논문 2005-42SD-8-4

LDM 원격 제어를 위한 임베디드 시스템 구성

(Designing a Embedded System for Remote Control of LDM)

문 철 홍*, 강 상 우**

(Cheol-Hong Moon and Sang-Woo Kang)

요 약

본 논문에서는 LDM(LED Dot Matrix) 원격 제어를 위해 임베디드 시스템의 FPGA H/W 및 S/W를 구현하였다. 개발된 시스템에서는 고성능의 XScale CPU를 사용했고, 통신방식은 이더넷 및 시리얼을 사용했다. CPU와 인터페이스 되는 H/W에서는 LDM 회전부와 LDM을 구동하기 위한 FPGA 로직을 구현하였으며, 원거리에서 LDM 데이터를 전송하기 위한 명령 패킷을 구성하였고, S/W는 임베디드 리눅스에 사용되는 리눅스 디바이스 드라이버와 리눅스 응용프로그램을 작성했다. 이 S/W는 모듈에 의해 동작되기 때문에 리눅스용 파일시스템에 모듈로 적재를 시켜서 원하고자 하는 동작을 실행한다. 또한 운영체제로는 시스템의 최적화를 할 수 있는 임베디드 리눅스를 시스템에 맞게 컴파일 함으로서 불필요한 메모리를 사용하지 않기 때문에 시스템의 가격을 줄일 수 있다. 본 논문에서 구현된 H/W 및 S/W 원리를 이용한다면 다른 임베디드 시스템에도 유용하게 활용할 수 있다.

Abstract

In this paper, FPGA H/W and S/W Embedded system for LDM remote control is implemented. XScale CPU is used on developed system, and in communcation ethenet and serial is used. CPU interface with H/W LDM rotation and to drive LDM FPGA logic is implemented, to transmit LDM data from long distance command packet is composed, for S/W Embedded linux is used to design linux device driver and linux application program. This S/W is run by module so by loading this module to linux file system it can do any movement. Also by compiling Embedded linux to the system it can lower the price of the system. By using this h/w and s/w theory it can be used on any other embedded system.

Keywords: Linux, Embedded System, Device Driver, Remote Control

I. 서 론

임베디드 시스템으로 8bit, 16bit, 32bit 프로세서가 사용되고, 데이터의 처리용량의 증가에 따라 32비트 코어에서 64비트 코어로 발전해나가고 있다. 특히 기능면에서 고성능의 파이프라인, DSP, 자바 처리 전용 하드웨어가 추가되며, 많은 종류의 마이크로프로세서/컨트롤러들 중에서 응용에 최적인 제품을 찾아서 설계를 하고, 메모리는 고속/대용량화되고 있으며, 사용이 증가되고 있다.[1] 또한, 네트워크 분야에서는 유·무선 네트워크의 대역폭이 계속적으로 증가되고 있고 정보기기를 통해 어디서나 다른 정보 시스템에 접근 가능한 네트워크 구축이 핵심기술이다.[1] 그러나 임베디드 시스템은 시스템마다 각각 다른 디바이스 드라이버를 사용하기 때문에 상용화를 위해서는 호환 가능한 디바이스 드라이버와 응용 프로그램을 만들어야 한다. 본 논문에서는 여러 임베디드 시스템에서도 사용할 수 있는 디바이스 드라이버를 만들기 위해서 LDM과 스텝모터를 구동할 수 있는 디바이스 드라이버를 구현해 보았으며, 또한 임베디드 시스템과 디바이스 드라이버를 네트워크를 통한 원격 제어를 할 수 있도록 구현하였다. 그림 1은 임베디드 시스템부와 임베디드 소프트웨어부로 구성되는 전체 시스템을 나타내고 있다. 본 논문에서 임베디드 시스템부는 LDM 모듈을 회전시키기 위한 스텝 모터부와 LDM을 구동시키기 위한 FPGA[2] 모듈부로 구성되

* 정회원, ** 학생회원, 광주대학교 전자공학과
(Division of Electronic Engineering, Gwang-Ju Univ.)

※ 본 연구는 산업자원부의 지역혁신 인력양성 사업의 연구 결과로 수행되었음.

접수일자: 2004년12월6일 수정완료일: 2005년7월21일

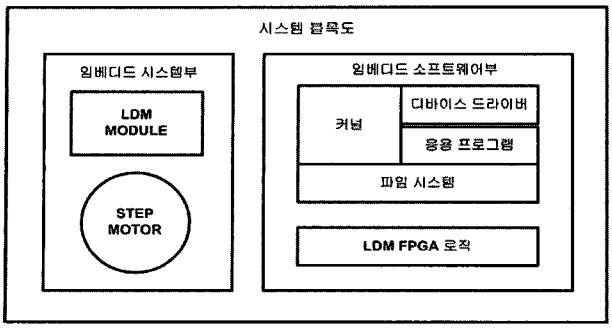

그림 1. 시스템 블록도
Fig. 1. System block diagram.

어져 있다. 임베디드 소프트웨어부는 부트로더, 임베디드 리눅스 커널 버전2.4.19[3], 파일시스템[4], 디바이스 드라이버[9], 디바이스 응용프로그램을 컴파일해서 시스템에 적재한다.

Ⅱ. 임베디드 하드웨어

1. 시스템 전체 블록

그림 2는 시스템 전체 구성도를 나타내고 있다. Host PC에서 이더넷 또는 시리얼을 통해 시스템에 커널, 파일시스템. 디바이스 드라이버, 디바이스 응용프로그램을 다운로드 하고, 시리얼 에뮬레이터를 통해서 LDM과 스텝 모터를 동작시킨다. 그리고 동작 과정을 Host PC에서 확인하도록 구현하였다.[5]

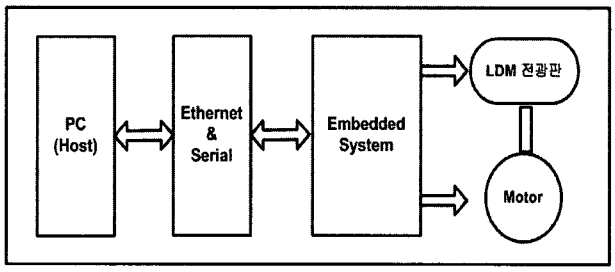

그림 2. 시스템 전체 구성도
Fig. 2. The whole system organization.

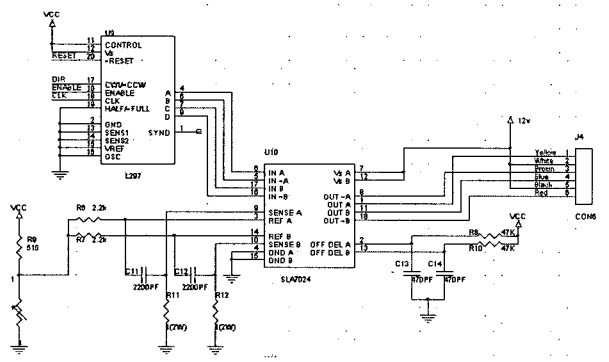

그림 3. 스텝 모터 회로도
Fig. 3. Step motor circuit.

2. LDM 회전부

본 논문에서는 LDM을 회전시키기 위해 스텝 모터를 사용했으며, 그림 3과 같이 스텝 모터(NK243)을 제어하기 위한 모터 상발생기(L297)와 모터 컨트롤 드라이버(SLA7024M)를 사용했다. 스텝 모터는 입력되는 펄스(pulse)로부터 일정 각을 회전하기 때문에 회전 각도나 이동 거리 제어를 정확히 할 수 있다. 모터 상발생기는 스텝 모터에 필요한 4개의 펄스신호를 만들어 내고, 모터 컨트롤 드라이버는 그 신호들을 조작(delay, power 등)해서 스텝 모터에 실제 입력되는 신호를 만들어 낸다.[2]

3. LDM(LED Dot Matrix) 모듈부

LDM 모듈은 '16×16'(가로×세로, DOT)으로 구성되어 데이터(영문, 한글, 한문, 숫자 특수기호 등 문자정보)표현의 최소단위가 된다. DOT의 크기에 따라 여러 종류의 모듈이 있으나 가장 많이 사용되는 것은 Indoor용 모듈로 '16×16' DOT 이며 영문은 한 모듈에 두 문자를

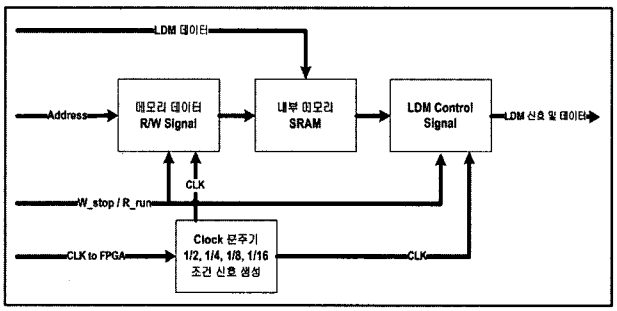

그림 4. LDM FPGA 블록도
Fig. 4. LDM FPGA block diagram.

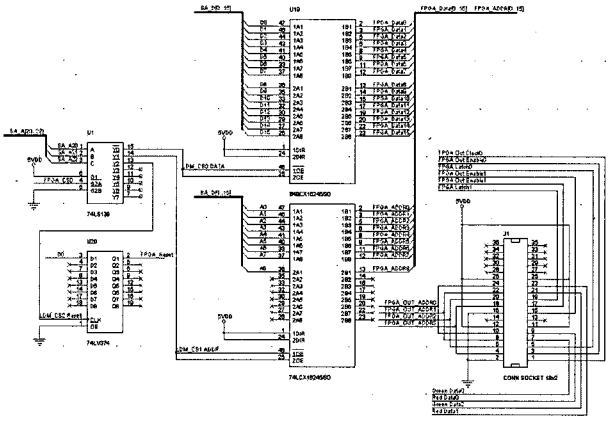

그림 5. LDM FPGA 인터페이스 회로도
Fig. 5. LDM FPGA interface circuit.

표현할 수 있으며 한글은 한 문자를 표현 할 수 있다. 그림 4는 LDM FPGA의 내부 블럭도를 나타내고 있다. FPGA에 데이터를 쓸 경우 어드레스, 데이터, 클럭을 동시에 인가하면 내부 메모리에 데이터가 저장이 되며 FPGA가 데이터를 읽을 경우 LDM 컨트롤 신호와 클럭에 의해 LDM 신호 및 데이터가 LDM 모듈로 입력되게 된다.[6]

그림 5는 CPU에서 입력된 신호를 받아서 FPGA의 LDM 제어 로직에 신호를 인가 시켜준다.[6]

4. LDM FPGA 제어 로직 설계

본 논문에서는 LDM을 동작시키기 위해서 Quartus V3.0을 이용하여 VHDL로 LDM 제어 로직을 설계 하였으며 로직 블록은 크게 클럭 발생부, LDM 신호 발생부, 메모리 읽기/쓰기부 3 블록으로 구성되어져 있다.[7]

(1) 클럭 발생부

그림 6의 RDATACTRL 블록은 클럭을 분주시키는 클럭 분주 블록이며, RSTDLY 블록은 LDM에 출력될 데이터를 먼저 읽어 들이고 Latch 신호가 Enable 되기 전에 Address와 데이터를 LDM 모듈에 전송하는 시간

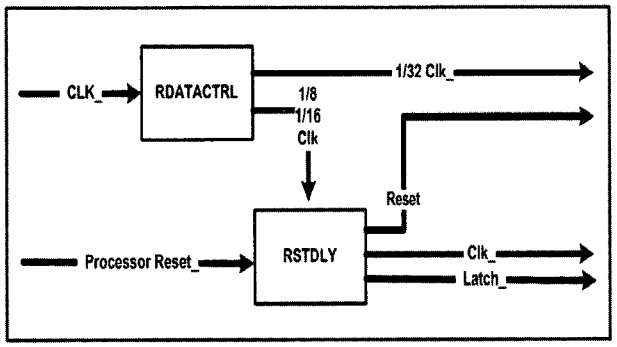

그림 6. 클럭 발생 블록
Fig. 6. Clock generation block.

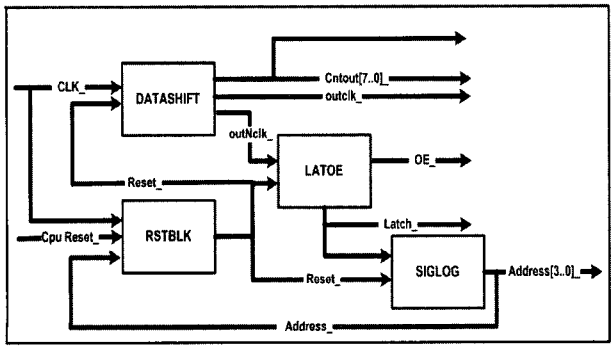

그림 7. 신호 발생 블록
Fig. 7. Signal generation block.

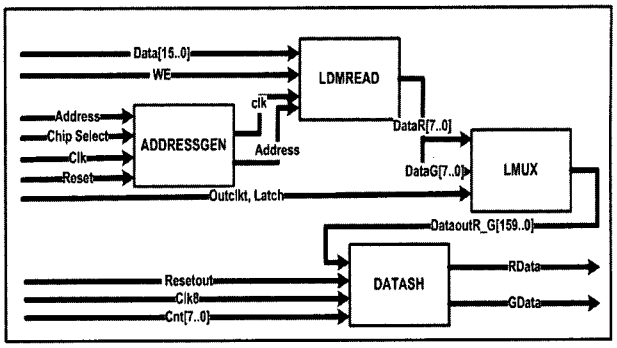

그림 8. 메모리 읽기/쓰기 블록
Fig. 8. Memory read/write block.

을 유지하기 위해서 reset을 지연시키고 메모리에게 발생될 클럭을 발생 한다.

(2) 신호 발생부

그림 7의 DATASHIFT 블록은 LDM 구동 클럭과 현재 진행 중인 모듈의 픽셀 위치를 나타내는 신호를 출력하며, RSTBLK 블록은 모듈에 글자 표현하는 중에 CPU에서 다른 문자를 출력하라는 인터럽트를 방지하도록 한다. LATOE 과 SIGLOG 블록은 각각 모듈의 Enable, Address를 발생시킨다.

(3) 메모리 읽기/쓰기부

그림 8의 ADDRESSGEN 블록은 FPGA의 내부메모리에 데이터 Write시 CPU Address를 디코딩하여 메모리의 Address를 발생시켜준다. LDMREAD 블록은 LDM 모듈 160*16의 픽셀의 크기를 데이터를 저장할 수 있는 320*16 크기의 내부 메모리이며 we신호가 '1' 일 때 쓰기, '1'일 때 읽기가 이루어진다. LMUX 블록은 메모리에서 읽어 들인 데이터를 160개의 병렬 데이터로 변경하며, DATASH 블록은 LDM 모듈의 클럭과 픽셀 Counter에 맞춰 병렬데이터를 시리얼로 변환하여 출력한다.

Ⅲ. 임베디드 소프트웨어

1. 임베디드 시스템(Embedded System)

임베디드 시스템을 동작시키기 위해 시스템에 그림 9와 같이 JFlash을 이용해서 부트 이미지, 커널 이미지, 루트 파일시스템을 다운로드 시킨다. 커널 이미지는 시스템에 사용되는 부분만 선택한 최적화된 이미지이며, 부트 이미지는 시스템을 초기화 시키고 커널 이미지를 다운로드 할 수 있는 명령어가 포함된 이미지이다. 또

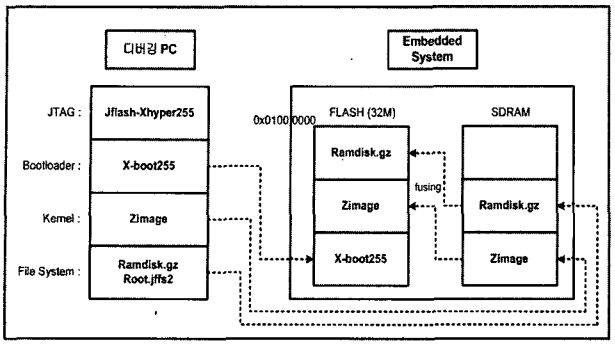

그림 9. 이미지 다운로드 블록도
Fig. 9. Image download block.

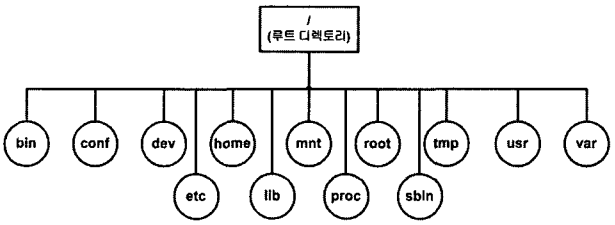

그림 10. 루트 파일시스템 구조
Fig. 10. Root file System.

한 파일시스템은 커널이 동작하기 위한 공간과 라이브러리, 유틸리티, 디렉토리, 소프트웨어 실행파일, 디바이스 드라이버 등이 포함된다.[3]

본 논문에서는 빠른 접근이 필요 없기 때문에 루트 파일 시스템으로 램디스크를 사용하지 않고 플래시 메모리 기반 파일 시스템을 사용한다. 그림 10은 루트 파일시스템의 최소 디렉토리 구조를 나타내고 있다. bin, dev, etc, lib, proc, tmp, usr가 있다.[8]

그림 10에서 root는 마운트 되는 리눅스 파일 시스템이 있는 최상위 디렉토리이며, 파일 시스템의 근간을 이루는 중요한 기본적인 디렉토리을 나타내고 있다.

2. 디바이스 드라이버와 응용프로그램 흐름도

그림 11은 디바이스 드라이버와 응용프로그램과의 인터페이스 흐름도를 보여준다. install이라는 쉘 스크립트를 만들어 동시에 LDM과 Step Motor에 대한 Object를 등록하고 각각에 대한 주 번호와 부 번호를 부여한다. Symbolic Link로 각 디바이스에 대한 핸들을 얻어 실행시키고자 하는 기능들에 대한 ioctl Value를 디바이스 드라이버에 전송함으로써 하드웨어를 제어 하였다. 본 논문에서는 문자 디바이스 드라이버를 사용해서 순차적으로 사용자 프로그램과 1바이트 단위로 데이터를 송수신 한다.[9][12]

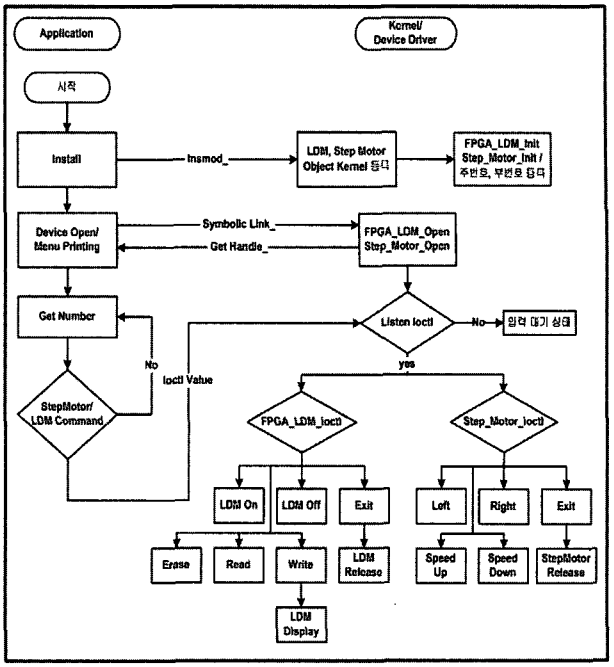

그림 11. 디바이스 드라이버와 응용프로그램 흐름도
Fig. 11. Flow Chart of Device Driver and Application.

표 1. 데이터 패킷의 구현
Table 1. Implement of Data packet.

	1	2	3	4	5	6	7	8
명령	데이터 길이				0xDD	0xDD	폰트구별	0x00
데이터	8바이트의 데이터가 연속적으로 전송된다.							

3. LDM 데이터 및 명령 패킷

표 1은 이더넷과 시리얼통신을 이용해서 LDM 모듈에 데이터를 전송하기 위한 패킷이며 총 8byte로 이루어져 있다. 1번째에서 4번째 바이트 까지는 전송될 데이터의 바이트 수를 나타내고 5번째와 6번째 바이트는 위의 패킷의 데이터 패킷임을 나타내는 플래그 비트이다. 그리고 7번째 바이트는 출력될 문자 폰트 크기를 구별하기 위해서 사용되어지며 이 플래그가 0일 경우 12포인트의 글자가 전송되고 1일 경우 22포인트의 글자가 전송됨을 나타낸다. 마지막 8번째 바이트는 사용되어지지 않는다.

표 2는 데이터 패킷을 통해서 전송된 데이터를 어떻게 처리 할 것인가 결정하기 위한 정의된 명령 패킷을 나타내고 있다. 위의 명령 패킷 또한 데이터 패킷과 마찬가지로 8바이트로 구성되었으며 데이터 패킷의 데이터 길이에 해당하는 1번째 바이트와 4번째 바이트는 사용하지 않으며, 데이터 패킷에서 5번째와 6번째 바이트는 명령패킷임을 나타내기 위해서 0xBB를 사용하였다. 그리고 7번째 바이트는 위의 명령 패킷으로서 어떠한

표 2. 명령 패킷의 구현
Table 2. Implement of command packet.

	1	2	3	4	5	6	7	8
일반형식	Reserved				Command Packet		명령 종류	시프트 속도
좌로	0x00	0x00	0x00	0x00	0xBB	0xBB	0x11	스피드
우로	0x00	0x00	0x00	0x00	0xBB	0xBB	0x12	스피드
위로	0x00	0x00	0x00	0x00	0xBB	0xBB	0x13	스피드
아래로	0x00	0x00	0x00	0x00	0xBB	0xBB	0x14	스피드
좌우교차	0x00	0x00	0x00	0x00	0xBB	0xBB	0x15	스피드
우좌교차	0x00	0x00	0x00	0x00	0xBB	0xBB	0x16	스피드
상하교차	0x00	0x00	0x00	0x00	0xBB	0xBB	0x17	스피드
하상교차	0x00	0x00	0x00	0x00	0xBB	0xBB	0x18	스피드
정지	0x00	0x00	0x00	0x00	0xBB	0xBB	0x19	스피드
깜박이기	0x00	0x00	0x00	0x00	0xBB	0xBB	0x20	스피드

명령을 실행할 것인가를 나타내고 있다. 마지막 8번째 바이트는 7번 바이트에서 나타내는 명령의 종류에 의해 어느 정도의 속도로 시프트 할 것인가를 나타내며, 이 값은 0부터 100까지의 범위를 갖는다. 값이 작으면 시프트 속도가 빨라지고, 값이 커지면 시프트 속도가 느려지게 된다.

Ⅳ. 실험 및 결과

1. 개발 PC 환경

본 논문에서는 Linux 레드헷 9.0을 설치하고 임베디드 시스템에 다운로드 할 수 있도록 시리얼을 이용한 RS232C, 이더넷을 이용한 tftp(Trivial File Transfer Protocol) 환경을 설정, 크로스 컴파일러을 구축했으며, 임베디드 시스템을 구성하기 위해 커널 컴파일, 파일 시스템, 디바이스 드라이버, 응용프로그램을 제작해서 직접 하드웨어에 다운로드를 하고 동작 과정을 테스트 하도록 했다. 그림 12는 PC에서 실험환경을 나타내고 있다. 첫 번째는 호스트 PC와 임베디드 시스템 간의 시리얼을 연결해서 LDM 회전과 LDM 모듈을 동작 시킬 수 있으며, 두 번째는 다른 장소(클라이언트 PC)에서 랜 케이블을 통해 원격으로 LDM 모듈을 동작 시킬 수 있다.

2. LDM FPGA 로직 시뮬레이션을 위한 타이밍

그림 13은 LDM 모듈 구동 시 요구되는 타이밍도를 보여준다. LDM 모듈 하나는 16×16으로 구성되어 있다. LDM 모듈의 픽셀 각각의 데이터를 입력시키기 위한 동기신호로 Out Enable이 LOW인 동안 데이터를 LDM 모듈이 받아들이게 된다. 다시 말해서 Out Enable이 LOW 이고 Clock이 상승에지 일 때 LDM 모듈의 한

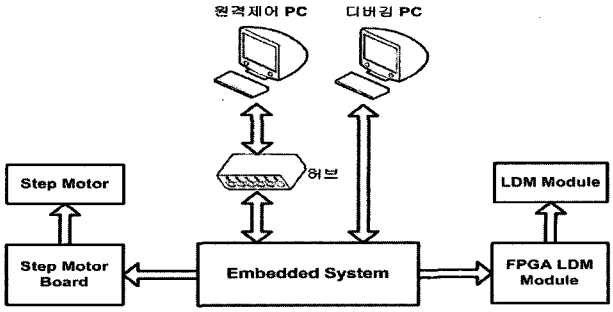

그림 12. PC 실험환경
Fig. 12. Experiment environment of PC.

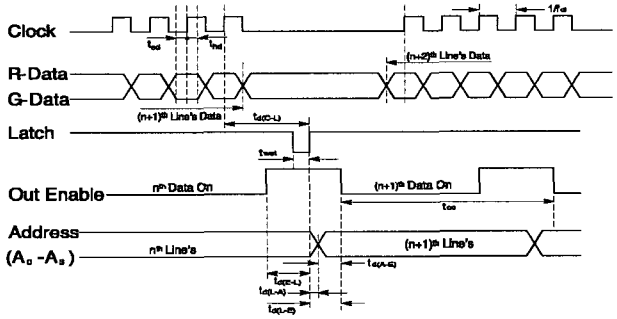

그림 13. LDM 타이밍도
Fig. 13. LDM timing diagram.

표 3. 시뮬레이션 신호
Table 3. Simulation signal.

CLK	FPGA 의 동작클럭
OutCLK	LDM 모듈의 입력 클럭
address	LDM모듈의 열 신호
Latch	LDM 모듈의 Latch 신호
OE	LDM 모듈의 OutEnable 신호
Reset	FPGA 의 신호 출력 유무
Cntout	현재 진행중인 모듈 픽셀의 위치
Rdata	LDM 모듈의 R-Data 신호
Gdata	LDM 모듈의 G-Data 신호
DATA	프로세서에서 입력되는 LDM 데이터로서 FPGA의 내부메모리에 데이터가 저장되게 된다.
Ramaddress	FPGA 내부메모리 address
CS	FPGA 내부메모리 WE 신호이다.

픽셀의 R-Data(red) 와 G-Data(Green)을 입력 받게 된다. 그러므로 Clock이 16번 발생하면 LDM 모듈의 1열 데이터가 입력되게 된다. 만약 LDM 모듈이 증가 한다면 클럭 수는 16×모듈수가 발생하여야 한다. 모듈의 열을 나타내는 신호는 Address 신호로서 Latch 가 LOW 일 때 Clock 에 의해 입력된 데이터 열을 선택하게 된다.

표 3은 FPGA로 설계된 블록의 시뮬레이션으로 각 신호에 대한 설명을 나타내었다.

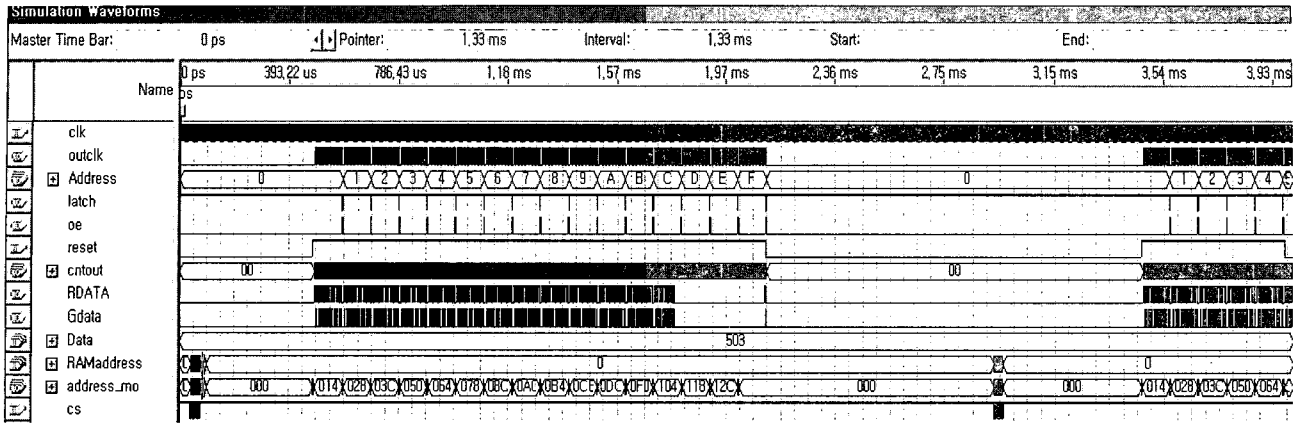

그림 14. LDM 전체 로직 시뮬레이션
Fig. 14. LDM whole logic simulation.

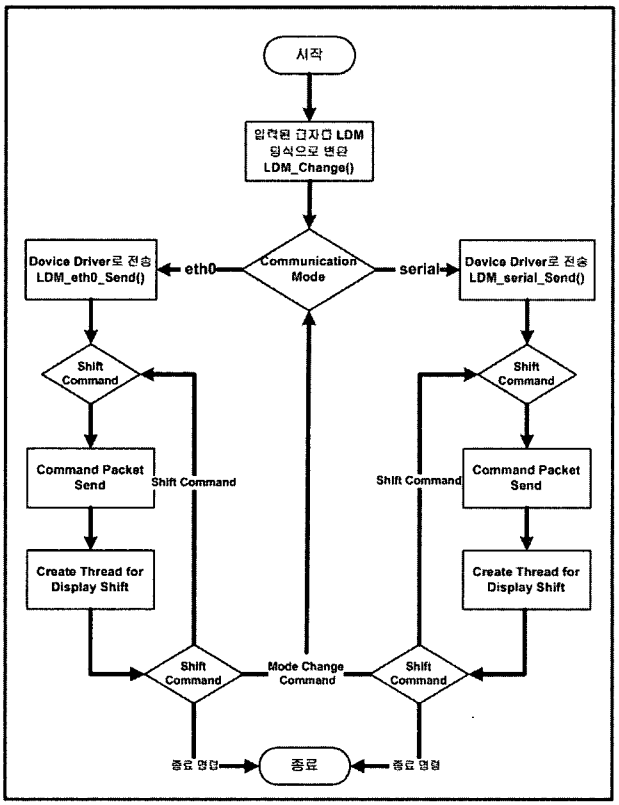

그림 15. 통신 디바이스 드라이버 순서도
Fig. 15. Communication Device Driver Diagram.

그림 14는 LDM 모듈을 동작시키기 위한 전체 VHDL 시뮬레이션 한 결과를 나타낸다. Reset이 Low, 즉 We이 high일 때 CPU의 Chip Select와 Address를 디코딩함으로써 FPGA의 512*16bit 크기의 내부 메모리에 LDM 모듈에 출력하고자 하는 데이터를 저장하게 된다. Reset이 High, 즉 we이 Low임과 동시에 LDM 모듈에 출력할 데이터를 읽어 들이기 위해서 내부 메모리에 대한 Address를 Counting하게 된다. 읽어 들인 데이터는 Red Data, Green Data로 분리, LDM 모듈 수에

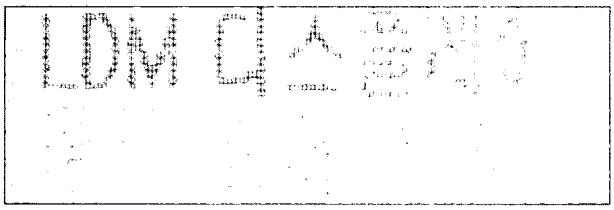

그림 16. LDM 원격 실행 결과
Fig. 16. Result of LDM remote control.

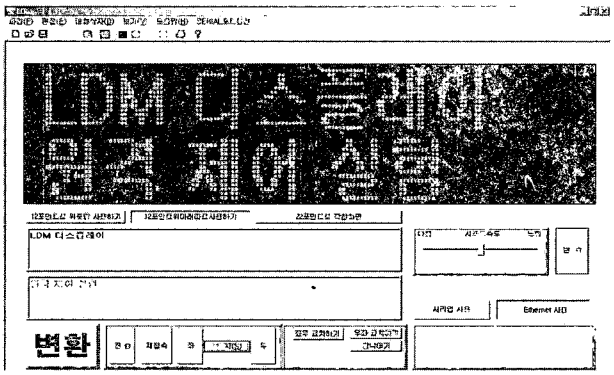

그림 17. LDM 원격 제어 응용프로그램
Fig. 17. LDM remote control application program.

맞게 160개의 병렬 데이터로 변환하고 다시 직렬 데이터로 변환함으로써 LDM 모듈에 전송하게 된다.

3. LDM 통신 흐름

본 논문에서 LDM을 원격으로 제어하기 위해 그림 15와 같이 이더넷과 시리얼에 대한 디바이스 드라이버 순서도를 나타냈다.[10][11]

4. 결과

본 논문에서는 LDM을 동작 시키고 회전시키기 위해 LDM FPGA 보드와, 스텝 모터 보드를 설계했으며, 임베디드 시스템 상에서 하드웨어를 동작시킬 수 있는 디

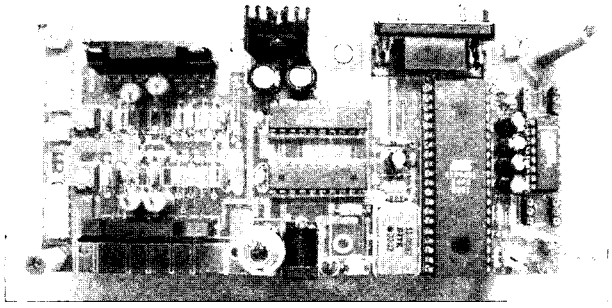

그림 18. 스텝 모터 제어보드
Fig. 18. Step Motor Control Board.

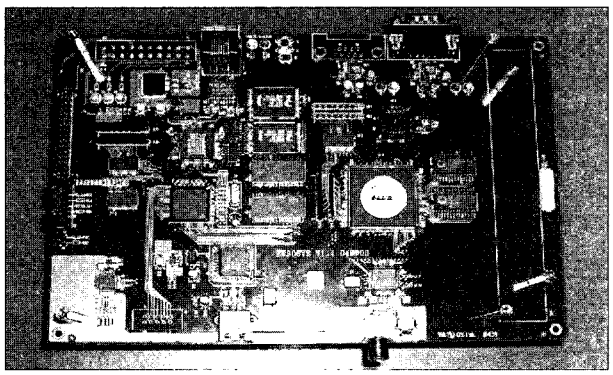

그림 19. 임베디드 시스템 및 LDM FPGA 보드
Fig. 19. Embedded System and LDM FPGA Board.

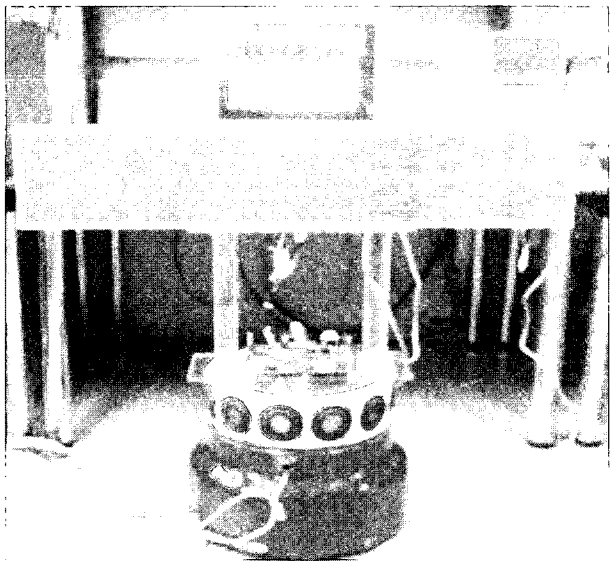

그림 20. LDM 원격 제어를 위한 임베디드 시스템
Fig. 20. A Embedded System for Remote Control of LDM.

바이스 드라이버를 작성, 이 드라이버를 동작 시킬 수 있는 응용프로그램을 작성해서 동작이 되는지 확인해 봄으로서 임베디드 시스템을 구현 할 수 있었다. 다음 그림 16은 LDM 원격 실행 결과이며, 그림 17은 PC상에서 실행되는 LDM 원격 제어 응용프로그램을 보여주고 있으며 그림 18은 스테모터를 제어하기 위한 제어 보드를 보여주고 있다. 그림 19는 구현된 임베디스 시스템과 LDM FPGA 보드를 보여주며 그림 20은 구현된 전체 시스템을 보여주고 있다 .

V. 결 론

본 논문은 LDM을 원격으로 제어하기 위한 방법으로 임베디드 시스템을 구성했다. 과거의 시스템들은 한 장소에서 시리얼을 통한 임베디드 시스템의 제어가 주를 이루고 있으며 이러한 요소는 많은 노동력, 노동시간, 비용이 많이 드는 단점을 가지고 있다. 이러한 단점을 보완하기 위해서 32bit RISC 프로세서인 Intel사의 XScale 임베디드 시스템과 인터페이스 시킬 수 있는 액튜에이터 보드, LDM FPGA 보드 제작과 리눅스 파일 시스템, 리눅스 디바이스 드라이버, 디바이스 드라이버 응용프로그램을 작성 했다. 또한 임베디드 시스템에 최적의 운영환경을 제공해 줄 수 있는 운영체제인 임베디드 리눅스와 파일시스템을 포팅 하였고, 일반 PC에서 시스템을 동작 시킬 수 있도록 응용프로그램을 제작했다. PC와 임베디드 시스템은 시리얼 포트, 이더넷으로 연결이 되어져 PC에서 개발된 디바이스 드라이버와 응용프로그램을 리눅스 에뮬레이터 Minicom을 이용해서 임베디드 시스템에 적재 시켰으며, 적재 시킨 이미지를 에뮬레이터를 통해 임베디드 시스템의 커널에 적재하고 삭제 시킬 수 있었으며, 응용프로그램을 실행함으로서 하드웨어 동작을 디버깅할 수 있었다. 향후에 이러한 임베디드 시스템의 원리를 이용한다면 다른 임베디드 시스템 개발에 많은 시간을 줄일 수 있으며, 쉽게 디바이스 드라이버를 작성할 수 있을 것으로 본다.

참 고 문 헌

[1] 박재호, “임베디드 리눅스”, 한빛미디어, 2002.
[2] Altera, "FLEX 10K Embedded Programmable Logic Device Family Ver 4.2", Jan 2003.
[3] O'REILLY "Understanding th Linux Kernel", Daniel Pierre Bovert", Dec 2002.
[4] 윤성철 · 윤영기, “Linux System & Shell Programming", 영진닷컴, 2003.
[5] Intel, "PXA255 Processor Developer's Manual, March 2003.
[6] Hybus. X-Hyper255A Developer's Manual, Jun 2003.
[7] 박세현, “VHDL 기본과 활용”, 그린출판사, 1998.
[8] 서자룡, “리눅스 7.1 그대로 따라하기”, 혜지원 도

서출판, 2001.
[9] O'REILLY, "Linux Device Drivers, 2nd Edition", Alessandro Rubini. Jun 2001.
[10] 이연조 "임베디드 리눅스 프로그래밍", PC'BOOK, 2002.
[11] 김종훈 · 김종진 · 김동균, "Linux & Unix C 프로그래밍", 한빛미디어, 2003.
[12] 김인성 · 류태중, "Linux Device Drivers", 한빛미디어, 2000.

저 자 소 개

문 철 홍(정회원)
1978년 인하대학교 전자공학과 졸업 (공학사)
1987년 인하대학교 전자공학과 졸업 (공학석사)
1989년 인하대학교 전자공학과 졸업 (공학박사)
1984년 삼성전자 반도체 사업부
1989년 아륭유기
1994년 초당대학교 전임강사
1996년~현재 광주대학교 전자공학과 부교수
<주관심분야 : 마이크로프로세서, 영상 · 신호처리>

강 상 우(학생회원)
2003년 광주대학교 전자공학과 졸업 (공학사)
2005년 광주대학교 전자공학과 졸업 (공학석사)
2005년~현재 Uni-Test 연구원
<주관심분야 : SoC, Embedded Linux>