論文2002-39SD-4-10

초고집적 FPGA 디버깅의 문제점 및 해결책

(Debugging Problem for Multi-Million Gates FPGAs and the Way to Solve It)

梁 世 陽 *

(Seiyang Yang)

요 약

최근의FPGA는 매우 높은 집적도와 빠른 동작속도 때문에 많은 응용분야에서 널리 사용되고 있다. 그러나, FPGA에 구현된 설계를 디버깅하는 과정은, FPGA의 내부에 존재하는 수많은 신호선들을 탐침 하는 과정이 매우 오랜 시간을 요하는 FPGA 재-컴파일을 최소 수 차례 이상 필요로 함으로서 많은 문제점을 가지고 있다. 본 논문에서는, 이와 같은 FPGA 디버깅의 문제점을 분석하고, 새로운 디버깅 방법을 제안한다. 제안되는 방법은 FPGA 내부에 존재하는 모든 신호선들에 대한 100% 탐침을 한 차례의 FPGA 재-컴파일 과정 없이도 수행하는 것을 가능하게 할 뿐만 아니라, 한번의 FPGA 컴파일 과정으로 최소 한 개의 설계 오류를 찾을 수 있도록 한다. 본 논문에서 제안된 방법은 실험을 통하여서도 매우 효과적이며 실용적임이 확인되었다.

Abstract

As today's field programmable gate arrays have very large logic capacity as well as relatively fast operation speed, they're widely used in many application areas. However, debugging the design implemented in FPGA's is very time-consuming and painful as the internal signal probing usually requires large number of FPGA re-compilations, which take tremendously long time. In this paper, we analyze the problems in FPGA debugging and propose a new powerful debugging solution. With the proposed FPGA debugging solution, we can guarantee not only to provide 100% internal signal visibility without FPGA re-compilation for the design in FPGA's, but also to identify at least one design bug per FPGA compilation. An experimental result has clearly shown the proposed approach to FPGA debugging very powerful and practical.

Key words : ASIC, VLSI-CAD, FPGA, 설계검증, 로직에뮬레이션

* 正會員, 釜山大學校 電子電氣情報 컴퓨터工學部
(School of Electrical & Computer Engineering, Pusan National University)

※ 본 연구는 부분적으로 반도체설계교육센터(IDEC)의 설계 툴 지원과 부산대학교 컴퓨터정보통신연구소의 지능형단말기 연구센터(정보통신부 지원 IT 분야 기술인력양성 센터)의 연구비 지원에 의하여 이루어졌습니다.

接受日字:2001年8月1日, 수정완료일:2002年4月2日

Ⅰ. 서 론

최근의 반도체 집적기술의 빠른 발전은 SRAM 기술을 채용한 FPGA의 성능 및 용량을 크게 증가시킴으로 FPGA가 게이트어레이(gate array)를 대체할 수 있을 수준을 넘어서 프로세서 코어를 자체 내장한 시스템급의 FPGA로까지 발전하게 하고 있다.[1,2] 이로 인하여 FPGA는 현재 유무선 통신, 초고속 네트워킹, 멀티미디어, 디지털가전, 자동차 분야에서 폭 넓게 사용되고 있

을 뿐만 아니라 이의 활용은 앞으로 내장형 시스템(embedded system) 전 분야로 확대될 것으로 예상된다. 뿐만 아니라 현재의 SOC(System-On-a-Chip)에서의 치솟는 설계복잡도는 전체 설계 과정에서 검증을 위하여 전체 시간과 비용의 70%까지를 투입하여야만 하는 심각한 설계 검증(design verification) 위기 상황을 초래하고 있는데, FPGA를 이용한 에뮬레이션이나 프로토타이핑을 이용하여 설계 검증을 수행하는 경우에는 시뮬레이터를 이용하여 설계 검증을 수행하는 경우에 비하여 최소 10,000배에서 최대 1,000,000배까지의 검증 속도 향상을 얻을 수 있어서 FPGA를 이용한 하드웨어 기반의 설계 검증도 매우 보편화되는 추세이다.[3,8~11]

과거의 PLD(Programmable Logic Device)와 같은 집적도가 크지 않은 프로그래머블 칩들을 사용하는 경우에는 설계된 회로를 프로그래머블 칩에 구현하기 앞서서 회로의 동작 검증을 위하여 시뮬레이션을 수행하지 않고 프로그래머블 칩을 집적 수행시켜 동작 검증을 수행하는 것이 일반적이었으나, 최근의 고집적 FPGA에 설계된 회로를 구현하기에 앞서서는 ASIC 설계와 같이 시뮬레이션을 수행하여 설계 검증을 먼저 소프트웨어적으로 수행하는 것이 필요하다. 그러나 최근의 높은 설계 복잡도에서 소프트웨어적인 시뮬레이션으로 설계 검증을 수행하는 것은 시뮬레이션 시간의 폭발적인 증가를 초래하는 문제점이 있을 뿐만 아니라, FPGA가 사용되는 타겟 환경에 대한 정확한 모델링의 어려움 등으로 인하여 매우 제한적일 수 밖에 없다. 이와 같은 문제점들의 구체적인 예를 언급하면, 우선 표 1에 있는 것과 같이 현재의 일반적인 설계 복잡도를 갖는 설계들에서 시뮬레이션은 더 이상 time-to-market을 만족시키는 검증 방법이 될 수 없다. 구체적으로 이 표에 따르면 설계된 100MHz 펜티엄 호환 프로세서를 검증하기 위하여 윈도우 운영체제를 부팅하는 과정을 로직시뮬레이션으로 수행하는 경우에 19년 정도가 소요되는 것으로 나타나 있다. 뿐만 아니라, 타겟 보드에 FPGA이외에 SDRAM, 마이크로프로세서 등이 함께 사용되는 경우에 이와 같은 주변 상황을 정확히 모델링하여 시뮬레이션하는 것은 적어도 많은 시간과 노력이 투입되어야만 함으로 이로 인하여 추가적인 비용의 증가를 초래하게 된다.

이상과 같은 이유에서 FPGA에 구현되는 설계를 단기간 내에 완벽하게 검증하기 위해서는 실제 환경에서

표 1. 예를 통한 시뮬레이션 수행시간 비교 [자료처: 멘토그래픽스]

Table 1. Examples of Simulation Time [Source : Mentor Graphics].

Design	Task	Logic simulation run time
100MHz Pentium	Boot Windows	19 years
Echo-canceling ASIC	Execute 1 minute of real time	1 year
MPEG2 video processor	Decode single frame	16 days

FPGA에 구현된 설계를 수행시켜서 올바른 동작을 하는지를 검증하는 것이 필수적이다. 그러나 이와 같은 수행 과정 중에서FPGA에 구현된 설계를 디버깅하기 위해서는 FPGA 내부에 존재하는 수 많은 사용자 회로의 신호선들에 대한 탐침(probing)이 반드시 필요하게 되는데, FPGA 내부에 존재하는 이와 같은 신호선들에 대한 탐침은 수행시간이 극히 오래 걸리는FPGA 재컴파일 과정(기술매핑, 배치 및 배선, 비트스트림 생성)을 수반하게 되는 심각한 문제점을 가지고 있으며, 이로 인하여 설계 검증에 투입되는 시간과 비용의 과도한 증가를 초래하게 되는데 이와 같은 문제는 고집적 FPGA에서 더욱 심각하게 된다. 본 논문에서는 이와 같은 FPGA 디버깅 과정에서의 문제점들을 분석하고, 이들 문제점들을 효과적으로 완벽하게 제거할 수 있는 새로운 해결책을 제안한다. 제안되는 디버깅 방법을 사용하는 경우에 설계자는 FPGA 내부에 존재하는 모든 신호선들에 대하여 FPGA에 대한 재컴파일 과정 없이 100% 가시도(visibility)를 완벽하게 보장받을 수 있을 뿐만 아니라, 기존의 디버깅 방법들은 설계오류(버그)를 발견하기 위하여 다 수의 FPGA 컴파일 과정을 요구하는데 비하여 설계에 설계오류가 존재하는 경우에는 한번의 FPGA컴파일을 통하여서도 1개 이상의 설계오류를 발견할 수 있게 된다. 본 논문에서 제안된 해결책들의 효능을 검증하는 실험을 수행한 결과는 본 논문에서 제안된 방법들이 FPGA 디버깅의 문제점을 매우 효과적으로 완벽하게 제거하는 방법이 될 수 있음을 보여준다.

앞으로 본 논문의 구성은 다음과 같다. 2장에서는 FPGA 디버깅의 문제점과 현재 사용되는 방법들에 대하여 알아보고, 3장에서는 본 논문에서 제안되는 방법

과 이 방법을 이용한 디버깅에서의 장점들을 설명한다. 4장에서는 본 논문에서 제안된 방법을 적용하여 실험을 수행한 결과에 대하여 언급하고, 마지막으로 5장에서 결론을 언급한다.

Ⅱ. FPGA 디버깅

1. 문제점

FPGA에 구현된 설계에 대하여 이를 실제 동작시키면서 디버깅하는 과정에서는 설계에 존재하는 수 많은 내부 및 외부 신호선들에 대한 탐침과 분석이 반드시 필요하다. 현재까지 이를 위한 가장 일반적인 방법은 FPGA 외부에서 별도의 상용 논리분석기를 사용하는 것이지만 이와 같은 논리분석기를 사용하여 탐침을 수행하는 것은 더 이상 현실적이지 못하게 되었다. 왜냐하면, 최근의 고집적 FPGA는 많은 수의 사용자 입출력 핀(user I/O pin)을 확보하기 위하여 마이크로 BGA(Ball Grid Array)나 플립칩(flip-chip) 팩키징을 사용하고 있는데 이와 같은 팩키징 방식에서는 인쇄회로기반(PCB) 위에 장착된 칩 외부에서 핀들을 직접 억세스하는 것이 원천적으로 불가능하기 때문에 억세스를 위한 별도의 커넥터 등을 보드에 장착하여야만 한다. 현재 FPGA에 구현되는 설계의 복잡도가 매우 큰 경우에 FPGA 내부에 존재하면서 디버깅 시에 탐침 대상이 될 수 있는 신호선들의 수는 많게는 수십만 내지는 수백만개가 될 수 있는데 이와 같은 많은 수의 신호선들에 비하여 탐침을 위하여 확보될 수 있는 커넥터의 핀 수는 상대적으로 극히 적을 수 밖에 없다. 이와 같은 상황에서는 탐침 대상이 되는 모든 신호선들을 한번에 탐침용 커넥터에 연결시킬 수 없음으로 이들 신호선들을 그룹으로 나누어 번갈아 가면서 탐침을 수행하여야 한다. 이를 위해서는 해당 탐침 대상 내부 신호선들을 탐침용 커넥터 핀들에 연결된 FPGA 칩의 특정 핀들에 연결시켜야만 하는데, 이 과정은 FPGA 컴파일(compilation)이라 부르는 기술매핑, 배치및배선, 비트스트림생성 단계들을 모두 거쳐야만 한다. 이 단계들 중에서 특히 배치및배선 과정은 고집적 FPGA의 경우에는 수 시간 이상의 수행 시간을 요구하는 과정으로 컴파일 시간의 대부분을 차지한다. 이 뿐만 아니라 FPGA 디버깅 도중에서는 그때까지의 디버깅 결과에 따라서 새로운 내부 신호선들에 대한 새로운 혹은 추가적인 탐침의 필요성이 자주 발생하게 된다. 이와 같은 상황에서도 해당 내부 신호선들을 FPGA 칩의 특정 핀들에 연결시키는 재컴파일(re-compilation) 과정이 수 차례에서 수십차례까지 필요하다.

이상과 같은 전형적인FPGA 디버깅 과정의 흐름도가 그림 1에 나타나 있다. 맨 처음, FPGA에 설계되어진 대상을 구현하기 위하여 컴파일(그림 1에서 왼쪽 상단) 과정이 필요하며, 이어서 설계가 구현되어진 FPGA를 수행(그림 1에서 왼쪽 중단)하고 수행되어진 결과를 바탕으로 설계에 설계오류들이 존재하는 경우에 이들 설계오류가 발견될 때(그림 1에서 왼쪽 하단)마다 HDL 코드에서 설계오류를 초래하는 코드 세그먼트를 찾아서 이를 수정하고 난 후(그림 1에서 왼쪽 상단)에 컴파일부터 같은 과정들을 반복하게 된다. 그러나, FPGA 디버깅 과정에서 설계오류를 발견하는 과정은 시뮬레이션에서의 디버깅 과정과는 다르게 매우 높은 비용을 초래하게 되는 과정이다. 즉 설계오류를 찾는 그림 1의 왼쪽 하단을 구체화시킨 것이 그림 1의 오른쪽에 나타나 있다. 그림에서 나타나 있듯이 이 과정에서도 또 다른 컴파일 과정(그림 1에서 오른쪽)이 있다. FPGA에 구현된 설계 대상에 존재하는 설계오류를 찾기 위해서는 FPGA 내부의 특정 신호선들을 탐침하여야 하는데 앞에서 이미 언급한 것처럼 내부 신호선 탐침을 위하여 확보된 탐침핀들의 수가 FPGA 내부에 존재하는 신호선들의 수에 비하여 상대적으로 매우 적음으로 특정 시점에 이들 탐침핀들에 연결시킬 수 있는 내부 신호선들의 수도 적을 수 밖에 없다. 따라서 탐침이 필요한 내부 신호선들의 수가 탐침핀의 수보다 많은 경우에는 탐침이 필요한 내부 신호선들을 수 차례에 걸쳐서 순차적으로 탐침핀에 연결하는 방법밖에 없음으로 이와 같은 과정에서 수 차례의 컴파일과 수행 과정들이 반복되어져야 한다. 또한 이와 같이 특정 시점에 탐침핀들로 연결시키게 되는 내부 신호선들을 FPGA 내부에 존재하는 전체 신호선들 중에서 선정하는 것도 또 다른 부수적인 컴파일과 수행 과정들이 필요하다. 즉, FPGA 디버깅 과정에서 이와 같은 탐침대상 신호선들을 선정하는 것은 결과에서 원인을 거꾸로 분석하는 과정에서 수행되어 지는 것임으로 예측을 통하여 이루어지게 된다. 따라서 이와 같은 예측이 잘못되어진 것으로 판정되어지는 시점에서는 탐침 대상으로서 다른 신호선들에 대한 선정이 필요하거나 추가적인 신호선들의 선정이 필요하게 된다. 이는 또 다른 별도의 컴파일과 수행 과정들이 필요하게 되는 결과를 초래하게

된다.

결론적으로 FPGA에 구현된 설계를 디버깅하는 목적으로 통상 생각할 수 있는 논리분석기는 FPGA 내부에 존재하는 신호선들의 탐침을 위하여 많은 횟수의 FPGA 컴파일 과정을 필요로 함으로서 디버깅 시간을 크게 증가시키는 치명적인 문제점을 가지고 있다.

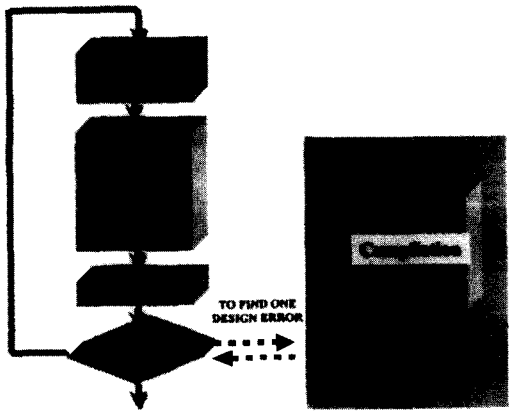

그림 1. 전형적인 FPGA 디버깅 과정의 흐름도
Fig. 1. Conventional FPGA Debugging Flow.

2. 현재의 해결책

FPGA 디버깅에서의 상기와 같은 문제에 대하여 Xilinx나 Altera, 혹은 Actel과 같은FPGA 벤더들의 해결책은 논리분석기를 FPGA 내부에 내장(embedding)시키는 것으로 모아진다.[4~6] 이와 같은 내장형 논리분석기(embedded logic analyzer)의 사용은 FPGA 패키징에 상관없이 JTAG(Joint Test Access Group)[13] 포트를 이용하여 내부 신호선들에 대한 접근을 용이하게 한다는 장점을 제공하지만, 위에서 이미 언급되어진 FPGA 디버깅에서의 문제점을 완전히 해결하는 방법이라고 할 수 없다.

즉, 이들 내장형 논리분석기들이 탐침할 수 있는 최대 신호선의 수는 정해져 있으며(예로 Xilinx의 ChipScope는 최대 256개의 신호선까지만 탐침 가능함으로 초고집적 FPGA에 구현되는 회로들에 존재하는 신호선들에 비하여 극히 적은 신호선들만 FPGA 재컴파일 과정 없이 탐침할 수 있음), 이들 탐침을 하고자 하는 신호선들을 반드시 디버깅 수행에 앞서서 미리 지정하여야만 한다. 따라서 디버깅 과정에서 탐침을 하고자 하는 내부 신호선들이 새롭게 나타나게 되는 경우에는 내장형 논리분석기를 사용하는 경우에도 최소한 부분적인 FPGA 재컴파일 과정을 피할 수는 없을 뿐만 아니라, FPGA의 CLB(Configurable Logic Block)의 LUT(Look-Up Table) 내부로 할당되는 신호선들에 대해서는 탐침이 원천적으로 불가능하다. 뿐만 아니라, 내장형 논리분석기는 FPGA 내부에 구현되어져서 FPGA의 자원을 사용함으로 한번에 탐침 가능한 신호선들의 수와 신호값들을 저장하는 버퍼의 크기가 외장형 논리분석기보다 크게 부족하여 폭 넓은 검증시간대에서의 탐침이 불가능하다는 약점도 가지고 있다.

Ⅲ. 집적검증 기법을 이용한 FPGA 디버깅

1. 집적검증 기법

집적검증 기법(IV: Integrated Verification)은 에뮬레이션 기반의 검증과 시뮬레이션 기반의 검증을 효과적으로 통합시킨 새로운 설계검증 기술로, 검증브리지(verification bridge)[7,12]를 이용하여 에뮬레이션과 시뮬레이션 간에 실시간으로 수행 전환을 자유롭게 이루어지게 함으로서 에뮬레이션 기반의 검증의 장점과 시뮬레이션 기반의 장점 모두를 취할 수 있는 매우 강력한 설계검증 기술이다. 검증브리지는 FPGA가 장착된 임의의 보드와 PC 혹은 워크스테이션 상의 임의의HDL 시뮬레이터를 연결시키고 FPGA에 설계검증 대상회로와 함께 탐침용 부가회로를 내장시켜서FPGA를 수행시키는 도중의 임의의 시점에서 설계검증 대상회로의 상태(모든 기억소자(storage element: 즉 플립플롭이나 래치)의 값)를 HDL 시뮬레이터로 순간적으로 이전시켜서 시뮬레이션을 임의의 시점 이후에서 연속적으로 가능하도록 한다. 뿐만 아니라 시뮬레이션 도중의 임의의 시점에서 설계검증 대상회로의 상태를 FPGA로 순간적으로 이전시켜서 FPGA에 구현된 설계검증 대상회로를 임의의 시점 이후에서 연속적으로 수행시키는 것도 가능하다.[7,12] 이와 같은 것들을 가능하게 하기 위하여 검증브리지는 그림 2와 같이 임의의 FPGA 보드를 FPGA의 사용자 I/O 포트를 통하여 제어할 수 있도록 하드웨어적으로 구성된 보드 제어기인 하드웨어 부분과 HDL 시뮬레이터와의 인터페이스를 담당하는 소프트웨어 부분으로 크게 나눌 수 있으며, 이들 하드웨어 부분과 소프트웨어 부분은 버스 드라이버(예로 PCI 드라이버)로 연결되어 있다.

이와 같은 검증브리지를 통하여 FPGA를 통한 에뮬레이션 수행과 시뮬레이터를 이용한 시뮬레이션 수행 간에 동작 전환이 순간적으로 일어나기 위해서는 임의의 시점에서 FPGA에 구현된 설계검증 대상회로의 상태를 추출하여 보드 제어기와 PCI 인터페이스로 개인용컴퓨터에 전송되어 HDL 시뮬레이터에서 컴파일된 설계검증 대상회로의 시뮬레이션 초기 조건으로 사용될 수 있어야 하며(에뮬레이션에서 시뮬레이션으로의 전환시), 혹은 반대로 HDL 시뮬레이터로 수행되는 도중의 임의의 시점에서 설계검증 대상회로의 상태를 시뮬레이터에서 추출하여 PCI 인터페이스와 보드 제어기를 통하여 FPGA에 구현된 설계검증 대상회로의 초기 조건으로 사용될 수 있어야(시뮬레이션에서 에뮬레이션으로의 전환시) 한다.

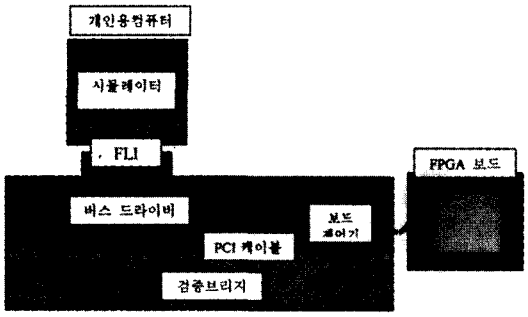

그림 2. 검증브리지를 이용한 FPGA 디버깅 환경
Fig. 2. FPGA Debugging Environment with Verification Bridge.

시뮬레이터는 원래 소프트웨어적인 모델의 특성 상, 설계검증 대상회로의 모든 신호선들에 대한 100% 제어(controllable)와 관측(observable)이 가능함으로 별 문제가 없지만, FPGA에 구현된 설계검증 대상회로에 대하여 상태 정보를 임의의 시점에서 관측하고 제어하기 위해서는 추가적인 탐침 부가회로가 부가 되어져야만 한다. 그림 3은 이와 같은 탐침 부가회로가 설계검증 대상회로에 부가된 구현 예이다. 이상과 같은 탐침용 부가회로를 통하여 FPGA를 이용한 에뮬레이션 시에도 설계검증 대상회로에 모든 기억소자들에 대한 100% 제어와 관측이 가능해진다. 이상과 같은 탐침용 부가회로와는 별도로, 에뮬레이션에서 시뮬레이션으로의 전환 시점을 결정하기 위하여 본 논문에서는 설계검증 대상회로에 존재하는 신호선들 중에서 트리거 대상 신호선들을 선정하고 이들 신호선들을 트리거용 부가회로에 입력으로 사용하여 특정 조건이 만족 될 경우에 에뮬레이션에서 시뮬레이션으로 전환이 가능하도록 하였다.

FPGA 디버깅에 이와 같은 집적검증 기법을 적용하면 앞에서 지적된FPGA 디버깅의 문제점들을 완벽하게 해결하는 것이 가능하다. 최근의 HDL 시뮬레이터들은 C/C++ 언어와의 인터페이스를 PLI(Programming Language Interface)나 FLI(Foreign Language Interface)를 통하여 제공하고 있음으로 C 언어를 사용하여 FPGA 보드와 HDL 시뮬레이터 중간의 인터페이스를 매우 용이하게 구성하는 것이 가능하다.

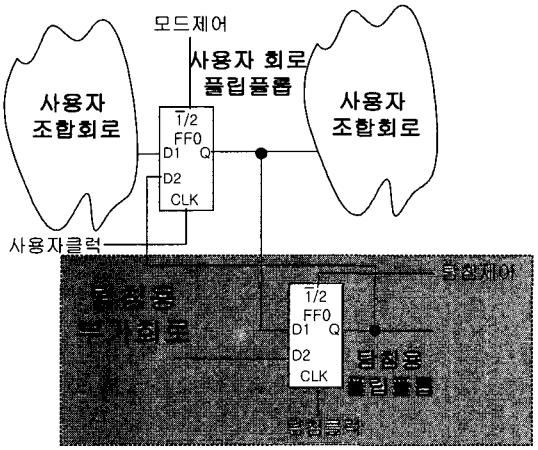

그림 3. 탐침 회로가 부가된 설계검증 대상회로의 예
Fig. 3. An Example of User Design for Verification.

제안되는 FPGA 디버깅 방법을 사용하는 경우의 흐름도는 그림 4와 같다. 그림 4는 FPGA 디버깅의 문제점으로 지적되었던 그림 1의 오른쪽 과정들을 완벽하게 제거한 상태이다. 이의 의미는 본 논문에서 제안되는 FPGA 디버깅 방법을 사용하게 되면 설계오류를 찾기 위하여 단 한번의 추가적인 컴파일이나 수행 과정들이 필요하지 않음을 보장할 수 있다는 것이다. 이를 다르게 설명한다면, 설계검증 대상에 한 개 이상의 설계오류들이 존재하는 경우에 본 논문에서 제안되는 FPGA 디버깅 방법은 한번의 컴파일에서 최소 한개 이상의 설계오류를 발견할 수 있음을 보장하는 방법이다. 디버깅 환경만을 보았을 때는 시뮬레이션이 100% 신호가시도(signal visibility)를 제공하게 됨으로 에뮬레이션에서의 디버깅 환경과는 비교할 수 없을 정도로 완벽하다고 할 수 있다. 그러나 현실적으로 수백만 게이트급의 설계검증에서 시뮬레이션으로 회로 내에 존재하는 모든 신호선들에 대하여 신호가시도를 확보하기 위해서는 모든 신호선들을 탐침대상으로 선정하여야만

하는데 이는 가뜩이나 느린 시뮬레이션의 속도를 더욱 떨어뜨리게 됨으로서 현실적으로는 시뮬레이션에서의 높은 신호가시도 확보는 최소 수차례의 시뮬레이션을 반복적으로 수행하여야만 가능하다. 본 논문에서 제안된 집적검증 기법을 이용한 FPGA 디버깅 환경은 시뮬레이션과 같이 100% 신호가시도를 확보할 수 있을 뿐만 아니라, 동시에 검증 과정의 대부분을 에뮬레이션 방식으로 수행하게 됨으로서 시뮬레이션 속도의 일백만배 속도를 쉽게 얻을 수 있음으로 시뮬레이션에서의 디버깅 환경보다도 훨씬 강력하다고 할 수 있다.

다음에는 구체적으로 집적검증 기법을 이용하면 어떻게 이와 같은 궁극적인 디버깅 환경을 제공할 수 있게 되는가를 설명한다.

2. 재컴파일 없는 100% 신호 가시도 보장

시뮬레이션의 최대 장점은 디버깅의 용이성이라 할 수 있는데, 이와 같은 디버깅의 용이성은 시뮬레이션이 시뮬레이션 도중에 추가적인 시뮬레이션 컴파일 과정 없이 설계된 회로에 존재하는 모든 신호선들에 대하여 100% 신호가시도가 보장된다는 것에 기인한다. 반면에 지금까지의 순수한 FPGA 디버깅 과정은 디버깅 과정에서 모든 신호선들에 대한 100% 신호가시도를 확보하기 위해서는 극히 수 많은 컴파일이 필요로 함으로서 현실적으로는 불가능하다.

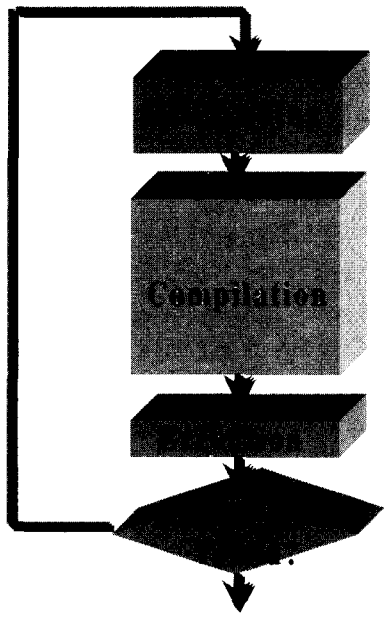

그림 4. 본 논문에서 제안된 FPGA 디버깅 과정의 흐름도
Fig. 4. Proposed FPGA Debugging Flow.

그러나 FPGA디버깅 과정에서 설계검증 대상에 존재하는 수많은 신호선들을 탐침 하고자 하는 경우에 집적검증 기술을 이용하여 FPGA상에서 수행 중인 설계의 동적 이미지(동적 이미지란 검증대상이 대상 설계에 존재하는 모든 기억소자(storage element)들에 대한 특정 시점에서의 값들을 말함)를 검증브리지를 거쳐서 HDL 시뮬레이터로 이전시키면 새로운 신호선들의 탐침을 위하여FPGA 재컴파일 과정을 완전히 생략하면서 FPGA 내부에 존재하는 모든 신호선들에 대한 100% 탐침이 가능해진다. 즉, HDL 시뮬레이터로 이전된 설계의 동적 이미지를 이용하여 시뮬레이션을 수행하는 과정을 통하여 설계검증 대상에 존재하는 어떠한 신호선들에 대해서도 탐침을 수행하게 되며, 탐침이 완료된 후에는 역으로 시뮬레이터 상에서 수행 중인 설계의 동적 이미지를 검증브리지를 거쳐서 FPGA로 이전시켜 고속에서의 추가적인 수행이 FPGA 상에서 연속적으로 가능하게 한다.

3. 컴파일당 최소 하나의 버그 발견 보장

FPGA 디버깅에서 FPGA에 구현된 설계에 존재하는 모든 신호선들에 대한 탐침을 재컴파일 과정 없이 진행하는 것으로는 완벽한 디버깅 환경을 구성하는 데에는 한계가 있다. 즉, FPGA 내부에 존재하는 모든 신호선들에 대하여 재컴파일 과정이 필요하지 않는 100% 가시도를 제공하는 경우라도 다음과 같은 문제점이 있다.

설계자가 FPGA 디버깅을 수행하면서 특정 조건이나 특정 시점에서 내부에 존재하는 특정 신호선들에 대한 탐침을 원하게 되는 경우에 본 논문에서 제안된 방법은 3.2에서 설명된 바와 같이 집적검증 기법을 이용하여 그 순간에 HDL 시뮬레이터의 도움을 받아 탐침을 수행하게 된다. 만일 이와 같은 과정 중에 설계 오류의 확인을 위하여 과거에 탐침대상에 선정되지 못했던 신호선들에 대하여 현재부터가 아닌 특정 과거시점에서의 탐침이 요구된다고 하면 FPGA의 수행을 처음부터 다시 시작하여야 한다. 다시 말하면, 일반적인 디버깅 과정은 시간축 상에서 항상 앞으로만 진행되지는 않으며 매우 빈번하게 과거로 다시 되돌아가 디버깅을 진행하여야만 하는 것이 일반적이다. 이와 같은 상황은 시뮬레이션에서도 적용이 될 수 있다. 시뮬레이션에서 이를 지원하는 유일한 방법은 시뮬레이션 대상이 되는 회로 내에 존재하는 모든 신호선들을 탐침 리스트에 포함시키고 시뮬레이션을 진행하는 것이지만, 이와 같

이 시뮬레이션을 수행하게 되면 시뮬레이션의 수행시간이 더욱 폭발적으로 증가하게 됨으로 수십만-수백만 게이트급의 설계에 대하여 실제 적용하는 것은 불가능하다.

따라서 일반적으로 디버깅 과정에서는 탐침의 필요성이 있는 일부의 신호선들을 "예측"하여 이들만을 대상으로 탐침을 수행하고 미래에 추가적인 신호들에 대한 탐침이 필요하다면 이들 또한 예측하여 탐침을 수행하게 되는 반면에, 과거에 탐침 대상으로 예측된 이외의 다른 신호선들에 대한 탐침을 위해서는 디버깅 대상이 되는 대상을 처음부터 재수행을 하는 것이 필요하다. 그러나 집적검증 기술을 이용하여 FPGA상에서의 수행, 즉 에뮬레이션과 HDL 시뮬레이터 상에서의 수행을 통합하게 되면 수행시간을 FPGA가 수행되는 수십 MHz의 실시간으로 유지하면서도 수행시간 전 과정 상에서의 모든 신호선들에 대한 탐침이 가능하게 된다. 이와 같은 탐침 방법의 가장 큰 특징은 FPGA 수행에 앞서 앞으로의 디버깅 과정에서 탐침의 필요성이 있는 일부의 신호선들을 예측하는 과정을 완전히 생략하고 설계자가 자유롭게 선정하는 특정 시간대에서의 특정 신호선들에 대한 탐침을 수행할 수 있게 한다. 따라서 이와 같은 FPGA 디버깅 환경은 하드웨어 기반의 디버깅 환경임에도 불구하고 소프트웨어적인 시뮬레이션 디버깅 환경보다 훨씬 강력하다고 평가될 수 있다.

이상과 같은 집적검증 기술을 이용한FPGA 디버깅 환경에서는 설계검증 대상에 설계 오류가 존재하는 경우에 한번의 FPGA 컴파일을 통하여 적어도 하나이상의 설계오류를 제거하는 것이 가능하다.

Ⅳ. 실 험

본 논문에서 제안된 FPGA 디버깅 기법의 효능을 검증하기 위하여 두 가지 설계사례를 가지고 실험을 진행하였다.

실험사례 1에서는 정지 이미지에 대한 윤곽선검출(edge detection) 회로를 VHDL로 설계하여 FPGA (Xilinx XCV150-6PQ240)에 구현하고 디버깅을 수행하였다. 전체 환경 구성은 PC를 FPGA와 SRAM(4MB)이 장착된 프로토타이핑 보드와 병렬케이블로 연결시켜 구성하였다(그림 5). 동작 과정은 우선 PC에 저장되어 있는 정지 이미지를 병렬케이블을 통하여 보드 상의 SRAM에 저장시키고 난 후에 FPGA에 구현된 윤곽선검출 회로를 동작시켜서 윤곽선검출을 수행하고 결과 이미지를 SRAM에 저장시킨 후에 병렬케이블을 통하여 PC로 역전송 하는 것으로 되어 있다. 이 전체의 과정 수행을 위해서는 총 125,000,000개의 사용자 클럭이 12.5MHz 클럭속도로 공급되어야 한다.

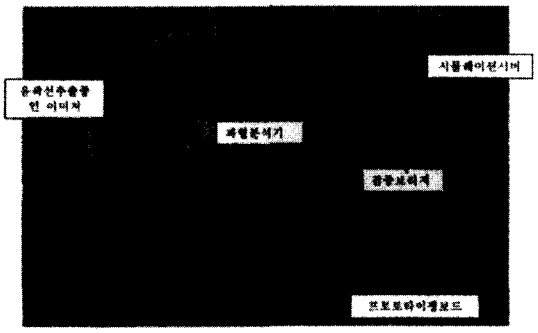

그림 5. 윤곽선검출 예제의 FPGA 디버깅 환경 구성
Fig. 5. FPGA Debugging Environment for An Edge Detection Example.

FPGA 디버깅을 위하여 윤곽선검출 회로에 고의적으로 설계 오류를 삽입하였는데 이 설계 오류는 이미지의 가로, 세로 크기를 뒤바꾸어 줌으로서 설계에 포함되어지게 하였다. 디버깅을 위하여 우선 보드를 12.5MHz로 동작시켜서 약 10초 이내에 전체 수행을 한 번만 진행시키고 윤곽선검출을 실제 수행하는 시점에서 윤곽선검출 회로에서 SRAM을 억세스하기 위하여 생성하는 어드레스버스 신호선들을 모두 탐침한 결과 주소의 생성이 바르게 되고 있지 않음을 확인할 수 있었다. 이의 원인을 분석하고자 추가적인 탐침을 다른 시간대에 다른 신호선들에 대하여서 즉각적으로 수행하는 과정에서 이미지 크기를 저장하고 있는 레지스터의 값이 올바르지 못함을 예측할 수 있었으며, 최종적으로 이 레지스터에 내용을 탐침함으로서 설계 오류가 PC에서 전송하는 이미지의 가로, 세로 크기 파라메터를 뒤바꾸어 특정 레지스터에 로딩하고 있는 것이 확인되었다. 이상과 같이 FPGA 디버깅 과정에서 FPGA에 대한 재컴파일을 하지 않고도 FPGA 내부에 존재하는 모든 신호선들에 대하여 수행 전 과정에 걸쳐서 현재부터 뿐만 아니라 과거 시간 영역에서도 자유자재의 탐침을 할 수 있게 됨으로서 FPGA 재컴파일 과정을 전혀 거치지 않고서 완벽한 디버깅을 신속하게 수행할 수 있음이 실제적으로 확인되었다.

실험사례 2에서는 미리 알려져 있는 설계오류에 대한 디버깅을 행한 실험사례 1과는 달리 실제 국내의 설계업체 한 곳을 방문하여 실험을 수행하였다. 실험 대상 회로로서는 이 회사의 SOC 칩에 사용되는 ARM7TDMI 코어를 위한 외부 메모리콘트롤러를 선정하였다. 이 외부 메모리콘트롤러는 Verilog로 설계되어 Synplify Pro로 합성되어 최종적으로 Xilinx XCV2000E-6BG560 칩에 구현되었다. 이 실험에서는 제 3자에 의하여 삽입된 설계오류를 설계자에게 미리 알리지 않고 본 논문에서 제안된 방법을 이용하여 FPGA 디버깅을 수행하였다. 정상적인 동작에서는 프로토타이핑 보드에 장착된 LED가 규칙적으로 깜빡임을 반복하여야 하나 설계오류에 의하여 이것이 나타나지 않았다. 곧 이어진 내부신호선들의 연속적인 탐침을 통하여 UART에 있는 특정 레지스터의 값이 잘못되었음이 즉각적으로 발견되었으며 이의 원인도 설계자에 의하여 즉각 분석이 되었다.

실험사례 1과 실험사례 2를 통하여 검증브리지를 이용하여 본 논문에서 제안된 FPGA 디버깅 기법의 효능이 실제적으로 확인되었다. 즉 기존의 통상적인 로직 디버깅 방법인 논리분석기를 이용한 FPGA 디버깅을 수행하는 경우에 설계오류를 찾기 위하여 소요되는 디버깅 시간을 본 논문에서 제안된 디버깅 기법을 채용함으로써 대폭적으로 단축시킬 수 있음이 실증적으로 확인되었다. 본 방법을 적용하기 위하여서는 이미 설명된 바와 같이 FPGA에 설계검증 대상회로만이 아니라, 탐침용 부가회로와 트리거 부가회로가 같이 내장되어야만 한다. 이와 같은 부가회로는 면적과 속도 면에서 오버헤드가 됨으로 이에 대한 분석이 필요하다. 그림 3에서 보는 것과 같이 탐침용 부가회로는 설계검증 대상회로에 존재하는 기억소자 당 하나의 추가적인 플립플롭이 필요하게 되며, 트리거 부가회로는 트리거 대상 신호선을 몇 개나 선정하는가에 따라 달라질 수 있다. 그러나 일반적으로 최근의 복잡도가 큰 회로 설계에서는 탐침용 부가회로의 오버헤드가 전체 오버헤드에 대부분을 차지하게 되며, 실제로 실험사례 1과 실험사례 2에서는 트리거 대상 신호선을 64개를 선정하여 실험을 진행한 결과 탐침용 부가회로의 오버헤드가 오버헤드의 대부분을 차지 하였다. 구체적으로 면적 오버헤드는 구현시 소요되는 FPGA의 CLB(Configurable Logic Block) 개수로 측정한 결과 설계검증 대상회로만을 구현하는 경유보다 평균 30% 정도의 증가치를 보였는데, 최근의 고집적 FPGA에서는 이와 같은 오버헤드는 크게 문제가 되지는 않을 것으로 판단된다. 속도에서는 부가 회로를 내장하고 나면 평균 10% 정도의 성능 저하가 초래되는 것으로 관측되었는데, 이 또한 프로토타이핑이나 에뮬레이션에서는 크게 문제가 될 수치는 아니다.

V. 결 론

초고집적FPGA에 구현된 사용자 시스템을 디버깅하기 위해서는 FPGA 내부에 존재하는 수 많은 사용자 회로의 신호선들에 대한 탐침과 분석이 반드시 필요하게 되는데, FPGA 내부에 존재하는 이와 같은 신호선들에 대한 탐침은 수행시간이 극히 오래 걸리는FPGA 재컴파일 과정을 수반하게 되는 심각한 문제점을 가지고 있으며, 이로 인하여 설계 검증에 투입되는 시간과 비용의 과도한 증가를 초래하게 되며 이와 같은 문제는 고집적 FPGA에서 더욱 심각하게 된다.

본 논문에서는 이와 같은 FPGA 디버깅의 문제점들을 분석하고, 집적검증 기법을 이용하여 이들 문제점들을 효과적으로 완벽하게 제거할 수 있는 새로운 해결책을 제안하였다. 제안되는 디버깅 방법을 사용하는 경우에 설계자는 FPGA 내부에 존재하는 모든 신호선들에 대하여 FPGA에 대한 재컴파일 과정없이100% 가시도를 보장받을 수 있을 뿐만 아니라, 설계에 버그가 존재하는 경우에는 한번의 FPGA컴파일을 통하여 어떠한 경우라도 반드시 1개 이상의 설계 오류를 발견할 수 있게 한다. 본 논문에서 제안된 해결책들의 효능을 검증하는 실험을 수행한 결과는 본 논문에서 제안된 방법들이 FPGA 디버깅의 문제점을 매우 효과적으로 완벽하게 제거하는 방법이 될 수 있음을 보여주고 있다.

참 고 문 헌

[1] Virtex-II Platform FPGA Datasheet, Xilinx (http://www.xilinx.com), 2001.

[2] Excalibur SOPC FPGA Datasheet, Altera (http://www.altera.com), 2001.

[3] Certify Datasheet, Synplicity(http://www.synplify.com), 2000

[4] ChipScope Datasheet, Xilinx (http://www.xilinx.com), 2001.

[5] SignalTap Embedded Logic Analyzer Datasheet, Altera (http://www.altera.com), 2001.

[6] SiliconExplorer Dasheet, Actel (http://www.actel.com), 2001.

[7] 양세양, "시뮬레이션과 에뮬레이션의 결혼: 검증 위기의 새로운 희망," 대한전자공학회 CAD및 VLSI연구회 신진박사논문발표 및 종합학술대회 발표논문집, 2000

[8] J. Babb, et al., Logic Emulation with Virtual Wires, *in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, June 1997.

[9] G. Ganapathy, et al, "Hardware Emulation for Functional Verification of K5," *Proceeding of 33rd Design Automation Conference*, June 1996.

[10] Accelerating Functional Closure Using Solidify and Hardware Emulation, Whitepaper, Averant Inc.(http://www.averant.com), 2001.

[11] N. Kim, et al, "Virtual Chip: Making Functional Models Work on Real Target Systems," *Proceeding of 35th Design Automation Conference*, June 1998.

[12] Seiyang Yang, "Probing Apparatus and Probing Method Using the Same, and Mixed Emulation/Simulation Based on It" US Patent Pending, 1999.

[13] ANSI/IEEE Std 1149.1-1990 Standard Test Access Port and Boundary-Scan Architecture (included IEEE Std 1149.1a-1993 and IEEE Std 1149.1b-1994), IEEE Standards, Piscataway, N.J., USA.

저 자 소 개

梁 世 陽(正會員) 論文誌 第39卷 SD編 第2號 參照