SoC 프로그램의 원격 디버깅을 위한 실시간 추적도구의 구현

박 명 철[†] · 김 영 주[††] · 하 석 운[†††] · 전 용 기[††††] · 임 채 덕[†††††]

요 약

임베디드 시스템에서 요구하는 SoC 프로그램을 개발하기 위해서는 자원이 풍부한 호스트 시스템에서 원격으로 디버깅할 수 있는 도구가 필요하다. 그러나 GDB를 이용하는 기존의 원격 디버깅 도구는 SoC 프로그램의 수행 시에 정보를 실시간으로 제공하지 못하므로 프로그램의 수행 양상을 실시간으로 감시하기 어렵고, 도구에 제한적인 고가의 어댑터를 사용한다. 본 논문은 지정된 명령문의 수행시마다 SoC의 상태를 수행 중에 기록할 수 있는 실시간 추적도구를 소개하고, 원격 디버깅을 위한 경제적인 USB-JTAG 어댑터를 제안한다. 그리고 본 도구가 PXA255 프로세서 기반의 타겟 시스템에서 합성 프로그램의 수행을 실시간으로 추적할 수 있음을 보인다.

키워드 : SoC 프로그램, 디버깅, 실시간, 추적, JTAG, USB, GDB

Implementation of a Real-Time Tracing Tool for Remote Debugging of SoC Programs

Myeong-Chul Park[†] · Young-Joo Kim[††] · Seok-wun Ha[†††] · Yong-Kee Jun[††††] · Chae-Deok Lim[†††††]

ABSTRACT

To develop SoC program for embedded systems, a tool that can remotely debug from host system is needed. Because the existing remote debugging systems using GDB don't offer information of the SoC program execution in real-time, it is difficult to observe condition of the program execution, and also they have limited characteristics to tools and use costly adaptors. In this paper, a real-time tracking tool that can record SoC status on the fly each execution of the assigned instructions is introduced and an economical USB-JTAG adaptor is proposed. And it is shown that this tool can track the execution of a composed program in the target system based on PXA255 processor.

Key Words : SoC Program, Debugging, Real-time, Tracing, JTAG, USB, GDB

1. 서 론

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)[1, 2] 프로그램[3, 4, 5]을 개발하기 위해서는 자원이 풍부한 호스트 시스템에서 원격으로 디버깅할 수 있는 도구가 필요하다. 그러나 리눅스 환경에서 GDB를 이용하는 기존의 원격 디버깅 도구[6, 7]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. 또한, 원격 디버깅을 위한 필수 요소인 어댑터는 특정 도구에 제한적으로 동작하므로 도구간의 호환성 지원이 원활하지 못하였다.

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 타겟 시스템으로부터 전송받아서 실시간으로 기록하는 기능을 가지고 있다. 또한, 산업 표준화된 JTAG(Joint Test Action Group)[8, 9, 10]을 기반으로 원격지의 타겟 시스템에 접근하여 SoC 프로그램을 디버깅할 수 있는 경제적인 어댑터를 제안한다. 제안된 어댑터는 호스트 시스템의 USB(Universal Serial Bus)포트와 타겟 시스템의 JTAG을 연결하는 구조로

※ 본 논문은 한국전자통신연구원 2004년도 연구비 지원에 의하여 연구되었음.
† 정 회 원 : 경상대학교 대학원 컴퓨터과학과
†† 준 회 원 : 경상대학교 대학원 컴퓨터과학과
††† 종신회원 : 경상대학교 컴퓨터과학과 교수
†††† 종신회원 : 경상대학교 컴퓨터과학과 교수(교신저자)
††††† 정 회 원 : 한국전자통신연구원 임베디드SW연구단 팀장
논문접수 : 2005년 9월 12일, 심사완료 : 2005년 10월 26일

서 타겟 시스템에 영향을 주지 않고 고속의 USB포트를 사용하여 보다 경제적이고 효과적인 원격 디버깅을 가능하게 한다.

본 논문은 이러한 도구의 실험을 위해서 PXA255 프로세서 기반의 타겟 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 제시하는 실시간 추적 도구와 어댑터의 설계에 대해 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 살펴보고 이들 기법 중에서 JTAG 인터페이스를 통하여 리눅스 기반의 GDB를 이용하는 원격 디버깅 도구의 문제점을 살펴본다.

2.1 SoC 프로그램 디버깅

제한적인 시스템 자원을 가지는 타겟 시스템에서 SoC 프로그램 디버깅하는 것은 현실적으로 어렵다. 이런 문제점을 극복하기 위한 방안으로 호스트 기반, On-Chip 기반, 원격 기반 등의 다양한 디버깅 기법들이 이용된다.

첫 번째, 호스트 기반의 디버깅 기법은 SoC 프로그램이 수행될 타겟 시스템이 개발되지 않을 경우에 타겟 시스템의 명령어 집합을 이용한 시뮬레이터를 사용하여 호스트 시스템에서 디버깅을 수행할 수 있게 하는 방법이다. 이러한 명령어 집합 시뮬레이터는 개발이 용이한 반면에 프로그램 실행시간을 정확하게 예측할 수는 없는 문제점을 가진다.

두 번째, On-Chip 기반의 디버깅 기법은 칩의 일부분으로 제공되는 디버그 커널을 이용하여 프로세서가 시스템의 다른 부분과 통신할 수 없더라도 지속적으로 실행제어를 호스트 시스템이나 전용 애뮬레이터 등으로 전송하여 시스템 자원을 감시할 수 있는 방법이다. 이러한 디버그 커널의 세 가지 표준 인터페이스 프로토콜은 BDM(Background Debug Mode)[11], IEEE 1140.1 JTAG(Joint Test Action Group)[12], IEEE-50001 ISTO(Nexus)[13]가 있다.

세 번째, 원격 기반의 디버깅 기법은 타겟 시스템의 제한적인 자원으로 인해 타겟 시스템과 호스트 시스템으로 분산되어 있는 디버거를 이용하는 방법이다. 그리고 이들 간의 통신은 시리얼이나 이더넷과 같은 통신채널을 통하게 된다. 원격 디버깅의 일반적인 구성은 타겟 시스템에 상주하는 Debug Handler와 호스트 시스템에 상주하는 Debug Engine으로 구성된다. Debug Handler와 Debug Engine간에는 원격 디버깅을 위한 프로토콜[14]이 존재하여 디버깅 명령 및 각종 정보를 교환한다.

2.2 GDB/JTAG 기반의 원격 디버깅 도구의 문제

리눅스 환경의 GDB 기반에서 타겟 시스템의 JTAG를 이용한 기존의 원격 디버깅 도구는 크게 두 가지 방법으로 분류된다. 첫 번째는 타겟 시스템과 호스트 시스템간의 정보 교환을 위해서 타겟 시스템의 특성에 맞는 에뮬레이터를 이용하는 방법과 두 시스템간의 신호 변환만을 담당하는 어댑터를 이용하는 방법이 있다. 에뮬레이터를 이용한 도구는 타겟 시스템에서 수행되는 프로그램을 OPENice32-Axxx 나 BDI2000[6]등과 같은 에뮬레이터을 이용하여 원격 디버깅을 지원한다. 그리고 Jelie[7]와 같이 어댑터를 이용한 도구는 타겟 시스템에서 JTAG 포트를 이용하여 GDB 기반의 디버깅 명령어를 사용할 수 있도록 지원하며, 호스트 시스템의 Parallel/USB 인터페이스와타겟 시스템의 JTAG 포트를 제어하기 위한 기능 등을 가지고 있다.

이러한 에뮬레이터를 사용하는 기존의 도구들은 고가의 전용 에뮬레이터에서만 수행되므로 수행 환경이 제한적이라는 단점을 가진다. 그리고 어댑터를 사용하는 Jelie 도구는 호환성있는 저가의 어댑터가 없다는 문제점을 가진다.

대표적인 디버깅 기능으로 중지점(breakpoint) 기능과 추적점(tracepoint) 기능을 들 수 있는데, 기존의 도구들은 중지점 기능만을 갖는 문제점을 가지고 있다. 중지점 기능은 프로그램에 오류 가능성이 있는 지점에 중지점를 설정하여, 프로그램 수행 시에 설정된 중지점에 도달하면 프로그램의 수행을 중지시키고 사용자의 디버깅 관련 명령어를 입력받아서 변수들의 값을 확인하거나 변경하는 기능이다. 그러나 이러한 중지점 기능은 프로그램의 논리적인 실행 흐름과 변수들에 대한 메모리 값의 변화 등을 파악할 수 없기 때문에 프로그램의 수행 양상을 실시간으로 감시하기는 어렵다. 이에 반해 프로그램에 설정된 특정한 위치에서 프로그램에 대한 제어나 간섭이 없이 수행정보를 실시간으로 기록할 수 있는 기능을 추적점이라 하는데, 기존의 도구에서는 지원되지 않는다.

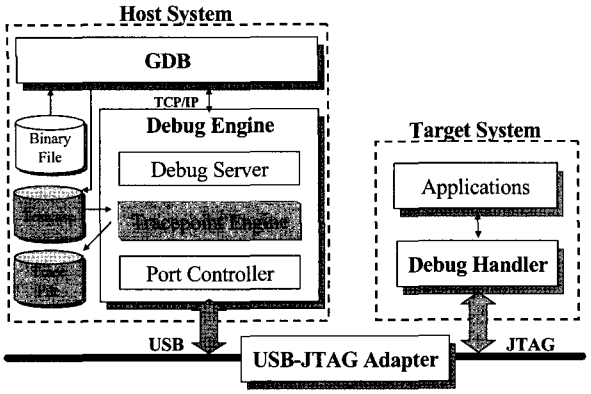
3. 실시간 추적도구

본 절에서는 어댑터를 이용한 GDB 기반의 디버깅 도구인 Jelie[7]를 이용한 추적점 엔진의 설계에 대해 기술하고, 타겟 시스템과 통신할 수 있는 경제적인 어댑터의 설계에 대해 기술한다.

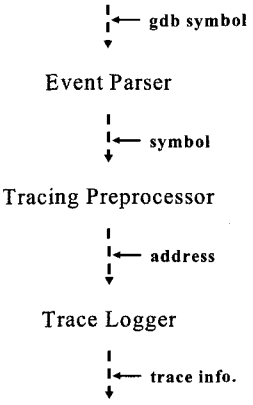
3.1 추적도구의 설계와 구현

본 논문에서 제안하는 실시간 추적도구인 EKDebugger는 GDB 기반의 호스트 시스템과 XScale PXA255 CPU가 탑재된 타겟 시스템을 대상으로 디버깅 환경을 제공한다. 두 시스템간의 통신을 위해서 USB-JTAG 어댑터를 이용하는데, 전체적인구조는 (그림 1)과 같다.

(그림 1)에서와 같이, 호스트 시스템은 리눅스 환경에서 추적점 명령어를 인식하는 GDB와 그 명령어의 처리를 지원하는 Debug Engine으로 구성되어 있다. GDB는 디버깅 모드로 컴파일된 Binary File을 사용자로 부터 입력받고, Binary File에 대해 감시해야 할 정보를 Testcase 파일로



(그림 1) 제안된 EKDebugger의 구성



(그림 2) 추적점 엔진의 설계

생성한다. Debug Engine은 TCP/IP 기반의 RSP(Remote Serial Protocol)[14]로 연결하며, 추적점에 관련된 명령어와 그 결과들을 송수신한다. Debug Engine는 크게 세 영역으로 구성되었다. 최상위의 Debug Server는 GDB로부터 전달된 원격 디버깅 명령어를 처리하는 모듈이며, Tracepoint Engine은 타겟 프로그램의 수행을 감시하거나 제어하는 모듈로서 프로그램 수행 중에 Testcase 파일을 이용하여 감시한 정보를 Trace File에 기록한다. 마지막으로 Port Controller는 호스트 시스템의 USB 포트와 타겟 시스템의 JTAG 포트를 제어하는 모듈이다.

타겟 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 타겟 시스템의 JTAG 포트간의 통신을 위해서 신호를 변환하는 장비이다.

(그림 2)는 추적점 기능의 핵심적인 영역인 Tracepoint Engine 부분을 세 가지의 세부 모듈별로 분류하여 나타낸 그림이다. 먼저, 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 수

〈표 1〉 tracepoint를 위한 RSP 명세서

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

행하기 위해서 호스트 시스템의 GDB와 EKDebugger간에 정해진 프로토콜이다.

Tracing Preprocessor는 <표 1>의 추적점에 관련된 심볼들을 이용하여 프로그램 수행 중에 추적점을 동작시키기 위한 모듈이다. 이 모듈에서는 프로그램의 특정 위치에 설정된 추적점에서 실시간으로 기록할 변수나 레지스터에 대한 정보를 생성하고, 디버깅 모드를 추적을위한 모드로 변경하며, 프로그램의 수행 중에 추적된 결과들을 실시간으로 기록하기 위한 파일을 생성한다. 마지막으로 Trace Logger는 Tracing Preprocessor에서 생성한 추적점의 정보를 이용하여 프로그램의 수행 중에 SoC의 상태나 프로그램의 수행정보를 파일에 기록한다.

본 논문에서 소개하는 실시간 추적도구인 EKDebugger는 기존의 GDB와 Jelie에 추적점을 위한 관련 명령어와 명령에 대한 실제 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 설정및 변경, 추적점 삭제와 추가, 추적점 실행및 정지 등에 관련된 명령들이다. Jelie에서는 소스레벨 디버깅 방법을 지원하는 중지점 기능을 응용하여 추적점 관련 명령어를 처리하는 루틴을 추가하였다.

(그림 3)은 제안된 도구에서 심볼들을 처리하기 위한 함수인 parseValidPacket()를 보인 것이다. 이 함수는 Event

```
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

Parser에 의해서 분리된 심볼들을 원격 디버깅 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.

3.2 어댑터의 설계와 구현

어댑터의 중심적인 역할을 담당하는 USB 칩셋은 EZ-USB로 널리 알려진 Cypress 사의AN2131QC[15]을 사용하였다. 어댑터는 크게 Signal Interface영역과 Fuctional interface영역으로 구분될 수 있는데, Signal Interface영역은 USB 포트에서 전달된 신호의 잡음으로부터 회로를 보호하고 5V 신호를 타겟 시스템의 JTAG에 적합한 3.3V 신호로 변환하는 역할을 한다.

Fuctional Interface영역은 USB 직렬 신호를 JTAG 포트단에 연결하기 위한 신호 핸들링 기능를 담당하는 역할을 한다. 이를 위하여 칩셋의 내부 메모리(8K)에 핸드쉐이킹을 위한 펌웨어를 두었다. 펌웨어는 C 코드로 작성하여 Hex code로 변환 후 도구의 초기화 루틴 수행 시 호스트의 디바이스 드라이브를 통해 내부 메모리에 다운로드 되게 된다. <표 2>는 펌웨어가 가져야 할 중요 모듈에 대한 요약이다. 물론, 사용자의 요구사항에 따라 추가 및 변경이 가능하다.

전원이 칩에 인가되게 되면, EZ-USB 코어는 I2C 포트에 접속하고 있는 EEPROM를 찾게 된다. 만일 EEPROM이 탐지되게 되고 0 번지의 내용이 '0xB0' 이면, EZ-USB 코어는 내부 기억 장치로 Vendor ID, Product ID, Device ID을 EEPROM에서 복사한다. EZ-USB 코어는 이 때 Get_Descriptor-Device 요구의 일부로서 호스트에게 이 정보들을 공급한다. 만일, 0 번지의 내용이 '0xB2'일 때는 7 번지부터 펌웨어가 있는 것으로 간주하고 내부 메모리에 다운로드하게 된다. 본 연구에서는 24LC00칩을 사용하여 해당 칩의 ID만을 제공한다. 호스트 시스템에서는 이 정보에 일치되는 드라이버가 OS에 의해 로드되게 되고, 드라이버가 EZ-USB의 RAM에 8051를 위한 펌웨어를 다운로드하고, 8051 코어가 동작을 개시하게 된다. 타겟 시스템에 전달된 JTAG신호는 해당 프로세서에 인식되기 위하여 Debug Handler에게 전달되게 되는데, 호스트 시스템으로부터 전송된 디버깅 명령를 감시하여 해당 명령을 처리하는 Moniter 모듈과 실질적인 디버깅을 처리하는 Command Handler 에게 전달되게 된다. 참고로 (그림 4)는 펌웨어 세부 모듈 중에서 USB 칩의 입출력 포트(B포트)를 초기화시키기 부분과 JTAG를 초기화 시키는 모듈을 C 코드로 보인 것이다.

<표 2> 펌웨어의 중요 모듈

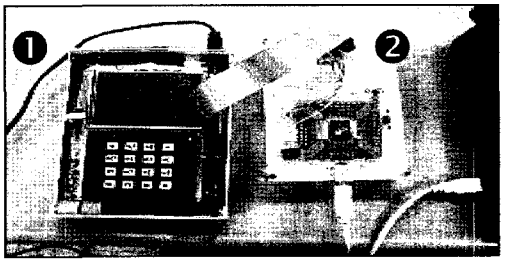
Module	Description of Function
initPorts	Initialize the I/O port of USB chip
jtagReset	Initialize the JTAG port
cpuReset	Initialize the Processor of Target System
ireg	Access of Instruction Register in JTAG
dreg	Access of Data Register in JTAG
jtagReg	Access of Register in JTAG

```
void initPorts(void) {
        OUTC=0x00;
        OEC=0xff;
        PORTCCFG=0x00;
        OUTB=0x00;
        OEB=JTAG_TMS|JTAG_CLK|JTAG_TDI|JTAG_PWR;
        PORTBCFG=0x00;
        OEA=0x00;
        PORTACFG=0x00;
}
void jtagReset(void) {
        unsigned char loop_count;
        OUTB=JTAG_PWR;
        OUTB=JTAG_PWR | JTAG_CLK;
        for(loop_count = 0; loop_count < 5; loop_count++) {
           OUTB=JTAG_PWR | JTAG_TMS;
           OUTB=JTAG_PWR | JTAG_TMS | JTAG_CLK;
        }
        OUTB=JTAG_PWR;
        OUTB=JTAG_PWR | JTAG_CLK;
        OUTB=JTAG_PWR;
        OUTB=JTAG_PWR | JTAG_CLK;
        OUTB=JTAG_PWR;
        jtagRestarted = 1;
}
```

(그림 4) initPorts() 와 jtagReset() 함수

4. 도구의 실험

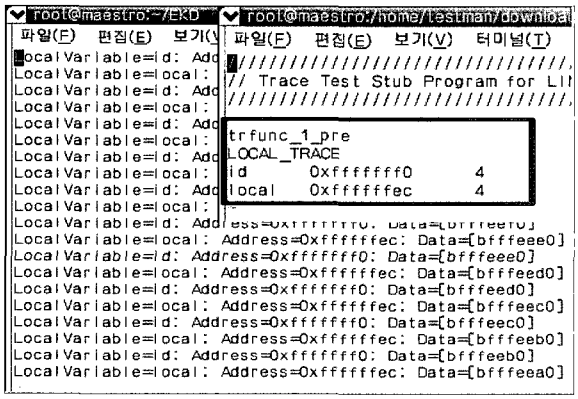
(그림 5)는 본 논문에서 제안하는 도구의 수행 환경이다. (그림 5)에서 ①은 실험에서 사용한 타겟 시스템으로 Palm 사에서 제작한 Tynux Box[16]로서 XScale 기반의 PXA255 프로세서와 JTAG 포트를 사용할 수 있는 장비이다. ②는 본 연구에서 구현한 어댑터로 Cypress사의 AN2131QC Chip을 사용하여 호스트 시스템의 USB 포트와 타겟 시스템의 JTAG 포트간의 신호를 서로 변환할 수 있는 통신 인터페이스이다.



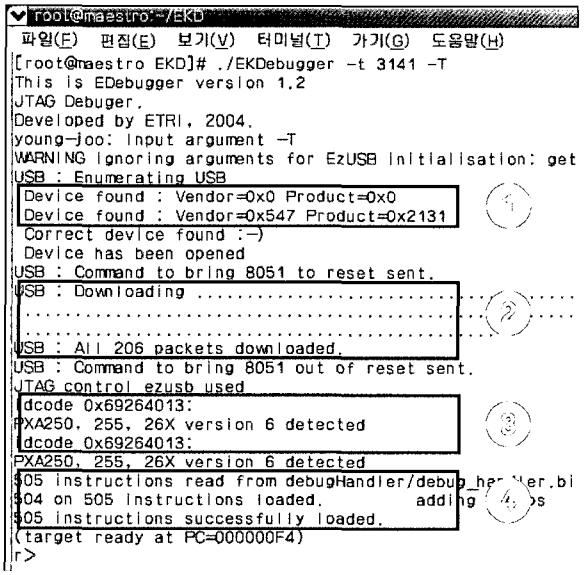
(그림 5) 실시간 추적도구의 수행환경

```
- 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 6) 추적점의 실험명세서



(그림 7) 추적점의 수행결과



(그림 8) 어댑터 동작 확인결과

(그림 6)은 제안된 추적점 기능이 올바르게 동작하는지 확인하기 위해서 사용한 명령어들을 나열한 명세서이다. 먼저, GDB에서 타겟 시스템을 대상으로 한 원격 디버깅 모드임을 알리고 테스트 모듈을 로드한다. 여기서 '3141'는 EKDebugger 와의 연결 포트 번호이다. 그리고 해당 모듈의 특정 지점에 중지점과 추적점을 설정하고, 해당 테스트케이스 파일을 생성한다. 다음으로, TRACE 모드로 변경한 후에 추적을 실시한다.(그림 7)은 (그림 6)의 명세서에 있는 명령어를 이용하여 실시간 추적도구를 수행하고 프로그램의 수행정보를 기록한 결과를 보인 것이다. (그림 7)의 우측 상단 파일이 GDB에서 생성된 테스트케이스 파일이다.

추적기능의 타입에는 네 가지가 있는데, 지역변수에 대한 LOCAL_TRACE와 전역변수에 대한 GLOBAL_TRACE, 그리고 레지스터에 대한 REGISTER_TRACE, 마지막으로 프로세스간의 통신 메세지와 관련한 MSG_TRACE이 있다. (그림 7)에서는지역 변수 id(0xffffffff0,4byte), local(0xfffffec, 4byte)에 대한 추적점 수행결과를 보이고 있다.

(그림 8)은 제안된 도구를 사용하여 어댑터의 동작 유무를 확인 할 수 있는 화면이다.①은 운영체제에서 USB-JTAG 어댑터를 정상적으로 발견하여 인식되었음을 표시하고 있고, ②은 어댑터의 내부메모리에 펌웨어를 다운로드 하는 부분이다. 그리고 ③은 해당 타겟 시스템의 JTAG으로 접근하여 IDCODE를 읽어온 후 타겟 시스템의 종류를 보이고 있다. 끝으로 ④에서는 타겟 시스템으로 Debug Handler가 정상적으로 다운로드 되었음을 표시한다. 최종적인 프롬프트 'r' 이 표시되므로서 디버깅을 위한 호스트 시스템과 타겟 시스템의 연결이 완료되었음을 알수 있다.

제안된 어댑터의 동작 속도를 측정하기 위하여 테스트 프로그램을 사용하여 측정한 결과를 <표 3>에서 보인다. 사용된 테스트 프로그램은 1부터 100까지의 합을 구하는 프로그램(Test1)과 타겟 시스템의 LED를 ON시키는 프로그램(Test2)를 사용하였다. <표 3>의 Test Item은 모두 명령어이며 Break, Continue, Step, Next에 있는 Variable은 변수에 중지점을 설정한 것이고, Loop은 반복문에 중지점을 설정한 것이다. 간단한 예로 특정 변수에 Break 명령으로 중지점을 설정한 경우 0.141s, 0.613s의 수행시간이 소요되었고,반복문에 중지점을 설정할 경우 0.171s, 0.164s의 수행시간이 소요되었음을 의미한다. Jelie에 이용되는 어댑터가 기존에 없는 관계로 비교 분석은 할 수 없지만, 호스트 기반의 디버깅 속도와 비교해 볼 때 타겟 시스템에서 수행되는 SoC프로그램을 원활히 디버깅할 수 있음을 증명하였다.

<표 3> 어댑터의 타임체크 결과

Test Item		Test Program	
		Test1 (35022byte)	Test2 (42646byte)
Break	Variable	0.141000	0.613000
	Loop	0.171000	0.164000
Continue	Variable	1.651753	1.807288
	Loop	1.657538	1.805799
Step	Variable	4.623500	3.484234
	Loop	6.743353	3.478684
Next	Variable	1.817398	1.975666
	Loop	6.679034	6.797380
Info Reg		0.405000	0.428000

5. 결 론

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

참 고 문 헌

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

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

[3] 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.

[4] Yaghmour, K., 'Building Embedded LinuxSystems,' O'Reilly & Associate, 2003.

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

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

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

[8] IEEE, 'IEEE Standard Test Access Port and Boundary- Scan Architecture,' Std 1149.1-1990, 1993.

[9] Asset InterTech, Inc., and R. G. Bennefits, 'Boundary-Scan Tutorial,' 2000.

[10] XJTAG Lt d., 'JTAG-A technical overview,' 2003.

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

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

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

[14] Rosenberg, J. B., 'How Debuggers Work,' John Wiley & Sons, 1996.

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

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

박 명 철

e-mail : africa@gsnu.ac.kr

1999년 한국방송통신대학교 컴퓨터과학과 (학사)

2002년 경상대학교 정보과학대학원 소프트웨어학과(석사)

2005년 경상대학교 컴퓨터과학과 박사수료

관심분야 : 영상처리, 시각화, 시뮬레이션, 임베디드 프로그램, 병렬프로그램 디버깅

김 영 주

e-mail : yjkim@race.gsnu.ac.kr

1999년 경상대학교 컴퓨터과학과(학사)

2001년 경상대학교 컴퓨터과학과(석사)

2003년 경상대학교 컴퓨터과학과 박사수료

관심분야 : 운영체제, 리눅스 클러스터링, 임베디드 시스템, 병렬프로그램 디버깅

전 용 기

e-mail : jun@gsnu.ac.kr

1980년 경북대학교 컴퓨터공학과(학사)

1982년 서울대학교 컴퓨터공학부(석사)

1993년 서울대학교 컴퓨터공학부(박사)

1982년~1985년 한국전자통신연구소 연구원

1985년~현재 경상대학교 컴퓨터과학과 교수, 컴퓨터정보통신연구소 연구원

관심분야 : 병렬/분산 컴퓨터딩, 임베디드 시스템, 시스템 소프트웨어

하 석 운

e-mail : swha@gsnu.ac.kr

1979년 부산대학교 전자공학과(학사)

1985년 부산대학교 전자공학과(석사)

1995년 부산대학교 전자공학과(박사)

1993년~현재 경상대학교 컴퓨터과학과 교수, 컴퓨터정보통신연구소 연구원

관심분야 : 디지털 신호처리, 신경망, 컴퓨터 비젼, 영상처리

임 채 덕

e-mail : cdlim@etri.re.kr

1989년 전남대학교 전산통계학과(학사)

1999년 충남대학교 전산학(학사)

2005년 충남대학교 전산학(박사)

1989~현재 한국전자통신연구원 임베디드 SW연구단 팀장

국산 임베디드 SW 통합 개발도구 Esto 시리즈 연구 개발

관심분야 : 임베디드 시스템 소프트웨어, 실시간 분산 컴퓨팅, 저전력 소프트웨어