TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-05-08 (49회차)

강사 - Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 - 정유경

ucong@naver.com

Cortex R5 부트코드 분석 (part2)

```
151     /* Initialize System - Clock, Flash settings with Efuse self check */
152     systemInit();
153
```

들어간다

```
389     /** - Configure the LPO such that HF LPO is as close to 10MHz as possible */
390     trimLPO();
391
```

LPO(저전력 Oscillator)는 Datasheet를 찾아보면 다음과 같다.

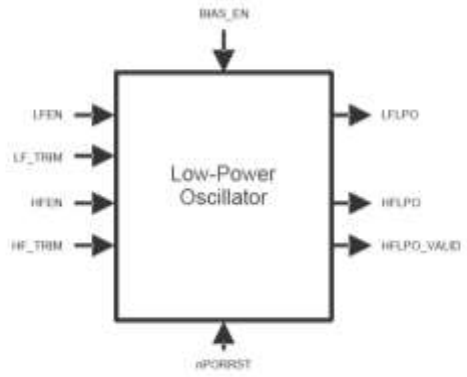


Figure 6-4. LPO Block Diagram

LPO - HF oscillator	Untrimmed frequency	5.5	9	19.5	MHz
	Trimmed frequency	8.0	9.6	11.0	MHz
	Start-up time from STANDBY (LPO BIAS_EN high for at least 900 µs)			10	µs
	Cold start-up time			900	µs

LPO를 초기화 하는 부분이다.

```
146 void trimLPO(void)
147 {
148
```

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

LPO_TRIM_VALUE가 뭔지 알아보면,

F008로 검색한다. 목차부분에서 위치를 추측한다 -> 347p

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

31 – 24	23 – 16
HFLPO_TRIM	LFLPO_TRIM
R	R

15 – 0
MAX_GCLK
R

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

우리가 찾는 주소인 F008 01B4h는 TI OTP location이고 여기서 HF LPO trim solution, LF LPO trim solution, maximum GCLK1 frequency를 읽을 수 있다고 나온다.

*. OTP(One Time Programable) : Flash memory 에는 OTP 영역이 존재한다. OTP 는 One Time Programmable 로 한번 기록하면 추가기록이 영구적으로 불가능하다. 따라서 Flash memory 의 특정영역을 OTP 명령으로 셋팅 하면 그 영역은 마치 PROM 화 된다. 정리하면 OTP 는 한번 프로그램을 입력하면 바꾸지 못하는 메모리로써 초기의 Setting 값을 간단히 처리하여 유지하려는 목적으로 사용되는 메모리 영역이다.

포인터로 접근한 주소 안에 있는 값과 & 하면 최상위 비트만 뽑아낼 수 있다 그걸 오른쪽으로 16비트 쉬프트하여 옮긴다. (최상위 16비트 빈자리에는 0이 채워진다) 결과적으로 LPO_TRIM_VALUE는 0x0000F008이된다. 즉, HF LPO trim, LF LPO trim을 읽어서 하위 16비트에 채워 넣은 것이다.

```
152   /** @b Initialize Lpo: */
153   /** Load TRIM values from OTP if present else load user defined values */
154   /*SAFETYMCUSW 139 S MR:13.7 <APPROVED> "Hardware status bit read check" */
155   if(LPO_TRIM_VALUE != 0xFFFFU)
156   {
157       systemREG1->LPOMONCTL  = (uint32)((uint32)1U << 24U) // 24 번째 비트를 1로 설정
158                              | LPO_TRIM_VALUE; //0x0000F008
159   }
```

if문 안으로 들어간다.

SystemREG1구조체의 LPOMONCTL멤버를 설정한다.

[24] LPO내부의 bias circuit을 활성화하고, Low-frequency oscillator trim value를 60.86으로 설정한다.

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.

15-13	Reserved	0	Reads return 0. Writes have no effect.

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

Bit	Field	Value	Description
4-0	LFTRIM		Low-frequency oscillator trim value. This four-bit value is used to center the LF oscillator's frequency. **Caution: This value should only be changed when the LF oscillator is not the source for a clock domain, otherwise a system failure could result.** The following values are the ratio: f / fo in the F021 process.
		0	20.67
		1h	25.76
		2h	30.84
		3h	35.90
		4h	40.93
		5h	45.95
		6h	50.97
		7h	55.91
		8h	60.86

Else 들어가지 않고 빠져나온다.

// 수업 중에는 바이어스 넣어주고 100퍼센트 출력으로 쓰겠다고 하신 것 같은데 제 해석과 다르네요 다시 해보겠습니다.

*. 주파수 trim : 특정한 주파수로 깎아낸다. 즉, 파형을 정형하는 회로이다

10h 100.00% Default at Reset. // 감쇠를 시키지 않고 1000퍼센트 쓰겠다는 의미

PLL은 처음에 오버슈트 있고 서서히 안정화 된다. 따라서 lock될 때까지 기다린다.

또한 clock domain을 해당하는 clock source와 맵핑 시킨다.

(lock: 주파수 고정, 안정화)

```
/** - Wait for PLLs to start up and map clock domains to desired clock sources */
mapClocks(); // 디바이스 모델과 실제 클록을 매칭시켜주는 작업을 한다
```

```
systemREG2->HCLKCNTL = 1U;
```

Table 2-77. HCLK Control Register (HCLKCNTL) Field Descriptions

Bit	Field	Value	Description
31-2	Reserved	0	Reads return 0. Writes have no effect.
1-0	HCLKR		HCLK divider value. The value of HCLKR bits determine the HCLK frequency as a ratio of GCLK1.
		0	HCLK is equal to GCLK1 divide by 1.
		1h	HCLK is equal to GCLK1 divide by 2.
		2h	HCLK is equal to GCLK1 divide by 3.
		3h	HCLK is equal to GCLK1 divide by 4.

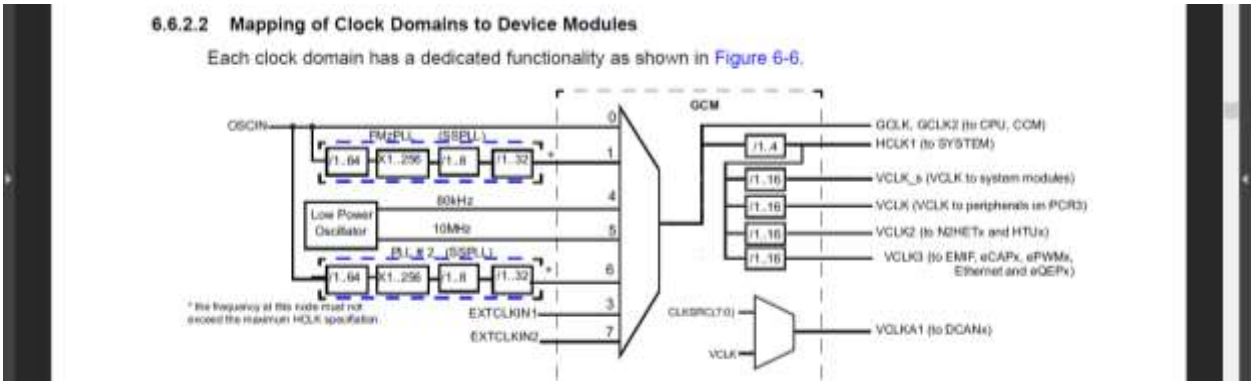
HCLK를 GCLK1을 2분주한 값으로 설정한다

(즉, 글로벌 클록을 2분주하여 사용한다. 이 값은 데이터 시트를 참고하면 Up to 300-MHz)

- Operating Conditions
  - Up to 300-MHz CPU Clock
  - Core Supply Voltage (VCC): 1.14 to 1.32 V
  - I/O Supply Voltage (VCCIO): 3.0 to 3.6 V
  - Four UART (SCI) Interfaces, Two With Local Interconnect Network (LIN 2.1) Interface Support
- Two Next Generation High-End Timer (N2HET)

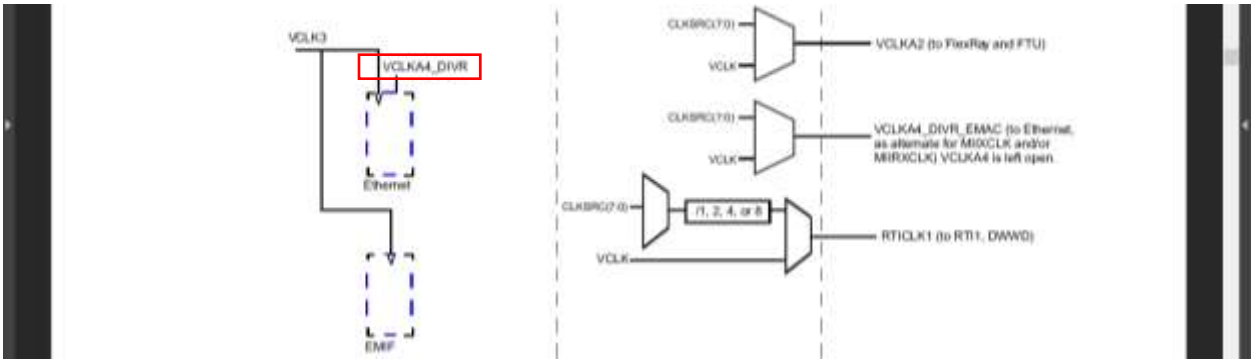
```
/** - Disable / Enable clock domain */
systemREG1->CDDIS = (uint32)((uint32)0U << 4U ) /* AVCLK1 , 1 - OFF, 0 - ON */
                  | (uint32)((uint32)1U << 5U ) /* AVCLK2 , 1 - OFF, 0 - ON */
                  | (uint32)((uint32)0U << 8U ) /* VCLK3 , 1 - OFF, 0 - ON */
                  | (uint32)((uint32)0U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

CDDIS의 각 비트에 1을 설정하면 비활성화, 0을 설정하면 활성화하는 것이다.



VCLKA1은 DCAN(CAN통신)

VCLK3는 EMIF, eCAPx, ePWMx, 이더넷, eQEPx에 사용된다.



VCLKA4는 분주하여 이더넷에 사용된다.

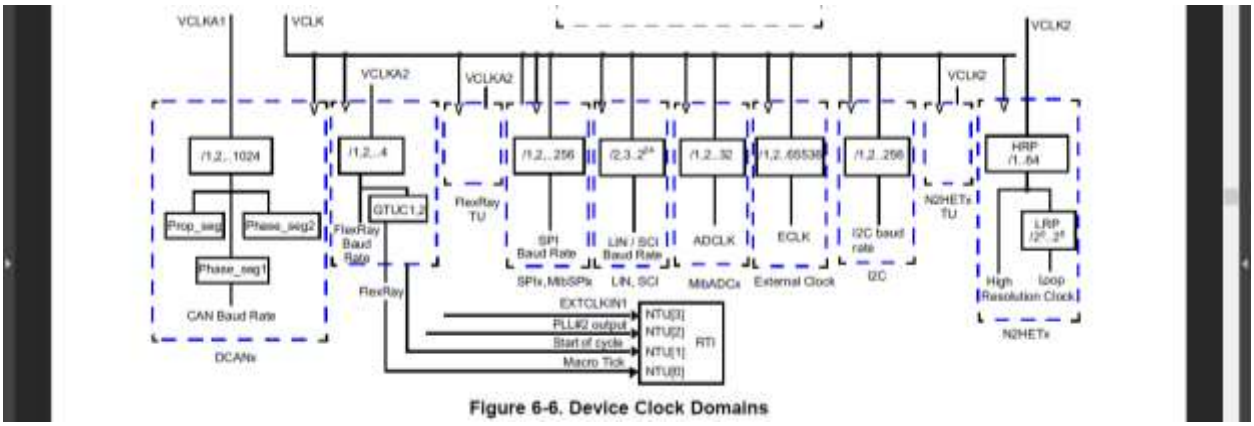


Figure 6-6. Device Clock Domains

*. asynchronous 즉, 'A' 가 붙어있는 clock 도메인은 비동기 클럭이다. (신호가 언제 들어올지 모르므로 신호가 들어올 때 동기화 시키라는 의미)

*. VCLKA2 (to FlexRay and FTU) : HTU를 쓰면 굉장히 작은 단위(pico sec)까지 감지 가능 엔진에 사용한다.

<참고> 데이터시트를 참고하면 각 clock domain들의 frequency값을 알 수 있다.

f_{OSC}	OSC - oscillator clock frequency using an external crystal		5	20	MHz
f_{GCLK1}	GCLK - R5F CPU clock frequency			300	MHz
f_{GCLK2}	GCLK - R5F CPU clock frequency			300	MHz
f_{HCLK}	HCLK - System clock frequency			150	MHz
f_{VCLK}	VCLK - Primary peripheral clock frequency			110	MHz
f_{VCLK2}	VCLK2 - Secondary peripheral clock frequency			110	MHz
f_{VCLK3}	VCLK3 - Secondary peripheral clock frequency			150	MHz
f_{VCLKA1}	VCLKA1 - Primary asynchronous peripheral clock frequency			110	MHz
f_{VCLKA2}	VCLKA2 - Secondary asynchronous peripheral clock frequency			110	MHz
f_{VCLKA4}	VCLKA4 - Secondary asynchronous peripheral clock frequency			110	MHz

*. 주기가 너무 짧다! pwm신호로 150이 가면 모터가 돌아가지 않는다

우리가 사용하는 모터는 20ms로 사용한다. 즉 50hz 로 만들어 주려면 PLL을 건드려야 한다.

(ex. cortex R5: LED 잘 돌아가는데 모터는 안 돌아가게 된다. 이때, PLL을 수정해 준다)

*. AVR은 pwm에서 나오는 신호가 자동으로 20ms로 주기가 맞추어져 있다.

따라서 PLL건드릴 필요없다.

```
/** - 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 */ // 안정화 될때까지 계속 루프를 돈다
```

CSVSTAT, CDDIS값을 계속 읽어서 clock이 안정화 될 때까지 기다린다.

CSVSTAT(0x0054) & (CDDIS(0x0030) ^ 0xFFU하여 하위 8비트를 반전시키고) & 0xFFU 하위 8비트값을 읽어서

결국 CSVSTAT에 clock source가 활성화되었다는 표시를 하는 비트가 제대로 1이 설정되었는지를 보고

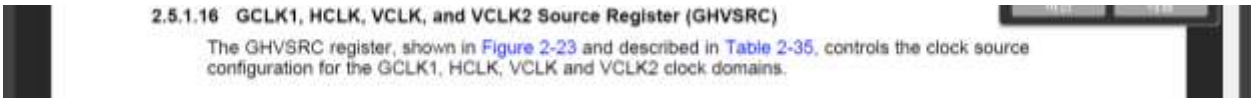
앞에서 활성화, 비활성화 설정해준 CDDIS부분과 같은지를 체크하는 부분이다.

같아질 때까지 즉, 안정화 될 때까지 루프를 빠져나오지 않는다.

*. CSVSTAT: Clock Source가 안정화 되어있는지 보는 비트, VCLK 활성화 되어 안정화 되면 여기서 1 아니면 0 이 비트설정은 하드웨어가 해준다

디바이스 클록과 VCLK clock source를 설정한다.

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



2.5.1.16 GCLK1, HCLK, VCLK, and VCLK2 Source Register (GHVSRC)

The GHVSRC register, shown in Figure 2-23 and described in Table 2-35, controls the clock source configuration for the GCLK1, HCLK, VCLK and VCLK2 clock domains.

SYS_PLL1이 1이므로 24,16,0번 비트에 1을 설정한다. 다음과 같은 의미이다.

Clock source1 is the source for GCLK1, HCLK, VCLK on wakeup.

Clock source1 is the source for HCLK, VCLK, VCLK2 on wakeup.

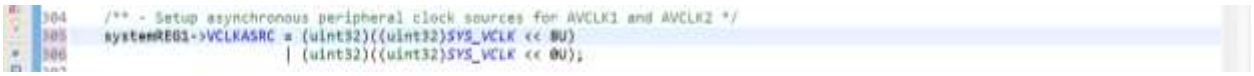
(142 를 보면 Clock Sources1번이 PLL1임을 알 수 있다)

Clock source1 is the source for GCLK1, HCLK, VCLK, VCLK2.

```
79 enum systemClockSource
80 {
81     SYS_OSC            = 0x0U,  /**< Alias for oscillator clock Source              */
82     SYS_PLL1           = 0x1U,  /**< Alias for PLL1 clock Source                    */
83     SYS_EXTERNAL1      = 0x3U,  /**< Alias for external clock Source                */
84     SYS_LPO_LOW        = 0x4U,  /**< Alias for low power oscillator low clock Source  */
85     SYS_LPO_HIGH       = 0x5U,  /**< Alias for low power oscillator high clock Source */
86     SYS_PLL2           = 0x6U,  /**< Alias for PLL2 clock Source                    */
87     SYS_EXTERNAL2      = 0x7U,  /**< Alias for external 2 clock Source              */
88     SYS_VCLK           = 0x9U,  /**< Alias for synchronous VCLK1 clock Source       */
89     SYS_PLL2_ODCLK_8   = 0xEU,  /**< Alias for PLL2_post_ODCLK/8                    */
90     SYS_PLL2_ODCLK_16  = 0xFU   /**< Alias for PLL2_post_ODCLK/8                    */
91 };
```

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

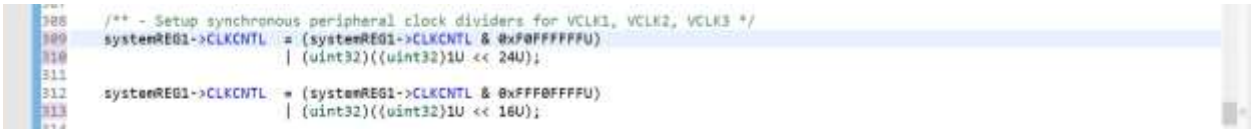
RCLKSRC는 RTI Clock Source Register 이다. 즉, 리얼타임 인터럽트인데 인터럽트를 사용하지 않으므로 컴파일러가 쓸데없는 코드를 만들었다는 것을 알 수 있다.



AVCLK에 대한 비동기 clock source를 지정해주는 부분이다.

VCLKASRC 11,8 // VCLK is the source for peripheral asynchronous clock2.

3,0 비트에 1을 설정 // VCLK is the source for peripheral asynchronous clock1.

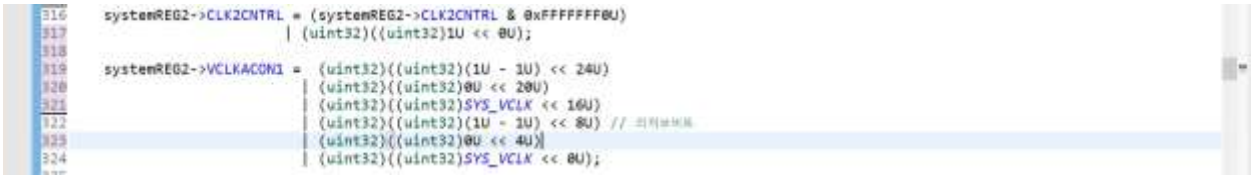


VCLK에 대한 동기 clock divider를 지정해주는 부분이다.

The VCLK2 speed is HCLK divided by 2

The VCLK speed is HCLK divided by 2

(즉, fHCLK = 150 MHz를 2분주하니까 fVCLK = 75 MHz, fVCLK2 = 75 MHz 가 된다)



CLK2CNTRL레지스터는 (Clock 2 Control Register) VCLK3의 주파수를 결정한다. The ratio is HCLK divide by 2 (2분주를 사용한다)

VCLKACON1레지스터는 (Peripheral Asynchronous Clock Configuration 1 Register) VCLKA4 즉 앞의 데이터 시트에서 확인했듯이 이더넷 관련 설정하는 레지스터이다.

하나씩 살펴보면,

24비트에 0이므로, The ratio is VCLKA4 divided by 1. 분주없다.

20비트에 0이므로 Enable the prescaled VCLKA4 clock on VCLKA4_DIVR.

1001 <<16 // 19,16 VCLK 이나, 분주된 VCLK가 비동기 clock4의 clock source이다.

```
/* 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);
```

PLL이 안정화되었으므로, PLL출력을 증폭하는 부분인 것 같다.

PLL Control Register 1 (PLLCTL1)를 31~28에 0110, 24번비트에 0을 셋팅한다.

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	$f_{PLL\ CLK} = f_{post\_ODCLK} / 1$

앞에서(setupPLL()) PLL클록을 32분주로 지정했었다. 여기서는 1분주 즉 그대로 사용하는 것으로 설정한다.

마찬가지로 PLLCTL3는

fPLL2 CLK = fpost_ODCLK2 / 1 분주없음.

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

마지막으로 frequency modulation을 비활성화 시킨다.

```
/** - set ECLK pins functional mode */
systemREG1->SYSPC1 = 0U;
/** - set ECLK pins default output value */
systemREG1->SYSPC4 = 0U;
/** - set ECLK pins output direction */
systemREG1->SYSPC2 = 1U;
/** - set ECLK pins open drain enable */
systemREG1->SYSPC7 = 0U;
/** - set ECLK pins pullup/pulldown enable */
systemREG1->SYSPC8 = 0U;
/** - set ECLK pins pullup/pulldown select */
systemREG1->SYSPC9 = 1U;
```

ECLK의 핀들을 설정하는 부분이다.

SYS Pin Control Register 1 (SYSPC1) : ECLK(external clock prescaler) 가 GIO mode로 동작한다.

ECLK is in functional mode as a clock output

ECLK pins default output value가 logic 0이다

ECLK pins output direction이 출력모드이다.

ECLK pins는 푸시-풀 모드(push/pull (normal GIO) mode)로 동작한다.

ECLK pins pullup/pulldown enable은 active이다.

ECLK pins pullup/pulldown select는 풀업으로 설정되었다

```
/** - Setup ECLK */
/*ECP Control Register : configures the ECLK pin in functional mode*/
systemREG1->ECPCNTL = (uint32)((uint32)0U << 24U)
                    | (uint32)((uint32)0U << 23U)
                    | (uint32)((uint32)(8U - 1U) & 0xFFFFU);
```

[24] VCLK is selected as the ECP clock source.

[23] 중지모드에서 ECLK 아웃풋이 꺼진다. 즉 디바이스의 아웃풋에서 감지되지 않는다는 의미이다

7 +1 = 8분주 (ECP divider value. determine the external clock (ECP clock) frequency)

*. systemInit()이 완료되면 다음과 같은 값으로 클럭 설정이 완료되는 것을 알 수 있다.

Over Recommended Operating Conditions

PARAMETER		TEST CONDITIONS	MIN	TYP[1]	MAX	UNIT
I_{CC}	V_{CC} digital supply and PLL current (operating mode)	f_{GCLK} = 300 MHz, f_{HCLK} = 150 MHz, f_{VCLK} = 75 MHz, f_{VCLK2} = 75 MHz, f_{VCLK3} = 150 MHz		510	990 [2]	mA

```
/* Enable IRQ offset via Vic controller */
_coreEnableIrqVicOffset_();

/* Initialize VIM table */
vimInit();  //벡터 인터럽트 매니저를 설정함

/* USER CODE BEGIN (12) */
/* USER CODE END */
/* Configure system response to error conditions signaled to the ESM group1 */
/* This function can be configured from the ESM tab of HALCoGen */
esmInit(); // ecc로 관련되어 인해 들어가지 않았음
```

```
/* Enable CPU Event Export */
/* This allows the CPU to signal any single-bit or double-bit errors detected
 * by its ECC logic for accesses to program flash or data RAM.
 */
_coreEnableEventBusExport_();
```


mpuInit();
/* USER CODE BEGIN (23) */
/* USER CODE END */
cacheEnable();
/* USER CODE BEGIN (24) */
/* USER CODE END */
/* USER CODE BEGIN (25) */
/* USER CODE END */
/* initialize global variable and constructors */
__TI_auto_init(); //
/* USER CODE BEGIN (26) */
/* USER CODE END */
/* call the application */
/*SAFETYMCUSW 296 S MR:8.6 <APPROVED> "Startup code(library functions at block scope)" */
/*SAFETYMCUSW 326 S MR:8.2 <APPROVED> "Startup code(Declaration for main in library)" */
/*SAFETYMCUSW 60 D MR:8.8 <APPROVED> "Startup code(Declaration for main in library;Only doing an extern for
main(); //
Updates Available
Updates are available for your software.

부트코드 분석

<FIN>