Hybrid 내장형 시스템의 설계공간탐색을 위한 시간분석 시뮬레이터의 설계 및 구현

안 성 용[†] · 심 재 홍[††] · 이 정 아[†††]

요 약

최근의 내장형 시스템은 유연성을 유지하고 시간 제약사항을 만족하기 위해서 일반적인 프로세서와 FPGA와 같은 재구성 가능한 부품을 결합하는 Hybrid 시스템을 사용하는 추세이다. 이러한 내장형 시스템은 구축하는 설계 시간을 단축하여 짧은 시간 안에 시장에 진입하는 것이 아주 중요하다. 새로이 주목받고 있는 연구분야인 설계공간탐색은 실제 시스템을 제작하지 않고도 시스템 수준에서 어플리케이션의 성능을 분석하여 최소의 비용으로 시스템에서 요구하는 제약사항을 만족하는 구조를 예측하는 것을 가능하게 한다. 본 논문에서는 Hybrid 내장형 시스템의 설계공간탐색을 위한 시간분석 시뮬레이터를 설계하고 구현하였다. 시스템 설계변수를 변화하면서 정량적인 성능 데이타를 이용하여 설계공간탐색을 가능하게 하는 Y-Chart 방법을 Hybrid 시스템의 경우에 적용하여 시뮬레이터를 확장 구현하였으며, 기존의 소프트웨어 시간 분석 도구 및 하드웨어 시간분석도구를 활용한다. 본 논문에서 제시하는 시간분석 시뮬레이터는 Hybrid 내장형 시스템의 설계 비용과 시간을 현저하게 줄이면서, 최적의 하드웨어 구성을 찾는 설계공간탐색의 핵심 모듈로 활용될 것으로 기대된다.

A Design and Implementation of a Timing Analysis Simulator for a Design Space Exploration on a Hybrid Embedded System

Seong-Yong Ahn[†] · Jea-Hong Shim[††] · Jeong-A Lee[†††]

ABSTRACT

Modern embedded system employs a hybrid architecture which contains a general micro processor and reconfigurable devices such as FPGAs to retain flexibility and to meet timing constraints. It is a hard and important problem for embedded system designers to explore and find a right system configuration, which is known as design space exploration (DSE). With DSE, it is possible to predict a final system configuration during the design phase before physical implementation. In this paper, we implement a timing analysis simulator for a DSE on a hybrid embedded system. The simulator, integrating exiting timing analysis tools for hardware and software, is designed by extending Y-chart approach, which allows quantitative performance analysis by varying design parameters. This timing analysis simulator is expected to reduce design time and costs and be used as a core module of a DSE for a hybrid embedded system.

키워드 : 시스템 수준 설계(System Level Design), 재구성 가능 시스템(Reconfigurable system), 시간분석(Timing Analysis), 내장형 시스템(Embedded System)

1. 서 론

전통적으로 특정 응용 시스템을 구현하는데 크게 두 가지 방법이 사용되고 있다. 첫 번째는 하드웨어에서 특정연산을 수행하기 위해서 ASIC(Application Specific Integrated Circuit)과 같은 hardwired 기술을 이용하는 것이다. ASIC은 주어진 특정 연산을 수행할 수 있도록 특별하게 제작되기 때문에 설계하고자 하는 기능에 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. 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 구조를 가지는 반면에 성능 면에서는 그리 뛰어나지 못하다. 마이크로 프로세서들은 하나의 명령어를 수행시키기 위해서 명령어를 메모

※ 이 논문은 1999년도 조선대학교 학술연구비의 지원을 받아 연구되었음.
† 준 회 원 : 조선대학교 대학원 전자계산학과
†† 정 회 원 : 조선대학교 인터넷소프트웨어공학부 교수
††† 정 회 원 : 조선대학교 컴퓨터공학부 교수
논문접수 : 2002년 9월 30일, 심사완료 : 2002년 11월 29일

리로부터 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. 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 계산요소들과 라우팅 자원들을 이용하여 하드웨어에 재구성되어 구현될 수 있다[5, 6]. 재구성 가능한 컴퓨팅은 특히 계산량이 많은 암호화 알고리즘, 이미지 처리, 패턴 매칭, 데이터 압축과 같은 어플리케이션들에 적용하여 10~100배 가량의 성능향상을 보여주는 많은 연구 결과들이 제시되고 있다[4].

소프트웨어의 성능 향상을 위해서 현재의 내장형 시스템은 일반적으로 FPGA와 같은 재구성 가능한 장치와 일반적인 마이크로 프로세서를 결합하여 설계되는데, 이러한 시스템을 Hybrid 시스템이라고 한다. Hybrid 시스템에서 프로세서는 재구성 가능한 장치를 사용했을 때 효율적이지 못한 데이터 종속적인(Data-dependent) 제어나 메모리 접근과 같은 기능을 수행하게 되고, 많은 양의 계산이 필요한 기능은 재구성 가능한 장치가 처리하게 된다.

Hybrid 시스템을 사용함으로써 설계의 유연성이 증대되는 효과를 얻을 수 있지만, 설계상의 구현조건과 제약조건을 만족하는 구조를 찾아야 하는 문제가 여전히 어려운 문제로 남아있다. 이러한 문제를 해결하기 위하여 시스템 설계의 초기 단계에서 주어진 구현조건 및 제약조건을 만족하는 다양한 구조를 살펴보고, 이에 따른 최종결과물의 성능을 신뢰성 있는 정확도를 가지고 비교, 평가, 예측할 수 있는 시스템 개발 환경의 필요성이 대두되었다. Hybrid 시스템 설계환경에서 중요한 요소들은 하드웨어 소프트웨어 통합 시뮬레이션 관련 기술과 시스템 개발 제약 조건을 만족시키는 최적의 설계 변수를 찾을 수 있는 설계 공간 탐색(Design Space Exploration) 기술이다. 특히 설계 공간 탐색을 위해 성능 분석을 위한 통합 시뮬레이션과정은 필수적이다. 설계공간 탐색은 적용 가능한 설계변수에 따라 다양한 형태의 시뮬레이션을 반복적으로 수행할 수 있어야 하고, 여러 소프트웨어들을 다양한 하드웨어 플랫폼에 적용 가능하도록 시뮬레이터 자체도 목표시스템에 무관한 재사용 가능한(Retargetable) 구조이어야 한다. 추가적으로 특정 어플리케이션이 동작 중에 발생할 수 있는 자원 할당 및 충돌문제를 고려할 수 있어야 한다.

내장형 소프트웨어의 경우에는 이의 성능을 예측하기 위해 많은 연구들이 진행되고 있다. 이러한 연구들 중에서 명령어 수준에서 성능을 분석하는 경우는 매우 정확한 성능 예측을 가능하게 하여 마이크로 프로세서만을 이용하여 설계하는 경우에 특정 어플리케이션의 성능의 예측이 가능하다[7, 9]. 다중 처리가 가능한 내장형 시스템의 성능 분석을 위한 연구는 여러 개의 작업들이 각각 다른 프로세서에서 분산 처리되는 경우에 대하여도 성능분석을 가능하게 한다[8]. 그러나, 이와 같은 내장형 소프트웨어 성능분석에 관한 기존의 연구들은 실제 어플리케이션 실행 중에 성능에 큰 영향을 미칠 수 있는 컴퓨팅 자원 스케줄링 문제와, Hybrid 시스템 설계 시 선택적으로 적용 가능한 설계변수가 변화에 따른 시뮬레이션이 불가능하기 때문에 Hybrid 시스템의 설계공간 탐색을 위한 기본구조로 적합하지 못하다.

본 논문에서는 Hybrid 시스템에서 동작하는 내장형 소프트웨어의 수행 시간을 분석하는 새로운 구조를 제시한다. 수행 시간을 분석하기 위한 구조는 소프트웨어 수행 시간 분석과 하드웨어 수행시간 분석을 위한 도구를 목표시스템에 무관한 재사용 가능한(Retargetable) 시뮬레이터에 결합하는 구조를 가지고 있다. 제안된 구조는 소프트웨어 수행시간 분석을 위해서는 Princeton University에서 개발된 실시간 내장형 소프트웨어 분석도구인 "Cinderella"를 사용하고, 하드웨어 수행시간 분석을 위해서는 Xilinix사의 Foundation 시리즈[11]와 같은 벤더에서 제공하는 도구를 활용한다. 그리고, TU. Delft대학에서 단일형 하드웨어들만으로 구성된 시스템의 설계 공간 탐색을 위하여 개발한, 시스템의 성능의 정량적인 분석을 활용하는 Y-Chart 개념을 Hybrid System의 경우에 적용 확장하여 재구성 가능한 시뮬레이터를 구현하였다. 제안된 구조는 컴퓨팅 자원 스케줄링문제를 고려하기 위해서 Trace-driven 시뮬레이션 방법을 사용하고 각각의 하위 작업들에 대한 사상의 경우(Partitioning case)와 선택적으로 적용 가능한 설계변수들을 고려한 시뮬레이션이 가능하도록 구현되었다.

본 논문의 구성은 다음과 같다. 서론에 이어 2장에서는 소프트웨어 수행시간 분석 도구인 "Cinderella"와 Y-Chart 성능분석 방법에 대하여 간략히 설명한다. 3장에서는 구현된 목표시스템에 무관한 재사용 가능한(Retagetable) 시뮬레이터에 대하여 설명하고, 4장에서는 이를 이용한 수행시간 분석 과정을 설명한다. 5장에서는 실험환경과 실험결과를 제시하고, 6장에서 결론을 도출한다.

2. 수행 시간 분석을 위한 기반 기술

Hybrid 시스템에서 동작하는 내장형 소프트웨어의 시간 분석을 위해서는 먼저 해당 어플리케이션을 하위 작업으로

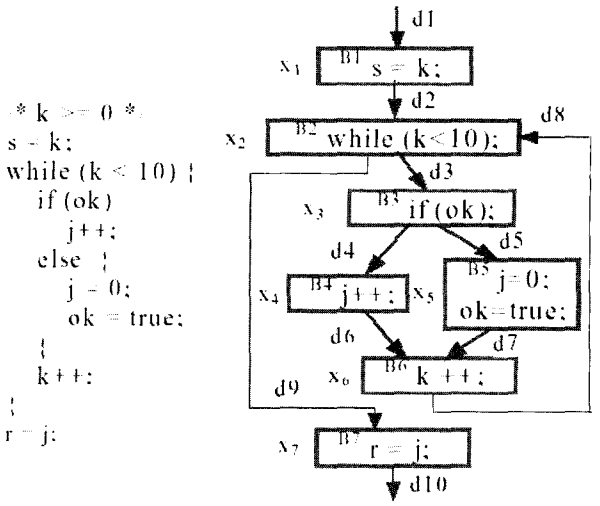
분할하여야 한다. 하위 작업들은 마이크로 프로세서에서 처리되는 경우와 재구성 가능한 장치에서 처리되는 경우에 대하여 수행 시간을 분석하는 과정이 필요하다. 본 논문에서는 이러한 과정을 위해서 마이크로 프로세서에서 처리되는 수행시간 분석을 위해서 'Cinderella'를 사용한다. 재구성 가능한 장치에서 처리되는 수행 시간 분석을 위해서는 재구성 가능한 장치에 맞게 동작하는 상용 하드웨어 수행 시간 분석 도구를 사용할 수 있다(예, Xilinx Foundation). 이렇게 각 하위 작업들 별로 산출된 시간 분석 정보를 기초로 시뮬레이션 과정을 수행하기 위한 도구는 Y-Chart의 개본 개념을 응용한 재구성 가능한 시뮬레이터를 이용한다. 본 장에서는 수행시간 분석을 위한 기반 기술인 'Cinderella'와 'Y-Chart'에 대해서 간략히 설명한다.

2.1 소프트웨어 성능 분석도구(Cinderella)

'Cinderella'는 ILP(Integer Linear Programming)기반으로 되어있는 내장형 소프트웨어 성능분석도구이다[10]. 이 도구는 WCET(Worst Case Execution Time)를 구하는 문제를 해결하기 위해서 두 부분으로 구성되어 있다. 첫 번째는 프로그램 흐름을 분석하는 것이고 두 번째는 마이크로 아키텍처를 모델링하는 것이다.

프로그램 흐름 분석은 구조적 제약(Structural Constraints)과 기능적 제약(Functionality Constraints)이라고 정의되는 두 가지 형태의 선형 제약(Linear Constraints)을 이용하여 프로그램 흐름을 표현한다(그림 1). (그림 1)에서 각각의 노드와 지시선에 붙어있는 레이블은 해당되는 노드와 지시선의 실행/흐름(Execution/Flow) 빈도(Count)를 나타낸다.

구조적 제약은 각각의 노드들에 대한 흐름 제약을 기반으로 구성되어지는 CFG(Control Flow Graph)로부터 자동적으로 추출되어질 수 있다. 기본 블록(Basic Block)의 수행 빈도(Execution Count)는 제어가 해당 노드로 진입한 횟수와 같고, 이것은 또한 제어가 그 노드에서 벗어난 횟수와 같다. (그림 1)에서의 예를 들면, $x_1 = d_1 = d_2$ 라는 의미이다. 기능적 제약은 데이터 흐름(Data Flow) 분석을 수행하거나, 사용자가 직접 제약사항을 정의함으로써 얻어질 수 있다. 기능적 제약은 두 가지 형태 논리적인 흐름(Logical Flow) 정보를 제공해준다. 첫 번째는 프로그램 내에 존재하는 순환 영역(Loop Bounds)을 결정한다. 예를 들어 (그림 1)의 소스 코드를 보면, 매번 while 루프에 들어갈 때마다 루프 내의 코드는 최소 0번, 최대 10번 수행되도록 되어 있다. 이러한 정보는 $0x_1 \le x_3 \le 10x_1$과 같이 표현될 수 있다. 두 번째 형태는 분석결과를 좀더 정확하게 산출할 수 있는 또 다른 패스 정보를 정의한다. 예를 들어, (그림 1)에서 기본 블록 B_4와 B_5의 상관관계를 관찰해보면, else 문장은 루프안에서 최대 1번만 수행되도록 되어있다. 이러한 정보는 $x_5 \le 1x_1$과 같이 표현될 수 있다.



(a) Code (b) Control flow graph

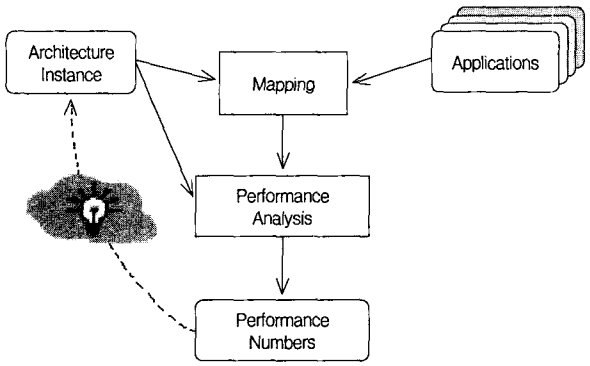
(그림 1) 프로그램 코드의 예와 CFG

임의의 프로그램의 기본 블록 B_i의 수행시간이 상수 C_i라하고, X_i가 해당 기본 블록의 실행 빈도라고 한다면, 이 프로그램의 전체 수행 시간은 $\sum C_i X_i$와 같이 표현될 수 있다. 위에서 설명한 바와 같이 X_i와 관련된 모든 제약사항은 정수형 선형 공식들(Integer Linear Formulas)로 표현되기 때문에, 위에서 추출된 제약들을 기반으로 ILP 기술을 이용하여 전체 수행시간을 계산하는 함수(Cost Function)를 최대화함으로써 WCET 문제를 풀 수 있다.

최근의 마이크로 컴퓨터의 구조에서 명령어의 수행시간은 피 연산자 값(Operand Value)과 시스템의 상태(Machine State)와 같은 요소들에 영향을 많이 받는다. 마이크로 아키텍처 모델링은 프로그램이 수행 중에 동적으로 발생하는 이와 같은 영향을 고려하기 위해서 사용된다. 예를 들어, 알려져 있는 명령어들을 순차적으로 처리할 때 파이프라인 구조를 사용할 수 있는데, 이러한 구조는 기본블록의 실행시간에 영향을 미친다. 추가적으로 캐쉬메모리의 상태도 마찬가지로 중요한 부분이라고 할 수 있다. 'Cinderella'에서는 파이프라인구조와 캐쉬메모리의 상태 모델링을 마이크로 아키텍처 모델링에 추가하여 수행시간 분석의 정확도를 높였다.

2.2 Y-Chart 방법

단일형 하드웨어 구조를 전제로 응용프로그램들을 하드웨어 구조에 맵핑하여 예상 성능 수치를 측정하고 분석하는 체계적인 방법론으로 개발된 것이 Y-chart 설계방법이다(그림 1). 이 설계방법의 효용성은 Y-Chart 설계환경의 제안자인 Delft 대학의 Bart Kienhuis와 Ed Deprettere교수 연구팀의 TV 응용의 비디오 프로세싱에 관련된 시스템 수준의 시뮬레이션에 관련된 결과에서 잘 나타나 있다[3].

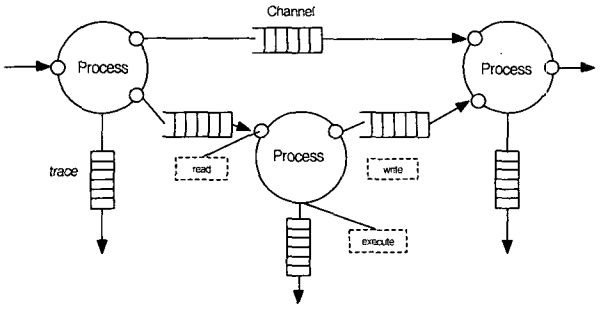


(그림 2) Y-chart방법

(그림 2)에서 보여진 봐와 같이 Y-Chart 방법에서는 다양한 어플리케이션이 특정 아키텍처에서 수행될 경우의 성능을 분석할 수 있는 구조로 되어있다. 또한 아키텍처 인스턴스에 적용 가능한 설계변수를 변화시켜가며 각각의 성능을 비교 분석할 수 있는 구조로 되어있다.

3. 재구성 가능한 시뮬레이터의 구현

본 논문에서 구현된 시뮬레이터는 Hybrid 시스템의 설계과정에서 실제 프로토타입을 구축하지 않고 구현될 시스템의 성능을 예측할 수 있도록, 기존의 Y-chart 설계환경을 보완하여 개발하였다. 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. 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 하드웨어자원에 할당하는 Trace-Driven 시뮬레이션 방법을 사용하여 구현되었다.



(그림 3) 어플리케이션 모델(Kahn Process Network)

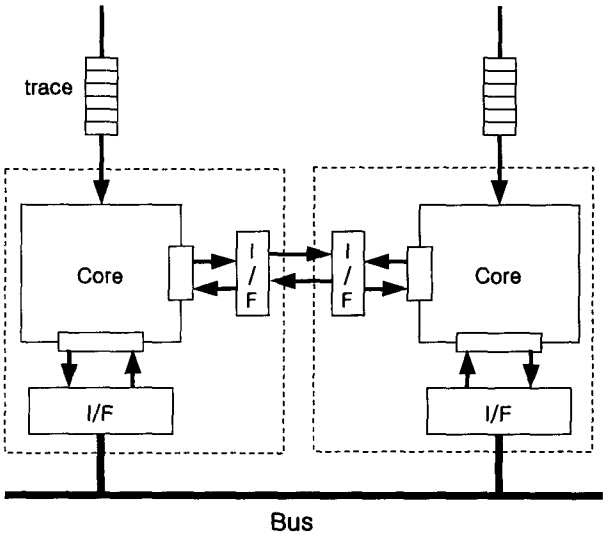
(그림 3)은 응용 시뮬레이터를 구현하기 위한 어플리케이션 모델의 예를 보여준다. 어플리케이션 모델은 기본적으로 blocking-read, non-blocking-write의 특성을 가지는 프로세스들로 구성되는 Kahn 프로세스 네트워크를 기반으로 설계되었다. 각각의 프로세스는 입력포트와 출력포트를 가지며 입력포트를 통하여 패킷을 소모하고 출력포트를 통하여 패킷을 생산한다.

각각의 프로세스는 실행조건을 가지며 이 실행조건에 만족할 경우에만 프로세스가 활성화된다. 프로세스들간의 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.

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

위의 예에서는 프로세스가 활성화되기 위해서 세 가지 조건들(R1, R2, R3)중의 하나만 만족하면 되고, 각각의 조건들은 해당되는 입력포트에 조건집합에 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.

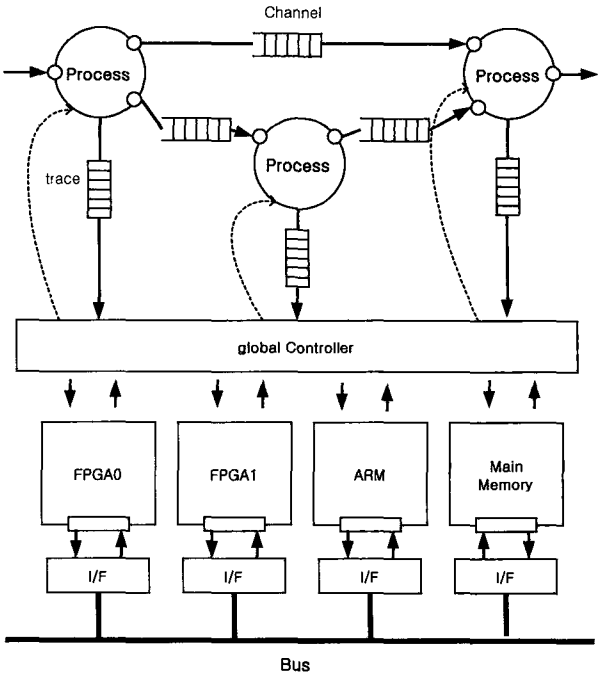
(그림 4)는 하드웨어 시뮬레이터를 위한 하드웨어 모델의 예를 보여준다.



(그림 4) 하드웨어 모델

하드웨어 모델은 사상 제어기에서 할당된 패킷들을 실제로 실행하는 과정을 시뮬레이션 한다. 실제로 시뮬레이션 상에서는 수행시간만을 고려한다. (그림 4)에서 코어 부분은 특정기능을 수행하는 CPU나 FPGA를 의미하고 CPU는 한순간에 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 버스구조 내의 버퍼에 저장되게 되고 사용 가능한 버스가 있을 때까지 대기한다.

(그림 5)에서는 응용 시뮬레이터와 구조시뮬레이터의 구조를 보여줌으로써 구현된 시뮬레이터의 전체적인 구조를 보여준다. (그림 5)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 되돌려 주고 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.



(그림 5) 재구성 가능한 시뮬레이터의 구조

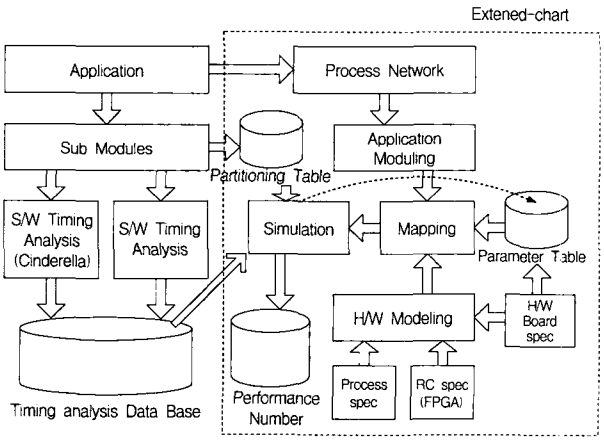
하드웨어 구조 시뮬레이터는 데이터 단위의 처리를 시뮬레이션하며 각 데이터 단위에 대하여 처리하는데 얼마의 시간이 걸렸는지를 계산한다. 프로세스에서 처리를 요구한 자료 단위가 사상 제어기를 통해서 어떤 하드웨어 구성 요소의 기능을 사용해야 하는지가 정해져서 전달되면 이를 받아서 기능 요소에서 처리하는 시간동안 이 기능 요소는 다른 처리 요청을 받지 않도록 한다. 또한 기능 요소에 배제가 설정되어 있을 경우에는 해당 하드웨어 구성 요소가 처리하고 있는 처리 요구가 모두 끝난 다음에 하드웨어 구성 요소를 점유하도록 한다. 이렇게 하여 여러 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.

사상 제어기는 시뮬레이션 수행 전에 하드웨어 요소와

각 하드웨어 요소가 수행할 수 있는 기능 요소에 대한 정보, 그리고 응용 프로그램의 하위 작업들이 요청하게 될 처리 요구에 대한 정보를 이용하여 어떤 하위 작업을 어떤 하드웨어 요소의 기능 요소에 사상할 수 있는지를 판단한다. 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 하위작업의 자원요구가 있을 때 해당되는 자원이 현재 사용중인가를 확인하고 사용중이면 대기 버퍼에 그대로 남겨두고 요구하는 자원이 사용 가능한 하위 작업만 해당되는 자원에 할당하게 된다. 이러한 방법을 기반으로 사상 제어기에서는 실시간 운영체제(Real-time operating system)에서 적용 가능한 간단한 스케줄링 알고리즘을 적용하여 하드웨어 자원 할당에 관한 시뮬레이션이 가능하도록 구현되었다.

4. 수행 시간분석

(그림 6)은 Hybrid 시스템을 위한 시간 분석을 위한 흐름을 나타낸 것이다. 전체적으로 어플리케이션의 하위 작업별 성능 분석하는 부분과 확장된 Y-chart를 이용한 성능 분석을 위한 시뮬레이션 부분으로 나누어진다.



(그림 6) 시간 분석을 위한 흐름도

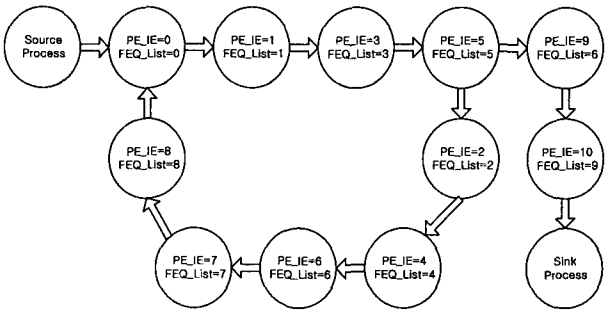
먼저 특정 어플리케이션 하위 작업별로 나누고 각각에 대해서 수행시간 분석을 수행한다. 2장에서 설명한 봐와 같이 소프트웨어 수행시간분석을 위해서는 "Cinderella"를 사용하고, 하드웨어 수행시간 분석을 위해서는 Xilinx Foundation 시리즈와 같은 상용 도구를 활용한다. 이러한 방법으로 분석된 각각의 분석결과는 데이터 베이스 형태로 저장되게 되고 추후에 시뮬레이션을 위한 값으로 사용되게 된다. 하드웨어 모델은 사용되는 프로세스와 FPGA와 같은 재구성 가능한 장치들의 특성과 버스구조의 특성을 고려하여 모델링된다. Hybrid 시스템을 위한 기본적인 구조는 3장에서 설명한 바와 같이 목표시스템에 무관한 재사용 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.

$$Parallelism = \frac{Total\ Execution\ Time\ of\ the\ Workloads}{T\_end}$$
$$Utilization = \frac{Time\ a\ Resoure\ is\ Used}{T\_end} \times 100\% \quad (1)$$

성능수치로 사용되는 병렬성(Parallelism), 하드웨어자원들의 활용도(Utilization)는 식 (1)에 의해서 구해진다. 식 (1)에서 T_end는 패킷처리 시뮬레이션 종료시간을 의미한다. 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.

5. 실 험

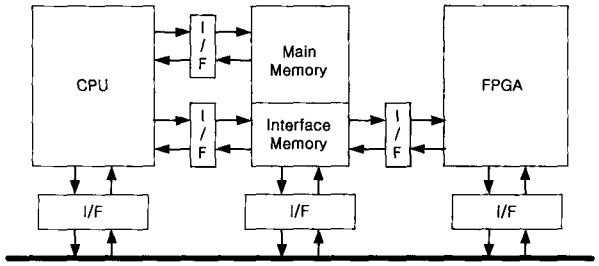
4장에서 설명된 수행 시간 분석을 위한 환경을 이용한 시뮬레이션을 수행하기 위하여 어플리케이션모델로는 H.263 Encoder을 참조하여 작성된 모델을 선택하였고, 하나의 마이크로 프로세서와 하나의 재구성가능한 장치 결합된 가상의 Hybrid 시스템을 하드웨어 모델로 가정하였다. (그림 7)은 실험에 사용된 어플리케이션 모델을 Kahn Process Network 형태로 표현한 것이다.



(그림 7) 실험에 사용된 어플리케이션 모델의 Kahn Process Network 표현

(그림 8)은 시뮬레이션에서 가정되는 하드웨어 모델을 보여준다. (그림 8)에서 보여지는 하드웨어 모델은 CPU, FPGA,

메모리 장치 그리고 이들간의 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 8) 실험에 적용된 하드웨어 모델

〈표 1〉 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

<표 1>은 (그림 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 2>는 (그림 7)의 어플리케이션 모델이 (그림 8)의 하드웨어모델에서 동작되었을 때 가능한 모든 하드웨어 소프트웨어 분할 경우에 대한 시뮬레이션을 진행하여 얻어진 결과 중 우수한 성능을 보여주는 경우를 선택하여 보여준다. 아래 표에서 분할 경우에 보여지는 숫자는 <표 1>에서 보여지는 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 스케줄링방법 : FCFS(First Come First Serve)

6. 결 론

본 논문에서는 FPGA와 같은 재구성 가능한 장치와 일반적인 CPU을 결합하여 구성되는 Hybrid 내장형 시스템의 설계공간탐색을 위한 시뮬레이터를 설계하고 구현하였다. 구현된 도구에는 소프트웨어 수행 시간 및 하드웨어 수행 시간 분석을 위한 도구들이 목표 시스템에 무관한 재사용 가능한(Retargetable) 시뮬레이터에 통합되어 있다. 이러한 (Retargetable) 시뮬레이터는 Y-Chart 설계 환경의 기본 개

〈표 2〉 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

념을 Hybrid System에 적용할 수 있도록 확장하여 구현하였다. 시뮬레이터는 컴퓨팅 자원 스케줄링문제를 고려하기 위해서 Trace-driven 시뮬레이션 방법을 사용하고 각각의 하위 작업들에 대한 사상의 경우(Partitioning case)와 선택적으로 적용 가능한 설계변수들을 고려한 시뮬레이션이 가능하도록 구현되었다. 본 논문에서 구현된 시뮬레이터는 Hybrid 내장형 시스템을 설계하는 과정에서 실제 프로토타입을 개발하지 않고도 개략적인 성능을 미리 예측할 수 있도록 하여 설계 비용과 시간을 현저하게 줄여주고, 최적의 하드웨어 구성을 검색하게 해주는 설계공간탐색의 핵심 모듈로 활용될 것으로 기대된다.

참 고 문 헌

[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] 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, pp.338-349, 1997.

[3] G. Kahn, "The semantics of a simple language for parallel programming," Info. Proc., Stockholm, pp.471-475, Aug., 1974.

[4] J. Vullemin, P. Bertin, D. Roncin, M. Shand, H. Touati, P. Boucard, "Programmable Active Memories : Reconfigurable Systems Come of Age," IEEE Transactions on VLSI Systems, Vol.4, No.1, pp.56-69, March, 1996.

[5] Andre DeHon, "The Density Advantage of Configurable Computing," IEEE Computer, 33(4), pp.41-49, April, 2000.

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

[7] 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.

[8] Ti-Yen Yen, Wayne Wolf, "Performance Estimation for Real-Time Distributed Embedded Systems," IEEE Transactions on Parallel and Distributed Systems, Vol.9, November, 1998.

[9] George Hadjiyiannis, Pierro Russo, Srinivas Devadas, "A Methodology for Accurate Performance Evaluation in Architecture Exploration," Design Automation Conference, 1999.

[10] Yau-Tsun Steven Li, Sharad Malik, "Performance analysis of Real-Time Embedded Software," Kluwer Academic Publishers, 1999.

[11] http://www.xilinx.com.

안 성 용

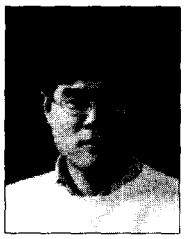
e-mail : dis@chosun.ac.kr

1996년 조선대학교 전자계산학과(이학사)

1998년 조선대학교 전자계산학과 (이학석사)

1999년~현재 조선대학교 전자계산학과 박사과정 재학중

관심분야 : 재구성 가능한 시스템, 하드웨어 소프트웨어 통합설계, 내장형 시스템



심 재 홍

e-mail : jhshim@chosun.ac.kr

1987년 서울대학교 자연과학대학 전산과학과(이학사)

1989년 아주대학교 공과대학 컴퓨터공학과 (공학석사)

2001년 아주대학교 정보통신대학 컴퓨터공학과(공학박사)

1987년~1994년 서울시스템(주) 공학연구소

1999년~2000년 University of Arizona 객원연구원

2001년~2001년 아주대학교 정보통신전문대학원 BK21 전임연구원

2001년~현재 조선대학교 인터넷소프트웨어공학부 전임강사

관심분야 : 운영체제, 실시간 및 멀티미디어 시스템, 내장형시스템

이 정 아

e-mail : jalee@chosun.ac.kr

1982년 서울대학교 공과대학 컴퓨터공학과 (공학사)

1985년 미국 인디애나 주립대학교 컴퓨터학과(공학석사)

1990년 미국 가주주립대학(UCLA) 컴퓨터공학과(공학박사)

1990년~1995년 Assistant Professor, University of Houston USA

2000년~2002년 Stanford University 교환교수

1995년~현재 조선대학교 컴퓨터공학부 교수

관심분야 : 재구성가능한 시스템, 컴퓨터 연산, 컴퓨터 시스템