이동단말기용 ARM 프로세서 기반 ROM 프로그래밍

박 관 민*, 정 민 수*, 김 도 우*, 진 민 식*
*경남대학교 컴퓨터공학과

ARM Processor ROM programming for Mobile Terminal

Kwan-Min Park*, Min-Soo Jung*, Do-Woo Kim*, Min-Sik Jin*
*Dept. of Computer Engineering, Kyungnam University

요 약

ARM 프로세서는 이동 단말기의 표준으로 채택된 32 비트 범용 마이크로프로세서이다. 이동 단말기 내에서 ARM 프로세서는 실시간 운영체제 커널과 운영체제 그리고 API 로 구성되며 소형의 단말기에서 응용 프로그램을 운용하기 위해서는 ROM 에 응용 프로그램을 상주시켜 실행시켜야 한다. 이러한 ROM 상주를 위해서는 ROM 영역에 적재하기위한 메모리 구성이 중요하다.

본 논문은 메모리 구성을 위한 메모리 이미지의 형성과 메모리의 맵을 통한 ROM 상주 프로그램의 적재 그리고 시스템 초기화를 통해서 응용 프로그램의 수행을 보장하는 방법론에 대해 분석한다.

1. 서 론

ARM 프로세서(processor)는 범용 마이크로프로세서의 일종으로 32 비트 RISC(Reduced Instruction Set Computer) 구조를 가지며 전력 소비가 적고 비용이 저렴할 뿐만 아니라 높은 성능을 가지고 있어 디지털 무선 통신의 표준으로 채택된 프로세서이다.

기본적으로 이동 단말기들의 구조는 ARM 프로세서를 기반으로 실시간 운영체제(RTOS) 커널과 운영 체제 그리고 API 들로 구성되어진다. ARM 프로세서에 사용하는 일반적인 응용 프로그램(application)은 ROM 에 적재되어 사용되며 내장되는 ARM 응용 프로그램의 설계에 있어 가장 중요한 고려사항은 메모리 맵(memory map)의 배치에 있다.

효과적인 메모리 맵의 배치를 위해서는 적재(load)시 메모리 맵의 이미지(image)를 생성하여 링크시에 실행을 위한 메모리 이미지를 메모리에 재배치한다.

또한, ARM 프로세서를 위해 두 가지 상태의 메모리 맵을 가지며 각 상태에 따른 메모리 맵의 구성도 달라지게 된다. 본 논문의 구성은 2 장에서 메모리 맵을 구성하기위한 메모리 이미지에 대하여 설명하고 3 장에서 메모리 맵의 구성에 대하여 기술한다. 그리고 4 장에서 ROM 프로그램을 위해 필요한 시스템 초기화하기위한 코드에 대하여 기술하였고 5 장에서 ROM 상주 프로그램의 메모리 맵의 전반적인 구성을 살펴보고 6 장에서 결론 및 향후 연구 방향에 대해 제시한다.

2. 메모리 이미지(Memory Image)

메모리 이미지를 위해서는 메모리를 리젼(region), 세션(section) 그리고 에어리어(area) 영역으로 구분하여 블록(block)화한다. 에어리어 영역은 코드(code)와 데이터 항목(date item)들이 존재하며 세션은 동일한 속성(attribute)을 가지는 주소 공간의 연속적인 에어리어들의 연속이다. 그리고 리젼 영역은 인접한 세션들의 연속으로 동일한 속성을 가지지는 않는다.

메모리 이미지는 하나 또는 그 이상의 세션 블록들로 구성된 것을 말하며 가장 간단한 이미지의 경우 read-only(RO) 세션(section), read-write(RW) 세션 그리고 zero-initialized(ZI) 세션으로 구성된다.

이미지 리젼은 적재(load)시에 시스템 메모리 맵에 위치되어지고 이미지를 실행하기 전에 실행 위치에 대한 영역으로 할당되고 이미지 파일에 존재하는 정보를 통해 zero-initialized 세션을 생성한다.

read-write 세션은 코드를 가지며 read-only 세션은 read-only 데이터를 가진다. 따라서, 이미지의 메모리 맵은 적재와 실행(execution) 두가지 관점에서 살펴 볼 수 있다.

그림 2-1 과 같이 적재의 관점에서 이미지는 read-only 세션과 read-write 세션은 메모리 맵에 연속적으로 위치하며 실행시의 관점에서 살펴보면 3 개의 실행 세션을 가지는 두 개의 실행 리젼을 가진다.

read-only 실행 세션과 zero-initialized 세션은 연속적으로 위치한다.

read-only 세션은 읽는 것만이 요구되는 항목들 즉, 프로그램 코드나 상수등이 존재하고 read-write 세션에는 읽고 쓰기위한 항목인 데이터나 변수 등이 존재한다. 그리고, zero-initialized 세션에는 0 으로 초기화될 항목들이 놓여지게 된다.

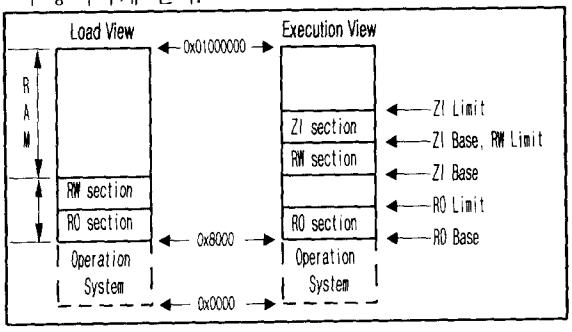

그림 1 적재관점과 실행 관점에서의 이미지 맵

3. 메모리 맵의 구성

ARM 프로세서의 메모리의 맵은 그림 2 와 같이 응용 프로그램의 수행에 의한 리셋(Reset) 상태와 Address Remap register 에 의한 전형(Normal) 상태의 두가지 상태를 가진다.

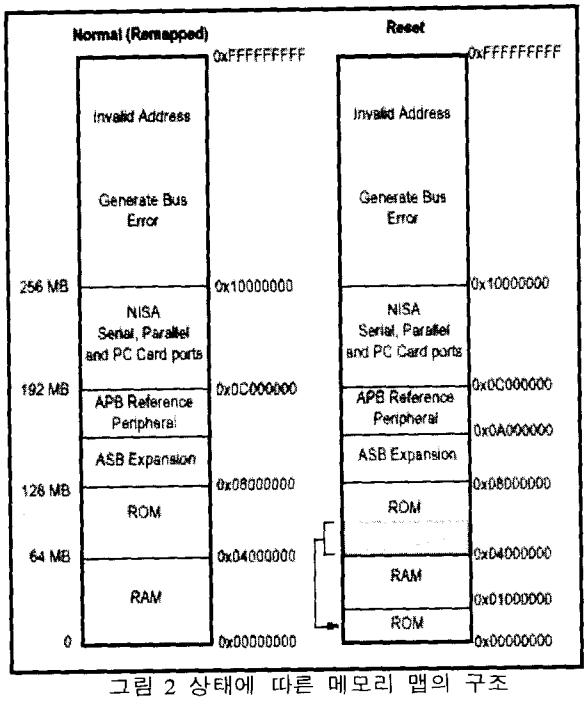

그림 2 상태에 따른 메모리 맵의 구조

전형 상태에서는 RAM 의 영역에 응용 프로그램이 적재되어 사용되며 RAM 내에서 인접한 메모리 영역을 사용한다. 그리고 리셋 상태에서는 ROM 에 올려진 응용 프로그램이 존재할 경우 응용 프로그램의 읽고 쓰는 것이 요구되는 항목들을 RAM 의 영역에 지정된 ROM 의 대체 영역으로 리맵되어진다.

3.1 RAM 과 ROM 의 메모리 영역

RAM 영역은 그림 3 에서와 같이 16MB 단위로 DRAM, SRAM, SSRAM 그리고 대체를 위한 영역의 4 개로 나누어진다. 0x0 번지에서 시작하는 4 개의 16MB 블록은 전형 환경에서는 대체 영역에 RAM 의 분할된 각 영역들을 오버라이드(overiad)하여 사용한다. 리셋 상태에서는 대체를 위한 영역이 ROM 영역의 베이스(base)를 지시하도록 변경된다

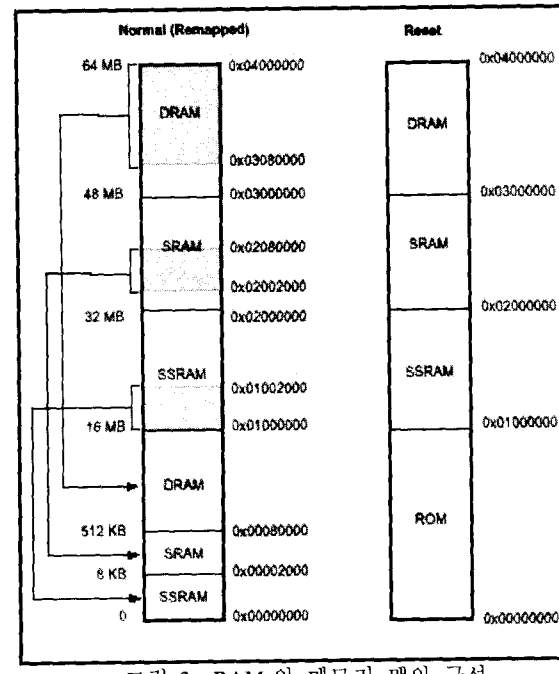

그림 3 RAM 의 메모리 맵의 구성

그림 3 에서 RAM 영역에 대한 대체는 각 상태에 따라 지정된 영역을 가진다. 디폴트(default) 값으로 리셋 상태를 가지면 remap register 에 의해 ClearResetMap 이 set 되어지면 전형 상태는 리셋으로 변경된다.

SSRAM 영역은 인터럽터 조절자(interrupt handler)와 같은 임계시간 루틴(time-critical routine)에 이용되면 SRAM 은 다양한 메모리 스키마(scheme)를 에뮬레이트(emulate)하기위해 사용된다. 그리고 DRAM 은 페이지 모드(page mode)와 버스터 모드(burst mode)의 순차적인 접근을 지원하고 바이트(byte), 하프워드(halfword) 그리고 워드(word) 의 처리를 지원한다.

ROM 영역은 64MB 크기로 전형 상태와 리셋 상태 모두 동일하게 메모리 맵을 형성하고 있다. ROM 의 영역은 0x04000000 부터 0x08000000 까지 제공된다.

일반적인 응용 프로그램이 ROM 에 할당되었을 경우

리셋 상태에서 remap register 는 RAM 의 영역의 가장 아래부분으로 ROM 에 할당된 응용 프로그램을 대체하는데 이 경우 RAM 에 대체되는 것은 ROM 전체 코드가 아니라 읽고 쓰는 것이 요구되는 항목만이 RAM 의 대체영역으로 리맵되어지고 다른 읽기만 가능한 항목은 계속 ROM 에 존재하며 RAM 에 대체된 항목에서 ROM 으로 제어가 넘어갈 때는 link register 의 값을 참조하여 제어가 전환된다.

4. 초기화 코드를 통한 시스템 초기화

ARM 프로세서에서는 운영 체제 없이 ROM 의 애플리케이션 코드가 스스로 초기화하고 실행을 시작하는 것이 중요하다. 이를 위해 초기화 코드가 필요하다.
초기화 코드를 통해서 시스템에 전반에 대한 초기화를 정의하고 ROM 에 상주하는 응용 프로그램의 실행으로 전환할 수 있도록 해준다.
시스템 초기화 코드는 엔트리 포인트(entry point)를 정의하여 응용 프로그램의 메인(main)으로 프로그램의 포인터를 전환하게 해주고 ROM 이 0 번지에 할당될 경우 예외 처리기로 분기하도록 하거나 0 번지가 아닌 경우 초기화 코드를 통해 벡터(vector)를 동적으로 초기화되도록 read-write 세션의 베이스(base)를 명시하여 예외 벡터(exceptoion vector)를 설정한다.
그리고, 인터럽트와 특정한 번지에 접근하기위해 코드를 호출하기 전에 메모리 시스템을 초기화하고 인터럽트와 예외를 위한 스택 포인터를 초기화하기위해 스택 포인터 레지스터를 초기화한다.
또한, 입출력 디바이스를 초기화하고 인터럽트 시스템에 필요한 RAM 변수들을 초기화한다.
메인 프로그램에 필요한 메모리 영역을 초기화하며 인터럽트가 가능하도록 설정해 준다. 그리고 프로세서 모드를 사용자 모드로 전환시켜 준다.
모든 설정이 완료되면 안전하게 메인 프로그램으로 전환하게 응용 프로그램을 실행할 수 있다.

5. ROM 상주 프로그램의 메모리 맵 대체

그림 4 에서 보는 것과 같이 원시 프로그램의 코드는 일반적으로 0 번지부터 순차적으로 초기화 코드 그리고 메인 프로그램 코드와 예외 벡터, 예외 처리기, 초기화된 read-write 데이터가 놓여지게 된다.
적재시에 ROM 에 상주될 프로그램이 ROM 의 영역으로 대체되어지고 연결(link)를 통해서 지정된 메모리 영역으로 리맵되어 실행을 위한 메모리 구성을 가지게 된다
초기화 코드를 통해서 예외벡터와 예외 처리기가 메모리 맵의 지정된 영역으로 놓여진다. 즉, 시스템의 제어를 지정하고 ROM 의 읽고 쓰기위한 데이터들을 지정된 영역으로 리맵한다. 그리고 읽고 쓰는 영역의 인접해서 zero-initialized 코드가 놓여지게 된다.
결국 실행시 ROM 의 영역에는 읽기만을 위한 프로그램 코드와 시스템 초기화에 사용된 초기화 코드만이 존재한다.

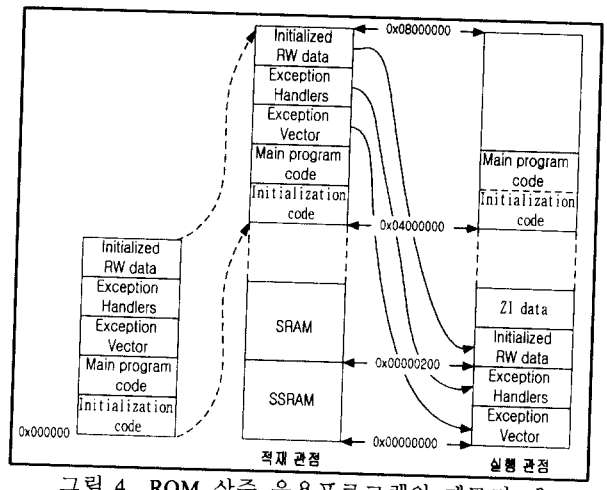

그림 4 ROM 상주 응용프로그램의 메모리 맵

6. 결론 및 연구 결과의 활용

ARM 프로세서를 기반으로한 ROM 영역의 응용 프로그램은 특정한 운영 체제없이 수행가능하며 이를 위해서는 시스템을 초기화하고 안전한 애플리케이션 수행을 보장한다.
본 논문의 기법을 이용하면 스마트폰, PDA 와 같은 디지털 이동 단말기나 디지털 가전기기의 다양한 기능을 지원하기위한 응용 프로그램의 개발에 적용될 수 있을 것이다.

참 고 문 헌

[1] Steve Furver, "ARM System Architecture", Addison-Wesley
[2] http://www.arm.com
[3] ARM Limited, "ARM Architecture Reference Manual"
[4] ARM Limited "Interfacing a Memory System to the ARM7TDM without using AMBA"
[5] ARM Limited "AMBA Specification"
[6] ARM Limited "Reference Peripherals Specification"