SoC 프로그램의 원격디버깅을 위한 실시간 추적도구

김영주,* 최석주,* 전인걸,** 전용기,* 임채덕**
*경상대학교, **한국전자통신연구소
{yjkim, sjchoi}@race.gsnu.ac.kr, igchun@etri.re.kr,
jun@gsnu.ac.kr, cdlim@etri.re.kr

A Real-Time Tracing Tool for Remote Debugging of SoC Programs

Young-Joo Kim,* Seok-Joo Choi,* In-Geol Chun,**
Yong-Kee Jun,* and Chea-Deok Lim**
*Gyeongsang National University, and **ETRI

요 약

임베디드 시스템에서 요구하는 SoC 프로그램을 개발하기 위해서는 자원이 풍부한 호스트 시스템에서 원격으로 디버깅할 수 있는 도구가 필요하다. 그러나 GDB를 이용하는 기존의 원격디버깅 도구는 SoC 프로그램의 수행 시에 정보를 실시간으로 제공하지 못하므로, 프로그램의 수행 양상을 실시간으로 감시하기 어렵다. 본 논문은 지정된 명령문의 수행시마다 SoC의 상태를 수행 중에 기록할 수 있는 실시간 추적도구를 소개한다. 그리고 본 도구가 PXA255 프로세서 기반의 타켓 시스템에서 합성 프로그램의 수행을 실시간으로 추적할 수 있음을 보인다.

1. 서론

임베디드 시스템은 산업용 제어장치에서 오래전부터 사용되어 왔으며, 마이크로프로세서의 소형화 및 집적화에 따라 점차로 고도의 기능을 요구하고 있다. 이런 요구조건에 적합한 SoC (System-on-a-Chip)[JuPu02, Pete99] 프로그램[KiKL03, Yagh03, RuCo01]을 개발하기 위해서는 자원이 풍부한 호스트 시스템에서 원격으로 디버깅할 수 있는 도구가 필요하다. 그러나 리눅스 환경에서 GDB를 이용하는 기존의 원격 디버깅 도구[Open04, Metr98, PiMa03]들은 수행중인 프로그램의 임의의 지점에서 레지스터나 메모리 정보를 얻기 위해서 프로그램에 대한 제어나 간섭을 유발하기 때문에, 시간적 제약 조건을 가지고 있는 SoC 프로그램을 디버깅하기에는 적절하지 못하다.

본 논문은 지정된 명령문의 수행시마다 SoC의 상태를 수행 중에 기록할 수 있는 실시간 추적도구인 EKDebugger를 제안한다. 이 도구는 추적점을 이용하여 SoC 프로그램의 원격 디버깅을 지원하는 기능과 디버깅된 상태 정보를 타켓 시스템으로부터 전송받아서 실시간으로 기록하는 기능을 가지고 있다. 제안된 도구는 호스트 시스템에 debug engine이 존재하고 이를 원격으로 지원해주는 태스크인 debug handler는 타켓 시스템에 존재한다. 본 논문은 이러한 도구의 실험을 위해서 PXA255 프로세서 기반의 타켓 시스템을 사용하고, 합성 프로그램의 수행을 실시간으로 추적할 수 있음을 보였다.

2절에서는 본 연구의 배경으로 SoC 프로그램을 위해서 GDB를 이용하는 기존의 원격 디버깅 도구를 살펴본다. 3절에서는 본 논문에서 제시하고자 하는 실시간 추적 도구를 기술한다. 4절에서는 구현된 도구의 실험환경과 실시간 원격 디버깅에 관한 실험에 대해서 설명한다. 마지막으로 5절에서는 결론 및 향후과제를 제시한다.

2. 연구배경

본 절에서는 SoC 프로그램의 디버깅을 위한 기법들을 살펴본다. 그리고 이들 기법 중에서 JTAG 인터페이스를 이용하여 리눅스 기반의 GDB를 이용하는 기존의 원격 디버깅 도구의 문제점을 살펴본다.

2.1 SoC 프로그램 디버깅

SoC 프로그램을 위해서 제한적인 시스템 자원을 가지는 타켓 시스템에서 수행되는 디버거를 이용하기는 힘들다. 이를 위해서 호스트 기반, On-Chip 기반, 원격 기반 등의 디버깅 기법들이 있다.

첫 번째, 호스트 기반의 디버깅 기법은 SoC[JuPu02, Pete99] 프로그램[KiKL03, Yagh03, RuCo01]이 수행될 타켓 시스템이 개발되지 않을 경우에 타켓 시스템의 명령어집합을 이용한 시뮬레이터를 사용하여 호스트 시스템에서 디버깅을 수행할 수 있게 하는 방법이다. 이러한 명령어집합 시뮬레이터는 개발이 용이한 반면에 프로그램 실행시간을 정확하게 예측할 수는 없다.

두 번째, On-Chip 기반의 디버깅 기법은 칩의 일부분으로 제공되는 디버그 커널을 이용하여 프로세서가 시스템의 다른 부분과 통신을 할 수 없더라도 지속적으로 실행제어를 호스트 시스템이나 전용 애뮬레이터 등으로 전송하여 시스템 자원을 감시할 수 있는 방법이다. 이러한 디버그 커널의 세가지 표준 인터페이스 프로토콜은 BDM(Background Debug Mode)[Moto99], IEEE 1140.1 JTAG(Joint Test Action Group)[Inte03ix], IEEE-50001 ISTO(Nexus)[Nath00]가 있다.

세 번째, 원격디버깅 기법은 타켓 시스템의 제한적인 자원으로 인해 타켓 시스템과 호스트 시스템으로 분산되어 있는 디버거를 이용하는 방법이다. 그리고 이들간의 통신은 시리얼이나 이더넷과 같은 통신채널을 통하게 된다. 타켓 시스템에 상주하는 디버거의 일부를 debug handler라고 하고, 호스트 시스템에 상주하는 디버거의 일부를 debug engine이라고 한다. debug handler와 debug engine간에는 원격디버깅을 위한 프로토콜[Rose96]이 존재하여 디버깅 명령 및 각종 정보를 교환한다.

2.2 GDB/JTAG 기반의 원격디버깅 도구

리눅스 환경에서 GDB를 기반으로 타켓 시스템의 JTAG 인터페이스를 이용하여 원격 디버깅을 지원하는 기존의 도구는 타켓 시스템과 호스트 시스템간의 정보교환을 위해서 타켓 시스템의 특성에 맞는 에뮬레이터를 이용하는 방법과 타켓과 호스트 시스템간의 신호변환만을 할 수 있는 어댑터를 이용하는 방법이 있다. 첫 번째, 에뮬레이터를 이용한 도구는 타켓 시스템에서 수행되는 프로그램을 에뮬레이터인 OPENice32-Axxx[Open04] 혹은 BDI2000[Metr98]를 이용하여 원격으로 디버깅할 수 있도록 지원된다. 두 번째, 어댑터를 이용한 도구는 Jelie[PiMa03]를 이용할 수 있다. 이 도구는 XScale 프로세서를 탑재한 타켓 시스템에서 JTAG 포트를 이용하여 디버깅할 수 있도록 지원하고, GDB 기반의 디버깅 명령어를 사용할 수 있도록 지원하며, 호스트 시스템의 Parallel/USB와 타켓 시스템의 JTAG을 제어하기 위한 기능 등이 있다.

이러한 도구에서 사용하는 GDB의 디버깅 기능 중에서 중지점(breakpoint)은 프로그램에 오류 가능성이 있는 지점에 대한 설정 기능, 프로그램 수행 시에 설정된 지점에 도달하면 프로그램의 수행을 중지시키고 사용자의 디버깅 관련 명령어를 입력받아

서 변수들의 값을 확인하거나 변경하는 기능, 그리고 그 이후에 프로그램의 수행을 계속하거나 중지하는 기능이 있다. 이러한 중지점은 프로그램의 논리적인 실행흐름과 변수들에 대한 메모리 값의 변화 등을 파악할 수 있기 없기 때문에 프로그램의 수행 양상을 실시간으로 감시하기 어렵다. 이에 반해 프로그램에 설정된 특정한 위치에서 프로그램에 대한 제어나 간섭이 없이 수행 정보를 실시간으로 기록할 수 있는 기능을 추적점(tracepoint)이라 하는데, 기존의 도구에서는 지원되지 않는다.

3. 실시간 추적도구

본 절에서는 GDB 기반으로 어댑터를 이용하는 도구인 EKDebugger를 위해서 추적점 엔진을 설계하였고, 이를 구현하기 위해서 Jelie[PiMa03] 소스레벨 디버깅 방법을 지원하는 중지점 기능을 응용하였다.

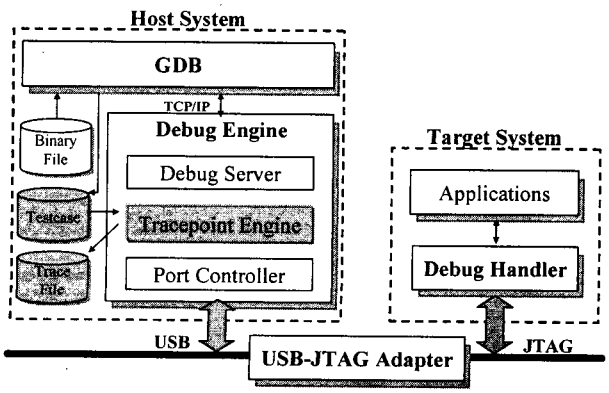
3.1 도구의 설계

본 논문에서 소개하는 실시간 추적도구인 EKDebugger는 리눅스 환경의 호스트 시스템과 XScale PXA255 CPU가 탑재된 타켓 시스템간의 통신을 위해서 USB-JTAG 신호변환 어댑터를 이용하는 도구로서, 그 구조는 [그림 1]과 같다.

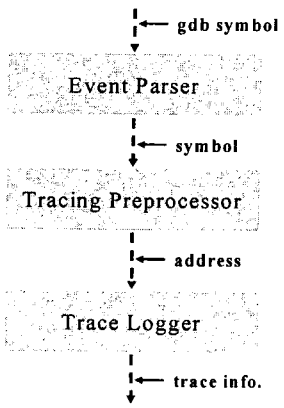
그림에서 호스트 시스템은 리눅스 환경에서 추적점 명령어를 인식하는 GDB와 그 명령어의 처리를 지원하는 Debug Engine으로 구성되어 있다. GDB는 디버깅 모드로 컴파일된 Binary File을 사용자로부터 입력받고, Binary File에 대해 감시해야 할 정보를 Testcase 파일로 생성한다. Debug Engine은 TCP/IP 기반의 RSP(Remote Serial Protocol)[Rose96]로 연결하며, 추적점에 관련된 명령어와 그 결과들을 송수신한다. Debug Engine는 Debug Server, Tracepoint Engine, Port Controller 등으로 구성되어 있다. Debug Server는 원격디버깅 명령어를 처리하는 모듈이고, Tracepoint Engine은 타켓 프로그램의 수행을 감시하거나 제어하는 모듈이며, Port Controller는 호스트 시스템의 USB 포트와 타켓 시스템의 JTAG 포트를 제어하는 모듈이다. Tracepoint Engine은 프로그램 수행중에 Testcase 파일을 이용하여 감시한 정보를 Trace File에 기록한다.

타켓 시스템은 Debug Handler와 Applications로 구성되어 있다. Debug Handler는 호스트 시스템에서 전송된 명령어를 분석하고, 수행된 프로그램의 상태정보를 호스트 시스템으로 전송한다. 그리고 Applications는 호스트 시스템에서 전송된 Binary File의 프로그램이다. USB-JTAG Adapter는 호스트 시스템의 USB 포트와 타켓 시스템의 JTAG 포트간의 통신을 위해서 신호를 변환하는 장비이다.

[그림 2]는 [그림 1]의 Tracepoint Engine 부분을 세 가지의 모듈로 나타낸 것이다. Event Parser는 GDB Shell로부터 디버깅에 관련된 명령어나 정보들을 심볼(symbol) 형태로 입력받아 분석하고 관련 명령들을 수행할 수 있는 모듈을 호출한다. 심볼 형태는 [표 1]에 명시되어 있는데, 추적점 기능을 수행하기 위해서 호스트 시스템의 GDB와 EKDebugger간에 정해진 프로토콜이다. Tracing Preprocessor는 [표 1]의 추적점에 관련된 심볼들을 이



[그림 1] EKDebugger의 구성



[그림 2] 추적점 엔진의 설계

용하여 프로그램 수행 중에 추적점을 동작시키기 위한 모듈이다. 이 모듈에서는 프로그램의 특정 위치에 설정된 추적점에서 실시간으로 기록할 변수나 레지스터에 대한 정보를 생성하고, 디버깅 모드를 추적을 하기 위한 모드로 변경하며, 프로그램의 수행중에 추적된 결과들을 실시간으로 기록하기 위한 파일을 생성한다. Trace Logger는 Tracing Preprocessor에서 생성한 추적점의 정보를 이용하여 프로그램의 수행중에 SoC의 상태나 프로그램의 수행정보를 파일에 기록한다.

3.2 도구의 구현

본 논문에서 소개하는 실시간 추적도구는 GDB와 기존의 EKDebugger에 추적점과 관련된 명령어와 처리루틴을 추가하였다. GDB에서는 중지점에 관련된 명령어들을 수정없이 사용할 수 있고, 추적점에 관련된 명령어들을 추가하였다. 추가된 명령어들은 추적점이 설정된 부분에서 Testcase의 생성, 추적점 기능으로 전환하기 위한 모드 설정/변경, 추적점 삭제/추가, 추적점 실행/정지 등에 관련된 것들이다. EKDebugger에서는 Jelie[PiMa03]의 소스레벨 디버깅 방법을 지원하는 중지점 기능을 응용하여 추적점 관련 명령어를 처리루틴을 추가하였다.

[그림 3]은 EKDebugger에서 심볼들을 처리하기 위한 함수인 parseValidPacket()를 보인 것이다. 이 함수는 Event Parser에 의해서 분리된 심볼들을 원격디버깅 명령에 대응시키기 위한 함수들로 구성되어 있다. 여기서 'Q'로 시작하는 심볼인 경우에는 "prepare_tracepoint()"를 호출하여 추적점에 관련한 초기화 작업인 추적파일 생성작업과 새롭게 설정된 추적점을 추가하는 작업 등을 수행한다. 'c'로 시작하는 심볼인 경우에는 "cont()"를 호출하여 프로그램 카운터(program counter)의 주소값이 프로그램에 설정된 중지점이나 추적점의 주소값인지 비교한다. 중지점과 추적점이 동일한 continue 명령어를 사용하나, 중지점과 추적점의 동작원리는 다르다. 추적점의 동작원리는 설정된 추적점의 메모리 값과 동일한 위치에서 SoC 프로그램의 메모리 값이 동일하면, 그 위치에서 프로그램의 수행을 멈추지 않고 실시간으로 변수나 메모리에 대한 정보를 기록하는 방식이다.

RSP Type	Symbols	Explanation
통신	+	sucess
	-	fail (error)
메모리	m	read memory
	M	write memory
레지스터	g	return the values
	G	set registers
추적점	QTDebugMode	set tracepoint
	QTInit	create tracelog
	QTf	set testcase
	QTDP	set tracepoint in tr struct

[표 1] tracepoint를 위한 RSP 명세서

```
void GdbRemote::parseValidPacket() {
 int startOfPacket = ptr;

 switch(current()) {
    case 'H': setThread(); break;
    case 'd': remote_debug = !remote_debug; break;
    case 'q': query(); break;
    ...
    case 'M': writeMem(); break;
    case 'c': cont(); break;
    case 'Q': prepare_tracepoint(); break;
    case 'T': debug_tracemode(); break;
    default: printf("command %s not nderstand\n", current());
             message[0] = 0;
    ... }}
```

[그림 3] parseValidPacket() 함수

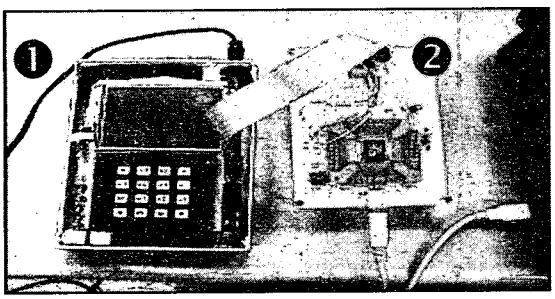
4. 도구의 실험

[그림 4]는 [그림 1]의 수행 환경이다. 그림에서 ①은 Palm 사에서 제작한 Tynux Box[Palm03]로서 XScale 기반의 PXA255 프로세서와 JTAG 포트를 사용할 수 있는 장비이다. ②는 Cypress사의 AN2131QC[Minf02] Chip을 사용하여 호스트 시스템의 USB 포트와 타켓 시스템의 JTAG 포트간의 신호를 서로 변환할 수 있는 통신 인터페이스이다.

Jelie에 추가된 추적점의 기능이 올바르게 동작하는지 확인하기 위해서 타켓 시스템의 LED를 ON시키는 프로그램을 이용하여 실험하였다. [그림 5]는 추적점의 실험을 위해서 사용한 명령어들을 나열한 명세서이다. [그림 6]은 명세서에 있는 명령어를 이용하여 실시간 추적도구를 수행하고 프로그램의 수행정보를 기록한 결과를 보인 것이다.

5. 결론

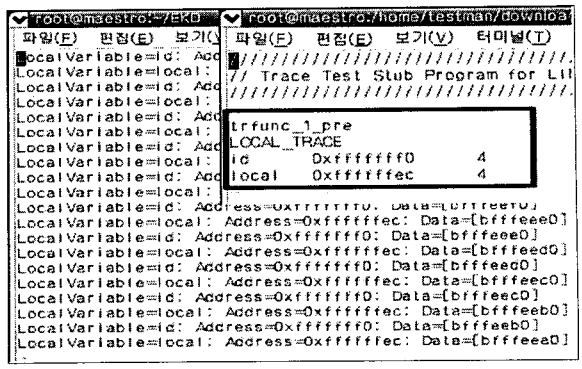
본 논문에서는 지정된 명령문의 수행시마다 SoC의 상태를 수행중에 기록할 수 있는 실시간 추적도구를 소개하였다. 이 도구는 추적점을 이용하여 프로그램의 수행정보를 실시간으로 추적 가능하므로 프로그램의 수행양상을 실시간으로 감시할 수 있다. 그리고 저렴한 디버깅 도구의 제작을 위해서 GDB를 사용하였으며, 기존의 GDB를 지원하는 그래픽 사용자 인터페이스를 수정없이 사용하게 하였다. 이러한 도구는 프로그램 수행정보를 이용



[그림 4] 실시간 추적도구의 수행환경

```
- GDB
    target remote localhost:3141; load sample1
    break main; trace 9
    action 1; collect $local; end
    trmake -c testcase; trsetmode TRACE
    tstart; break 10; continuce ; tstop
- EKDebugger
    ./EKDebugger -t 3141 -T &
```

[그림 5] 추적점의 실험명세서



[그림 6] 추적점의 수행결과

하여 프로그램 수행양상에 대한 시각화나 성능분석 분야에 적용이 가능하다. 향후과제는 추적도구의 성능 향상을 위해서 프로그램 카운터를 이용하지 않고 PXA255 프로세서에서 제공하는 trace buffer를 이용하여 SoC 프로그램을 위한 실시간 추적도구를 설계하는 것이다.

참고문헌

[Aiji04] AIJI System Co., *OPENice & Spider User's Manual*, 1122-10, Ingye-dong, Suwon-city, Korea, 2004.

[Inte03ix] Intel Co., *Intel XScale Microarchitecture for the PXA255 Processor User's Manual*, 2200 Mission College Blvd. Santa Clara, CA 95052-8119 USA, Mar. 2003.

[JoPu02] Johnson, M., and N. Puthuff, *Debugging Embedded SoC Systems*, RF Design, Feb. 2002.

[KiKL03] Kim, C., H. Kim, and C., Lim, Technology Trends and Development Strategies on Embedded Software for Ubiquitous Computing Era, *Telecommunications Review*, 13(1): 105-116, SKTelecom, 2003.

[LKLK03] Lee, K., J. Kim, C. Lim, and H. Kim, "A Development of Remote Tracepoint Debugger for Run-time Monitoring and Debugging of Timing Constraints on Qplus-P RTOS," *IEEE Workshop on Software Technologies for Future Embedded Systems*, pp. 93-96, Hakodate, Hokkaido, Japan, May 2003.

[Metr98] Metrowerks Inc., *CodeWarrior IDE Plugin Manual*, 9801 Metric, Suite #100 Austin, TX 78758 U.S.A., 1998.

[Minf02] Minford Technology Inc., *MF3001A EZ-USB AN2131QC Prototyping and Demo Board User's Manual*, Unit 86, 201 Alexmuir Blvd, Toronto Ontario, Canada, 2002.

[Moto99] Motorola Inc., *M68HC12B32EVB-User's Manual*, 5405 Denver, Colorado 80217, U.S.A., 1999.

[Nath00] Nath, N. M., On-chip Debugging Reaches a Nexus," EDN, May 2000.

[Palm03] Palm Inc., *Embedded Linux Development Kit Tynux Box X*, Hanyang Bldg 5F, 14-31 Youido-dong, Seoul, Korea, 2003.

[Pete99] Peters, K. H., "Software Development and Debug for System-On-a-Chip," *Embedded Systems Conference*, 1999.

[PiMa03] Pilet, J. and S. Magnenat, *Jelie: Manuel de L'utilisteur*, Ecole Polytechnique Federale De Lausanne Lap., 2003.

[Rose96] Rosenberg, J. B., *How Debuggers Work*, John Wiley & Sons, 1996.

[RuCo01] Rubini, A., and J. Corbert, *Linux Device Drivers*, 2nd Ed., O'Reilly & Associate, June 2001.

[Yagh03] Yaghmour, K., *Building Embedded Linux Systems*, O'Reilly & Associate, 2003.