# 정적 시간 분석을 이용한 저전력 태스크내 전압 스케줄링

## (Low-Energy Intra-Task Voltage Scheduling using Static Timing Analysis)

신 동 군 † 김 지 홍 †† 이 성 수 †††
(Dongkun Shin) (JiHong Kim) (Seongsoo Lee)

**요 약** CMOS 회로의 전력 소모는 공급 전압의 제곱에 비례하기 때문에 공급 전압을 낮추는 것이 전력 소모를 줄이는 데 매우 효과적이다. 본 논문에서는 저전력 경성 실시간 응용프로그램을 위한 태스크내 전압 스케줄링 알고리즘을 제안한다. 정적 시간 분석 기법을 바탕으로 제안된 이 알고리즘은 각각의 태스크 내부에서 프로세서의 공급 전압을 조정한다. 제안된 알고리즘에 의해 전압 스케줄링된 프로그램은 모든 유휴 시간을 완전히 이용함으로써 항상 프로그램의 수행을 마감 시간에 근접하여 끝나도록 하여 많은 전력 감소 효과를 얻을 수 있다. 제안된 알고리즘의 효과를 검증하기 위해 일반적인 프로그램을 동적 전압을 사용하는 같은 기능의 프로그램으로 자동으로 변환하는 소프트웨어 도구도 개발되었다. 실험 결과, 자동화 소프트웨어 도구에 의해 변환된 MPEG-4 부호기와 복호기의 저전력 버전이 전원 차단 기능을 가진 고정 전압 시스템에서 실행된 원래 프로그램에 비하여 전력 소모가 7~25%에 불과함을 알 수 있었다.

***Abstract*** Since energy consumption of CMOS circuits has a quadratic dependency on the supply voltage, lowering the supply voltage is the most effective way of reducing energy consumption. We propose an intra-task voltage scheduling algorithm for low-energy hard real-time applications. Based on a static timing analysis technique, the proposed algorithm controls the supply voltage within an individual task boundary. By fully exploiting all the slack times, a scheduled program by the proposed algorithm always complete its execution near the deadline, thus achieving a high energy reduction ratio. In order to validate the effectiveness of the proposed algorithm, we built a software tool that automatically converts a DVS-unaware program into an equivalent low-energy program. Experimental results show that the low-energy version of an MPEG-4 encoder/decoder (converted by the software tool) consumes less than 7~25% of the original program running on a fixed-voltage system with a power-down mode.

### 1. 서 론

정보통신 기술이 발달함에 따라 휴대전화나 개인용 휴대 단말기(PDA), 그리고 동영상 휴대전화와 같은 이동용 시스템(mobile system)의 중요성은 날로 증가하고 있다. 한 번 배터리를 충전했을 때 얼마나 오랫동안 사용할 수 있는지를 나타내는 연속 동작 가능 시간은 상업적인 이동용 시스템에 있어서 가장 중요한 성능 척도의 하나이며, 이동용 시스템의 전력 소모를 줄이는 것은 VLSI 시스템 설계에서 중요한 요소로 부각되고 있다. 특히, VLSI 시스템이 고성능, 고집적화 되어감에 따라 전력 소모는 지수적으로 증가하는 반면에 배터리의 전력 용량은 산술적인 증가에 그치고 있어서, 배터리 자체 개량보다는 VLSI 시스템의 전력 소모를 줄이는 저전력 기법이 중점적으로 연구되는 추세이다. 고성능 마이크로프로세서와 같은 비이동용 시스템에 있어서도 전력 소모는 중요한 설계 목표의 하나가 된다. 높은 전력 소모

· 본 논문은 정보통신부에서 지원하는 대학기초연구지원사업으로 수행되었음.
† 학생회원 : 서울대학교 전기컴퓨터공학부
sdk@davinci.snu.ac.kr
†† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
jihong@davinci.snu.ac.kr
††† 비 회 원 : 이화여자대학교 정보통신학과 교수
sslee1@mm.ewha.ac.kr
논문접수 : 2000년 12월 1일
심사완료 : 2001년 8월 7일

는 VLSI 내부에서 많은 열을 발생시켜 반도체의 성능 저하나 기능 이상, 심지어는 고장을 일으키기도 하기 때문이다. 이러한 문제점들 때문에 최근 들어 소프트웨어 및 하드웨어 측면에서 다양한 저전력 기법이 요구되고 있다.

일반적인 VLSI 시스템의 전력 소모는 CMOS 회로의 동적 전력 소모(dynamic power)가 대부분이며, 이때의 전력 소모 $E$는 $E \propto C_L \cdot N_{cycle} \cdot V_{DD}^2$[1]로 주어진다. 여기서 $C_L$은 CMOS 회로의 부하 커패시턴스(load capacitance), $N_{cycle}$은 프로그램이 실행된 사이클 수, 그리고 $V_{DD}$는 공급 전압(supply voltage)을 의미한다. 전력 소모 $E$가 $V_{DD}$의 제곱에 비례하기 때문에 공급 전압 $V_{DD}$를 낮추는 것은 전력 소모를 줄이는 데 매우 효과적인 방법이다. 그러나 CMOS 회로의 지연시간(propagation delay) $T_D$가 $T_D \propto V_{DD}/(V_{DD}-V_T)^{\alpha}$[2]로 주어지기 때문에, 공급 전압을 감소시키면 회로내의 지연시간이 증가하여 클록 속도를 낮추어야 회로가 안정적으로 동작하게 된다. 여기서 $V_T$는 임계 전압, $\alpha$는 속도 포화 계수(velocity saturation index)를 나타낸다.

실시간 시스템(real-time system)의 경우, 시스템의 최대 클록 속도는 모든 태스크를 주어진 마감 시간(deadline)이내에 끝낼 수 있는 클록 속도보다 크거나 같아야 실시간 동작을 보장할 수 있다. 이때, 각 태스크의 동작 상태를 살펴가면서 마감 시간 제약 조건을 만족하는 가장 낮은 클록 속도까지 시스템의 클록 속도를 조절해가면서 낮출 수 있으며, 클록 속도에 따른 공급 전압도 함께 낮아져서 전력 소모를 크게 줄일 수 있다. 이것이 동적 전압 조정(dynamic voltage scaling: DVS) 기법의 기본 개념이다. 예를 들어 그림 1(a)와 같이 어떤 태스크가 50Mhz의 클록 속도와 5.0V의 공급전압을 가진 프로세서 상에서 실행된다고 가정하자. 만약 이 태스크가 실행되는데 $5\times10^5$ 사이클이 걸리고 마감 시간 조건이 25msec이라면, 프로세서는 주어진 태스크를 10msec만에 끝낼 수 있고 태스크의 마감 시간까지 15msec의 유휴 시간(idle time)을 가지게 된다. 그러나 그림 1(b)에서와 같이 클록 속도와 공급 전압이 20Mhz와 2.0V로 낮아지면 유휴 시간 없이 주어진 태스크를 마감 시간에 맞추어서 끝낼 수 있고 전력 소모는 $2.0^2/5.0^2$ = 16%로 낮아지게 된다.

동적 전압 조정 기법에 있어서 중요한 문제의 하나는 마감 시간 제약 조건을 지키면서 전력 소모를 최소화하기 위한 클록 속도와 이에 따른 공급 전압을 어떤 방법으로 결정하느냐 하는 것이다. 특히 경성 실시간 시스템(hard real-time system)에서는 마감 시간 제약 조건이

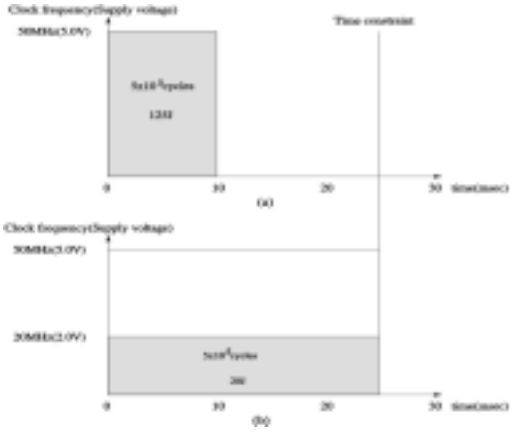



그림 1 전압 조정의 예; (a) 125J의 전력을 소비하며 15msec의 유휴시간을 가지는 경우, (b) 20J의 전력을 소비하며 유휴시간이 없는 경우

반드시 지켜져야 하기 때문에, 공급 전압을 조정할 때 연성 실시간 시스템(soft real-time system)에서보다 주의를 요한다. 최근에 동적 전압 조정 기법에 대한 많은 연구가 진행되었는데[3, 4, 5, 6, 7], 대부분의 연구들은 다중 태스크(multi-task)를 가진 경성 실시간 시스템에서 주어진 여러 태스크의 마감 시간 제약 조건을 지키면서 전력 소모를 최소화하도록 각각의 태스크에 적당한 클록 속도와 이에 따른 공급 전압을 결정하는 기법에 초점을 두었다. 이 기법들의 기본 아이디어는 현재 태스크에서 발생한 유휴 시간을 다음 태스크로 넘겨주어 다음 태스크가 클록 속도 및 공급 전압을 낮출 수 있도록 하는 것이다. 이론적으로는 주어진 태스크의 실행 사이클 수를 미리 알아서 최적의 클록 속도와 공급 전압을 계산한 후 이 클록 속도로 태스크를 수행시키는 방법이 이상적이지만 (오프라인 최적 전압 스케줄링), 실제적으로는 동일한 태스크라 하더라도 입력 데이타에 따라서 수행될 때마다 걸리는 사이클 수가 매번 다르므로, 태스크가 끝나기 전에는 실행 사이클 수를 알 수가 없다. 클록 속도 및 공급 전압은 “실행-계산-할당-실행”의 방법으로 결정되는데, (1) 현재의 태스크를 수행시키고, (2) 다음 태스크가 마감 시간 제약 조건을 지키면서 최대한 사용 가능한 시간을 계산하고, (3) 다음 태스크를 위한 클록 속도 및 공급 전압을 결정하고, (4) 다음 태스크를 실행시키는 방법을 말한다. 공급 전압은 태스크별로 결정되는데. 각각의 태스크가 실행될 때마다 그 태스크에 하나의 공급 전압이 할당되고 그 값은 주어진 태스크가 실행되는 동안 바뀌지 않고 일정한 값을

유지한다. 본 논문에서는 이러한 기법을 태스크간 전압 스케줄링(inter-task voltage scheduling) 기법이라고 부른다.

이러한 태스크간 전압 스케줄링 기법은 몇 가지 문제점을 가진다. 예를 들어, 운영체제(operating system)에 있는 스케줄러가 태스크의 사용 전압을 결정하기 때문에 운영체제의 수정이 불가피하다. 더욱이 이 방법은 단일 태스크(single-task) 환경에서는 사용할 수 없는데, 이것은 실질적으로 상업적 이동용 시스템에서 주로 사용되는 내장형 응용프로그램(embedded application program)에 대해서 동적 전압 조정 기법을 사용할 수 없음을 의미한다. 태스크간 전압 스케줄링 기법의 또 다른 문제점은 다중 태스크 환경이라 할지라도 하나의 태스크가 실행 시간과 유휴 시간 모두 다른 태스크에 비해서 상당히 크면 전력 소모를 줄이는데 별로 효과적이지 않다는 것이다. 전력 소모는 대체적으로 태스크의 실행 시간에 비례하고, 클록 속도 및 공급 전압은 다른 태스크에게서 넘겨받는 유휴 시간이 클수록 더 많이 낮출 수 있으므로, 결과적으로 전력 소모가 가장 큰 태스크는 유휴 시간을 조금밖에 못 넘겨받아서 전력 소모를 작은 비율로밖에 줄일 수 없게 된다. 이때 다른 태스크들이 유휴 시간을 많이 넘겨받아서 전력소모를 큰 비율로 줄일 수 있다고 해도 그들이 전체에서 차지하는 비중이 작으므로, 결과적으로 시스템 전체의 전력 소모는 그다지 줄일 수 없게 된다. 표 1은 4개의 태스크를 가진 전형적인 동영상 휴대전화의 응용프로그램의 예를 든 것인데, 오프라인 최적 전압 스케줄링이 90% 정도의 전력 소모 감소를 가져오는 반면, [6]에서 제시된 태스크간 스케줄링 기법은 단지 17%의 전력 소모 감소만을 가져온다.

이러한 문제점의 대안으로는 하나의 태스크를 여러 개의 영역으로 나누고, 태스크 내부에서 각 영역에 서로 다른 공급 전압을 할당하며, 운영체제가 아닌 응용프로그램 내부에서 클록 속도 및 공급 전압을 결정하는 태스크내 전압 스케줄링 (intra-task voltage scheduling) 기법을 들 수 있다. 이 기법은 운영체제의 수정이 필요없고, 단일 태스크 환경에 적용이 가능하며, 태스크들의 실행 시간이나 유휴 시간이 서로 크게 차이가 나더라도 비교적 효과적으로 전력 소모를 줄일 수 있다는 장점을 가지고 있다. Lee 등[8]은 각각의 태스크를 고정된 길이의 시간 간격으로 나누어 공급 전압을 할당하는 태스크내 전압 스케줄링 기법을 제시했다. 이 방법은 태스크간 전압 스케줄링 기법에 비해서 상당히 전력 소모를 줄일 수 있었지만, 동적 전압 기법을 적용하기 위해서 프로그램 개발자가 일일이 손으로 응용프로그램을 고쳐야 하며 태스크 내부를 여러 구간으로 나누는 위치를 선택하기 위한 체계적인 방법론을 제시하지 못했다. 일반적인 프로그램 개발자들은 동적 전압 조정 기법뿐만 아니라 시간 분석(timing analysis) 기법에 대해서도 익숙하지 않기 때문에, 체계적인 방법론이 없이 실시간 응용프로그램에 대해서 태스크내 전압 스케줄링을 사용하는 것은 현실적으로 매우 어렵다.

표 1 전형적인 동영상 휴대 전화기의 응용프로그램

| | MPEG-4 Video Encoding | MPEG-4 Video Decoding | VSELP Speech Encoding | VSELP Speech Decoding |
|---|---|---|---|---|
| 주기(= 마감시간)(msec) | 66.667 | 66.667 | 40.000 | 40.000 |
| 평균 유휴 시간(msec) | 37.287 | 8.366 | 0.937 | 0.703 |
| 평균 실행 시간(msec) | 13.099 | 1.460 | 0.907 | 0.680 |
| 정규 전력 소모(Inter)[a] | 0.826 | | | |
| 정규 전력 소모(Ideal)[b] | 0.106 | | | |

[a]태스크간 스케줄링[6]에 의해 소모된 정규화된 전력 소모값.
[b]오프라인 최적 전압 스케줄링에 의해 소모된 정규화된 전력 소모값.
*전력 소모값은 전원 차단 기능을 가지며 DVS를 사용하지 않는 시스템에 의해서 정규화 되었다.

본 논문에서는 응용프로그램에 대한 정적 시간 분석(static timing analysis) 기법을 이용한 새로운 태스크내 전압 스케줄링 기법을 제시한다. 제시된 기법은 다음과 같은 특징을 가진다.

1. 입력 데이타가 변함에 따라 실제 실행 시간이 변하면서 발생하는 모든 유휴 시간을 이용하기 때문에 전력 소모 감소에 있어서 상당한 효과가 있다. 이 기법은 태스크 내부에서 동적으로 전압을 조정하기 때문에, 유휴 시간을 완전히 이용함으로서 얻어지는 전력 소모 감소 효과는 다중 태스크 환경뿐만 아니라 단일 태스크 환경에서도 유효하다.
2. 동적 전압 조정 기법을 고려하지 않고 개발된 응용프로그램을 동적 전압 조정을 사용하는 응용프로그램으로 자동으로 바꾸는 변환 도구를 위한 체계적인 방법론을 제시하고, 자동화된 프로그램 변환 도구를 구현한다. 따라서 프로그램 개발자에게 동적 전압 조정에 대한 어떠한 지식도 요구하지 않는다.
3. 운영체제의 어떤 도움도 없이 각각의 태스크가 다른 태스크로부터 독립적으로 전압을 조정할 수 있도록 한다. 따라서 기존의 운영체제를 수정하지 않고 그대로 사용할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 사용될 용어를 정의하고 가변 전압 프로세서의 모델

을 설명한다. 3장에서는 제안된 태스크내 전압 스케줄링을 설명하고, 4장에서는 소프트웨어 구조 및 모의 실험 결과를 논의한다. 5장에서는 결론과 향후 작업에 대한 방향을 제시한다.

## 2. 용어 정의 및 가변 전압 프로세서

### 2.1 용어 정의

마감시간 $D_\tau$[1]를 가진 실시간 태스크 $\tau$가 있을 때, 이 태스크는 제어 흐름 그래프(control flow graph: CFG) $G$로 표현된다. 만약 태스크 $\tau$가 $N$개의 기본 블록(basic block) $b_1, b_2, \cdots, b_N$을 가지고 있다면 $G$는 $N$개의 노드(node)로 구성된다. 여기서 $b_1$은 태스크 $\tau$의 첫번째 기본 블록으로 가정하며, $G$의 노드 $b_i$는 기본 블록 $b_i$를 나타낸다. $G$의 각 에지(edge) $e=(b_i, b_j)$는 기본 블록 $b_i$와 $b_j$사이에 제어 의존성(control dependency)이 있음을 의미한다. 각각의 기본 블록 $b_i$는 관련된 기본 블록 정보 BBI($b_i$)를 가지고 있는 데, BBI($b_i$)는 $C^{EC}(b_i)$, $C^{RWEC}(b_i)$ 그리고 $S(b_i)$의 3개의 요소로 이루어져 있다. $C^{EC}(b_i)$는 $b_i$를 실행하는 데 필요한 클록 사이클 수를 나타내며, $C^{RWEC}(b_i)$는 $b_i$로부터 출발하여 태스크의 끝까지 가는 모든 가능한 실행 경로(execution path)중에서 가장 긴 실행 경로의 남은 실행 사이클 수인 잔여 최악 실행 사이클 수(remaining worst case execution cycle: RWEC)를 나타낸다. $S(b_i)$는 $b_i$가 실행될 때의 클록 속도를 주파수 값으로 나타낸 것이다. BBI를 정의하는 데, 실행 시간이 아닌 실행 사이클 수를 사용하였는 데, 이것은 가변 전압 프로세서에서 전압과 클록 속도를 조정하면 실행 시간은 변하지만 실행된 사이클 수는 불변하기 때문이다. 실제의 실행 시간은 주어진 실행 사이클 수를 클록 속도로 나눈 값으로 계산된다.

실행 경로에 대해서도 유사한 용어를 다음과 같이 정의한다.

- $p_i$ : 태스크 $\tau$의 하나의 실행 경로. $p_i$는 연속된 여러 개의 기본 블록들로 표현된다. 특히, 최악 실행 경로는 $p_{worst}$로 표현한다.
- $C^{EC}(p_i)$ : $p_i$가 실행될 때 실행 사이클의 수. $p_{worst}$의 실행 사이클 수인 최악 실행 사이클 수(worst case execution cycle: WCEC)는 $C^{WCEC}$로 표현한다.

### 2.2 가변 전압 프로세서 모델

본 논문 전체에서는 다음과 같은 가변 전압 프로세서(variable voltage processor)를 가정하고, 본 논문에서 제안된 태스크내 전압 스케줄링은 이 가변 전압 프로세서에서 동작한다고 가정한다.

1. 가변 전압 프로세서는 change_f_V($f_{CLK}$)라는 특별한 명령어를 제공한다. 이 명령어는 프로세서의 클록 속도 $f_{CLK}$와 이에 따른 해당하는 공급 전압 $V_{DD}$를 조정한다.
2. $f_{CLK}$과 $V_{DD}$는 가변 전압 프로세서가 동작할 수 있는 범위 내에서 임의의 연속된 값으로 조정할 수 있다.
3. 가변 전압 프로세서가 클록 속도와 공급 전압을 ($f_{CLK1}$, $V_{DD1}$)에서 ($f_{CLK2}$, $V_{DD2}$)로 바꿀 때, 클록과 공급 전압을 바꾸는 데는 $C_{VTO}$만큼의 사이클이 소요된다.[2]
4. 클록과 공급 전압을 바꾸는 동안에 가변 전압 프로세서는 실행을 멈추거나(프로세서 I형) 또는 클록 속도를 min($f_{CLK1}$, $f_{CLK2}$)로 하여 실행된다(프로세서 II형).

가정 1-3은 현재 존재하는 대부분의 가변 전압 프로세서에 있어서 유효하다[8, 9, 10]. 최신의 주파수 합성기(frequency synthesizer)나 직류-직류 변환기(DC-DC converter)는 클록 속도와 공급 전압을 변화하는 데 200μsec이하의 시간이 걸리는 데 이것은 100Mhz에서 20,000cycle에 해당한다. 가정 4는 클록 속도와 공급 전압을 변화하는 동안에 가변 전압 프로세서가 어떻게 동작하는지를 나타낸다. 이 동작 형태는 하드웨어 구조에 따라 달라지는데, 예를 들어 기존의 고정 전압 프로세서를 수정하지 않고 외부에 부가 장치만을 추가하는 방식[8]은 프로세서 I형에 속하며, Transmeta사의 Crusoe 프로세서[10]는 프로세서 II형이다. 본 논문에서 제안하는 태스크내 전압 스케줄링 기법은 클록 속도와 공급 전압에 대한 모델링을 약간 수정함으로서 두 가지 종류의 프로세서를 모두 지원할 수 있다.

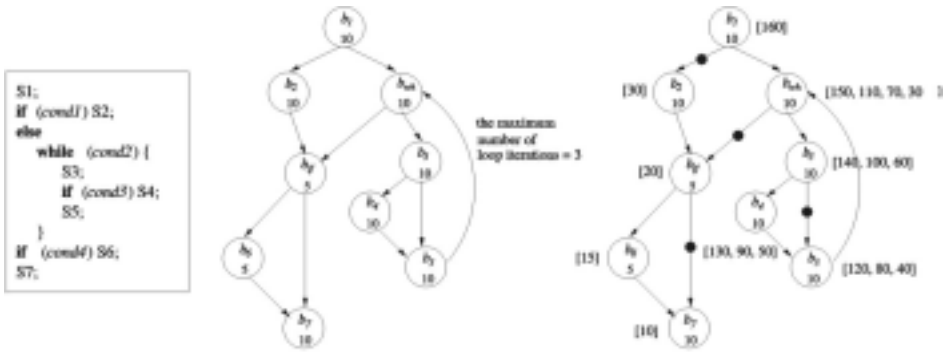
## 3. 태스크내 전압 스케줄링

### 3.1 기본 동작 방식

그림 2(a)처럼 마감시간 2μsec의 조건을 가진 경성

1) 본 논문에서 제안하는 태스크내 전압 스케줄링에서는 각각의 태스크가 다른 태스크로부터 독립적으로 클록 속도 및 공급 전압을 조정하므로, 단일 태스크 환경의 경우에 대해서만 설명해도 다중 태스크 환경에 대해서 동일한 방법으로 적용된다. 따라서 앞으로의 설명에서 용어 표기시 $\tau$를 생략한다.

2) 클록 속도와 공급 전압을 변화하는 데는 하드웨어 특성에 따라 일정 시간이 소요되며 이 값은 사이클 수로 나타낼 때 현재의 클록 속도에 따라 변한다. 분석을 간단하게 하기 위해서 $C_{VTO}$는 최대 클록 속도로 환산했을 때의 사이클 수라고 가정한다.




(a) (b) (c)

실시간 프로그램 $P$를 생각해 보자. 프로그램 $P$의 제어 흐름 그래프 $G_P$는 그림 2(b)에 나타나 있다. $G_P$에서 각각의 노드는 프로그램 $P$의 기본 블록을 의미하며 각 에지는 기본 블록간의 제어 의존성을 나타낸다. 노드안의 숫자는 기본 블록의 실행 사이클 수를 나타낸다. 기본 블록 $b_5$로부터 $b_{wh}$로의 에지는 프로그램 $P$의 **while** 반복문을 나타내고 있다.

모든 태스크가 엄격하게 시간 제약 조건(즉, 마감 시간)을 지켜야 하는 경성 실시간 프로그램을 개발할 때는 시간 제약 조건이 지켜지는 것을 보장하기 위해서 실행 전에 미리 최악 실행 시간(worst case execution time: WCET)을 측정한다. 현존하는 WCET 분석 도구[11, 12, 13]는 몇가지 사용자 제공 정보(반복문의 최대 반복 회수 등)를 이용하여 최악 실행 경로(WCEP)를 찾아내고 이때의 최악 실행 시간(WCET)을 정확하고 안전하게 예상할 수 있다. 예로 든 프로그램 $P$에서 **while** 반복문의 최대 반복 회수가 사용자에 의해 3으로 주어졌을 때, WCET 분석 도구를 사용하여 $p_{worst}$=($b_1$, $b_{wh}$, $b_3$, $b_4$, $b_5$, $b_{wh}$, $b_3$, $b_4$, $b_5$, $b_{wh}$, $b_3$, $b_4$, $b_5$, $b_{wh}$, $b_{if}$, $b_6$, $b_7$)를 WCEP로 찾을 수 있다. 이때 $p_{worst}$의 실행 사이클 수는 160cycle이며 이 값이 WCEC이다. 만약에 프로세서가 클록 속도 80Mhz로 동작한다면, 프로그램 $P$는 유휴시간을 전혀 발생시키지 않으며 2μsec에 끝나게 된다.

제안하는 알고리즘은 서로 다른 실행 경로에 따라 실행 시간에 큰 변동이 있다는 점에 착안하고 있다. 특히, 평균 실행 경로(average case execution path: ACEP)는 최악 실행 경로(WCEP)보다 훨씬 이전에 끝난다는 사실[6]을 이용한다. 예를 들면 그림 2(b)에 있는 예제 프로그램에는 서로 다른 32개의 실행 경로가 있다. 최악 실행 경로 $p_{worst}$는 160 사이클이 걸리지만 32개의 가능한 경로 중 8개의 경로는 80 사이클보다 작은 시간이 걸린다. 만약 프로그램이 어느 실행 경로를 택하는지 미리 알 수 있다면 이러한 짧은 실행 경로에 대해서는 클록 속도를 충분히 낮추어 전력 소모를 크게 줄일 수 있다.

실행에 40 사이클이 걸리는 그림 2(b)의 실행 경로 $p_1$=($b_1$, $b_2$, $b_{if}$, $b_6$, $b_7$)를 생각해 보자. 실행될 경로가 $p_1$이라고 미리 예상할 수 있다면 20Mhz의 클록 속도로 태스크를 시작해도 2μsec의 마감 시간을 지킬 수 있으며, 클록 속도가 80Mhz에서 20Mhz로 낮춰짐에 따라서 전력 소모를 크게 줄일 수 있다. 하지만 실제로는 어떤 실행 경로가 선택될 지 미리 알 수 없기 때문에, $b_1$에서부터 20Mhz의 클록 속도로 태스크를 시작하는 것은 불가능하다. 본 논문에서 제시하는 알고리즘은 최악 실행 시간에 대한 정적 프로그램 분석 기법을 이용한 적응적(adaptive) 접근법을 사용한다. 본 논문에서 사용된 분석 도구는 기존의 WCET 분석 도구를 수정하여 각각의 기본 블록 $b_i$에 대해서 $C^{RWEC}(b_i)$를 컴파일 시간에 계산하고, 이를 제어 흐름 그래프에 추가한다. 그림 2(c)는 $C^{RWEC}(b_i)$를 추가한 제어 흐름 그래프 $G_P{}^A$를 보여주고 있다.

**while** 반복문과 관련된 기본 블록들 $b_{wh}$, $b_3$, $b_4$, $b_5$에 대해서는 여러 개의 $C^{RWEC}(b_i)$ 값들이 나타나 있는 데, 이것은 **while** 반복문이 3번 반복되기 때문이다. 일단 $G_P{}^A$가 만들어지면 잔여 실행 사이클 수(RWEC)가 현재 실행 속도보다 더 빨리 감소하는 분기문 에지를 찾을 수 있다. 예를 들어, 그림 2(c)에서 ($b_1$, $b_2$), ($b_{wh}$, $b_{if}$), ($b_{if}$, $b_7$), ($b_3$, $b_5$)가 이러한 에지에 해당한다. 이 에지들은 그림에서 ●로 표시되어 있다. 프로그램의 실행 흐름이 이들 에지중의 하나인 ($b_1$, $b_2$)를 거쳐서 다음 기본 블록으로 갈 경우, 프로그램에서 앞으로 수행해야 하는 남은 부분이 $C^{RWEC}(b_{wh})$와 $C^{RWEC}(b_2)$의 차이만큼

줄어들기 때문에 클록 속도가 그만큼 낮추어 질 수 있다. 즉, 제안된 알고리즘은 $C^{RWEC}(b_2)$ 사이클이 마감시간까지 정확히 끝날 수 있도록 클록 속도를 줄임으로서 시간 제약 조건을 항상 만족시킨다. 이때 전압 조정 지점은 실행 시(run-time)가 아닌 컴파일 시(compile-time)에 결정됨으로 실행 시에 추가되는 작업은 없다. 더욱이 컴파일 시간의 정적 분석 과정은 일반적인 경성 실시간 프로그램 개발 시에 요구되는 정보, 이를테면 반복문의 최대 반복 회수 등과 같은 정보 외에는 특별히 프로그램 개발자의 관여를 요구하지 않는다.

그림 3은 태스크내 전압 스케줄링의 사용 여부에 따라 클록 속도와 공급 전압이 어떻게 바뀌는 지 비교하고 있다. 유휴 시간 동안에는 프로세서가 동작을 멈추고 전력이 전혀 소모되지 않는다고 가정하면, 태스크의 실행이 경로 $p_1$=($b_1$, $b_2$, $b_{if}$, $b_6$, $b_7$)를 따라 갈 때, 그림 3(b)의 전력 소모는 그림 3(a)의 31%에 불과하므로, 태스크내 스케줄링을 사용하여 전력 소모를 69% 줄일 수 있다.

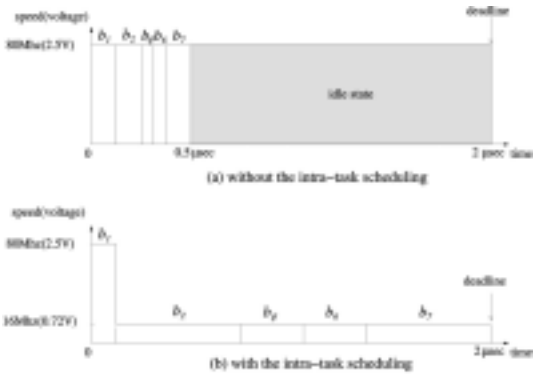



그림 3 태스크내 스케줄링에 의한 클록 속도와 전압의 변화

### 3.2 속도 할당 알고리즘

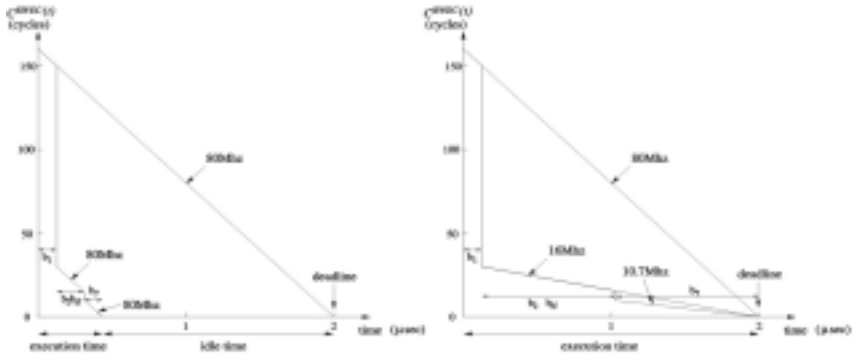
이 장에서는 태스크 내에서 각 기본 블록 별로 적당한 클록 속도를 할당하는 속도 할당 알고리즘을 자세히 설명한다. 경성 실시간 태스크에게 있어 속도 할당 알고리즘의 목표는 마감 시간 제약 조건을 만족시키면서 전력 소모가 최소화가 되도록 속도를 할당하는 것이다.

만약 마감 시간이 $D$인 태스크 $\pi$의 실제 실행 경로인 $p_{act}$를 미리 알 수 있으면, 최적의 실행 속도는 [14]에 의해 쉽게 구할 수 있다. 즉, $p_{act}$상의 각 기본 블록 $b_i$에 대해 $S(b_i) = C^{EC}(p_{act})/D$가 된다. 그러나, 일반적으로 실제 실행 경로를 프로그램의 실행 완료 전에는 미리 알 수 없기 때문에 본 알고리즘에서는 잔여 최악 실행 사이클 수 인 $C^{RWEC}(b_i)$의 값에 따라 $S(b_i)$를 조정해 간다. 본 논문에서는 [12]에서 사용된 것과 같은 WCET 분석 도구를 수정하여 각 $b_i$에 대해 $C^{RWEC}(b_i)$를 구할 수 있도록 하였다. $S(b_i)$는 남아있는 부분이 앞으로 $C^{RWEC}(b_i)$ 사이클만큼 수행된다고 가정하고, 태스크가 마감 시간에 정확히 끝나도록 할당된다. $C^{RWEC}(b_i)$의 계산 또한 실행 시(run-time)가 아닌 컴파일 시(compile-time)에 결정되기 때문에, 실행 시에 추가되는 작업은 없다.

태스크의 시작 기본 블록인 $b_1$에서 $C^{RWEC}(b_1)$는 $C^{WCEC}$로 결정되며 출발 속도는 $C^{WCEC}/D$로 정해진다. 시간 $t$에서 잔여 최악 실행 사이클 수를 $C^{RWEC}(t)$로 나타낼 때, 태스크의 실행이 최악 실행 경로 $p_{worst}$를 따라 간다면 $C^{RWEC}$(t)는 태스크가 진행됨에 따라 클록 속도에 따라 선형적으로 줄어들게 된다. 그러나, 만약 실행 경로가 최악 실행 경로인 $p_{worst}$상의 기본 블록 $b_i$로부터 벗어나 $p_{worst}$상에 있지 않은 기본 블록 $b_j$로 가게 되면 $C^{RWEC}(t)$는 $b_i$의 실행이 완료된 후 $C^{RWEC}(b_i) - C^{EC}(b_i)$와 $C^{RWEC}(b_j)$의 차이만큼 값이 줄어들게 된다.

그림 4는 그림 2의 예제 프로그램 $P$의 경로 $p$ = ($b_1$, $b_2$, $b_{if}$, $b_7$)가 실행될 때 $C^{RWEC}(t)$가 어떻게 동적으로 변하는 지를 보여주고 있다. 속도 스케줄링을 하지 않는 그림 4(a)에서 $C^{RWEC}(t)$는 $C^{EC}(b_1)$/80Mhz와 ($C^{EC}(b_1)$+$C^{EC}(b_2)$+$C^{EC}(b_{if})$)/80Mhz의 두 지점에서 떨어진다. 그림 4(a)에서는 속도 스케줄링을 사용하지 않기 때문에 $C^{RWEC}(t)$는 80Mhz의 속도로 줄어들며 1.5625μsec의 유휴시간을 발생시킨다. 그림 4(b)는 같은 실행 경로에 대해서 속도 스케줄링을 사용했을 때 효과를 보여 주고 있다. $b_1$의 완료 후 $C^{RWEC}(t)$가 갑자기 감소하므로 속도가 80Mhz에서 마감시간까지 잔여 프로그램을 완료할 수 있는 속도인 16Mhz로 바뀌었다. $C^{RWEC}(t)$가 $b_{if}$이후에 떨어질 때도 같은 이유로 속도는 낮추어진다.

본 논문에서 제안된 태스크내 스케줄링 알고리즘은 RWEC를 바탕으로 모든 경로가 마감시간에 인접해서 수행을 끝내도록 해주기 때문에 유휴 시간이 거의 남지 않아 전력 소모 측면에서 효율적이며, 스케줄 된 프로그램이 어떠한 경우에도 항상 마감 시간 제약 조건을 만족하도록 보장한다. 그림 4에서 보면 $C^{RWEC}(t)$가 수직으로 떨어지는 지점들에서 전압과 속도가 조정될 수 있으며, 본 논문에서는 이들 지점을 전압 조정 에지(voltage scaling edge: VSE)라고 부르고 VSE에서 줄어든 사이클을 $C_{saved}$로 나타낸다.



(a) (b)

그림 4 속도 조정 알고리즘에 따른 $C^{RWEC}(t)$의 변화: (a) 태스크내 스케줄링을 사용하지 않은 경우, (b) RWEC를 이용한 태스크내 스케줄링을 사용한 경우

### 3.3 B형 전압 조정 에지(VSE)

VSE는 B형과 L형의 두 가지 종류로 구분되는 데, B형 VSE는 if문과 같은 조건문내의 CFG 에지에 해당한다. if문에 대해서 WCET는 then 경로와 else경로에 해당하는 실행 시간 중에서 큰 쪽의 값이 된다. if문의 조건이 기본 블록 $b_{cond}$에서 검사되며 then 경로는 $b_{then}$에서부터, else 경로는 $b_{else}$에서부터 시작한다고 가정하자. 만약 if문의 조건이 참(true)으로 결정되고 then 경로가 else 경로보다 짧다고 하면, $C^{RWEC}(t)$는 ($C^{RWEC}(b_{else})$ $-C^{RWEC}(b_{then})$)만큼 감소한다. 이때 프로세서 속도는 $b_{then}$ 블록이 실행되기 직전에 $\frac{C^{RWEC}(b_{then})}{C^{RWEC}(b_{else})}$의 비율로 감소될 수 있다. 이 비율을 속도 *조정율* (speed update ratio)이라고 부르고 $r(b_{cond} \rightarrow b_{then})$로 나타낸다. 같은 기본 블록도 여러 가지 다른 클록 속도로 실행될 수 있기 때문에 각각의 VSE에는 고정된 속도 값이 아닌 속도 조정율이 할당된다. 예를 들어, 그림 2에서 $b_{if}$와 $b_7$사이의 VSE에 고정 속도 값이 할당된다면 $p_{worst}$가 2μsec의 마감시간 이전에 마칠 수 있도록 53.5Mhz(=80Mhz× $\frac{10}{15}$)가 할당되어야 한다. 그러나, 만약 $b_{if}$까지 실행된 경로가 ($b_1$, $b_2$, $b_{if}$)이라면 태스크의 실행은 53.3Mhz의 속도로 수행되어 마감 시간보다 훨씬 일찍 끝나게 되고, 긴 유휴 시간이 발생한다. 이러한 문제는 VSE에 고정 속도 값을 할당하는 대신에 속도 조정율 ($r(b_{if} \rightarrow b_7) = \frac{2}{3}$)을 할당함으로써 피할 수 있다.

VSE에서 클록 속도와 전압을 조정할 때에는 하드웨어적으로 클록 속도와 공급 전압을 바꿔주는 명령어인 change_f_V($f_{CLK}$)외에도 약간의 부가적인 계산이 필요하다. B형 VSE에서 이러한 부가 계산을 실행하는 데 필요한 사이클 수를 $C_{VSO,B}$로 나타내면 B형 VSE에서의 속도를 바꾸기 위해서 늘어나는 실행 사이클 수 $C_{overhead,B}$는 $C_{VTO}+C_{VSO,B}$로 주어진다. 이때 B형 VSE ($b_i$,$b_j$)를 위한 속도 조정율 $r(b_i \rightarrow b_j)$는 다음과 같이 계산된다.

$$r(b_i \rightarrow b_j) = \frac{C^{RWEC}(b_j)}{C^{RWEC}(\text{succ}_{worst}(b_i)) - C_{\text{overhead},B}} \quad (1)$$

여기서 $succ_{worst}(b_i)$는 $b_i$의 직후에 실행될 수 있는 기본 블록중에서 가장 큰 $C^{RWEC}(b_k)$를 가진 기본 블록 $b_k$를 나타낸다.

제안된 알고리즘은 두 가지 형태의 가변 전압 프로세서(프로세서 I형과 II형)를 모두 지원할 수 있지만 여기서는 설명을 간단하게 하기 위해서 전압이 변하는 동안 프로세서는 실행을 멈춘다고 가정한다. 이때 $C^{RWEC}(b_j)$ $\geq$ $C^{RWEC}(succ_{worst}(b_i))-C_{overhead,B}$라서 $r(b_i \rightarrow b_j) \geq 1$가 되면 이 에지 ($b_i$, $b_j$)는 속도를 변경하여 이득을 얻지 못하므로 VSE로 선택되지 않는다. $b_i$와 $b_j$사이의 VSE에서는 $b_j$가 실행되기 전에 현재의 속도에 속도 조정율 $r(b_i \rightarrow b_j)$이 곱해져 속도가 변경된다. 즉, $S(b_j)$는 현재 속도 × $r(b_i \rightarrow b_j)$로 정해진다. 예를 들어, $C_{overhead,B}$가 0이라고 가정하고 그림 2의 실행 경로 중 하나인 ($b_1$, $b_2$, $b_{if}$, $b_7$)를 따라서 태스크가 실행될 때 B형 VSE에서 속도가 어떻게 변하는지 생각해 보자. 기본 블록 $b_2$가 실행되기 직전에 RWEC는 150으로부터 30으로 감소하게 되고 클록 속도는 80Mhz에서 16Mhz(=80Mhz× $\frac{30}{150}$)로 줄어든다. 기본 블록 $b_7$을 위한 속도도 RWEC가 15에서 10으로 줄어들기 때문에 16Mhz에서 10.7Mhz (=16Mhz× $\frac{10}{15}$)로 줄어든다.

### 3.4 L형 전압 조정 에지(VSE)

$C^{WCEC}$는 사용자가 제공한 반복문의 최대 반복 회수

만큼 실행될 것으로 가정하고 계산되지만, 일반적으로 반복문은 이 최대값보다는 작은 회수만큼 실행하게 된다. 이 경우에 유휴시간이 생기며 클록 속도와 공급 전압은 더 낮춰질 수 있다. 이러한 형태의 전압 조정을 L형 전압 조정이라고 한다. L형 VSE는 CFG의 반복문 출구 부분에 해당한다. L형 VSE에서 반복문 $l$에 대한 남은 사이클 수 $C_{saved}$는 다음과 같이 주어진다.

$$C_{saved}(l) = C^{WCEC}(l) \cdot (N_{worst}(l) - N_{exec}(l)) \quad (2)$$

여기서, $C^{WCEC}(l)$는 반복문 $l$을 한번만 실행하는 데 걸리는 최악 실행 사이클 수이고, $N_{worst}(l)$는 반복문 $l$에 대해 사용자가 제공한 최대 반복 회수이며, $N_{exec}(l)$는 실행시에 실제 반복문 $l$이 반복되는 회수이다. 그림 2에서 L형 VSE인 에지 ($b_{wh}$, $b_{if}$)를 생각해 보자. B형 VSE에서처럼 속도 조정을 위해서 늘어나는 실행 사이클수를 $C_{overhead,L}$로 나타내고

$N_{exec}(l)$ = 1, 그리고 $C_{overhead,L}$=0라고 가정할 때, $S(b_{if})$는 다음과 같이 정해진다.

$$\begin{aligned} S(b_{\text{if}}) &= S(b_{wh}) \cdot \frac{C^{RWEC}(b_{\text{if}})}{C^{RWEC}(b_{\text{if}}) + C_{saved}(l) - C_{\text{overhead},L}} \\ &= S(b_{wh}) \cdot \frac{20}{20 + 40 \cdot (3-1)} \\ &= S(b_{wh}) \cdot r(b_{wh} \rightarrow b_{\text{if}}) \end{aligned} \quad (3)$$

$S(b_{wh})$가 80Mhz일 때, $S(b_{if})$는 $b_{if}$를 실행하기 전에 16Mhz로 된다. B형 VSE와는 달리, L형 VSE에서의 속도 조정율을 결정할 때는 $N_{exec}(l)$와 같은 실행 시 정보가 필요하다.[3] 속도 조정율은 $N_{exec}(l)$와 $C_{overhead,L}$의 값에 따라 1보다 커질 수도 있다. 이 문제를 해결하기 위해 L형 VSE는 두 단계로 선택된다. 먼저 $C^{WCEC}(l) > C_{overhead,L}$를 만족하는 반복문 $l$의 출구 에지를 L형 VSE의 후보로 선택한다. 이 조건은 $N_{exec}(l) < N_{worst}(l)$이기만 하면 속도 조정율이 항상 1보다 작다는 것을 의미한다. 하지만 만약 $N_{exec}(l) = N_{worst}(l)$이면 클록 속도는 변하지 않으며 $N_{exec}(l) = N_{worst}(l)$의 여부를 검사하는 코드에 의해서 원래 프로그램의 시간적인 동작이 달라지게 된다. 그래서 L형 후보들 중에서 다시 3.5장에서 설명할 알고리즘을 사용하여 최종 L형 VSE들을 선택하게 된다. L형 VSE는 B형 VSE보다 좀 더 복잡하지만, 반복문으로부터 생기는 유휴시간이 조건문으로부터 생기는 유휴시간보다 일반적으로 크기 때문이 L형 VSE가 전체 전력 소모를 감소시키는 효과도 B형 VSE에 비해서 더 크다.

### 3.5 VSE 선택 알고리즘

B형 VSE를 위한 전압 조정 코드는 주어진 프로그램의 $C^{WCEC}$를 증가시키지 않지만, L형 VSE를 위한 전압 조정 코드는 반복문의 실행된 회수에 따라서 $C^{WCEC}$를 증가시킬 수 있다. 만약에 반복문이 사용자가 제공한 최대 반복회수만큼 실행되고 이 반복문의 출구 에지가 L형 VSE의 후보로 선택되었다면, 프로그램의 $C^{WCEC}$는 반복문의 실행 회수를 검사하는 코드의 크기만큼 증가하게 된다. 이러한 $C^{WCEC}$의 증가는 누적될 경우, 수정된 프로그램이 원래 프로그램의 마감 시간 제약 조건을 어기는 경우가 발생할 수도 있다.

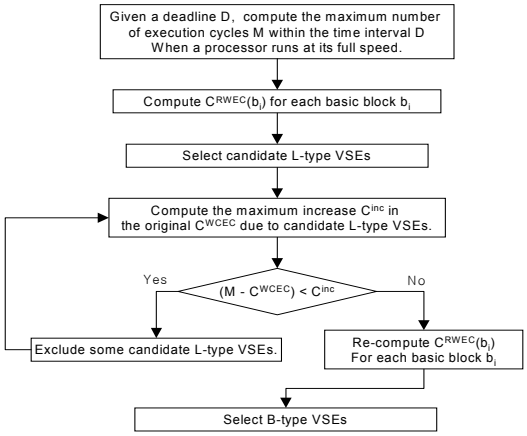



그림 5 전체 VSE 선택 알고리즘

본 논문에서 제안된 알고리즘에서는 이러한 상황을 피하기 위해서 그림 5에 나타난 알고리즘에 의해 L형 VSE들로부터 마감 시간 제약 조건을 어기는 것들을 배제한다. 가변 전압 프로세서가 주어진 마감시간 안에 최대 클록 속도로 $M$ 사이클을 실행할 수 있다고 할 때, 먼저 L형 VSE의 후보들이 모두 VSE로 선택될 경우 추가된 코드에 의해서 요구된 시간 제약 조건을 어기는지 살펴본다. 만약 어긴다면 L형 VSE의 후보들 중에서 몇 개를 $C^{WCEC}$가 마감시간을 만족시킬 때까지 하나씩 제외시킨다. L형 VSE들이 모두 결정된 다음에 각 $C^{RWEC}(b_i)$가 새로 계산되고 B형 VSE가 선택된다.

VSE들이 선택된 후에는 프로그램 코드 안에 전압 조정 코드가 삽입된다. 그림 6은 VSE를 위해서 생성된 코드의 예를 보여주고 있다. 에지 ($b_1$, $b_2$)가 B형 VSE이기 때문에 그림 6에서 보듯이 전압 조정 코드인 코드 B가 에지 ($b_1$, $b_2$)에 들어가 있다. 속도와 전압을 바꾸기 위해서는 식 (1)에 의해 컴파일 시간에 만들어진 클록 속도 표로부터 VSE의 해당하는 속도 조정율을 읽게 된다. 그리고 현재의 클록 속도에 읽어 들인 속도 조정

3) L형 VSE의 위치는 컴파일 시에 결정되며, 속도 조정율을 결정할 때만 실행 시 정보가 필요하게 된다.

율을 곱하게 되고 이 값이 사용할 새로운 클록 속도가 된다. 끝으로 가변 전압 프로세서의 클록 속도와 전압을 바꾸는 change_f_V이라는 명령어가 새로운 클록 속도 값과 함께 실행된다.

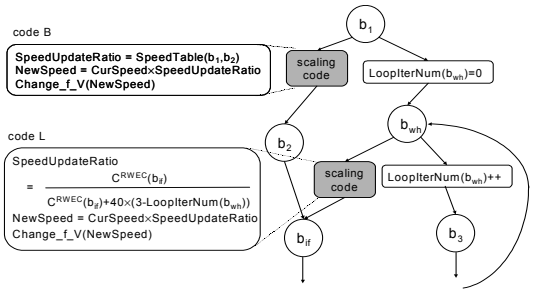



그림 6 VSE를 위한 코드 생성

또한 그림 6은 L형 VSE를 위한 전압 조정 코드인 코드 L의 예도 보여주고 있다. L형 VSE는 반복문이 끝날 때까지 모두 몇 번이나 반복되었는지를 알아야 하기 때문에 현재의 반복 회수를 기록할 추가의 코드가 필요하다. 그림 6에서 ($b_1$, $b_{wh}$)와 ($b_{wh}$, $b_3$) 사이의 두 군데에 이 코드가 삽입되어 있다.

## 4. 실험 결과

### 4.1 소프트웨어 구조

본 연구에서는 3장에서 제안된 태스크내 전압 스케줄링 기법을 바탕으로 기존 응용프로그램을 가변 전압 프로세서에서 동작할 수 있도록 자동으로 바꿔주는 자동 전압 조정기(Automatic Voltage Scaler: AVS)라는 소프트웨어 도구를 개발하였다. AVS는 가변 전압 프로세서를 고려하지 않은 일반적인 응용프로그램 $P$와 이것의 마감 시간 제약 조건을 입력받은 다음, 가변 전압 프로세서에서 마감 시간 제약 조건을 지키면서 동작하는 저전력 응용프로그램인 $P_{DVS}$로 변환한다. 변환된 응용프로그램 $P_{DVS}$에는 가변 전압 프로세서에서 클록 속도와 공급 전압을 조절하기 위한 전압 조정 코드가 필요한 부분마다 삽입되어있다. AVS를 사용하면 가변 전압 프로세서에 대한 아무런 고려 없이 개발된 경성 실시간 프로그램이 소프트웨어 개발자의 관여 없이 자동적으로 가변 전압 프로세서에서 동작할 수 있는 저전력 프로그램으로 변환된다. 본 논문에서 개발된 AVS는 현재 MIPS R3000 명령어 집합 구조를 가정하고 있다.

그림 7은 AVS의 전체 구조를 보여주고 있다. AVS는 크게 최악 실행 시간 예측기 (WCET Predictor: WP) 모듈과 전압 조정기(Voltage Scaler: VS) 모듈의 두 부분으로 나누어진다. WP 모듈은 입력 프로그램의 모든 기본 블록에 대해서 $C^{RWEC}(b_i)$를 측정하는 역할을 담당하며, 주어진 기본 블록 $b_i$의 $C^{RWEC}(b_i)$를 측정하기 위해서 Lim 등[12]에 의해서 개발된 시간 분석 도구를 수정하여 사용하였다. [12]의 시간 분석 도구는 원래 확장된 시간 스키마(extended timing schema) [12]의 시간 공식을 사용하여 상향식 방법으로 전체 프로그램의 구문 트리(syntax tree)를 탐색하여 WCET를 측정한다. 그러나 AVS는 각각의 기본 블록으로부터의 잔여 실행 사이클 수(RWEC)를 알아야 하기 때문에 원래의 시간 분석 도구를 이 목적에 맞게 수정하였다. 그림 7에서 보듯이 WP 모듈도 [12]의 시간 분석 도구처럼 $C^{RWEC}(b_i)$를 계산하기 위해서 상위 수준의 언어(예를 들면 C/C++언어)로 된 프로그램소스와 반복문의 최대 반복 회수와 같이 일반적인 경성 실시간 응용 프로그램을 개발하기 위한 사용자 제공 정보를 입력으로 받는다.

VS 모듈은 프로그램 구문 트리와 $C^{RWEC}(b_i)$값을 가지고 VSE를 찾고 이들 에지에 적당한 클록 속도 조정율을 할당하며 전압 조정 코드를 삽입한 다음, 변환된 프로그램을 생성한다. 그림 7에서 속도 할당기(Speed Allocator) 모듈은 3장에서 설명된 VSE 알고리즘을 사용하여 VSE를 선택하며 각 VSE에 적당한 속도 조정율 $r$을 할당한다.

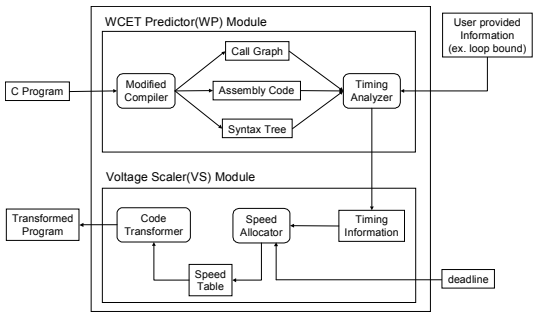


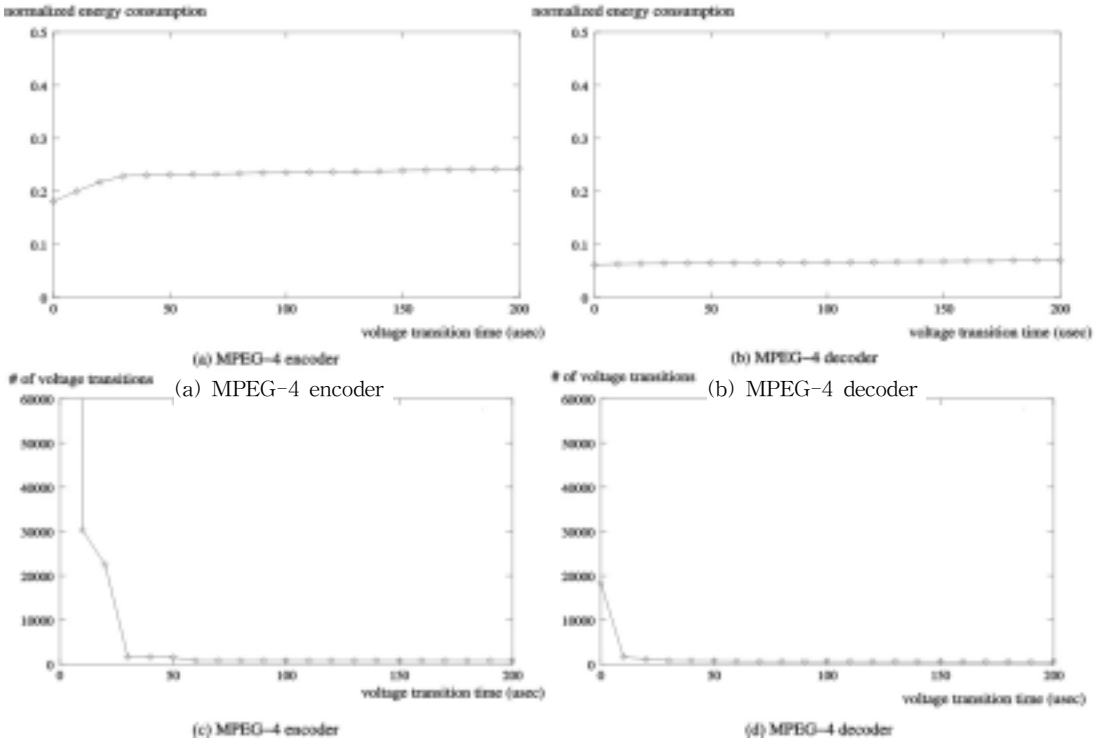

그림 7 자동 전압 조절기(AVS)의 전체 구조

### 4.2 모의 실험 결과

AVS가 어느 정도 전력 소모를 감소시킬 수 있는지에 대한 성능을 측정하기 위해 MPEG-4 비디오 부호기(encoder)와 복호기(decoder)를 가지고 실험하였다. 국내에서는 아직까지 상용화된 가변 전압 프로세서 칩과 전력 소모를 정밀하게 사이클 단위로 측정할 장비가 개발되지 않아서, 본 논문에서는 하드웨어적으로 전력 소

모를 측정하는 대신에 전력 소모 모의 실험기(simulator)를 제작하였다. 전력 소모 모의 실험기는 응용프로그램의 기계어 실행 코드와 실행 트레이서(trace)를 입력으로 받아 프로그램의 전력 소모 값을 계산한다. 모의 실험에서는 공정한 비교를 위해서 동적 전압 조정 기법을 사용하는 시스템과 사용하지 않는 시스템 모두 유휴 시간 동안에는 전력 소모를 줄이기 위해서 전력 차단(power-down) 모드로 바뀐다고 가정한다. 주어진 클록 주파수에 대한 공급전압은 $f_{CLK} = 1/T_D \propto (V_{DD}-V_T)^{\alpha}/V_{DD}$ [2]로부터 구해지는데, 이때 $V_{DD}$, $V_T$, $\alpha$는 각각 2.5V, 0.5V, 1.3으로 가정했다. 클록 속도와 공급 전압을 변경하는데 걸리는 부가 사이클인 $C_{VTO}$는 100Mhz 클록 주파수에서 0~200μsec에 해당하는 0~20,000 사이클로 가정되었다. 가변 전압 프로세서는 2.2장의 프로세서 I형처럼 클록 속도와 공급 전압을 변경하는데 걸리는 $C_{VTO}$ 사이클 동안 실행을 멈추고 전력 차단 모드로 진입한다고 가정하였다.

그림 8(a)와 8(b)는 AVS로 변환된 MPEG-4 부호기와 복호기의 전력 소모를 보여주고 있다. 여기에서는 원래의 프로그램을 동적 전압 조정을 사용하지 않는 시스템에서 실행시킬 때의 전력 소모를 1로 가정하고, AVS로 변환된 프로그램이 동적 전압 조정을 사용하는 시스템에서 실행될 때의 전력 소모는 이 값을 기준으로 정규화하였다. 또한 각 경우에 대해서 전력 차단 모드는 일반 모드가 소모하는 전력의 5%를 소모한다고 가정했다[1]. 모의 실험 결과, AVS로 변환된 MPEG-4 부호기와 복호기 프로그램은 원래 프로그램의 각각 25%와 7% 이내의 전력을 소모했다. 전력 소모 효율성에 있어서 두 가지 프로그램이 큰 차이를 보이는 이유는 이들의 시간적인 동작에 있어서 차이점 때문이다. 원래 프로그램을 수행했을 때 MPEG-4 복호기는 WCET와 ACET(average case execution time)이 큰 차이를 보이는 반면, MPEG-4 부호기는 상대적으로 WCET와 ACET의 차이가 적은데, 이는 MPEG-4 복호기의 원래 프로그램이 더 많은 유휴 시간을 가지고 있어서 전력 소모를 줄일 수 있는 여지가 MPEG-4 부호기에 비해서 더 많다는 것을 의미한다. 그림 8(c)와 8(d)는 프로그램 실행 중 전압 조정 코드에 의한 전압 변경 회수를 나타낸다. MPEG-4 부호기에서 $C_{VTO}$ < 3,000 사이클(=30μsec)인 범위에서는 전압 변경 회수가 급격하게 감소




(a) MPEG-4 encoder (b) MPEG-4 decoder

(c) MPEG-4 encoder (d) MPEG-4 decoder

그림 8 AVS에 의한 부호기와 복호기 프로그램의 정규화된 전력 소모 및 조정 회수

하며 전력 소모를 빨리 증가시키도록 만든다. 반면 MPEG-4 부호기와 복호기에 있어 $C_{VTO}$ > 5,000 사이클(=50μsec)인 범위에서는 전압 조정 회수가 거의 일정하게 유지되고 전력 소모도 크게 증가하지 않는다.

AVS의 성능을 나타내는 또다른 척도는 VSE 개수와 변환 시간이다. VSE 개수는 AVS로 변환된 프로그램 안에 얼마나 많은 전압 조정 코드가 삽입되어야 하는지를 나타내며, 인라인 확장(in-line expansion)에 의해서 전압 조정 코드가 삽입될 경우 전체 코드 크기가 얼마나 증가하느냐를 의미한다. MPEG-4 부호기와 복호기의 경우, $C_{VTO}$ > 5,000사이클에서 VSE의 개수는 20여개면 충분하므로, 전압 조정 코드에 의해서 전체 코드 크기가 별로 증가하지 않는다. 그 이유는 소수의 VSE가 전체 전력 소모의 감소의 대부분을 차지하기 때문이다. 만약 VSE의 개수가 너무 늘어날 경우에는 $C_{overhead}$ 값을 크게 하여 효과가 큰 VSE들만을 선별하여 전체 코드 크기의 증가를 줄일 수 있다. 또한 AVS가 컴파일시(compile-time)에 응용 프로그램을 변환하는데 걸리는 변환 시간도 MPEG-4 부호기와 복호기의 경우 100msec이하에 불과하였다.

본 실험에서는 또한 제안된 태스크내 전압 스케줄링 기법을 기존의 태스크간 전압 스케줄링 기법과 비교하였다. 1장에서 태스크내 스케줄링 기법의 필요성을 설명하기 위해서 사용된 표 1의 동영상 휴대 전화기의 태스크 집합에 대해서 전력 감소 효과를 비교하였다. 실험에서는 태스크간 스케줄링 기법으로 LPEDF 스케줄링 기법을 사용하였다[6].

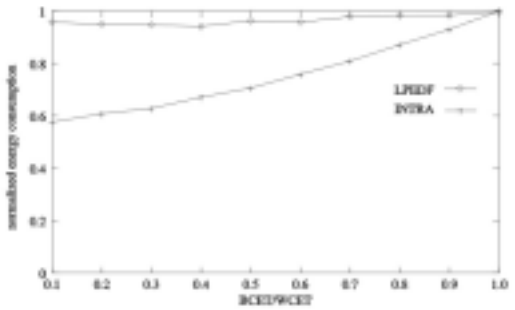



그림 9 태스크 내 스케줄링과 태스크간 스케줄링 기법의 비교

그림 9는 두 가지 기법에 대해서 각 태스크의 유휴시간을 나타내는 최소 실행시간(BCET)과 최악 실행시간(WCET)의 비율인 BCET/WCET의 값을 변화 시켜가며 전력 소모값을 측정한 결과를 보여주고 있다.[4] 전력 소모값은 전력 차단 모드만을 사용하는 기법에 대해 정규화하여 나타낸 것이다. BCET/WCET가 작을 수록 각 태스크가 가지는 유휴시간이 증가하여 태스크내 스케줄링 기법은 좋은 효과를 나타내고 있지만 LPEDF기법은 1장에서 설명한 것과 같이 큰 태스크가 발생시키는 유휴시간을 작은 태스크들이 사용하게 되어 그 효과가 크지 않은 것을 알 수 있다.

## 5. 결 론

본 논문에서는 저전력 경성 실시간 프로그램을 위한 태스크내 전압 스케줄링 기법을 제안하였다. 제안된 알고리즘은 최악 실행 시간 분석 도구를 사용하여 각 기본 블록의 잔여 최악 실행 사이클 수(RWEC)를 계산하고, 계산된 RWEC 정보를 바탕으로 클록 속도 및 공급 전압을 결정한다. 제안된 알고리즘을 사용하면 주어진 태스크들이 모두 마감 시간에 인접하여 실행을 마치게 되어 유휴 시간을 거의 발생시키지 않으며, 이에 따라서 클록 속도와 공급 전압을 가능한 한 낮게 조정할 수 있어서 전력 소모를 크게 줄일 수 있다. 이 알고리즘은 동적 전압 조정 기법(DVS)을 고려하지 않고 개발된 기존 응용프로그램에 태스크내 전압 스케줄링을 적용하는 데에 있어서, 반드시 필요하지만 매우 시간 소모적이고 복잡한 다음 두가지 과정, 즉 (1) 프로그램 코드 상에서 클록 속도 및 공급 전압이 조정될 수 있는 적당한 위치를 선택하는 과정, (2) 선택된 위치에 전압 조정 코드를 삽입하는 과정을 자동화할 수 있다. 따라서 프로그램 개발자의 관여 없이 전압 조정 코드를 자동으로 삽입하여 줌으로서, 프로그램 개발자가 동적 전압 조정에 대한 지식이 전혀 없이 가변 전압 프로세서 상에서 동적 전압 조정 기법을 사용할 수 있게 해준다.

또한 본 논문에서는 이를 바탕으로 AVS라는 자동화된 프로그램 변환 도구를 개발하였다. AVS는 DVS를 이용하지 않는 프로그램을 동일한 기능과 동일한 마감 시간 제약조건을 가지면서 DVS를 이용하는 저전력 프로그램으로 자동 변환해 준다. MPEG-4 부호기와 복호기를 사용한 실험 결과에서 AVS로 변환된 프로그램은 원래 프로그램에서보다 전력 소모 효율성을 4배에서 14배까지 향상시킬 수 있었다.

4) 태스크내 스케줄링 기법과 태스크간 스케줄링 기법은 태스크 집합의 특성에 따라 그 효과가 많이 달라지기 때문에 태스크들의 특성을 바꿔가며 비교하는 것이 필요하지만 본 논문에서는 표1의 태스크 집합만을 가정하고 실험을 실시하였으며 유휴시간의 비율을 바꿔가며 결과를 관찰하였다.

본 논문에서 제안된 태스크내 전압 스케줄링 기법은 전력 소모 효율을 좀 더 높이기 위해서 여러 가지 방향으로 확장될 수 있다. 예를 들어, VSE를 선택할 때 RWEC를 사용하였는데, ACET와 같은 다른 측정치를 사용하면서도 태스크의 마감 시간 제약조건을 보장하는 전압 스케줄링 기법을 연구하고 있는 중이다. 또한 프로그램이 지금까지 수행된 총 수행 시간과 같은 실행 시 정보(run-time information)를 이용하는 방법 등도 연구할 계획이다.

## 감사의 글

이 연구를 위해 연구장비를 지원하고 공간을 제공한 서울대학교 컴퓨터신기술공동연구소에 감사드립니다.

## 참 고 문 헌

[1] T. Burd and R. Broderson. Processor design for portable systems. *Journal of VLSI Signal Processing*, Vol. 13, No. 2, pp. 203-222, 1996.

[2] T. Sakurai and A. Newton. Alpha-power law MOSFET model and its application to CMOS inverter delay and other formulas. *IEEE Journal of Solid State Circuits*, Vol. 25, No. 2, pp. 584-594, 1990.

[3] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. of the 36th Annual Symposium on Foundations of Computer Science*, pp. 374-382, 1995.

[4] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava. Synthesis techniques for low-power hard real-time systems on variable voltage processor. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pp. 178-187, 1998.

[5] T. Okuma, T. Ishihara, and H. Yasuura. Real-time task scheduling for a variable voltage processor. In *Proc. of the 12th International Symposium On System Synthesis*, pp. 24-29, 1999.

[6] Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. of the 36th Design Automation Conference*, pp. 134-139, 1999.

[7] Y. Lee and C. M. Krishna. Voltage-clock scaling for low energy consumption in real-time embedded systems. In *Proc. of the 6th International Conference on Real-Time Computing Systems and Applications*, pp. 272-279, 1999.

[8] S. Lee and T. Sakurai. Run-time voltage hopping for low-power real-time systems. In *Proc. of the 37th Design Automation Conference*, pp. 806-809, 2000.

[9] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *Proc. of IEEE International Solid-State Circuits Conference*, pp. 294-295, 2000.

[10] M. Fleischmann. Crusoe power management: reducing the operating power with LongRun. In *Proc. of HotChips 12 Symposium*, 2000.

[11] C. A. Healy, D. B. Whalley, and M. G. Harmon. Integrating the timing analysis of pipelining and instruction caching. In *Proc. of the 16th IEEE Real-Time Systems Symposium*, pp. 288-297, 1995.

[12] S.-S. Lim, Y. H. Bae, G. T. Jang, B.-D. Rhee, S. L. Min, C. Y. Park, H. Shin, K. Park, and C. S. Kim. An accurate worst case timing analysis for RISC processors. *IEEE Transactions on Software Engineering*, Vol. 21, No. 7, pp. 593-604, 1995.

[13] Y. S. Li, S. Malik, and A. Wolfe. Cache modeling for real-time software: beyond direct mapped instruction caches. In *Proc. of the 17th IEEE Real-Time Systems Symposium*, pp. 254-263, 1996.

[14] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of International Symposium On Low Power Electronics and Design*, pp. 197-202, 1998.

신 동 군
정보과학회논문지 : 시스템 및 이론
제 28 권 제 10 호 참조

김 지 홍
정보과학회논문지 : 시스템 및 이론
제 28 권 제 10 호 참조

이 성 수
1991년 서울대학교 전자공학과 학사. 1993년 서울대학교 전자공학과 석사. 1998년 서울대학교 전기공학부 박사. 2000년 ~ 현재 이화여자대학교 정보통신학과 전임강사. 관심분야는 내장형 시스템, 저전력 시스템, MPEG-2, MPEG-4