DOI: 10.3745/KIPSTA.2010.17A.3.153

EISC 임베디드 시스템을 위한 USB-JTAG Interface기반의 디버깅 시스템 개발

이 호 균[†] · 한 영 선[††] · 김 선 욱[†††]

요 약

많은 개발자들은 프로세서 디버깅을 위해 GDB를 사용한다. 임베디드 시스템에서 GDB의 원격 디버깅은 시리얼 통신을 사용한다. 그러나, 시리얼 통신은 속도에 제한이 있으며, 시리얼 포트 마저 점차 사라져 가는 추세이다. 이를 극복하기 위해 많은 임베디드 시스템이 JTAG 인터페이스를 탑재하고 있으며, USB 인터페이스를 사용하여 통신을 한다. 이 논문에서는 EISC 아키텍처 기반의 임베디드 시스템을 디버깅하기 위한 USB-JTAG 인터페이스 개발 방법을 제안하고, GDB환경에서의 USB 인터페이스 구축 방법과 디버깅 패킷을 분석하기 위한 JTAG 모듈의 개발 방법을 소개한다.

키워드 : USB-JTAG 인터페이스, GNU 디버거, EISC 아키텍처

Debugging Environment Via USB-JTAG Interface for EISC Embedded System

Hokyoon Lee[†] · Youngsun Han[††] · Seon Wook Kim[†††]

ABSTRACT

Most of software developers use the GNU Debugger (GDB) in order to debug code execution. The GDB supports a remote debugging environment through serial communication. However, in embedded systems, the speed is limited in the serial communication. Due to this reason, the serial communication is rarely used for the debugging purpose. To solve this problem, many embedded systems adapt the JTAG and the USB interface. This paper proposes debugging environment via USB-JTAG interface to debug the EISC processor, and introduces how the USB interface works on the GDB and how the JTAG module handles debugging packets.

Keywords : USB-JTAG, GDB, EISC architecture

1. 서 론

현재 임베디드 시스템의 수요가 확산 되면서, 많은 개발자들은 편리한 디버깅 환경을 필요로 하고 있다. 특히 대부분의 임베디드 시스템이 JTAG[1, 7] 인터페이스를 탑재하면서 JTAG을 이용한 디버깅 기법[6]이 주목 받고 있다. EISC[2] 아키텍처 기반의 임베디드 시스템 역시 JTAG 인터페이스를 포함하고 있으며, 이 인터페이스는 각종 어플리케이션을 디버깅하기 위해 사용된다.

GDB[3]는 오픈 소스로써 폭넓게 사용되고 있는 GNU 디버거이다. 원격 디버깅을 위해 GDB는 시리얼 통신만을 지원하고 있으며, 이는 개발자들에게 많은 제약을 가한다. EISC 프로세서가 탑재된 임베디드 시스템은 USB-JTAG 인터페이스를 제공하기 때문에 GDB에서 USB를 이용한 통신이 가능해야 한다. GDB는 디버깅 과정에서 끊임없이 레지스터와 메모리의 값을 읽기 때문에, USB 인터페이스를 이용한 데이터의 전송은 개발자들에게 보다 빠른 디버깅 환경을 제공할 것이다.

이 논문은 USB-JTAG 인터페이스의 개발 방법을 제안하고, GDB와 JTAG 모듈에서 수행하는 패킷의 분석과 흐름에 대하여 설명한다. 2장에서는 이 논문에서 제안한 디버깅 프레임워크를 논의하고, 3장에서는 USB-JTAG 인터페이스의 구체적인 구현을 설명한다. 4장은 Cygwin 환경에서 실제

※ 본 논문은 산업자원부가 지원하는 국가 반도체연구개발사업인 시스템집적반도체기반기술개발사업(시스템IC2010)과 서울시 산학연협력사업(10920)의 지원 하에 개발된 결과임을 밝힙니다.
† 준 회 원 : 고려대학교 전자전기공학과 석박사통합과정
†† 정 회 원 : 삼성전자 시스템 LSI 책임 연구원
††† 종신회원 : 고려대학교 전기전자전파공학과 교수
논문접수 : 2010년 5월 31일
심사완료 : 2010년 6월 1일

USB-JTAG 인터페이스를 사용한 GDB 실행화면을 보여주고, 5장에서 이 논문의 결론을 맺는다.

2. Debugging Framework

(그림 1)은 USB-JTAG 인터페이스를 이용한 전체 디버깅 환경을 나타내고 있다. GDB 프로그램은 사용자가 입력한 명령을 패킷 형태로 변환하여 JTAG 모듈로 USB 인터페이스를 통해 전송한다. JTAG 모듈은 GDB로 부터 전송받은 패킷을 분석하여 TDI(Test Data Input) 패킷으로 변환하여 JTAG을 통해 EISC 타겟 보드로 전송한다. 디버깅 모드로 부팅된 타겟 보드는 JTAG 모듈의 명령을 기다리고 있는 상태이며, 명령이 전달 되면 그 명령을 수행한다. TDI 명령 수행 후 타겟 보드는 실행 결과를 TDO(Test Data Output) 패킷으로 구성하여 JTAG을 통해 JTAG 모듈로 전송하고, JTAG 모듈은 결과 데이터를 GDB가 분석 가능한 형태의 패킷으로 변경 후 USB를 통해 전송한다. 마지막으로, GDB는 패킷 분석 후 사용자가 내린 명령에 대한 결과를 화면으로 출력한다.

3. USB-JTAG Interface

이 장에서는 GDB에서 USB 디바이스를 사용하기 위해 필요한 USB 인터페이스의 구현과, EISC 프로세서의 JTAG TDI/TDO패킷을 설명하고, GDB 명령에 따른 JTAG 모듈의 TDI 패킷 구성을 서술 할 것이다.

3.1 USB Interface [5]

사용자가 입력한 디버깅 명령은 GDB 내에서 패킷 형태로 재구성 되며, 이 패킷을 정해진 통신방법을 통해 외부로 전송한다. USB-JTAG 인터페이스를 사용하기 위해, GDB는

〈표 1〉 USB I/O 함수

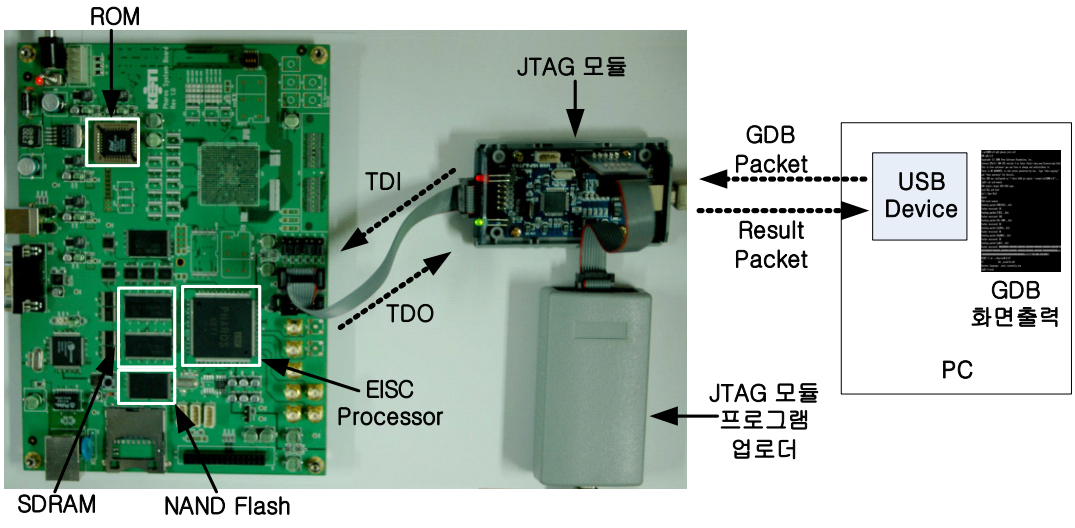
USB Function	Description
OpenDevice	USB 디바이스 사용을 시작한다.
Write	USB를 통해 데이터를 전송한다.
Read	USB를 통해 데이터를 수신한다.
Close	USB 디바이스를 중지한다.

USB 디바이스 라이브러리를 로딩한다. USB 디바이스 라이브러리는 DLL(Dynamic Link Library)로 구현되어 있으며, USB I/O 함수를 포함한다. <표 1>은 USB I/O DLL에 구현되어 있는 함수들을 설명하고 있다.

GDB는 EISC 프로세서와의 통신 방법을 USB-JTAG으로 설정 후, USB 디바이스 라이브러리를 로딩하여 OpenDevice 함수를 통해 USB connection을 수립한다. USB 를 통해 GDB와 EISC 프로세서 간에 통신을 할 수 있게 되면, GDB의 모든 패킷은 Read/Write함수를 통해 송/수신 된다. 명령에 대한 결과로써 수신된 패킷은 결과값만을 선택하여 화면에 출력한다. 마지막으로, GDB 종료 시에는 Close함수를 호출하고, EISC프로세서와의 연결을 종료한다.

(그림 2)는 USB-JTAG 인터페이스를 사용하기 위한 GDB 내에 구현된 함수의 일부를 보여준다. init_usb_ remote_ops 함수는 GDB의 원격 통신 방법을 USB로 설정하였을 때 실행되는 초기화 함수이다.

usb_remote 구조체는 GDB에서 기본적으로 제공하는원격 통신 구조체(remote_ops)의 함수와 USB 인터페이스를 사용하기 위한 함수를 포함한다. USB 인터페이스를 사용하여, EISC 프로세서와 통신을 연결하면 usb_remote_open 함수가 실행된다. usb_remote_open 함수는 우선 <표 1>에서 설명한 USB I/O 함수들과 GDB내에서 사용하는 함수를 연결한다. 즉, USB 함수인 OpenDevice는 GDB내에서 OpenDevice로, Write 함수는 WriteUSB, Read는 ReadUSB, Close는 CloseUSB로 사용된다. USB I/O 함수들을 wrapper 함수로



(그림 1) EISC 타겟 보드, JTAG 모듈 및 패킷의 흐름 [4]

```
static void init_usb_remote_ops(void)
{
    usb_remote = remote_ops;

    usb_remote.to_shortname = "usb-remote";
    usb_remote.to_longname = "AE32000 remote debugging via USB";
    usb_remote.to_doc = "Use a remote JTAG via an USB";

    usb_remote.to_open = usb_remote_open;
    usb_remote.to_close = usb_remote_close;
}
static void usb_remote_open(void)
{
  GetProcAddresses(&hLib,"ADCBulkUSB.dll",4,
      &OpenDevice,"OpenDevice",
      &WriteUSB,"Write",
      &ReadUSB,"Read",
      &CloseUSB,"Close");

  OpenDevice(USB_JTAG_UID);
}
static void usb_remote_close(void)
{
        CloseUSB();
}
static int usb_read_pkt(char **buf)
{
    int size;
    ReadUSB(*buf, strlen(*buf), &size, 0x5000000);
    return size;
}
static void usb_send_one_char(char *buf)
{
    int size;
    WriteUSB(buf, 1, &size, remote_timeout);
}
```

(그림 2) USB 인터페이스를 사용하기 위한 초기화 및 GDB wrapper함수

연결한 후, GDB는 OpenDevice함수를 호출하여 USB 디바이스 사용을 시작한다. GDB에서 패킷을 수신하기 위해 사용하는 함수는 usb_read_pkt이며, 이 함수는 ReadUSB 함수를 호출한다. 또한, GDB 명령에 따라 생성된 패킷은 WriteUSB 함수를 통해 송신한다. (그림 2)에서는 한 개의 문자를 송신 할 때 사용하는 함수 usb_send_one_char에서 WriteUSB 함수를 호출하는 것을 보여준다.

3.2 JTAG Packet Description

GDB 패킷은 명령의 종류에 따라 <표 2>의 패킷 시퀀스로 변환 되며, JTAG을 통해 타겟 보드로 그 명령을 전달한다. JTAG 모듈은 USB를 통해 GDB로부터 전송 받은 명령을 타겟 보드가 실행 할 수 있는 패킷으로 변환한다. <표 2>는 EISC 프로세서가 수행 할 수 있는 디버깅 명령과 그에 따른 JTAG 패킷 시퀀스를 보여준다. TDI는 JTAG의 입력이며, TDO는 출력이다. EISC 프로세서가 수행 할 수 있는 명령은 모두 다섯 가지이며, 각각 Break, Resume, Step Execution, Register Read, Register Write이다.

JTAG으로부터 Break TDI 입력(100101)을 받으면, EISC 프로세서는 instruction fetch를 수행하지 않고 대기 상태가 된다. 이 때, Resume TDI 입력(100111)을 받으면, breakpoint 레지스터의 값과 PC값이 같을 때까지 또는, Break TDI 입력을 받을 때까지 프로그램을 실행한다. Break 상태인 EISC 프로세서가 Step Execution TDI 입력(101001)을 받으면, 하나의 instruction을 fetch하고 실행한 후, 다시 대기 생태가 된다.

EISC프로세서는 JTAG을 통해 최대 10개의 연속된 레지스터의 값을 읽거나 쓸 수 있다. Register Read의 경우 TDI 패킷은 11100 / 레지스터 번호 / 읽어올 연속된 레지스터의 개수 / 레지스터의 개수X 32-zeros / 00001 로 구성되며, TDO의 경우 18-zeros / 레지스터의 데이터 / 0000으로 구성된다. Register Write의 경우 TDI 패킷은 11101 / 레지스터 번호 / 연속된 레지스터의 개수 / 레지스터의 데이터 / 00001로 구성되며, 레지스터 쓰기가 완료 되면, (22+레지스터 개수 X 32)-zeros를 TDO로써 전송한다.

〈표 2〉 EISC 프로세서 디버깅 명령과 JTAG 패킷

Processor Command	JTAG Module Packet Sequence
Break	TDI: 1 0 0 1 0 1 ← TDI TDO: 0 0 0 0 0 0 ← TDO
Resume	1 0 0 1 1 1 0 0 0 0 0 0
Step Execution	1 0 1 0 0 1 0 0 0 0 0 0
Register Read	1 1 1 0 0 | reg_addr[7:0] | beat[3:0] | 32 zeros ••• 32 zeros | 0 0 0 0 1 (beat) x 32 zeros 0 0 0 0 0 0 | 8 zeros | 4 zeros | $REG_{(adr)}$ data[31:0] ••• $REG_{(adr+beat-1)}$ data[31:0] | 0 0 0 0
Register Write	1 1 1 0 1 | reg_addr[7:0] | beat[3:0] | $REG_{(adr)}$ data[31:0] ••• $REG_{(adr+beat-1)}$ data[31:0] | 0 0 0 0 1 0 0 0 0 0 0 | 8 zeros | 4 zeros | 32 zeros ••• 32 zeros | 0 0 0 0 (beat) x 32 zeros

3.3 GDB Command and JTAG TDI Packet

<표 3>은 GDB 명령과 그에 따른 GDB 패킷 형태, 그리고, JTAG TDI 패킷을 나타내고 있다. GDB는 EISC 프로세서와 통신을 시작하기 위해 $\031#(*CRC*) 패킷을 JTAG 모듈로 전송하고, JTAG모듈은 Break TDI를 전송하여, EISC 프로세서가 대기 상태가 되도록 한다. CRC 는 16-bit 길이의 문자열로 표현되며, GDB패킷을 송/수신 할 때 패킷이 변경되는 에러를 확인하기 위해서 필요하다. GDB의 Continue와 Step Execution 명령의 경우, JTAG 모듈은 각각 Resume TDI와 Step Execution TDI를 EISC프로세서로 전송한다.

EISC 프로세서의 모든 레지스터의 값을 읽기 위하여, GDB는 $g#(*CRC*) 패킷을 JTAG 모듈로 전송한다. EISC 프로세서의 레지스터는 모두 27개이므로, JTAG 모듈은 Register Read TDI를 세 번 전송한다. 그 후, EISC 프로세서로부터 받은 세 개의TDO 패킷으로부터 레지스터 값을 추출한 후 27개의 레지스터 값을 GDB로 전송한다.

EISC 프로세서는 JTAG을 통해 메모리를 접근하기 위한 세 개의 특정 레지스터가 있다. R32는 디버깅을 위한 어드레스 레지스터이며, 메모리의 데이터는 R33을 통해 읽거나 쓸 수 있다. 또한, R34는 접근하는 메모리 데이터의 길이와 읽기/쓰기를 컨트롤하기 위한 레지스터이다. 4-byte 메모리를 읽기 위해서 세 개의 TDI가 필요하다. 첫 번째 TDI는 R32에 메모리의 어드레스를 쓰는 패킷이며, 두 번째 TDI는 R34에 컨트롤 데이터를 쓰는 패킷이다. 마지막으로, R33 의 값을 읽으면, 접근하고자 하는 어드레스의 값을 알 수 있다. 반면에, 메모리에 데이터를 쓴다면, R33에 그 데이터를 Register write TDI를 통해 쓰면 된다.

하드웨어 breakpoint와 watchpoint를 지원하기 위해, EISC 프로세서에는 8개의 레지스터를 지원한다. breakpoint 레지스터의 값은 Register write TDI를 통해 명령어의 주소 값 또는 데이터의 메모리 주소 값으로 설정 할 수 있다. Breakpoint

```
$ ae32000-elf-gdb pharos_test.elf
GNU gdb 6.8
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-cygwin --target=ae32000-elf"...
(gdb) tar usb-remote
GDB remote target USB-JTAG open
Load DLL and bind
Let's Open 0x16
Opend
USB-start-remote
Sending packet:$W031#19...Ack
Packet received: OK
Sending packet:$?#3f...Ack
Packet received: S05
Sending packet:$Hc-1#09...Ack
Packet received: OK
Sending packet:$qC#b4...Ack
Packet received: OK
Sending packet:$Hg0#df...Ack
Packet received: OK
Sending packet:$g#67...Ack
Packet received: 0000000001000000020000000300000004000000050000000600000007000
008000000090000000a0000000b0000000c0000000d0000000e0000000f000000000000000ecfe
f000000000000000000000000000000000000000de36ffff001000c040200043
START (0) at ../dev/crt0.S:17
17              ldi _stack-8,%r8
Current language:  auto; currently asm
(gdb) b main
```

(그림 3) GDB 실행 화면

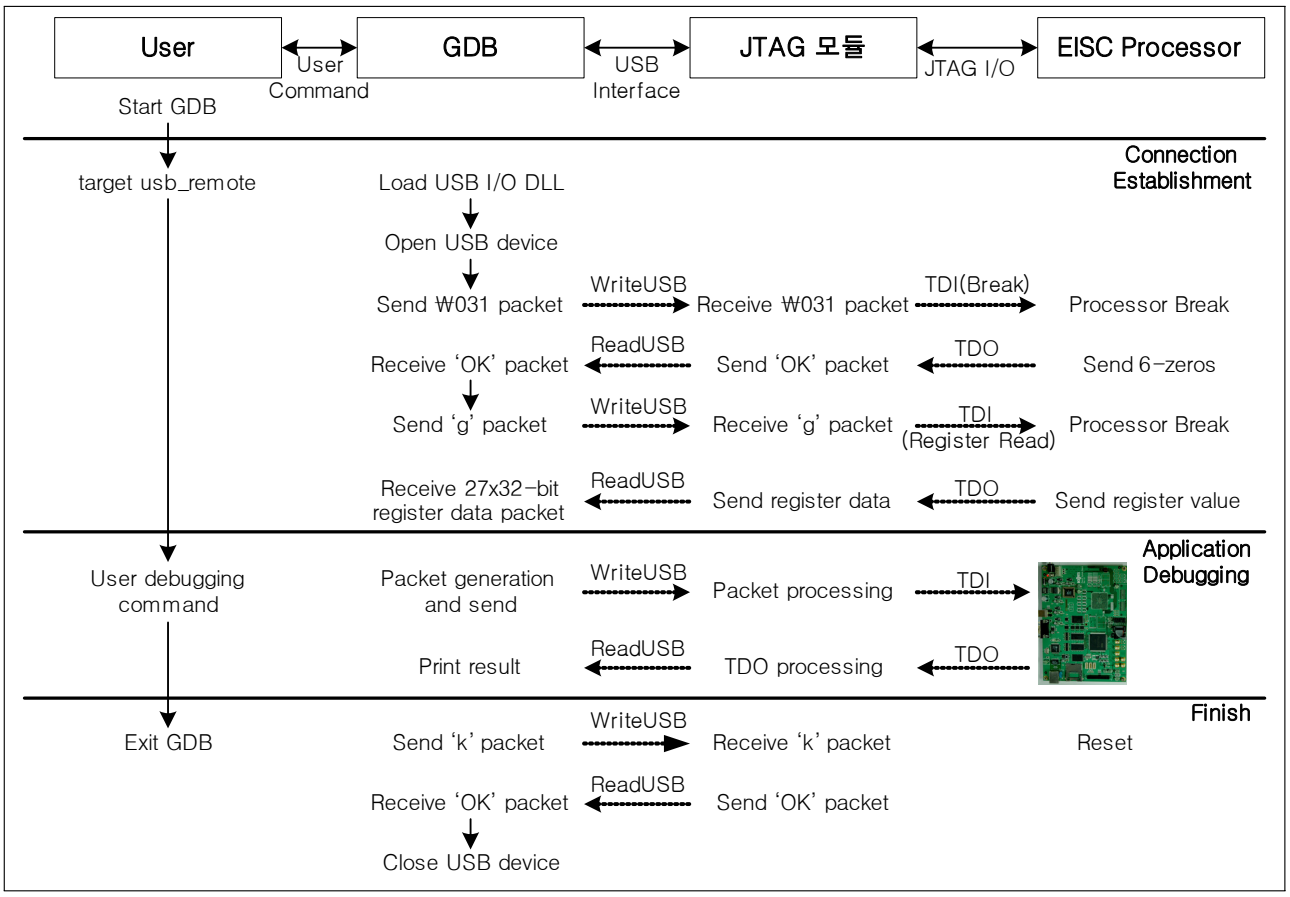
또는 watchpoint가 설정되어 있는 경우, EISC 프로세서는 breakpoint의 주소와 PC값이 같을 때 또는 watchpoint의 데이터가 변경되었을 때, break 상태가 되며 JTAG 모듈로 현재의 break 어드레스를 전송한다. GDB는 전송 받은 어드레스의 값을 토대로, 디버깅 중인 소스 파일의 위치를 가리키게 된다.

4. 구현 결과

(그림 4)는 디버깅 명령에 따른 GDB, JTAG 모듈, EISC 프로세서의 동작을 나타내고 있다. 그림 3은 그림 4의 동작을 기반으로 Cygwin 환경에서 GDB를 실행 후 EISC 시스템을 디버깅하는 화면이다. EISC 타겟 보드는 JTAG 디버깅 모드로 부팅되어 있으며, JTAG을 통해 전송 받은 명령

〈표 3〉 GDB 명령에 따른 GDB 패킷 형태와 JTAG TDI 패킷

GDB Command	GDB packet	JTAG TDI packet
Connection Establish	$\031#(*CRC*)	100101
Continue	$c#(*CRC*)	100111
Step Execute	$s#(*CRC*)	101001
All Register Read	$g#(*CRC*)	11100 / 00000000 / 1010 / 32 X 10 zeros / 0001 11100 / 00001010 / 1010 / 32 X 10 zeros / 0001 11100 / 00010100 / 0111 / 32 X 7 zeros / 0001
Register Write	$P(register num)=(data)#(*CRC*)	11101 / register num / 0001 / data / 0001
Memory Read	$m(addr),(length)#(*CRC*)	11101 / 0x20 / 0001 / addr / 0001 11101 / 0x22 / 0001 / 0x21(length=4) / 0001 11100 / 0x21 / 0001 / data / 0001
Memory Write	$X(addr),(length):(data)#(*CRC*)	11101 / 0x20 / 0001 / addr / 0001 11101 / 0x21 / 0001 / data / 0001 11101 / 0x22 / 0001 / 0x29(length=4) / 0001
Insert Breakpoint	$Z1,(addr),(length) #(*CRC*)	11101 / Breakpoint register num / 0001 / addr / 0001
Remove Breakpoint	$z1,(addr) ,(length) #(*CRC*)	11101 / Breakpoint register num / 0001 / 32 zeros / 0001
Insert Watchpoint	$Z2,(addr),(length) #(*CRC*)	11101 / Watchpoint register num / 0001 / addr / 0001
Remove Watchpoint	$z2,(addr) ,(length) #(*CRC*)	11101 / Watchpoint register num / 0001 / 32 zeros / 0001



(그림 4) 디버깅 명령에 따른 GDB, JTAG 모듈, EISC 프로세서의 동작

을 수행 할 준비가 되어 있다. 우선, 통신 방법 설정을 usb-remote로 설정한 것을 볼 수 있다. GDB는 USB I/O DLL을 로딩한 후 USB 디바이스를 Open하였다. 연결을 확립하기 위해, 기본적인 패킷을 전송한 후, 마지막으로 $g#67 패킷을 전송하여 EISC 프로세서의 모든 레지스터 값을 읽어 온 것을 확인 할 수 있다.

5. 결 론

시리얼 통신만을 지원하는 GDB는 사용상의 제약이 있으며, JTAG 모듈을 통한 디버깅이 불가능 하다. 이를 극복하기 위하여, 이 논문에서는 GDB에서 USB 통신을 지원하기 위한 개발 방법을 소개 하였으며, JTAG 모듈이 탑재된 임베디드 시스템을 디버깅 하기 위한 USB-JTAG 인터페이스를 제안하고, 개발하였다.

USB-JTAG 인터페이스를 사용한 디버깅 환경은 시리얼 통신을 사용하는 방법보다 더 빠르고 쉽게 이용 가능하다. 또한, JTAG 디버깅 모듈이 장착된 임베디드 시스템에서는 USB-JTAG 인터페이스를 이용한 디버깅 환경이 요구된다. 본 논문에서 제안한 개발 방법으로 GDB에서 USB 인터페이스를 제공하고, 패킷을 분석하는 JTAG 모듈을 사용한다면, 임베디드 시스템 개발자에게 보다 쉬운 디버깅 환경을 제공할 수 있을 것이다.

참 고 문 헌

[1] IEEE Std 1149.1, IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Computer Society, New York, 1993.

[2] EISC (Extendable Instruction Set Computer). http://www.adc.co.kr

[3] GDB. http://www.gnu.org/software/gdb

[4] 이호균, 김선욱. "USB-JTAG Interface를 이용한 EISC 프로세서 디버거 개발", 제 32회 한국정보처리학회 추계학술발표대회, 제16권, 제2호, pp. 47-48, 건국대학교, 2009년 11월.

[5] 전세일, 이두복. "USB 인터페이스를 이용한 데이터 전송프로그램 개발", 한국정보처리학회 논문지, v.7, no.5, pp. 1553-1558, 2000년 5월.

[6] 박명철, 김영주, 하석운, 전용기, 임채덕. "SoC 프로그램의 원격 디버깅을 위한 실시간 추적도구의 구현", 정보처리학회논문지 A, v. 12A, no. 7, pp.583-588, 2005년 12월.

[7] XJTAG Ltd., JTAG-A Technical Overview, March 2003.

이 호 균

e-mail : hokyoon79@korea.ac.kr

2006년 고려대학교 전기전자전파공학부(학사)

2006년~현 재 고려대학교 전자전기공학과 석박사통합과정

관심분야 : 컴파일러, 임베디드 시스템 등

한 영 선

e-mail : ysun.han@samsung.com

2003년 고려대학교 전기전자전파공학부(학사)

2009년 고려대학교 전자컴퓨터공학과(공학박사)

2009년~현 재 삼성전자 시스템 LSI 책임연구원

관심분야 : 컴파일러, 임베디드 시스템, 컴퓨터 구조

김 선 욱

e-mail : seon@korea.ac.kr

1988년 고려대학교 전자전산공학과(학사)

1990년 Ohio State Univ. 전기공학과(공학석사)

2001년 미국 퍼듀대학 전기컴퓨터공학과(공학박사)

2002년~2005년 고려대학교 전기전자전파공학과 조교수

2005년~2009년 고려대학교 전기전자전파공학과 부교수

2009년~현 재 고려대학교 전기전자전파공학과 교수

관심분야 : 컴파일러, 프로세서 및 SoC 디자인, 병렬처리, 성능평가 등