SoC 액츄에이터 IP 구동을 위한 디바이스 드라이버

강상우, 박종성, 문철홍
광주대학교 전자광통신 대학원
전화 : (062)670-2652 / 팩스(062)670-2191 핸드폰 : 011-640-4716

Device driver for SoC actuator IP driving

Sang-Woo Gang, Jong-Seong Park, Cheol-Hong Moon
Gwagnju University
E-mail:zang-be@hanmail.net

Abstract

This paper describes an embedded system to put a SoC actuator IP in motion and linux drivers. The IP that a embedded linux among embedded OS is ported is implemented as linux driver. The actuator IP is controlled by application programming. To make users use this easily, a QT is ported on the system. Application program can operate the actuator IP device driver on TFT LCD.

I. 서론

현재 System on Chip 기술이 발전함에 따라 대형화 시스템이 소형화된 시스템으로 발전하고 있는 추세이다. 하나의 반도체 칩(chip)안에 시스템을 집적화 함으로서 여러 가지 기능들을 수행할 수 있는데 이런 시스템을 구축하기 위해서 임베디드 리눅스를 포팅하고 각각에 디바이스 드라이버를 구현함으로서 독립적인 시스템을 구현할 수 있다.

본 논문에서는 SoC 액츄에이터 IP 구동을 위한 임베디드 시스템을 구축하고, 리눅스 디바이스 드라이버를 구현한다. 임베디드 OS중 임베디드 리눅스를 탑재해 온칩화 된 IP을 리눅스 디바이스 드라이버로 제작해서 어플리케이션 프로그래밍으로 액츄에이터 IP을 제어하고 , 여러 사용자들이 편리하게 사용할 수 있도록 시스템에 QT를 적재해서 TFT LCD에 액츄에이터 IP 디바이스 드라이버를 구동시킬 수 있도록 어플리케이션을 프로그래밍 해서 임베디드 시스템을 구현하였다.

II. 임베디드 시스템

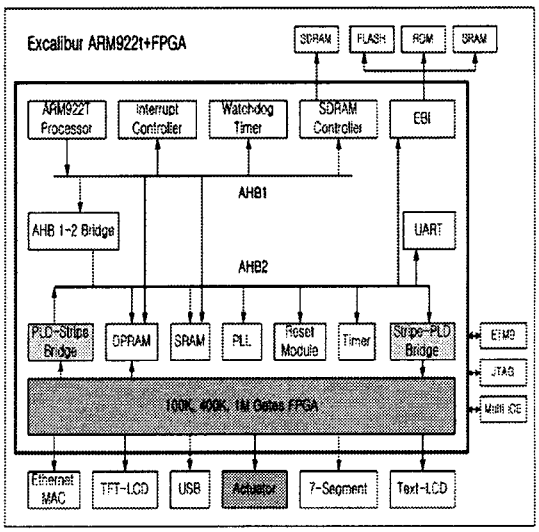

그림 1. ARM922T 임베디드 프로세서 내부 구조

임베디드 시스템은 일반적으로 특별한 업무를 수행하기 위한 하드웨어와 소프트웨어를 포함하는 특정한 응용시스템이라고 할 수 있다.

본 논문에서는 임베디드 시스템을 구성하기위해 엑스

칼리버 ARM922T 32bit RISC(최대 200MHz동작) 프로세서와 10만 게이트를 지원하는 APEX20KE를 사용하고, 리눅스 운용체제와 응용 어플리케이션 소프트웨어를 시스템에 탑재시키고 모니터링 할 수 있도록 TFT LCD에 리눅스용 윈도우즈를 적재 시키는 시스템을 구축한다.

III. 액츄에이터 IP(Intellectual Property)

3.1 SoC 액츄에이터 IP 블록도

SoC(System on Chip)는 하드웨어 로직뿐만 아니라 프로세서, 롬, 램, 컨트롤러, 주변장치의 회로를 하나의 칩에 집적화한 시스템이며 IP(Intellectual Property)들은 마이크로프로세서, DSP, 메모리 및 인터페이스 기능과 같고, 반도체 디바이스 내에 미리 정의된 기능을 블록화 하여 새로운 칩 설계시 재사용하므로 복잡한 칩을 짧은 시간 내에 생산하게 한다.

본 논문에서는 IP를 한 개의 블록으로 설계 했으며 이 블록과 엑스칼리버의 ARM922T 프로세서와 인터페이스 시켜서 시스템을 구성했다. 그리고 더 많은 IP들을 각각의 블록으로 설계하고 이 블록들을 프로세서와 인터페이스 시켜주면 하나의 시스템에서 여러개의 IP을 구동시킬 수 있다.

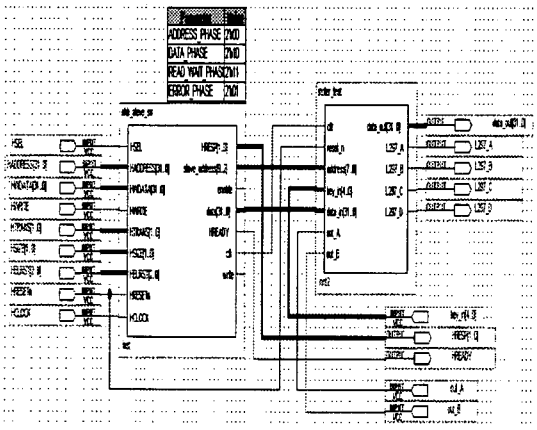

그림 2. 액츄에이터 모듈 블럭

그림 2는 액츄에이터 IP 모듈 블록을 보여준다. ARM922T 프로세서와 액츄에이터 모듈 블록을 인터페이스 시키면 엑스칼리버에서 동작 시킬 수 있는 하나의 IP가 된다.

3.2 액츄에이터 회로도

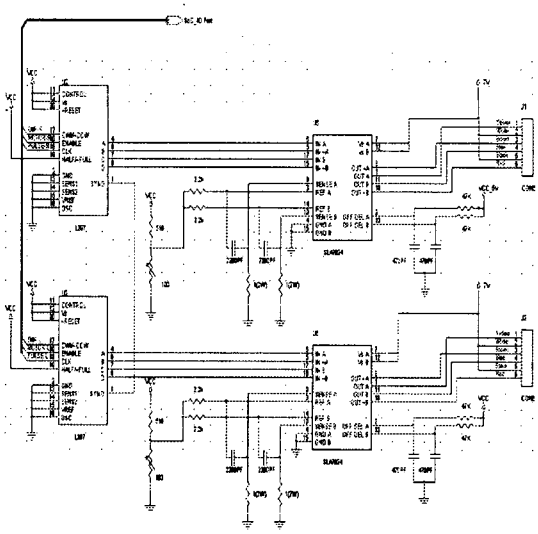

그림 3. Actuator 인터페이스 회로도

그림 3은 기계 부분과 SoC 시스템 하드웨어의 IP 로직을 인터페이스 시켜주는 회로도 이다.

3.3 액츄에이터 기계부와 시스템

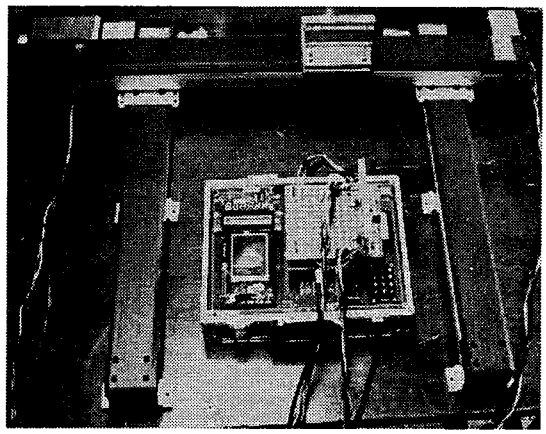

그림 4. 액츄에이터 기계부와 SoC 시스템

그림 4는 액츄에이터 2축 기계부와 SoC 시스템를 나타내고 있다. 2축 기계부는 상하 좌우로 움직일 수 있으며 SoC 시스템에는 IP, 인터페이스 하드웨어, 커널, 리눅스 응용프록그램이 동작 된다..

본 논문에서는 기계부의 상하동작은 하지 않고 좌우 동작만 제어 하도록 응용프그램을 작성하였다.

IV.액츄에이터 디바이스 드라이버

4.1 캐릭터 디바이스 드라이버

디바이스 드라이버는 컴퓨터에 장착된 여러 종류의 장치들을 하나의 공통된 인터페이스(파일 연산의 형태)로 접근할 수 있도록 만들어 주는 데이터 추상화 형식을 말하며, 사용자 응용프로그램이 드라이버에게 요청을 하면, 드라이버는 하드웨어 장치를 구동시킨다. 디바이스 드라이버에는 문자 장치(character device), 블록 장치(block device), 네트웍 장치(network device) 3가지 종류로 나눌 수 있다.

문자 장치는 버퍼를 통하지 않고 바로 읽고 쓸 수 있는 장치이고, 터미널, 시리얼, 패러럴, 키보드, 마우스, pc, 스피커등 있다. 블록 장치는 버퍼 캐시를 통해 입출력을 하며 장치의 어디든 접근이 가능하며 블록단위로 입출력을 하며 파일시스템을 구축할 수 있고, 플로피 디스크, 하드디스크, 램 디스크 등이 있다. 네트워크 장치는 네트웍 패킷을 송수신 할 수 있는 장치이고, 이더넷, ppp에 사용된다.

본 논문에서는 캐릭터 디바이스 드라이버를 사용해서 프로그래밍 했다.

4.2 디바이스 드라이버 커널 동작

디바이스 드라이버는 디바이스를 관리할 수 있는 인터페이스를 구현하기 위한 함수와 자료 구조의 집합이라고 할 수 있다. 커널은 이렇게 구현된 인터페이스를 통해 물리적인 디바이스에게 I/O 함수를 용청하고, 제어할 수 있다. 디바이스 드라이버는 인터페이스를 통하여 접근하는 데이터 형식이다.

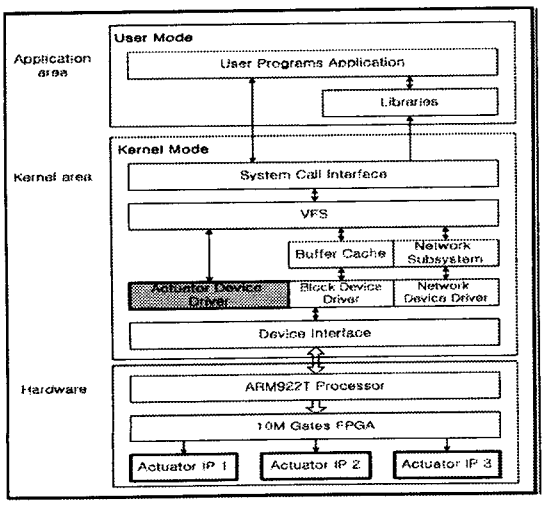

그림 5. 커널 동작 모드

그림 5는 커널 동작 모드를 보여준다. 커널이 존재하는 모든 OS는 커널 모드와 유저모드로 나뉜다. 커널 모드와 유저 모드를 연결 시켜주는 역할을 하는 것이 디바이스 드라이버이다.

4.3 디바이스 드라이버 기본 구조

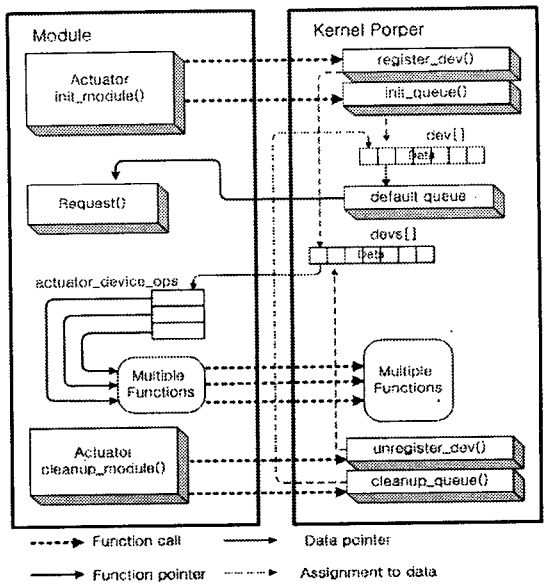

그림 6. 액츄에이터 디바이스 드라이버 엔트리 구조

그림 6은 액츄에이터 디바이스 드라이버를 등록시키고 삭제 시키는 구조를 나타내고 있으며 커널에서 액츄에이터 디바이스 드라이버를 관리 한다. 응용프로그램에서 데이터을 주면 커널에서 데이터를 처리해서 디바이스 드라이버로 그 값을 넘겨준다.

4.4 디바이스 드라이버 동작

유닉스와 리눅스는 모든 디바이스를 파일로 인식을 한다. 그래서 디바이스를 사용하기 위해서는 파일을 열고 모든 수행이 끝나면 파일을 닫아야 한다.

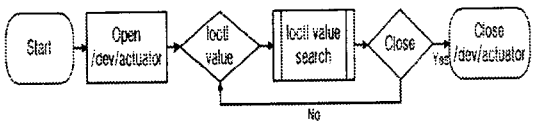

그림 7. 디바이스 드라이버 동작 개략도

그림 7은 본 논문에서 작성된 디바이스 드라이버가 동작하는 개략도를 보여준다.

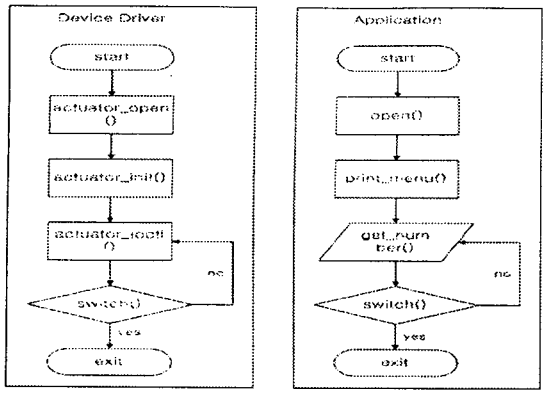

그림 8. 액츄에이터 순서도

그림 8은 디바이스 드라이버 와 어플리케이션의 내부 동작 순서를 나타내고 있다. 디바이스 드라이버를 모듈을 통해서 커널에 등록을 시켜주고 등록된 커널의 장치를 응용프로그램에 의해서 동작을 시킨다.

4.5 디바이스 드라이버 응용프로그래밍

그래픽 유저 인터페이스(GUI)를 위하여 윈도우즈에서는 API 함수를 이용하면 윈도우 프로그래밍이 가능하다. 리눅스도 윈도우즈 프로그래밍이 가능하다. KDE는 리눅스의 데스크탑 환경에서 QT 라이브러리를 사용하며 GNOME보다 프로그램 시작이 빠르다. QT는 C++로 만들어졌으며 그 구조는 MFC와 비슷하며 기존의 C 개발 방식에 비해서 훨씬 쉽고, 기능 확장이 편리해서 생산성이 높다.

본 논문에서는 QT를 선택했으며 QT를 실행하기 위해 유저 파일시스템에 QT 라이브러리와 실행파일을 포함시켜서 시스템에 적재 시켰다. 그리고 시스템의 TFT LCD 상에서 디바이스 드라이버를 실행가능 하도록 구축했다.

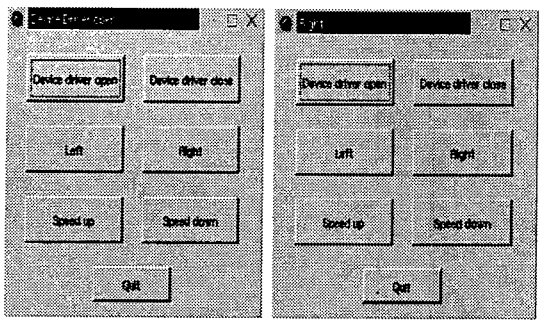

그림 8. QT를 이용한 어플리케이션

그림 8은 QT를 이용한 리눅스 윈도우즈 프로그래밍을 나타내고 있다.

V. 결론

본 논문에서는 SoC 시스템 엑스칼리버 (ARM922T 32bit RISC 프로세서와 프로그래머블 로직 디바이스 FPGA)을 사용했으며 이 엑스칼리버 소자의 FPGA 영역에 액츄에이터 IP를 다운로드 시킨다.

액츄에이터 IP을 임베디드 리눅스에서 동작시키기 위해 커널을 포팅하고, 리눅스 디바이스 드라이버를 작성하고, 리눅스 윈도우즈에서 디바이스 드라이버를 동작 시킬 수 있는 어플리케이션 프로그래밍을 구현해 보았다. SoC 시스템상에 여러개의 IP을 동작시키기 위해 각각의 디바이스 드라이버와 어플리케이션 프로그래밍은 제작 하지는 않았지만, 한개의 IP에 대한 디바이스 드라이버 제작과 어플리케이션을 프로그래밍해서 동작시킬 수 있었다.

만일, 여러개의 IP을 엑스칼리버의 FPGA 영역에 집적시킨다면 임베디드 리눅스에서 각각에 IP을 디바이스 드라이버로 제작과 IP을 동작 시킬 수 있는 어플리케이션 프로그래밍 가능하다.

참고문헌(또는 Reference)

[1]O'REILLY, "LINUX DEVICE DRIVERS", RUBINI & CORBET.
[2]영한출판사 "Verilog HDL" SAMIR PALNITKAR
[3]한빛미디어 "Linux&Unix C 프로그래밍" 김종훈·김종진·김동균
[4]PC'BOOK "임베디드 리눅스 프로그래밍" 이연조
[5]ARM Architecture Reference Manual
[6]ARM922T(Rev 0) Techical Reference Manual
[7]Excalibur Device
[8]AMBA™ Specification(Rev 2.0)