

임베디드 프로세서 응용 HW3

21800471

유준호

The screenshot displays an IDE with two main windows: **Registers** and **Disassembly**.

Registers Window:

Register	Value
Core	
R0	0x20000060
R1	0x20000260
R2	0x20000260
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000060
R14 (LR)	0x0800020B
R15 (PC)	0x080003EC
xPSR	0x21000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	973
Sec	0.00009730
FPU	

Disassembly Window:

```
0x080003E8 B003 ADD sp,sp,#0x0C
0x080003EA 4770 BX lr
15: {
16:
→ 0x080003EC B580 PUSH {r7,lr}
0x080003EE B084 SUB sp,sp,#0x10
0x080003F0 2000 MOVS r0,#0x00
0x080003F2 9003 STR r0,[sp,#0x0C]
0x080003F4 2005 MOVS r0,#0x05
17: NVIC_SetPriorityGrouping(5); // [5:0] : subpriority
<
```

Source Code Window (main.c):

```
1 #include "stm32f4xx.h"
2 //21800471 YooJunho
3 void EXTI0_IRQHandler(void);
4
5 void EXTI0_IRQHandler(void) {
6     volatile int32_t i;
7
8     // meanderingless delay
9     for(i=10 ; i>=0 ; i--){
10     }
11 }
12
13
14 int main(void)
15 {
16
17     NVIC_SetPriorityGrouping(5); // [5:0] : subpriority
18
19     NVIC_SetPriority(6, 0x40>>4); // EXTI0 priority = 0x40
20     NVIC_SetPriority(7, 0x40>>4); // EXTI1 priority = 0x40
21
22     NVIC_EnableIRQ(6);
23     NVIC_EnableIRQ(7);
24
25     NVIC_SetPendingIRQ(6);
26
27     while(1){
28     }
29 }
30
```

(a) 17번, 19번, 22번과 25번 라인의 코드의 동작을 각각 설명하시오. (8점)

[NVIC_SetPriorityGrouping(5)]

Cortex-M 프로세서에서는 우선순위는 그룹 우선순위와 하위 우선순위로 나뉜다. 그 둘을 구분하는 것은 SCB에 위치한 AIRCB의 비트 [10:8]에 위치한 PRIGROUP값이다. PRIGROUP에 설정한 값이 5이면 우선순위 8비트 중에서 하위 6비트([5:0])가 하위 우선순위에 속하게 된다. NVIC_SetPriorityGrouping(5)에서는 우선순위 그룹을 5로 설정하여 하위 우선순위에 8비트 중 [5:0]비트가 속하게 된다.

[NVIC_SetPriority()]

IRQn=6인 인터럽트의 우선순위 값을 0x40, IRQn=7인 인터럽트의 우선순위 값을 0x00으로 설정하는 모습입니다. 이 때 >>4 비트 연산을 하는 이유는 STM32f4xx프로세서의 __NVIC_PRIO_BITS의 값이 4이므로 priority인자값을 상위 4비트값만 주어야 하기 때문이다.

[NVIC_EnableIRQ()]

IRQn=6 , IRQn=7의 인터럽트를 활성화합니다. 이 인터럽트가 발생할 때 인터럽트 핸들러 함수가 실행되도록 한다.

[NVIC_SetPendingIRQ()]

이 함수는 지정된 IRQn의 pending bit을 1로 설정한다. 펜딩이 설정된 인터럽트는 처리될 때까지 기다리는 대신 다른 작업을 수행할 수 있게됨.

(b) 디버거를 시작한 다음, 25번 라인과 5번 라인에 breakpoint를 설정하고, Peripherals -> Core Peripherals -> NVIC를 선택 다음, Idx 22번과 23번의 상태를 설명하시오. (E, P, A, Priority) (4점) 그리고 Interrupt Control & State register (scb->ICSR)의 주요 필드들의 상태를 설명하시오 (VECTACTIVE, RETTOBASE, VECTPENDING, ISRPREEMPT, ISRPENDING) (5점)

[실행 결과]

Idx	Source	Name	E	P	A	Priority
22	EXTI Line0 interrupt	EXTI0	0	0	0	0 = 0 s0
23	EXTI Line1 interrupt	EXTI1	0	0	0	0 = 0 s0

현재 EXTI0, EXTI1의 E, P, A, Priority의 필드 모두 0값이므로

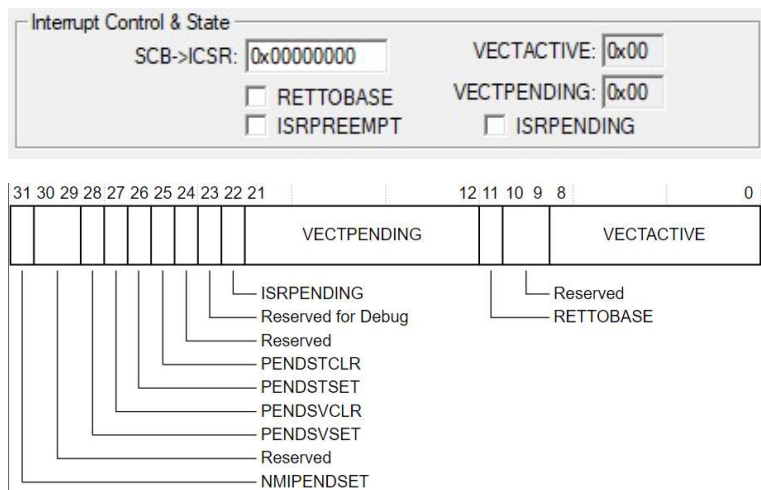
E=0 -> 두 IRQ 모두 비활성 상태이다

P=0 -> 처리할 IRQ가 없다

A=0 -> IRQ가 현재 활성화 되어 실행 중이지 않음

Priority=0 -> 우선순위가 0x00임

을 알 수 있다.



VECTACTIVE : 0x00 -> IRQ가 실행 중이지 않아 현재 0값을 나타내고 있다.

RETTORBASE : 0x0 -> 현재 실행중인 ISR에서 다른 ISR실행될 수 있음을 의미함. 현재 ISR이 끝나면 이전 ISR의 실행 위치로 돌아감(반환주소가 이전 ISR의 실행 위치로)

VECTPENDING : 0x00 -> 처리되지 않은 IRQ의 번호를 나타냄. 즉 대기중인 IRQ의 번호임. 0인 경우에는 대기 중인 IRQ가 없음. 있을 때는 처리해야 할 IRQ의 번호가 저장됨.

ISRPREEMPT : 0x0 -> 현재 처리중인 IRQ에 더 높은 우선순위의 IRQ가 발생했을 때, 현재 IRQ가 중단 가능한지. 1인 경우에는 Preemption이 가능함을 의미. 지금처럼 0인 경우에는 불가능.

ISRPENDING : 0x0 -> 처리되지 않은 IRQ가 있는지 여부를 나타냄. 이 필드가 1인 경우, 처리되지 않은 IRQ가 있음. 0인 경우는 처리되지 않은 IRQ가 없음.

(c) Run 버튼을 클릭하시오. 그러면 25번 breakpoint에서 실행이 일단 멈출 것이다. 이때 SP, 값은 무엇인가? (1점) 그리고 CONTROL register 값이 무엇인지 기록하고, 그 값의 의미를 설명하시오. (3점) 그리고 Peripherals -> Core Peripherals -> NVIC에서 Idx 22 번과 23번의 바뀐 상태를 설명하시오. (E, P, A, Priority) (4점)

R13 (SP) 0x20000648
R14 (LR) 0x0800041D
R15 (PC) 0x0800041E
xPSR 0x41000000
Banked
System
 BASEPRI 0x00
 PRIMASK 0
 FAULTMASK 0
 CONTROL 0x04
Internal
 Mode Thread
 Privilege Privileged
 Stack MSP
 States 1145
 Sec 0,00011450
FPU

```

1  #include "stm321xx.h"
2  //21800471 YooJunho
3  void EXTI0_IRQHandler(void);
4
5  void EXTI0_IRQHandler(void){
6      volatile int32_t i;
7
8      // meaningless delay
9      for(i=10 ; i>=0 ; i--){
10         }
11     }
12
13
14     int main(void)
15     {
16
17         NVIC_SetPriorityGrouping(5);
18
19         NVIC_SetPriority(6, 0x40>>4);
20         NVIC_SetPriority(7, 0x00>>4);
21
22         NVIC_EnableIRQ(6);
23         NVIC_EnableIRQ(7);
24
25         NVIC_SetPendingIRQ(6);
26
27         while(1){
28             }
29     }
30

```

R13(SP) = 0x20000648

CONTROL = 0x04

= 0000 0100

Cortex-M3/M4에서는 프로세서의 동작 모드가 스레드, 핸들러 모드로 구성되어있음. 스레드 모드는 일반 소프트웨어를 실행하는 모드이고, 핸들러 모드는 익셉션을 처리하는 모드임. 핸들러 모드에서는 항상 특권을 가지며, 스레드 모드에서는 특권이 없는 사용자모드 또는 특권모드로 동작할 수 있음.

CONTROL[0] :	CONTROL[1]	CONTROL[2]
0 = privileged thread mode	0 = 디폴트 스택(MSP)	0 = 부동 소수점 관련 명령어가 사용되지 않았다.
1 unprivileged(user) thread mode	1 = 프로세스 스택(PSP)	1 = 부동소수점 관련 명령어가 사용되었다.

CONTROL = 0x04 = 0000 0100(2진수)

여기서 0비트는 0, 1비트는 0, 2비트는 1이다. 따라서 부동소수점 관련 명령어가 사용됨을 알 수 있다.

Idx	Source	Name	E	P	A	Priority
22	EXTI Line0 interrupt	EXTI0	1	0	0	4 = 1 s0
23	EXTI Line1 interrupt	EXTI1	1	0	0	0 = 0 s0

EXTI0과 EXTI1이 Enable되어있음을 알 수 있다. 또한 EXTI0의 Priority값이 0x40의 상위 4비트인 4로 바뀌어 있음을 알 수 있다.

(d) Run 버튼을 클릭하시오. 그러면 5번 breakpoint에서 실행이 일단 멈출 것이다. 이때 SP 값은 무엇인가? Stack에 저장된 내용들을 설명하시오. (7점) 그리고 Peripherals -> Core Peripherals -> NVIC와 Interrupt Control & State register (scb->ICSR)의 주요 필드들에서 바뀐 상태 상태와 (4점) LR에 나타난 EXC_RETURN 값의 의미를 설명하시오. (4점)

Register Window:

Register	Value
R0	0x00000040
R1	0xE000E200
R2	0x00000000
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x200005D8
R14 (LR)	0xFFFFFE9
R15 (PC)	0x080002D8
xPSR	0x41000016

System Peripherals:

BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00

Internal:

Mode	Handler
Privilege	Privileged
Stack	MSP
States	160472663
Sec	16,04726630

Source Code (main.c):

```

1 #include "
2 //21800471
3 void EXTI0
4
5 void EXTI0
6     volati
7
8     // mea
9     for(i=
10 }
11 }
12
13
14 int main(v
15 {
16
17 NVIC_R13(SP) = 0x200005D8

```

Memory 1	
0x200005D8	
0x200005D8: 40 00 00 00	R0 : 0x00000040
0x200005DC: 00 E2 00 E0	R1 : 0xE000E200
0x200005E0: 00 00 00 00	R2 : 0x00000000
0x200005E4: 60 02 00 20	R3 : 0x20000260
0x200005E8: 40 00 00 20	R12 : 0x20000040
0x200005EC: 23 04 00 08	LR : 0x08000423
0x200005F0: 66 03 00 08	PC : 0x08000366
0x200005F4: 00 02 00 41	xPSR : 0x41000200
0x200005F8: 00 00 00 00	
0x200005FC: 00 00 00 00	

서브 인터럽트로 들어가기전 사용하는 레지스터를 스택킹하여 보존하는 모습을 볼 수 있다

Registers

Register	Value
R0	0x00000040
R1	0xE000E200
R2	0x00000000
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000644
R14 (LR)	0x08000423
R15 (PC)	0x08000368
xPSR	0x41000000

Banked
System
Internal

Mode
Privilege
Stack
States
Sec

Thread
Privileged
MSP
1352
0.00013520

FPU

Disassembly

```

1766: }
0x08000368 B001 ADD sp,sp,#0x04
0x0800036A 4770 BX lr
1815: {
0x0800036C B082 SUB sp,sp,#0x08
0x0800036E F88D0007 STRB r0,[sp,#0x07]
0x08000372 9100 STR r1,[sp,#0x00]
1816: if ((int32_t)(IRQn) >= 0)
1817: {
0x08000374 F99D0007 LDRSB r0,[sp,#0x07]

```

main.c
startup_stm32f411xe.s
core_cm4.h
system_stm32f4xx.c

```

1750 }
1751 }
1752
1753
1754 /**
1755 \brief Set Pending Interrupt
1756 \details Sets the pending bit of a device specific interrupt
1757 \param [in] IRQn Device specific interrupt number
1758 \note IRQn must not be negative.
1759 */
1760 _STATIC_INLINE void __NVIC_SetPendingIRQ(IRQn_Type IRQn)
1761 {
1762 if ((int32_t)(IRQn) >= 0)
1763 {
1764 NVIC->ISPR[(((uint32_t)IRQn) >> 5UL)] = (uint32_t)1;
1765 }
1766
1767

```

Registers

Register	Value
R0	0x00000006
R1	0xE000E100
R2	0x00000000
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000644
R14 (LR)	0x08000423
R15 (PC)	0x0800034C
xPSR	0x21000000

Banked
System
Internal

Mode
Privilege
Stack
States
Sec

Thread
Privileged
MSP
3508
0.00035080

FPU

Disassembly

```

0x08000348 D40E BMI 0x08000368
0x0800034A E7FF B 0x0800034C
1764: NVIC->ISPR[(((uint32_t)IRQn) >> 5UL)] = (
0x0800034C F99D1003 LDRSB r1,[sp,#0x03]
0x08000350 F001021F AND r2,r1,#0x1F
0x08000354 2001 MOVS r0,#0x01
0x08000356 4090 LSLS r0,r0,r2
0x08000358 094A LSRS r2,r1,#5
0x0800035A F24E2100 MOVW r1,#0xE200
0x0800035E F2CE0100 MOVT r1,#0xE000

```

main.c
startup_stm32f411xe.s
core_cm4.h
system_stm32f4xx.c

```

1748 {
1749 return(0U);
1750 }
1751 }
1752
1753
1754 /**
1755 \brief Set Pending Interrupt
1756 \details Sets the pending bit of a device specific interrupt
1757 \param [in] IRQn Device specific interrupt number
1758 \note IRQn must not be negative.
1759 */
1760 _STATIC_INLINE void __NVIC_SetPendingIRQ(IRQn_Type IRQn)
1761 {
1762 if ((int32_t)(IRQn) >= 0)
1763 {
1764 NVIC->ISPR[(((uint32_t)IRQn) >> 5UL)] = 1;
1765 }
1766
1767
1768
1769 /**

```

23	EXTI Line1 interrupt	EXTI1	1	0	0	0 = 0s0
24	EXTI Line2 interrupt	EXTI2	0	0	0	0 = 0s0

Interrupt Control & State

SCB->ICSR: 0x00000816
VECTACTIVE: 0x16

☒ RETTOBASE
☐ ISRPREEMPT
VECTPENDING: 0x00
☐ ISR_PENDING

Diagram illustrating the structure of the VECTPENDING register (bits 31 to 0):

- Bits 31 to 21: VECTPENDING
 - Bit 31: ISRPENDING
 - Bit 30: Reserved for Debug
 - Bit 29: Reserved
 - Bit 28: PENDSTCLR
 - Bit 27: PENDSTSET
 - Bit 26: PENDSVCLR
 - Bit 25: PENDSVSET
 - Bit 24: Reserved
 - Bit 23: NMIPENDSET
 - Bit 22: Reserved
 - Bit 21: Reserved
- Bits 11 to 8: VECTACTIVE
- Bits 7 to 0: Reserved

R14 (LR)	0xFFFFFFFFE9
----------	--------------

LR[31:4] = 0xFFFFFFFFE

= 0 (handler return mode)

= 0(main stack)

$$LR[0] = 1$$

CCR[9]의 값이 0이다. 즉 스택이 4바이트 즉 워드 단위로 정렬이 된다는 의미이다.

(e) 디버거를 reset시키고 (RSR 버튼 클릭) Run 버튼을 클릭하시오. 그러면 25번 breakpoint에서 실행이 일단 멈출 것이다. ((c)에서의 SP 값이 같음을 일단 확인하시오.) 이 때 CONTROL register의 값을 0x00으로 변경한 다음, Run 버튼을 클릭하시오. 그러면 5번 breakpoint에서 실행이 일단 멈출 것이다. 이때 SP 값은 무엇인가? 그리고 Stack에 저장된 내용들을 설명하시오. (5점) 그리고 LR에 나타난 EXC_RETURN 값의 의미를 설명하시오. (2점)

Register	Value
Core	
R0	0x00000006
R1	0xE000E100
R2	0x00000000
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000648
R14 (LR)	0x0800041D
R15 (PC)	0x0800041E
xPSR	0x41000000
Banked	
System	
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x04
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	131994315
Sec	13,19943150
FPU	


```

0x08000418 F7FFF78 BL.W
0x0800041C 9802 LDR
25: NVIC_SetPendingIF
26:
0x0800041E F7FFF8D BL.W
27: while(1){
0x08000422 E7FF B
0x08000424 E7FE B
0x08000426 EEF10A10 VMRS
0x0800042A F64F71FF MOVW

```



```

main.c startup_stm32f411xe.s
1 #include "stm32f4xx.h"
2 //21800471 YooJunho
3 void EXTI0_IRQHandler()
4
5 void EXTI0_IRQHandler()
6 volatile int32_t
7
8 // meaningless
9 for(i=10 ; i>=0 ;
10 )
11 }
12
13
14 int main(void)
15 {
16
17 NVIC_SetPriority(
18
19 NVIC_SetPriority
20 NVIC_SetPriority
21
22 NVIC_EnableIRQ(6)
23 NVIC_EnableIRQ(7)
24
25 NVIC_SetPendingIF
26
27 while(1){
28 }
29
30

```

SP = 0x20000648 (C에서와 같음)

Register	Value
Core	
R0	0x00000040
R1	0xE000E200
R2	0x00000000
R3	0x20000260
R4	0x00000000
R5	0x20000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x08000450
R11	0x00000000
R12	0x20000040
R13 (SP)	0x20000620
R14 (LR)	0xFFFFFFFF9
R15 (PC)	0x080002D8
MPSR	0x41000016
Banked	
System	
BASEPRI	0x00
PRIMASK	0
FAULTMASK	0
CONTROL	0x00
Internal	
Mode	Handler
Privilege	Privileged
Stack	MSP
States	131994346
Sec	13.19943460
FPU	

```

0x080002D6 0
5: void
6:
7:
8:
0x080002D8 1
0x080002DA 2
9:
0x080002DC 3
<
main.c
1 #inc
2 //21
3 void
4
5 void
6
7
8
9
10
11 }
12
13
14 int
15 {
16

```

SP의 값이 0x20000620이 되었음

Memory 1

0x20000620

0x20000620:	40	00	00	00
0x20000624:	00	E2	00	E0
0x20000628:	00	00	00	00
0x2000062C:	60	02	00	20
0x20000630:	40	00	00	20
0x20000634:	23	04	00	08
0x20000638:	66	03	00	08
0x2000063C:	00	02	00	41

Call Stack + Locals | Memory 1

R0, R1, R2, R3, R12값이 낮은 주소값 으로부터 큰 주소값으로 저장되어 있으며

(f) (d)와 (e)의 결과가 다른 이유를 설명하시오. (5점)

CONTROL[0] :	CONTROL[1]	CONTROL[2]
0 = privileged thread mode	0 = 디폴트 스택(MSP)	0 = 부동 소수점 관련 명령어가 사용되지 않았다.
1 unprivileged(user) thread mode	1 = 프로세스 스택(PSP)	1 = 부동소수점 관련 명령어가 사용되었다.

CONTROL register의 값을 0x04에서 0x00으로 바꿨기 때문에 부동 소수점 관련 명령어가 사용되지 않았다.