임베디드 개발자 입문

2000년대는 포스트 PC, 즉 임베디드 시스템의 시대라고 해도 과언이 아니다. 많은 임베디드 시스템의 사양이 32비트 프로세서를 사용하는 시스템으로 급격히 변화되고 있다. 특집 1부에서는 임베디드 개발자가 되기를 원하는 독자들을 위하여 임베디드 분야와 다른 분야의 차이점이 무엇이며, 임베디드 개발 자체가 무엇인가와 임베디드 개발자가 되기 위해서 무엇을 준비해야 하는가를 설명하고자 한다.

프로그래밍은 잘하는데 하드웨어를 잘 몰라도 임베디드 시스템 개발자가 될 수 있을까? 전자공학을 전공했는데 임베디드 시스템 소프트웨어 프로그램을 잘할 수 있을까? 필자 또한 임베디드 시스템을 처음 시작할 때 이와 같은 심정이었다. 필자는 학부와 대학원에서 전자통신을 전공하였다. 그리고 학부 때 배운 컴퓨터 언어는 포트란이 전부였다. 그것도 컴퓨터 없이 칠판수업을 받았다. OS와 32비트 프로세서를 접하기 시작한 것도 서른 살이 넘어서였으며, 프로그램 언어도 8비트 마이콤용 어셈블리 정도만 사용하는 수준이었다. 상상이 가는가?

필자는 임베디드 리눅스 기반 프로젝트를 다년간 수행하였으며, 현재 임베디드 리눅스 교육을 짧게는 5일 길게는 6개월 과정을 주관하며 가르치고 있다. 이 글은 그러한 경험을 바탕으로 임베디드 개발자가 되기 위해 필요한 기술적인 범위에 대해 말하고자 한다.

먼저 PC 환경 개발과 임베디드 시스템 환경 개발과의 차이를 다룰 것이며, 임베디드 시스템이 무엇인가를 효율적으로 설명하기 위해서 밥솥이라는 가전제품을 예로 들어 임베디드 시스템 개발자가 되기 위해서 필요한 것이 무엇인가를 말하고자 한다.

PC와 임베디드 시스템 환경의 개발 차이

두 시스템에서 개발 환경의 차이는 소프트웨어 개발 환경의 차이와 소프트웨어 개발 범위의 차이 등이 있다.

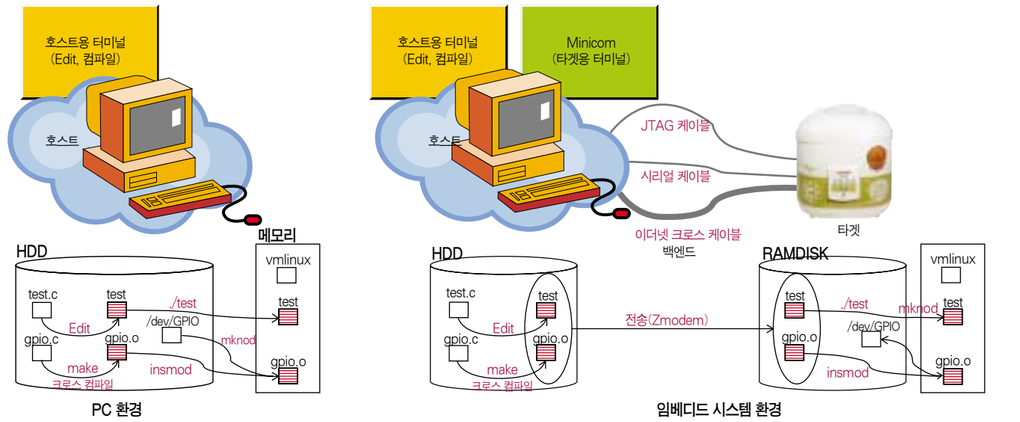
현재까지 대부분의 소프트웨어 개발자들은 PC에서 작업을 수행하여 왔다. <그림 1>과 같이 기존의 소프트웨어 개발과정은 소스 코딩을 하고 컴파일하고 실행하는 형태가 모두 PC 환경에서 이루어졌다는 것이다. 하지만 임베디드 시스템 소프트웨어 개발 과정은 소스 코딩을 하고 컴파일을 수행한 후 이를 타겟 시스템으로 전송한다. 그리고 전송된 실행 코드를 타겟에서 실행한다. 기존의 소프트웨어 개발자가 가장 먼저 어려움을 겪고 있는 것은 개발 환경이 다르다는 것이다.

임베디드 시스템에서 프로그램의 대상이 PC가 아닌 타겟 보드라는 것이다. PC 소프트웨어 개발은 별도의 보드가 필요 없이 PC상에서 모든 것이 이루어지기 때문에 보드가 없어도 개발이 진행된다. 그러나 임베디드 시스템의 소프트웨어 개발은 타겟 보드가 필요하다. 그러므로 개발 보드가 없이는 임베디드 환경에서 소프트웨어 개발이 불가능하다.

박철 | pc@hybus.net

임베디드 기반의 개발과 특히 임베디드 교육에 많은 관심을 가지고 있으며, 이에 대한 사업화 방안에 관해 항상 고심하고 있는 사람이다. 실제 강의 현장에서 수강생들과 함께 호흡하는 길 좋아하며 지인들과의 술자리를 즐기는 편이다. 개인적인 바람이라면 운전에 능숙해지고 싶다고.

<그림 1> PC와 임베디드 시스템 환경



그리고 소프트웨어 개발자라 할지라도 타겟 보드에 대한 지식이 필요하다. 타겟 보드의 CPU 구조에 대한 이해와 어셈블러, 개발 보드의 주변장치의 이해 등이 필요하다. 이러한 것이 순수 소프트웨어만을 하던 개발자들이 겪는 어려움이라고 할 수 있다.

현재의 소프트웨어 개발 환경은 임베디드 환경으로 급속히 변해가고 있기 때문에 이러한 개발 환경의 이해 없이는 개발에 매우 큰 어려움을 겪을 수 있다. 이렇게 임베디드 시스템 소프트웨어 개발자가 되기 위해서는 개발 환경을 이해하고 이러한 시스템을 PC를 사용하듯이 능숙히 조작하는 것이 첫 번째 과제라고 할 수 있다.

SW 개발 범위의 차이

PC에서 소프트웨어 작업은 커널이나 디바이스 드라이버가 이미 준비된 상태에서 애플리케이션 작업만을 수행하는 것이었다. 물론 하드웨어도 거의 비슷한 형태로 제공된다. 임베디드 시스템에서는 사용되는 CPU가 다르고 이를 위한 개발 환경을 구축해야 하며, 직접 커널을 포팅해야 한다. 그리고 디바이스 드라이버도 해당 시스템에 맞게 제작하여야 한다. 그 이후에야 애플리케이션 프로그램을 작성한다.

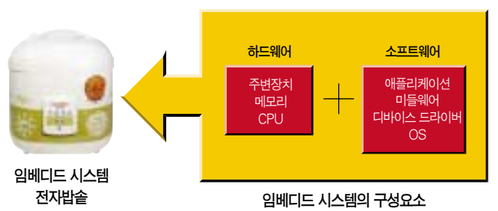
임베디드 시스템이란 무엇인가?

임베디드 시스템은 우리 생활에서 쓰이는 각종 전자기기, 가전제품, 제어장치 등을 말한다. 이러한 장비의 특징은 단순히 전기, 전자회로로만 구성된 것이 아니라 마이크로프로세서가 내장되어 있다는 것이다. 이렇게 내장된 마이크로프로세서는 시스템을 구동하여 그 장비가 해야 하는 특정한 기능을 수행하도록 프로그램이 내장되어 있는 시스템을 가리킨 것이다.

이러한 임베디드 시스템은 산업, 가전, 사무, 군사 등의 다양한 응용 분야를 가지고 있으며 적용 사례도 휴대폰, PDA, 사이버 아파트의 홈 관리 시스템, 홈 네트워크 게이트웨이 장치, 교통관리 시스템, 주차 관리시스템, 홈 관리 시스템, 엘리베이터 시스템, 현금지급기(ATM), 항공 관제 시스템, 우주선 제어 장치, 군사용 제어 장치 등 다양한 곳에 응용이 된다.

임베디드 시스템을 좀더 쉽게 이해하기 위해서 생활가전 제품인 밥솥을 예로 들어 임베디드 시스템을 설명하고자 한다. 다음은 밥솥의 발전 단계를 나타낸 것이다.

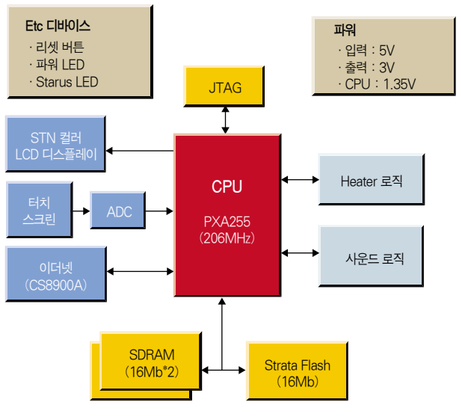
<그림 2> 임베디드 시스템의 구성



1 전기 회로로만 구성된 전기밥솥(초기)
2 간단한 4비트 또는 8비트 마이콤이 내장된 전자밥솥(현재)
3 일반 PC 같은 기능을 가진 32비트급 프로세서가 내장된 전자밥솥(미래)

1은 단순히 히터라는 전기회로를 통해서 열을 가하여 밥을 짓는 장치이다. 1은 사람이 모든 제어를 수행하며, 독립적인 기능을 수행할 수 없기 때문에 임베디드 시스템이라 할 수 없다. 2와 3은 독립적으로 기능을 수행할 수 있기 때문에 임베디드 시스템이라 말할 수 있다. 이렇듯이 임베디드 시스템은 CPU가 내장되어 있어서 시스템의 기능을 제어하는 시스템이라 할 수 있다. 2와 같은 경우는 사람이 하던 제어 기능을 4비트 또는 8비트의 마이콤이 펌웨어를 통해서 수행하도록 하도록 하였다. 여기서 펌웨어란 OS 없이 동작되는 프로그램을 말하며, 4비트 8비트 마이콤에서 사용된다. 즉 프로세스가 하나라는 의미이다. 3은 밥을 하는 고유 기능 이외에 외부와 네트워크로 연결되어서 밥을 짓는 정보를 영상이나 음성 정보를 통해서 전달받으며 이를 사용자가 쉽게 사용할 수 있도록 GUI 환경을 제공하는 기능이 추가된 시스템을 말한다. <그림 2>와 같이 32비트를 사용하며 여러 개의 프로세스가 존재하기 때문에 펌웨어를 사용할 수 없고, OS 환경하에 동작하는 애플리케이션이 동작되는 시스템이다. 그리고 통신 기능을 구현하여야 하기 때문에 TCP/IP와 같은 프로토콜이 구현되어 있어야 하며 그래픽 LCD 기반의 GUI 환경이 기본적인 시스템이기 때문에 시스템 규모가 2와는 매우 다르다. 마치 PC 하나가 밥솥에 들어가 있는 형태라고 할 수 있다.

<그림 3> 임베디드 시스템 하드웨어 구조



3과 같이 임베디드 시스템은 2000년대로 들어서면서 트렌드의 변화가 오기 시작하였다. 2와 같이 마이콤에 펌웨어 기반의 고유 기능을 가지고 있는 것이 기존의 임베디드 시스템이었다면 3은 고유 기능 이외에 부가 기능을 가지고 있으며, 이러한 부가 기능은 많은 정보량을 요구하기 때문에 단순한 8비트의 구조를 가지고 사용할 수 없으며 32비트 프로세서 기반은 여러 개의 애플리케이션을 운영하기 위한 운영체제를 필요로 하게 된다. 이러한 임베디드 시스템의 수요는 날로 커져가고 있다.

현재 임베디드 시스템의 특징은 PC가 가지고 있는 기능이 임베디드 시스템에 적용된다는 것이다. 그래서 이를 포스트 PC라고도 하고, 가전에 정보 전달 기능이 강화된 형태를 유지하기 때문에 이를 정보가전이라고도 표현하고 있다.

임베디드 시스템 개발 과정

임베디드 개발 과정은 크게 3가지로 나눌 수 있다.

1 임베디드 시스템 하드웨어 개발 과정
2 임베디드 시스템 교차 개발환경 구축 과정
3 임베디드 시스템 소프트웨어 개발 과정

임베디드 시스템의 개발 과정을 일반 PC 같은 기능을 가진 32비트급 프로세서가 내장된 전자밥솥을 예로 설명하고자 한다. 개발 순서는 먼저 밥솥의 기능을 정하고 그에 맞는 하드웨어와 소프트웨어 기능을 협의한 후 각각의 기능을 구현하는 것이다. 개략적으로 하드웨어의 32비트 CPU는 XScale 기반 PXA255이고, 운영체제는 임베디드 리눅스를 사용하기로 한다.

임베디드 시스템 HW 개발 과정

<그림 3>은 밥솥의 하드웨어 구성을 나타낸 것이다. CPU는 ARM 기반의 인텔에서 제공하는 PXA 255를 사용하였으며 메모리는 SDRAM 32MB, 플래시 16MB를 사용하였으며 밥솥의 기능을 수행하기 위해서 히터 로직(Heater Logic)이 있으며 외부에 TCP/IP 기반으로 통신하기 위해서 이더넷 컨트롤러를 달았다. 그리고 사용자 인터페이스를 위하여 3.5인치 TFT LCD를 달았고, 터치스크린을 통하여 사용자로부터 입력을 받을 수 있도록 하였다. 그리고 사운드 로직을 통하여 음향 및 음성 정보를 사용자에 전달할 수 있도록 하드웨어를 설계하였다.

여기서 사용되는 PXA 255는 400MHz의 속도를 가진 고성능 CPU이다. 이는 몇 년 전 PC의 CPU 속도와 같은 수준의 CPU인 것이다. 그리고 메모리 또한 PC의 메모리와 비교해도 손색이 없을 정도의 용량이다. 일반 PC는 파일 시스템을 하드디스크에 구현하지만 여기서는 플래시 메모리에 구축하는 것으로 하였다.

크로스 개발 환경 구축과정

하드웨어 제작이 끝나면 교차(cross) 개발 환경을 구축하여야 한다. 크로스 개발 환경이란 호스트에 타겟 디바이스용 리눅스를 개발하기 위한 모든 환경을 말한다. 그리고 부트로더를 컴파일하여 해당 코드를 플래시 메모리에 넣어야 한다. 그 다음이 부트로더를 수정하여 원하는 기능을 구현한다. 임베디드 시스템의 개발 환경은 크게 3개 사항을 고려하여야 한다.

1 컴파일 환경인 XScale용 크로스 툴 체인(tool chain)
2 부트로더를 플래시에 올리기 위한 JTAG fusing 시스템
3 부트로더 제작

개발 환경의 설명에 앞서 용어를 정리하고 설명하기로 하자 <그림 1>의 임베디드 시스템 환경에서 나오는 용어를 설명하면 다음과 같다.

◆ **타겟 디바이스** : 개발하고자 하는 임베디드 시스템 보드, 여기서는 전자밥솥이 하드웨어이며, 사양은 <그림 3>과 같다.
◆ **호스트 시스템** : 타겟을 개발하기 위한 환경을 제공하는 시스템으로서 교차 컴파일러, 모니터, 디버거 등을 제공하며, 일반적으로 PC가 호스트가 된다.
◆ **백엔드(backend)** : 호스트와 타겟이 통신을 하기 위한 매개체로, <그림 1>에서와 같이 시리얼은 통신 에뮬레이터를 통해 타겟과 통신할 수 있는 통신 채널을 제공한다. 패러럴은 JTAG을 통해서 플래시에 fusing할 수 있는 통신 채널을 제공한다. 이더넷은 zImage, root filesystem image를 호스트에서 타겟으로 다운로드할 수 있는 통신 채널을 제공한다.
◆ **타겟 터미널** : 타겟의 상황을 호스트에 표시해 주는 프로그램, 즉 통신 에뮬레이터를 말한다.

컴파일 환경 - XScale용 크로스 툴 체인

먼저 해당 CPU에 맞는 툴 체인 환경을 구축해야 한다. 툴 체인이란 타겟 디바이스의 소프트웨어 개발을 진행하기 위해 필요한 호스트 시스템의 크로스 컴파일 환경을 말한다. 툴 체인은 각종 소스들을 컴파일하고 빌드해 실행 바이너리를 생성하는 데 필요한 각종 유틸리티 및 라이브러리의 모음이다. 기본적으로 어셈블리, 링커, C 컴파일러, C 라이브러리 등으로 구성되어 있다. 여기서는 GNU에서 제공하는 툴 체인을 사용하며 다음과 같다.

- GNU gcc compilers for C, C++
- GNU 바이너리 유틸리티(어셈블러, linker various object file utilities)
- GNU C 라이브러리

XScale에 사용하기 위한 ARM 툴 체인은 다음과 같은 사항으로 구성되어 있다.

- binutils-arm-2.11.2 : 유틸리티
- gcc-arm-2.95.3 : 컴파일러
- glibc-arm-2.2.3 : 라이브러리

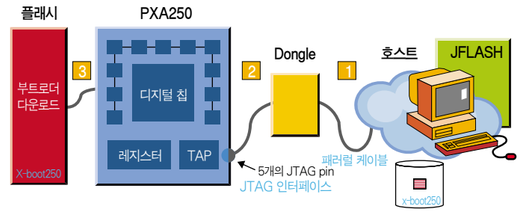
부트로더를 플래시에 올리기 위한 fusing 시스템

일반적으로 JTAG(Joint Test Access Group)란 말보다 Boundary-Scan이란 말이 더 많이 사용된다. 그럼 Boundary-Scan이라는 '주변을 훑어보다'라고 할 수 있다. 먼저 Boundary-Scan의 역사를 살펴보자.

◆ 1980년대 후반의 JTAG라는 곳에서 연구 중이던 Boundary-scan 설계를 IEEE에서 1990년에 표준화하였고 IEEE std 1149.1가 제정되었다.

<그림 4>와 같이 호스트 시스템으로부터 타겟 보드에 있는 스트라타(strata) 플래시에 부트로더를 쓰기 위해서는 그림과 같은 구성이 필요하다. 먼저 호스트 시스템의 구성부터 살펴보면 호스트 시스템에서 동작하는 jflash라는 프로그램이 필요하다. 이 jflash는 패러럴 포트를 이용하여 타겟 보드에서 필요로 하는 JTAG 신호를 생성한다. 호스트 시스템에서 생성된 JTAG 신호는 패러럴 케이블을 통해 동글(dongle)에 전달된다. 동글은 TTL 74HCT541을 사용하여 구현하였으며, 이 기능은 단지 호스트 시스템의 패러럴 포트에서 발생한 5V 전압을 PXA255에 적합한 3.3V 전압 레벨로 변환하는 기능이다.

<그림 4> JTAG을 이용한 플래시 메모리 퓨징



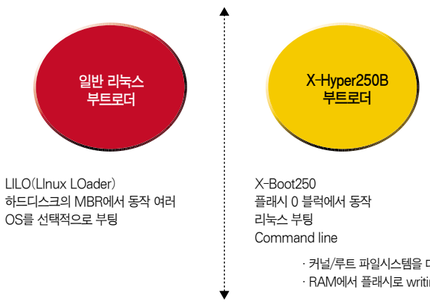
동글을 통해서 전달된 JTAG 신호 중 TMS와 TCLK는 TAP에 전달되어 JTAG의 state 머신을 결정하고, TDO와 TDI는 JTAG의 명령의 상태에 따라 bypass register, boundary scan cell, ID 레지스터 등의 입력과 출력 부분에 연결된다. PXA255의 JTAG을 통해서 플래시 메모리가 필요로 하는 버스 타이밍(bus timing)을 발생하여 플래시에 전달한다.

호스트 시스템의 셸(shell)에서 jflash blob을 입력하면 x-boot250의 바이너리 코드가 플래시 메모리의 0번지부터 퓨징(fusing)을 한다. 메모리에 쓰기 전에 기본적으로 쓰고자 하는 0번 블럭을 지우고, blob을 퓨징한 후, 에러 없이 퓨징이 되었는지를 검사하기 위해 검증한 후에 이상이 없으면 x-boot250이 정상적으로 로딩을 완료하게 된다. 이 상태에서 시리얼 통신 환경의 설정에 이상이 없다면 정상적으로 x-boot250이 동작하는 것을 볼 수 있을 것이다.

부트로더 제작

<그림 5>와 같이 일반적으로 부트로더라 하면 일반 x86 리눅스에서는 LILO를 많이 사용할 것이다. LILO란 Linux Loader로서 DOS나 윈도우 NT, 리눅스 등 다른 OS를 선택적으로 부팅할 수 있도록 하는 기능을 제공한다. LILO는 하드디스크의 MBR에서 동작하는 프로그램으로 OS가 실행할 수 있도록 점프하는 기능을 수행한다.

<그림 5> LILO와 X-boot250 과 비교



<그림 6> 임베디드 소프트웨어의 구조

그럼 우리가 사용하는 x-boot250이란 부트로더는 플래시 0 블럭에서 실행되고 여러 가지 다양한 기능들을 수행한다. 먼저 커널이나 램디스크(ramdisk) 등의 데이터를 호스트로부터 SDRAM 영역으로 다운로드할 수 있는 기능이 있고 SDRAM에 있는 데이터를 플래시 영역으로 라이팅할 수도 있다. 그리고 커널이 이미 올라가 있다면 부팅할 수 있는 기능도 같이 제공한다. 부트로더의 기능을 요약해보면 다음과 같다.

◆ **부트로더 기능**

1 하드웨어의 초기화 : 부트로더에서 제일 먼저 실행, CPU, 속도, 메모리, 인터럽트, UART 등을 초기화해 준다.

2 리눅스 부팅 : 부트로더 상에서 리눅스를 부팅하는 기능이 있다.

3 커널 또는 램디스크 다운로드 : 부트로더의 가장 중요한 기능인 커널이나 램디스크 이미지를 다운로드하는 기능이 있다. 호스트 상에서 컴파일된 이미지를 시리얼이나 tftp를 이용해 이더넷을 통해 SDRAM 상으로 다운로드가 가능하다.

4 다운로드한 커널 및 램디스크를 플래시에 라이트 : 다운로드한 커널과 램디스크 이미지는 SDRAM 상에 있기 때문에 전원이 꺼지면 다운로드한 이미지는 날라가 버린다. 그러므로 플래시 기능을 통하여 SDRAM 상의 커널과 램디스크를 지정된 플래시 주소 영역에 라이팅하는 기능이다.

5 tftp를 통한 SDRAM에 다운로드 : 시리얼을 통해 커널 등을 다운로드하기에는 속도가 너무 느리다. 그러나 tftp를 이용한 이더넷으로 다운로드할 수 있는 기능을 추가하여 고속으로 커널이나 램디스크를 다운로드할 수 있다.

임베디드 시스템 SW 개발과정

하드웨어가 제작되고 개발 환경이 구축되면, 다음으로 진행되는 과정은 <그림 6>과 같이 소프트웨어 개발 과정이다.

1 OS 포팅(임베디드 리눅스 포팅)
2 디바이스 드라이버 제작
3 TCP/IP 등 미들웨어 제작
4 GUI 등 애플리케이션 제작

OS 포팅(임베디드 리눅스 포팅)

타겟 보드에 부트로더가 올라가 있으면 이제는 커널을 이식하는 순서이다. 현재 전자밥솥에는 임베디드 리눅스를 포팅을 한다. 포팅에 필요한 내용은 다음과 같다.

우선 리눅스의 정식 버전을 어느 것으로 사용할 것인가 결정한다. 현재 포팅하려고 하는 리눅스 정식 버전은 2.4.18이다. 여기에 ARM 패치를 인가한다. 그리고 PXA255용 패치를 인가한다. 이 두 가지 패치는 일반적으로 공개되어 있다. 나머지 밥솥용 패치는 개발자가 개발을 하여 패치 처리를 해야 한다. 여기에는 밥솥용 타겟 보드의 특성에 맞는 리눅스 초기화 코드와 하드웨어에 있는 디바이스 드라이버가 추가된다. 이렇게 해서 개발된 리눅스 커널은 밥솥을 위한 전용 커널이 된다.

디바이스 드라이버 제작

다양한 하드웨어 장치를 구동시키기 위해서 운영체제는 장치 드라이버를 필요로 한다. 임베디드 리눅스는 디바이스 드라이버를 캐릭터, 블럭, 네트워크 디바이스 드라이버로 구분한다. <그림 3>의 임베디드 시스템 하드웨어 구조를 참조하여 디바이스 드라이버를 분리해 보면 다음과 같다.

◆ **캐릭터 디바이스 드라이버** : 히터 디바이스 드라이버 제작, 사운드 디바이스 드라이버 제작, LCD 디바이스 드라이버 제작, 터치 디바이스 드라이버 제작

◆ **블럭 디바이스 드라이버** : 여기서 블럭 디바이스 드라이버는 파일 시스템과 밀접한 관계를 가지고 있다. 밥솥용 임베디드 시스템에서는 파일 시스템을 플래시에 구현하였기 때문에 여기서 필요한 블럭 디바이스 드라이버는 플래시를 제어하는 디바이스 드라이버이다. 이를 흔히 MTD(Memoty Technology Device)라고 하며 임베디드 시스템에서는 매우 중요한 개념이다.

◆ **네트워크 디바이스 드라이버** : TCP/IP 제어를 받는 이더넷 디바이스 드라이버를 구현한다.

TCP/IP 등 미들웨어 제작

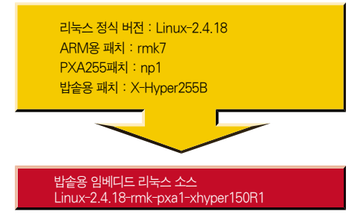
또한 앞으로의 임베디드 시스템에서는 통신이 기본으로 자리를 잡게 될 것이다. TCP/IP를 기반으로 하는 프로토콜을 사용하여 임베디드 시스템이 다른 기기와 정보를 공유할 수 있다면 다양한 네트워크 서비스를 사용자에게 제공할 수 있다.

GUI 등 애플리케이션 제작

애플리케이션은 크게 다음과 같이 3가지로 볼 수 있다.

1 기본 기능 : 밥솥의 본 기능
2 부가 기능 1 : TCP/IP를 이용한 통신 기능
3 부가 기능 2 : LCD를 이용한 GUI 기능

<그림 7> 밥솥용 임베디드 리눅스 소스 개발과정



이 시스템의 가장 기본 기능은 밥솥의 기능이다. 이를 하나의 프로세스에 할당하여 애플리케이션을 제작한다. 그리고 TCP/IP를 통해서 각종 네트워크 기능을 구현한다. 마지막으로 GUI 기능을 제공한다. 임베디드 시스템에서는 대부분 제한된 크기의 디스플레이를 사용하고 있으며 이러한 장치에서 텍스트, 그래픽 및 영상을 표시하기 위해서는 특별한 GUI 기술이 필요하다. 현재 널리 사용되고 있는 내장형 시스템 GUI 기술들(타이니 X, 피코구이, Qt, 마이크로윈도우)이 적용되어 텍스트, 그래픽, 영상 등을 사용자에게 제공한다.

무엇을 준비해야 할까?

앞서 살펴봤던 것과 같이 우리가 이제껏 알고 있던 그리고 흔히 접해왔던 일반적인 컴퓨터 기반의 하드웨어, 소프트웨어 개발과는 달리 임베디드 시스템의 경우엔 그 개발 환경 구축부터 개발에 이르기까지 여러 가지 상이한 점들이 존재하기 때문에 처음 이를 접하는 개발자들은 다소 혼란스럽고 적응에 실패하기도 한다.

그러나 임베디드 시스템 역시 CPU(프로세서)와 메모리로 되어 있는(어느 특정 목적을 위해 개발된) 조그만 컴퓨터라 생각하고 개발에 임한다면 그리 어렵지만은 않을 거라고 필자는 확신한다. 즉, 임베디드 시스템 개발시 다음의 기본 기술 요소들만 확실하게 닦아 놓는다면 나름대로 쉽게 적응할 수 있지 않을까 싶다.

소프트웨어와 하드웨어의 기본 동작 원리

처음 컴퓨터 전원 버튼을 누른 순간 컴퓨터 보드 안에 이미 내장되어 있던 BIOS 프로그램이 실행되어 부팅 순서를 결정하고 부팅 가능한 운영체제가 선택됨과 동시에 해당 운영체제가 부팅하기 시작한다. 부팅이 끝난 운영체제 상에서 원하는 애플리케이션을 실행시키기 위해 마우스로 더블 클릭하는 순간 운영체제는 해당 애플리케이션을 메모리에 상주시킨 뒤 이를 실행시키게 되며 이때 만약 USB 같은 외부 디바이스들을 제어할 필요가 있을 경우 이를 제어하게 된다.

임베디드 시스템 역시 이와 마찬가지다. 전원이 들어감과 동시에 이미 내장되어 있던 부트로더가 실행되고, 이후 해당 임베디드 리눅스나 윈도우 CE 같은 운영체제가 실행되는, 이와 같은 소프트웨어와 하드웨어의 동작 원리에 대해 개발자는 이들이 실행되는 구조를 그려가며 반드시 이해해야만 할 것이다.

하드웨어 개발

필자의 생각에 이는 모든 개발자들에게 해당되는 사항은 아니라고 본다. 자신의 전공 분야에 따라 전문적으로 개발하는 분야가 다른 만큼 소프트웨어 개발자들은 하드웨어 개발 방법에 관해 사실 몰라도 된다. 그러나 하드웨어를 개발하는 방법을 몰라도 된다는 얘기가 하드웨어를 몰라도 된다는 얘기는 아니다.

아무리 컴퓨터상에서 소프트웨어를 전문적으로 개발하는 사람이라 할지라도 해당 컴퓨터의 하드웨어 사양(스펙)을 모른다면 어찌 원하는 성능을 얻을 수 있겠는가?

이런 관점에서 보면 어떤 개발자라도 하드웨어에 대해 기본적으로 알아야 하겠지만 임베디드 개발자의 경우엔 소프트웨어 전문 개발자들보다 조금 더 깊이 알아둘 필요가 있다. 그래야 개발자가 원하는 대로 하드웨어를 제어할 수 있을 테니까.

프로그래밍 언어

이는 사실 언급할 필요도 없는 사항일지 모른다. 전쟁에 참여하는 군인이 총을 놓고 간다면 어찌 되겠는가? 그러나 필자가 겪어 본 바에 따르면 총을 놓고 가는 군인들이 더러 눈에 보여 굳이 언급을 하고자 한다. 요즘에 가끔 TV를 보면 정말 화려한 가수나 연예인들이 많이 보이는 데 이는 눈에 보이는 것만을 쫓고 있는 요즘 세태가 반영되어 그런 듯 싶어 썩 반갑지가 않다. 이는 개발에서도 그대로 적용되는데, 요즘 보면 결과가 바로 눈에 보이는 언어들에 사람들이 몰리며 이에 대한 책들도 많이 나오고 있다. 그러나 이들 중 C 언어부터 차근차근 밟아 나가는 사람들은 그리 흔하지 않은 건 왜일까?

어떤 프로그래밍 언어이든 하나는 확실하게 알고 있어야 한다. 그래야 확장이 가능하기 때문이다. 그러나 이런 언어들은 공부에 대한 결과가 바로 눈에 띄지 않기 때문에 중도에 포기하는 개발자들이 많다. 이는 영어를 공부하는 것으로 생각해보면 알 수 있다. 영어에는 왕도가 없다지 않은가. 그저 묵묵히 최선을 다해 습득하는 것 그 길뿐이다. 프로그래밍 언어 역시 마찬가지라고 생각된다. 조급해 하지 말고 묵묵히 습득하다 보면 어느새 늘어 있는 자신의 실력에 놀랄 때가 반드시 있을 것이다.

임베디드에 거는 기대감과 희망

이 글에서는 임베디드 시스템을 32비트 CPU 기반 임베디드 하드웨어 시스템, OS, 미들웨어, 애플리케이션의 계층 구조를 가지는 임베디드 시스템에 대하여 살펴보고 이들을 개발하기 위해서는 어떤 식으로 접근해야 할지 살펴보았다.

'무어의 법칙'에서 말해주듯 하드웨어 사양들은 하루가 다르게 변화하고 있고, 이보다 더 빠른 속도로 소프트웨어 환경은 급변하고 있다. 그리고 이러한 환경에서 다양한 분야가 생겨나고 또 사라져 가고 있다. 이런 광속의 시대를 살면서 과연 우리의 개발자들은 어디에 서야 할 것인가에 대해 오랜 기간 개발자의 길을 걸어온 사람들은 누구나 고민하고 있으리라 생각된다. 과연 이 땅의 개발자들은 어디로 가야 하는가?

필자는 임베디드라는 분야에 우리가 서야 할 길이 있다고 확신하고 있으며, 이를 다양한 개발 경험과 강의 현장에서 몸소 느끼고 있다. 그러나 아직 인식이 많이 되어 있지 않기에 어찌하면 이 분야에 쉽게 접근할 수 있을지를 생각해봤다.

앞서도 언급하였지만 임베디드는 우리가 이제껏 접해왔던 컴퓨터 환경과는 그 운용 방법이나 개발 방법 등 모든 면에서 다르다 할 수 있다. 하지만 그 속내를 들여다보면 우리가 이제까지 등한시했던 기본적인 내용들을 다시금 돌아보고 숙지한다면 결국 개발이라는 방법 자체가 아닌 개발 과정의 흐름은 동일하다는 사실을 이해하리라 생각한다.

CPU, 프로그래밍 언어, 컴파일러와 같은 기본 개발 분야에 있어서 이미 선진 유럽과 미국을 우리가 뛰어 넘기란 사실상 불가능해 보인다. 그러나 이제 막 꽃을 피우기 시작한 임베디드라면 기대해도 좋을 희망이 있지 않을까? 뛰어난 임베디드 개발자들이 무수히 쏟아져 나오길 마음 속으로 간절히 기도하며 글을 맺는다.

정리 | 조규형 | jokyu@korea.cnetcom.com

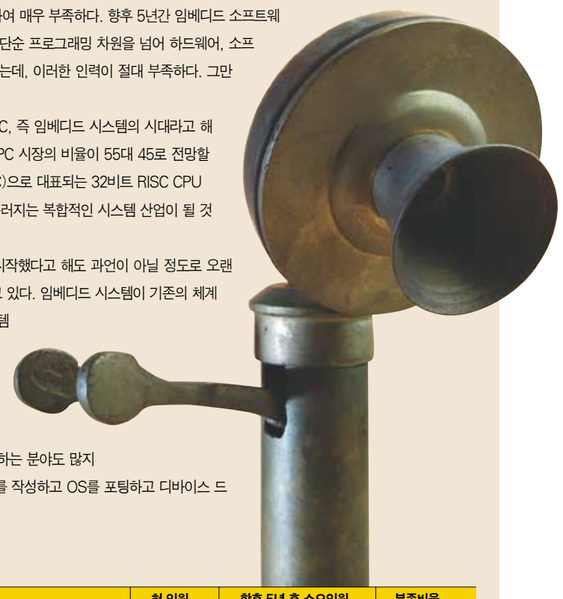
참 + 고 + 자 + 료

1. Embedded World 2003, 1 vol. 1
2. 임베디드 소프 트웨어 기술동향 및 산업 발전 전망, 정보통신연구진흥 제 4권 제 3 호, 임채덕 등, 정보통신연구진흥원,http://iita6.iita.re.kr:8888/korean/journal/13/focus_01.htm, 2002. 9.
3. 2002 Embedded Software Tools Worldwide Forecast, Gartner Dataquest Market Statistics 110850, 2002. 10.
4. Worldwide Embedded Software Tools Outlook, 2002, Gartner Dataquest Alert, 2002. 10.
5. Worldwide Embedded Operating Environments Forecast, 2003-2007, IDC #29308, 2003. 5.
6. The Embedded Software Strategic Market Intelligence Program 2002/2003, Volume II, VDC, 2003. 3.
7. Embedded Software Developers Maintain the Status Quo, Gartner, 2003. 5.
8. 2001 Worldwide Embedded Software Tools Market Share, Gartner Dataquest Market Statistics, SWTA-WW-MS-0113, 2002. 7.
9. 이것이 성장엔진이다 임베디드 S/W, 전자신문 2003. 3. 11.

[임베디드 시스템 산업 인력 현황]

매년 대학에서 배출되는 임베디드 소프트웨어 개발인력이 업체에서 필요로 하는 인력에 비하여 매우 부족하다. 향후 5년간 임베디드 소프트웨어 개발인력의 수급 부족이 약 1만 2000명으로 예상되고 있다. 임베디드 시스템 개발자는 단순 프로그래밍 차원을 넘어 하드웨어, 소프트웨어 지식을 겸비하고 프로젝트 관리 능력을 갖춘 시스템 아키텍트 수준의 인력을 요구하는데, 이러한 인력이 절대 부족하다. 그만큼 개발자로서 매력 있는 분야라 할 수 있다.

1990년대에 마이크로소프트의 윈도우 기반의 PC 시대를 지나 2000년대에는 포스트 PC, 즉 임베디드 시스템의 시대라고 해도 과언이 아니다. 앞으로 멀지 않은 장래에 임베디드 시스템인 포스트 PC에 대한 기존의 PC 시장의 비율이 55대 45로 전망할 만큼 임베디드 시스템 산업이 뜨겁게 성장하고 있다. 임베디드 시스템은 사회간접자본(SOC)으로 대표되는 32비트 RISC CPU가 내장된 반도체 사업, 임베디드 OS, 그리고 다양한 애플리케이션 소프트웨어 산업이 어우러지는 복합적인 시스템 산업이 될 것이라 생각한다.



임베디드 시스템은 컴퓨터가 등장한 이후 공장 자동화 분야에 CPU가 사용되면서부터 시작했다고 해도 과언이 아닐 정도로 오랜 역사를 가지고 있다. 그리고 2000년을 기점으로 임베디드 시스템의 트렌드 변화가 일어나고 있다. 임베디드 시스템이 기존의 체계에 변화를 촉진하고 있으며 전체 산업영역으로 급속히 확산되고 있다. 이러한 임베디드 시스템의 범위 및 분야가 광범위한 체계를 갖추고 있기 때문에 이에 종사하는 개발자가 되기 위해서는 많은 시간이 필요하다. 그래서 다른 영역에 있는 개발자와 입문자가 쉽게 접근하는 데 어려움이 있다는 것이 문제가 된다. 하지만 이를 효율적이고 체계적으로 학습하여 빠르게 개발자가 되기 위한 방법론을 제시하는 것이 이 글의 목적이라고 할 수 있다.

모든 분야의 일이 마찬가지지만, 임베디드 시스템 관련 개발 분야만큼 실무를 바탕으로 하는 분야도 많지 않을 것이다. 임베디드 분야는 보드를 설계하고 이 보드에 개발 환경을 구축하고 부트 코드를 작성하고 OS를 포팅하고 디바이스 드라이버를 작성하고 애플리케이션을 작성하는 등의 일련의 작업들을 하는 것을 의미한다.

<표 1> 임베디드 시스템 예상인력 모습

구분	표준 스펙 정의	현 인원	향후 5년 후 소요인원	부족비율
초급	디바이스 드라이버를 작성할 수 있을 정도의 기본 skill을 습득한 3년 이내의 실무 경험이 있는 인력	13,200	25,000	89%
중급	하드웨어 기본 지식이 있고 5년 정도의 시스템 개발 실무 경험을 갖춘 학사 또는 석사급 인력	3,500	11,500	228%
고급	하드웨어, 소프트웨어 지식을 겸비하고 10년 이상의 프로젝트 관리 경험이 있는 시스템 아키텍트 수준의 인력	300	2,500	733%