한국시뮬레이션학회 2001춘계학술발표대회 논문집 2001.5.12 서울산업대학교

실시간 운영체제 시뮬레이터의 구현

김방현, 이종은, 김종현

Implementation of the Simulator for a Real-Time Operating System

Bang Hyun Kim, Jong Eun Lee, Jong Hyun Kim
Computer System Lab, Yonsei University

요약문

실시간 운영체제 개발환경에서 제공하는 도구들 중의 하나인 실시간 운영체제 시뮬레이터는 타겟 하드웨어(target hardware)가 호스트에 연결되어 있지 않은 상태에서도 사용자가 응용프로그램의 개발과 디버깅을 가능하도록 해주는 시뮬레이션 환경이다. 본 연구에서는 현재 국내에서 실시간 컴퓨터시스템을 위하여 자체 개발중인 실시간 운영체제인 Q+를 위한 시뮬레이터를 구현하였다. 또한 본 연구에서는 상용화될 제품에 실제 적용이 가능한 방법을 개발하는데 중점을 두었으며, 실행 시간을 추정하는 기능도 포함하였다. 본 연구에서 대상으로 한 타겟 하드웨어는 ARM 계열의 StrongARM SA-110 마이크로프로세서와 21285 주제어기가 장착된 EBSA-285 보드이며, 개발 환경은 윈도우 상에서 동작하는 Q+Esto이다.

1. 서론

실시간 운영체제(Real-Time Operating System: 이하 RTOS라 함)는 실시간 컴퓨터시스템에서 자원관리와 프로세스 스케쥴링 등을 지원하는 핵심 시스템 소프트웨어이다. 최근 정보가전기기가 급속히 발전함에 따라 소형의 내장 컴퓨터시스템(embedded computer system)들을 위한 고성능 저가격 RTOS의 개발이 필수적이다. 또한 그러한 시스템의 개발 과정에서 타겟 하드웨어(target hardware)가 개발되기 전에 시스템 디버깅이나 응용프로그램의 개발이 가능할 수 있도록 해주는 개발 환경들도 제공되고 있다. 그러한 환경은 RTOS의 선택에 중요한 조건이 되기 때문에 독립성, 개방성, 신뢰성 및 사용자의 편의성을 충족하는 개발환경에 관한 연구는 필수적이라고 할 수 있다[1].

현재 상용화되어 보급되고 있는 제품들로는 미국의 VxWorks의 VxSim[2]과 VRTX의 MIPS XRAY Simulator[3], 그리고 RT-Linux의 Carbon- Kernel[4] 등이 있다. VxSim은 VxWorks에 포함된 시뮬레이

터로서, 타겟 시스템에서 실행될 VxWorks를 호스트에서 하나의 프로세스로 간주하고 실행함으로서 가상 타겟 환경을 제공하며, VxSim 실행 창은 타겟 보드의 표준 입출력 역할을 한다 [2]. 이 시뮬레이터는 현재 구현된 실시간 운영체제 시뮬레이터 중에서 가장 다양한 기능들을 제공하고 있으며, 실시간 운영체제 기능 자체도 시뮬레이션 하는 강력한 기능을 가지고 있다.

VxSim과 대조적인 시뮬레이션 방법으로 구현된 MIPS XRAY Simulator[3]는 VRTX의 시뮬레이터로서 XRAY 디버거의 부분적인 기능으로 동작한다. 이 시뮬레이터는 MIPS32 프로세서 명령어-세트 시뮬레이터로서, 응용프로그램의 소스 단계에서 명령어 자체를 해석하여 시뮬레이션 한다.

CarbonKernel[4]은 사건-구동(event-driven) 시뮬레이션 기법에 기반을 둔 RT-Linux 시뮬레이터로서, 소스 코드를 분석하여 소스 코드가 실행될 때 발생하는 이벤트들을 시간선(time-line)에 스케줄링하고, 가상 실시간 운영체제 커널 API를 통하여 실시간 운영체제 동작을 수행한다.

국내에서도 네이버웰드의 Symphos[5]나 아로마 소프트의 TeaPot [6] 등과 같이 특정 응용분야를 위한 실시간 운영체제가 구현된 적은 있으나, 활용범위가 제한적이고 통합되어 있는 개발도구의 부족으로 인하여 실제 사용되지는 못하고 있는 실정이다. 따라서 RTOS가 필요한 국내업체들 대부분은 통합개발도구를 제공하는 RTOS들을 외국으로부터 도입하거나 분야별로 개량하여 사용하고 있다. 최근 국내에서는 그로 인한 막대한 기술료 지불로부터 탈피하고자 많은 노력을 기울이고 있으며, 특히 한국전자통신연구원(ETRI)에서는 프로세스 기반의 RTOS인 Q+와 실시간 운영체제 개발환경인 Q+Esto를 2000년에 개발하였으며, 현재 계속 보완 중이다 [7].

본 연구에서 개발하고자 하는 RTOS 시뮬레이터는 실시간 내장컴퓨터시스템인 타겟 하드웨어가 호스트(host)에 실제 연결되지 않은 상태에서도 응용프로그램의 개발과 디버깅을 가능하게 해주는 타겟 시뮬레이션 환경을 제공하는 도구로서, 개발자로 하여금 타겟 하드웨어의 개발이 완료되지 않은 상태에서도 응용 프로그램을 개발할 수 있도록 해준다. 그러한 유용성 때문에 현재 대부분의 상용 실시간 운영체제 개발환경들은 RTOS 시뮬레이터를 제공하고 있지만, 이들은 대부분 기능적인 시뮬레이션만 가능하며 운영체제나 응용프로그램이 실제 타겟에서 동작할 때의 실행 시간 추정은 불가능하다는 문제점을 가지고 있다.

따라서, 본 연구에서는 그러한 문제를 해결하고자 명령어 기반(instruction-based) RTOS 시뮬레이터를 구현하는 방법을 개발하였다. 대상으로 한 RTOS는 ETRI에서 개발 중인 Q+이며, 타겟 하드웨어는 ARM 계열의 StrongARM SA-110 마이크로프로세서와 21285 제어기가 장착된 EBSA-285 보드이다.

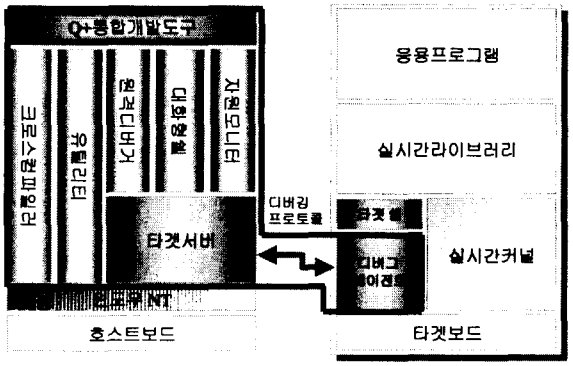
2. 시뮬레이션 환경

2.1 Q+ 및 Q+Esto

Q+는 ETRI에서 2000년에 개발된 정보가전용 실시간 운영체제로서, 디지털 TV와 인터넷 TV 등에 활용이 가능한 기능을 갖추고 있으며, 국제 표준 및 업계 표준과의 호환성을 최대한 보장한다. 또한 Q+는 다양한 정보가전기기들에 쉽게 활용될 수 있게 하기 위하여 조립성 및 확장성을 가질 수 있는 다중 계층 구조로 구성되어 있다[7].

Q+의 통합개발도구 Q+Esto는 원격지 호스트에서 목적프로그램을 개발하여 타겟 보드(targer board)에 적재하여 실행하고, 실행상태와 자원사용을 추적 관찰하며 발생하는 오류의 원인을 찾아 제거하도록 크로스 컴파일러, 유틸리티, 원격 디버거, 대화형 쉘, 자원모니터 등의 도구와 타겟서버, 디버그 에이전트 등으로 구성한 서브시스템이다.

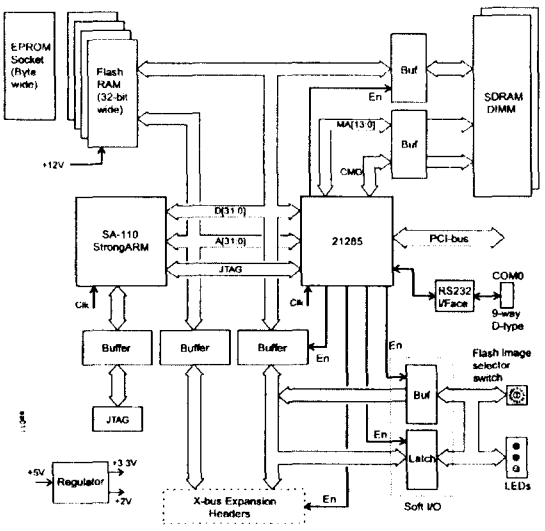
<그림 1>은 Q+Esto의 구성과 개발환경을 보여주고 있다.



<그림 23> Q+Esto의 구성과 개발환경

2.2 테스트용 플랫폼(EBSA-285 보드)

Q+와 Q+Esto의 테스트용 플랫폼으로는 ARM 계열의 StrongARM SA-110 마이크로프로세서와 21285 주제어기(Core Logic Controller)가 장착된 인텔의 EBSA-285 보드를 사용하였다. EBSA-285 보드는 내장시스템에 들어가는 SA-110과 21285 주제어기를 평가하기 위한 보드로서, SA-110을 이용하는 내장시스템에 사용되는 하드웨어와 소프트웨어를 개발할 때에 프로토타입으로 많이 사용된다. <그림 2>는 EBSA-285의 내부 구조를 보여주고 있다 [8].



<그림 24> EBSA-285의 구조

SA-110 프로세서는 휴대용 제품이나 대화형 디지털 비디오와 같은 내장시스템에 사용될 수 있는 저전력, 고성능의 범용 32비트 RISC 마이크로프로세서이다. 캐쉬는 16K바이트의 명령어 캐쉬와 16K바이트의 쓰기 지연(write-back) 방식의 데이터 캐쉬가 내장되어 있고, 사상방식은 32-way 집합 연관 사상 방식을 사용하며, 교체방식은 라운드 로빈(round robin)을 사용한다[9].

21285 제어기는 SA-110 마이크로프로세서를 위한 칩으로, SA-110과 DRAM 메모리, 플래쉬 롬, 그리고, PCI 버스 사이의 인터페이스 역할을 한다. 또한 전원 관리기, DMA 제어기, 인터럽트 제어기, 버스 중재기 등의 역할도 수행하는 다기능 단일 칩 반도체이다[10].

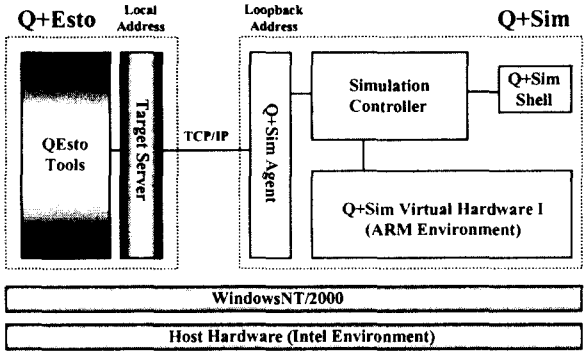
3. Q+ 시뮬레이터의 설계와 구현

3.1 설계 개념

본 연구는 제2장에서 설명한 실시간 운영체제인 Q+를 위한 시뮬레이터(Q+ Simulator: 이하 Q+Sim이라 함)를 실제 상용화가 가능하도록 설계하는데 중점을 두었다. 즉 실시간 운영체제 개발환경인 Q+Esto에서 제공하는 하나의 도구로서, EBSA-285 보드와 같은 타겟이 호스트에 연결되어 있지 않더라도 타겟을 연결하여 작업할 때와 동일한 결과를 개발자에게 제공할 수 있게 하는 것을 목표로 Q+Sim을 설계하였다.

이를 위해 본 연구에서 적용한 방법은 내부 프로세서간 통신을 위한 재귀(loopback) IP 주소와 가상 하드웨어(virtual hardware) 위에서 타겟 프로세서의 명령어들을 가상적으로 실행되도록 하는 명령어-수준 시뮬레이션(instruction-level simulation) 방식을 사용하였다.

3.2 Q+Sim의 구조와 기능

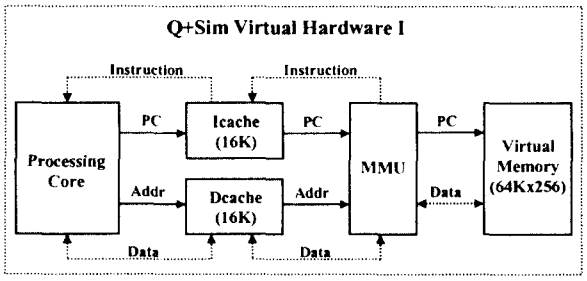


<그림 25> Q+Sim의 구조

Q+Sim은 전체 구조는 <그림 3>에서 보는 바와 같이 Q+ 에이전트(Q+ agent), 시뮬레이션 제어기(simulation controller), 그리고 가상 하드웨어(virtual hardware)와 Q+ 쉘(Q+ shell)로 구성되어 있다.

Q+ 에이전트는 타겟 서버와 TCP/IP 통신을 통해 Q+Sim과 Q+Esto의 인터페이스 역할을 하며, <그림 1>의 실제 타겟에서의 디버거 에이전트와 동일한 역할을 수행한다. 타겟 서버는 Q+ 에이전트를 재귀 IP 주소로 인식하기 때문에 Q+Sim을 실제 타겟으로 여기고 동작하게 된다. 이렇게 Q+ 에이전트가 재귀 IP 주소를 사용하는 방법은 Q+Sim을 위해 Q+Esto가 변경되는 부분이 없고, 단지 네트워크 환경 설정 부분에서 IP 주소만 변경하기만 된다는 이점이 있다.

시뮬레이션 제어기는 Q+ 에이전트와 가상 하드웨어간의 인터페이스 역할을 수행하며, Q+Sim 전체를 제어한다. 그리고, Q+Sim 쉘은 가상 타겟 쉘로서 실제 타겟의 표준 입출력으로 동작하며, Q+Sim의 실행창의 일부분으로 나타난다.



<그림 26> Q+Sim 가상 하드웨어 구조

EBSA285를 모델링한 가상 하드웨어는 위의 <그림 4>와 같이 처리기(Processing Core)와 가상 기억장치로 구성되어 있다. 처리기는 마이크로프로세서에 해당하는 부분으로, 32비트 레지스터 37개가 가상으로 변수로 구현되었고, 가상 기억장치는 16K바이트의 명령어 캐쉬(Icache), 16K바이트의 데이터 캐쉬(Dcache), 그리고 16M바이트의 메모리를 가상으로 구현한 부분으로, Q+Sim이 동작하는 시스템 자원의 효율적인 이용을 위해 파일을 이용한다. 특히 가상 메모리는 빠른 가상 메모리 엑세스를 수행하기 위하여 각 세그먼트 단위로 64K바이트 크기의 주소를 가진 256개의 파일로 나뉘어 구성되었다.

3.2 Q+Sim의 동작

Q+Sim을 사용하기 위해서는 Q+의 네트워크 설정을 재귀 IP 주소로 지정하여 Q+를 Q+Esto에서 타겟 환경으로 크로스 컴파일하여 목적 파일을 생성해야 한다. 그리고 Q+Sim을 실행시켜 생성된 Q+의 목적 파일을 지정하면, 본 연구에서 개발한 변환기가 이를 가상 메모리의 구조에 맞게 변환하여 가상 메모리 파일을 생성한다. 이후 실제 타겟에서 Q+가 초기화 작업을 0x100064 번지부터 수행하는 것과 같이 Q+Sim도 초기화 작업을 동일하게 수행한다. 이 작업이 끝나면 Q+ 에이전트가 구동되어 Q+Esto와 연결을 기다리는 상태가 된다. Q+Esto에서는 타겟을 재귀 IP 주소로 생성된 목적 파일로 지정하고, 타겟과 연결을 시도하면 Q+Sim과 연결된다.

3.3 명령어 시뮬레이션의 동작 원리

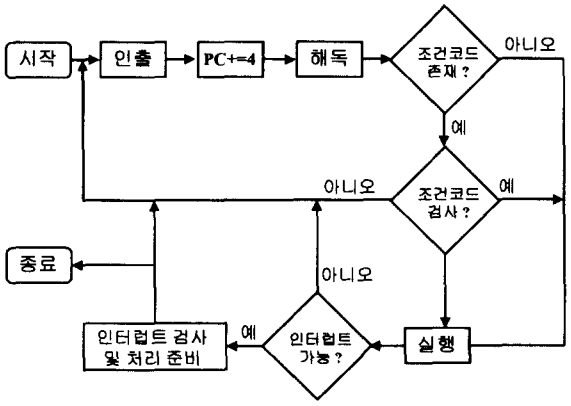
SA-110의 마이크로프로세서의 RISC 명령어는 인텔 계열의 CISC 명령어와 비교하여 연산코드(opcode)에 접미 코드가 붙는 특이한 점이 있다. 예를 들어 SA-110에는 'MOV'라는 기본 명령에

'EQ'라는 조건 코드가 추가된 'MOVEQ'라는 명령어가 있다. 이 명령어는 영(Z) 플래그가 0일 때만 실행되고, 그렇지 않으면 명령어를 실행하지 않고 다음 명령어로 넘어간다[11]. 이러한 명령어들을 시뮬레이션하기 위해서 Q+Sim의 가상 하드웨어는 <그림 5>와 같은 명령어 사이클로 명령어를 시뮬레이션 한다.

3.4 명령어 시뮬레이션의 예

본 연구에서는 Q+Sim의 명령어 시뮬레이션의 동작을 확인하기 위해 다음과 같은 함수를 타겟 환경으로 크로스 컴파일하여 생성된 목적 파일을 시뮬레이션 할 수 있는 간단한 시뮬레이터 프로그램을 작성하여 실험하였다.

```
int my_sum(int x, int y)
{
  return x+y;
}
```



<그림 27> 명령어 사이클

<표 10> my_sum의 가상메모리 내용

Addr	Data	Opcode	Operand
3818	e1a0c00d	MOV	r12,r13
381c	e92dd800	STMDB	r13!,{r11,r12,r14,pc}
3820	e24cb004	SUB	r11,r12,#4
3824	e24dd008	SUB	r13,r13,#8
3828	e50b0010	STR	r0,[r11,#-0x10]
382c	e50b1014	STR	r1,[r11,#-0x14]
3830	e51b3010	LDR	r3,[r11,#-0x10]
3834	e51b2014	LDR	r2,[r11,#-0x14]
3838	e0833002	ADD	r3,r3,r2
383c	e1a00003	MOV	r0,r3
3840	eafffffff	B	0x103844
3844	e91ba800	LDMDB	r11,{r11,r13,pc}

'my_sum' 함수를 타겟 환경으로 크로스 컴파일하여 목적 파일을 생성하고, 명령어 변환기로 변환하면 <표 1>와 같이 0x10의 세크먼트 주소를 갖는 가상 메모리 파일 '10.mem'이 생성된다. 시뮬레이터는 이 가상 메모리의 3818번지부터 명령을 읽어서 시뮬레이션을 수행하게 된다. 처리기로 읽혀진 연산코드와 오퍼랜드는 구문 분석기를 통해 해독되어 <표 2>와 같은 C 명령어로 변환되어 실행된다. 그리고, 메모리의 데이터 입출력은 'Mem_Read'와 'Mem_Write' 함수를 통해 가상 메모리의 'Data' 항목을 사용한다.

시뮬레이션을 수행한 후에 레지스터들과 기억장치의 내용들을 조사함으로써 시뮬레이터가 명령어를 정상적으로 시뮬레이션하고 있다는 것을 확인하였다.

<표 11> Q+Sim의 실행 코드

Addr	C Code
3818	r[12] = r[13];
381c	r[13] = r[13] - 4; Mem_Write(r[13], r[11]); r[13] = r[13] - 4; Mem_Write(r[13], r[12]); r[13] = r[13] - 4; Mem_Write(r[13], r[14]); r[13] = r[13] - 4; Mem_Write(r[13], r[15]);
3820	r[11] = r[12] - 4;
3824	r[13] = r[13] - 8;
3828	Mem_Write(r[11]-0x10, r[0]);
382c	Mem_Write(r[11]-0x14, r[1]);
3830	r[3] = Mem_Read(r[11]-0x10);
3834	r[2] = Mem_Read(r[11]-0x14);
3838	r[3] = r[3] + r[2];
383c	r[0] = r[3];
3840	r[15] = 0x103844;

3844	r[11] = Mem_Read(r[11]);
	r[11] = r[11] - 4;
	r[13] = Mem_Read(r[11]);
	r[11] = r[11] - 4;
	r[15] = Mem_Read(r[11]);
	r[11] = r[11] - 4;

4. 결론

본 연구에서는 실제 타겟에서 동작할 때의 실질적인 시간 추정이 가능하고 상용화가 가능한 명령어 기반의 실시간 운영체제 시뮬레이터를 설계하였으며, 간단한 시뮬레이션 프로그램을 시뮬레이션함으로서 명령어 시뮬레이션이 정상적으로 동작하는 것을 확인할 수 있었다. 가상 하드웨어 부분은 병렬처리나 분산처리와 같은 연구를 위한 시뮬레이션 작업에 응용이 가능하다.

본 연구가 실제 타겟의 동작을 완벽하게 시뮬레이션하기 위해서는 실제 하드웨어에서 실행에 영향을 주는 요소들도 시뮬레이션 할 필요가 있다. 이를 위해 인터럽트 처리나 캐쉬 처리, 그리고 버스 중재 등과 같은 요소들에 대한 시뮬레이션 기법이 추후 연구되어야 될 과제이다.

참고문헌

[1] 한국전자통신연구원, "조립형 실시간 OS 사용자 요구사항 정의서 1.0", 1998. 12.

[2] WindRiver, VxWorks 5.3.1 Programmer's Guide, 1997. 4.

[3] http://www.mentor.com/embedded, 2001. 1.

[4] Realiant Systems, "Carbon Kernel User Manual 1.2" 2000. 1.

[5] http://www.neiworld.co.kr

[6] http://www.aromasoft.com

[7] 한국전자통신연구원, "실시간 OS 커널 상세 설계서 1.0", 1999. 7.

[8] Intel, StrongARM EBSA-285 Evaluation Board Reference Manual, 1998. 10.

[9] Intel, SA-110 Microprocessor Technical Reference Manual, 2000. 12.

[10] Intel, 21285 Core Logic for SA-110 Microprocessor Datasheet, 1998. 9.

[11] ARM, ARM SDT 2.50 User Guide, 1998.