실시간 시스템의 DMA I/O 요구를 위한 최악 시간 분석

(Worst Case Timing Analysis for DMA I/O Requests in Real-time Systems)

한 주 선 † 하 란 †† 민 상 렬 †††
(Joosun Hahn) (Rhan Ha) (Sang Lyul Min)

요 약 CPU의 수행과 병행하여 I/O가 수행되도록 DMA 방식을 채택한 실시간 시스템의 스케줄 가능성을 보장하기 위해서는 CPU 태스크 뿐만 아니라 I/O 요구의 스케줄 가능성도 반드시 검증되어야 한다. 본 논문에서는 CPU에게 최상위 우선순위가 할당된 고정우선순위 버스 프로토콜을 기반으로 CPU와 다수의 DMA 컨트롤러가 시스템 버스를 공유하는 환경에서 DMA I/O 요구의 최악 응답시간을 분석하는 기법을 제안한다. 제안하는 분석 기법의 첫 번째 단계에서는 CPU 상에서 수행 중인 각 CPU 태스크별로 최악 버스 요구 패턴을 구한다. 두 번째 단계에서는 이들 CPU 태스크의 최악 버스 요구 패턴을 모두 통합해 CPU 전체의 최악 버스 요구 패턴을 구한다. 최종 세 번째 단계에서는 CPU의 최악 버스 요구 패턴으로부터 DMA 컨트롤러의 버스 가용량을 구하고 DMA I/O 요구의 최악 응답시간을 산출한다. 모의 실험을 통해 제안하는 분석 기법이 일반적인 DMA 전송량에 대해 20% 오차 범위 이내에서 안전한 응답시간을 산출하며, DMA 전송량이 증가할수록 오차가 점차 감소함을 보였다.

키워드 : 실시간 시스템, DMA I/O, 최악 응답시간 분석, 고정우선순위 버스 프로토콜

Abstract We propose a technique for finding the worst case response time (WCRT) of a DMA request that is needed in the schedulability analysis of a whole real-time system. The technique consists of three steps. In the first step, we find the worst case bus usage pattern of each CPU task. Then in the second step, we combine the worst case bus usage patterns of CPU tasks to construct the worst case bus usage pattern of the CPU. This second step considers not only the bus requests made by CPU tasks individually but also those due to preemptions among the CPU tasks. Finally, in the third step, we use the worst case bus usage pattern of the CPU to derive the WCRT of DMA requests assuming the fixed-priority bus arbitration protocol. Experimental results show that overestimation of the DMA response time by the proposed technique is within 20% for most DMA request sizes and that the percentage overestimation decreases as the DMA request size increases.

Key words : real-time system, DMA I/O, timing analysis, fixed-priority bus arbitration protocol

1. 서 론

실시간 시스템(real-time system)의 가장 중요한 특징 중의 하나는 각 태스크에게 부여된 시간 제약 조건이다. 이 시간 제약 조건을 시스템 내의 모든 태스크들이 만족하는지 검증하기 위해 여러 스케줄 가능성 분석 기법들이 제시되었다[1-4]. 그러나 이들 연구를 통해 다양한 CPU 태스크 스케줄 가능성 조건이 제시된 반면, 근래에 이르기까지 I/O 요구의 스케줄 가능성에 대한 연구는 많지 않았다[5,6]. 최근 상당수의 실시간 시스템, 특히 내장형 실시간 시스템(embedded real-time system)에서 DMA(direct memory access) 방식의 I/O를 채택하고 있다. 이러한 실시간 시스템의 스케줄 가능성을 보장하기 위해서는 CPU 태스크뿐만 아니라 I/O 요구의 스케줄 가능성도 반드시 검증되어야 한다.

· 본 연구는 2004년도 두뇌한국21 사업에 의하여 지원되었음. 또한, 본 연구는 대학 IT 연구센터 육성 지원사업의 연구결과로서 HY-SDR 연구센터의 연구비 지원으로 수행되었음. 본 연구를 위해 연구 장비를 지원하고 공간을 제공한 서울대학교 컴퓨터연구소에 감사드립니다.

† 정 회 원 : 서울대학교 전기컴퓨터공학부
jshan@archi.snu.ac.kr
†† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
rhanha@cs.hongik.ac.kr
††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
symin@dandelion.snu.ac.kr

논문접수 : 2004년 5월 24일
심사완료 : 2004년 12월 1일

DMA I/O 요구의 시간 분석에 관한 기존의 연구로서, [6]은 사이클 스틸링(cycle-stealing) 방식으로 전송되는 DMA I/O 요구의 최악 응답시간을 예측하였다. [6]은 DMA I/O 요구의 응답시간을 DMA 전송이 수행되는 동안 CPU에서 수행되는 CPU 명령어들의 최악 수행시간의 합으로 정의하였다. 따라서 DMA I/O 요구의 최악 응답시간을 산출하기 위해서는 각 CPU 태스크의 모든 가능한 수행 경로를 열거하고 이들을 조합하여 최악 수행시간을 보이는 명령어 순서열을 구하는 것이 필요하다. 그러나 각 CPU 태스크의 모든 가능한 수행 경로를 열거하는 것이 불가능하기 때문에, 실제로는 각 CPU 태스크의 임의의 한 수행 경로를 조합하여 최악 수행시간을 보이는 명령어 순서열을 동적 프로그래밍 기법을 이용하여 산출하였다. 따라서 [6]은 (1) 임의의 한 CPU 수행 트레이스에 대해 DMA I/O 요구의 최악 응답시간을 분석함으로써 최악 시간 분석임을 보장할 수 없고, (2) 각 명령어의 최악 수행시간을 단순히 합산함으로써 최근의 프로세서에서는 일반화된 캐쉬 메모리의 영향을 고려하지 못했으며, (3) 하나의 DMA I/O 요구만을 대상으로 하였다는 취약점을 가지고 있다.

본 논문에서는 CPU와 다수의 DMA 컨트롤러가 시스템 버스를 공유하는 환경에서 DMA I/O 요구의 최악 응답시간을 예측하는 기법을 제안한다. 이 때, 시스템 버스는 CPU에게 최상위 우선순위가 할당된 고정우선순위 버스 프로토콜로 동작함을 가정한다. 이와 같은 버스 프로토콜 환경에서 DMA 요구의 시간 분석을 어렵게 하는 요소는 CPU와 DMA 컨트롤러의 버스 요구가 충돌할 때마다 DMA 전송이 지연된다는 점이다. 이러한 문제를 해결하기 위해, 제안하는 분석 기법에서는 CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$와 DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수 $g_{dma}(t)$를 이용해 DMA I/O 요구의 최악 응답시간을 산출한다.

본 논문은 (1) 최악 시간 분석을 통해 DMA I/O 요구의 최악 응답시간을 산출하였고, (2) 최악 시간 분석 시 명령어 캐쉬의 영향을 고려했으며, (3) 다수의 DMA I/O 요구에 대해서도 적용 가능하다는 점에서 기존의 연구와 차별화된다.

2. 모델 및 정의

2.1 시스템 모델과 태스크 모델

본 논문에서 분석의 대상이 되는 시스템 모델을 그림 1에 나타내었다. 시스템은 CPU와 주 메모리, 그리고 다수의 DMA 컨트롤러로 구성되며 시스템 버스를 공유하고 있다. CPU에는 캐쉬가 장착되어 있어서 캐쉬 접근 실패가 발생하는 경우에만 시스템 버스 요구를 생성한다. 이 중, 명령어 참조의 캐쉬 접근 실패로 인한 영향을 분석하기 위해 데이타 참조의 캐쉬 접근 실패로 인한 영향을 배제하고자 모든 데이타 참조는 캐쉬 접근에 성공한다고 가정하였다. 다수의 DMA 컨트롤러는 시스템 버스의 사용을 위해 서로 간에 혹은 CPU와 경쟁한다. 이 때, CPU와 각 DMA 컨트롤러에는 고정우선순위가 할당되어 있어서 가장 높은 우선순위를 가진 장치에게 시스템 버스 사용이 허용되는데, 모든 장치 중에 CPU에게 최상위 우선순위가 할당된다고 가정하였다. 즉, DMA 컨트롤러는 CPU 혹은 상위 우선순위를 가지는 다른 DMA 컨트롤러로부터 버스 요구가 없을 경우에만 자신의 I/O를 수행할 수 있다.

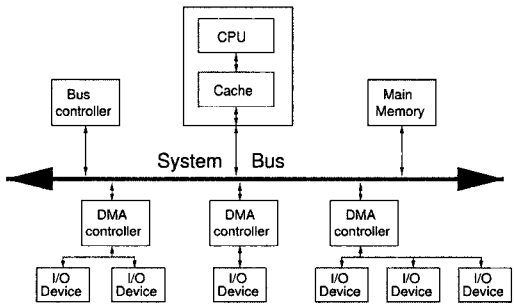


그림 1 시스템 모델

본 논문에서 시스템 버스에 대해 설정한 가정은 다음과 같다.

(1) 버스 사이클 시간 t^{bus}_{cycle}은 CPU 사이클 시간 t^{cpu}_{cycle}의 정수배이다.

(2) CPU는 캐쉬 접근 실패가 발생하는 경우에만 시스템 버스 요구를 생성하며, 이를 위해 $t^{bus}_{cache\ miss}$ bus cycle 또는 $t^{cpu}_{cache\ miss}(=\frac{t^{bus}_{cycle}}{t^{cpu}_{cycle}}\times t^{bus}_{cache\ miss})$ CPU cycle의 시간이 필요하다.

(3) 시스템 버스의 마스터링을 위해 Δ bus cycle의 시간이 필요하다.

CPU 태스크 집합은 n_{cpu}개의 주기적 태스크 τ^{cpu}_1, τ^{cpu}_2, …, $\tau^{cpu}_{n_{cpu}}$로 구성된다. 각 CPU 태스크 τ^{cpu}_i는 (주기 T^{cpu}_i, 최악 수행시간 C^{cpu}_i, 마감시간 D^{cpu}_i)로 정의되며, 고정우선순위 스케줄링 정책에 의해 스케줄된다. DMA 컨트롤러 i의 I/O 요구는 주기적인 DMA 태스크 τ^{dma}_i (주기 T^{dma}_i, 마감시간 D^{dma}_i, 전송량 S^{dma}_i)로 정의되며, $i<j$인 경우 DMA 컨트롤러 i의 τ^{dma}_i가 DMA 컨트롤러 j의 τ^{dma}_j보다 높은 우선순위를 가진다.

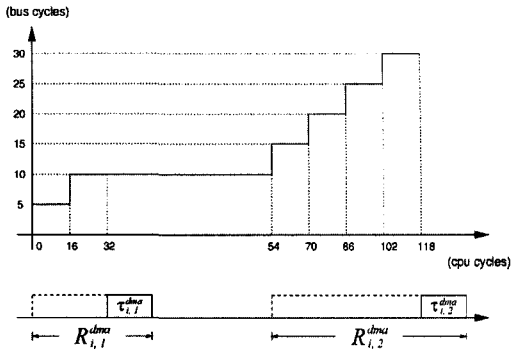
2.2 최대 도착 함수 $f_{cpu}(t)$와 최소 서비스 함수 $g_{dma}(t)$

CPU의 버스 요구에 대한 최대 도착 함수(maximum arrival function) $f_{cpu}(t)$는 길이 t의 모든 구간 $[x, x+t]$에 대해서 CPU로부터 생성되는 최대 버스 요구량으로 정의된다. 이 정의로부터, $f_{cpu}(t)$는 길이 t의 모든 CPU 수행 구간에서 생성되는 버스 요구량의 상한이 됨을 알 수 있다. DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수(minimum service function) $g_{dma}(t)$는 길이 t의 모든 구간 $[y, y+t]$에 대해서 DMA 컨트롤러들에게 보장되는 최소 버스 가용량으로 정의된다. 이 정의로부터, $g_{dma}(t)$ 만큼의 버스 가용량이 특히 DMA 수행 구간 $[0, t]$에 제공되는 경우에 DMA I/O 요구의 최악 응답시간이 산출됨을 알 수 있다.

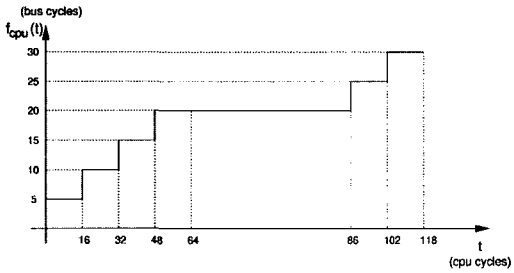
$f_{cpu}(t)$와 $g_{dma}(t)$의 관계를 설명하기 위해, CPU로부터 생성되는 버스 요구의 최악 도착 함수(worst case arrival function)의 한 예를 그림 2(a)에 나타내었다. 최악 도착 함수는 CPU가 일련의 명령어를 수행하는 동안 최대로 생성 가능한 버스 요구량을 분석해서 구할 수 있다. 즉, 이상적인 파이프라이닝(pipelining)에 의해 매 CPU cycle마다 명령어 페치(fetch)를 위해 캐쉬 접근이 시도되고, 그 결과 캐쉬 접근 실패가 발생하면 시스템 버스 요구를 생성하는 경우에 최대 버스 요구량이 유발된다. 그림 2(a)는 2번의 캐쉬 접근 실패, 22번의 캐쉬 접근 성공, 4번의 캐쉬 접근 실패가 발생하는 명령어 수행 예를 나타낸 것이며, 한 번의 캐쉬 접근 실패를 서비스하기 위해 5 bus cycle 또는 15 CPU cycle이 소요된다고 가정하였다.

이 때, 임의의 DMA 태스크 τ_i^{dma}가 각각 시점 0과 시점 54에 도착하는 경우를 살펴보자. 그림 2(a)에서 보듯이, DMA 태스크는 시점 0보다 시점 54에서 CPU의 버스 요구량이 폭주할수록 최악의 응답시간을 보임을 알 수 있다. 따라서 길이 t의 수행 구간을 모든 시점에 대해 고려해 CPU의 최대 버스 요구량, 즉 최대 도착 함수 $f_{cpu}(t)$를 결정해야, 최악 응답 시간을 보이는 시점에 도착한 DMA 태스크에게 제공되는 최소 버스 가용량, 즉 최소 서비스 함수 $g_{dma}(t)$를 구할 수 있다.

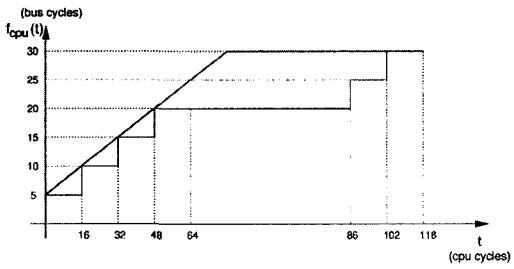
그림 2(b)는 이 예에 대한 최대 도착 함수 $f_{cpu}(t)$를 나타낸 것이다. 최대 도착 함수는, 이 예를 수행하는 동안, 길이 t CPU cycle인 어떠한 수행 구간에서도 $f_{cpu}(t)$ bus cycle 이상의 버스 사용량을 요구하지 않음을 의미한다. 그림 2(c)는 계단식 함수인 최대 도착 함수로부터 2-구간 선형 함수(2-piecewise linear function)를 도출한 것이다. 제안하는 분석 기법에서 최대 도착 함수는 그림과 같은 2-구간 선형 함수의 형태를 유지할 것이다.



(a) CPU 버스 요구의 최악 도착 함수



(b) CPU 버스 요구의 최대 도착 함수



(c) CPU 버스 요구의 최대 도착 함수 (2-구간 선형 함수)

그림 2 CPU의 최대 도착 함수

3. MAFA 자료구조 및 접속 연산

CPU의 최대 도착 함수를 산출하기 위해서는 먼저 CPU 태스크 별로 캐쉬 접근 실패를 분석하여 각 CPU 태스크의 버스 요구에 대한 최대 도착 함수를 산출하는 것이 필요하다. 이는 CPU 태스크에서 캐쉬 접근 실패가 발생하는 경우에만 시스템 버스 요구를 생성하여 DMA 전송을 지연시키기 때문이다. 본 장에서는 CPU 태스크의 최대 도착 함수를 산출하는 과정에서 형성되는 중간 정보들을 유지하기 위한 MAFA(maximum arrival function abstraction) 자료구조와 두 MAFA

자료구조 간의 접속 연산(concatenation operation)을 정의한다.

3.1 MAFA 자료구조

MAFA 자료구조는 프로그램 구문 내의 각 수행 경로에 연관되어 있으면서, 해당 구문이 선행하는 또는 후행하는 구문과 접속되어 보다 긴 수행 구간의 구문을 형성할 때, 접속되어 형성된 구문의 최대 도착 함수를 산출하는 데에 필요한 정보를 담고 있다. 그림 3에서 보듯이, MAFA 자료구조는 $f(t)$, $z(t)$, $z^d(t)$로 표시되는 3개의 도착 함수와 first_reference, last_reference로 지칭되는 2개의 메모리 참조 정보로 구성된다. 이 중, $f(t)$는 분석의 목표가 되는 최대 도착 함수이고, $z(t)$, $z^d(t)$, first_reference, last_reference는 인접한 블록 간의 접속 시에 $f(t)$를 구하기 위해 사용되는 정보이다.

$z(t)$는 길이 t의 수행 구간 $[t^0, t^0+t]$에서 CPU 태스크로부터 생성되는 최대 버스 요구량을 나타내는 함수이다. 여기서, t^0은 해당 경로의 수행이 시작되는 시점이다. $z^d(t)$는, $z(t)$의 대응 함수(dual function)로서, 길이 t의 수행 구간 $[t^d-t, t^d]$에서 CPU 태스크로부터 생성되는 최대 버스 요구량을 나타내는 함수이다. 여기서, t^d는 해당 경로의 수행이 완료되는 시점이다. 각 도착 함수는 2-구간 선형 함수로서 $(n_{initial}, n_{max}, t_{initial}, t_{max})$라는 4개의 점 정보로 표시된다.

first_reference는 각 캐쉬 블록에 대해 해당 수행 경로 상에서 최초로 참조되는 메모리 블록의 주소 정보를 가지고 있다. 이들 메모리 참조는 이전에 적재되어 있던 캐쉬 내용에 따라 캐쉬 접근 성공/실패가 결정되는, 즉 캐쉬 접근 실패로 잠정 간주되는 참조들이다. last_reference는 각 캐쉬 블록에 대해 해당 수행 경로 상에서 최후로 참조되는 메모리 블록의 주소 정보를 가지고 있다. 이 정보는 후행하는 구문의 first_reference, 즉 캐쉬 접근 실패로 잠정 간주되었던 메모리 참조의 캐쉬 접근 성공/실패를 결정하는 역할을 한다.

분석의 최소 단위인 기본 블록(basic block)의 MAFA 자료구조는 다음과 같이 정의된다. 그림 4에서 보듯이, 도착 함수 $f(t)$, $z(t)$, $z^d(t)$는 모두 같은 형태를 지닌다. 이는 기본 블록에 대해서는 모든 명령어 참조를 잠정적으로 캐쉬 접근 실패로 간주하고 모든 캐쉬 접근 실패가 $t_{initial}$ 시점에 발생한다고 가정하기 때문이다. 만일 기본 블록에서 $I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow I_3$ 명령어가 참조되고 I_0와 I_2는 캐쉬 블록 c_0에, I_1와 I_3는 캐쉬 블록 c_1에 사상된다면, c_0에 대한 first_reference

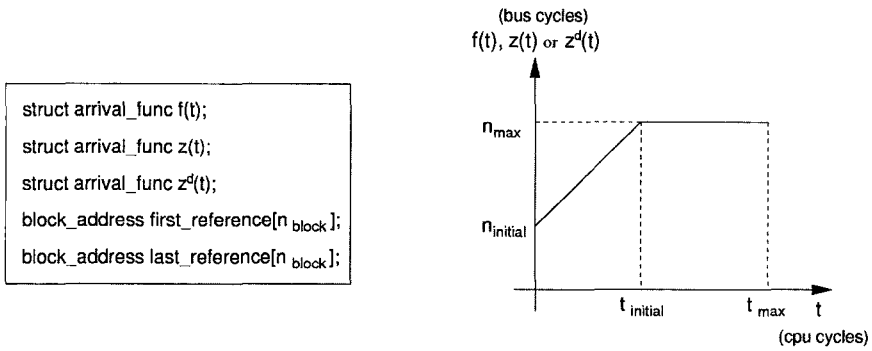


그림 3 MAFA 자료구조

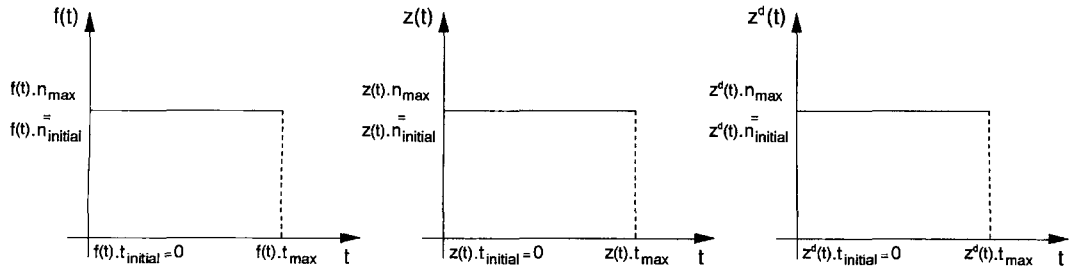


그림 4 기본 블록의 도착 함수

와 last_reference는 각각 I_0과 I_2의 메모리 주소가 되고 c_1에 대한 first_reference와 last_reference는 각각 I_1과 I_3의 메모리 주소가 된다.

3.2 접속 연산

각 기본 블록에 연관되어 있는 MAFA 자료구조를 바탕으로, 분석은 구문에 따른 접속식에 의해 계층적으로 진행된다. 다음은 순차 구문, 조건 구문, 순환 구문에 대한 접속식이다.

(순차 구문: S ⇒ S_1; S_2)

$$M(S) = M(S_1) \uplus M(S_2)$$

(조건 구문: S ⇒ if (exp) then S_1 else S_2)

$$M(S) = (M(\text{exp}) \uplus M(S_1)) \cup (M(\text{exp}) \uplus M(S_2)) = M(\text{exp}) \uplus (M(S_1) \cup M(S_2))$$

(순환 구문: S ⇒ while (exp) S_1)

$$M(S) = (\cup_{i=N_l}^{N_u} (\uplus_{j=1}^{i} (M(\text{exp}) \uplus M(S_1)))) \uplus M(\text{exp})$$

여기서, N_u와 N_l은 각각 순환 횟수의 상한과 하한이다.

위의 접속식에서 $M(S)$는 구문 S 내의 각 수행 경로에 연관되어 있는 MAFA 자료구조의 집합으로, 두 MAFA 자료구조 집합 간의 연산 ⊎는 다음과 같이 정의된다.

$$M(S_1) \uplus M(S_2) = \{ m_1 \oplus m_2 \mid m_1 \in M(S_1),\ m_2 \in M(S_2) \}$$

여기서, m_1과 m_2는 각각 MAFA 자료구조이며 ⊕는 m_1과 연관된 수행 경로와 m_2와 연관된 수행 경로의 순차적인 수행을 모델링하는 접속 연산이다. 이 접속 연산의 절차를 그림 5와 그림 6에서 설명하였다. 접속 연산은 두 MAFA 자료구조 m_1과 m_2를 입력받고, 접속된 수행 경로에 연관되는 MAFA 자료구조 m을 출력한다.

접속 연산의 단계 1에서는 m의 first_reference와 last_reference를 구한다. 이 과정에서, m_1의 last_reference 정보와 m_2의 first_reference 정보가 일치하면 캐쉬 접근 실패로 잠정 분석되었던 m_2의 first_reference가 캐쉬 접근 성공으로 판명된 것이다. 캐쉬 접근 성공으로 판명된 참조가 n_{hits} 개라고 할 때, 이는 m_2의 $f(t)$, $z(t)$, $z^d(t)$에 n_{hits} 번의 캐쉬 접근 실패를 서비스하기 위한 비용이 과다하게 포함되어 있음을 의미하므로 접속 연산의 단계 2에서 m_2의 $f(t)$, $z(t)$, $z^d(t)$를 그림과 같은 방식으로 보정한다.

접속 연산의 단계 3에서는 m의 $z(t)$와 $z^d(t)$를 구한다. 그림에서 보듯이, m의 $z(t)$는 m_1의 $z(t)$ 뒤에 m_2의 $z(t)$를 연결하는 방식으로 함수를 형성한다. 이는 접속 연산이 m_1과 연관된 수행 경로와 m_2와 연관된 수행 경로의 순차적인 수행을 모델링하기 때문이다. 반면, m의 $z^d(t)$는 m_2의 $z^d(t)$의 뒤에 m_1의 $z^d(t)$를 연결하는 방식으로 함수를 형성한다.

접속 연산의 단계 4와 단계 5에서는 m의 최대 도착 함수 $f(t)$를 구한다. 이 때, 길이 t의 수행 구간은 ① m_1의 영역에 형성되거나, ② m_2의 영역에 형성되거나, ③ m_1과 m_2의 영역에 걸쳐서 형성될 수 있다. 먼저, 단계 4는 ③의 경우에 대해 도착 함수의 상한을 구하는 과정이다. m_1의 $z^d(t)$는 m_1의 말미에서 구한 버스 요구의 도착 함수이고 m_2의 $z(t)$는 m_2의 초미에서 구한 버스 요구의 도착 함수이므로, 두 함수를 그림과 같이 접속함으로써 m_1과 m_2의 영역에 걸쳐서 형성되는 길이 t의 수행 구간에 대해 도착 함수의 상한을 구할 수 있다. 그림의 예는 m_2의 $z(t)$가 더 급격히 증가하는 경우로 3-구간 선형 함수 $f(t)$는

$$f(t) = \begin{cases} \beta t + m_1.z^d(t).n_{initial} + m_2.z(t).n'_{initial} \\ \quad \text{if } 0 \le t \le m_2.z(t).t'_{initial} \\ \alpha t - \alpha m_2.z(t).t'_{initial} + m_1.z^d(t).n_{initial} \\ \quad + m_2.z(t).n'_{\max} \\ \quad \text{if } m_2.z(t).t'_{initial} \le t \le m_1.z^d(t).t_{initial} \\ \quad + m_2.z(t).t'_{initial} \\ m_1.z^d(t).n_{\max} + m_2.z(t).n'_{\max} \\ \quad \text{if } m_1.z^d(t).t_{initial} + m_2.z(t).t'_{initial} \\ \quad \le t \le m_1.z^d(t).t_{\max} + m_2.z(t).t'_{\max} \end{cases}$$

가 된다. 반면, m_1의 $z^d(t)$가 더 급격히 증가하는 경우에 3-구간 선형 함수 $f(t)$는

$$f(t) = \begin{cases} \alpha t + m_1.z^d(t).n_{initial} + m_2.z(t).n'_{initial} \\ \quad \text{if } 0 \le t \le m_1.z^d(t).t_{initial} \\ \beta t - \beta m_1.z^d(t).t_{initial} + m_1.z^d(t).n_{initial} \\ \quad + m_2.z(t).n'_{\max} \\ \quad \text{if } m_1.z^d(t).t'_{initial} \le t \le m_1.z^d(t).t_{initial} \\ \quad + m_2.z(t).t'_{initial} \\ m_1.z^d(t).n_{\max} + m_2.z(t).n'_{\max} \\ \quad \text{if } m_1.z^d(t).t_{initial} + m_2.z(t).t'_{initial} \\ \quad \le t \le m_1.z^d(t).t_{\max} + m_2.z(t).t'_{\max} \end{cases}$$

가 된다. 이 때, 위의 두 식에서 $\alpha = (m_1.z^d(t).n_{\max} - m_1.z^d(t).n_{initial}) / (m_1.z^d(t).t_{initial})$이고 $\beta = (m_2.z(t).n'_{\max} - m_2.z(t).n'_{initial}) / (m_2.z(t).t'_{initial})$로서, 각각 m_1의 $z^d(t)$와 m_2의 $z(t)$의 기울기를 의미한다.

단계 5에서는 단계 4에서 구한 3-구간 선형 함수 $f(t)$, m_1의 $f(t)$, m_2의 $f(t)$를 모두 포함하면서 이들에 가장 근사하는 2-구간 선형 함수를 m의 $f(t)$로 산출한다. 이 마지막 단계가 필요한 이유는 길이 t의 수행 구간이 ① m_1의 영역에 형성되거나 ② m_2의 영역에 형성되는 경우 m의 $f(t)$가 m_1의 $f(t)$ 또는 m_2의 $f(t)$의 값을 각각 가질 것이기 때문이다.

concatenate
Inputs: MAFA m_1, m_2
Outputs: MAFA m

1. Compute m's first_reference, last_reference and n_{hits}.

```
n_hits = 0;
for (i = 0; i < n_blocks; i++) {
    if (m1.first_reference[i] != NULL)
        m.first_reference[i] = m1.first_reference[i];
    else
        m.first_reference[i] = m2.first_reference[i];
    if (m2.last_reference[i] != NULL)
        m.last_reference[i] = m2.last_reference[i];
    else
        m.last_reference[i] = m1.last_reference[i];
    if (m1.last_reference[i] != NULL) && (m2.first_reference[i] != NULL)
        if (m1.last_reference[i] == m2.first_reference[i])
            n_hits++;
}
```

2. Update m_2's $z(t)$, $z^d(t)$ and maximum arrival function $f(t)$.

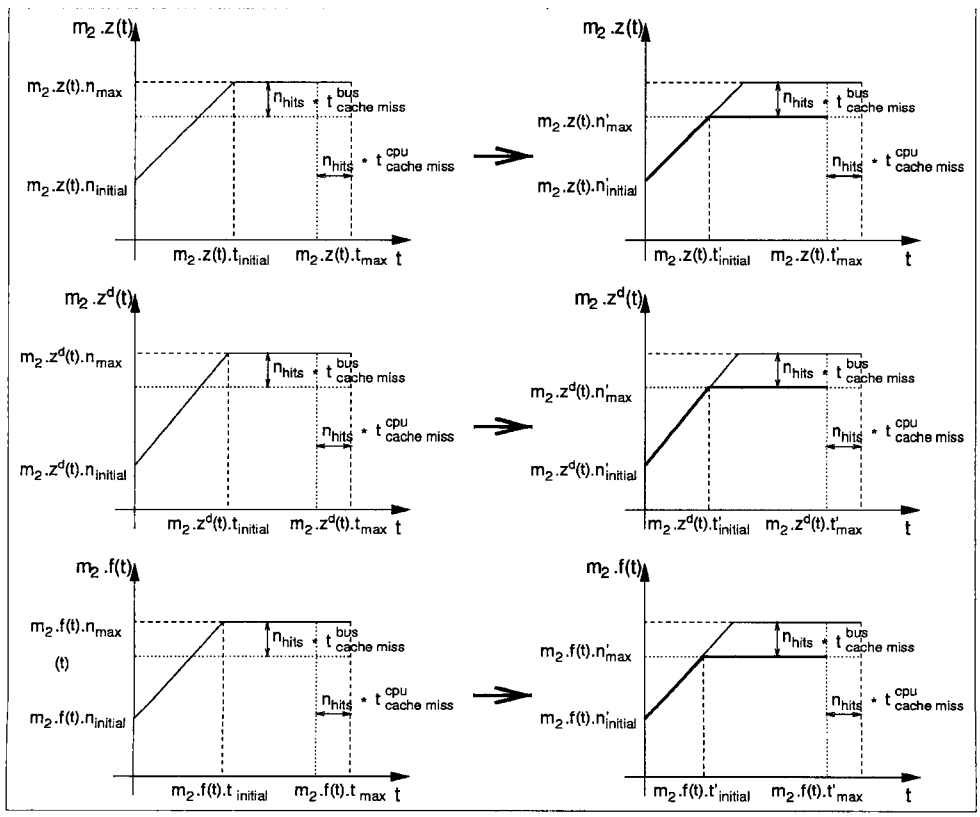
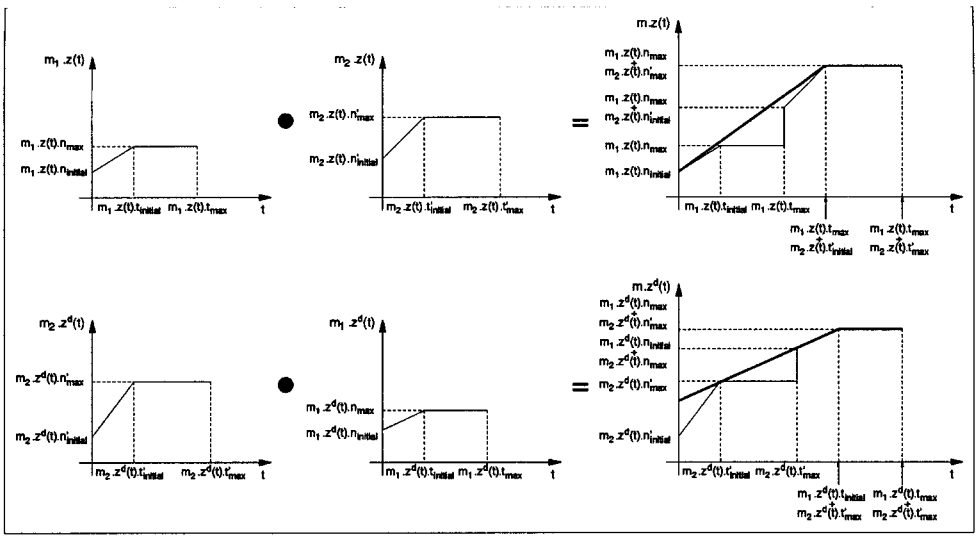
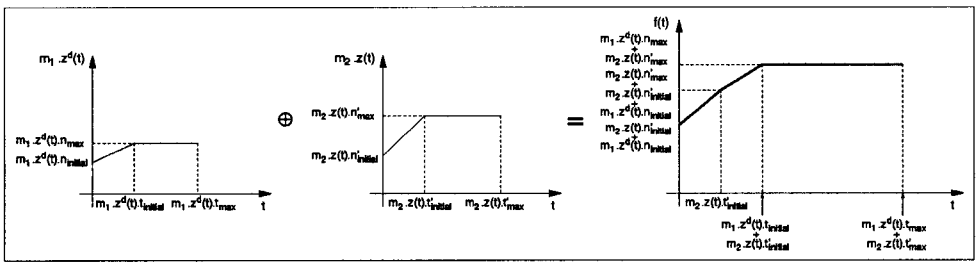


그림 5 접속 연산의 절차

3. Construct m's $z(t)$ and $z^d(t)$.



4. Construct 3-piecewise linear function from m_1's $z^d(t)$ and m_2's updated $z(t)$.



5. Construct m's maximum arrival function $f(t)$.

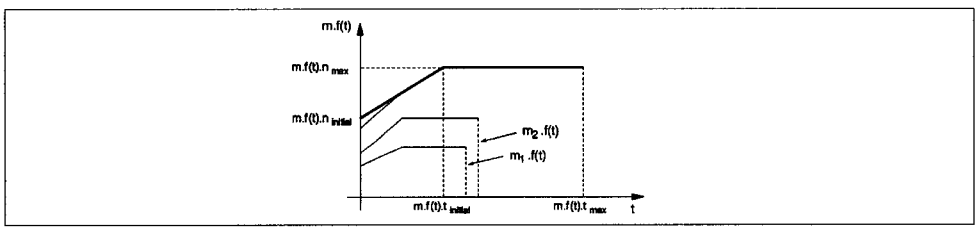


그림 6 접속 연산의 절차(계속)

4. 최악 응답시간 분석 기법

제안하는 최악 응답시간 분석 기법은 다음의 세 단계로 구성된다. 첫 번째 단계에서는 CPU에서 수행하는 각 CPU 태스크 τ_i^{cpu}의 버스 요구에 대한 최대 도착 함수 $\tau_i^{cpu}.f(t)$를 구한다. 두 번째 단계에서는 이들 CPU 태스크의 최대 도착 함수를 모두 통합해 CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$를 구한다. 이렇게 산출된 $f_{cpu}(t)$는 길이 t의 수행 구간에서 CPU에 의해 DMA 전송이 지연되는 시간의 상한이 된다. 따라서 세 번째 단계에서는 DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수 $g_{dma}(t) = t - f_{cpu}(t)$를 구하고 DMA 태스크 τ_i^{dma}의 최악 응답시간을 산출한다.

4.1 CPU 태스크 τ_i^{cpu}의 최대 도착 함수 $\tau_i^{cpu}.f(t)$의 산출

CPU 태스크 τ_i^{cpu}의 최대 도착 함수 $\tau_i^{cpu}.f(t)$의 산출

과정은 다음과 같다.

(1) 우선 각 기본 블록에 대해, 모든 명령어 참조를 캐쉬 접근 실패로 간주하고 최대 도착 함수를 정의한다.

(2) 이들을 구문 구조에 따라 인접한 블록끼리 계층적으로 접속해가면서 보다 긴 수행 구간에 대해 최대 도착 함수를 구한다. 이 때, 캐쉬 접근 실패로 잠정 간주되었던 참조들에 대해 개선된 캐쉬 접근 성공 정보가 반영된다.

(3) 최종 결과로 τ_i^{cpu}의 전체 수행 구간에 대해 최대 도착 함수를 구할 수 있다.

이 때, 인접한 블록끼리 계층적으로 접속해가는 과정에서 형성되는 중간 블록들에 대한 최대 도착 함수를 유지하기 위해 MAFA 자료구조와 두 인접한 블록의 MAFA 자료구조 간의 접속을 위해 접속 연산을 이용한다.

4.2 CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$의 형성

그림 7은 세 CPU 태스크 τ_1^{cpu}, τ_2^{cpu}, τ_3^{cpu}의 최대 도착 함수를 통합해 $f_{cpu}(t)$를 형성하는 예를 보여주고 있다. 그림에서 보듯이, 각 태스크의 도착 주기마다 $\tau_i^{cpu}.f(t)$의 기울기가 급한 순서대로 선택해 연접한다. 이 때, 태스크 간의 최악 선점 상황을 고려해 선점 비용으로 생성되는 버스 요구량 $\Delta_{preemption}(t)$도 부가시킨다. 부가적인 선점 비용 $\Delta_{preemption}(t)$란, 태스크가 선점형 스케줄링 방식에 의해 스케줄될 때 태스크 간 캐쉬 간섭에 의해 부가적으로 발생하는 캐쉬 접근 실패를 서비스하기 위해 필요한 버스 요구량을 의미한다. 즉, 한 태스크가 캐쉬에 적재해 놓은 메모리 블록이 선점되어 있는 동안 다른 태스크에 의해 다른 메모리 블록으로 치환되는 경우, 선점된 태스크가 수행을 재개했을 때 부가적으로 캐쉬 접근 실패를 겪게 된다. $\Delta_{preemption}(t)$는 선형 계획법(linear programming)을 이용해 구할 수 있다[7,8].

이와 같이 산출된 통합 함수를 $CPU.f(t)$라고 할 때, $CPU.f(t)$를 bus cycle 단위로 정규화시켜 $f_{cpu}(t)$를 구한다. 이 때, CPU와 DMA 컨트롤러 간의 두 번의 버스 마스터링을 위해 2Δ bus cycle이 소요된다는 점을 고려한다. 또한, 길이 t bus cycle인 수행 구간에서 제공되는 최대 버스 가용량은 t bus cycle이라는 점도 고려한다.

$$f_{cpu}(t) = \min\left(t, \left\lceil \frac{CPU.f\left(t \times \frac{t_{cycle}^{bus}}{t_{cycle}^{cpu}}\right)}{t_{cache\ miss}^{bus}} \right\rceil \times \left(t_{cache\ miss}^{bus} + 2\Delta\right)\right)$$

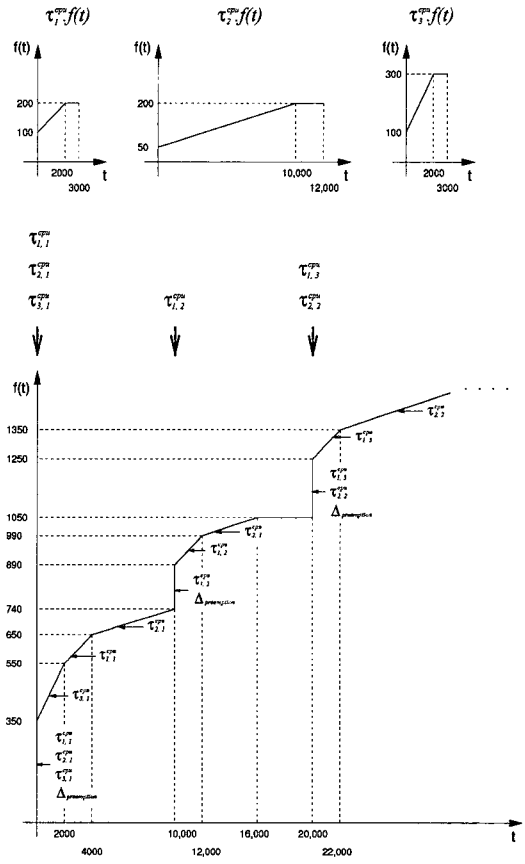


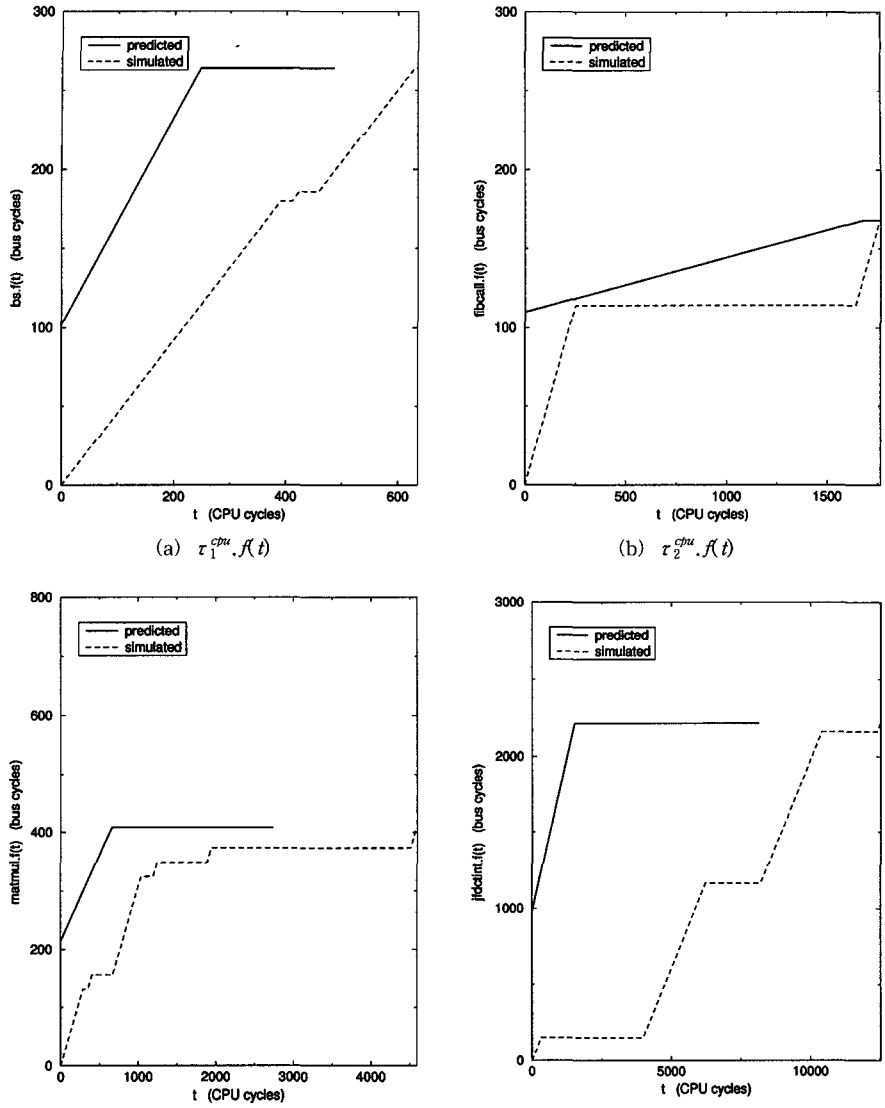
그림 7 각 CPU 태스크의 최대 도착 함수 통합

4.3 DMA 태스크 τ_i^{dma}의 최악 응답시간 R_i^{dma}의 산출

CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$는 길이 t의 수행 구간에서 CPU에 의해 DMA 전송이 지연되는 시간의 상한이 된다. 따라서 DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수는 $g_{dma}(t) = t - f_{cpu}(t)$로 구할 수 있다. 그리고 $g_{dma}(t)$의 역함수 $g_{dma}^{-1}(t)$를 이용해 DMA 태스크 τ_i^{dma}의 최악 응답시간을 산출할 수 있다.

$$R_i^{dma} = g_{dma}^{-1}\left(S_i^{dma} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{dma}}{T_j^{dma}} \right\rceil S_j^{dma}\right)$$

여기서, $g_{dma}^{-1}(s) = \min\{t \mid s = g_{dma}(t)\}$이며 $hp(i)$는 τ_i^{dma}보다 상위 우선순위를 가지는 DMA 태스크들의 집합을 의미한다. 위의 재귀식을 다음과 같이 반복적인 방법으로 풀어서 수렴하는 $R_i^{dma,n} = R_i^{dma,n+1}$를 DMA 태스크 τ_i^{dma}의 최악 응답시간으로 산출한다. 이와 같이 산출된 최악 응답시간에는 CPU 및 상위 우선순위 DMA 컨트롤러에 의해 τ_i^{dma}의 DMA 전송이 지연되는



(a) $\tau_1^{cpu}.f(t)$ (b) $\tau_2^{cpu}.f(t)$

(c) $\tau_3^{cpu}.f(t)$ (d) $\tau_4^{cpu}.f(t)$

그림 8 CPU 태스크의 최대 도착 함수

시간이 모두 포함되어 있다.

$$R_i^{dma,0} = g_{dma}^{-1}(S_i^{dma})$$

$$R_i^{dma,1} = g_{dma}^{-1}(S_i^{dma} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{dma,0}}{T_j^{dma}} \right\rceil S_j^{dma})$$

$$\vdots$$

$$R_i^{dma,k+1} = g_{dma}^{-1}(S_i^{dma} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{dma,k}}{T_j^{dma}} \right\rceil S_j^{dma})$$

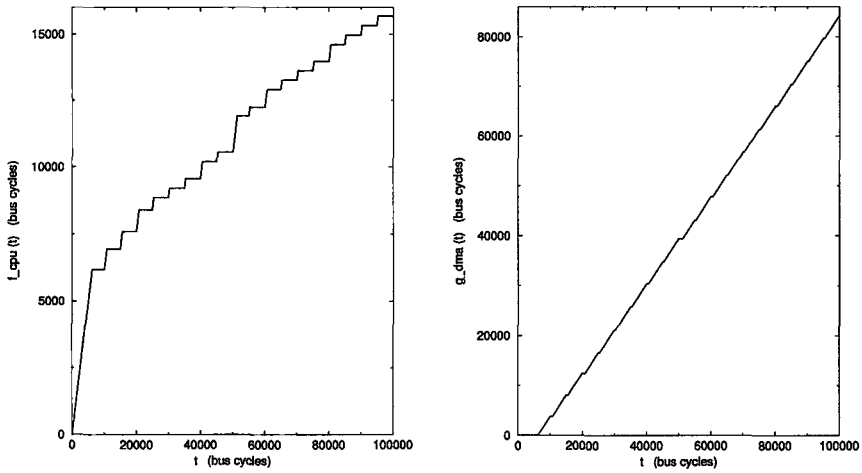
5. 성능 평가

제안하는 분석 기법의 성능 평가를 위해, 소정의 CPU 태스크 집합에 대해 제안하는 분석 기법으로 예측한 DMA 태스크의 최악 응답시간과 모의실험으로 산출한 응답시간을 비교하였다. 실험에 사용된 대상 시스템

표 1 CPU 태스크 집합

(단위: CPU cycle)

CPU 태스크	주 기	WCET	WCRT	명령어 블록수
τ_1^{cpu}	10000	635	635	44 블록
τ_2^{cpu}	40000	1763	2482	28 블록
τ_3^{cpu}	100000	4783	7709	68 블록
τ_4^{cpu}	200000	13376	24263	369 블록



(a) $f_{cpu}(t)$ (b) $g_{dma}(t)$

그림 9 CPU의 최대 도착 함수와 DMA 컨트롤러의 최소 서비스 함수

은 IDT사의 IDT7RS383 실험보드로서, 20 MHz R3000 RISC CPU와 R3010 FPA(floating point accelerator)가 장착되어 있다. 명령어 캐쉬는 16 Kbyte 크기의 직접 사상 캐쉬이며 한 캐쉬 블록의 크기는 4 byte이다. 명령어 캐쉬 접근 실패로 인한 영향을 분석하고자 모든 데이타 참조는 캐쉬 접근에 성공하도록 설정하였다. 그리고 캐쉬 접근 실패를 서비스하기 위해 소요되는 시간은 6 bus cycle 또는 12 CPU cycle로, 버스 마스터를 전환하기 위해 소요되는 시간은 1 bus cycle로 설정하였다. CPU 태스크 집합은 4개의 태스크로 구성되며 각 태스크의 특성은 표 1과 같다. 각 태스크에는 주기에 따라 τ_1^{cpu}이 최상위 우선순위를 가지고 τ_4^{cpu}가 최하위 우선순위를 가지도록 고정우선순위를 할당하였다.

그림 8(a), (b), (c), (d)는 각 CPU 태스크에 대해 제안하는 분석 기법으로 예측한 최대 도착 함수와 모의실험으로 산출한 도착 함수를 비교한 것이다. 그림에서 보듯이, 제안하는 분석 기법으로 예측한 최대 도착 함수가 모의실험으로 산출한 도착 함수를 안전하게 한정함을 알 수 있다. 두 함수 간의 오차는 제안하는 분석 기법에서 최대 도착 함수가 2-구간 선형 함수 형태를 유지함에서 비롯된 것이다. 제안하는 분석 기법으로 예측한 CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$와 DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수 $g_{dma}(t)$를 각각 그림 9(a)와 (b)에 나타내었다.

그림 10(a)는 DMA 태스크의 전송량을 100000 bus cycle까지 증가시켜 가면서 응답시간을 산출한 결과이다. 이 결과로부터 전송량이 5000 bus cycle 이상인 경우 20% 오차 범위 이내에서 안전한 응답시간이 산출되며, 그림 10(b)에서 보듯이 전송량이 증가할수록 오차율이 점차 감소함을 알 수 있다. 이와 같은 오차는

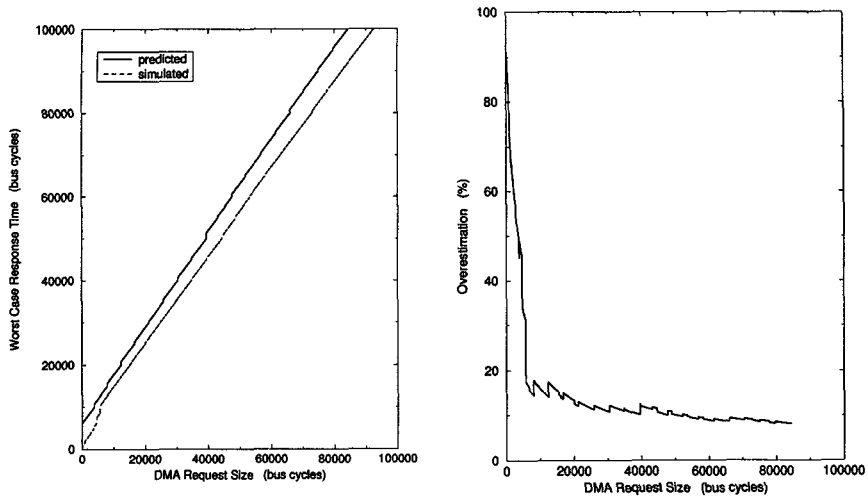
(1) 각 CPU 태스크의 캐쉬 접근 실패 분석 및 최대 도착 함수 통합 과정에서 가정했던 최악 수행 시나리오와 실제 수행 시나리오간의 차이와,

(2) 분석 시 고려하지 않았던, 이전 주기의 수행에서 적재된 캐쉬 내용의 재사용 등에서 비롯된 것이다.

그림 11은 상위 우선순위 DMA 태스크가 존재하는 경우 하위 우선순위 DMA 태스크의 최악 응답시간을 산출한 결과이다. 상위 우선순위 DMA 태스크의 주기는 20000 bus cycle이며 전송량이 5000 bus cycle인 경우(그림 11(a))와 10000 bus cycle인 경우(그림 11(b))에 대해 실험하였다. 두 경우 모두 그림 10(a)의 그래프와 유사한 형태를 보이는데, 상위 우선순위 DMA 태스크의 주기마다 응답시간이 급상승하며 상위 우선순위 DMA 태스크의 전송량과 응답시간의 상승량이 서로 상관관계를 가짐을 관찰할 수 있다.

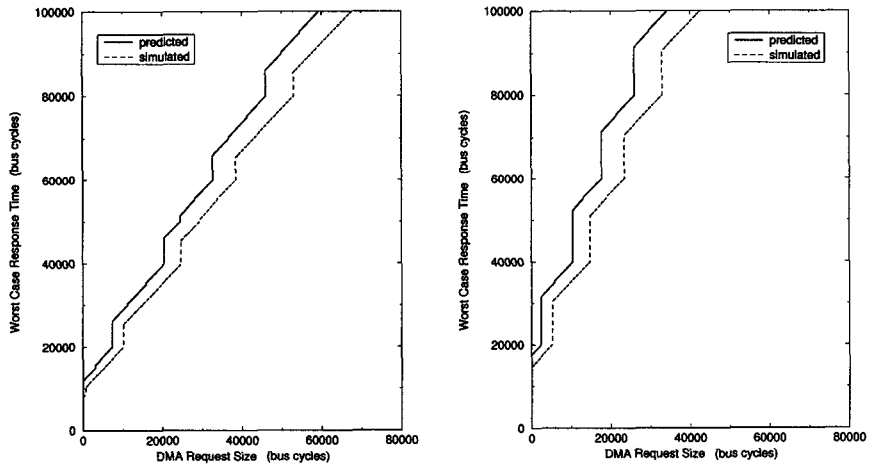
6. 결론 및 향후과제

본 논문에서는 CPU와 다수의 DMA 컨트롤러가 시스템 버스를 공유하는 환경에서 DMA I/O 요구의 최악 응답시간을 예측하는 기법을 제안하였다. 제안하는 분석 기법에서는 CPU에게 최상위 우선순위가 주어진 고정우선순위 버스 프로토콜을 가정하고, 먼저 CPU의 버스 요구에 대한 최대 도착 함수 $f_{cpu}(t)$를 CPU에 의한 DMA 전송 지연시간의 상한으로 구한 후, 이로부터 DMA 컨트롤러의 버스 요구에 대한 최소 서비스 함수 $g_{dma}(t)$를 구해 DMA 태스크의 최악 응답시간을 산출



(a) 최악 응답시간 (b) 오차율

그림 10 DMA 태스크의 최악 응답시간과 오차율



(a) 상위 우선순위 DMA 요구량 = 5000 bus cycles

(b) 상위 우선순위 DMA 요구량 = 10000 bus cycles

그림 11 상위 우선순위 DMA 태스크가 존재하는 경우 DMA 태스크의 최악 응답시간

한다. 모의실험의 결과, 통상적인 DMA I/O 전송에 있어서 20% 오차 범위 이내의 안전한 응답시간이 산출됨을 보였다. 향후과제로는 데이타 참조의 캐쉬 접근 실패로 인한 영향을 분석에 포함시키는 것과 라운드 로빈(round-robin) 버스 프로토콜 환경에서 DMA I/O 요구의 최악 응답시간을 분석하는 것을 고려하고 있다.

참 고 문 헌

[1] Joseph, M. and Pandya, P., "Finding Response Times in a Real-Time System," The BCS Computer Journal, Vol. 29, No. 5, pp. 390-395, 1986.

[2] Lehoczky, J. P., Sha, L. and Ding, Y., "The Rate Monotonic Scheduling Algorithm - Exact Characterization and Average Case Behavior," In Proceedings of the 10th Real-Time Systems Symposium, pp. 166-171, 1989.

[3] Liu, C. L. and Layland, J. W., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[4] Tindell, K. W., Burns, A. and Wellings, A. J., "An Extendable Approach for Analyzing Fixed Priority Hard Real-Time Tasks," Real-Time Systems, Vol. 6, No. 2, pp. 133-151, 1994.

[5] Hahn, J., Ha, R., Min, S. L. and Liu, J. W.-S., "Analysis of Worst Case DMA Response Time in Fixed-Priority Bus Arbitration Protocol," Real-Time Systems, Vol. 23, No. 3, pp. 209-238, 2002.
[6] Huang, T.-Y., Worst-Case Timing Analysis of Concurrently Executing DMA I/O and Programs, Ph.D. thesis, University of Illinois, 1997.
[7] Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S. L., Ha, R., Hong, S., Park, C. Y., Lee, M. and Kim, C. S., "Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling," In Proceedings of the 17th Real-Time Systems Symposium, pp. 264-274, 1996.
[8] Lee, C.-G., Hahn, J., Seo, Y.-M., Min, S. L., Ha, R., Hong, S., Park, C. Y., Lee, M. and Kim, C. S., "Enhanced Analysis of Cache-related Preemption Delay in Fixed-priority Preemptive Scheduling," In Proceedings of the 18th Real-Time Systems Symposium, pp. 187-198. 1997.

한 주 선

1994년 숭실대학교 전자계산학과 졸업
1996년 서울대학교 컴퓨터공학과 석사
2004년 서울대학교 컴퓨터공학과 박사
1999년~현재 지인정보기술(주) 책임 연구원. 관심분야는 실시간 시스템, 내장형 시스템, 플래시 메모리 파일 시스템

하 란

1987년 서울대학교 컴퓨터공학과 졸업
1989년 서울대학교 컴퓨터공학과 석사
1989년 3월~1990년 7월 한국통신 전임연구원. 1995년 University of Illinois at Urbana- Champaign 컴퓨터공학과 박사. 1995년 9월~현재 홍익대학교 컴퓨터공학과 부교수. 관심분야는 실시간 시스템, 멀티미디어 시스템, 이동 컴퓨팅, 내장형 시스템

민 상 렬

1983년 서울대학교 컴퓨터공학과 졸업
1985년 서울대학교 컴퓨터공학과 석사
1989년 University of Washington 전산학 박사. 1989년~1990년 IBM T.J. Watson Research Center 객원 연구원
1990년~1992년 부산대학교 컴퓨터공학과 조교수. 1992년~현재 서울대학교 전기컴퓨터공학부 교수. 관심분야는 내장형 시스템, 스토리지 시스템, 플래시 메모리