프로젝트1_코딩(300줄1)

```
#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 enable한다.

```
//  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.

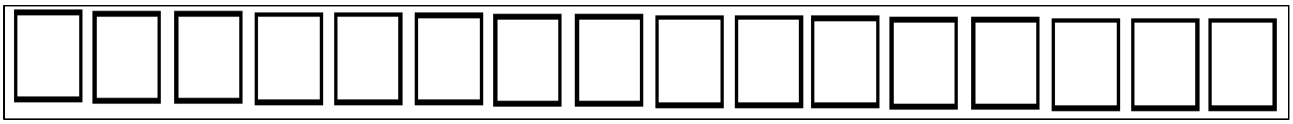
㉥ **IER |= M_INT12**: CPU 인터럽트를 **Enable 상태로 설정**해준다.

```
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
   }
}
```

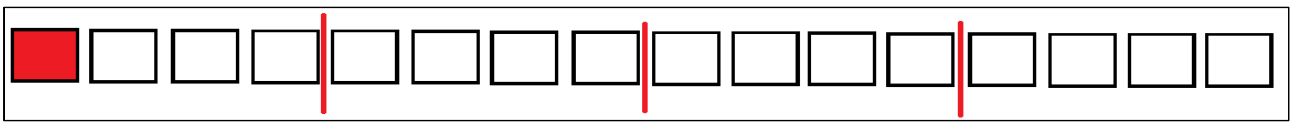
```
interrupt void Xint3_isr(void)          // (+) 인터럽트 실행
{
    SW1_cnt++;
    mode++;
    if(mode > 6 )
    {mode = 1;}
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
}
interrupt void Xint4_isr(void) // (-) 인터럽트 실행
{
    SW2_cnt++;
       mode--;
       if(mode <1 )
       {mode = 6;}
    PieCtrlRegs.PIEACK.all = 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)

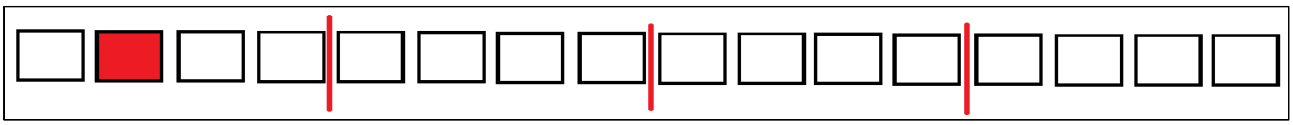
```
{
    i = 0;
    for(;;)
    {
        GpioDataRegs.GPCCLEAR.all = 0x0000FFFF;
        GpioDataRegs.GPCSET.all = 0x00008000 >> i;

         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

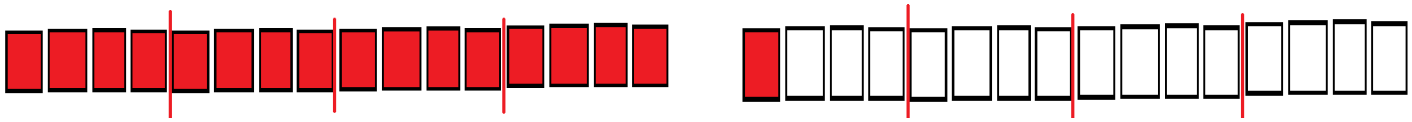
Bits	Field	Value	Description
31-24	Reserved		Reserved
23-0	GPIO87-GPIO64		Each GPIO port C pin (GPIO64-GPIO87) corresponds to one bit in this register as shown in Figure 70.
		0	Writes of 0 are ignored. This register always reads back a 0.
		1	Writing a 1 forces the respective output data latch to high. 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;
    }
}
```

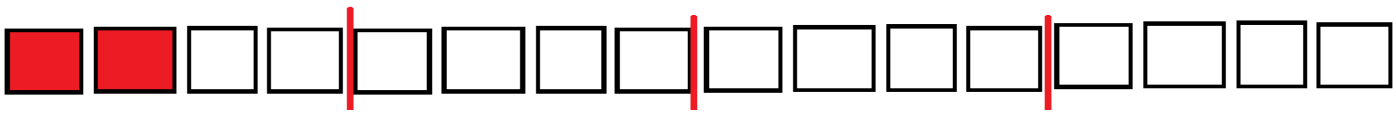
"GpioDataRegs.GPCSET.all = 0x0000FFFF;"에 의해, 모든 LED가 "On"이 된다. for문이 실행되면, I, j의 값은 0이기에,



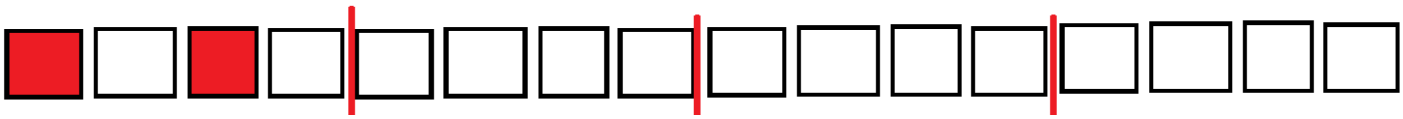
GpioDataRegs.GPCSET.all = 0x0000FFFF; => GpioDataRegs.GPCCLEAR.all = 0x00007FFF >> i;

"GpioDataRegs.GPCSET.all = (0x00008000 >> i) >> j;"의 동작도, "i, j"의 값이 모두 0이기에, 1번째 Led만 동작.

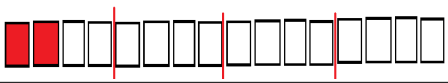
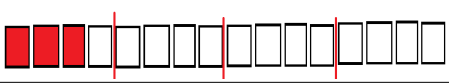
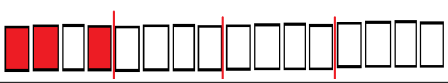
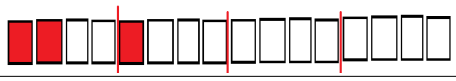
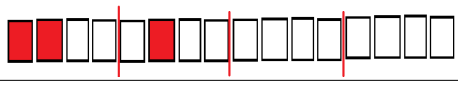
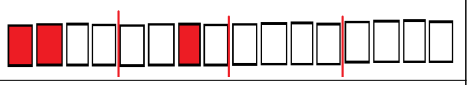
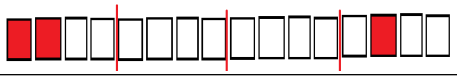
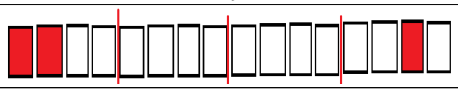
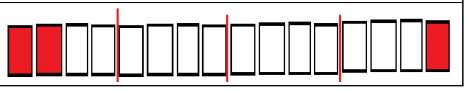
반복문이 실행되어, j=1이 되면, 다음과 같이 동작. 레지스터 Clear의 경우, "i=0"에 의해, 그대로이지만, Set의 경우, "GpioDataRegs.GPCSET.all = (0x00008000 >> 0) >> 1;"에 의해, 1칸 이동하게 된다. 다음과 같이 led가 동작한다.



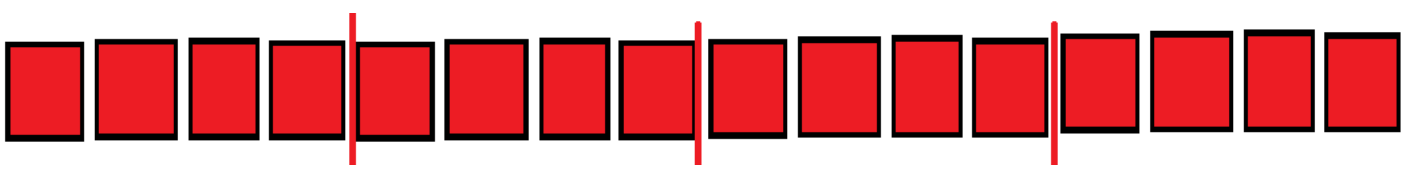
반복문이 실행되어, j=2이 되면, 다음과 같이 동작. 레지스터 Clear의 경우, "i=0"에 의해, 그대로이지만, Set의 경우, "GpioDataRegs.GPCSET.all = (0x00008000 >> 0) >> 2;"에 의해, 2칸 이동하게 된다. 다음과 같이 led가 동작한다.



이와 같은 방식으로 동작이 계속 반복된다.

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"

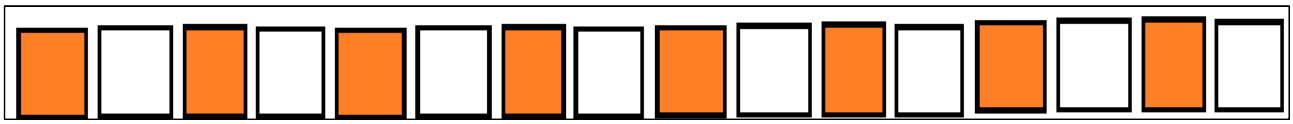


```
void Printmode_3(void)
{
    GpioDataRegs.GPCSET.all = 0x0000AAAA;
    GpioDataRegs.GPCCLEAR.all = 0x00005555;
    for(;;)
    {
        DELAY_US(usec_delay);
        GpioDataRegs.GPCTOGGLE.all = 0x0000FFFF;
        if((mode != 3)||(state == 0))
            break;
    }
}
```

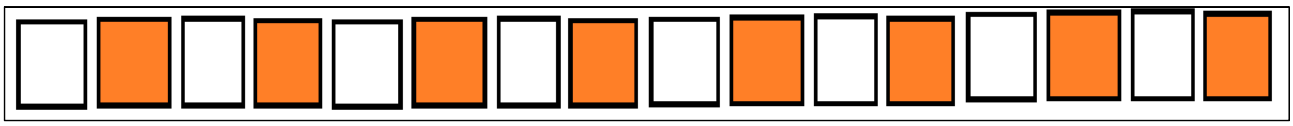
Table 79. GPIO Port C Toggle (GPCTOGGLE) Register Field Descriptions

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”가 되도록 반복 동작.
처음에는 레지스터“Set”에 의해, ㉠과 같이 led가 작동하고 있다. “toggle”을 통해, “Set => Clear”, “Clear => Set”로 동작을 하여, **“Mode3”의 수행 조건인 “번갈아 LED 점등” 수행.**

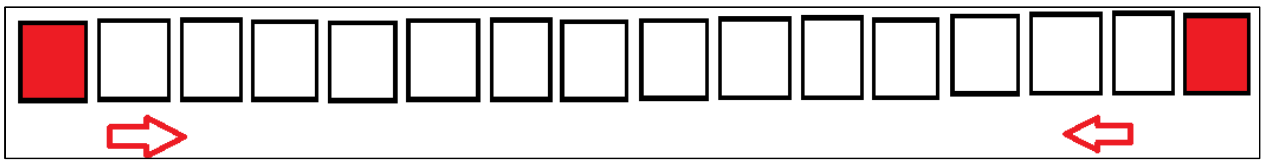


㉠ GpioDataRegs.GPCSET.all = 0x0000AAAA;



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 off, 동작 시, 1, 16번째 led가 켜지면서, 동작 시작.

"Mode 4"의 경우, 양쪽에서 Led는 각자의 반대 방향으로 이동하는 것이다. 왼쪽에서 시작한 것은 오른쪽 끝으로 이동하는 것처럼. 이를 위해, Set을 통해 "0x00008000"은 왼쪽 1번째 led가 켜지도록 하였고, "0x00000001"은 오른쪽 1번째 led가 켜지도록 하였다.

led의 불이 이동하도록 ">>, <<"이라는 시프트 연산자를 사용하였다. 따라서, 위의 코드를 보면 ">>i"를 통해, 반복되는 횟수만큼(반복이 될수록) led가 한 칸씩 움직이는 것처럼 보이도록 설정하였다.

그리고 "if(i == 16)"문을 통해, 왼쪽에서 출발한 led가 오른쪽 끝에 도착하면, 다시 처음으로 시작하는 것을 설정했다.

출처 : https://dojang.io/mod/page/view.php?id=174
시프트 연산은 **"변수 << 이동할 비트 수"** 또는 **"변수 >> 이동할 비트 수"** 형식으로 사용합니다.

즉, 지정한 횟수대로 비트를 이동시키며 모자라는 공간은 0으로 채웁니다.
연산자 모양 그대로 <<는 왼쪽, >>는 오른쪽 방향입니다.

num1 << 3은 0000 0011을 왼쪽으로 3번 이동하므로 0001 1000이 되고, 10진수로 24입니다.

```
0000 0011(3)
_________ << 3
0001 1000(24)
```

num2 >> 2는 0001 1000이 오른쪽으로 2번 이동하므로 0000 0110이 되고, 10진수로 6입니다.

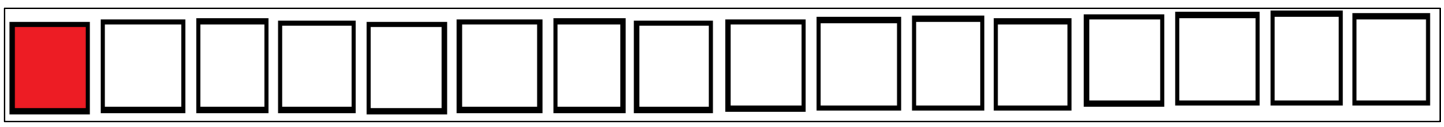
```
0001 1000(24)
_________ >> 2
0000 0110(6)
```

3 << 3은 $3 * 2^3$과 같으므로 24가 되고, 24 >> 2는 $24 / 2^2$과 같으므로 6이 됩니다. 즉, 시프트 연산 <<은 2의 거듭제곱을 곱하기, >>은 2의 거듭제곱을 나누기입니다.

"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;
      }
}
```

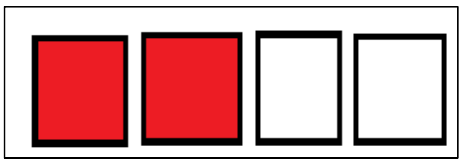
Case 0(I=0, num=0)



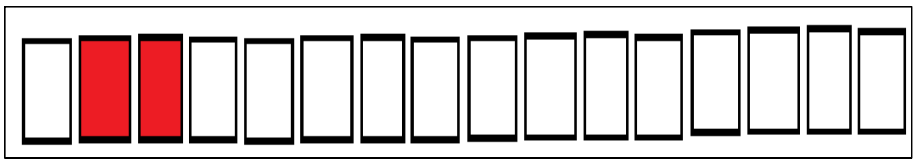
초기조건이 I=0, num=0 이기에, "case 0"이 실행된다. 그렇게 되면 위와 같이, 1번째 led가 점등되고, delay 이후에 그다음 동작이 되는데, "I++"에 의해, i=1이 되고, "num"의 값은 1이 된다.

Case 1(i=1, num=1)

"Case 1"이 시작되면, 0x0000C000에 의해, led가 점등이 되어야 하는데, ">>"라는 시프트 연산에 의해, 1칸 이동한 ㉠과 같이 동작을 하게 된다. 그리고 동작이 끝나면, "I++"에 의해, i=2가 되고, "num"의 값은 "2+1=3"이 된다.



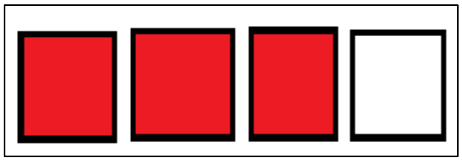
0x0000C000



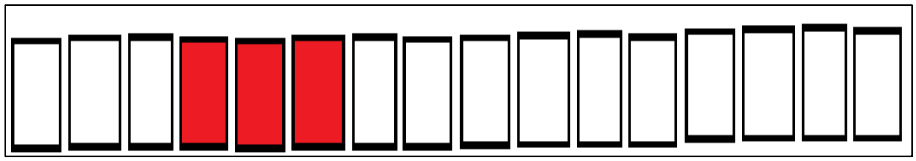
㉠0x0000C000 >> num;

Case 2(i=2, num=3)

"Case 2"이 시작되면, 0x0000E000에 의해, led가 점등이 되어야 하는데, ">>"라는 시프트 연산에 의해, 3칸 이동한 ㉡과 같이 동작하고, "I++"에 의해, i=3가 되고, "num"의 값은 "3+3=6"이 된다. 동작이 계속 반복된다.



0x0000E000



㉡0x0000E000 >> num;

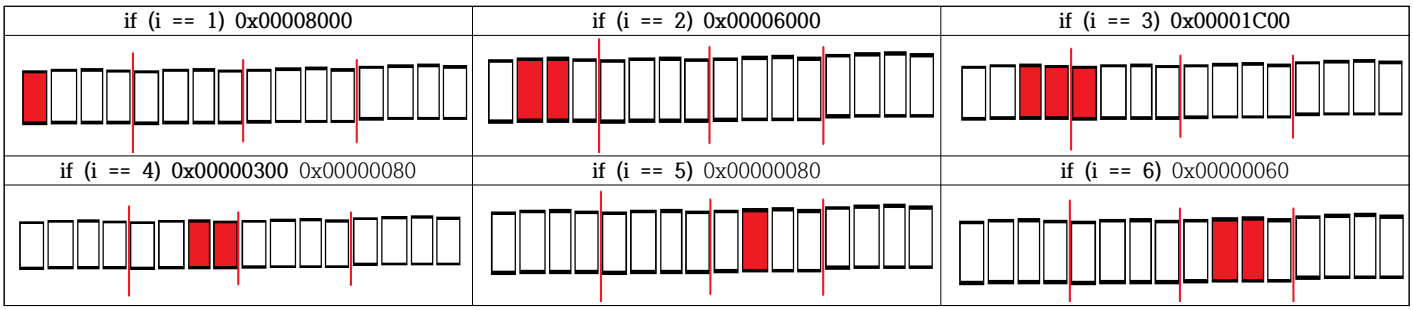
case 4까지 진행되면, break가 되어, 다시 case 0부터 case4까지 차례대로 작동이 된다.

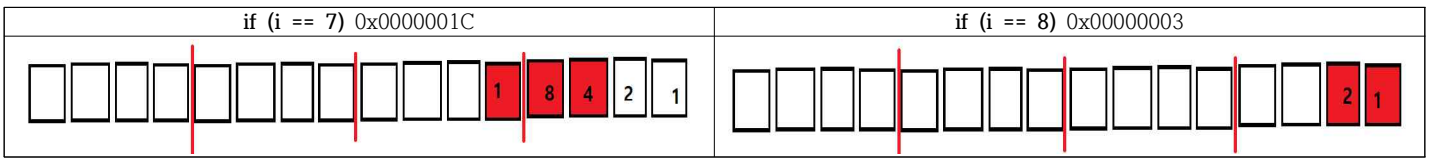
```
void Printmode_6(void)
{
      i++;
    if (i == 1)
      {
          GpioDataRegs.GPCDAT.all = 0x00008000;
           DELAY_US(usec_delay_1);
      }

      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);
      }
}
```

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.

초기조건은 I가 "0"이지만, "i++"에 의해 1이 되어 "if(i ==1)" 구문이 실행 된다.
동작 내용은 16진수에 의한 1번째 Led가 "On"이 되고, delay시간 후에, 다음 동작으로 진행한다. 코드 자체가 하나의 if문으로 구성되어 있기에, 해당 if문이 실행 후, 다시 "i++"로 되돌아간다.

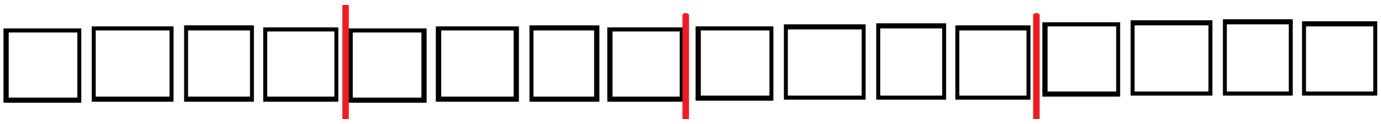




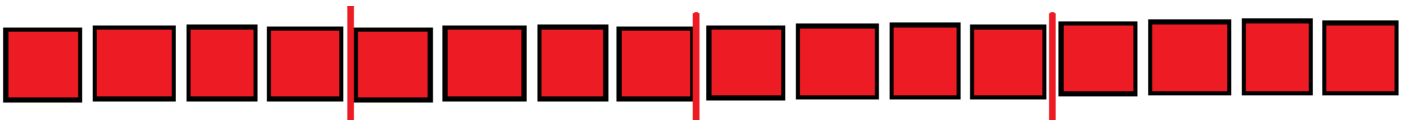
다음과 같은 순서로 "i++"에 의해 달라지는 I값에 따라 서로 다른 if문이 실행된다.

```
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; => off였던 Led를 on으로 반전시킨다.



일정 시간 delay된 후, 하나의 사이클의 동작이 끝난다. 이런 동작이 계속 반복된다.