백플레인 버스 네트워크를 위한 최악 응답 시간 분석

(Analysis of Worst-Case Response Time for Backplane Bus Network)

성 민 영 † 장 래 혁 †† 신 현 식 ††
(Minyoung Sung)(Naehyuck Chang)(Heonshik Shin)

요 약 근래에 들어, 백플레인 버스를 기반으로 하는 멀티프로세서 시스템의 프로세서간 통신에도 TCP/IP와 같은 표준 네트워크 프로토콜을 사용하는 것이 보편화되었다. 표준 프로토콜을 사용하기 위해서는 백플레인 버스 프로토콜을 이용하여 표준 MAC 계층을 구현하는 것이 일반적이다. 본 논문은 이러한 MAC 에뮬레이션 기반 버스 네트워크상에서 내장형 실시간 응용을 지원하기 위한 최악 응답 시간 분석법을 제시한다. 본 논문의 분석법은 구체적으로 MAC 에뮬레이션 방법의 하나인 ANSI BusNet 프로토콜을 대상으로 진행된다. 각 실시간 태스크를 주기, CPU 시간, 종료시한, 메시지 패킷 개수로 모델링하고 스케줄 가능성, 즉 주어진 종료 시한 내에 작업을 완료할 수 있는지의 여부를 검사하는 수식을 유도한다. 이를 위해 물리적인 버스 특성을 고려한 버스 전송 모델을 제시하고, 버스 중재 방식과 버스 하드웨어의 캐쉬 지원 여부에 따른 스케줄 가능성을 분석한다. 또한 본 논문에서는 실험을 통해 블록 전송이 실시간 통신 성능에 미치는 영향을 살펴본다. 비록 본 논문의 분석법이 BusNet에 기반하여 개발되었지만, BusNet이 대부분의 백플레인 하드웨어가 지원하는 기본적인 기능만을 가정하고 있으므로, 본 논문의 분석법은 다른 종류의 백플레인 네트워크 프로토콜에도 쉽게 적용될 수 있다.

Abstract Nowadays, backplane bus-based multiprocessor systems often utilize the standard network protocol such as TCP/IP for communication between processors over the backplane bus. Typically a software module emulates the standard MAC protocol through the backplane bus protocol. This paper presents a worst-case response analysis for embedded real-time applications over the MAC emulation-based backplane bus network. In particular we choose BusNet as a target protocol, an ANSI standard protocol for MAC emulation. We model each real-time task using its period, CPU time, deadline and the number of packets transferred, and derive an expression to test the schedulability, i.e., whether the task can finish its job within the specified deadline. Our schedulability test reflects physical bus features such as bus arbitration schemes and write caches in the bus interface. We also give an experiment result which shows the effect of block transfer mechanism on the real-time performance. Since BusNet assumes only basic hardware features commonly used by most backplane hardware, our analysis can be applied to other backplane network protocols without difficulty.

1. 서 론

백플레인 버스는 실시간 내장형 시스템(real-time embedded system)내의 프로세서, 메모리 서브시스템, 입출력 장치 사이의 상호연결에 널리 사용되고 있다. 백플레인으로 연결된 멀티프로세서 시스템은 일반적으로 가격 대비 성능을 높이기 위해서 그리고 실시간 시스템의 가용성 (availability)과 결함 허용성(fault tolerance)을 높이기 위해 널리 사용된다. 이러한 시스템에서의 프로세서간 통신은 일반적으로 백플레인 버스 프로토콜을

† 비 회 원 : 서울대학교 컴퓨터공학부
minyoung@cslab.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학부 교수
naehyuck@comp.snu.ac.kr
shinhs@comp.snu.ac.kr
논문접수 : 2000년 3월 14일
심사완료 : 2000년 11월 23일

이용하여 직접 상대방의 공유 메모리를 접근함으로써 수행되어 왔다. 그러나, 컴퓨터 하드웨어 및 통신 기술의 급속한 발전에 따라 내장형 소프트웨어도 대량의 멀티미디어 데이타를 다루게 되고 복잡한 프로세서간 동기화 기법을 요구하기에 이르렀다. 이에 따라, 백플레인 버스 상에서도 TCP/IP와 같은 표준 네트워크 프로토콜을 사용하는 것이 일반화되고 있는 추세다. 표준 네트워크 프로토콜의 사용은 약간의 성능 저하를 가져오지만, 이식성(portability), 일관성(consistency), 견고성(robustness), 신속한 프로토타이핑 (prototyping) 등의 큰 이점을 제공한다. 이러한 백플레인 네트워크 프로토콜의 구현은 대개 CSMA/CD와 같은 표준 MAC (Media Access Control)을 에뮬레이션하여 상위 프로토콜(일반적으로 IP)과의 인터페이스를 맞추는 방식으로 진행된다. 이는 각 프로세서 보드마다 전용 NIC (Network Interface Card)를 부착할 필요 없이 모든 응용 소프트웨어가 표준 프로토콜을 사용할 수 있다는 점에서 비용상의 이점도 제공한다. 그러나, 이러한 장점에도 불구하고, 백플레인 버스 상에서의 표준 MAC 에뮬레이션 기반 실시간 통신을 위한 분석 기법은 거의 연구되지 않았었다. 실시간 응용에 대한 분석은 스케쥴 가능성(schedulability), 즉 주어진 일을 정해진 시간 내에 마칠 수 있는가의 여부에 대한 검사를 기반으로 수행된다. 일반적으로 실시간 태스크는 주기적으로 실행되며 높은 논리적 정확도(logical correctness)를 요구할 뿐 아니라 엄격한 시간 제약 조건(timing constraint)을 가진다. 태스크는 매번 실행될 때마다 실행을 마쳐야 하는 시간의 한계값 즉, 종료시한을 지켜야 한다. 최근에는 높은 성능을 위해 실시간 태스크들을 여러 프로세서에 분산시키고, 이들이 협조/통신하는 방법이 널리 사용되고 있다. 예를 들어 한 태스크가 원격 센서의 데이타를 수집하여 이를 다른 태스크에게 전달하면, 이 태스크는 데이타를 처리하여 출력하거나 또 다른 태스크에게 전송하는 것이다. 이러한 시스템에서 프로세서들은 LAN 혹은 백플레인 버스로 연결된다. 따라서 스케쥴 가능성 분석을 위해서는 태스크의 CPU 실행 시간 분석 뿐 아니라, 태스크간 통신에 소요되는 시간에 대한 정확한 분석도 필요하다. 특히, 비행기 제어, 산업 공정 제어, 공장 자동화 등과 같은 경성 실시간 (hard real-time) 응용은 종료시한을 준수하지 못했을 때 인명 손실이나 대규모 재산 손실과 같은 큰 재앙으로 이어질 수 있다. 따라서, 실시간 분석은 필수적이다. 본 논문에서는 MAC 에뮬레이션 기반 백플레인 네트워크 상에서의 경성 실시간 응용을 위한 분석 방법을 제시한다. 특히 태스크간 패킷 전송에 소요되는 최악 시간 분석에 주안점을 둔다.

백플레인 버스의 실시간 분석은 근래에 들어 다양한 상용 버스에 대해 활발히 연구되었다. Sha 외는 [1]에서 Futurebus+를 이용한 실시간 계산에 있어서의 고려사항을 연구하였다. 특히, 이 논문은 요구되는 수준의 우선 순위 단계가 버스에서 지원되지 않을 때의 스케쥴 가능성 손실에 대해서도 분석하고 있다. Kettler와 Strosnider [2]는 MCA (Micro Channel Architecture) 버스의 구체적인 물리적 메커니즘을 분석하여 데이타 전송 시간을 파악하고, 버스 이용률 (utilization) 을 기반으로 실시간 통신 시스템의 스케쥴 가능성 모델을 제시하였다. Tindel과 Wellings는 [3]에서 CAN (Control Area Network) 버스에 대한 실시간 통신 분석을 수행되었다. 이들은 최악 응답 시간 분석 기법 [4] 을 응용하여 버스상의 데이타 전송시간을 실행시간으로 하는 주기적 태스크 모델을 제시하였다. 또한 CAN의 일반적인 스케쥴 가능성 모델과 더불어 두가지 상용 CAN NIC의 성능을 실시간 응용의 관점에서 기술하였다.

본 논문에서는 MAC 에뮬레이션 기반 백플레인 버스상의 실시간 응용을 위한 분석 기법을 제시한다. 대부분의 기존 실시간 버스 연구는 MAC 에뮬레이션을 통한 패킷-기반 통신에 대한 적절한 모델을 제공하지 못하고 있다. 기존 연구들이 전용 NIC를 가정한 것과는 달리 MAC 에뮬레이션-기반 통신에서는 전용 NIC가 없다. 따라서, 버스 인터페이스의 특성에 따라 버스상의 데이타 전송 시간도 CPU 사용 시간으로 고려되어야 한다. 또, 대부분의 기존 연구에서 우선 순위 기반 중재 방식, 혹은 특정 백플레인 버스에 국한된 중재 방식을 가정한 것에 반해, 본 논문에서는 일반성을 위해 추상적인 중재 방식을 제안하고 대상 백플레인 버스로 매핑하는 방식을 취한다. 백플레인 버스상의 각 실시간 태스크는 주기, CPU 사용시간, 종료시한, 메시지 패킷수로 모델링된다. 스케쥴 가능성 분석 수식은 버스 중재 방식을 고려하는데, 특히 버스 하드웨어가 전송 요청을 캐슁할 수 있을 경우와 그렇지 않은 경우를 분리하여 유도된다. 버스 인터페이스에 캐슁 기능이 존재할 경우, 로컬 버스와 백플레인 버스가 독립적으로 동작할 수 있으므로, 전용 NIC을 갖추었을 때와 비슷한 방식으로 분석될 수 있다. 반면, 캐슁 기능이 존재하지 않을 경우, 버스상의 데이타 전송 시간도 CPU 사용 시간의 연장으로 분석되어야 한다. 본 논문에서는 또한 블록 전송 설정이 실시간 성능에 미치는 영향을 파악하기 위해, 실험을 통해 다양한 블록 전송 스케일 값에 따른 통신 응답 성능을 관찰한다. 일반성을 위해 본 논문은 VMEbus [5] 상에서 표

준화된 통신을 지원하기 위해 개발된 **BusNet** 프로토콜 [6, 7, 8] 을 분석 대상으로 선택한다. BusNet은 ANSI 표준 프로토콜로서, 대부분의 백플레인 하드웨어에 공통적으로 사용되는 기본적인 기능만을 가정하고 있다. 따라서, 본 논문의 분석법은 다른 종류의 백플레인 버스 네트워크에 대해서도 쉽게 적용될 수 있다.

본 논문은 다음과 같이 구성된다. 제 2 절에서는 백플레인 네트워크 프로토콜의 개념을 소개한다. 아울러, VMEbus의 버스 중재 방식과 데이타 전송 방식을 설명한다. 제 3 절에서는 실시간 스케쥴 가능성을 위한 검사 수식을 유도하며, 제 4 절에서는 실험을 통해 버스의 실시간 통신 성능을 기술한다. 본 논문은 제 5 절 결론을 마지막으로 마친다.

2. 표준 MAC 에뮬레이션 기반 백플레인 버스 네트워크

2.1 BusNet

근래에 들어, 대부분의 내장형 운영체제는 NIC 상에서 뿐 아니라, 백플레인 버스상에서도 표준 네트워크 프로토콜을 지원하고 있다. 그림 1에서 볼 수 있듯이, 백플레인 하드웨어와 ISO/OSI 네트워크 계층(network layer)간에 소프트웨어 모듈이 존재하여, Ethernet과 같은 표준 MAC 프로토콜을 에뮬레이션하는 방식으로 구현된다. 이 소프트웨어 모듈은 패킷 기반 통신에 필요한 자료구조, 프로토콜 상태, 전송 관련 함수들로 구성된다. 본 절에서는 표준 MAC 에뮬레이션 기법의 하나인 BusNet을 소개한다.

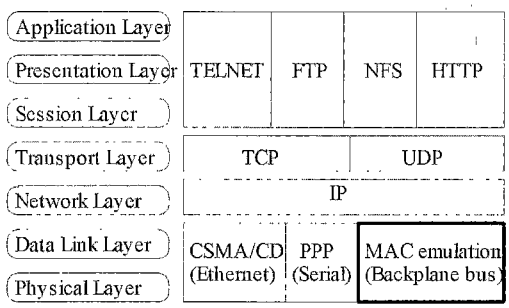


그림 1 백플레인 버스 네트워크와 OSI 모델

BusNet은 ANSI 표준 프로토콜로서, 원래 VMEbus 상에서 CSMA/CD 에뮬레이션을 위해 Force Computers사가 개발한 것이다. 다른 백플레인 네트워크 프로토콜과는 달리, BusNet은 기본적인 하드웨어 기능만을 가정하고 있다. 즉, BusNet 규정은 read-modify-write 혹은 mailbox와 같은 특수한 기능에 의존하지 않는다. 그림 2는 BusNet을 이용한 시스템의 구성을 보이고 있다.

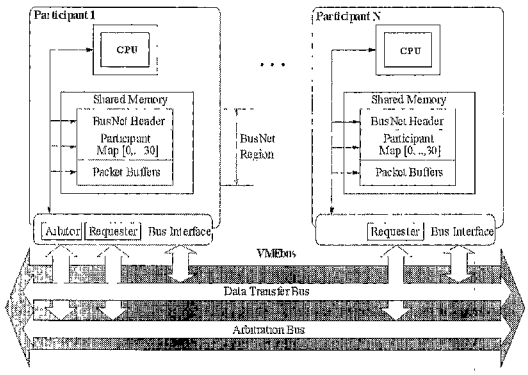


그림 2 BusNet 백플레인 네트워크

각 프로세서 혹은 참가자(participant)는 0 에서 30 사이의 논리적 주소로 식별된다. 각 참가자는 다른 참가자들이 접근할 수 있도록 자신의 로컬 메모리 일부를 공유하게 된다. 이 공유된 메모리는 BusNet 영역이라고 불리며, 헤더, 참가자 맵, 패킷 버퍼를 포함한다. 각 참가자 맵은 프로토콜 디스크립터 (PD) 를 포함하는데, 여기에는 두 참가자간의 패킷 교환을 위한 상태 정보, 포인터들이 들어 있다. 각 PD는 receive_status, transmit_status, buffer_offset, buffer_size, sequence_number로 구성된다. receive_status는 수신자가 패킷 버퍼에 대한 접근을 제어할 때 사용된다. receive_status가 RDY 상태이면, 송신자는 수신자의 버퍼에 패킷 데이타를 저장할 수 있다. transmit_status는 수신자에게 패킷 버퍼가 새로운 패킷으로 채워졌음을 알릴 때 사용된다. buffer_offset과 buffer_size는 각각 패킷 버퍼의 주소와 크기를 담고 있다. sequence_number는 오류 검사를 위해 포함되어 있다.

패킷 전송은 표 1에서 기술된 단계들에 의해 진행된다. 표에서 map_i는 참가자 i의 공유 메모리에 존재하는 참가자 맵을 지칭한다. 한가지 주목할 만한 점은 전체 전송 과정이 효율성을 위해 오직 쓰기 연산으로 이루어져 있다는 것이다. receive_status와 transmit_status 플래그도 각각 송신자와 수신자의 공유 메모리에 존재하므로, 상태를 검사하는 작업도 버스를 거치지 않는다. 쓰기 연산은 버스 인터페이스상의 저장 버퍼를 이용한

표 1 BusNet의 패킷 전송 단계

단계	설명
1	송신자(T)는 수신자(R)가 패킷을 받을 준비가 되었는지 검사 map_T.PD[R].receive_status == RDY
2	T는 버퍼 오프셋과 크기 변수를 파악한다 (map_T.PD[R].buffer_offset과 map_T.PD[R].buffer_size)
3	T는 receive_status를 IDLE로 세트. map_T.PD[R].receive_statue = IDLE
4	T는 R의 패킷 버퍼에 대한 VME 주소를 파악한다.
5	T는 R의 패킷 버퍼에 패킷을 전송한다.
6	T는 R의 PD에 있는 sequence_number를 갱신한다. map_R.PD[T].sequence_number += 1
7	T는 transmit_status를 RDY로 세트 map_R.PD[T].transmit_status = RDY
8	T는 R에게 mailbox 인터럽트를 건다 (선택적)
9	R은 map_R.PD[T].sequence_number에 저장되어 있는 번호를 확인
10	R은 transmit_status를 IDLE로 세트 map_R.PD[T].transmit_status = IDLE
11	R은 새 버퍼를 할당하고 그 주소를 map_T.PD[R].buffer_ offset에 저장
12	R은 receive_status을 RDY로 세트 map_T.PD[R].receive_status = RDY
13	R은 T에게 mailbox 인터럽트를 건다 (선택적)

cycle decoupling 메커니즘에 의해 고속화될 수 있으며, 이는 읽기 기반의 프로토콜에 비해 뛰어난 성능 향상을 보인다 [6, 9].

2.2 VMEbus 상에서의 데이타 전송

그림 2에서 인터페이스 모듈은 중재자(arbiter)와 요청자(requester)를 이용하여 데이타 전송 버스(data transfer bus)로의 접근을 제어한다. 표기상의 편의를 위해 앞으로 버스는 데이타 전송 버스를 지칭한다. 중재 버스(arbitration bus)는 버스 중재에 사용되며 버스 요청선(bus request lines)과 버스 인가선(bus grant lines)으로 구성된다. 요청자는 자신이 소속된 보드로부터의 전송 요청에 따라 4개 요청선 중 하나를 활성화시킴으로써 버스에 대한 접근을 요청한다. 중재자는 버스 요청에 대한 응답으로 버스 인가 신호를 활성화시킴으로써 해당 요청자에게 버스를 인가하게 된다. 같은 우선순위를 갖는 요청선들은 wired-OR 형태로 묶여 있지만, 버스 인가선들은 데이지-체인(daisy-chain)형식으로 보드 슬롯들을 연결한다.

VMEbus에서는 <중재자 종류, 요청자 종류>의 조합에 의해 여러 가지 중재 방식이 만들어질 수 있다. VMEbus 표준은 세가지 중재자 종류를 설명하고 있다. prioritized, single-level, round-robin 중재자가 그것이다. prioritized 중재자는 감지된 버스 요청들 중 가장 높은 우선순위에게 버스를 인가한다. single-level 중재자는 가장 높은 우선순위 요청선의 요청에만 응답한다. round-robin 중재자는 네 개의 요청 라인간에 공평한 버스 공유를 제공한다. 버스 중재에는 이러한 중재자들에 의한 중재 뿐 아니라, 버스 인가 데이지-체인에 의한 이차적인 중재가 존재한다. 요청자는 demand 혹은 fair 모드로 설정될 수 있다. demand 모드에서는 동일한 요청선을 공유하는 요청자들이 그 슬롯 위치에 의해 우선순위가 지정된다. 중재자에 가장 가까운 요청자가 가장 높은 우선순위를 갖게 되는 것이다. 따라서, 데이지-체인의 끝에 존재하는 프로세서는 버스 부하가 높을 때 거의 서비스되지 않을 수 있다. 공평한 접근을 위해서는 fair 요청자가 사용되어야 한다. fair 요청자는 일단 버스를 인가받게 되면, 자신의 우선순위에 해당하는 다른 요청들이 존재하는 동안 추가적인 요청을 발생시키지 않는다.

본 논문에서는 특정 백플레인 버스에 국한된 중재 방식보다는 일반적인 버스 중재 방식을 고려하고 실제 버스로의 매핑 방법을 기술한다. 추상적인 중재 방식으로 PRI와 FAIR 방식을 생각할 수 있다. PRI 방식에서는 각 프로세서 혹은 참가자에게 유일한 우선순위가 할당되고, 버스 소유권은 항상 최상위 우선순위를 갖는 참가자에게 부여된다. <prioritized, demand>와 <single-level, demand>조합이 이 방식에 포함된다. FAIR 방식에서는 이상적으로 공평한 버스 공유가 보장된다. <single-level, fair>조합이 FAIR 방식에 해당된다. <round-robin, fair>조합은 각 우선 순위 요청선에 동일한 수의 참가자가 연결되어 있을 때, FAIR 방식에 포함될 수 있다. PRI와 FAIR 중재 방식은 대부분의 백플레인 버스가 지원하고 있으며, 쉽게 매핑될 수 있다.

버스상의 데이타 전송은 트랜잭션(transaction) 단위로 진행된다. 트랜잭션은 가장 낮은 수준의 전송 메커니즘으로서, 버스상의 독립적인 전송을 나타낸다. 트랜잭션은 전송 모드에 따라 다르게 정의된다. 본 논문에서는 기본적인 전송모드인 디폴트(default)와 블록(block) 전송모드를 고려한다. 그림 3에서 볼 수 있듯이, 디폴트 모드에서의 각 트랜잭션은 중재 사이클과 주소/데이타 사이클로 구성된다. 그러나, 블록 트랜잭션은 일단 마스

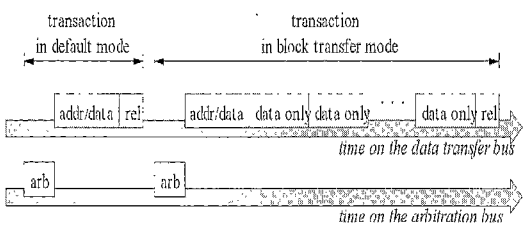


그림 3 버스 전송 모드

터(master)가 버스를 획득한 후에는, 하나 이상의 데이타 사이클에 대해 단 한번의 중재 사이클만을 필요로 한다. 메모리 주소는 슬레이브(slave)에 의해 자동적으로 증가된다.

3. 스케쥴 모델 및 실시간 분석

3.1 패킷 전송 시간

패킷 전송 시간은 하나의 패킷을 전송하는데 버스 상에서 걸리는 시간으로 정의한다. 각 버스 트랜잭션에 걸리는 시간을 σ로 표기할 때, 디폴트 모드와 블록 모드의 트랜잭션 시간은 각각 다음과 같다.

$$\sigma = \pi_b + \pi_a + \pi_r$$

$$\sigma = \pi_b + \pi_a + (m-1)\pi_d + \pi_r$$

표 2에는 패킷 전송 시간 모델에 사용된 표기들이 전형적인 값과 함께 설명되어 있다.[1] 일반적으로 메모리 블록의 전송은 전송 효율을 위해 블록 모드에서 수행된다. 블록 전송을 가정하고, 표 1의 제 5 단계, 즉 패킷 버퍼로의 데이타 전송에 중점을 두면, 패킷 전송 시간은 다음과 같다.

$$\nu = l\sigma = \left\lceil \frac{s}{wm} \right\rceil \sigma$$

나머지 단계들에 대한 고려는 제 3.2 절 실시간 분석에서 상세 기술한다.

3.2 최악 응답 시간 분석

내장형 실시간 응용들은 주기적으로 수행되며 주어진 종료시한 내에 수행을 마쳐야 한다는 제약 조건을 가진다. 실시간 태스크의 시간 제약 준수 여부를 파악하기 위해서는 태스크의 최악 응답 시간 (worst-case response time), 즉 태스크가 도착한 시점부터 성공적으로 작업을 완료할 때까지의 최대 시간이 분석되어야 한다. 응답 시간은 CPU 실행에 따른 시간과 통신 지연 (communication delay) 으로 나누어 볼 수 있다. CPU 실행에 따른 시간에는 CPU를 할당받아 이용한 시간과 높은 우선 순위 태스크들에 의해 선점된 시간, 그리고 낮은 우선 순위 태스크에 의한 간섭이 포함된다. 통신 지연은 송신할 메시지가 발생한 시점부터 수신 태스크가 존재하는 프로세서에게 전달되기까지의 시간을 나타낸다.

본 논문에서는 상대방 프로세서가 도착한 메시지를 처리하는데 따른 지연은 통신 지연에 포함하지 않았다. 즉 BusNet의 단계 6~7과 단계 11~12가 그것인데, 이 부분은 프로세서간의 동기화와 관련되어 구현마다 다를

1) 표에서 전형적인 버스 사이클 시간값은 Tundra Semiconductor 사의 SCV64 VMEbus 인터페이스 칩 [10]에 기반한 것이다.

표 2 버스 전송 모델에 사용되는 표기들

표기	설명	값
ν	패킷 전송 시간.	
σ	버스 트랜잭션 시간.	
l	s-바이트 패킷을 전송하는데 필요한 트랜잭션의 개수 $l = \left\lceil \frac{s}{wm} \right\rceil$.	
s	패킷 크기를 바이트로 표현한 값.	2048
w	버스 폭(bus width)을 바이트로 표현한 값.	4
m	블록 전송 스케일. 각 블록 트랜잭션에 포함되는 데이타 사이클 개수.	64
π_b	중재 사이클에 걸리는 시간.	78 ns
π_a	주소/데이타 사이클에 걸리는 시간.	159 ns
π_d	데이타 사이클에 걸리는 시간.	149 ns
π_r	버스를 놓는데 걸리는 시간.	41 ns

수 있으며 그 처리시간이 해당 프로세서 내의 주기적 태스크로 쉽게 고려될 수 있기 때문이다. 특히, 이 부분은 프로세서의 인터럽트 처리 루틴으로 구현되어 최상위 우선순위로 수행되는 것이 일반적이므로, 처리 시간의 한계를 비교적 쉽게 결정할 수 있다.[2]

일반적으로, 실시간 태스크는 주기, 최악 CPU 실행 시간, 종료시한으로 모델링된다. 최악 CPU 실행 시간은 다른 태스크의 간섭 없이 CPU를 사용할 때 최악의 경우 걸리는 시간을 지칭하며 다양한 분석 기법이 활발히 연구되고 있다. 주기와 종료시한은 시스템 설계 단계에서 결정된다. 여기에 통신지연을 고려하기 위해서 매 주기마다 생성하는 패킷의 최대 데이타량이 정해지게 된다. 따라서, 본 논문에서는 최악 응답 분석을 위해 다음과 같은 태스크 모델을 가정한다. MAC 에뮬레이션 기반 버스 네트워크 상의 프로세서 p에 존재하는 실시간 태스크 i는 (T_i , C_i , D_i , n_i)로 정의된다. T_i, C_i, D_i 는 태스크 i의 주기, 최악 CPU 실행 시간, 그리고 종료시한을 나타낸다. n_i는 태스크가 매주기마다 생성하는 패킷수의 최대값이다. 태스크 i의 최악 응답 시간을 R_i로 표기하면, 태스크 i는 $R_i \le D_i$일 때 스케쥴 가능하다. R_i의 분석을 위해 다음 사항을 가정한다.

1. 태스크의 최악 CPU 실행 시간 (worst-case execution time) 을 이미 알고 있다. 이는 대부분의 실시간 분석 논문에서 가정하고 있는 사항이다. 특히, 최근의 활발한 연구들은 CISC 뿐 아니라 RISC에 대해서도 최

2) 그러나, BusNet은 실시간 이용에 있어서 우선순위 역전 (priority inversion) 문제 [11]를 야기할 수 있다. 왜냐하면, VMEbus의 특성상 단계 11~12가 수신 프로세서의 버스 우선순위로 수행되기 때문이다.

악 실행 시간 분석을 가능하게 하고 있다 [12, 13].

2. 태스크는 실행의 마지막 단계에서 메시지를 발생시키며, 매 주기에 전송할 최대 데이타량이 결정되어 있다. 실시간 응용은 센서에 의해 주기적으로 발생되는 데이타를 처리하여 출력하거나, 다음 태스크에게 전달하는 전형적인 통신 모델을 취하고 있다. 이 때, 발생되는 데이타는 그 크기가 고정되어 있는 경우가 많고, 높은 결함 허용(fault tolerance)을 위해 압축을 사용하지 않는다. 매 주기마다 발생되는 데이타량이 가변일 경우라 하더라도 제한된(bounded) 통신 지연을 위해 그 최대 크기를 고정하고 있다 [14].

3. 최대 패킷 크기가 정해져 있다. 이는 주기마다 생성되는 패킷 데이타량의 최대값을 결정하기 위해 필요한 가정이다. 패킷 크기의 최대값은 일반적으로 각 프로토콜 스택마다 PDU (Protocol Data Unit) 로 정의되어 있다. BusNet의 경우 기본적으로 2048 바이트이다. 따라서 태스크 i가 매 주기마다 생성할 수 있는 최대 데이타량은 2048 $\times n_i$이다.

4. 프로세서 보드 내의 패킷 큐는 우선순위 기반으로 관리된다. 이 가정은 높은 우선순위의 태스크가 낮은 우선순위 태스크들의 패킷들로 인해 기다리게 되는 시간, 즉 블록킹 시간(blocking time)을 제한하기 위해 필요하다.

최악 응답 시간 분석을 위해서는 우선, 버스 인터페이스에 캐싱 기능 (write posting 혹은 cycle decoupling) [13] 이 존재하는지를 파악해야 한다. 로컬 버스와 백플레인 버스간의 데이타 전송은 일반적으로 결합된 핸드쉐이킹 (coupled handshaking) 프로토콜을 수반한다. 즉, 데이타 전송을 시작한 버스는 데이타 전송 체인 (마스터 로컬 버스, 백플레인 버스, 슬레이브 로컬 버스) 상의 마지막 버스가 전송 사이클을 마쳐야만 전송 사이클이 종료된다. 그러나 캐싱 기능이 지원되는 경우, 한쪽 버스 (로컬 혹은 백플레인 버스) 에서 사이클이 도착하면, 이는 인터페이스 컨트롤러의 내부 FIFO에 등록되면서 바로 전송 사이클을 종료한다. 인터페이스가 다른쪽 버스 (백플레인 혹은 로컬 버스) 를 획득하고 데이타 전송을 수행하는 동안 원래의 버스는 다른 사이클을 수행할 수 있다. 패킷을 전송하는데 필요한 충분한 FIFO 버퍼가 존재할 경우, 전용 네트워크 어댑터를 갖춘 시스템에서와 마찬가지 방식으로 분석할 수 있다. 일반적으로 네트워크 인터페이스의 송신 루틴은 패킷을 송신 큐에 삽입하고 패킷 송신 인터럽트를 건다. 그 이후의 패킷 송신은 현재의 CPU 작업과 상관없이 진행될 수 있다. 만약 캐싱이 지원되지 않거나, 충분한 FIFO 용량이 존재하지 않는다면, 패킷 전송은 로컬 CPU 사이클과 결합되어 수행되어야 한다. 이 경우, 낮은 버스 우선순위의 프로세서들은 다른 프로세서의 간섭으로 인해 낮은 CPU 효율을 갖게 될 수 있다.

가. 캐싱이 지원되는 경우

버스 인터페이스에 패킷을 전송하는데 충분한 캐쉬가 존재할 경우, 최악 응답 분석은 전용 네트워크 카드가 존재하는 시스템에서와 비슷한 방식으로 진행된다. 즉, 버스 전송은 CPU 작업과 독립적으로 분석될 수 있다. 매 주기의 시작에서 메시지가 발생하기까지의 시간 간격의 최대 편차를 지터 (jitter) 라 정의하면, 지터는 태스크 CPU 실행의 최악 응답 시간과 동일하게 된다. τ_i의 메시지 지터와 최악 통신 지연을 각각 R_i^{CPU}, R_i^{MSG}로 표기하면, $R_i = R_i^{\mathrm{CPU}} + R_i^{\mathrm{MSG}}$이다 (그림 4 참조). R_i^{CPU}는 다음과 같이 분석될 수 있다 [4].

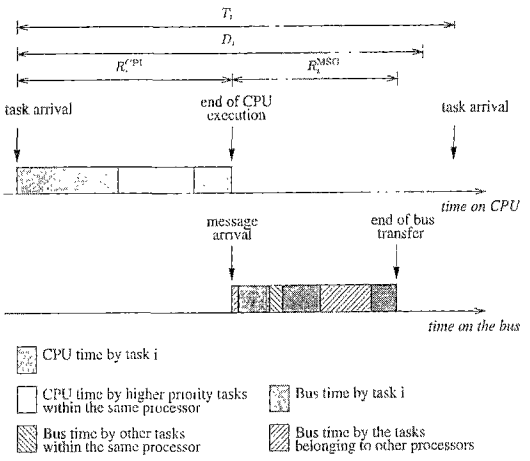


그림 4 캐싱이 지원되는 시스템에서의 최악 응답 시간

$$R_i^{\mathrm{CPU}} = \max_{q=0,1,2,\ldots}(w_{i,q}^{\mathrm{CPU}} - qT_i),$$
$$w_{i,q}^{\mathrm{CPU}} = (q+1)C_i + \sum_{\forall j \in \overline{\Delta_p}(i)} \left\lceil \frac{w_{i,q}^{\mathrm{CPU}}}{T_j} \right\rceil C_j. \qquad (1)$$

$\overline{\Delta_p}(i)$는 태스크 i와 동일한 프로세서에 존재하면서 태스크 i보다 높은 우선순위를 갖는 태스크들의 집합이다. 따라서, $w_{i,q}^{\mathrm{CPU}}$의 두번째 항은 높은 우선순위 태스크에 의한 간섭을 표현한다.

Ω를 시스템에 존재하는 모든 프로세서들의 집합, 그리고 $\overline{\Omega}(p)$를 프로세서 p보다 높은 버스 우선순위를 갖는 프로세서들의 집합이라 하자. 또, 프로세서 p에 속하

는 모든 태스크들의 집합을 Δ_p로 표기하자. Sung 외의 분석법 [9] 을 PRI 방식으로 확장하면, PRI 하에서 프로세서 p의 태스크 i에 대한 최악 통신 지연 R_i^{MSG}는 B_i, L_i, O_i를

$$B_i = \nu + \sigma, \tag{2}$$

$$L_i = (q+1)n_i\nu + \sum_{\forall j \in \Delta_p(i)} \left\lceil \frac{w_{i,q}^{MSG} + R_j^{CPU}}{T_j} \right\rceil n_j\nu, \tag{3}$$

$$O_i = \sum_{\forall u \in \Omega(p)} \sum_{\forall j \in \Delta_u} \left\lceil \frac{w_{i,q}^{MSG} + R_j^{CPU}}{T_j} \right\rceil n_j\nu \tag{4}$$

로 정의할 때 다음과 같다.

$$R_i^{MSG} = \max_{q=0,1,2,\ldots} (w_{i,q}^{MSG} - qT_i),$$
$$w_{i,q}^{MSG} = B_i + L_i + O_i. \tag{5}$$

B_i는 블록킹 시간, 즉 낮은 우선순위 작업에 의한 지연시간이다. 로컬 프로세서 내에서의 패킷 전송 루틴과 백플레인 버스상의 트랜잭션은 선점 (preemption) 불가능한 형태로 진행되므로, B_i는 ν와 σ의 합이다. L_i는 프로세서 p에 의한 버스 시간을 나타내며, 태스크 i와 그보다 높은 우선순위의 태스크들에 의한 메시지 전송 시간으로 구성된다. O_i는 다른 프로세서들의 버스 사용으로 인한 간섭을 표현한다.

FAIR 방식 하에서의 최악 통신 지연은 다른 프로세서에 의한 버스 간섭만이 달라지므로, O_i를 다음처럼 정의하면 된다.

$$O_i = \sum_{\forall u \in \Omega, u \neq p} \min\left(L_i, \sum_{\forall j \in \Delta_u} \left\lceil \frac{w_{i,q}^{MSG} + R_j^{CPU}}{T_j} \right\rceil n_j\nu\right). \tag{6}$$

나. 캐쉬이 지원되지 않는 경우

캐쉬이 지원되지 않거나 혹은 지원되더라도 충분한 캐쉬가 존재하지 않을 경우, 버스를 통한 메시지 송신은 로컬 CPU 사이클과 결합되어 수행된다. 따라서 $w_{i,q}^{CPU}$, B_i, L_i, O_i를 수식 (1)-(4), (6)과 같이 정의할 때 최악 응답 시간 R_i는 다음과 같다.

$$R_i = \max_{q=0,1,2,} (w_{i,q} - qT_i),$$
$$w_{i,q} = w_{i,q}^{CPU} + B_i + L_i + O_i. \tag{7}$$

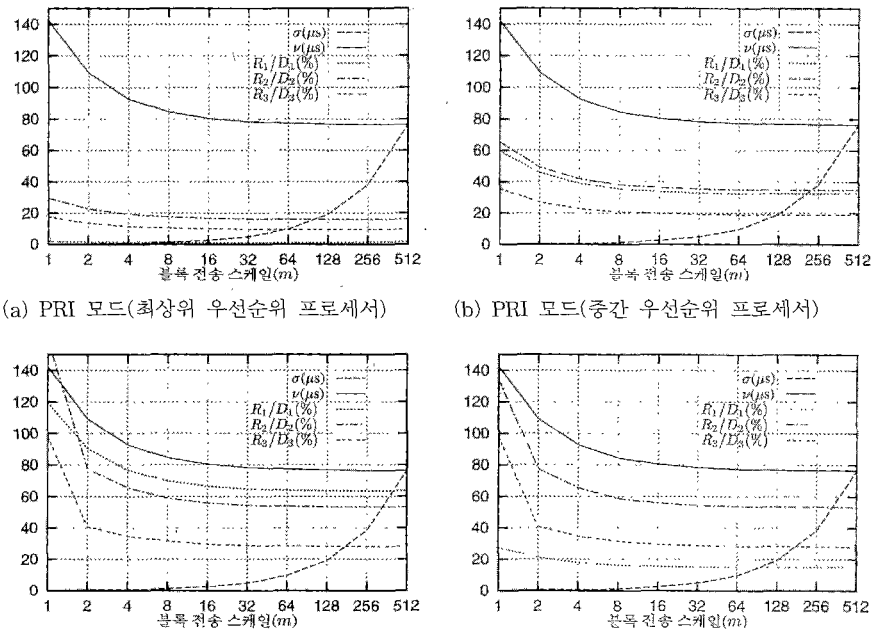
4. 실시간 통신 성능

본 절에서는 버스 채널 특성이 실시간 성능에 미치는 영향을 보이기 위해, 다양한 블록 전송 스케일에 따른 최악 응답 성능에 대한 실험 결과를 제시한다. 통신 성능에 초점을 맞추기 위해 CPU 실행시간이 없다고 가정한다. 따라서 버스 캐쉬의 영향을 무시할 수 있다. 모든 태스크는 도착과 동시에 버스 전송을 시작하게 된다. 또한 PRI 모드 하에서 프로세서의 우선순위에 따른 최악 응답 시간을 비교하기 위해, 각 프로세서에 동일한 태스크 집합이 존재하도록 설정하였다. 실험에 사용된 태스크 집합은 표 3과 같다. 패킷은 2048 바이트의 고정 크기를 가진다고 가정했다. 이는 대용량의 메시지를 전송하는 경우, 대부분의 패킷이 최대크기 (BusNet은 기본적으로 2048바이트) 로 생성된다는 점을 고려하여 설정한 것이다.

표 3 실험에 사용된 태스크 집합

	T_i	C_i	D_i	n_i	우선 순위(PRI 모드)
태스크 1	15 ms	0 ms	15 ms	1	최상위
태스크 2	25 ms	0 ms	25 ms	50	중간
태스크 3	50 ms	0 ms	50 ms	10	최하위

그림 5는 블록 전송 스케일 값에 따른 태스크의 최악 응답 시간을 보여주고 있다. 실시간 성능의 척도로서 태스크의 최악 응답도 R_i/D_i를 사용한다. 이 값은 태스크 자신의 종료 시한에 대한 상대적인 응답도를 표현하며 그 값이 1보다 클 경우 스케줄 불가능함을 뜻한다. 그림에서 알 수 있듯이, 블록 전송 스케일(m)이 증가함에 따라 응답 성능은 전반적으로 개선되었다. 그 이유는 m이 증가함에 따라 버스 중재 시간과 버스 놓는 시간 등의 트랜잭션 오버헤드가 급격히 감소하기 때문이다. 실제로 m값이 1 일 때와 비교하여 m값이 64 이상일 때 트랜잭션 오버헤드가 40% 이상 감소함을 볼 수 있다. 그러나, 대략 64 이상의 m값에 대해서는 성능 향상이 그리 뚜렷하지 못함을 알 수 있다. 이 m값은 우연히도 Tundra SCV64가 지원하는 최대 블록 전송 스케일과 일치하였다. 그러나, 긴 트랜잭션 시간은 높은 우선 순위 태스크들의 블록킹 시간을 증가시킨다. 위 실험의 경우는 높은 우선 순위 태스크들도 m값이 커짐에 따라 전반적으로 성능이 향상되는 것을 보이고 있다. 실험에서는 태스크의 주기 (15 ms) 가 트랜잭션 시간 (최대 80 μs) 에 비해 상당히 크게 설정되어 있다. 따라서 블록킹 시간의 영향이 미미하다. 하지만, 주기가 트랜잭션 시간에 견줄 만큼 작은 실시간 태스크의 경우에는 그만큼 블록킹 영향이 커지게 되어 결국 응답 성능을 크게 악화시킬 우려가 있다. 또한 대규모 블록 스케일은 버스 인터페이스의 캐쉬 메모리 비용을 높이게 된다. 따라서 블록 전송 스케일은 전반적인 최악 응답 성능과 비용 그리고 높은 우선 순위 태스크의 블록킹 시간을 고려하여 적절한 값에서 결정되어야 할 것이다. 그림 5 (a)~



(a) PRI 모드(최상위 우선순위 프로세서) (b) PRI 모드(중간 우선순위 프로세서)

(c) PRI 모드(최하위 우선순위 프로세서) (d) FAIR 모드

그림 5 블록 전송 스케일에 따른 최악 응답 성능

(c)에서 볼 수 있듯이, PRI 중재 모드에서의 최악 응답 성능은 예상대로 프로세서의 버스 우선순위에 따라 큰 편차를 보였다. 특히 최하위 우선순위 프로세서의 태스크 1과 2는 블록 전송 스케일이 매우 작을 때, 스케쥴 불가능할 수도 있음을 관찰할 수 있다. FAIR 모드에서의 최악 응답 성능은 그림 5 (d)에 보였다.

5. 결 론

백플레인 버스위에서 표준 MAC을 에뮬레이션하는 기법은 내장형 멀티프로세서 시스템의 프로세서간 통신에 널리 사용되고 있다. 본 논문에서는 이러한 백플레인 네트워크 상의 실시간 응용을 위한 분석 기법을 제시했다. 물리적 채널 특성을 고려한 버스 전송 모델을 개발하였으며, 이를 기반으로 실시간 태스크의 스케쥴 가능성 검사 방법을 제시하였다. 이를 위해 버스 중재 방식과 버스 인터페이스의 캐쉉 지원 여부를 고려하여 최악 응답 시간에 대한 수식을 유도하였다. 버스 중재 방식은 크게 PRI와 FAIR 방식으로 분류하여 각각의 중재 방식 하에서의 다른 프로세서의 간섭을 고려하였다. 또한, 백플레인 네트워크에서는 일반적인 네트워크 어뎁터를 갖춘 시스템과는 달리 버스 하드웨어의 캐쉉 여부를 고려해야 했다. 그것은 패킷 전송에 충분한 캐쉬가 존재할 경우, 버스상의 데이타 전송은 로컬 CPU 사이클과 독립적으로 수행될 수 있지만, 캐쉉이 지원되지 않으면, 데이타 전송이 CPU 실행에 직접적인 영향을 주기 때문이다. 실험을 통해, 블록 전송 스케일이 실시간 성능에 미치는 영향을 살펴보았다. 실험 결과, 블록 전송 스케일이 커짐에 따라 전송 효율이 높아지고 전반적인 응답 성능이 개선됨을 알 수 있었다. 그러나, 일정값 이상의 블록 전송 스케일에 대해서는 성능 향상이 뚜렷하지 못했다. 특히 긴 블록 전송이 짧은 주기의 높은 우선순위 태스크의 블록킹 지연을 증가시킨다. 따라서 전체적인 응답 성능과 블록킹 지연간의 장단점을 고려하여 적절한 블록 스케일 값을 결정해야 함을 알 수 있었다.

본 논문의 분석 수식은 블록 전송 스케일 뿐 아니라, 버스 대역폭, 패킷 크기, 각종 트랜잭션 인자 등의 다양한 변수들에 따른 응답 성능을 분석하는데 사용될 수 있으며, 이러한 결과는 실시간 시스템의 안정성을 검증하고, 효율적인 시스템을 설계하는데 중요한 지침으로 사용될 수 있다는 점에서 의의가 있다. 본 논문의 분석은 구체적으로 BusNet을 대상으로 진행되었다. 그러나, BusNet이 다른 프로토콜과 달리 대부분의 백플레인 버

스에서 일반적으로 사용되는 기본적인 하드웨어 특성만을 가정하고 있으므로, 본 논문의 분석이 다른 MAC 에뮬레이션 기반 백플레인 네트워크에도 쉽게 적용할 수 있으리라 기대한다.

참 고 문 헌

[1] L. Sha, R. Rajkumar and J. P. Lehoczky, "Real-Time Computing with IEEE Futurebus+," *IEEE Micro*, pp.30-99, Jun. 1991.

[2] K. A. Kettler and J. K. Strosnider, "Scheduling Analysis of the Micro Channel Architecture for Multimedia Applications," *Proc. of the IEEE Int'l Conf. on Multimedia and Computing Systems*, pp.403-414, 1994.

[3] K. Tindell, H. Hansson and A. Wellings, "Analysing Real-Time Communications: Controller Area Network (CAN)," *Proc. of the IEEE Real-Time Systems Symposium*, pp.259-263 1994.

[4] N. Audsley, A. Burns, M. Richardson, K. Tindell and A. Wellings, "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling," *Software Engineering Journal*, vol.8, no.5, pp.284-292, 1993.

[5] IEEE, "VMEbus specification: Draft Specification Revision D1.6," May 1993.

[6] VITA 19.0-1997, "Summary and introduction to the BusNet standard," 1997.

[7] VITA 19.1-1997 Draft D5, "BusNet Media Access Control(MAC) specification," 1997.

[8] VITA 19.2-1997 Draft D3, "BusNet Link Layer Control(LLC) specification," 1997.

[9] M. Sung and T. Kim and N. Chang and H. Shin, "Analysis of Real-Time Backplane Bus Network Based on Write Posting," *Proc. of the Real-Time Computing Systems and Applications*, pp.166-169, 1998.

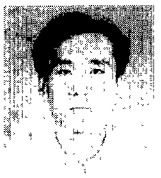
[10] SCV64 User Manual, Tundra Co., 1998.

[11] L. Sha, R. Rajkumar and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Trans. on Computers*, vol.39, no.9, pp.1175-1185, Sep. 1990.

[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, July 1995

[13] C. A. Healy, D. B. Whalley and M. G. Harmon, "Integrating the Timing Analysis of Pipelining and Instruction Caching,". *Proc. of the 16th Real-Time Systems Symposium*, pages 288-297, Dec. 1995.

[14] K. Tindell and A. Burns and J. Wellings, "Analysis of Hard Real-Time Communications," *Real Time Systems*, vol.9, pp.147-171, 1995.



성 민 영

1995년 2월 서울대학교 컴퓨터공학과 학사. 1997년 2월 서울대학교 컴퓨터공학과 석사. 1997년 3월 ~ 현재 서울대학교 컴퓨터공학부 박사과정. 관심분야는 실시간 시스템, 내장형 시스템 소프트웨어, 통신 시스템 성능 분석, 실시간 무선 통신 기법 등.

장 래 혁

1989년 서울대학교 제어계측공학과 공학사. 1992년 서울대학교 제어계측공학과 공학석사. 1996년 서울대학교 제어계측공학과 공학박사. 1997년 9월 ~ 현재 서울대학교 컴퓨터공학부 조교수. 관심분야는 저전력 시스템, 디지털 시스템 설계, 실시간 내장형 시스템 및 실시간 자바

신 현 식

1973년 서울대학교 응용물리학과 공학사. 1980년 미국 텍사스 대학교 의공학과 공학석사. 1985년 미국 텍사스 대학교 전기·컴퓨터공학과 공학박사. 1886년 ~ 현재 서울대학교 컴퓨터공학부 교수. 관심분야는 실시간 계산, 분산 시스템, 입출력 처리