전자종합설계2 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: 2020.06.01.(월)

과목명: 융합캡스톤디자인2

교수명: 이교범 교수님

조 : 4 조

학 번: 201420820, 201623432

성 명: 이승복, 김희서

[중간 프로젝트]

1. 코드 분석

<코드 1. 300줄>

```
#include "DSP28x_Project.h"
//Device Headerfile and Examples Include File
interrupt void Xint3_isr(void);
interrupt void Xint4_isr(void);
interrupt void Xint5_isr(void);
interrupt void Xint6_isr(void);

void Printmode_1(void);
void Printmode_2(void);
void Printmode_3(void);
void Printmode_4(void);
void Printmode_5(void);
void Printmode_6(void);
void Printmode_stp(void);

Uint16 Loop_cnt;
Uint16 SW1_cnt, SW2_cnt, SW3_cnt, SW4_cnt;
Uint16 mode, state;
Uint16 i, j, num;
```

```
float32 usec_delay, usec_delay_1;
void main(void)
{

    DINT;

    InitSysCtrl();

    InitPieCtrl();
    IER = 0x0000;
    IFR = 0x0000;
    InitPieVectTable();

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

```
    EALLOW;
    GpioCtrlRegs.㉠GPBMUX1.bit.GPIO44 = 0;     // 핀 기능선택: GPIO44
    GpioCtrlRegs.GPBMUX1.bit.GPIO45 = 0;       // 핀 기능선택: GPIO45
    GpioCtrlRegs.GPBMUX1.bit.GPIO46 = 0;       // 핀 기능선택: GPIO46
    GpioCtrlRegs.GPBMUX1.bit.GPIO47 = 0;       // 핀 기능선택: GPIO47
    GpioCtrlRegs.㉡GPBDIR.bit.GPIO44 = 0;       // GPIO44 입출력 선택: Input
    GpioCtrlRegs.GPBDIR.bit.GPIO45 = 0;         // GPIO45 입출력 선택: Input
    GpioCtrlRegs.GPBDIR.bit.GPIO46 = 0;         // GPIO46 입출력 선택: Input
    GpioCtrlRegs.GPBDIR.bit.GPIO47 = 0;         // GPIO47 입출력 선택: Input
    EDIS;
```

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

Bit	Field	Value	Description
31:30	GPIO47		Configure this pin as:
		00	GPIO 47 - general purpose I/O 47 (default)
		01	Reserved
		10 or 11	XA7 - External interface (XINTF) address line 7 (O)
29:28	GPIO46		Configure this pin as:
		00	GPIO 46 - general purpose I/O 46 (default)
		01	Reserved
		10 or 11	XA6 - External interface (XINTF) address line 6 (O)
27:26	GPIO45		Configure this pin as:
		00	GPIO 45 - general purpose I/O 45 (default)
		01	Reserved
		10 or 11	XA5 - External interface (XINTF) address line 5 (O)
25:24	GPIO44		Configure this pin:
		00	GPIO 44 - general purpose I/O 44 (default)
		01	Reserved
		10 or 11	XA4 - External interface (XINTF) address line 4 (O)

㉠ GPBMUX: 핀의 역할을 정해주는 레지스터로, 0을 입력하면 핀을 GPIO로 사용하는 것이고, 1을 입력하면 핀을 GPIO가 아닌 다른 용도로 사용한다는 의미이다. 이번 예제에서는 모든 핀이 GPIO로 사용하기 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

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

```
EALLOW;
GpioCtrlRegs.GPCMUX1.all = 0x00000000;      // GPIO64-GPIO79, GPIO 기능으로 설정
GpioCtrlRegs.GPCDIR.all = 0x0000FFFF;       // GPI064-GPIO79, 출력으로 설정
EDIS;

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;
```

Table 4-15. GPIO Registers

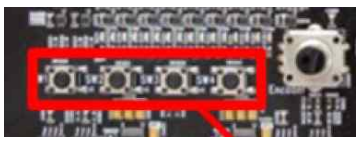
NAME	ADDRESS	SIZE (x16)	DESCRIPTION
GPIO CONTROL REGISTERS (EALLOW PROTECTED)			
GPACTRL	0x6F80	2	GPIO A Control Register (GPIO0 to 31)
GPAQSEL1	0x6F82	2	GPIO A Qualifier Select 1 Register (GPIO0 to 15)
GPAQSEL2	0x6F84	2	GPIO A Qualifier Select 2 Register (GPIO16 to 31)
GPAMUX1	0x6F86	2	GPIO A MUX 1 Register (GPIO0 to 15)
GPAMUX2	0x6F88	2	GPIO A MUX 2 Register (GPIO16 to 31)
GPADIR	0x6F8A	2	GPIO A Direction Register (GPIO0 to 31)
GPAPUD	0x6F8C	2	GPIO A Pull Up Disable Register (GPIO0 to 31)
Reserved	0x6F8E – 0x6F8F	2	
GPBCTRL	0x6F90	2	GPIO B Control Register (GPIO32 to 63)
GPBQSEL1	0x6F92	2	GPIO B Qualifier Select 1 Register (GPIO32 to 47)

GPBCTRL : GPIO40 ~ GPIO47 사용. GPBQSEL : GPIO40 ~ GPIO47 사용.

```
EALLOW;
GpioIntRegs.⊟GPIOXINT3SEL.bit.GPIOSEL = 47;   // 외부 인터럽트 XINT3로 사용할 핀 선택: GPIO47
GpioIntRegs.GPIOXINT4SEL.bit.GPIOSEL = 46;   // 외부 인터럽트 XINT4로 사용할 핀 선택: GPIO46
GpioIntRegs.GPIOXINT5SEL.bit.GPIOSEL = 45;   // 외부 인터럽트 XINT5로 사용할 핀 선택: GPIO45
GpioIntRegs.GPIOXINT6SEL.bit.GPIOSEL = 44;   // 외부 인터럽트 XINT6로 사용할 핀 선택: GPIO44
EDIS;
```

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



GPIO 47 46 45 44 => ABCD

```
XIntruptRegs.XINT3CR.bit.⊟POLARITY = 2;      // XINT3 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
XIntruptRegs.XINT4CR.bit.POLARITY = 2;      // XINT4 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
XIntruptRegs.XINT5CR.bit.POLARITY = 2;      // XINT5 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
XIntruptRegs.XINT6CR.bit.POLARITY = 2;      // XINT6 인터럽트 발생 조건 설정: 입력 신호의 하강 엣지
```

Table 121. External Interrupt *n* Control Register (XINT*n*CR) 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 XINT*n*.
		0	Disable interrupt
		1	Enable interrupt

㉲ **Polarity**: 입력값에 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.

```
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
```

㉳ **Enable**: 1값을 입력하게 되면 인터럽트를 **enable**한다.

```
//  백터 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
```

PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register

◎ **PIEIER12**: Group Enable Register이며, INTx1, 2, 3, 4를 사용하여 외부 인터럽트 XINT3, 4, 5, 6에 해당하는 각각의 PIE 인터럽트를 **Enable 상태로 설정**해준다.

㉴ **IER |= M_INT12**: CPU 인터럽트를 **Enable 상태로 설정**해준다.

```
SW1_cnt = 0;
SW2_cnt = 0;
SW3_cnt = 0;
SW4_cnt = 0;
Loop_cnt = 0;
usec_delay    = 500000;// 500 msec
usec_delay_1 = 1000000;// 1sec
mode = 0;
state = 0;

EINT;   // Enable Global interrupt INTM
ERTM;   // Enable Global realtime interrupt DBGM

for(;;)
{
   if (state == 1)
   {
      switch(mode)
      {
      case 1:             // Mode 1
         Printmode_1();
         break;
      case 2:             // Mode 2
         Printmode_2();
         break;
      case 3:             // Mode 3
         Printmode_3();
         break;
      case 4:             // Mode 4
         Printmode_4();
         break;
      case 5:             // Mode 5
         Printmode_5();
         break;
      case 6:             // Mode 6
        Printmode_6();
         break;
      }
   }
   else
      Printmode_stp();    // STOP
  }
}
```

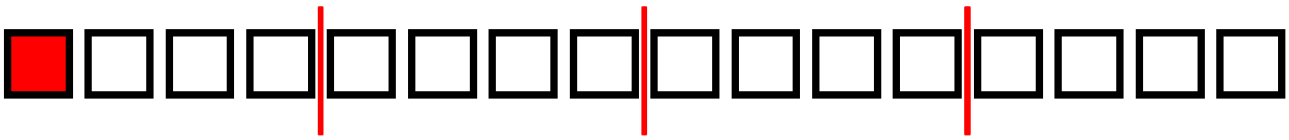
```
interrupt void Xint3_isr(void)          // (+) 인터럽트 실행
{
   SW1_cnt++;
   mode++;
   if(mode > 6 )
   {mode = 1;}
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
interrupt void Xint4_isr(void) // (-) 인터럽트 실행
{
```

```
    SW2_cnt++;
        mode--;
        if(mode <1 )
        {mode = 6;}
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
interrupt void Xint5_isr(void)//(STOP) 인터럽트 실행
{
    SW3_cnt++;
    state = 0;
    mode = 0;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
interrupt void Xint6_isr(void)//(START)인터럽트실행
{
    SW4_cnt++;
    state = 1;
    mode = 1;
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
```

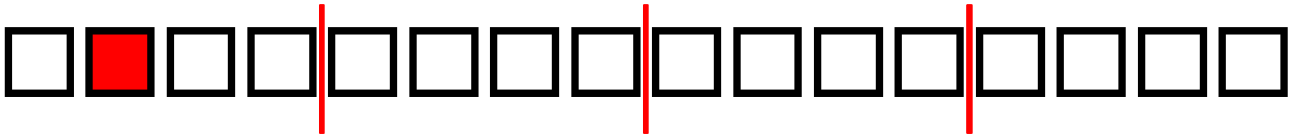
void Printmode_1(void)

```
{
    i = 0;
    for(;;)
    {
        GpioDataRegs.GPCCLEAR.all = 0x0000FFFF;
        GpioDataRegs.GPCSET.all = 0x00008000 >> i;
     // i만큼 shift 진행
        i++;
        DELAY_US(usec_delay_1); 시간간격 : 1s
        if(i == 16)
//1~16번째까지 순서대로 led의 빛이 점등되는데,
            i = 0;
//16번째인 마지막 led가 점등되면, 다시 처음부터로 되돌아간다.
        if((mode != 1)||(state == 0))
            break;
    }
}
```

GpioDataRegs.GPCCLEAR.all = 0x0000FFFF;



GpioDataRegs.GPCSET.all = 0x00008000 >> I=0;



GpioDataRegs.GPCSET.all = 0x00008000 >> I=1;

Table 77. GPIO Port C Set (GPCSET) Register Field Descriptions

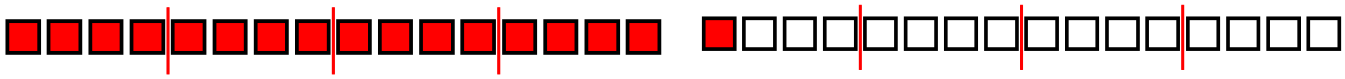
Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 70.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to high. If the pin is configured as a GPIO output then it will be driven high. If the pin is not configured as a GPIO output then the latch is set but the pin is not driven.

Table 78. GPIO Port C Clear (GPCCLEAR) Register Field Descriptions

Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 70.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to low. If the pin is configured as a GPIO output then it will be driven low. If the pin is not configured as a GPIO output then the latch is cleared but the pin is not driven.

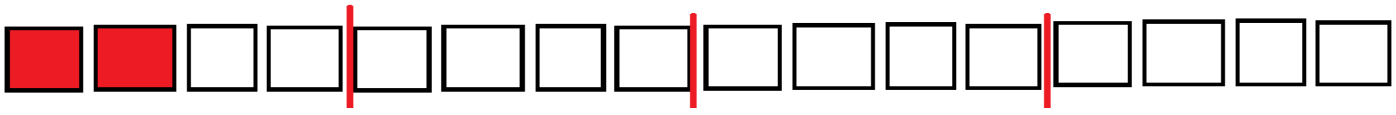
```
void Printmode_2(void)
{
    i = 0;
    for(;;)
    {
        GpioDataRegs.GPCSET.all = 0x0000FFFF;
        for(j = 0; j<16-i; j++)
        {
          GpioDataRegs.GPCCLEAR.all = 0x00007FFF >> i;
          GpioDataRegs.GPCSET.all = (0x00008000 >> i) >> j;
            DELAY_US(usec_delay);
            if((mode != 2)||(state == 0))
                break;
        }
        i++;
        if(i == 16)
            i = 0;
        if((mode != 2)||(state == 0))
            break;
    }
}
```

"GpioDataRegs.GPCSET.all = 0x0000FFFF;"에 의해, 모든 LED가 "On"이 된다. for문이 실행되면 i, j의 값은 0이 되고, "GpioDataRegs.GPCSET.all = (0x00008000 >> i) >> j;"의 동작도 "i, j"의 값이 모두 0이기 때문에 첫 번째 LED만 동작한다.

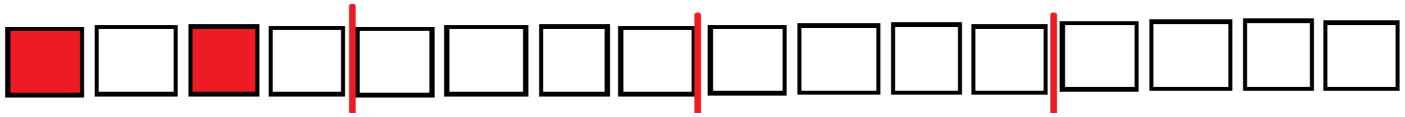


GpioDataRegs.GPCSET.all = 0x0000FFFF; → GpioDataRegs.GPCSET.all = (0x00008000 >> 0) >> 0

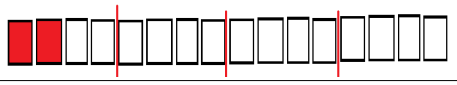
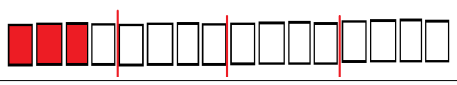
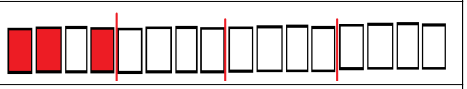
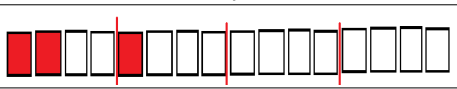
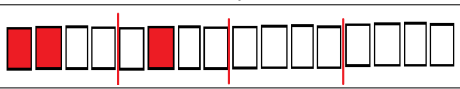
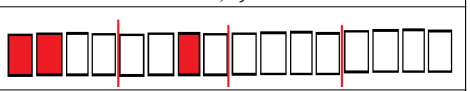
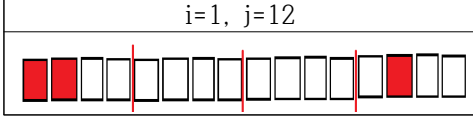
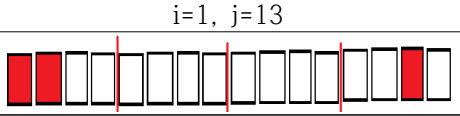
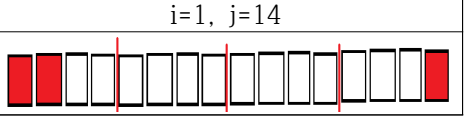
반복문이 실행되어 j=1이 되면 Clear의 경우 "i=0"에 의해 그대로이지만,
Set의 경우 "GpioDataRegs.GPCSET.all = (0x00008000 >> 0) >> 1;"에 의해 오른쪽으로 1칸 이동하여 아래 그림과 같이 동작한다.



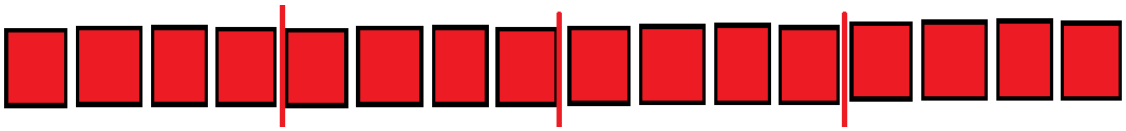
반복문이 실행되어 j=2이 되면 Clear의 경우 "i=0"에 의해 그대로이지만,
Set의 경우 "GpioDataRegs.GPCSET.all = (0x00008000 >> 0) >> 2;"에 의해 오른쪽으로 2칸 이동하여 아래 그림과 같이 동작한다.



이와 같은 방식으로 동작이 계속 반복되면 다음과 같은 결과를 갖게 된다.

i=1, j=0	i=1, j=1	i=1, j=2
		
i=1, j=3	i=1, j=4	i=1, j=5
		
i=1, j=12	i=1, j=13	i=1, j=14
		

"i=15"



```
void Printmode_3(void)
{
    GpioDataRegs.GPCSET.all = 0x0000AAAA;
    GpioDataRegs.GPCCLEAR.all = 0x00005555;
    for(;;)
    {
        DELAY_US(usec_delay);
        GpioDataRegs.GPCTOGGLE.all = 0x0000FFFF;
            if((mode != 3)||(state == 0))
                break;
    }
}
```

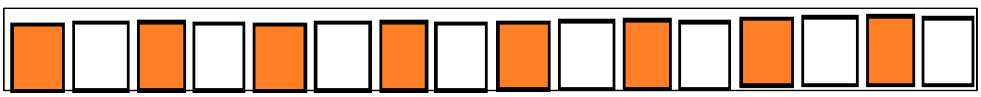
Table 79. GPIO Port C Toggle (GPCTOGGLE) Register Field Descriptions

Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 70.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to toggle from its current state. If the pin is configured as a GPIO output then it will be driven in the opposite direction of its current state. If the pin is not configured as a GPIO output then the latch is cleared but the pin is not driven.

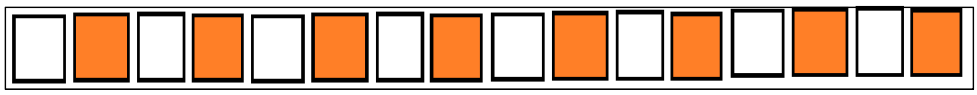
레지스터"Toggle"을 통해 LED의 불빛이 "On-Off" 동작을 반복 하게끔 한다.
처음에는 레지스터"Set"에 의해, ㉠과 같이 led가 작동하고 있다.

여기서 AAAA의 의미는 LED 16개를 4개씩 나눠서 봤을 때 (1010)(1010)(1010)(1010)를 AAAA로 16진수를 이용해서 나타낸 것이다. 마찬가지로 (0101)(0101)(0101)(0101)은 5555의 값을 갖는다.

"Toggle"을 통해, "Set => Clear", "Clear => Set" 동작을 반복하여 **"Mode3"의 수행 조건인 "번갈아 LED 점등"을 수행한다.**

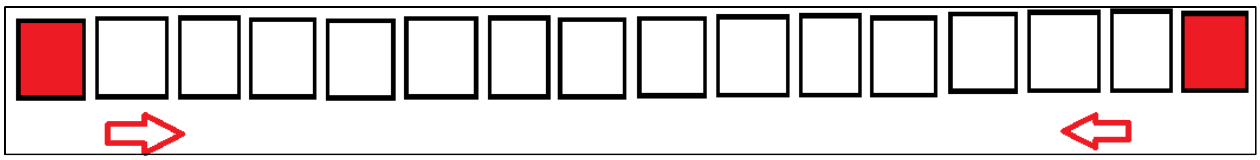


㉠ GpioDataRegs.GPCSET.all = 0x0000AAAA;



GpioDataRegs.GPCCLEAR.all = 0x00005555;

```
void Printmode_4(void)
{
    i = 0;
    for(;;)
    {
        GpioDataRegs.GPCCLEAR.all = 0x0000FFFF;
       GpioDataRegs.GPCSET.all = 0x00008000 >> i;
       GpioDataRegs.GPCSET.all = 0x00000001 << i;
        i++;
        DELAY_US(usec_delay_1);
        if(i == 16)
            i =0;
        if((mode != 4)||(state == 0))
            break;
    }
}
```



GpioDataRegs.GPCSET.all = 0x00008000 >> i; & GpioDataRegs.GPCSET.all = 0x00000001 << i;

* 초기조건 : LED 모두 Off한다. 이후 첫 번째와 열여섯 번째 LED가 켜지면서 동작을 시작한다.

"Mode 4"의 경우 양 끝의 LED는 서로 반대 방향으로 이동한다. 해당 동작 수행을 위해 SET 레지스터로 "0x00008000"은 왼쪽 첫 번째 LED가 켜지도록 하였고, "0x00000001"은 오른쪽 첫 번째 LED가 켜지도록 하였다.

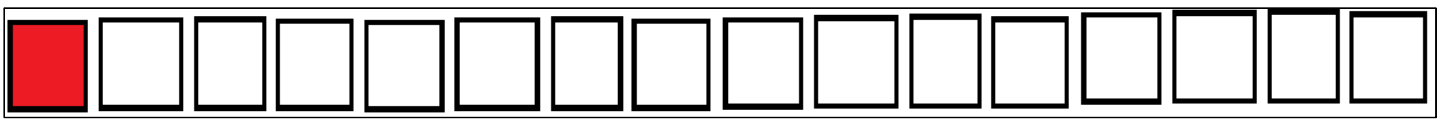
LED의 불이 이동하도록 ">>, <<"로 표기되는 Shift 연산자를 사용하였다. 시프트 연산은 "변수 << 이동할 비트 수" 또는 "변수 >> 이동할 비트 수" 형식으로 사용한다. 즉, 지정한 횟수대로 비트를 이동시키며 모자라는 공간은 0으로 채우는 것이다. 연산자 모양 그대로 <<는 왼쪽, >>는 오른쪽 방향을 의미한다.

위 코드를 보면 ">>i"를 통해, 반복되는 횟수만큼 LED가 한 칸씩 움직이는 것처럼 보이도록 설정하였다.
그리고 "if(i == 16)"문을 통해, 왼쪽에서 출발한 LED가 오른쪽 끝에 도착하면, 다시 처음 상태부터 시작하도록 설정했다.

"DELAY_US(usec_delay_1);"
delay 시간은 초기조건인 "usec_delay_1 = 1000000;// 1sec"에 의해, 1sec이다.

```
void Printmode_5(void)
{
    i = 0, num = 0;
    for(;;)
    {
        GpioDataRegs.GPCCLEAR.all = 0x0000FFFF;
        switch(i)
        {
        case 0:
  GpioDataRegs.GPCSET.all = 0x00008000 >> num;
            DELAY_US(usec_delay_1);
            i++;
            num = num + i;
            break;

        case 1:
  GpioDataRegs.GPCSET.all = 0x0000C000 >> num;
            DELAY_US(usec_delay_1);
            i++;
            num = num + i;
            break;
        case 2:
  GpioDataRegs.GPCSET.all = 0x0000E000 >> num;
            DELAY_US(usec_delay_1);
            i++;
            num = num + i;
            break;
        case 3:
  GpioDataRegs.GPCSET.all = 0x0000F000 >> num;
            DELAY_US(usec_delay_1);
            i++;
            num = num + i;
            break;
        case 4:
  GpioDataRegs.GPCSET.all = 0x0000F800 >> num;
            DELAY_US(usec_delay_1);
            i++;
            num = num + i;
            i=0;
            break;
        }
        if((mode != 5)||(state == 0))
            break;
      }
}
```

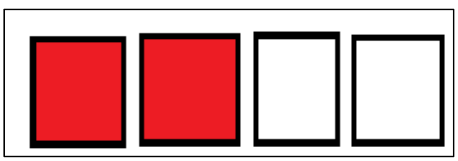
Case 0(i=0, num=0)



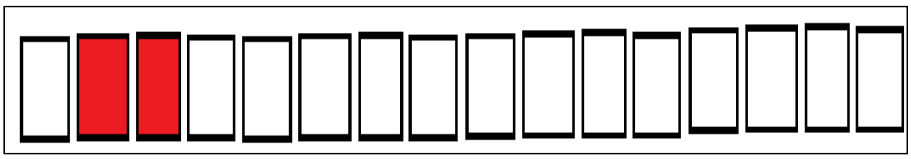
초기조건이 i=0, num=0 이기에, "case 0"이 실행된다. 그렇게 되면 위와 같이 첫 번째 LED가 점등되고, Delay 이후에 그 다음 동작이 되는데, "i++"에 의해 i는 1이 되고, num의 값은 1이 된다.

Case 1(i=1, num=1)

i가 1이 되어 "Case 1"이 시작되면, 0x0000C000에 의해 (1100)형태로 LED가 점등 되는데, ">>"라는 시프트 연산에 의해 ㉠과 같이 한 칸 이동한 동작을 하게 된다. 그리고 동작이 끝나면, "i++"에 의해, i=2가 되고, "num"의 값은 "2+1=3"이 된다.



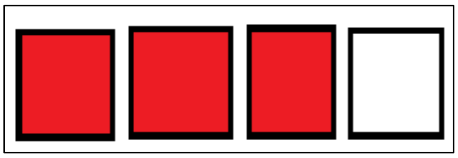
0x0000C000



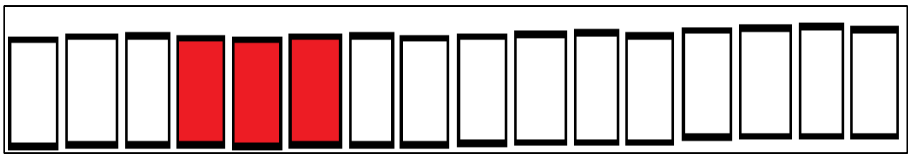
㉠0x0000C000 >> num;

Case 2(i=2, num=3)

i가 2가 되어 “Case 2”가 시작되면, 0x0000E000에 의해 LED가 점등 되는데, “>>”라는 시프트 연산에 의해, 세 칸 이동한 ㉡과 같이 동작하고, “i++”에 의해, i=3이 되고, “num”의 값은 “3+3=6”이 된다. 이후 이와 같은 동작이 계속 반복되며, case 4까지 진행되면 i는 0으로 초기화 되어, 다시 case 0부터 case4까지 차례대로 실행된다.



0x0000E000



㉡0x0000E000 >> num;

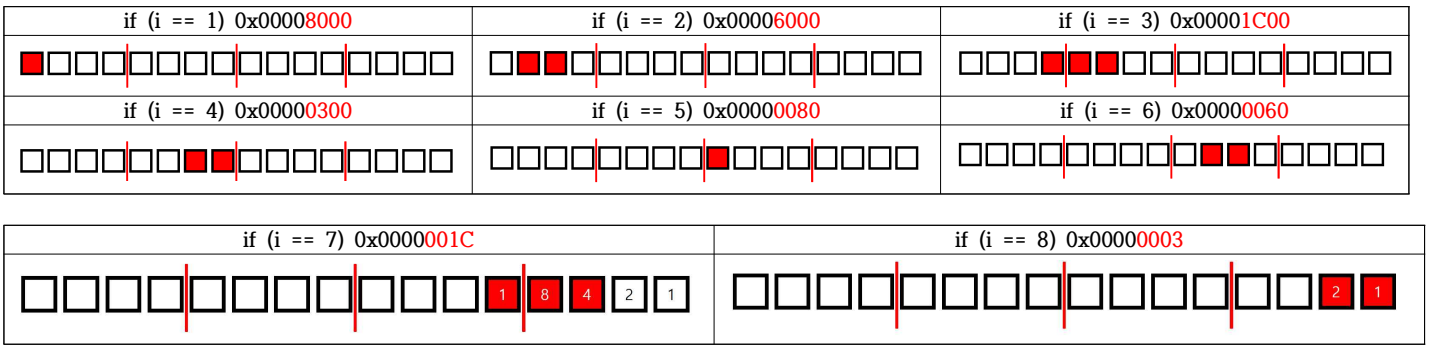
void Printmode_6(void)

```
{
      i++;
     if (i == 1)
       {
          GpioDataRegs.GPCDAT.all = 0x00008000;
           DELAY_US(usec_delay_1);
       }

       else if (i == 2)
       {
          GpioDataRegs.GPCDAT.all = 0x00006000;
           DELAY_US(usec_delay_1);
       }

       else if (i == 3)
       {
          GpioDataRegs.GPCDAT.all = 0x00001C00;
           DELAY_US(usec_delay_1);
       }

       else if (i == 4)
       {
          GpioDataRegs.GPCDAT.all = 0x00000300;
       DELAY_US(usec_delay_1);
       }

       else if (i == 5)
       {
          GpioDataRegs.GPCDAT.all = 0x00000080;
           DELAY_US(usec_delay_1);
       }

       else if (i == 6)
       {
          GpioDataRegs.GPCDAT.all = 0x00000060;
      DELAY_US(usec_delay_1);
       }

       else if (i == 7)
       {
          GpioDataRegs.GPCDAT.all = 0x0000001C;
       DELAY_US(usec_delay_1);
       }

       else if (i == 8)
       {
          GpioDataRegs.GPCDAT.all = 0x00000003;
       DELAY_US(usec_delay_1);
       }
}
```

초기 조건은 i가 “0”이지만, “i++”에 의해 1이 되어 “if(i ==1)” 구문이 실행된다. 동작 내용은 16진수에 의한 첫 번째 LED가 “On”이 되고, Delay시간 후에 다음 동작이 진행한다. 코드 자체가 하나의 if문으로 구성되어 있기 때문에, 해당 if문이 실행 된 뒤 다시 처음의 “i++”로 되돌아간다.

Table 70. GPIO Port C Data (GPCDAT) Register Field Descriptions

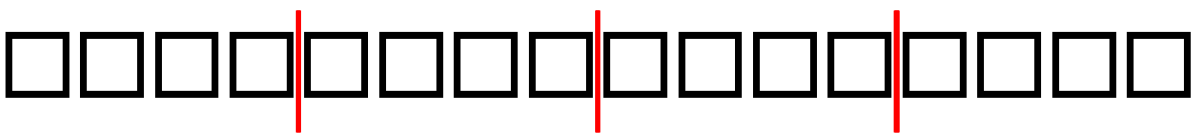
Bit	Field	Value	Description
31-3	Reserved		Reserved
2-0	GPIO87-GPIO64		Each bit corresponds to one GPIO port B pin (GPIO64-GPIO87) as shown in Figure 67
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for. Writing a 0 will force an output of 0 if the pin is configured as a GPIO output in the appropriate GPCMUX1 and GPCDIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured for. Writing a 1 will force an output of 1 if the pin is configured as a GPIO output in the GPCMUX1 and GPCDIR registers; otherwise, the value is latched but not used to drive the pin.



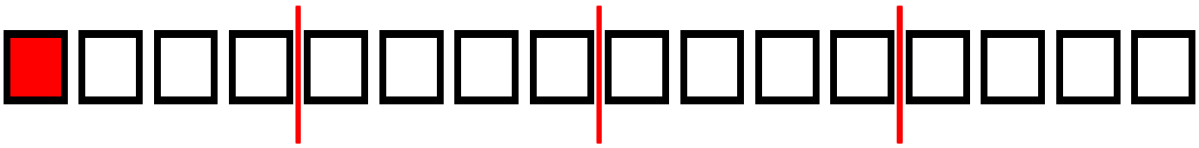
다음과 같은 순서로 "i++"에 의해 달라지는 i값에 따라 서로 다른 if문이 실행된다. 해당 코드에서 GPCDAT에 입력되는 숫자의 의미는 다음과 같다. 16개의 LED를 네 부분으로 나눈 뒤 각자 부분에 맞는 값을 16진수로 표현한 것이다. 예를 들어 i=7일 경우, 8+4는 C이기 때문에 (001C)로 나타낸다. 마찬가지로 i=8일 경우 (0,0,0,2+1)=(0003)이 된다.

```
void Printmode_stp(void)
{
    GpioDataRegs.GPCDAT.all = 0x00000000;
    for(;;)
    {
        GpioDataRegs.GPCTOGGLE.bit.GPIO79 = 1;
        DELAY_US(usec_delay);

        if(state == 1)
            break;
    }
}
```

GpioDataRegs.GPCDAT.all = 0x00000000; => 모든 LED를 Off시킨다.



GpioDataRegs.GPCTOGGLE.bit.GPIO79 = 1; => GPIO79번을 On으로 반전시킨다.



일정 시간 Delay된 후, 하나의 사이클의 동작이 끝난다. 이러한 동작은 state가 1이 될 때까지 (start를 누를 때까지) 반복된다.

<코드 2. 1000줄>

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

#define timeA 1000000
#define timeB 500000
#define timeC 500000
#define timeD 1000000
#define timeE 1000000
#define timeF 1000000
#define timeStop 500000

//   함수선언

interrupt void Xint3_isr(void);
interrupt void Xint4_isr(void);
interrupt void Xint5_isr(void);
interrupt void Xint6_isr(void);
int modeA(int);
int modeB(int);
int modeC(int);
int modeD(int);
int modeE(int);
int modeF(int);
int modeS(int);

// 시스템에서 사용할 전역 변수 선언

Uint16 Loop_cnt;
Uint16 SW1_cnt, SW2_cnt, SW3_cnt, SW4_cnt;
(스위치 4개 설정.)
volatile unsigned int i = 0;
volatile unsigned int mode = 0, stop = 0;
```

```
// 사용자 함수
int modeA(int i)
{                                                    // LED가 16개이므로 i=1~16까지 값을 설정해준다.
        if (i == 1)
        {
        GpioDataRegs.GPCDAT.all = 32768;     DELAY_US(timeA);  // 16번째 LED를 timeA 동안 On 시킨다. 2^15 = 32768
        }
```

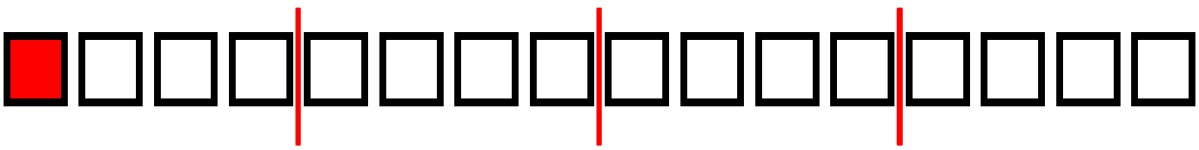


Table 70. GPIO Port C Data (GPCDAT) Register Field Descriptions

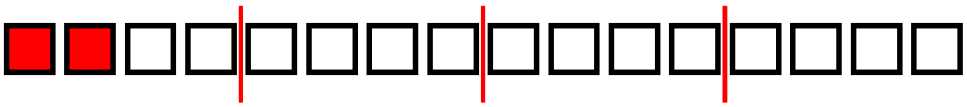
Bit	Field	Value	Description
31-3	Reserved		Reserved
2-0	GPIO87-GPIO64		Each bit corresponds to one GPIO port B pin (GPIO64-GPIO87) as shown in Figure 67
		0	Reading a 0 indicates that the state of the pin is currently low, irrespective of the mode the pin is configured for. Writing a 0 will force an output of 0 if the pin is configured as a GPIO output in the appropriate GPCMUX1 and GPCDIR registers; otherwise, the value is latched but not used to drive the pin.
		1	Reading a 1 indicates that the state of the pin is currently high irrespective of the mode the pin is configured for. Writing a 1 will force an output of 1 if the pin is configured as a GPIO output in the GPCMUX1 and GPCDIR registers; otherwise, the value is latched but not used to drive the pin.

```
else if (i == 2)
{
GpioDataRegs.GPCDAT.all = 16384;     DELAY_US(timeA);
} // 15번째 LED를 timeA 동안 On 시킨다. 2^14 = 16384
else if (i == 3)
{
GpioDataRegs.GPCDAT.all = 8192;      DELAY_US(timeA);
} // 14번째 LED를 timeA 동안 On 시킨다. 2^13 = 8192
else if (i == 4)
{
GpioDataRegs.GPCDAT.all = 4096;      DELAY_US(timeA);
} // 13번째 LED를 timeA 동안 On 시킨다. 2^12 = 4096
else if (i == 5)
{
GpioDataRegs.GPCDAT.all = 2048;      DELAY_US(timeA);
} // 12번째 LED를 timeA 동안 On 시킨다. 2^11 = 2048
else if (i == 6)
{
GpioDataRegs.GPCDAT.all = 1024;      DELAY_US(timeA);
} // 11번째 LED를 timeA 동안 On 시킨다. 2^10 = 1024
else if (i == 7)
{
GpioDataRegs.GPCDAT.all = 512;       DELAY_US(timeA);
} // 10번째 LED를 timeA 동안 On 시킨다. 2^9 = 512
else if (i == 8)
{
GpioDataRegs.GPCDAT.all = 256;     DELAY_US(timeA);
```

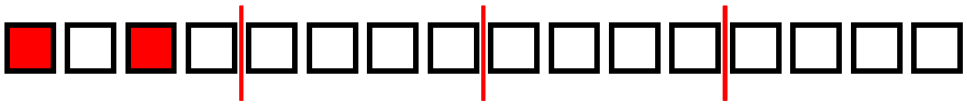
```
} // 9번째 LED를 timeA 동안 On 시킨다. 2^8 = 256
else if (i == 9)
{
GpioDataRegs.GPCDAT.all = 128;      DELAY_US(timeA);
} // 8번째 LED를 timeA 동안 On 시킨다. 2^7 = 128
else if (i == 10)
{
GpioDataRegs.GPCDAT.all = 64;       DELAY_US(timeA);
} // 7번째 LED를 timeA 동안 On 시킨다. 2^6 = 64

else if (i == 11)
{
GpioDataRegs.GPCDAT.all = 32;       DELAY_US(timeA);
} // 6번째 LED를 timeA 동안 On 시킨다. 2^5 = 32
else if (i == 12)
{
GpioDataRegs.GPCDAT.all = 16;       DELAY_US(timeA);
} // 5번째 LED를 timeA 동안 On 시킨다. 2^4 = 16
else if (i == 13)
{
GpioDataRegs.GPCDAT.all = 8;        DELAY_US(timeA);
} // 4번째 LED를 timeA 동안 On 시킨다. 2^3 = 8
else if (i == 14)
{
GpioDataRegs.GPCDAT.all = 4;        DELAY_US(timeA);
} // 3번째 LED를 timeA 동안 On 시킨다. 2^2 = 4
else if (i == 15)
{
GpioDataRegs.GPCDAT.all = 2;        DELAY_US(timeA);
} // 2번째 LED를 timeA 동안 On 시킨다. 2^1 = 2
else if (i == 16)
{
GpioDataRegs.GPCDAT.all = 1;        DELAY_US(timeA);
} // 1번째 LED를 timeA 동안 On 시킨다. 2^0 = 0
return 1;
}
```

int modeB(int i)

```
{
    if (i == 1)
    {
    GpioDataRegs.GPCDAT.all = 32768;                DELAY_US(timeB);
    } // 16번째 LED를 timeB 동안 On 시킨다. 2^15 = 32768
    else if (i == 2)
    {
    GpioDataRegs.GPCDAT.all = 49152;                DELAY_US(timeB);
    } // 16번째, 15번째 LED를 timeB 동안 On 시킨다. 2^15 + 2^14 = 49152 = 32768 + 16384
```



```
    else if (i == 3)
    {
        GpioDataRegs.GPCDAT.all = 40960; DELAY_US(timeB);
    } // 16번째, 14번째 LED를 timeB 동안 On 시킨다. 2^15 + 2^13 = 40960 = 32768 + 8192
```



위와 같은 방법으로 나머지의 경우에 대해서도 16개의 LED를 2진수로 표현한 뒤 2^0~2^{15}의 덧셈으로 LED의 On 상태를 표현할 수 있다.

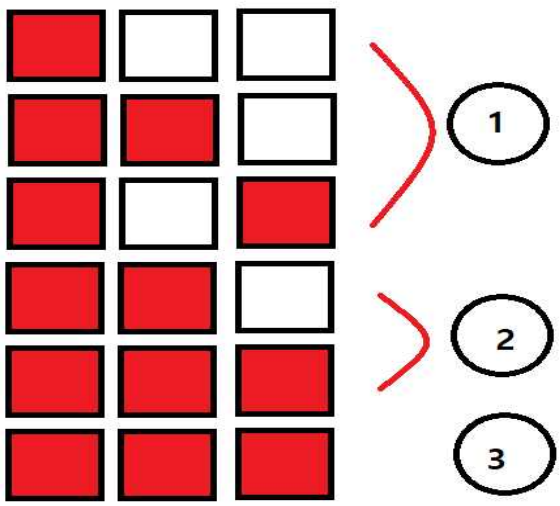
```
 else if (i == 4)
{
        GpioDataRegs.GPCDAT.all = 36864; DELAY_US(timeB);
}
else if (i == 5)
{
        GpioDataRegs.GPCDAT.all = 34816; DELAY_US(timeB);
}
else if (i == 6)
{
        GpioDataRegs.GPCDAT.all = 33792; DELAY_US(timeB);
}
else if (i == 7)
{
        GpioDataRegs.GPCDAT.all = 33280; DELAY_US(timeB);
}
else if (i == 8)
{
        GpioDataRegs.GPCDAT.all = 33024; DELAY_US(timeB);
}
else if (i == 9)
{
        GpioDataRegs.GPCDAT.all = 32896; DELAY_US(timeB);
}
else if (i == 10)
{
```

```
                GpioDataRegs.GPCDAT.all = 32832; DELAY_US(timeB);
}
else if (i == 11)
{
                GpioDataRegs.GPCDAT.all = 32800; DELAY_US(timeB);
}
else if (i == 12)
{
                GpioDataRegs.GPCDAT.all = 32784; DELAY_US(timeB);
}
else if (i == 13)
{
                GpioDataRegs.GPCDAT.all = 32776; DELAY_US(timeB);
}
else if (i == 14)
{
                GpioDataRegs.GPCDAT.all = 32772; DELAY_US(timeB);
}
else if (i == 15)
{
                GpioDataRegs.GPCDAT.all = 32770; DELAY_US(timeB);
}
else if (i == 16)
{
                GpioDataRegs.GPCDAT.all = 32769; DELAY_US(timeB);
}
else if (i == 17)
{
                GpioDataRegs.GPCDAT.all = 49152; DELAY_US(timeB);
}
else if (i == 18)
{
                GpioDataRegs.GPCDAT.all = 57344; DELAY_US(timeB);
}
else if (i == 19)
{
                GpioDataRegs.GPCDAT.all = 53248; DELAY_US(timeB);
}
else if (i == 20)
{
                GpioDataRegs.GPCDAT.all = 51200; DELAY_US(timeB);
}
else if (i == 21)
{
                GpioDataRegs.GPCDAT.all = 50176; DELAY_US(timeB);
}
else if (i == 22)
{
                GpioDataRegs.GPCDAT.all = 49664; DELAY_US(timeB);
}
else if (i == 23)
{
                GpioDataRegs.GPCDAT.all = 49408; DELAY_US(timeB);
}
else if (i == 24)
{
                GpioDataRegs.GPCDAT.all = 49280; DELAY_US(timeB);
}
else if (i == 25)
{
                GpioDataRegs.GPCDAT.all = 49216; DELAY_US(timeB);
}
else if (i == 26)
{
                GpioDataRegs.GPCDAT.all = 49184; DELAY_US(timeB);
}
else if (i == 27)
{
                GpioDataRegs.GPCDAT.all = 49168; DELAY_US(timeB);
}
else if (i == 28)
{
                GpioDataRegs.GPCDAT.all = 49160; DELAY_US(timeB);
}
else if (i == 29)
{
                GpioDataRegs.GPCDAT.all = 49156; DELAY_US(timeB);
}
else if (i == 30)
{
                GpioDataRegs.GPCDAT.all = 49154; DELAY_US(timeB);
}
else if (i == 31)
{
                GpioDataRegs.GPCDAT.all = 49153; DELAY_US(timeB);
}
else if (i == 32)
{
                GpioDataRegs.GPCDAT.all = 57344; DELAY_US(timeB);
}
else if (i == 33)
{
                GpioDataRegs.GPCDAT.all = 61440; DELAY_US(timeB);
}
else if (i == 34)
{
                GpioDataRegs.GPCDAT.all = 59392; DELAY_US(timeB);
}
else if (i == 35)
{
                GpioDataRegs.GPCDAT.all = 58368; DELAY_US(timeB);
}
else if (i == 36)
{
                GpioDataRegs.GPCDAT.all = 57856; DELAY_US(timeB);
}
else if (i == 37)
{
                GpioDataRegs.GPCDAT.all = 57600; DELAY_US(timeB);
}
else if (i == 38)
{
                GpioDataRegs.GPCDAT.all = 57472; DELAY_US(timeB);
}
else if (i == 39)
{
                GpioDataRegs.GPCDAT.all = 57408; DELAY_US(timeB);
}
else if (i == 40)
{
                GpioDataRegs.GPCDAT.all = 57376; DELAY_US(timeB);
}
else if (i == 41)
{
                GpioDataRegs.GPCDAT.all = 57360; DELAY_US(timeB);
}
else if (i == 42)
{
                GpioDataRegs.GPCDAT.all = 57352; DELAY_US(timeB);
}
else if (i == 43)
{
                GpioDataRegs.GPCDAT.all = 57348; DELAY_US(timeB);
}
else if (i == 44)
{
                GpioDataRegs.GPCDAT.all = 57346; DELAY_US(timeB);
}
else if (i == 45)
{
                GpioDataRegs.GPCDAT.all = 57345; DELAY_US(timeB);
}
else if (i == 46)
{
                GpioDataRegs.GPCDAT.all = 61440; DELAY_US(timeB);
}
else if (i == 47)
{
                GpioDataRegs.GPCDAT.all = 63488; DELAY_US(timeB);
}
else if (i == 48)
{
                GpioDataRegs.GPCDAT.all = 62464; DELAY_US(timeB);
}
```

```
else if (i == 49)
{
        GpioDataRegs.GPCDAT.all = 61952; DELAY_US(timeB);
}
else if (i == 50)
{
        GpioDataRegs.GPCDAT.all = 61696; DELAY_US(timeB);
}
else if (i == 51)
{
        GpioDataRegs.GPCDAT.all = 61568; DELAY_US(timeB);
}
else if (i == 52)
{
        GpioDataRegs.GPCDAT.all = 61504; DELAY_US(timeB);
}
else if (i == 53)
{
        GpioDataRegs.GPCDAT.all = 61472; DELAY_US(timeB);
}
else if (i == 54)
{
        GpioDataRegs.GPCDAT.all = 61456; DELAY_US(timeB);
}
else if (i == 55)
{
        GpioDataRegs.GPCDAT.all = 61448; DELAY_US(timeB);
}
else if (i == 56)
{
        GpioDataRegs.GPCDAT.all = 61444; DELAY_US(timeB);
}
else if (i == 57)
{
        GpioDataRegs.GPCDAT.all = 61442; DELAY_US(timeB);
}
else if (i == 58)
{
        GpioDataRegs.GPCDAT.all = 61441; DELAY_US(timeB);
}
else if (i == 59)
{
        GpioDataRegs.GPCDAT.all = 63488; DELAY_US(timeB);
}
else if (i == 60)
{
        GpioDataRegs.GPCDAT.all = 64512; DELAY_US(timeB);
}
else if (i == 61)
{
        GpioDataRegs.GPCDAT.all = 64000; DELAY_US(timeB);
}
else if (i == 62)
{
        GpioDataRegs.GPCDAT.all = 63744; DELAY_US(timeB);
}
else if (i == 63)
{
        GpioDataRegs.GPCDAT.all = 63616; DELAY_US(timeB);
}
else if (i == 64)
{
        GpioDataRegs.GPCDAT.all = 63552; DELAY_US(timeB);
}
else if (i == 65)
{
        GpioDataRegs.GPCDAT.all = 63520; DELAY_US(timeB);
}
else if (i == 66)
{
        GpioDataRegs.GPCDAT.all = 63504; DELAY_US(timeB);
}
else if (i == 67)
{
        GpioDataRegs.GPCDAT.all = 63496; DELAY_US(timeB);
}
else if (i == 68)
{
        GpioDataRegs.GPCDAT.all = 63492; DELAY_US(timeB);
}
else if (i == 69)
{
        GpioDataRegs.GPCDAT.all = 63490; DELAY_US(timeB);
}
else if (i == 70)
{
        GpioDataRegs.GPCDAT.all = 63489; DELAY_US(timeB);
}
else if (i == 71)
{
        GpioDataRegs.GPCDAT.all = 64512; DELAY_US(timeB);
}
else if (i == 72)
{
        GpioDataRegs.GPCDAT.all = 65024; DELAY_US(timeB);
}
else if (i == 73)
{
        GpioDataRegs.GPCDAT.all = 64768; DELAY_US(timeB);
}
else if (i == 74)
{
        GpioDataRegs.GPCDAT.all = 64640; DELAY_US(timeB);
}
else if (i == 75)
{
        GpioDataRegs.GPCDAT.all = 64576; DELAY_US(timeB);
}
else if (i == 76)
{
        GpioDataRegs.GPCDAT.all = 64544; DELAY_US(timeB);
}
else if (i == 77)
{
        GpioDataRegs.GPCDAT.all = 64528; DELAY_US(timeB);
}
else if (i == 78)
{
        GpioDataRegs.GPCDAT.all = 64520; DELAY_US(timeB);
}
else if (i == 79)
{
        GpioDataRegs.GPCDAT.all = 64516; DELAY_US(timeB);
}
else if (i == 80)
{
        GpioDataRegs.GPCDAT.all = 64514; DELAY_US(timeB);
}
else if (i == 81)
{
        GpioDataRegs.GPCDAT.all = 64513; DELAY_US(timeB);
}
else if (i == 82)
{
        GpioDataRegs.GPCDAT.all = 65024; DELAY_US(timeB);
}
else if (i == 83)
{
        GpioDataRegs.GPCDAT.all = 65280; DELAY_US(timeB);
}
else if (i == 84)
{
        GpioDataRegs.GPCDAT.all = 65152; DELAY_US(timeB);
}
else if (i == 85)
{
        GpioDataRegs.GPCDAT.all = 65088; DELAY_US(timeB);
}
else if (i == 86)
{
        GpioDataRegs.GPCDAT.all = 65056; DELAY_US(timeB);
}
else if (i == 87)
{
```

```
                GpioDataRegs.GPCDAT.all = 65040; DELAY_US(timeB);
}
else if (i == 88)
{
                GpioDataRegs.GPCDAT.all = 65032; DELAY_US(timeB);
}
else if (i == 89)
{
                GpioDataRegs.GPCDAT.all = 65028; DELAY_US(timeB);
}
else if (i == 90)
{
                GpioDataRegs.GPCDAT.all = 65026; DELAY_US(timeB);
}
else if (i == 91)
{
                GpioDataRegs.GPCDAT.all = 65025; DELAY_US(timeB);
}
else if (i == 92)
{
                GpioDataRegs.GPCDAT.all = 65280; DELAY_US(timeB);
}
else if (i == 93)
{
                GpioDataRegs.GPCDAT.all = 65408; DELAY_US(timeB);
}
else if (i == 94)
{
                GpioDataRegs.GPCDAT.all = 65344; DELAY_US(timeB);
}
else if (i == 95)
{
                GpioDataRegs.GPCDAT.all = 65312; DELAY_US(timeB);
}
else if (i == 96)
{
                GpioDataRegs.GPCDAT.all = 65296; DELAY_US(timeB);
}
else if (i == 97)
{
                GpioDataRegs.GPCDAT.all = 65288; DELAY_US(timeB);
}
else if (i == 98)
{
                GpioDataRegs.GPCDAT.all = 65284; DELAY_US(timeB);
}
else if (i == 99)
{
                GpioDataRegs.GPCDAT.all = 65282; DELAY_US(timeB);
}
else if (i == 100)
{
                GpioDataRegs.GPCDAT.all = 65281; DELAY_US(timeB);
}
else if (i == 101)
{
                GpioDataRegs.GPCDAT.all = 65408; DELAY_US(timeB);
}
else if (i == 102)
{
                GpioDataRegs.GPCDAT.all = 65472; DELAY_US(timeB);
}
else if (i == 103)
{
                GpioDataRegs.GPCDAT.all = 65440; DELAY_US(timeB);
}
else if (i == 104)
{
                GpioDataRegs.GPCDAT.all = 65424; DELAY_US(timeB);
}
else if (i == 105)
{
                GpioDataRegs.GPCDAT.all = 65416; DELAY_US(timeB);
}
else if (i == 106)
{
                GpioDataRegs.GPCDAT.all = 65412; DELAY_US(timeB);
}
else if (i == 107)
{
                GpioDataRegs.GPCDAT.all = 65410; DELAY_US(timeB);
}
else if (i == 108)
{
                GpioDataRegs.GPCDAT.all = 65409; DELAY_US(timeB);
}
else if (i == 109)
{
                GpioDataRegs.GPCDAT.all = 65472; DELAY_US(timeB);
}
else if (i == 110)
{
                GpioDataRegs.GPCDAT.all = 65504; DELAY_US(timeB);
}
else if (i == 111)
{
                GpioDataRegs.GPCDAT.all = 65488; DELAY_US(timeB);
}
else if (i == 112)
{
                GpioDataRegs.GPCDAT.all = 65480; DELAY_US(timeB);
}
else if (i == 113)
{
                GpioDataRegs.GPCDAT.all = 65476; DELAY_US(timeB);
}
else if (i == 114)
{
                GpioDataRegs.GPCDAT.all = 65474; DELAY_US(timeB);
}
else if (i == 115)
{
                GpioDataRegs.GPCDAT.all = 65473; DELAY_US(timeB);
}
else if (i == 116)
{
                GpioDataRegs.GPCDAT.all = 65504; DELAY_US(timeB);
}
else if (i == 117)
{
                GpioDataRegs.GPCDAT.all = 65520; DELAY_US(timeB);
}
else if (i == 118)
{
                GpioDataRegs.GPCDAT.all = 65512; DELAY_US(timeB);
}
else if (i == 119)
{
                GpioDataRegs.GPCDAT.all = 65508; DELAY_US(timeB);
}
else if (i == 120)
{
                GpioDataRegs.GPCDAT.all = 65506; DELAY_US(timeB);
}
else if (i == 121)
{
                GpioDataRegs.GPCDAT.all = 65505; DELAY_US(timeB);
}
else if (i == 122)
{
                GpioDataRegs.GPCDAT.all = 65520; DELAY_US(timeB);
}
else if (i == 123)
{
                GpioDataRegs.GPCDAT.all = 65528; DELAY_US(timeB);
}
else if (i == 124)
{
                GpioDataRegs.GPCDAT.all = 65524; DELAY_US(timeB);
}
else if (i == 125)
{
                GpioDataRegs.GPCDAT.all = 65522; DELAY_US(timeB);
}
```

```
else if (i == 126)
{
        GpioDataRegs.GPCDAT.all = 65521; DELAY_US(timeB);
}
else if (i == 127)
{
        GpioDataRegs.GPCDAT.all = 65528; DELAY_US(timeB);
}
else if (i == 128)
{
        GpioDataRegs.GPCDAT.all = 65532; DELAY_US(timeB);
}
else if (i == 129)
{
        GpioDataRegs.GPCDAT.all = 65530; DELAY_US(timeB);
}
else if (i == 130)
{
        GpioDataRegs.GPCDAT.all = 65529; DELAY_US(timeB);
}
else if (i == 131)
{
        GpioDataRegs.GPCDAT.all = 65532; DELAY_US(timeB);
}
else if (i == 132)
{
        GpioDataRegs.GPCDAT.all = 65534; DELAY_US(timeB);
}
else if (i == 133)
{
GpioDataRegs.GPCDAT.all = 65533; DELAY_US(timeB);
}
else if (i == 134)
{
GpioDataRegs.GPCDAT.all = 65534; DELAY_US(timeB);
}
else if (i == 135)
{
GpioDataRegs.GPCDAT.all = 65535; DELAY_US(timeB);
}
else if (i == 136)
{
GpioDataRegs.GPCDAT.all = 65535; DELAY_US(timeB);
}

return 1;
}
```



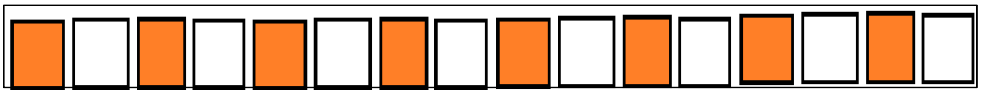
만약 LED가 3개이면 1+2+3=6으로 6가지의 경우에 대한 조건문을 설정해야 한다.

이번 프로젝트에서는 16개의 LED를 이용했으므로 $\sum_{i=1}^{16} i = 136$으로 136가지의 경우에 대한 조건문을 설정해야 한다.

int modeC(int i)

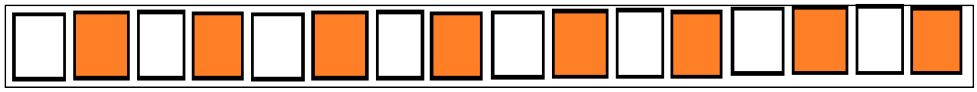
```
{
        if (i == 1)
        {
                GpioDataRegs.GPCDAT.all = 43690;      DELAY_US(timeC);
        } // 16번째, 14번째 … 2번째 LED를 timeC 동안 On 시킨다.
```

$2^{15}+2^{13}+2^{11}+2^{9}+2^{7}+2^{5}+2^{3}+2^{1}=43690$



```
        else if (i == 2)
        {
                GpioDataRegs.GPCDAT.all = 21845;      DELAY_US(timeC);
        } // 15번째, 13번째 … 1번째 LED를 timeC 동안 On 시킨다.
```

$2^{14}+2^{12}+2^{10}+2^{8}+2^{6}+2^{4}+2^{2}+2^{0}=65535-43690=21845$



```
        return 1;
}
```

int modeD(int i)

```
{
        if (i == 1)
        {
GpioDataRegs.GPCDAT.all = 32769;      DELAY_US(timeD);
        }
// 16번째, 1번째 LED를 timeD 동안 On 시킨다.
```

$2^{15}+2^{0}=1+32768=32769$

```
        else if (i == 2)
        {
GpioDataRegs.GPCDAT.all = 16386;      DELAY_US(timeD);
        }
// 15번째, 2번째 LED를 timeD 동안 On 시킨다.
```

$2^{14}+2^{1}=16386$

```
        else if (i == 3)
        {
GpioDataRegs.GPCDAT.all = 8196;       DELAY_US(timeD);
        }
// 14번째, 3번째 LED를 timeD 동안 On 시킨다.
```

$2^{13}+2^{2}=8196$

```
        else if (i == 4)
        {
GpioDataRegs.GPCDAT.all = 4104;       DELAY_US(timeD);
        }
// 13번째, 4번째 LED를 timeD 동안 On 시킨다.
```

$2^{12}+2^{3}=4104$

```
        else if (i == 5)
        {
GpioDataRegs.GPCDAT.all = 2064;       DELAY_US(timeD);
        }
// 12번째, 5번째 LED를 timeD 동안 On 시킨다.
```

$2^{11}+2^{4}=2064$

```
        else if (i == 6)
        {
GpioDataRegs.GPCDAT.all = 1056;       DELAY_US(timeD);
        }
// 11번째, 6번째 LED를 timeD 동안 On 시킨다.
```

$2^{10}+2^{5}=1056$

```
        else if (i == 7)
        {
GpioDataRegs.GPCDAT.all = 576;        DELAY_US(timeD);
        }
// 10번째, 7번째 LED를 timeD 동안 On 시킨다.
```

$2^{9}+2^{6}=576$

```
        else if (i == 8)
        {
GpioDataRegs.GPCDAT.all = 384;        DELAY_US(timeD);
        }
// 9번째, 8번째 LED를 timeD 동안 On 시킨다.
```

$2^{8}+2^{7}=384$

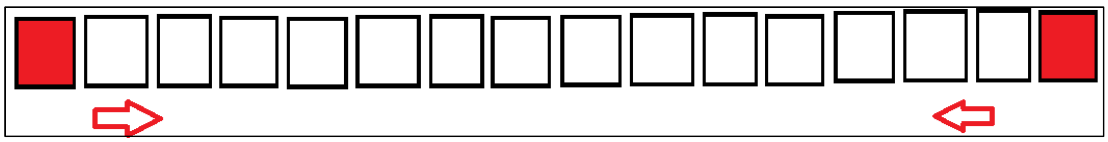
```
        else if (i == 9)
        {
GpioDataRegs.GPCDAT.all = 384;        DELAY_US(timeD);
        }
// 9번째, 8번째 LED를 timeD 동안 On 시킨다.
```

$2^{8}+2^{7}=384$

```
        else if (i == 10)
        {
GpioDataRegs.GPCDAT.all = 576;        DELAY_US(timeD);
        }
// 10번째, 7번째 LED를 timeD 동안 On 시킨다.
```

$2^{9}+2^{6}=576$

```
        else if (i == 11)
        {
GpioDataRegs.GPCDAT.all = 1056;      DELAY_US(timeD);
        }
// 11번째, 6번째 LED를 timeD 동안 On 시킨다.
   2^10 + 2^5 = 1056
        else if (i == 12)
        {
GpioDataRegs.GPCDAT.all = 2064;      DELAY_US(timeD);
        }
// 12번째, 5번째 LED를 timeD 동안 On 시킨다.
   2^11 + 2^4 = 2064
        else if (i == 13)
        {
GpioDataRegs.GPCDAT.all = 4104;      DELAY_US(timeD);
        }
// 13번째, 4번째 LED를 timeD 동안 On 시킨다.
   2^12 + 2^3 = 4104
        else if (i == 14)
        {
GpioDataRegs.GPCDAT.all = 8196;      DELAY_US(timeD);
        }
// 14번째, 3번째 LED를 timeD 동안 On 시킨다.
   2^13 + 2^2 = 8196
        else if (i == 15)
        {
GpioDataRegs.GPCDAT.all = 16386;     DELAY_US(timeD);
        }
// 15번째, 2번째 LED를 timeD 동안 On 시킨다.
   2^14 + 2^1 = 16386
        else if (i == 16)
        {
GpioDataRegs.GPCDAT.all = 32769;     DELAY_US(timeD);
        }
// 16번째, 1번째 LED를 timeD 동안 On 시킨다.
   2^15 + 2^0 = 1 + 32768 = 32769
        return 1;
}
```



GpioDataRegs.GPCSET.all = 0x00008000 >> i; & GpioDataRegs.GPCSET.all = 0x00000001 << i;

"Mode 4"의 경우 LED가 왼쪽, 오른쪽에서 출발하여 중앙으로 한 칸씩 이동하는 과정을 표현한 것이다. 위 그림의 경우는 16번째 LED와 1번째 LED가 켜진 상태로 GPCDAT은 $2^{15}+2^{0}$에 의해 32769 값을 갖는다.
이 경우 (L, R) = (i번째 LED, j번째 LED) = (16,1), (15, 2), (14,3)..... (8, 7) 의 순서쌍이 나올 수 있다.

int modeE(int i)

```
{
        if (i == 1)
        {
GpioDataRegs.GPCDAT.all = 32768; DELAY_US(timeE);
        }
// 16번째 LED를 timeE 동안 On 시킨다.
   2^15 = 32768
        else if (i == 2)
        {
GpioDataRegs.GPCDAT.all = 24576; DELAY_US(timeE);
        }
// 15번째, 14번째 LED를 timeE 동안 On 시킨다.
   2^14 + 2^13 = 24576
        else if (i == 3)
        {
GpioDataRegs.GPCDAT.all = 7168; DELAY_US(timeE);
        }
// 13번째, 12번째, 11번째 LED를 timeE 동안 On 시킨다.
   2^12 + 2^11 + 2^10 = 7168
        else if (i == 4)
        {
        GpioDataRegs.GPCDAT.all = 960; DELAY_US(timeE);
        }
// 10번째, 9번째, 8번째, 7번째 LED를 timeE 동안 On 시킨다.
   2^9 + 2^8 + 2^7 + 2^6 = 960
        else if (i == 5)
        {
        GpioDataRegs.GPCDAT.all = 62; DELAY_US(timeE);
        }
// 6번째, 5번째, 4번째, 3번째, 2번째 LED를 timeE 동안 On 시킨다.
   2^5 + 2^4 + 2^3 + 2^2 + 2^1 = 62
        else if (i == 6)
        {
        GpioDataRegs.GPCDAT.all = 1; DELAY_US(timeE);
        }
// 1번째 LED를 timeE 동안 On 시킨다.
   2^0 = 1
        return 1;
}
```

위와 같이 점등되는 LED의 개수가 1개부터 5개까지 증가하며 우측으로 이동하는 것을 LED 덧셈을 이용하여

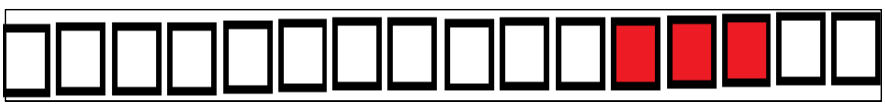
나타내었다.

int modeF(int i)

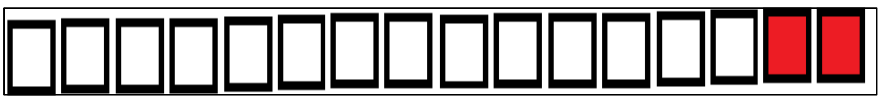
```
{
        if (i == 1)
        {
GpioDataRegs.GPCDAT.all = 32768;      DELAY_US(timeF);
        }
// 16번째 LED를 timeF 동안 On 시킨다.
   2^15 = 32768
        else if (i == 2)
        {
GpioDataRegs.GPCDAT.all = 24576;      DELAY_US(timeF);
        }
// 15번째, 14번째 LED를 timeE 동안 On 시킨다.
   2^14 + 2^13 = 24576
        else if (i == 3)
        {
GpioDataRegs.GPCDAT.all = 7168;       DELAY_US(timeF);
        }
// 13번째, 12번째, 11번째 LED를 timeF 동안 On 시킨다.
   2^12 + 2^11 + 2^10 = 7168
        else if (i == 4)
        {
GpioDataRegs.GPCDAT.all = 768;        DELAY_US(timeF);
        }
// 10번째, 9번째 LED를 timeF 동안 On 시킨다.
   2^9 + 2^8 = 768

        else if (i == 5)
        {
GpioDataRegs.GPCDAT.all = 128;        DELAY_US(timeF);
        }
// 8번째 LED를 timeF 동안 On 시킨다.
   2^7 = 128
        else if (i == 6)
        {
GpioDataRegs.GPCDAT.all = 96;         DELAY_US(timeF);
        }
// 7번째, 6번째 LED를 timeF 동안 On 시킨다.
   2^6 + 2^5 = 96
        else if (i == 7)
        {
GpioDataRegs.GPCDAT.all = 28;         DELAY_US(timeF);
        }
// 5번째, 4번째, 3번째 LED를 timeF 동안 On 시킨다.
   2^4 + 2^3 + 2^2 = 28
        else if (i == 8)
        {
GpioDataRegs.GPCDAT.all = 6→3;        DELAY_US(timeF);
        }
// 2번째, 1번째 LED를 timeF 동안 On 시킨다.
   2^1 + 2^0 = 3
        return 1;
}
```

Mode F는 점등되는 LED의 개수가 3개까지 증가와 감속을 반복하며 우측으로 이동하는 동작이다. I의 값이 7일 경우 5번째, 4번째, 3번째의 LED가 timeF동안 On되며 이때의 값은 $2^4+2^3+2^2$에 의해 28이다. 마찬가지로 I의 값이 8일 경우 2번째, 1번째의 LED가 timeF동안 On되며 이때의 값은 2^1+2^0에 의해 3이다. 하지만 기존 코드에는 그 값이 6으로 잘못 표기되어 있는 것을 확인하였다.

else if (i == 7) : GpioDataRegs.GPCDAT.all = 28;



else if (i == 8) : GpioDataRegs.GPCDAT.all = 3;



```
int modeS(int i)
{
        if (i == 1)
        {
                GpioDataRegs.GPCDAT.all = 32768;      DELAY_US(timeStop);
        } // 16번째 LED를 timeStop 동안 On 시킨다. 2^15 = 32768
        else if (i == 2)
        {
                GpioDataRegs.GPCDAT.all = 0;      DELAY_US(timeStop);
        } // 모든 LED를 timeStop 동안 Off 시킨다.
}
```

초기조건은 #define timeStop 500000 이므로, "스위치 C"를 누르면 500ms의 시간 간격으로 16번째 LED가 On, Off를 반복하며 정지 상태가 표현된다.

```
//   메인함수- 시작
void main(void)
{
        // Step 1. Disable Global Interrupt
        DINT;
        ====================================
        // Step 2. 시스템 컨트롤 초기화:
        InitSysCtrl();
====================================
        // Step 3. 인터럽트초기화:

        InitPieCtrl();
        IER = 0x0000;
        IFR = 0x0000;
        InitPieVectTable();

        // Vector Remapping
        EALLOW;
        PieVectTable.XINT3 = &Xint3_isr;
        PieVectTable.XINT4 = &Xint4_isr;
        PieVectTable.XINT5 = &Xint5_isr;
        PieVectTable.XINT6 = &Xint6_isr;
        EDIS;
```

* DINT : 칩의 활성 여부를 결정하는 전역 인터럽트 스위치를 Off하고, 칩의 비정상적인 작동을 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.

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

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

* IER = 0x0000, IFR = 0x0000 : 인터럽트를 사용하기 위한 준비단계로, 레지스터의 상태를 0으로 설정하여 혹시 모를 시스템 오동작을 방지한다.

* 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.

* PieVectTable.XINT3 = &Xint3_isr;

Default로 설정된 인터럽트 벡터를 Remapping 해준다. 위의 코드는 외부 인터럽트 3이 걸렸을 때 Xint3_isr 이라는 ISR 함수를 실행하도록 설정한 코드이다. 이를 외부인터럽트 4, 5, 6에 대해서도 지정하였다.

```
// Step 4. GPIO 초기화
        EALLOW;
        GpioCtrlRegs.GPBMUX1.bit.GPIO44 = 0;
// 핀기능선택: GPIO44
        GpioCtrlRegs.GPBMUX1.bit.GPIO45 = 0;
// 핀기능선택: GPIO45
        GpioCtrlRegs.GPBMUX1.bit.GPIO46 = 0;
// 핀기능선택: GPIO46
        GpioCtrlRegs.GPBMUX1.bit.GPIO47 = 0;
// 핀기능선택: GPIO47
        GpioCtrlRegs.GPBDIR.bit.GPIO44 = 0;
// GPIO44 입출력선택: Input
        GpioCtrlRegs.GPBDIR.bit.GPIO45 = 0;
// GPIO45 입출력선택: Input
        GpioCtrlRegs.GPBDIR.bit.GPIO46 = 0;
// GPIO46 입출력선택: Input
        GpioCtrlRegs.GPBDIR.bit.GPIO47 = 0;
// GPIO47 입출력선택: Input

        GpioCtrlRegs.GPCMUX1.all = 0x00000000;
// GPIO64-GPIO79, GPIO 기능으로설정
        GpioCtrlRegs.GPCDIR.all = 0x0000FFFF;
// GPI064-GPIO79, 출력으로설정
        GpioDataRegs.GPCDAT.all = 0x0000FFFF;
        EDIS;
```

GPIO Pin을 외부 인터럽트로 지정하기 전에 GPIO 설정을 한다.

GPIO44 / XA4	External General-Purpose I/O Port	TACT Switch 4	EMIF Port (XA4)
GPIO45 / XA5	External General-Purpose I/O Port	TACT Switch 3	EMIF Port (XA5)
GPIO46 / XA6	External General-Purpose I/O Port	TACT Switch 2	EMIF Port (XA6)
GPIO47 / XA7	External General-Purpose I/O Port	TACT Switch 1	EMIF Port (XA7)

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

Bit	Field	Value	Description
31:30	GPIO47		Configure this pin as:
		00	GPIO 47 - general purpose I/O 47 (default)
		01	Reserved
		10 or 11	XA7 - External interface (XINTF) address line 7 (O)
29:28	GPIO46		Configure this pin as:
		00	GPIO 46 - general purpose I/O 46 (default)
		01	Reserved
		10 or 11	XA6 - External interface (XINTF) address line 6 (O)
27:26	GPIO45		Configure this pin as:
		00	GPIO 45 - general purpose I/O 45 (default)
		01	Reserved
		10 or 11	XA5 - External interface (XINTF) address line 5 (O)
25:24	GPIO44		Configure this pin:
		00	GPIO 44 - general purpose I/O 44 (default)
		01	Reserved
		10 or 11	XA4 - External interface (XINTF) address line 4 (O)

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

* 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.

* GPCDIR : GpioCtrlRegs.GPCDIR.all = 0x0000FFFF;(16진수 이용)

Table 64. GPIO Port C Direction (GPCDIR) Register Field Descriptions

Bits	Field	Value	Description [1]
31-0	GPIO87-GPIO64		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

- GPIO핀의 입출력을 선택하는 레지스터로, "0"을 입력하면 GPIO핀을 입력으로 이용하고, "1"을 입력하면, GPIO핀을 출력으로 이용한다는 의미이다. 해당 예제에서 1을 입력함으로써 GPIO 64~79 핀을 출력으로 설정한다.

```
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;
```

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 × $T_{SYSCLKOUT}$
		0x02	Sampling Period = 4 × $T_{SYSCLKOUT}$
		...	...
		0xFF	Sampling Period = 510 × $T_{SYSCLKOUT}$

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

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.

* GPBQSEL: Input qualification type을 선택한다. 샘플의 사용 횟수를 정하는데, 1값을 입력할 경우 3개의 샘플을 이용하고 2값을 입력할 경우 6개의 샘플을 이용한다. 해당 예제에서는 2값을 입력했기 때문에 6개의 샘플을 qualification에 이용한다.

// Step 5. XINT 초기화

```
EALLOW;
        GpioIntRegs.GPIOXINT3SEL.bit.GPIOSEL = 47;  // 외부인터럽트XINT3로사용할핀선택: GPIO47
        GpioIntRegs.GPIOXINT4SEL.bit.GPIOSEL = 46;  // 외부인터럽트XINT4로사용할핀선택: GPIO46
        GpioIntRegs.GPIOXINT5SEL.bit.GPIOSEL = 45;  // 외부인터럽트XINT5로사용할핀선택: GPIO45
        GpioIntRegs.GPIOXINT6SEL.bit.GPIOSEL = 44;  // 외부인터럽트XINT6로사용할핀선택: GPIO44
EDIS;

        XIntruptRegs.XINT3CR.bit.POLARITY = 2;      // XINT3 인터럽트발생조건설정: 입력신호의 하강엣지
        XIntruptRegs.XINT4CR.bit.POLARITY = 1;      // XINT4 인터럽트발생조건설정: 입력신호의 상승엣지
        XIntruptRegs.XINT5CR.bit.POLARITY = 1;      // XINT5 인터럽트발생조건설정: 입력신호의 하강엣지
        XIntruptRegs.XINT6CR.bit.POLARITY = 1;      // XINT6 인터럽트발생조건설정: 입력신호의 하강& 상승엣지

        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
```

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

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

Table 121. External Interrupt *n* Control Register (XINT*n*CR) 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 XINT*n*.
		0	Disable interrupt
		1	Enable interrupt

* Polarity: 입력값에 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.

* Enable: 1값을 입력하게 되면 인터럽트를 enable한다.

```
// 외부인터럽트 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
```

Table 113. PIE Configuration and Control Registers

Name	Address	Size (x16)	Description
PIEIER12	0x0000 - 0CF8	1	PIE, INT12 Group Enable Register
PIEIFR12	0x0000 - 0CF9	1	PIE, INT12 Group Flag Register

* 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 6. Initialize Application Variables
        SW1_cnt = 0;
        SW2_cnt = 0;
        SW3_cnt = 0;
        SW4_cnt = 0;
        Loop_cnt = 0;
```

앞서 선언해준 변수 SW1, SW2, SW3, SW4, Loop_cnt의 값을 초기화해준다.

```
// Enable global Interrupts and higher priority real-time debug events:

        EINT;   // Enable Global interrupt INTM
        ERTM;   // Enable Global realtime interrupt DBGM
```

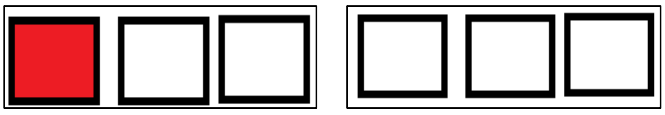
* EINT: 앞의 DINT와 반대로 전역 인터럽트를 활성화 시킨다.
* ERTM: DBGM을 clear하여 디버그 이벤트를 Enable 시킨다.

// IDLE loop. Just sit and loop forever :

```
for (;;)
{
        if (stop == 1)
        {
                for (i = 1; i < 3; i++)
                {
        if (stop != 1) { break; }
                        modeS(i);
                }
        }
        else if (mode == 1)
        {
                for (i = 1; i < 17; i++)
                {
        if (mode != 1 || stop != 0) { break; }
                        modeA(i);
                }
        }
        else if (mode == 2)
        {
                for (i = 1; i < 137; i++)
                {
        if (mode != 2 || stop != 0) { break; }
                        modeB(i);
                }
        }
        else if (mode == 3)
        {
                for (i = 1; i < 3; i++)
                {
        if (mode != 3 || stop != 0) { break; }
                        modeC(i);
                }
        }
        else if (mode == 4)
        {
                for (i = 1; i < 17; i++)
                {
        if (mode != 4 || stop != 0) { break; }
                        modeD(i);
                }
        }
        else if (mode == 5)
        {
                for (i = 1; i < 7; i++)
                {
        if (mode != 5 || stop != 0) { break; }
                        modeE(i);
                }
        }
        else if (mode == 6)
        {
                for (i = 1; i < 9; i++)
                {
        if (mode != 6 || stop != 0) { break; }
                        modeF(i);
                }
        }
        else if (mode >= 6)
        {
                mode = 1;
        }
        else if (mode <= 1)
        {
                mode = 6;
        }
        Loop_cnt++;
    }
}
```

해당 코드는 Stop과 Mode로 구분되기 때문에, Stop이 1이 되는 경우를 if 문으로 설정하고, 그 이외의 값을 else if로 설정하여 Stop이 아닐 경우 Mode가 동작하도록 설정하였다. Stop에 “1”이 입력되면, 500ms의 시간 간격으로 16번째 LED가 On, Off를 반복하며 정지 상태가 표현되는 ModeS가 실행된다.

for(;;)문을 통해, 무한 반복문임을 알 수 있는데, “i++”을 통해, 반복문이 진행될 때마다 i의 값이 늘어나고, 그 값에 따라 LED의 변화가 나타나게 된다. Stop Mode는 맨 왼쪽의 LED만 On, Off가 반복하기 때문에 필요한 Led가 2개이다. 따라서, “i<3”라고 설정하였다.



On Off

ModeS의 탈출조건은 “if(stop !=1) {break;}” 이다. 이는 “stop = 0” 일 때를 의미한다.
Stop 모드 이후에는 동일한 방법으로 코드가 실행된다. Mode의 값이 “1, 2, 3, 4, 5, 6”여부에 따라 실행되는 동작은 다르고, 그 값에 따라 해당하는 Mode “A, B, C, D, E, F”가 진행된다.

```
else if (mode == 1)
{
        for (i = 1; i < 17; i++)
        {
if (mode != 1 || stop != 0) { break; }
                modeA(i);
        }
}
```

탈출조건은 자신에 해당하는 mode와 다른 값이 입력되었을 때 혹은 stop의 값이 0이 아닐 때이다.

```
    else if (mode >= 6)
    {
            mode = 1;
    }

    else if (mode <= 1)
    {
            mode = 6;
    }

    Loop_cnt++;
```

위 코드는 동작을 결정하는 mode가 1부터 6사이의 값을 벗어나지 않도록 설정해준 코드이다. 만약 mode의 값이 6일 때, 버튼 A로 인해 mode의 값이 증가되면 mode1이 실행된다. 혹은 mode의 값이 1일 때, 버튼 B로 인해 mode의 값이 감소되면 mode6이 실행된다.

```
//   ISR 함수정의
interrupt void Xint3_isr(void)
{
        SW1_cnt++;
        mode++;
        stop = 0;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}

interrupt void Xint4_isr(void)
{
        SW2_cnt++;
        mode--;
        stop = 0;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
```

```
interrupt void Xint5_isr(void)
{
        SW3_cnt++;
        stop = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}

interrupt void Xint6_isr(void)
{
        SW4_cnt++;
        stop = 0;
        mode = 1;
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
```

* ISR: Interrupt Service Routine의 약자로, 인터럽트가 발생할 때 수행할 callback 함수이다.

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

Bits	Field	Value	Description
15-12	Reserved		Reserved
11-0	PIEACK		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
		bit x = 0 (1)	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.
		bit x = 1	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

* PIEACK: PIE controller를 거친 인터럽트 신호를 한 번에 통제하기 위한 장치이다. PIEIER을 거친 신호를 다음 단계로 전달하기 위해서는 PIEACK에 1값을 write해야 한다. 왜냐하면 동작이 Active Low이기 때문에 1을 write함으로써 해당 영역이 0으로 clear되기 때문이다. 인터럽트 요청 신호가 PIEACK를 통과하고 나면 하드웨어적으로 PIEACK는 1로 set되기 때문에 다음 인터럽트 신호가 통과하지 못하게 된다. 따라서 인터럽트를 사용할 때는 인터럽트 서비스 루틴의 마지막 부분에 PIEACK 레지스터를 Clear 해주어야 한다.

ISR함수 정의는 Xint5와 Xint6은 정지와 동작을 의미한다. Xint5의 경우 stop모드가 실행되기 위해, "stop = 1"이 되고, Xint6의 경우 동작을 하기 위해, "stop = 0", "mode = 1"이 된다.

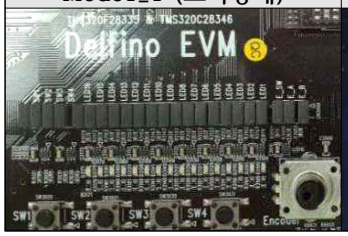
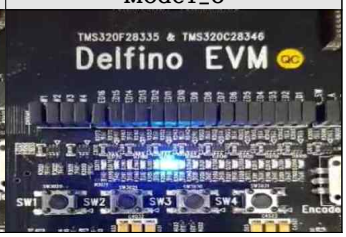
Xint3과 Xint4는 mode값의 증가 감소이기 때문에 mode의 값을 변화시키는 "mode++", "mode--"으로 표현하였다. 또한 동작 동안에는 stop이 일어나면 안되기 때문에 "stop=0"으로 설정하였다.

2. 실험 결과

Mode 1.

• **실험 목표**: 점등되는 1개의 LED가 우측으로 이동하도록 한다.

• **실험 분석**:

Mode1_1 (초기상태)	Mode1_2 (초기상태)	Mode1_3	Mode1_4
			
Mode1_5	**Mode1_6**	**Mode1_7**	**Mode1_8**
			
Mode1_9	**Mode1_10**	**Mode1_11**	**Mode1_12**
			
Mode1_13	**Mode1_14**		
			

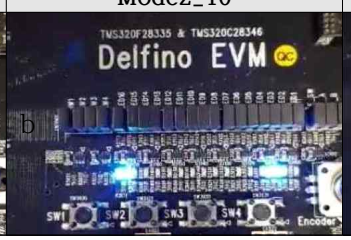
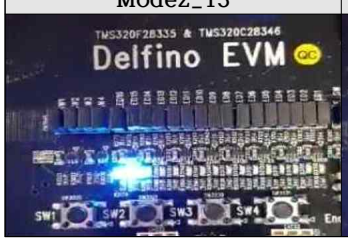
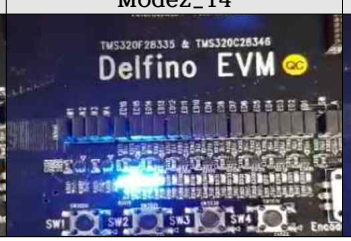
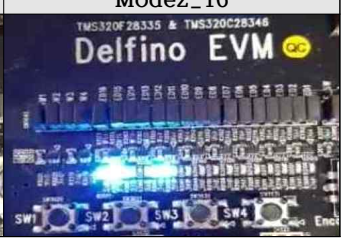
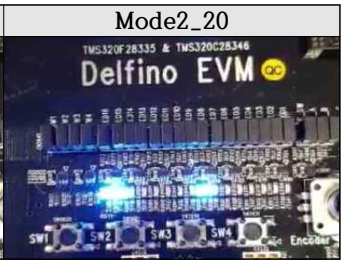
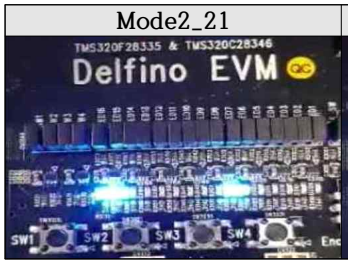
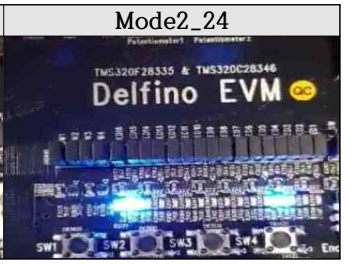
초기 상태에는 GPIO79번인 16번째 LED가 500ms 시간 간격을 가진 채 On, Off를 반복하여 정지 상태를 표현한다. 이 상태에서 스위치 D를 눌러주어 Mode1의 동작을 시작한다.

모드 1은 마치 한 개의 LED가 왼쪽부터 오른쪽까지 이동하는 것처럼 보이게 설정해준 것이다. 이때의 시간 간격은 1초이며, 첫 번째 LED인 GPIO64가 켜진 이후에는 다시 초기 상태로 돌아와 GPIO79번부터 켜지게 된다. 이렇게 GPIO79번부터 GPIO64번까지 LED가 차례대로 On되며 동작이 반복된다.

Mode 2.

• **실험 목표**: 점등되는 1개의 LED가 우측으로 이동한 뒤, 돌아와서 비어있는 LED를 왼쪽부터 채운다. 모든 LED가 점등될 때까지 반복한다.

• **실험 분석**:

Mode2_1 (초기상태)	Mode2_2 (초기상태)	Mode2_3	Mode2_4
			
Mode2_5	Mode2_6	Mode2_7	Mode2_8
			
Mode2_9	Mode2_10	Mode2_11	Mode2_12
			
Mode2_13	Mode2_14	Mode2_15	Mode2_16
			
Mode2_17	Mode2_18	Mode2_19	Mode2_20
			
Mode2_21	Mode2_22	Mode2_23	Mode2_24
			

Mode2_25	Mode2_26	Mode2_27	Mode2_28
			

b

Mode2_29	Mode2_30	Mode2_31	Mode2_32
			

Mode2_33	Mode2_34
	

모드 2는 모드 1에서 Mode 증가 기능을 하는 스위치 A버튼을 눌러서 실행시키거나 모드 3에서 Mode 감소 기능을 하는 스위치 B버튼을 눌러서 실행시킬 수 있다. 모드 2에서는 500ms의 시간 간격으로 점등되는 1개의 LED가 우측으로 이동한 뒤, 돌아와서 비어있는 LED를 왼쪽부터 채운다. GPIO64~79 번의 16개 LED가 모두 켜질 때까지 동작이 진행되며 이 이후에는 다시 초기 상태로 돌아가서 Mode2_2부터 Mode2_34까지 반복된다.

Mode 3.

• **실험 목표**: 홀수 번째 LED 점등 → 짝수 번째 LED 점등 → 홀수 번째 LED 점등 …과 같은 과정이 반복된다.

• **실험 분석**:

Mode3_1 (초기상태)	Mode3_2 (초기상태)	Mode3_3	Mode3_4
			

Mode3_5	Mode3_6
	

모드 3은 모드 2에서 Mode 증가 기능을 하는 스위치 A버튼을 눌러서 실행시키거나 모드 4에서 Mode 감소 기능을 하는 스위치 B버튼을 눌러서 실행시킬 수 있다. 모드 3에서는 500ms의 시간 간격으로 홀수 번째 LED와 짝수 번째 LED가 번갈아 켜진다. 여기서 홀수는 GPIO 79, 77, 75, 73, 71, 69, 67, 65를 의미하며 짝수는 GPIO 78, 76, 74, 72, 70, 68, 66, 64를 의미한다.

Mode 4.

• **실험 목표**: 양 끝에서 점등된 LED가 중앙으로 이동하여 만난 뒤, 다시 양 끝으로 이동한다.

• **실험 분석**:

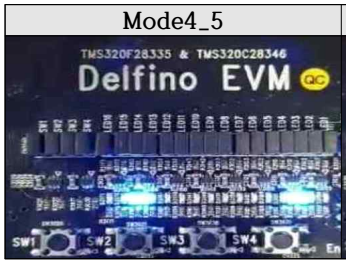
Mode4_1 (초기상태)

Mode4_2 (초기상태)

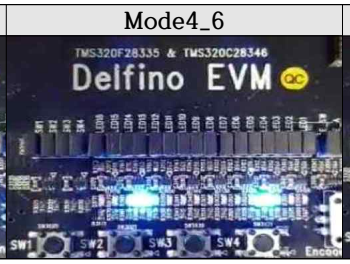
Mode4_3

Mode4_4

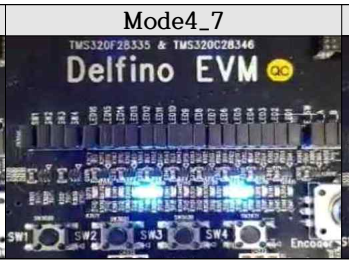
Mode4_5



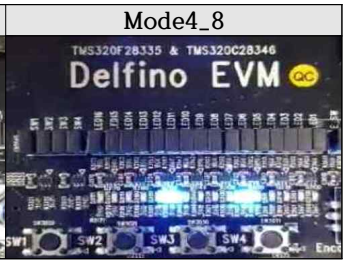
Mode4_6



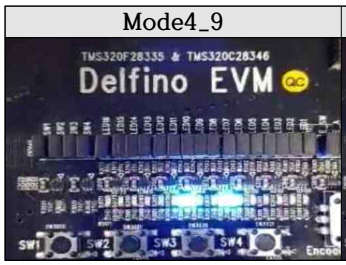
Mode4_7



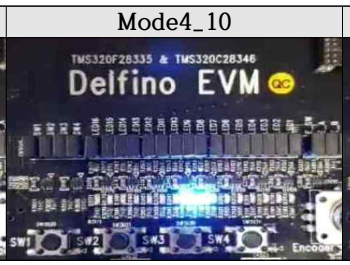
Mode4_8



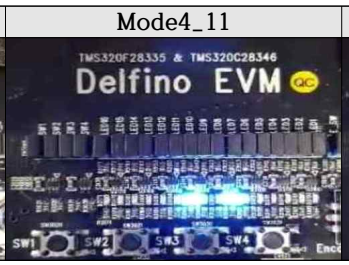
Mode4_9



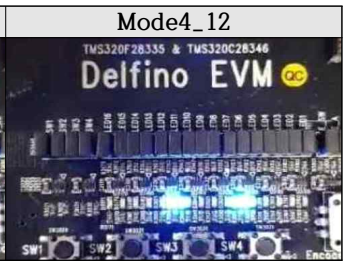
Mode4_10



Mode4_11



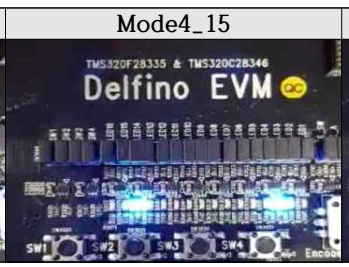
Mode4_12



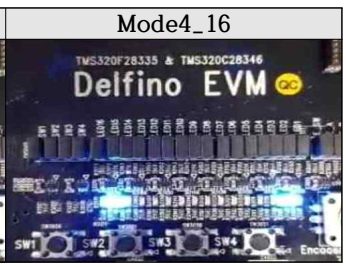
Mode4_13

Mode4_14

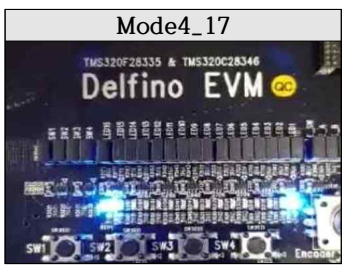
Mode4_15



Mode4_16



Mode4_17

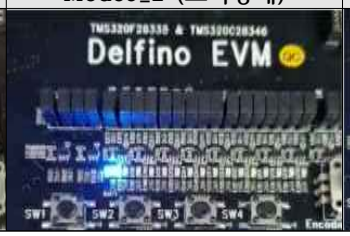
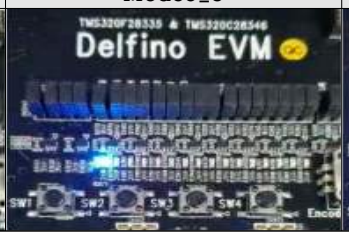
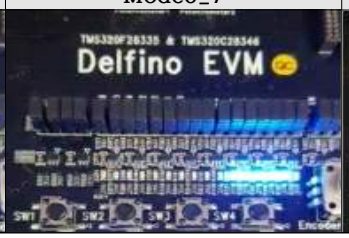
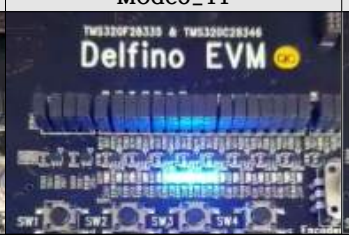
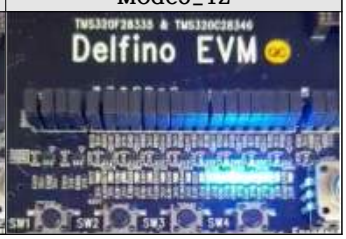
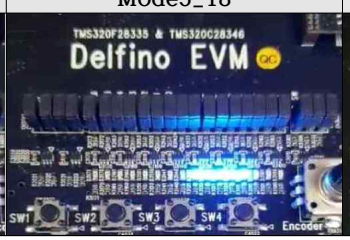
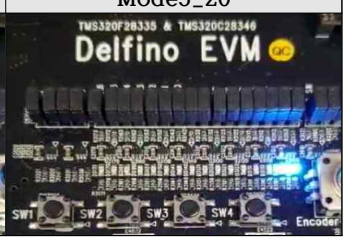
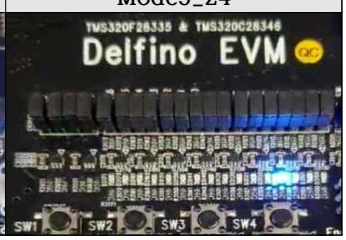


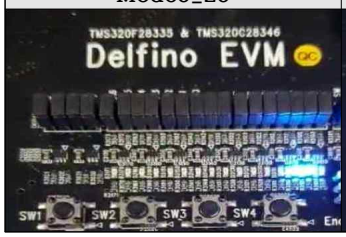
모드 4는 모드 3에서 Mode 증가 기능을 하는 스위치 A버튼을 눌러서 실행시키거나 모드 5에서 Mode 감소 기능을 하는 스위치 B버튼을 눌러서 실행시킬 수 있다. 모드 4에서는 양 끝에서 점등된 LED가 1초 간격으로 서로 양 끝으로 이동한다. 즉, GPIO 79번은 오른쪽 끝으로 이동하고 GPIO 64번은 왼쪽 끝으로 이동하게 된다. 이후 같은 동작을 반복한다.

Mode 5.

• **실험 목표**: 점등되는 LED 개수가 1개부터 5개까지 증가하며 우측으로 이동한다.

• **실험 분석**:

Mode5_1 (초기상태)	Mode5_2 (초기상태)	Mode5_3	Mode5_4
			
Mode5_5	**Mode5_6**	**Mode5_7**	**Mode5_8**
			
Mode5_9	**Mode5_10**	**Mode5_11**	**Mode5_12**
			
Mode5_13	**Mode5_14**	**Mode5_15**	**Mode5_16**
			
Mode5_17	**Mode5_18**	**Mode5_19**	**Mode5_20**
			
Mode5_21	**Mode5_22**	**Mode5_23**	**Mode5_24**
			

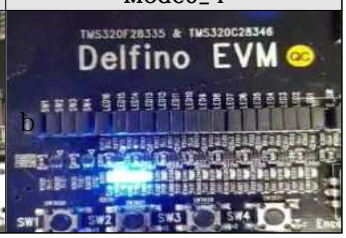
Mode5_25	Mode5_26
	

모드 5는 모드 4에서 Mode 증가 기능을 하는 스위치 A버튼을 눌러서 실행시키거나 모드 6에서 Mode 감소 기능을 하는 스위치 B버튼을 눌러서 실행시킬 수 있다. 모드 5에서는 점등되는 LED의 개수가 1개부터 5개까지 1초 간격으로 증가하며 우측으로 이동한다. 즉, 처음에는 GPIO 79번만 켜지고 1초 뒤에는 GPIO 78, 77번이 켜지고, 또 1초 뒤에는 GPIO 76, 75, 74번이 켜진다. 이러한 동작을 반복하면서 LED가 5개까지 켜진 뒤에는 다시 1개부터 점등을 시작한다.

Mode 6.

• **실험 목표**: 점등되는 LED 개수가 3개까지 증가와 감소를 반복하며 우측으로 이동한다.

• **실험 분석**:

Mode6_1 (초기상태)	Mode6_2 (초기상태)	Mode6_3	Mode6_4
			

Mode6_5	Mode6_6
	

모드 6은 모드 5에서 Mode 증가 기능을 하는 스위치 A버튼을 눌러서 실행시킬 수 있다. 모드 6에서는 점등되는 LED의 개수가 1개, 2개, 3개, 2개, 1개 순서로 증가와 감소를 반복하며 1초의 시간 간격을 갖고 우측으로 이동한다. 즉, 처음에는 GPIO 79번만 켜지고 1초 뒤에는 GPIO 78, 77번이 켜지고, 또 1초 뒤에는 GPIO 76, 75, 74번이 켜지고, 또 1초 뒤에는 GPIO 73, 72번이 켜진다.

3. 실험 고찰

이번 프로젝트에서는 기존 실험에서 사용했던 예제 코드 GPIO_Input_Qual 코드와 ADC_1ch_Timer_SOC_Interrupt 코드를 활용하여 LED의 점등 변화를 모드 별로 동작하도록 하였다. 입력으로 사용할 GPIO 44번부터 47번을 각각 스위치 D, C, B, A라고 하고 각각의 스위치에 역할을 정해주었다. 스위치 A는 Mode를 증가하는 역할, 스위치 B는 Mode를 감소하는 역할, 스위치 C는 정지 역할, 스위치 D는 시작 역할을 의미한다. 정지의 경우 LED 하나를 번갈아 점등하여 정지 상태를 표현하며, 시작의 경우 스위치 D를 누르면 Mode 1의 동작이 시작된다. 각 모드의 동작과 시간 간격은 아래 표와 같이 설정해주었다.

모드	동작	시간 간격
Mode 1	점등되는 1개의 LED가 우측으로 이동한다.	1s
Mode 2	점등되는 1개의 LED가 우측으로 이동한 뒤, 돌아와서 비어있는 LED를 왼쪽부터 채운다. 모든 LED가 점등될 때까지 반복한다.	500ms
Mode 3	홀수 번째 LED 점등 → 짝수 번째 LED 점등 → 홀수 번째 LED 점등 … 위 과정을 반복한다.	500ms
Mode 4	양 끝에서 점등된 LED가 중앙으로 이동하여 만난 뒤, 다시 양 끝으로 이동한다.	1s
Mode 5	점등되는 LED 개수가 1개부터 5개까지 증가하며 우측으로 이동	1s
Mode 6	점등되는 LED 개수가 3개까지 증가와 감소를 반복하며 우측으로 이동	1s

프로젝트는 300줄로 작성한 코드와 1000줄로 작성한 두 가지의 코드를 이용하여 진행되었다. 두 코드의 차이를 Mode 1을 예로 들어서 설명하면 다음과 같다.

300줄 코드의 경우 우측으로 LED 1개를 이동하는 코드로 GpioDataRegs.GPCSET.all = 0x00008000 >> i;을 사용하였다. 여기서 '>>'는 Shift 연산을 의미하며 i의 값이 증가함에 따라 점등된 1개의 LED가 우측으로 이동하게 된다. i의 값이 16이 되면 i는 0으로 초기화되며 break 조건이 걸리기 전까지 동작이 반복된다.

1000줄로 작성한 코드는 GPCDAT 레지스터로 각각의 LED를 2진수로 표현했다. 총 16개의 LED가 있기 때문에 GPCDAT의 값은 2^0~2^{15}의 덧셈으로 계산을 하여 해당 LED를 점등시켜주었다. 따라서 Mode 1을 동작시키기 위해서 아래의 동작을 반복한다.

GpioDataRegs.GPCDAT.all = 32768; DELAY_US(timeA); // 16번째 LED를 timeA 동안 On 시킨다. $2^{15} = 32768$
GpioDataRegs.GPCDAT.all = 16384; DELAY_US(timeA); // 15번째 LED를 timeA 동안 On 시킨다. $2^{14} = 16384$
…
GpioDataRegs.GPCDAT.all = 1; DELAY_US(timeA); // 1번째 LED를 timeA 동안 On 시킨다. $2^0 = 1$

즉, 300줄을 이용한 코드는 Shift 연산을 이용하여 코드를 효율적으로 짤 수 있었으며, 1000줄을 이용한 코드는 각각의 LED에 값을 부여해야하기 때문에 코드의 줄 수가 많아질 수밖에 없었다. 1000줄의 코드를 이용하게 되면 CCS 메모리 문제로 인한 빌드 오류가 나타날 수 있다. 이를 해결하기 위해서는 'Project Explorer'에서 'F28335_RAM.cmd'를 클릭한 뒤, 저장 공간을 늘려주면 해결된다. 코드 변경을 통해 저장 공간을 늘려줄 수 있으며 해당 코드는 아래와 같다.

```
9    RAML0     : origin = 0x008000, length = 0x003000     /* on-chip RAM block L0 */
10   RAML1     : origin = 0x00C000, length = 0x001000     /* on-chip RAM block L1 */
11   RAML2     : origin = 0x00D000, length = 0x001000     /* on-chip RAM block L2 */
12   RAML3     : origin = 0x00F000, length = 0x001000     /* on-chip RAM block L3 */
```

4. 참고문헌

- Shift 연산, https://dojang.io/mod/page/view.php?id=174
- DSP 보드, http://www.mcublog.co.kr/
- 데이터 시트, TMS320F28335 Datasheet