



EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정

[TMS 570_etPWM(w/UART, 서보모터)]

제 1기

2022. 03. 04

박태인

etPWM(w/UART)



1월 22 13 : 46 • workspace_v11 - noos_uart_recieve/source/HL_sys_main.c - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer: external_gpio_test, nonos_rti_led, noos_etpwm_control_with_uart [Active], noos_uart_recieve, noos_uart_send

HAL Code Generator - [GIO]

TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4

Port A Port B

VIM: Low Priority: Falling Edge:

Bit 4

DOUT: 0 DIR: PDR: PSL: GIOA[4]

DIN: VIM: High Priority: Enable: Rising Edge:

Bit 5

DOUT: 0 DIR: PDR: PSL: GIOA[5]

Output

EQEP
FEE
AJSM
Code complete

For Help, press F1

New Project

Family: TMS570LS04x, TMS570LS03x, TMS570LS02x, RM42x, RM41x, TMS570LS09x_07x, RM44x, TMS570LC43x, RM57Lx

Device: TMS570LC4357ZWT, TMS570LC4357ZWT_FREE...

Name: noos_etpwm_control_with_uart

Location: /home/taein/workspace_v11/noos_etpwm_control_with_uart

Create directory for project

OK Cancel

Device Explorer

TMS570LC4357ZWT

- SYSTEM
 - HL_stdtypes.h
 - HL_sys_common.h
 - HL_reg_system.h
 - HL_reg_flash.h
 - HL_reg_l2ramw.h
 - HL_reg_vim.h
 - HL_reg_pbist.h
 - HL_reg_stc.h
 - HL_reg_efc.h
 - HL_reg_pcr.h
 - HL_reg_pmm.h
 - HL_reg_dma.h
 - HL_reg_ccm5.h
 - HL_sys_core.h
 - HL_system.h
 - HL_sys_vim.h
 - HL_sys_mpu.h
 - HL_sys_pmu.h
 - HL_sys_pcr.h
 - HL_sys_pmm.h
 - HL_sys_dma.h
 - HL_sys_core.asm
 - HL_sys_intvecs.asm
 - HL_sys_mpu.asm
 - HL_sys_pmu.asm
 - HL_sys_pcr.c
 - HL_sys_pmm.c
 - HL_sys_dma.c

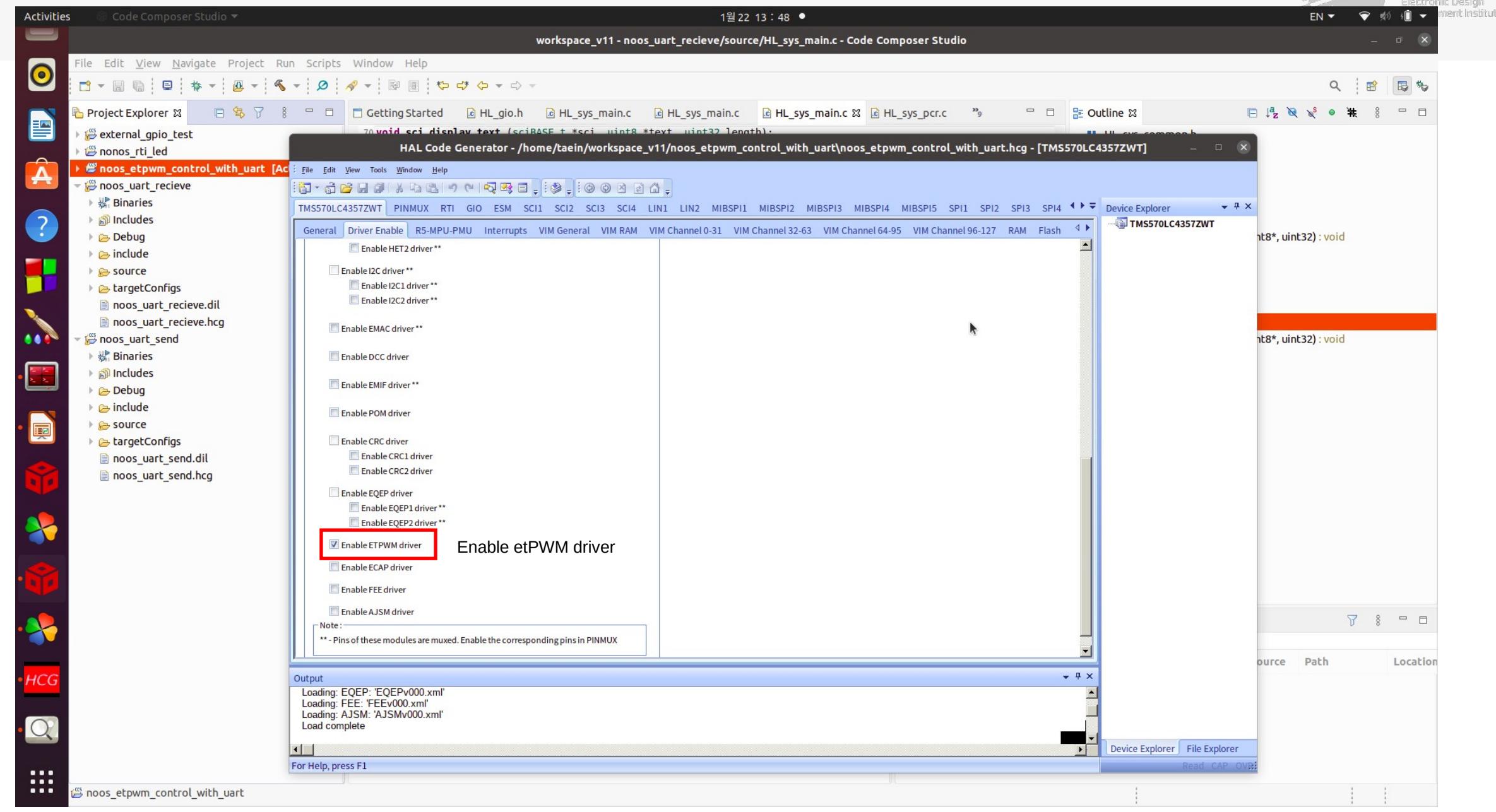
File Explorer

source Path Location

Read CAP OVR

noos_etpwm_control_with_uart

etPWM(w/UART)



etPWM(w/UART)



Code Composer Studio - workspace_v11 - noos_uart_recieve/source/HL_sys_main.c

1월 22 13 : 48

File Edit View Navigate Project Run Scripts Window Help

Project Explorer: noos_etpwm_control_with_uart [Active]

- external_gpio_test
- nonos_rti_led
- noos_uart_recieve
- Binaries
- Includes
- Debug
- include
- source
- targetConfigs
 - noos_uart_recieve.dil
 - noos_uart_recieve.hcg
- noos_uart_send
- Binaries
- Includes
- Debug
- include
- source
- targetConfigs
 - noos_uart_send.dil
 - noos_uart_send.hcg

HAL Code Generator - /home/taein/workspace_v11/noos_etpwm_control_with_uart/noos_etpwm_control_with_uart.hcg - [TMS570LC4357ZWT]

TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 SPI3 SPI4

Enable Driver Compilation

Pwm 신호 출력을 위한 GIO driver

Enable RTI driver Mark/Unmark all drivers

Enable GIO driver **

Sci 통신을 위한 driver

Enable SCI drivers

- Enable SCI3 driver **
- Enable SCI4 driver **

Enable LIN drivers

- Enable LIN1 driver ** / Enable SCI1 driver **
- Enable LIN2 driver ** / Enable SCI2 driver **

Enable MIBSPI drivers

- Enable MIBSPI1 driver ** Enable SPI1 driver **
- Enable MIBSPI2 driver ** Enable SPI2 driver **
- Enable MIBSPI3 driver ** Enable SPI3 driver **
- Enable MIBSPI4 driver ** Enable SPI4 driver **
- Enable MIBSPI5 driver ** Enable SPI5 driver **

Enable CAN drivers

- Enable CAN1 driver
- Enable CAN2 driver
- Enable CAN3 driver
- Enable CAN4 driver **

Enable ADC drivers

Output

Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete

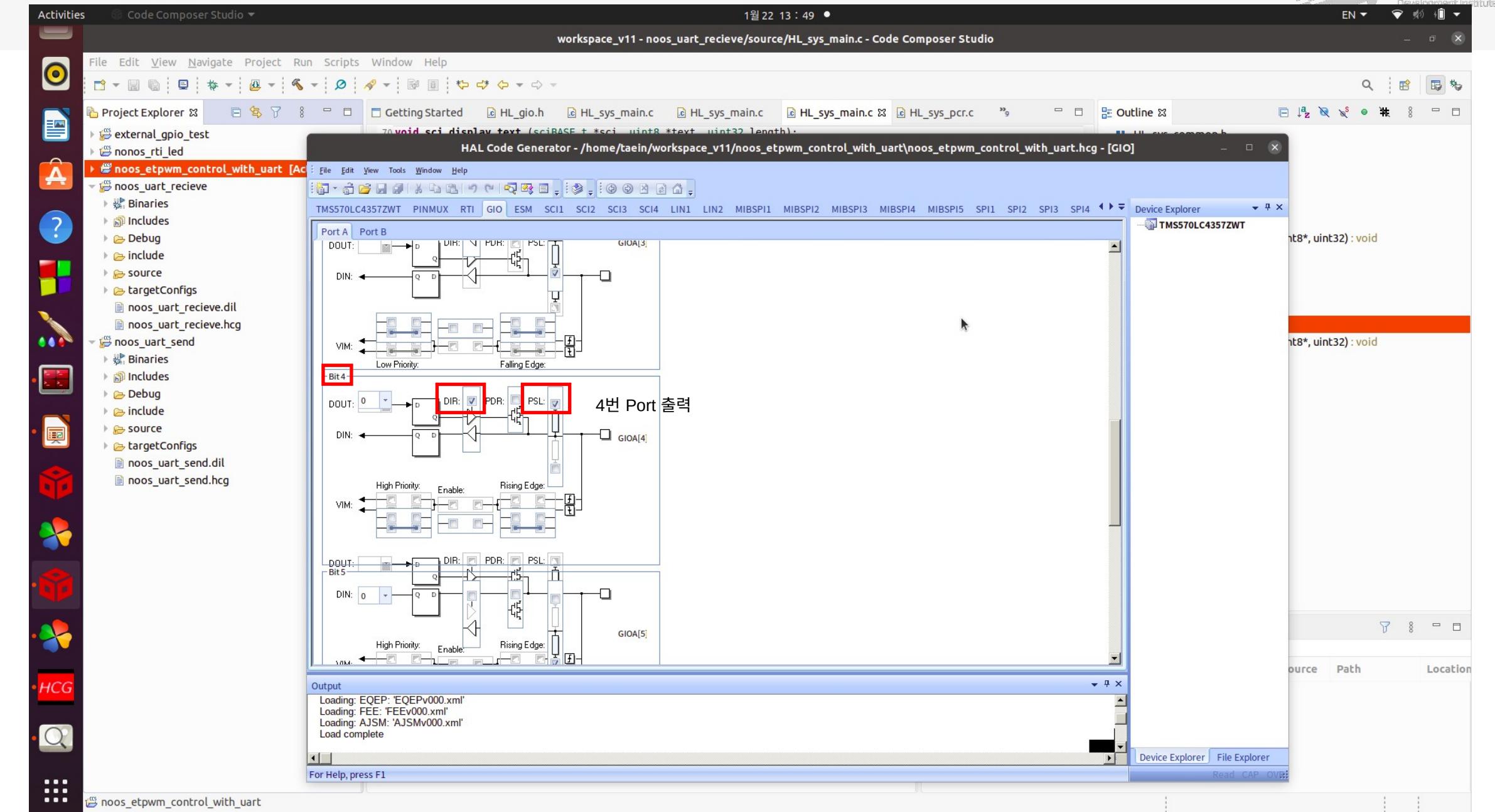
Device Explorer

File Explorer

For Help, press F1

Read CAP OVF

etPWM(w/UART)



etPWM(w/UART)

Activities Code Composer Studio • 1월 22 13 : 50 • workspace_v11 - noos_uart_recieve/source/HL_sys_main.c - Code Composer Studio EN ▾

File Edit View Navigate Project Run Scripts Window Help

Project Explorer ▾ Getting Started HL_gio.h HL_sys_main.c HL_sys_main.c HL_sys_main.c HL_sys_pcr.c

noos_uart_recieve [Ac] noos_etpwm_control_with_uart [Ac]

Binaries Includes Debug include source targetConfigs noos_uart_recieve.dil noos_uart_recieve.hcg

noos_uart_send Binaries Includes Debug include source targetConfigs noos_uart_send.dil noos_uart_send.hcg

HAL Code Generator - /home/taein/workspace_v11/noos_etpwm_control_with_uart\noos_etpwm_control_with_uart.hcg - [PINMUX]

Start Page TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2

Pin Muxing Input Pin Muxing Special Pin Muxing

Enable / Disable Peripherals

HET1	GIOA	MIBSPI2	MIBSPI1	SCI3	RMI
HET2	GIQB	MIBSPI4	MIBSPI3	SCI4	MII
EMIF	EQEP	AD1EVT	MIBSPI5	LIN2/SCI2	CAN4
ETPWM	ECAP	AD2EVT	I2C1	I2C2	

Note: GIO pins are mapped to two terminals. The checkboxes enable both the default and alternate terminals. Remove the unwanted terminal to avoid conflicts. MII have dedicated pins. Alternate terminals are enabled using the MII RMI and MII checkboxes does not set the functional mode. Enable them in Special Pinmuxing tab.

Total Conflicts 0 List Conflicts

Ball	Default Mux	Mux Option 1	Mux Option 2	Mux Option 3	Mux Option 4	Mux Option 5	Conflict?
A4	N2HET1[16]	NONE	NONE	ETPWM1SYNC1	NONE	ETPWM1SYNC0	
A13	N2HET1[17]	EMIF_nOE	SCI4RX	NONE	NONE	NONE	
A14	N2HET1[26]	NONE	MII_RXD[1]	RMII_RXD[1]	NONE	NONE	
B2	MIBSPI3NCS[2]	I2C1_SDA	NONE	N2HET1[27]	NONE	nTZ1_2	
B3	N2HET1[22]	EMIF_nDQM[3]	NONE	NONE	NONE	NONE	
B4	N2HET1[12]	MIBSPI4NCS[5]	MII_CRS	RMII_CRS_DV	NONE	NONE	
B5	GIOA[5]	NONE	NONE	EXTCLKIN	NONE	eTPWM1A	

PWM 포트(hw상 GIOA[5])

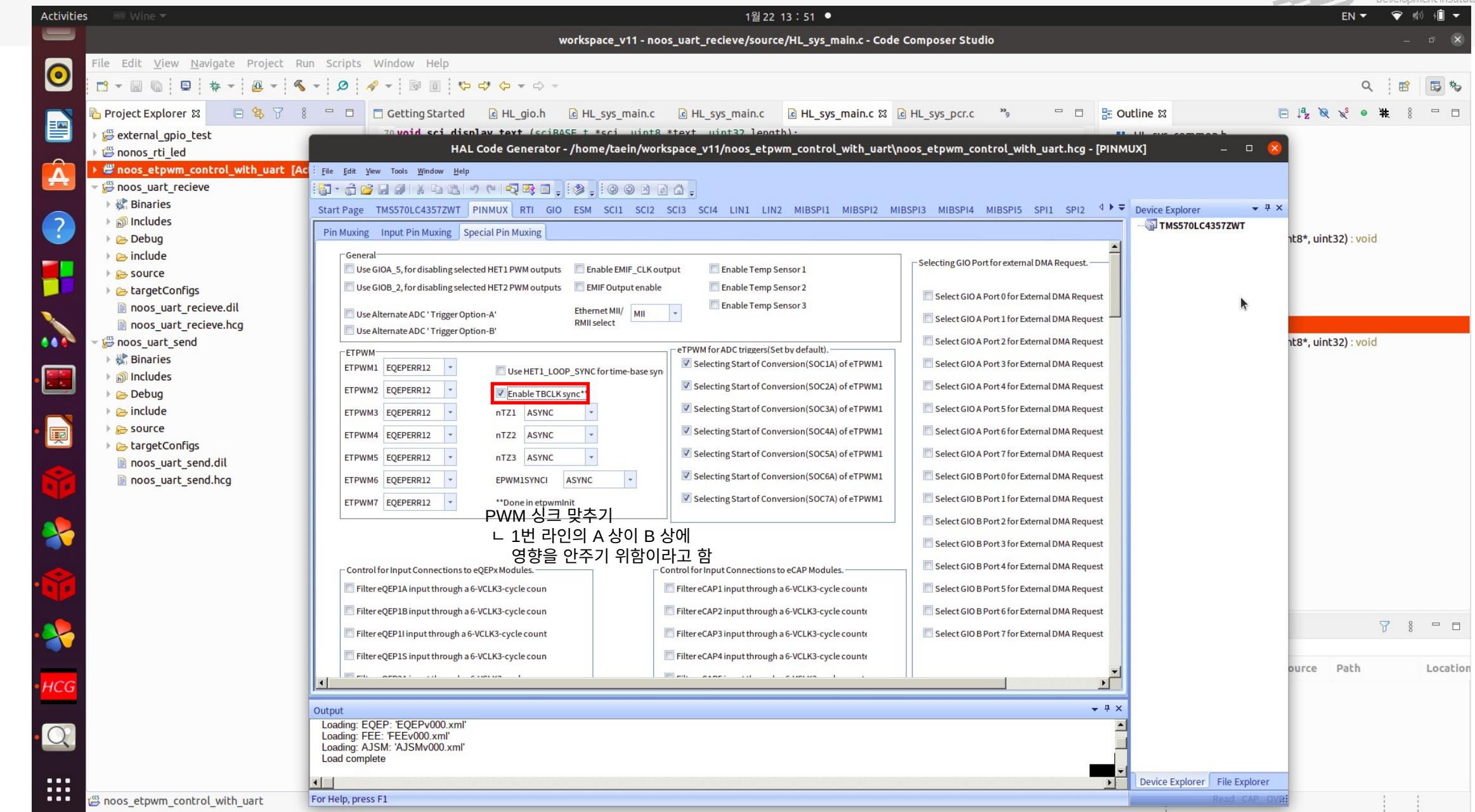
Output

```
Loading: EQEP: 'EQEPv000.xml'  
Loading: FEE: 'FEEv000.xml'  
Loading: AJSM: 'AJSMv000.xml'  
Load complete
```

Device Explorer File Explorer Read CAP OVER

For Help, press F1

etPWM(w/UART)



etPWM(w/UART)

Activities Wine ▾ 1월 22 13 : 54 ● workspace_v11 - noos_uart_recieve/source/HL_sys_main.c - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer  HAL Code Generator - /home/taein/workspace_v11/noos_etpwm_control_with_uart\noos_etpwm_control_with_uart.hcg - [ETPWM]

File Edit View Tools Window Help

Device Explorer Device Explorer TMS570LC4357ZWT

General ETPWM1 ETPWM2 ETPWM3 ETPWM4 ETPWM5 ETPWM6 ETPWM7

Enable ETPWM modules

Enable ETPWM1 **PWM enable**

Enable ETPWM2

Enable ETPWM3

Enable ETPWM4

Enable ETPWM5

Enable ETPWM6

Enable ETPWM7

Note:

** - etpwmlnit function sets the time-base counters in up-count mode. Application can configure the module in a different mode using other functions in this driver(Sample code provided in the examples folder). In that case, application need not call etpwmlnit function.

Output

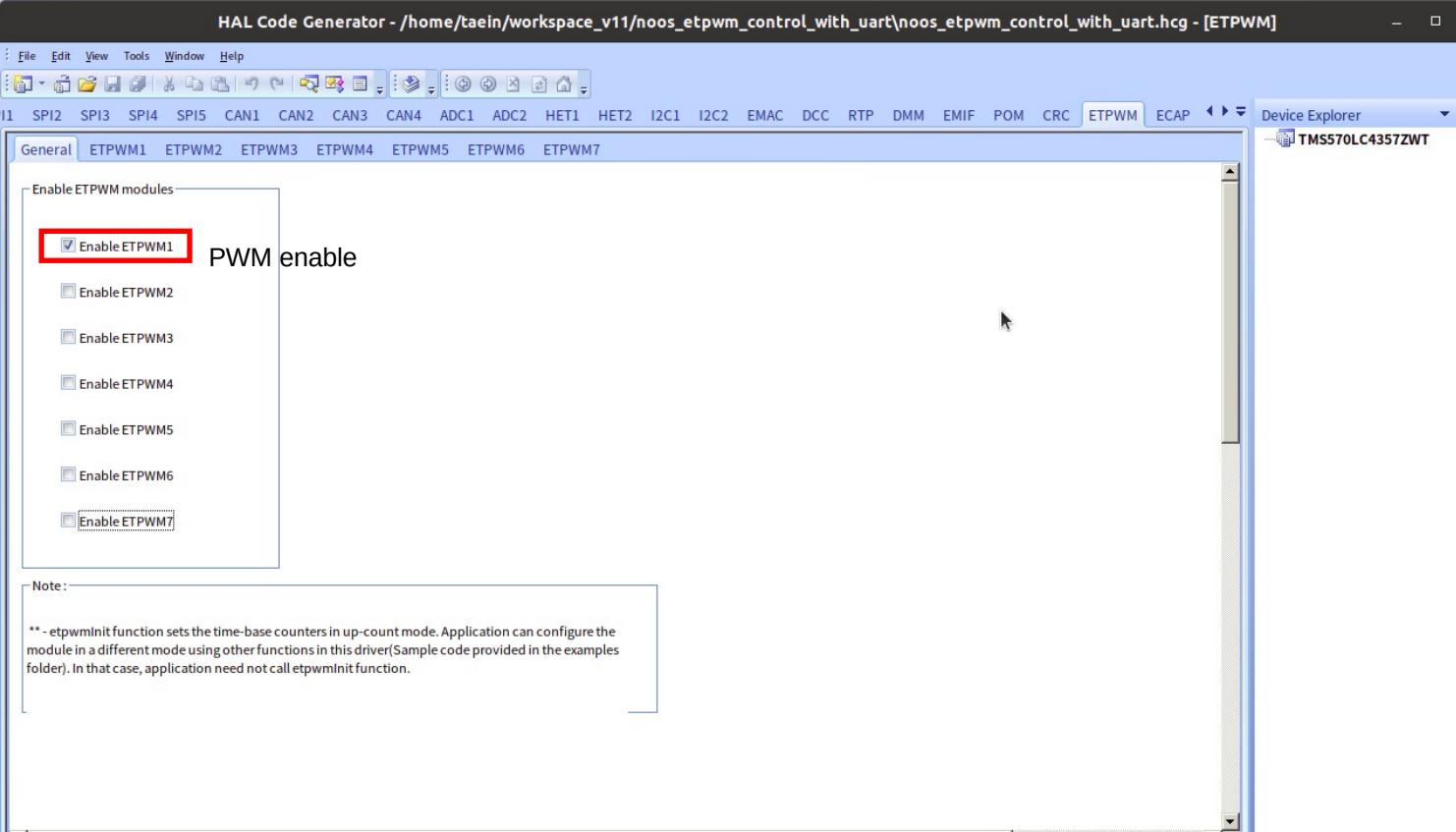
Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete

For Help, press F1

Source Path Location

Device Explorer File Explorer Read CAP OVR

noos_etpwm_control_with_uart



etPWM(w/UART)

Activities Wine 1월 22 13 : 59 workspace_v11 - noos_uart_recieve/source/HL_sys_main.c - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer noos_uart_recieve [Active]

- external_gpio_test
- nonos_rti_led
- noos_etpwm_control_with_uart
- Binaries
- Includes
- Debug
- source
- targetConfigs
- noos_uart_recieve.dil
- noos_uart_recieve.hcg
- noos_uart_send
- Binaries
- Includes
- Debug
- source
- targetConfigs
- noos_uart_send.dil
- noos_uart_send.hcg

HAL Code Generator - /home/taein/workspace_v11/noos_etpwm_control_with_uart/noos_etpwm_control_with_uart.hcg - [TMS570LC4357ZWT]

Start Page TMS570LC4357ZWT PINMUX RTI GIO ESM SCI1 SCI2 SCI3 SCI4 LIN1 LIN2 MIBSPI1 MIBSPI2 MIBSPI3 MIBSPI4 MIBSPI5 SPI1 SPI2 Device Explorer TMS570LC4357ZWT

GCM CLK Srcs Standard Src PLL1 HCLK Divider GCLK Config GCLK 300.000 Notes GCLK Max -- 330 MHz HCLK Max -- 150 MHz VCLK Max -- 110 MHz VCLK2 Max -- 110 MHz VCLK3 Max -- 110 MHz VCLKA1 Max -- 110 MHz VCLKA2 Max -- 110 MHz VCLKA4 Max -- 110 MHz RTICLK Max -- VCLK Freq CLK... 10Mhz

VCLKA1 Src VCLK VCLKA2 Src VCLK VCLKA4 Src VCLK

VCLKA1 Config VCLKA2 Config VCLKA4 Config

VCLKA1 Divider VCLKA2 Divider VCLKA4 Divider

VCLKA1 Post Src PLL1 RTI1 Pre Src RTI1 Post Src RTI1 Config RTI1 CLK

VCLKA2 Divider VCLKA4 Divider

VCLKA2 Post Src VCLKA4 Post Src VCLKA2_DIV VCLKA4_DIVR

VCLKA2 Config VCLKA4 Config

VCLKA2 DIV VCLKA4 DIV

VCLKA2_S VCLKA4_S

RTI1 CLK

PLL2

Oscillator 16.0

PLL1 300.0

External 1 00.0

Clock Tree

Output

Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete

For Help, press F1

Device Explorer File Explorer Read CAP OVE

noos_etpwm_control_with_uart

etPWM(w/UART)

Activities Wine 1월 22 14 : 00 workspace_v11 - noos_uart_recieve/source/HI_sys_main.c - Code Composer Studio EN

File Edit View Navigate Project Run Scripts Window Help

Project Explorer noos_etpwm_control_with_uart [Active]

external_gpio_test

noos_rti_led

noos_uart_recieve

Binaries

Includes

Debug

include

source

targetConfigs

noos_uart_recieve.dil

noos_uart_recieve.hcg

noos_uart_send

Binaries

Includes

Debug

include

source

targetConfigs

noos_uart_send.dil

noos_uart_send.hcg

HAL Code Generator - /home/taein/workspace_v11/noos_etpwm_control_with_uart\noos_etpwm_control_with_uart.hcg - [ETPWM]

File Edit View Tools Window Help

General ETPWM1 ETPWM2 ETPWM3 ETPWM4 ETPWM5 ETPWM6 ETPWM7

Clock Configuration

TB Clock (MHz): 110.000

VCLK3 (MHz): 10.000

Actual TB Clock (MHz): 10.000

10 Mhz 확인

HSPCLKDIV: 0

CLKDIV: 0

PWM Configuration

High Polarity

Low Polarity

tPeriod

tDuty

Disable delay

Enable delay

Rising Edge Delay

Delay(ns) 1000.000

Falling Edge Delay

Invert Polarity

Enable delay

Invert Polarity

Duty[%] 50

Period[ns] 1000

High Polarity

Period

Output

Loading: EQEP: 'EQEPv000.xml'
Loading: FEE: 'FEEv000.xml'
Loading: AJSM: 'AJSMv000.xml'
Load complete

Device Explorer TMS370LC4357ZWT

Source Path Location

For Help, press F1

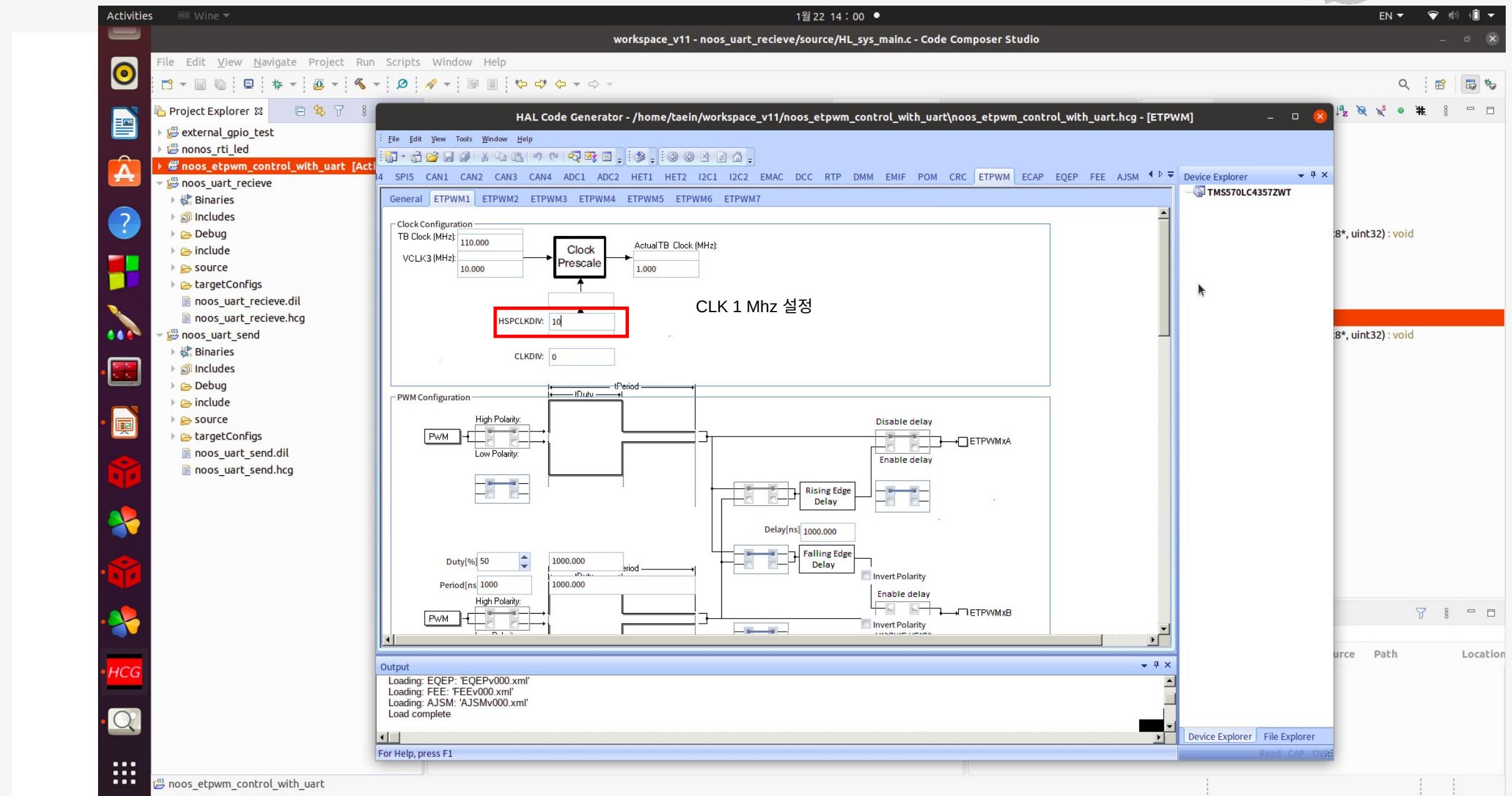
Read CAP OVR

File Explorer

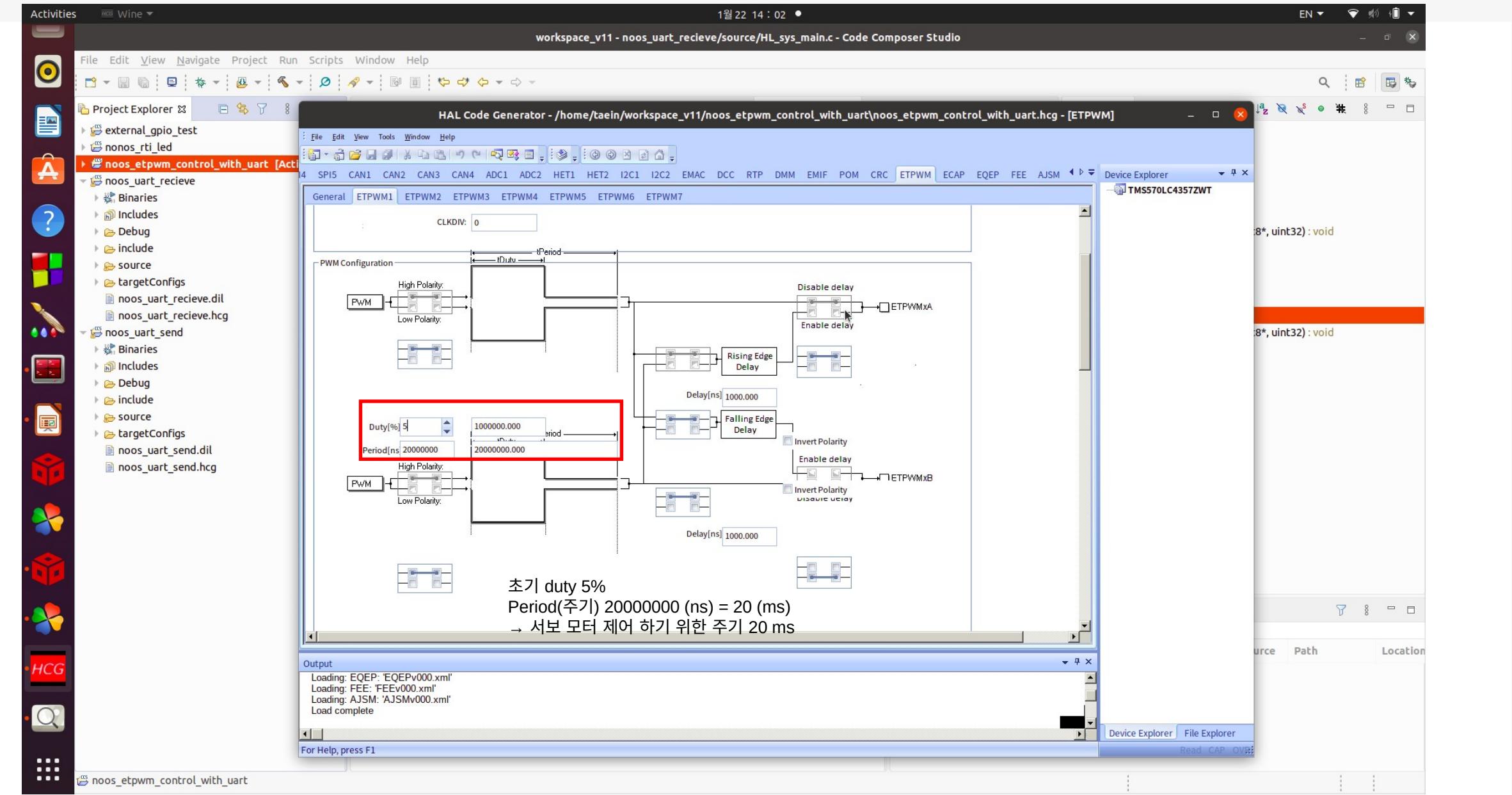
noos_etpwm_control_with_uart

The screenshot displays the Code Composer Studio interface for a TMS370LC4357ZWT microcontroller. The main window shows the HAL Code Generator for the 'noos_etpwm_control_with_uart' project. The ETPWM1 tab is active, showing the configuration of two PWM channels. The top section, 'Clock Configuration', shows the TB Clock at 110.000 MHz and the VCLK3 at 10.000 MHz, resulting in an Actual TB Clock of 10.000 MHz, which is highlighted with a red box. The bottom section, 'PWM Configuration', shows two PWM signals with a 50% duty cycle and a 1000 ns period. These signals are processed through various logic blocks (Rising Edge Delay, Falling Edge Delay, Invert Polarity, Enable/Disable delay) before being output as ETPWMxA and ETPWMxB. The output panel indicates that the EQEP, FEE, and AJSM modules are being loaded.

etPWM(w/UART)



etPWM(w/UART)



etPWM(w/UART) – code 및 내용 분석



```
50 #include "HL_sys_common.h"          * 헤더 선언
51
52 /* USER CODE BEGIN (1) */           ↳ gio, sci, etpwm, system
53 #include "HL_gio.h"                 sci에 활용할 string.h, stdio.h
54 #include "HL_sci.h"
55 #include "HL_etpwm.h"
56 #include "HL_system.h"
57
58 #include <string.h>
59 #include <stdio.h>

70 /* USER CODE BEGIN (2) */
71 void sci_display_text (sciBASE_t *sci, uint8 *text, uint32 length);
72 void wait (uint32 time);           Putty 창에 uart 통신 나타내기 위한 함수
73 void set_pwm();                  UART는 sciREG1
74
75 #define UART      sciREG1
76
77 uint32 receive_data;
78 uint32 pwm_value;
79 uint32 tmp;                      Duty 비 제어에 활용 예정
80
81 uint32 pwm_duty[] = { 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500 }; // 20000000ns(20ms)에서 500이
82                                         // 쉽게 생각해서 전체 pwm 이 20이라고 봤을 때
83                                         // 500이 0.5 정도
```

```
86 void sci_display_text(sciBASE_t *sci, uint8 *text, uint32 length)
87 {
88     while(length--)
89     {
90         while ((UART->FLR & 0x4) == 4)           // 조건문에 해당되면 busy 하다는 얘기
91         ;
92
93         sciSendByte(UART, *text++);           // 1 바이트씩 움직일 거니까 포인터로
94     }
95 }
96
97 void wait(uint32 time)
98 {
99     int i;                                앞 서 있었던 내용 이므로
00     for(i=0; i<time; i++)
01     ;
02 }
03 }
```

앞 서 있었던 내용 이므로
설명은 생략

etPWM(w/UART) – code 및 내용 분석



```
int main(void)
{
/* USER CODE BEGIN (3) */
    char buf[128];
    unsigned int buf_len;

    sciInit();

    sprintf(buf, "*SCI Init Success!\n\r\0");
    buf_len = strlen(buf);
    sci_display_text(UART, (uint8 *)buf, buf_len);

    gioInit();

    sprintf(buf, "*GIO Init Success!\n\r\0");
    buf_len = strlen(buf);
    sci_display_text(UART, (uint8 *)buf, buf_len);

    etpwmInit(); // This line is highlighted with a red box

    sprintf(buf, "*EPWM Init Success!\n\r\0");
    buf_len = strlen(buf);
    sci_display_text(UART, (uint8 *)buf, buf_len);

    sprintf(buf, "Press Number!\n\r\0");
    buf_len = strlen(buf);
    sci_display_text(UART, (uint8 *)buf, buf_len);
```

Sci , gio, etpwm init 과
숫자 입력 받도록 만들 → 입력 하는 숫자에 따라 duty 비 설정하게 됨

이번에 중점적으로 알아 봐야 할 부분은 etpwmInit() !!

35.1 Introduction

An effective PWM peripheral must be able to generate complex pulse width waveforms with minimal CPU overhead or intervention. It needs to be highly programmable and very flexible while being easy to understand and use. The ePWM unit described here addresses these requirements by allocating all needed timing and control resources on a per PWM channel basis. Cross coupling or sharing of resources has been avoided; instead, the ePWM is built up from smaller single channel modules with separate resources that can operate together as required to form a system. This modular approach results in an orthogonal architecture and provides a more transparent view of the peripheral structure, helping users to understand its operation quickly.

In this document the letter x within a signal or module name is used to indicate a generic ePWM instance on a device. For example output signals EPWMxA and EPWMxB refer to the output signals from the ePWMx instance. Thus, EPWM1A and EPWM1B belong to ePWM1 and likewise EPWM4A and EPWM4B belong to ePWM4.

이 문서에서 신호 또는 모듈 이름 내의 문자 x는 일반 ePWM 장치에 인스턴스를
나타내는 데 사용됩니다.

예를 들어 출력 신호 EPWMxA 및 EPWMxB는 ePWMx 의 인스턴스.
따라서 EPWM1A 및 EPWM1B는 ePWM1에 속하고 마찬가지로
EPWM4A 및 EPWM4B는 ePWM4에 속합니다.

etPWM(w/UART) – code 및 내용 분석

ePWM 모듈은 2개의 PWM 출력(EPWMxA 및 EPWMxB)으로 구성된 하나의 완전한 PWM 채널을 나타냅니다. 여러 ePWM 모듈은 그림 35-1과 같이 장치 내에서 인스턴스화됩니다. 각 ePWM 인스턴스는 동일하며 1부터 시작하는 숫자 값으로 표시됩니다. 예를 들어, ePWM1은 시스템의 첫 번째 인스턴스이고 ePWM3은 세 번째 인스턴스이며 ePWMx는 임의의 인스턴스를 나타냅니다.

35.1.1 Submodule Overview

The ePWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. Multiple ePWM modules are instanced within a device as shown in Figure 35-1. Each ePWM instance is identical and is indicated by a numerical value starting with 1. For example, ePWM1 is the first instance and ePWM3 is the third instance in the system, and ePWMx indicates any instance.

The ePWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral modules (eCAP). Modules can also operate stand-alone.

Each ePWM module supports the following features:

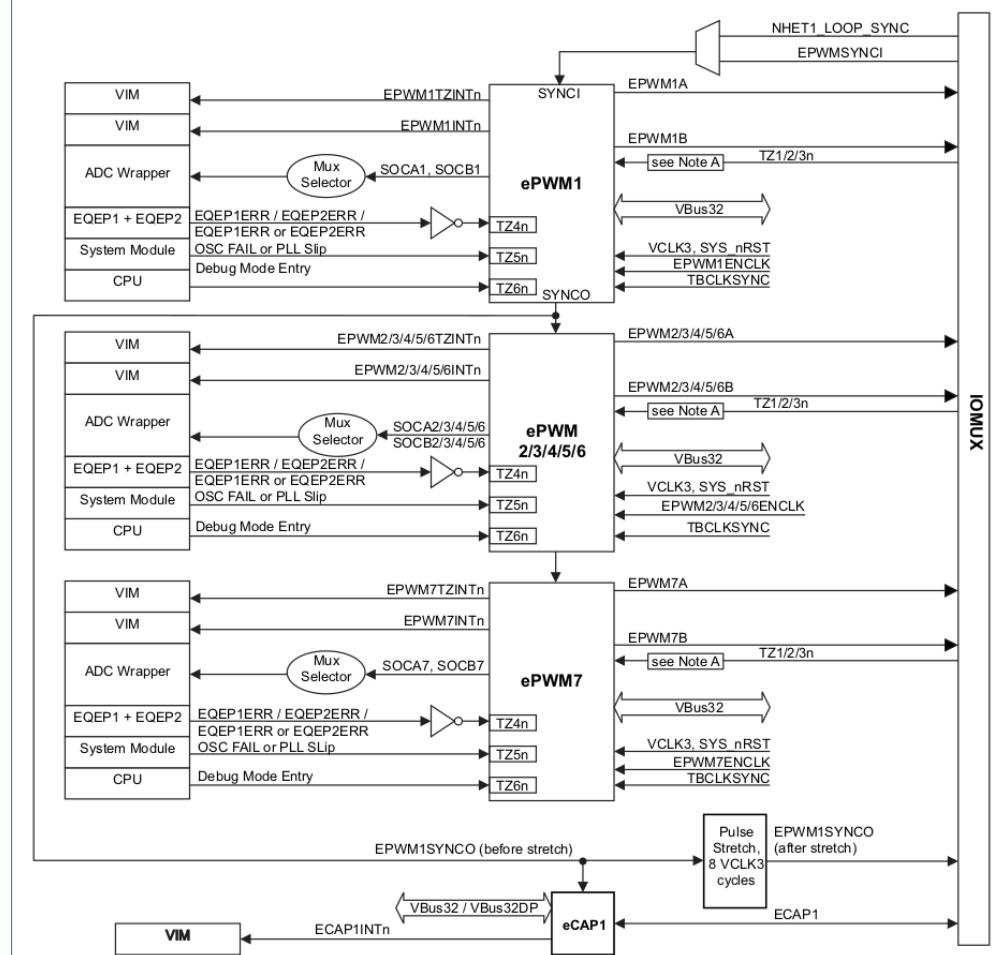
- Dedicated 16-bit time-base counter with period and frequency control
- Two PWM outputs (EPWMxA and EPWMxB) that can be used in the following configurations:
 - Two independent PWM outputs with single-edge operation
 - Two independent PWM outputs with dual-edge symmetric operation
 - One independent PWM output with dual-edge asymmetric operation
- Asynchronous override control of PWM signals through software.
- Programmable phase-control support for lag or lead operation relative to other ePWM modules.
- Hardware-locked (synchronized) phase relationship on a cycle-by-cycle basis.
- Dead-band generation with independent rising and falling edge delay control.
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions.
- A trip condition can force either high, low, or high-impedance state logic levels at PWM outputs.
- All events can trigger both CPU interrupts and ADC start of conversion (SOC)
- Programmable event prescaling minimizes CPU overhead on interrupts.
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives.

Each ePWM module is connected to the input/output signals shown in Figure 35-1. The signals are described in detail in subsequent sections.

Each ePWM module consists of eight submodules and is connected within a system via the signals shown in Figure 35-2.

ePWM 모듈은 클록 동기화 방식을 통해 함께 연결되어 필요할 때 단일 시스템으로 작동할 수 있습니다. 또한 이 동기화 체계는 eCAP(Capture Peripheral Module)로 확장될 수 있습니다. 모듈은 독립 실행형으로도 작동할 수 있습니다.

Figure 35-1. Multiple ePWM Modules



etPWM(w/UART) – code 및 내용 분석

35.1.1 Submodule Overview

The ePWM module represents one complete PWM channel composed of two PWM outputs: EPWMxA and EPWMxB. Multiple ePWM modules are instanced within a device as shown in [Figure 35-1](#). Each ePWM instance is identical and is indicated by a numerical value starting with 1. For example, ePWM1 is the first instance and ePWM3 is the third instance in the system, and ePWMx indicates any instance.

The ePWM modules are chained together via a clock synchronization scheme that allows them to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral modules (eCAP). Modules can also operate stand-alone.

Each ePWM module supports the following features:

- Dedicated 16-bit time-base counter with period and frequency control
- Two PWM outputs (EPWMxA and EPWMxB) that can be used in the following configurations:
 - Two independent PWM outputs with single-edge operation
 - Two independent PWM outputs with dual-edge symmetric operation
 - One independent PWM output with dual-edge asymmetric operation
- Asynchronous override control of PWM signals through software.
- Programmable phase-control support for lag or lead operation relative to other ePWM modules.
- Hardware-locked (synchronized) phase relationship on a cycle-by-cycle basis.
- Dead-band generation with independent rising and falling edge delay control.
- Programmable trip zone allocation of both cycle-by-cycle trip and one-shot trip on fault conditions.
- A trip condition can force either high, low, or high-impedance state logic levels at PWM outputs.
- All events can trigger both CPU interrupts and ADC start of conversion (SOC)
- Programmable event prescaling minimizes CPU overhead on interrupts.
- PWM chopping by high-frequency carrier signal, useful for pulse transformer gate drives.

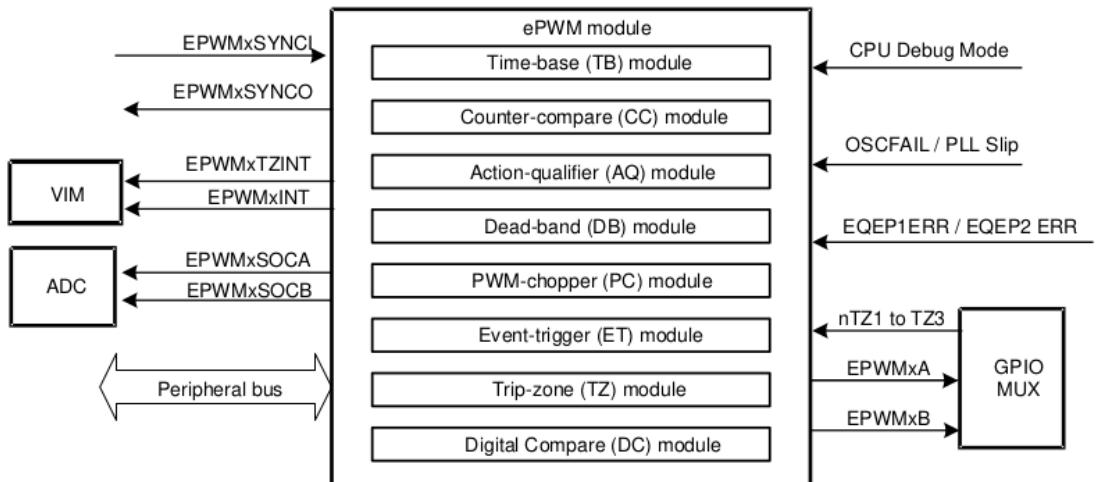
Each ePWM module is connected to the input/output signals shown in [Figure 35-1](#). The signals are described in detail in subsequent sections.

Each ePWM module consists of eight submodules and is connected within a system via the signals shown in [Figure 35-2](#).

각 ePWM 모듈은 다음 기능을 지원합니다.

- 주기 및 주파수 제어 기능이 있는 전용 16비트 타임베이스 카운터
- 다음 구성에서 사용할 수 있는 2개의 PWM 출력(EPWMxA 및 EPWMxB): – 단일 에지 작동이 가능한 2개의 독립적인 PWM 출력 – 듀얼 에지 대칭 작동이 가능한 2개의 독립적인 PWM 출력 – 듀얼 에지 비대칭 작동을 지원하는 1개의 독립적인 PWM 출력
- 소프트웨어를 통한 PWM 신호의 비동기식 오버라이드 제어
- 다른 ePWM 모듈에 비해 지연 또는 선행 작동을 위한 프로그래밍 가능한 위상 제어 지원.
- 주기별로 하드웨어 잠금(동기화) 단계 관제.
- 독립적인 상승 및 하강 에지 지연 제어를 통한 불감대 생성.
- 사이클별 트립 및 오류 조건에 대한 원샷 트립 모두의 프로그래밍 가능한 트립 영역 할당.
- 트립 조건은 PWM 출력에서 하이, 로우 또는 하이 임피던스 상태로직 레벨을 강제할 수 있습니다.
- 모든 이벤트는 CPU 인터럽트와 ADC 변환 시작(SOC)을 모두 트리거할 수 있습니다.
- 프로그래밍 가능한 이벤트 프리스케일링은 인터럽트 시 CPU 오버헤드를 최소화합니다.
- 고주파수 캐리어 신호에 의한 PWM 초핑, 펄스 트랜스포머 게이트 드라이브에 유용합니다.
- 각 ePWM 모듈은 그림 35-1과 같은 입력/출력 신호에 연결됩니다.
- 신호는 다음 섹션에서 자세히 설명합니다.
- 각 ePWM 모듈은 8개의 하위 모듈로 구성되며 그림 35-2에 표시된 신호를 통해 시스템 내에서 연결됩니다.

Figure 35-2. Submodules and Signal Connections for an ePWM Module



etPWM(w/UART) – code 및 내용 분석

```

1 void etpwmInit(void)
2 {
3 /* USER CODE BEGIN (1) */
4 /* USER CODE END */
5     uint16 TBCTL;           /**< 0x0002 Time-Base Control Register*/
6
7     /** @b initialize @b ETPWM1 */
8
9     /** - Sets high speed time-base clock prescale bits */
10    etpwmREG1->TBCTL = (uint16)5U << 7U;
11
12     7을 왼쪽으로 5bit shift → 0111 → 1110 0000
13
14     /** - Sets time-base clock prescale bits */
15    etpwmREG1->TBCTL |= (uint16)((uint16)0U << 10U);
16
17
18     10을 왼쪽으로 0 bit shift → 1010
19
20     따라서 TBCTL은 1110 1010

```

15	14	13	12	10	9	8
FREE	SOFT	PHSDIR		CLKDIV	HSPCLKDIV	
R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	
7	6	5	4	3	2	1 0
HSPCLKDIV	SWFSYNC	SYNCOSEL		PRDLD	PHSEN	CTRMODE
R/W-1	R/W-0	R/W-0		R/W-0	R/W-0	R/W-3h

LEGEND: R/W = Read/Write; -n = value after reset

1을 쓰면 일회성 동기화 펄스가 생성됩니다. 이 이벤트는 ePWM 모듈의 EPWMxSYNC1 입력과 OR 연결됩니다. SWFSYNC는 EPWMxSYNC1이 SYNCOSEL = 00으로 선택된 경우에만 유효(작동)합니다.

CTR = CMPB : 카운터 비교 B와 동일한 시간 기반 카운터(TBCTR = CMPB)

Table 35-1. ePWM Module Control and Status Register Set Grouped by Submodule

Name	Address Offset ⁽¹⁾	Size (x16)	Shadow	Privileged Mode Write Only?	Description
Time-Base Submodule Registers					
TBSTS	00h	1	No	No	Time-Base Status Register
TBCTL	02h	1	No	No	Time-Base Control Register

9-7	HSPCLKDIV		High Speed Time-base Clock Prescale Bits. These bits determine part of the time-base clock prescale value: TBCLK = VCLK3 / (HSPCLKDIV × CLKDIV)
		0 /1	
		1h /2 (default on reset)	

Bit	Field	Value	Description
6	SWFSYNC	0 1	Software Forced Synchronization Pulse. Writing a 0 has no effect and reads always return a 0. Writing a 1 forces a one-time synchronization pulse to be generated. This event is ORed with the EPWMxSYNC1 input of the ePWM module. SWFSYNC is valid (operates) only when EPWMxSYNC1 is selected by SYNCOSEL = 00.

5-4	SYNCOSEL	0 1h 2h 3h	Synchronization Output Select. These bits select the source of the EPWMxSYNCO signal. CTR = zero: Time-base counter equal to zero (TBCTR = 0x0000) CTR = CMPB : Time-base counter equal to counter-compare B (TBCTR = CMPB) Disable EPWMxSYNCO signal
-----	----------	---------------------	--

etPWM(w/UART) – code 및 내용 분석

따라서 TBCTL은 1110 1010

Figure 35-64. Time-Base Control Register (TBCTL) [offset = 02h]							
15	14	13	12	10	9	8	
FREE	SOFT	PHSDIR		CLKDIV		HSPCLKDIV	
R/W-0	R/W-0	R/W-0		R/W-0		R/W-0	
7	6	5	4	3	2	1	0
HSPCLKDIV	SWFSYNC	SYNCOSEL		PRDLD	PHSEN	CTRMODE	
R/W-1	R/W-0	R/W-0		R/W-0	R/W-0	R/W-3h	

LEGEND: R/W = Read/Write; -n = value after reset

3	PRDLD	0	Active Period Register Load From Shadow Register Select. The period register (TBPRD) is loaded from its shadow register when the time-base counter, TBCTR, is equal to zero.
		1	A write or read to the TBPRD register accesses the shadow register. Load the TBPRD register immediately without using a shadow register. A write or read to the TBPRD register directly accesses the active register.

섀도우 레지스터를 사용하지 않고 즉시 TBPRD 레지스터를 로드합니다. TBPRD 레지스터에 대한 쓰기 또는 읽기는 활성 레지스터에 직접 액세스합니다.

1-0	CTRMODE		Counter Mode. The time-base counter mode is normally configured once and not changed during normal operation. If you change the mode of the counter, the change will take effect at the next TBCLK edge and the current counter value shall increment or decrement from the value before the mode change. These bits set the time-base counter mode of operation as follows:
0			Up-count mode
1h			Down-count mode
2h			Up-down-count mode
3h			Stop-freeze counter operation (default on reset)

카운터 모드.

시간 기반 카운터 모드는 일반적으로 한 번 구성되며 정상 작동 중에는 변경되지 않습니다. 카운터 모드를 변경하면 다음 TBCLK 에지에서 변경 사항이 적용되고 현재 카운터 값은 모드 변경 전 값에서 증가하거나 감소합니다.

etPWM(w/UART) – code 및 내용 분석

```
void etpwmInit(void)
{
    /* - Sets time period or frequency for ETPWM block both PWMA and PWMB*/
    etpwmREG1->TBPRD = 19999U;
    uint16 TBPRD;           /*< 0x0008 Time-Base Period Register*/
}
```

08h

TBPRD

Time-Base Period Register

Table 35-26. Time-Base Period Register (TBPRD) Field Descriptions

Bits	Name	Description
15-0	TBPRD	<p>These bits determine the period of the time-base counter. This sets the PWM frequency.</p> <p>Shadowing of this register is enabled and disabled by the TBCTL[PRDLD] bit. By default this register is shadowed.</p> <ul style="list-style-type: none">If TBCTL[PRDLD] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the active register will be loaded from the shadow register when the time-base counter equals 0.If TBCTL[PRDLD] = 1, then the shadow is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.The active and shadow registers share the same memory map address.

이 비트는 시간축 카운터의 주기를 결정합니다. PWM 주파수를 설정합니다. 이 레지스터의 셜도잉은 TBCTL[PRDLD] 비트에 의해 활성화 및 비활성화됩니다. 기본적으로 이 레지스터는 숨겨져 있습니다.

- TBCTL[PRDLD] = 0 이면 셜도우가 활성화되고 쓰기 또는 읽기가 자동으로 셜도우 레지스터로 이동합니다. 이 경우 활성 레지스터는 시간이 흐를 때 셜도우 레지스터에서 로드됩니다. 기본 카운터는 0입니다.
- TBCTL[PRDLD] = 1 이면 셜도우가 비활성화되고 모든 쓰기 또는 읽기가 활성 레지스터, 즉 하드웨어를 능동적으로 제어하는 레지스터로 직접 이동합니다.
- 활성 및 셜도우 레지스터는 동일한 메모리 맵 주소를 공유합니다.

etPWM(w/UART) – code 및 내용 분석

```
void etpwmInit(void)
{
    /** - Setup the duty cycle for PWMA */
    etpwmREG1->CMPA = 1000U;
    uint16 CMPA;
    /**< 0x0010 Counter-Compare A Register*/
    CMPB 도 유사하다.
}
```

10h

CMPA

Counter-Compare A Register

활성 CMPA 레지스터의 값은 TBCTR(타임 베이스 카운터)과 지속적으로 비교됩니다. 값이 같을 때 카운터 비교 모듈은 "시간 기반 카운터가 카운터 비교 A와 같음" 이벤트를 생성합니다. 이 이벤트는 자격이 부여되고 하나 이상의 작업으로 변환되는 action-qualifier로 전송됩니다. 이러한 작업은 AQCTLA 및 AQCTLB 레지스터의 구성에 따라 EPWMxA 또는 EPWMxB 출력에 적용될 수 있습니다. AQCTLA 및 AQCTLB 레지스터에서 정의할 수 있는 작업은 다음과 같습니다.

Table 35-29. Counter-Compare A Register (CMPA) Field Descriptions

Bits	Name	Description
15-0	CMPA	<p>The value in the active CMPA register is continuously compared to the time-base counter (TBCTR). When the values are equal, the counter-compare module generates a "time-base counter equal to counter compare A" event. This event is sent to the action-qualifier where it is qualified and converted it into one or more actions. These actions can be applied to either the EPWMxA or the EPWMxB output depending on the configuration of the AQCTLA and AQCTLB registers. The actions that can be defined in the AQCTLA and AQCTLB registers include:</p> <ul style="list-style-type: none"> • Do nothing; the event is ignored. • Clear: Pull the EPWMxA and/or EPWMxB signal low. • Set: Pull the EPWMxA and/or EPWMxB signal high. • Toggle the EPWMxA and/or EPWMxB signal. <p>Shadowing of this register is enabled and disabled by the CMPCTL[SHDWAMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> • If CMPCTL[SHDWAMODE] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the CMPCTL[LOADAMODE] bit field determines which event will load the active register from the shadow register. • Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full. • If CMPCTL[SHDWAMODE] = 1, then the shadow register is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware. • In either mode, the active and shadow registers share the same memory map address.

Do nothing ; 이벤트는 무시됩니다.

- 지우기: EPWMxA 및/또는 EPWMxB 신호를 로우로 당깁니다.
- 설정: EPWMxA 및/또는 EPWMxB 신호를 하이로 당깁니다.
- EPWMxA 및/또는 EPWMxB 신호를 전환합니다.

이 레지스터의 셜도잉은 CMPCTL[SHDWAMODE] 비트에 의해 활성화 및 비활성화됩니다. 기본적으로 이 레지스터는 숨겨져 있습니다.

etPWM(w/UART) – code 및 내용 분석

```
void etpwmInit(void)
```

```
{
```

카운터가 0에 도달하면 강제 EPWMxA 출력을 높이고 카운터가 비교 A 값에 도달하면 로우

```
/* - Force EPWMxA output high when counter reaches zero and low when counter reaches Compare A value */
etpwmREG1->AQCTLA = ((uint16)((uint16)ActionQual_Set << 0U)
| (uint16)((uint16)ActionQual_Clear << 4U));      Set은 '2' Clear는 '1'
```

```
uint16 AQCTLA;
```

```
/*< 0x0014 Action-Qualifier Control Register for Output A (ETPWMxA)*
```

결론은 1000

14h

AQCTLA

Action-Qualifier Control Register for Output A (EPWMxA)

Figure 35-71. Action-Qualifier Output A Control Register (AQCTLA) [offset = 14h]

15	12	11	10	9	8
Reserved		CBD		CBU	
R-0		R/W-0		R/W-0	
7	6	5	4	3	2
CAD	CAU		PRD		ZRO
R/W-0	R/W-0		R/W-0		R/W-0

LEGEND: R/W = Read/Write; R = Read only; -n = value after reset

3-2	PRD	0	Action when the counter equals the period. 주기와 동일시 Action Note: By definition, in count up-down mode when the counter equals period the direction is defined as 0 or counting down. Do nothing (action is disabled). Clear: force EPWMxA output low. Set: force EPWMxA output high. Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.
1-0	ZRO	0	Action when counter equals zero. Note: By definition, in count up-down mode when the counter equals 0 the direction is defined as 1 or counting up. Do nothing (action is disabled). Clear: force EPWMxA output low. Set: force EPWMxA output high. Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.

etPWM(w/UART) – code 및 내용 분석

```
void etpwmInit(void)
{
    /* - Mode setting for Dead Band Module
     *   -Select the input mode for Dead Band Module
     *   -Select the output mode for Dead Band Module
     *   -Select Polarity of the output PWMs
    */
    etpwmREG1->DBCTL = ((uint16)((uint16)0U << 5U) /* Source for Falling edge delay(0-PWMA, 1-PWMB) */
                           | (uint16)((uint16)0U << 4U) /* Source for Rising edge delay(0-PWMA, 1-PWMB) */
                           | (uint16)((uint16)0U << 3U) /* Enable/Disable EPWMxB invert */
                           | (uint16)((uint16)0U << 2U) /* Enable/Disable EPWMxA invert */
                           | (uint16)((uint16)0U << 1U) /* Enable/Disable Rising Edge Delay */
                           | (uint16)((uint16)0U << 0U)); /* Enable/Disable Falling Edge Delay */

    1Ch          DBCTL          Dead-Band Generator Control Register
    uint16 DBCTL;           /*< 0x001C Dead-Band Generator Control Register*/
}
```

Dead-Band Generator Submodule Registers

Dead-Band Generator Control Register

Section 35.4.4.1

각 설정을 모두 마치면 0111 h

35.4.4.1 Dead-Band Generator Control Register (DBCTL)

Figure 35-75. Dead-Band Generator Control Register (DBCTL) [offset = 1Ch]

15	14	Reserved	8
HALFCYCLE	R/W-0	R-0	
7	6	5	4
Reserved	IN_MODE	POLSEL	OUT_MODE
R-0	R/W-0	R/W-0	R/W-0
LEGEND: R/W = Read/Write; R = Read only; -n = value after reset		0	1
		1	1

etPWM(w/UART) – code 및 내용 분석

35.4.4.1 Dead-Band Generator Control Register (DBCTL)

Figure 35-75. Dead-Band Generator Control Register (DBCTL) [offset = 1Ch]

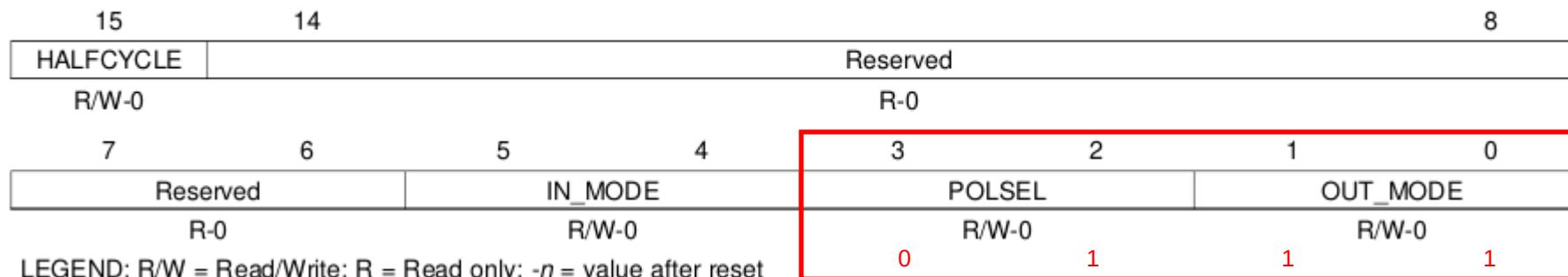


Table 35-35. Dead-Band Generator Control Register (DBCTL) Field Descriptions (continued)

Bits	Name	Value	Description
1-0	OUT_MODE		Dead-band Output Mode Control. Bit 1 controls the S1 switch and bit 0 controls the S0 switch shown in Figure 35-28 . This allows you to selectively enable or bypass the dead-band generation for the falling-edge and rising-edge delay.
0		0	Dead-band generation is bypassed for both output signals. In this mode, both the EPWMxA and EPWMxB output signals from the action-qualifier are passed directly to the PWM-chopper submodule. In this mode, the POLSEL and IN_MODE bits have no effect.
1h		1h	Disable rising-edge delay. The EPWMxA signal from the action-qualifier is passed straight through to the EPWMxA input of the PWM-chopper submodule. The falling-edge delayed signal is seen on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].
2h		2h	The rising-edge delayed signal is seen on output EPWMxA. The input signal for the delay is determined by DBCTL[IN_MODE]. Disable falling-edge delay. The EPWMxB signal from the action-qualifier is passed straight through to the EPWMxB input of the PWM-chopper submodule.
3h		3h	Dead-band is fully enabled for both rising-edge delay on output EPWMxA and falling-edge delay on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].

```
/** - Set the rising edge delay */
etpwmREG1->DBRED = 110U;

/** - Set the falling edge delay */
etpwmREG1->DBFED = 110U ;
```

→ 각 Delay time 설정

3-2	POLSEL		Polarity Select Control. Bit 3 controls the S3 switch and bit 2 controls the S2 switch shown in Figure 35-28 . This allows you to selectively invert one of the delayed signals before it is sent out of the dead-band submodule. The following descriptions correspond to classical upper/lower switch control as found in one leg of a digital motor control inverter. These assume that DBCTL[OUT_MODE] = 1,1 and DBCTL[IN_MODE] = 0,0. Other enhanced modes are also possible, but not regarded as typical usage modes.
		0	Active high (AH) mode. Neither EPWMxA nor EPWMxB is inverted (default).
		1h	Active low complementary (ALC) mode. EPWMxA is inverted.
		2h	Active high complementary (AHC). EPWMxB is inverted.
		3h	Active low (AL) mode. Both EPWMxA and EPWMxB are inverted.

Dead-band 라는게 상승 엣지와 하강 엣지에 각각 enable

위상 inverted 을 했다는 의미

(질문) Dead-band 가 뭘까요...? PWM 파형 상승 or 하강 엣지 동작 전 delay?

etPWM(w/UART) – code 및 내용 분석



```
for(;;)
{
    tmp = sciReceiveByte(UART);

    receive_data = tmp - 48;      // uart 값은 아스키 이므로 48을 빼서 숫자화

    sprintf(buf, "recv_data = %d\n\r\0", receive_data);
    buf_len = strlen(buf);
    sci_display_text(UART, (uint8 *)buf, buf_len);

    gioToggleBit(gioPORTA, 4);
    set_pwm();
```

```
sprintf(buf, "PWM Duty = %d \n\r\0", pwm_value);
buf_len = strlen(buf);
sci_display_text(UART, (uint8 *)buf, buf_len);

wait(5000000);
}
```

```
sprintf(buf, "Press Number!\n\r\0");
buf_len = strlen(buf);
sci_display_text(UART, (uint8 *)buf, buf_len);
```

sci_display_text에서 입력한 UART 를
SciReceiveByte 의 인자로 받아서 tmp에 담는다.
이 tmp의 값은 아스키 문자 값으로 받은 것이므로 -48을 해서 숫자 데이터화 해주어야 한다.

그렇게 숫자 데이터화 된 값을 buf에 담고
Uart display 해준다.

GioToggleBit는 해당 Port의 bit 값을 toggle 시켜 준다.

```
uint32 pwm_duty[] = { 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500 };
```

```
void set_pwm(void)
{
    pwm_value = pwm_duty[receive_data];
    etpwmSetCmpA(etpwmREG1, pwm_value);
    wait(100000);
}
```

```
void etpwmSetCmpA(etpwmBASE_t *etpwm, uint16 value)
{
    etpwm->CMPA = value;    Value 값 도착시 interrupt
}
```

Receive_data를 0~8의 값을 받을 건데,
그 배열의 값이 pwm_value로 들어가게 되고

EtpwmSetCmpA를 통해 compare 방식 etpwm 동작을 하게 한다.

etPWM(w/UART) – code 및 내용 분석

```
HL system.h
/dev/ttyUSB0 - PuTTY

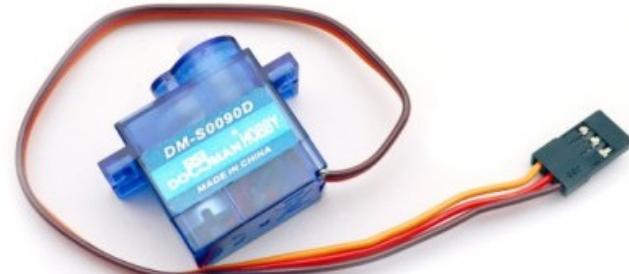
PWM Duty = 2000
recv_data = 4
PWM Duty = 1500
recv_data = 5
PWM Duty = 1750
recv_data = 3
PWM Duty = 1250
recv_data = 4
PWM Duty = 1500
recv_data = 4
PWM Duty = 1500
recv_data = 1
PWM Duty = 750
recv_data = 1
PWM Duty = 750
recv_data = 1
PWM Duty = 750
recv_data = 3
PWM Duty = 1250
recv_data = 1
PWM Duty = 750
recv_data = 3
PWM Duty = 1250
```

오실로스코프 파형

주기 18ms
Duty 값 확인

(주기 20ms로 설정하긴 했었는데,
18Ms 가 측정되는 이유는 모르겠다.)

t8 좌측은 결과 창이며
입력하는 숫자에 따라
Duty 비를 다르게 되었고,
우측과 같은 서버모터 제어시 활용 되었다.



선생님 피드백

태인님:

etPWM 모듈에 대해 나름대로 잘 정리하셨습니다.

아쉬운 부분이라면 스코프 사진이 etPWM 파형 사진이 아니라 eCAP 사진을 올리신것 같습니다.

RC 계열에서 모터를 제어하는데 활용하는 파형은 변함없이 20ms 주기를 맞춰주셔야 합니다.

또 아쉬운 부분이라면 데이터시트 해석 형태가 된 것 같은 부분입니다.

[앞서 있던 파형 스크린 샷은 eCAP에서 활용 된 파형이고,

선생님의 말씀대로

20ms 가 나오게 된 사유를 아는게 중요 하므로,

선생님께서 잘 정리 해 주신 내용을 복기해 보자.]

<https://cafe.naver.com/eddicorp/326>



```
/* - Sets time-base clock prescale bits */
etpwmREG1->TBCTL = (uint16)5U << 7U;

/** - Sets time-base clock prescale bits */
etpwmREG1->TBCTL |= (uint16)((uint16)0U << 10U);

/** - Sets time period or frequency for ETPWM block both PWMA and PWMB */
etpwmREG1->TBPRD = 19999U;

/** - Setup the duty cycle for PWMA */
etpwmREG1->CMPA = 1000U;

/** - Setup the duty cycle for PWMB */
etpwmREG1->CMPB = 10000U;

/** - Force EPWMxA output high when counter reaches zero and low when it reaches maximum value */
etpwmREG1->AQCTLA = ((uint16)((uint16)ActionQual_Set << 0U)
| (uint16)((uint16)ActionQual_Clear << 4U));

/** - Force EPWMxR output high when counter reaches zero and low when it reaches maximum value */
etpwmREG1->AQCTLR = ((uint16)((uint16)ActionQual_Set << 0U)
| (uint16)((uint16)ActionQual_Clear << 4U));
```

[링크램 컬럼] etPWM 동작 분석 요약 (20ms 와 Duty 제어)

안녕하세요. 에디로봇아카데미의 링크램입니다. 이번에는 etPWM을 구동시킨 예에 대해 간략한 요약을 드립니다.

위에서 정리한 사항을 보시면 결국 20 ms 가 어디서 결정되는지 보실 수 있습니다.

그래도 데이터시트 정리를 나름대로 빡세게 잘 해주셨네요.

고생하셨습니다.

etPWM(w/UART)

이번에는 etPWM을 구동시킨 예에 대해 간략한 요약을 해보도록 하겠습니다.

분석에 들어가기 이전에 결론을 먼저 정리해보겠습니다.

실질적으로 우리가 다루는 etPWM은 PMSM이나 ACIM 제어가 없으므로 TBCTL과 CMPA, 그리고 etpwmStartTBCLK()가 다 처리한다고 볼 수 있습니다.

TBCLK은 1 MHz 이므로 여기에 19999를 설정한다는 것은 결론적으로 $1 \times 10^{-6} * 2 * 10^4$ 을 의미합니다.

이것은 $2 * 10^{-2}$ 으로 20 ms가 됩니다.

CMPA가 1000 이므로 20000의 5%이며 이것은 1 ms가 됩니다.

RC에서 제어하는 내용들은 0.5 ms ~ 2.5 ms로 전부 처리된다는 부분을 상기하도록 합시다!

실제 etpwmInit() 부분에 대한 분석부터 코드상에 존재했던 핵심적인 부분들을 빠르게 정리해보겠습니다.

TBCLK 1 MHz 이므로 여기에 19999 설정 한다.

$1 \times 10^{-6} * 2 * 10^4$

이것은 $2 * 10^{-2}$ sec 으로 20 ms 가 된다.

CMPA가 1000 이면 20000 의 5% 가 되고, 이것은 1 ms 가 된다.

RC에서 제어는 0.5 ms ~ 2.5 ms 로 전부 처리 된다는 것을 기억.

Figure 35-64. Time-Base Control Register (TBCTL) [offset = 02h]							
15	14	13	12	10	9	8	
FREE	SOFT	PHSDIR		CLKDIV		HSPCLKDIV	
R/W-0	R/W-0	R/W-0		R/W-0		R/W-0	
7	6	5	4	3	2	1	0
HSPCLKDIV	SWFSYNC	SYNCOSEL		PRLDL	PHSEN		CTRMODE
R/W-1	R/W-0	R/W-0		R/W-0	R/W-0		R/W-3h

LEGEND: R/W = Read/Write; -n = value after reset

Table 35-29. Counter-Compare A Register (CMPA) Field Descriptions

Bits	Name	Description
15-0	CMPA	<p>The value in the active CMPA register is continuously compared to the time-base counter (TBCTR). When the values are equal, the counter-compare module generates a "time-base counter equal to counter compare A" event. This event is sent to the action-qualifier where it is qualified and converted into one or more actions. These actions can be applied to either the EPWMxA or the EPWMxB output depending on the configuration of the AQCTLA and AQCTLB registers. The actions that can be defined in the AQCTLA and AQCTLB registers include:</p> <ul style="list-style-type: none">Do nothing: the event is ignored.Clear: Pull the EPWMxA and/or EPWMxB signal low.Set: Pull the EPWMxA and/or EPWMxB signal high.Toggle the EPWMxA and/or EPWMxB signal. <p>Shadowing of this register is enabled and disabled by the CMPCTL[SHDWAMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none">If CMPCTL[SHDWAMODE] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the CMPCTL[LOADAMODE] bit field determines which event will load the active register from the shadow register.Before a write, the CMPCTL[SHDWAFULL] bit can be read to determine if the shadow register is currently full.If CMPCTL[SHDWAMODE] = 1, then the shadow register is disabled and any write or read will go directly to the active register, that is the register actively controlling the hardware.In either mode, the active and shadow registers share the same memory map address.

```
/** - Sets time period or frequency for ETPWM block both PWMA and PWMB */
etpwmREG1->TBPRD = 19999U;

/** - Setup the duty cycle for PWMA */
etpwmREG1->CMPA = 1000U;
```

etPWM(w/UART)

실제 etpwmInit() 부분에 대한 분석부터 코드상에 존재했던 핵심적인 부분들을 빠르게 정리해보겠습니다.

```
/** @b initialize @b ETPWM1 */

/** - Sets high speed time-base clock prescale bits */
etpwmREG1->TBCTL = (uint16)5U << 7U;

/** - Sets time-base clock prescale bits */
etpwmREG1->TBCTL |= (uint16)((uint16)0U << 10U);

/** - Sets time period or frequency for ETPWM block both PWMA and PWMB */
etpwmREG1->TBPRD = 19999U;

/** - Setup the duty cycle for PWMA */
etpwmREG1->CMPA = 1000U;

/** - Setup the duty cycle for PWMB */
etpwmREG1->CMPB = 10000U;

/** - Force EPWMA output high when counter reaches zero and low */
etpwmREG1->AQCTLA = ((uint16)((uint16)ActionQual_Set << 0U)
                      | (uint16)((uint16)ActionQual_Clear << 4U));

/** - Force EPWMB output high when counter reaches zero and low */
etpwmREG1->AQCTLB = ((uint16)((uint16)ActionQual_Set << 0U)
                      | (uint16)((uint16)ActionQual_Clear << 8U));
```

먼저 TB, CC, AQ 모듈에 대한 부분입니다.

```
etpwmREG1->TBCTL = (uint16)5U << 7U;
```

Time-Base Control Register

고속 시간 기반 클록 프리스케일 비트.

이 비트는 시간 기반 클록 프리스케일 값의 일부를 결정합니다.

(구체적으로 이 비트는 HSPCLKDIV를 결정함)

TBCLK = VCLK3 / (HSPCLKDIV × CLKDIV)

CLKDIV는 디폴트로 1이며 HSPCLKDIV는 10이므로 VCLK3를 10으로 나누는 작업이 됨

실제 PLL 설정하는 부분에서 VCLK3를 10MHz로 맞췄으므로

TBCLK은 1MHz가 되었음을 알 수 있습니다.

/** - Sets time-base clock prescale bits */

```
etpwmREG1->TBCTL |= (uint16)((uint16)0U << 10U);
```

시간 기반 클록 프리스케일 비트.

이 비트는 시간 기반 클록 프리스케일 값의 일부를 결정합니다.

TBCLK = VCLK3 / (HSPCLKDIV × CLKDIV)

구체적으로 CLKDIV 값을 결정하는데 0 비트가 셋되므로 1입니다.

/** - Sets time period or frequency for ETPWM block both PWMA and PWMB */

```
etpwmREG1->TBPRD = 19999U;
```

Time-Base Period Register

이 비트는 시간축 카운터의 주기를 결정합니다.

PWM 주파수를 설정합니다.

이 레지스터의 새도잉은 TBCTL[PRDLD] 비트에 의해 활성화 및 비활성화됩니다.

기본적으로 이 레지스터는 숨겨져 있습니다.

etPWM(w/UART)

* TBCTL[PRDLD] = 0이면 샘플링은 활성화되고 쓰기 또는 읽기가 자동으로 샘플링 레지스터로 이동합니다.

이 경우 활성 레지스터는 타임베이스 카운터가 0일 때 샘플링 레지스터에서 로드됩니다.

* TBCTL[PRDLD] = 1이면 샘플링은 비활성화되고 모든 쓰기 또는 읽기가 활성 레지스터,
즉 하드웨어를 능동적으로 제어하는 레지스터로 직접 이동합니다.

* 활성 및 샘플링 레지스터는 동일한 메모리 맵 주소를 공유합니다.

우리의 케이스에서 3번 비트가 0이므로 Shadow가 활성화됩니다.

Shadow 활성화의 의미??

```
/** - Setup the duty cycle for PWMA */
```

```
etpwmREG1->CMPA = 1000U;
```

Counter-Compare A Register

활성 CMPA 레지스터의 값은 TBCTR(타임 베이스 카운터)과 지속적으로 비교됩니다.

값이 같을 때 카운터 비교 모듈은 "시간 기반 카운터가 카운터 비교 A와 같음" 이벤트를 생성합니다.

이 이벤트는 자격이 부여되고 하나 이상의 작업으로 변환되는 action-qualifier로 전송됩니다.

이러한 작업은 AQCTLA 및 AQCTLB 레지스터의 구성에 따라

EPWMxA 또는 EPWMxB 출력에 적용될 수 있습니다.

AQCTLA 및 AQCTLB 레지스터에서 정의할 수 있는 작업은 다음과 같습니다.

* 아무것도하지 마세요; 이벤트는 무시됩니다.

* 지우기: EPWMxA 및/또는 EPWMxB 신호를 로우로 당깁니다.

* 설정: EPWMxA 및/또는 EPWMxB 신호를 하이로 당깁니다.

* EPWMxA 및/또는 EPWMxB 신호를 전환합니다.

이 레지스터의 샘플링은 CMPCTL[SHDWAMODE] 비트에 의해 활성화 및 비활성화됩니다.

* CMPCTL[SHDWAMODE] = 0이면 샘플링은 활성화되고 쓰기 또는 읽기가 자동으로 샘플링 레지스터로 이동합니다.

이 경우 CMPCTL[LOADAMODE] 비트 필드는 샘플링 레지스터에서 활성 레지스터를 로드할 이벤트를 결정합니다.

* 쓰기 전에 CMPCTL[SHDWAFULL] 비트를 읽어서 샘플링 레지스터가 현재 가득 찬지 확인할 수 있습니다.

* CMPCTL[SHDWAMODE] = 1이면 샘플링 레지스터가 비활성화되고 쓰기

또는 읽기가 하드웨어를 능동적으로 제어하는 레지스터인 활성 레지스터로 직접 이동합니다.

* 두 모드에서 활성 및 샘플링 레지스터는 동일한 메모리 맵 주소를 공유합니다.

결국 이 부분은 AQ에 의해 어떻게 처리될지 결정됩니다.

```
/** - Setup the duty cycle for PWMB */
```

```
etpwmREG1->CMPB = 10000U;
```

A상과 B상의 값을 10배
차이 나게 설정함

실질적으로 A상과 B상의 차이는 10배가 나므로 비교하는 수치값에서도 10배가 차이나는 것을 볼 수 있습니다.

```
/** - Force EPWMxA output high when counter reaches zero and low when counter reaches Compare A value */
```

```
etpwmREG1->AQCTLA = ((uint16)((uint16)ActionQual_Set << 0U)
```

```
| (uint16)((uint16)ActionQual_Clear << 4U));
```

Action-Qualifier Output A Control Register

ActionQual_Set = 2

ActionQual_Clear = 1

즉 1번 비트가 1인 경우와 4번 비트가 1인 경우를 살펴봐야 합니다.

주석에도 적혀 있듯이 0을 만나면 HIGH로 뜨고 CMPA를 만나면 LOW로 가라앉습니다.

etPWM(w/UART)

```
/** - Mode setting for Dead Band Module
 *  -Select the input mode for Dead Band Module
 *  -Select the output mode for Dead Band Module
 *  -Select Polarity of the output PWMs
 */
etpwmREG1->DBCTL = ((uint16)((uint16)0U << 5U)      /
| (uint16)((uint16)0U << 4U)      /
| (uint16)((uint16)0U << 3U)      /
| (uint16)((uint16)0U << 2U)      /
| (uint16)((uint16)0U << 1U)      /
| (uint16)((uint16)0U << 0U));  /

/** - Set the rising edge delay */
etpwmREG1->DBRED = 110U;

/** - Set the falling edge delay */
etpwmREG1->DBFED = 110U;
```

극성 선택 제어.

비트 3은 위 그림의 S3 스위치를 제어하고 비트 2는 S2 스위치를 제어합니다.

이를 통해 지연된 신호 중 하나를 데드밴드 서브모듈에서 보내기 전에 선택적으로 반전할 수 있습니다.

다음 설명은 디지털 모터 제어 인버터의 한 다리에서 볼 수 있는 고전적인 상부/하부 스위치 제어에 해당합니다.

이들은 DBCTL[OUT_MODE] = 1,1 및 DBCTL[IN_MODE] = 0,0이라고 가정합니다.

다른 고급 모드도 가능하지만 일반적인 사용 모드로 간주되지 않습니다.

액티브 하이(AH) 모드.

EPWMxA 또는 EPWMxB는 반전되지 않습니다(기본값).

Dead-Band 출력 모드 제어.

비트 1은 위 그림과 같이 S1 스위치를 제어하고 비트 0은 S0 스위치를 제어합니다.

이를 통해 하강 에지 및 상승 에지 지연에 대한

Dead-Band 생성을 선택적으로 활성화하거나 우회할 수 있습니다.

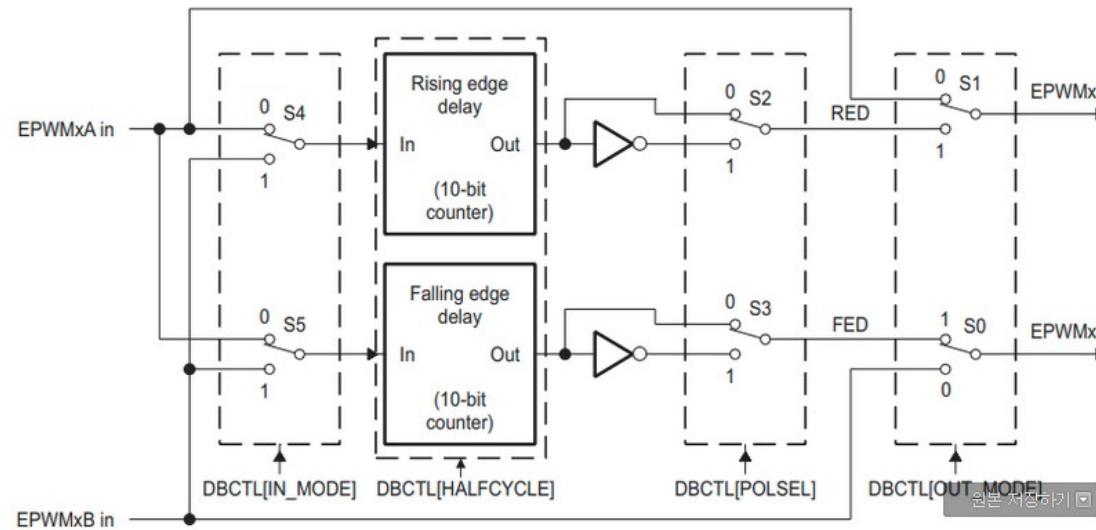
의미를 잘 모르겠다..??

Dead-Band 생성은 두 출력 신호 모두에 대해 바이패스됩니다.

이 모드에서 동작 한정자의 EPWMxA 및 EPWMxB 출력 신호는 PWM 초퍼 서브모듈로 직접 전달됩니다.

Dead-Band 입력 모드 제어.

비트 5는 아래 그림과 같이 S5 스위치를 제어하고 비트 4는 S4 스위치를 제어합니다.



이를 통해 하강 에지 및 상승 에지 지연에 대한 입력 소스를 선택할 수 있습니다.

고전적인 Dead-Band 파형을 생성하려면 기본값은 EPWMxA입니다.

입력은 하강 및 상승 에지 지연의 소스입니다.

EPWMxA In(액션 한정자에서)은 하강 에지 및 상승 에지 지연의 소스입니다.

```
/** - Set the rising edge delay */
etpwmREG1->DBRED = 110U;
```

Dead-Band Generator Rising Edge Delay Register

상승 에지 지연 수.

10비트 카운터.

```
/** - Set the falling edge delay */
etpwmREG1->DBFED = 110U;
```

Dead-Band Generator Falling Edge Delay Register

하강 에지 지연 횟수.

10비트 카운터.

하강 에지, 상승 에지 부분은 부가적 요소이며 실질적으로 etPWMA 신호가 발생될 것임을 알 수 있습니다.

etPWM(w/UART)

PWM-Chopper Control Register

원샷 펄스 폭

1 x VCLK3 / 8 wide (= 80 nS at 100 MHz VCLK3)

실제로 우리는 VCLK를 10 MHz로 떨굼

초평 클럭 뉴티 사이클

Duty = 4/8 (50.0%)

초평 클럭 주파수.

1로 나누기(사전 스케일 없음, = 100MHz VCLK3에서 12.5MHz)

Chopper 부분에 대한 해석은 잘 이해가 안되지만 중요한 요소를 차지 하진 않는다고 하니 일단 넘어 가야 할 것 같다..

Trip-Zone Select Register

ePWM 모듈에 대한 원샷 트립 소스로 TZn를 비활성화합니다.

전반적으로 이 부분은 현재 해석에서 중요한 요소를 차지하지 않습니다.

왜냐하면 Chopper는 ACIM이나 PMSM같은 3상 모터에 특수한 정밀 제어를 하는 경우 많이 활용하기 때문입니다.

실제로 코드에서도 초퍼 기능을 꺼놓은 것을 볼 수 있습니다.

```
/** - Set interrupt enable */
etpwmREG1->TZEINT = 0x0000U      /** - Enable/Disable Di
| 0x0000U      /** - Enable/Disable one
| 0x0000U;     /** - Enable/Disable cyc
```

```
/** - Sets up the event for interrupt */
etpwmREG1->ETSEL = (uint16)NO_EVENT;
```

```
if ((etpwmREG1->ETSEL & 0x0007U) != 0U)
{
    etpwmREG1->ETSEL |= 0x0008U;
}
```

```
/** - Setup the frequency of the interrupt generation */
etpwmREG1->ETPS = 1U;
```

```
/** - Sets up the ADC SOC interrupt */
etpwmREG1->ETSEL |= ((uint16)(0x0000U)
| (uint16)(0x0000U)
| (uint16)((uint16)DCAEVT1 << 8U)
| (uint16)((uint16)DCBEVT1 << 12U));
```

```
/** - Sets up the ADC SOC period */
etpwmREG1->ETPS |= ((uint16)((uint16)1U << 8U)
| (uint16)((uint16)1U << 12U));
```

원본 저작하기

etPWM(w/UART)

이제 이 부분 코드에서는 결국 etpwmStartTBCLK() 부분에 의미가 강하게 부여됩니다.

```
172     etpwmStartTBCLK();  
173 /* USER CODE BEGIN (2) */  
175 /* USER CODE END */  
176 }  
177  
178 /** @fn void etpwmStartTBCLK()  
179 *  @brief Start the time-base clocks of all eTPWMx modules  
180 *  
181 *  This function starts the time-base clocks of all eTPWMx modules.  
182 */  
183 /* SourceId : ETPWM_SourceId_002 */  
184 /* DesignId : ETPWM_DesignId_002 */  
185 /* Requirements : HL_CONQ_EPWM_SR45 */  
186 void etpwmStartTBCLK(void)  
187 {  
188     /* Enable Pin Muxing */  
189     pinMuxReg->KICKER0 = 0x83E70B13U;  
190     pinMuxReg->KICKER1 = 0x95A4F1E0U;  
191  
192     pinMuxReg->PINMUX[166U] = (pinMuxReg->PINMUX[166U] & PINMUX_ETPWM_TBCLK_SYNC_MASK)  
193         | (PINMUX_ETPWM_TBCLK_SYNC_ON);  
194  
195     /* Disable Pin Muxing */  
196     pinMuxReg->KICKER0 = 0x00000000U;  
197     pinMuxReg->KICKER1 = 0x00000000U;  
198 }
```

키커 0 레지스터.

값 83E7 0B13h는 PINMMRnn 레지스터에 대한

CPU 쓰기 액세스를 잠금 해제하는 프로세스의 일부로 KICK0에 기록되어야 합니다.

키커 1 레지스터.

값 95A4 F1E0h는 PINMMRnn 레지스터에 대한

CPU 쓰기 액세스를 잠금 해제하는 프로세스의 일부로 KICK1에 기록되어야 합니다.

PINMUX_ETPWM_TBCLK_SYNC_MASK = 0xFFFFFFF00

PINMUX_ETPWM_TBCLK_SYNC_ON = 2

이 레지스터는 마이크로컨트롤러의 특수 기능을 제어합니다.

PINMMR160에서 PINMMR179까지 20개의 이러한 레지스터가 있습니다.

PINMMR 레지스터의 각 8비트 필드는 하나의 특수 기능을 제어합니다.

PINMMRx 제어 레지스터와 주어진 터미널에서 선택된 기능 간의 매핑은 표 6-3에 정의되어 있습니다.

ePWMx TBCLKSYNC Enable에 & 0xFFFFFFF00을 하고 2를 OR 연산함

즉 TBCLKSYNC가 1로 활성화됨을 의미합니다.

결론적으로 etPWM 신호가 활성화되는 것을 의미합니다.

etPWM(w/UART)

* Control for Synchronizing Time Bases for All ePWMx Modules

ePWMx 모듈은 시간 기반을 동기화할 수 있는 메커니즘을 구현합니다.

이것은 모든 ePWMx 모듈에 대한 공통 입력인 TBCLKSYNC라는 신호를 사용하여 수행됩니다.

이 TBCLKSYNC는 I/O 다중화 모듈의 레지스터 비트에 의해 생성됩니다.

PINMMR166[1]은 TBCLKSYNC 신호입니다.

이 비트는 기본적으로 지워집니다(0).

TBCLKSYNC = 0일 때 모든 ePWMx 모듈의 시간축 클록이 중지됩니다. 이것이 기본 조건입니다.

TBCLKSYNC = 1일 때 모든 ePWMx 모듈의 시간 기반 클록은
TBCLKSYNC 신호의 상승 에지에 맞춰 시작됩니다.

모든 ePWMx 모듈의 시간 기반 클록을 활성화하고 동기화하는 올바른 절차는 다음과 같습니다.

1. 비활성화된 경우 원하는 개별 ePWMx 모듈에 대한 클럭을 활성화합니다.
2. TBCLKSYNC = 0으로 설정합니다. 이렇게 하면 활성화된 ePWMx 모듈의 시간 기반 클록이 중지됩니다.
3. 시간 기반 클록 프리스케일러 값과 원하는 ePWM 모드를 구성합니다.
4. TBCLKSYNC = 1로 설정합니다.

- 시간 축 클록을 사용하는
경우는 어떤 상황이 있는지
잘 모르겠습니다..

```

433 /** @fn void etpwmSetCmpA(etpwmBASE_t *etpwm, uint16 value)
434 * @brief Set the Compare A value
435 *
436 * @param etpwm      The pulse width modulation (ETPWM) object
437 * @param value      16-bit Compare A value
438 *
439 * This function sets the compare A value
440 */
441 /* SourceId : ETPWM_SourceId_015 */
442 /* DesignId : ETPWM_DesignId_015 */
443 /* Requirements : HL_CONQ_EPWM_SR14 */
444 void etpwmSetCmpA(etpwmBASE_t *etpwm, uint16 value)
445 {
446     etpwm->CMPA = value;
447 }
...

```

이 API는 CMPA를 조정하므로 결국 Pulse Width를 조정하는데 활용한다고 볼 수 있습니다.
실질적으로 종합을 해보자면 결국 TBCTL와 CMPA, 그리고 etpwmStartTBCLK()가 다 했다고 볼 수 있습니다.

TBCLK는 1 MHz 이므로 여기에 19999를 설정한다는 것은 결론적으로
 $1 \times 10^{-6} \times 2 \times 10^4$ 을 의미합니다.
이것은 2×10^{-2} 으로 20 ms가 됩니다.
CMPA가 1000 이므로 20000의 5%이며 이것은 1 ms가 됩니다.
RC에서 제어하는 내용들은 0.5 ms ~ 2.5 ms로 전부 처리된다는 부분을 상기하도록 합시다!

요게 제일 중요 포인트!