임베디드 기반 SW 개발 프로젝트

Driving Analog World using Digital Time-based Modulation

- PWM-based Energy Transfer Modulation for TC275 Infineon Embedded Processors-

현대자동차 입문교육
박대진 교수

실습 목표

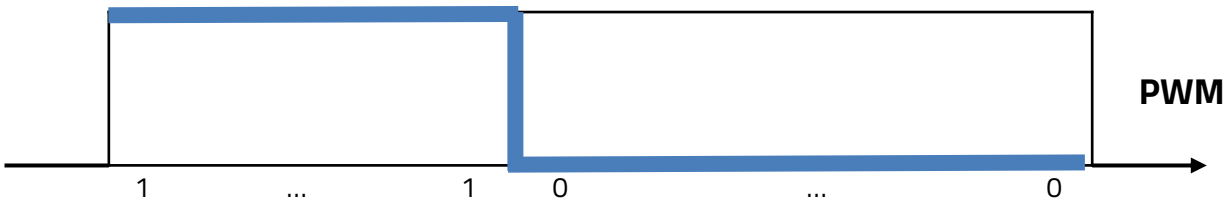
1. 확장 보드의 가변 저항에 의한 전압을 **ADC**로 읽고,
2. 샘플링된 **ADC**값의 범위에 따라 **LED RED**에 대한 **GPIO** 출력의 **PWM duty cycle**을 제어해서 **LED**의 밝기를 변화시켜 본다.

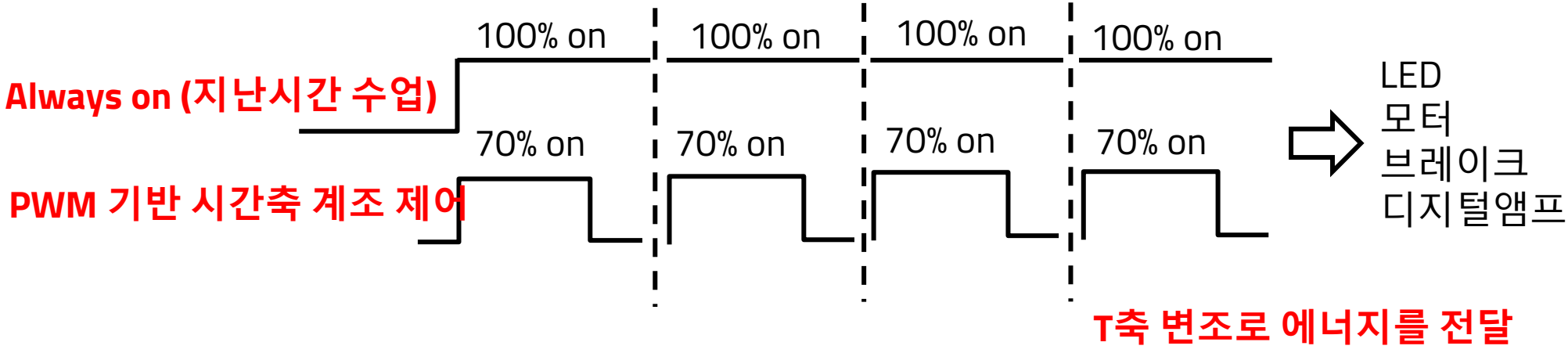
PWM (Pulse Width Modulation) 이란?

- PWM (Pulse Width Modulation) : One sample ➔ express **ratio of HIGH value** in one cycle
  = duty cycle

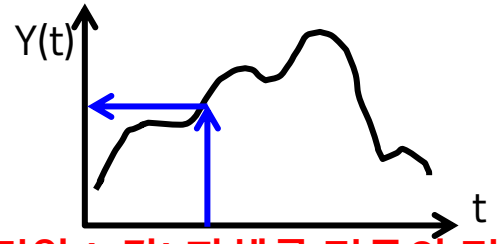
sample value : 100

256 listed '1' and '0'
(100 '1' and 156 '0')



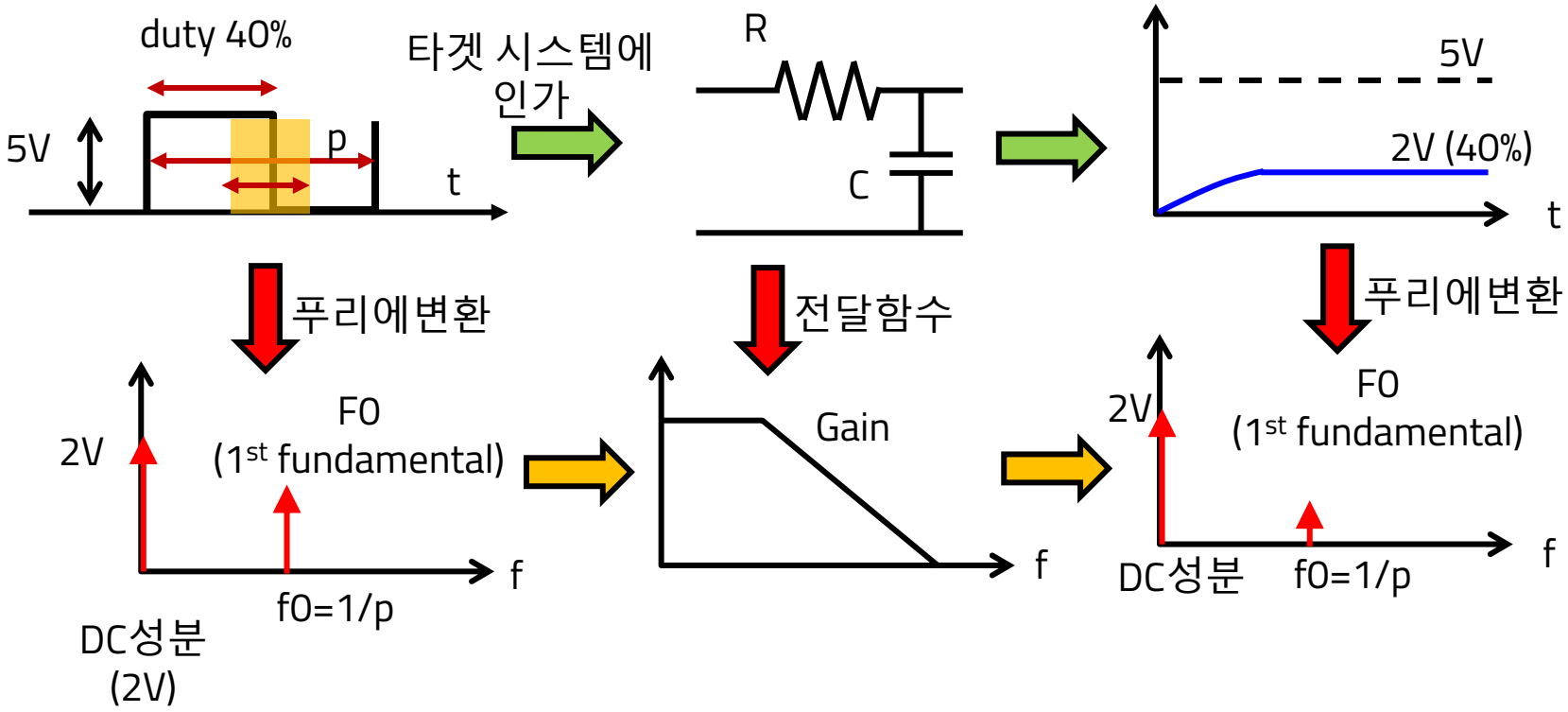


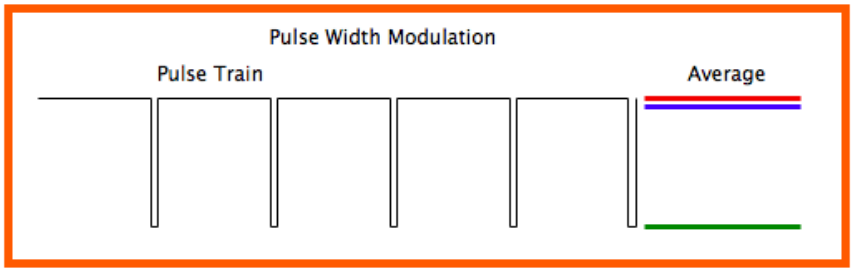
T축의 변조로, Y값을 제어하는 방법



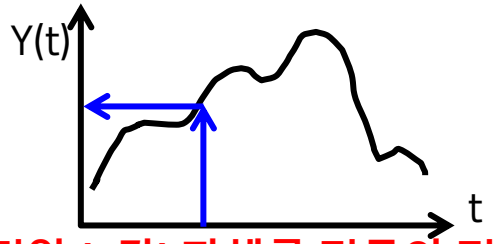
전압 Y를 고정(5V) 시켜두고
t축 Edge를modulation

전압 (Y값) 자체를 전류와 전압 제어로modulation (DAC이용)



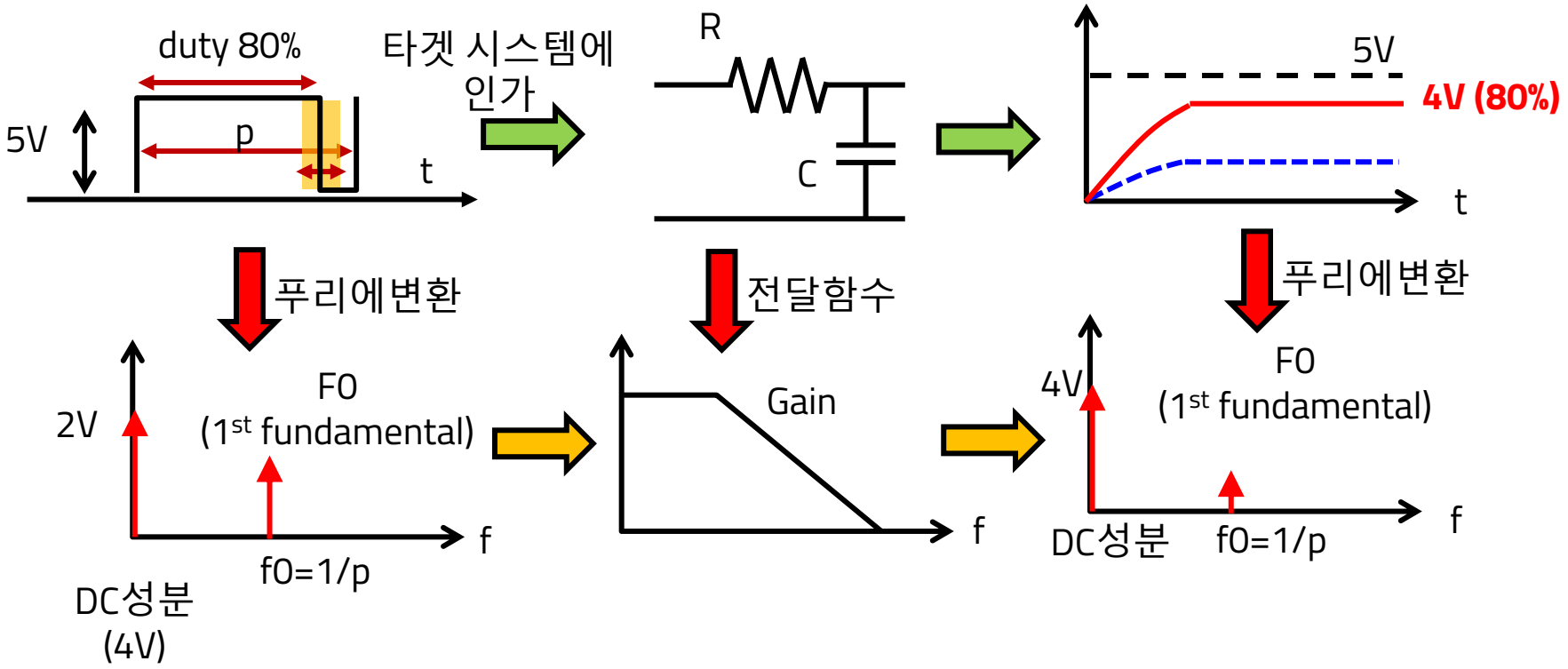


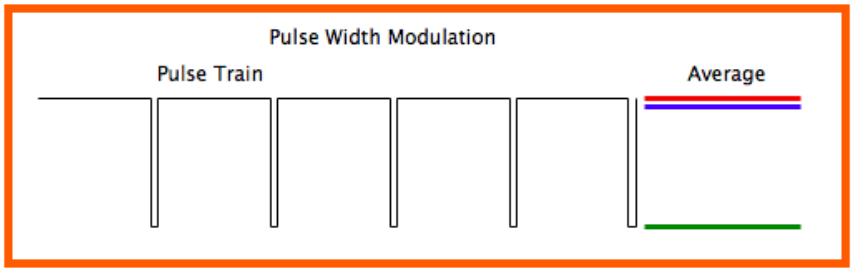
T축의 변조로, Y값을 제어하는 방법



전압 Y를 고정(5V) 시켜두고
t축 Edge를modulation

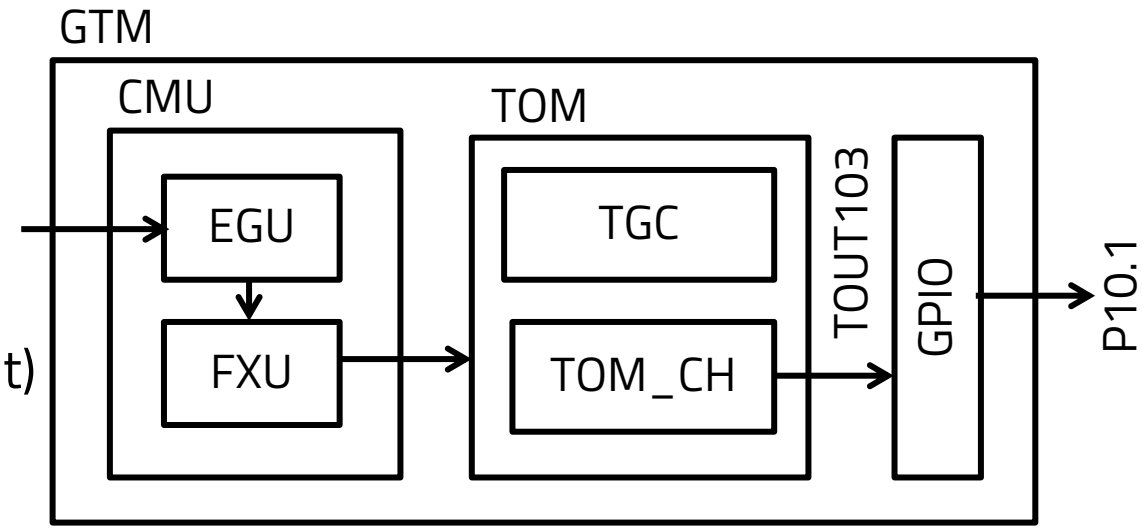
전압 (Y값) 자체를 전류와 전압 제어로modulation (DAC이용)





사용된 약어 정리

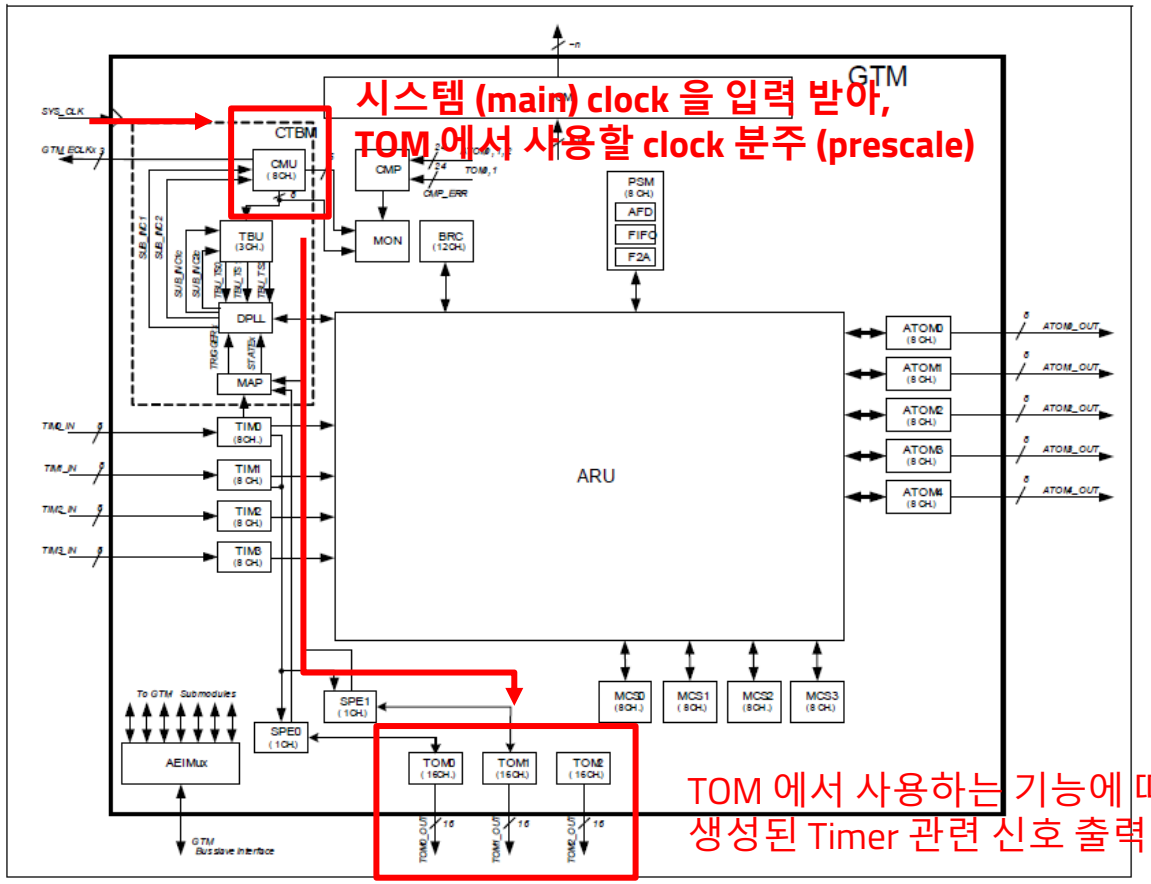
- PWM (Pulse Width Modulation)
- GTM (Generic Timer Module)
- CMU (Clock Management Unit)
- EGU (External Generation Unit)
- FXU (Fixed Clock Generation Unit)
- TOM (Timer Output Module)
- TGC (TOM Global Control Unit)



PWM 신호 생성 flow

: GTM 모듈의 시스템 clock 사용한 PWM 신호 출력

- GTM (Generic Timer Module) 에는 다양한 기능을 가지는 submodule 들이 존재
  - ATOM, BRC, MCS, PSM, SPE, TIM, TOM 등 …
- 본 실습에서 사용하는 **PWM** 기능은 **TOM (Timer Output Module)** 을 사용함



Infineon-TC27x_D-step-UM-v02_02-EN.pdf
p.2731

Figure 25-1 GTM Architecture Block Diagram

GTM (Generic Timer Module) 사용을 위한 레지스터 설정 : GTM 모듈 enable 위해서는 보호 레지스터 잠금 해제 필요

- GTM 모듈 사용을 위해 Clock Control 레지스터에서 **GTM 모듈 enable** 필요

- GTM 레지스터 항목에서
  - **CLC** 레지스터 설정 필요

- GTM_CLC 레지스터는 System Critical 레지스터이므로 CPU ENDINIT 해제 후 수정해야함
- (CPU0 만을 사용하므로) SCU 레지스터 항목에서
  - **WDTCPU0CON0** 레지스터 설정 필요

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3469

Table 25-64 Registers Overview - GTM Control Registers

Short Name	Description	Offset Addr.	Access Mode		Reset	Description See
			Read	Write		
CLC	Clock Control Register	$9FD00_H$	U, SV	SV, E, P	Application	Page 25-749
TIM0INSEL	TIM0 Input Select Register	$9FD10_H$	U, SV	U, SV, P	Application	Page 25-773

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.650

- "CE0"- writeable only when CPU0 ENDINIT bit is zero
- "CE1" - writeable only when CPU1 ENDINIT bit is zero
- "CE2" - writeable only when CPU2 ENDINIT bit is zero
- "E" - writeable when any (one or more) CPUx ENDINIT bit is zero
- "SE" - writeable only when Safety ENDINIT bit is zero
- None of the above - accessible at any time

SCU 레지스터 설정 – WDTCPU0CON0

: GTM 모듈 사용 설정 과정을 보호하는 레지스터

1. SCU 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address)
   - **= 0xF0036000**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.230

Reserved	F003 5200$_H$ - F003 5FFF$_H$	–	SPBBE	SPBBE
System Control Unit (SCU)	F003 6000$_H$ - F003 63FF$_H$	1 Kbyte	access	access
Reserved	F003 6400$_H$ F003 67FF$_H$	–	SPBBE	SPBBE
Safety Management Unit (SMU)	F003 6800	2	access	access

2. 사용할 레지스터의 주소 찾기
   - **WDTCPU0CON0** 의 Offset Address = **0x100**

→ WDTCPU0CON0 레지스터 주소 = 0xF0036000 + 0x100 = **0xF0036100**

Table 7-28 Register Overview of SCU (Offset from Main Register Base)

Short Name	Long Name	Offset Addr. 1)	Access Mode Read	Access Mode Write	Reset	Description See
EMSR	Emergency Stop Register	0FC$_H$	U, SV	SV, SE, P	Application Reset	Page 7-291
WDTCPU0CON0	CPU0 WDT Control Register 0	100$_H$	U, SV	U, SV, 32,(CPU 0[2)])	Application Reset	Page 7-276
WDTCPU0CON1	CPU0 WDT Control	104$_H$	U, SV	SV,	Application	Page

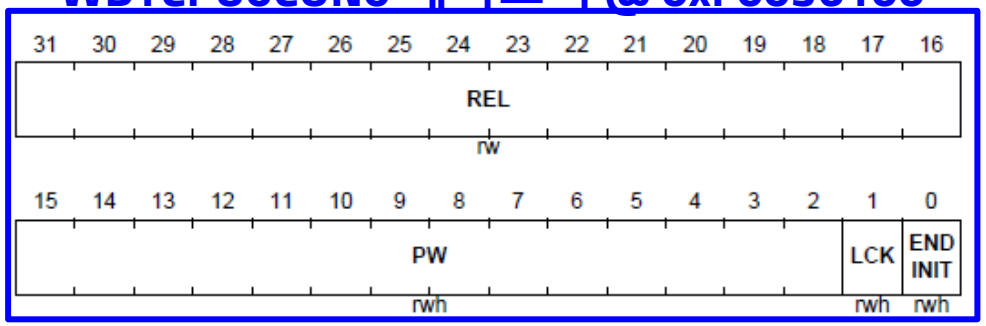
Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.697

SCU 레지스터 설정 – WDTCPU0CON0
:GTM 모듈 사용하도록 설정 위해 보호 레지스터 잠금 해제

3. **Password Access** 를 통해 WDTCPU0CON0 레지스터의 Lock 상태를 해제해야 함

1) WDTCPU0CON0 레지스터를 읽어 **WDTCPU0CON0.REL, WDTCPU0CON0.PW** 영역의 값을 확인

WDTCPU0CON0 레지스터 @ 0xF0036100



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.661

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.662

PW	[15:2]	rwh	**User-Definable Password Field for Access to WDTxCON0** This bit field is written with an initial password value during a Modify Access. A read from this bitfield returns this initial password, but bits [7:2] are inverted (toggled) to ensure that a simple read/write is not sufficient to service the WDT.

단, PW 영역의 일부 PW[7:2] 는 반전해서 읽어야 함

2) Write 할 값은 1)에서 읽은 REL 과 PW 의 조합으로 결정

[31:16] = REL, [15:2] = PW, [1] = “0”, [0] = “1”

3) 결정한 32bit 값을 WDTCPU0CON0 레지스터에 한 번에 write

4) WDTCPU0CON0 레지스터의 1번째 bit LCK 를 읽어서 Lock 상태가 해제되었는지 확인

Lock 상태가 해제되면 LCK bit 가 0으로 읽힘

Lab1: 헤더 파일 작성

:GTM 레지스터의 각 영역(필드) LSB 비트 시작 위치 define 정의

- 레지스터에 값을 write할 때 shift되는 offset을 쉽게 사용하기 위한 define 작성

```
26  ***********************************
27  #include "Ifx_Types.h"
28  #include "IfxCpu.h"
29  #include "IfxScuWdt.h"
30
31  #include "IfxCcu6_reg.h"
32  #include "IfxVadc_reg.h"
33  #include "IfxGtm_reg.h"
34
```

→ 헤더 파일 참조 추가

```
89
90  // GTM registers
91  #define DISS_BIT_LSB_IDX            1
92  #define DISR_BIT_LSB_IDX            0
93  #define SEL7_BIT_LSB_IDX            14
94  #define EN_FXCLK_BIT_LSB_IDX        22
95  #define FXCLK_SEL_BIT_LSB_IDX       0
96
97  // GTM - TOM0 registers
98  #define UPEN_CTRL1_BIT_LSB_IDX      18
99  #define HOST_TRIG_BIT_LSB_IDX       0
100 #define ENDIS_CTRL1_BIT_LSB_IDX     2
101 #define OUTEN_CTRL1_BIT_LSB_IDX     2
102 #define CLK_SRC_SR_BIT_LSB_IDX      12
103 #define OSM_BIT_LSB_IDX             26
104 #define TRIGOUT_BIT_LSB_IDX         24
105 #define SL_BIT_LSB_IDX              11
106
```

} GTM 레지스터 bit shift offset

Lab2: SCU 레지스터 설정 – WDTCPU0CON0

:GTM 모듈 사용하도록 설정 위해 보호 레지스터 잠금 해제

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

GTM 레지스터 설정 – CLC

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address)
   - **= 0xF0100000**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.232

(I2C0)	$F00D\ 00FF_H$	byte		
Reserved	$F00D\ 0100_H$ - $F00F\ FFFF_H$	–	SPBBE	SPBBE
Global Timer Module (GTM)	$F010\ 0000_H$ - $F019\ FFFF_H$	640 Kbyte	access	access
Reserved	$F01A\ 0000_H$ - $F7FF\ FFFF_H$	–	SPBBE	SPBBE
Reserved	$F800\ 0000$		SPIRE	SPIRE

2. 사용할 레지스터의 주소 찾기
   - **CLC** 의 Offset Address = **0x9FD00**

→ CLC 레지스터 주소 = 0xF0100000 + 0x9FD00 = **0xF019FD00**

Table 25-64 Registers Overview - GTM Control Registers

Short Name	Description	Offset Addr.	Access Mode		Reset	Description See
			Read	Write		
CLC	Clock Control Register	$9FD00_H$	U, SV	SV, E, P	Application	Page 25-749
TIM0INSEL	TIM0 Input Select Register	$9FD10_H$	U, SV	U, SV, P	Application	Page 25-773
TIM1INSE	TIM1 Input Select	$9FD14_H$	U, SV	U, SV,	Application	Page 25-77

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3469

GTM 레지스터 설정 – CLC

:GTM 모듈 사용 설정

3. 레지스터 write 값 결정
   - GTM 모듈을 enable 하기 위해 **DISR 영역에 0x1 write**
   - GTM 모듈을 enable 된 것을 확인하기 위해 **DISS 영역의 값이 "0"인지 확인 (만약 enable 되지 않았다면 "1"의 값 유지)**

CLC 레지스터 @ 0xF019FD00

CLC
Clock Control Register ($9FD00_H$) Reset Value: $0000\ 0003_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0												EDIS		DIS S	DIS R
r												rw	r	r	rw

Field	Bits	Type	Description
DISR	0	rw	**Module Disable Request Bit** Used for enable/disable control of the module. 0_B Module disable is not requested. 1_B Module disable is requested.
DISS	1	rh	**Module Disable Status Bit** Bit indicates the current status of the module. 0_B Module is enabled. 1_B Module is disabled.
EDIS	3	rw	**Sleep Mode Enable Control**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3476

Lab3: GTM 레지스터 설정 – CLC

:GTM 모듈 사용 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

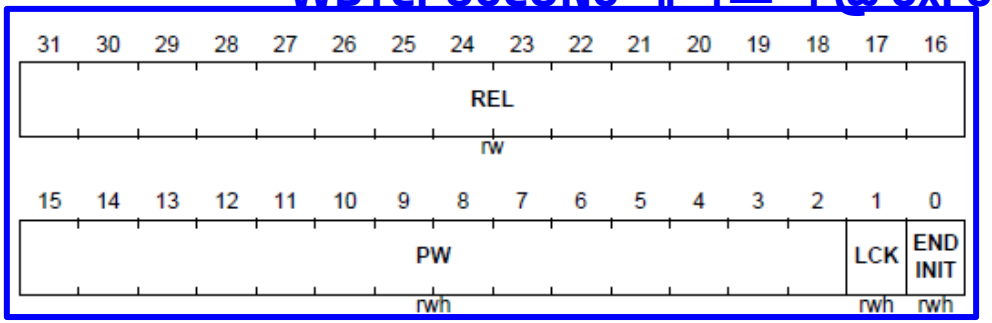
SCU 레지스터 설정 – WDTCPU0CON0

:GTM 모듈 사용 설정 후, 보호 레지스터 잠금 설정

4. Modify Access 를 통해 WDTCPU0CON0 레지스터의 CPU0 ENDINIT을 set/clear 함

1) WDTCPU0CON0 레지스터를 읽어 **WDTCPU0CON0.REL, WDTCPU0CON0.PW** 영역의 값을 확인

WDTCPU0CON0 레지스터 @ 0xF0036100



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.661

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.662

PW	[15:2]	rwh	**User-Definable Password Field for Access to WDTxCON0** This bit field is written with an initial password value during a Modify Access. ~~A read from this bitfield returns this initial password,~~ but bits [7:2] are inverted (toggled) to ensure that a ~~simple read/write is not sufficient to service the WDT.~~

단, PW 영역의 일부 PW[7:2] 는 반전해서 읽어야 함

2) Write 할 값은 1)에서 읽은 REL 과 PW 의 조합으로 결정

[31:16] = REL, [15:2] = PW, [1] = "1"

3) 0번째 bit **ENDINIT을 설정하려면 [0] = "1", 해제하려면 "0" 값을 write**

4) 결정한 32bit 값을 WDTCPU0CON0 레지스터에 한 번에 write

5) WDTCPU0CON0 레지스터의 1번째 bit LCK 를 읽어서 Lock 상태가 설정되었는지 확인

Lock 상태가 설정되면 LCK bit 가 1로 읽힘

5. Modify Access 를 통해 CPU0 ENDINIT을 해제하면 레지스터 수정 후 반드시 CPU0 ENDINIT을 재설정해줘야 함

Lab4: SCU 레지스터 설정 – WDTCPU0CON0

:GTM 모듈 사용 설정 후, 보호 레지스터 잠금 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

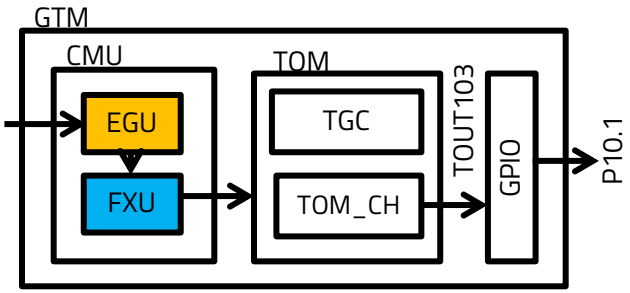
PWM 신호 생성 flow

:CMU 내부에 FXU 에서 TOM이 사용할 clock 생성

- GTM – TOM 에서 사용되는 clock 신호의 생성은 **CMU (Clock Management Unit)** 가 담당

- TOM은 CMU 에서 생성되는 clock 신호 중, **FXU (Fixed Clock Generation Unit)** 에서 생성되는 **clock** 들을 사용함

→ FXU 에 입력되는 clock 신호, FXU 에서 TOM으로 공급되는 clock 신호에 대한 설정 필요



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2829

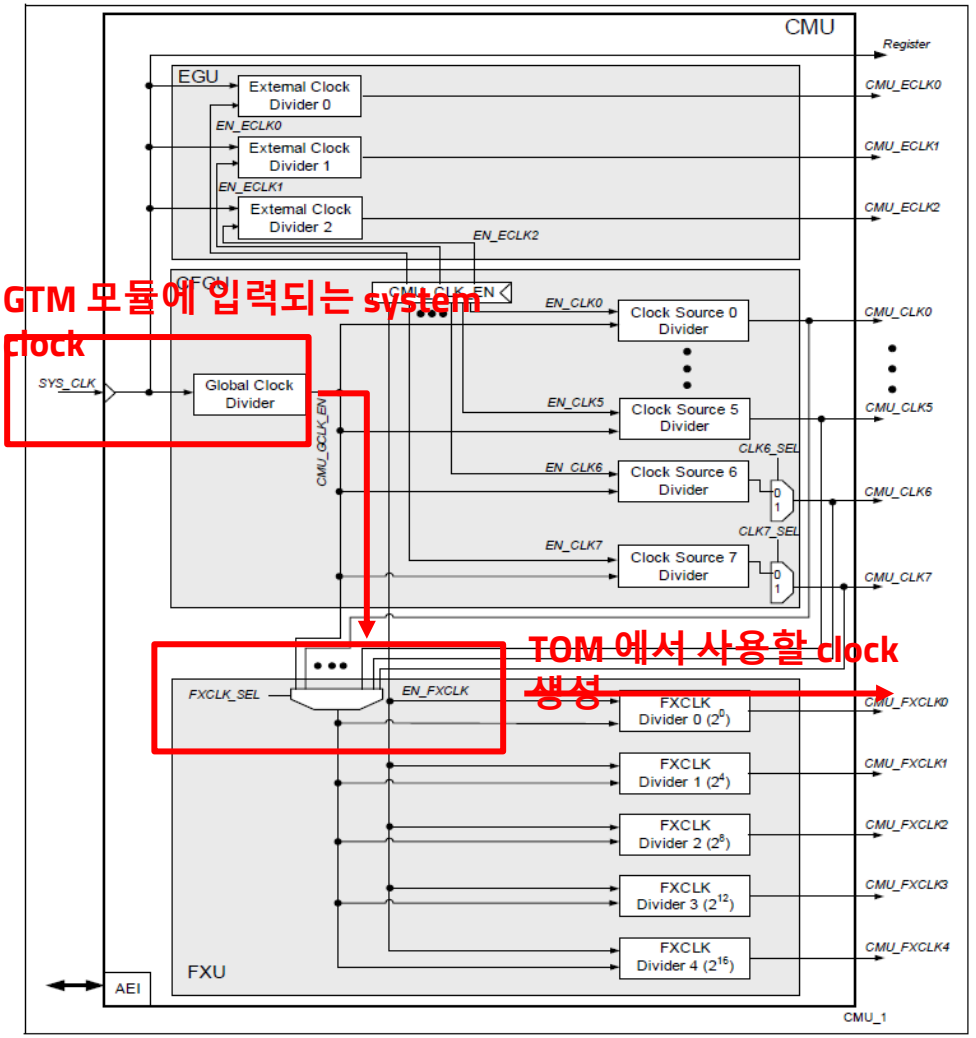


Figure 25-17 CMU Block Diagram

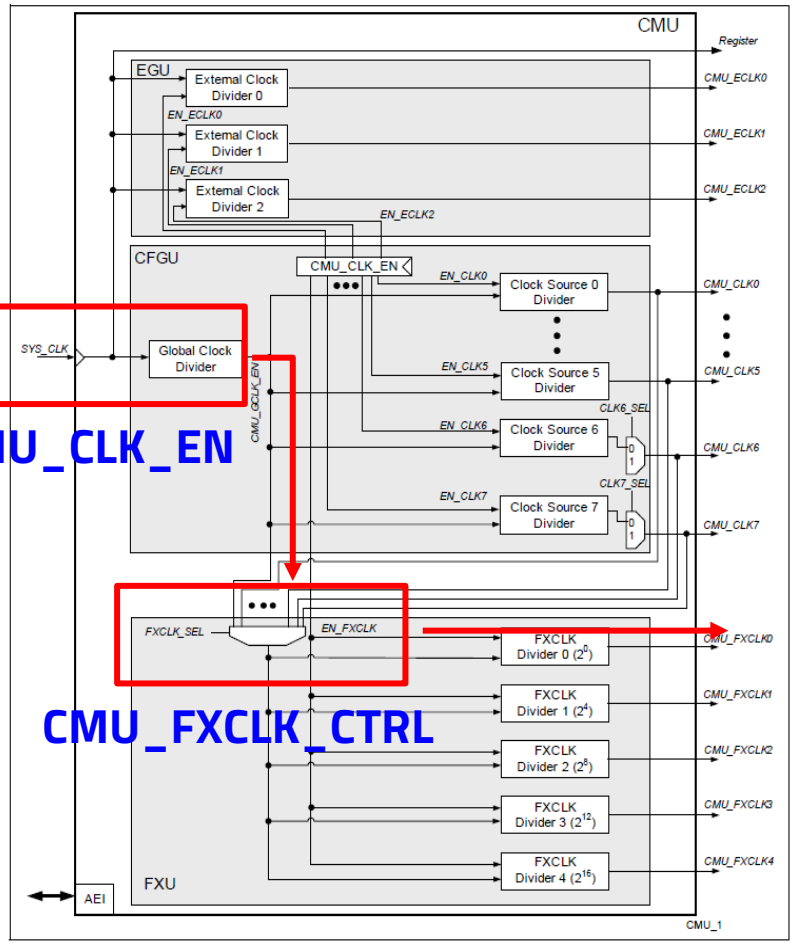
GTM 사용을 위한 레지스터 설정
:FXU 에서 GTM 내부 TOM이 사용할 clock 생성

- GTM 모듈에서 TOM 으로 clock 을 전달해주는 역할은 CMU 가 담당하며, **TOM은 FXU에서 생성하는 FXCLK를 사용함**

→ **FXU가 사용할 Clock, FXCLK의 주파수, 번호 등의 결정 필요**

- GTM 레지스터 항목에서
  - **CMU_CLK_EN** 레지스터 설정 필요
  - **CMU_FXCLK_CTRL** 레지스터 설정 필요

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2829



CMU_CLK_EN

CMU_FXCLK_CTRL

Figure 25-17 CMU Block Diagram

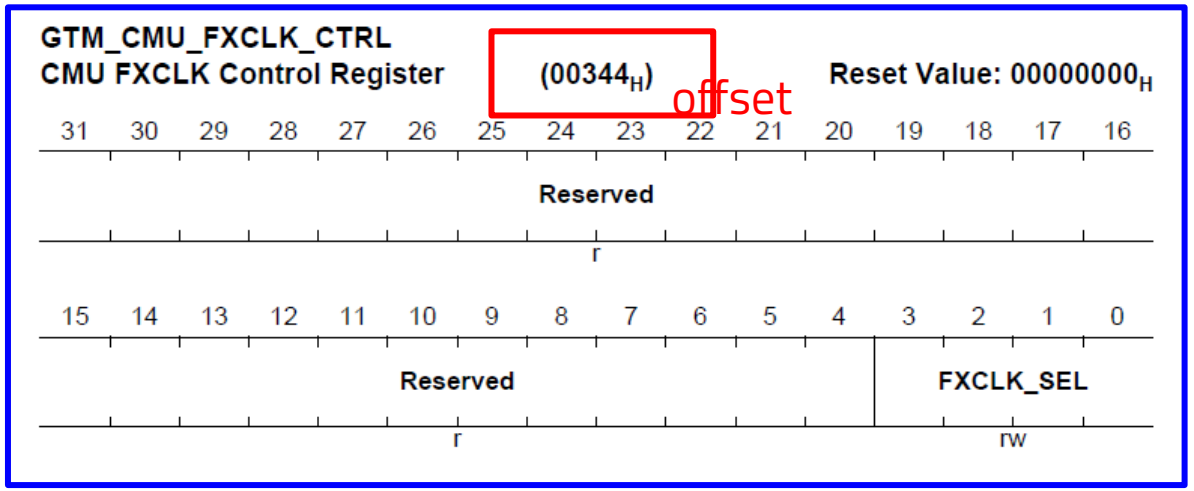
GTM 레지스터 설정 – CMU_FXCLK_CTRL

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기
   - **CMU_FXCLK_CTRL** 의 Offset Address = **0x344**

   → CMU_FXCLK_CTRL 레지스터 주소 = 0xF0100000 + 0x344 = **0xF0100344**

CMU_FXCLK_CTRL 레지스터 @ 0xF0100344



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2844

GTM 레지스터 설정 – CMU_FXCLK_CTRL

:FXU에서 생성할 clock 에 대한 설정

3. 레지스터 write 값 결정
   - CMU 내부의 FXU 에 대한 clock으로 GTM 모듈의 입력 clock인 **CMU_GCLK_EN** 을 사용하기 위해 **FXCLK_SEL 영역에 0x0 write**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2829

CMU_FXCLK_CTRL 레지스터 @ 0xF0100344

GTM_CMU_FXCLK_CTRL
CMU FXCLK Control Register (00344$_H$) **Reset Value: 00000000$_H$**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved												FXCLK_SEL			
r												rw			

Field	Bits	Type	Description
FXCLK_SEL	[3:0]	rw	**Input clock selection for EN_FXCLK line** 0000_BCMU_GCLK_EN selected **100MHz** 0001_BCMU_CLK0 selected 0010_BCMU_CLK1 selected 0011_BCMU_CLK2 selected 0100_BCMU_CLK3 selected 0101_BCMU_CLK4 selected 0110_BCMU_CLK5 selected 0111_BCMU_CLK6 selected 1000_BCMU_CLK7 selected *Note: This value can only be written, when the CMU_FXCLK generation is disabled. See bits 23…22 in register CMU_CLK_EN.* *Note: Other values for FXCLK_SEL are reserved and should not be used. but they behave like*

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2844

GTM 모듈에 입력되는 system clock을 FXU로 입력

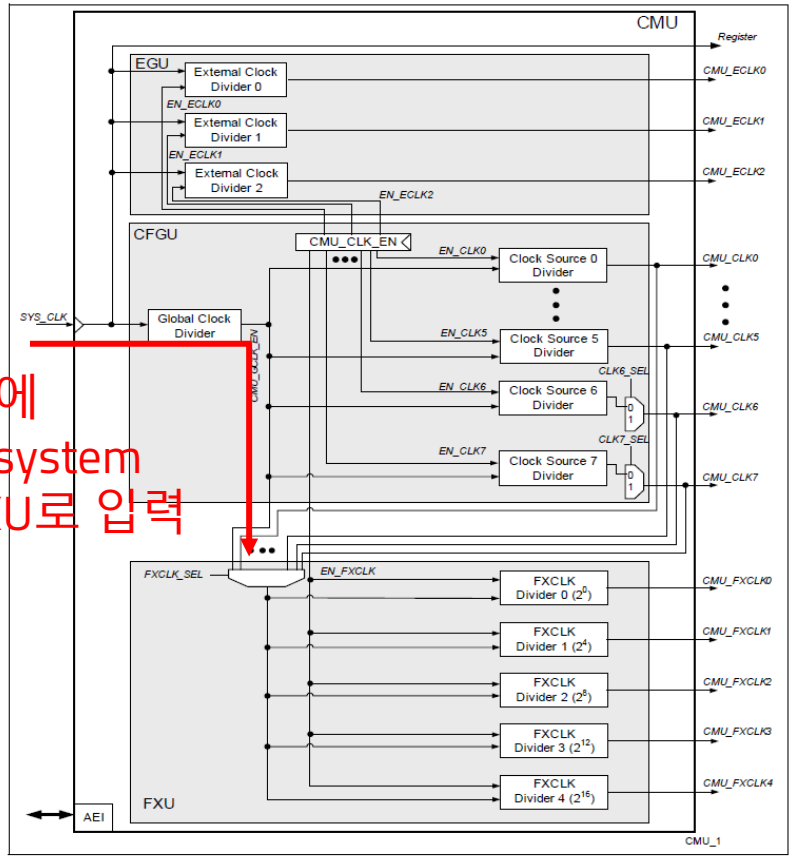


Figure 25-17 CMU Block Diagram

Lab5: GTM 레지스터 설정 – CMU_FXCLK_CTRL

:FXU에서 생성할 clock 에 대한 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
95  #define FXCLK_SEL_BIT_LSB_IDX       0
```

GTM 레지스터 설정 – CMU_CLK_EN

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기
   - **CMU_CLK_EN** 의 Offset Address = **0x300**

   → CMU_CLK_EN 레지스터 주소 = 0xF0100000 + 0x300 = **0xF0100300**

CMU_CLK_EN 레지스터 @ 0xF0100300

GTM_CMU_CLK_EN
CMU Clock Enable Register

(00300$_H$) offset

Reset Value: 00000000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								EN_FXCLK		EN_ECLK2		EN_ECLK1		EN_ECLK0	
r								rw		rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN_CLK7		EN_CLK6		EN_CLK5		EN_CLK4		EN_CLK3		EN_CLK2		EN_CLK1		EN_CLK0	
rw		rw		rw		rw		rw		rw		rw		rw	

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2835

GTM 레지스터 설정 – CMU_CLK_EN

:FXU에서 생성할 clock 에 대한 설정

3. 레지스터 write 값 결정
   - TOM 은 FXU 에서 생성하는 FXCLK 를 사용
   - FXCLK를 사용하기 위해 **EN_FXCLK_SEL 영역에 0x2 write**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2829

CMU_CLK_EN 레지스터 @ 0xF0100300

GTM_CMU_CLK_EN
CMU Clock Enable Register (00300$_H$) **Reset Value: 00000000$_H$**

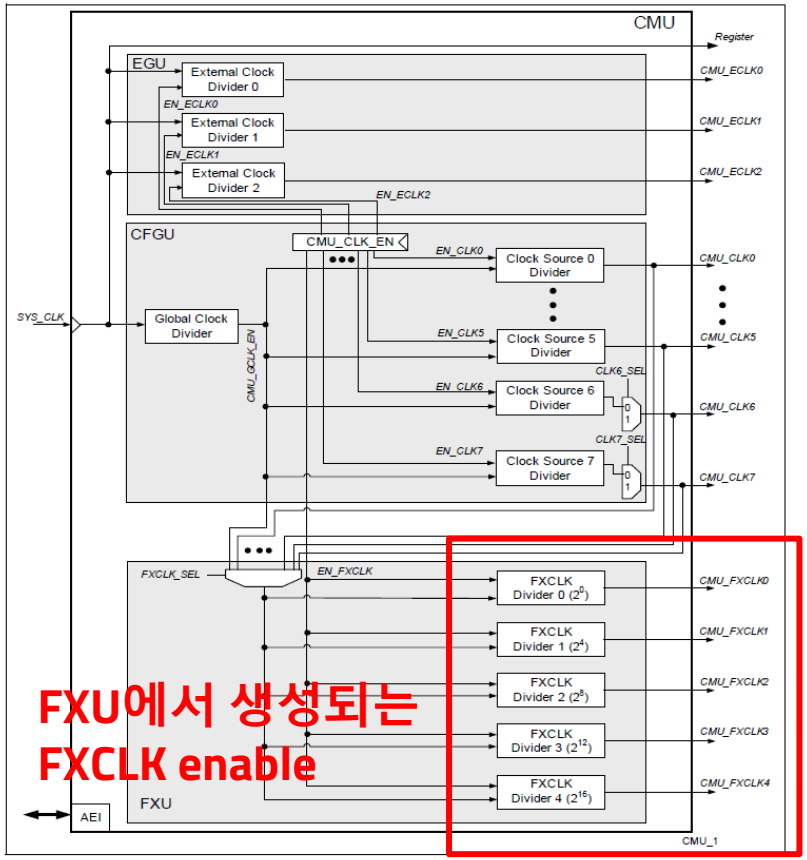
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								EN_FXCLK		EN_ECLK2		EN_ECLK1		EN_ECLK0	
r								rw		rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EN_CLK7		EN_CLK6		EN_CLK5		EN_CLK4		EN_CLK3		EN_CLK2		EN_CLK1		EN_CLK0	
rw		rw		rw		rw		rw		rw		rw		rw	

2			see bits [1:0]
EN_FXCLK	[23:22]	rw	**Enable all CMU_FXCLK** see bits [1:0]
			Note: An enable to EN_FXCLK from disable state will be reset internal fixed clock counters.
Reserved	[31:24]	r	Reserved

00_B clock source is disabled (ignore write access)
01_B disable clock signal and reset internal states
10_B enable clock signal
11_B clock signal enabled (ignore write access)

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2835



FXU에서 생성되는 FXCLK enable

Figure 25-17 CMU Block Diagram

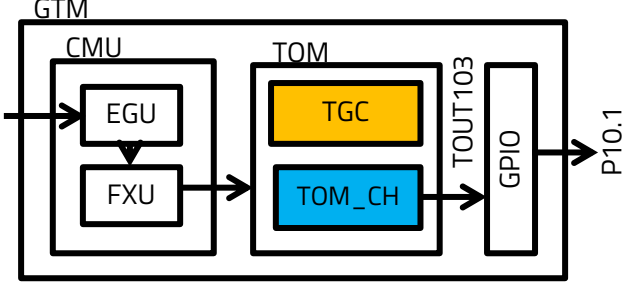
Lab6: GTM 레지스터 설정 – CMU_CLK_EN

:FXU에서 생성할 clock 에 대한 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
94  #define EN_FXCLK_BIT_LSB_IDX        22
```

PWM 신호 생성 flow
:TOM에서 PWM 출력 신호 생성



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2918

- PWM 기능을 사용하기 위한 TOM의 구조

- TOM은 TGC (TOM Global Channel Control)의 제어를 받아 출력 신호를 생성

→ TOM의 기능 중, PWM 기능을 사용하기 위한 설정 필요

→ TOM의 출력은 MCU의 핀으로 바로 출력 가능, <u>사용하려는 핀이 **TOM**의 어떤 **Channel**과 연결되어 있는지 확인 필요</u>

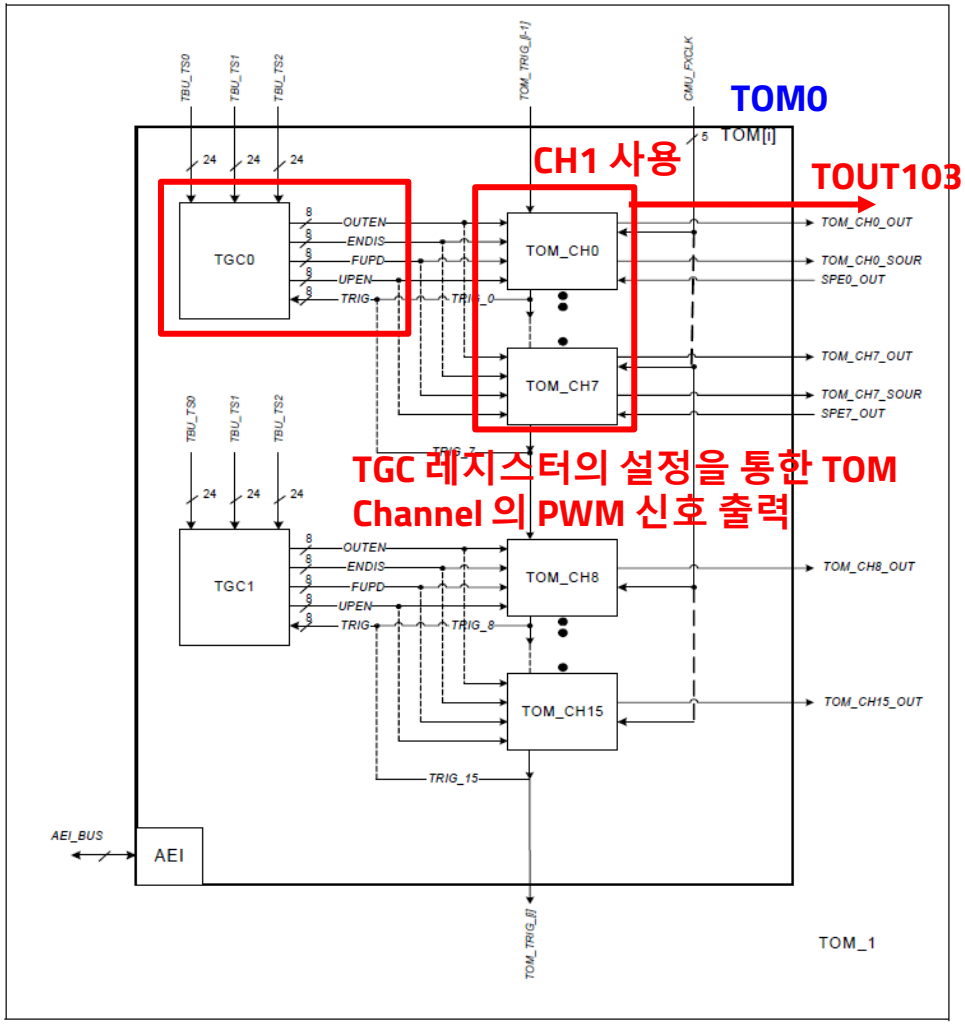


Figure 25-32 TOM Block diagram

GTM 사용을 위한 레지스터 설정
:GPIO 출력을 GTM 출력으로 사용하는 설정

- PWM 신호를 통해 밝기를 제어할 LED RED가 연결된 핀은 **P10.1**

Table 13-16 Port 10 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P10.1	I	General-purpose input	P10_IN.P1	P10_IOCR0. PC1	$0XXXX_B$
		GTM input	TIN103		
		QSPI1 input	MRST1A		
		GPT120 input	T5EUDB		
	O	General-purpose output	P10_OUT.P1		$1X000_B$
		GTM output	TOUT103		$1X001_B$
		QSPI1 output	MTSR1		$1X010_B$
		QSPI1 output	MRST1		$1X011_B$
		MSC0	EN01		$1X100_B$
		VADC output	VADCG6BFL1		$1X101_B$
		MSC0 output	END03		$1X110_B$
		Reserved	–		$1X111_B$

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.1163

P10.1 핀의 GPIO 출력을 GTM 출력으로 사용하기 위해 →
포트의 I/O Control 레지스터 설정 필요

- GPIO P10 레지스터 항목에서
  - **IOCR0** 레지스터 설정 필요

GPIO 레지스터 설정 – P10 Address 계산

: Port 10의 각 레지스터가 위치한 주소 확인

3. 사용할 특정 레지스터의 주소 찾기
   - **P10_IOCR0** Offset Address = 0x0010

   → P10_IOCR0 레지스터 주소

   → = 0xF003B000 + 0x0010 = **0xF003B010**

Table 13-4 Registers Overview

Register Short Name	Register Long Name	Offset Address	Access Mode Read	Access Mode Write	Reset	Desc. see
Pn_OUT	Port n Output Register	0000_H	U, SV	U, SV, P	Application Reset	Page 13-38
Pn_OMR	Port n Output Modification Register	0004_H	U, SV	U, SV, P	Application Reset	Page 13-39
ID	Module Identification Register	0008_H	U, SV	BE	Application Reset	Page 13-13
Pn_IOCR0	Port n Input/Output Control Register 0	0010_H	U, SV	U, SV, P	Application Reset	Page 13-14
Pn_IOCR4	Port n Input/Output Control Register 4	0014_H	U, SV	U,	Application	Page 13-1

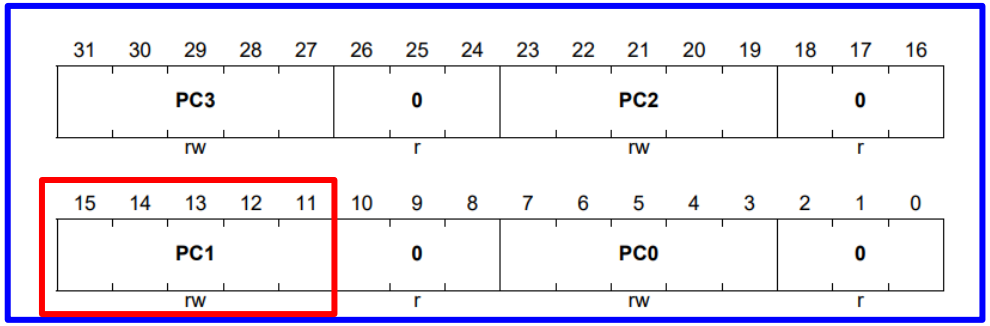
Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.1074

GPIO 레지스터 설정 – P10 I/O 모드

: Port 10의 핀 1을 GTM 출력 모드로 사용

- **P10.1** 을 General Purpose Output (push-pull)가 아닌 **GTM** 모듈의 출력으로 사용하기 위해 P10_IOCR0 레지스터에 어떤 값을 써야 하는지 확인 → **0x11 write**
  (예전에 단순 GPIO출력으로 쓸 때는 0x10를 썼다)

P10_IOCR0 레지스터 @ 0xF003B010



Field	Bits	Type	Description
PC0, PC1, PC2, PC3	[7:3], [15:11], [23:19], [31:27]	rw	**Port Control for Port n Pin 0 to 3** This bit field determines the Port n line x functionality (x = 0-3) according to the coding table (see Table 13-5).
0	[2:0], [10:8], [18:16], [26:24]	r	**Reserved** Read as 0; should be written with 0.

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.1080

Table 13-16 Port 10 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select. Reg./Bit Field	Port I/O Control Select. Value
P10.1	I	General-purpose input	P10_IN.P1	P10_IOCR0. PC1	$0XXXX_B$
		GTM input	TIN103		
		QSPI1 input	MRST1A		
		GPT120 input	T5EUDB		
	O	General-purpose output	P10_OUT.P1		$1X000_B$
		GTM output	TOUT103		$1X001_B$
		QSPI1 output	MTSR1		$1X010_B$
		QSPI1 output	MRST1		$1X011_B$
		MSC0	EN01		$1X100_B$
		VADC output	VADCG6BFL1		$1X101_B$
		MSC0 output	END03		$1X110_B$
		Reserved	–		$1X111_B$

what?
- GTM 출력 모드로 사용하기 위해 0x11 (10001b) 값을 PC1 영역에 write

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.1163

Lab7: P10 레지스터 설정 – IOCR0

:GPIO 출력 모드를 GTM 생성 출력으로 설정

```
void initLED(void)
{
    P10_IOCR0.U &= ~(0x1F << PC1_BIT_LSB_IDX);      // reset P10_IOCR0 PC1
    P10_IOCR0.U &= ~(0x1F << PC2_BIT_LSB_IDX);      // reset P10_IOCR0 PC2

    P10_IOCR0.U |= 0x10 << PC1_BIT_LSB_IDX;         // set P10.1 push-pull general output
    P10_IOCR0.U |= 0x10 << PC2_BIT_LSB_IDX;         // set P10.2 push-pull general output
}


void initPWMLED(void)
{
    P10_IOCR0.U &= ~(0x1F << PC1_BIT_LSB_IDX);      // reset P10_IOCR0 PC1
    P10_IOCR0.U &= ~(0x1F << PC2_BIT_LSB_IDX);      // reset P10_IOCR0 PC2

    P10_IOCR0.U |= 0x11 << PC1_BIT_LSB_IDX;         // set P10.1 GTM Out, (PWM Generation)
    P10_IOCR0.U |= 0x10 << PC2_BIT_LSB_IDX;         // set P10.2 push-pull general output
}
```

GTM 사용을 위한 레지스터 설정
:GTM 모듈의 PWM 출력을 MCU 외부 핀으로 연결

- PWM 신호를 통해 밝기를 제어할 LED RED가 연결된 핀은 **P10.1**
- 핀 P10.1 에 **GPIO 대신 GTM 출력으로 연결 필요 → TOUT103**

Table 13-16 Port 10 Functions (cont'd)

Port Pin	I/O	Pin Functionality	Associated Reg./ I/O Line	Port I/O Control Select.	
				Reg./Bit Field	Value
P10.1	I	General-purpose input	P10_IN.P1	P10_IOCR0. PC1	$0XXXX_B$
		GTM input	TIN103		
		QSPI1 input	MRST1A		
		GPT120 input	T5EUDB		
	O	General-purpose output	P10_OUT.P1		$1X000_B$
		GTM output	TOUT103		$1X001_B$
		QSPI1 output	MTSR1		$1X010_B$
		QSPI1 output	MRST1		$1X011_B$
		MSC0	EN01		$1X100_B$
		VADC output	VADCG6BFL1		$1X101_B$
		MSC0 output	END03		$1X110_B$
		Reserved	–		$1X111_B$

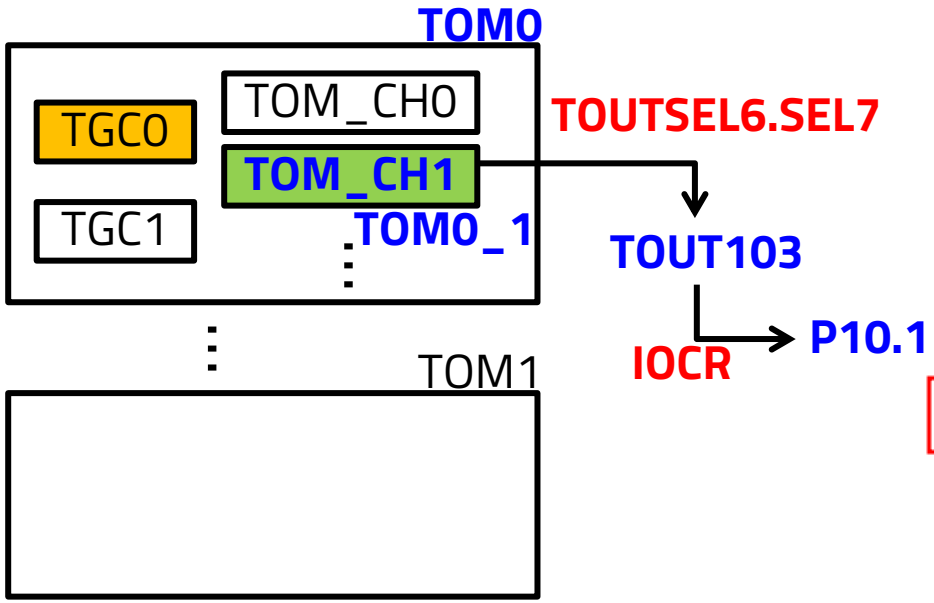
→ **GTM 출력으로 사용하기 위해 TOUT103 관련 레지스터 설정 필요**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.1163

GTM 사용을 위한 레지스터 설정
:사용하는 GPIO 핀이 TOM의 어느 출력과 연결되는지 확인

- 앞서 핀 P10.1 에 연결되는 GTM의 출력은 TOUT103 인 것을 확인
- TOUT 출력은 GTM – TOM 에서 사용하는 TOM 번호와 채널이 정해져 있음

→ P10.1 은 TOUT103



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3484

Table 25-67 GTM to Port Mapping for QFP-176

Port	Input	Output	Input Timer Mapped		Output Timer Mapped			
			A	B	A	B	C	D
P02.2	TIN2	TOUT2	TIM0_2	TIM1_2	TOM0_10	TOM1_10	ATOM 0_2	ATOM 1_2
P02.3	TIN3	TOUT3	TIM0_3	TIM1_3	TOM0_11	TOM1_11	ATOM 0_3	ATOM 1_3
P02.4	TIN4	TOUT4	TIM0_4	TIM1_4	TOM0_12	TOM1_12	ATOM 0_4	ATOM 1_4
P02.5	TIN5	TOUT5	TIM0_5	TIM1_5	TOM0_13	TOM1_13	ATOM 0_5	ATOM 1_5
P02.6	TIN6	TOUT6	TIM0_6	TIM1_6	TOM0_14	TOM1_14	ATOM 0_6	ATOM 1_6
P02.7	TIN7	TOUT7	TIM0_7	TIM1_7	TOM0_15	TOM1_15	ATOM 0_7	ATOM 1_7
P02.8	TIN8	TOUT8	TIM2_0	TIM3_0	TOM0_8	TOM1_0	ATOM 0_0	ATOM 1_0
P10.0	TIN102	TOUT102	TIM0_4	TIM1_4	TOM0_4	TOM2_12	ATOM 1_4	ATOM 4_4
P10.1	TIN103	TOUT103	TIM0_1	TIM1_1	TOM0_1	TOM2_9	ATOM 1_1	ATOM 4_1
P10.2	TIN104	TOUT104	TIM0_2	TIM1_2	TOM0_2	TOM2_10	ATOM	ATOM

GTM 레지스터 설정

: Timer Output Selection Register (TOUTSELx)

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기
   - **1개의 TOUTSELx 레지스터는 16개의 TOUT 핀을 제어함**
   - TOUT0 부터 16개씩 묶을 때 핀 **TOUT103** (P10.1)은 **TOUTSEL6**에 포함되어 있음(**TOUT96~ TOUT111 안에 TOU103이 포함)**
   - **TOUTSEL6** 의 Offset Address = **0x9FD48**

   → TOUTSEL6 레지스터 주소 = 0xF0100000 + 0x9FD48 = **0xF019FD48**

TOUTSEL0 → TOUT00 ~~ TOUT15
TOUTSEL1 → TOUT16 ~~ TOUT31
TOUTSEL2 → TOUT32 ~~ TOUT47
TOUTSEL3 → TOUT48 ~~ TOUT63
TOUTSEL4 → TOUT64 ~~ TOUT79
TOUTSEL5 → TOUT80 ~~ TOUT95
TOUTSEL6 → TOUT96 ~~ TOUT103

Table 25-64 Registers Overview - GTM Control Registers

Short Name	Description	Offset Addr.	Access Mode		Reset	Description See
			Read	Write		
TOUTSEL 5	Timer Output Select 5 Register	$9FD44_H$	U, SV	U, SV, P	Application	Page 25-814
TOUTSEL 6	Timer Output Select 6 Register	$9FD48_H$	U, SV	U, SV, P	Application	Page 25-814
TOUTSEL	Timer Output Select 7	$9FD4C_H$	U, SV	U, SV,	Application	Page 25-81

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3470

GTM 레지스터 설정 – TOUTSEL6
:TOM에서 출력되는 PWM 신호를 MCU 핀에 연결 설정

3. 레지스터 write 값 결정
   - TOM0 채널1로부터 생성된 PWM 신호를 TOUT103 핀으로 출력하기 위한 설정
   - **TOUTSE6은 TOUT96부터 TOUT111까지 제어함.**
   - **TOUT103 핀은 7번째임**
     - **Output Timer Mapped 에서 A 항목을 사용하도록 설정하기 위해 SEL7 영역에 0x0 write**

TOUTSEL6 레지스터 @ 0xF019FD48

TOUTSELn (n = 0-14)
Timer Output Select Register ($9FD30_H+n*4_H$) **Reset Value: 0000 0000_H**

31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16
SEL15	SEL14	SEL13	SEL12	SEL11	SEL10	SEL9	SEL8
rw	rw	rw	rw	rw	rw	rw	rw

15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
SEL7	SEL6	SEL5	SEL4	SEL3	SEL2	SEL1	SEL0
rw	rw	rw	rw	rw	rw	rw	rw

Field	Bits	Type	Description
SELx (x = 0-15)	[x*2+1: x*2]	rw	**TOUT(n*16+x) Output Selection** This bit defines which timer out is connected as TOUT(n*16+x). The mapping for each pin is defined by Table 25-67Table 25-68 00_B Timer A form Table 25-67Table 25-68 is connected as TOUT(n*16+x) to the ports 01_B Timer B form Table 25-67Table 25-68 is connected as TOUT(n*16+x) to the ports 10_B Timer C form Table 25-67Table 25-68 is connected as TOUT(n*16+x) to the ports 11_B Timer D form Table 25-67Table 25-68 is connected as TOUT(n*16+x) to the ports *Note: If TOUT(n*16+x) is not defined in Table 25-67Table 25-68 this bit field has to be treated as reserved.*

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3541

GTM 사용을 위한 레지스터 설정

: P10.1은 TOUT3에 연결되어 있으며, TOM0 모듈의 채널 1을 사용하도록 설정됨

→ P10.1 (TOUT103)은 **TOM0** 의 채널 **1** 사용

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.3484

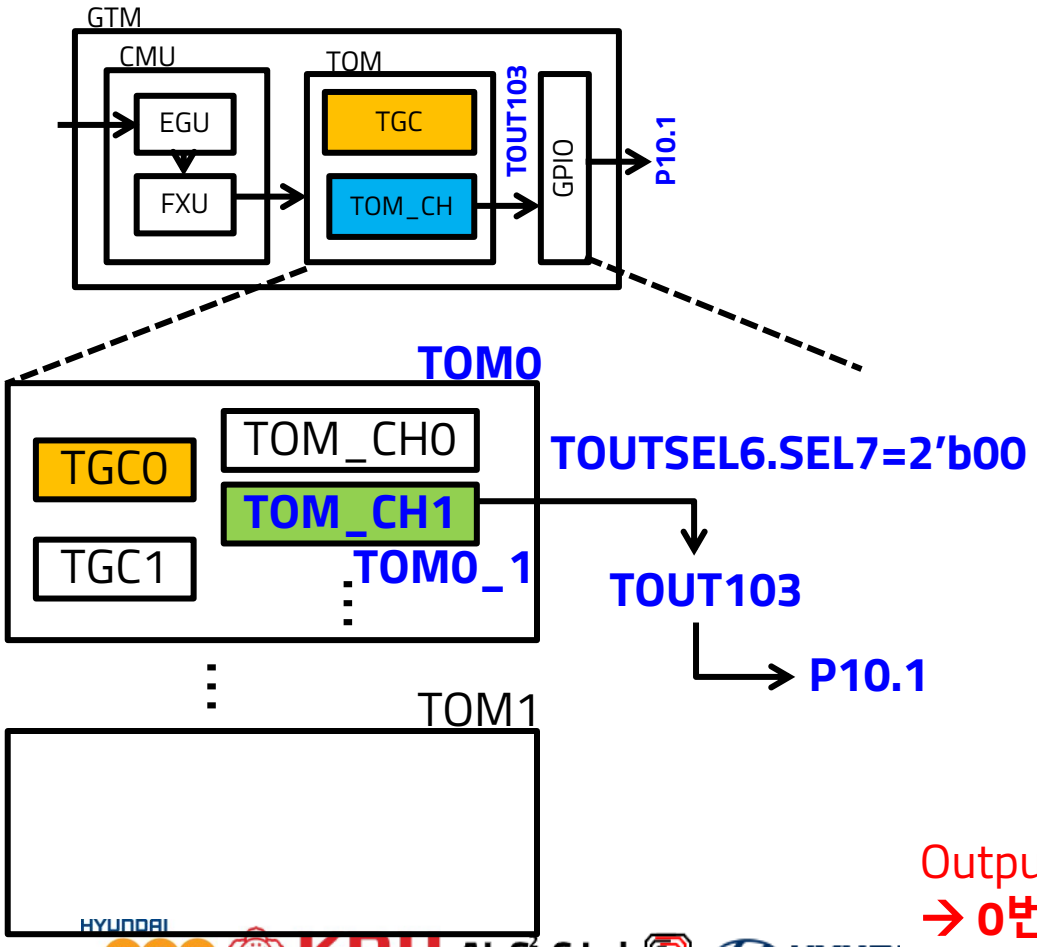


Table 25-67 GTM to Port Mapping for QFP-176

Port	Input	Output	Input Timer Mapped		Output Timer Mapped			
			A	B	A	B	C	D
P02.2	TIN2	TOUT2	TIM0_2	TIM1_2	TOM0_10	TOM1_10	ATOM 0_2	ATOM 1_2
P02.3	TIN3	TOUT3	TIM0_3	TIM1_3	TOM0_11	TOM1_11	ATOM 0_3	ATOM 1_3
P02.4	TIN4	TOUT4	TIM0_4	TIM1_4	TOM0_12	TOM1_12	ATOM 0_4	ATOM 1_4
P02.5	TIN5	TOUT5	TIM0_5	TIM1_5	TOM0_13	TOM1_13	ATOM 0_5	ATOM 1_5
P02.6	TIN6	TOUT6	TIM0_6	TIM1_6	TOM0_14	TOM1_14	ATOM 0_6	ATOM 1_6
P02.7	TIN7	TOUT7	TIM0_7	TIM1_7	TOM0_15	TOM1_15	ATOM 0_7	ATOM 1_7
P02.8	TIN8	TOUT8	TIM2_0	TIM3_0	TOM0_8	TOM1_0	ATOM 0_0	ATOM 1_0
P10.0	TIN102	TOUT102	TIM0_4	TIM1_4	TOM0_4	TOM2_12	ATOM 1_4	ATOM 4_4
P10.1	TIN103	TOUT103	TIM0_1	TIM1_1	TOM0_1	TOM2_9	ATOM 1_1	ATOM 4_1

Output Timer Mapped 의 **A** 항목 사용하도록 설정시

→ **0번째 TOM (TOM0)의 채널 1 사용**

Lab8: GTM 레지스터 설정 – TOUTSEL6

:TOM에서 출력되는 (TOM0모듈의 채널1) PWM 신호를 MCU 핀에 연결 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
93  #define SEL7_BIT_LSB_IDX            14
```

GTM 사용을 위한 레지스터 설정
:TOM에서 출력으로 사용할 채널 설정

- TOM 설정은 TGC (TOM Global Control Unit) 가 담당

Infineon-TC27x_D-step-UM-v02_02-EN.pdf
p.2918

25.11.2 TOM Global Channel Control (TGC0, TGC1)

25.11.2.1 Overview

There exist two global channel control units (TGC0 and TGC1) to drive a number of individual TOM channels synchronously by external or internal events.

Each TGC[y] can drive up to eight TOM channels where TGC0 controls TOM channels 0 to 7 and TGC1 controls TOM channels 8 to 15.

The TOM submodule supports four different kinds of signalling mechanisms:

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2919

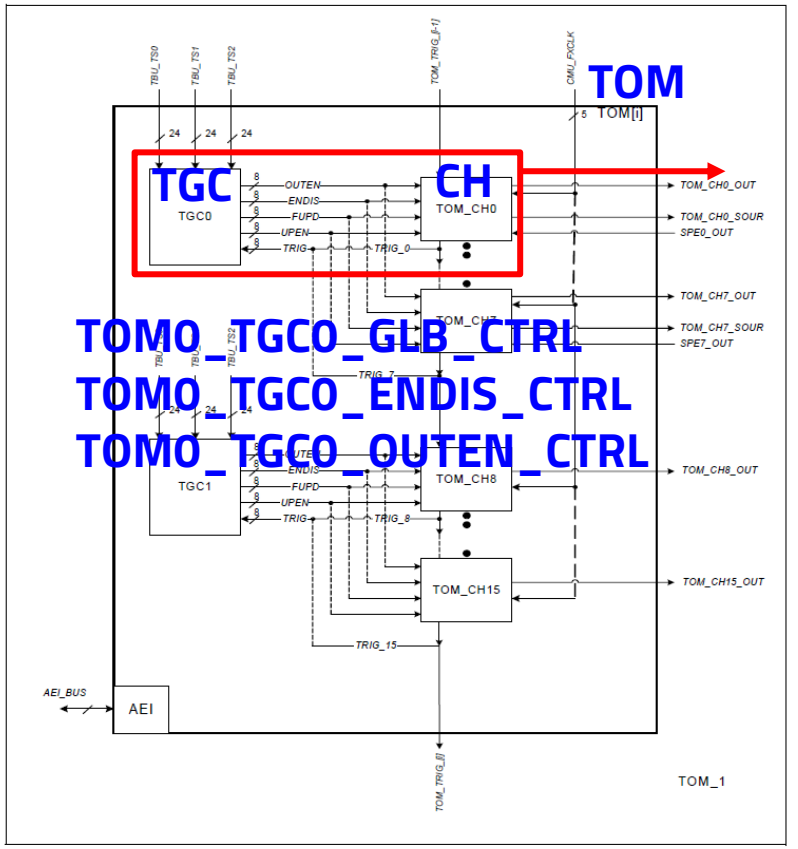


Figure 25-32 TOM Block diagram

- 본 실습에서는 **TOM0의 채널 1**을 사용하기 위해 **TGC0 에 해당하는 레지스터 설정 필요**

- GTM 레지스터 항목에서
  - **TOM0_TGC0_GLB_CTRL** 레지스터 설정 필요
  - **TOM0_TGC0_ENDIS_CTRL** 레지스터 설정 필요
  - **TOM0_TGC0_OUTEN_CTRL** 레지스터 설정 필요

GTM 레지스터 설정 – TOM0_TGC0_GLB_CTRL

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기 (TOM0 이므로 i = 0)
   - **TOM0_TGC0_GLB_CTRL** 의 Offset Address = 0x8030 + (i * 0x800) = **0x8030**

   → TOM0_TGC0_GLB_CTRL 레지스터 주소 = 0xF0100000 + 0x8030 = **0xF0108030**

TOM0_TGC0_GLB_CTRL 레지스터 @ 0xF0108030

GTM_TOMi_TGC0_GLB_CTRL (i=0-2) offset

TOMi TGC0 Global Control Register (08030_H+i*800_H) Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UPEN_CTRL7		UPEN_CTRL6		UPEN_CTRL5		UPEN_CTRL4		UPEN_CTRL3		UPEN_CTRL2		UPEN_CTRL1		UPEN_CTRL0	
rw		rw		rw		rw		rw		rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RST_CH7	RST_CH6	RST_CH5	RST_CH4	RST_CH3	RST_CH2	RST_CH1	RST_CH0	Reserved							HOST_TRIG
w	w	w	w	w	w	w	w	r							w

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2937

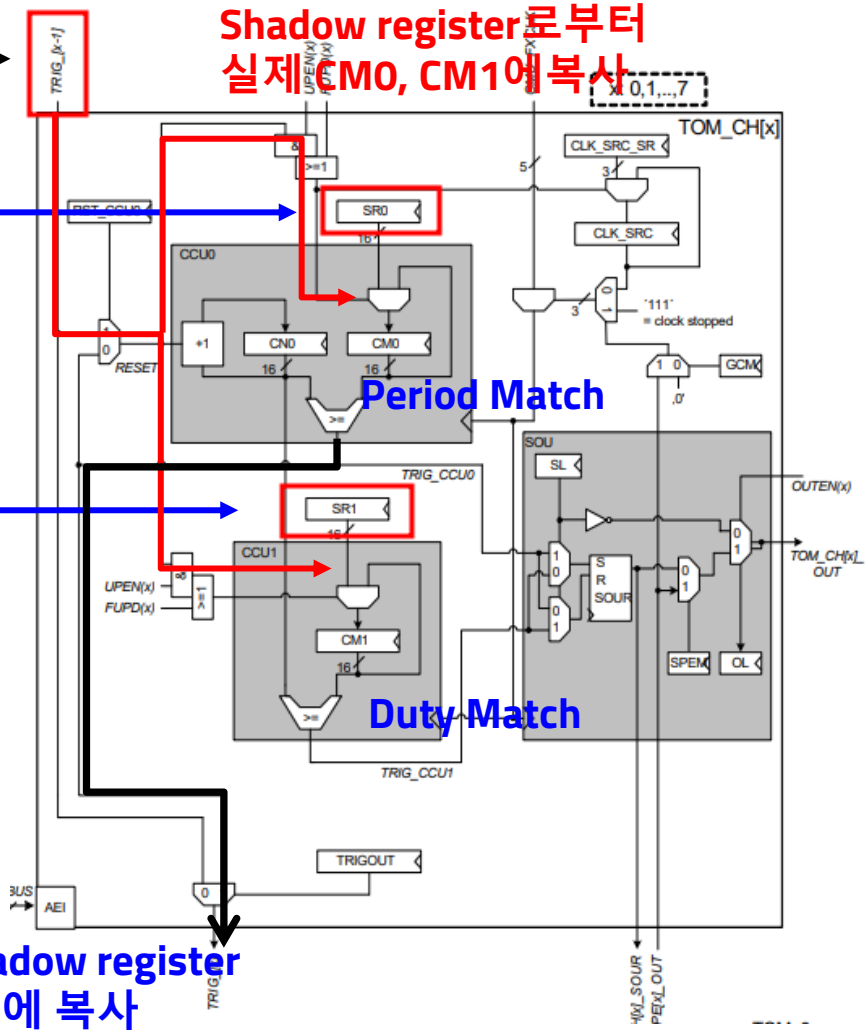
GTM 설정 레지스터의 write 구조

:shadow 레지스터 write되어 update 신호 발생해야 실제 write

Update Trigger 신호

PWM 동작 설정에 필요한 shadow 레지스터

Shadow register로부터 실제 CM0, CM1에 복사



- SW 에서 레지스터 write를 하면 CM0, CM1에 **write 되기 전에 먼저, shadow 레지스터에 저장**

→ Update Trigger 신호 발생하면 실제 레지스터로 옮겨 write 해야 함

→ **TOM0 의 채널 1에서 이러한 update가 가능하도록 설정 필요**

Period Match 신호에서 Trig (즉 shadow register에 셋팅된 SR0, SR1값을 CM0, CM1에 복사

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2923

GTM 레지스터 설정 – TOM0_TGC0_GLB_CTRL
:TOM 에서 사용할 채널 설정

3. 레지스터 write 값 결정
   - TOM0의 채널 1이 동작하기 위해서는 해당 채널에서 PWM 설정 값들이 update 되어야 함
   - Update가 가능하도록 설정하기 위해 **UPEN_CTRL1 영역에 0x2 write**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2923

TOM0_TGC0_GLB_CTRL 레지스터 @ 0xF0108030

GTM_TOMi_TGC0_GLB_CTRL (i=0-2)
TOMi TGC0 Global Control Register(08030_H+i*800_H) Reset Value: 00000000_H

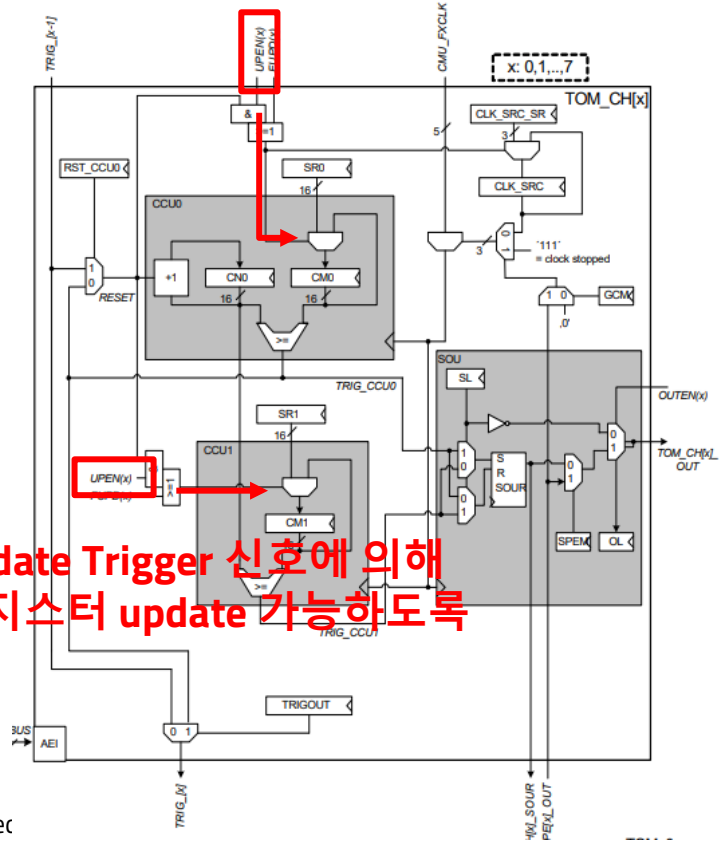
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UPEN_CTRL7		UPEN_CTRL6		UPEN_CTRL5		UPEN_CTRL4		UPEN_CTRL3		UPEN_CTRL2		UPEN_CTRL1		UPEN_CTRL0	
rw		rw		rw		rw		rw		rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RST_CH7	RST_CH6	RST_CH5	RST_CH4	RST_CH3	RST_CH2	RST_CH1	RST_CH0	Reserved							HOST_TRIG
w	w	w	w	w	w	w	w	r							w

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2937

UPEN_CTRL1	[19:18]	rw	TOM channel 1 enable update of register CM0, CM1 and CLK_SRC See bits 17:16

00_B don't care, bits 1:0 will not be changed
01_B update disabled: is read as 00 (see below)
10_B update enabled: is read as 11 (see below)
11_B don't care, bits 1:0 will not be changed



Update Trigger 신호에 의해 레지스터 update 가능하도록

Lab9: GTM 레지스터 설정 – TOM0_TGC0_GLB_CTRL
:TOM 에서 사용할 채널1의 update enable 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
97  // GTM - TOM0 registers
98  #define UPEN_CTRL1_BIT_LSB_IDX      18
```

GTM 레지스터 update 하는 Trigger 신호 생성

:1) SW에서 생성하는 Update Trigger 신호

- 앞으로 설정할 GTM 레지스터들이 실제 하드웨어에 적용될 수 있도록 하는 **Update Trigger 신호를**
- **1) SW에서 생성할 수 있음**
- 2) PWM의 1주기에 도달했을 때 생성할 수 있음 (뒤에서 설명)

- Update Trigger 신호 발생 전까지는 레지스터에 write 해도 실제 하드웨어에 적용되지 않음 (shadow register 개념)

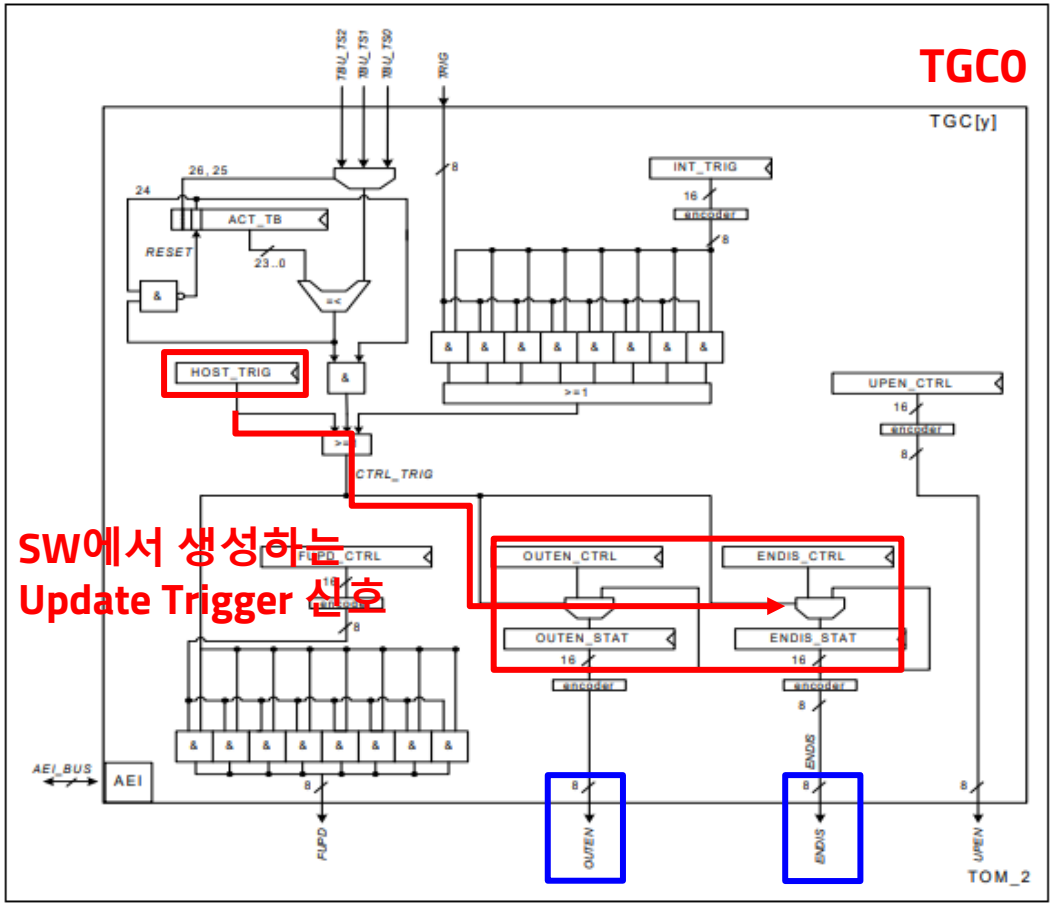


Figure 25-33 TOM Global channel control mechanism

25.11.2.2 TGC Subunit

Each of the first three individual mechanisms (enable/disable of the channel, output enable and force update) can be driven by three different trigger sources.

The three trigger sources are:

- the host CPU (bit HOST_TRIG of register TOMi_TGCy_GLB_CTRL)
- the TBU time stamp (signal TBU_TS0., TBU_TS1, TBU_TS2)
- the internal trigger signal TRIG (bunch of trigger signals TRIG_[x])

Note: The trigger signal is only active for one configured CMU clock period.

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2919

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2921

GTM 레지스터 설정 – TOM0_TGC0_GLB_CTRL
:TOM 레지스터 설정이 적용되도록 update trigger event 발생

3. 레지스터 write 값 결정
   - TOM0의 채널 1에 대한 레지스터 설정이 shadow 레지스터로부터 하드웨어에 적용되도록 하기 위해 **HOST_TRIG 영역에 0x1 write** (GTM 레지스터 설정 후 마지막에 수행)

TOM0_TGC0_GLB_CTRL 레지스터@ 0xF0108030

GTM_TOMi_TGC0_GLB_CTRL (i=0-2)
TOMi TGC0 Global Control Register(08030_H+i*800_H) Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
UPEN_CTRL7		UPEN_CTRL6		UPEN_CTRL5		UPEN_CTRL4		UPEN_CTRL3		UPEN_CTRL2		UPEN_CTRL1		UPEN_CTRL0	
rw		rw		rw		rw		rw		rw		rw		rw	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RST_CH7	RST_CH6	RST_CH5	RST_CH4	RST_CH3	RST_CH2	RST_CH1	RST_CH0	Reserved							HOST_TRIG
w	w	w	w	w	w	w	w	r							w

Field	Bits	Type	Description
HOST_TRIG	0	w	**Trigger request signal (see TGC0, TGC1) to update the register ENDIS_STAT and OUTEN_STAT** 0_B no trigger request 1_B set trigger request Read as 0. *Note:* This flag is cleared automatically after triggering the update

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2937

Lab10: GTM 레지스터 설정 – TOM0_TGC0_GLB_CTRL

:TOM 에서 출력 생성 시작하도록 trigger event 발생

```
99  #define HOST_TRIG_BIT_LSB_IDX       0
```

```
//initERU();
initCCU60();
//initLED();
initPWMLED();
initRGBLED();
initVADC();
initGTM();
//initButton();

// trigger update request signal
GTM_TOM0_TGC0_GLB_CTRL.U |= 0x1 << HOST_TRIG_BIT_LSB_IDX;

while(1)
{
    VADC_startConversion();
    unsigned int adcResult = VADC_readResult();
    for(unsigned int i = 0; i < 100; i++);

    if( adcResult >= 3096 )
    {
        P02_OUT.U |= 0x1 << P7_BIT_LSB_IDX;
        P10_OUT.U &= ~(0x1 << P5_BIT_LSB_IDX);
        P10_OUT.U &= ~(0x1 << P3_BIT_LSB_IDX);
```

PWM 신호가 생성되기 원하는 시점에 trigger 발생시켜야 함 (ADC conversion start와 비슷한 개념임)

GTM 레지스터 설정 – TOM0_TGC0_ENDIS_CTRL

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기 (TOM0 이므로 i = 0)
   - **TOM0_TGC0_ENDIS_CTRL** 의 Offset Address = 0x8070 + (i * 0x800) = **0x8070**

   → TOM0_TGC0_ENDIS_CTRL 레지스터 주소 = 0xF0100000 + 0x8070 = **0xF0108070**

TOM0_TGC0_ENDIS_CTRL 레지스터 @ 0xF0108070

GTM_TOMi_TGC0_ENDIS_CTRL (i=0-2)

TOMi TGC0 Enable/Disable Control Register

$(08070_H+i*800_H)$ offset

Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
r															

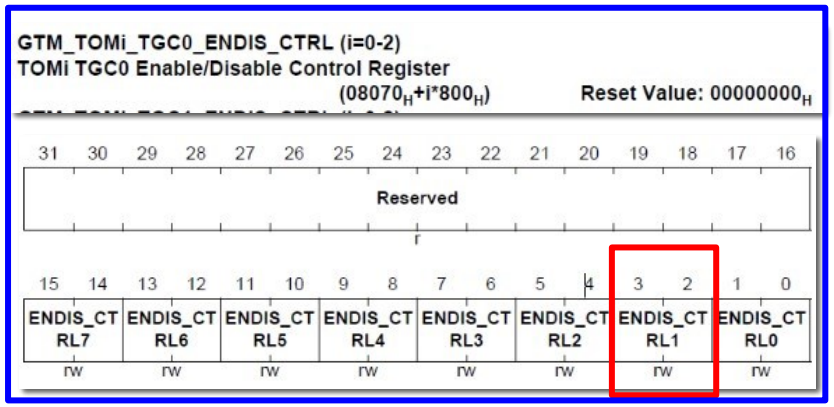
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIS_CTRL7		ENDIS_CTRL6		ENDIS_CTRL5		ENDIS_CTRL4		ENDIS_CTRL3		ENDIS_CTRL2		ENDIS_CTRL1		ENDIS_CTRL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2940

GTM 레지스터 설정 – TOM0_TGC0_ENDIS_CTRL :TOM에서 사용할 채널에 대한 설정

3. 레지스터 write 값 결정
   - TOM0_TGC0_GLB_CTRL 레지스터에서 Update Trigger 신호 발생 시, 하드웨어에 적용되어 **TOM0 채널 1이 enable** 되도록 하기 위해 **ENDIS_CTRL1 영역에 0x2 write**

TOM0_TGC0_ENDIS_CTRL 레지스터 @ 0xF0108070



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2940

ENDIS_CTRL1	[3:2]	rw	(A)TOM channel 1 enable/disable update value See bits 1:0

00_B don't care, bits 1:0 of register ENDIS_ not be changed on an update trigger
01_B disable channel on an update trigger
10_B enable channel on an update trigger
11_B don't change bits 1:0 of this register

Lab11: GTM 레지스터 설정 – TOM0_TGC0_ENDIS_CTRL

:TOM에서 사용할 채널에 대한 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
100 #define ENDIS_CTRL1_BIT_LSB_IDX     2
```

GTM 레지스터 설정 – TOM0_TGC0_OUTEN_CTRL

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기 (TOM0 이므로 i = 0)
   - **TOM0_TGC0_OUTEN_CTRL** 의 Offset Address = 0x8078 + (i * 0x800) = **0x8078**

   → TOM0_TGC0_OUTEN_CTRL 레지스터 주소 = 0xF0100000 + 0x8078 = **0xF0108078**

TOM0_TGC0_OUTEN_CTRL 레지스터 @ 0xF0108078

GTM_TOMi_TGC0_OUTEN_CTRL (i=0-2)

TOMi TGC0 Output Enable Control Register

(08078_H+i*800_H) offset

Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
r															

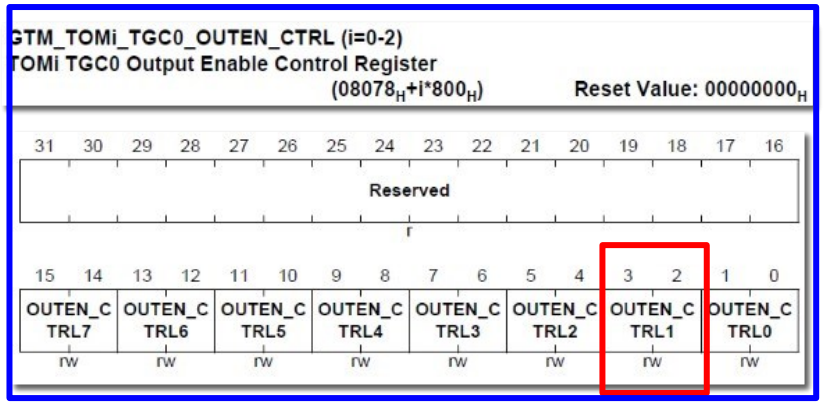
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUTEN_CTRL7		OUTEN_CTRL6		OUTEN_CTRL5		OUTEN_CTRL4		OUTEN_CTRL3		OUTEN_CTRL2		OUTEN_CTRL1		OUTEN_CTRL0	
rw		rw		rw		rw		rw		rw		rw		rw	

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2945

GTM 레지스터 설정 – TOM0_TGC0_OUTEN_CTRL :TOM의 출력이 trigger 이벤트에 반응하도록 설정

3. 레지스터 write 값 결정
   - TOM0_TGC0_GLB_CTRL 레지스터에서 Update Trigger 신호 발생 시, 하드웨어에 적용되어 **TOM0 채널 1의 출력이 enable** 되도록 하기 위해 **OUTEN_CTRL1 영역에 0x2 write**

TOM0_TGC0_OUTEN_CTRL 레지스터 @ 0xF0108078



Field	Bits	Type	Description
OUTEN_CTRL0	[1:0]	rw	**Output (A)TOM_OUT(0) enable/disable update value** Write of following double bit values is possible: 00_B don't care, bits 1:0 of register OUTEN_STAT will not be changed on an update trigger 01_B disable channel output on an update trigger 10_B enable channel output on an update trigger 11_B don't change bits 1:0 of this register Note: if the channel is disabled (ENDIS[0]=0) or the output is disabled (OUTEN[0]=0), the TOM channel 0 output TOM_OUT[0] is the inverted value of bit SL.
OUTEN_CTRL1	[3:2]	rw	**Output (A)TOM_OUT(1)enable/disable update value** See bits 1:0

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2945

Lab12: GTM 레지스터 설정 – TOM0_TGC0_OUTEN_CTRL

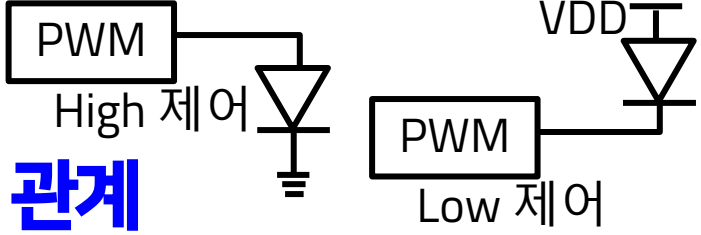
:TOM의 출력이 trigger 이벤트에 반응하도록 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
101  #define OUTEN_CTRL1_BIT_LSB_IDX     2
```

PWM 신호 생성 flow

:PWM 생성과정의 counter와 duty cycle 관계



- TOM에서 이루어지는 PWM 동작 개요

- TOM Channel은 PWM 신호를 생성하기 위해 3가지 값 사용
  - CN0 : TOM clock 주기마다 1씩 증가하는 counter 값
  - CM0 : PWM 신호의 주기를 결정하는 값
  - CM1 : PWM 신호의 Duty Cycle 을 결정하는 값

- CN0 이 clock 주기마다 증가하다가
  - → **CM1 (PWM Duty Cycle 길이)에 도달하면 출력 값 반전**
  - → **CM0 (PWM 1주기 길이)에 도달하면 0으로 초기화**

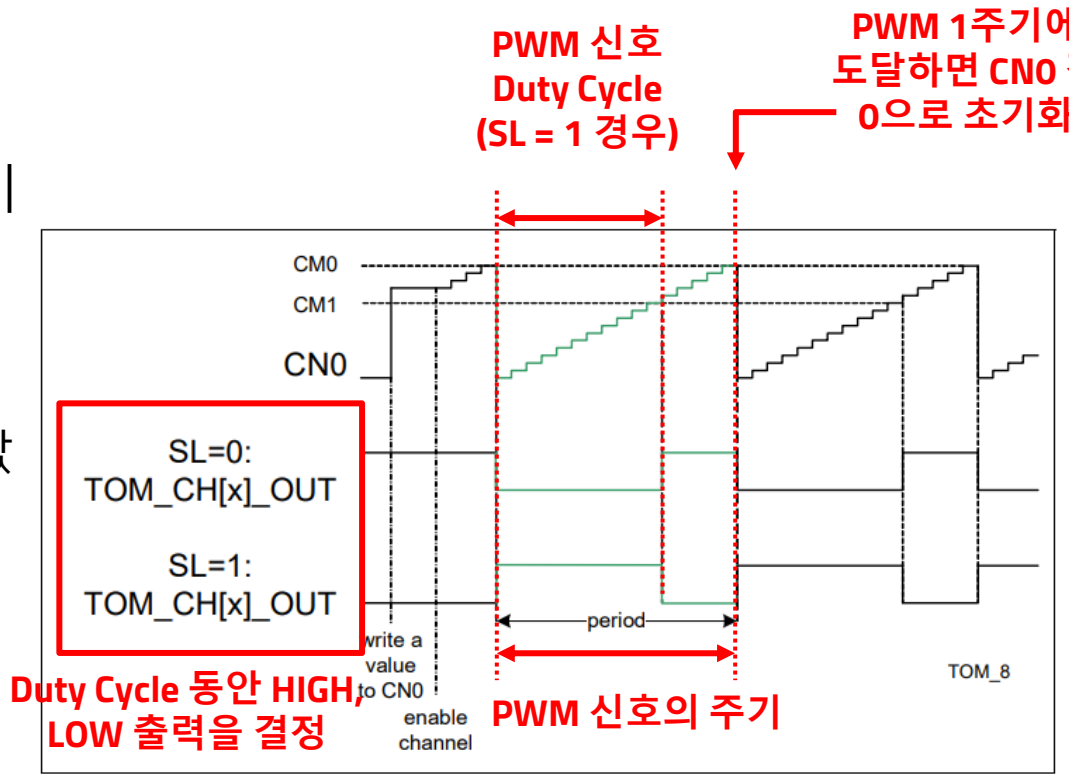


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2930

GTM 사용을 위한 레지스터 설정
:원하는 PWM 신호 생성 위한 TOM 설정

- 본 실습에서 사용하는 **TOM0**의 **채널 1**에 대한 레지스터 설정 필요
  - TOM0에 입력될 FXCLK 주파수 선택
  - PWM 신호의 주기 및 Duty Cycle 결정을 위한 값
  - Duty Cycle 동안 출력될 신호의 HIGH 또는 LOW 선택

- GTM 레지스터 항목에서
  - **TOM0_CH1_CTRL** 레지스터 설정 필요
  - **TOM0_CH1_SR0** 레지스터 설정 필요
  - **TOM0_CH1_SR1** 레지스터 설정 필요

Infineon-TC27x_D-step-UM-v02_02-EN.pdf
p.2930

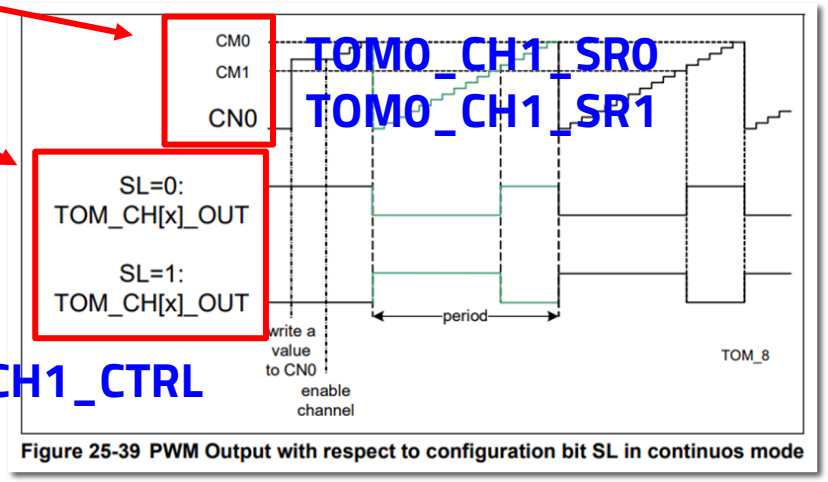


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

GTM 레지스터 설정 – TOM0_CH1_CTRL

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**
2. 사용할 레지스터의 주소 찾기 (채널 1 이므로 x = 1)
   - **TOM0_CH1_CTRL** 의 Offset Address = 0x8000 + (x * 0x40) = **0x8040**

   → TOM0_CH1_CTRL 레지스터 주소 = 0xF0100000 + 0x8040 = **0xF0108040**

TOM0_CH1_CTRL 레지스터 @ 0xF0108040

GTM_TOM0_CHx_CTRL (x=0-14)
TOM0 Channel x Control Register
(08000_H+x*0040_H)
Reset Value: 00000800_H

offset

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		GCM	SPEM	Reserved	OSM	Reserved	TRIGOUT	Reserved			RST_CCU0	Reserved			
r		rw	rw	r	rw	r	rw	r			rw	r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CLK_SRC_SR			SL	Reserved										
r	rw			rw	r										

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2953

GTM 레지스터 설정 – TOM0_CH1_CTRL
:PWM 신호 중 duty cycle을 어떤 값으로 출력할 지 설정

3. 레지스터 write 값 결정
   - TOM0 채널1의 동작을 설정
   - PWM 신호의 Duty Cycle 영역에서 신호의 값을 HIGH로 출력하기 위해 **SL 영역에 0x1 write**

TOM0_CH1_CTRL 레지스터 @ 0xF0108040

GTM_TOM0_CHx_CTRL (x=0-14)
TOM0 Channel x Control Register'
(08000_H+x*0040_H) Reset Value: 00000800_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		GCM	SPEM	Reserved	OSM	Reserved	TRIGOUT	Reserved			RST_CCU0	Reserved			
r		rw	rw	r	rw	r	rw	r			rw	r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CLK_SRC_SR			SL	Reserved										
r	rw			rw	r										

Field	Bits	Type	Description
SL	11	rw	**Signal level for duty cycle** 0_B Low signal level 1_B High signal level If the output is disabled, the output TOM_OUT[x] is set to inverse value of SL.

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2953

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2930

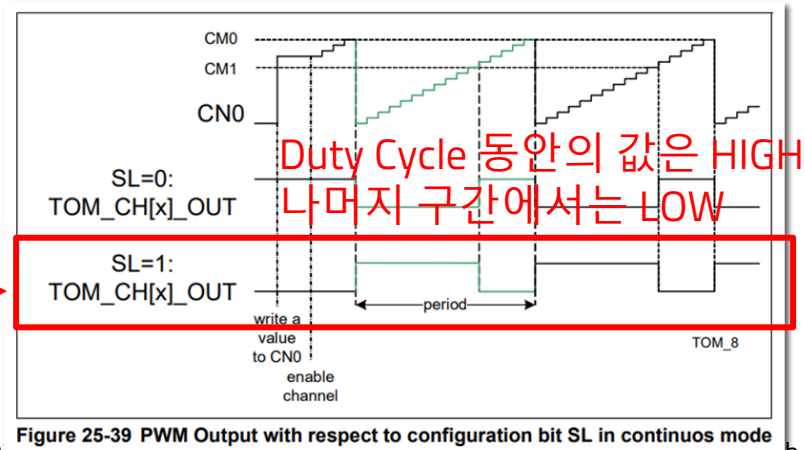


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

Lab13: GTM 레지스터 설정 – TOM0_CH1_CTRL

:PWM 신호 중 duty cycle을 어떤 값으로 출력할 지 설정

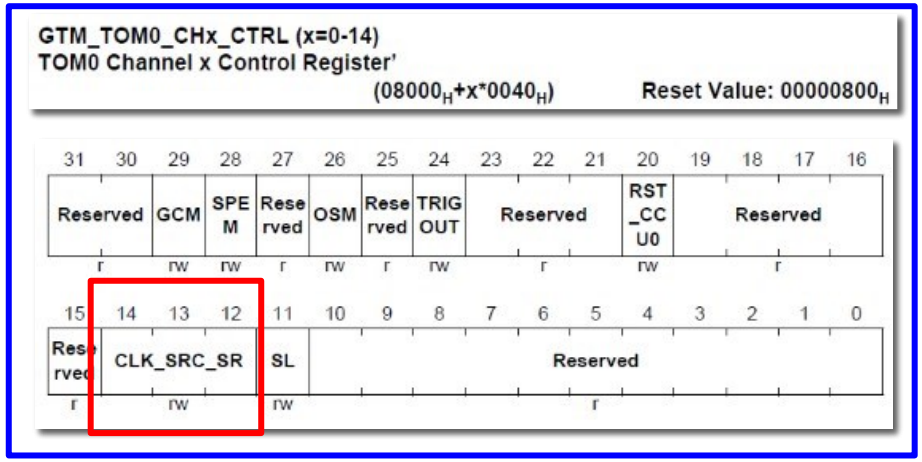
```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source  -> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
105 #define SL_BIT_LSB_IDX              11
106
```

GTM 레지스터 설정 – TOM0_CH1_CTRL
:FXU에서 생성하고 TOM에서 사용할 clock 주파수 설정

3. 레지스터 write 값 결정
   - TOM0 채널1의 동작을 설정
   - TOM0 에서 사용할 FXCLK clock 신호의 주파수를 결정하기 위해 **CLK_SRC_SR 영역에 0x0 write**
   - 현재 GTM 에서 사용하는 system clock 주파수는 100 MHz → **FXCLK = 6250 kHz**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2953



TOM0_CH1_CTRL 레지스터 @ 0xF0108040

FXU로 입력되는 system clock을 2^4 (= 16)으로 나눠서 FXCLK 로 사용
→ **100 MHz / 16 = 6250 kHz**

CLK_SRC_SR	[14:12]	rw	**Clock source select for channel** The register CLK_SRC is updated with the value of CLK_SRC_SR together with the update of register CM0 and CM1. The input of the FX clock divider depends on the value of FXCLK_SEL (see CMU). 000_B CMU_FXCLK(0) selected: clock selected by FXCLKSEL 001_B CMU_FXCLK(1) selected: clock selected by FXCLKSEL/ 2^4 010_B CMU_FXCLK(2) selected: clock selected by FXCLKSEL / 2^8 011_B CMU_FXCLK(3) selected: clock selected by FXCLKSEL / 2^12 100_B CMU_FXCLK(4) selected: clock selected by FXCLKSEL / 2^16 101_B no CMU_FXCLK selected, clock of channel stopped 110_B no CMU_FXCLK selected, clock of channel stopped 111_B no CMU_FXCLK selected, clock of channel stopped Note: if clock of channel is stopped (i.e. CLK_SRC = 101/110/111), the channel can only be restarted by resetting CLK_SRC_SR to a value of 000 to 100 and forcing an update via the force update mechanism.

Lab14: GTM 레지스터 설정 – TOM0_CH1_CTRL

:FXU에서 생성하고 TOM에서 사용할 clock 주파수 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << GSC_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
102 #define CLK_SRC_SR_BIT_LSB_IDX      12
```

GTM 레지스터 설정 – TOM0_CH1_CTRL
:연속적으로 PWM 신호가 생성되는 Continuous 모드

3. 레지스터 write 값 결정
   - TOM0 채널1에서 생성되는 PWM 신호를 continuous 모드로 사용하기 위해 **OSM** 영역에 **0x0 write**

PWM 주기가 연속으로 생성되는 모드

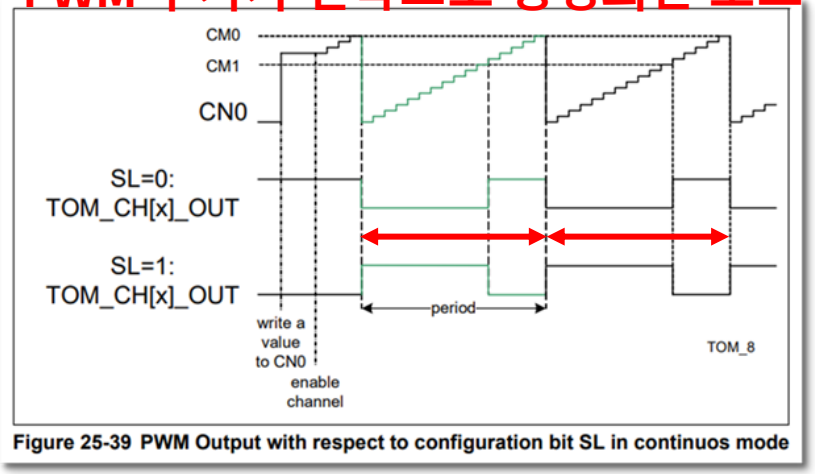


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2930

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2953

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		GCM	SPEM	Reserved	OSM	Reserved	TRIGOUT	Reserved			RST_CCU0	Reserved			
r		rw	rw	r	rw	r	rw	r			rw	r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CLK_SRC_SR			SL	Reserved										
r	rw			rw	r										

TOM0_CH1_CTRL 레지스터 @ 0xF0108040

OSM	26	rw	**One-shot mode** In this mode the counter CN0 counts for only one period The length of period is defined by CM0 A write access to the register CN0 triggers the start of counting 0_B One-shot mode disabled 1_B One-shot mode enabled

One-shot 모드 disable
= Continuous 모드

Lab15: GTM 레지스터 설정 – TOM0_CH1_CTRL
:연속적으로 PWM 신호가 생성되는 Continuous 모드

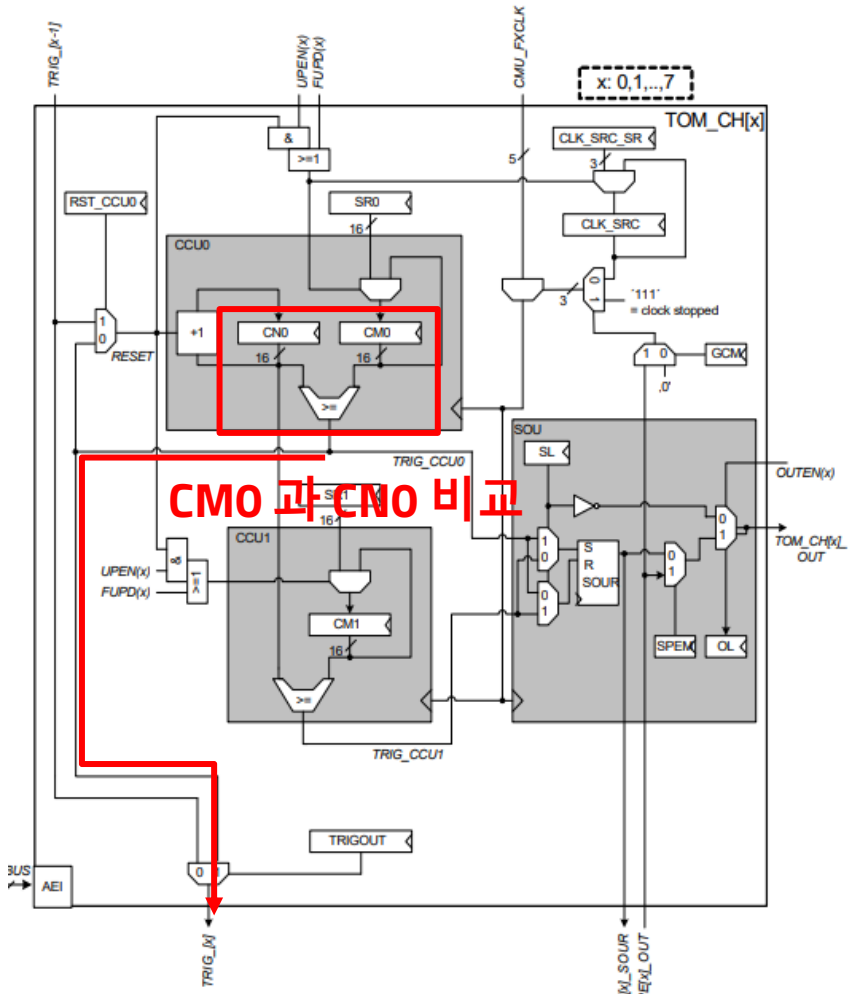
```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[n] = TRIG[n-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
103  #define OSM_BIT_LSB_IDX          26
```

GTM 레지스터 update 하는 Trigger 신호 생성
:PWM의 1주기에서 생성하는 Update Trigger 신호

- 앞으로 설정할 GTM 레지스터 (SR0, SR1 등...) 들이 실제 하드웨어에 적용될 수 있도록 하는 **Update Trigger 신호를**
- 1) SW에서 생성할 수 있음
- **2) PWM의 1주기에 도달했을 때 생성할 수 있음**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2923



25.11.2.2 TGC Subunit

Each of the first three individual mechanisms (enable/disable of the channel, output enable and force update) can be driven by three different trigger sources.

The three trigger sources are:

- the host CPU (bit HOST_TRIG of register TOMi_TGCy_GLB_CTRL)
- the TBU time stamp (signal TBU_TS0.. TBU_TS1, TBU_TS2)
- the internal trigger signal TRIG (bunch of trigger signals TRIG_[x])

Note: The trigger signal is only active for one configured CMU clock period.

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2919

GTM 레지스터 update 하는 Trigger 신호
:Continuous 모드에서 PWM 주기, duty cycle 적용

- PWM의 1주기에 도달할 때마다 update trigger 신호 발생

= CN0 카운터 값이 CM0에 도달하는 시점

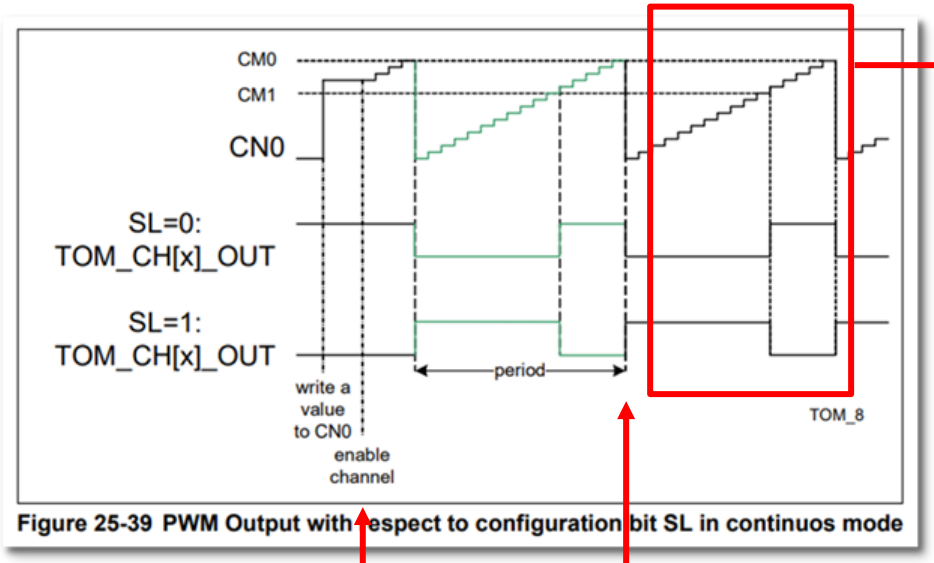


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

새로 업데이트된
CM0, CM1 값으로
PWM 출력

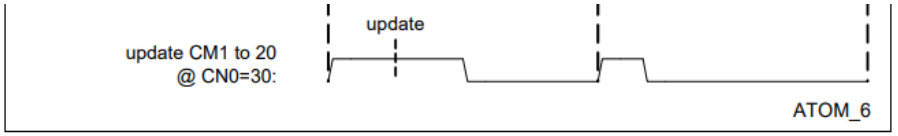


Figure 25-37 synchronous update of duty cycle

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2928

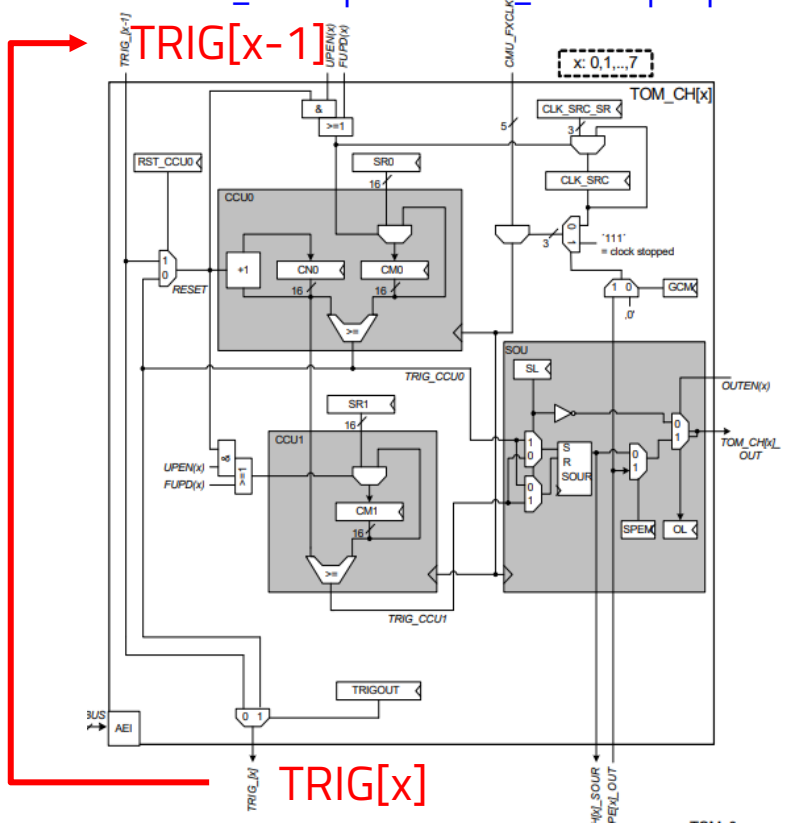
최초 SW에서 발생한
Update Trigger에 의해
시작

PWM 1주기 도달
→ SR0, SR1 레지스터 (shadow) 에 저장된 값을 각각 CM0, CM1로 copy

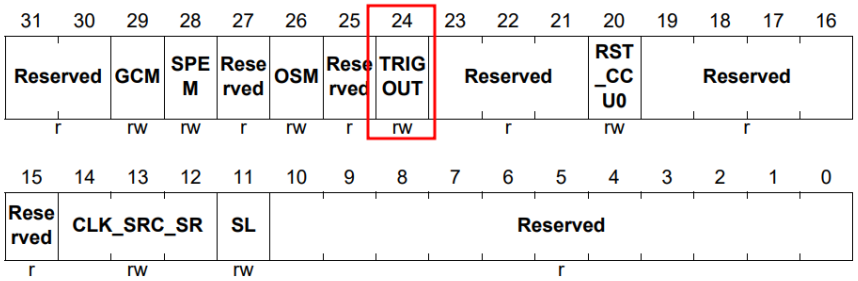
GTM 레지스터 설정 – TOM0_CH1_CTRL
:PWM 신호에서 발생하는 trigger 신호 설정

3. 레지스터 write 값 결정
   - **PWM 신호의 1주기 도달 (CN0 = CM0)시 발생하는 trigger (= TRIG) 신호를 Next trigger 신호**로 사용하기 위해 **TRIGOUT** 영역에 **0x0 write**

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2923



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2953



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		GCM	SPEM	Reserved	OSM	Reserved	TRIGOUT	Reserved			RST_CCU0	Reserved			
r		rw	rw	r	rw	r	rw	r			rw	r			

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	CLK_SRC_SR			SL	Reserved										
r	rw			rw	r										

TOM0_CH1_CTRL 레지스터 @ 0xF0108040

TRIGOUT	24	rw	Trigger output selection (output signal TRIG_[x]) of module TOM_CH[x]
			0_B TRIG_[x] is TRIG_[x-1]
			1_B TRIG_[x] is TRIG_CCU0

TRIG 신호 선택

Lab16: GTM 레지스터 설정 – TOM0_CH1_CTRL
:PWM 신호에서 발생하는 trigger 신호 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

```
104 #define TRIGOUT_BIT_LSB_IDX         24
```

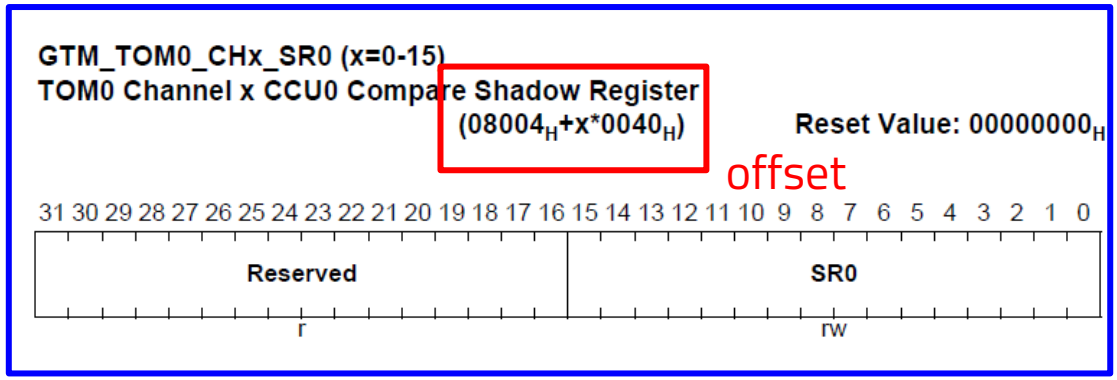
GTM 레지스터 설정 – TOM0_CH1_SR0

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기 (채널 1 이므로 x = 1)
   - **TOM0_CH1_SR0** 의 Offset Address = 0x8004 + (x * 0x40) = **0x8044**

   → TOM0_CH1_SR0 레지스터 주소 = 0xF0100000 + 0x8044 = **0xF0108044**

TOM0_CH1_SR0 레지스터 @ 0xF0108044



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2962

GTM 레지스터 설정 – TOM0_CH1_SR0

:PWM 신호의 주기 설정

$$PWM\ Period = \frac{CM0 + 1}{Freq.of\ CMU\_FXCLK1} = \frac{12500}{6250\ kHz} = 2ms$$

3. 레지스터 write 값 결정
   - TOM0 채널1의 동작을 설정
   - 1초 후 CN0= 6250k-1, 1ms →CN0=6250-1, 2ms후→CN0=6250*2-1 값이 됨
   - PWM 신호의 주기를 2ms 로 설정하기 위해 **SR0 영역에 10진수 (12500 – 1) write**
   - SR0 레지스터에 설정할 CM0 값을 저장하면 Trigger로 update 시 CM0에 반영됨

TOM0_CH1_SR0 레지스터@ 0xF0108044

GTM_TOM0_CHx_SR0 (x=0-15)
TOM0 Channel x CCU0 Compare Shadow Register
(08004_H+x*0040_H) Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																SR0															
r																rw															

Field	Bits	Type	Description
SR0	[15:0]	rw	TOM channel x shadow register SR0 for update of compare register CM0
Reserved	[31:16]	r	Reserved Read as zero, should be written as zero

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2962

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2930

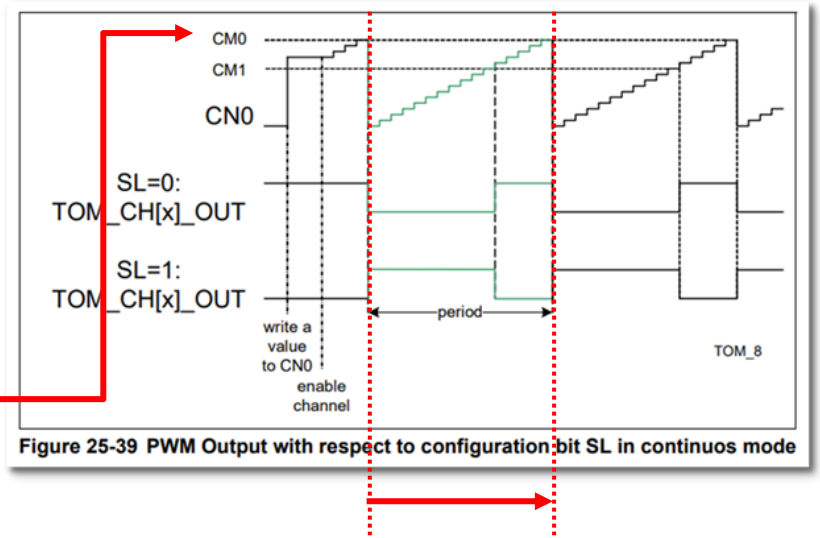


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

CN0 값이 0부터 증가하기 시작하여 CM0 값에 만나기 까지의 시간이 PWM 신호의 1주기

Lab17: GTM 레지스터 설정 – TOM0_CH1_SR0
:PWM 신호의 주기 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

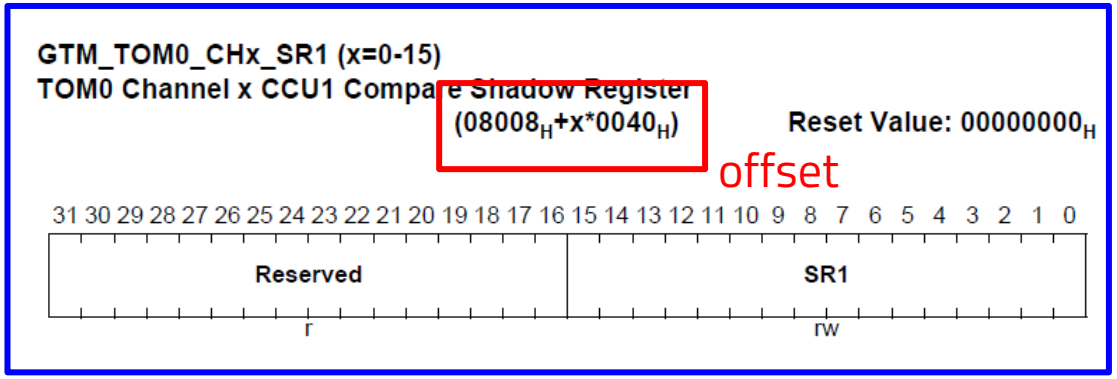
GTM 레지스터 설정 – TOM0_CH1_SR1

1. GTM 레지스터 영역의 주소 찾기
   - 시작 주소 (Base address) **= 0xF0100000**

2. 사용할 레지스터의 주소 찾기 (채널 1 이므로 x = 1)
   - **TOM0_CH1_SR1** 의 Offset Address = 0x8008 + (x * 0x40) = **0x8048**

   → TOM0_CH1_SR1 레지스터 주소 = 0xF0100000 + 0x8048 = **0xF0108048**

TOM0_CH1_SR1 레지스터@ 0xF0108048



Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2964

GTM 레지스터 설정 – TOM0_CH1_SR1

:PWM 신호의 duty cycle % 설정

3. 레지스터 write 값 결정
   - TOM0 채널1의 동작을 설정
   - PWM 신호의 Duty Cycle을 설정하기 위해 **SR1 영역에 write**
   - SR1 레지스터에 설정할 CM1 값을 저장하면 Trigger로 update 시 CM1에 반영됨

$$PWM\ Duty\ Cycle = \frac{\boldsymbol{CM1} + 1}{CM0 + 1} \times 100\ [\%]$$

TOM0_CH1_SR1 레지스터@ 0xF0108048

GTM_TOM0_CHx_SR1 (x=0-15)
TOM0 Channel x CCU1 Compare Shadow Register
(08008_H+x*0040_H) Reset Value: 00000000_H

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																SR1															
r																rw															

Field	Bits	Type	Description
SR1	[15:0]	rw	TOM channel x shadow register SR1 for update of compare register CM1
Reserved	[31:16]	r	Reserved Read as zero, should be written as zero

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2964

Infineon-TC27x_D-step-UM-v02_02-EN.pdf p.2930

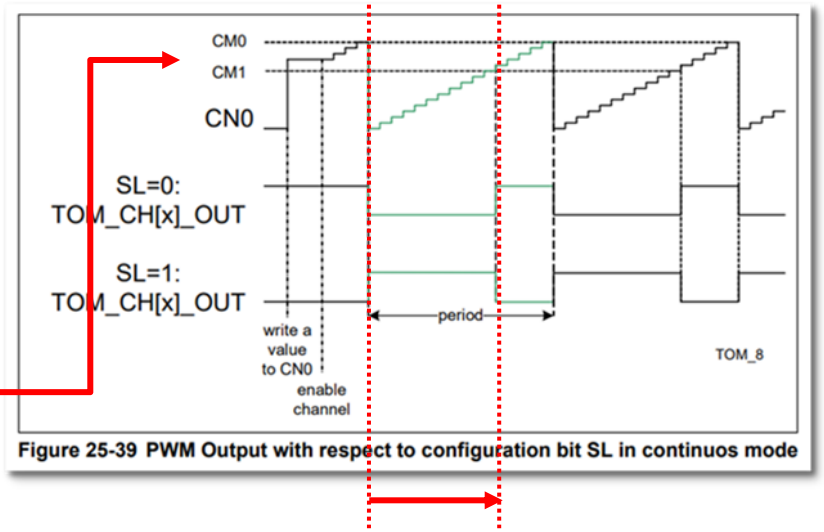


Figure 25-39 PWM Output with respect to configuration bit SL in continuos mode

CN0 값이 0부터 증가하기 시작하여 CM1 값에 만나기 까지의 시간이, PWM 신호의 1주기 시간 중 차지하는 비율이 Duty Cycle (%)

Lab18: GTM 레지스터 설정 – TOM0_CH1_SR1

:PWM 신호의 duty cycle % 설정

```
70  void initGTM(void)
71  {
72      // Password Access to unlock SCU_WDTSCON0
73      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
74      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
75
76      // Modify Access to clear ENDINIT
77      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
78      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked
79
80      GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM
81
82      // Password Access to unlock SCU_WDTSCON0
83      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
84      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked
85
86      // Modify Access to set ENDINIT
87      SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
88      while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
89
90      while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled
91
92
93      // GTM clock configuration
94      GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
95      GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK
96
97      // GTM TOM0 PWM configuration
98      GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable
99
00      GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger
01
02      GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger
03
04      GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
05      GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
06      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
07      GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]
08
09      GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
10
11      GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)
12
13      GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
14                                                                      // 103 = 16 * 6 + 7
15  }
16
```

SW 프로그래밍

:GTM 레지스터의 각 영역(필드) LSB 비트 시작 위치 define 정의

- 레지스터에 값을 write할 때 shift되는 offset을 쉽게 사용하기 위한 define 작성

```
26  **********************************
27  #include "Ifx_Types.h"
28  #include "IfxCpu.h"
29  #include "IfxScuWdt.h"
30
31  #include "IfxCcu6_reg.h"
32  #include "IfxVadc_reg.h"
33  #include "IfxGtm_reg.h"
34
```

→ 헤더 파일 참조 추가

```
89
90  // GTM registers
91  #define DISS_BIT_LSB_IDX            1
92  #define DISR_BIT_LSB_IDX            0
93  #define SEL7_BIT_LSB_IDX            14
94  #define EN_FXCLK_BIT_LSB_IDX        22
95  #define FXCLK_SEL_BIT_LSB_IDX       0
96
97  // GTM - TOM0 registers
98  #define UPEN_CTRL1_BIT_LSB_IDX      18
99  #define HOST_TRIG_BIT_LSB_IDX       0
100 #define ENDIS_CTRL1_BIT_LSB_IDX     2
101 #define OUTEN_CTRL1_BIT_LSB_IDX     2
102 #define CLK_SRC_SR_BIT_LSB_IDX      12
103 #define OSM_BIT_LSB_IDX             26
104 #define TRIGOUT_BIT_LSB_IDX         24
105 #define SL_BIT_LSB_IDX              11
106
```

} GTM 레지스터 bit shift offset

SW 프로그래밍

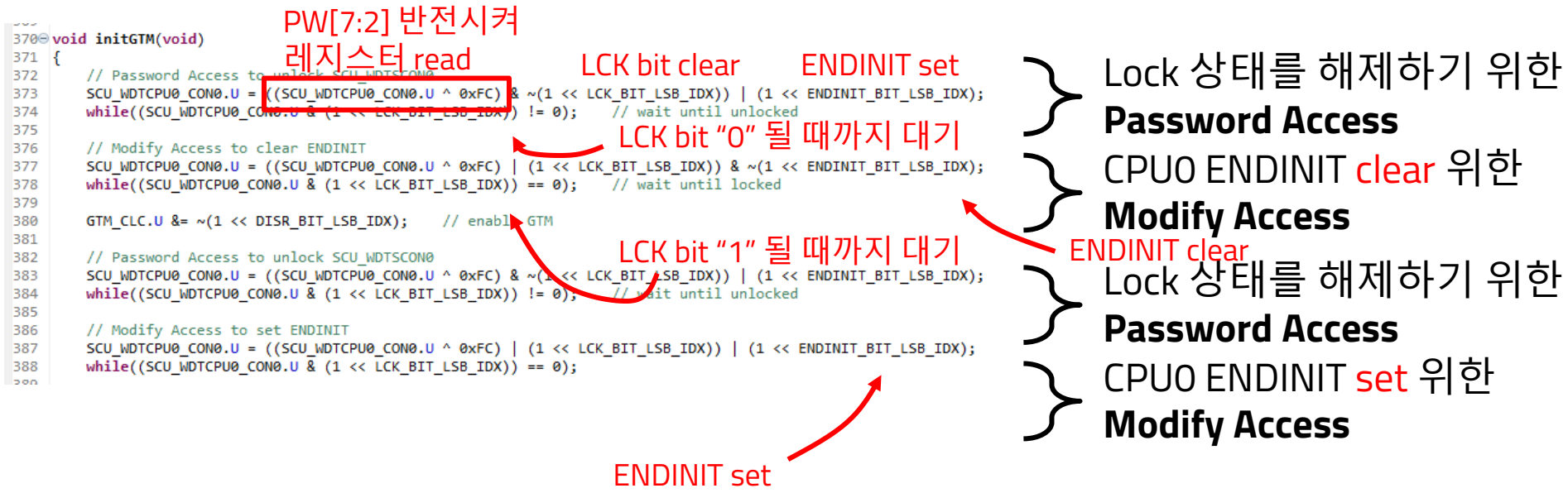
- GTM 모듈 사용을 위한 초기화 함수 *initGTM()*

```
void initGTM(void)
{
    // Password Access to unlock SCU_WDTSCON0
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked

    // Modify Access to clear ENDINIT
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked

    GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM

    // Password Access to unlock SCU_WDTSCON0
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked

    // Modify Access to set ENDINIT
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);

    while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled


    // GTM clock configuration
    GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
    GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK

    // GTM TOM0 PWM configuration
    GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable

    GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger

    GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger

    GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
    GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
    GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
    GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]

    GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz

    GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)

    GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
                                                                    // 103 = 16 * 6 + 7
}
```

SW 프로그래밍

:GTM 모듈 사용 설정 전, 보호 레지스터 잠금 해제 필요
:GTM 모듈 사용 설정 후, 보호 레지스터 잠금 재설정 필요

- GTM 모듈 사용을 위한 초기화 함수 *initGTM()*




```
void initGTM(void)
{
    // Password Access to unlock SCU_WDTSCON0
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked

    // Modify Access to clear ENDINIT
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) & ~(1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);    // wait until locked

    GTM_CLC.U &= ~(1 << DISR_BIT_LSB_IDX);    // enable GTM

    // Password Access to unlock SCU_WDTSCON0
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) & ~(1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) != 0);    // wait until unlocked

    // Modify Access to set ENDINIT
    SCU_WDTCPU0_CON0.U = ((SCU_WDTCPU0_CON0.U ^ 0xFC) | (1 << LCK_BIT_LSB_IDX)) | (1 << ENDINIT_BIT_LSB_IDX);
    while((SCU_WDTCPU0_CON0.U & (1 << LCK_BIT_LSB_IDX)) == 0);
```

SW 프로그래밍
:GTM 모듈 설정

- GTM 모듈 사용을 위한 초기화 함수 *initGTM()*

```
    while((GTM_CLC.U & (1 << DISS_BIT_LSB_IDX)) != 0); // wait until GTM module enabled


    // GTM clock configuration
    GTM_CMU_FXCLK_CTRL.U &= ~(0xF << FXCLK_SEL_BIT_LSB_IDX);        // input clock of CMU_FXCLK --> CMU_GCLK_EN
    GTM_CMU_CLK_EN.U |= 0x2 << EN_FXCLK_BIT_LSB_IDX;                // enable all CMU_FXCLK

    // GTM TOM0 PWM configuration
    GTM_TOM0_TGC0_GLB_CTRL.U |= 0x2 << UPEN_CTRL1_BIT_LSB_IDX;      // TOM channel 1 update enable

    GTM_TOM0_TGC0_ENDIS_CTRL.U |= 0x2 << ENDIS_CTRL1_BIT_LSB_IDX;   // enable channel 1 on update trigger

    GTM_TOM0_TGC0_OUTEN_CTRL.U |= 0x2 << OUTEN_CTRL1_BIT_LSB_IDX;   // enable channel 1 output on update trigger

    GTM_TOM0_CH1_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                   // high signal level for duty cycle
    GTM_TOM0_CH1_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;           // clock source --> CMU_FXCLK(1) = 6250 kHz
    GTM_TOM0_CH1_CTRL.U &= ~(0x1 << OSM_BIT_LSB_IDX);               // continuous mode enable
    GTM_TOM0_CH1_CTRL.U &= ~(0x1 << TRIGOUT_BIT_LSB_IDX);           // TRIG[x] = TRIG[x-1]

    GTM_TOM0_CH1_SR0.U = 12500 - 1;                                 // PWM freq. = 6250 kHz / 12500 = 500 Hz

    GTM_TOM0_CH1_SR1.U = 1250 - 1;                                  // duty cycle = 1250 / 12500 = 10 % (temporary)

    GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);                   // TOUT103 --> TOM0 channel 1
                                                                    // 103 = 16 * 6 + 7
}
```

GTM 모듈 설정
- GTM 모듈 enable
- TOM에서 사용하는 FXCLK 주파수 설정
- TOM을 제어하는 TGC에 대한 설정
- TOM0의 채널1에 대한 설정
- PWM 신호의 주기, Duty Cycle 설정
- PWM 신호가 출력될 핀 설정

SW 프로그래밍
:VADC 모듈에서 변환된 디지털 값 사용 PWM Duty Cycle 제어

초기화 함수 호출

```
147
148     //initERU();
149     initCCU60();
150     initLED();
151     initRGBLED();
152     initVADC();
153     initGTM();
154     //initButton();
155
```

- main 함수 작성

```
146
147     //initERU();
148     initCCU60();
149     initLED();
150     initRGBLED();
151     initVADC();
152     initGTM();
153     //initButton();
154
155     GTM_TOM0_TGC0_GLB_CTRL.U |= 0x1 << HOST_TRIG_BIT_LSB_IDX;       // trigger update request signal
156
157     while(1)
158     {
159         VADC_startConversion();
160         unsigned int adcResult = VADC_readResult();
161         for(unsigned int i = 0; i < 100; i++);
162
163         if( adcResult >= 3096 )
164         {
165             P02_OUT.U |= 0x1 << P7_BIT_LSB_IDX;
166             P10_OUT.U &= ~(0x1 << P5_BIT_LSB_IDX);
167             P10_OUT.U &= ~(0x1 << P3_BIT_LSB_IDX);
168
169             GTM_TOM0_CH1_SR1.U = 0;
170         }
171         else if( adcResult >= 2048 )
172         {
173             P02_OUT.U &= ~(0x1 << P7_BIT_LSB_IDX);
174             P10_OUT.U |= 0x1 << P5_BIT_LSB_IDX;
175             P10_OUT.U &= ~(0x1 << P3_BIT_LSB_IDX);
176
177             GTM_TOM0_CH1_SR1.U = 1000;
178         }
179         else if( adcResult >= 1024 )
180         {
181             P02_OUT.U &= ~(0x1 << P7_BIT_LSB_IDX);
182             P10_OUT.U &= ~(0x1 << P5_BIT_LSB_IDX);
183             P10_OUT.U |= 0x1 << P3_BIT_LSB_IDX;
184
185             GTM_TOM0_CH1_SR1.U = 7000;
186         }
187         else
188         {
189             P02_OUT.U |= 0x1 << P7_BIT_LSB_IDX;
190             P10_OUT.U |= 0x1 << P5_BIT_LSB_IDX;
191             P10_OUT.U |= 0x1 << P3_BIT_LSB_IDX;
192
193             GTM_TOM0_CH1_SR1.U = 12500;
194         }
195
196     }
197     return (1);
198 }
199
```

처음 update trigger 발생
→ 이후에는 continuous 모드이므로 자동으로 trigger 발생

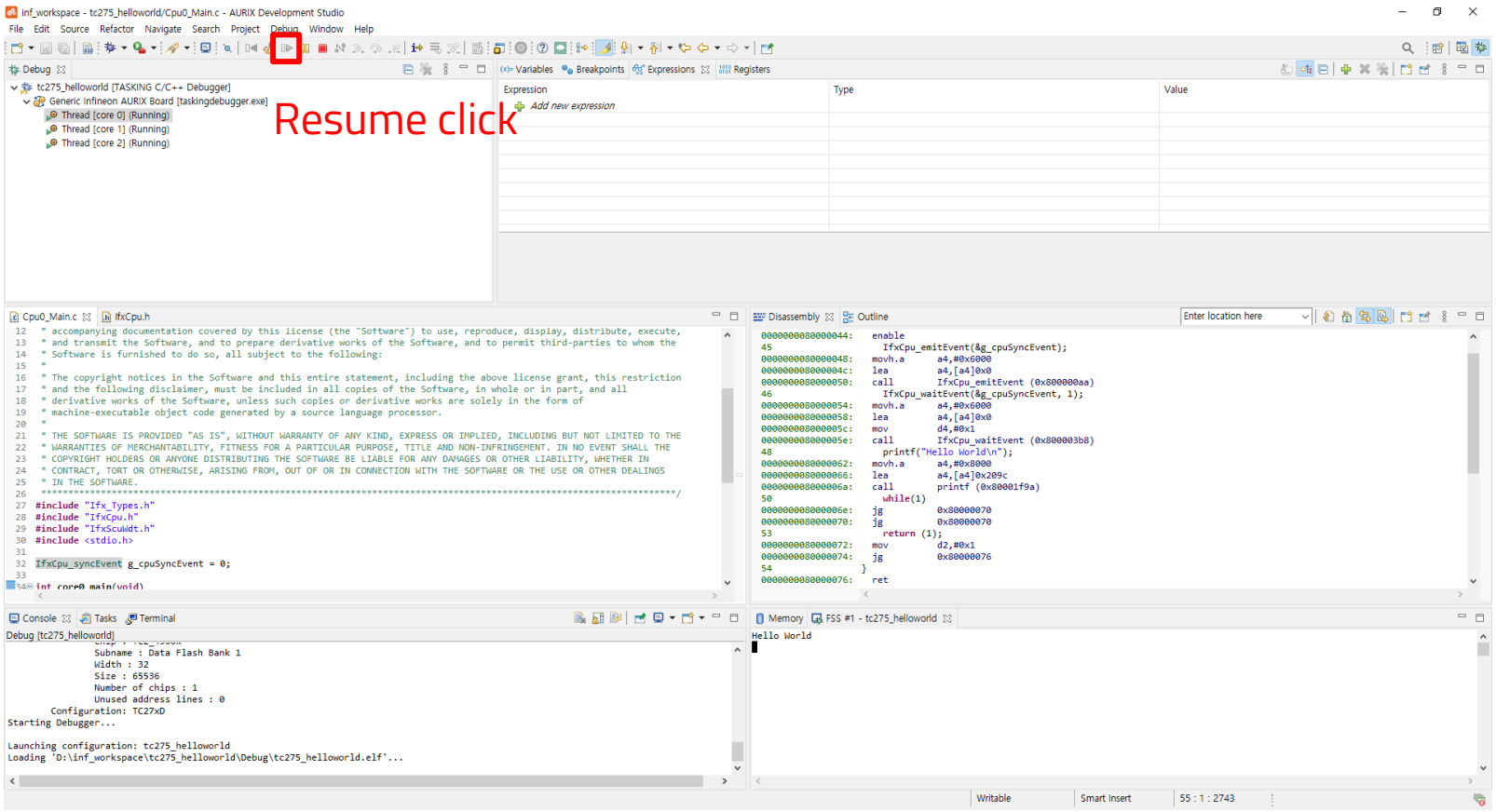
가변 저항에서 읽은 ADC 값의 범위에 따라 RGB LED on/off 결정

가변 저항에서 읽은 ADC 값의 범위에 따라 LED RED에 연결된 PWM 신호 Duty Cycle 변경

- **RGB LED는 지난주 그대로.**
- **Red LED는 가변저항 전압 범위에 따라 계단형 밝기 변화 제어**

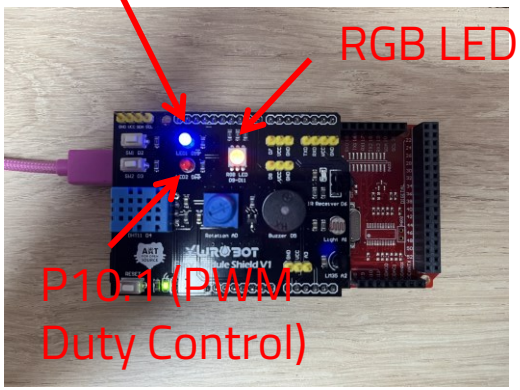
Build 및 Debug

- 프로젝트 빌드 (ctrl + b)
- 디버그 수행하여 보드에 실행 파일 flash

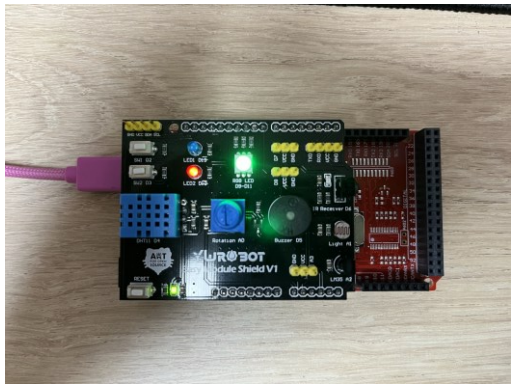


동작 확인

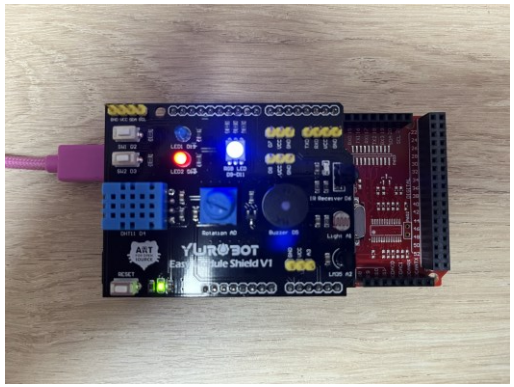
- 가변 저항을 돌리면 변화하는 ADC 값에 따라 RGB LED 색 뿐만 아니라, PWM 신호 Duty Cycle 변화에 의해 LED RED 의 밝기 또한 변하는 모습 확인 가능



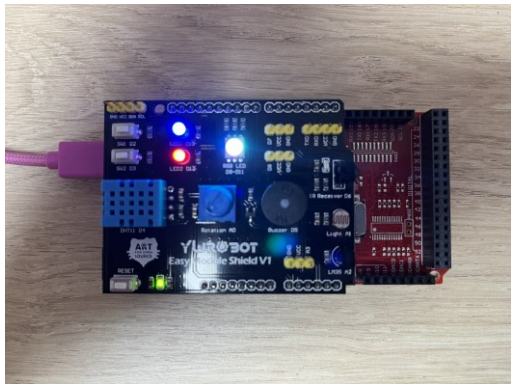
adcValue >= 3096



adcValue >= 2048



adcValue >= 1024



adcValue < 1024

보충 Lab: 가변저항값에 따라 연속적인 LED Dimming

```
GTM_TOM0_TGC0_GLB_CTRL.U |= 0x1 << HOST_TRIG_BIT_LSB_IDX;
unsigned short duty = 0;


while(1)
{
    VADC_startConversion();
    unsigned int adcResult = VADC_readResult();


    duty = 12500 * adcResult / 4096;
    GTM_TOM0_CH1_SR1.U = duty;


}
return (1);
```

보충 Lab: RGB LED에 대해 Dimming을 시도해보자 (1/3)

```
// set TGC0 to enable GTM TOM0 channel 2, 3, 15
GTM_TOM0_TGC0_GLB_CTRL.B.UPEN_CTRL2 |= 0x2;  // TOM0 channel 2 enable
GTM_TOM0_TGC0_GLB_CTRL.B.UPEN_CTRL3 |= 0x2;  // TOM0 channel 3 enable
GTM_TOM0_TGC1_GLB_CTRL.B.UPEN_CTRL7 |= 0x2;  // TOM0 channel 15 enable

GTM_TOM0_TGC0_ENDIS_CTRL.B.ENDIS_CTRL2 |= 0x2;   // enable channel 2 on update trigger
GTM_TOM0_TGC0_ENDIS_CTRL.B.ENDIS_CTRL3 |= 0x2;   // enable channel 3 on update trigger
GTM_TOM0_TGC1_ENDIS_CTRL.B.ENDIS_CTRL7 |= 0x2;   // enable channel 15 on update trigger

GTM_TOM0_TGC0_OUTEN_CTRL.B.OUTEN_CTRL2 |= 0x2;   // enable channel 2 output on update trigger
GTM_TOM0_TGC0_OUTEN_CTRL.B.OUTEN_CTRL3 |= 0x2;   // enable channel 3 output on update trigger
GTM_TOM0_TGC1_OUTEN_CTRL.B.OUTEN_CTRL7 |= 0x2;   // enable channel 15 output on update trigger
```

보충 Lab: RGB LED에 대해 Dimming을 시도해보자 (2/3)

```
// TOM 0_2
GTM_TOM0_CH2_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                    // h

GTM_TOM0_CH2_CTRL.U &= ~(0x7 << CLK_SRC_SR_BIT_LSB_IDX);
GTM_TOM0_CH2_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;             //

GTM_TOM0_CH2_SR0.U = 12500 - 1;                                  // PWM
//GTM_TOM0_CH2_SR1.U = 12500 - 1;                                  //

// TOM 0_3
GTM_TOM0_CH3_CTRL.U |= 0x1 << SL_BIT_LSB_IDX;                    // h

GTM_TOM0_CH3_CTRL.U &= ~(0x7 << CLK_SRC_SR_BIT_LSB_IDX);
GTM_TOM0_CH3_CTRL.U |= 0x1 << CLK_SRC_SR_BIT_LSB_IDX;             //

GTM_TOM0_CH3_SR0.U = 12500 - 1;                                  // PWM
//GTM_TOM0_CH3_SR1.U = 125 - 1;                                    // du

// TOM 0_15
GTM_TOM0_CH15_CTRL.B.SL |= 0x1;                          // high signal lev
GTM_TOM0_CH15_CTRL.B.CLK_SRC_SR |= 0x1;                    // clock source

GTM_TOM0_CH15_SR0.U = 12500 - 1;                                 // PW
//GTM_TOM0_CH15_SR1.U = 125 - 1;                                   // c
```

```
// TOUT pin selection
GTM_TOUTSEL6.U &= ~(0x3 << SEL7_BIT_LSB_IDX);
GTM_TOUTSEL0.U &= ~(0x3 << SEL7_BIT_LSB_IDX);
GTM_TOUTSEL6.U &= ~(0x3 << SEL11_BIT_LSB_IDX);
GTM_TOUTSEL6.U &= ~(0x3 << SEL9_BIT_LSB_IDX);
```

보충 Lab: RGB LED에 대해 Dimming을 시도해보자 (3/3)

```
GTM_TOM0_TGC0_GLB_CTRL.U |= 0x1 << HOST_TRIG_BIT_LSB_IDX;
GTM_TOM0_TGC1_GLB_CTRL.U |= 0x1 << HOST_TRIG_BIT_LSB_IDX;

unsigned short duty = 0;

while(1)
{
    VADC_startConversion();
    unsigned int adcResult = VADC_readResult();

    duty = 12500 * adcResult / 4096;

    GTM_TOM0_CH2_SR1.U  = duty;
    GTM_TOM0_CH3_SR1.U  = duty;
    GTM_TOM0_CH15_SR1.U = duty;

}
return (1);
```

보충 Lab: Buzzer를 Drive해보자 (1/2)

```
void initBuzzer(void)
{
    P02_IOCR0.B.PC3 = 0x11;
}
```

PWM에 인가되는 클럭은
6250KHz

```
void initGTM(void)
{
  // set GTM TOM0 channel 11 - Buzzer
  GTM_TOM0_TGC1_GLB_CTRL.B.UPEN_CTRL3     |= 0x2;      // TOM0 channel 11 enable
  GTM_TOM0_TGC1_ENDIS_CTRL.B.ENDIS_CTRL3  |= 0x2;      // enable channel 11 on update trigger
  GTM_TOM0_TGC1_OUTEN_CTRL.B.OUTEN_CTRL3  |= 0x2;      // enable channel 11 output on update trigger


  // TOM 0_11
  GTM_TOM0_CH11_CTRL.B.SL = 0x1;                       // high signal level for duty cycle
  GTM_TOM0_CH11_CTRL.B.CLK_SRC_SR = 0x1;               // clock source --> CMU_FXCLK(1) = 6250 kHz
  GTM_TOM0_CH11_SR0.B.SR0 = 12500 - 1;                 // PWM freq. = 6250 kHz / 12500 = 500 Hz
  GTM_TOM0_CH11_SR1.B.SR1 = 6250 - 1;                  // duty cycle = 6250 / 12500 = 50 %

  // TOUT pin selection
  GTM_TOUTSEL0.B.SEL3 = 0x0;                           // TOUT3  --> TOM0 channel 11
```

보충 Lab: Buzzer를 Drive해보자 (1/2)

```
// from 3 octave C ~ 4 octave C {C, D, E, F, G, A, B, C}
unsigned int duty[8] = {130, 146, 164, 174, 195, 220, 246, 262};
```

```
while(1)
{
    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[0];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[0];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[1];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[1];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[2];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[2];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[3];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[3];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[4];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[4];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[5];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[5];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[6];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[6];

    for(unsigned int i = 0; i < 1000000000; i++);
    GTM_TOM0_CH11_SR0.B.SR0 = 6250000 / duty[7];
    GTM_TOM0_CH11_SR1.B.SR1 = 3125000 / duty[7];
}
```

6250khz -> 초당 카운터값 6250000번 증가
➔ SR0 (period) 6250000 설정시 1초 지나면 period match됨, 따라서 주기 1Hz PWM 생성됨

따라서 SR0 = 1/130인가시, 카운터 match값이 1/130로 줄어들므로, 130배 빠른속도로 match된다 따라서 130Hz PWM만들어진다.

SR1은 50%로 해서 duty 50%로 설정

감사합니다. 휴식~~