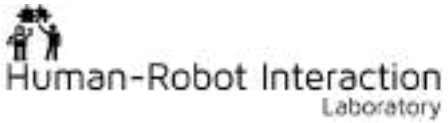
임베디드시스템설계
EMBEDDED SYSTEM DESIGN

CHAPTER 08

타이머를 이용한 입출력 제어



8.1 범용 타이머의 구조 및 기능

STM32F 타이머의 개요

- **범용(General-purpose) 타이머**

  출력비교, 원펄스, 입력캡처, 센서 인터페이스(엔코더, 홀 센서)

- **고급(Advanced) 타이머**

  모터제어, 디지털 파워변환, 비상 셧다운 입력

- **기본(Basic) 타이머**

  시간기반 타이머, DAC의 트리거

- **채널(Channel) 타이머**

  범용타이머와 동일하나 채널을 1~2개만 가짐

8.1 범용 타이머의 구조 및 기능

범용 타이머의 개요

종 류	STM32F103C8	STM32F429ZI
범용 타이머 (General-purpose timer)	3개(TIM2 ~ TIM4)	10개 (TIM2~TIM5, TIM9~TIM14)
고급제어 타이머 (Advanced-control timer)	1개(TIM1)	2개(TIM1, TIM8)
기본 타이머 (Basic timer)	없음	2개(TIM6 ~ TIM7)

- **1개의 16비트 카운터를 가짐**

  업, 다운, 업/다운 모드 오토-리로드(Auto-reload) 기능

- **16비트의 프리스케일러(Presclaer)를 이용하여 카운터의 동작 클럭을 1 ~ 1/65,563까지 다운시킬 수 있음**

***프리스케일러(Presclaer)**
타이머에 공급하는 입력클럭 속도를 조절하는 분주기

8.1 범용 타이머의 구조 및 기능

범용 타이머의 개요

- **4개의 16비트 캡처/비교(Captuer/Compare) 채널을 가짐**

  카운터 모드　입력 캡처 모드　출력 비교 모드　PWM출력 모드　PWM입력 모드　원 펄스(One-pulse) 모드
  강제출력 모드　엔코더 인터페이스 모드　타이머 동기화(Timer synchronization)

- **엔코더 및 홀 센서 인터페이스 가능**

  모터제어, 디지털 파워 변환, 비상 셧다운 입력

- **6개의 독립적인 IRQ/DMA 요청 신호를 발생**

  업데이트 이벤트 발생시　캡처/비교 이벤트 발생시　입력트리거 발생시

*IRQ(Interrupt ReQuest)
*DMA(Direct Memory Access)

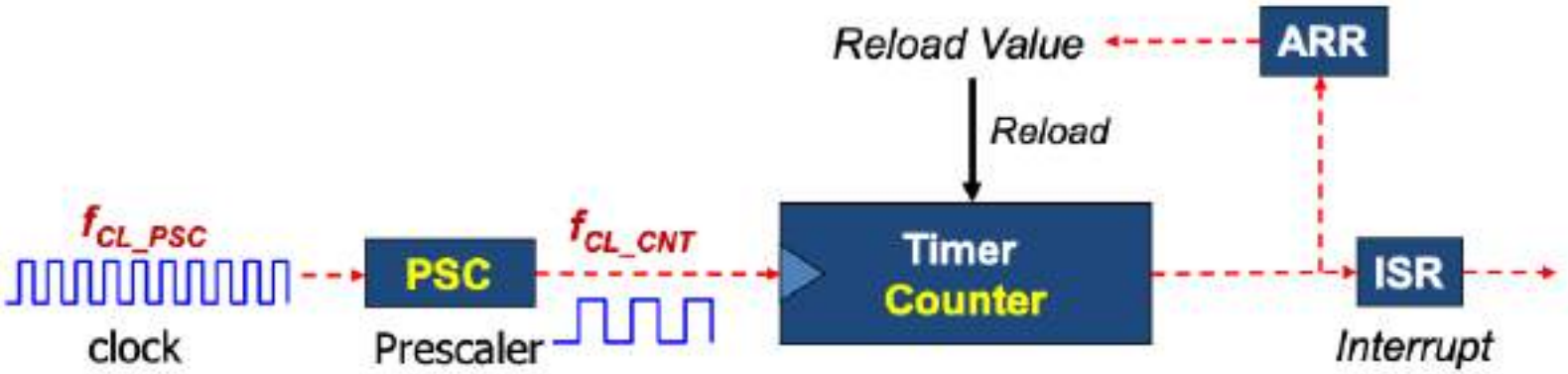
참고자료. 로체스터 공과대학 강의자료

Timer

- Free-run counter (independent of processor)
- Functions
  - **Input capture**
  - Output compare
  - Pulse-width modulation (PWM) generation
  - One-pulse mode output
- STM has many application notes (on all aspects of the STM32)
  - App note AN4776 – General Purpose Timer Cookbook

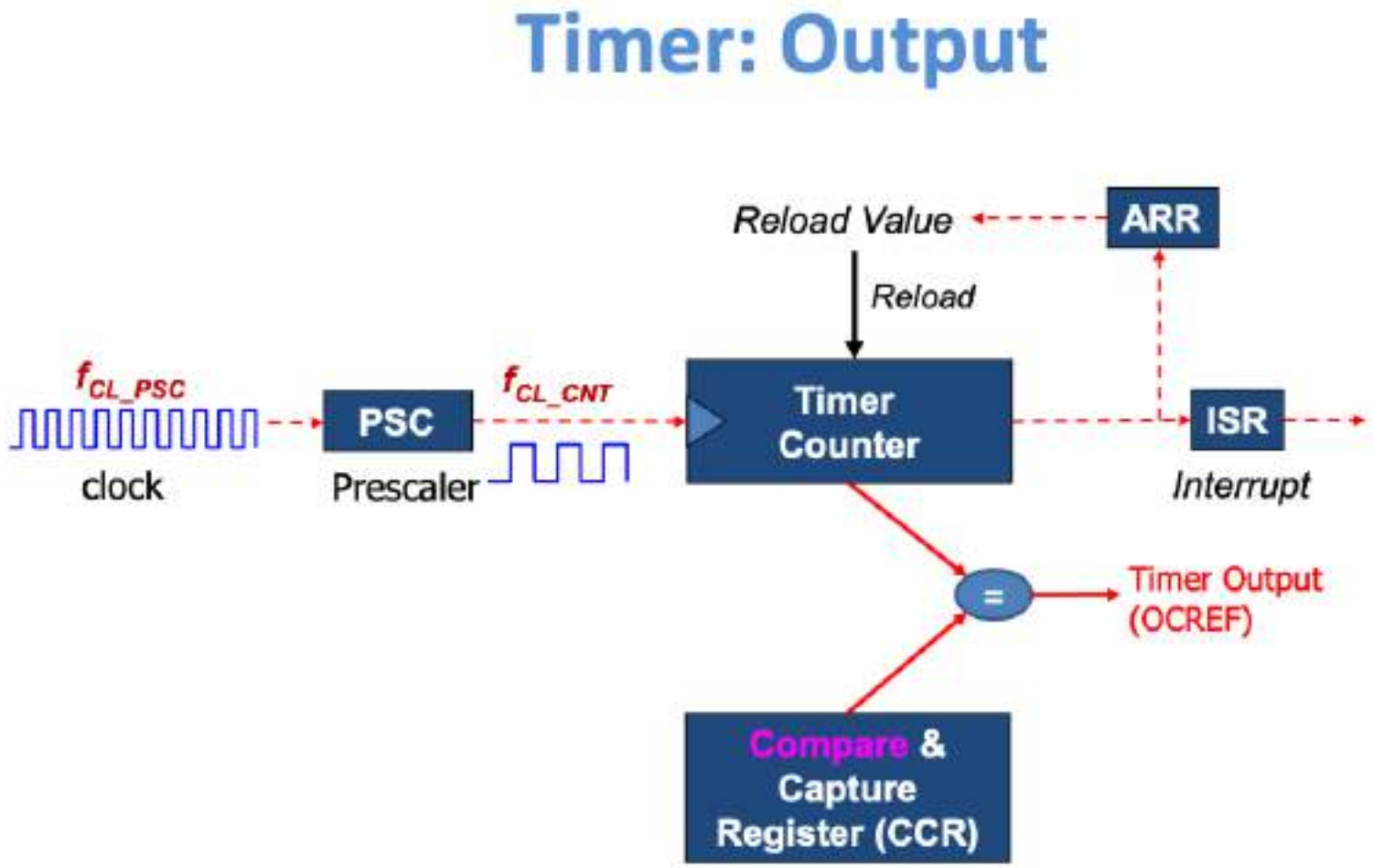
참고자료. 로체스터 공과대학 강의자료

Timer: Clock

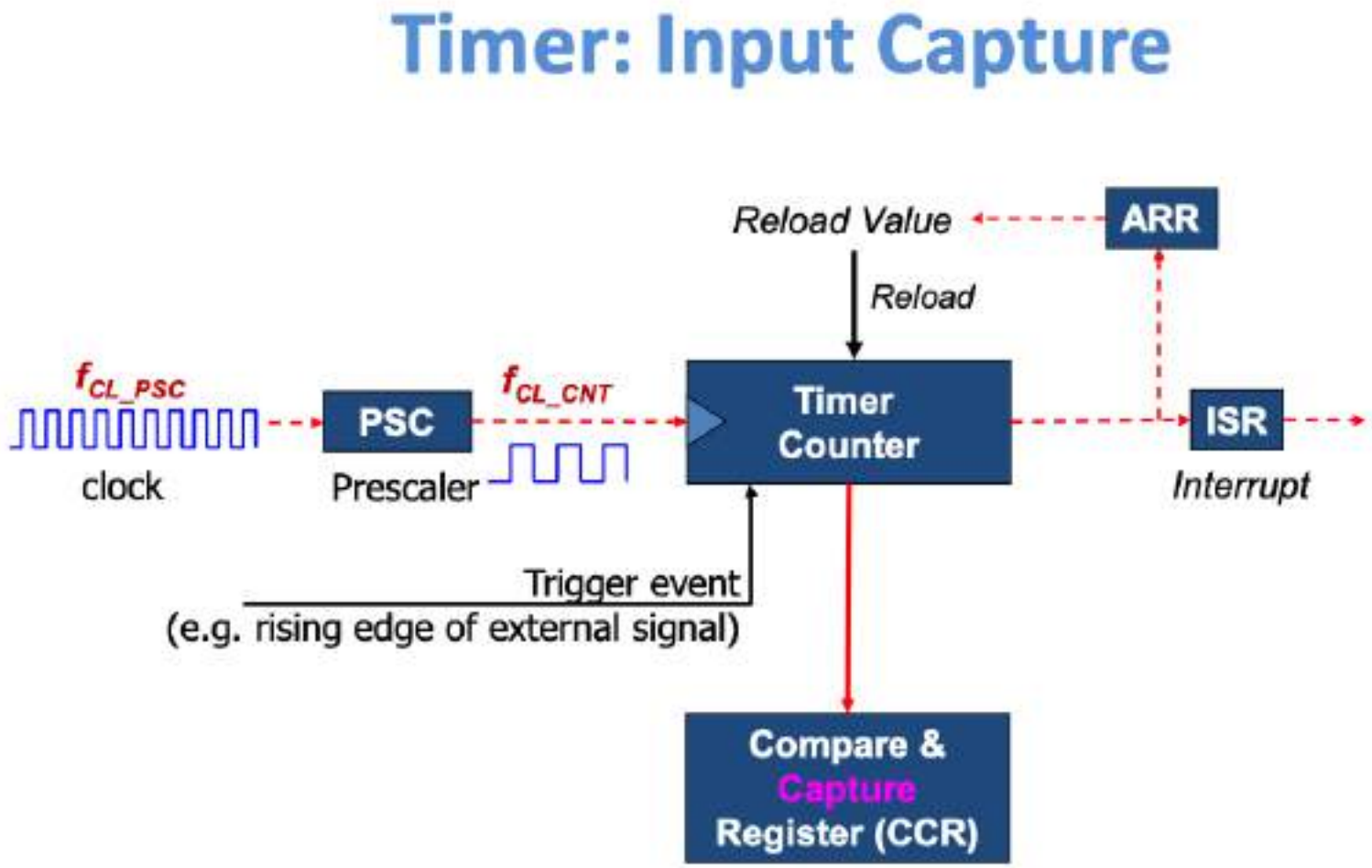


$$f_{CK\_CNT} = \frac{f_{CL\_PSC}}{PSC + 1}$$

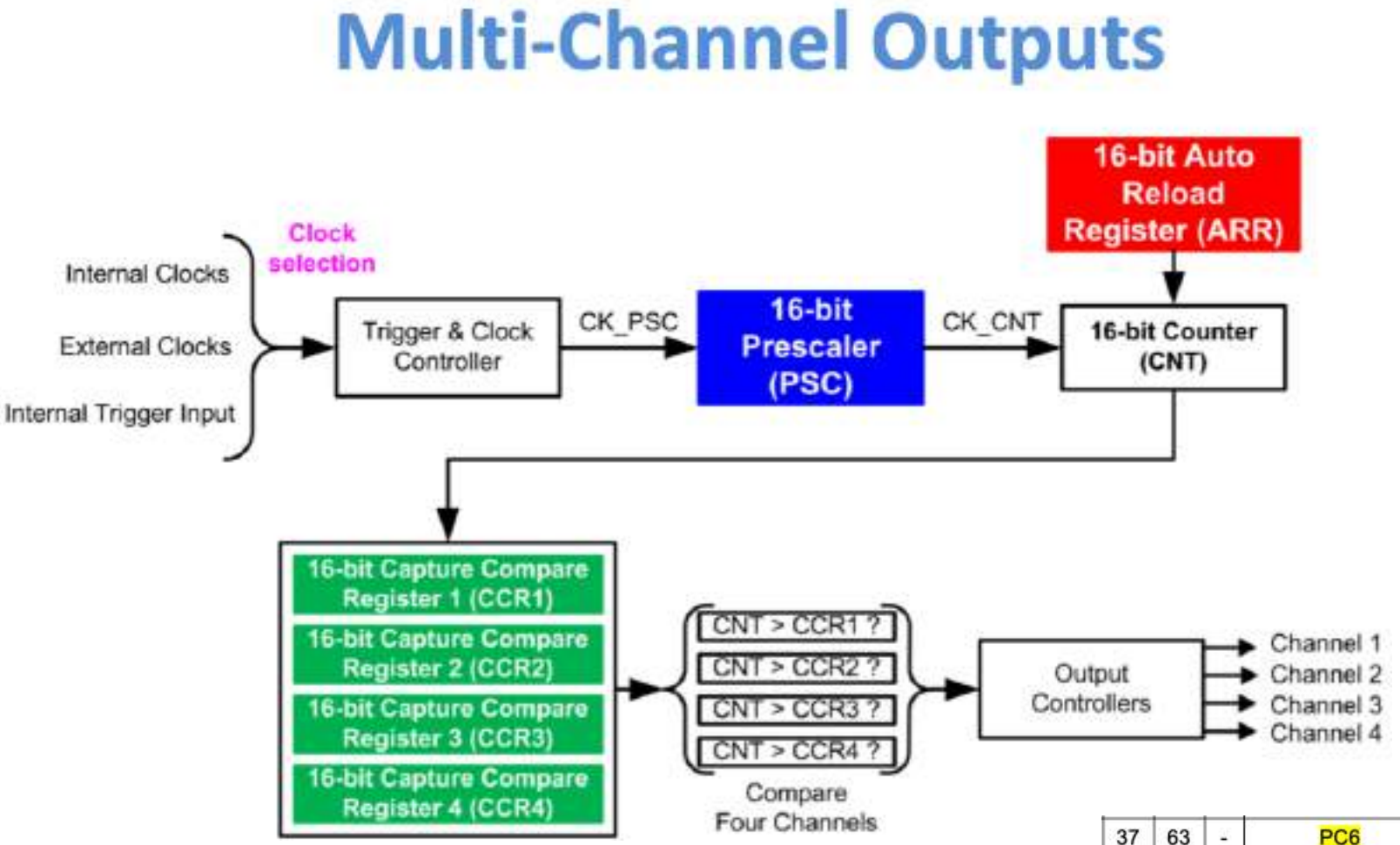
참고자료. 로체스터 공과대학 강의자료



참고자료. 로체스터 공과대학 강의자료



참고자료. 로체스터 공과대학 강의자료

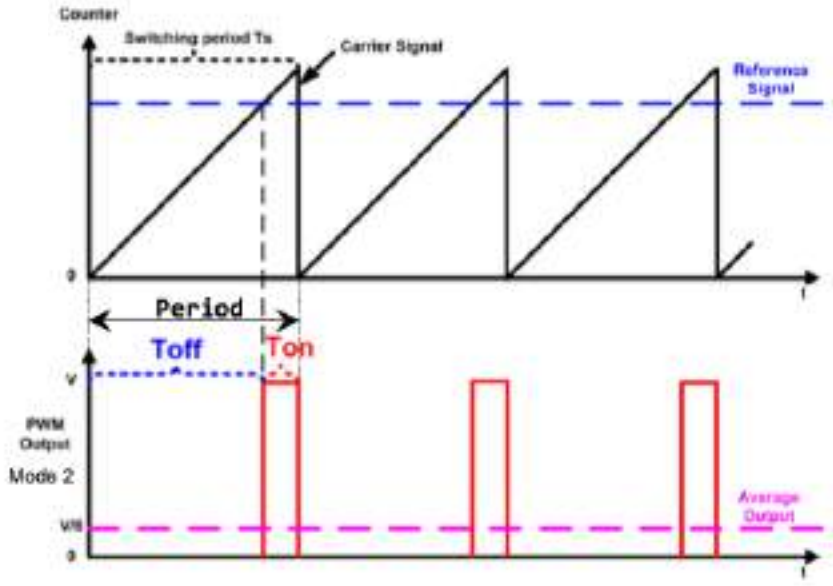


37	63	-	PC6	I/O	FT	PC6	-	TIM3_CH1
38	64	-	PC7	I/O	FT	PC7	-	TIM3_CH2
39	65	-	PC8	I/O	FT	PC8	-	TIM3_CH3
40	66	-	PC9	I/O	FT	PC9	-	TIM3_CH4

참고자료. 로체스터 공과대학 강의자료

PWM Mode

(Pulse Width Modulation)



$$\text{Period} = T_{off} + T_{on}$$

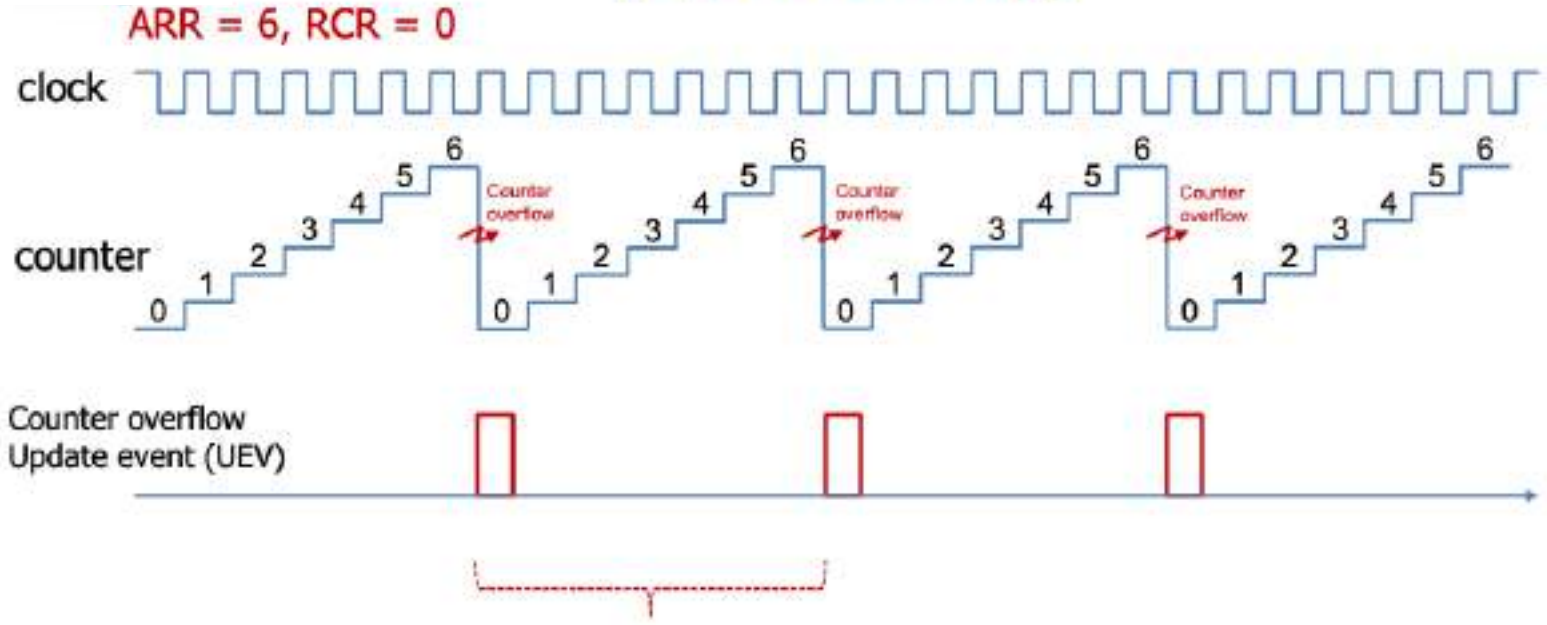
$$\text{Duty Cycle} = \frac{T_{on}}{T_{off}+T_{on}}$$

Mode	Counter < Reference	Counter ≥ Reference
PWM mode 1 (Low True)	Active Low	Inactive
PWM mode 2 (High True)	Inactive	Active High

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

Edge-aligned Mode (Up-counting)

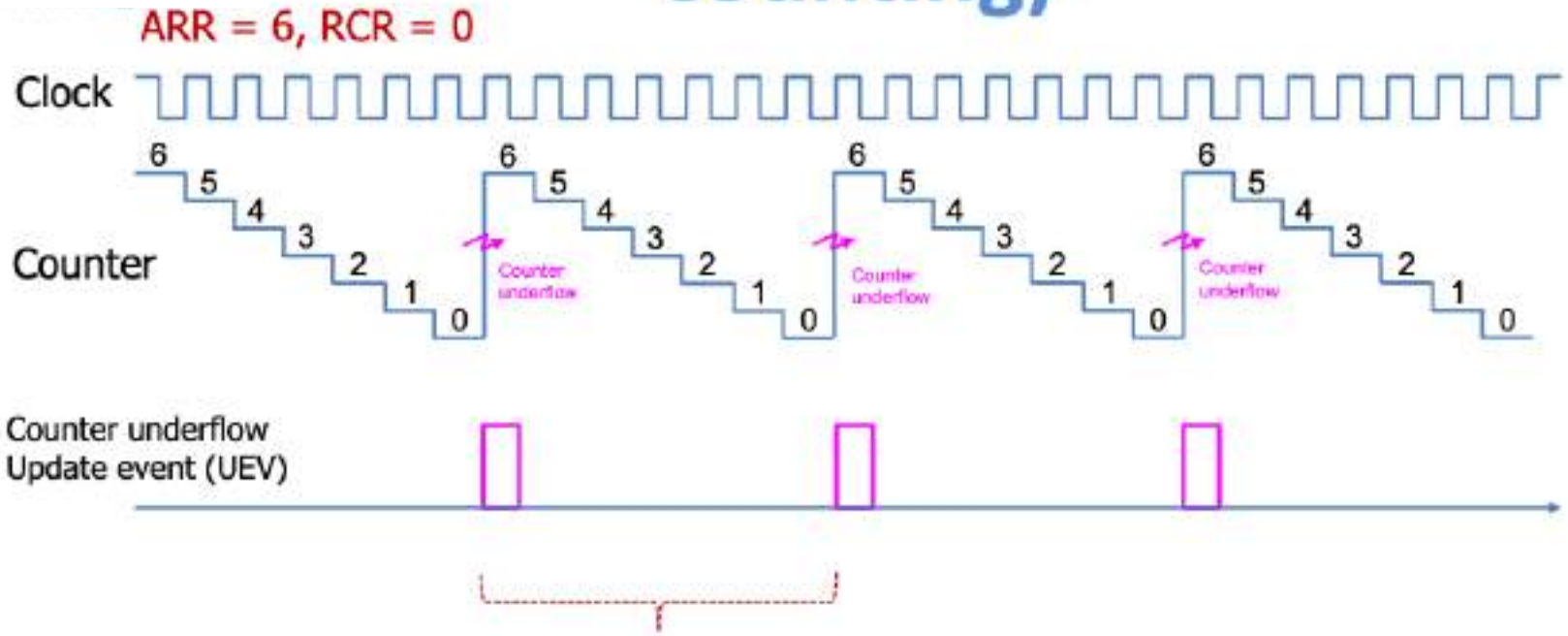


Period = (1 + ARR) * Clock Period
= 7 * Clock Period

ARR = Auto-Reload Register
UEV = Update Event
RCR = Repetition Count Register

참고자료. 로체스터 공과대학 강의자료

Edge-aligned Mode (down-counting)

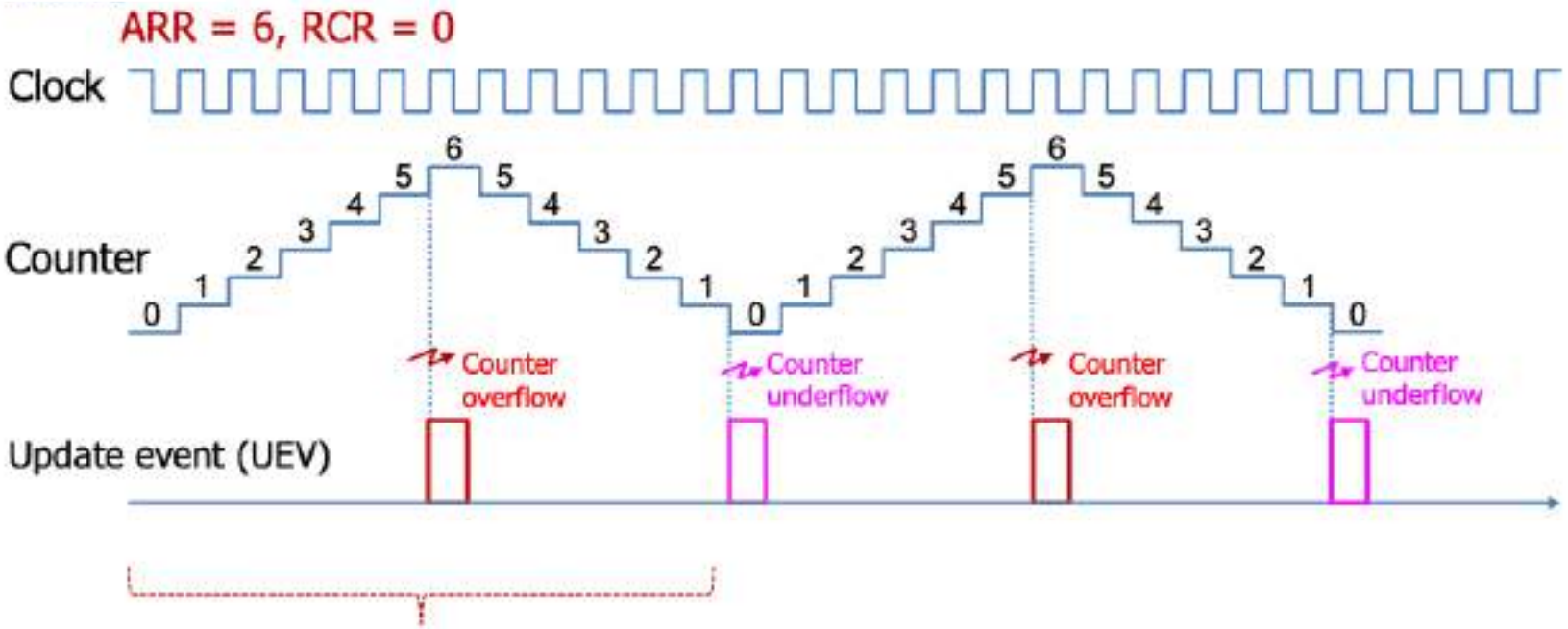



```
Period = (1 + ARR) * Clock Period
       = 7 * Clock Period
```

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

Center-aligned Mode



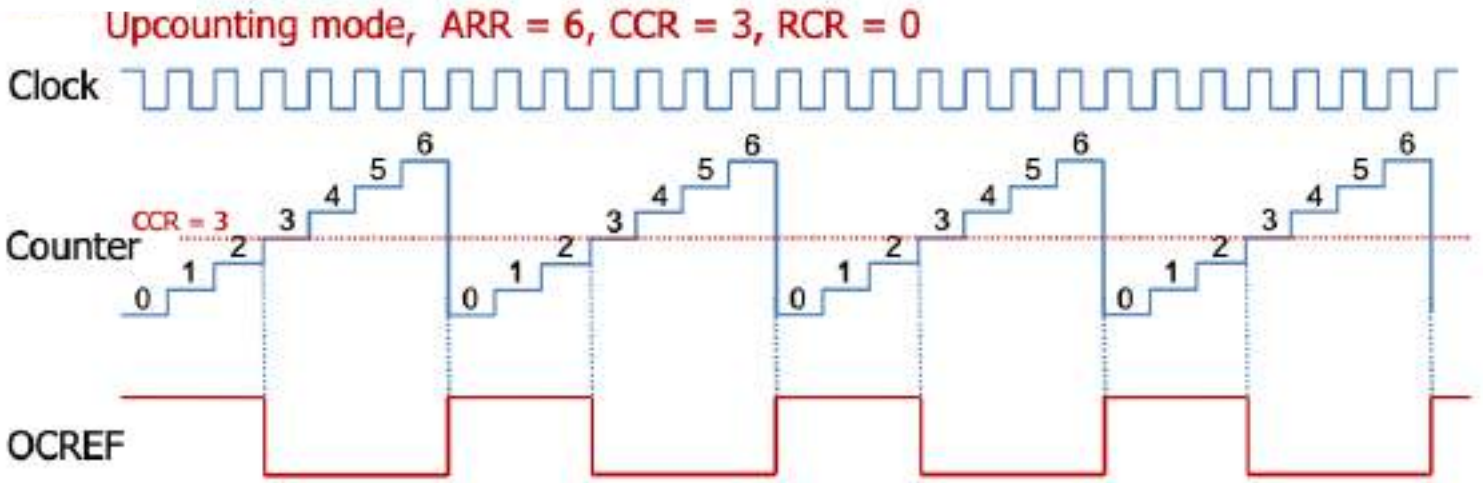

```
Period = (2 * ARR) * Clock Period
       = 12 * Clock Period
```

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

PWM
Mode 1 (Low-True)

Timer Output = { High if counter < CCR / Low if counter ≥ CCR



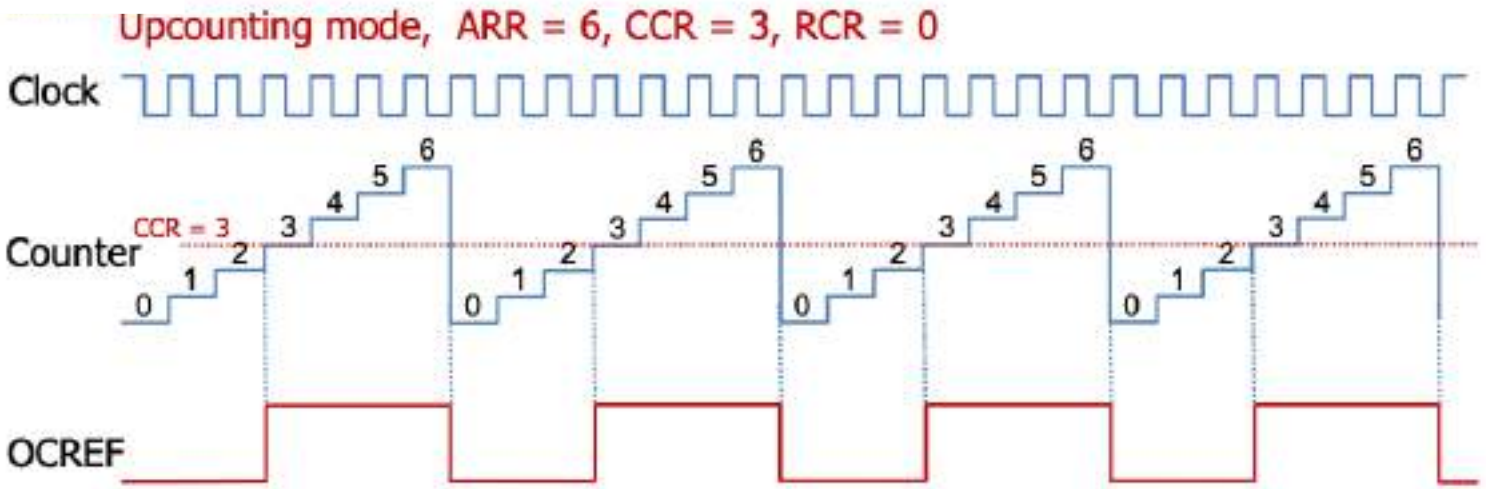
$$\text{Duty Cycle} = \frac{CCR}{ARR + 1} = \frac{3}{7}$$

$$\text{Period} = (1 + ARR) * \text{Clock Period} = 7 * \text{Clock Period}$$

참고자료. 로체스터 공과대학 강의자료

PWM
Mode 2 (High-True)

Timer Output = { Low if counter < CCR / High if counter ≥ CCR



$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR + 1} = \frac{4}{7}$$

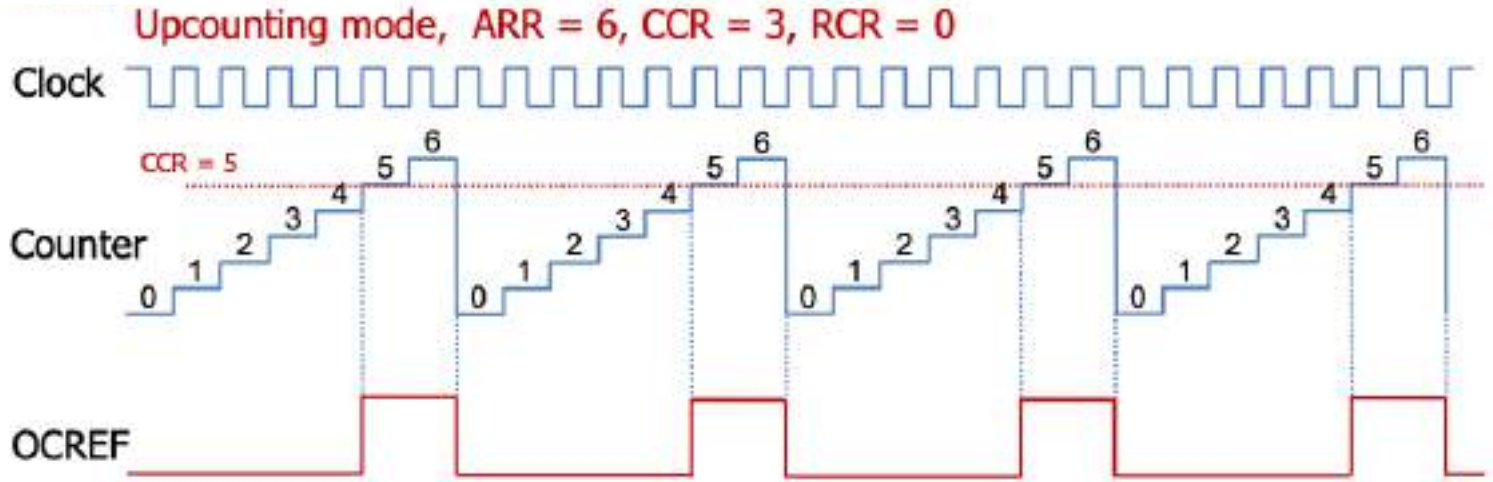
$$\text{Period} = (1 + ARR) * \text{Clock Period} = 7 * \text{Clock Period}$$

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

PWM
Mode 2 (High-True)

Timer Output = { Low if counter < CCR / High if counter ≥ CCR



$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR + 1} = \frac{2}{7}$$

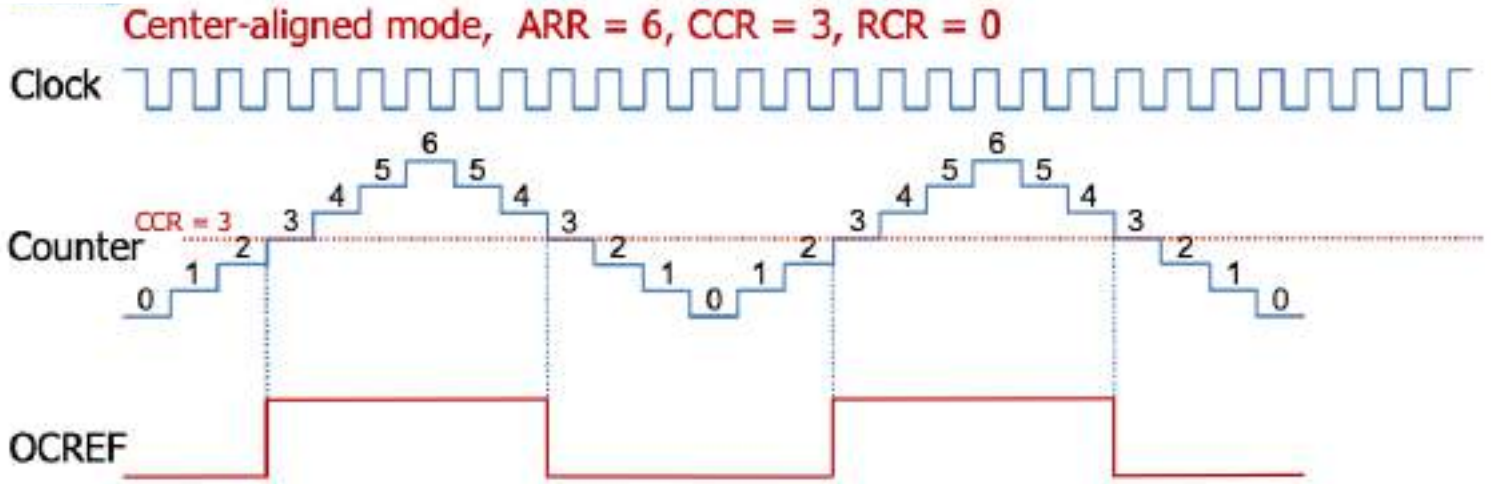
$$\text{Period} = (1 + ARR) * \text{Clock Period} = 7 * \text{Clock Period}$$

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

PWM
Mode 2 (High-True)

Timer Output = { Low if counter < CCR; High if counter ≥ CCR }



$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR} = \frac{1}{2}$$

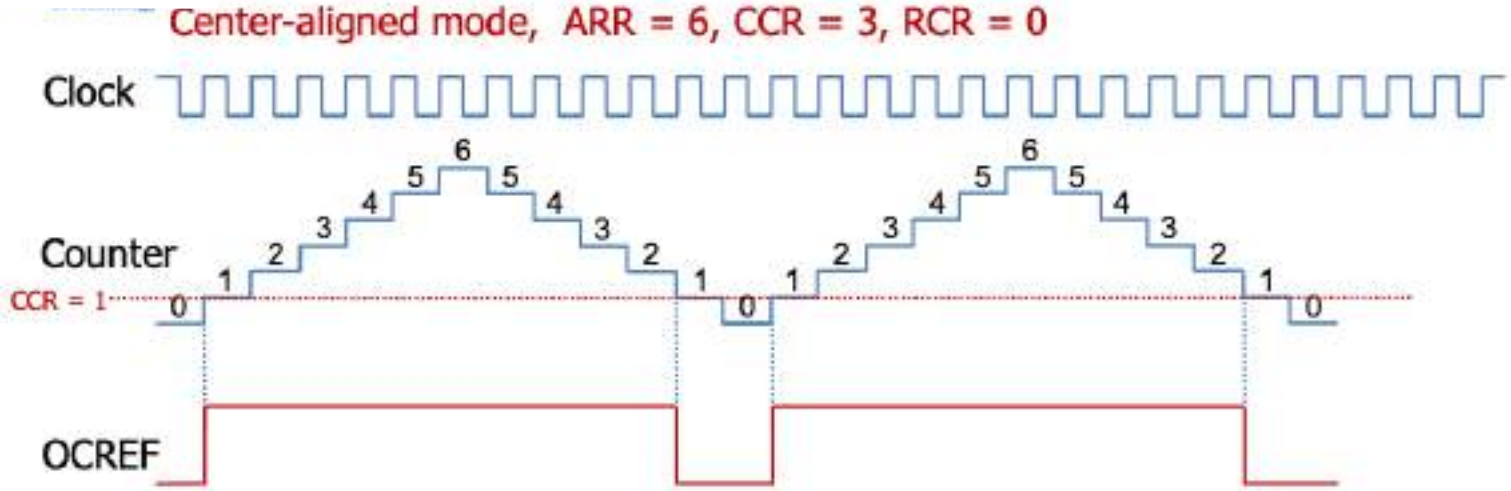
$$\text{Period} = 2 * ARR * \text{Clock Period} = 12 * \text{Clock Period}$$

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

PWM
Mode 2 (High-True)

Timer Output = { Low if counter < CCR / High if counter ≥ CCR }



$$\text{Duty Cycle} = 1 - \frac{CCR}{ARR} = \frac{5}{6}$$

$$\text{Period} = 2 * ARR * \text{Clock Period} = 12 * \text{Clock Period}$$

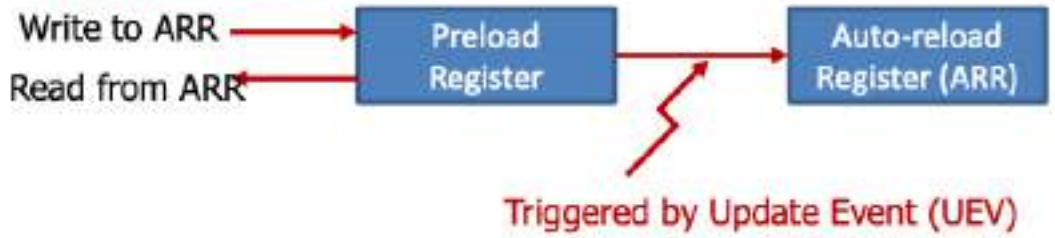
KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

Auto-Reload Register (ARR)

- Auto-Reload Preload Enable (ARPE) bit in TIMx_CR1

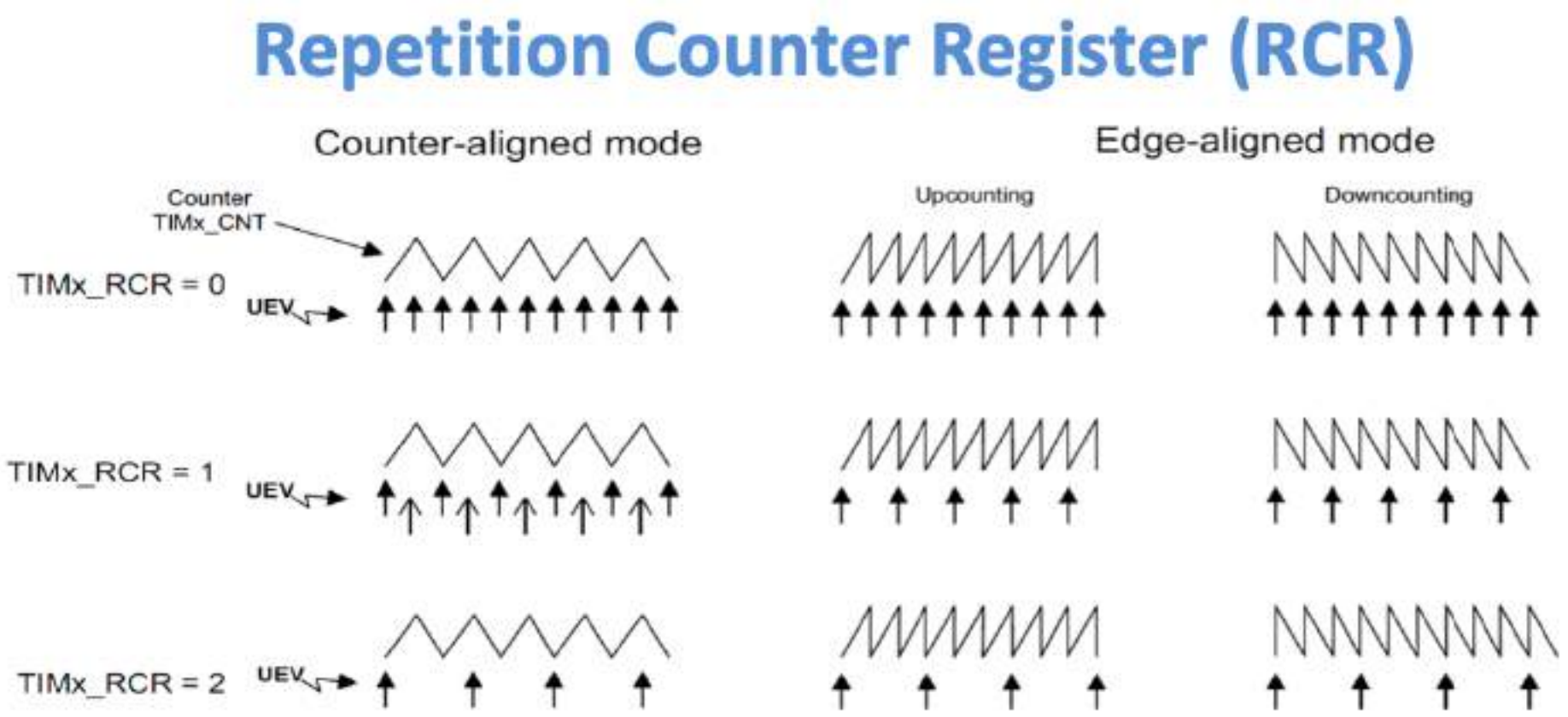
ARPE = 1 (Synchronous Update)



If UDIS bit in TIMx_CR1 is 1, UEV event is disabled.

ARPE = 0 (Asynchronous Update)

참고자료. 로체스터 공과대학 강의자료



KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료

PWM Output Polarity

Mode	Counter < CCR	Counter ≥ CCR
PWM mode 1 (Low True)	Active (Low)	Inactive
PWM mode 2 (High True)	Inactive	Active (High)

Output Polarity:

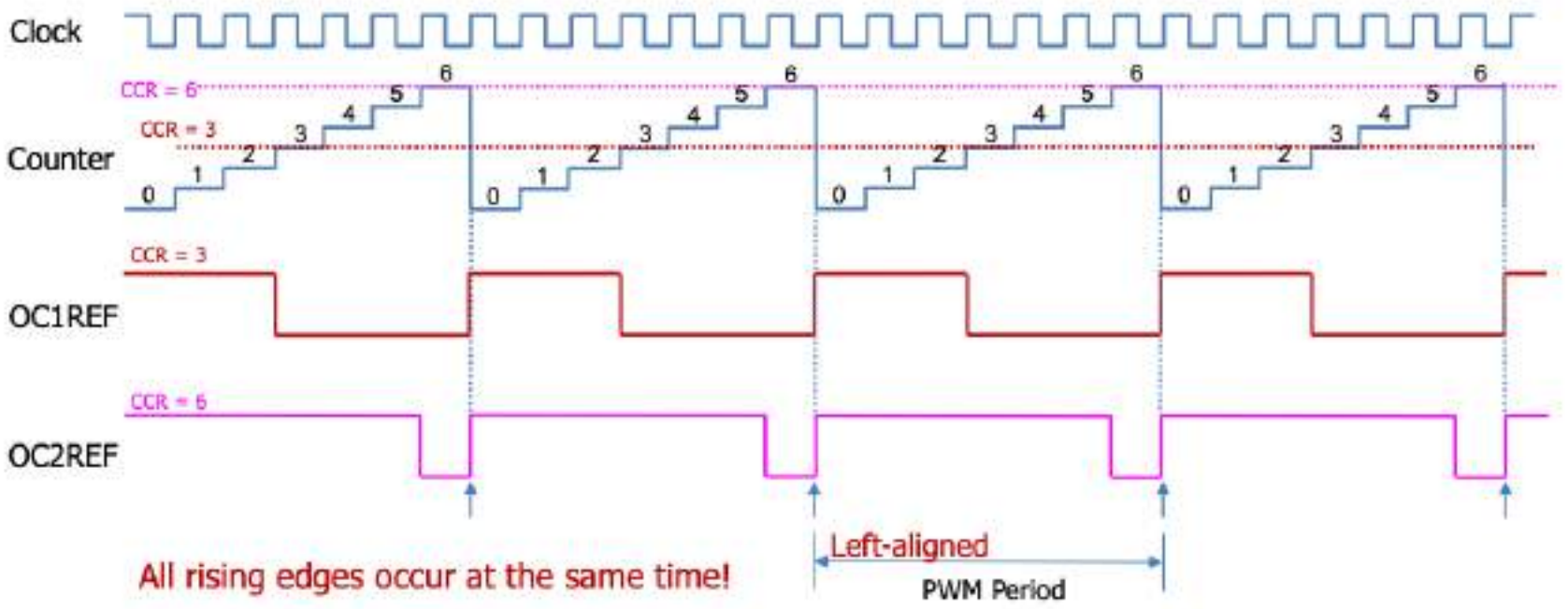
- Software can program the CCxP bit in the TIMx_CCER register

	Active	Inactive
Active High	High Voltage	Low Voltage
Active Low	Low Voltage	High Voltage

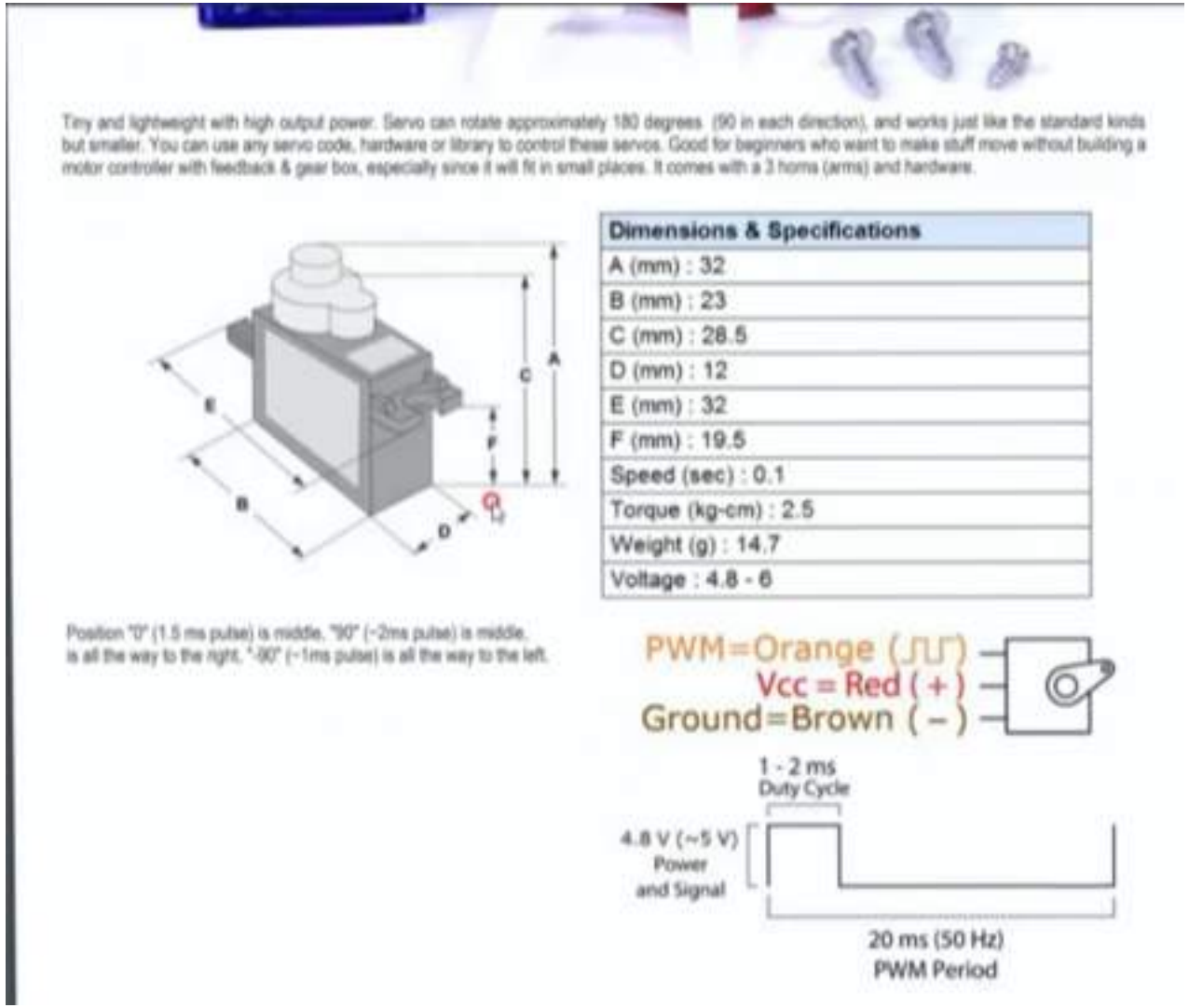
참고자료. 로체스터 공과대학 강의자료

Up-Counting: Left Edge-aligned

Upcounting mode, ARR = 6, CCR = 3, RCR = 0



참고자료. 로체스터 공과대학 강의자료



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

Dimensions & Specifications
A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

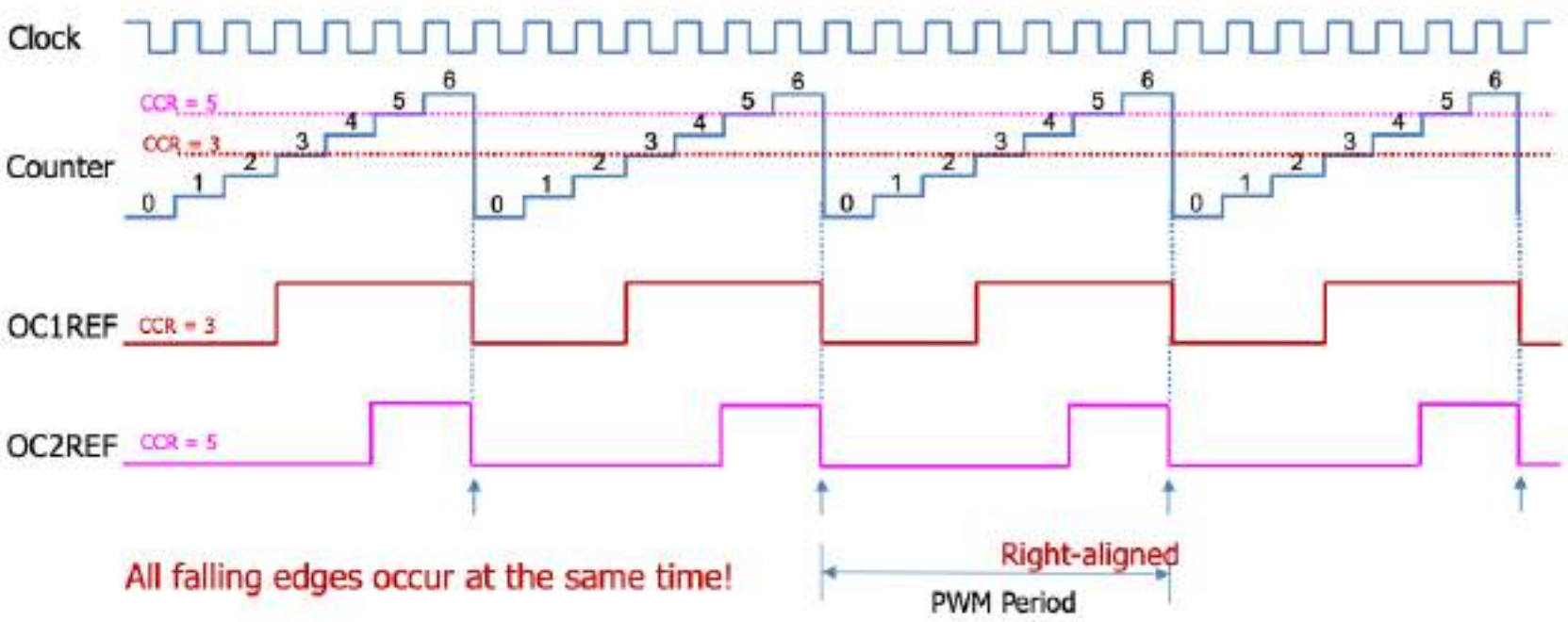
Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

KYUNG HEE UNIVERSITY

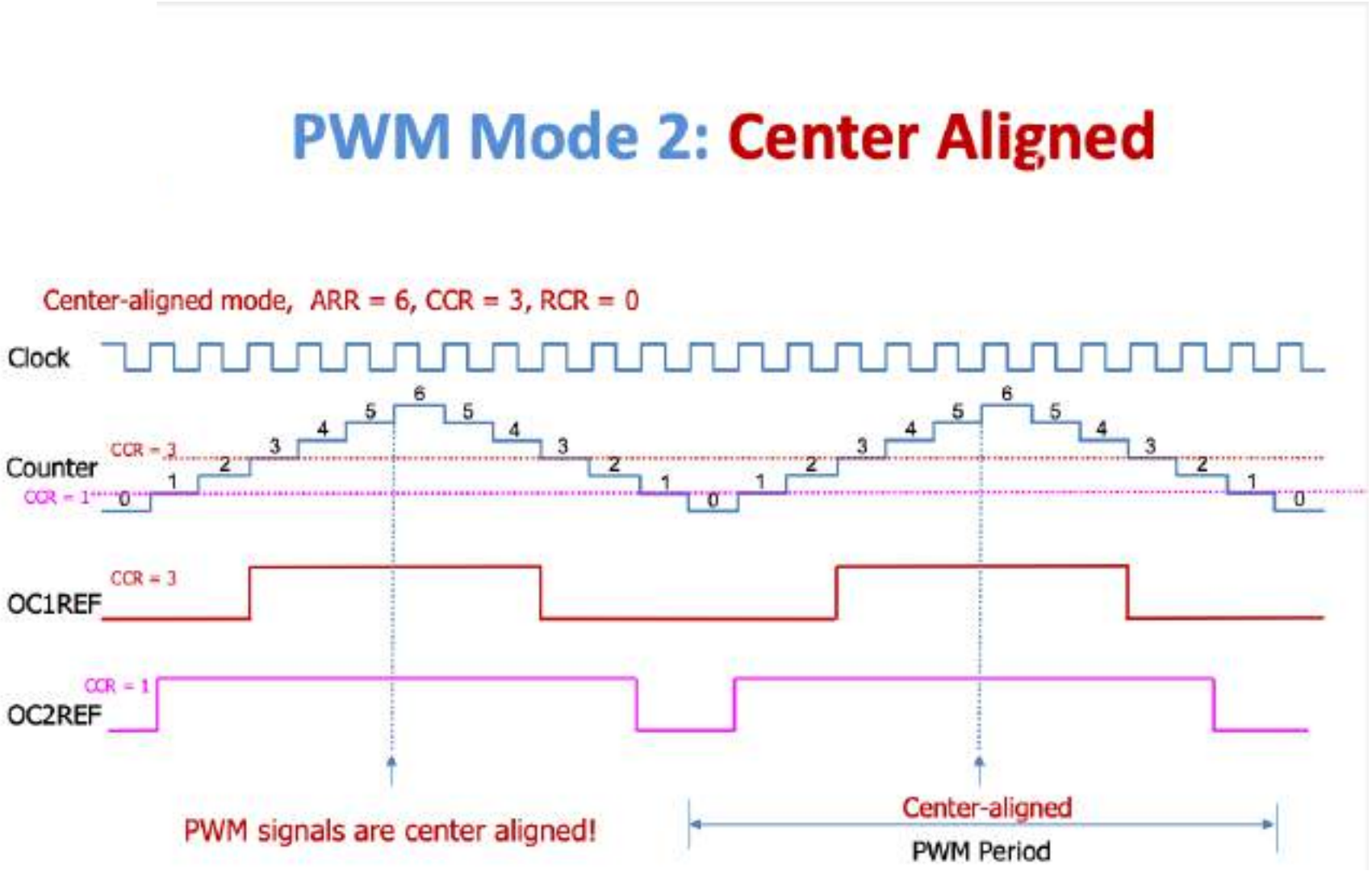
참고자료. 로체스터 공과대학 강의자료

PWM Mode 2: Right Edge-aligned

Upcounting mode, ARR = 6, CCR = 3, RCR = 0

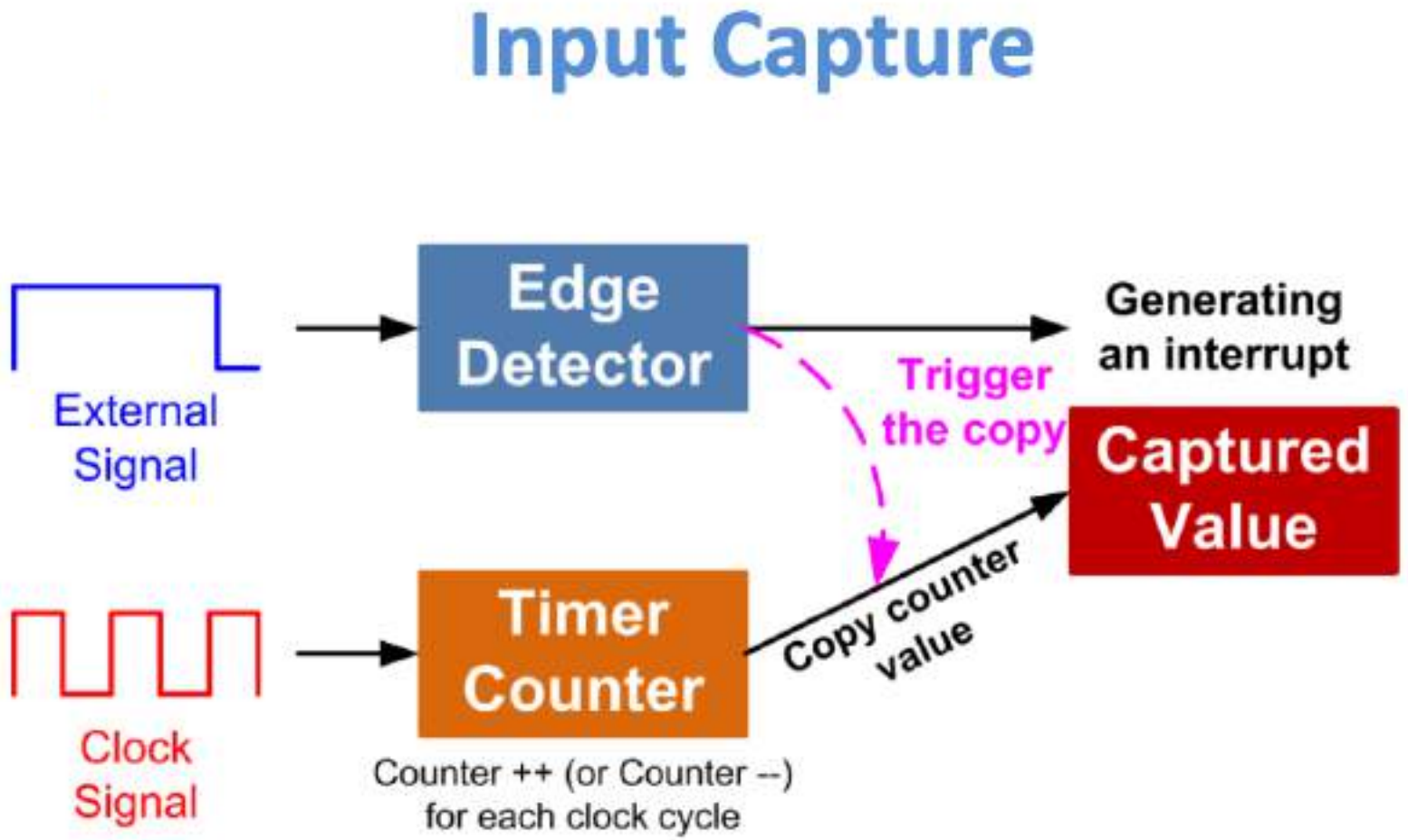


참고자료. 로체스터 공과대학 강의자료

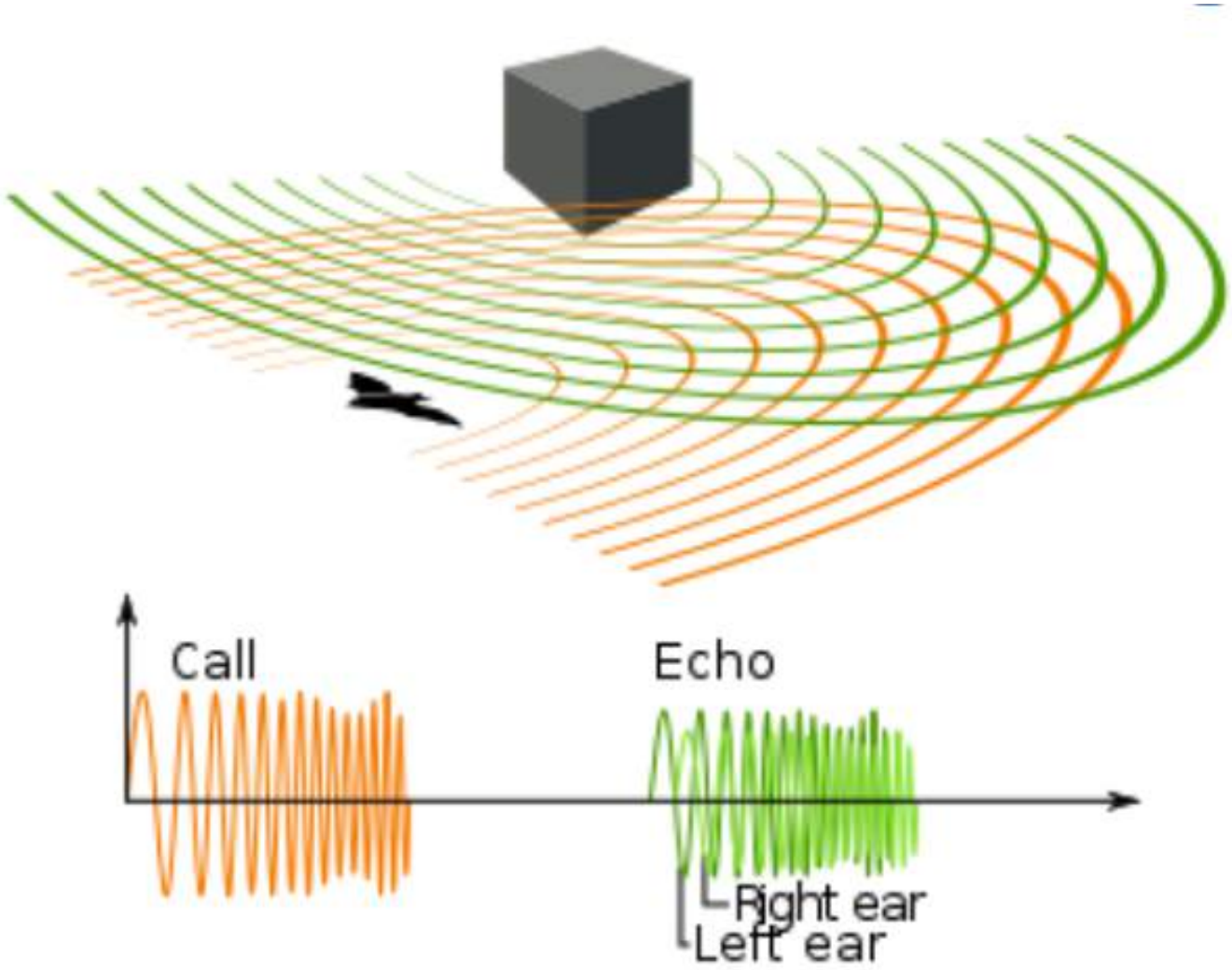


KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료



참고자료. 로체스터 공과대학 강의자료



참고자료. 로체스터 공과대학 강의자료

Ultrasonic Distance Sensor

$$Distance = \frac{Round\ Trip\ Time \times Speed\ of\ Sound}{2}$$

$$= \frac{Round\ Trip\ Time(\mu s) \times 10^{-6} \times 340 m/s}{2}$$

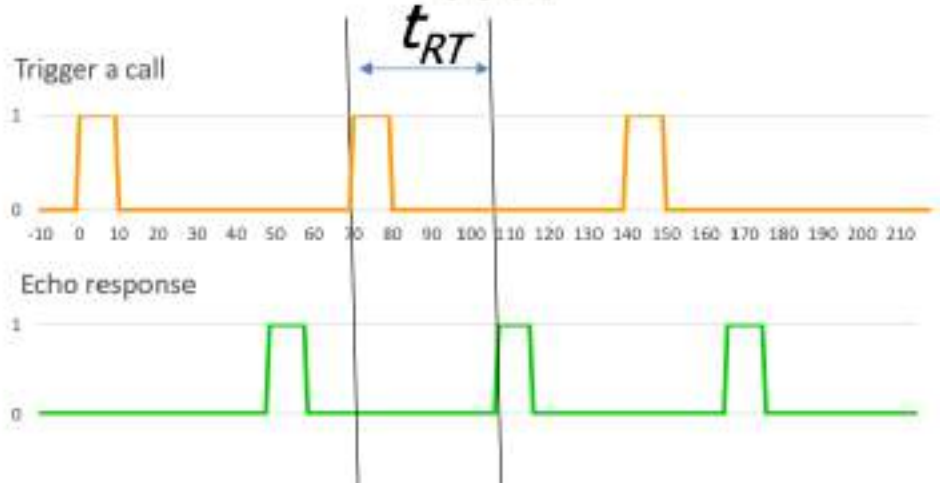
$$= \frac{Round\ Trip\ Time(\mu s)}{58}$$

참고자료. 로체스터 공과대학 강의자료

Ultrasonic Distance Sensor

The delay from triggered chirp to echo response is the round-trip time t_{RT}.

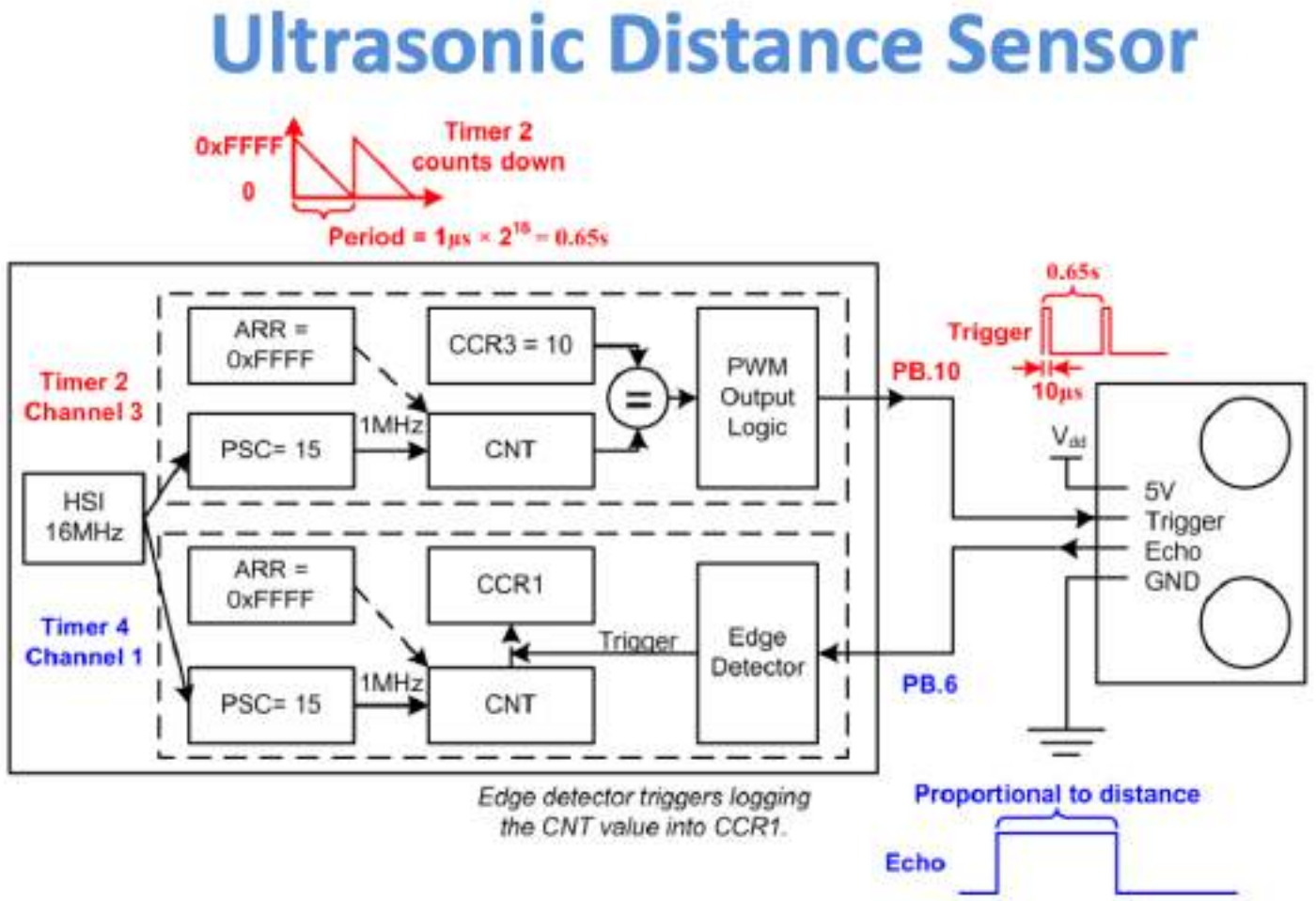
$$Distance\ (cm) = \frac{Round\ trip\ time\ (\mu s)}{58}$$



If $t_{RT>}$ is 60ms, no obstacle is detected.

KYUNG HEE UNIVERSITY

참고자료. 로체스터 공과대학 강의자료



8.1 범용 타이머의 구조 및 기능

범용 타이머의 구조

- **프리스케일러(PSC: Prescaler)**

16비트의 프리스케일러는 공급되는 클럭 (TIMx_CLK, 또는 CK_PSC)을 1~65,536의 값으로 나누어 카운터의 동작 클럭 (CK_CNT)을 만들어내는 분주기

- **카운터(CNT: Counter)**

16비트의 카운터는 타이머의 핵심적인 요소로써, 공급되는 클럭을 이용한 업, 다운, 업/다운 카운팅이 가능함

- **오토 리로드 레지스터(ARR: Auto Reload Register)**

- 업 카운터의 경우 '카운터 (CNT) 값 = ARR의 설정 값'이 되면 CNT는 0부터 다시 카운팅을 함

- 다운 카운터의 경우 '카운터 (CNT) 값 = 0 '이 되면 CNT는 ARR의 설정 값부터 다시 다운 카운팅을 함

8.1 범용 타이머의 구조 및 기능

범용 타이머의 구조

- **캡처/비교기(CCR: Capture/Compare Register)**

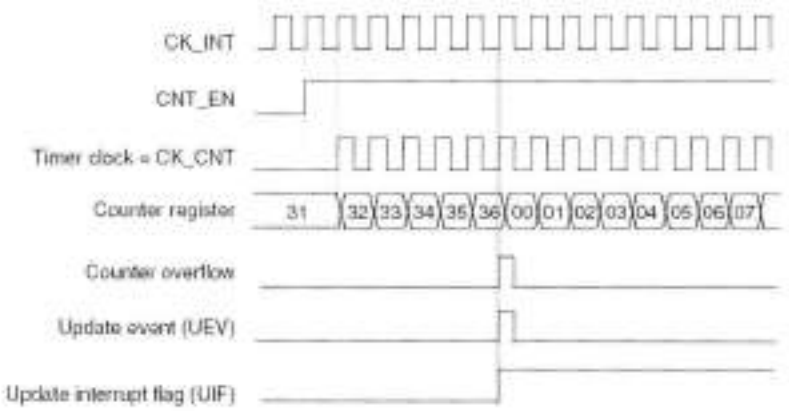
- 입력 신호가 주어질 때 카운터 (CNT) 값을 캡처
- '카운터 (CNT) 값 = CCR의 설정 값' 이 되면 인터럽트를 발생하거나 출력 채널로 0 또는 1을 출력

8.1 범용 타이머의 구조 및 기능

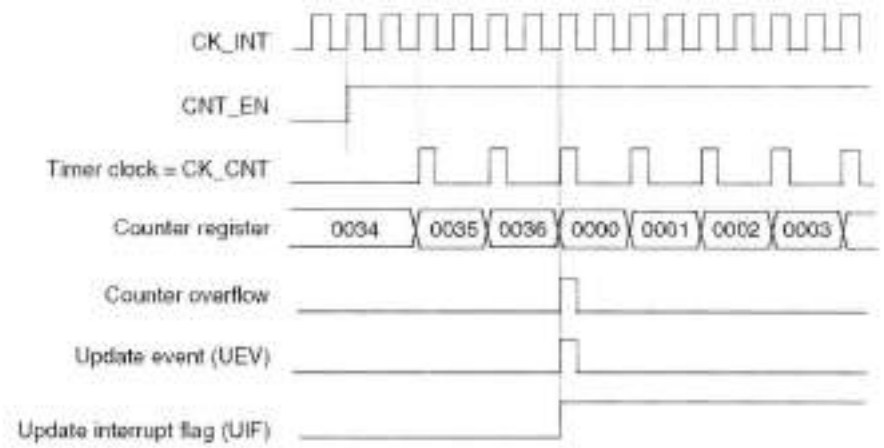
범용 타이머의 주요기능 - (1) 카운터 모드

- **업 카운팅(Up counting) 모드**

- 카운터(CNT)의 값이 증가하면서 카운팅을 하는 모드
- CNT = 0부터 시작하여 CNT = ARR(오토-리로드)값까지 증가한 후, 다시 0부터 카운팅을 시작하는 작업을 계속하여 반복함
- CNT = 0이 될 때 오버플로우, 업데이트 이벤트(UEV)와 업데이트 인터럽트(UI)가 발생함



△ 그림 8-1-3 업 카운팅 모드의 동작 시간도(오토 리로드 ARR=36, 분주비=1인 경우)



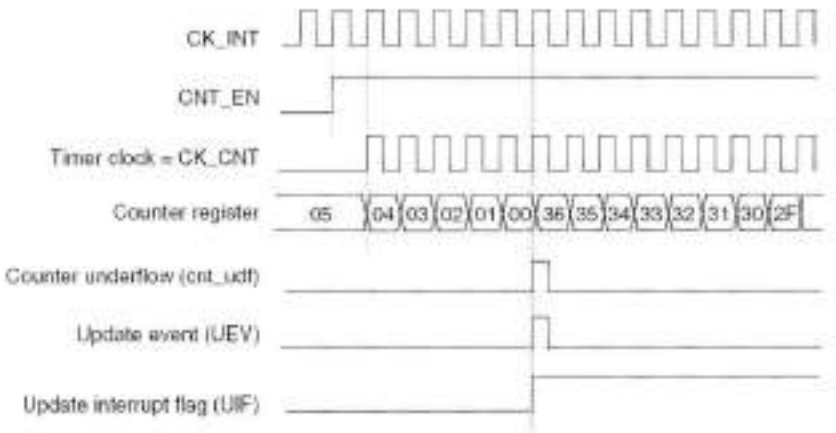
△ 그림 8-1-4 업 카운팅 모드의 동작 시간도(오토 리로드 ARR=36, 분주비=2인 경우)

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (1) 카운터 모드

- 다운 카운팅(Down counting) 모드

- 카운터 (CNT) 의 값이 감소하면서 카운팅을 하는 모드
- CNT = ARR부터 CNT=O까지 감소한 후에 다시 CNT =ARR부터 카운팅을 시작하는 작업을 계속하여 반복함
- CNT = ARR이 될 때 언더플로우 업데이트 이벤트(UEV)와 업데이트 인터럽트(UI)가 발생함



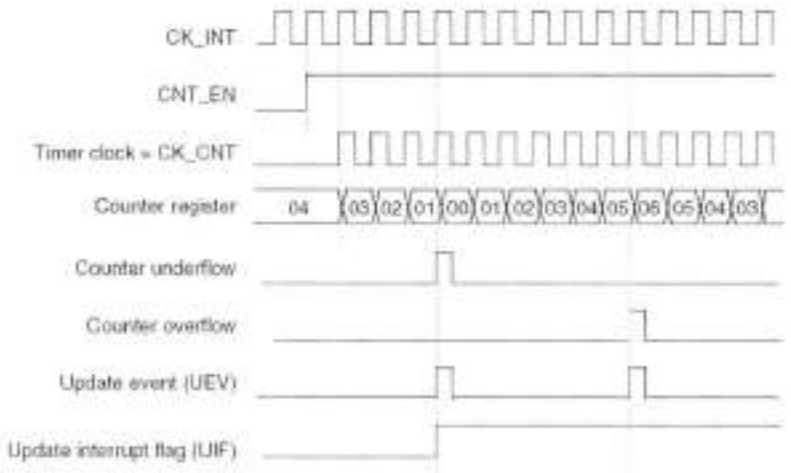
△ 그림 8-1-5 다운 카운팅 모드의 동작 시간도(오토 리로드 ARR=36, 분주비=1인 경우)

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (1) 카운터 모드

- **업/다운 카운팅(Up/Down counting) 모드**

- 카운터의 값(CNT) 이 증가한 후 다시 감소하면서 카운팅을 하는 모드
- 0부터 오토-리로드 값(ARR)까지 증가한 후에 다시 0까지 감소함. 이 후, 다시 증가 -> 감소 -> 증가 -> 감소 .. 계속 카운팅
- 업 카운팅을 할 때는 CNT = ARR이 될 때 오버 플로우, 업데이트 이벤트(UEV)와 업데이트 인터럽트(UI)가 발생함
- 다운 카운팅을 할 때는 CNT =0 이 될 때 언더 플로우, 업데이트 이벤트(UEV)와 업데이트 인터럽트(UI) 가 발생함



△ 그림 8-1-6 업/다운 카운팅 모드의 동작 시간(오토 리로드 ARR=6, 분주비=1인 경우)

KYUNG HEE UNIVERSITY

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (1) 카운터 모드

- TIM_Base_InitTypeDef 구조체에서 타이머가 가질 수 있는 카운팅 모드

```
TIM_COUNTERMODE_UP              : 업 카운팅 모드
TIM_COUNTERMODE_DOWN            : 다운 카운팅 모드
TIM_COUNTERMODE_CENTERALIGNED1  : 센터 얼라인 모드1. 이 모드에서는 다운 카운팅 때 OC(출력비교) 인터럽트가
                                  발생한다.
TIM_COUNTERMODE_CENTERALIGNED2  : 센터 얼라인 모드2. 이 모드에서는 업 카운팅 때 OC 인터럽트가 발생한다.
TIM_COUNTERMODE_CENTERALIGNED3  : 센터 얼라인 모드3. 이 모드에서는 업/다운 카운팅 때 모두 OC 인터럽트가
                                  발생한다.
```

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (2) 입력 캡처(Input capture) 모드

외부의 입력 값이 들어오는 순간 그 때의 카운터 값을 캡처하는 동작을 하는 모드

입력 캡처신호인 ICx(x=1 ~ 4)가 입력되면 그 때의 카운터 값이 캡처/비교기에 저장되며
캡처 플래그(CCxIF)가 설정됨

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (3) 출력 비교(OC: Output Compare) 모드

카운터 (CNT) 의 출력값이 캡처/비교기에 설정된 비교값(CCRx)과 일치할 때 인터럽트(CCx)나 해당 핀에 출력(OCx)이 발생하는 모드. 핀의 출력 값은 다음과 같은 모드로 설정할 수 있음

- Output compare timing

- 캡처/비교기에 설정된 비교 값(CCRx)이 출력에 아무런 영향을 주지 않음. 일반적인 타이머의 동작과 유사함

- Output compare active

- 카운터(CNT) 의 값 = 캡처/비교기의 설정 값(CCRx) 일 때, 출력(OCx)값이 high

- Output compare inactive

- 카운터(CNT) 의 값 = 캡처/비교기의 설정 값(CCRx) 일 때, 출력(OCx)값이 low

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (3) 출력 비교(OC: Output Compare) 모드

카운터 (CNT) 의 출력값이 캡처/비교기에 설정된 비교값(CCRx)과 일치할 때 인터럽트(CCx)나 해당 핀에 출력(OCx)이 발생하는 모드. 핀의 출력 값은 다음과 같은 모드로 설정할 수 있음

- Output compare toggle

- 카운터 (CNT) 의 값 = 캡처 /비교기의 설정값(CCRx) 일 때, 출력(Ocx)값이 toggle

- Output compare forced active/inactive

- 카운터 (CNT) 의 값과 상관없이 출력(OCx)값이 high (active mode) 또는 low(inactive mode)

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (4) PWM(Pulse Width Modulation) 출력 모드

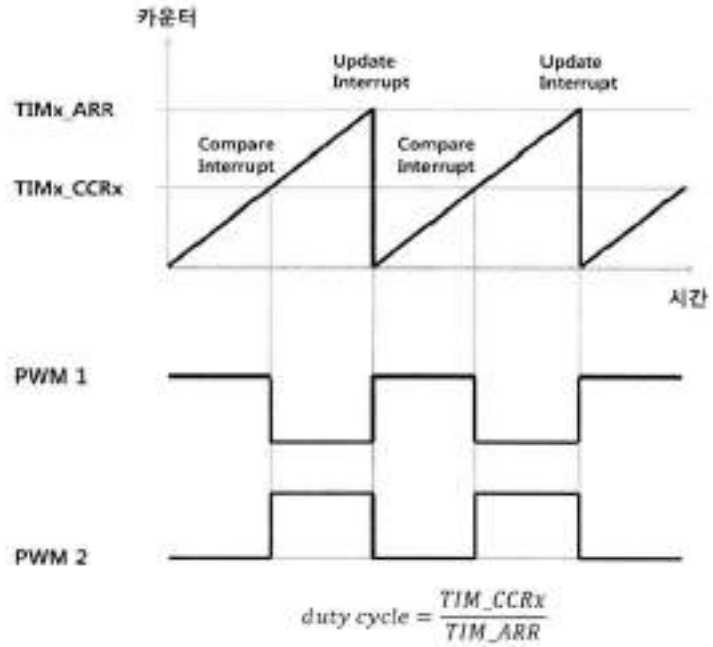
PWM 출력을 발생시키는 모드

출력 주파수는 오토 리로드 값(ARR)에 의해 결정되며 듀티비(duty ratio)는 캡처/비교기 값(CCR)에 의해 결정됨

- **PWM 출력 발생 모드에 따른 구분(PWM mode1, PWM mode2)**

① PWM mode 1
- up-counting일 때 : CNT < CCRx이면 : active 출력
  CNT ≥ CCRx이면 : inactive 출력
- down-counting일 때 : CNT ≤ CCRx이면 : active 출력
  CNT > CCRx이면 : inactive 출력

② PWM mode 2 : PWM mode 1과 출력이 반대이다.
- up-counting일 때 : CNT < CCRx, : inactive 출력
  CNT ≥ CCRx : active 출력
- down-counting일 때 : CNT ≤ CCRx : inactive 출력
  CNT > CCRx : active 출력



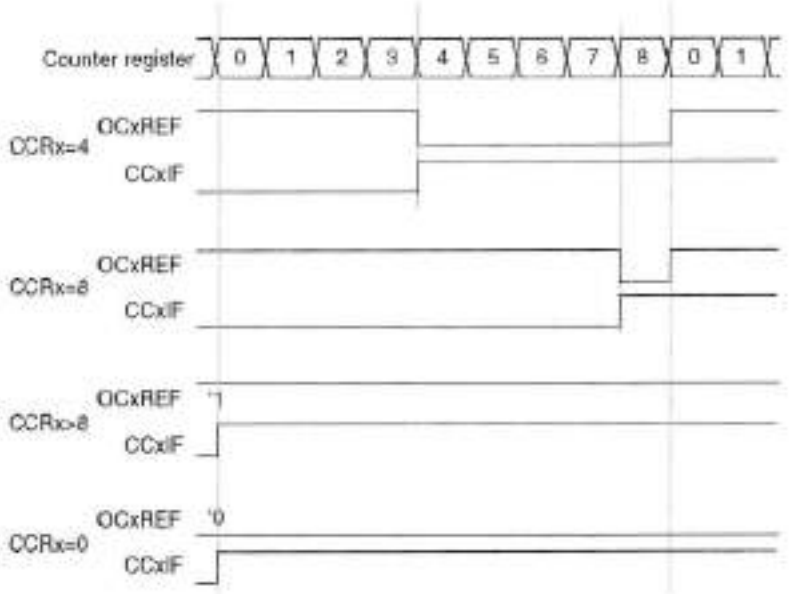
$$duty\ cycle = \frac{TIM\_CCRx}{TIM\_ARR}$$

△ 그림 8-1-6 PWM 모드의 동작

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (4) PWM(Pulse Width Modulation) 출력 모드

- 카운터의 동작 모드에 따른 구분(Edge-aligned 모드, Center-aligned 모드)



△ 그림 8-1-9 Edge-aligned PWM의 동작 시간도(업 카운팅, ARR=8, PWM Mode 1의 경우)

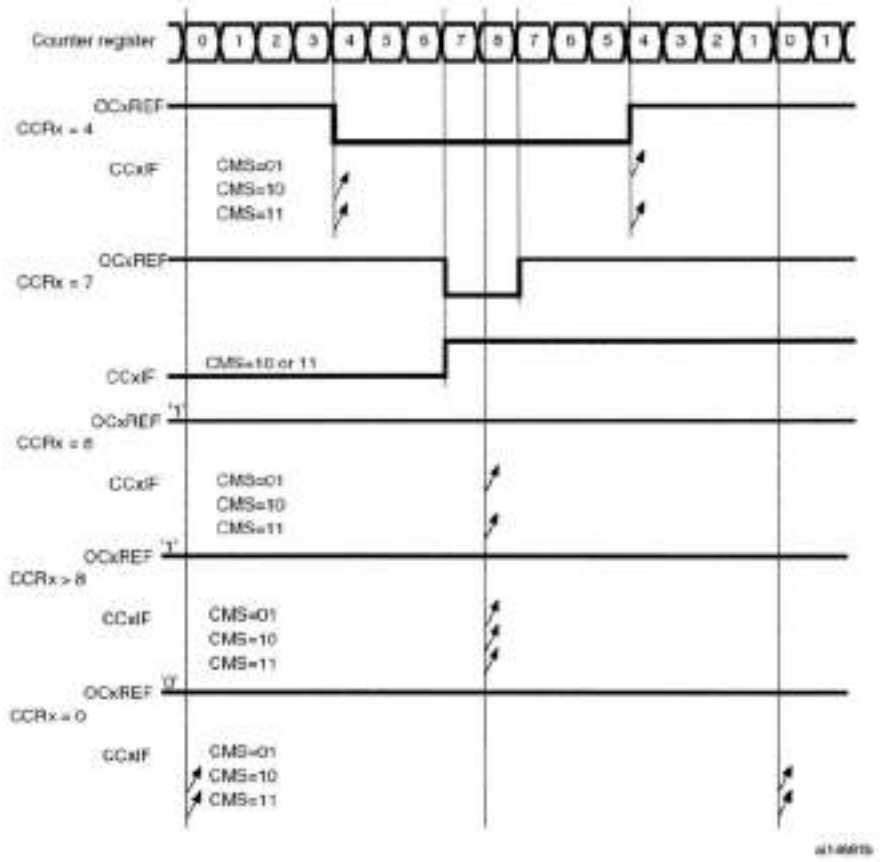
① Edge-aligned PWM 모드 : 카운터가 업 카운팅으로 설정된 경우
〈그림 8-1-9〉는 ARR=8일 경우 PWM 신호 출력의 예를 보여준다. 이 그림을 보면 카운터 값 CNT < CCRx이면 OCxREF = 1이 되고, 그 반대의 경우는 OCxREF = 0이 됨을 알 수 있다.

② Edge-aligned PWM 모드 : 카운터가 다운 카운팅으로 설정된 경우
이 경우는 카운터 레지스터(Count register) CNT > CCRx이면 OCxREF = 1이 되고, 그 반대의 경우는 OCxREF = 0이 된다.

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (4) PWM(Pulse Width Modulation) 출력 모드

- 카운터의 동작 모드에 따른 구분(Edge-aligned 모드, Center-aligned 모드)



△ 그림 8-1-10 Center-aligned PWM의 동작 시간도(ARR=8, PWM Mode 1의 경우)

③ Center-alinged PWM 모드 : 카운터가 업/다운 카운팅으로 설정된 경우

카운터는 업/다운 카운터로 동작한다. 업 카운터일때는 Edge-aligned 모드의 업 카운팅과 동일한 동작을 하며, 다운 카운터일 때는 Edge-aligned 모드의 다운 카운팅과 동일한 동작을 한다.

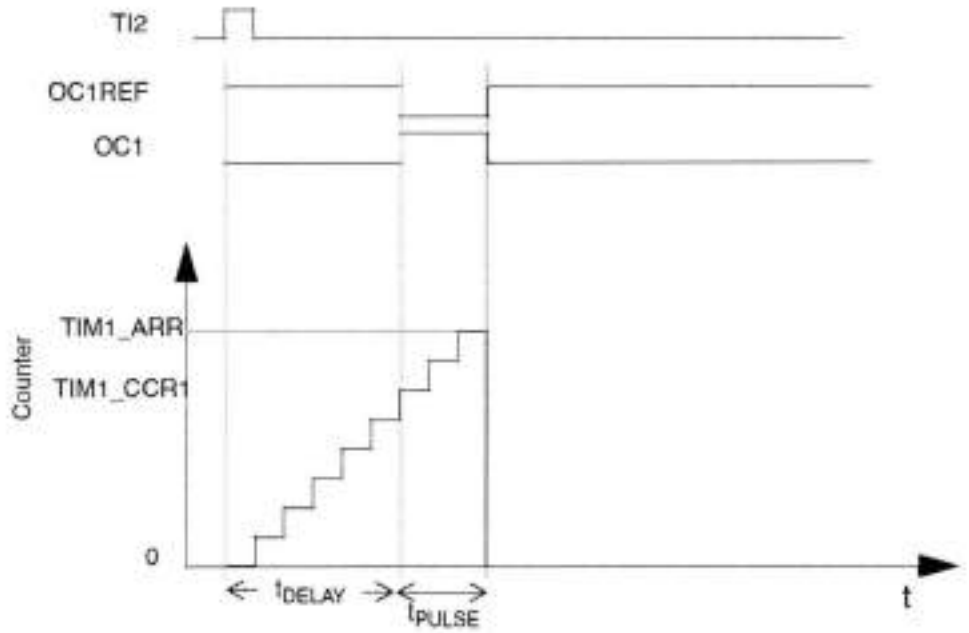
〈그림 8-1-10〉는 ARR = 8이고, PWM mode = PWM Mode1인 경우의 PWM 신호 출력의 예를 보여준다. 이 그림을 보면 CCRx = 4인 경우에는 다음과 같이 동작함을 알 수 있다.

- 업 카운터일 때 : Counter register = 4일 경우(즉, 카운터가 3에서 4로 될 때) OCxREF가 Low가 된다
- 다운 카운터일 때 : Counter register = 4일 경우(즉, 카운터가 5에서 4로 될 때) OCxREF가 High가 된다.

8.1 범용 타이머의 구조 및 기능

범용 타이머의 주요기능 - (5) 원 펄스(One pulse) 모드

- 외부 입력이 인가될 경우, 일정 시간이 지난 후에 펄스를 1번 발생시키는 모드
- TI2 입력 핀에서 상승 엣지가 검출되면 t_DELAY 시간 후에 t_PULSE 폭의 펄스가 OC1 출력 핀에 발생함
- t_DELAY 값과 t_PULSE 값은 ARR과 CCR 값의 설정에 따라 결정됨



△ 그림 8-1-11 One-pulse 모드의 동작 시간도

8.2 고급제어 타이머의 구조 및 기능

고급제어 타이머의 개요

- 범용 타이머와 동일한 특징을 가짐
- 고급제어 타이머만 가지는 특징
  - 3개의 반전된 출력 신호 발생 채널 (TIMx_CH1N ~ TIMx_CH3N)을 가짐
  - 브레이크 입력 신호 채널 (BKIN)을 가짐
  - 브레이크 입력 시 IRQ/DMA 요청 신호를 발생
  - 반복(Repeatition) 타이머 기능을 가짐

8.2 고급제어 타이머의 구조 및 기능

고급제어 타이머의 주요 기능

- 카운터 모드
- 입력 캡처(Input capture) 모드
- 출력 비교(Output Compare) 모드
- PWM 출력 모드
- 원 펄스(One pulse) 출력 모드

8.2 고급제어 타이머의 구조 및 기능

고급제어 타이머의 주요 기능

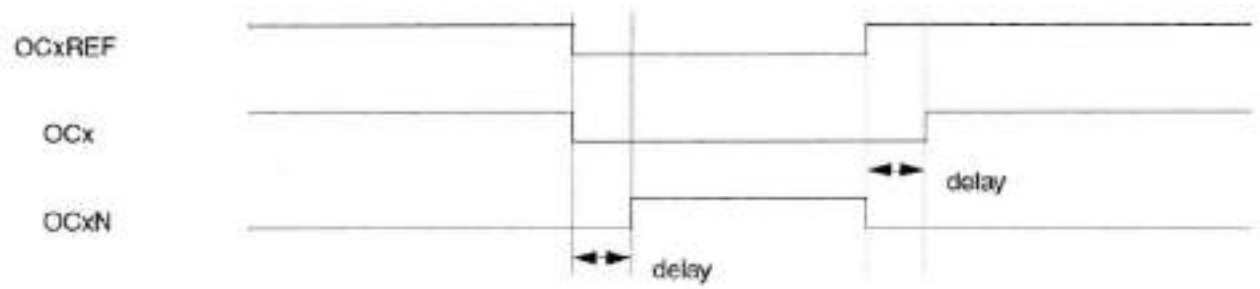
- **반전 출력(Complementary outputs)**

고급제어 타이머의 3개 채널의 출력은 OC1 ~ OC3와 반전된 출력 OC1N ~ OC3N을 가짐

OCx(x=1 ~ 3)와 OCxN는 데드 타임(dead time 또는 delay)을 줄 수 있음

3개의 채널 출력과 반전 출력은 3상 AC 모터를 구동할 때 유용하게 사용할 수 있음

모터 구동의 안정성을 위해 일반적으로 반전된 출력에 데드타임을 설정함



△ 그림 8-2-3 데드 타임이 설정된 반전 출력의 예

[3상 AC모터 구동 원리]
http://makeshare.org/bbs/board.php?bo_table=arduinomotor&wr_id=10

8.4 타이머 관련 HAL 드라이버

8.4.1.1 구조체의 종류

타이머의 동작조건을 설정하기 위한 구조체는 다음과 같다.

- TIM_HandleTypeDef
  - 타이머의 핸들 설정을 위한 구조체
- TIM_Base_InitTypeDef
  - 타이머의 초기 설정을 위한 구조체
- TIM_OC_InitTypeDef
  - 타이머의 출력 비교(OC : Output Compare) 초기설정을 위한 구조체
- TIM_OnePulse_InitTypeDef
  - 타이머의 원 펄스(one pulse) 초기설정을 위한 구조체
- TIM_IC_InitTypeDef
  - 타이머의 입력 캡쳐(IC : Input Capture) 초기설정을 위한 구조체
- TIM_Encoder_InitTypeDef
  - 타이머의 엔코더 초기설정을 위한 구조체
- TIM_ClockConfigTypeDef
  - 타이머의 클럭 동작설정을 위한 구조체
- TIM_ClearInputConfigTypeDef
  - 타이머의 클리어 입력(clear input) 설정을 위한 구조체
- TIM_SlaveConfigTypeDef
  - 타이머의 슬레이브 모드(slave mode) 설정을 위한 구조체

8.4.1.2 주요 구조체의 상세 설명

다음은 타이머 설정용 구조체 중에서 중요한 구조체에 대한 상세설명을 나타낸다. 나머지 구조체에 대한 자세한 설명은 ST사의 [HAL 드라이버 User Manual]을 참조하기 바란다.

TIM_HandleTypeDef

타이머의 핸들 설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- TIM_TypeDef* Instance
- TIM_Base_InitTypeDef Init
- HAL_TIM_ActiveChannel Channel
- DMA_HandleTypeDef* hdma
- HAL_LockTypeDef Lock
- __IO HAL_TIM_StateTypeDef State

[데이터 형의 설명]

- Instance : 사용할 타이머를 지정(예 : TIM3)
- Init : TIM의 Time Base 설정에 필요한 TIM_Base_InitTypeDef 구조체 변수
- Channel : Active channel
- hdma : DMA 핸들러 배열. 이 배열은 TIM_DMA_Handle_index에 의해 접속가능
- Lock : Locking object
- State : TIM 구동 상태

8.4 타이머 관련 HAL 드라이버

TIM_Base_InitTypeDef

타이머의 Time Base 초기설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- uint32_t Prescaler
- uint32_t CounterMode
- uint32_t Period
- uint32_t ClockDivision
- uint32_t RepetitionCounter

[데이터 형의 설명]

- Prescaler : 프리스케일러 값은 TIM 클럭에 의해 나눠진다. 이 값은 Min_Data = 0x0000에서 Max_Data = 0xFFFF까지 가질 수 있다.
- CounterMode : 카운터 모드를 설정하며 가질 수 있는 값은 다음과 같다.
  TIM_COUNTERMODE_UP(업 카운터로 동작)
  TIM_COUNTERMODE_DOWN(다운 카운터로 동작)
  TIM_COUNTERMODE_CENTERALIGNED1(업/다운 카운터로 동작, 다운 카운팅 때 OC 인터럽트가 발생)
  TIM_COUNTERMODE_CENTERALIGNED2(업/다운 카운터로 동작, 업 카운팅 때 OC 인터럽트가 발생)
  TIM_COUNTERMODE_CENTERALIGNED3(업/다운 카운터로 동작, 업/다운 카운팅 때 모두 OC 인터럽트가 발생)
- Period : 다음 업데이트 이벤트가 발생할 때 액티브한 ARR(Auto-Reload Register)에 로드되는 값이다. 그러면 카운터는 0 ~ Period 값 사이를 업(또는 다운) 카운팅하게 된다. 이 파라미터는 0x0000 ~ 0xFFFF 사이의 값을 가져야 한다.
- ClockDivision : clock division을 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  TIM_CLOCKDIVISION_DIV1
  TIM_CLOCKDIVISION_DIV2
  TIM_CLOCKDIVISION_DIV4
- RepetitionCounter : 반복 카운터(repetition counter) 값을 설정한다. RCR 다운 카운터가 0에 도달하면 업데이트 이벤트가 발생되고 카운트를 RCR 값(N)에서 다시 시작한다. 따라서 각 모드에서 다음과 같이 동작하게 된다.
- PWM 모드 : (N+1)이 PWM 주기가 된다.
- edge-aligned 모드 : (N+1)이 PWM 주기의 반값이 된다.
- center-aligned 모드 : 이 파라미터는 Min_Data = 0x00 ~ Max_Data = 0xFF 사이의 값을 가져야 한다.

참고 이 파라미터는 고급제어 타이머(TIM1, TIM8)에서만 사용 가능하다.

TIM_OC_InitTypeDef

타이머의 출력 비교(OC : Output Compare) 초기설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- uint32_t OCMode
- uint32_t Pulse
- uint32_t OCPolarity
- uint32_t OCNPolarity
- uint32_t OCFastMode
- uint32_t OCIdleState
- uint32_t OCNIdleState

[데이터 형의 설명]

- OCMode : TIM 모드를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  TIM_OCMODE_TIMING
  TIM_OCMODE_ACTIVE
  TIM_OCMODE_INACTIVE
  TIM_OCMODE_TOGGLE
  TIM_OCMODE_PWM1
  TIM_OCMODE_PWM2
  TIM_OCMODE_FORCED_ACTIVE
  TIM_OCMODE_FORCED_INACTIVE
- Pulse : Capture Compare Register에 펄스 값을 저장한다. 이 값은 Min_Data = 0x0000 ~ Max_Data = 0xFFFF의 값을 가진다.
- OCPolarity : output polarity를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  TIM_OCPOLARITY_HIGH
  TIM_OCPOLARITY_LOW
- OCNPolarity : complementary output polarity를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  TIM_OCNPOLARITY_HIGH
  TIM_OCNPOLARITY_LOW

참고 이 파라미터는 고급제어 타이머(TIM1, TIM8)에서만 유효하다.

- OCFastMode : Fast 모드 상태를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  TIM_OCFAST_DISABLE
  TIM_OCFAST_ENABLE

참고 이 파라미터는 PWM1 and PWM2 모드에서만 유효하다.

- OCIdleState : Idle 상태에서 TIM Output Compare 핀 상태를 설정

참고 이 파라미터는 고급제어 타이머(TIM1, TIM8)에서만 유효하다.

- OCNIdleState : Idle 상태에서 TIM Output Compare 핀 상태를 설정한다.

참고 이 파라미터는 고급제어 타이머(TIM1, TIM8)에서만 유효하다.

8.4 타이머 관련 HAL 드라이버

TIM_ClockConfigTypeDef

타이머의 클럭 동작설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- uint32_t ClockSource
- uint32_t ClockPolarity
- uint32_t ClockPrescaler
- uint32_t ClockFilter

[데이터 형의 설명]

- TIM_ClockConfigTypeDef::ClockSource : TIM의 클럭 소스
- TIM_ClockConfigTypeDef::ClockPolarity : TIM의 클럭 극성(polarity)
- TIM_ClockConfigTypeDef::ClockPrescaler : TIM의 클럭 프리스케일러
- TIM_ClockConfigTypeDef::ClockFilter : TIM의 클럭 필터

TIM_IC_InitTypeDef

타이머의 입력 비교(IC : Input Compare) 초기설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- uint32_t ICPolarity
- uint32_t ICSelection
- uint32_t ICPrescaler
- uint32_t ICFilter

[데이터 형의 설명]

- ICPolarity : 입력 신호의 active edge를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_ICPOLARITY_RISING
  - TIM_ICPOLARITY_FALLING
  - TIM_ICPOLARITY_BOTHEDGE
- ICSelection : 입력 설정을 한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_ICSELECTION_DIRECTTI : TIM 입력 1, 2, 3, 4가 각각 IC1, IC2, IC3, IC4에 연결됨
  - TIM_ICSELECTION_INDIRECTTI : TIM 입력 1, 2, 3, 4가 각각 IC1, IC2, IC3, IC4에 연결됨
  - TIM_ICSELECTION_TRC : TIM 입력 1, 2, 3, 4가 TRC에 연결됨
- ICPrescaler : Input Capture Prescaler를 설정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_ICPSC_DIV1 : 캡처 입력(capture input)에 에지가 발생할 때마다 캡처됨
  - TIM_ICPSC_DIV2 : 매 2회의 이벤트 발생시에 1번 캡처됨
  - TIM_ICPSC_DIV4 : 매 4회의 이벤트 발생시에 1번 캡처됨
  - TIM_ICPSC_DIV8 : 매 8회의 이벤트 발생시에 1번 캡처됨
- ICFilter : Input Capture Filter를 설정한다. 이 파라미터는 0x0 ~ 0xF 사이의 값을 가질 수 있다.

TIM_Encoder_InitTypeDef

타이머의 엔코더 동작설정을 위한 구조체이며 stm32f1xx_hal_tim.h에 정의되어 있다.

[데이터 형]

- uint32_t EncoderMode
- uint32_t IC1Polarity
- uint32_t IC1Selection
- uint32_t IC1Prescaler
- uint32_t IC1Filter
- uint32_t IC2Polarity
- uint32_t IC2Selection
- uint32_t IC2Prescaler
- uint32_t IC2Filter

[데이터 형의 설명]

- EncoderMode : Specifies the active edge of the input signal. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_ENCODERMODE_TI1
  - TIM_ENCODERMODE_TI2
  - TIM_ENCODERMODE_TI12

- IC1Polarity : 입력 신호의 active edge를 설정한다. 이 파라미터가 가질 수 있는 값은 TIM_IC_InitTypeDef 구조체의 ICPolarity와 동일하다.
- IC1Selection : 이 파라미터가 가질 수 있는 값은 TIM_IC_Init TypeDef 구조체의 ICSelection과 동일하다.
- IC1Prescaler : 이 파라미터가 가질 수 있는 값은 TIM_IC_InitTypeDef 구조체의 ICPrescaler와 동일하다.
- IC1Filter : 이 파라미터는 0x0 ~ 0xF 사이의 값을 가질 수 있다.
- IC2Polarity : 이 파라미터가 가질 수 있는 값은 TIM_IC_InitTypeDef 구조체의 ICPolarity와 동일하다.
- IC2Selection : 이 파라미터가 가질 수 있는 값은 TIM_IC_ InitTypeDef 구조체의 ICSelection과 동일하다.
- IC2Prescaler : 이 파라미터가 가질 수 있는 값은 TIM_IC_InitTypeDef 구조체의 ICPrescaler와 동일하다.
- IC2Filter : 이 파라미터는 0x0 ~ 0xF 사이의 값을 가질 수 있다.

8.4 타이머 관련 HAL 드라이버

8.4.2 타이머 구동용 HAL 함수

8.4.2.1 타이머 구동용 HAL 함수의 종류

타이머를 구동하기 위한 HAL 함수는 다음과 같은 것들이 있다.

참고

여기에 설명된 함수 중에서 굵은고딕체로 된 것은 이 책의 예제에서 사용하는 함수를 나타낸다.

(1) 타임 베이스 함수(Time Base function)

이 함수는 타이머의 기본적인 기능인 타이머/카운터 동작을 제어하는 함수이다.

- **HAL_TIM_Base_Init(), HAL_TIM_Base_DeInit()**
  - TIM base의 초기화, 초기화 해제
- HAL_TIM_Base_MspInit(), HAL_TIM_Base_MspDeInit()
  - TIM base Msp의 초기화, 초기화 해제
- **HAL_TIM_Base_Start()**
  - Time Base의 동작을 시작한다.
- **HAL_TIM_Base_Stop()**
  - Time Base의 동작을 정지한다.
- **HAL_TIM_Base_Start_IT()**
  - Time Base의 동작을 시작하고 인터럽트 발생을 활성화한다.
- **HAL_TIM_Base_Stop_IT()**
  - Time Base의 동작을 정지하고 인터럽트 발생을 비활성화한다.
- HAL_TIM_Base_Start_DMA()
  - Time Base의 동작을 시작하고 DMA 전송을 활성화한다.
- HAL_TIM_Base_Stop_DMA()
  - Time Base의 동작을 정지하고 DMA 전송을 비활성화한다.

(2) 타임 출력 비교 함수(Time Output Compare function)

이 함수는 타이머의 출력비교 동작을 제어하는 함수이다.

- **HAL_TIM_OC_Init(), HAL_TIM_OC_DeInit()**
  - TIM Output Compare의 초기화, 초기화 해제
- HAL_TIM_OC_MspInit(), HAL_TIM_OC_MspDeInit()
  - TIM Output Compare MSP의 초기화, 초기화 해제
- **HAL_TIM_OC_Start()**
  - TIM Output Compare의 동작을 시작한다.
- HAL_TIM_OC_Stop()
  - TIM Output Compare의 동작을 정지한다.
- **HAL_TIM_OC_Start_IT()**
  - Time Output Compare의 동작을 시작하고 인터럽트 발생을 활성화한다.
- HAL_TIM_OC_Stop_IT()
  - Time Output Compare의 동작을 정지하고 인터럽트 발생을 비활성화한다.
- HAL_TIM_OC_Start_DMA()
  - Time Output Compare의 동작을 시작하고 DMA 전송을 활성화한다.
- HAL_TIM_OC_Stop_DMA()
  - Time Output Compare의 동작을 정지하고 DMA 전송을 비활성화한다.

(3) 타임 PWM 함수(Time PWM function)

이 함수는 타이머의 PWM 동작을 제어하는 함수이다.

- **HAL_TIM_PWM_Init(), HAL_TIM_PWM_DeInit()**
  - TIM PWM의 초기화, 초기화 해제
- HAL_TIM_PWM_MspInit(), HAL_TIM_PWM_MspDeInit()
  - TIM PWM MSP의 초기화, 초기화 해제
- **HAL_TIM_PWM_Start()**
  - TIM PWM을 시작
- HAL_TIM_PWM_Stop()
  - TIM PWM을 중지
- **HAL_TIM_PWM_Start_IT()**
  - Time PWM을 시작하고 interrupt를 활성화 함
- HAL_TIM_PWM_Stop_IT()
  - Time PWM을 중지하고 interrupt를 비활성화 함
- HAL_TIM_PWM_Start_DMA()
  - Time PWM의 동작을 시작하고 DMA 전송을 활성화 함
- HAL_TIM_PWM_Stop_DMA()
  - Time PWM의 동작을 정지하고 DMA 전송을 비활성화 함

(4) 타임 입력 캡쳐 함수(Time Input Capture function)

이 함수는 타이머의 출력비교 동작을 제어하는 함수이다.

- HAL_TIM_IC_Init(), HAL_TIM_IC_DeInit()
  - TIM Input Capture의 초기화/초기화 해제
- HAL_TIM_IC_MspInit(), HAL_TIM_IC_MspDeInit()
  - TIM Input Capture MSP의 초기화, 초기화 해제
- HAL_TIM_IC_Start()
  - Time Input Capture를 시작
- HAL_TIM_IC_Stop()
  - Time Input Capture를 정지
- HAL_TIM_IC_Start_IT()
  - Time Input Capture를 시작하고 인터럽트를 활성화 함
- HAL_TIM_IC_Stop_IT()
  - Time Input Capture를 정지하고 인터럽트를 비활성화 함
- HAL_TIM_IC_Start_DMA()
  - Time Input Capture를 시작하고 DMA 전송을 활성화 함
- HAL_TIM_IC_Stop_DMA()
  - Time Input Capture를 정지하고 DMA 전송을 비활성화 함

(5) 타임 원펄스 함수(Time One Pulse function)

이 함수는 타이머의 원펄스 동작을 제어하는 함수이다.

- HAL_TIM_OnePulse_Init(), HAL_TIM_OnePulse_DeInit()
  - TIM One Pulse의 초기화/초기화 해제
- HAL_TIM_OnePulse_MspInit(), HAL_TIM_OnePulse_MspDeInit()
  - TIM One Pulse의 MSP를 초기화/초기화 해제
- HAL_TIM_OnePulse_Start()
  - Time One Pulse를 시작함.
- HAL_TIM_OnePulse_Stop()
  - Time One Pulse를 정지함.
- HAL_TIM_OnePulse_Start_IT()
  - Time One Pulse를 시작하고 인터럽트를 활성화 함
- HAL_TIM_OnePulse_Stop_IT()
  - Time One Pulse를 정지하고 인터럽트를 비활성화 함

8.4 타이머 관련 HAL 드라이버

- HAL_TIM_OnePulse_Start_DMA()
  - Time One Pulse를 시작하고 DMA 전송을 완성화 함
- HAL_TIM_OnePulse_Stop_DMA()
  - Time One Pulse를 정지하고 DMA 전송을 비활성화 함

(6) 타임 엔코더 함수(Time Encoder function)

이 함수는 타이머의 엔코더 동작을 제어하는 함수이다.

- HAL_TIM_Encoder_Init()/HAL_TIM_Encoder_DeInit()
  - TIM Encoder의 초기화/초기화 해제
- HAL_TIM_Encoder_MspInit()/HAL_TIM_Encoder_MspDeInit()
  - TIM Encoder MSP의 초기화, 초기화 해제
- HAL_TIM_Encoder_Start()
  - Time Encoder를 시작
- HAL_TIM_Encoder_Stop()
  - Time Encoder를 정지
- HAL_TIM_Encoder_Start_IT()
  - Time Encoder를 시작하고 interrupt를 활성화 함
- HAL_TIM_Encoder_Stop_IT()
  - Time Encoder를 정지하고 interrupt를 비활성화 함
- HAL_TIM_Encoder_Start_DMA()
  - Time Encoder를 시작하고 DMA 전송을 활성화 함
- HAL_TIM_Encoder_Stop_DMA()
  - Time Encoder를 정지하고 DMA 전송을 비활성화 함

(7) IRQ 핸들러 관리(IRQ handler management)

이 함수는 타이머의 IRQ 핸들러 동작을 관리(제어)하는 함수이다.

- HAL_TIM_IRQHandler()

(8) 주변장치 제어 함수(Peripheral Control functions)

이 함수는 타이머의 주변장치의 동작을 제어하는 함수이다.

① OC, PWM, IC, One Pulse 모드의 입출력 채널 설정

- HAL_TIM_OC_ConfigChannel()
  - Input Output 채널을 OC 모드로 설정
- HAL_TIM_IC_ConfigChannel()
  - Input Output 채널을 IC 모드로 설정
- HAL_TIM_PWM_ConfigChannel()
  - Input Output 채널을 PWM 모드로 설정
- HAL_TIM_OnePulse_ConfigChannel()
  - Input Output 채널을 One Pulse 모드로 설정

② 관련, Complementary channel, 브레이크 등의 설정

- HAL_TIM_ConfigClockSource()
- HAL_TIM_GenerateEvent()
- HAL_TIM_ConfigOCrefClear()
- HAL_TIM_ConfigTIInput()
- HAL_TIM_ReadCapturedValue()

③ Master와 Slave 동기화 설정

- HAL_TIM_SlaveConfigSynchronization()
- HAL_TIM_SlaveConfigSynchronization_IT()

④ DMA Burst Mode의 설정

- HAL_TIM_DMABurst_WriteStart()
- HAL_TIM_DMABurst_WriteStop()
- HAL_TIM_DMABurst_ReadStart()
- HAL_TIM_DMABurst_ReadStop()

(9) 타이머 콜백 함수(TIM Callbacks functions)

이 함수는 타이머 관련 인터럽트 발생시에 호출되는 콜백 함수이다.

- HAL_TIM_PeriodElapsedCallback()
  - Timer Period elapsed 콜백 함수
- HAL_TIM_OC_DelayElapsedCallback()
  - Timer Output Compare 콜백 함수
- HAL_TIM_IC_CaptureCallback()
  - Timer Input capture 콜백 함수
- HAL_TIM_PWM_PulseFinishedCallback()
  - Timer PWM pulse finished 콜백 함수
- HAL_TIM_TriggerCallback()
  - Timer Trigger 콜백 함수
- HAL_TIM_ErrorCallback()
  - Timer Error 콜백 함수

(10) 주변장치 상태 함수(Peripheral State functions)

이 함수는 실행 중에 해당 주변장치의 상태와 데이터의 흐름 상태를 받아오는 함수들이다.

- HAL_TIM_Base_GetState()
- HAL_TIM_OC_GetState()
- HAL_TIM_PWM_GetState()
- HAL_TIM_IC_GetState()
- HAL_TIM_OnePulse_GetState()
- HAL_TIM_Encoder_GetState()

8.4.2.2 구동용 함수의 사용 방법

TIM 구동용 함수의 기본적인 사용 방법은 다음과 같다.

1) 클럭을 활성화한다.

a) __HAL_RCC_TIMx_CLK_ENABLE() 함수(__TIMx_CLK_ENABLE()도 동일한 함수명)을 이용하여 사용할 TIMx의 클럭을 활성화시킨다.

b) __HAL_RCC_GPIOx_CLK_ENABLE() 함수를 이용하여 TIM 핀으로 사용할 GPIO의 클럭을 활성화한다.

2) TIM로 사용할 GPIO 핀을 대체 기능(Alternate function) 모드로 하고 TIM 핀으로 사용하도록 설정한 다음 HAL_GPIO_Init() 함수를 이용하여 초기화한다.

3) TIM의 사용 용도에 따라 다음의 함수를 이용하여 타이머의 동작조건을 설정한다

- HAL_TIM_Base_Init() : 타이머를 시간 베이스(time base)의 작업에 사용하는 경우에 이 함수를 이용한다. 이때는 TIM_HandleTypeDef 구조체 변수를 사용한다.
- HAL_TIM_OC_Init(), HAL_TIM_OC_ConfigChannel() : 타이머를 출력비교(Output Compare) 신호를 생성하는데 사용하는 경우에 이 함수를 이용한다. 이때는 TIM_OC_InitTypeDef 구조체 변수를 사용한다.
- HAL_TIM_PWM_Init(), HAL_TIM_PWM_ConfigChannel() : 타이머를 PWM 신호를 생성하는데 사용하는 경우
- HAL_TIM_IC_Init(), HAL_TIM_IC_ConfigChannel() : 타이머를 외부 신호를 측정하는데 사용하는 경우
- HAL_TIM_OnePulse_Init(), HAL_TIM_OnePulse_ConfigChannel() : 타이머를 펄스(One Pulse) 모드로 사용하는 경우
- HAL_TIM_Encoder_Init() : 타이머 엔코더 인터페이스를 사용하는 경우

4) TIM의 사용 용도에 따라 다음의 함수를 이용하여 타이머를 동작시킨다.

- Time Base : HAL_TIM_Base_Start(), HAL_TIM_Base_Start_DMA(), HAL_TIM_ Base_Start_IT()
- Input Capture : HAL_TIM_IC_Start(), HAL_TIM_IC_Start_DMA(), HAL_TIM_ IC_Start_ IT()
- Output Compare : HAL_TIM_OC_Start(), HAL_TIM_OC_Start_DMA(), HAL_TIM_ OC_Start_IT()

8.4 타이머 관련 HAL 드라이버

- PWM generation : HAL_TIM_PWM_Start(), HAL_TIM_PWM_Start_DMA(), HAL_ TIM_PWM_Start_IT()
- One-pulse mode output : HAL_TIM_OnePulse_Start(), HAL_TIM_OnePulse_ Start_ IT()
- Encoder mode output : HAL_TIM_Encoder_Start(), HAL_TIM_Encoder_Start_ DMA(), HAL_TIM_Encoder_Start_IT().

5) TIM에서 인터럽트를 발생시키는 경우는 다음과 같이 처리한다.

- HAL_NVIC_SetPriority() 함수를 이용하여 해당 인터럽트의 우선 순위를 설정하고 HAL_EnableIRQ() 함수를 이용하여 인터럽트를 활성화시킨다.
- 해당 인터럽트의 ISR(Interrupt Service Routine)을 작성한다. (stm32f1xx_it.c에서 작성)
  - ISR의 예 : TIM2_IRQHandler() 등
- 해당 인터럽트의 콜백 함수를 작성한다. 인터럽트의 발생시에 처리해야 할 내용을 여기에 작성하면 된다.
  - 콜백 함수의 예 : HAL_TIM_PeriodElapsedCallback(), HAL_TIM_OC_Delay ElapsedCallback() 등

8.4.3 주요 함수의 상세 설명

8.4.3.1 타이머 구동용 함수

타이머 관련 함수는 그 종류가 매우 많으므로 여기서는 중요한 함수 위주로 설명한다. 나머지 함수에 대한 자세한 설명은 ST사의 "STM32F1 (또는 STM32F4) 시리즈 Reference manual"의 타이머 부분을 참고하기 바란다.

(1) 타임 베이스 함수(Time Base function)

HAL_TIM_Base_Init (TIM_HandleTypeDef * htim)

TIM의 카운터 부분(Time base Unit)을 TIM_HandleTypeDef 구조체의 설정값에 맞추어 초기화하고 TIM Base handle을 생성한다.

[파라미터]
- htim : TIM Base handle

[반환값] HAL status

HAL_TIM_Base_DeInit (TIM_HandleTypeDef * htim)

TIM의 카운터 부분(Time base Unit)을 초기화 해제한다.

[파라미터]
- htim : TIM Base handle

[반환값] HAL status

HAL_TIM_Base_Start (TIM_HandleTypeDef * htim)

TIM Base의 동작을 시작한다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

HAL_TIM_Base_Stop (TIM_HandleTypeDef * htim)

TIM Base의 동작을 정지한다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

HAL_TIM_Base_Start_IT (TIM_HandleTypeDef * htim)

TIM Base의 동작을 인터럽트 모드로 시작한다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

HAL_TIM_Base_Stop_IT (TIM_HandleTypeDef * htim)

TIM Base의 인터럽트 모드 동작을 끝낸다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

(2) 타임 출력 비교 함수(Time Output Compare function)

HAL_TIM_OC_Init (TIM_HandleTypeDef * htim)

TIM의 Output Compare를 TIM_HandleTypeDef 구조체의 설정값에 맞추어 초기화하고 TIM Output Compare handle을 생성한다.

[파라미터]
- htim : TIM Output Compare handle

[반환값] HAL status

HAL_TIM_OC_DeInit (TIM_HandleTypeDef * htim)

TIM의 Output Compare의 초기화를 해제한다.

[파라미터]
- htim : TIM Output Compare handle

[반환값] HAL status

HAL_TIM_OC_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

TIM Output Compare의 signal generation을 시작한다.

[파라미터]
- htim : TIM Output Compare handle
- Channel : 활성화 할 TIM 채널을 지정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_OC_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

TIM Output Compare의 동작을 정지한다.

[파라미터]
- htim : TIM Output Compare handle
- Channel : 비활성화 할 TIM 채널을 지정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_OC_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

TIM Output Compare의 signal generation을 인터럽트 모드로 시작한다.

[파라미터]
- htim : TIM Output Compare handle
- Channel : 활성화 할 TIM 채널을 지정한다. 이 파라미터가 가질 수 있는 값은 다음과 같다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_OC_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

TIM Output Compare의 인터럽트 모드 동작을 정지한다.

[파라미터]
- htim : TIM Output Compare handle
- Channel : 정지할 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

8.4 타이머 관련 HAL 드라이버

(3) 타임 PWM 함수(Time PWM function)

HAL_TIM_PWM_Init (TIM_HandleTypeDef * htim)

TIM의 PWM Time Base를 TIM_HandleTypeDef 구조체의 설정값에 맞추어 초기화하고 대응되는 handle을 생성한다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

HAL_TIM_PWM_DeInit (TIM_HandleTypeDef * htim)

TIM의 PWM Time Base의 초기화를 해제한다.

[파라미터]
- htim : TIM handle

[반환값] HAL status

HAL_TIM_PWM_Start (TIM_HandleTypeDef * htim, uint32_t Channel)

PWM signal generation을 시작한다.

[파라미터]
- htim : TIM handle
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_PWM_Stop (TIM_HandleTypeDef * htim, uint32_t Channel)

PWM signal generation을 정지한다.

[파라미터]
- htim : TIM handle
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_PWM_Start_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

PWM signal generation을 인터럽트 모드로 시작한다.

[파라미터]
- htim : TIM handle
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_PWM_Stop_IT (TIM_HandleTypeDef * htim, uint32_t Channel)

PWM signal generation의 인터럽트 모드 동작을 정지한다.

[파라미터]
- htim : TIM handle
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

(4) IRQ 핸들러 함수(IRQ handler management)

HAL_TIM_IRQHandler (TIM_HandleTypeDef * htim)

TIM interrupt의 핸들러 함수

[파라미터]
- htim : TIM handle

[반환값] 없음

(5) 주변장치 설정용 함수(Peripheral Control functions)

HAL_TIM_OC_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

TIM Output Compare Channel을 TIM_OC_InitTypeDef 구조체의 설정값에 맞추어 초기화한다.

[파라미터]
- htim : TIM handle
- sConfig : TIM Output Compare configuration 구조체
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

HAL_TIM_PWM_ConfigChannel (TIM_HandleTypeDef * htim, TIM_OC_InitTypeDef * sConfig, uint32_t Channel)

TIM PWM Channel을 TIM_OC_InitTypeDef 구조체의 설정값에 맞추어 초기화 한다.

[파라미터]
- htim : TIM handle
- sConfig : TIM PWM configuration 구조체
- Channel : 인에블시킬 채널. 이 파라미터는 다음의 값을 가질 수 있다.
  - TIM_CHANNEL_1
  - TIM_CHANNEL_2
  - TIM_CHANNEL_3
  - TIM_CHANNEL_4

[반환값] HAL status

(6) 타이머 콜백 함수(TIM Callbacks functions)

HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef * htim)

블로킹되지 않은 모드(non blocking mode)에서 Period elapsed callback 함수이다.

[파라미터]
- htim : TIM handle

[반환값] None

HAL_TIM_OC_DelayElapsedCallback (TIM_HandleTypeDef * htim)

블로킹되지 않은 모드(non blocking mode)에서 Output Compare callback 함수이다.

[파라미터]
- htim : TIM OC handle

[반환값] None

HAL_TIM_IC_CaptureCallback (TIM_HandleTypeDef * htim)

블로킹되지 않은 모드(non blocking mode)에서 Input Capture callback 함수이다.

[파라미터]
- htim : TIM IC handle

[반환값] None

8.4 타이머 관련 HAL 드라이버

HAL_TIM_PWM_PulseFinishedCallback (TIM_HandleTypeDef * htim)

블로킹되지 않은 모드(non blocking mode)에서 PWM Pulse finished callback 함수이다.

[파라미터]
- htim : TIM handle

[반환값] None

HAL_TIM_TriggerCallback (TIM_HandleTypeDef * htim)

블로킹되지 않은 모드(non blocking mode)에서 Hall Trigger detection callback 함수이다.

[파라미터]
- htim : TIM handle

[반환값] None

HAL_TIM_ErrorCallback (TIM_HandleTypeDef * htim)

TIM Base의 상태값을 반환하는 함수이다.

[파라미터]
- htim : TIM Base handle

[반환값] HAL state

7) 주변장치 상태 함수(Peripheral State functions)

HAL_TIM_Base_GetState (TIM_HandleTypeDef * htim)

TIM Base의 상태값을 반환하는 함수이다.

[파라미터]
- htim : TIM Base handle

[반환값] HAL state

HAL_TIM_OC_GetState (TIM_HandleTypeDef * htim)

TIM OC의 상태값을 반환하는 함수

[파라미터]
- htim : TIM Ouput Compare handle

[반환값] HAL state

HAL_TIM_PWM_GetState (TIM_HandleTypeDef * htim)

TIM PWM의 상태값을 반환하는 함수

[파라미터]
- htim : TIM handle

[반환값] HAL state

HAL_TIM_IC_GetState (TIM_HandleTypeDef * htim)

TIM Input Capture의 상태값을 반환하는 함수

[파라미터]
- htim : TIM IC handle

[반환값] HAL state

HAL_TIM_OnePulse_GetState (TIM_HandleTypeDef * htim)

TIM One Pulse Mode의 상태값을 반환하는 함수

[파라미터]
- htim : TIM OPM handle

[반환값] HAL state

HAL_TIM_Encoder_GetState (TIM_HandleTypeDef * htim)

TIM Encoder Mode의 상태값을 반환하는 함수

[파라미터]
- htim : TIM Encoder handle

[반환값] HAL state

8.5 타이머 응용 예제

TIM 예제 1: 업 카운터(Up Counter)를 이용하여 LED를 1초 간격으로 ON/OFF

- TIM2를 업 카운팅 모드로 동작시켜 1초마다 업데이트 인터럽트(UI)를 발생시킴
- 이를 이용하여 1초마다 LED1을 토글함
- TIM2가 1초마다 업데이트 인터럽트를 발생시키려면 동작 주파수를 1Hz로 설정

참고

타이머의 동작 주파수 계산 방법 : Nucleo-F103 확장보드의 경우
Prescaler = 6399, Period = 9999로 설정한 경우를 예를 들자.
1) 메인 클럭의 공급 주파수 : 64MHz = 64,000,000Hz 이다.

2) Presclaer는 이것을 1 / (6399+1) = 1 / 6,400 으로 다운 시킨다. 따라서 TIM에 공급되는 주파수는 10,000Hz 이다.
3) 따라서 타이머 내의 카운터(Counter)는 1초에 10,000번을 카운트한다. 그러므로 카운터가 0-9,999까지 카운팅을 하는데 걸리는 시간은 1초이다.
4) 그러면 타이머의 동작 주파수는 1Hz가 되고, 1초마다 1번씩 업데이트 인터럽트가 발생하게 된다.
위의 과정을 식으로 나타내면 다음과 같다.

타이머 동작 주파수 = (메인 클럭 / (TIM_Prescaler+1)) / (TIM_Period+1)
= (64,000,000 / (6399+1)) / (9999+1)
= 1Hz

8.5 타이머 응용 예제

TIM 예제 1: 업 카운터(Up Counter)를 이용하여 LED를 1초 간격으로 ON/OFF

[main.c]

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

// ------------------------------------------------------------------------ //

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    LED_Config();
    // -- <1> Timer의 초기설정용 함수를 호출
    TIM2_Config(6399,9999);    // Nucleo-F103 확장보드의 경우
    // TIM2_Config(8999,9999); // Nucleo-F429 확장보드의 경우

    LED_OnOff(GPIO_PIN_LedAll, 500);

    // -- <2> 무한 루프로 계속 동작
    while (1) { }
}

// ------------------------------------------------------------------------ //
// -- <3> TIM 인터럽트 Callback 함수 : Period Elapsed Callback in non blocking mode.
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // -- <3-1> TIM 인터럽트가 발생하면 LED를 모두 Toggle 시킨다.
    HAL_GPIO_TogglePin(GPIONucleo, GPIO_PIN_LedAll);
    HAL_GPIO_TogglePin(GPIOExt, GPIO_PIN_LedAll);
}
```

[main.c]

<1> TIM2_Config(int a, int b) 함수 : TIM2의 초기설정을 위한 함수이다. 이 함수는 각각 <Nucleo_F103.c> / <Nucleo_F429.c>에 정의되어 있다. [14.3절. 예제 소스코드 추가 설명 : Nucleo_F103.c 및 Nucleo_F429.c]을 참고하기 바란다.

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

인터럽트가 1초에 1번씩 발생하도록 주파수를 1 Hz로 설정하기 위해 다음과 같이 설정한다. 첫 번째 인자인 a에는 TIM_Prescaler 값을 넣고 두 번째 인자인 b에는 TIM_Period의 값을 넣는다.

[Nucleo-F103 확장보드의 경우] TIM2_Config(6399,9999);
TIM_Prescaler = 6399;
TIM_Period = 9999;

[Nucleo-F429 확장보드의 경우] TIM2_Config(8999,9999);
TIM_Prescaler = 8999;
TIM_Period = 9999;

<2> while(1) { } 부분은 항상 참(True)이다. 그러므로 이 부분은 아무런 동작이 없는 무한 루프로 동작한다. LED를 On/Off 하는 작업은 TIM 인터럽트 Callback 함수에서 하게 된다.

<3> HAL_TIM_PeriodElapsedCallback( ) 함수 : 타이머의 업데이트 인터럽트(TIM_Base 인터럽트)가 발생하면 호출되는 Callback 함수이다. 따라서 타이머의 업데이트 인터럽트 발생 시에 처리하고 싶은 내용을 이 함수 내에 코딩하면 된다. 그러면 타이머의 ISR 함수인 HAL_TIM_IRQHandler( ) 함수가 이 함수를 호출하게 된다.

<3-1> TIM 인터럽트가 발생하면 LED를 모두 Toggle 시킨다.

8.5 타이머 응용 예제

TIM 예제 1: 업 카운터(Up Counter)를 이용하여 LED를 1초 간격으로 ON/OFF

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------- //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------- //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}


void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------- //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------- //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.5 타이머 응용 예제

TIM 예제 2: 시간 지연 함수 HAL_Delay()를 이용하여 LED를 1초 간격으로 ON/OFF

- 타이머를 사용하지 않고 HAL에서 제공하는 시간 지연 함수인 HAL_Delay()를 이용하여 1초마다 LED를 토글함

참고

이 예제에서는 타이머가 아니라 HAL_Delay() 함수를 이용하여 1초의 시간 간격을 구현하였다. 이 방법을 이용하면 프로그램은 간단해지지만 시간 지연이 일어나는 동안(즉, HAL_Delay() 함수가 동작하는 동안) MCU는 단지 시간 지연을 기다릴 뿐, 다른 어떤 작업도 하지 않는다. 따라서 MCU가 여러 개의 작업을 서로 정해진 시간 간격으로 수행하려면 타이머의 인터럽트를 이용하여야 한다.

8.5 타이머 응용 예제

TIM 예제 2: 시간 지연 함수 HAL_Delay()를 이용하여 LED를 1초 간격으로 ON/OFF

[main.c]

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

// ------------------------------------------------------------------------------ //

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

    LED_OnOff(GPIO_PIN_LedAll, 500);

    // 무한 루프로 계속 동작
    while (1)  {
        // -- <1> HAL_Delay() 함수를 이용하여 1000msec 시간 지연을 함
        HAL_Delay(1000);

        // -- <2>  LED를 모두 Toggle 시킨다.
        HAL_GPIO_TogglePin(GPIONucleo, GPIO_PIN_LedAll);
        HAL_GPIO_TogglePin(GPIOExt, GPIO_PIN_LedAll);
    }
}
```

8.5 타이머 응용 예제

TIM 예제 3: 업 카운터와 OC(출력 비교) 모드를 이용한 LED의 ON/OFF

- TIM2를 업 카운터와 OC(출력 비교) 모드로 동작시켜 LED를 On/Off 시키는 예제
- 업 카운터의 인터럽트를 이용하여 LED를 ON, OC 의 인터럽트를 이용하여 LED를 OFF

a) LED의 ON
- TimHandler 구조체 변수의 TimHandler.Init.Prescaler = 6399 (Nucleo-F429 확장보드는 8999), TimHandler.Init.Period = 9999 로 설정하고 업 카운터 모드로 동작을 한다.
- 그러면 카운터는 0~9999 사이를 업 카운팅을 하게되고, 1초 마다 UI(업데이트 인터럽트)가 발생한다.
- UI가 발생하면 UI 콜백함수인 HAL_TIM_PeriodElapsedCallback()가 호출되며, 이 함수에서 LED를 On 시켜준다.

b) LED 의 OFF
- OC(출력비교) 모드에서 TIM_OCInit 구조체 변수의 "TIM_OCInit.Pulse의 설정값 = 업 카운터의 카운팅 값"이 될때 출력 비교 인터럽트(CC2I)가 발생한다.
- 그러면 OC 인터럽트 콜백함수인 HAL_TIM_OC_DelayElapsedCallback()가 호출되며, 이 함수에서 LED를 Off 시켜준다.

8.5 타이머 응용 예제

TIM 예제 3: 업 카운터와 OC(출력 비교) 모드를 이용한 LED의 ON/OFF

[main.c]

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

// ------------------------------------------------------------------------- //

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    LED_Config();
    SwEXTI_Config();          // 이 함수는 이 예제에서 사용되지 않는다.

    // -- <1> Timer의 초기설정용 함수를 호출
    TIM2_Config(6399,9999);     // Nucleo-F103 확장보드의 경우
    // TIM2_Config(8999,9999); // Nucleo-F429 확장보드의 경우

    // -- <2> Timer의 Output Compare 초기설정용 함수를 호출
    TIM_OC_Config(999);

    // -- <3> 무한 루프로 계속 동작
    while (1)  { }
}
```

[main.c]

<1> TIM2_Config(int a, int b) 함수 : TIM 예제 1과 동일하다.

<2> TIM_OC_Config( ) 함수 : TIM의 Output Compare 동작조건을 설정한다. 여기서는 TIM의 카운터 값 = 999 일 때 OC 인터럽트를 발생시켜 LED을 OFF하기 위해 다음과 같이 설정한다. 여기서 TIM_OCInit.Pulse의 설정 값(현재는 999)을 변경하면 LED가 OFF되는 시간이 달라진다.

<3> while(1) { } 부분은 항상 참(True)이다. 그러므로 이 부분은 아무런 동작이 없는 무한 루프로 동작한다. LED를 On/Off 하는 작업은 TIM 인터럽트 Callback 함수에서 하게 된다.

8.5 타이머 응용 예제

TIM 예제 3: 업 카운터와 OC(출력 비교) 모드를 이용한 LED의 ON/OFF

```
// ------------------------------------------------------------------------------ //
// -- <4> Timer의 TIM_Base 인터럽트 Callback 함수
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // -- <4-1> 인터럽트가 발생하면 LED를 모두 ON 시킨다.
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_SET );
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_SET );
}

// ------------------------------------------------------------------------------ //
// -- <5>  Timer의 OC 인터럽트 Callback 함수
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
{
    // -- <5-1> 인터럽트가 발생하면 LED를 모두 OFF 시킨다.
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_RESET );
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_RESET );
}
```

<4> HAL_TIM_PeriodElapsedCallback() : Timer의 업데이트 인터럽트(TIM_Base 인터럽트)가 발생하면 호출되는 Callback 함수이다.

<4-1> 이 함수 내에서는 인터럽트가 발생하면 LED를 모두 ON 시킨다.

<5> HAL_TIM_OC_DelayElapsedCallback() : Timer의 OC 인터럽트가 발생하면 호출되는 Callback 함수이다.

<5-1> 이 함수 내에서는 인터럽트가 발생하면 LED를 모두 OFF 시킨다.

8.5 타이머 응용 예제

TIM 예제 3: 업 카운터와 OC(출력 비교) 모드를 이용한 LED의 ON/OFF

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------------ //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------------ //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}

// ------------------------------------------------------------------------------ //

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------------ //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------------ //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

<3> TIM3_IRQHandler( ) 함수 : TIM3에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다. 이 함수는 이번 예제에서는 사용하지 않으며, TIM3를 이용하는 추후의 예제를 위한 것이다.

<3-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.5 타이머 응용 예제

TIM 예제 4: 업 카운터와 OC 모드(출력 펄스 폭 변경)

- TIM2를 업 카운터와 OC(출력 비교)모드로 동작시켜 LED1 ~ LED8을 On/Off 시킴
- 프로그램 실행 중에 SW를 누르면 이에 대응하여 OC 값이 변함
- OC 값이 변화함에 따라 LED가 Off되는 시간이 변함
- 작동개요

  (1) 프로그램 실행 시: TIM_OCInit.Pulse = 9999 (LED가 가장 밝음)
  (2) SW1을 누를 시: TIM_OCInit.Pulse = 999 (LED가 가장 어두움)
  (3) SW2를 누를 시: TIM_OCInit.Pulse = 2999 (LED가 조금 어두움)
  (4) SW3을 누를 시: TIM_OCInit.Pulse = 4999 (LED가 조금 밝아짐)
  (5) SW4를 누를 시: TIM_OCInit.Pulse = 9999 (LED가 가장 밝아짐)

8.5 타이머 응용 예제

TIM 예제 4: 업 카운터와 OC 모드(출력 펄스 폭 변경)

[main . c]

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

// TimHandler, TIM_OCInit 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
extern TIM_OC_InitTypeDef   TIM_OCInit;

// ------------------------------------------------------------------------------ //
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    LED_Config();
    // -- <1> SW의 EXTI 입력 발생을 위한 초기설정 함수
    SwEXTI_Config();
   // -- <2> Timer의 초기설정용 함수를 호출
    TIM2_Config(0,9999);
    // -- <3> Timer Output Compare의 초기설정용 함수를 호출
    TIM_OC_Config(9999);

    LED_OnOff(GPIO_PIN_LedAll, 500);

    // 무한 루프로 계속 동작
    while (1)  { }
}
```

```
// ------------------------------------------------------------------------------ //
// -- <4> Timer의 TIM_Base 인터럽트 Callback 함수

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    // 인터럽트가 발생하면 LED를 모두 ON 시킨다.
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_SET );
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_SET );
}

// ------------------------------------------------------------------------------ //

// -- <5>  Timer의 OC 인터럽트 Callback 함수
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
{

    // 인터럽트가 발생하면 LED를 모두 OFF 시킨다.
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_RESET );
```

[main . c]

<1> SwEXTI_Config() 함수 : SW의 EXTI를 사용하여야 하므로 이 함수를 호출한다.
<2> TIM2_Config(0 , 9999) 함수 : 최고의 타이머 주파수를 얻기 위해 첫 번째 인자인 a에는 TIM_Prescaler 값을 0으로 넣는다.
<3> TIM_OC_Config() 함수 : TIM의 Output Compare 동작조건을 설정한다. 여기서는 TIM의 카운터 값 = 9999 일 때 OC 인터럽트를 발생시켜 LED를 OFF하도록 설정한다.
<4> HAL_TIM_PeriodElapsedCallback() 함수 : Timer의 TIM_Base 인터럽트가 발생하면 호출되는 Callback 함수이다.
<5> HAL_TIM_OC_DelayElapsedCallback() 함수 : Timer의 OC 인터럽트가 발생하면 호출되는 Callback 함수이다.

8.5 타이머 응용 예제

TIM 예제 4: 업 카운터와 OC 모드(출력 펄스 폭 변경)

```
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_RESET );
}

// ------------------------------------------------------------------------------ //
// -- <6> EXTI 인터럽트 Callback 함수의 구현
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN)
{
    // -- <6-1> SW1 ~ SW4가 눌러지면 TIM_OCInit.Pulse 값을 변경
    if ( GPIO_PIN == GPIO_PIN_Nucleo_Sw)        TIM_OCInit.Pulse =  999;
    if ( GPIO_PIN == GPIO_PIN_Sw1)              TIM_OCInit.Pulse =  999;
    if ( GPIO_PIN == GPIO_PIN_Sw2)              TIM_OCInit.Pulse = 2999;
    if ( GPIO_PIN == GPIO_PIN_Sw3)              TIM_OCInit.Pulse = 4999;
    if ( GPIO_PIN == GPIO_PIN_Sw4)              TIM_OCInit.Pulse = 9999;

    // -- <6-2> TIM OC의 Channel을 TIM_OCInit에 설정된 값으로 초기화함
    HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_1);
    // -- <6-3> TIM OC를 동작함
    HAL_TIM_OC_Start_IT(&TimHandle,TIM_CHANNEL_1);
}
```

<6> HAL_GPIO_EXTI_Callback() 함수 : SW가 눌러져서 EXTI가 발생하면 호출되는 Callback 함수이다.
<6-1> 프로그램 실행 중에 SW를 누르면 이에 대응하여 TIM_OCInit.Pulse 값을 다음과 같이 변경하여 준다.
SW1이 눌러지면 : TIM_OCInit.Pulse = 999로 변경 (LED가 가장 어두워진다)
SW2이 눌러지면 : TIM_OCInit.Pulse = 2999로 변경 (LED가 조금 어두워진다)
SW3이 눌러지면 : TIM_OCInit.Pulse = 4999로 변경 (LED가 조금 밝아진다)
SW4이 눌러지면 : TIM_OCInit.Pulse = 9999로 변경 (LED가 가장 밝아진다)
<6-2> HAL_TIM_OC_ConfigChanne() 함수 : 호출을 하여 변경된 pulse 값을 적용시킨다.
<6-3> HAL_TIM_OC_Start_IT() 함수 : TIM OC를 동작시킨다. 그러면 눌러진 SW에 대응하여 LED의 밝기가 변화한다.

8.5 타이머 응용 예제

TIM 예제 4: 업 카운터와 OC 모드(출력 펄스 폭 변경)

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------------ //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------------ //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}

// ------------------------------------------------------------------------------ //

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------------ //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------------ //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

<3> TIM3_IRQHandler( ) 함수 : TIM3에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다. 이 함수는 이번 예제에서는 사용하지 않으며, TIM3를 이용하는 추후의 예제를 위한 것이다.

<3-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.5 타이머 응용 예제

TIM 예제 5: 업 카운터와 OC 모드(2채널 사용, 출력 펄스 폭 변경)

- TIM2를 업 카운터와 OC모드로 동작시켜 SW의 입력에 따라 출력 펄스 폭을 조절함

- 예제 4와의 차이점

  a) 업 카운터를 이용하여 LED1~2와 LED7~8를 On 시킴
  b) OC 모드에서 타이머의 채널을 2개를 사용
  - Channel_1의 출력을 이용하여 LED1~2를 Off
  - Channel_2의 출력을 이용하여 LED7~8를 Off
  c) 입력용 SW를 누르면 LED의 밝기가 변경됨
  - SW1을 누르면: LED 1,2와 LED 7,8이 최대 밝기로 됨 (TIM_OCInit.Pulse=9999)
  - SW2를 누르면: LED 1,2가 어두워짐 (TIM_OCInit.Pulse=2500)
  - SW3를 누르면: LED 7,8이 어두워짐 (TIM_OCInit.Pulse=5000)
  - SW4를 누르면: LED 7,8이 더 어두워짐 (TIM_OCInit.Pulse=1200)

8.5 타이머 응용 예제

TIM 예제 5: 업 카운터와 OC 모드(2채널 사용, 출력 펄스 폭 변경)

[main . c]

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

// TimHandler, TIM_OCInit 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
extern TIM_OC_InitTypeDef   TIM_OCInit;

// ------------------------------------------------------------------------------ //
int main(void)
{
    HAL_Init();
    SystemClock_Config();
    LED_Config();
    SwEXTI_Config();
    // -- <1> Timer의 초기설정용 함수를 호출
    TIM2_Config(0,9999);
    // -- <2> Timer Output Compare의 초기설정용 함수를 호출
    TIM_OC_Config(9999);
    LED_OnOff(GPIO_PIN_LedAll, 500);

    // 무한 루프로 계속 동작
    while (1)  { }
}
```

```
// ------------------------------------------------------------------------------ //
// -- <3> Timer의 TIM_Base 인터럽트 Callback 함수
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_Led1 | GPIO_PIN_Led2
                             | GPIO_PIN_Led7 | GPIO_PIN_Led8, GPIO_PIN_SET );
}

// ------------------------------------------------------------------------------ //
// -- <4>  Timer의 OC 인터럽트 Callback 함수
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
{
    // -- <4-1>  CHANNEL_1에서 인터럽트가 발생한 경우
    if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
        HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_Led1 | GPIO_PIN_Led2, GPIO_PIN_RESET);
    }

    // <4-2>  CHANNEL_2에서 인터럽트가 발생한 경우
    if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) {
        HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_Led7 | GPIO_PIN_Led8, GPIO_PIN_RESET);
    }
}
```

[main . c]

<1> TIM2_Config(0 , 9999) 함수 : 최고의 타이머 주파수를 얻기 위해 첫 번째 인자인 a에는 TIM_Prescaler 값을 0으로 넣는다.

<2> TIM_OC_Config() 함수 : TIM의 Output Compare 동작조건을 설정한다. 여기서는 TIM의 카운터 값 = 999 일 때 OC 인터럽트를 발생시켜 LED을 OFF하도록 같이 설정한다.

<3> HAL_TIM_PeriodElapsedCallback() : Timer의 TIM_Base 인터럽트가 발생하면 호출되는 Callback 함수이다.

<4> HAL_TIM_OC_DelayElapsedCallback() : Timer의 OC 인터럽트가 발생하면 호출되는 Callback 함수이다.

8.5 타이머 응용 예제

TIM 예제 5: 업 카운터와 OC 모드(2채널 사용, 출력 펄스 폭 변경)

```
// ------------------------------------------------------------------------------ //
// -- <5>  EXTI 인터럽트 Callback 함수의 구현
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN)
{

    // -- <5-1> SW1이 눌러지면 channel 1, 2의 TIM_OCInit.Pulse = 9999로 변경
    if ( GPIO_PIN == GPIO_PIN_Sw1 ) {
        TIM_OCInit.Pulse = 9999;
        HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_1);
        HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_2);
    }

    // -- <5-2> SW2가 눌러지면 channel 1의 TIM_OCInit.Pulse = 2500으로 변경
    else if ( GPIO_PIN == GPIO_PIN_Sw2 ) {
        TIM_OCInit.Pulse = 2500;
        HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_1);
    }
    // -- <5-3> SW3이 눌러지면 channel 2의 TIM_OCInit.Pulse = 5000으로 변경
    else if ( GPIO_PIN == GPIO_PIN_Sw3 ) {
        TIM_OCInit.Pulse = 5000;
        HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_2);
    }
    // -- <5-4> SW4가 눌러지면 channel 2의 TIM_OCInit.Pulse = 1200으로 변경
    else if ( GPIO_PIN == GPIO_PIN_Sw4 ) {
        TIM_OCInit.Pulse = 1200;
        HAL_TIM_OC_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_2);
    }
}
```

<5> HAL_GPIO_EXTI_Callbck() : SW가 눌러져서 EXTI가 발생하면 호출되는 Callback 함수이다.

<5-1> ~ <5-4> 프로그램 실행 중에 SW를 누르면 이에 대응하여 TIM_OCInit.Pulse 값을 다음과 같이 변경하여 준다.

① SW1이 눌러지면 : 채널1 과 채널2의 TIM_OCInit.Pulse = 9999로 변경하고
SW2이 눌러지면 : 채널1의 TIM_OCInit.Pulse = 2500로 변경
SW3이 눌러지면 : 채널2의 TIM_OCInit.Pulse = 5000로 변경
SW4이 눌러지면 : 채널2의 TIM_OCInit.Pulse = 1200로 변경

② 그 다음은 HAL_TIM_OC_ConfigChanne() 함수를 호출하여 변경사항을 적용시킨다. 그러면 LED가 Off되는 시간이 변화되며, 이에 따라, LED의 밝기가 변화한다.

8.5 타이머 응용 예제

TIM 예제 5: 업 카운터와 OC 모드(2채널 사용, 출력 펄스 폭 변경)

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------- //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------- //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}

// ------------------------------------------------------------------------- //

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------- //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------- //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

<3> TIM3_IRQHandler( ) 함수 : TIM3에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다. 이 함수는 이번 예제에서는 사용하지 않으며, TIM3를 이용하는 추후의 예제를 위한 것이다.

<3-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.5 타이머 응용 예제

TIM 예제 6: PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용)

- TIM3의 Channel 1에서 듀티 비(duty ratio)가 변화하는 PWM 출력을 발생시킴
- GPIO 핀 PA6를 이용하여 LED의 밝기를 제어
- 예제 4와의 차이점: LED On/Off의 신호가 PWM핀(PA6)으로 직접 출력됨

  (1) SW1을 누를 시: TIM_OCInit.Pulse = 999 (LED가 가장 밝아짐)
  (2) SW2를 누를 시: TIM_OCInit.Pulse = 2999 (LED가 조금 밝아짐)
  (3) SW3을 누를 시: TIM_OCInit.Pulse = 4999 (LED가 조금 어두움)
  (4) SW4를 누를 시: TIM_OCInit.Pulse = 9999 (LED가 가장 어두움)

8.5 타이머 응용 예제

TIM 예제 6: PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용)

[main. c]

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

// -- <1> 주변장치 초기화용 구조체
GPIO_InitTypeDef            GPIO_Init;
extern TIM_HandleTypeDef    TimHandle;
extern TIM_OC_InitTypeDef   TIM_OCInit;

// ------------------------------------------------------------------------- //

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

    // -- <2> GPIOA의 클럭을 enable
    __HAL_RCC_GPIOA_CLK_ENABLE();
      // -- <3>  TIM3의 channel 1 핀(PA6)의 동작 조건을 설정
    GPIO_Init.Pin   = GPIO_PIN_6;                   // GPIO에서 사용할 PIN 설정
    GPIO_Init.Mode  = GPIO_MODE_AF_PP;              // Output Push-Pull 모드
    GPIO_Init.Pull  = GPIO_PULLUP;                  // Pull Up 모드
    GPIO_Init.Speed = GPIO_SPEED_FREQ_HIGH;         // 동작속도를 HIGH로

    // GPIO_Init.Alternate = GPIO_AF2_TIM3;     // Nucleo-F429 확장보드 구동 시 주석을 풀고 사용
```

[main. c]

<1> GPIO의 초기화를 위하여 구조체 GPIO_InitTypeDef 형의 변수 GPIO_Init를 선언한다.
<2> 추가 LED용 GPIO의 클럭을 enable한다.
<3> 추가 LED(TIM3의 channel 1 핀 : PA6)의 동작 조건을 설정하기 위해 구조체 변수 GPIO_Init의 멤버인 GPIO_Init.Pin, GPIO_Init.Mode, GPIO_Init.Pull, GPIO_Init.Speed 의 값을 설정한다. LED는 동작 조건이 Output Push-Pull 모드, Pull Up 사용, 동작속도를 HIGH로 설정한다.

```
GPIO_Init_Struct.Pin   = GPIO_PIN_6;                   // GPIO에서 사용할 PIN 설정
GPIO_Init_Struct.Mode  = GPIO_MODE_OUTPUT_PP;          // Output Push-Pull 모드
GPIO_Init_Struct.Pull  = GPIO_PULLUP;                  // Pull Up 사용
GPIO_Init_Struct.Speed = GPIO_SPEED_FREQ_HIGH;         // 동작속도를 HIGH로
```

8.5 타이머 응용 예제

TIM 예제 6: PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용)

```
    // -- <4> GPIOA를 GPIO_Init에 설정된 조건으로 초기화함
    HAL_GPIO_Init(GPIOA, &GPIO_Init);

    LED_Config();
    SwEXTI_Config();
    // -- <5> Timer의 초기설정용 함수를 호출
    TIM3_Config(0,9999);
    // -- <6> Timer PWM의 초기설정용 함수를 호출
    TIM_PWM_Poll_Config(9999);

    LED_OnOff(GPIO_PIN_LedAll, 500);

    while (1)  { }
}

// ------------------------------------------------------------------------------ //
// -- <7> EXTI 인터럽트로 TIM PWM 설정 조건을 변경
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_PIN)
{
    // -- <7-1> SW1 ~ SW4가 눌러지면 TIM_OCInit.Pulse 값을 변경
    if ( GPIO_PIN == GPIO_PIN_Sw1)    TIM_OCInit.Pulse= 999;

    if ( GPIO_PIN == GPIO_PIN_Sw2)    TIM_OCInit.Pulse= 2999;

    if ( GPIO_PIN == GPIO_PIN_Sw3)    TIM_OCInit.Pulse= 4999;

    if ( GPIO_PIN == GPIO_PIN_Sw4)    TIM_OCInit.Pulse= 9999;

    // -- <7-2> TIM PWM의 Channel을 TIM_OCInit에 설정된 값으로 초기화 함
    HAL_TIM_PWM_ConfigChannel(&TimHandle, &TIM_OCInit, TIM_CHANNEL_1);

    // -- <7-3> TIM PWM를 동작함
    HAL_TIM_PWM_Start(&TimHandle,TIM_CHANNEL_1);
}
```

<4> HAL_GPIO_Init() : GPIO를 위에서 설정된 동작조건으로 초기화하기 위해 이 함수를 호출한다.
<5> TIM3_Config(0 , 9999) 함수 : Timer3의 초기설정용 함수이다. 최고의 타이머 주파수를 얻기 위해 첫 번째 인자에는 TIM_Prescaler 값을 0으로 넣는다.
<6> TIM_PWM_Poll_Config() : TIM PWM을 폴링(polling)으로 설정한다. TIM PWM의 Channel을 TIM_OCInit에 설정된 값으로 초기화하고 TIM PWM을 동작시키며 설정조건이 되면 PWM을 발생시킨다.
<7> HAL_GPIO_EXTI_Callback() : SW가 눌러져서 EXTI가 발생하면 호출되는 Callback 함수이다.
<7-1> 프로그램 실행 중에 SW를 누르면 이에 대응하여 TIM_OCInit.Pulse 값을 다음과 같이 변경하여 준다.
SW1이 눌러지면 : TIM_OCInit.Pulse = 9999로 변경 (LED가 가장 밝아진다)
SW2이 눌러지면 : TIM_OCInit.Pulse = 4999로 변경 (LED가 조금 밝아진다)
SW3이 눌러지면 : TIM_OCInit.Pulse = 2999로 변경 (LED가 조금 어두워진다)
SW4이 눌러지면 : TIM_OCInit.Pulse = 999로 변경 (LED가 가장 어두워진다)
<7-2> HAL_TIM_PWM_ConfigChannel() 함수 : 함수를 호출하여 변경사항을 적용시킨다.
<7-3> HAL_TIM_PWM_Start() 함수 : TIM PWM을 발생시킨다.

8.5 타이머 응용 예제

TIM 예제 6: PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용)

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------------ //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------------ //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}

// ------------------------------------------------------------------------------ //

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------------ //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------------ //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

<3> TIM3_IRQHandler( ) 함수 : TIM3에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다. 이 함수는 이번 예제에서는 사용하지 않으며, TIM3를 이용하는 추후의 예제를 위한 것이다.

<3-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.5 타이머 응용 예제

TIM 예제 7: PWM 출력 모드를 이용한 LED의 밝기 제어
(PWM 출력핀이 없는 경우 - PWM 인터럽트를 이용하여 GPIO 핀을 On/Off)

- TIM2를 업 카운터와 PWM 모드로 동작시켜 LED를 On/Off 시킴

- TIM의 OC기능을 사용하는 [TIM 예제3]과 전체적인 구성과 유사

  a) LED 1~8의 On: 업 카운터를 이용하여 LED1~LED8를 On시킴
  b) LED 1~8의 Off: PWM모드에서 TIM_OCInit 구조체 변수의 TIM_OCInit.Pulse의 설정 값이 업 카운터의 카운팅 값과 일치할 때, 인터럽트가 발생함. 이때마다 PWM 인터럽트 콜백함수가 호출되며, LED1~LED8을 Off시킴

8.5 타이머 응용 예제

TIM 예제 7: PWM 출력 모드를 이용한 LED의 밝기 제어 (PWM 출력핀이 없는 경우 - PWM 인터럽트를 이용하여 GPIO 핀을 On/Off)

[main.c]

<1> TIM_PWM_IT_Config() : TIM PWM을 인터럽트(Interrupt)로 설정한다. TIM PWM의 Channel을 TIM_OCInit에 설정된 값으로 초기화하고 TIM PWM을 동작시키며 설정조건이 되면 PWM을 발생시킨다.
<2> TIM_BASE 인터럽트 Callback 함수이며 업 카운터를 이용하여 LED1 ~LED8을 ON 시켜준다.
<3> TIM PWM 인터럽트 Callback 함수이며 TIM_OCInit.Pulse의 설정값이 업 카운터의 카운팅값과 일치할 때 PWM 인터럽트가 발생한다. 그러면 이때마다 PWM 인터럽트 콜백함수인 HAL_TIM_PWM_PulseFinishedCallback가 호출되며, 이 함수에서 LED1 ~ LED8를 Off 시켜준다.

[main.c]

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

// TimHandler, TIM_OCInit 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
extern TIM_OC_InitTypeDef   TIM_OCInit;

// ------------------------------------------------------------------------------ //

int main(void)
{

    HAL_Init();
    SystemClock_Config();

    LED_Config();
    SwEXTI_Config();
    TIM2_Config(6399,9999);          // Nucleo-F103 확장보드일 경우
    // TIM2_Config(6399,9999);       // Nucleo-F429 확장보드일 경우
    // -- <1> Timer PWM의 초기설정용 함수를 호출
    TIM_PWM_IT_Config(999);

    LED_OnOff(GPIO_PIN_LedAll, 500);

    while (1) { }
}

// ------------------------------------------------------------------------------ //
// -- <2> TIM 인터럽트 TIM_BASE 인터럽트 Callback 함수
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_SET );
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_SET );

}
// -- <3> TIM 인터럽트 TIM PWM 인터럽트 callback 함수
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
{
    HAL_GPIO_WritePin(GPIONucleo, GPIO_PIN_All, GPIO_PIN_RESET );
    HAL_GPIO_WritePin(GPIOExt, GPIO_PIN_All, GPIO_PIN_RESET );
}
```

8.5 타이머 응용 예제

TIM 예제 7: PWM 출력 모드를 이용한 LED의 밝기 제어
(PWM 출력핀이 없는 경우 - PWM 인터럽트를 이용하여 GPIO 핀을 On/Off)

[stm32f1xx_it.c]

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

// ------------------------------------------------------------------------------ //

void SysTick_Handler(void)
{
    HAL_IncTick();
}

// ------------------------------------------------------------------------------ //

void EXTI15_10_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_4);
}

// ------------------------------------------------------------------------------ //

void EXTI9_5_IRQHandler(void)
{
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_5);
    HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_8);
}

// ------------------------------------------------------------------------------ //
// -- <1> TIM 인터럽트  IRQ handler 함수를 위한 TimHandler 변수를 외부정의 변수로 선언
extern TIM_HandleTypeDef    TimHandle;
// ------------------------------------------------------------------------------ //
// -- <2> TIM 인터럽트 IRQ handler 함수
void TIM2_IRQHandler(void)
{
    // -- <2-1> TIM 인터럽트 Callback 함수
    HAL_TIM_IRQHandler(&TimHandle);
}
```

[stm32f1xx_it.c]

<1> main.c 에서 정의한 TimHandle 변수를 외부정의 변수로 선언하였다. 이 변수는 TIM2_IRQHandler( ) 함수에서 사용한다.

<2> TIM2_IRQHandler( ) 함수 : TIM2에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다.

<2-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

<3> TIM3_IRQHandler( ) 함수 : TIM3에서 인터럽트가 발생하면 이를 처리하는 Handler 함수이다. 이 함수는 이번 예제에서는 사용하지 않으며, TIM3를 이용하는 추후의 예제를 위한 것이다.

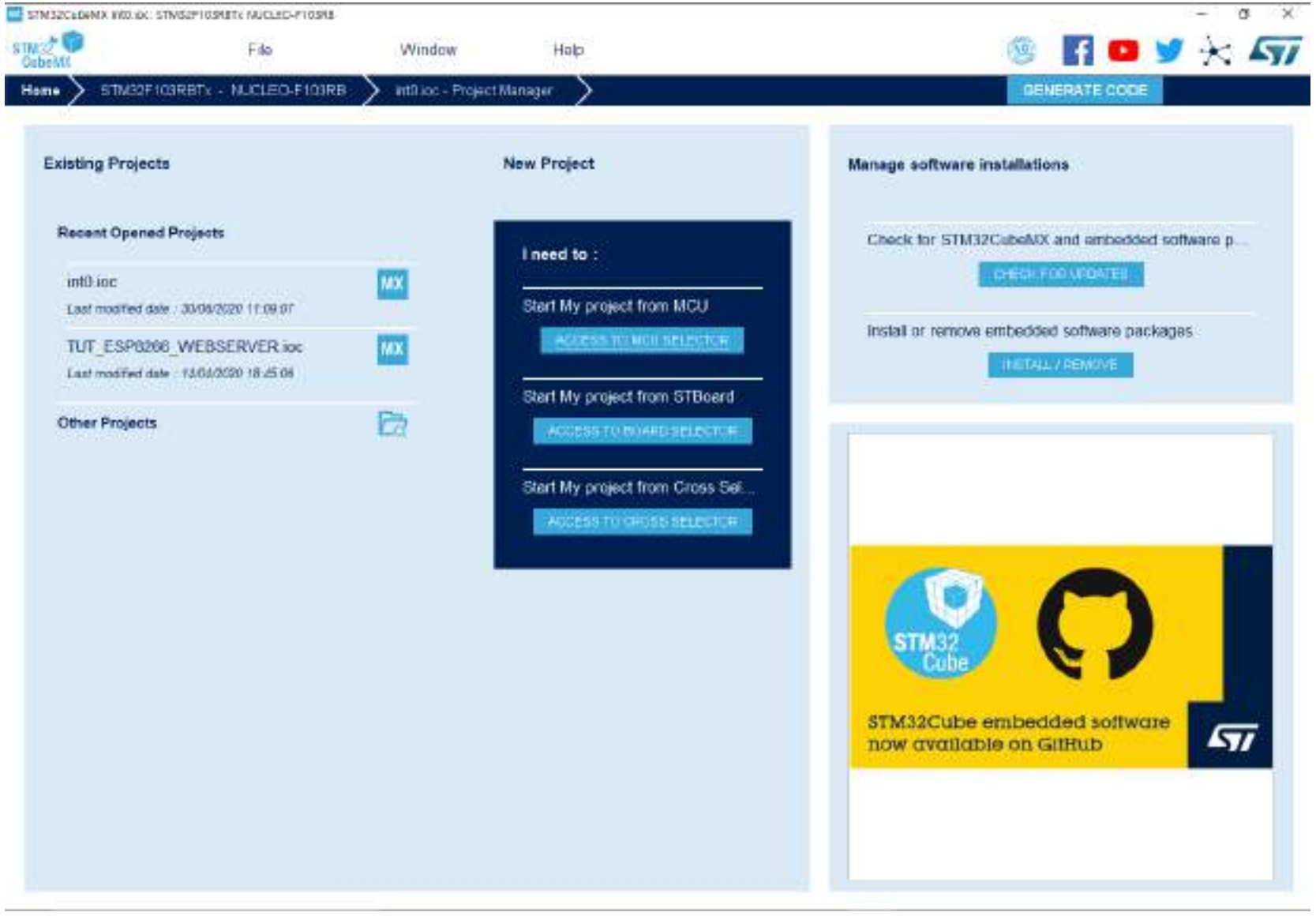
<3-1> HAL_TIM_IRQHandler( ) 함수 : TIM에서 발생한 여러 가지 인터럽트를 처리하는 Handler 함수. 이 함수의 내부에서는 발생된 TIM 인터럽트의 종류에 맞는 콜백함수를 다시 호출해준다.

8.6 타이머 CubeMX과제

CubeMX로 예제 8.1 구현하기

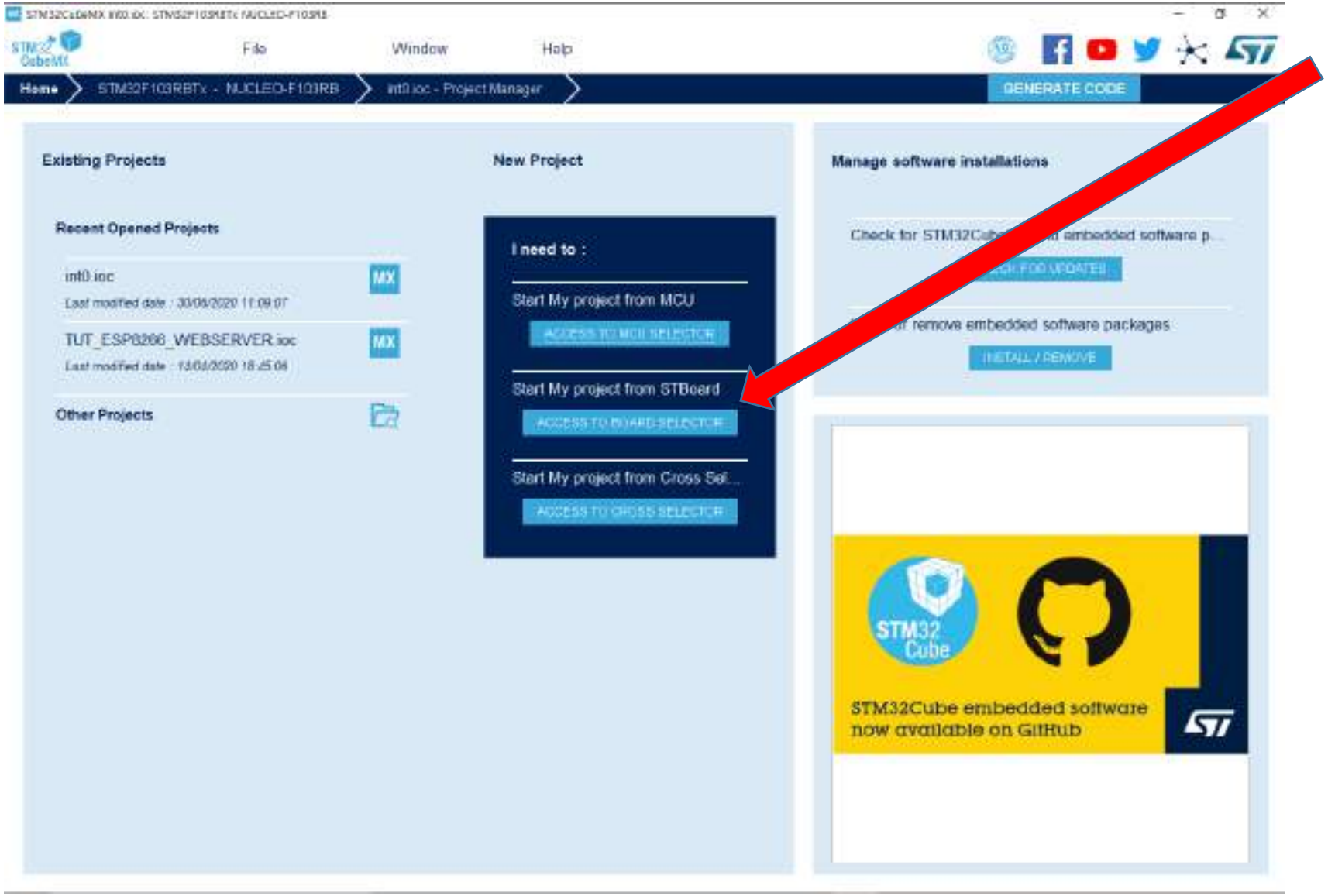
CubeMX 실행

초기화면



8.6 타이머 CubeMX과제

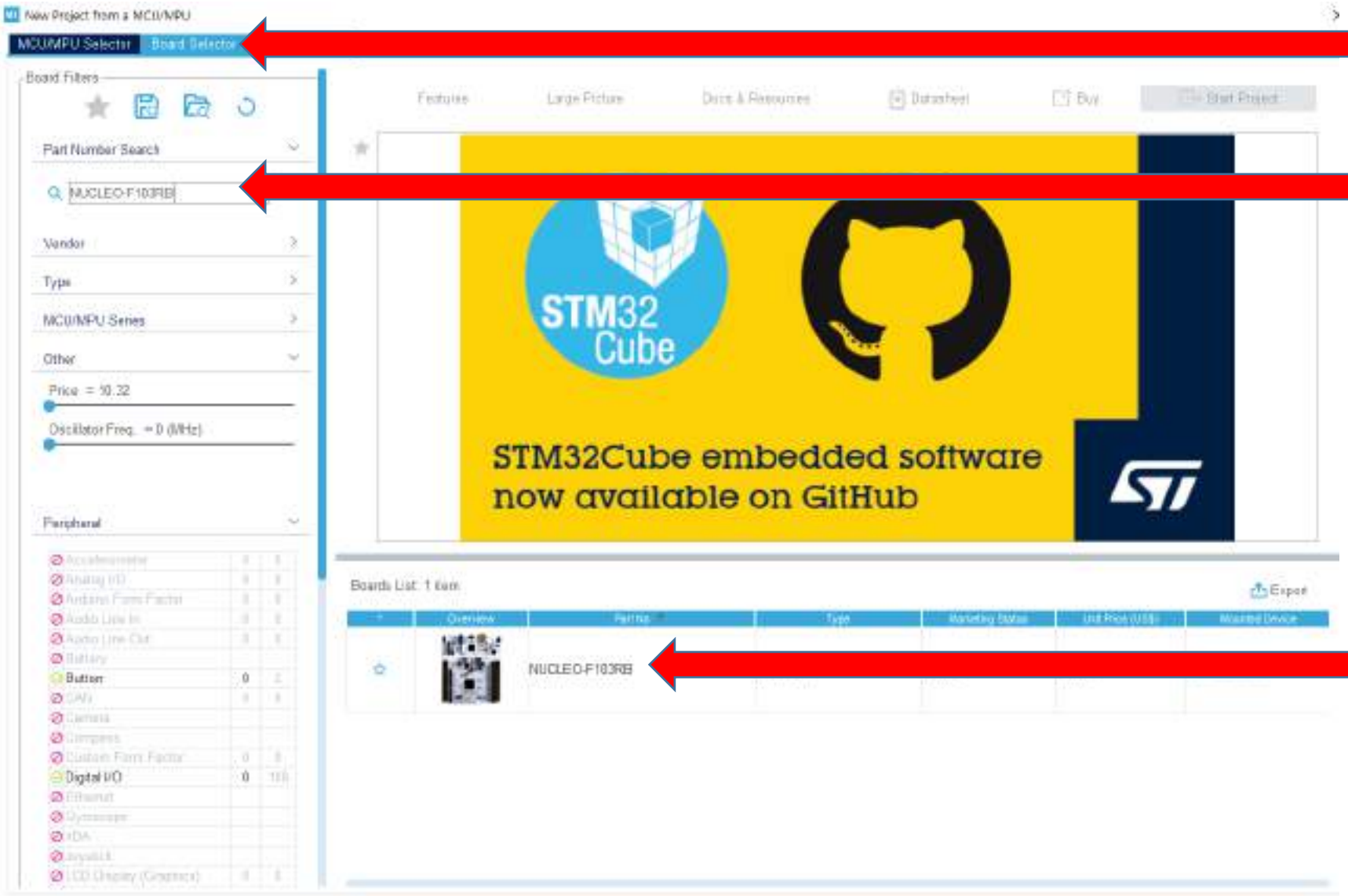
CubeMX로 예제 8.1 구현하기



보드 선택

8.6 타이머 CubeMX과제

CubeMX로 예제 8.1 구현하기



보드 선택

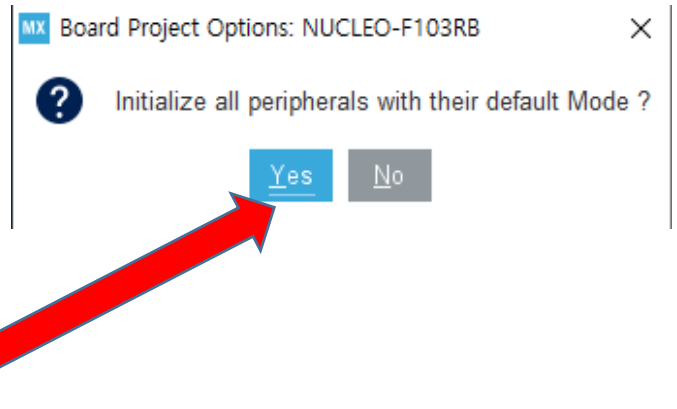
NUCLEO-F103RB 선택

NUCLEO-F103RB 더블클릭

KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

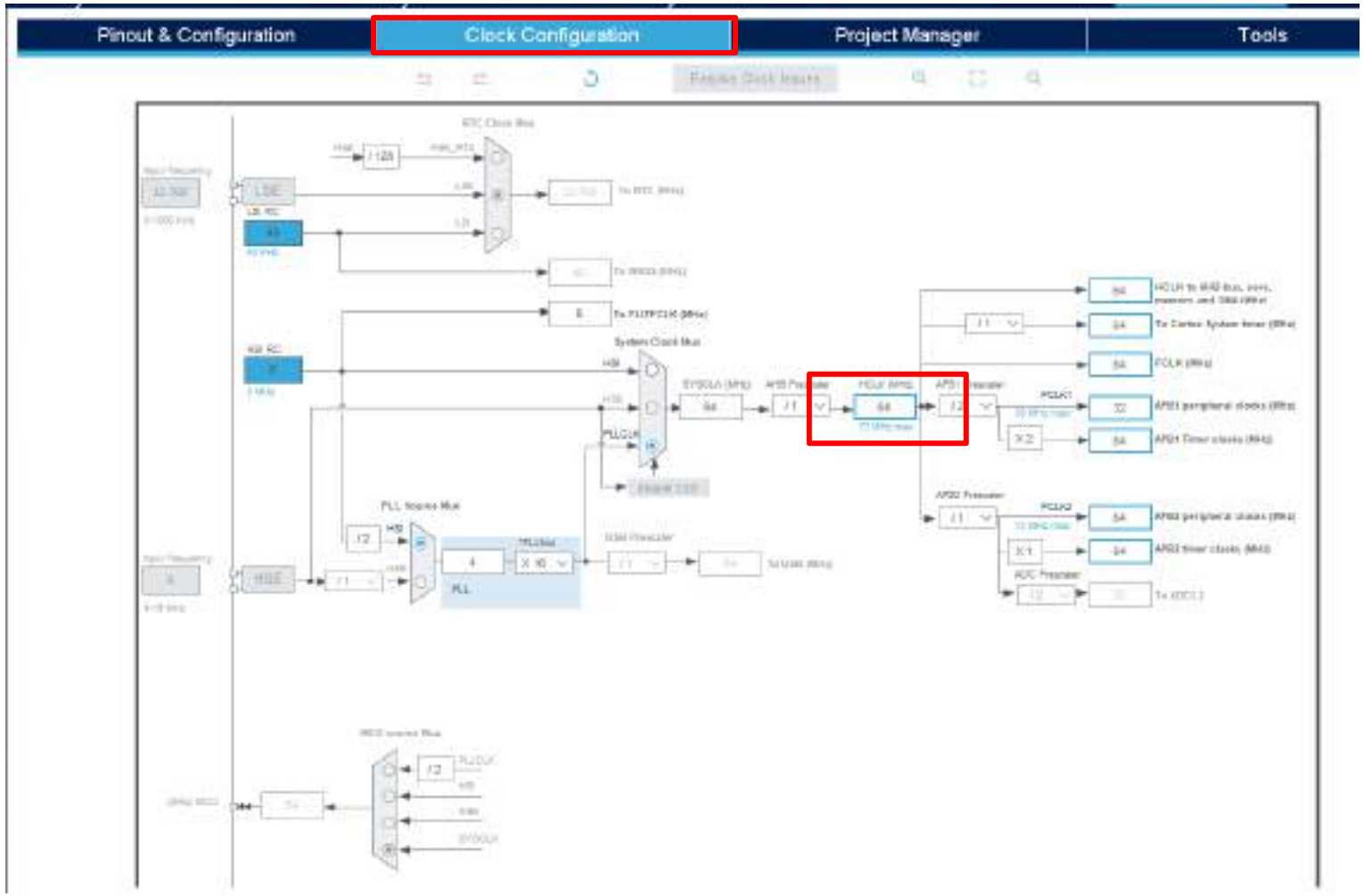
CubeMX로 예제 8.1 구현하기



Yes 선택하면 오른쪽과 같이 칩이 보임

8.6 타이머 CubeMX과제

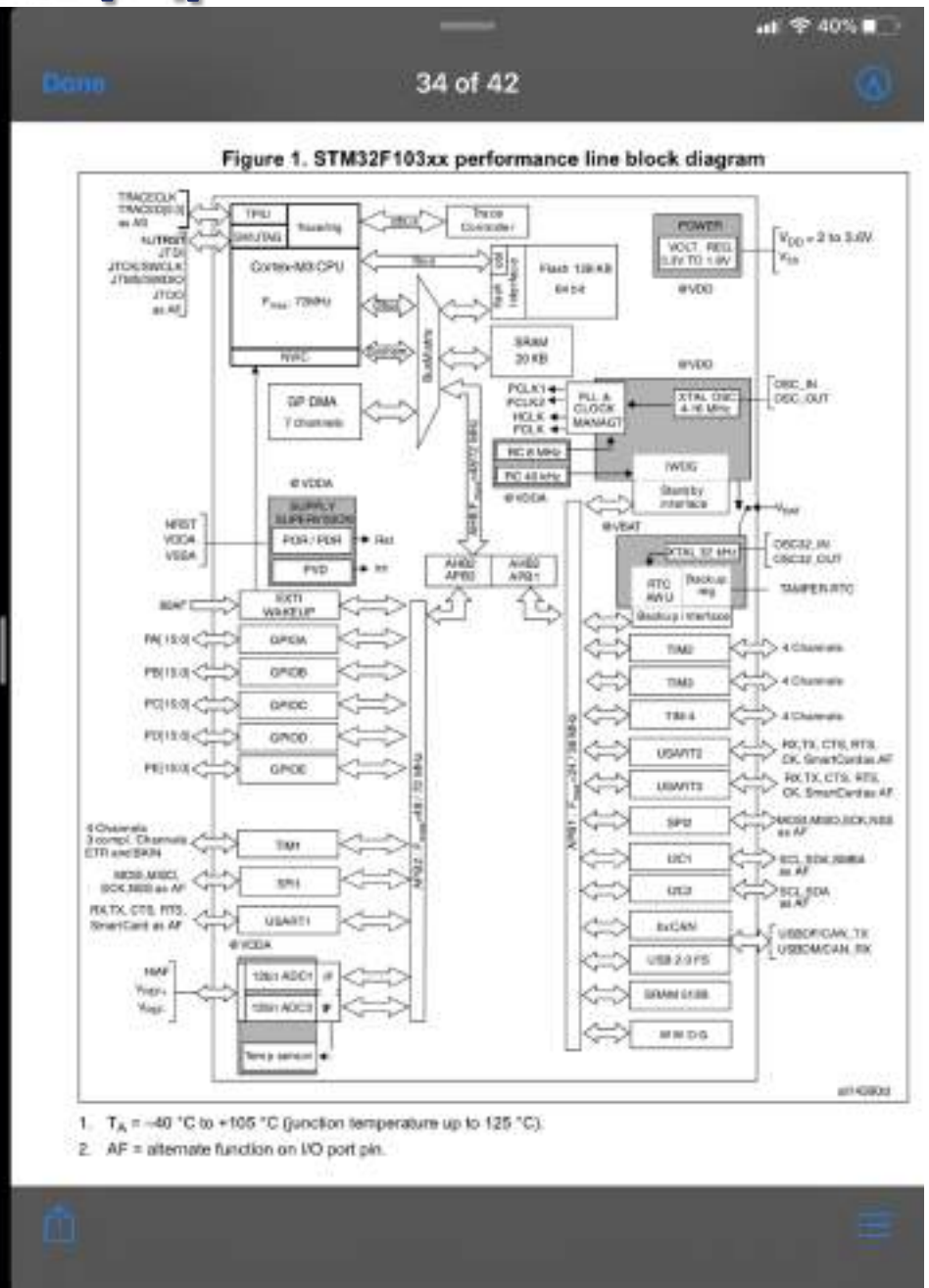
CubeMX로 예제 8.1 업 카운터(Up Counter)를 이용하여 LED를 1초 간격으로 ON/OFF 구현:



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

8.6 타이머 (인터럽트)CubeMX과제

```
  *       APB2 Prescaler          = 1
  *       PLLMUL                  = 16
  *       Flash Latency(WS)       = 2
  */
// ------------------------------------------------------------------------- //

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

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

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

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

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

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



KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

8:41 PM Mon Oct 28

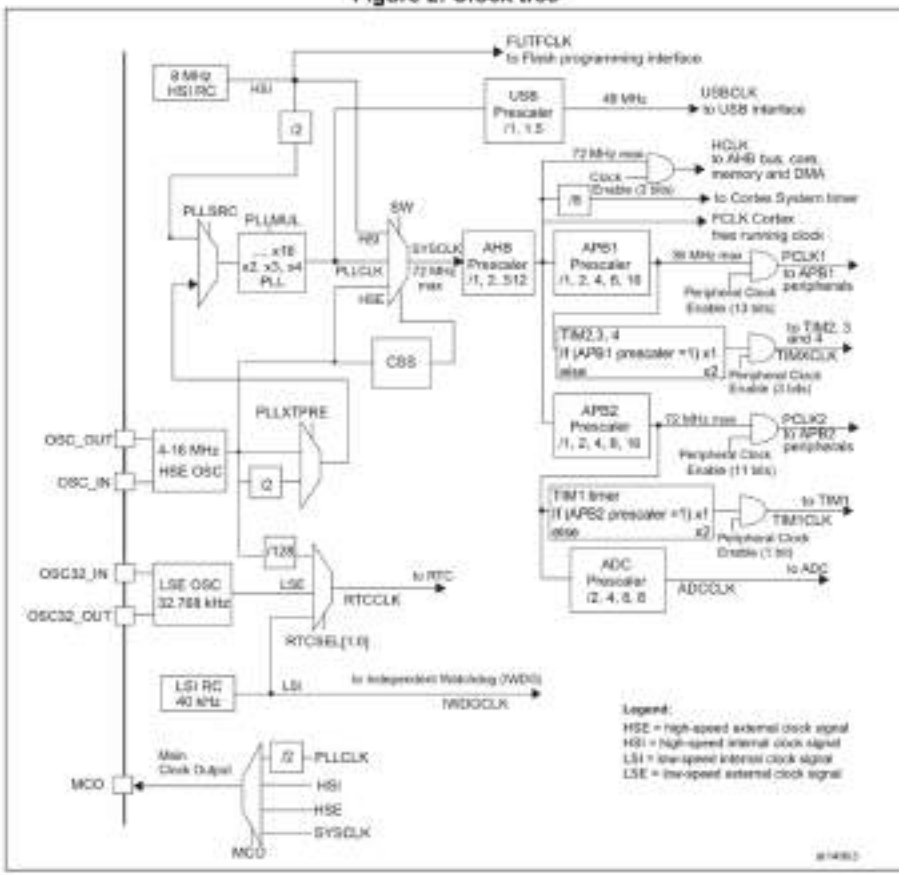
Done 2 of 4

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

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

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

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

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

42%

Done 34 of 42

Figure 2. Clock tree

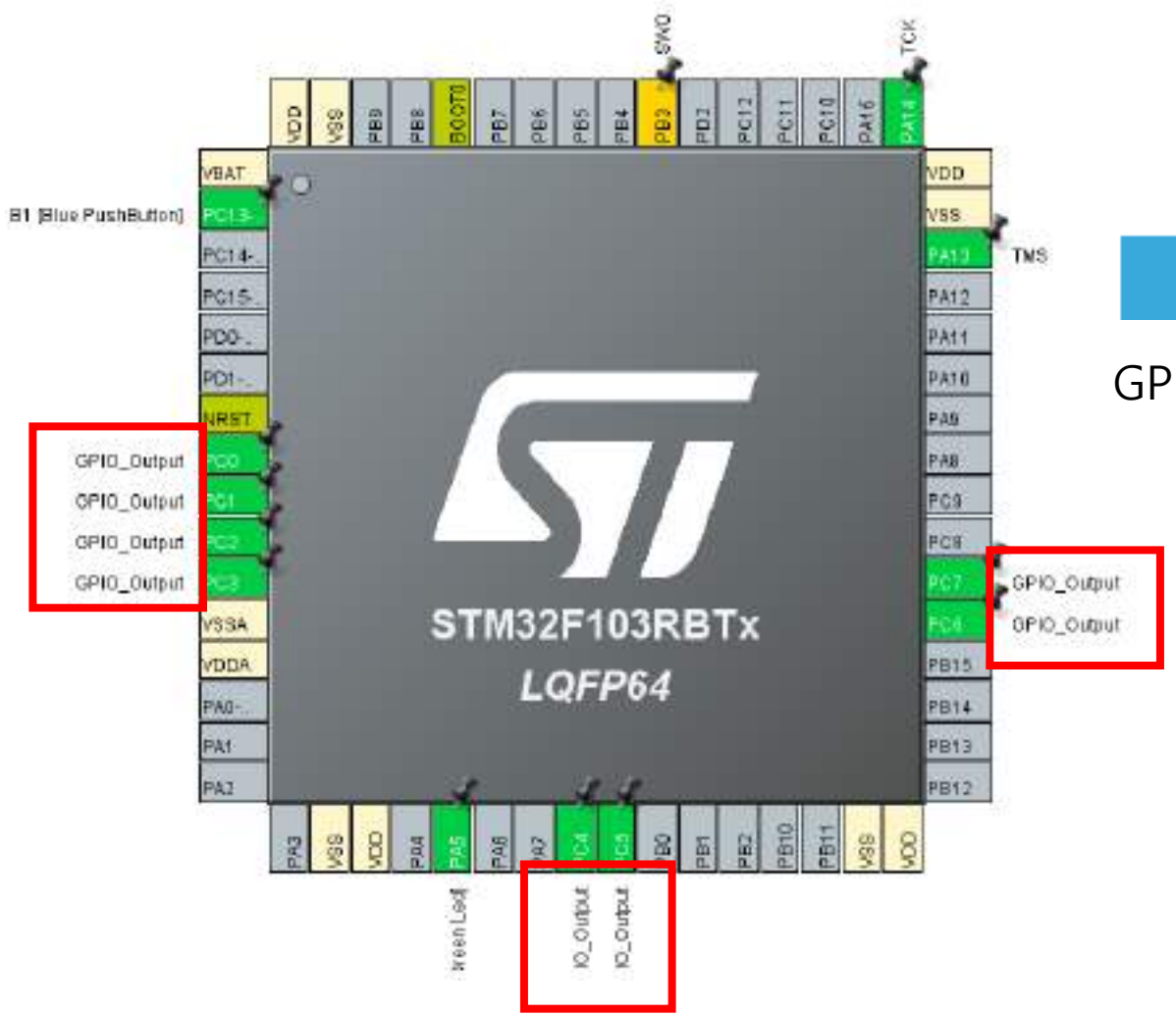


1. When the HSI is used as a PLL clock input, the maximum system clock frequency that can be achieved is 64 MHz.
2. For the USB function to be available, both HSE and PLL must be enabled, with USBCLK running at 48 MHz.
3. To have an ADC conversion time of 1 µs, APB2 must be at 14 MHz, 28 MHz or 56 MHz.

KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

CubeMX로 예제 8.1 업 카운터(Up Counter)를 이용하여 LED를 1초 간격으로 ON/OFF 구현:

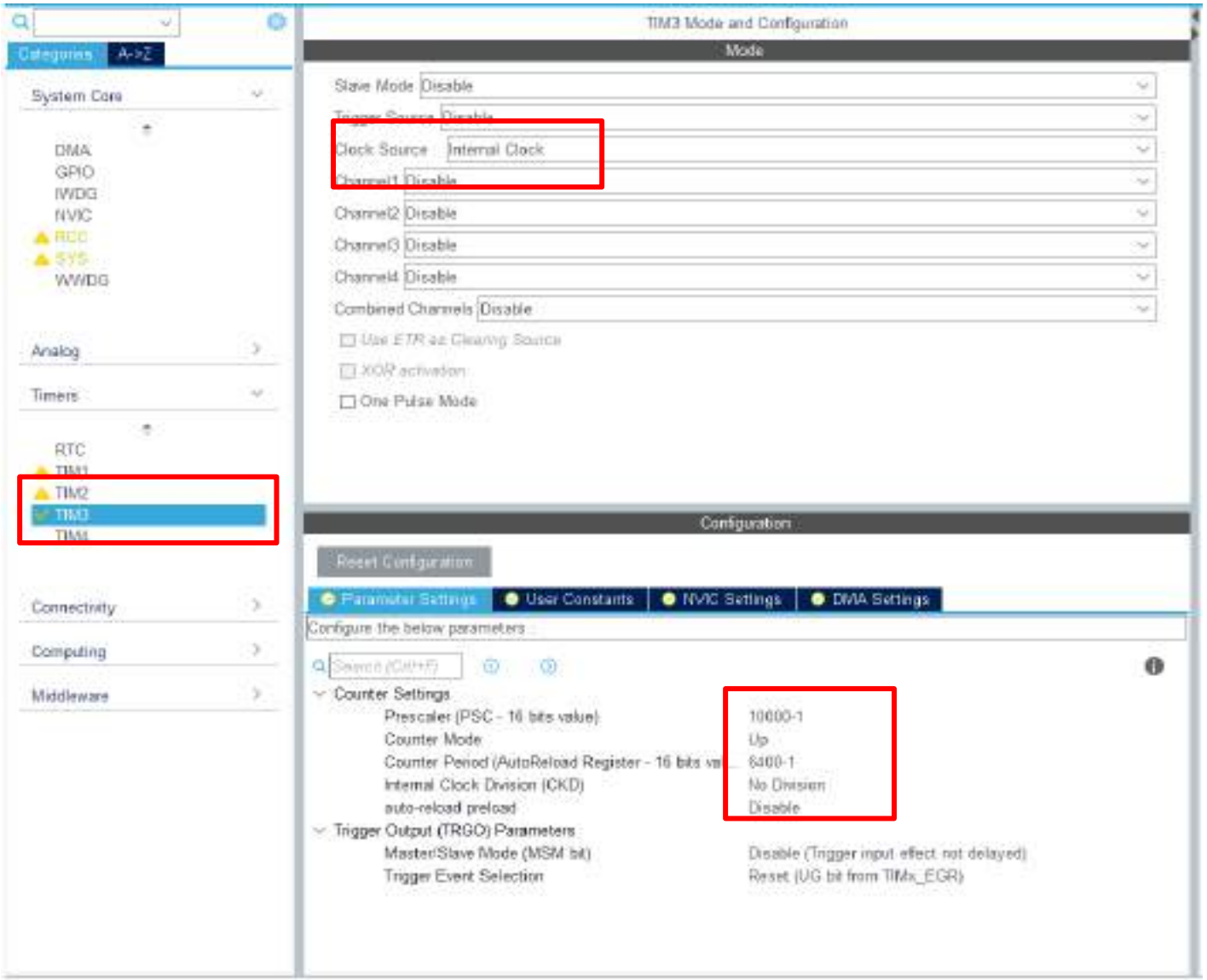


Pinout & Configuration 탭에서

GPIO 출력핀 설정 : PC0~7에 LED 연결됨

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기



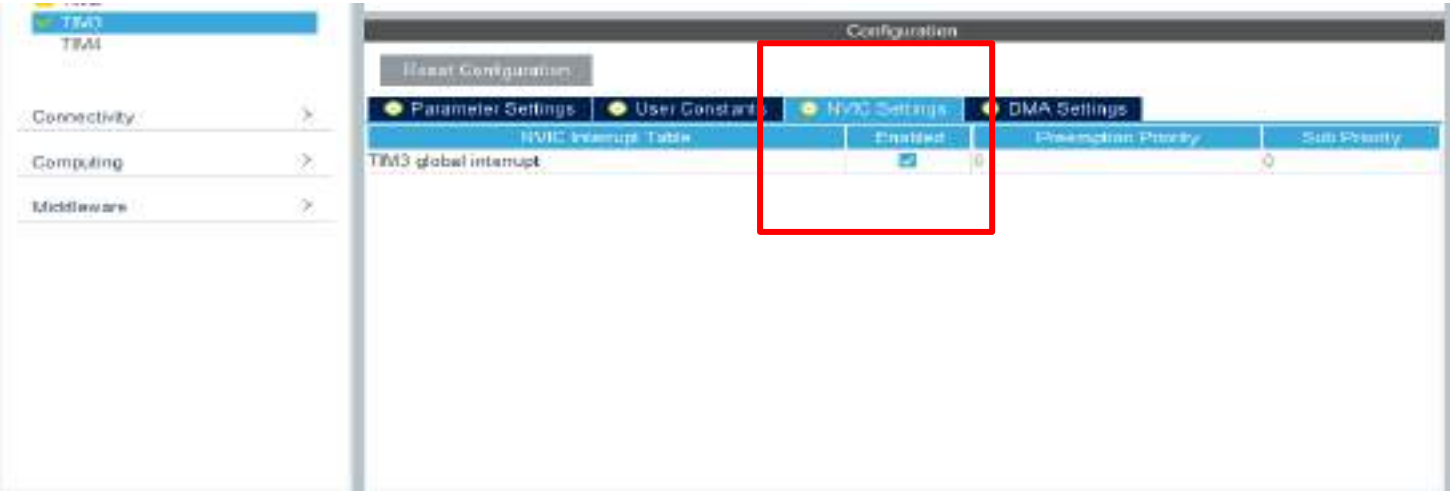
TIM3을 사용해보자.
Clock source → Internal clock

Parameter Setting 탭에서
Prescaler와 CounterPeriod(ARR) 값을 설정하여 64MHz --> 1Hz 로 분주

Prescaler를 10000으로하면, 64MHz가 6.4KHz로 분주되고, ARR을 6400으로하면 1Hz(1초에 한번씩) 인터럽트 되는 타이머인터럽트를 구현할 수 있음. 여기서 -1을 해주는 이유는 모든 카운터가 0부터 시작이기 때문임 (예를 들어, 10개를 세기 위해서는 0부터 9까지만 세야 함)

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기

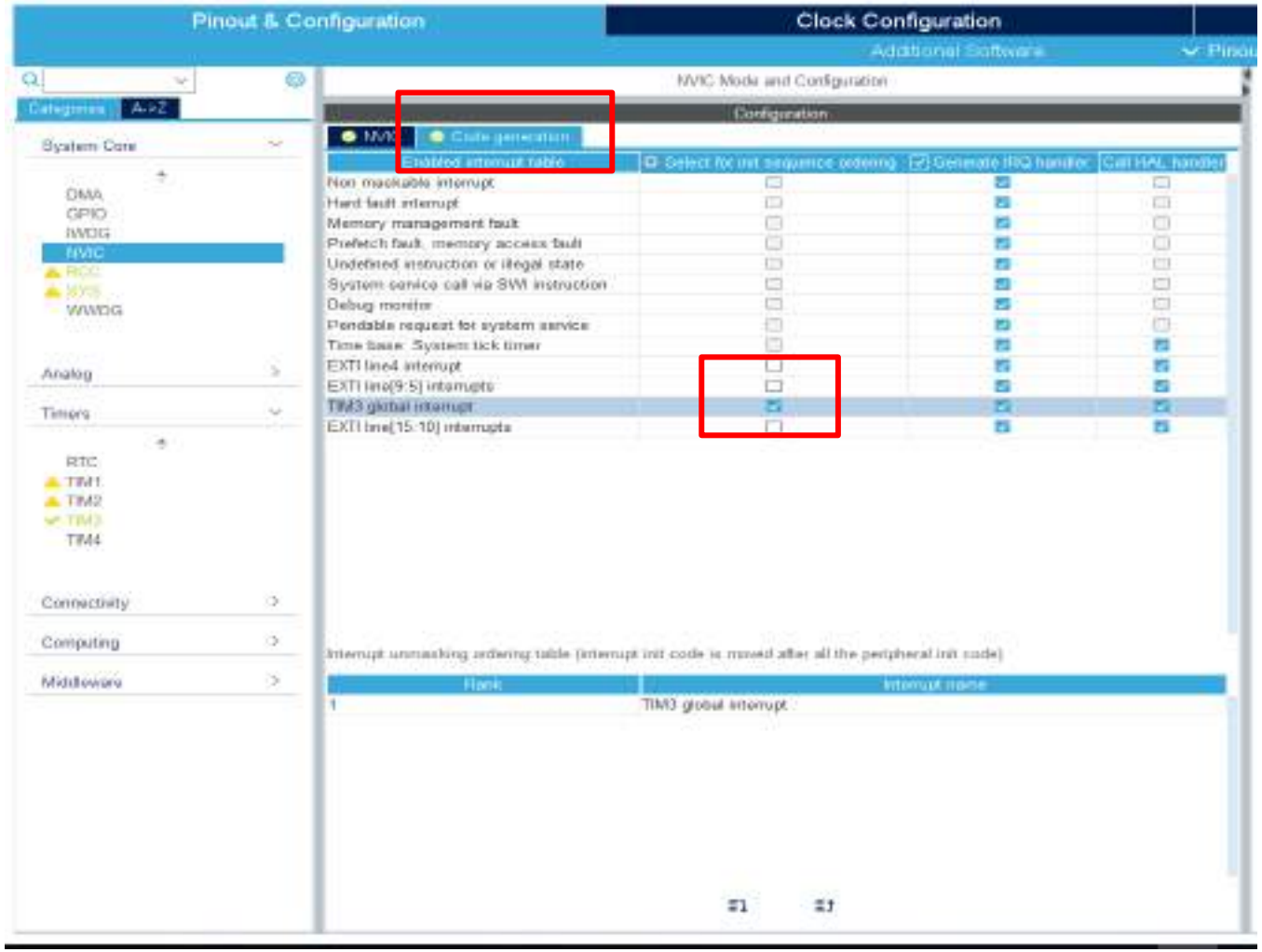


NVIC(Nested Vector Interrupt Controller) 설정 : 인터럽트 우선 순위를 설정하는 기능.

NVIC탭에서 TIM3 global interrup를 Enabled

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기

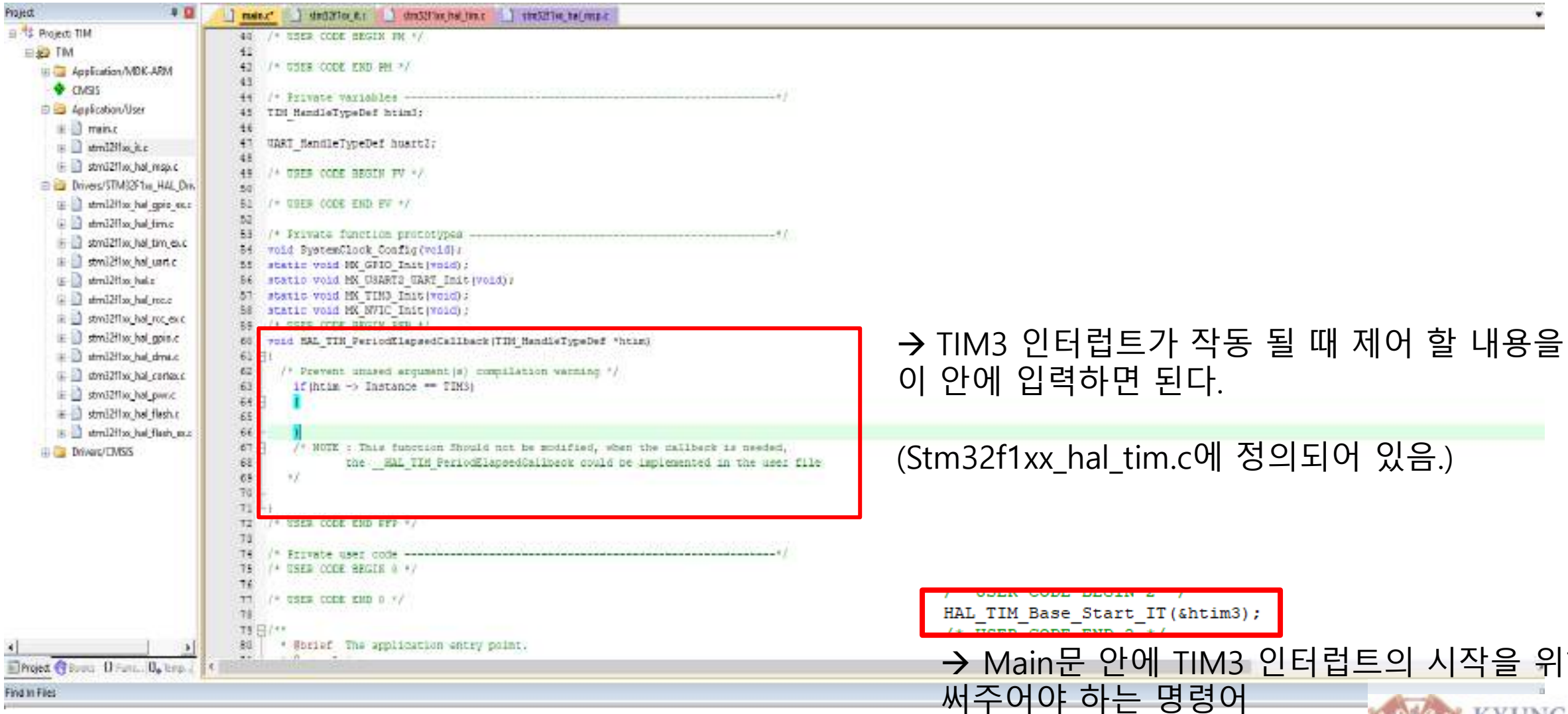


System Core의 NVIC탭에서
Code generation 탭을 클릭하여

TIM3 global interrupt를 Enabled 됐는지 확인

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기



→ TIM3 인터럽트가 작동 될 때 제어 할 내용을 이 안에 입력하면 된다.

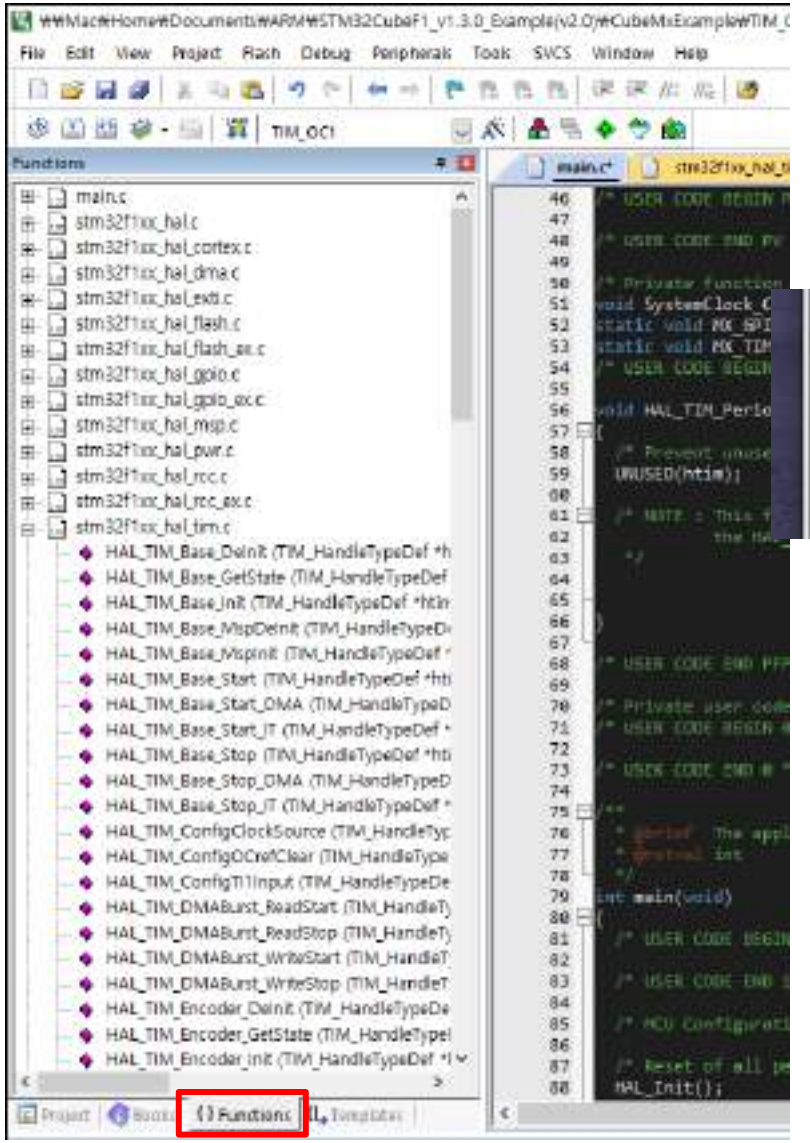
(Stm32f1xx_hal_tim.c에 정의되어 있음.)

```
HAL_TIM_Base_Start_IT(&htim3);
```

→ Main문 안에 TIM3 인터럽트의 시작을 위해 써주어야 하는 명령어

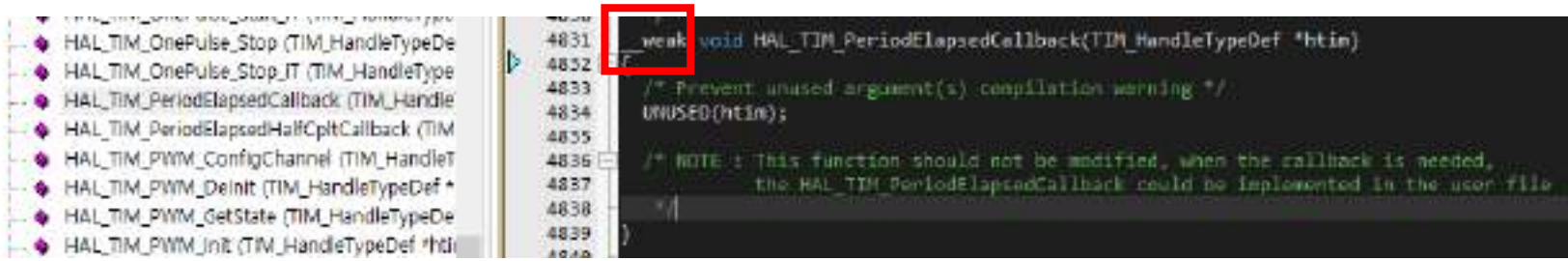
8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기



→ HAL 함수들은 MDK uVision의 왼쪽 윈도우에서 {} Function 탭을 열고 해당하는 c 라이브러리를 열면 찾을 수 있음

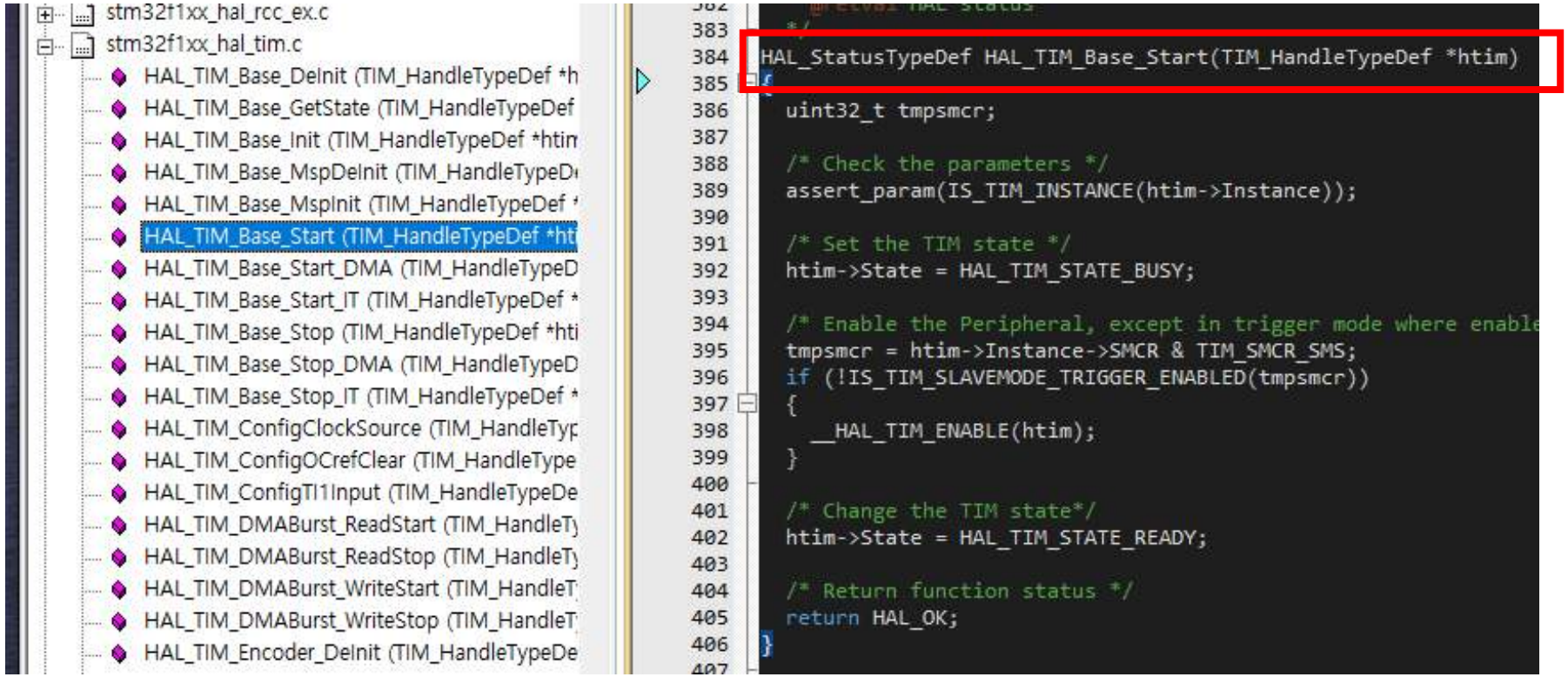
→ 예를 들어 앞 슬라이드의 HAL_TIM_PeriodElapsedCallback() 함수는 아래와 같이 찾을 수 있고 __weak 부분을 제외하고 copy하면 됨



KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기



→ c 라이브러리에 있는HAL 함수들 중 이름 앞에 __weak 가 없는 함수들은 main.c에 따로 저장할 필요없이 호출해서 사용하면 됨 (주의할 점)

8.6 타이머 (인터럽트)CubeMX과제

CubeMX가 자동생성

직접 코딩

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------

  /* Reset of all peripherals, Initializes the Flash interfa
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.1 구현하기

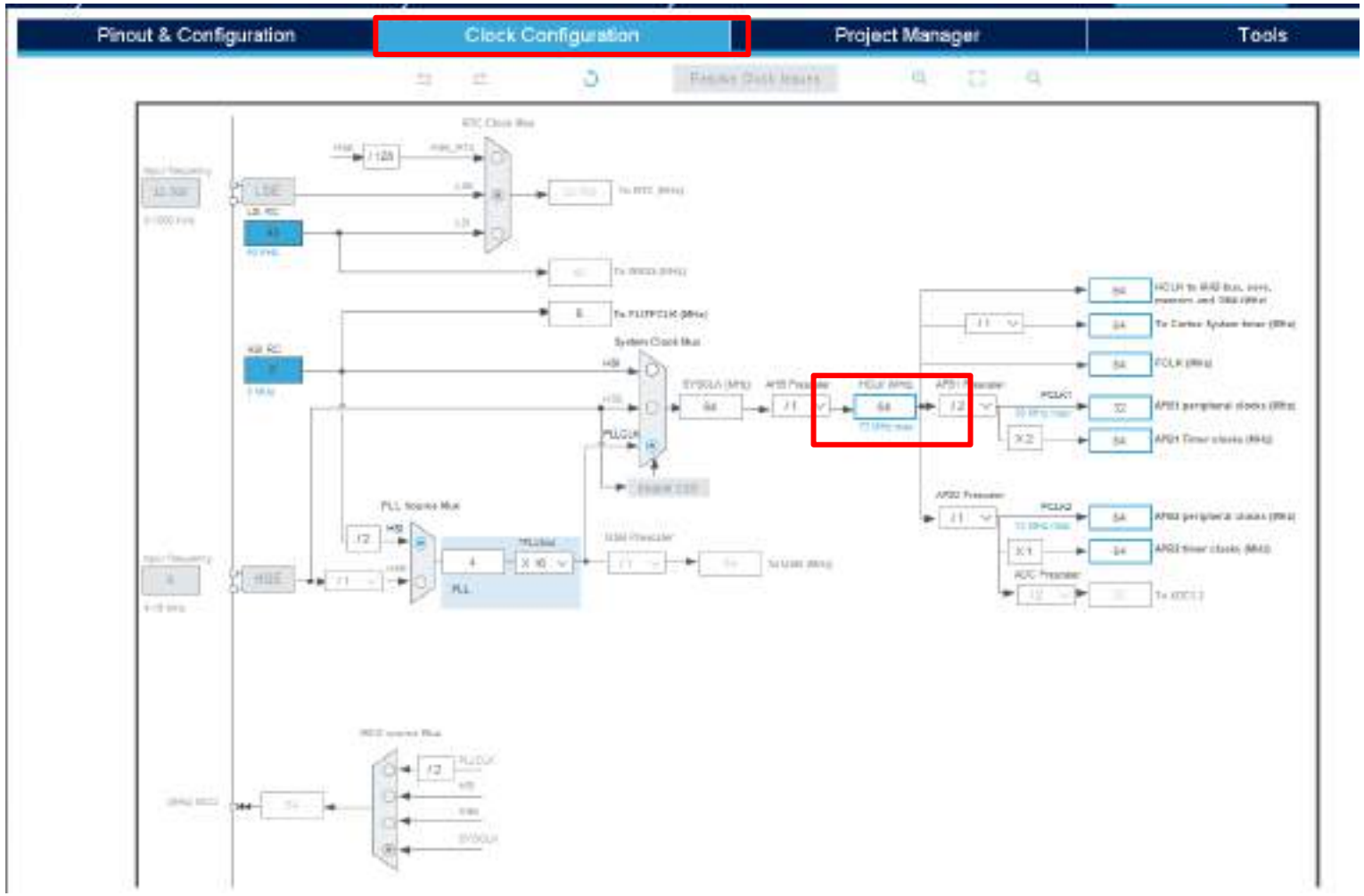
```
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
/* USER CODE BEGIN PFP */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if(htim->Instance == TIM3)
  {
      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_0|GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7);
  }

}
```

→ TIM3 인터럽트가 작동 될 때
PORT C의 0~7번 핀에 연결 된 I/O보드의 LED를 1초 간격으로 토글 하는 코드.
(Delay함수를 쓰지 않아도 미리 설정한 타이머 세팅에 의해 1초마다 작동하는 것을 확인 할 수 있다.)

→ 최종 코드(교과서의 코드와 CubeMX의 코드)를 비교해보세요

8.6 타이머 CubeMX과제

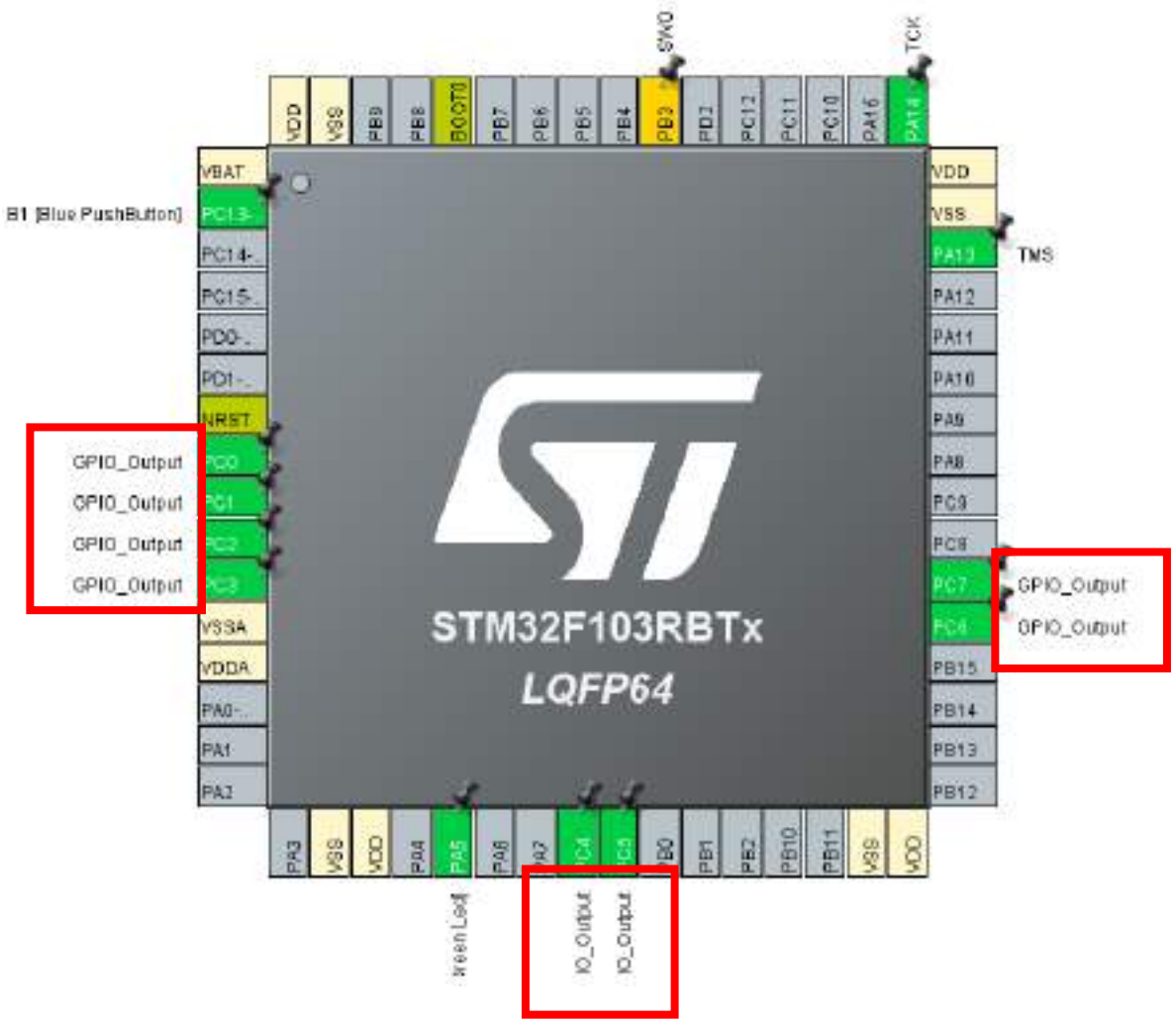
CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

8.6 타이머 CubeMX과제

CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:

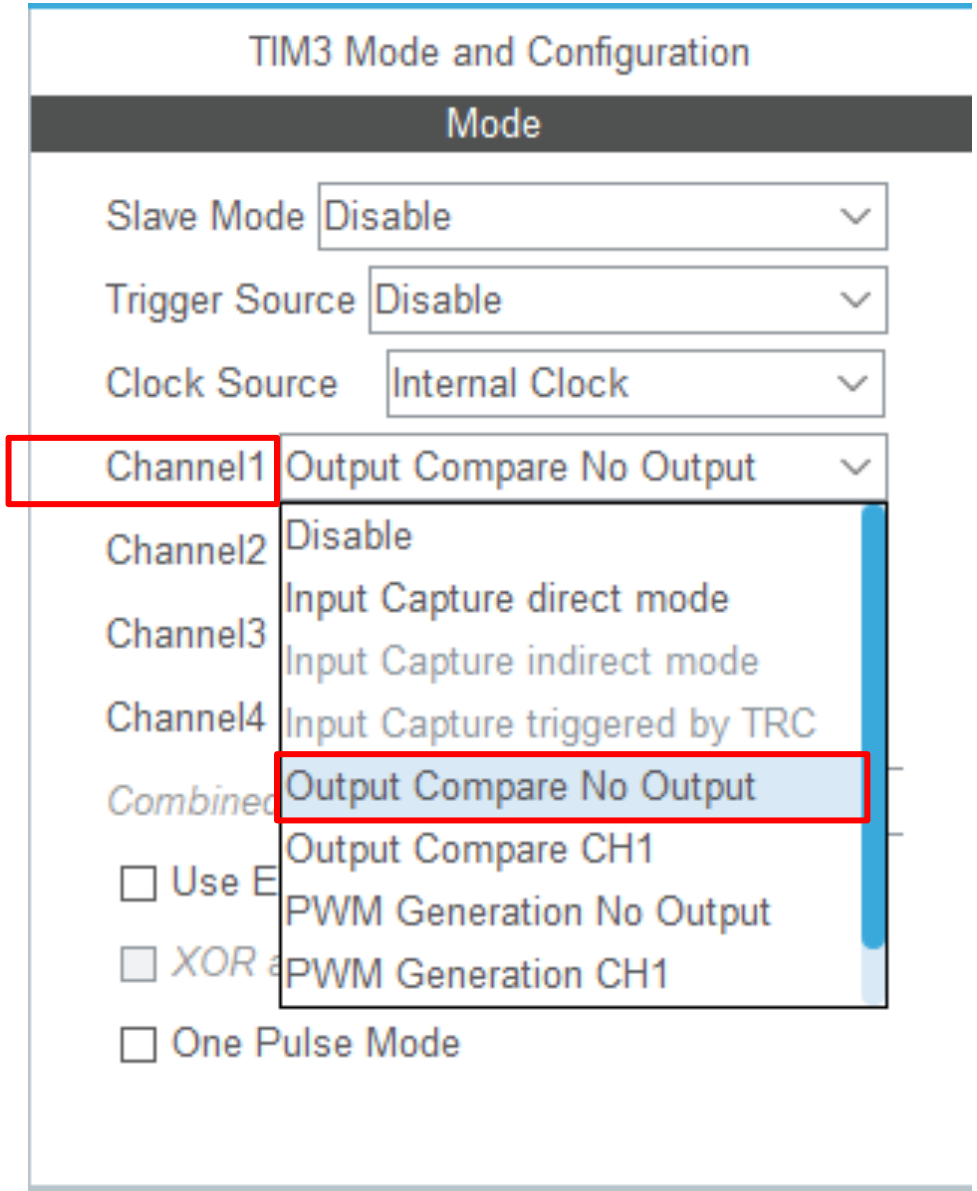


Pinout & Configuration 탭에서

GPIO 출력핀 설정 : PC0~7에 LED 연결됨

8.6 타이머 CubeMX과제

CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:

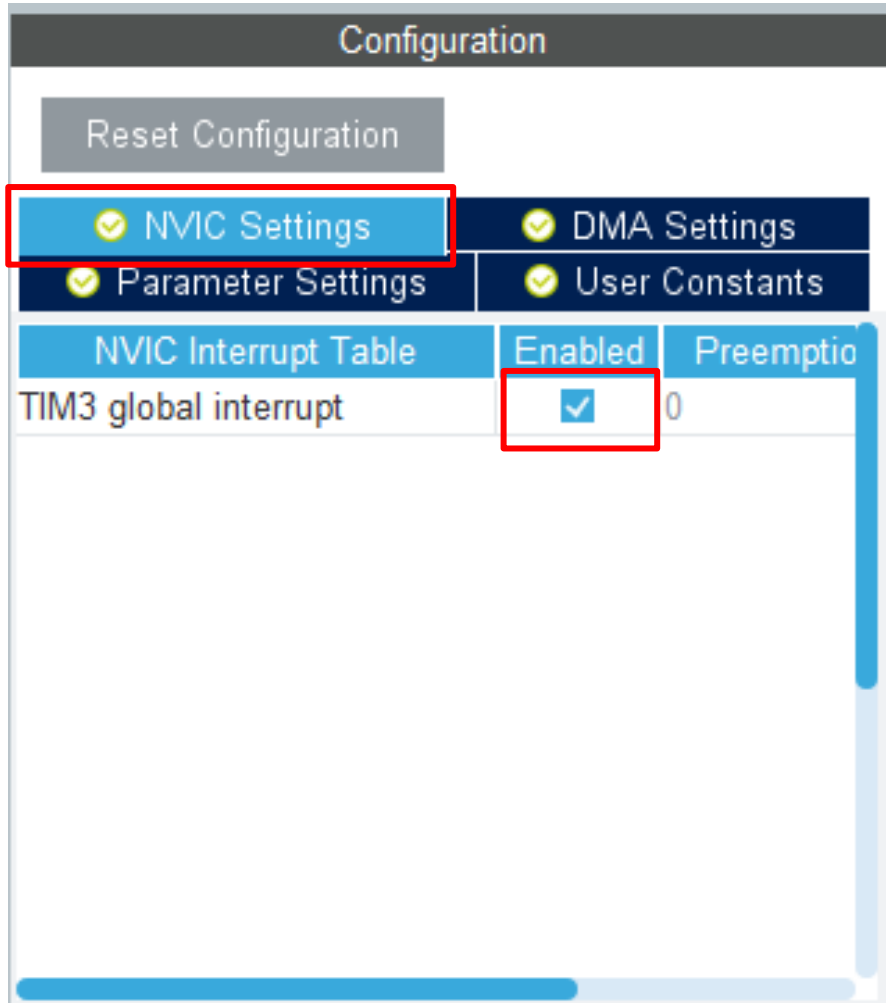


TIM3의 Channel1을 Output Compare No Output으로 모드 설정

KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

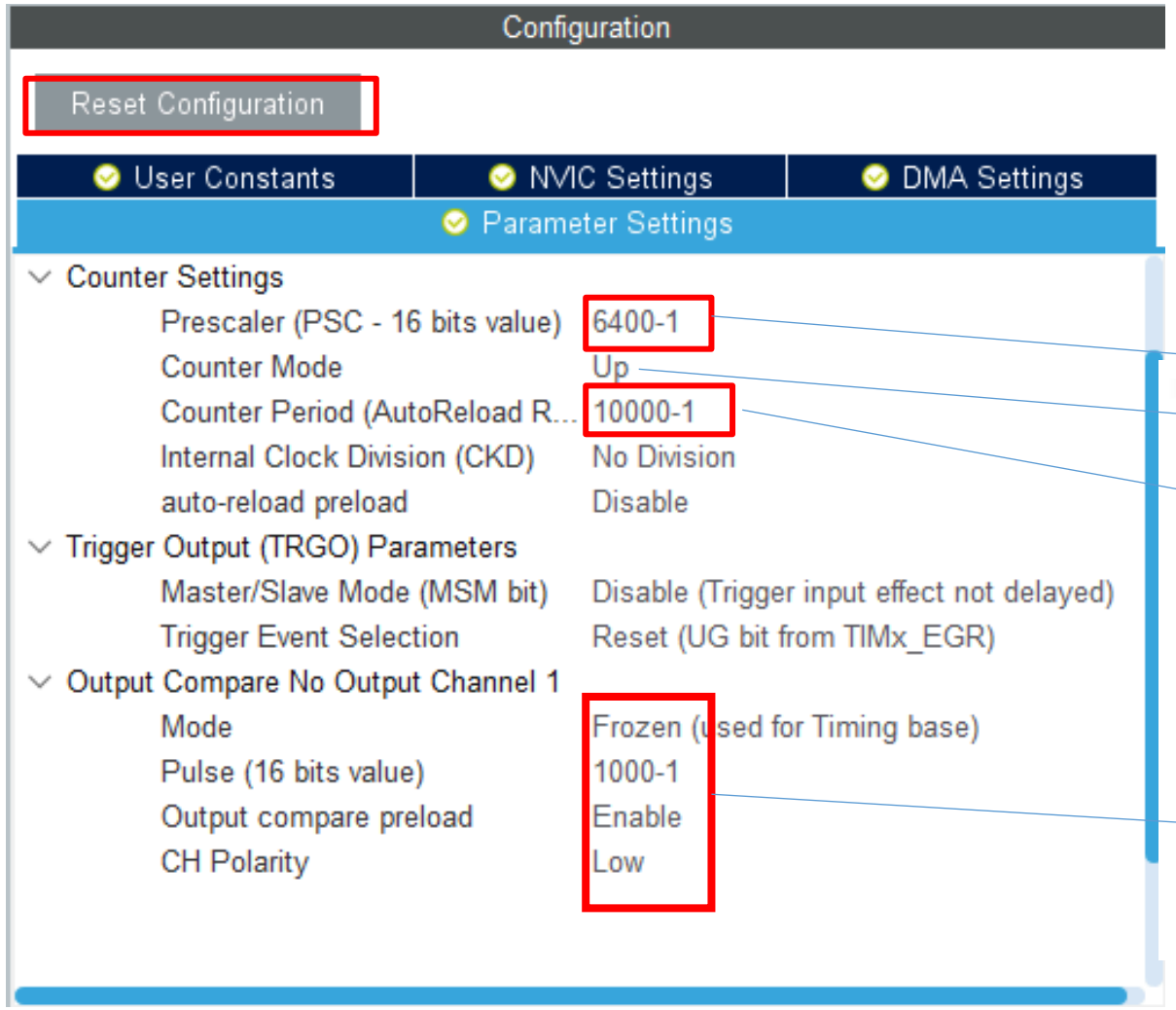
CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:



Configuration에서 NVIC Settings 탭을 선택하고 TIM3 global interrupt를 Enabled 체크함

8.6 타이머 CubeMX과제

CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:



출력 : 6장 과제와 동일(PC 0~7)

Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

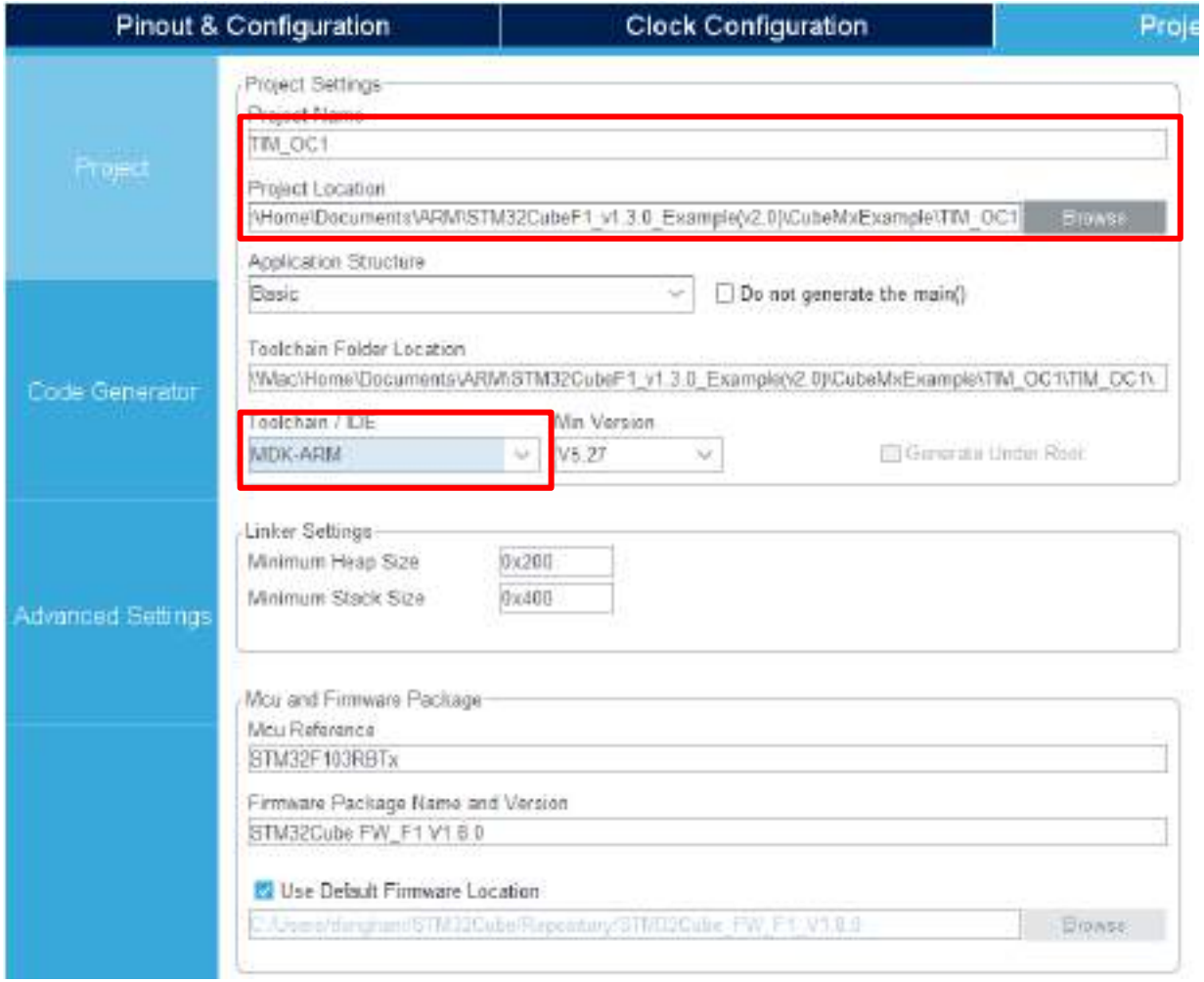
a) LED의 ON
- TimHandler 구조체 변수의 TimHandler.Init.Prescaler = 6399 (Nucleo-F429 확장보드는 8999), TimHandler.Init.Period = 9999 로 설정하고 업 카운터 모드로 동작을 한다.
- 그러면 카운터는 0~9999 사이를 업 카운팅을 하게되고, 1초 마다 UI(업데이트 인터럽트)가 발생한다.
- UI가 발생하면 UI 콜백함수인 HAL_TIM_PeriodElapsedCallback()가 호출되며, 이 함수에서 LED를 On 시켜준다.

(앞 슬라이드)

b) LED 의 OFF
- OC(출력비교) 모드에서 TIM_OCInit 구조체 변수의 "TIM_OCInit.Pulse의 설정값 = 업 카운터의 카운팅 값"이 될때 출력 비교 인터럽트(CC2I)가 발생한다.
- 그러면 OC 인터럽트 콜백함수인 HAL_TIM_OC_DelayElapsedCallback()가 호출되며, 이 함수에서 LED를 Off 시켜준다.

8.6 타이머 CubeMX과제

CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:



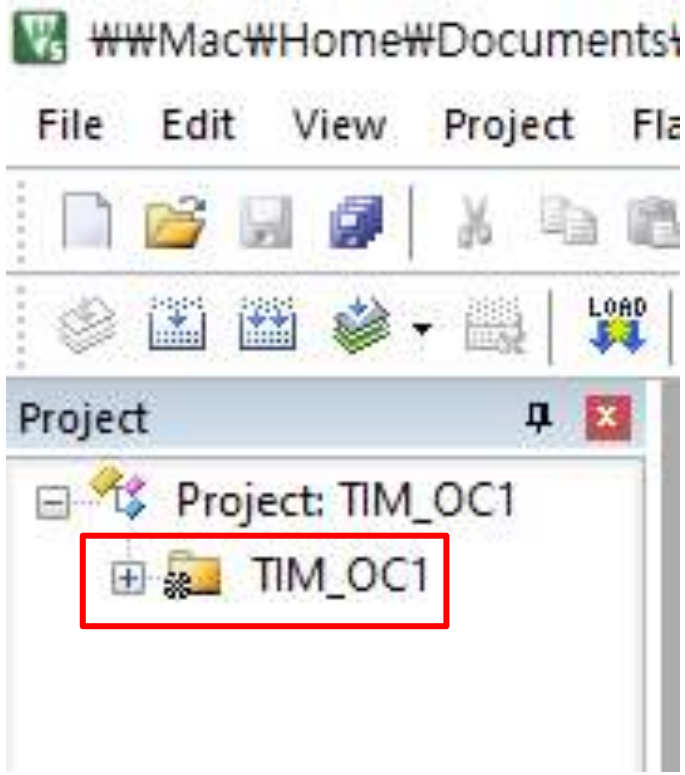
Project Manager 탭을 누르면, 왼쪽의 그림처럼 project 생성 단계가 되며, project name과 location을 정해줌.

Toolchain / IDE는 꼭 MDK-ARM을 선택

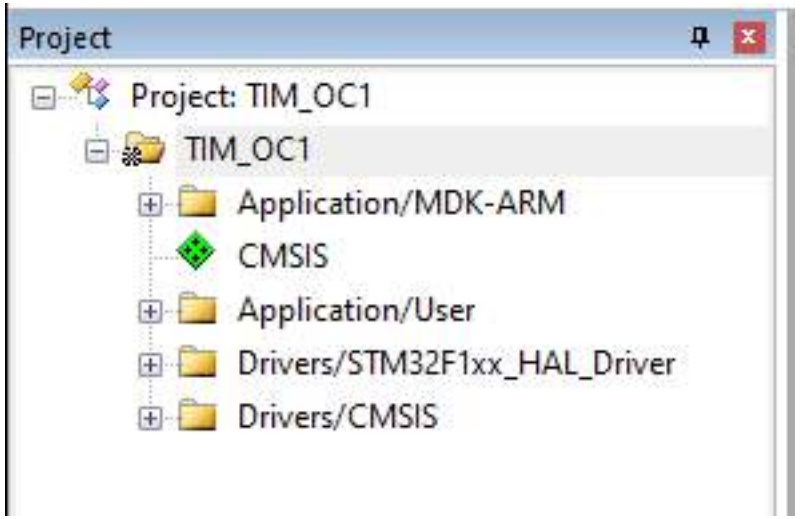
마지막으로 GENERATE CODE 탭을 누르면, 코드가 생성되면서 MDK uVision이 실행됨

8.6 타이머 CubeMX과제

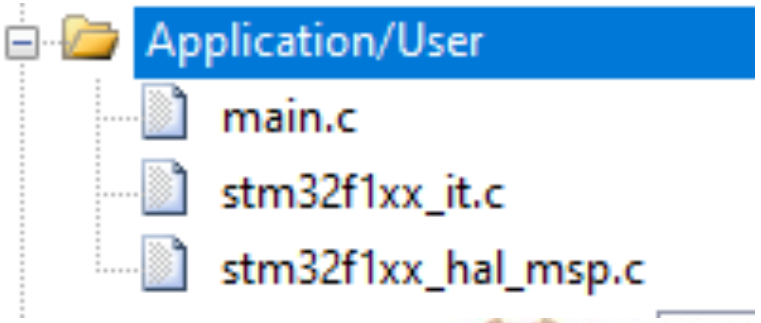
CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:



MDK uVision에서 왼쪽의 Project 창에 CubeMX에 만든 project인 TIM_OC1이 보이고 있으며, + 를 눌러서 펼치면



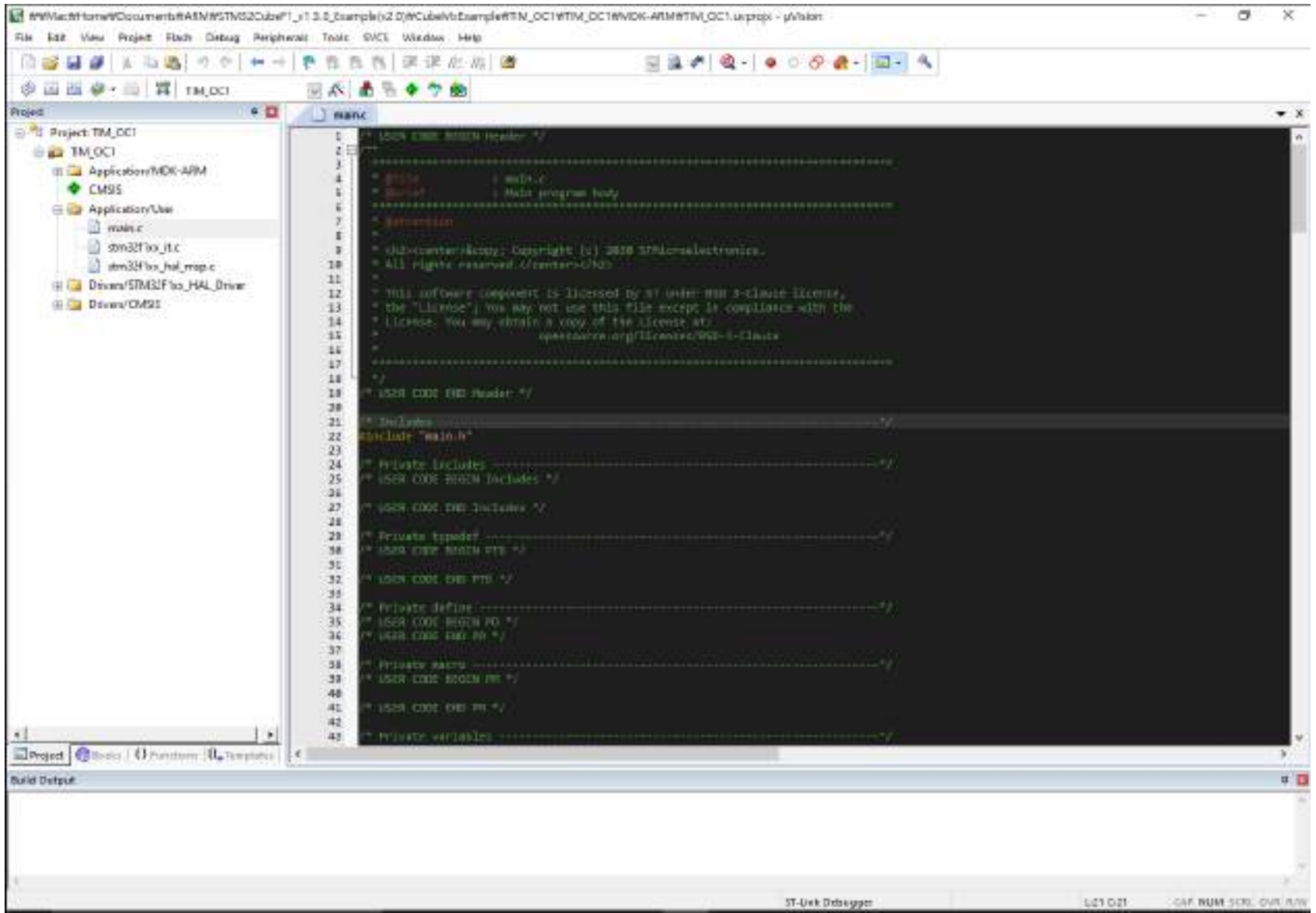
오른쪽 그림의 Application/User를 펼치면, main.c 가 보임. 이를 더블클릭함



KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

CubeMX로 예제 8.3 업 카운터와 OC(출력비교)를 이용하여 LED의 ON/OFF 구현:

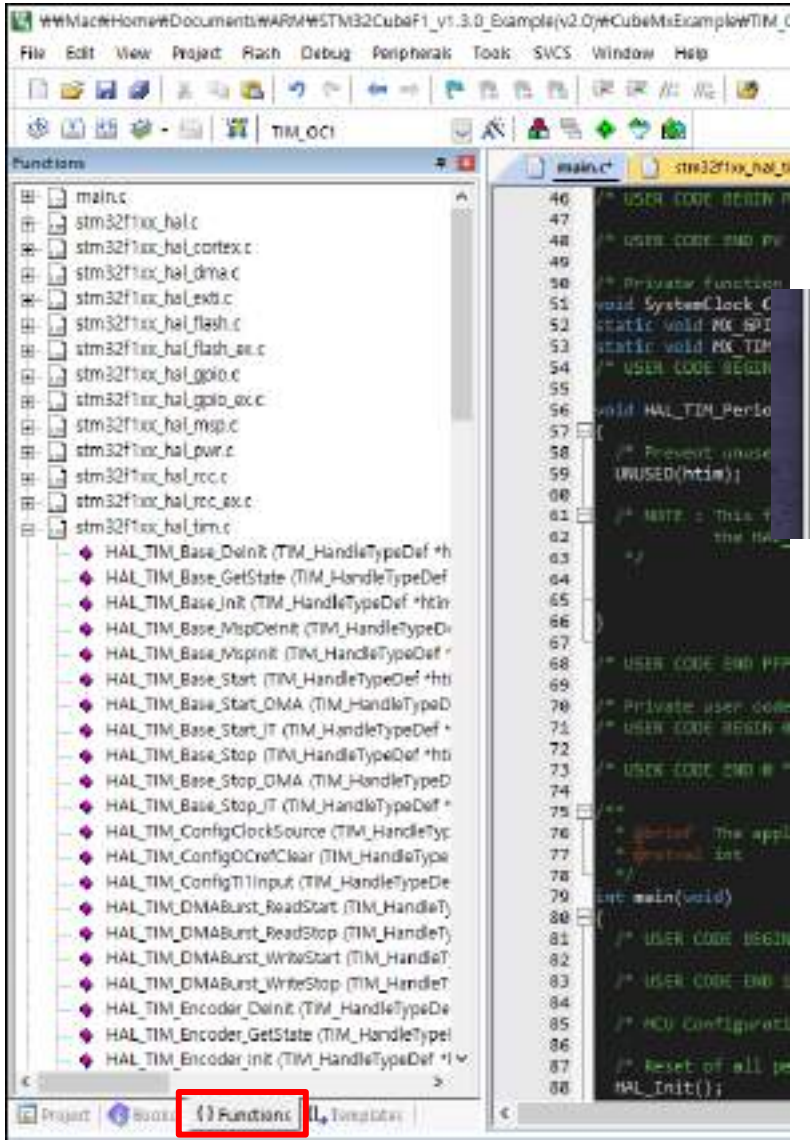


그림의 오른쪽 editor window에 CubeMX가 자동으로 생성해준 기본 코드가 보이며, 이를 수정함

KYUNG HEE UNIVERSITY

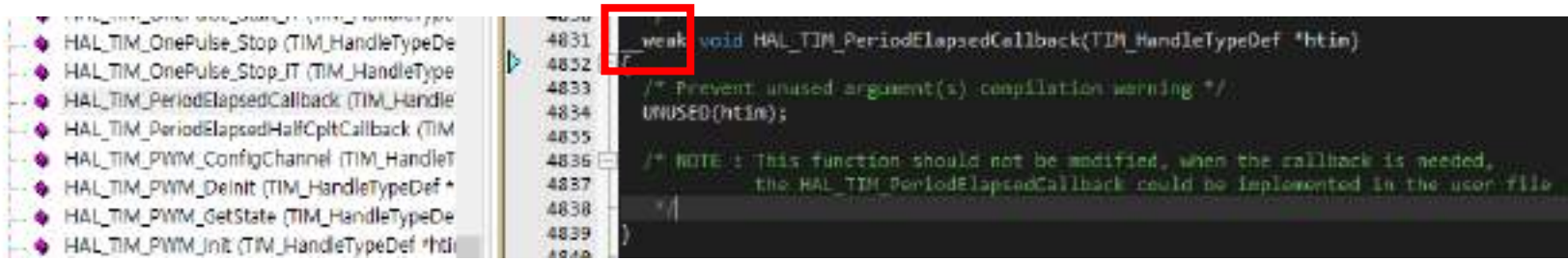
8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.3 구현하기



→ HAL 함수들은 MDK uVision의 왼쪽 윈도우에서 {} Function 탭을 열고 해당하는 c 라이브러리를 열면 찾을 수 있음

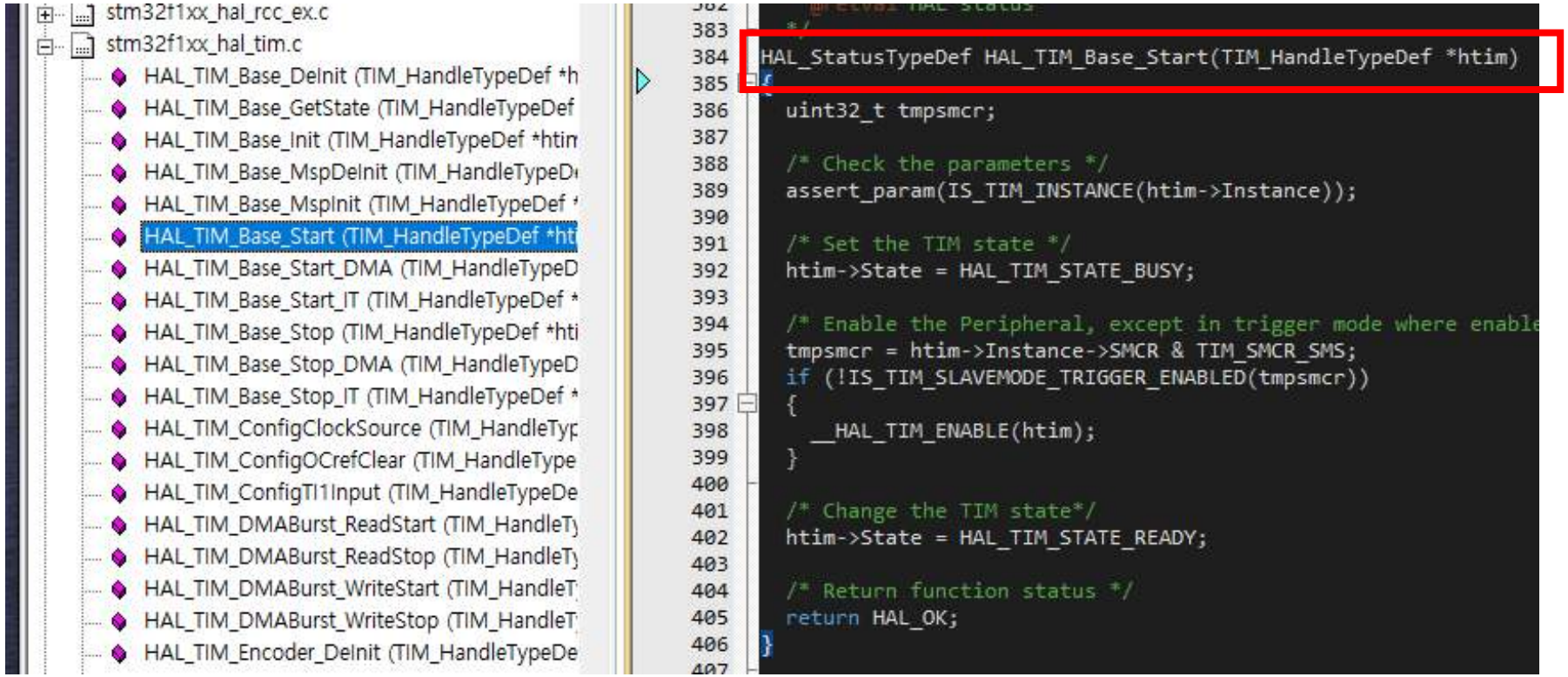
→ 예를 들어 앞 슬라이드의 HAL_TIM_PeriodElapsedCallback() 함수는 아래와 같이 찾을 수 있고 __weak 부분을 제외하고 copy하면 됨



KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.3 구현하기



→ c 라이브러리에 있는HAL 함수들 중 이름 앞에 __weak 가 없는 함수들은 main.c에 따로 저장할 필요없이 호출해서 사용하면 됨 (주의할 점)

8.6 타이머 (인터럽트)CubeMX과제

main.c | stm32f1xx_hal_tim.c | stm32f1xx_hal_gpio.h | stm32f1xx_it.c | stm32f1xx_hal_gpio.c

```
50  /* Private function prototypes -----------------------------------------------*/
51  void SystemClock_Config(void);
52  static void MX_GPIO_Init(void);
53  static void MX_TIM3_Init(void);
54  /* USER CODE BEGIN PFP */
55
56  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
57  {
58    /* Prevent unused argument(s) compilation warning */
59    UNUSED(htim);
60
61    /* NOTE : This function should not be modified, when the callback is needed,
62              the HAL_TIM_PeriodElapsedCallback could be implemented in the user file
63     */
64    if (htim->Instance == TIM3)
65    {
66        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
67                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_SET);
68    }
69
70
71  }
72
73  void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
74  {
75    /* Prevent unused argument(s) compilation warning */
76    UNUSED(htim);
77
78    /* NOTE : This function should not be modified, when the callback is needed,
79              the HAL_TIM_OC_DelayElapsedCallback could be implemented in the user file
80     */
81    if (htim->Instance == TIM3)
82    {
83        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
84                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_RESET);
85    }
86
87  }
```

→ TIM3의 PeriodElapsedCallback() 함수는 1초에 한번씩 발생하고, 본 예제에서는 이때 LED를 켬

→ TIM3의 Output Compare 인터럽트는 CNT가 0에서 시작해서 1000번째 될 OC_DelayElapsedCallback() 함수를 실행해서 LED를 끔

→ 최종 코드(교과서의 예제 8.3코드와 CubeMX의 코드)를 비교해보세요

8.6 타이머 (인터럽트)CubeMX과제

main.c | stm32f1xx_hal_tim.c | stm32f1xx_hal_gpio.h | stm32f1xx_it.c | stm32f1xx_hal_gpio.c

```
50  /* Private function prototypes -----------------------------------------------*/
51  void SystemClock_Config(void);
52  static void MX_GPIO_Init(void);
53  static void MX_TIM3_Init(void);
54  /* USER CODE BEGIN PFP */
55
56  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
57  {
58    /* Prevent unused argument(s) compilation warning */
59    UNUSED(htim);
60
61    /* NOTE : This function should not be modified, when the callback is needed,
62              the HAL_TIM_PeriodElapsedCallback could be implemented in the user file
63     */
64    if (htim->Instance == TIM3)
65    {
66        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
67                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_SET);
68    }
69
70
71  }
72
73  void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
74  {
75    /* Prevent unused argument(s) compilation warning */
76    UNUSED(htim);
77
78    /* NOTE : This function should not be modified, when the callback is needed,
79              the HAL_TIM_OC_DelayElapsedCallback could be implemented in the user file
80     */
81    if (htim->Instance == TIM3)
82    {
83        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
84                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_RESET);
85    }
86
87  }
```

→ TIM3의 PeriodElapsedCallback() 함수는 1초에 한번씩 발생하고, 본 예제에서는 이때 LED를 켬

→ TIM3의 Output Compare 인터럽트는 CNT가 0에서 시작해서 1000번째 될 OC_DelayElapsedCallback() 함수를 실행해서 LED를 끔

→ 최종 코드(교과서의 예제 8.3코드와 CubeMX의 코드)를 비교해보세요

8.6 타이머 (인터럽트)CubeMX과제

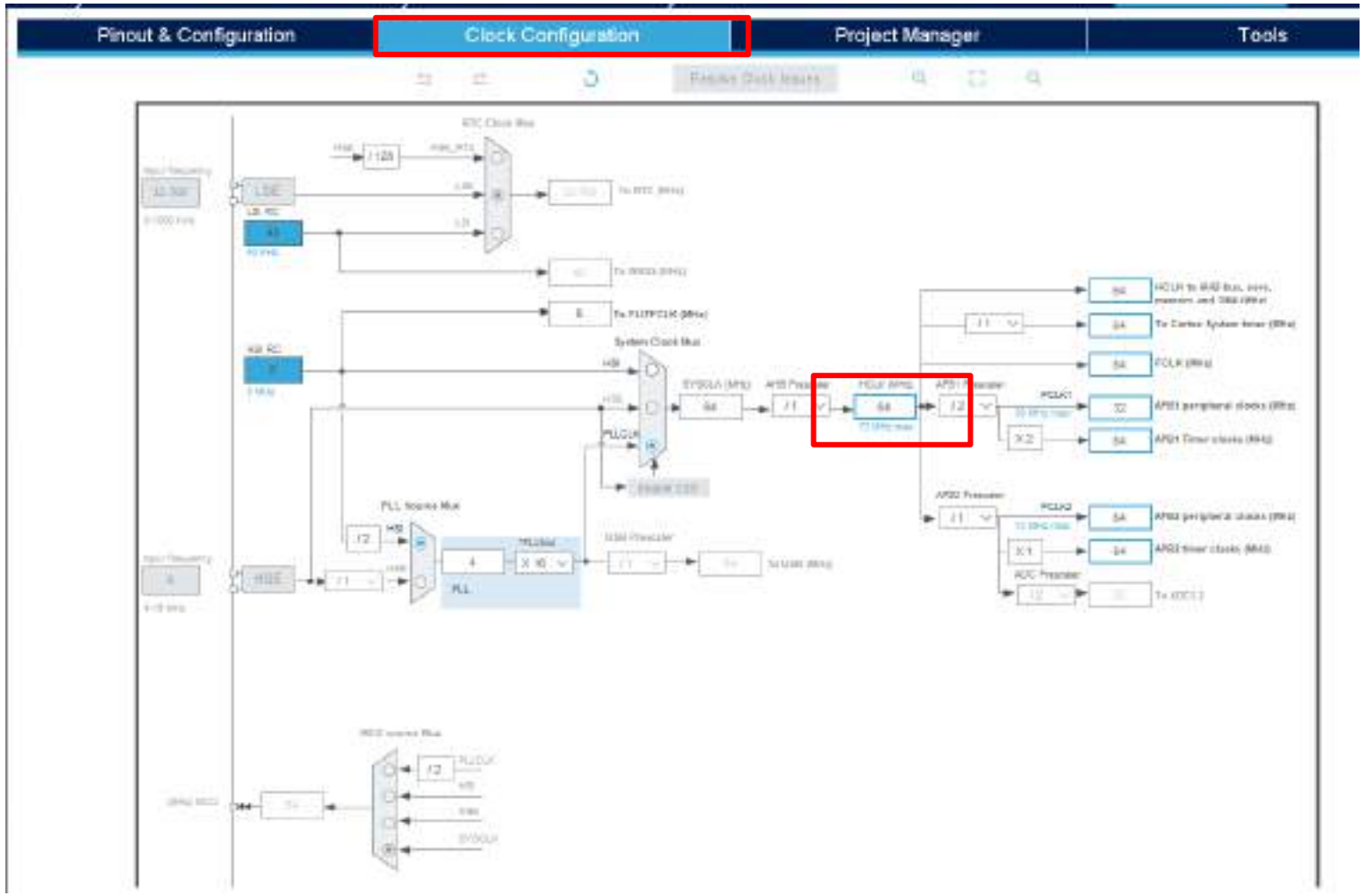
CubeMX가 자동생성

직접 코딩

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------

  /* Reset of all peripherals, Initializes the F
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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

8.6 타이머 CubeMX과제

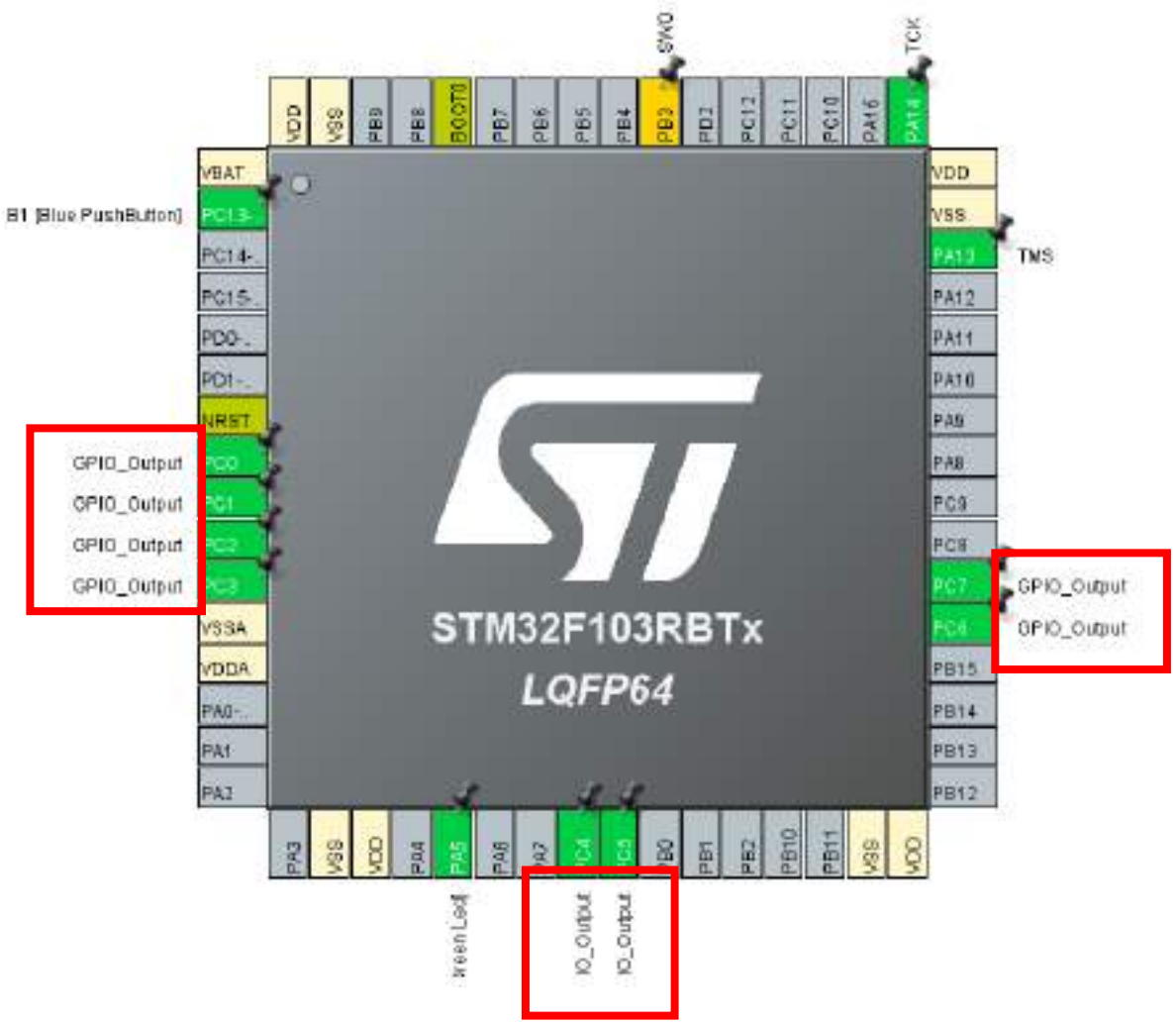
CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

8.6 타이머 CubeMX과제

CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):

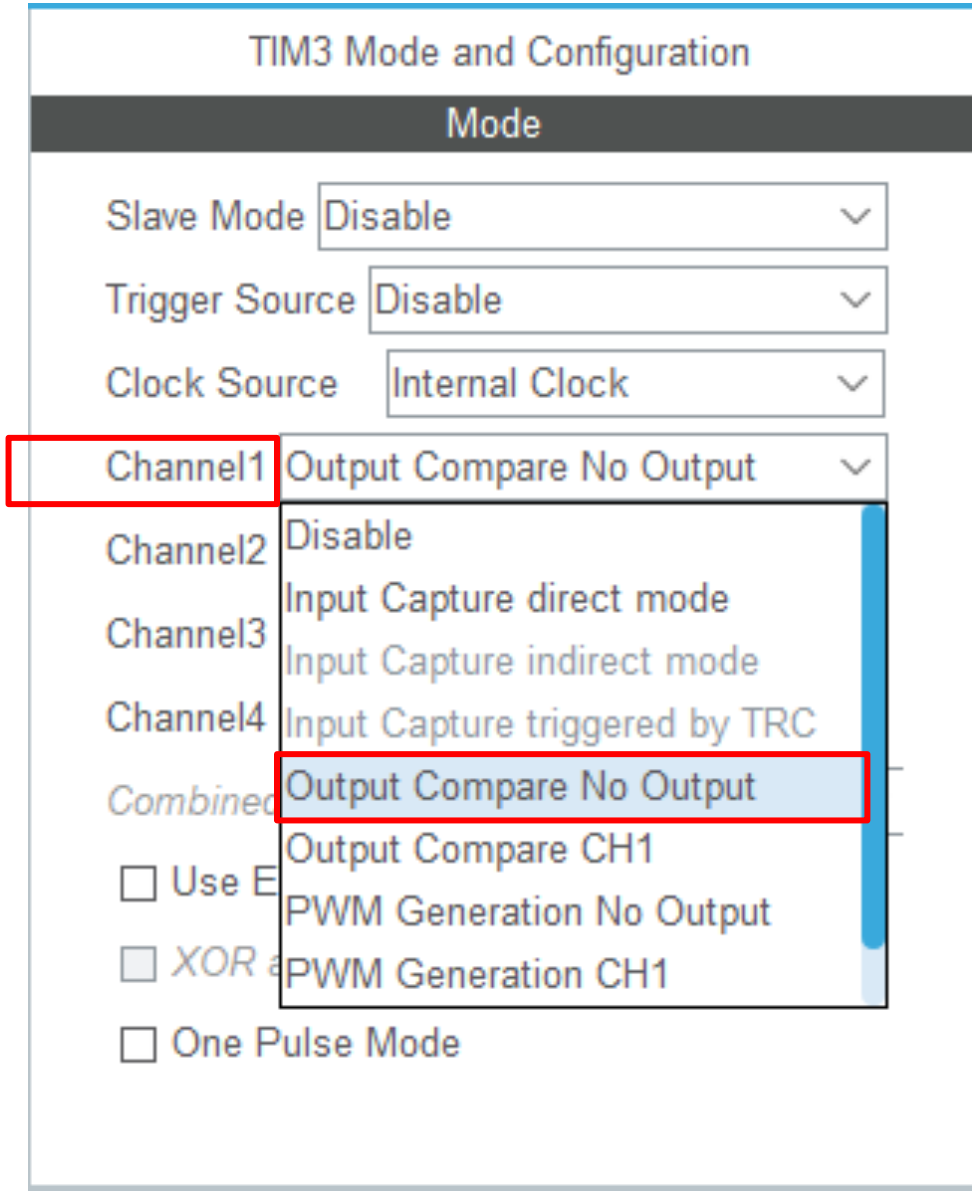


Pinout & Configuration 탭에서

GPIO 출력핀 설정 : PC0~7에 LED 연결됨

8.6 타이머 CubeMX과제

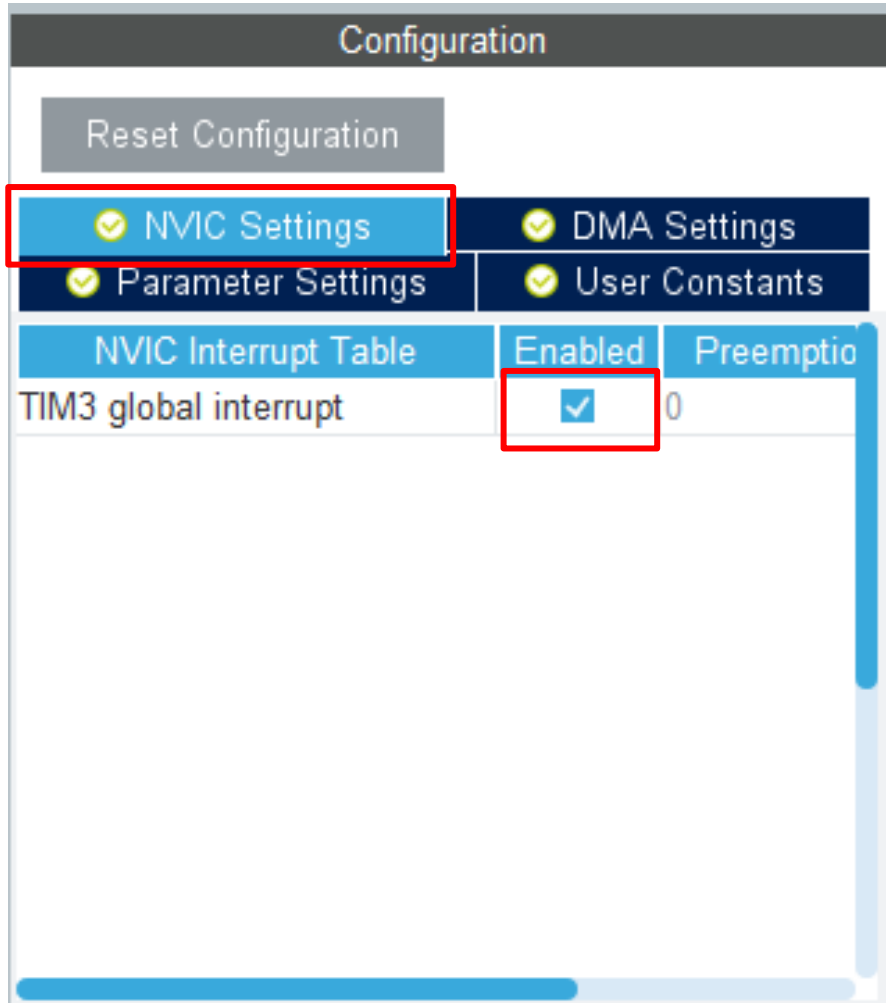
CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



TIM3의 Channel1을 Output Compare No Output으로 모드 설정

8.6 타이머 CubeMX과제

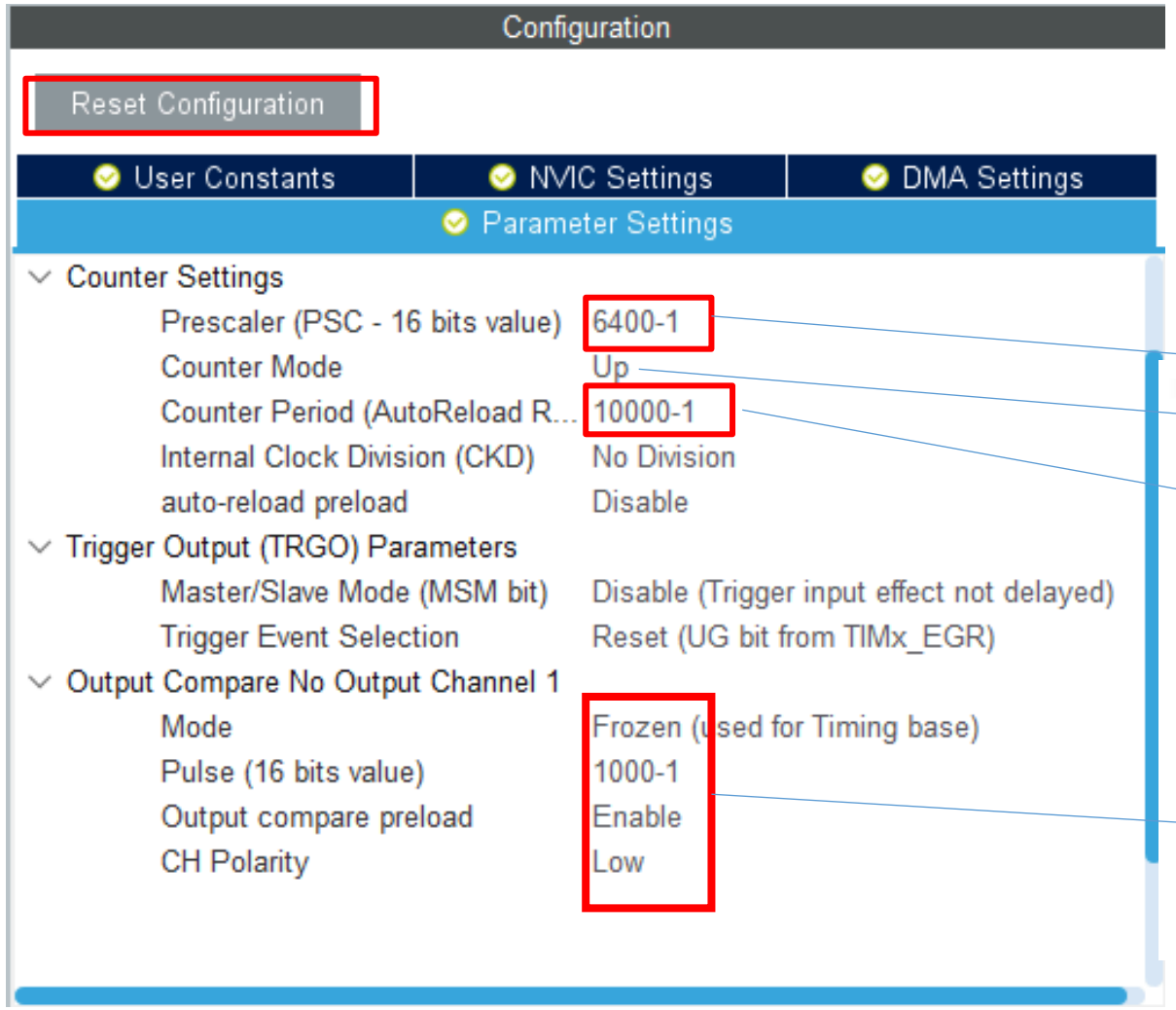
CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



Configuration에서 NVIC Settings 탭을 선택하고 TIM3 global interrupt를 Enabled 체크함

8.6 타이머 CubeMX과제

CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



출력 : 6장 과제와 동일(PC 0~7)

Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

a) LED의 ON
- TimHandler 구조체 변수의 TimHandler.Init.Prescaler = 6399 (Nucleo-F429 확장보드는 8999), TimHandler.Init.Period = 9999 로 설정하고 업 카운터 모드로 동작을 한다.
- 그러면 카운터는 0~9999 사이를 업 카운팅을 하게되고, 1초 마다 UI(업데이트 인터럽트)가 발생한다.

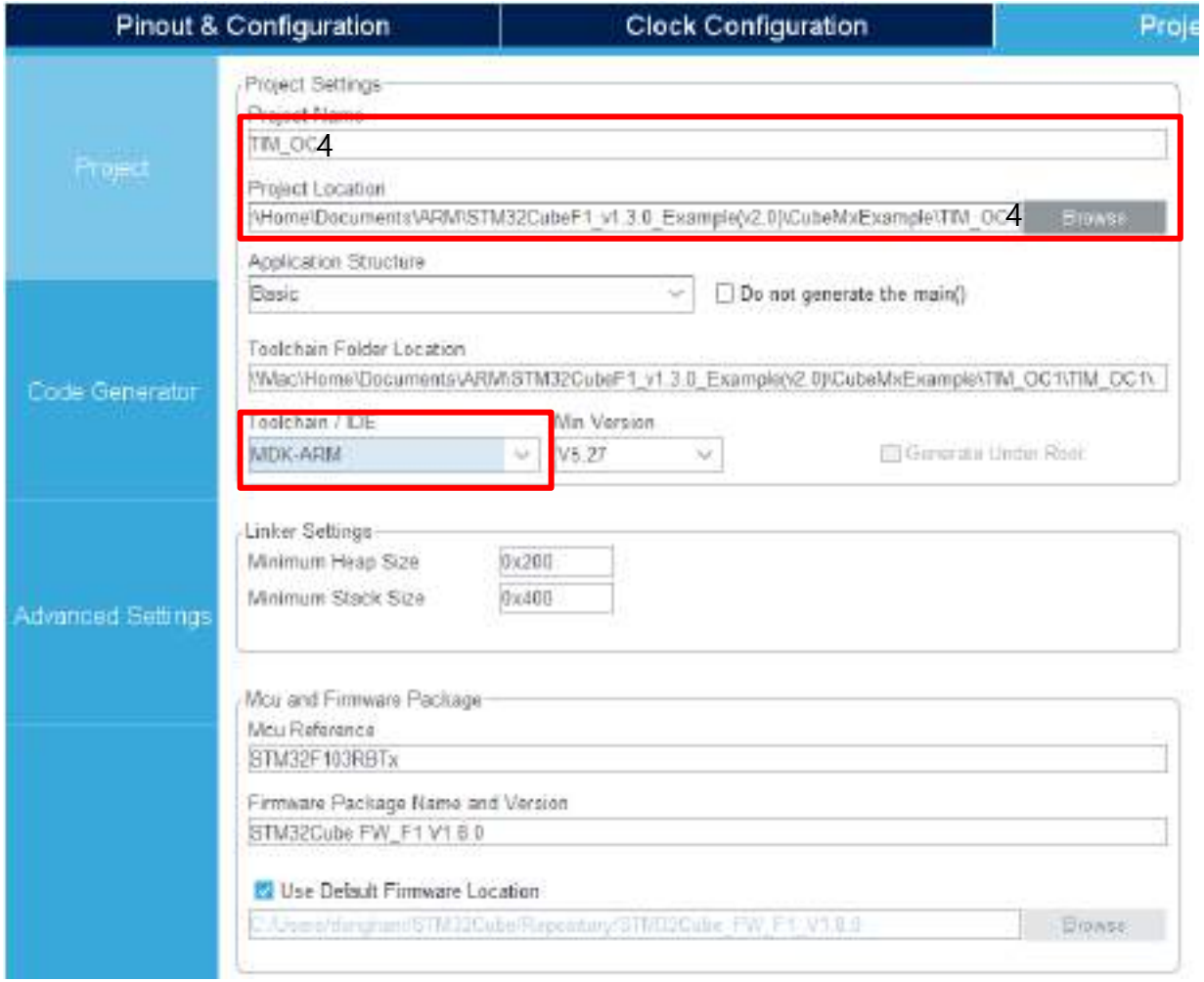
(앞 슬라이드)

- UI가 발생하면 UI 콜백함수인 HAL_TIM_PeriodElapsedCallback()가 호출되며, 이 함수에서 LED를 On 시켜준다.

b) LED 의 OFF
- OC(출력비교) 모드에서 TIM_OCInit 구조체 변수의 "TIM_OCInit.Pulse의 설정값 = 업 카운터의 카운팅 값"이 될때 출력 비교 인터럽트(CC2I)가 발생한다.
- 그러면 OC 인터럽트 콜백함수인 HAL_TIM_OC_DelayElapsedCallback()가 호출되며, 이 함수에서 LED를 Off 시켜준다.

8.6 타이머 CubeMX과제

CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



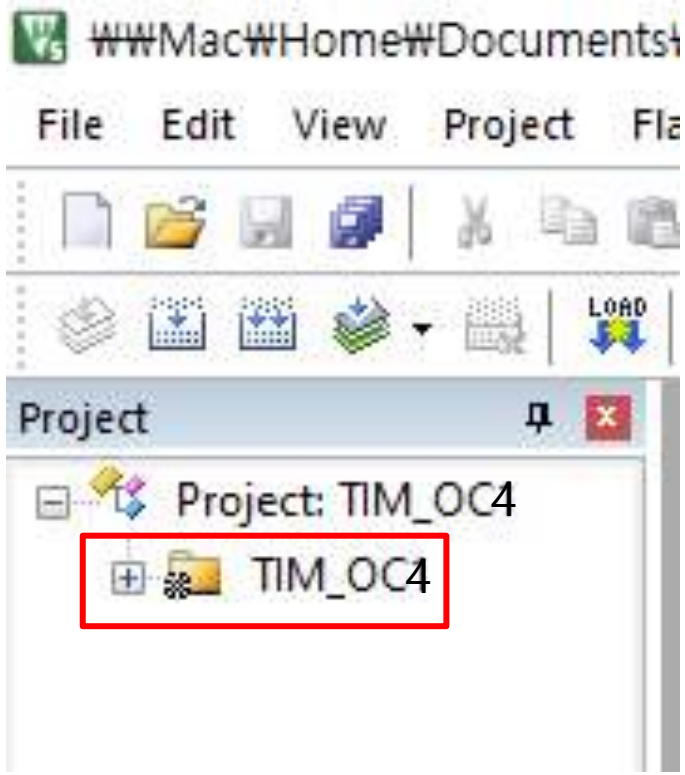
Project Manager 탭을 누르면, 왼쪽의 그림처럼 project 생성 단계가 되며, project name과 location을 정해줌.

Toolchain / IDE는 꼭 MDK-ARM을 선택

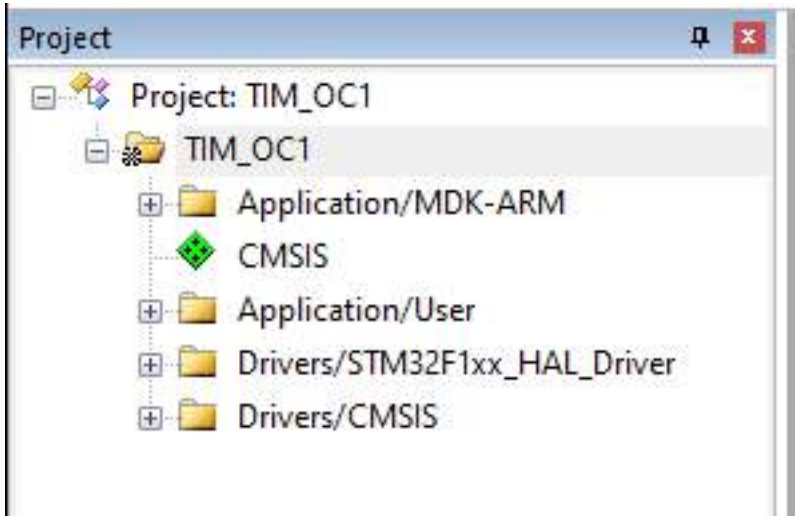
마지막으로 GENERATE CODE 탭을 누르면, 코드가 생성되면서 MDK uVision이 실행됨

8.6 타이머 CubeMX과제

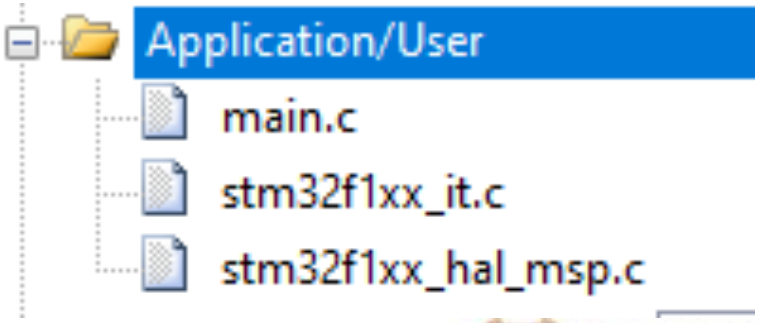
CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



MDK uVision에서 왼쪽의 Project 창에 CubeMX에 만든 project인 TIM_OC4이 보이고 있으며, + 를 눌러서 펼치면



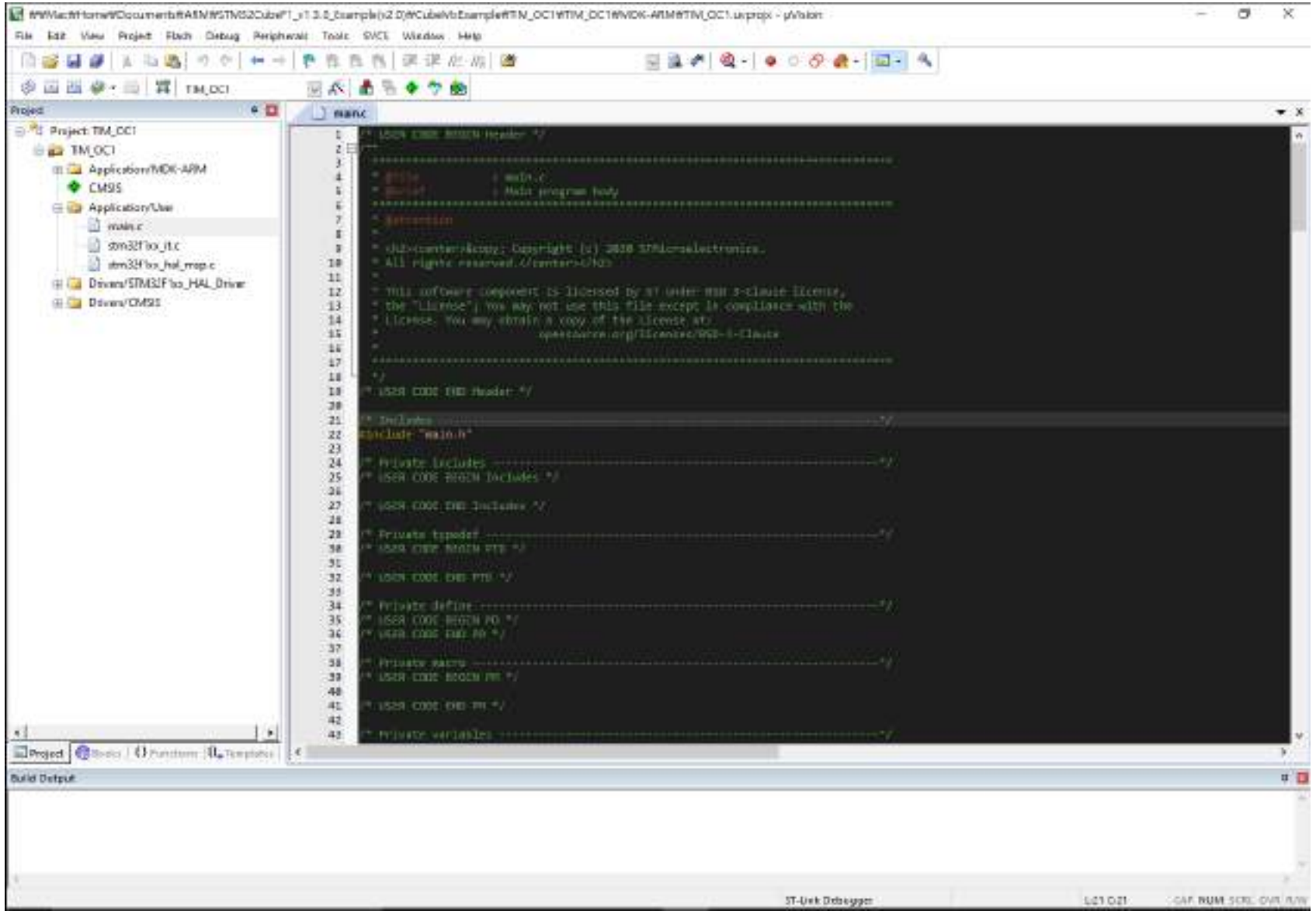
오른쪽 그림의 Application/User를 펼치면, main.c 가 보임. 이를 더블클릭함



KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

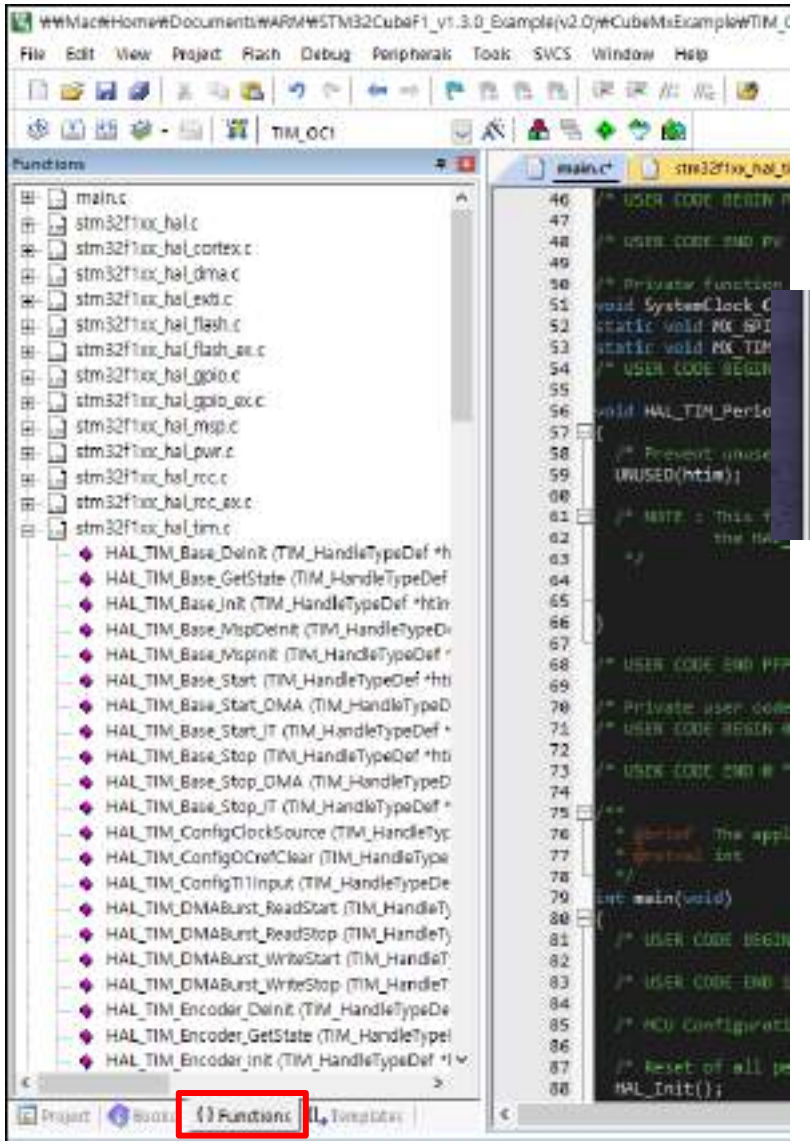
CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



그림의 오른쪽 editor window에 CubeMX가 자동으로 생성해준 기본 코드가 보이며, 이를 수정함

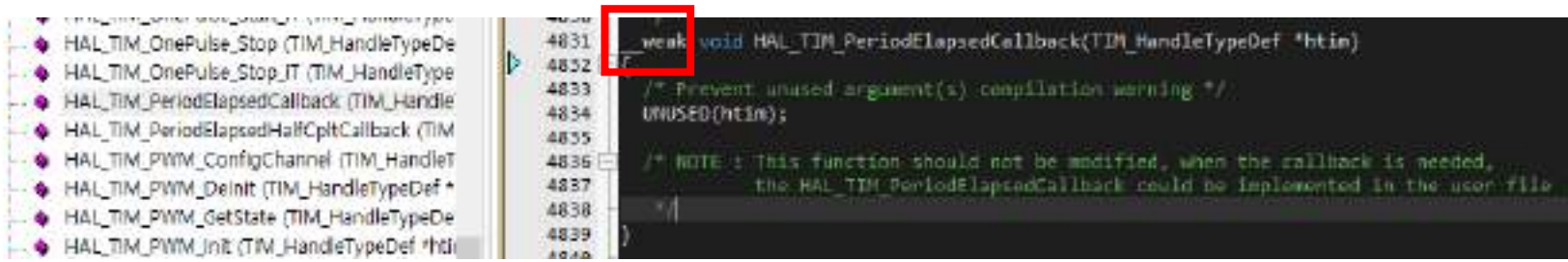
8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



→ HAL 함수들은 MDK uVision의 왼쪽 윈도우에서 {} Function 탭을 열고 해당하는 c 라이브러리를 열면 찾을 수 있음

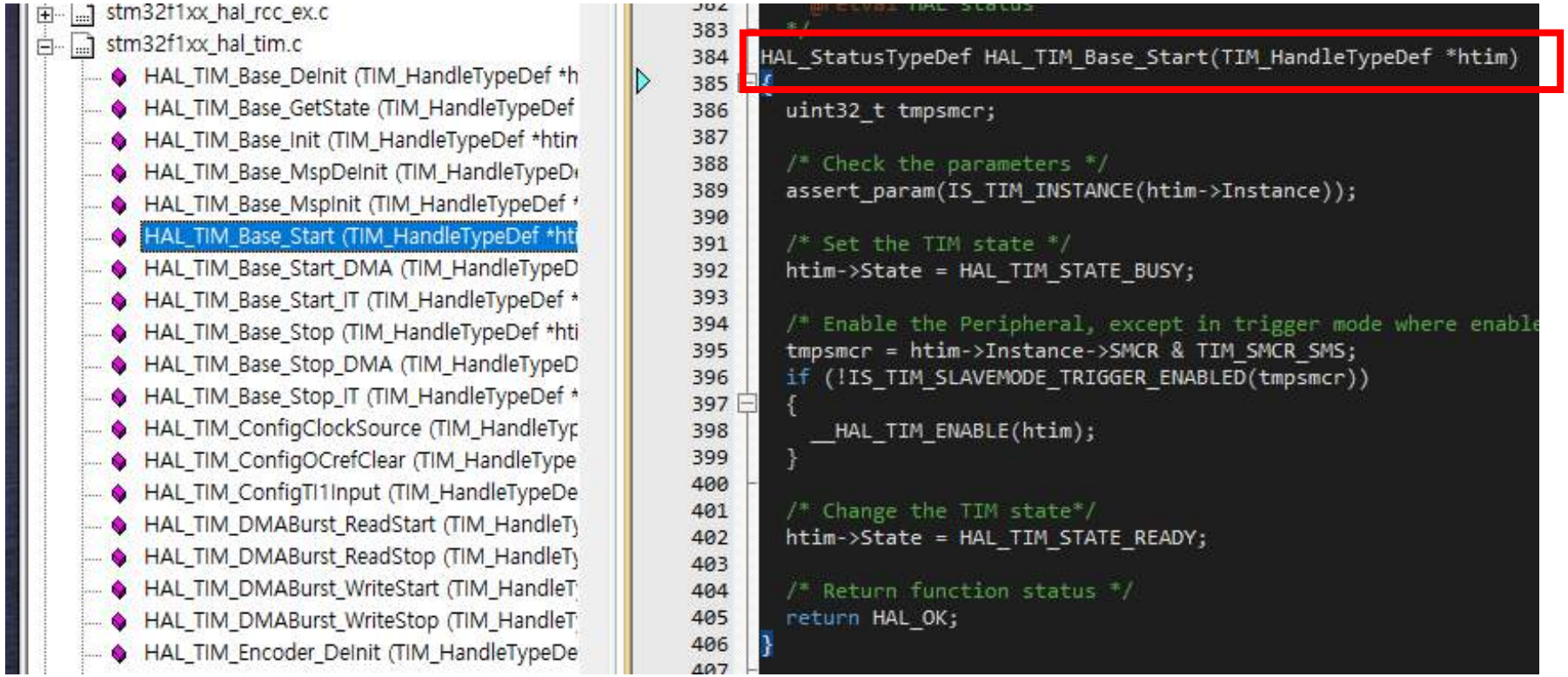
→ 예를 들어 앞 슬라이드의 HAL_TIM_PeriodElapsedCallback() 함수는 아래와 같이 찾을 수 있고 __weak 부분을 제외하고 copy하면 됨



KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.4 업 카운터와 OC모드(출력 펄스 폭 변경):



→ c 라이브러리에 있는HAL 함수들 중 이름 앞에 __weak 가 없는 함수들은 main.c에 따로 저장할 필요없이 호출해서 사용하면 됨 (주의할 점)

8.6 타이머 (인터럽트)CubeMX과제

```
/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */
TIM_OC_InitTypeDef  TIM_OCInit;
/* USER CODE END PV */
```

→ TIM_HandleTypeDef htim3; 부분은 CubeMX에서 자동생성한 부분이며, 추가로 TIM_OC_InitTypeDef TIM_OCInit; 선언했음

8.6 타이머 (인터럽트)CubeMX과제

main.c | stm32f1xx_hal_tim.c | stm32f1xx_hal_gpio.h | stm32f1xx_it.c | stm32f1xx_hal_gpio.c

```
50  /* Private function prototypes -----------------------------------------------*/
51  void SystemClock_Config(void);
52  static void MX_GPIO_Init(void);
53  static void MX_TIM3_Init(void);
54  /* USER CODE BEGIN PFP */
55
56  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
57  {
58    /* Prevent unused argument(s) compilation warning */
59    UNUSED(htim);
60
61    /* NOTE : This function should not be modified, when the callback is needed,
62              the HAL_TIM_PeriodElapsedCallback could be implemented in the user file
63     */
64    if (htim->Instance == TIM3)
65    {
66        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
67                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_SET);
68    }
69
70
71  }
72
73  void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
74  {
75    /* Prevent unused argument(s) compilation warning */
76    UNUSED(htim);
77
78    /* NOTE : This function should not be modified, when the callback is needed,
79              the HAL_TIM_OC_DelayElapsedCallback could be implemented in the user file
80     */
81    if (htim->Instance == TIM3)
82    {
83        HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
84                          GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_RESET);
85    }
86
87  }
```

→ TIM3의 PeriodElapsedCallback() 함수는 1초에 한번씩 발생하고, 본 예제에서는 이때 LED를 켬

→ TIM3의 Output Compare 인터럽트는 CNT가 0에서 시작해서 1000번째 될 OC_DelayElapsedCallback() 함수를 실행해서 LED를 끔

→ 최종 코드(교과서의 예제 8.4코드와 CubeMX의 코드)를 비교해보세요

8.6 타이머 (인터럽트)CubeMX과제

```
void HAL_TIM_OC_DelayElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(htim);

  /* NOTE : This function should not be modified, when the callback is needed,
            the HAL_TIM_OC_DelayElapsedCallback could be implemented in the user file
   */
  if (htim->Instance == TIM3)
  {
      HAL_GPIO_WritePin(GPIOC, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 |
                        GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7, GPIO_PIN_RESET);
  }

}

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  /* Prevent unused argument(s) compilation warning */
  UNUSED(GPIO_Pin);
  /* NOTE: This function Should not be modified, when the callback is needed,
           the HAL_GPIO_EXTI_Callback could be implemented in the user file
   */
  if(GPIO_Pin == GPIO_PIN_8) TIM_OCInit.Pulse = 999;
  if(GPIO_Pin == GPIO_PIN_4) TIM_OCInit.Pulse = 2999;
  if(GPIO_Pin == GPIO_PIN_5) TIM_OCInit.Pulse = 4999;
  if(GPIO_Pin == GPIO_PIN_10) TIM_OCInit.Pulse = 9999;

  HAL_TIM_OC_ConfigChannel(&htim3, &TIM_OCInit, TIM_CHANNEL_1);
  HAL_TIM_OC_Start_IT(&htim3,  TIM_CHANNEL_1);

}
```

→ 앞 슬라이드 부분 뒤에 HAL_GPIO_Callback() 함수 부분만 추가됨

→ HAL_GPIO_EXTI_Callback()함수는 버튼을 누를때 불리는 외부인터럽트 콜백함수임. 여기서 눌린 버튼이 몇번 버튼인지 OC의 폭(TIM_OCInit.Pulse)을 변화시킴

→ 마지막으로 변화된 OC의 폭을 적용시킴

→ 최종 코드(교과서의 예제 8.4코드와 CubeMX의 코드)를 비교해보세요

KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

main() 함수 부분은 8.3 예제와 동일함

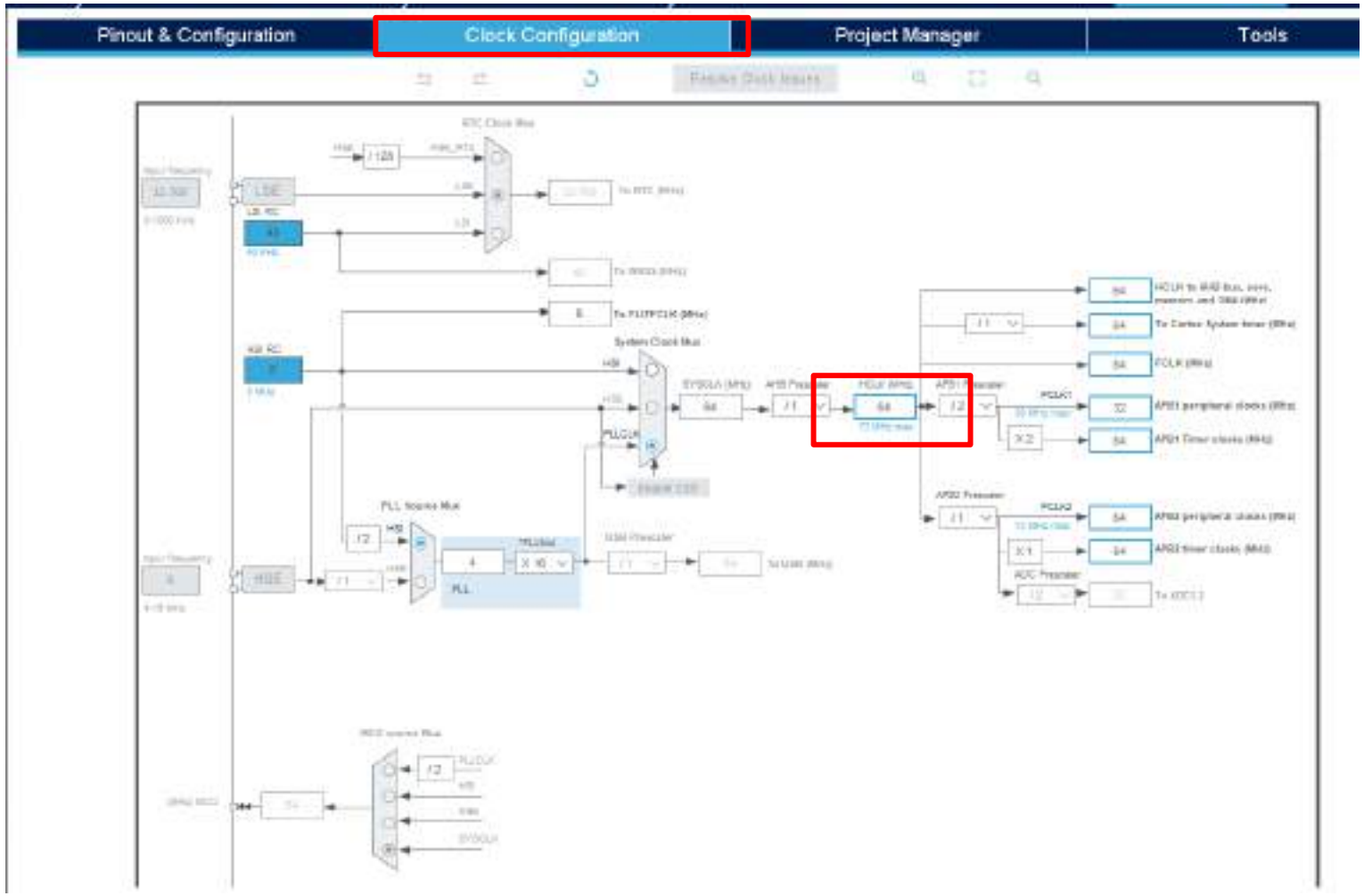
CubeMX가 자동생성

직접 코딩

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------

  /* Reset of all peripherals, Initializes the F
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  HAL_TIM_OC_Start_IT(&htim3, TIM_CHANNEL_1);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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

8.6 타이머 CubeMX과제

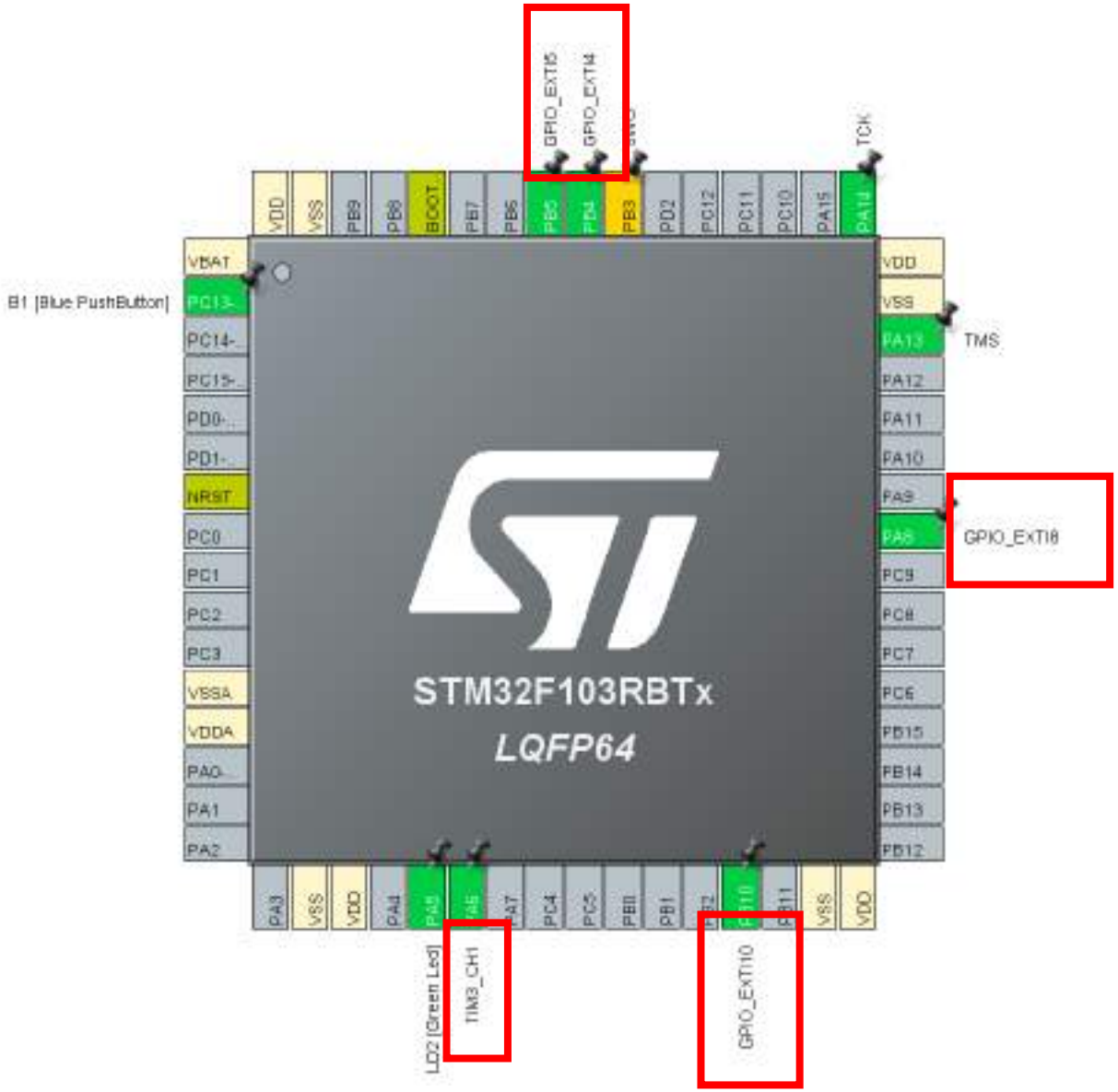
CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

8.6 타이머 CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



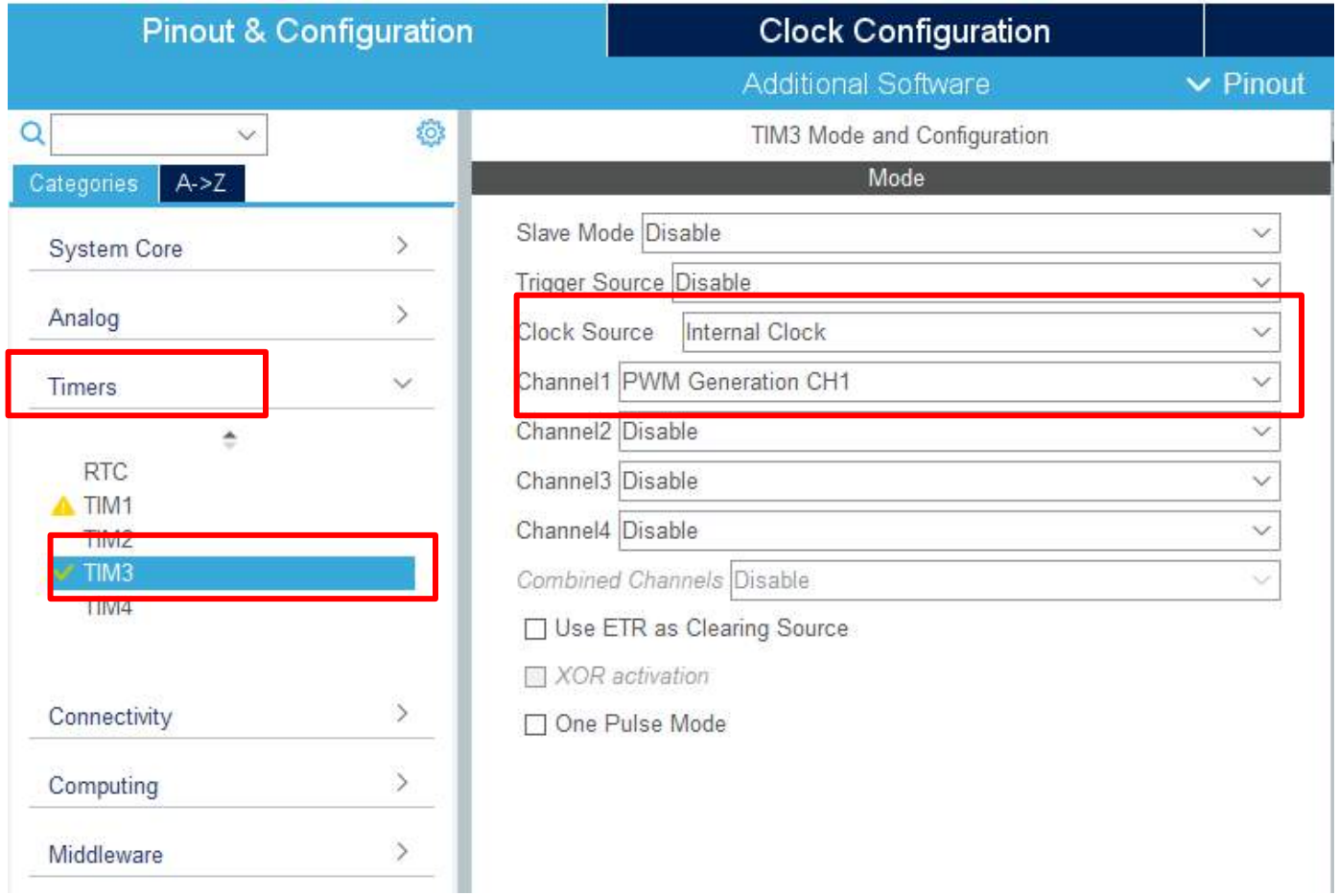
Pinout & Configuration 탭에서

GPIO 입력핀: PA8, PB10, PB4, PB5 를 EXTI 로 설정

PA6를 TIM3_CH1으로 설정 → 여기서 PWM이 나옴

8.6 타이머 CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):

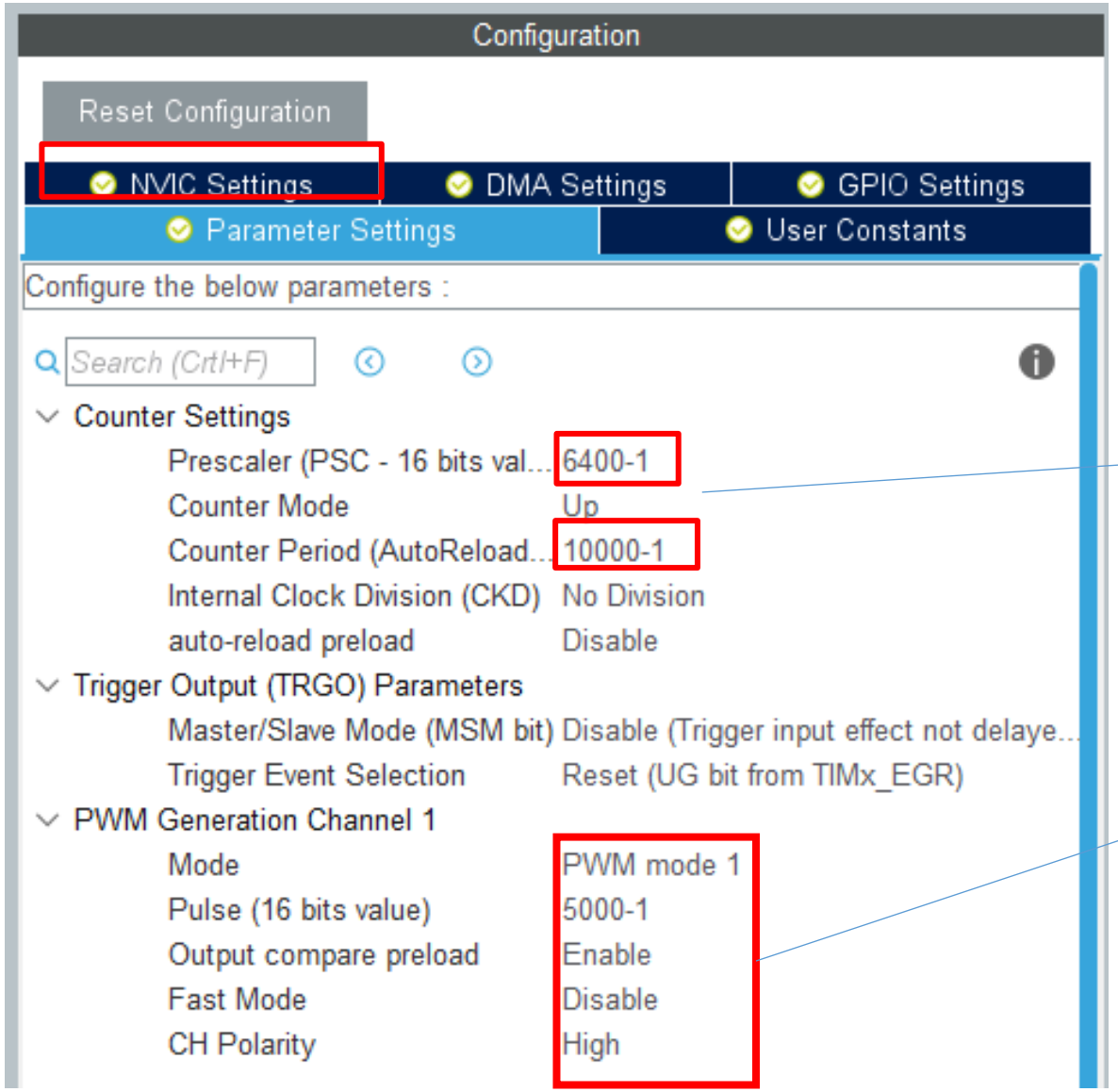


TIM3의 Channel1을 PWM Generation CH1으로 모드 설정

Clock Source는 Internal Clock으로 설정

8.6 타이머 CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



Clock configuration 탭에서 Main Clk를 설정 및 확인 → 64MHz

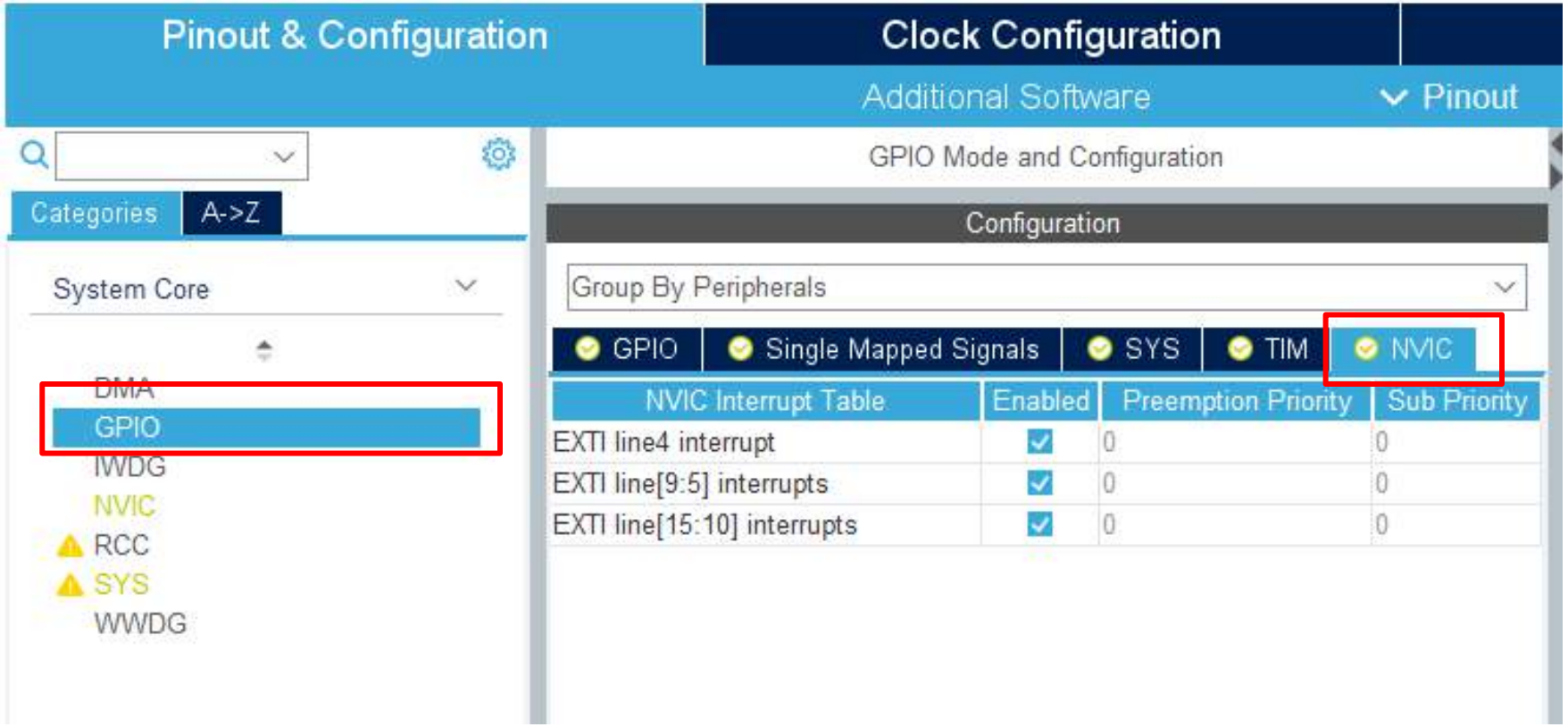
Prescaler 6399로 10KHz로 스케일링하고, ARR을 9999으로 세팅해서 PWM 주기는 1초(1Hz)로 만듬

여러분들이 이 값들을 조절해서 다양한 PWM 파형을 만들어봐요.

다른 예제 들과 다른 부분: PWM 세팅이고, 초기값으로 4999를 정함

8.6 타이머 CubeMX과제

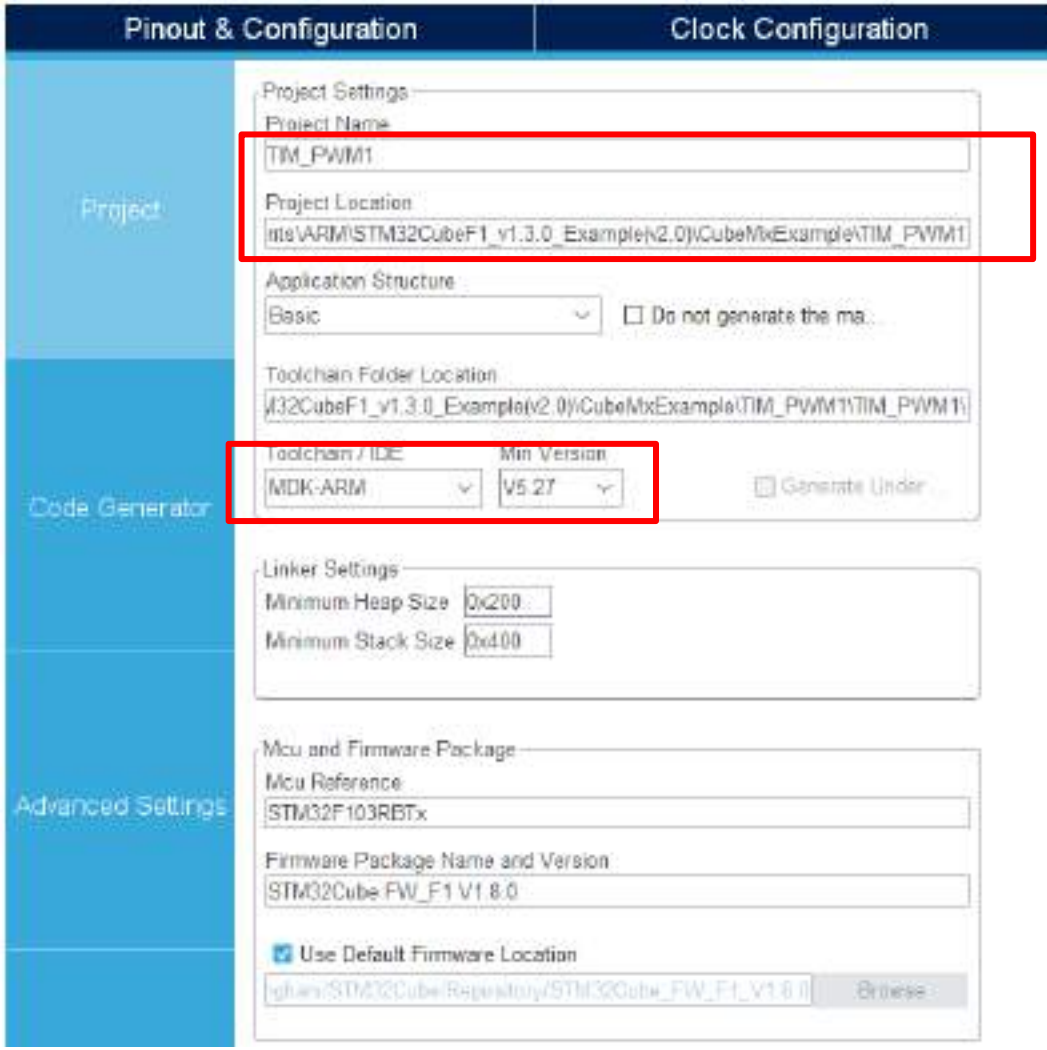
CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



System Core의 GPIO에 있는 Configuration에서 NVIC Settings 탭을 선택하고 EXTI interrupt들을 모두 Enabled 체크함

8.6 타이머 CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



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

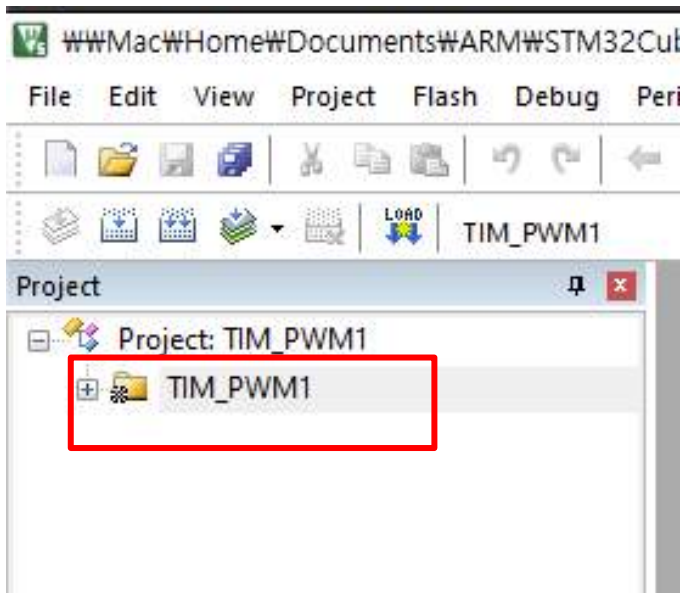
Toolchain / IDE는 꼭 MDK-ARM을 선택

마지막으로 GENERATE CODE 탭을 누르면, 코드가 생성되면서 MDK uVision이 실행됨

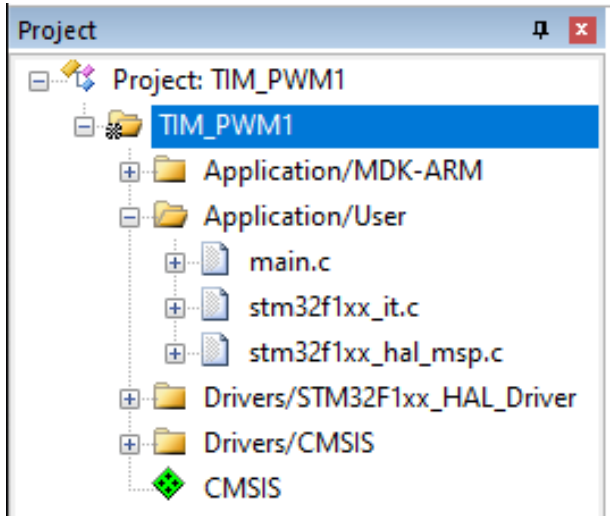
KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

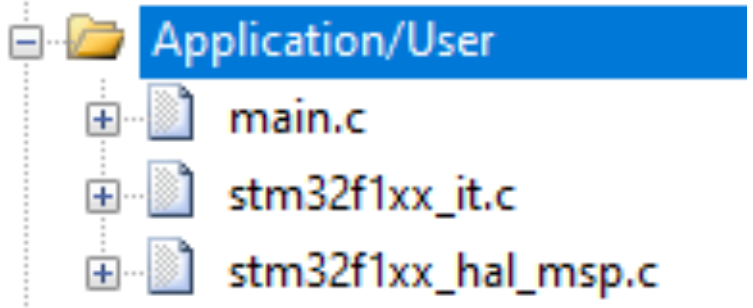
CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



MDK uVision에서 왼쪽의 Project 창에 CubeMX에 만든 project인 TIM_PWM1이 보이고 있으며, + 를 눌러서 펼치면



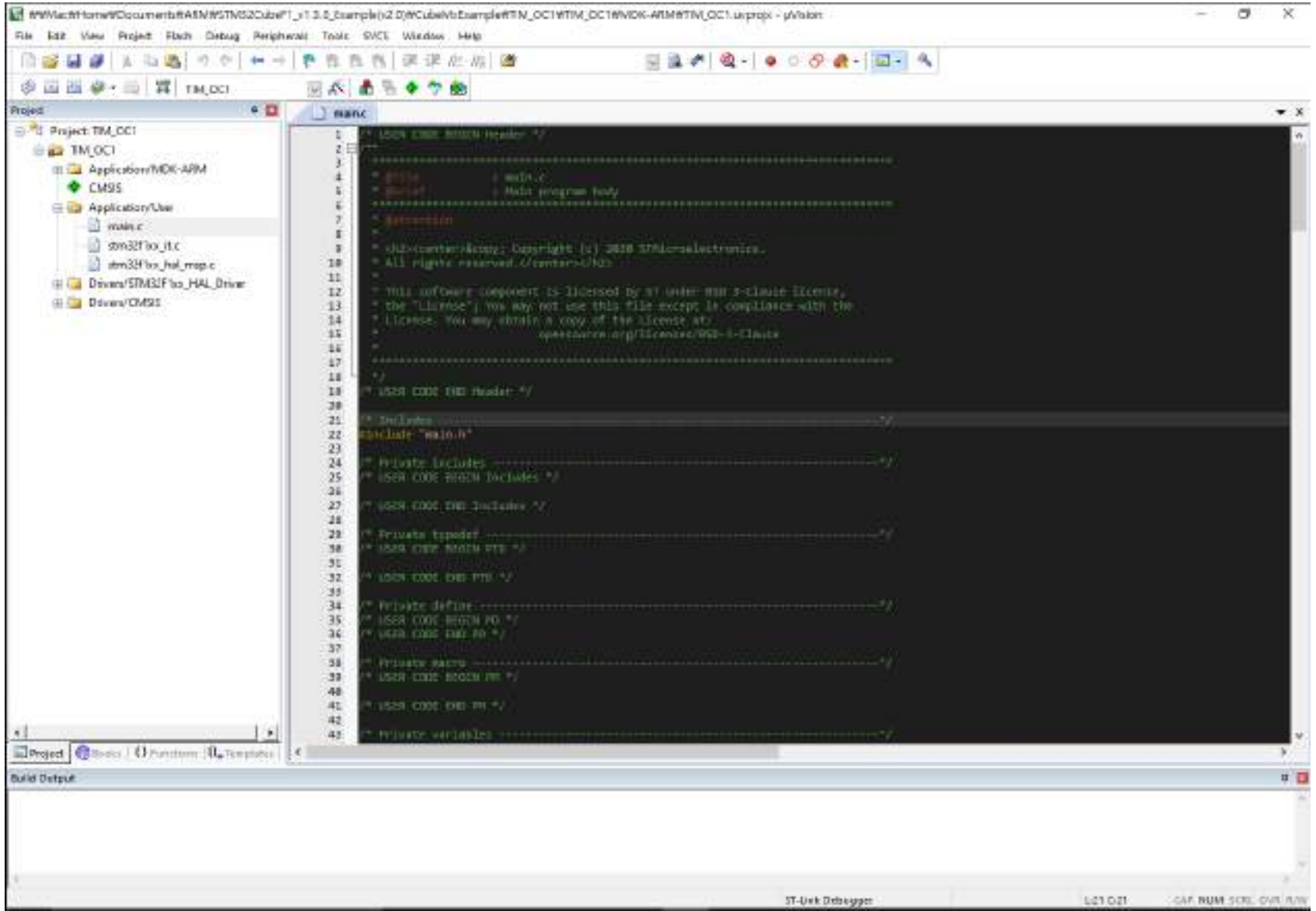
오른쪽 그림의 Application/User를 펼치면, main.c 가 보임. 이를 더블클릭함



KYUNG HEE UNIVERSITY

8.6 타이머 CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):

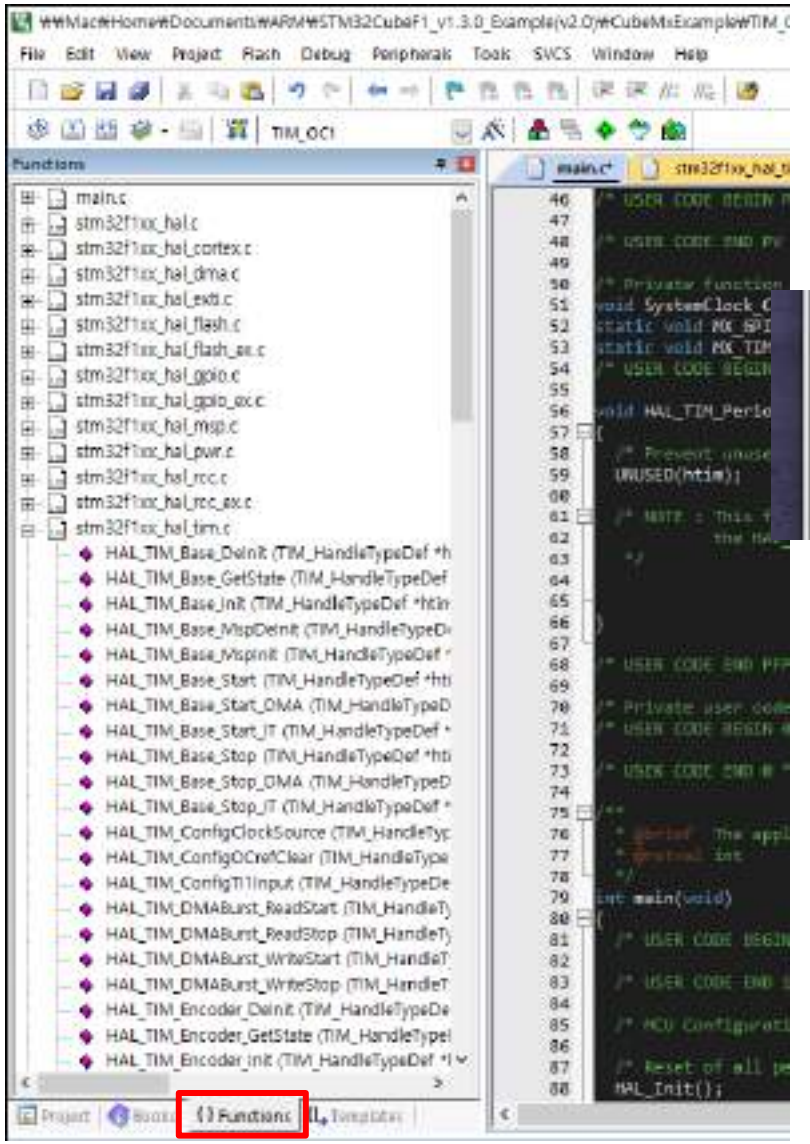


그림의 오른쪽 editor window에 CubeMX가 자동으로 생성해준 기본 코드가 보이며, 이를 수정함

KYUNG HEE UNIVERSITY

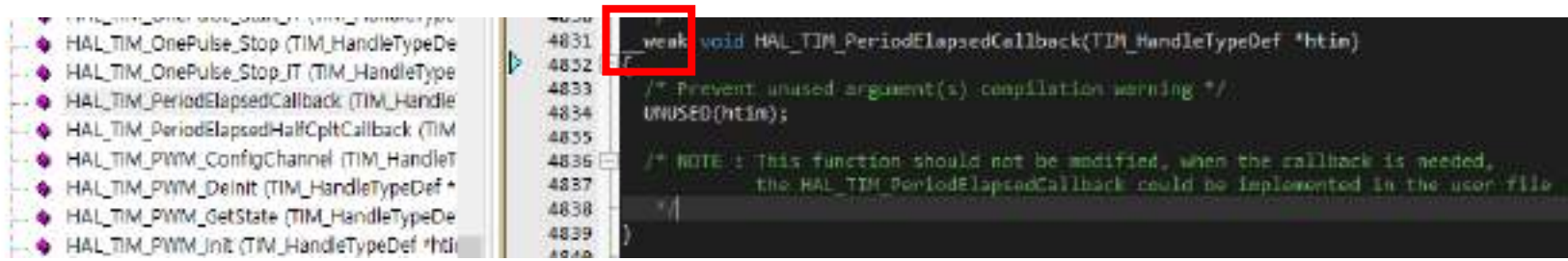
8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



→ HAL 함수들은 MDK uVision의 왼쪽 윈도우에서 {} Function 탭을 열고 해당하는 c 라이브러리를 열면 찾을 수 있음

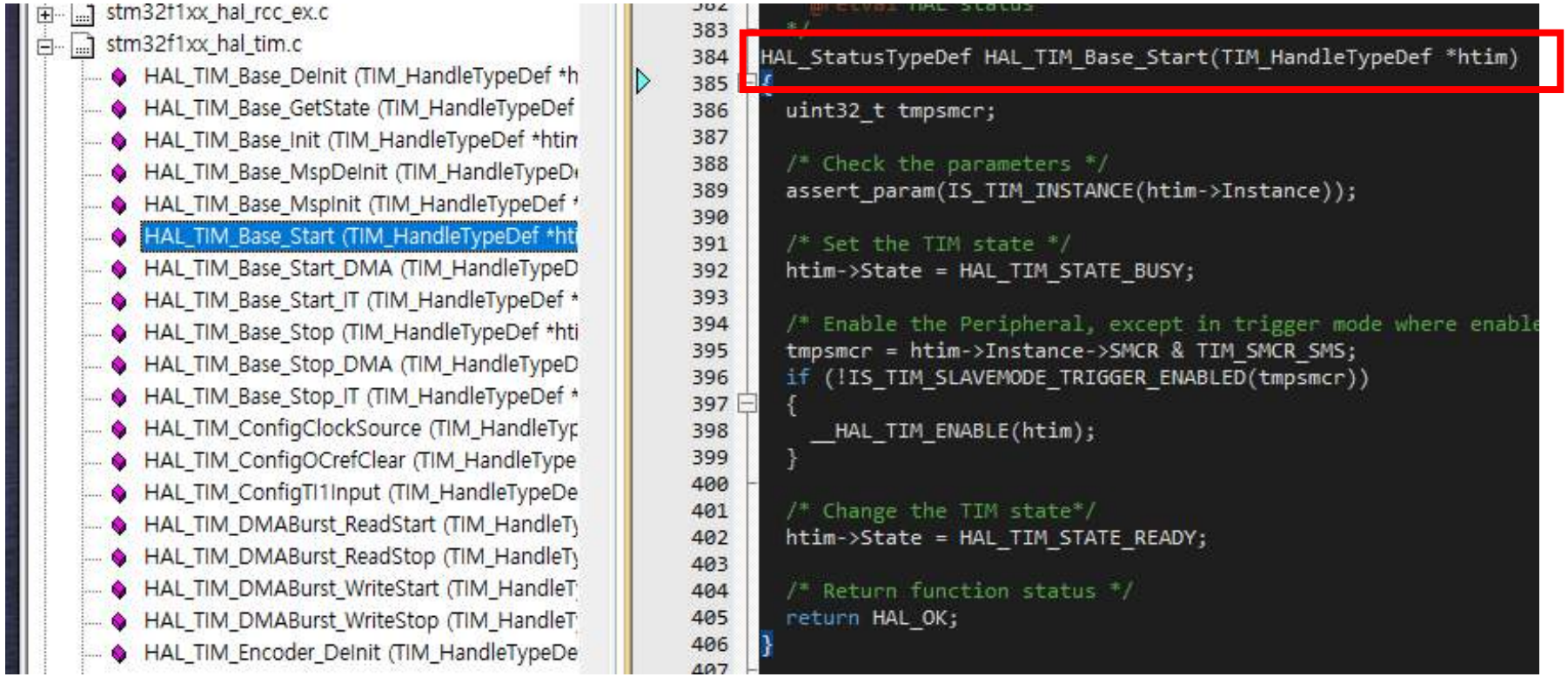
→ 예를 들어 앞 슬라이드의 HAL_TIM_PeriodElapsedCallback() 함수는 아래와 같이 찾을 수 있고 __weak 부분을 제외하고 copy하면 됨



KYUNG HEE UNIVERSITY

8.6 타이머 (인터럽트)CubeMX과제

CubeMX로 예제 8.6 PWM 출력 모드를 이용한 LED의 밝기 제어(PWM 출력 핀을 직접 이용):



→ c 라이브러리에 있는HAL 함수들 중 이름 앞에 __weak 가 없는 함수들은 main.c에 따로 저장할 필요없이 호출해서 사용하면 됨 (주의할 점)

8.6 타이머 (인터럽트)CubeMX과제

```
/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim3;

/* USER CODE BEGIN PV */
TIM_OC_InitTypeDef  TIM_OCInit;
/* USER CODE END PV */
```

→ TIM_HandleTypeDef htim3; 부분은 CubeMX에서 자동생성한 부분이며, 추가로 TIM_OC_InitTypeDef TIM_OCInit; 선언했음

8.6 타이머 (인터럽트)CubeMX과제

main.c

```
49
50  /* Private function prototypes -----------------------------------------------*/
51  void SystemClock_Config(void);
52  static void MX_GPIO_Init(void);
53  static void MX_TIM3_Init(void);
54  /* USER CODE BEGIN PFP */
55  void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
56  {
57    /* Prevent unused argument(s) compilation warning */
58    UNUSED(GPIO_Pin);
59    /* NOTE: This function Should not be modified, when the callback is needed,
60             the HAL_GPIO_EXTI_Callback could be implemented in the user file
61    */
62    TIM_OCInit.OCMode = TIM_OCMODE_PWM1;
63
64    if(GPIO_Pin == GPIO_PIN_8) TIM_OCInit.Pulse = 999;
65    if(GPIO_Pin == GPIO_PIN_4) TIM_OCInit.Pulse = 1999;
66    if(GPIO_Pin == GPIO_PIN_5) TIM_OCInit.Pulse = 2999;
67    if(GPIO_Pin == GPIO_PIN_10) TIM_OCInit.Pulse = 3999;
68
69    HAL_TIM_PWM_ConfigChannel(&htim3, &TIM_OCInit, TIM_CHANNEL_1);
70    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
71
72  }
73  /* USER CODE END PFP */
```

→ HAL_GPIO_EXTI_Callback()함수는 버튼을 누를때 불리는 외부인터럽트 콜백함수임. 여기서 눌린 버튼이 몇번 버튼인지 확인해서 PWM의 폭(TIM_OCInit.Pulse)을 변화시킴

→ 마지막으로 변화된 PWM의 폭을 적용시킴

→ 최종 코드(교과서의 예제 8.6코드와 CubeMX의 코드)를 비교해보세요

8.6 타이머 (인터럽트)CubeMX과제

main() 함수 부분은 다른 예제들과 비슷함

CubeMX가 자동생성

직접 코딩

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------

  /* Reset of all peripherals, Initializes the Flash
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

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