하드웨어-소프트웨어 통합 설계 시스템을 위한 상위 단계에서의 검증 기법

(High-Level Design Verification Techniques for Hardware-Software Codesign Systems)

이 종 석 † 김 충 희 † 신 현 철 †††

(Jongsuk Lee) (Chunghee Kim) (Hyunchul Shin)

요 약 설계되는 시스템의 규모가 커지고 복잡해지므로 이를 빠른 시간 내에 효율적으로 검증하기 위한 상위 단계에서의 검증 기술의 개발이 중요하게 되었다. 본 연구에서는 하드웨어와 소프트웨어가 혼합되어 있는 시스템을 위한 상위 단계에서의 검증기술을 개발하였다. 에뮬레이션 또는 시뮬레이션만을 수행하는 것보다 빠르고 우수하게 기능적으로 검증하기 위해, 하드웨어와 소프트웨어 부분으로 분할한 후 인터페이스 회로를 이용하여 구현 가능하도록 하였다. 그리고, 상위 단계의 회로를 쉽게 하드웨어를 이용하여 검증하기 위한 설계 지침들을 제시하였다. 본 방법을 이용하여 리드-솔로몬 디코더 회로에 대한 검증을 수행한 결과 시뮬레이션만을 수행한 경우에 비하여 modified Euclid 알고리즘 수행 블록은 12,000배 이상의 속도로 검증을 수행할 수 있었으며, 전체 검증 시간도 반 이하로 줄었다.

Abstract As the system complexity increases, it is important to develop high-level verification techniques for fast and efficient design verifications. In this research, fast verification techniques for hardware and software co-design systems have been developed by using logic emulation and algorithm-level simulation. For faster and superior functional verification, we partition the system being designed into hardware and software parts, and implement the divided parts by using interface modules. We also propose several hardware design techniques for efficient hardware emulation. Experimental results, obtained by using a Reed-Solomon decoder system, show that our new verification methodology is more than 12,000 times faster than a commercial simulation tool for the modified Euclid's algorithm block and the overall verification time is reduced by more than 50%.

1. 서 론

집적회로 설계 및 공정기술이 급속히 발달함에 따라 설계 시스템의 규모가 급격히 커지고 복잡해진다. 특히 embedded core, firmware 그리고 부가적인 하드웨어 같은 복합적인 블록들이 하나의 칩으로 구성되는 시스템이 크게 증가함에 따라 이러한 시스템을 빠른 시간 내에 효율적으로 검증하기 위한 기술 개발의 필요성이 크게 증가하고 있다.[1]

일반적으로 회로 검증에 사용되는 시뮬레이션은 소프트웨어에 의하여 순차적으로 수행된다. 그러므로 실제 상황의 일부만을 관찰할 수 있으며, 시뮬레이션을 위하여 먼저 주변 환경의 소프트웨어적인 모델을 만들어야 한다. 그러나 소프트웨어적인 모델만을 이용하여 실제 상황을 표현하기에는 여러 어려움이 있다. 소프트웨어 모델을 완성 후, 소프트웨어 알고리즘에 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. 최근에는 대규모 동기 시스템의 빠른 검증을 위해 cycle-based simulation[2,3]을 사용하기도 한다. 최근 실용적인 cycle-based simulator[4]

· 본 연구는 한국과학재단 특정기초연구(과제번호 1999-2-302-002-3)의 지원을 받았음.

† 비 회 원 : 현대전자 메모리연구소 응용제품팀 연구원
musasy75@ichum.com
†† 비 회 원 : 한양대학교 전자공학과
chkim@warhol.hanyang.ac.kr
††† 종신회원 : 한양대학교 전자공학과 교수
shin@mail.hanyang.ac.kr

논문접수 : 2000년 1월 29일
심사완료 : 2000년 6월 24일

는 다중 클록을 가진 회로를 허용하고 event-based simulation과의 연결도 가능하다. 시뮬레이션에서 회로의 복잡도에 따라 수행시간이 크게 증가하는 단점을 극복하기 위해 하드웨어를 이용한 acceleration 방법[5]도 제안되었으며, 프로세서의 VHDL 모델과 프로세서 컴파일러를 동시에 이용하여 컴파일러도 함께 검증하는 통합-시뮬레이션 기법[6]도 개발되었다. 하드웨어-소프트웨어 통합 설계 툴인 POLIS[7]에서 설계 검증을 위하여 사용하는 Ptolemy [8]는 서로 다른 설계 환경 혹은 수준에서 설계된 회로를 지원하는 Multiparadigm 시뮬레이션 기법을 사용하여 통합 검증을 수행한다.

다양하고 복잡한 검증을 빠르고 정확하게 수행하거나 실제 환경에서의 하드웨어적인 검증을 위하여 에뮬레이션을 이용한다. 에뮬레이션은 FPGA 등의 프로그램 가능한 소자들을 이용하여 회로를 구현한 후 실제 환경과 연결하여 검증하는 방법이다. 에뮬레이션은 실제 하드웨어에서 수행되기 때문에 병렬로 처리된다. 그리고 주변 하드웨어와 연결되어 수행되므로, 시뮬레이션에서처럼 외부 상황에 대한 모델을 별도로 만들 필요가 없다. 에뮬레이션은 실제 하드웨어에 비하여 10배정도 느린 것이 보통이지만, 소프트웨어 시뮬레이션에 비하여 100만 배 정도까지 빠른 것으로 알려져 있으며, 칩이 제작되기 전에 시뮬레이션으로는 찾기 힘든 에러를 찾을 수 있다. 그러나 하드웨어적인 검증방법인 에뮬레이션은 에뮬레이션하기 위한 준비과정에 많은 시간이 소요되며, 일부 설계의 수정을 필요로 하는 단점이 있다. 지금까지 여러 가지 에뮬레이션 방법들이 제안되었다. Quickturn사에서 개발한 System Realizer[9]는 대규모 시스템도 구현 가능하도록 6백만 게이트까지 확장 가능하도록 개발되었으며, Partial crossbar 연결 구조를 사용하여 FPGA 이용률이 최대가 되도록 하였고, 1-4MHz의 속도로 동작한다. IKOS사의 VirtualLogic은 Virtual Wires[10] 기술을 이용하여 안정적이고 빠른 자동 컴파일이 가능하고, 또한 node reconstruction이라는 소프트웨어적인 방법을 이용하여 모든 내부 신호를 관찰할 수 있게 하여 시뮬레이터와 유사한 디버깅 환경을 제공한다. Aptix사의 System-on-Chip Explorer[11]는 FPGA와 일반용도의 IC, 그리고 DSP칩 등을 연결하여 시스템을 구현 가능하도록 개발되었다.

에뮬레이터를 이용하여 하드웨어와 소프트웨어를 동시에 검증하는 방법도 제안되었다[12]. 순차 회로의 fault 시뮬레이션 속도를 높이기 위한 fault 에뮬레이션 시스템도 개발되었는데, 이 시스템은 병렬 fault 시뮬레이터 보다 20배정도 빠르다[13]. 시뮬레이션과 에뮬레이션의 단점을 보완하기 위해 reconfigurable computing system상에서 event-driven simulation을 수행하는 기법도 개발되었다[14].

에뮬레이션과 시뮬레이션에는 각각 장점 및 단점이 있으며 이것은 상호 보완될 수 있다. 즉, 계산량이 많고 복잡하여 수행 시간이 긴 블록은 에뮬레이션을 수행하여 검증 시간을 단축시킬 수 있고, test bench의 제작이 오래 걸리거나, 정확히 모델링하기 힘든 블록은 에뮬레이션을 수행하면 정확하고 빠른 검증을 할 수 있다. 그리고 하드웨어 구현이 어렵거나 대용량의 메모리를 포함하여 FPGA의 사용율을 저하시키는 블록, 비동기 회로를 가지는 블록, 아날로그 회로가 존재하는 블록들은 에뮬레이션을 하기 위해 재설계를 필요로 할 뿐 아니라 이러한 블록으로 인해서 에뮬레이터의 용량을 초과할 수 있으므로 시뮬레이션을 수행하는 것이 유리하다. 또한 에뮬레이션은 정확하게 타이밍 에러를 검증하기 어려우므로 기능적인 검증보다 타이밍이 더욱 중요한 회로에서는 타이밍 시뮬레이션을 수행할 필요도 있다. 그러므로 그림 1과 같이 시뮬레이션과 에뮬레이션을 함께 이용하면 이들 단점을 보완하고 빠르고 정확한 검증을 수행할 수 있다.

일반적으로 RTL 수준 설계를 이용하여 에뮬레이션을 정확하고 빠르게 수행하기 위해서는 많은 재설계가 요구된다. 특히 에뮬레이션을 수행하기 위한 FPGA 매핑 과정에서 많은 에러가 발생할 수 있으며, FPGA 자원을 낭비하여 용량을 초과하거나, 에뮬레이터가 느린 속도로 동작하거나 잘못된 결과를 초래할 수도 있다.

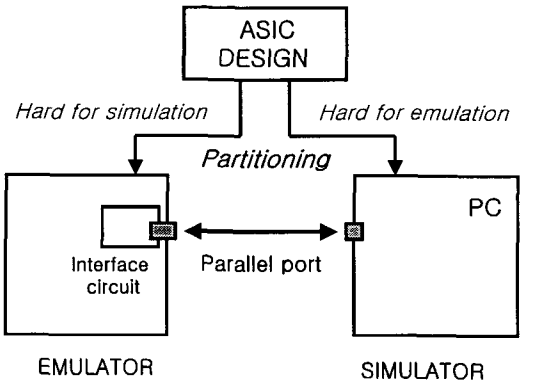


그림 1 에뮬레이션과 시뮬레이션 기법을 이용한 기능적 검증 방법

본 논문에서는 하드웨어와 소프트웨어가 혼합된 회로의 빠른 검증을 위해 에뮬레이션과 시뮬레이션 기법을

동시에 사용하는 검증 방법을 기술하였다. 또한 에뮬레이션을 빠르고 정확하게 수행하기 위한 설계 지침을 제시하였다.

제 2 장에서는 하드웨어-소프트웨어 통합 설계 시스템의 검증을 위한 검증 방법을 제안하고, 에뮬레이션 시스템과 PC를 효율적으로 연결하기 위한 인터페이스 설계 방법을 기술하였다. 3 장에서는 에뮬레이터를 이용하여 하드웨어 부분을 정확하고 빠르게 검증하기 위한 RTL 수준에서의 설계 지침에 대하여 설명하였다. 4 장에서는 Reed-Solomon Codec 회로를 제안된 방법을 이용하여 검증한 실험 결과에 대하여 기술하였고, 5 장에서 결론을 기술하였다.

2. 하드웨어-소프트웨어 통합 검증 시스템의 설계

최근 컴퓨터와 제어 및 통신 기기를 포함하는 대부분의 전자 시스템은 주문형 반도체(ASIC)나 표준 소자(off-the self component)와 같은 하드웨어 부분과 운영체계나 응용 프로그램과 같이 마이크로 프로세서(microprocessor)에서 수행되는 소프트웨어 부분이 어우러져 복합적으로 구성된다. 이러한 고밀도, 고성능의 시스템을 효과적으로 설계하기 위해서는 체계적인 하드웨어와 소프트웨어의 통합설계(Hardware-Software Codesign) 시스템이 필요하며, 이를 위해서는 효율적인 분할방법 및 검증방법의 개발이 필수적이다. 소프트웨어적인 통합검증만을 수행하면 검증에 소요되는 시간이 회로의 복잡도에 비례하여 지수적으로 증가할 뿐만 아니라, 하드웨어에 대한 정확한 모델링이 어려우므로 잠재된 설계 에러가 검출되지 않을 우려가 있다. 에뮬레이션과 시뮬레이션을 함께 이용하여 통합 검증을 수행할 경우, 하드웨어와 소프트웨어를 작은 단위로 분할하면 에뮬레이터와 시뮬레이터간의 통신에 많은 시간이 소모되어 소프트웨어적인 통합 검증보다 느리게 수행될 수 있다. 특히 소프트웨어 부분은 소프트웨어적으로 시뮬레이션을 수행하면서, 하드웨어 부분일지라도 에뮬레이션을 하기 힘든 부분은 VHDL 시뮬레이터를 이용하여 소프트웨어적으로 검증하는 것이 유리하다. 그리고 시뮬레이션을 하기보다는 병렬 처리로 에뮬레이션을 하는 것이 효과적인 부분만을 에뮬레이션하는 것이 빠르고 정확한 검증에 효과적이다.

본 연구에서는 하드웨어와 소프트웨어의 통합 설계 및 검증 실험을 위하여 설계된 RTL VHDL 코드를 입력으로 사용하였다. 그러므로 기존의 통합 설계 툴들과의 인터페이스가 용이하다. 입력된 RTL VHDL 코드는 큰 블록 단위로 통신비용에 중점을 두어 분할하였다. 분할에서는 하드웨어와 소프트웨어간의 통신시간이 최소화되고, 인터페이스가 단순하며 변형이 용이하도록 하였다. 하드웨어 부분은 FPGA를 이용한 로직 에뮬레이터에서 에뮬레이션을 수행하고, 소프트웨어 부분은 PC에서 알고리듬 수준 시뮬레이션을 수행하며, 통신은 PC의 병렬 포트를 이용하여 통합 검증을 수행하였다. 또한 에뮬레이터를 사용하여 테스트를 수행하므로 에뮬레이션하는 회로에 대해서는 정확한 검증을 시뮬레이션(event-driven simulator)보다 빠르게 수행할 수 있다. 그리고 주변 시스템과 연결하여 시스템 수준의 검증을 수행할 수 있으므로 여러 가지 잠재적인 에러도 찾아낼 수 있다. 이와 같이 수행하면 실제로 설계되는 회로가 사용되어지는 주변 환경과 연결되어 실제와 동일한 환경에서 수행될 수 있으므로 시뮬레이션만을 위한 별도의 테스트 벡터의 제작 없이 검증을 수행할 수 있다. 그리고 전체 시스템의 설계 단계에서 일부 하드웨어로 구현이 완료된 부분만은 에뮬레이션하고 나머지 부분들은 C로 시뮬레이션(알고리즘 수준 시뮬레이션)할 수 있다는 장점도 있다. 이러한 방법을 이용하면 설계 기간도 단축시킬 수 있으며, 설계의 초기 단계에서 에뮬레이션을 수행함으로써 오류를 설계 초기에 발견할 수 있는 동시에 컴파일에 소요되는 시간도 줄일 수 있다.

2.1 로직 에뮬레이션 시스템

로직 에뮬레이터는 빠른 시간 내에 효율적으로 설계된 회로를 구현하고 검증하기 위해 하나의 FPGA 보다 많은 게이트 용량을 가지는 프로그램 가능한 하드웨어, 게이트 단위의 설계를 하드웨어에 프로그램하여 구현하는 소프트웨어, 하드웨어에 프로그램된 회로를 에뮬레이션하고 회로의 동작을 해석할 수 있는 디버깅 시스템 등으로 구성된 시스템으로 정의할 수 있다.

본 연구에서 사용한 로직 에뮬레이터의 구조는 논리회로의 구현을 위한 Xilinx 4025EPG299 24개와 외부와의 인터페이스를 위한 4005HPG223 4개 등 28개의 FPGA 칩으로 구성되어있어서 10만 게이트 이상의 회로 구현도 가능하다.[1] 사용된 에뮬레이터 및 컴파일러에 대한 자세한 사항은 참고문헌 [1]에 기술되어 있다.

로직 에뮬레이션 시스템을 이용한 검증은 다음과 같이 수행하였다. 첫째로 RTL VHDL로 기술된 하드웨어 부분은 로직 합성 및 최적화 (Design compiler -Synopsys)를 수행하고 소프트웨어 부분은 알고리듬 수준의 C 언어로 기술하여 PC상에서 컴파일을 수행한다. 둘째로 FPGA 기술 매핑 및 최적화 (FPGA Express -Synopsys)를 수행한 후, 셋째로 컴파일러[1]

에 의해 분할 및 매핑을 수행한다. 넷째로 분할된 네트리스트 파일의 배치와 배선(XACT -Xilinx)을 수행하고, 다섯째로 비트 파일을 PROM 파일로 만들어(XACT -Xilinx) 다운로드를 수행한다. 여섯째로 logic analyzer를 이용하여 출력 값 확인 및 PC와의 통신을 통한 하드웨어 소프트웨어 통합 검증을 수행한다.

2.2 인터페이스 회로의 설계

하드웨어-소프트웨어 통합 검증을 위해 시뮬레이션과 에뮬레이션을 함께 수행하기 위해 그림 2와 같은 인터페이스 회로를 설계하였다. 인터페이스 회로는 에뮬레이터 외부에 두지 않고 RTL-VHDL로 설계되어 검증할 회로와 함께 에뮬레이터 내부에 매핑된다. 그리고 소프트웨어 부분이 검증될 PC와 연결되어 회로의 수정 없이 검증이 수행될 수 있도록 에뮬레이션이 수행될 블록의 입출력과 같은 비트 폭을 갖도록 입출력 신호를 바꾸어 준다. 그리고 handshaking에 필요한 신호를 입출력하며, PC의 병렬 포트와 연결을 위한 버스를 제어한다.

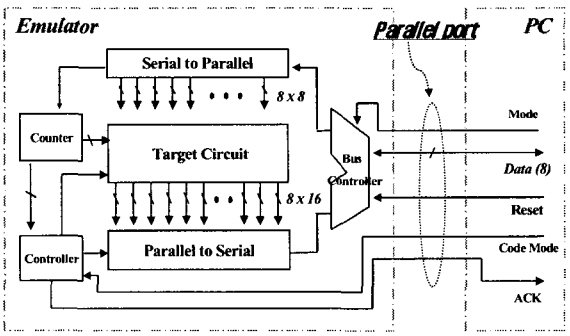


그림 2 에뮬레이터와 PC의 인터페이스

"Bus control" 블록에서는 Mode 신호에 맞추어 데이타의 방향을 결정하여 PC의 병렬 포트와 데이타 통신이 수행되도록 한다. "Serial to Parallel"에서는 병렬 포트를 통해 들어온 8비트의 입력을 에뮬레이션을 수행할 target circuit 블록에서 필요한 비트로 만들어 주고, "Parallel to Serial" 블록에서는 하드웨어의 출력 데이타를 병렬 포트로 보내기 위하여 Ack 신호에 맞추어 8비트로 변환한다. "Controller" 블록에서는 통신에 필요한 Ack 신호와 "Parallel to Serial" 블록을 제어하기 위한 신호를 발생시키며, "Counter" 블록은 target 및 Controller 블록의 수행을 제어하는 제어 신호를 발생한다. 위에서 설명한 인터페이스 회로는 "Serial to Parallel", "Parallel to Serial" 블록의 비트 폭만을 변경하여 합성을 수행함으로써 다른 회로의 검증에도 쉽게 적용할 수 있다. 또한 RTL code로 합성되어 에뮬레이터의 FPGA에 구현되므로 부분적 혹은 전체적인 재구성이 가능하다.

3. RTL 에뮬레이션을 위한 설계 지침

일반적으로 에뮬레이션의 어려운 점은 에뮬레이션을 하기 위한 합성과 FPGA 매핑 및 다운로드 과정이 길며, 각 단계에서 FPGA가 아닌 일반 설계에서는 나타나지 않는 에러가 많이 발생하여 일부 혹은 전체적인 재설계가 필요하다는 점이다. 특히 RTL로 기술된 회로를 이용하여 에뮬레이션을 수행할 경우, 부분적으로 설계를 수정해 주어야만 성공적인 합성 및 정확한 검증을 수행할 수 있다. 그래서 숙련된 기술자가 아니면 재수정을 하는 과정에서 많은 시간을 낭비할 수밖에 없다. 본 장에서는 에뮬레이션을 하기 위한 설계에 대한 지침을 기술하였다. 본 설계지침들을 이용하여 상위 단계에서의 합성부터 이를 고려하여 설계하면 에뮬레이션의 위한 별도의 설계 수정 없이 에뮬레이션을 수행할 수 있고, 에뮬레이션에서 발생할 수 있는 에러를 최소화하여 에뮬레이션을 쉽고 빠르게 할 수 있다. 다음에 기술한 경우들은 ASIC 라이브러리를 이용한 설계에서는 일어나지 않는 에러가 에뮬레이션을 하기 위한 FPGA 구현단계에서 발생하거나, 에뮬레이션을 수행할 때 오동작을 일으킬 수 있는 경우들이다. 다음의 설계 지침을 이용하여 여러 가지 실제 예제를 설계한 후 에뮬레이션을 수행한 결과 빠르고 정확한 검증을 수행할 수 있었다.

1) 불필요한 bidirectional 포트를 제거한다. 출력 값을 다시 읽어 들이는 포트가 포함된 블록을 설계할 때, 이러한 포트를 설계의 편의 또는 코드를 간단히 하기 위해 일반적으로 inout 또는 buffer mode로 선언하여 설계하는데 이는 에뮬레이션을 위한 합성 및 매핑 과정에서 오류를 유발한다.

첫째로, inout port로 선언할 경우에는 이 포트는 bidirectional 포트 또는 three-state bus로 해석되어, 로직 합성 단계 중 패드 셀을 삽입하는 과정인 insert pad 과정에서 오류를 유발하게 된다.

둘째로, FPGA 매핑단계(XACT 6.0) 수행 중 PPR (partitioning, placement and routing) 단계에서 global buffer의 신호가 직접 output buffer에 연결되는 것은 오류로 간주한다. 그래서 이를 buffer mode로 선언하여 기술하면 블록의 출력단에 output buffer를 연결하여, 분할 결과에 따라 FPGA 내부의 global buffer와 output buffer가 직렬로 연결되어 오류를 발생시키는 경우도 발생하게 된다.

이러한 에러를 막기 위해 출력을 다시 읽는 신호선을 추가로 선언하여 연결해 주면 합성 또는 매핑 단계의 에러 없이 정확한 결과를 가져오는 것을 실험으로 알 수 있었다.

2) 저전력 설계 등을 목적으로 사용되는 Gated clock은 그림 3(a) 의 경우와 같이 클록이 논리 게이트를 통과하여 플립플롭의 입력으로 들어가도록 설계하는 것을 말한다. FPGA는 특히 노이즈에 민감하고 지연 시간이 크기 때문에 gated clock은 에뮬레이션에서 클럭의 지연시간을 증가시키고 clock skew와 glitch를 일으키며 hold-time violation을 발생을 발생시키는 원인이 되며 FPGA 내부의 추가 배선 자원을 사용하게 된다. Gated clock이 있는 회로를 에뮬레이션을 통하여 정확히 검증하기 위해서는 회로를 VHDL로 기술할 때 high-fanout 용 글로벌 신호 중 하나에 Gated clock을 할당하여 구현할 수 있으나, CLB (configurable logic blocks)의 사용율이 떨어지게 된다. 가장 좋은 방법은 플립플롭의 클럭의 enable 신호를 이용하여 그림 3(b)의 경우처럼 변환하는 것이다. 대부분의 FPGA 내에 포함되어 있는 플립플롭은 clock enable 신호를 가지므로 기술 매핑과정에서 gated clock 회로를 clock enable을 가지는 회로로의 변환이 가능하고, 이를 통하여 clock skew, glitch 및 timing violation을 제거하는 데도 유용하다.

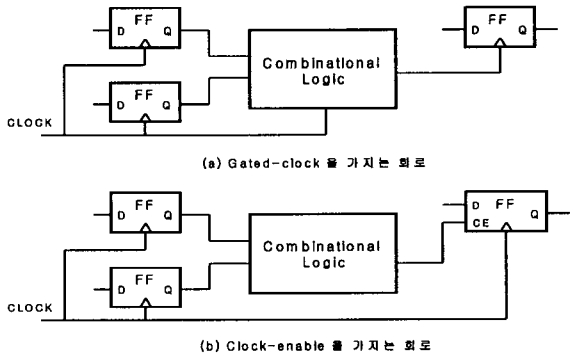


그림 3 Gated clock을 가지는 회로의 최적화

3) 하나의 블록 크기를 작게 설계해야 한다. 블록이 커지면 분할하여 여러 개의 에뮬레이션 보드에서 구현하여야 하므로 보드간의 연결이 복잡해지고 지연 시간이 증가할 수 있다. 하나의 보드를 이용하여 에뮬레이션을 수행하기 위해 실험적으로 하나의 블록의 크기는 20 ~ 40 K 게이트(ASIC 라이브러리로 합성했을 경우) 정도가 적합하다. 또한 팬아웃이 큰 네트의 수를 가능하면 최소화시켜야 빠른 검증을 수행할 수 있다.

4) 설계의 편의와 하드웨어의 단순화를 위해 많은 Tristate-Net을 사용하는 것은 FPGA 사용율을 저하시켜 에뮬레이션을 어렵게 한다. 일반적으로 FPGA는 Tristate-Net를 위한 별도의 채널을 두고 이를 지원하지만, 이 채널의 수가 한정되어 있기 때문에 과도한 Tristate-Net의 사용은 FPGA의 사용율에 있어 심각한 저하 요인이 된다. 예를 들면 FPGA에는 1,024개의 CLB(configurable logic block)가 들어 있고, 각 CLB 내에는 두 개의 Tristate-Net을 위한 채널이 있다. 이 경우 Tristate-Net로 기술된 RTL 코드를 MUX를 이용하여 설계를 수정하면 CLB 내부의 한정된 네트를 사용하지 않게 되어 FPGA 사용율을 높여 더 많은 블록을 에뮬레이션할 수 있다. 그림 4와 같은 회로를 3-state buffer와 MUX를 이용하여 Tristate-Net를 RTL VHDL code로 기술하여 FPGA로 구현하면, 각각의 Tristate-Net을 특정 global net에 정해주는 과정을 생략할 수 있으며 CLB 내부의 global net을 사용함으로써 FPGA 사용율을 증가시킬 수 있다.

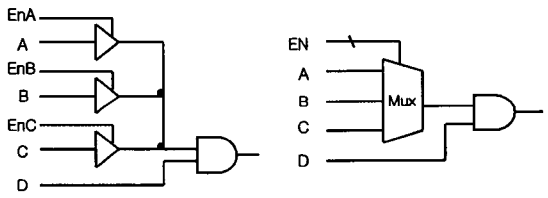


그림 4 Tristate-Net 구현

표 1 Tristate buffer와 MUX를 이용하여 구현한 결과

	PORTS	NETS	CLBS	CELLS
3-state buffer	43	155	7	74
MUX	42	145	9	51

표 1은 16 비트의 Tristate-Net을 Tristate buffer와 MUX를 사용하여 회로를 RTL VHDL code로 설계하여 리드-솔로몬 디코더의 합성 및 매핑을 수행한 결과이다. 표에서 CLB의 수는 조금 증가하였지만, 전체 net의 수와 cell의 수가 크게 줄어든다. 그러므로 MUX를 이용하여 설계하면 적은 FPGA 자원을 사용하여 더 큰 회로의 에뮬레이션을 수행할 수 있다.

5) VHDL을 이용한 RTL 설계에서 case나 if 절이 입력의 모든 가능한 조합을 표현하도록 하여 필요 없는 래치(latch)의 합성을 막는다. case나 if 절이 입력의 모든 가능한 조합을 표현하지 못할 때 즉, if 혹은 case

절에서 마지막 else 문항이나 when other 문항이 빠졌을 경우, 래치가 첨가되어 구현된다. FPGA에서는 조합 논리회로의 feedback을 이용하여 래치를 구현한다. 특히 클록의 변화를 조건으로 갖는 설계에서 가능한 모든 조합을 기술하지 않으면 레지스터가 래치로 구현된다. 에뮬레이션을 수행할 때 래치가 있는 설계는 레지스터들을 가진 설계보다 더 많은 로직을 요구하게 되고 더 낮은 성능을 가질 수밖에 없으므로, 에뮬레이터의 용량을 초과하지 않고 빠른 속도로 검증을 수행하기 위해서는 래치의 합성을 막아야 한다.

6) RTL VHDL code를 기술하는 과정에서 정수형은, 특히 음수를 갖게 기술된 port를 가지는 설계는 에뮬레이션을 위해 FPGA로 구현되는 과정에서 최적화되지 않거나 혹은 부정확한 결과를 초래할 수 있으므로 벡터형으로 바꾸어 준다. 또한 최상위 블록에서 정수형으로 기술하여 연결한 블록은 FPGA 라이브러리를 가지고 합성한 뒤 연결이 끊어진 결과를 얻게됨을 실험을 통하여 알 수 있었다.

표 2는 정수형(정수범위는 -1에서 6)을 이용하여 -1에서 6까지 계수하는 카운터를 구현 것과, 0에서 7까지 계수하도록 벡터형(std_logic_래치(2 downto 0), 3bit)을 이용하여 구현한 카운터의 합성 결과를 비교한 것이다. 위의 결과는 같은 최적화를 수행하였을 경우에 정수형을 이용하여 구현한 결과가 훨씬 더 많은 자원이 필요하다는 것을 알 수 있다. 즉, 에뮬레이션을 수행하기 위해 FPGA로 구현할 때 벡터형을 이용하여 RTL code를 기술하여야만 최적화된 구현 결과를 얻을 수 있다. 설계에 따라서는 부호를 필요로 하는 경우가 있는데 이 때는 크기를 비교하거나 부호 비트를 추가함으로써 해결 할 수 있다. 예를 들어 정수형으로 -2에서 2의 범위를 가지는 포트는 3 비트의 벡터형으로 바꾼 뒤 음수가 필요한 부분은 2 이하, 0은 3, 양수는 4이상을 확인하게 설계를 수정하여 기술함으로써 해결할 수 있다.

7) 논리식은 가능하면 간단히 기술하여야 한다. 특히, 에뮬레이션 속도를 향상시키기 위해서는 한 클럭 주기 동안에 수행되어야 하는 논리식의 지연시간이 최소가 되어야 한다. 그래야만 지연시간이 최소가 되도록 FPGA에 쉽게 구현할 수 있어서 빠른 속도로 에뮬레이션을 수행할 수 있다. 논리식이 최소 지연시간을 가지도록 설계하기 위해서 retiming 등의 설계 방법을 잘 활용하는 것이 효과적이다.

8) Global 신호선을 잘 활용하도록 한다. 클럭이나 reset 등 많은 수의 핀을 가지거나 최소의 지연시간을 필요로 하는 신호선은 보통의 신호선과 같이 처리하면 비효율적이며 구현하는데 많은 연결선을 필요로 하고, 구현하였을 때 지연시간이 크게 증가하는 등의 어려운 점이 있다. 보통의 FPGA 칩과 에뮬레이터들은 이러한 신호선을 별도로 처리하기 위한 수 개의 global 신호선을 제공한다. 그러므로 이러한 신호들은 global 신호선에 할당하여 컴파일 시간을 줄이면서 효과적으로 구현하는 것이 바람직하다.

표 2 카운터를 구현한 결과의 비교

# of / type	PORTS	NETS	CELLS
정수형 사용	6	23	21
벡터형 사용	5	15	12

4. 실험 결과

RS 코드는 random error 와 burst error를 정정할 수 있는 가장 강력한 error control code 중의 하나로써, 통신 회로 및 디지털 데이타 저장 회로 설계에 중요한 역할을 수행한다. 본 실험에 이용한 RS 인코더/디코더 회로[15]의 구조는 그림 5와 같이 인코딩, modified syndrome 다항식, erasure locator 다항식을 계산하는 하드웨어 블록들로 이루어져, modified Euclid 알고리듬을 수행한다. 비디오/오디오의 내부/외부 코드를 디코딩 가능하도록 내부에 MUX를 사용하여, 여러 가지의 RS 코드를 디코딩 할 수 있다. 디코더 회로는 크게 다섯 개의 블록들이 파이프라인 방식으로 동작된다.

Synopsys사의 ASIC 라이브러리를 사용하여 합성하였을 경우 RS CODEC 회로는 58,667 게이트로 합성되어 100,000 게이트 급의 에뮬레이터[1]에서 용이하게 구현 될 수 있는 것처럼 보이나, 일반적으로 ASIC 게이트 수에 비해 에뮬레이션 게이트가 3배 정도 크며, 또한 그림 5에서 알 수 있듯이 FIFO (Delay buffer)등 많은 수의 메모리 소자로 인해 CLB 사용율이 저하되어 에뮬레이션을 어렵게 한다.

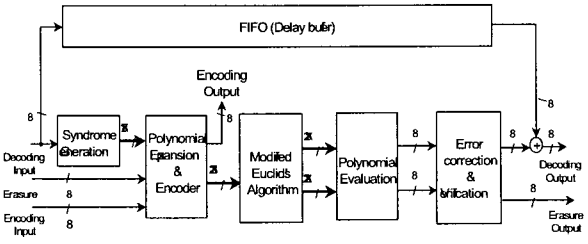


그림 5 RS 인코더/디코더 구조

modified Euclid's 알고리듬 수행 블록은 하나의 codeword(예를 들어 비디오의 outer 코드는 149 X 8비트)를 수행할 때 8비트의 데이타 12 개를 입력으로 받아 수행하여 8 비트의 데이타 24개를 출력한다. 반면에 다른 블록들은 수행 도중에 계속 출력을 내보내거나 입력을 읽으므로 modified Euclid's 알고리듬을 수행하는 블록의 전, 후단에서 통신에 소요되는 비용이 가장 낮음을 알 수 있다. 표 3에서 보듯이 modified Euclid's 알고리듬 수행 블록의 게이트가 제일 많다.

그림 6은 앞에서 언급한 검증 기법을 리드-솔로몬 인코더/디코더[15]에 대해 적용한 것으로 계산량이 많은 modified euclid 블록은 에뮬레이션을 수행하고 나머지 블록, 즉 제어 신호가 많은 블록과 FIFO 메모리 블록 등은 시뮬레이션을 수행하여 빠르게 검증할 수 있도록 구현한 것이다. 이와 같이 분할하는 것이 통신에서의 overhead를 줄여 전체적으로 우수한 하드웨어-소프트웨어 통합 설계 결과를 얻는다.

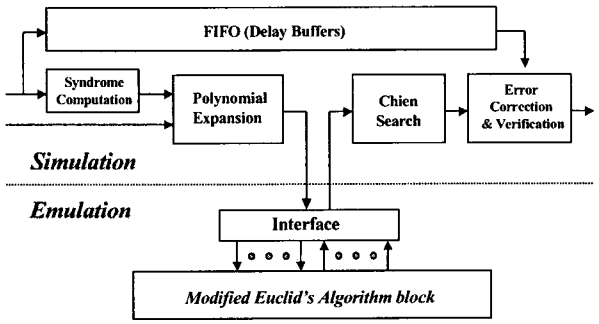


그림 6 RS 인코더/디코더의 분할 및 통합 검증

Reed-solomon codec 회로는 디코딩 모드로 고정시키고, 코드 모드는 시뮬레이션으로부터 입력으로 받아들여 비디오 신호(inner 149/138, outer 85/77)에 대하여 검증을 수행하였다. RTL 시뮬레이션 결과 전체 블록의 검증 시간은 outer 코드 1 frame (11,472개의 테스트 벡터)에 대하여 9분 30초가 걸리고 그중 modified Euclid's 알고리듬을 수행하는 블록은 3분 3초가 걸려 이 블록의 계산량이 다른 블록에 비하여 많은 것을 알 수 있었다. 그래서 modified Euclid's 알고리듬 수행블록만을 에뮬레이션을 수행하면, 통신에 걸리는 시간을 합하여 14,800 μsec로 RTL 시뮬레이션의 3분 3초와 비교하여 약 12,300배 빠르게 수행함으로써 전체 검증 시간을 최대한 줄였다. 에뮬레이션이 시뮬레이션보다 12,000배 이상 빠른 이유는 시뮬레이션이 소프트웨어 프로그램에 의하여 순차적으로 수행되는데 비하여 에뮬레이션은 FPGA에 프로그램되어 하드웨어로 병렬로 동작되기 때문이다.

또한 이 블록만 에뮬레이션을 수행함으로써 에뮬레이터의 용량을 초과하는 것을 방지하고, 컴파일 시간도 줄일 수 있었다. modified Euclid's 알고리듬 수행 블록을 제외한 나머지 블록들은 PC상에서 C 언어로 설계하여 알고리즘 수준 시뮬레이션을 수행한다.

리드 솔로몬 디코더를 검증하기 위하여 인코딩과 에러의 첨가를 소프트웨어적으로 수행하여 테스트 벡터의 작성을 별도로 하지 않았고, 검증결과를 확인하기 위해 전체를 C로 시뮬레이션한 결과와 비교하여 검증 결과가 정확한지 확인하였다. 에뮬레이션은 1MHz에서 수행되었다. 시뮬레이터 및 에뮬레이터와의 통신을 제어하는 부분은 PC에서 (Pentium) Visual C++을 이용하였으며, 비교를 위한 VHDL simulation은 SUN Workstation(Ultra Sparc)에서 Synopsys VSS를 이용하여 수행하였다.

표 3 리드-솔로몬 인코더/디코더를 ASIC 라이브러리를 이용하여 합성한 결과

	Syndrome Computation	Polynomial Expansion	Modified Euclid	chien search	Error Correction and Verification	Delay buffer
Gate Count	1,750	6,046	18,885	4,760	9,726	17500

표 3은 리드-솔로몬 인코더/디코더를 Synopsys의 ASIC 라이브러리로 합성하였을 때의 게이트의 수를 나타낸 것이다. 위에서 상대적으로 modified Euclid's 알고리듬을 수행하는 블록의 크기가 크다는 것을 알 수 있다.

표 4는 1 frame의 outer code(11,472개의 테스트 벡터)에서, 비디오 1 frame 그리고 4 frame에 대해서 본 연구에서 제안한 방법으로 검증하였을 경우와 Syno-

표 4 리드-솔로몬 디코더에 대한 수행시간 비교

method / 벡터수 (8bit)	RTL simulation	Our method
11,472개	9분 30초(100 %)	4분 53초(51 %)
24,137개	21분 (100 %)	8분 47초(41 %)
72,411개	87분 (100 %)	35분 (40 %)
2,413,700개	2235 분 (100 %)	1053 분 (47 %) (Emulation time 1470분+ Build time 6 시간

psys를 이용한 RTL 시뮬레이션의 수행시간과 비교한 것이다.

표 4의 마지막 줄은 100개의 frame에 대해서 검증을 수행한 결과이며, 이때 제안된 기법에 의한 수행 시간은 에뮬레이션을 위한 합성 및 매핑 등의 준비 시간을 합한 것이다. 표에 나타난 것처럼 에뮬레이션을 위한 준비 시간을 합해도 본 논문의 방법이 상용 시뮬레이터에 비해 2배 이상 빠르게 검증함을 알 수 있다. 이는 modified Euclid's 알고리듬 수행 블록이 RTL 시뮬레이션을 수행할 때 보다 에뮬레이션을 수행할 때가 12,000배 이상 빠르고, 알고리즘 수준(C 언어)에서 수행된 블록들이 RTL 시뮬레이션 보다 약간 빠르기 때문이다.

5. 결 론

본 논문에서는 에뮬레이션과 시뮬레이션을 동시에 사용하여 두 방법의 단점을 보완하고, 설계 검증 시간을 단축시키기 위한 방법을 제안하였다. 제안된 방법을 이용하면 RTL 시뮬레이션에 비해 빠르고 효율적인 설계 검증을 수행할 수 있었다. 특히 크기가 큰 회로의 에뮬레이션에 필요한 컴파일 시간도 상당히 줄일 수 있고, 테스트 벡터를 모델링하는 번거로움도 덜 수 있다.

또한 에뮬레이터를 이용한 하드웨어 구현을 용이하게 할 수 있도록 여러 가지 설계 지침들을 제안하였다. 이를 이용하여 설계하면 에뮬레이션을 더욱 빠르고 정확히 수행할 수 있고, 에뮬레이션을 위한 재설계 시간도 줄일 수 있다. 본 방법들을 사용하여 system-level의 검증을 용이하게 수행 할 수 있고, 하드웨어와 소프트웨어간의 통신에 소요되는 비용이 중요한 통합설계에서 하드웨어와 소프트웨어의 검증을 동시에 수행할 수 있다.

실제 예제를 이용하여 실험한 결과 본 방법들을 사용하면 설계 검증을 빠르고 정확하게 수행할 수 있는 것으로 나타났다.

참 고 문 헌

[1] C. Kim and H. Shin, "A Performance-Driven Logic Emulation System: FPGA Network Design and Performance-Driven Partitioning," IEEE Transactions on CAD/ICAS, Vol. 15, No 5, pp. 560-568, May 1996.

[2] Z. Barzilai, J. L. Carter, B. K. Rosen, and J. D Rutledge, "Hss- a high-speed simulator," IEEE Trans. on CAD/ICAS, pp. 601-617, July 1987.

[3] C. Hansen, "Hardware logic simulation by compilation," Proc. ACM/IEEE 24th Design Automation Conference, pp. 712-715 June 1987.

[4] C. J. DeVane, "Efficient circuit partitioning to extend cycle simulation beyond synchronous circuits," Proc. IEEE/ACM Int. Conf. on CAD, pp. 154-161 Nov 1997

[5] Tom Blank, "A Survey of Hardware Accelerators Used in Computer-Aided Design," IEEE Design & Test, Vol. 1, Aug. 84, pp. 21-39.

[6] F. Nacabal, O. Deygas, P. Paulin, and M. Harrand, "C-VHDL co-simulation: Industrial requirements for embedded control processors," in Proceedings, EuroDAC/EuroVHDL Designer Sessions, pp. 55-60, 1996.

[7] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabbara, Hardware-Software Co-Design of Embedded Systems: The Polis Approach, Kluwer Academic Publishers, 1997.

[8] J. Buck, S. Ha, E. Lee, and D. Messerschmitt, "Ptolemy: a framework for simulating and prototyping heterogeneous systems," Int. Journal of Computer Simulation, vol. 4, Apr. 1994.

[9] J. Varghese, M. Butts and J. Batcheller, "An Efficient Logic Emulation System," IEEE Trans. on VLSI System, VOL. 1, NO. 2, pp. 171-174, June 1993

[10] J. Babb, R. Tessier and A. Agarwal, "Virtual Wires: Overcoming Pin Limitations in FPGA-based Logic Emulators," IEEE Workshop on FPGA for CCM, pp. 142-151, 1993.

[11] H. Verheyen and G. Lara, "Rapid Prototyping Using System Emulation Technology for DSP Validation," Proc. EDA&T '95, August, 1995

[12] M. Borgatti, R. Rambaldi, G. Gori, and R. Guerrieri, "A smoothly upgradable approach to virtual emulation of HW/SW systems," Proc. Int. Workshop on Rapid System Prototyping, June 1996.

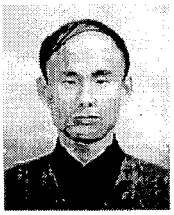
[13] J.-H. Hong, S.-A. Hwang, and C.-W. Wu, "An FPGA-based hardware emulator for fast fault emulation," in Proc. Midwest Symp. on Circuits and Systems, Aug. 1996.

[14] Jerry Bauer, Michael Bershteyn, Ian Kaplan and Paul Vyedin, "A Reconfigurable Logic Machine for Fast Event-Driven Simulation," Proc. ACM/IEEE 24th Design Automation Conference, June 1998.

[15] S. Kwon, H. Shin, "An Area-Efficient VLSI Architecture of A Reed-Solomon Decoder/Encoder for Digital VCRs," IEEE Trans. on Consumer Electronics, Vol. 43, No 4, pp. 1019-1027, Nov. 1997.

이 종 석

1998년 2월 한양대학교 전자공학과 학사. 2000년 2월 한양대학교 대학원 전자공학과 석사. 현재 현대전자 메모리연구소 응용제품팀 연구원. 관심분야는 로직에뮬레이션 시스템 설계, Signal integrity 등임.



김 충 희

1991년 2월 한양대학교 전자공학과 졸업. 1993년 2월 한양대학교 대학원 전자공학과 공학석사. 1997년 2월 한양대학교 대학원 전자공학과 공학박사. 1998년 9월 ~ 1999년 9월 미국 U.C. Berkeley Post-Doc. 1992년 3월 ~ 현재 한양대학교 공학기술연구소 연구원 관심분야는 VLSI CAD, 디지탈 시스템 설계 등임.

신 현 철

1978년 2월 서울대학교 전자공학과 졸업. 1980년 2월 한국과학기술원 전기 및 전자공학과 공학석사. 1987년 미국 U.C. Berkeley 전기 및 컴퓨터공학과 공학박사. 1980년 ~ 1983년 금오공과대학 전자공학과 전임강사 조교수. 1983년 ~ 1987년 Fulbright scholarship. 1987년 ~ 1989년 미국 AT&T Bell Laboratories, Murray Hill 연구원. 1989년 9월 ~ 현재 한양대학교 전자공학과 교수. 관심분야는 집적회로 설계자동화, 디지털 신호처리 시스템 설계 등임.