특 집

SoC 설계 방법의 최근 동향

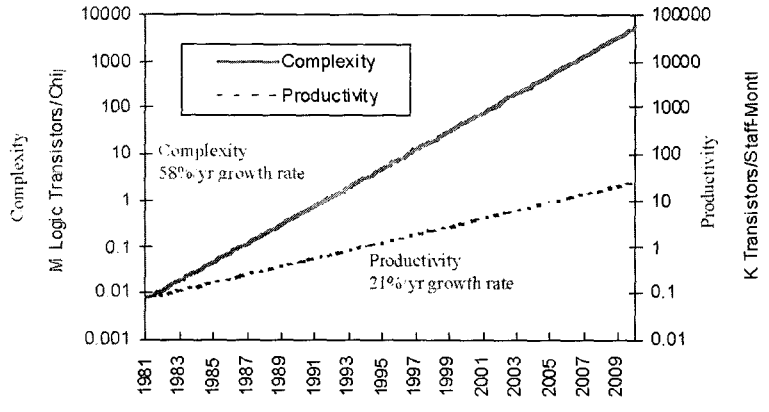
최 기 영, 조 영 철
서울대학교 전기컴퓨터공학부

I. 서 론

최근 많이 거론되고 있는 design productivity gap(시스템의 복잡도에 대한 요구 증가와 deep submicron 공정에 의한 시스템 용량 증가 속도를 시스템 설계 능력의 향상 속도가 따라가지 못하고 있는 현상)이 심화됨에 따라〈그림 1〉 지금까지와는 다른 훨씬 효율적인 SoC 설계 방법이 절실히 요구되고 있다. 이러한 문제를 해결하기 위해 학계뿐만 아니라 시스템 설계 관련 업계에서도 새로운 방법 모색을 위한 연구, 개발을 활발히 진행해오고 있다. 이와 같은 SoC 설계의 생산성에 관한 연구, 개발은 주로 design reuse의 방향으로 나타나고 있으며, 특히 IP-based design과 platform-based design이라는 방법론이 많은 주목을 받고 있다.

SoC에서 소프트웨어는 점점 그 중요성을 더해가고 있다. 그것은 주로 쉽게 개발하고 고칠 수 있는 소프트웨어의 유연성 때문이며, 또한 소프트웨어를 수행하는 프로세서의 급속한 성능 향상으로 하드웨어로만 가능하던 일이 소프트웨어로도 가능하게 되었기 때문이다. 이에 따라 프로세서가 점점 더 보편적으로 사용되고 있으며, 특히 응용에 따라 최적화 되어야 하는 SoC의 특성에 따라 다양한 종류의 프로세서 코어가 개발되어 SoC에 사용되고 있다.

많은 IP들이 하나의 SoC에 집적됨에 따라 이제는 단순한 on-chip bus로는 IP들 사이의 많은 통신량을 감당하기가 점점 어려워지고 있다. IP들의 수가 늘어날 수록 점점 더 문제가 되는 것은 computation 보다는 communication이 되는 것은 당연한 것이다. 주어진 platform에 새로운 IP를 추가하자면 새로운 bus interface



Source:International Technology Roadmap for Semiconductors: 1999

〈그림 1〉 Design productivity gap.

개발에 새로운 communication scheduling을 해야 하고 이는 IP의 수가 늘어감에 따라 점점 더 힘든 작업이 된다. 이러한 문제에 대한 대안으로 제시되고 있는 것이 network-on-chip이다.

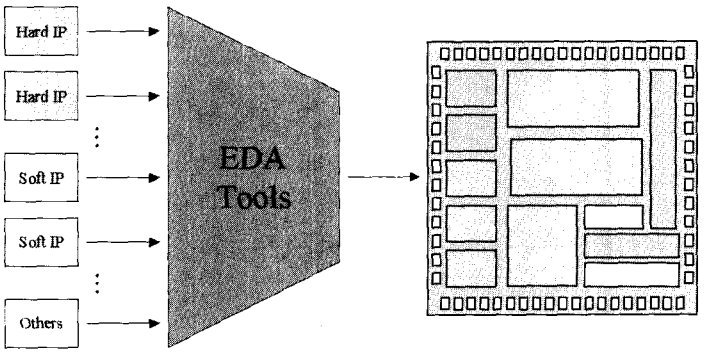
시스템이 복잡해지면서 다루어야 할 재사용 component들의 크기가 커지고 있다. standard cell이나 좀더 큰 macro-cell을 재사용하다가 요즈음에는 IP, platform을 재사용하고 나아가 IC 자체를 재사용하는 방향으로 시스템 설계 방법론이 발전하고 있다. 이와 같이 component의 크기가 커지면서 이제는 이를 이용한 시스템 설계가 예전과 같이 단순하지 않게 되었다. 몇 개의 port를 가진 cell을 다루던 문제가 이제는 수백 개의 port를 가진 IP를 다루는 문제로 발전하였다. 시스템 수준에서부터 체계적으로 설계해 내려가는 설계 방법론과 이를 뒷받침 해주는 설계 도구/환경의 도움이 절실해지고 있다.

경쟁력 있는 SoC의 설계를 위해서는 수많은 것들을 고려해야 하지만 지면 관계로 여기에서는 위에 언급한 IP/platform-based design, 내장형 프로세서, network-on-chip, 시스템 수준의 SoC 설계 도구/환경 등 몇 가지에 대해서 간단히 살펴보기로 한다.

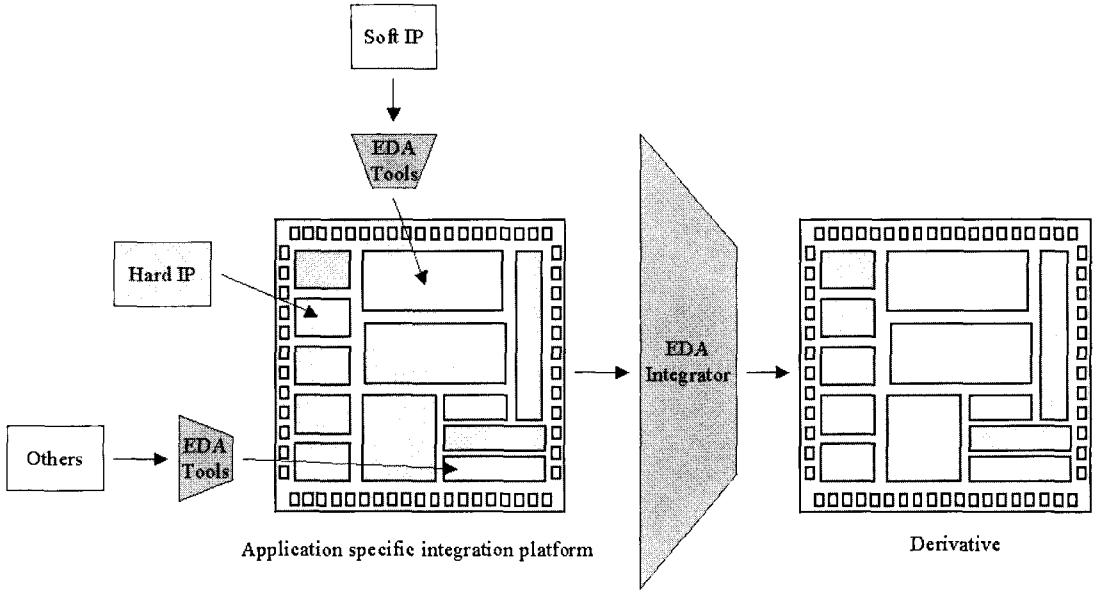
II. IP/Platform-Based Design

IP-based design은 IP(Intellectual Property; 예를 들어 무선 CDMA 이동전화의 경우, CDMA 모뎀을 하드웨어 블록으로 구현한다거나 vocoder를 프로세서에서 소프트웨어로 수행한다고 할 때, 그러한 하드웨어 블록이나 프로세서가 IP가 될 수 있다)를 기본적인 설계 단위로 보고 IP들을 bottom-up 형식으로 구성하여 전체 시스템을 설계하는 방법이다〈그림 2〉[1]. 이에 비해 platform-based design은 여러 시스템들에 대한 공통적인 아키텍처를 구현하는 platform을 가지고 있고 신제품들은 이러한 기본 platform에 새로운 블록을 추가하거나 일부를 변경하여 개발하는 방법이다〈그림 3〉[2].

IP-based design에서 설계자는 다양한 IP 제작/공급자가 제공하는 다양한 IP들 중에서 자신이 개발하려는 시스템의 성능 및 비용 등의 사양에 맞는 것들을 선택하여 시스템을 구성하게 된다. 이러한 IP들은 구현의 정도에 따라 soft IP, hard IP, firm IP 등으로 나뉜다. Soft IP는 VHDL이나 Verilog와 같은 언어로 표현되어 하드웨어를 합성할 수 있는 형태로 제공된다. Soft IP의 장점은 설계자가 원하는 target architecture나 library를 이용하여 하드웨어를 합성할 수 있다는 점이다. 이에 비해 hard IP는 layout의 형태로 제공된다. Hard IP의 장점으로는 timing에 대한 검증을 설계자가 수행할 필요가 없다는 점을 들 수 있고, 단점으로는 새로운 library로의 전환이 용이하지 않다는 점을 들 수 있다. Firm IP는 중간 형태로 게이트 수준의 netlist로 제공된다.



〈그림 2〉 IP-based design.



〈그림 3〉 Platform-based design.

IP-based design에서 설계자가 수행하는 주된 역할로는 구현하려는 IP들로 구성된 시스템의 기능적 동작을 검증하는 작업을 들 수 있다. 이와 함께 최적화된 시스템 성능을 얻고, 주어진 비용 제한 조건을 맞추기 위해 parameterize되어 있는 IP(예를 들어, cache를 IP로 보면 cache의 size, line size, associativity 등)들의 parameter를 찾는 작업이 필요하다. 특히, 이 작업은 광범위한 설계 공간 탐색이 요구되기 때문에 이를 자동화하려는 연구가 최근 활발히 진행되고 있다.

Platform-based design은 개념적으로 이미 설계했던 시스템에 대한 derivative system의 설계를 위해 기존의 많은 시스템 설계자들이 이미 사용해오던 방법을 좀더 체계화 한 것이라고 볼 수 있다. 최근에는 deep submicron 기술의 발전으로 충분히 넓어진 칩 위에 필요성이 예상되는 기능 블록들을 미리 최대한 집적시켜 놓고, 구현하려는 응용에서 필요로 하는 기능 블록만 활성화시키고 나머지 블록들은 사용하지 않는 방법들도 제시된다. 이러한 방법의 장점으로는 설계 시간을 줄인다는 것과 칩의 테스트 비용을 줄일 수 있다는 것 등을 들 수 있다.

더욱 다양해지고 고성능을 요구하는 응용 시스템을 더욱 짧아지는 시장 출시 시간 내에 설계해야 하는 현실적인 과제와 deep submicron으로의 공정 기술 발전으로 인한 기존 하드웨어 설계 방법론의 개편과의 사이에서 EDA(Electronic Design Automation) vendor들은 아직 IP-based design과 platform-based design에 대한 완벽한 설계 환경을 제시하지 못하고 있다. 이러한 상황에서 EDA vendor들은 시스템 개발 환경 구축이라는 소극적인 측면에서부터 design service를 통해 시스템의 일부를 대신 개발해주는 out-sourcing의 적극적인 측면에까지 시장 영역을 넓히고 있다. 이러한 현상의 한 예로, 최근 무선 통신 시스템 응용에 대한 급격한 시장 확대가 예측됨에 따라 EDA vendor들이 무선 통신 시스템 설계 회사에 대한 활발한 인수, 합병에 나선 것을 들 수 있다.

설계자의 생산성을 향상시키기 위한 방법으로 제안되고 있는 두 가지 방법론에 대하여 언급했으나 시스템 설계와 관련하여 해결해야 할 연구 과제는 수없이 많다. 현재 활발한 연구가 진행되고 있는 일부 분야들로는 설계에 대한 검증의 효

율성, 생산시의 테스트 문제, 전력 소비, interconnect 관련 문제, IP protection, 하드웨어-소프트웨어 통합 설계 등을 나열해 볼 수 있다. 어느 것 하나 중요하지 않은 것이 없다. 그러나 더욱 중요한 것은 이러한 다양한 요소를 얼마나 광범위하게 보고 잘 조화시킬 수 있는가 하는 문제라고 생각한다.

III. 내장형 프로세서

프로세서의 성능이 급속히 증가함에 따라 예전에는 하드웨어로 처리해야만 원하는 성능을 얻을 수 있었던 많은 작업들이 이제는 프로세서에서 수행되는 소프트웨어만으로도 다 처리할 수 있게 되었다. 따라서 SoC에서도 하드웨어에서 소프트웨어로 그 비중이 점점 옮겨가고 있는 추세이다. 소프트웨어는 많은 유연성을 제공하여, 오류가 있어도 쉽게 고칠 수 있게 해 준다. 오류가 있는 하드웨어는 많은 비용을 지불하며 리콜을 해야 하지만 소프트웨어는 오류가 많아도 업그레이드만 해 주면 그만이다. 심지어는 하드웨어에 있는 오류를 소프트웨어로 감추거나 피해갈 수도 있다.

SoC에는 하나 또는 그 이상의 프로세서가 사용되는데, 이에는 기존의 고정된 범용 프로세서가 사용되는 것이 일반적이다. 그러나 요즈음의 SoC와 같이 매우 제한된 소비 전력과 비용으로 최대의 성능을 낼 것을 요구하는 경우에 기존의 고정된 범용 프로세서로는 그 요구를 만족시키기가 점점 더 어려워지고 있다. 이에 대해 프로세서의 유연성과 소비 전력, 성능 사이의 trade-off를 고려하여, 응용에 따라 최적의 구조를 가진 프로세서를 선택하는 것이 하나의 해결 방법으로 대두되고 있다[3].

현재 사용되고 있는 내장형 프로세서들은 여러 가지로 분류해 볼 수 있다. 예를 들면 그것들이 사용되는 응용 범위에 따라 ARM(ARM), MIPS(MIPS), PowerPC(IBM, Motorola) 등의 범용 프로세서가 있고, TMS320x(TI), ADSP-TS0xx(Analog Devices), Saturn (Adelante), SH3-DSP(Hitachi), Carmel (Infineon), ST100(ST) 등의 DSP(Digital Signal Processor)가 있다. Xtensa(Tensilica), ARCtangent(ARC), Jazz(Improv) 등의 구성 가능한 프로세서(configurable processor)들은 어떠한 응용에도 사용할 수 있다는 면에서 범용 프로세서로 분류될 수도 있으나 대개 특정 응용에 맞추어 구성된다는 점에서 다소 다르다. 특히 이들은 DSP 응용에 맞게 구성되기도 한다.

프로세서 구조의 관점에서 보면, ARM, SH3-DSP, Xtensa, ARCtangent 등은 single issue RISC 구조로, MIPS, PowerPC 등은 Superscalar 구조로, TMS320C6x, ADSP-TS0xx, Saturn, Jazz 등은 VLIW 구조로 분류될 수 있다. 한편, ST100, Carmel 등은 하나의 프로세서에서 이러한 여러 가지 형태의 구조를 지원한다고 볼 수 있다.

또 응용에 따른 구성 가능성의 관점에서 분류해 볼 수도 있다. 예를 들면 프로세서에서 지원하는 ISA(Instruction Set Architecture)가 고정되어 있는가, 아니면 응용에 따라 구성을 다르게 할 수 있는가에 따라 구별할 수도 있고, 나아가 제조 공정을 거친 후에도 구성을 바꿀 수 있도록 해주는 재구성 가능 논리회로(reconfigurable logic)의 지원 여부에 따른 분류도 가능하다.

8051, ARM, MIPS, PowerPC, SH3-DSP, TMS320x, Saturn, Carmel, ST100과 같은 프로세서들은 고정된 프로세서(코어)에 해당된다. 프로세서 코어의 경우 그들은 고정된 레이아웃을 가진 하드 코어나 HDL로부터 합성 가능한 형태의 소프트 코어의 형태로 주어진다. 그러나 명령어 집합은 고정되어 있어서 응용에 맞게 구성하여 사용하도록 되어 있지는 않다.

시스템의 전체 성능을 높이기 위해서 이러한 고정된 프로세서와 함께 보조프로세서(coprocessor)가 사용되기도 한다. HP Pico 프로젝트에서와 같이 특정 응용에 적합한 보조프로세서를 자동 합성하는 접근 방법도 있다. 고정된 프로세

서를 재구성 가능한 논리회로와 함께 집적하여 사용하는 방법도 많이 상용화 되고 있다. Virtex II Pro(Xilinx), Excalibur(Altera), E5와 A7(Triscend), QuickMIPS(Quicklogic), RCP(Chameleon), FPSLIC(Atmel) 등이 그것이다. RCP는 구성 가능한 프로세서인 ARC를 포함하고 있으나, 이것은 미리 구성이 된 것으로 응용에 따라 그 구성을 바꿀 수 있는 형태로 제공되는 것은 아니다. MorphoSys(Morpho Technologies)도 고정된 프로세서와 재구성 가능한 블록이 함께 집적되어 있는 형태이다. 그러나 여기 사용되는 재구성 가능 블록은 일반적인 FPGA에서의 게이트나 LUT 보다는 크기가 훨씬 큰 단위(processing element)로 재구성할 수 있는 구조이다. 고정된 프로세서 코어가 합성 과정을 거쳐서 재구성 가능한 논리회로 상에 프로그램 될 수 있는 소프트 코어의 형태로 주어지기도 한다. MicroBlaze(Xilinx)가 그 예이다. 주변 장치나 버스 인터페이스에 대해 제한적인 구성 변경이 가능하나 명령어 자체는 고정되어 있다.

구성 가능한 프로세서(configurable processor)는 보통 응용에 따라 ISA를 다르게 구성할 수 있는 경우를 말한다. 이들은 때로는 ASIP(Application Specific Instruction set Processors)이라고 부르기도 한다. Xtensa, ARCtangent, Jazz 등은 일단 기본적인 코어와 명령어 집합을 제공한다. 이들은 응용에 따라 다르게 구성되거나 확장될 수 있도록 되어 있다. NIOS(Altera)는 구성 가능한 프로세서로서 재구성 가능한 논리회로 상에 프로그램 될 수 있는 형태로 제공된다. 그러나 구성을 할 수 있는 정도는 제한되어 있어서 단지 다섯 개의 opcode만이 사용자가 지정하는 명령어를 만드는데 사용될 수 있다. LISATek(CoWare에 병합됨)이나 Target Compiler Technologies는 기본 코어를 제공하지 않고 ADL(Architecture Description Language) 표현으로부터 전체 구조를 만들어내는 접근 방식을 취한다. 컴파일러와 시뮬레이터, 그리고 하드웨어 합성을 위한 HDL 표현 등도 자동적으로 생성한다. 이러한 방법은 응용에 최적화시킬 수 있다는 장점이 있으나 프로세서의 최적 구성을 찾아야 한다는 어려움 외에도 여기에서 수행될 소프트웨어와 이를 위한 모든 종류의 개발 도구들을 새로운 구성의 프로세서에 맞게 새로 만들어야 한다는 부담이 따르게 된다. 특히 이러한 것들은 추가의 시간을 필요로 하기 때문에 시장 진입 시간(time-to-market)에 대한 요구가 점점 더 강해지고 있는 것을 고려하면 심각한 문제가 될 수 있다. 따라서 이러한 방법을 채택하려면 고정된 범용 프로세서에서와 달리 손으로 설계하는 것을 가능하면 줄이고 자동화에 상당 부분 의존해야 한다.

모든 종류의 응용에 다 잘 맞는 프로세서라는 것은 있을 수 없다. 각각의 다른 응용에 대해 그것에 맞는 고정된 프로세서를 골라서 사용하거나 구성 가능한 프로세서를 응용에 맞추어서 사용해야 한다. 하나의 프로세서로는 충분하지 않을 수도 있다. 응용 대상의 규모가 급속도로 커짐에 따라 점점 더 많은 수의 프로세서 코어가 하나의 칩에 집적되어 사용되고 있다. 주어진 하나의 응용 대상 내에서도 하나의 칩에 집적되는 여러 개의 프로세서는 다양하게 서로 다른 일을 하게 되므로 각각 하는 일에 최적화 된 이질 다중 프로세서 SoC(heterogeneous multi-processor SoC)의 형태를 띠게 된다.

재구성 가능한 구조는 설계 재사용의 관점에서 매우 효과적이어서 같은 설계를 여러 다른 응용에 사용할 수 있다. 심지어는 새로 제조 공정을 거치지 않고도 IC 자체를 재사용할 수도 있다. 주어진 하나의 응용 내에서도 제한된 (재구성 가능한) 자원을 동적으로 재구성 해 가면서 시분할하여 다양한 작업에 계속 재사용할 수 있다. 구성 가능한 프로세서 기술과 재구성 가능한 구조를 결합할 수도 있다. 이 경우 프로세서의 구성은 재구성 가능한 블록을 이용해서 수행하게 되는데, 여기에서 재구성 가능한 블록은 보조프로세서의 형태가 될 수도 있고 datapath, 제어기, 또는 연결선의 일부가 될 수도 있다.

기존의 구성 가능한 프로세서는 대개 RISC 구조나 VLIW 구조로 되어 있다. 최근에 병렬

성(thread-level parallelism)을 더욱 많이 활용하기 위한 SMT(Simultaneous Multiple Threading) 구조가 대두되고 있는데, 이러한 SMT 기반의 프로세서를 응용에 맞게 최적화하는 것 또한 좋은 연구 대상이다.

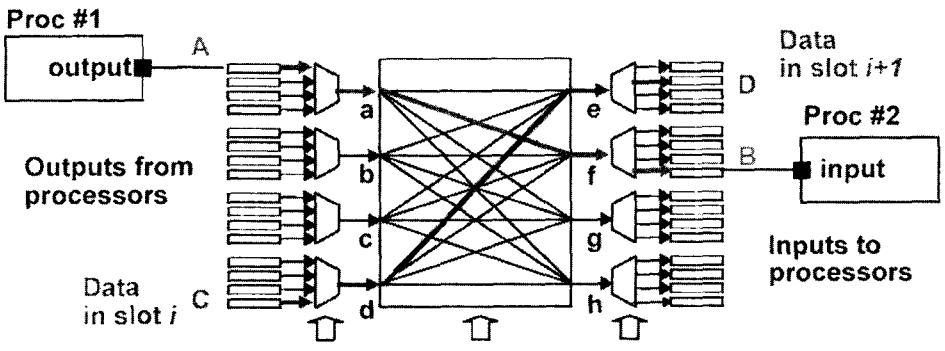
프로세서는 그 자체의 성능도 중요하지만 그것을 지원하는 개발 환경 또한 매우 중요하다. 특히 재구성 가능한 프로세서의 미래는 설계 공간 탐색 도구, 하드웨어 합성 도구, 컴파일러, 시뮬레이터, debugger, 검증 도구 등 하드웨어와 소프트웨어를 개발하는 데 필요한 제반 환경이 얼마나 잘 지원되는가에 달려있다.

Ⅳ. Network-on-Chip

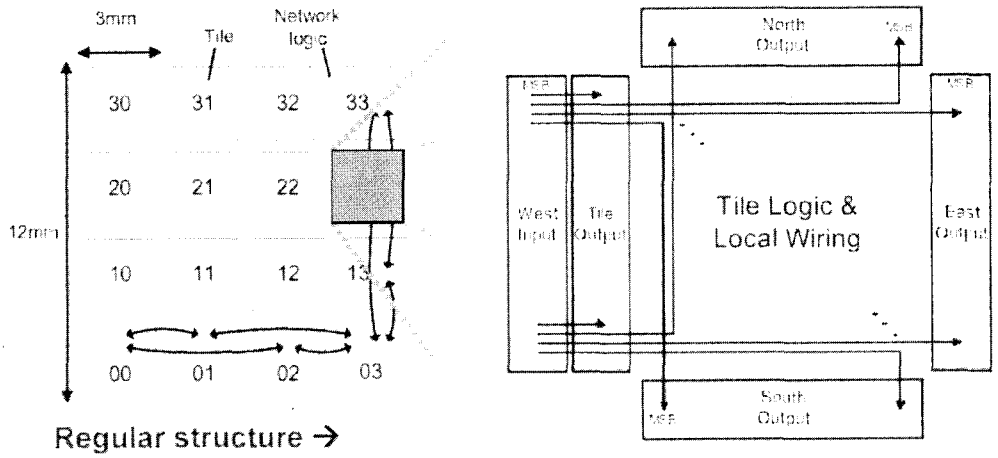
SoC에 집적이 되는 component의 개수가 증가함에 따라 각 component 사이에 communication을 해야 하는 데이터의 양이 늘어나게 된다. 앞으로의 SoC의 설계는 각각의 기능 블록을 설계하는 일보다는 여러 component들을 연결하는 on-chip communication의 설계가 대부분이라고 해도 과언이 아니다. 보다 힘든 on-chip communication requirement를 만족하기 위해서 성능이 더욱 좋은 on-chip bus를 개발하거나 혹은 circuit-switching network와 같은 networks-on-chip(NoC)에 대한 연구가 활발히 진행되고 있다.

On-chip bus는 arbitration 방법에 따라 AMBA(ARM), STbus(ST), CoreConnect(IBM)와 같은 priority에 기반을 둔 on-chip bus들이 있다. 그리고 uNetwork(Sonics)와 같이 TDMA 방식으로 arbitration을 하는 on-chip bus가 있다. 이 중 AMBA는 ARM 계열의 마이크로 프로세서가 SoC에 많이 사용됨에 따라 on-chip bus에서 de facto standard가 되었다. 그리고 많은 양의 데이터 전송이 필요 없는 제어 중심의 실시간 시스템에서는 CAN bus와 TTP bus같은 실시간 요구 조건을 만족시키는 버스들이 사용이 되고 있다.

NoC는 Philips에서 digital TV chip을 개발할 때 multi-window를 OSD로 구현을 하기 위해서 circuit-switch network 구조를 가진 PROPHID 아키텍쳐를 SoC에 적용함으로써 처음 소개가 되었다[4]. NoC가 SoC 설계에 적합한 이유로 scalability와 reusability를 들 수 있다. 큰 bandwidth를 유지하면서 새로운 component를 쉽게 network에 추가해 나갈 수 있다. 또한 같은 network protocol과 이를 구현하는 하드웨어, 소프트웨어를 계속 재사용함으로써 쉽고 빠른 SoC 설계를 가능하게 해준다. 특히 공정 기술이 deep sub micron으로 가면서 routing wire의 timing 검증이 어려운데, timing이 이미 검증된 NoC를 재사용함으로써 이러한 어려움을 줄일 수 있다.



〈그림 4〉 PROPHID circuit-switch network의 구조.



〈그림 5〉 Regular structure packet-switch network의 구조.

NoC는 두 가지 구조로 나누어 볼 수 있다. PROPHID(Philips)와 같은 circuit-switch network 구조와 Regular architecture[5]와 Octagon architecture[6], SPIN architecture[7]와 같은 packet-switch network 구조가 있다. Circuit-switch network 구조는 QoS에 관련된 실시간 응용에 적합하다. Packet-switching network 구조는 utilization이 높고, 다양한 network traffic에의 적용이 쉬우므로 양이 많고 다양한 종류의 데이터를 처리해야 하는 응용에 적합하다. 〈그림 4〉는 PROPHID circuit-switch network의 구조를 보여준다. 여기에서 각 time slot에 input과 output사이의 connection을 scheduling하여 최고의 성능을 내거나 데이터 수송량을 보장을 해주는 라우팅을 할 수도 있다. 〈그림 5〉는 일반적인 packet-switch network 구조를 가지는 Regular architecture를 보여준다. 이러한 타일 기반의 구조에서 라우터의 area 오버헤드는 약 6% 정도이다.

앞에서 NoC의 하드웨어 구조에 대해서만 설명을 하였지만, NoC는 하드웨어 인프라만을 의미하는 것이 아니라 OS 통신 관련 API, 미들웨어, 설계 방법론, 설계 도구 및 환경을 모두 포함하는 NoC platform의 개념으로 보아야 할 것이다. NoC의 설계와 관련해서 설계 방법론 연구, 소프트웨어 환경 및 시뮬레이션에 관한 연구, 저전력 설계에 관련된 연구 등 여러 분야의 연구들이 행해지고 있으며, 앞으로 SoC 설계에서 NoC는 떼어 놓을 수 없는 중요한 요소임에는 틀림이 없다[8].

V. System-Level SoC Design Environment

시스템 수준의 SoC 설계란 하드웨어 구조를 고려하지 않은 상위 수준에서의 설계를 한 뒤 추상화 수준을 낮추어 가는 top-down 형식의 설계 방법이다. 추상화 수준을 낮추는 과정은 시스템 구조를 결정하고 필요한 component를 선택하는 과정과 상위 수준의 기능 블록을 시스템 구조에 맞게 분할 할당하는 과정을 의미한다. Component를 선택하는 과정은 필요한 IP 선택뿐만 아니라 platform-based design을 할 경우 적당한 platform을 선택하고 기본 platform에 추가 혹은 제거해야 하는 블록 결정하는 것을 포함해 derivative system을 결정하는 과정이 된다.

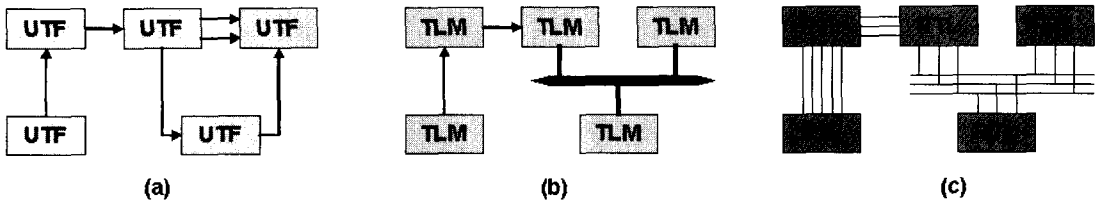
시스템 수준의 설계에서 보는 추상화 수준은 크게 메시지 수준, transaction 수준, register transfer(RT) 수준으로 나눌 수 있다. 메시지 수준 모델은 timing 정보가 없는 추상화 수준(UTF : un-timed functional model)을 의미

한다. 즉, 메시지 수준 모델은 각각의 기능 블록이 일대일로 연결되어 있고 각 블록들 사이의 통신은 function call을 통해 이루어진다. Transaction 수준 모델(TLM)은 버스 기능 모델(BFM : bus functional model)을 통해 구현이 된다. 여기서 BFM은 버스의 기능을 비트나 시그널 수준의 모델이 아닌 좀더 추상화된 모델(예를들어 SystemC의 interface method call[9])로 구현되며, 메시지 수준 모델보다는 구체화된 통신 프로토콜을 가진다. Transaction 수준 모델은 BFM을 사용하여 기능적인 면만을 고려한 모델과 timing 정보를 추가한 cycle accurate 모델로 나눌 수 있다. Cycle accurate 모델은 하드웨어-소프트웨어 통합 시뮬레이션에서 거의 같은 정확도를 가지면서 RT 수준의 모델 보다 수십 배 빠른 시뮬레이션이 가능하다. Register transfer 수준 (RTL) 모델은 비트나 시그널 수준(게이트 수준)으로 합성이 가능한 HDL로 구현된 모델을 의미한다. 이 수준의 모델은 가장 구체적이고 정확한 모델이지만 구현 및 검증에 많은 시간과 노력이 필요하다. Transaction 수준은 시뮬레이션 속도만을 고려하고 합성에 대해서는 고려하고 있지 않아, transaction 수준에서 RT 수준으로 구체화하는 합성이 어렵다. 최근 학계 및 EDA vendor들의 동향은, 설계 과정에서 추상화 수준을 낮은 수준으로 구체화해 나갈 때 합성을 하여 추상화 수준을 낮추기 보다는 각 추상화 수준에 해당하는 모든 설계물을 제공하여 설계자가 필요한 수준에 해당하는 설계를 라이브러리에서 가져와 쓸 수 있도록 한다. 예를 들어, 시스템 수준의 EDA 설계 도구, CoCentric (Synopsys)과 ConvergenSC(CoWare)에서는 on-chip 버스인 AMBA (ARM)를 ARM에서는 TLM 모델과 RTL 모델을 모두 제공한다. <그림 6>은 메시지 수준 모델(UTF), TLM, 그리고 RTL을 개념적으로 보여준다.

일반적인 시스템 수준에서의 SoC 설계 과정은 1) 메시지 수준으로 응용을 표현하여 기능적 명세를 만족하는지 검증한다. 2) 필요한 component를 선택하고, 시스템의 구조를 결정하여, 3) 메시지 수준에서 기술된 기능 블록들을 시스템 구조에 맞게 분할/할당한다. 4) 시스템이 timing 제한을 만족하는지 검증을 한다. 5) 마지막으로 timing 제한을 만족하지 못하거나 혹은 보다 최적에 가까운 구조를 찾을 필요가 있는 경우 2, 3, 4과정을 반복한다.

효과적인 SoC 설계를 위해서 시스템 수준의 SoC설계 과정에서 가장 중요한 부분은 2와 3 단계이다. 설계자는 경험에 미루어 임의로 component를 선택하고 구조를 결정하고 분할/할당을 한다. 하지만, 한번 선택된 component는 쉽게 바꾸기 어려우며, 시스템의 구조를 여러 번 바꾸기란 불가능하다. 그리고 최적의 구조를 결정했다고 하더라도 실제 구현이 어려워 쉬운 구현을 택하는 경우도 있다. 일부 설계 도구에서는 이러한 과정을 자동으로 해 주기도 하지만, 대부분은 설계자가 수동적으로 결정하게 된다. 현재 대부분의 설계 도구들은 설계자의 결정에 도움을 주는 빠른 시뮬레이션과 profiling에 초점을 맞추고 있다.

CoCentric(Synopsys), ConvergenSC(CoWare), VCC(Cadence), PlatformExpress (Mentor Graphics) 등의 대부분의 시스템 수준 설계 도구들에서는 정형화된 기술 모델을 사



<그림 6> (a) Un-timed functional model (b) Transaction level model (c) Register transfer model.

〈표 1〉 시스템 수준 설계 도구

회사명	제품명	특징
Synopsys	COSSAP	디지털 신호처리용 설계 최적화 도구
	Eaglei	HW/SW cosimulator
	CoCentric	가상 프로토타이핑 도구
Cadence	SPW	디지털 신호처리용 설계 최적화 도구
	VCC	component 매핑 및 성능 분석 도구
HP	EEsof	무선단말기 설계 최적화 도구
iLogix	STATEMATE MAGNUM	제어 응용 시스템 최적화 도구
CoWare	N2C Design System	가상 프로토타이핑 도구
	ConvergenSC	가상 프로토타이핑 도구
Mentor Graphics	SeamlessCVE	HW/SW cosimulator
	Platfrom Express	가상 프로토타이핑 도구

용한다. 즉, 소프트웨어 하드웨어를 모두 기술하여 하나의 검증 환경에서 빠른 시뮬레이션을 통해 하드웨어, 소프트웨어의 기능을 검증하고 매우 넓은 설계 공간 탐색을 빠르게 해주는 가상 프로토타이핑 환경을 제공한다. 이러한 가상 프로토타이핑 환경을 이용하면 실시간 시스템도 하드웨어가 만들어지기 이전에 시뮬레이션을 통해 검증할 수도 있다. 〈표 1〉은 이러한 설계 환경을 제공하는 설계 도구를 보여준다.

시스템 수준의 기술을 위한 정형화된 기술 모델로는 디지털 신호처리를 효율적으로 모델링 할 수 있는 data flow(DF) model, 컨트롤이 많은 시스템을 기술하기 위한 finite state machine(FSM) model, event-driven simulation에 적합한 discrete event model, synchronous reactive system model 등이 있다. 이러한 모델들은 데이터 집중적 시스템 혹은 컨트롤 집중적 시스템만을 기술을 할 수 있었고, 시스템의 병렬성(parallelism), 계층 구조(hierarchy) 등을 기술하는데 한계가 있어 이러한 모델들을 확장하여 finite state machine with data flow(FSMD), 계층 구조를 고려한 hierarchical finite state machine(HFSM), StateChart, 소프트웨어의 특성을 고려한 codesign finite state machine(CFSM) 등 다양한 모델이 소개되었다.

시스템 수준의 설계를 위해서는 이를 기술하기 위한 언어도 필요하다. 기존에 디지털 시스템을 설계할 때 응용의 명세를 검증하는 데는 C/C++, 하드웨어를 기술하기 위해서는 VHDL이나 Verilog와 같은 언어를 사용한다. C/C++는 순차적 수행을 기본으로 하기 때문에 병렬성을 표현하기 어렵고, VHDL이나 Verilog는 알고리즘을 구현 하기에는 적합하지 않다. 즉, 이러한 언어들은 표현력에 한계가 있어 구체적인 구현과 상관 없이 시스템 전체를 표현하기에는 부적합하다. 시스템을 표현하기 위한 언어 들로는 SystemC(OSCI), SpecC(UCI), CoWareC(CoWare) 등의 언어가 있다. 이들 언어들은 C에 기반을 두고 있지만 소프트웨어뿐만 아니라 하드웨어까지 기술할 수 있도록 확장을 한 것이다. 기존의 C를 이용하던 설계자들이 쉽게 배울 수 있고 표현력이 좋다는 장점이 있기는 하지만, 합성의 관점에서는 표현에 제한이 많다. 이러한 언어로부터 하드웨어를 합성할 경우 VHDL이나 Verilog에 비해 모든 표현을 다 합성할 수 있는 것이 아니기 때문에 합성이 어렵고, 소프트웨어 합성의 경우 합성 결과가 설계자에 의해 직접 설계된 것보다 코드 크기 및 속도 면에서 좋은 결과를 얻기 어렵다.

VI. 결 론

여기에서는 SoC 설계 방법의 최근 동향을 단편적으로 살펴보았다. 특히 관심이 집중되고 있다고 생각하는 것들, 즉 설계의 생산성 제고를 위한 재사용 방법으로서의 IP나 platform 기반의 설계 방법론, 다양하게 발전하고 있는 내장형 프로세서들, communication 문제 해결을 위한 대안으로서의 network-on-chip, 시스템 수준에서의 설계 방법과 도구들 등에 대해 간단히 알아보았다.

SoC 설계에서는 기존의 ASIC 설계나 소프트웨어 설계와 달리 시스템의 차원에서 문제를 종합적으로 해결하는 것이 필수적이다. 이제는 부분적인 지식만을 가지고 시스템을 설계하는 것이 점점 어려워지고 있다. 응용 자체의 알고리듬부터 이해하고, 이의 구현을 위한 시스템 표현 방법, 응용 소프트웨어, compiler, middleware, OS, driver, multi-processor architecture, 하드웨어 모델, IP/platform architecture, synthesis, network/bus, bus interface, low power design 등 시스템 설계에 대한 전반적인 지식을 갖추고 있는 시스템 아키텍트의 역할이 점점 중요해지고 있는 것 같다.

참 고 문 헌

〔1〕 M. Keating and P. Bricaud, Reuse Methodology Manual for System-on-a-Chip Designs, Kluwer Academic Publishers, 1999.

〔2〕 H. Chang, L. Cooke, M. Hunt, G. Martin, A. McNelly, and L. Todd, Surviving the SOC Revolution-A Guide to Platform-Based Design, Kluwer Academic Publishers, 1999.

〔3〕 N. Dutt and K. Choi, "Configurable processors for embedded computing," IEEE Computer, vol.36 no.1, pp.120-123, Jan. 2003.

〔4〕 J. Leijten, J. Van Meerbergen, A. Timmer, and J. Jess, "Stream communication between real-time tasks in a high-performance multiprocessor," in Proc. of Design Automation and Test conference in Europe, pp.125-131, Mar. 1998.

〔5〕 W. J. Dally and B. Towles, "Route Packet, not wires: on-chip interconnection networks," in Proc. of Design Automation Conference, pp.684-689, June 2001.

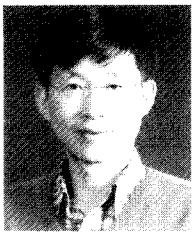
〔6〕 F. Karim, A. Nguyen, S. Dey, R. Rao, "On-chip communication architecture for OC-768 network processors," in Proc. of Design Automation Conference, pp.673-677, June 2001.

〔7〕 P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in Proc. of Design Automation and Test in Europe, pp.250-256, Mar. 2000.

〔8〕 A. Jantsch, Hannu Tenhunen, Networks on Chip, Kluwer Academic Publishers, 2003.

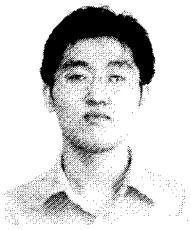
〔9〕 T. Grotker, S. Liao, G. Martin, and S. Swan, System Design with SystemC, Kluwer Academic Publishers, 2002.

저 자 소 개



최 기 영

1978년 2월 서울대학교 전자공학과 학사 1980년 2월 KAIST 전기및전자공학과 석사, 1989년 6월 Stanford University Electrical Engineering 박사, 1989년 8월~1991년 1월 : Cadence SMTS, 1991년 3월~현재 : 서울대학교 전기컴퓨터공학부 교수, 〈주관심 분야 : CAD, SoC 설계, 저전력 설계〉



조 영 철

1999년 2월 KAIST 전기및전자공학과 학사, 2001년 2월 서울대학교 전기컴퓨터공학부 석사, 2001년 3월 서울대학교 전기컴퓨터공학부 박사 과정, 〈주관심 분야 : CAD, SoC 설계〉