[UCI]I804:24011-200000234659

2005년 8월
박사학위논문

재구성 가능 SoC 설계 공간의 효율적인 탐색을 위한 연구

조선대학교 대학원

전자계산학과

안 성 용

재구성 가능 SoC 설계 공간의 효율적인 탐색을 위한 연구

A Study on Efficient Design Space Exploration for a Configurable System on a Chip(CSoC)

2005년 8월 일

조선대학교 대학원

전자계산학과

안 성 용

재구성 가능 SoC 설계 공간의 효율적인 탐색을 위한 연구

지도교수 이 정 아

이 논문을 이학박사학위신청 논문으로 제출함.

2005년 6월 일

조선대학교 대학원

전자계산학과

안 성 용

안성용의 박사학위논문을 인준함

위원장 조선대학교 교수 _____ 인

위 원 광주과학원 조교수 _____ 인

위 원 조선대학교 부교수 _____ 인

위 원 조선대학교 조교수 _____ 인

위 원 조선대학교 교수 _____ 인

2004년 6월 일

조선대학교 대학원

목 차

그 림 목 차

표 목 차

ABSTRACT

A Study on Efficient Design Space Exploration for a Configurable System on a Chip(CSoC)

Ahn, Seong-Yong
Advisor : Prof. Lee, Jeong-A Ph.D.
Department of Computer Science,
Graduate School of Chosun University

Reconfigurable architecture is a hardware system which can adapt its hardware structure to process given application programs more efficiently while conventional architecture implies a fixed hardware structure. In this thesis, we select a CSoC(Configurable System on a Chip) composed of a conventional CPU and a FPGA among many alternatives as a reconfigurable platform for embedded systems. It is a hard task to find an efficient hardware configuration which satisfies design constraints for an application and becomes harder when a set of applications will be executed since we need to deal with the huge design space to explore trade-offs caused by adjustments of architecture and mapping strategy.

We adopted a design space exploration approach known as Y-chart approach for this challenging task as it allows to measure quantatively the performance for different mapping strategy from algorithms to homogeneous hardware configurations.

In this thesis, we extended the Y-chart approach for a CSoC by developing a DSE(design space exploration) tool for CSoC(Configurable System on a Chip) and obtained experimental data assuming H.263 application. The DSE tool, a retargetable simulator consists of three parts which are an application simulator, a hardware simulator and a mapping controller. We assume that the input description for the application simulator is based on the Kahn-Process-Network which is wildly adopted for a DSP modeling so that the semantic gap between the application model and architecture model is minimized.

The simulator was developed using the Trace-Driven simulation method which takes traces produced by the application simulator and allocates them to available hardware resources such as a CPU or a reconfigurable FPGA. Using this tool, a designer can vary design parameters and estimate performance numbers for each potential mapping as a software to be run in a CPU or a hardware configuration to be implemented, without building a prototype. The tool can provide useful design information including scheduling of tasks to a system designer who wants to know which partitioning cases, i.e., a hardware configuration satisfies the time and resource constraints in a timely and cost-effective way. We also

propose a heuristic algorithm improving the simulation time by reducing the mapping set on the basis of the relationship between workload and parallelism. Simulation results show that we can reduce the size of mapping set which poses difficulties on hardware-software partitioning. We expect the developed DSE tool based on the retargetable simulator will play an important role in an embedded system design.

Ⅰ. 서론

A. 연구 배경 및 목적

전통적으로 특정 응용 시스템을 구현하는데 크게 두 가지 방법이 사용되고 있다. 첫 번째는 하드웨어에서 특정연산을 수행하기 위해서 ASIC (Application Specific Integrated Circuit)과 같은 기술을 이용하는 것이다. ASIC은 주어진 특정 연산을 수행할 수 있도록 특별하게 제작되기 때문에 설계하고자 하는 연산에 맞는 수행을 할 경우에 아주 빠르고 효율적이다. 하지만 ASIC은 일단 제작이 끝난 후에는 변경될 수 없는 단점을 가지고 있다. 이러한 특징은 칩 제작 후에 회로의 어떤 부분이 변경되어야 할 경우 칩을 다시 설계하고 다시 제작하여야 하는 문제를 야기한다. 이러한 과정은 매우 많은 비용을 소모할 뿐 만 아니라 많은 시간을 필요로 하게 되고 보드레벨에서 다른 컴포넌트들과 같이 동작하는 경우 새롭게 보드를 설계해야 할 경우가 발생하기도 한다. 이러한 하드웨어적인 설계방법은 해당 어플리케이션이 자주 변경되고 재설계되고 보드 레벨에서 재배치가 필요한 경우에 유연하지 않은 구조라고 볼 수 있다.

두 번째는 소프트웨어적으로 프로그램되는 마이크로 프로세서를 이용하는 것으로 하드웨어적인 설계방법보다 훨씬 더 유연한 구조를 제공해준다. 일반적인 마이크로 프로세서들은 특정 어플리케이션을 수행하기 위해서 여러 명령어들을 수행시킨다. 이렇게 소프트웨어 명령어들을 바꾸어가며 수행할 수 있기 때문에 시스템에서는 여러 기능(functions)들을 하드웨어의 변경 없이 수행시킬 수 있게 된다. 하지만 이러한 소프트웨어적인 구조는 유연한 구조

를 가지는 반면에 성능 면에서는 그리 뛰어나지 못하다. 마이크로 프로세서들은 하나의 명령어를 수행시키기 위해서 명령어를 메모리로부터 읽고, 해독하고, 오퍼랜드를 읽어온 다음에 수행을 하기 때문에 칩 제작 전에 어떤 기능을 수행할 것인가가 이미 정해져 있는 ASIC보다 훨씬 느리게 된다.

재구성 가능한 컴퓨팅은 이러한 하드웨어와 소프트웨어의 장단점을 적절히 결합하여 소프트웨어로 처리하는 경우보다는 빠른 성능을 보이면서 전통적인 하드웨어적인 설계보다 한층 더 유연한 구조를 가능하게 한다[1]. FPGA(Field Programmable Gate Array)와 같은 재구성 가능한 장치들은 프로그램 가능한 구성 비트들(Configuration bits)들에 의해서 어떤 기능을 수행하게 될 것인가가 결정되는 계산 요소들(Computational elements)들의 배열(Array)로 구성되어 있다. 이러한 계산 요소들은 역시 프로그램 가능한 라우팅 자원들을 이용하여 연결되어 있기 때문에, 구현하고자하는 특정 디지털 회로는 이러한 계산 요소들과 라우팅 자원들을 이용하여 하드웨어에 재구성되어 구현될 수 있다.

재구성 가능한 컴퓨팅은 특히 계산양이 많은 암호와 알고리즘, 이미지 처리, 패턴 매칭, 데이터 압축과 같은 어플리케이션들에 적용되고 있으며 10~100배 가량의 성능향상을 보여주는 많은 연구 결과들이 제시되고 있다[2,3,4,5,6]. 최근에는 재구성 가능한 컴퓨팅 자원, 지역적인 메모리, 외부와의 인터페이스를 위한 회로 그리고 내장형 마이크로프로세서를 하나의 칩에 집적시킨 재구성 가능한 SoC(System on Chip)가 등장하여 어느 정도의 유연성을 제공하면서 성능을 더욱 향상시킬 수 있는 구조들이 등장하고 있다[7,8,9,10,11].

재구성 가능한 SoC를 위한 설계환경에서 중요한 문제들 중에 하나가 재구성 가능한 장치가 가지는 자원들을 효율적으로 이용하는 방법이다. 특히, 수행하고자 하는 어플리케이션이 요구하는 자원이 현재 사용 가능한 자원들보다

훨씬 많을 경우, 실행시간에 재구성 가능한 장치를 사용하여 시스템의 효율을 높이는 방법을 사용하여야 한다. 이때의 하드웨어 자원, 즉 FPGA와 같은 재구성 가능한 장치가 가지는 자원을 가상 자원(Virtual Resource)라고 하는데 이것은 일반적인 컴퓨팅 환경에서 가상메모리와 같은 개념이다. 이러한 가상 자원을 효율적으로 이용하기 위해서는, 어느 시점에 어떤 구성(Configuration)을 동적으로 재구성 할 것인가를 결정하는 스케줄링 알고리즘이 아주 중요하다. 재구성 가능한 SoC 시스템은 응용프로그램의 자원할당요구 분석을 정적(Static)으로 또는 동적(Dynamic)으로 처리할 수 있는데 시스템의 자원 활용도(Utilization)를 높이기 위해서는 동적인 분석이 선호되지만 재구성에 따른 추가적인 복잡도가 전체 시스템 수행시간의 약 30% ~40%를 차지한다고 알려져 있어, 이에 관련된 연구의 필요성이 제기되고 있다[12].

특히 이러한 자원 할당 요구 분석의 결과를 이용한 설계공간탐색(Design Space Exploration)에 관련된 연구는 진행되고 있지 않다. 설계 공간 탐색도구는 설계변수의 변화에 따라 설계하고자 하는 시스템이 어떤 성능을 보이는가를 미리 예측할 수 있는 도구이다. 이러한 도구를 재구성 가능한 SoC 설계방법에 적용함으로써 프로토타입을 제작하지 않고도 시스템의 설계변수들이 변화함에 따라 시스템 성능을 미리 예측할 수 있는 방법을 제공해 준다. 재구성 가능한 SoC을 위한 설계 공간 탐색에서 FPGA자원, 시스템 버스의 용량, 메모리 용량, FPGA 처리 속도, CPU의 처리 속도 등이 설계변수로 고려될 수 있을 것이다.

재구성 가능한 SoC를 위한 설계 공간 탐색을 위하여 기본적으로 필요한 요소들은 앞에서 언급한 바와 같이 설계변수 변화에 따른 성능 분석이 가능한 재구성 가능한(Retagetable) 시뮬레이터의 개발과 하드웨어/소프트웨어 분할

방법에 관한 연구라고 볼 수 있다. 하드웨어/소프트웨어 분할은 어플리케이션을 구성하고 있는 각각의 기능 요소들이 어떤 컴퓨팅 자원에서 수행하게 할 것인가를 결정해주는 것으로 전통적인 ASIC 기반의 하드웨어/소프트웨어 통합설계 영역에서도 NP-Hard 문제라고 알려져 있으며 전체적인 시스템 성능에도 많은 영향을 미친다. 그러므로 실행 중 동적으로 하드웨어 자원의 재구성이 가능한 시스템에서도 하드웨어/소프트웨어 분할 문제가 중요한 요소라고 볼 수 있다.

본 논문에서는 재구성한 가능한 컴퓨팅을 이용한 SoC에서 가상자원을 효율적으로 분할하기 위한 방법에 관한 연구와 이의 성능을 분석할 수 있는 설계 공간탐색환경에 관하여 고찰한다.

B. 연구 내용

재구성 가능한 SoC에 관한 연구의 궁극적인 목적은 현재의 컴퓨팅 환경보다 획기적으로 성능을 향상시킬 수 있는 새로운 구조를 제시하는 것이다. 일반적인 마이크로프로세서를 사용하는 경우 소프트웨어로 어플리케이션 개발을 용이하게 하는 다양한 도구들을 쉽게 활용할 수 있는 환경이 충분히 구축되어 있다. 하지만 하드웨어 개발환경은 아직까지는 소프트웨어 개발환경에 비해서 많은 부분에서 전문적인 지식을 필요로 하고 있다. 특히 설계자동화 도구를 사용할 경우에는 쉽게 회로를 설계할 수 있는 장점이 있는 반면, 자원을 효율적으로 이용하는 측면에서는 아직 미흡하다고 볼 수 있다. 재구성 가능한 시스템의 견지에서 보면 다양한 설계방법들이 개발되어져 오고는 있으나 설계 도구의 부족과 비효율성 때문에 아직까지 일반적으로 활용되는데 걸림돌이 되고 있다. 재구성 가능한 시스템을 편리하게 활용할 수 있도록 하여, 고성능 연산을 수행하는 적응형 응용시스템들을, 소프트웨어로 개발하는 것처럼, 쉽고 효율적으로 구현할 수 있는 환경을 구축하는 것이 재구성 가능한 SoC에 관련된 연구의 궁극적인 목표라 할 수 있다.

본 논문에서는 재구성 가능한 SoC를 이용하여 어플리케이션을 구현할 경우 발생할 수 있는 두 가지 주제에 관하여 연구한다. 첫 번째는 주어진 어플리케이션이 설계변수에 따라서 어떠한 성능을 보이는 가를 예측할 수 있는 재구성 가능한 시뮬레이터를 개발하는 것이다. CSoC는 여러 부품들이 하나의 칩에 포함되어 있는 형태이기 때문에 제한된 자원만을 사용해야 하는 단점을 가지고 있다. 일반적으로 고성능의 부품을 칩 안에 포함시킬 경우 그 비용은 큰 폭으로 상승하게 되고 다른 부품의 성능을 줄여야 하는 문제가 생

긴다. 따라서 적절한 비용으로 성능 및 전력소모 같은 설계 제약사항을 만족시키는 구조를 빠른 시간 안에 결정하는 것은 Time-to-Market에 적응하는데 중요한 문제가 된다.

특정 어플리케이션에 대하여 현재의 시스템 구성이 제약사항들에 만족하는 여부는 실제 시제품을 만들어 정확한 벤치마킹을 수행해 보기 전에는 판단하기 어렵다. 하지만 시제품을 제작하는 것은 많은 시간과 비용을 필요로 하기 때문에 정확한 값은 아니더라도 대략적인 근사치를 얻어 될 수 있는 분석도구를 개발하는 것은 설계 시간을 획기적으로 줄여줄 수 있는 대안이 될 수 있다. 특히 CSoC 안에 집적되게 될 부품들의 특성을 모델링하고 그 부품들이 취할 수 있는 범위를 설계변수로 하여 그 설계변수가 변화함에 따라 성능이 어떻게 변화하는지 파악하는 알 수 있는 설계 공간 탐색도구의 개발은 내장형 시스템 개발과정의 핵심 이라고 볼 수 있다.

본 논문에서는 CSoC를 모델링하는 방법론을 제시하고 특정 어플리케이션이 CSoC기반의 내장형 시스템에서 수행될 때 그 성능을 상위수준에서 분석할 수 있는 도구를 개발한다. 개발된 분석 도구는 설계변수의 변화에 따라 시스템의 성능이 어떻게 변화하는지 파악할 수 있도록 구현된다. 두 번째는 재구성 가능한 SoC를 기반으로 작성되는 어플리케이션에 대하여 재구성 가능한 컴퓨팅 자원과 일반적인 내장형 프로세서간의 하드웨어/소프트웨어 분할을 효율적으로 수행하는 알고리즘에 관하여 연구하는 것이다. 하드웨어/소프트웨어 분할의 문제는 꾸준히 연구가 진행되어 왔지만 아직까지 확실한 해결책을 제시해주는 연구결과는 나오지 않고 있다. 특히 일반적인 ASIC 기반의 하드웨어/소프트웨어 통합설계를 그 대상으로 삼고 있는 연구가 대부분이고 재구성 가능한 장치를 기반으로 하는 CSoC에 대한 분할 문제는 크게 다

루어지지 않고 있다.

CSoC를 대상으로 하는 하드웨어/소프트웨어 분할 문제는 일반적인 ASIC 기반의 시스템을 가정하는 경우보다 훨씬 더 많은 경우의 수를 다루어야 하기 때문에 한층 더 복잡한 문제를 해결해야 한다. 본 논문에서는 이러한 문제를 해결하기 위하여 기존의 접근방법과는 다른 최대병렬성과 작업량의 분석에 따른 경험적인 알고리즘을 제안하고 그 효율성을 검증한다.

C. 논문의 구성

본 논문의 구성은 다음과 같다. 1장 서론에 이어 2장은 CSoC의 개념과 기본적인 구조에 대하여 설명하며 아울러 설계 공간 탐색에 대하여 약술한다. 3장에서는 설계 공간 탐색을 위한 재구성 가능한 시뮬레이터의 구현에 관한 내용을 다루며, 구현된 재구성 가능한 시뮬레이터를 이용하여 다양한 구성을 적용했을 때의 성능 분석 결과를 보인다. 4장에서는 CSoC를 이용한 어플리케이션을 구현하는 과정에서 하드웨어/소프트웨어 분할을 위한 휴리스틱 알고리즘을 제안하고 이 알고리즘의 효용성에 대하여 기술한다. 마지막으로 5장에서 본 논문의 결론과 향후 연구방향을 제시한다.

Ⅱ. CSoC와 설계공간탐색

A. 재구성 가능 SoC(CSoC)

1. 재구성 가능 SoC의 개요

MPU(Micro Processing Unit)와 DSP(Digital Signal Processor)는 상대적으로 높은 유연성을 가지고 쉽게 프로그래밍할 수 있는 장점을 가지고 있으나 대용량 처리를 필요로 하는 어플리케이션 동작에 적합하지 않으며 ASIC(Application-Specific IC)은 빠른 속도를 보장해 주지만 하드웨어 전문가만이 구현할 수 있고 프로그래밍하기 어려운 단점을 가지고 있다. 전통적인 FPGA(Field Programmable Logic Array)는 ASIC보다는 낮은 비용으로 어느 정도 빠른 처리를 할 수 있지만 유연성이 적고 여전히 많은 비용을 요구한다. 반면 RPU(Reconfigurable Processing Unit)는 쉽게 프로그래밍이 가능하고 유연성이 높고 빠른 처리속도를 가지고 있어서 다양한 어플리케이션을 동작시켜야하는 어플리케이션 설계에 많이 이용된다. 최근에는 RPU로 재구성 가능한 FPGA가 널리 사용되고 있다[1].

Configurable SoC는 이러한 특성을 가진 재구성 가능한 장치와 일반적인 목적의 MPU를 하나의 칩 안에 포함시킨 하드웨어를 말하는 것으로 하드웨어/소프트웨어 인터페이스가 단일 칩 안에서 이루어지기 때문에 인터페이스를 위한 오버헤드를 획기적으로 줄일 수 있다. 특히 내장형 시스템 개발을 위한 하드웨어 보드 설계자들로 하여금 쉽게 프로토타입을 개발할 수 있는 기본 바탕을 마련해 준다. (그림 2.1)은 하드웨어로 구현될 각 부품들의 특성

을 보여준다.

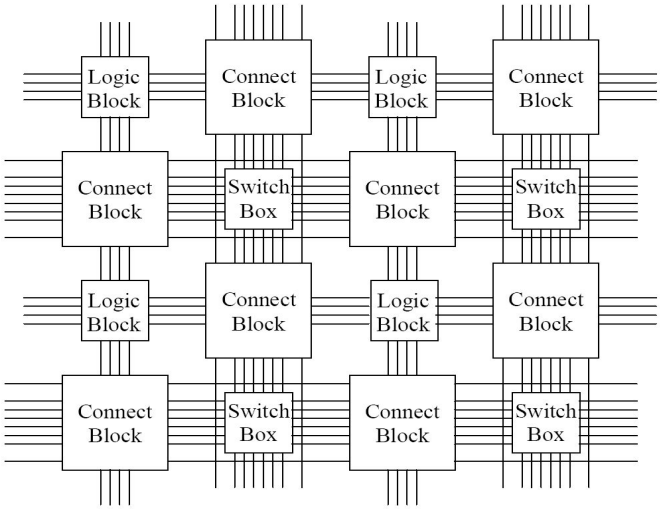
	MPU	DSP	RPU	FPGA	ASIC
Performance	Limited	Better than MPU	Better than DSP	Near ASIC	Very Fast
Architecture	OPEN	OPEN	OPEN	CLOSED	CLOSED
Programmable	Easily	Programmable	Programmable	Difficult	Not
Development Tools	Lots Low Cost	Good Low Cost	First Low Cost	High Cost	Very High Cost
Who Programs	Software Engineer	Software Engineer	Software Engineer	Hardware Engineer	Hardware Engineer
Use	General Purpose	Embedded	Embedded & Gen. Purpose	Embedded	Embedded

← PROGRAMMABILITY POWER →

(그림 2.1) 하드웨어 부품들의 특성

2. 재구성 가능 로직의 컴퓨팅 특징

재구성 가능한 구조는 로직 블록의 매트릭스와 이들을 연결하는 네트워크로 구성된 FPGA로부터 발전되어 왔다[13]. (그림 2.2)는 일반적인 섬 스타일(island-style) FPGA 라우팅 구조를 보여준다. 로직 블록들의 기능 구현과 연결 네트워크의 구성은 비트 스트림을 하드웨어에 다운로드 함으로써 이루어 질 수 있다. 위에서 언급한 바와 같이 현재는 FPGA와 다른 통신 채널 및 기타 하드웨어 자원을 하나의 칩으로 구성하는 기술이 계속적으로 발전해오고 있고 상업적으로 개발되어져 오고 있으며 이에 대한 많은 연구들이 진행되고 있다[14,15,16,17].



(그림 2.2) 일반적인 island-style FPGA 라우팅 구조

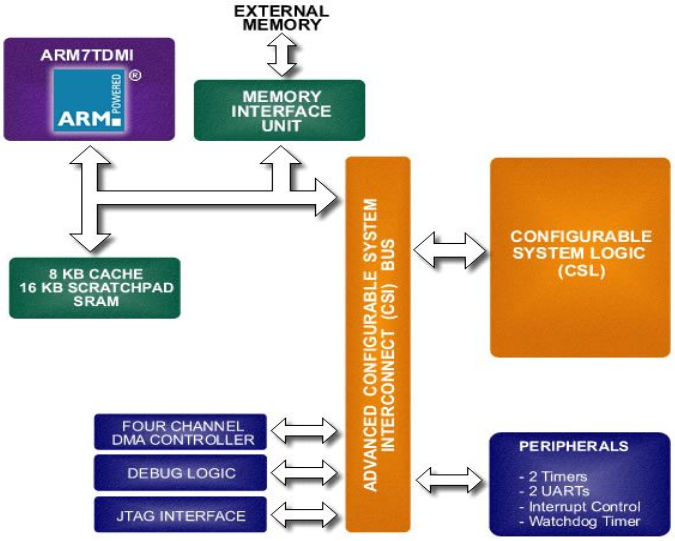
이와 같은 재구성 가능 로직은 기존의 전통적인 컴퓨팅 환경과 비교하여 다음과 같은 특성을 가지고 있다.

- 공간 컴퓨팅(Spatial Computing) : 공유 컴퓨팅 자원을 통한 시간적으로 순차적인 처리를 하는 것이 아니라 처리 되는 데이터는 공간적으로 분산되어 처리된다.
- 재구성 데이터 흐름(Configurable Datapath) : 각 기능 요소의 실제 기능과 통신 채널은 재구성 메카니즘을 이용하여 실행 중에 적용될 수 있다.
- 분산 제어(Distributed Control) : 각 기능 요소는 모든 기능 요소들에게 보내지는 특정 명령어에 의해 제어되는 것이 아니라 지역적인 구성(configuration)에 기반하여 데이터를 처리한다.
- 분산 자원(Distributed Resources) : 데이터 처리에 필요한 연산 장치나 메모리같은 요구 자원들은 분산되어져 있다.

3. 재구성 가능 SoC의 구조

재구성 가능한 컴퓨팅 플랫폼은 전형적으로 시스템 버스나 I/O 채널을 이용하여 호스트 시스템과 연결되어 동작하게 된다. 이러한 시스템은 특정 어플리케이션에 대하여 괄목할만한 속도 향상을 보여준다[18]. 이와 같은 경우에 다양한 어플리케이션에 대해서는 재구성 가능 로직과의 인터페이스가 성능 향상을 저해하는 요소가 된다. 재구성 가능 로직과의 인터페이스 과정에서의 지연(delay)은 주로 많은 데이터 전달과 재구성에 따른 오버헤드에서 비롯되게 된다. 이러한 문제를 해결하기 위하여 재구성 가능한 로직을 프로세서 다이(die)안에 포함시킨 시스템들이 설계되고 있다. 이러한 종류의 구조를 일반적으로 하이브리드 구조라고 하며 Configurable System-on-Chip(CSoC), Reconfigurable Systems-on-Chip(RSoC) 그리고 Systems on Programmable Chip(SoPC)등 여러 가지 용어들로 일컬어진다.

(그림 2.3)은 Triscend 사에서 개발한 CSoC의 일종인 A7S의 기본 구조를 보여준다[9]. (그림 2.3)에서 보여 진 바와 같이 Triscend A7S는 기본적으로 재구성 가능 장치인 CSL(Configurable System Logic)과 내장형 프로세서인 ARM7TDMI와 이것들 간의 인터페이스를 위한 CSI(Configurable Sysem Interconnect) 버스로 구성되어 있음을 알고 있다. CSI 버스는 ARM 프로세서와 Memory-mapped I/O를 사용하여 통신을 하게 되고 CSL과의 통신은 내부 레지스터를 이용하여 연결을 설정한다. 이외에 A7S는 메모리 관리를 위한 메모리 인터페이스 장치와 내부 캐쉬 메모리를 가지고 있다. 프로세서에서 동작하는 코드는 내부 캐쉬 메모리나 외부 메모리 또는 외부 플래쉬 메모리에 저장시켜 놓고 프로그램을 동작 시킬 수 있다. 나머지 장치들은 외부 기기와의 연결을 위한 것으로 DMA나 디버깅 등을 위한 것들이다.



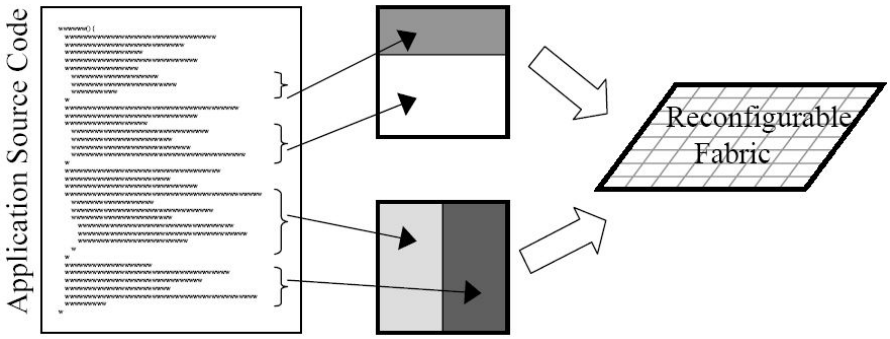
Block Diagram of the Triscend A7S Field Configurable System-on-Chip

(그림 2.3) Triscend A7S의 블록도

4. 실행 중 재구성을 위한 가상 하드웨어 개요

CSoC를 기반으로 하여 어플리케이션을 개발하는 경우에 프로그램의 일부분을 재구성 가능 장치를 이용하여 성능을 향상시킬 수 있다. 하지만 재구성 가능한 장치의 용량이 한정되어 있고 하드웨어로 처리해야 하는 부분이 상대적으로 많을 경우에 하드웨어로 처리해야 되는 기능을 모두 재구성 가능 장치에 할당할 수 없다. 이러한 경우 프로그램이 동작하는 도중에 실행해야 하는 기능이 필요할 때 현재의 재구성 가능 장치의 구성을 동적으로 교체할 수 있는 방법이 필요하게 된다. 이러한 방식의 동작을 실행 중 재구성(Run-Time Reconfiguration)이라고 한다[19]. (그림 2.4)는 재구성 가능한 장치가 어플리케이션 전체가 요구하는 기능을 수용하지 못할 때 하드웨어로 구성이 가능한 두개 이상의 구성으로 어플리케이션을 분할하는 모습을 보여준

다. 분할된 두개의 구성은 서로 다른 시간에 재구성 가능 장치에 구성되게 된다. 이는 재구성 가능 장치의 기본 특성인 공간(Spatial) 컴퓨팅과 시간(Temporal) 컴퓨팅을 결합한 형태라고 볼 수 있다.

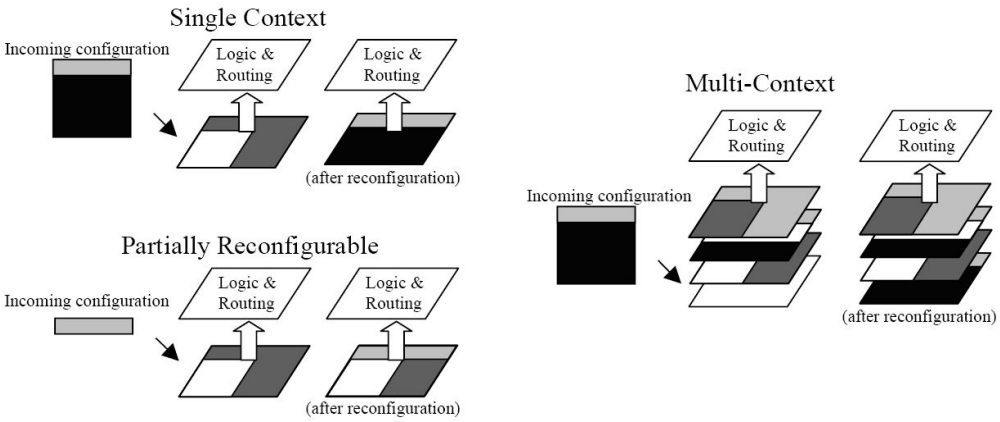


(그림 2.4) 실행 중 재구성을 위한 하드웨어 구성의 분할

실행 중 재구성의 기본개념은 가상 메모리(Virtual Memory)의 유사한 가상 하드웨어(Virtual Hardware)에 기반을 두고 있다. 실행 중 재구성은 특정 어플리케이션에 대하여 고정된 재구성 장치를 사용하는 것보다 더 많은 부분을 하드웨어로 매핑시킬 수 있기 때문에 프로그램의 더 많은 영역을 재구성 가능 시스템을 통하여 가속 시킬 수 있는 장점이 있다. 반면에 실행 중 재구성에 대한 오버헤드가 지나치게 커질 경우에 오히려 성능을 저하 시킬 수 도 있지만 일반적인 경우에 있어 전체적인 성능을 향상시킬 수 있는 가능성을 제공해 준다고 볼 수 있다.

전통적인 FPGA 구조는 어느 한 순간에 재구성 장치의 모든 구성을 한꺼번에 바꿔야 하는 단일문맥(Single-Context) 구조를 사용한다. 이러한 구조는 상대적으로 느리고 제한점이 많기 때문에 효율적이지 못하다. 때문에 현재는

단일 문맥 구조 보다는 다중문맥(Multi-Context) 구조나 부분적 재구성(Partially Reconfigurable) 구조를 많이 적용하려고 하고 있다. (그림 2.5)는 단일 문맥구조와 다중문맥 그리고 부분적 재구성 구조에 대하여 간략히 보여준다.



(그림 2.5) 재구성 가능 컴퓨팅 구조들

각각의 모델들에 대하여 간략히 설명하면 다음과 같다.

- 단일문맥(Single context) : 단일문맥 구조 FPGA는 구성 정보의 직렬(serial) 스트림을 이용하여 프로그램된다. 이 구조에서는 순차적인 접근만이 지원되기 때문에 FPGA의 어느 한 부분이라도 변경을 해야 하는 상황이 되면 FPGA를 완전히 재구성해야 한다. 이는 재구성을 처리하는 과정을 단순화 할수 있는 장점이 있지만, 전체 FPGA의 구성정보 중에 작은 부분만을 변경하고자 할 때 높은 오버헤드를 발생시키기 때문에 비효율적이다. 하지만 하드웨어/소프트웨어 분할이 이미 고정적으로 정해져 있는 시스템(none-run-time-reconfiguration)에서는 어플리케이션 구현이 상대적으로 간단하기 때문에 현재에도 많은 상용 FPGA들이 단일 문맥

구조를 사용하고 있다[20,21,22].

- 다중문맥(Multi-context) : (그림 2.5)에서 보여 지는 것과 같이 다중 문맥 구조 FPGA는 기본적으로 두개 이상의 독립적인 구성 정보 평판(plane)을 가지고 있다. 어플리케이션이 동작하는 동안 어느 한순간에 하나의 평판만이 활성화 되며, 현재 활성화된 평판에 요구하는 기능이 없을 경우 제어비트들을 사용하여 빠르게 활성화된 평판을 바꾸는 방법을 사용한다. 다시 말하면 다중 문맥 구조는 단일 문맥구조가 여러 개 겹쳐져 있는 형태라고 볼 수 있다. 실행 중 재구성은 평판 단위로 이루어지고, 재구성 과정을 다른 평판이 동작하고 있는 동안에 백그라운드로 처리할 수 있기 때문에 단일 문맥에 비하여 상대적으로 성능을 향상시킬 수 있는 가능성을 제공해준다. 따라서 다중 문맥 구조에서는 서로 연관성이 있는 기능 단위들을 하나의 평판으로 구성하는 그룹화 작업이 성능에 중요한 요소가 될 수 있다[8,23].
- 부분적 재구성(Partially Reconfigurable) : 부분적 재구성은 실행 중에 FPGA 일부분을 재구성할 수 있는 구조를 의미한다. 현재 개발되고 있는 구조 중 가장 진보된 형태로 복잡한 제어 로직이 필요하기 때문에 활용하기가 쉽지 않은 구조라고 볼 수 있다. 이 구조는 부분적으로 재구성이 수행되는 동안에도 다른 부분들은 수행을 계속 진행 할 수 있는 특성을 가지고 있다[24,25,26,27].

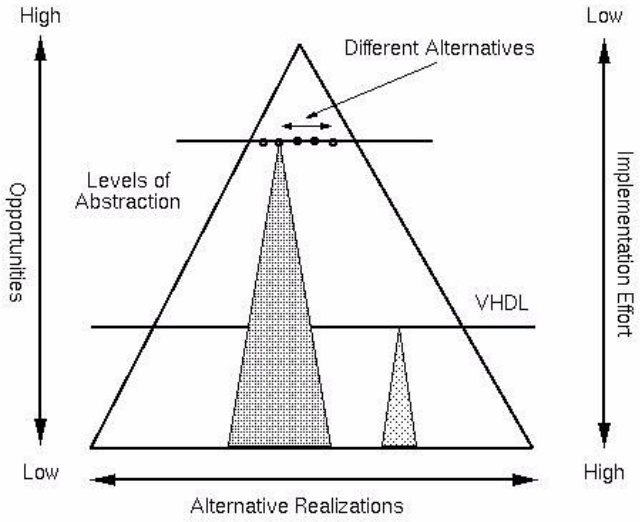
실행 중 재구성에 필요한 오버헤드를 줄이기 위하여 재구성 선 인출(Configuration Prefetching)[28,29], 구성정보압축(Configuration Compression)[30], 부분적 재구성(Partially Reconfiguration)을 위한 재배치(Relocation and

Defragmentation)[31], 재구성 정보 캐싱 등에 관한 많은 연구들이 진행되고 있다.

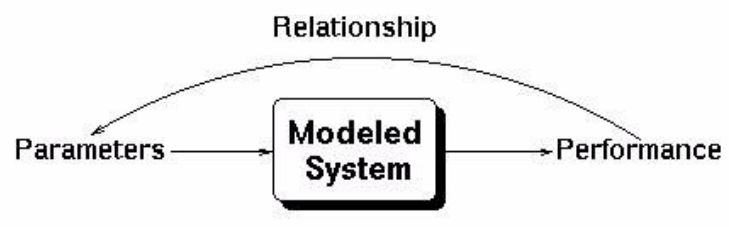
B. 설계공간탐색

1. 설계 공간 탐색의 정의

설계 공간 탐색은 가장 적합한 시스템 사양을 결정하기 위하여 설계변수들이 시스템 성능에 미치는 상관관계를 정량적으로 분석하고, 각 각의 응용프로그램에 따른 설계변수를 결정하는 것이다. 설계 공간 탐색은 설계변수가 증가함에 따라, 또 설계단계의 구체화가 추가됨에 따라, 다루어야 할 공간이 급격하게 증가한다. 시스템 제약 및 요구 조건을 만족시키는 시스템 구조의 집합은 (그림 2.6)에서 보인 것처럼 추상화(abstraction) 단계에 따라 삼각형으로 줄어든다. 이렇게 추상화 단계를 높임으로써 DSE에서 이용하는 도메인을 줄이지 않으면, 현실적인 DSE가 가능하지 않게 된다.



(그림 2.6) 추상화 단계에 따른 설계 공간

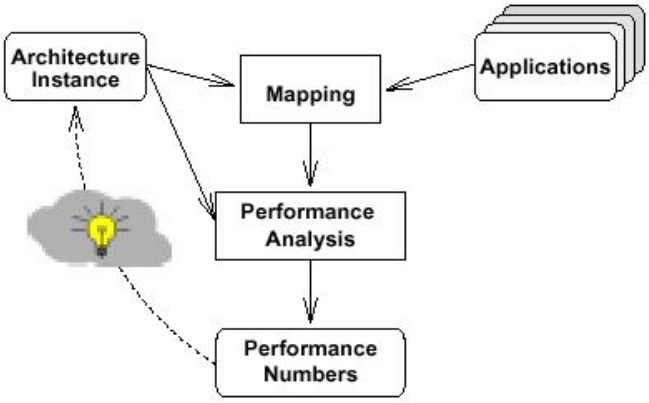


(그림 2.7) 시스템의 성능과 설계변수의 상관관계

(그림 2.7)에서 보인 것처럼 모델의 성능은 모델에 주어진 설계변수에 영향을 받는다. 설계변수가 다르게 적용될 때다 설계될 시스템은 다른 성능 수치를 보이게 된다. 따라서 고 수준 설계 공간 탐색은 각각의 설계변수들을 적용하였을 때 성능수치가 어떻게 변하는지를 분석함으로써 이루어질 수 있다.

2. 단일형 하드웨어 설계공간탐색을 위한 Y-Chart 방법

단일형 하드웨어 구조를 전제로 응용프로그램들을 하드웨어 구조에 맵핑하여 예상 성능 수치를 측정하고 분석하는 체계적인 설계공간탐색 방법론으로 개발된 것이 Y-chart 설계방법이다(그림 2.7). 이 설계방법의 효용성은 Y-Chart 설계환경의 제안자인 Delft 대학의 Bart Kienhuis와 Ed Deprettere 교수 연구팀의 TV 응용의 비디오 프로세싱에 관련된 시스템 수준의 시뮬레이션에 관련된 결과에서 잘 나타나 있다[33,34].



(그림 2.8) Y-chart방법

(그림 2.8)에서 보여 진 봐와 같이 Y-Chart방법에서는 다양한 어플리케이션이 특정 아키텍처에서 수행될 경우의 성능을 분석할 수 있는 구조로 되어 있다. 또한 아키텍처 인스턴스에 적용 가능한 설계변수를 변화시켜가며 각각의 성능을 비교 분석할 수 있는 구조로 되어있다.

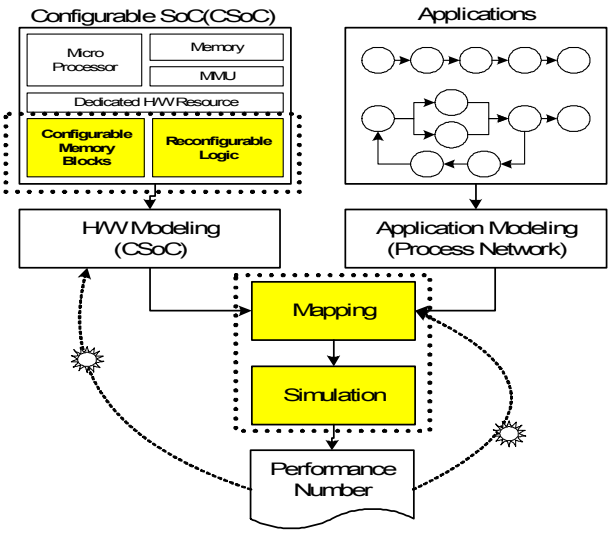
3. CSoC를 위한 설계 공간 탐색의 문제

CSoC를 위한 설계 공간 탐색을 하기 위해서 필요한 점을 들어보면 다음과 같다. 먼저 CSoC기반의 어플리케이션의 성능을 분석하기 위한 시뮬레이터를 개발하여야 한다. 두 번째로 하드웨어/소프트웨어 분할 방법에 대한 연구가 이루어 져야 한다. 이러한 설계 공간 하위수준에서 보다는 상위수준에서 이루어져야 한다. 하위수준에서의 시뮬레이션은 다양한 설계변수를 고려해야 하는 설계공간탐색 과정에서는 시뮬레이션 시간이 지나치게 길어지기 때문에 효율적이지 못하기 때문이다.

CSoC를 위한 시뮬레이터는 재구성 가능해야(Retagetable) 한다. 이는 하드

웨어 템플릿이 만들어지면 여기에 다양한 하드웨어 구성이 적용이 가능해야 한다는 의미이다. 다시 말하면 다양한 설계변수가 적용이 가능하고 추가적인 장치에 대한 모델링에 문제가 없어야 하며 모든 하드웨어/소프트웨어 분할의 경우에 대한 처리까지 원활해야 된다는 것이다. 재구성 가능 시뮬레이터를 구현하기 위해서는 먼저 CSoC에 대한 하드웨어 모델링이 선행되어야 한다. 또한 CSoC에서 동작하는 어플리케이션에 대한 모델링도 이루어져야 하며, 이들 간의 매핑 방법을 확립해야 한다. 이러한 매핑 방법에서 중요한 요소가 바로 하드웨어/소프트웨어 분할 방법이 될 것이다.

(그림 2.9)는 본 논문에서 고려하고 있는 설계 공간 탐색 환경의 전체 구조를 보여주고 있다. 점선으로 왼쪽 점선 부분이 재구성 가능한 하드웨어를 모델링 하는 부분이고 아래쪽 점선부분은 재구성 가능한 시뮬레이터를 이용한 시뮬레이션을 진행 하는 과정을 보여주는 것이다. 추가적으로 매핑 문제를 해결하는 과정에서 하드웨어/소프트웨어 분할이 고려되어질 것이다.



(그림 2.9) CSoC를 위한 설계공간 탐색 환경

Ⅲ. CSoC 설계공간 탐색을 위한 시뮬레이터

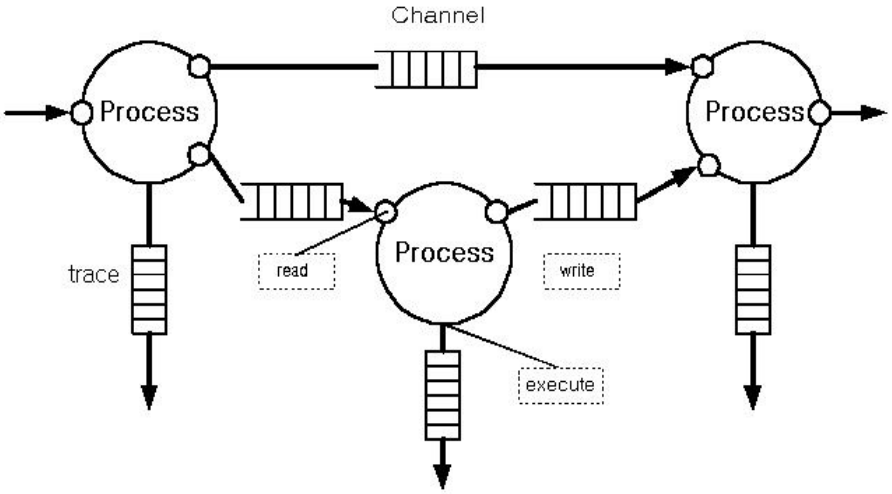
A. 시스템 모델링

본 논문에서 구현된 시뮬레이터는 CSoC를 기반으로 하는 어플리케이션의 설계과정에서 실제 프로토타입을 구축하지 않고 구현될 시스템의 성능을 예측할 수 있도록, 이질적인 신호처리 시스템을 위한 아키텍처 탐색도구인 SPADE를 응용하여 개발하였다[44]. 구현된 재구성 가능한 시뮬레이터는 주어진 어플리케이션을 시뮬레이션하는 부분과 하드웨어를 시뮬레이션하는 부분 그리고 어플리케이션과 하드웨어를 사상(Mapping)시켜주는 사상제어기로 구성되어있다. 어플리케이션과 하드웨어간의 의미론적인 차이를 최대한 줄이기 위하여 어플리케이션의 입력은 디지털 신호처리의 모델링 방법으로 널리 사용되는 Kahn 프로세스 네트워크로 전제하였다[35]. 또한 재구성 가능한 하드웨어인 FPGA 또한 데이터 흐름(data-flow)을 활용한 스트림 기반 하드웨어로 구성되는 것을 전제함으로써 어플리케이션과 하드웨어의 사상을 용이하게 하였다. 전체적으로, 구현된 시뮬레이터는 어플리케이션 시뮬레이터가 발생시키는 트레이스를 사상제어기가 해당되는 CPU 또는 스트림 기반의 하드웨어 자원에 할당하는 Trace- Driven 시뮬레이션 방법을 사용하여 구현되었다.

1. 어플리케이션 모델링

(그림 3.1)은 응용 시뮬레이터를 구현하기 위한 어플리케이션 모델의 예를 보여준다. 어플리케이션 모델은 기본적으로 패킷을 읽을 때는 블록되고 쓸 때

는 블록되지 않는(blocking-read, non-blocking-write) 특성을 가지는 프로세스들로 구성되는 Kahn 프로세스 네트워크를 기반으로 설계되었다. 각각의 프로세스는 입력포트와 출력포트를 가지며 입력포트를 통하여 패킷을 소모하고 출력포트를 통하여 패킷을 생산한다.



(그림 3.1) 어플리케이션 모델(Kahn Process Network)의 예

각각의 프로세스는 실행조건을 가지며 이 실행조건에 만족할 경우에만 프로세스가 활성화된다. 프로세스들 간의 통신은 각각의 프로세스들을 연결하고 있는 채널을 통하여 이루어진다. 프로세스의 실행 조건(F)은 아래와 같이 가능한 실행조건들(R)의 집합으로 정의된다. 가능한 실행조건은 다음과 같이 해당 프로세스의 입력포트들에서 요구하는 패킷 개수들의 집합으로 이루어진다. 아래의 경우는 가능한 실행조건들이 세 가지이고 입력포트가 두 개인 경우의 예이다.

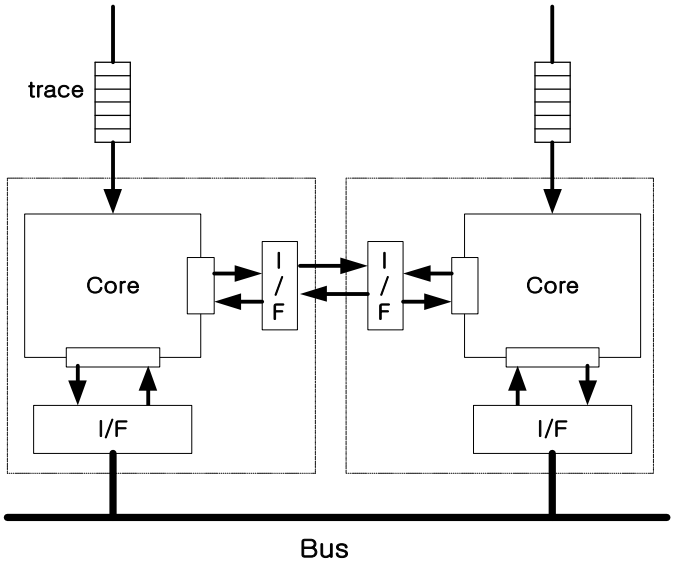
$F=\{R1, R2, R3\}$, $R1=\{2,3\}$, $R2=\{1,2\}$, $R3=\{2,0\}$

위의 예에서는 프로세스가 활성화되기 위해서 세 가지 조건들(*R1, R2, R3*) 중의 하나만 만족하면 되고, 각각의 조건들은 해당되는 입력포트에 조건 집합에 명시된 것보다 같거나 많은 수의 패킷이 존재해야만 된다. 하나의 프로세스가 활성화되면 일단 트레이스를 발생시켜 사상제어기에 보내고 실행이 완료되면 출력포트를 통하여 정해진 수만큼 패킷을 생산한다. 응용 시뮬레이터는 그 시작과 끝에 특별한 기능을 하는 SOURCE 프로세스에 SINK 프로세스를 가진다. SOURCE 프로세스는 단위 시간마다 패킷을 생산하기만 하고 소모하지는 않으며 SINK 프로세스는 처리가 끝난 패킷을 소멸시키고 패킷을 생산하지 않는다.

2. 하드웨어 모델링

(그림 3.2)는 하드웨어 시뮬레이터를 위한 하드웨어 모델의 예를 보여준다. 하드웨어 모델은 사상 제어기에서 할당된 패킷들을 실제로 실행하는 과정을 시뮬레이션 한다. 실제로 시뮬레이션 상에서는 수행시간만을 고려한다. (그림 3.2)에서 코어 부분은 특정기능을 수행하는 CPU나 FPGA를 의미하고 CPU는 한 순간에 하나의 기능만을 수행하는 반면 FPGA는 한순간에 여러 개의 기능의 수행이 가능하도록 여러 개의 기능 요소(Functional Element)를 가질 수 있다. 하드웨어 시뮬레이터는 통신상의 부하(Communication Overhead)를 고려하기 위해 버스구조를 갖는다. 하드웨어 구성요소들 간에는 버스 또는 독립적인 인터페이스를 통하여 통신하게 된다. 만약 현재 처리되고 있는 패킷이 이전에 다른 처리기에서 처리되었었다면 바로 해당되는 처리기를 점유하기

않고 버스구조로 들어가 버스가 요구되는 시간만큼 대기한 후에 처리되도록 구현되었다. 예를 들어 하나의 패킷이 CPU에서 처리되었었고 이 패킷이 다음 번에 하드웨어자원 중 FPGA를 요구하게 된다면 FPGA에 할당되지 않고 일단 버스에 할당되고 버스 요구 시간만큼 대기한 다음 FPGA로 할당이 이루어지게 된다. 버스구조는 시스템 설정변수에 따라 여러 개의 버스를 가질 수 있고 만약 모든 버스가 사용 중일 때 새로운 버스할당 요구가 있으면 요구하는 패킷은 버스구조내의 버퍼에 저장되게 되고 사용 가능한 버스가 있을 때까지 대기한다.

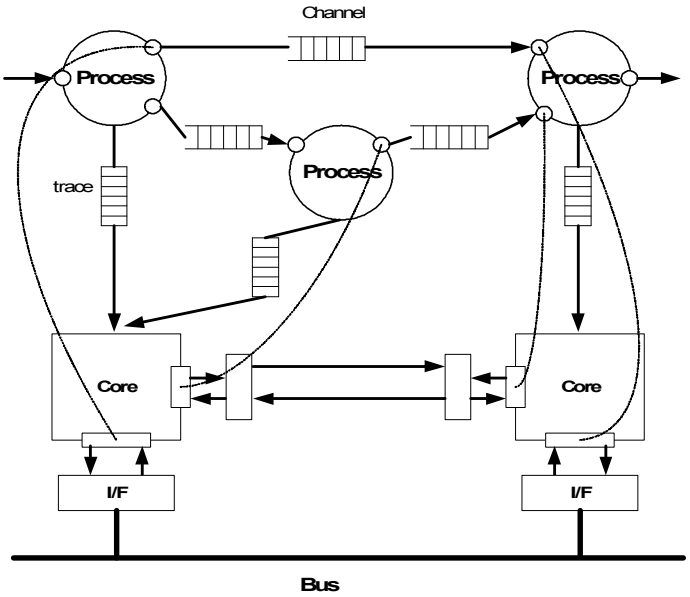


(그림 3.2) 하드웨어 모델

3. 사상 제어(Mapping) 모델링

사상 제어기는 시뮬레이션 수행 전에 하드웨어 요소와 각 하드웨어 요소가 수행할 수 있는 기능 요소에 대한 정보, 그리고 응용 프로그램의 하위 작업들

이 요청하게 될 처리 요구에 대한 정보를 이용하여 어떤 하위 작업을 어떤 하드웨어 요소의 기능 요소에 사상할 수 있는지를 판단한다.



(그림 3.3) 사상 제어 모델링

그림에서 보여 지는 봐와 같이 사상 제어는 기본적으로 N-to-1 사상에 기초하고 있다. 이는 여러 개의 프로세스가 하나의 하드웨어 자원에 사상된다는 것을 의미하며 하나의 프로세스는 2개 이상의 하드웨어 자원에 사상될 수 없음을 의미한다. N-to-1 사상은 현재 활성화에 대해서만 유효하고 다음번 활성화 될 때는 하드웨어 자원의 실행 상태를 고려하여 다른 하드웨어 자원으로 할당될 수도 있다.

실제 시뮬레이션 과정에서 연속적으로 데이터 단위들이 발생되고 처리되는 동안 하위 작업들이 같은 자원을 요구하는 자원의 충돌이 빈번하게 일어나게

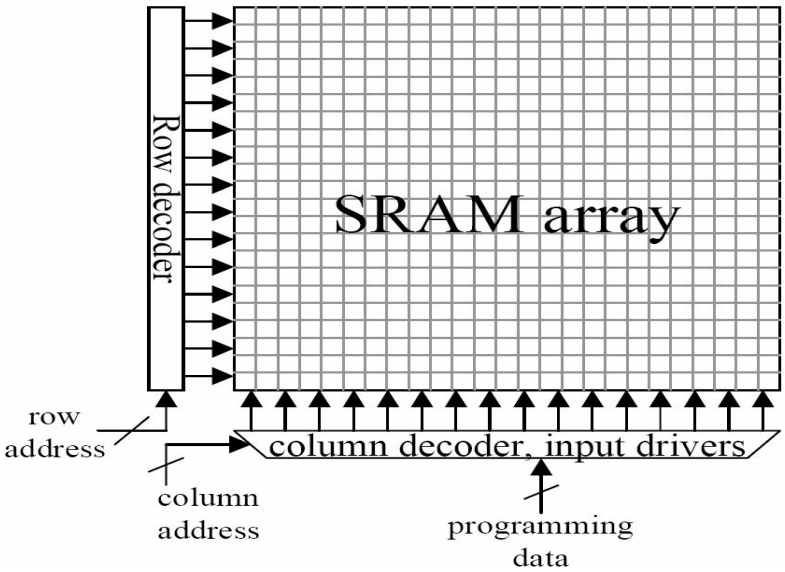
되는데 이는 사상 제어기에서 내부에서 작동하는 작업 스케줄러의 제어에 의해서 해결된다. 작업 스케줄러는 하위 작업의 자원요구가 있을 때 해당되는 자원이 현재 사용 중인가를 확인하고 사용 중이면 대기 버퍼에 그대로 남겨두고 요구하는 자원이 사용 가능한 하위 작업만 해당되는 자원에 할당하게 된다. 이러한 방법을 기반으로 사상 제어기에서는 간단한 스케줄링 알고리즘을 적용하여 하드웨어 자원 할당에 관한 시뮬레이션이 가능하도록 구현되었다.

4. 실행 중 재구성을 위한 모델링

본 논문을 통하여 구현된 시뮬레이터는 고정된 하드웨어 장치인 전통적인 FPGA 구조 뿐 만 아니라 실행 중 부분적 재구성 가능한 장치(Run-Time Partially Reconfigurable Device)에 대한 시뮬레이션을 진행할 수 있도록 설계 되었다. 실행 중 부분적 재구성 장치에 대한 모델링을 하기 위하여 본 논문에서는 먼저 FPGA의 구조가 (그림 3.4)과 같은 SRAM Array 기반의 사각형 구조로 구성되어 있다고 가정한다. (그림 3.4)에서 보여 지는 바와 같이 FPGA Cell들은 2차원 배열과 같은 형태로 이루어져 있으며 구성을 하기 위한 Cell들을 선택하기 위한 주소 매핑을 위한 행, 열 해석기(Row Decoder, Column Decoder) 등으로 이루어져 있다.

특정 기능이 FPGA에서 실행되기 위해서는 로직 합성(Synthesis) 단계를 거쳐 구현(Implementation)되고 FPGA Bit Stream이 생성된다. 이렇게 생성된 Bit Stream이 실제 장치에 다운로드되기 위해서는 배치 및 경로 설정(Place & Route)과정이 필요하게 되는데 이는 실제 FPGA 전체 영역 중 어느 위치에 매핑되고 통신 네트워크를 어떤 방법으로 연결시킬 것인지를 결정

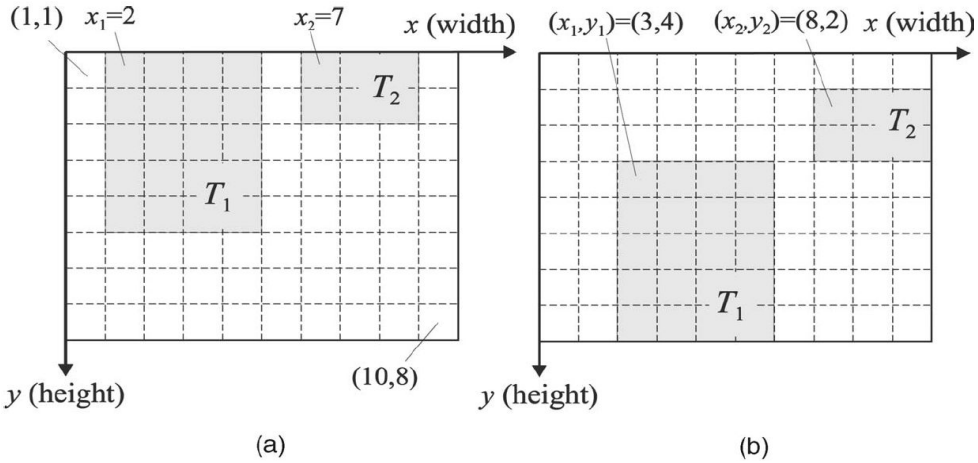
하는 과정이다. 이 과정들을 통하여 특정 기능을 수행하는 구성(Configuration)이 어떠한 방법으로 FPGA에 매핑 되는가를 결정하게 된다. 이때 그 구성이 어떤 모양을 가지고 있는가가 실행 중 부분적 재구성 가능한 장치를 제어하는 과정에서 매우 중요하다.



(그림 3.4) FPGA의 사각형 모델

일반적으로 특정 기능을 수행하는 구성(기능요소: Functional Element)의 모양은 불규칙적이다. 또한 합성 및 구현, 배치 및 경로 설정(Place&Route)을 담당하는 도구에 따라서 매우 다르게 나타난다. 이렇게 불규칙적으로 생성되는 구성들을 실행 중 부분적 재구성을 한다는 것은 매우 어려운 문제라고 할 수 있으며 현실적으로 완벽하게 제어하는 것과 FPGA 전체 영역의 100%를 활용하는 것은 불가능하다. 때문에 이를 간단화 하여 문제를 해결하려고 하는 많은 연구들이 진행되고 있다[36].

본 논문에서는 문제의 간단화를 위하여 특정 기능을 위한 재구성 정보의 모양이 사각형으로 이루어 있다고 가정한다. 전체적인 FPGA Cell들의 2차원 배열의 형태를 가지고 있고, 기능요소들이 궁극적으로 인접한 FPGA Cell들의 집합이라고 볼 수 있기 때문에 사각형 형태로 제어하는 것이 현실적으로 가능하게 된다. 물론 기능요소들이 정확한 사각형이 아니기 때문에 이에 따른 FPGA Cell들의 낭비를 가져온다. 하지만 불규칙적인 정확한 모양은 제어 로직의 지나친 오버헤드로 인하여 현실적이지 못하다.



(그림 3.5) 1D & 2D 재구성 모델: (a) 1D area model; (b) 2D area model

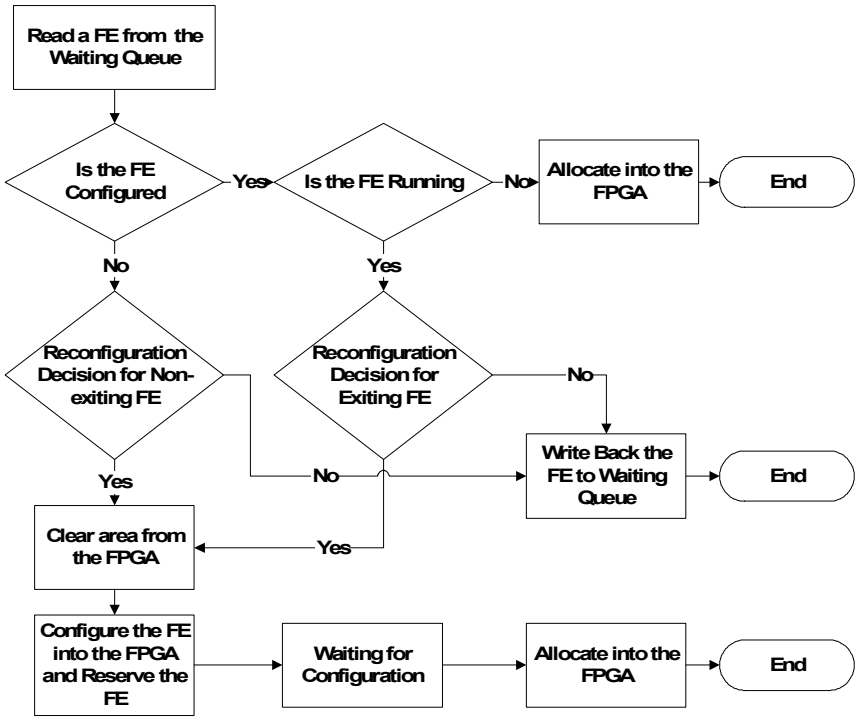
(그림 3.5)는 FPGA의 영역에 기능요소들이 구성되는 예를 보여준다. (그림 3.5)의 (a)는 1D 모델을 나타낸 것이고 (b)는 2D 모델의 예를 보여주는 것이다. 두 가지 모델 모두 사각형 모델을 사용한다는 것에서는 동일하나 기능요소가 매핑 되는 위치를 제어하는 방법에 있어서 차이가 있다. 먼저 1D 모델은 사각형의 형태를 가진 기능요소가 FPGA에 매핑될 때 사각형 영역의 시작 위치의 수직 또는 수평 주소를 고정시키는 방법을 사용한다.

(그림 3.5)에서의 예는 y 좌표를 고정시킨 예를 보여준다. 그림에서 기능요소 T_1 은 가로, 세로의 크기가 (4, 5) 이고 T_2 는 (3, 2) 이다. 전체 FPGA의 크기는 (10, 8) 이라고 가정한다. 1D 구조는 y 좌표가 고정되어 있기 때문에 기능요소의 매핑을 위하여 주소 정보를 x 좌표 값만 계산하면 되기 때문에 간단한 구조로 재구성을 수행할 수 있는 장점이 있다. 하지만 그림의 예에서 추가적으로 (5, 2)의 크기를 가지는 기능요소를 매핑하고자 할 때 여유 공간이 충분함에도 불구하고 매핑을 하지 못하는 불이익 감수해야만 한다. 반면 2D 구조에서는 어느 위치에든 기능요소를 매핑할 수 있기 때문에 공간 활용 측면에서 많은 이익을 얻을 수 있으나 제어 로직의 구현이 어렵다[41].

현재 실행 중 부분적 재구성을 위하여 1D구조는 실제 구현가능하고 활용되고 있으나, 2D구조는 구현은 가능하지만 실제 활용 면에서는 아직 연구 단계에 머무르고 있다. 하지만 현재 개발되고 있는 CSoC의 하드웨어구조가 2D구조를 지원하기 때문에 가까운 미래에는 2D구조가 일반화될 것이라 예상되며, 본 논문에서도 2D구조를 기반으로 실행 중 부분적 재구성 장치를 모델링 하였다.

(그림 3.6)은 실행 중 부분적 재구성을 위한 알고리즘에 대하여 간략하게 보여준다. 첫 번째 과정은 현재 실행해야 하는 기능요소(FE)를 대기 큐에서 읽어 오는 과정이다. 대기 큐에는 어플리케이션 모델에서 실행을 요청하는 기능요소들이 사상 제어기에 의해서 저장되고 관리되고 있다. 읽혀진 기능요소에 대하여 현재 그 기능이 FPGA에 구성되어있는지를 검사한다. 구성되어 있다면 그 기능요소가 실행중인지를 검사한다. 그 기능요소가 실행중이 아니라면 바로 실행될 수 있도록 FPGA 자원을 할당한다. 만약 실행중이라면 똑같은 기능을 두개 이상 구성할 것인지 아니면 한개만 유지할 것인지 판단하

는 과정을 거친다. 실제적으로 이 과정은 설계자가 상황에 따라 임의로 결정할 수 있는 부분이기도 하다. 만약 같은 기능을 수행하는 기능요소를 두개 이상 구성하지 않겠다면 기능 요소를 다시 대기큐에 되돌리게 된다.



(그림 3.6) 실행 중 부분적 재구성을 위한 흐름도

두개 이상의 같은 기능요소를 허용하는 경우에는 현재 FPGA 전체 공간 중에 이 기능요소를 구성할 수 있는 공간이 충분한지를 판단하여 충분하면 해당공간을 초기화 시키고 재구성하는 과정을 거치게 되고 그렇게 않다면 대기큐에 되돌린다. 다시 기능요소가 구성되어있는가 판단하는 부분으로 되돌아 와서, 만약 기능요소가 존재하지 않는다면 그 기능요소를 구성하기 위한 자원이 충분한지 검사하고 충분치 않으면 대기큐에 되돌리고 충분하면 해당

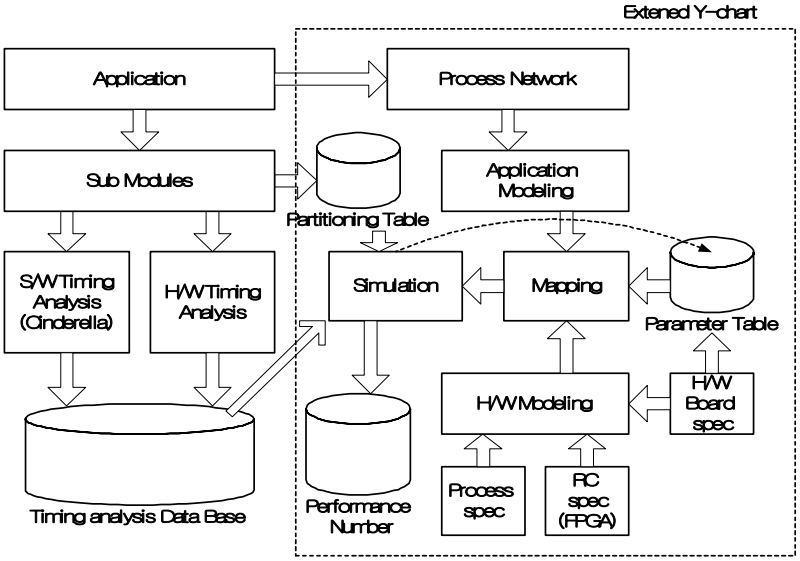
공간을 초기화하고 재구성하는 과정을 거치게 된다. 이러한 과정들은 대기큐에 있는 모든 기능요소들에 대하여 반복적으로 수행하게 된다.

현재 FPGA 공간 내부에서 사용할 수 있는 공간을 탐색하기 위해서 다양한 알고리즘을 적용할 수 있으며 이러한 알고리즘은 해결하기 어려운 문제로 다양한 연구들이 진행되고 있다[41]. 이는 본 논문의 주된 내용이 아니기 때문에 구현을 간단하게 하기 위해서 전체 공간을 검사하여 유휴공간을 찾아 First-Fit 방법을 사용하여 기능요소가 위치할 주소를 탐색하였는데 이 방법은 실제시스템에의 오버헤드가 가중 될 수도 있는 부분이다. 따라서 실행 중 부분적 재구성에 관한 부분은 타 연구결과와 결합되어 최적화 과정을 거쳐야 하는 부분이다.

5. 성능분석을 위한 모델링

(그림 3.7)는 CSoC 기반 어플리케이션의 성능 분석을 위한 구조를 나타낸 것이다. 전체적으로 어플리케이션의 하위 작업별 성능을 분석하는 부분과 확장된 Y-chart를 이용한 성능 분석을 위한 시뮬레이션 부분으로 나누어진다.

먼저 특정 어플리케이션 하위 작업별로 나누고 각각에 대해서 수행시간 분석을 수행한다. 소프트웨어 수행시간분석을 위해서는 “Cinderella”와 같은 소프트웨어 성능분석 도구를 활용할 수 있을 것이며[45], 하드웨어 수행시간 분석을 위해서는 Xilinx Foundation 시리즈와 같은 상용 도구를 활용할 수 있을 것이다[10]. 이러한 방법으로 분석된 각각의 분석결과는 데이터베이스 형태로 저장되게 되고 추후에 시뮬레이션을 위한 값으로 사용되게 된다.



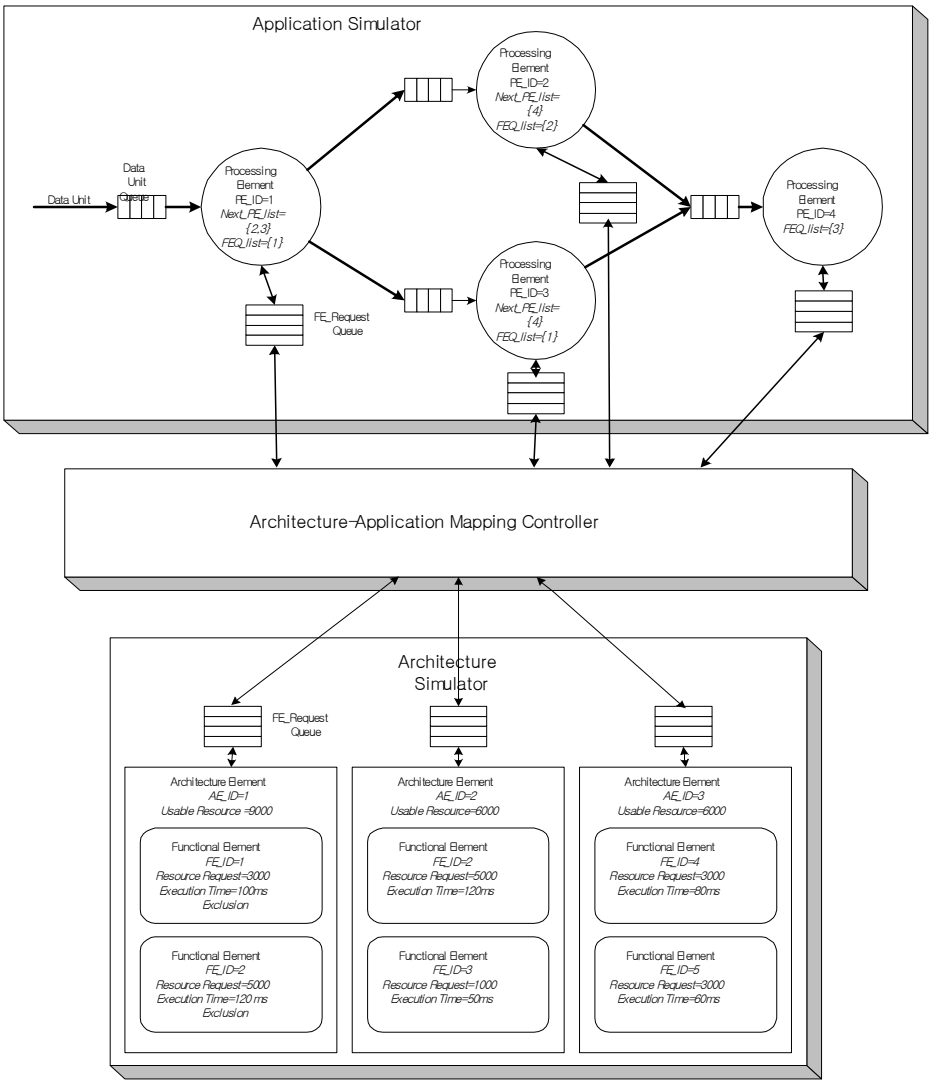
(그림 3.7) 성능분석을 위한 구조

하드웨어 모델은 사용되는 프로세스와 FPGA와 같은 재구성 가능한 장치들의 특성과 버스구조의 특성을 고려하여 모델링 된다. CSoC를 위한 기본적인 구조는 앞에서 설명한 바와 같이 목표시스템에 무관한 재구성 가능한(Retargetable) 시뮬레이터에 구현되어 있으며 사용자는 FPGA의 자원의 용적, 구현될 수 있는 기능요소, 각 기능요소들이 필요로 하는 자원의 용적만을 입력하면 된다. 이러한 방법으로 시뮬레이션 환경이 완성되면 하위 작업들이 어떤 하드웨어 자원에 사상이 되는가를 명시하는 분할 테이블(Partitioning Table)에 따라 시뮬레이션을 진행한다.

B. 재구성 가능 시뮬레이터의 구현

1. 재구성 가능 시뮬레이터의 구조

(그림 3.8)에서는 응용 시뮬레이터와 구조시뮬레이터의 구조를 포함한 전체적인 구조를 보여준다.



(그림 3.8) 재구성 가능한 시뮬레이터의 구조

(그림 3.8)의 예는 두 개의 FPGA와 하나의 CPU를 가지는 시스템을 예를 들어 보여준다. 응용 시뮬레이터의 각각의 프로세스는 자신의 프로세스 식별자(PE_ID)와 자신과 연결된 다음 프로세스 식별자들의 리스트(Next_PE_list)를 가지고 응용시뮬레이터의 흐름을 제어하고 요구되는 기능요소(FEQ_list)에 명시된 자원을 요구한다. 하드웨어 시뮬레이터는 하드웨어 자원마다 하드웨어 자원 식별자(AE_ID)를 가지고 있으며 하드웨어 자원은 자신이 사용할 수 있는 최대 크기(eg. FPGA cell의 개수)를 명시하며 기능요소들(Functional Elements)를 가진다. 각각의 기능요소들은 자신의 식별자(FE_ID)를 가지는데 이는 응용 시뮬레이터의 각각의 프로세스들의 FEQ_list에 있는 기능요소와 부합된다. 또한 각각의 기능요소들은 자신이 점유하는 크기(Resource Request)와 실행시간(Execution Time)을 명시한다.

응용 시뮬레이터는 데이터 단위의 흐름을 시뮬레이션하며 주어진 응용 프로그램 모델의 하위 작업들 간에 데이터 단위가 이동하도록 한다. 응용 시뮬레이터 내에서 하위 작업들은 프로세스(process)로 표현되며 이들은 전달된 데이터 단위에 대해 자신이 수행하는 기능 요구를 더하여 응용 시뮬레이터를 통해 프로세서-구성요소 사상 제어기로 데이터 단위를 전달한다. 하드웨어 구조 시뮬레이터에서 처리가 끝나서 프로세서-구성요소 사상 제어기를 통해 응용 시뮬레이터로 되돌아온 자료 단위는 처리를 요청한 프로세스에게 되돌려주고 이를 받은 프로세스는 다음 프로세스에게 이 데이터 단위를 넘겨준다.

응용 시뮬레이터는 프로세스가 처리할 자료 단위를 생성해서 처음 프로세스에 넣어주고 마지막 프로세스까지 거쳐서 나온 데이터 단위를 처리하는데 걸린 시간과 시스템 내에서의 대기 시간 등에 대한 자료를 모아 각 사상의 경우에 대한 시뮬레이션이 끝났을 때 성능 수치를 계산하는 근거로 사용한다.

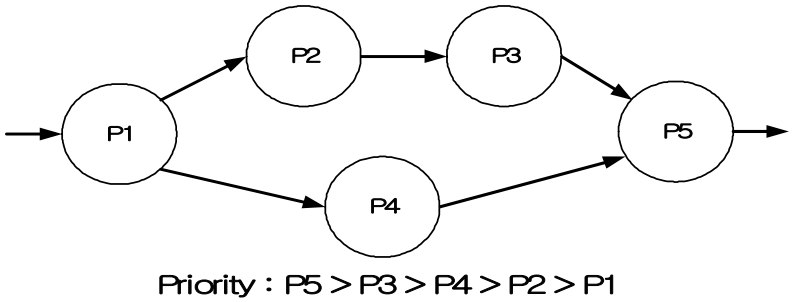
하드웨어 구조 시뮬레이터는 데이터 단위의 처리를 시뮬레이션하며 각 데이터 단위에 대하여 처리하는데 얼마의 시간이 걸렸는지를 계산한다. 프로세스에서 처리를 요구한 자료 단위가 사상 제어기를 통해서 어떤 하드웨어 구성 요소의 기능을 사용해야 하는지가 정해져서 전달되면 이를 받아서 기능 요소에서 처리하는 시간 동안 이 기능 요소는 다른 처리 요청을 받지 않도록 한다. 또한 기능 요소에 배제가 설정되어 있을 경우에는 해당 하드웨어 구성 요소가 처리하고 있는 처리 요구가 모두 끝난 다음에 하드웨어 구성 요소를 점유하도록 한다. 이렇게 하여 여러 하위 작업에서 동일한 하드웨어 구성 요소로 그리고 동일한 하드웨어 구성 요소의 동일한 기능 요소로 사상된 경우에 발생하는 하드웨어 자원 처리 요구 충돌까지 고려한 시뮬레이션을 수행하게 된다. 처리가 모두 끝난 뒤에는 얼마의 시간이 걸렸는지 자료 단위에 기록한 뒤에 처리 요청을 한 하위 작업에 결과를 돌려준다. 그리고 각 하드웨어 요소가 사용된 총 시간을 기록하여 각 사상의 경우에 대한 시뮬레이션이 끝났을 때 성능수치를 구하는 근거로 사용한다.

2. 하드웨어사상을 위한 스케줄링

시뮬레이션 하는 동안 매핑 제어기는 대기하고 있는 트레이스들을 할당 가능한 하드웨어 자원에 사상시키기 위하여 여러 가지 다양한 스케줄링 방법을 고려해 볼 수 있다. 내장형 시스템에 적용되는 스케줄링 방법은 일반적인 운영체제에서 사용하고 있는 여러 스케줄링 방법과는 달리 간단한 구조를 가지며 실시간 시스템에서 가장 중요하게 생각되는 시간 제약성을 만족하기 위하여 스케줄링 알고리즘이 간단해야 한다. 본 논문에서는 기본적으로 FCFS(First Come First Serve)스케줄링 방법을 적용하였다. FCFS 알고리즘은 구조가 간단하고

쉽게 구현될 수 있기 때문에 큐잉 시뮬레이션에서가 가장 보편적으로 이용되고 있다.

이는 간단하게 대기 큐에 먼저 들어온 트레이스를 하드웨어 자원 할당에 먼저 고려하는 것으로 요구되는 하드웨어 자원이 현재 실행중이면 대키큐에서 계속 대기하고 실행중이 아니면 할당하는 간단한 구조를 가진다. 이에 추가로 간단한 우선순위 스케줄링 방법을 구현하여 재구성 가능한 시뮬레이터의 매핑 제어기에 적용하였다. 구현된 우선순위 스케줄링 방법은 기본적으로 시스템 설계자가 우선순위를 직접 입력하거나 아래 그림과 같이 SINK 쪽에 가까운 프로세스에 높은 우선순위를 부여하게 되어있으며 우선순위는 대기 시간에 비례하여 높아지도록 하여 무한 대기로 인한 교착상태를 피할 수 있도록 하였다. (그림 3.9)는 특정 어플리케이션 모델에 우선순위를 부여한 예를 보여준다.



(그림 3.9) 우선순위 부여의 예

시뮬레이션 도중 응용 시뮬레이터는 발생되는 트레이스에 우선순위를 부여하여 매핑 제어기에 전달하고 매핑 제어기는 스케줄링 과정에서 같은 자원을 요구하는 트레이스가 두 개 이상 존재할 때 해당되는 트레이스들의 우선순위를 비교하여 우선순위가 높은 트레이스를 해당 하드웨어 자원에 할당하게 된

다. 우선순위가 같을 경우 대기 버퍼에 먼저 입력된 순서대로 할당된다. 할당 가능한 모든 트레이스에 대한 할당작업이 끝나면 대기 버퍼에 남아있는 트레이스들을 순회하면서 입력버퍼 안에서 대기한 시간에 비례하여 우선순위를 높이는 우선순위를 재조정 과정을 거친다.

3. 성능 측정 규준의 정의

성능수치로 사용되는 병렬성(Parallelism), 하드웨어자원들의 활용도(Utilization) 그리고 평균대기시간(ADT: Average Delay Time)은 (수식 1)에 의해서 구해진다. 병렬성은 현재의 시스템이 평균적으로 한 순간에 몇 개의 작업을 처리하고 있는가를 보여주는 성능 수치로 각각의 작업들이 대기시간을 제외하고 실제로 수행되었던 시간들의 총 합계를 시뮬레이션 종료시간으로 나누어서 구해진다. 활용도는 특정한 하드웨어 자원이 얼마나 사용되었는가를 보여주는 성능수치로 각각의 자원들이 사용된 시간을 시뮬레이션 종료시간으로 나누어 구해진다. U_{fpga}는 FPGA의 활용도를 나타내는 것으로 FPGA에 할당된 각각의 기능요소들의 면적을 수행시간으로 곱하여 합한 것을 전체 수행시간과 전체 FPGA면접을 곱한 값으로 나누어준 것이다. U_{cpu}는 CPU 활용도를 나타내며, CPU 수행시간을 전체 수행시간으로 나누어준 값이다.

수식은 [34]에서 성능수치로 사용된 것들을 본 논문에 맞게 재구성한 것이다. (수식 1)에서 T_{End}는 패킷처리 시뮬레이션 종료시간을 의미한다. CPU_{Time}은 CPU실행시간의 합을 의미하고 FE_i는 FPGA에 매핑된 특정 기능요소의 FPGA 수행시간을 의미한다. FS_i 는 특정 기능요소의 FPGA 요구 면적을 의미하며 FFS는 전체 FPGA의 면적을 의미한다. S_i는 패킷의 Sink시간을 나타내고 A_i는 패킷의 도착시간을 의미한다.

$$
\begin{aligned}
&Parallelism = (CPU_{Time} + \sum_{i=0}^{n} FE_i) / T_{End} \\
&U_{cpu} = CPU_{Time} / T_{End} \\
&U_{fpga} = (\sum_{i=0}^{n} (FE_i * FS_i)) / FFS * T_{End} \\
&ADT = 1/n(\sum_{i=0}^{n} (S_i - A_i)
\end{aligned}
\qquad \text{(수식 1)}
$$

4. 성능 수치 산출

구현된 설계 공간 탐색 도구를 이용하여 설계 공간 탐색을 위한 성능수치를 산출하는 과정은 다음 절차로 기술될 수 있다.

```
Initialize_Parameter_Table(Parameter_Table);
Build_Application_Model();
while(!End_of_Parameter_Table){
        Set_Next_Parameter(Parameter_Table);
        while(!End_of_Paremeter){
                Increase_Parameter();
                Build_Architecture_Model();
                while(!End_of_Partitioning_Case){
                        Set_Next_Partitioning_Case(Partitioning_Table);
                        Run_Simulator();
                        Store_Performance_Number();
                }
        }
}
```

(소스코드 1) DSE를 위한 과정

위의 코드의 각 행의 의미는 다음과 같다.

1. 파라메터 테이블을 초기화하여 시뮬레이션을 시작할 파라메터를 지정한다. 이 과정은 하드웨어 모델을 구성하기위한 전 처리 과정이라고 볼 수 있다.
2. 어플리케이션 명세 파일을 읽어 어플리케이션 모델을 만든다. 어플리케이션을 모델을 만들기 위하여 어플리케이션의 각 프로세스간의 연결을 확립한다.
3. 파라메터 테이블의 마지막 파라메터가 적용되어 시뮬레이션이 완료될 때까지 반복한다.
4. 파라메터 테이블에서 현재 시뮬레이션에 적용될 파라메터를 얻는다.
5. 설정된 파라메터의 범위가 모두 적용될 때까지 반복한다.
6. 현재 설정된 파라메터를 파라메터 테이블에 정의된 증가 값만큼 증가시킨다.
7. 설정된 파라메터들을 바탕으로 아키텍처 모델을 만든다.
8. 모든 분할의 경우가 적용될 때까지 반복하는 과정이다.
9. 시뮬레이션에 적용될 분할의 경우를 결정한다. 이 과정은 하나의 분할의 경우만을 시뮬레이션 할 경우에는 한번만을 수행하며 필요에 따라서는 상위 순환(loop)에 위치할 수 도 있다.
10. 만들어진 아키텍처모델, 어플리케이션 모델, 그리고 분할의 경우를 적용하여 시뮬레이션을 수행한다.
11. 산출된 성능 수치를 저장한다.

<표 3.1>은 파라메타 테이블의 예이다. 위의 표는 시스템 설계자가 선택적으

로 적용할 수 있는 설계변수들의 범위와 시뮬레이션과정에서 적용될 증가 값의 예를 보여준다. 예를 들어 설명하면 시스템 설계자는 구현하고자 하는 시스템의 버스 구조의 버스 지연시간(BUS TIME)을 1단위시간에서 100단위시간까지가 선택적으로 적용할 수 있는 설계변수의 범위가 되고 구현된 시뮬레이터는 이를 1 단위시간씩 증가시키면서 시뮬레이션을 진행하게 된다.

각각의 설계변수들의 의미는 다음과 같다. CPU의 Rate는 사용할 수 있는 CPU 중 속도가 가장 느린 것을 기준 CPU Rate 100으로 하고 CPU Rate이 증가하면 그만큼 CPU에서 처리하는 속도를 빠르게 한다. 예를 들어 CPU Rate이 150 이면 이 CPU는 기준 CPU보다 1.5배 빠르다는 의미이다. 따라서 어떤 기능요소가 기준 CPU에서 처리되는 시간이 100이라는 단위 시간이라면 CPU Rate이 150인 CPU에서는 처리시간이 약 67단위시간이 된다는 의미이다. FPGA Rate도 CPU Rate와 같은 의미로 이해 될 수 있다. FPGA Size는 활용 가능한 FPGA의 자원 량을 나타낸다. BUS TIME은 버스에 어떤 패킷이 할당되었을 때 버스안에서 대기하는 시간이고, 버스의 개수(# of Bus)는 버스구조내의 버스들의 개수를 의미한다. 버스에 관한 변수들은 하드웨어/소프트웨어 간의 통신이 빈번한 어플리케이션의 경우 중요한 설계변수가 될 수 있다. 스케줄링 방법(Scheduling Methods)는 하드웨어/소프트웨어 사상 제어기에서 선택할 수 있는 것으로 이 예에서는 FCFS와 우선순위(Priority)중 한 개를 선택하는 경우이다.

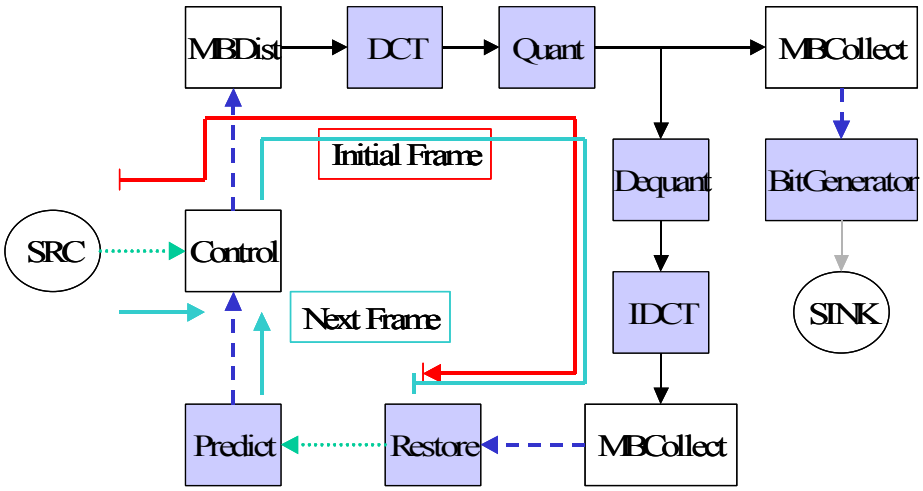
<표 3.1> 파라메타(설계변수) 테이블의 예

설계변수	범위	증가값
CPU Rate	{100 ... 300}	50
FPGA Rate	{100 ... 300}	50
FPGA Size	{3000 ... 20000}	1000
Bus Time	{1 ... 100}	1
# of Bus	{1 ... 10}	1
Scheduling Methods	{FCFS, Priority}	1

C. 하드웨어 구성 변경에 따른 시뮬레이션 예제

1. 시뮬레이션 환경

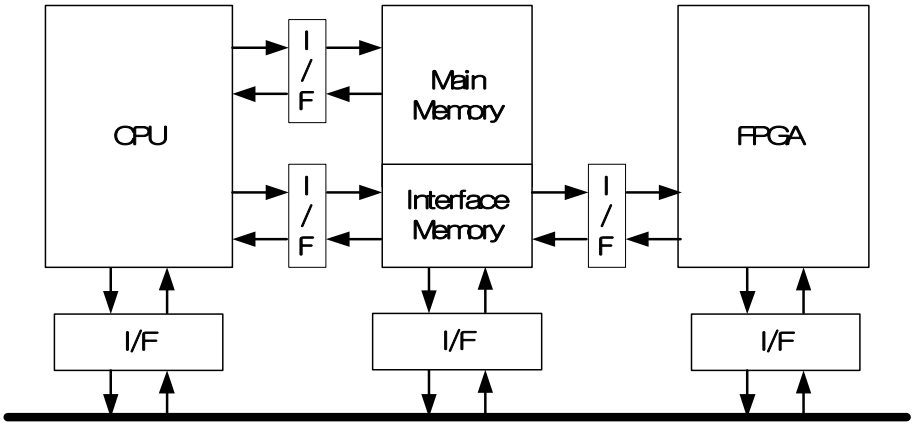
앞에서 설명된 수행 시간 분석을 위한 환경을 이용한 시뮬레이션을 수행하기 위하여 어플리케이션모델로는 H.263 Encoder을 참조하여 작성된 모델을 선택하였고, 하나의 마이크로 프로세서와 하나의 재구성가능한 장치 결합된 가상의 CSoC를 하드웨어 모델로 가정하였다. (그림 3.10)은 실험에 사용된 어플리케이션 모델을 Kahn Process Network 형태로 표현한 것이다.



(그림 3.10) H.263 Encoder의 Kahn Process Network 모델

(그림 3.11)은 시뮬레이션에서 가정되는 하드웨어 모델을 보여준다. 그림 8에서 보여 지는 하드웨어 모델은 CPU, FPGA, 메모리 장치 그리고 이들 간의 데이터 및 신호 교환을 위한 인터페이스 장치들로 이루어진다. 메모리 장치는 CPU에서 사용되는 Main Memory와 FPGA와 CPU간의 데이터 교환을

위한 Interface Memory장치들로 이루어진다고 가정한다. CPU에서 처리된 데이터가 바로 다음에 FPGA에 의해서 처리되는 경우에 이 Interface Memory에 저장되어지게 되고 FPGA에서는 이를 통하여 데이터를 읽어 처리하게 된다. 반대의 경우도 똑같은 과정을 거쳐 처리된다고 가정한다.



(그림 3.11) 실험에 적용된 하드웨어 모델

<표 3.2>는 (그림 3.7)에서 표현된 어플리케이션 모델의 각각의 프로세스(PE_ID)에 상응하는 기능요소(FE_ID)와 기능요소들의 소프트웨어 수행시간 하드웨어 수행시간 그리고 FPGA에 구현되었을 때 요구되어지는 자원 요구량을 보여준다. 표에서 제시되는 수치는 임의로 선택된 값들이다.

<표 3.2> 기능요소들의 수행시간과 FPGA자원 요구량

PE_ID	FE_ID	소프트웨어 수행시간(ns)	하드웨어 수행시간(ns)	FPGA 자원요구량(cell)
0	0	600	150	2000
1	1	1100	360	2500
2	2	5800	1250	2300
3	3	10300	2550	2200
4	4	8000	1790	2300
5	5	7600	1670	2000
6	6	1400	450	2300
7	7	5600	1100	2200
8	8	2440	840	2300
9	6	1400	450	2300
10	9	1500	350	2000

2. 모든 하드웨어/소프트웨어 분할의 경우에 대한 시뮬레이션

<표 3.3>은 (그림 3.10)의 어플리케이션 모델이 (그림 3.11)의 하드웨어 모델에서 동작되었을 때 가능한 모든 하드웨어 소프트웨어 분할 경우에 대한 시뮬레이션을 진행하여 얻어진 결과 중 우수한 성능을 보여주는 경우를 선택하여 보여준다. 아래 표에서 분할 경우에 보여 지는 숫자는 <표 3.2>에서 보여 지는 기능요소들을 나타내고, “F”는 해당 기능요소가 FPGA로 사상되었음을 의미하고 “C”는 CPU에서 처리되었음을 의미한다. 시뮬레이션을 위한 가정은 다음과 같다.

a. 처리되는 데이터 단위의 개수 : 100개

b. 데이터 단위의 발생간격 : 5000ns

c. 최대 활용 가능한 FPGA 자원 : 15000cell

d. 하드웨어 소프트웨어 인터페이스를 위한 버스 점유시간 : 50ns

e. 사상제어기에서 하드웨어 매핑을 위한 스케줄링 방법 : FCFS

<표 3.3> 하드웨어/소프트웨어 분할의 경우에 대한 성능분석 결과

분할 경우(FE_ID)										종료시간 (ns)	CPU 활용도	FPGA 활용도	병렬성
0	1	2	3	4	5	6	7	8	9				
F	F	C	F	C	F	F	C	C	F	255200	0.969612	0.294743	3.030094
C	C	F	F	F	F	F	F	C	C	568100	0.892449	0.200599	2.260077
C	C	F	F	F	F	C	F	F	C	616910	0.875776	0.196687	2.215056
F	C	F	F	F	F	C	F	C	F	633120	0.970598	0.195499	2.313906
C	C	F	F	F	F	C	F	C	F	698805	0.921981	0.160357	2.018589
C	F	F	F	F	F	C	F	C	C	740005	0.903933	0.157960	1.977662
F	C	F	F	F	F	C	F	C	C	790635	0.947726	0.152412	1.991880
C	C	F	F	F	F	C	F	C	C	856915	0.878144	0.127770	1.750897
C	C	F	F	F	F	F	C	F	C	885390	0.939863	0.122556	1.772541
F	C	C	F	F	F	F	F	C	F	899860	0.989332	0.116367	1.789401
C	C	C	F	F	F	F	F	F	C	916800	0.951298	0.123221	1.791378
C	C	F	F	F	F	F	C	C	F	935180	0.921753	0.101962	1.617806
C	C	C	F	F	F	F	F	C	F	968665	0.971595	0.098452	1.647484
F	C	C	F	F	F	C	F	F	F	983735	0.996706	0.120030	1.821024

<표 3.3>에서 결과를 보면 하드웨어로 매핑된 기능요소들의 개수가 5개에서 7개정도로 비슷한 것으로 나타나 있다. 각 기능요소들이 차지하는 FPGA 공간에 따라 약간의 차이는 있겠지만 이는 각각의 사상의 경우가 거의 비슷한 FPGA 공간을 차지한다고 볼 수 있다. 비슷한 FPGA 공간을 차지하고 있음에도 불구하고 종료시간은 최대 4배정도의 차이를 보이고 있다. 이러한 결

과는 FPGA 공간 활용도나 병렬성을 나타내는 성능 수치에도 잘 나타나 있다. 위 실험결과는 같은 자원을 사용하고 있더라도 하드웨어/소프트웨어 분할에 따라 시스템의 성능은 크게 차이가 남을 알 수 있다. 따라서 하드웨어/소프트웨어 분할에 대한 최적화가 시스템 성능을 향상시키기 위하여 매우 중요한 요소라는 것을 알 수 있다.

3. 설계변수의 변화에 따른 시뮬레이션

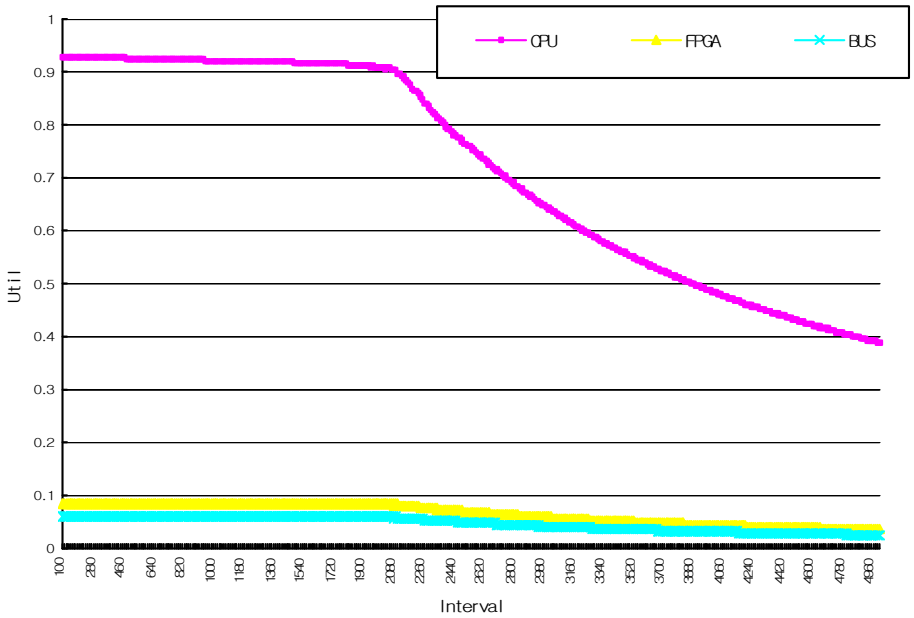
<표 3.4>은 실험에 적용 가능한 설계변수들의 범위와 기본값을 보여준다. 실험 그래프들에서 보여 지는 범위는 의미 없는 값을 삭제하거나 정규화된 것으로 표에서 제시된 것과 차이가 있을 수 있다. 설험에서 변화가 일어나는 설계변수를 제외한 나머지 설계 변수 값들은 특별한 언급이 없는 한 표에 제시된 기본값으로 고정되어 있다.

<표 3.4> 실험에 사용된 설계 변수

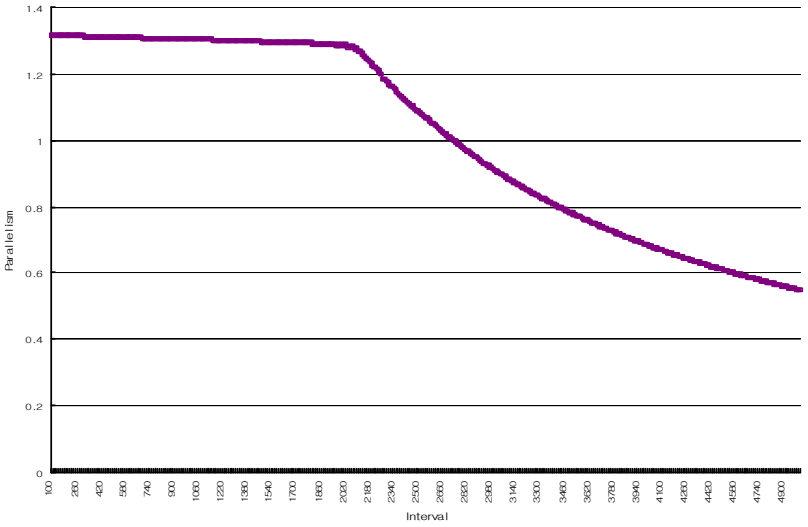
설계변수	범위	기본값
CPU Rate	{100 ... 300}	200
FPGA Rate	{100 ... 300}	200
FPGA Size	{10000 ... 10000}	10000
Bus Time	{100 ... 300}	200
# of Bus	{1 ... 1}	1
Scheduling Methods	{FCFS, Priority}	1

(그림 3.11)와 (그림 3.12)은 패킷이 입력되는 간격에 따른 각 하드웨어 자

원들의 활용도와 시스템의 병렬성을 보여준다. 그림들에서 보여 지는 바와 같이 입력되는 패킷의 간격이 어떤 특정 간격이 되면 활용도와 병렬성이 현격하게 저하되는 것을 볼 수 있다. 이것은 성능수치들이 저하되기 시작하는 지점 이상의 단위시간간격(2000)으로 패킷들이 발생하게 되면 시스템의 유휴 자원들이 늘어나게 된다는 것을 의미한다. 또한 이것은 이 시스템이 2000정도 이상의 간격으로 패킷이 발생하면 원활하게 처리를 할 수 있다는 의미이기도 하다.



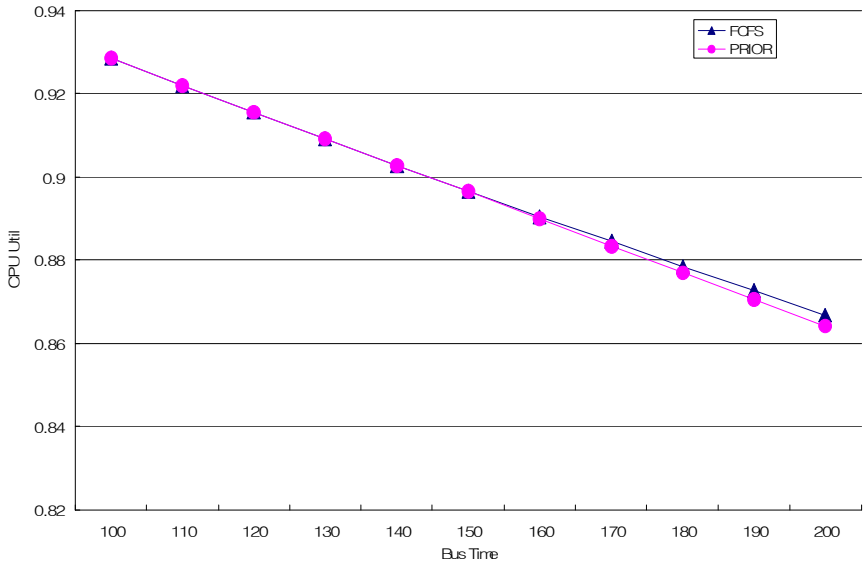
(그림 3.12) 패킷의 입력간격에 따른 하드웨어 활용도



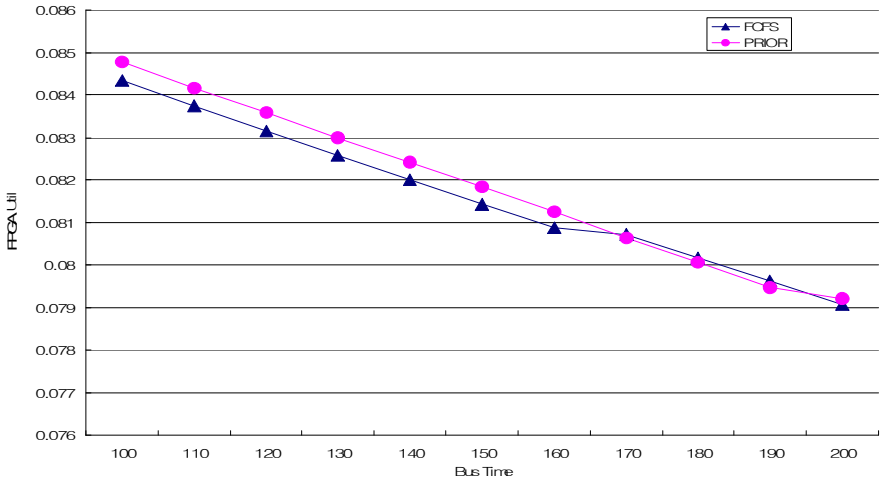
(그림 3.13) 패킷의 입력간격에 따른 병렬성

(그림 3.14)에서 (그림 3.18)는 버스에서 점유하는 시간에 따른 각 하드웨어 자원의 활용도와 병렬성 그리고 패킷들이 시스템에서 종료될 때까지 머문 시간의 평균값을 보여준다. (그림 3.14)을 보면 CPU의 활용도는 버스에서 점유하는 시간이 길어질수록 낮아지는 것을 볼 수 있고 두 가지(FCFS, 우선순위)스케줄링 방법들은 거의 차이가 없는 것으로 나타났다. (그림 3.15)를 보면 버스에서 점유하는 시간이 170 단위시간이하일 때는 우선순위 스케줄링 알고리즘을 사용하는 시뮬레이션 결과가 약간 높은 FPGA에 활용도를 보여주고 있으며, 버스의 활용도와 시스템의 병렬성은 (그림 3.14)의 CPU의 활용도와 마찬가지로 거의 동일한 결과를 보여주고 있다. 반면 버스의 점유시간 증가에 따른 처리된 패킷들이 평균적으로 시스템에 처리되고 종료된 시간 즉 평균적으로 시스템에 머문 시간을 나타낸 (그림 3.18)를 보면 우선순위 스케줄링 알고리즘을 사용한 시뮬레이션 결과가 모든 영역에서 빠른 처리를 하는

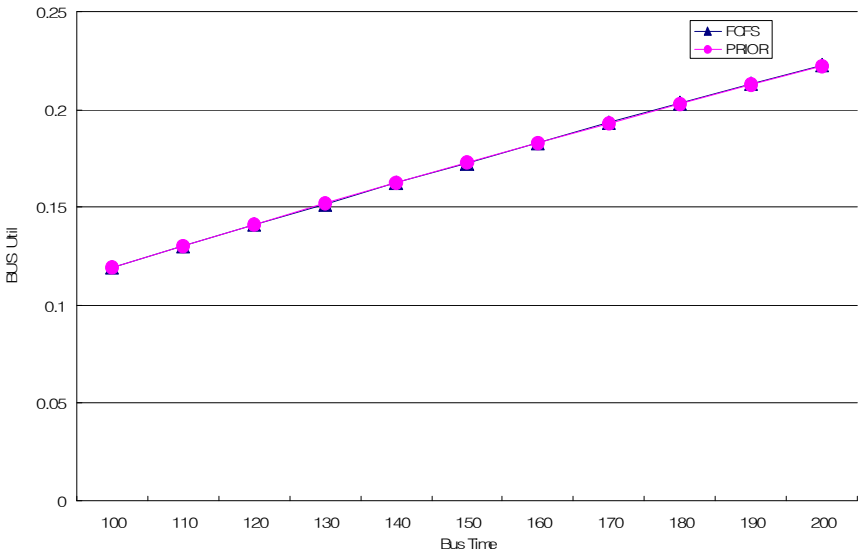
것으로 나타났다.



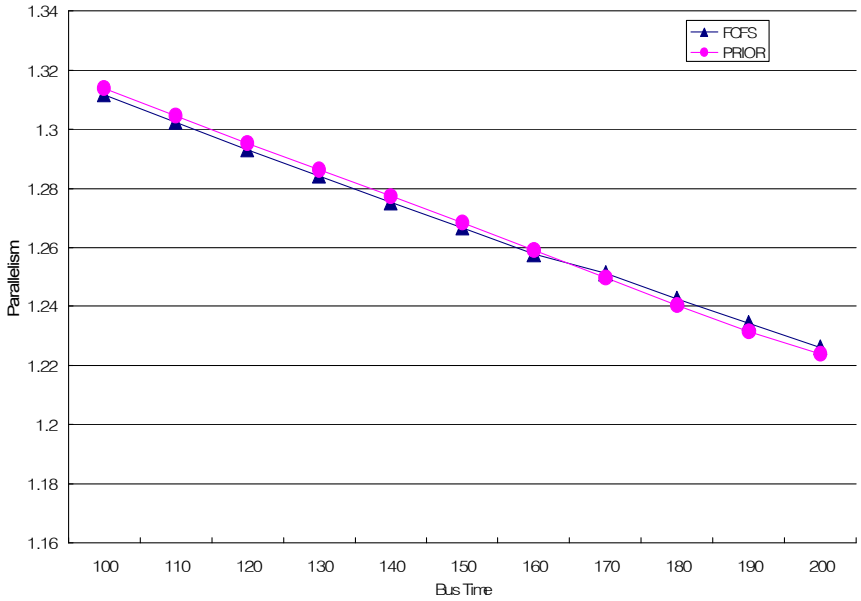
(그림 3.14) 버스 시간에 따른 CPU의 활용도(FCFS, 우선순위)



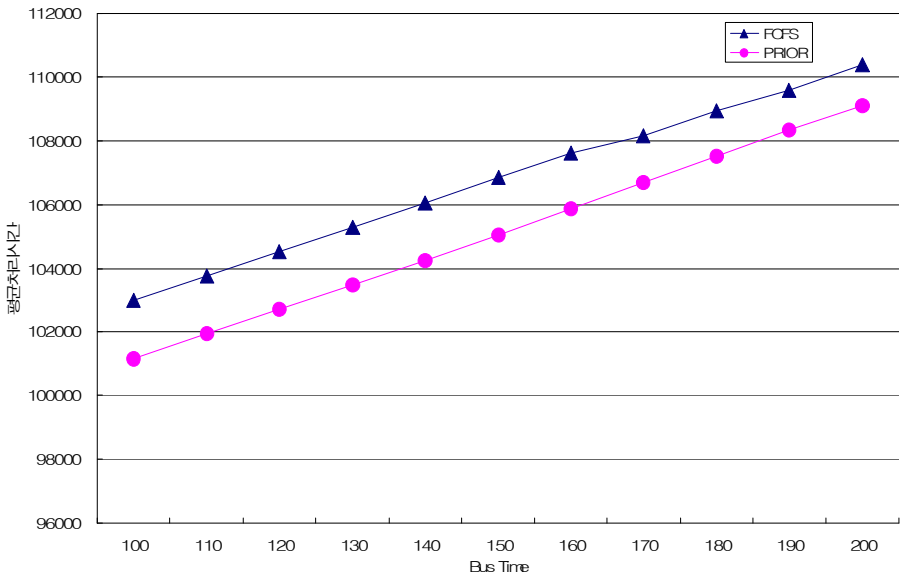
(그림 3.15) 버스 시간에 따른 FPGA의 활용도
(FCFS, 우선순위)



(그림 3.16) 버스 시간에 따른 BUS의 활용도(FCFS, 우선순위)



(그림 3.17) 버스 시간에 따른 병렬성(FCFS, 우선순위)

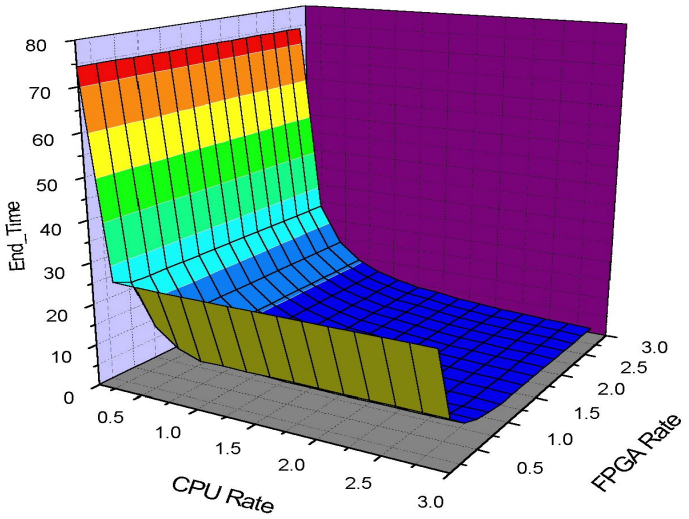


(그림 3.18) 버스 시간에 따른 평균처리시간
(FCFS, 우선순위)

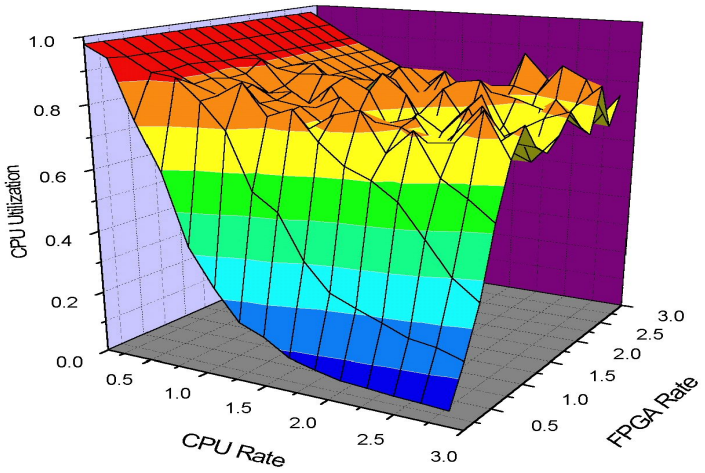
(그림 3.19)에서 (그림 3.222)는 CPU속도와 FPGA속도의 변화에 따른 시뮬레이션 종료시간, CPU 활용도, FPGA활용도 그리고 병렬성을 보여준다. 그림들에서 보여 지는 CPU 속도와 FPGA 속도는 기준이 되는 프로세서의 속도를 “1”로 정규화한 것으로 속도가 “2.0” 이라는 것은 기준 프로세서보다 2배 빠르다는 것이고 “0.5”는 두 배 느리다는 것을 의미한다.

(그림 3.19)을 보면 CPU속도와 FPGA속도가 증가함에 따라 성능은 계속 좋아 지는 모습을 보이다가 CPU속도가 1.2이고 FPGA 속도가 1.0 정도 되는 지점에서 기울기가 급격하게 완만해지는 것을 관찰 할 수 있다. 이것은 어느 정도까지는 하드웨어의 성능을 증가시키면 그에 상응하는 성능향상을 얻을 수 있지만 특정 지점(Knee Point)를 초과하게 되면 성능 향상은 크지 않다는 것을 알 수 있다. CPU 활용도와 FPGA 활용도, 병렬성을 보여주는 그림들에서

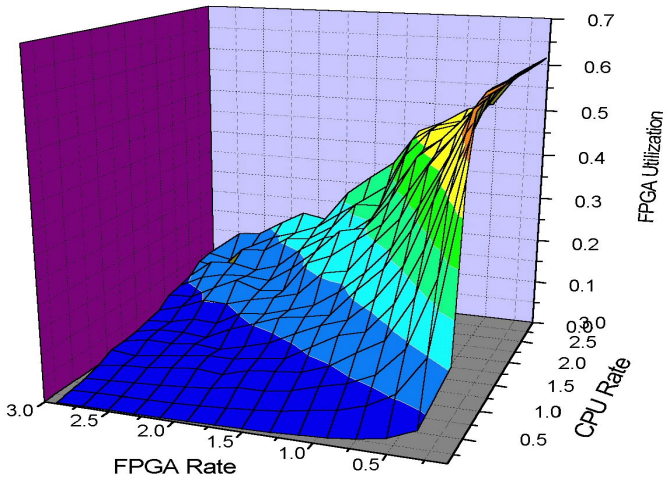
도 비슷한 결과를 관찰 할 수 있다.



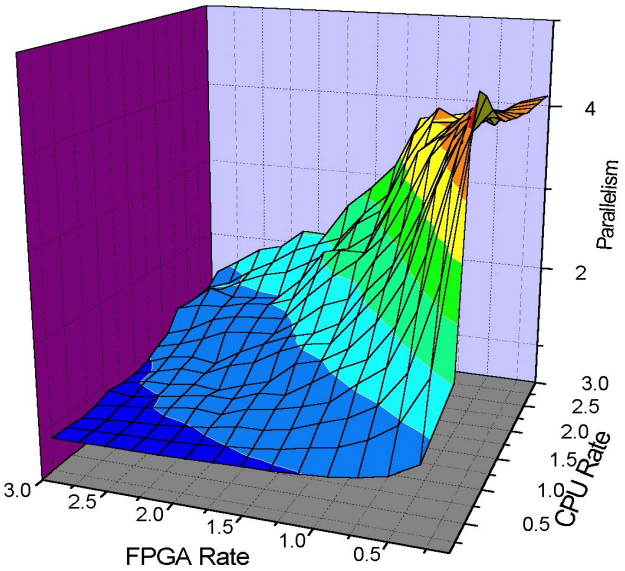
(그림 3.19) CPU 속도와 FPGA 속도의 변화에 따른 종료시간



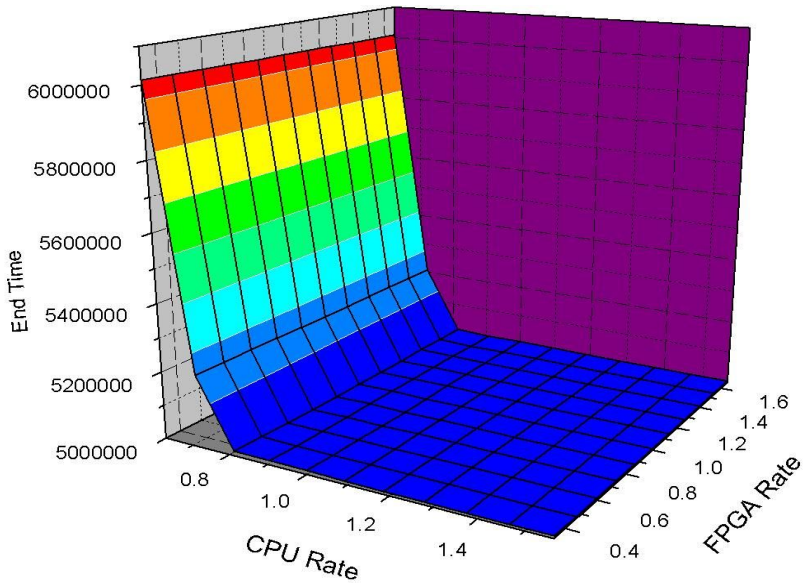
(그림 3.20) CPU 속도와 FPGA 속도의 변화에 따른 CPU의 활용도



(그림 3.21) CPU 속도와 FPGA 속도의 변화에 따른 FPGA 활용도



(그림 3.22) CPU 속도와 FPGA 속도의 변화에 따른 병렬성



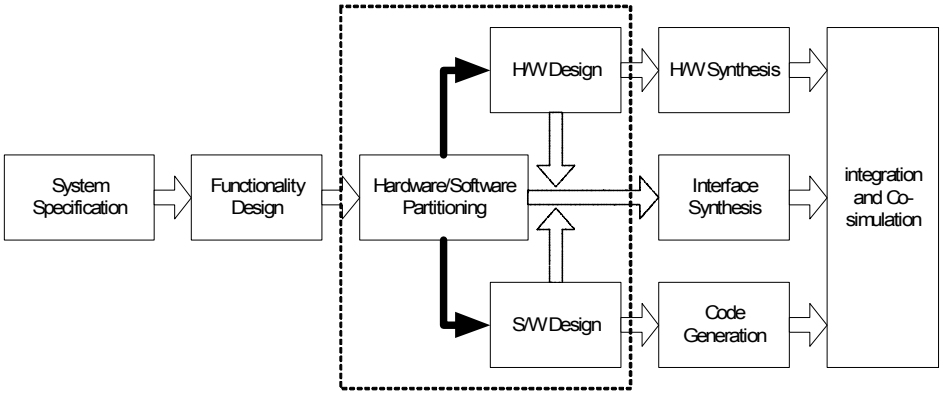
(그림 3.23) 실행 중 부분적 재구성 적용 시 시뮬레이션 종료시간

(그림 3.23) 실행 중 부분적 재구성 가능한 경우에 대한 시뮬레이션 결과이다. 전체적으로 실행 중 부분적 재구성을 적용하지 않았을 때와 유사한 결과를 볼 수 있으며 FPGA 속도 보다 CPU 속도의 영향을 이전의 실험 결과보다 더 많이 받은 것을 볼 수 있다.

Ⅳ. CSoC를 위한 HW/SW 분할

A. HW/SW 통합설계 및 분할

(그림 4.1)는 하드웨어 소프트웨어 통합설계를 위한 전형적인 구조를 보여준다. 그림에서 보여 진 것과 같이, 통합설계과정에서 하드웨어 소프트웨어 분할은, 최적의 시스템 성능을 보여주는 시스템을 설계하기 위한 핵심부분이다[37].



(그림 4.1) 하드웨어 소프트웨어 통합설계 과정

본 논문에서는 설계 공간 탐색 방법의 하나인 Y-chart 설계 공간 탐색 기법을 응용하여 수행 전 시스템 구성 정책을 가지는 정적 및 동적 재구성 가능 시스템에서 발생하는 하드웨어-소프트웨어 분할 문제를 푸는 방법을 제시

하고 이 방법을 사용하여 도구를 제작하였다. 이 방법은 먼저 하드웨어 요소와 응용 프로그램을 모델링한 결과를 이용하여 응용 프로그램의 기능들이 하드웨어 요소에 속하는 주 프로세서나 FPGA들에 어떻게 사상(mapping)될 수 있는지를 나타내는 사상 집합(mapping set)을 생성한다. 사상 집합을 생성할 때에 응용 프로그램의 여러 부분이 동일한 기능을 수행하게 될 경우에 발생할 수 있는 자원 요구 충돌 문제까지 고려한다. 그 다음 생성된 사상의 경우(mapping case)들을 시뮬레이션 하여 각각의 사상의 경우에 시스템이 얼마의 처리율을 가지는지를 보여준다. 각 사상의 경우에 대한 처리율을 비교해봄으로서 주어진 시스템에서 응용 프로그램이 가장 높은 처리율을 낼 수 있도록 하기 위해 응용 프로그램의 어떠한 기능들이 FPGA들로 구성해야 하는가를 알 수 있다. 이는 3장에서 하드웨어/소프트웨어 분할의 경우에 대한 시뮬레이션을 통하여 예를 들었다.

추가적으로 본 논문에서는 이 방법을 사용할 경우에 발생하는 사상 집합의 크기 문제를 해결하기 위해 사상 집합을 축소하는 휴리스틱 알고리즘을 제시한다. 응용 프로그램이 수행하는 기능 중에서 범용 프로세서와 FPGA들 중에서 선택할 수 있는 기능의 개수가 i이고 각 기능을 실행할 수 있는 프로세서의 수가 f_i일 때 사상 집합의 크기는 Πf_i 로 주어진다. 예를 들어, 응용 프로그램의 기능 중에 8개가 FPGA에서 동작할 수 있고, 시스템이 1개의 범용 프로세서와 3개의 FPGA를 가질 때 나타나는 사상 집합의 크기는 최대 $8^4 = 4096$이 된다. 이러한 사상 집합의 크기는 시스템의 복잡도에 따라 급수로 증가하여 시뮬레이션 시간에 심각한 영향을 미치기 때문에 시뮬레이션을 하지 않아도 되는 사상 집합을 제외할 필요가 있다. 그렇게 하기 위해, 시뮬레이션 이전에 각 사상의 경우에서 식별할 수 있는 두 가지 요소인 작업량과

병렬성이 FPGA를 사용하는 재구성 가능한 시스템의 처리율과 가지는 관계를 근거로 사상 집합을 축소하고, 그 결과 최적의 시스템 구성을 찾아내기 위해 Y-chart 설계 공간 탐색으로 소요되는 시뮬레이션 시간이 줄어든다.

B. CSoC를 위한 HW/SW 분할 휴리스틱

1. 관련연구

기존의 하드웨어-소프트웨어 분할에 대한 연구는 기존의 ASIC 설계나 병렬 처리 시스템 설계 분야에서 이루어져 왔다. 여기서는 하나의 응용 프로그램의 기능을 주어진 시스템의 하드웨어 자원에 적절히 배분하여 수행하도록 나누는 기법들과 비용이나 처리율과 같은 제약 조건들을 만족하는 설계 방법을 찾는 기법들이 연구되어 재구성 가능한 시스템을 설계하는데 참고할 수 있지만 FPGA와 같이 설계 자유도가 높은 재구성 가능한 장치를 적절히 이용하기 위해서는 발전된 연구가 필요하다. 따라서 최근에 재구성 가능한 시스템에 대한 하드웨어-소프트웨어 통합 설계에 대한 연구가 많이 수행되었다.

Lee등은 다양한 프로그램의 수행이 요구되는 PDA(Personal Digital Assistant)와 IMT2000(International Mobile Telecommunication) 터미널과 같은 다중 응용 내장형 시스템을 설계하는 데 적합한 재구성 가능한 시스템의 하부 구조와 하나의 FPGA에 여러 기능이 수행 시간에 구성되는 모델을 제시하였다[37].

Keinhuis는 Y-chart 설계 공간 탐색 방법을 제안하였다. 이 방법은 설계자가 응용 프로그램이 사상될 시스템 구성 요소에 대한 중요한 성능 척도를 정량적으로 정하도록 한다. 그는 이 방법을 사용하여 ORAS라고 하는 재구성 가능한 시뮬레이터를 만들었다[6]. ORAS는 시뮬레이션을 통해 시스템 구성 요소들을 이용해서 만들 수 있는 시스템의 구성에 대한 성능 수치(performance number)를 얻는다. 하지만 ORAS는 SBF라고 불리는 특정한 계산 모델에 한정되어 있기 때문에 재구성 가능한 시스템을 시뮬레이션하기 위

해서는 ORAS와는 다른 하부 구조가 필요하다[33].

Bakshi등은 시스템 수준의 명세와 처리율이 제한된 경우에 하드웨어-소프트웨어를 분할하기 위해 하드웨어와 소프트웨어를 모델링하고 파이프라인의 단계 형태로 나누어 시뮬레이션 하는 방법을 사용했다[38].

Kalavade등은 다양한 응용을 수행하는 내장형 시스템을 설계하는 문제를 지적했다. 하지만 이 연구에서는 FPGA에서의 분할은 고려되지 않았다[39].

Pruna등은 재구성 가능한 시스템에서 응용 프로그램을 재구성 가능한 장치들의 크기에 맞게 하위 작업들로 분할하고 자료 흐름 그래프(data flow graph)를 스케줄 한다. 하지만 이 연구에서는 하나의 FPGA에서 여러 기능이 수행되는 경우는 고려하지 않고 있다[40].

Hyunok Oh등은 다중 모드 다중 타스크를 지원하는 내장형 시스템의 하드웨어 소프트웨어 통합 설계 및 합성과정에서 실시간 제약사항에 만족하는 분할은 경우를 찾기 위하여 BILI 알고리즘을 이용하는 방법을 제시하였다[42].

하드웨어/소프트웨어 통합설계과정에서의 분할의 문제는 여러 연구를 통하여 진행되어오고 있지만 현재 내장형시스템의 중요 부품으로 활용되고 있는 CSoC를 기반으로 하는 어플리케이션에 대한 분할의 문제에 대하여는 연구가 상대적으로 미진하다.

2. 사상 집합 축소 휴리스틱 알고리즘

하드웨어-소프트웨어 사상 제어기에서 구한 가능한 사상의 경우의 개수가 Πf_i로 주어지기 때문에 응용프로그램의 모델에 속하는 하위 작업의 수와 그 기능 요구가 많아짐에 따라 그리고 하드웨어 모델에서 여러 하드웨어 요소들이 속하고 각 하드웨어 요소들이 같은 기능 요소들을 가지는 경우가 많아짐

에 따라 사상 집합의 크기는 매우 커지게 된다. 이러한 사상 집합의 크기는 시뮬레이션 시간에 심각한 영향을 미치기 때문에 가능하다면 시뮬레이션 이전에 사상 집합의 크기를 줄임으로 하드웨어-소프트웨어 분할을 위한 시뮬레이션과 성능 평가에 걸리는 시간을 줄일 수 있다. 그 방법으로는 먼저 FPGA의 사용 가능한 자원 양(FPGA 셀 개수)과 기능 요소들을 FPGA에 구성하기 위해 소요되는 자원 요구량을 고려하여 실제로 시스템이 구성할 수 없는 사상의 경우를 배제하는 방법이 있다. 그리고 각 사상의 경우에서 식별할 수 있는 두 가지 요소인 작업량과 병렬성이 FPGA를 사용하는 재구성 가능한 시스템의 처리율과 가지는 관계를 근거로 사상 집합을 축소함으로 시뮬레이션 시간을 줄일 수 있다.

사상 집합 축소 휴리스틱(HARMS: Heuristic Algorithm for Reducing Mapping Set)은 자원요구량에 따른 축소 단계와 작업량과 병렬성과의 관계를 이용하는 축소의 두 단계로 나누어진다. 또한 제안된 휴리스틱은 정적인 분할의 경우와 동적인 분할의 경우로 나누어 생각해 볼 수 있다. 정적인 분할의 경우는 하드웨어/소프트웨어 분할이 결정되게 되면 하드웨어로 분할된 기능요소들이 시스템이 동작하는 동안 고정된다는 의미이다. 동적인 분할의 경우는 시스템이 동작하는 동안 부분적 재구성이 가능하다는 것이다. 때문에 동적인 분할의 경우는 FPGA 전체 면적보다 더 큰 면적을 요구하는 기능요소가 존재하지 않는다면 첫 번째 단계 즉 자원요구량의 의한 축소는 의미가 없다고 볼 수 있다. 본 논문에서는 FPGA 전체 면적보다 더 큰 면적을 요구하는 기능요소는 무조건 소프트웨어로 사상된다고 가정한다.

a. 자원요구량에 근거한 사상집합의 축소

하나의 FPGA에 구성된 여러 가지의 기능을 동시에 수행하는 것은 가능하지

만 여러 기능을 하나의 FPGA에 구성하려면 각 기능이 FPGA에서 차지하는 셀의 개수의 합이 FPGA에서 허용하는 셀의 개수를 넘지 않아야 한다. 예를 들어 Xilinx사의 XC5202는 256개의 논리 셀(logic cell)을 가지고 있고 여기에 구현 가능한 논리 회로의 크기는 3,000개의 게이트(Gate)이하이다. XC5210의 경우에는 1296개의 논리 셀을 가지고 있고 최대 16000개의 게이트까지를 구현할 수 있으며 보통 10000개에서 16000개의 게이트 사이의 크기를 가지는 논리 회로를 구성할 수 있다. 이러한 하드웨어의 제약에 따라 하드웨어-소프트웨어 사상 제어기에서 생성된 사상 집합의 일부 사상의 경우는 실제로 구성될 수 없을 수 있다. 이것은 각 사상의 경우에 하드웨어 요소에 어떤 기능이 구성되는지 와 그 때 소요되는 FPGA의 셀 개수를 계산하여 하드웨어 요소가 실제로 그만큼의 셀 개수를 지원해 줄 수 있는지를 비교함으로 이루어진다.

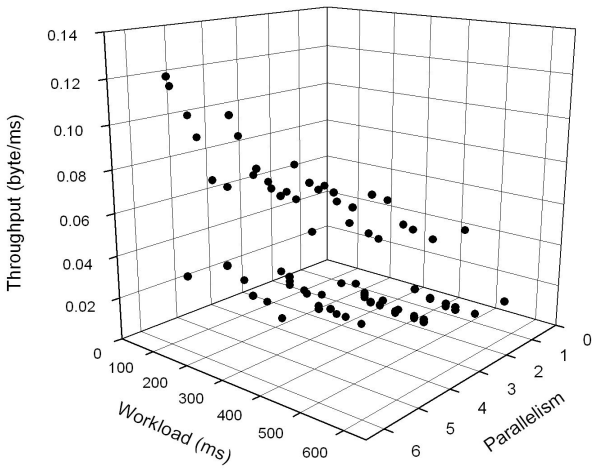
자원요구량에 의한 축소는 다음과 같은 과정을 거쳐서 수행된다. 먼저 모든 사상의 경우에 대하여 기능요소들은 앞장에서 설명한 바와 같이 사각형 영역을 요구한다고 가정한다. 특정 분할의 경우에 FPGA로 사상되는 것으로 결정된 기능 요소들을 사각영역의 FPGA에 매핑시켜서 실제로 매핑이 가능한지를 검사한다. 매핑은 기능요소들 중에서 자원요구량이 많은 순서대로 First Fit 방법을 사용하여 진행하였다. 그 결과 FPGA로 사상되는 모든 기능요소가 FPGA에 매핑이 되지 않으면 사상집합에서 제외시킨다. 동적인 경우는 언급한 바와 같이 FPGA 전체 면적보다 더 큰 FPGA 면적 요구를 가진 기능요소가 FPGA로 할당된 경우만 배제한다.

b. 작업량과 병렬성에 근거한 사상집합의 축소

재구성 가능한 시스템은 시스템의 성능을 높이기 위해 보조 프로세서나 하드웨어 가속기를 FPGA와 같이 재구성 가능한 장치에 구성하며 같은 기능을 수행할 경우 클럭 속도의 차이가 많이 날 때에도 범용 프로세서에서보다 재구성 가능한 장치에서 수행할 때에 더 높은 성능을 보인다. 또한 FPGA에서 수행되는 기능은 범용 프로세서와 병렬로 수행될 수 있고 FPGA내에 구성된 여러 기능 역시 서로 간에 간섭이 없이 병렬로 수행될 수 있기 때문에 병렬성이 높아지게 된다. 이에 따라 FPGA를 사용하는 시스템의 경우에서는 어떤 사상 집합 M내의 사상의 경우를 $m1$과 $m2$라고 할 때 각각의 사상의 경우에 응용 프로그램의 하위 작업 각각에 소요되는 수행 시간의 합인 작업량(workload) $W1$, $W2$와 하위 작업의 기능들이 최대로 동시에 수행될 수 있는 수인 병렬성(parallelism) $P1$, $P2$의 관계는 각각의 처리율 $T1$, $T2$에 대해 대체적으로 다음을 만족한다.

$$\begin{array}{l} \text{If } W1 > W2 \text{ and } P1 < P2 \rightarrow T1 < T2 \\ \text{If } W1 < W2 \text{ and } P1 > P2 \rightarrow T1 > T2 \end{array} \qquad \text{(식 2)}$$

즉, FPGA를 사용함으로서 작업량을 줄이면서 병렬성을 높인 경우의 처리율은 그렇지 않은 경우보다 대체적으로 더 높다. (그림 4.2)에서는 H.263 해석기(decoder)를 하나의 범용 프로세서와 세 개의 FPGA에서 수행할 때를 시뮬레이션한 결과로부터 병렬성에 따른 작업량과 자료 단위 당 수행시간의 관계를 구한 그래프를 보이고 있으며 위의 식을 만족함을 볼 수 있다. 이러한 관계에 근거하여 사상 집합 축소 휴리스틱 알고리즘을 만들 수 있다.



(그림 4.2) H.263 해석기의 처리량의 병렬성과 작업량 관계

작업량과 병렬성의 이용한 축소 휴리스틱은 다음과 같은 과정을 거쳐서 이루어진다[42].

```
M1 = Find_Max_Parallism(Find_Min_Workload(Mapping Set))
M2 = Find_Min_Workload(Find_Max_Parallism(Mapping Set))
For All m in Mapping Set
        if P(m) < P(M1) and W(m) > W(M2)
                Exclude m from Mapping Set
```

(알고리즘 1) 사상 집합 축소 휴리스틱

Find_Min_Work(Mapping Set)은 주어진 사상 집합 안에서 최소 작업량을 가지는 사상의 경우들을 탐색하여 새로운 사상 집합을 생성하는 과정이다. 이는 하드웨어로 할당된 기능요소들의 수행시간들의 합과 소프트웨어로 할당된 수행시간들의 합을 더함으로써 구해진다.

Find_Max_Parallism(Mapping Set)은 주어진 사상 집합 안에서 최대 병렬성을 가지는 사상의 경우들을 탐색하여 새로운 사상 집합을 생성하는 과정이다. 이것은 정적인 경우에는 하드웨어 할당된 기능요소들의 수에 1을 더하여 구해지고, 동적인 경우에는 FPGA에 할당될 수 있는 기능요소들의 최대 값에 1을 더해서 구해진다. 여기에서 1을 더하는 것은 FPGA가 동작하는 동안 CPU에서도 하나의 기능요소를 수행할 수 있음을 의미한다.

본 논문에서는 사상 집합 내의 각 사상의 경우들에 대해 다른 사상들과 작업량과 병렬성에 대한 비교를 하여 그 사상의 경우보다 높은 작업량과 낮은 병렬성을 가지는 사상의 경우들은 사상 집합에서 배제하는 알고리즘을 제안하고 실험하였다.

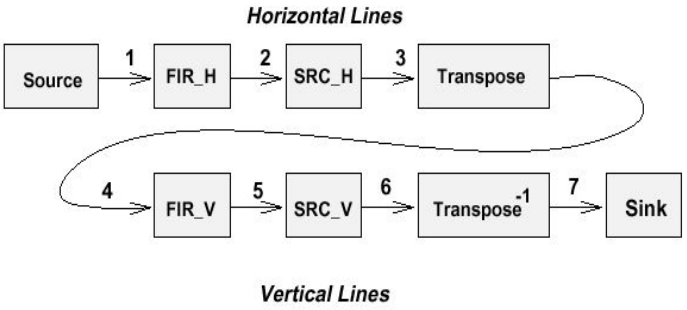
3. 실험 결과

본 논문에서 제안한 사상 집합 축소 휴리스틱 알고리즘을 수행하였을 때 사상 집합을 얼마나 축소할 수 있는가와 높은 성능을 내는 시스템 구성을 선택해 내는가를 실험하였다.

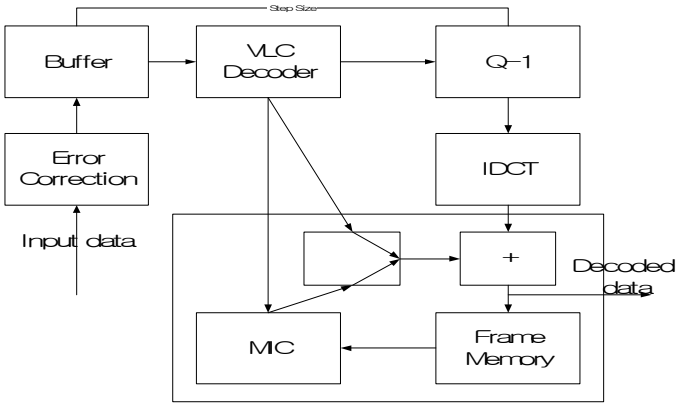
a. 시뮬레이션 환경

하드웨어 구조 모델로서 일반적으로 재구성 가능한 내장형 시스템에서 주로 사용하는 하나의 범용 프로세서와 FPGA와 같은 재구성 가능 장치를 가진 CSoC 모델을 사용하였다. 응용 프로그램 모델로서는 이미지크기를 변경하여 두 개의 이미지를 하나의 TV화면에 보여주는 Picture in Picture 알고리즘(그림 4.3), H.263 해석기(decoder)(그림 4.4) 그리고 3장에서 사용된 H.263 Encoder 모델(그림 3.10)을 택하였다. H.263 해석기의 경우 순환이 있는 부분

을 하나의 하위 작업으로 묶어서 데이터가 한 방향으로 흘러가도록 재 모델링 하였다. H.263은 널리 쓰이는 비디오 신호 처리 모델로서 데이터의 흐름을 나타내기에 적합하고 Picture in Picture는 여러 하위 작업들이 같은 기능을 사용하기 때문에 이 때 발생하게 되는 처리 요구 충돌 문제가 시스템에 어떤 영향을 미칠 수 있는지 고려할 수 있기 때문에 이 세 응용 프로그램을 선택하였다.



(그림 4.3) Picture in Picture 처리기 모델



(그림 4.4) H.263 해석기 모델

각 프로그램의 모델에 대해 일반 프로세서에서 수행될 경우와 FPGA에서 수행될 경우에 기능요소들의 수행시간과 FPGA 면적요구에 관한 정보는 다음 표들에 나타내었다.

<표 4.1> 기능요소들의 수행시간과 FPGA자원 요구(PiP)

PE_ID	FE_ID	소프트웨어 수행시간 (ns)	하드웨어 수행시간 (ns)	가로	세로	FPGA 자원요구량 (cell)
0	0	600	150	25	20	500
1	1	500	100	25	30	750
2	2	800	160	30	25	750
3	3	600	120	10	40	400
4	4	500	100	30	25	750
5	5	800	160	30	30	900

<표 4.2>기능요소들의 수행시간과 FPGA자원 요구(H.263 Decoder)

PE_ID	FE_ID	소프트웨어 수행시간 (ns)	하드웨어 수행시간 (ns)	가로	세로	FPGA 자원요구량 (cell)
0	0	1200	210	40	20	800
1	1	2500	420	30	30	900
2	2	1600	350	25	30	750
3	3	5800	1250	30	25	750
4	4	8000	1790	35	10	350
5	5	3400	750	40	30	1200

<표 4.3> 기능요소들의 수행시간과 FPGA자원 요구(H.263 Encoder)

PE_ID	FE_ID	소프트웨어 수행시간 (ns)	하드웨어 수행시간 (ns)	가로	세로	FPGA 자원요구량 (cell)
0	0	600	150	30	15	450
1	1	1100	360	40	30	1200
2	2	5800	1250	30	25	750
3	3	10300	2550	50	20	1000
4	4	8000	1790	35	10	350
5	5	7600	1670	20	45	900
6	6	1400	450	30	25	750
7	7	5600	1100	45	15	675
8	8	2440	840	15	30	450
9	6	1400	450	30	25	750
10	9	1500	350	35	40	1400

b. 실험 결과

<표 4.4>는 정적인 경우에 대한 HARMS 알고리즘의 축소 효율을 보여준다. 세 가지 어플리케이션에 대하여 FPGA 자원이 적은 경우와 중간인 경우 많은 경우에 대하여 시뮬레이션하고 축소 효율을 보였다. 표에서 보여진 바와 같이 PiP 어플리케이션의 경우 상대적으로 충분한 FPGA 자원을 가정할 경우 축소 효율이 작은 것을 볼 수 있는데 이는 충분한 FPGA를 사용하는 경우에는 많은 사상의 경우를 적용시킬 수 있기 때문에 좋은 성능을 나타내는 사상을 그 만큼 많이 선택할 수 있기 때문이다. H.263 Encoder의 경우도 비슷한 결과를 관측할 수 있다. 이 실험 결과는 전체적으로 80%이상 사상집이 축소

되었음을 알 수 있는데 이는 어플리케이션의 특성과 수행시간 그리고 면적정보에 따라 다르기 때문에 절대적으로 평가할 수는 없는 수치이다.

<표 4.4> 정적인 경우에 대한 축소 효율

Application		Full Space	Phase1 (Resource)	Phase2 (HARMS)	Efficiency (%)
PiP	Small FPGA (50*50)	64	20	3	85
	Medium FPGA (55*55)	64	37	3	91.9
	Large FPGA (60*60)	64	45	23	48.9
H.263 (Decoder)	Small FPGA (50*50)	64	16	8	50
	Medium FPGA (60*60)	64	37	4	89.2
	Large FPGA (70*70)	64	44	6	86.4
H.263 (Encoder)	Small FPGA (70*70)	1024	374	49	86.9
	Medium FPGA (80*80)	1024	653	27	95.9
	Large FPGA (90*90)	1024	941	304	67.7

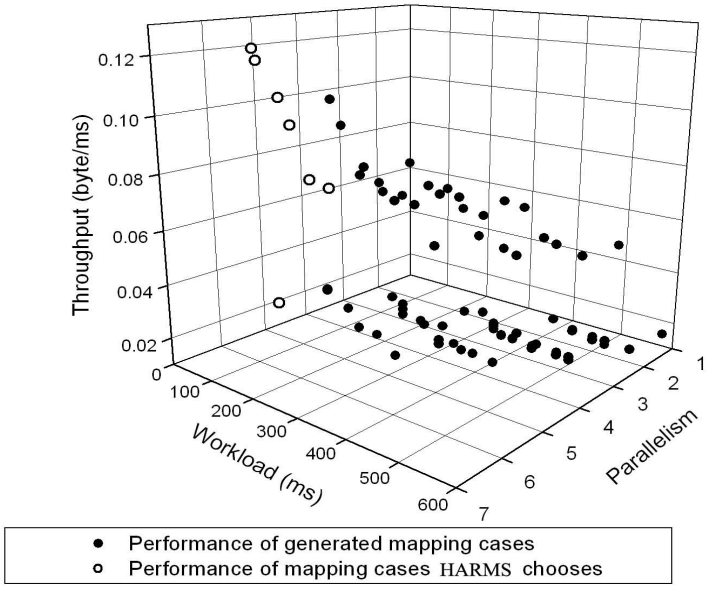
<표 4.5> 는 동적인 경우에 대한 축소 효율을 보여주는 것이다. 간단한 형태를 가진 PiP나 H.263 Decoder 어플리케이션의 경우는 축소 효율이 정적인 경우보다 낮게 나타난다. 하지만 H.263 Encoder와 같은 복잡한 어플리케이션에 대해서는 축소 효율이 상대적으로 좋게 나타난다. 이는 HARMS 알고리즘이 복잡한 어플리케이션에 더 효율적이라는 것을 알 수 있다.

<표 4.5> 동적인 경우에 대한 축소 효율

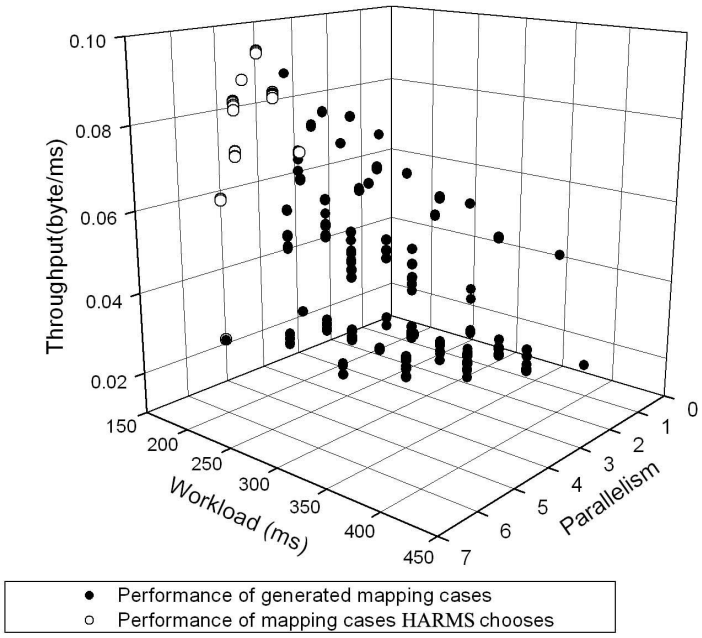
Application		Full Space	Phase1 (Resource)	Phase2 (HARMS)	Efficiency (%)
PiP	Small FPGA (50*50)	64	64	32	50
	Medium FPGA (55*55)	64	64	17	73.4
	Large FPGA (60*60)	64	64	19	70.3
H.263 (Decoder)	Small FPGA (50*50)	64	64	37	42.2
	Medium FPGA (60*60)	64	64	20	68.8
	Large FPGA (65*65)	64	64	21	67.2
H.263 (Encoder)	Small FPGA (70*70)	1024	1024	68	93.3
	Medium FPGA (80*80)	1024	1024	10	99
	Large FPGA (90*90)	1024	1024	30	97.1

c. 사상 집합 축소 휴리스틱의 효용성

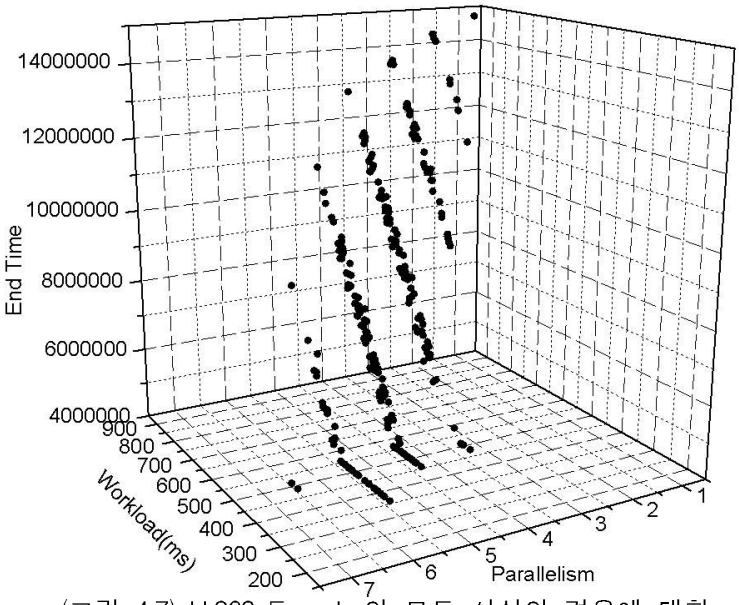
(그림 4.5) 과 (그림 4.6) 는 H.263 Decoder 와 PiP(Picture in Picture) 어플리케이션에 대하여 전체 사상 집합에 대한 성능과 HARMS 알고리즘을 수행했을 때 선택된 사상집합을 보여준다. 그림들에서 보여진 바와 같이 HARMS 에 의해서 생성된 사상집합이 일반적으로 우수한 성능을 나타낸다는 것을 알 수 있다.



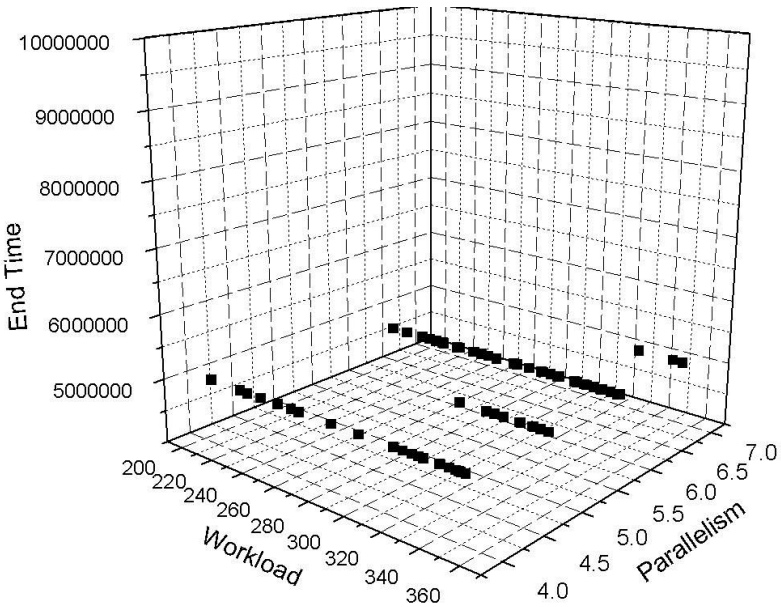
(그림 4.5) H.263 Decoder의 HARMS 수행 전후의 사상집합의 비교



(그림 4.6)PiP의 HARMS 수행 전후의 사상집합의 성능비교



(그림 4.7) H.263 Encoder의 모든 사상의 경우에 대한 종료시간(실행 중 재구성 적용)



(그림 4.8) H.263 Encoder의 HARMS 수행이후 선택된 사상의 경우에 대한 종료시간(실행 중 재구성 적용)

(그림 4.7) 과 (그림 4.8) 는 H.263 Encoder 어플리케이션에 대하여 실행 중 부분적 재구성을 적용했을 경우 전체 사상 집합에 대한 성능과 HARMS 알고리즘을 수행했을 때 선택된 사상집합의 시뮬레이션 종료 시간을 보여준다. 다른 어플리케이션들에서와 마찬가지로 좋은 성능을 보이는 사상의 경우들이 HARMS 알고리즘에 의해서 선택되는 것을 볼 수 있다.

V. 결론 및 향후 연구과제

A. 결론

본 논문에서는 CSoC를 위한 설계 공간 탐색 도구를 개발하였고 H.263모델에 적용하여 실험하였다. 설계 공간 탐색을 위하여 내장형 시스템 설계 과정에서 선택적으로 적용할 수 있는 설계변수들을 적용하여 시뮬레이션하기 위하여 Y-chart 설계 방법을 응용하여 설계 공간 탐색 도구를 개발하였다. 또한 선택 가능한 설계변수들을 적용하여 다양한 하드웨어 구성을 생성하여 시뮬레이션 수행이 가능한 재구성 가능한 시뮬레이터를 개발하였다.

구현된 재구성 가능한 시뮬레이터는 주어진 어플리케이션을 시뮬레이션하는 부분과 하드웨어를 시뮬레이션하는 부분 그리고 어플리케이션과 하드웨어를 매핑시켜 주는 매핑제어기로 구성되어있다. 어플리케이션 모델은 디지털 신호처리의 모델링 방법으로 널리 사용되는 Kahn 프로세스 네트워크를 사용하였다. 전체적으로 구현된 시뮬레이터는 어플리케이션 시뮬레이터가 발생시키는 트레이스를 매핑 제어기가 해당되는 하드웨어자원에 할당하는 Trace-Driven 시뮬레이션 방법을 사용하여 구현되었다.

구현된 설계 공간 탐색도구는 일반적인 CPU와 FPGA와 같은 재구성 가능한 하드웨어자원을 같이 사용하는 내장형 시스템을 위해 개발되었다. 이 도구를 이용하여 시스템 설계자는 설계하고자하는 시스템에서 적용 가능한 하드웨어/소프트웨어 분할의 경우에 대한 성능수치를 산출하여 최적의 하드웨어/소프트웨어 분할의 경우를 설계자가 결정하는데 기본적인 정보를 제공해 줄 수 있다. 또한 이 도구는 설계상에서 선택적으로 적용 가능한 설계변

수들의 변화에 따라 성능 수치가 어떻게 변하는지 시뮬레이션을 통하여 산출하여 시스템 설계자로 하여금 최소 비용으로 시간 및 자원 제약사항을 만족하는 시스템구조를 설계하는데 기본적인 자료를 제공해준다. 추가로 구현된 설계공간 탐색도구는 실시간 운영체제의 가장 중요한 부분인 스케줄링 알고리즘에 따른 성능을 예측할 수 있도록 실시간 체제를 위한 가장 간단한 구조의 FCFS 스케줄링 방법과 우선순위 스케줄링 방법을 적용했을 때의 성능 수치를 산출할 수 있도록 함으로써 실시간 운영체제 설계자에게도 기본 정보를 제공한다.

추가적으로 본 논문에서는 정적인 경우와 동적인 경우에 대하여 하드웨어/소프트웨어 분할 문제를 해결하기 위한 사상집합 축소 휴리스틱을 제안하고 그 실험결과를 통하여 그 효용성을 입증하였다. 이와 같이 본 논문을 통하여 구현된 재구성 가능한 시뮬레이터를 이용한 설계 공간 탐색도구와 하드웨어/소프트웨어 분할을 위한 사상집합 휴리스틱은 시스템 설계자로 하여금 실제 프로토타입을 구축하지 않고 설계하고자하는 시스템을 다양한 설계변수를 선택적으로 적용하여 시뮬레이션하고 성능수치를 산출할 수 있게 함으로써 설계시간과 설계비용을 현저하게 줄여줄 것으로 기대된다.

B. 향후 연구 과제

본 논문을 통하여 구현된 재구성 가능 시뮬레이터는 각 기능요소 소프트웨어 및 하드웨어 실행시간을 이미 알고 있다고 가정하고 시뮬레이션을 진행한다. 이는 추후에 소프트웨어 수행시간 분석도구와 하드웨어 실행시간 분석도구와 통합이 요구된다. 또한 본 논문에서 사용된 실행 중 부분적 재구성을 위한 절차에 관하여 향후 오버헤드를 줄이고 효율적인 FPGA 자원 활용을 위한 방법에 대한 추가적인 연구가 필요할 것이다. 추가적으로 Kahn Process Network 이외에 다양한 어플리케이션 모델도 지원하고 설계변수에 대해서도 다양화하고 구체화하여 다양한 내장형 플랫폼에 대한 하드웨어 구성을 고려할 수 있는 구조로 확장되는 것이 필요하다.

본 논문을 통하여 구축된 설계 공간 탐색 환경은 상위 수준의 성능 분석을 수행하기 때문에 실제 시스템에서의 결과와 100% 일치한다고 보장하기는 어렵다. 때문에 실제 시스템에서 검증을 하는 것이 향후 연구 과제로 제시 될 수 있다. 하지만 현재 개발된 하드웨어 기술로 구현은 가능하다고 판단되지만 실제 사례는 아직까지 없다. 하드웨어 기술에 관하여 2D 모델 관리를 위하여 기본적인 CSoC장치에 추가적인 회로가 필요할 것이고 이에 대한 많은 연구가 필요하다.

추가적으로 본 논문을 통하여 구축된 설계 공간 탐색환경은 CSoC를 위한 설계 자동화 도구에 통합되어야 하며 설계 변수의 변화와 하드웨어/소프트웨어 분할을 동시에 고려하는 설계 공간 축소에 대한 연구가 필요하다.

참고문헌

[1] E. Sanchez. M. Sipper, J. O. Haenni, J. L. Beuchat, A. Stauffer, and A. Perez-Uribe, "Static and Dynamic Configurable Systems," IEEE Transactions on Computers vol.48, no.6, June 1999.

[2] P. Athanas, A. Abbott, "Real-Time Image Processing on a Custom Computing Platform," IEEE Computer, February 1995.

[3] S. Choi and V. K. Prasanna, "Fast Parallel Implementation of DFT using Configurable Devices," in International Conference on Parallel and Distributed Systems, December 1997.

[4] A. Danadalis and V.K. Prasanna, "Fast Parallel Implementation of DFT using Configurable Devices," in 7th International Workshop on Field-Programmable Logic and Applications, September 1997.

[5] R.J. Pertersen and B. Hutchings, "An Asessment of the Suitability of FPGA-Based Systems for use in Digital Signal Processing," in 5th International Workshop on Field-Programmable Logic and Applications, 1995.

[6] A. Rashid, J. leonard, and W. H. Mangione-Smith, "Dynamic Circuit Generation for Solving Specific Problem Instances of Boolean Satisfiability," IEEE Symposium on FPGAs for Custom Computing Machines, April 1998.

[7] J. Hauser and J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor," in IEEE Symposium on FPGAs for

Custom Computing Machines, April 1997.

[8] Chameleon Systems, "http://www.chameleonsystems,com/,".

[9] Triscend Corporation, "http://www.triscend,com/,".

[10] Xilinx Inc."http://www.xilinx.com/," Xilinx Platform FPGAs.

[11] Altera Inc. "http://www.altera.com/,".

[12] K. Bondalapati, P. Diniz, P. Duncan, J. Granacki, M. Hall, R. Jain, and H. Ziegler, "DEFACTO: A Design Environment for Adaptive Computing Technology," in Reconfigurable Architecture Workshop RAW'99 , April 1999.

[13] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli, "Architecture of Field Programmable Gate Arrays," Proceedings of the IEEE, July 1993.

[14] K. Bondalapati, "Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures," Ph.D. thesis, University of Southern California, May 2001.

[15] K. Bondalapati and V. K. Prasanna, "Reconfigurable Computing: Architectures, Models and Algorithms," Current Science, vol.78, no. 7 April 2000.

[16] S. Hauck, "The Roles of FPGAs in Programmable Systems," Proceedings of the IEEE, vol. 86, April 1998.

[17] P. Schaumont, I. Verbauwhede, K. Kuetzer, and M. Sarrafzadeh, "A Quick Safari through the Reconfiguration Jungle," 38th Design Automation Conference, June 2001.

[18] P. Master and K. Lane, "Powering up 3G Handsets for MPEG-4

Video," Communication Systems Design, January 2001.

[19] Katherine Compton, Scott Hauck, "Reconfigurable Computing: A Survey of Systems and Software," ACM Computing Surveys, 2002.

[20] The Programmable Logic Data Book, San Jose, CA: Xilinx, Inc., 1994.

[21] Data Book, San Jose, CA: Altera Corporation, 1998.

[22] PGA Data Book, Allentown, PA: Lucent Technnologies, Inc., 1998.

[23] S. Trimberger, D. Carberry, A. Johnson, J.Wong, "A Time-Multiplexed FPGA," IEEE Symposium on Field-Programmable Custom Computing Machines, 1997.

[24] S. Hauck, T. W. Fry, M. M. Hosler, J. P. Kao, "The Chimaera Reconfigurable Functional Unit," IEEE Symposium on Field-Programmable Custom Computing Machines, 1997.

[25] S. Cadambi, J. Weener, S. C. Goldstein, H. Schmit, D. E. Thomas, "Managing Pipeline-ReConfigurable FPGAS," ACM/SIGDA International Symposium on FPGAs, 1998.

[26] C. R. Rupp, M. Landguth, E. Gomersall, H. Holt, J. M. Arnold, M. Gokhale, "The NAPA Adaptive Processing Architecture," IEEE Symposium on Field-Programmable Custom Computing Machines, 1998.

[27] Virtex 2.5V Field Programmable Gate Arrays: Advance Product Specification, San Jose, CA: Xilinx, Inc., 1999.

[28] S. Hauck, "Configuration Prefetch for Single Context Reconfigurable Coprocessors," ACM/SIGDA International Symposium on FPGAs, 1998.

[29] Sung Hyun Lee, Sung Joo Yoo, Kiyoung Choi, "Reconfigurable SoC

Design with Hierarchical FSM and Synchronous Dataflow Model," International Workshop on Hardware/Sotfware Codesign, May 2002.

[30] A. Dandalis, V. Prasanna, "Configuration Compression for FPGA-based Embedded Systems," ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2001.

[31] K. Comptom, Z. Li, J. Cooley, S. knol, S. Hauck, "Configuration Relocation and Defragmentation for Run-time Reconfigurable Computing," IEEE Transactions on VLSI Systems, vol. 10, no. 3, pp. 209-220, June 2002.

[32] K. Comptom, Z. Li, J. Cooley, S. knol, S. Hauck, "Configuration Relocation and Defragmentation for FPGAs," Northwestern University Technical Report, 2000.

[33] B. Kienhuis, E. Deprettere, K. A. Vissers, and P. Wolf. "An approach for quantitative analysis of application-specific dataflow architectures," In Proceedings of 11th Intl. Conference of Applications-specific Systems, Architectures and Processors (ASAP'97), Zurich, Switzerland, 1997.

[34] A.C.J. Kienhuis, "Design Space Exploration of Stream-based Dataflow Architectures," PhD thesis, Delft University of Technology, Netherlands, 1998.

[35] G. Kahn, "The semantics of a simple language for parallel programming," Info. Proc. of the IFIP Congress 74, August 1974.

[36] Matthias Dyer, Chistian Plessl, and Marco Platzner, "Partially

Reconfigurable Cores for Xilinx Virtex," FPL 2002, LNCS 2438, Springer-Verlag Berlin Heidelberg, 2002.

[37] S. Lee, B. Jeong, S. Yoo, K. Choi, S. Hong, S. Moon, "A New Design Framework for Multiple-Application Embedded Java Systems wih Reconfigurable Target Architectures," ACM SIGPLAN 1999 Workshop on Languages, Compilers, and Tools for Embedded Systems, 1999.

[38] S. Bakshi, D. D. Gajaski, "Hardware/ Software Partitioning and Pipelining," In Proceedings of the 34th annual conference on Design Automation Conference, 1997.

[39] A. Kalavade, P. A. Subrahmanyam, "Hardware/Software Partitioning for Multifunction Systems," In Proceedings of International Conference on Computer Aided Design, 1997.

[40] E. Caspi, M. Chu, R.Huang, J. Yeh, Y. Markovskiy, J. Wawrzynek, and A. DeHon, "Stream Computations Organized for Reconfigurable Execution(SCORE): Introduction and Tutorial, UC Berkeley BRASS research group technical report," August 2000.

[41] Christoph Steiger, Herbert Walder, Marco Platzner, "Operating Systems for Reconfigurable Embedded Platforms: Online Scheduling of Real-Time Tasks," IEEE Transactions on Computers, vol. 53, no. 11, November 2004.

[42] Hyunok Oh, Soonhoi Ha, "Hardware-Software Cosynthesis of Multi-Mode Multi-Task Embedded Systems with Real-Time Constraints," 2002 International Workshop on Hardware/Software

Codesign, May 2002.

[43] Jun-Yong Kim, Seong-Yong Ahn, Jenog-A Lee, "Hardware/Software Partitioning Methodology for Reconfigurable System," The Transactions of the KIPS, vol 11-A, no 5, October 2004.

[44] Paul Lieverse, Pieter Van Der Wolf, Kees Vissers, Ed Deprettere, "A Methodology for Architecture Exploration of Heterogeneous Signal Processing Systems," Journal of VLSI Signal Processing Systems vol. 29, issue 3, November 2001.

[45] Kaiyu Chen, Sharad Malik, David I. August, "Retargetable Static Timing Analysis for Embedded Software," Proceedings of the International Symposium on System Synthesis (ISSS), October, 2001.

국문초록

재구성 가능 SoC 설계 공간의 효율적인 탐색을 위한 연구

안 성 용

지도교수 : 이 정 아(Ph.D)

조선대학교 대학원 전자계산학과

재구성 가능한 아키텍처란, 기존의 고정된 하드웨어와는 달리, 주어진 응용프로그램을 효율적으로 수행하기 위하여 하드웨어 구조를 변환시킬 수 있는 시스템을 지칭한다. 본 논문에서 다루는 재구성 가능 SoC는 여러 다양한 하드웨어 구성방식 중, 두 가지 형태의 아키텍처의 혼합을 전제한다. 첫 번째 형태는 기존의 고정된 아키텍처인 CPU이고 두 번 째 형태는 응용프로그램에 따른 하드웨어 재구성이 가능한 FPGA로 구성되어 있다고 전제한다. 단일 응용프로그램을 효율적으로 수행하기 위한 ASIC을 개발하는 것과 달리, 일련의 연관된 특정 응용프로그램들을 효율적으로 수행할 수 있는 하드웨어 구성을 찾기 위하여는, 다양한 설계변수의 상관관계를 찾아내는 복잡한 문제를 해결하여야 한다. 이러한 문제를 해결하기 위하여 단일형 하드웨어 구조를 전제로 응용프로그램들을 하드웨어 구조에 맵핑하여 예상 성능 수

치를 측정하고 분석하여, 여러 다양한 설계변수를 결정짓는 체계적인 방법론으로 개발된 Y-chart 설계방법을 적용하였다.

본 논문에서는 기존의 Y-Chart 설계공간 탐색방법을 재구성 가능한 SoC를 기반으로하는 내장형 시스템에서 적용할 수 있도록, 재구성 가능한 시뮬레이터 기능을 보완하여 개발하였고 이를 H.263 모델에 적용하여 실험하였다. 구현된 재구성 가능한 시뮬레이터는 주어진 어플리케이션을 시뮬레이션하는 부분과 하드웨어를 시뮬레이션하는 부분 그리고 어플리케이션과 하드웨어를 매핑시켜주는 매핑제어기로 구성되어있다. 어플리케이션과 하드웨어 간의 의미론적 차이를 최대한 줄이기 위하여 어플리케이션의 입력은 디지털 신호처리의 모델링 방법으로 널리 사용되는 Kahn 프로세스 네트워크로 전제하였다.

전체적으로 구현된 시뮬레이터는, 어플리케이션 시뮬레이터가 발생시키는 트레이스를 매핑제어기가 해당되는 CPU 또는 스트림 기반의 하드웨어자원에 할당하는 Trace-Driven 시뮬레이션 방법을 사용하여 구현되었다. 구현된 설계공간 탐색도구를 이용하여, 시스템 설계자는 설계하고자하는 시스템에서 적용 가능한 하드웨어 소프트웨어 분할을 설계변수를 변화하여가며 체계적으로 살펴볼 수 있다. 각각의 경우에 선택적으로 적용 가능한 설계변수들의 변화에 따라 성능 수치가 어떻게 변하는지, 시뮬레이션을 통하여 산출하여 시스템 설계자로 하여금 최소 비용으로 시간 및 자원 제약사항을 만족하는 시스템구조를 설계할 수 있도록 도움을 준다. 추가적으로 본 논문에서는 또한 시뮬레이션 속도를 향상시키기 위하여 작업량과 병렬성과의 관계에 기초하여 사상집합의 크기를 줄이는 휴리스틱을 제안한다. 제안된 사상집합 축소 휴리스틱을 적용한 시뮬레이션 결과 사상집합의 크기를 효율적으로 줄일 수 있었다.

이와 같이 본 논문을 통하여 구현된 재구성가능한 시뮬레이터를 이용한 설

계공간 탐색도구는 내장형 시스템의 설계과정에서 중요한 역할을 담당할 것으로 기대된다.