DPM 기법을 적용한 저전력 실시간 운영체제 설계 및 구현*

조문행°, 이철훈
충남대학교 컴퓨터공학과
{root4567°, clee}@cnu.ac.kr

The Design and Implementation of Low Power Real-Time Operating System Using Dynamic Power Management

Moon-Haeng Cho° and Cheol-Hoon Lee
Dept. of Computer Engineering, Chungnam National University

요 약

배터리로 동작하는 휴대용 기기와 같은 임베디드 시스템은 복잡한 애플리케이션을 보다 오랜 시간 동안 동작할 수 있도록 하기 위해 하드웨어와 소프트웨어 측면 모두에서 저전력 기법의 구성이 필요하다. 한정된 하드웨어 시스템의 자원을 효율적으로 관리하고 보다 적은 전력 소모를 위해서는 저전력 기법이 탑재된 저전력 실시간 운영체제가 필요하다. 본 논문에서는 IBM과 MontaVista Software에서 제안한 DPM(Dynamic Power Management) 기법을 적용한 저전력 실시간 운영체제를 설계 및 구현하였다.

1. 서 론

근래에는 PDAs, Cellular Phone, MP3 Player 와 같이 휴대용이면서 복잡한 애플리케이션 구동을 위해 보다 높은 CPU 속도를 요구하는 임베디드 시스템의 사용이 증가하고 있다.

시스템 하드웨어의 자원이 높아지면서 시스템에서 소비하는 전력 역시 증가하였으며, 소비전력의 증가는 휴대용기기의 사용시간 단축과 발열량 증가로 인한 기기 오동작 가능성의 증가를 초래하였다. 이런 단점을 극복하기 위해 하드웨어적인 측면에서의 저전력 기법과 소프트웨어적인 측면에서 저전력을 실현하는 기법에 대한 많은 연구가 진행되어 왔다 [2].

본 논문에서는 임베디드 시스템의 한정된 자원을 효율적으로 관리하면서 소프트웨어적인 기법으로 저전력을 실현할 수 있는 저전력 실시간 운영체제(Low Power Real-Time Operating System)를 설계 및 구현하였다.

본 논문의 구성은 2 장에서는 관련연구로서 실시간 운영체제 UbiFOS™과 저전력 실시간 운영체제 구현을 위한 DPM 기법에 대해 소개하고, 3 장에서는 DPM 기법을 적용한 실시간 운영체제의 설계 및 구현 내용을 제시하며, 4 장에서는 테스트 환경 및 결과를 기술한다. 마지막으로, 5 장에서는 결론 및 향후 연구과제에 대해서 기술한다.

2. 관련 연구

2.1 실시간 운영체제 UbiFOS™

연성 실시간 운영체제인 UbiFOS™은 멀티 태스킹 환경을 지원하고 256 단계 우선순위 기반의 선점형 스케줄러를 제공하며, 동일한 우선순위에 대해 FIFO 와 라운드로빈 스케줄링을 제공한다. 또한 태스크에 관련된 태스크 관리기능과 메모리를 동적으로 관리하기 위한 메커니즘을 가지고 있으며, 태스크간 동기화를 위해 세마포와 이벤트 플래그를, 태스크간 통신을 위해서 메시지 큐, 메시지 포트, 메시지 메일박스, 태스크 포트를 가지고 있다 [1][3][5].

2.1.1 태스크 관리 기능

태스크 생성에 필요한 스택을 전역변수를 통해 할당받거나, 동적 메모리 관리를 위한 힙(Heap) 영역에서

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

할당 받아 태스크를 생성할 수 있다. 또한 필요에 따라 태스크의 삭제가 가능하고, 동적으로 태스크의 우선순위를 바꿀 수 있다 [1][3].

2.1.2 동적 메모리 관리

동적 메모리 관리를 위해 가변 크기의 메모리를 할당, 해제할 수 있는 힙 스토리지 매니저(Heap Storage Manager)와 고정 크기의 메모리를 할당 및 해제할 수 있는 메모리 풀(Memory Pool)을 제공한다 [5].

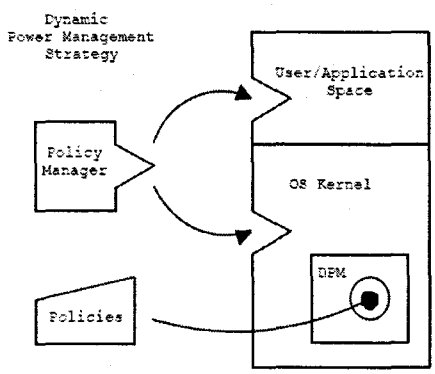
2.1.3 태스크간 동기화

태스크간 동기화를 위해 세마포(Semaphore)와 이벤트 플래그(Event Flag)를 제공한다. 세마포는 바이너리 세마포와 카운팅 세마포가 있으며, 바이너리 세마포의 특수한 경우로, 우선순위 역전현상(Priority Inversion)을 해결하기 위한 우선순위 상속(Priority Inheritance), 삭제 안전장치(Deletion Safety), 재귀(Recursion)를 제공하는 상호 배제 세마포가 있다 [1][3].

2.1.4 태스크간 통신

독립적인 태스크들 사이에 정보를 주고 받기 위해서 메시지 메일박스(Message MailBox)와 메시지 큐(Message Queue), 메시지 포트(Message Port), 태스크 포트(Task Port), 시그널(Signal)을 제공한다 [3].

2.2 DPM 기법



[그림 1] DPM 구조모델

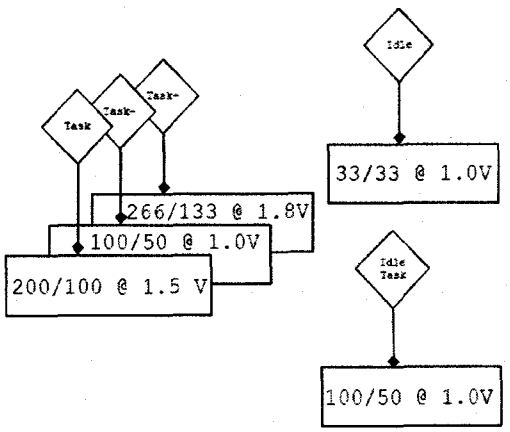
IBM 과 MotaVista Software 에서 제안한 DPM 기법은 소비전력이 가장 큰 CPU 뿐만 아니라 Bus Frequency 를 조절하여 전체 시스템에서의 소비전력을 감소시키는 전력 관리 기법으로, 시스템에 탑재되는 애플리케이션과 각 디바이스에서 요구하는 Bus Frequency 까지 고려하여 소비전력을 감소시키는 융통성을 갖춘 전력 관리 기법이다 [4].

2.2.1 DPM 구조모델

시스템 디자이너는 정책관리자를 통해 시스템에서 제공하는 CPU Frequency 와 Bus Clock 에 맞춰 각 애플리케이션이 요구하는 CPU Frequency 와 Bus Clock 의 set 을 정의한 정책들(Policies)을 생성한다.

[그림 1]은 커널에서 제공하는 DPM 관련 API 를 통해 정책을 결정하고 각 응용프로그램에 미리 정의된 정책을 할당하여 수행하는 DPM 구조모델이다 [4].

2.2.2 DPM 구동정책(Operating Policies)

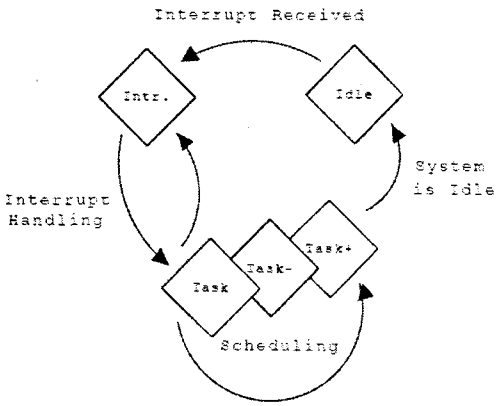


[그림 2] DPM 구동정책

IBM 과 MontaVista Software 에서는 IBM PowerPc 405LP 보드를 사용하였는데, 최대 266MHz 에서 33MHz 까지 지원하는 하드웨어를 통해 5 단계의 구동정책을 제안하였다.

[그림 2]를 보면, Task, Task+, Task-, Idle, IdleTask 의 단계로 구분하여, 각 단계별로 CPU Frequency 와 Bus Clock, 그에 따른 전압 레벨을 집합으로 하는 구동정책을 정의하였다 [4].

2.2.3 DPM 동작상태(Operating States)



[그림 3] DPM 동작 상태

DPM 에서 각 태스크는 시스템의 동작 상태에 따라 태크스에 할당된 정책(Task, Task+, Task- 등)에 의해 정상 동작하는 Run 상태와 인터럽트를 처리하는 인터럽트상태(Intr.), 시스템 유휴상태인 Idle 상태로 천이한다.

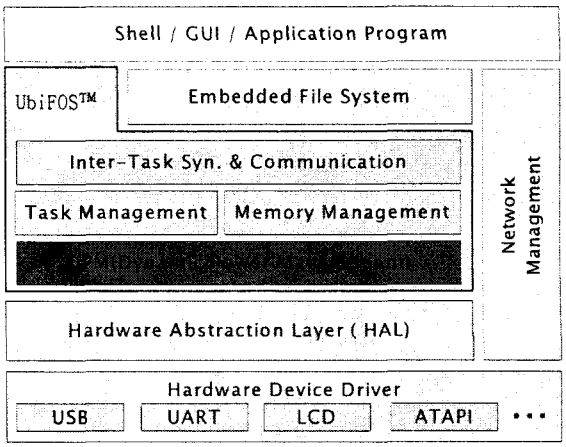
[그림 3]은 DPM 동작상태로, 각 태스크는 자신에게 할당된 정책에 따라 동작하며, DPM 스케줄링 API 에 의해 할당된 정책을 변화시킬 수 도 있다 [4].

3. DPM 기법을 적용한 저전력 실시간 운영체제의 설계 및 구현

3.1 저전력 실시간 운영체제 구현 시 고려사항

본 논문에서 구현한 저전력 실시간 운영체제는 공급 전압을 조정하는 기법인 DVS(Dynamic Voltage Scaling)는 불가능하지만, 공급주파수를 조정하는 기법인 DFS(Dynamic Frequency Scaling)는 가능한 하드웨어 플랫폼(i.MX21 보드)에서 개발하였으며, 기존 UbiFOS™의 커널 수정없이 저전력 구현이 가능한 DPM 기법을 적용하였다.

본 논문의 하드웨어 플랫폼은 초기에 266MHz Frequency 로 동작하며, Frequency 를 Setting 하는 레지스터의 값에 따라 CPU Frequency 를 조절할 수 있어 DFS 가 가능하다. 또한, 시스템의 Bus Clock 도 조절이 가능하다.



[그림 4] DPM 기법을 적용한 저전력 실시간 운영체제

3.2 저전력 실시간 운영체제의 설계 및 구현

DPM 기법을 적용한 저전력 실시간 운영체제는 기존 UbiFOS™에 하드웨어 플랫폼에서 제공하는 Frequency 레벨을 기준으로 구동정책을 구성하였으며, 각 태스크는 실행 중에 DPM 스케줄링 API 에 의해서 구동정책을 동적으로 변경할 수 있다.

[그림 4]는 구현한 저전력 실시간 운영체제의 구성도로 UbiFOS™의 DPM 을 통해 각 태스크에 할당할 CPU Frequency 와 Bus Clock 의 집합인 구동정책을 설정, 변경 및 관리할 수 있으며, 실시간 운영체제가 갖추어야 할 태스크 관리, 메모리 관리, 태스크간 통신과 동기화 기능을 제공한다.

[표 1] DPM 구동정책

구동정책	CPU Frequency	Bus Clock
Task++	266	133
Task+	266	88
Task	266	66
Task-	133	133
Task--	133	66
IdleTask	266	16
Idle	133	16

[표 1]은 본 논문에서 구현한 DPM 구동정책으로 총 7 단계를 지원하며, 애플리케이션에 따라 동적으로 CPU Frequency 와 Bus Clock 의 변경이 가능하게 구성

하였다.

```
/* This Value defines a i.MX21 operating policies for
DPM. */

#define   LP_TASK_PLUS_LEVEL_2    0x00000080 //266, 133
#define   LP_TASK_PLUS_LEVEL_1    0x00000100 //266, 88
#define   LP_TASK_LEVEL           0x00000200 //266, 66
#define   LP_TASK_MINUS_LEVEL_1   0x00000400 //133, 133
#define   LP_TASK_MINUS_LEVEL_2   0x00000800 //133, 66
#define   LP_TASK_IDLETASK_LEVEL  0x00001000 //266, 16
#define   LP_TASK_IDLE_LEVEL      0x00002000 //133, 16


/* This structure defines a i.MX21 Frequency Control
for DPM. */

typedef   struct LP_dpm_regs {
   LP_U32_t cscr;  /* Clock Source Control Register */
   LP_U32_t cscr_mask;
       /* Clock Source Control Register mask */
   LP_U32_t mpctl0; /* MCU PLL Control Register 0 */
   LP_U32_t freq_up_flag; /* Updated Flag */
}dpm_regs;


/* This structure defines a valid i.MX21 operating
points for DPM. */

typedef   struct LP_dpm_md_opt {
   LP_U32_t v;  // voltage (not used - Impossible DVS)
   LP_U32_t cpu_freq;   /* cpu fequency in Hz */
   LP_U32_t bus_freq;   /* bus fequency in Hz */
   dpm_regs regs;          /* Register values */
}dpm_md_opt;

/* CPU Frequency Setting Function */
CLK_CSCR_PRESC()
CLK_CSCR_BCLKDIV()
CLK_CSCR_IPDIV()

/* Bus Clock Setting Function */
CLK_MPCTL0_PD()
CLK_MPCTL0_MFD()
CLK_MPCTL0_MFI()
CLK_MPCTL0_MFN()
```

[그림 5] 저전력 실시간 운영체제의 구조체와 변수

[그림 5]는 저전력 실시간 운영체제 구현을 위해 작성한 구조체와 변수로 각 태스크에 할당할 구동정책은 변수로 정의된 값을 기존의 TCB(Task Control Block) 필드 중 태스크의 상태필드인 t_Status 필드에 추가하는 방식으로 TCB 필드 증가로 인한 메모리 사용량 증가를 최소화 하였으며, dpm_md_opt 는 CPU Frequency 와 Bus Clock 값을 저장하는 구조체이고, dpm_regs 는 구동정책에 따라 레지스터에 설정할 값을 저장하는 구조체이다.

CLK_CSCR_PRESC(),CLK_CSCR_BCLKDIV(),CLK_CSCR_IPDIV()는 CPU 의 Frequency 를 조절하기 위한 CSCR(Clock Source Control Register)에 값을 설정하는 함수이다.

또한, CLK_MPCTL0_PD(),CLK_MPCTL0_MFD(),CLK_MPCTL0_MFI(), CLK_MPCTL0_MFN()는 Bus Clock 값을 조절하기 위한 MPCTL(MPLL Control Register)에 값을 설정하는 함수이다.

[표 2] DPM 관련 API

함 수	설 명
LP_Set_DPM_Policies()	구동정책 설정 함수
LP_Chg_DPM_Policies()	구동정책 변경 함수
LP_MX21_Fscaler()	구동정책에 따른 레지스터 설정함수
LP_Cal_Frequency()	CPU 주파수 계산 함수
LP_Set_Status()	구동정책 추가 함수

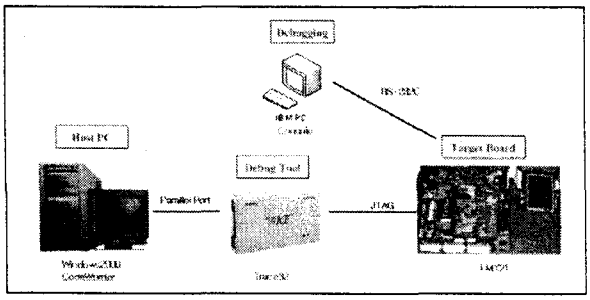
[표 2]는 본 논문에서 구현한 DPM 기법의 저전력 관련 API 로 LP_Set_DPM_Policies()함수는 구동정책을 인자로 CPU Frequency 와 Bus Clock 을 설정하는 함수이고, LP_Chg_DPM_Policies()는 구동정책을 변경하는 함수이며, LP_MX21_Fscaler()는 구동정책에 따라 Frequency 와 Clock 값을 변경하기 위해 관련 레지스터에 값을 변경하는 함수이다.

LP_Cal_Frequency()는 현재 태스크에 할당된 CPU Frequency 를 계산하기 위한 함수이고 LP_Set_Status()는 태스크의 상태필드에 구동정책을 추가하는 함수이다.

4. 테스트 환경 및 결과

본 논문에서 구현한 DPM 기법을 적용한 저전력 실시간 운영체제는 freescale 사에서 제작한 ARM926EJ-S 기반의 i.MX21 보드에 탑재하여 실험하였으며, 컴파일러는 CodeWarrior 를, 에뮬레이터로 Multi-ICE 를 사용하였다.

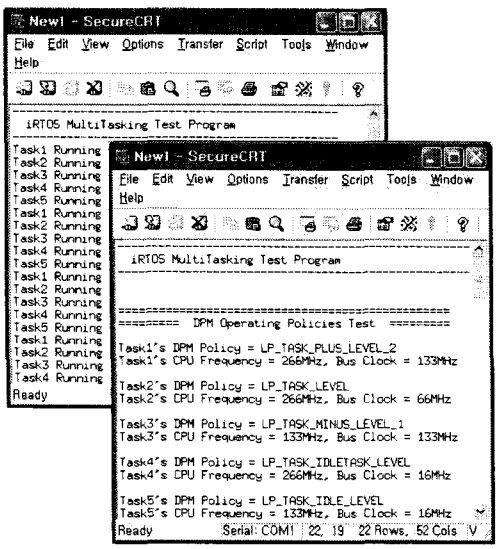
[그림 6]은 저전력 실시간 운영체제를 개발하기 위한 실험환경이다.



[그림 6] 실험환경

본 논문에서 구현한 저전력 실시간 운영체제는 DVS가 불가능한 하드웨어 플랫폼에서 개발하였는데 이는 시스템의 가장 큰 소비 전력을 갖는 CPU 에서의 소비 전력 감소효과가 미미하다는 것을 의미한다. 또한, 소비전력 측정이 용이하지 못한 환경에서 실험하여 실제 DPM 기법의 동적인 CPU Frequency 와 Bus Clock 인가로 인한 소비전력 측정은 수행하지 못하였다.

본 논문에서는 이를 보완하기 위하여 설정한 CPU Frequency 와 Bus Clock 의 동작을 해당 레지스터 값의 변화와 CPU Frequency 값을 계산하는 함수의 값 확인을 통해 실험하였으며, 특정 루틴을 반복적으로 수행하는 예제 프로그램을 구동정책에 따라 CPU Frequency 와 Bus Clock 에 변화를 주어 그 수행속도의 차이를 측정하였다.



[그림 7] 실험결과

본 논문에서 구현한 저전력 실시간 운영체제를 실험하기 위해 5 개의 태스크를 생성하여 스케줄링이 제대로 이루어지는 여부와 각 태스크에 DPM 구동 정책에 따라 할당된 CPU Frequency 와 Bus Clock 으로 동작하는지 여부를 확인하였다.

[그림 7]의 실험결과에서 왼쪽 그림을 통해 스케줄링이 정상동작 한다는 것을 확인할 수 있고, 오른쪽 그림을 통해 DPM 구동정책에 따라 각 태스크들에 할당된 CPU Frequency 와 Bus Clock 으로 동작한다는 것을 확인 할 수 있다

구동 정책	수행 완료 시간(Sec)
LP_TASK_PLUS_LEVEL_2	15.7
LP_TASK_PLUS_LEVEL_1	16.0
LP_TASK_LEVEL	16.0
LP_TASK_MINUS_LEVEL_1	28.3
LP_TASK_MINUS_LEVEL_2	28.8
LP_TASK_IDLETASK_LEVEL	16.7
LP_TASK_IDLE_LEVEL	29.0

[표 3] 구동정책에 따른 수행 속도 차이

[표 3]은 구동정책에 따른 수행 속도를 측정한 자료로 CPU Frequency 변화에 따른 수행 속도 차이는 확인할 수 있었으나, Bus Clock 변화에 따른 수행 속도 차이는 거의 미미하였다. 이는 수행했던 예제 프로그램이 LCD 와 같은 외부 디바이스에 접근하지 않는 프로그램이기 때문일 것이다.

5. 결론 및 향후 연구 과제

본 논문에서는 DPM 기법을 적용한 저전력 실시간 운영체제를 설계 및 구현한 내용을 기술하였다.

구현한 저전력 실시간 운영체제는 기존 실시간 운영체제의 변경없이 저전력 구현이 가능하도록 하였으며, 실시간 운영체제의 특징인 경량성까지 고려하여 설계 및 구현하였다. 하지만 저전력 구현으로 인한 소비전력 감소에 대한 측정 자료가 없다는 한계가 있다.

향후 연구과제는 DVS 가 가능한 타겟 임베디드 플랫폼에 본 논문에서 구현한 저전력 실시간 운영체제를 탑재하여 DPM 구동정책에 따른 소비전력에 대한 자료를 얻는 것과 다른 저전력 기법인 DVS, APM(Advanced Power Management), DPM(Device Power Management)등

의 저전력 실현을 위한 여러 기법을 적용한 저전력 실시간 운영체제에 대해 연구하는 것이다.

참고문헌

[1] Embedded System & RTOS, http://www.aijisystem.com.

[2] MARCUS T.SCHMITZ 외 2 명, "System-Level Design Techniques for Energy-Efficient Embedded Systems"

[3] 박희상, 정명조, 조희남, 이철훈 , "Design of Open-Architecture Real-Time OS Kernel" , 한국정보과학회, 제 31권 제 1호, p.163-165, 2002.04.

[4] "Dynamic Power Management for Embedded Systems" , IBM and MontaVista Software, Version 1.1, November 19, 2002

[5] 조문행, 정명조, 이철훈, "The Design and Implementation for Preventing A memory leakage of Memory Pool on Memory Management of Real-Time Operating Systems" , 한국정보과학회, 제 31 권 제 1 호, p.163-165, 2005.04.

[6] "i.MX21 Applications Processor Reference manual" , Rev 2, 2005. 07.