**Bootloader code : Cortex-M4(32bit)**

| | | | |
|---|---|---|---|
| **노트북:** | SW | | |
| **만든 날짜:** | 2018-05-02 오전 10:28 | **수정한 날짜:** | 2018-05-03 오전 12:30 |
| **작성자:** | 정상요 | | |

2018. 5. 2 수 – 46회차

과정 : TI, DSP, Xilinx Znq FPGA, MCU 기반의 프로그래밍 전문가 과정

Prof. 이상훈
gcccompil3r@gmail.com
Stu. 정상용
fstopdg@gmail.com

Cortex-M4 의 Bootloader code 분석 -> ARM assembly language로 구성되어 있다.
분석 순서 : Code -> Register -> Data sheet(Circuit) or Reference (Data sheet(요약본)를 먼저 확인하고 없을 경우, Reference(디테일) 확인)
*참고
Cortex -M 시리즈는 비교적 간단한 편, Cortex -A 시리즈는 훨씬 복잡.
Register -> CPU에서 General Register 를 통해서만 연산한다. Coprocessor Register은 필요할 때 General Register 에 넣어서 사용한다.
General Register
R0 ~ R15, cpsr
Coprocessr Register(CPU를 보조해줄수 있는 보조 Processor Register)

Generic User Guide

CP10, CP11 - 부동소수점을 표현하기 위한 부분(둘다 활성화 시켜야 한다)(FPU사용가능 - 부동소수점)

BLX = 함수호출 branch link

peripheral(사전적 의미 : 주변장치) -> 주변회로를 관리

RCC -> CR을 확인하고 싶다.
Data sheet 확인
2개의 데이트 시트 오픈
1. RCC register
HS -> 클럭 컨트롤 관련 부분
HSI : RC발진기(Xtal)sdf
INTERNAL HS
EXTERNAL HS
HSE : 크리스탈 기반으로 만든 오실레이터
PLL : 블록으로 되어있다. 소자를 꽂아서 만들수가 없다. 블록들로 분리해서 FPGA로 만든다.

CFGR : 클럭 구성 제어
HSERDY : 초기 튀는 신호가 지나고 정상적인 신호가 들어올때까지 기다리겠다는 의미

PLL : Phase Locked Loop -> 클럭을 생성하는 회로(고주파를 생성)
초기화 이유 : feedback를 받는다. 주파수가 높으면 신호가 안정이 어렵다. floating 값//결국 아무것도 하지말라고 초기값

AHB/APBx  데이터들이 버스를 어떤 속도로 움직이는지 확인
peripheral이 움직이는 버스 : APB

동작주파수를 낮춘다. 주파수가 너무 높으면 열이 너무 많이난다. Peripheral, timer clock을 나눈다. 주파수로 인하여

reference manual -> clock tree 확인(회로도) -> 디지털회로 -> FPGA 사용

---
SetSysClock
PLL의 동작주파수 설정,패리패럴 버스 스피드 진행등

Clock 주파수가 안정화가 되었다.

2. Memory map

## Reset handelr (크게 3가지로 나누어 분석 : Reset_Handler, SystemInit, __main)

```
; 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       .
```

**FPU settings**(상위 4번째 줄까지)
-> FPU : 부동소숫점(실수를 사용하지 못하면, 아무것도 할 수가 없으므로 굉장히 중요한 작업이다.)
1. LDR R0, =0xE000ED88 ;Enable CP10, CP11
*참고 : 주소를 직접 넣어줄 경우에는 "="을 꼭 사용해주어야 한다.
- CP10, CP11 레지스터들을 사용할 수 있게 해주는 Code
- R0에 0xE000ED88 주소값을 넣어준다.
그럼 E000ED88이 무엇인지 확인해야한다.(Googling을 하게 되면, Cortex-M4 Technical Reference Manual: 7.3.1. Enabling the FPU 가 나온다.)
pdf에서 찾게되면, E000ED88에 관한 정보를 찾을 수 있다.
Address : 0xE000ED99
Name : CPACR

Type : RW(Reading & Writing)
Description : Coprocessor Access Control Register
- 이제 CPACR 에 대하여 찾으면 된다.

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:

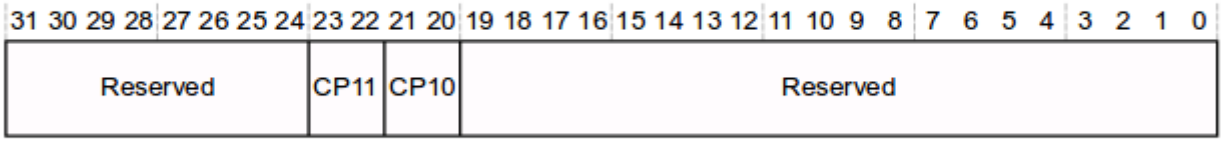


**Table 4-50 CPACR register bit assignments**

마찬가지로, CPACR을 googling하여 Generic User Guide를 통하여 확인한다.

| 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:<br>`0b00` = Access denied. Any attempted access generates a NOCP UsageFault.<br>`0b01` = Privileged access only. An unprivileged access generates a NOCP fault.<br>`0b10` = Reserved. The result of any access is `Unpredictable`.<br>`0b11` = Full access. |
| [19:0] | - | Reserved. Read as Zero, Write Ignore. |

CPACR register에 존재하는 CP11, CP10의 의미 또한 파악.
0b00 ~ 0b11까지 기능 파악 : 현재는 0b11 이므로 Full Access 이다.
2. LDR R1,[R0]
[R0]일 경우, 포인터로 생각 : R0에 들어있는 주소가 가리키는 값이 R1에 들어간다.
3. ORR R1, R1,#(0xF << 20)
#(0xF << 20) : 0000 000F -> 00F0 0000
R1 = R1 | 00F0 0000 : 즉, CP11, CP10을 전부 Set 한다. (Set 반대말, Clear)
4. STR R1, [R0]
R1에 존재하는 값을 R0가 가지고 있는 주소가 가리키는 곳에 저장한다.

5. LDR R0, =SystemInit
R0에 SystemInit을 저장한다.
6. BLX R0
BLX(Branch Link) : 함수호출 명령어

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

1. RCC -> CR |= (unit32_t)0x00000001
RCC -> CR 0번째 bit의 기능을 확인

```
  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. */
  }
}
```

---

asm 시스템콜

asm volatile(명령어 : 입력 : 출력:특수지시사항) - memory : memory barrier : instruction scheduler 를 하지 마라 여기 인스트럭션이 끝날때까지 스케줄러를 하지말아라

r0 = child's pid