내장형 시스템의 효율적인 개발 환경 설계 및 구현

김 성 훈, 유 진 호, 신 형 철, 하 정 현, 한 동 원
한국전자통신연구원 컴소연 인터넷정보가전연구부
전화 : 042-860-6771

A Design and Implementation of efficient SDKs for the embedded system

Sung-Hun Kim, Jin-Ho Yoo, Hyung-Cheol Shin, Jung-Hyun Ha, Dong-Won Han
Internet Appliance Technology Dept. , CSTL, ETRI
E-mail : sunghun@etri.re.kr

Abstract

An embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function.

In this paper, we design and implement efficient SDKs for the embedded system. When we develop the embedded system, we use the cross development method because of limited resource. Simulator, stub for GDB, and monitor are implemented in order. Simulator consists of 4 threads; CPU, display, I/O, timer thread. Stub is developed to connect GDB. Monitor is programed to improve stub and can debug the application program without the debugger in the host.

I. 서론

최근 인터넷의 폭발적 성장과 함께 나타난 두드러진 현상은 컴퓨터뿐만 아니라 다양한 정보기기 들이 인터넷에 직접 접속되기 시작했다는 것이다. 전화기, TV, PDA 등은 물론이고 냉장고, 전자레인지 등의 가전제품 들의 인터넷 접속까지 논의되고 있는 실정이다. 이러한 기기들의 특징은 마이크로콘트롤러를 내장하여 여러 개의 실시간 응용 프로그램을 수행하는 내장형 시스템(embedded system)이라는 것이다.

주변 기술이 발전하고 응용 프로그램이 복잡해지면서 내장형 시스템에서도 과거와는 다른 개발 방법들이 나타나고 있다. 첫 번째로, 과거의 내장형 시스템 개발은 운영체계(OS)의 지원 없이 주로 어셈블리언어를 사용하여 장치 제어에서부터 사용자 인터페이스에 이르기까지 전체를 구현하였으나 인터넷 접속 기능 등 응용에서 요구하는 기능이 더욱 복잡하고 다양해지면서 운영체계의 지원 없이는 개발이 불가능하게 되었다[1]. 두 번째로, 고급언어를 사용하여 응용 프로그램을 작성하는 것이 일반적으로 되었는데, 내장형 시스템은 메모리나 디스크 등 자원이 한정되어 있어서 호스트 컴퓨터에서 주로 개발을 하여 타겟 시스템의 실행코드를 생성해내는 교차개발(cross development) 방법이 많이 사용되고 있다[2][3][4]. 고급언어를 사용한 실시간 응용프로그램의 교차 개발을 위해서는 호스트 컴퓨터 상에서 실행되는 다양한 개발도구의 지원이 필수적이다. 여기에는 디버거와 대화형 쉘(interactive shell) 등이 포함된다. 세 번째로, 과거에는 주로 가격이나 성능에 기준을 두어 시스템을 개발하였지만, 현재에는 time-to-market이 새로운 개발 기준이 되고 있다[5]. 개발 기간을 줄이면서 복잡한 응용 프로그램을 쉽게 작성할 수 있는 확실한 방법은 효율적인 개발 환경(SDK) 설계 및 구현에 달려 있다.

본 논문은 현재 Motorola와 ETRI가 함께 수행하고 있는, M341 마이크로프로세서(microprocessor)와 USB, Bluetooth, LCDC등의 주변장치들이 원칩화되어 있는

아직 미출시된 DB-X 마이크로콘트롤러를 이용한 PDA 개발중에서 시간과 비용을 줄일 수 있는 효율적인 개발 환경 설계 및 구현에 대하여 기술한다. 우선 내장형 시스템을 위한 개발 환경의 전체 구현 절차에 대해 설명하고 다음으로 각각의 개발 환경 구조에 대해 기술한 후 결론을 맺는다.

II. 개발 환경의 전체 구현 절차

PDA와 같이 기술이 급변하는 내장형 시스템에서는 기존에 이미 출시된 칩과 개발 환경을 이용하여 시스템을 개발하면 시간과 비용 때문에 경쟁력이 없게 된다. 그러므로, 칩 제조회사와 연계를 통해 칩을 기획하는 단계부터 칩을 테스트하고 응용 프로그램을 개발하는 일이 동시에 이루어져야 한다. 즉, 칩의 출시 시점과 응용 프로그램의 작성 시점을 잘 파악하여 저 비용으로 가장 빠르게 개발 환경을 구현해야 한다. 그림 1은 개발 환경의 전체 구현 절차를 보여 주고 있다.

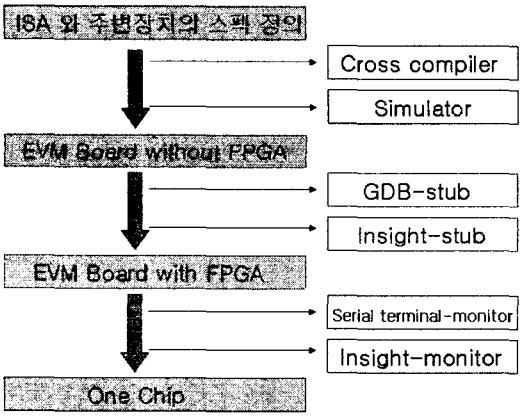

그림 1. 개발 환경의 전체 구현 절차

완전한 형태의 칩이 출시되기 전에, 마이크로프로세서의 ISA(Instruction Set Architecture)와 주변 장치들의 스펙이 정의된다. ISA 등이 정의되면 고급 언어를 이용하여 호스트 컴퓨터에서 응용 프로그램을 작성하기 위해 MD(Machine Description)을 이용한 교차 컴파일러(cross compiler)를 만든다. 그 후에, 기본적인 칩 테스트를 위한 M341 코어, 입출력, 메모리만을 가지는 EVM 보드와 여기에 주변 장치를 FPGA로 구현한 성능 향상된 EVM 보드를 차례로 만든다. 그런데, FPGA가 탑재된 EVM 보드를 만들기까지는 많은 시간이 필요하기 때문에 탑재될 운영체계와 응용 프로그램의 테스트를 위해 ISA와 주변 장치의 스펙을 이용하여 시뮬레이터를 먼저 개발한다. 시뮬레이터를 이용하면 OS kernel, shell등의 주로 메모리를 이용하는 프로그램의 테스트와 UART, LCD등 에뮬레이션 하기가 쉬운 주변 장치를 이용하는 응용 프로그램을 테스트하게 된다.

시뮬레이터는 모든 주변 장치를 에뮬레이션하는데 한계가 있으므로 EVM 보드가 출시되면 EVM 보드를 이용한 교차 디버깅을 주로 하게 된다. EVM 보드를 이용하여 소스 레벨 교차 디버깅(source-level cross debugging)을 하기 위해서는 소스 코드가 공개되어 있고 무료인 GDB를 이용하는 것이 효율적이다. GDB의 소스 코드를 수정하지 않고 가장 간단하게 GDB를 이용하는 방법은 GDB의 시리얼 프로토콜(serial protocol)에 맞도록 호스트의 GDB와 통신할 수 있는 M341 코어용 스텁(stub)을 타겟 시스템쪽에 구현하는 것이다. 스텁은 EVM 보드에서 뿐만 아니라 시뮬레이터에서의 교차 디버깅을 위해서도 사용되며 GDB에서 기본적으로 제공하는 M68k 코어용 스텁을 참조하면 쉽게 구현할 수 있다. 그 후에, 텍스트 기반인 GDB에 GUI를 붙인 Insight를 이용하면 개발자의 편리성이 증가하게 된다. Insight 또한 소스 코드가 공개된 무료 소프트웨어이다.

GDB의 시리얼 프로토콜을 이용한 디버깅은 그 기능이 제한되어 있고 반드시 호스트에 GDB가 있어야 하므로 높은 수준의 디버깅을 하기에는 한계가 있다. 이를 극복하기 위해서 GDB없이 일반적인 시리얼 터미널만으로도 교차 디버깅이 가능하도록 스텁에 더 많은 기능을 붙인 모니터(monitor)프로그램을 작성한다. 모니터 프로그램은 호스트 컴퓨터의 GDB가 하는 일들을 EVM 보드 내에서 모든 것을 해결하도록 작성된다. 예를 들면, GDB와 스텁을 이용한 디버깅에서는 정지점 관리를 GDB 내에서 했지만 모니터 프로그램에서는 모니터 프로그램 자체 내에서 구조체를 만들어 관리한다. 모니터 프로그램이 정상적으로 잘 동작하면 소스 레벨 디버깅을 위해 Insight의 소스 코드를 수정하여 모니터 프로그램과 통신할 수 있도록 하면 EVM 보드에서의 개발 환경은 모두 완성된다. 즉, 호스트의 디버거 없이 시리얼 터미널만으로도 디버깅을 수행할 수 있는 모니터 프로그램과 GUI기능을 가진 호스트의 소스레벨 디버거를 이용하여 디버깅을 수행한다.

III. 시뮬레이터의 구조

시뮬레이터는 ISA만 있어도 구현할 수 있으며, 필요한 주변장치를 에뮬레이션하여 그 기능을 추가하게 된

다. 그런데, 호스트의 윈도우 NT 기반으로 시뮬레이터를 작성하기 때문에 모든 주변 장치를 에뮬레이션하는데 한계가 있다. 작성된 시뮬레이터는 크게 CPU, UART, LCD, 타이머 시뮬레이션으로 구성되며 각각은 Win32의 쓰레드(Thread)를 이용하여 구현된다. 쓰레드 간의 동기는 Win32의 이벤트를 이용하며 각 쓰레드의 구현 방법은 다음과 같다.

(1) CPU : CPU는 ALU(Arithmetic and Logic Unit), CU(Control Unit), 레지스터(register)로 구성되어 있으며, 메모리에서 명령어를 가지고 온 후에 명령어에 해당하는 ISA을 수행하는 기능을 한다. ISA에서는 메모리나 레지스터의 값을 변화시키는 일을 하게 된다. 그러므로, 시뮬레이터에서도 PC(program counter)가 지정하는 주소에서 명령어를 가지고 와서 해석하여 해당하는 ISA를 수행한 후, 다시 PC 값을 증가하여 다음 명령어를 가지고 오는 일을 반복하면 된다. 이 때 주의할 점은 타겟 시스템에서 사용하는 메모리 주소와 호스트에서 사용하는 메모리 주소가 다르므로 가상 메모리 맵핑(virtual memory mapping)기법을 통해 메모리 주소를 전환하여 사용해야 한다. 또한, 인터럽트 처리는 인터럽트가 발생했다는 것을 플래그를 이용하여 나타낸 후 다음 PC값으로 업데이트하기 전에 항상 플래그를 체크하게 하여 플래그에 따라서 PC값을 각각의 인터럽트 처리 루틴의 주소로 변경하도록 구현한다. 그리고, 레지스터는 메모리 변수로 구현한다.
(2) UART : 표준 입출력에 해당하는 부분으로 호스트의 COM 포트를 이용한다. 출력은 COM 포트로 바로 보내지만 입력은 내부에서 입력 데이터를 처리하는데 시간이 걸리는 경우등의 데이터 손실을 막기 위해 버퍼를 둔다. 이 때 버퍼를 읽고 쓸때 상호 배제 접근과 사용할 수 있는 버퍼수를 제한하기 위해 Win32의 뮤텍스(Mutex), 세마포어(semaphore)를 이용한다. 또한, 1대의 호스트로 디버깅을 하기 위해서 COM1과 COM2를 물리적으로 서로 연결한 후 COM2에는 윈도우의 하이퍼터미널을 연결한다. 그런데, 전송할 데이터 양이 작을 때는 COM 포트를 이용하면 되지만 데이터 양이 커지면 속도가 느리기 때문에 COM 포트 대신 TCP/IP의 소켓을 이용하여 루프백(loopback)시킨다.
(3) LCD : 일반적으로 LCD는 레지스터 값이 초기화되면 디스플레이될 데이터의 처음 주소가 저장되어 있는 레지스터를 이용하여 데이터의 시작 위치를 찾은 후에 그 데이터를 화면에 디스플레이 해준다. 시뮬레이터에서도 레지스터 값이 초기화되었다면 데이터의 처음 주소가 저장된 레지스터의 값을 이용하여 데이터를 가져온 후 비트맵 형태로 화면에 디스플레이한다.
(4) Timer : 레지스터 값에 따라 Win32의 타이머를 이용하여 구현한다.

그림 2는 쓰레드, 가상 메모리 맵핑, 비트맵, 타이머, 소켓을 이용하여 윈도우 NT에서 구현한 시뮬레이터의 전체 구조를 보여 주고 있다.

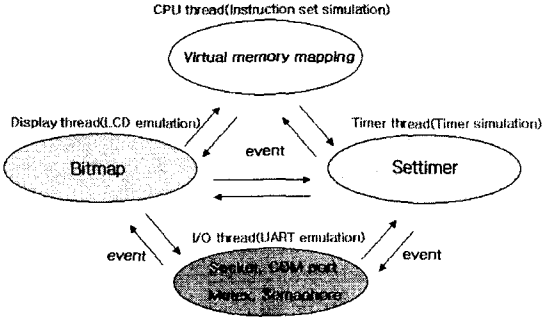

그림 2. 시뮬레이터의 전체 구조

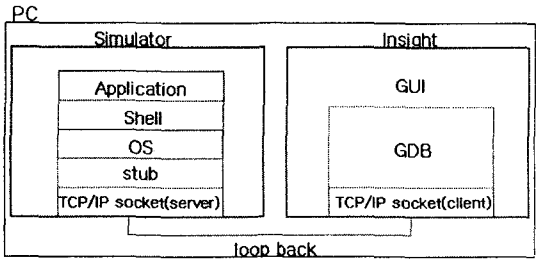

그림 3. 시뮬레이터를 이용한 응용 프로그램 디버깅

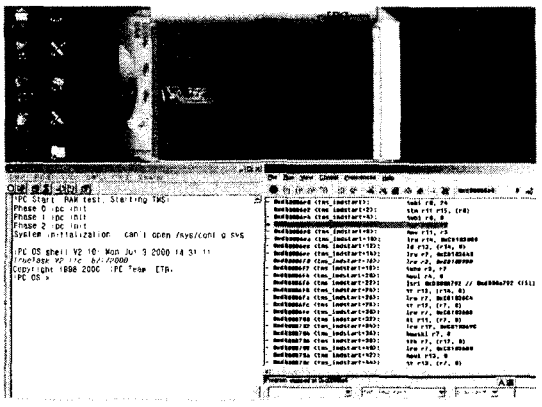

그림 4. 윈도우 NT에서의 실제 디버깅 모습

구현된 시뮬레이터를 이용하면 여러 가지 디버깅을 할 수 있다. 즉, 운영 체계 내부에 대한 디버깅도 할 수 있으며 운영체계 내에서 수행되는 응용 프로그램에 대한 디버깅도 할 수 있다. 그림 3은 시뮬레이터를 이용하여 응용 프로그램을 디버깅하는 방법을 나타내고

있으며 그림 4는 실제 윈도우 NT에서 시뮬레이터, Insight, 하이퍼터미널을 이용한 응용 프로그램 디버깅 모습을 나타내고 있다.

Ⅳ. GDB 스텁과 모니터 구조

호스트의 GDB와 연동할 수 있도록 타겟 시스템쪽에서 구현되는 스텁의 구현 순서는 다음과 같다.

(0) 전원을 켰을 때 코어가 명령어를 처음 가지고 오는 주소에 초기화된 인터럽트 처리 테이블을 둔다
(1) 모든 인터럽트를 막는다.
(2) 스택과 VBR(vector base register)를 초기화한다.
(3) ROM 영역의 .data부분을 RAM 영역으로 복사하고 .bss 부분을 초기화한다.
(4) 인터럽트를 가능하게 한다.
(4) 시리얼 통신 장치를 초기화시킨다.
(5) Breakpoint exception 명령어나 그 명령어가 없는 경우는 Illegal instruction exception을 발생시킬 수 있는 명령어를 이용하여 예외 상황(exception)을 발생시킨다.
(6) (5)로 인해서 발생된 예외 처리 루틴에서 레지스터 값들을 메모리에 저장시킨 후, GDB와 통신할 수 있도록 셉업 신호를 GDB의 시리얼 프로토콜을 이용하여 호스트의 GDB로 보낸다.
(7) GDB와 연결이 되고 GDB에서 미리 정해진 명령어을 보내면 스텁에서 대응하는 처리 루틴을 수행한다.

내장형 시스템을 위한 시스템 프로그램을 작성할 때는 기존의 표준 라이브러리조차도 사용할 수 없다는 가정에서 출발해야 한다. 그러므로, 앞의 (1)번부터 (4)번까지는 startup에서 해 주어야 하는 부분이지만 내장형 시스템에서는 항상 이 부분을 작성해야 한다.

최근의 내장형 소프트웨어에서 요구하는 매우 복잡한 기능을 지원하기 위해서는 강력한 개발도구의 지원이 필수적이다. 또한 호스트시스템에서 수행되는 도구들의 기능을 타겟시스템과 효율적으로 연결시키기 위해서는 플랫폼 독립적인 프로토콜을 갖고 수행되는 모니터 프로그램의 개발이 요구된다. 모니터는 호스트의 디버거 없이 시리얼 터미널만으로도 디버깅할 수 있도록 스텁에 대화형 쉘, 정지점 관리등의 기능이 추가된 프로그램이다. 그림 5는 모니터 프로그램에 포함된 기능들을 나타내고 있다.

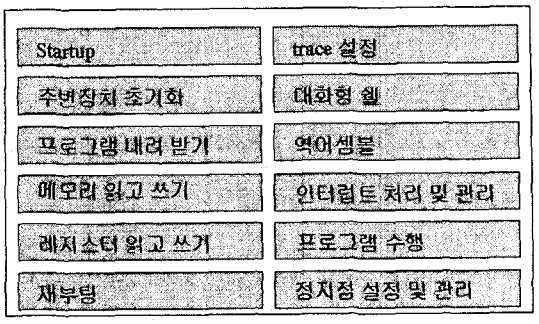

그림 5 모니터에 포함된 기능

Ⅴ. 결론

본 논문에서는 인터넷에 접속되는 정보기기등의 내장형 시스템에서 사용되는 개발 환경의 효율적인 구현 절차와 각각의 개발 환경 구조와 기능을 제시하였다.

기술이 급속히 발전하면서 누가 저비용으로 가장 먼저 시스템을 개발하는지가 가장 중요한 개발 기준이 되고 있다. 본 논문에서 제시된 개발 환경 구현 절차 및 구조들은 앞으로 개발될 내장형 시스템의 개발 환경 구현의 하나의 모델이 될 수 있을 것이다.

참고문헌(또는 Reference)

[1] 유진호, "Implementation of A Handheld Embedded Kernel", 대한전자공학회 추계학술대회,1999
[2] Jack G. Ganssle, "Debuggers for Modern Embedded Systems," Embedded Systems Programming, Nov. 1998
[3] 이은향 외 , "교환 소프트웨어의 교차 디버깅을 위한 디버그 서버 구현 ," JCCI, 광주, 1996
[4] Jonathan B. Rosenberg, "How Debuggers Work," John Wiley & Sons, 1996
[5] Bill Giovino, "Overlaps Between Microcontrollers and DSPs", Embedded Systems Programming, Jan. 2000