논문 2006-43SD-11-4

Silicon RTOS을 위한 하드웨어 구성에 관한 연구

(A Study on the Hardware Architecture for Silicon RTOS)

송 문 빈*, 정 연 모**

(Moonvin Song and Yunmo Chung)

요 약

RTOS(Real Time Operating System)의 빠른 수행 능력은 임베디드 시스템의 성능을 결정하는 중요한 요소이다. 멀티미디어 및 통신 환경이 발달하면서 더 높은 처리 성능의 시스템을 요구하고 있다. 그러나 마이크로프로세서를 기반으로 하는 소프트웨어로 이루어진 RTOS의 처리 능력을 획기적으로 개선하는 데는 어려운 점이 많다. 따라서 본 논문에서는 RTOS의 성능을 개선하기 위하여 소프트웨어로 이루어진 일부 기능을 하드웨어로 구현하기 위한 Silicon RTOS의 구성에 대하여 연구하였으며 실제로 uC/OS-II의 해당 부분을 하드웨어로 구현하였으며 성능을 비교 분석하였다.

Abstract

The fast processing ability of an RTOS (Real Time Operatiing System) is one of important factors in determining the performance of embedded systems. With the development of multimedia and telecommunication technology, the higher level of performance environments is required. Moreover there is some difficulty in improving the performance of an RTOS which is based on a microprocessor. In this paper, we propose a hardware architecture to implement some functions of uC/OS-II as a target RTOS for the purpose of its performance improvement. The proposed architecture for uC/OS-II is implemented and analyzed with the performance comparison.

Keywords: RTOS, Silicon, uC/OS-II, Hardware

Ⅰ. Silicon RTOS의 개념

RTOS는 시스템을 효율적으로 관리하기 위하여 마이크로프로세서와 사용자 프로그램 사이에서 멀티태스킹, 스케줄링, 문맥전환과 같은 여러 가지 서비스와 이벤트, 메일 박스, 큐와 같은 태스크 간의 통신을 제공한다[1].

RTOS의 성능을 개선하기 위하여 소프트웨어적인 알고리즘을 개선하거나 프로세서의 처리 능력을 높이는 방법을 사용하였다. 그러나 반도체 설계 기술의 발전으로 하드웨어와 소프트웨어를 통합으로 설계할 수 있는 SoC(System On a Chip) 설계 기술이 발전하고 합성 가능한 마이크로프로세서가 사용 가능해지면서 소프트웨어로 이루어진 RTOS의 일부 기능을 하드웨어로 구현한 Silicon RTOS의 구현이 가능해 지고 있다. Silicon RTOS는 기존의 RTOS와 비교해 시스템의 크기를 줄일 수 있고 처리속도의 향상과 같은 성능을 개선할 수 있다[2].

Silicon RTOS는 FPGA의 자원이 충분히 크다는 가정 하에 다중 레지스터 구조, 태스크 컨트롤 블록(TCB), 이벤트 컨트롤 블록(ECB), 클럭 틱 ISR, ISR, 스케줄러, 타이머, 딜레이 관련 함수, 메일 박스, 큐 등을 하드웨어로 구현한다.

Ⅱ. 기존 연구

RTOS의 성능을 개선하기 위해 많은 선진국들이 다양한 연구를 하고 있다. Silicon RTOS의 경우 아직 상용성을 갖춘 결과는 없다. Silicon RTOS는 주로 하드웨어로 구성하는 방법에 대한 연구가 주로 이루어지고 있다.

* 학생회원, ** 정회원, 경희대학교 전자공학과
Dept. of Electronic Eng., Kyung Hee University
접수일자: 2006년6월26일, 수정완료일: 2006년10월30일

상용화된 많은 수의 RTOS를 가지고 있는 미국의 경우에는 Silicon RTOS에 대한 연구를 하기에 좋은 환경을 갖고 있다. 그리고 일본의 경우에는 일본 RTOS 시장의 상당 부분을 차지하고 있는 자국의 RTOS인 TRON(The Real-time Operating system Nucleus)을 하드웨어와 연동하는 연구를 활발히 진행하고 있다.

TRON을 하드웨어 화하는 연구에서는 시스템 호출(system call)과 스케줄링 기능을 하드웨어로 구현하고 [그림 1]과 같은 동작을 수행한다[2].

그림에서 소프트웨어 만으로만 구현된 RTOS와 같이 시스템 호출을 처리한다. 특히 임계 영역(critical section)에서는 인터럽트를 비활성화해야 하는 시간이 발생한다. 그러나 시스템 호출과 스케줄링 기능을 하드웨어로 구현하면 수행 시간을 줄일 수 있다. 이와 같이 하드웨어로 구현한 부분을 'uITRON', 'Silicon TRON', 또는 '하드웨어 커널' 이라고 부르고 [그림 2]와 같은 구조를 갖는다.

일반적인 RTOS의 경우는 그림의 좌측과 같이 CPU를 기반으로 RTOS가 존재하고 그 위에 응용 프로그램

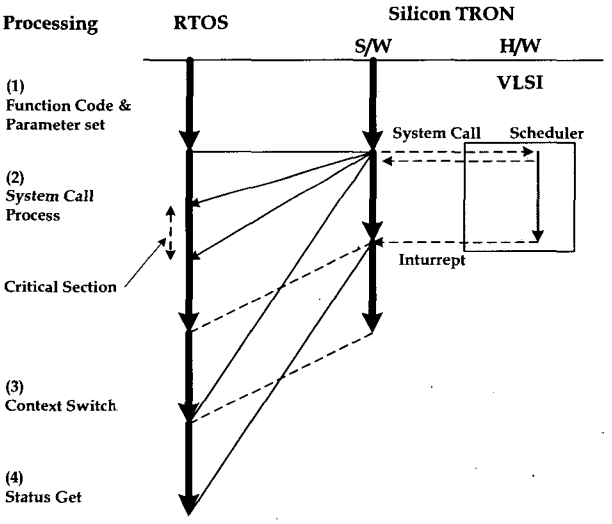

그림 1. 하드웨어로 구현한 시스템 호출 동작 순서
Fig. 1. Hardware-implemented system call sequence.

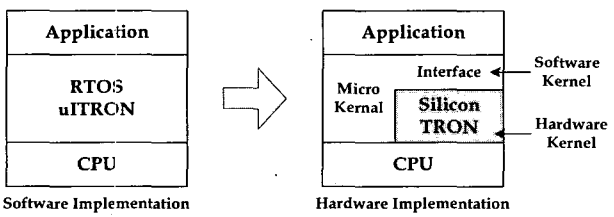

그림 2. Silicon TRON의 구조
Fig. 2. Silicon TRON architecture.

이 있다. 그러나 Silicon TRON의 경우에는 CPU를 기반으로 하드웨어와 소프트웨어로 나누어진 커널과 인터페이스가 존재하고 그 위에 응용 프로그램이 있는 구조이다.

III. Silicon RTOS의 구조

SoC 환경에서는 RTOS의 성능을 높이는 방법으로는 세 가지를 고려할 수 있다. 첫 번째는 RTOS의 소프트웨어적인 구조를 변경하여 성능을 개선하는 것이다[3]. 두 번째는 프로세서의 구조 자체를 RTOS의 특성에 맞게 바꾸는 것이다[4]. 마지막으로 RTOS에서 시스템의 성능을 좌우하는 특정 기능 중에 하드웨어로 구현 가능한 부분을 하드웨어로 구현하는 것이다[5, 6]. 이런 것들은 PLD(Programmable Logic Device)의 크기 및 속도가 충분히 높아지고 마이크로프로세서가 합성 가능한 RTL 형태의 IP로 제공되기 때문에 가능해지고 있다.

RTOS의 성능을 평가하는 주요한 요소로는 응답 시간과, 태스크들의 동작 순서를 관리하는 스케줄링 기능이다. 그리고 RTOS의 성능을 좌우하는 요소로는 문맥전환, 인터럽트 응답시간, 및 임계 영역 등이 있다.

3.1 문맥전환

문맥전환은 RTOS에서 멀티태스킹을 제공하기 위해서 반드시 필요한 동작이다. 그러나 문맥전환은 시스템의 부가적인 부하요소로 작용하여 성능을 저하시키는 중요한 요인이다.

문맥전환은 세 가지 상황에서 발생한다. 첫 번째는 현재 CPU를 사용하는 태스크보다 더 높은 우선순위를 갖는 태스크가 활성화 된 경우이다. 두 번째는 가장 높은 우선순위를 가지고 현재 CPU를 사용하던 태스크가 지연 함수를 호출하거나 이벤트를 기다리기 위해 대기 상태로 바뀌는 경우이다. 이때 대기 중이던 태스크 중 우선순위가 높은 태스크로 문맥전환을 한다. 마지막으로 인터럽트가 발생하여 ISR을 처리할 때 이다.

문맥전환은 레지스터를 메모리에 저장하고 복귀하는 동작으로 이루어져 있으므로 소요되는 시간은 레지스터의 수에 비례한다.

마이크로프로세서는 내부 구조상 [그림 3]와 같이 한 순간에는 하나의 태스크만이 활성화 된다.

여러 사용자가 책상의 사용권을 갖기 위해 대기하고 있다. 그러나 작업 공간에는 하나의 작업을 수행할 수 있는 책상이 하나이고, 그 방을 사용할 수 있는 열쇠

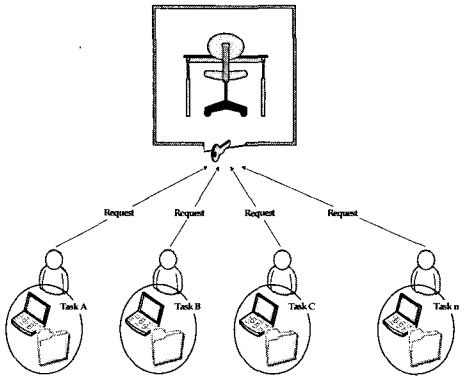

그림 3. 단일 레지스터의 문맥전환 개념
Fig. 3. Context switch on a single register.

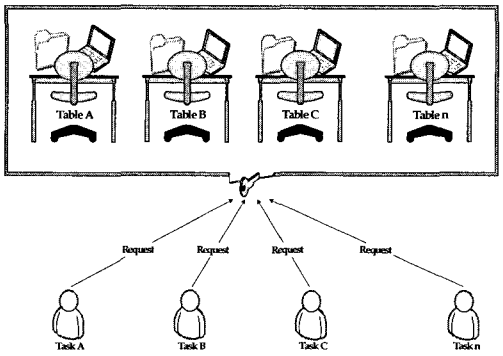

그림 4. 다중 레지스터의 문맥전환 개념
Fig. 4. Context switch on multiple registers

또한 하나 밖에 없는 상황이다. 이 책상을 사용하려는 여러 개의 사용자는 작업할 파일이나 컴퓨터를 자기가 보관하고 있어야 한다. 예를 들어 태스크 C가 책상을 사용하고 있다가 태스크 A가 책상을 사용할 상황이 되면 태스크 C는 뒤에 다시 책상을 사용할 때를 대비해 현재 작업하던 내용을 자신의 파일과 컴퓨터에 저장을 하고 파일과 컴퓨터를 들고 방을 나와야 한다. 그러면 태스크 A는 자신의 파일과 컴퓨터를 들고 방으로 들어가 작업을 계속하기위해 다시 책상위에 기존에 작업하던 파일과 컴퓨터를 설치해야 한다. 이런 일은 책상을 사용하는 사용자가 바뀔 때 마다 반복해야 한다. 이런 단점을 보완하기 위해 방의 구조를 [그림 4]과 같이 변경할 수 있다.

방에는 사용자 숫자만큼의 책상을 설치하고 각 사용자는 본인만이 사용할 수 있는 책상을 배정해 준다. 물론 방을 사용할 수 있는 사용자는 한 명뿐이다. 이런 경우 사용자를 변경해야할 상황이 발생하면 단지 기존의 사용자가 방에서 나오고 새로운 사용자가 방으로 들어가기만 하면 된다. 현재의 문맥전환 방식인 [그림 3]와 과 비교해 많은 시간을 줄일 수 있다.

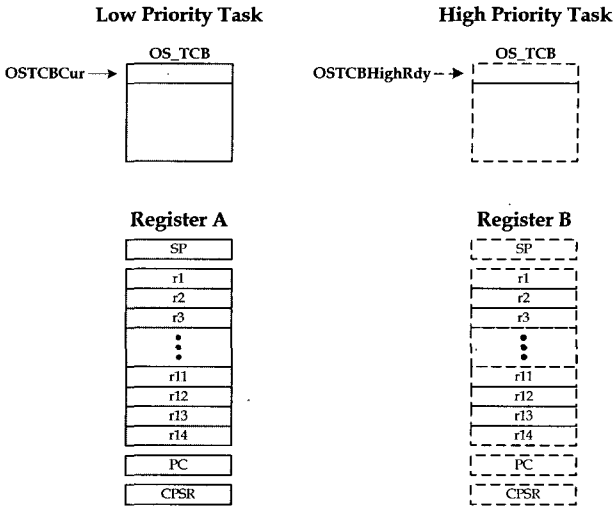

그림 5. 다중 레지스터 구조에서 문맥전환 수행 전
Fig. 5. Before context switch with multiple registers.

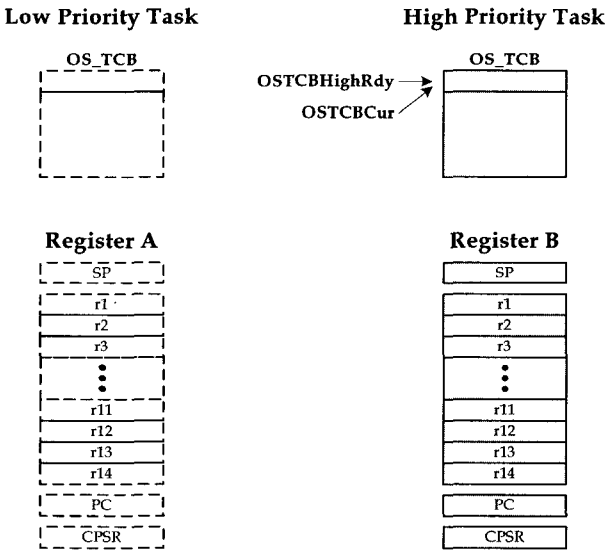

그림 6. 다중 레지스터 구조에서 문맥전환 수행 후
Fig. 6. After context switch multiple registers.

다중 레지스터 구조에서 문맥전환을 수행하기 직전의 상태는 [그림 5]와 같고, 수행한 결과는 [그림 6]와 같다.

현재 CPU를 점유하고 있는 태스크를 가리키는 포인터인 OSTCBCUr와 우선순위가 제일 높은 태스크를 가리키는 포인터인 OSTCBHighRdy가 서로 다르므로 문맥전환이 필요하다.

다중 레지스터 구조에서는 문맥전환을 [그림 6]와 같이 OSTCBCur을 우선순위가 제일 높을 태스크로 변경하기만 하면 된다.

3.2 인터럽트

임계 영역(critical section)은 현재 태스크가 공유 변수나 특정한 커널 서비스를 사용하는 도중 다른 태스크에게 선점돼서는 안 되는 특정한 영역이다.

인터럽트는 임계 영역의 처음과 끝 부분에 비활성화와 활성화 시켜야 한다. 이러한 동작을 구현하기 위해 [표 1]에서와 같이 기존의 RTOS에서는 인터럽트 컨트롤러 레지스터를 제어하여 구현한다.

그러나 하드웨어로 구현하기 위해서는 인터럽트 신호선 자체의 연결을 외부에서 하드웨어 적으로 제어한다. 그러면 인터럽트를 제어하기 위해서는 단지 특정 어드레스를 지정하기만 하면 된다.

이러한 기능을 하드웨어로 구현하기 위해서는 우측과 같이 특정 어드레스를 설정하는 것만으로 구현할 수 있다. 프로그램에서 인터럽트를 제어하기 위해 특정 어드레스를 지정하면 마이크로프로세서로 전달되는 인터럽트 신호선 자체의 연결을 외부에서 하드웨어 적으로 제어한다. 그러면 제어 시간을 반으로 주일 수 있다. 하드웨어 인터럽트 제어 방식의 구조는 [그림 7]과 같다.

외부의 인터럽트 소스들은 마이크로프로세서 내부의 인터럽트 컨트롤러를 통해서 IRQ와 FIQ 두 선으로 전달한다. 메모리상에는 인터럽트를 활성화하거나 비활성화하기위한 주소가 설정되어 있다. 외부의 인터럽트 소스들은 FPGA를 통하여 마이크로프로세서의 인터럽트 컨트롤러로 전달한다. FPGA에서는 단순히 메모리상의 주소와 데이터 버스 상태만을 보고 스위치 역할만 수행한다.

표 1. 하드웨어 인터럽트를 위한 코드 변경
Table 1. Code modification for H/W interrupt.

기존 RTOS	제안하는 방법

EXPORT ARMDisableInt
ARMDisableInt
MRS r12, CPSR
ORR r12, r12, #NoInt
MSR CPSR, r12
MOV pc, lr
EXPORT ARMEnableInt
ARMEnableInt
MRS r12, CPSR
BIC r12, r12, #NoInt
MSR CPSR, r12
MOV pc, lr
	EXPORT ARMDisableInt
	ARMDisableInt
	MOV r0, #OffIntAdd
	STR r1, [r0]
	
	EXPORT ARMEnableInt
	ARMEnableInt
	MOV r0, #OnIntAdd
	STR r1, [r0]

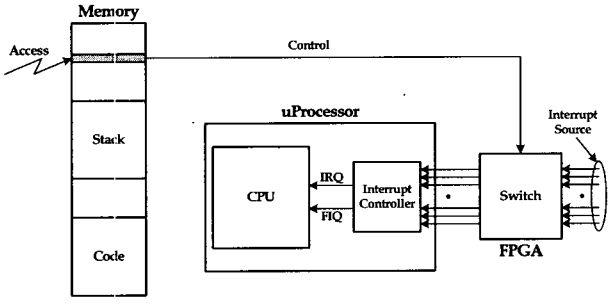

그림 7. 하드웨어 인터럽트 제어 블록 구조
Fig. 7. Hardware interrupt control block architecture.

3.3 태스크 컨트롤 블록

태스크 컨트롤 블록은 태스크의 상태를 저장하기 위한 데이터 구조체로서 [그림 8]과 같은 구조를 가지고 있다.

태스크가 우선순위가 높은 태스크에 선점되면 레지스터 상태를 해당 태스크 컨트롤 블록에 저장하고, 대기 중이던 태스크가 활성화되면 해당 태스크 컨트롤 블록의 레지스터 상태를 CPU로 복사한다.

태스크 컨트롤 블록은 해당 태스크가 마지막으로 사용한 스택의 위치를 가리키는 포인터, 사용하는 스택 영역의 끝을 가리키는 포인터, 그리고 스택의 크기를 나타내는 필드를 가지고 있다. 또한 옵션을 저장하는 필드, 커널 내부에서 생성된 태스크 컨트롤 블록을 연결하기 위한 필드, 이벤트 컨트롤 블록을 가리키는 포인터, 태스크로 보낸 메시지를 가리키는 포인터, 이벤트 플래그 노드를 가리키는 포인터, 지연이나 이벤트를 기다릴 때 사용하는 필드, 태스크의 상태를 나타내는 필드, 태스크의 우선순위를 나타내는 필드, 각종 연산을 수행하기 위해 사용하는 필드, 그리고 태스크가 삭제 요청을 받았는지를 나타내는 필드를 가지고 있다.

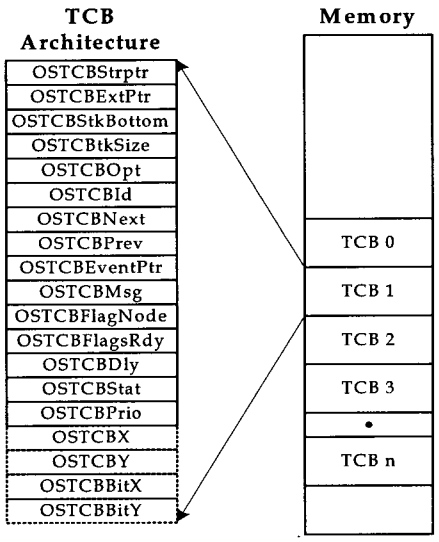

그림 8. 태스크 컨트롤 블록 구조
Fig. 8. TCB architecture.

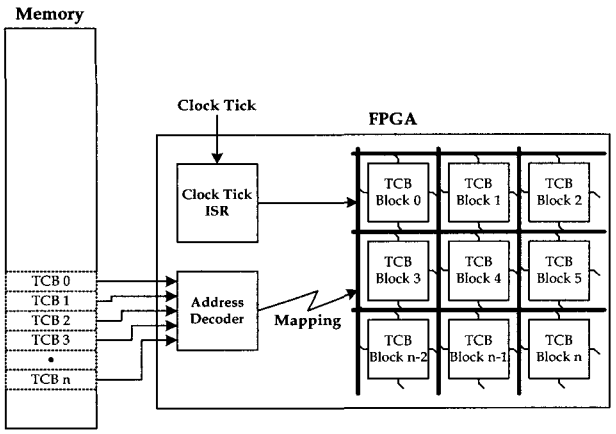

그림 9. 하드웨어 태스크 컨트롤 블록 구조
Fig. 9. Hardware task control block architecture.

태스크 컨트롤 블록을 Silicon RTOS에서 하드웨어로 구현하면 [그림 9]과 같은 구조를 갖는다.

하드웨어로 구현한 태스크 컨트롤 블록은 FPGA에서 어드레스 디코더를 통하여 메모리와 연결한다. 메모리에는 실제로 태스크 컨트롤 블록의 동작을 수행하는 코드는 없고 단지 각종 정보들을 프로그램과 연결하기 위한 주소만 존재한다.

예를 들어 TCB3의 우선순위를 알기 위해서 응용프로그램이 태스크 컨트롤 블록의 OSTCBId를 읽으면 FPGA 내의 어드레스 디코더에서 해당하는 내용을 TCB Block 3의 내용과 연결시킨다. 또한 주기적으로 발생하는 클럭 틱 ISR과도 상호 연계되어 있다.

3.4 제어 블록 구조

제어 블록은 명령어 디코더, 보조 제어 회로, 그리고 원격 제어 장치로 [그림 10]과 같이 구성한다. Decode 블록은 명령어의 일부와 사이클 카운터를 이용하여 다음 사이클에서 수정해야 할 데이터 흐름의 동작이 무엇인지 결정한다. 하단 부분의 각종 제어 블록에서는 디코더에서 발생한 정보를 이용해 명령어의 일부와 CPU의 상태에 따라 각종 제어 블록을 제어 한다. Load/Store Multiple과 코프로세서처럼 명령어의 수행이 일반 명령어와는 다르게 여러 cycle에 걸쳐 연산을 수행하는 경우에는 그 일을 다 마칠 때 까지 디코더는 동작을 멈춘다. 또한 하드웨어로 구현된 클럭 틱 ISR을 포함한 ISR이나 문맥전환을 위한 동작을 수행하는 경우에도 디코더는 동작을 멈추고 있어야한다.

H_Kernel을 구성하는 register select control, clock tick control, scheduler control, 그리고 delay control 블록이 추가 된다.

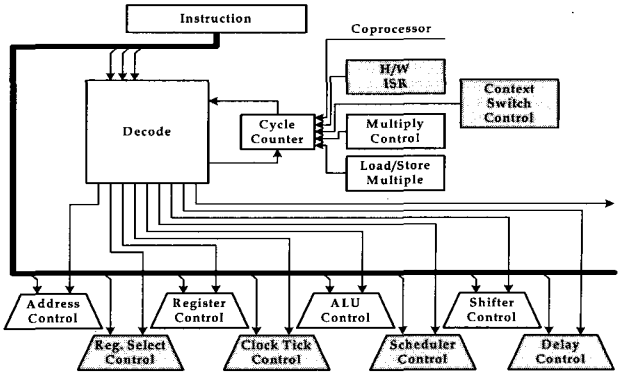

그림 10. 제어 구조
Fig. 10. Controller architecture.

3.5 H_Kernel의 메모리 매핑 방법

Silicon RTOS에서 하드웨어로 구현한 부분은 CPU와 상호 연동 시켜야 한다. CPU의 경우 외부와의 인터페이스 방법은 버스를 사용하는 방법밖에 없다.

하드웨어로 구현한 각 블록들을 CPU와 메모리 매핑 방법으로 연결한다. 기존의 RTOS의 구성 및 동작 예는 [그림 11]과 같다.

RTOS의 각 구성 요소들은 메모리상에 자료 구조 형태로 존재한다. 또한 각 태스크들은 고유한 스택 영역을 가지고 있다. RTOS에서 제공하는 특정 서비스를 수행하려면 메모리상의 코드를 실행해야 하므로 코드 길이만큼의 수행 시간이 필요하다. [그림 11]에서와 같이 예를 들어 클럭 틱 ISR을 수행하는 경우 메모리상의 클럭 틱 ISR 수행 함수를 수행한다. 그리고 태스크 A를 수행하고 지연(delay) 함수를 호출하면 메모리상의 지연 함수를 수행하는 시간이 필요하다. 하드웨어로 구현

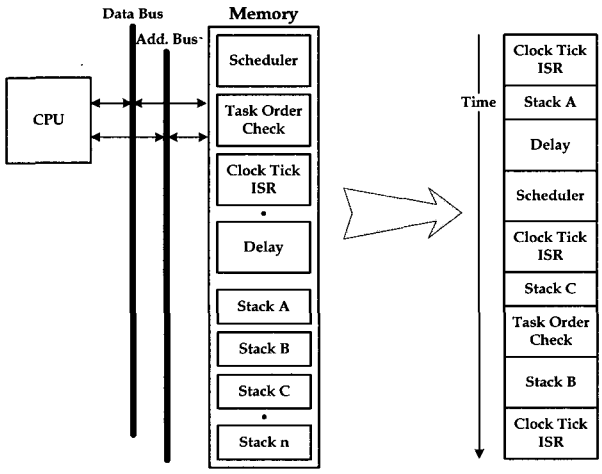

그림 11. RTOS 구성
Fig. 11. RTOS organization.

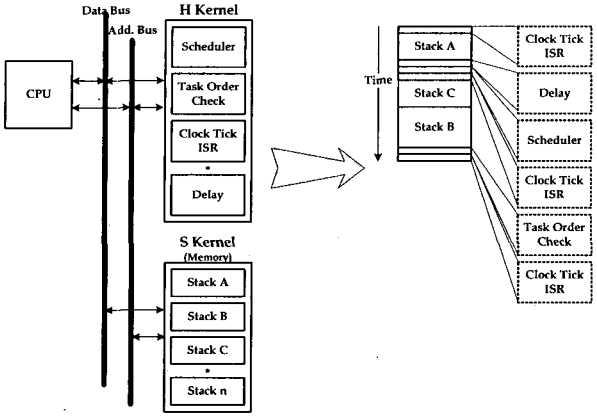

그림 12. Silicon RTOS의 하드웨어 구성
Fig. 12. Hardware organization on Silicon RTOS.

한 부분의 구성 및 동작은 [그림 12]와 같다.

Silicon RTOS에서 하드웨어로 구현한 특정 컴포넌트들을 H_Kernel이라고 한다. 나머지 기능은 메모리상에서 S_Kernel을 구성한다. H_Kernel은 커널 서비스 중에서 스케줄러, 우선순위 검출기, 클럭 틱 ISR, ISR, 메일 박스, 큐, 이벤트, 태스크 컨트롤 블록, 그리고 이벤트 컨트롤 블록 등이 있다. S_Kernel 부분은 나머지 자료 구조들을 포함하고 있다.

수행은 H_Kernel 부분을 수행해야 할 경우 실시간으로 처리가 되어 있기 때문에 단지 S_Kernel에서 사용하기만 하면 된다. 따라서 전체적인 수행 시간이 [그림 11]의 RTOS에서 소프트웨어만으로 처리한 것과 비교해 대폭 준 것을 알 수 있다.

3.6 Silicon RTOS 하드웨어 구조

Silicon RTOS는 [그림 13]과 같이 메모리에 일정한 데이터를 저장하는 자료구조 형식을 갖는 부분만 남겨두고 나머지 기능은 하드웨어로 구현한 H_Kernel에서 처리한다.

Silicon RTOS는 소프트웨어로 이루어진 S_Kernel, 하드웨어로 이루어진 H_Kernel, 그리고 메모리는 데이터 및 어드레스 버스로 서로 연결되어 있다.

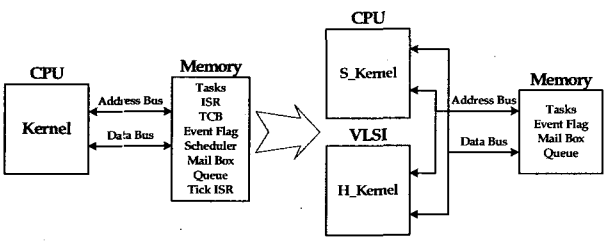

그림 13. Silicon RTOS의 구성
Fig. 13. Silicon RTOS architecture.

Ⅳ. 수행 결과 및 결론

Silicon RTOS는 Altera의 Excalibur FPGA에 uC/OS-II RTOS를 사용하였으며 클럭 틱 ISR의 수행 시간, 구현 상태에 따른 코드의 크기와 게이트 사이즈를 비교한 결과는 [그림 14, 15]와 같다.

RTOS 에서는 태스크 들의 상태에 따라 클럭 틱 ISR의 처리 시간이 다르다. 그러나 [그림 14] 에서와 같이 Silicon RTOS는 항상 일정한 시간을 나타내고 있으며 RTOS와 비교해 2 배 이상 빠른 것을 알 수 있다.

[그림 15]와 같이 Silicon RTOS에서 하드웨어 부분의 게이트 크기는 ISR을 포함한 클럭 틱 ISR에 비해 문맥 전환을 수행하는 부분이 크다. 또한 소프트웨어만으로 구성된 RTOS는 하드웨어적인 게이트가 없다.

Silicon RTOS는 멀티미디어 및 통신 환경이 발달하면서 요구하는 성능을 쉽게 만족 시킬 수 있으며 시스

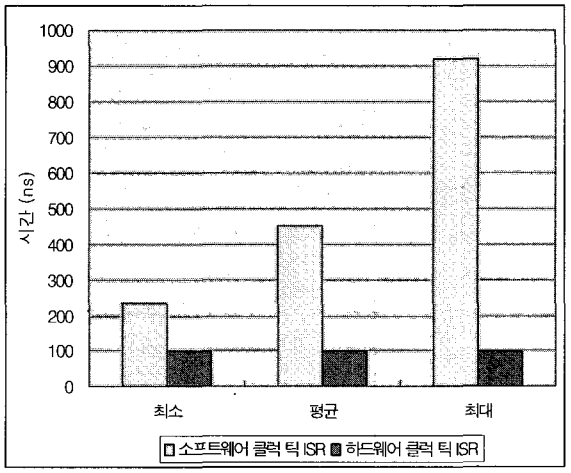

그림 14. 클럭 틱 ISR 성능 비교
Fig. 14. Comparisons between H/W and S/W of clcok tick ISRR.

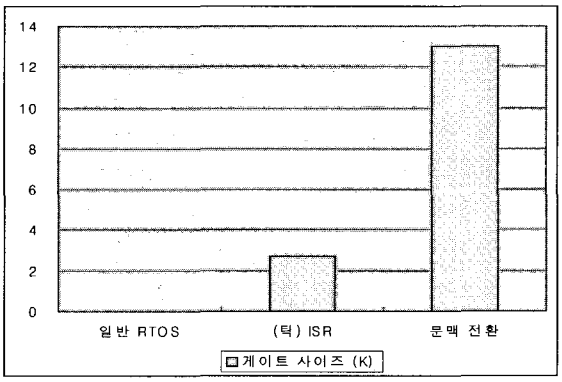

그림 15. 게이트 크기 비교
Fig. 15. Gate size Comparison.

템의 처리 능력을 높이고, 응답 시간을 확정적으로 개선할 수 있다. 또한 구현 단가를 줄일 수 있다.

소프트웨어와 하드웨어를 시스템에 적용하는 과정을 살펴보면, RTOS를 대상으로 하는 마이크로프로세서에 맞게 커널의 각종 부분을 C/C++ 그리고 어셈블리 코드를 수정하는 포팅 작업을 수행한다. 그리고 C/C++로 응용 프로그램을 작성하여 실행 파일을 생성 시킨 후 시스템의 메모리에 저장하여 실행 시킨다. 수정할 상황이 발생하면 C/C++로 구성된 코드를 수정한 후 동일한 작업을 반복한다. PLD를 기반으로 하는 하드웨어는 HDL로 회로를 설계한 후 합성과정을 거친 결과를 PLD에 저장하여 실행 시킨다. 수정할 상황이 발생하면 HDL 코드를 수정한 후 동일한 작업을 반복한다. 즉, 시스템 관점에서 살펴보면 소프트웨어를 수정하는 것과 PLD를 기반으로 하는 하드웨어를 수정하는 일의 작업 과정이 동일하다.

Silicon RTOS는 소프트웨어만으로 구현한 RTOS에서 시간 관리, 태스크 관리, 스케줄 관리, 그리고 각종 커널 서비스를 하드웨어로 구현한 시스템으로 비용, 처리 속도, 시스템 성능 면에서 많은 이점이 있다.

참 고 문 헌

[1] J. J. Labrosse, *MicroC/OS-II, 2th Edition : the real-time kernal*, Pub. Group West, 2006.

[2] T. Nakano, A. Utrama, M. Itabashi, A. Shiomi, and M. Imai, " Hardware Implementation of a Real-Time Operating System," *IEEE proc. of TRON '95*, pp34-42, 1995.

[3] Zhaohui Wu, Hong Li, Zhigang Gao, Jie Sun, Jiang Li, "An Improved Method of Task Context Switching in OSEK Operating System," *AINA 2006*, pp217-222, Apr. 2006.

[4] C. Boke, M. Gotz, T. Heimfath, D. Kebbe, J. Rammig, S. Rips, "Re-Configurable Real-Time Operating Systems and Their Applications," *Proc. of the Eighth International Workshop on Object-Oriented Real-Time Dependable System*, 2003.

[5] J. Lee, V. Mooney, "A Framework for Automatic Generation of Configuration Files for a Custom Hardware/Software RTOS," *Proc. of the International Conference on Engineering of Reconfigurable System and Algorithms (ERSA '2)*, pp31-37, June 2002.

[6] V. Mooney, M. Blough, "A Hardware-Software Real-Time Operating System Framework for SoCs," *IEEE Design & Test of Computers 2002*, pp44-51, Dec. 2002.

[7] A. Jantscb, *Modeling Embedded Systems and SoCs*, Morgan Kaufmann Publishers, 2004.

[8] A. Siberschatz, P. Galvin, G. Gagne, *Operating System Principles (7th Edition)*, John Wiley & Sons. Inc., 2005.

[9] Homepage of the Altera, http://www.altera.com.

[10] Homepage of the ARM, http://www.arm.com.

저 자 소 개

송 문 빈(학생회원)

1998년 한밭대학교 전자공학과 학사.

2002년 경희대학교 전자공학과 공학석사.

2006년 현재 경희대학교 전자공학과 박사과정.

<주관심분야 : SoC 설계, 임베디드 시스템, RTOS>

정 연 모(정회원)

1980년 경북대학교 학사.

1982년 KAIST 공학석사.

1987년 경제기획원 전산처리관.

1992년 미시간주립대학교 공학박사.

2006년 현재 경희대학교 전자정보대학 교수

<주관심분야 : SoC 설계, 임베디드 시스템, RTOS>