특집

하드웨어-소프트웨어 통합 설계 기술

신 현 철
한양대학교 전자공학과

I. 서 론

최근 컴퓨터와 제어 및 통신 기기를 포함하는 대부분의 전자 시스템은 주문형 반도체(ASIC)나 표준부품(off-the-shelf component)과 같은 하드웨어 부분과 운영체계나 응용 프로그램과 같은 마이크로프로세서(microprocessor)에서 수행되는 소프트웨어 부분이 통합되어 복합적으로 구성된다. 기존의 설계에서는 하드웨어와 소프트웨어 부분을 설계 초기부터 분리하여 각각을 독립적으로 설계하였으나, 고밀도 고성능의 복잡한 시스템을 효과적으로 설계하기 위해서는 하드웨어와 소프트웨어가 혼합된 시스템을 체계적으로 설계하는 기술의 개발이 필요하다. 체계적인 하드웨어와 소프트웨어의 통합 설계(hardware-software codesign) 시스템은 각 설계 계층에서의 시스템 사양의 표현, 시뮬레이션, 하드웨어와 소프트웨어 부분으로의 분할, 목적에 적합한 회로의 합성, 이들의 적절한 접속(interface) 등으로 구성된다. 따라서 하드웨어와 소프트웨어가 혼합된 고밀도 시스템을 단기간 내에 경제적으로 신뢰도 높게 설계하는 것은 파급효과가 큰 중요한 문제이다.

근래에 마이크로프로세서의 성능은 상당이 빠른 속도로 발전하여 단위 시간에 수행할 수 있는 연산의 수가 크게 증가하였다. 이와 같은 마이크로프로세서의 성능 향상에도 불구하고 시스템의 고성능화로 인하여 정보처리 기기, 통신 기기, 실시간 시스템 등 많은 전자 시스템은 특정한 연산을 보다 빠르게 수행할 수 있는 하드웨어를 마이크로프로세서와 함께 사용하고 있다. 가능하다면 주어진 기능을 소프트웨어로 마이크로프로세서에서 수행하는 것이 경제적이다. 일반적으로 가격에 대한 성능을 비교한다면, 마이크로프로세서가 ASIC에 비하여 저가이기 때문이다. 따라서 통합 설계는 하드웨어가 소프트웨어에 비하여 고가이며 고성능이라는 가정 하에서 이루어진다.

여기에서는 통합설계에 관한 기술해설과 전반적인 내용을 설명함으로써 통합설계 기술에 관한 소개를 하고자 한다. 제 2장에서는 하드웨어-소프트

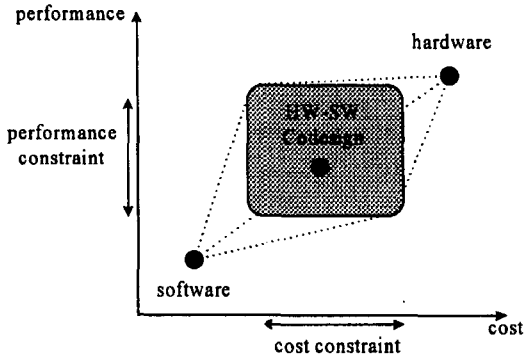
웨어 통합설계의 전반적인 내용으로, 필요성, 기본 구조, 전체적인 흐름, 하드웨어-소프트웨어 통신등에 대해서 설명한다. 제 3장에서는 통합설계의 주요 세부과제를 설명한다. 제 4장에서는 효율적인 시스템 설계를 위하여 중요한 설계 재사용에 대해서 기술한다. 끝으로 제 5장에서 결론을 맺는다.

II. 하드웨어-소프트웨어 통합설계

1. 통합설계의 필요성 및 기본 구조

최근들어 가전 제품, 통신 시스템, 차량 제어 시스템 등과 같은 내장형 시스템의 응용분야가 다양해지고, 복잡도도 증가하고 있다. 또한 현재의 마이크로프로세서의 계산 성능은 설계자가 쉽게 이들을 효과적으로 이용하여 최적 설계할 수 있는 한계를 넘어섰다. 전통적인 시스템 설계방법으로는 8비트의 간단한 마이크로프로세서 정도의 이용만이 효과적이며, 32비트 또는 64비트 마이크로프로세서가 제공할 수 있는 성능을 충분히 사용하기 어렵다. 또한, 경쟁력 확보를 위하여 내장형 시스템을 개발하는데 소요되는 비용과 시간을 최소화해야 하기 때문에 통합설계에 대한 관심이 점점 늘어나고 있다.

그림 1은 통합설계 공간(codesign space)을 보여준다. 즉, 시스템의 모든 기능을 소프트웨어로 구현할 경우 대량생산되는 고성능의 마이크로프로세서에서 프로그램을 수행할 수 있으므로 비용이 적게 들지만, 오퍼레이션들을 순차적으로 실행해야하기 때문에 성능이 떨어지게 된다. 반면에, 하드웨어로 모든 기능을 구현할 경우에는 병렬처리가 가능하여 수행 시간을 빠르게 할 수 있어서 성능 요구조건을 만족시키기 쉬우나, 하나 혹은 그 이상의 ASIC 또는 FPGA 등이 필요하게 되어 비용이 많이 드는 단점이 있다. 따라서 통합설계를 이용하여 시스템을 하드웨어와 소프트웨어의 복합적인 요소들로 구현할 경우 하드웨어의 성능상의 장점과 소프트웨어의 비용면에서의 장점이 고려된, 적절한 성능과 비용 제약 조건을 만족하는 (그늘



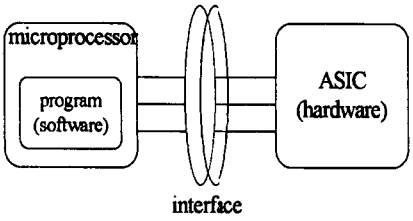
〈그림 1〉 통합설계 공간 (Codesign space)

부분) 효율적인 시스템을 구현할 수 있다.

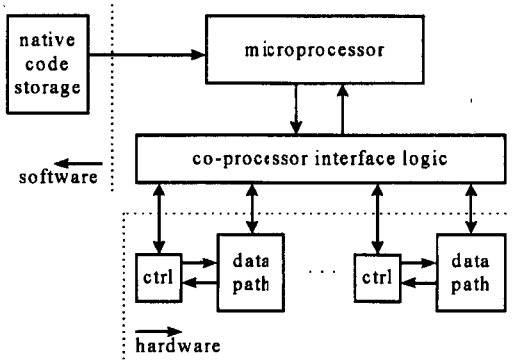
통합설계는 이미 오래전부터 embedded system과 같이 하드웨어와 소프트웨어로 구성된 복합적인 디지탈 시스템 설계에 응용되어 왔다. 그러나 이런 전통적인 설계 방식에서는 시스템 설계의 시작 단계에서부터 하드웨어와 소프트웨어를 완전히 분리하여 개별적으로 설계 및 구현을 했기 때문에, 최종적인 시스템 통합전까지는 이들 설계 작업간의 상호작용이 거의 없다. 그러므로 그림 1과 같은 하드웨어와 소프트웨어간의 trade-off가 전혀 이루어지지 않으며, 시스템 통합에 상당한 시간과 노력이 소요된다. 그리고, 시스템 통합 결과 성능과 비용에 대한 요구조건을 만족하지 못할 경우 재설계 및 재구현을 해야하므로, 추가적인 설계비용과 시간이 소요되는 경우가 많다.

이에 반해 통합설계를 이용하여 디지탈 시스템을 설계할 경우, 주문형 하드웨어의 성능과 소프트웨어의 저렴성 사이의 균형을 맞추어 최적의 가격대 성능을 이끌어 낼 수 있으며, 시스템 합성 과정을 자동화함으로써 설계 기간(design time)이 줄고, 변경이 잦은 부분이나 개선의 여지가 많은 부분을 소프트웨어로 구현하면 시스템의 개량이나 유지보수가 용이해지는 장점이 있다.

그림 2에서는 특정한 목적을 위한 마이크로프로세서와 ASIC 등의 하드웨어, 그리고 이들의 인터페이스 회로 등으로 구성된 혼합시스템(mixed hardware-software system)의 간단한 예를 보여준다. 그림 3에는 여러 종류의 embedded 프로세



〈그림 2〉 혼합 시스템의 예



〈그림 3〉 복합 코프로세서들을 가진 하드웨어-소프트웨어 시스템

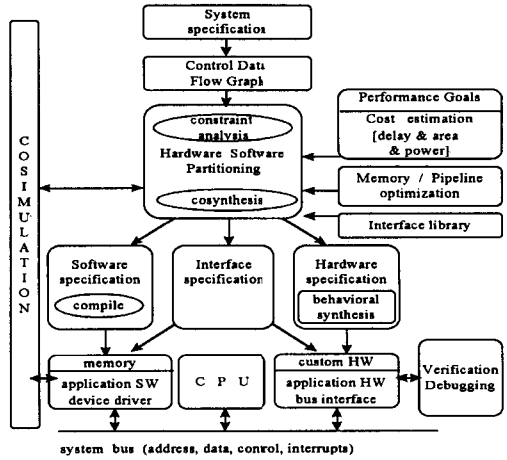
서들을 코프로세서(coprocessor)로 사용하는 좀 더 복잡한 통합설계의 한 예를 보여준다[1].

하드웨어-소프트웨어 분할에서 고려하여야 할 문제는 다음과 같이 여러 가지들이 있다. 즉, 성능을 떨어뜨리지 않고 하드웨어 비용을 줄이는 문제, 소프트웨어 중 성능을 가장 많이 증가시킬 수 있는 부분을 하드웨어로 옮기는 문제, 하드웨어-소프트웨어 구성요소 사이의 통신을 줄이는 문제, 병렬성을 최대한으로 증가시키는 문제, 성능과 비용간의 trade-off를 이용하는 문제 등 여러 가지 해결하여야 할 과제들이 있다.

전부터 많은 공학자들이 통합설계에 관한 관심을 가졌지만 실용화할 정도로 구체화되지 못했던 것은 통합설계 방법론을 뒷받침할 수 있을 정도로 CAD 기술이 발전되지 못했기 때문이었다. 그러나, CAD 기술이 감당할 수 있는 추상화 수준(abstraction level)이 도면 수준, 논리 수준, 상위 수준을 거쳐 거의 시스템 수준까지도 감당할 수 있는 정도로 발전함에 따라, 마이크로프로세서 설계, 디지탈 신호처리용 IC 설계, 그리고 embedded system 설계 등 여러 응용 분야에서 통합설계를 이용한 설계방법들이 개발되고 있다.

2. 통합설계의 전체 흐름

전체적인 하드웨어-소프트웨어 통합설계의 흐름은 그림 4와 같다. 시스템 사양은 C 나 C++, HardwareC, VHDL, Verilog, SpecChart 등의 기술언어로 기술될 수 있다. 시스템 사양은 그 시스템의 특성 및 사양이 적절히 기술된 Control Data Flow Graph (CDFG)나 Finite State Machine (FSM)으로 변환된 후, 각 오퍼레이션은 수행시간·면적·소모전력·메모리 등의 목적함수에 따라 하드웨어 또는 소프트웨어로 분할된다. 하드웨어 부분은 ASIC이나 FPGA등으로 합성 되며, 소프트웨어 부분은 주어진 마이크로프로세서에서 실행된다. 코시뮬레이션(co-simulation)은 혼합된 하드웨어-소프트웨어 시스템의 동작의 모형을 이용하여 설계 과정의 여러 단계에서 각각의 기능들을 검증한다.



〈그림 4〉 하드웨어-소프트웨어 통합 설계의 전체 흐름도

3. 하드웨어-소프트웨어 통신

통합설계에서 하나의 중요한 결정 사항은 하드웨어와 소프트웨어 간의 통신 방식을 결정하는 것

이다. 하드웨어-소프트웨어 간의 통신 방식은 전체 시스템의 성능, 비용 등에 큰 영향을 미치기 때문이다. 그러므로, 상위 단계에서 시스템의 특성에 적합한 통신 방식을 선택하여야 한다. 현재에 가장 많이 사용되는 통신 방식은 메모리를 공유하는 방식 (shared memory method)과 메시지 전달 방식 (message passing method)이 있다. 메모리를 공유하는 방식은 메모리를 여러 모듈에서 접근 가능하도록 설계하여, 하나의 모듈에서 전달할 데이터를 메모리의 일정한 위치에 저장하면 다른 모듈에서 데이터를 읽어가는 방식이다. 그러므로, 메모리를 공유하는 방식은 메모리에 데이터를 쓰고 읽는 것으로 데이터를 전송할 수 있으므로, 비교적 구현하기 간단한 방법이다. 그러나, 한정되어 있는 버스 수, 메모리의 포트 수 등으로 인하여 여러 개의 데이터가 전달되는 경우에는 성능이 떨어질 수 있다.

메시지 전달 방식은 더욱 일반적인 방식으로, 알맞은 프로토콜을 미리 정해 놓고 프로토콜에 맞게 데이터를 전달하는 방식이다. 그러므로, 이 방식을 사용하면 프로토콜에 따라서 데이터를 전송하기 위한 인터페이스 모듈들이 필요하다.

[2]에서는 통신 모듈의 종류를 다음과 같이 4가지로 구분하였다.

(1) 하나의 포트를 가지는 전역 기억장치(global memory)
(2) 지역 기억장치(local memory)와 하나의 포트를 가지는 전역 기억장치
(3) 지역 기억장치와 여러 개의 포트를 가지는 전역 기억장치
(4) 지역 기억장치와 버스를 이용한 인터페이스

이외에도, 통신 방식에는 인터럽트 (interrupt)에 의한 전송방식도 있고, 하나의 중앙 스케쥴러 (scheduler)에 의하여 순차적으로 통신이 수행되는 방식 등이 있다.

III. 통합설계의 세부과제

1. 시스템 사양 및 설계환경

시스템의 동작이나 기능, 제한조건들이 기술되는 시스템 사양은 하드웨어는 HardwareC, VHDL, Verilog, SpecChart 등으로, 소프트웨어는 C 또는 C++등으로 표현하고 있다. 하드웨어와 소프트웨어 간에 존재하는 특성의 차이로 인해 하나의 언어로 하드웨어와 소프트웨어가 공존하는 복합 시스템을 일관성있게 표현하는 데는 여전히 많은 어려움이 있다.

하나의 시스템을 하드웨어-소프트웨어 통합 설계 기술로 설계하기 위해서는 특정 언어로 기술된 시스템 사양을 통합설계 과정에 적합한 표현으로 기술할 필요가 있다. 기술 방식이 너무 추상적(abstract)일 경우에는 최종적으로 합성하기 위해 필요한 여러 가지 자세한 정보가 부족하기 때문에 합성에 어려움이 따르고, 기술 방식이 너무 상세할 경우 최종 구현의 의존도가 커지기 때문에 여러 가지 설계 대안을 고려하기 어렵다. 디지탈 시스템을 표현하는 대표적인 추상화 방법으로는 Control/Data Flow Graph (CDFG)나 Finite State Machine (FSM) 등이 있다. CDFG는 디지탈 시스템의 동작을 task (또는 operation)와 이들간의 data/control dependency로 표현하는 방식으로, 하드웨어나 소프트웨어 기술언어로부터 곧바로 얻어낼 수 있어, 지금까지 하드웨어 합성이나 소프트웨어 컴파일러에 많이 사용되어 왔다. FSM 모델은 시스템의 동작을 상태(state)와 상태간의 천이(transition)로 표현하는데, SpecChart와 같이 동시적(concurrent)이며 계층적인 FSM을 직접적으로 표현할 수 있는 언어로 시스템이 기술된 경우에는 FSM 내부표현을 얻기 쉬우나, 다른 언어로 표현된 경우에는 간접적인 과정을 거쳐 FSM 내부표현으로 유도해야 한다. FSM 모델은 초기 시스템 사양이 시스템 분할에 제약을 가한다는 단점이 있어 현재로는 CDFG 모델을 많이 사용하고 있다.

2. 시뮬레이션과 설계 검증

설계되는 시스템의 기능과 동작, 성능등의 검증은 어려운 문제이다. 검증하기위한 방법으로는 시뮬레이션과 formal verification이 있는데, 증명 방법을 이용한 formal verification은 완벽한 검증을 추구하는 이상적인 방법이다. 예를 들어 dead lock과 같은 상태를 체크하는데 유용하게 쓰인다. 그러나, 복잡도가 커서 아주 간단한 회로가 아니면 실제로 검증이 어렵다는 단점이 있다. 시뮬레이션은 주어진 입력 벡터에 의존하므로 완벽한 검증을 할 수 없다는 제약이 있음에도 불구하고 현재 사용하고 있는 설계 검증 방법 중에서 가장 효과적이고 널리 사용되는 방법이다.

시뮬레이션은 시스템의 표현을 입력으로 받아 사용자로 하여금 주어진 기술이 정확하게 동작하는 것인지를 확인할 수 있도록 해준다. 아울러 주어진 표현의 문법적, 의미적 오류를 찾아내는 기능을 부가적으로 제공하게 된다. 통합설계의 관점에서 시뮬레이션은 좀더 복잡한 의미를 지닌다. 하드웨어와 소프트웨어가 함께 설계되므로 인터페이스 등을 고려하여 여러 개의 혼합 모듈이 함께 시뮬레이션되어야 하기 때문에 일반적인 시뮬레이션보다 더 복잡하다.

기존의 설계 과정을 보면 소프트웨어와 하드웨어 부분이 따로 설계되고 합성되어 설계의 마지막 단계에 이르러서야 통합이 이루어지므로 여러 가지 비용·성능적으로 최적화가 어려웠다. 이러한 단점을 극복하려는 시도가 통합설계이므로 통합설계를 지원하는 시뮬레이션 또한 통합 시뮬레이션을 제공해야 한다. 이때, 통합시뮬레이터(cosimulator)는 하드웨어 모델과 소프트웨어 모델을 동시에 시뮬레이션하는 툴로 복합 시스템의 동작을 검증하는데 필수적인 시스템 설계 도구라고 할 수 있다.

기존의 하드웨어 시뮬레이터와 달리 통합시뮬레이터에서는 하드웨어와 소프트웨어가 서로 다른 기술언어로 표현될 경우, 이질적으로 표현된 이들 구성요소들의 모델에 관한 데이타와 시뮬레이션 수행시에 발생하는 데이타 등을 일관성있고 통일된 형태로 관리하여야 한다. 그리고, 병렬로 실행되는 여러개의 하드웨어 모듈과 순차적으로 실행되는 소프트웨어 모듈들이 공존하기 때문에 이들 사이의 상호작용을 고려하여 존재하는 복합 시스템을 정확하게 시뮬레이션하기 위해서는 복합 시스템이 실행될 때 발생하는 상호작용인 동기화와 프로세스 사이의 통신 (inter-process commun-ication) 등의 문제들도 잘 고려하여 해결해야 한다.

설계의 초기 단계에 통합 시뮬레이션이 가능하면 다양한 종류의 오류를 미리 발견·보완할 수 있다. 또한, 설계가 진행되어 구체화되면서 시스템의 동작 특성이 바뀌는지도 세밀히 분석해 볼 수 있으며, 주어진 시스템 사양을 시뮬레이션함으로써 사양이 설계자의 의도대로 표현되었는가를 검증할 수 있다. 통합설계된 시스템이 원래의 사양대로 동작하는지를 확인하고, 구현된 시스템이 주어진 사양의 기능을 수행하며 성능과 비용등의 제약조건을 만족하는가를 확인할 수 있다. 또한, 시스템 분할시 성능 추정을 위한 모델을 구하기 어려울 경우에는 시뮬레이션을 통해 성능을 추정해 볼 수 있다. 시스템이 복잡해질수록 분석적인 추정 모델을 구하기 어려워지므로, 시뮬레이션이 보다 더 적합한 성능 추정 방법이 될 수 있다. 그러나, 상이한 구조와 방식으로 기술되고 병렬적으로 동작하는 하드웨어 부분과 소프트웨어 부분을 함께 시뮬레이션 하는 것은 동기화 문제를 포함한 복잡한 문제들을 내포하고 있다.

3. 분할과 합성

하드웨어-소프트웨어 분할은 주어진 성능에 대한 제한 조건을 만족하는 범위안에서 최소의 비용으로 디지탈 시스템을 하드웨어 부분과 소프트웨어 부분으로 나누어주는 것이다[10]. 분할은 상위 레벨 (behavioral level)에서 수행되기 때문에 논리설계 등의 다른 설계과정에 비하여 최종적으로 구현한 시스템의 성능에 크게 영향을 미치게 된다. 그러나, 상위 단계에서 하위 단계 설계를 추정하여 최적화 되도록 분할하기가 어렵기 때문에 이에 대한 연구는 많이 이루어지지 않았고, 많은 통합 설계 시스템들이 전문가에 의한 수동 분할에 의존하고 있다. 그러나, 최근에 활발한 연구가 이루어지

고 있고, 일부 시스템에서 자동화되어 사용되어지는 추세이다.

쉽게 분할을 할 수 있는 경우도 있다. 예를 들면, OS call을 많이 사용하는 부분은 소프트웨어로 구현하고 시스템 사양의 변경이 잦은 부분은 소프트웨어나 프로그램 가능한 하드웨어(programmable hardware)로 구성하는 것이 유리하다. 또한 고속의 병렬처리 등을 요하는 ALU나 customized 메모리, 그리고 multiple threads of control등은 하드웨어로 구현하는 것이 좋다.

하드웨어-소프트웨어 분할에서의 비용계산은 하드웨어의 면적·지연시간·소프트웨어의 크기 및 수행시간·인터페이스 회로의 크기 및 지연시간 등 여러 가지 변수들에 가중치를 곱하여 합한 값으로 나타내어진다. 통합설계의 설계공간은 하드웨어나 소프트웨어만의 설계공간에 비해 훨씬 크고 불규칙적이므로, 이러한 설계공간을 탐색하기 위해서는 실제로 구현해보지 않고도 그 성능과 비용을 정확하게 추정하는 것이 필요하다. 일반적으로 추정의 정확도와 설계공간의 탐색 시간 사이에는 상반되는 관계가 있다. 즉, 추정이 정확할수록 탐색 시간이 많이 소요되므로 추정의 정확도와 탐색 시간의 절충이 필요하다.

통합설계에서는 추상화 수준(abstraction level)이 시스템 수준으로 매우 상위 단계이기때문에 추정을 행하는 수준과 추정의 대상이 되는 물리적 수준간의 간격이 매우 크므로 정확한 추정 모델을 구하기가 매우 어렵다. 현재로는 통합설계를 위한 추정 모델이 거의 없으며, 하드웨어의 상위 수준 합성을 위한 면적·지연 시간·전력 소모 등에 관한 추정 모델이 연구되었으나, 특정한 아키텍쳐나 설계방식에 국한되어 있다. 또한, 추정 모델의 정확도는 시스템 분할이 어느 단계에서 이루어지는가에 달려있다. 그러므로 시스템 분할 단계가 하위단계이며, 분할 단위(partitioning granularity)가 작을수록 보다 정확한 추정이 가능하다.

하드웨어-소프트웨어 분할 단위는 그 크기가 큰 순서대로 테스크(task), 펑션(function), 기본 블록(basic block), 오퍼레이션(operation, statement)으로 나눌 수 있다. 테스크나 펑션, 그리고 기본 블록 등의 상위 단계(coarse-grain)에서 분할을 수행할 경우, 분할이 간단해지고 소프트웨어 코드가 분산되는 것을 막을 수 있으나, 하드웨어 부분에서 리소스 쉐어링(resource sharing) 등을 고려하기 힘든 단점이 있다. 반면 오퍼레이션 단위의 하위 단계(fine-grain)에서 분할을 수행할 경우, 리소스 쉐어링을 고려하기가 용이하여 하드웨어의 비용을 줄여 최적화에는 유리하지만, 소프트웨어 코드가 잘게 분할되어 분산되며, 통신 시간(communication time), 면적, 그리고 소프트웨어 컴파일 효과 등을 추정하는데 어려움이 있다.

하드웨어-소프트웨어 분할에서 여러 가능한 해에 대한 전체 시스템의 성능과 비용을 정확하게 추정하는 것은 매우 중요하다. 왜냐하면 그 추정 결과에 따라 분할 결과가 크게 달라지기 때문이다. 좋은 분할결과를 얻기 위해서는 시스템의 비용·성능·전력 소모 등을 정확히 추정해야 하지만 하드웨어-소프트웨어 분할이 아직 하드웨어가 설계되기 전에 수행되고, 소프트웨어 컴파일러와 캐쉬, 메모리 구조 등에 따라 최종 합성 결과가 많이 달라지기 때문에 정확한 추정을 하기가 어렵다. 그러나, 하드웨어의 경우는 스케쥴링(scheduling) 및 리소스 할당(resource allocation) 등을 통해 성능과 비용을 분할 단계에서 추정할 수 있다.

대표적인 통합설계 시스템에 사용된 분할 비용함수들을 살펴보면 다음과 같다.

VULCAN 시스템 [3]은 하드웨어 지향적 설계방식으로서 HardwareC언어로 표현된 시스템의 하드웨어 모델로부터 CDFG를 추출하고 greedy 알고리듬을 사용하여 성능에 별 영향을 미치지 않는 noncritical operation들을 8086이나 R3000과 같은 표준형 프로세서로 옮겨 실행되도록 한다. 이렇게 함으로써 필요한 하드웨어의 양을 줄이게 된다. 이 때 한번 옮겨진 노드는 다시 다른 그룹으로 옮겨질 수 없다. 여기서 사용한 비용함수는 다음과 같다.

$$C = k_1 * SH + k_2 * B + k_3 * P^{-1} + k_4 * m$$

여기서 SH는 하드웨어 크기이며, B는 Bus bandwidth, P는 프로세서 이용율(utilization), m은 하드웨어와 소프트웨어간에 전송되는 변수의

수를 나타낸다. 분할 알고리듬은 이 비용함수를 사용해 각 분할 단계마다 성능 제약 조건을 만족시키며, 하드웨어 크기와 통신 오버 헤드를 가장 적게 하는 오퍼레이션을 소프트웨어로 옮긴다.

COSYMA 시스템 [4]은 소프트웨어 지향적 접근 방식이다. 분할은 특정 블럭을 소프트웨어에서 하드웨어로 옮김으로써 얻을 수 있는 성능 향상과 통신 비용을 추정하는 비용함수에 기반을 둔다. 분할은 몇 개의 오퍼레이션들의 집합인 기본 스케쥴링 블록(basic scheduling block) 단위로 수행된다. 이 논문에서 사용한 비용함수는 다음과 같다.

$$\triangle C(b) = \omega^*(t_{hw}(b)) - t_{sw}(b) + t_{com}(b) * It(b)$$

여기서 $t_{hw}(b)$는 b 블럭을 하드웨어로 구현할 때 b 블럭의 수행 시간이며, $t_{sw}(b)$는 b 블럭을 소프트웨어로 구현할 때 b 블럭의 수행 시간이고, $t_{com}(b)$는 b 블럭을 소프트웨어로 구현할 때 하드웨어-소프트웨어 통신에 걸리는 시간이다. ω는 weight factor이며, It(b)는 b 블럭의 반복 실행 횟수이다. 즉, 한 블럭을 소프트웨어로 옮길 때 그 블럭에 의한 성능 향상을 추정하는 것이다.

[5]에서의 비용함수는 면적과 수행시간에 가중치를 두었다.

$$C = \alpha^* \ln(\text{area}) + \beta^* \ln(\text{delay})$$

알맞은 비용함수를 선택하는 것은 매우 중요하다. 예를 들면, 총 비용 계산에 메모리 비용을 고려함으로써 [6], 다른 요소들에 영향을 받지 않고도 메모리 비용을 줄일 수 있다.

Co-synthesis는 분할된 내부표현으로부터 하드웨어와 소프트웨어를 합성하는 과정이다. 일단 분할된 후에는 기존의 하드웨어 합성과 소프트웨어 합성의 문제로 귀착되므로 기존의 합성 기법을 이용하면 된다. 하드웨어 합성을 위해서는 상위 수준 합성과 논리합성을 이용하며, 소프트웨어 합성은 내부표현으로부터 소프트웨어 프로그램을 생성하는 과정이다.

이때, 하드웨어-소프트웨어 간의 인터페이스의 문제는 복합 시스템의 설계에서 새로이 발생되는 문제이다. 하드웨어는 동시성(concurrency)을 지니며, 소프트웨어는 순차성(sequentiality)를 지닌다. 이러한 이질적인 두 요소간의 상호작용을 위한 데이타 전송이나 통신을 위해서는 인터페이스가 필요하다. 인터페이스 합성 단계에서는 시스템 전체의 성능에 최소한의 부담을 주면서 최소 비용(면적, 전력 등)으로 하드웨어, 소프트웨어, 또는 하드웨어와 소프트웨어가 공존하는 형태로 인터페이스 회로를 합성하는 것이 목표이다. 이를 위해서는 시스템 분할단계에서 두 요소간의 통신이나 동기화(synchronization)를 최소화하는 방향으로 분할이 수행되어야 한다. 시스템 분할이 잘못되면 하드웨어와 소프트웨어 사이의 과도한 통신과 동기화의 영향으로 오히려 소프트웨어만 사용하여 구현한 경우보다도 성능이 떨어질 수도 있기 때문이다.

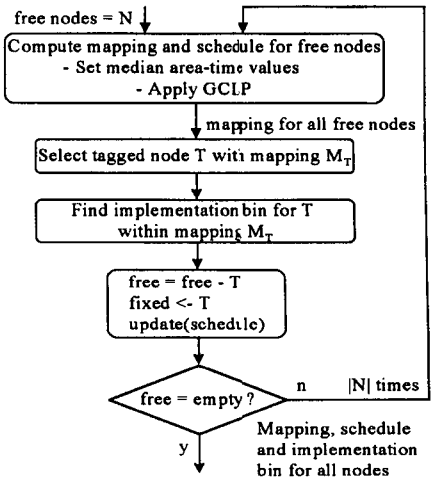
인터페이스의 영향을 정확히 고려하기위해서는 인터페이스에 대한 적절한 모델링이 필요하며 시스템의 구현을 위해서는 상호작용하는 하드웨어와 소프트웨어의 요구에 따라 인터페이스의 기능을 자세하게 정의할 수 있어야 하며, 요구된 기능대로 인터페이스를 device driver code와 register, flip-flop, gate의 구현 수준으로 합성할 수 있어야 한다.

인터페이스의 역할은 FIFO queue와 같은 소규모 버퍼 (buffer)를 이용한 데이타 완충이나 직병렬 변환 등의 간단한 데이타 조작기능도 있지만, 대부분은 입출력 신호의 종류, 신호간의 선후 관계 내지는 원인-결과 관계, 관련 신호사이의 최소/최대 시간 간격 등 주로 상호작용하는 두 모듈간의 시간적, 인과관계적 요구조건을 만족시키는 기능을 갖는다. 따라서 하드웨어-소프트웨어 인터페이스의 경우에는 기존의 하드웨어 설계에 사용되는 모델링, 기능 정의 및 합성 방법을 그대로 적용할 수 없으므로, 인터페이스의 특성을 효율적으로 고려하는 합성 방법이 필요하다.

시스템 통합은 합성된 시스템 구성요소인 하드웨어, 소프트웨어, 인터페이스를 하나의 복합 시스템(mixed system)으로 통합하는 과정으로, 일반적으로 하드웨어는 상위 수준 합성, 논리 합성 및 기술 매핑 (technology mapping)을 거쳐 네트리스트 (netlist)형태로 생성되며, 소프트웨어는 사용하는 프로세서의 구조에 적합하게 최적화되고 컴

파일되어 object code로 생성된다. 인터페이스는 하드웨어 부분과 소프트웨어 부분으로 구성되며 동일한 형태로 합성된다.

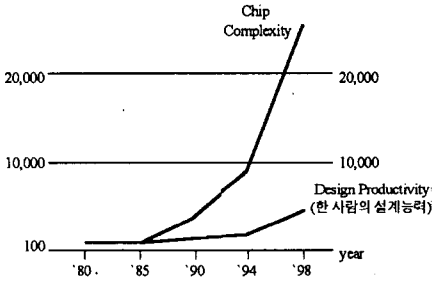
분할과 합성 그리고 추정과 분석은 서로 밀접하게 연결된 세부분야들이다. [Kala95]에서는 통합 설계 구조를 제안했다. 이 구조에는 분할과 합성, 시뮬레이션 툴들이 포함되어 있다. 각각의 오퍼레이션을 구현하기 위하여 구현 가능한 여러개의 옵션들을 허용하며, 하드웨어-소프트웨어 분할은 스케쥴링과 구현 가능한 옵션의 선택과 동시에 수행된다. 이 방법의 전체적인 흐름은 그림 5와 같다. 그림과 같이 전역적 정보(global information)가 사용되는 global criticality(GC)와 지역적인 특성(local characteristic)이 강조되는 local phase (LP)가 사용된다. 하드웨어-소프트웨어 분할 MT {SW, HW}는 GC와 LP 분석에 의해서 결정된다.



〈그림 5〉 연관된 복잡한 분할문제를 풀기위한 greedy 방법

IV. 설계의 재사용

전자 시스템의 하드웨어와 소프트웨어의 복잡도는 계속 증가하고 있는 추세이며, 전문가의 소프트웨어 설계 능력의 발전 속도는 복잡도의 증가 속도를 따르지 못하고 있다. 하드웨어 설계에 있어서도 칩의 복잡도의 증가가 설계자의 설계 능력 증가를 크게 앞지르고 있으며, 이의 연도별 추이를 그리면 그림 6 [8]과 같다. 그림에서 1980년도 칩의 복잡도와 설계 능력을 각각 100으로 했을 때의 연도별 증가를 보여준다.



〈그림 6〉 칩의 복잡도 증가와 설계자의 설계 능력 증가 추이

이렇게 계속 증가하는 시스템의 복잡도에 잘 대처하기 위한 가장 효과적인 방법은 바로 설계의 재사용(design reuse)이다. 기존에 설계된 것을 사용하지 않고, 처음부터 새로 설계하면 많은 시간이 소요된다. 그러므로 이미 만들어 진 설계를 재사용하면 시간이나 비용면에서 유리할 수 있다.

하드웨어와 소프트웨어의 효과적인 설계 재사용을 위해서는 다음과 같은 점이 고려되어야 한다.

- 이전에 만들어진 설계에 대한 데이터베이스(database) 즉, 라이브러리를 만들어야 한다.
- I/O 인터페이스의 확실한 정의가 제공되어야 한다.
- 외부에서 콘트롤(control) 신호를 줄 수 있게 해야 한다.
- 계층적이고 모듈화된 설계가 효과적이다.
- 파라미터화되고(parameterized), 프로그램가능한(programmable), 형태를 바꾸어 다른 설계를 만들 수 있는(reconfigurable) 모듈들을 설계하면 재사용 횟수를 늘일 수 있다.
- 프로세서 코아(core)와 오퍼레이팅 시스템의 표준화가 이루어져야 한다.
- 유연한(portable) 언어들, 즉 여러곳에서 사용할 수 있는 자바나 C와 같은 언어들이 사용되

어야 한다.

IC(integrated-circuit) 칩이 점점 조밀해 짐에 따라, 칩을 만드는데 필요한 비용이 점자 증가한다. 대량생산은 보다 경제적이기 때문에 reconfiguration에 의한 소자의 재사용은 설계 비용을 줄일 수 있다. reconfiguration 설계는 재사용할 수 있는 가능성을 보다 많이 제공하지만, 설계의 성능은 주문형 (application-specific) 설계의 성능보다는 떨어질 수 있다.

Siemens사의 설계 재사용 연구결과 재사용 가능한 구성요소의 개발에 걸리는 시간은 독립적으로 사용되는 구성요소의 개발에 걸리는 시간에 비해 120~150% 정도가 걸린다[9]. 또한, 설명문(document)작성의 경우, 독립적으로 사용되는 설계의 경우에는 사양 (specification)이 곧 설명이 되므로 비교적 쉬운 작업이나, 재사용 가능하게 설계할 경우에는 새로운 설명 작성에 구성요소를 설계하는 것보다도 많은 시간과 노력이 필요하다. 그러므로, 전체 비용이나 시간을 생각한다면, 5번 이상을 재사용할 수 있는 경우가 경제적이라고 한다. 재사용할 수 있는 요소들은 다음과 같다.

- 자주 사용하는 것(frequency)과 재사용할 때 이익이 많은 것(utility)
- 간단하게 보고 쓸 수 있도록 요약을 해 놓은 것(abstraction)
- 사용자가 사용하기 편하게 해 놓은 것 (habitability)

V. 결론 및 향후 방향

하드웨어-소프트웨어 통합 설계는 기존의 하드웨어와 소프트웨어의 여러 설계 기술을 이용한다는 점에서 크게 새로운 분야는 아니다. 그러나, 다양한 CAD tool의 개발과 시스템의 복잡도의 증가, 그리고 다양한 전자 시스템의 설계에 적용할 수 있는 넓은 응용 분야 등으로 해서 실용화되면 매우 유용한 분야이다. 통합 설계는 하드웨어와 소프트웨어가 구분되지 않은 상위 단계의 시스템의 기술·시뮬레이션 및 검증·하드웨어-소프트웨어 분할·접속 부분 등의 설계 등으로 이루어진다. 현재 활발한 연구가 국내외에서 이루어지고 있으나, 아직은 기술 수준이 일반적인 실용화에는 미흡한 실정이다[10]. 특히, 상위 단계에서의 하드웨어-소프트웨어 성능 및 비용 추정기술, 다양한 설계의 대안을 찾아서 최적의 해를 구하는 방법, 하드웨어-소프트웨어 분할과 스케쥴 기술, 코드 생성 기술 및 설계 재사용 기술 등이 미흡하므로, 이들 기술의 개발이 계속 이루어진다면 앞으로 매우 유용한 분야라고 할 수 있다.

참 고 문 헌

[1] J. K. Adams and D. E. Thomas, "The Design of Mixed Hardware/Software Systems," Proc. of DAC, pp.515-520, 1996.

[2] G. Gong, D. Gajski, and S. Bakshi, "Model Refinement for Hardware-Software Codesign," ACM Trans. on Design Automation of Electronic Systems, Vol.2, No.1, pp.22-41, Jan 1997.

[3] R. Gupta and G. De Micheli, "System-level Synthesis Using-Reprogrammable Component," Proc. of EDAC, pp.2-7, March 1992.

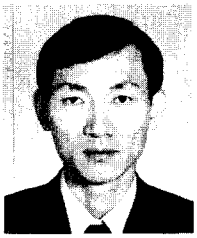
[4] R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers," IEEE Design & Test of Computers, Vol.10, pp.64-75, Dec. 1993.

[5] E. Barros and A. Sampaio, "Towards provably Correct Hardware/Software Partitioning Using Occam," 3rd Int'l Workshop on Hardware/Software Codesign, pp.210-217, 1994.

[6] D. Park and H. Shin, "Partitioning for Minimal Memory in Hardware-Software Codesign," Proc. of ISCAS, pp.647-650,

May 1996.
〔7〕 A. Kalavade, "Systm-level Codesign of Mixed Hardware-Software Systems," UCB /ERL M 95/88, Univ. of Calif., Berkeley, Oct. 1995.
〔8〕 Business Week, McGRAW-HILL, pp.59, April 25, 1994.
〔9〕 M. Mrva, Computer, IEEE, pp.93-95, Aug. 1997.
〔10〕 김보관, 최기영, 하순회, 신현철, 하드웨어-소프트웨어 통합설계 기술 개발 결과보고서, 한국과학재단, 1997.

저 자 소 개



申 鉉 哲

1955年 9月 12日生
1978年 2月 서울대학교 전자공학과 졸업(학사)
1980年 2月 한국과학기술원 전기 및 전자과 졸업(석사)
1987年 10月 미국 U.C.Berkeley 전기 및 컴퓨터공학과 졸업(박사)

1980年 8月～1983年 8月 금오공과대학 전자공학과 조교수
1987年 11月～1989年 7月 미국 AT & T Bell Lab. Murray Hill 연구원
1989年 9月～현재 한양대학교 전자공학과 부교수

주관심분야 : 반도체 집적회로 설계 자동화, 디지탈 신호처리 시스템 설계 등임