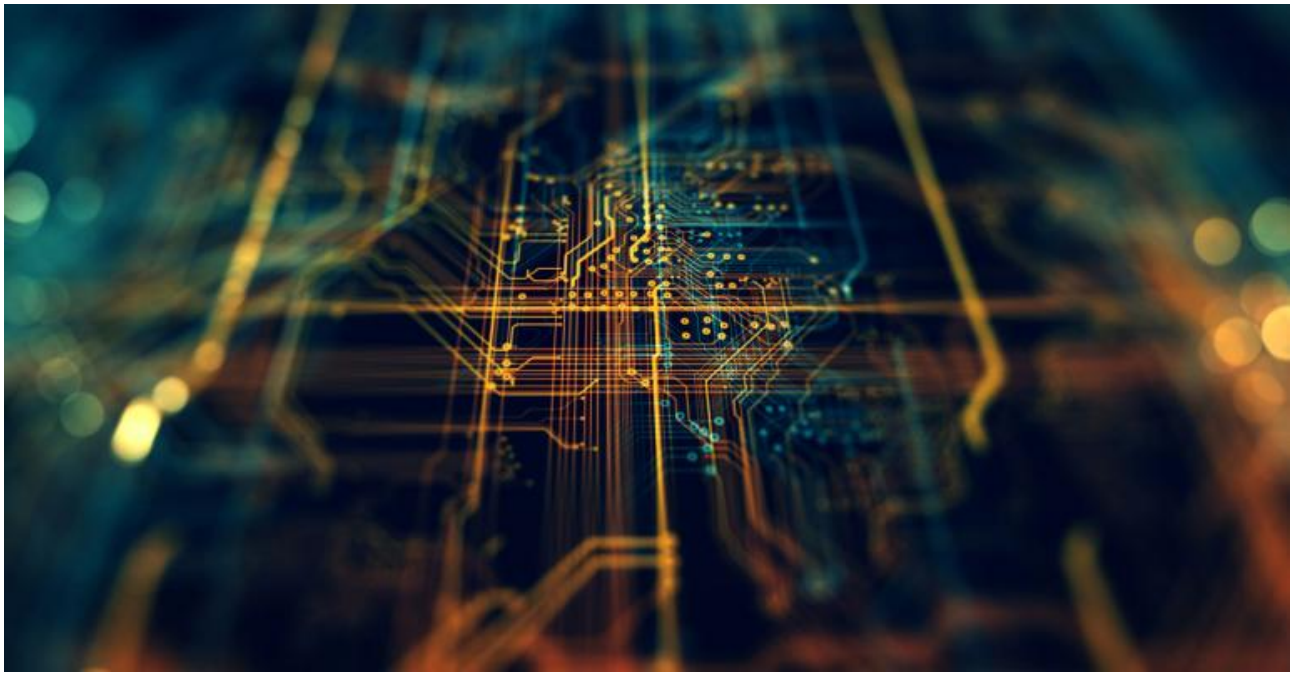
TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정



2018.05.02
46 일차
강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 – 신민철
akrn33@naver.com

```
if(LPO_TRIM_VALUE != 0xFFFFU)
{

    systemREG1->LPOMONCTL  = (uint32)((uint32)1U << 24U)
                           | LPO_TRIM_VALUE;
}
else
{

    systemREG1->LPOMONCTL   =  (uint32)((uint32)1U << 24U)
                            | (uint32)((uint32)16U << 8U)
                            | 16U;
}
```

12 번부터 8 번비트까지 사용하는데 맨앞의 비트가 1 이 되어야 하니까 else 로 넘어가게 된다.

```
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;

    /** - Disable / Enable clock domain */
    systemREG1->CDDIS = (uint32)((uint32)0U << 4U ) /* AVCLK1 , 1 - OFF, 0 - ON */
                      | (uint32)((uint32)1U << 5U ) /* AVCLK2 , 1 - OFF, 0 - ON */
                      | (uint32)((uint32)0U << 8U ) /* VCLK3 , 1 - OFF, 0 - ON */
                      | (uint32)((uint32)0U << 9U ) /* VCLK4 , 1 - OFF, 0 - ON */
                      | (uint32)((uint32)0U << 10U) /* AVCLK3 , 1 - OFF, 0 - ON */
                      | (uint32)((uint32)0U << 11U); /* AVCLK4 , 1 - OFF, 0 - ON */


    /* Always check the CSDIS register to make sure the clock source is turned on and check
     * the CSVSTAT register to make sure the clock source is valid. Then write to GHVSRC to switch the clock.
     */
    /** - 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))
    {
        SYS_CSVSTAT = systemREG1->CSVSTAT;
        SYS_CSDIS = systemREG1->CSDIS;
    } /* Wait */

/* USER CODE BEGIN (12) */
/* USER CODE END */

    /** - Map device clock domains to desired sources and configure top-level dividers */
    /** - All clock domains are working off the default clock sources until now */
    /** - The below assignments can be easily modified using the HALCoGen GUI */

    /** - Setup GCLK, HCLK and VCLK clock source for normal operation, power down mode and after wakeup */
    systemREG1->GHVSRC = (uint32)((uint32)SYS_PLL1 << 24U)
                       | (uint32)((uint32)SYS_PLL1 << 16U)
                       | (uint32)((uint32)SYS_PLL1 << 0U);

    /** - Setup RTICLK1 and RTICLK2 clocks */
    systemREG1->RCLKSRC = (uint32)((uint32)1U << 24U)        /* RTI2 divider (Not applicable for lock-step device)  */
                        | (uint32)((uint32)SYS_VCLK << 16U) /* RTI2 clock source (Not applicable for lock-step device) */
                        | (uint32)((uint32)1U << 8U)         /* RTI1 divider */
                        | (uint32)((uint32)SYS_VCLK << 0U); /* RTI1 clock source */

    /** - Setup asynchronous peripheral clock sources for AVCLK1 and AVCLK2 */
    systemREG1->VCLKASRC = (uint32)((uint32)SYS_VCLK << 8U)
                         | (uint32)((uint32)SYS_VCLK << 0U);

    /** - Setup synchronous peripheral clock dividers for VCLK1, VCLK2, VCLK3 */
    systemREG1->CLKCNTL  = (systemREG1->CLKCNTL & 0xF0FFFFFFU)
                         | (uint32)((uint32)1U << 24U);
    systemREG1->CLKCNTL  = (systemREG1->CLKCNTL & 0xFFF0FFFFU)
                         | (uint32)((uint32)1U << 16U);

    systemREG2->CLK2CNTRL = (systemREG2->CLK2CNTRL & 0xFFFFFFF0U)
                          | (uint32)((uint32)1U << 0U);
```

```
    systemREG2->VCLKACON1 =  (uint32)((uint32)(1U - 1U) << 24U)
                           | (uint32)((uint32)0U << 20U)
                           | (uint32)((uint32)SYS_VCLK << 16U)
                           | (uint32)((uint32)(1U - 1U) << 8U)
                           | (uint32)((uint32)0U << 4U)
                           | (uint32)((uint32)SYS_VCLK << 0U);

/* USER CODE BEGIN (13) */
/* USER CODE END */

    /* Now the PLLs are locked and the PLL outputs can be sped up */
    /* The R-divider was programmed to be 0xF. Now this divider is changed to programmed value */
    systemREG1->PLLCTL1 = (systemREG1->PLLCTL1 & 0xE0FFFFFFU) | (uint32)((uint32)(1U - 1U) << 24U);
    /*SAFETYMCUSW 134 S MR:12.2 <APPROVED> " Clear and write to the volatile register " */
    systemREG2->PLLCTL3 = (systemREG2->PLLCTL3 & 0xE0FFFFFFU) | (uint32)((uint32)(1U - 1U) << 24U);

    /* Enable/Disable Frequency modulation */
    systemREG1->PLLCTL2 |= 0x00000000U;

/* USER CODE BEGIN (14) */
/* USER CODE END */

}
```

실제 디바이스 모듈과 거기에 들어가는 클록들을 매칭시켜주는것이 mapClocks 이다.

Coprocessor 명령어 세트

명령어	동작	어셈블리
CDP	Data operation	CDP{cond} ⟨cp_num⟩, ⟨op1⟩, ⟨CRd⟩, ⟨CRn⟩, ⟨CRm⟩{, ⟨op2⟩}
MRC	코프로세스 레지스터의 정보를 ARM레지스터로 읽어 온다.	MRC{cond} ⟨cp_num⟩, ⟨op1⟩, ⟨Rd⟩, ⟨CRn⟩, ⟨CRm⟩{, ⟨op2⟩}
MCR	ARM레지스터의 정보를 코프로세스 레지스터로 읽어 온다.	MCR{cond} ⟨cp_num⟩, ⟨op1⟩, ⟨Rd⟩, ⟨CRn⟩, ⟨CRm⟩{, ⟨op2⟩}
MRRC	코프로세스의 정보를 한 쌍의 ARM 레지스터들에 전송한다.	MRRC{cond} ⟨cp_num⟩, ⟨op1⟩, ⟨Rd⟩, ⟨Rn⟩, ⟨CRm⟩
MCRR	한 쌍의 ARM 레지스터들의 정보를 코프로세스에 전송한다.	MCRR{cond} ⟨cp_num⟩, ⟨op1⟩, ⟨Rd⟩, ⟨Rn⟩, ⟨CRm⟩
LDC	메모리의 내용을 코프로세스 레지스터에 저장한다.	LDC{cond} ⟨cp_num⟩, ⟨CRd⟩, ⟨a_mode5⟩
STC	코프로세스 레지스터의 내용을 메모리에 저장한다.	STC{cond} ⟨cp_num⟩, ⟨CRd⟩, ⟨a_mode5⟩