한국산학기술학회논문지
Vol. 12, No. 4 pp. 1947-1953, 2011

ARM 프로세서를 기반으로 한 OSEK 운영체제의 태스크 전환 및 인터럽트 핸들링 메커니즘 구현

임성락[1*], 권오용[2]
[1]호서대학교 컴퓨터공학부, [2]호서대학교대학원 메카트로닉스학과

An Implementation of Task Switching and Interrupt Handling Mechanisms of OSEK Operating System based on ARM Processor

Seong-Rak Rim[1*] and O-Yong Kwon[2]
[1]**Division of Computer Engineering, Hoseo University**
[2]**Division of Mechatronics Engineering, Hoseo University**

요 약 OSEK/VDX는 자동차 ECU를 위한 산업계 표준을 제시하고자 구성된 공동 프로젝트이며 OSEK OS는 OSEK/VDX에서 제안한 사양을 준수하는 실시간 운영체제이다. 본 논문에서는 ARM 프로세서를 기반으로 한 OSEK OS의 태스크 전환 및 인터럽트 핸들링 메커니즘 구현을 제시한다. OSEK OS의 요구사항과 ARM 프로세서의 특성을 고려하여 태스크 전환 및 인터럽트 핸들링 메커니즘을 설계하였다. 제시한 메커니즘의 타당성을 검토하기 위하여 ARM 프로세서가 탑재된 실험용 임베디드 보드에서 기능적 정확성을 확인하고 태스크 전환과 인터럽트 핸들링에 소요되는 시간을 측정하였다.

Abstract OSEK/VDX is a joint project aiming at an industry standard for ECUs in vehicles and OSEK OS is a real-time operating system that meets OSEK/VDX specifications. In this paper, we suggest an implementation of task switching and interrupt handling mechanisms of OSEK operating system based on ARM processors. Considering the requirements of OSEK OS and characteristics of ARM processor, we have designed task switching and interrupt handling mechanisms. For evaluating the validation of the suggested mechanisms, we have checked the functional correctness on an experimental embedded board with ARM processor and calculated the time of task switching and interrupt handling.

Key Words : OSEK, OSEK OS, Task Switching, Interrupt Handling

1. 서 론

OSEK/VDX[1]는 1990년대 중반 유럽 자동차 생산자들(BMW, Daimler-Benz 등)과 부품 업체들(Bosch, Siemens등)이 서로 연합하여 자동차 전자 제어 장치(ECU:Electronic Control Unit)를 위한 산업계 표준을 제시하고자 구성된 공동 프로젝트이다[2]. OSEK 운영체제[3]는 OSEK/VDX에서 제안한 사양을 준수하는 실시간 운영체제로서 기본적으로 태스크 단위의 다중처리, 태스크간의 동기화를 위한 자원 및 이벤트 관리, 인터럽트, 알람, 카운터 그리고 오류처리 기능을 제공한다[4][5].

본 논문에서는 ARM 프로세서[6]를 기반으로 한 OSEK 운영체제의 태스크 전환 및 인터럽트 핸들링 메커니즘 구현을 제시한다. 이를 위하여 OSEK 운영체제의 요구사항과 ARM 프로세서의 특성을 고려하여 태스크 전환 및 인터럽트 핸들링 메커니즘을 설계 구현한다. 제시한 메커니즘의 기능적 정확성을 확인하기 위하여 시스템 시작 및 테스트에 필요한 API 함수들을 구현하여

*교신저자 : 임성락(srrim@hoseo.edu)
접수일 11년 02월 09일 수정일 (1차 11년 03월 23일, 2차 11년 04월 06일) 게재확정일 11년 04월 07일

ARM 프로세서가 탑재된 실험용 임베디드 보드[7]에서 테스트한다.

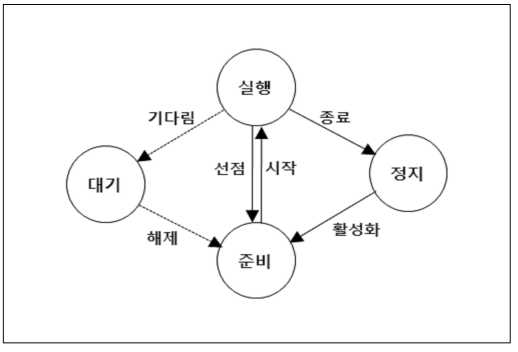
본 논문의 주된 목적은 태스크 전환 및 인터럽트 핸들링 메커니즘을 개념적으로 설명한 자료는 많이 있지만, OSEK 운영체제 개발과정에서 이를 구현할 경우 직면하게 되는 어려움에 도움을 주기 위한 것이다. 국내외적으로 OSEK 운영체제 개발이 진행되고 있기 때문에 태스크 전환 및 인터럽트 핸들링 메커니즘을 본 논문에서 처음 구현한 것이라 할 수 없다. 하지만 이를 주제로 한 기존 연구 자료를 찾을 수 없어 태스크 전환시간 및 인터럽트 핸들링 소요시간에 대한 비교 평가가 어려워 제시한 기법에 대한 소요시간만을 정량적으로 계산하였다.

2. OSEK 운영체제의 요구사항

태스크 전환 및 인터럽트 핸들링 메커니즘을 설계하기 위하여 OSEK 운영체제의 다음과 같은 요구사항들을 고려한다.

2.1 태스크 스케줄링

OSEK OS에서는 두 가지 종류(기본, 확장)의 태스크를 지원한다. 기본 태스크와 확장 태스크의 차이점은 오직 확장 태스크에만 어떤 사건발생을 기다리는 대기상태가 존재하는 점이며 이들의 상태 전환도는 그림 1과 같다.



[그림 1] 태스크 상태 전환도

태스크는 시스템이 초기화될 때 정적으로 생성되며 그 속성(자동 활성화 여부)에 따라 준비 혹은 정지 상태로 등록된다. 초기에 부여된 태스크의 우선순위는 PCP(Priority Ceiling Protocol)[2] 경우를 제외하고 변경되지 않는다.

태스크 스케줄링 정책은 태스크의 속성(선점성 여부)에 따라 다르게 적용된다. 선점형 태스크일 경우엔 선점 스케줄링 정책으로, 비선점형일 경우엔 비선점 스케줄링 정책으로 처리된다.

이론적으로 스케줄링이 수행되어야 경우는 다음과 같으며 각 경우에 해당하는 API 함수는 표 1과 같다.

① 실행 중인 태스크가 종료되어 정지 상태로 전환될 경우
② 실행 중인 확장 태스크가 어떤 사건발생 기다림으로 대기상태로 전환될 때
③ 비선점형 태스크가 우선순위가 더 높은 태스크에게 자발적으로 선점되어 준비상태로 전환될 때
④ 선점형 태스크가 우선순위가 더 높은 태스크에 의해 강제적으로 선점되어 준비상태로 전환될 때

[표 1] 스케줄링 관련 API 함수

경우	API 함수
①	TerminateTask(), ChainTask()
②	WaitEvent()
③	Schedule()
④	ActivateTask(), SetEvent(), ChainTask(), ReleaseResource(), (Return from ISR2)

2.2 인터럽트 핸들링

OSEK OS에서는 ISR(ISR : Interrupt Service Routine)을 Category 1(ISR1)과 Category 2(ISR2)로 구별한다. ISR1과 ISR2의 차이점은 오직 ISR2 에서만 API 함수 호출이 허용되는 점이다. 따라서 ISR2가 수행되는 동안 API 함수 호출에 의해 새로운 태스크가 활성화 될 수 있기 때문에 ISR2로부터 복귀될 때 태스크 스케줄링이 요구된다(표 1의 경우-4).

OSEK 운영체제에서는 세 가지 종류의 처리 수준을 정의하고 있다. 태스크가 처리되는 태스크 수준과 인터럽트가 처리되는 인터럽트 수준 그리고 스케줄링을 위한 논리적 수준이다. 인터럽트 수준은 태스크 수준보다 우선순위가 높기 때문에 태스크 수준으로 돌아가기 전에 모든 인터럽트 처리가 완료되어야 한다. ISR마다 서로 다른 고유의 우선순위가 정적으로 부여되며 인터럽트 지연시간을 최소화하기 위하여 인터럽트의 중첩을 허용한다.

기본적으로 ISR 내부에서는 태스크 전환이 발생하지 않지만 ISR로부터 복귀되는 위치를 결정하기 위하여 다음과 같은 요소들을 고려한다.

① 처리수준(태스크 혹은 인터럽트)
② 발생한 인터럽트 종류(ISR1 혹은 ISR2)
③ 태스크 속성(선점형 혹은 비선점형)

3. 설계 및 구현

3.1 태스크 전환 메커니즘

스케줄링이 수행되는 시점들 중에서 실행 태스크가 정상적으로 종료될 경우와 자동실행으로 태스크가 실행될 경우, 태스크 전환을 위한 문맥저장 단계가 필요 없다. 이 단계를 생략하여 태스크 전환에 소요되는 시간을 최소화하도록 문맥저장과 문맥복구를 구분하여 설계 한다.

3.1.1 태스크 문맥저장

태스크 문맥저장 과정은 그림 2와 같다.

```
context_save:
   ;정상 종료 검사
   ldr     r0, =current_task
   ldr     r1, [r0]
   cmp     r1, #0
   beq     context_restore

;레지스터 값 저장
   stmfd sp!, {r0-r12,r14}
;sp 값을 TCB[SP]에 저장
   str       sp, [r1, #SP]
;레지스터 복구 루틴주소를 TCB[PC]에 저장
   ldr       r2, =reg_restore
   str       r2, [r1, #PC]

reg_restore: ;레지스터 복구 루틴
   ldmfd sp!, {r0-r12, pc}
```

[그림 2] 태스크 문맥저장

그림 2에서 이 단계의 생략 여부를 검사한다. 필요할 경우, CPU의 레지스터 값을 스택영역에 저장하고 스택 레지스터의 값을 TCB에 저장한다. 마지막으로 스택영역에 저장된 CPU의 레지스터 값을 복구하기 위한 루틴의 시작주소를 TCB에 저장한다.

3.1.2 태스크 문맥복구

태스크 문맥복구 과정은 그림 3과 같다.

```
context_restore:
;TCB[SP]로부터 SP 값 복구
    ldr     sp, [r0, #SP]
;TCB{PC]로부터 레지스터 복구 루틴주소 복구
    ldr     r1, [r0, #PC]
    stmfd sp!, {r1}

;인터럽트 활성화/SPSR에 저장 후 전환
    msr     r1, CPSR
    bic     r1, r1, #IRQ_DISABLE
    msr     SPSR, r1
    ldmfd sp!, {pc}^
```

[그림 3] 태스크 문맥복구

그림 3에서 TCB로부터 스택 레지스터 값을 복구한 후, TCB로부터 스택영역에 저장된 CPU의 레지스터 값을 복구하기 위한 루틴의 시작주소를 PC 레지스터에 설정함으로써 새로운 태스크로 전환이 이루어진다.

이 과정에서 유의해야 할 점은 새로운 태스크로 전환하기 전에 인터럽트를 활성화시켜주어야 하는데 인터럽트 활성화와 선택된 태스크의 실행 위치로 분기가 분리되어서는 안 된다. 인터럽트 활성화와 선택된 태스크의 실행 위치로의 분기가 분리될 경우 그 사이에서 ISR2의 인터럽트가 발생하게 되면 ISR 내부에서 API 함수를 호출 할 수 있기 때문에 정상적인 태스크 전환을 보장 할 수 없다.

본 논문에서는 그림 3과 같이ARM 프로세서의 특성을 이용하여 현재 CPU의 상태 값 중 인터럽트 부분을 활성화하여 SPSR에 보관한 후, 하나의 명령어(ldmfd sp!, {pc}^)를 이용하여 인터럽트 활성화 및 선택된 태스크의 실행 위치로 분기토록 하였다.

3.2 인터럽트 핸들링

ARM 프로세서에서는 인터럽트가 발생하면 다음과 같은 과정이 하드웨어적으로 수행된다[8].

① 프로세서 모드가 IRQ 모드로 변환된다.
② 이전 모드의 CPSR이 IRQ_SPSR에 저장된다.
③ PC(R15)가 IRQ 모드의 IRQ_LR에 저장된다.
④ 인터럽트가 비활성화 된다.
⑤ IRQ 벡터 테이블에 선언된 위치로 분기된다.

위와 같은 ARM 프로세서의 하드웨어적인 특성을 바탕으로 인터럽트 핸들링 루틴(IHR: Interrupt Handling Routine)을 IHR-전처리와 IHR-후처리로 구분하여 설계한다.

3.2.1 IHR-전처리

IHR-전처리는 인터럽트에 의한 하드웨어적인 표준과정이 수행된 후부터 ISR이 호출될 때까지의 루틴으로써 그림 4와 같다.

그림 4에서 범용 레지스터(R0-R12)와 링크 레지스터(LR)의 값을 인터럽트 당한 태스크의 스택에 저장하고, 하드웨어적으로 IRQ_SPSR에 저장된 이전 모드의 SPSR과 IRQ_LR에 저장된 복귀주소(PC)를 R0, R1에 각각 저장한 후, 다시 R0, R1을 인터럽트 당한 태스크의 스택에 저장함으로써 CPU 문맥 저장을 처리한다. CPU 문맥을 저장한 후 현재 처리수준을 스택에 저장하고, 현재 처리수준을 인터럽트 수준으로 전환한다. 마지막으로 인터럽트를 활성화하고, ISR을 호출한다.

```
;CPU 문맥저장
msr     CPSR_c,#(SVC|IRQ_DISABLE|FIRQ_DISABLE)
stmfd  sp!, {r0-r12, r14}
msr     CPSR_c,#(IRQ|IRQ_DISABLE|FIRQ_DISABLE)
msr     r0, SPSR
mov     r1, lr
msr     CPSR_c,#(SVC|IRQ_DISABLE|FIRQ_DISABLE)
stmfd  sp!, {r0-r1}
;처리수준 저장 및 인터럽트 수준 설정
ldr     r2, =_level
ldr     r1, [r2]
stmfd  sp!, {r1}
ldr     r1, #INT_LEVEL
str     r1, [r2]
;인터럽트 활성화 및 ISR 호출
msr     r0, cpsr
bic     r0, r0, #(IRQ_DISABLE|FIRQ_DISABLE)
msr     cpsr, r0
ldr     r0, =_ISR_ROUTINE
mov     lr, pc
bx      r0
```

[그림 4] IHR-전처리

3.2.2 IHR-후처리

IHR-후처리는 ISR이 수행된 후부터 인터럽트 당한 태스크로 되돌아갈 때까지의 루틴으로써 그림 5와 같다.

```
;인터럽트 비활성화
msr     r0, cpsr
orr     r0, r0, #(IRQ_DISABLE|FIRQ_DISABLE)
msr     cpsr, r0

ldr     r2, =_level ;처리수준 비교
ldmfd  sp!, {r0}
str     r0, {r2]

cmp     r0, #INT_LEVEL ;인터럽트 수준일 경우
bne     int_ret

mov     r0, =_ISR_TYPE ;ISR 종류 비교

cmp     r0, #ISR1          ;ISR1일 경우
beq     int_ret

ldr     r1, =_current_task ;ISR2일 경우
ldr     r1, [r1]

ldr     r0, [r1,#TASK_TYPE] ;태스크 속성 비교

cmp     r0, #NON_PREEMPTIVE ;비선점일 경우
beq     int_ret

;선점일 경우 스케줄러 호출
bl      select_higher_priority_task
ldr     r1, =_current_task
ldr     r1, [r1]

cmp     r0, r1
beq     int_ret ;자기 자신일 경우
b       int_save_context ;새로운 태스크일 경우
```

[그림 5] IHR-후처리

그림 5에서 우선적으로 인터럽트를 비활성화하고, 처리수준을 복구한다. 처리수준이 인터럽트 수준 일 경우 중첩 인터럽트이므로 이전 인터럽트를 처리하기 위해 CPU 문맥을 복구한다. 반면, 태스크 수준일 경우 인터럽트 타입을 비교한다.

인터럽트 타입이 ISR1일 경우, 이전 태스크의 수행을 재개하기 위하여 CPU 문맥을 복구한다. ISR2일 경우 태스크 속성을 검사하여 비선점형이면 이전 태스크 수행을 재개하기 위하여 CPU문맥을 복구하고, 선점형이면 스케줄러를 호출한다. 선택된 태스크가 이전에 실행중인 태스크일 경우 CPU 문맥을 복구하고, 새로운 태스크 일 경우 태스크 전환 메커니즘을 수행한다.

IHR-후처리의 마지막 과정(“int_ret:”)은 그림 6과 같이 CPU 문맥을 복구한다.

```
int_ret:
ldmfd  sp!, {r0-r1} ;SPSR과 IRQ_LR을
msr     CPSR_c,#(IRQ|IRQ_DISABLE|FIRQ_DISABLE)
stmfd  sp!, {r0-r1} ;IRQ 스택에 저장

msr     CPSR_c,#(SVC|IRQ_DISABLE|FIRQ_DISABLE)
ldmfd  sp!, {r0-r12, r14} ;범용 레지스터 복구

;IRQ_CPSR과 ISR_LR복구
msr     CPSR_c,#(IRQ|IRQ_DISABLE|FIRQ_DISABLE)
ldmfd  sp!, {r14}
msr     SPSR, r14
ldmfd  sp!, {pc}^
```

[그림 6] CPU 문맥 복구

그림 6에서 태스크 스택에 저장되어 있는 SPSR과 복귀주소 IRQ_LR을 R0, R1에 각각 저장한 후, 다시 R0, R1을 IRQ 스택에 저장한다. 다음으로 태스크 스택에 있는 CPU의 범용 레지스터(R0-R12)와 링크 레지스터(LR)의 값을 복구한다. 마지막으로 IRQ_SPSR와 IRQ_LR을 IRQ스택으로부터 복구한다.

4. 테스트 및 평가

제시한 메커니즘의 기능적 정확성을 확인하기 위하여 ARM 프로세서(48MHz)가 탑재된 실험용 임베디드 보드에서 다음과 같은 과정으로 테스트하고 태스크 전환 및 인터럽트 핸들링에 소요되는 시간을 계산하여 제시한 메커니즘의 타당성을 평가한다.

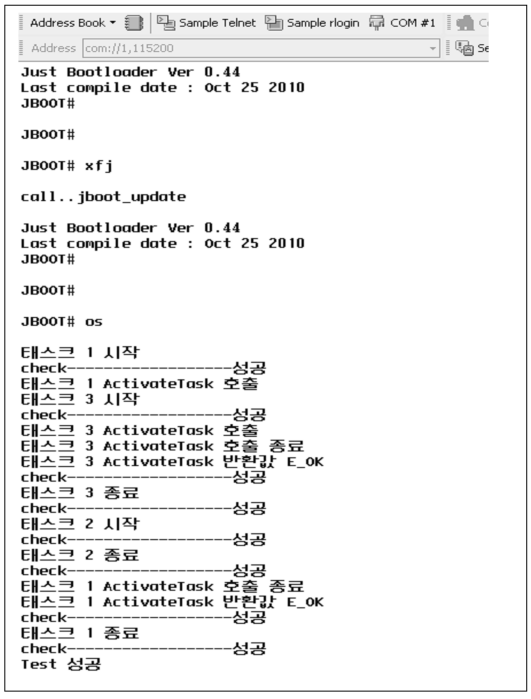
4.1 태스크 전환 메커니즘

태스크 전환 메커니즘을 테스트하기 위하여 표 2와 같은 태스크(T1, T2, T3)를 사용한다.

[표 2] 태스크 속성 및 API 호출

태스크	자동	선점여부	API 호출
T1	Y	선점	ActivateTask(T3) TerminateTask()
T2	N	선점	TerminateTask()
T3	N	선점	ActivateTask(T2) TerminateTask()

태스크 번호가 클수록 우선순위가 높으며 선점에 의한 태스크 전환이 이루어지는 과정을 확인하기 위하여 우선순위가 가장 낮은 태스크(T1)만을 자동 활성화로 설정하였으며 테스트 결과는 그림 7과 같다.



[그림 7] 태스크 전환 메커니즘 테스트 결과

그림 7에서 T1만이 자동 활성화로 설정되어 있으므로 초기에 실행되는 T1은 ActivateTask(T3)를 호출하여 T3를 활성화시키면 T3의 우선순위가 T1보다 높기 때문에 선점이 일어난다. T3는 ActivateTask(T2)를 호출하여 T2를 활성화시키지만 T3의 우선순위가 T2보다 높기 때문에 T3가 종료된 후에 T2가 실행되고, 마지막으로 T3에 의해 선점되었던 T1이 재개되어 종료됨을 확인할 수 있었다.

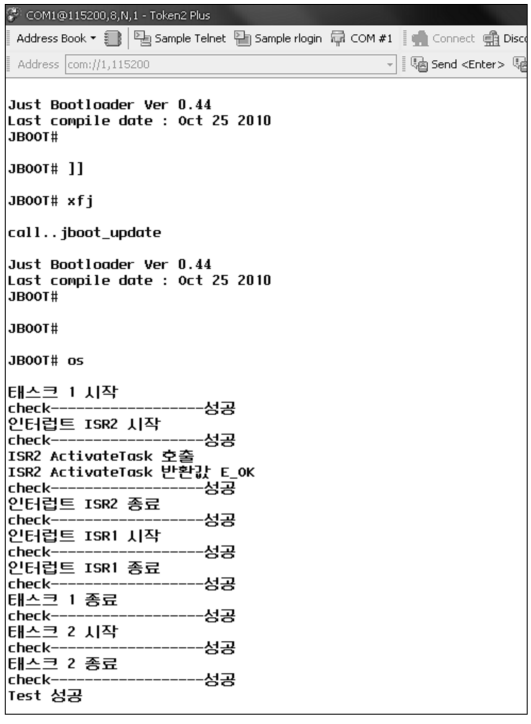
4.2 인터럽트 핸들링 메커니즘

인터럽트 핸들링 메커니즘을 테스트하기 위해 표 3과 같은 태스크(T1, T2)와 ISR(ISR1, ISR2)를 사용한다.

[표 3] 태스크와 ISR의 속성 및 기능

태스크	자동	선점여부	기 능
T1	Y	비선점	ISR2 인터럽트 발생 TerminateTask()
T2	N	비선점	TerminateTask()
ISR1	-	-	-
ISR2	-	-	ISR1 인터럽트 발생 ActivateTask(T2)

ISR1은 실험용 임베디드 보드의 버튼-1 인터럽트에 대한 ISR 이며, ISR2는 버튼-2 인터럽트에 대한 ISR로서 내부에서 API 호출이 가능하다. T2의 우선순위가 T1보다 높고 ISR2의 우선순위가 ISR1 보다 높다.



[그림 8] 인터럽트 핸들링 메커니즘 테스트 결과

그림 8에서 T1만이 자동 활성화되므로 초기에 T1이

실행된다. T1이 실행되는 동안에 버튼-2를 눌러 ISR2 인터럽트를 발생시키면 인터럽트 핸들러에 의해 ISR2가 실행된다. ISR2가 실행되는 동안에 버튼-1를 눌러 중첩 인터럽트를 발생시키면 ISR2의 우선순위가 더 높기 때문에 ISR2가 계속 실행된다. 한편, ISR2가 ActivateTask(T2)를 호출하여 T2를 활성화 시키고 종료되면 ISR1이 실행된다. ISR1이 종료되면 T2의 우선순위가 T1 보다 높지만 비선점형 태스크이기 때문에 T1의 실행이 재개된다. T1이 종료되면 마지막으로 ISR2에서 활성화 되었던 T2가 실행됨을 확인할 수 있다.

4.3 소요 시간

실시간 운영체제에서 매우 중요한 요소인 태스크 전환과 인터럽트 핸들링에 소요되는 시간을 ARM7 명령어 실행 사이클[8]을 이용하여 48MHz에서 계산하였다.

4.3.1 태스크 전환시간

태스크 전환시간은 실행중인 태스크의 문맥을 저장하고 선택된 태스크의 문맥을 복구하는데 소요되는 시간으로 정의한다[9].

[표 4] 태스크 문맥 저장 및 복구 시간

구 분	실행 사이클(수)	소요시간(μs)
문맥저장	22	0.45
문맥복구	43	0.89

표 4에서 태스크 문맥 저장시간의 비중은 태스크 전환 시간의 34%(22/(22+43))로서, 태스크 문맥저장을 생략할 경우 48MHz에서 0.45μs정도의 소요시간 단축을 기대할 수 있다.

4.3.2 인터럽트 핸들링 시간

인터럽트 핸들링 시간은 인터럽트 응답 및 복귀 시간으로 구성된다.

인터럽트 응답시간은 인터럽트를 받는 순간부터 인터럽트를 처리하는 사용자 코드가 시작되는 순간 사이의 시간으로서[9], 본 논문에서 IHR-전처리 소요시간에 해당된다. 인터럽트 복귀 시간은 인터럽트에 의해 중단됐던 코드가 다시 수행되기까지 걸리는 시간으로서[9], 본 논문에서 IHR-후처리 소요시간에 해당된다.

인터럽트 복귀시간은 처리수준, 인터럽트 타입, 태스크 속성에 의존적이다. 이러한 의존성을 검토하기 위하여 다음과 같은 네 가지 경우로 구분하여 계산하였다.

① 인터럽트 처리수준일 경우
② 태스크 처리수준이면서 ISR!일 경우
③ 비선점형 태스크 처리수준이면서 ISR2일 경우
④ 선점형 태스크 처리수준이면서 ISR2일 경우

[표 5] 인터럽트 핸들링 시간

구 분		실행 사이클(수)	소요시간(μs)
응답시간		44	0.91
복귀 시간	경우-①	49	1.02
	경우-②	51	1.06
	경우-③	61	1.27
	경우-④	71	1.47

표 5에서 인터럽트 응답시간은 처리수준, 인터럽트 타입, 태스크 속성에 무관하게 일정하다. 그러나 인터럽트 복귀시간은 처리수준, 인터럽트 타입, 태스크 속성 그리고 스케줄링 결과에 따라 차이가 있음을 확인할 수 있다.

5. 결 론

OSEK 운영체제는 태스크(기본 혹은 확장) 단위로 다중처리를 지원하고 있으며 다중처리를 위한 스케줄링 정책과 태스크 전환 메커니즘은 사용자에 의해 부여된 태스크 속성에 의존한다. 또한 인터럽트 핸들링 메커니즘은 OSEK 운영체제에서 지원하는 ISR 타입과 인터럽트가 발생한 인터럽트가 발생한 처리수준, 태스크 속성에 의존한다.

본 논문에서는 이러한 의존성을 고려하여 OSEK 운영체제의 태스크 관리에서 필수적으로 요구되는 태스크 전환 및 인터럽트 핸들링 메커니즘을 ARM 프로세서 기반으로 설계 및 구현하였다.

마지막으로 ARM 프로세서가 탑재된 실험용 임베디드 보드에서 구현한 메커니즘의 기능적 정확성을 확인하고, 태스크 전환 및 인터럽트 핸들링 소요시간을 측정하였다.

참 고 문 헌

[1] OSEK/VDK, http://portal.osek-vdx.org
[2] 임진택, 금한홍, 박지용, 홍성수, “동적 메모리 사용

감소를 위한 OSEK OS 커널 구현 메커니즘", 한국자동차공학학회논문집, 17권3호, pp.127-141, 2009.
[3] OSEK/VDK, "Operating System Specification 2.2.3", 2월, 2005.
[4] 신민석, 이우택, 선우명호, 한석영, "OSEK/VDX 표준과 CAN 프로토콜을 사용한 차체 네트웍 시스템 개발", 한국자동차공학회논문집, 제 10권 4호, pp.175-180, 2002.
[5] 서영빈, 김상철, 마평수, 최태영, "ROSEK: OSEK 기반 자동차용 운영체제", 정보처리학회지, 제15 권 5호, 9월, 2008.
[6] 윤덕용, "ARM7TDMI AT91SAM7S256으로 시작하기", Ohm사, 2007.
[7] EZ-AT 임베디드 보드, http://www.falinux.com
[8] Andrew N. Sloss, Dominic Symes, Chris Wright, "ARM System Developer's Guide", Elsevier Inc., 2004.
[9] 성원호, "MicroC/OS-II 실시간 커널", 에이콘, 2005.

임 성 락(Seong-Rak Rim) **[정회원]**

- 1983년 2월 : 서울대학교 공과대학원 컴퓨터공학과 (공학석사)
- 1992년 8월 : 서울대학교 공과대학원 컴퓨터공학과 (공학박사)
- 1983년 3월 ~ 1990년 2월 : (주) 금성반도체연구소 선임연구원
- 1993년 3월 ~ 현재 : 호서대학교 컴퓨터공학과 교수

<관심분야>
임베디드 시스템, OSEK/VDX 운영체제

권 오 용(O-Yong Kwon) **[준회원]**

- 2009년 2월 : 호서대학교 컴퓨터공학과 (공학사)
- 2011년 2월 : 호서대학교 메카트로닉스 공학과 (공학석사)

<관심분야>
임베디드 시스템, OSEK/VDX 운영체제