온 칩 버스 구조와 메모리 할당에 대한 효율적인 설계 공간 탐색

(Efficient exploration of on-chip bus architectures and memory allocation)

요 약 시스템 수준 설계에서 계산 부분과 통신 부분의 분리는 프로세서의 선택이나 기능 블록의 프로세서에 대한 할당 결과에 관계없이 설계자로 하여금 독립적인 통신 구조의 설계 공간 탐색을 가능하게 해준다. 본 논문은 버스 기반의 온 칩 통신 구조와 메모리 할당의 최적화를 위한 2 단계 설계 공간 탐색 방법을 제안한다. 제안된 설계 공간 탐색 방법은 정적 성능 예측 방법을 사용하여 통신 구조에 대한 방대한 설계 공간을 빠르고 효과적으로 줄인다. 이렇게 축소된 통신 구조들의 설계 공간에 대해서는 정확한 성능 예측을 위하여 프로세서들의 메모리 트레이스를 이용한 트레이스 기반 시뮬레이션을 적용한다. 프로세서들의 동시적인 접근에 의한 버스의 충돌은 프로세서간 공유 메모리뿐 아니라 프로세서의 로컬 메모리에서도 기인하므로 메모리 할당 또한 중요하게 다루어져야 하는 부분이다. 제안된 설계 공간 탐색 방법의 효율성은 4-채널 DVR 과 OFDM DVB-T 용 수신기 내부의 이퀄라이저 부분을 이용하여 검증하였다.

키워드 : 설계 공간 탐색, 온 칩 버스, 메모리 할당

Abstract Separation between computation and communication in system design allows the system designer to explore the communication architecture independently of component selection and mapping. In this paper we present an iterative two-step exploration methodology for bus-based on-chip communication architecture and memory allocation, assuming that memory traces from the processing elements are given from the mapping stage. The proposed method uses a static performance estimation technique to reduce the large design space drastically and quickly, and applies a trace-driven simulation technique to the reduced set of design candidates for accurate performance estimation. Since local memory traffics as well as shared memory traffics are involved in bus contention, memory allocation is considered as an important axis of the design space in our technique. The viability and efficiency of the proposed methodology are validated by two real-life examples, 4-channel digital video recorder (DVR) and an equalizer for OFDM DVB-T receiver.

Key words : design space exploration, on-chip bus, memory allocation

1. 서론

내장형 시스템의 성능 요구치가 끊임없이 증가함에 따라 이를 만족하기 위하여 하나의 SoC (System-on-a-Chip)에 점점 더 많은 프로세서들을 집적하는 것이 불가피하게 되었다. 이러한 고성능 SoC의 설계를 위하여 시스템의 기능과 구조의 분리, 계산과 통신의 분리를 고려하는 시스템 설계 방법들이 대두되고 있다 [1]. 본 논문에서는 이러한 설계 방법론에 따라 시스템을 기능 블록들의 조합으로 표현하고, 각 기능 블록들은 명시된 타깃 아키텍처에서 사용되는 프로세서들에 할당된다고 가정한다.

시스템을 통신 부분과 계산 부분으로 분리하는 것은 어떤 프로세서가 사용되는가와 기능 블록들이 어떤 프로세서에 할당되는가에 관계없이 설계자로 하여금 통신 구조에 대한 최적화를 가능하게 해 준다. 이러한 설계 방법론에서 통신 구조의 결정은 사용할 프로세서들의 선택과 시스템을 구성하는 기능 블록들의 프로세서에 대한 할당 후 이루어진다. 통신 구조에 대한 설계 공간 탐색의 목적은 선택된 모든 프로세서들의 통신 구조에 대한 요구 조건과 성능, 전력 소모, 가격 등의 상관 관계를 고려하여 최적화된 구조를 찾기 위함이다. 본 논문에서 집중적으로 다루어질 이러한 통신 구조의 최적화는 매우 큰 설계 공간을 대상으로 하므로 효율적이고 빠른 탐색 방법이 필수적이다.

통신 구조는 일반적으로 가장 많이 사용되는 버스 기반으로 한정한다 [2][3][4]. 그럼에도 불구하고, 버스의 개수와 버스 브리지에 의한 토폴로지, 프로세서의 버스에 대한 할당, 버스 사용 요청 중재 방법, 버스 동작 클록, 데이터 버스의 폭 등의 다양한 성능 인자들에 의해 여전히 매우 방대한 설계 공간이 형성된다.

이러한 설계 공간을 탐색하는 방법이 실효성을 가지려면 정확성과 빠른 실행 속도라는 두 가지의 상충하는 조건들을 가진 성능 예측 방법이 중요한 요소가 된다. 시뮬레이션 기반의 성능 예측 방법은 정확한 결과를 제공하지만 너무 긴 실행 시간으로 인해 넓은 설계 공간을 탐색하기에 부적당하다. 따라서 이를 이용한 탐색 방법은 제한된 성능 인자들에 대해서만 구성된 설계 공간에 적용할 수 있다. 반면 실행 속도 면에서 효율적인 정적인 성능 예측 방법은 동적인 버스 충돌을 고려하지 못해 시뮬레이션 기반의 방법에 비해 부정확한 실행시간 예측이 단점이다. 본 논문에서는 통신 구조의 설계 공간을 두 단계에 걸쳐 탐색함으로써 두 가지 성능 예측 방법의 장점을 모두 이용한다. 첫 번째 단계에서 다양한 통신 구조들을 정적인 성능 예측 방법을 이용하여 빠르게 검증한 후 전체 탐색 공간의 상당 부분을 줄인다. 두 번째 단계에서는 메모리 트레이스 기반의 시뮬레이션 방법을 이용하여 이미 줄어든 탐색 공간의 통신 구조의 성능을 정확하게 성능을 예측하여 버스 기반의 통신 구조에 대하여 다양한 성능 인자들에 대한 파레토-최적화된 결과들을 얻을 수 있다.

시스템은 프로세서간의 통신을 위해 공유 메모리를 이용하는 것으로 가정한다. 또한 각 프로세서들은 메모리의 접근을 위해 하나의 데이터 버스만을 이용한다. 이는 실제 시스템 설계에서 일반적으로 이용되는 구조이다. 이에 따라서 공유 메모리뿐 아니라 로컬 메모리에 대한 접근도 버스 충돌을 일으킬 수 있으므로 메모리 할당 또한 설계 공간을 구성하는 중요

한 하나의 축이 된다. 기존의 연구들에서는 공유 메모리의 접근만을 다루었기 때문에 메모리 할당 문제를 버스 최적화와 동시에 고려하지 않았다 [5][6].

본 논문은 다음과 같이 구성된다. 2장에서는 제안하는 설계 공간 탐색 방법을 간단한 예제를 통하여 개략적으로 설명한다. 3장에서는 관련된 연구들이 소개된다. 4장과 5장에서는 실생활 예제를 이용하여 제안하는 탐색 방법에 구체적인 설명이 이루어진다. 6장에서는 몇 가지 예제들을 이용한 실험 결과가 제시되고 7장에서 결론을 맺을 것이다.

2. 제안하는 탐색 방법의 개요

제안하는 통신 구조의 설계 공간 탐색 방법에 대한 개략적인 이해를 위하여 그림 1에 나타나 있는 간단한 예제를 이용한다. 제시된 예제 시스템은 4개의 기능 블록들로 이루어진 다이어그램으로 표현되었다. 기능 블록들 사이의 연결선은 실행순서의 종속 관계와 데이터 통신을 나타낸다. 예를 들어, 기능 블록 B와 D는 기능 블록 A가 완료될 때까지 실행될 수 없다. 4개의 기능 블록들은 3개의 프로세서에 할당된다. 기능 블록 A와 C는 프로세서 PE0에, B는 PE1에, C는 PE2에 각각 할당되었다고 가정한다.

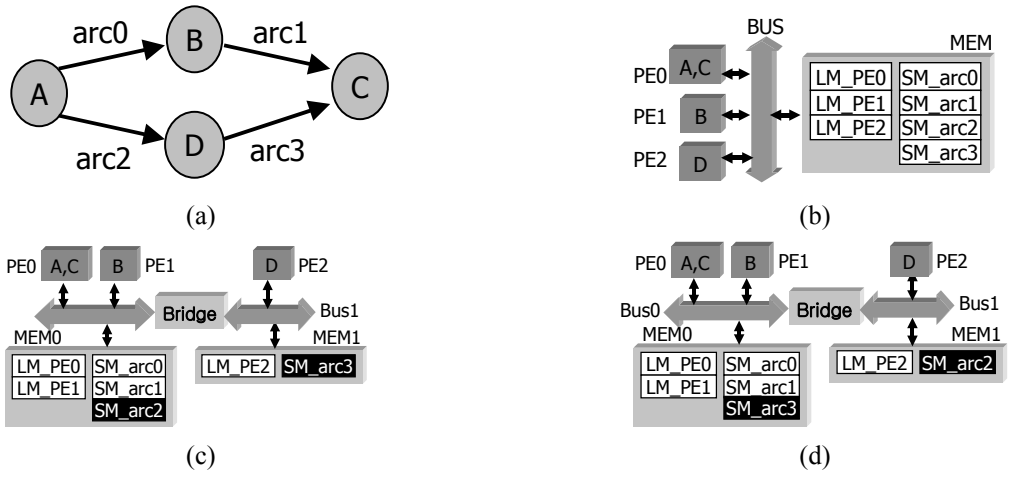


그림 1. (a) 제시된 예제에 대한 블록 다이어그램 표현, (b) 단일버스구조 구현, (c)와 (d) 이중 버스구조를 이용한 구현, 각 구현은 공유 메모리 세그먼트인 SM_arc2와 SM_arc3이 각각 다른 버스에 할당되어 있다.

그림 1(b)은 단일버스구조를 이용한 구현을 나타낸다. 버스에는 하나의 물리적인 메모리가 연결되어 있고 그 안에 여러 개의 논리적인 메모리 세그먼트가 포함되어 있다. 여기에서는 3개의 로컬 메모리 세그먼트와 4개의 공유 메모리 세그먼트가 해당된다. 메모리 세그먼트 LM_PE0, LM_PE1, LM_PE2는 각각 프로세서 PE0, PE1, PE2의 로컬 메모리에 해당한다. 서로 다른 프로세서에 할당된 기능 블록들 사이의 연결선은 프로세서간 통신을 위한 공유 메모리 세그먼트를 나타낸다. 예를 들어 SM_arc0은 그림 1(a)의 arc0을 나타내고 이는 기능 블록 A와 B사이의 통신에 사용된다. 제안하는 탐색 방법의 전체 흐름도는 그림 2에

나타나 있다. 탐색 과정은 단일버스구조에서 시작한다. 이는 전체 탐색의 처음 부분인 "Set of architecture candidates"의 유일한 구성 요소가 된다.

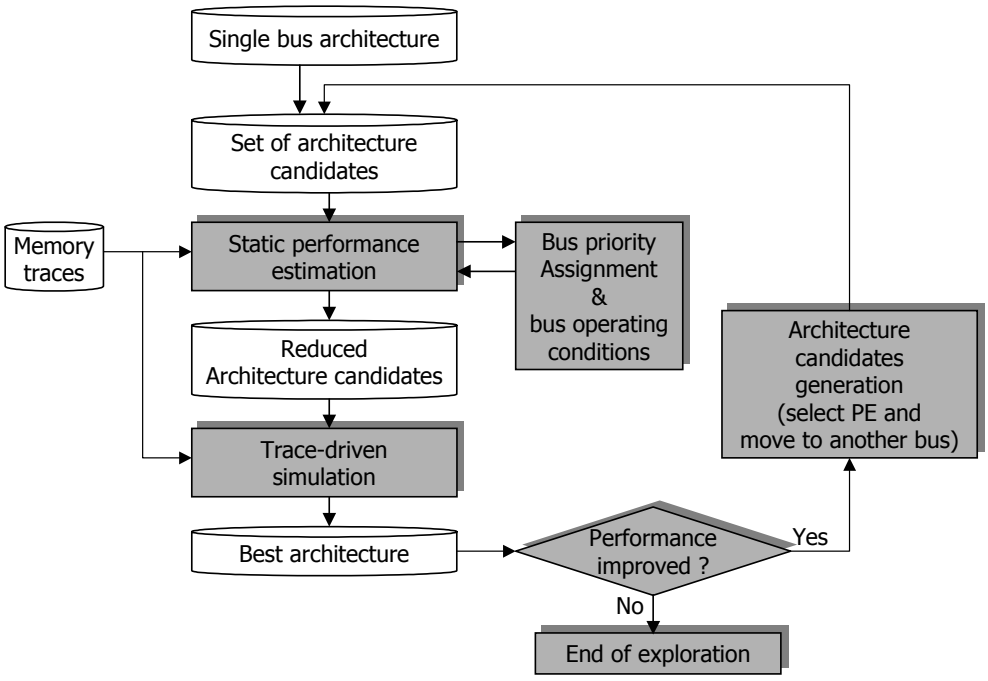


그림 2. 제안하는 설계 공간 탐색의 흐름도.

사용되는 모든 프로세서에 대한 로컬 메모리, 공유 메모리를 포함하는 메모리 접근 트레이스는 제안하는 탐색 방법에 대한 입력 중의 하나이다. 기능 블록의 프로세서에 대한 할당이 결정된 후, 메모리 트레이스는 마이크로프로세서의 경우 사이클 단위의 정확성을 갖는 프로세서 시뮬레이터에서, ASIC 부분은 HDL 시뮬레이터나 IP 시뮬레이터로부터 얻어진다. 메모리 트레이스는 코드 메모리, 데이터 메모리, 공유 메모리의 3부분으로 나뉜다. 코드 메모리와 데이터 메모리는 로컬 메모리 접근 시에, 공유 메모리는 프로세서간 통신에 발생하는 트레이스에 해당된다. 캐시를 사용하는 프로세서의 경우 코드 메모리와 데이터 메모리에 관련된 트레이스는 캐시 미스에 의해 발생하는 메모리 접근을 의미한다.

설계 공간의 탐색은 그림 2에 나타난 바와 같이 탐색 과정의 반복적인 수행으로 이루어진다. 한번의 탐색 과정은 3단계로 이루어져 있다. 첫 번째 단계의 목적은 다양한 통신 구조들로 이루어진 설계 공간을 빠르게 탐색하여 두 번째 단계에서 자세하고 정확하게 탐색되어야 할 통신 구조들을 선택한다. 주어진 통신 구조에 대하여 우선순위 할당과 기타 버스 동작에 필요한 조건들은 첫 번째 단계에서 최적화 된다. 본 논문에서 사용되는 정적인 성능 평가 방법은 시뮬레이션 기반의 성능 평가 방법과 비교하여 10% 이내의 오차를 갖고 있으므로, 첫 번째 단계에서 전체 통신 구조의 설계 공간 중 가장 성능이 우수한 통신 구조와 비교하여 10% 이내의 오차를 갖는 통신 구조만을 가려낸다. 이를 이용하여 전체 탐색 공간 중 상당 부분을 제거할 수 있다.

첫 번째 단계에서 사용된 성능 예측 방법은 단일 버스 구조의 시스템에서 버스와 메모리

시스템이 단일 서버가 되고 버스에 연결된 프로세서들이 클라이언트가 되는 큐잉 모델에 기반하고 있다 [7]. 큐잉 모델을 구성하기 위해서는 프로세서에서 실행되는 기능 블록 fb의 평균 버스 요구 비율 ar(fb), 스케줄 길이 sl(fb), 버스 사용권을 얻는 데 소요되는 시간이 없고 단위 데이터 전송에 한 사이클이 걸리는 이상적인 버스에서의 평균 버스 이용 시간 ba(fb)의 세 가지 인자가 필요하다. 버스 요구 비율은 버스 접근 횟수 bc(fb)를 sl(fb)로 나눈 것이다. 이러한 인자들을 이용하여 각 프로세서들이 버스의 사용권을 얻기 위해 소모하는 평균 시간은 동적인 버스 충돌을 고려하여 그림 3과 같이 얻어진다.

기능 블록들의 스케줄이 미리 결정되어 있는 하나의 태스크에서 기능 블록들이 버스를 요구하는 구간은 스케줄에 따라 초기에는 그림 3(a)와 같이 정적으로 결정된다. 스케줄의 시작인 T_0에서 세 개의 기능 블록 A, B, C는 각각 프로세서 PE_0, PE_1, PE_2에서 병렬적으로 실행될 수 있다. 각 프로세서들이 한번의 버스의 사용권을 얻기 위해 소모하는 평균 시간을 구하기 위한 큐잉 시스템은 그림 3(a)와 같이 구성된다. 이를 풀면 각 프로세서들이 버스를 얻기 위해 기다려야 하는 평균대기 시간을 구할 수 있고, 한번의 버스 접근에서의 평균 버스 이용 시간, 기능 블록 실행 시 프로세서 내부 실행시간 등을 이용하면, 가장 먼저 실행을 마치는 기능 블록의 종료 시간을 구할 수 있다. 이 때 기능 블록의 실제 실행 시간은 버스에서의 동적 충돌에 의해 버스를 얻기 위해 기다려야 하는 평균 대기 시간을 고려하여 초기 스케줄 길이보다는 길어지게 된다. PE_2가 실행하는 기능 블록 C가 가장 먼저 T_1에서 끝났다고 가정하면, 이 후에 병렬적으로 실행할 수 있는 기능 블록들을 고려해야 한다. 따라서 그림 3(b)처럼, 기능 블록 A와 B가 T_1 이후 병렬적으로 수행될 수 있으므로, PE_0와 PE_1로 이루어진 새로운 큐잉 모델이 구성된다. 그림 3의 (a), (b)에서 스케줄들의 회색으로 채워진 부분들은 정적인 성능 예측 방법에 의해 평가 되어야 하는 부분들이다. 이러한 과정은 모든 기능 블록들이 평가 될 때까지 반복된다.

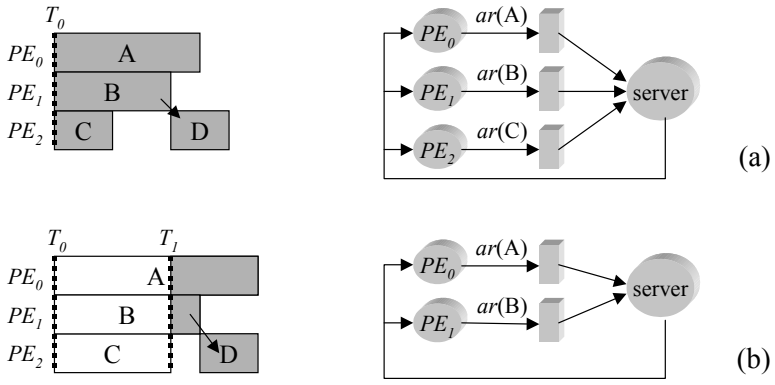


그림 3. (a) 프로세서 PE_0, PE_1, PE_2의 스케줄과 초기 큐잉 모델, (b) T_1에서 기능 블록 C의 실행이 완료된 이후의 새로운 큐잉 모델.

두 번째 단계는 첫 번째 단계에서 선택된 설계 공간들에 대해 트레이스 기반의 시뮬레이션 방법을 적용하는 것이다. 이를 이용하여 축소된 설계 공간들의 통신 구조들에 대한 정확한 성능 평가가 가능하며 가장 우수한 성능을 갖는 통신 구조를 찾아낼 수 있다. 가장 우수

한 통신 구조의 성능이 이전 탐색 과정에서 선택된 가장 우수한 통신 구조에 비해 성능의 개선이 없으면 탐색이 종료된다. 그렇지 않으면 세 번째 단계를 거친 후 다시 탐색 과정의 첫 번째 단계로 가는 과정을 반복한다.

본 논문에서 사용된 트레이스 기반의 시뮬레이터는 AMBA AHB 버스[3]의 간략화 된 프로토콜을 사이클 단위로 정확하게 모델링하고 있다. 기본적인 동작은 사이클 단위로 버스와 버스에 연결된 프로세서들의 내부 상태를 갱신함으로써 시뮬레이션 되는 시간을 진행시켜 나가는 것이다. 현재의 구현에서 파이프라인 방식의 주소/데이터 버스, 스플릿 트랜잭션, 에러 처리 등의 기능은 지원되지 않는다. 좀더 살펴보면, 시뮬레이터에서 버스의 상태는 bus-arbitration, start-address-drive, sequential-burst-transfer, last-data-drive의 4가지로 구분할 수 있다. 그림 4는 앞에서 설명된 버스 모델에 따른 데이터 전송의 한 예를 보여주고 있다. ‘A'로 표시된 구간은 버스 사용권을 얻기 위해 프로세서가 기다리는 시간을 나타낸다. 이는 버스에서 다른 프로세서와의 충돌에 따라 가변적이다. 구간 B, C, E는 버스 프로토콜에 따라 정적으로 소모되는 시간을 나타내며 각각 1 사이클씩 걸리는 것으로 설정되었다. 구간 D는 메모리에서 하나 이상의 단위 데이터를 읽어오는데 걸리는 시간을 나타낸다. 메모리는 연속 전송을 위한 초기화에 필요한 시간, 하나의 단위 데이터를 읽어오는데 걸리는 시간의 두 가지 인자로 표현된다. 그림 4에 나타난 바와 같이 초기화는 1 사이클, 단위 데이터를 읽어오는데 걸리는 시간은 2 사이클로 설정되어 있다.

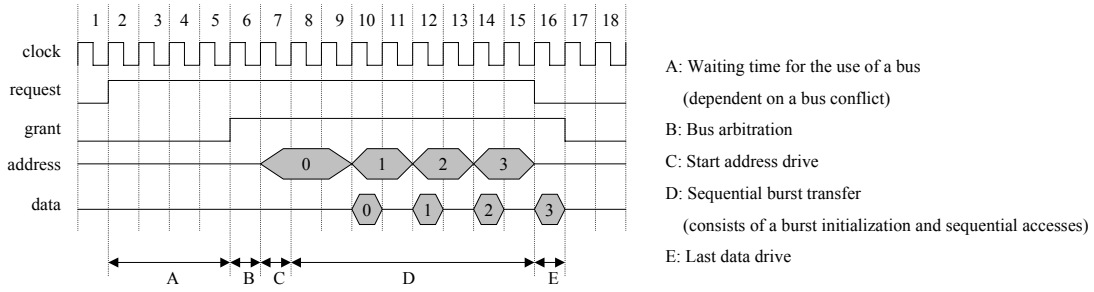


그림 4. 트레이스 기반 시뮬레이터의 버스 프로토콜에 따른 4-워드 데이터 전송의 예.

세 번째 단계에서는 두 번째 단계에서 선택된 가장 우수한 통신 구조로부터 다음번의 탐색 과정에 적용될 설계 공간이 되는 통신 구조들을 생성한다. 임의의 선택된 통신 구조로부터 하나의 프로세서를 선택하여 이를 기존의 다른 버스에 할당하거나 새로운 버스를 생성하여 할당하는 방법으로 새로운 설계 공간을 생성할 수 있다. 그림 1의 예제를 다시 생각해 보자. 초기에는 단일버스구조로 이루어진 하나의 통신 구조만 존재한다고 생각하면, 이는 한번의 전체 탐색 과정 중 세 번째 단계에 대한 입력이 된다. 이로부터 이중버스구조를 만들기 위해 새로운 버스가 생성되었고 여기에 PE2가 연결된다고 가정해 보자. 프로세서의 로컬 메모리 세그먼트는 프로세서가 할당되어 있는 버스에 같이 할당되는 것이 당연하므로, 공유 메모리의 할당만을 고려한다면, 2개의 공유 메모리 SM_arc2, SM_arc3가 어느 버스에 할당되는가에 따라서 4개의 통신 구조를 생성할 수 있다. 그러나 공유 메모리 SM_arc0는

기능 블록 A와 B에 의해 이용되므로 이들과 관련된 프로세서들이 연결되어 있는 bus0에만 할당되어야 한다. 4개의 가능한 통신 구조들 중 2가지를 그림 1(c), (d)에 나타내었다. PE0를 선택하여 이를 새로운 버스에 할당할 경우 PE0는 4개의 공유 메모리 세그먼트를 가지므로 16개의 통신 구조를 생성할 수 있다. 이러한 방법으로 초기의 단일버스구조로부터 프로세서를 선택하여 새로운 버스에 할당하고, 이와 관련된 공유 메모리들의 버스에 대한 할당을 고려하는 방법으로 24개의 통신 구조를 생성할 수 있다. 이는 두 번째 반복되는 탐색 과정의 입력으로 이용된다.

탐색이 반복적으로 진행됨에 따라, 사용되는 버스의 개수에 따라 가장 좋은 성능을 갖는 구조를 기록하여 파레토-최적화된 탐색 결과를 얻을 수 있다. 일반적으로 버스의 개수가 늘어날수록 성능도 향상될 것이다. 이전 탐색에서 얻은 결과에 비해 현재의 탐색에서 성능의 개선이 없을 경우 전체 탐색을 종료한다. 본 논문에서 제안하는 탐색 방법은 모든 설계 공간을 탐색하는 것은 아닌 이전 탐색에서 얻어진 가장 우수한 통신 구조에서 생성된 설계 공간만을 탐색하는 휴리스틱을 사용한다. 만일 가장 우수한 하나의 통신 구조만을 선택하는 것이 아니라 여러 개를 선택한다면 더 많은 설계 공간을 탐색할 수 있을 것이지만 더 많은 탐색 시간이 소요될 것이다.

3. 관련 연구들과 본 논문의 의의

통신 구조는 계산 부분에 대한 합성과 기능 블록의 프로세서에 대한 할당이 이루어지는 과정에서 동시에 고려되거나 이후에 별도로 고려할 수 있다. 전자의 경우에 통신에 소요되는 시간은 기능 블록을 프로세서에 할당하는 과정에서 중요한 정보가 된다. 이를 위해 통신 구조의 성능을 정적으로 평가하는 방법에 대한 많은 연구가 이루어졌다. Yen과 Wolf는 실시간 스케줄링의 최악시간 응답을 분석하는 방법을 이용하여 통신에 소요되는 시간을 예측하는 방법을 제시하였다 [8]. 또한 Knudsen과 Madsen은 통신 구조의 프로토콜 및 다양한 인자들, 프로세서의 클록 속도 등을 고려하여 프로세서간 점대점 방식으로 연결된 채널에 대한 통신 구조의 성능을 분석하였다 [9]. Nandi와 Marculescu는 연속 시간 마르코프 프로세스에 기반을 둔 통신 구조의 성능 분석 방법을 제시하였다 [10]. 그러나 이러한 방법들은 버스 충돌과 같은 동적인 거동을 효과적으로 분석할 수 없고, 이로 인해 제한된 설계 공간의 탐색에만 사용될 수밖에 없는 단점이 있다.

통신 구조의 설계 공간 탐색을 위해서 시뮬레이션 기반의 성능 평가 방법이 상업화 된 툴과 기존 연구들에 적용되었다 [11][12]. 시뮬레이션 기반의 성능 평가 방법은 정확한 결과를 제공하지만 넓은 설계 공간을 탐색하기에는 너무 긴 실행 시간을 요구한다. 따라서 이에 기반을 둔 연구들은 설계 공간을 줄이기 위해 제한된 성능 요소들에 국한된 탐색이 가능했다. 이러한 단점을 극복하기 위하여 정적인 성능 분석 방법과 시뮬레이션 기반의 성능 분석 방법의 장점을 모두 이용한 통신 구조의 설계 공간 탐색 방법이 Lahiri 외 2인에 의해 개발되었다 [13]. 이 방법은 메모리 트레이스를 정적인 분석을 통하여 여러 개의 그룹으로

나눈 후 이를 트레이스 기반의 시뮬레이션에 적용한다. 이는 시뮬레이션에 소요되는 시간을 줄이기 위하여 트레이스를 이용한 정적인 성능 분석 방법을 적용한다는 측면에서는 본 논문에서 제시하는 탐색 방법과 비슷하다. 그러나 전체적인 접근방법은 상이하며, 로컬 메모리에 대한 접근 또한 고려되지 않았다.

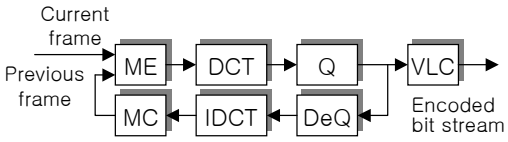
일반적으로 설계 공간은 매우 방대하기 때문에, 대부분의 이전 연구들은 설계 공간을 구성하는 요소들 중에서 일부분만을 다루었다. Gong외 2인의 연구는 시스템 명세 후 성능의 최적화를 위하여 미리 결정된 4개의 통신 구조 중 하나를 선택하는 방법을 제시하였다 [14]. 또한 Gastier외 2인의 연구에서는 데이터 폭, 데이터 전송량, 사용되는 프로세서의 입출력 포트 구성 등과 같은 정적인 정보를 이용하여 상위 수준에서 버스 기반 통신 구조의 토폴로지를 결정하는 방법을 제시하였다 [15]. Meeuwen외 3인은 미리 결정된 분산 메모리 시스템에서 버스의 개수에 따른 데이터 전송의 시간분포를 고려한 효율적인 버스 구조를 생성하는 방법을 제시하였다 [6]. 또한 Meftaili외 3인은 점대점 통신 구조에서 정수선형프로그램(ILP)를 이용하여 통신 채널과 메모리 시스템의 하드웨어 면적을 고려하여 성능을 최적화하는 공유 메모리의 할당 방법을 제시하였다 [16]. Lahiri외 2인의 연구에서는 고정된 버스 토폴로지에 대하여 DMA 전송시의 사용되는 데이터 블록의 크기와 버스 우선순위 할당 등의 버스 프로토콜과 프로세서의 버스에 대한 할당을 최적화하는 설계 공간 탐색 방법이 제시되었다 [5].

기존의 연구들과 비교해 볼 때, 본 논문의 크게 두 가지 의의를 갖는다. 첫 번째로, 본 논문은 버스의 개수, 버스 토폴로지, 프로세서의 버스에 대한 할당, 버스 우선순위 할당 및 기타 버스 동작에 관련된 구성 요소 등 다양한 인자들을 고려하여 방대한 설계 공간을 탐색한다. 본 논문에서 제시된 탐색 방법은 체계적이고 확장 가능하기 때문에 보다 많은 설계 공간에 관련된 인자들을 쉽게 추가할 수 있다. 두 번째로 본 논문에서는 공유 메모리 뿐 아니라 로컬 메모리까지 고려한다. 통신 구조의 설계 공간 탐색에 관련된 기존의 연구들은 로컬 메모리에 관련된 메모리 접근을 고려하지 않았다. 그러나 로컬 메모리의 접근에 의한 버스 충돌도 생길 수 있기 때문에 로컬 메모리의 접근도 고려되어야 한다.

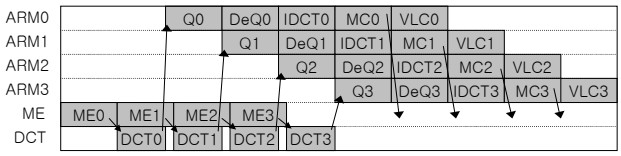
4. 4-채널 DVR에 대한 설계 공간 탐색

이 장에서는 실생활 예제인 4-채널 디지털 비디오 녹화기(DVR)를 통하여 제안하는 탐색 방법이 어떻게 이루어지는지 알아본다. DVR은 4개의 외부 영상을 입력 받아서 각각을 H.263 부호화 알고리즘을 이용하여 부호화한다. 그림 5(a)는 H.263의 시스템 명세를 나타낸다. 하나의 H.263 부호기에서는 ME와 DCT 블록을 제외한 모든 블록이 하나의 ARM720T 프로세서에 할당된다. 4 채널의 ME와 DCT들은 각각 한 개의 전용 ME 하드웨어와 DCT 하드웨어에 할당되었다. 따라서 ME 하드웨어와 DCT 하드웨어는 4개의 H.263 부호기들에 의해 공유된다. 그림 5(b)~(d)는 4-채널 DVR의 스케줄링과 기능 블록의 프로세서에 대한 할당 결과, 단일버스구조의 구현을 보여주고 있다. 모든 채널들은 QCIF 크기의

P-프레임을 부호화하는 과정에서 얻어진 동일한 메모리 트레이스를 사용한다. 4-채널 DVR 시스템은 5개의 로컬 메모리와 9개의 공유 메모리로 이루어진 14개의 메모리 세그먼트들을 갖고 있다. 예를 들어 그림 5(d)에서, 공유 메모리 세그먼트 'MC0, ME0'는 기능 블록 MC0와 ME0를 각각 실행하는 MC 하드웨어와 ME 하드웨어 사이의 통신에 이용된다.



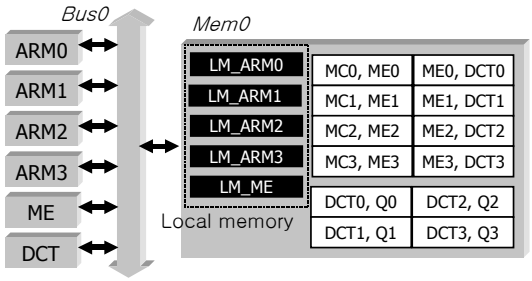
(a)



(b)

PE	Access count
ARM0	1422860
ARM1	1422860
ARM2	1422860
ARM3	1422860
ME	139392
DCT	304128

(c)



(d)

그림 5. (a) H.263 부호기의 명세, (b) 4-채널 DVR의 초기 스케줄, (c) 메모리 트레이스, (d) 단일버스구조 구현.

4.1. 다양한 통신 구조들의 생성

통신 구조의 성능 향상은 하나의 버스에서 동시에 발생하는 서로 다른 메모리 접근을 다른 버스로 분산시켜 충돌을 줄이고 병렬성을 증가시킴으로써 이루어질 수 있다. 이러한 목적으로 임의의 프로세서를 선택하여 이를 새로운 버스나 기존에 다른 버스에 할당한다. 그림 5(d)의 단일버스구조 구현에서 ARM0를 선택하여 이동시킨다고 가정해 보자. 이 경우 ARM0를 다른 버스에 할당하는 방법은 새로운 버스 bus1을 생성하는 것뿐이다. 이 때 공유 메모리 세그먼트인 'MC0, ME0'와 'DCT0, Q0'는 bus0나 bus1 중의 한 곳에 할당될 수 있으므로 이를 이용하여 그림 6에 나타난 바와 같이 4개의 통신 구조가 생성된다. 설명의 편의성을 위해 버스를 3개 이상 거치는 메모리 접근은 없다고 가정한다.

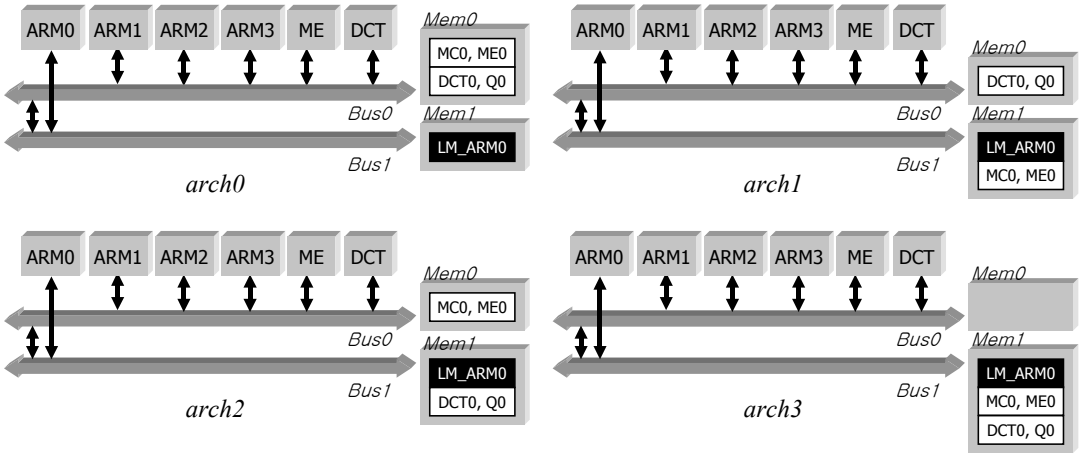


그림 6. 프로세서 ARM0가 할당될 새로운 버스를 생성하고 관련된 공유 메모리를 할당함으로써 생성되는 통신 구조들.

표 1. 4-채널 DVR의 단일버스구조로부터 공유 메모리 세그먼트들을 이동하여 생성되는 통신 구조들의 개수.

할당되는 프로세서	ARM0	ARM1	ARM2	ARM3	ME	DCT
공유 메모리 세그먼트 개수	2	2	2	2	8	8
생성된 통신 구조 수	2^2	2^2	2^2	2^2	2^8	2^8
생성된 통신 구조 총합	528					

표 1은 그림 5(d)의 단일버스구조로부터 프로세서들을 새로운 버스로 할당함으로써 생기는 통신 구조의 수를 나타내고 있다. 전체적으로 528개의 통신 구조가 생성된다. 이를 통하여 모든 설계 공간을 고려하지 않더라도 단지 휴리스틱에 의해서도 얼마나 많은 설계 공간을 탐색해야 하는가를 쉽게 알 수 있다.

4.2. 생성된 통신 구조들에 대한 프로토콜 최적화

버스 토폴로지와 버스에 대한 메모리 세그먼트들의 할당이 결정된 후에 생성된 각 통신 구조들의 프로토콜들이 결정되어야 한다. 여기에서 버스 프로토콜은 버스 우선순위 할당, 버스의 클록 속도, 데이터 버스의 폭 등을 의미한다. 이러한 프로토콜의 요소들 중, 버스 우선순위 할당은 [5]에서 보고된 것처럼 통신 구조의 성능에 지대한 영향을 끼칠 수 있으므로 이에 대한 최적화는 상당히 중요하다. 가능한 모든 우선순위 할당 경우를 찾아보는 방법은 항상 최적의 결과를 가질 것이지만, 실제적으로 쓰이기 어렵다. 예를 들어 그림 6의 첫 번째 통신 구조 arch0를 살펴보면, bus0에는 버스 브리지를 포함한 6개의 버스 마스터와 bus1에는 2개의 버스 마스터가 존재한다. 그러므로 bus0에 대해서는 6! 또는 1440개, bus1에 대해서는 2!개의 우선순위 할당에 대한 경우의 수가 존재하므로 arch0 하나에 대해

서 최적의 우선순위 할당 방법을 찾기 위하여 1440×2개에 해당하는 우선순위 할당 경우의 수에 대하여 조사해야 한다. 이를 개선하기 위하여 본 논문에서는 우선순위 할당 방법에 관한 휴리스틱을 개발하였다. 기본 원리는 메모리 접근 횟수가 많거나 시스템의 임계 실행경로에 있는 프로세서일수록 버스 접근에 있어 높은 우선순위를 할당하는 것이다.

기능 블록 fb_i의 단위 시간당 데이터 전송량을 $BW(fb_i)$로 나타내자. $C(fb_i)$는 기능 블록 fb_i가 끝나야 실행될 수 있는 기능 블록들의 실행 경로들 중에서 가장 긴 실행 경로에 놓인 기능 블록들의 실행 시간의 합을 나타낸다. $BW(fb_i)$은 메모리 트레이스로부터 얻을 수 있고, $C(fb_i)$는 기능 블록들의 스케줄로부터 얻을 수 있다. 여기에서, $R(fb_i)$을 기능 블록 fb_i의 $C(fb_i)$과 $BW(fb_i)$의 곱으로 정의하자. 또한 R(PE)는 프로세서 PE 위에서 실행되는 기능 블록들의 $R(fb_i)$들의 합을 나타낸다. 이를 이용하여 아래의 식을 얻는다.

$$R(PE) = \sum_{fb_i \in FB_{PE}} R(fb_i) = \sum_{fb_i \in FB_{PE}} \{BW(fb_i) \cdot C(fb_i)\}. \quad (1)$$

여기에서 FB_{PE}는 프로세서 PE에 할당된 기능 블록들의 집합을 나타낸다. R(PE)가 높을수록 프로세서 PE는 높은 우선순위를 할당 받는다. 이렇게 하여 초기 우선순위할당이 끝나고 나면 두 번째 최적화 방법으로써 좀더 최적화된 우선순위 할당을 위하여 같은 버스에 할당된 2개의 프로세서들의 우선순위를 바꾸어서 각각의 경우에 대하여 성능을 평가해 본다. 2개 이상의 프로세서들에 대하여 한꺼번에 우선순위를 바꾸는 경우 보다 높은 성능을 얻을 수도 있지만 보다 많은 탐색시간을 요구할 것이다. 따라서 2개의 프로세서들의 우선순위를 바꾸어 보는 것으로 최적화방법을 제한한다. 예를 들어 그림 6에 나타난 arch0의 경우 bus0와 bus1에 대하여 두 번째 최적화 방법을 통해 각각 $\binom{6}{2}$와 $\binom{2}{2}$개의 우선순위 할당방법만을 조사하면 된다. 제안된 우선순위 할당방법은 모든 경우를 평가하는 방법과 비교하여 탐색해야 할 설계 공간을 상당 부분 줄이면서도 그 결과는 크게 떨어지지 않는 우수한 성능을 나타내었다. 6장에서 실험 결과를 통하여 본 논문에서 제안한 우선순위 할당법의 효율성을 보일 것이다.

버스 클록과 데이터 버스의 폭은 사용되는 메모리에 따라 달라진다. 모든 버스들은 하나의 클록으로 동기화 되어 있다고 가정하고 클록 주기는 사용되는 메모리의 접근 시간의 역수로 가정한다. 이에 대한 좀더 효율적인 탐색 방법은 향후 연구과제 중의 하나이다.

5. 탐색 알고리즘

그림 7은 본 논문에서 제안하는 탐색 방법인 함수 Select_Archtcture를 의사 코드를 기술한 것이다. 이 함수는 3개의 입력이 필요한데, 각각은 탐색을 시작하기 위한 초기 통신 구조인 Initial_Arch, 시스템 명세의 스케줄을 나타내는 Sched, 사용되는 프로세서들의 메모

리 트레이스인 Mem_Trace이다. 3번째 행의 while 구문은 반복적으로 수행되는 한번의 탐색 과정을 의미한다.

```
1:  Select_Architecture (Initial_Arch, Sched, Mem_Trace)
2:    arch_list→Initialize(Initial_Arch)
3:    while (true) do
4:      Num_Qualified_1st = 0
5:      // 1st pruning step
6:      for each arch_i ∈ arch_list, i=1,2,...,N_1st do
7:        Bus_Protocol_Synthesis(arch_i)
8:        Do_Performance_Estimation(arch_i, Sched, Mem_Trace)
9:      end for
10:     Best_Exe_Time = Find_Best_Exe_Time(arch_list)
11:     for each arch_i ∈ arch_list , i=1,2,...,N_1st do
12:       if ((arch_i→Exe_Time - Best_Exe_Time)
13:             / Best_Exe_Time > ESTIMATION_ERROR) then
14:         arch_list→Delete(arch_i)
15:       else
16:         Num_Qualified_1st = Num_Qualified_1st + 1
17:       end if
18:     end for
19:     if (Num_Qualified_1st > MAX_ARCH) then
20:       for each arch_i ∈ arch_list , i=1,2,..., N_1st do
21:         if (i > Num_Qualified_1st) then
22:           arch_list→Delete(arch_i)
23:         end if
24:       end for
25:     end if
26:     // 2nd pruning step
27:     Prev_Seed_Arch = Curr_Seed_Arch
28:     for each arch_i ∈ arch_list , i=1,2,...,N_2nd do
29:       Do_Trace_Driven_Simulation(arch_i, Sched, Mem_Trace)
30:       if (Curr_Seed_Arch→Exe_Time < arch_i→Exe_Time) then
31:         Curr_Seed_Arch = arch_i
32:        end if
33:     end for
34:     // check the termination condition
35:     if ((Curr_Seed_Arch→Exe_Time <= Prev_Seed_Arch→Exe_Time) or
36:        (Curr_Seed_Arch→Num_Bus == Curr_Seed_Arch→Num_Pe)) then
37:        Quit_Exploration( );
38:      end if
39:     Generate_Architecture_Candidates(arch_list, Curr_Seed_Arch)
40:   end while
41: end Select_Architecture
```

그림 7. 제안하는 설계 공간 탐색 방법의 의사 코드 기술.

Select_Architecture는 크게 3부분으로 나뉘어 있다. 첫 번째 부분은 6번째 행에서 시작하는 첫 번째 단계인 설계 공간을 축소하는 과정이다. 초기에는 탐색해야 할 설계 공간은 Initial_Arch 하나뿐이다. 6~9행의 첫 번째 for 구문에서, 하나의 통신 구조에 대해 4.2장에서 설명된 다양한 우선순위 할당의 경우들에 대해 정적인 성능 예측 방법을 사용해 성능을

평가한다. 그 후 가장 좋은 성능을 가진 우선순위 할당의 경우를 선택한다.

모든 통신 구조들에 대해서 우선순위 할당이 끝난 후 각 통신 구조들을 성능에 따라 오름차순으로 정렬한다. 여기에서 가장 짧은 실행 시간인 Best_Exe_Time을 갖는 통신 구조가 선택된다. 본 논문에서 사용하는 정적인 성능 평가 방법은 10%정도의 예측 오차를 갖고 있으므로, 예측된 성능이 Best_Exe_Time과 10% 이내의 오차를 보이는 통신 구조의 경우 실제로는 Best_Exe_Time보다 좋은 성능을 나타낼 수도 있다. 이러한 오차 범위는 설계 공간을 줄이는 데 유용하게 사용될 수 있다. 그러므로 인자 ESTIMATION_ERROR의 값은 0.1로 놓는다. 11~18행의 for 구문에서 예측된 성능이 Best_Exe_Time과 ESTIMATION_ERROR보다 큰 오차를 보이는 통신 구조는 설계 공간에서 제외된다. 정확한 정적인 성능 예측 방법을 사용할수록 두 번째 단계에서 평가되어야 할 설계 공간의 크기는 줄어들게 된다.

첫 번째 단계에서 축소된 설계 공간의 크기가 여전히 클 경우에는 두 번째 단계에서 탐색되어야 할 통신 구조의 수를 제한한다. 그러므로 MAX_ARCH라는 상수를 정의하여 19~25행에 나타난 것과 같이 축소된 설계 공간에 남아 있는 통신 구조의 수가 MAX_ARCH를 넘지 않도록 한다.

탐색 과정의 두 번째 단계는 첫 번째 단계에서 선택된 통신 구조들에 대하여 트레이스 기반의 시뮬레이션을 적용하는 것이다. 2장에서 설명된 트레이스 기반의 시뮬레이터가 사용된다. 트레이스 기반의 시뮬레이션에 의해 예측된 통신 구조의 성능은 매우 정확하기 때문에 이를 바탕으로 통신 구조들의 성능을 비교하여 가장 좋은 성능을 갖는 통신 구조를 선택한다. 그리고 선택된 가장 우수한 통신 구조를 이전 탐색 과정에서 선택된 가장 좋은 통신 구조와 비교하여 성능의 개선 여부를 확인한다. 만약 현재 탐색 과정에서의 가장 우수한 통신 구조가 이전 탐색 과정의 그것에 비해 성능 향상이 없거나 사용되는 버스의 개수가 프로세서의 개수와 같아지면 탐색이 끝나게 된다.

39행에 기술된 함수 Select_Architecture의 마지막 부분은 두 번째 단계에서 선택된 가장 우수한 성능을 갖는 통신 구조로부터 4.1장에서 설명된 방법을 이용하여 다양한 통신 구조들을 생성해 내는 것이다. 각 통신 구조들의 성능을 예측하는 과정에서, 통신 구조의 파레토-최적화된 결과를 얻기 위하여 사용되는 버스의 개수에 따른 가장 좋은 성능을 갖는 통신 구조를 기록한다.

6. 실험

이 장에서는 다양한 실험을 통하여 제안한 탐색 방법의 효율성을 검증한다. 실험은 Xeon 2.8GHz CPU, 리눅스 기반의 컴퓨터에서 수행되었다. 첫 번째 실험은 버스 우선순위 할당을 위한 휴리스틱에 관한 것이다. 제안한 우선순위 할당 휴리스틱을 그림 5(d)의 DVR 시스템의 단일버스구조로부터 시작하는 설계 공간 탐색을 이용하여 가능한 모든 우선순위 할당의 경우를 전부 조사하는 방법과 비교하였다. 가능한 우선순위 할당 경우를 전부 조사하는

경우 실행 시간이 너무 오래 걸리기 때문에 버스의 개수가 4개 이하인 통신 구조들에 대해서만 실험을 진행하였다. 두 가지 방법에 대한 비교의 결과는 표 2에 나타나 있다.

표 2. 제안된 우선 순위 할당 방법의 모든 경우를 조사하는 방법과의 비교 및 그 효율성.

할당 방법	휴리스틱			모든 경우를 전부 조사		
	최소	최대	평균	최소	최대	평균
탐색 된 경우의 수	7	11	9	720	25920	3851
최적의 결과와의 성능차(%)	0	0.99	0.061	-	-	-
순위	0	57.08	5.54	-	-	-

첫 번째 항목 '탐색 된 경우의 수'는 주어진 하나의 통신 구조에 대해 우선순위 할당을 결정하기 위하여 평가해야 하는 우선순위 할당의 경우의 수를 나타낸다. 제안하는 휴리스틱은 평균적으로 모든 경우를 다 평가해야 하는 방법에 필요한 탐색 공간의 0.56% 정도만을 탐색한다. 두 번째와 세 번째 항목은 제안하는 휴리스틱이 어느 정도의 최적화된 우선순위를 할당하는지 보여준다. 두 번째 항목에서 나타나듯이 모든 경우의 수를 조사하여 할당된 우선순위를 갖는 통신 구조와의 성능차이는 미미하다. 세 번째 항목에서는 제안된 휴리스틱에 의해 결정된 우선순위가 최적의 결과에 어느 정도 접근했는가를 순위로써 표현하였다. 예를 들어 임의의 통신 구조에서 100개의 우선순위 할당법이 가능하다고 할 때, 휴리스틱에 의해 결정된 우선순위 할당에 의한 통신 구조의 성능이 전체 100개의 우선순위 할당에 의한 통신 구조의 성능들 중 7번째로 우수할 때, 휴리스틱에 의한 결과의 순위는 7이 된다. 즉 순위가 작을수록, 휴리스틱에 의한 우선순위 할당방법은 최적에 가까워지는 것이다.

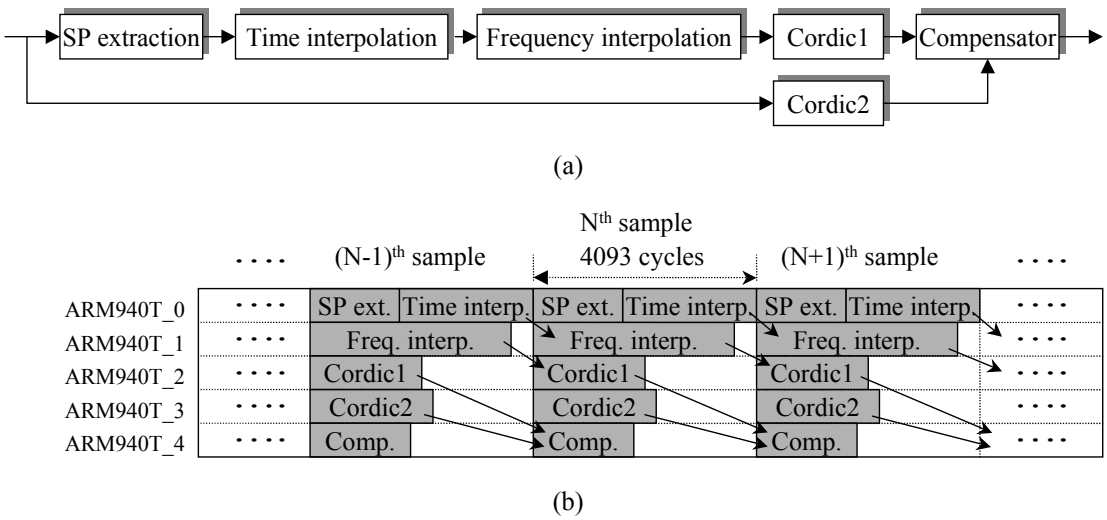
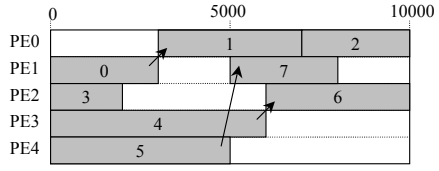


그림 8. (a) OFDM DVB-T 수신기를 위한 이퀄라이저의 명세와 (b) 스케줄.

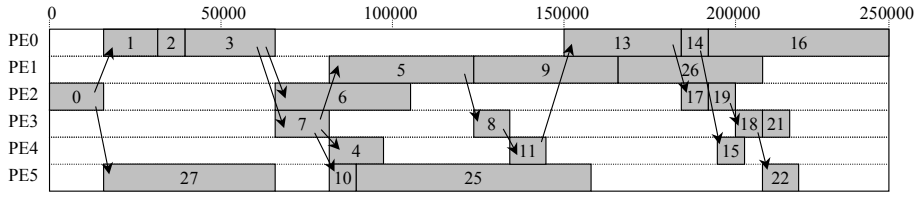
두 번째 실험에서는 제안한 설계 공간 탐색 방법을 2개의 실생활 예제인 DVR 시스템과 OFDM 방식의 디지털 비디오 방송(DVB-T) 수신기를 위한 이퀄라이저, 3개의 추가적인 가

상의 예제에 대하여 적용하였다. 이는 그림 8과 그림 9에 나타나 있다. DVB-T 수신기에서 이퀄라이저는 수신된 신호의 왜곡을 보상하기 위해 사용된다 [17]. 이퀄라이저의 모든 기능 블록은 5개의 ARM940T 프로세서에 할당되었고 그림 8(b)에 나타난 바와 같이 파이프라인 방식으로 스케줄 되어 병렬적으로 실행될 수 있다.

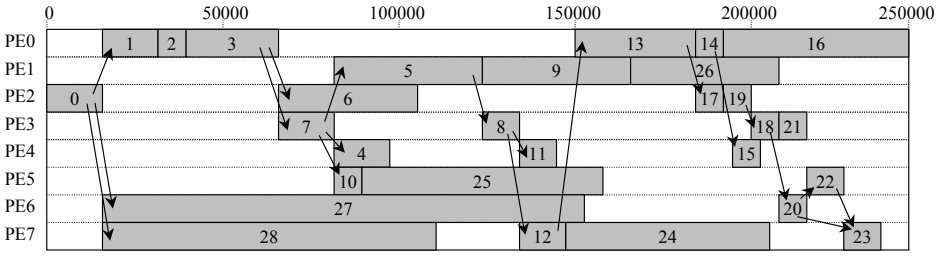
그림 9에 기술된 3개의 추가 예제에서 기능 블록들은 무작위로 생성된 토폴로지를 가지고 있고 정해진 개수의 프로세서들에 대해 임의로 할당되었다. 서로 다른 프로세서에 할당된 기능 블록들 사이의 연결선들은 공유 메모리 세그먼트를 의미한다. 한번의 탐색 과정 중 두 번째 단계에서 평가될 수 있는 최대 통신 구조의 개수를 나타내는 그림 7의 상수 MAX_ARCH의 값은 20으로 하였다. 표 3은 5개의 각 예제들에 대한 설계 공간 탐색의 결과를 나타낸다.



(a) System1



(b) System2



(c) System3

그림 9. 3개의 예제 시스템들.

표 3의 각 예제에서, 첫 번째 항목 '탐색 된 통신구조'는 전체 탐색 과정 동안 탐색 된 해당하는 버스의 개수를 갖고 있는 통신 구조의 수를 나타낸다. 통신 구조가 가질 수 있는 최대의 버스 개수는 사용되는 프로세서의 개수이다. 즉 모든 프로세서가 자신의 버스를 갖는 경우이다.

두 번째 항목 '누적성능 향상치'는 각 버스의 개수에서 가장 우수한 성능을 갖는 통신 구조들이 초기 입력이었던 단일버스구조와 비교하여 얼마나 성능이 향상되었는가를 나타낸다. 버스의 개수가 늘어나면서 탐색의 끝부분에 이르면 성능의 향상은 거의 이루어지지 않는 것

을 볼 수 있다. 대부분의 성능 향상은 탐색의 중간 부분에서 수렴한다. 최대로 향상된 통신 구조의 성능은 대략 초기 단일버스구조의 성능에 비해 2배 정도이다. 이러한 성능 향상은 단지 통신 구조의 변형과 메모리 할당에 의해서만 기인한다는 것을 생각해 볼 때, 제안하는 탐색 방법의 의의를 찾을 수 있다. 표 3의 밑에서 3번째 항목은 설계 공간 전체에서 탐색된 통신 구조의 개수를 나타낸다. 밑에서 두 번째 항목은 탐색 된 통신 구조들 중 탐색 과정의 첫 번째 단계에서 제거된 통신 구조의 비율을 나타낸 것이고 마지막 항목은 탐색이 끝날 때까지 걸린 수행 시간을 의미한다. DVR 예제의 경우 탐색 과정의 첫 번째 단계에서 대부분의 통신 구조가 제거된 것을 알 수 있다. 반면에 무작위로 생성된 예제(System1, System2, System3)들의 경우, 비교적 많은 통신 구조들이 탐색 과정의 첫 번째 단계를 통과하였다. 이는 탐색 된 통신 구조들의 성능 차이가 DVR 예제에 비해 그다지 크지 않았기 때문이다. 이러한 사실은 두 번째 단계에서 트레이스 기반의 시뮬레이션에 적용할 수 있는 통신 구조의 수를 제한해야 하는 이유가 된다. 표 3의 마지막 행의 수행시간은 트레이스 기반 시뮬레이션에 소요된 시간을 포함한 것이다.

표 3. 5개 예제의 통신 구조 설계 공간 탐색에 대한 결과

예제	*DVR*		*Equalizer* for DVB-T		*System1*		*System2*		*System3*	
공유 메모리 개수	14		4		3		14		19	
버스 개수	탐색 된 통신구조	누적성능 향상치	탐색 된 통신구조	누적성능 향상치	탐색 된 통신구조	누적성능 향상치	탐색 된 통신구조	누적성능 향상치	탐색 된 통신구조	누적성능 향상치
1	1	1	1	1	1	1	1	1	1	1
2	541	1.5511	27	1.5871	19	1.4443	501	1.3438	721	1.6022
3	541	1.6585	31	1.7664	21	1.5592	441	1.4485	845	1.8431
4	533	1.7651	34	1.7661	23	1.6387	377	1.4921	801	1.9861
5	529	1.7686	12	1.7671	6	1.6390	121	1.4974	737	2.1153
6	260	1.7689					48	1.4963	177	2.1180
7									460	2.1181
8									176	2.1145
탐색 된 통신구조의 총합	2405		105		70		1489		3196	
첫번째 단계에서 추려진 통신구조 비율(%)	98		80		77		54		30	
수행시간 (초)	12063		24		17		3239		36234	

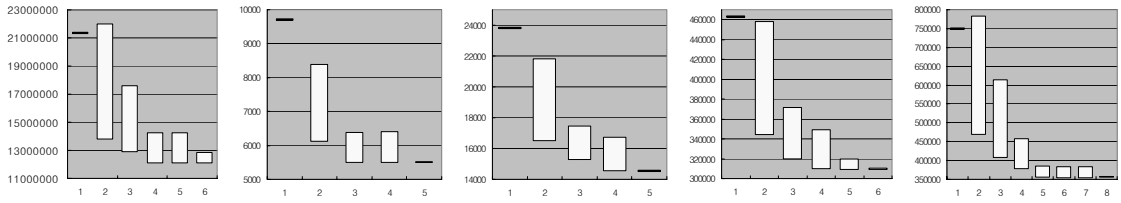


그림 10. 5개의 예제에서 사용되는 버스의 개수에 따른 성능의 변화폭. 각 그래프에서 가로축은 버스의 개수를, 세로축은 사이클 단위의 예측된 실행 시간을 나타낸다.

사용되는 버스의 개수에 따른 각 시스템의 성능의 변화량이 그림 10에 나타나 있다. 모든 예제에서, 이중버스구조가 가장 폭 넓은 성능의 변화를 보인다. 이는 잘못된 프로세서나 메모리 세그먼트의 버스에 대한 할당이 성능에 큰 영향을 미칠 수 있음을 보여준다. 예를 들어 DVR과 System3들의 경우, 이중버스구조에서의 최악의 성능은 단일버스구조의 성능보

다도 오히려 떨어진다. 그러나 사용되는 버스의 개수가 늘어날수록, 메모리 접근이 여러 버스에서 동시에 일어날 수 있는 병렬성의 증가로 인해 잘못된 프로세서나 메모리 할당에 의해 발생하는 성능의 감소치는 그다지 크지 않다. 이로부터 각 버스에서 가장 우수한 성능을 갖는 통신 구조를 선택한 파레토-최적화된 통신 구조의 집합을 만들어 낼 수 있다.

7. 결론

본 논문은 버스 기반의 온 칩 통신 구조와 메모리 할당을 최적화하기 위하여 반복적인 두 단계로 구성된 탐색 방법을 제안하였다. 반복되는 각각의 탐색 과정에서, 첫 번째 단계는 큐잉 모델을 기반으로 하는 효율적인 정적 성능 예측 방법을 사용하여 방대한 설계 공간을 빠르게 탐색하여 줄인다. 두 번째 단계에서는 첫 번째 단계로부터 축소된 설계 공간들의 통신 구조들에 트레이스 기반의 시뮬레이션을 적용하여 이들 중 가장 성능이 좋은 통신 구조를 찾아낸다. 실생활 예제인 DVR 시스템과 OFDM DVB-T 수신기를 위한 이퀄라이저, 3개의 가상 시스템을 통하여 본 논문에서 제시된 방법이 방대한 설계 공간을 탐색하는데 유용하게 쓰일 수 있음을 검증하였다.

본 논문에서는 통신 구조의 수행 시간에 관한 부분만 탐색의 대상으로 삼았지만, 제안된 방법은 전력 소모 등과 같은 다른 성능 요소에 대한 탐색에도 쉽게 확장될 수 있다. 또한 이는 현재 진행되고 있는 연구이다. 또 다른 향후 과제로서는 제안하는 탐색 방법의 대상을 온 칩 시스템이 아닌 오프 칩 시스템, 즉 보드 레벨로 확장하는 것이다.

8. 감사의 글

저자들은 본 논문이 개선될 수 있도록 도와 주신 심사 위원들의 조언에 깊이 감사 드린다. 본 연구는 국가 지정 연구실 프로그램(번호 M1-0104-00-0015), 두뇌 한국 21 프로젝트, IT-SoC 프로젝트에 의해 지원되었다. 또한 서울대학교 컴퓨터 연구소는 본 연구에 필요한 기자재들을 지원해 주었다.

참고문헌

[1] K. Keutzer, S. Malik, R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, “System-level design: Orthogonalization of concerns and platform-based design,” in *IEEE Trans. Computer-Aided Design*, Vol.19, pp. 1523-1543, Dec, 2000.

[2] IBM On-chip CoreConnect Bus Architecture. Available: http://www.chips.ibm.com/products/coreconnect/index.html

[3] ARM Advanced Micro Bus Architecture (AMBA). Available: http://www.arm.com/products/solutions/AMBAHomePage.html

[4] Sonics Integration Architectures, Sonics Inc. Available: http://www.sonicsinc.com

[5] K. Lahiri, A. Raghunathan, and S. Dey, “Efficient exploration of the SoC communication architecture design space,” in *Proc. Intl. Conf. Computer Aided Design*, pp. 424-430, Nov, 2000.

[6] T. van Meeuwen, A. Vandecappelle, A. van Zelst, and F. Catthoor, “System-level interconnect architecture exploration for custom memory organizations,” in *Proc. Intl. Symp. System Synthesis*, pp. 13-18, Oct, 2001.

[7] S. Kim, C. Im, and S. Ha, “Schedule-aware performance estimation of communication architecture for efficient design space exploration,” in *Proc. Intl. Conf. Hardware/Software Codesign and System Synthesis*, pp. 195-200, Oct. 2003.

[8] T. Yen and W. Wolf, “Communication synthesis for distributed embedded systems,” in *Proc. Intl. Conf. Computer Aided Design*, pp. 288-294, Nov, 1995.

[9] P. V. Knudsen and J. Madsen, “Communication estimation for hardware/software codesign,” in *Proc. Intl. Symp. Hardware/Software Codesign*, pp. 55-59, Dec, 1998.

[10] A. Nandi and R. Marculescu, “System-level power/ performance analysis for embedded systems design,” in *Proc. Intl. Conf. Design Automation*, pp. 599-604, Jun, 2001.

[11] P. Lieverse, P. van der Wolf, E. Deprettere, and K. Vissers, “A methodology for architecture exploration of heterogeneous signal processing systems,” in *Proc. IEEE Workshop Signal Processing Systems*, pp. 181-190, Oct, 1999.

[12] J. A. Rowson and A. Sangiovanni-Vincentelli, “Interface based design,” in *Proc. Intl. Conf. Design Automation*, pp. 178-183, Jun, 1997.

[13] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing system-on-chip communication architecture,” in *IEEE Trans. Computer-Aided Design*, Vol.20, pp. 768-783, Jun, 2001.

[14] J. Gong, D. D. Gajski, and S. Bakashi, “Model refinement for hardware-software codesign,” in *Proc. European Design and Test Conference*, pp. 270-274, Mar, 1996.

[15] M. Gasteier, M. Munch, and M. Glensner, “Generation of interconnect topologies for communication synthesis,” in *Proc. Intl. Conf. Design Automation and Test in Europe*, pp. 36-43, Feb, 1998.

[16] S. Meftali, F. Gharsalli, F. Rousseau, and A. A. Jerraya, “An optimal memory allocation for application-specific multiprocessor system-on-chip,” in *Proc. Intl. Symp. System Synthesis*, pp. 19-24, Oct, 2001.

[17] F. Frescrua, S. Pielmeier, G. Reali, G. Baruffa, and S. Cacopardi, “DSP based OFDM demodulator and equalizer for professional DVB-T receivers,” in *IEEE Trans. Broadcasting*, Vol.45, pp. 323-332, Sep, 1999.