

SRAM 형 FPGA를 이용한 Rapid Prototyper 개발

申 鉉 哲*, 柳 泳 翩**
*漢陽大學校 電子工學科, **(주)서두로직

I. 서 론

본 고에서는 먼저 에뮬레이션의 기본 개념과 기준의 에뮬레이터를 간단히 설명하고, 본 저자들이 개발한 새로운 구조의 에뮬레이터를 소개한다. 새로운 에뮬레이터는 고속 에뮬레이션이 가능하도록 동작속도 향상에 역점을 두었으며, FPGA 칩들을 연결하는 network 구조와 각각의 FPGA 칩에 회로를 분할, 배치하는 방법을 새롭게 개발하였다. 그리고 실제로 에뮬레이션을 수행한 몇가지 예를 기술하고, 에뮬레이터를 하드웨어 accelerator로 이용하여 하드웨어-소프트웨어 codesign에의 활용 전망에 대하여 소개한 후 결론을 맺는다.

최근에 IC 설계 및 공정 기술이 급격히 발달함에 따라 설계 시스템의 규모가 커지고 복잡해지는 추세이며, 시장에서의 경쟁은 더욱 치열해지므로 빠른 시간 내에 우수한 제품을 개발하지 않으면 경쟁력을 확보할 수 없다. 그러므로, 빠른 시간 내에 효율적으로 설계된 회로를 구현하고 검증하기 위한 로직 에뮬레이터(logic emulator)의 필요성이 증가하고 있다. 로직 에뮬레이터는 프로그램 가능한 칩들을 사용하여 설계된 회로를 실제 제작한 칩과 동일한 동작을 하는 하드웨어로 구현 가능하도록 개발된 시스템이다. 로직 에뮬레이터를 이용하면 설계 회로의 검증, prototyping 등을 빠른 시간 내에 수행할 수 있다는 장점이 있다. 최근에는 여러 가지 DSP 시스템, CPU 등의 에뮬레이션을 성공적으로 수행하는 등, 여러 분야에서 널리 사용되고 있으며, 앞으로의 수요는 더욱 증가할 것이다.

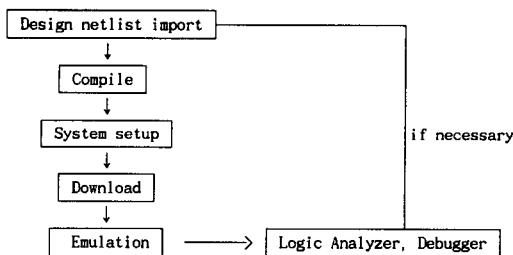
설계 중인 IC의 검증이외에도 에뮬레이션은 시스템의 동작확인에 매우 중요하다. 예를 들면, CPU의 에뮬레이션을 이용하면, CPU칩의 제조 이전에도, 응용프로그램의 수행과 interrupt 수행기능 또는 여러 사용자에 의한 다양한 부하 하에서의 CPU의 동작을 점검할 수 있고, 소프트웨어 등 CPU 이외의 부분에 존재하는 오류도 검증할 수 있도록 도와준다.

에뮬레이터는 간단하게 다음과 같은 3 가지 요

소로 이루어진 시스템으로 정의할 수 있다.

- 하나의 FPGA 보다 많은 게이트 용량을 가지는 프로그램 가능한 하드웨어.
- 게이트 단위의 설계를 하드웨어에 프로그램하여 구현하는 소프트웨어.
- 회로를 에뮬레이션하고 회로의 동작을 해석할 수 있는 하드웨어 및 소프트웨어.

로직 에뮬레이터를 이용한 에뮬레이션의 순서도는 <그림 1>과 같다. 그림과 같이 먼저 회로를 설계한 다음 에뮬레이션 시스템에 입력시킨다. 입력된 회로는 에뮬레이터의 회로 기술 형식으로 변환되고, 논리 및 기술매핑의 최적화가 수행된 후 컴파일된다. 컴파일러는 에뮬레이터의 구조에 맞는 회로의 분할 및 매핑을 수행하고, 배치 및 배선을 수행한다. 컴파일이 성공적으로 완료되면 에뮬레이션을 수행할 수 있도록 에뮬레이터와 컴퓨터, 응용 보드 등을 연결한다. 그리고, 설계된 회로를 에뮬레이터에 프로그램하여 에뮬레이터가 설계한 회로와 같은 동작을 하도록 한다. 에뮬레이션을 수행하면서 결과를 논리 분석기, 디버깅을 위한 장치 등으로 관찰하여 에러가 있는 경우에 이를 수정하게 된다. 수정된 회로는 같은 과정을 반복하여 검증이 완료될 때까지 수행한다.



<그림 1> 에뮬레이션 순서도

설계한 시스템을 검증하는 방법에는 시뮬레이션, prototyping, 에뮬레이션 등의 여러 가지 방법이 있다. 이중에서 로직 에뮬레이션을 수행하면 다음과 같은 여러 가지 장점이 있다.

- IC가 만들어지기 전에 전체 시스템 설계의 검증을 수행할 수 있다. 즉, 회로의 설계가 완료되면 바로 에뮬레이터를 이용하여 하드

웨어로 구현한 후, 주변장치와 연결하여 동작 시킬 수 있다. 그러므로, 설계에서 제품화까지의 시간을 크게 단축할 수 있고 개발비용을 낮출 수 있다.

- 설계 초기에 미리 하드웨어적인 검증을 수행하므로, 설계의 최종 단계에서 설계 변경을 줄일 수 있어서 설계의 질을 향상시킬 수 있다.
- 여러 설계 대안을 에뮬레이션을 통하여 비교하여 사양이 critical한 설계의 질을 높일 수 있다.
- 많이 사용되는 소프트웨어적인 시뮬레이션에만 의존하는 설계 검증 방법에 비하여 다양하고 복잡한 검증을 수행할 수 있는 능력을 가지므로, 우수한 검증을 수행할 수 있다. 예를 들면 interrupt 또는 통신에서의 congestion 등을 실험할 수 있으며, 설계된 시스템의 사양에 에러가 존재하는 경우에도 이를 쉽게 찾아서 수정할 수 있다.
- 시스템의 복잡도가 증가함에 따라 검증을 수행 할 수 있는 유일한 방법이다. 다른 검증 방법과는 달리 에뮬레이션은 거의 실시간으로 동작이 가능하므로, 설계의 복잡도가 증가하더라도 수행에 큰 어려움이 없다.
- 일부 설계에서는 검증에서 실시간 동작이 필요할 경우가 있다. 예를 들면 실제 관찰에 의해서 검증하는 것이 쉬운 표시 장치, 아날로그 하드웨어와의 연결이 페루프를 형성하고 있는 회로, 기계 및 전자 시스템이 결합되어 있는 경우, 그리고 오디오나 비디오 등 사람의 지각에 의한 검증이 필요한 경우 등에 대해서는 에뮬레이션에 의하여 실시간으로 동작을 확인하는 것이 유용하다.
- 설계된 회로를 칩으로 제작하여 응용 시스템에 장착하여 동작 실험을 수행하였을 경우에 원하는 대로 동작하지 않는 경우, 디버깅을 위하여 내부 신호의 관찰이 불가능하다. 그러나, 에뮬레이션에서는 내부 신호를 프로그램에 의하여 쉽게 관찰 가능하게 배선을 변경할 수 있으므로 유용하다.

그러나, 현재에 에뮬레이션을 수행하는데 다음과 같은 몇 가지 단점이 있다.

- 고가의 하드웨어적인 에뮬레이션 시스템이 필요하다. 에뮬레이터는 고가의 FPGA 칩 등을 사용하여 구현하므로 시스템이 고가이다.
- 에뮬레이션의 수행 시간이 실제 제작된 시스템의 동작 속도에 비하여 5~10배 느린다. 현재 상용화되는 에뮬레이터의 동작 속도는 1~4MHz 이므로, 에뮬레이션을 수행하기 위하여 응용 시스템의 동작 속도도 이에 맞추어 느리게 동작시켜야 하는 어려움이 있다.
- 에뮬레이션은 FPGA 칩에 프로그램에 의하여 회로를 구현하여 수행하므로, 수행시 지연시간이 실제 설계의 지연시간과 다르다. 그러므로, 지연시간이 중요한 설계에 대하여 정확히 동작하지 않을 수 있다. 예를 들어 hold-time violation 등의 어러가 발생할 수 있다. 그러나, 대부분의 설계에서 ASIC 칩으로 제작하면 에뮬레이터보다 훨씬 빠른 속도로 동작하므로, 에뮬레이션을 성공적으로 수행하면 ASIC 칩으로 제작하여도 큰 문제가 없다. 현재에 많이 사용되는 설계 검증의 방법은 시뮬레이션, 에뮬레이션, prototyping 등의 방법이 있다. 시뮬레이션은 소프트웨어에 의하여 순차적으로 수행되므로, 실제 상황의 일부분만을 관찰할 수 있다. 시뮬레이션을 위하여 먼저 주변 환경의 소프트웨어적인 모델을 만들어야 한다. 그러나, 소프트웨어적인 모델만을 이용하여 실제 상황을 표현하기에는 여러 어려움이 있다. 소프트웨어 모델을 완성한 후, 소프트웨어 알고리듬에 따라서 입력되는 테스트 벡터에 대한 동작을 검증한다. 시뮬레이션의 가장 큰 단점은 많은 시간이 소요되는 것이다. 예를 들어, 설계된 회로의 크기가 2배가 되면 4배로 일의 양이 증가하는 경향이 있다.

에뮬레이션은 하드웨어적으로 수행되기 때문에 병렬로 처리된다. 그리고 주변 하드웨어와 연결하여 수행되므로, 시뮬레이션과 같은 모델을 만들 필요가 없다. 시뮬레이션과 에뮬레이션을 비교하는 경우의 가장 큰 차이점은 검증에 소요되는 시간이다. 에뮬레이션은 실제 하드웨어에 비하여 10배정

도 느리지만, 시뮬레이션에 비하여 10^6 배정도까지 빠르다.

Prototyping은 설계한 회로를 실제적인 하드웨어로 구현하여 성능을 평가하는 것이다. 일반적인 prototyping 방법은 제작하고 디버깅을 수행하는데 있어서 많은 시간과 비용이 소모된다. 로직 에뮬레이터를 사용하여 prototyping을 수행할 수도 있는데, 에뮬레이터를 이용한 prototyping의 장점은 하드웨어를 프로그램에 의하여 구현하므로, 구현 및 디버깅이 용이하므로 구현의 시간을 크게 줄일 수 있는 것이다. 그러나, 빠른 동작 속도를 가지는 DSP 칩 등 실시간 검증이 필요한 경우에는, 에뮬레이터를 이용하면 원하는 동작 속도를 얻기가 어려우므로 실제 prototyping을 수행하여야 한다.

에뮬레이터는 하드웨어 accelerator로 확장하여 하드웨어 소프트웨어 통합설계 시스템으로 이용 가능하다. 에뮬레이터를 기준의 mixed-level 시뮬레이터나 gate-level 시뮬레이터 등과 연결하여 하드웨어 accelerator로 이용하면 회로에서 많은 시간이 소요되는 부분을 하드웨어로 구현하므로 시뮬레이션의 속도를 수백 배까지 높일 수 있다. 회로를 하드웨어로 구현하면 속도가 빠르고 병렬처리 등이 용이하므로, 소프트웨어로 CPU에서 순차적으로 수행되는 것에 비하여 상당히 빠른 속도를 얻을 수 있기 때문이다. 그러나, 에뮬레이터를 이용하여 하드웨어 accelerator로 구현하기 위해서는 하드웨어-소프트웨어의 분할, 인터페이스 등을 최적화하는 기술이 필요하다.

II. 기존의 에뮬레이터 소개

본 장에서는 현재까지 개발되어 사용되고 있는 에뮬레이터에 대하여 소개한다. 먼저, 상용화되어 많이 사용되는 MARS와 Quickturn사의 제품 등에 대하여 기술하고, 대학 등에서 개발되어 발표된 에뮬레이터 등에 대하여 기술한다.

MARS 「1」는 비교적 초기의 에뮬레이터로 PIE

사에서 개발하였다. MARS II는 MARS를 개선하여 Xilinx 사 「2」의 4005 칩을 사용하였고, 8MHz 의 애플레이션 속도와 자동화된 컴파일러, 그리고 디버깅을 위한 환경을 제공한다. MARS II는 크게 소프트웨어 요소와 하드웨어 요소로 나눌 수 있다. 소프트웨어 요소는 애플레이션 커널(Kernel), 컴파일러, Function 테스터, Analyzer 등으로 구성되며, 하드웨어 요소는 Xilinx 사의 4005를 사용하여 25K 게이트까지 구현가능하고 720 개의 I/O 핀을 가지는 LBM (Logic Block Module)과 애플레이션 서버, 논리 분석기, function 테스터의 역할을 하는 DebugWare 및 데이터와 클럭을 외부 시스템과 연결하는데 사용되는 Pods 등이다. MARS III는 Quickturn 사에서 MARS II를 개선하여, 10 만 게이트에서 100만 게이트까지 확장 가능하도록 개발한 것이다.

Enterprise Model 330 「3」은 11개의 독립적인 애플레이션 모듈을 연결 가능하도록 개발하여 33 만 게이트의 크기를 가지는 회로의 애플레이션이 가능하다. 또한 32개의 메모리 애플레이션 모듈을 지원하므로 multi-port RAM을 포함하여 64 Mbyte의 용량을 구현할 수 있다. 또한 소프트웨어를 개선하여 매우 복잡한 회로의 구현도 가능하도록 하였고, EDA 시스템 통합 툴들이 빠르고, incremental한 설계의 수정이 용이하도록 지원한다. 컴파일러에서 입력회로의 기술 매핑을 먼저 수행하고, 시스템 분할을 수행한 후, 시스템 단계의 연결을 수행한다. 그리고, 각 FPGA 칩 및 연결 하드웨어를 위한 컴파일을 수행한다. 마지막으로, 컴파일된 결과를 통합하여 타이밍을 분석하는 시스템도 있다.

Realizer 「3」는 Quickturn 사에서 개발한 애플레이터로 partial crossbar 연결 구조 「4」를 사용하여 만든 제품이다. Partial crossbar 연결 구조를 사용하여 논리 구현을 위한 FPGA와 연결을 위한 FPGA를 별도로 사용하여 FPGA 칩의 이용률이 최대가 되도록 하였고, 일정하며 적은 지연 시간을 가지도록 하였다. 이 방법은 시스템 단계의 복잡도를 줄일 수 있고, FPGA 간의 연결에서 비교적 일정한 지연시간을 가진다. 그리고, FPGA의 핀의

수에 비례하여 시스템의 규모를 크게 만들 수 있다. 또한, 시스템이 커지는 경우에 같은 연결 구조로 지연시간이 크지 않게 계층적으로 확장이 가능하다.

System Realizer 「3」의 M6000 모델은 대규모 시스템도 구현 가능하도록 6 백만 게이트까지 확장 가능하도록 개발하였다. 즉, 4-8 MHz의 동작 속도를 가지는 250 K 게이트를 가지는 M250 애플레이터 모듈 24개를 연결하여 만들 수 있는 modular한 package를 가진다. M6000은 1-4 MHz의 동작 속도를 가진다. 모듈간의 연결을 위하여 Xilinx 4013 및 자체 제작한 연결 칩을 사용하였다. 컴파일러에서는 RTL 단계의 Verilog 또는 VHDL을 FPGA에 매핑할 수 있도록 개발하였으며, 설계 변환을 빠르게 수행할 수 있도록 하였다.

Aptix 사에서 개발한 시스템은 빠른 설계 구현을 위한 시스템으로 연결을 위한 칩인 FPIC를 이용하여 프로그램 가능하도록 제작한 보드인 FPCB를 개발하였다. FPCB 구조는 FPGA 등의 칩을 위한 Free holes이 만들어져 있다. Free holes은 FPIC와 연결이 되어 있으므로, 여기에 칩을 장착하고 FPIC를 프로그램하여 원하는 연결을 수행한다. Aptix FPIC는 어레이 구조를 가진다. 이러한 FPCB 구조를 가지는 애플레이터인 MP-3 FPCB 가 개발되었다. MP-3 FPCB는 FPGA와 일반용도의 IC, 그리고 DSP 칩 등을 연결하여 시스템을 구현 가능하도록 개발되었다. 입력 및 출력을 위하여 별도의 FPGA 칩을 하나씩 사용하였다.

COBRA 시스템은 소규모 모듈들의 연결이 용이하도록 개발된 시스템으로 각 모듈은 하나의 모서리 면에 90 핀의 케넥터와 연결되어 있다. 모듈 내에서는 Xilinx 4025 4개를 사용하였는데 2개의 이웃한 FPGA와 75개의 선이 연결되어 있다. 그러므로, COBRA 시스템은 보드의 확장이 용이한 장점이 있지만, 하나의 보드의 용량이 제한적이라는 단점이 있다. 시스템에서 메모리의 구현을 위하여 RAM 모듈을 별도로 연결하여 사용하였다. COBRA 시스템에서의 소프트웨어는 시스템의 분할 및 하드웨어 합성 소프트웨어와 하드웨어 디버깅을 위한 소프트웨어 등으로 구성되어 있어서, 다

른 애뮬레이션 시스템과 유사하다.

AnyBoard는 FPGA에 기초한 reconfigurable 시스템으로 외부 RAM, 마이크로프로세서, DSP 등과의 결합이 가능하도록 구현되었다. AnyBoard의 하드웨어 구조는 FPGA와 RAM으로 구성되어 있고, PC와 시스템 인터페이스가 가능하도록 제작되었다. 시스템에서 하나의 FPGA는 download 및 컨트롤을 위하여 별도로 사용하였다. 설계 시스템에서 모듈의 할당을 반복적으로 수행한 것이 특징이다. 모듈의 할당을 위하여 모듈을 이동하고 모듈과 핀을 할당한 후, 모듈의 이동을 받아들일 것인지를 결정하는 과정을 반복한다.

PRISM은 C언어로 기술된 프로그램을 입력으로 받아 이를 쉽게 수행할 수 있도록 하드웨어 구조를 구현하는 시스템이다. 기존의 PRISM I은 10MHz에서 동작하는 M68010 DSP 칩과 4개의 Xilinx 3090을 결합하여 만들었지만, PRISM II는 33MHz에서 동작하는 Am29050 DSP 칩과 3개의 Xilinx 4010 칩을 사용하여 성능을 개선하였다. PRISM II의 특징은 C 언어로 기술된 입력에서 for 문, while 문, switch 문, break 문, continue 문 등을 처리할 수 있도록 개발하였다. 그러나, 아직 double, float 등은 처리되지 않는다.

Virtual Computer는 계산량이 많은 부분을 하드웨어를 이용하여 수행 가능하도록 개발한 시스템으로 논리 구현을 위하여 52 개의 Xilinx 4010 칩이 사용되었다. 그리고, 칩간의 연결을 위하여 24 개의 I-Cube FPIC 칩 IQ160이 사용되었다. I-Cube FPIC 칩을 대각선으로 배치하여 FPGA 칩간의 연결이 용이하도록 하였다. Virtual Computer는 3 개의 64-bit I/O 포트를 가지는데 이중 하나는 configuration을 위한 버스로 사용된다. 메모리의 구현을 위하여 8Mbyte의 SRAM을 사용하였고, 8K 16-bit dual port RAM을 사용하였다. 구현한 보드는 520,000 게이트의 용량을 가진다.

Splash 2는 Xilinx 4010 칩을 이용하여 제작한 애뮬레이터이다. 특징은 16 장 까지의 보드를 연결 가능하도록 제작하였고, SUN SBus를 통하여 DMA로 호스트 워크스테이션과 연결된다. 하나의 보드는 17 개의 Xilinx 4010 칩으로 이루어졌으며,

이들 칩간의 연결은 crossbar 구조에 의하여 이루어진다. 호스트 시스템과의 연결을 위하여 별도의 인터페이스 보드를 구성하였으며, 4 개의 DMA 채널을 통하여 연결된다.

III. 새로운 구조의 애뮬레이터의 개발

본 장에서는 본 저자들이 개발한 애뮬레이터를 소개한다. 먼저, 여러 개의 FPGA 칩간의 연결 구조를 비교하고, 애뮬레이션 보드간의 연결 구조에 대하여 기술한다. 그리고, 애뮬레이터 컴파일러의 주요 부분인 FPGA를 위한 분할에 대하여 기술하고, 칩간의 배선을 동시에 고려한 분할 방법에 대하여 기술한다. 자세한 사항은 참고문헌^[5, 6, 7]를 참고하기 바란다.

1. FPGA 칩간의 연결구조

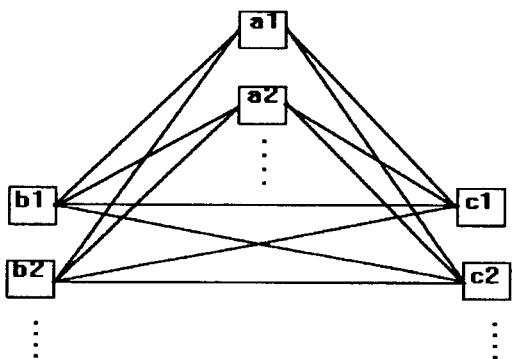
본 절에서는 애뮬레이터의 구조에 관하여 설명한다. 애뮬레이터의 구조는 칩의 효율적인 사용, 배선의 용이성 및 전체 지연 시간 등을 고려하여 설계하여야 한다. 즉, 칩간의 배선이 hardware 적으로 설계되므로, 설계 후 분할에 많은 영향을 미치게 된다.

RPM에서는 상, 하, 좌, 우 및 대각선 방향으로 바로 인접한 칩과 연결되어 있으며, 또한 하나 건너서 각 방향으로 인접한 칩들과도 연결되어 있다. 그리고 가장자리에 배치된 칩은 반대편의 가장자리 칩과 연결되어 전체가 구를 형성하고 있다. 그러나, 서로 멀리 위치한 칩간의 연결에서는 여러 개의 칩을 통하여야 하므로 지연 시간이 증가하게 된다.

Partial crossbar 방식은 현재 가장 많이 사용되고 있는 애뮬레이터 구조이다. 이 방법의 특징은 로직을 구현하기 위한 FPGA 칩과 FPGA 칩의 연결을 위한 crossbar 칩을 분리하여 사용하는 것이다. 모든 로직 칩은 모든 crossbar 칩과 배선을 위한 수 개의 선이 연결되어 있으므로 이를 이용하여 배선을 수행한다. 두 FPGA 칩간의 연결은

crossbar 칩을 통해서 수행한다. 모든 배선은 하나의 crossbar 칩을 통하여 이루어지므로 칩간의 배선 길이를 1로 하면 모든 연결은 2의 배선 길이를 가진다.

본 저자들이 설계하여 검토한 몇 가지의 에뮬레이터의 구조는 partial crossbar 방법의 에뮬레이터 구조와는 다른 특징을 가진다. 먼저 칩간의 연결을 위한 crossbar 칩을 사용하지 않고 FPGA 칩만을 사용하여 논리의 구현 및 칩간의 배선을 수행하였다. 기본 구조는 K분할(K-partite) 그래프이다. 첫 번째의 구조는 2분할(bi-partite) 구조로 전체 칩을 두 그룹으로 나눈 후, 같은 그룹에 속하지 않은 모든 칩간에 배선을 위한 연결선을 만든 회로망이다. 칩간의 배선을 더욱 용이하게 수행하기 위하여 전체 칩을 3그룹 또는 4그룹으로 나누고, 이를 각각 3분할(tri-partite) 그래프와 4분할(quadri-partite) 그래프를 형성하도록 연결한 구조를 생각할 수 있다. <그림 2>는 전체 구조의 일부만을 나타낸 3분할 FPGA 망의 그림이다. 일반적으로 $K \geq 2$ 인 정수 K 에 대하여 K분할 구조가 가능하다.



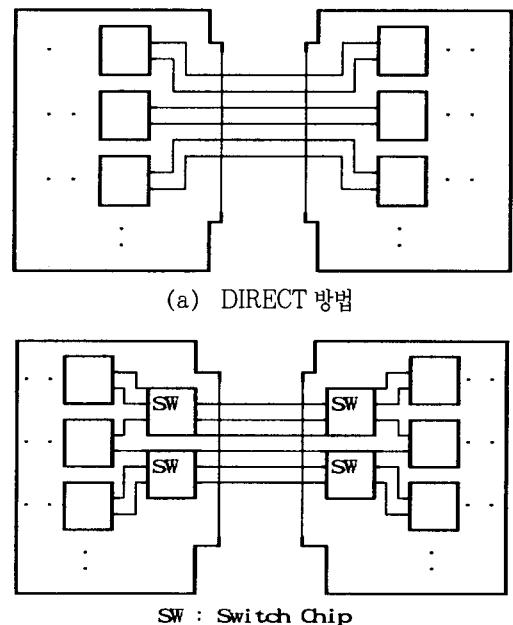
<그림 2> Tri-partite 그래프

MCNC Partition 벤처마크의 예제를 사용하여 실험적으로 여러 구조를 비교하였다. 실험은 여러 가지 network 구조들에 대하여, 가능하면 작은 크기의 칩을 사용하여 회로를 구현하는 것을 목적으로 하였다. 실험 결과 4분할 구조와 3분할 구조가 가장 작은 칩을 사용하여 모든 회로를 구현할 수 있었고, RPM 및 partial crossbar 구조는 더 큰 칩

을 사용하여야 구현 가능하였다. 전체적으로 3분할 구조가 가장 우수한 결과를 보여 주어 가장 적은 비용으로 회로를 구현할 수 있는 것으로 나타났다.

2. 에뮬레이터 보드간의 연결구조

삼분할 구조를 가지는 보드를 서로 연결할 수 있도록 보드간의 연결 구조를 비교하였다 [9]. 보드를 서로 연결하는 구조는 여러 가지가 있을 수 있지만, 본 연구에서는 쉽게 사용 가능한 두 가지 구조를 비교하였다. 하나는 DIRECT 방법으로 다음 <그림 3> (a)의 구조를 가진다. DIRECT 방법에서는 각 칩에서 다른 보드와의 연결을 위한 선이 직접 연결되어 있다. 그러므로 다른 보드와 연결하면 보드 내의 같은 위치의 핀들끼리 직접 연결된다. 다른 하나는 SWITCH 방법으로 <그림 3> (b)의 구조를 가진다. SWITCH 방법에서 다른 보드와의 연결은 스위치 칩을 통하여 연결하도록 되어 있다. 그러므로 SWITCH 방법에서 보드간의 연결시 보드의 스위치 칩간을 연결하는 구조가 필요하다.



<그림 3> DIRECT 방법 및 SWITCH 방법

DIRECT 방법과 SWITCH 방법을 구현하여, MCNC Partition 벤취마크 예제를 사용하여 실험적으로 비교하였다. 실험은 크기가 다른 Xilinx 칩들을 사용하여 보드를 구현하였을 경우, 각각 몇 개의 보드를 사용하여 벤취마크 회로를 구현할 수 있는지를 알아보았다. 큰 칩을 사용하여 보드 수가 많지 않은 경우는 비슷한 결과를 얻었지만, 펈의 수가 작은 경우에는 DIRECT 방법을 사용하여 구현하지 못하는 문제가 발생하였다. 결과적으로 보면, DIRECT 방법이 지연시간 면에서 우수하고, 보드의 확장도 용이하다. 그리고, connection pin이 충분하고 보드 수를 4~5개 이내로 사용하는 경우는 SWITCH 방법에 비하여 구현 가능한 예제의 크기에 별로 차이가 없으므로 DIRECT 방법을 사용하는 것이 우수한 것으로 나타났다.

3. 여러 개의 FPGA 칩을 위한 분할

Emulator compiler의 주요 부분인 FPGA를 위한 분할 방법을 소개한다. 개발한 방법에서의 컴파일러는 custom board를 위한 성능을 고려한 분할 시스템과 보드를 위한 배선을 고려한 분할 시스템으로 구성된다. Custom board를 위한 분할 시스템은 분할 후에 칩간의 배선을 수행하므로 칩간의 연결선 수가 최소가 되도록 분할을 수행하는 것이다. 에뮬레이터를 위한 분할은 칩간의 연결이 고정되어 있으므로 분할시 칩간의 배선을 고려하여 분할하여야 한다.

본 분할 알고리듬은 회로를 구현하는데 필요한 칩들의 전체 비용을 최소화하며 지연 시간이 최소화되도록 개발하였다. 이때, 웨이트를 이용하여 비용과 지연 시간 중에서 특히 고려하여야 할 사항을 우선적으로 고려하도록 하였다. 알고리듬의 구성은 전체적인 최적화를 위한 클러스터를 이용한 계층적 초기 분할과 만족되지 않은 제약 조건을 만족시켜 주기 위한 반복적인 분할의 개선 단계로 나누어진다. 클러스터를 이용한 분할은 전체적으로 우수한 분할을 얻기 위하여 [7]방법을 개선하여 사용하였고, 분할의 개선 단계에서는 만족되지 않은 제약 조건을 만족시켜 주기 위하여 가중치(weight)를 점진적으로 조정하는 방법을 사용하였

다.

본 방법에서 제약 조건을 만족시키며 효율적인 분할을 수행하기 위하여 사용한 비용함수는 다음 식과 같다.

$$\text{Cost} = \text{Net_cost} + W1 \times \text{CLB_cost} + W2 \times \text{Pin_cost} \\ + W3 \times \text{Delay_cost}$$

식에서 Net_cost는 분할된 그룹을 연결하는 네트에 대한 비용이며, CLB_cost는 각 그룹에서 칩의 CLB 수보다 더 많이 할당된 CLB 수의 합이다. 그리고, Pin_cost는 칩의 펈의 수보다 더 많이 할당된 펈 수의 합이고, Delay_cost는 회로에서 가장 지연 시간이 큰 패스의 지연 시간이다. 식에서 W1, W2, W3는 각 제약 조건에 대한 가중치(weight)이다. 분할 후 제약 조건이 모두 만족되면, 둘째 항과 셋째 항의 값은 0이 되므로, 비용은 네트와 지연 시간에 의한 비용으로만 구성된다.

지연 시간을 고려하여 분할을 수행하기 위하여 클러스터의 형성시 critical 패스에 웨이트를 주었고 분할시 비용함수에 최대 지연 시간을 비용으로 고려하여 최소가 되도록 수행하였다. 클러스터의 생성시 웨이트를 주는 이유는 critical path 상의 셀들을 하나의 클러스터로 만들어 같은 그룹 내에 분할되게 하여 칩간의 연결에 의한 지연 시간을 최소화하기 위한 것이다. 입력 회로에서 지연 시간이 큰 패스의 순서로 큰 웨이트를 주었다. 하나의 에지(edge)는 여러 개의 패스에 속하게 되므로, 그 에지를 지나는 패스중 가장 지연 시간이 큰 패스의 웨이트를 주었다.

분할 단계에서 지연 시간을 고려하기 위하여 비용함수에 최대 지연 시간을 비용으로 고려하였다. 분할을 하면 칩간을 연결하는 패스의 지연 시간이 증가하게 된다. 그러므로, 초기 분할을 수행하고 최대 지연 시간을 계산하여 비용을 구하게 된다. 그리고, 분할의 개선 단계에서 셀을 이동하기 위한 이득을 구할 경우와 셀을 이동한 후 이득을 수정하는 경우, 이동하는 셀부터 시작하여 fanout tree에 속한 셀들의 최대 도달 시간을 update하여 패스들의 지연 시간을 수정하여 준다. 이때, fanout tree에 속한 셀중 최대 도달 시간이 변한 경우에만 이하의 셀을 update 하는 효율적인 event-driven

방식을 사용하였다.

계층적 분할은 클러스터를 이용하여 주어진 칩의 수에 맞도록 회로를 초기 분할하는 과정으로 전체적인 최적화에 중점을 두었다. 그러므로, 전체적으로 칩간을 연결하는 네트의 수를 최소화하고, 분할된 회로의 크기의 균형을 맞추는데 중점을 두었다. 클러스터 단계의 분할은 클러스터의 형성, 여러 번의 클러스터의 분할, 그리고 클러스터의 해체 등으로 구성된다. 먼저 클러스터를 형성하여 전체 회로의 규모를 줄인 후, 이를 임의의 초기 분할에서 시작하여 여러 번 분할하여 가장 좋은 결과를 저장한다. 그리고, 클러스터 단계에서의 저장된 결과를 이용하여 셀 단계의 분할을 수행한다. 이 방법은 기존의 셀 단계의 분할만을 수행하는 방법들에 비하여 초기 해에 의한 영향을 적게 받는다는 장점이 있다.

분할의 개선은 계층적 분할을 수행한 후, 만족되지 않은 제약 조건을 만족시키며 지역 시간을 더욱 줄이기 위하여 수행한다. 계층적 분할은 전체적인 최적화를 위하여 그룹의 크기의 균형 및 그룹 간을 연결하는 네트의 수, 즉 그룹들의 핀의 수를 줄이는데 중점을 두었다. 그러므로 전체적으로 좋은 분할이 되었더라도 일부 그룹에 CLB 및 핀의 수가 주어진 수보다 많이 할당되어질 수 있다. 분할의 개선 단계에서는 만족되지 않은 제약 조건을 만족시키며 비용의 최소화를 위한 셀의 이동을 수행한다. 즉, 핀과 CLB 수에 의한 제약 조건을 모두 만족시키며 칩간의 연결도와 지역 시간에 의한 비용을 최소화하는 것이 목적이다. 분할의 개선에서는 모든 그룹으로부터 셀이 이동이 수행되면 한 번의 iteration을 마치게 된다. 한번의 iteration을 수행한 후, 핀 및 CLB 수의 제약 조건이 만족되지 않았으면, 그 제약 조건에 대한 가중치를 크게 함으로써, 다음의 이동에서 우선적으로 그 조건이 만족되도록 하였다.

벤чу마크 예제 중에서 Xilinx 3000 series의 구조에 맞게 기술매핑 된 회로에 대하여, 「8」의 방법과 결과를 비교하였다. 전반적으로 본 분할 알고리듬은 여러 개의 FPGA를 위한 분할을 성공적으로 수행하는 우수한 결과를 보여 주었다. 예제를

구현하는데 필요한 전체 비용이 「8」의 방법은 132이지만, 새로운 방법은 114.16의 비용으로 구현 가능하여 14.7% 우수한 결과를 보여 주었다.

4. 에뮬레이션 보드를 위한 배선을 고려한 분할

여러 개의 에뮬레이션 보드를 위한 분할은 custom board를 위한 분할과 유사하다. 다만, 분할에서 배선을 고려하여 분할 후 100% 칩간의 배선이 완료되도록 하였다. 두 칩간의 배선은 두 칩을 연결하는 핀에 신호선을 할당하는 것으로 수행하였다. 여러 개의 보드를 위한 분할에서는 먼저 보드간의 분할을 수행하고, 각각의 보드에 대하여 칩간의 배선을 고려하지 않은 초기 분할을 수행한 후에 배선을 고려한 분할의 개선 과정을 수행하였다. 보드간의 분할이 완료되면 보드간에 연결하여야 하는 네트를 알 수 있으므로, 이들 네트를 보드간의 연결선에 할당한다. 할당은 보드 내부의 분할 및 배선이 수행되지 않았으므로 임의로 할당하였다. 그리고, 각 보드의 초기 분할 및 배선을 고려한 분할의 개선 과정을 수행하였다. 이렇게 수행하면 계층적으로 수행할 수 있는 장점이 있다. 즉, 먼저 보드간의 연결선을 최소화한 후에 보드 내부의 분할을 수행할 수 있으므로 각 보드의 분할은 독립적으로 수행할 수 있다. 여러 개의 보드에 있는 칩을 전체적으로 한번에 분할 및 배선을 수행하는 것은 수행 시간 면에서 효율적이지 못하며, 분할도 초기 분할의 영향을 많이 받으며 효율적이지 못하다.

보드간의 분할 및 칩간의 배선을 고려하지 않은 초기 분할은 custom 보드를 위한 분할 알고리듬을 이용하였다. 분할의 개선 알고리듬은 그룹간에 셀을 이동시켜서 배선에 대한 제약 조건을 만족시킨다. 즉 초기 배선을 수행하면 칩간의 연결이 많은 부분에 필요한 신호선의 수가 배선 용량을 초과하게 된다. 이를 셀 이동시의 이득에 고려하여 셀을 이동하므로 용량을 초과하는 부분의 배선 밀도를 감소시키는 쪽으로 분할을 개선하게 된다.

배선은 패스의 패턴(pattern)을 이용하여 효율적으로 수행한다. 칩간의 연결 가능한 패스의 패턴들을 배선을 수행하기 전에 미리 구하고, 경로를

탐색할 때 이 패턴들을 이용하여 수행 시간을 줄이도록 하였다. 이 때, 각 패스의 패턴의 길이를 제한하면 여러 개의 칩을 거쳐 연결되어 지연 시간이 증가하는 것을 방지할 수 있다. 본 에뮬레이터에서는 지연 시간을 줄이기 위하여, 패턴을 구할 때, 패스의 길이가 2이상인 패스는 탐색에서 제외하였다.

배선을 고려한 분할의 개선은 다음과 같다. 먼저 각 셀의 이득을 구하고, 이득이 큰 순서대로 다른 그룹으로 이동한다. 셀의 이득은 칩의 크기의 제약 조건과 편 수의 제약 조건, 그리고 배선에 의한 제약 조건을 비용에 포함시켰을 때, 그 셀을 이동하였을 때 줄어드는 비용이다. 모든 그룹에 대하여 셀의 이동을 수행한 후 일부 제약 조건이 만족되지 않았으면, 그 제약 조건에 대한 가중치 (weight)를 크게 함으로써 다음 이동에서 우선적으로 개선시키도록 한다. 이와 같이 만족되지 않은 제약 조건의 가중치를 증가하여 셀의 이동을 수행하면 다른 제약 조건이 만족되지 않게 되거나 네트에 의한 비용이 증가할 수도 있다. 그러나, iteration을 반복할수록 점차 제약 조건들을 만족시키게 된다. 비용의 감소 없이 3번 이상 iteration을 반복하면 수행을 마치게 된다. 비용의 감소 없이 3번 이상의 iteration이 반복된 경우에는 더 이상 수행하여도 수행 시간만 증가하고 더 이상 개선될 가능성이 적기 때문이다.

IV. 에뮬레이션 실험 결과

본 장에서는 저자들이 개발한 에뮬레이터를 이용한 몇 가지 에뮬레이션 예를 기술한다.

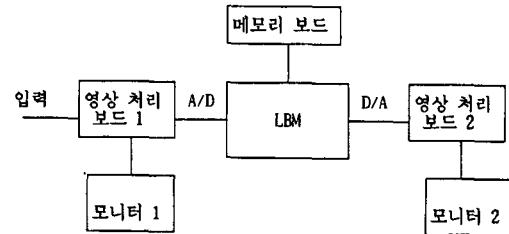
1. CR8010 에뮬레이션

개발한 에뮬레이터를 이용하여 8-bit micro-processor인 CR8010을 5MHz 이상의 동작 속도로 성공적으로 에뮬레이션을 수행하였다. 하나님의 에뮬레이션 보드에 CR8010의 회로를 구현하였고, 동작을 확인하기 위하여 CR8010 target 보드를 사

용하였다. CR8010 target 보드는 CR8010 micro-processor을 이용한 응용 보드로 CR8010 micro-processor, keyboard, display, interface controller 등으로 구성되어 있다. 실험을 위하여 CR8010 micro-processor를 제거하고 대신에 에뮬레이션 보드와 연결하여 실험하였다. 실험 결과 5 MHz에서 정상적으로 동작함을 확인하였다. Target board의 동작 속도가 5 MHz 이내로 제한되기 때문에 5 MHz 까지만 실험 가능하였다.

2. 자동 영상 안정화 시스템(AIS) 에뮬레이션

자동 영상 안정화 시스템의 에뮬레이션을 수행하였다. 자동 영상 안정화 시스템은 소형 비디오 카메라의 손떨림으로부터 움직임 값을 찾아내어 손 떨림을 보상해 주는 시스템이다. 영상 안정화 시스템을 에뮬레이션 하기 위하여 <그림 4>와 같이 구현하였다. 비디오 카메라로부터 입력 신호가 들어오면 첫 번째 영상 처리 보드를 통해 원래 화면을 모니터 1에 디스플레이 하는 동시에 A/D 변환된 신호가 로직 에뮬레이터로 입력된다. 로직 에뮬레이터는 별도로 설계된 메모리 보드를 이용하여 움직임 값을 찾아 움직임이 보상된 데이터와 동기 신호를 두 번째 영상 처리 보드로 내보낸다. 두 번째 영상 처리 보드는 입력된 신호를 D/A 변환하여 모니터 2에 디스플레이 함으로써 원래의 영상과 움직임이 보상된 영상을 비교할 수 있게 하였다.



<그림 4> 영상 안정화 시스템 블록도

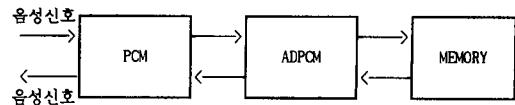
움직임 보상은 다음과 같이 수행한다. 영상이 입력되면 필터링을 함으로써 입력 영상에 존재하는 노이즈를 완화시킨다. 필터링이 끝난 데이터는 마스킹 과정을 거치는데 마스킹 과정은 움직임 검출

에 사용되는 특징점을 잡기 위한 에지 검출 과정이다. 마스킹 과정을 거친 영상은 16개의 블록으로 나누어지고 각각의 블록에 대하여 특징점을 잡고 각 특징점을 기준으로 탐색 영역을 정한다. 특징점을 정한 후에는 이전 프레임과 현재 프레임간의 상관 관계를 구한다. 상관 관계는 이전 프레임의 특징점 값과 현재 프레임의 탐색 영역 내에 있는 데이터 값들과의 차의 절대값이 된다. 각 블록에 대해 상관 관계 계산이 완료된 후에 16개 블록의 결과를 모두 더한다. 이렇게 되면 이전 프레임으로부터 움직인 거리만큼의 위치에 있는 데이터 값은 탐색 영역 내의 데이터 중 가장 작은 값이 된다. 따라서 가장 작은 값이 기준 되는 특징점으로부터 떨어진 위치가 이전 프레임으로부터 현재 프레임이 움직인 위치가 된다. 움직임을 보상하기 위해서 프레임 메모리에 저장되어 있는 이전 프레임의 데이터를 읽을 때 움직인 거리만큼 어드레스를 보상하여 읽는다. 자동 영상 안정화 시스템의 크기를 2-입력 게이트로 환산할 때 전체로는 약 22.4K 게이트이다.

영상 안정화 시스템은 실시간으로 입력되는 영상을 처리하여야 하므로, 사용한 영상 처리 보드의 샘플링(sampling) 속도로 에뮬레이션을 수행하여야 한다. 사용하는 영상 처리 보드의 샘플링 속도는 12.7 MHz이다. 에뮬레이션 과정은 먼저 회로를 몇 개의 부분 회로로 분할하고, 각각의 부분 회로에 대하여 에뮬레이션을 수행하여 정확히 동작하는지를 확인한 후 전체 시스템에 대한 에뮬레이션을 수행하였다. 에뮬레이션 결과, 흔들림이 있는 영상이 12.7 MHz의 실시간으로 보상됨을 확인하였다. 현재 상용되는 로직 에뮬레이터의 에뮬레이션 속도가 4-10 MHz임을 감안할 때, 12.7 MHz의 에뮬레이션 속도는 상당히 빠른 것이다.

3. ADPCM 에뮬레이션

2장의 에뮬레이션 보드를 연결하여 ADPCM 회로의 에뮬레이션을 수행하였다. <그림 5>는 ADPCM을 이용한 음성신호의 인코딩과 디코딩 과정을 나타낸다. PCM 부분은 아날로그 신호인 음성신호를 입력받아서 8-bit/sample의 디지털 신



<그림 5> 음성신호의 인코딩/디코딩

호로 만들고 ADPCM 부분은 PCM 부분으로부터 8-bit/sample의 데이터를 입력받아 4-bit/sample로 압축한다. ADPCM 부분에서 압축된 데이터는 메모리에 저장된다. 디코딩은 위 과정의 역순으로 한다. 에뮬레이션을 수행한 부분은 그림의 ADPCM 부분으로 2 MHz의 속도로 에뮬레이션을 수행하였다. ADPCM은 2-입력 게이트로 환산할 때 약 21K 게이트의 크기를 갖는다.

V. 하드웨어 Accelerator로 Codesign에의 활용

현재의 설계 검증의 많은 부분이 시뮬레이션에 의하여 이루어지고 있는데, 설계 복잡도의 증가와 다양한 기능의 추가로 인하여 시뮬레이션 시간이 크게 증가하게 되는 문제가 발생하게 되었다. 이러한 문제를 해결하고자 빠른 속도의 시뮬레이션을 위한 알고리듬의 개발 및 다수의 프로세서를 이용한 시뮬레이션 알고리듬의 연구 등이 진행되었다. 그러나, 설계의 복잡도가 크게 증가함에 따라, 보다 상위 단계에서 시뮬레이션을 하거나, 또는 특수 기능의 하드웨어 accelerator 등을 개발하여 사용하게 되었다. 하드웨어 accelerator는 일반적인 컴퓨팅 환경에서 수행하기 어려운 규모가 크고 다기능의 시뮬레이션을 빠르고 쉽게 수행할 수 있도록 고안된 하드웨어나 특수 목적 컴퓨터이다. 일반 환경의 소프트웨어 시뮬레이터는 보통 초당 1,000 개의 게이트 연산을 수행하지만, 하드웨어를 이용하는 경우 백만 개의 게이트 연산을 수행하는 것이 보통이다. 그러므로 복잡한 시스템의 시뮬레이션을 수행할 때 가격, 기능, 수행 시간에 따라 장단점을 비교하여 소프트웨어만을 이용하여 수행할 것인지, 다수의 프로세서를 사용할 것인지, 또는

하드웨어 accelerator로 이용할 것인지를 먼저 결정하여야 한다.

Rapid prototyping 기술은 꼭 하드웨어 시스템만을 구현하는 것에 제한되지 않고, 하드웨어 accelerator 등으로 응용할 수 있다. 실제로 요즈음은 virtual prototyping 개념으로 하드웨어-소프트웨어 통합 설계 환경을 이용한 하드웨어 accelerator 쪽으로 발전하는 추세이다. 이러한 하드웨어 accelerator의 구현에서 가장 문제가 되는 것이 시뮬레이션의 속도이다. 현재에 사용되고 있는 ZyCAD나 IKOS의 하드웨어 accelerator는 속도가 시뮬레이션 속도의 10~100배로 알려져 있다. 또한, 현재에 cycle-based 시뮬레이터로 기존 시뮬레이터 속도의 10~100배 정도 되는 것이 개발되어 많은 기업체 등에서 널리 사용되고 있다. 그러나, 이러한 소프트웨어에 기초한 시뮬레이터의 속도에는 개선에 한계가 있으므로, 하드웨어-소프트웨어 통합 설계 환경을 이용한 virtual prototyping 기술로 발전해나가는 추세이며, 앞으로 수행시간의 향상이 좀더 이루어지면 시장이 크게 확장될 전망이다.

앞에서 기술한 본 저자들이 개발한 에뮬레이터도 하드웨어 accelerator로 확장 가능하며, 앞으로 이에 대한 연구를 수행하고자 한다. 본 에뮬레이터는 동작 속도가 기존의 에뮬레이터에 비하여 빠른 장점이 있으므로, 이러한 장점을 이용하여 고속의 시스템을 개발하려고 한다. 현재의 cycle-based 시뮬레이터나 하드웨어 accelerator는 기존 시뮬레이터의 10~100배이지만, 본 저자들이 개발한 새로운 에뮬레이터를 기존의 mixed-level 시뮬레이터나 gate-level 시뮬레이터 등과 연결하면, 에뮬레이터와 유사한 동작 속도를 얻을 수 있을 것으로 기대된다.

통합 설계에 하드웨어 accelerator로 사용하기 위하여 다음과 같은 구조가 필요하다. 먼저 VHDL 또는 gate-level 시뮬레이터가 PC 또는 workstation 등의 CPU에서 수행되고, 이중에서 많은 시간이 소요되는 부분을 인터페이스를 이용하여 에뮬레이터에서 하드웨어로 구현한다. 회로를 본 에뮬레이터를 이용하여 하드웨어로 구현하

면 병렬처리 등이 용이하므로, 소프트웨어로 CPU에서 순차적으로 수행되는 것에 비하여 상당히 빠른 속도를 얻을 수 있다. 하드웨어를 이용한 accelerator를 사용하기 위하여서는 시뮬레이션 중에서 어떤 부분을 하드웨어로 구현하여야 가장 빠른 효과를 낼 수 있는지를 결정하기 위한 소프트웨어 알고리듬의 개발이 이루어져야 한다. 그러므로, 하드웨어 accelerator를 구현하기 위하여 수행 시간 절감을 위한 하드웨어 시스템 뿐만 아니라, 이를 효율적으로 이용하기 위한 소프트웨어 컴파일러의 개발이 필요하다.

에뮬레이터만을 이용하면 설계의 검증시 에뮬레이터와 서로 연결되어 수행되는 하드웨어적인 target 보드가 필요하다. 그러나, 에뮬레이터를 하드웨어 accelerator로 구현하면 target 보드 대신에 소프트웨어 프로그램에 의하여 동작시키므로, 더 간단히 검증할 수 있는 장점이 있다. 그러나, accelerator로의 구현을 위하여 시스템 버스를 이용하여 인터페이스를 수행하기 위한 버스 컨트롤러, 인터페이스 회로 등이 필요하다. 일반적으로 하드웨어 accelerator의 동작 속도에 가장 큰 영향을 미치는 것이 하드웨어와 소프트웨어 시뮬레이터와의 인터페이스이다. 인터페이스는 보통 주어진 버스를 사용하여 수행하므로, 데이터의 전송을 위하여 정해진 protocol을 이용하여 인코딩한 후, 이를 직렬로 전송하여야 한다. 데이터를 받는 쪽에서는 이를 다시 디코딩하여 원하는 데이터 값을 얻을 수 있다. 현재의 많은 하드웨어-소프트웨어 통합 설계 시스템에서도 인터페이스에 의하여 동작 속도에 많은 제약을 받는 것으로 나타났다. 본 에뮬레이터를 이용한 하드웨어 accelerator에서도 하드웨어 소프트웨어 통합 설계의 개념을 도입하여 하드웨어-소프트웨어의 분할, 인터페이스 등을 최적화하면 우수한 성능을 가질 수 있을 것으로 기대된다.

VI. 결 론

로직 에뮬레이터에 대하여 전반적으로 기술하였

다. 먼저 로직 에뮬레이터의 정의, 구조, 특징 및 장단점 등을 기술하였고, 현재까지 개발되어 사용되고 있는 에뮬레이터에 대하여 소개하였다. 그리고, 본 저자들이 개발한 FPGA 칩간의 연결 구조를 설명하였고, 에뮬레이터 보드간의 연결 구조에 대하여 기술하였다. 새로 개발한 FPGA 칩 간의 연결 구조는 3분할 구조로 큰 회로가 고속으로 동작되도록 구현하는데 효과적으로 나타났다. 보드 간의 연결 구조는 DIRECT 방법을 사용하여 속도가 빠르면서 보드의 확장이 용이한 방법을 사용하였다. 에뮬레이터 컴파일러의 주요 부분인 FPGA를 위한 분할에 대하여 기술하였고, 칩간의 배선을 동시에 고려한 분할 방법에 대하여 기술하였다. FPGA를 위한 분할에서는 큰 회로를 효과적으로 처리할 수 있도록 클러스터를 이용한 계층적 분할 방법을 사용하였으며, 여러 제약조건을 만족시키기 위하여 가중치를 동적으로 조절하는 방법을 사용하였다. 칩간의 배선은 패턴을 이용하여 분할과 동시에 수행하였다. 그리고, 여러 가지 회로에 대한 에뮬레이션의 예를 소개하였다. 마지막으로, 에뮬레이터를 하드웨어 accelerator로 확장하여 하드웨어 소프트웨어 통합설계 시스템으로 이용하는 방법에 대하여 소개하였다.

현재까지 여러 에뮬레이터가 개발되어 사용되고 있지만, 도입단계라고 판단된다. 우선 현재의 에뮬레이터가 너무 고가이다. 1994년을 기준으로 게이트당 1.25 달러의 비용이 드는 것으로 조사되어 있다. 그러므로 에뮬레이터의 비용을 낮추는 것이 중요한 과제이다. 그리고, 에뮬레이터는 사용이 용이하여야 한다. 현재 개발된 에뮬레이터를 잘 사용하기 위하여 많은 시간을 투자하여 설계와 에뮬레이터를 모두 잘 이해해야 하는 단점이 있다. 에뮬레이터를 위한 컴파일러의 성능이 개선되어야 하며, 계층 구조를 가지는 회로의 에뮬레이션을 용이하게 수행할 수 있어야 하고, 여러 개의 주변 시스템과 연결이 용이하여야 한다. 에뮬레이터를 사용하는 중요한 이유가 디버깅을 용이하게 수행하기 위한 것이므로 디버깅을 용이하게 수행할 수 있는 환경을 개발하는 것도 중요한 과제이다. 그리고 에뮬레이터를 하드웨어 accelerator 등으로 확

장하여 하드웨어 소프트웨어 통합설계 시스템으로 이용하기 위한 인터페이스를 빠르고 용이하게 수행할 수 있는 방법의 개발도 필요하다. 이러한 문제들이 보완된다면 에뮬레이터를 이용한 설계 방식이 더욱 유용하게 사용되어질 수 있고, 사용도 크게 증가할 것이다.

참 고 문 현

- [1] PIE Design Systems, Int., Using MARSII for Logic Emulation," OHP 자료, 1993.
- [2] Xilinx, Inc., The Programmable Gate Array Data Book, Xilinx, San Jose, 1995.
- [3] Quickturn, Inc., Introducing the System Realizer Modular Emulation System, ICCAD Tutoial, 1995.
- [4] J. Varghese, M. Butts and J. Batcheller, "An Efficient Logic Emulation System," IEEE Trans. on VLSI Systems, VOL. 1, NO. 2, pp. 171-174, June 1993.
- [5] C. Kim and H. Shin, "A Performance-Driven Logic Emulation System: FPGA Network Design and Performance-Driven Partitioning", IEEE Transactions on CAD/ ICAS, pp. 560-568, May 1996.
- [6] S. Yong, C. Kim and H. Shin, "Design of a Multi-Board Logic Emulation System", International conference on VLSI and CAD, pp.405-408, Oct, 1995.
- [7] H. Shin and C. Kim, "A Simple Yet Effective Technique for Partitioning," IEEE Trans. on VLSI Systems, VOL. 1, NO. 3, pp. 380-386, Sep. 1993.
- [8] R. Kuznar, F. Brglez and K. Kozminski, "Cost Minimization of Partitioning into Multiple Device," Proc. 30th Design Automation Conference, pp. 315-320, 1993.

- [9] C. Kim, H. Shin and Y. Yu, "Performance-Driven Circuit Partitioning for Prototyping

by Using Multiple FPGA Chips," Proc. ASP-DAC'95, pp. 113-118, 1995.

저자 소개



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)

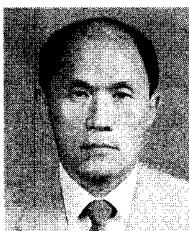
1987年 10月 미국 U.C. Berkeley 전기 및 컴퓨터공학과 졸업(박사)

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

주관심 분야: 직접회로 설계 자동화, 디지털 신호처리 시스템 설계 등임



柳 泳 昱

1946年 6月 2日生

1973年 2月 한양대학교 전자공학과 졸업(학사)

1975年 8月 한국과학기술원 전기 및 전자과 졸업(석사)

1975年 9月~1977年 1月 한국과학기술원 반도체기술 개발센터 연구원

1977年 2月~1985年 1月 한국전자기술연구소 선임연구원, LSI설계실장, 미국 사무
소 소장

1983年 3月~1985年 7月 VLSI Technology, Inc. Design Center 근무

1985年 8月~1989年 1月 한국전자통신연구소 연구위원

1989年 1月~1990年 3月 Valid Logic Systems, Inc. 한국지사장

1990年 4月~현재 (주)서두로직 대표이사, 서두 전자기술연구소 소장

주관심 분야: CAD 기술 개발, 통신, 음성, 영상용 ASIC 개발, Multimedia chip/board
개발