uClinux기반의 Real-Time Clock 모듈 인터페이스 최적화 방안에 관한 연구 및 구현

하성준*, 김홍규, 문승진
수원대학교 IT대학 컴퓨터학과
e-mail : {e11even21*, exxxfx, sjmoon}@suwon.ac.kr

The Study and Implementation of a Real-Time Clock Module interface optimizer based on the uClinux

Sung-jun Ha*, Hong-kyu Kim, Seung-jin Moon
Dept. of Computer Science, Suwon University

요 약

오늘날 대부분의 임베디드 시스템에서 사용하는 uClinux에서 기본적으로 프로세스가 이용할 수 있는 범위의 시스템 클럭은 10m/s 이상이다. 기존에는 무리하게 시스템 클럭의 속도를 무리하게 높여 더 높은 정밀도를 요구하는 프로세스를 처리해 왔다. 이는 시스템 리소스를 많이 사용함과 동시에 타이머 인터럽트를 처리하는 오버헤드도 상대적으로 증가하여 전체적으로 시스템의 성능과 안정성에 좋지 못했다. 이에 본 논문에서는 uClinux기반의 임베디드 장치와 Real-Time Clock(RTC)모듈과의 인터페이스 최적화 방안에 관하여 제안한다. 이로써 시스템 클럭을 사용하지 않고, RTC 자체의 인터럽트를 사용해서 작업을 진행해 나가기 때문에 시스템 리소스를 적게 사용하며, 시스템의 성능에 영향을 적게 준다. 또한 알고리즘적인 최적화를 사용 코드최적화를 사용하여 임베디드 시스템에서 가장 효율적으로 관리해야할 리소스인 메모리를 절약, 기존의 방식과 차별을 두었다.

1. 서 론

임베디드 장치의 내장형 운영체제는 대부분 임베디드 리눅스를 사용하고 있으나 일부는 포켓 PC나 WinCE를 사용하고 있다. 포켓 PC나 WinCE는 다양한 디바이스 드라이버와 커널 간 모듈 인터페이스가 상당히 잘 구성되어 있다. 하지만 대부분의 임베디드 장치에서 사용되고 있는 임베디드 리눅스기반의 오픈 소스 기반의 RTC 모듈은 커널과의 인터페이스 문제점을 갖고 있다. 이러한 문제점은 임베디드 장치의 제약조건으로 인한 RTC 모듈과 운영체제 커널의 인터페이스를 잘못 구현하여 발생하며 시스템 리소스의 불필요한 낭비로 다운되는 안정성의 문제가 발생한다.

이에 본 논문에서는 임베디드 리눅스 기반의 uClinux를 사용하여 RTC모듈과 운영체제 커널간의 인터페이스 문제를 해결하기 위한 RTC 디바이스 드라이버와 운영체제 커널 인터페이스 최적화 하는 방법에 관하여 제안하고자 한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서 RTC와 uClinux에 관하여 간략하게 설명하고, 3장에서 RTC 사용을 위한 하드웨어 분석과 디바이스 드라이버 구현에 관하여 논의하고, 4장에서는 본 논문의 결론과 앞으로 해결해야 할 과제들을 제시한다.

2. 관련 연구

2.1 Real-Time Clock(RTC)

RTC는 32.768Khz (20ppm 이내 정확도) crystal oscillaor의 진동을 사용하여 시간을 세는 전자 부품이다. 10개의 핀으로 구성된 이 소자는 전원이 처음 공급될 때 알 수 없는 어떤 시간으로 설정되고, 시스템 설계자에 의해 적당한 현재 시간이 입력되도록 프로그램 된다.

RTC는 네 개의 신호라인(CE, SCLK, SI, SO)을 통해 CPU와의 인터페이스를 구현하며, 보통은 BCD

형태로 년, 월, 시, 분, 초를 설정하면 별도의 CMOS 전지를 사용하여 컴퓨터가 꺼져도 데이터의 손실 없이 계속 동작한다. 컴퓨터가 각종 업무를 처리할 때는 현재의 시각과 날짜를 이용하는 경우가 많으므로, 최근에는 대부분의 컴퓨터에 설치되어 있다. RTC의 주요 기능으로는 운영체제가 기본적으로 제공하는 system clock보다 정확한 시간과 달력 제공과 자체 리소스 사용으로 시스템의 전체적인 성능에 영향을 적게 주면서 보다 정밀한 작업에 효율적이다. 또한 알람 기능을 제공해주는데 RTC는 지정된 시간에 알람 신호를 발생하고, 그때 알람 Interrupt가 작동하게 된다.

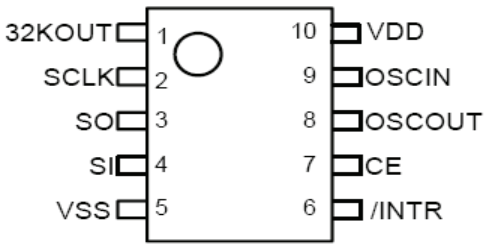
2.2 uClinux

Mobile device에 대한 사람들의 관심이 높아지면서, 기존의 embedded 장비에 linux를 적용하고자 하는 사람들이 많아지고 있다. 게다가 linux가 네트워크 분야에서 많은 장점을 가지고 있고, 여타의 embedded OS에 비해 개발 환경 구축 및 개발 작업이 손쉽게 이루어 지면서, 저렴한 비용으로 사용이 가능하므로, embedded linux에 대한 관심도는 가히 폭발적으로 증가하고 있다. 일반적으로 사람들이 사용하고 있는 서버용 이나 데스크 탑 용의 리눅스 배포판과 같이, embedded linux에도 여러가지 배포판이 존재한다. 그 중에서, MMU-less Processor를 위한 embedded linux의 한 종류인 uClinux 임베디드 리눅스 분야에서 널리 알려져 있는 리눅스 커널의 변형된 모습 중 하나이다.

uClinux(MicroControllerLinux)는 1998년에 Dragonball 68k 시리즈에 처음으로 포팅 되었으며, 그 이후로 수많은 타겟 프로세서에 포팅이 되었다. 이는 uClinux는 MMU(Memory Management Unit)를 가지고 있지 않은 32비트 프로세서를 지원하기 위하여 MMU지원 기능을 커널에서 삭제하였기 때문에, 현재 데스크 탑 또는 서버에서 널리 사용되고 있는 주류의 linux 커널과는 구별되는 리눅스 커널이다. 또한 uClinux는 작은 자원을 가지고 있는 마이크로 컨트롤러에서 사용하기 위해, footprint가 매우 작으며, BFLT(Binary Flat) 실행 형태를 가지고 있다.

3. Using RTC in uClinux

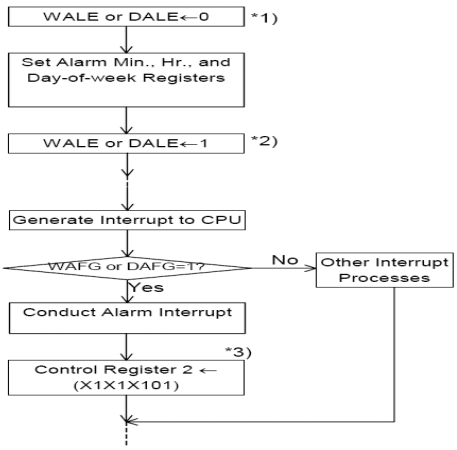
3.1 RTC 하드웨어의 분석



(그림 1) Pin Configuration

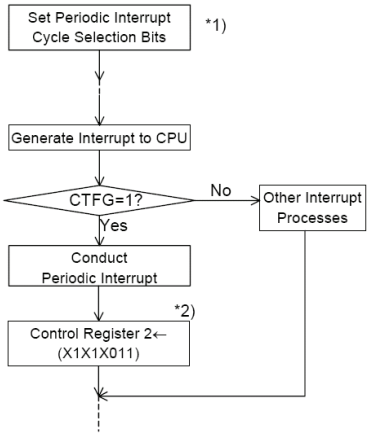
- Chip Enable(CE) : CE pin은 CPU와의 인터페이스를 위해 사용된다. CPU에 연결을 하기위해서는 high로 유지해 주어야 한다. CPU이 전원이 끊겼을 때는 반드시 low로 유지하거나 open해주어야 한다.
- Serial Clock(SCLK) : SCLK Pin은 SI 와 SO pin으로의 입·출력 데이터와 동기화하는 클럭펄스 입력에 사용된다.
- Serial Input(SI) : SI pin은 writing을 위한 데이터의 입력에 사용된다.
- Serial Output(SO) : SO pin은 reading을 위한 데이터의 출력에 사용된다.
- Interrupt Output(/INTR) : /INTR pin은 alarm interrupt 밟생과 CPU로 주기적인 interrupt 신호 발생을 위해 사용된다.
- 32kHZ Clock Output(32KOUT) : 32KOUT pin은 32.768kHz 클럭펄스 발생을 위해 사용된다.
- Oscillation Circuit Input/Output(OSCIN/OSCOUT) : OSCIN / OSCOUT pin은 32.768kHz 크리스탈 Oscillator 연결에 사용된다.
- Positive / Negative Power Supply(VDD/VSS) : VDD / VSS pin은 +/- 공급전압에 연결된다.

3.2 Interrupt Algorism



(그림 2) Alarm Interrupt Process

RTC가 사용하는 interrupt의 source가 될 수 있는 것은 크게 두 가지로 나눌 수 있다. 첫 번째는 alarm interrupt로써 초당 한번부터 하루에 한번까지 가능하다.



(그림 3) Periodic Interrupt Process

두 번째 source는 periodic interrupt로 0.5초에 한 번에서 부터 30.517m/s에 한번까지가 가능하다.

3.3 Initialization

시리얼 통신을 시작하기 이전에 사용하게 될 포트를 초기화 해야한다. 즉, 모든 포트를 low로 만들어 준다.

<표 1> Initializaion 구현 소스

```
static int __init rtc_init(void)
{
#if defined(__alpha__) || defined(__mips__)
unsigned int year, ctrl;
unsigned long uip_watchdog;
char *guess = NULL;
#endif
#ifdef __sparc__
struct linux_ebus *ebus;
struct linux_ebus_device *edev;
#ifdef __sparc_v9__
struct isa_bridge *isa_br;
struct isa_device *isa_dev;
}
#ifdef __sparc__
for_each_ebus(ebus) {
for_each_ebusdev(edev, ebus) {
if(strcmp(edev->prom_name, "rtc") == 0) {
rtc_port = edev->resource[0].start;
rtc_irq = edev->irqs[0];
goto found;
                    }
                }
        }
#ifdef __sparc_v9__
for_each_isa(isa_br) {
for_each_isadev(isa_dev, isa_br) {
if (strcmp(isa_dev->prom_name, "rtc") == 0) {
rtc_port = isa_dev->resource.start;
rtc_irq = isa_dev->irq;
goto found;
                    }
                }
        }
#endif
printk(KERN_ERR "rtc_init: no PC rtc found\n");
return -EIO;
found:
if (rtc_irq == PCI_IRQ_NONE) {
rtc_has_irq = 0;
goto no_irq;
}
```

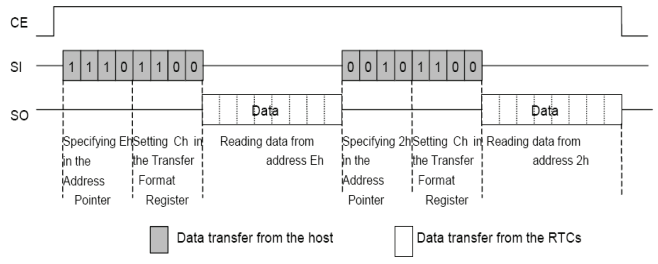
rtc_init() 함수의 첫부분에서는 먼저 시스템에 RTC가 있는지를 확인하는 절차를 지닌다. Sparc의 경우에는 각 EBUS에 대해서 RTC가 있는지를 찾는다. 이때 사용하는 리소스의 시작번지를 rtc_port로 두고, 사용하는 인터럽트 번호를 rtc_irq로 둔다. Sparc v9에서는 ISA 버스상에서도 이와 같은 것이 있는지를 확인 한다. 없다면, RTC 디바이스를 찾을 수 없다는 에러메시지를 출력한다. RTC를 찾았다면, ISA 디바이스로서 가지고 있는 RTC의 자원을 가져와서 이를 RTC가 사용하는 포트 번호와 IRQ번호를 둔다.

3.4 Read Operation

<표 2> Read 구현소스

```
_buffer = 0;
_outbit(PPCE,RTC_SDA,HIGH);
_bit_init(PPCE,RTC_SDA,0);
for(_counter = 0; _counter < 8; _counter++) {
_buffer >>= 1;
if(_inbit(PPCE,RTC_SDA))_buffer |= 0x80;
rtc_delay(1);
_outbit(PPCE,RTC_SCL,HIGH);
rtc_delay(2);
_outbit(PPCE,RTC_SCL,LOW);
rtc_delay(1);
}
_bit_init(PPCE,RTC_SDA,1);
return _buffer;
```

사용하고자 하는 레지스터의 값을 읽어서 반환해 준다. 매개변수(addr)은 읽고자 하는 내부 레지스터 어드레스입니다.



(그림 4) Reading Data Transfer

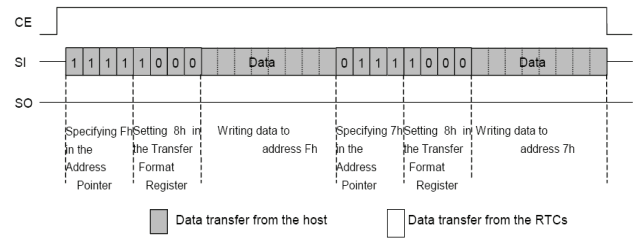
그림 4는 RTC의 데이터 reading 프로토콜을 보여준다. 데이터의 reading은 한번에 1byte씩 writing이 지정된 레지스터로 전송되며, CE pin이 low상태가 되어 완료되거나 다음 pointer 시작주소로부터 다시 같은 방식으로 전송 된다.

3.5 Write Operation

<표 3> Write 구현소스

```
void _write(int addr, int buffer)
{
_outbit(PPCE,RTC_SCL,LOW);
_outbit(PPCE,RTC_nRST,HIGH);
rtc_delay(2);
_snd_address(addr);
_snd_data(buffer);
rtc_delay(1);
_outbit(PPCE,RTC_nRST,LOW);
}
```

RTC 내부 레지스터에 임의의 설정값을 저장하기 위한 함수로서, 매개변수로는 어드레스(addr), 설정값(buffer)가 전달 된다. void _refresh()로부터 호출을 받도록 되어 있다. void _refresh()는사용자가 RTC의 현재의 시간이나, 날짜를 설정하기 위해서 호출하는 함수이다.



(그림 5) Writing Data Transfer

그림 5는 RTC의 데이터 writing 프로토콜을 보여준다.. writing은 reading과 마찬가지로 한번에 1byte씩 지정된 레지스터에 저장되며, CE pin이 low상태가 되어 완료되거나 다음 pointer 시작주소로부터 다시 같은 방식으로 전송 된다.

4. 결론 및 연구과제

본 논문에서는 uClinux에서 높은 정밀도의 주기적인 clock을 사용하기 위해서 RTC를 사용할 수 있다는 것과 RTC 모듈과 임베디드 운영체제인 uClinux의 인터페이스를 최적화 하여 결과적으로 시스템의 안정성과 성능, 메모리를 향상시키기 위한 방법에 관하여 제안하였다. 이러한 방법을 경량 임베디드 시스템에 사용하게 된다면, RTC를 사용하여 적은 시스템 리소스로 효율적인 작업을 할 수 있다. 향후 본 논문을 바탕으로 소형화된 임베디드 시스템에 적용하여 발생될 수 있는 RTC 모듈과 uClinux 만이 아닌 이기종의 임베디드 운영체제와의 문제점을 해결할 수 있도록 연구할 것이다. 마지막으로 응용프로그램의 특성이 RTC를 요구한다면, 그 적절한 사용 방법도 중요하지만 실제 내부구조는 어떻게 되어 있는가를 이해하는 것이 디바이스 드라이버와 응용프로그램의 상호작용을 이해하는데 핵심이 된다는 것을 간과해선 안될 것이다.

참고문헌

[1] 유영창, "리눅스 디바이스 드라이버", 한빛미디어, 2004

[2] 권수호, “Linux Programming Bible”, 글로벌, 2002

[3] 차영배, “ARM7TDMI(마이크로컴퓨터 LPC2194)”, 다다미디어, 2004

[4] "Korea Embedded System for Linux", (http://www.kesl.co.kr)

[5] "Korea Embedded Linux Project", (http://www.kelp.or.kr)

[6] "Ricoh Global", (http://www.ricoh.com/)

[7] "Embedded Linux System" (http://calab1.pknu.ac.kr/embedded/)