Protocol Mapping을 이용한 인터페이스 자동생성 기법 연구

준회원 이 서 훈*, 강 경 구*, 정회원 황 선 영*

A Study on Automatic Interface Generation by Protocol Mapping

Ser-Hoon Lee*, Kyung-Goo Kang* *Associate Members*, Sun-Young Hwang* *Regular Member*

요 약

SoC 설계는 복잡도 증가 및 빠른 time-to-market에 만족하기 위해 IP에 기반한 설계방식을 채택하고 있다. Mobile 기기의 고성능에 대한 시장의 요구로 인해 embedded용 SoC는 멀티미디어, DMB 및 이미지처리 등 복잡도와 데이터 처리량이 높은 프로그램을 실시간으로 동작시키기 위해 다중 프로세서를 사용한 설계가 요구된다. 시스템 버스와 프로토콜이 상이한 프로세서를 단일 SoC내에서 사용하기 위해선 프로세서 프로토콜을 시스템 버스 프로토콜에 맞도록 변화하여 주는 인터페이스 회로의 설계가 요구된다. 고속으로 동작하는 프로세서의 인터페이스 회로는 데이터 쓰기와 읽기 시의 전송 지연을 최소화하여 시스템 전체의 성능을 향상시켜야 한다. 버퍼를 사용한 인터페이스 회로의 구조는 버퍼에 데이터를 일시 저장하는 동작으로 인하여 데이터 전송 latency가 증가하게 되므로 본 논문에서는 버퍼를 사용하지 않고 버스와 마스터 모듈 프로토콜이 가진 공통된 동작 시퀀스를 이용하여 단일 FSM 구조를 가진 인터페이스 회로를 자동생성하는 방법을 제안한다. 제안된 방법으로 자동생성된 인터페이스 회로는 버퍼를 사용한 인터페이스 회로에 비해 면적은 평균 48.5%의 감소를 보였으며, 데이터 전송 latency는 단일 데이터 전송 시 평균 59.1%의 감소를 보였고 버스트 모드 데이터 전송 시 13.3%의 감소를 보였다. 본 논문에서 제안한 시스템을 사용하여 데이터 전송 latency를 최소화하는 고성능의 인터페이스 회로를 자동으로 생성할 수 있다.

Key Words : SoC, Interface generation, FSM, Protocol mapping

ABSTRACT

IP-based design methodology has been popularly employed for SoC design to reduce design complexity and to cope with time-to-market pressure. Due to the request for high performance of current mobile systems, embedded SoC design needs a multi-processor to manage problems of high complexity and the data processing such as multimedia, DMB and image processing in real time. Interface module for communication between system buses and processors are required, since many IPs employ different protocols. High performance processors require interface module to minimize the latency of data transmission during read-write operation and to enhance the performance of a top level system. This paper proposes an automatic interface generation system based on FSM generated from the common protocol description sequence of a bus and an IP. The proposed interface does not use a buffer which stores data temporally causing the data transmission latency. Experimental results show that the area of the interface circuits generated by the proposed system is reduced by 48.5% on the average, when comparing to buffer-based interface circuits. Data transmission latency is reduced by 59.1% for single data transfer and by 13.3% for burst mode data transfer. By using the proposed system, it becomes possible to generate a high performance interface circuit automatically.

※ 본 연구는 대학 IT 연구센터(연세대학교 ITRC) 육성 지원 사업 지원으로 수행되었습니다.
* 서강대학교 전자공학과 CAD & ES 연구실 (hwang@ccs.sogang.ac.kr)
논문번호 : KICS2006-02-084, 접수일자 : 2006년 6월 29일, 최종논문접수일자 : 2006년 6월 29일

Ⅰ. 서 론

모바일 기기의 사용 증가는 제품의 빠른 time-to-market 사이클과 embedded system의 저전력화 및 복잡한 어플리케이션의 실시간 수행 등의 요구를 야기 시킨다[1]. Embedded system 설계는 빠른 time- to-market 요구에 부합하기 위해 IP를 사용한 설계를 채택하고 있다[2, 3]. 이미 설계 검증된 IP는 특정한 모듈과의 데이터 통신을 위한 특정한 프로토콜을 갖고 있어 다른 모듈과의 통신을 위해서는 이들 간에 통신을 가능하게 프로토콜을 맞추어 주는 인터페이스 회로를 필요로 한다[4-6]. 인터페이스 회로는 데이터 전송 latency에 영향을 주어 전체 시스템 성능을 저하시킬 수 있으므로 대부분 매뉴얼로 설계되고 있는 실정이다. 인터페이스 회로의 매뉴얼 설계는 프로토콜을 분석하고 IP의 동작 속도와 특성을 고려하여 제약 조건에 만족하는 설계를 하여야 하므로, 다양한 IP 모듈의 모든 인터페이스 회로를 매뉴얼로 작성하는 것은 긴 설계 시간과 비용을 요구한다. 이의 극복을 위하여 인터페이스 회로를 자동으로 생성할 수 있는 자동 생성 시스템이 요구된다[7-9]. 서로 다른 프로토콜을 갖는 모듈간 인터페이스 회로의 매뉴얼 설계는 각 모듈의 동작 프로토콜을 기술한 데이터 쉬트를 기반으로 통신 프로토콜을 분석하여 신호간 인과 관계를 찾아 한 모듈에서 발생된 신호에 대하여 통신코자 하는 모듈에 인과관계가 있는 신호로 변환하여 해당하는 포트에 신호를 인가할 수 있는 로직을 구현하는 것이다[6]. 인과관계가 있는 신호가 데이터 전송 latency를 최소화 하는 타이밍에 출력되기 때문에 시스템의 동작 속도를 향상 시킬 수 있는 장점이 있으나 사용되는 IP에 대해 모든 인터페이스 회로를 설계해 주어야 하는 단점이 있다. 인터페이스 회로의 자동생성을 위해선 신호의 특성을 분석하고 신호 간 인과관계에 대한 정보를 찾아내는 알고리듬이 필요하다. 임의의 프로토콜을 갖는 IP 모듈간 포트 이름이 상이하고 동일 이름을 갖는 포트간에도 요구되는 신호의 특성이 다를 수 있어 주어진 프로토콜 정보만으로 해당 포트에서 나오는 신호의 특성을 분석하는 알고리듬의 구현은 매우 어렵다.

UC Berkeley의 Sangiovanni-Vincentelli는 각 타겟 모듈의 프로토콜을 제어하는 FSM을 이용하여 인터페이스를 생성하는 알고리듬을 제안하였다[10]. 프로토콜을 제어하는 FSM의 모든 상태를 통신하고자 하는 모듈의 프로토콜 제어 FSM에 해당하는 모든 상태와 연결한 후, 각 노드에서 분기될 가능성이 없는 노드를 삭제하는 방식으로 인터페이스를 생성하였다. 단일 FSM으로 인터페이스 회로가 생성되므로 적은 면적의 인터페이스 회로를 얻을 수 있으나, 노드간 모든 분기 가능성을 분석해야 하므로 FSM의 상태가 많아질수록 자동생성 복잡도가 높아진다는 문제점을 갖는다.

UC Irvine의 D. Gajski는 프로토콜을 PSG(Protocol Sequence Graph)를 이용하여 기술하였으며, 각 모듈의 프로토콜과 통신할 수 있는 듀얼 PSG 기술을 IP 설계자에게 요구하였다[11]. 각 모듈의 데이타 write/read 동작에 대한 듀얼 PSG 정보를 가지고 있어 모듈이 데이터 WRITE/READ 동작을 하기 위해선 모듈의 데이터 WRITE 시퀀스에 있을 때 통신하고자 하는 모듈의 데이터 READ 시퀀스를 동작시키게 되며 데이터 READ 시퀀스에 있을 때 모듈의 데이터 WRITE 시퀀스를 동작시킨다. 데이터를 전송하고 하는 모듈이 데이터 WRITE 시퀀스에 있을 때 통신하고자 하는 모듈의 데이터 READ 시퀀스를 동작하게 하므로 실제적으로 데이터가 전송되는 시간은 통신하고자 하는 모듈의 데이터 READ 시퀀스의 동작 중 데이터를 전송하는 상태가 되어야 하므로 데이터 전송 지연이 발생한다. 뿐만 아니라 데이터 전송 시간의 차이를 극복하기 위해 버퍼를 사용하여 버퍼에 데이터를 입력하고 출력하는 동작이 소비하는 지연도 전체적인 데이터 전송 latency를 크게 만든다.

기존에 제시한 인터페이스 구조는 버스의 동작속도 보다 느린 슬레이브 모듈에 대해 듀얼 버퍼를 사용하여, 동작 클럭 타이밍의 차이를 극복했으며 버퍼의 데이터를 슬레이브 모듈이 전송받는 시간동안의 버스 점유 시간을 절약하여 전체적인 버스 사용 효율을 증가시켰으나 버퍼에 데이터를 쓰고 읽는 사이클이 추가적으로 요구되므로 데이터의 전송 사이클은 두 사이클 이상의 부가적인 지연을 갖게 된다[11].

멀티미디어, DMB 및 이미지 처리를 수행해야 하는 고성능 모바일기기에 사용되는 SoC는 데이터 처리량이 많은 프로그램을 실시간으로 동작시키기 위해 다중 프로세서의 사용이 필수이다[12]. 버스 속도 이상의 고성능으로 동작하는 프로세서에 버퍼로 구성된 인터페이스 회로를 사용할 경우, 버퍼에 데이터를 저장하는 사이클과 출력하는 사이클이 추가적으로 소비되어 데이터 전송 latency가 증가되게 되며 증가된 데이터 전송 latency는 시스템 전체의

성능을 저하시킨다.

본 논문에서는 상이한 프로토콜을 갖는 모듈 간 동작 시퀀스의 공통점을 접합하는 방법을 사용하여 데이터 전송 latency를 최소화하는 인터페이스를 자동생성 하는 기법에 대해 제안한다. 2절에서는 인터페이스 회로와 자동생성기의 구조를 설명하고, 3절에서는 IP의 프로토콜 기술 방식 및 기술된 프로토콜을 이용하여 인터페이스 회로를 생성하는 알고리듬을 제시한다. 4절에서는 제안된 알고리듬을 이용하여 생성된 인터페이스 회로와 매뉴얼로 작성된 인터페이스 회로 및 버퍼를 사용하여 설계한 인터페이스 회로간의 성능을 비교하며, 마지막으로 5절에서는 결과 및 추후과제를 제시한다.

Ⅱ. 시스템 개관

IP에 기반한 SoC 설계는 그림 1과 같이 각 기능을 하는 블록을 기존에 설계 검증된 IP를 사용하여 구성한다[13].

기존에 설계되어진 IP의 프로토콜은 시스템에서 사용하고자 하는 버스의 프로토콜과 상이하므로 데이터 송/수신을 위해선 IP의 프로토콜을 버스의 프로토콜에 맞게 변형시켜주어야 하는 인터페이스 회로가 요구되어진다[4-6]. 시스템의 다양성 및 time-to-market에 만족하기 위해 IP 사용이 증가하며, 이에 따른 인터페이스 회로 설계비용도 증가하게 되므로

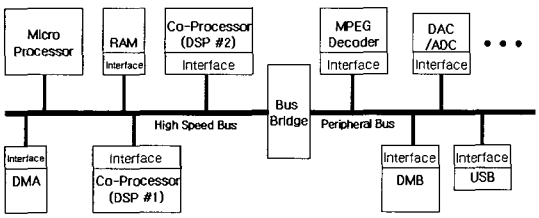


그림 1. IP 기반으로 설계된 시스템.

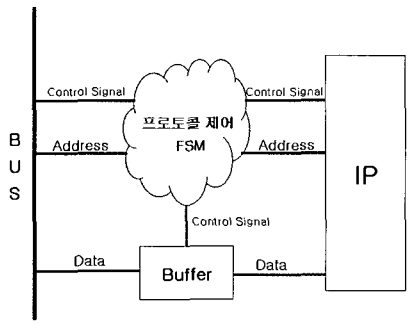


그림 2. 전형적인 인터페이스 회로

인터페이스 회로 설계비용을 감소하기 위해 인터페이스 회로 자동생성 기법이 요구된다[7-9]. 그림 2는 전형적인 인터페이스 회로 구조를 보인다[14].

동작속도가 상이한 버스와 IP간의 인터페이스 회로는 FSM을 통해 통신에 필요한 적절한 제어신호를 출력해 주며 동작속도의 상이함을 극복하기 위해 버퍼에 데이터를 저장하는 방식을 사용한다. 버퍼를 사용하여 동작속도가 상이한 IP간의 통신을 하는 인터페이스 구조는 2003년 AMBA V3.0 AXI (Advanced Microcontroller Bus Architecture V3.0 Advanced eXtensible Interface) 프로토콜에서도 제안된 바 있다[15]. 그림 2와 같은 전형적인 인터페이스 회로를 자동생성하기 위해선 버스 및 IP 입출력 포트의 신호 특성 및 신호 간 상관관계를 분석하여 각 상태에 맞는 제어신호를 출력해 주는 FSM을 생성해야만 한다. 각 포트에서 입출력되는 신호의 특성을 분석하는 것은 각 포트별 이름이 상이하고 동일 이름의 포트라 하더라도 각 모듈에 따른 신호 특성이 상이함으로 어려움이 따르게 된다. 이러한 신호간 상호관계 분석의 어려움을 극복하고자 본 연구실에서 그림 3의 (a)와 같은 구조를 제안한 바 있다[16].

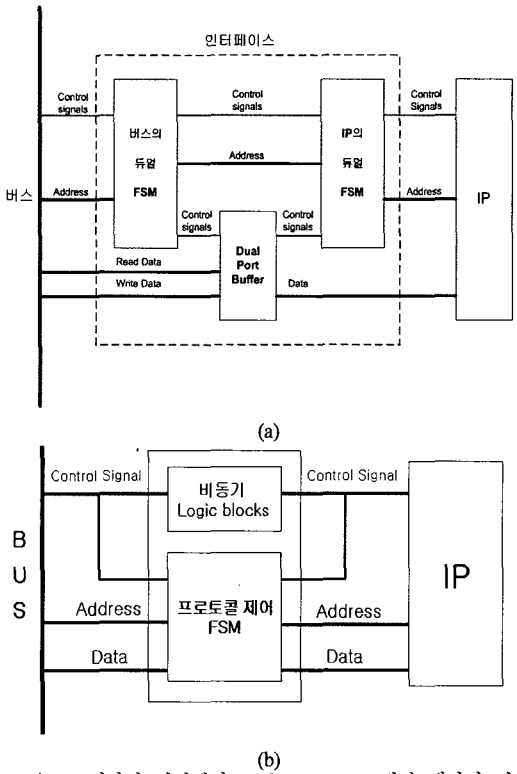


그림 3. 제안된 인터페이스 구조. (a) [16]에서 제안된 인터페이스 구조, (b) 본 논문에서 제안한 인터페이스 구조.

각 모듈과 통신할 수 있는 구조의 듀얼 FSM을 자동생성 한 후 간단한 FSM 제어신호를 연결하여 인터페이스 회로를 생성하였다. 각 모듈과 통신할 수 있는 듀얼 FSM을 자동생성하는 기법을 제안하여 포트간 신호의 인과관계를 분석해야 하는 어려움을 극복하였다. 그림 2와 그림 3-(a)의 인터페이스 구조는 버퍼를 사용하여 버스보다 느린 속도로 동작하는 모듈에 대해 데이터 전송 효율을 증가시킬 수 있게 되어있으나 IP의 동작 속도가 버스의 동작속도와 동일하거나 그 이상일 경우에는 오히려 데이터 전송 latency를 증가시키는 문제점이 발생한다.

그림 1의 SoC 시스템에서 저속으로 동작하는 모듈의 인터페이스 회로가 그림 3의 (a)에서 제시한 구조를 가진다고 가정할 경우, 버스에서 IP로 데이터가 전송되는 동작은 내부 버퍼에 버스의 속도로 데이터가 저장된 후 IP의 동작 속도로 버퍼에 있는 데이터가 출력되어 IP에 전송되게 된다. 전송하고자 하는 데이터는 IP에 버스 속도로 전송되는 것과 동일하게 동작하는 이점을 갖는다. 버스의 최대 동작 속도를 결정하는 저속 동작 모듈들이 버퍼를 가진 인터페이스 회로의 사용으로 시스템 버스의 동작 속도에 제약 사항이 되지 않으므로, 고성능 모듈의 동작 속도가 버스의 동작 속도를 결정하게 된다. 고속으로 동작하는 모듈의 인터페이스 회로를 그림 2, 그림 3의 (a)와 같은 구조로 한다면 데이터를 내부 버퍼에 저장하고 출력해야 하는 동작 사이클이 데이터 전송 속도에 부담으로 작용하여 전체적인 시스템 성능을 저하시킨다. 본 논문에서는 전체 시스템 속도 향상을 위해 고속으로 동작하는 마스터 모듈의 인터페이스 회로 구조를 그림 3의 (b)와 같이 단일 FSM 형태로 제안하여 자동생성 시스템을 구축한다. 각 모듈의 포트에서 출력되는 신호 간 상호 관계를 분석하는 어려움을 극복하기 위해 프로토콜마다 가지는 데이터 송/수신시 행하는 공통된 동작 시퀀스를 결합하여, 공통된 동작 시퀀스를 중심으로 통신에 필요한 제어신호를 출력하도록 하였다. 인터페이스 회로 내에 버퍼를 사용하지 않아 데이터 전송 latency에 가중되는 버퍼 동작 사이클을 감소시켰으며, 버스와 IP간 직접 변경이 가능한 신호에 대해선 비동기로직을 사용하여 동기식 신호변환에 따른 지연을 감소시켰다.

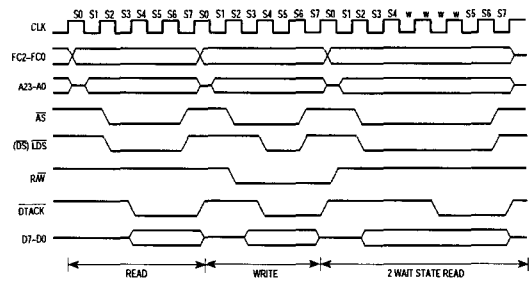
제안한 알고리듬을 사용하여 최소 데이터 전송 latency를 갖는 인터페이스 회로를 자동생성 하였다.

Ⅲ . Protocol Mapping을 이용한 인터페이스 자동생성

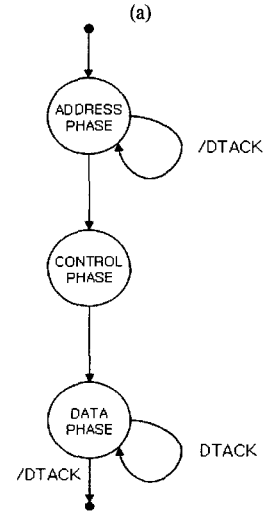
본 절에서는 마스터 형태 모듈의 프로토콜 기술로부터 인터페이스 회로를 자동생성하는 과정을 설명한다. 최초 마스터 형태 IP의 프로토콜을 기술하는 방법과 이를 사용하여 버스의 PSG와 비교하며, 공통된 동작 시퀀스를 찾아 인터페이스 회로인 제어 FSM을 자동생성해 낸다.

3.1 프로토콜 기술

IP의 동작을 나타내는 타이밍 다이어그램을 기반으로 인터페이스 회로 자동생성기가 인지할 수 있는 문자열 형태로 기술해 주어야 한다. 본 논문에서 제안한 인터페이스 자동생성기는 IP 프로토콜 동작 기술로써 PSG(Protocol Sequence Graph)를 이용하였다[11]. 그림 4는 Motorola 사의 MC68000 프로세서의 타이밍 다이어그램을 PSG 형태로 표현한 것을 보인다.



(a)



(b)

그림 4. 프로토콜 기술. (a) MC68000의 타이밍 다이어그램, (b) PSG형태로의 표현

PSG는 동작 상태를 시간의 흐름에 따라 기술한 것으로 각 노드별 분기 에지가 하나만 존재하는 단방향 스테이트 머신이다. 타이밍 다이어그램에 기반하여 입/출력 신호에 따른 상태를 정의한 후 시간의 흐름에 따라 변화하는 상태도를 기술한다. 타이밍 다이어그램은 시간에 따라 순차적으로 변하는 입출력 신호에 대한 정보를 나타내므로, 동일동작을 연속으로 수행하는 상태외의 동작상태는 순차적인 상태의 진행으로 나타낼 수 있다. 순차적인 상태로의 진행이 아닌 특정한 상태로의 천이가 존재할 경우 피드백 되는 상태들을 펼쳐(unfolding) 단방향 천이 상태로 변환하여 표현할 수 있다.

3.2 공통된 동작 Sequence 분석

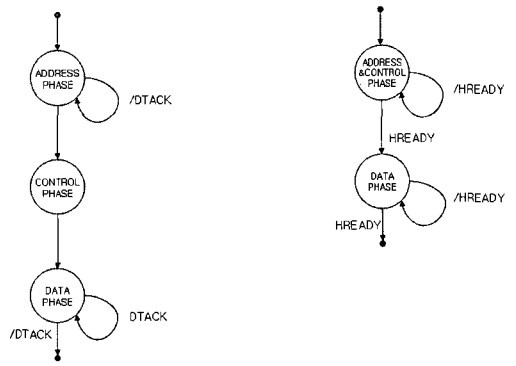
시스템 설계시 슬레이브 모듈의 선택 신호는 직접선택 방법을 사용하거나 주소를 이용한 디코딩 방법을 사용한다[17]. 사용되는 슬레이브 모듈의 칩선택 신호가 마스터 모듈의 I/O로 직접 연결될 수 있을 만큼 적은 슬레이브 모듈이 존재할 경우에는 직접선택 방법을 사용하지만, 시스템의 복잡도가 증가함에 따라 사용되는 슬레이브 모듈의 수가 증가하면 칩 선택 신호로 주소를 디코딩하는 방법이 사용된다. 주소 디코딩 방식으로 구성된 시스템에서 마스터 모듈의 데이터 송/수신 동작은 주소를 출력하여 슬레이브 모듈을 선택하는 사이클과 선택된 모듈에 데이터를 전송해 주는 사이클로 구성된다. 뿐만 아니라 버스의 데이터 전송 프로토콜도 주소 출력 사이클과 데이터 출력 사이클이 존재 한다. 데이터 통신을 위해 가장 중요시 되며 정보의 손실이 없어야 하는 사이클은 주소와 데이터를 전송하는 사이클이며 마스터 모듈과 버스의 프로토콜에 공통으로 존재한다. 데이터 통신을 위해 공통으로 존재하며 가장 중요시 되는 주소 전송 사이클과 데이터 전송 사이클은 상이한 프로토콜 간에 공통점이 되며, 이를 중심으로 프로토콜간 PSG의 정렬을 가능하게 한다.

3.3 인터페이스 회로 자동생성

PSG로 기술된 마스터 모듈의 프로토콜과 버스의 프로토콜은 상호 주소 전송 사이클과 데이터 전송 사이클이 존재하는 것을 알 수 있다.

그림 5는 마스터 모듈로써 Motorola 사의 MC 68000 프로세서와 AMBA AHB 버스의 데이터 write 프로토콜을 PSG로 나타낸 것이다.

인터페이스 회로는 MC68000 프로세서에서 요구되는 상태 천이 신호를 발생시켜 줘야 하며 프로세서가 출력하는 주소 및 데이터를 버스의 동작 사이클에 맞게 전송시켜 주어야 한다. 데이터 전송 latency를 줄이기 위해선 프로세서에서 출력되는 데이터의 지연을 최소화하여 버스로 전송해야 하므로 프로세서의 데이터 전송 사이클과 버스의 데이터 전송 사이클을 동일 시간대에 수행하도록 제어해야 한다. 그림 6은 데이터 전송 사이클을 동일 사이클에 수행하도록 정렬한 PSG를 나타낸다.



(a) (b)

그림 5. 프로토콜 기술. (a) MC68000 프로토콜, (b) AMBA AHB 프로토콜.

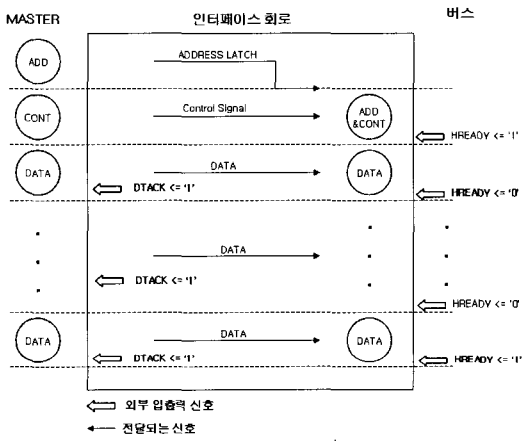


그림 6. 데이터 전송 사이클의 정렬

상호 데이터 전송 사이클을 동일 사이클로 정렬하여 마스터 모듈에서 출력되는 데이터를 지연 없이 버스에 전송할 수 있게 한다. 주소의 전송 사이클은 서로 다른 사이클에서 수행되는 것을 볼 수 있으며, 마스터 모듈의 주소 전송 사이클과 버스의 주소 전송 사이클의 차이만큼을 랫치 시켜 주게 한다. 버스의 상태를 지연하는 입력신호는 마스터 모듈의 동일 사이클에 해당하는 상태를 지연하는 신호로 사용되며 신호의 전달과정에서 발생할 수 있는 부가적인 지연을 최소화하기 위해 비동기식 로

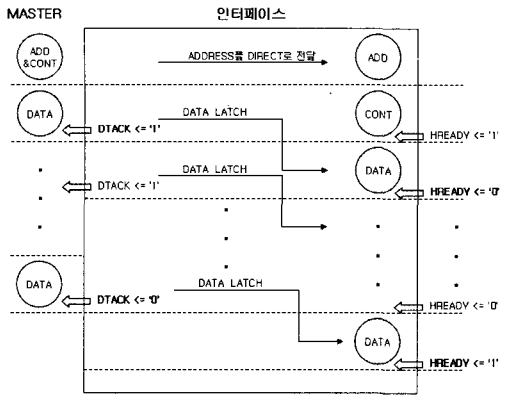


그림 7. 주소 전송 사이클을 동일하게 정렬한 경우.

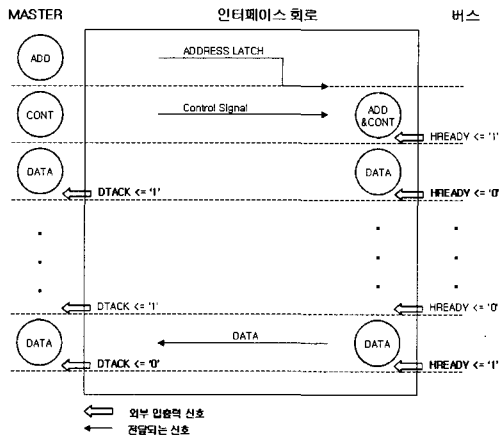


그림 8. 데이터 Read 동작시 PSG 정렬.

직을 사용하였다. 버스의 상태가 지연될 때 마스터 모듈의 상태도 동일하게 지연되므로 주소를 랫치하는 사이클은 버스의 상태가 지연되는 사이클에 마스터 모듈과 버스의 주소 상태 사이클의 차이만큼이 된다. 그림 6에서 데이터 전송 사이클을 동일하게 정렬했을 경우 마스터 모듈의 주소 전송 사이클이 버스의 주소 전송 사이클 이전에 존재하므로 주소의 랫치가 가능하지만, 그림 7과 같이 마스터 모듈의 주소 전송 사이클이 버스의 주소 전송 사이클 이후에 존재한다면, 주소 정보의 손실을 막기 위해 데이터 전송 사이클을 동일하게 정렬하지 않고 주소 전송 사이클을 동일하게 정렬해야 한다.

주소가 출력되는 상태에서 마스터 모듈에 출력되는 주소는 버스에 지연없이 직접 출력되며 데이터의 출력은 데이터 출력 상태 차이와 상태 지연 신호가 입력되는 시간만큼 랫치되어 출력되게 된다. 이 경우 데이터 출력이 지연되므로 데이터 전송 latency가 증가하는 것을 알 수 있다.

슬레이브 모듈에서 데이터를 읽어 오는 경우에는 그림 8와 같이 버스에서 데이터가 입력되는 사이클과 마스터 모듈의 데이터 입력 사이클이 동일 사이클로 정렬되거나 버스에서 데이터가 입력되는 사이클이 선행되어야 한다.

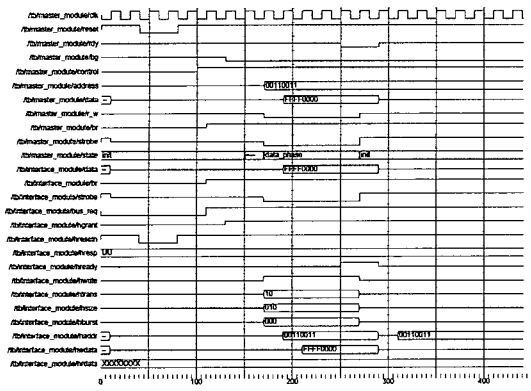
버스의 데이터 입력 사이클과 마스터 모듈의 데이터 입력 사이클을 동일 사이클에 정렬했을 경우, 마스터의 주소 전송 사이클이 버스의 주소 전송 사이클 이전에 수행된다면, 주소를 랫치해서 출력할 수 있다. 마스터 주소 전송 사이클이 버스의 주소 전송 사이클 이후에 수행 된다면, 데이터 입력 사이클로 동일하게 정렬하지 않고 주소 전송 사이클을 기준으로 정렬하게 된다. 이 경우 버스의 데이터 전송 사이클이 마스터 모듈의 데이터 입력 사이클 이전에 수행 된다면 마스터 모듈의 데이터 입력 사이클까지 데이터를 랫치하여 전송해 주며, 이후에 수행 된다면 마스터 모듈의 동작 상태를 지연 또는 현 상태를 유지하도록 제어 신호를 출력하여 데이터를 전송받을 수 있게 한다. 이때, 마스터 모듈에서는 부가적인 데이터 전송 latency를 갖게 된다.

입력되는 프로토콜의 PSG 기술을 사용하여 데이터 write일 경우와 read일 경우에 동작되는 FSM을 각각 따로 생성하며 마스터 모듈의 데이터 write/read 동작을 구분하는 신호에 따라 동작되어지는 FSM을 선택하게 된다. 데이터 write/read 동작의 구분 신호가 주소를 출력하는 상태와 동일하거나 이후 사이클에 존재한다면, 분기 신호가 존재하기 이전까지는 동일한 상태를 수행하며, 분기 신호 입력 이후에 각기 다른 상태를 수행하게 한다.

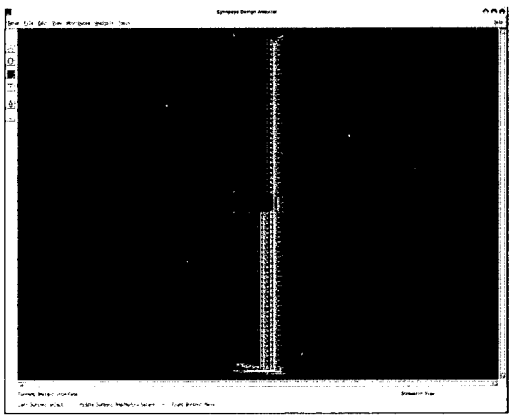
위와 같은 알고리듬을 사용하여 최종 생성되는 인터페이스 회로는 버스의 프로토콜에 맞는 신호를 출력하며 마스터 모듈의 상태를 제어하는 신호를 출력해 주는 단일 FSM으로 생성되게 된다.

Ⅳ. 실험결과

본 시스템은 마스터 모듈의 프로토콜을 PSG형태로 기술하여 입력하도록 하였으며 결과 인터페이스 회로의 VHDL 코드를 출력하도록 구현하였다. 출력된 VHDL 코드를 UNIX 환경에서 ModelSim SE PLUS 5.8b를 사용하여 시뮬레이션 하였으며, Hynix 0.35 um 공정 라이브러리를 사용한 Synopsys의 Design Analyzer로 합성하였다. 그림 9는 TI사의 TMS320 DSP칩의 프로토콜을 이용하여 AMBA AHB 버스의 프로토콜로 변환하여 주는 자동생성된 인터페이스



(a)



(b)

그림 9. 자동생성된 인터페이스 회로 검증. (a) Modelsim을 이용한 시뮬레이션 결과 파형, (b) Design Analyzer를 이용한 합성 파형.

회로의 합성된 결과 및 시뮬레이션 파형을 보여준다.

본 시뮬레이션은 마스터 모듈에서 임의의 주소값에 32비트 데이터 값을 전송하는 동작을 검증하였다. 마스터 모듈의 버스 사용 요청 신호 'br'은 AMBA AHB 버스의 버스 사용 신호인 'bur-req' 신호로 변화되었으며 마스터의 데이터 read와 write를 결정하는 'r_w' 제어 신호는 AMBA AHB 버스의 'hwrite', 'htrans','hsize','hburst' 신호로 변환되어 출력되는 것을 보인다. 슬레이브 모듈의 상태를 체크하는 버스의 'ready' 입력 신호는 마스터 모듈의 'rdy' 신호로 변환되어 마스터 모듈로 입력되는 동작을 보인다. 데이터 전송 latency를 최소화하기 위해 제어를 위한 입/출력 신호의 변환 과정은 동작 클럭에 동기하여 처리하지 않고 비동기로 처리하였다. 결과 마스터 모듈에서 출력하는 주소값, 데이터 값 및 제어신호가 인터페이스 회로에 입력되어 AMBA AHB 프로토콜에 맞는 신호로 변환되어 출력하는 것을 볼 수 있다.

표 1. 결과의 면적 비교

모듈명	매뉴얼 설계	버퍼를 사용한 설계	자동설계
TMS320	569	944	409
MC68000	692	1173	612
ADSP-2103	631	1023	596
평균	630.67	1,046.67	539.00

구축한 시스템에서 생성한 인터페이스 회로 모듈과 Altera 사의 Excalibur 설계 예제에서 지원하는 구조와 동일하게 설계한 인터페이스 회로 및 본 연구실에서 제안한 버퍼를 사용한 인터페이스 회로 간의 성능을 비교하였다[18, 19].

표 1은 TI사의 TMS320 DSP, Motorola 사의 MC68000 및 ANALOG DEVICES사의 ADSP-2103 칩의 프로토콜을 타겟으로 설계 및 생성한 인터페이스 모듈 코드를 Hynix 0.35 um 공정 라이브러리를 사용한 Synopsys의 Design Analyzer로 합성하여 면적을 비교하였다. 면적 측정 단위는 2-input NAND 게이트를 1로 하였다.

표 1은 TI사의 TMS320 DSP, Motorola사의 MC 68000 및 ANALOG DEVICES사의 ADSP-2103칩의 프로토콜을 타겟으로 설계 및 생성한 인터페이스 모듈 코드를 Hynix 0.35 um 공정 라이브러리를 사용한 Synopsys의 Design Analyzer로 합성하여 면적을 비교하였다. 면적 측정 단위는 2-input NAND 게이트를 1로 하였다.

위의 인터페이스 회로 중 버퍼를 사용한 설계는 버퍼로 인한 과도한 면적차이를 최소화하기 위하여 버퍼의 용량을 2 word (2X32 bit)로 제한하였다. 결과에서 볼 수 있듯이 자동 생성된 인터페이스 회로는 매뉴얼로 설계한 인터페이스 회로와 버퍼를 사용하여 설계한 인터페이스 회로에 비해 각각 평균 14.5%, 48.5%의 면적 감소를 보인다. 버퍼를 사용하여 설계한 인터페이스 회로는 버스의 프로토콜을 제어하는 FSM 외에 버퍼 및 마스터 모듈의 프로토콜을 제어하는 듀얼 FSM이 추가로 부과되어 단일 FSM으로 구성된 자동생성 인터페이스 회로와 큰 면적차이를 보인다. 매뉴얼로 설계된 인터페이스 회로는 입/출력 신호를 동기식으로 처리하기 때문에 비동기식으로 처리하는 자동생성 인터페이스 회로에 비해 추가적인 동작 상태가 부과되어 상호 면적 차이를 보인다. 각 인터페이스 모듈은 전체 시스템 관점에서 미비한 면적을 차지하므로 수치상의 면적 차이는 큰 의미가 없다.

표 2. 결과의 단일 데이터 전송 latency 비교

모듈명	매뉴얼 설계	버퍼를 사용한 설계	자동설계
TMS320	4	7	3
MC68000	3	6	2
ADSP-2103	4	7	3
평균	3.67	6.67	2.67

데이터 전송 latency는 50 Mhz의 클럭을 사용하여 버스와 마스터 모듈을 구동시켰을 경우 클럭을 기준으로 마스터 모듈에서 데이터 전송이 시작한 시점부터 버스에 전송이 종료되는 시점까지의 소비된 클럭 수를 측정하였다. 이때 데이터를 수신해야 할 슬레이브 모듈은 항상 데이터를 받을 수 있는 상태로 가정하였다. 마스터 모듈에서 임의의 주소에 단일 워드(32 bit) 데이터를 전송하였을 경우의 latency를 측정하여 표 2에 제시하였다.

자동 생성된 인터페이스 회로는 상태 천이 신호를 비동기식으로 전달하기 때문에 동기식으로 전달하는 매뉴얼로 설계한 인터페이스 회로에 비해 적은 데이터 전송 latency를 보인다. 타겟 모듈 및 버스의 프로토콜이 복잡해지고, 동작 상태 및 천이 신호가 증가할 경우 천이 신호의 동기식처리로 인해 소요되는 클럭 사이클이 증가하므로 비동기로 처리하는 자동생성된 인터페이스 회로의 이점이 증가하게 된다. 버퍼를 사용한 인터페이스 회로는 버퍼에 데이터를 입력한 후 출력하는 구조를 가지고 있기 때문에 버퍼 동작 사이클만큼의 데이터 전송 latency가 증가하게 된다. 그림 10은 각기 다른 구조로 설계된 인터페이스 회로의 동작 타이밍 다이어그램을 보여 데이터 전송 latency가 다른 이유를 설명한다.

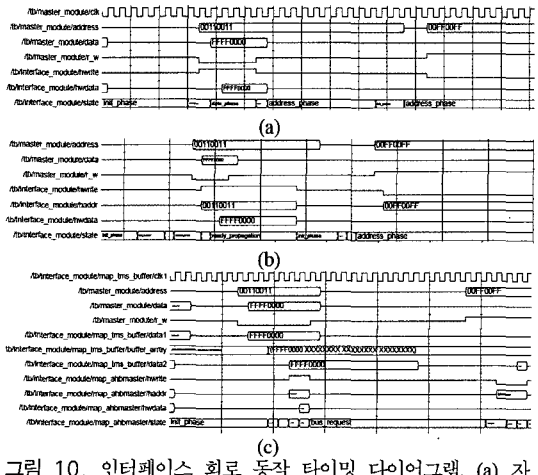


그림 10. 인터페이스 회로 동작 타이밍 다이어그램. (a) 자동설계, (b) 매뉴얼설계, (c) 버퍼를 사용한 설계.

표 3. 결과의 버스트 모드 데이터 전송 latency 비교

모듈명	매뉴얼 설계	버퍼를 사용한 설계	자동설계
TMS320	44	38	33
MC68000	42	37	32
ADSP-2103	44	38	33
평균	43.33	37.67	32.67

자동설계 된 인터페이스 회로는 마스터 모듈의 'r_w' 신호를 버스의 'hwrite' 신호로 변환하여 출력하는 동작을 비동기로 처리하여 동일 사이클에 수행하는 반면 매뉴얼 설계된 인터페이스 회로는 동기식으로 처리하므로 'r_w'신호가 입력되면 다음 상태로 천이하여 'hwrite' 신호를 출력하게 된다. 버퍼를 사용한 인터페이스 회로는 마스터 모듈에서 출력한 데이터를 버퍼에 저장한 후 로딩하여 버스에 출력하므로 제어신호의 입출력 지연뿐만 아니라 데이터의 전송 지연이 부과되게 된다. 결과 자동 설계된 인터페이스 회로의 데이터 전송 latency는 표 2에서 본 것과 같이 다른 구조를 가진 인터페이스 회로들에 비해 적다는 것을 알 수 있다.

표 3에는 16개의 32비트 데이터를 버스트로 전송시 전송 latency를 비교하였다.

버스트 모드 전송 시 전송 latency는 매뉴얼 설계된 인터페이스 회로와 버퍼를 사용한 인터페이스 회로에 비해 각각 평균 24.7%, 13.3%의 감소를 보였다. 자동 설계된 인터페이스 회로는 단일 데이터 전송 모드와 동일하게 제어신호가 비동기로 동작하는 장점을 가지고 있으므로 버스트 모드 데이터 전송 시에도 다른 구조를 가진 인터페이스 회로에 비해 적은 데이터 전송 latency를 갖는 것을 볼 수 있다. 버퍼를 사용한 인터페이스 회로는 버퍼에 데이터가 순차적으로 입력됨과 동시에 버스에 출력되므로 초기 버퍼에 데이터가 전송되는 시점까지의 지연만이 추가적으로 소요된다. 반면 매뉴얼 설계된 인터페이스 회로는 마스터의 상태를 천이시키는 신호가 입력신호에 의해 동기식으로 동작하므로 한 사이클 지연된다. 버스트 모드 전송 시 한 사이클씩의 지연이 누적되어 자동설계된 인터페이스 회로 및 버퍼를 사용한 인터페이스 회로에 비해 큰 데이터 전송 latency를 갖는 것을 볼 수 있다.

자동생성된 인터페이스 회로는 매뉴얼로 설계된 인터페이스 회로 및 버퍼를 사용한 인터페이스 회로와 성능 비교한 결과 면적은 각각 평균 14.5%, 48.5%의 감소를 보이고 50Mhz로 동작하는 버스와 마스터 모듈로 구성된 시스템에서 단일 데이터 전송 latency는 각

각 평균 27.3%, 59.1%의 성능 향상을 보이며 버스트 모드 데이터 전송 시 각각 평균 24.7%, 13.3%의 성능 향상을 보였다. 구현한 자동생성 시스템을 사용하여 매뉴얼 설계한 인터페이스에 준한 성능을 가진 인터페이스 회로를 생성해 낼 수 있었다.

V . 결론 및 추후과제

본 논문에서는 SoC 설계 환경에서 마스터 모듈형 고성능 IP의 사용으로 요구되는 적은 데이터 전송 latency를 가진 인터페이스 회로의 설계비용을 감소시키며 짧은 time-to-market의 만족을 위해 자동으로 인터페이스 회로를 생성하는 시스템을 구축하였다. 인터페이스 회로를 생성하기 위해 마스터 모듈의 프로토콜 PSG와 버스의 PSG를 정렬하여 요구되는 제어 신호들을 변화하여 주었으며, 데이터 전송 latency를 줄이기 위해 비동기 회로를 사용하여 신호들을 변환하였다. 제안된 방법으로 TI사의 TMS320 DSP 프로세서, Motorola사의 MC68000 프로세서 및 ANALOG DEVICES사의 ADSP-2103 프로세서에 대한 인터페이스 회로를 자동생성 한 결과 매뉴얼로 작성한 인터페이스 회로 및 버퍼를 사용한 인터페이스 회로에 비해 면적과 데이터 전송 latency가 감소한 인터페이스 회로를 얻을 수 있었다. 매뉴얼로 인터페이스 회로를 설계할 시 소요되는 많은 시간을 절약할 수 있다.

현재 본 연구실에서 구현되어 있는 인터페이스 자동생성 시스템은 버스와 상이한 클럭으로 동작하는 모듈에 대해 버퍼를 사용하여 버스 사용 점유율을 감소시키는 구조를 가진 인터페이스 회로와 본 논문에서 제안하고 있는 버스와 동일한 클럭으로 동작하는 모듈에 대해 최소의 데이터 전송 latency를 가진 인터페이스 회로를 자동생성 해 준다. 추후 HW IP를 사용할 경우 OS 수준에서 HW를 동작하기 위해 작성되어야 하는 디바이스 드라이버를 자동생성하는 시스템을 개발한다. 이미 설계 검증된 IP를 사용할 시 상이한 프로토콜을 맞추기 위해 필요한 인터페이스 회로와 OS 수준에서 동작시키기 위해 필요한 디바이스 드라이버를 자동생성 시스템을 이용하여 설계함으로써 설계비용을 낮출 수 있으며 짧은 time-to-market을 만족시킬 수 있는 파급효과를 가져올 것이다.

참 고 문 헌

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

[2] P. Chou, R. Ortega, and G. Borriello, "IPCHINOOK : An Integrated IP-based Design Framework for Distributed Embedded Systems", in Proc. Design Automation Conference, pp. 44-49, June 1999.

[3] S. Abdi, D. Shin, and D. Gajski, "Automatic Communication Refinement for System Level Design", in Proc. Design Automation Conference, Anaheim, CA, pp. 300-305, June 2003.

[4] R. Ortega, L. Lavagno, and G. Borriello, "Models and Methods for HW/SW Intellectual Property Interfacing", in Proc. System Synthesis, pp. 397-432, July 1998.

[5] A. Wenban, J. O'Leary, and G. Brown, "Codesign of Communication Protocols", IEEE Trans. Computer, Vol. 26 No. 12, pp. 46-52, Dec. 1993.

[6] P. Chou, B. Ortega, and G. Borriello, "Interface Co-Synthesis Techniques for Embedded System", in Proc. Int. Conf. CAD, pp. 280-287, Nov. 1995.

[7] R. Passersome, J. Rowson, and A. Sangiovanni-Vincentelli, "Automatic Synthesis of Interface between Incompatible Protocols", in Proc. Design Automation Conference, pp. 8-13, June 1998.

[8] E. Walkup and G. Borriello, "Automatic Synthesis of Device Drivers for Hardware/ Software Co-design", Technical Report #94-06-04, Univ. of Washington, Aug. 1994.

[9] D. Gajski, "IP-based Design Methodology", in Proc. Design Automation Conference, New Orleans, LA, June 1999.

[10] R. Passerone, L. Alfaro, A. Henzinger, and A. Sangiovanni-Vincentelli, "Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin", in Proc. Int. Conf. CAD, pp. 132-139, Nov. 2002.

[11] D. Shin and D. Gajski, "Interface Synthesis from Protocol Specification", Technical

Report CECS-TR-02-13, Univ. of California, April 2002.

[12] C. Ravikumar. "Multiprocessor Architectures for Embedded System-on-chip Applications", in Proc. Int. Conf. VLSI Design, pp. 512-519, Jan. 2004.

[13] M. Keating and P. Bricaud, *Reuse Methodology Manual for System-on-a-Chip*, Kluwer Academic Pub., 2002.

[14] A. Rajawat, M. Balakrishnan, and A. Kumar, "Interface synthesis: issues and approaches", in Proc Int. Conf. VLSI Design, pp. 92-97, Jan. 2000.

[15] AMBA AXI Specification, ARM Ltd., June 2003.

[16] 이서훈, 문종욱, 황선영, "FSM을 이용한 표준화된 버스와 IP간의 인터페이스 회로 자동생성에 관한 연구" 한국통신학회 논문지, 제30권, 2A호, pp. 137-146, 2005년 2월.

[17] W. A. Triebel and A. Singh, *The 8088 and 8086 Microprocessors: Programming, Interfacing, Software, Hardware, and Applications*, Prentice Hall, 2000.

[18] Excalibur Hardware Design Tutorial, Available: http://www.altera.com/literature/lit-exc.jsp, Altera Inc., Aug. 2002.

[19] AN 287: Design Files, Available : http://www.altera.com/literature/lit-exc.jsp, Altera Inc., Feb. 2003.

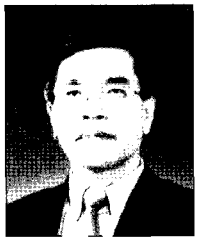
이 서 훈 (Ser-Hoon Lee) 준회원

2003년 2월 서강대학교 전자공학과 졸업
2005년 2월 서강대학교 전자공학과 석사학위 취득
2005년 3월~현재 서강대학교 전자공학과 대학원 CAD & Embedded Systems 연구실 박사과정
<관심분야> SoC 설계, IP Interface 자동설계기법

강 경 구 (Kyung-Goo Kang) 준회원

2005년 8월 서강대학교 전자공학과 졸업
2005년 9월~현재 서강대학교 전자공학과 대학원 CAD & Embedded Systems 연구실 석사과정
<관심분야> SoC 설계, HW/SW Partitioning

황 선 영 (Sun-Young Hwang) 정회원



1976년 2월 서울대학교 전자공학과 졸업
1978년 2월 한국 과학원 전기 및 전자공학과 공학석사 취득
1986년 10월 미국 Stanford대학 전자공학 박사학위 취득
1976년~1981년 삼성반도체 주식회사 연구원, 팀장
1986년~1989년 Stanford대학 Center for Integrated System 연구소 책임연구원
Fairchild Semiconductor Palo Alto Research Center 기술자문
1989년~1992년 삼성전자(주) 반도체 기술자문
1989년 3월~현재 서강대학교 전자공학과 교수
<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Computer Architecture 및 DSP System Design 등