 한다
- I2Cx_OAR1, I2Cx_OAR2
 - I2C Own Address Register (Slave mode only)
 - Slave mode로 동작할 때 자신의 slave address 를 OAR에 설정하고 자신의 주소로 데이터가 수신되면 자동으로 ACK 응답을 하고 address match 이벤트도 발생된다

- I2Cx_ISR

- I2C Interrupt and Status Register

- ADDRCODE[6:0] : Address match 가 발생한 주소를 알려준다 (Slave mode only)
 - DIR : Master 로 부터 쓰기 요청인지 읽기 요청인지를 알려준다 (Slave mode only)
 - BUSY : 송수신이 진행중임을 나타낸다 (set by hardware when Start condition, cleared by hardware when Stop condition or PE=0)
 - OVR : Clock stretch 가 disable 되어 있는 경우 overrun 또는 underrun 이 발생했음을 알려준다
 - BERR : 송수신 중에 버스에 Start 또는 Stop 이 비정상적인 타이밍에 발생했음을 알려준다
 - STOPF : 버스에 Stop 이 발생했음을 알려준다
 - NACKF : NACK 를 수신하였음을 알려준다
 - ADDR : 수신한 Slave address 가 OAR 중 하나와 일치하였음을 알려준다

STM32G4 I2C Registers

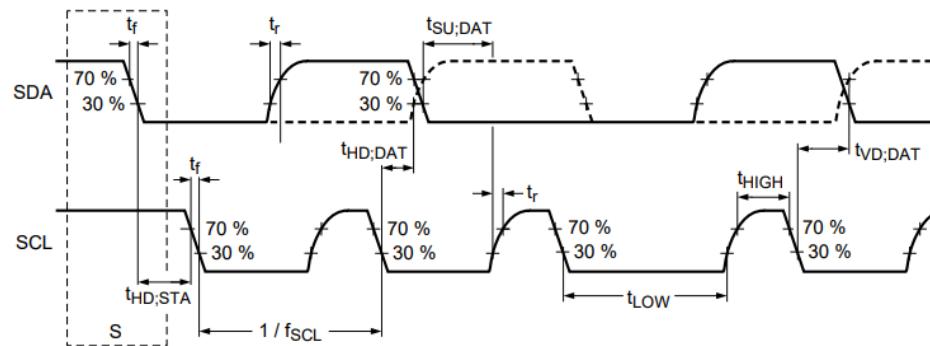
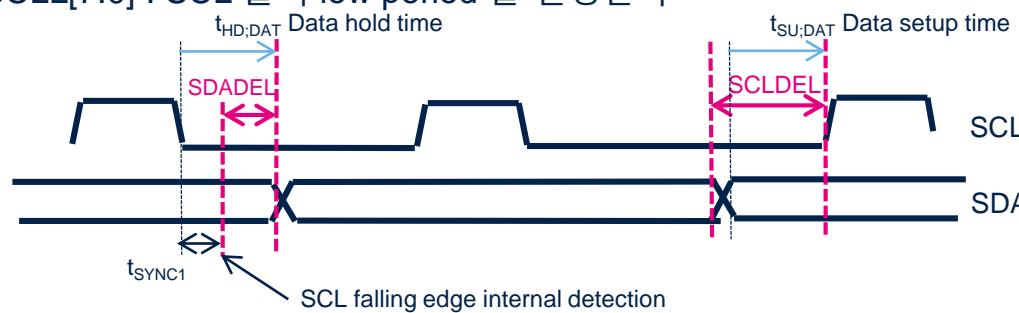
- I2Cx_ICR
 - I2C Interrupt Clear Register
 - OVRCF, BERRCF, STOPCF, NACKCF 등 에러 플래그를 clear 한다
- I2Cx_RXDR
 - I2C Receive Data Register
 - 8 bit 의 수신 데이터를 담고 있다
- I2Cx_TXDR
 - I2C Transmit Data Register
 - 8 bit 의 전송할 송신 데이터를 담고 있다

STM32G4 I2C Registers

- I2Cx_TIMINGR

- I2C Timing Register

- PRESC[3:0] : I2CCLK 를 PRESC+1 로 분주한 clock을 가지고 Data setup/hold 시간과 SCL high/low period 시간을 카운트한다
- SCLDEL[3:0] : Data setup time 을 설정한다
- SDADEL[3:0] : Data hold time 을 설정한다
- SCLH[7:0] : SCL 출력 high period 를 설정한다
- SCLL[7:0] : SCL 출력 low period 를 설정한다



$t_{SYNC1} =$

- SCL falling slope 시간 (tf) +
- Enable 된 경우, analog filter 시간 (tAF) +
- Enable 된 경우, digital filter 시간 ($tDNF$) +
- SCL synchronization to I2CCLK

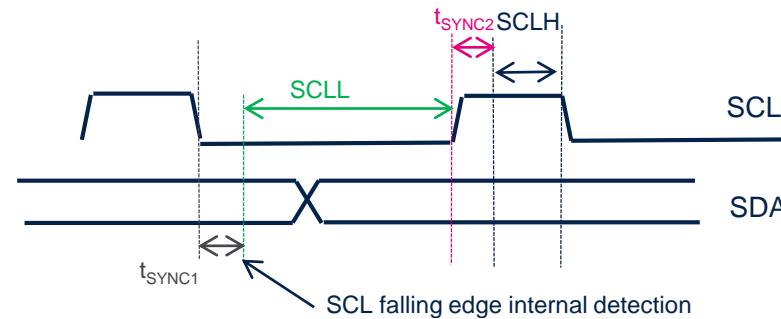
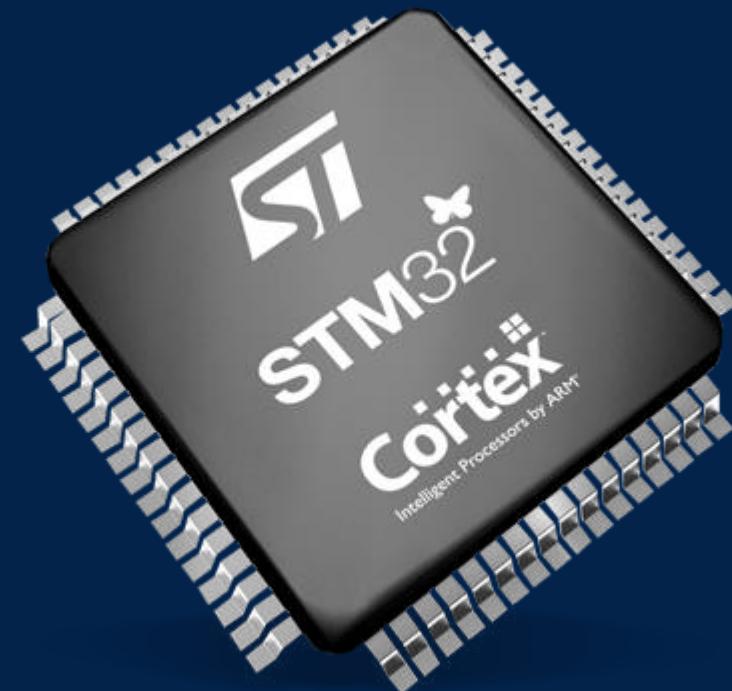


Table 380. I²C-SMBus specification data setup and hold times

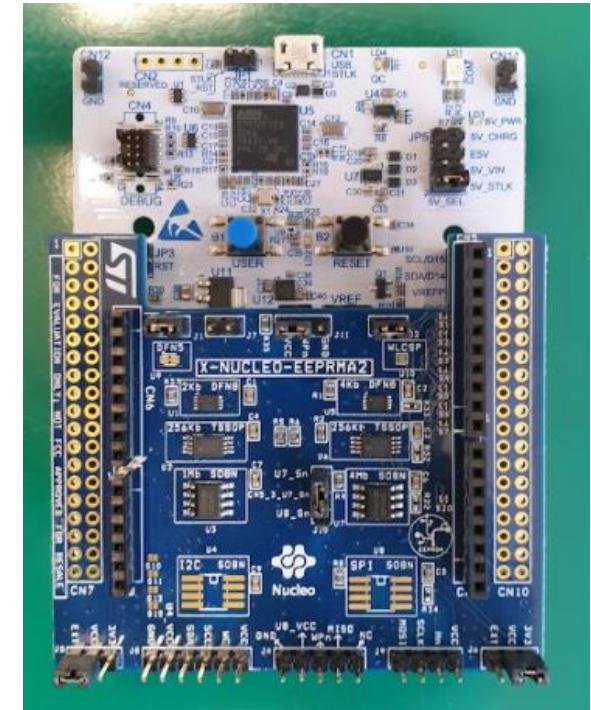
Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min.	Max	Min.	Max	Min.	Max	Min.	Max	
$t_{HD:DAT}$	Data hold time	0	-	0	-	0	-	0.3	-	μs
$t_{VD:DAT}$	Data valid time	-	3.45	-	0.9	-	0.45	-	-	ns
$t_{SU:DAT}$	Data setup time	250	-	100	-	50	-	250	-	ns
t_r	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	ns
t_f	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	ns

Hands-On Session: I2C



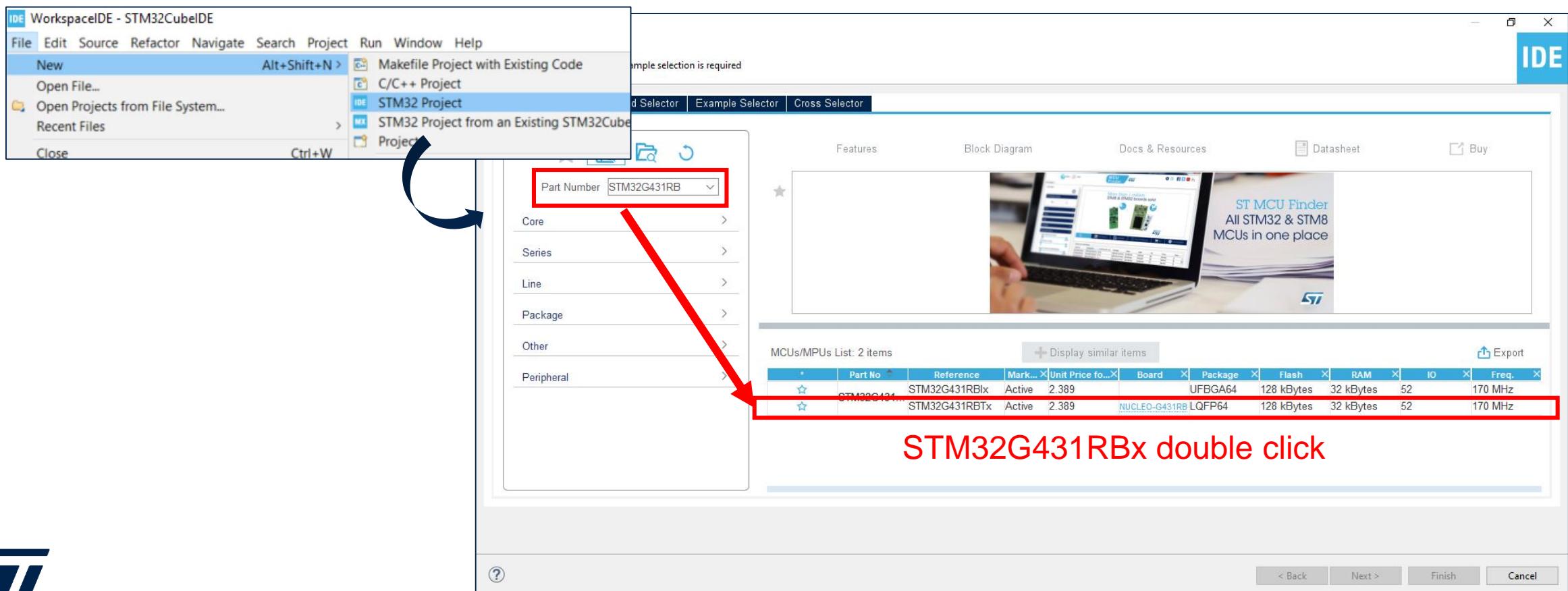
I2C Hands-On

- NUCLEO-G431RB 보드와 X-NUCLEO-EEPRMA2 와 I2C 통신
- STM32CubeIDE를 사용하여 'I2C1' 설정 후 Initialization code를 생성
- X-NUCLEO-EEPRMA2에 탑재된 EEPROM을 쓰고 읽음
- 데이터가 올바르게 써져 있는지 비교



Create STM32CubeIDE Project

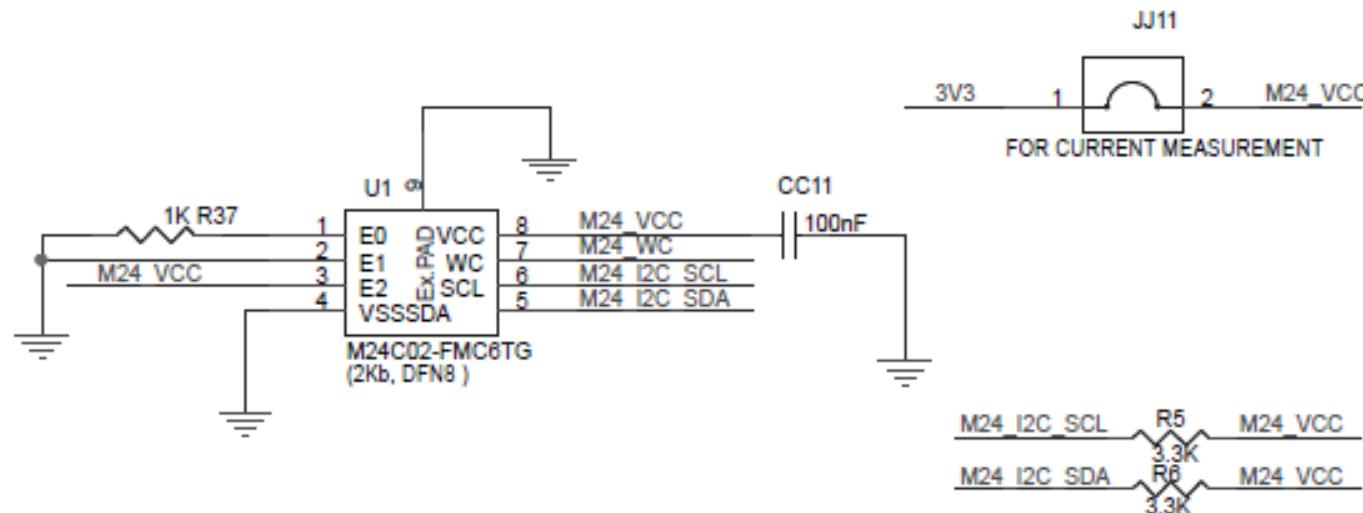
- File > New > STM32 Project
 - Part Number Search: **STM32G431RB**



- Connect to the X-NUCLEO-EEPRM A2

- **M24C02-FMC6TG**

- **M24_I2C_SCL: SPI1_SCK (CN5 – 10pin)**
 - **M24_I2C_SDA: SPI1_MISO (CN5 – 9pin)**



• WC Function

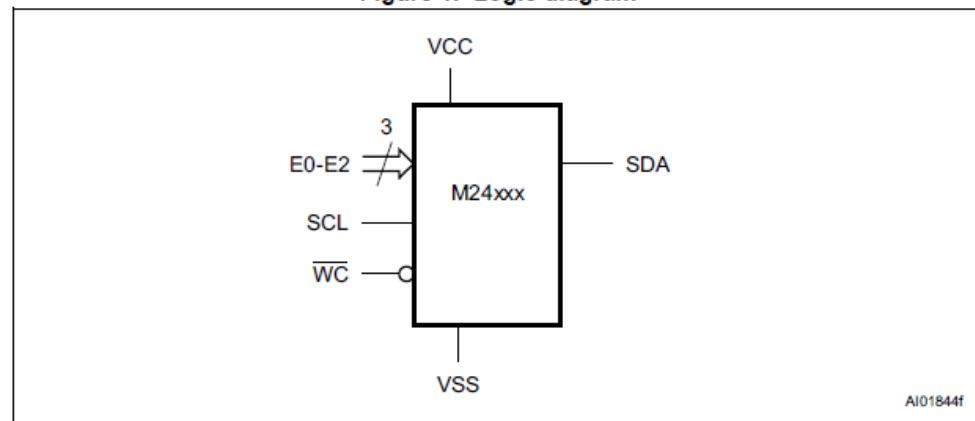
- High : Disable Write
- Low or Floating : Enable Write

Write Control (\overline{WC})

This input signal is useful for protecting the entire contents of the memory from inadvertent write operations. Write operations are disabled to the entire memory array when Write Control (WC) is driven high. Write operations are enabled when Write Control (WC) is either driven low or left floating.

When Write Control (\overline{WC}) is driven high, device select and address bytes are acknowledged, Data bytes are not acknowledged.

Figure 1. Logic diagram



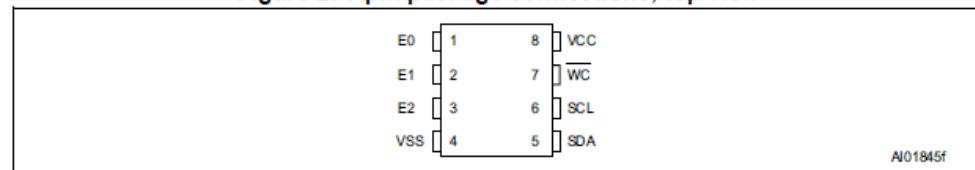
AI01844f

Table 1. Signal names

Signal name	Function	Direction
E2, E1, E0 ⁽¹⁾	Chip Enable	Input
SDA	Serial Data	I/O
SCL	Serial Clock	Input
\overline{WC}	Write Control	Input
V _{cc}	Supply voltage	-
V _{ss}	Ground	-

1. Signal not connected in the DFN5 package.

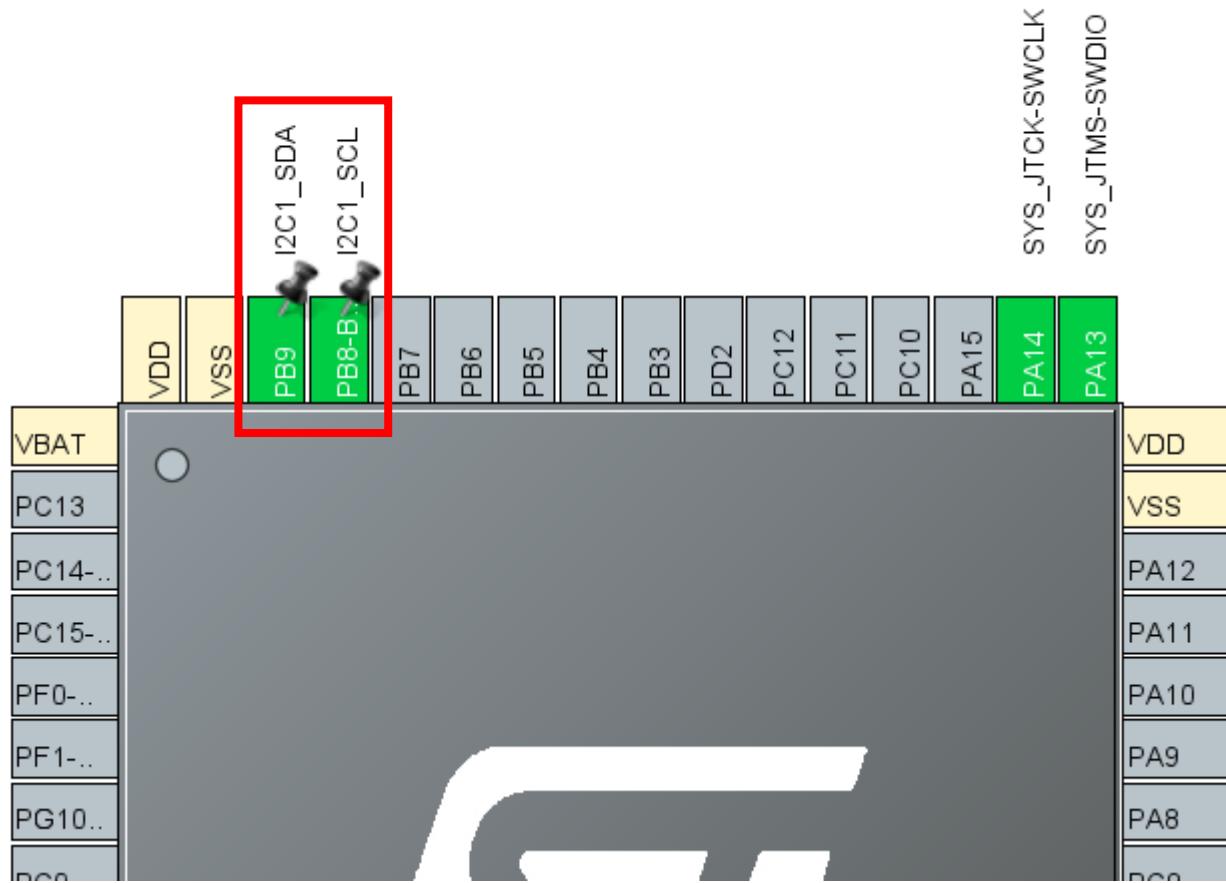
Figure 2. 8-pin package connections, top view



AI01845f

Peripheral and pin-out

- System Core → SYS → Debug → Serial Wire
- Connectivity → I2C1 → I2C [I2C]
 - PB8 : I2C1_SCL
 - PB9 : I2C1_SDA

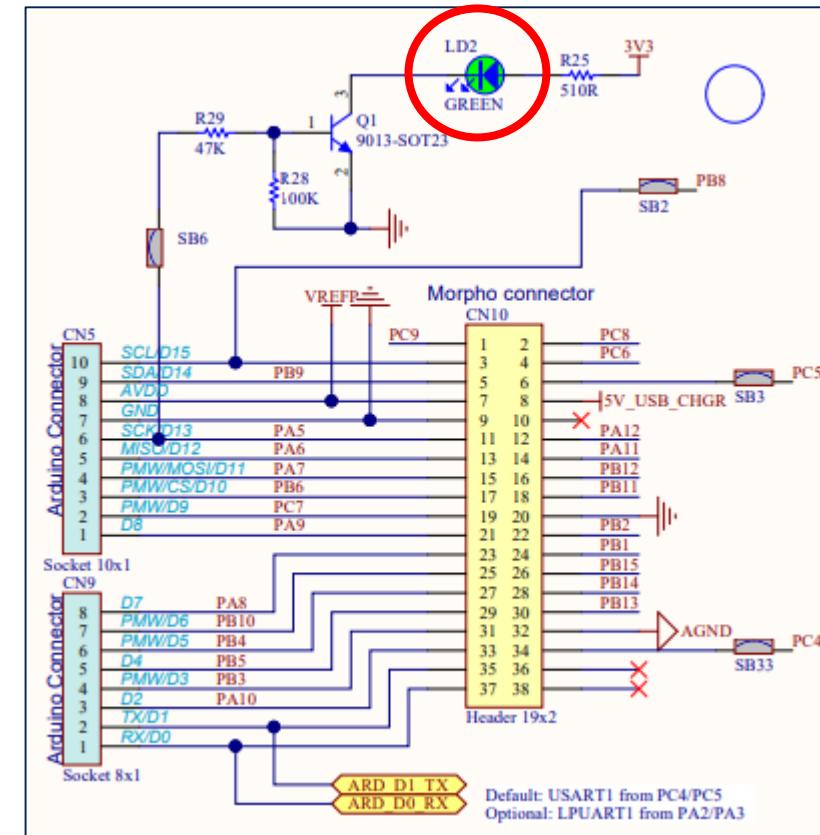


Configuration

- Indicators

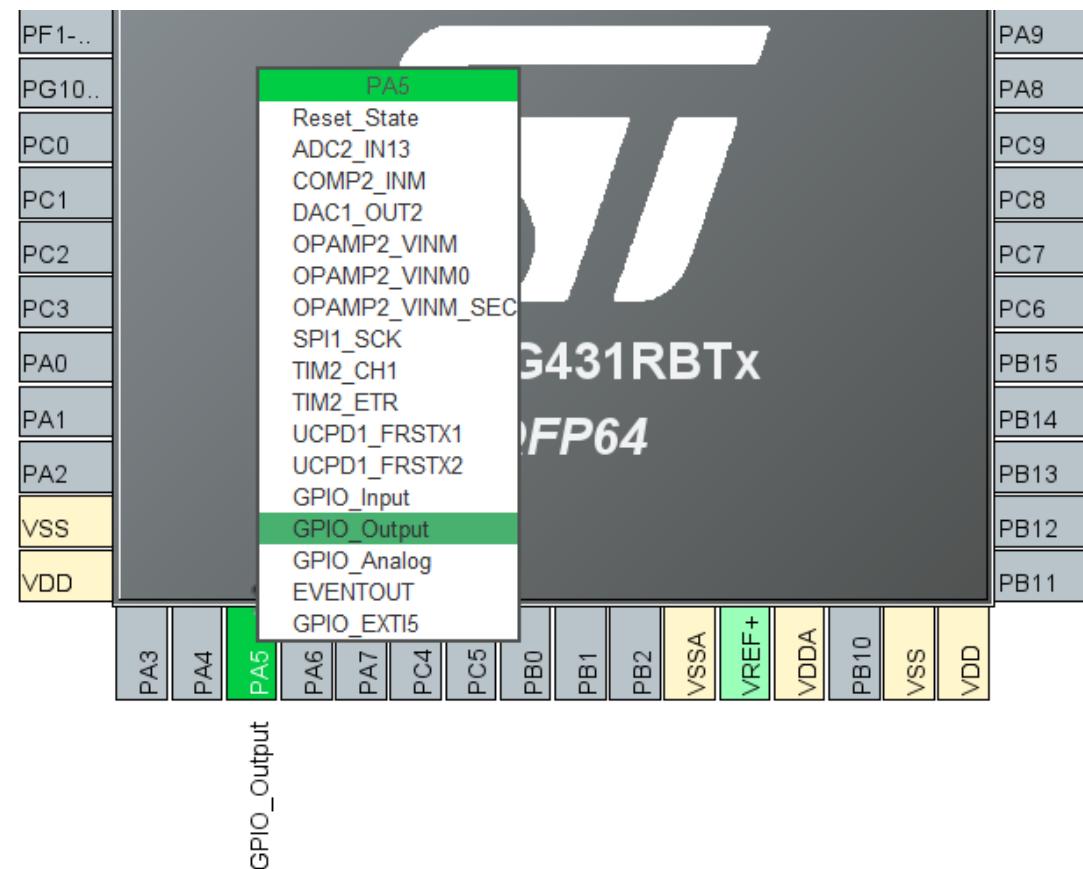
- 데이터 확인용으로 LD2 점멸 시킬 예정
- LEDs (GPIO_Output): User LED LD2 (PA5)

Solder bridge control	Solder bridge (SB)	State ⁽¹⁾	Description ⁽¹⁾
100 nF on PG10-NRST	SB1	ON	100 nF capacitor grounded to PG10-NRST of the STM32G4
		OFF	100 nF capacitor disconnected from PG10-NRST of the STM32G4
PB8 on ARD_D14	SB2	ON	PB8 connected to ARDUINO® D14
		OFF	PB8 not connected to ARDUINO® D14
PC5 on ST morpho	SB3	ON	PC5 connected to ST morpho CN10 pin 6
		OFF	PC5 not connected to ST morpho CN10 pin 6
PB8 on Morpho	SB4	ON	PB8 connected to ST morpho CN7 pin 7
		OFF	PB8 not connected to ST morpho CN7 pin 7
3.3 V LDO output	SB5	ON	U12 LDO output provides 3.3 V
		OFF	U12 LDO output does NOT provide 3.3 V, user must connect an external 3.3 V source.
User LED LD2	SB6	ON	User LED driven by PA5 (ARD_D13)
		OFF	User LED not driven



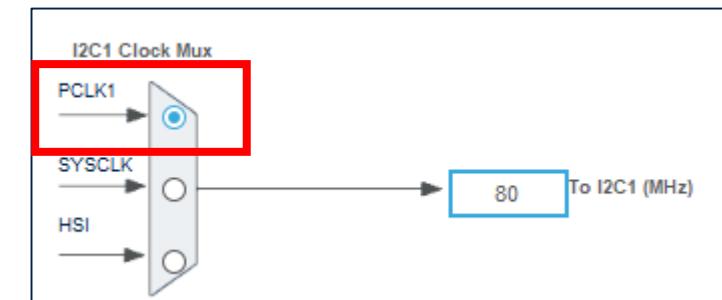
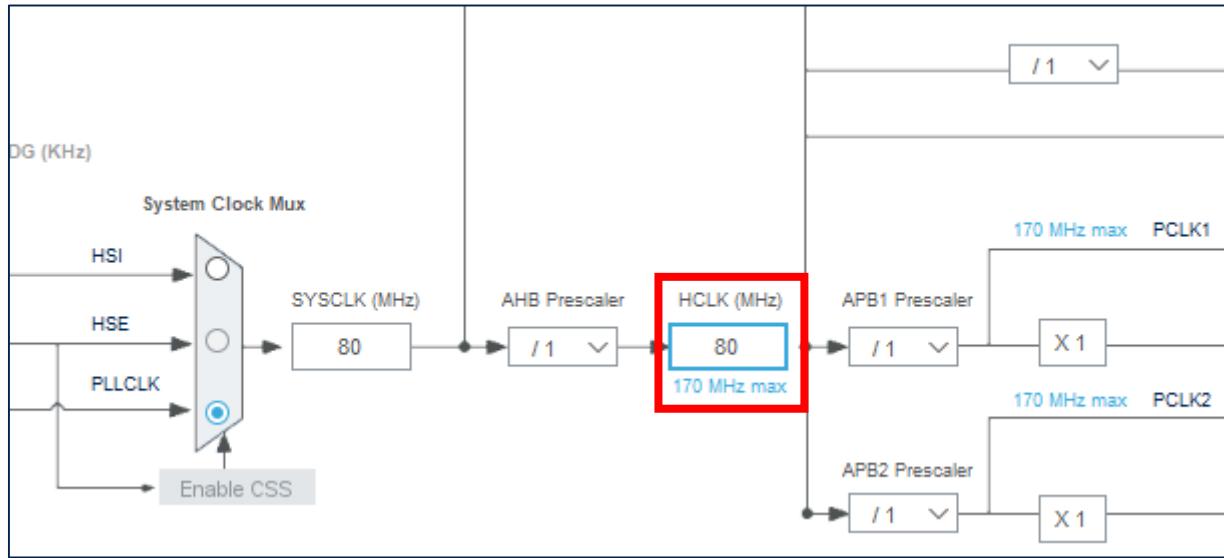
Configuration

- Pinout & Configuration
 - Pinout view > PA5: GPIO_Output



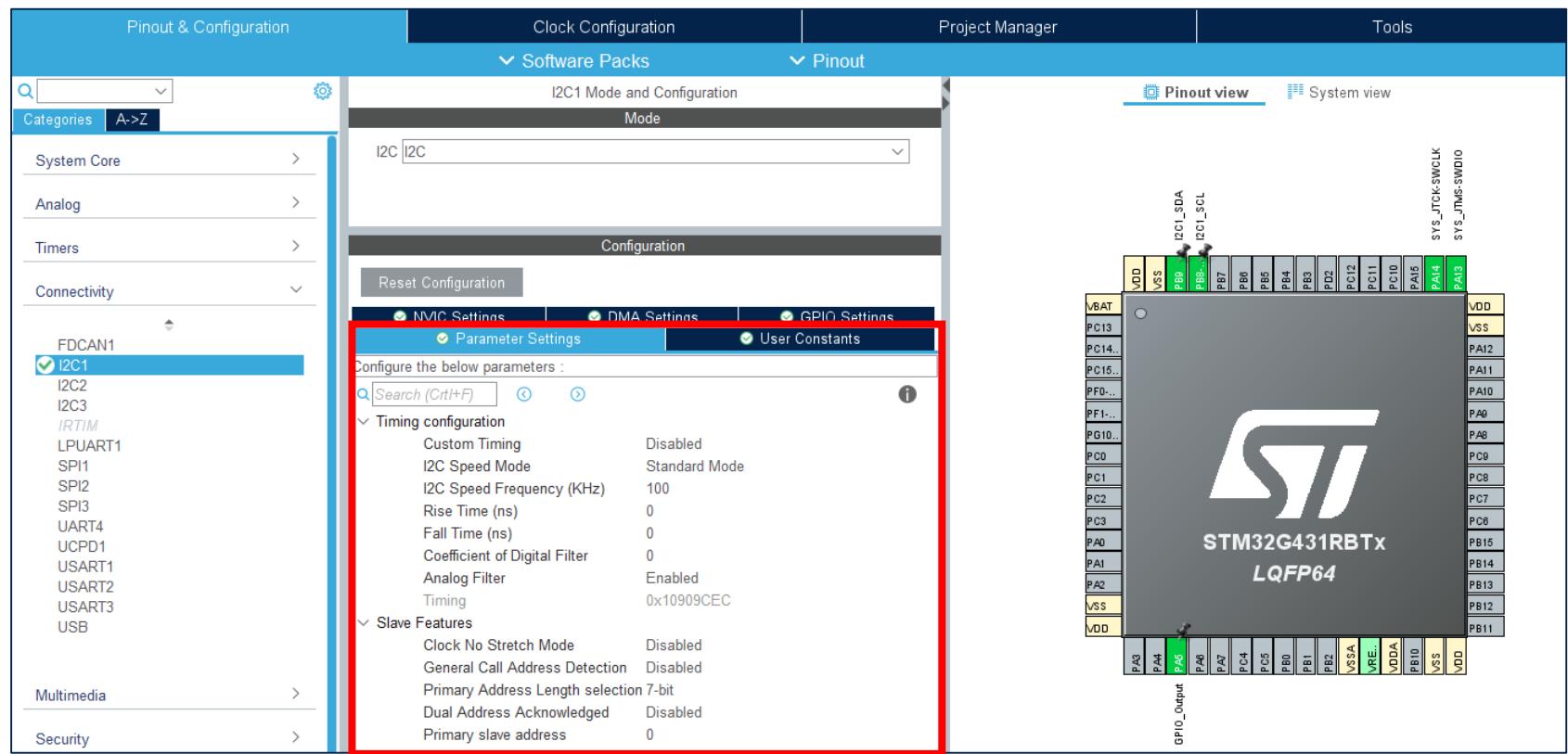
Clock configuration

- STM32G431RBx 클럭 설정
 - System clock : 80 MHz
 - PLL clock source : HSI 16MHz
 - APB1 Prescaler : /1, APB2 Prescaler : /1
 - Cortex system timer Prescaler : / 1
 - I2C1 Clock Mux : PCLK1, 80MHz



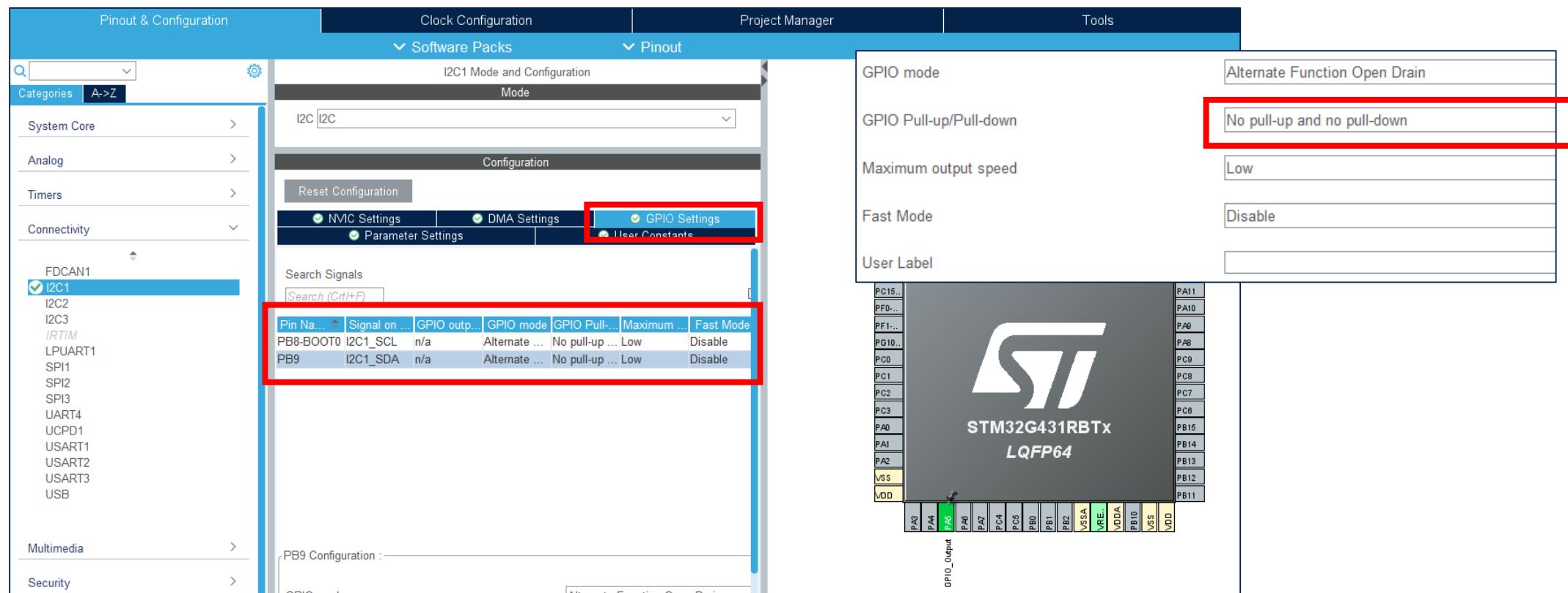
I2C configuration

- I2C1 Parameter 설정
 - I2C Speed Mode : Standard Mode
 - I2C Speed Frequency (KHz) : 100 (자동 설정)
 - Rise Time (ns) : 0
 - Fall Time (ns) : 0



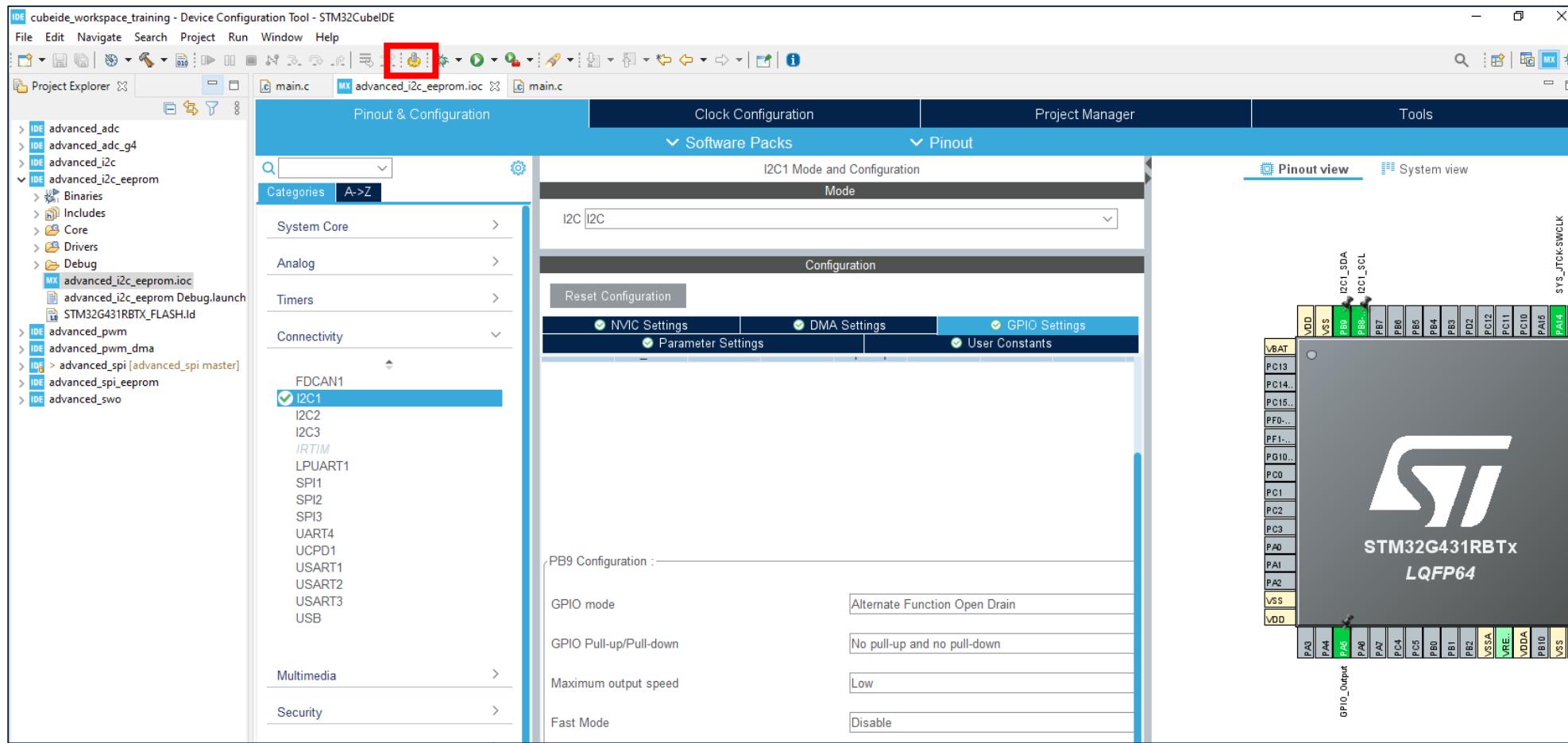
I2C configuration

- I2C1 GPIO 설정
 - GPIO Pull-up / Pull-down
 - 회로도상에 외부 pull-up 저항이 있으므로 SDA, SCL 두핀 모두 내부 pull-up/down 을 disable 한다 (No pull-up and no pull-down)



Code Generation

- Project > Generate Code (Alt + K)



Code Description

- EEPROM defines

```
/* USER CODE BEGIN PD */  
#define EEPROM_ADDRESS          (0xA8)    /* EEPROM M24LR64 Address */  
#define EEPROM_PAGESIZE        (16)      /* EEPROM M24LR64 used */  
/* Maximum Timeout values for flags waiting loops. These timeouts are not based  
on accurate values, they just guarantee that the application will not remain  
stuck if the I2C communication is corrupted.  
You may modify these timeout values depending on CPU frequency and application  
conditions (interrupts routines ...). */  
#define I2C_XFER_TIMEOUT_MAX   (300)  
#define EEPROM_MAX_TRIALS     (3000U)  
  
/* USER CODE END PD */
```

Code Description

- Buffer used for transmission and reception

```
/* USER CODE BEGIN PV */  
/* Buffer used for transmission */  
/* Size of Transmission buffer */  
#define COUNTOF(__BUFFER__)    (sizeof(__BUFFER__) / sizeof(*(__BUFFER__)))  
#define TXBUFFERSIZE          (COUNTOF(aTxBuffer) - 1)  
/* Size of Reception buffer */  
#define RXBUFFERSIZE           TXBUFFERSIZE  
  
uint8_t aTxBuffer[] = /* I2C HAL API EEPROM driver example: \  
                         This firmware provides a basic example of how to use the I2C HAL API based and \  
                         an associate I2C EEPROM driver to communicate with M24C02-FMC6TG EEPROM */;  
uint8_t aRxBuffer[RXBUFFERSIZE];  
  
/* USER CODE END PV */
```

Code Description

- Private function prototypes

```
/* USER CODE BEGIN PFP */  
static uint16_t Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength);  
  
/* USER CODE END PFP */
```

Code Description

- Global 변수선언

```
/* USER CODE BEGIN 0 */
uint16_t Memory_Address;
int Remaining_Bytes;
__IO uint32_t I2C_Tx_Complete_Flag = 0;
__IO uint32_t I2C_Rx_Complete_Flag = 0;
__IO uint32_t I2C_Error_Flag = 0;

/* USER CODE END 0 */
```

Code Description

- main 함수 내 Local 변수 선언

```
/* USER CODE BEGIN 1 */  
  uint32_t Memory_Write_Trials = 0;  
  uint32_t Memory_Read_Trials = 0;  
/* USER CODE END 1 */
```

Code Description

- 데이터 검증을 위한 비교 함수

```
/* USER CODE BEGIN 4 */
static uint16_t Buffercmp(uint8_t* pBuffer1, uint8_t* pBuffer2, uint16_t BufferLength)
{
    while (BufferLength--)
    {
        if ((*pBuffer1) != *pBuffer2)
        {
            return BufferLength;
        }
        pBuffer1++;
        pBuffer2++;
    }

    return 0;
}
/* USER CODE END 4 */
```

Code Description - 계속

- main 함수 내 User code 추가

```
/* USER CODE BEGIN 2 */
/* Initialize Remaining Bytes Value to TX Buffer Size */
Remaining_Bytes = TXBUFFERSIZE;
/* Initialize Memory address to 0 since EEPROM write will start from address 0 */
Memory_Address = 0;

while(Remaining_Bytes > 0)
{
    do
    {
        /* Write EEPROM_PAGESIZE */
        if (HAL_I2C_Mem_Write(&hi2c1 , (uint16_t)EEPROM_ADDRESS, Memory_Address, I2C_MEMADD_SIZE_8BIT, (uint8_t*)(aTxBuffer + Memory_Address), EEPROM_PAGESIZE, 10000)!= HAL_OK)
        {
            if (HAL_I2C_GetError(&hi2c1) != HAL_I2C_ERROR_AF)
            {
                /* Writing process Error */
                Error_Handler();
            }
        }

        /* Increment Trials */
        Memory_Write_Trials++;
    }
    while((Memory_Write_Trials < EEPROM_MAX_TRIALS) && (HAL_I2C_GetError(&hi2c1) == HAL_I2C_ERROR_AF));

    /* Clear Trials */
    Memory_Write_Trials = 0;

    /* Wait for the end of the transfer */
    while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY)
    {

    }

    /* Check if the EEPROM is ready for a new operation */
    while (HAL_I2C_IsDeviceReady(&hi2c1, EEPROM_ADDRESS, 10, 300) == HAL_TIMEOUT);
```

Code Description

```
/* Wait for the end of the transfer */
while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY)
{
}

/* Update Remaining bytes and Memory Address values */
Remaining_Bytes -= EEPROM_PAGESIZE;
Memory_Address += EEPROM_PAGESIZE;
}

/*##-3- Start reading process #####*/
do
{
    if(HAL_I2C_Mem_Read(&hi2c1 , (uint16_t)EEPROM_ADDRESS, 0, I2C_MEMADD_SIZE_8BIT, (uint8_t*)aRxBuffer, RXBUFFERSIZE, 10000)!= HAL_OK)
    {
        if (HAL_I2C_GetError(&hi2c1) != HAL_I2C_ERROR_AF)
        {
            /* Reading process Error */
            Error_Handler();
        }
    }

    /* Increment Trials */
    Memory_Read_Trials++;
}
while((Memory_Read_Trials < EEPROM_MAX_TRIALS) && (HAL_I2C_GetError(&hi2c1) == HAL_I2C_ERROR_AF));

/* Clear Trials */
Memory_Read_Trials = 0;

/* Wait for the end of the transfer */
while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY)
{
}

/*##-4- Compare the sent and received buffers #####*/
if(Buffercmp((uint8_t*)aTxBuffer, (uint8_t*)aRxBuffer, RXBUFFERSIZE))
{
    /* Processing Error */
    Error_Handler();
}
/* USER CODE END 2 */
```

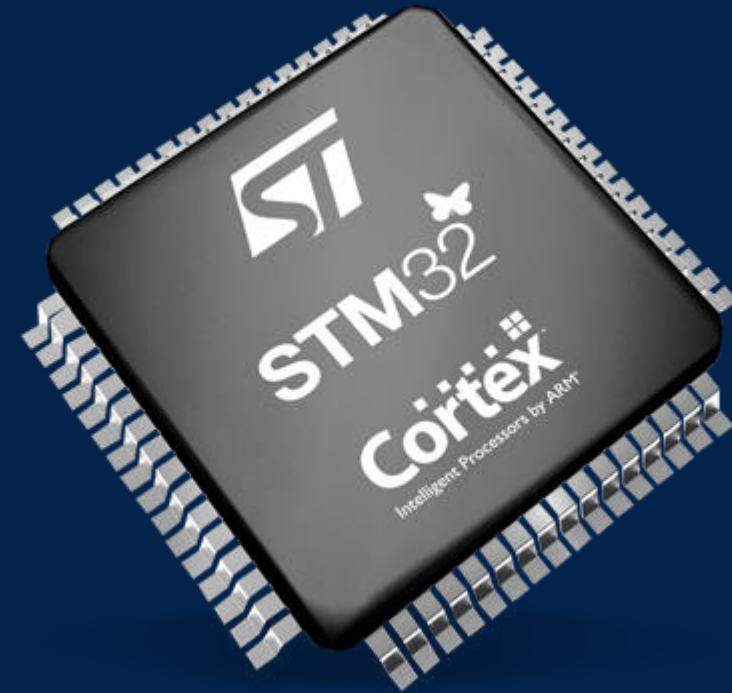
Code Description

- 데이터 읽기/쓰기가 올바를 시 LED LD2 100ms 마다 점멸

```
/* USER CODE BEGIN WHILE */
while (1)
{
    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
    HAL_Delay(100);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

ADC(Analog-to-Digital Converter)

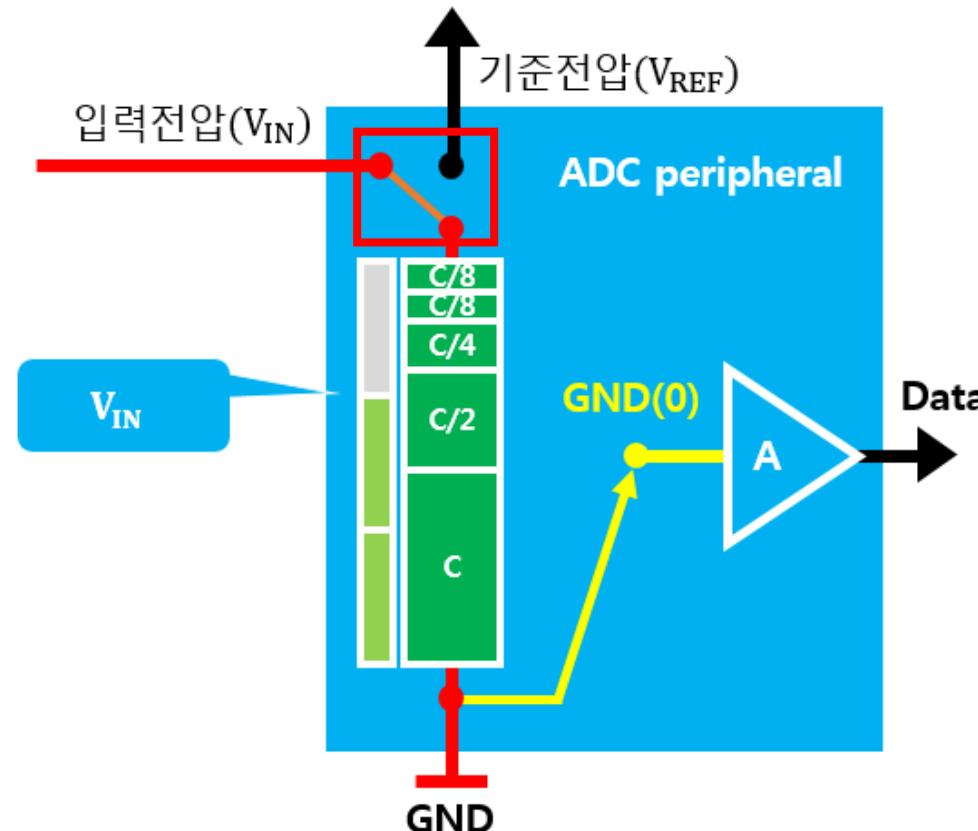


What is ADC

- ADC (Analog to Digital Converter)
 - Analog 입력 신호의 전압 레벨을 레퍼런스 전압 기준의 Digital 숫자로 변환하는 기능
 - Resolution(분해능)
 - 입력 신호 전압의 크기를 몇 단계로 세분화하는지 분해능을 의미하며 8bit, 10bit, 12bit, 16bit 등 다양하게 사용된다 ($10\text{bit} = 2^{10} = 1024$)
 - 레퍼런스 전압이 3.3V 이고 10bit resolution 의 ADC 를 사용하는 경우 ADC 1 값으로 측정할 수 있는 단위는 $3.3\text{V} / (2^{10}) = 3.2 \text{ mV}$ 가 된다
 - Sampling Rate
 - Analog 입력 신호를 얼마의 주기를 갖는 연속된 Digital 숫자로 변환하는지를 나타낸다
 - 한 주기마다 Sample and Hold(conversion) 과정을 거치며 Sampling 시간 동안 입력 신호로 내부 Capacitor 를 충전하고 Hold 시간 동안은 입력 신호를 분리한 상태에서 Digital 숫자로 변환 과정을 거친다
 - Types
 - ADC는 다양한 방법이 있으며 Success Approximation Register (SAR), Sigma Delta ADC 등이 있다

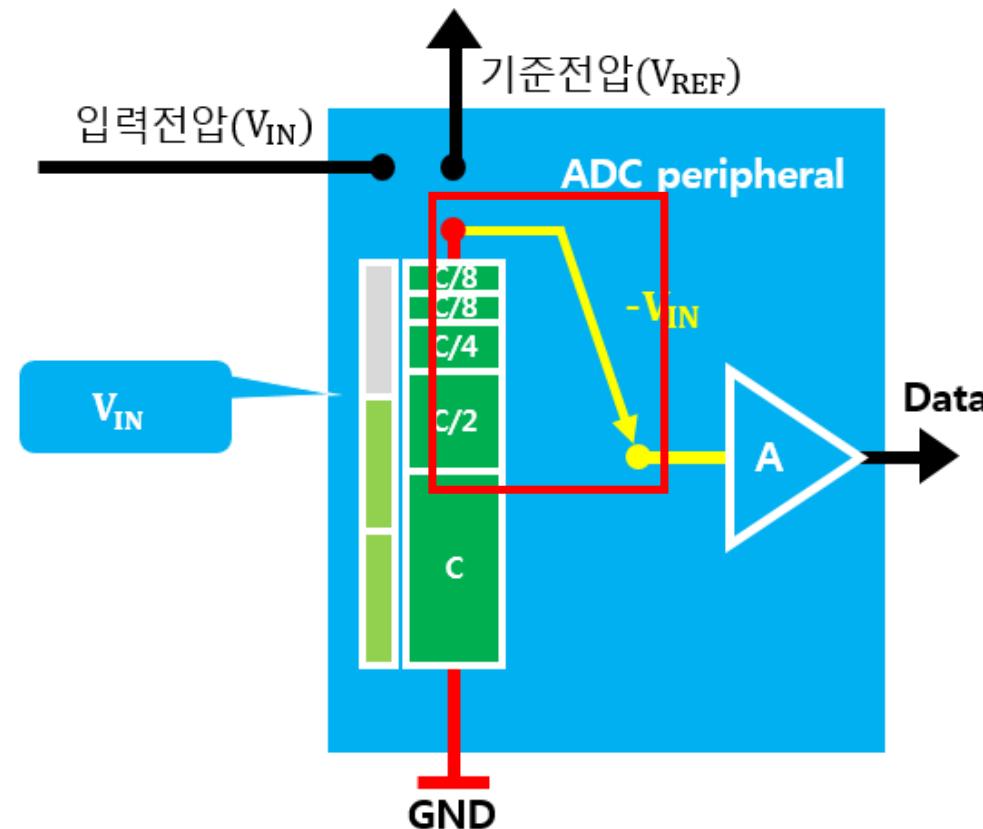
SAR ADC principle

- Sample State
 - 4-bit 비교 검출 예시
 - 입력전압(V_{IN})을 MCU 내부 ADC peripheral의 캐패시터(2C)로 충전



SAR ADC principle

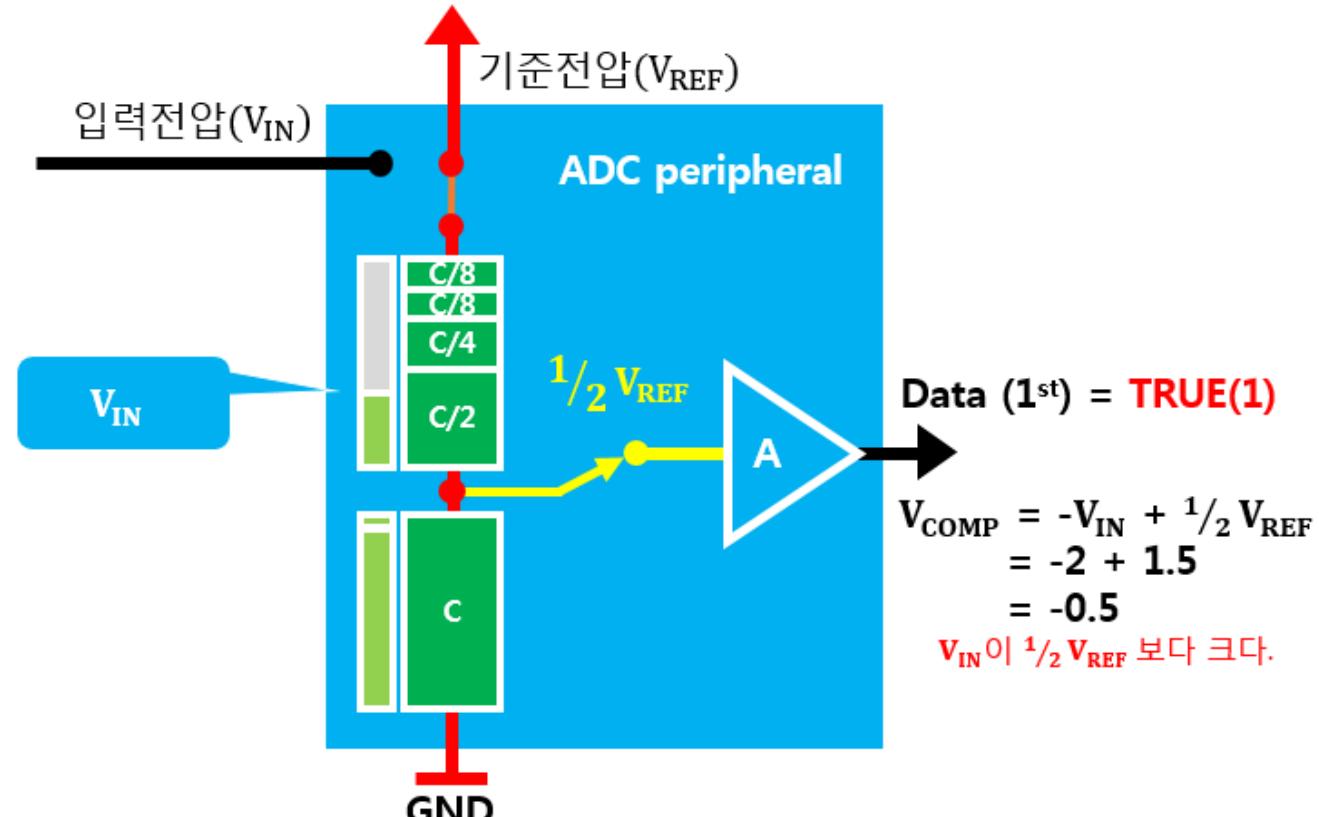
- Hold state
 - 기준전압 (V_{ref}) 과의 비교 및 변환을 위해 비교기 (Comparator) 입력으로 전환



SAR ADC principle

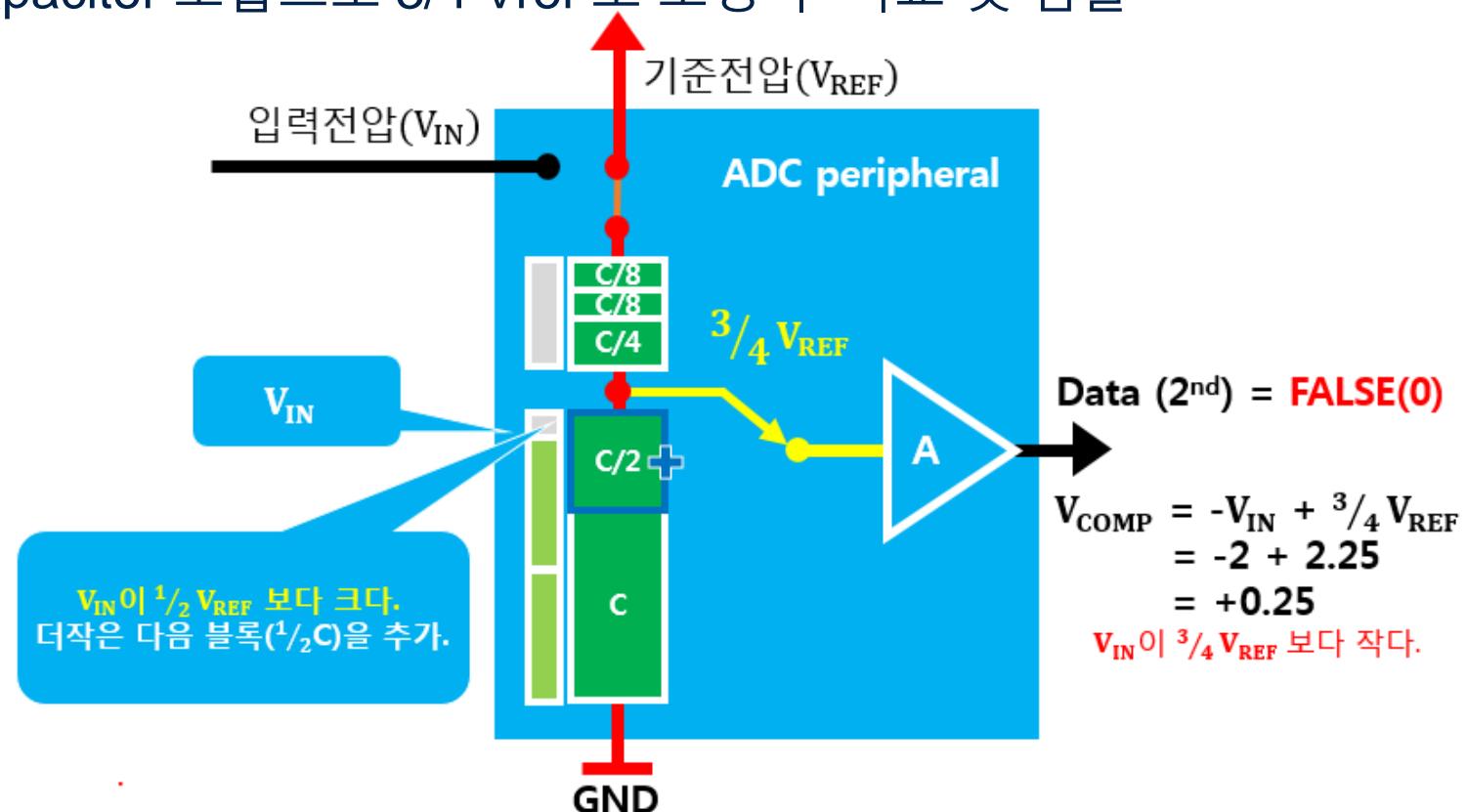
- Conversion – 1st Step

- 이분 검색(Binary search)과 동일한 방식으로 MSB 부터 확인
- 최상위 비트(MSB) 부터 입력전압 V_{IN} 이 기준전압 V_{REF} 의 $1/2$ 을 넘어서는지 확인



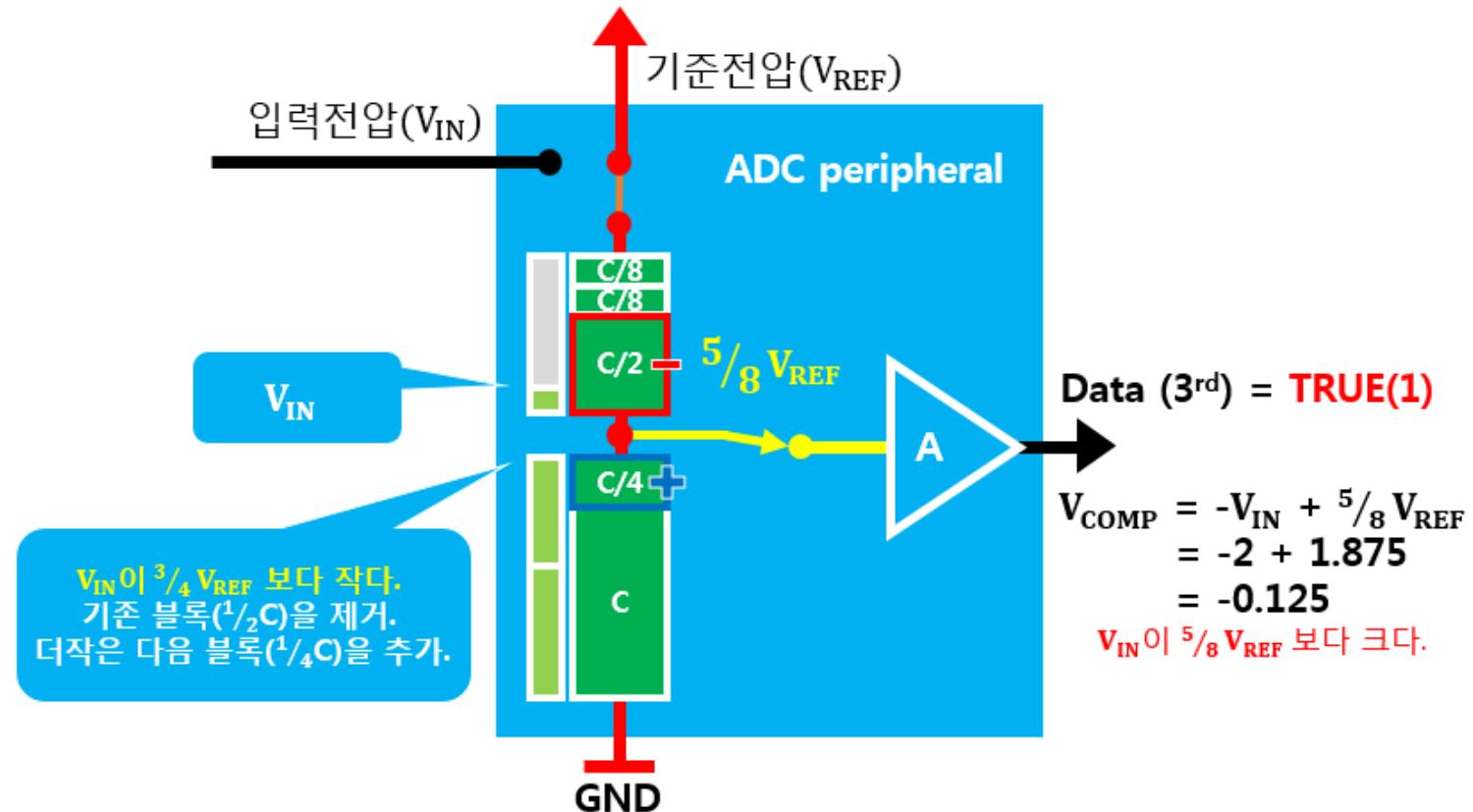
SAR ADC principle

- Conversion – 2nd Step
 - 내부 병렬 Capacitor 조합으로 $3/4 V_{REF}$ 로 조정 후 비교 및 검출



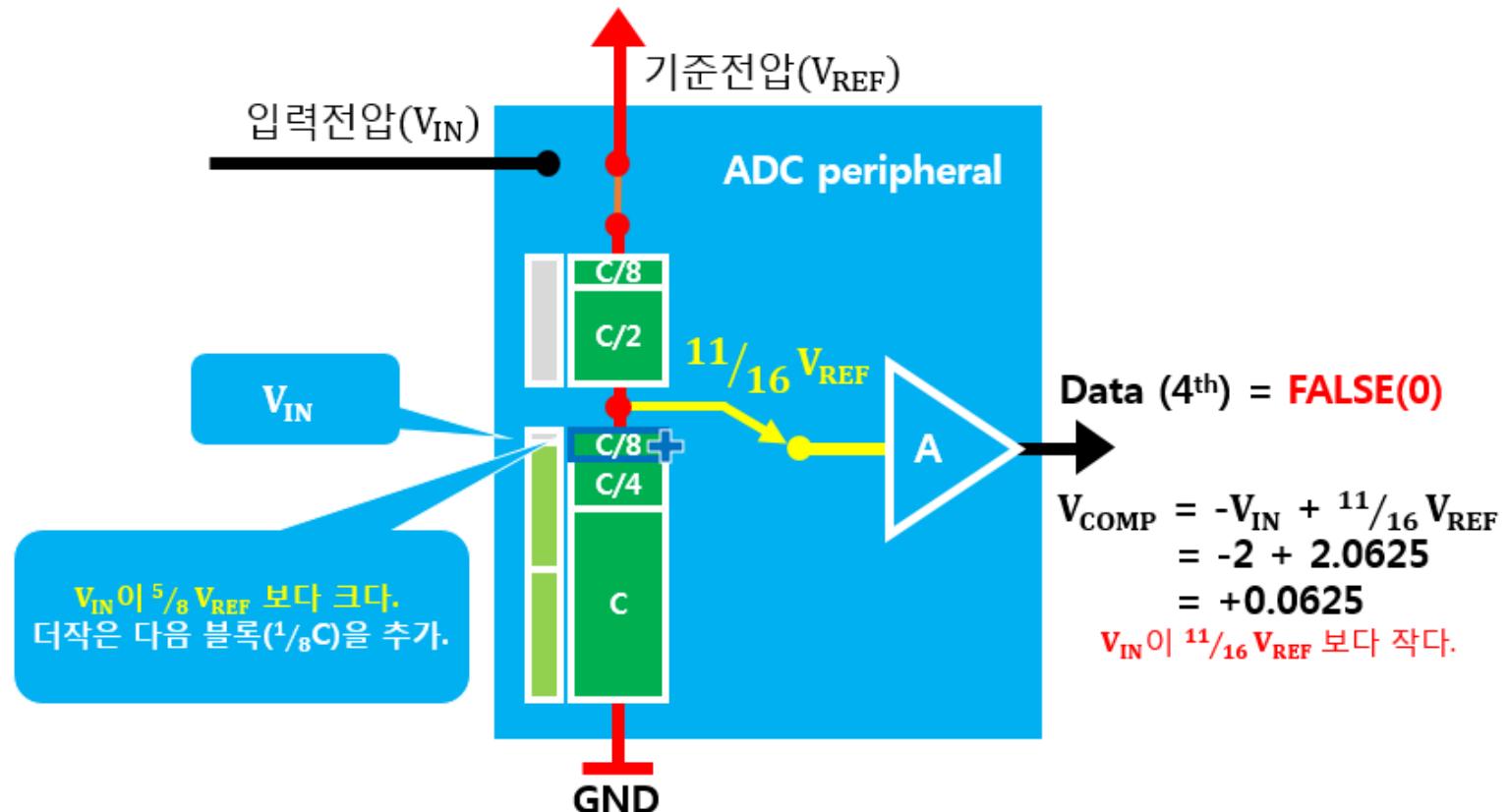
SAR ADC principle

- Conversion – 3rd Step
 - V_{IN} 와 $5/8 V_{REF}$ 비교 및 검출



SAR ADC principle

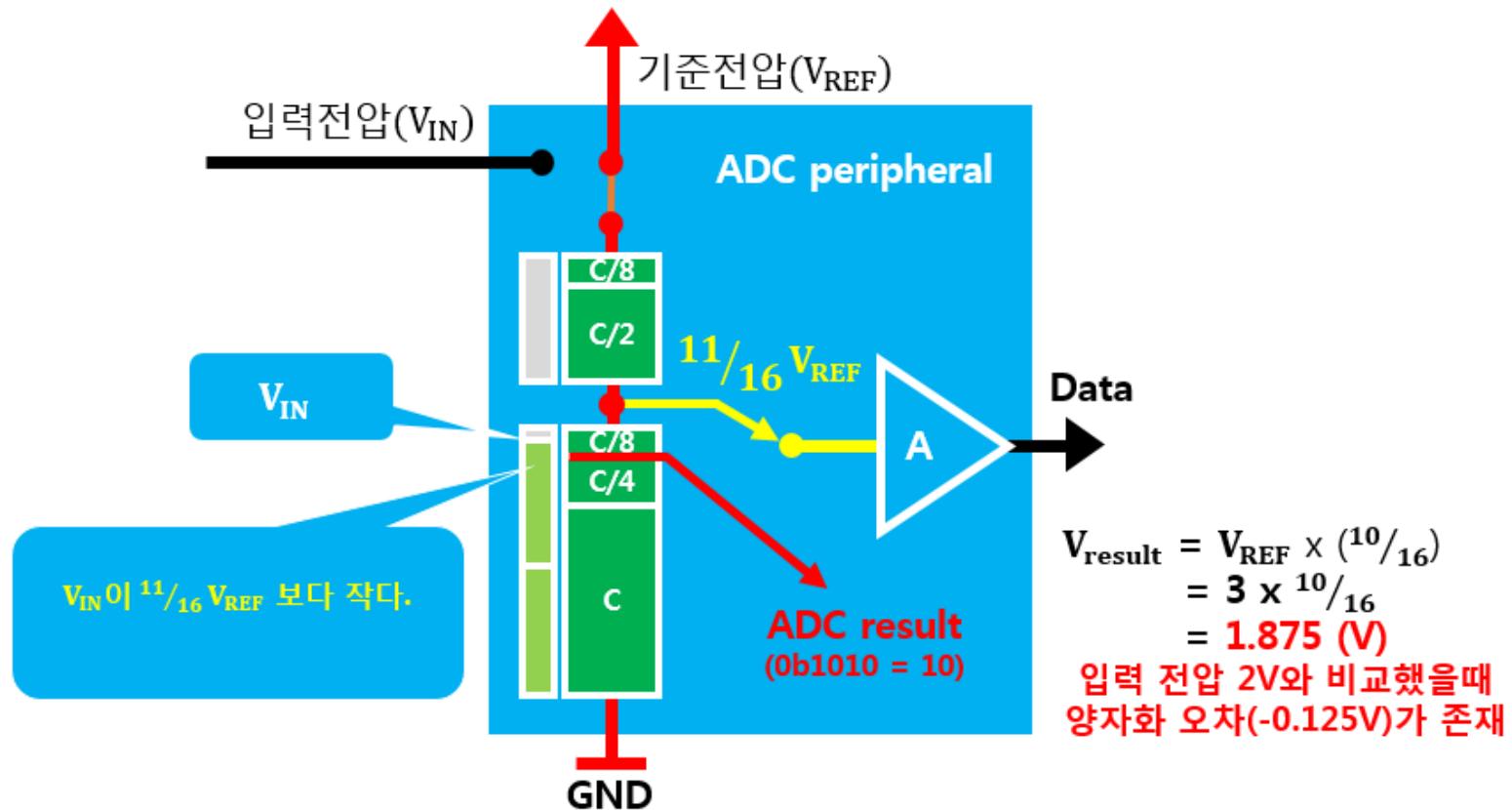
- Conversion – 4th Step
 - V_{IN}와 11/16 V_{REF} 비교 및 검출



SAR ADC principle

- Conversion – Result

- 4-bit ADC로 4번의 비교 검출, 실제 12-bit ADC인 경우에는 12번의 비교 검출 변환 과정을 가짐

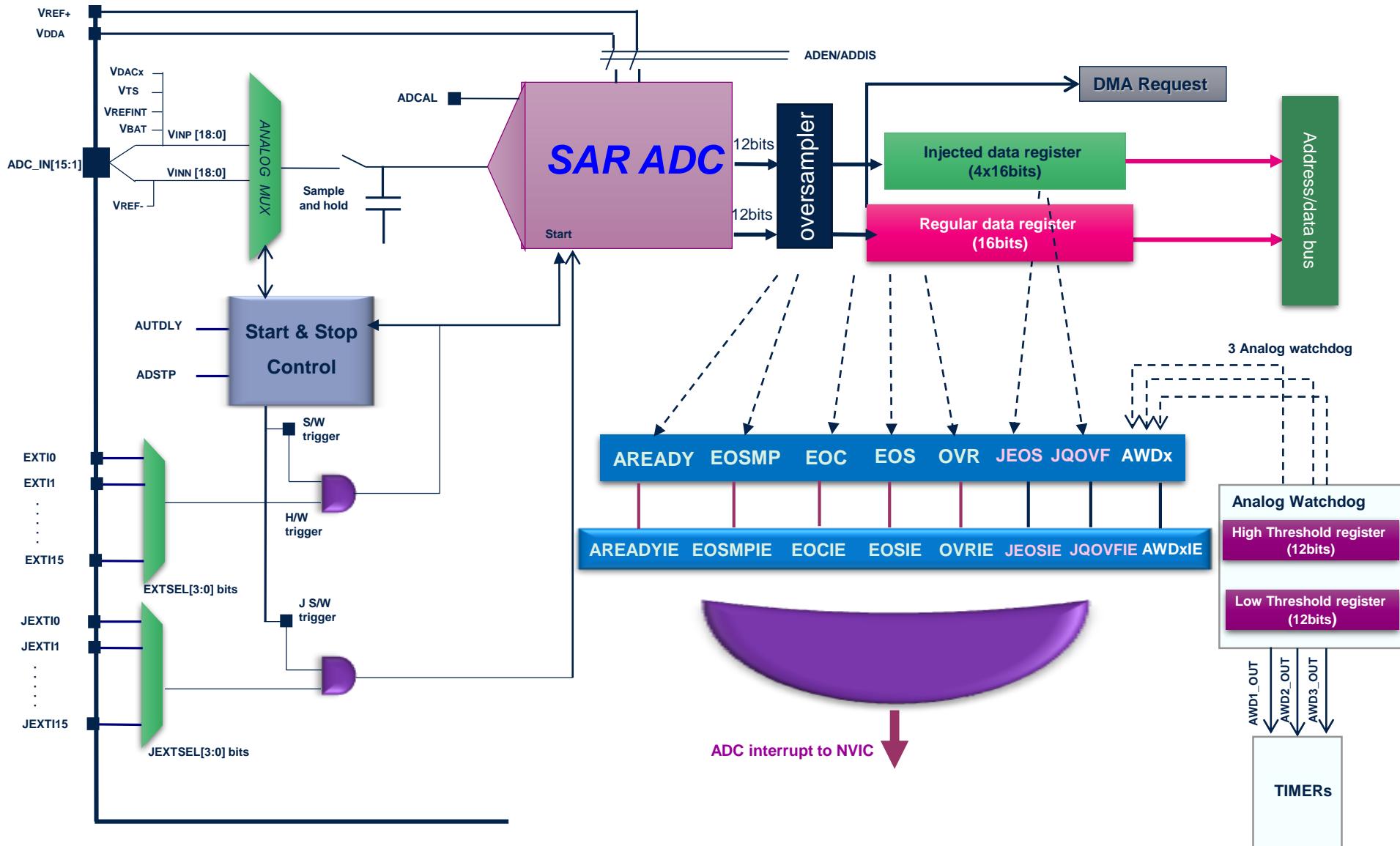


- 특징

- SAR 방식의 2 개의 ADC 블록 지원
 - 각 ADC 블록별로 최대 23 개의 외부 ADC 입력 채널 제공
 - ADC 블록 1번과 2번은 dual 모드로 동기 맞춰서 동작 가능 (Master / Slave)
- 다양한 종류의 ADC 입력 채널
 - Single ended 와 differential 외부 입력 채널
 - Regular 와 Injected 입력 채널
 - 5개의 내부 입력 채널 (Temperature sensor, VREFINT, VBAT/3, DAC1/DAC2)
- Programmable 블록 별 6, 8, 10, 12bit 샘플링 resolution 지원
- Programmable 채널 별 샘플링 시간 지원
 - 2.5 cycle, 6.5 cycle, 12.5 cycle, ..., 247.5 cycle, 640.5 cycle
 - 12bit resolution, 2.5 cycle 로 일반 sampling 시 최대 4 MSPS 지원
 - 12bit resolution, 2.5 cycle 로 interleaved sampling 시 최대 8 MSPS 지원

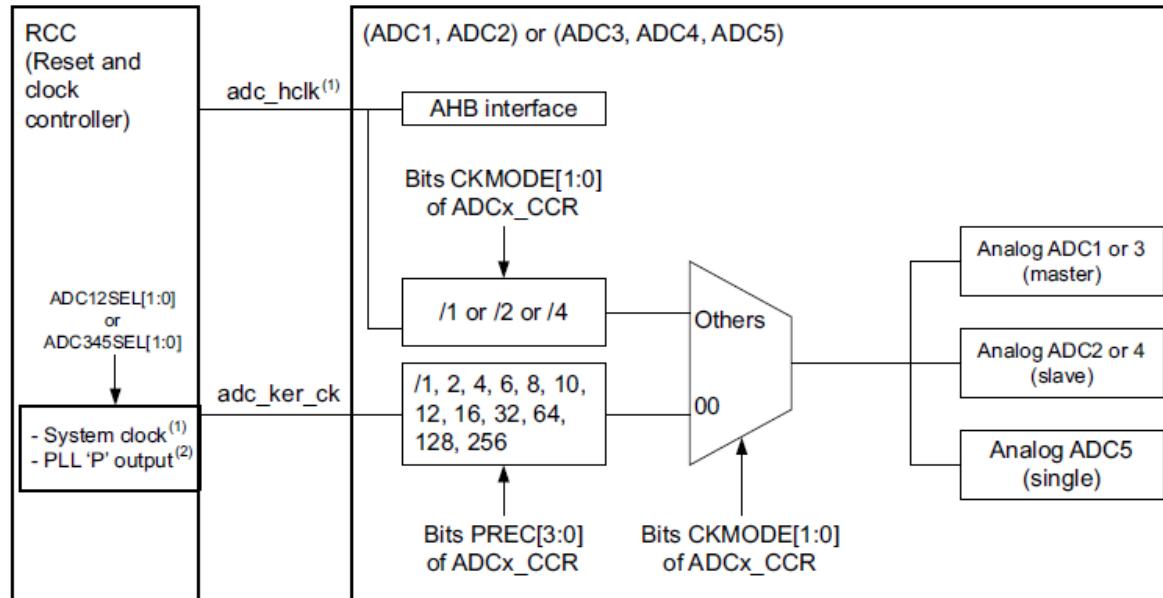
- 소프트웨어 및 하드웨어 내부 (타이머) 외부 (I/O) 샘플링 시작 트리거 지원
- Sequencer 와 Single, Continuous, Discontinuous conversion 모드 지원
- Analog watchdog 기능 지원
- Oversampler 지원
- DMA 와 Interrupt 지원
- 내부 OPAMP와 연결 지원

Block diagram



Clock Source

- Synchronous clock
 - HCLK 클럭을 ADC 내부 분주기 (prescaler)로 /1, /2, /4 분주 한 클럭이 최종 ADCCLK 클럭으로 사용된다
- Asynchronous clock
 - ADC12_CK(or ADC345) 클럭을 ADC 내부 분주기 (prescaler)로 /1, /2, /4, /8, /10, /12, /18, /32, /64, /128, /256 분주 한 클럭이 최종 ADCCLK 클럭으로 사용된다
 - ADC12_CK 클럭의 소스는 SYSCLK, PLL'P'에서 선택한다

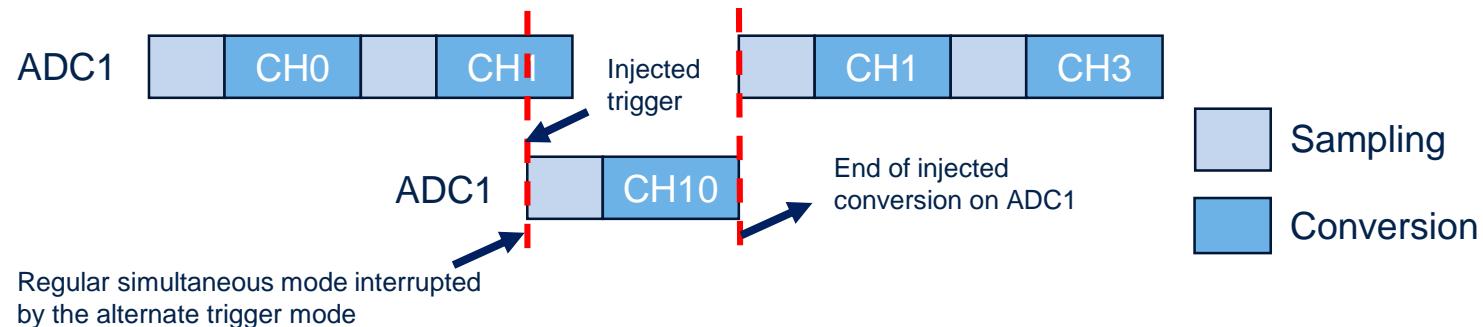


Total Conversion Time

- $(\text{Sampling cycle} + \text{Conversion cycle}) \times t_{\text{ADCCLK}}$
 - Sampling cycle
 - 2.5, 6.5, 12.5, 24.5, 47.5, 92.5, 247.5, 640.5 cycle 중에서 채널별로 선택한다
 - Conversion cycle
 - 12bit resolution = 12.5 cycle, 10bit resolution = 10.5 cycle, 8bit resolution = 8.5 cycle, 6bit resolution = 6.5 cycle 중에서 블록 공통으로 선택한다
 - tADCCLK
 - 만약 ADCCLK 가 60 MHz 라면 t_{ADCCLK} 는 $1/60 \text{ MHz} = 16.6 \text{ ns}$ 가 된다
- Total Conversion Time 계산 예시
 - 2.5 sampling, 12bit resolution, 60 MHz ADCCLK 의 total conversion time
 - $(2.5 \text{ cycle} + 12.5 \text{ cycle}) \times 16.6 \text{ ns} = 249 \text{ ns}$
 - Continuous conversion 모드로 바로 다시 시작하기를 반복하면 $1/249 \text{ ns} = 4.01 \text{ Msps}$
 - 2.5 sampling, 10bit resolution, 80 MHz ADCCLK 의 total conversion time
 - $(2.5 \text{ cycle} + 10.5 \text{ cycle}) \times 16.6 \text{ ns} = 215.8 \text{ ns}$
 - Continuous conversion 모드로 바로 다시 시작하기를 반복하면 $1/215.8 \text{ ns} = 4.63 \text{ Msps}$

Injected Channel

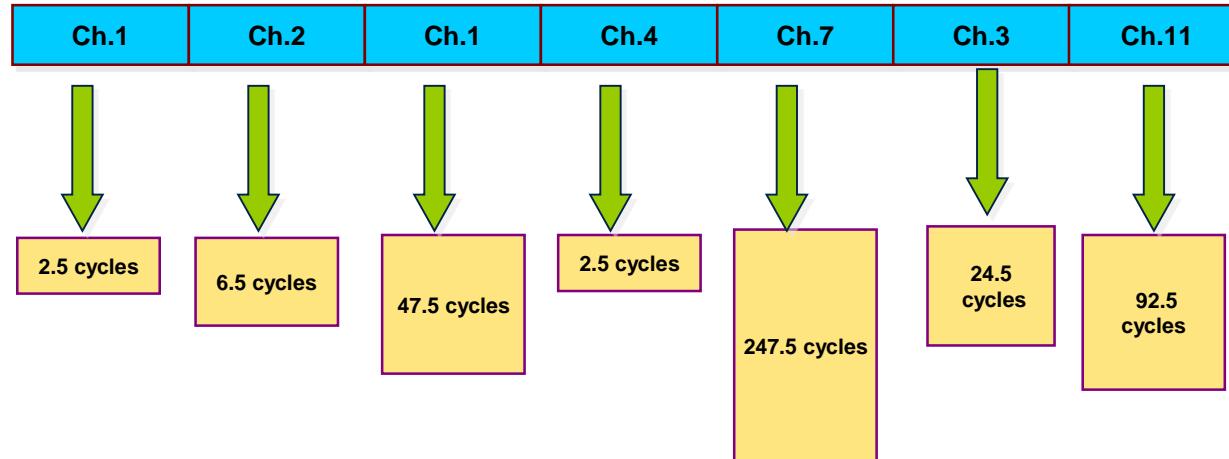
- Injected channel 은 일반 regular channel 이 진행 중일 때 중단하고 먼저 실행할 수 있는 우선권을 갖는다



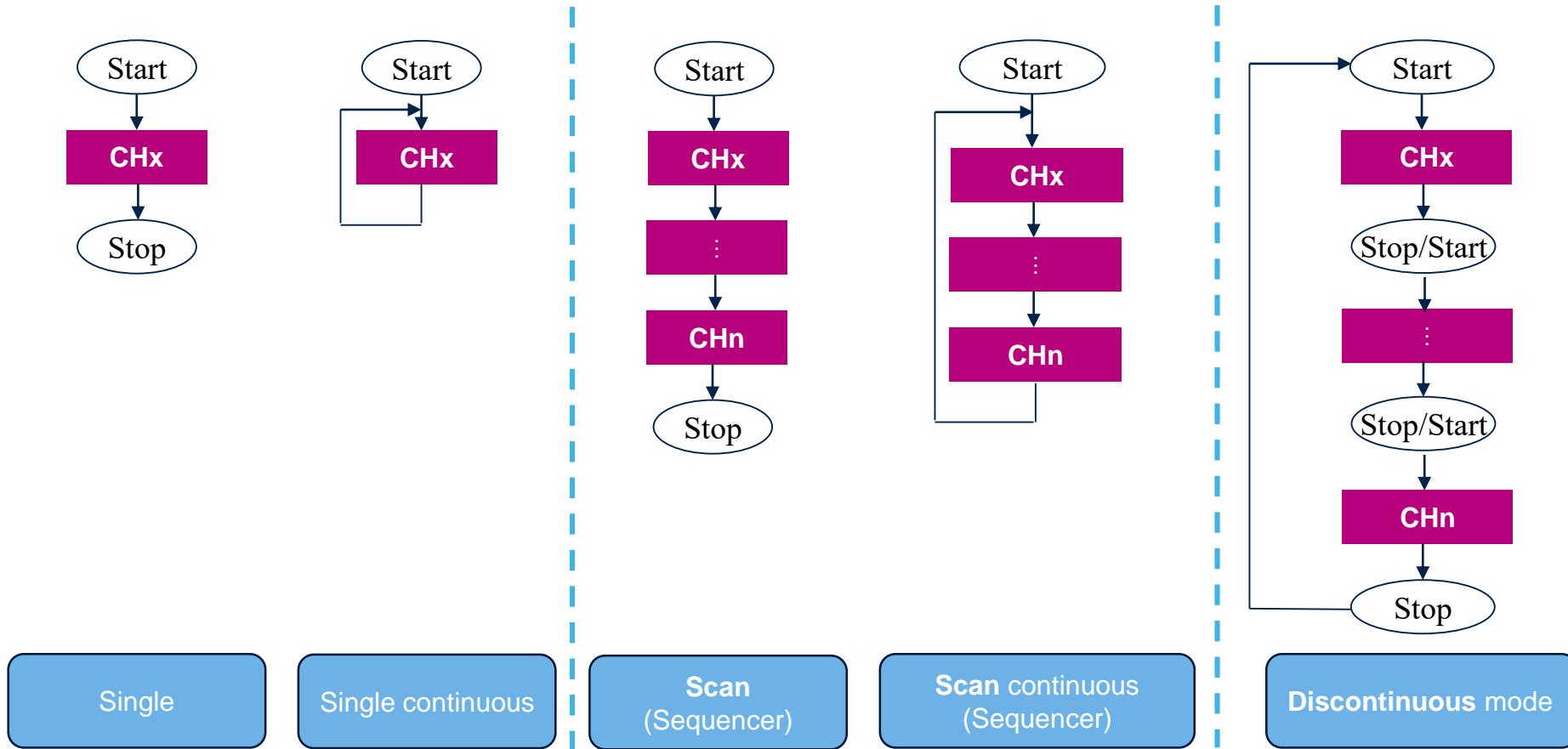
- Injected channel 핀과 regular channel 핀은 물리적으로 구분되지 않고 단지 해당 핀을 regular channel 로 쓸지 injected channel 로 쓸지에 따라 정해진다
- Trigger 발생시 regular conversion을 중단하고 injected conversion 시작되며 단일 trigger로 최대 4개의 injected conversion 이 가능
- Injected conversion 종료 시점에서 regular conversion auto-resume 발생
- 4개의 전용 16-bit data register는 injected conversion 결과를 위해 사용 가능
- Injected conversion 은 ADC 동작 중에도 채널 변경이 가능하다

Sequencer

- 최대 16 개 regular 채널 또는 4 개 injected 채널을 샘플링 순서와 시간을 지정해서 regular conversion 그룹 또는 injected conversion 그룹으로 묶고 하드웨어적으로 한번만 연속되게 (Scan) 또는 계속 연속되게 (Continuous) 또는 비 연속되게 여러번 나눠서 (Discontinuous) 자동으로 샘플링 되게 처리되게 할 수 있다
 - 예시, Regular 채널 그룹
 - 채널 1, 2, 1, 4, 7, 3, 11 을 각각 샘플링 시간을 다르게 해서 regular 그룹으로 설정

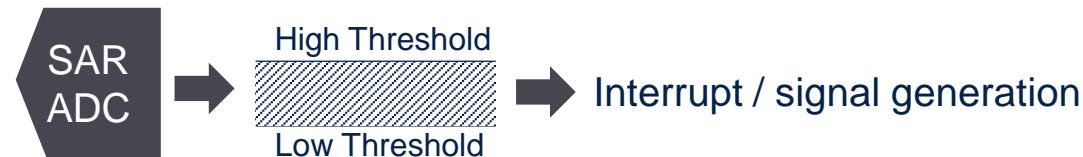


Conversion Modes



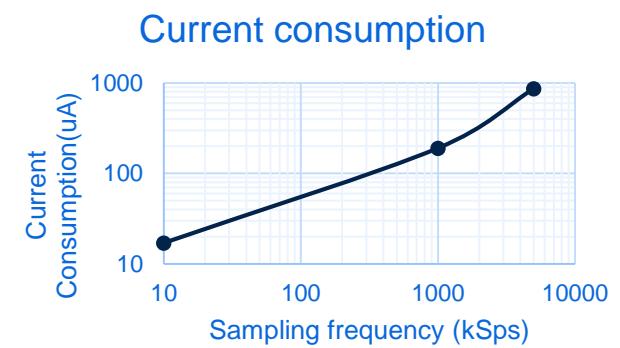
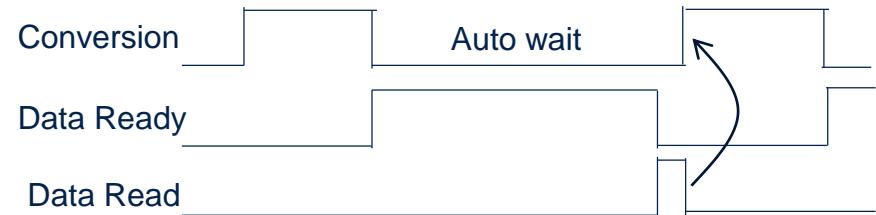
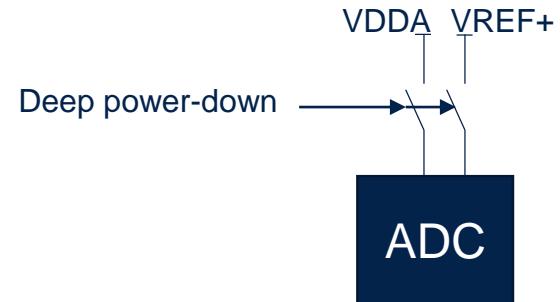
Analog Watchdog

- 각 ADC 블록은 window comparator 처럼 동작하는 3개의 Analog watchdog 을 지원한다
 - Analog watchdog 은 상시로 입력 채널의 전압을 모니터링하고 있다가 상위, 하위 threshold 를 벗어나는 입력이 발생하면 인터럽트를 발생하거나 내부 이벤트 신호 (ADC_AWDx_OUT) 를 출력해서 타이머에 입력 신호 (ETR) 를 보낼 수 있다
 - 하나의 12bit Analog watchdog 는 선택한 채널 하나 또는 모든 채널을 모니터링 할 수 있다
 - 두개의 8bit Analog watchdog 은 몇개의 선택한 채널만 모니터링 할 수 있다



Low_power features

- Deep power-down mode
 - 대기전력을 감소를 위해 ADC를 사용하지 않을 때 ADC peripheral로 공급되는 전원을 비활성화 할 수 있다
- Auto-delayed conversion
 - ADC는 마지막 data가 읽혀질 때까지 다음 번 conversion 시작을 멈추고 대기할 수 있으며 소모전류를 감소시키는 효과를 낸다
- Sampling time에 따른 전력 소비
 - Max 730 μ A @ 4 Msamples/s
 - Max 220 μ A @ 1 Msamples/s
 - Max 50 μ A @ 10 ksamples/s



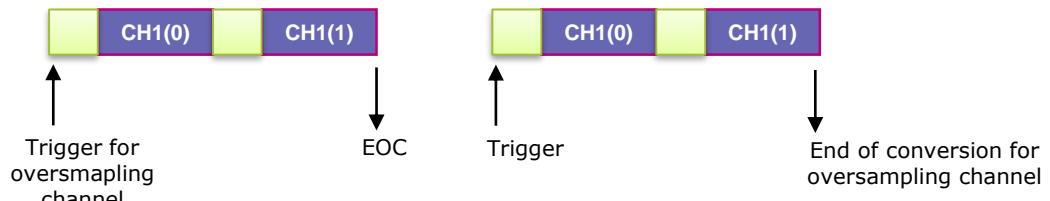
Oversampler

- Oversampler는 소프트웨어의 부하를 덜기 위해 하드웨어적으로 최대 256개의 ADC 샘플링 값을 평균(No moving avg)낸 값을 제공한다



- Right alignment만 사용 가능
- Programmable Oversampling ratio
 - x2, x4, x8, x16, x32, x64, x128, x256
- Programmable Data shifter, truncator
 - Right shift 0 - 8 bit

Example: Oversampling ratio x2



Sampling
Conversion

STM32G4 ADC Registers

- ADCx_CR
 - ADC Control Register
 - ADC 을 enable/disable, Regular/Injected conversion software start/stop, Deep power down enable/disable, ADC calibration 시작 등을 설정한다
- ADCx_ISR
 - ADC Interrupt and Status Register
 - ADRDY : ADC 를 enable (ADEN=1) 한 이후 conversion 시작 준비가 되었음을 나타낸다
 - EOSMP : Regular 채널의 sampling 이 끝나고 conversion 이 시작함을 나타낸다
 - EOC : Regular 채널의 sampling 과 conversion 이 모두 끝났음을 나타낸다
 - EOS : Regular 채널 sequencer 의 sampling 과 conversion 이 모두 끝났음을 나타낸다
 - OVR : EOC 가 여전히 세팅 되어 있는 동안 conversion 이 완료되면 overrun 이 발생한다
 - JEOC : Injected 채널의 EOC
 - JEOS : Injected 채널의 EOS
 - AWD1/2/3 : Analog watchdog 이 LT 와 HT 의 threshold 를 벗어났음을 나타낸다

STM32G4 ADC Registers

- ADCx_IER
 - ADC Interrupt Enable Register
 - ADCx_ISR에서 제공하는 상태를 인터럽트로 발생할지 여부를 enable//disable 한다
- ADCx_CFGR1/2
 - ADC Configuration Register
 - Analog watchdog enable/disable, Single, Continuous, Discontinuous 설정, Overrun 발생시 overwrite 여부 설정, ADC resolution 설정, ADC left/right align 설정, DMA 설정, External trigger 설정, Oversampling 설정을 한다
- ADCx_SMPR1/2
 - ADC Sample Time Register
 - 채널 별 sampling time 을 설정한다

STM32G4 ADC Registers

- ADCx_SQR1/2/3/4
 - ADC Regular Sequencer Register
 - 1 – 16 번 까지 Regular Sequencer 를 위한 채널 번호를 설정한다
- ADCx_JSQR
 - ADC Injected Sequencer Register
 - 1 – 4 번 까지 Injected Sequencer 를 위한 채널 번호를 설정한다
- ADCx_DR
 - ADC Regular Data Register
 - Conversion 이 완료된 ADC 결과 값을 담고 있다

STM32G4 ADC Registers

- ADC_CSR
 - ADC Common Status Register
 - ADC 를 Master / Slave 로 동작의 상태를 나타낸다
- ADC_CCR
 - ADC Common Control Register
 - ADC 블록 전체에 적용되는 설정을 한다
 - ADC clock source, prescaler, Dual mode, Vrefint, VBAT 설정 등

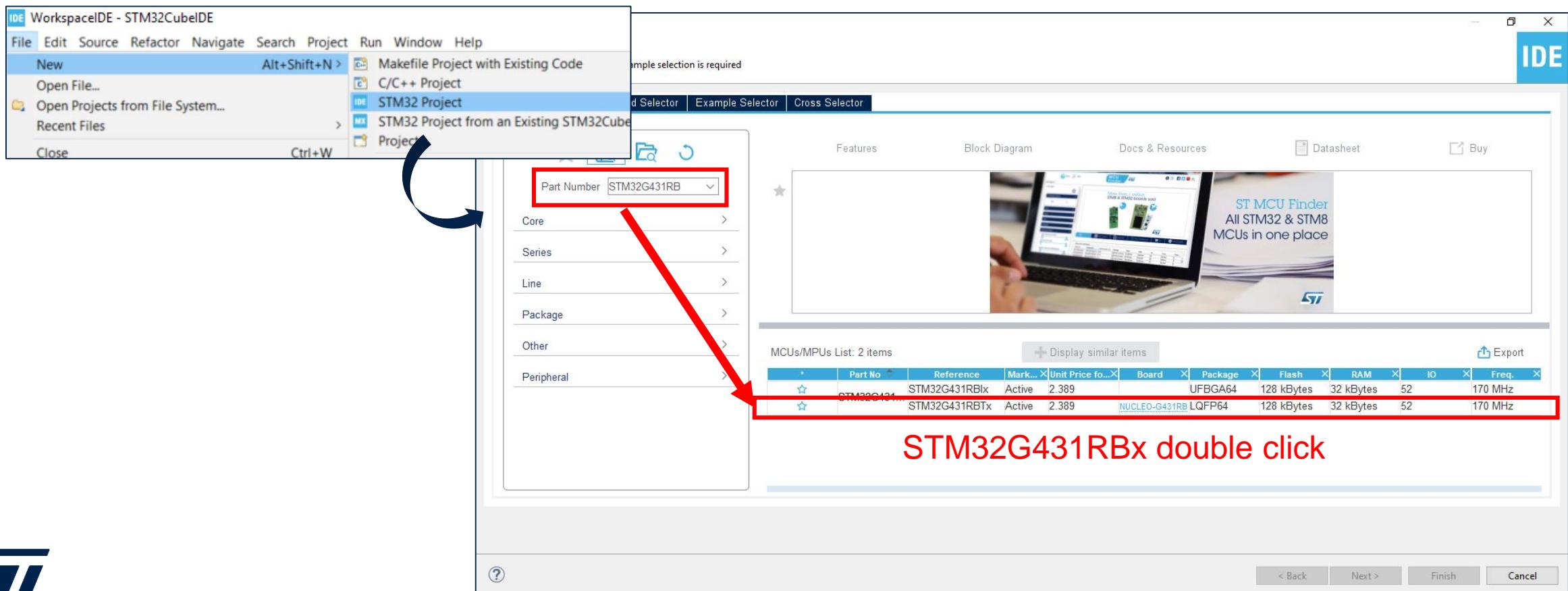
Hands-On Session: ADC



- STM32G431RBx 의 DAC를 통해 삼각 파형을 출력하고, 이 파형을 ADC 입력으로 받아 버퍼에 저장, SWV 그래프로 결과 확인
- STM32CubeIDE를 사용하여 DAC, TIM6, ADC 설정 후 Initialization code를 생성

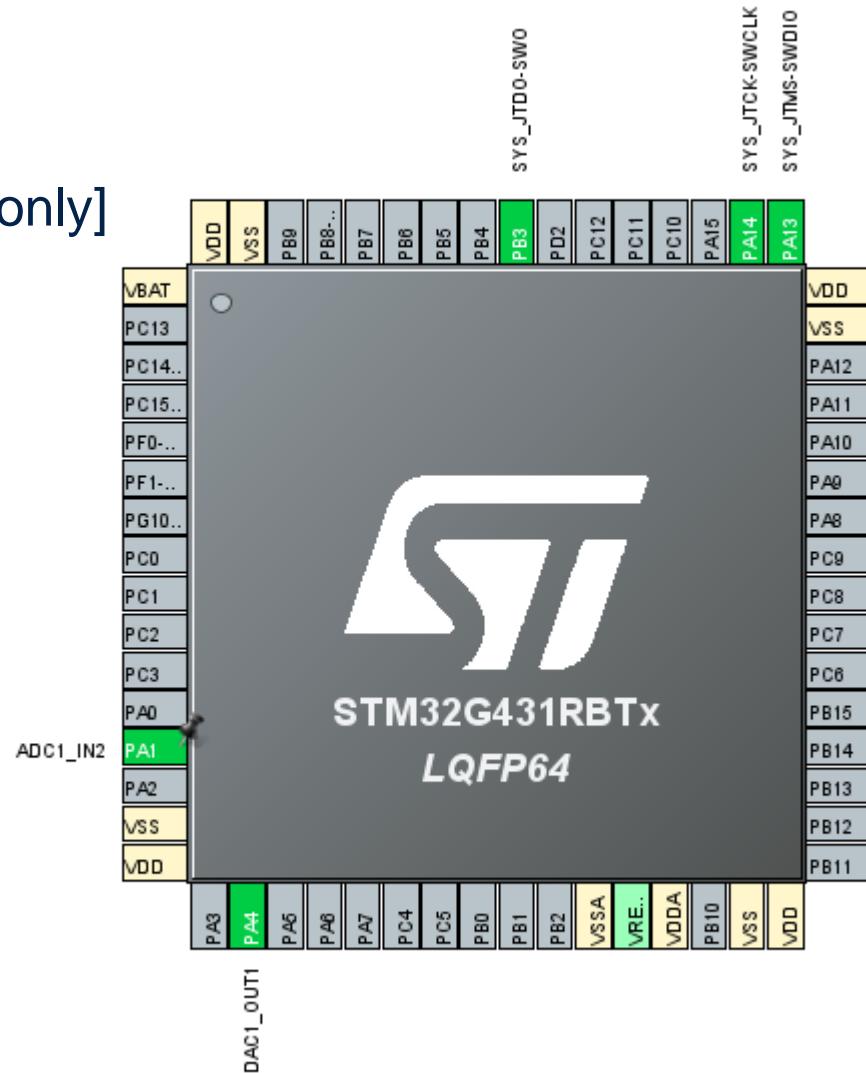
Create STM32CubeIDE Project

- File > New > STM32 Project
 - Part Number Search: **STM32G431RB**



Peripheral and pin-out

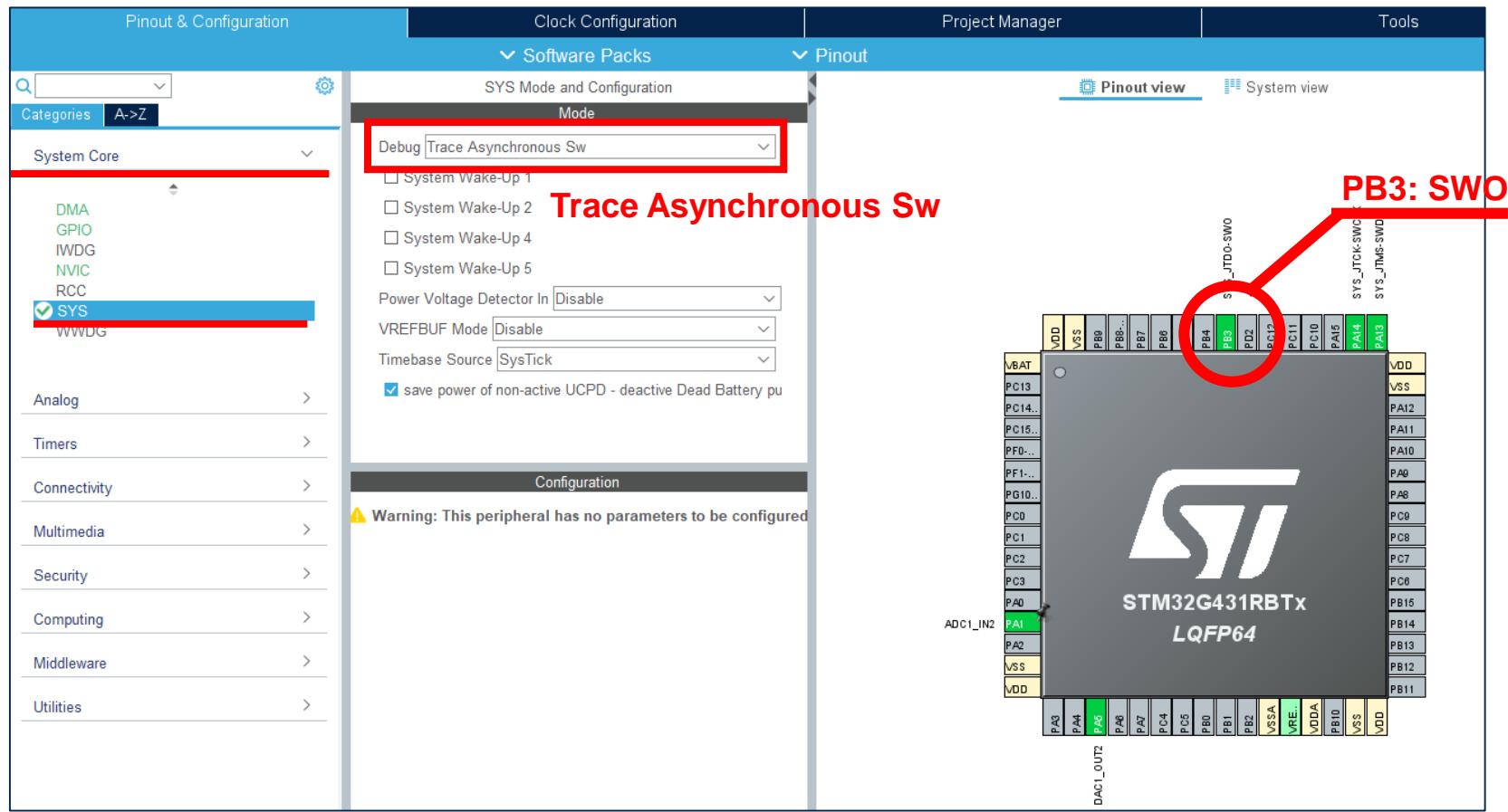
- System Core → SYS → Debug → Trace Asynchronous Sw
 - **PB3** : SWO
- Analog → DAC1 → Out1 mode [Connected to external pin only]
 - **PA4** : DAC1_OUT1
- Analog → ADC1 → IN2 [IN2 Single-ended]
 - **PA1** : ADC1_IN2
- Timers → TIM6 -> Activated



Configuration

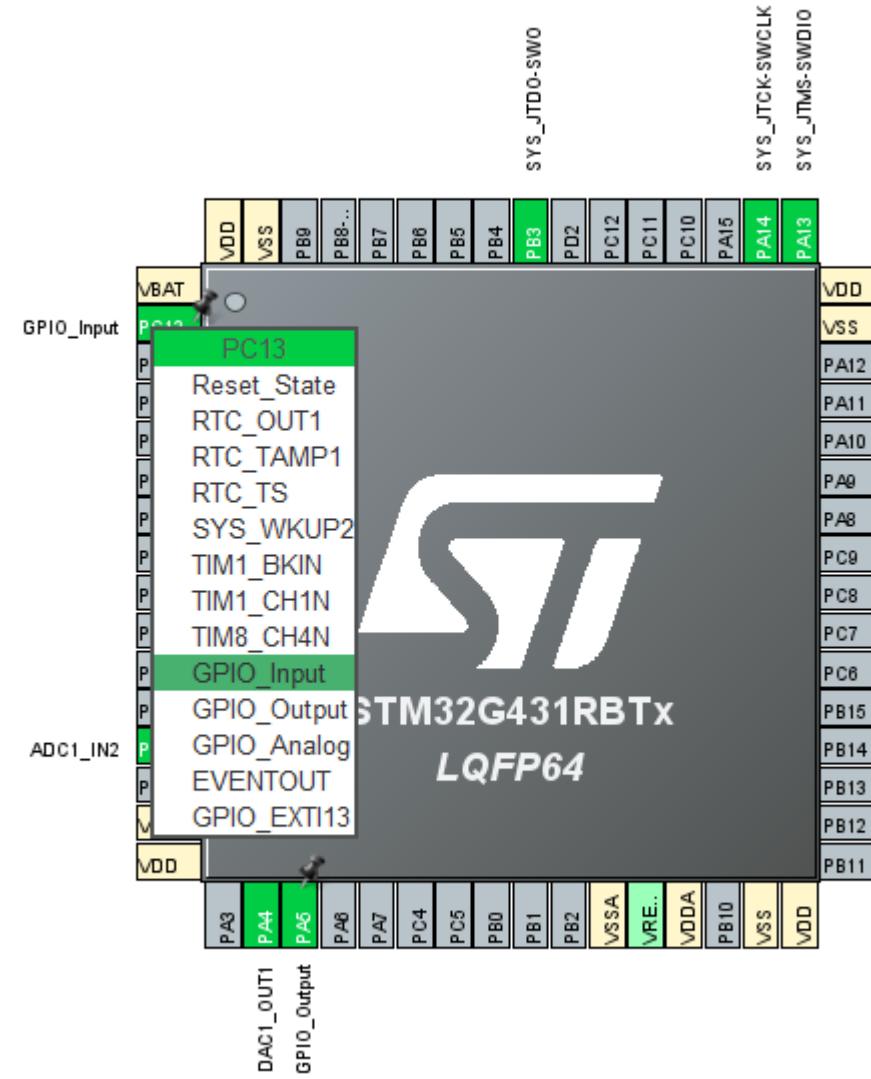
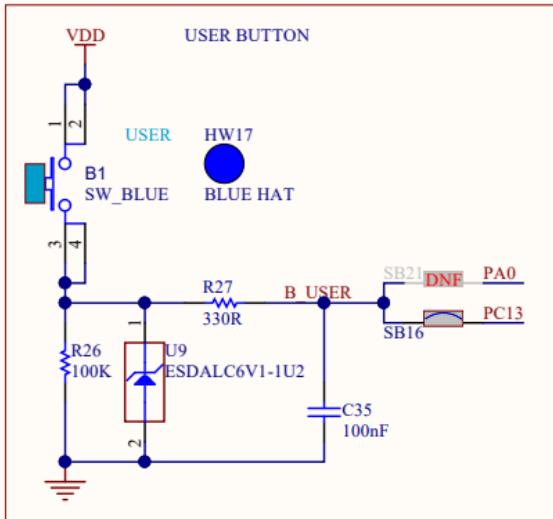
- **Pinout & Configuration**

- **System Core > SYS > Debug: Trace Asynchronous Sw**



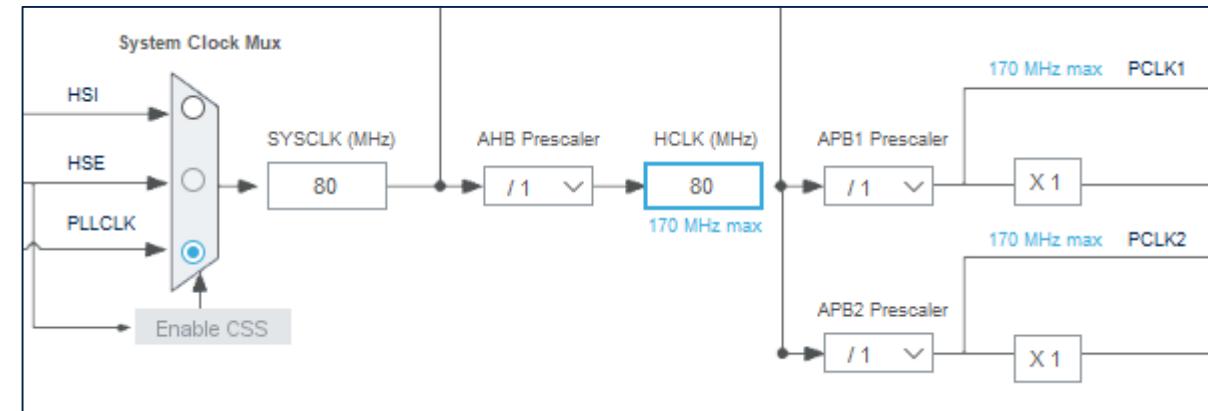
Peripheral and pin-out

- Button 설정
 - User Button PC13을 GPIO_INPUT으로 설정



Clock configuration

- STM32G431RBx 클럭 설정
 - System clock : 80 MHz
 - PLL clock source : HSI 16MHz
 - APB1 Prescaler : /1
 - APB2 Prescaler : /1
 - Cortex system timer Prescaler : / 1



DAC1 configuration

- DAC Out1 Parameter Settings

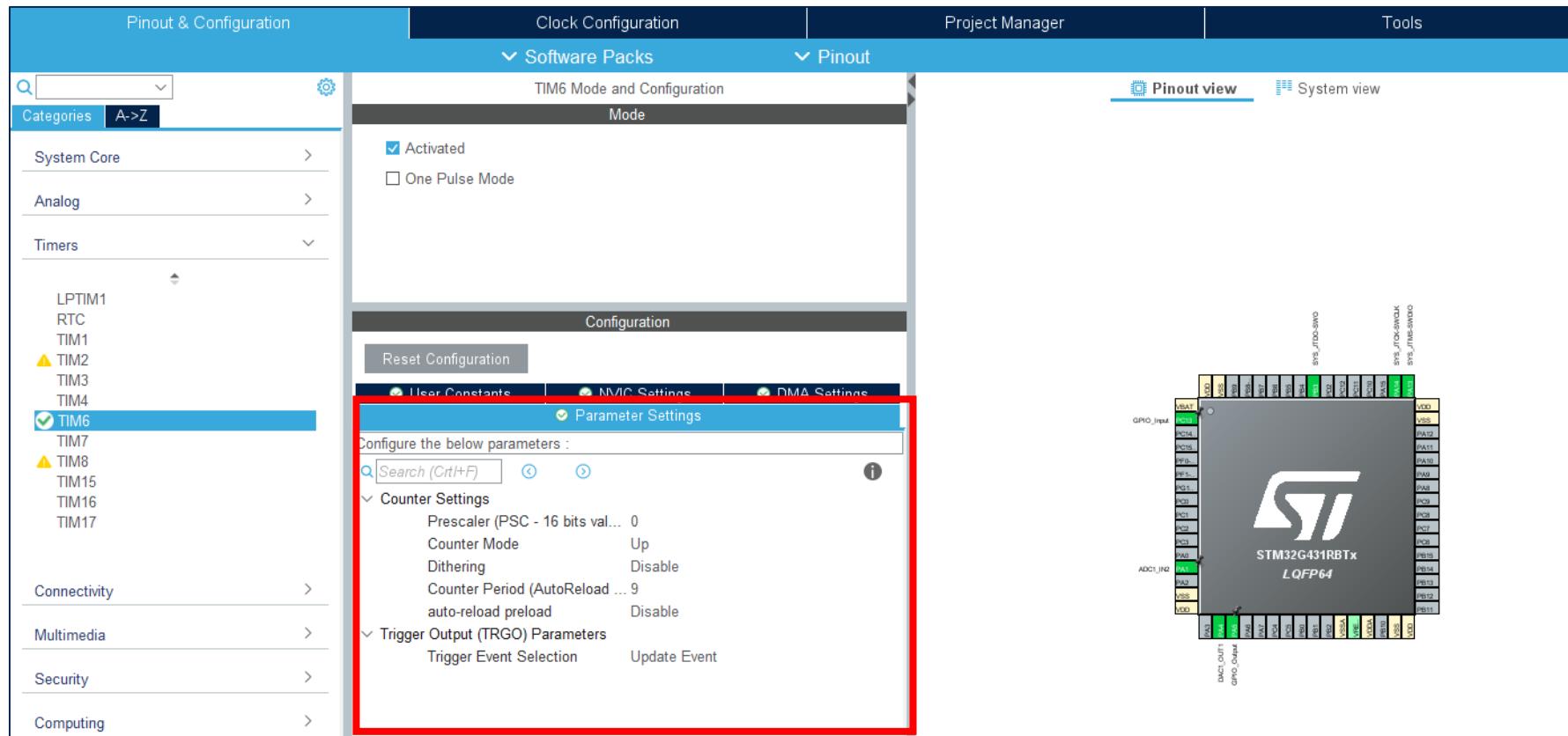
- Trigger : Timer 6 Out event (삼각파형 주기를 TIM6로 제어)
- Wave generation mode : Triangle wave generation
- Maximum Triangle Amplitude : 1023

The screenshot shows the STM32CubeMX software interface for pinout and configuration. The left sidebar lists various peripheral components under categories like System Core, Analog, and Timers. The 'DAC1' component is selected and highlighted with a blue background. The main central area displays the 'Software Packs' and 'Pinout' tabs, with the 'Pinout view' tab currently active. On the right side, a detailed pinout diagram for the STM32G431RBTx LQFP64 package is shown, mapping specific pins to their functions. A red box highlights the 'Parameter Settings' section of the configuration window, which contains the following parameters:

Parameter	Setting
Output Buffer	Enable
DAC High Frequency	Mode Automatic
DMA Double Data	Disable
Signed Format	Disable
Trigger	Timer 6 Trigger Out event
Trigger2	None
Wave generation mode	Triangle wave generation
Maximum Triangle Amplitude	1023
User Trimming	Factory trimming
Sample And Hold	Sampleandhold Disable

TIM6 configuration

- Timer 6 Parameter Settings
 - Counter period : 9 (이 값에 따라 삼각파의 주기가 결정)
 - Trigger Output parameter : Update Event



ADC1 configuration

- ADC1 Parameter Settings
 - Clock Prescaler : Synchronous / 4
 - Scan Conversion Mode : Disabled
 - Continuous Conversion Mode : Enabled

The screenshot shows the STM32CubeMX software interface with the following details:

- Pinout & Configuration** tab is selected.
- Clock Configuration** tab is visible.
- Project Manager** tab is visible.
- Tools** tab is visible.
- Categories A-Z** dropdown shows **ADC1** is selected.
- ADC1 Mode and Configuration** section:
 - Mode**:
 - IN1: Disable
 - IN2: IN2 Single-ended
 - IN3: Disable
 - IN4: Disable
 - IN5: Disable
 - Configuration**:
 - Reset Configuration
 - Parameter Settings (highlighted with a red box)
 - User Constants
 - ADCs_Common_Settings**:
 - Mode: Independent mode
 - ADC_Settings**:
 - Clock Prescaler**: Synchronous clock mode divided by 4
 - Resolution: ADC 12-bit resolution
 - Data Alignment: Right alignment
 - Gain Compensation: 0
 - Scan Conversion Mode: Disabled
 - End Of Conversion Selection: End of single conversion
 - Low Power Auto Wait: Disabled
 - Continuous Conversion Mode: Enabled
 - Discontinuous Conversion Mode: Disabled
 - DMA Continuous Requests: Disabled
 - Overrun behaviour: Overrun data preserved
- Pinout view** and **System view** tabs are present.
- STM32G431RBTx LQFP64** pinout diagram is shown on the right.

ADC1 configuration

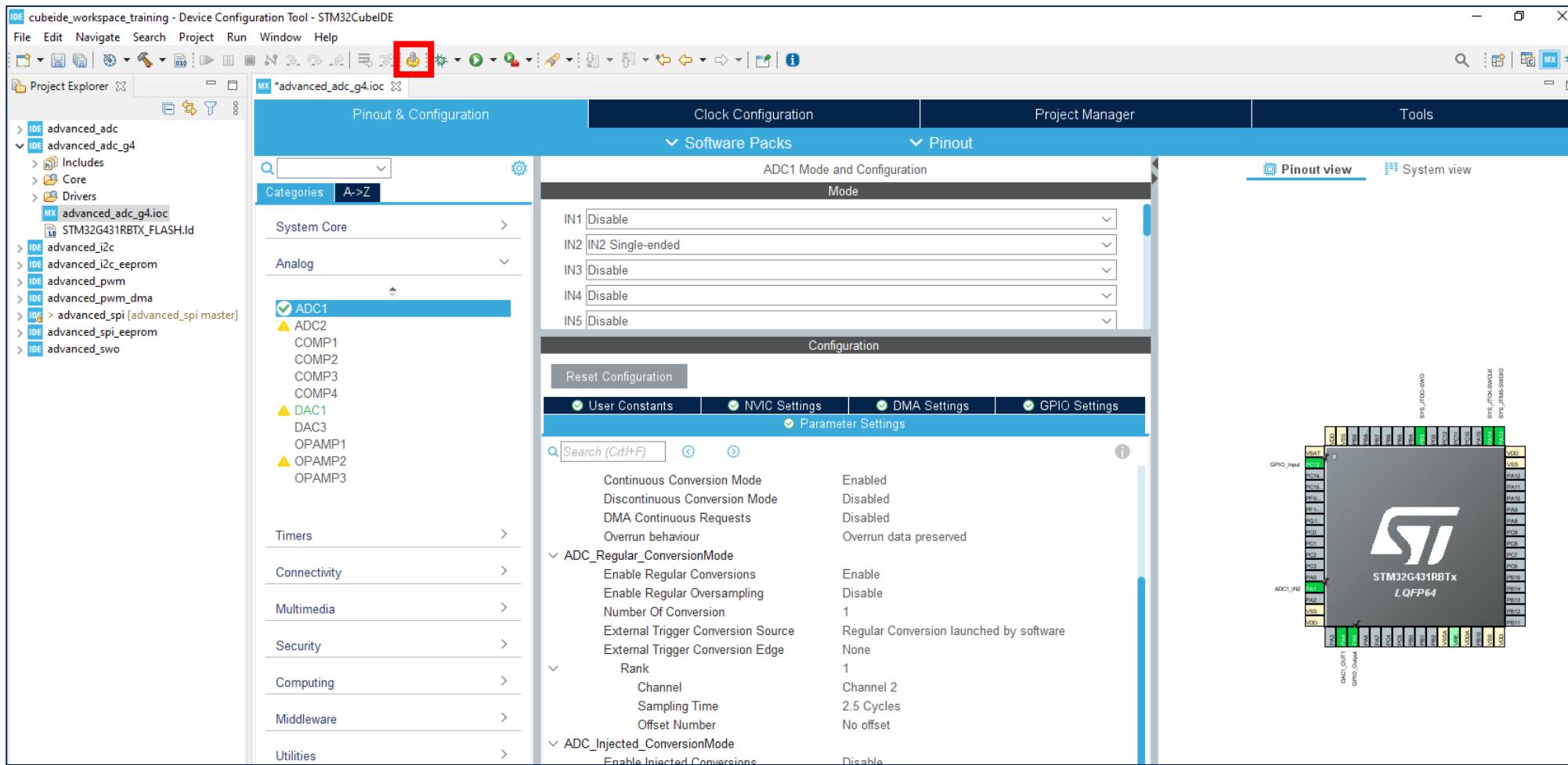
- ADC1 DMA Settings
 - Channel : DMA1 Channel 1
 - Direction : Peripheral to Memory
 - Mode: Normal
 - Data Width : Half Word

The screenshot shows the STM32CubeMX software interface with the following details:

- Pinout & Configuration** tab is selected.
- Software Packs** dropdown is open, showing **ADC1 Mode and Configuration**.
- Configuration** section is highlighted with a red box:
 - DMA Request**: Set to **ADC1**, **DMA1 Channel 1**, **Peripheral To Memory**, **Low Priority**.
 - DMA Request Settings**:
 - Mode**: **Normal**
 - Increment Address**:
 - Peripheral** and **Memory** sections both have checked.
 - Data Width**: **Half Word**
- Pinout view** and **System view** tabs are visible on the right.
- STM32G431RBTx LQFP64** pinout diagram is shown on the right side.

Code Generation

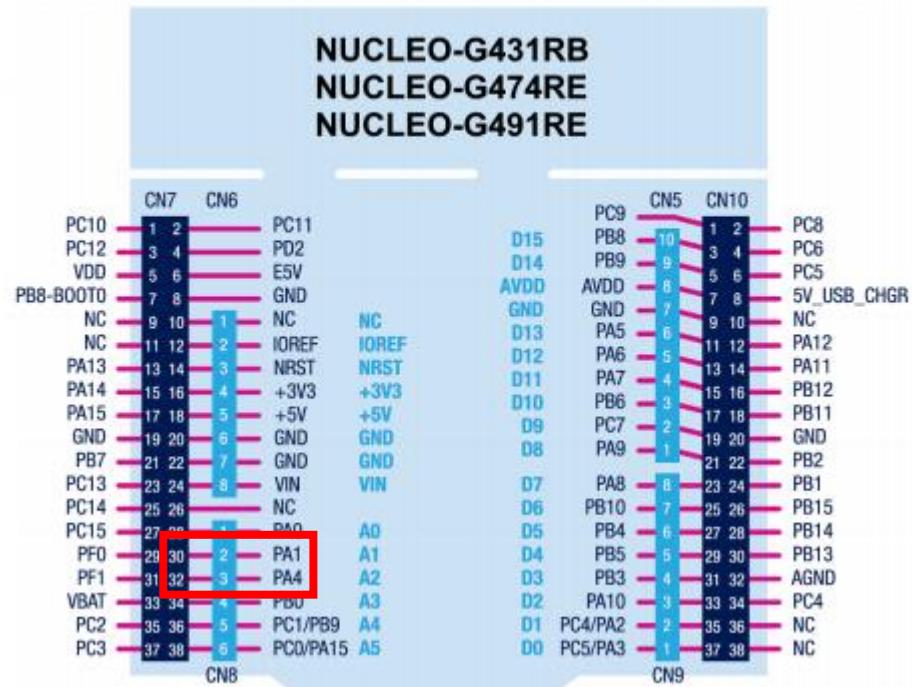
- Project > Code Generate (Alt + K)



SWO 핀 연결

- DAC 출력을 ADC 입력으로 연결
 - 꽂혀 있는 X-NUCLEO-EEPROMA2 뺀다.
 - PA1 와 PA4 를 점퍼케이블로 (또는 그라운드 핀에 있는 점퍼 캡으로) 연결해 준다.

Figure 18. ARDUINO® and ST morpho connectors pinout



Code description

- **DAC_Ch1_Start()**

```
/* USER CODE BEGIN 0 */
static void DAC_Ch1_Start(void)
{
    /* Enable DAC channel 2 */
    if(HAL_DAC_Start(&hdac1, DAC_CHANNEL_1) != HAL_OK){
        Error_Handler();
    }

    /* Set DAC channel 2 DHR12RD register */
    if(HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R, 0x100) != HAL_OK){
        Error_Handler();
    }

    /* Enable TIM peripheral */
    HAL_TIM_Base_Start(&htim6);
}
/* USER CODE END 0 */
```

Code description

- **HAL_ADC_ConvCpltCallback()**

```
/* USER CODE BEGIN 4 */  
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)  
{  
    ADC_ConvCpltFlag = 1;  
}  
/* USER CODE END 4 */
```

Code description

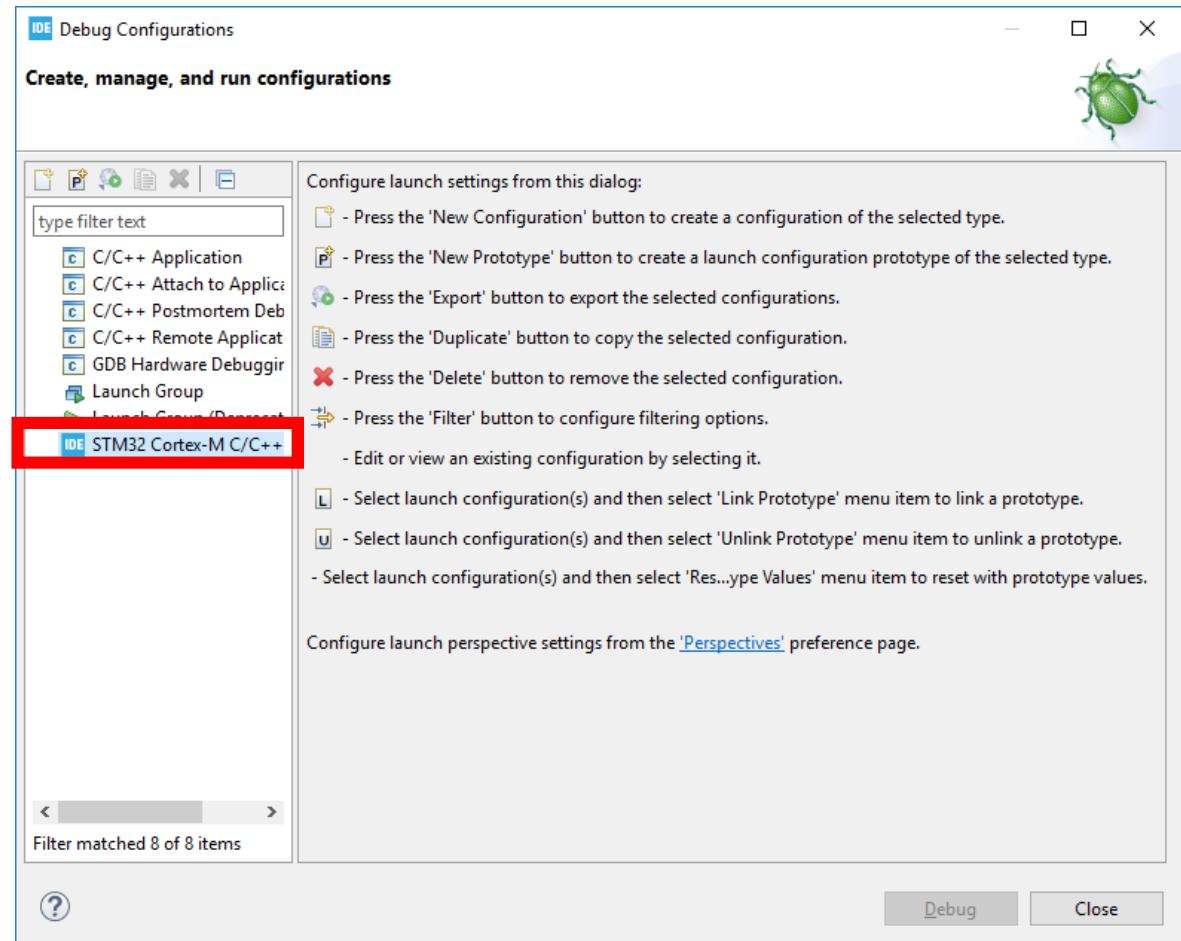
- main()

```
/* USER CODE BEGIN PV */  
volatile uint32_t ADC_ConvCpltFlag;  
volatile uint16_t ADC_monitor;  
uint16_t ADC_buff[4096];  
/* USER CODE END PV */  
  
/* USER CODE BEGIN 1 */  
uint16_t tmp;  
/* USER CODE END 1 */
```

```
/* USER CODE BEGIN 2 */  
DAC_Ch1_Start();  
/* USER CODE END 2 */  
  
/* Infinite loop */  
/* USER CODE BEGIN WHILE */  
while (1)  
{  
    while(HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_13) == GPIO_PIN_RESET){  
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);  
        HAL_Delay(100);  
    }  
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);  
  
    /* Start ADC conversion from PA1 */  
    HAL_ADC_Start_DMA(&hadc1, (uint32_t*)ADC_buff, 4096);  
    while(ADC_ConvCpltFlag == 0);  
    ADC_ConvCpltFlag = 0;  
  
    /* Display ADC_monitor value using SWV data trace */  
    for(tmp=0;tmp<4096;tmp++){  
        ADC_monitor = *((uint16_t*)ADC_buff+tmp);  
        HAL_Delay(1);  
    }  
    /* USER CODE END WHILE */
```

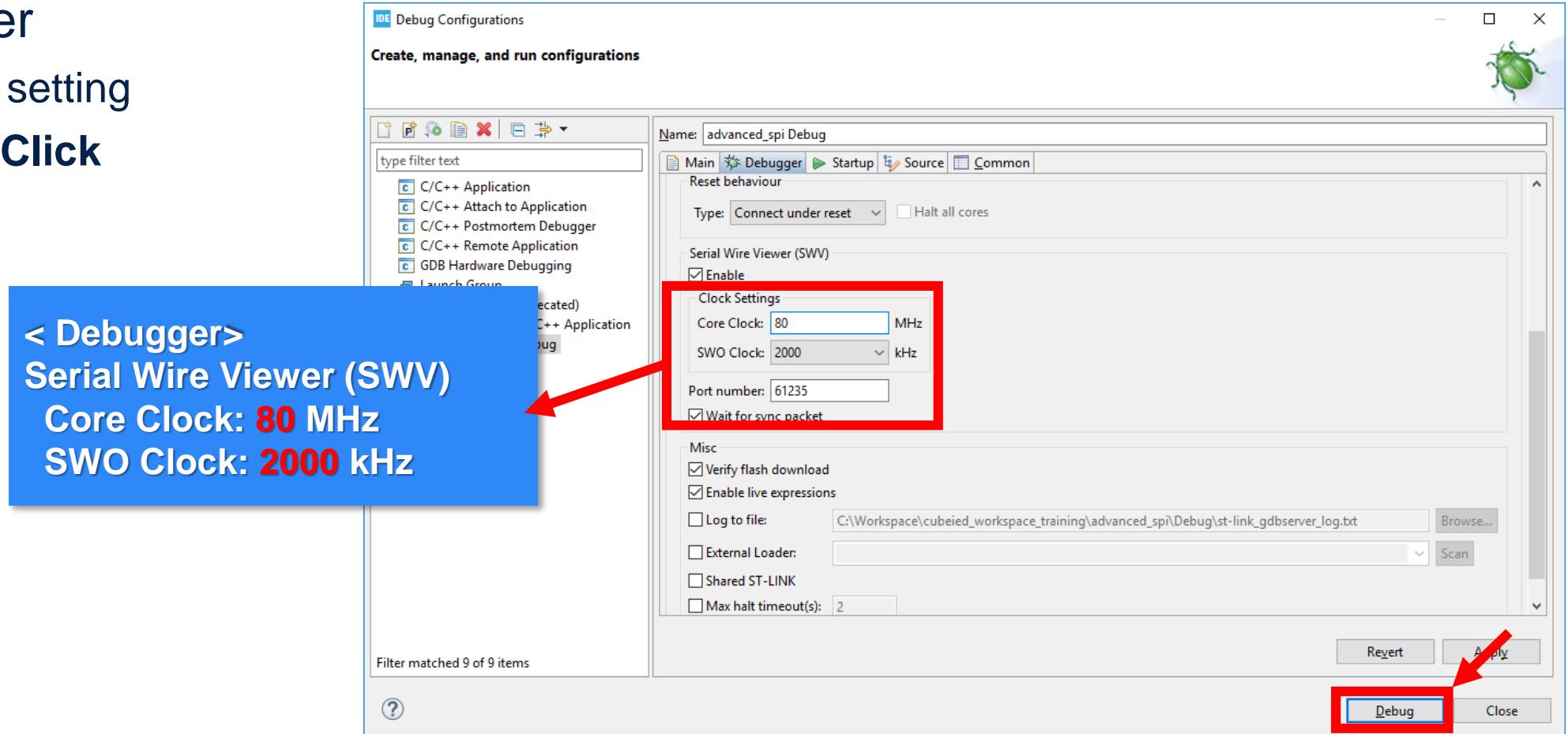
SWV Data Trace Setting

- Debug Configurations
 - Run > Debug Configurations...
 - **STM32 Cortex-M C/C++ Double Click**



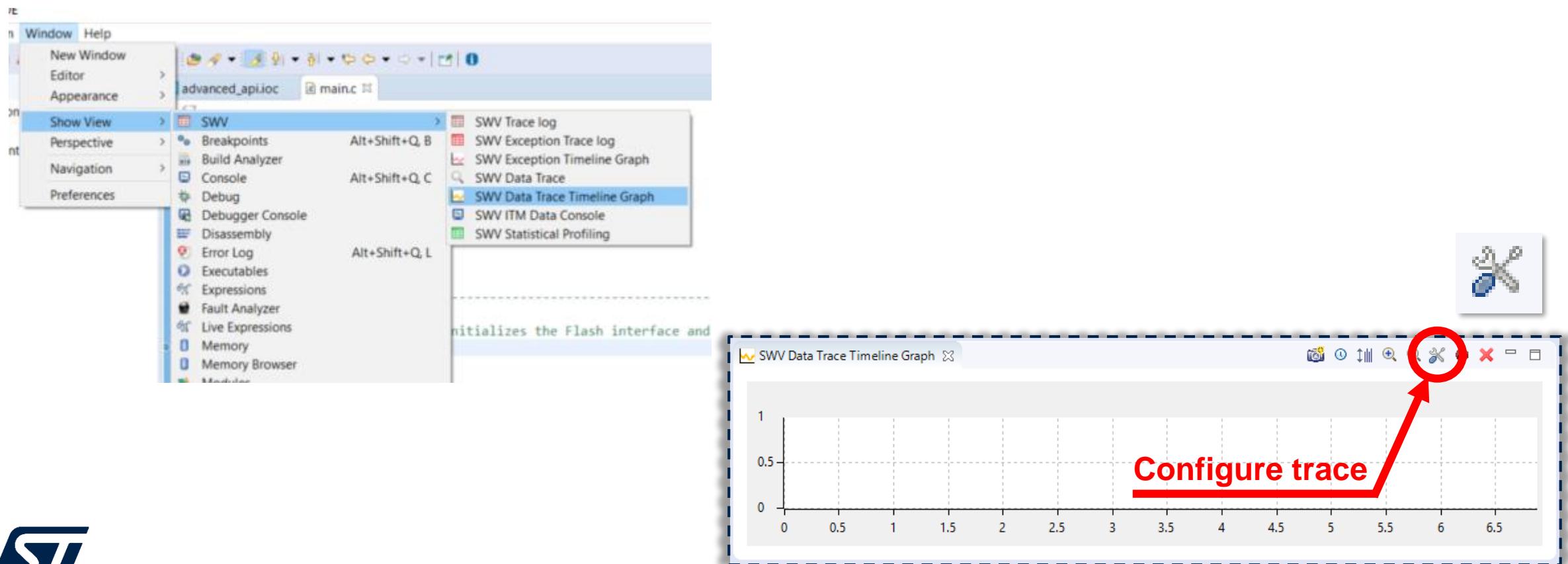
SWV Data Trace Setting

- Debugger
 - 아래 값 setting
 - Debug Click



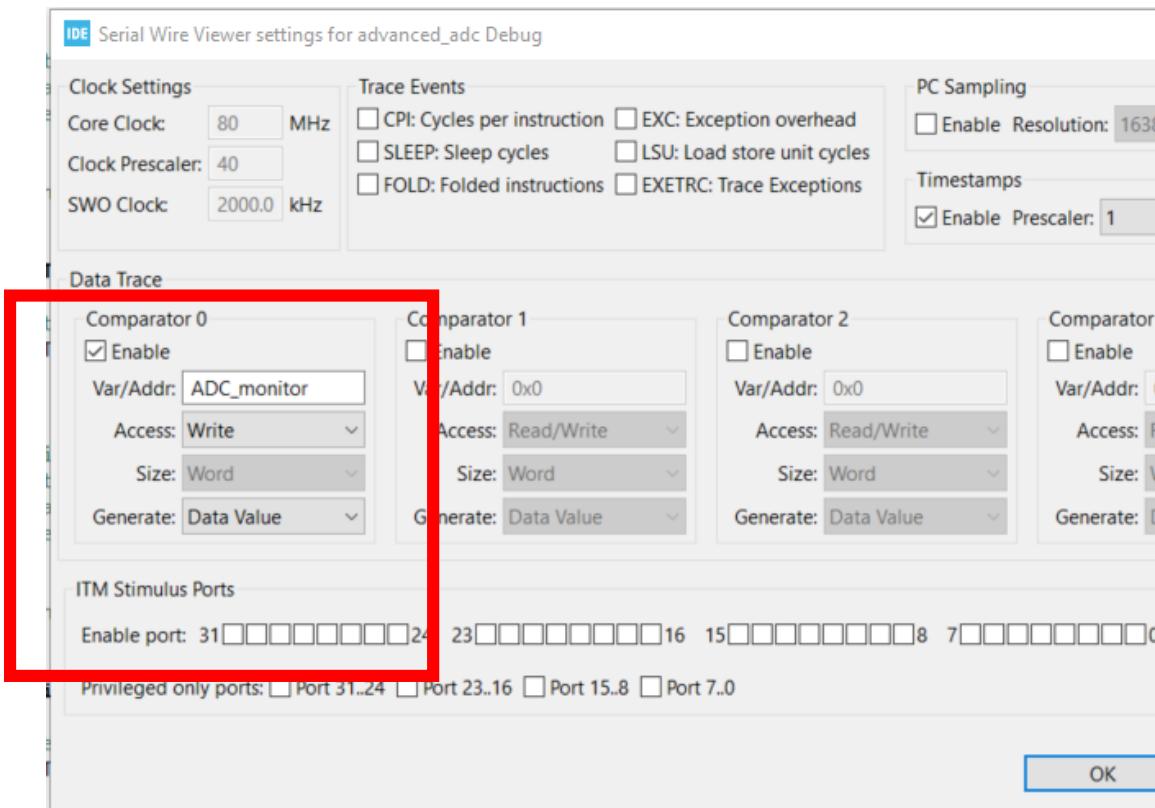
SWV Data Trace Setting

- SWV Data Trace Timeline Graph 뷰어 열기 (Debug 모드에서 수행)
 - Window > Show View > Other... > SWV > **SWV Data Trace Timeline Graph Open**



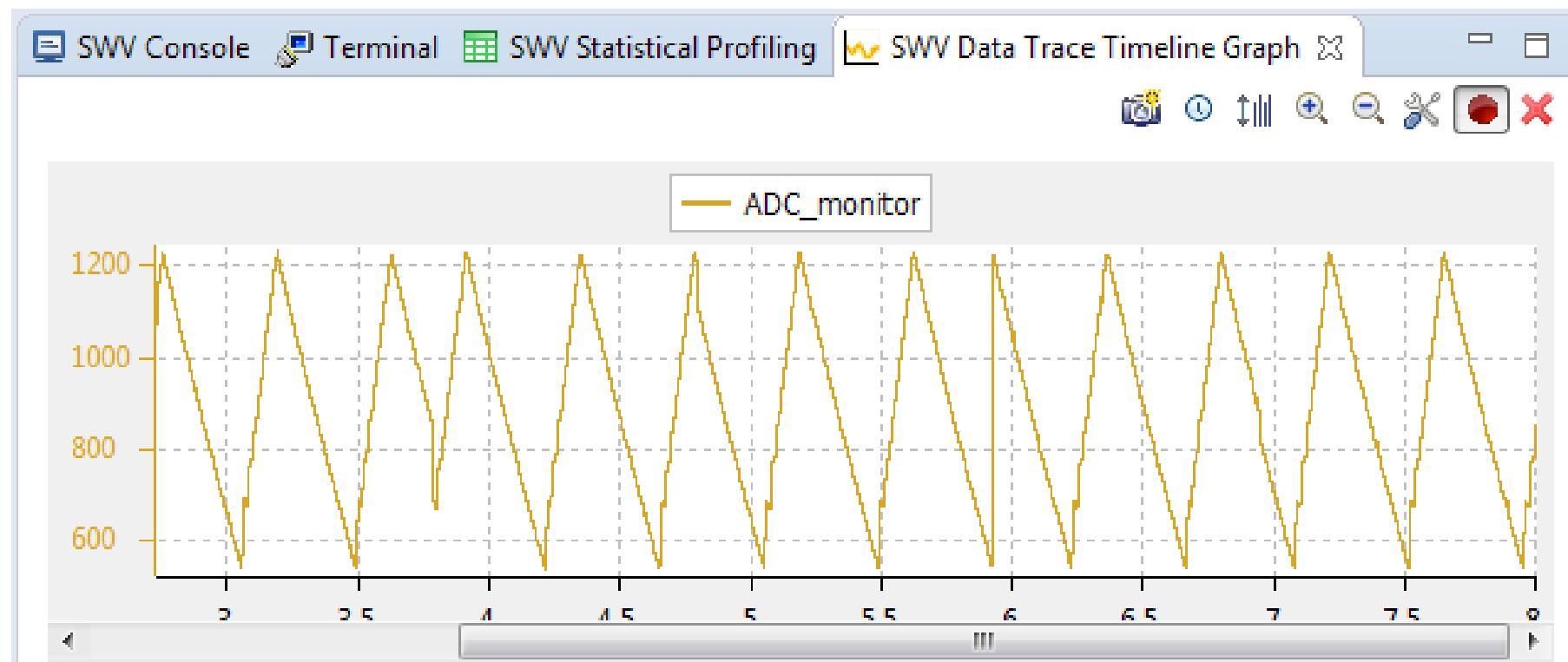
SWV Data Trace Setting

- SWV settings



ADC 측정 결과

- SWV Data Trace Timeline Graph를 통해 버퍼 값 확인



Thank you

© STMicroelectronics - All rights reserved.

ST logo is a trademark or a registered trademark of STMicroelectronics International NV or its affiliates in the EU and/or other countries.

For additional information about ST trademarks, please refer to www.st.com/trademarks.

All other product or service names are the property of their respective owners.

