논문 2010-47SD-1-13

성능 제약 조건 하에서의 SAMBA 형 MPSoC 버스 구조 최적화

(SAMBA Type MPSoC Bus Architecture Optimization under Performance Constraints)

김 홍 염*, 정 성 철*, 신 현 철**

(Hongyeom Kim, Sungchul Jung, and Hyunchul Shin)

요 약

최근 여러 개의 프로세서 및 메모리를 한 개의 칩에 구현하여 다양한 알고리즘을 구현하는 Multi-Processor System-on-Chip (MPSoC) 설계가 가능해지면서, 프로세서 간 interconnection을 최적화 하는 문제가 중요해졌다. Application에 따라서 최적 interconnection이 다르기 때문에, 체계적으로 다양한 사양에 적합한 interconnection 구조를 설계하는 방법이 필요하다. 본 논문에서는 프로세서가 4~16개 정도인 MPSoC application에서는 버스 구조가 적절한 점에 주목하여, 간단한 arbitration이 특징인 Single Arbitration Multiple Bus Accesses (SAMBA) 형 버스 구조를 이용하여, 다양한 application에 대한 성능 제약 조건을 만족하는 저비용 버스 구조를 찾는 새로운 방법을 제안하였다. 다양한 Application을 실험에 이용하여, 제안한 방법으로 성능 제약 조건 내에서 저비용 버스 구조를 찾았다. 같은 성능으로 최적화 전의 구조에 비해서 버스 분할에 필요한 로직 사용이 경우에 따라 약 50% 이상 감소한다. 또한 다양한 성능 조건에 대한 저비용 버스 구조를 찾을 수 있었다.

Abstract

Opimization of interconnects among processors and memories becomes important as multiple processors and memories can be integrated on a Multi-Processor System-on-Chip (MPSoC). Since the optimal interconnection architecture is usually dependent on the applications, systematic design methodology for various data transfer requirements is necessary. In this paper, we focus on bus interconnection for MPSoC applications which use 4 ~ 16 processors. We propose a new systematic bus design methodology under performance constraints using Single Arbitration Multiple Bus Accesses (SAMBA) style bus architectures. Optimized bus architecture is found to satisfy performance constraints for a single or multiple applications. When compared to the unoptimized architecture, our method can reduce the bus switch logic circuits significantly (by more than 50% sometimes). Furthermore, low cost bus architectures can be found to satisfy the performance constraints for multiple applications.

Keywords : MPSoC, Bus, Interconnect, On-Chip Communication, SAMBA

I. 서 론

최근 다양한 멀티미디어 시스템이 디지털화 및 고성능화 되면서, System-on-Chip (SoC) 시장에서는 더 높은 성능의 다양한 제품이 요구되고 있다. 이러한 시장에서 경쟁력을 갖기 위해서는 개발 기간의 단축과 전체 시스템 성능의 향상이 중요하다. 현재 SoC 산업에서는 하나의 집적회로에서 다양한 기능을 처리하는 고성능의

* 학생회원, ** 평생회원, 한양대학교 전자전기제어계측공학과
(Electric Engineering and Computer Science, Hangyang University)

※ 본 연구는 지식 경제부 출연금으로 ETRI 시스템 반도체진흥센터에서 수행한 IT-SoC 핵심설계인력 양성사업과 지식 경제부 및 정보 통신 연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2009-C1090-0902-0024)
접수일자: 2009년6월9일, 수정완료일: 2010년1월7일

제품 개발을 위해 여러 개의 프로세서 및 메모리를 한 개의 칩에 구현하여 다양한 알고리즘을 구현하는 MPSOC가 널리 사용된다.

MPSoC 설계에서 프로세서나 메모리 사이에서 데이터 통신 시, 병목 현상 등에 의해 Interconnection 구조가 전체 시스템 성능에 많은 영향을 미치기 때문에, 저비용 최적 설계가 중요하다.

Interconnection 구조로는 전통적으로 AMBA, Core-Connect, Silicon MicroNetwokrs 같은 하나의 공유된 채널을 이용하는 shared bus architecture를 많이 이용하였다. [1]에서 전반적인 SoC 버스 구조에 대해서 소개하고 있다.

Shared 버스 구조는 간단한 topology와 낮은 hardware cost, 확장성 등의 장점을 갖는다. 하지만 이런 형태의 버스 구조는 모든 통신을 하나의 공유된 채널을 이용하기 때문에, component가 늘어 날수록, 버스 구조에서 병목 현상이 발생되어 성능 저하가 크다. 이를 해결하기 위해 bandwidth를 증가시킨 hierarchical bus architecture[2], multiple access를 구현한 split bus architecture[3] 등이 개발되어 성능을 증가시켰다.

Bandwidth를 더욱 증가시키기 위해, crossbar switches나 point-to-point 형식의 connection을 사용하여 증가시킨다[4~5]. 이보다 더욱 확장성 있는 통신 구조로 Network-on-Chip (NoC) 또한 사용된다[6].

복잡한 버스 구조는 많은 면적을 차지하며, 시간이 오래 걸린다. 또한 많은 모듈을 연결할 경우, arbitration의 복잡도가 높아지고, 시간이 오래 걸린다. arbitration 시간이 길어질 수록, 성능이 감소된다. 이를 해결하기 위해 SAMBA 버스 구조가 제안되었다.

Single Arbitration Multiple Bus Accesses (SAMBA) 버스는 Multiplexer (MUX)를 이용하여 버스에서 모듈 사이를 여러 segments로 분할하였고, 자동적인 arbitration으로 conflict가 없는 여러 개 transaction의 동시 통신이 가능하다[7].

본 논문에서는 SAMBA style의 버스를 이용하여 다양한 application에 대해 저비용 버스 구조 설계 방법을 제안하였다. 하나의 버스로 요구되는 성능 조건을 만족하지 못한 경우에는, 두 개 이상의 버스들을 사용하여 성능 조건을 만족시킨다. 주어진 성능 제약 조건을 만족시키면서 최소 개수의 connection과 switch block을 사용함으로써 저비용 버스 구조를 설계한다.

본 논문의 Ⅱ장에서는 버스 구조를 설명하고, Ⅲ장에서는 성능 제약 조건 내에서 저비용 버스 구조를 찾는 방법을 설명한다. Ⅳ장에서는 실험에 사용된 application들을 소개한다. Ⅴ장에서는 실험 결과를 설명한다. 마지막으로, Ⅵ장에서 결론을 맺는다.

Ⅱ. 버스 구조

Shared 버스 구조는 간단한 topology와 낮은 hardware cost 등의 장점으로 인해 SoC 산업에서 Interconnection 구조로 많이 사용된다. 그러나 긴 wire로 인한 load capacitance가 크기 때문에 전력 소모가 크다. 또한 낮은 bandwidth 때문에 data transfer 시 delay가 크다. 또한 한 번에 한 모듈만 데이터를 전송할 수 있기 때문에, 버스에 많은 모듈이 연결될수록 모듈 당의 bandwidth가 감소한다.

Shared 버스 구조의 단점을 극복한 것이 Split 버스 구조다. 이 구조는 버스에서 모듈 사이를 MUX나 Switch 등으로 분리한 버스 구조다. Split 버스 구조는 각 모듈 사이가 모두 분리 되어 있어, 하나 이상의 모듈이 동시에 데이터 통신을 할 수 있다. 이로 인해 성능이 증가한다. 또한 분리된 구조로 인해 wire 길이가 줄어들면, Load Capacitance가 작아짐으로 소모 전력이 감소하고 데이터 통신 딜레이가 감소한다.

그림 1의 Single Arbitration Multiple Bus Accesses (SAMBA) 버스 구조는 MUX를 이용하여 버스를 분할하였고, 각 모듈은 interface unit을 통해서 버스와 통신한다. 버스는 forward sub-bus와 backward sub-bus로 구성되어 양 방향 통신이 가능하다.

Forward sub-bus나 backward sub-bus는 arbitration에 의해 버스 사용 권한을 받고, 데이터 통신을 한다. 이와 동시에, arbitration을 통해 버스 사용 권한을 얻지 못한 transaction이 분할된 버스 구조에 의해 다른 데이터 통신에 방해되지 않을 경우에는 동시에 통신한다.

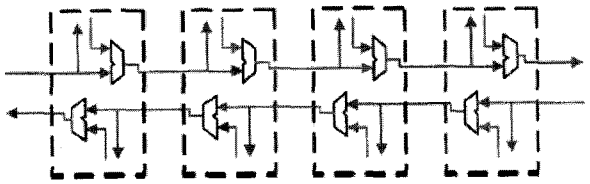


그림 1. SAMBA 버스 구조
Fig. 1. Structure of a SAMBA bus.

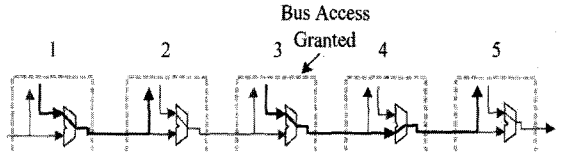


그림 2. 단일 arbitration을 이용한 다중 버스 접속
Fig. 2. Multiple bus accesses with single arbitration.

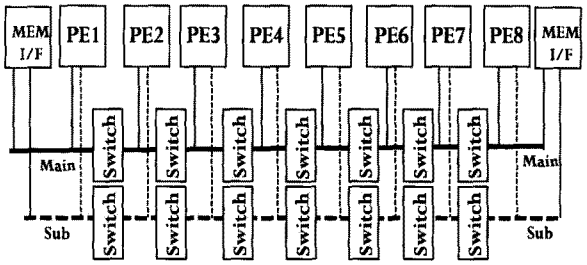


그림 3. 본 논문에서 이용한 기본 버스 구조
Fig. 3. Basic bus architecture in our paper.

그림 2는 arbitration을 통해 버스 사용 권한을 얻지 못한 transaction이 통신되는 예이다. 모듈 3번이 버스 사용 권한을 얻어서 모듈 5번과 통신하는 동시에, 모듈 1번이 모듈 2번과 통신 한다. SAMBA 버스 구조는 실험에서 3.5배 이상의 bandwidth 증가 효과를 갖고, 1.5배 이상의 데이터 통신 시간 감소 효과를 얻는다[7].

최근 SoC 산업에서는 다양한 프로세서와 메모리가 사용되면서 하나의 버스 구조로는 요구되는 성능을 만족하지 못하는 경우가 많다. 이 경우에 여러 개의 버스를 이용하여 성능을 만족한다. 그림 3의 버스 구조는 8개의 Processing Element (PE)와 2개의 메모리를 두 개의 버스를 이용하여 연결한 구조다. 이 구조는 SAMBA 버스 구조에서 각 PE 사이를 switch block을 이용하여 버스를 분할한다.

본 논문에서는 그림 3의 구조를 이용하여 실험한다. 그러나 제안한 버스 구조 최적화 방법은 SAMBA 버스 구조 외에 다른 버스 구조에도 사용 가능하다.

III. 최적 버스 구조 설계 방법

그림 4는 본 논문에서 제안한 버스 구조 최적화 과정이다. 프로세서와 메모리로 구성된 Design Specification을 받아서, 그림 3의 초기 버스 구조를 생성한다. 초기 버스 구조는 두 개의 버스를 이용한 구조로, 모든 PE가 두 개의 버스에 연결되고, 각 버스의 PE 사이에 switch block이 위치한 구조다. 그림 3에서 점선

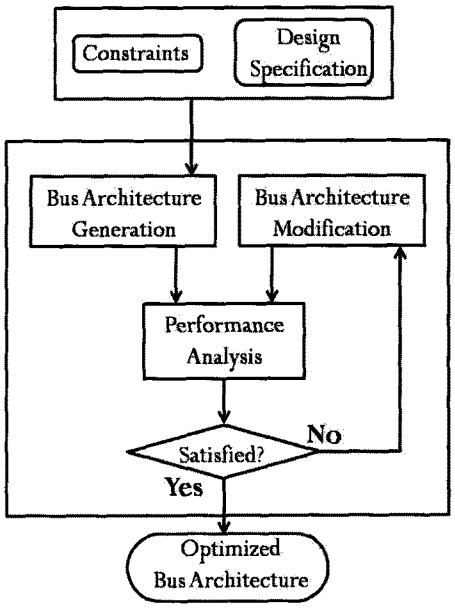


그림 4. 버스 구조 설계 방법
Fig. 4. Bus architecture design methodology.

으로 표시된 sub-bus와 PE의 connection과 switch block을 Removal과 Insertion 방법으로 저비용 버스 구조를 찾아서 두 방법을 비교하였다. Removal 방법은 그림 3의 버스 구조에서 connection을 하나 씩 제거하며 성능 측정한 뒤, switch block을 하나 씩 제거하여 성능 제약 조건 내에서 저비용 버스 구조를 찾는 것이다. 또한 Insertion 방법은 그림 3에서 sub-bus로의 connection이 없는 초기 버스 구조에 connection을 하나 씩 삽입하여 성능 사양을 만족시키도록 한 후, switch block을 모두 제거한 후, 사양을 만족할 때까지 하나 씩 삽입하며 성능 측정하여, 성능 제약 조건 하에서 저비용 버스 구조를 얻는 방법이다.

사용된 Removal 최적화 알고리즘은 다음과 같다.

< Removal >

1. Generate initial bus architecture;
2. Find the connection or switch sel, with minimal performance decrease when it is removed;
3. If the performance constraints can be met after sel is removed, then remove sel;
   Go to step 2;
4. Else output the optimized bus architecture;

한 개의 PE와 sub-bus 사이의 connection을 제거할 경우, PE의 sub-bus에 대한 interface module 및 다른

모듈과 통신을 위한 wire 등 한 개의 switch block을 제거했을 때보다 면적 감소가 크다. 또한 성능에 미치는 영향이 한 개의 switch block보다 connection이 미치는 영향이 크다. 따라서 보다 효과적인 최적화를 위해 connection을 먼저 제거 및 삽입하였다.

Removal 방법은 그림 3의 모든 PE가 sub-bus에 connection을 가지고, 가능한 모든 자리에 switch block이 존재하는 상태를 초기 버스 구조로 한다. 먼저, connection을 모든 위치에서 하나 씩 제거해보며 성능을 측정한다. 이 중 성능이 가장 적게 떨어지는 connection을 삭제한 뒤, 나머지 connection을 하나 씩 제거해보며 성능 측정한다. 이 과정을 성능 제약 조건을 만족하는 범위 내에서 반복하여, connection 사용을 최소화한다. 이 때, 하나 씩 제거해보며 성능 측정한 결과, 성능에 영향을 주는 정도가 같은 connection들이 하나 이상 존재하는 경우에는 lookahead 방법을 사용하여 선택한다. 예를 들어서, connection A와 connection B의 제거 비용이 같다고 하면, lookahead를 이용하여, A가 제거되었다고 가정하고 나머지 connection들 중 성능 저하가 최하인 connection을 제거 했을 때의 성능과 B가 제거 되었다고 가정하고 나머지 connection들 중 성능 저하가 최하인 connection을 제거 했을 때의 성능을 비교하여 우수한 성능을 가지는 connection으로 A, B 중 하나를 선택 제거 한다. 만약 Lookahead에서도 같은 성능을 갖는 connection이 하나 이상 발생할 경우에는 connection 중 보다 왼쪽에 있는 것을 제거한다.

성능을 만족시키기 위하여 더 이상 connection을 줄일 수 없게 되면, 최소화된 connection 구조를 이용하여, switch block을 connection을 제거한 방법과 같은 방법으로 제거한다.

Insertion 방법은 그림 3의 구조에서 모든 PE가 connection을 사용하지 않고, 가능한 모든 자리에 switch block이 존재하는 상태를 초기 버스 구조로 한다. 이 구조에서 성능 증가가 가장 큰 connection을 하나 씩 삽입하며, 성능 제약 조건을 만족할 때까지 반복한다. 이 때, 성능에 영향을 주는 정도가 같은 connection이 하나 이상 존재하는 경우에는 lookahead 방법을 사용하여 선택한다. 그 뒤, 최소화된 connection 구조에서, switch block을 모든 위치에서 제거한 뒤, connection을 삽입한 방법과 같은 방법으로 switch block을 삽입한다.

Ⅳ. Applications

제안한 설계 방법의 성능을 평가하기 위하여 3 가지 application에 대하여 실험하였다. 각 application은 8개의 PE와 2 개의 메모리를 이용하도록 하여, 그림 3의 버스 구조를 사용할 수 있도록 하였다.

1. Application A : Deblocking Filter

H.264 / AVC 디코더 중 Debloking Filter에서 1 Frame 처리하는 Data Transfer이다.

그림 5는 자료를 재사용하기 위해 인접한 블록 단위로 필터링하는 필터링 순서를 나타낸 그림이고, 그림 6은 PE 할당 및 데이터 전송을 나타내 그림이다[8]. PE 2, PE 4, PE 6, PE 8에서 그림 5의 ①, ⑤, ⑨, ⑬에 대한 수직 방향 필터링 연산을 위해 4 X 4 블록을 각각 받아서 연산을 수행한 뒤, ⓐ, ⓑ, ⓒ, ⓓ에 대한 수평 방향 필터 연산을 수행하기 위해 PE 1, PE 3, PE 5, PE 7에게 각각 데이터를 전송한다. 이와 동시에 수직 방향 필터 연산을 하는 PE 들은 ②, ⑥, ⑩, ⑭에 대한 수직 방향 필터링 연산을 위해 새로운 4 X 4 블록을 받아서 연산한다. 입력 영상의 포맷을 4:2:0 YUV 기준으로 한 픽셀 당 12bit의 정보를 가질 때, 4 X 4 블록의 전체 데이터 양은 192bit이다. AMBA 버스에서는 1 cycle에 32bit 데이터를 전송할 수 있다[9]. 따라서 4 X 4 블록 데이터 전송에 6 cycle이 필요하다. 또한 각 PE에서 필터링 연산에 소요되는 시간은 경계면의 강도 및 경계면 양쪽에 위치한 픽셀들의 관계에 따라 적절한 3~5 탭 FIR 필터가 사용된다[10]. 본 논문에서는 연산이 많이 이루어 질 수 있는 상황을 가정하여 JM에서 사용된 연산에 따라 +, -, >> 연산을 각각 1 cycle로 가정하여, 5 tap FIR filter가 사용될 때는 8 cycle, 4 tap

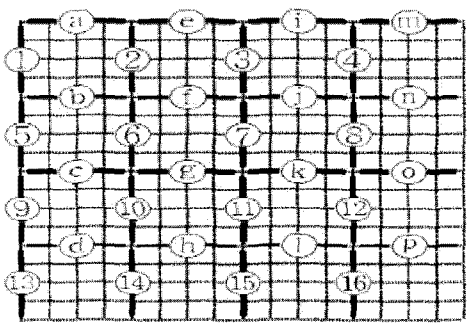
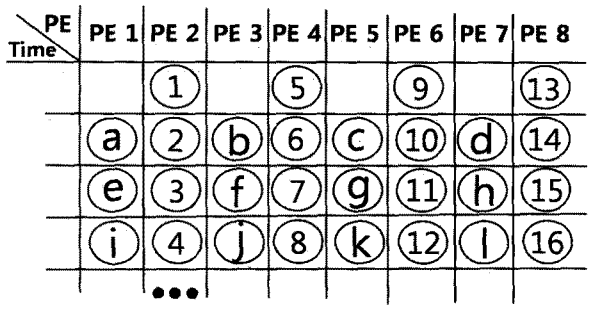
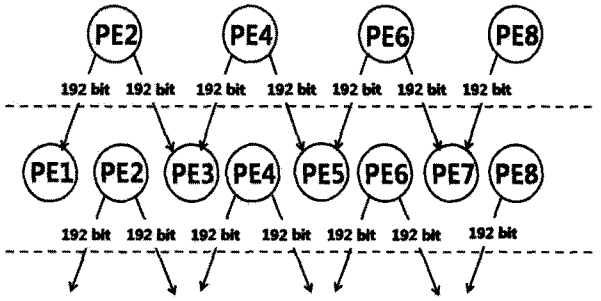


그림 5. 필터링 순서
Fig. 5. Filtering order.



(a) 각 processor에 할당된 연산
(a) Boundary to processor assignment



(b) 데이터 전송
(b) Data transfers

그림 6. Applicaion A의 PE 할당 및 데이터 전송
Fig. 6. PE assignments and data transfers in Applicaion A.

FIR filter가 사용될 때는 7 cycle로 정했다. 이와 같은 내용을 담은 transaction 약 7500개를 이용하여 실험하였다.

2. Application B : Random

그림 7은 8개의 PE 간 데이터 통신 양이 4 Kbyte로 일정한 random application이다. 각 PE에서 processing cycle은 1~9 cycle 사이의 임의의 값을 가진다. 또한 transfer cycle도 1~9 cycle 사이의 임의의 값을 가진다. Data dependency는 memory interface를 통한 input과 output을 고려하여 결정하였다. 이와 같은 내용의 transaction 7200개를 이용하여 실험하였다.

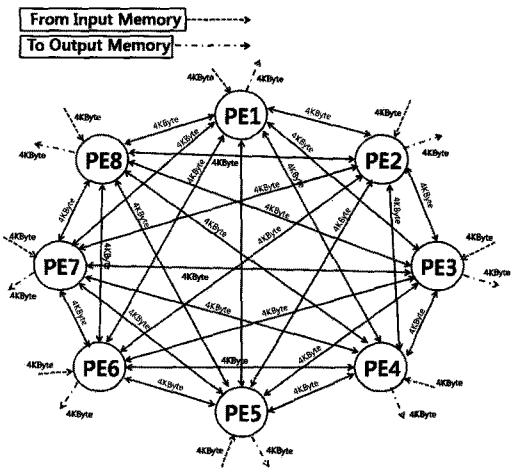


그림 7. Application B의 데이터 흐름
Fig. 7. Data flow of Application B.

3. Application C : Clustered Random

그림 8은 { PE 1, PE 2, PE 3, PE 4}와 { PE 5, PE 6, PE 7, PE 8}의 두 cluster로 나누어 각 cluster 내 PE 간 데이터 통신량이 10Kbyte이고 cluster 간에는 4KByte인 random application이다. 각 PE에서 processing cycle은 1~9 cycle 사이의 임의의 값을 가진다. 또한 transfer cycle도 1~27 cycle 사이의 임의의 값을 가진다. Data dependency는 memory interface를 통한 input과 output을 고려하여 결정하였다. 이와 같은 내용의 transaction 3200개를 이용하여 실험하였다.

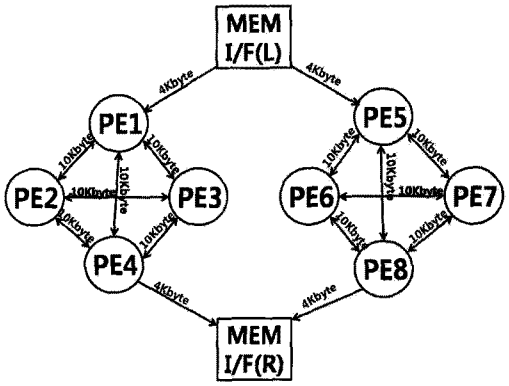


그림 8. Application C의 데이터 흐름
Fig. 8. Data flow of application C.

V. 실 험

제안된 설계 방법의 성능을 평가하기 위하여 버스 구조를 C언어로 모델링하여 실험하였다. 버스 구조에 따른 데이터 통신 특성을 고려하여, cycle 단위의 동작을 모델링하였다. 이를 이용하여 버스 구조에 따른 각 application의 completion cycle을 이용하여 성능을 평가하였다.

1. 각 Application에 대한 최적화

앞 장에서 설명한 3 가지 application에 대해 Removal와 Insertion 두 가지 방법으로 최적 버스 구조

표 1. Removal 방법을 사용하여 Connection 수 변화에 따른 각 Application의 Completion cycles
Table 1. Completion cycles of each application for different number of connections using Removal.

Connection 수	Application A	Application B	Application C
8	34848	15076	11834
7	34848	16292	11834
6	34848	15076	11834
5	34848	15887	12648
4	34848	18319	13462
3	34848	20345	13462
2	35244	21561	14201
1	36531	23587	14201
0	40491	24398	19379

표 2. Insertion 방법을 사용하여 Connection 수 변화에 따른 각 Application의 Completion cycles
Table 2. Completion cycles of each application for different number of connections using Insertion.

Connection 수	Application A	Application B	Application C
0	40491	24398	19379
1	36531	23587	14201
2	35244	21561	13831
3	34848	20750	13610
4	34848	18319	13240
5	34848	17103	11834
6	34848	15076	11834
7	34848	16292	11834
8	34848	15076	11834

표 3. 각 Removal와 Insertion 방법의 최적 버스 구조
Table 3. Optimized buses using Removal and Insertion.

	Removal		Insertion	
	Connection 수	Switch Block 수	Connection 수	Switch Block 수
Application A	3	3	3	6
Application B	6	4	6	5
Application C	6	4	6	4

를 설계하였다. 표 1은 각 application에 대해 Removal 방법을 적용하여, connection에 대해 completion cycle을 측정한 결과다.

표 2는 각 application에 대해 Insertion 방법을 적용하여, connection에 대해 completion cycle을 측정한 결과다.

각 Application A, B, C에 대한 성능 제약 조건 34900 cycle, 15100 cycle, 11900 cycle을 이용하여, 저비용 버스 구조를 구하였다.

표 3은 Removal와 Insertion 방법의 최적 버스 구조를 나타낸 표다. 그림 3의 구조는 connection 수 8개, switch block 수 14개를 이용한다. 이와 비교하였을 때, connection 수는 경우에 따라 25% 이상 감소하고, switch block 수 또한 경우에 57.1% 이상 감소한다. 또한 그림 9는 application A에 대해, Removal 방법으로 구한 저비용 버스 구조를 나타낸 그림이다.

실험 결과 Removal 방법이 Insertion 방법보다 우수한 결과를 보여주었는데 이 원인은 뒤에서 설명한다.

2. 다양한 Application에 대한 최적화

설계된 MPSoC 시스템은 한 순간에는 하나의 application을 수행하지만, 경우에 따라 여러 응용에 사용될 수 있다. 그림 9와 같은 하나의 application에 대한 최적 버스 구조로는 다양한 application에 대한 성능 제약 조건을 만족하지 못한다. 또한, 각 application에 대한 최적 버스 구조의 connection 및 switch block을 단순히 추가 사용하는 것으로는 다양한 application에 대한 성능 제약 조건을 만족하기 어렵다. 예를 들어, 표 1과 표 2의 application B에서 connection 7개를 사용한 경우보다 6개를 사용한 경우에 completion cycle이 감소하였다. 이는 application B의 transaction들이 수행되는 순서 변환에 따른 문제다. Connection 또는 switch block 추가에 따라 동시에 수행되는 transaction의 조합이 달라지고, transaction 수행 순서에 변화가 생기면서, completion cycle이 증가하는 경우도 있다.

따라서 다양한 application에 대해 성능 제약 조건을 만족하는 저비용 버스 구조를 찾기 위해서는 한 application에 대해 적용한 방법 외에 별도의 방법이 필요하다. 따라서, 앞에서 설명한 Removal와 Insertion 방법을 다양한 application에 대해 저비용 버스 구조를 구할 수 있도록 변형하였다.

Removal 방법에서는 connection을 하나 씩 모든 위치에서 제거하며 모든 application의 성능을 측정한다.

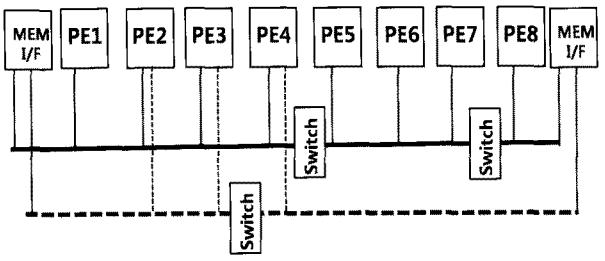


그림 9. Application A를 이용한 저비용 버스 구조
Fig. 9. Low cost bus architectures for Application A.

표 4. 성능 제약 조건 내에서 Removal 방법을 이용하여 Connection 수 변화에 따른 다양한 application의 completion cycles

Table 4. Completion cycles of each application for different number of connections under performance constraints using Removal.

Connection 수	Application A	Application B	Application C
8	34848	15076	11834
7	34848	16292	11834
6	34848	17103	11834
5	34848	16698	12796
4	34848	21156	13240
3	35838	21561	16864
2	36432	23182	19379
1	36630	23587	19379
0	40491	24398	19379

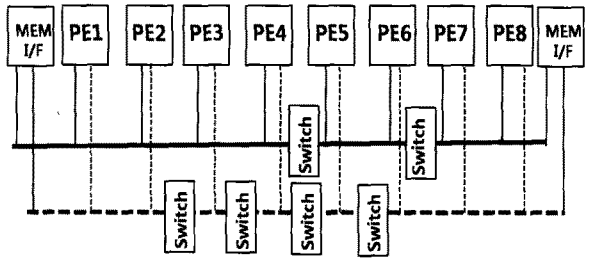


그림 10. 성능 제약 조건 내에서 저비용 버스 구조

Fig. 10. Low cost bus architecture under performance constraints.

측정된 성능이 성능 제약 조건과 차이가 가장 적은 (가장 critical한) application에서 가장 성능이 적게 떨어지는 위치의 connection을 제거한다. 이 과정을 성능 제약 조건을 만족하는 범위 내에서 반복하여, connection을 최소화 한다. 이 때, 가장 critical한 application에서 성능이 가장 적게 감소하는 connection이 하나 이상 발생할 경우에는 그 다음으로 critical한 application에서 가장 성능이 적게 떨어지는 위치의 connection을 제거한다. 이 방법을 모든 application에 대해 적용한 후에도 성능이 가장 적게 감소하는 connection이 하나 이상 존재할 경우에는 앞서 사용한 lookahead 방법을 이용하여 사양을 만족하는 범위 내에서 제거한다. 이와 같은 방법으로 얻어진 최소화된 connection 구조에서 switch block을 같은 방법을 적용하여 성능 사양을 만족하는 범위 내에서 제거한다.

Insertion 방법은 가장 성능을 만족시키지 못하는 application의 성능을 최대로 증가시키는 connection/switch를 하나 씩 추가하는 방법으로, 앞에서 설명한 방법과 비슷하므로, 자세한 설명은 생략한다.

실험을 위해, 최적화 전 구조가 갖는 성능과 비슷한 34900 cycle, 15100 cycle, 11900 cycle을 성능 제약 조건으로 하여 실험하였다. 표 4는 다양한 application에 대해 성능 제약 조건을 이용하여 Removal 방법을 적용하여, connection에 대해 completion cycle을 측정한 실험한 결과이다.

그림 10은 성능 제약 조건 내에서 Removal 방법을 이용하여 구한 저비용 버스 구조를 나타낸 그림이다. 기본 버스 구조에서 성능 사양을 만족하는 저비용 버스 구조를 구한 결과, 최적화 전에 비해 switch block이 57.1% 감소하였다.

Insertion 방법은 다른 switch block들이 없는 상태에서 삽입한다. 이로 인해 switch block 간의 위치 관계를 고려한 삽입이 되지 않는다. 그러나 Removal 방법은 가능한 모든 위치에 switch block이 삽입된 상태에서 삭제한다. 따라서, 다른 switch block과의 위치 관계를 고려하게 된다. 이러한 이유로, Removal 방법이 Insertion에 비해 사용되는 connection이나 switch block이 적게 되어 우수한 최적화 결과를 보여준다.

참 고 문 헌

[1] E. Salminen, V. Lahtinen, K. Kuusilinna, and T. Hamalainen, "Overview of Bus-Based System-on-Chip Interconnection," in Proc. IEEE ISCAS, pp. 372-375, 2002.

[2] IBM, Armonk, NY, "CoreConnect Bus Architecture," 1999.

[3] R. Lu and C. K, "A High Performance Bus Communication Architecture through Bus Splitting," in proc. ASPDAC, pp.751-755, 2004.

[4] M. Jun, S. Yoo, and E. Chung, "Mixed Integer Linear Programming-based Optimal Topology Synthesis of Cascaded Crossbar Switches," Proc. ASPDAC, pp.583-588, 2008.

[5] S. Pasricha, N. Dutt, "COSMECA : Co-synthesis of Memory and Communication Architectures for MPSoC," proc. DATE, pp 700-705, 2006.

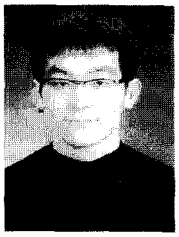
[6] S. Pasricha and N. Dutt. "On-Chip Communication Architectures -System on Chip Interconnect," Morgan Kaufmann. 2008.

[7] R. Lu, A. Cao and C. Koh, "SAMBA-Bus : A High Performance Bus Architecture for System-on-Chips," IEEE Transaction on VLSI Systems, Vol.15, No.1, pp. 69-79, Jan. 2007.

[8] H. Lee, Y. Lee, "An Efficient Data-Reuse

Deblocking Filter Algorithm for H.264/AVC," IEEK journal Vol.44, No.6, pp30-35, Nov. 2007.
[9] AMBA, Limitred, Cambridge, U.K., "AMBA Specification," 1999.
[10] ITU-T Rec. H.264 / ISO/IEC 11496-10, "Advanced Video Coding," Final Committee Draft, Document JVT-G050, Mar. 2003.

저 자 소 개

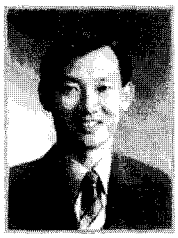


김 홍 염(학생회원)
2008년 한양대학교 전자컴퓨터 공학부 졸업
2008년~현재 한양대학교 전기전자제어계측공학과 석사과정

<주관심분야 : CAD&VLSI, 반도체 설계, SoC 설계방법론>

정 성 철(학생회원)
2008년 백석대학교 정보통신학부 학사 졸업
2008년~현재 한양대학교 전기전자제어계측공학과 석사과정

<주관심분야 : CAD&VLSI, 반도체 설계, SoC 설계방법론>



신 현 철(평생회원)
1978년 서울대학교 전자공학과 학사 졸업.
1980년 한국과학기술원 전기 및 전자공학과 석사 졸업
1983년~1987년 U.C. Berkeley 박사 졸업.
1983년~1987년 Fulbright scholarship
1987년~1989년 MTS, AT&T Bell Lab's, Murray Hill N.J., USA
1997년~2008 IDEC 한양대학교 지역센터 센터장
1989년~현재 한양대학교 전자컴퓨터공학부 교수
2008년~현재 ITRC Multi-core Design Methodology 연구센터 소장.

<주관심분야 : CAD&VLSI, 통신용 반도체 설계, 저전력 설계>