TI DSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

강사 : Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 : 황수정
sue100012@naver.com
46 일차 (2018. 05. 02)

목차

```
; 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       .
```

※ 어셈블리어 코드에서 ; 은 주석을 쓸 때, 사용한다.

※ = 숫자 는 데이터를 그대로 저장한다. [ ]는 참조하는 것이다.

IMPORT SystemInit → IMPORT 다른 컴퓨터 시스템으로부터 자기 시스템 안에 데이터를 들여놓는 행위. / 함수 선언...?

IMPORT __main

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:

31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0
Reserved		CP11	CP10	Reserved			

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 R0, = 0xE000ED88

- R0 에 0xE000ED88 저장한다. LDR 은 메모리에서 레지스터로 읽어 들이는 명령어이다. 0xE000ED88 를 데이터베이스로 찾아보면 CPACR 로 read, write 가 가능하며, Coprocessor Access Control Register 을 뜻한다. Coprocessor 에 대한 접근권한을 말한다. 0~19, 24~31 은 값이 예약되어 있는 값으로 여기선 신경 쓰지 않아도 된다. 20~23 은 CP10, CP11 로 선언되어 있다.
  ARM 에는 MMU 를 제어하기 위해 별도의 하드웨어를 가진다. 이는 Coprocessor, CP 로, 별도로 존재한다. CP0~15 까지에 레지스터를 가지고 있는데, 각각이 프로세스 동작을 서포트 해주기 위한 동작을 가지고 있다. 여기서 CP10, CP11 은 부동소수점 floating point 를 표현하기 위해 필요한 레지스터이다. 그래서 설정해주어야 한다.

LDR R1, [R0]

- R1 에 [R0]의 주소로 가서 데이터를 가져와 저장한다.

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

- 0xF <<20 어셈블리어는 32 비트이므로 0x0000 000F 를 나타낸다. F 는 2 진법으로 바꾸면 1111 이므로 이를 20 시프트 연산을 하면 1 이 23~20 비트에 존재하게 된다. 0x00F0 0000 으로 바뀌게 되기 때문이다. R1 에 0xF <<20 값을 넣어주고 기존 R1 과 OR 연산을 해준다. 이러면 20~23 비트에다가 1 을 셋팅해주는 것으로 CP10, CP11 에 1 이 셋팅된 것이다. 위의 그림을 보면 0b11 이 되는데 이는 완벽하게 접근할 수 있다라고 되어 있다. 이는 사용자와 커널도 부동 소수점을 다 사용할 수 있다는 뜻이다.
  0b01 은 시스템 즉, 커널만 사용 가능한다. 0b10 은 아무 기능이 없다. 셋팅을 하던 말던 같은 것이다.

STR R1,[R0]

- LDR 과 반대로 레지스터에서 메모리로 저장하는 것이다. [R0]에 R1 을 저장한다.

⇒ 4 줄의 코드가 유저와 커널 모두 부동 소수를 사용 가능하도록 만들어준 것이다.

LDR R0,=SystemInit

- R0 에 SystemInit 이 저장된다. 함수 이름은 포인터로 SystemInit 이라는 함수 포인터를 저장한 것이다.

BLX R0

- R0 주소로 jump 하겠다는 뜻이다.

```
/**
  * @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→CR |= (unit32_t)0x0000000001

- RCC_CR 의 0 비트를 1 로 셋팅해주는 것이다. 0 비트는 HSEON 로 HSE oscillator ON

* ELF

디버깅

전역변수 만드는 것

= 숫자 ← 데이터 그대로 저장한다.

= 쓰면 그대로 가져오고 드그는 참조.

ldr R0, = 0xE R0에 넣는다.

ldr R1, [R0]
↳ 여기 주소로 가서 데이터를 가져와라 (데이터를 읽어서).
↳ R0 주소에 무엇이 있는가.

0xE000ED88 : 구글링을 해보자.
→ 바로 나오는 pdf
→ CPACR.
→ 밑에 무엇이 뜬다 (레지스터에 대해).
↳ 4.6.6 의 설명에 부동소수에 대한 설명이 있다.

ORR R1, R1 # (0xF << 20)
↳ 32bit니까 0x 0000 000F (<< 20).
0x 0000 00F0 4번 << 한 것. (2진수니까).
R1 = R1 | 0x00F0 0000이 된다.
0000 → 31bit
1111 → 23bit ~ 20bit.

STR
↳ 읽어오는 것이 아니라 저장하는 것

0b01 – 시스템만 가능.
0b10 – 아무 기능 X 셋팅을 하던 말던 같은 것.

4줄의 코드가 유저와 커널 모두 부동소수 사용 가능.

LDR R0, = systemInit
↳ 이 저장된다
SystemInit – 함수 포인터.

함수포인터 저장하는 것
(함수의 이름은 함수 포인터).

~~BARBOE~~ BLX R0.
↳ R0 주소로 jump 하겠다.

systemInit 이라는 곳으로 가겠다.

RCC → CR |= (uint32_t)
↳ RCC 검색

or 이니까 0bit 만 1로 셋팅.
HSION → 0번 비트
clock tree. 검색.
217쪽 그림 HSI RC로 보고.
→ SW 가는 것과 밑의 사다리꼴로.
HSI oscillator : OFF / ON.
↳ 발진기.

oscillator가 꺼지면 clock이 꺼진다
↗ Clock 수가 높을수록 성능이 좋다
발진파형. ↳ 발진.

함수 명령을 하면 clock이 해준다. : 명령어 실행에는 clock 필요하다.

CFGR.
and 연산은 0으로 셋팅 : 초기화
or 연산은 1로 셋팅.

RCC → CR &= 0xFFFF.
↳ 그대로 유지. 1111
So, 0으로 하고 싶은 부분을 0으로 하면 0으로 셋팅된다.

0xFEF6 FFFF.
1110 0110 (24, 19, 16)
→ 24, 19, 16은 0으로 해준다.
↓
PLL OFF
PLL을 꺼준다
↓
clock tree를 보다
↓
PLL Box를 꺼준다
clock은 기본적으로 3개 (HSI, HSE, PLL).
↗ 내부에 없을 때 외부를 쓴다.
HSE가 외부. (HSI 내부, 도움.)
26보다 크게 쓰고 싶을 때. PLL을 쓴다

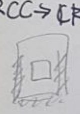
SW 값에 따라
3개 중에 원하는 clock을 쓸 수 있다.
→ 프로그래밍.

Bit 19
→ 보안
보안 발생 레지스터를 끈다.

Bit 16
외부 발진기 끈다.

RCC → PLLCFGR.
→ PLL에 관련된.

RCC → CR. 0x FFFB.
18번째 비트.



HSEBYP.

3번째를, 보면 안다 ←
앞에 꺼를 0으로 만든 이유
bit 0으로 bypassed X
CIR에 0으로 설정.

ifdef SRAM을 장착하면
실행하다
→ SRAM도 초기화 해야해서.
중간 함수는 없으니
→ 메모리를 1개 판다? : 외부에서 메모리를 판 것인지 아닌지. (SD카드!).

초기화를 잘 해야하기 때문.

setsysclock 함수
2 변수 0으로 초기화.
HSEON – 외부 clock을 켠다.
|=
숫자로 되어있지 않지만
→ 위에 디파인 되어 있을 것이다
16bit를 키는 것으로.
숫자 되어 있을 것.
& 연산 : 값을 읽어오는 것.
bit 17가 1이 되었는지 아닌지
확인하는 거 (하드웨어가 해준다.)
set by hardware.

counter++;
↳ 타임아웃 재려고.
while 문 안에 · 준비되면
while 문 빠져나가고 아니면 돈다.
타임아웃이 아니다 / 타임아웃이 정의되어
있는 숫자만큼 무한루프 한계 값.
17번 bit 값. 리셋이 아니면 if

* 분주비
↓
주파수를 낮춘다.
↓
속도 ↓.
↓
AHB
1개 있는데
APB
주파수 ↑
에너지 ↑
탄다는 것.
↓
분주비 2번 거치는 이유
↓
Peripheral
하게 분리 다시.

CPU 주위에 버스로 연결되어있는

if (HSEStatus == (uint)0x01)
정상 set 되었다.

Clock이 3개.
분주로 더.
DIV
↓
분주비 값 설정.
DIV 1은
↳ 0xxx로 설정되어 있다.
(디폴트적으로)

28bit 1이 된다.
PWR 리셋 코드
→ 처음 초기화 코드.
↓
Reset value.
→ 요청할 때 만드는 것.

APB가 하나 있는데
사실 여러 개 더 있다.
X (숫자)는 여러 개 더 있다는 것을 말함.

RCC_CFGR_PPRE2_DIV2

0	0	0	0	1	4	0
31 28	28 25	24 21	19 16	15 12	11 8	7 4

0100

↳ RCC_CFGR_PPRE1_DIV

시스템 콜 어셈블리

```
#include<stdio.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=0x3
                        "mov r2, r1, lsl #2₩n" // r2=0b1100 = 12
                        "mov r4, #0x2₩n" // r4 = 0x2
                        "add r3, r1, r2, lsl r4₩n" // r3 = 48+3=51
                        "stmia r0!, {r1,r2,r3}₩n" // r0 에 r1,r2,r3 를 넣고 위치 값 갱신
                        "str r4, [r0]₩n" // r0 에 r4 레지스터 값을 넣음. 4 번째 배열에 2 가 들어감.
                        "mov r5, #128₩n" // r5 = 128
                        "mov r6, r5, lsr #3₩n" // r6 = 128/8 = 16
                         "stmia r0, {r4, r5, r6}₩n" //r0 에 r4,r5,r6 넣기
                        "sub r0, r0, #12₩n" // r0 의 위치 원래대로
                        "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; // r7 : system call 저장

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

```
        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;
 }
```

아두이노 LED 예제

```
int led = 2;
void setup(){
  pinMode(led, OUTPUT);   //led : 핀번호 , 2 번 핀을 출력모드로 설정
}
void loop(){
    digitalWrite(led, HIGH); //led 켜기
    delay(1000);
    digitalWrite(led, LOW); //led 끄기
    delay(1000);
}
```

pinMode(led, OUTPUT)
GPI0 구조의 Direction 레지스터를 출력으로 설정
Pull Up 을 쓸건지 Pull Down 을 쓸건지 지정함
동작 주파수를 지정함 보편적을 25 나 50 임

digitalWrite(led, HIGH)
GPI0 구조 데이터 입출력 레지스터 값을 0 을 넣거나 1 을 넣으면 ON/OFF 가 됨