PWM using FTM (FlexTimer)

Hyeongrae Kim
Architecture and Compiler for Embedded System LAB.
School of Electronics Engineering, KNU, KOREA
2020-11-05

ACE Lab.

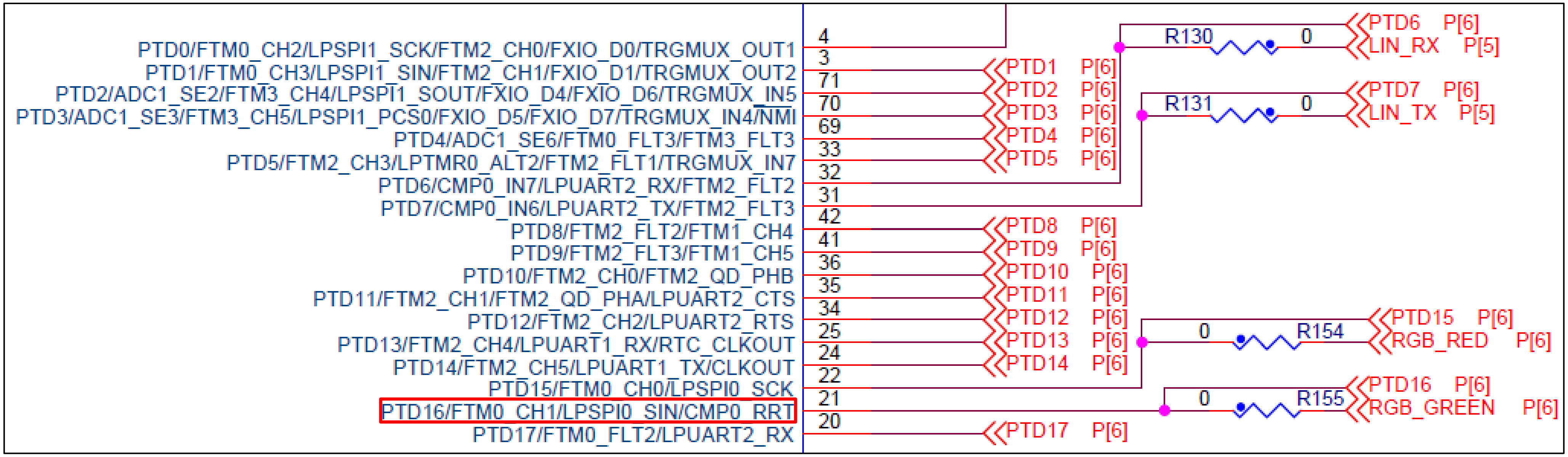
PWM Example

- PWM pulse width에 따른 LED 밝기 변화

1. 새로운 예제를 위한 프로젝트를 생성한다.
2. 원하는 동작을 위해 레지스터와 메모리에 직접 접근해서 값을 써야한다.
3. Board Schematic을 통해 해당 보드의 가변 저항에 대한 정보를 파악한다.
4. 사용할 ADC 모듈의 동작 원리를 파악하고 메모리 맵을 분석한다.
5. 보드 정보를 포함한 프로젝트를 생성했을 때, 해당 보드의 메모리 맵 정보가 헤더파일로 추가되기 때문에 이를 참고할 수 있다.
6. 분석 결과를 활용해 임베디드 프로그래밍을 한다.

PWM Example

1. Schematic 분석
   - ✓ 해당 보드의 Schematic을 확인했을 때, 여러 FTM0_CH이 있다.
   - ✓ FlexTimer Module (FTM)에서 PWM 신호를 생성할 수 있다.
   - ✓ 21번 Pin의 Alternative functionality인 FTM0_CH1을 사용하면 PWM 신호를 Green LED의 입력으로 사용할 수 있다.

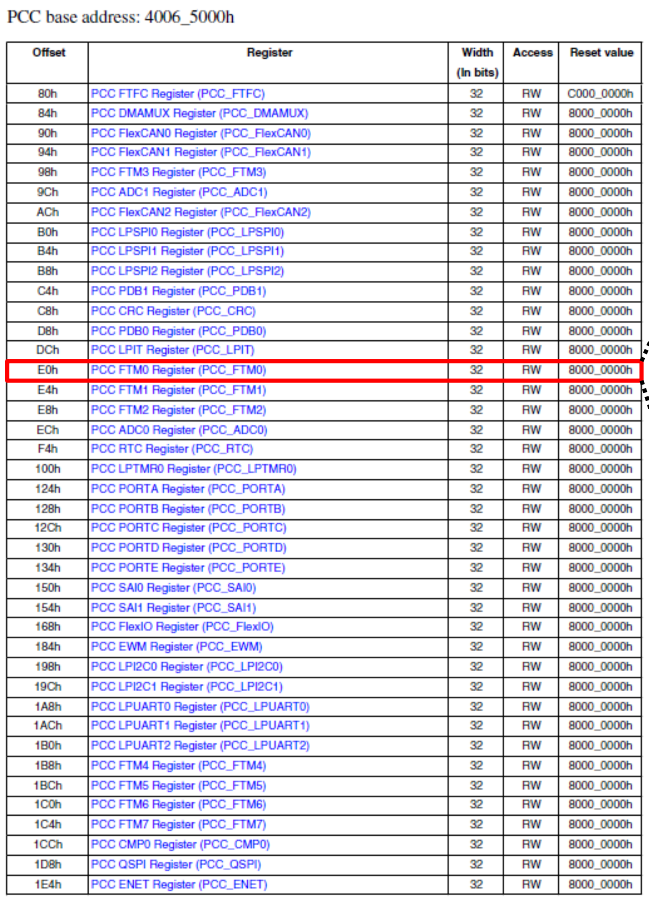


PTD0/FTM0_CH2/LPSPI1_SCK/FTM2_CH0/FXIO_D0/TRGMUX_OUT1 4
PTD1/FTM0_CH3/LPSPI1_SIN/FTM2_CH1/FXIO_D1/TRGMUX_OUT2 3
PTD2/ADC1_SE2/FTM3_CH4/LPSPI1_SOUT/FXIO_D4/FXIO_D6/TRGMUX_IN5 71
PTD3/ADC1_SE3/FTM3_CH5/LPSPI1_PCS0/FXIO_D5/FXIO_D7/TRGMUX_IN4/NMI 70
PTD4/ADC1_SE6/FTM0_FLT3/FTM3_FLT3 69
PTD5/FTM2_CH3/LPTMR0_ALT2/FTM2_FLT1/TRGMUX_IN7 33
PTD6/CMP0_IN7/LPUART2_RX/FTM2_FLT2 32
PTD7/CMP0_IN6/LPUART2_TX/FTM2_FLT3 31
PTD8/FTM2_FLT2/FTM1_CH4 42
PTD9/FTM2_FLT3/FTM1_CH5 41
PTD10/FTM2_CH0/FTM2_QD_PHB 36
PTD11/FTM2_CH1/FTM2_QD_PHA/LPUART2_CTS 35
PTD12/FTM2_CH2/LPUART2_RTS 34
PTD13/FTM2_CH4/LPUART1_RX/RTC_CLKOUT 25
PTD14/FTM2_CH5/LPUART1_TX/CLKOUT 24
PTD15/FTM0_CH0/LPSPI0_SCK 22
PTD16/FTM0_CH1/LPSPI0_SIN/CMP0_RRT 21
PTD17/FTM0_FLT2/LPUART2_RX 20
PTD1 P[6]
PTD2 P[6]
PTD3 P[6]
PTD4 P[6]
PTD5 P[6]
R130 0
PTD6 P[6]
LIN_RX P[5]
R131 0
PTD7 P[6]
LIN_TX P[5]
PTD8 P[6]
PTD9 P[6]
PTD10 P[6]
PTD11 P[6]
PTD12 P[6]
PTD13 P[6]
PTD14 P[6]
0 R154
PTD15 P[6]
RGB_RED P[6]
0 R155
PTD16 P[6]
RGB_GREEN P[6]
PTD17 P[6]

PWM Example

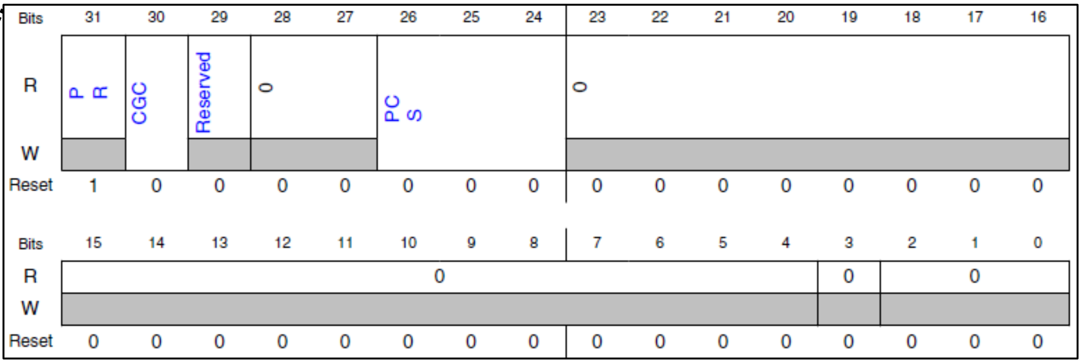
2. Data sheet 분석 : FTM0 클럭 설정

✓ PCC_FTM0 Register에서 CGC bit를 set하여 Clock enable 설정을 한다.



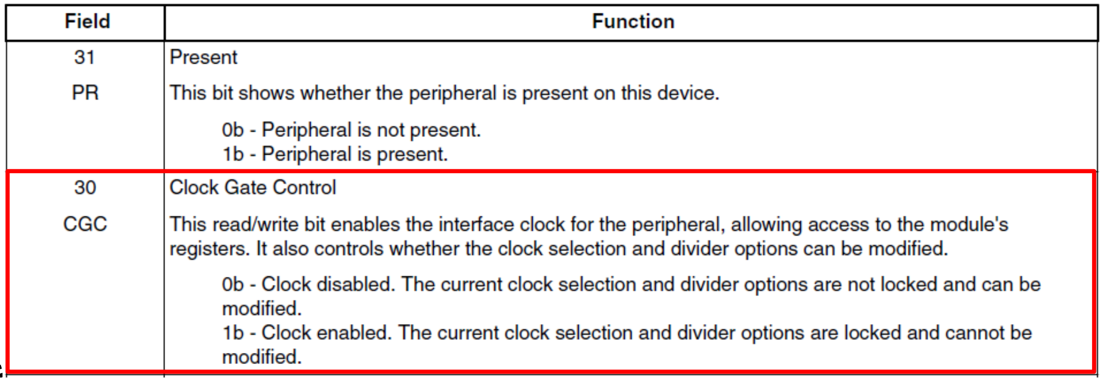
PCC base address: 4006_5000h

Offset	Register	Width (In bits)	Access	Reset value
80h	PCC FTFC Register (PCC_FTFC)	32	RW	C000_0000h
84h	PCC DMAMUX Register (PCC_DMAMUX)	32	RW	8000_0000h
90h	PCC FlexCAN0 Register (PCC_FlexCAN0)	32	RW	8000_0000h
94h	PCC FlexCAN1 Register (PCC_FlexCAN1)	32	RW	8000_0000h
98h	PCC FTM3 Register (PCC_FTM3)	32	RW	8000_0000h
9Ch	PCC ADC1 Register (PCC_ADC1)	32	RW	8000_0000h
ACh	PCC FlexCAN2 Register (PCC_FlexCAN2)	32	RW	8000_0000h
B0h	PCC LPSPI0 Register (PCC_LPSPI0)	32	RW	8000_0000h
B4h	PCC LPSPI1 Register (PCC_LPSPI1)	32	RW	8000_0000h
B8h	PCC LPSPI2 Register (PCC_LPSPI2)	32	RW	8000_0000h
C4h	PCC PDB1 Register (PCC_PDB1)	32	RW	8000_0000h
C8h	PCC CRC Register (PCC_CRC)	32	RW	8000_0000h
D8h	PCC PDB0 Register (PCC_PDB0)	32	RW	8000_0000h
DCh	PCC LPIT Register (PCC_LPIT)	32	RW	8000_0000h
E0h	PCC FTM0 Register (PCC_FTM0)	32	RW	8000_0000h
E4h	PCC FTM1 Register (PCC_FTM1)	32	RW	8000_0000h
E8h	PCC FTM2 Register (PCC_FTM2)	32	RW	8000_0000h
ECh	PCC ADC0 Register (PCC_ADC0)	32	RW	8000_0000h
F4h	PCC RTC Register (PCC_RTC)	32	RW	8000_0000h
100h	PCC LPTMR0 Register (PCC_LPTMR0)	32	RW	8000_0000h
124h	PCC PORTA Register (PCC_PORTA)	32	RW	8000_0000h
128h	PCC PORTB Register (PCC_PORTB)	32	RW	8000_0000h
12Ch	PCC PORTC Register (PCC_PORTC)	32	RW	8000_0000h
130h	PCC PORTD Register (PCC_PORTD)	32	RW	8000_0000h
134h	PCC PORTE Register (PCC_PORTE)	32	RW	8000_0000h
150h	PCC SAI0 Register (PCC_SAI0)	32	RW	8000_0000h
154h	PCC SAI1 Register (PCC_SAI1)	32	RW	8000_0000h
168h	PCC FlexIO Register (PCC_FlexIO)	32	RW	8000_0000h
184h	PCC EWM Register (PCC_EWM)	32	RW	8000_0000h
198h	PCC LPI2C0 Register (PCC_LPI2C0)	32	RW	8000_0000h
19Ch	PCC LPI2C1 Register (PCC_LPI2C1)	32	RW	8000_0000h
1A8h	PCC LPUART0 Register (PCC_LPUART0)	32	RW	8000_0000h
1ACh	PCC LPUART1 Register (PCC_LPUART1)	32	RW	8000_0000h
1B0h	PCC LPUART2 Register (PCC_LPUART2)	32	RW	8000_0000h
1B8h	PCC FTM4 Register (PCC_FTM4)	32	RW	8000_0000h
1BCh	PCC FTM5 Register (PCC_FTM5)	32	RW	8000_0000h
1C0h	PCC FTM6 Register (PCC_FTM6)	32	RW	8000_0000h
1C4h	PCC FTM7 Register (PCC_FTM7)	32	RW	8000_0000h
1CCh	PCC CMP0 Register (PCC_CMP0)	32	RW	8000_0000h
1D8h	PCC QSPI Register (PCC_QSPI)	32	RW	8000_0000h
1E4h	PCC ENET Register (PCC_ENET)	32	RW	8000_0000h



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PR	CGC	Reserved	0		PCS			0							
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												0	0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



Field	Function
31 PR	Present This bit shows whether the peripheral is present on this device. 0b - Peripheral is not present. 1b - Peripheral is present.
30 CGC	Clock Gate Control This read/write bit enables the interface clock for the peripheral, allowing access to the module's registers. It also controls whether the clock selection and divider options can be modified. 0b - Clock disabled. The current clock selection and divider options are not locked and can be modified. 1b - Clock enabled. The current clock selection and divider options are locked and cannot be modified.

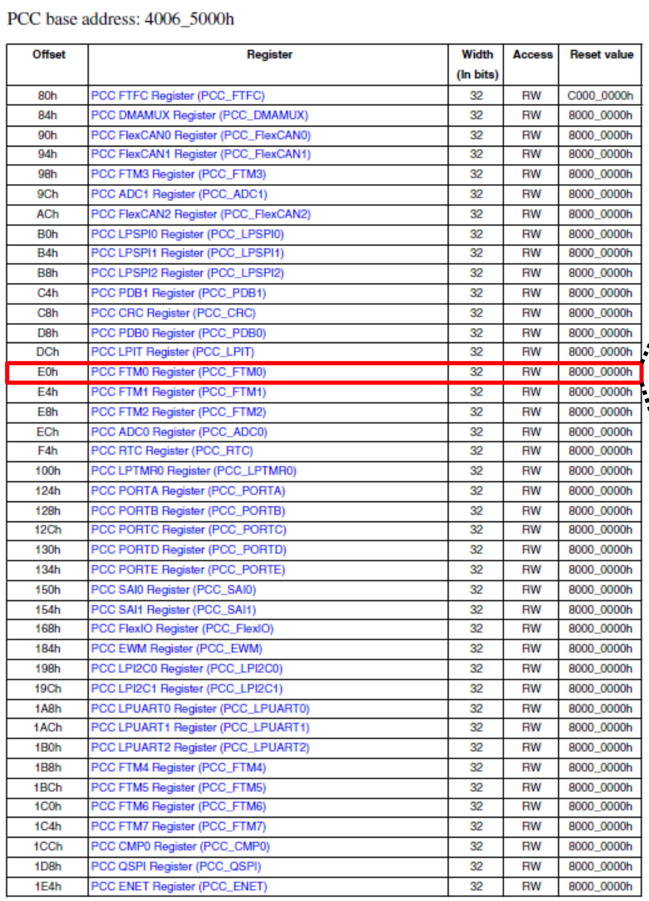
PCC_FTM0 Register 구조

ACE Lab.

PWM Example

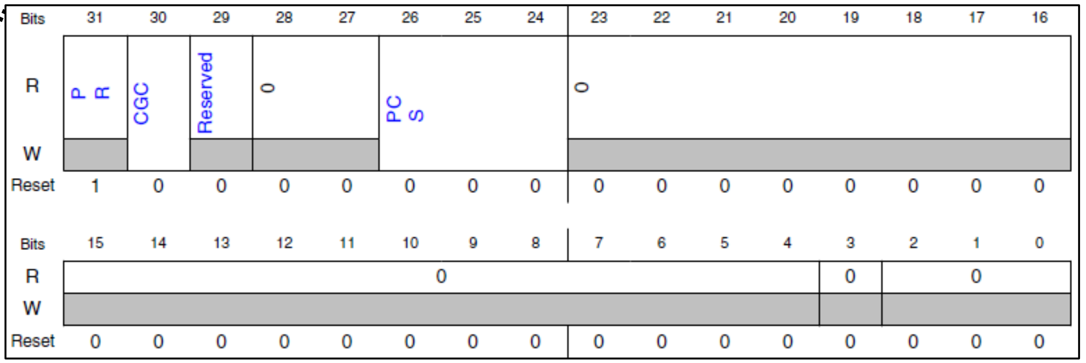
2. Data sheet 분석 : FTM0 클럭 설정

✓ PCC_FTM0 Register에서 PCS bit를 set하여 Clock source select 설정을 한다.



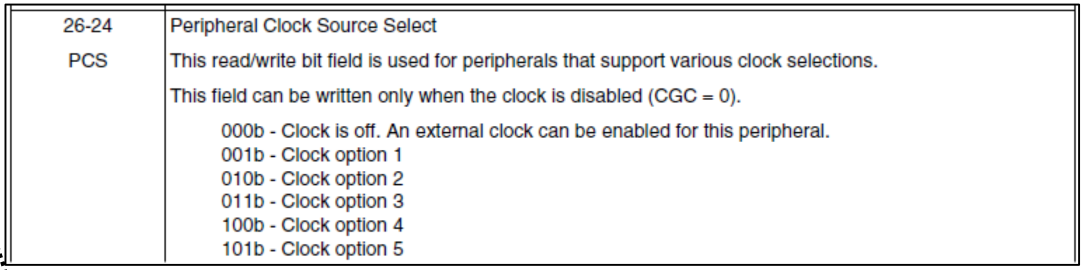
PCC base address: 4006_5000h

Offset	Register	Width (In bits)	Access	Reset value
80h	PCC FTFC Register (PCC_FTFC)	32	RW	C000_0000h
84h	PCC DMAMUX Register (PCC_DMAMUX)	32	RW	8000_0000h
90h	PCC FlexCAN0 Register (PCC_FlexCAN0)	32	RW	8000_0000h
94h	PCC FlexCAN1 Register (PCC_FlexCAN1)	32	RW	8000_0000h
98h	PCC FTM3 Register (PCC_FTM3)	32	RW	8000_0000h
9Ch	PCC ADC1 Register (PCC_ADC1)	32	RW	8000_0000h
ACh	PCC FlexCAN2 Register (PCC_FlexCAN2)	32	RW	8000_0000h
B0h	PCC LPSPI0 Register (PCC_LPSPI0)	32	RW	8000_0000h
B4h	PCC LPSPI1 Register (PCC_LPSPI1)	32	RW	8000_0000h
B8h	PCC LPSPI2 Register (PCC_LPSPI2)	32	RW	8000_0000h
C4h	PCC PDB1 Register (PCC_PDB1)	32	RW	8000_0000h
C8h	PCC CRC Register (PCC_CRC)	32	RW	8000_0000h
D8h	PCC PDB0 Register (PCC_PDB0)	32	RW	8000_0000h
DCh	PCC LPIT Register (PCC_LPIT)	32	RW	8000_0000h
E0h	PCC FTM0 Register (PCC_FTM0)	32	RW	8000_0000h
E4h	PCC FTM1 Register (PCC_FTM1)	32	RW	8000_0000h
E8h	PCC FTM2 Register (PCC_FTM2)	32	RW	8000_0000h
ECh	PCC ADC0 Register (PCC_ADC0)	32	RW	8000_0000h
F4h	PCC RTC Register (PCC_RTC)	32	RW	8000_0000h
100h	PCC LPTMR0 Register (PCC_LPTMR0)	32	RW	8000_0000h
124h	PCC PORTA Register (PCC_PORTA)	32	RW	8000_0000h
128h	PCC PORTB Register (PCC_PORTB)	32	RW	8000_0000h
12Ch	PCC PORTC Register (PCC_PORTC)	32	RW	8000_0000h
130h	PCC PORTD Register (PCC_PORTD)	32	RW	8000_0000h
134h	PCC PORTE Register (PCC_PORTE)	32	RW	8000_0000h
150h	PCC SAI0 Register (PCC_SAI0)	32	RW	8000_0000h
154h	PCC SAI1 Register (PCC_SAI1)	32	RW	8000_0000h
168h	PCC FlexIO Register (PCC_FlexIO)	32	RW	8000_0000h
184h	PCC EWM Register (PCC_EWM)	32	RW	8000_0000h
198h	PCC LPI2C0 Register (PCC_LPI2C0)	32	RW	8000_0000h
19Ch	PCC LPI2C1 Register (PCC_LPI2C1)	32	RW	8000_0000h
1A8h	PCC LPUART0 Register (PCC_LPUART0)	32	RW	8000_0000h
1ACh	PCC LPUART1 Register (PCC_LPUART1)	32	RW	8000_0000h
1B0h	PCC LPUART2 Register (PCC_LPUART2)	32	RW	8000_0000h
1B8h	PCC FTM4 Register (PCC_FTM4)	32	RW	8000_0000h
1BCh	PCC FTM5 Register (PCC_FTM5)	32	RW	8000_0000h
1C0h	PCC FTM6 Register (PCC_FTM6)	32	RW	8000_0000h
1C4h	PCC FTM7 Register (PCC_FTM7)	32	RW	8000_0000h
1CCh	PCC CMP0 Register (PCC_CMP0)	32	RW	8000_0000h
1D8h	PCC QSPI Register (PCC_QSPI)	32	RW	8000_0000h
1E4h	PCC ENET Register (PCC_ENET)	32	RW	8000_0000h



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PR	CGC	Reserved	0		PCS			0							
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0												0	0		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



26-24 PCS	Peripheral Clock Source Select This read/write bit field is used for peripherals that support various clock selections. This field can be written only when the clock is disabled (CGC = 0). 000b - Clock is off. An external clock can be enabled for this peripheral. 001b - Clock option 1 010b - Clock option 2 011b - Clock option 3 100b - Clock option 4 101b - Clock option 5

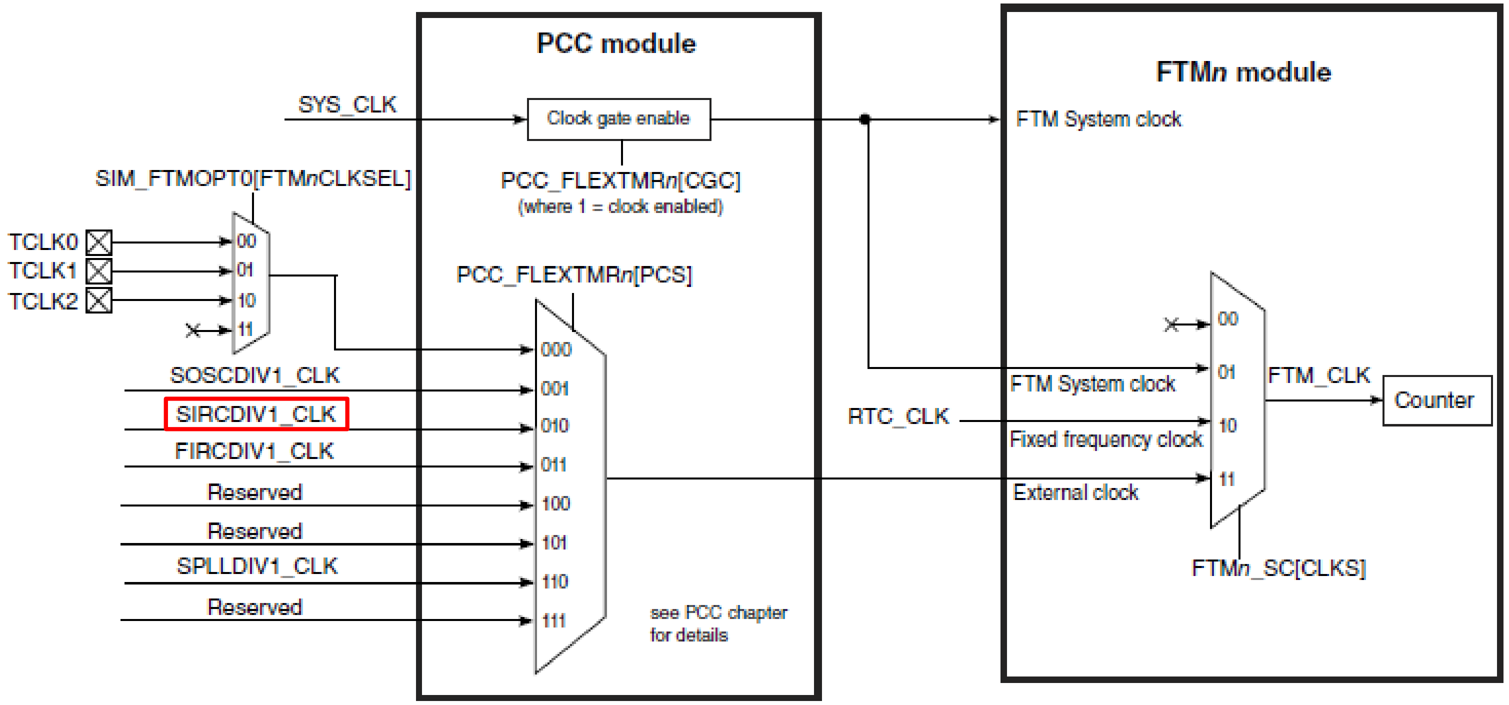
PCC_FTM0 Register 구조

ACE Lab.

PWM Example

2. Data sheet 분석 : FTM0 클럭 설정

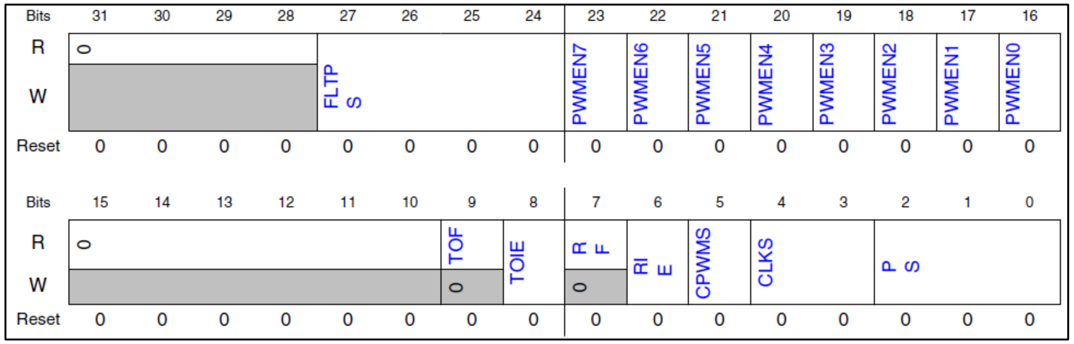
✓ 해당 FTM PWM 예제에서는 SIRCDIV1_CLK (PCS = 010, 8MHz)을 사용한다.



PWM Example

2. Data sheet 분석 : SC (Status and Control Register) 설정

- ✓ PWMENn 비트는 PWM enable bit이다.
- ✓ Green LED가 FTM0_CH1에 연결되어 있으므로 PWMEN1 비트를 set하면 된다.
- ✓ PS 비트에 따라 입력 클럭을 Prescale할 수 있다.
- ✓ 해당 예제에선 SIRCDIV1_CLK(8MHz)를 2로 나눠 4MHz 클럭으로 사용하므로 해당 비트는 1으로 세팅한다.

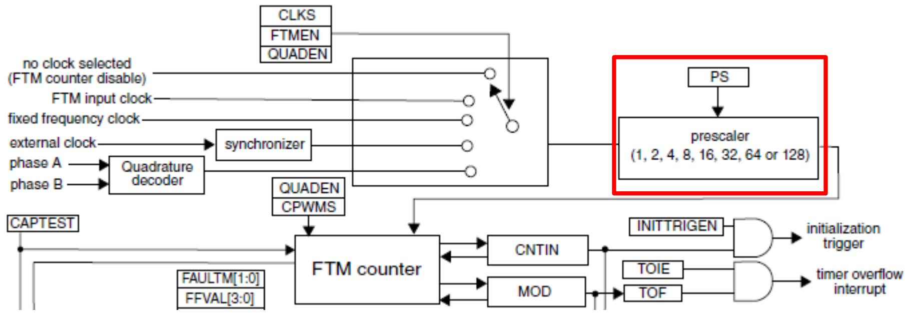


Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0				FLTPS				PWMEN7	PWMEN6	PWMEN5	PWMEN4	PWMEN3	PWMEN2	PWMEN1	PWMEN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0						TOF	TOIE	RF	RIE	CPWMS	CLKS		PS		
W							0		0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

23-16 PWMENn	Channel n PWM enable bit This bit enables the PWM channel output. This bit should be set to 0 (output disabled) when an input mode is used. 0b - Channel output port is disabled. 1b - Channel output port is enabled.

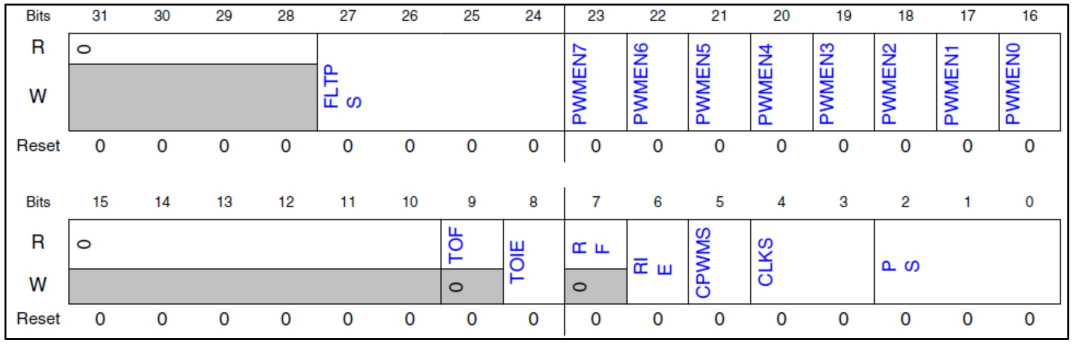
2-0 PS	Prescale Factor Selection Selects one of 8 division factors for the clock source selected by CLKS. The new prescaler factor affects the clock source on the next FTM input clock cycle after the new value is updated into the register bits. This field is write protected. It can be written only when MODE[WPDIS] = 1. 000b - Divide by 1 001b - Divide by 2 010b - Divide by 4 011b - Divide by 8 100b - Divide by 16 101b - Divide by 32 110b - Divide by 64 111b - Divide by 128



PWM Example

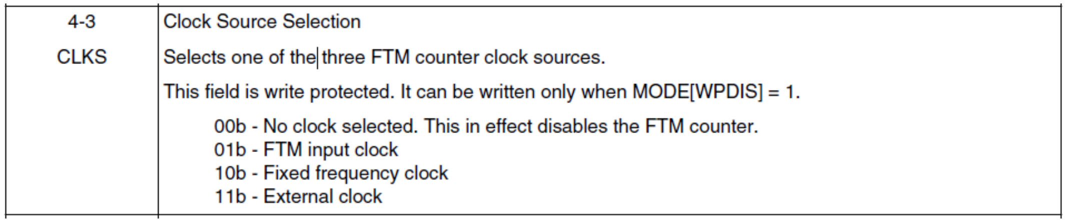
2. Data sheet 분석 : SC (Status and Control Register) 설정

- ✓ CLKS 비트를 설정함으로써 Counter 동작이 시작한다.
- ✓ 이 예제에서는 External clock을 사용하므로 3으로 세팅한다.



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0				FLTPS				PWMEN7	PWMEN6	PWMEN5	PWMEN4	PWMEN3	PWMEN2	PWMEN1	PWMEN0
W	▨	▨	▨	▨												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0						TOF	TOIE	RF	RIE	CPWMS	CLKS		PS		
W	▨	▨	▨	▨	▨	▨	0		0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



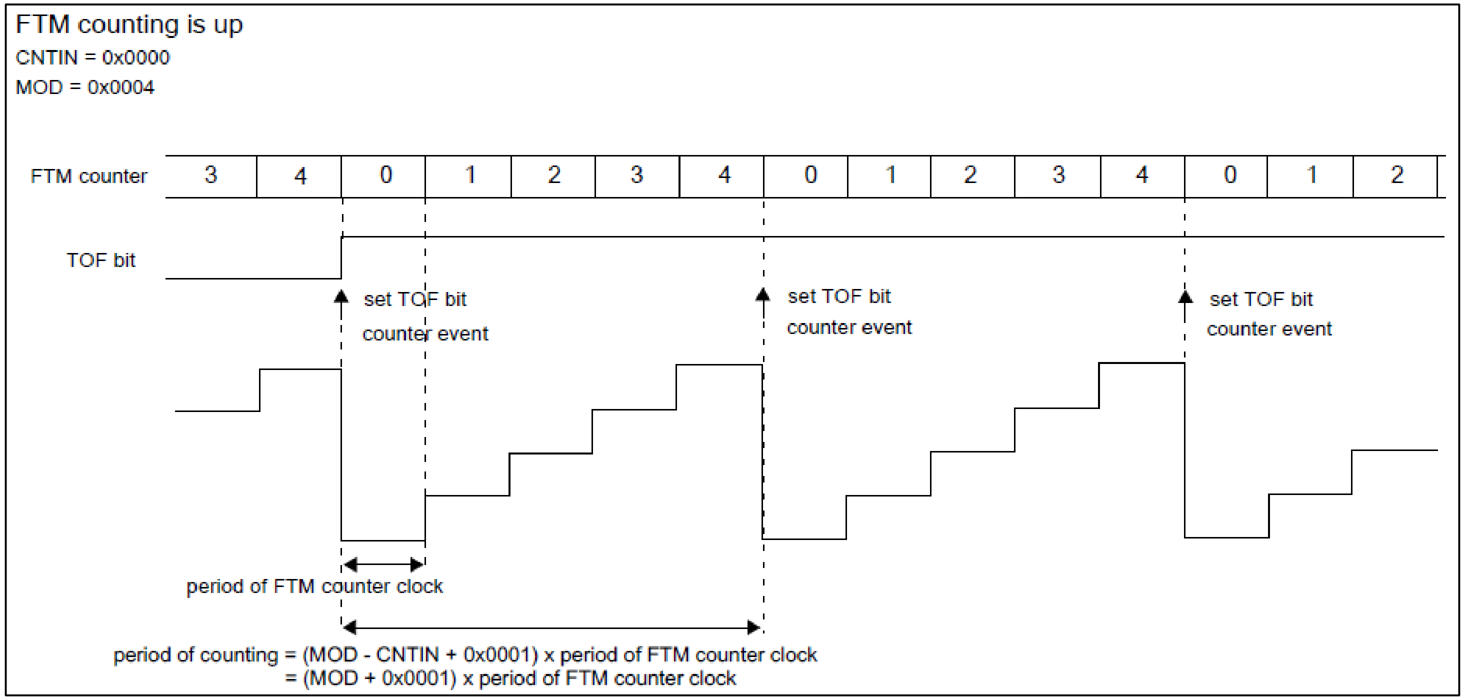
4-3 CLKS	Clock Source Selection Selects one of the three FTM counter clock sources. This field is write protected. It can be written only when MODE[WPDIS] = 1. 00b - No clock selected. This in effect disables the FTM counter. 01b - FTM input clock 10b - Fixed frequency clock 11b - External clock

ACE Lab.

PWM Example

2. Data sheet 분석 : MOD (Modulo) 설정

- ✓ Modulo 레지스터는 FTM Counter를 위한 Modulo 값을 포함한다.
- ✓ FTM Counter가 Modulo 값에 도달하면 TOF bit가 set되고, Counter는 CNTIN 값으로 reload된다.
- ✓ FTM의 주기는 아래 그림과 같이 (MOD-CNTIN+0x0001) x period of FTM counter clock이 된다.

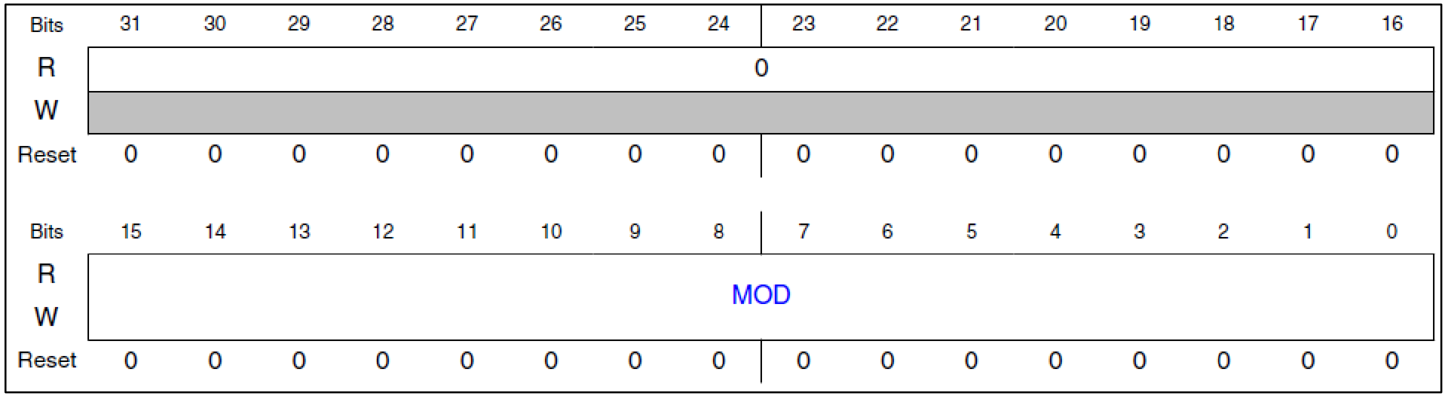


ACE Lab.

PWM Example

2. Data sheet 분석 : MOD (Modulo) 설정

- ✓ 해당 예제에서 MOD 값은 16000-1로 설정한다.
- ✓ 이 값으로 설정 시 FTM0의 주기는 MOD-CNTIN(0)+0x0001 = **16000 counter clocks**이 된다.
- ✓ PWM의 주기는 FTM0 카운터 주기와 동일하므로 16000 x 250ns (4MHz 주파수 pulse의 주기) = 4ms가 된다.



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MOD															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

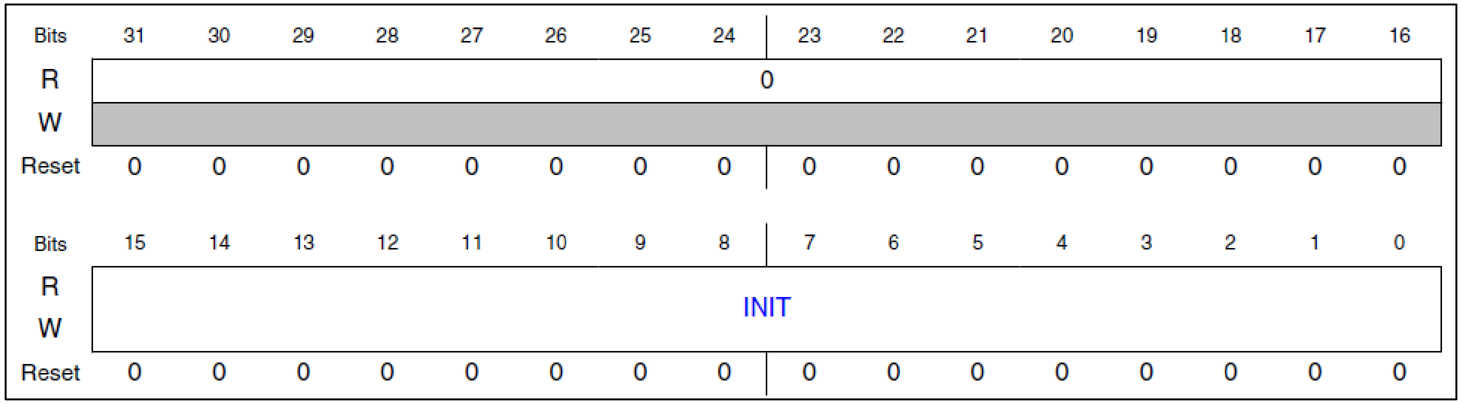
Field	Function
31-16 —	Reserved
15-0 MOD	MOD Modulo Value

ACE Lab.

PWM Example

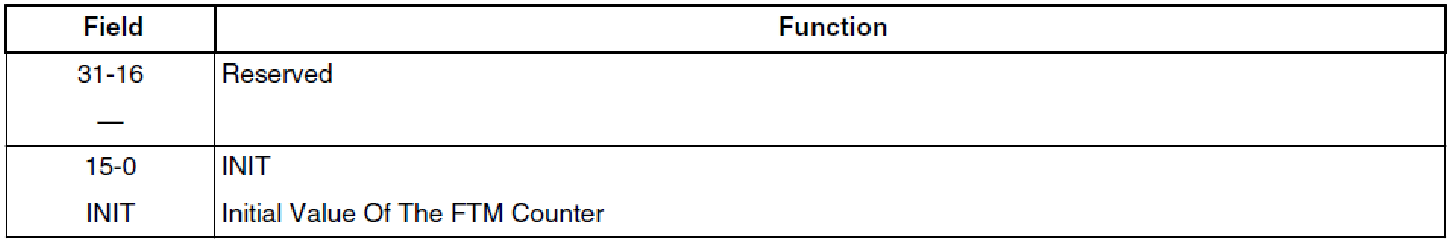
2. Data sheet 분석 : CNTIN (Counter Initial Value) 설정

- ✓ Counter의 초기값을 설정하는 레지스터다.
- ✓ 해당 예제에서 해당 레지스터 값은 0으로 세팅한다.



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	INIT															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



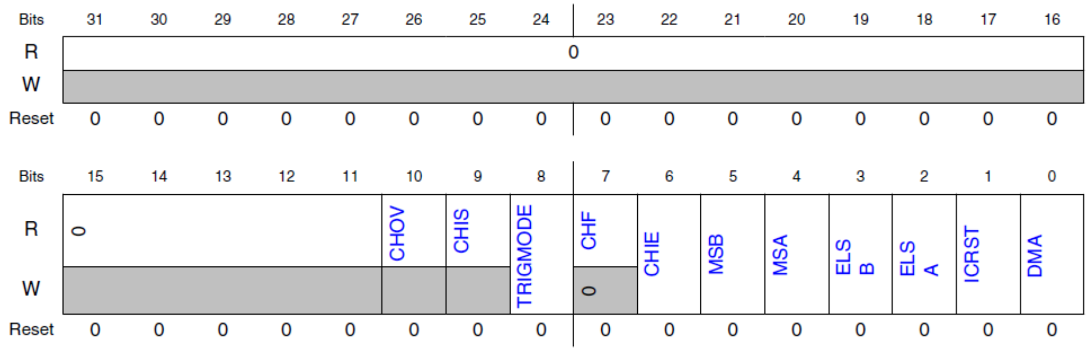
Field	Function
31-16 —	Reserved
15-0 INIT	INIT Initial Value Of The FTM Counter

ACE Lab.

PWM Example

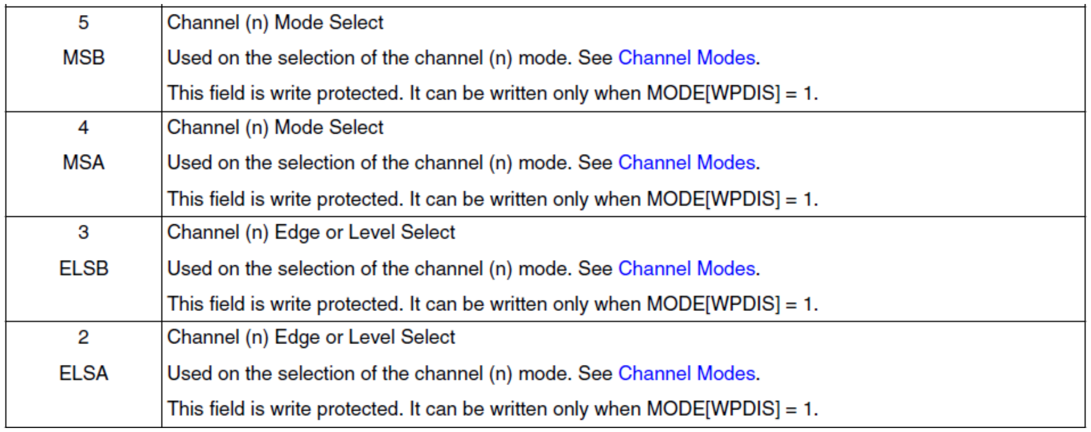
2. Data sheet 분석 : CnSC (Channel (n) Status And Control) 설정

   ✓ MSB, MSA, ELSB, ELSA 비트 설정을 통해서 PWM 설정을 한다.



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0					CHOV	CHIS	TRIGMODE	CHF	CHIE	MSB	MSA	ELSB	ELSA	ICRST	DMA
W	▨					▨	▨		0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



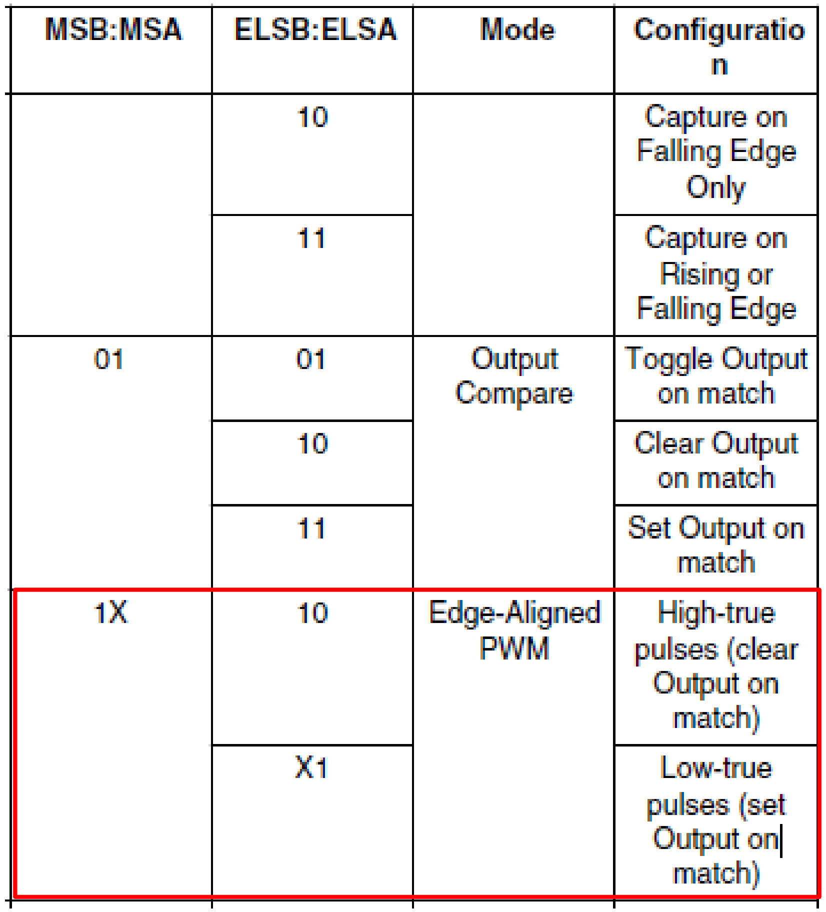
5 MSB	Channel (n) Mode Select Used on the selection of the channel (n) mode. See Channel Modes. This field is write protected. It can be written only when MODE[WPDIS] = 1.
4 MSA	Channel (n) Mode Select Used on the selection of the channel (n) mode. See Channel Modes. This field is write protected. It can be written only when MODE[WPDIS] = 1.
3 ELSB	Channel (n) Edge or Level Select Used on the selection of the channel (n) mode. See Channel Modes. This field is write protected. It can be written only when MODE[WPDIS] = 1.
2 ELSA	Channel (n) Edge or Level Select Used on the selection of the channel (n) mode. See Channel Modes. This field is write protected. It can be written only when MODE[WPDIS] = 1.

ACE Lab.

PWM Example

2. Data sheet 분석 : CnSC (Channel (n) Status And Control) 설정

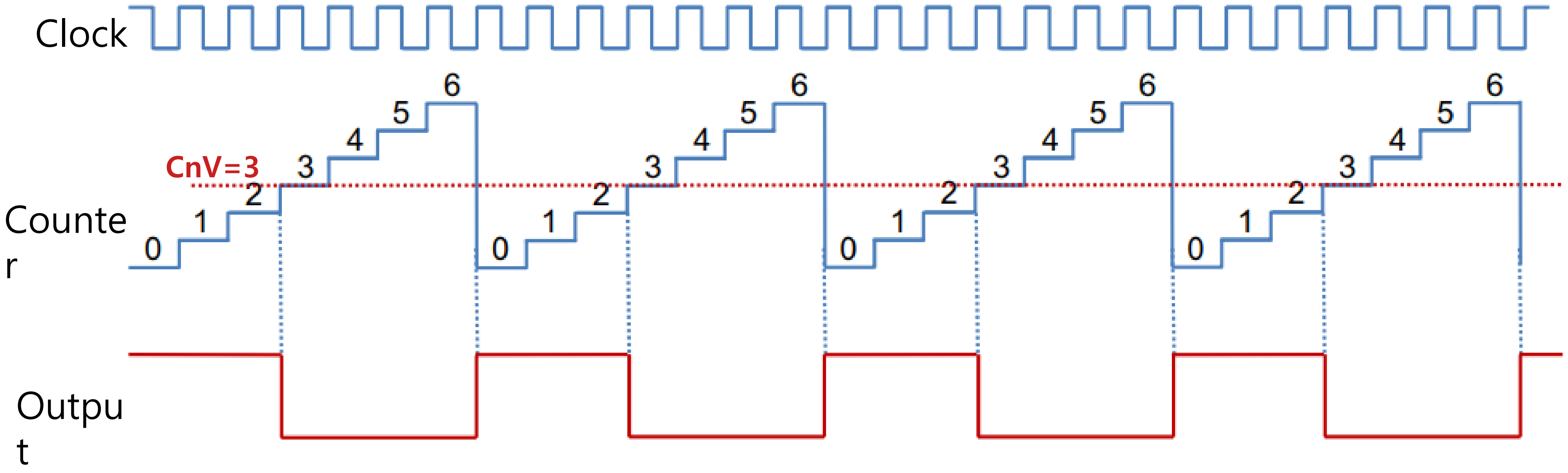
- ✓ 해당 예제에서는 Edge-Aligned PWM Mode로 설정한다.
- ✓ 해당 예제에서는 Low-true pulses로 설정한다.
- ✓ Low-True pulses는 현재 Timer가 CnV 값보다 작을 때 1, CnV 값 이상이면 0이다.
- ✓ High-True pulses는 현재 Timer가 Cnv 값보다 작을 때 0, CnV 값 이상이면 1이다.



MSB:MSA	ELSB:ELSA	Mode	Configuratio n
	10		Capture on Falling Edge Only
	11		Capture on Rising or Falling Edge
01	01	Output Compare	Toggle Output on match
	10		Clear Output on match
	11		Set Output on match
1X	10	Edge-Aligned PWM	High-true pulses (clear Output on match)
	X1		Low-true pulses (set Output on match)

PWM Example

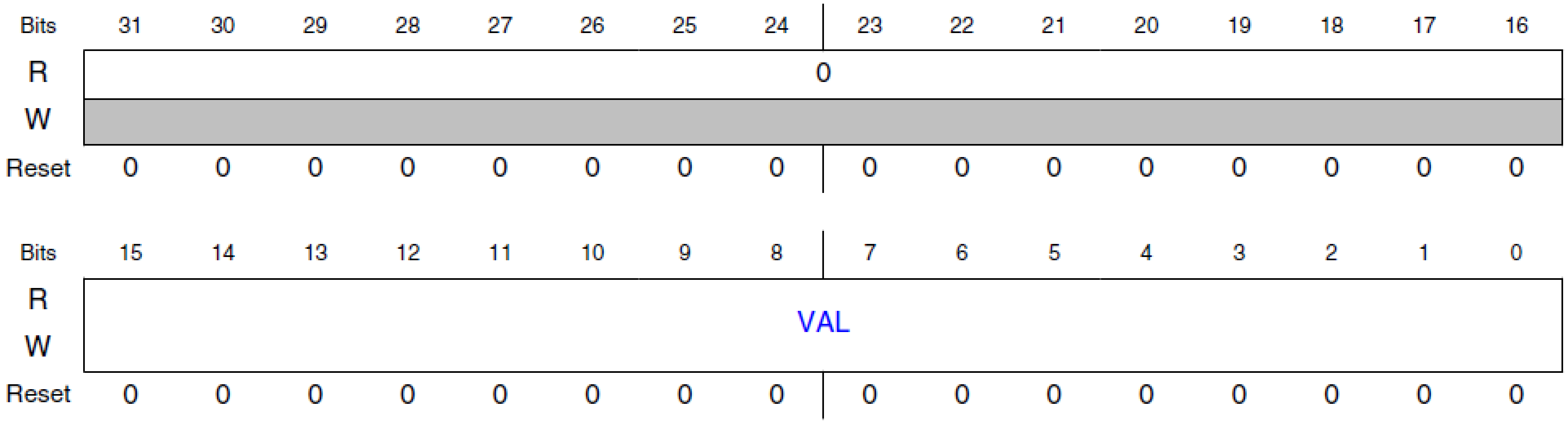
2. Data sheet 분석 : CnSC (Channel (n) Status And Control) 설정
   - ✓ Low-True pulses의 동작 모습 (MOD=7, CNTIN=0, CnV=3)



PWM Example

2. Data sheet 분석 : CnV (Channel (n) Status And Control) 설정

✓ 해당 예제에서는 CnV 값 변경을 통해 LED의 밝기를 제어한다.



Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

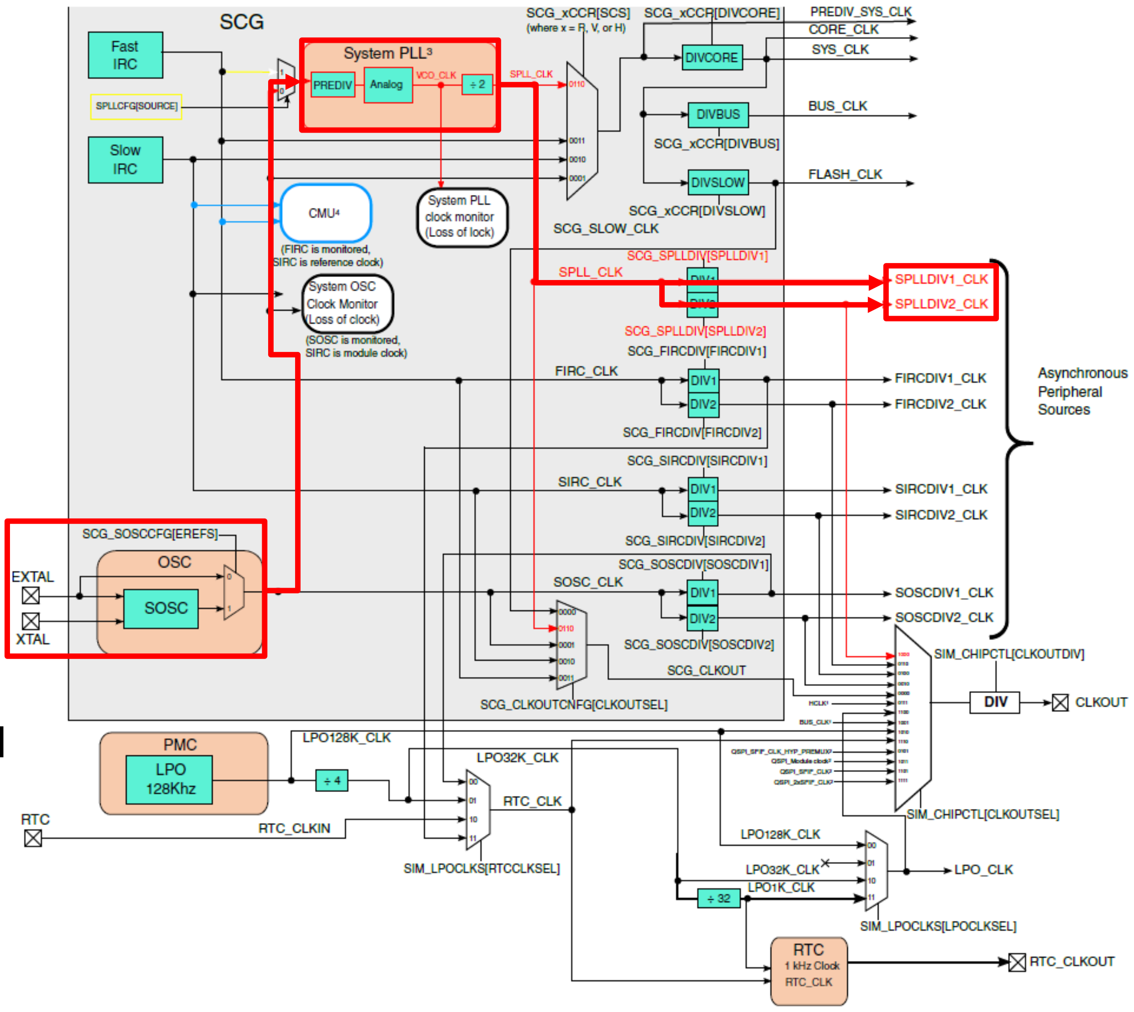
Bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VAL															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Field	Function
31-16 —	Reserved
15-0 VAL	Channel Value Captured FTM counter value of the input modes or the match value for the output modes

ACE Lab.

PWM Example

2. Data sheet 분석 : Clock 설정
   - ✓ Oscilloscope (OSC)를 통해 Clock (SOSC_CLK)을 생성한다.
   - ✓ System PLL을 통해 SOSC_CLK의 주파수를 **변조하여 고주파** Clock (SPLL_CLK)을 생성한다.
   - ✓ **고주파** Clock (SPLL_CLK)을 **분기하여** Asynchronous Peripheral Source Clock (SPLLDIV1_CLK/SPLLDIV2_CLK)을 생성한다.
   - ✓ **생성된** Asynchronous Peripheral Source Clock을 **주변 장치에 인가하여 주변 장치를 동작 시킬 수** 있다.
   - ✓ Clock의 **상세한 설정 방법은 실습 내용을 벗어나기 때문에 구조만 파악하도록** 한다.

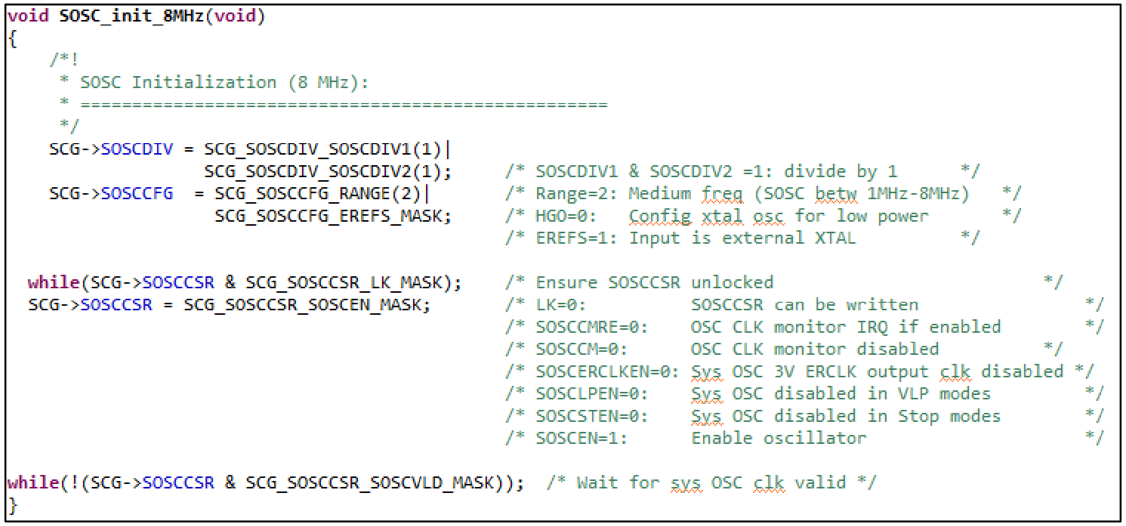


PWM Example

3. 프로그래밍

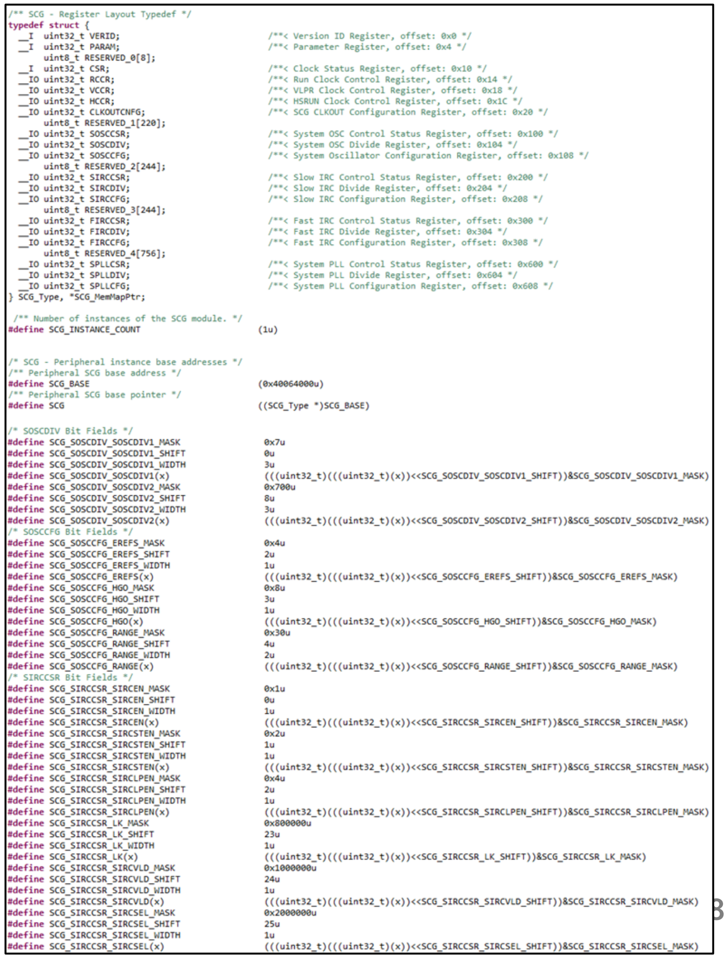
1) LPIT 모듈에 인가될 SPLLDIV2_CLK를 생성하기 위해 SOSC_CLK를 먼저 생성한다.

- ✓ 구체적인 클럭 생성 설정은 실습 내용을 벗어나기 때문에 함수를 그대로 구현하도록 한다.
- ✓ 생성된 SOSC_CLK는 8MHz이다.



```c
void SOSC_init_8MHz(void)
{
    /*!
     * SOSC Initialization (8 MHz):
     * ===================================================
     */
    SCG->SOSCDIV = SCG_SOSCDIV_SOSCDIV1(1)|
                   SCG_SOSCDIV_SOSCDIV2(1);     /* SOSCDIV1 & SOSCDIV2 =1: divide by 1      */
    SCG->SOSCCFG  = SCG_SOSCCFG_RANGE(2)|       /* Range=2: Medium freq (SOSC betw 1MHz-8MHz)   */
                    SCG_SOSCCFG_EREFS_MASK;     /* HGO=0:   Config xtal osc for low power       */
                                                /* EREFS=1: Input is external XTAL          */

  while(SCG->SOSCCSR & SCG_SOSCCSR_LK_MASK);    /* Ensure SOSCCSR unlocked                      */
  SCG->SOSCCSR = SCG_SOSCCSR_SOSCEN_MASK;       /* LK=0:          SOSCCSR can be written           */
                                                /* SOSCCMRE=0:    OSC CLK monitor IRQ if enabled   */
                                                /* SOSCCM=0:      OSC CLK monitor disabled      */
                                                /* SOSCERCLKEN=0: Sys OSC 3V ERCLK output clk disabled */
                                                /* SOSCLPEN=0:    Sys OSC disabled in VLP modes    */
                                                /* SOSCSTEN=0:    Sys OSC disabled in Stop modes   */
                                                /* SOSCEN=1:      Enable oscillator                */

while(!(SCG->SOSCCSR & SCG_SOSCCSR_SOSCVLD_MASK));  /* Wait for sys OSC clk valid */
}
```

SOSC_CLK 생성 함수



```c
/** SCG - Register Layout Typedef */
typedef struct {
  __I  uint32_t VERID;                          /**< Version ID Register, offset: 0x0 */
  __I  uint32_t PARAM;                          /**< Parameter Register, offset: 0x4 */
       uint8_t RESERVED_0[8];
  __I  uint32_t CSR;                            /**< Clock Status Register, offset: 0x10 */
  __IO uint32_t RCCR;                           /**< Run Clock Control Register, offset: 0x14 */
  __IO uint32_t VCCR;                           /**< VLPR Clock Control Register, offset: 0x18 */
  __IO uint32_t HCCR;                           /**< HSRUN Clock Control Register, offset: 0x1C */
  __IO uint32_t CLKOUTCNFG;                     /**< SCG CLKOUT Configuration Register, offset: 0x20 */
       uint8_t RESERVED_1[220];
  __IO uint32_t SOSCCSR;                        /**< System OSC Control Status Register, offset: 0x100 */
  __IO uint32_t SOSCDIV;                        /**< System OSC Divide Register, offset: 0x104 */
  __IO uint32_t SOSCCFG;                        /**< System Oscillator Configuration Register, offset: 0x108 */
       uint8_t RESERVED_2[244];
  __IO uint32_t SIRCCSR;                        /**< Slow IRC Control Status Register, offset: 0x200 */
  __IO uint32_t SIRCDIV;                        /**< Slow IRC Divide Register, offset: 0x204 */
  __IO uint32_t SIRCCFG;                        /**< Slow IRC Configuration Register, offset: 0x208 */
       uint8_t RESERVED_3[244];
  __IO uint32_t FIRCCSR;                        /**< Fast IRC Control Status Register, offset: 0x300 */
  __IO uint32_t FIRCDIV;                        /**< Fast IRC Divide Register, offset: 0x304 */
  __IO uint32_t FIRCCFG;                        /**< Fast IRC Configuration Register, offset: 0x308 */
       uint8_t RESERVED_4[756];
  __IO uint32_t SPLLCSR;                        /**< System PLL Control Status Register, offset: 0x600 */
  __IO uint32_t SPLLDIV;                        /**< System PLL Divide Register, offset: 0x604 */
  __IO uint32_t SPLLCFG;                        /**< System PLL Configuration Register, offset: 0x608 */
} SCG_Type, *SCG_MemMapPtr;

 /** Number of instances of the SCG module. */
#define SCG_INSTANCE_COUNT                      (1u)


/* SCG - Peripheral instance base addresses */
/** Peripheral SCG base address */
#define SCG_BASE                                (0x40064000u)
/** Peripheral SCG base pointer */
#define SCG                                     ((SCG_Type *)SCG_BASE)

/* SOSCDIV Bit Fields */
#define SCG_SOSCDIV_SOSCDIV1_MASK                0x7u
#define SCG_SOSCDIV_SOSCDIV1_SHIFT               0u
#define SCG_SOSCDIV_SOSCDIV1_WIDTH               3u
#define SCG_SOSCDIV_SOSCDIV1(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SOSCDIV_SOSCDIV1_SHIFT))&SCG_SOSCDIV_SOSCDIV1_MASK)
#define SCG_SOSCDIV_SOSCDIV2_MASK                0x700u
#define SCG_SOSCDIV_SOSCDIV2_SHIFT               8u
#define SCG_SOSCDIV_SOSCDIV2_WIDTH               3u
#define SCG_SOSCDIV_SOSCDIV2(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SOSCDIV_SOSCDIV2_SHIFT))&SCG_SOSCDIV_SOSCDIV2_MASK)
/* SOSCCFG Bit Fields */
#define SCG_SOSCCFG_EREFS_MASK                   0x4u
#define SCG_SOSCCFG_EREFS_SHIFT                  2u
#define SCG_SOSCCFG_EREFS_WIDTH                  1u
#define SCG_SOSCCFG_EREFS(x)                     (((uint32_t)(((uint32_t)(x))<<SCG_SOSCCFG_EREFS_SHIFT))&SCG_SOSCCFG_EREFS_MASK)
#define SCG_SOSCCFG_HGO_MASK                     0x8u
#define SCG_SOSCCFG_HGO_SHIFT                    3u
#define SCG_SOSCCFG_HGO_WIDTH                    1u
#define SCG_SOSCCFG_HGO(x)                       (((uint32_t)(((uint32_t)(x))<<SCG_SOSCCFG_HGO_SHIFT))&SCG_SOSCCFG_HGO_MASK)
#define SCG_SOSCCFG_RANGE_MASK                   0x30u
#define SCG_SOSCCFG_RANGE_SHIFT                  4u
#define SCG_SOSCCFG_RANGE_WIDTH                  2u
#define SCG_SOSCCFG_RANGE(x)                     (((uint32_t)(((uint32_t)(x))<<SCG_SOSCCFG_RANGE_SHIFT))&SCG_SOSCCFG_RANGE_MASK)
/* SIRCCSR Bit Fields */
#define SCG_SIRCCSR_SIRCEN_MASK                  0x1u
#define SCG_SIRCCSR_SIRCEN_SHIFT                 0u
#define SCG_SIRCCSR_SIRCEN_WIDTH                 1u
#define SCG_SIRCCSR_SIRCEN(x)                    (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_SIRCEN_SHIFT))&SCG_SIRCCSR_SIRCEN_MASK)
#define SCG_SIRCCSR_SIRCSTEN_MASK                0x2u
#define SCG_SIRCCSR_SIRCSTEN_SHIFT               1u
#define SCG_SIRCCSR_SIRCSTEN_WIDTH               1u
#define SCG_SIRCCSR_SIRCSTEN(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_SIRCSTEN_SHIFT))&SCG_SIRCCSR_SIRCSTEN_MASK)
#define SCG_SIRCCSR_SIRCLPEN_MASK                0x4u
#define SCG_SIRCCSR_SIRCLPEN_SHIFT               2u
#define SCG_SIRCCSR_SIRCLPEN_WIDTH               1u
#define SCG_SIRCCSR_SIRCLPEN(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_SIRCLPEN_SHIFT))&SCG_SIRCCSR_SIRCLPEN_MASK)
#define SCG_SIRCCSR_LK_MASK                      0x800000u
#define SCG_SIRCCSR_LK_SHIFT                     23u
#define SCG_SIRCCSR_LK_WIDTH                     1u
#define SCG_SIRCCSR_LK(x)                        (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_LK_SHIFT))&SCG_SIRCCSR_LK_MASK)
#define SCG_SIRCCSR_SIRCVLD_MASK                 0x1000000u
#define SCG_SIRCCSR_SIRCVLD_SHIFT                24u
#define SCG_SIRCCSR_SIRCVLD_WIDTH                1u
#define SCG_SIRCCSR_SIRCVLD(x)                   (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_SIRCVLD_SHIFT))&SCG_SIRCCSR_SIRCVLD_MASK)
#define SCG_SIRCCSR_SIRCSEL_MASK                 0x2000000u
#define SCG_SIRCCSR_SIRCSEL_SHIFT                25u
#define SCG_SIRCCSR_SIRCSEL_WIDTH                1u
#define SCG_SIRCCSR_SIRCSEL(x)                   (((uint32_t)(((uint32_t)(x))<<SCG_SIRCCSR_SIRCSEL_SHIFT))&SCG_SIRCCSR_SIRCSEL_MASK)
```

SOSC_CLK 관련 메모리 맵/매크로
(참고)

ACE Lab.

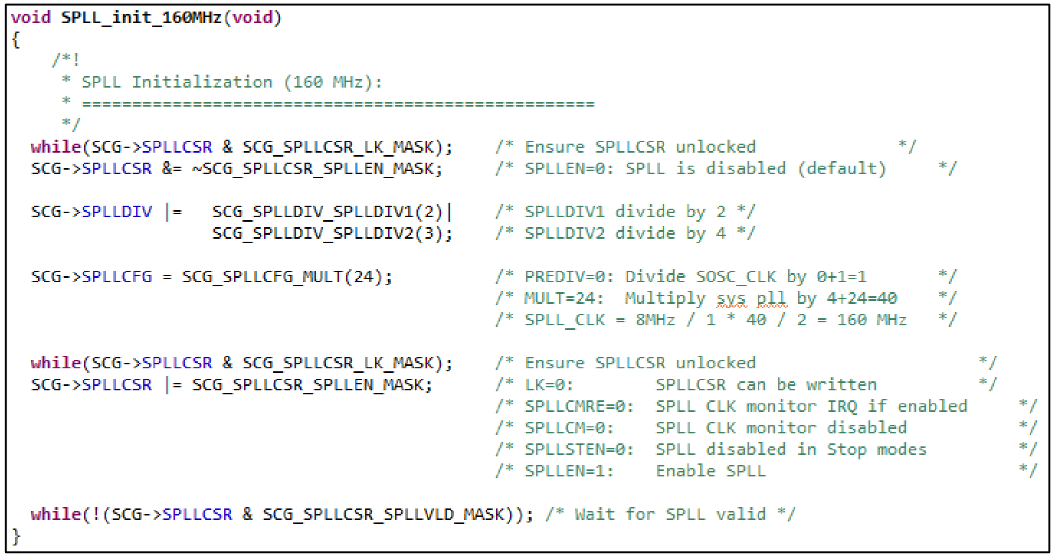
PWM Example

3. **프로그래밍**

   2) SOSC_CLK를 **바탕으로** SPLL_CLK를 **생성하고 이를 분기하여** SPLLDIV2_CLK를 **생성한다.**

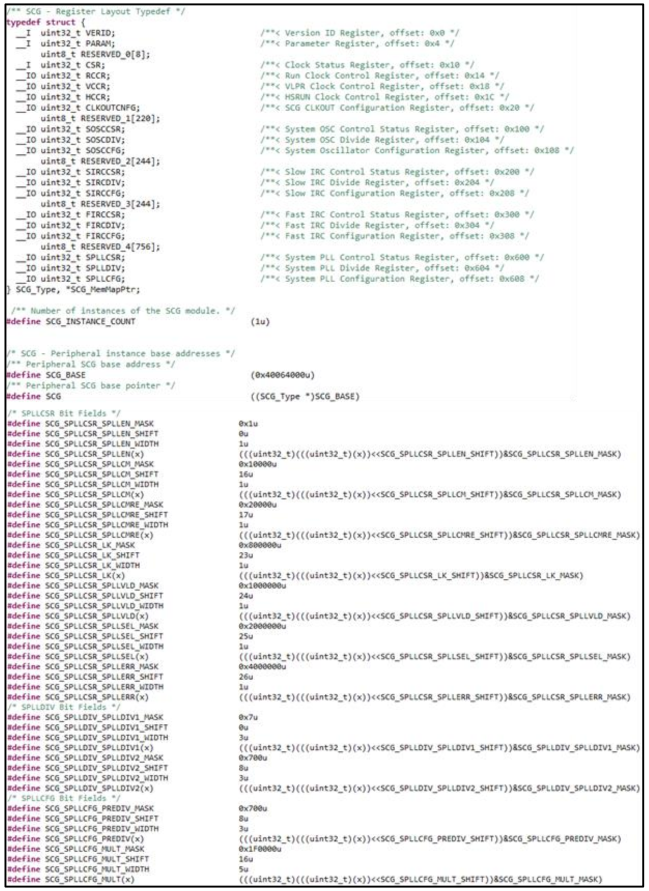
      ✓ **구체적인 클럭 생성 설정은 실습 내용을 벗어나기 때문에 함수를 그대로 구현하도록 한다.**

      ✓ **생성된** SPLL_CLK는 160MHz**이고,** SPLLDIV2_CLK는 40MHz**이다.**



```c
void SPLL_init_160MHz(void)
{
    /*!
     * SPLL Initialization (160 MHz):
     * ====================================================
     */
  while(SCG->SPLLCSR & SCG_SPLLCSR_LK_MASK);    /* Ensure SPLLCSR unlocked              */
  SCG->SPLLCSR &= ~SCG_SPLLCSR_SPLLEN_MASK;     /* SPLLEN=0: SPLL is disabled (default)     */

  SCG->SPLLDIV |=   SCG_SPLLDIV_SPLLDIV1(2)|    /* SPLLDIV1 divide by 2 */
                    SCG_SPLLDIV_SPLLDIV2(3);    /* SPLLDIV2 divide by 4 */

  SCG->SPLLCFG = SCG_SPLLCFG_MULT(24);          /* PREDIV=0: Divide SOSC_CLK by 0+1=1       */
                                                /* MULT=24:  Multiply sys pll by 4+24=40    */
                                                /* SPLL_CLK = 8MHz / 1 * 40 / 2 = 160 MHz   */

  while(SCG->SPLLCSR & SCG_SPLLCSR_LK_MASK);    /* Ensure SPLLCSR unlocked                      */
  SCG->SPLLCSR |= SCG_SPLLCSR_SPLLEN_MASK;      /* LK=0:        SPLLCSR can be written          */
                                                /* SPLLCMRE=0:  SPLL CLK monitor IRQ if enabled     */
                                                /* SPLLCM=0:    SPLL CLK monitor disabled           */
                                                /* SPLLSTEN=0:  SPLL disabled in Stop modes         */
                                                /* SPLLEN=1:    Enable SPLL                         */

  while(!(SCG->SPLLCSR & SCG_SPLLCSR_SPLLVLD_MASK)); /* Wait for SPLL valid */
}
```

SPLL_CLK/SPLLDIV2_CLK 생성 함수



```c
/** SCG - Register Layout Typedef */
typedef struct {
  __I  uint32_t VERID;                          /**< Version ID Register, offset: 0x0 */
  __I  uint32_t PARAM;                          /**< Parameter Register, offset: 0x4 */
       uint8_t RESERVED_0[8];
  __I  uint32_t CSR;                            /**< Clock Status Register, offset: 0x10 */
  __IO uint32_t RCCR;                           /**< Run Clock Control Register, offset: 0x14 */
  __IO uint32_t VCCR;                           /**< VLPR Clock Control Register, offset: 0x18 */
  __IO uint32_t HCCR;                           /**< HSRUN Clock Control Register, offset: 0x1C */
  __IO uint32_t CLKOUTCNFG;                     /**< SCG CLKOUT Configuration Register, offset: 0x20 */
       uint8_t RESERVED_1[220];
  __IO uint32_t SOSCCSR;                        /**< System OSC Control Status Register, offset: 0x100 */
  __IO uint32_t SOSCDIV;                        /**< System OSC Divide Register, offset: 0x104 */
  __IO uint32_t SOSCCFG;                        /**< System Oscillator Configuration Register, offset: 0x108 */
       uint8_t RESERVED_2[244];
  __IO uint32_t SIRCCSR;                        /**< Slow IRC Control Status Register, offset: 0x200 */
  __IO uint32_t SIRCDIV;                        /**< Slow IRC Divide Register, offset: 0x204 */
  __IO uint32_t SIRCCFG;                        /**< Slow IRC Configuration Register, offset: 0x208 */
       uint8_t RESERVED_3[244];
  __IO uint32_t FIRCCSR;                        /**< Fast IRC Control Status Register, offset: 0x300 */
  __IO uint32_t FIRCDIV;                        /**< Fast IRC Divide Register, offset: 0x304 */
  __IO uint32_t FIRCCFG;                        /**< Fast IRC Configuration Register, offset: 0x308 */
       uint8_t RESERVED_4[756];
  __IO uint32_t SPLLCSR;                        /**< System PLL Control Status Register, offset: 0x600 */
  __IO uint32_t SPLLDIV;                        /**< System PLL Divide Register, offset: 0x604 */
  __IO uint32_t SPLLCFG;                        /**< System PLL Configuration Register, offset: 0x608 */
} SCG_Type, *SCG_MemMapPtr;

 /** Number of instances of the SCG module. */
#define SCG_INSTANCE_COUNT                       (1u)


/* SCG - Peripheral instance base addresses */
/** Peripheral SCG base address */
#define SCG_BASE                                 (0x40064000u)
/** Peripheral SCG base pointer */
#define SCG                                      ((SCG_Type *)SCG_BASE)

/* SPLLCSR Bit Fields */
#define SCG_SPLLCSR_SPLLEN_MASK                 0x1u
#define SCG_SPLLCSR_SPLLEN_SHIFT                0u
#define SCG_SPLLCSR_SPLLEN_WIDTH                1u
#define SCG_SPLLCSR_SPLLEN(x)                   (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLEN_SHIFT))&SCG_SPLLCSR_SPLLEN_MASK)
#define SCG_SPLLCSR_SPLLCM_MASK                 0x10000u
#define SCG_SPLLCSR_SPLLCM_SHIFT                16u
#define SCG_SPLLCSR_SPLLCM_WIDTH                1u
#define SCG_SPLLCSR_SPLLCM(x)                   (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLCM_SHIFT))&SCG_SPLLCSR_SPLLCM_MASK)
#define SCG_SPLLCSR_SPLLCMRE_MASK               0x20000u
#define SCG_SPLLCSR_SPLLCMRE_SHIFT              17u
#define SCG_SPLLCSR_SPLLCMRE_WIDTH              1u
#define SCG_SPLLCSR_SPLLCMRE(x)                 (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLCMRE_SHIFT))&SCG_SPLLCSR_SPLLCMRE_MASK)
#define SCG_SPLLCSR_LK_MASK                     0x800000u
#define SCG_SPLLCSR_LK_SHIFT                    23u
#define SCG_SPLLCSR_LK_WIDTH                    1u
#define SCG_SPLLCSR_LK(x)                       (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_LK_SHIFT))&SCG_SPLLCSR_LK_MASK)
#define SCG_SPLLCSR_SPLLVLD_MASK                0x1000000u
#define SCG_SPLLCSR_SPLLVLD_SHIFT               24u
#define SCG_SPLLCSR_SPLLVLD_WIDTH               1u
#define SCG_SPLLCSR_SPLLVLD(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLVLD_SHIFT))&SCG_SPLLCSR_SPLLVLD_MASK)
#define SCG_SPLLCSR_SPLLSEL_MASK                0x2000000u
#define SCG_SPLLCSR_SPLLSEL_SHIFT               25u
#define SCG_SPLLCSR_SPLLSEL_WIDTH               1u
#define SCG_SPLLCSR_SPLLSEL(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLSEL_SHIFT))&SCG_SPLLCSR_SPLLSEL_MASK)
#define SCG_SPLLCSR_SPLLERR_MASK                0x4000000u
#define SCG_SPLLCSR_SPLLERR_SHIFT               26u
#define SCG_SPLLCSR_SPLLERR_WIDTH               1u
#define SCG_SPLLCSR_SPLLERR(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCSR_SPLLERR_SHIFT))&SCG_SPLLCSR_SPLLERR_MASK)
/* SPLLDIV Bit Fields */
#define SCG_SPLLDIV_SPLLDIV1_MASK               0x7u
#define SCG_SPLLDIV_SPLLDIV1_SHIFT              0u
#define SCG_SPLLDIV_SPLLDIV1_WIDTH              3u
#define SCG_SPLLDIV_SPLLDIV1(x)                 (((uint32_t)(((uint32_t)(x))<<SCG_SPLLDIV_SPLLDIV1_SHIFT))&SCG_SPLLDIV_SPLLDIV1_MASK)
#define SCG_SPLLDIV_SPLLDIV2_MASK               0x700u
#define SCG_SPLLDIV_SPLLDIV2_SHIFT              8u
#define SCG_SPLLDIV_SPLLDIV2_WIDTH              3u
#define SCG_SPLLDIV_SPLLDIV2(x)                 (((uint32_t)(((uint32_t)(x))<<SCG_SPLLDIV_SPLLDIV2_SHIFT))&SCG_SPLLDIV_SPLLDIV2_MASK)
/* SPLLCFG Bit Fields */
#define SCG_SPLLCFG_PREDIV_MASK                 0x700u
#define SCG_SPLLCFG_PREDIV_SHIFT                8u
#define SCG_SPLLCFG_PREDIV_WIDTH                3u
#define SCG_SPLLCFG_PREDIV(x)                   (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCFG_PREDIV_SHIFT))&SCG_SPLLCFG_PREDIV_MASK)
#define SCG_SPLLCFG_MULT_MASK                   0x1F0000u
#define SCG_SPLLCFG_MULT_SHIFT                  16u
#define SCG_SPLLCFG_MULT_WIDTH                  5u
#define SCG_SPLLCFG_MULT(x)                     (((uint32_t)(((uint32_t)(x))<<SCG_SPLLCFG_MULT_SHIFT))&SCG_SPLLCFG_MULT_MASK)
```

SPLL_CLK/SPLLDIV2_CLK
관련 메모리 맵/매크로
(참고)

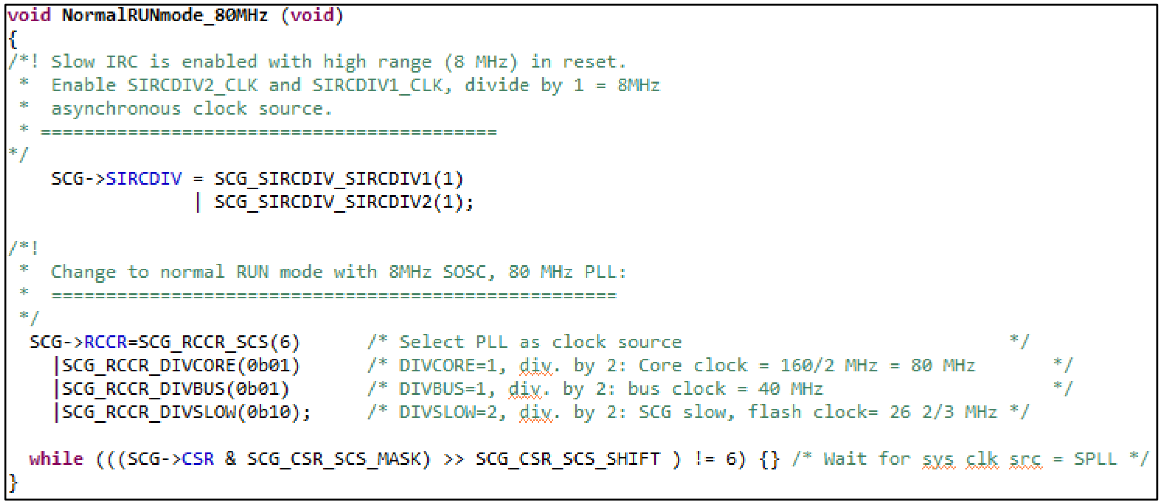
ACE Lab.

PWM Example

3. 프로그래밍

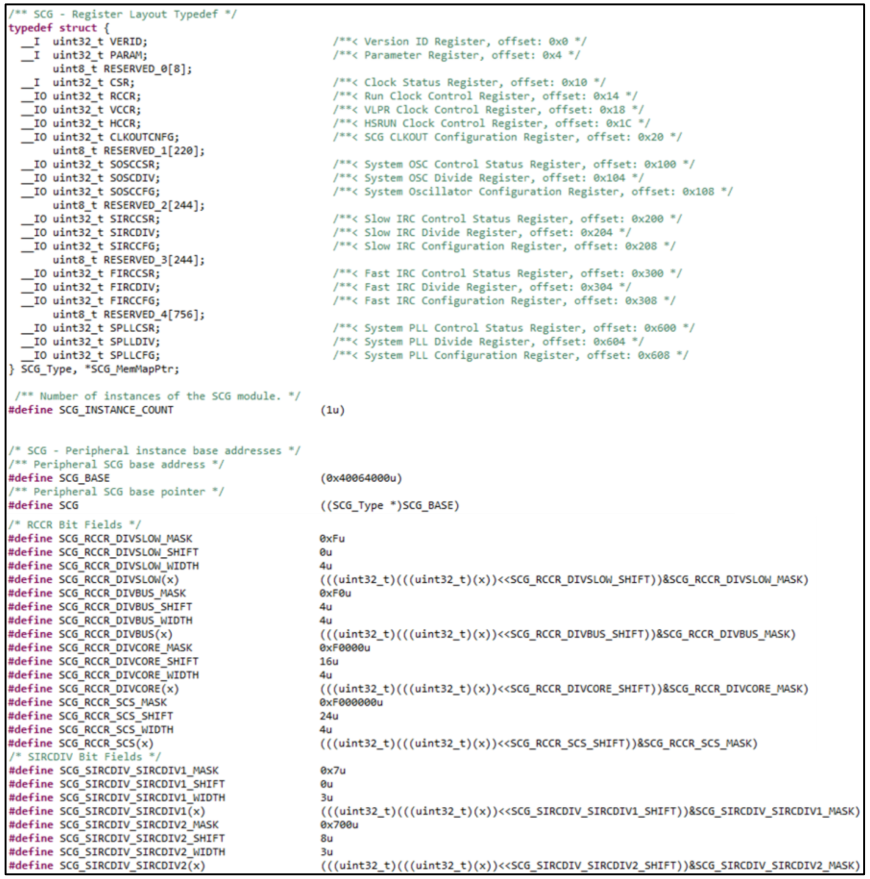
3) 시스템이 동작하기 위한 System Clock/Bus Clock/Flash Clock을 생성한다.

- ✓ 구체적인 클럭 생성 설정은 실습 내용을 벗어나기 때문에 함수를 그대로 구현하도록 한다.
- ✓ 생성된 System Clock은 80MHz이고, Bus Clock은 40MHz이고, Flash Clock은 26.67MHz이다.



```c
void NormalRUNmode_80MHz (void)
{
/*! Slow IRC is enabled with high range (8 MHz) in reset.
 *  Enable SIRCDIV2_CLK and SIRCDIV1_CLK, divide by 1 = 8MHz
 *  asynchronous clock source.
 * ==========================================
*/
    SCG->SIRCDIV = SCG_SIRCDIV_SIRCDIV1(1)
                 | SCG_SIRCDIV_SIRCDIV2(1);

/*!
 *  Change to normal RUN mode with 8MHz SOSC, 80 MHz PLL:
 *  ====================================================
 */
  SCG->RCCR=SCG_RCCR_SCS(6)      /* Select PLL as clock source                    */
    |SCG_RCCR_DIVCORE(0b01)      /* DIVCORE=1, div. by 2: Core clock = 160/2 MHz = 80 MHz     */
    |SCG_RCCR_DIVBUS(0b01)       /* DIVBUS=1, div. by 2: bus clock = 40 MHz                   */
    |SCG_RCCR_DIVSLOW(0b10);     /* DIVSLOW=2, div. by 2: SCG slow, flash clock= 26 2/3 MHz */

  while (((SCG->CSR & SCG_CSR_SCS_MASK) >> SCG_CSR_SCS_SHIFT ) != 6) {} /* Wait for sys clk src = SPLL */
}
```

System Clock/Bus Clock/Flash Clock 생성 함수



```c
/** SCG - Register Layout Typedef */
typedef struct {
  __I  uint32_t VERID;                          /**< Version ID Register, offset: 0x0 */
  __I  uint32_t PARAM;                          /**< Parameter Register, offset: 0x4 */
       uint8_t RESERVED_0[8];
  __I  uint32_t CSR;                            /**< Clock Status Register, offset: 0x10 */
  __IO uint32_t RCCR;                           /**< Run Clock Control Register, offset: 0x14 */
  __IO uint32_t VCCR;                           /**< VLPR Clock Control Register, offset: 0x18 */
  __IO uint32_t HCCR;                           /**< HSRUN Clock Control Register, offset: 0x1C */
  __IO uint32_t CLKOUTCNFG;                     /**< SCG CLKOUT Configuration Register, offset: 0x20 */
       uint8_t RESERVED_1[220];
  __IO uint32_t SOSCCSR;                        /**< System OSC Control Status Register, offset: 0x100 */
  __IO uint32_t SOSCDIV;                        /**< System OSC Divide Register, offset: 0x104 */
  __IO uint32_t SOSCCFG;                        /**< System Oscillator Configuration Register, offset: 0x108 */
       uint8_t RESERVED_2[244];
  __IO uint32_t SIRCCSR;                        /**< Slow IRC Control Status Register, offset: 0x200 */
  __IO uint32_t SIRCDIV;                        /**< Slow IRC Divide Register, offset: 0x204 */
  __IO uint32_t SIRCCFG;                        /**< Slow IRC Configuration Register, offset: 0x208 */
       uint8_t RESERVED_3[244];
  __IO uint32_t FIRCCSR;                        /**< Fast IRC Control Status Register, offset: 0x300 */
  __IO uint32_t FIRCDIV;                        /**< Fast IRC Divide Register, offset: 0x304 */
  __IO uint32_t FIRCCFG;                        /**< Fast IRC Configuration Register, offset: 0x308 */
       uint8_t RESERVED_4[756];
  __IO uint32_t SPLLCSR;                        /**< System PLL Control Status Register, offset: 0x600 */
  __IO uint32_t SPLLDIV;                        /**< System PLL Divide Register, offset: 0x604 */
  __IO uint32_t SPLLCFG;                        /**< System PLL Configuration Register, offset: 0x608 */
} SCG_Type, *SCG_MemMapPtr;

 /** Number of instances of the SCG module. */
#define SCG_INSTANCE_COUNT                       (1u)


/* SCG - Peripheral instance base addresses */
/** Peripheral SCG base address */
#define SCG_BASE                                 (0x40064000u)
/** Peripheral SCG base pointer */
#define SCG                                      ((SCG_Type *)SCG_BASE)
/* RCCR Bit Fields */
#define SCG_RCCR_DIVSLOW_MASK                    0xFu
#define SCG_RCCR_DIVSLOW_SHIFT                   0u
#define SCG_RCCR_DIVSLOW_WIDTH                   4u
#define SCG_RCCR_DIVSLOW(x)                      (((uint32_t)(((uint32_t)(x))<<SCG_RCCR_DIVSLOW_SHIFT))&SCG_RCCR_DIVSLOW_MASK)
#define SCG_RCCR_DIVBUS_MASK                     0xF0u
#define SCG_RCCR_DIVBUS_SHIFT                    4u
#define SCG_RCCR_DIVBUS_WIDTH                    4u
#define SCG_RCCR_DIVBUS(x)                       (((uint32_t)(((uint32_t)(x))<<SCG_RCCR_DIVBUS_SHIFT))&SCG_RCCR_DIVBUS_MASK)
#define SCG_RCCR_DIVCORE_MASK                    0xF0000u
#define SCG_RCCR_DIVCORE_SHIFT                   16u
#define SCG_RCCR_DIVCORE_WIDTH                   4u
#define SCG_RCCR_DIVCORE(x)                      (((uint32_t)(((uint32_t)(x))<<SCG_RCCR_DIVCORE_SHIFT))&SCG_RCCR_DIVCORE_MASK)
#define SCG_RCCR_SCS_MASK                        0xF000000u
#define SCG_RCCR_SCS_SHIFT                       24u
#define SCG_RCCR_SCS_WIDTH                       4u
#define SCG_RCCR_SCS(x)                          (((uint32_t)(((uint32_t)(x))<<SCG_RCCR_SCS_SHIFT))&SCG_RCCR_SCS_MASK)
/* SIRCDIV Bit Fields */
#define SCG_SIRCDIV_SIRCDIV1_MASK                0x7u
#define SCG_SIRCDIV_SIRCDIV1_SHIFT               0u
#define SCG_SIRCDIV_SIRCDIV1_WIDTH               3u
#define SCG_SIRCDIV_SIRCDIV1(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SIRCDIV_SIRCDIV1_SHIFT))&SCG_SIRCDIV_SIRCDIV1_MASK)
#define SCG_SIRCDIV_SIRCDIV2_MASK                0x700u
#define SCG_SIRCDIV_SIRCDIV2_SHIFT               8u
#define SCG_SIRCDIV_SIRCDIV2_WIDTH               3u
#define SCG_SIRCDIV_SIRCDIV2(x)                  (((uint32_t)(((uint32_t)(x))<<SCG_SIRCDIV_SIRCDIV2_SHIFT))&SCG_SIRCDIV_SIRCDIV2_MASK)
```

System Clock/Bus Clock/Flash Clock
관련 메모리 맵/매크로
(참고)

ACE Lab.

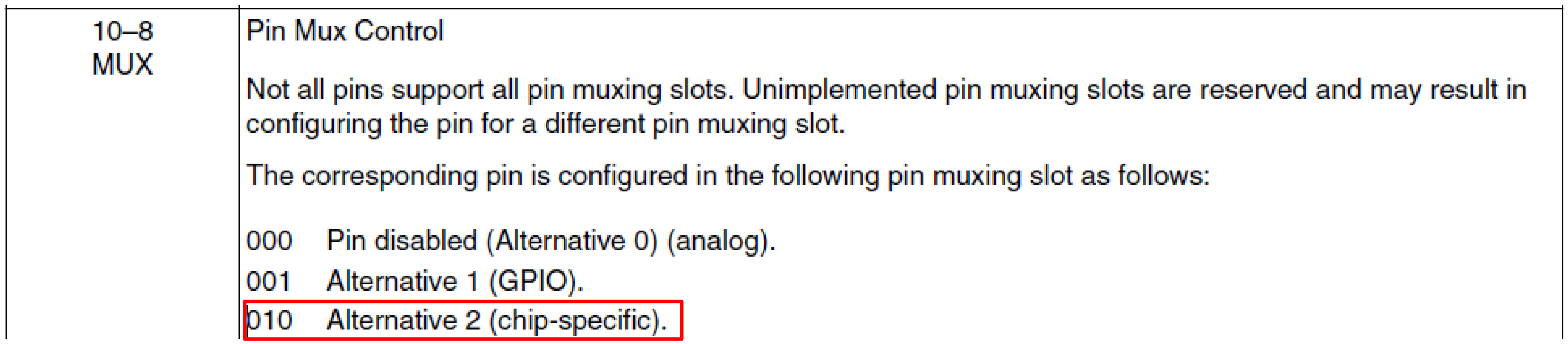
PWM Example

3. 프로그래밍

   4) LED Example을 참조하여 PORT_init()을 작성한다.

      ✓ FTM0_CH1을 사용할 것이기 때문에 PCC 레지스터에서 PORTD의 클럭을 Enable한다.

      ✓ PCR 레지스터의 MUX 비트를 2로 세팅하여 Alternative 2 Functionality를 사용한다.



10–8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (Alternative 0) (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific).

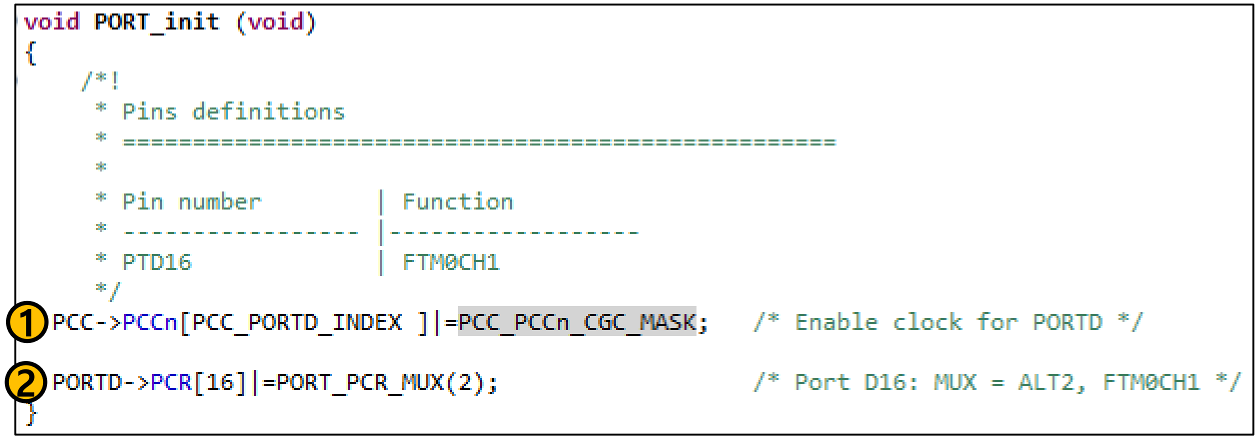
PWM Example

3. 프로그래밍

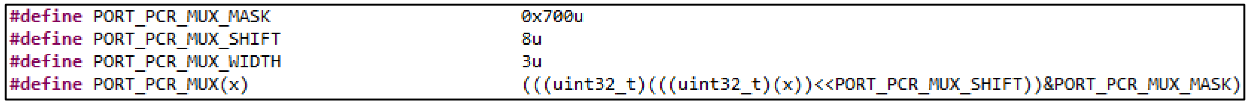
   4) LED Example을 참조하여 PORT_init()을 작성한다.

      ① FTM0_CH1을 사용할 것이기 때문에 PCC 레지스터에서 PORTD의 클럭을 Enable한다.

      ② PCR 레지스터의 MUX 비트를 2로 세팅하여 Alternative 2 Functionality를 사용한다.



```
void PORT_init (void)
{
    /*!
     * Pins definitions
     * ===================================================
     *
     * Pin number        | Function
     * ----------------- |------------------
     * PTD16             | FTM0CH1
     */
①PCC->PCCn[PCC_PORTD_INDEX ]|=PCC_PCCn_CGC_MASK;   /* Enable clock for PORTD */

②PORTD->PCR[16]|=PORT_PCR_MUX(2);                  /* Port D16: MUX = ALT2, FTM0CH1 */
}
```



```
#define PORT_PCR_MUX_MASK                          0x700u
#define PORT_PCR_MUX_SHIFT                         8u
#define PORT_PCR_MUX_WIDTH                         3u
#define PORT_PCR_MUX(x)                            (((uint32_t)(((uint32_t)(x))<<PORT_PCR_MUX_SHIFT))&PORT_PCR_MUX_MASK)
```

ACE Lab.

PWM Example

3. 프로그래밍

   5) FTM의 PWM을 사용하기 위한 설정을 한다. (FTM0_CH1_PWM())

      ① PCC_FTM0 레지스터를 통해 FTM의 클럭을 선택(SIRCDIV1_CLK)하고 Enable한다.

      ② SC 레지스터를 통해 FTM0 모듈의 채널을 Enable하고, Prescale factor를 세팅한다.

      ③ MOD 레지스터를 통해 MOD 값 (카운터의 최종 값)을 설정한다.

      ④ CNTIN 레지스터를 통해 CNTIN 값 (카운터의 초깃 값)을 설정한다.

      ⑤ CnSC 레지스터를 통해 Edge Align PWM, Low-True Mode로 설정한다.

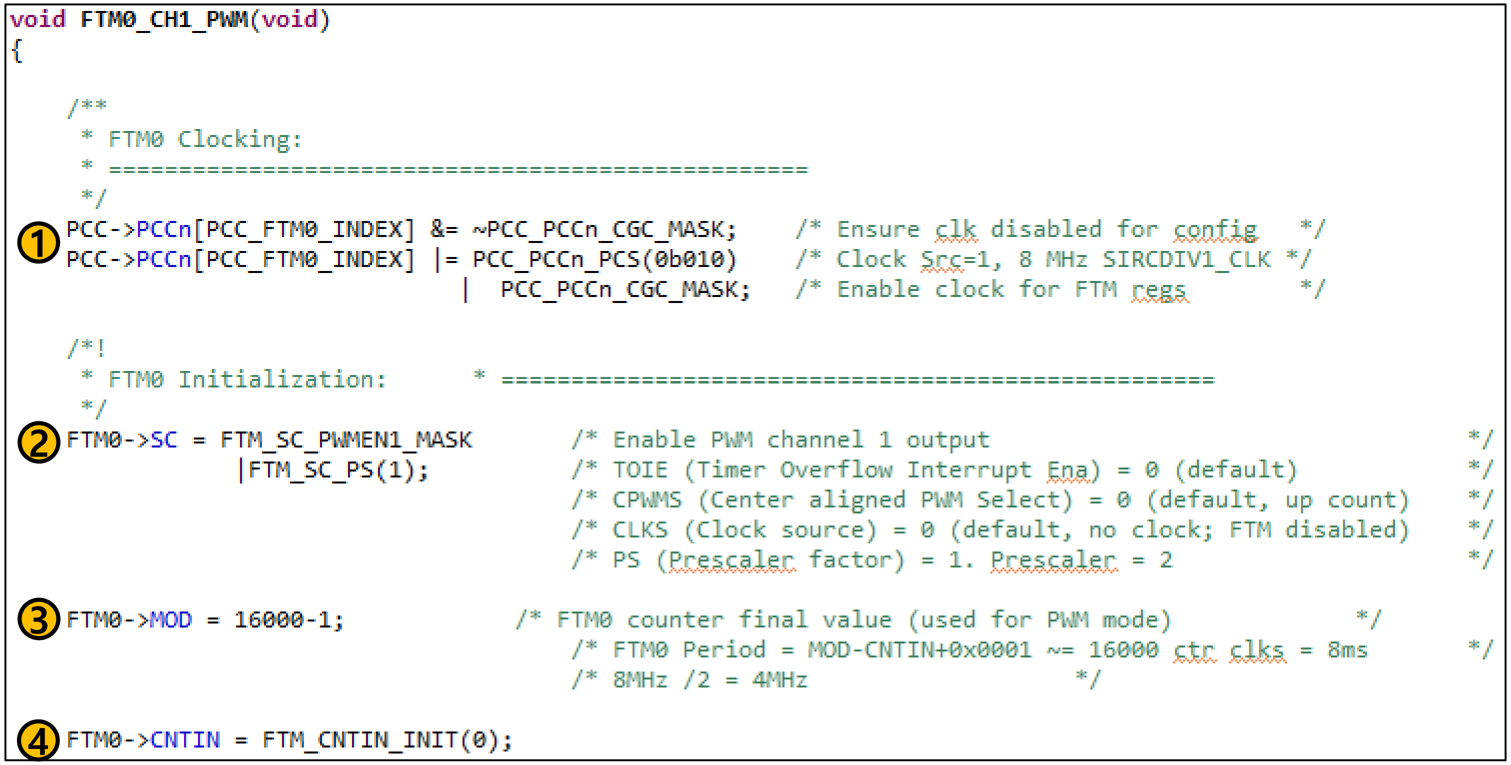
      ⑥ CnV 레지스터를 통해 CnV 값 (PWM의 Duty Cycle)을 설정한다.

      ⑦ SC 레지스터의 CLKS 비트를 3으로 세팅, external clock source로 세팅하여 FTM 모듈을 동작시킨다.

PWM Example

3. **프로그래밍**

   5) FTM0_CH1_PWM() **함수는 다음과 같다.**



```
void FTM0_CH1_PWM(void)
{

    /**
     * FTM0 Clocking:
     * ==================================================
     */
①  PCC->PCCn[PCC_FTM0_INDEX] &= ~PCC_PCCn_CGC_MASK;    /* Ensure clk disabled for config   */
    PCC->PCCn[PCC_FTM0_INDEX] |= PCC_PCCn_PCS(0b010)    /* Clock Src=1, 8 MHz SIRCDIV1_CLK */
                            |  PCC_PCCn_CGC_MASK;    /* Enable clock for FTM regs        */

    /*!
     * FTM0 Initialization:      * ==================================================
     */
②  FTM0->SC = FTM_SC_PWMEN1_MASK        /* Enable PWM channel 1 output                          */
               |FTM_SC_PS(1);           /* TOIE (Timer Overflow Interrupt Ena) = 0 (default)    */
                                        /* CPWMS (Center aligned PWM Select) = 0 (default, up count) */
                                        /* CLKS (Clock source) = 0 (default, no clock; FTM disabled) */
                                        /* PS (Prescaler factor) = 1. Prescaler = 2             */

③  FTM0->MOD = 16000-1;             /* FTM0 counter final value (used for PWM mode)        */
                                        /* FTM0 Period = MOD-CNTIN+0x0001 ~= 16000 ctr clks = 8ms   */
                                        /* 8MHz /2 = 4MHz                   */

④  FTM0->CNTIN = FTM_CNTIN_INIT(0);
```

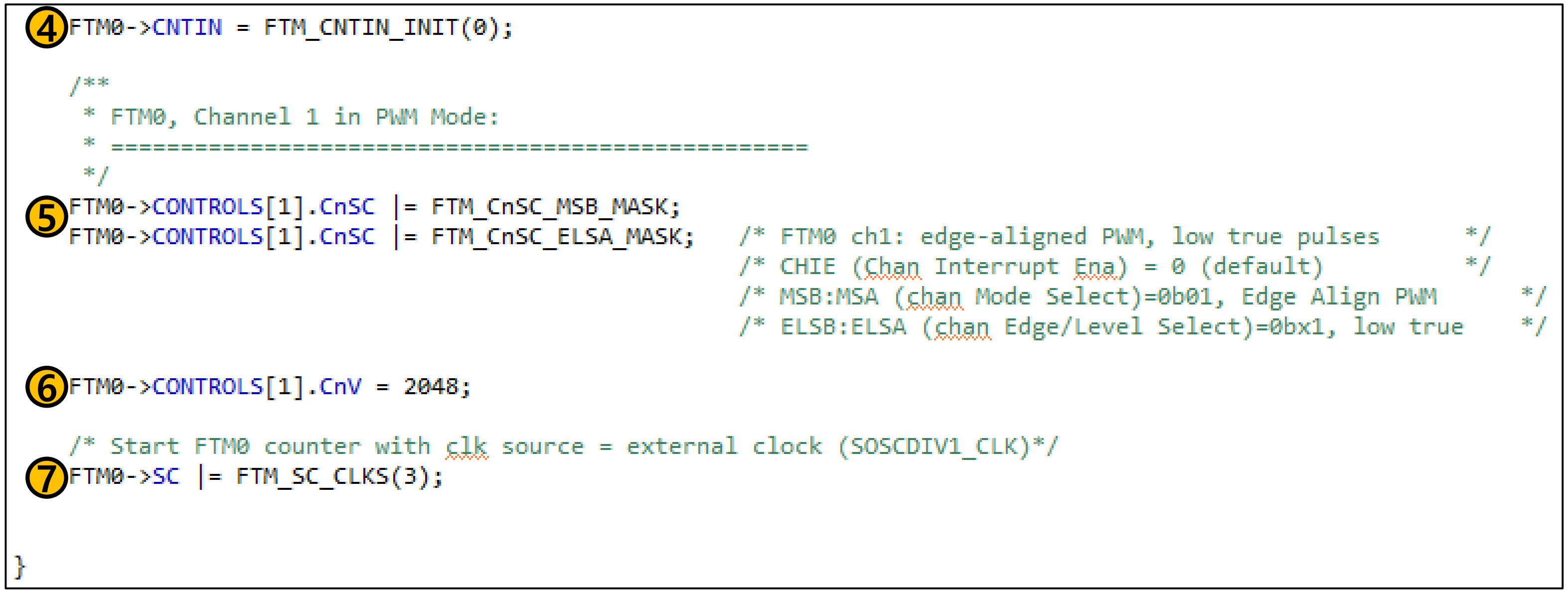
ACE Lab.

PWM Example

3. **프로그래밍**

   5) FTM0_CH1_PWM() 함수는 다음과 같다.

      ✓ 6번의 CnV 값에 따라 밝기가 바뀐다. (추가 실습에서 ADC 출력을 그대로 사용하기 위해 0~4096로 세팅)



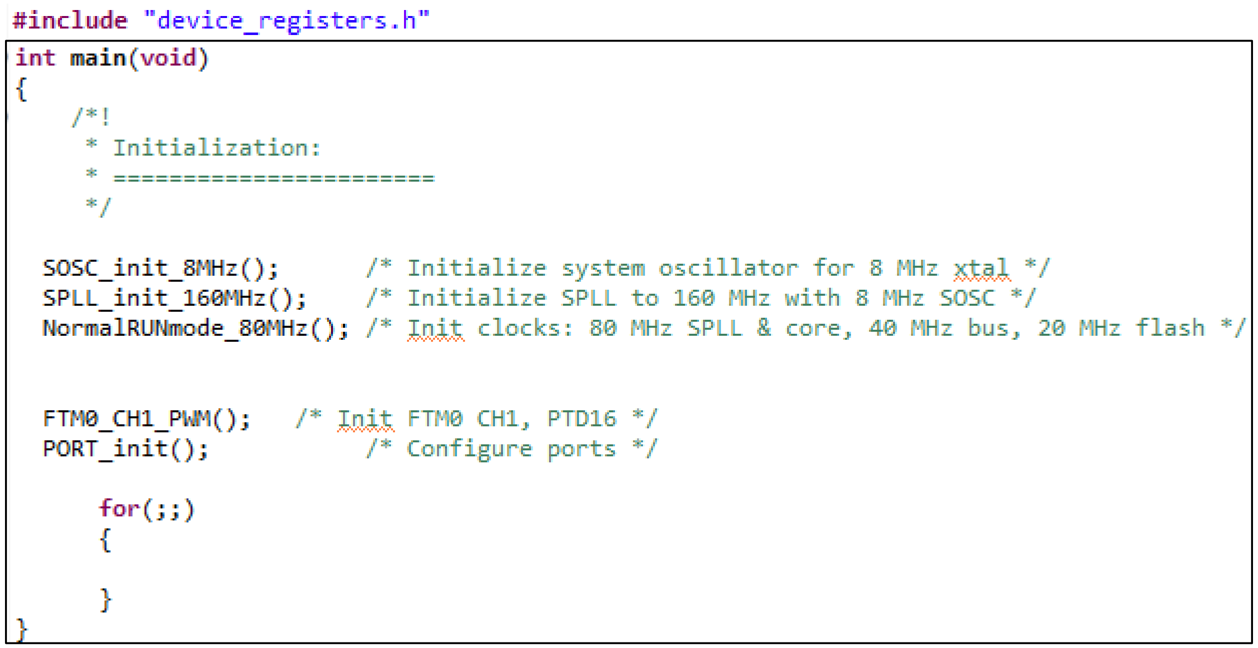
```
FTM0->CNTIN = FTM_CNTIN_INIT(0);

  /**
   * FTM0, Channel 1 in PWM Mode:
   * ==================================================
   */
FTM0->CONTROLS[1].CnSC |= FTM_CnSC_MSB_MASK;
FTM0->CONTROLS[1].CnSC |= FTM_CnSC_ELSA_MASK;   /* FTM0 ch1: edge-aligned PWM, low true pulses      */
                                                /* CHIE (Chan Interrupt Ena) = 0 (default)          */
                                                /* MSB:MSA (chan Mode Select)=0b01, Edge Align PWM      */
                                                /* ELSB:ELSA (chan Edge/Level Select)=0bx1, low true    */

FTM0->CONTROLS[1].CnV = 2048;

  /* Start FTM0 counter with clk source = external clock (SOSCDIV1_CLK)*/
FTM0->SC |= FTM_SC_CLKS(3);


}
```

ACE Lab.

PWM Example

3. 프로그래밍

   6) 동작에 따라 ‘main’ 함수를 구현한다.

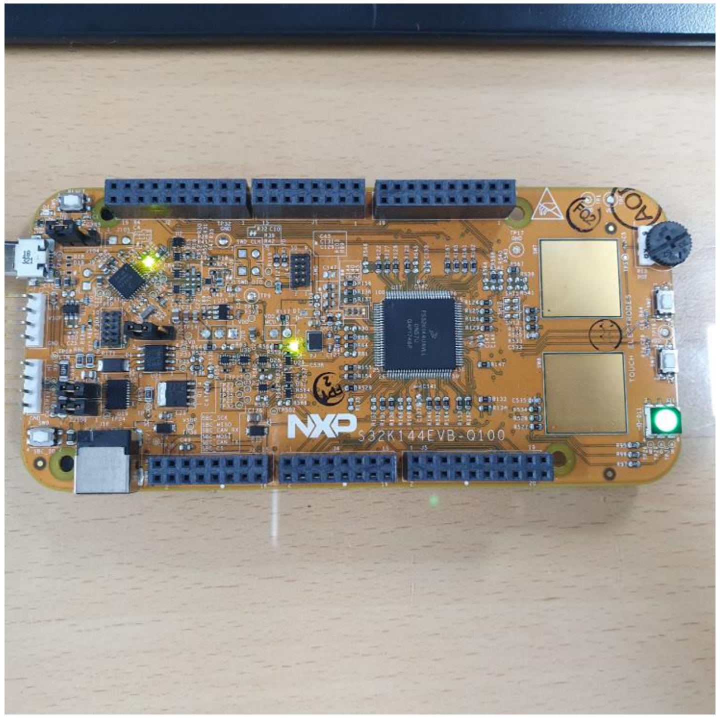


```c
#include "device_registers.h"
int main(void)
{
    /*!
     * Initialization:
     * =======================
     */

  SOSC_init_8MHz();      /* Initialize system oscillator for 8 MHz xtal */
  SPLL_init_160MHz();    /* Initialize SPLL to 160 MHz with 8 MHz SOSC */
  NormalRUNmode_80MHz(); /* Init clocks: 80 MHz SPLL & core, 40 MHz bus, 20 MHz flash */


  FTM0_CH1_PWM();   /* Init FTM0 CH1, PTD16 */
  PORT_init();           /* Configure ports */

      for(;;)
      {

      }
}
```

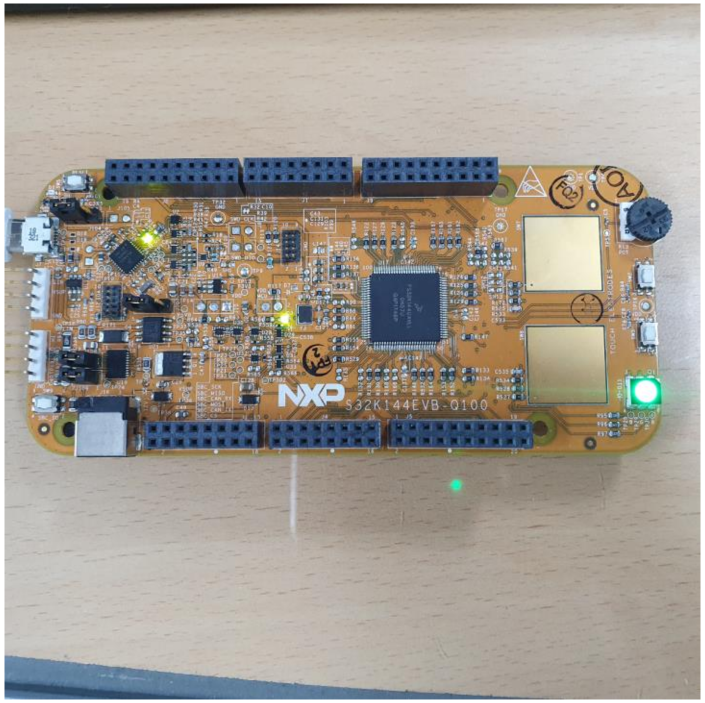
ACE Lab.

PWM Example

- 동작 모습
  - ✓ CnV 값을 바꾸면서 밝기가 바뀌는 것을 확인한다.



CnV =
1024



CnV =
4096

PWM Example

- 실습 예제 – 가변 저항에 걸리는 전압 값에 따라 LED 밝기 조절하기
  ① 지난 강의 시간에 실습한 ADC 예제를 활용한다.
  ② ADC의 해상도는 12bit로 5V 값이 0~4096에 매핑된다.
  ③ ADC로 읽는 값을 이용해 PWM Duty Cycle을 조절한다.
  ④ 가변 저항을 돌렸을 때 LED의 밝기가 바뀌는 것을 확인한다.

Q & A

Thank you for your attention

Architecture and
Compiler
for Embedded Systems Lab.

School of Electronics Engineering, KNU

ACES Lab (hn02301@gmail.com)