

전자융합설계2 실험 보고서

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: <http://www.ieee.org>)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임:** 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제:** 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성:** 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지:** 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해:** 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성:** 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세:** 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기:** 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.
9. **도덕성:** 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애:** 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

학 부: 전자공학부

제출일: 2020.06.24.(수)

과목명: 융합캡스톤디자인2

교수명: 이교범 교수님

조 : 4 조

학 번: 201420820, 201623432

성 명: 이승복, 김희서

[기말 프로젝트]

1. 코드 분석

<코드 1. 400줄>

```
#include "DSP28x_Project.h"      /* Device Headerfile and Examples Include File */

#define SYSTEM_CLOCK      150E6  /* 150MHz */
#define TBCLK              150E6  /* 150MHz */
#define PWM_CARRIER      20E3   /* 20kHz */
```

- **SYSTEM_CLOCK** : 시스템 클럭을 150MHz로 설정한다. 이때의 주기는 $1/150\text{MHz}=0.0067\mu\text{s}$ 이다.
- **TBCLK** : 시스템 클럭의 사전 정의 된 버전(prescaled version of the system clock)으로, 시스템 클럭 내의 모든 하위 항목에 사용된다. 카운터 주파수 결정 변수이며, 150MHz로 설정한다.
- **PWM_CARRIER** : PWM 삼각파의 주파수는 20kHz로 설정한다.

```
/* Prototype statements for functions found within this Example */
void InitEPwm5Module(void);
interrupt void EPwm5Isr(void);
interrupt void adc_isr(void);          // ADC 인터럽트 함수 선언

/* Global variables used in this Example */
Uint16 BackTicker;
Uint16 EPwm5IsrTicker;
```

- 프로젝트는 EPWM 5A, 5B만 사용하기 때문에 그에 맞는 함수와 PWM 인터럽트, ADC 인터럽트에 대한 함수를 선언한다.

```
interrupt void Xint3_isr(void);
interrupt void Xint4_isr(void);
interrupt void Xint5_isr(void);
interrupt void Xint6_isr(void);

float32 PwmCarrierFrequency;
float32 PwmDutyRatio;
float32 FallingEdgeDelay1;
float32 RisingEdgeDelay1;
float32 PWM_CARRIER2, PWM_CARRIER1;

Uint16 ADC_value01, ADC_value02, deadt = 0;
Uint16 Loop_cnt, ADC_cnt;
Uint16 SW1_cnt, SW2_cnt, SW3_cnt, SW4_cnt;
```

- 프로젝트에서 사용할 전역변수에 대한 설정을 한다.
- **FallingEdgeDelay1, RisingEdgeDelay1**
: Dead band 모듈에서 사용되는 상승, 하강 지연시간을 나타내는 변수이다.
- **PWM_CARRIER1, PWM_CARRIER2**
: 가변저항 A, B를 사용하는 PWM 주파수 가변 실험에서 사용할 PWM 주파수변수이다.
- **ADC_value01, ADC_value02**
: 가변저항 A, B의 변화에 대한 전압 변동의 상대적인 디지털 값을 나타내는 변수이다.

```

/*-----
Step 2
2.1 InitSysCtrl()
2.1.1 Disables the watchdog
2.1.2 Set the PLLCR for proper SYSCLKOUT frequency
2.1.3 Set the pre-scaler for the high and low frequency peripheral clocks
2.1.4 Enable the clocks to the peripherals
2.2 Initialize GPIO MUX
-----*/

InitSysCtrl();

EALLOW;
GpioCtrlRegs.GPAPUD.bit.GPIO8 = 0; /* Enable pull-up on GPIO8 (EPWM4A) */
GpioCtrlRegs.GPAPUD.bit.GPIO9 = 0; /* Enable pull-up on GPIO9 (EPWM4B) */
GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1; /* Configure GPIO8 as EPWM4A */
GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1; /* Configure GPIO9 as EPWM4B */
SysCtrlRegs.HSPCLK.bit.HSPCLK = 1; // HSPCLK = SYSCLKOUT/(HSPCLK*2)
EDIS;

```

- **DINT**: 칩의 활성 여부를 결정하는 전역 인터럽트 스위치를 Off하고, 칩의 비정상적인 작동을 방지한다.
- **IER = 0x0000, IFR = 0x0000**: 인터럽트를 사용하기 위한 준비단계로, 레지스터의 상태를 0으로 설정하여 혹시 모를 시스템 오동작을 방지한다.
- **InitSysCtrl()** : 시스템 컨트롤을 초기화한다. 이 함수가 하는 역할은 다음과 같다.
 - (1) 시스템에 이상이 생기면 시스템을 리셋하는 Watchdog기능을 disable한다.
 - (2) PLLCR 레지스터를 설정하여 적당한 SYSCLKOUT을 생성한다.
 - (3) 프리스케일러를 설정하여 고속 주변장치 클럭과 저속 주변장치 클럭을 조정한다.
 - (4) 클럭을 ON하여 주변장치에 공급한다.
- **GpioCtrlRegs.GPAPUD.bit.GPIO8, 9 = 0** : : PWM의 출력신호를 사용하기 위해서, GPAPUD에 '0'의 값이 들어오면, 지정된 핀(GPIO8)에서 내부 풀업(Pull up)을 활성화한다.

Table 65. GPIO Port A Internal Pullup Disable (GPAPUD) Register Field Descriptions

| Bits | Field | Value | Description ⁽¹⁾ |
|------|--------------|-------|--|
| 31-0 | GPIO31-GPIO0 | | Configure the internal pullup resistor on the selected GPIO Port A pin. Each GPIO pin corresponds to one bit in this register. |
| | | 0 | Enable the internal pullup on the specified pin. (default for GPIO12-GPIO31) |
| | | 1 | Disable the internal pullup on the specified pin. (default for GPIO0-GPIO11) |

- **GpioCtrlRegs.GPAMUX1.bit.GPIO8, 9 = 1**;
: GPAMUX에 '1'의 값을 입력하여, GPIO8, 9의 출력을 EPWM5A, 5B로 설정한다.

Table 50. GPIO Port A Multiplexing 1 (GPAMUX1) Register Field Descriptions (continued)

| 17-16 | GPIO8 | | Configure the GPIO8 pin as: |
|-------|-------|----|---|
| | | 00 | GPIO8 - General purpose I/O 8 (default) (I/O) |
| | | 01 | EPWM5A - ePWM5 output A (O) |
| | | 10 | CANTXB - eCAN-B transmit (O) |
| | | 11 | ADCSOCA0 - ADC Start of conversion A |

* **HSPCLK**: 이 비트는 SYSCLKOUT을 기준으로 high-speed peripheral 클럭 속도를 설정한다. 해당 코드에서 HSPCLK에 '1'값을 입력해주었기 때문에 위 표로부터 'High speed clock'은 'SYSCLKOUT/2'이 되는 것을 알 수 있다. 해당 시스템에서 SYSCLKOUT이 150MHz이기 때문에 HSPCLK 값은 그 절반인 75MHz가 된다.

Table 18. High-Speed Peripheral Clock Prescaler (HISPCP) Field Descriptions

| Bits | Field | Value | Description ⁽¹⁾ |
|------|----------|-------|--|
| 15-3 | Reserved | | Reserved |
| 2-0 | HSPCLK | | These bits configure the high-speed peripheral clock (HSPCLK) rate relative to SYSCLKOUT: If HISPCP ⁽²⁾ ≠ 0, HSPCLK = SYSCLKOUT/(HISPCP X 2) If HISPCP = 0, HSPCLK = SYSCLKOUT |
| | | 000 | High speed clock = SYSCLKOUT/1 |
| | | 001 | High speed clock = SYSCLKOUT/2 (reset default) |
| | | 010 | High speed clock = SYSCLKOUT/4 |
| | | 011 | High speed clock = SYSCLKOUT/6 |
| | | 100 | High speed clock = SYSCLKOUT/8 |
| | | 101 | High speed clock = SYSCLKOUT/10 |
| | | 110 | High speed clock = SYSCLKOUT/12 |
| | | 111 | High speed clock = SYSCLKOUT/14 |

⁽¹⁾ This register is EALLOW protected. See [Section 7.2](#) for more information.

⁽²⁾ HISPCP in this equation denotes the value of bits 2:0 in the HISPCP register.

- **EALLOW-EDIS**: DSP에서 일정 Protected 영역에 값을 쓰기 위해서는 Protect를 해지하고 다시 Protect를 해주는 작업이 필요하다. GpioCtrlRegs 그룹의 레지스터는 모두 Protected 영역에 있으므로 레지스터를 설정하기 위해서는 EALLOW로 보호설정을 풀어주고, EDIS로 보호를 해주어야 한다.

```

Step 3
3.1 Initialize Peripheral Interrupt Expansion circuit
-----*/
InitPieCtrl();
IER = 0x0000;
IFR = 0x0000;
/*-----

Step 4
4.1 Pie Vector Table Re-allocation
-----*/
InitPieVectTable();

/*-----

Step 5
5.1 Interrupt Service routine re-mapping and Interrupt vector enable
-----*/

/* Interrupt Service Routine Re-mapping */
EALLOW;
PieVectTable.EPwm5_INT = &EPwm5Isr;
PieVectTable.ADCINT = &adc_isr;
EDIS;

EALLOW;
PieVectTable.XINT3 = &Xint3_isr;
PieVectTable.XINT4 = &Xint4_isr;
PieVectTable.XINT5 = &Xint5_isr;
PieVectTable.XINT6 = &Xint6_isr;
EDIS;

```

- **InitPieCtrl()** : PIE회로를 초기화하기 위한 함수이다. 먼저 PIE회로를 disable 시키고, PIEIER과 PIEIFR 레지스터를 clear 시켜 놓는다.

Table 112. PIE Vector Table (continued)

| Name | VECTOR ID ⁽¹⁾ | Address ⁽²⁾ | Size (x16) | Description ⁽³⁾ | CPU Priority | PIE Group Priority |
|--|--------------------------|------------------------|------------|----------------------------|--------------|--------------------|
| INT11.6 | 117 | 0x0000 0DEA | 2 | Reserved | 15 | 6 |
| INT11.7 | 118 | 0x0000 0DEC | 2 | Reserved | 15 | 7 |
| INT11.8 | 119 | 0x0000 0DEE | 2 | Reserved | 15 | 8 (lowest) |
| PIE Group 12 Vectors - Muxed into CPU INT12 | | | | | | |
| INT12.1 | 120 | 0x0000 0DF0 | 2 | XINT3 | 16 | 1 (highest) |
| INT12.2 | 121 | 0x0000 0DF2 | 2 | XINT4 | 16 | 2 |
| INT12.3 | 122 | 0x0000 0DF4 | 2 | XINT5 | 16 | 3 |
| INT12.4 | 123 | 0x0000 0DF6 | 2 | XINT6 | 16 | 4 |
| INT12.5 | 124 | 0x0000 0DF8 | 2 | XINT7 | 16 | 5 |
| INT12.6 | 125 | 0x0000 0DFA | 2 | Reserved | 16 | 6 |
| INT12.7 | 126 | 0x0000 0DFC | 2 | LVF | 16 | 7 |
| INT12.8 | 127 | 0x0000 0DFE | 2 | LUF | 16 | 8 (lowest) |

- **InitPieVectTable()** : PIE회로를 활성화 시키며, 28x DSP에서 인터럽트를 쓰기 위한 기본 단계이다. 제조사인

TI에서 모든 인터럽트 벡터들을 모두 Mapping 시켜놨기 때문에 이 함수를 사용하게 되면 모든 인터럽트 벡터들이 TI에서 정한 default 인터럽트 서비스 루틴에 연결된다. 0 ~ 127의 Vector ID를 가지는 PIE vector의 함수를 사용한다.

Table 111. PIE MUXed Peripheral Interrupt Vector Table

| INTx.7 | INTx.6 | INTx.5 | INTx.4 | INTx.3 |
|--------------------|------------------------|------------------------|------------------------|------------------------|
| TINT0 (TIMER 0) | ADCINT (ADC) | XINT2 Ext. int. 2 | XINT1 Ext. int. 1 | Reserved - |
| 0xD4C | 0xD4A | 0xD48 | 0xD46 | 0xD44 |
| Reserved | EPWM6_TZINT (ePWM6) | EPWM5_TZINT (ePWM5) | EPWM4_TZINT (ePWM4) | EPWM3_TZINT (ePWM3) |
| 0xD5C | 0xD5A | 0xD58 | 0xD56 | 0xD54 |
| Reserved | EPWM6_INT (ePWM6) | EPWM5_INT (ePWM5) | EPWM4_INT (ePWM4) | EPWM3_INT (ePWM3) |
| 0xD6C | 0xD6A | 0xD68 | 0xD66 | 0xD64 |
| Reserved | ECAP6_INT (eCAP6) | ECAP5_INT (eCAP5) | ECAP4_INT (eCAP4) | ECAP3_INT (eCAP3) |
| 0xD7C | 0xD7A | 0xD78 | 0xD76 | 0xD74 |

• PieVectTable.EPWM5_INT = &EPwm5Isr, PieVectTable.ADCINT = &adc_isr

: 각각의 함수(&EPwm5Isr, &adc_isr)값으로 벡터를 Remapping 해준다.

• PieVectTable.XINT3 = &Xint3_isr;

: Default로 설정된 인터럽트 벡터를 Remapping 해준다. 위의 코드는 외부 인터럽트 3이 걸렸을 때 Xint3_isr이라는 ISR 함수를 실행하도록 설정한 코드이다. 이를 외부인터럽트 4, 5, 6에 대해서도 지정하였다.

```
// Step 4. GPIO 초기화
//-----
EALLOW;
GpioCtrlRegs.GPBMUX1.bit.GPI044 = 0; // 핀 기능선택: GPIO44
GpioCtrlRegs.GPBMUX1.bit.GPI045 = 0; // 핀 기능선택: GPIO45
GpioCtrlRegs.GPBMUX1.bit.GPI046 = 0; // 핀 기능선택: GPIO46
GpioCtrlRegs.GPBMUX1.bit.GPI047 = 0; // 핀 기능선택: GPIO47
GpioCtrlRegs.GPBDIR.bit.GPI044 = 0; // GPIO44 입출력 선택: Input
GpioCtrlRegs.GPBDIR.bit.GPI045 = 0; // GPIO45 입출력 선택: Input
GpioCtrlRegs.GPBDIR.bit.GPI046 = 0; // GPIO46 입출력 선택: Input
GpioCtrlRegs.GPBDIR.bit.GPI047 = 0; // GPIO47 입출력 선택: Input
EDIS;
//=====
```

• **GPBMUX**: 핀의 역할을 정해주는 레지스터로, 0을 입력하면 핀을 GPIO로 사용하는 것이고, 1을 입력하면 핀을 GPIO가 아닌 다른 용도로 사용한다는 의미이다. 이번 프로젝트에서는 모든 핀이 GPIO로 사용하기 때문에 0을 입력해주었다.

Table 52. GPIO Port B MUX 1 (GPBMUX1) Register Field Descriptions

| Bit | Field | Value | Description |
|-------|--------|----------------------|---|
| 31:30 | GPI047 | 00 01 10 or 11 | Configure this pin as: GPIO 47 - general purpose I/O 47 (default) Reserved XA7 - External interface (XINTF) address line 7 (O) |
| 29:28 | GPI046 | 00 01 10 or 11 | Configure this pin as: GPIO 46 - general purpose I/O 46 (default) Reserved XA6 - External interface (XINTF) address line 6 (O) |
| 27:26 | GPI045 | 00 01 10 or 11 | Configure this pin as: GPIO 45 - general purpose I/O 45 (default) Reserved XA5 - External interface (XINTF) address line 5 (O) |
| 25:24 | GPI044 | 00 01 10 or 11 | Configure this pin: GPIO 44 - general purpose I/O 44 (default) Reserved XA4 - External interface (XINTF) address line 4 (O) |

- **GPBDIR**: GPIO핀의 입출력을 선택하는 레지스터로, 0을 입력하면 GPIO핀을 입력으로 이용하고, 1을 입력하면 GPIO핀을 출력으로 이용한다는 의미이다. 이번 프로젝트에서 GPIO핀은 스위치를 누르는 입력 역할로 동작하기 때문에 0을 입력해주었다.

Table 63. GPIO Port B Direction (GPBDIR) Register Field Descriptions

| Bits | Field | Value | Description ⁽¹⁾ |
|------|---------------|-------|---|
| 31-0 | GPIO63-GPIO32 | | Controls direction of GPIO pin when GPIO mode is selected. Reading the register returns the current value of the register setting |
| | | 0 | Configures the GPIO pin as an input. (default) |
| | | 1 | Configures the GPIO pin as an output |

```
// Step 5. Qualification 초기화
//-----
EALLOW;
GpioCtrlRegs.GPBCTRL.bit.QUALPRD1 = 0xFF; // (GPIO40~GPIO47) Qual period 설정
GpioCtrlRegs.GPBQSEL1.bit.GPIO44 = 2;    // Qualification using 6 samples
GpioCtrlRegs.GPBQSEL1.bit.GPIO45 = 2;    // Qualification using 6 samples
GpioCtrlRegs.GPBQSEL1.bit.GPIO46 = 2;    // Qualification using 6 samples
GpioCtrlRegs.GPBQSEL1.bit.GPIO47 = 2;    // Qualification using 6 samples
EDIS;
//=====
```

- **GPBCTRL**: 샘플링 주기를 설정 가능하다. 0xFF값을 갖는 경우 샘플링 주기는 $510 \times T_{SYSCLKOUT}$ 값을 갖는다.

Table 57. GPIO Port B Qualification Control (GPBCTRL) Register Field Descriptions

| Bits | Field | Value | Description ⁽¹⁾ |
|------|----------|-------|---|
| 15-8 | QUALPRD1 | | Specifies the sampling period for pins GPIO40 to GPIO47 |
| | | 0x00 | Sampling Period = $T_{SYSCLKOUT}$ ⁽²⁾ |
| | | 0x01 | Sampling Period = $2 \times T_{SYSCLKOUT}$ |
| | | 0x02 | Sampling Period = $4 \times T_{SYSCLKOUT}$ |
| | | ... | ... |
| | | 0xFF | Sampling Period = $510 \times T_{SYSCLKOUT}$ |

- **GPBQSEL**: Input qualification type을 선택한다. 샘플의 사용 횟수를 정하는데, 1값을 입력할 경우 3개의 샘플을 이용하고 2값을 입력할 경우 6개의 샘플을 이용한다. 이번 프로젝트에서는 GPIO44~47 모두 2값을 입력했기 때문에 6개의 샘플을 qualification에 이용한다.

Table 60. GPIO Port B Qualification Select 1 (GPBQSEL1) Register Field Descriptions

| Bits | Field | Value | Description ⁽¹⁾ |
|------|---------------|-------|---|
| 31-0 | GPIO47-GPIO32 | | Select input qualification type for GPIO32 to GPIO47. The input qualification of each GPIO input is controlled by two bits as shown in Figure 55. |
| | | 00 | Synchronize to SYSCLKOUT only. Valid for both peripheral and GPIO pins. |
| | | 01 | Qualification using 3 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register. |
| | | 10 | Qualification using 6 samples. Valid for pins configured as GPIO or a peripheral function. The time between samples is specified in the GPACTRL register. |
| | | 11 | Asynchronous. (no synchronization or qualification). This option applies to pins configured as peripherals only. If the pin is configured as a GPIO input, then this option is the same as 0,0 or synchronize to SYSCLKOUT. |

⁽¹⁾ This register is EALLOW protected. See Section 7.2 for more information.

```

// Step 6. XINT 초기화
//-----
EALLOW;
GpioIntRegs.GPIOXINT3SEL.bit.GPIOSEL = 47; // 외부 인터럽트 XINT3로 사용할 핀 선택: GP1047
GpioIntRegs.GPIOXINT4SEL.bit.GPIOSEL = 46; // 외부 인터럽트 XINT4로 사용할 핀 선택: GP1046
GpioIntRegs.GPIOXINT5SEL.bit.GPIOSEL = 45; // 외부 인터럽트 XINT5로 사용할 핀 선택: GP1045
GpioIntRegs.GPIOXINT6SEL.bit.GPIOSEL = 44; // 외부 인터럽트 XINT6로 사용할 핀 선택: GP1044
EDIS;

XIntruptRegs.XINT3CR.bit.POLARITY = 0; // XINT3 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
XIntruptRegs.XINT4CR.bit.POLARITY = 0; // XINT4 인터럽트 발생 조건 설정: 입력 신호의 상승 엣지
XIntruptRegs.XINT5CR.bit.POLARITY = 0; // XINT5 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
XIntruptRegs.XINT6CR.bit.POLARITY = 0; // XINT6 인터럽트 발생 조건 설정: 입력 신호의 하강 & 상승 엣지

XIntruptRegs.XINT3CR.bit.ENABLE = 1; // XINT3 인터럽트 : Enable
XIntruptRegs.XINT4CR.bit.ENABLE = 1; // XINT4 인터럽트 : Enable
XIntruptRegs.XINT5CR.bit.ENABLE = 1; // XINT5 인터럽트 : Enable
XIntruptRegs.XINT6CR.bit.ENABLE = 1; // XINT6 인터럽트 : Enable

// 외부 인터럽트 포함된 벡터 활성화
PieCtrlRegs.PIEIER12.bit.INTx1 = 1; // PIE 인터럽트(XINT3) : Enable
PieCtrlRegs.PIEIER12.bit.INTx2 = 1; // PIE 인터럽트(XINT4) : Enable
PieCtrlRegs.PIEIER12.bit.INTx3 = 1; // PIE 인터럽트(XINT5) : Enable
PieCtrlRegs.PIEIER12.bit.INTx4 = 1; // PIE 인터럽트(XINT6) : Enable
IER |= M_INT12; // CPU 인터럽트(INT12) : Enable
//-----

```

- **GPIOXINnSEL**: 외부 인터럽트로 사용할 핀을 선택한다. XINT3부터 XINT6까지 외부 인터럽트로 사용하며 그에 해당하는 Interrupt Select Register와 Configuration Register는 위 표와 같다.

Table 82. GPIO XINT3 - XINT7 Interrupt Select (GPIOXINTnSEL) Register Field Descriptions⁽¹⁾

| Bits | Field | Value | Description ⁽²⁾ |
|------|--------------|-------|---|
| 15-5 | Reserved | | Reserved |
| 4-0 | GPIOXINTnSEL | | Select the port B GPIO signal (GPIO32 - GPIO63) that will be used as the XINTn interrupt source. In addition, you can configure the interrupt in the XINTnCR register described in Section 8.6. |
| | | 00000 | Select the GPIO32 pin as the XINTn interrupt source (default) |
| | | 00001 | Select the GPIO33 pin as the XINTn interrupt source |
| | | ... | ... |
| | | 11110 | Select the GPIO62 pin as the XINTn interrupt source |
| | | 11111 | Select the GPIO63 pin as the XINTn interrupt source |

⁽¹⁾ n = 3, 4, 5, 6, or 7

⁽²⁾ This register is EALLOW protected. See Section 7.2 for more information.

Table 83. XINT3 - XINT7 Interrupt Select and Configuration Registers

| n | Interrupt | Interrupt Select Register | Configuration Register |
|---|-----------|---------------------------|------------------------|
| 3 | XINT3 | GPIOXINT3SEL | XINT3CR |
| 4 | XINT4 | GPIOXINT4SEL | XINT4CR |
| 5 | XINT5 | GPIOXINT5SEL | XINT5CR |
| 6 | XINT6 | GPIOXINT6SEL | XINT6CR |
| 7 | XINT7 | GPIOXINT7SEL | XINT7CR |

- **Polarity**: 입력 값에 따라 인터럽트가 언제 발생할 것인지를 설정하게 된다. 0일 경우 하강edge, 1일 경우 상승 edge, 2일 경우 하강 edge, 3일 경우 하강&상승 edge에서 인터럽트가 발생하게 된다. 이번 프로젝트에서는 모두 '0'을 입력하여 하강edge일 때 인터럽트가 발생하도록 설정하였다.

Table 121. External Interrupt n Control Register (XINTnCR) Field Descriptions

| Bits | Field | Value | Description |
|------|----------|-------|--|
| 15-4 | Reserved | | Reads return zero; writes have no effect. |
| 3-2 | Polarity | | This read/write bit determines whether interrupts are generated on the rising edge or the falling edge of a signal on the pin. |
| | | 00 | Interrupt generated on a falling edge (high-to-low transition) |
| | | 01 | Interrupt generated on a rising edge (low-to-high transition) |
| | | 10 | Interrupt is generated on a falling edge (high-to-low transition) |
| | | 11 | Interrupt generated on both a falling edge and a rising edge (high-to-low and low-to-high transition) |
| 1 | Reserved | | Reads return zero; writes have no effect |
| 0 | Enable | | This read/write bit enables or disables external interrupt XINTn. |
| | | 0 | Disable interrupt |
| | | 1 | Enable interrupt |

- **Enable:** 1값을 입력하게 되면 인터럽트를 enable한다.
- **PIEIER12:** Group Enable Register이며, INTx1, 2, 3, 4를 사용하여 외부 인터럽트 XINT3, 4, 5, 6에 해당하는 각각의 PIE 인터럽트를 Enable 상태로 설정해준다.
- **IER |= M_INT12:** CPU 인터럽트를 Enable 상태로 설정해준다.

Step 4. ADC 초기화

```

InitAdc();

// ADC 설정
AdcRegs.ADCR3.bit.ADCCLKPS = 3;           // ADCCLK = HSPCLK/(ADCCLKPS*2)/(CPS+1)
AdcRegs.ADCR1.bit.CPS = 1;               // ADCCLK = 75MHz/((3*2)/(1+1)) = 6.25MHz
AdcRegs.ADCR1.bit.ACQ_PS = 3;             // 샘플/홀드 사이클 = ACQ_PS + 1 = 4 (ADCCLK기준)
AdcRegs.ADCR1.bit.SEQ_CASC = 1;          // 시퀀스 모드 설정: 직렬 시퀀스 모드 (0:병렬 모드, 1:직렬 모드)
AdcRegs.ADCR1.bit.MAX_CONV1 = 1;         // ADC 채널수 설정: 1개(=MAX_CONV+1)채널을 ADC
AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0;     // ADC 순서 설정: 첫번째로 ADCINA2 채널을 ADC
AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 8;     // ADC 순서 설정: 2번째로 ADCINB0 채널을 ADC

AdcRegs.ADCR2.bit.EPWM_SOCB_SEQ = 1;     // ePWM_SOCB로 ADC 시퀀스 시동
AdcRegs.ADCR2.bit.INT_ENA_SEQ1 = 1;      // ADC 시퀀스 완료시 인터럽트 발생 설정

```

- **InitAdc():** 사용자가 ADC 회로를 이용하여 데이터를 변환할 수 있도록 ADC 회로를 초기화한다.
- **ADCCLKPS:** 코어 클럭을 나누는 역할을 한다. ADCCLK 계산 과정에서 HSPCLK는 ADCCLKPS*2 값으로 나뉘지게 된다. 해당 프로젝트에서는 ADCCLKPS 값에 3을 입력하였다.

$$ADCCLK = \frac{HSPCLK}{ADCCLKPS * 2} * \frac{1}{CPS + 1} \quad \cdots \text{식 (1)}$$

| Bit(s) | Name | Value | Description |
|--------|----------------|--|--|
| | | 0 | All analog circuitry inside the core except the bandgap and reference circuitry is powered down. |
| | | 1 | The analog circuitry inside the core is powered up. |
| 4-1 | ADCCLKPS [3:0] | <div>Core clock divider. 28x peripheral clock, HSPCLK, is divided by 2*ADCCLKPS[3:0], except when ADCCLKPS[3:0] is 0000, in which case HSPCLK is directly passed on. The divided clock is further divided by ADCTRL1[7]+1 to generate the core clock, ADCLK.</div> <div>ADCCLKPS [3:0] Core Clock Divider ADCLK</div> <div>0 HSPCLK/(ADCTRL1[7] + 1)</div> <div>0001 1 HSPCLK/[2*(ADCTRL1[7] + 1)]</div> <div>0010 2 HSPCLK/[4*(ADCTRL1[7] + 1)]</div> <div>0011 3 HSPCLK/[6*(ADCTRL1[7] + 1)]</div> | |

- **CPS:** 코어 클럭 전치 분주기 역할을 한다. ADCCLK 계산 과정에서 HSPCLK는 CPS가 0이면 1로, 1이면 2로 나뉜다. 해당 예제에서는 CPS 값을 1을 입력하였기 때문에 식 (1)에 값을 대입하면 식 (2)와 같이 계산된다.

$$ADCCLK = \frac{75MHz}{3 * 2} * \frac{1}{1 + 1} = 6.25MHz \quad \cdots \text{식 (2)}$$

| | | | |
|---|-----|---|--|
| 7 | CPS | | Core clock prescaler. The prescaler is applied to divided device peripheral clock, HSPCLK. |
| | | 0 | ADCCLK = F _{clk} /1 |
| | | 1 | ADCCLK = F _{clk} /2 |
| Note: F _{clk} = Prescaled HSPCLK (ADCCLKPS[3:0]) | | | |

• **ACQ_PS**: 데이터 수집 윈도우 크기이다. SOC 펄스 길이를 조절하기 위한 비트 필드로, 샘플링 주기가 끝나는 시간을 결정한다. SOC 펄스의 길이는 ADCLK 주기의 (ADCTRL1[11:8]+1)배가 된다. 해당 예제에서는 ACQ_PS 값에 3을 입력하여 샘플/홀드 사이클을 4로 설정하였다.

| | | |
|------|-------------|---|
| 11-8 | ACQ_PS[3:0] | Acquisition window size. This bit field controls the width of SOC pulse, which, in turn, determines for what time duration the sampling switch is closed. The width of SOC pulse is ADCTRL1[11:8] + 1 times the ADCLK period. |
|------|-------------|---|

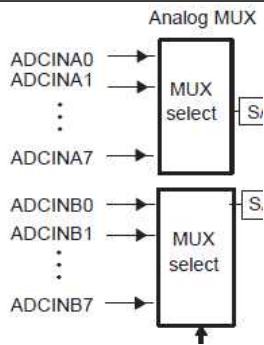
• **SEQ_CASC**: 시퀀스 모드를 설정한다. 해당 비트에 0이 입력되면 병렬 시퀀스 모드가 되어 SEQ1, SEQ2가 두 개의 8-state 시퀀서로 동작하고, 1이 입력되면 직렬 시퀀스 모드가 되어 SEQ1, SEQ2가 하나의 16-state 시퀀서로 동작한다. 해당 예제에서는 1이 입력되었기 때문에 직렬 시퀀스 모드로 동작한다.

| | | |
|---|----------|---|
| 4 | SEQ_CASC | Cascaded sequencer operation. This bit determines whether SEQ1 and SEQ2 operate as two 8-state sequencers or as a single 16-state sequencer (SEQ). 0 Dual-sequencer mode. SEQ1 and SEQ2 operate as two 8-state sequencers. 1 Cascaded mode. SEQ1 and SEQ2 operate as a single 16-state sequencer (SEQ). |
|---|----------|---|

• **MAX_CONV1**: 시퀀서의 입력받을 채널의 수를 설정한다. (MAX_CONV1 + 1)개의 채널을 사용할 수 있다. 해당 예제에서는 MAX_CONV1 값이 0이기 때문에 1개의 채널만 이용한다.

Table 2-4. Maximum Conversion Channels Register (ADCMAXCONV) Field Descriptions

| Bit(s) | Name | Description |
|--------|-----------|--|
| 15-7 | Reserved | Reads return a zero. Writes have no effect. |
| 6-0 | MAX_CONVn | MAX_CONVn bit field defines the maximum number of conversions executed in an autoconversion session. The bit fields and their operation vary according to the sequencer modes (dual/cascaded). For SEQ1 operation, bits MAX_CONV1[2:0] are used. For SEQ2 operation, bits MAX_CONV2[2:0] are used. For SEQ operation, bits MAX_CONV1[3:0] are used. An autoconversion session always starts with the initial state and continues sequentially until the end state if allowed. The result buffer is filled in a sequential order. Any number of conversions between 1 and (MAX_CONVn + 1) can be programmed for a session. |



CONV00에 0, CONV01에 8을 입력하여, 첫 번째 채널인 ADCINA0와 9번째 채널인 ADCINB0 채널을 이용한다.

```
//ePWM_SOCB 이벤트 트리거 설정
EPwm5Regs.ETSEL.bit.SOCBEN = 1;           // SOCB 이벤트 트리거 Enable
EPwm5Regs.ETSEL.bit.SOCBSEL = 2;          // SOCB 트리거 조건 : 카운터 주기 일치 시
EPwm5Regs.ETPS.bit.SOCBPRD = 1;           // SOCB 이벤트 분주 설정 : 트리거 조건 한번 마다
EPwm5Regs.TBCTL.bit.CTRMODE = 0;           // 카운트 모드 설정: Up-count 모드
EPwm5Regs.TBCTL.bit.HSPCLKDIV = 1;          // TBCLK = [SYSCLKOUT / ((HSPCLKDIV*2) * 2^(CLKDIV))]
EPwm5Regs.TBCTL.bit.CLKDIV = 1;             // TBCLK = [150MHz / (2*2)] = 37.5MHz
EPwm5Regs.TBPRD = (TBCLK / PwmCarrierFrequency) - 1; // TB주기는 (TBPRD+1)/TBCLK = 1875/37.5MHz = 50us(20KHz)
EPwm5Regs.TBCTR = 0x0000;                  // TB 카운터 초기화

// PIE의 ADC 인터럽트 활성화
PieCtrlRegs.PIEIER1.bit.INTx6 = 1;          // PIE 인터럽트(ADCINT) 활성화
IER |= M_INT1;                               // CPU 인터럽트(INT1) 활성화
//=====
```

• EPwm5Regs.ETSEL.bit.SOCBEN = 1; : '1'을 입력함으로써, EPWMxSOCB pulse를 활성화한다.

| Bits | Name | Value | Description |
|------|--------|-------|--|
| 15 | SOCBEN | 0 | Enable the ADC Start of Conversion B (EPWMxSOCB) Pulse |
| | | 1 | Disable EPWMxSOCB. |
| | | 1 | Enable EPWMxSOCB pulse. |

- EPwm5Regs.ETSEL.bit.SOCBSEL = 2; : 언제 EPWMxSOCB 펄스가 생성될지를 결정하는 비트로, '2'를 입력함으로써, TBCTR=TBPRD 일 때, 즉 카운터 주기 일치 시 트리거 된다.

| | | | |
|-------|---------|-----|---|
| 14-12 | SOCBSEL | | EPWMxSOCB Selection Options These bits determine when a EPWMxSOCB pulse will be generated. |
| | | 000 | Reserved |
| | | 001 | Enable event time-base counter equal to zero. (TBCTR = 0x0000) |
| | | 010 | Enable event time-base counter equal to period (TBCTR = TBPRD) |
| | | 011 | Reserved |
| | | 100 | Enable event time-base counter equal to CMPA when the timer is incrementing. |
| | | 101 | Enable event time-base counter equal to CMPA when the timer is decrementing. |
| | | 110 | Enable event: time-base counter equal to CMPB when the timer is incrementing. |
| | | 111 | Enable event: time-base counter equal to CMPB when the timer is decrementing. |

- EPwm5Regs.ETPS.bit.SOCBPRD = 1; : EPWMxSOCB 펄스가 생성되기 전에 몇 개의 선택된 ETSEL 이벤트가 발생해야 하는 지를 결정한다. 생성하려면 펄스를 활성화 해야 한다.(ETFLG[SOCB]=1). 이전 변환 시작 (ETFLG[SOCB]=1)에서 상태 플래그를 설정해도 SOCV펄스가 생성된다. SOCB 펄스가 생성되면 ETPS[SOCBCNT]비트가 자동으로 지워진다. 이번 프로젝트에서는 '1'을 입력함으로써, 첫 번째 이벤트에서 EPWMxSOCB펄스가 생성된다. ETPS[SOCBCNT]=0, 1.

| | | | |
|-------|---------|----|---|
| 13-12 | SOCBPRD | | ePWM ADC Start-of-Conversion B Event (EPWMxSOCB) Period Select These bits determine how many selected ETSEL[SOCBSEL] events need to occur before an EPWMxSOCB pulse is generated. To be generated, the pulse must be enabled (ETSEL[SOCBEN] = 1). The SOCB pulse will be generated even if the status flag is set from a previous start of conversion (ETFLG[SOCB] = 1). Once the SOCB pulse is generated, the ETPS[SOCBCNT] bits will automatically be cleared. |
| | | 00 | Disable the SOCB event counter. No EPWMxSOCB pulse will be generated |
| | | 01 | Generate the EPWMxSOCB pulse on the first event: ETPS[SOCBCNT] = 0,1 |
| | | 10 | Generate the EPWMxSOCB pulse on the second event: ETPS[SOCBCNT] = 1,0 |
| | | 11 | Generate the EPWMxSOCB pulse on the third event: ETPS[SOCBCNT] = 1,1 |

- EPwm5Regs.TBCTL.bit.CTRMODE = 0; : 입력되는 값에 따라 Counter mode를 결정한다. 해당 프로젝트에서는 '0'을 입력하였기 때문에 "Up-Count Mode"가 진행된다.

| | | | |
|-----|---------|----|---|
| 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: |
| | | 00 | Up-count mode |
| | | 01 | Down-count mode |
| | | 10 | Up-down-count mode |
| | | 11 | Stop-freeze counter operation (default on reset) |

- EPwm5Regs.TBCTL.bit.HSPCLKDIV=1; : High Speed Time-base Clock Prescale Bits.
 - Time-base clock prescale value의 일부를 결정한다.
 - 구분자(divisor)는 이벤트 관리자 주변장치에 사용되는 TMS320x281x 시스템의 HSPCLK를 에뮬레이트 한다.
- EPwm5Regs.TBCTL.bit.CLKDIV = 1; : Time-base Clock Prescale Bits.
 - 비트는 타임베이스 클럭 사전 설정 값의 일부를 결정한다.
 - TBCLK를 시스템 클럭을 "HSPCLKDIV와 CLKDIV의 곱"으로 나눠서 설정한다. 이를 나타내면 아래 식과 같다.

$$T_{BCLK} = \frac{SYSCLKOUT}{(HSPCLKDIV * CLKDIV)} = \frac{150MHz}{2*2} = 37.5MHz$$

[SYSTEM_CLOCK : 150E6, HSKPCLKDIV : 1, CLKDIV : 1]

| | | |
|-----|-----------|--|
| 9:7 | HSPCLKDIV | High Speed Time-base Clock Prescale Bits These bits determine part of the time-base clock prescale value. TBCLK = SYSCLKOUT / (HSPCLKDIV * CLKDIV) This divisor emulates the HSPCLK in the TMS320x281x system as used on the Event Manager (EV) peripheral. |
| | 000 | /1 |
| | 001 | /2 (default on reset) |
| | 010 | /4 |
| | 011 | /6 |
| | 100 | /8 |
| | 101 | /10 |
| | 110 | /12 |
| | 111 | /14 |

| | | |
|-------|--------|---|
| 12:10 | CLKDIV | Time-base Clock Prescale Bits These bits determine part of the time-base clock prescale value. TBCLK = SYSCLKOUT / (HSPCLKDIV * CLKDIV) |
| | 000 | /1 (default on reset) |
| | 001 | /2 |
| | 010 | /4 |
| | 011 | /8 |
| | 100 | /16 |
| | 101 | /32 |
| | 110 | /64 |
| | 111 | /128 |

- EPwm5Regs.TBPRD = (TBCLK/PWM_CARRIER)-1;

| Table 21. Time-Base Period Register (TBPRD) Field Descriptions | | | |
|--|-------|------------|--|
| Bits | Name | Value | Description |
| 15-0 | TBPRD | 0000-FFFFh | <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[PRDL] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> • If TBCTL[PRDL] = 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 zero. • If TBCTL[PRDL] = 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. |

- TBPRD는 PRD의 값을 설정하는 레지스터이다.
- 초기에 설정해준 값을 이용해서 TBPRD의 값을 구하면 다음과 같다.

$$TBPRD = (TBCLK/PWM_CARRIER) - 1 = \frac{37.5MHz}{20kHz} - 1 = 1,874$$

$$TB\text{주기} = (TBPRD + 1) / TBCLK = \frac{1875}{37.5MHz} = 50\mu s = 20kHz$$

- EPwm5Regs.TBCTR = 0; : 이 비트를 read하면, 현재 Time-Base Counter의 값이 제공된다.
위 코드에서는 '0'의 값을 입력함으로써 카운터를 클리어한다.

| Table 23. Time-Base Counter Register (TBCTR) Field Descriptions | | | |
|---|-------|-----------|--|
| Bits | Name | Value | Description |
| 15-0 | TBCTR | 0000-FFFF | <p>Reading these bits gives the current time-base counter value.</p> <p>Writing to these bits sets the current time-base counter value. The update happens as soon as the write occurs; the write is NOT synchronized to the time-base clock (TBCLK) and the register is not shadowed.</p> |

```

Step 7
7.1 Initialize S/W modules and Variables
-----*/

BackTicker = 0;
EPwm5IsrTicker = 0;

PwmCarrierFrequency = PWM_CARRIER;

FallingEdgeDelay1 = (1.0 / TBCLK) * EPwm5Regs.DBFEED;
RisingEdgeDelay1 = (1.0 / TBCLK) * EPwm5Regs.DBRED;
//=====
// Step 4. Initialize Application Variables
//=====
ADC_value01 = 0;
ADC_value02 = 0;
Loop_cnt = 0;
//=====

```

- **InitEPwm5Module();** : 가변저항 변화에 대한 PWM 주파수 가변이 가능하도록 설정하는 ePWM 함수이다.
- **PwmCarrierFrequency = PWM_CARRIER;** : PwmCarrierFrequency를 20kHz로 설정한다.
- FallingEdgeDelay는 하강지연시간으로 카운터 하나의 주기에 레지스터 DBFEED를 곱하여 구할 수 있고, RisingEdgeDelay는 상승지연시간으로 카운터 하나의 주기에 레지스터 DBRED를 곱하여 구할 수 있다. 두 지연 시간을 식으로 나타내면 아래와 같다.

$$\text{하강지연시간} = DBFEED * T_{BCLK}$$

$$\text{상승지연시간} = DBRED * T_{BCLK}$$

- ADC_value01, 02의 값과 Loop_cnt 값을 0으로 초기화한다.

```

/*-----*/
Step 8
8.1 Enable Global realtime interrupt DBGEM
8.2 Enable Global Interrupt
-----*/

ERTM; /* Enable Global realtime interrupt DBGEM */
EINT; /* Enable Global interrupt INTM */

```

- **ERTM:** DBGEM을 clear하여 디버그 이벤트를 Enable 시킨다.
- **EINT:** 앞의 DINT와 반대로 전역 인터럽트를 활성화 시킨다.

```

for (;;)
{
    BackTicker++;
    if (mode == 1)
    {
        EPwm5Regs.TBPRD = (TBCLK / PwmCarrierFrequency) - 1;
        PwmDutyRatio = ADC_value01 * 0.475 / 65536;
        EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1) * PwmDutyRatio); /* Set Compare A Value to 50% */
        EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatio); //상보적
        /* Setup Counter Mode and Clock */
        EPwm5Regs.TBCTL.bit.CTRMODE = 2; //up-downcount
        /* Set Dead-time */
        EPwm5Regs.DBCTL.bit.IN_MODE = 2; /* EPWMxA is the source for both falling-edge & rising-edge delay */
        EPwm5Regs.DBCTL.bit.OUT_MODE = 3; /* Dead-band is fully enabled for both rising-edge delay on EPWMxA and falling-edge */
        EPwm5Regs.DBCTL.bit.POLSEL = 2; /* Active High Complementary (AHC). EPWMxB is inverted */
        EPwm5Regs.DBFEED = 450; /* 3usec, Falling Edge Delay */
        EPwm5Regs.DBRED = 450; /* 3usec, Rising Edge Delay */
    }
}

```

모드 1일 경우 아래와 같은 동작을 수행한다.

- **EPwm5Regs.TBPRD = (TBCLK/PWM_CARRIER)-1;** :

TBPRD는 PRD의 값을 설정하는 레지스터이다. 초기에 설정해준 값을 이용해서 TBPRD의 값을 구하면 다음과 같다.

$$TBPRD = (TBCLK/PWM_CARRIER) - 1 = \frac{37.5MHz}{20kHz} - 1 = 1,874$$

$$TB\text{주기} = (TBPRD+1)/TBCLK = \frac{1875}{37.5MHz} = 50\mu s = 20kHz$$

• PWM_DUTY_RATIO_A = ADC_value01 * 0.475/65536;

: 듀티 비를 ADC_value01 값에 따라 달라지는 변수로 설정한다. 여기서 ADC_value01은 0V에서 0을 갖고, 3V에서 최댓값인 65536을 가진다. 프로젝트에서는 최댓값이 PWM 주기 절반의 95%로 설정하기 위해, 0.475/65536을 곱해주었다.

• EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1) * PwmDutyRatio);

: CMPA의 값을 설정해주는 레지스터로, CMPA의 값은 (TBPRD + 1) * PwmDutyRatio에 의해 정해진다.

• EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatio);

: EPWM5B의 PWM파형은 EPWM5A와 상보적으로 동작해야하기 때문에, CMPB의 값은 (TBPRD + 1)의 값에 (1-PwmDutyRatio) 값을 곱해줌으로써 결정된다.

Table 27. Counter-Compare B Register (CMPB) Field Descriptions

| Bits | Name | Description |
|------|------|---|
| 15-0 | CMPB | <p>The value in the active CMPB 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 B" 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. 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[SHDWBMODE] bit. By default this register is shadowed.</p> <ul style="list-style-type: none"> • If CMPCTL[SHDWBMODE] = 0, then the shadow is enabled and any write or read will automatically go to the shadow register. In this case, the CMPCTL[LOADBMODE] bit field determines which event will load the active register from the shadow register: • Before a write, the CMPCTL[SHDWBFULL] bit can be read to determine if the shadow register is currently full. • If CMPCTL[SHDWBMODE] = 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. |

• EPwm5Regs.TBCTL.bit.CTRMODE = 2;

: 일반적으로 한 번 구성되면, 정상 작동 중에는 변경되지 않는다. 카운터의 모드를 변경하면, 다음 TBCLK Edge에서 변경사항이 적용되며, 현재 카운터 값은 모드 변경 전 값에서 증가 또는 감소한다. 위 프로젝트에서는 '2'를 입력함으로써 "Up-Down-count mode"가 동작된다.

| | | | | | | | | | | |
|-----|--|--|----|---------------|----|-----------------|----|--------------------|----|--|
| 1:0 | CTRMODE | <p>Counter Mode</p> <p>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.</p> <p>These bits set the time-base counter mode of operation as follows:</p> <table><tr><td>00</td><td>Up-count mode</td></tr><tr><td>01</td><td>Down-count mode</td></tr><tr><td>10</td><td>Up-down-count mode</td></tr><tr><td>11</td><td>Stop-freeze counter operation (default on reset)</td></tr></table> | 00 | Up-count mode | 01 | Down-count mode | 10 | Up-down-count mode | 11 | Stop-freeze counter operation (default on reset) |
| 00 | Up-count mode | | | | | | | | | |
| 01 | Down-count mode | | | | | | | | | |
| 10 | Up-down-count mode | | | | | | | | | |
| 11 | Stop-freeze counter operation (default on reset) | | | | | | | | | |

• EPwm4Regs.DBCTL.bit.IN_MODE=2;

: Dead Band의 입력 모드를 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S4는 0, S5는 1의 값을 갖게 되어, EPWMxA 신호는 Rising-edge delay를, EPWMxB 신호는 Falling-edge delay를 결정하게 된다.

| | | | |
|-----|---------|--|--|
| 5-4 | IN_MODE | | <p>Dead Band Input Mode Control</p> <p>Bit 5 controls the S5 switch and bit 4 controls the S4 switch shown in Figure 31.</p> <p>This allows you to select the input source to the falling-edge and rising-edge delay.</p> <p>To produce classical dead-band waveforms the default is EPWMxA In is the source for both falling and rising-edge delays.</p> <p>00 EPWMxA In (from the action-qualifier) is the source for both falling-edge and rising-edge delay.</p> <p>01 EPWMxB In (from the action-qualifier) is the source for rising-edge delayed signal. EPWMxA In (from the action-qualifier) is the source for falling-edge delayed signal.</p> <p>10 EPWMxA In (from the action-qualifier) is the source for rising-edge delayed signal. EPWMxB In (from the action-qualifier) is the source for falling-edge delayed signal.</p> <p>11 EPWMxB In (from the action-qualifier) is the source for both rising-edge delay and falling-edge delayed signal.</p> |
|-----|---------|--|--|

• EPwm4Regs.DBCTL.bit.OUT_MODE=3;

: Dead Band의 출력 모드를 제어하는 레지스터이다. 코드와 같이 '3'을 입력하게 되면 스위치 S1과 S0이 모두 1의 값을 갖게 되어, EPWMxA는 상승엣지 지연을, EPWMxB는 하강엣지 지연을 갖게된다.

| | | | |
|-----|----------|--|--|
| 1-0 | OUT_MODE | | <p>Dead-band Output Mode Control</p> <p>Bit 1 controls the S1 switch and bit 0 controls the S0 switch shown in Figure 31.</p> <p>This allows you to selectively enable or bypass the dead-band generation for the falling-edge and rising-edge delay.</p> <p>00 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.</p> <p>In this mode, the POLSEL and IN_MODE bits have no effect.</p> <p>01 Disable rising-edge delay. The EPWMxA signal from the action-qualifier is passed straight through to the EPWMxA input of the PWM-chopper submodule.</p> <p>The falling-edge delayed signal is seen on output EPWMxB. The input signal for the delay is determined by DBCTL[IN_MODE].</p> <p>10 The rising-edge delayed signal is seen on output EPWMxA. The input signal for the delay is determined by DBCTL[IN_MODE].</p> <p>Disable falling-edge delay. The EPWMxB signal from the action-qualifier is passed straight through to the EPWMxB input of the PWM-chopper submodule.</p> <p>11 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].</p> |
|-----|----------|--|--|

• EPwm4Regs.DBCTL.bit.POLSEL=2;

: 극성을 선택을 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S3과 S2에는 각각 1, 0 값을 갖게 되어 EPWMxB는 반전된다. 이 동작을 Active high complementary (AHC)라고 한다. 즉, 하강엣지 지연 신호를 반전 시키는 것이다.

| | | | |
|-----|--------|--|--|
| 3-2 | POLSEL | | <p>Polarity Select Control</p> <p>Bit 3 controls the S3 switch and bit 2 controls the S2 switch shown in Figure 31.</p> <p>This allows you to selectively invert one of the delayed signals before it is sent out of the dead-band submodule.</p> <p>The following descriptions correspond to classical upper/lower switch control as found in one leg of a digital motor control inverter.</p> <p>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.</p> <p>00 Active high (AH) mode. Neither EPWMxA nor EPWMxB is inverted (default).</p> <p>01 Active low complementary (ALC) mode. EPWMxA is inverted.</p> <p>10 Active high complementary (AHC). EPWMxB is inverted.</p> <p>11 Active low (AL) mode. Both EPWMxA and EPWMxB are inverted.</p> |
|-----|--------|--|--|

• EPwm4Regs.DBFED=450;

: Dead-Band Generator Falling Edge Delay Register (DBFED)는 아래 식과 같이 하강 지연시간을 결정한다.

$$\text{하강지연시간} = \text{DBFED} * T_{TCLK} = 450 * \frac{1}{150 * 10^6} = 3\mu s$$

• EPwm4Regs.DBRED=450;

: Dead-Band Generator Rising Edge Delay Register (DBRED)는 아래 식과 같이 상승 지연시간을 결정한다.

$$\text{상승지연시간} = \text{DBRED} * T_{\text{TBCLK}} = 450 * \frac{1}{150 * 10^6} = 3\mu\text{s}$$

```

else if (mode == 2)
{
    EPwm5Regs.CMPCTL.bit.SHDWBMODE = 0;    /* Compare B Register is loaded from its shadow when CNTR=Zero */
    EPwm5Regs.CMPCTL.bit.LOADBMODE = 0;
    PWM_CARRIER2 = 100E3 + 200E3*ADC_value01 / 65536;
    EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER2) - 1; /* Set Timer Period, (150MHz/100KHz)-1 = 1,499 (0x050B) */
    EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1)*0.5); /* Set Compare A Value to 50% */

    EPwm5Regs.TBCTL.bit.CTRMODE = 0; //upcount
    EPwm5Regs.AQCTLA.bit.ZRO = 2;    /* Set EPWM4A on CNTR=Zero */
    /* Set Dead-time */
    EPwm5Regs.DBCTL.bit.IN_MODE = 0;    /* EPWMxA is the source for both falling-edge & rising-edge delay */
    EPwm5Regs.DBCTL.bit.OUT_MODE = 3;    /* Dead-band is fully enabled for both rising-edge delay on EPWMxA and falling-edge */
    EPwm5Regs.DBCTL.bit.POLSEL = 2;    /* Active High Complementary (AHC), EPWMxB is inverted */
    EPwm5Regs.DBFED = 75;    /* 2usec, Falling Edge Delay */
    EPwm5Regs.DBRED = 75;    /* 1usec, Rising Edge Delay */
}

```

모드 2일 경우 아래와 같은 동작을 수행한다.

- EPwm5Regs.CMPCTL.bit.SHDWBMODE = 0;

: Counter-compare B(CMPB) Register Operating Mode로, '0'을 입력하면 Shadow Mode로 동작한다. 이중 버퍼로써 동작하며, 모든 write는 CPU를 통해 Shadow register에 접속한다.

| | | | |
|---|-----------|---|--|
| 6 | SHDWBMODE | | Counter-compare B (CMPB) Register Operating Mode |
| | | 0 | Shadow mode. Operates as a double buffer. All writes via the CPU access the shadow register. |
| | | 1 | Immediate mode. Only the active compare B register is used. All writes and reads directly access the active register for immediate compare action. |

- EPwm5Regs.CMPCTL.bit.LOADBMODE = 0;

: Shadow Selct Mode로부터 CMPB를 활성화시킨다. '0'을 입력할 경우 CTR=Zero가 된다. (TBCTR=0x0000)

| | | | |
|-----|-----------|----|---|
| 3-2 | LOADBMODE | | Active Counter-Compare B (CMPB) Load From Shadow Select Mode This bit has no effect in immediate mode (CMPCTL[SHDWBMODE] = 1). |
| | | 00 | Load on CTR = Zero: Time-base counter equal to zero (TBCTR = 0x0000) |
| | | 01 | Load on CTR = PRD: Time-base counter equal to period (TBCTR = TBPRD) |
| | | 10 | Load on either CTR = Zero or CTR = PRD |
| | | 11 | Freeze (no loads possible) |

- PWM_CARRIER2 = 100E3 + 200E3*ADC_value01 / 65536;

: PWM_CARRIER2 값은 ADC_value01에 의해 결정된다.

- EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER2) -1;

: TBPRD의 값은 TBCLK와 PWM_CARRIER2의 연산에 의해 결정된다.

- EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1) *0.5);

: Mode 1과 달리 CMPA 값을 변수인 PWM Duty값을 통해 변경하는 것이 아니라, 상보적인 동작을 위해 PWM Duty를 50%로 고정해 주었다.

Table 26. 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. |

- EPwm5Regs.TBCTL.bit.CTRMODE = 0;

: CTRMODE에 '0'을 입력함으로써 "Up-count mode"가 동작된다.

- EPwm5Regs.AQCTLA.bit.ZRO =2;

: 카운터가 0일 때 동작한다. 위 코드에서는 '1'을 입력함으로써 "Set : Force EPWMxA output high" 동작을 한다.

| | | | |
|-----|-----|--|---|
| 1-0 | ZRO | | <p>Action when counter equals zero.</p> <p>Note: By definition, in count up-down mode when the counter equals 0 the direction is defined as 1 or counting up.</p> <p>00 Do nothing (action disabled)</p> <p>01 Clear: force EPWMxA output low.</p> <p>10 Set: force EPWMxA output high.</p> <p>11 Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low.</p> |
|-----|-----|--|---|

- EPwm4Regs.DBCTL.bit.IN_MODE=0;

: Dead Band의 입력 모드를 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S4와 S5 모두 0의 값을 갖게 되어, Rising-edge delay와 Falling-edge delay모두 EPWMxA에 의해 결정된다.

- EPwm4Regs.DBCTL.bit.OUT_MODE=3;

: Dead Band의 출력 모드를 제어하는 레지스터이다. 코드와 같이 '3'을 입력하게 되면 스위치 S1과 S0이 모두 1의 값을 갖게 되어, EPWMxA는 상승엣지 지연을, EPWMxB는 하강엣지 지연을 갖게된다.

- EPwm4Regs.DBCTL.bit.POLSEL=2;

: 극성을 선택을 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S3과 S2에는 각각 1, 0 값을 갖게 되어 EPWMxB는 반전된다. 이 동작을 Active high complementary (AHC)라고 한다. 즉, 하강엣지 지연 신호를 반전 시키는 것이다.

- EPwm4Regs.DBFED=75;

: Dead-Band Generator Falling Edge Delay Register (DBFED)는 아래 식과 같이 하강 지연시간을 결정한다.

$$\text{하강지연시간} = \text{DBFED} * T_{TBCLK} = 75 * \frac{1}{150 * 10^6} = 0.5 \mu s$$

- EPwm4Regs.DBRED=75;

: Dead-Band Generator Rising Edge Delay Register (DBRED)는 아래 식과 같이 상승 지연시간을 결정한다.

$$\text{상승지연시간} = \text{DBRED} * T_{TBCLK} = 75 * \frac{1}{150 * 10^6} = 0.5 \mu s$$

```

else if (mode == 3)
{
    EPwm5Regs.TBPRD = (TBCLK / PwmCarrierFrequency) - 1;
    EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1) * 0.5);
    deadt = (float32)75 * ADC_value01 / 65536;
    EPwm5Regs.TBCTL.bit.CTRMODE = 0; // upcount
    EPwm5Regs.AQCTLA.bit.ZRO = 2; /* Set EPWM4A on CNTR=Zero */
    /* Set Dead-time */
    EPwm5Regs.DBCTL.bit.IN_MODE = 0; /* EPWMxA is the source for both falling-edge & rising-edge delay */
    EPwm5Regs.DBCTL.bit.OUT_MODE = 3; /* Dead-band is fully enabled for both rising-edge delay on EPWMxA and falling-edge
    EPwm5Regs.DBCTL.bit.POLSEL = 2;
    EPwm5Regs.DBFED = deadt; /* 2usec, Falling Edge Delay */
    EPwm5Regs.DBRED = 0; /* 1usec, Rising Edge Delay */
}

```

모드 3의 경우 아래와 같이 동작을 하게된다.

- EPwm5Regs.TBPRD = (TBCLK/PWM_CARRIER)-1;

: TBPRD는 PRD의 값을 설정하는 레지스터이다. 초기에 설정해준 값을 이용해서 TBPRD의 값을 구하면 다음과 같다.

$$TBPRD = (TBCLK/PWM_CARRIER) - 1 = \frac{37.5MHz}{20kHz} - 1 = 1,874$$

$$TB주기 = (TBPRD+1)/TBCLK = \frac{1875}{37.5MHz} = 50us = 20kHz$$

- EPwm5Regs.CMPA.half.CMPA = (Uint16)((EPwm5Regs.TBPRD + 1) * 0.5);

: Mode 2와 같이 상보적인 동작을 위해 PWM Duty를 50%로 고정해 주었다.

- deadt = (float32)75 * ADC_value01 / 65536; : 데드타임을 ADC_value01값에 의해 변경되도록 설정하였다.

- EPwm4Regs.DBCTL.bit.IN_MODE=0;

: Dead Band의 입력 모드를 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S4와 S5 모두 0의 값을 갖게 되어, Rising-edge delay와 Falling-edge delay 모두 EPWMxA에 의해 결정된다.

- EPwm4Regs.DBCTL.bit.OUT_MODE=3;

: Dead Band의 출력 모드를 제어하는 레지스터이다. 코드와 같이 '3'을 입력하게 되면 스위치 S1과 S0이 모두 1의 값을 갖게 되어, EPWMxA는 상승엣지 지연을, EPWMxB는 하강엣지 지연을 갖게된다.

- EPwm4Regs.DBCTL.bit.POLSEL=2;

: 극성을 선택을 제어하는 레지스터이다. 코드와 같이 '2'를 입력하게 되면 스위치 S3과 S2에는 각각 1, 0 값을 갖게 되어 EPWMxB는 반전된다. 이 동작을 Active high complementary (AHC)라고 한다. 즉, 하강엣지 지연 신호를 반전 시키는 것이다.

- EPwm4Regs.DBFED=deadt;

: Dead-Band Generator Falling Edge Delay Register (DBFED)는 아래 식과 같이 하강 지연시간을 결정한다. ADC_value01에 의해 정해진 deadt값에 의해 하강지연시간이 결정된다.

$$\text{하강지연시간} = DBFED * T_{TBCLK} = deadt * \frac{1}{150 * 10^6}$$

- EPwm4Regs.DBRED=0;

: Dead-Band Generator Rising Edge Delay Register (DBRED)는 아래 식과 같이 상승 지연시간을 결정한다. 코드에서는 DBRED 값이 '0'이기 때문에 상승지연시간은 0초이다.

$$\text{상승지연시간} = DBRED * T_{TBCLK} = 0$$


```

    if (stop1 % 2 == 1)
    {
        EPwm5Regs.AQCSFRC.bit.CSFA = 1;
    }
    else if (stop1 % 2 == 0)
    {
        EPwm5Regs.AQCSFRC.bit.CSFA = 3;
    }
    else if (stop2 % 2 == 1)
    {
        /*mode = 0;*/
        EPwm5Regs.AQCSFRC.bit.CSFB = 1;
    }
    else if (stop2 % 2 == 0)
    {
        EPwm5Regs.AQCSFRC.bit.CSFB = 3;
    }
}
FallingEdgeDelay1 = (1.0 / TBCLK) * EPwm5Regs.DBFED;
RisingEdgeDelay1 = (1.0 / TBCLK) * EPwm5Regs.DBRED;
}

```

AQ모듈에서 여섯 가지의 레지스터에 의해 PWM 상태가 결정되지만, 이러한 상태를 무시하고 CSFA, CSFB 레지스터를 통해 강제적으로 PWM 상태를 결정할 수 있다. 위 코드에서는 stop1과 stop2의 값에 따라 강제로 스위칭의 상태를 결정하게끔 설정하여야다.

• **CSFA** : Continuous Software Force on Output A.

- stop1을 2로 나눴을 때, 나머지가 1인 경우, CSFA에 '1'을 입력함으로써 출력 A를 LOW로 고정한다.
- stop1을 2로 나눴을 때, 나머지가 0인 경우, CSFA에 '3'을 입력함으로써 소프트웨어 강제 고정을 비활성화한다.

| | | |
|-----|------|---|
| 1-0 | CSFA | Continuous Software Force on Output A In immediate mode, a continuous force takes effect on the next TBCLK edge. In shadow mode, a continuous force takes effect on the next TBCLK edge after a shadow load into the active register. |
| | 00 | Forcing disabled, i.e., has no effect |
| | 01 | Forces a continuous low on output A |
| | 10 | Forces a continuous high on output A |
| | 11 | Software forcing is disabled and has no effect |

• **CSFB** : Continuous Software Force on Output B.

- stop2를 2로 나눴을 때, 나머지가 1인 경우, CSFB에 '1'을 입력함으로써 출력 B를 LOW로 고정한다.
- stop2를 2로 나눴을 때, 나머지가 0인 경우, CSFB에 '3'을 입력함으로써 소프트웨어 강제 고정을 비활성화한다.

| | | |
|-----|------|---|
| 3-2 | CSFB | Continuous Software Force on Output B In immediate mode, a continuous force takes effect on the next TBCLK edge. In shadow mode, a continuous force takes effect on the next TBCLK edge after a shadow load into the active register. To configure shadow mode, use AQSFRC[RLDCSF]. |
| | 00 | Forcing disabled, i.e., has no effect |
| | 01 | Forces a continuous low on output B |
| | 10 | Forces a continuous high on output B |
| | 11 | Software forcing is disabled and has no effect |

• **FallingEdgeDelay1** = (1.0 / TBCLK) * EPwm5Regs.DBFED,

RisingEdgeDelay1 = (1.0 / TBCLK) * EPwm5Regs.DBRED

: FallingEdgeDelay1과 RisingEdgeDelay의 값은 TBCLK와 DBFED, DBRED의 값에 의해 결정이 된다. TBCLK는 150MHz로 고정인 값을 갖지만, DBFED와 DBRED는 모드에 따라 값이 변하기 때문에 각 모드마다 갖는 FallingEdgeDelay1의 값과, RisingEdgeDelay의 값이 다르다.


```

/*-----*/
Step 10
10.1 Local Interrupt Service Routines & Functions
/*-----*/

interrupt void EPwm5Isr(void)
{
    EPwm5IsrTicker++;

    /* Clear INT flag for this timer */
    EPwm5Regs.ETCLR.bit.INT = 1;

    /* Acknowledge this interrupt to receive more interrupts from group 3 */
    PieCtrlRegs.PIEACK.bit.ACK3 = 1;
}

```

함수가 실행되면 EPwm5IsrTicker의 값을 증가시켜준다.

- EPwm5Regs.ETCLR.bit.INT = 1; : ePWM interrupt (EPWMx_INT) Flag Clear Bit.
'1'을 입력함으로써, ETFLG[INT] 플래그 비트를 지우고, 추가 인터럽트 펄스가 생성된다.

Figure 89. Event-Trigger Clear Register (ETCLR)

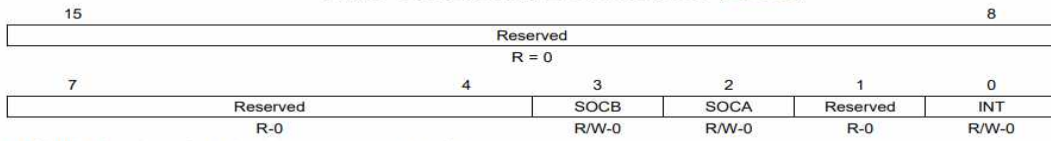


Table 47. Event-Trigger Clear Register (ETCLR) Field Descriptions

| Bits | Name | Value | Description |
|------|----------|--------|--|
| 15-4 | Reserved | | Reserved |
| 3 | SOCB | 0 1 | ePWM ADC Start-of-Conversion B (EPWMxSOCB) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[SOCB] flag bit |
| 2 | SOCA | 0 1 | ePWM ADC Start-of-Conversion A (EPWMxSOCA) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[SOCA] flag bit |
| 1 | Reserved | | Reserved |
| 0 | INT | 0 1 | ePWM Interrupt (EPWMx_INT) Flag Clear Bit Writing a 0 has no effect. Always reads back a 0 Clears the ETFLG[INT] flag bit and enable further interrupts pulses to be generated |

- PieCtrlRegs.PIEACK.bit.ACK3 =1; : ePWM interrupt (EPWMx_INT) Flag Clear Bit.

PIEACK의 각 비트는 특정 PIE 그룹을 가리킨다. 비트 0은 INT1에서 비트 11까지 MUX된 PIE 그룹 1의 인터럽트를 말하며, CPU INT12로 MUX된 PIE 그룹 12를 가리킨다.

'1'의 값을 읽는 것은 해당 그룹의 인터럽트가 CPU로 전송되고 그룹의 다른 모든 인터럽트의 차단 여부를 나타내며, '1'의 값을 쓰는 것은 각 인터럽트 비트에 1을 기록하면서, 비트가 지워지고 해당 그룹에 대한 인터럽트가 보류 중인 경우, PIE 블록이 CPU 인터럽트 입력으로 펄스를 구동할 수 있다는 것을 나타낸다.

Table 115. PIE Interrupt Acknowledge Register (PIEACK) Field Descriptions

| Bits | Field | Value | Description |
|-------|----------|---------------------------------------|--|
| 15-12 | Reserved | | Reserved |
| 11-0 | PIEACK | bit x = 0 ⁽¹⁾ bit x = 1 | Each bit in PIEACK refers to a specific PIE group. Bit 0 refers to interrupts in PIE group 1 that are MUXed into INT1 up to Bit 11, which refers to PIE group 12 which is MUXed into CPU INT12 If a bit reads as a 0, it indicates that the PIE can send an interrupt from the respective group to the CPU. Writes of 0 are ignored. Reading a 1 indicates if an interrupt from the respective group has been sent to the CPU and all other interrupts from the group are currently blocked. Writing a 1 to the respective interrupt bit clears the bit and enables the PIE block to drive a pulse into the CPU interrupt input if an interrupt is pending for that group. |

⁽¹⁾ bit x = PIEACK bit 0 - PIEACK bit 11. Bit 0 refers to CPU INT1 up to Bit 11, which refers to CPU INT12

```

void InitEPwm5Module(void)
{
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = 0;    /* TBCLK = SYSCLKOUT / (HSPCLKDIV * CLKDIV) = 150MHz */
    EPwm5Regs.TBCTL.bit.CLKDIV = 0;

    /* Setup Phase */
    EPwm5Regs.TBPHS.half.TBPHS = 0;      /* Phase is 0 */
    EPwm5Regs.TBCTL.bit.PHSEN = 0;       /* Disable phase loading */

    /* Setup Period (Carrier Frequency) */
    EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) - 1; /* Set Timer Period. (150MHz/100KHz)-1 = 1,499 (0x050B) */
    EPwm5Regs.TBCTR = 0;                  /* Clear Counter */

    /* Setup shadowing */
    EPwm5Regs.TBCTL.bit.PRDL0 = 0;        /* Period Register is loaded from its shadow when CNTR=Zero */
    EPwm5Regs.CMPCTL.bit.SHDWAMODE = 0;   /* Compare A Register is loaded from its shadow when CNTR=Zero */
    EPwm5Regs.CMPCTL.bit.LOADAMODE = 0;

    /* St actions */
    EPwm5Regs.AQCTLA.bit.CAD = 2;         /* Set EPWM4A on CNTR=Zero */
    EPwm5Regs.AQCTLA.bit.CAU = 1;         /* Clear EPWM4A on CNTR=CMPA, Up-Count */
    EPwm5Regs.AQCTLB.bit.CBU = 2;         /* Set EPWM4A on CNTR=Zero */
    EPwm5Regs.AQCTLB.bit.CBD = 1;         /* Clear EPWM4A on CNTR=CMPA, Up-Count */

    /* Set Interrupts */
    EPwm5Regs.ETSEL.bit.INTSEL = 1;       /* Select INT on CNTR=Zero */
    EPwm5Regs.ETPS.bit.INTPRD = 1;       /* Generate INT on 1st event */
    EPwm5Regs.ETSEL.bit.INTEN = 1;       /* Enable INT */
}

```

• EPwm5Regs.TBCTL.bit.HSPCLKDIV = 0; EPwm5Regs.TBCTL.bit.CLKDIV = 0;

: TBCLK는 시스템 클럭을 “HSPCLKDIV와 CLKDIV의 곱”으로 나눠서 설정한다. 두 값에 모두 ‘0’을 입력하면, 1의 값을 갖기 때문에 TBCLK는 150MHz가 된다.

$$T_{BCLK} = \frac{SYSCLKOUT}{(HSPCLKDIV * CLKDIV)} = \frac{150MHz}{1 * 1} = 150MHz$$

• EPwm5Regs.TBPSH.Half.TBPSH = 0; : 동기화 입력 신호를 공급하는 타임 베이스에 상대적인 선택된 ePWM의 타임베이스 카운터 위상을 설정한다.

Table 22. Time-Base Phase Register (TBPHS) Field Descriptions

| Bits | Name | Value | Description |
|-------|-------|-----------|---|
| 15-0 | TBPHS | 0000-FFFF | These bits set time-base counter phase of the selected ePWM relative to the time-base that is supplying the synchronization input signal. <ul style="list-style-type: none"> If TBCTL[PHSEN] = 0, then the synchronization event is ignored and the time-base counter is not loaded with the phase. If TBCTL[PHSEN] = 1, then the time-base counter (TBCTR) will be loaded with the phase (TBPHS) when a synchronization event occurs. The synchronization event can be initiated by the input synchronization signal (EPWMxSYNCl) or by a software forced synchronization. |
| PHSEN | | 0 | Counter Register Load From Phase Register Enable |
| | | 1 | Do not load the time-base counter (TBCTR) from the time-base phase register (TBPHS) |
| | | | Load the time-base counter with the phase register when an EPWMxSYNCl input signal occurs or when a software synchronization is forced by the SWFSYNC bit |

TBCTL[PHSEN]이 0이면 동기화 이벤트가 무시되고 타임베이스 카운터가 위상과 함께 로드되지 않는다. 위 코드에서는 PHSEN이 ‘0’으로 입력되었기 때문에 위상지연이 없다.

• EPwm5Regs.TBCTL.bit.PHSEN = 0; : counter Register Load From Phase Register Enable.

‘0’을 입력하였기 때문에 time-base phase register(TBPHS)에서 time-base counter(TBCTR)를 로드하지 않는다.

• EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) -1;

: TBPRD의 값은 TBCLK와 PWM_CARRIER2의 연산에 의해 결정된다.

• EPwm5Regs.TBCTR = 0; : 이 비트를 read하면, 현재 Time-Base Counter의 값이 제공된다.

위 코드에서는 ‘0’의 값을 입력함으로써 카운터를 클리어한다.

- EPwm5Regs.PRDL = 0; : Shadow Register Select로부터 Period Register를 활성화한다. 코드에서 '0'을 입력해주었기 때문에, 카운터가 0이 될 때, TBPRD는 Shadow Register로부터 로드된다.

| | | | |
|---|------|---|---|
| 3 | PRDL | 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. A write or read to the TBPRD register accesses the shadow register. |
| | | 1 | Load the TBPRD register immediately without using a shadow register. A write or read to the TBPRD register directly accesses the active register. |

- EPwm5Regs.CMPCTL.bit.SHDWBMODE = 0;

: Counter-compare B(CMPB) Register Operating Mode로, '0'을 입력하면 Shadow Mode로 동작한다. 이중 버퍼로써 동작하며, 모든 write는 CPU를 통해 Shadow register에 접속한다.

- EPwm5Regs.CMPCTL.bit.LOADBMODE = 0;

: Shadow Selct Mode로부터 CMPB를 활성화시킨다. '0'을 입력할 경우 CTR=Zero가 된다. (TBCTR=0x0000)

| | | | |
|-------|-----|----|--|
| 7-6 | CAD | 00 | Action when the counter equals the active CMPA register and the counter is decrementing. Do nothing (action disabled) |
| | | 01 | Clear: force EPWMxA output low. |
| | | 10 | Set: force EPWMxA output high. |
| | | 11 | Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low. |
| 5-4 | CAU | 00 | Action when the counter equals the active CMPA register and the counter is incrementing. Do nothing (action disabled) |
| | | 01 | Clear: force EPWMxA output low. |
| | | 10 | Set: force EPWMxA output high. |
| | | 11 | Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low. |
| 9-8 | CBU | 00 | Action when the counter equals the active CMPB register and the counter is incrementing. Do nothing (action disabled) |
| | | 01 | Clear: force EPWMxA output low. |
| | | 10 | Set: force EPWMxA output high. |
| | | 11 | Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low. |
| 11-10 | CBD | 00 | Action when the time-base counter equals the active CMPB register and the counter is decrementing. Do nothing (action disabled) |
| | | 01 | Clear: force EPWMxA output low. |
| | | 10 | Set: force EPWMxA output high. |
| | | 11 | Toggle EPWMxA output: low output signal will be forced high, and a high signal will be forced low. |

1) CAD : Counter가 active CMPA 레지스터와 같고, Counter가 감소하고 있을 때 작동한다.

코드에서는 '2'를 입력하였기 때문에 "Set: force EPWMxA output High"동작을 한다.

2) CAU : Counter가 active CMPA 레지스터와 같고, Counter가 증가하고 있을 때 작동한다.

코드에서는 '1'을 입력하였기 때문에 "Clear: force EPWMxA output Low"동작을 한다.

3) CBU : Counter가 active CMPB 레지스터와 같고, Counter가 증가하고 있을 때 작동한다.

코드에서는 '2'를 입력하였기 때문에 "Set: force EPWMxA output High"동작을 한다.

4) CBD : Counter가 active CMPB 레지스터와 같고, Counter가 감소하고 있을 때 작동한다.

코드에서는 '1'을 입력하였기 때문에 "Clear: force EPWMxA output Low"동작을 한다.

- EPwm4Regs.ETSEL.bit.INTSEL=1;

: ePWM 인터럽트로 어떤 옵션을 사용할지 결정하는 레지스터이다. 코드와 같이 '1'을 입력하게 되면, 인터럽트는 Counter의 값이 0일 때 발생한다.

| | | | |
|-----|--------|-----|---|
| 2-0 | INTSEL | | ePWM Interrupt (EPWMx_INT) Selection Options |
| | | 000 | Reserved |
| | | 001 | Enable event time-base counter equal to zero. (TBCTR = 0x0000) |
| | | 010 | Enable event time-base counter equal to period (TBCTR = TBPRD) |
| | | 011 | Reserved |
| | | 100 | Enable event time-base counter equal to CMPA when the timer is incrementing. |
| | | 101 | Enable event time-base counter equal to CMPA when the timer is decrementing. |
| | | 110 | Enable event: time-base counter equal to CMPB when the timer is incrementing. |
| | | 111 | Enable event: time-base counter equal to CMPB when the timer is decrementing. |

• EPwm4Regs.ETPS.bit.INTPRD=1;

: ePWM 인터럽트 주기를 결정하는 레지스터이다. 코드와 같이 '1'을 입력하게 되면, 인터럽트는 한 주기에 한 번씩 인터럽트를 요청한다.

| | | | |
|-----|--------|----|--|
| 1-0 | INTPRD | | ePWM Interrupt (EPWMx_INT) Period Select |
| | | | These bits determine how many selected ETSEL[INTSEL] events need to occur before an interrupt is generated. To be generated, the interrupt must be enabled (ETSEL[INT] = 1). If the interrupt status flag is set from a previous interrupt (ETFLG[INT] = 1) then no interrupt will be generated until the flag is cleared via the ETCLR[INT] bit. This allows for one interrupt to be pending while another is still being serviced. Once the interrupt is generated, the ETPS[INTCNT] bits will automatically be cleared. |
| | | | Writing a INTPRD value that is the same as the current counter value will trigger an interrupt if it is enabled and the status flag is clear. |
| | | | Writing a INTPRD value that is less than the current counter value will result in an undefined state. |
| | | | If a counter event occurs at the same instant as a new zero or non-zero INTPRD value is written, the counter is incremented. |
| | | 00 | Disable the interrupt event counter. No interrupt will be generated and ETFRC[INT] is ignored. |
| | | 01 | Generate an interrupt on the first event INTCNT = 01 (first event) |
| | | 10 | Generate interrupt on ETPS[INTCNT] = 1,0 (second event) |
| | | 11 | Generate interrupt on ETPS[INTCNT] = 1,1 (third event) |

• EPwm4Regs.ETSEL.bit.INTEN=1;

: ePWM 인터럽트를 활성화할 결정하는 레지스터이다. 코드와 같이 '1'을 입력하게 되면, 인터럽트 발생이 활성화가 된다.

| | | | |
|---|-------|---|--|
| 3 | INTEN | | Enable ePWM Interrupt (EPWMx_INT) Generation |
| | | 0 | Disable EPWMx_INT generation |
| | | 1 | Enable EPWMx_INT generation |

| | |
|---|--|
| <pre> interrupt void adc_isr(void) { ADC_value01 = AdcRegs.ADCRESULT0; //가변저항 a ADC_value02 = AdcRegs.ADCRESULT1; //가변저항 b AdcRegs.ADCCTRL2.bit.RST_SEQ1 = 1; // Reset SEQ1 AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1; // Clear INT SEQ1 bit PieCtrlRegs.PIEACK.all = PIEACK_GROUP1; // Acknowledge interrupt to PIE } </pre> | |
|---|--|

계단식 시퀀서 모드에서 ADCRESULT8~15를 통해 ADCRESULT8을 등록하면 9번째에서 16번째 변환 결과가 유지된다. ADCRESULTn 레지스터는 대기 상태가 2개인 주변 프레임 2(0x7108-0x7117)에서 읽을 때 왼쪽 정당화되고, 대기 상태가 0인 주변 프레임 0(0x0B00-0x0F)에서 읽을 때 오른쪽 정당화된다. ADC 변환 된 값이 저장되는 ADCRESULT0, 1의 값을 전역변수 ADC_value01, 02에 저장하였다. 여기서 ADC_value01은 가변저항 a의 값을 의미하고, ADC_value02는 가변저항 b의 값을 의미한다.

• `AdcRegs.ADCCTRL2.bit.RST_SEQ1 = 1;`

: 해당 비트에 '1'을 입력함으로써, 시퀀서를 상태 CONV00으로 재설정한다.

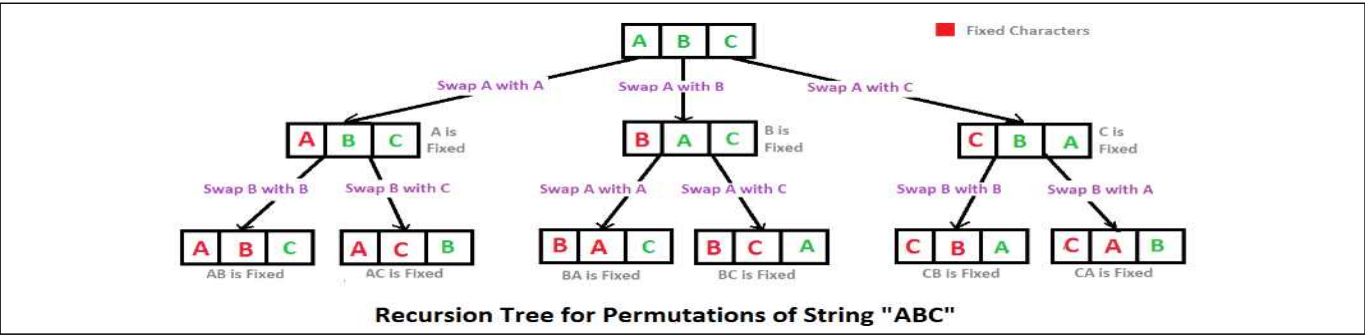
| | | | |
|----|----------|---|---|
| 14 | RST_SEQ1 | | Reset sequencer1 Writing a 1 to this bit resets SEQ1 or the cascaded sequencer immediately to an initial "pretriggered" state, i.e., waiting for a trigger at CONV00. A currently active conversion sequence will be aborted. |
| | | 0 | No action |
| | | 1 | Immediately reset sequencer to state CONV00 |

• `AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;`

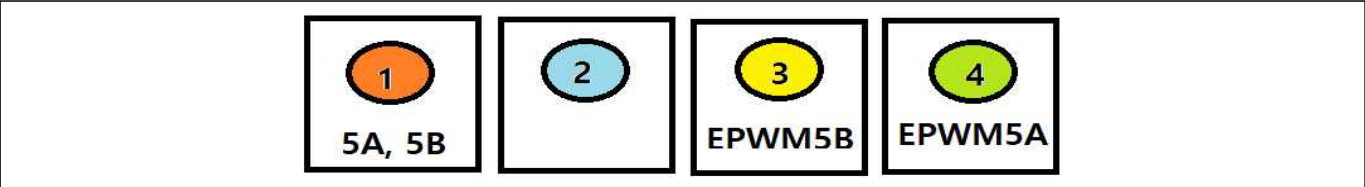
: 해당 비트에 '1'을 입력함으로써, SEQ1 interrupt flag bit을 clear한다.

| | | | |
|---|--------------|---|--|
| 4 | INT_SEQ1_CLR | | Interrupt clear bit. Read of this bit always returns 0. The clear action is a one-shot event following a write of 1 to this bit. |
| | | 0 | Writing a zero to this bit has no effect. |
| | | 1 | Writing a 1 to this bit clears the SEQ1 interrupt flag bit, INT_SEQ1. This bit does not affect the EOS_BUF1 bit. |

<코드 2. 800줄>



위 그림은 코드 1에서의 구현 동작을 설명하기 위해, 그림의 알고리즘과 같이 모드 1, 2, 3에 따라 동작하는 값들을 각각 설정하였다. 예를 들면, 모드 1은 가변저항 변화에 따른 PWM duty의 변화를 나타내고, 모드 2는 PWM 주파수 변화, 모드 3는 데드 타임 변화를 나타낸다.



코드 1과 코드 2의 차이점은 다음과 같다. 코드 1은 mode 1, 2, 3으로 설정하여 프로젝트에서 요구하는 값에 대한 동작을 조건문에 따라 실행되도록 설정하였다. 반면 코드 2는 위의 그림과 같이 스위치에 따라 서로 다른 기능을 부여하여, 스위치 입력에 따라 조건에 맞는 동작을 시행한다.

| 코드 1. 400줄 | 코드 2. 800줄 |
|--|---|
| <pre> 31 Uint16 SW1_cnt, SW2_cnt, SW3_cnt, SW4_cnt; 32 33 volatile unsigned int mode = 0, mode2 = 0, stop1 = 0, stop2 = 0; 34 35 interrupt void Xint3_isr(void) //버튼1~ 36 { </pre> | <pre> 56 57 Uint16 SW1_cnt, SW2_cnt, SW3_cnt, SW4_cnt, SW1; 58 59 Uint16 ADC_value01; 60 </pre> |

- 400줄과 800줄의 코드를 비교하면, 400줄의 경우 stop 1,2라는 변수를 선언하였지만, 800줄의 경우 전역변수로 SW1만을 설정하였다는 것을 알 수 있다.
- 앞서 언급했던 것처럼, 400줄의 경우 조건문을 통해 mode값에 따라 그에 해당하는 동작들이 실행되었고, 그 동작을 멈추기 위해서 stop 변수를 사용했다.
- 그러나 800줄의 경우, SW1을 누르는 횟수에 따라 mode 변환이 진행되었다. Model일 때를 진행하기 위해, “SW1 == 1 && SW3_cnt % 2 == 0 && SW4_cnt % 2 == 0”와 같이 SW@_cnt의 값에 각각 해당하는 동작들을 설정한 뒤 “1과 0”을 대입하여 On, Off를 진행하였다.
- 따라서 400줄과 같이 별도의 stop을 위한 전역변수는 설정하지 않았다.

| 코드 1. 400줄 | 코드 2. 800줄 |
|---|--|
| <pre> if (stop1 % 2 == 1) { EPwm5Regs.AQCSFRC.bit.CSFA = 1; } else if (stop1 % 2 == 0) { EPwm5Regs.AQCSFRC.bit.CSFA = 3; } </pre> | <pre> 500 PwmDutyRatioA = ((float32)ADC_value01 / (float32)65535) * (float32)0.475; 501 502 EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) / 2; 503 504 EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA); 505 506 Loop_cnt++; 507 508 BackTicker++; 509 SW1_cnt = 0; 510 511 </pre> |

| |
|--|
| <pre> if (SW1 == 1 && SW3_cnt % 2 == 0 && SW4_cnt % 2 == 0) { EPwm5Regs.DBFED = 0; /* 1usec, Falling Edge Delay */ EPwm5Regs.DBRED = 0; /* 1usec, Rising Edge Delay */ FallingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBFED; RisingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBRED; PWM_CARRIER = 20E3; PwmDutyRatioA = ((float32)ADC_value01 / (float32)65535) * (float32)0.475; EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) / 2; EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * PwmDutyRatioA; EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA); Loop_cnt++; BackTicker++; SW1_cnt = 0; } </pre> |
|--|

- if (SW1 == 1 && SW3_cnt % 2 == 0 && SW4_cnt % 2 == 0)

버튼을 한 번 누른 것을 의미하기 때문에 SW1=1이라고 설정하였고, EPWM 5A, 5B를 기본설정에서 사용한다고 하였기 때문에 SW3와 SW4의 2에 대한 나머지가 0이라고 설정하였다.

- PwmDutyRatioA = ((float32)ADC_value01 / (float32)65535) * (float32)0.475;

- float32 : IEEE-754 32비트 단정밀도 부동소수점, 7자리 정밀도 보장

- uint16 : 부호 없는 16비트, 2바이트 정수

코드 1과 마찬가지로 듀티 비를 ADC_value01 값에 따라 달라지는 변수로 설정한다. 여기서 ADC_value01은 0V에서 0을 갖고, 3V에서 최댓값인 65536을 가진다. 프로젝트에서는 최댓값이 PWM 주기 절반의 95%로 설정하기 위해, 0.475/65536을 곱해주었다.

| 코드 1. 400줄 | 코드 2. 800줄 |
|---|---|
| <pre> EPwm5Regs.TBPRD = (TBCLK / PwmCarrierFrequency) - 1; PwmDutyRatio = ADC_value01 * 0.475 / 65536; EPwm5Regs.CMPA.half.CMPA = (uint16)((EPwm5Regs.TBPRD + 1) * EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatio); /* Setup Counter Mode and Clock */ </pre> | <pre> PwmDutyRatioA = ((float32)ADC_value01 / (float32)65535) * (float32)0.475; EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) / 2; </pre> |

- EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * PwmDutyRatioA;

ADC_Value01는 0V일 때 최솟값인 0을 갖고, 3V일 때는 최댓값인 65535의 값을 가진다.

이에 비례하는 duty비를 설정하였고, 최댓값이 PWM 주기 절반의 95%로 설정하기 위해 코드를 설정해주었다.

- EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA);

EPWM5B와 EPWM5A와 상보적으로 나타나야 하기 때문에 1에서 CMPA에서 설정한 Duty값을 빼준 값이 CMPB로 설정 된다.

- SW1_cnt = 0;

: if 문의 동작이 모두 진행된 이후에, '0'을 대입함으로써, SW1을 누른 횟수가 0으로 초기화 된다.

```

else if (SW1 == 1 && SW3_cnt % 2 == 1 && SW4_cnt % 2 == 0)
{
    EPwm5Regs.DBFED = 0;
    EPwm5Regs.DBRED = 0;
    FallingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBFED;
    RisingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBRED;
    PWM_CARRIER = 20E3;
    PwmDutyRatioA = ((float32)ADC_value01 / (float32)65535) * (float32)0.475;
    EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) / 2;
    EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA);
    Loop_cnt++;
    BackTicker++;
    SW1_cnt = 0;
}

```

- **else if (SW1 == 1 && SW3_cnt % 2 == 1 && SW4_cnt % 2 == 0)**

SW1이 한 번 눌리고, SW3만 눌리게 되면 SW3_cnt에 1이 입력되면 EPWM5A의 동작이 중단되고 EPWM5B만 동작한다. SW3을 다시 한 번 누를 경우 EPWM5A가 다시 동작해야 하기 때문에 SW3_cnt의 조건을 2로 나누 나머지로 설정하였다. 루프 내의 코드는 스위치 1만 눌렀을 때의 경우와 동일하다.

- **else if (SW1 == 1 && SW3_cnt % 2 == 0 && SW4_cnt % 2 == 1)**

위의 경우와 반대로 SW1이 한 번 눌리고, SW4만 눌리게 되면 SW4_cnt에 1이 입력되기 때문에 EPWM5A가 동작한다. 루프 내의 코드는 스위치 1만 눌렀을 때의 경우와 동일하다.

```

else if (SW3_cnt % 2 == 1 && SW4_cnt % 2 == 1)
{
    EPwm5Regs.CMPA.half.CMPA = 0;
    EPwm5Regs.CMPB = 5000;
    Loop_cnt++;

    BackTicker++;
}

```

- **else if (SW3_cnt % 2 == 1 && SW4_cnt % 2 == 1)**

EPwm5Regs.CMPA.half.CMPA = 0; , EPwm5Regs.CMPB = 5000;

스위치 3과 스위치 4가 둘 다 눌렀을 경우, 동시에 On 되는 구간이 발생하지 않도록 하기 위해서 CMPA에는 0을 대입하고, CMPB에는 5000을 대입하였다. 따라서 EPWM5A와 EPWM5B는 상보적인 동작이 가능하다.

```

else if (SW1 == 2)
{
    PWM_CARRIER = 100E3 + 200E3 * (float32)ADC_value01 / (float32)65535;
    EPwm5Regs.TBPRD = (TBCLK / PWM_CARRIER) / 4;
    EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * PwmDutyRatioA;
    EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA);
    Loop_cnt++;
    BackTicker++;
    SW1_cnt = 0;
}

```

- **else if (SW1 == 2)**

Mode 2는 스위치 1을 두 번 클릭하여 동작이 가능하다. 이 모드에서는 가변저항 A의 변경에 따라 PWM 주파수가 변화한다. SW2를 사용하지 않고, A를 2번 눌렀다는 것을 “SW1 == 2”로 표현하였다.

- $EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * PwmDutyRatioA;$
- $EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - PwmDutyRatioA);$

EPWM 5A, 5B를 상보적으로 동작한다고 하였기 때문에, 위와 같이 설정하였다.

가장 중요한 추가설정 사항인 “EPWM 5A와 5B는 상보적으로 동작하며, 가변 저항 A의 변경에 따라 PWM의 주파수는 최소 100kHz, 최대 300kHz로 가변”을 구현하기 위해서, 아래와 같은 코드를 사용하였다.

- $PWM_CARRIER = 100E3 + 200E3 * (float32)ADC_value01 / (float32)65535;$

이때 100E3는 최소 주파수인 100kHz를 의미하고, “**200E3 * (float32)ADC_value01 / (float32)65535;**”는 가변 저항 A 값의 변화에 따른 주파수 값을 설정하기 위해, 사용하였다. ADC_value01이 최댓값인 65535를 갖게 되면 PWM_CARRIER는 최대 주파수인 300kHz가 된다.

마지막으로 스위치 1을 세 번 클릭한 경우를 나타내주기 위해서, **else if (SW1 >= 3)**를 사용하였다.

```

else if (SW1 >= 3)
{
    EPwm5Regs.DBFED = 75 * (float32)ADC_value01 / (float32)65535;
    EPwm5Regs.DBRED = 75 * (float32)ADC_value01 / (float32)65535;
    FallingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBFED;
    RisingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBRED;
    EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * 0.5;
    EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - 0.5);
    Loop_cnt++;
    BackTicker++;
    SW1_cnt = 0;
}

```

Mode 3는 스위치 1을 세 번 클릭하여 동작이 가능하다. 이 모드에서는 가변저항 A의 변경에 따른 데드 타임 변화를 알아볼 수 있다.

- $EPwm5Regs.DBFED = 75 * (float32)ADC_value01 / (float32)65535;$
- $FallingEdgeDelay = (1.0 / TBCLK) * EPwm5Regs.DBFED;$

데드타임을 0ns부터 500ns까지 조정할 수 있도록 설정하기 위해서, 위와 같은 코드를 사용하였다.
예를 들어 설명하기 위해, “Case1_가변저항 A 최대_Expression”을 사용하였다.

| Expression | Type | Value | Address |
|--------------------|--------------|--------------|-----------------|
| ⇒ FallingEdgeDelay | float | 4.933333e-07 | 0x0000C00C@D... |
| ⇒ RisingEdgeDelay | float | 4.933333e-07 | 0x0000C012@D... |
| ⇒ ADC_value01 | unsigned int | 65520 | 0x0000C00A@D... |
| ⇒ PwmCarrierFrequ | float | 150635.5 | 0x0000C016@D... |
| ⇒ PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... |
| ⇒ usec_delay | float | 20000.0 | 0x0000C010@D... |

$$EPwm5Regs.DBFED : 75 * \frac{65520}{65535} = 75 * \frac{(float32)ADCvalue01}{(float32)65535} = 74.983$$

$$FallingEdgeDelay : \frac{1}{150 * 10^6} * 74.938 = 4.9959 * 10^{-7} \simeq 500ns$$

ADC_Value01값이 65520으로 최댓값을 가지게 되면, Epwm5Regs.DBFED는 74.983을 가지게 되고, 따라서 FallingEdgeDelay의 값은 500ns값이 된다.

프로젝트 설명서에 있는 것처럼 데드타임을 300ns로 조절하기 위해서는,

$$FallingEdgeDelay : \frac{1}{150 * 10^6} * x = 2.9959 * 10^{-7} \simeq 300ns, \quad x = 44.9385 \simeq 45 \text{으로 DBFED가 45를 갖게 되고,}$$

$$EPwm5Regs.DBFED : 75 * \frac{y}{65535} = 45, \quad y = 39321 \text{를 통해 ADC_value01 값을 구할 수 있다. 즉, 500ns를 출력}$$

하는 Value01의 값인 65520값과 300ns를 출력하는 Value01의 값인 39321의 비율은 $\frac{39321}{65520} = 0.6$ 이 나왔으며,
이를 실험 5장에서 했던 계산 방식을 이용하면 아래와 같이 입력 전압을 구할 수 있다.

$$39321 * \frac{3}{4096} * \frac{1}{16} = 1.799 V \simeq 1.80 V$$

- $EPwm5Regs.CMPA.half.CMPA = (EPwm5Regs.TBPRD + 1) * 0.5;$
- $EPwm5Regs.CMPB = (EPwm5Regs.TBPRD + 1) * (1 - 0.5);$

앞서 언급한 것과 같은 이유로 EPWM 5A, 5B를 상보적으로 동작시키기 위해 위와 같이 설정하였다.


```

        if (SW3_cnt % 2 == 1)
        {
            EPwm5Regs.CMPA.half.CMPA = 0;
        }

        if (SW4_cnt % 2 == 1)
        {
            EPwm5Regs.CMPB = 5000;
        }
    }

```

앞선 그림에서 스위치에 따라 동작하는 것들이 다르다고 하였는데, SW3 (버튼 C)가 홀수 번 눌리는 경우, “EPwm5Regs.CMPA.half.CMPA = 0;”을 통해 EPWM5A의 동작이 중단되는 것을 설정하였고, “%”라는 나머지를 나타내는 연산자를 통해, 짝수 번 클릭하면 0의 값을 갖기 때문에 EPWM5A가 다시 동작한다.

마찬가지로, SW4 (버튼 D)가 홀수 번 눌리는 경우, “EPwm5Regs.CMPB = 5000;”을 통해, EPWM5B의 동작이 중단되는 것을 나타내었고, 짝수 번 클릭하면 다시 EPWM5B가 동작하게 된다.

```

void InitEPwm5Module(void)
{
    /* Setup Counter Mode and Clock */
    EPwm5Regs.TBCTL.bit.CTRMODE = 2;    /* Count Up (Asymmetric) */
    EPwm5Regs.TBCTL.bit.HSPCLKDIV = 0;    /* TBCLK = SYSCLKOUT / (HSPCLKDIV * CLKDIV) = 150MHz */
    EPwm5Regs.TBCTL.bit.CLKDIV = 0;
}

```

코드 1과 달리 코드 2에서는 위 함수에서 “EPwm5Regs.TBCTL.bit.CTRMODE = 2;”를 사용하였다.

* TBCTL : Time-Base Control Register

| | | |
|-----|---------|---|
| 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: |
| | 00 | Up-count mode |
| | 01 | Down-count mode |
| | 10 | Up-down-count mode |
| | 11 | Stop-freeze counter operation (default on reset) |

CTRMODE에 ‘2’ 값을 입력함으로써 Up-down-count mode로 동작한다.

- **Up-down-count mode** : 타임베이스 카운터는 0에서 시작하여 주기(TBPRD) 값에 도달할 때까지 증가한다. 주기 값에 도달하면 타임베이스 카운터는 0에 도달할 때까지 감소한다. 이 지점에서 카운터가 패턴을 반복하고 증가하기 시작한다.

```

/* Set Dead-time */
EPwm5Regs.DBCTL.bit.IN_MODE = 2;
EPwm5Regs.DBCTL.bit.OUT_MODE = 3;
EPwm5Regs.DBCTL.bit.POLSEL = 2;
EPwm5Regs.DBFED = 450;
EPwm5Regs.DBRED = 450;

```

코드 2의 마지막 부분에서는 Dead-time을 별도로 설정하였다. 이 설정한 값은 코드 1에서의 mode 1과 같은 조건으로 값을 설정하였다.

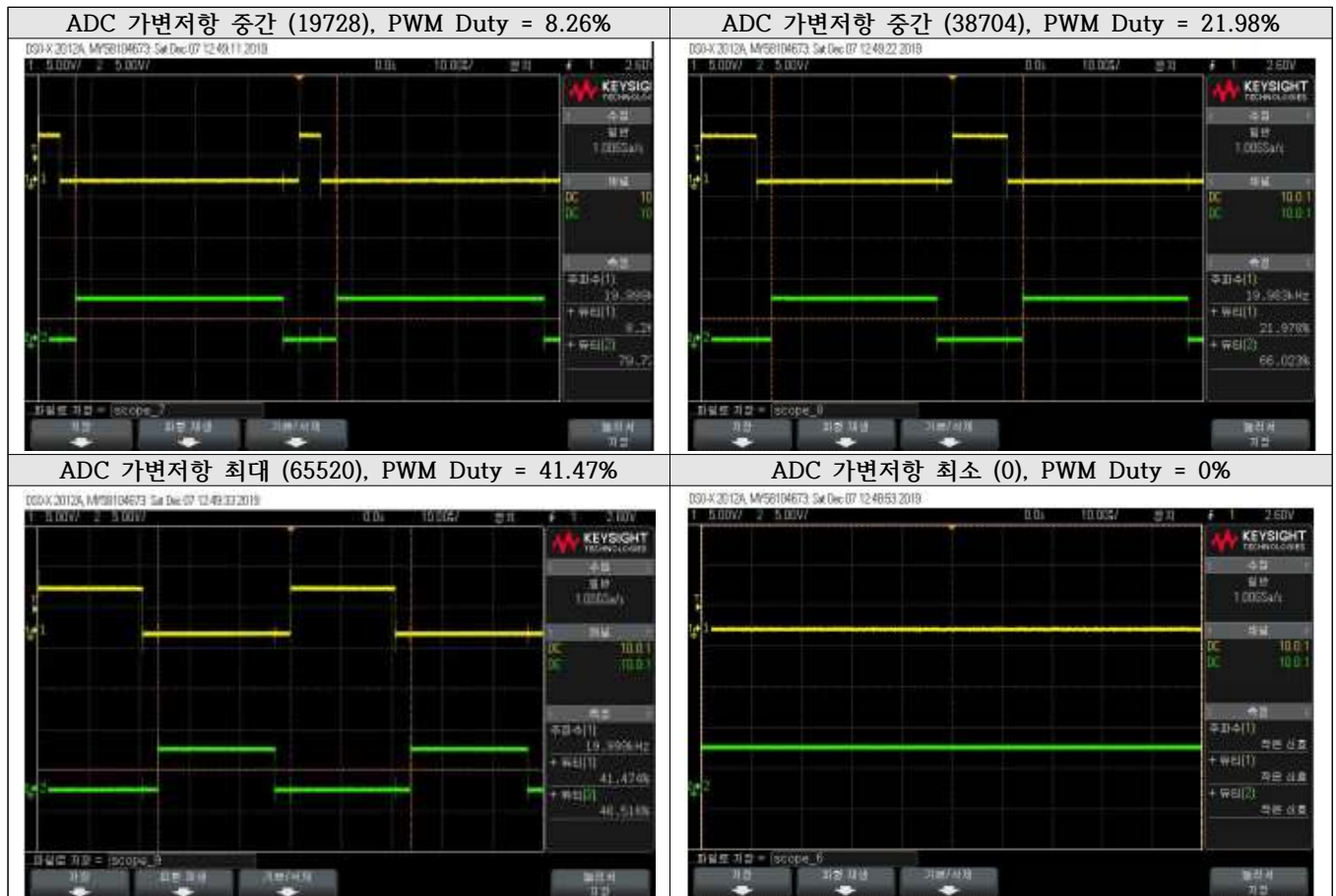
2. 실험 결과

결과 1. 버튼 A 한 번 클릭

• 실험 목표: 가변저항 A의 변화에 따라 PWM duty가 변화한다.

• 실험 분석:

1) Case 1 (코드 2. 800줄)



ADC 가변저항의 값에 따라 PWM Duty가 변하는 것을 확인 할 수 있다. 코드에서 PWM의 듀티 비는 아래 식과 같이 설정해주었다.

$$\text{PwmDutyRatio} = \text{ADC_value01} * (0.475 / 65535)$$

실제로 위 코드대로 동작하는지 확인해 보기 위해 계산을 해보면 다음과 같다.

1) ADC 가변저항 중간 (19728): $19728 * \frac{0.475}{65535} = 0.1429$ 즉 14.3%

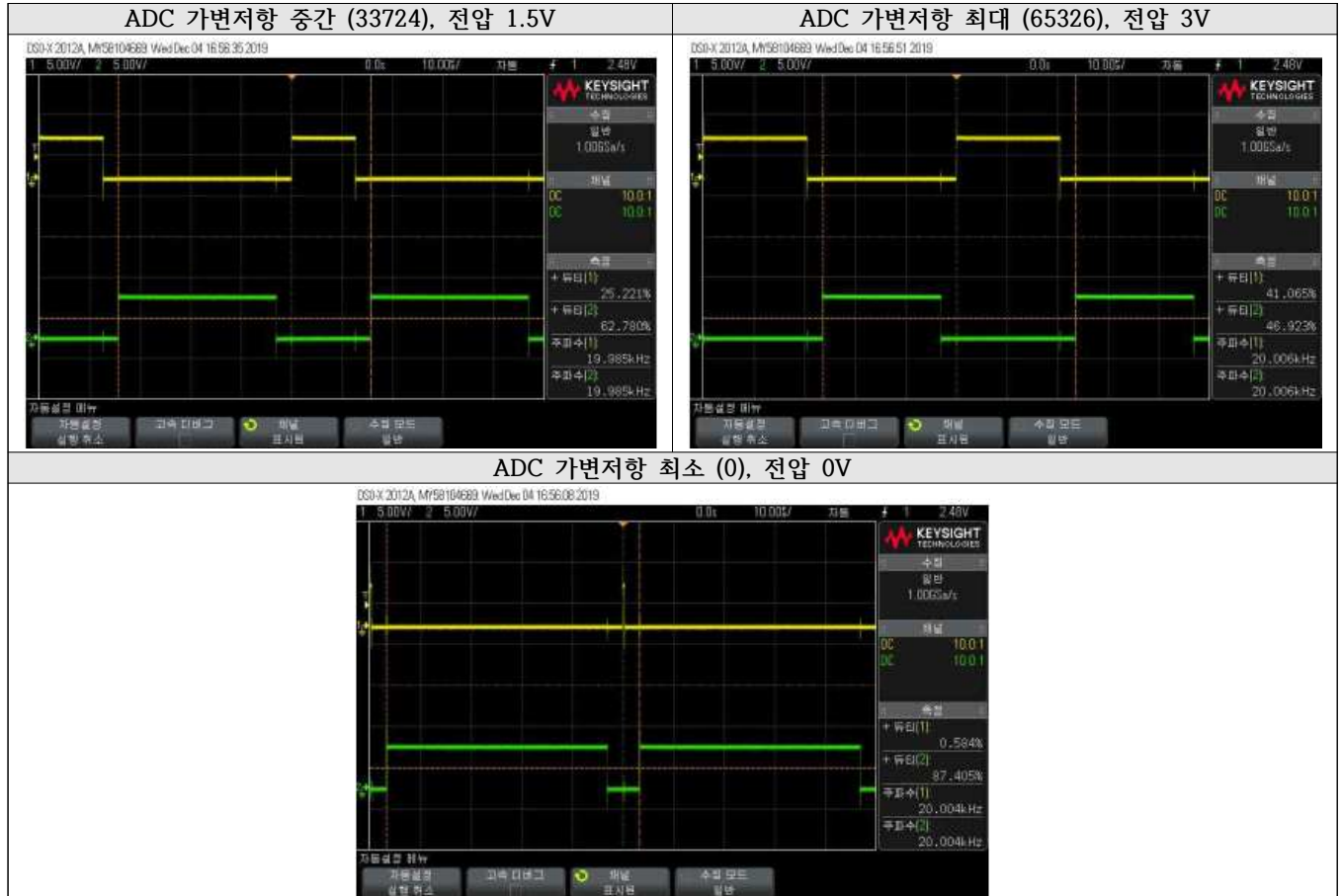
2) ADC 가변저항 중간 (38704): $38704 * \frac{0.475}{65535} = 0.2805$ 즉 28.1%

3) ADC 가변저항 최대 (65520): $65520 * \frac{0.475}{65535} = 0.4749$ 즉 47.5%

4) ADC 가변저항 최소 (0): $0 * \frac{0.475}{65535} = 0.00$ 즉 0%

위 결과를 살펴보면 이론값과 실험값 차이에 약간의 오차를 갖지만, 가변저항 측 전압이 증가할수록 PWM duty가 증가한다는 경향성은 일치함을 확인할 수 있었다. 또 가변저항 A에 의한 전압의 최대값에서 PWM duty는 PWM 주기 절반의 95%와 가까운 값인 82.94%의 값을 가짐을 확인하였다.

2) Case 2 (코드 1. 400줄)



Case 1과 같이 400줄의 코드에서도 PWM 듀티 비를 아래와 같이 정의했다.

$$\text{PwmDutyRatio} = \text{ADC_value01} * (0.475 / 65536)$$

실험을 통해 얻는 ADC_value01 값을 이용하여 듀티를 구할 수 있다.

1) ADC 가변저항 중간 (33724): $33724 * \frac{0.475}{65536} = 0.2444$ 즉 24.4%

2) ADC 가변저항 최대 (65326): $65326 * \frac{0.475}{65536} = 0.4734$ 즉 47.3%

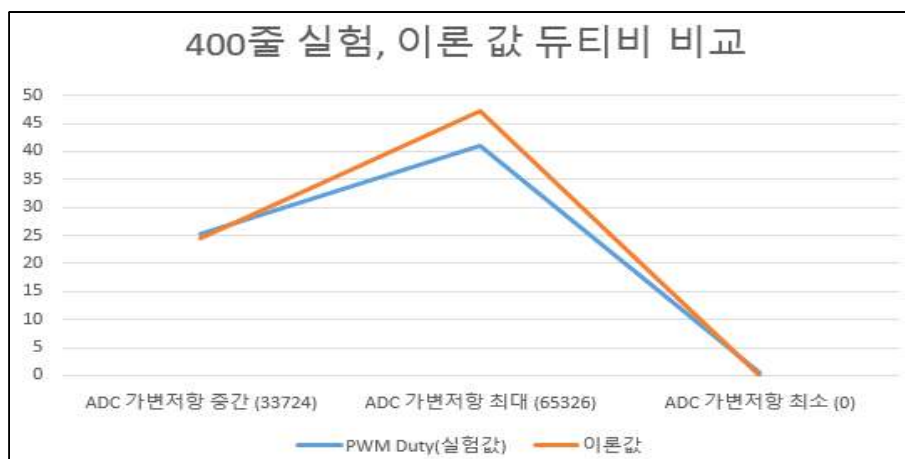
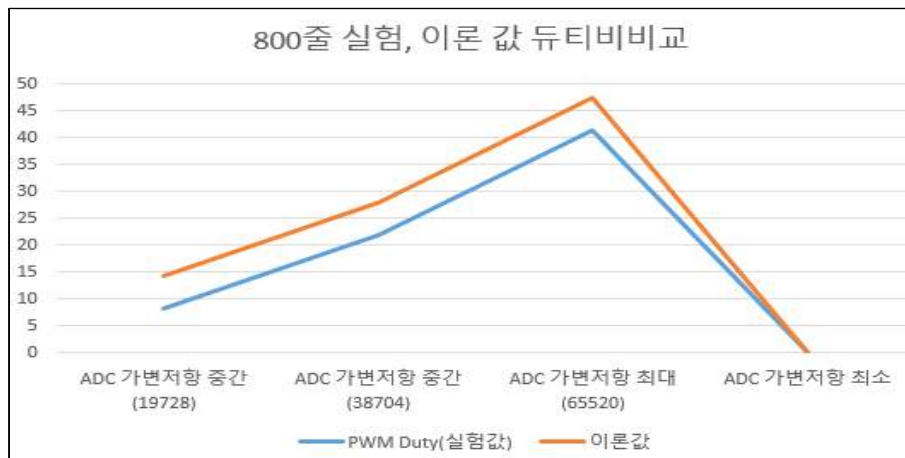
3) ADC 가변저항 최소 (0): $0 * \frac{0.475}{65536} = 0.00$ 즉 0%

<800줄>

| | ADC 가변저항 중간 (19728) | ADC 가변저항 중간 (38704) | ADC 가변저항 최대 (65520) | ADC 가변저항 최소 |
|------------------|------------------------|------------------------|------------------------|-------------|
| PWM Duty(실험값) | 8.26 | 21.98 | 41.47 | 0 |
| 이론값 | 14.3 | 28.1 | 47.5 | 0 |

<400줄>

| | ADC 가변저항 중간 (33724) | ADC 가변저항 최대 (65326) | ADC 가변저항 최소 (0) |
|---------------|------------------------|------------------------|-----------------|
| PWM Duty(실험값) | 25.221 | 41.065 | 0.584 |
| 이론값 | 24.4 | 47.3 | 0 |



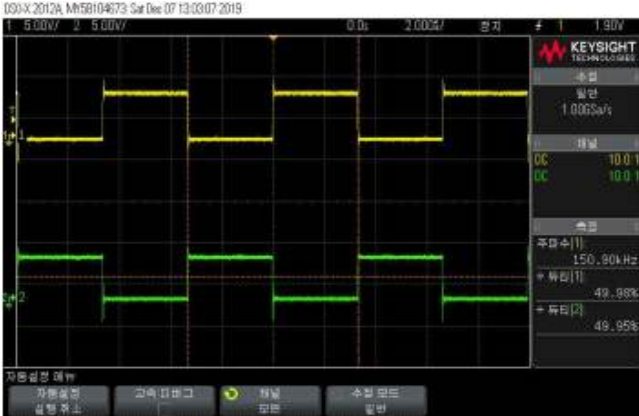
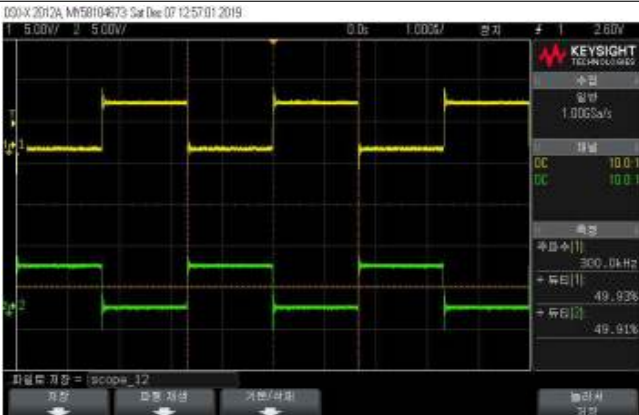

위 결과 역시 Case 1과 같이 이론값과 실험값 차이에 약간의 오차를 갖지만, 가변저항 측 전압이 증가할수록 PWM duty가 증가한다는 경향성은 일치함을 확인할 수 있다.

결과 2. 버튼 A 두 번 클릭

• 실험 목표: 가변저항 A의 변화에 따라 PWM 주파수가 변화한다.

• 실험 분석:

1) Case 1 (코드 2. 800줄)

| <div>주파수 중간 (150kHz)</div> <div></div> | <div>주파수 중간 (150kHz)</div> <table><thead><tr><th>Expression</th><th>Type</th><th>Value</th><th>Address</th></tr></thead><tbody><tr><td>SW4_cnt</td><td>unsigned int</td><td>0</td><td>0x0000C008@D...</td></tr><tr><td>SW3_flag</td><td>unsigned int</td><td>1</td><td>0x0000C005@D...</td></tr><tr><td>SW4_flag</td><td>unsigned int</td><td>1</td><td>0x0000C006@D...</td></tr><tr><td>MODE</td><td>unsigned int</td><td>0</td><td>0x0000C002@D...</td></tr><tr><td>ADC_value01</td><td>unsigned int</td><td>16752</td><td>0x0000C00A@D...</td></tr><tr><td>PwmCarrierFrequ</td><td>float</td><td>150537.9</td><td>0x0000C016@D...</td></tr><tr><td>PwmDutyRatioA</td><td>float</td><td>0.5</td><td>0x0000C00E@D...</td></tr><tr><td>usec_delay</td><td>float</td><td>20000.0</td><td>0x0000C010@D...</td></tr><tr><td colspan="4">Add new expressi</td></tr></tbody></table> | Expression | Type | Value | Address | SW4_cnt | unsigned int | 0 | 0x0000C008@D... | SW3_flag | unsigned int | 1 | 0x0000C005@D... | SW4_flag | unsigned int | 1 | 0x0000C006@D... | MODE | unsigned int | 0 | 0x0000C002@D... | ADC_value01 | unsigned int | 16752 | 0x0000C00A@D... | PwmCarrierFrequ | float | 150537.9 | 0x0000C016@D... | PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | usec_delay | float | 20000.0 | 0x0000C010@D... | Add new expressi | | | |
|---|---|------------|-----------------|-------|---------|---------|--------------|---|-----------------|----------|--------------|---|-----------------|----------|--------------|---|-----------------|------|--------------|---|-----------------|-------------|--------------|-------|-----------------|-----------------|-------|----------|-----------------|---------------|-------|-----|-----------------|------------|-------|---------|-----------------|------------------|--|--|--|
| Expression | Type | Value | Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_cnt | unsigned int | 0 | 0x0000C008@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW3_flag | unsigned int | 1 | 0x0000C005@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_flag | unsigned int | 1 | 0x0000C006@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODE | unsigned int | 0 | 0x0000C002@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ADC_value01 | unsigned int | 16752 | 0x0000C00A@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmCarrierFrequ | float | 150537.9 | 0x0000C016@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| usec_delay | float | 20000.0 | 0x0000C010@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Add new expressi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <div>주파수 최대 (300kHz)</div> <div></div> | <div>주파수 최대 (300kHz)</div> <table><thead><tr><th>Expression</th><th>Type</th><th>Value</th><th>Address</th></tr></thead><tbody><tr><td>SW4_cnt</td><td>unsigned int</td><td>0</td><td>0x0000C008@D...</td></tr><tr><td>SW3_flag</td><td>unsigned int</td><td>1</td><td>0x0000C005@D...</td></tr><tr><td>SW4_flag</td><td>unsigned int</td><td>1</td><td>0x0000C006@D...</td></tr><tr><td>MODE</td><td>unsigned int</td><td>0</td><td>0x0000C002@D...</td></tr><tr><td>ADC_value01</td><td>unsigned int</td><td>65520</td><td>0x0000C00A@D...</td></tr><tr><td>PwmCarrierFrequ</td><td>float</td><td>299954.2</td><td>0x0000C016@D...</td></tr><tr><td>PwmDutyRatioA</td><td>float</td><td>0.5</td><td>0x0000C00E@D...</td></tr><tr><td>usec_delay</td><td>float</td><td>20000.0</td><td>0x0000C010@D...</td></tr><tr><td colspan="4">Add new expressi</td></tr></tbody></table> | Expression | Type | Value | Address | SW4_cnt | unsigned int | 0 | 0x0000C008@D... | SW3_flag | unsigned int | 1 | 0x0000C005@D... | SW4_flag | unsigned int | 1 | 0x0000C006@D... | MODE | unsigned int | 0 | 0x0000C002@D... | ADC_value01 | unsigned int | 65520 | 0x0000C00A@D... | PwmCarrierFrequ | float | 299954.2 | 0x0000C016@D... | PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | usec_delay | float | 20000.0 | 0x0000C010@D... | Add new expressi | | | |
| Expression | Type | Value | Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_cnt | unsigned int | 0 | 0x0000C008@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW3_flag | unsigned int | 1 | 0x0000C005@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_flag | unsigned int | 1 | 0x0000C006@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODE | unsigned int | 0 | 0x0000C002@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ADC_value01 | unsigned int | 65520 | 0x0000C00A@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmCarrierFrequ | float | 299954.2 | 0x0000C016@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| usec_delay | float | 20000.0 | 0x0000C010@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Add new expressi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <div>주파수 최소 (100kHz)</div> <div></div> | <div>주파수 최소 (100kHz)</div> <table><thead><tr><th>Expression</th><th>Type</th><th>Value</th><th>Address</th></tr></thead><tbody><tr><td>SW4_cnt</td><td>unsigned int</td><td>0</td><td>0x0000C008@D...</td></tr><tr><td>SW3_flag</td><td>unsigned int</td><td>1</td><td>0x0000C005@D...</td></tr><tr><td>SW4_flag</td><td>unsigned int</td><td>1</td><td>0x0000C006@D...</td></tr><tr><td>MODE</td><td>unsigned int</td><td>0</td><td>0x0000C002@D...</td></tr><tr><td>ADC_value01</td><td>unsigned int</td><td>128</td><td>0x0000C00A@D...</td></tr><tr><td>PwmCarrierFrequ</td><td>float</td><td>100146.5</td><td>0x0000C016@D...</td></tr><tr><td>PwmDutyRatioA</td><td>float</td><td>0.5</td><td>0x0000C00E@D...</td></tr><tr><td>usec_delay</td><td>float</td><td>20000.0</td><td>0x0000C010@D...</td></tr><tr><td colspan="4">Add new expressi</td></tr></tbody></table> | Expression | Type | Value | Address | SW4_cnt | unsigned int | 0 | 0x0000C008@D... | SW3_flag | unsigned int | 1 | 0x0000C005@D... | SW4_flag | unsigned int | 1 | 0x0000C006@D... | MODE | unsigned int | 0 | 0x0000C002@D... | ADC_value01 | unsigned int | 128 | 0x0000C00A@D... | PwmCarrierFrequ | float | 100146.5 | 0x0000C016@D... | PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | usec_delay | float | 20000.0 | 0x0000C010@D... | Add new expressi | | | |
| Expression | Type | Value | Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_cnt | unsigned int | 0 | 0x0000C008@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW3_flag | unsigned int | 1 | 0x0000C005@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SW4_flag | unsigned int | 1 | 0x0000C006@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| MODE | unsigned int | 0 | 0x0000C002@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ADC_value01 | unsigned int | 128 | 0x0000C00A@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmCarrierFrequ | float | 100146.5 | 0x0000C016@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| usec_delay | float | 20000.0 | 0x0000C010@D... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Add new expressi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

위 결과를 통해 ADC 가변저항의 값에 따라 PWM 주파수가 변하는 것을 확인 할 수 있다. 코드에서 PWM의 주파수는 아래 식과 같이 설정해주었다.

$$\text{PwmCarrier} = 100\text{kHz} + 200\text{kHz} * (\text{ADC_value01} / 65535)$$

실제로 위 코드대로 동작하는지 확인해 보기 위해, 실험으로 얻은 ADC_value값을 위 식에 대입하여 PWM 주파수를 계산하면 아래와 같다.

- 1) 주파수 중간 (150kHz) : $100k + 200k * \frac{16752}{65535} = 151.12$ 즉 151kHz
- 2) 주파수 최대 (300kHz): $100k + 200k * \frac{65520}{65535} = 299.95$ 즉 300kHz
- 3) 주파수 최소 (100kHz): $100k + 200k * \frac{128}{65535} = 100.39$ 즉 100kHz

위 결과는 실제 측정된 주파수와 오차가 매우 작기 때문에 실험의 목표대로 가변저항에 따라 PWM 주파수가 변한다는 것을 확인할 수 있었다.

2) Case 2 (코드 1. 400줄)



Case 2역시 Case 1과 마찬가지로 ADC 가변저항의 값에 따라 PWM 주파수가 변하는 것을 확인 할 수 있다. 코드에서 PWM의 주파수는 아래 식과 같이 설정해주었다.

$$\text{PwmCarrier} = 100\text{kHz} + 200\text{kHz} * (\text{ADC_value01} / 65536)$$

실제로 위 코드대로 동작하는지 확인해 보기 위한 계산을 하기에 앞서, 먼저 입력 전압으로부터 ADC_value 값을 계산해야한다. Case1의 값을 참고하면 주파수가 최대인 300kHz일 때는 ADC_value값이 65520이고 최소인 100kHz일 때는 128이기 때문에 중간인 150kHz일 때는 그 중간 값인 32824를 가짐을 유추할 수 있다. 이 값을 통해 주파수 값을 계산해보면 아래와 같다.

- 1) 주파수 중간 (200kHz) : $100k + 200k * \frac{32824}{65536} = 200.17$ 즉 200kHz
- 2) 주파수 최대 (300kHz): $100k + 200k * \frac{65520}{65536} = 299.95$ 즉 300kHz
- 3) 주파수 최소 (100kHz): $100k + 200k * \frac{128}{65536} = 100.39$ 즉 100kHz

<800줄>

| | 주파수 최소 | 주파수 중간 | 주파수 중간 |
|-----|---------|---------|----------|
| 실험값 | 100.146 | 150.537 | 299.9542 |
| 이론값 | 100 | 151 | 300 |

<400줄>

| | 주파수 최소 | 주파수 중간 | 주파수 중간 |
|-----|--------|--------|--------|
| 실험값 | 100.53 | 203.5 | 299.8 |
| 이론값 | 100 | 200 | 300 |



위 결과는 실제 측정된 주파수와 오차가 매우 작기 때문에 실험의 목표대로 가변저항에 따라 PWM 주파수가 변한다는 것을 확인할 수 있었다.

결과 3. 버튼 A 세 번 클릭

• 실험 목표: 가변저항 A의 변화에 따라 데드 타임이 변화한다.

• 실험 분석:

1) Case 1 (코드 2. 800줄)

가변저항 A 최대

000-X 2013A, MS56104573, Set Dec 07 13 16 15 2019

KEYSIGHT TECHNOLOGIES

주입: 2.00V

일반: 1.000ns/div

채널: 2

0C: 10.01

0C: 10.01

주파수: 150.76kHz

+ 듀티: 42.43%

+ 듀티: 42.58%

ADC_value01: 65520

PwmCarrierFrequency: 150635.5

PwmDutyRatioA: 0.5

usec_delay: 20000.0

가변저항 A 최소

000-X 2013A, MS56104573, Set Dec 07 13 16 40 2019

KEYSIGHT TECHNOLOGIES

주입: 2.00V

일반: 1.000ns/div

채널: 2

0C: 10.01

0C: 10.01

주파수: 150.76kHz

+ 듀티: 49.92%

+ 듀티: 50.01%

ADC_value01: 0

PwmCarrierFrequency: 150635.5

PwmDutyRatioA: 0.5

usec_delay: 20000.0

가변저항 A 최대

| Expression | Type | Value | Address |
|---------------------|--------------|--------------|-----------------|
| FallingEdgeDelay | float | 4.933333e-07 | 0x0000C00C@D... |
| RisingEdgeDelay | float | 4.933333e-07 | 0x0000C012@D... |
| ADC_value01 | unsigned int | 65520 | 0x0000C00A@D... |
| PwmCarrierFrequency | float | 150635.5 | 0x0000C016@D... |
| PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... |
| usec_delay | float | 20000.0 | 0x0000C010@D... |

가변저항 A 최소

| Expression | Type | Value | Address |
|---------------------|--------------|----------|-----------------|
| FallingEdgeDelay | float | 0.0 | 0x0000C00C@D... |
| RisingEdgeDelay | float | 0.0 | 0x0000C012@D... |
| ADC_value01 | unsigned int | 0 | 0x0000C00A@D... |
| PwmCarrierFrequency | float | 150635.5 | 0x0000C016@D... |
| PwmDutyRatioA | float | 0.5 | 0x0000C00E@D... |
| usec_delay | float | 20000.0 | 0x0000C010@D... |

위 결과를 통해 가변저항이 변함에 따라 인가되는 데드타임이 변하는 것을 알 수 있다. 코드에서 데드타임은 아래의 식과 같이 구할 수 있었다.

$$\text{DBFED} = 75 * (\text{ADC_value01} / 65535)$$

$$\text{FallingEdgeDelay} = (1 / \text{TBCLK}) * \text{DBFED}$$

$$\text{DBRED} = 75 * (\text{ADC_value01} / 65535)$$

$$\text{RisingEdgeDelay} = (1 / \text{TBCLK}) * \text{DBRED}$$

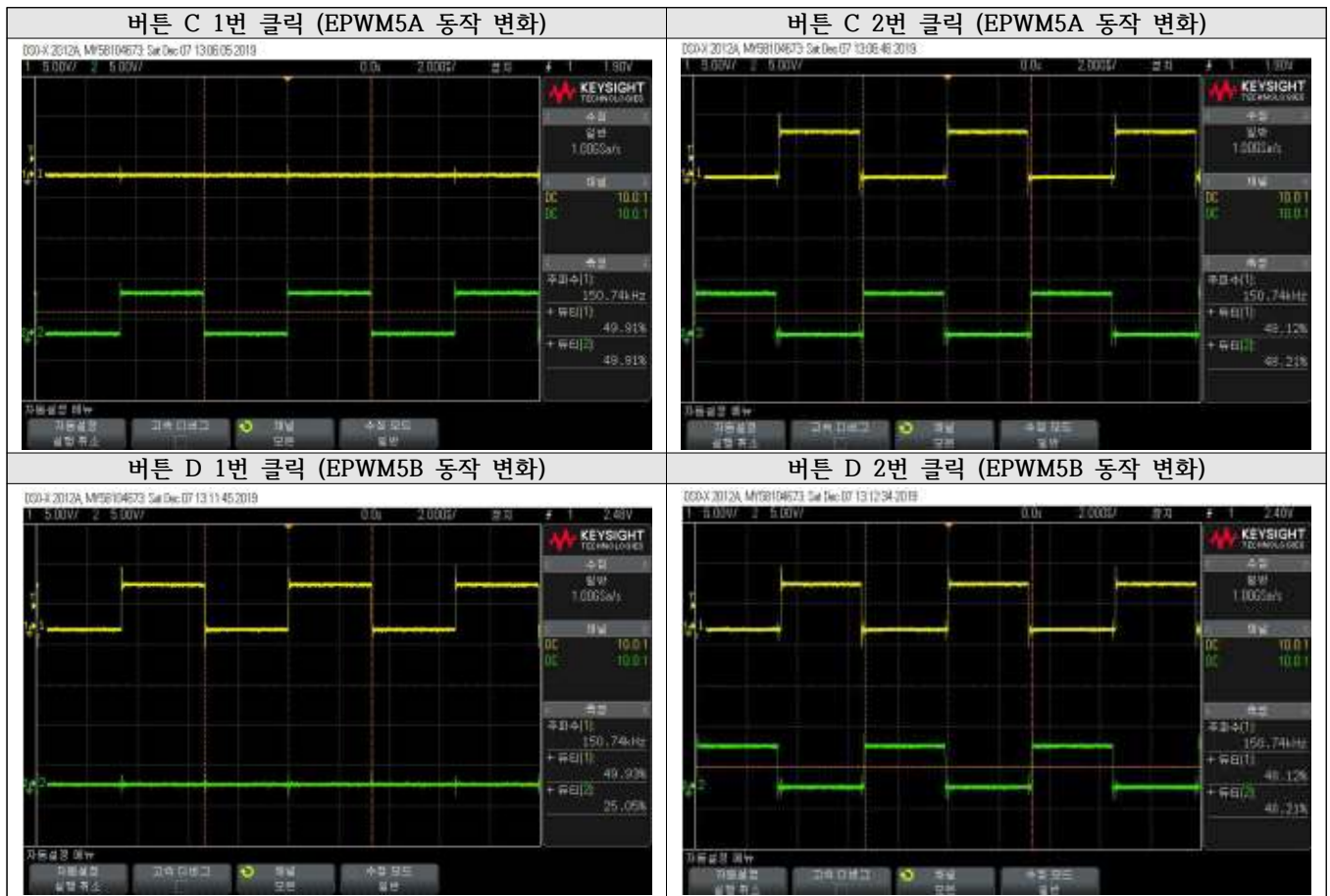
실제로 위 코드대로 동작하는지 확인해 보기 위해, 실험으로 얻은 ADC_value값을 위 식에 대입하여 데드타임을 계산하면 아래와 같다.

1) 가변저항 A 최대

$$\text{DBFED} = 75 * \frac{65520}{65535} = 74.98, \text{ FallingEdgeDelay} = \frac{1}{150 * 10^6} * 74.98 = 4.999 * 10^{-7} \text{ (Rising도 동일)}$$

2) 가변저항 A 최소

$$\text{DBFED} = 75 * \frac{0}{65535} = 0, \text{ FallingEdgeDelay} = \frac{1}{150 * 10^6} * 0 = 0 \text{ (Rising도 동일)}$$



• else if (SW1 == 1 && SW3_cnt % 2 == 1 && SW4_cnt % 2 == 0)

SW1이 한 번 눌리고, SW3만 눌리게 되면 SW3_cnt에 1이 입력되면 EPWM5A의 동작이 중단되고 EPWM5B만 동작한다. SW3을 다시 한 번 누를 경우 EPWM5A가 다시 동작해야 하기 때문에 SW3_cnt의 조건을 2로 나누 나머지로 설정하였다.

• else if (SW1 == 1 && SW3_cnt % 2 == 0 && SW4_cnt % 2 == 1)

위의 경우와 반대로 SW1이 한 번 눌리고, SW4만 눌리게 되면 SW4_cnt에 1이 입력되기 때문에 EPWM5B의 동작이 중단되고 EPWM5A만 동작한다. SW4를 다시 한 번 누를 경우 EPWM5B가 다시 동작해야 하기 때문에 SW4_cnt의 조건을 2로 나누 나머지로 설정하였다.

따라서 버튼 C와 D를 홀수 번 클릭하면 각각 EPWM5A와 EPWM5B가 중단되고, 짝수 번 클릭하면 EPWM5A와 EPWM5B가 시작되어 원상태로 돌아가는 것을 확인할 수 있다.

2) Case 2 (코드 1. 400줄)



Case 2역시 Case 1과 마찬가지로 ADC 가변저항의 값에 따라 데드타임이 변하는 것을 확인 할 수 있다. 코드에서 데드타임은 아래의 식과 같이 구할 수 있었다.

$$\text{deadT} = 75 * (\text{ADC_value01} / 65536)$$

$$\text{DBFED} = \text{deadT}$$

$$\text{FallingEdgeDelay} = (1 / \text{TBCLK}) * \text{DBFED}$$

$$\text{DBRED} = 0$$

$$\text{RisingEdgeDelay} = (1 / \text{TBCLK}) * \text{DBRED}$$

실제로 위 코드대로 동작하는지 확인해 보기 위한 계산을 하기에 앞서, 먼저 입력 전압으로부터 ADC_value 값을 계산해야한다. Case1의 값을 참고하면 가변저항이 최대인 3V일 때는 ADC_value값이 65520이고 최소인 0V일 때는 0이기 때문에 중간인 1.50V일 때는 그 중간 값인 32760를 가짐을 유추할 수 있다. 이 값을 통해 데드타임 값을 계산해보면 아래와 같다.

$$1) \text{가변저항 측 전압 } 0V : \text{deadT} = 75 * \frac{0}{65536} = 0, \text{FallingEdgeDelay} = \frac{1}{150 * 10^6} * 0 = 0\text{ns}$$

$$2) \text{가변저항 측 전압 } 1.5V : \text{deadT} = 75 * \frac{32760}{65536} = 37.49, \text{FallingEdgeDelay} = \frac{1}{150 * 10^6} * 37.49 = 250\text{ns}$$

$$3) \text{가변저항 측 전압 } 3V : \text{deadT} = 75 * \frac{65520}{65536} = 74.98, \text{FallingEdgeDelay} = \frac{1}{150 * 10^6} * 74.98 = 500\text{ns}$$

따라서 가변저항의 변화에 따라 데드타임이 변하는 것을 확인할 수 있었다. 추가로, 버튼 C와 D를 누르게 되면 EPWM5A와 EPWM5B가 모두 정지되는 파형을 관찰할 수 있다.

3. 실험 고찰

이번 프로젝트는 Delfino_EVM 보드의 ePWM 기능을 이용하여 PWM 파형을 출력하며 가변저항을 통해 PWM의 Duty, 주파수, 데드 타임을 변화시키는 세 가지 모드에 대한 실험을 진행했다. 보드 내 스위치를 통해 각각의 모드가 변경되며 각 모드의 목적은 다음과 같다. 모드 1은 가변저항 A의 변경에 따른 PWM Duty 변화를 살펴보는 것이고, 모드 2는 가변저항 A의 변경에 따른 PWM 주파수의 변화를 살펴본다. 모드 3은 가변저항 A의 변경에 따른 데드 타임 변화를 살펴본다. 각 모드마다 이동은 버튼 A를 누르는 횟수에 따라 결정되는데, 모드 1은 버튼 A를 한 번, 모드 2는 버튼 A를 두 번, 모드 3은 버튼 A를 세 번을 눌러 설정할 수 있다.

각 모드마다 목적은 다르지만 공통된 점은 가변저항의 값인 ADC_value에 따라 그 결과가 달라진다는 점이었다. ADC_value 값은 가변저항에 입력되는 0V부터 3V에 해당하는 전압을 0에서 65536의 디지털 값으로 받아들이는 것이다. 때문에 모든 과정에서 ADC_value 값을 65536으로 나눠주는 정규화하는 과정을 진행하였다.

모드 1은 ADC 가변저항의 값에 따라 PWM Duty가 변하는 것을 확인 할 수 있었다. 코드에서 PWM의 듀티비는 아래 식과 같이 설정해주었다.

$$\text{PwmDutyRatio} = \text{ADC_value01} * (0.475 / 65535)$$

실제로 위 코드대로 동작하는지 확인해 보기 위해 계산을 해봤을 때, 이론값과 실험값 차이에 약간의 오차가 있었지만, 가변저항 측 전압이 증가할수록 PWM duty가 증가한다는 경향성은 일치함을 확인할 수 있었다.

모드 2는 ADC 가변저항의 값에 따라 PWM 주파수가 변하는 것을 확인 할 수 있었다. 코드에서 PWM의 주파수는 아래 식과 같이 설정해주었다.

$$\text{PwmCarrier} = 100\text{kHz} + 200\text{kHz} * (\text{ADC_value01} / 65535)$$

실제로 위 코드대로 동작하는지 확인해 보기 위해, 실험으로 얻은 ADC_value값을 위 식에 대입하여 PWM 주파수를 계산을 했을 때, 실제 측정된 주파수와 오차가 매우 작기 때문에 실험의 목표대로 가변저항에 따라 PWM 주파수가 변한다는 것을 확인할 수 있었다.

모드 3은 가변저항이 변함에 따라 인가되는 데드타임이 변하는 것을 알 수 있었다. 코드에서 데드타임은 아래의 식과 같이 구할 수 있었다.

$$\begin{aligned}\text{DBFED} &= 75 * (\text{ADC_value01} / 65535) \\ \text{FallingEdgeDelay} &= (1 / \text{TBCLK}) * \text{DBFED} \\ \text{DBRED} &= 75 * (\text{ADC_value01} / 65535) \\ \text{RisingEdgeDelay} &= (1 / \text{TBCLK}) * \text{DBRED}\end{aligned}$$

실제로 위 코드대로 동작하는지 확인해 보기 위해, 실험으로 얻은 ADC_value값을 위 식에 대입하여 데드타임을 계산을 했을 때, 실험값과 계산값 사이에 오차가 매우 작기 때문에 실험의 목표대로 가변저항에 따라 데드타임이 변한다는 것을 확인할 수 있었다.

추가로 버튼 C와 D를 홀수 번 클릭하면 각각 EPWM5A와 EPWM5B가 중단되고, 짝수 번 클릭하면 EPWM5A와 EPWM5B가 시작되어 원상태로 돌아가는 것을 확인할 수 있다.

4. 참고문헌

- TMS320x2833x System Control and Interrupts Reference Guide (레지스터 정보)
- Schematic_Delfino_EVM_V101 : EMIF & McBSP Expansions 커넥터 핀 정의
- 데이터시트, 6.TMS320x2833x, 2823x Enhanced Pulse Width
- 김상훈, [모터제어], 북두, 2016, 397-403쪽