DSC/NBDP시스템의 제어기설계를 위한 실시간운영체제 기술 개발

이 헌 택*

The development of RTOS technique for designing the controller of DSC/NBDP system

Lee Hountaek*

요 약

리눅스(Linux)는 여러 가지 실시간 운용체제의 하나로 기술개발이 적극 이루어지고 있으며, 내장형 RTOS로서 가능성과 기술적 안정성을 검증하는 연구가 진행되고 있다. 본 연구에서는 해상통신분야의 통신설비에 적용될 수 있는 내장형 실시간 운영체제의 분석과 기술개발을 통해 계측분야에서 요구되는 실시간 계측과 통신기능이 강화된 내장형 실시간 운영체제기술에 대해 정리하였다. 본 연구에서는 타켓보드를 설계하여 내장형 리눅스 커널(Embedded Linux Kernel)을 분석하고, 이를 포팅(Porting)하고 테스트 환경하에서 사용자 코드를 실행시켜 내장형 리눅스 커널을 기반으로 하는 소형 네트워크 제어기 응용 설계기술을 연구하였다.

Abstract

Linux is the one of various RTOS, also embedded linux has being studied with focus on technical stability and commercial utilities. In this paper, the technical trial was discussed on the development of real-time operating system that provides real time capability and extends the network communications ability and would be applied to the maritime mobile communication system through analysis the embedded linux kernel. Some techniques for Analyzing the embedded linux kernel and designing the target board, making the kernel image and porting the kernel are summarized in this paper.

I. 서 론

내장형 운영체제(Embedded OS)는 특수목적용으로 만든 하드웨어에 포팅된 운영체제이다. 임베디드 시스템은 그 종류가 매우 광범위하므로 그 운영체제 역시 다양한 종류가 존재한다. 초기의 임베디드 시스템은 비교적 단순해서 운영체제가 불필요했으나, 최근에서는 그 역할이 매우 많아지고 복잡해졌기 때문에 운영체제 개념의 중요성이 대두되고 있다. 또한, 임베디드 OS중에서도 실시간 운영체제(Real-time Operating System : RTOS)는 제한된 시간내에 작업이 이루어져야 하는 시스템으로 논리적인 정확성뿐만 아니라 시간적인 정확성까지 요구되는 시스템이다. 특히 계측시스템에 RTOS를 적용하여 시스템의 성능을 획기적으로 향상시킬 수 있으며, 이러한 요구조건에 의해 특별히

*인천전문대학

접수일자 : 2004. 2. 28

설계되는 RTOS도 개발되고 있다.

내장형 RTOS는 여러 가지 기술들이 개발되고 있으나, 현재까지는 고가의 비용과 취급기술의 난이도를 요구하고 있다. 그러나, 취급기술의 난이도는 기술개발활동에 극복할 수 있으며, 따라서 고가의 비용이 요구되지 않는 형태의 RTOS도 적극 검토되고 있다. 이중에서 리눅스(Linux)는 Open Source로 운영되므로 현재 많은 분야에서 기술개발이 적극 이루어지고 있으며, 내장형 RTOS로서 가능성과 기술적 안정성을 검증하고 있으며 개선시키고 있다. 그러나, 아직까지 용이하게 사용할 수 있는 16비트 또는 32비트 마이크로 콘트롤러에 적용시킬 수 있는 내장형 운영체제는 개발되어 있지 않으며, 따라서 본 연구에서는 해상통신분야의 DSC/NDBP설비의 제어기 및 기타 통신시스템 등에 적용될 수 있는 내장형 실시간 운영체제의 분석과 기술개발을 통해 계측분야에서 요구되는 실시간 계측과 통신기능이 강화된 내장형 실시간 운영체제기술을 개발하였다.

본 연구를 수행함에 있어서 리눅스기반의 내장형 실시간 운영체제기술개발을 목적으로 타켓보드를 설계하여 내장형 리눅스 커널(Embedded Linux Kernel)을 분석하고, 이를 포팅(Porting)하고 테스트 환경하에서 사용자 코드를 실행시켜 내장형리눅스커널을 기반으로 하는 소형 네트워크 제어기 응용 설계기술을 연구하였다.

본 논문의 구성은 다음과 같다. 제2장에서는 Embedded Linux Kernel의 구성요소 및 포팅을 위한 개발환경 구축방법 등을 연구하여 Embedded Linux의 Booting 과정을 정리하였다. 제3장에서는 내장형 리눅스커널의 포팅을 위해 SA-1110을 사용한 시스템의 분석 및 이를 응용한 설계기술 및 성능평가에 대해서 연구결과를 정리하였다.

II. Embedded Linux Kernel의 분석

2.1 Embedded Network System과 실시간 운영체제(RTOS)의 현황

하드웨어 기술 발전에 힘입어 가격이 하락하고 미래 정보화 사회 환경에서 그 효용성이 커짐에 따라서 전문화된 기능을 수행하는 임베디드 시스템의 수는 기존의 범용 기능을 수행하는 PC보다 많아지고 있다. 지리적 혹은 기능적으로 분산된 수천 혹은 수만 개의 임베디드 시스템들은 네트워크를 수단으로 상호협력 능력을 갖춤으로써 더욱 강력하고 유용한 시스템으로 발전할 것으로 보인다.

임베디드 시스템은 임의의 시스템에 내장(Embedded)된 형태로 구성되어 특정 하드웨어 구성요소를 제어하기 위하여 사용되며, 제한되고 전문화된 기능을 수행하는 장치로 정의된다. 임베디드 시스템들의 연결 형태는 일부분의 임베디드 시스템 집합이 네트워크 그룹을 형성하고 이러한 그룹들이 모여서 상위 네트워크 그룹을 구성하는 계층적 구조를 갖는다. 이때 하위 네트워크 그룹들은 지리적 위치 및 관리자에 따라 그룹단위 노드 기능은 물론 각 네트워크 그룹내에서 사용하는 상호연결 기술은 현실적으로 다른 경우가 대부분이다.

내장형 운영체제로는 팜 파일럿에 기반한 팜 운영체제(Palm OS), 마이크로소프트사의 윈도우즈 CE(Windows CE), 각종 실시간 운영체제(RTOS : Realtime OS), 오랜 기술축척을 통해 상당부분 시장을 점유하고 있는 Wind River System 사의 VxWorks 등의 운영체제가 있다. 이러한 상용 임베디드 OS는 다음과 같은 문제점을 가지고 있다.

-. 크기가 너무크고 커널의 재구성이 쉽지 않다.
-. 임베디드 휴대용 장비가 지원하는 자원의 제약을 극복하기 어렵다.
-. 다양한 시스템을 구성하기 힘들다.
-. 초기 구입비와 로열티가 비싸다.
-. 개발 인력이 많지 않다.

최근에는 내장형 프로세서 제작사들이 자사 프로세서를 탑재한 내장형 시스템에 리눅스를 포팅함과 동시에 리눅스의 가격이 저렴하다는 장점을 갖추고 있다. 또한, GNU 일반 공용 라이선스 덕분에 Kernel과 모든 시스템 프로그램의 소스 코드를 자유롭게 읽고 수정할 수 있기 때문에, 서버급·중소형 개인용 컴퓨터뿐만 아니라, 내장형 운영체제로도 각광을 받을 것으로 예상된다.[11]

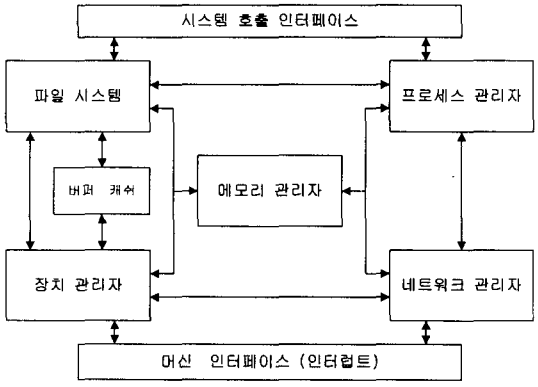
2.2 Embedded Linux Kernel의 분석

1. Linux 커널의 구조

Linux Kernel은 크게 프로세서 관리자 , 메모리 관리자, 파일시스템, 네트워크 관리자, 그리고 디바이스 드라이버의 장치관리자 이렇게 5가지 부분으로 구성된다. Kernel은 자원 관리자이며, Kernel이 관리하는 자원에는 물리적인 자원과 추상적인 자원으로 나뉘어 진다.[7]

프로세서 관리자는 프로세서의 생성, 실행, 상태전이(state transition), 스케줄링, 시그널 처리, 프로세스간 통신(Inter Process Communication) 등의 서비스를 제공한다. 메모리 관리자는 가상 메모리, 주소 변환(address translation), 페이지 부재 결함

처리 등의 서비스를 제공한다. 파일시스템은 파일의 생성, 접근 제어, Inode 관리, 디렉토리 관리, 슈퍼 블록 관리, 버퍼 캐쉬 관리 등의 서비스를 제공한다. 네트워크 관리자는 소켓(Socket) 인터페이스, TCP/IP 같은 통신 프로토콜 등의 서비스를 제공하다. 장치 관리자는 디스크 드라이버, 터미널, CDROM, LAN Card 등과 같은 주변 장치를 구동하는 드라이버들로 구성된다.

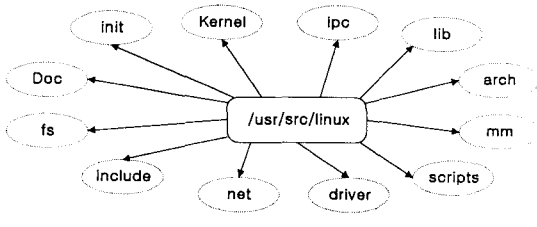


<그림 2-1> 리눅스 Kernel의 내부 구조
<Fig 2-1>Internal block diagram for Linux kernel

리눅스 Kernel의 내부 구조는 <그림 2-1>에서 보는 바와 같으며, Application Level과 Hardware Level의 사이에 위치하게 된다. 그리하여 Kernel은 기계 인터페이스 및 인터럽트 처리 알고리즘 등을 사용하여 Hardware와 통신하며, 시스템 호출 인터페이스를 이용하여 Application Level과 통신하게 된다.

2. Linux 커널의 소스 트리 구조

아래 <그림 2-2>은 Linux 커널 소스 트리 구조를 도시화한 것이다.



<그림 2-2> Linux 커널의 소스 트리 구조
<Fig 2-2> The source tree block diagram for Linux kernel

① Kernel : 프로세스 관리자가 구현된 디렉토리이다. 태스크의 생성과 소멸, 프로그램의 실행, 스케줄링, 시그널 처리 등의 기능이 이 디렉토리에 구현되어 있다.

② arch : Linux 커널 기능 중 하드웨어 종속적인 부분들이 구현된 디렉토리이다. 이 디렉토리는 CPU 종류에 따라 하위 디렉토리가 다시 구분된다. 32비트 ARM 처리기는 Linux/arch/arm 하위 디렉토리에 구현되어 있다. arm/boot 디렉토리는 시스템의 초기화에 사용하는 부트스트랩 코드가 구현되어 있으며, arm/kernel 에는 프로세스 관리자

③ fs : Linux에서 지원하는 다양한 파일 시스템과 시스템 호출이 구현된 디렉토리이다. 각 파일 시스템은 하위 디렉토리에 구현되어 있는데, 대표적인 파일 시스템으로는 ext2, nfs, ufs, msdos, ntfs, proc, coda 등이 있다.

④ mm : 메모리 관리자가 구현된 디렉토리이다. 가상메모리, 태스크마다 할당되는 메모리 객체 관리, 커널 메모리 할당자 등의 기능이 구현되어 있다.

⑤ driver : 디스크, 터미널, 네트워크 카드 등 주변 장치를 추상화시키고 관리하는 커널 구성요소인 디바이스 드라이버가 구현된 디렉토리이다. Linux에서 디바이스 드라이버는 크게 블록(Block) 디바이스 드라이버, 문자(Character) 디바이스 드라이버, 네트워크 디바이스 드라이버로 구분된다.

⑥ net : Linux에서 지원하는 통신 프로토콜이 구현된 디렉토리이다. 현재는 Linux는 대표적인 통신 프로토콜인 TCP/IP뿐만 아니라 Unix 도메인 통신 프로토콜, X.25, 802 통신 프로토콜, IPX, AppleTalk 등이 구현되어 있다.

⑦ ipc : Linux 커널이 지원하는 프로세스간 통신기능이 구현된 디렉토리이다. 이 디렉토리에는 메시지 패싱(message passing), 공유 메모리(Shared Memory), 그리고 세마포어(Semaphore)가 구현되어 있다.

⑧ init : 커널의 초기화 부분, 즉, 커널의 메인 시작 함수가 구현된 디렉토리이다.

⑨ include : Linux 커널이 사용하는 헤더 파일이 구현된 디렉토리이다. 헤더 파일 중에서 하드웨어 독립적인 부분은 하위에 linux 디렉토리에 구현되어 있으며, 하드웨어 종속적인 부분은 처리기 이름으로 구성된 하드웨어 디렉토리에 구현되어 있다.

⑩ other : Linux 커널 및 명령어들에 대한 자세한 문서 파일들이 존재하는 Doc 디렉토리,

커널 라이브러리 함수들이 구현된 lib 디렉토리, 컴파일된 모듈 함수들이 존재하는 Module 디렉토리, 그리고 커널 구성 및 컴파일할 때 이용되는 스크립터들이 존재하는 scripts 디렉토리 등이 존재한다.[5]

3. Embedded Linux Kernel의 부트로더(bootloader) 및 부팅과정

ROM 시작 코드와 레지스터 구성 같은 초기 단계들은 마이크로프로세서 하드웨어에 의존한다. 커널 그 자체는 초기화 코드가 처음 실행되어진 마이크로프로세서 구조를 포함한다. 이 초기화 코드는 보호 모드 작동을 위한 마이크로프로세서 레지스터들을 구성하고 그 다음에 start_kernel이라고 불리 우는 아키텍처에 독립적인 커널 시작지점을 호출한다.

커널 부트 과정은 모든 구조들에 동등하며, 리눅스 부팅은 다음과 같은 단계를 포함한다.

① 프로세스 리셋 후에 ROM 시작 코드를 실행한다.
② ROM 시작 코드는 CPU, 메모리 제어기, 그리고 디바이스를 초기화하고 메모리 맵을 구성한다. ROM 시작 코드 그 다음에는 부트로더를 실행한다.
③ 부트 로더는 플레시 메모리 또는 TFTP 서버 전송으로부터 RAM 내의 리눅스 커널에 압축을 해제한다. 그것은 그 다음에 커널의 첫 번째 명령으로 점프하여 실행한다. 커널은 처음으로 마이크로프로세서 레지스터들을 구성한 후, 아키텍처에 독립적인 시작점인 start_kernel을 시작한다.
④ 커널은 캐쉬와 여러 디바이스 드라이버를 초기화한다.
⑤ 커널은 루트 파일시스템을 마운트(mount)한다.
⑥ 커널은 init 프로세스를 실행한다.
⑦ init 프로세스 실행은 공유한 런타임 라이브러리들을 적재하고 사용자 프로세스를 실행한다.

4. Embedded Linux Booting Code 분석

일반적으로 Boot loader라 하면 x86계열 리눅스에서 LILO를 많이 사용한다. LILO란 Linux Loader로써 도스나 Windows NT, 리눅스등 다른 OS를 선택적으로 부탕할 수 있도록 하는 기능을 제공한다. LILO를 하드디스크의 MBR에서 동작이 되는 프로그램으로 OS가 실행 할 수 있도록 점프하는 기능을 수행한다. 부트로더는 하드웨어가 리셋된 이후 메모리의 설정 및 I/O장치의 점검 이후 커널이 부팅되기 전까지 커널을 로딩하고, 메모리를 초기화 및 세그먼트 설정을 조정하는 역할을 한다. Bootloader의 실질적인 시작으로, SA1110 board의 구동시 0번지로 점프하고, 0번지에는 vector table이 존재한다. hardware reset시 이 vector table의 setup이 가리키는 주소를 찾아서 실행되는데, 이 vector table을 start.S에서 만들어 준다.부트로더는 실질적으로 스타트업(Start-up)루틴으로 수행되는데, 스타트업루틴의 역할을 다음과 같다. vector table(start.S는 link시 항상 가장 앞에 있어야 함. vector table을 가장 앞에 놓기 위해) 모든 Interrupt를 막는다. 또한, CPU의 Clock을 지정하고, SDRAM 및 Flash 영역을 설정한 후 C code(BeforeMain())로 Jump한 후 커널을 로딩하게 된다.

III. 내장형 리눅스 커널 포팅을 위한 시스템 설계

임베디드 리눅스 커널의 포팅과 응용 기술을 위한 타켓보드 시스템설계에 있어서, 각 시스템 설계 사양결정에 따라 인텔사의 SA-1110 CPU를 선정하였으며, 평가회로에 리눅스커널을 포팅하기 위한 부트로더와 스타트업 루틴 및 관련 루트이미지와 램 이미지를 생성하고 포팅하기 위한 소요기술들에 대해서 연구하였다. 부팅까지의 과정에 대한 연구를 바탕으로 어플리케이션에서의 원격계측 소프트웨어를 구축하고 개발하기 위한 과정이 구현되었다.

3.1 kernel porting

일반적인 임베디드 리눅스 포팅이란 CPU level porting과 Board level porting으로 나눌 수 있다. CPU level porting은 Cross 개발환경과 Linux kernel에 있는 Arch부분을 특정 CPU환경에 맞도록 coding하는 것이다. 일체의 H/W에 관련된 Linux Kernel을 다시 만드는 작업이다. Board level porting은 해당 CPU에 관계되는 Arch 부분이 일반 Linux에 포함되었거나 patch가 존재하는 경우이다. 이러한 경우는 Cross 개발환경을 구축하고 해당보드 특성과 관련된 부분만 수정하여 사용하면 된다. Poring의 작업순서는 H/W 제작하여

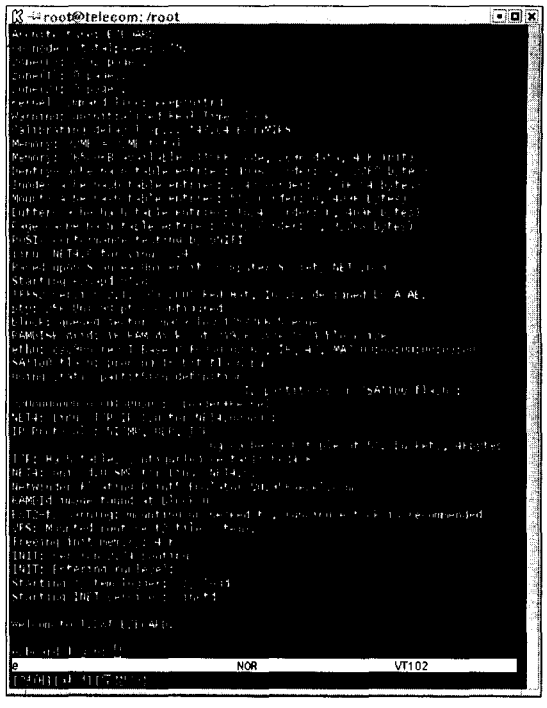
Cross 개발 환경 구축하고, Arch 관련 부분 coding 및 porting을 준비한다. 먼저 용도에 적합한 H/W를 설계하고 임베디드 시스템에 Linux를 porting하기 위해서는 분석 작업을 선행하여야한다. H/W의 규격에 따라 시험회로를 제작하고, 커널의 다운로딩을 위한 부트로더 이식 및 OS Kernel porting, Device driver 제작 및 Application 구현으로 진행한다.

1. Target Board System 사양 :
-. H/W : StrongARM(SA-1110)
-. 개발환경 : JTAG, Boot loader,
ARM Tool chain 등의 개발환경 제공
-. Kernel 및 Device driver : 2.4.0
-. Application : open source

2. Cross 개발환경 구축 : Cross 개발 환경이란 Host System에 Target device용 Linux를 개발하기 위한 모든 환경을 말한다. 먼저 해당 CPU에 맞는 개발 툴 체인(tool chain)환경을 구축해야 한다. 하드웨어 설계 후의 보드시험을 위해 JTAG을 이용하며, 개발초기의 부트로더 이식을 위해 SA1110용 Boot loader인 blob은 lart사의 소스를 활용하여 이용하였다.

3. Arch 관련 부분 coding 및 porting 순서
① Boot loader 설치
② Cross Compiler를 이용하여 zImage 생성
③ Root File System(ramdisk.gz) 생성
④ Boot loader, zImage ,ramdisk.gz Loading : 모든 file은 /usr/src/에 있다고 가정하고 작업을 한다.
⑤ Kernel의 zImage 생성과 컴파일 : menuconfig를 실행한다. 필요한 설정과 선택사항을 설정하여 환경파일을 생성한다. 그리고, library 의존성을 확인하며, 마지막으로 make을 실행하면, 성공적으로 컴파일되어, arch/ arm/boot에 zImage가 만들어 졌는지 확인한다.

4. Linux 커널의 부팅 및 실행과정 : 부트로더의 화면이 나타나고, TFTP를 통해 생성된 커널 이미지를 플래쉬메모리에 써넣는다. 그리고, 전원을 켜면 target board에서는 리눅스커널이 정상적으로 부팅되었을 때 다음과 같은 화면을 볼 수 있다.



3.2 평가회로(Evaluation Board)의 설계

본 연구에서는 32비트 RISC프로세서를 이용하여 내장형 리눅스운영체제를 포팅하기 위한 평가회로를 설계하였다.인텔 SA-1110을 이용하여 평가회를 설계하였으며, 시스템사양은 리눅스를 포팅하고 모니터링 하기 위한 최소한의 하드웨어로 구현하였다.

1. INTEL SA-1110의 특징

SA-1110은 32비트 마이크로컨트롤러로서 고성능으로 평가받고 있으며, 내부에 ARM 코어를 내장하고 있을 뿐만 아니라, LCD 제어모듈과 Memory와 PCMCIA 제어모듈 및 Clock 발생기를 내장하고 있어 시스템의 소형화 및 저전력화 등 많은 장점을 가지고 있다. 연구개발 과정에서 C언어를 뿐만 아니라, 어셈블러와 같이 프로그램을 구현할 수 있는 점을 채택하여 연구에 활용토록 하였다.

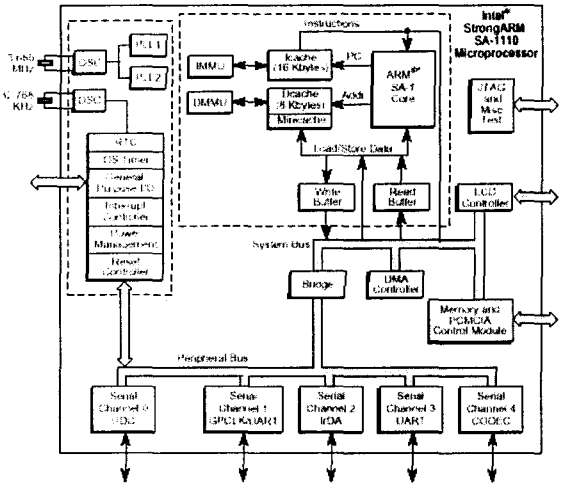
다음은 SA-1110의 특징을 설명한 것이다.[2][3]

-. 32-bit RISC processor Core
-. Low Power 400㎽
-. 256 mini-ball grid array(mBGA Type)
-. 3.3V I/O Interface
-. Integrated Clock generation
-. Power-management features

-. Big and little endian operating modes
-. 32-entry MMUs
-. 32-bit command and data cache
-. Write and read buffer

2. INTEL SA-1110의 내부구성도

다음 <그림 3-2>는 SA-1110의 내부구성도이다.



<그림 3-2 > SA-1110의 내부 다이어그램
<Fig 3-2>Internal block diagram for SA-1110

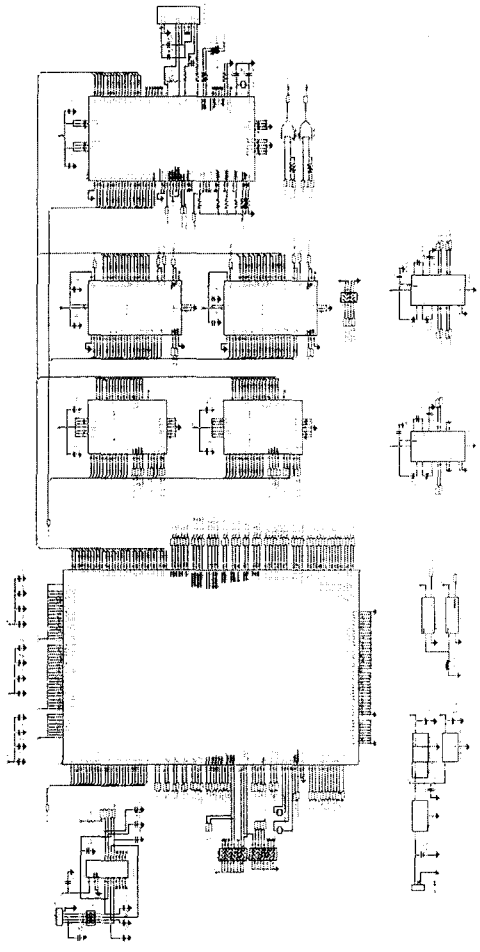
3. INTEL SA-1110의 매모리 맵 배치

SA-1110 의 메모리 공간은 크게 Static 메모리 영역과 PCMCIA 메모리 영역, 내부 레지스터 영역, 다이나믹 메모리 영역, 캐쉬 영역으로 나뉜다 평가회로의 메모리맵은 외부장치를 붙여 사용 가능한 공간은 0x00000000H ~ 0x50000000H, 0xC000 0000H ~ 0xE0000000H 까지이다. 그 외의 영역에는 내부적으로 이미 정의된 영역이거나 사용이 불가능한 영역이다. 따라서 평가보드에서 사용되는 메모리맵과 디바이스는 다음과 같다.

Device	Signal	Memory Address
Flash ROM(32M)	nCS0	0x00000000H~
CS8900	nCS2	0x10000000H~
SDRAM(32M)	nCSDC0	0xC0000000H~

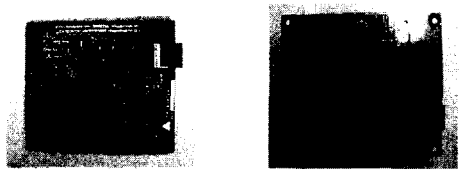
4. Intel SA-1110의 평가회로 설계 및 제작

아래 그림은 위에서 연구한 결과를 토대로, 32 비트 CPU 보드 제작기술에 의하여 설계된 Intel SA-1110의 평가회로의 완성된 도면이다.



5. Intel SA-1110의 평가회로 제작

<그림 5-1>은은 평가회로의 PCB 아트워크작업을 진행한 후 제작된 시제품을 나타낸 것이며, 다층으로 구성되었다.



<그림 5-1>제작된 평가보드의 외형
<Fig 5-1> The shape appraised board for implementation

IV. 결 론

본 연구에서는 실시간 내장형 운영체제기술을 개발하였으며, 여러 가지 운영체제중에서 리눅스를 내장형시스템에 포팅하기 위한 포팅기술 및 평가보드 회로에 대해 연구개발하였다. 본 연구에서 수행된 기술개발 결과를 요약하면 다음과 같다.

-. 리눅스를 내장형시스템에 이식(porting)하기 위한 개발방법 및 개발환경 구축
-. 이식을 위한 부트로더의 기능과 소스 분석, 평가회로에 맞는 환경변수의 수정 등
-. 평가보드에 이식 및 커널의 부팅과정까지의 디버깅 기술
-. 32비트 CPU보드 평가회로의 설계기술 및 제작기술
-. RTOS의 어플리케이션 개발을 위한 환경설정 및 기술개발

본 연구를 수행함에 있어 32비트 CPU를 기반으로 하는 내장형 리눅스 이식기술과 부트로더의 분석과 수정 등 운영체제 이식기술 및 회로설계기술을 확보하였다. 따라서, 본 연구를 기반으로 하여 차후 네트워크 어플리케이션을 위한 네트워크 접속과 디바이스 드라이버 설계기술 등에 대해서도 지속적인 연구가 확보된다면 해상통신분야의 DSC/NDBP설비의 제어기 및 기타 통신시스템 등과 같이 다양한 산업용 제어기의 설계기술을 확보할 수 있을 것으로 판단된다.

참고문헌

[1] 김문자외 2명, "네트워크 기반 대규모 임베디드시스템의 상호협동을 위한 Smart Message 기법," 정보처리학회지, pp.60~69, 2002. 1
[2] Steve furber, 「ARM system-on-chip architecture」, addison-wsley, 2000.
[3] INTEL Inc., SA-1110 User-manual, 2000.
[4] 박재희 역, 「임베디드 리눅스-하드웨어, 소프트웨어, 인터페이스", 정보문화사」, 2002.
[5] 조유근외 2명, 「커널 프로그래밍」, 교학사, 2002.
[6] 임근수, "범용 운영체제 구현을 위한 리눅스 커널 완전 분석", 연세대학교, 2001.
[7] 이호외 1명 역, 「리눅스 커널의 이해」, 한빛미디어, 2001.
[8] 권덕용외 1명, 「Linux 서버 관리」, 이비컴, 2002.
[9] 박준철 역, 「TCP/IP 소켓 프로그래밍」, 사이텍미디어, 2001.
[10] Warren W. Gay, 「Linux Socket Programming」, QUE, 2000.
[11] 노영욱외 1명, "임베디드 리눅스 개발도구 기술동향," 정보처리학회지, pp.35~42. 2002.
[12] 셈틀기획 편저, 「비주얼베이직 인터넷 프로그래밍」, 영진출판사, 2001.
[13] 박제현, 정재원 역, 「리눅스 C 프로그래밍」, 인포북, 2000.
[14] CIRRUS LOGIC INC., CS8900A Datasheet, 2001
[15] J.D&T INC., FALINUX EZ Board Mauual, 2002

저자소개

이헌택(Lee Hountaek)

인천전문대학