# 실시간 시스템에서 효율적인 동적 전력 관리를 위한 태스크 스케줄링 알고리듬에 관한 연구

준회원 이 원 규*, 정회원 황 선 영*

# An Improved Task Scheduling Algorithm for Efficient Dynamic Power Management in Real-Time Systems

Won-Gyu Lee* *Associate Member*, Sun-Young Hwang* *Regular Member*

요 약

배터리로 동작하는 휴대용 임베디드 시스템에서 에너지 소모는 중요한 설계 파라미터이며, 동적 전력 관리는 잘 알려진 저전력 설계 기법중의 하나이다. 본 논문에서는 실시간 시스템에서 에너지를 고려한 태스크 스케줄링 알고리듬을 제안한다. 제안한 스케줄링 알고리듬은 시스템에 여유 시간이 존재할 경우 장치 중첩도가 높은 태스크가 우선적으로 수행되도록 스케줄링 하여 장치의 전력 상태 전환 횟수를 줄여준다. 전력 상태 전환 횟수가 줄어들 경우 상태 전환에 따른 전력 소모가 감소하고, 동적 전력 관리의 기회를 더욱 얻을 수 있다. 실험 결과 EDF 알고리듬으로 동작 하는 시스템에서 동적 전력 관리를 한 경우와 비교하였을 때 에너지 소모가 약 23% 감소하였다.

Key Words : DPM, real-time scheduling, low-power systems

ABSTRACT

Energy consumption is an important design parameter for battery-operated embedded systems. Dynamic power management is one of the most well-known low-power design techniques. This paper proposes an online realtime scheduling algorithm, which we call energy-aware realtime scheduling using slack stealing (EARSS). The proposed algorithm gives the highest priority to the task with the largest degree of device overlap when the slack time exists. Scheduling result enables an efficient power management by reducing the number of state transitions. Experimental results show that the proposed algorithm can save the energy by 23% on average compared to the DPM-enabled system scheduled by the EDF algorithm.

## Ⅰ. 서 론

배터리로 동작하는 휴대용 전자기기의 설계에서 배터리 용량의 한계로 인해 시스템에서 사용될 수 있는 에너지는 한정되어 있기 때문에 시스템의 수명을 최대한 연장하기 위해서는 시스템의 에너지 소모를 줄이는 것이 바람직하다. 휴대용 전자기기가 고성능, 고집적화 되어감에 따라 전력 소모량이 지수적으로 증가하는 반면에 배터리의 에너지 용량은 산술적인 증가에 그치고 있어, 배터리 자체 개량보다는 시스템의 전력 소모를 줄이는 저전력 기법이 중점적으로 연구되고 있다[1, 2].

저전력 기법은 크게 정적인 기법과 동적인 기법으로 분류된다[2]. 정적인 기법은 설계 혹은 컴파일

※본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음
* 서강대학교 전자공학과 CAD & Embedded System 연구실 (hwang@sogang.ac.kr)
논문번호 : KICS2005-12-490, 접수일자 : 2005년 12월 8일, 최종논문접수일자 : 2006년 3월 28일

시에 적용하는 것으로 전력 소모가 적도록 하드웨어를 합성하는 방법, 소프트웨어를 컴파일 할 때 코드의 재구성을 통해 전력 소모가 적도록 하는 방법 등이 있다. 동적인 기법은 실행 시간에 적용하는 것으로 작업 부하량의 변동을 이용하여 시스템의 전력 소모를 줄여준다. 정적인 기법을 통해서도 시스템의 에너지 소모를 상당히 줄일 수 있으나 시스템이 실행되는 가운데 계속해서 변하는 수행 환경에 대처할 수 없기 때문에 최근에는 동적인 기법에 더욱 초점이 맞추어지고 있다[3].

동적 전력 관리(Dynamic Power Management: DPM)는 시스템의 실행 시간 중 작업 부하량의 변동을 이용하여 하드웨어의 동작 전압을 동적으로 조절하는 방법이다. 프로세서의 경우 동적으로 전압과 동작 주파수를 변경하여 전력 소모를 줄이는 기법을 사용하는 동적 전압 조절(Dynamic Voltage Scaling: DVS)이 제안되었다. Transmeta의 Crusoe 프로세서[4]와 Intel의 Xscale 프로세서[5]는 DVS를 지원하기 위해 여러 동작 전압을 제공한다. 동적 전력 관리는 하드 디스크나 네트워크 카드 또는 디스플레이 장치와 같은 입출력 장치의 전력 소모를 줄일 수 있다[6]. 입출력 장치는 프로세서에 비해 적은 전력 상태를 제공하며 전력 상태 전환시 긴 시간을 요구한다. 동적 전력 관리는 입출력 장치가 사용되지 않는 idle 상태에 있을 때 저전력 상태인 sleep 상태로 전환하여 전력 소모를 줄일 수 있다.

최근에는 운영체제 수준에서 동적 전력 관리를 하려는 연구가 주목을 받고 있다[7]. 운영체제는 시스템 자원과 실행되는 태스크에 대한 총체적인 정보를 알고 있어 하드웨어 수준에서의 동적 전력 관리보다 사용하기 편리하고 유연하므로, 보다 효율적인 전력 관리에 대한 결정을 통해 더욱 효율적으로 에너지를 감소시킬 수 있다. 전력 관리 기능이 포함된 시스템의 설계 주기를 단축하기 위해 Intel, Microsoft, Toshiba가 함께 전력 관리 기능의 표준화 작업을 하였으며 수행결과로 ACPI(Advanced Configuration and Power Interface)가 제안되었다[8]. ACPI는 전력 관리자와 입출력 장치나 VLSI 칩과 같은 하드웨어 장치 사이의 인터페이스를 유연성 있게 정의한다.

동적 전력 관리 기법에 있어서 중요한 문제의 하나는 마감 시간(deadline)을 지키면서 전력 소모를 최소화하는 것이다. 임베디드 시스템의 경우 대부분 마감 시간 제약 조건이 존재하는 실시간 시스템이며, 동적 전력 관리를 적용할 경우 전력 상태 전환에 따른 시간적 오버헤드가 존재하기 때문에 마감 시간을 어기지 않도록 주의를 하여야 한다.

본 논문의 구성은 다음과 같다. Ⅱ절에서는 기존의 동적 전력 관리에 대한 연구에 대해 설명하고, Ⅲ절에서는 제안한 에너지를 고려한 실시간 스케줄링 알고리듬(EARSS)에 대하여 자세히 설명한다. Ⅳ절에서는 제안된 알고리듬을 EDF 알고리듬과 비교하며, 끝으로 Ⅴ절에서는 결론을 제시한다.

## Ⅱ. 관련 연구

CMOS 회로의 전력 소모는 동적 전력(dynamic power) 소모가 대부분이다. 동적 전력 소모는 공급 전압의 제곱에 비례하기 때문에 공급 전압을 낮추는 것은 전력 소모를 줄이는 데 매우 효과적인 방법이다. DVS 기법은 태스크의 동작 상태를 살펴가면서 프로세서의 클록과 전압을 조절하여 에너지 소모를 줄인다. 실시간 시스템의 경우, 각 태스크의 실제 수행 시간은 최대 수행 시간(Worst Case Execution Time: WCET)보다 작은 경우가 많으며 이로 인해 작업 부하량 변동에 따른 여유 시간(slack time)이 발생한다. 동적 전압 조절 알고리듬은 이 여유 시간을 활용하여 가변 전압 프로세서(variable voltage processor)의 동작 전압과 동작 주파수를 낮추어줌으로써 에너지 소모를 줄인다[9, 10].

프로세서는 여러 동작 전압을 제공하는 반면 입출력 장치는 보통 동작 상태와 sleep 상태의 두 가지 전력 상태를 제공한다. 장치 $k_i$가 sleep 상태에서 동작 상태로의 전환하는 시간을 $t_{wu,i}$, 동작 상태에서 sleep 상태로의 전환하는 시간을 $t_{sd,i}$로 표현한다. 또한 sleep 상태에서 동작 상태로 전환할 때 소모되는 전력을 $P_{wu,i}$, 동작 상태에서 sleep 상태로 전환할 때 소모되는 전력을 $P_{sd,i}$로 표현하며, 동작 상태에서 소모되는 전력을 $P_{a,i}$, sleep 상태에서 소모되는 전력을 $P_{s,i}$로 표현한다. 입출력 장치는 특성상 전력 상태 전환시 시간과 전력이 많이 필요하다. 입출력 장치의 전력 상태를 부적절하게 전환하게 되면 오히려 입출력 장치에서의 에너지 소모가 더 늘어날 수 있다. 만약 입출력 장치 $k_i$의 idle 시간 $t_{idle,i}$가 $t_{wu,i}$와 $t_{sd,i}$의 합보다 작다면 입출력 장치는 sleep 상태에 있지 못하게 된다. 이 경우 입출력 장치의 전력 상태를 전환하지 않는 것이 오히려 에너지를 덜 소비하게 되는데, 전력 상태를 언제 전환할 것인가에 대한 기준으로 break-even

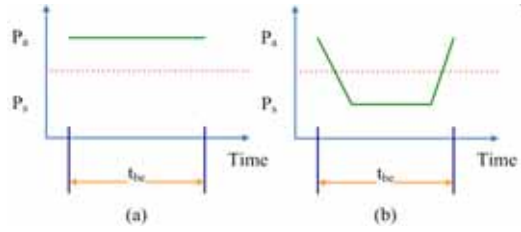


그림 1. breakeven time에 대한 설명. (a)와 (b)에서 에너지 소모가 같게 되는 시간을 breakeven time이라고 한다.

time $t_{be}$가 쓰인다[11]. 그림 1은 breakeven time의 개념을 제시한다. 여기서 $t_{be}$는 식(1)~식(3)으로부터 구할 수 있다.

$$P_{wu,i}t_{wu,i}+P_{sd,i}t_{sd,i}+P_{s,i}t_{s,i} < P_{a,i}t_{a,i} \quad (1)$$

$$t_{idle} > \frac{t_{wu,i}(P_{wu,i}-P_{s,i})+t_{sd,i}(P_{sd,i}-P_{s,i})}{P_{a,i}-P_{s,i}} \quad (2)$$

$$t_{be} = \frac{t_{wu,i}(P_{wu,i}-P_{s,i})+t_{sd,i}(P_{sd,i}-P_{s,i})}{P_{a,i}-P_{s,i}} \quad (3)$$

입출력 장치에 대한 동적 전력 관리 알고리듬의 대부분은 비실시간 시스템에서 입출력 장치의 전압을 조절한다. 입출력 장치에 대한 동적 전력 관리 알고리듬은 크게 timeout 기반 방식, predictive 방식, stochastic 방식으로 분류된다[11-14].

Timeout 기반의 동적 전력 관리[12] 알고리듬은 입출력 장치가 일정 시간동안 idle 상태를 유지하면 장치를 sleep 상태로 전환시키는 방식으로, 장치는 다음 요청이 있기 전까지 sleep 상태에 머무른다. 이 때 timeout은 고정된 값으로 설정하거나 동적으로 변경할 수 있다. 이 알고리듬은 간단하므로 프로세서나 모니터, 하드디스크 등에 적용된다. 이 알고리듬의 단점은 timeout 시간동안 전력을 소비하고 있으므로 에너지를 낭비한다는 점이다. Predictive 동적 전력 관리 알고리듬[11, 13]은 장치의 idle 시간을 예측하여 그 길이가 breakeven time보다 길면 장치의 전원을 차단하는 방식으로, 이 알고리듬은 idle 주기가 시작되면 장치의 전원을 바로 차단하므로 위의 timeout 기반에서와 같은 에너지 낭비가 없다. 이 알고리듬은 idle 시간에 대한 정확한 예측을 필요로 하며, 예측 방식은 오프라인 방식과 온라인 방식으로 나뉜다. 오프라인 방식은 시스템이 실행을 시작하기 전에 과거의 정보로부터 idle 시간을 예측하고, 온라인 방식은 시스템이 실행 중에 idle 시간을 예측을 한다. Srivastava[13]는 과거의 장치 사용 내역을 오프라인으로 분석하여 idle주기를 예측하였다. Hwang과 Wu[11]는 과거의 idle 시간에 대한 기하급수적 가중 평균(exponentially weighted average)을 온라인으로 계산하여 idle 주기를 예측하였다. Stochastic 동적 전력 관리 알고리듬[14]은 시스템을 확률적으로 모델링하여 입출력 장치의 전력 스케줄을 구하는 방식으로, 입출력 장치, 장치 사용 요청, 전력 관리자 등을 Markov chain으로 모델링하여 시스템의 에너지 소모가 최소가 되도록 입출력 장치의 전력 상태를 결정한다.

이들 알고리듬은 비실시간 시스템에서 에너지 소모를 상당히 줄일 수 있으나 마감시간 제약 조건을 보장하지 못하므로 실시간 시스템에서는 사용될 수 없다. 실시간 시스템에서는 마감시간 제약 조건을 고려한 새로운 전력 관리 알고리듬이 필요하다.

Swaminathan[15]은 태스크에 대한 lookahead를 이용한 실시간 동적 전력 관리 알고리듬을 제안하였다. 여기서는 미리 결정된 태스크 스케줄과 입출력 장치 사용 목록을 입력받는다. 실시간 시스템에서는 마감 시간을 만족시키기 위하여, 장치의 전력 상태 전환을 하기 전에 미래에 수행될 태스크의 장치 사용 목록을 살펴보아야 한다. 저자는 미리 살펴봐야 하는 태스크의 최소 개수를 lookahead라 정의하고, 이를 사용하여 스케줄링 시각에 각 장치의 전력 상태를 결정하였다. 이 알고리듬은 시스템의 에너지를 상당히 줄일 수 있었으나, 태스크의 스케줄링과 전력 관리 알고리듬이 독립적이기 때문에 에너지 소모를 줄이는데 한계가 있다. 이 한계를 해결하기 위해 같은 저자는 최대 장치 중첩 (Maximum Device Overlap: MDO)을 이용한 오프라인 태스크 스케줄링 알고리듬을 제시하였다[16]. 장치 중첩은 현재 job에서 사용하는 장치 중 인접한 job이 사용하는 장치의 개수이다. 이 알고리듬은 우선 EDF(Earlist Deadline First)나 RM(Rate Monotonic) 스케줄링 알고리듬으로 가능한 스케줄(feasible schedule)을 구한다. EDF는 마감 시간이 가장 빠른 job을 먼저 수행되도록 스케줄하고, RM은 주기가 가장 작은 job을 먼저 수행되도록 스케줄한다. 제안된 알고리듬은 미리 계산된 스케줄을 장치 중첩이 최대가 되도록 job을 스왑하여 스케줄을 변경한다. 위 알고리듬은 오프라인으로 스케줄을 결정하기 때문에, 시스템 실행 환경이 변하거나 새로운 태스크가 추가되는 경우 시스템이 동작할 수 없으므로 유연성이 떨어진다.

## Ⅲ. 제안된 EARSS 알고리듬

본 절에서는 제안된 에너지를 고려한 실시간 스케줄링 알고리듬(EARSS)에 대하여 자세히 설명한다. 제안된 스케줄링 알고리듬은 시스템에 존재하는 여유 시간을 활용하여 에너지를 줄일 수 있다.

### 3.1 용어 및 문제 정의

시스템에서 스케줄되고 수행되는 기본 단위를 job이라 하고, 특정한 시스템 함수를 구성하는 관련된 job의 집합이 태스크이며, job이 수행 가능하게 된 시각이 release time이고, 실시간 시스템의 특성상 job의 실행이 끝나야 하는 시각을 마감시간이라 정의한다. Job의 release time부터 수행을 끝마치는 시각까지의 시간을 응답 시간, 특정 job에 허용되는 응답 시간의 최대값을 상대 마감 시간(relative deadline)이라 정의한다. $n$개의 태스크로 이루어진 태스크 집합 $T=\{\tau_1, \tau_2, \dots, \tau_n\}$이 존재할 때 $\tau_i \in T$인 각각의 태스크에 대해 주기는 $p_i$, 마감 시간은 $d_i$, 최대 수행 시간은 $c_i$, 태스크가 사용하는 장치 목록을 $L_i$로 표현한다.

모든 태스크가 가지는 주기의 최소 공배수를 hyperperiod라 정의하고, 각 태스크의 상대 마감 시간은 주기와 같다고 가정한다. Job $J_i$는 주기 $p_i$의 시간 중 $c_i$의 시간동안 프로세서를 사용하고 그 비율이 이용도(utilization)가 되며, 이용도는 $u_i = c_i/p_i$가 된다. 시스템 이용도(system utilization)은 시스템에 존재하는 모든 태스크의 이용도를 합친 값이다.

또한 앞서 정의한 태스크를 수행하는 시스템은 $p$개의 입출력 장치 $K=\{k_1, k_2, \dots, k_p\}$를 사용하며, 각각의 장치 $k_i$는 두 전력 상태(sleep 상태와 동작 상태)를 지원한다고 가정한다. 각 장치 $k_i$가 sleep 상태에서 동작 상태로의 전환하는 시간을 $t_{wu,i}$, 동작 상태에서 sleep 상태로의 전환하는 시간을 $t_{sd,i}$로 표현한다. 또한 sleep 상태에서 동작 상태로 전환할 때 소모되는 전력을 $P_{wu,i}$, 동작 상태에서 sleep 상태로 전환할 때 소모되는 전력을 $P_{sd,i}$로 표현하며, 동작 상태에서 소모되는 전력을 $P_{a,i}$, sleep 상태에서 소모되는 전력을 $P_{s,i}$로 표현한다.

입출력 장치는 동작 상태에 있을 때만 job을 수행할 수 있으며, job이 수행을 시작하기 전에 job에 의해 사용되는 모든 입출력 장치는 동작 상태로 되어 있어야 한다. 이 때 입출력 장치 $k_i$에 의해서 소모되는 에너지는 식 (4)와 같다.

$$E_i = P_{a,i} t_{w,i} + P_{s,i} t_{s,i} + M\left(P_{wu,i} t_{wu,i} + P_{sd,i} t_{sd,i}\right) \quad (4)$$

이 때 $M$은 입출력 장치 $k_i$가 전력 상태를 전환한 총 횟수이며, $t_{w,i}$는 $k_i$가 동작 상태로 있던 총 시간, 그리고 $t_{s,i}$는 저전력 상태로 있던 총 시간이다.

본 논문에서는 다음과 같이 문제를 정의한다.

- 입출력 장치 $K=\{k_1, k_2, \dots, k_p\}$의 집합을 사용하는 실시간 태스크 $T=\{\tau_1, \tau_2, \dots, \tau_n\}$의 집합이 존재하고 태스크 $\tau_i$가 소모한 에너지를 $E_i$라고 할 때, 모든 태스크들이 마감 시간을 만족하면서 $\sum_{i=1}^{n} E_i$가 최소가 되도록 하는 태스크의 스케줄을 구한다.

### 3.2 알고리듬의 동기

주기가 $p$인 하나의 태스크가 존재하며, 이 태스크는 하나의 입출력 장치를 사용한다고 가정한다. 태스크의 마감 시간은 주기와 일치한다고 할 때, EDF(Earliest Deadline First) 스케줄링 알고리듬으로 스케줄을 하게 되면 그림 2 (a)와 같이 시스템이 동작하게 된다. 운영체제에서 동적 전력 관리 기능을 지원해준다고 가정하면 그림 2 (b)와 같이 입출력 장치의 전압이 변화하게 된다. 입출력 장치의 경우 전력 상태를 전환(shutdown/wakeup)할 때 동작 상태에서 소모하는 전력보다 더욱 많은 전력이 소모되며 전환 시간도 상당하기 때문에 되도록 전력 상태의 변화가 적게 일어나도록 하는 것이 에너지 소모를 줄이는 데에 효과적이다. 따라서, 새로운 job이 release 되었을 때, 입출력 장치가 idle 상태에 있고 시스템 여유 시간(slack)이 존재한다면, 그 job의 수행을 여유 시간만큼 연기해주는 것이 효과적이다. 반대로 새로운 job이 release 되었을 때, 입출력 장치가 동작 상태에 있다면 전에 수행되던 job이 끝나고 바로 이어서 새로운 job을 수행하는 것이 효과적이다. 이런 아이디어를 적용하여 스케줄링을 하게 되면 그림 3 (a)와 같이 시스템이 동작하게 될 것이며, 그림 3 (b)와 같이 입출력 장치의 전력 상태 변화가 줄어들게 된다.

주기적인 태스크의 주기를 15, 최대 수행 시간(WCET)을 3이라 하자. 태스크가 사용하는 입출력 장치가 전력 상태를 전환할 때 소모하는 전력을 6,

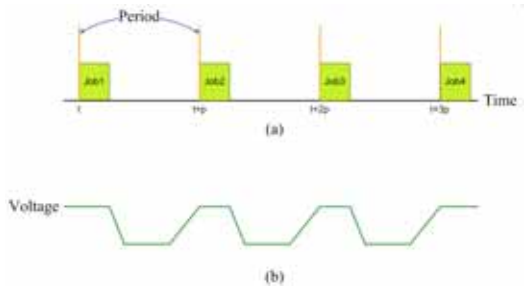


그림 2. EDF 알고리듬으로 스케줄링 된 시스템에 동적 전력 관리를 적용한 경우. (a) 스케줄링 결과. (b) 입출력 장치의 전력 상태 변화

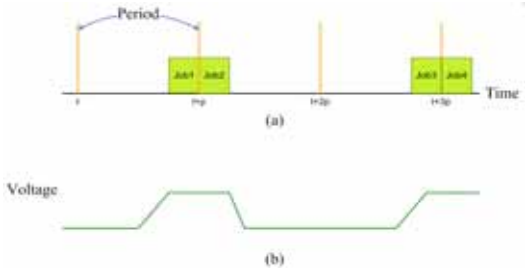


그림 3. 제안된 EARSS 알고리듬으로 스케줄링 된 시스템에 동적 전력 관리를 적용한 경우. (a) 스케줄링 결과. (b) 입출력 장치의 전력 상태 변화

동작 상태에 있을 때 소모하는 전력을 3, idle 상태에서 소모하는 전력은 0이라고 하자. 또한 wakeup 시 걸리는 시간을 3, shutdown시 1.5의 시간이 걸린다고 하자. 이러한 조건으로 EDF 스케줄링 알고리듬을 적용한 그림 2와 같은 시스템에서 3 주기 동안 에너지 소모를 계산해보면 108 unit이 된다. 하지만 제안된 EARSS 스케줄링 알고리듬을 적용한 그림 3과 같은 시스템에서 3 주기 동안 에너지 소모를 계산해보면 64 unit이 된다. 이 예에서 제시한 바와 같이 스케줄링 단계에서 동적 전력 관리를 고려할 때 상당한 에너지 이득을 기대할 수 있다.

### 3.3 여유 시간(slack time) 계산

본 논문에서 시스템의 여유 시간은 시스템의 어떠한 job도 마감 시간을 어기지 않으면서 현재 시스템의 실행을 연기시킬 수 있는 최대 시간으로 정의한다. 여유 시간을 계산할 때 각각의 job이 어느 태스크에 속하는지는 무시한다. Hyperperiod 내에 존재하는 모든 job의 수가 $N$개라고 할 때, 각각의 job을 $J_i$ $(i=1,2,...,N)$라고 표시한다. Job $J_i$의 마감 시간을 $d_i$라 하고, job $J_k$의 마감 시간을 $d_k$라 한다. 임의의 두 job에 대해 $i<k$ 이면, $d_i<d_k$ 이도록 마감 시간 순서대로 job을 나열한다. 여유 시간 계산이 수행되는 시각을 $t_c$로 표시하며, $t_c$의 시각에서 job $J_i$의 여유 시간을 $\sigma_i(t_c)$로 표시한다. 시스템의 여유 시간 $\sigma(t_c)$는 각 job의 $\sigma_i(t_c)$ 중 최소값으로 정의한다.

Hyperperiod가 시작될 때 먼저 hyperperiod내에 있는 모든 job $J_1, J_2, ..., J_N$에 대해 초기 여유 시간 $\sigma_i(0)$을 구해야 하며 그 수식은 식 (5)와 같다.

$$\sigma_i(0) = d_i - \sum_{d_k \le d_i} c_k \tag{5}$$

시스템이 동작하면서 각 job의 여유 시간은 변하게 된다. 실행 중에 여유 시간을 계산하기 위해서는 다음과 같은 과거의 시스템 정보를 저장해야 한다.

- 총 idle 시간 $I$ : hyperperiod의 시작 시점 이후로 발생한 시스템 idle 시간의 총합
- 실행 시간 $\xi_k$ : hyperperiod내의 모든 job $J_k$의 수행된 시간

미리 계산된 초기 여유 시간은 위의 정보를 고려하지 않았기 때문에 이를 반영하여 여유 시간을 재계산 해주어야 한다. 위의 정보를 이용하여 $t_c$에 구한 각 job의 여유 시간은 식 (6)이 된다.

$$\sigma_i(t_c) = \sigma_i(0) - I - \sum_{d_i < d_k} \xi_k \tag{6}$$

시스템의 여유 시간은 각 job의 여유 시간 중 최소값이므로 식 (7)이 된다.

$$\sigma(t_c) = \min(\sigma_i(t_c)) \qquad (i=1,2,...,N) \tag{7}$$

실시간 시스템에서 비주기적인(aperiodic) 태스크에 대한 평균 응답 시간을 줄이기 위하여 여유 시간 스틸링(slack stealing) 기법을 사용한다[17]. 여유 시간 스틸링은 시스템에 여유 시간이 존재할 경우 마감시간이 존재하지 않는 비주기 태스크가 실행되도록 하여 비주기 태스크의 평균 응답 시간을 줄인다. 이 때 여유 시간은 식 (8)과 같이 구한다. $ST$는 비주기 태스크에게 빼앗긴 총 시간이다.

$$\sigma_i(t_c) = \sigma_i(0) - I - ST - \sum_{d_i < d_k} \xi_k \tag{8}$$

본 논문의 알고리듬에서는 비주기 태스크에 의해 여유 시간을 빼앗긴 것이 아니기 때문에 $ST$와 $\sum_{d_i < d_k} \xi_k$는 같은 의미이다. 따라서 여유 시간을 식

(6)과 같이 구한다.

### 3.4 EARSS 알고리듬

EDF 스케줄링 알고리듬은 가장 마감 시간이 빠른 job이 수행되도록 스케줄하며, 대기 큐(ready queue)에 job이 존재하는 한 job의 수행을 연기하지 않는다. 제안된 EARSS 스케줄링의 경우 현재 시스템에서 수행을 끝마친 job이 입출력 장치 $k_i$를 사용하였고 시스템에 여유 시간이 존재 한다면, 마감 시간이 가장 빠른 job을 스케줄하는 대신 대기 큐에서 장치 $k_i$를 사용하는 job을 선택하여 여유 시간동안 실행한다. 반대로 새로운 job이 release되어 스케줄러가 호출되었는데 장치 $k_i$가 idle 상태에 있고 시스템에 여유 시간이 존재한다면, idle 상태를 최대한 유지하기 위하여 job의 수행을 여유 시간만큼 연기한다. 이런 방법으로 입출력 장치의 전력 상태 전환 횟수를 최소화하여 에너지의 소모를 줄인다.

하나의 입출력 장치 관점에서 보면 장치의 현재 상태를 최대한 유지하도록 하여 장치의 전력 상태 전환 횟수를 최소화함으로써 전력 소모를 감소시킬 수 있으나, 시스템의 여러 입출력 장치를 사용하며 수행되는 job의 관점에서 보면 장치 사용 목록이 가장 비슷한 job을 연달아 스케줄 해주어 장치 전체의 전력 상태 전환 횟수를 최소화한다. 따라서 시스템에 여유 시간이 존재할 때 가장 빠른 마감 시간을 가진 job의 수행을 뒤로 미루고 대기 큐에서 장치 중첩(device overlap)이 가장 큰 job을 여유 시간동안 수행한다. 여유 시간이 0이 되면 다시 가장 빠른 마감시간을 가진 job으로 스위칭 해주어 마감시간을 위반하지 않도록 한다.

입출력 장치의 경우 전력 상태 전환 시간 및 전환 시 소모되는 전력의 양이 모두 다르기 때문에 단지 장치 중첩의 개수가 가장 많은 job으로 여유 시간을 빼앗는 방법은 효율성이 떨어진다. 전환 시간과 전환 시 소모되는 전력이 큰 장치일수록 전력 상태 전환이 적게 일어나도록 해주어 더욱 큰 에너지 이득을 기대할 수 있다. 따라서 장치 고유의 전력 소모 특징을 표현하기 위하여 식 (9)와 같이 상태 전환 전력 특성(state transition power characteristic) $\gamma_i$를 정의한다. $P_a$는 동작 상태 전력을 나타낸다.

$$\gamma_i = \frac{P_{wu}t_{wu} + P_{sd}t_{sd}}{P_a(t_{wu} + t_{sd})} \tag{9}$$

시스템에 여유 시간이 존재할 경우 전력 소모를 줄일 수 있는 job을 대기 큐로부터 선택하여 스케줄링을 해주게 되는데 이 때 선택의 기준으로 장치 중첩도(degree of device overlap: $\delta$)를 사용하며 식 (10)과 같이 정의한다.

$$\delta = \sum_{i=1}^{p} \alpha_i \gamma_i \tag{10}$$

$$\alpha_i = \begin{cases} 2 & , \text{양방향 중첩인 경우} \\ 1 & , \text{단방향 중첩인 경우} \\ -2 & , \text{중첩이 없을 경우} \end{cases}$$

그림 4에서 job $J_i$가 실행을 마치는 시점을 스케줄링 시간이라고 하고, 여유 시간 후에 실행될 job을 $J_k$로, 장치 중첩도를 구하고자 하는 job을 $J_{slack}$이라 표현한다. $J_i$와 $J_k$가 입출력 장치 $k_i$를 사용하며 $J_{slack}$이 $k_i$를 사용한다면 장치 $k_i$는 계속 동작 상태를 유지할 수 있다. 반대로 $J_i$와 $J_k$가 입출력 장치 $k_i$를 사용하지 않으며 $J_{slack}$도 $k_i$를 사용하지 않는다면 장치 $k_i$는 계속 sleep 상태를 유지할 수 있다. 이와 같이 여유시간을 빼앗는 job이 전후에 존재하는 job과 같은 장치를 사용한다면 양방향 중첩이라고 표현한다. 입출력 장치 $k_i$가 양방향 중첩이라면 상태 전환이 일어나지 않으므로 중첩도가 가장 높은 경우이며 2의 가중치를 준다. 같은 방식으로 $J_i$와 $J_k$ 중 하나와 장치의 사용이 겹치는 경우 단방향 중첩이라 하며, 상태 전환이 한번 일어나므로 양방향 중첩 가중치의 반에 해당하는 1의 가중치를 준다. 양방향으로 모두 중첩이 없을 경우 -2의 가중치를 준다. 중첩이 없을 경우는 여유 시간을 빼앗는 job에 의해 상태 전환이 두 번 일어나게 되고, 오히려 에너지 소모를 증가시킬 가능성이 있기 때문에 -2의 가중치를 주어 장치 중첩도를 떨어뜨린다.

그림 5는 제안된 EARSS 알고리듬의 pseudo 코드를 보인다. 스케줄러가 호출되면 식 (5)~식 (7)을

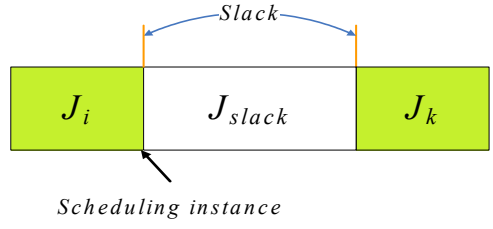


그림 4. 장치 중첩도 계산을 위한 표현

```
Scheduler ()
{
  Compute slack σ(t);
  if (σ(t) < t_be)
    perform EDF_schedule;
  else
    perform EARSS_schedule;
}

EARSS_schedule ()
{
  if (system is in idle state)
     Keep in idle state until σ(t) = 0;
  else
  {
    Find a job with max δ value;
    if (δ < 0)
      perform EDF_schedule;
    else
      Schedule the job with max δ value
         until σ(t) = 0;
}
```

그림 5. Pseudo 코드

이용하여 시스템에 존재하는 여유 시간을 계산한다. 여유 시간이 $t_{be}$보다 적을 경우 마감 시간이 가장 빠른 job이 수행되도록 한다. 여유 시간이 $t_{be}$보다 크면 이를 활용하기 위하여 제안된 EARSS_Schedule을 호출한다. 제안된 EARSS 알고리듬은 스케줄링 시점 전에 수행되던 job이 없고 각 입출력 장치도 sleep 상태에 있었다면 여유 시간동안 새로운 job의 수행을 연기한다. 스케줄링 시점 전에 수행되던 job이 있다면 식 (9) 와 식 (10)을 이용하여 장치 중첩도가 가장 높은 job이 여유 시간동안 수행되도록 한다. 만약 최대 장치 중첩도가 0보다 작다면 여유 시간을 빼앗는 것이 오히려 에너지 소모를 증가시킬 수 있기 때문에 마감 시간이 가장 빠른 job이 수행되도록 한다.

## Ⅳ. 실 험

제안된 EARSS 알고리듬의 에너지 효율성을 평가하기 위하여 C++로 시뮬레이터를 구현하여 사용하였다. 보다 현실적인 시뮬레이션 환경을 구축하기 위하여 입출력 장치는 Fujitsu사의 하드 디스크[18]와 TI사의 DSP[19], 그리고 SST사의 flash[20]를 모델링하였다. 표 1은 각 장치의 동작 상태 전력($P_a$)과 sleep 상태 전력($P_s$), shutdown과 wakeup시의 전력($P_{sd}$와 $P_{wu}$), 그리고 전력 상태 전환 시 소모되는 시간 ($t_{sd}$와 $t_{wu}$)을 보인다. 장치의 전력 소모 특징

표 1. 실험에 쓰인 장치 목록

| | $P_a$ (W) | $P_s$ (W) | $P_{sd}$ (W) | $P_{wu}$ (W) | $t_{sd}$ (sec) | $t_{wu}$ (sec) | $\gamma$ |
|---|---|---|---|---|---|---|---|
| HDD[18] | 0.95 | 0.13 | 0.54 | 1.61 | 0.67 | 2.72 | 1.47 |
| DSP[19] | 0.63 | 0.2 | 0.4 | 0.4 | 0.5 | 0.5 | 0.63 |
| Flash[20] | 0.125 | 0.001 | 0.05 | 0.05 | 0.01 | 0.01 | 0.4 |

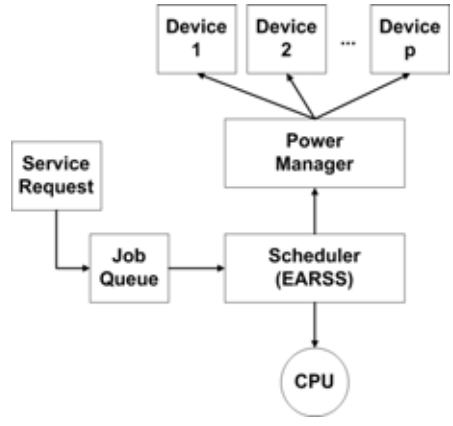


그림 6. 시뮬레이터의 구조

을 표현하기 위해 정의한 상태 전환 전력 특성 $\gamma$도 함께 표시하였다. $\gamma$ 값의 경우 의도한 바와 같이 상태 전환 시 전력과 시간이 많이 소모되는 장치일수록 큰 값을 가지는 것을 확인할 수 있다. 그림 6은 시뮬레이터의 구조를 보인다. 서비스 요청기는 시뮬레이터에서 사용되는 태스크를 생성하고 생성된 job은 시스템의 큐(Job Queue)에 입력된다. 스케줄러는 시에 존재하는 여유 시간을 계산하고 큐에 대기 중인 job에 대해 스케줄링을 수행하며, 스케줄링 결과를 전력 관리자(Power Manager)에게 넘겨준다. 전력 관리자는 스케줄링 결과를 받아 각 입출력 장치의 전력 상태를 조절한다. 전력 관리자는 입출력 장치에게 GO_TO_SLEEP과 GO_TO_ACTIVE 명령을 보낸다. GO_TO_SLEEP 명령은 입출력 장치의 전력 상태를 sleep 상태로 전환하며, GO_TO_ACTIVE는 장치의 전력 상태를 동작 상태로 전환한다.

실험에 사용된 태스크의 집합과 입출력 장치 사용 목록은 무작위로 생성되었다. 각 태스크는 하나 이상의 입출력 장치를 사용한다고 가정하였으며 시스템 이용도(system utilization)는 20%부터 90%까지 변화시키며 알고리듬의 성능을 평가하였다. 100개의 태스크 집합에 대해 시뮬레이션을 수행하였으며, 시뮬레이션 결과의 평균값으로 알고리듬을 평가하였다.

표 2. 실험 결과

| System Utilization (%) | EDF 적용시 에너지 소모 (J) | EARSS 적용시 에너지 소모 (J) | 감소율 (%) |
|---|---|---|---|
| 20 | 160.4 | 103.7 | 35.3 |
| 30 | 166.4 | 119.1 | 28.4 |
| 40 | 172.4 | 133.7 | 22.5 |
| 50 | 182.7 | 142.2 | 22.2 |
| 60 | 184.0 | 145.3 | 21.1 |
| 70 | 194.0 | 150.0 | 22.7 |
| 80 | 195.4 | 154.8 | 20.8 |
| 90 | 201.4 | 165.0 | 18.1 |
| 평균 감소율 | | | 23.6 |

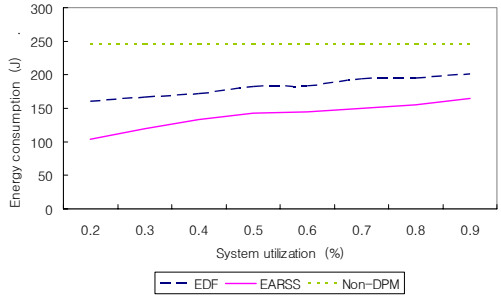


그림 7. EDF와 EARSS의 비교

표 2는 EDF로 스케줄링 된 시스템에서 동적 전력 관리를 수행했을 때 소모되는 평균 에너지와 제안된 EARSS 알고리듬으로 스케줄링 된 시스템에서 동적 전력 관리를 수행했을 때 소모되는 평균 에너지를 보인다. 그림 7은 표 2의 결과를 그래프로 보인다. 표 2를 통해 제안한 알고리듬으로 동작하는 시스템이 EDF로 동작하는 시스템에 비해 약 23% 정도 에너지 소모가 적은 것을 볼 수 있다. 표 2와 그림 7을 통해 시스템의 이용도가 높아짐에 따라 에너지 소모가 커지는 것을 확인할 수 있으며, 시스템의 이용도가 높아짐에 따라 에너지 감소율이 줄어드는 것을 확인할 수 있다. 이는 시스템의 이용도가 높아지면 그만큼 동적 전력 관리를 수행할 수 있는 기회가 줄어드는 것을 의미한다.

## V. 결 론

본 논문에서는 실시간 시스템에서 여유 시간이 존재할 경우 장치 중첩도가 높은 job을 우선적으로 수행하는 태스크 스케줄링 알고리듬을 제안하였다. 제안된 알고리듬은 장치의 전력 상태 전환 횟수를 줄여줌으로써 에너지 소모를 줄인다. 제안한 태스크 스케줄링 알고리듬은 스케줄링 시간에 시스템에 존재하는 여유 시간을 계산하여 시스템이 idle 상태로 있었다면 여유 시간만큼 idle 시간을 유지하고, job이 실행되고 있었다면 대기 큐에서 장치 중첩도가 가장 높은 job을 여유 시간동안 스케줄 해주어 에너지 소모의 감소를 가능하게 한다.

시뮬레이션을 통한 실험 결과, EDF 알고리듬으로 스케줄링 된 시스템에서 동적 전력 관리를 적용한 경우에 비해서 평균적으로 약 23% 에너지 소모가 감소하는 것을 확인할 수 있었다.

## 참 고 문 헌

[1] A. Chandrakasan and R. Brodersen, *Low Power Digital CMOS Design*. Kluwer Academic Pub., 1995.

[2] J. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Kluwer Academic Pub., 1996.

[3] L. Benini and G. De Micheli, *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Pub., 1997.

[4] Crusoe processor. *http://www.transmeta.com/crusoe/*

[5] Xscale Processor. *http://developer.intel.com/design/intelxscale/index.htm*

[6] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Computer-Aided Design*, vol. 8, no. 3, pp. 299-316, June 2000.

[7] Y. Lu, L. Benini, and G. De Micheli, "Power aware operating systems for interactive systems," *IEEE Trans. VLSI Systems*, vol. 10, no. 2, pp. 119-134, April 2002.

[8] Advanced Configuration and Power Interface (ACPI) *http://www.acpi.info*

[9] D. Shin and J. Kim, "Intra-task voltage scheduling on DVS-enabled hard real-time systems," *IEEE Trans. Computer-Aided Design*, vol. 24, no. 10, pp. 1530-1549, Oct. 2005.

[10] J. Seo, T. Kim, and J. Lee, "Optimal intra-task dynamic voltage scaling technique

and its practical extensions," *IEEE Trans. Computer-Aided Design*, vol. 25, no. 12, pp. 1-9, Dec. 2005.

[11] C. Hwang and A. C.-H. Wu, "A predictive system shutdown method for energy saving of event-driven computation," *ACM Trans. Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 226-241, 2000.

[12] R. Golding, P. Bosh, and J. Wilkes, "Idleness is not sloth," in *Proc. Winter USENIX Technical Conf.*, pp. 201-212, 1995.

[13] M. Srivastava, A. Chandrakasan, and R. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Trans. VLSI Systems*, vol. 4, no. 1, pp. 42-55, Mar. 1996.

[14] L. Benini, A. Bogliolo, G. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, vol. 18, no. 6, pp. 813-833, June 1999.

[15] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic I/O device scheduling in hard real-time systems," *IEEE Trans. Computer-Aided Design*, vol. 22, no. 7, pp. 847-858, July 2003.

[16] V. Swaminathan and K. Chakrabarty, "Pruning based, energy-optimal, deterministic I/O device scheduling for hard real-time systems," *ACM Trans. Embedded Computing Systems*, vol. 4, no. 1, pp. 141-167, Feb. 2005.

[17] J. W. S. Liu, *Real-Time Systems*. Prentice-Hall: Englewood Cliffs, 2000.

[18] Fujitsu, "MHL2300AT Hard Disk Drive Product Manual". *http://www.fujitsu.com/downloads/AU/MHL_MHM2_2.5inch_IDE.pdf*

[19] "TMS320C6411 Power Consumption Summary". http://focus.ti.com/lit/an/spra373a/ spra373a.pdf

[20] "SST multi-purpose flash SST39LF020". *http://www.sst.com/downloads/datasheet/ S71150.pdf*

이 원 규 (Won-Gyu Lee) 준회원

2004년 2월 서강대학교 전자공학과 졸업
2006년 2월 서강대학교 전자공학과 석사학위 취득
2006년 3월~현재 삼성전자 근무 중
<관심분야> 저전력 설계, 시스템 수준 설계

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자공학과 졸업
1978년 2월 한국과학원 전기 및 전자공학과 공학석사 취득
1986년 10월 미국 Stanford 대학교 전자공학 박사학위 취득
1976년~1981년 삼성 반도체(주) 연구원, 팀장
1986년~1989년 Stanford대학 Center for Integrated Systems 연구소 책임 연구원 및 Fairchild Semiconductor Palo Alto Research Center 기술자문
1989년~1992년 삼성전자(주) 반도체 기술자문
2002년 4월~2004년 3월 서강대학교 정보통신대학원장
1989년 3월~현재 서강대학교 전자공학과 교수
<관심분야> SoC 설계 및 framework 구성, CAD시스템, Com. Architecture 및 DSP System Design 등