FPGA 프로토타이핑, 소프트웨어가 관건이다

소프트웨어는 100% 완전하게 만들기가 어려운 법이다. 소프트웨어의 예상치 못했던 버그는 운영체제, 애플리케이션 및 하드웨어를 하나로 통합해야 하는 통합 과정의 복잡성에 의해 흔히 발생한다. 여기에 독특한 환경을 제공하는 엣-스피드 기반의 FPGA 프로토타이핑을 적용한다면 가장 중요한 통합 단계에서 정밀한 소프트웨어 테스트를 수개월간 추가로 실시할 수 있을 만큼 전체 개발 일정을 효율화할 수 있을 것이다

글 : 더그 에이모스(Doug Amos) / Synplicity Europe
www.synplicity.com

FPGA 프로토타이퍼(Prototyper)들에게 있어 테이프 아웃(Tape out; 칩 설계 검증 완료 단계) 시점을 지키는 것은 무엇보다 중요하다. 이들은 정해진 기한 내에 버그 없는 ASIC 디자인을 준비할 수 있을 때 가장 큰 보람을 느낀다. 이런 요구로 인해 칩 설계를 효율화할 수 있는 대안으로 FPGA 프로토타이핑 기법이 점점 더 많은 사람들에게 알려지고 있고 또 인정받고 있다. 칩 설계 과정에서 제품 및 기능의 차별화와 직접 관련된 소프트웨어의 중요성 역시 날로 높아지고 있는데, 프로토타이핑 기법을 이용하면 ASIC 또는 SoC에 임베디드 되는 소프트웨어를 프로젝트 초기 단계부터 실제 동작속도(엣-스피드)로 하드웨어와 통합할 수 있어 효율적이다.

소프트웨어와 FPGA 프로토타이핑

EDA 분석가인 개리 스미스(Gary Smith)가 관찰한 결과(**그림 1** 참조)에서도 나타나듯이 칩 설계 과정에서 소프트웨어의 중요성은 갈수록 그 정도를 더하고 있다. SoC에서 더 나은 CMOS 기하학 노드(Geometry Nodes)를 사용하거나 설계의 복잡성이 더욱 늘어날수록, 소프트웨어가 개발 과정을 효율화하고 제품의 기능을 차별화하는 데 있어 가장 중요한 요소임을 알 수 있다.

최근 출시되는 다양한 디지털 기기들은 수많은 임베디드 프로세서와 수십만 혹은 수백만 라인의 코드로 구성된 복잡한 SoC에 의해 구동된다. 최소한 3개의 ARM 프로세서를 사용하는 것으로 알려진 애플사의 아이폰(iPhone)이 좋은 예가 될 것이다.

이런 제품을 출시할 때는 시장적시공급(Time to market)이 아주 중요한 요소가 된다. 하지만 이 경우 모든 소프트웨어를 어떻게 제한된 시간 안에 SoC의 맥락에서 검증할 수 있을 것인가? SoC 개발 담당자들은 최근 들어 적시 공급이 필요한 제품들을 주어진 시간에 맞춰 개발하고 검증하기 위한 해결책으로 FPGA 기반의 프로토타이핑 기법을 점점 더 많이 사용하고 있다. 그 가치와 효용성을 인정하고 있기 때문이다.

왜 프로토타입인가?

소프트웨어는 100% 완전하게 만들기가 어려운 법이다. 소프트웨어의 예상치 못했던 버그는 운영체제, 애플리케이션 및 하드웨어를 하나로 통합해야 하는 통합 과정의 복잡성에 의해 흔히 발생한다. 여기에 독특한 환경을 제공하는 엣-스피드 기반의 FPGA 프로토타이핑을 적용한다면 가장 중요한 통합 단계에서 정밀한 소프트웨어 테스트를 수개월간 추가로 실시할 수 있을 만큼 전체 개발 일정을 효율화할 수 있을 것이다. 만약 반도체 회사들이 고객 인도 기간(customer acceptance trials)에 FPGA 프로토타입을 활용하고, 더 많은 소프트웨어 연구조직에서 FPGA 프로토타입이 활용된다면 그 혜택은 배가될 것이다.

부연하자면, 신플리시티의 상용 HAPS(High-speed ASIC Prototyping System)와 같은 고품질 FPGA 보드가

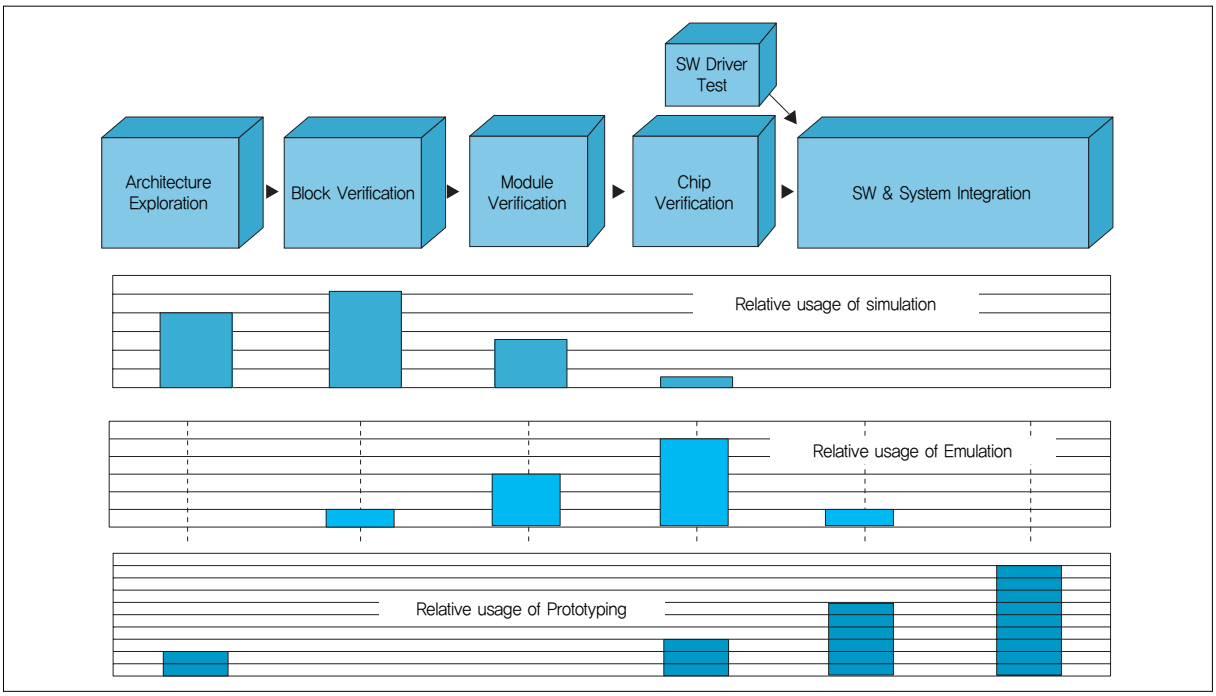

그림 1. 프로젝트 진행 중 다른 검증 기법의 상대적 사용률

공급되면서 프로토타입의 효용성은 시간에서 경제성 측면으로 확대되고 있다. 즉, "더 많은 보드 사용 = 더 많은 버그 발견"의 등식이 성립되는 것이다. 소프트웨어는 사람들의 손길이 더해지면 더해질수록 오염되거나 훼손될 가능성이 커지게 마련이다. 소프트웨어 엔지니어가 미처 상상하지 못했던 방식으로 운영체제를 망가뜨리는 사람이 있는 가 하면, 애플리케이션 간의 충돌과 같이 심각한 문제도 일상적으로 일어난다.

프로토타이핑 시점

SoC 설계 완료 이전에 디버깅 작업을 마칠 수 있다면 FPGA 프로토타이핑의 이점은 명백하게 나타날 것이다. 그림 1은 규모가 큰 ASIC 프로젝트의 다양한 단계 중에 시뮬레이션, 에뮬레이션 및 프로토타이핑에 부여되는 전형적인 강조점을 나타낸 것이다. 여기서 알 수 있듯이 FPGA 프로토타입은 대부분 통합 단계에서 빠른 속도와 적절한 용량이 필요할 때 사용된다.

소프트웨어 통합과정에서 발생한 문제를 해결할 수 있는 방법이 '하드웨어를 변경하는 것뿐'이라는 상황을 가정해 가장 이상적인 해결책을 찾아보자. 이 사례에서는 사이클이 부족한 DSP 알고리즘을 보조 프로세서나 맞춤형 로직 디바이스(Tailored device logic)로 추출하는 과정이 포함될 수 있을 것이다. 하지만 문제 해결을 위해 하드웨어의 변경이 필요하다는 것을 알아냈는데 이미 SoC가 테이프 아웃 되었거나 개발이 완료되어 샘플링까지 끝난 후라면, 변경 작업이 최종 제품에 어떤 이득을 주든지 상관없이 작업 자체가 실행될 가능성은 아주 낮다.

사실 FPGA 프로토타이핑을 더 이른 시점부터 적용할수록 적시에 디자인을 변경하는 데 긍정적인 영향을 미칠 것이다. 또한 FPGA 프로토타입의 사용 빈도가 늘어나는 사용자는 프로젝트 시작 단계에서부터 이런 하드웨어와 소프트웨어 파티셔닝간의 균형점을 찾고자 프로토타입을 시도해보게 된다(그림 1).

프로토타이핑의 보완 과제

하나의 ASIC을 위한 FPGA 프로토타입을 구현하려면 다양한 개발상의 문제를 해결해야 한다. 하드웨어를 만드는 것은 실제로 그리 어렵지 않다. HAPS와 같은 맞춤형 ASIC 프로토타이핑 보드를 여러 업체로부터 구입할 수 있기 때문이다. 진짜 문제는 FPGA 안에서 그 설계를 구현하려 할 때 발생한다.

파티셔닝 및 I/O 처리

현재 가장 큰 FPGA는 각각 2백만 개 이상의 ASIC 게이트를 처리할 수 있지만, FPGA보다 크게 설계되는 ASIC들이 여전히 더 많다. 따라서 SoC의 중요 부분만 프로토타입을 제작하거나 설계를 여러 FPGA로 파티셔닝 해 진행할 필요가 있다. 이렇게 되면 FPGA를 검증하거나 나중에 FPGA를 조합하는 과정에서 몇 가지 장애가 발생할 수 있기 때문에 ASIC RTL을 가능한 최소한으로 변경하여 파티셔닝 하는 것이 목표가 되어야 한다.

설계를 파티셔닝 해 진행하면 인위적인 경계가 생긴다. 또 와이드 인터널 버스(Internal Bus)나 와이드 데이터패스(Datapath) 파티셔닝을 포함한 설계의 경우, 검증에 필요한 I/O 핀 수가 폭증할 수 있다. 그런데 각각 1,000개 이상의 I/O가 있는 최신 FPGA를 사용할 때라도 FPGA 핀은 제한적일 수밖에 없는 자원이라는 게 문제다.

그러나 동일한 FPGA 핀에 여러 신호를 먹싱(Muxing)한 후, 대상 FPGA에서 디먹싱(Demuxing) 하면 여분의 I/O 리소스를 만들 수 있다. 다시 강조하지만 자동 핀 멀티플렉싱(Pin-Multiplexing)이 수행되도록 I/O 리소스를 더 만들기 위해 ASIC RTL을 변경하려는 것이 아니다. 이는 분명히 약간의 성능 저하를 가져오지만 FPGA에는 빠른 I/O가 있어서 멀티플렉싱 수행 시에도 여전히 복잡한 임베디드 소프트웨어를 실제 동작속도로 검증할 수 있다.

RTL 변경없이 토포그래픽 변경(Topographical Change)

약간의 수동 조절 작업과 설계자의 지식을 이용하면 하나 이상의 FPGA에 하위 블록을 복제하여 I/O 한계를 경감시키는 방식으로 I/O 사이의 상호 연결 요구를 감소시킬 수 있다. 하위 수준 게이트를 비트 슬라이싱(Bit-slicing) 하거나 복잡한 블록을 지퍼링(Zippering) 하는 것과 같이 RTL에 대해 보다 전문적인 작업을 해주면(둘 다 RTL 변경 없이 가능함) FPGA I/O에 대한 필요성이 현저히 감소한다.

ASIC 클록과 FPGA 클록은 같지 않다

SoC 설계를 FPGA로 포팅할 때 발생하는 또 다른 문제는 원래 RTL을 만든 사람이 FPGA에 대해 거의 또는 전혀 고려하지 않았을 수 있다는 것이다. ASIC 셀 인스턴스 생성이나 RAM BIST(메모리 내장 자체 시험)를 RTL에 내장한 FPGA의 적대적 개체 생성의 경우를 예로 들 수 있는데, 여기에서 가장 중요한 문제는 IP와 클록의 복잡성이다.

FPGA는 절전을 위해 ASIC에서 흔히 사용되는 게이트 클록 설계 방식(gated-clock design styles)을 지원하도록 발전해왔다. 대신 클록 게이팅 신호(clock gating signals)는 FPGA 하드웨어에서 클록 인에이블 상태로 변환되어야 한다. 이러한 수동 변환 작업은 매우 지루하면서도 새로운 버그를 생성할 위험이 있어 주의해야 한다. 몇몇 합성 툴(Synthesis tool)은 수동 변환 대신, 복잡한 클록 생성 회로에서도 이 게이트 방식 클록 변환을 자동으로 수행할 수 있으며 그림 3은 신플리시티의 신플리파이 프리미어(Synplify Premier)에서 자동으로 생성된 클록 변환의 예를 나타낸 것이다.

IP 처리

ARM 프로세서와 같은 커다란 IP 블록은 하나의 완전한 칩으로 만들 수 있으며, 나머지 프로토타이핑 하드웨어와 상호 연결할 수 있다. 여기에 필요한 유일한 요구사항은 하드웨어가 ARM 코어타일과 같이 다양한 IP 기반 도터 카드를 수용할 수 있을 정도로 유연해야 한다는 것이다. 하지만 작고 공통적인 기능을 갖춘 메모리, FIFO, 멀티플라이어(Multipliers) 및 애더(Adders) 등을 이런 식으로 처리하기에는 그 수가 너무 많다. 다시 강조하지만 알맞은 EDA 소프트웨어는 이런 기능을 FPGA 호환 구현으로 자동 변환하거나 외부 구현을 위해 이들을 쉽게 격리할 수 있는 방법을 제공할 수 있다.

FPGA 보드 관련 작업

설계가 FPGA로 포팅되고 실제 동작속도로 작동되더라도 여기에서 디버그 작업을 어떻게 효과적으로 수행할 수 있을 것인가라는 또 다른 문제가 발생한다. 이때 로직 분석기(내장/외장형 여부 불문)로 도움을 받을 수 있지만 일반적으로 게이트 레벨의 네임스페이스를 이용해야 한다. 따라서 RTL(즉, 설계 단계에 처음으로 진입한 곳)까지 이 신호를 역추적하는 작업은 결합 단계 전체를 역순으로 작업해야 하기 때문에 많은 시간을 소모하는 과정이 될 수 있다. 따라서 게이트와 RTL 네임스페이스 사이의 기호 연결을 완벽하게 유지하는 도구를 사용하여 디버그 작업을 하는 것이 좋다.

신플리시티는 일반적인 오류나 어써션(assertion) 오류, 소프트웨어 디버거 트리거와 같은 외부 입력에 의해 발생하는 모든 경우의 수에 대해 완벽한 테스트케이스(Testcase)를 프로토타입에서 자동으로 추출하는 '토탈리콜 풀 비저빌리티 테크놀로지(TotalRecall Full Visibility Technology)'라는 신기술을 개발했다.

그런 다음 추출된 테스트케이스는 다른 사용자들을 위해서 프로토타입 하드웨어와 상관없이 일반적인 RTL 시뮬레이터에서 언제든지 재현되고 분석될 수 있다. 또 다른 사용자를 위해 프로토타입 하드웨어를 확보(freeing up) 할 수 있는 데, 이 기능은 FPGA 프로토타입을 더 넓은 검증 환경으로 확장시켜준다. 이 기술은 신플리시티의 '아이덴티파이 프로(Identify Pro)' 제품에 적용되어 있다.

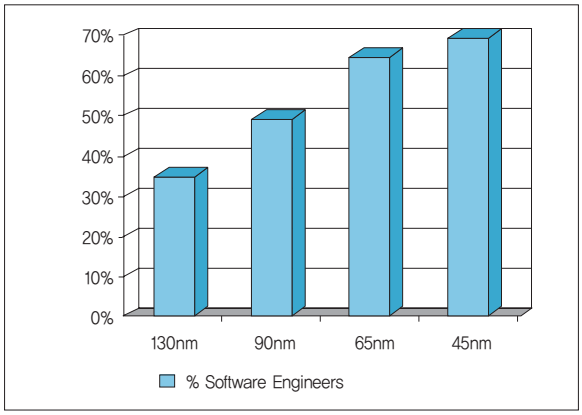

그림 2. 전체적인 엔지니어링 노력 'v' SoC 기술 노드의 비율 증가

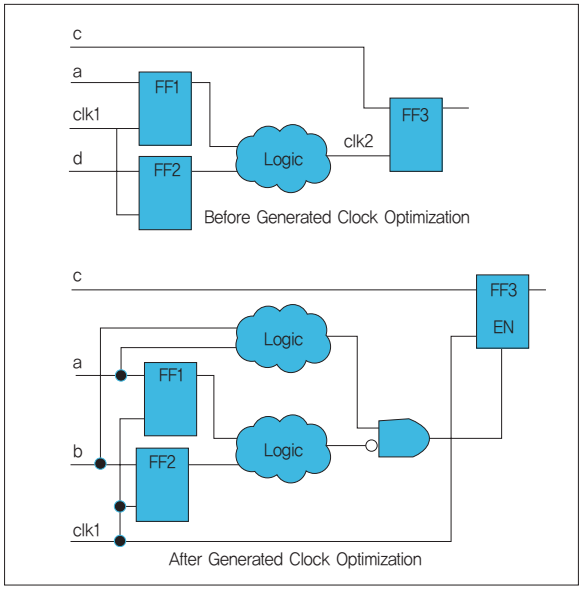

그림 3. 자동 게이트 방식 클록 변환

차세대 FPGA 프로토타이핑은?

실제 동작속도 기반의 FPGA 프로토타입은 복잡한 임베디드 시스템과 SoC의 검증을 위해 필요한 강력한 솔루션이다. 앞에서 설명한 것과 같이 EDA 도구들은 임베디드 시스템과 SoC의 검증과 관련된 오랜 문제들을 극복해왔으며, 기성품 형태의 프로토타이핑 보드로 사용할 수 있다. 그렇지만 프로토타이핑 하드웨어 및 도구 구성 요소를 더욱 완전하게 통합하기 위해서는 여전히 새로운 개선책이 필요한 상황이다. 이와 관련해 신플리시티는 새로운 프로토타입 출시 시간을 단축하는 동시에 프로토타입 제작과 수정 반복에 필요한 시간을 줄이기 위해 노력할 것이다. 이런 기술의 발달에 힘입어 신플리시티의 컨퍼마(Confirma) 플랫폼과 같은 실제 동작속도 검증 시스템은 모든 SoC 개발의 필수적 요소로 기존의 여타 검증 방법론과 함께 폭넓게 사용될 것이다. EP&C

▶ ASIC, SoC 또는 ASSP라는 용어는 이 글의 문맥상 서로 바꿔 쓸 수 있는 용어이다.