KIPS Tr. Comp. and Comm. Sys.
Vol.3, No.7 pp.209~218 pISSN: 2287-5891

http://dx.doi.org/10.3745/KTCCS.2014.3.7.209

Design and Implementation of Snapshot Startup Method for Fast Subsystem Startup

Jun Kim† · Joonwon Lee†† · Jinkyu Jeong†††

ABSTRACT

An AP that is used by smart device is going to be complicated because smart devices support diverse functions. As a result, multiple low-level IPs including a dedicated CPU are integrated in a high-level subsystem for supporting complicated function such as multimedia codec and camera. A subsystem has software that executes separately from main system, and the software needs to be initialized for every execution of the subsystem. This causes increase of the subsystem startup time so it should be improved because startup time of subsystem affects launching time of application. Methods in applied to computer system for fast startup also could be applied to fast startup of subsystem because subsystem is similar with computer system. In this paper, we apply snapshot method that is used in computer system to subsystem and analyzes the pros and cons. And snapshot method could not be applied to register of IP without modification because register of IP offers restricted read and write. So this paper suggests technique that applying snapshot to each characteristic of register.

Keywords : Subsystem Initialization, Fast Startup, Snapshot Startup

서브시스템의 빠른 구동을 위한 스냅샷 구동 기법 설계 및 구현

김 준† · 이 준 원†† · 정 진 규†††

요 약

스마트 디바이스가 다양한 기능을 지원하면서 스마트 디바이스에서 사용되는 응용프로그램 프로세서 또한 복잡해졌다. 그 결과 멀티미디어 코덱과 카메라 같은 복잡한 기능을 지원하기 위해 AP내부에서 전용 CPU를 포함한 여러 개의 저수준 IP가 하나의 고수준 기능을 제공하는 서브시스템으로 통합되고 있다. 서브시스템은 메인 시스템과는 별도의 소프트웨어를 가지며, 서브시스템의 구동 시 자체 소프트웨어를 초기화하는 과정이 필요하다. 이는 서브시스템의 구동 시간을 늘리는 원인이 되며 서브시스템의 기능을 사용하는 응용프로그램의 구동 시간에도 영향을 미치기 때문에 개선될 필요성이 있다. 서브시스템은 컴퓨터 시스템과 유사하므로 컴퓨터 시스템의 빠른 구동을 위해 연구되었던 기법들을 서브시스템의 빠른 구동을 위해 적용할 수 있다. 본 논문에서는 컴퓨터 시스템에서 사용하고 있는 스냅샷 기법을 서브시스템에 적용한 후 장단점을 고찰하였다. 그리고 IP의 레지스터는 제한된 읽기와 쓰기를 제공하기 때문에 스냅샷 기법을 수정하지 않고 적용할 수 없다. 이를 위해 본 논문에서는 IP의 레지스터 특성별로 스냅샷 기법을 적용하는 기법을 제시하였다.

키워드 : 서브시스템 초기화, 빠른 구동, 스냅샷 구동

1. 서 론

Apple이 2007년에 iPhone을 선보이면서 시작된 스마트폰 시장의 폭발적인 성장세는 현재까지 계속되어 2013년 1분기에는 스마트폰이 피처폰의 출하량을 넘어섰다[1]. 시장이 치열해짐과 더불어 성능에 대한 소비자의 요구 사항은 지속적으로 상승하여 스마트폰을 필두로 한 스마트 디바이스 제품들은 카메라, 멀티미디어, 그리고 3D 등의 기능과 성능을 고도화시키는 방향으로 발전하고 있다. 이에 따라 스마트 디바이스의 하드웨어는 점차 복잡해져 가고 있는데 그 중심에는 응용프로그램 프로세서(AP: application processor)가 있다.

AP는 스마트 디바이스의 핵심이 되는 칩셋으로 명령어를 해석하고 연산을 수행하는 CPU와 여러 가지 추가 기능을 제공하는 하드웨어 블록 또는 IP(intellectual property)을 하

※ 이 논문은 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음[10041244, 스마트TV 2.0 소프트웨어 플랫폼].
† 비 회 원 : 성균관대학교 반도체디스플레이공학과 석사과정
†† 정 회 원 : 성균관대학교 컴퓨터공학과 교수
††† 정 회 원 : 성균관대학교 반도체시스템공학과 조교수
Manuscript Received : April 7, 2014
First Revision : June 23, 2014
Accepted : June 24, 2014
* Corresponding Author : Jinkyu Jeong(jinkyu@skku.edu)

나의 칩으로 제작한 것이다. AP에 포함되는 하드웨어 블록은 2D/3D 그래픽 프로세서, 이미지 처리 프로세서 (ISP: image signal processor) 및 외부 인터페이스 블록 등이 있다.

AP내의 여러 개의 IP들은 공통된 목적을 위해 서브시스템을 구성할 수 있다. 일반적으로 하나의 IP는 하나의 기능을 목적으로 설계되며 시스템에서 소프트웨어 드라이버 작성의 기준이 된다. 하지만 멀티미디어 코덱과 카메라와 같이 하드웨어로 구현되기에 복잡도가 높은 기능들은 여러 개의 IP가 하나의 목표로 통합적으로 설계가 될 필요성이 대두되었다. 서브시스템은 이러한 포괄적인 목적을 위해 다수의 IP들이 하나의 통합적 기능으로 구동하는 시스템이다. 예를 들어 이미지 서브시스템은 카메라라는 큰 기능을 목표로 하면서, 사진 촬영을 위한 노이즈 제거(noise reduction), 영상 안정화(image stabilization), 스케일러(scalar) 등의 세부적 기능들이 각각 IP의 형태로 구성되어 있다.

이러한 서브시스템은 각 서브시스템에 필요한 IP들과 제어하기 위한 소프트웨어 및 저사양 CPU로 구성된다. 또한 서브시스템은 메인 시스템에서 서브시스템의 기능이 요구될 때 초기화 되고 구동된다. 예를 들어 메인 시스템에서 카메라 응용프로그램이 구동되면 이미지 서브시스템은 초기화 과정을 거친 후 카메라 응용프로그램과 상호작용을 수행한다. 문제는 보다 다양한 기능을 지원하기 위해 서브시스템의 소프트웨어가 복잡해질수록 서브시스템의 초기화 시간이 증가된다는 것이다. 이런 초기화 시간 증가는 결국 카메라 응용프로그램과 같이 서브시스템 의존적인 응용프로그램의 구동 시간을 증가시켜 사용자의 불만으로 이어질 수 있다.

본 논문은 서브시스템의 구동(startup) 시간을 개선하기 위해 스냅샷에 기반한 서브시스템 구동 기법을 제시한다. 이는 하이버네이션(hibernation)을 통해 생성된 메모리 스냅샷을 이용하여 빠른 부팅을 제공하는 스냅샷 구동 기법[2]을 서브시스템에 적용하는 방법이다. 하지만 서브시스템은 범용 시스템과는 다르게 각 장치드라이버가 하이버네이션을 제공하지 않는 경우가 많고, 메모리 기반 I/O(memory mapped I/O)를 통해 관리되는 IP들의 레지스터들의 특성에 따라 메모리 스냅샷의 정확도가 떨어지는 문제점을 가진다. 이러한 문제점을 해결하기 위해 본 논문에서는 각 장치의 레지스터 특성에 맞는 스냅샷 저장 기법을 제안하였다.

제안하는 방법은 삼성 Exynos 5250 기반의 시스템에서 이미지 처리 서브시스템에 적용 구현되어 실험되었다. 이미지 처리 시스템의 구동 성능은 제안된 방법을 적용한 결과 일반 구동의 17% 수준의 빠른 구동 시간을 보이며, 관련 연구 대비 33%의 구동 시간 단축을 보였다.

본 논문의 이후 구성은 다음과 같다. 먼저 2장에서는 서브시스템에 대한 일반적인 구조를 설명한 후, 이해를 돕기 위한 예제로 이미지 서브시스템에 대해서 살펴볼 것이다. 그리고 관련된 연구로 범용 시스템에서 부팅 시간을 줄이기 위해서 고안된 스냅샷 기반 부팅 방식의 장점과 응용 및 한계점을 서술한다. 3장에서는 서브시스템의 구동 시간을 개선하기 위해 제안한 스냅샷 기반 부팅 방식의 서브시스템 적용 방안에 대해 서술한다. 4장에서는 제안하는 방법의 성능 평가 결과를 제시한다. 5장에서는 결론과 앞으로의 연구 계획을 제시한다.

2. 배경지식 및 관련 연구

2.1 서브시스템

서브시스템은 다수의 저수준 기능을 제공하는 IP가 모여 하나의 고수준 기능을 제공하는 시스템이다. 각각의 IP는 독립적인 기능을 가지고 있지만 개별적인 구동은 의미가 없고 하나의 목적을 위해 서브시스템 단위로 구동된다. 예를 들어 멀티미디어 코덱(multimedia codec)의 경우 메모리로부터 데이터를 읽고 쓰기 위한 DMA(direct memory access) IP, 인코딩과 디코딩을 위한 IP가 각각 존재한다. 하지만 응용프로그램 수준에서는 각각의 IP를 구분하지 않고 멀티미디어 코덱이라는 집합의 개념으로 접근한다.

서브시스템이 하드웨어로만 구성되어 있으면 AP 내의 주 CPU만으로 서브시스템 내 IP의 관리에 있어 실시간성(realtimeness)과 유연성(flexibility)이 떨어지기 때문에 각 서브시스템은 IP를 구동하기 위한 CPU(보조 CPU)와 소프트웨어를 포함하여 구성된다[3]. Texas Instrument의 Omap AP에 속한 Ducati가 서브시스템의 좋은 예로, Ducati는 코덱을 위한 비디오 서브시스템과 카메라를 위한 이미지 서브시스템이 합쳐진 형태로 구성되어 있다[4]. 또한 운영을 위해서 2개의 서브 CPU를 포함하며 각각 호스트와 통신과, 내부 하드웨어의 제어를 담당한다.

보조 CPU가 포함된 서브시스템은 단일 컴퓨터 시스템과 유사하다. 메모리는 메인 시스템의 메모리 일부를 공유하여 사용하며, IP들이 디바이스로 장착되어 있다. 서브시스템을

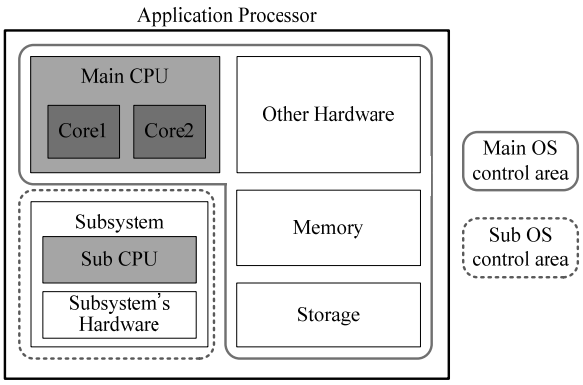

Fig. 1. The architecture diagram of a subsystem

제어하기 위해 서브 CPU에서 구동되는 전용 소프트웨어가 필요하며 이는 메인 시스템의 소프트웨어와는 독립적으로 동작한다. 특히 복잡한 서브시스템의 경우에는 여러 개의 태스크를 실시간으로 운영해야 하므로 전용 소프트웨어에 간단한 실시간 운영체제(RTOS: Real-time Operating System)가 탑재된다. 즉 그림 1과 같이 하나의 AP내에서 두 개의 운영체제가 동시에 동작하게 되는 것이다. 보통 메인 OS는 리눅스나 윈도우와 같은 범용 운영체제를 사용하고 서브시스템은 훨씬 단순한 펌웨어나 실시간 운영체제가 구동된다.

서브시스템은 주로 응용프로그램이 하드웨어 기능을 필요로 할 경우 구동되어 동작한다. 서브시스템의 구동 과정 자체는 그림 2와 같으며, 일반적인 컴퓨터 시스템과 크게 다를 바가 없다. 메인 시스템은 미리 약속된 특정 주소의 메모리에 서브시스템에서 사용할 소프트웨어 바이너리를 탑재한 후에 보조 CPU를 깨운다. 이후 메인 시스템은 서브시스템이 초기화될 때까지 대기하게 된다. 보조 CPU는 일반적인 운영체제의 부팅 과정과 유사하게 자체적인 운영체제 초기화를 수행한 후, 하드웨어와 소프트웨어 초기화 과정을 거치게 된다. 모든 초기화가 완료되어 서브시스템이 고수준 기능을 제공할 준비가 되면, 메인 시스템에 인터럽트를 전달하여 준비된 상태임을 보고한다. 서브시스템이 정상적으로 초기화된 것을 확인한 후에 메인 시스템은 관련된 응용프로그램 구동 등 다음 작업을 수행하게 된다.

서브시스템의 종류에 따라서 하드웨어 초기화와, 소프트웨어 초기화의 비중은 다르게 된다. 본 논문에서 지칭하는 서브시스템의 구동 시간은 서브시스템이 지원하는 핵심 기능을 수행할 수 있도록 보조 CPU가 깨어난 이후부터 하드웨어, 소프트웨어 초기화를 포함한 모든 준비과정을 마치는 순간까지를 지칭한다.

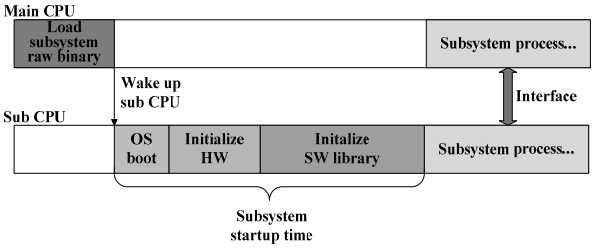

Fig. 2. The startup sequence of a subsystem

서브시스템 중에 대표적인 것으로 이미지 서브시스템을 들 수 있다. 이미지 서브시스템은 스마트폰과 같은 기기에서 카메라 기능을 담당하는 장치이다. 이미지 서브시스템은 이미지 센서로부터 들어오는 Bayer 데이터[1]를 YUV 또는 RGB로 변환하는 동시에 3A라고 불리는, 자동 초점(auto focusing), 자동 노출(auto exposure), 그리고 자동 화이트 밸런스(auto white balance) 연산을 포함하여 영상에 관한 알고리즘을 수행 한다(그림 3 참고). 또한 이미지 센서로부터 나오는 영상이 프레임 단위로 처리가 되기 때문에 정확한 타이밍에 이미지 센서를 제어하기 위해서 실시간성이 보장되어야 한다.

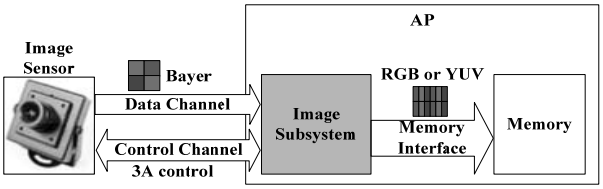

Fig. 3. The structure of an image subsystem

이미지 서브시스템의 경우에는 그림 2의 과정을 기준으로 살펴보았을 때 하드웨어 초기화 과정에 서브시스템 내부의 하드웨어뿐만 아니라 AP 외부에 위치한 이미지 센서의 초기화도 포함되어야 한다. 외부의 이미지 센서 초기화는 i2c와 같은 통신을 통해 보조 CPU에 의해 수행된다. 또한 소프트웨어 초기화 과정에 3A를 비롯한 영상 처리 알고리즘 소프트웨어 라이브러리도 같이 초기화 된다. 이미지 센서 초기화 시간 및 영상 처리 알고리즘의 초기화 시간은 제공하는 벤더별로 다양하기 때문에 이미지 서브시스템의 구동 시간은 하드웨어 및 소프트웨어의 구성에 따라 다양하게 나타난다.

서브시스템의 구동 과정은 전적으로 보조 CPU에 의해서 수행되므로 보조 CPU의 성능이 서브시스템 구동 시간의 가장 중요한 결정 요소가 된다. 하지만 보조 CPU는 범용적인 환경에서 동작하는 것을 고려하지 않고 특정 하드웨어가 동작할 때 보조하는 역할을 가정하고 설계가 되었기 때문에 주 CPU에 비해서 낮은 성능을 가지고 있다. 또한 서브시스템이 제공하는 주 기능(예: 카메라 구동)에 필요한 작업량을 기반으로 보조 CPU의 사양을 결정하므로 초기화 과정에서의 작업량은 보조 CPU의 사양을 결정할 때 고려 대상이 되지 않는다. 그렇기 때문에 서브시스템에서 제공하는 기능이 보다 고도화 되면서, 필요한 라이브러리와 소프트웨어의 초기화 시간이 길어지면서 서브시스템의 구동시간 전체를 길게 가져오는 문제를 발생시킨다. 이는 결국 해당 서브시스템을 사용하는 응용프로그램의 응답성을 떨어뜨리는 문제로 연결된다.

2.2 관련 연구

서브시스템은 일반 컴퓨터 시스템과 유사하므로 범용 운영체제의 구동 시간을 줄이기 위한 연구들과 유사성이 높다. 이 중 Bird는 리눅스의 구동 시간을 줄이기 위해서 여러 가지 방법을 제시하였다[5]. 여러 방법들 중에서, 압축되지 않은 이미지를 이용해서 ROM에서 바로 실행하는 Execute-

1) 한 픽셀당 한 개의 색상 정보만 가지고 있는 데이터로 이미지 센서가 피사체를 캡쳐(capture)할 때 일반적으로 생성하는 포맷.

In-Place(XiP) 기법이나 지연(delay) 분석을 통해 관련 항목을 최대한 회피한 것은 서브시스템의 구동에도 적용할 수 있다. 또한 quiet 옵션으로 디버깅 메시지를 비활성화 시킴으로써, 구동 시 출력되는 다수의 디버깅 메시지로 인한 병목을 회피하였다. 서브시스템과 같은 내장형 시스템은 주로 UART(universal asynchronous receiver and transmitter) 시리얼 출력을 이용하여 메시지를 출력한다. 이때 UART는 상대적으로 저속이기 때문에 많은 메시지 출력은 병목으로 작용하기 때문이다.

Kaminaga가 제시한 스냅샷 구동(snapshot boot)은 리눅스 커널에서 사용하는 suspend-resume 기법을 부팅 시에 이용하여 부팅 시간을 단축하는 방법이다[2]. 이 방법은 리눅스의 suspend-to-disk를 활용하여 suspend시에 스냅샷(snapshot) 바이너리를 한 번 생성하고, 그 이후의 부팅 시에는 스냅샷 바이너리를 바로 로딩하여 일반적인 부팅 과정 대신에 재개(resume) 과정을 진행한다. 복잡한 시스템일수록 부팅 과정에서 수행하는 전역 구조체 초기화, 객체 생성, 응용프로그램 초기화 및 프로세스 생성 등 소프트웨어의 초기화 과정이 부팅 시간 중 가장 많은 시간을 차지한다. 스냅샷 구동 기법은 이런 소프트웨어 초기화 과정을 생략할 수 있으므로 시스템 구동 시간을 상당히 줄일 수 있다. 하지만 Kaminaga가 제안한 스냅샷 기법은 커널의 초기화 시간을 줄이는 데 초점이 맞춰져 있다. 즉 스냅샷 바이너리를 이용한 저장/복구에는 커널 이미지만이 포함된다. 커널 이미지의 스냅샷 바이너리는 IP를 포함한 모든 시스템을 suspend 시킨 후 만들어진다. 그러므로 디바이스의 상태 일관성(consistency)을 유지하기 위해서 스냅샷 구동을 할 때는 모든 디바이스 혹은 IP들을 초기화하고 suspend 모드로 진입시킨 후, 스냅샷 바이너리를 이용하여 커널 재개 과정을 수행한다.

이러한 방법은 스냅샷 구동의 대상을 커널로 한정시켰기 때문에 어플리케이션의 초기화 시간이 긴 경우에는 구동 시간을 개선하기 어렵다. 또한 디바이스의 상태와 커널의 구동 시의 상태를 일관되게 하기 위해서 스냅샷 기반 구동 시 IP를 초기화하고 suspend 모드로 진입시켜야 하기 때문에 IP의 초기화 시간이 긴 경우에도 구동 시간을 개선하기 어렵다.

Baik 등은 스냅샷 구동 기법을 확장하여 스냅샷 바이너리에 일부 디바이스 혹은 IP의 suspend 정보를 포함시켰다[6]. 이는 디바이스 혹은 IP가 suspend 모드를 지원하는 경우 resume을 위한 정보가 메모리에 남아 있게 되고, 커널의 스냅샷 이미지 생성 시 resume 정보도 함께 보존될 수 있기 때문이다. 스냅샷 바이너리에 suspend 정보가 포함된 디바이스나 IP는 스냅샷 구동 시에 초기화된 후 보존된 resume정보를 이용하여 상태 복구를 함으로써 스냅샷 기반 구동을 수행한 후의 상태가 스냅샷을 생성하는 시점과 동일한 상태가 된다. 하지만 모든 장치들이 suspend-resume을 지원하진 않고, 특히 subsystem과 같은 범용 운영체제를 지원하지 않는 장치 시스템에서는 그 내부에 속한 IP들의 suspend-resume이 지원되지 않기 때문에 본 기법은 바로 적용될 수 없다.

Jo 등은 스냅샷 구동 기법을 디지털 TV 플랫폼에 적용하여 구동 시간을 줄인 연구를 수행했다[7]. 디지털 TV의 구동 과정에는 어플리케이션의 초기화 시간이 50% 정도를 차지하기 때문에 기본적인 스냅샷 구동 기법을 적용하여 커널의 초기화 시간을 줄이는 것으로는 큰 효과를 볼 수 없다. 그러므로 Jo 등은 스냅샷을 이용한 시스템 상태의 저장 범위를 어플리케이션 영역까지 확장함으로써, 어플리케이션의 초기화 시간을 줄일 수 있었다. 스냅샷의 영역을 어플리케이션까지 확장할 경우에는 다음과 같은 문제점을 고려해야 한다. 어플리케이션이 순수 소프트웨어가 아닌, IP를 사용하는 경우에는 IP의 상태까지 스냅샷 기법을 이용하여 저장 및 복구를 해야 한다. IP의 상태는 레지스터에 반영되기 때문에 스냅샷 바이너리에 IP의 레지스터 정보까지 포함되어야 한다. 하지만 test and set과 같이 특이한 성질을 가지는 IP의 레지스터들의 경우 스냅샷을 생성하고, 이를 resume 시 복구하는 것이 어려워, resume 후 이런 IP의 상태를 초기화 하는 추가 작업을 수행하였다.

스냅샷 구동의 단점은 스냅샷의 크기가 클 경우, 부팅 시 스냅샷을 메모리에 탑재하는 데 오랜 시간이 소모된다는 점이다. 이를 개선하기 위하여 일부 페이지(page)만 선택하여 스냅샷 바이너리를 만드는 기법도 연구되었다[8]. 이 연구에서는 부팅에 필수적인 페이지만 선별하여 스냅샷 바이너리를 구성하고 나머지 페이지들은 스왑(swap) 영역에 두는 방식을 통해 부팅 시 필요한 스냅샷의 크기를 줄였다. 이를 통해 스냅샷 구동 방법에서 스냅샷을 메모리에 탑재하는 시간을 줄이는 결과를 보였다.

이러한 범용시스템에서의 스냅샷 기반 구동 기법의 서브시스템의 적용은 한계점을 가진다. 우선 서브시스템은 범용 운영 체제가 아닌 비교적 간단한 운영 체제로 운영되기 때문에 운영 체제를 초기화하는 데 걸리는 시간은 구동 시간 중 작은 부분을 차지한다. 반면에 어플리케이션으로 표현되는 서브시스템을 위한 하드웨어와 소프트웨어의 초기화가 대부분의 구동 시간을 차지하고 있다. 그러므로 Kaminaga가 제시한 커널 스냅샷 구동 방식이 아닌, Jo의 기법과 같이 어플리케이션 정보까지 포함하는 어플리케이션 스냅샷 구동 방식을 적용해야 한다. 또한 서브시스템의 소프트웨어는 범용 소프트웨어가 아니기 때문에 각 IP를 담당하는 장치 드라이버가 suspend기능을 제공해주지 않는다. 각 IP의 레지스터들은 메모리 기반 I/O를 통해 상태가 관리되지만, 각양각색의 특성을 가지기 때문에 메모리 영역의 읽기를 통한 스냅샷 생성, 메모리 영역의 쓰기를 통한 스냅샷 복구가 IP의 상태를 저장/복구하는 데 쓰이기 어려운 문제점이 존재한다. DTV 환경에서 어플리케이션 스냅샷 구동 방식

을 구현한 Jo 등의 연구에서 수만개의 레지스터를 저장 및 복구하는 방법에 대해 어려움을 토로하였으며, 상태복구가 불가능한 장치의 경우 초기화를 다시 수행하는 방법을 선택하였다.

본 연구는 이러한 문제점을 개선하기 위해 다양한 레지스터의 특성과 그에 맞춘 스냅샷 생성 및 적용 기법을 제안하는 점에서 기존의 연구와 차별화된다. 또한 본 연구에서 제안하는 레지스터의 특성별로 저장 및 복구하는 기법은 서브시스템을 포함하여 일반적인 IP의 suspend-resume 기능 구현 시에도 이용할 수 있다. 이는 범용적인 suspend- resume 기능을 지원하지 않는 IP의 스냅샷 생성에 이용할 수 있으므로 Baik 등의 연구에서 suspend로 복구되는 IP를 늘려서 구동 시간을 더욱 개선할 수 있다.

3. 스냅샷 구동 기법의 서브시스템 적용

본 연구에서 목표로 하는 서브시스템의 구동 시간 단축은 서브시스템의 구동 시 보이는 특성이 일반 컴퓨터 시스템과 다르지 않다는 점을 착안하여 기존의 스냅샷 기반 빠른 구동 기법을 적용하는 것을 목표로 한다.

3.1 스냅샷 생성

서브시스템에 스냅샷 구동 기법을 적용하기 위해서는 먼저 스냅샷을 생성할 시점을 잘 선택해야 한다. 즉 서브시스템이 초기화가 완료되고 메인 시스템에 인터럽트를 통해 준비 완료를 알리기 직전의 상태에서 서브시스템의 메모리 상태를 복사하고 저장하여 스냅샷을 생성한다.

스냅샷 구동을 위해서는 일반 구동용 바이너리에 서브시스템의 상태를 저장하는 추가 영역이 필요하다. 스냅샷 바이너리에 추가되는 첫 번째 영역은 서브시스템의 소프트웨어에 포함된 데이터 중 초기값이 0으로 설정된 전역 변수 영역이나 힙(heap) 영역은 ZI(zero-initialized) 데이터 영역에 속하기 때문에, 정상 바이너리에서는 이 영역의 위치 및 크기정보만 필요할 뿐 내용은 필요하지 않다. 구동 시 초기화 과정에서 0으로 초기화된 페이지를 할당하면 되기 때문이다. 하지만 스냅샷 기반 구동 방법에서는 스냅샷을 생성하는 시점에 이미 전역 변수 영역이나 힙 영역은 의미 있는 내용을 포함하고 있기 때문에 이러한 영역들 또한 스냅샷에 포함되어야 한다.

서브시스템의 IP의 상태를 설정하기 위한 SFR(special function register)들 또한 스냅샷 바이너리에 추가되어야 한다. IP의 상태는 서브시스템의 정상적인 구동 과정에서는 보조 CPU가 초기화 하는 과정에 SFR영역에 메모리 기반 I/O를 통한 쓰기를 수행하여 IP의 상태를 초기화 하기 때문에, SFR영역이 일반 구동용 바이너리에 있을 필요는 없다. 하지만 스냅샷에 기반한 구동 과정에는 서브시스템의 정상적인 초기화를 생략하기 때문에 SFR영역도 스냅샷에 포함되어 IP의 레지스터 상태를 복구해야 한다. 따라서 스냅샷을 생성하는 시점에 SFR영역의 내용을 보존하여 스냅샷 바이너리에 추가함으로써 스냅샷 기반 구동 시 IP들의 상태를 복원할 수 있게 된다.

하지만 SFR영역에 속한 레지스터들 중에는 단순한 메모리 복사만으로 스냅샷 구동에 적용할 수 없는 종류의 레지스터들이 존재한다. 이러한 레지스터들은 그 특징에 맞게 적절히 제어를 해야 스냅샷 구동이 정상적으로 동작하게 된다. 이러한 레지스터들의 특징과 해결 방법은 3.3절에서 다룬다.

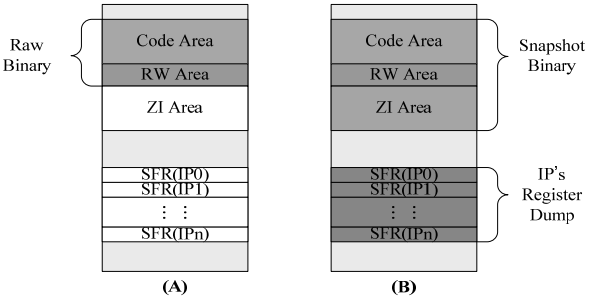

Fig. 4. Comparison of memory copy size, (a) normal startup, (b) snapshot startup

ZI 영역과 SFR 영역의 스냅샷에의 추가는 일반 구동 대비 메인 시스템이 탑재해야 할 바이너리의 크기를 그림 4와 같이 증가시킨다. 이는 메인 시스템이 수행하는 메모리 복사 시간을 길게 만들어 서브시스템의 구동 시간을 오래 걸리게 만드는 문제를 야기할 수 있다. 하지만 이렇게 추가되는 메모리 영역은 초기화로 인해 수행된 영역에 국한되므로 그 크기가 작기 때문에 메모리 복사로 인한 구동 시간 지연이 스냅샷 기반의 구동으로 인한 단축 시간에 비해 미비하다.

3.2 스냅샷 기반 구동

일반적인 스냅샷 기반 구동 방법에서는 스냅샷 기반 구동 시 스냅샷 바이너리를 메모리의 지정된 영역에 탑재시킨 후, 스냅샷 생성시의 CPU의 레지스터 값을 복구하여 스냅샷이 생성된 시점부터 소프트웨어가 재개할 수 있도록 한다. 서브시스템 환경에서 스냅샷 기반 구동이 시작되는 시점은 메인 시스템에서 응용프로그램의 요청이 발생할 경우이다. 이때 메인 시스템은 기 확보된 스냅샷 바이너리를 미리 약속된 영역에 올려놓음으로써 스냅샷 기반 구동이 시작된다.

하지만 서브시스템에서는 일반적인 스냅샷 기반 구동 방법을 적용하기에는 몇 가지 수정 사항이 필요하다. 먼저 서브시스템에 탑재되는 장치 드라이버들은 범용 운영체제를 기준으로 작성되지 않았기 때문에 suspend-resume 기능을

제공하지 않는 경우가 많다. 그 중 SFR을 통해서 관리되는 IP들의 경우에는 앞 절에서 설명한 방법과 3.3절에서 설명될 방법을 통해 해결이 가능하다. 하지만 외부 장치들(예, 이미지 센서)은 SFR영역을 이용해서 스냅샷을 생성할 수 없고, 장치 드라이버 또한 suspend를 제공하지 않기 때문에 특별하게 관리해야 한다.

이를 위해 본 연구에서는 이러한 외부 장치들을 초기화하는 단계를 스냅샷 기반 구동 과정에 추가하였다. 즉 서브시스템의 보조 CPU가 재개된 후 외부 장치들을 초기화 하는 과정을 수행함으로써 스냅샷 기반 구동 과정이 완료되게 된다.

그림 5는 서브시스템에서의 스냅샷 기반 구동 과정을 보여주는 그림이다. 일반적인 스냅샷 기반 구동과 비교하여 Initialize external device 구간이 추가된 것을 알 수 있다. 서브시스템의 일반 구동에서도 하드웨어를 초기화하는 구간(그림 2의 Initialized HW 구간)이 존재한다. 하지만 이 구간은 외부 장치뿐만 아니라 다수의 내부 IP들도 포함하기 때문에 스냅샷 기반의 구동 방법에서 외부 장치를 초기화 하는 시간은 일반 구동에서의 하드웨어 초기화 시간에 비해 짧은 시간을 보이게 된다.

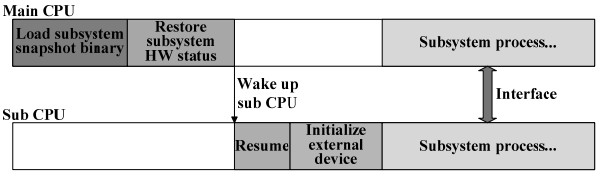

Fig. 5. The startup sequence of a subsystem when using the snapshot startup method

3.3 SFR영역의 스냅샷 생성

3.1절에서 언급한 바와 같이, IP들 내에 존재하는 레지스터들 중 특수한 레지스터들은 그 특징을 고려하여 스냅샷 생성과, 스냅샷 구동을 적용해야 한다. 범용 레지스터와 같이 읽기 쓰기가 일반적으로 동작하는 레지스터들은 스냅샷 생성 시 SFR영역의 내용을 메모리 읽기를 통해 스냅샷 바이너리를 생성하고, 스냅샷 구동 시 바이너리를 다시 지정된 메모리 영역에 복사하여 레지스터의 내용을 바로 복구할 수 있다.

하지만 IP들은 범용 레지스터뿐만 아니라, 각양각색의 특징을 가지는 레지스터들이 존재하고, 이는 전술한 방법으로 스냅샷 기반 구동 기법을 적용할 수 없게 만드는 문제점을 야기한다. Jo의 연구에서 언급되었던 test-and-set 형태의 레지스터뿐만 아니라, 읽기 전용 레지스터, IP 내부 버퍼로 연결된 레지스터, 쓰기 초기화(write-clear) 레지스터, 레지스터 간의 설정 순서 의존성을 가진 레지스터 등이 포함된다. 이러한 레지스터에 해당되는 메모리 영역은 스냅샷 기법을 적용하기 위해서 특별히 처리해야 한다.

첫 번째로 읽기 전용 레지스터는 주로 인터럽트 제어 유닛(ICU) IP에서 사용된다. 즉 펜딩(pending)된 인터럽트가 있을 경우 레지스터의 값이 설정되며, 인터럽트의 처리가 완료되면 레지스터는 자동으로 클리어된다. 읽기 전용 레지스터는 주로 IP의 상태를 반영하기 때문에 레지스터의 값을 백업해 두었다가 복구하는 방식으로 구현할 수 없다. 예컨대 스냅샷을 생성할 시점에 펜딩된 인터럽트가 존재하는 경우, 스냅샷을 이용한 구동 시점에는 펜딩된 인터럽트가 없기 때문에 잘못된 상태를 나타내게 된다. 따라서 이러한 레지스터들이 포함된 상태에서 스냅샷 구동 기법이 적용되기 위해서는 스냅샷을 생성하는 시점을 조절하여 IP의 상태가 스냅샷 구동을 수행할 때의 시점과 동일한 상태가 되도록 해야 한다. 예를 들어 ICU의 경우 펜딩된 인터럽트가 없는 순간에서 스냅샷을 생성하는 방법으로 문제를 해결할 수 있다.

두 번째는 내부 버퍼로 연결이 된 레지스터가 있다. 이러한 레지스터는 들어오는 값을 차례로 IP내부의 버퍼로 복사하게 된다. 즉 레지스터에 쓰여진 값이 동일하게 보이더라도 IP 내부의 상태는 다를 수 있다. 전송 및 수신 기능을 하는 IP가 내부에 존재하는 선입선출 큐의 입구를 하나의 레지스터로 구현하는 경우가 그 예이다. 이 경우도 읽기 전용 레지스터와 마찬가지로 내부 버퍼가 비워 있는 순간에서 스냅샷이 생성되도록 시점을 조절함으로써 스냅샷 구동이 가능하도록 하게 한다.

세 번째는 쓰기 초기화(write clear) 레지스터다. 쓰기 초기화 레지스터는 특정한 값을 쓰면 해당하는 기능이 동작한 후 자동으로 레지스터 값이 기본값으로 초기화 된다. 이러한 특성의 레지스터들은 기본값이 0인 경우가 많기 때문에 주로 0으로 스냅샷에 저장되며, 스냅샷을 이용한 구동 시에 레지스터에 0을 쓰게 된다. 하지만 레지스터에 0을 쓰는 것이 특정한 동작으로 연결되어 있을 가능성이 있기 때문에 쓰기 초기화 레지스터는 복구 시에 어떤 값도 쓰지 말아야 한다. 이를 위해서 SFR의 복구를 메모리 복사로 구현할 경우 쓰기 초기화 레지스터의 앞과 뒤로 레지스터 덤프(dump) 파일을 쪼개야 한다. 그림 6의 경우와 같이 0x24번지에 쓰기 초기화 레지스터가 있으면 이전 영역(0x0~0x23)과 이후 영역(0x28~0x3F)로 나누어서 레지스터 값의 백업 및 복구가 이루어져야 한다.

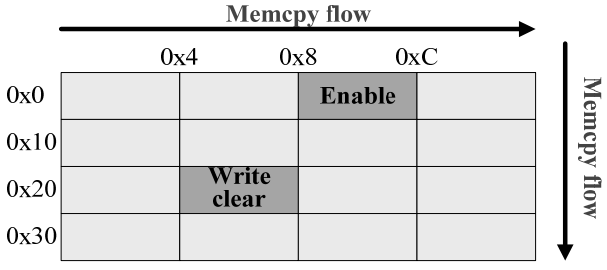

Fig. 6. Special function register organization example

네 번째로 레지스터 간의 설정 순서 의존 특성이 있다. IP의 활성화(enable) 레지스터는 다른 모든 설정이 완료된 후에 마지막으로 켜지는 것을 조건으로 하는 경우가 많다. 활성화 레지스터의 주소가 SFR 영역의 중간에 위치할 경우 메모리 복사로 구현하면 다른 레지스터가 설정되기 이전에 활성화 레지스터가 먼저 설정된다. 그림 6의 경우에는 0xC~0x3F 영역의 레지스터 설정이 끝나기 전에 0x8의 활성화 레지스터가 쓰여지게 되어 IP가 오동작할 수 있다. 이 경우에도 쓰기 초기화 레지스터와 마찬가지로 레지스터 덤프에서 활성화 레지스터를 제외한 후, 모든 값이 설정된 이후에 별도로 활성화 레지스터를 설정해야 한다.

4. 테스트 및 평가

이번 장에서는 제안하는 서브시스템에의 스냅샷 기반 구동 적용 방법을 실험한 결과를 제시한다. 실험은 삼성 Exynos 5250[9]기반 개발 보드에서 이루어졌다. 개발보드에 탑재된 AP의 주 CPU는 1.6GHz ARM Cortex A15 프로세서이며, 이미지 서브시스템은 400MHz ARM Cortex A5를 탑재하고 있다. 서브시스템에서 구동되는 운영체제는 UbiFOS[10]이며, 메인 시스템에서도 실험의 편의성을 위해 UbiFOS를 사용하여 실험하였다. 실제 제품에는 메인 시스템에 리눅스 등 범용 운영 체제가 사용될 것이지만, 본 실험에서 메인 시스템의 역할은 서브시스템의 바이너리를 메모리 내에서 복사해 주는 것뿐이기 때문에 UbiFOS를 사용하여도 범용 운영 체제와 동일한 실험결과가 나올 것이다.

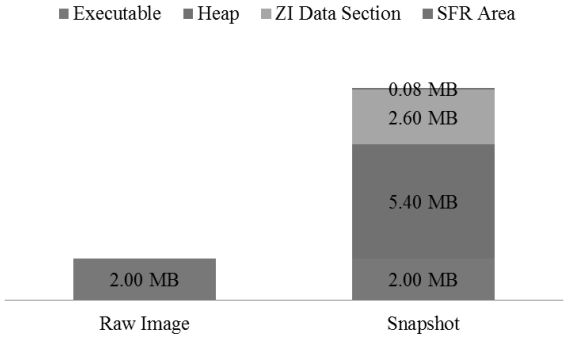

Fig. 7. Comparison of a raw binary and snapshot binary

먼저 그림 7은 서브시스템을 구동하는 데 필요한 스냅샷 바이너리의 크기와 일반 구동에 필요한 일반 바이너리의 크기 비교이다. 일반 바이너리의 크기는 1.98 MB이며 코드 영역이 대부분을 차지하고 있다. 이에 비해 스냅샷 바이너리

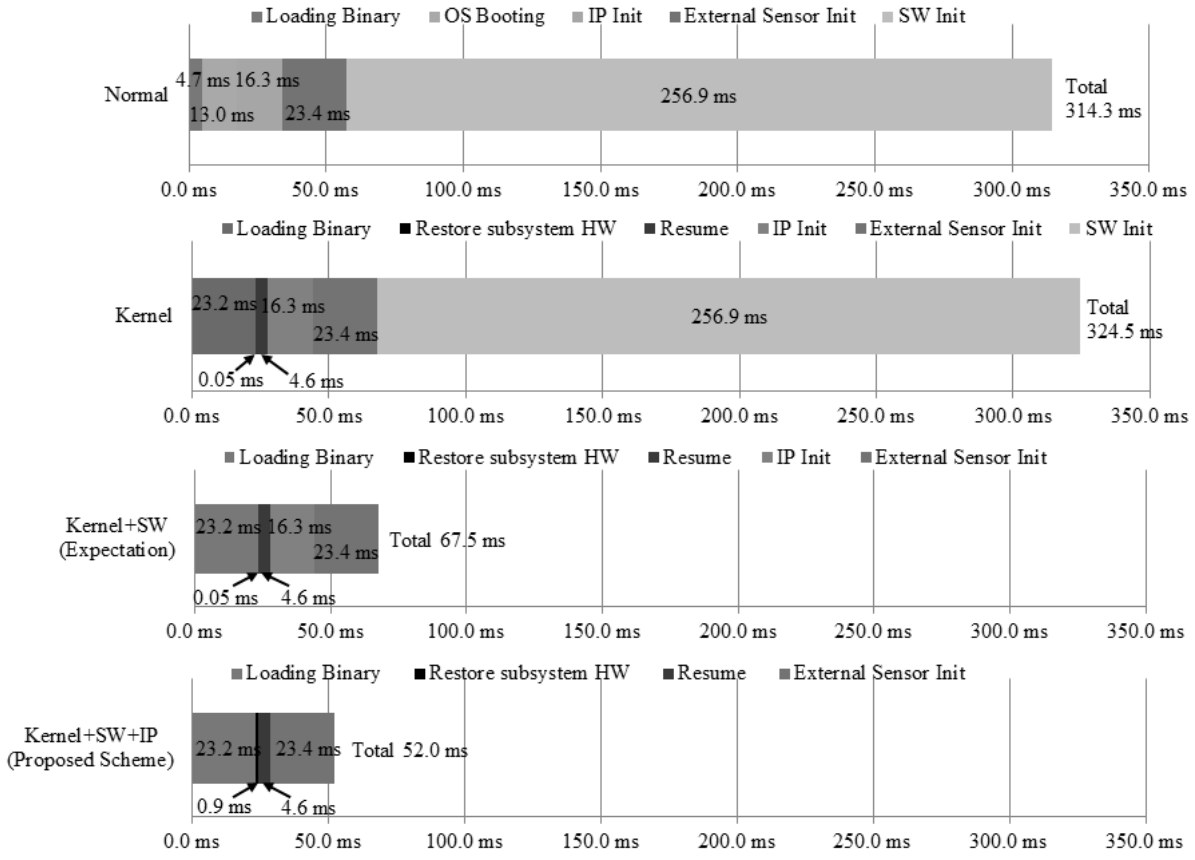

Fig. 8. Image subsystem startup time of normal startup and snapshot-based startup schemes

의 크기는 10 MB이다. 힙 영역이 5.4 MB를 차지하고 있고 나머지 부분은 호스트와 통신을 위한 공유 메모리 영역, 설정 파일을 로딩하는 영역, 그리고 디버깅 메시지를 저장하는 공간 등이다. 또 스냅샷 바이너리 외에 IP의 레지스터를 저장한 영역(SFR Area)의 총 크기가 86 KB이다.

서브시스템 구동을 위한 바이너리 이미지는 일반적으로 호스트 시스템(예: 안드로이드)의 부팅 시점에 서브시스템을 위해 예약된 메모리 공간에 탑재되어 보존되며, 매 서브시스템의 구동 시 서브시스템이 사용하는 메모리에 복제되어 사용된다. 바이너리에는 쓰기 가능한 데이터 영역을 포함하고 있기 때문에 차후 이미지의 재사용이 불가능하기 때문에 원본 보존을 위해 이와 같은 복제 방법을 이용하는 것이다. 또한 스냅샷 바이너리가 2차 저장장치에서 읽힌다고 가정하여도, 타겟 시스템의 eMMC 4.5는 순차 읽기 성능이 약 200MB/s의 수준이며[11], 스냅샷 바이너리를 읽는 데 걸리는 추가 시간은 약 40 ms로, 스냅샷 구동 기법을 통한 구동 시간 단축이 약 262 ms인 점을 감안한다면 스냅샷 구동을 위한 바이너리의 크기 증가는 스냅샷 기반 구동의 이점을 상쇄시키지 않는다.

다음으로 스냅샷 기반 구동으로 인한 구동시간 단축 효과를 보기 위해 일반 구동 기법(Normal)과 제안하는 스냅샷 기반 구동기법(Kernel+SW+IP)의 이미지 서브시스템 구동 시간을 측정하였다. 비교 실험을 위해서 커널만을 대상으로 하는 스냅샷 구동 기법(Kernel)[2], 응용프로그램 소프트웨어를 포함하지만 IP의 다양한 레지스터들의 상태 복구의 어려움으로 인해 IP를 초기화 하는 기법(Kernel+S/W)[7]을 구현하여 함께 성능 측정을 수행하였다. 각 구동 기법의 실험 결과와 세분화된 구동 시간은 그림 8에서 보여준다.[2] 서브시스템의 바이너리 이미지는 예약된 메모리 영역에 탑재되어 서브시스템 구동 시 복사되는 환경에서 실험되었다.

먼저 일반 구동(Normal)과 커널만을 스냅샷 기반 구동을 적용한 방법(Kernel)을 비교해보면, 전자는 314.3 ms, 후자는 324.5 ms로 일반 구동 방법보다 오히려 3% 정도 구동 시간이 늘어났다. 이는 스냅샷 기법을 사용하면서 스냅샷 바이너리를 로딩하기 위해 늘어난 시간이 스냅샷 기법에 의해 개선된 운영 체제 초기화 시간보다 크기 때문이다. 세분화된 구동 시간을 살펴보면 Loading Binary로 표시되는 부분이 바이너리 탑재를 위해 임의의 메모리 영역에서 실제 서브시스템이 사용하는 메모리 영역으로 복사하는 시간이다. 스냅샷 바이너리의 크기는 정상 바이너리 대비 약 8 MB가 크고, 8 MB의 추가 메모리 복사는 18.5 ms의 추가 구동 시간을 야기한다. 또한, 커널이 기본적으로 사용하는 타이머 및 ICU 등의 IP 레지스터를 복구하기 위해 5 KB의 추가적인 메모리 복사가 필요하며 이때 0.05 ms가 소요된다. 서브시스템에 사용되는 운영 체제는 범용 운영 체제가 아니기 때문에 운영 체제가 사용하는 기본적인 IP들도 suspend 기능을 제공하지 않는다. 그러므로 직접 IP의 SFR을 저장 및 복구해야 한다. 커널 스냅샷 구동에서는 보조 CPU를 재개하는데 드는 4.6ms의 시간(Resume)은 일반 부팅 과정에서 보조 CPU가 깨어나서 운영 체제를 초기화 16.3ms의 시간(OS Booting)의 28% 수준을 보인다. 커널 스냅샷 구동에서 운영 체제가 재개된 이후는 일반 구동과 동일하다. 구동 시간의 대부분을 차지하는 소프트웨어 라이브러리 초기화 시간(SW Init)이 그대로이기 때문에 전체적인 구동 시간은 일반 구동 시와 유사하다.

커널과 소프트웨어 라이브러리를 스냅샷에 함께 포함하지만, 하드웨어 IP의 상태는 구동 후 초기화를 수행하는 기법(Kernel+S/W)은 67.5 ms의 구동 시간을 보였으며, 이는 일반 구동 방법의 21% 수준이다. 이는 구동 시간 중 가장 큰 영역을 차지하고 있던 소프트웨어 라이브러리 초기화 시간이 생략되었기 때문이다.

하지만 이는 서브시스템과 같이 하드웨어에 종속된 구조에서는 사용하기 힘들다. 커널 정보와 초기화된 소프트웨어 라이브러리 정보를 스냅샷에 포함하기 위해서는 구동 과정에서 소프트웨어 라이브러리가 초기화되는 순서를 운영 체제가 초기화된 직후로 바꾸어야 한다. 즉 IP 초기화 및 외부 센서 초기화가 수행되기 전에 소프트웨어 라이브러리 초기화가 수행되어야 한다. 하지만 서브시스템의 경우에는 소프트웨어 라이브러리가 내부적으로 IP의 버전 정보 및 센서의 특성 정보 등 하드웨어 구성 정보를 바탕으로 초기화가 수행되는 경우가 많기 때문에 소프트웨어 라이브러리 초기화를 먼저 수행하기 어렵다.

제안하는 커널과 소프트웨어 및 하드웨어 IP를 모두 스냅샷 구동을 적용하는 기법(Kernel+SW+IP)은 52.0 ms의 구동 시간을 보였으며, 일반 구동 방법의 17% 수준의 매우 단축된 구동 시간을 보여 주는 것을 알 수 있다. 또한 커널과 소프트웨어 라이브러리 스냅샷 구동과 비교하여도 77%의 수준으로 개선되었다. 관련 기법 중 가장 빠른 Kernel+S/W기법과 비교하면, IP의 초기화 시간이 단순히 IP의 SFR을 복사하는 시간으로 바뀌었기 때문에 구동 시간이 개선되었다.

세부적인 사항을 살펴보면 스냅샷 바이너리를 로딩하는 시간(Loaing Binary)과 보조 CPU를 재개하는 시간(Resume)이 커널 스냅샷 구동과 동일하다. 이는 서브시스템의 어플리케이션은 태스크의 형태로 운영되며 운영 체제와 메모리 영역이 분리되어 있지 않기 때문이다. 즉 운영체제만을 복

2) Kernel+S/W 기법은 소프트웨어 초기화 부분과 하드웨어(IP) 초기화 부분 간의 의존성 때문에 소프트웨어만의 스냅샷 적용이 정상 동작하지 않아, 예상되는 서브시스템 구동 시간을 계산하여 그래프를 그렸다.

구하는 것과 어플리케이션을 포함한 시스템 전체를 복구하는 것이 동일한 방식으로 이루어진다. IP의 레지스터를 복구하기 위해 86 KB의 추가적인 메모리 복사가 필요하며 이때 0.9 ms가 소요된다. 운영 체제에서 사용하는 기본적인 IP외에 서브시스템의 모든 IP의 레지스터를 복구해야 하기 때문에 기본적인 스냅샷 구동과 비교하여 복구해야 할 레지스터의 영역이 커졌다. 하지만 이 추가 시간은 스냅샷 기반 구동 기법을 통해 보는 시간 단축에 상쇄되어 전체 구동 시간은 여전히 일반 구동보다 빠른 것을 알 수 있다.

또한 어플리케이션 스냅샷 구동은 구동 과정 중 가장 오랜 시간이 걸리는 소프트웨어 라이브러리 초기화 시간(SW Init)이 필요하지 않고, 하드웨어 초기화 시간도, 외부 센서 초기화(External Sensor Init)만을 필요로 하기 때문에 일반 구동 대비 빠른 성능을 보인 것이다. 라이브러리 초기화 시간이 일반 구동에서 차지하는 비중이 큰 점을 고려하면 서브시스템과 같은 저사양 CPU에서 구동되는 초기화 과정을 회피하는 것은 그 효과가 큰 것을 알 수 있다.

5. 결론 및 향후 연구

본 연구에서는 AP 내부의 서브시스템의 구동 속도를 개선하기 위해 스냅샷 기반의 구동 기법을 적용하는 연구를 수행하였다. 또한 suspend 기능을 제공하지 않는 IP에 스냅샷 기법을 적용하기 위해 레지스터 특성별 저장 및 복구 가이드를 제시하였다. 서브시스템 또한 일반 컴퓨터 시스템과 구동 방법이 유사한 특징으로 인해 스냅샷 기반의 구동 기법은 상당한 구동시간 단축을 보여줬다. 특히 서브시스템이 보다 고도화 되면서 하드웨어와 소프트웨어의 초기화 과정이 많고 길어질수록 스냅샷 기반의 구동 기법의 적용으로 인한 이득은 더욱 커질 것이다.

하지만 스냅샷 기반의 구동을 서브시스템에 적용하기에는 많은 노력이 필요하다. 특히 서브시스템에 탑재되는 운영체제는 범용이 아닌 극히 제한적인 기능만 제공하는 운영체제인 경우가 많기 때문에 스냅샷을 생성하기 위한 범용 루틴이 존재하지 않는다. 따라서 서브시스템 내부의 하드웨어의 특징, 레지스터의 특징에 맞게 스냅샷을 생성하는 추가 작업이 필요하다. 이는 결국 스냅샷 생성 기법의 이식성(portability)을 떨어뜨림으로 인해 다른 하드웨어 플랫폼에의 적용이 쉽지 않게 된다.

이러한 단점을 고려해본다면 서브시스템의 구동 시간을 빠르게 가져오면서, 구동 기법의 적용이 손쉽게 될 수 있는 이식성 높은 구동 시간 단축 기법의 연구가 필요할 것으로 판단된다.

Reference

[1] IDC, More Smartphones Were Shipped in Q1 2013 Than Feature Phones, An Industry First According to IDC, http://www.idc.com/getdoc.jsp?containerId=prUS24085240, 2013.

[2] H. Kaminaga, “Improving Linux Startup Time Using Software Resume,” In *Proceedings of the Linux Symposium*, 2006.

[3] A. A. Jerraya and W. Wolf, Multiprocessor Systems-on-Chips, Morgan Kaufmann, 2004.

[4] Texas Instrument, OMAP4430 Multimedia Device Silicon Revision 2.0, Technical Reference Manual, 2010.

[5] T. R. Bird, “Methods to Improve Bootup Time in Linux, Sony Electronics,” In *Proceedings of the Linux Symposium*, 2004.

[6] K. Baik, S. Kim, S. Woo, and J. Choi, “Boosting up Embedded Linux device: experience on Linux-based Smartphone,” In *Proceedings of the Linux Symposium*, 2010.

[7] H. Jo, H. Kim, H. Roh, and J. Lee, “Improving the Startup Time of Digital TV,” IEEE Transactions on Consumer Electronics, Vol.52, No.2, pp.485-493, May, 2009.

[8] S. Park, J. Song, C. Park, “A Fast Booting Technique using Improved Snapshot Boot in Embedded Linux,” Journal of KIISE: Computing Practices and Letters, Vol.14, No.6, pp.594-598, 2008 (in Korean).

[9] Samsung Exynos 5 Dual (Exynos 5250), http://www.samsung.com/global/business/semiconductor/file/product/Exynos_5_Dual_User_Manaul_Public_REV100-0.pdf, 2012.

[10] H. Ahn, M. Cho, M. Jung, Y. Kim, J. Kim, and C. Lee, “UbiFOS: A Small Real-Time Operating System for Embedded Systems,” ETRI Journal, Vol.29, No.3, June, 2007, pp.259-269.

[11] Samsung eMMC: Managed NAND Flash memory solution supports mobile applications, http://www.samsung.com/global/business/semiconductor/file/media/Samsung_eMMC_2013_Final_HR-0.pdf, 2013.

김 준

e-mail : jun7.kim@samsung.com

2005년 연세대학교 컴퓨터과학과(학사)

2013년~현 재 성균관대학교 반도체디스플레이공학과 석사과정

2005년~현 재 삼성전자 System LSI 사업부 책임연구원

관심분야 : Embedded System & AP Architecture

이 준 원

e-mail : joonwon@skku.edu

1983년 서울대학교 컴퓨터과학과(학사)

1990년 Georgia Institute of Technology (석사)

1991년 Georgia Institute of Technology (박사)

1992년~1998년 KAIST 교수

2009년~현 재 성균관대학교 컴퓨터공학과 교수

관심분야 : Low Power Embedded Software & Virtual Machines & System Software & Virtual Machines

정 진 규

e-mail : jinkyu@skku.edu

2005년 연세대학교 컴퓨터과학과(학사)

2013년 KAIST(박사, 석박사통합)

2014년~현 재 성균관대학교 반도체시스템 공학과 조교수

관심분야 : Real-time System & Operating Systems & Virtualization & Embedded Systems