Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 장성환
redmk1025@gmail.com

대용량 데이터 처리하는게 **coretex A**
coretex R 선박**,** 차량 등등

Reset handler // 초기화 재설정

cp10 cp11 은 부동 소수점을 사용할 수 있게 함**.**

FPU (부동소수점 장치**)**

HSE clock bypass (외부 발진기 사용 할지**)**
HSE clock ready flag (서지 발생 피하기 위하여 오실레이터 준비되면 사용할 수 있도록 함**)**
RCC_PLLCFGR : phase locked loop (clock 생성 회로**) FFT** 하는 칩
-초기화 이유 **? pll** 회로는 기본적으로 피드백을 받음**.** 아무것도 하지말라고 초기화

AHB/APBs data 의 이속 결정

```
; Reset handler
Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
        IMPORT  SystemInit
        IMPORT  __main

                ;FPU settings
                LDR     R0, =0xE000ED88         ; Enable CP10,CP11
                LDR     R1,[R0]
                ORR     R1,R1,#(0xF << 20)
                STR     R1,[R0]

                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP

; Dummy Exception Handlers (infinite loops which can be modified)

NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP
HardFault_Handler₩
                PROC
                EXPORT  HardFault_Handler         [WEAK]
                B       .
```

LDR R0, =0xE000ED88

0xE000ED88 은 **CPACR** 이 주소가 저장된 어드레스이다**. R0** 에 **0xE000ED88**

4.6.1 Coprocessor Access Control Register

The CPACR register specifies the access privileges for coprocessors. See the register summary in *Cortex-M4F floating-point system registers* for its attributes. The bit assignments are:

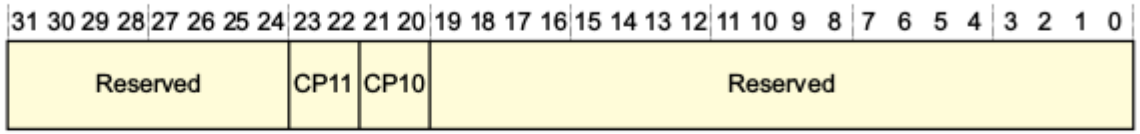


Table 4-50 CPACR register bit assignments

Bits	Name	Function
[31:24]	-	Reserved. Read as Zero, Write Ignore.
[2*n*+1:2*n*] for *n* values10 and 11	CP*n*	Access privileges for coprocessor *n*. The possible values of each field are: `0b00` = Access denied. Any attempted access generates a NOCP UsageFault. `0b01` = Privileged access only. An unprivileged access generates a NOCP fault. `0b10` = Reserved. The result of any access is `Unpredictable`. `0b11` = Full access.
[19:0]	-	Reserved. Read as Zero, Write Ignore.

LDR R1, [R0]

R0 의 주소값을 참조하여 변수를 **R1** 에 저장한다**. (**즉**, R0=CPACR** 의 주소이므로 **CPACR** 레지스터의 값을 저장하는 것이다**.)**

ORR R1, R1, #(0xF << 20)

R1 | (0xF << 20)

-> **R1 | 0b 0000 0000 0000 0000 0000 0000 0000 1111**

-> **R1 | 0b 0000 0000 1111 0000 0000 0000 0000 0000 (CPACR** 의 레지스터에서 **20 ~ 23** 까지의 비트가 **CP11** 과 **CP10** 을 나타낸다**.** 이 값을 **1** 넣어줌**)**

각각을 **0b11** 로 하면**, coprocessor cp10, cp 11** 에 **full access** 가능**. (**유저와 시스템 모두 부동 소수점이 가능케 한다 **FPU** 에 **FULL access** 가능 **)**

STR R1, [R0]
위에서 마스킹 된 **R1 (CP10** 과 **CP11** 이 **0b11** 로 초기화 된 상태**)**를 **R0** 주소를 참조하여 값을 저장한다**.**
(즉**, R0** 는 **CPACR** 을 가리키므로 **CPACR** 의 레지스터 자체의 값을 마스킹 된 **R1** 으로 변경시키는 것**)**

LDR R0, =SystemInit
상단에 선언된 **IMPORT SystemInit** 을 통하여 이미 **C** 언어에서 구현한 함수포인터를 **SystemInit** 로 가져왔다**.**
따라서 **R0 = SystemInit addr** 이다**.**

BLX R0
branch link x 복귀주소 **lr** = 다음 명령어 **LDR R0, __main** 이 들어가며 다음 실행은 **R0** 가 가리키는 주소값을 실행
(즉**, R0** 는 **SystemInit addr** 이므로 해당 함수로 들어감**)**

LDR R0, =0xE000ED88
LDR R1, [R0]
ORR R1, R1, #(0xF << 20)
STR R1, [R0]
LDR R0, =SystemInit
BLX R0

R0 = *0xE000ED88 = &CPACR
R1 = *R0 = CPACR
R1 |= 0xF << 20
***R0 = R1 = 0b 0000 0000 1111 0000 0000 0000 0000 0000**
R0 = *SystemInit
lr = &LDR R0, __main, goto R0(SystemInit)

```
/**
  * @brief  Micro Controller System을 설정한다.
  *         Embedded Flash Interface, PLL을 초기화하고 SystemFrequency 변수를 갱신한다.
  * @param  None
  * @retval None
  */
void SystemInit(void)
{
  /* RCC Clock 구성을 Default Reset State로 reset(재설정)한다. */
  /* Set HSION bit */
  RCC->CR |= (uint32_t)0x00000001;

  /* Reset CFGR register */
  RCC->CFGR = 0x00000000;

  /* Reset HSEON, CSSON and PLLON bits */
  RCC->CR &= (uint32_t)0xFEF6FFFF;

  /* Reset PLLCFGR register */
  RCC->PLLCFGR = 0x24003010;

  /* Reset HSEBYP bit */
  RCC->CR &= (uint32_t)0xFFFBFFFF;

  /* 모든 Interrupt를 비활성화한다. */
  RCC->CIR = 0x00000000;

#ifdef DATA_IN_ExtSRAM
  SystemInit_ExtMemCtl();
#endif /* DATA_IN_ExtSRAM */

  /* System Clock Source, PLL 곱셈기, 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다. */
  SetSysClock();

  /* Offset Address를 더한 Vector Table 위치를 구성한다. */
#ifdef VECT_TAB_SRAM
  SCB->VTOR = SRAM_BASE | VECT_TAB_OFFSET; /* 내부 SRAM에 Vector Table 재배치 */
#else
  SCB->VTOR = FLASH_BASE | VECT_TAB_OFFSET; /* 내부 FLASH(NAND)에 Vector Table 재배치 */
#endif
}
```

RCC register (reset & clock control for MCU)

RCC->CR |= (unit32_t)0x00000001;

6.3.1 RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		PLLSAIRDY	PLLSAION	PLLI2SRDY	PLLI2SON	PLLRDY	PLLON	Reserved				CSSON	HSEBYP	HSERDY	HSEON
		r	rw	r	rw	r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSIRDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

HSION = **1** 로 세팅

Bit 0 **HSION**: Internal high-speed clock enable
Set and cleared by software.
Set by hardware to force the HSI oscillator ON when leaving the Stop or Standby mode or in case of a failure of the HSE oscillator used directly or indirectly as the system clock. This bit cannot be cleared if the HSI is used directly or indirectly as the system clock.
0: HSI oscillator OFF
1: HSI oscillator ON

HSI 오실레이터를 스탑**,** 대기**,** 실패 등등 의 경우에 강제로 **ON** 시키는 비트
시스템 클락으로 **HSI** 가 직접적이든 간접적이든 사용이 되었다면 이 비트는 **clear** 가 불가능 하다**.**

RCC->CFGR |= (unit32_t)0x00000000;

6.3.3 RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved		HPRE[3:0]				SWS1	SWS0	SW1	SW0
rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	r	r	rw	rw

대략 오실레이터 종류 및 상태 설정하는 레지스터 같다**.**

RCC->CR |= (unit32_t)0xFEF6FFFF;

Bit 24 PLLON: Main PLL (PLL) enable
Set and cleared by software to enable PLL.
Cleared by hardware when entering Stop or Standby mode. This bit cannot be reset if PLL clock is used as the system clock.
0: PLL OFF
1: PLL ON
0 으로 세팅

Bit 19 CSSON: Clock security system enable
Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if an oscillator failure is detected.
0: Clock security system OFF (Clock detector OFF)
1: Clock security system ON (Clock det 팅 ector ON if HSE oscillator is stable, OFF if not)
0 으로 세팅

Bit 18 HSEBYP: HSE clock bypass
Set and cleared by software to bypass the oscillator with an external clock. The external clock must be enabled with the HSEON bit, to be used by the device.
The HSEBYP bit can be written only if the HSE oscillator is disabled.
0: HSE oscillator not bypassed
1: HSE oscillator bypassed with an external clock

.... 비트 마스킹으로 레지스터 값을 조작하여 원하는 **Hz** 를 만드는게 목표이다**.**

HSI, HSE, PLL

HSI 는 칩 내부에 존재하는 오실레이터
HSE 는 칩 외부에 존재하는 오실레이터
PLL 은 **HSI** 나 **HSE** 를 선택하여 원하는 주파수로 고정 **or** 고주파로 변환한다**.**
클록 트리를 보면 **HSI, HSE, PLL** 중 **SW** 로 선택하여 출력하고**,**
분주기를 통하여 원하는 **Hz** 를 구성하여 원하는 곳에 사용할 수 있도록 레지스터를 조작한다**.**

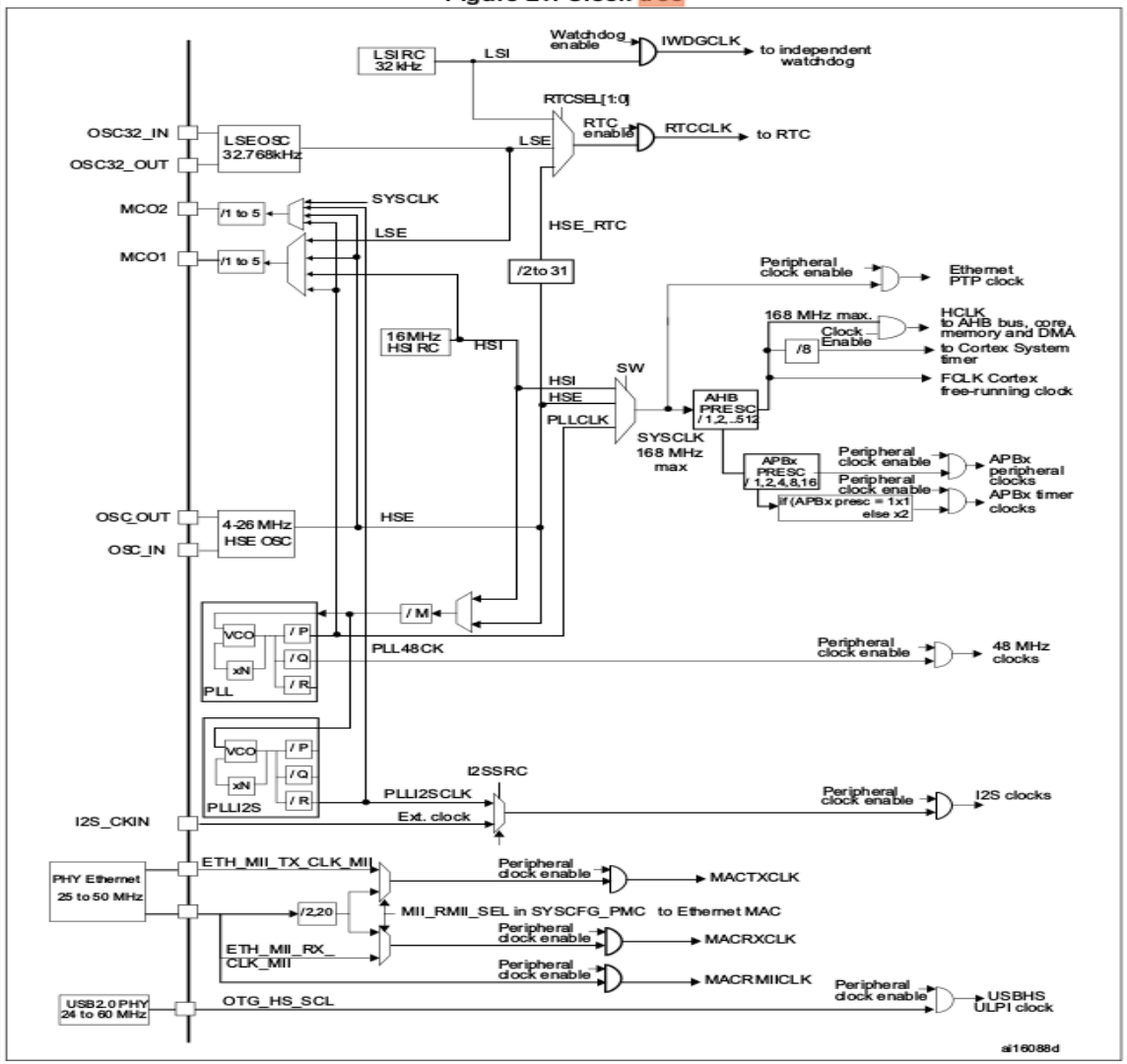
```
/**
  * @brief  System Clock Source, PLL 곱셈기 & 나눗셈기, AHB/APBx Prescalers와 Flash 설정을 구성한다.
  * @Note   이 함수는 RCC Clock 구성을 Default Reset State로 Reset하기 위해 단 한 번만 호출된다.
  * @param  None
  * @retval None
  */
static void SetSysClock(void)
{
/*****************************************************************************/
/*                  PLL (clocked by HSE)을 System Clock Source로 사용한다.                  */
/*****************************************************************************/
  __IO uint32_t StartUpCounter = 0, HSEStatus = 0;

  /* HSE를 활성화 */
  RCC->CR |= ((uint32_t)RCC_CR_HSEON);

  /* Time Out되서 종료되거나 HSE가 종료될때까지 대기한다. */
  do
  {
    HSEStatus = RCC->CR & RCC_CR_HSERDY;
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

  if ((RCC->CR & RCC_CR_HSERDY) != RESET)
  {
    HSEStatus = (uint32_t)0x01;
  }
  else
  {
    HSEStatus = (uint32_t)0x00;
  }
```

HSE_STARTUP_TIMEOUT의 값은 0x05000

RESET = 0

```
  if (HSEStatus == (uint32_t)0x01)
  {
    /* High Performance Mode를 활성화하고, System Frequency를 168 MHz로 올린다. */
    RCC->APB1ENR |= RCC_APB1ENR_PWREN;
    PWR->CR |= PWR_CR_PMODE;          RCC->APB1ENR |= RCC_APB1ENR_PWREN;

    /* HCLK = SYSCLK / 1*/
    RCC->CFGR |= RCC_CFGR_HPRE_DIV1;

    /* PCLK2 = HCLK / 2*/
    RCC->CFGR |= RCC_CFGR_PPRE2_DIV2;

    /* PCLK1 = HCLK / 4*/
    RCC->CFGR |= RCC_CFGR_PPRE1_DIV4;

    /* main PLL을 구성한다. */
    RCC->PLLCFGR = PLL_M | (PLL_N << 6) | (((PLL_P >> 1) -1) << 16) |
                   (RCC_PLLCFGR_PLLSRC_HSE) | (PLL_Q << 24);

    /* main PLL을 활성화 */
    RCC->CR |= RCC_CR_PLLON;

    /* main PLL이 준비될때까지 대기한다. */
    while((RCC->CR & RCC_CR_PLLRDY) == 0)
    {
    }

    /* Flash Prefetch, Instruction Cache, Data Cache를 구성하고 대기 상태 */
    FLASH->ACR = FLASH_ACR_ICEN |FLASH_ACR_DCEN |FLASH_ACR_LATENCY_5WS;

    /* System Clock Source로 main PLL을 선택한다. */
    RCC->CFGR &= (uint32_t)((uint32_t)~(RCC_CFGR_SW));
    RCC->CFGR |= RCC_CFGR_SW_PLL;

    /* System Clock Source로 main PLL이 사용될때까지 대기한다. */
    while ((RCC->CFGR & (uint32_t)RCC_CFGR_SWS ) != RCC_CFGR_SWS_PLL);
    {
    }
  }
  else
  { /* HSE가 Start-Up에 실패하면 Application은 잘못된 Clock을 구성할 것이다.
       사용자(학생분들)가 이러한 오류를 다루기 위한 Code를 이곳에 추가하면 된다. */
  }
}
```

Figure 21. Clock tree



1. For full details about the internal and external clock source characteristics, refer to the Electrical characteristics section in the device datasheet.

system_call.c

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>

int main(void){
   int i;
   unsigned int test_arr[7] = {0};

   register unsigned int *r0 asm("r0") =0;
   register unsigned int r1 asm("r1") =0;
   register unsigned int r2 asm("r2") =0;
   register unsigned int r3 asm("r3") =0;
   register unsigned int r4 asm("r4") =0;
   register unsigned int r5 asm("r5") =0;
   register unsigned int r6 asm("r6") =0;
   register int r7 asm("r7") =0;

   r0 = test_arr;

   asm volatile("mov r1, #0x3\n" // r1 = 3
                "mov r2, r1, lsl #2\n" // r2 = r1 * 2^2 = 12
                "mov r4, #0x2\n" // r4 = 2
                "add r3, r1, r2, lsl r4\n" // r3 = 3 + 12 * 2^2 = 51
                "stmia r0!, {r1, r2, r3}\n"
                "str r4, [r0]\n"
                "mov r5, #128\n"
                "stmia r0, {r4, r5, r6}\n"
                "sub r0, r0, #12\n"
                "ldmia r0, {r4, r5, r6}\n"
                "swp r6, r3, [r0]");
```

```
hwan@HWAN:~/Documents/arm/5-2$ qemu
test_arr[0] = 51
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 128
test_arr[5] = 0
test_arr[6] = 0
r4 = 3, r5 = 12, r6 = 3
r0 = 0x10c4, Parent
r0 = (nil), Child
hwan@HWAN:~/Documents/arm/5-2$
```

```
  for(i=0; i<7; i++){
        printf("test_arr[%d] = %d\n", i, test_arr[i]);
  }

  printf("r4 = %u, r5 = %u, r6 = %u\n", r4, r5, r6);

  r7 = 2; // system call 전용 r7

  asm volatile("swi #0" : "=r" (r0) : "r" (r7) : "memory");
  // swi software interrupt 약자
  // asm 지시어 -> / 명령어 / 출력 / 입력 / 특수 지시어 / 순으로 넣는다.
  // instruction schedulling 을 막는다. 서로의 종속성이 없는 상황에서 명령어의 위치를 바꿀수 있다
  // 즉 해당 명령어 swi 가 끝나기 전까지 ins sch 를 막는다.
  // 이유는 다른 명령어를 스케쥴링 하는 과정에서 r7 의 값이 변할 수 있기 때문이다.
  // 특수 지시어 memory 는 이러한 기능을 수행한다.

  if(r0 > 0)
        printf("r0 = %p, Parent\n", r0);
  else if(r0 == 0)
        printf("r0 = %p, Child\n", r0);
  else{
        perror("fork() ");
        exit(-1);
  }

  return 0;
}
```

aduino proc

int led = 2;
 	
void setup(){
  pinMode(led, OUTPUT);
}
 	
void loop(){
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
	pinMode(led, OUTPUT);
	
	GPIO 구조의 Direction 레지스터를 출력으로 설정 Pull up 을 쓸건지 Pull Down 을 쓸 건지 지정함. 동작 주파수를 지정함 보편적으로 25 나 50 임.
	
	digitalWrite(led, HIGH);
	
	GPIO 구조 데이터 입출력 레지스터
	값을 0 을 넣거나 1 을 넣으면 ON/OFF 가 된다.