論文95-32A-7-15

HDL 모델 마이크로프로세서의 MS-DOS 호환성 검증 환경 구현

(The Environment for Verifying MS-DOS Compatibility of HDL modeled Microprocessor)

李 文 基 *, 李 政 燁 *, 金 盈 完 *, 徐 光 壽 *

(Moon Key Lee, Jeong Yop Lee, Young Wan Kim, and Kwang Soo Seo)

요 약

본 논문은 HDL (Hardware Description Language)로 기술된 마이크로프로세서가 MS-DOS와 호환성이 있는지를 검증할 수 있는 환경의 구축에 대하여 기술하였다. 여기서 MS-DOS 호환성이란 설계된 마이크로프로세서가 MS-DOS의 이진 코드를 변환 없이 그대로 수행할 수 있음을 의미한다. 제안된 검증 환경의 구성은 크게 HDL 시뮬레이터와 그래픽 사용자 인터페이스(GUI) 모듈로 나뉘어지며 둘간에 유닉스의 소켓 기능을 통하여 정보를 교환할 수 있도록 되어 있다. HDL 시뮬레이터는 PLI (Programming Language Interface)로 제작된 여러 함수들을 내장하고 있으며 이들 함수를 이용하여 ROM-BIOS 대체 루틴이 제작되었고 GUI 모듈은 PC의 모니터와 키보드 기능을 담당하도록 X/MOTIF를 이용하여 제작되었다. HDL로 기술된 마이크로프로세서를 구축된 환경으로 여러 MS-DOS 명령들 (DIR, FORMAT, DATE, TIME 등) 로 시뮬레이션한 결과 검증 환경이 올바로 동작되고 있음을 GUI 모듈에 의해 나타난 화면으로 확인하였다. 본 논문에서는 검증 환경 구축 방법을 기술하고 그 실험 결과를 나타내었다.

Abstract

This paper presents the simulation environment that verifies whether a new microprocessor described with HDL is compatible with MS-DOS. The phrase 'compatible with MS-DOS' means that the microprocessor can execute MS-DOS without any modification of MS-DOS's binary code. The proposed verification environment consists of HDL simulator and user interface module. And the communications between them are performed by using sockets which UNIX provide. The HDL simulator is equipped with several functions, which use PLI to emulate ROM-BIOS facilities. The ROM-BIOS emulation routine is described by using these functions. User interface module utilizes X/MOTIF and participates in emulating PC monitor and keyboard. The verification environment is tested by executing the MS-DOS commands (DIR, FORMAT, DATE, TIME etc.) with the HDL model of microprocessor, and the display of user interface module verifies that the environment works correctly. In this paper, the method of constructing the verification environment is presented, and the simulation results are summarized.

* 正會員, 延世大學校 電子工學科

(Dept. of Elec. Eng., Yonsei Univ.)

接受日字:1994年10月15日, 수정완료일:1995年6月26日

I. 서 론

HDL로 기술된 마이크로프로세서의 기능 검증을 위해서 사용되어 온 방법에는 각 명령에 해당하는 테스

트 벡터를 입력하여 검증하는 방법과 이를 좀더 발전시킨 것으로 조그만 크기의 테스트 프로그램을 실행시켜 올바른 결과가 나오는지를 보는 방법 등이 있었다[1][2]. 이러한 용도에 사용된 프로그램들에는 실제 응용 프로그램 내에서 많이 발견되는 루틴들이 포함되어 있다. 이와 같은 방법들은 마이크로프로세서의 규모가 크지 않았던 초기의 마이크로프로세서에 대한 검증 방법에 사용되었는데 마이크로프로세서의 규모가 커져감에 따라 하나의 테스트 프로그램이 검증할 수 있는 명령들이 전체에 대해 차지하는 비중은 상대적으로 적어져 경우에 따라 수천 ~ 수만 개의 테스트 프로그램이 사용되어야 하는 등의 어려움이 나타났다. 그러나 새로운 명령 체계를 갖는 마이크로프로세서에 대한 검증에는 이러한 방법이 계속 사용되고 있다. 이와는 달리 기존에 발표된 마이크로프로세서와 호환되는 명령 코드를 갖는 마이크로프로세서의 설계 시에는 위의 방법 외에 보다 발전된 형태로서 실제 사용되는 응용 프로그램 자체를 이용하여 검증하는 방법이 사용된다[3]~[6].

이 방법은 수많은 테스트 프로그램으로 검증하는 것에 비해 적은 수의 응용 프로그램으로도 기능 검증을 할 수 있다는 장점과 함께 실제 사용되는 응용 프로그램과의 호환성이 검증 된다는 점에서 최근 초집적 대규모 마이크로프로세서 설계에서 많이 사용되고 있는 검증 방법이다.

본 논문에서는 HDL로 기술된 마이크로프로세서의 기능 검증 및 MS-DOS와의 호환성 여부를 검증하기 위한 환경을 제시한다. 이 검증 환경은 MS-DOS를 수행시킬 수 있도록 마이크로프로세서의 주변 환경을 가상으로 구현해 놓은 것이다.

환경 구축을 위한 방법은 크게 세 가지가 있을 수 있다. 첫 번째 방법은 PC 내의 모든 디바이스들을 Behavioral 또는 RTL 수준에서 HDL로 모델링하고 ROM-BIOS 코드를 메모리에 담아 시뮬레이션하는 방법이다. 이 방법은 가장 원칙적인 방법이나 시뮬레이션 시간이 너무 길어지는 단점과 디바이스 모델들에 대한 기능 검증도 해야 하는 부담이 있다. 두 번째 방법은 시뮬레이션 시 마이크로프로세서가 외부로 전달하는 신호를 받아 실제 PC 시스템 보드에 전달하거나 시스템 보드에서 전달하는 신호를 받아 마이크로프로세서로 전해 주는 기능을 가진 인터페이스 모듈을 제작하여 HDL로 기술된 마이크로프로세서를 실제 PC 시스템에 맞물려 시뮬레이션하는 방법이다. 이 방법은 속도도 빠르고 실제와 거의 같은 결과를 보장할 수 있다는 장점이 있으나 PC 시스템과 마이크로프로세서간의 실행 속도의 차이를 해결해야 하며, 고가의 하드웨어 에뮬레이터를 이용해야 한다는 어려움이 있다. 세 번째 방법은 PC 내의 각종 디바이스와 ROM-BIOS의 기능을 대신할 수 있는 대체 프로그램을 만들고 이를 HDL 시뮬레이터와 연결시켜 시뮬레이션하는 방법이다. 이 방법은 설계한 마이크로프로세서를 HDL로 시뮬레이션함과 동시에 주변환경을 C 언어로 프로그래밍하여, 시뮬레이션 속도가 빠르고 검증 환경 개발 기간을 단축할 수 있다는 장점이 있다.

본 논문에서 채택한 방법은 세 번째 방법으로서 구현된 환경은 HDL 시뮬레이터를 축으로 하여 MS-DOS가 사용하는 ROM-BIOS 기능을 구현하여 대체시킨 부분과 PC의 모니터와 키보드를 가상적으로 구현한 부분으로 구성되어 있다.

본 논문은 4 장으로 구성되어 있는데 2장에서는 전체 검증 환경의 구성을 논하고 검증 환경의 각 부분에 대한 구현 방법을 설명하였으며 3장에서는 HDL로 기술된 마이크로프로세서 모델로 시뮬레이션 하는 과정과 결과를 그림을 통해 나타내었고 4장에서 본 검증 환경에 대한 결론을 내렸다.

II. 검증 환경의 구성

MS-DOS 호환성 검증을 위한 환경의 전체 구성은 크게 HDL 시뮬레이터와 GUI 모듈로 나누어지며 시뮬레이터와 GUI 모듈간의 정보 교환은 소켓을 통해 이루어진다. 검증 환경의 전체 구성을 (그림 1)에 나타내었다.

본 연구에서 사용한 HDL 시뮬레이터는 Verilog-HDL 시뮬레이터로서 두 부분이 하나로 합쳐져 존재하는데 하나는 기존의 Verilog-HDL 시뮬레이터 자체이며 또 하나는 본 연구에서 제안한 검증 환경 구축을 위해 필수적인 ROM-BIOS 기능을 대체해줄 부분으로서 Verilog-HDL 용 PLI(Programming Language Interface)를 이용하여 구현되었다.

GUI 모듈은 시뮬레이션중 모니터에 문자를 표시하거나 사용자가 입력한 킷값을 시뮬레이터에 보내는 기능을 담당하고 있으며 문자를 표시하는 등 그래픽과 관련된 작업을 하기 위해 X/MOTIF 라이브러리를 이용하여 제작되었다. 이때 두 모듈간의 정보는 유닉스에

서 제공하는 소켓을 통해 교환된다.

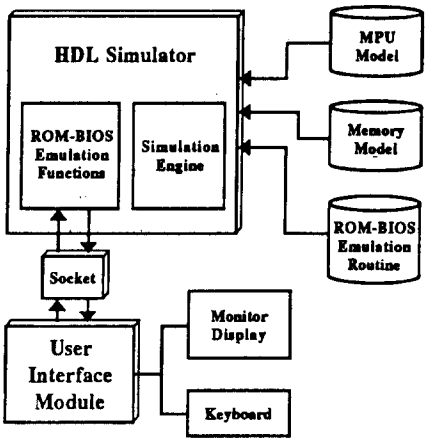


그림 1. 검증 환경의 전체 구성
Fig. 1. Verification Environment.

1. GUI 모듈

GUI 모듈은 독립적인 프로그램으로 동작하며 HDL 시뮬레이터가 보내 오는 비디오 화면, 레지스터값, 현재 수행 중인 명령어의 이름, 시뮬레이션의 진행 상황 등에 관한 정보를 소켓을 통해 받아 이를 X/MOTIF로 구현한 화면에 나타내거나, 사용자가 시뮬레이션 중에 입력한 킷값을 HDL 시뮬레이터로 보내게 된다. X/MOTIF는 서로 다른 호스트간의 그래픽 데이터 교환을 위해 만들어진 도구이다[7][8][9]. 본 논문에서는 GUI 모듈의 그래픽 관련 작업을 위해 이용 되었으며 제작된 GUI 모듈의 외형을 (그림 2)에 나타내었다.

이 모듈은 무한 루프를 돌고 있으며 매 루프마다 소켓에 정보가 있는지 검사하고, 정보가 있다면 그 종류에 따라 화면을 갱신하거나 킷값을 HDL 시뮬레이터 쪽으로 보내게 된다. 킷값은 사용자가 키보드를 누를 때마다 모듈 안의 키버퍼에 저장되며, HDL 시뮬레이터가 킷값을 요구하면 하나씩 버퍼에서 꺼내진 후 그 키에 해당하는 아스키(ASCII)와 스캔(SCAN) 코드로 변환되어 HDL 시뮬레이터 쪽으로 보내진다. 만약 키버퍼가 비어 있을 때 HDL 시뮬레이터가 킷값을 요구하게 되면 사용자가 키를 입력할 때까지 기다리게 되는데 이는 ROM-BIOS의 키보드 관련 기능 중 입력된 킷값을 키버퍼로부터 가져오는 기능이 키버퍼가 비었을 경우 사용자가 키를 입력할 때까지 기다리게 되어 있기 때문이다.

GUI 모듈은 (그림 2)에서 보이듯이 PC의 기본적인 동작에 필수적인 키보드와 모니터 외에 시뮬레이션 진행 과정 및 결과를 보다 쉽게 관찰할 수 있도록 하여 주는 창(Window)을 가지고 있다. 이는 부팅이 어느 정도 진행되었는지를 알려 주고, 현재 수행되고 있는 명령및 마이크로프로세서의 내부 레지스터들의 값 또한 나타내게 된다.

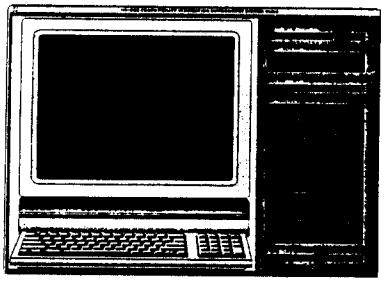


그림 2. GUI 모듈의 외형
Fig. 2. The Picture of GUI Module.

2. GUI 모듈과 HDL 시뮬레이터의 인터페이스

GUI 모듈은 Verilog-PLI를 이용하여 C 언어로 작성됨으로써 HDL 시뮬레이터와 하나로 묶여질 수 있다. 하지만 이러한 경우에 시뮬레이션과 GUI 모듈이 동시에 움직이게 되므로, 프로그램이 많은 메모리를 차지하게 되며 하나의 컴퓨터가 시뮬레이션과 그래픽의 디스플레이라는 두 가지 작업을 함께 하여 전체 시뮬레이션 속도가 상당히 저하되게 된다.

본 검증 환경에서는 이러한 속도 저하를 피하기 위해 GUI 모듈을 C 언어만을 사용하여 독립된 프로그램으로 동작하도록 작성하였고, 소켓을 이용한 통신기능을 양쪽에 두어 GUI 모듈과 HDL 시뮬레이터가 통신할 수 있도록 하였다. 따라서 두 개의 모듈이 각각 서로 다른 컴퓨터에서 작동함으로써 효율적인 메모리 사용 및 속도 향상을 가져올 수 있다.

GUI 모듈과 HDL 시뮬레이터 간의 통신은 소켓(Berkeley Socket)[10]을 이용하여 이루어지는데, 양 모듈이 인터페이스부를 가지고 있고 인터페이스부에선 C 언어의 소켓 라이브러리를 이용하여 서로 데이터를 교환할 수 있도록 하였다. GUI 모듈은 서버로 동작하며, 클라이언트인 HDL 시뮬레이터가 실행될 때까지 기다리게 된다. HDL 시뮬레이터가 실행되고 연

결이 완성되면 C 함수인 read()나 write()를 통하여 통신이 이루어지고, 두 모듈은 시뮬레이션이 끝날 때까지 서로 연결된 상태로 남아 있게 되는 것이다. 여기서 서버와 클라이언트의 구별은 소켓 연결의 편의상 구분일 뿐 그 이상의 의미를 가지지는 않는다.

통신을 위해 사용한 프로토콜은 아래의 (그림 3)과 같다.

Frame ID	Size of Data	CX	CY	Data

그림 3. GUI 모듈과 HDL 시뮬레이터 간의 통신 프로토콜

Fig. 3. The Protocols between GUI Module and HDL Simulator.

'Frame ID' 부분은 데이터의 종류 또는 요청이나 응답의 종류를 나타내게 된다. 각 'Frame ID'의 이름과 기능은 (표 1)과 같다. 이 'Frame ID'의 값에 따라 그 뒤를 따라오는 부분들에 대한 해석방법이 달라지는데 예를 들어 'CX'와 'CY'는 모니터상에 나타나는 커서의 위치를 나타내는데 쓰이는 용도 이외에 'Frame ID'가 F_BOOT_INFO인 경우에는 'CX'가 부팅 과정이 어느 정도 진행되고 있는지를 나타내는데 쓰이게 된다.

표 1. Frame ID 및 기능

Table 1. Value and Functionality of Frame ID.

이 름	값	기 능
F_NO	0	함수 호출에 대한 결과값 '0'
F_YES	1	함수 호출에 대한 결과값 '1'
F_QUIT	2	시뮬레이션의 종료
F_TEXT_IMAGE	3	비디오 메모리의 내용
F_KEYBOARD_HIT	4	키보드 입력이 있는 지 검사
F_KEYBOARD_GET	5	키보드 버퍼에서 입력된 키 값을 얻음
F_BOOT_INFO	6	부팅 과정에 대한 정보
F_INSTRUCTION	7	마이크로프로세서에서 수행되고 있는 명령
F_INFORMATION	8	현재 수행되는 일에 대한 정보
F_REGISTERS	9	내부 레지스터의 값

3. PLI (Programming Language Interface)

PLI는 Verilog 시뮬레이터를 만들 수 있는 기본 모듈과 기본 모듈에 새로운 기능을 추가시킬 수 있도록 C 언어 형식으로 제공되는 여러 가지 함수들로 구성되어 있다[11]

```
s_tfcell    veriusertfs[] = {
        { usertask, 0, 0, 0, _hello, 0, "$Hello" },
        { 0 }
};

void    _hello()
{
        printf ( "Hello\n" );
}
```

그림 4. '$Hello' 함수의 제작 예

Fig. 4. A Sample Program for a function '$Hello'.

표 2. PLI 로 구현된 함수들

Table 2. Functions implemented by using PLI.

분 야	함 수 명
디 스 크	$get_drive_parameter(drive, side, sector, track, err); $set_drive_parameter(drive, side, sector, track, err); $read_status(status); $clear_buffer(data); $read_sector(drive, side, sector, track, err); $write_sector(drive, side, sector, track, err); $name_disk(drive, drivename); $peek(count, data); $poke(count, data);
비디오 카드 (모니터)	$crt_name(crtname); $putch(c); $poke_imageb(offset, data); $poke_imagew(offset, data); $send_image(curx, cury);
키 보 드	$kbhit(hit, ascii, scan); $get_keyboard(ascii, scan);
기 타	$view(); $print_op(...); $print_reg(...); $connect(); $disconnect(); $send_reg(reg,value); $send_op(...); $send_info(info); $send_bootp(value);

즉, 기본적으로 Verilog 시뮬레이터에는 '$'로 시작되는 여러 가지 시스템 함수들이 정의되어 있는데 사용자가 새로운 시스템 함수를 만들려고 할 경우 이 Verilog-PLI에서 제공하는 라이브러리 함수들을 이용하여 제작할 수 있다. 예를 들어 화면에 'Hello'를 출력하는 '$Hello' 라는 함수를 만들고자 할 경우 (그림 4)와 같은 코드를 작성하여 컴파일한 후 기존 Verilog 시뮬레이터의 기본 모듈과 함께 링크하여 새로운 Verilog 시뮬레이터를 만들면 되는데 이 시뮬레이터를 사용할 때는 언제나 '$Hello' 함수를 사용할 수 있게 된다.

본 연구에서는 ROM-BIOS 기능을 대체시키기 위한 기본 함수들과 시뮬레이션의 편의를 위한 몇 가지

기능을 PLI로 작성하였으며 이를 (표 2)에 나타내었다.

4. ROM-BIOS 에뮬레이션 루틴

ROM-BIOS는 PC에 부착된 여러 디바이스에 대한 제어를 위해 만들어진 프로그램들과 PC의 동작 수행을 위해 반드시 필요한 기본 기능을 담당하는 프로그램들로 구성되어 있다[12][13][14]. 이 프로그램은 ROM 내에 위치하고 있으며 각 디바이스를 이용하는 응용 프로그램들은 MS-DOS를 포함하여 모두 이 ROM-BIOS가 제공하는 기능을 이용하게 된다.

MS-DOS가 시뮬레이션 환경 하에서 제대로 수행될 수 있도록 하기 위해서는 ROM-BIOS의 기능을 대체시킨 프로그램이 필요한데 이것이 ROM-BIOS 에뮬레이션 루틴이다. 이 루틴은 여러 가지 다양한 ROM-BIOS 기능들을 대체하게 되며 C 언어와 PLI로 구현된 기본 함수들을 사용하여 Verilog-HDL로 작성되었다.

인터럽트 형태로 제공되는 ROM-BIOS의 기능을 대체하기 위해 본 검증 환경에서는 'iret' 명령어만을 가진 가상 인터럽트 처리 루틴을 메모리 상에 위치 시켜 놓고 초기의 인터럽트 벡터에 이 가상 인터럽트 처리 루틴이 위치한 주소 값을 입력시키는 방법을 택했다. 이 방법을 사용했을 경우 프로그램 수행시 인터럽트 명령을 만나게 되면 마이크로프로세서는 인터럽트 벡터를 참조하여 가상 인터럽트 처리 루틴이 위치한 곳으로 수행을 옮기게 된다. 이 주소의 메모리에는 'iret' 명령이 위치해 있으므로 마이크로프로세서가 이 명령을 가져와 해석하게 되고 이 때 마이크로프로세서는 특정 변수를 1로 만들게 된다. 이 변수는 현재 수행하고 있는 명령이 'iret' 명령임을 나타내게 되며 이 때 특정 변수를 1로 만드는 기능은 마이크로프로세서를 HDL로 기술하는 과정에서 특별한 기술이나 노력없이 쉽게 추가될 수 있다.

HDL 시뮬레이터는 이 변수가 1로 변했음을 감지한 후 현재 수행하고 있는 주소 값이 가상 인터럽트 처리 루틴의 위치와 일치한지 검사한 후 일치 하다고 판단되면 시뮬레이션 시간이 정지한 상태에서 HDL로 작성된 ROM-BIOS 대체 프로그램이 수행되도록 한다. 대체 프로그램은 담당하는 기능에 따라 순수한 HDL 코드만으로 구현되는 경우와 PLI로 작성된 함수들을 이용하여 구현되는 경우가 있으며 시뮬레이션 시간을 정지시키는 문제는 대체 프로그램 내에 어떠한 시간 지연 문장도 넣지 않음으로써 해결할 수 있다.

시뮬레이션 시간이 정지된 상태에서 대체 프로그램의 수행이 끝나면 다시 시뮬레이션이 계속되어 'iret' 명령의 수행이 계속되고 이 명령에 의해 스택에 저장된 주소와 상태 레지스터가 복원되어 인터럽트 명령 다음의 명령이 위치한 곳으로 수행을 옮겨 시뮬레이션이 계속된다.

ROM-BIOS 에뮬레이션이 진행되는 과정을 (그림 5)에 나타내었다.

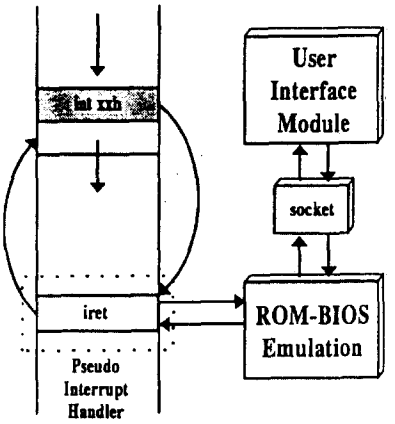


그림 5. ROM-BIOS 에뮬레이션 진행 과정
Fig. 5. ROM-BIOS Emulation Flow.

Ⅲ. 검증 환경을 이용한 시뮬레이션 및 고찰

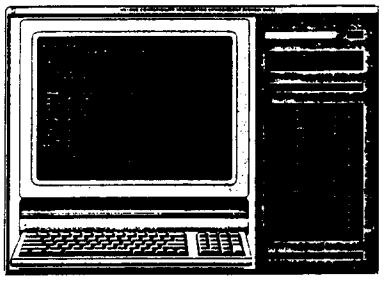
실제 본 검증 환경이 올바로 동작하는지를 확인하기 위해 HDL로 기술된 마이크로프로세서를 본 검증 환경에 넣어 다음과 같은 순서로 시뮬레이션을 진행하였다.

(1) MS-DOS가 담긴 디스켓을 하나의 화일로 만들어 저장한다.
- MS-DOS의 버전은 3.3, 5.0, 6.0 등을 사용하였고 디스켓의 용량은 1.2M및 1.44M 바이트중에서 선택할 수 있도록 하였다.

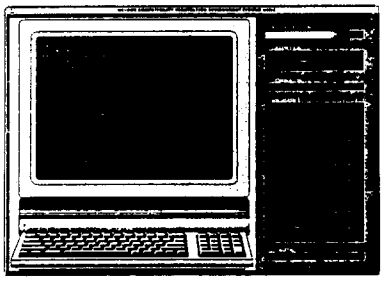
(2) 여러 변수들을 적절하게 지정한다.
- HDL 시뮬레이터가 동작할 호스트의 IP (Internet Protocol) 주소, (1)번 과정에서 생성된 화일의 이름, 디스크 드라이브의 용량, 시뮬레이션 파형의 표시 여부등

(3) MS-DOS의 부팅과정을 시뮬레이션 한다.
(4) MS-DOS 내부 명령어 동작을 시뮬레이션 한다.
(5) MS-DOS 외부 명령어 동작을 시뮬레이션 한다.
(6) PC 상에서 컴파일 된 프로그램들을 검증 환경 상에서 수행시켜 본다.

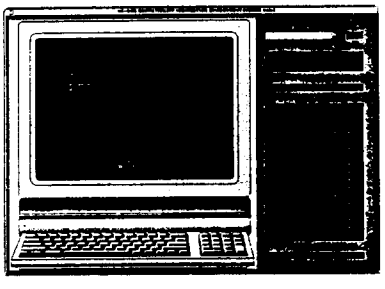
위 단계에서 (3), (4), (5), (6)번 과정을 진행하는데 사용한 MS-DOS 명령들 및 테스트 프로그램들을 (표 3)에 나타내었다. 이중 (4), (5), (6)번 과정을 진행한 화면중 몇가지를 선택하여 (그림 6)-a,b,c에 각각 나타내었다.



(a)



(b)



(c)

그림 6. 'TETRIS' 프로그램의 수행 화면
Fig. 6. The picture of executing 'CHKDSK'.

그림 왼쪽 부분에 GUI 모듈에 의해 구현된 가상 모니터와 키보드가 나타나있다. 오른쪽 아랫부분에 사용자가 누른 키보드를 표시하는 부분이 있는데 이것이 키버퍼를 나타낸다. 그림에서 보이듯이 'DIR' 명령, 'CHKDSK' 명령의 수행 결과와 'TETRIS' 게임의 실행화면등이 실제 PC에서의 모양과 일치함을 알 수 있다. 그림 오른쪽 부분에는 현재 수행하고 있는 코드및 레지스터값들이 나타나있고 MS-DOS의 부팅과정이 처음 시작때부터 'A:\>'가 표시될 때까지를 전체 범위로 하여 막대 그래프로 표시되어 있다.

표 3. 시뮬레이션에 사용된 MS-DOS 명령및 테스트 프로그램
Table 3. MS-DOS commands and test programs executed in simulation.

과 정	MS-DOS 명령	테스트 프로그램
(3)	'TIME', 'DATE' 'RAMDRIVE.SYS'	
(4)	'DIR', 'DEL', 'REN', 'COPY', 'CLS', 'PROMPT', 'MKDIR', 'CHDIR', 'RMDIR', 'TIME', 'DATE'	
(5)	'CHKDSK', 'FORMAT', 'MEM', 'DEBUG', 'LABEL', 'TREE', 'MORE', 'SYS'	
(6)		'TETRIS', '오목', 'HANGMAN', 'HANOI', 'QSORT', 'BASEBALL'

표 4. 시뮬레이션의 수행 시간
Table 4. Time consumption for simulations.

시 간 (초) / 구 분	GUI 모듈 수행 시간		HDL 시뮬레이터 수행 시간		실제 수행 시간
	user	system	user	system	
부팅 과정	1474	2728	4198	8	8408
'DIR'	1172	2754	3884	18	7828
'CHKDSK'	1796	4290	6022	18	12126
'TETRIS'	4446	10618	14904	44	30012
'HANGMAN'	452	2020	2432	8	4912

시뮬레이션은 두대의 SUN4/370 워크스테이션이 사용되었는데 각 호스트에는 GUI 모듈과 HDL 시뮬레이터가 각각 수행되었으며 GUI 모듈의 화면은 HDL 시뮬레이터가 동작하고 있는 호스트쪽에 함께 보

여지도록 하였다. 시뮬레이션은 CPU를 기준으로 약 4~5 ips(instruction per second)의 속도로 이루어졌고 UNIX의 'time'명령을 사용하여 계산된 시뮬레이션 수행 시간을 (표 4)에 정리하였다.

본 논문에서 제작한 검증 환경은 기본적으로 MS-DOS 만을 수행시키기 위한 것이므로 MS-DOS에서 수행되는 응용 프로그램이라 할지라도 MS-DOS의 기능을 이용하지 않은 채 하드웨어를 직접 엑세스하거나 하드웨어 인터럽트를 이용한다면 올바르게 수행시킬 수 없다. 하지만 본 논문에서 구현한 검증 환경에는 인터럽트 발생시 이를 감지하여 시뮬레이션을 일시 중단시키는 기능이 이미 구현되어 있으므로 프로그램이 이용하는 하드웨어 인터럽트에 대한 ROM-BIOS 에뮬레이션 루틴을 구현하고 하드웨어 인터럽트를 CPU에 전달할 수 있는 인터럽트 콘트롤러를 HDL로 구현한다면 하드웨어 인터럽트를 이용하는 프로그램들도 검증환경 내에서 수행시킬 수 있게 된다. 이때 발생할 수 있는 문제는 타이머 인터럽트의 구현에 따른 전체 시뮬레이션 시간의 증가이다. PC 시스템은 기본적으로 1/18.2 초마다 한 번의 타이머 인터럽트가 발생하므로 시뮬레이션시에도 프로그램 수행 중간 중간에 타이머 인터럽트에 대한 처리를 해주어야 하며 이는 결국 전체 시뮬레이션 시간의 증가로 나타나게 되고 타이머 인터럽트와 무관한 프로그램들에 대해서는 필요없는 수행을 계속 반복하는 것이 된다.

아울러 현재 구현된 검증 환경으로서도 마이크로프로세서가 처리하는 다양한 명령들에 대한 기능 검증에는 문제가 없다. 이는 외부 인터럽트 발생시 이를 감지하는 로직이 마이크로프로세서의 명령을 해석하여 수행하는 로직과 서로 구분되기 때문이다. 따라서 기본 명령들에 대한 동작 검증을 이 검증 환경을 통해 수행한 후 각 하드웨어 인터럽트에 대한 반응은 따로 검증하는 방법을 사용한다면 검증 환경을 개선하지 않더라도 마이크로프로세서의 동작여부를 확인 할 수 있다. 본 논문에서는 이러한 고려하에 하드웨어 인터럽트에 대한 처리를 구현하지 않았다.

Ⅳ. 결 론

본 논문에서는 설계된 마이크로프로세서의 MS-DOS 호환성을 검증할 수 있는 검증 환경에 대하여 기술하였다.

검증 환경은 마이크로프로세서를 시뮬레이션하는 HDL 시뮬레이터 모듈과 사용자 인터페이스를 위한 GUI 모듈, 이 두 모듈을 인터페이스하기 위하여 PLI로 구성한 인터페이스 모듈로 이루어진다. 시뮬레이션 속도를 향상시키기 위하여 HDL 시뮬레이터와 GUI 모듈은 서로 다른 컴퓨터에서 수행되었으며, 인터페이스 모듈이 두 모듈의 통신을 맡아 하나의 프로그램으로 동작할 수 있도록 하였다.

검증 환경은 MS-DOS가 수행될 수 있는 환경을 만족시켜 줌으로써 마이크로프로세서의 MS-DOS 호환성 검증뿐만 아니라 설계된 마이크로프로세서의 기능 검증에도 활용할 수 있다. 기능 검증을 위한 Hangman, Tetris 등 다수의 테스트 프로그램을 MS-DOS에서 사용되는 컴파일러들을 이용하여 수행코드를 생성한 뒤 이 프로그램이 담긴 디스켓을 화일로 만듦으로써 각 테스트 프로그램을 검증 환경을 통해 수행시킬 수 있었다.

구현한 MS-DOS 호환성 검증 환경이 올바로 동작됨을 보이기 위하여 HDL로 기술된 마이크로프로세서를 이용하여 시뮬레이션 하였으며, 실행 결과 초당 4~5 인스트럭션을 처리하는 속도를 얻었다. 또한 호환성 검증 환경에서의 시뮬레이션 결과는 실제 PC에서의 실행 결과와 동일함을 확인함으로써 검증 환경이 올바로 동작함을 알 수 있었다.

참 고 문 헌

[1] Moon Key Lee, "32bit YONSEI SPARK RISC Microprocessor", *Journal of The Research Institute of ASIC Design*, Vol.1 No.1, 1994.

[2] Raj Jain, The Art of Computer Systems Performance Analysis, John Wiley & Sons, 1988.

[3] M. Gupta and B. Zivkov, "A Structured Verification Approach for MIPS Microprocessors : A Case Study of the R4200," digest of papers COMPCON94, Feb 28 - Mar 4, 1994.

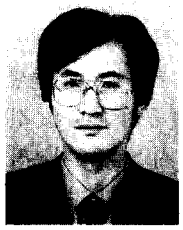
[4] 이문기, VLSI 기술을 이용한 RISC 마이크로프로세서 개발에 관한 연구, 상공부 중간 보고서(Ⅰ), 연세대학교 ASIC 설계 공동 연구소, 1991

[5] 이문기, VLSI 기술을 이용한 RISC 마이크로프로세서 개발에 관한 연구, 상공부 중간 보고서(II), 연세대학교 ASIC 설계 공동 연구소, 1992
[6] 장훈, "메모리 관리 유닛/캐쉬 제어기 평가 시스템 설계 및 제작", 연세대학교 대학원 전자공학과 석사학위 논문, 1994
[7] Marshall Brain, MOTIF Programming : The Essentials ... and More, Digital Press, 1992.
[8] Dan Heller, MOTIF Programming Manual for OSF/MOTIF Version 1.1, O'Reilly & Associates, 1991.
[9] Adrian Nye, Xlib Programming Manual for Version 11, O'Reilly & Associates, 1992.
[10] W. Richard Stevens, UNIX Network Programming, Prentice-Hall, 1991.
[11] Cadence, Verilog Programming Language Interface, Cadence, 1994.
[12] Michael Tisher, PC Intern System Programming, Abacus, 1992.
[13] Ray Duncan, Advanced MS-DOS, Microsoft Press, 1986.
[14] 한 성 국, IBM-PC 기술 사전, 집문당, 1989

저 자 소 개

李 文 基(正會員)

1941년 8월 23일생. 1967년 연세대학교 전기공학과 강사. 1969년 광운공과대학교 전자공학과 전임 강사. 1970년 경희대학교 전자공학과 조교수. 1973년 경희대학교 전자공학과 부교수겸 학과장. 1976년 University of OKLAHOMA 연구소 연구원. 1980년 한국전자기술 연구소(현 ETRI) 설계자동화(CAD) 국책연구과제 수행 책임 연구원. 1983년 금성 반도체 주식회사 비상임 기술 고문. 1984년 기아 산업 주식회사 종합연구소 비상임 기술 고문. 1989년 연세대학교 부설 아식설계 공동연구소 부소장. 1990년 연세대학교 전자공학과 학과장. 현재 연세대학교 교수, 연세대학교 부설 아식설계 공동연구소 소장, 대한 전자공학회 회장. 주관심 분야는 마이크로프로세서 설계, 디지탈 영상 처리 칩 설계, 설계자동화 등

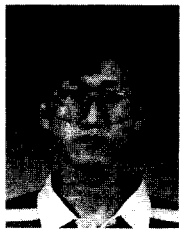


徐 光 壽(正會員)

1960년 8월 21일생. 1983년 2월 경희대학교 전자공학과 졸업. 1985년 3월 연세대학원 전자공학과 졸업(공학석사). 1985년 - 1992년 2월 LG 반도체(주) 선임연구원. 현재 연세대학교 전자공학과 박사 과정 재학중. 주관심 분야는 상위수준합성, 데이타 압축, 모듈생성 및 Smart 메모리 아키텍처등

李 政 燁(準會員)

1970년 5월 19일생. 1993년 2월 연세대학교 전자공학과 졸업. 1995년 2월 연세대학원 전자공학과 졸업(공학석사). 현재 C & S Technology (주) 재직. 주관심 분야는 마이크로프로세서 설계, 설계자동화 등



金 盈 完(準會員)

1971년 2월 21일생. 1994년 2월 연세대학교 전자공학과 졸업. 현재 연세대학원 전자공학과 석사 과정. 주관심 분야는 설계자동화 등