논문 2007-44SD-8-12

Network-on-Chip에서의 최적 통신구조 설계

(Optimal Design of Network-on-Chip Communication Sturcture)

윤 주 형*, 황 영 시*, 정 기 석*

(Joo-Hyeong Yoon, Young-Si Hwang, and Ki-Seok Chung)

요 약

매우 복잡한 시스템의 보다 효율적인 설계를 위한 차세대 SoC를 위해 중요한 것은 시스템의 고적용성과 고확장성이다. 이를 위해 최근 들어 급속히 관심이 높아지는 것이 계산 모듈 중심의 시스템 설계를 탈피하여 통신 중심으로 시스템 설계를 보는 communication-based 설계 방법론이며, 그 중 대표적으로 많은 관심을 모으고 있는 것이 Network-on-Chip (NoC)이다. 이는 모듈간의 직접적인 연결에 의한 데이터의 통신 구조를 가진 일반적인 SoC 설계에서의 취약한 확장성과 통신 구조의 고정성을 극복하기 위해, 데이터를 패킷화하고, 이를 네트워크 인터페이스 및 라우터에 의한 가변적인 구조에 의해 전송함으로써 통신 구조의 적용성과 확장성을 제공하려는 노력이다. 하지만 확장성과 적용성에 치중하다 보면 성능과 면적에 대한 비용이 너무 커져서 실제로 기존의 연결 방법과 비교하여 실용성이 없을 수 있다. 그래서 본 연구에서는 통신 패턴의 면밀한 분석을 통하여 매우 성능에 중요하고 또 빈번한 통신 패턴에 대해서는 기존의 연결 방식을 고수하면서, 전체적인 연결성 및 확장성을 유지하는 알고리즘을 제시한다. 이 방법을 통해서 최소 30%의 네트워크 인터페이스 및 라우터 구조가 훨씬 간단한 구조로 바뀔 수 있었으며, 이로 인한 연결성 (connectivity) 및 확장성에 대한 손실은 거의 없었다. 시뮬레이션 결과에 의하면 통신 구조의 최적화를 통해서 연결에 소요되는 시간적 성능은 49.19% 향상되었고 면적의 측면에서도 24.03% 향상되었음이 입증되었다.

Abstract

High adaptability and scalability are two critical issues in implementing a very complex system in a single chip. To obtain high adaptability and scalability, novel system design methodology known as communication-based system design has gained large attention from SoC designers. NoC (Network-on-Chip) is such an on-chip communication-based design approach for the next generation SoC design. To provide high adaptability and scalability, NoCs employ network interfaces and routers as their main communication structures and transmit and receive packetized data over such structures. However, data packetization, and routing overhead in terms of run time and area may cost too much compared with conventional SoC communication structure. Therefore, in this research, we propose a novel methodology which automatically generates a hybrid communication structure. In this work, we map traditional pin-to-pin wiring structure for frequent and timing critical communication, and map flexible and scalable structure for infrequent, or highly variable communication patterns. Even though, we simplify the communication structure significantly through our algorithm, the connectivity or the scalability of the communication modules are almost maintained as the original NoC design. Using this method, we could improve the timing performance by 49.19%, and the area taken by the communication structure has been reduced by 24.03%.

Keywords: System-on-Chip, Network-on-Chip, Design Methodology

I. 서 론

매우 복잡한 시스템의 보다 효율적인 설계를 위한 차세대 SoC를 위해 중요한 것은 시스템의 고적용성과 고확장성이다. 이를 위해 최근 들어 급속히 관심이 높아지는 것이 계산 모듈 중심의 시스템 설계를 탈피하여 통신 중심으로 시스템 설계를 보는 통신 기반 (communication-based) 설계 방법론이며, 그 중 대표적으로 많은 관심을 모으고 있는 것이 Network-on-Chip (NoC)이다[1, 4~5, 7, 9]. 이는 모듈간의 직접적인 연결에 의한 데이터의 통신 구조를 가진 일반적인 SoC

* 정회원, 한양대학교
(Hanyang University)
※ 이 논문은 2004년도 한국학술진흥재단의 지원에 의하여 연구되었음 (KRF-2004-003-D00286)
접수일자: 2007년5월7일, 수정완료일: 2007년7월31일

설계에서의 취약한 확장성과 통신구조의 고정성을 극복하기 위해, 데이터를 패킷화(packetization)하고, 이를 고정적인 연결구조가 아닌 프로토콜 스택을 처리하는 네트워크 인터페이스 및 라우터에 의한 가변적인 구조에 의해 전송함으로써 복잡한 시스템 설계에 필수적으로 중요한 통신구조에 적용성과 확장성을 제공하려는 노력이다[2~3, 6].

하지만 이 Network-on-Chip 구조에 관한 연구는 이제 시작 단계에 있으며, 아직까지는 몇 가지 문제 때문에 널리 사용되지 못하고 있다. NoC 구조는 기존의 SoC 구조에 비해 설계하기가 복잡하며, 라우터에 의한 통신구조와 데이터의 패킷화에 의한 오버헤드가 상당히 크다. 게다가 온칩 버스[7]와 핀과 핀의 직접 연결과 같은 기존의 통신구조들은 NoC에 비해 설계가 쉽고 성능면에서 더 빠르기 때문에 이미 널리 사용되고 있다.

따라서 본 연구에서는 기존의 NoC 구조의 단점을 극복할 수 있는 새로운 NoC 설계 방법론을 제시하고자 한다. 우선 그 중의 가장 핵심이라고 할 수 있는 통신구조의 복잡성에 대한 예측 및 최적구조의 설계에 대해 연구한다.

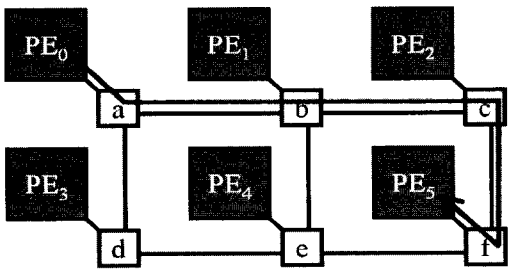

그림 1. 라우터를 통한 PE 들간의 연결 예
Fig. 1. An Example of Communication Patterns.

Ⅱ. 최적화문제의 정의

1. NoC 통신 복잡도 모델링 및 기본 가정

본 연구의 핵심은 NoC 구조상에서 모든 모듈들 (PE, processing elements)의 위치 선정(placement)이 주어졌고 또 이들이 구현하는 특정 응용 프로그램이 한정되어 있어서 이런 전형적인 응용 프로그램을 실행할 때, 가장 최적인 연결 구조를 설계하는 일이다. 특정 응용 프로그램이 위치가 고정된 모듈사이에서 실행될 때, 통신 패턴 역시 응용 프로그램의 특성과 각 모듈의 위치에 따라 거의 결정된다고 볼 수 있다. 다만, 다양한 연결구조가 가능한 NoC에서 어떤 모듈에서 다른 모듈로 데이터 전송이 이루어질 때, 반드시 하나의 연결 방법만이 존재한다고 말하기 어렵다. 다양한 방법의 라우팅을 고려할 수 있지만, 본 연구에서는 문제를 간단히 하기 위해 주어진 모듈에서 다른 모듈로 통신이 이루어질 때, 연결구조상 상하 방향의 연결 방법과 좌우방향의 연결 방법이 모두 가능할 때는 좌우 방향의 연결에 더 높은 우선순위를 둔다고 가정하고 통신패턴 분석을 한다. 이러한 가정에 따라 고정된 위치의 두 모듈간의 통신은 일정한 경로를 거쳐 가게 되며 이 때, 각 모듈간의 통신 패턴은 이 경로 상에 거쳐 가는 라우터들을 나열함으로써 명시할 수 있다.

2. 문제 정의

통신패턴의 모델링을 통해서 알 수 있는 통신의 복잡도는 각 응용 프로그램 마다 다르다. 본 연구에서는 이런 다양한 종류의 통신패턴을 자동으로 분석하는 알고리즘을 제시한다. 이는 이 보고서의 저자들이 판단하기로는 기존의 연구와 매우 차별되는 설계 방법이다. 기존의 구조는 대부분 정형화된 NoC 구조에 특정 응용프로그램을 매핑하여 처리하는 방향이었다. 하지만, 이런 접근 방법은 현실적으로 몇 가지 문제점을 갖는다. 첫째, NoC는 다수의 프로세서 코어와 다수의 IP들, 그리고 분산된 메모리 블록이 있다는 것을 가정하고, 이들 간의 복잡한 통신구조를 가정한다. 그러나, 현실적으로 멀티 코어 SoC가 점차 일반화 되고는 있지만, 그들 사이의 통신이 비교적 고정적이고 단순하여 반드시 매우 일반적인 형태의 라우터를 요구하지는 않는 것이 대부분이다. 이에 모든 프로세싱 요소 (processing element)들이 가장 일반화된 스위치 구조인 라우터 형태로 연결된다고 가정한다는 것은 무리가 있다. 둘째로 NoC의 확장성과 재구성성이 중요하지만, 이 점만이 지나치게 강조될 경우 특정 응용에 적합한 구조를 최적화시키는 것이 목적인 임베디드 시스템 설계라는 본질적인 문제를 간과하기 쉽다. 이에 시스템이 구현하는 응용프로그램이 수시로 변하는 것은 아니기 때문에 재구성성과 확장성을 유지하면서도 가장 전형적인 응용 프로그램의 실행에 대해 최적화되어 있는 구조를 유지해야만 한다.

위의 문제점들은 NoC의 구조가 적용목적에 부합하여 최적화되어야 하고, 또 이를 위해서는 정형적인 동일 구조로 이루어지기보다, 다양한 형태의 온칩 통신구조를 복합적으로 갖는 혼합적 통신구조로 설계되어야 함을 의미한다. 이에 본 논문에서는 다음과 같이 문제를 정의한다.

3. 최적화 문제

다양한 프로세서 요소와 IP, 메모리 블록 등으로 구성된 NOC 구조에서, 각 블록들이 특정 위치에 맵핑되어 있을 때, 특정 응용에 대한 전형적인 통신패턴이 주어졌다고 가정한다. 이 때, 이 통신패턴을 최적으로 구현하면서도 확장성과 재구성성을 유지하는 최적의 통신구조를 설계한다. 결과의 최적성은 통신 연결구조에서의 속도면 성능과 연결구조가 차지하는 면적, 그리고 유연성(또는 확장성) 이 세 가지을 기준으로 판단된다.

Ⅲ. 통신패턴 기반 최적구조 설계 알고리즘

1. 부분 통신 빈도수를 고려한 통신패턴 분석

본 연구에서 제시하는 알고리즘의 핵심은 빈번한 통신패턴을 위한 연결구조는 되도록 간단하고 빠른 구조로 정하고자하는 것이다. 이는 시스템의 평균 성능은 "일반적인 경우(common cases)를 더 빠르게 함(faster)으로써" 크게 향상될 수 있다는 생각에 근거한다. NoC의 연결구조에서 라우터와 NI는 유연성과 확장성에서 뛰어하지만, 속도나 면적면에서의 오버헤드는 상당하다고 할 수 있다. 특히 온칩에서 pin-to-pin 연결과 비교할 때, 상대적 속도차이가 최대 100배까지 발생할 수 있다[9]. 이는 실제로 NoC구조가 널리 실용화되지 못하는 큰 이유이기도 하다. 그래서 본 연구에서는 특정 통신 빈도수가 매우 많은 연결에 대해서 우선적으로 가장 단순한 연결구조로 연결될 것을 유도하며, 대신 빈도수가 많지 않거나 다양한 통신에 관련된 연결구조는 라우터구조를 유지하게 하여 확장성과 성능의 최적화를 동시에 고려한다.

앞서 언급한 바와 같이 본 연구에서는 특정 응용 목적을 갖는 NoC의 연결 구조를 최적화하는 것을 목표로 하기 때문에 통신패턴에 대한 프로파일링(profiling)이 존재한다고 가정한다. 통신 프로파일링이라고 하는 것은 각 시간에 따라 발생하는 통신의 패턴을 연결 경로 형태로 나타낸 것이라 하겠다. 다만, 우리가 관심을 갖는 것은 통신 끝단간의 송신부와 수신부 만은 아니며, 통신에 자주 가장 자주 나타나는 부분 경로의 처음과 끝에 대한 정보라고 하겠다.

간단히 될 수 있는 부분 경로를 탐색하는데 있어, 우리는 두 가지의 기준에 따른다. 첫째는 자주 나오는 경로를 찾는 것이며, 두 번째는 되도록이면 부분 경로가 긴 것을 찾는 것이다. 부분 경로가 길어진다는 것은 그만큼 긴 부분 경로상의 라우터들이 더 간단한 연결구조로 바뀐다는 것이고 그만큼 전체적으로 NOC 연결구조의 총 오버헤드가 줄어들어 최적화 될 수 있기 때문이다. 이에 따라 간단히 되어야 할 부분 경로에 대한 선호도는 다음과 같은 값을 계산하여 정한다.

$$MERIT_{a,b} = \alpha \times frequency + \beta \times length \qquad (1)$$

여기서 $MERIT_{a,b}$는 a에서 b까지의 부분 경로의 중요성을 나타내며, 이는 얼마나 자주 나타나는가와 또 얼마나 많은 라우터를 경유하는 경로인가의 가중치 합(weighted sum)으로 구해진다. α와 β는 사용자 정의 가중치이며 빈도수와 길이의 상대적 중요도를 결정하는데 사용된다. 본 논문의 실험에서는 α=10, β=15로 하여 빈도수와 길이의 균형 있는 가중치로 실험하였다. 이 설정에 관련한 성능 평가는 실험 결과 부분에서 다루기로 한다.

표 1. 허프만 코딩의 예
Table 1. An Example of Huffman Coding.

	a	b	c	d	e	f
빈도수	45	13	12	16	9	5
고정길이 코딩	000	001	010	011	100	101
가변길이 코딩	0	101	100	111	1101	1100

2. 허프만 트리를 이용한 통신구조 매핑

가. 수정된 허프만 알고리즘을 적용한 트리 생성

전통적 데이터 통신에 있어 데이터는 동일한 길이의 코드로 인코딩되어 전송된다. 이를 흔히 고정 길이 인코딩 (fixed-length encoding)이라고 한다. 예를 들어, 영문자의 전송이나 저장이 7 비트의 동일 크기 ASCII 코드로 인코딩되는 것이 그 예라할 수 있다. 하지만, 실제 전송에 있어서 모든 알파벳이 동일한 빈도수로 나타나는 것이 아니기 때문에 이런 동일 크기로 인코딩하는 것은 통신 효율적인 측면에서 바람직하지 못하다. 보다 통신의 효율을 높이기 위해는 통신 빈도수에 따라 자주 나오는 데이터를 되도록 짧은 코드로 인코딩(encoding)하고 비교적 뜸하게 발생하는 데이터를 상대적으로 긴 코드로 인코딩하면 자주 나오는 데이터에 대해 상대적으로 짧은 코드로 전송되기 때문에 전체적으로 적은 비트수로 같은 양의 데이터를 전송할 수 있다. 이렇듯 차등길이를 갖는 코드 (variable-length coding)의 대표적인 예가 허프만 인코딩(Huffmann Encoding)[8]이다.

본 연구에서는 이 허프만 알고리즘을 통신패턴의 빈도수와 연계하여 적용시킨다. 즉, 통신패턴의 프로파일

링 데이터에서 각 부분 경로에 대해서 위에 주어진 <식 1>의 $MERIT_{a,b}$를 구한다. 임의의 통신 방식에 대해서 이 경우의 수는 온 칩 상의 PE나 라우터수에 대해 지수적으로 증가한다. 하지만, 모든 경우를 다 고려할 필요는 없으며, 본 논문의 핵심 아이디어가 기본적으로는 라우터를 이용한 통신을 하고, 빈도수가 많은 부분적 통신에 대해 더 간단한 포트간의 직접 연결이나 멀티플렉서/디멀티플렉서를 이용한 연결구조로 바꾸는 것이므로, 최대 빈도수이면서 길이가 되도록이면 긴 부분 경로를 Merit 값에 따라 N개 선택하여 이들로만 이루어진 허프만 트리를 구성한다. N의 값은 사용자 정의 값이며, 대략 전체 연결구조 (초기의 라우터 개수)의 10-20% 정도로 한다. 이렇게 선택된 N개의 부분 경로는 허프만 트리의 리프 노드가 된다. 허프만 트리 생성 알고리즘처럼 가장 작은 Merit 값을 가지는 리프 노드 두 개를 찾아 부모 노드로써 새로운 노드를 만든다. 생성되는 부모 노드의 Merit 값은 다음 <식 2>와 같이 계산한다.

$$MERIT_{new} = \alpha \bullet (frequency_{left\,child} + frequency_{rightchild}) + \beta \bullet 0.5(length_{leftchicd} + length_{rightchild}) \quad (2)$$

이와 같은 노드의 생성은 루트 노드가 생성될 때까지 계속하며, 최종적으로 N개의 리프 노드를 가지는 허프만 트리가 완성된다.

나. 허프만 트리를 이용한 연결구조 매핑

허프만 트리가 완성되면 이로부터 연결구조 매핑을 구한다. 허프만 트리가 만들어 졌다고 가정했을 때, 우리가 가정하는 연결구조의 종류가 M개 있다고 하자. 이 때 가장 일반적인 초기 연결 형태를 라우터/NI 형태라 볼 수 있으며, 이는 기본(default) 구조이므로 상대적으로 자주 발생한다는 조건으로 선택된 허프만 트리에 있는 리프 노드들의 연결구조 형태로는 라우터/NI는 적합하지 않다고 하겠다. 그러므로 본 연구에서는 M개의 구조 중 하나를 뺀, M-1개의 연결구조로 트리 상에 있는 부분 연결 경로들을 매핑한다.

직관적으로 연결구조의 매핑은 실제로 고려해야할 조건들이 매우 많기 때문에 아무리 연결구조의 종류가 몇 개로 한정되어 있다고 해도, 최적으로 맵핑한다는 것은 NP-Complete한 문제라고 할 수 있다. 그래서 본 연구에서는 간단한 휴리스틱을 통해서 연결구조를 매핑한다. 이 연구에 사용하는 휴리스틱은 다음과 같다.

H를 허프만 트리의 높이라 하고, D(i)를 허프만 트리의 루트노드에서 노드 I에 이르는 거리라고 하자. 즉,

$$H = MAX_{\forall i} D(i) \quad (3)$$

이 때 가능한 한 연결구조 M개 중 오버헤드가 가장 큰 라우터 구조를 제외한 나머지 연결구조를 가장 간단한 것부터 가장 복잡한 것 까지 C0, C1, ... CM-2 로 부를 때, 어떤 노트 i 의 연결구조는 <식 4>를 만족하는 경우 이를 C0에서 이를 만족하는 최소값 d 에 에 해당하는 Cd 까지의 연결구조가 그 설계의 대상이 된다고 할 수 있다.

$$D(j) \le (d+1) \times \lceil H/(M-1) \rceil \quad (4)$$

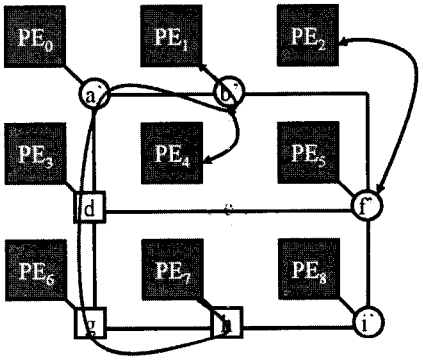

그림 2. 연결 경로의 재설정
Fig. 2. Rerouting of Communication Pat.

3. 연결성을 위한 통신구조의 재구성

위의 방법을 통해 부분 연결들을 간소화하면, 초기에 존재하던 연결구조들이 많이 간략화 됨을 알 수 있다. 그래서 기존의 연결성의 유연성 및 확장성이 약화되고, 또 빈번하진 않지만, 꼭 존재하여야하는 노드와 노드간의 연결성이 사라지는 것을 알 수 있다. 이를 위해서 이 단계에서는 통신패턴 프로파일링 정보 상에 나타나는 모든 연결에 대해 연결성이 존재하는지를 확인하고, 필요에 따라서는 연결성 유지를 위해 추가의 연결구조를 추가하는 단계를 거친다. 통신패턴의 연결성 유지는 크게 연결구조의 증가 방법과 연결 통로의 재설정으로 나누어 구현된다.

가. 연결 통로의 재설정

연결 경로의 재설정은 되도록이면 현재 존재하는 연결구조를 이용하여 통신패턴을 구현하도록 하는 것이다. 다양한 방법이 존재하나, 본 연구에서는 탐욕적 알

고리즘을 (greedy algorithm) 사용한다. 본 연구에서는 경로를 찾는 방법에 있어 되도록 방향이 데이터 전송에 가까워 질 수 있도록 정한다. 이렇게 네 방향 중, 되도록이면 데이터 전송의 수신 쪽 방향부터 연결 경로를 검색하는 탐욕적(greedy) 방법을 이용한다. 본 연구에서의 시뮬레이션 결과에 따르면 이와 같은 탐욕적(greedy) 방법이 쉽게 연결 경로를 재설정하는 것에 대해서 거의 대부분 문제가 없음을 알 수 있었다.

나. 연결구조의 최소 추가

어떤 데이터 전송에 대해 위와 같은 탐욕적으로(greedy) 연결구조를 재설정하는 것으로는 연결성이 회복되지 않을 경우에는 연결구조를 새로이 추가하여 연결성을 유지하도록 한다. 연결구조를 추가하는 것은 크게 세 가지의 방법에 의존한다. 첫째는 멀티플렉서/디멀티플렉서를 추가하는 방법이다. a->

f로의 실제적인 적용을 할 때, a->b 로의 연결을 만들고 노드 b에서 멀티플렉서를 더하면 된다. 단 b->f로의 통신과 a->b->f 의 통신이 시간상 동시에 이루어지지 않도록 하는 스케쥴링은 필요하다고 보고, 필요에 따라서는 버퍼가 필요하다고도 볼 수 있다. 이와 비슷하게 하나의 긴 연결 경로에 여러 노드로 갈라지는 경로가 생겨야하면, 버스 구조를 이용한 매핑을 통해 구현할 수 있다.

IV. 전체 알고리즘 요약

이 장에서는 연결구조의 최적화를 위한 알고리즘을 설명하였다. 이를 간략히 요약하면 다음과 같다.

ALGORITHM OON (Placement Information, Typical Application, Comm. Structure){
1. Profiling the communication pattern
2. Analyze profiling information to identify long & common partial communication paths
3. Build Huffmann Tree using N common paths
4. Map each communication paths to desirable communication structures
5. Re-route and restructure the communication structure to maintain connectivity.
}

본 연구에서 제시된 알고리즘 OON (Optimal On-Chip Networking)을 요약하면, 먼저, NoC 상의 모든 모듈들의 위치 정보가 확정되고 또 이들이 전형적으로 구현하는 응용 프로그램의 종류가 입력 정보로 주어진다고 가정하자. 초기 연결구조는 가장 유연성과 확장성이 좋은 라우터 및 네트워크 인터페이스로 연결되어 있다고 하자. 본 연구는 이로부터 실제로는 통신패턴이 매우 비균등하게 이루어짐을 이용하여, 매우 빈번히 일어나는 통신패턴에 대해서는 더 오버헤드가 작은 통신구조로 바꿈으로써 전체적인 통신구조의 비용 및 통신 오버헤드를 줄이는 알고리즘을 제시한다. 이를 위해 첫 단계에서는 특정 응용 프로그램을 수행할 때의 나타나는 통신패턴에 대해 기록하며, 이 통신패턴의 분석을 통해서 가장 빈번히 일어나면서도 많은 라우터를 거치는 부분적 연결 경로를 찾는다. 다음으로는 이들 부분 경로를 하나의 리프 노드로 갖는 허프만 트리를 만든다. 이 허프만 트리에서 루트 노드와 각 부분 경로에 해당하는 리프 노드 사이의 거리를 바탕으로 가장 단거리에 있는 노드들은 되도록이면 가장 간단한 통신구조로 맵핑하는 방법을 사용하여 통신구조를 최적화시킨다.

V. 실험 설정 및 결과

1. 실험 설정

본 연구에서는 제시했던 가정을 토대로 실험을 설정하였다. NoC 구조상에서 모듈 (PE, processing element)의 개수는 25개로 정해졌으며 모든 모듈들의 위치 선정이 주어졌다고 가정하였다. 이 모듈들이 구현하는 특정 응용 프로그램이 한정되어 있어서 이런 전형적인 응용 프로그램을 실행할 때의 상당히 고정된 통신패턴이 정해진 모듈 사이에서 실행된다고 보고 통신패턴을 몇 개의 특정 모듈 사이의 통신을 다른 모듈에 비해 더 많은 가중치를 두어 빈번하게 발생하는 것으로 설정하였다. 그리고 고정된 두 모듈이 선택된 후 모듈 사이의 통신패턴, 즉 라우팅 방식을 고려할 때, 문제를 간단히 하기 위해 연결구조상 상하 방향의 연결 방법과 좌우방향의 연결 방법이 모두 가능할 때는 좌우 방향의 연결에 더 높은 우선순위를 둔다고 가정하였다. 이러한 가정 하에 이 실험에서는 총 1000개의 통신패턴을 임의로 생성하였다.

이렇게 생성된 1000개의 통신패턴에서 공통된 문자열 찾는 알고리즘을 통하여 가능한 모든 문자열과 빈도수와 테이블에 저장하였다. 저장된 테이블을 이용하여 빈도수와 문자열 길이에 각각 가중치를 변화시켜가며

$MERIT_{a,b}$을 구하였다. 기본 가중치는 앞에서 언급한 것과 같이 통신패턴 샘플 수 100개를 기준으로 빈도수에 10, 문자열 길이에 15를 주었다.

허프만 알고리즘에서의 결과 값인 코드길이는 트리 구조에서의 깊이로서 이용하였으며 이와 트리의 높이에 따라 네트워크 구조를 변화시켜 주었다. 허프만 알고리즘에서 입력 값을 10개 정도 주었을 시에는 보통의 경우 높이가 4에서 8까지로 고른 분포가 많이 나왔다. 그래서 이러한 높이를 이용하여 각각의 깊이에 따라 통신패턴의 우선순위가 높은 부분은 공통된 문자열의 끝을 제외하고는 모두 직접 연결로 단순화 시켰으며, 끝부분은 라우터 대신 멀티플렉서 구조로 대체하였다. 그리고 보다 우선순위가 낮은 부분에서는 라인이 아닌 멀티플렉서와 버스로 구조를 대체하였으며 대체된 경로를 지나게 되는 통신패턴에 대해서는 앞서 대체시킨 멀티플렉서로 갈 수 있는 경우를 제외하고는 멀티플렉서와 같은 회로를 추가하거나 새로운 경로를 찾아가는 등의 오버헤드를 고려하였다.

마지막으로는, 이렇게 변화된 네트워크 구조와 앞에서 샘플로 추출한 1000개의 통신패턴 및 라우터, 멀티플렉서, 라인의 시간비용과 면적비용을 이용하여 변화된 지연시간(delay cost) 및 면적(area)을 구하였다. 기본설정에서는 Rent's Rule[10]에 의거하여 5x5 라우터의 지연 시간은 10, 멀티플렉서의 지연 시간은 2, 라인을 통한 지연 시간은 1로 설정하였으며, 5x5 라우터의 면적은 50, 멀티플렉서의 면적은 16, 라인의 면적은 1로 설정하였다.

2. 실험 결과

첫 번째 실험은 가장 좋은 가중치의 비율을 찾기 위해 출력한 결과이다. 여기서 가중치란 <식 1>에서 Merit 값을 구할 때 사용한 α, β 값이다. <그림 3>은 가중치의 비율에 따라 지연비용(delay cost)이 얼마나 감소하는가를 보여준다. 각각의 가중치마다 100번 이상의 프로그램 수행을 통해 평균값을 그래프에서 표현하였다. 그래프의 가장 왼쪽 막대는 전체 구조가 라우터와 네트워크 인터페이스로 이루어진 전형적인 NoC 구조의 지연비용을 나타낸다. 먼저 빈도수의 가중치인 α 값을 10으로 고정시켰을 때, 부분경로 길이의 가중치인 β을 10에서 15, 20, 30, 40으로 변화시켜가며 측정한 것이 그래프의 왼쪽 부분이다. 이를 보면 β 값이 15에서 40으로 변화하면서 전체적인 지연비용이 크게 증가함을 볼 수 있다. 그래프의 오른쪽 부분은 부분경로길이의

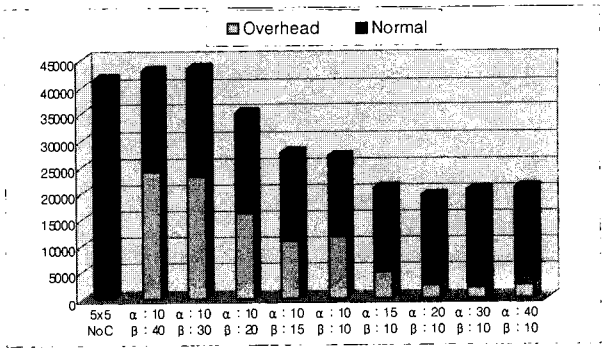

그림 3. 다양한 가중치에 따른 성능
Fig. 3. Performance about Variable Weights.

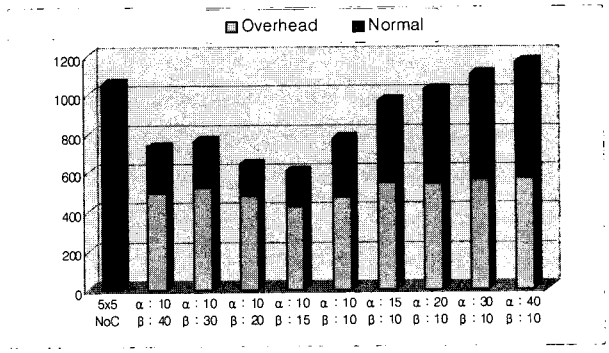

그림 4. 다양한 가중치에 따른 면적
Fig. 4. Area about Variable Weights.

가중치인 β 값을 고정시키고 빈도수의 가중치인 α 값을 10에서 15, 20, 30, 40으로 변화시키며 측정한 결과이다. α 값이 증가함에 따라 지연비용의 오버헤드는 줄어들고 전체적인 지연비용 역시 줄어듦을 볼 수 있다.

<그림 3>과 유사하게, <그림 4>는 가중치의 비율에 따라 면적비용(area cost)이 얼마나 감소하는가를 보여준다. 이 그래프 역시 빈도수의 가중치인 α 값을 10으로 고정시켰을 때, 부분경로 길이의 가중치인 β을 10에서 15, 20, 30, 40으로 변화시켜가며 측정한 것이 그래프의 왼쪽 부분이다. 이를 보면 β 값이 증가함에 따라 전체적인 면적비용이 그다지 향상되지 않음을 볼 수 있다. 그래프의 오른쪽 부분을 보면, α 값이 증가함에 따라 전체적인 면적비용이 크게 증가함을 볼 수 있다. 결국 성능과 면적이라는 관점 모두를 고려할 때, 최적화된 10:15의 비율을 기본 설정으로 사용하게 되었다.

이후의 실험은 앞에서 설정한 기본 설정을 그대로 하였다. <그림 5>은 프로그램을 1회 수행하였을 때 출력되는 결과의 예를 보여준다. 허프만에 쓰일 부분 경로는 10개로 설정되어 Merit 값의 순서대로 (i,n,s), (s,x), (f,g,h,i), (d,i), (b,c,d), (s,r), (r,q), (i,h,g,f), (s,n,i), (p,k,f)가 리프노드로써 포함됨을 알 수 있다. "FREQUENCY"

OOP Test

PATTERN	FREQUENCY	MERIT	DEPTH
ins	653	6980	2
sx	513	5430	2
fghi	336	3960	3
di	348	3780	3
bcd	140	1850	4
sr	80	1100	4
rq	75	1050	5
ihgf	20	800	5
sni	19	640	5
pkf	16	610	5

PRE_DELAY	:: 53326	PRE_AREA	:: 1050
REDUCED DELAY	:: 24873	REDUCED AREA	:: 187
DELAY OVERHEAD	:: 16292	AREA OVERHEAD	:: 396

그림 5. 프로그램 수행 결과의 한 예
Fig. 5. An Example of Program Execution.

라는 것은 빈도수로 (i,n,s)라는 부분 경로가 총 1000개의 샘플에서 653개의 빈도로 나온다는 것이다.

이를 바탕으로 네트워크 구조를 변화시키는데 "DEPTH" 값이 2와 3인 (i,n,s), (s,x), (f,g,h,i), (d,i)의 경우에는 끝점을 제외한 가운데 경로에 존재하는 라우터들은 라인으로 대체가 된다. 그리고 끝점의 경우에는 멀티플렉서 형태의 구조를 가지게 되는데 여기서는 i,s,x,f,d 의 라우터들은 멀티플렉서 구조로 바뀌게 되며, 나머지 부분 경로들은 간단한 멀티플렉서와 버스의 구조로 바뀌게 된다. 그리고 이 경로들을 지나게 되는 나머지 경로들은 경우에 따라 재 경로 탐색 또는 새로운 구조 추가가 필요하게 되고 오버헤드로 값이 처리된다.

<그림 5>에서 출력되는 "PRE_AREA"는 알고리즘 적용 전에 고정된 면적 값으로 1050이라는 값을 가지며, "REDUCED AREA"는 알고리즘 적용 후의 감소된 면적 값으로 기존의 1050보다 187 줄어들었다는 뜻으로, 최종적인 면적은 863의 값을 가지게 된다. "PRE_DELAY" 역시 알고리즘 적용 전의 지연 비용으로 53326이라는 값을 가지게 되며, "REDUCED DELAY"는 알고리즘 적용 후 감소하게 되는 지연비용으로 24873이라는 값만큼 줄었다는 말이다. "DELAY OVERHEAD"와 "AREA OVERHEAD"는 알고리즘을 적용할 때, 회로를 추가하게 되거나 경로 재설정 시에 늘어나게 되는 오버헤드로 인한 지연 시간과 면적 비용의 변화를 말하며 각각 지연비용 오버헤드는 16292, 면적비용 오버헤드는 396이라는 값을 가진다.

<그림 6>는 응용 프로그램 특성에 따라 통신패턴을 다르게 설정해 주었을 때의 실험 결과이다. 왼쪽에는 면적비용에 대한 그래프이고, 오른쪽에는 지연비용에 대한 그래프다. OON의 성능을 보여주기 위해, 허프만

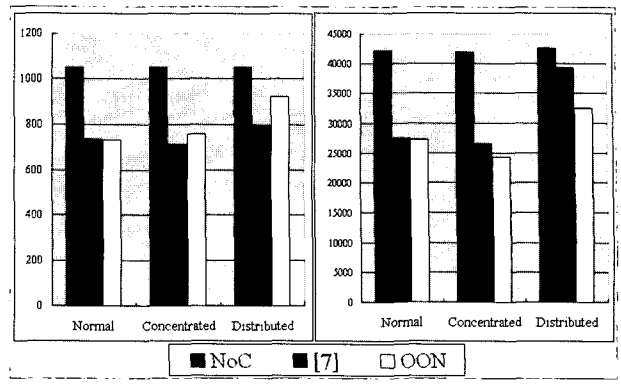

그림 6. 응용 프로그램 특성에 따른 실험 결과
Fig. 6. Experiment Results about Application Specification.

알고리즘을 적용하지 않은 [5]의 구조와 일반적인 NoC 구조와도 비교를 하였다. 그래프의 가로축을 보면 알 수 있듯이 통신패턴의 특성에 대한 분류로 "Normal", "Concentrated", "Distributed"라는 세 가지의 항목을 두었다. "Normal"은 통신패턴 생성 시에 특정 통신 모듈 간의 통신 확률을 다른 통신 모듈 간의 통신 확률보다 더 높게 한 기본 설정으로 한 경우다. "Concentrated"는 "Normal"의 경우에 비해 고정된(fixed) 형태의 통신패턴을 추가하고, 특정 모듈 간의 통신 확률을 더욱 높여준 경우다. "Distributed"는 "Concentrated"와는 반대로 특정 모듈 간의 통신 확률을 낮추고 대부분 임의적으로 (randomly) 생성하였다. "Normal"의 경우, [5]의 통신구조나 OON의 통신구조는 큰 차이가 없다. 하지만 OON의 경우는 통신패턴이 집중되거나 또는 분산될 경우, 성능 면에서 [5]에 비해 큰 향상을 보이고 있다. 결론적으로, OON을 사용하면 통신패턴의 변화에 큰 영향을 받지 않으면서 어느 정도 성능을 보장할 수 있다는 것이다. OOP의 통신패턴에 따른 향상 비율은 <표 2>에 정리되어 있다.

마지막으로 일반적인(Normal) 환경에서 OOP의 성능 향상에 대해 정리하면 <표 3>과 같다. 면적은 임의적으로 생성된 통신패턴에 따라서 자주 변화하지만 평균적으로 318 정도 감소되어 약 732의 값으로 측정되었다. 면적의 평균 오버헤드는 394.62의 값으로 측정되었

표 2. 통신패턴 관련도에 따른 향상
Table 2. Performance and Area Improvements about Communications Pattern.

통신패턴 관련도	매우 높음	높음	보통	낮음	매우 낮음
면적 향상 (%)	34.10	31.12	30.12	19.71	13.24
성능 향상 (%)	37.32	34.92	34.49	28.02	18.48

표 3. OOP의 성능 향상
Table 3. Improvements According to Experiments.

Criteria	NoC	Results	Improvement (%)
Total area cost	1050	732	30.29
Total delay cost	42121	27596	34.48
Total power	52621	34816	33.84

다. 지연비용은 통신패턴에 큰 변화 없이 평균 27596.11 정도로 측정되었고, 감소한 지연비용은 평균 26468.46이며 오버헤드는 평균 13875.61로 측정되었다. 이러한 총 면적비용과 총지연비용의 가중치 합을 이용하여 총전력이 33.84% 향상될 수 있음을 계산할 수 있었다.

VI. 결론 및 향후 계획

본 연구에서는 고적용성과 고확장성을 유지하면서 성능과 면적이라는 측면에서 최적의 통신 구조를 생성해주는 새로운 NoC 설계 방법론을 제시하였다. 이 설계 방법론은 응용 프로그램에 따른 통신패턴의 면밀한 분석을 통하여 매우 성능에 중요하고 또 빈번한 통신패턴에 대해서는 기존 NoC의 구조의 기반인 라우터와 네트워크 인터페이스의 오버헤드를 줄이기 위하여 보다 간단한 구조로 매핑하였다. 또한 통신구조의 특성인 유연성 및 적용성을 유지하기 위해 라우터 및 네트워크 인터페이스와 다른 통신구조들 간 비율을 선택하고, 통신패턴의 우선순위를 정해주는 알고리즘을 제시하였다. 허프만 트리 생성을 기반으로 하는 탐욕적인 알고리즘은 최적의 통신 구조를 찾는데 매우 유용하였다.

앞으로는 더 많은 응용 프로그램에서도 이 알고리즘이 얼마나 유용한지를 보일 것이다. 또한 통신 모듈들의 배치를 변경할 수 있는 경우에 다양한 변인들에 대해서 분석하고, 알고리즘을 어떻게 변형시켜 최적의 상태를 만들 것인가에 대한 연구가 필요할 것이다.

참 고 문 헌

[1] A. Hemani, et. al. "Network on a Chip: An architecture for billion transistor era," Proc. IEEE NorChip Conference, Nov. 2000.
[2] E. Rijpkema, et al., "A Router Architecture for Networks on Silicon", Proc. of Progress 2001, 2nd Workshop on Embedded Systems
[3] M. Millberg, et. al. "The Nostrum backbone - a communication protocol stack for networks on chip," Proc. VLSI Design, Jan. 2004.
[4] D. Bertozzi, et. al. "NoC synthesis flow for customized domain specific multiprocessor System-on-Chip," IEEE Trans. on Parallel and Distributed Systems, vol. 16, Feb. 2005.
[5] J. Yoon, "Incremental Mapping Technique of Optimal On-Chip Communication Structure", International SoC Design Conference, 2006.
[6] M. Coppola, et al. "OCCN: A Network-On-Chip Modeling and Simulation Framework", Proc. of DATE'04
[7] Steve Furber, "ARM: System-on-chip Architecture," 2nd Ed. Addision-Wesley, 2000.
[8] Leiserson, Cormen and Rivest, "Introduction to Algorithms", 2nd Edition, MIT Press, 2002.
[9] J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, "Interconnect-Centric Design for Advanced SoC and NoC", 1st Edition, Springer, pp. 25-124 , 2004.
[10] W. E. Donath, "Placement and Average Interconnection Lengths of Computer Logic", IEEE Trans. Circuits & Syst., vol. CAS-26, pp. 272-277, 1979.

저 자 소 개

윤 주 형(정회원)

2005년 한양대학교 정보통신 공학과 학사 졸업.
2007년 한양대학교 정보통신 공학과 석사 졸업.
2007년 현재 이오테크닉스기술 연구소 연구원

<주관심분야 : 결함 포용 컴퓨팅, SoC를 위한 동적 암호화 기법>

정 기 석(정회원)

1989년 서울대학교 컴퓨터공학과 공학사
1998년 University of Illinois at Urbana-Champaign 전산학 박사
1998년 University of Illinois at Urbana-Champaign, 강의 전담 교수
2000년 Synopsys, Inc. Sr. R&D Engineer
2001년 Intel Corp. Staff Engineer
2004년 홍익대학교 컴퓨터공학과 조교수
2007년 현재 한양대학교 정보통신대학 조교수

<주관심분야 : 임베디드 시스템, System-on-Chip 설계, 저전력 시스템 설계, VLSI/CAD>

황 영 시(정회원)

2003년 대진대학교 컴퓨터 공학과 학사 졸업.
2005년 (주)한국정보통신 기술연구소 연구원.
2007년 현재 한양대학교 정보통신 공학 석사 재학.

<주관심분야 : 임베디드 시스템, 저전력 시스템 설계, RTOS>