I DSP,Xilinx zynq FPGA,MCU 및 Xilinx zynq FPGA 프로그래밍 전문가 과정

강사-INNOVA LEE(이상훈)

Gccompil3r@gmail.com

학생-윤지완

Yoonjw789 @naver.com

<_coreInitStackPointer_(); c 코드 이부분의 어셈블리어 해석)

```
mov r0, lr
        mov r1, #0x0000
        mov r2, #0x0000
        mov r3, #0x0000
        mov r4, #0x0000
        mov r5, #0x0000
        mov r6, #0x0000
        mov r7, #0x0000
        mov r8, #0x0000
        mov r9, #0x0000
        mov r10, #0x0000
        mov r11, #0x0000
        mov r12, #0x0000
        mov r13, #0x0000
        mrs r1, cpsr // cpsr 값을 r1 에다가 넣는다.spsr(special register)
        msr spsr_cxsf, r1 //spsr_cxsf 에다가 cpsr 값을 넣는다,동작 마다 spsr 을 분리해 놓는다.
```

arm 은 동작모드가 kernel 모드(supervisor 모드)와 user 모드가 있다.

이렇게 나눈이유:kernel 과 user 모드를 왔다리갔다리 하기위해서는 각각 spsr 값이 달라야 한다. 각 register 마다 spsr 이 다르다 회로상에서도.

user:cpsr 이외는 모두 spsr 을 쓴다.

```
        cps #17  //10001(fast intrrupt request)
        mov lr, r0
        mov r8,  #0x0000
        mov r9,  #0x0000
        mov r10, #0x0000
        mov r11, #0x0000
        mov r12, #0x0000
        mrs r1, cpsr
        msr spsr_cxsf, r1 //s:23~16 ,x=15~8 (spsr 구조에서),7~0=c
```

supervisor 는 모든 기능을 해야 한다. 모든 비트를 사용 하겠다. 인터럽트를 실행하면서 센서신호가 들어오면 연산을 하거나 다른 인터럽트가 들어올수 있기 때문에

모든 비트를 사용 하는 이유다.

```
cps #18 //10010(intrrupt request)
        mov lr, r0
        mrs r1,cpsr
```

msr spsr_cxsf, r1// 이것도 마찬가지로 인터럽트이기때문에 모든 비트를 사용한다.

cps #23 10111:abort(내부 인터럽트)

mov lr, r0

mrs r1,cpsr

msr spsr_cxsf, r1

cps #23 //10111(abort)

mov lr, r0

mrs r1,cpsr

msr spsr_cxsf, r1

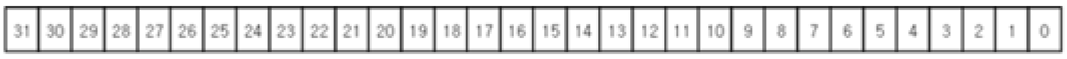
; Switch to Undefined Instruction Mode (M = 11011)// 정의되지않는 명령어를 처리할 때 사용

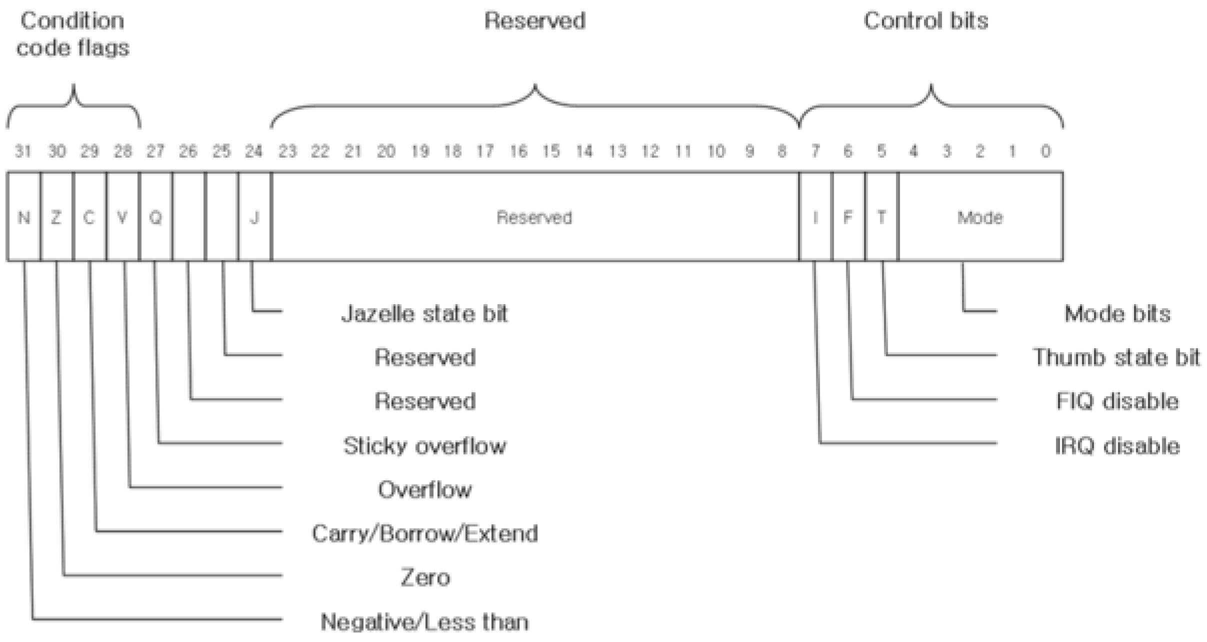
cps #31 /11111

mov lr, r0

mrs r1,cpsr

msr spsr_cxsf, r1 //동작은 이하동문





<spsr 구조>

구조를 보면 마지막 비트에 의해 FIQ 나 여러가지 동작을 정의한다.

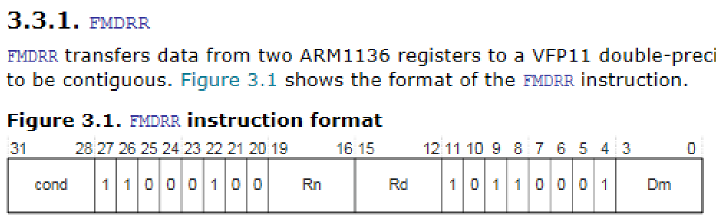
```
fmdrr d0,       r1,     r1 //d0:double-precision
        fmdrr d1,       r1,     r1
        fmdrr d2,       r1,     r1
        fmdrr d3,       r1,     r1
        fmdrr d4,       r1,     r1
        fmdrr d5,       r1,     r1
        fmdrr d6,       r1,     r1
        fmdrr d7,       r1,     r1
        fmdrr d8,       r1,     r1
        fmdrr d9,       r1,     r1
        fmdrr d10,      r1,     r1
        fmdrr d11,      r1,     r1
        fmdrr d12,      r1,     r1
        fmdrr d13,      r1,     r1
        fmdrr d14,      r1,     r1
        fmdrr d15,      r1,     r1
```

:d 의 범용 레지스터들을 0 으로 초기화 시켜준다.

```
      bl    next1
next1
        bl    next2(jump=bl)
next2
        bl    next3
next3
        bl    next4
next4
        bx    r0
```

부동 소수점은 1CLOCK 에 끝나지 않기에 한 부동 소수점이 끝날때 까지 delay 시키는 코드

<_coreInitStackPointer_ c 언어는 stack 에 무조건 올려야 된다.이 부분이 stack 에 올리는 에셈블리어>



3.3.1. FMDRR

FMDRR transfers data from two ARM1136 registers to a VFP11 double-precision register. The ARM1136 registers do not have to be contiguous. Figure 3.1 shows the format of the FMDRR instruction.

Figure 3.1. FMDRR instruction format

31–28	27	26	25	24	23	22	21	20	19–16	15–12	11	10	9	8	7	6	5	4	3–0
cond	1	1	0	0	0	1	0	0	Rn	Rd	1	0	1	1	0	0	0	1	Dm

```
cps   #17 //10001(fast intrrupt request)
        ldr   sp,       fiqSp(스택도 각각 다르게 설정을 해야 한다)
        cps   #18
        ldr   sp,       irqSp
        cps   #19
        ldr   sp,       svcSp
        cps   #23
        ldr   sp,       abortSp
        cps   #27
        ldr   sp,       undefSp
        cps   #31
        ldr   sp,       userSp
        bx    lr
```

userSp .word 0x08000000+0x00001000(user 만 stack 공간을 4096 할당 나머지는 256 으로 할당)

svcSp .word 0x08000000+0x00001000+0x00000100

fiqSp .word 0x08000000+0x00001000+0x00000100+0x00000100

irqSp .word 0x08000000+0x00001000+0x00000100+0x00000100+0x00000100

abortSp .word 0x08000000+0x00001000+0x00000100+0x00000100+0x00000100+0x00000100

undefSp .word 0x08000000+0x00001000+0x00000100+0x00000100+0x00000100+0x00000100+0x00000100

:이 부분이 어셈에서의 #define 이 되는 부분이다.

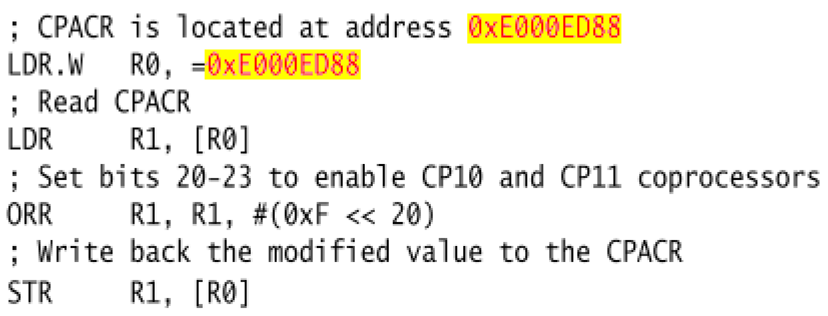
```
_memInit_
        ldr   r12, MINITGCR     ;Load MINITGCR register address
        mov   r4, #0xA
        str   r4, [r12]


    ldr   r5, MSTCGSTAT
        ldr   r4, [r5]
        tst   r4, #0x100 //AND 연산
        beq   mloop
```

R4 의 0X100 에 값이 들어있다면 좆건이 만족하지 않으니 LOOP 를 빠져나오고 값이 없다면 0 이므로 bwq 가 동작을 해서 loop 를 돈다.

이것이 어셈블리어의 for 문이다.

coreDisableEventBusExport

```
    mrc   p15, #0x00, r0,        c9, c12, #0x00//
    bic   r0,  r0,    #0x10
    mcr   p15, #0x00, r0,        c9, c12, #0x00
    bx    lr
```



```
; CPACR is located at address 0xE000ED88
LDR.W   R0, =0xE000ED88
; Read CPACR
LDR     R1, [R0]
; Set bits 20-23 to enable CP10 and CP11 coprocessors
ORR     R1, R1, #(0xF << 20)
; Write back the modified value to the CPACR
STR     R1, [R0]
```

<CPACR 어셈블리 코드>

```
if ((esmREG->SR1[2]) != 0U)
    {
      esmGroup3Notification(esmREG,esmREG->SR1[2]);
    }
```

fuse 기능:과전류가 흐르면 자동으로 회로를 끊어주겠다.

systemInit=PLL

systemREG1->CSDISSET = 0x00000002U | 0x00000040U;

PLL:1 번과 6 번을 사용하겠다는 의미이다.

/*SAFETYMCUSW 28 D MR:NA <APPROVED> "Hardware status bit read check" */

while((systemREG1->CSDIS & 0x42U) != 0x42U) //PLL 1 번과 2 번을 끄는 코드.

{

/* Wait */

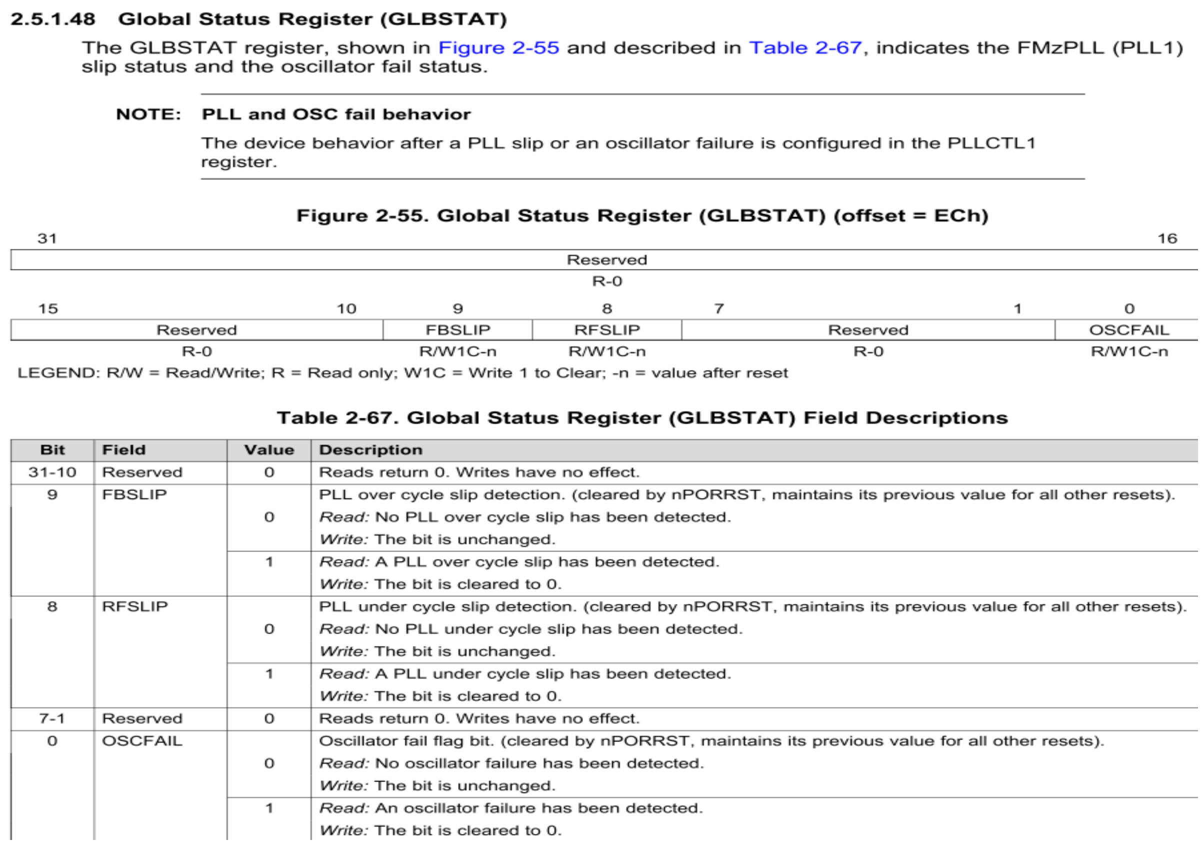
```
    }
```

SLIP:갑자기 노이즈같은 것이 확튀는것(진동)

systemREG1->GBLSTAT = 0x301U;

PLL 이 동작을 시작할 때 노이즈에 대해 보호해준는 코드이다.

GBLSTAT 의 8 번과 9 번이 CYCLE OVER,CYCLE UNDER 부분의 NOISE 에 대해 보호해준다.



2.5.1.48 Global Status Register (GLBSTAT)

The GLBSTAT register, shown in Figure 2-55 and described in Table 2-67, indicates the FMzPLL (PLL1) slip status and the oscillator fail status.

> **NOTE: PLL and OSC fail behavior**
>
> The device behavior after a PLL slip or an oscillator failure is configured in the PLLCTL1 register.

Figure 2-55. Global Status Register (GLBSTAT) (offset = ECh)

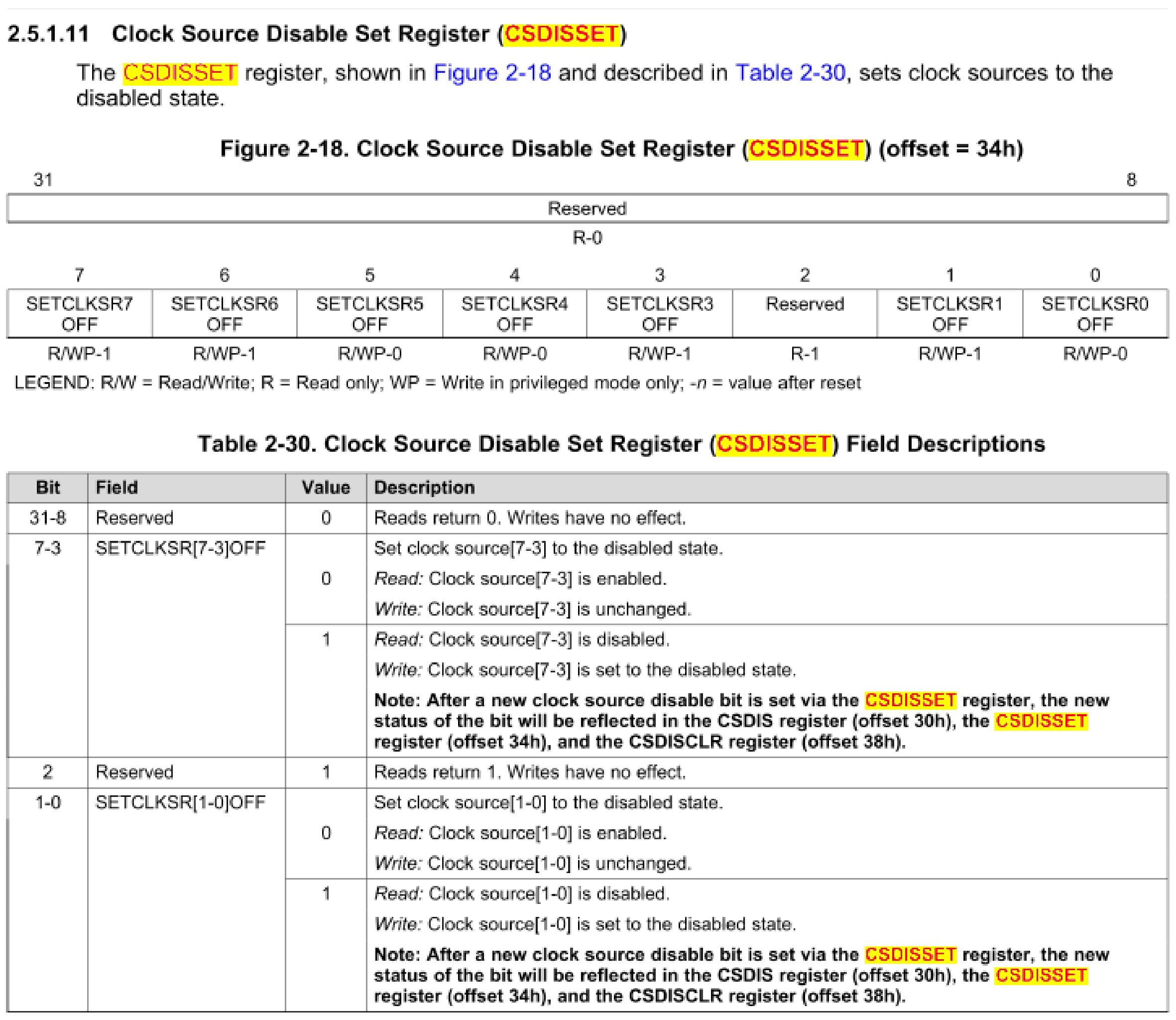
31–16
Reserved
R-0

15–10	9	8	7–1	0
Reserved	FBSLIP	RFSLIP	Reserved	OSCFAIL
R-0	R/W1C-n	R/W1C-n	R-0	R/W1C-n

LEGEND: R/W = Read/Write; R = Read only; W1C = Write 1 to Clear; -n = value after reset

Table 2-67. Global Status Register (GLBSTAT) Field Descriptions

Bit	Field	Value	Description
31-10	Reserved	0	Reads return 0. Writes have no effect.
9	FBSLIP		PLL over cycle slip detection. (cleared by nPORRST, maintains its previous value for all other resets).
		0	*Read:* No PLL over cycle slip has been detected. *Write:* The bit is unchanged.
		1	*Read:* A PLL over cycle slip has been detected. *Write:* The bit is cleared to 0.
8	RFSLIP		PLL under cycle slip detection. (cleared by nPORRST, maintains its previous value for all other resets).
		0	*Read:* No PLL under cycle slip has been detected. *Write:* The bit is unchanged.
		1	*Read:* A PLL under cycle slip has been detected. *Write:* The bit is cleared to 0.
7-1	Reserved	0	Reads return 0. Writes have no effect.
0	OSCFAIL		Oscillator fail flag bit. (cleared by nPORRST, maintains its previous value for all other resets).
		0	*Read:* No oscillator failure has been detected. *Write:* The bit is unchanged.
		1	*Read:* An oscillator failure has been detected. *Write:* The bit is cleared to 0.



2.5.1.11 Clock Source Disable Set Register (CSDISSET)

The CSDISSET register, shown in Figure 2-18 and described in Table 2-30, sets clock sources to the disabled state.

Figure 2-18. Clock Source Disable Set Register (CSDISSET) (offset = 34h)

31–8
Reserved
R-0

7	6	5	4	3	2	1	0
SETCLKSR7 OFF	SETCLKSR6 OFF	SETCLKSR5 OFF	SETCLKSR4 OFF	SETCLKSR3 OFF	Reserved	SETCLKSR1 OFF	SETCLKSR0 OFF
R/WP-1	R/WP-1	R/WP-0	R/WP-0	R/WP-1	R-1	R/WP-1	R/WP-0

LEGEND: R/W = Read/Write; R = Read only; WP = Write in privileged mode only; -*n* = value after reset

Table 2-30. Clock Source Disable Set Register (CSDISSET) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reads return 0. Writes have no effect.
7-3	SETCLKSR[7-3]OFF		Set clock source[7-3] to the disabled state.
		0	*Read:* Clock source[7-3] is enabled. *Write:* Clock source[7-3] is unchanged.
		1	*Read:* Clock source[7-3] is disabled. *Write:* Clock source[7-3] is set to the disabled state. **Note: After a new clock source disable bit is set via the CSDISSET register, the new status of the bit will be reflected in the CSDIS register (offset 30h), the CSDISSET register (offset 34h), and the CSDISCLR register (offset 38h).**
2	Reserved	1	Reads return 1. Writes have no effect.
1-0	SETCLKSR[1-0]OFF		Set clock source[1-0] to the disabled state.
		0	*Read:* Clock source[1-0] is enabled. *Write:* Clock source[1-0] is unchanged.
		1	*Read:* Clock source[1-0] is disabled. *Write:* Clock source[1-0] is set to the disabled state. **Note: After a new clock source disable bit is set via the CSDISSET register, the new status of the bit will be reflected in the CSDIS register (offset 30h), the CSDISSET register (offset 34h), and the CSDISCLR register (offset 38h).**

위에 systemREG1->CSDISSET = 0x00000002U | 0x00000040U; 이분에서 두 숫자를 OR 하면 0X42 가 나오고 SR1 과 SR6 번을 1 으로 설정시킨다.

```
systemREG1->PLLCTL1 =  (uint32)0x00000000U
                    |  (uint32)0x20000000U
                    |  (uint32)((uint32)0x1FU << 24U)
                    |  (uint32)0x00000000U
                    |  (uint32)((uint32)(8U - 1U)<< 16U)
                    |  (uint32)(0x9500U);
```

f_pllclk=f_post_odclk/32

(8U - 1U)<< 16U)

70000

0111 0000 0000 0000 0000 :18~16BIT 사용->f_INTCLK=f_oscim/8

0x9500U

1001 0110 0000 0000 14,13,10,9BIT 사용->f_VCOCLK=f_INTCLK*150

9*16+6=150

systemREG1->PLLCTL2 = (uint32)((uint32)255U << 22U) // 255=FF00000->1111 1111 0000 0000 0000 0000<<22->0011 1111 1100 0000 0000 0000 0000=3FC0000

```
                        |  (uint32)((uint32)7U << 12U) //7000 ->14~12->NO ADDER(0)
                        |  (uint32)((uint32)(1U - 1U) << 9U) //9 번비트 0 으로 고정
                        |  (uint32)61U; // 1111111->111101=5,4,3,2,1 BIT->62/2048=NV
```

systemREG2->PLLCTL3 = (uint32)((uint32)(1U - 1U) << 29U) //29 번 BIT 0 으로 고정

```
                        | (uint32)((uint32)0x1FU << 24U)
                        | (uint32)((uint32)(8U - 1U)<< 16U)
                        | (uint32)(0x9500U);
```

1F000000=0001 1111 0000 0000 0000 0000 0000 0000 ->

28~24BIT 사용

70000=0111 0000 0000 0000 0000 :18~16bit 사용

9500= 1001 0101 0000 0000 15,12,10,8bit 사용->   단톡 확인

```
   systemREG1->CSDIS = 0x00000000U
                     | 0x00000000U
                     | 0x00000008U
                     | 0x00000080U
                     | 0x00000000U
                     | 0x00000000U
```

| 0x00000000U

| 0x00000004U;

0X0000008CU-> 0X000000 1000 1100->7번 2,3번= OFF , 6,5,4,1,0=ON

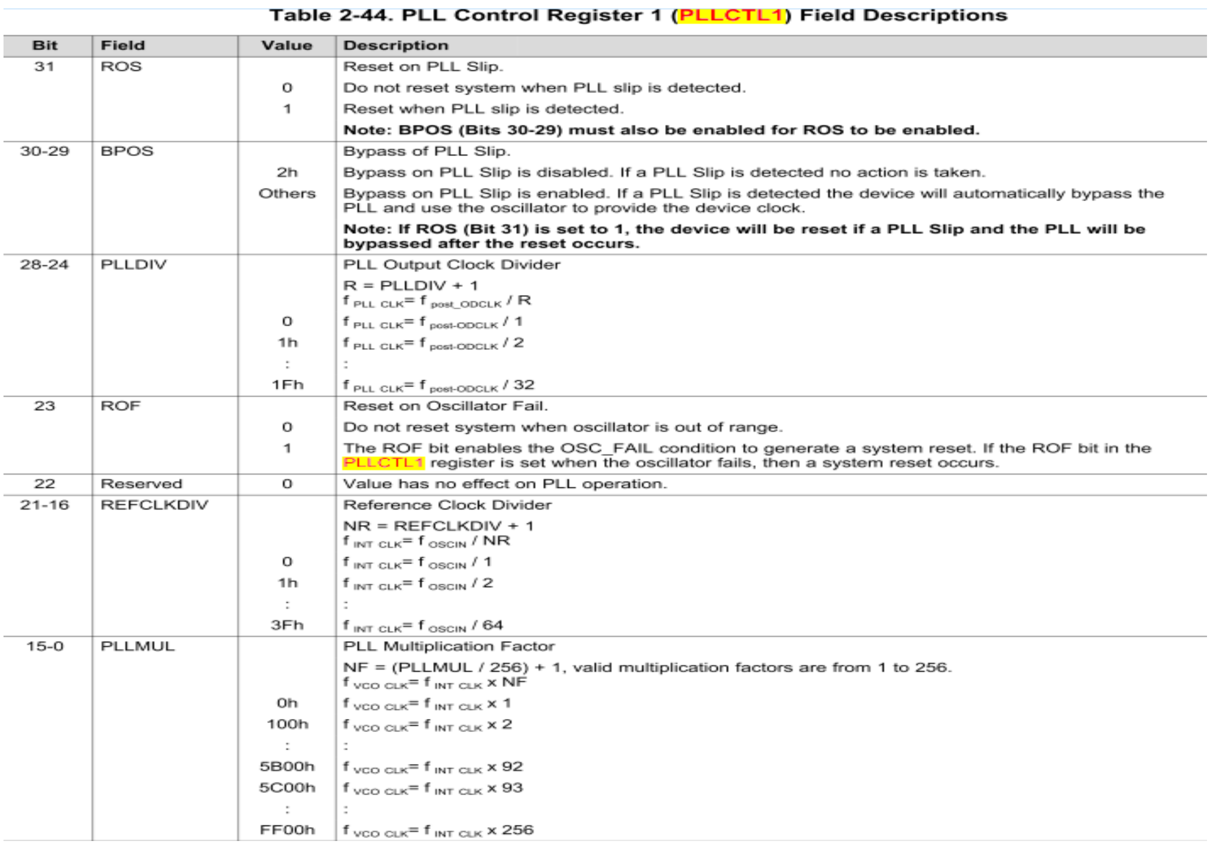


Table 2-44. PLL Control Register 1 (PLLCTL1) Field Descriptions

Bit	Field	Value	Description
31	ROS		Reset on PLL Slip.
		0	Do not reset system when PLL slip is detected.
		1	Reset when PLL slip is detected.
			Note: BPOS (Bits 30-29) must also be enabled for ROS to be enabled.
30-29	BPOS		Bypass of PLL Slip.
		2h	Bypass on PLL Slip is disabled. If a PLL Slip is detected no action is taken.
		Others	Bypass on PLL Slip is enabled. If a PLL Slip is detected the device will automatically bypass the PLL and use the oscillator to provide the device clock.
			Note: If ROS (Bit 31) is set to 1, the device will be reset if a PLL Slip and the PLL will be bypassed after the reset occurs.
28-24	PLLDIV		PLL Output Clock Divider R = PLLDIV + 1 $f_{PLL\ CLK} = f_{post\_ODCLK} / R$
		0	$f_{PLL\ CLK} = f_{post\text{-}ODCLK} / 1$
		1h	$f_{PLL\ CLK} = f_{post\text{-}ODCLK} / 2$
		:	:
		1Fh	$f_{PLL\ CLK} = f_{post\text{-}ODCLK} / 32$
23	ROF		Reset on Oscillator Fail.
		0	Do not reset system when oscillator is out of range.
		1	The ROF bit enables the OSC_FAIL condition to generate a system reset. If the ROF bit in the PLLCTL1 register is set when the oscillator fails, then a system reset occurs.
22	Reserved	0	Value has no effect on PLL operation.
21-16	REFCLKDIV		Reference Clock Divider NR = REFCLKDIV + 1 $f_{INT\ CLK} = f_{OSCIN} / NR$
		0	$f_{INT\ CLK} = f_{OSCIN} / 1$
		1h	$f_{INT\ CLK} = f_{OSCIN} / 2$
		:	:
		3Fh	$f_{INT\ CLK} = f_{OSCIN} / 64$
15-0	PLLMUL		PLL Multiplication Factor NF = (PLLMUL / 256) + 1, valid multiplication factors are from 1 to 256. $f_{VCO\ CLK} = f_{INT\ CLK} \times NF$
		0h	$f_{VCO\ CLK} = f_{INT\ CLK} \times 1$
		100h	$f_{VCO\ CLK} = f_{INT\ CLK} \times 2$
		:	:
		5B00h	$f_{VCO\ CLK} = f_{INT\ CLK} \times 92$
		5C00h	$f_{VCO\ CLK} = f_{INT\ CLK} \times 93$
		:	:
		FF00h	$f_{VCO\ CLK} = f_{INT\ CLK} \times 256$

<PLLCTL1 구조>

0X1F=0X0001 1111 0000 0000 0000 0000 0000 0000

위에 구조를 보면 28~24 번지가 모두 "1"로 셋팅이 됐으며 식의 계산을 보면 1FH=f_pllclk=f_post_odclk/32 아래 값들고 각 비트번호의 맞게 계산을 하면 된다.

(8U - 1U)<< 16U)

70000

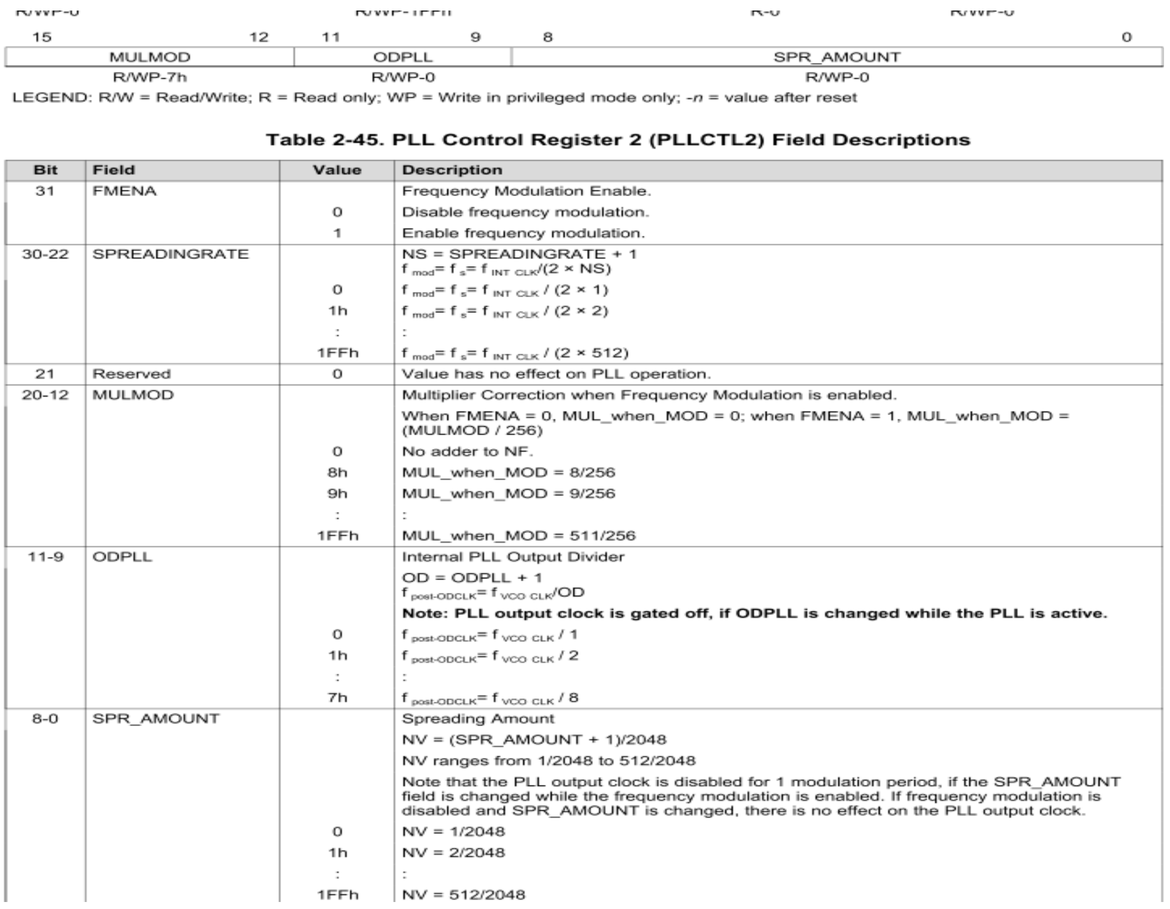
0111 0000 0000 0000 0000 :18~16BIT 사용->f_INTCLK=f_oscim/8

0x9500U

1001 0110 0000 0000 14,13,10,9BIT 사용->f_VCOCLK=f_INTCLK*150

9*16+6=150

<PLLCTL2 구조>



15	12	11	9	8	0
MULMOD		ODPLL		SPR_AMOUNT	
R/WP-7h		R/WP-0		R/WP-0	

LEGEND: R/W = Read/Write; R = Read only; WP = Write in privileged mode only; -*n* = value after reset

Table 2-45. PLL Control Register 2 (PLLCTL2) Field Descriptions

Bit	Field	Value	Description
31	FMENA		Frequency Modulation Enable.
		0	Disable frequency modulation.
		1	Enable frequency modulation.
30-22	SPREADINGRATE		NS = SPREADINGRATE + 1 $f_{mod} = f_s = f_{INT\ CLK}/(2 \times NS)$
		0	$f_{mod} = f_s = f_{INT\ CLK} / (2 \times 1)$
		1h	$f_{mod} = f_s = f_{INT\ CLK} / (2 \times 2)$
		:	:
		1FFh	$f_{mod} = f_s = f_{INT\ CLK} / (2 \times 512)$
21	Reserved	0	Value has no effect on PLL operation.
20-12	MULMOD		Multiplier Correction when Frequency Modulation is enabled.
			When FMENA = 0, MUL_when_MOD = 0; when FMENA = 1, MUL_when_MOD = (MULMOD / 256)
		0	No adder to NF.
		8h	MUL_when_MOD = 8/256
		9h	MUL_when_MOD = 9/256
		:	:
		1FFh	MUL_when_MOD = 511/256
11-9	ODPLL		Internal PLL Output Divider
			OD = ODPLL + 1 $f_{post\text{-}ODCLK} = f_{VCO\ CLK}/OD$
			Note: PLL output clock is gated off, if ODPLL is changed while the PLL is active.
		0	$f_{post\text{-}ODCLK} = f_{VCO\ CLK} / 1$
		1h	$f_{post\text{-}ODCLK} = f_{VCO\ CLK} / 2$
		:	:
		7h	$f_{post\text{-}ODCLK} = f_{VCO\ CLK} / 8$
8-0	SPR_AMOUNT		Spreading Amount
			NV = (SPR_AMOUNT + 1)/2048
			NV ranges from 1/2048 to 512/2048
			Note that the PLL output clock is disabled for 1 modulation period, if the SPR_AMOUNT field is changed while the frequency modulation is enabled. If frequency modulation is disabled and SPR_AMOUNT is changed, there is no effect on the PLL output clock.
		0	NV = 1/2048
		1h	NV = 2/2048
		:	:
		1FFh	NV = 512/2048

systemREG1->PLLCTL2 = (uint32)((uint32)255U << 22U) // 255=FF00000->1111 1111 0000 0000 0000 0000<<22->0011 1111 1100 0000 0000 0000 0000=3FC0000

```
                    |  (uint32)((uint32)7U << 12U) //7000 ->14~12->NO ADDER(0)
                    |  (uint32)((uint32)(1U - 1U) << 9U) //9 번비트 0 으로 고정
                    |  (uint32)61U; // 1111111->111101=5,4,3,2,1 BIT->62/2048=NV
```

F_POSR-ODCLK=F_VCOCLK/1

F_MOD= F_S=F_INTCLK/512

```
 systemREG2->PLLCTL3 = (uint32)((uint32)(1U - 1U) << 29U) //29 번 BIT 0 으로 고정
                    | (uint32)((uint32)0x1FU << 24U)
                    | (uint32)((uint32)(8U - 1U)<< 16U)
                    | (uint32)(0x9500U);
```

1F000000=0001 1111 0000 0000 0000 0000 0000 0000

28~24BIT 사용

70000=0111 0000 0000 0000 0000 :18~16bit 사용

9500= 1001 0101 0000 0000 15,12,10,8bit 사용

```
   systemREG1->CSDIS = 0x00000000U
                    | 0x00000000U
                    | 0x00000008U
                    | 0x00000080U
                    | 0x00000000U
                    | 0x00000000U
                    | 0x00000000U
                    | 0x00000004U;
```

0X0000008CU-> 0X000000 1000 1100->7 번 2,3 번= OFF , 6,5,4,1,0=ON

F_POST-ODCLK2-F_OUTPIT-CLK2/1

F_PLL2CLK=F_POST-ODCLK2/32

F_INTCLK2=F_PSCIN/8

F_VCOCLK2=F_INTCLK2*150

```
void periphInit(void)
{
```

```
/* USER CODE BEGIN (9) */
/* USER CODE END */

    /** - Disable Peripherals before peripheral powerup*/
    systemREG1->CLKCNTL &= 0xFFFFFEFFU;

    /** - Release peripherals from reset and enable clocks to all
peripherals */
    /** - Power-up all peripherals */
    pcrREG1->PSPWRDWNCLR0 = 0xFFFFFFFFU;
//PSPWRDWNSET1,PSPWRDWNCLR1 두개다 모두 초기화
    pcrREG1->PSPWRDWNCLR1 = 0xFFFFFFFFU;  //초기화
    pcrREG1->PSPWRDWNCLR2 = 0xFFFFFFFFU;  //초기화
    pcrREG1->PSPWRDWNCLR3 = 0xFFFFFFFFU;  //초기화

    pcrREG2->PSPWRDWNCLR0 = 0xFFFFFFFFU;  //초기화
    pcrREG2->PSPWRDWNCLR1 = 0xFFFFFFFFU;  //초기화
    pcrREG2->PSPWRDWNCLR2 = 0xFFFFFFFFU;  //초기화
    pcrREG2->PSPWRDWNCLR3 = 0xFFFFFFFFU;  //초기화

    pcrREG3->PSPWRDWNCLR0 = 0xFFFFFFFFU;  //초기화
    pcrREG3->PSPWRDWNCLR1 = 0xFFFFFFFFU;  //초기화
    pcrREG3->PSPWRDWNCLR2 = 0xFFFFFFFFU;  //초기화
    pcrREG3->PSPWRDWNCLR3 = 0xFFFFFFFFU;  //초기화

    /** - Enable Peripherals */
```

```
void muxInit(void){


/* USER CODE BEGIN (1) */
/* USER CODE END */


            /* Enable Pin Muxing */
            pinMuxReg->KICKER0 = 0x83E70B13U;
            pinMuxReg->KICKER1 = 0x95A4F1E0U;
->둘다 값이 들어와야 MUX 를 쓸수가 있다는 코드
    systemREG1->CLKCNTL |= 0x00000100U; /이 부분은 모든 비트가 RESET 이 되었다고 알려주는것
pinMuxReg->PINMUX[0] = PINMUX_BALL_N19_AD1EVT | PINMUX_BALL_D4_EMIF_ADDR_00 |
PINMUX_BALL_D5_EMIF_ADDR_01 | PINMUX_BALL_C4_EMIF_ADDR_06;


            pinMuxReg->PINMUX[1] = PINMUX_BALL_C5_EMIF_ADDR_07 |
PINMUX_BALL_C6_EMIF_ADDR_08 | PINMUX_BALL_C7_EMIF_ADDR_09 |
PINMUX_BALL_C8_EMIF_ADDR_10;


            pinMuxReg->PINMUX[2] = PINMUX_BALL_C9_EMIF_ADDR_11 |
PINMUX_BALL_C10_EMIF_ADDR_12 | PINMUX_BALL_C11_EMIF_ADDR_13 |
PINMUX_BALL_C12_EMIF_ADDR_14;


            pinMuxReg->PINMUX[3] = PINMUX_BALL_C13_EMIF_ADDR_15 |
PINMUX_BALL_D14_EMIF_ADDR_16 | PINMUX_BALL_C14_EMIF_ADDR_17 |
PINMUX_BALL_D15_EMIF_ADDR_18;


            pinMuxReg->PINMUX[4] = PINMUX_BALL_C15_EMIF_ADDR_19 |
PINMUX_BALL_C16_EMIF_ADDR_20 | PINMUX_BALL_C17_EMIF_ADDR_21;
            pinMuxReg->PINMUX[5] = 0U;
            pinMuxReg->PINMUX[6] = 0U;
            pinMuxReg->PINMUX[7] = 0U;
            pinMuxReg->PINMUX[8] = PINMUX_BALL_D16_EMIF_BA_1;


            pinMuxReg->PINMUX[9] = PINMUX_BALL_R4_EMIF_nCAS | PINMUX_BALL_N17_EMIF_nCS_0 |
PINMUX_BALL_L17_EMIF_nCS_2;


            pinMuxReg->PINMUX[10] = PINMUX_BALL_K17_EMIF_nCS_3 |
PINMUX_BALL_M17_EMIF_nCS_4 | PINMUX_BALL_R3_EMIF_nRAS | PINMUX_BALL_P3_EMIF_nWAIT;


            pinMuxReg->PINMUX[11] = PINMUX_BALL_D17_EMIF_nWE | PINMUX_BALL_E9_ETMDATA_08
| PINMUX_BALL_E8_ETMDATA_09 | PINMUX_BALL_E7_ETMDATA_10;
```

```
            pinMuxReg->PINMUX[12] = PINMUX_BALL_E6_ETMDATA_11 |
PINMUX_BALL_E13_ETMDATA_12 | PINMUX_BALL_E12_ETMDATA_13 |
PINMUX_BALL_E11_ETMDATA_14;

            pinMuxReg->PINMUX[13] = PINMUX_BALL_E10_ETMDATA_15 |
PINMUX_BALL_K15_ETMDATA_16 | PINMUX_BALL_L15_ETMDATA_17 |
PINMUX_BALL_M15_ETMDATA_18;

            pinMuxReg->PINMUX[14] = PINMUX_BALL_N15_ETMDATA_19 |
PINMUX_BALL_E5_ETMDATA_20 | PINMUX_BALL_F5_ETMDATA_21 | PINMUX_BALL_G5_ETMDATA_22;

            pinMuxReg->PINMUX[15] = PINMUX_BALL_K5_ETMDATA_23 |
PINMUX_BALL_L5_ETMDATA_24 | PINMUX_BALL_M5_ETMDATA_25 | PINMUX_BALL_N5_ETMDATA_26;

            pinMuxReg->PINMUX[16] = PINMUX_BALL_P5_ETMDATA_27 |
PINMUX_BALL_R5_ETMDATA_28 | PINMUX_BALL_R6_ETMDATA_29 | PINMUX_BALL_R7_ETMDATA_30;

            pinMuxReg->PINMUX[17] = PINMUX_BALL_R8_ETMDATA_31 |
PINMUX_BALL_R9_ETMTRACECLKIN | PINMUX_BALL_R10_ETMTRACECLKOUT |
PINMUX_BALL_R11_ETMTRACECTL;

            pinMuxReg->PINMUX[18] = PINMUX_BALL_B15_FRAYTX1 | PINMUX_BALL_B8_FRAYTX2 |
PINMUX_BALL_B16_FRAYTXEN1 | PINMUX_BALL_B9_FRAYTXEN2;

            pinMuxReg->PINMUX[19] = PINMUX_BALL_C1_GIOA_2 | PINMUX_BALL_E1_GIOA_3 |
PINMUX_BALL_B5_GIOA_5 | PINMUX_BALL_H3_GIOA_6;

            pinMuxReg->PINMUX[20] = PINMUX_BALL_M1_GIOA_7 | PINMUX_BALL_F2_GIOB_2 |
PINMUX_BALL_W10_GIOB_3 | PINMUX_BALL_J2_GIOB_6;

            pinMuxReg->PINMUX[21] = PINMUX_BALL_F1_GIOB_7 | PINMUX_BALL_R2_MIBSPI1NCS_0 |
PINMUX_BALL_F3_MIBSPI1NCS_1 | PINMUX_BALL_G3_MIBSPI1NCS_2;

            pinMuxReg->PINMUX[22] = PINMUX_BALL_J3_MIBSPI1NCS_3 |
PINMUX_BALL_G19_MIBSPI1NENA | PINMUX_BALL_V9_MIBSPI3CLK |
PINMUX_BALL_V10_MIBSPI3NCS_0;

            pinMuxReg->PINMUX[23] = PINMUX_BALL_V5_MIBSPI3NCS_1 |
PINMUX_BALL_B2_MIBSPI3NCS_2 | PINMUX_BALL_C3_MIBSPI3NCS_3 |
PINMUX_BALL_W9_MIBSPI3NENA;

            pinMuxReg->PINMUX[24] = PINMUX_BALL_W8_MIBSPI3SIMO |
PINMUX_BALL_V8_MIBSPI3SOMI | PINMUX_BALL_H19_MIBSPI5CLK |
PINMUX_BALL_E19_MIBSPI5NCS_0;
```

```
            pinMuxReg->PINMUX[25] = PINMUX_BALL_B6_MIBSPI5NCS_1 |
PINMUX_BALL_W6_MIBSPI5NCS_2 | PINMUX_BALL_T12_MIBSPI5NCS_3 |
PINMUX_BALL_H18_MIBSPI5NENA;


            pinMuxReg->PINMUX[26] = PINMUX_BALL_J19_MIBSPI5SIMO_0 |
PINMUX_BALL_E16_MIBSPI5SIMO_1 | PINMUX_BALL_H17_MIBSPI5SIMO_2 |
PINMUX_BALL_G17_MIBSPI5SIMO_3;


            pinMuxReg->PINMUX[27] = PINMUX_BALL_J18_MIBSPI5SOMI_0 |
PINMUX_BALL_E17_MIBSPI5SOMI_1 | PINMUX_BALL_H16_MIBSPI5SOMI_2 |
PINMUX_BALL_G16_MIBSPI5SOMI_3;


            pinMuxReg->PINMUX[28] = PINMUX_BALL_K18_N2HET1_00 | PINMUX_BALL_V2_N2HET1_01
| PINMUX_BALL_W5_N2HET1_02 | PINMUX_BALL_U1_N2HET1_03;


            pinMuxReg->PINMUX[29] = PINMUX_BALL_B12_N2HET1_04 | PINMUX_BALL_V6_N2HET1_05
| PINMUX_BALL_W3_N2HET1_06 | PINMUX_BALL_T1_N2HET1_07;


            pinMuxReg->PINMUX[30] = PINMUX_BALL_E18_N2HET1_08 | PINMUX_BALL_V7_N2HET1_09
| PINMUX_BALL_D19_N2HET1_10 | PINMUX_BALL_E3_N2HET1_11;


            pinMuxReg->PINMUX[31] = PINMUX_BALL_B4_N2HET1_12 | PINMUX_BALL_N2_N2HET1_13 |
PINMUX_BALL_N1_N2HET1_15 | PINMUX_BALL_A4_N2HET1_16;


            pinMuxReg->PINMUX[32] = PINMUX_BALL_A13_N2HET1_17 | PINMUX_BALL_J1_N2HET1_18 |
PINMUX_BALL_B13_N2HET1_19 | PINMUX_BALL_P2_N2HET1_20;


            pinMuxReg->PINMUX[33] = PINMUX_BALL_H4_N2HET1_21 | PINMUX_BALL_B3_N2HET1_22 |
PINMUX_BALL_J4_N2HET1_23 | PINMUX_BALL_P1_N2HET1_24;


            pinMuxReg->PINMUX[34] = PINMUX_BALL_A14_N2HET1_26 |
PINMUX_BALL_K19_N2HET1_28 | PINMUX_BALL_B11_N2HET1_30 | PINMUX_BALL_D8_N2HET2_01;


            pinMuxReg->PINMUX[35] = PINMUX_BALL_D7_N2HET2_02 | PINMUX_BALL_D3_N2HET2_12 |
PINMUX_BALL_D2_N2HET2_13 | PINMUX_BALL_D1_N2HET2_14;


            pinMuxReg->PINMUX[36] = PINMUX_BALL_P4_N2HET2_19 | PINMUX_BALL_T5_N2HET2_20 |
PINMUX_BALL_T4_MII_RXCLK | PINMUX_BALL_U7_MII_TX_CLK;


            pinMuxReg->PINMUX[37] = PINMUX_BALL_E2_N2HET2_03 | PINMUX_BALL_N3_N2HET2_07;
```

110h-1A4h->0~37 번까지 의 배열

```
pinMuxReg->PINMUX[80] = (SIGNAL_AD2EVT_T10 | 0x02020200U);

            pinMuxReg->PINMUX[81] = 0x02020202U;

            pinMuxReg->PINMUX[82] = 0x02020202U;

            pinMuxReg->PINMUX[83] = (SIGNAL_GIOA_0_A5 | 0x00020202U);

            pinMuxReg->PINMUX[84] = SIGNAL_GIOA_1_C2 | SIGNAL_GIOA_2_C1 | SIGNAL_GIOA_3_E1 |
SIGNAL_GIOA_4_A6;

            pinMuxReg->PINMUX[85] = SIGNAL_GIOA_5_B5 | SIGNAL_GIOA_6_H3 | SIGNAL_GIOA_7_M1 |
SIGNAL_GIOB_0_M2;

            pinMuxReg->PINMUX[86] = SIGNAL_GIOB_1_K2 | SIGNAL_GIOB_2_F2 | SIGNAL_GIOB_3_W10
| SIGNAL_GIOB_4_G1;

            pinMuxReg->PINMUX[87] = SIGNAL_GIOB_5_G2 | SIGNAL_GIOB_6_J2 | SIGNAL_GIOB_7_F1 |
SIGNAL_MDIO_F4;

            pinMuxReg->PINMUX[88] = (SIGNAL_MIBSPI1NCS_4_U10 | SIGNAL_MIBSPI1NCS_5_U9 |
0x00020000U) ;

            pinMuxReg->PINMUX[89] = SIGNAL_MII_COL_W4 | SIGNAL_MII_CRS_V4;

            pinMuxReg->PINMUX[90] = SIGNAL_MII_RX_DV_U6 | SIGNAL_MII_RX_ER_U5 |
SIGNAL_MII_RXCLK_T4 | SIGNAL_MII_RXD_0_U4;

            pinMuxReg->PINMUX[91] = SIGNAL_MII_RXD_1_T3 | SIGNAL_MII_RXD_2_U3 |
SIGNAL_MII_RXD_3_V3 | SIGNAL_MII_TX_CLK_U7;

            pinMuxReg->PINMUX[92] = SIGNAL_N2HET1_17_A13 | SIGNAL_N2HET1_19_B13 |
SIGNAL_N2HET1_21_H4 | SIGNAL_N2HET1_23_J4;

            pinMuxReg->PINMUX[93] = SIGNAL_N2HET1_25_M3 | SIGNAL_N2HET1_27_A9 |
SIGNAL_N2HET1_29_A3 | SIGNAL_N2HET1_31_J17;

            pinMuxReg->PINMUX[94] = SIGNAL_N2HET2_00_D6 | SIGNAL_N2HET2_01_D8 |
SIGNAL_N2HET2_02_D7 | SIGNAL_N2HET2_03_E2;

            pinMuxReg->PINMUX[95] = SIGNAL_N2HET2_04_D13 | SIGNAL_N2HET2_05_D12 |
SIGNAL_N2HET2_06_D11 | SIGNAL_N2HET2_07_N3;
```

pinMuxReg->PINMUX[96] = SIGNAL_N2HET2_08_K16 | SIGNAL_N2HET2_09_L16 | SIGNAL_N2HET2_10_M16 | SIGNAL_N2HET2_11_N16;

pinMuxReg->PINMUX[97] = SIGNAL_N2HET2_12_D3 | SIGNAL_N2HET2_13_D2 | SIGNAL_N2HET2_14_D1 | SIGNAL_N2HET2_15_K4;

pinMuxReg->PINMUX[98] = SIGNAL_N2HET2_16_L4 | SIGNAL_N2HET2_18_N4 | SIGNAL_N2HET2_20_T5 | SIGNAL_N2HET2_22_T7;

pinMuxReg->PINMUX[99] = SIGNAL_nTZ1_1_N19 | SIGNAL_nTZ1_2_F1 | SIGNAL_nTZ1_3_J3;

80~99:19 개->250H~29CH

PINMUX_BALL_D4_EMIF_ADDR_00-> PINMUX_BALL_D4_SHIFT))->#define PINMUX_BALL_D4_SHIFT 8U :8 비트를 세팅한다.

PWM 갯수=하드웨어 갯수

ETPWM1:CHIP 을 제어한다.

```
void setupFlash(void)
{

/* USER CODE BEGIN (6) */
/* USER CODE END */

    /** - Setup flash read mode, address wait states and data wait states */
    flashWREG->FRDCNTL =  0x00000000U
                       | (uint32)((uint32)3U << 8U)
                       |  3U;

    /** - Setup flash access wait states for bank 7 */
    FSM_WR_ENA_HL     = 0x5U; // 이 부분이 flash 를 사용 하기 위해 5h 를 넣어야 한다.
    EEPROM_CONFIG_HL = 0x00000002U
                     | (uint32)((uint32)9U << 16U) ; // EEPROM 이 9 개 있다.
```