SoC 기반의 프로토타이핑 미들웨어 개발에 관한 연구

김문석° 권용진
한국항공대학교 정보통신공학과
{tostone°, yjkwon}@tikwon.hangkong.ac.kr

A Study on Development of Prototyping Middleware Based on SoC

Moonseok Kim°, Yongjin Kwon
Dept. of Info. & Telecom. Eng., Hankuk Aviation University

요 약

임베디드 시스템이 점점 다기능화 · 고기능화 됨에 따라 구현과정도 복잡해지고 있으며, 빠른 제품 주기 등의 영업적인 이유로 개발 기간을 단축하려는 요구는 증대되고 있다. 특히 SoC 분야는 하드웨어와 소프트웨어를 동시에 개발하기 때문에, 두 부분이 동시에 정상 동작하는지의 검증이 어렵다.

이러한 문제들을 해결하기 위해, 다양한 시스템에 적용 가능한 플랫폼을 제안하고, 이 플랫폼을 기본 골격으로 사용하는 플랫폼 기반 개발 방법이 등장하게 되었다. 이 방법은 개발 과정에서 플랫폼 모듈을 재사용하기 때문에 안전성을 확보할 수 있고, 많은 자동화를 이룰 수 있어 개발 시간을 단축할 수 있다.

본 논문에서는 이 플랫폼 기반 개발 방법을 프로토타이핑 과정에 적용한 프로토타이핑 미들웨어를 제안한다. 이 미들웨어를 통해, 구현된 하드웨어의 정상 동적 여부를 빠르게 확인할 수 있고, 프로토타이핑 과정을 마치고 최종 소프트웨어를 작성할 때에도 참고가 될 수 있다. 또한 설정 부분은 미들웨어 사용자의 편의를 위해 XML을 이용해 사용자가 직관적으로 알 수 있게 하였고, 사용자 작성 모듈은 미들웨어에서 정해진 인터페이스를 통해 미들웨어의 재 컴파일 없이, 사용자 모듈만 컴파일하면 동작하는 특징을 갖는다. 그리고 이 미들웨어를 도어락 제어 시스템에 적용하여 편의성을 평가한다.

1. 서 론

최근 임베디드 시스템이 점점 다기능화 · 고기능화 됨에 따라 구현과정도 복잡해지고 있으며, 빠른 제품 주기 등의 영업적인 이유로 개발 기간을 단축하려는 요구는 증대되고 있다. 특히 SoC 분야는 하드웨어와 소프트웨어를 동시에 개발하기 때문에, 두 부분이 동시에 정상 동작하는지의 검증이 어렵다[1].

이러한 문제들을 해결하기 위해, 다양한 시스템에 적용 가능한 플랫폼을 제안하고, 이 플랫폼을 기본 골격으로 사용하는 플랫폼 기반 개발 방법이 등장하게 되었다. 이 방법은 개발 과정에서 플랫폼 모듈을 재사용하기 때문에 안전성을 확보할 수 있고, 많은 자동화를 이룰 수 있어 개발 시간을 단축할 수 있다[2][3].

기존 개발 방법 중 어플리케이션을 설정 관리자를 이용한 방법이 제안되었지만, 설정과 사용에 어려움이 있다[4]. 이 문제를 XML을 통해 쉽게 사용할 수 있도록 제안한다.

본 논문에서는 플랫폼 기반 개발 방법을 프로토타이핑 과정에 적용한 프로토타이핑 미들웨어를 제안한다. 이를 위해 2장에서는 제안한 미들웨어에 사용된 기술에 대해서 간략하게 설명한다. 3장에서는 제안한 미들웨어를 설명하고, 4장에서는 이 미들웨어를 도어락 시스템에 적용해서 평가한다. 마지막으로 5장에서 결론을 맺는다.

2. 기본 기술

제안한 프로토타이핑 미들웨어는 임베디드 리눅스를 기반으로 한다. 사용자의 미들웨어 설정은 XML 문서를 이용하였고, 설정만으로 사용자 모듈을 구동시키기 위해 사용자 모듈을 동적 라이브러리로 만들어 사용하는 방법을 택했다. 이번 섹션에서는 미들웨어에 사용된 기본 기술을 설명한다.

2.1 플랫폼 기반 개발 방법론

플랫폼 기반 방법이 적용되기 이전에는 개발 그룹 안이나 그룹 사이에서 같은 기능을 하는 디자인들이 반복적으로 디자인되고, 거의 재사용되지 못했다. 또한 작성된 기능의 디자인 오류들은 디자인을 여러 번 반복하게 만들었다. 그러면서도 처음 시장을 차지하기 위한 경쟁은 심해져서 제품을 더 빨리 출시해야 했다. 이러한 문제를 해결하기 위해 플랫폼 기반 방법이 제안 되었다.

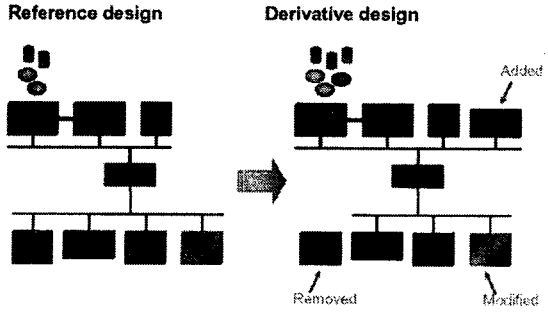

그림 1. 플랫폼 기반 개발

이 방법을 이용하면 디자인 주기를 상당히 줄일 수 있고, 이미 검증된 모듈을 다시 사용하므로 복잡한 디자인에 대한 실제 병목을 찾는데도 도움이 된다[5]. 또한 기본 플랫폼부터 목적 플랫폼을 얻는 것이 빠르게 가능하기 때문에, 빠른 프로토타이핑과 빠른 H/W의 작성 및 S/W 개발이 가능하다.

그러나 미리 정해진 플랫폼 모듈을 이용하기 때문에 혁신적이고 새로운 제품을 개발할 수 없는 단점이 있다.

2.2 임베디드 리눅스

임베디드 리눅스는 임베디드 시스템을 위한 리눅스이다. 임베디드 시스템은 범용 시스템과는 달리 저 성능의 프로세서, 소 용량의 메모리를 가지게 되는데, 이러한 시스템에 최적화

된 커널과, 시스템에 연결되어 있는 디바이스 드라이버, 그리고 이것을 사용하는 응용 프로그램 및 개발환경 전체를 임베디드 리눅스라고 한다.

최근 들어서 임베디드 리눅스를 더 많이 이용하는 이유를 몇 가지 살펴보면, 임베디드 시스템이 다양한 응용 프로그램을 수용 하면서 메모리 보호 문제가 대두되고 있는데, 리눅스는 이미 이 부분을 해결하고 있기 때문에 한 응용 프로그램의 문제가 전체 시스템의 장애로 이어지는 사태를 막을 수 있다. 또한 좀 더 좋은 성능을 가진 하드웨어가 임베디드 시스템에 사용되고, 예전과는 비교할 수 없는 복잡한 소프트웨어 기능이 요구되고 있기 때문이다.

2.3 XML

XML(eXtensible Markup Language)은 1996년 W3C에서 제안한 것으로 웹상에서 구조화된 문서를 전송 가능하도록 설계된 표준화된 텍스트 형식이다. SGML(Standard Generalized Markup Language)에서 사용되는 많은 기술들을 받아들여 간소화하고 보완하면서 발전해 나가고 있기 때문에 구조적으로 SGML 문서 형식을 따르고 있다.

XML은 웹드 와이드 맵, 인트라넷 등 에서 데이터와 포맷 두 가지 모두를 공유하려고 할 때 유용한 방법이다. 특히 어떤 플랫폼에서나 읽을 수 있는 포맷을 제공하고, CSV (Comma-Separated Value) 형식이나 fixed width 형식과는 달리 직관적으로 데이터의 의미나 구조를 파악할 수 있는 장점이 있다. 데이터 교환과 공유의 수단으로 매우 편리하기 때문에 RSS, 트랙백 표준, XML-RPC와 같은 형태로 다양하게 활용되고 있다.

2.4 동적 라이브러리

라이브러리란 특정한 코드를 포함하고 있는 컴파일 된 파일이다. 라이브러리를 만들어서 자주 사용되는 특정한 기능을 main함수에서 분리시켜 놓음으로써, 프로그램을 유지, 디버깅을 쉽게 하고 컴파일 시간을 좀 더 빠르게 할 수 있게 된다.

라이브러리는 적재 시간에 따라 크게 3종류로 분류할 수 있다. 첫째, 정적 라이브러리이다. 이는 object file의 단순한 모음으로 보통 .a의 확장자를 갖는다. 컴파일 시 적재되기 때문에 유연성이 떨어진다. 둘째, 공유 라이브러리이다. 프로그램이 시작될 때 적재되는 라이브러리로 만약 하나의 프로그램이 실행되어서 공유 라이브러리를 사용했다면, 그 뒤에 이를 사용하는 모든 프로그램은 자동적으로 이미 적재되어 있는 라이브러리를 사용하게 된다. 마지막은 동적 라이브러리이다. 프로그램 동작중 이 라이브러리가 필요할 때에 적재된다. 플러그인 모듈 등을 구현할 때 적합하고, 설정 파일 등에 읽어 들일 라이브러리를 등록시키고 원하는 라이브러리를 실행 시키게 하는 등 매우 유연하게 작동하는 프로그램을 만들고자 할 때 유용하다.

3. 제안한 미들웨어의 구현

3.1 타겟 하드웨어 아키텍처

목표로 하는 하드웨어는 임베디드용 일반 목적의 프로세서와 메모리, 그리고 하드웨어 로직을 넣을 수 있는 FPGA 등이 SoC된 하드웨어 구조를 택한다. 프로세서를 통해 유연성을 높이고, FPGA를 통해 성능을 높인 이러한 구조를 갖는 하드웨어는 Xilinx의 Vertex II Pro, Altera의 Excalibur, Triscend의 E5 and A7, Quicklogic의 QuickMIPS 등 다양하다.

본 연구에서는 이 중에서 Altera의 Excalibur 하드웨어를 이용한다[6]. 이 하드웨어에는 ARM922T 프로세서에 메모리 관리 유닛과 캐쉬를 더하고 32KB SRAM, 16KB DPRAM, 10만 게이트 FPGA, 인터럽트 컨트롤러 등이 SoC되어 있는 아키텍처를 갖고 있다.

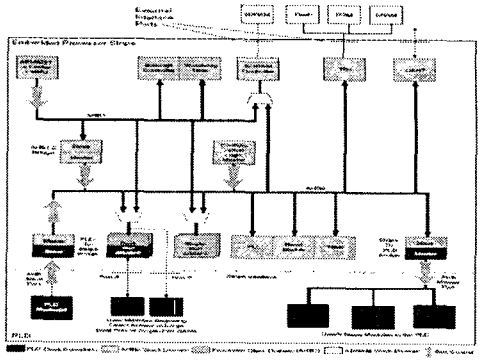

그림 2. 타겟 하드웨어 구조

비록 제안한 프로토타이핑 미들웨어가 특정 하드웨어에서 개발되었지만, 임베디드 리눅스 기반으로 동작하기 때문에 리눅스만 정상적으로 포팅되면 어떤 하드웨어에서도 제안한 미들웨어의 동작이 가능하다.

3.2 프로토타이핑 미들웨어

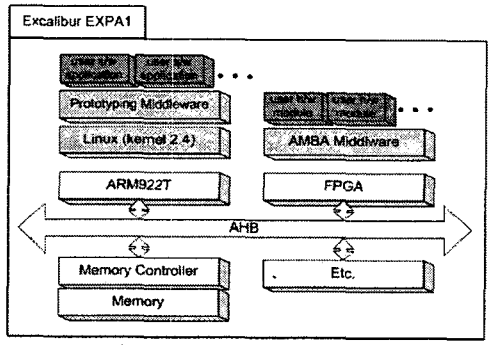

그림 3. 제안 미들웨어 아키텍처

제안한 미들웨어의 S/W부분은 미들웨어 구동 부분, 설정 로딩 부분, 공통 디바이스 드라이버 구현 부분, 사용자 모듈 실행 부분의 크게 4부분으로 되어 있다.

미들웨어 구동 부분은 미들웨어를 시작시키는 부분으로 나머지 부분들을 실행시키고 데이터를 넘겨주는 역할을 한다.

설정 로딩 부분은 미들웨어 구동 부분을 통해 동작하고, 정해진 설정 파일을 읽어 트리 구조로 만들고 사용자 모듈에서 사용될 구조체에 해당 내용을 설정한다. 설정 파일이 XML 형태이므로 파싱을 위해 expat이라는 SAX 형태의 라이브러리와 expat의 결과물을 DOM 형태로 바꾸어주는 scew라는 라이브러리를 사용하여 구현하였다[7]. 설정 파일에 XML 형식을 이용함으로서 설정 내용을 직관적으로 알 수 있고, 설정하면서 생기는 오류를 줄일 수 있다.

```
<pmconf>
    <debug>1</debug>
    <module>
        <name>libled.so</name>
        <device>/dev/pmdev</device>
        <defaddr>...</defaddr>
    </module>
</pmconf>
```

그림 4. 미들웨어 설정 XML 파일

공통 디바이스 드라이버 구현 부분은 사용자가 별도의 디바

이스 드라이버의 구현 없이 디바이스 드라이버 역할 대신 해준다. 이 구현부에서 제공하는 형태에 맞게 데이터 구조체를 작성한 후 이 구현부에 넘겨주면 된다. 작성하기 까다롭고 반복적인 작업인 디바이스 드라이버를 미들웨어에서 제공하므로 사용자는 더 쉽게 개발할 수 있다.

```
struct pm_dd_param_strct {
    char is_uint;
    unsigned int uint;
};
struct pm_dd_strct {
    unsigned int addr_base;
    int delay;
    int entry_cnt;
    struct pm_dd_param_strct params[MAX_ENTRY_NUM];
};

FILE* dd_open(char* dd_name);
int dd_close(FILE *fp);
void dd_write(struct pm_dd_strct* dd, FILE *fp);
void dd_read(struct pm_dd_strct* dd, FILE *fp);
```

그림 5. 공통 디바이스 드라이버 구조체 및 API

사용자 모듈 실행 부분은 사용자가 작성한 모듈을 동적 라이브러리 로딩을 이용하여 실행하는 부분으로 여러 모듈을 동시에 실행하는 것도 가능하다. 단, 사용자는 다음과 같은 함수를 반드시 구현해야 한다. 이 함수를 미들웨어가 실행함으로써 사용자 모듈이 동작한다.

```
void execute(FILE* fp, struct module* md_conf)
```

그림 6. 사용자 모듈 실행을 위한 필수 구현 함수

4. 제안한 미들웨어의 적용

제안한 프로토타이핑 미들웨어를 도어락 시스템에 적용한다. 확인한 모듈은 도어락 시스템에서 외부로부터 받은 도어락 명령 데이터를 3DES를 이용해서 복호화 하는 복호 모듈이다. H/W로는 DES를 구현하고, S/W로 이를 세 번 적용해 3DES를 구현한다. 적용 절차는 다음과 같다.

```
<pmconf>
    <debug>1</debug>
    <module>
        <name>lib3des.so</name>
        <device>/dev/pmdev</device>
        <defaddr>
            <addr name="core_sel" type="int" count="1">
                0x8000001c</addr>
            <addr name="key" type="int" count="1">
                0x80000004</addr>
            <addr name="data" type="int" count="1">
                0x80000008</addr>
            <addr name="eof" type="int" count="1">
                0x8000000c</addr>
            <addr name="en_de" type="int" count="1">
                0x80000010</addr>
            <addr name="read" type="int" count="1">
                0x80000014</addr>
            <addr name="read_eof" type="int" count="1">
                0x80000018</addr>
        </defaddr>
    </module>
</pmconf>
```

그림 7. 3DES 설정 XML 파일

사용자는 우선 HDL 언어를 이용해 구현한 DES 모듈, 미들웨어 설정 XML 파일, 3DES의 사용 로직을 구현한 복호 모듈을 준비한다. 그런 후 임베디드 리눅스에서 3DES 모듈을 pldconfig 명령을 이용해 FPGA 영역에 다운로드 한다. 마지막으로 설정 XML 파일과 작성한 복호 모듈을 지정된 곳에 놓고 미들웨어를 실행한다.

그림 8에서도 알 수 있듯이, 미들웨어를 통해 디바이스 드라이버를 작성하고, 이를 커널에 등록하고, 파일 디스크립터를 만드는 과정 없이, 간단히 작성한 모듈의 동작 확인을 위한 프로토타이핑을 할 수 있다.

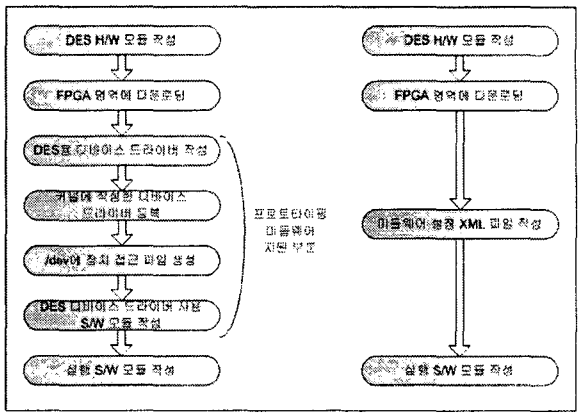

그림 8. 일반 방법과의 비교

5. 결론

본 논문에서는 플랫폼 기반 방법론을 프로토타이핑 과정에 적용한 프로토타이핑 미들웨어를 제안하였다. 제안한 방식은 플랫폼 기반 방법을 적용했기 때문에 빠른 제품 개발과 안정성 확보에 대한 요구를 충족시킬 수 있고, OS에 임베디드 리눅스를 사용해서 타겟 플랫폼의 제한이 없이 동작한다. 그리고 미들웨어의 구성이 XML 파일의 명세만으로 가능하게 하여 사용자의 접근성을 높였고, 통적 라이브러리를 통해 사용자 모듈만 컴파일 후 사용 가능하게 해서 편의성을 높였다.

향후에는 제안한 미들웨어의 H/W 부분과 관련해서 사용자에게 더욱 편하게 이용 가능하도록 하는 기능과 H/W, S/W 를 동시에 검증이 가능하도록 도와주는 기능에 대한 연구가 더 필요하다.

참고 문헌

[1] DeMicheli, G., "Readings in Hardware/Software Codesign", *Morgan-Kaufman, 2001*

[2] Kurt Keutzer, et.al., "System-Level Design: Orthogonalization of Concerns and Platform-Based Design", IEEE TCAD, 2000

[3] Neil C. Audsley, "Towards the Codesign of Large Complex Hard Real-Time Embedded Systems", CERTS'03, 2003

[4] Roman Gumzej, "A Configuration Manager for Embedded Real-Time Applications", CERTS'03, 2003

[5] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. A. Longstaff, and N. R. Mead, "An Approach to Survivable Systems," CERT Coordination Center, Software Engineering Institute, Carnegie Mellon University 1999.

[6] Altera Corporation, "Excalibur Hardware Reference Manual", available at
http://www.altera.com/literature/lit-exc.jsp

[7] W3C, http://www.w3.org/