논문 2008-45SD-3-5

E-ACPI : 임베디드 시스템에서 적극적 전력 관리를 위한 전력관리 인터페이스 구현

(E-ACPI : An Implementation of An Active Power Management Interface for Embedded Systems)

황 영 시*, 정 기 석*

(Young-Si Hwang and Ki-Seok Chung)

요 약

운영체제 수준의 전력 소모 최적화 기법에 대한 관심이 고조되고 있다. 운영체제는 시스템의 실행에 있어 종합적인 관리를 하기 때문에 시스템의 전력 소모에 막대한 영향을 미칠 수 있고, 이에 따라 전력 소모를 고려한 운영체제(Power Aware OS)에 대한 관심이 매우 높아지고 있다. 마이크로소프트, 인텔, 도시바 등의 회사에서 공동으로 제정한 ACPI 표준은 다양한 장치들의 전력관리를 BIOS/Firmware 수준이 아닌 운영체제 수준에서 효과적으로 고급 기법을 적용할 수 있도록 한다는 점에서 매우 효용성이 높은 것으로 평가 받고 있다. 본 논문에서는 데스크탑, 노트북 플랫폼 기반의 ACPI를 임베디드 리눅스 시스템에 구현한 포팅 방법에 대하여 설명한다. 또한 구현 과정 중에 극복되어 왔던 기술적인 측면들에 대하여 고찰한다.

Abstract

The OS has the manager of the overall system operation, and has the exact information of the running system. Power management by the OS may have great impact for the optimization of the power consumption. We implement E-ACPI, an extended ACPI which is designed for an advanced power management of embedded systems. In this paper, we address (i) how we extend the exiting ACPI to E-ACPI, (ii) technical challenges to overcome in implementation, and (iii) how we port our E-ACPI to an embedded linux system in this paper. Experimental results show that our E-ACPI is very useful and effective in practice.

Keywords : Embedded System, Dynamic Power Management, Embedded OS, Low Power Design, Ubiquitous

I. 서 론

21세기 지식 산업의 중심축을 이루는 전자 산업의 최근 경향은 컴퓨팅 기술과 통신기술들의 통합화에 따른 유비쿼터스 환경으로 대표된다. 서비스를 제공하는데 있어서의 가장 큰 걸림돌은 이들 유비쿼터스 제품들이 요구하는 전력 사용량이 크게 증가될 수 있다는 것이다. 예를 들어, 이동 정보 단말기의 가치는 많은 부분이 이동 정보 단말기 상에서 제한된 배터리의 용량 하에 얼마 동안이나 서비스가 제공될 수 있느냐에 달려있다. 특히, 급속히 발전하는 반도체 기술에 비해 배터리 기술은 그 발전 속도가 매우 느리며, 이로 인해 에너지 사용의 효율성을 높이는 저전력 설계 기술은 차세대 유비쿼터스 제품들의 핵심 기술 요소로 인식되고 있다.

저전력 설계에 대한 요구가 높아지면서 기존의 회로, 로직 중심의 저전력 기법들의 개발에서 아키텍쳐, 소프트웨어, 시스템 수준의 기법 개발로 연구의 영역이 확대되고 있는 추세이다. 제품의 시장진입시간을 줄이기 위해서, 대부분의 유비쿼터스 SoC 제품들이 많은 기능들을 소프트웨어로 지원할 것으로 예견되기 때문에 응용 프로그램을 비롯하여 운영체제나 컴파일러, 통신 프

* 정회원, 한양대학교
(Hanyang University)

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2008-C1090-0801-0045)

접수일자: 2007년11월30일, 수정완료일: 2008년2월27일

로토콜 등의 소프트웨어/시스템 수준의 저전력 SoC 설계 기술의 역할이 커지리라 예상된다.[1]

소프트웨어/시스템 수준에서의 저전력 SoC설계와 더불어 구체적인 아키텍쳐 설계에서의 전력 소모를 줄이는 것도 긴요하며, IP 블록 간의 SoC 인터페이스 회로에서의 전력 소모를 줄이는 설계 또한 매우 필수적인 설계 기술이 된다. 이러한 저전력 설계 기술은 전력 측정 도구의 활용 없이는 용이하지 않으며, 신호 잡음, 연결선의 전력 소모 등 물리적 변수를 이용한 전력 소모 측정에 대한 필요성도 중요하다. 이제 저전력 SoC 설계는 선택이 아닌 필수이며, 이는 시스템 설계 단계부터 레이아웃 단계까지 전 설계 과정에서 고려하여야 하며, 멀티미디어 및 휴대기기 등의 고부가 가치의 전자제품 설계에 적용이 된다.

현재 임베디스 시스템에서는 시스템의 목적이나 구조에 따라 다양한 전력 관리 시스템이 적용되고 있다. 이와 같은 다양한 전력 관리 시스템 구조는 이종기기 간의 호환성이나 이식성을 제한하며, 전체 시스템 개발 비용을 증대시킨다. 따라서, 효과적인 전력 관리를 위해서는 시스템 수준에서 임베디드 시스템의 특성을 충족시키며 표준화된 전력 관리 시스템의 표준이 요구된다.

본 논문에서는 시스템 수준의 저전력 설계 기법들에 대해서 고찰한다. 그리고 이들 설계 방식을 기반으로 표준화된 시스템 수준의 전력 관리 표준으로서 널리 알려져 있는 ACPI (Advanced Configuration and Power Interface)를 임베디드 플랫폼에 구현하는 과정 및 적극적 전력 관리를 위한 확장에 대해서 설명한다. ACPI는 전력 관리를 위한 하드웨어와 운영체제 수준의 소프트웨어에 관한 명세를 포함한다. 따라서 본 논문에서는 이를 임베디드 플랫폼에 구현하기 위해 ACPI 표준안에서 요구하는 하드웨어 구조는 FPGA (Field Programmable Gate Array)에 설계하였다. 또한 소프트웨어 구조는 리눅스를 운영체제로 채택하고 ACPI 관련 디바이스 드라이버 코드를 대상 하드웨어 플랫폼에 구현하였다. 그리고 임베디드 시스템 환경을 기반으로 구현된 ACPI의 성능을 평가하였다.

본 논문의 구성은 다음과 같다. Ⅱ장에서는 ACPI 표준안에 관한 분석을 하고, Ⅳ장에서는 임베디드 시스템에서의 ACPI 구현 방안에 대하여 논할 것이다. Ⅴ장은 구현된 임베디드 리눅스 기반의 ACPI 를 바탕으로 전력 관리 기법의 평가를 기술하며, 마지막으로 결론에서 본 연구의 결과를 정리하고, 향후 연구 내용에 대해서 논의한다.

Ⅱ. ACPI

1. ACPI 기본 특성

ACPI 표준 규격은 Intel, Microsoft, Toshiba 등의 회사의 주도로 제정된 운영체제 기반의 전력 관리 기법으로서, BIOS (Basic Input Output System)와 운영체제 사이의 전력 설정과 관리 기능을 정의한다. ACPI는 기존의 컴퓨터 시스템에서 널리 사용되던 전원관리 산업 표준인 APM (Advanced Power Management)을 대체하는 표준이며, 운영체제 수준에서주변장치를 인식하고 전력 관리하는 일반적인 접근 방법을 제시한다.[2]

ACPI의 사양은 ACPI 하드웨어 인터페이스, ACPI 소프트웨어, ACPI 데이터 구조를 정의하고 있다. <그림 1>은 ACPI에서 제시한 소프트웨어와 하드웨어 구성 요소 사이에서 OSPM/ACPI가 어떠한 관계를 가지고 있는지 나타낸다. ACPI는 소프트웨어와 하드웨어에 관한 인터페이스를 정의하여 운영체제에서 하드웨어 플랫폼의 모든 구성요소의 전력을 다양한 전력관리 정책을 선정하여 직접 제어할 수 있는 구조를 제시한다. ACPI는 응용 소프트웨어와 운영체제, 운영체제와 하드웨어 사이의 인터페이스를 모두 지원하며 이와 같은 환경을 바탕으로 다음과 같은 기능을 제공한다.

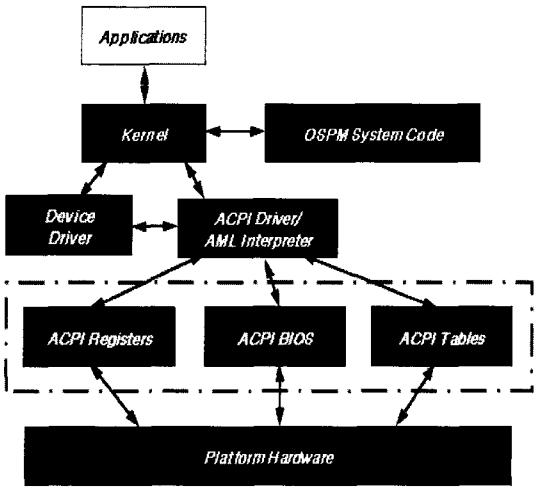


그림 1. ACPI 전체 시스템 구성도
Fig. 1. ACPI Architecture.

2. ACPI 시스템 테이블

ACPI 시스템 테이블은 헤더 정보와 함께 데이터 영역에 하드웨어 인터페이스에 대한 정보를 저장하고 있다. 즉, 레지스터의 주소, ACPI의 초기 설정 값, BIOS와의 인터페이스 관련 정보 등을 내장하고 있다. 테이블의 정보는 시스템 부팅 시에 BIOS로부터 운영체제로

전달되어 운영체제에 내장된 ACPI 드라이버의 구동 데이터로 사용된다. 테이블의 정보에는 AML (ACPI Machine Language) 이 포함되어 있으며, 하드웨어에 대한 제어 방법이 기록되어 있다. 이는 운영체제의 커널에 내장된 ACPI 인터프리터에 의해 해석된다.

3. ACPI 레지스터

ACPI는 시스템 전체 상태 저장과 제어를 위한 하드웨어 인터페이스를 담당하는 레지스터를 포함한 하드웨어 구조를 요구한다. 운영체제는 이러한 ACPI 레지스터를 활용하여 각 하드웨어의 상황 파악 및 제어 동작을 수행할 수 있다.

4. ACPI 하드웨어 모델

ACPI는 하드웨어 구조로 구현된다. 전통적인 하드웨어 제어와 호환되는 구조를 가지고 있어야 한다. 운영체제는 레지스터를 사용하여 운영체제 수준에서 외부의 입력 이벤트 처리를 ACPI 모드나 전통적인 처리 방식 중 하나를 선택해서 시스템을 관리할 수 있다. 그리고 하드웨어의 입력은 크게 3가지로 구분할 수 있는데, 전통적인 이벤트를 처리하는 부분, ACPI와 전통전적인 입력이 공용되는 부분, ACPI 전용 이벤트를 발생시키는 부분이다. 이러한 입력 구성은 처리 및 출력에 대한 구분에도 명확한 기준을 제공하여 전체 시스템 구조를 그룹화 시키는 환경을 조성한다.

5. ACPI 시스템 펌웨어

ACPI 시스템 펌웨어는 IBM PC 구조에서 BIOS를 제어하기 위한 펌웨어로서 주변장치를 제어하는 기존의 기능과 운영체제로부터 ACPI 표준에 의거된 요청을 처리하는 기능을 모두 수용한다. 또한 ACPI 이전의 전력 관리 기법을 허용할 수 있는 레거시(Legacy) 상태의 처리도 요구된다.

III. E-ACPI 구현

1. 환경설정

임베디드 시스템에서 ACPI를 구현하기 위해서 <그림 2>의 구조와 같이XScale PXA255, SDRAM 64MByte, Intel Strata Flash 32MByte, FPGA(Cyclone : 20,000 Gates)로 구성된 운영체제 구동 환경과 FPGA를 중심으로 제어되는 주변 장치를 가지고 있는 하드웨어 플랫폼을 선정하였다. 그리고 ACPI 기능을 탑재한

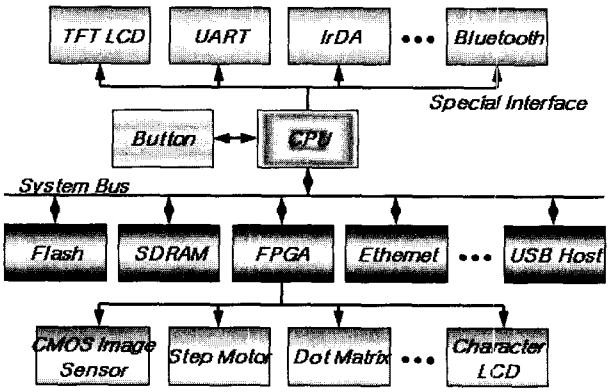


그림 2. 타겟 하드웨어 플랫폼 구조도
Fig. 2. Structure of the Target H/W Platform.

운영체제로는 XScale이 패치된 커널 버전 2.4.19의 리눅스를 사용하였다.

2. ACPI 레지스터 구현

ACPI 레지스터는 하드웨어 형태의 제어 인터페이스이기 때문에 FPGA에서 구현하였다. <표 1>에 나타난 것과 같이 ACPI 레지스터 관련 명세에 따르면 레지스터의 구현 형태는 기능적으로 서로 관련 있는 레지스터를 규합하여 레지스터 블록을 형성하고, 규합된 레지스터는 다시 상위 개념의 레지스터 그룹을 형성하여 관리하는 구조를 가진다. 본 논문에서는 레지스터를 대상 하드웨어 플랫폼에 존재하는 FPGA 내부에 설계하고 메인 프로세서에서 접근할 수 있는 시스템 주소를 할당하였다. 그리고 레지스터의 비트는 각각의 기능에 맞게 ACPI BIOS에서 사용되는 하드웨어의 시스템의 상태를 저장하거나 제어할 수 있도록 설계한다. 하나의 예로서 레지스터 중에서 타이머와 관련된 레지스터의 경우, 그

표 1. ACPI 레지스터 그룹
Table 1. ACPI register group.

Register Groupings	System Address	Register Blocks	Registers
PM1 EVT Grouping	0x14C00000	PM1a EVT BLK	PM1a_STS
			PM1a EN
	Not support	PM1b EVT BLK	PM1b STS
			PM1b_EN
PM1 CNT Grouping	0x14C00004	PM1a_CNT_BLK	PM1a_CNT
	Not support	PM1b_CNT_BLK	PM1b_CNT
PM2 Control Block	Not support	PM2_CNT	PM2_CNT
PM Timer Block	0x14C00008	PM_TMR_BLK	PM_TMR
Processor Block	0x14C0000A	P BLK	P_CNT
	0x14C00010		P LVL2
	0x14C00012		P LVL3
General Purpose Event0 Block	0x14C00020	GPE0 BLK	GPE0 STS
			GPE0 EN
General Purpose Event1 Block	0x14C00024	GPE1_BLK	GPE1_STS
			GPE1_EN

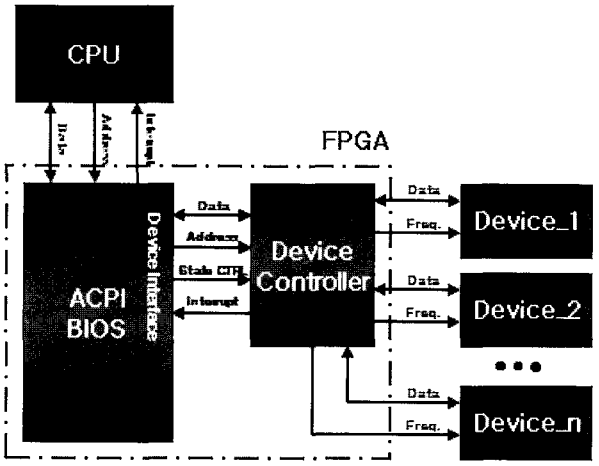


그림 3. ACPI BIOS 구현
Fig. 3. Implementation of the ACPI BIOS.

림 4에서 나타나는 것과 같이 ACPI 타이머 카운터를 중심으로 TRM_VAL 레지스터는 타이머의 수치의 상태를 저장하고 제어하는 역할을 담당하고, PM1x_STS 레지스터의 TMR_STS 비트는 타이머의 상태를 표시한다. 그리고 PM1x_EN 레지스터의 TMR_EN 비트는 타이머의 사용여부를 결정한다. 임베디드 시스템 전체의 전력 관리를 위해서 ACPI 레지스터는 임베디드 시스템에 적합한 초기 설정을 가지고 있어야 한다. 이와 같은 레지스터 설정 작업은 시스템 부팅 시 운영체제에서 ACPI 테이블에 기재된 레지스터 설정 수치를 반영하며 수행한다.

3. ACPI BIOS 구현

ACPI는 IBM PC 구조를 모태로 제정된 표준안이기 때문에 BIOS와 이를 제어하는 Firmware에 대한 인터페이스 및 제어 구조에 관한 정의를 하고 있다. 하지만 임베디드 시스템에서 사용되는 프로세서의 경우 주변장치를 직접 제어가 가능한 마이크로컨트롤러의 구조로 설계되어 있기 때문에 별도의 BIOS를 내장하지 않는다. 또한 ACPI 표준 제정에 Intel 회사가 참여하여 자사의 x86 프로세서에 ACPI에서 요구하는 하드웨어 기능을 내장하여 x86 프로세서를 사용하는 데스크탑이나 노트북의 경우에는 ACPI에 최적화된 프로세서를 사용한다고 볼 수 있다. 반면에 임베디드 시스템에서 주로 사용되는 ARM 프로세서의 경우에는 x86 프로세서와는 다르게 ACPI 관련 하드웨어를 내장하고 있지 않기 때문에 이에 대한 보완책이 마련되어야 한다. 따라서 IBM PC 기반의 ACPI와 임베디드 시스템에서의 ACPI는 BIOS의 사용여부와 메인 프로세서에서의 ACPI 지원여부에서 근본적인 차이를 갖는다.

본 논문에서는 이와 같은 차이를 극복하기 위해서 ACPI에서 BIOS에 관하여 명세된 기본적인 기능과 ACPI에서 요구하는 프로세서의 기능을 탑재한 임베디드 시스템을 위한 ACPI BIOS를 FPGA에 설계하였다. 설계된 ACPI BIOS에는 ACPI에서 사용되는 레지스터를 비롯하여 ACPI 전용 타이머, 인터럽트 컨트롤러, 주변장치 제어 회로 등의 기능을 내장하였다. 그리고 설계된 BIOS는 <그림 11>과 같이 메인 프로세서와 버스 인터페이스로 데이터를 교환하며, 주변장치로부터 인터럽트가 발생했을 경우, 인터럽트 신호를 전달하는 역할을 담당한다. 그리고 ACPI BIOS와 연동하여 주변장치를 제어하는 디바이스 컨트롤러는 ACPI BIOS로부터 장치의 상태 제어 정보를 수신 받아서 주변장치에 대한 주파수 변조를 실행함으로써 간접적인 전력 제어를 수행한다.

ACPI BIOS 구현에 앞서 선결과제가 있다. 이는 하드웨어 플랫폼에 따라 제약되는 하드웨어 및 소프트웨어 인터페이스 코드를 수정하는 것이다. ACPI는 메인 프로세서와 ACPI 관련 하드웨어 사이의 인터페이스를 시스템 I/O, 시스템 메모리, PCI 설정, SMBus (System Management Bus), 내장형 컨트롤러와 같이 다양한 방식으로 지원할 수 있다. 하지만 임베디드 시스템은 일반적인 컴퓨터 구조에 비해 축약된 시스템 구조로 인한 한계 때문에 대부분 시스템 I/O나 시스템 메모리 방식의 인터페이스 방식만을 지원한다. 더욱이 ARM 프로세서는 시스템 I/O 명령어를 지원하지 않기 때문에 인터페이스 방식을 시스템 메모리 방식으로 한정한다.

4. 주변장치 제어 기능 확장

ACPI의 주변장치에 대한 전력 제어는 4단계로 구분되어 실행된다.[3~4] 이는 PC 구조에서 사용되는 주변장치의 종류가 일정한 표준을 따르며, 주변장치에 대한 전력 제어의 형태도 사용하지 않는 장치를 셧-다운 (Shut-Down) 하는 방식만으로도 충분한 효과를 얻을 수 있다. 그러나 임베디드 시스템의 경우, 메인 프로세서와 연계 동작을 취하는 주변장치가 PC의 경우와는 비교할 수 없을 정도로 많으며, 경우에 따라서는 성능 향상을 위하여 DSP (Digital Signal Processor)와 같은 특수 목적 프로세서를 탑재할 수도 있다. 이와 같은 주변장치의 다양성 때문에 단순한 셧-다운 방식의 전력제어는 그다지 큰 효과를 볼 수가 없다. 따라서 본 논문에서는 ACPI의 주변장치에 대한 4단계 상태에 별도의 전

Address Allocation		
Flash Memory	0x02000000	Flash Memory 256Mbit(32Mbyte)
	0x01ffffff 0x00920000	User Space (JFFS2 File System)
	0x0091ffff 0x00900000	ACPI Table (128KByte)
	0x008fffff 0x00200000	Ramdisk Image
	0x001fffff 0x000c0000	Linux Kernel Image
	0x000bffff 0x00000000	Bootloader

그림 4. ACPI를 포함한 대상 시스템의 메모리 맵
Fig. 4. Target system memory map with the ACPI.

력 관리 정책을 할당하여 각 상태와 상황에 맞는 DFS (Dynamic Frequency Scaling)[5]을 적용하여 최적화된 전력 관리 기능을 추가하였다.

5. ACPI 테이블 구현

ACPI는 규격에 기술된 FADT (Fixed ACPI Description Table) DSDT (Differentiated System Description Table)를 비롯한 20여 개의 테이블을 기반으로 전력 제어와 관련된 하드웨어 플랫폼의 전반적인 구성 내용과 동작 방향을 정의하고, 이를 운영체제에 알려주는 기능을 가지고 있다. 따라서 ACPI 설정 테이블을 구현하고 적용하는 과정이 필요하다. IBM PC에서는 메인보드 제작자가 ACPI 테이블을 구성하고 BIOS의 메모리 저장하여, 운영체제에 하드웨어의 정보를 알려주는 방식으로 구현되었다. 본 논문에서는 ACPI 설정 테이블을 구현하고, <그림 4>에서 표현된 것과 같이 플래시 메모리의 정해진 파티션 영역에 저장하고 리눅스 커널에서 테이블에 접근할 수 있도록 시스템을 구축하였다. 이 때, ACPI 테이블과 함께 저장되는 일반 하드웨어 제어를 위한 AML은 ASL (ACPI Source Language)을 컴파일 하여 생성된다.

ACPI 테이블 생성은 <그림 5>에서 구체적으로 확인할 수 있다. 임베디드 시스템의 플래시 메모리에 적용하기 위해서는 위와 같이 ACPI 표준에 명시된 것과 동일한 형태의 테이블 구성이 필요하다.

```
00000000h: 52 53 44 20 50 54 52 20 69 48 55 45 53 6F 43 00 ; RSD PTR iHUESoC.
00000010h: 00 01 90 00 24 00 00 00 00 02 90 00 00 00 00 00 ; ..?$.....?....
00000020h: 4A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; J...............
                                  :
00000100h: 52 53 44 54 44 00 00 00 01 22 48 55 45 53 6F 43 ; RSDTD...."HUESoC
00000110h: 45 53 6F 43 5F 41 44 54 04 06 06 20 45 53 6F 43 ; ESoC_ADT... ESoC
00000120h: 04 06 06 20 00 03 90 00 00 50 90 00 00 90 90 00 ; ... ..?.P?.띥.
00000130h: 00 91 90 00 00 93 90 00 00 94 90 00 00 96 90 00 ; .멪..벡..빓..뼉.
00000140h: 00 98 90 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .뮁.............
                                  :
00000200h: 58 53 44 54 64 00 00 00 01 FC 48 55 45 53 6F 43 ; XSDTd....?UESoC
00000210h: 45 53 6F 43 5F 41 44 54 04 06 06 20 45 53 6F 43 ; ESoC_ADT... ESoC
00000220h: 04 06 06 20 00 03 90 00 00 00 00 00 00 50 90 00 ; ... ..?.....P?
00000230h: 00 00 00 00 00 90 90 00 00 00 00 00 00 91 90 00 ; .....띥......멪.
00000240h: 00 00 00 00 00 93 90 00 00 00 00 00 00 94 90 00 ; .....벡......빓.
00000250h: 00 00 00 00 00 96 90 00 00 00 00 00 00 98 90 00 ; .....뼉......뮁.
                                  :
00000500h: 44 53 44 54 70 2D 00 00 02 1F 48 55 45 53 6F 43 ; DSDTp-....HUESoC
00000510h: 45 53 6F 43 5F 41 44 54 04 06 06 20 45 53 6F 43 ; ESoC_ADT... ESoC
00000520h: 04 06 06 20 44 53 44 54 4C 2D 00 00 01 D9 48 55 ; ... DSDTL-...?U
00000530h: 45 53 6F 43 45 53 6F 43 5F 41 44 54 04 06 06 20 ; ESoCESoC_ADT...
00000540h: 49 4E 54 4C 21 04 06 20 10 12 5F 50 52 5F 5B 83 ; INTL!.. .._PR_[?
00000550h: 0B 43 50 55 31 01 10 04 00 00 06 08 5F 53 30 5F ; .CPU1......._S0_
00000560h: 12 06 04 00 00 00 00 08 5F 53 31 5F 12 06 04 01 ; ........_S1_....
00000570h: 01 00 00 08 5F 53 34 5F 12 08 04 0A 06 0A 06 00 ; ...._S4_........
00000580h: 00 08 5F 53 35 5F 12 08 04 0A 07 0A 07 00 00 14 ; .._S5_..........
00000590h: 4F 04 4D 43 54 48 02 A0 08 95 87 68 87 69 A4 00 ; O.MCTH.?븮h뇱?
000005a0h: 72 87 68 01 60 08 42 55 46 30 11 02 60 08 42 55 ; r뇯.`.BUF0..`.BU
000005b0h: 46 31 11 02 60 70 68 42 55 46 30 70 69 42 55 46 ; F1..`phBUF0piBUF
000005c0h: 31 A2 1B 60 76 60 A0 12 93 83 88 42 55 46 30 60 ; 1?`v`?뱆꺪UF0`
000005d0h: 00 83 88 42 55 46 31 60 00 A1 03 A4 00 A4 01 08 ; .꺪BUF1`.???.
000005e0h: 50 49 43 4D 00 14 0C 5F 50 49 43 01 70 68 50 49 ; PICM..._PIC.phPI
000005f0h: 43 4D 08 4F 53 46 4C 01 10 84 85 02 5F 53 42 5F ; CM.OSFL..꼪._SB_
```

그림 5. ACPI 테이블의 예
Fig. 5. An example of ACPI Table.

6. ACPI 드라이버 포팅

리눅스의 커널 소스 코드 중에서 x86과 관련된 ACPI 소스 코드 내부에는 ACPI를 직접적으로 구동할 수 있는 ACPI 디바이스 드라이버, ACPI 규격에 준하며 AML을 해석하는 인터프리터, 커널 내부에서 ACPI 규격으로 구동되는 OSP과 같은 다양한 기능의 소스 코드가 존재한다. 이와 같은 리눅스 커널 내부에서의 ACPI 관련 코드의 적절한 수정은 ARM 기반의 임베디드 시스템에 ACPI를 효과적으로 이식하기 위해서 반드시 실행되어야 하며 다음과 같은 작업을 필요로 한다.

프로세서 의존적인 명령어를 수정한다. ACPI 관련 리눅스 커널 코드 중, 문맥전환(Context Switching)에 사용되며 메인 프로세서에 의존적인 어셈블리어 명령어들이 있다. 이러한 어셈블리어 명령어를 x86에서 ARM으로 변경하여 구성하는 일을 수행하여 커널의 ACPI 기능을 포함한 문맥전환을 가능하게 한다.

운영체제에 RSDP (Root System Description Pointer)의 주소를 지정한다. IBM PC에서는 BIOS에 ACPI 테이블이 존재하기 때문에 운영체제는 BIOS에 정해진 위치의 메모리를 참조하여 테이블의 설정을 불러오는 형태를 취하고 있다. 이 때 테이블의 시작을 나타내는 RSDP의 주소가 필요하다. 운영체제에는 RSDP의 주소가 고정되어 있다. 본 연구에서는 BIOS를 대체하고 ACPI 테이블을 플래시 메모리의 파티션 중 하나에 저장하였기 때문에 운영체제가 부팅하면서 테이블을 참조할 수 있도록 RSDP의 정보를 운영체제를 수정하였다.

Ⅳ. 실험 및 결과

본 논문에서는 E-ACPI를 평가하기 위해서 전력 제어의 대상이 되는 장치로서 프로세서 코어로 Leon2[13] 적용하고, FFT (Fast Fourier Transform), FIR (Finite Impulse Response), IIR (Infinite Impulse Response) 연산이 가능한 DSP를 FPGA에 탑재하였다. 그리고 운영체제 수준에서 연산 상태를 관찰하며 ACPI 표준에 적합한 제어 동작을 수행하였다.

이 때, 측정되는 DSP에서 소요되는 보편적인 전력의 양은 다음과 같은 식에 의해서 유도될 수 있다.

$$E = \sum_{i=1}^{n} V_{DD}^2 \bullet C_{eff} \bullet f_i \bullet t_i \qquad (1)$$

그리고 E-ACPI의 기본적인 성능을 평가하기 위한 실험이기 때문에 DSP 연산 시 식(1)과 부합되지 않고 이외의 상황에서 발생하는 DSP의 전력 소모에 대해서는 고려하지 않는다. 그리고 다음과 같은 연산 시간 및 연산량에 대한 몇 가지 가정을 추가한다.

T1은 FFT, T2는 FIR, T3은 IIR에 해당하는 DSP의 연산을 수행하는 태스크이다. 그리고 DSP의 전체 구동 시간을 기준으로 판단할 때, T1은 5초, T2는 7.1초, T3는 10초에서 데드라인을 형성하는 것으로 가정한다.

<그림 6>은 전력 관리를 하지 않고 DSP 연산을 수행하는 예로서 DSP에 인가되는 입력 클럭을 최대 50MHz로 설정한다. 이 경우, T1은 데드라인에 훨씬 미치지 못하는 2.5초에 수행을 완료하며, T2는 5초에서 6.3초 사이에 수행을 완료한다. 그리고 T3는 7.1초에서 8.9초 사이에서 수행을 종료한다.

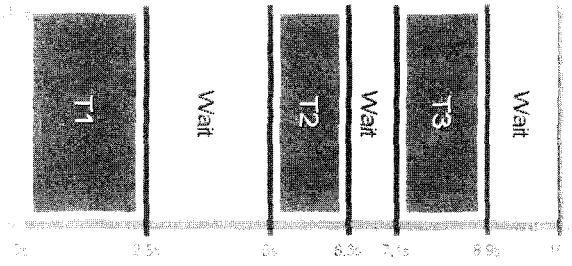


그림 6. 전력관리 정책을 적용하지 않은 결과
Fig. 6. Results after applying no policy.

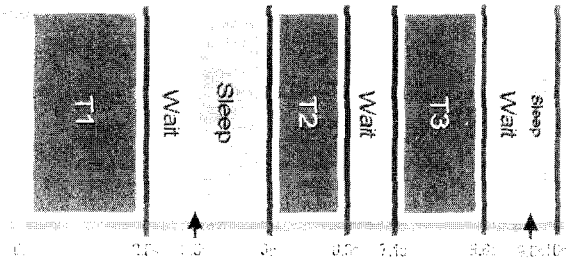


그림 7. 일반적인 ACPI 기법을 적용한 결과
Fig. 7. Results after applying ACPI.

<그림 7>은 DSP 연산 시, 일반적으로 ACPI에서 지원하는 DPM 기능을 적용했을 때의 결과를 나타내는 예이며, DSP가 슬립모드 (Sleep Mode)로 진입하는 것을 결정하는 대기시간은 1초로 가정한다. 따라서 최대 속도로 DSP 연산을 끝낸 이후, 대기시간 (wait time)이 지난 이후에도 DSP 연산이 재개되지 않을 경우 DSP는 슬립모드로 진입하여 전력 소모를 최소화 하는 상태가 된다. T1의 경우 최대속도로 2.5초 지점에서 연산을 끝내고 2.5초에서 3.5초 사이의 대기시간을 유지한 이후 슬립모드로 진입한다. T2는 5초부터 6.3초 사이에 연산을 끝내고 대기시간을 유지하지만 슬립모드 진입 조건인 대기시간 1초 유지 이전에 T3 연산이 시작되기 때문에 슬립모드에 진입하지 못한다. T3는 7.1초에서 9.6초 사이에 연산을 종료하고 1초간의 대기시간이 이후에 슬립모드로 진입한다.

<그림 8>은 DSP 연산 시 E-ACPI의 확장된 기능인

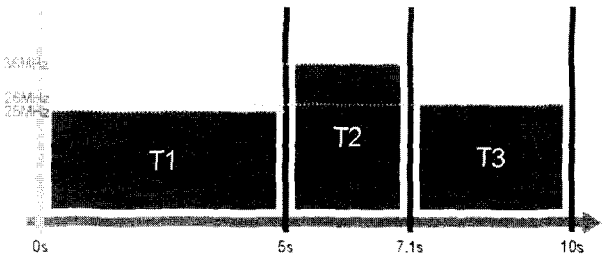


그림 8. E-ACPI 적용한 결과
Fig. 8. Results after applying E-ACPI.

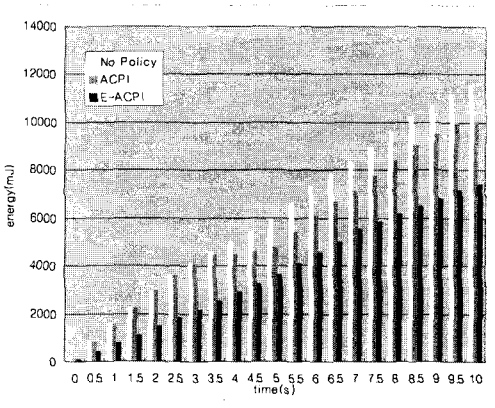


그림 9. 전력관리 정책 정용의 결과
Fig. 9. Please put the title of figure here. Please put the title of figure here.

DVS를 적용할 경우에 대한 예이다. DVS를 적용할 경우, 운영체제 내부의 E-ACPI 처리 루틴은 각 태스크의 데드라인 정보를 수집하고, 데드라인에 맞추어 태스크 실행 시 DSP에 적용되는 주파수를 결정한다. 따라서 T1의 경우 데드라인 5초까지의 연산을 고려하여 DSP에 인가되는 주파수를 25MHz로 설정하고, T2는 5초에서 7.1초 사이의 연산시간인 2.1초에 적합한 36MHz 주파수를 사용하고, T3의 경우 7.1초부터 10초 사이의 연산시간인 2.9초에 적합한 26MHz에 해당하는 주파수를 인가 받고 동작된다.

<그림 9>는 DSP의 연산시 소요되는 전체 누적 전력량과 DVS를 지원하는 E-ACPI 기능을 사용했을 때, DPM을 지원하는 일반 ACPI의 기능을 적용했을 때의 전력 절감 효과를 나타낸다. 일반 ACPI와 E-ACPI의 전력 절감의 특성에 대한 부분은 3.5초에서 5초 구간 사이에서 극명하게 드러난다. DSP에 대해서 DPM 기능을 적용할 경우, 연산 종료 후, 다음 DSP 연산 시간 사이에서 대기시간을 경과하면 DSP는 슬립모드로 진입하여 DSP의 전력 소모량을 최소화하는 특성을 갖는다. 이와 같은 결과는 DPM과 DVS의 특성 차이가 반영된 것으로 볼 수 있다.

V. 결 론

ACPI는 산업 표준인 만큼 시스템 수준에서 전력 관리 방법에 대한 일반적이고 포괄적인 방법을 제시한다. 따라서 시스템 구축에 있어서 전력 제어 방식에 관한 완성도를 높일 수 있고, 이식성도 매우 높다. 본 논문에서는 이와 같은 이점을 가지고 있는 ACPI 표준을 임베디드 시스템에 적용하기 위해서 ACPI 테이블, ACPI 레지스터, ACPI BIOS, ACPI 디바이스 드라이버를 구현하였다. 또한 구현된 ACPI를 임의의 상황에서 주변장치에 대한 주파수 제어 기능을 추가하여 보다 강화된 전력관리 기능을 구현하였다. 하지만 현재까지 구현된 기능은 ACPI의 모든 표준을 수용한 것이 아닌 기본적인 기능만을 적용한 것이기 때문에 앞으로 보다 효율적인 전력관리와 안정적인 시스템을 구축하기 위해서는 ACPI와 관련된 시스템 전체를 구현하고 임베디드 시스템에 보다 최적화하기 위한 연구가 필요할 것이다.

참 고 문 헌

[1] Luca Benini, Alessandro Bogliolo, Giovanni De Micheli"A Survey of Design Techniques for System-Level Dynamic Power Management" IEEE Transactions on VLSI Systems, Vol. 8, no. 3, pp. 299-316, June 2000

[2] Compaq, Intel, Microsoft, Phoenix, and Toshiba. ACPI Specification Version 2.0

[3] Saisanthosh Balakrishnan and Jyothir Ramanan, "Power-Aware Operating System using ACPI"

[4] Fred Douglis, P. Krishnan, and Brian Bershad. "Adaptive disk spin-down policies for mobile computers.", In USENIX Association, editor, Proceedings of the second USENIX Symposium on Mobile and Location-Independent Computing, pp. 121-137, Ann Arbor, Michigin, April 10-11, 1995

[5] J. Lorch and A. Smith "Improving dynamic voltage scaling algorithm with pace.", In ACM SIGMETRICS, pp. 50-61 Cambridge, Massachusetts, USA, June 2001.

[6] Jason Flinn and M. Satyanarayanan. "Energy-aware adaptation for mobile application." In Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP'99), pp. 48-63, Kiawah Island Resort, near Charleston, South Carolina, USA, December 12-15, 1999

[7] Eui-Young Chung, Luca Benini, Alessandro Bogliolo, Yung-Hsiang Lu, and Giovanni De Micheli, Fellow "Dynamic Power Management for Nonstationary Service Requests", IEEE Transactions on Computers, Vol. 51, no. 11, pp. 1345-1361, November 2002

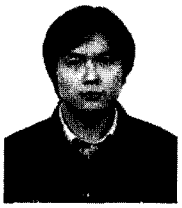
[8] R. Gonzalez and M. Horowitz. "Energy dissipation in general purpose microprocessors" In IEEE Jounral of Solid-State Circuits, Vol. 31, no. 9, pp. 1277-1284, September 1996

[9] Ramon Caceres, Fred Douglis, Kai Li, and Brian Marsh. "Operating system implications of solid-state mobile computers", In Fourth Workshop on Workstation Operating Systems, pp. 21-27, Napa, California USA, October 1993.

[11] URL : http://www.acpi.info

[12] URL : http://www.gaisler.com

저 자 소 개



황 영 시(정회원)

2003년 대진대학교 컴퓨터 공학과 학사 졸업.
2005년 (주)한국정보통신 기술연구소 연구원.
2008년 한양대학교 전자컴퓨터 통신공학과 석사 졸업
현재 한양대학교 정보통신공학 박사 재학.
<주관심분야 : 임베디드 시스템, 저전력 시스템 설계, RTOS>

정 기 석(정회원)

1989년 서울대학교 컴퓨터공학과 학사 졸업.
1998년 University of Illinois at Urbana-Champaign, 강의 전담 교수
2000년 Synopsys, Inc. Sr. R&D Engineer
2001년 Intel Corp. Staff Engineer
2004년 홍익대학교 컴퓨터공학과 조교수
현재 한양대학교 정보통신대학 조교수
<주관심분야 : 임베디드 시스템, System-on-Chip 설계, 저전력 시스템 설계, VLSI/CAD>