2005년도 대한전기학회 하계학술대회 논문집 2005. 7. 18 - 20

RTOS와 SOPC를 이용한 유연한 실시간 처리 임베디드시스템

안상민*, 최우창**, 공정식***, 이보희****, 김진걸*****, 허욱열******
*인하대학교 전기공학과(ahns@huro.inha.ac.kr)
**인하대학교 전자공학과(uchang2@huro.inha.ac.kr)
***인하대학교 자동화공학과(tempus@dreamwiz.com)
****세명대학교 전기공학과(bhlee@semyung.ac.kr)
*****인하대학교 전기공학과(john@inha.ac.kr)
******인하대학교 전기공학과(uyhuh@inha.ac.kr)

Flexible Real Time Embedded System Using RTOS and SOPC

Sang-Min Ahn*, Woo-Chang Choi**, Jung-Shik Kong***, Bo-Hee Lee****, Jin-Geol Kim*****, and Uk-Youl Huh******
*School of Electrical Engineering. Inha University, e-mail : ahns@huro.inha.ac.kr
**School of Electronic Engineering. Inha University, e-mail : uchang2@huro.inha.ac.kr
***Industrial Automation Engineering. Inha University, e-mail : tempus@dreamwiz.com
****School of Electrical Engineering. Semyung University, e-mail : bhlee@semyung.ac.kr
*****School of Electrical Engineering. Inha University, e-mail : john@inha.ac.kr
******School of Electrical Engineering. Inha University, e-mail : uyhuh@inha.ac.kr

Abstract - This paper deals with the design of real-time embedded system using RTOS (real-time operating system) and SOPC (system on a programmable chip). It is, in general, known that RTOS has the problem of time delay caused by the multiple tasks and task management approach. Since the increase in time delay in real-time embedded system makes the overall system have the poor performance or the critical behavior of instability and unreliability, the method employed RTOS and SOPC is proposed to attack the above problems. The proposed system is implemented on the RC-motor controller and is verified by real experiment.

1. 서 론

현재 임베디드 시스템 분야는 반도체기술과 소프트웨어기술의 빠른 발전으로 인하여 시스템의 처리능력이 급격히 향상되고 있다. 이런 추세에 발맞추어 임베디드 시스템은 적용 분야를 점점 넓혀가고 있다. 산업이나 생활에 사용하는 장치들의 시스템이 점점 복잡해지고 빠른 처리능력을 요구하게 되면서 임베디드 시스템의 실시간 처리 능력과 효율적인 시스템 관리 능력이 필요하게 되었다. 그리하여 현재에는 임베디드 시스템에 RTOS가 적용되는 것이 일반화 되는 추세이다. 그리고 이 시스템의 플랫폼은 이미 산업용이나 생활용 장비에 적용되어 사용되고 있다. 게다가 반도체 PLD(programmable logic device)의 성능과 가격경쟁력이 향상되면서 기존의 ASIC(application specific integrated circuit)기반에서만 구현되던 SoC(system on a chip)의 범위가 PLD기반의 SOPC의 범위로 확장되었다. 이로 인하여 시스템은 어플리케이션의 로직부분을 매우 유연하고 재구성이 가능한 하드웨어 레벨에서 구현이 가능하게 되었다[1-4].

일반적으로 사용되는 CPU플랫폼에서의 RTOS는 소프트웨어 태스크의 우선순위나 멀티태스킹 방식에 따라 태스크 실행에 있어서 지연시간이나 교착상태가 발생한다. 이것은 RTOS를 내장한 임베디드 시스템의 실시간성을 떨어트리는 요인으로 작용한다.

본 논문에서는 FPGA를 기반으로 한 SOPC(system on a programmable chip)를 이용하여 하드웨어 태스크 설계 기법을 RTOS에 적용한 새로운 실시간 임베디드 시스템을 설계하고 이 시스템을 실험을 통해 검증하였다.

2. 시스템 설계

2.1 구현 환경

시스템을 구성하기 위하여 RTOS, CPU, 개발 툴을 다음과 같이 선정하였다. 표 1은 시스템을 개발한 위한 환경을 보여 준다.

표 1. 시스템 개발 환경

종류	내용
RTOS	uCOS-II
CPU	Altera Nios
Hardware Design Tool	Altera Quartus
Hardware Design Language	Verilog HDL
CPU Design Tool	SOPC Builder
Software Compiler	GCC for Nios
Software Language	C and Assembly
FPGA Device	Altera Stratix EPF1S10F780

시스템을 개발하기 위한 SOPC 패키지는 FPGA 보드, Quartus, SOPC Builder 그리고 GCC Compiler로 이루어져 있다. 그리고 시스템에 RTOS를 내장하기 위하여 Nios에 포팅된 uCOS-II 소스 파일도 함께 사용하였다.

2.2 시스템 구조

시스템의 실시간성을 높이기 위하여 일부 소프트웨어 태스크를 하드웨어로 설계하는 방법을 선택하였다. 기존의 RTOS의 내부 인터페이스나 구조를 크게 변경하지 않으면서, 하드웨어 태스크를 쉽게 지원하기 위하여 다음과 같은 내용을 고려하였다.

1. RTOS입장에서 볼 때, 하드웨어 태스크와 소프트웨어 태스크의 액세스는 동일해야 한다.
2. 소프트웨어 태스크와 하드웨어 태스크 간의 메시지를 주고받을 수 있어야 한다.
3. 하드웨어 태스크는 소프트웨어 태스크와 유사한 동작 상태를 지원해야 한다.
4. 하드웨어 태스크에 대한 스케줄링은 따로 되어야하지만, 태스크 상태에 대한 설정은 소프트웨어 태스크에 의해 결정되어야 한다.

위의 내용을 만족하는 하드웨어 태스크 구조를 설계하기 위하여 하드웨어 태스크와 RTOS간의 인터페이스를 담당할 장치를 설계하였다. 그리고 각각의 태스크마다 독립적인 실행 주기를 만들기 위하여 별도로 타이머도 함께 포함 하였다.

2.2.1 하드웨어 태스크 설계

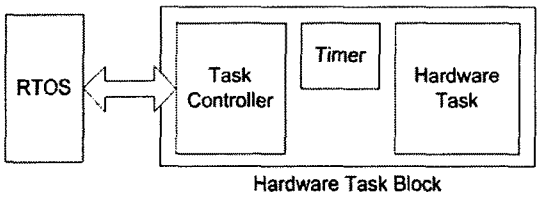

그림 1. 하드웨어 태스크 블록 구조

그림 1은 설계된 하드웨어 태스크의 구조를 보여준다. Hardware Task Block은 Verilog HDL을 이용하여 모듈로 설계되었다. 각각의 하드웨어 블록이 하는 역할은 다음과 같다.

Task Controller : RTOS와 Hardware Task간의 인터페이스 역할을 한다. 이 태스크는 *Memory map*에 매핑 되며, RTOS는 Memory address를 엑세스하여 Hardware Task를 엑세스한다.

Timer : Hardware Task의 Clock Tick을 발생한다. 주기는 Task Controller에 의해 결정된다.

Hardware Task : 유저가 설계할 Hardware Task

Task Controller는 하나의 하드웨어 태스크를 관리한다. RTOS는 Task Controller를 통하여 하드웨어 태스크의 Clock-tick과 상태를 결정한다. 하드웨어 태스크의 동작은 ON과 OFF의 두 가지 상태로 구성된다. 그리고 하나의 Task Controller는 4개의 메모리 어드레스를 가지며, 각 메모리 어드레스는 Task Controller 내부의 레지스터와 연결되어 있다. Nios CPU가 사용하는 메모리 엑세스 방식인 Avalon BUS Interface를 이용하여 RTOS는 Task Controller를 관리한다. Task Controller의 메모리 어드레스는 다음과 같은 역할을 한다.

표 2. 어드레스 할당

태스크 어드레스 : x	
x	Hardware Task 상태
	0 : OFF, 1 : ON
x+1	Timer 설정 Clock-tick = System-Clock / Data
x+2	다른 태스크들과 데이터 교환을 위한 메시지 박스
x+3	기타 기능을 위한 공간

2.2.2 RTOS 기능 추가

위에서 설계한 하드웨어 태스크를 지원하기 위하여 RTOS인 uCOS-II의 소스코드를 일부 추가하였다. 우선 RTOS가 하드웨어 태스크를 인식하도록 할 등록 구조를 만들고, 그 구조에 하드웨어 태스크를 등록하고 삭제하는 함수를 만들었다. 그림 2는 하드웨어 등록 테이블의 구조를 보여준다.

태스크 0	태스크 어드레스	태스크 상태
.	.	.
.	.	.
.	.	.
태스크 63	태스크 어드레스	태스크 상태

그림 2. 하드웨어 태스크 테이블 : OSHWTaskTbl

OSHWTaskTbl의 구조는 아래와 같다.

```
struct OSHWTask {
        INT08U TaskID;
        INT32U *TaskAddr;
        INT08U TaskStatus;
}
struct OSHWTask OSHWTaskTbl[64];
```

하드웨어 태스크 테이블에 하드웨어 태스크를 추가하기위하여 OSHWTaskCreat() 함수를 추가하였다. 이 함수는 하드웨어 태스크의 메모리 어드레스를 하드웨어 태스크 테이블의 태스크 어드레스영역에 기록하는 기능을 한다. 이 함수의 프로토타입은 아래와 같다.

```
void OSHWTaskCreat(INT08U ID, INT32U *Addr);
```

RTOS의 하드웨어 태스크 테이블에 등록된 하드웨어 태스크를 삭제하기 위하여 OSHWTaskDel()함수를 만들었다. 이 함수는 특정 하드웨어 태스크를 하드웨어 태스크 테이블에서 삭제하는 기능을 한다. 이 함수의 프로토타입은 다음과 같다.

```
void OSHWTaskDel(INT08U ID);
```

시스템의 하드웨어 태스크와 소프트웨어 태스크간의 메시지 전달을 위하여 메시지 전달 함수를 구현하였다. 여기서 전송되는 메시지 데이터의 크기는 1Byte로 설정하였다. 소프트웨어 태스크에서 하드웨어 태스크로 메시지를 보내기 위하여 OSHWMsgSend() 함수를 구현하였다. 이 함수의 프로토타입은 아래와 같다.

```
void OSHWMsgSend(INT08U ID, INT08U Msg);
```

시스템이 동작할 때, 소프트웨어 태스크가 하드웨어 태스크로부터 메시지 받는 기능을 지원하기 위하여 OSHWMsgRecv()함수를 구현하였다. 이 함수의 프로토타입은 아래와 같다.

```
INT08U OSHWMsgRecv(INT08U ID);
```

현재 RTOS에 등록된 하드웨어 태스크의 상태를 변경

하기 위하여 OSHWTaskStatus()함수를 구현하였다. 이 함수를 이용하여 하드웨어 태스크의 상태를 변경할 수 있다. 이 함수의 프로토타입은 아래와 같다.

void OSHWTaskStatus(INT08U ID, INT08U Status);

시스템에 구현된 하드웨어 태스크의 Clock-tick을 변경하기 위하여 OSHWTaskTick()함수를 구현하였다. 이 함수는 하드웨어 태스크의 Clock-tick을 실시간으로 변경이 가능하며 프로토타입은 아래와 같다.

void OSHWTaskTick(INT08U ID, INT16U Data);

2.2.3 시스템 초기화

위에서 추가된 내용을 시스템에 적용하기 위하여 기존 uCOS-II의 초기화 코드에 위의 내용을 초기화 하는 코드를 삽입하였다. 다음은 하드웨어 태스크의 초기화 과정을 보여준다. 이 과정은 OSStart()함수 내에서 실행된다.

1. 하드웨어 태스크 어드레스 초기화
2. 하드웨어 태스크 Clock-tick 설정
3. OSHWTaskTbl 구조 초기화 및 하드웨어 태스크 등록
4. 기본적인 OS 초기화

3. 실 험

이 시스템의 성능을 검증하기 위하여 RC-Servo 모터 제어기를 설계하였다. 성능 비교를 위하여 하나의 시스템은 소프트웨어 태스크로 RC-Servo 모터 제어기를 8개 구성하였고, 다른 하나의 시스템은 하드웨어 태스크로 RC-Servo 모터 제어기를 구성하였다.

RC-Servo 모터의 제어 신호는 PWM신호로 전체 주파수는 50Hz, 듀티 폭은 0.7ms ~ 2.3ms를 가진다. 그리고 RC-Servo 모터는 제어신호가 0.7ms일 때에는 -90도, 2.3ms일 때에는 +90도의 위치로 움직인다.

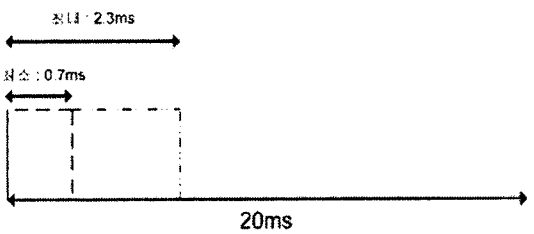

그림 3. RC-Servo 모터의 제어 신호

원래의 RTOS를 이용하여 8개의 서보모터 제어기를 구현하였다. RTOS의 Clock-tick은 1ms로 설정하였고, 하나의 태스크에 모터 하나를 제어 할 수 있도록 하였다. 오차를 확인하기 위하여 모터의 위치는 2.0ms, 주기는 50Hz로 모두 설정하였다. 그림 4는 기존의 RTOS에 구현된 모터 제어기에 의하여 출력되는 신호를 보여 준다. 그림 4에서 신호를 관찰하면 설정한 시간 이상의 신호가 지연되는 것을 확인 할 수 있었다. 이때 지연되는 시간은 시간에 대하여 변화하고 있었고, 그 크기는 약 0.5ms ~ 1ms 사이의 값을 나타내었다. 신호의 주기도 20ms에서 약 0.2ms 정도의 오차를 보여주었다.

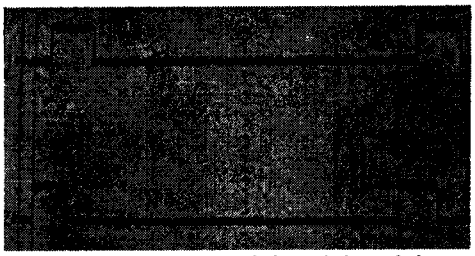

그림 4. 기존의 RTOS에서 구현된 모터신호

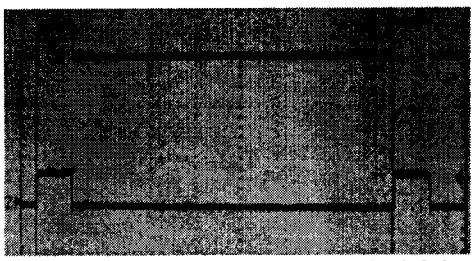

그림 5. 수정된 RTOS에서 구현된 모터 제어신호

성능 비교를 위하여 구현된 소프트웨어 태스크를 Verilog HDL을 이용하여 하드웨어 태스크로 구현하였다. 그리고 위의 실험과 동일한 실험 환경을 주고 신호를 관찰하였다. 관찰결과 신호의 오차는 10us로 나타났다. 이것은 첫 번째 실험보다 오차가 현저히 줄어든 결과로서, 소프트웨어 태스크의 결과와 다르게 신호의 변화가 없이 동일한 신호가 나오는 것을 확인할 수 있다.

4. 결 론

본 논문에서는 RTOS와 SOPC를 이용하여 소프트웨어 태스크와 하드웨어 태스크를 함께 사용하는 실시간 임베디드 시스템을 설계하고 구현하였다. 하드웨어 태스크를 RTOS와 인터페이스하기 위하여 하드웨어 태스크와 RTOS사이에 Task Controller와 Timer를 설계하였다. 그리고 하드웨어 태스크를 관리하기 위하여 uCOS-II의 소스를 수정하고 추가하였다. 또 하드웨어 태스크를 초기화 할 수 있는 코드를 시스템 시작부분에 삽입하였다. 이렇게 만들어진 시스템을 이용하여 간단한 RC-서보 모터 제어기를 구현하고 실험하여 성능을 검증하였다. 이 시스템은 기존 CPU플랫폼 기반의 RTOS시스템보다 실시간 성이 뛰어나고, 하드웨어 로직에 대한 재구성 및 재사용이 용이한 장점이 있다.

[참 고 문 헌]

[1] P.C. French and R.W.Taylor, "A self-reconfiguring processor," Proceeding of IEEE Workshop on FPGA for Custom Computing Machines, pp. 50-59, 1993

[2] R. Hartenstein, "A decade of reconfigurable computing: a visionary retrospective," Proceeding of IEEE Conference and Exhibition on Design, Automation and Test in Europe, pp. 642-649, 2001

[3] A. Kongmunvattana and P. Chongstivatana, "A FPGA-based behavioral control system for a mobile robot," Proceeding of IEEE Asia-Pacific Conference, Circuits and Systems, pp. 759-762, 2000

[4] 최기홍 외 2명, "SOPC기반의 재구성 가능한 로봇제어기 구현," Journal of control, automation, and systems engineering, Vol. 10, No. 3, pp. 261-266, 2004