특 집

단일칩시스템 설계검증을 위한 가상프로토타이핑

기 안 도
(주)다이나릿시스템

요 약

여러기능들이 복합적으로 통합되고 있는 단일칩시스템을 설계하는데 있어 소프트웨어와 하드웨어를 가능한 일찍 통합하여 검증하는 것이 무엇보다 중요하다. 이러한 조기 통합검증에 필요한 것이 가상프로토타입(Virtual-Prototype)이다. 본 고에서는 IP(Intellectual Property)와 단일칩시스템(SoC : System-on-a-Chip) 설계 및 검증에서 가상프로토타입의 필요성과 역할 그리고 이에 관련된 기술들에 대해 정리하고, 프로세싱 코어가 있는 단일칩시스템을 SystemC로 가상프로토타이핑한 사례를 통해 그 유용성을 설명한다.

I. 서 론

반도체의 집적도와 복잡도가 증가함에 따라 오류가 포함될 가능성은 더욱 높아지고 있다. 따라서 반도체 칩이 만들어지기 전에 완벽한 설계를 할 수 있는 방법과 도구들이 필요하다. 이는 짧아지고 있는 시장진입시간(TTM : Time-To-Market)에 맞추어 제품을 개발하는데 필수적인 것으로 첫번째 반도체 제작으로 제품화(First-Time-Right Silicon)해야 된다는 시장의 추이를 반영하는 것이다.

과거 디지털 하드웨어를 설계하는데 있어 소프트웨어 시뮬레이션은 필수적이었다. 즉 기능적으로 매우 단순한 디지털 게이트를 조합하여 복잡한 기능으로 구현함에 있어 미리 소프트웨어 시뮬레이션을 하고 그 출력파형을 보고 기능을 검증하여 하드웨어 설계를 보다 신뢰성 높게 설계할 수 있었다. 이와 유사한 상황이 요즈음 다시 재연되고 있다. 최근의 반도체 칩은 단순한 게이트의 조합이 아니라 하드웨어와 소프트웨어가 공존하는 시스템이 되고 있고, 대부분의 작은 하드웨어 블록도 소프트웨어적인 제어가 필요하게 되었다. 이는 신호파형으로 설계된 기능을 검증하는 것이 불가능하다는 뜻이고 기존의 디지털 하드웨어 시뮬레이션 기법에 더하여 추가적인 검증환경이 필요하다는 것이다.

이러한 추이로 볼 때 향후 대부분의 설계들이 기능적인 면에서 복잡해질 것이고, 이는 설계와 검증을 더욱 어렵게 할 것이다. 따라서 기존의 설계를 재활용하는 방법과 가능하면 소프트웨어로 해결하는 방법이 기본적인 설계기법으로 자연스럽게 받아들여지고 있다. 즉 모든 설계는 항상 재활용을 염두에 두고 설계하게 되었고, 이는 모든 설계가 다양한 응용환경에서 쉽게 활용되도록 설계되어야 하며 아울러 매우 용이하게 기능을 검증할 수 있는 환경까지 동시에 설계되어야 함을 의미한다.

아울러 칩설계는 기본 골격을 유지하면서 꼭 필요한 부분은 하드웨어로 구현하고, 가능한 칩에서 운용될 소프트웨어를 변화시킴으로써 전체 기능을 구현하는 플랫폼기반설계(platform-based design)가 필요하다. 이러한 것은 다음과 같은 점을 시사한다. 블록수준설계든 전체 칩수준설계든 하드웨어와 소프트웨어가 동시에 개발

되어야 하며, 이를 위해 반도체 소자로 만들어지기 전에 소프트웨어를 개발할 환경이 필요해진다. 이러한 요구를 충족시킬 수 있는 방법이 가상프로토타입이다.

가상프로토타입은 최종 하드웨어가 아니지만 실제 반도체 칩 또는 블록을 대신할 수 있는 가상적인 환경이다. 여기에는 FPGA(Field Programmable Gate Array), 하드웨어 가속기(hardware accelerator), 하드웨어 에뮬레이터(hardware emulator), 소프트웨어 모델 등이 포함된다.

이어지는 절에서 단일칩시스템 설계의 추이에 대해 살펴보고, III절에서 IP 설계에서의 가상프로토타이핑에 대해 설명하며, IV절에서 단일칩시스템에서의 가상프로토타입을 살펴본다. 그리고 V절에서 단일칩시스템을 가상프로토타입을 적용하여 실시간운영체제를 실행시킨 사례을 설명한 후, VI절에서 결론을 맺는다.

II. 단일칩시스템 설계의 추이

본 장에서는 점점 복잡해지고 있는 기능과 큰 규모의 하드웨어를 구현할 때 적용해 볼 수 있는 설계방법 및 검증방법에 대해 살펴본다.

반도체 칩의 집적도가 높아지고 성능이 증가함에 따라 새로운 응용들이 하드웨어로 구현되고 있다. 이러한 응용의 많은 것들이 데이터 처리에 연관이 있고, 이들 응용들은 복잡한 알고리즘으로 표현되는 것이 일반적이다. 예로써 암호관련 응용, 이미지처리 응용, 디지털통신 응용 등이 이러한 부류라고 할 수 있다. 또한 여러 기능블록들이 하나의 칩으로 통합되는 단일칩시스템(SoC : System-on-a-Chip)이 주류를 이루고 있다.

고성능 프로세서들을 사용할 경우 복잡한 알고리즘을 충분히 빨리 수행시킬 수 있으나, 여전히 반도체 하드웨어로 구현하는 것이 필요하다. 예로써 이동통신용 기기의 경우 그 사용환경의 특성상 고성능 프로세서를 사용할 수 없고, 저전력으로 구현해야 되는 등의 이유로 전용 하드웨어 즉 반도체 칩을 만드는 것이 필수적이라 할 수 있다. 따라서 집적도 측면에서 크고 기능 측면에서 복잡한 하드웨어를 위한 설계방법론과 설계환경이 필요하다. 또한 회로의 크기가 크며 기능이 복잡한 설계를 검증하는데는 설계 그 자체보다 더 큰 노력이 들므로 이를 극복하기 위한 검증환경과 방법이 필요하다.

1. 큰 디자인을 위한 방법들

구현할 설계가 매우 큰 경우 전체 설계를 계층구조를 갖도록 구분하고, 다루기 용이한 규모의 블록으로 나누고, 이들 블록 중 이미 설계된 블록이 있는 경우 재활용하는 방법을 적용해 볼 수 있다. 즉 계층적 설계(hierarchical design), 분할 후 설계(divide and conquer), 설계 재사용(design reuse) 기법 등을 사용하는 것이다. 이들 방법을 통해 완성된 블록을 하나씩 추가하는 축적설계(incremental design) 방법으로 전체 설계를 완성하는 것이다.

2. 복잡한 기능 블록을 위한 방법들

통신이나 멀티미디어 등에 관련되는 규격들이 매우 복잡하고 많은 부분들이 수식으로 규정되어 있는데 이런 것을 곧바로 하드웨어로 구현한다는 것은 매우 어렵고, 풍부한 하드웨어 설계경험이 요구된다. 따라서 일정한 수준의 훈련을 받으면 쉽게 적용해 볼 수 있는 새로운 설계 방법이 필요하다.

복잡한 기능은 우선 프로그래밍 언어로 기능수준모델(functional-level model)을 만든다. 이것을 기준으로 행위수준모델(behavioral-level model)로 구체화시킨 후, 기능수준모델과 입출력 값이 동일하거나 또는 허용범위를 만족하는지를 검증한다.

기능수준모델이 구현될 설계가 주어진 입력에 대해 어떤 동작과 처리를 할 것인지를 모델링 한 것이라면, 행위수준모델은 이에 더하여 최종하드웨어와 동일한 입출력 포터와 종류를 갖고 각 입

출력 포터를 통해 전달되는 정보의 값 또한 동일한 것이다. 그러나 행위수준모델의 입출력 동작과 내부동작이 반드시 최종 하드웨어와 동일할 필요는 없지만, 행위수준모델에 소요시간 정보를 지정하여 최종 하드웨어와 비슷한 또는 동일한 타이밍을 갖도록 구현할 수 있다.

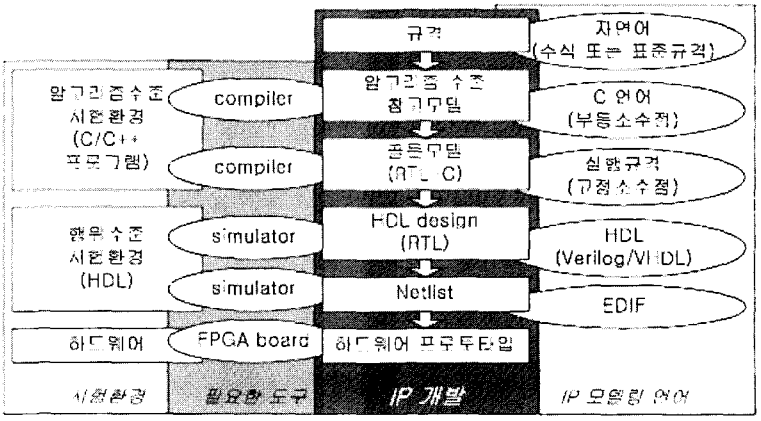
행위수준모델을 레지스터전송수준모델(register-transfer-level model)로 구체화시키고 이것을 합성 가능한 수준으로 개선한다. 이 과정을 행위수준합성기(behavioral-level synthesis)/상위수준합성기(high-level synthesis) 기술[1,2]을 통해 자동화할 수 있다. 행위수준모델이 입출력 측면에서 최종 하드웨어와 동일하지만, 모델의 내부는 구조적인 측면에서 최종 하드웨어의 모습을 갖도록 구체화하는 것이 일반적이며, 이것을 레지스터전송수준모델에서 유지하도록 한다. 또는 행위수준모델은 내부의 구조가 데이터 흐름에 초점을 맞추고, 이것을 레지스터전송수준 모델로 개선할 때 내부의 구조를 구체화하고 최적화 할 수 있다.

이러한 일련의 설계과정이 점진적개선(progressive refinement) 과정이고, 이 방법을 통해 상위수준 설계에서 하위수준 설계로 구체화할 때 각 단계에 적용하는 구체적인 설계 기법은 기계적 또는 정형적인 것이 될 수 있다. 즉, 일정 수준으로 훈련된 설계자가 설계하는 것이 가능하게 된다.

Ⅲ. IP 또는 블록 설계에서의 가상프로토타입

IP 설계의 전형적인 설계흐름은 〈그림 1〉과 같이 규격작성부터 시작하게 될 것이다. 규격은 통상 문서형식을 갖게 되는데, 수학적인 식이나 표준규격 등이 대표적인 것이다. 다음 단계로는 이 규격을 만족하는 알고리즘 수준의 참고모델(reference model)을 만드는 것이며, 이때 C언어와 같은 상위수준언어를 활용하여 프로그램을 작성하는 것이 여기에 해당한다. 특히 이 단계에서는 부동소수점(floating-point) 데이터타입(data type)과 연산(operation)을 자유롭게 활용하는 특징이 있다. 알고리즘 수준의 참고모델이 주어진 규격을 만족 할 경우, 하드웨어로 설계하는 중간 단계로 데이터 폭이 정확히 규정되는 모델(RTL-C 모델)로 변경한다. 이때 고정소수점(fixed-point) 데이터타입을 사용해야 되고, 필요한 경우 기능블록을 좀 더 구체적으로 분할하여 파이프라인이나 자원분배가 용이하게 적용될 수 있도록 구체적인 구조를 갖도록 할 수 있다. 이 RTL-C 모델(데이터 폭 정확도가 있는 모델)이 개발하고자 하는 IP의 실질적인 골든모델(golden reference mode)이 된다.

RTL-C 골든모델을 하드웨어 기술언어(HDL: hardware description language)[3,4]로 변환



〈그림 1〉 전형적인 IP 설계 회로도

하여 RTL 모델을 만든다. 이 과정은 경우에 따라 소프트웨어를 이용하여 자동화할 수 있다. RTL 모델이 만들어지면 이것을 합성하여 규격대로 동작하는지를 점검하는 것으로 IP 설계의 중요절차는 완료된다. 그러나 IP 설계의 대부분의 시간은 설계 자체보다 기능을 검증하는 과정에 사용된다. 예로써 C 언어로 프로그램 된 알고리즘 수준 참고모델은 역시 C 언어로 된 시험환경(testbench)으로 그 동작을 검증하게 될 것이고, 같은 시험환경을 RTL-C 모델에도 사용하게 될 것이다. 그러나 RTL 모델의 경우 PLI (Programming Language Interface)/VPI (Verilog Procedural Interface) 또는 FLI (Foreign Language Interface)와 같은 특별한 인터페이스 방법을 동원하여 기존의 C 언어 시험환경을 재활용할 수 있지만, 보통의 경우 HDL로 된 시험환경을 새로 구축한다. 이것이 의미하는 바는 설계의 동작을 검증하는 환경을 준비하는데 매우 큰 노력이 필요하다는 것이며, 이 검증환경이 점점 복잡해지고 있다. 이런 경향은 E verification language[5], OpenVera[6], TestBuilder[7], SCV(SystemC Verification Library)[8] 등과 같은 시험환경 자동화 전용언어들이 점점 많이 사용되고 있다는 점에서 확인할 수 있다.

IP를 최종적으로 검증하는 단계로 FPGA 소자를 이용하여 하드웨어 프로토타입을 만들며, 이 경우 IP 주변의 것들도 모두 하드웨어로 만들어야 하므로 비용이나 시간적으로 매우 심각한 부담이 된다. 따라서 이런 부담을 줄일 수 있는 방법이 필요하며 그 중 하나가 가상프로토타입이다.

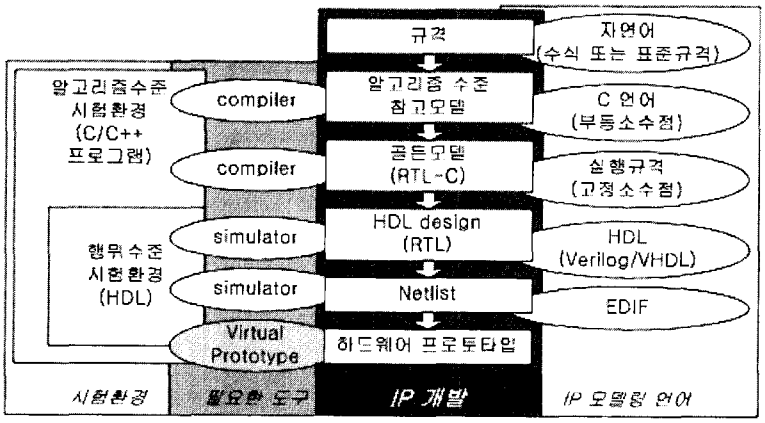
이제까지 설명한 일반적인 IP 설계 방법은 다음과 같은 어려운 점들이 있다.

- IP를 모델링 한 언어가 변하면 그에 따라 시험환경도 새로 구축해야 한다. 즉, RTL-C 모델인 경우 C 언어, RTL 모델인 경우는 HDL, FPGA 프로토타입의 경우는 하드웨어가 필요하다.
- IP의 모델이 구체화됨에 따라 검증에 보다 높은 성능이 필요해진다.
- 시험환경이 RTL-C와 RTL 그리고 FPGA 프로토타입에 따라 다르므로 디버깅 결과를 참고모델/골든모델에 반영하기 어렵게 되어 결국 참고모델/골든모델이 더 이상 의미가 없어질 수 있다.

이러한 어려운 점을 개선하는 방법으로 다음과 같은 접근법이 필요하다.

- 시험환경을 통일하여 검증을 용이하게 한다.
- 핀 수준보다는 트랜젝션수준으로 검증함으로써 검증성능을 개선한다.

이러한 개선방법을 적용할 때 <그림 2>와 같이 되고, 이것은 가상프로토타입을 이용하면 가능하



<그림 2> 가상프로토타이핑기법을 이용한 IP 설계 흐름도

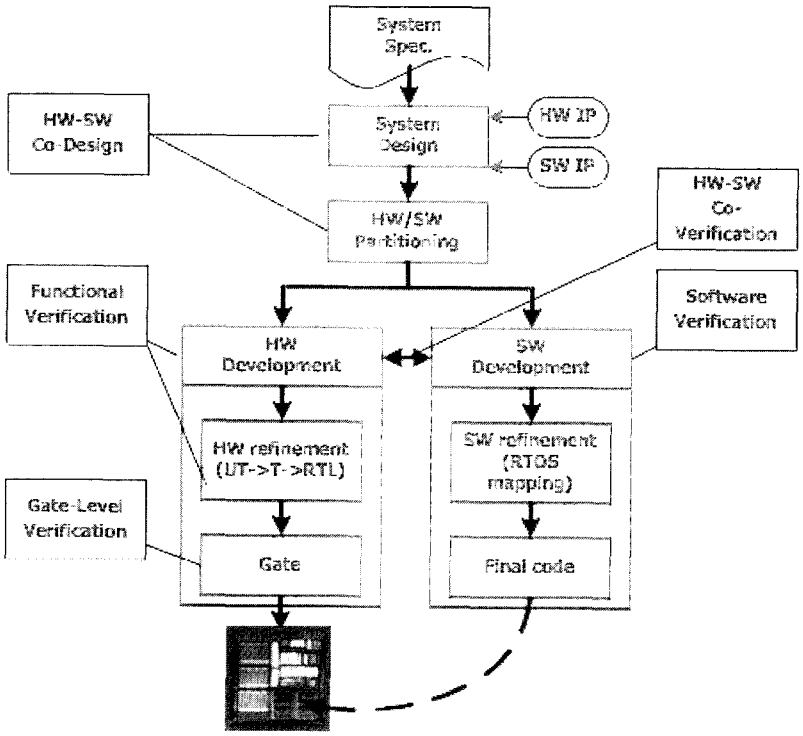
다. 즉, 한쪽은 컴퓨터에서 수행되는 시뮬레이션이고 다른 한쪽은 하드웨어로 구성되는 환경에서 사용자는 시험환경을 계속 재활용하고, 검증할 설계는 하드웨어에서 검증한다. 이런 가상프로토타이핑은 호스트 컴퓨터와 연동되는 하드웨어 가속기를 이용하여 구축할 수 있다.

이러한 접근법에 활용될 수 있는 기술로는 Quickturn(지금은 Cadence)사의 Q/Bridge, IKOS(지금은 Mentor Graphics)사의 'Co-Modeling' 기술[9]인 TIP(Transaction Interface Portal), TIP를 표준화할 목적으로 Accellera의 Interface Technical Committee에서 시작한 SCE-MI(Standard Co-Emulation Modeling Interface)[10] 등이 있다. 대표적인 제품으로는 Quickturn/Cadence사의 CoBalt, IKOS/Mentor Graphics사의 VStation, Alatek사의 HES, EVE사의 Zebu, Dynalith Systems사의 iPROVE[11] 등이 있다.

Ⅳ. 단일칩시스템에서의 가상프로토타입

단일칩시스템의 정의는 다양하다. 예로써 VSI (Virtual Socket Interface Alliance)에서는 'system on silicon, system-on-a-chip, system-LSI, system-ASIC, system-level integration device라고 하는 고집적디바이스'라고 정의하였고, Dataquest에서는 'on-chip memory와 프로그램 가능한 코어가 있는 10만 게이트 이상의 디바이스'라고 정의하였다. Chang[12]의 경우 '완전한 최종제품의 주요요소들이 단일칩 또는 칩셋에 집적된 복잡한 집적회로'라고 정의하였다.

이러한 정의에서 주목할 점은, 매우 복잡한 기능을 포함하며 프로세싱코어(즉, 마이크로 프로세서 또는 디지털신호처리프로세서)를 포함하는 시스템이라는 것이다. 따라서 단일칩시스템은 하



〈그림 3〉 단일칩시스템 설계 흐름도

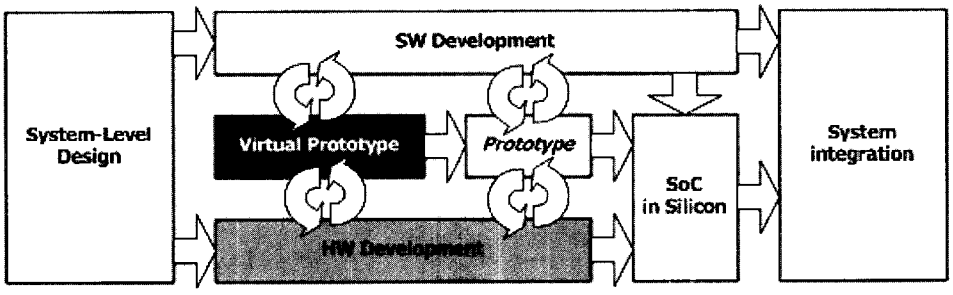
드웨어와 소프트웨어가 같이 있어야 되며, 이는 개발과정에서도 하드웨어와 소프트웨어가 같이 설계되어야 하고 검증되어야 함을 의미한다.

〈그림 3〉에 일반적으로 받아들여지고 있는 단일칩시스템 설계흐름을 보였는데, 크게 시스템설계, 하드웨어 설계, 소프트웨어 설계로 구분할 수 있다. 즉, 하드웨어와 소프트웨어가 분화되기 전까지를 시스템 설계라고 하고, 이후 각각을 하드웨어 설계와 소프트웨어 설계라 한다. 시스템 설계의 목적은 구현할 시스템을 정확히 규정하는 것과 선택 가능한 여러 구조 중 목적에 적합한 구조를 결정하는 것이며, 이때 무엇을 하드웨어로 구현하고 무엇을 소프트웨어로 구현할 것인지 결정하는 것이다. 하드웨어 설계의 목적은 합성 가능한 설계를 만드는 과정으로 시간, 전력, 성능 등을 최적화하는 것이 목적이라 할 수 있다. 반면 소프트웨어 설계는 실시간운영체제(RTOS : Real Time Operating System) 개발 또는 포팅, 새롭게 추가된 하드웨어 블록의 디바이스 드라이브 개발, 코드 최적화, 소프트웨어 개발 환경 구축 등이 목적이다.

통상적으로 하드웨어에 대한 설계와 검증은 HDL(Hardware Description Language) 시뮬레이터 또는 이를 지원하는 환경에서 충분히 가능하다. 그러나 단일칩시스템의 경우와 같이 프로세싱 코어가 포함될 경우, HDL 시뮬레이션 환경으로는 소프트웨어 실행이 어렵기 때문에 한계가 있다. 따라서 단일칩시스템 설계흐름에서 매우 중요하게 인식되는 것이 하드웨어와 소프트웨어를 동시에 검증하는 환경이며, 이것은 가능한 이른 시기에 소프트웨어를 개발하고 수행시켜 볼 수 있게 하는 환경이다.

일반적인 설계흐름에서는 설계의 초기단계에 하드웨어와 소프트웨어를 분할하고, 시스템의 각 부분을 여러 팀 또는 설계자들이 나누어 구현하는 것인데, 이 경우 이들 각 부분을 통합하는 과정에서 오류와 개선점들이 발견되므로, 결국 시스템 완성에 비용상승과 설계지연 등을 초래하게 된다. 따라서 설계 초기단계부터 하드웨어와 소프트웨어를 같이 통합하여 전체 기능을 검증하는 것이 바람직하다. 일반적으로 시스템수준 설계 단계에서 주로 활용되는 설계방법과 환경을 통칭하여 동시설계(co-design)이라 하고, 이에 속하는 것으로 CASTLE(C)[13], Chinook[14], COSYMA(Cx)[15], LYCOS(C)[16], MEIJE[17], PeaCE[18], Polis(Esterel)[19], Ptolemy[20], Vulcan(HardwareC)[21] 등이 있다. 이들은 대부분 1990년대 초부터 하드웨어와 소프트웨어가 공존하는 내장형시스템(embedded system) 설계를 위해 시도된 것들이지만, 그 기본 개념들이 단일칩시스템 설계에도 활용되고 있다. 이러한 동시설계 도구들은 하드웨어와 소프트웨어를 나누고 각각을 합성하고 동시시뮬레이션(co-simulation)하는 등의 기능을 포함한다.

상용 EDA 도구로는 Synopsys사의 COSSAP과 Eaglei, Mentor Graphics 사의 Seamless, Cadence SPW가 대표적인 동시시뮬레이션 도구이며, 이들은 시스템을 통합하는 형태로 발전하였다. 예로써, Cadence 사의 VCC(Virtual Component Co-Design), Synopsys 사의 Co-



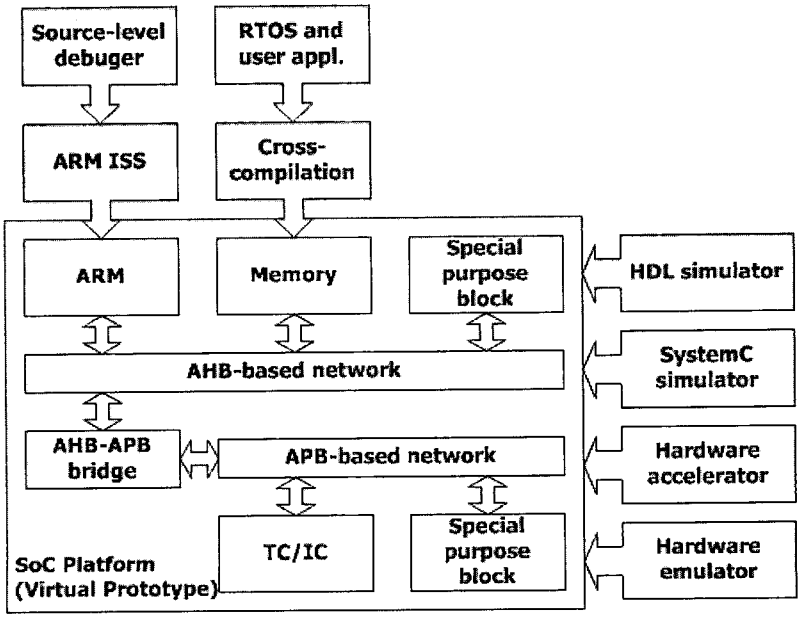
〈그림 4〉 가상프로토타입을 이용한 단일칩시스템 설계 흐름도

Centric System Studio, Mentor Graphics 사의 Platform Express, CoWare사의 ConvergenSC System Designer 등이 있다.

〈그림 4〉에 가상프로토타입을 이용한 단일칩시스템 설계의 개념적 설계 흐름을 보였다. 시스템 수준설계에서는 내부적으로 동시시뮬레이션(co-simulation) 기능을 포함할 수도 있으며, 최종적으로 하드웨어로 구현될 부분과 소프트웨어로 구현될 부분을 구분한다. 이후 하드웨어-소프트웨어 동시검증(co-verification) 과정에서 가상프로토타입이 활용될 수 있다. 가상프로토타입은 기본적인 구조가 결정된 행태의 플랫폼으로 프로세싱 코어, 메모리, 네트워크, 중요 입출력 블록 등으로 구성되며, 이들을 활용 가능한 여러 도구들을 이용하여 가상적으로 구축한 것이다. 그리고 필요한 경우 FPGA와 같은 하드웨어로 물리적인 프로토타입(physical prototype)을 만들어 실동작속도(real-time speed)에 근접하게 그리고 실제 사용환경에 가깝게 검증한다. 이런 과정을 통해 기능이 검증되면, 반도체로 만들어 시스템에 통합하여 통합시험을 함으로써 개발과정이 마무리된다.

여기서 물리적인 프로토타입은 하드웨어를 제작한다는 것에 기인하는 여러가지 단점이 있는 반면 가상프로토타입은 기본 뼈대는 결정되어 있지만, 필요와 상황에 따라 일부 구조를 바꾸거나, 하드웨어와 소프트웨어를 넘나들면서 성능변화를 분석하는 등 유연한 검증을 할 수 있다는 장점이 있다. 즉 원하는 모든 변화를 다 관측할 수 있고, 그 동작을 용이하게 변화시킬 수 있으므로 디버깅이 매우 용이하다. 특히 하드웨어 구조의 뼈대가 결정되면 곧바로 소프트웨어 개발자들이 프로그램 개발과 실행 그리고 디버깅을 할 수 있는 동작하는 플랫폼(working platform)을 확보한다는 점은 매우 유용한 것이다. 그러나 실제 하드웨어 블록을 사용하는 것이 아니므로 성능이 충분하지 않을 수 있다. 이런 경우 하드웨어 가속기나 에뮬레이터를 이용하여 부족한 성능을 보완할 수 있다.

이러한 접근법을 적용하는 한가지 예로 〈그림 5〉와 같이 가상프로토타입을 구축할 수 있다. ARM[22] 프로세서, AHB[23]와 APB 버스, 특별한 기능을 위한 블록들, 타이머제어기(TC), 인터럽트제어기(IC), 등으로 구성되는 단일칩시스



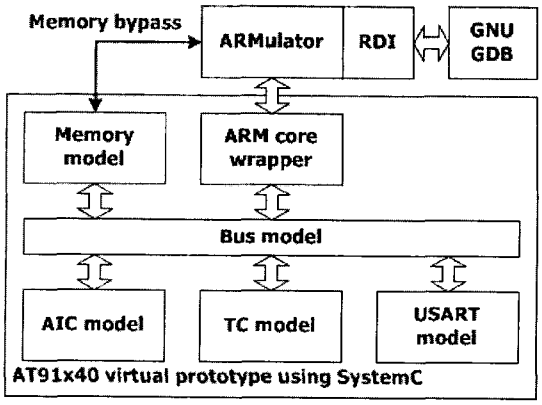
〈그림 5〉 가상프로토타입의 일례

템를 SystemC[24]로 모델링한다. 여기서 ARM은 ISS(Instruction Set Simulator, 명령어집합 시뮬레이터)[25]로 대신하고, 특별한 기능의 하드웨어 블록들은 하드웨어가속기[26] 또는 하드웨어 에뮬레이터[27]로 대신하여 성능을 높임으로써 어느 정도 수용 가능한 수준의 소프트웨어 실행 속도를 제공한다. 보통의 경우 타겟 프로세서와 호스트 컴퓨터의 프로세서가 다르므로 크로스컴파일 환경(cross-compilation environment)이 필요하다. 블록들의 기술언어에 따라 HDL 시뮬레이터, SystemC 시뮬레이터, 아날로그-디지털 혼합모드 시뮬레이터 등을 사용한다.

V. 가상프로토타입 사례

Atmel사에서 ARM 코어와 메모리 그리고 RTOS가 실행되는데 필요한 블록들을 단일칩에 내장한 소자를 여러 목적에 활용할 수 있도록 공급하고 있고, 해당 칩들에 μClinux(MMU(Memory Management Unit)가 없는 마이크로제어프로세서(microcontroller)용 Linux)[28]가 지원되므로, 이것을 앞서 설명한 가상프로토타입으로 구축하여 그 유용성을 살펴본다.

Atmel사의 AT91X40[29]을 SystemC로 모델링하는데 있어 가장 큰 문제는 이칩에 내장되어 있는 ARM7TDMI 코어이다. 이 ARM 코어는 GNU GDB[30]에 포함되어 있는 초기버전의 ARMulator[31]를 이용하여 ARM ISS로 구성하여 해결한다. 또한 RTOS가 수행되는데 꼭 필요한 메모리, TC(Timer/Counter), AIC(Advanced Interrupt Controller), USART(Universal Synchronous Asynchronous Receiver Transmitter) 등은 SystemC로 직접 모델링한다. 이때 모든 블록을 하드웨어 수준의 정밀도로 모델링하는 대신 프로그래밍 모델(Programming model)을 하드웨어와 동일하게 모델링하고, 그 동작은 필요최소한으로 모델링하여 개발기간을 단축한다.



〈그림 6〉 Atmel AT92x40의 가상프로토타입

〈그림 6〉에 보인 바와 같이 AT91X40을 SystemC로 가상프로토타입을 만들고, 여기에 μClinux를 포팅하여 수행시킨다. RDI(Remote Debugger Interface)[32]를 통해 GNU GDB가 원격으로 ARMulator를 디버깅 모드로 수행시킬 수 있도록 하여, 가상프로토타입에서 수행되는 소프트웨어를 원천코드수준에서 디버깅이 가능하다. 가상프로토타입에서 프로그램이 수행된다면 대부분의 동작은 메모리 참조일 것이므로 성능향상을 위해 메모리 참조는 버스모델을 통하지 않고 직접 메모리 모델을 참조하는 방법(memory bypass)도 준비한다. 가상프로토타이핑의 성능을 비교하기 위해 SystemC를 사용하지 않고 C 언어로 순수한 기능모델(functional model)만 구현한 경우도 준비한다. 더 나아가 네트워크모델(AT91x40의 경우 ASB와 APB로 구성되는 버스를 사용함)을 단순한 핸드세이크 방식을 이용하는 경우(simplified bus)와 AHB로 구체화한 경우 그리고 USART와 콘솔과의 연결기능을 더욱 구체화 한 경우 등도 모델링하여 준비한다.

이러한 여러 경우를 적용하여 구축한 가상프로토타입에 μClinux를 부팅하는데 소요되는 시간을 측정하면 〈표 1〉과 같다.

이상의 결과가 시사하는 바는 순수 소프트웨어로 구축된 가상프로토타입이 단일칩시스템 설계 및 검증 그리고 단일칩시스템에서 수행될 소프트

〈표 1〉 μClinux booting time (preliminary result under Sun Ultra 10)

	Functional model using the C language	Virtual prototype using SystemC (ARMulator & SystemC models)			
	ARMulator & C language models	Simplified bus & without memory bypass	Simplified bus & with memory bypass	AHB/APB & with memory bypass	Consol model & with memory bypass
Elapsed time	10 sec	Over 1 hour	1 min 47 sec	4 min 4 sec	10 min 25 sec

웨어 개발에 매우 유용하게 활용될 수 있다는 것이다. 특히 적절한 성능개선방법을 병행한다면 FPGA를 이용하여 구축하는 물리적 프로토타입에 근접한 성능도 가능할 것이다.

VI. 결 론

하드웨어와 소프트웨어가 공존해야만 기능을 발휘하는 시스템이 단일 반도체에 통합되고 있고, 이것을 단일칩시스템이라 한다. 단일칩시스템은 다양한 종류의 하드웨어 블록과 한 개 이상의 프로세싱 코어를 내장하며 전체 기능 중 많은 부분이 소프트웨어로 처리된다. 특히 각종 표준들을 지원해야 하는 요구조건도 만족해야 되므로, 모든 것을 처음부터 설계한다는 것은 현실적이지 못하다. 결국 단일칩시스템 설계는 이미 설계가 완료된 블록과 디바이스 드라이브 등을 최대한 사용하여 전체적인 구조를 결정하고, 이 구조 위에서 수행될 소프트웨어를 개발하여, 가능한 조기에 동작하는 칩을 만드는 것이라 할 수 있다. 이러한 일련의 과정은 시스템 설계, 하드웨어-소프트웨어 동시검증 및 개발, 반도체 칩 제작, 시스템 통합 등으로 구분해 볼 수 있고, 이들 과정 중 하드웨어-소프트웨어 동시검증 및 개발 과정에 최종 결과물을 대신할 수 있는 가상적인 칩 모델과 환경이 절대적으로 필요하고, 이것을 가상프로토타입이라 한다.

가상프로토타입은 하드웨어로 구축될 물리적 프로토타입을 대신할 수 있고, 많은 장점을 갖는다. 예로써 비교적 쉽게 만들 수 있고, 구조적 변형을 용이하게 적용할 수 있고, 모든 동작을 관측할 수 있으며, 어떤 동작이든 제어할 수 있다. 즉 모델링한 시스템의 동작을 관측할 수 있고 평가할 수 있으며, 하드웨어 제작 없이 소프트웨어 개발과 수행이 가능하다. 그러나 가상프로트타입이 소프트웨어 모델이기 때문에 성능이 충분하지 않을 수 있다는 점은 하드웨어 가속기 활용, 추상수준 높이는 방법, 관심의 대상이 아닌 동작들을 단순화시키는 방법 등을 통해 극복할 수 있다.

본 고에서는 IP 설계와 단일칩시스템 설계에서 가상프로토타입의 역할에 대해 살펴보았고, 실제적인 예로써 SystemC를 이용한 가상프로트타입을 만들어 그 유용성과 가능성을 설명하였다. 가상프로토타입은 향후 단일칩시스템 설계 및 검증에 유용하게 활용될 수 있는 접근법이 될 것이고, 이 접근법은 하드웨어와 소프트웨어를 동시에 설계하고 검증하는데 매우 유용한 환경으로 활용될 것으로 기대된다.

감사의 글

본 고가 발표될 수 있도록 기회를 주신 충북대학교 조경록 교수님께 감사드리며, V절 내용이 본 고에 포함되는 것을 허락하신 (주)다이나릿시스템의 이승종 박사님께 감사드립니다.

참 고 문 헌

[1] D. W. Knapp, *Behavioral Synthesis*

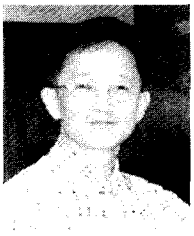
Digital System Design Using the Synopsys Behavioral Compiler, Prentice hall PTR, 1996.
[2] J. P. Elliott, *Understanding Behavioral Synthesis A Practical Guide to High-Level Design*, Kluwer Academic Publishers, 1999.
[3] Verilog, IEEE Std 1364-2001, IEEE Standard for Verilog Hardware Description Language.
[4] VHDL, IEEE Std 1076-2002, IEEE Standard VHDL Language Reference Manual.
[5] Verisity, e Language Reference Manual, 2002, http://www.verificationlanguage.org
[6] OpenVera, http://www.open-vera.com/
[7] TestBuilder, http://www.testbuilder.net/
[8] OSCI, SystemC Verification Library, http://www.systemc.org/
[9] Mentor Graphics, Co-modeling 10,000 times faster than your current software simulator, http://www.mentor.com/vstation/comodeling.html.
[10] SCE-MI, Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual, http://www.eda.org/itc/
[11] Ando Ki, iPROVE : A Block Design and Verification Platform, Dynalith Systems, 2003, http://www.dynalith.com.
[12] H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, L. Todd, *Surviving the SOC Revolution A Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999.
[13] CASTLE project at German National Research Center for Information Technology, http://set.gmd.de/EDS/SYDIS/castle/
[14] Chinook project at University of Washington, http://www.cs.washington.edu/research/chinook/index.html
[15] COSYMA project at Technical University of Braunschweig, http://www.ida.ing.tu-bs.de/research/projects/cosyma/home.e.shtml
[16] LYCOS project at Technical University of Denmark
[17] MEIJE project at INRIA, http://www.inria.fr/meije/
[18] PeaCE project at Seoul National University, http://peace.snu.ac.kr/research/peace/
[19] Police project at University of California at Berkeley, http://www-cad.eecs.berkeley.edu/~polis/
[20] Ptolemy project at University of California at Berkeley, http://ptolemy.eecs.berkeley.edu/
[21] Vulcan project at Stanford University
[22] D. Seal, *ARM Architecture Reference Manual*, 2nd Ed., Addison-Wesley, 2000.
[23] ARM Ltd., *AMBA Specification*, Rev. 2.0, http://www.arm.com
[24] SystemC 2.0. 1 Language Reference Manual, Rev. 1.0, OSCI, http://www.systemc.org
[25] RVARMulator, RealView Developer Suite-Instruction Set Simulators, http://www.arm.com/devtools/ISS
[26] Ando Ki, Cycle-Accurate Co-Emulation with SystemC, Dynalith Systems, 2003.
[27] iSAVE User Manual, Dynalith Systems, 2002.
[28] μCLinux, http://www.uclinux.org/
[29] Atmel Corporation, AT91 ARM

Thumb Microcontrollers, http://www.atmel.com
〔30〕 R. Stallman, R. Pesch, Debugging with GDB the GNU source-level debugger, Free Software Foundation, http://www.gnu.org/software/gdb
〔31〕 The ARMulator, Application Note 32, ARM DAI 0032E, ARM Limited, 2001.
〔32〕 ARM Limited, ARM Software Development Toolkit User Guide, http://www.arm.com

추가로 읽어볼 만한 자료

〔A〕 J. Staunstrup, W. Wolf, *Hardware/Software Co-Design Principles and Practice*, Kluwer Academic Publishers, 1997.
〔B〕 F. Balarin et. al., *Hardware-Software Co-design of Embedded Systems The POLIS Approach*, Kluwer Academic Publishers, 1997.
〔C〕 W. Muller, W. Rosenstiel, *SystemC Methodologies and Applications*, Kluwer Academic Publishers, 2003.
〔D〕 T. Grotker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, Kluwer Academic Publishers, 2002.
〔E〕 P. J. Ashenden, J. P. Mermet, R. Seepold, *System-on-chip Methodologies & Design Languages*, Kluwer Academic Publishers, 2001.
〔F〕 P. Rashinkar, P. Paterson, L. Singh, *System-on-a-chip Verification Methodology and Techniques*, Kluwer Academic Publishers, 2001.

저 자 소 개



기 안 도

1986년 한양대학교, 전자공학과, 학사, 1988년 한국과학기술원, 전기 및 전자공학과, 석사, 1997년University of Manchester, Department of Computer Science, Ph. D., 1989년 2월~1989년 7월 : University of Washington, Visiting Scholar, 1988년 3월~2000년 5월 : 한국전자통신연구원, 선임연구원, 2001년 4월~2001년 12월 : 중소기업청 기술혁신개발사업 에뮬레이터개발 과제, 과제책임자, 2002년 1월~2002년 12월 : 정보통신부 산업기술개발사업 가속기개발 과제, 과제책임자, 2003년 4월~2003년 12월 : 중소기업청 기술혁신개발사업 SystemC 가속기 개발 과제, 과제책임자, 2000년 6월~현재 : (주)다이나릿시스템. 부설연구소 연구소장, 〈주관심 분야 : SoC 설계 및 검증, 시스템 수준 설계, 하드웨어-소프트웨어 동시설계 및 동시검증, 컴퓨터구조, 병렬처리〉