TI DSP, MCU, Xilinx Zynq FPGA
기반의 프로그래밍 전문가 과정

<펌웨어 프로그래밍>

2018.05.08 - 49일차

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 안상재
sangjae2015@naver.com

1. _c_int00() => systemInit() => trimLP0()

```
void trimLPO(void)
{
     if(LPO_TRIM_VALUE != 0xFFFFU)
     {
         systemREG1->LPOMONCTL   = (uint32)((uint32)1U << 24U)
                                 | LPO_TRIM_VALUE;
       // The bias circuit inside the low-power oscillator (LPO) is enabled.
       // LPOMONCTL의 하위 16bit에 LPO_TRIM_VALUE 값 대입
     }
     else
     {
         systemREG1->LPOMONCTL    =  (uint32)((uint32)1U << 24U)
                                   | (uint32)((uint32)16U << 8U)
                                   | 16U;

     }
}
```

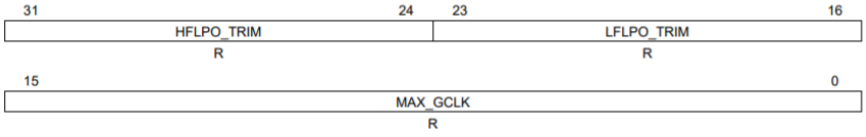
- 0xF0008 01B4 주소의 레지스터

#define LPO_TRIM_VALUE (((*(volatile uint32 *)0xF00801B4U) & 0xFFFF0000U)>>16U)

7.5.2.3 LPO Trim and Max HCLK

The HF LPO trim solution, LF LPO trim solution and maximum GCLK1 frequency can be read from TI OTP location F008 01B4h as shown in Figure 7-5 and described in Table 7-7.

Figure 7-5. TI OTP Bank 0 LPO Trim and Max HCLK Information



LEGEND: R = Read only

Table 7-7. TI OTP Bank 0 LPO Trim and Max HCLK Information Field Descriptions

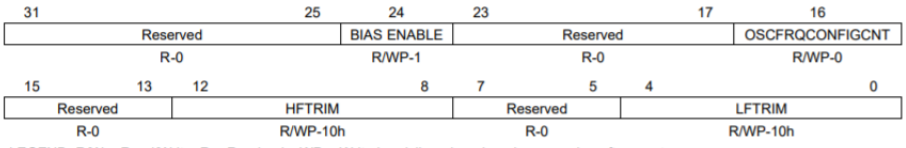
Bit	Field	Description
31-24	HFLPO_TRIM	HF LPO Trim Solution
23-16	LFLPO_TRIM	LF LPO Trim Solution
15-0	MAX_GCLK	Maximum GCLK1 Speed

- LPOMONCTL

2.5.1.30 LPO/Clock Monitor Control Register (LPOMONCTL)

The LPOMONCTL register, shown in Figure 2-37 and described in Table 2-49, controls the Low Frequency (Clock Source 4) and High Frequency (Clock Source 5) Low Power Oscillator's trim values.

Figure 2-37. LPO/Clock Monitor Control Register (LPOMONCTL) (offset = 088h)



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

Table 2-49. LPO/Clock Monitor Control Register (LPOMONCTL) Field Descriptions

Bit	Field	Value	Description
31-25	Reserved	0	Reads return 0. Writes have no effect.
24	BIAS ENABLE		Bias enable.
		0	The bias circuit inside the low-power oscillator (LPO) is disabled.
		1	The bias circuit inside the low-power oscillator (LPO) is enabled.
23-17	Reserved	0	Reads return 0. Writes have no effect.
16	OSCFRQCONFIGCNT		Configures the counter based on OSC frequency.
		0	*Read:* OSC freq is ≤ 20MHz.
			Write: A write of 0 has no effect.
		1	*Read:* OSC freq is > 20MHz and ≤ 80MHz.
			Write: A write of 1 has no effect.
15-13	Reserved	0	Reads return 0. Writes have no effect.

2. _c_int00() => systemInit() => mapClocks()

```
void mapClocks(void)
{
    uint32 SYS_CSVSTAT, SYS_CSDIS;

/* USER CODE BEGIN (11) */
/* USER CODE END */

    /** @b Initialize @b Clock @b Tree: */
    /** - Setup system clock divider for HCLK */
    systemREG2->HCLKCNTL = 1U;  // HCLK is equal to GCLK1 divide by 2.

    /** - Disable / Enable clock domain */
    systemREG1->CDDIS = (uint32)((uint32)0U << 4U ) // The VCLKA[1] domain is enabled.
                      | (uint32)((uint32)1U << 5U ) // The VCLKA[2] domain is disabled.
                      | (uint32)((uint32)0U << 8U ) // The VCLK3 domain is enabled.
                      | (uint32)((uint32)0U << 9U ) // 9번째 bit는 reserved
                      | (uint32)((uint32)0U << 10U) // 10번째 bit는 reserved
                      | (uint32)((uint32)0U << 11U); // The VCLKA4 domain is enabled

    /** - Wait for until clocks are locked */
    SYS_CSVSTAT = systemREG1->CSVSTAT;
    SYS_CSDIS = systemREG1->CSDIS;
    while ((SYS_CSVSTAT & ((SYS_CSDIS ^ 0xFFU) & 0xFFU)) != ((SYS_CSDIS ^ 0xFFU) & 0xFFU))
// CSVSTAT 과 CSDIS 이 반전 관계까 되지 않으면
    {
        SYS_CSVSTAT = systemREG1->CSVSTAT;
        SYS_CSDIS = systemREG1->CSDIS;
    } /* Wait */

    /** - Setup GCLK, HCLK and VCLK clock source for normal operation, power down mode and
after wakeup */
    systemREG1->GHVSRC = (uint32)((uint32)SYS_PLL1 << 24U) // Clock source1(PLL1) is the
source for GCLK1, HCLK, VCLK on wakeup
                       | (uint32)((uint32)SYS_PLL1 << 16U) // Clock source1(PLL1) is the
source for HCLK, VCLK, VCLK2 on wakeup.
                       | (uint32)((uint32)SYS_PLL1 << 0U); // Clock source0(Oscillator) is
the source for GCLK1, HCLK, VCLK, VCLK2.

    /** - Setup RTICLK1 and RTICLK2 clocks */
    systemREG1->RCLKSRC = (uint32)((uint32)1U << 24U)        // no effect
                        | (uint32)((uint32)SYS_VCLK << 16U) // no effect
                        | (uint32)((uint32)1U << 8U)         // Clock source1 is the source
for RTICLK1.
                        | (uint32)((uint32)SYS_VCLK << 0U); // VCLK is the source for RTI-
CLK1.

    /** - Setup asynchronous peripheral clock sources for AVCLK1 and AVCLK2 */
    systemREG1->VCLKASRC = (uint32)((uint32)SYS_VCLK << 8U) // VCLK is the source for periph-
eral asynchronous clock2.
                         | (uint32)((uint32)SYS_VCLK << 0U); // VCLK is the source for pe-
ripheral asynchronous clock1.


    /** - Setup synchronous peripheral clock dividers for VCLK1, VCLK2, VCLK3 */
    systemREG1->CLKCNTL  = (systemREG1->CLKCNTL & 0xF0FFFFFFU)
                         | (uint32)((uint32)1U << 24U);    // The VCLK2 speed is HCLK divided
by 2.
    systemREG1->CLKCNTL  = (systemREG1->CLKCNTL & 0xFFF0FFFFU)
                         | (uint32)((uint32)1U << 16U);    // The VCLK speed is HCLK divided
by 2.

    systemREG2->CLK2CNTRL = (systemREG2->CLK2CNTRL & 0xFFFFFFF0U)
```

```
                        | (uint32)((uint32)1U << 0U);   // The ratio is HCLK divide by 2.

    systemREG2->VCLKACON1 =  (uint32)((uint32)(1U - 1U) << 24U)   // The ratio is VCLKA4 di-
vided by 1.
                           | (uint32)((uint32)0U << 20U)    // Enable the prescaled VCLKA4
clock on VCLKA4_DIVR.
                           | (uint32)((uint32)SYS_VCLK << 16U)  // VCLK or a divided VCLK is
the source for peripheral asynchronous clock4.
                           | (uint32)((uint32)(1U - 1U) << 8U) // no effect
                           | (uint32)((uint32)0U << 4U)  // no effect
                           | (uint32)((uint32)SYS_VCLK << 0U); // no effect

    // Reset when PLL slip is detected.
    /* 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. */
    // f PLL CLK= f post-ODCLK / 1
    systemREG1->PLLCTL1 = (systemREG1->PLLCTL1 & 0xE0FFFFFFU) | (uint32)((uint32)(1U - 1U) <<
24U);

    // fpost_ODCLK2 = foutput_CLK2 / 8
    // fPLL2 CLK = fpost_ODCLK2 / 1
    systemREG2->PLLCTL3 = (systemREG2->PLLCTL3 & 0xE0FFFFFFU) | (uint32)((uint32)(1U - 1U) <<
24U);

    systemREG1->PLLCTL2 |= 0x00000000U;
}
```

3. _c_int00() => systemInit() 마지막 부분

```
/** - set ECLK pins functional mode */
    systemREG1->SYSPC1 = 0U;    // ECLK is in GIO mode.

    /** - set ECLK pins default output value */
    systemREG1->SYSPC4 = 0U;    // The ECLK pin is driven to logic low (0)

    /** - set ECLK pins output direction */
    systemREG1->SYSPC2 = 1U;    // The ECLK pin is an output.

    /** - set ECLK pins open drain enable */
    systemREG1->SYSPC7 = 0U;    // The ECLK pin is configured in push/pull (normal GIO) mode

    /** - set ECLK pins pullup/pulldown enable */
    systemREG1->SYSPC8 = 0U;    // ECLK pull enable is inactive

    /** - set ECLK pins pullup/pulldown select */
    systemREG1->SYSPC9 = 1U;  // ECLK pull up is selected, when pull up/pull down logic is
enabled.

    /** - Setup ECLK */
    systemREG1->ECPCNTL = (uint32)((uint32)0U << 24U)   // VCLK is selected as the ECP clock
source
                        | (uint32)((uint32)0U << 23U)   /* ECLK output is disabled in suspend
mode. ECLK output will be shut off and will not be seen on the I/O pin of the device. */
                        | (uint32)((uint32)(8U - 1U) & 0xFFFFU);// ECLK = VCLK / (ECPDIV + 1)
```