한국산학기술학회논문지
Vol. 4, No. 3, pp. 301-306, 2003

임베디드 웹서버를 이용한 원격 감시 및 제어 시스템 구현

최재우* · 노방현* · 이창근* · 차동현* · 황희융*

Implementation of Remote Control and Monitoring System using Embedded Web Server

Jaewoo Choi*, Banghyun Ro*, Changken Lee*, Donghyeun Cha* and Heeyeung Hwang*

요 약 임베디드 웹서버를 설계하고 리눅스 OS 를 포팅하여 원격지 하드웨어의 제어와 감시 시스템을 구현하였다. 리눅스는 2.4.1 버전을 ARM720T 보드에 포팅했으며 웹서버는 GPL(General Public License)규약인 Boa web server를 사용했다. 원격지 감시와 제어를 위해 Cirrus Logic사의 ARM720T 칩인 EP7312의 GPIO(General Purpose Inpout Output) 포트에 입출력 디바이스 연결시켜 실험하였다. GPIO 장치 드라이버를 작성하였고, 이를 구동시키는 응용프로그램은 리눅스용 C언어를 CGI프로그램화시켜 클라이언트 PC의 웹브라우저에서 제어와 감시가 가능하게 했다. 이는 기존의 PC기반의 웹서버를 사용하는 것 보다 하드웨어 설계 비용을 절감할 수 있고 운영체제없이 구현되는 웹서버보다는 응용범위의 다양성과 개발기간 단축이라는 장점을 가지고 있다.

Abstract We have designed embedded web server system and ported Linux operating system version 2.4.1 at our system. And then We implemented to control and monitor widely separated hardware. Web server is the Boa web server with General Public License. We designed for this system using of Cirrus logic's EP7312 ARM core base processor and connecting input and output device at GPIO port of EP7312. Device driver of General purpose I/O for Linux OS is designed. And then the application program controlling driver is implemented to use of common gate interface C language. User is available to control and monitor at client PC. This method have benefit to reduce the Expenditure of hardware design and development time against PC base system and have various and capacious application against firmware base system.

Key Words : Embedded linux system, Embedded web server

1. 서 론

PDA, Set-top box, 인터넷 장비 등에 대한 소비시장의 확충과 더불어 이에 필요한 기술 중 하나인 임베디드 시스템에 대한 연구가 최근 다시 활성화되고 있다[1]. 임베디드 시스템에 기존의 리소스를 재사용 가능하고 라이센스를 지불해야하는 고가의 상용화된 운영체제 사용하지 않고 이에 필적하는 안정성을 갖는 시스템을 개발한다면 비용면에서 생산자와 소비자의 이득은 매우 클 것으로 사료된다. 라이센스 비용이없는 GNU(GNU is not UNIX)와 GPL 정책을 따르는 LINUX 운영체제가 임베디드 시스템에서 적용되는 예가 늘어나고 있다[2].

*호서대학교 전자공학과
Tel: 041-548-5747, E-mail: cjwy@chol.net

본 논문은 EP7312라는 Cirrus logic사의 32비트 ARM7TDMI 코어를 채용한 프로세서로 웹서버 시스템을 제작하고 LINUX 2.4.1 버전을 포팅하여 사용자가 집안의 PC에 기본적으로 설치되어있는 웹브라우저를 이용해 원격지의 전기적인 신호에 대한 모니터링과 제어가 가능토록 했다. 포팅과정중 보드패치의 핵심 사항은 하드웨어의 가상메모리와 물리메모리를 중복없이 그리고 하드웨어가 설계된데로 정확하게 할당해주어야 리눅스가 부팅될 때 패닉상태에 들어가지 않는다는 것이다. 본 논문에서 사용된 웹서버는 임베디드용으로 적합한 Boa web 서버를 ARM용으로 컴파일하여 사용하였다. 리눅스 운영체제가 실행되고 있는 상황에서는 하드웨어 자원을 사용자 응용프로그램에서 직접 엑세스가 불가능하기 때문에 GPIO 디바이스 드라이버를 작성하여 CGI 응용프로그램이 제어와 감시의 접점이 되는 GPIO 포트에 접근하도록 하였다.

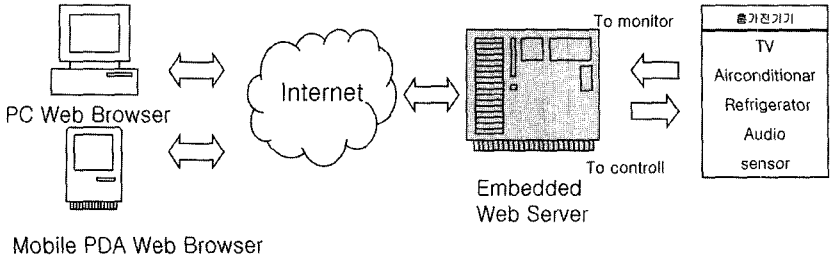

그림 1. 전체 시스템 구성도

2. 본 론

사용자는 인터넷이 사용 가능한 PC나 모바일 기기의 웹 브라우저에서 구현된 웹서버 주소에 접속하면 출력 디바이스를 On/Off 할 수 있는 GUI(Graphic User Interface)화면이 보이게 된다. 사용자가 원하는 디바이스 제어에 대한 입력을 선택하고 전송버튼을 누르면 원격지의 현재 입력 디바이스의 상태가 웹 브라우저에 표시되고 원격지 디바이스가 제어/감시 된다.

그림 1은 본 논문에서 구현된 전체 시스템의 개요가 되는 블록도이다.

2.1 하드웨어 구성

내장형 웹서버로 사용된 타겟 시스템은 Cirrus logic 사의 EP7312 CPU를 사용하여 구성하였다. 표 1은 타겟 시스템의 외부 부트 모드의 메모리 맵이다.

외부 부트 롬의 시작번지가 리셋번지(0x00000000)로 할당되고 GPIO 포트의 번지가 포함되어있는 내부레지스터 영역은 0x80000000 번지부터 할당되어있고 이더넷 통신칩인 CS8900의 칩 선택 번지는 0x40000000로 설계하였다. EP7312의 내부 부트 롬에는 내부 SRAM에 부트 코드를 프로그램 할 수 있도록 지원해주는 시리얼 통신프로그램이 내장되어있어서 외부의 부트 롬의 종류에 따른 부트 코드를 프로그래하게 되어있다. 이때에는 EP7312를 내부 부트 모드로 설정해야 하고 리셋 번지는 내부 부트 롬이 선택되도록 메모리 맵이 바뀌게 된다. 본 논문에서는 인텔 스트라타 플래시를 외부 부트 롬으로 사용해서 리눅스 운영체제를 저장하는 메모리로 사용하였다.

표 1. 웹서버 보드의 외부 부트모드 메모리 맵

Address	Contents	EP7312 BD
0xF000.0000 - 0xFFFF.FFFF	Reserved	Unused
0xE000.0000 - 0xEFFF.FFFF	Reserved	Unused
0xD000.0000 - 0xDFFF.FFFF	Reserved	Unused
0xC000.0000 - 0xCFFF.FFFF	SDRAM	SDRAM
0x8000.4000 - 0x8FFF.FFFF	Unused	Unused
0x8000.0000 - 0x8000.3FFF	내부 레지스터	내부 레지스터
0x7000.0000 - 0x7FFF.FFFF	내부 ROM(CS7)	내부 부트롬
0x6000.0000 - 0x6FFF.FFFF	내부 SRAM (CS8)	SRAM
0x5000.0000 - 0x5FFF.FFFF	Expansion nCS[5]	ETHERNET 2
0x4000.0000 - 0x4FFF.FFFF	Expansion nCS[4]	ETHERNET 1
0x3000.0000 - 0x3FFF.FFFF	Expansion nCS[3]	EXT GPIO
0x2000.0000 - 0x2FFF.FFFF	Expansion nCS[2]	USB
0x1000.0000 - 0x1FFF.FFFF	ROM Bank1 nCS[1]	NAND FLASH
0x0000.0000 - 0x0FFF.FFFF	ROM Bank0 nCS[0]	외부 부트롬

2.2 리눅스 커널 포팅작업

리눅스가 부팅된 이후에 커널상의 모든 메모리 관리는 가상메모리로 접근하도록 되어있다. 그리고 리눅스 포팅시 이를 위해 가상메모리와 물리메모리를 자신의 보드에 맞게 중복 없이 매핑 시키는 작업이 중요하다. 하드웨어의 실제 물리주소와 커널소스상의 주소가 다르거나 중복되어있다면 리눅스 자체가 부팅 되지 않고 패닉상태나 메모리 폴트를 만나게 된다.

리누즈 토발즈의 리눅스 커널은 원래 i386으로 되어있고 이를 ARM 프로세서로 패치된 리눅스 커널을 다운받아 이를 보드 레벨에서의 하드웨어의 특성에 맞추어 다시 패치 작업을 본 연구에서 수행했다.

참고로 Russell King, Nexus Electronics Ltd. 등은 리눅스 커널에 ARM 프로세서를 적용시킨 공헌자들이다 [3, 4]. 보드 패치시 중요한 커널 수정부분은 다음과 같다.

linux/include/asm-arm 경로의 헤더화일안에는 보드 전반에서 사용하는 하드웨어의 가상주소와 물리주소를 할당하는 부분을 하드웨어가 설계된 번지와 일치시켜 주어야한다. 다음에는 ARM 프로세서의 메모리관리와 관계가 있는 파일이 들어있는 linux/arch/arm/mm 경로의 소스에서 그 주소를 구조체에서 사용하도록 되어있다. 또한 각종 디바이스 드라이버의 소스파일이 들어있는 디렉토리인 linux/drivers내의 소스에서 자신의 보드의 하드웨어에 맞게 제어 신호 핀이 어느 것인지 살펴 하드웨어

번지나 제어 핀에 관계된 내용을 수정해주어야 한다.

다음은 linux/include/asm-arm/arch-cal/cal-hardware.h 경로의 하드웨어 가상번지와 물리번지가 정의되고 있는 포팅된 소스이다.

```
#define IO_BASE          0xd0000000  /* virtual address*/
#define IO_START         0x80000000  /* physical address*/
#define IO_SIZE          0x00100000  /* memory size */
#define ROM_BASE         0xe0000000
#define ROM_START        0x00000000
#define ROM_SIZE         0x00800000
#define ETH_BASE         0xc1000000
#define ETH_START        0x20000000
#define ETH_SIZE         0x00100000
#define KBD_BASE         0xc2000000
#define KBD_START        0x30000000
#define KBD_SIZE         0x00100000
#define NCS5_BASE        0xc3000000
#define NCS5_START       0x50000000
#define NCS5_SIZE        0x00100000
```

다음 코드는 linux/arch/arm/mm/mm-cal.c 경로의 하드웨어 가상번지와 물리번지를 초기화시키기 위한 데이터가 구조체 테이블에 할당되는 소스이다.

```
static struct map_desc io_desc[] __initdata = {
{ IO_BASE, IO_START,  IO_SIZE,     DOMAIN_IO, 0, 1 },
{ ETH_BASE,  ETH_START,  ETH_SIZE,  DOMAIN_IO, 0, 1 },
{ANDFLASH_BASE, NANDFLASH_START,NANDFLASH_SIZE, DOMAIN_IO, 0, 1 },
{ ROM_BASE,  ROM_START,  ROM_SIZE,     DOMAIN_IO, 0, 1 },
LAST_DESC
};
```

2.3 보아 웹서버

보아 웹서버는 Single-tasking 기반의 HTTP(Hyper Text transfer protocol)서버이다[5].

본 연구는 원격지 하드웨어 자원을 감시/제어를 목적으로 리눅스에 웹서버를 선택하는데 있어 웹서버 프로그램 자체의 크기가 임베디드 시스템처럼 기억장치의 용량이 부족한 시스템에서도 작은 용량으로 설치가 가능한 보아 웹서버를 ARM용 크로스 컴파일러로 컴파일하여 루트이미지 안에 포함시켰다. 여기서 루트이미지란 롬에 저장된 리눅스의 디렉토리와 파일의 내용자체를 의미한다.

2.4 GPIO 디바이스 드라이버

EP7312 프로세서의 GPIO B port를 사용하여 상위 4핀은 LED를 연결하여 원격제어 실험의 검증에 사용되었고 GPIO B port의 하위4핀은 push button switch를 연결하여 원격감시의 검증에 사용되었다.

GPIO 디바이스 드라이버는 리눅스의 캐릭터 디바이스 드라이버 작성법을 기초로 작성하였다[6]. 디바이스 드라이버는 순차방식이 아닌 이벤트방식으로 처리되어 나중에 커널에 삽입될 수 있는 모듈 프로그램 이다. 모듈이 리눅스 커널에 적재/제거 되려면 insmod /rmmod 라는 리눅스 명령어를 사용하는데 이때 실행되는 init_module()/cleanup_module() 함수 안에서 register_chrdev()/unregister_chrdev() 함수에 의해 커널에 등록/제거 된다. 디바이스 드라이버가 커널에 적재된 이후에 mknod 명령을 사용하여 GPIO 디바이스 장치 파일을 /dev 디렉토리 밑에 생성시키면 사용자 메모리 공간상의 응용프로그램이 장치화일에 접근하여 읽고 쓰기가 가능해 진다. 디바이스 드라이버도 역시 커널에 삽입되어 동작하는 커널의 일부이기 때문에 linux/arch/arm/mm/에 위치해있는 메모리 관리 소스파일의 메모리 디스크립터 구조체인 map _desc에 명시되어있는 가상메모리로 접근해야만 한다.

본 연구에서는 EP7312의 내부레지스터의 물리주소 베이스 번지가 0x80000000인 것을 가상주소 0xd000 0000 에 매핑해서 사용하였다[7].

CGI 응용프로그램은 먼저 디바이스를 오픈시키고 read(), write() 함수에 의해 디바이스 드라이버의 gpio_read(), gpio _write()함수를 실행하여 gpio 장치화일을 읽고 쓰는 것이 가능해진다.

gpio_read() 함수는 먼저 GPIO B port의 디렉션 레지스터를 읽기(0)로 하고 현재 포트에서 읽어들인 비트 데이터를 변수에 저장해놓고 copy_ to_user() 함수를 통해 커널공간의 변수값을 유저공간의 버퍼인 포인터형 변수로 넘겨준다. 또한 gpio_ write() 함수는 먼저 GPIO B port의 디렉션 레지스터를 쓰기(1)로 하고 copy_from_user() 함수를 통해 유저공간의 버퍼값을 변수에 저장하여 그값을 GPIO 포트에 출력시키게 된다.

다음 코드는 GPIO 디바이스 드라이버의 원시소스인 gpio.c의 gpio_read() 함수의 내용이다.

```
ssize_t gpio_read (struct file *filp,  char *buf, size_t count, loff_t
*f_pos)
{
        unsigned char data = 0;
        switch (count) {
        case PADR : IO_PADDR = 0;
                data = IO_PADR;
                break;
        case PBDR : IO_PBDDR = 0;
                data = IO_PBDR;
                break;
        case PDDR : IO_PDDDR = 0xFF;
                data = IO_PDDR;
                break;

        case PEDR : IO_PEDDR = 0;
                data = IO_PEDR;
                break;
        default :   printk("GPIO read error\n");
                break;
        }
    copy_to_user(buf, &data, 1);
    return count;
}
```

2.5 CGI 응용프로그램

CGI(Common Gateway Interface)란 웹서버와 외부 프로그램간에 서로 데이터를 주고 받기 위한 표준규약으로서 이질적인 두 프로그램간에 데이터를 주고받기 위한 공통된 데이터 입출력 통로를 의미한다[8-10].

CGI에 사용되는 언어는 JAVA, perl, c, shell script 등 무엇이든 상관없지만 여기서는 디바이스 드라이버에 접근하는 코드가 포함되므로 저 수준 제어에 용이한 C 언어를 사용하였다.

본 논문에서 구현한 CGI 프로그램에 기술된 내용 디바이스 드라이버를 접근하는 부분과 HTML 코드를 printf()로 처리하여 웹브라우저에 정보를 표시하는 부분으로 나뉘어져 있다.

먼저 open() 함수로 GPIO 디바이스 화일을 열고 write() 함수로 html 문서에서 넘어온 하드웨어 제어 데이터를 장치파일에 출력시킨다. 이때 write() 함수는 디바이스 장치 프로그램의 gpio_write()를 접근하여 실제 물리 디바이스에 접근하게 된다. 같은 방법으로 장치에서 값을 읽을 경우에는 이미 열려있는 디바이스에 read() 함수를 통해 디바이스 장치 화일의 gpio_read()함수가 실행되어 실제 물리 디바이스에서 전압 값을 읽게된다.

HTML이 CGI 프로그램에 파라메터를 넘겨주는 방법은 GET 방식과 POST 방식을 사용할 수 있는데 여기서는 URL과 함께 인코딩된 정보로 전송되는 GET 방식을 사용하였다.

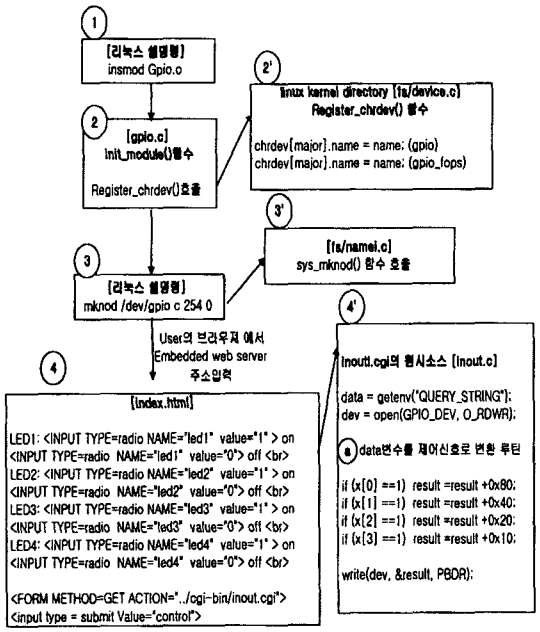

그림 2. 제어동작의 플로우차트

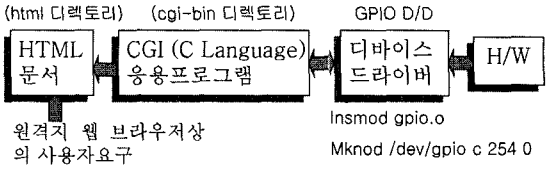

그림 3. 제어감시 시스템의 신호전달 체제

그림 2는 제어 화면 동작에 대한 소스와 명령어 관점에서 바라본 흐름도 이고 그림 3은 웹서버 시스템의 신호 흐름 관계를 나타낸 그림이다.

2.6 제어/감시 실험

그림 2와 그림 3은 사용자의 웹브라우저 화면에 나타나는 제어/감시 화면이다. 제어처리에 대한 HTML과 프로그램 핵심코드는 그림 2의 ④, ④'에 있고 감시 처리시에는 read(dev, (char *)&data1, PBDR); 함수를 write() 함수대신 사용하여 data 1의 내용을 화면에 보여줌으로써 원격지의 하드웨어에 대한 제어와 감시가 가능토록 하였다. 다음 코드는 그림 2의 ④'-ⓐ소스에 해당하는 것으로서 index.html이 inout.cgi에 넘겨준 파라메터를 실제 제어신호만을 배열변수에 저장해놓는 루틴이다.

```
for(i=0;i<MAX;i++)
{       if (data[i] == '&' || data[i] == '\0')
        {       a = data[i-1];
                x[index] - a-48;
                printf("<p> data[%d] = %d", index, x[index++] );
        }
}
```

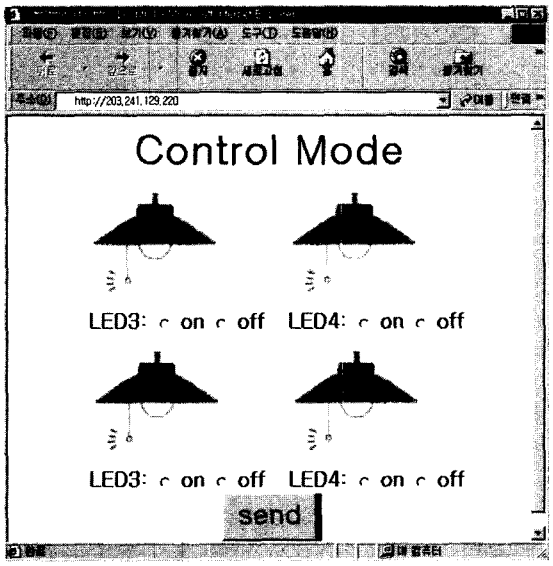

그림 4. 제어 화면

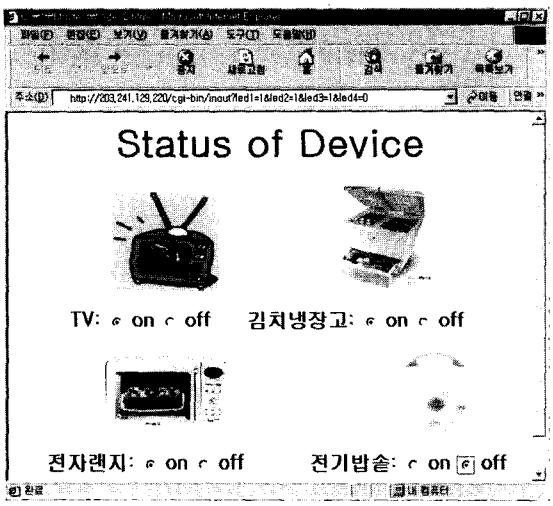

그림 5. 감시 화면

그림 4에서 send 버튼을 누르게 되면 현재의 HTML 문서가 CGI를 실행하게되는데 이때 인자전달방식이 GET 방식이므로 인자는 URL 뒤에 "led1=1&led2=1&led3=1&led4=0"이라는 문자열 값이 "query_stri ng" 이라는 환경변수로 넘어 오게된다. 위의 코드는 "&" 문자와 문자열의 끝인 널문자 바로 앞에 각 파라메터의 실제 값이 위치해 있는 것에 착안하여 구성한 코드이다.

그림 5는 감시화면을 보여주고 있으며 전반적으로 수행되는 코드는 제어동작의 경우와 유사하지만 CGI 프로그램에서 read() 함수를 사용하여 GPIO 포트를 읽어내어 표시해주는 부분만 다르게 처리하였다.

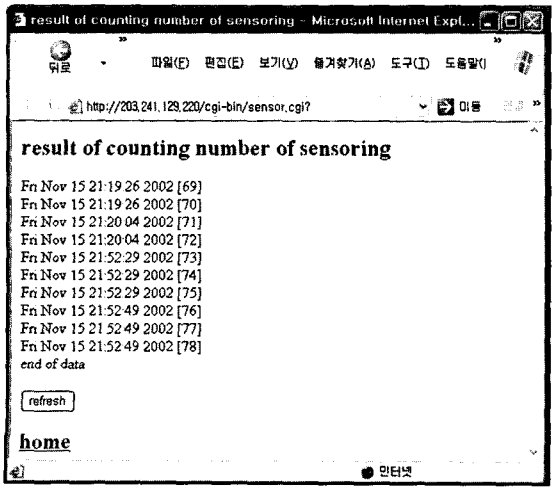

그림 6. 센서감시 화면

2.7 센서신호 감지 실험

운전을 하다보면 특정 도로의 시간대별 통행량을 사람을 고용해서 일일이 채크하는 것을 본적이 있다. 본 실험은 위와 같은 작업을 원격지에 임베디드 웹서버를 두고 유저의 컴퓨터에서 통행량을 모니터링 해볼수 있도록 적외선 센서를 웹서버 보드의 GPIO 포트에 연결해서 실험하였다. 원격지의 웹서버보드에 연결된 적외선 센서 사이에 물체가 지나가면 그때의 시간과 누적된 카운터 개수를 텍스트 화일에 실시간으로 저장해놓고 브라우저에 원격지 웹서버의 주소를 입력하면 그림 6과 같은 화면이 보이고 refresh 버튼을 누르면 현 시간까지의 통행량을 초단위로 누적해서 출력하도록 하였다.

3. 결 론

본 논문에서는 리눅스를 운영체제로 하는 내장형 웹서버를 설계하여 원거리에 있는 하드웨어 자원을 제어하고 감시할 수 있는 시스템을 구현하였다.

내장형 웹 서버 보드는 ARM720T 코어기반인 Cirrus logic사의 EP7312 프로세서를 사용한 보드에 리눅스 2.4.1 버전을 보드에 맞게 커널을 수정하여 포팅작업을 수행했고 Boa Web server를 루트이미지에 삽입하여 웹서버를 구현했다. 또한 EP7312의 GPIO 포트에 대한 디바이스 드라이버를 리눅스의 문자디바이스 드라이버로 작성해서 CGI 응용프로그램이 접근해서 원격지에 연결된 하드웨어의 제어와 감시가 가능하도록 하였다.

본 시스템의 장점은 다음과 같다. 첫째, 운영체제와 웹서버 프로그램이 상용화되어있는 것이 아니기 때문에 비용을 줄일 수 있다는 것이고 둘째, 웹서버 보드가 PC 기반이 아닌 내장형 시스템으로 설계 되어있어 소형화가 가능하고 이동이 자유롭다는 장점이 있고 또한 LINUX OS의 리소스를 활용할 수 있기 때문에 OS없이 firmware로만 작업하는 것보다 개발시간이 단축된다는 장점이 있다.

앞으로 PDA 등과 같은 무선 인터넷 접속방식이 더욱 활성화되면 장소에 상관없이 원격지의 하드웨어의 감시/제어가 가능하리라고 본다.

가능한 응용 분야로서는 HA(Home Automation), FA(Factory Automation), ITS(Intelligent Transport Systems) 등에 적용될 수 있다.

추후 연구과제는 USB 버스나 IEEE1394와 같은 고속 버스를 이용한 카메라를 내장형 웹 서버 상에서 동작시켜 더욱 광범위한 제어/감시 시스템을 구현하는 것이다.

참고문헌

[1] 이민석, "모바일 기기를 위한 임베디드 리눅스", 정보처리학회지, 제9권 제1호, 2001년 1월.
[2] John Lombardo, "Embedded Linux", Informa tion Publish, 2002. 2.
[3] http://www.arm.linux.org.uk/personal/aboutme .html
[4] http://www.nexus.co.uk/services/software.htm
[5] www.boa.org
[6] Alessandro rubini, "Linux device drivers", O'Reilly, 2000.
[7] EP73XX User's Guide, Cirrus Logic, 2001.
[8] http://eekim.com/pubs/cgiinc/
[9] http://hometown.weppy.com/~deadfire/cgic
[10] 김홍남, "CGI 파워프로그래밍", 도서출판 대림, 1998년 1월.