

TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 : 황수정

sue100012@naver.com

48 일차 (2018. 05. 04)

목차

- Cortex-R5F 부트로더 분석

Cortex-R5F 부트로더 분석

부트로더는 "HL_sys_intvecs.asm" 파일 안에 작성되어 있다.

```
.sect ".intvecs"
.arm

;-----
; import reference for interrupt routines

.ref _c_int00
.ref phantomInterrupt
.def resetEntry

;-----
; interrupt vectors

resetEntry
    b _c_int00
undefEntry
    b undefEntry
svcEntry
    b svcEntry
prefetchEntry
    b prefetchEntry
dataEntry
    b dataEntry
    b phantomInterrupt
    ldr pc,[pc,#-0x1b0]
    ldr pc,[pc,#-0x1b0]

;-----
```

코드를 보면, 인터럽트 루틴에 대한 참조를 가져오는 것을 정의하고 있다. 즉, 인터럽트 벡터는 인터럽트 처리를 위한 엔트리를 저장하고 있는 것이 된다.

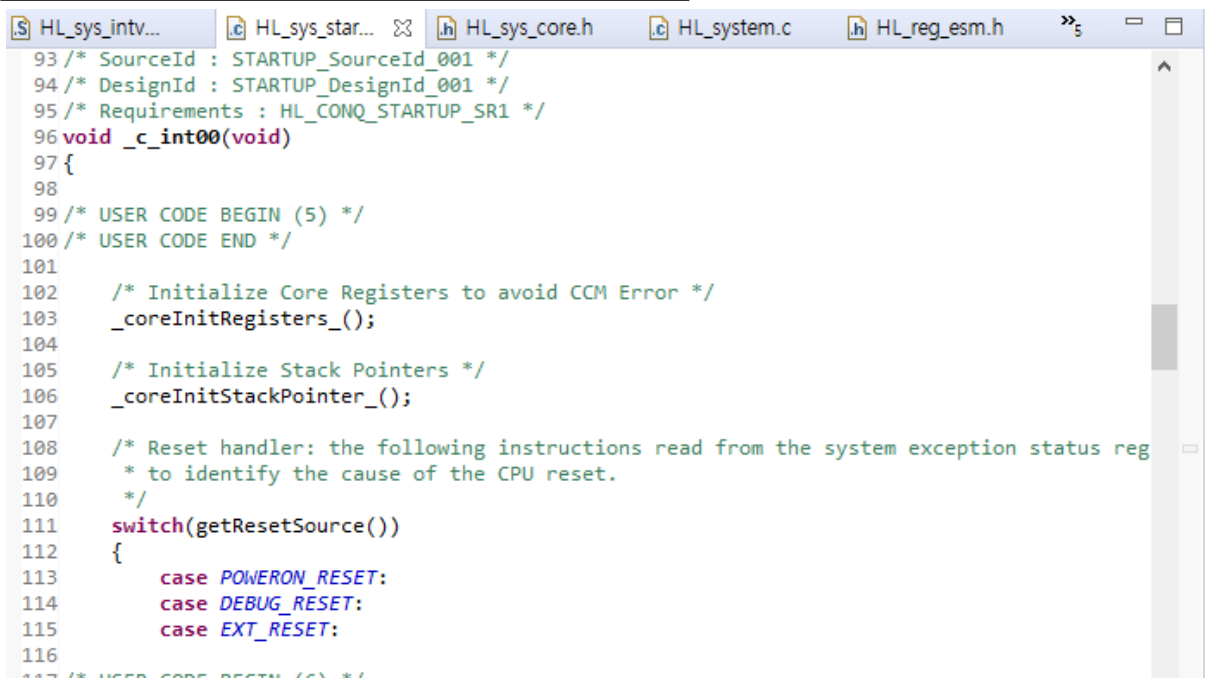
b는 branch로 점프와 같은 명령어이다. 이 명령어를 통해 _c_int00로 점프를 하고 있다.

이때, _c_int00에서 F3을 누르면 해당 함수의 정의부로 갈 수 있고, F2를 누르면 정의를 볼 수 있다.

HL_sys_startup.c에서

void _c_int00(void);로 이동하게 된다.

거기서 F3을 한 번 더 누르면 해당 함수로 이동하게 된다.



```
93 /* SourceId : STARTUP_SourceId_001 */
94 /* DesignId : STARTUP_DesignId_001 */
95 /* Requirements : HL_CONQ_STARTUP_SR1 */
96 void _c_int00(void)
97 {
98
99 /* USER CODE BEGIN (5) */
100 /* USER CODE END */
101
102 /* Initialize Core Registers to avoid CCM Error */
103 _coreInitRegisters();
104
105 /* Initialize Stack Pointers */
106 _coreInitStackPointer();
107
108 /* Reset handler: the following instructions read from the system exception status reg
109  * to identify the cause of the CPU reset.
110  */
111 switch(getResetSource())
112 {
113     case POWERON_RESET:
114     case DEBUG_RESET:
115     case EXT_RESET:
116
117 /* USER CODE BEGIN (6) */
```

이 부트로더를 해석하기 위해서는 안의 함수를 알아야 한다. 먼저 coreInitRegisters 함수로 들어가서 본다. 들어가면 헤더파일에서 void _coreInitRegisters(void); 선언만 된 것을 볼 수 있다.

정의는 다른 파일에서 찾아야 한다는 것을 알 수 있다. 함수 이름을 보면 core 와 관련된 것에 있을 것이다.

```
.def _coreInitRegisters_
.asmfunc

_coreInitRegisters_
; After reset, the CPU is in the Supervisor
mode (M = 10011)
    mov r0, lr
    mov r1, #0x0000
    mov r2, #0x0000
    mov r3, #0x0000
    mov r4, #0x0000
    mov r5, #0x0000
    mov r6, #0x0000
    mov r7, #0x0000
    mov r8, #0x0000
    mov r9, #0x0000
    mov r10, #0x0000
    mov r11, #0x0000
    mov r12, #0x0000
    mov r13, #0x0000
    mrs r1, cpsr
    msr spsr_cxsf, r1
    ; Switch to FIQ mode (M = 10001)
    cps #17
    mov lr, r0
    mov r8, #0x0000
    mov r9, #0x0000
    mov r10, #0x0000
    mov r11, #0x0000
    mov r12, #0x0000
    mrs r1, cpsr
    msr spsr_cxsf, r1
    ; Switch to IRQ mode (M = 10010)
    cps #18
    mov lr, r0
    mrs r1, cpsr
    msr spsr_cxsf, r1
    ; Switch to Abort mode (M = 10111)
    cps #23
    mov lr, r0
    mrs r1, cpsr
    msr spsr_cxsf, r1
; Switch to Undefined Instruction Mode (M =
11011)
    cps #27
    mov lr, r0
    mrs r1, cpsr
    msr spsr_cxsf, r1
; Switch to System Mode ( Shares User Mode
registers ) (M = 11111)
    cps #31
    mov lr, r0
    mrs r1, cpsr
    msr spsr_cxsf, r1
```

HL_sys_core.asm 에 정의되어 있다.

mov r0, lr 을 보면 lr 값을 r0 에 넣는 것 확인할 수 있다. lr 은 복귀주소가 저장 되어 있으므로 이 명령어는 복귀주소를 저장하는 것이라는 것을 알 수 있다.

그 후, r1 ~ r13 까지 0 으로 초기화하는 것을 확인 할 수 있다. 이는 각 범용 레지스터를 0 으로 초기화하는 코드이다. 이는 작업할 때, 엉뚱한 값이 계산 되지 않도록 부트 코드에서 해주어야 하는 작업이다.

mrs 스페셜 레지스터 값을 레지스터에 저장한다는 뜻이다. cpsr 값을 r1 에 저장한다는 것으로 이는 cpsr 에 어떤 값이 있는지 알아야 하기 때문이다.

msr 은 레지스터에서 스페셜 레지스터로 설정하는 것으로 아까 읽은 cpsr 정보를 spsr_cxsf 에 저장하는 것이다.

이는 커널모드에서는 cpsr(유저 모드용)을 쓸 수가 없기 때문에 행해주는 동작이다.

→ 데이터시트를 찾아보면, cpsr 은 Current Program Status Register 이다. 이는 프로그램의 상태를 저장하는 레지스터임을 뜻하는 것이다.

spsr 은 각 모드마다 1 개씩 존재한다. 각각 모드마다 하드웨어적으로 하나씩 가지고 있다는 것이다. 즉, 동작모드마다 레지스터가 다르다는 것이다. 이는 arm 은 동작 모드가 많은데, 이 특수 모드를 다 지정해주었다는 것으로, 레지스터 저장 때문에 하게 되는 push, pop 명령어를 하나라도 줄이는 것이 더 좋

기 때문이다. 이미 값을 저장하고 있다면 SW 적으로 동작이 줄기 때문이다. cxsf 는 슈퍼바이저는 모든 기능을 다 수행해야 하기 때문에 가져온 것이다.

cps #숫자는 m 값을 바꾸어주는데 모드를 변경해주는 것을 뜻한다.

mov lr, r0 는 r0 값을 lr 에 넣어주는 것으로 복귀주소를 저장해 주는 것이다. 그 후 모드마다 사용될 rx 를 초기화하는 것이다. 유저모드랑 커널모드의 동작모드를 맞추어주는 것이다.

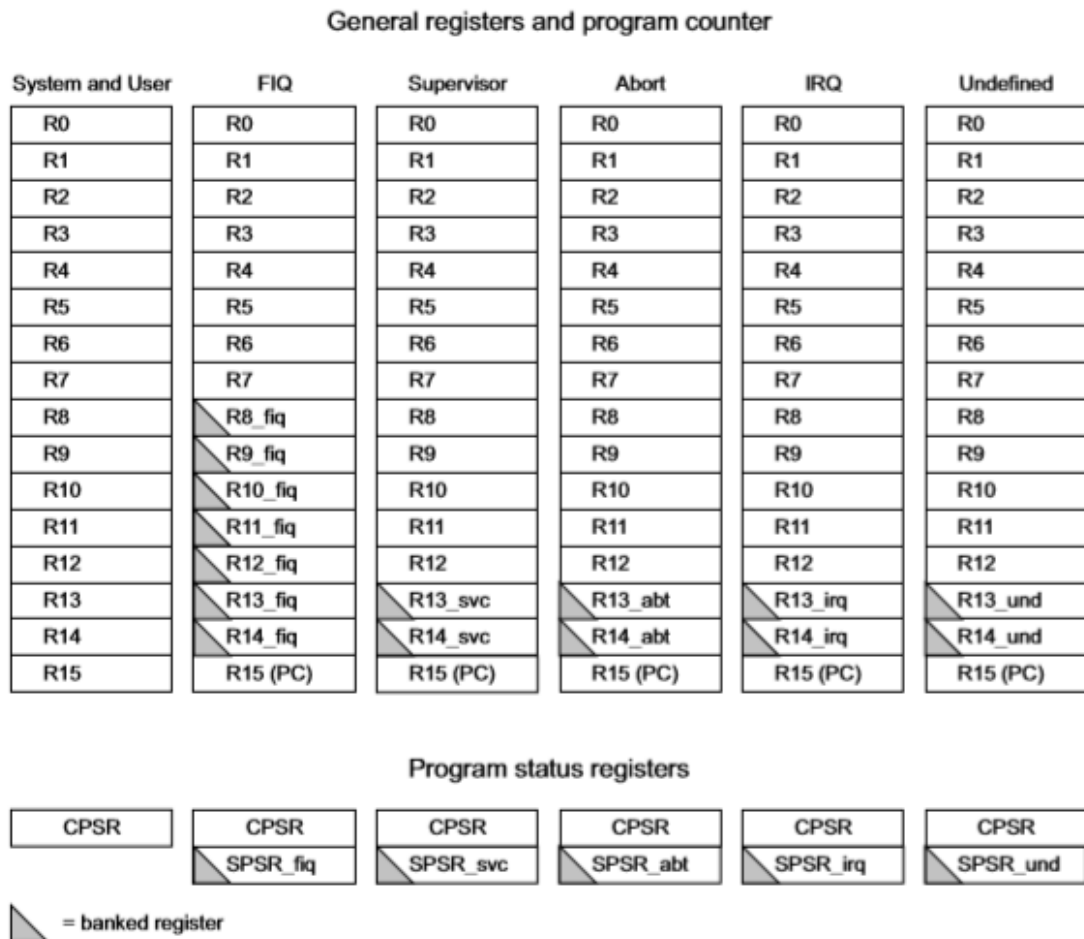


Figure 3-3 Register organization

각 모드마다 초기화되는 레지스터를 나타낸 것이다. 각 모드마다 같은 방식으로 동작한다.

```

mrc p15, #0x00, r2, c1, c0, #0x02
orr r2, r2, #0xF00000
mcr p15, #0x00, r2, c1, c0, #0x02

```

mrc 를 명령어를 데이터 시트에서 찾아보면 이 코드는 CPACR 값을 읽어서 r2 에 저장한다는 것을 알 수 있다.

orr r2, r2, #0xF00000 는 읽어온 CPACR 값에 0xF00000 값을 or 연산하는 것이다. 이는 23~20 번 bit 자리를 1로 바꾸어준다는 것이다.

mcr 의 명령어는 mrc 명령어 바로 아래 정의되어 있는데, 이는 r2 에 저장된 값을 CPACR 레지스터에 저장(기록)한다는 뜻으로 23~20 비트를 설정해주는 것이 된다.

CPACR 을 검색해서 보면 23~20 비트는 FPU 에 관한 것이고 이에 대한 접근 권한을 설정해주는 것이다. 즉, 이 3 줄은 FPU 접근 권한 설정이라는 것을 알 수 있다.

이는 부동소수 제어를 위한 것으로 위는 어셈블리어로 하기 위해 만든 코드가 된다. 이처럼 어셈블리어를 안다면 단가가 내려갈 수 있다. 어셈블리어를 알면 코드로 끝나는데, 어셈블리어를 모른다면 코드가 아닌 회로가 추가되어야 해서 단가가 올라가기 때문이다.

```

mov    r2,      #0x40000000
fmxr   fpexc,   r2

        fmdrr d0,      r1,      r1
        fmdrr d1,      r1,      r1
        fmdrr d2,      r1,      r1
        fmdrr d3,      r1,      r1
        fmdrr d4,      r1,      r1
        fmdrr d5,      r1,      r1
        fmdrr d6,      r1,      r1
        fmdrr d7,      r1,      r1
        fmdrr d8,      r1,      r1
        fmdrr d9,      r1,      r1
        fmdrr d10,     r1,      r1
        fmdrr d11,     r1,      r1
        fmdrr d12,     r1,      r1
        fmdrr d13,     r1,      r1
        fmdrr d14,     r1,      r1
        fmdrr d15,     r1,      r1
next1   bl      next1
next1   bl      next2
next2   bl      next3
next3   bl      next4
next4   bx      r0

        .endasmfunc

```

이제 r2 에 0x40000000 저장하고 이를 이용해 fpexc 를 설정해주는 것이다. 이는 fpexc 레지스터의 30 비트를 1 로 세팅해주는 작업이다. 데이터 시트를 보면 VFP 활성화 비트라는 것을 알 수 있다. 1 로 세팅되면 VFP 기능을 사용할 수가 있다.

그 밑은 d0 ~ d15 를 0 으로 초기화하는 것인데, 이 작업을 하게 된 이유는 fpexc 에 있다. Fmdrr 작업을 하기 전에 이 코드는 부동소수점에 관련된 작업을 하고 있었다. 소수를 표현하기 위해서는 정수부분과 소수부분이 필요하다. 이를 각각 32 비트로 하여 d 레지스터가 반반으로 나누어져서 설정되어있다. 이 부동소수점을 사용해주기 위해선 초기화가 필요(연산하면서 이상한 값을 얻지 않기 위해서)하다. 그래서 이 작업이 이루어지게 된 것이다.

그 밑에 next 로 이동하는 코드들이 있는데, 의미 없어 보이지만 이는 시간을 벌여주기 위해서 하는 작업이다.

위의 fmdrr 이 끝날 때까지 해주는 것인데,

```

_coreInitStackPointer_

        cps    #17
        ldr     sp,      fiqSp
        cps    #18
        ldr     sp,      irqSp
        cps    #19
        ldr     sp,      svcSp
        cps    #23
        ldr     sp,      abortSp
        cps    #27
        ldr     sp,      undefSp
        cps    #31
        ldr     sp,      userSp
        bx      lr

userSp   .word 0x08000000+0x00001000

```

이 작업이 1 클럭에 끝나지 않기 때문이다. 이 작업이 안정화되기 전까지 시간을 벌기 위한 것이다.

.endasmfunc 로 함수가 끝나고 다음 함수로 넘어가면 된다.

coreInitStackPointer 는 스택포인터를 초기화하는 것으로 보여진다. 각자의 스택을 가지고 있고 이것 설정하는 코드인 것이다.

cps 는 모드를 바꾸어주는 명령어이다. Ldr 은 sp 에 fiqsp(Fast interrupt) 값을 저장해준다. 이는 fiq 스택을 잡겠다는

뜻이다. 즉, 스택포인터를 쓴 이유는 mode 에 맞는 주소 공간을 확보하기 위한 것으로, 데이터가 꼬이게 하지 않기 위해서 필요한 것이다.

.word 는 #define 과 같은 뜻으로 어떠한 값을 갖는지 정의해준 것이다.