TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

2018-05-02 (46 회차)

강사: Innova Lee(이상훈)
gcccompil3r@gmail.com
학생: 정유경
ucong@naver.com

1. STM32F407IGT6 의 부트 코드 분석 (Coretex M4)

참고자료

Cortex-M4 Device Generic User Guide:
http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf
Cortex-M4 Device Technical Reference Manual:
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf
RM0090 Reference Manual(STM32F407):
http://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf

[1] Reset_Handler 분석

```
; 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       .
```

1. R0 = E000ED88 를 저장한다

Address	Name	Type	Reset	Description
0xE000ED88	CPACR	RW	0x00000000	Coprocessor Access Control Register

- CPACR 레지스터는 보조프로세서에 대한 접근권한을 설정한다.
- 여기서 보조프로세서는 Floating Point Unit (FPU)

2. R1 = R0 주소의 값
CPAC 레지스터의 값을 R0 에 저장한다.

3. R1 = R1 | 0x000F00000
Table 4-50 CPACR register bit assignments 를 보면,
F (1111) 을 CP11(23:22) CP10(21:20)비트에 설정하고 있다.
CPn : Access privileges for coprocessor n

0b11 = Full access.

4. R0 의 주소로 가서 CPACR 에 R1 값을 셋팅한다.
(CPACR 레지스터의 값을 변경한다)

5. R0 에 SystemInit 함수의 주소를 저장한다 (R0 는 함수포인터가 된다)
6. R0 의 주소로 점프한다
SystemInit() 함수 호출
7. R0 에 main 함수의 주소를 저장한다
8. R0 의 주소로 분기한다 (드디어 코드상의 메인함수가 돌아간다)

[2] SystemInit() 함수 내부 분석

```
/**
  * @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
}
```

SystemInit 함수 내부로 들어간다

Reset and clock control for
STM32F42xxx and STM32F43xxx (RCC)
→ RCC 는 STM32F407 의 Reset and clock control 과 관련된 것임을 알수 있다.

RCC clock control register (RCC_CR)
Bit 0 HSION: Internal high-speed clock enable
1: HSI oscillator ON
→ RCC 구조체의 CR 멤버의 0 번 비트에 1 을 셋팅한다.
→ Figure 21. Clock tree 의 16Mhz HSI RC 발진기를 ON 한다는 뜻이다

RCC clock configuration register (RCC_CFGR)
Reset value: 0x0000 0000
→ RCC 구조체의 CFGR 멤버에 0x0000000 을 저장하여 초기화한다.

RCC clock control register (RCC_CR)
Bit 24 PLLON: Main PLL (PLL) enable
0: PLL OFF
Bit 19 CSSON: Clock security system enable
0: Clock security system OFF (Clock detector OFF)
Bit 16 HSEON: HSE clock enable
0: HSE oscillator OFF
→ RCC_CR &= 0xFEF6FFFF = 0xF(1110)F(0110)FFFF
→ 24, 19, 16 번 비트에 0 을 셋팅한다.
→ PLL, Clock security system, HSE 발진기를 끈다

RCC PLL configuration register (RCC_PLLCFGR)
Reset value: 0x2400 3010
→ RCC_PLLCFGR: PLL clock outputs 을 결정하는 레지스터이다. 이 레지스터를 초기화한다.

RCC clock control register (RCC_CR)
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
→ 0xFFF(1011)FFFF 이므로 19 번 비트를 0 으로 셋팅하고 있음을 알 수 있다.
→ HSE(external clock)을 bypass 시키지 않는다는 뜻이다.

RCC clock interrupt register (RCC_CIR)
Reset value: 0x0000 0000
→ RCC 구조체의 CR 멤버에 0x00000000 을 저장한다
→ **모든** interrupt **를 비활성화 한다는 뜻이다**.

SetSysClock() 함수 호출

[3] SetSysClock( ) 함수 내부 분석

```
/**
  * @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;                    HSE_STARTUP_TIMEOUT의 값은 0x05000
    StartUpCounter++;
  } while((HSEStatus == 0) && (StartUpCounter != HSE_STARTUP_TIMEOUT));

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

SetSysClock() 함수 내부로 들어간다

변수가 2 개 선언되어 있다. (StartUpCounter, HSEStatus)

RCC-> CR 을 RCC_CR_HSEON 매크로로 변경해주고 있다
HSE 를 활성화 하는 동작이므로 RCC_CR_HSEON 은 16 번 비트에 1 나머지가 0 인 매크로임을 예상할 수 있다.
(Bit 16 HSEON: HSE clock enable)

조건에 관계없이 한번은 실행되므로 바로 Do while 문으로 들어간다.
RCC-> CR & RCC_CR_HSERDY
→ HSE clock ready 비트(17 번 비트)에 1 이 설정되어 있는지를 검사하여 HSEStatus 변수를 설정한다.
(Bit 17 HSERDY: HSE clock ready flag : Set by hardware to indicate that the HSE oscillator is stable)

카운터를 증가시킨다

HSE oscillator 가 stable 하지 않거나, 카운터가 0x05000 이 아니면 while 루프를 계속 반복한다
따라서 HSE oscillator 가 stable 하거나, 카운터가 0x05000 이면 루프를 빠져나온다.
카운터를 셋팅해주는 이유는 무한루프를 피하기 위해서 이다.

HSE oscillator 가 stable 하면 if 문으로 들어간다
HSEStatus = 0x01 로 설정한다.

```
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. */
 }
}
```

HSEStatus 가 0x01 이므로 if 문으로 들어간다
(HSEStatus 를 플래그로 사용하였다)

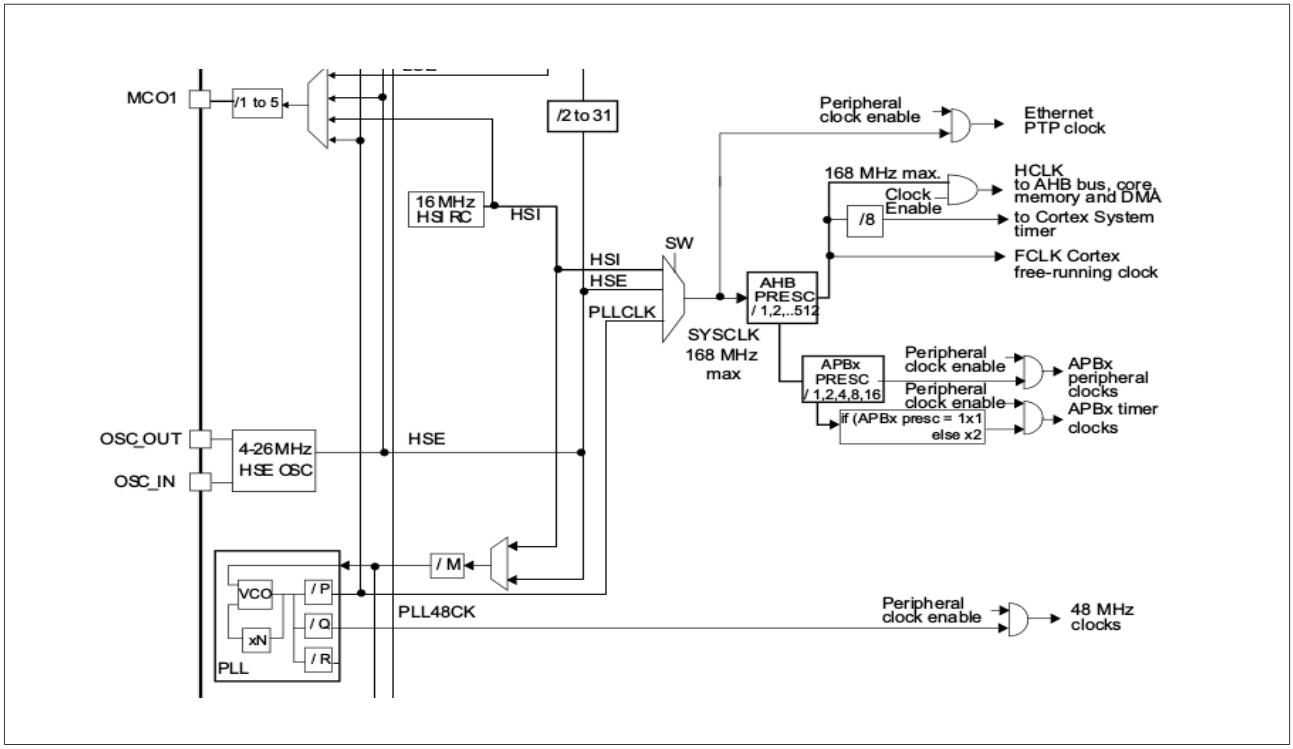
RCC 구조체의 APB1ENR 멤버를 RCC_APB1ENR_PWREN 매크로로 설정해준다.
RCC APB1 peripheral clock enable register(RCC_APB1ENR)
Bit 28 PWREN: Power interface clock enable
1: Power interface clock enable
Figure 21. Clock tree 에서 APBx PRESC 을 확인할 수 있다

PWR power control register (PWR_CR)
PWR 구조체의 CR 멤버에 PWR_CR_PMODE 를 설정한다

RCC clock configuration register (RCC_CFGR)
RCC_CFGR_HPRE_DIV1 // 168Mhz
RCC_CFGR_PPRE2_DIV2 // 84Mhz
RCC_CFGR_PPRE1_DIV4 매크로를 설정하고 있다. // 42Mhz

Bits 7:4 HPRE: AHB prescaler
control AHB clock division factor.
0xxx: system clock not divided
Bits 15:13 PPRE2: APB high-speed prescaler (APB2)
control APB high-speed clock division factor.
100: AHB clock divided by 2
Bits 12:10 PPRE1: APB Low speed prescaler (APB1)
control APB low-speed clock division factor.
101: AHB clock divided by 4

2. Clock Tree 동작 과정 설명(Figure 21. Clock tree)



16Mhz 의 내부클럭(HSI)이 PLL 로 들어가서 더 높은 주파수로 증폭된다.
외부클럭(HSE)은 4-26Mhz 를 사용한다
HSI, HSE, PLL_CLK 은 SW 에 의해 용도에 따라 시스템에서 선택하여 사용한다
이더넷은 빠른 속도를 위해 최대클럭인 168Mhz 를 그대로 사용한다.
AHB 에서 분주한 클럭은 HCLK, System timer, FCLK 에 사용된다
AHB 에서 분주한 클럭을 한번더 분주하여 낮춘 클럭을 Peripheral, APB timer 에 사용한다.

FCLK : cpu 에서 사용되는 클럭
HCLK : AHB 버스에 사용되는 클럭(메모리 컨트롤러, 인터럽트 컨트롤러, LCD 컨트롤러, DMA, USB 호스트 등에서 사용)
PCLK : APB 버스에 사용되는 클럭(WDT, 타이머, ADC, UART, GPIO, RTC, SPI 등에서 사용)

위상동기회로(PLL; Phase-Locked Loop)
입력 신호와 출력신호에서 되먹임된 신호와의 위상차를 이용해 출력신호를 제어하는 시스템을 말한다. 입력된 신호에 맞추어 출력 신호의 주파수 조절이 목적이다. 되먹임 루프 인 현재 출력 신호의 주파수 디바이더 결과와 입력된 신호와의

위상차를 검출하고, 검출된 위상차를 오차로 판단하여 오차가 줄도록 VCO 의 입력전압을 조절함으로써 출력 주파수를 변경하도록 조절한다. 입력과 출력의 되먹임 위상차가 동기되면 위상 잠금이 되고, 잠금 상태가 유지되도록 입력에 대한 출력의 주파수를 조절한다.
입력과 출력의 주파수 차이는 결국 주파수 디바이더 N 에 따라 달라진다. N 의 배수에 따라, 출력신호의 주파수는 입력신호의 주파수의 N 배가 된다. 대부분의 PLL 에서 입력에 비해 출력의 주파수가 높게 발진한다.

3. ARM 어셈블리로 System Call 호출하기

```
#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"
		"mov r2, r1, lsl#2\n"
		"mov r4, #0x2\n"
		"add r3,r1,r2,lsl r4\n"
		"stmla r0!,{r1,r2,r3}\n"
		"str r4, [r0]\n"
		"mov r5, #128\n"
		"mov r6, r5, lsr #3\n"
		"stmla r0, {r4,r5,r6}"
		"sub r0,r0, #12\n"
		"ldmia r0, {r4,r5,r6}\n"
		"swp r6, r3, [r0]");

	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;
        /* calling convention :  r7 은  System call 번호를 저장한다
        (fork 시스템 콜의 고유번호는 2 번)*/

    asm volatile("swi #0" : "=r" (r0) : "r" (r7) : "memory");

        /* 어셈블리 지시어  (명령어: 출력: 입력: 특수지시어) */

        /* 시스템콜 fork 를 호출한다
        1. “swi #0” : swi 0 을 호출하여 swi exception 을 발생시킨다
        (http://jake.dothome.co.kr/exception-1/)
        2. “=r” (r0) : 출력을 r0 레지스터에 저장한다
        3. "r" (r7) : 입력으로 시스템 콜 번호(2:fork)를 전달한다
        4. "memory" 특수지시어가 설정되어 있으면, *memory barrier 를 설정한다는 뜻이다
        → memory barrier 가 설정되면,  명령어가 끝날때까지 *instruction scheduling 을 하지 않는다
        (instruction scheduling 하면 r7 의 값이 변경될 수 있기 때문이다)
        */

        /* r0 에는 fork()의 반환값이 저장된다*/
    if(r0 >0) // Parent
        printf("r0 = %p, Parent\n", r0);
    else if(r0==0) // Child
        printf("r0 = %p, Child\n, r0");
    else // error
    {
        perror("fork()");
        exit(-1);
    }

    return 0;

}

test_arr[0] = 51
test_arr[1] = 12
test_arr[2] = 51
test_arr[3] = 2
test_arr[4] = 128
test_arr[5] = 16
test_arr[6] = 0
r4 = 3, r5=12, r6=3
r0 = 0x299d, Parent
r0 = (nil), Child
```

메모리 배리어(memory barrier)는 중앙 처리 장치나 컴파일러에게 특정 연산의 순서를 강제하도록 하는 기능이다.

중앙 처리 장치에서는 비순차적 명령어 처리 기법을 통해 연산 결과에 영향이 가지 않도록 연산의 순서를 뒤바꿀 수 있으며, 컴파일러에서도 역시 비슷한 최적화를 수행한다. 하지만, 이러한 기능은 여러 스레드가 동시에 돌아가는 경우, 코드의 실행 순서가 바뀌어 실행되는 동안 다른 스레드에서 그 부분에 대한 메모리를 접근하여 잘못된 결과를 내놓을 수 있다. 따라서 특정 부분에 대하여 실행 순서를 강제하는 메모리 배리어를 놓아야 한다.

(https://ko.wikipedia.org/wiki/%EB%A9%94%EB%AA%A8%EB%A6%AC_%EB%B0%B0%EB%A6%AC%EC%96%B4)

Instruction Scheduling

→ CPU 의 파이프라인에서 스톨이 발생할 수 있다. 이때, 연산이 끝나지 않았을때 연산할 수 없다.
따라서 서로의 종속성이 없을 때 명령어 위치를 바꿀 수 있다 (*OoO: 비순차실행)

OoO(Out-of-Order)인 비순차 실행

데이터 의존성이 존재하게되면 어쩔 수 없이 파이프라인 지연(Stall)이 발생하게 된다. 이를 최소화하기 위해 앞서 실행했던 코드와 의존성이 없는 코드를 찾아서 아래의 의존성이 있는 코드 위로 끌어올려서 실행하는 방식이다.