MCU를 위한 경량화된 RTOS 설계

박창규*

Design of Lightweight RTOS for MCU

Chang-Gyu Bak*

요 약

RTOS는 임베디드 시스템 환경에서 멀티태스킹 동작을 설계하는데 강력한 도구이다. 그러나 협소한 메모리를 가진 MCU에서는 응용프로그램에 비해 기존의 RTOS가 차지하는 비율이 크기 때문에 적용하기 어려웠다. 본 논문에서는 기존의 RTOS에서 사용빈도가 적은 기능을 제거하고, 스케줄링과 자원 관리의 기능만 가지고 최소한의 코드로 동작하는 경량화된 RTOS를 설계하였다. 공유 스택을 사용하여 사용자 메모리를 확보하며, 태스크의 문맥 전환시에 발생하는 오버헤드를 감소시키고, TCB등의 사이즈를 축소하는 기법을 사용하였다. 설계 및 검증 결과, 커널의 사이즈를 1KB이하로 축소할 수 있었고, 커널과 응용 프로그램의 비율을 고려해 볼 때, 본 논문에서 설계한 RTOS는 4KB이상의 프로그램 메모리를 가진 MCU에서 사용할 수 있다.

ABSTRACT

RTOS in the embedded system is a powerful tool for the design of multi-tasking. However, previous RTOS has large proportion in the MCU with limited memory. So it is difficult to apply RTOS. In this paper, I removed less frequently used features from the traditional RTOS, and designed lightweight RTOS that schedules and manages the resources with minimal code. I used techniques to obtain user memory using sharing stack, and to reduce the overhead at context. Considering ratio of kernel and applications, the RTOS designed in this paper is available on the MCU with more than 4KB of program memory.

키워드

실시간 운영체제, 비선점형, 임베디드 시스템, 태스크 제어 블럭

Key word

Real Time Operating System, Non-preemptive, Embedded System, TCB

* 정회원 : 한국 폴리텍Ⅶ대학 부산캠퍼스 전자과
(교신저자 : jdmh@kopo.ac.kr)

접수일자 : 2011. 04. 29
심사완료일자 : 2011. 05. 21

Ⅰ. 서 론

최근 들어 멀티미디어 정보를 처리해야 하는 임베디드 시스템이 늘어나면서 시스템이 복잡해지고 있으며 이를 구현하기 위해 임베디드 시스템은 멀티태스킹으로 동작된다. 멀티태스킹이란 스케줄러를 이용하여 각각의 실행중인 프로그램들의 실행 객체인 태스크의 실행시간을 관리 및 전환하여서 각각의 태스크가 동시에 실행되는 것처럼 보이게 하는 것으로 응용프로그램의 복잡성을 줄이고 관리의 용이성을 가지게 하는 방식이다[1].

초기의 임베디드 시스템은 비교적 단순해서 개발자가 순차적인 프로그램을 작성하였다. 이러한 시스템들을 전경/배경 시스템이라고 하는데, 이 방법은 복잡하기 때문에 프로그램을 작성하기 매우 어렵고, 관리하기도 쉽지 않다. 이를 해결하기 위하여 RTOS가 임베디드 시스템에 적용이 되었다. 그러나 협소한 메모리를 가진 MCU에서는 기존의 RTOS가 차지하는 비율이 크기 때문에 적용하기 어려운 것이 사실이다[2].

본 논문에서는 이를 해결하기 위해 핵심기능인 스케줄링과 자원을 관리하는데 필요한 최소한의 코드를 가지면서, 성능을 저하시키지 않고, 메모리를 많이 요구하지 않으면서 짧은 잠복시간(latency)를 제공하는 경량화된 RTOS를 설계하였다.

Ⅱ. RTOS

RTOS(Real Time Operating System)는 태스크를 시간에 따라 정확하게 관리하고, 시스템 자원을 관리하며, 응용프로그램 개발을 위한 일관된 기반을 제공하는 일종의 프로그램이다. 수행된 태스크에 대해 반응시간이 느리더라도 문제가 없는 범용 OS에 비하여 RTOS의 경우 사용되는 용도에 따라 안정성을 위한 반응성을 지녀야하므로 멀티태스킹에 있어서도 차이를 가지게 되는데 반응성을 충족시키기 위해 RTOS의 경우 시스템에 정해진 방식에 따라 우선순위를 두고 그에 따라 정해진 시간 내에 수행된 태스크에 대해 반응을 할 수 있게 해야 한다.

임베디드 시스템 설계시 RTOS을 사용하면 하드웨어 추상화로 인한 프로그래밍의 능률 향상, 코드의 재사용, 높은 신뢰성과 안정성, 유지보수의 용이성 등의 효과를 얻을 수 있다[1-2].

이상적인 RTOS는 성능을 저하시키지 않고, 메모리를 많이 요구하지 않으면서 코드를 복잡하게 하지 않고, 적절한 시스템 호출을 제공하고 태스크 스케줄링에서도 유연성을 줄 수 있어야 한다. 즉, 간단히 말해서 효율적이면서 쉽게 개발할 수 있도록 해야 한다는 것이다[3].

과거에는 RTOS가 응용프로그램에 부가하는 오버헤드가 작고 외부 이벤트에 빠르고 결정된 반응을 하기 위해서 어셈블리 언어로 작성되었다. 그러나 새로운 프로세서에 이식(porting)이 어렵기 때문에 최근에는 C언어로 전환되었다. 대개 time-critical 스케줄링과 태스크 스위칭 루틴은 어셈블리 언어로 작성하고, 그 외의 부분은 모두 C언어로 작성한다[3].

µC/OS-Ⅱ나 FreeRTOS를 비롯한 기존의 C로 작성된 RTOS들은 32비트 프로세서를 염두해 둔 것으로 처리속도나 메모리 측면에서 8비트 또는 16비트 프로세서에서 동작시키기에는 어려움이 있다.

Ⅲ. RTOS의 설계

본 논문에서는 아래에 해당하는 비교적 성능이 낮은 프로세서에서 동작 가능한 경량화된 RTOS 설계를 목표로 한다.

1. 처리속도(MIPS)가 느린 경우
2. 프로그램 메모리가 협소한 경우
3. 데이터 메모리 공간이 협소한 경우
4. OS 동작 모드가 제공되지 않는 경우

또한 대부분의 MCU들은 다양한 개발환경과 컴파일러가 존재하기 때문에 어떤 RTOS를 설계하든지 다양한 컴파일에서 호환성을 갖도록 제작하는 것이 중요하다.

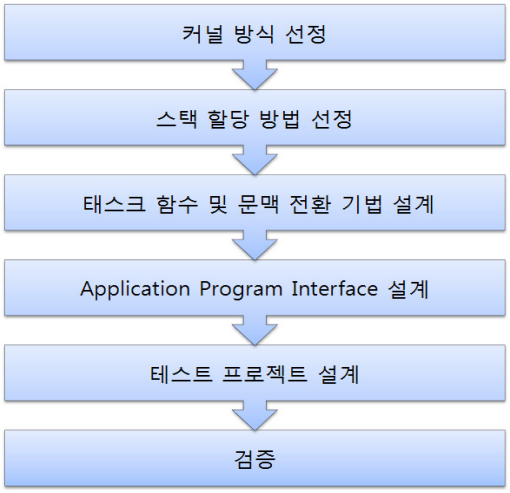

그림 1. RTOS 설계 과정
Fig. 1 RTOS Design Process

본 논문에서는 저사양의 프로세서에 적합한 RTOS를 설계하기 때문에 외부 인터럽트, 커널 서비스, 문맥전환과 관련된 잠복시간을 최소화에 관해 관심을 갖고 설계를 진행하였다. 설계 언어는 표준 C로만 구현하며, 여러 MCU에 이식하기 쉽도록 어셈블리 언어를 배제 하였으며, 이식 작업은 단순히 틱(tick)을 발생시키는 타이머 관련 루틴만 수정하면 되도록 하였다.

3.1. 커널 방식 선정

RTOS는 태스크의 CPU 점유권 선점(preemptive) 가능 여부에 따라 선점형과 비선점형으로 구분한다.

선점형은 OS가 CPU를 독점하게 되고, 태스크 들이 OS로부터 CPU의 점유권을 대여하여 사용하는 방식이다. 태스크는 언제든지 CPU의 소유권을 선점될 수 있어 비선점형 커널에 비해 자유로운 멀티태스킹이 구현된다. 그러나 OS의 권한이 커지기 때문에 구조가 복잡해지고 메모리의 소모도 많아진다[2].

비선점형 커널은 태스크가 CPU를 독점하게 된다. 다시 말해 아무리 우선순위가 낮은 태스크라도 CPU 점유권을 반환할 때까지 다른 태스크는 절대 실행되지 않는다. 멀티태스킹이 태스크 간의 협조로 이루어지므로 협조적 커널(cooperative kernel)라고도 한다. 이 커널 방식은 기능이 제한적이지만 커널의 코드 사이즈가 줄어들고, 인터럽트 응답이 좋은 효과를 얻을 수 있다[3].

본 논문에서 설계하는 RTOS는 커널의 좋은 성능보다 메모리 사용량의 절약에 비중을 더 두기 때문에 비선점형 커널 방식을 사용한다.

3.2. 스택의 할당 방법 선정

µC/OS-Ⅱ를 비롯한 선점형 RTOS의 실행되고 있는 태스크들은 어떤 시점에서 문맥 전환될 지 예측할 수 없기 때문에 각 태스크 마다 독립적인 스택을 가져야 한다[2][5].

반면에 비선점형 RTOS는 각 태스크 마다 독립 스택을 갖도록 설계할 수 있을 뿐만이 아니라, 모든 태스크가 하나의 스택을 공유하는 방법을 사용할 수도 있다. 표 1은 각 스택 할당 방법에 따른 특징을 나타내고 있다.

표 1. 독립스택방법과 공유스택방법 비교
Table. 1 Comparison of independent stacks method and sharing stacks method

	독립 스택	공유 스택
프로그램 메모리 사용	컴파일러의 스택 사용 기본 코드 + 스택 할당 코드 + 스택 관련 문맥전환 코드	컴파일러의 스택 사용 기본 코드
데이터 메모리 사용	TCB 내에 스택 정보 저장	없음
태스크 함수내의 지역변수	자유로움	static 선언 필요 (RAM 상주)
특징	태스크의 독립성 확보	비교적 빠른 문맥 전환 속도

스택의 할당방법 선정에 있어서 MCU의 제한된 사양을 고려했을 때 스택을 공유하여 문맥전환속도 및 메모리 사용에 이득을 취하는 것이 유리하다.

3.3. 태스크 함수 및 문맥 전환 기법 설계

앞의 두 절에서는 RTOS의 설계 방향을 비선점형 공유 스택 RTOS로 선정하였다. 이 방식은 다음과 같은 간단한 작성 규칙을 갖는다.[6]

- 시스템 호출은 문맥전환을 일으킬 수 있기 때문에 태스크의 최상위에서만 호출해야한다. 즉, 태스크 내에서 호출하는 함수에서는 시스템 호출을 할 수 없다.
- 태스크 내의 지역변수는 시스템 호출 동작에서 보존될 수 없다. 만약에 지역변수를 보존하려면 정적 변수로 선언하여야 한다.

스케줄러 동작 중 문맥전환이 이루어지는 동안은 실제 응용 프로그램에서 원하는 작업은 이루어지지 않기 때문에 오버헤드라고 볼 수 있다. 고속 또는 고성능의 프로세서에는 이 문제가 크게 발생하지 않지만 현재 본 논문에서는 낮은 사양의 프로세서를 타겟으로 하고 있기 때문에 설계 시에 다른 전략으로 접근할 필요가 있다.

태스크1에서 하던 작업이 중단되고 태스크2를 수행한다고 가정할 때 태스크2가 수행되기 이전에 태스크1의 상태를 저장해 둘 필요가 있다. 그래서 태스크1이 중단되기 전에 레지스터 값들을 태스크1의 스택에 보관을 하고, 태스크2가 수행되고 다시 태스크1이 수행될 시점이 오면 스택에 저장되었던 값들을 레지스터로 옮겨와서 태스크1을 다시 수행하게 된다. 이를 문맥전환(context switching)라 한다.[3-4]

문맥전환을 구현하는 방법은 대개 어셈블리 언어로 작성되고, 저장할 레지스터의 개수에 따라, 사용되는 프로세서 등에 따라 달라진다. 이 방법으로 구현할 경우 태스크 함수가 사용하는 레지스터와 무관하게 변동의 소지가 있는 모든 레지스터를 TCB에 저장 및 복원하게 된다.

C 컴파일러가 함수를 호출하는 과정을 번역한 결과를 확인하면, 먼저 함수가 호출되기 전에 함수 내에서 사용하는 범용 레지스터만을 스택에 대피시키고 함수가 종료될 때 복원시킨다. 이는 문맥 전환 시에 일반적인 커널의 동작과 유사하고, 저장위치가 TCB가 아닌 스택이라는 점이 다르다. 이를 이용하면 문맥 전환 기법은 다음과 같은 방식으로 설계할 수 있다.

```
void task_name(uint8_t *goto_line){
 if (goto_line) goto *goto_line;
 for(;;) {
  …
  //문맥전환
   __save_goto(&&___CONCAT(L,__LINE__));
   __CONCAT(L,__LINE__):
  //복귀조건
   if (! … ){ wait(1); return; }
  …
 }
}
```

C언어의 매크로를 이용하여 레이블을 생성하고, 다른 태스크로 전환 시에 레이블의 주소를 TCB에 저장하고, 복귀 시에 태스크 함수를 재호출하면서 매개변수로 입력되는 주소를 사용하여 다음 실행할 문장으로 복귀한다.

이 방법을 사용하면 TCB마다 모든 레지스터의 문맥정보를 저장할 필요가 사라져 가용메모리를 확장하는 효과가 있다.

3.4. Application Program Interface 설계

본 논문의 RTOS의 태스크는 그림 4와 같은 상태천이를 가진다. 태스크의 형태는 사이즈를 고려하여 가장 고전적인 형태를 가지도록 하였다. [1][5-6]

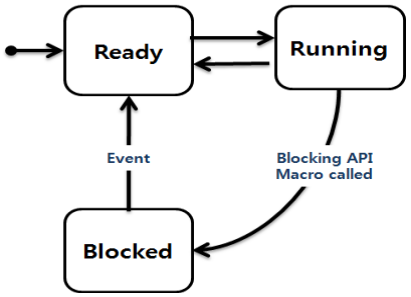

그림 2. 태스크의 상태천이도
Fig 2. Task State Transition Diagram

사용자 응용프로그램에서 RTOS의 시스템 호출을 지원하기 위한 API는 표 2와 같고 기존의 RTOS를 참고하여 설계하였다[5-8].

대부분 매크로로 작성되어 있으며 이진 세마포어를 옵션사항으로 두어 공유자원을 관리하도록 설계 하였다.

표 2 . RTOS API 설계
Table. 2 RTOS API Design

기본 API	세마포어 관련
osWAIT(ticks) osWAIT_WHILE(cond) osWAIT_UNTIL(cond) osWAIT_IRQ(cond) osYIELD() osSUSPEND() osGET_PRIORITY(task) osSET_PRIORITY(task,pri) osCLEAR_RTR(task) osDISABLE_TASK(task) osSET_RTR(task) osENABLE_TASK(task)	osGET_SEMA(s) osRELEASE_SEMA(s) osSET_SEMA_OWNER_RTR(s) osENABLE_SEMA_OWNER(s)

Ⅳ. 검 증

앞서 설계한 RTOS의 동작을 검증하기 위해 AVR 계열의 ATmega 128 MCU를 내장한 "RTEB-128"라는 RTOS training board 제품을 이용하였다[10].

먼저, 커널 사이즈를 확인하기 위해 가장 간단한 두 개의 LED 점멸 태스크를 작성하여 AVR Studio 5의 내장 컴파일러로 컴파일한 결과 800Byte 이내의 실행 파일을 얻을 수 있었다. 그리고 LIST 파일 및 메모리 맵을 확인 결과, 기존 의 경량화된 RTOS[12]에 비해 1/5 정도인 5 Byte의 TCB을 생성하여 효율적인 메모리 관리가 가능함을 확인하였다. AVR 시뮬레이터를 이용하여 문맥전환이 이루어지는 과정이 함수호출 레벨에서 신속하게 진행됨을 확인하였다.

커널 사이즈를 고려했을 때 프로그램 메모리가 4KB 이상인 AVR에서는 RTOS를 동작시킬 수 있을 것이다.

다음으로, 스케줄러 및 세마포어의 동작을 검증하기 위해 6개의 태스크가 각자 하나 이상의 하드웨어를 제어하고 공유자원인 ADC와 문자 LCD 모듈을 관리하기 위해 각각 세마포어를 할당하였다. 태스크의 구성은 그림 3과 같이 하였고, [9-11]을 참고하여 태스크를 작성하였다.

테스트 결과 각 태스크가 동시간대에 정상적으로 동작하고, 특히 문자 LCD가 세마포어를 이용하여 자원의 제어권을 충돌 없이 관리할 수 있음을 확인하였다. 오동작 유무를 확인하기 위하여 다른 하나의 UART 포트를 사용하여 PC로 모니터링 한 결과 48시간 이상 구동하여 안정적으로 구동함을 확인하였다.

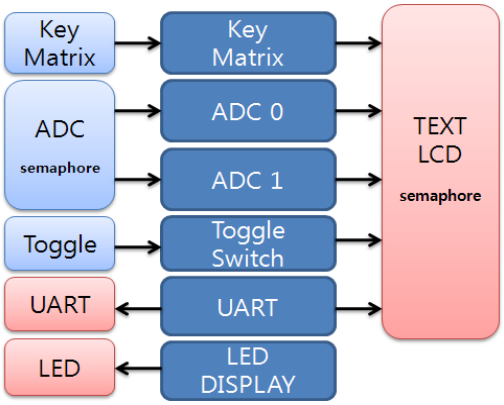

그림 3 . 검증을 위한 프로젝트
Fig. 3 Project for Verification

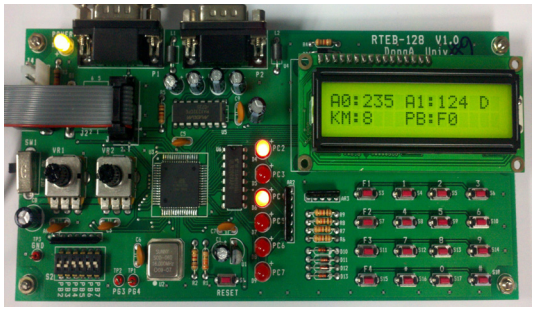

그림 4. 하드웨어를 이용한 검증
Fig. 4 Hardware Verification

Ⅴ. 결 론

본 논문에서는 저사양의 MCU에서 멀티태스킹환경을 쉽게 구현할 수 있도록 RTOS를 설계하였다.

프로그램 및 데이터 메모리의 절약, 문맥전환 및 인터럽트 응답 속도 개선의 목적에 맞도록 고려한 결과, RTOS의 설계 방향을 비선점형 공유 스택 RTOS로 정하고, 어셈블리 문맥전환 방식이 아닌 매크로에 의한 방식으로 하였다.

간단한 태스크 프로젝트를 작성하여 메모리 할당 조사 및 시뮬레이터 상에서 동작을 검증한 결과 본 논문에서 설계한 RTOS는 태스크 당 기존 경량화된 RTOS에 비해서도 1/5 정도인 5Byte의 TCB을 생성하며 효율적인 메모리 관리가 가능함을 확인하였다. 그리고 TCB에 현재 문맥을 모두 저장하는 기존 방식에 비해 함수호출 레벨에서 문맥전환이 신속하게 진행됨을 확인하였다. 이는 문맥 전환 성능 개선을 위한 다중 레지스터 파일을 지원하지 않는 저사양 MCU에 효율적으로 사용될 수 있음을 말한다.

하드웨어에 의한 검증결과, 6개의 태스크와 2개의 세마포어가 안정적인 동작을 함을 확인할 수 있었다.

본 논문에서 설계한 RTOS는 최소한의 코드로 스케줄링을 지원하기 때문에 다음 사항을 고려하여 태스크를 작성해야 한다[1][4].

첫째, 태스크의 응답속도는 태스크의 최대 실행시간과 밀접한 관련이 있다. 만약, 최대시간이 길어질 경우 응답속도를 개선하기 위하여 osYIELD()를 호출하여 다른 태스크에게 CPU점유권을 명시적으로 양보할 필요가 있다.

둘째, 태스크 간의 사용자의 자원관리 부주의로 인한 교착상태가 일어나지 않도록 주의해야 한다.

본 논문에서는 AVR 계열을 중점적으로 검증하였다. 향후에 다른 계열의 MCU에 이식하여 검증한 후 공개할 계획이다. 제안하는 RTOS가 다양한 프로젝트에 적용될 수 있으리라 사료된다.

참고문헌

[1] http://en.wikioedia.org/wiki/Real-time_operating_system

[2] Jean J. Lavrosse, “MicroC/OS-II 실시간 커널 제2판”, 에이콘 출판사, 2005

[3] 송용수, ″ARM 7 BIBLE″, 복두출판사,2004

[4] Silverschatz ,Galvin ,″Operating System Conceps 7th Edition″, JOHN WILEY & SONS. INC, 2004

[5] http://www.FreeRTOS.org

[6] http://www.circuitcellar.com/avr2004/DA3650.html

[7] 김범준, “만들면서 배우는 OS커널의 구조와 원리” 한빛미디어, 2005

[8] 카와이 히데미, “OS 구조와 원리:OS 개발 30일 프로젝트”,한빛미디어, 2007

[9] Jean J. Lavrosse, “Embedded Systems Building Blocks 2nd Edition”, 에이콘 출판사, 2008

[10] 김종욱, “혼자서도 즐기는 임베디드 시스템”, 홍릉과학출판사, 2010

[11] Atmel, ″ATmega128 DataSheet″, Atmel Corporation, 2011

[12] 손현승, “소형 다관절 로봇을 위한 RTOS”, 홍익대학교 대학원, 2008

저자소개

박창규(Chang-Gyu Bak)

1992년 부경대학교 전자공학과 공학사

1997년 부경대학교 전자공학과 공학석사

1977년~1997년 한독직업 전문학교교사

1998년~현재 한국폴리텍Ⅶ대학 부산캠퍼스 전자과 부교수

※관심분야 : 임베디드 시스템, 디지털 신호처리, 마이크로프로세서, 실시간 RTOS