Introduction to Development and V&V of FPGA-based Digital I&Cs

Dependable Software Laboratory

Date

2015-09-15

김의섭

Table of Contents

1 FPGA

1.1 소개

프로그램이 가능한 비 메모리 반도체의 일종. 회로 변경이 불가능한 일반 반도체와 달리 용도에 맞게 회로를 다시 새겨 넣을 수 있다. 따라서 사용자는 자신의 용도에 맞게 반도체의 기능을 소프트웨어 프로그램 하듯이 변형시킬 수 있다. 일반 반도체에 비해 가격이 수십~수백 배 비싸며 항공, 자동차, 통신 등의 분야에 주로 쓰인다. [1]

FPGA는 Field Programmable Gate Array 의 약자로 Field Program 이란 반도체 제조 업체(예를 들어 국내의 경우 삼성, 현대 등과 같은 반도체 업체)를 통하지 않고도 산업 현장에서 엔지니어가 직접 디바이스를 프로그래밍하여 설계한 회로를 반도체 칩상에 구현할 수 있다는 것을 의미한다. Gate Array는 원래 ASIC(Application Specific Integrated Circuits) 디바이스 종류의 하나로 FPGA의 내부 구조가 ASIC의 한 종류인 Gate Array와 유사하기 때문에 붙여진 것 이다. [2]

FPGA의 Field Programmable Gate Arrary의 약자로 일반 사용자가 프로그램 가능한 Gate Array라고 할 수 있다. 여기서 Field는 ASIC 공정을 통하여 원하는 회로를 만드는 것이 아니라 일반 field에서 일반 유저가 원하는 로직을 구현할 수 있다는 의미로 사용된 것으로 볼 수 있다. 그리고 여기서 사용되는 Gate Array라는 용어는 Standard Cell 공정시간을 단축시키기 위한 아이디어로 나온 것이 Gate Arrary 인데 FPGA라는 용어를 만든 사람들이 Field에서 프로그램 가능한 Gate Array라는 의미로, FPGA라는 용어를 만든 것으로 볼 수 있다. 현재 ASIC의 최대단점인 너무 긴 공정기간과 너무 높은 FAB비용으로 등으로 인하여, FPGA의 중요성은 점점 커지고 있는데 요즘 들어서는 FPGA 단가도 낮아지고 있으면 내장가능한 로직의 크기도 상당히 커지고 있다. 그래서 기존의 ASIC 영역의 중에서 수량이 아주 많지 않으면서 단가에 덜 민감한 시스템에 많은 FPGA가 채용되고 있다. 또한 ASIC의 만들기 전에, 일단은 FPGA로 회로를 구현하여 테스트를 하는 목적으로도 대부분 FPGA를 사용하고 있으므로, ASIC엔지니어에게 있어서도 FPGA에 대한 이해는 중요한 문제라고 할 수 있다. [3]

1.1 FPGA 개발 흐름 [4]

PLD는 지난 1970년대 중반 MMI라는 회사가 처음 개발한 PAL제품이 그 시초라 할 수 있다. 바이폴라 테크놀로지를 이용한 PAL은 OTP (One Time Programmable) 타입이기는 하지만 반도체 역사상 최초로 프로그램 가능한 아키텍처를 구현했다는 점에서 지금의 PLD 제품의 모체라 볼 수 있다.

이후 1983년에 알테라가 EPROM을 기반으로 한 EPLD (Erasable Programmable Logic Device) 를 개발함으로써 자외선과 전기적 신호를 이용해 쓰고, 지우기가 가능한 프로그래머블 디바이스인 Classic이라는 제품을 선보였는데 이를 통해 지금의 CPLD (Complex Programmable Logic Device) 시장이 본격화 되었다. 당시 반도체 칩 위에 유리창이 달려있던 제품들이 바로 EPROM을 사용해 프로그래밍을 했던 PLD 디바이스들이다.

AMD로 인수되었던 MMI의 기존 멤버들이 벤처로 창립한 래티스도 CPLD시장의 선도주자 중 하나다. CMOS 테크놀로지를 이용한 EEPROM 기반의 SPLD (Simple PLD) 인 GAL을 처음 개발했고 현재 High Density CPLD 시장에서 빠르게 성장하고 있다.

게이트 어레이 아키텍처에서 출발한 FPGA (Field Programmable Gate Array) 는 1985년, 자일링스가 XC2064라는 제품을 통해 처음으로 선보인 개념이다.

EEPROM 기반의 CPLD와는 달리 SRAM 아키텍처를 채택하고 있는 FPGA는 CPLD에 비해 속도와 성능면에서는 저조했지만 대용량을 구현하는데 있어 유리하다는 장점 때문에 ASIC 프로토타입으로 각광을 받았다. 최근에는 FPGA 기술이 발전하면서 성능면에서도 CPLD에 근접하고 있는데다 가격면에서도 크게 향상돼 마켓영역이 꾸준히 확대되고 있다.

현재 FPGA는 이러한 전통적인 SRAM 기반 제품들 이외에도 앤티퓨즈 (Antifuse) 방식의 FPGA를 비롯해 플래시 타입 제품들도 공급되고 있다.

액텔과 퀵로직이 공급하고 있는 앤티퓨즈 방식의 FPGA는 OTP 타입으로 프로그래밍을 한번밖에 할 수 없고 프로세스 공정상 대용량 구현이 어렵다는 단점을 가지고 있으나 라우팅 리소스가 풍부하고 스피드면에서 매우 탁월한 성능을 제공한다. 또한 보안성이 탁월해 군사용 어플리케이션 분야나 게임기, 가라오케 같은 니치 마켓에서 좋은 반응을 얻고 있다.

플래시타입 FPGA는 초기 시장진입 단계로 아직 사용자가 많지는 않지만 SRAM 타입 FPGA와는 달리 별도의 PROM을 쓰지 않고도 라우팅이 가능해 원 칩 솔루션을 구현할 수 있고 스위치 사이즈가 12분의 1 수준에 불과해 다이 사이즈를 줄이고 전력소모를 절감할 수 있다는 장점을 가지고 있다. 플래시 타입 FPGA는 GateField가 처음 개발한 제품으로 현재 액텔이 공급하고 있다.

1.2 구조

FPGA 는 프로그램 가능한 논리소자의 배열의 한 종류이다. 밭, 논 같은 경작지를 생각해보면 넓은 평야(field)의 경우 바둑판처럼 규칙적인 구획을 가진 배열이다. 가로 세로로 나누는 것은 사람들이 걸어 다닐 수 있는 "둑"과 같은 것, 즉 FPGA는 하나의 밭의 구역과 같은 로직블럭이 넓은 평야에 있는 밭처럼 규칙적, 반복적으로 배치되어 있으며, 그 각각의 로직블럭을 둑의 길과 같이 가로 및 세로방향으로 연결선이 배치된 구조로, 외부에서 프로그램하여 로직블럭의 동작과 로직블럭간의 연결선을 임의로 배치하여 다양한 동작을 할 수 있도록 설계된 IC라 할 수 있다

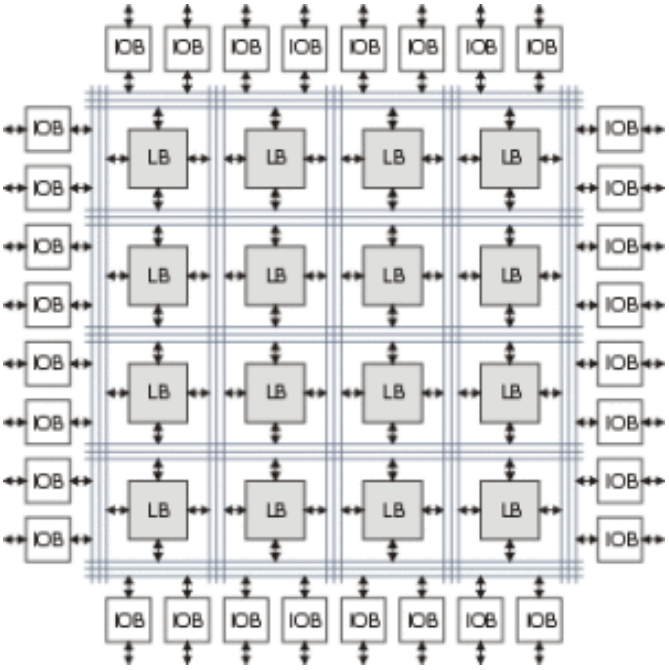


그림 1. FPGA 도식도

일반적인 FPGA의 논리 블록은 아래에 보이는 것처럼 4개의 입력 룩업 테이블 (lookup table)과 플립플롭으로 구성된다. 레지스터나 언레지스터 룩업 테이블이 가능한 하나의 출력만 있다. 논리블록에는 룩업 테이블을 위한 4개의 입력과 클럭 입력이 있다.

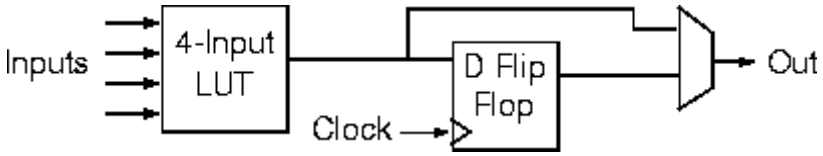


그림 2. 로직(논리)블록 [5]

- LUT(Look Up Table) [6]

FPGA는 Gate Array와 달리 NAND/NOR Gate는 사용하지 않고, 대신 4~6 input LUT를 사용하여 combinational logic을 구성합니다. LUT이란 그냥 memory라고 생각하면 됩니다. Address에 input을 가하면 memory 내부의 내용이 output으로 나오면서 그것이 logic이 됩니다. memory 내부의 내용을 바꾸어 다양한 combinational logic을 구현할 수 있습니다. 당연히 sequential logic을 설계할 수 있어야 하므로 LUT의 끝에는 flip-flop이 달려 있어서 이용할 수 있습니다. 기본적으로 LUT를 여러 개 넣은 것을 한 단위로 하여 Vendor별로 Slice니 ALM이니 하는 것으로 부르고 그것의 배열이 FPGA를 이루고 있습니다.

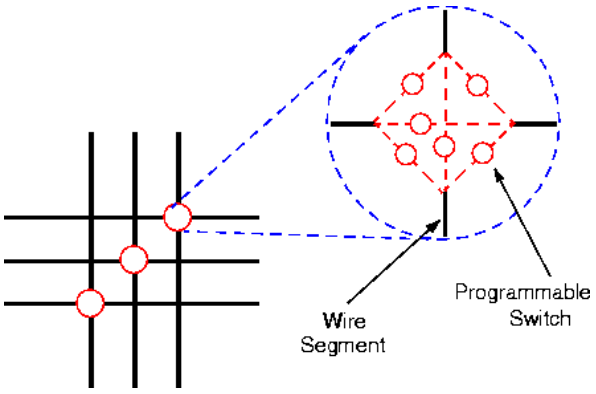


그림 3. 스위치 상자 [5]

- Programmable Interconnect [6]

LUT들 사이의 연결을 해야 완전한 logic이 됩니다. 이 연결도 당연히 programmable합니다. 일반적으로 FPGA 설계가 chip으로 구현하는 것보다 느린 이유는 이 Interconnect 부분에서 많은 delay가 있기 때문입니다. 될 수 있으면 routing을 잘 하여 interconnect의 delay를 줄여야 빨리 동작하는 설계가 됩니다.

1.3 FPGA 장단점

간편하게 설계한 로직을 반복적으로 이식할 수 있다

Time to market이 좋다

- 빠르게 시장에 내다 팔 수 있다. (ASIC 대비)
- ASIC은 한번 만드는데 대략 3~6개월 걸림 (설계상에 오류가 있다면 그만큼의 시간이 필요)
- Non-recurring Engineering(NRE) charge를 내지 않아도 된다는 것이다.
- 이름 그대로 NRE charge는 ASIC을 의뢰하여 샘플 받는데 까지 무조건 내야 하는(공정에

따라 한번에 수천 만원에서 수억 원에 이름) 비용이고 칩이 동작 안 한다고 하여 돌려받을 수 있는 돈이 아니다.

업데이트 가능

- FPGA는 SRAM타입의 경우 PROM파일만 바꿔주면 안에 내용을 바꿀 수 있음
- 예를 들어 해외에 수출했는데 오류가 발생 할 경우, ASIC은 전수 회수하여 칩을 교체해야 함 반면, FPGA는 SW업데이트로 해결이 가능합니다.

고비용

- 대량으로 생산하는 경우 (ASIC의 양산단가는 FPGA와 비교 안될 정도로 쌈)

Size문제

- FPGA에 비해 ASIC은 칩의 면적이 작음 (충분히 optimize 되기 때문)
- 핸드폰 처럼 작은 사이즈가 필요한 제품은 FPGA로 설계하기 어려움 (발열이나 사이즈 문제)

2 FPGA Development Process

FPGA-based system는 비록 최종 결과물이 hardware component 이지만 design과 implementation의 형태는 software-based system과 매우 유사하다. [7]

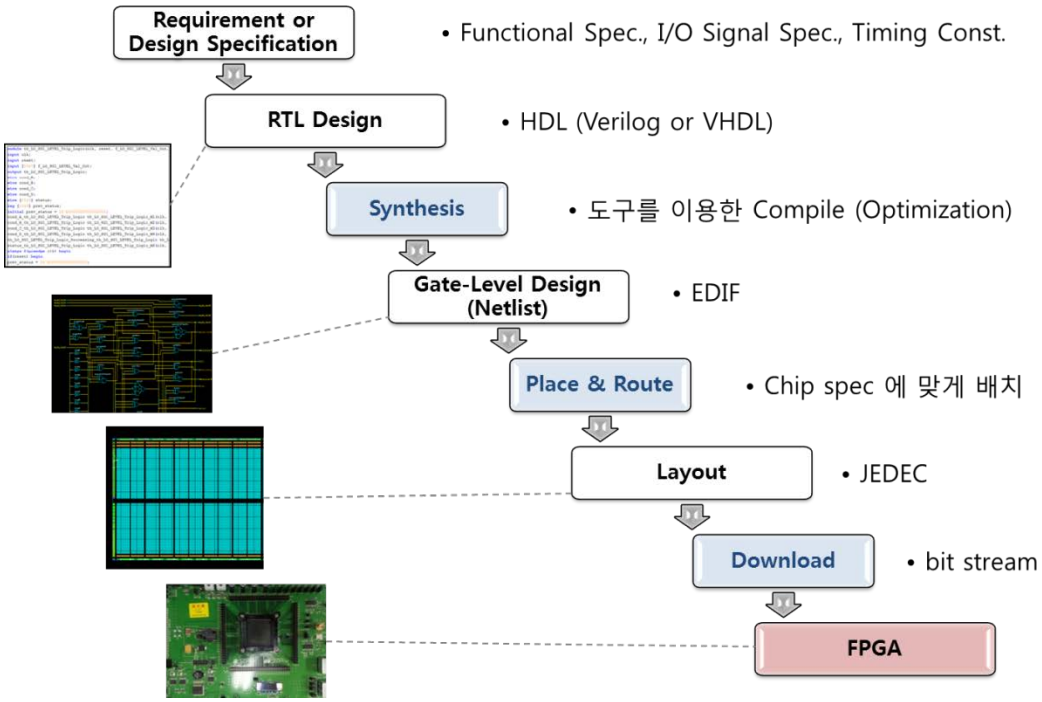


그림 4. FPGA Development Process

FPGA design level에서 사용되는 많은 수의 HDL(Hardware Description Language)와 higher-level 언어는 일반적인 software programming 언어와 매우 유사하며, 이는 자동화된 design tools을 이

용하여 자동으로 쉽게 얻을 수 있다. 또한, FPGA로 configuration 되기 전의 생성물 역시 SW와 비슷한 binary configuration code이다. 많은 FPGA vendor들은 이를 위한 IDE(integrated development environment)를 가지고 있으며 개발 process의 다양한 변환 V&V 활동을 지원하고 있다. 일반적인 FPGA 의 개발 프로세스 그림 4와 같다.

2.1 Requirement Specification or Design Specification

FPGA 개발을 위한 첫 단계는 요구사항을 분석하여 정의하는 단계이다. 해당 단계에서는 기능적인 부분뿐만 아니라 물리적인 하드웨어 관련 부분 역시 분석되어야 한다. 기능적인 부분은 시스템이 어떤 동작을 할지를 결정하는 단계이고 하드웨어 관련 부분은 기능을 수행하기 위해 어떤 chip을 선택할지, 외부 device와 어떻게 data를 주고 받을지를 결정해야 한다. 분석한 기능을 자연어 또는 block diagram, schematic으로 형상화할 수 있다.

또한 개발자는 어떤 FPGA chip을 사용할지 결정해야 한다. 선택된 FPGA의 물리적 특징 및 성능이 가 design에 영향을 주게 되기 때문이다. 예를 들면 전체 chip의 동작 clock 속도나 memory bandwidth 등에 따라 디자인을 고려해야 한다. Critical timing 이나 power, board size 역시 고려되어야 한다. 또한 chip이 담당할 부분과 외부 device와 어떻게 연결할지를 정해야 한다. Chip과 외부 device가 어떻게 연결 되느냐에 따라 chip 내부의 interface가 달라지기도 하기 때문이다.

또한 해당 단계에서 어떻게 FPGA 를 개발할지, V&V 활동을 어떻게 진행 할지, 프로젝트 진행은 어떻게 할지 등 계획한다. 잘 정립된 개발 프로세스와 V&V 계획은 FPGA의 Quality에 영향을 주게 될 것이다. 개빌 된 FPGA가 Safety critical system에 사용될 경우 해당 내용은 safety에 영향을 주게 될 것이다. 또한 요구사항의 trace는 어떻게 할지, 개발 중에 발생한 issue 는 어떻게 관리할지, 형상관리는 어떻게 할지 등을 모두 결정해야 한다.

2.2 RTL Design

RTL design 이란 디지털회로를 설계하는 방식중의 하나이다. 하드웨어 레지스터와 산술논리연산(조합)회로 간의 데이터흐름으로 정의된다. 공정에 독립적인 모델(technology-independent model)이며, 논리합성도구에 의해 게이트레벨 네트리스트로 추출된다. 특정 공정 셀라이브러리(cell library)와 설계제약조건(design constraint)이 필요하며, 주로 VHDL, Verilog-HDL이 사용된다. [8]

요구사항 분석 단계에서 분석된 기능은 HDL(Hardware Description Language)라고 하는 언어로 programming을 하게 된다. 현재 가장 많이 사용되는 HDL은 Verilog 와 VHDL이다. HDL은

software를 프로그래밍하는 언어와 매우 유사하다. 오랜 시간 동안 포트란, 파스칼, C 등의 프로그래밍 언어들이 SW의 기능을 기술하는데 사용되어 왔고 비슷하게, 디지털 설계 영역에서도 설계자들은 디지털 회로를 표현하기 위한 표준 언어의 필요성을 깨달게 되었고, 이런 이유로 하드웨어 표현 언어인 HDL이 등장하게 되었다. HDL은 개발자가 하드웨어 요소들에서 발생하는 반복성을 모델링 할수록 해준다.

개발자는 HDL을 이용하여 기능 구현에만 초점을 맞추어 개발을 진행 할 수 있다. FPGA의 물리적인 component나 connection은 HDL 프로그래밍에서는 고려하지 않아도 된다. 차후 synthesis 도구나 P&R 도구가 개발자가 프로그래밍한 기능적인 부분과 물리적인 부분을 자동으로 연결 해주기 때문이다.

또한 해당 단계에서 high-level 언어를 사용할 수도 있다. 하지만 해당 언어를 FPGA 와 같은 hardware를 개발하는데 사용하기 위해서는 명확하고 쉽게 이해될 수 있도록 strict한 rules이 잘 정의된 언어를 사용해야 한다. 예를 들면 MISRA C (Motor Industry Software Reliability Association) 또는 ESA (European Space Agency)의 사용이 가능하다.

2.2.1 Verilog

Verilog는 디지털 회로를 다루고 Verilog에서 모듈은 디지털 회로의 하나의 컴포넌트를 의미한다. 이러한 컴포넌트는 아주 간단하게는 게이트가 될 수 있고 ALU나 메모리와 같은 복잡한 회로가 될 수 도 있다. 모듈은 모듈 내에 필요한 요소들을 가지고 있다는 점에서 그리고 여러 개의 메소드로 외부와 인터페이싱된다는 점에서 C++의 클래스 개념과 유사하다. C++의 클래스처럼 모듈 역시 인스턴스화 시킬 수 있지만, 모듈은 100% 클래스와 같지는 않다. 모듈을 그래픽하게 표현한다면 여러 개의 포트를 가지고 있는 하나의 박스로 표현할 수 있다. 포트는 입력, 출력 혹은 입출력 둘다가 될 수 있으며, 포트는 한 개 비트 너비를 가질 수 있고 또 여러 개의 비트 너비를 가질 수도 있다. 아래의 그림은 여러 개의 입출력을 가진 모듈을 그린 그림이다. 입력, 출력의 숫자와 그것들의 너비와 방향은 모듈 기능에 의해 전적으로 종속된다. 포트는 입력이나 출력 혹은 양방향이 될 수 있다. Verilog의 기능을 간단히 말한다면 모듈들을 생성하고, 모듈을 서로 연결시키고, 상호동작의 타이밍을 관리하는 것이라고 볼 수 있습니다. [9]

그림 5. Verilog 모듈

2.2.1.1 Verilog 특징 [10]

- 전자 회로 및 시스템에 쓰이는 하드웨어 기술 언어이다. 당연하지만 베릴로그 HDL이라고도 부른다. 회로 설계, 검증, 구현 등 여러 용도로 사용된다.
- C언어와 비슷한 문법을 가진 것이 특징이다. 'if'나 'for', 'while' 같은 제어 구조도 동일하며, 출력 루틴 및 연산자들도 거의 비슷한 것 등 사용자들이 쉽게 접근할 수 있다. 다만, 블록의 시작과 끝을 중괄호 대신 Begin과 End를 사용하여 구분하고, HDL의 특징인 시간에 대한 개념이 포함되었다는 것 등이 다른 점.
- 좀 더 고수준 언어(high level language)로서 System-verilog 라는 언어가 있다. 설계자 성향에 따라 다르지만 대부분의 경우 설계 검증용으로 사용한다. 상당히 C언어와 흡사한 형태를 가지며 새로운 데이터 타입도 여럿 추가되었다. 위에 언급된 begin end 이외에도 fork join도 추가되었다.
- 본래는 Phil Morby가 모의시험용 언어로 개발하였다. 1980년대 말까지, 사실상의 표준 HDL으로써 독점소유였지만, 후에 IEEE 표준이 되어버렸다.
- VHDL과 함께 대표적인 하드웨어 기술 언어.
- 대부분의 엔지니어는 VHDL이 더 쉽고 직관적이라고 평가하고 있으나 대한민국 업계에선 사실상 verilog가 표준이다.
- 1983년 Gateway Design Automation 사에서 하드웨어 기술언어인 HiLo와 C언어의 특징을 기반으로 개발
- EEE 1364로 표준화된 Verilog는 전자 회로 및 시스템에 사용되는 하드웨어 기술 언어로, 회로 설계, 검증, 구현 등 여러 용도로 사용
- C 언어와 비슷한 문법
- if나 while, for loop 문 과 같은 제어 구조 / 연산자 : 논리 (&<^,|), 산술 (+,-,*,/)

- C 언어와 달리, HDL의 특징인 시간에 대한 개념이 포함

2.2.2 VHDL

IC기술이 발달로 인해 더 많은 소자를 단일 칩에 집적화함에 따라, 디지털시스템은 더욱 복잡해지고 있다. 이렇게 복잡해진 디지털시스템을 게이트나 플립플롭 레벨 설계는 많은 시간이 소비되고 힘들다. 그러므로 시스템 설계에서 하드웨어 기술언어인 HDL(hardware description language)사용의 중요성이 날로 높아지고 있다. VHDL은 디지털시스템의 동작과 구조를 기술하기 위하여 사용하는 하드웨어 기술언어이다. VHDL는 VHSIC Hardware Description Language의 약어이며, VHSIC는 Very High Speed Integrated Circuit를 의미한다. VHDL은 상위의 동작 레벨에서부터 하위의 게이트 레벨까지 하드웨어를 기술하고 설계하는 IEEE 표준언어인 하드웨어 설계 언어이다. VHDL은 미국 국방성에서 직접회로 관련 업체들 상호간에 개발을 위한 정보교환 및 문서화를 위해서 표준화된 하드웨어 기술언어의 필요성을 인식하여 초고속 집적회로프로그램의 일부로서 만들어졌다. 이때 당시 VHDL은 하드웨어를 설계하는 언어이기보다는 설계자의 생각을 표현 및 시뮬레이션 하는 정도의 목적이 더 컸다. 이후 여러차례 표준화 작업등을 거쳐 1987년 12월에 IEEE-1076으로 표준화가 되었다. 이로인해 VHDL을 이용하여 회로를 설계할 수 있는 여러 CAD tool들이 개발되기 시작하였고 또한 많은 디지털 회로 설계자들이 VHDL을 이용하여 회로를 설계하기 시작하였다.

현재는 VHDL을 실제회로로 만들어 주는 합성툴(synthesis tool)의 발달로 인해 설계자의 생각을 VHDL로 표현하면 합성툴이 실제 회로로 만들어 주고있다. 설계시간이 단축되며 그외에도 여러 장점들이 있다. [11]

2.2.3 VHDL 특징 [12]

- 전자 회로 및 시스템에 쓰이는 하드웨어 기술 언어이다. 당연하지만 베릴로그 HDL이라고도 부른다. 회로
- VHDL(VHSIC Hardware Description Languae)은 디지털 회로의 설계 자동화에 사용하는 하드웨어 기술 언어(hardware description language ,HDL), 즉 회로 설계 언어이다. 엄밀히 말하자면 프로그래밍 언어는 아닌데 어차피 요즘 차이가 점점 모호해지는 관계로 그냥 프로그래밍 언어 항목에 끼워넣은 듯 하다(?).
- 본래 개발목적은 미국 국방부에서 주문형 집적회로(ASIC = Application Specific Integrated Circuit)의 문서화에 사용하기 위하여 만든 언어이다. 복잡한 매뉴얼로 사용자의 뇌를 아프게 하는 대신, 회로의 동작 내용을 문서화하여 설명하기 위해 개발된 것. 그런데 언제부터인가 이런 문서를 회로 디자인 과정에서의 시뮬레이션에 사용하게 되었고, 여기에 재미들려서 VHDL 파일을 읽어들여서 논리 합성 후에 실제 회로 형태를 출력하는 기능을

덧붙이게 되었다. 결국, 오늘날에는 디지털 회로의 설계, 검증, 구현 등의 모든 용도로 사용하는 중.

- 기본적인 베이스는 본래 Ada의 부분집합에 시간 개념(디지털 회로에 필수적인)을 추가하는 방식이었으나, IEEE 표준화 작업을 거치면서 오늘날과 같은 형태와 문법을 가지게 되었다.
- Verilog와 함께 대표적인 하드웨어 기술 언어이며, 최근은 대부분의 회사는 고유 포맷을 이용하기보다는 이 두가지 표준 HDL을 사용하는 추세.

2.3 Synthesis

합성이란 RTL로 구현된 추상적인 회로(Verilog 또는 VHDL로 구현)를 실제 gate(또는 standard cell)들을 통해 구현하는 과정이다. 여기서 말하는 게이트레벨이란 CMOS회로의 NAND, AND, INVERTER, Flip-Flop 등의 각종 게이트로 구성되는 것을 말하며, 게이트의 종류는 물리적으로 레이아웃된 게이트의 라이브러리를 몇 종류를 갖고 사용하느냐에 의존한다. 예로 NAND 게이트 경우, 게이트의 입력의 수가 2,3,4,5,8 개의 입력을 갖는 게이트 각각 라이브러리로 준비되어 있어 합성단계에서 이용이 가능하다. 설계 과정에는 셀 라이브러리와 공정에 관한 자료를 반도체 회사로부터 제공받아 이용한다.

원래는 사용자가 직접 gate level의 Netlist를 작성해야 하지만 제품의 사이즈와 복잡도가 증가로 인해 직접 작성이 불가능 해졌다. 합성도구는 사용자는 사용자 친화적 언어인 HDL로 제품을 설계하면 이를 지동으로 Netlist 형태로 변환 해주게 된다. 사용자가 고려 해야 하는 복잡한 과정(Optimization)을 대신해서 수행해주며, 칩의 speed, cost, power 측면에서의 최적화를 자동으로 수행해준다. 따라서, 사용자는 위와 같은 복잡한 고려 없이 디자인 및 설계에 집중하여 칩 개발이 가능해졌다고 할 수 있다.

```
module Example_1 (clk, in, out)

    input clk;
    input [3:0] in;
    output out;
    reg out;

    parameter a = 4 ;
    parameter b = 1 ;

    initial begin
        out = 0;
    end

    always @(negedge clk)
    begin
        if (in > a + b)
            out = 1;
        else
            out = 0;
    end
endmodule
```

Synthesis →

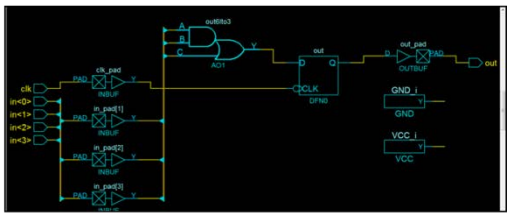


그림 6. 합성

합성이 해주는 첫번째 일은 바로 Verilog를 이용하여 구현한 기능을 게이트들을 이용하여 구현해 주는 것이다.

합성이 해주는 두번째 일은 이 디지털(로직)연산을 이용하여 구현된 게이트와 블럭들을 Standard Cell에 미리 선언되어있는 각각의 Cell들과 매칭을 시켜주는 것이다. 대부분의 디지털연산에서는 AND, OR, AOI(AND+OR+Inverter) 등의 combinational logic과 D Flip Flop등과 같은 기본적인 Gate들의 조합으로 모든 기능을 구현할 수 있기 때문에, 반복적으로 사용되는 이런 Cell들을 일일이 구현하지 않도록 Fab (회로를 실제 반도체를 통해 구현해주는 회사 혹은 기관) 에서는 디지털 라이브러리, 즉 Standard Cell을 제공한다.

합성이 해주는 세번째 일은 합성 시 선언한 조건들을 이용하여 구현 뿐 아니라 안전성이나 타이밍 문제, 로드 등 다양한 제한조건에 맞도록 구현한 회로를 수정해준다. 디지털회로시간에 배우는 Fan-out이나 Drive의 개념을 알고계신 분들은 이해가 편하시겠지만, 그렇지 않은 분들을 위해 아주 간단히만 설명하자면, 사이즈가 최소로 매우 작은 Inverter의 출력단이 1과 0을 왔다갔다할 때, 출력단에 1uF과 같이 엄청 큰 load 캐패시터를 동작시켜야 한다면 캐패시터의 큰 용량을 다 채우지 못한 채로 충방전하는데 많은 시간과 전하가 필요해 결국 신호가 사라져버린다. 입력이나 출력단에 얼마나 큰 캡이 연결될지, 입력부터 출력까지는 몇 주기 이내로 도착해야하는지, Setup Time이나 Hold Time이 잘 지켜져 data가 metastable한 값으로 빠지지는 않는지, 모두 합성 시 조건으로 선언해줍니다. 이 조건들을 고려하여 너무 빨리 transition이 일어나는 곳은 Delay Buffer Cell을 달고, 출력에서 큰 load를 drive를 해야한다면 사이즈가 매우 큰 MOS를 이용한 Flip-flop모델을 이용하는 등 다양한 조건에 맞는 회로를 구현해준다.

요약하자면, 로직합성이란 디지털회로를 구현하는 과정에서 반복적이고 사람이 할 경우 완벽하지 않을 수 있는 과정들을 Design Compiler와 같은 컴퓨터프로그램에 내장된 알고리즘과 최적화과정을 통해 구현하는 것입니다. 따라서 설계자는 Verilog로 구현한 디지털코드를 자신의 회로에 맞는 조건들을 선언하여 합성하면, Fab에서 제공하는 라이브러리에 선언되어 있는 Standard Cell들의 조합으로 회로를 구현해 주게 된다. [13]

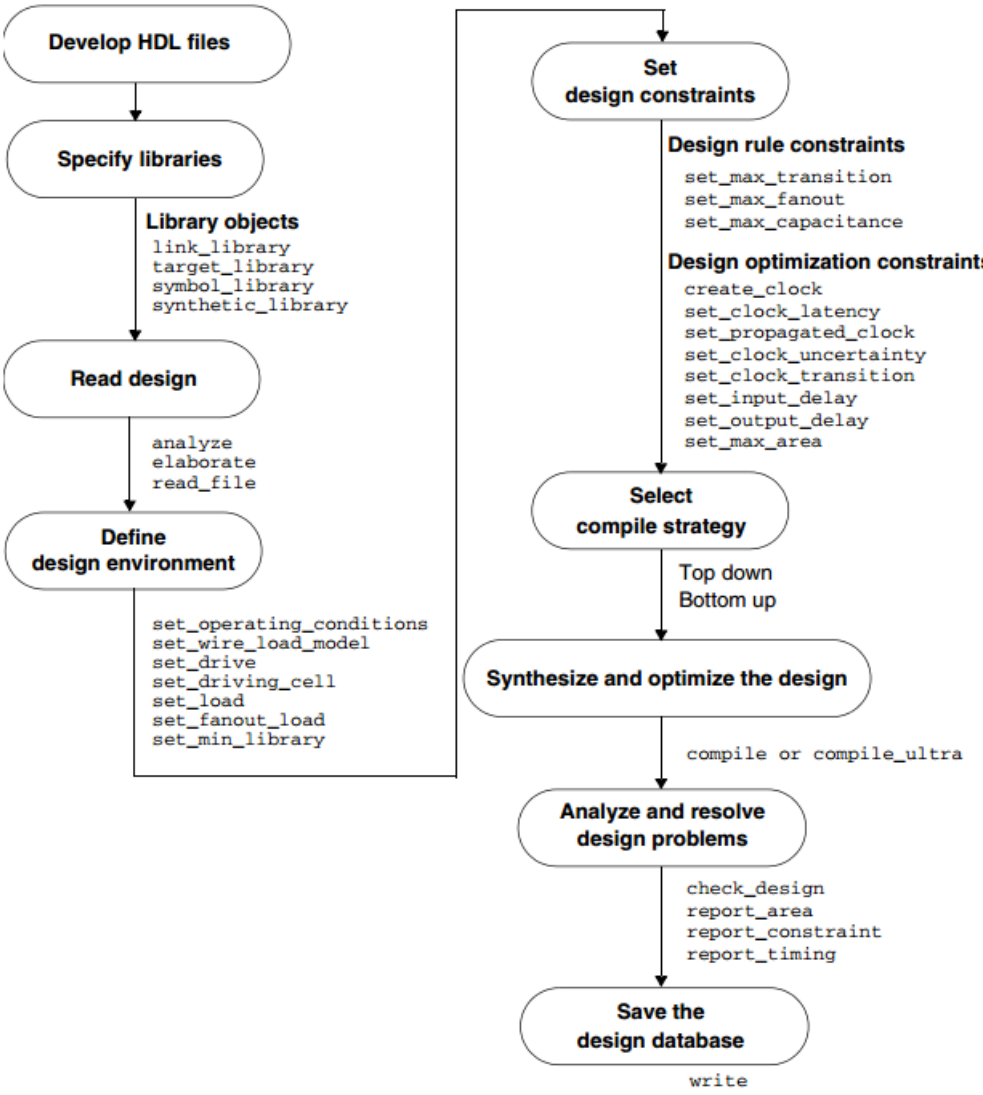


그림 7. Basic synthesis flow (Synopsys 社 - Design Compiler) [14]

2.3.1 셀 라이브러리 [15]

RTL 설계와 합성 단계 이후의 과정의 중요한 차이점은 각각 공정에 비의존적, 의존적이라는 것이다. Verilog 를 이용한 설계의 가장 큰 장점이 바로 공정에 비의존적이라는 것이다. 하지만, 이러한 특징은 RTL 수준까지만 적용된다. 합성을 위해서는 칩을 제작하고자 하는 공정의 정보가 필요하며 합성된 회로는 해당 공정에서만 사용할 수 있게 된다. 공정의 정보는 셀 라이브러리라고 불리는 형태를 갖고 있으며, 합성과 P&R 을 하기 위해서는 반드시 필요하다. 셀 라이브러리가 포함하고 있는 정보는 그림 2 에서 와 같이 합성에 사용될 셀(로직 게이트)에 대한 특성(속도, 크기, 등), 게이트 레벨 검증에 필요한 동작 특성(Verilog 모델), 그림 3 과 같이 P&R 에 사용될 각각의 셀의 레이아웃이다. 이러한 정보는 해당 공정을 개발한 회사의 기밀 자료이므로 실제 칩을 제작

할 설계자들에게만 제공되며, 외부에 유출하지 않겠다는 서약서 (NDA: Non Disclosure Agreement)를 작성해야 사용이 가능하다.

합성을 위해서는 각 foundry에서 제공하는 synthesis를 위한 library가 필요하며 이 library 에는 각 logic gate 의 timing, function, power 등의 정보를 포함하고 있다. 설계자는 이런 정보를 이용하여 시스템 Spec에 적절한 clock속도, 면적을 얻기 위한 합성의 과정을 수행한다. 합성된 logic 은 gate level 의 Netlist로 출력하여 SDF(Standard Delay Format) file과 함께 prelayout simulation(timing simulation)에서 활용된다.

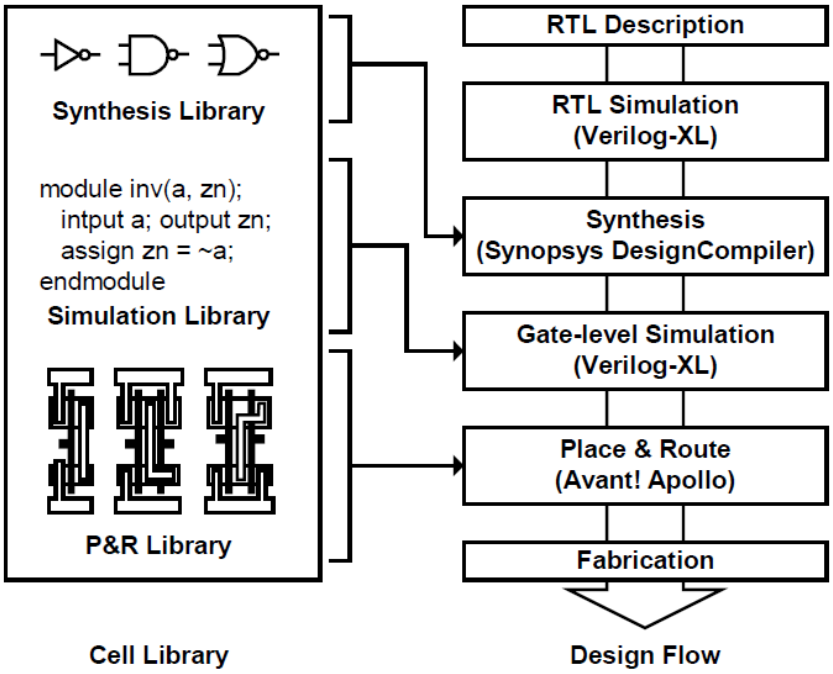


그림 8. 셀 라이브러리

2.3.2 합성 가능한(synthesizable) 설계 [15]

Verilog는 HDL이지만 Verilog로 기술한 모든 설계가 실제 회로로 구현이 가능한 것은 아니다. 그 이유는 다음과 같이 정리할 수 있다.

- Verilog 언어에서 제공하는 기능 중 회로(또는 시스템)의 모델링을 위한 기능은 구현이 불가능 하다.
  - 예) 지연시간 (a=#10b), 트랜지스터 수준의 기술, 메모리 등

- 합성 툴이 지원하지 않는 경우
  - 예) 나눗셈 및 %(modulo) 연산은 피연산자가 모두 상수인 경우만 합성 가능
- 하드웨어로 구현은 가능하지만, 제한된 면적을 초과하거나 목표성능(동작 주파수)에 미치지 못하는 경우

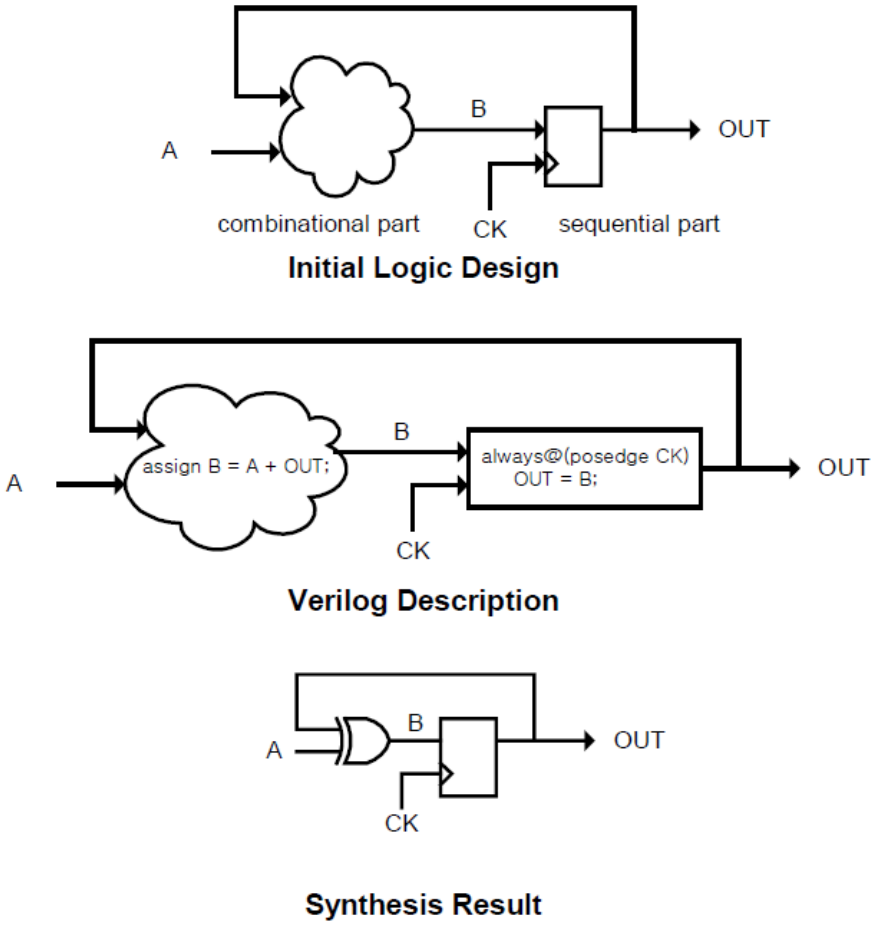


그림 9. 합성 가능한 설계방법

Verilog 는 실제회로로 구현하는 것 이외에도 모델링에도 사용되므로 매우 많은 기능을 지원한다. 이 기 능들은 Verilog-XL 시뮬레이터에서는 올바르게 동작하지만, 합성이 불가능하거나 합성 결과가 RTL 과 다른 동작을 보이게 된다. Verilog 에 처음 입문한 사람들이 공통적으로 범하는 실수는 C 언어로 프로그램 하듯이 설계를 하는 것이며, 이런 경우 대부분 합성이 불가능하다.

성공적으로 합성을 하기 위해서는 마치 로직 게이트를 이용하여 회로를 그리듯이(schematic capture) Verilog 기술을 해야 한다. 즉, 회로를 미리 구상하고 Verilog 로 기술하는 것이다. 그렇다면, 합성 툴이 어떻게 합성할지를 미리 예상하여 Verilog 기술을 해야 하는데 이것은 어찌 보면 참 번거로운 작업이다. 차라리 schematic capture 방식(GUI 를 이용하여 회로를 제작하는)이 더

편한 방법일 수 있을 것이다. 하지만, 앞서 설명한 것처럼 Verilog 로 설계된 회로는 공정에 비의존적이라는 장점을 갖고 있으며, 합성 툴에서 다양 한 최적화를 수행할 수 있다.

아래 그림의 예와 같이 먼저 회로의 구조를 대략적으로 설계한다. 이때, 조합논리회로(combinational logic)과 순차회로(sequential logic)으로 구분한다. 다음 Verilog 를 이용하여 조합논리회로는 assign 문을, 순차회로는 always 블록으로 기술한다. 이를 합성하면 기본구조는 유지되면서 최적화된 회로를 얻게 된다.

2.3.3 Synthesis의 장단점

- 설계 사이클 단축
  - 설계 변경용이, 설계오류 가능성 저하, 간단한 기술 방식, 빠른 설계, 빠른 변경
- 설계의 질적 향상
  - 여러 가지 architecture 시도 가능: 최적화
  - 면적 또는 속도 최적화 가능(논리 합성기의 기능)
    - Timing analyzer 기능 포함(critical path delay, setup, hold time 계산)
- 판매자, 기술(technology)에 무관한 설계 가능
  - ASIC or FPGA 로 구현 가능
- 설계 비용 절감
  - 설계 기간 단축, 스키메틱 설계 과정 생략, 설계 오류 저하
  - Design reasability – IP, Library 그대로 또는 수정해서 사용가능
  - 한 engineer가 30 ~ 40 k 게이트의 설계 담당 기능
- 집적 기술에 따른 성장
  - 현재 수백만 게이트급 설계(SoC) 가능
  - 고성능 워크스테이션 도움 – 논리 합성, 시뮬레이션 필수
- 논리 합성 결과, 자동 생성된 회로 도면을 이해하기 어렵다.
  - 합성 도구 성능에 크게 좌우된다

2.3.4 Commercial tool for logic synthesis (3rd party synthesis tool)

다양한 합성도구들이 HDL code를 gate-level의 표현으로 변환해주기 위해 존재하고 있다. 이런 다양한 합성도구는 HDL code를 digital logic으로 최적화 해주며, EDIF file이나 graphical (schematic) 형태를 제공해준다. 대부분의 합성도구는 FPGA-independent한 schematic을 제공한다. 해당 schematic은 AND 나 OR, NOT 같은 gate와 flip-flops으로 구성되어 있다

- XST (delivered within ISE) by Xilinx
- Quartus II integrated Synthesis by Altera

- IspLever by Lattice Semiconductor
- Encounter RTL Compiler by Cadence Design Systems
- LeonardoSpectrum and Precision (RTL / Physical) by Mentor Graphics
- Synplify (PRO / Premier) by Synopsys
- BlastFPGA by Magma Design Automation

2.4 Gate-level Design

합성의 결과로 나오는 gate level netlist를 보통 pre (layout) netlist라고 부르고, Netlist의 형태는 Verilog 또는 EDIF등을 주로 사용합니다. 여기서 말하는 게이트레벨이란 CMOS회로의 NAND, AND, INVERTER, Flip-Flop 등의 각종 게이트로 구성되는 것을 말한다.

회로 설계자들이 원하는 기능을 만족하는 회로를 만들 때 사용하는 방법이 HDL 언어로 구현을 하거나 CAD 툴을 써서 직접 회로를 그리는 방법이다. 이때, HDL 언어를 사용하는 경우 보통 사람들이 구현하기 쉬운 방법으로 사용하기 때문에 실제 하드웨어에 적용해야 하는 경우(예를 들면 칩으로 구현하는 경우) 회로 합성(Synthesis) 과정을 거치게 된다. [16]

예를 들면 : c = a + b;

와 같은 표현은 하드웨어가 알기에는 너무 높은 수준의 언어이다. 따라서 이를 회로적으로 나타내기 위해서는 a와 b를 입력으로 하는 adder(덧셈기)를 하드웨어적으로 표현해야 한다. 결국 회로 합성툴을 사용하게 되면 위 예에서 보여주는 구문은 1비트 입력 2개를 사용하는 덧셈기(adder)로 구성하는 회로의 그림을 보여주게 된다.

이러한 과정을 회로 합성 과정이라 하고, 입력과 출력 그리고 이들이 사용하는 adder의 형태을 가리켜 netlist라고 한다. 여기서 synthesis 결과는 하나의 파일형태로 나오는데 이를 netlist 파일이라고 한다.

파일 내용에는 c = add_1(b,a);

와 같은 내용으로 표현되게 된다. 물론 CAD툴을 사용하는 경우에는 circuit 자체가 netlist 파일로 표현이 된다. 이렇게 만들어진 netlist 파일은 layout 과정을 통해 실제 칩으로 구현 가능한 형태의 circuit netlist를 도출하게 된다.

2.5 Place and Routing

P&R 과정은 후단계 설계(bacj-end)라고도 부르며 설계된 회로를 반도체 공정에 사용되는 레이아

웃으로 만드는 과정을 말한다. 또한, 논리합성을 통해 생성된 게이트 수준 netlist를 매스크 제작에 사용될 레이아웃 도면으로 변환하는 과정을 말한다. 칩 제작 공정에 맞게 미리 설계된 표준 셀 라이브러리(standard cell library), RAM, ROM 등의 매크로 셀, I/O 패드 셀과 설계자가 지정한 타이밍 조건을 적용하여 최적화된 레이아웃 도면을 생성하게 된다.

Mapping된 각 설계 블록들을 FPGA의 특정 논리 블록 위치에 지정하는 단계를 배치(Placement)라 한다. 배치된 각 블록들을 회로의 연결 정보에 따라 연결선 및 전기적 스위치를 사용하여 상호 연결하는 단계를 배선(Routing)이라고 한다. 배치와 배선은 그 결과가 상호 의존적인 작업이다. 좋은 배치는 좋은 배선을 보장한다. 좋은 배선이란 회로가 칩 내에서 100% 배선을 이루면서, 각 논리를 연결하는 와이어의 연결 경로가 가능하면 짧게 되어 신호의 지연 시간의 작게 되도록 하는 것이다. 이렇게 되기 위하여서는 연결도가 많은 회로들은 인접된 논리 블록에 배치되고 외부로 출력되는 신호를 가진 블록들은 입출력 셀과 가까운 쪽에 위치하게 되도록 배치가 일어날 것이다.

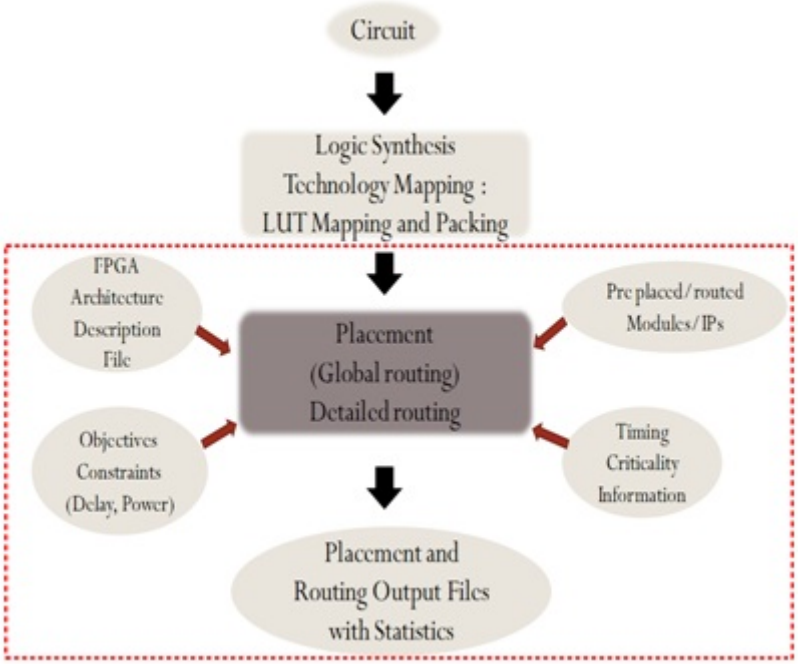


그림 10. Placement & Routing Flow

위 그림은 Placement & Routing Flow를 나타낸 것이다. Logic synthesis및technology mapping을 수행한 후 얻어진 netlist 파일과 FPGA architecture description file을 입력으로 받는다. Timing, criticality 정보와 pre placed/routed된 블록들을 고려하며, delay 및 power와 같은 제약조건을 만족시켜 placement and routing을 수행하게 된다.

3 Verification and Validation

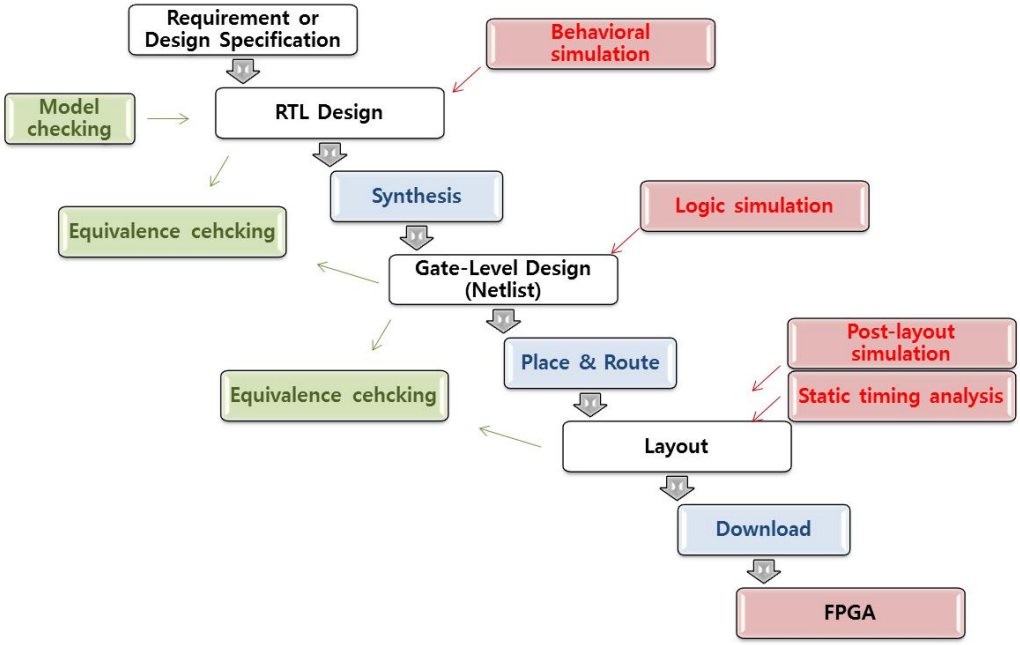
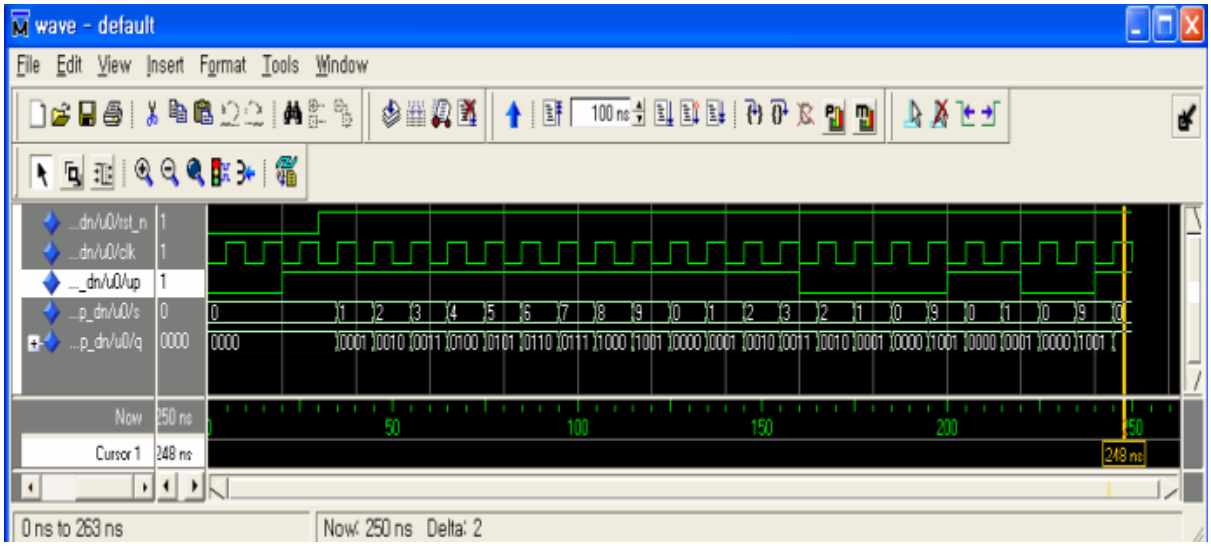


그림 11. FPGA V&V Activities

3.1 Simulation

설계한 논리 회로를 하드웨어를 이용해서 검증하는 경우에, 처리하는 데이터량이 많거나 속도가 빠를 경우에는 검증하는데 시간이 많이 걸리고 그 원인을 파악하지 못하는 경우가 있기 때문에 소프트웨어 적으로 먼저 검증하는 것이 필요하다. 시뮬레이션을 지원하는 도구들에서는 이를 위해 파형으로 검증하는 시뮬레이션을 제공하는데, 이 시뮬레이션에서 동작하는데 필요한 입력 파형의 조건을 설정해 주면 그에 따른 출력 결과의 파형이 출력되어 쉽게 확인해 볼 수 있다. 시뮬레이션을 하기 위해서는 설계 모듈뿐만 아니라 Test Bench를 작성 필요하다. Test Bench 란 설계한 모듈을 포함하여 테스트 입력을 인가하는 모듈이고, 매 clock cycle 마다 제대로 동작하는지 simulation vector라는 input data를 넣고 output data가 specification 대로 나오는지 확인하게 된다.



3.1.1.1 Simulation의 단점

- Simulation은 느리다
- 모든 경우의 수를 확인하는 것은 불가능
- 따라서 충분한 Coverage 또는 Corner Case 를 커버하는 Test bench의 작성이 중요
- 회로의 기능을 정확하게 검증할 수 있는 의미 있는 시뮬레이션 입력 벡터의 생성이 중요
- 특정 기능이나 특성을 목표로 하는 입력 벡터들은 설계자가 알고 있는 입력 공간의 영역으로 편향되기 때문에 설계자가 알지 못하는 영역에서 버그들이 종종 발생하게 됨

3.1.1.2 Behavior Simulation (Functional Simulation)

HDL로 기술된 code는 그 기능을 검증하기 위해 simulation이 필요하다. 이때 HDL simulator를 사용하게 된다. 통용되는 simulator 로는 Cadence 사의 NC-sim, Synopsys 사의 VCS-MX Mentor 사의 Model-sim 등이다. 예전에는 VHDL, Verilog HDL language를 각각의 simulator(Verilog -XL, VCS, VSS)를 사용하여 simulation 하였으나 현재는 Mixed language를 test 할 수 있는 simulator가 일반적으로 사용되고 있다. 나아가 Mixed signal(analog-digital 혼성모드)을 simulation 할 수 있는 CAD tool도 속속 출시되어 design verification methodology의 향상에 큰 기여를 하고 있다.

기능적 검사는 대부분의 다른 설계도 마찬가지겠지만 검증은 반도체 칩 설계에서 가장 중요한 부분이다. Software야 잘못되면 patch를 release한다거나 할 수 있지만 hardware가 잘못되면 chip을 다시 만들어야 하기 때문이다. 문제는 chip을 만드는 비용이 상당히 비싸기 때문에, 검증을 통해서 될 수 있으면 error가 없도록 만들어야 한다. [17]

검증 방법으로는 가장 중요한 것은 simulation이다. HDL은 hardware description 뿐 아니라

simulation을 위해서도 사용된다. 보통 test bench라고 불리는 HDL program을 작성하는데 쉽게 말하면 C 언어의 main 함수라고 생각하면 편하다. 매 clock cycle 제대로 동작하는지 살펴보기 위해서 simulation vector라고 해서 input data를 만들어서 넣어주고 output data가 specification 대로 나오는지 살펴보는 program을 작성하게 된다. HDL을 simulation하기 위해서 별도의 simulation tool이 있는데 다음과 같은 것을 많이 사용한다.

- Cadence의 ncsim(ncverilog)
- Mentor Graphics의 modelsim
- Synopsys VCS

3.1.1.3 Logic Simulation (Gate-Level Simulation)

FPGA를 이용해서 Project를 진행하면서 합성을 한 후에 실시하는 Post-synthesis Simulation은 Timing이 아닌 Function simulation이라고 봐야 한다. 그 이유는 합성 툴에서 만들어주는 Netlist는 Timing data가 없는 Function에 대한 부분만, 즉, Logic에 대한 기능적인 부분만을 포함하는 것이 대부분이기 때문이다. 그렇지만 합성 후에 Simulation을 한다는 것은 RTL source code를 합성 툴이 맞게 합성을 했는지를 검증해 보는 것이다. 하지만 종종 합성 후에는 내가 보고자 하는 신호들이 Optimize 되어서 Netlist에는 안 나타나는 경우가 있다. 그래서 합성 후에 Simulation 단계에서 내가 체크 해보고자 하는 신호들이 Optimize 되어서 다른 이름으로 바뀌거나 보이지 않게 되는 것을 합성 툴에서 막아줄 필요가 있다.

합성이 끝나고 timing 검증 및 test를 위한 design을 삽입하고 나면 physical layer로 place & route를 하기 전 최종적인 검증이 필요하다. 이 과정을 Gate-Level Simulation 이라 한다. Gate Level simulation을 위해서는 foundry에서 제공하는 simulation을 위한 library 가 필요하다.

이 library 에는 각 gate의 function 과 기본적인 timing 정보가 포함되어있다. 그리고 SDF file과 gate 수준으로 합성된 Design Netlist 가 필요하다. 여기서 SDF file은 synthesis 단계에서 사용된 Wire load model 에 기초한 Interconnect Delay를 포함하고 있어 functional simulation 에 비해 정확성이 높은 simulation 결과를 얻을 수 있게 한다. 현재 대다수 foundry 에서는 보다 정확은 SDF file 의 추출을 위해 Cubic(삼성), Vela2(하이닉스) 와 같은 In-house tool을 개발하여 사용하고 있다.

Gate-Level Simulation을 위해 사용되는 대표적인 CAD tool 로는 functional simulation에서와 같은 NC-sim, VCS-MX, Model-Sim 이 대표적이다.

3.1.2 Equivalence Checking

Equivalence Checking은 상호 다른 두 디자인간의 functional equivalence 를 체크하는 방법이다. Netlist 가 원래의 RTL 코드를 정확하게 구현하였는가를 검증하기 위해 사용 (또는, Layout이 Gate-Level Netlist를 정확하게 구현하였는가?) 된다. 만약 Synthesis Tool을 완전하게 믿는다면 EC 는 필요하지 않다. 하지만, 대부분의 Synthesis Tool 들이 좋은 성능을 보여주지만, 이런 시스템도 밝혀지지 않은 오류가 존재 할 수 있기 때문에 검증이 필요하다.

EC 장점

- 효율적인 검증
  - 테스트 벤치를 생성할 필요가 없다.
- 효과적인 검증
  - 시뮬레이션에 비해 속도가 빠르다.
- 정형 검증
  - 가능한 모든 입력에 대해 확인이 가능 하다.
  - 시뮬레이션 기반 방법론을 사용하는 것은 불가능 하다.
- 특히, ECO (Engineering change orders) 경우에 사용하면 효율적
  - ECO 개발자게 의해 기능은 동일하지만 보다 효율적인 설계 등의 목적으로 HDL 또는 Netlist를 수동으로 변경하는 것
  - 이럴 경우 원래의 기능이 변경 되었는지 확인해볼 필요가 있는데, ECO의 경우 내부적인 Structural 변경이 크지 않기 때문에 EC 를 사용하기 적절.

지원 Tools

- FormalPro by Mentor Graphics / Conformal by Cadence / Jasper Gold by Cadence / Formality by Synopsys / Conformal LEC by Cadence Design Systems / Quartz Formal by Magma Design Automation / 360 EC by OneSpin Solutions

현재 사용 가능한 다양한 상용 EC 도구들이 존재한다. 하지만, 해당 도구들이 모든 Synthesis 의 결과물을 검증하지 못한다. 다시 말해 Synthesis 도구와 IDEs 환경에 따라 검증 가능한 EC 도구가 따로 있다고 말할 수 있다. 예로, 현재 Synplify Pro + Actel Libero SoC 환경을 지원하는 EC 도구가 없다.

Logic Synthesis	IDE	Mentor Graphics FormalPro	Cadence Encounter Conformal EC	Synopsys Formality	
Mentor Graphics Precision RTL	Xilinx ISE	O			
	Actel Libero Soc	O			
Synopsys Synplify Pro	Xilinx ISE	O	O		
	Actel Libero Soc				No LEC available
	Altera Quartus II		O		
Xilinx XST	Xilinx ISE		O	O	
Synopsys DC Ultra	-			O	

3.1.3 Static Timing Analysis

현재의 복잡한 Design은 gate level 의 HDL simulation으로는 market의 시간적 요구에 부합할 수 없다. 그렇게 때문에 Logic의 검증을 위해 여러 가지 방법론이 제시되고 있다. 그 대표적인 것이 STA(Static Timing Analysis), Formal Verification 이다.

STA는 각 논리 gate 가 요구하는 setup, hold time, clock skew를 SDF, SPF, DSPF 등의 delay정보를 고려하여 측정하고 IC에서 발생할 수 있는 timing에 대한 문제를 사전에 처리해주는 기능을 한다. 그리고 전체 Design의 Critical path를 지정해주어 Design Debug, 성능향상에 도움을 주는 과정이다. 또한 Timing Driven Layout 의 timing source 가 되는 SDC file을 출력해주기도 한다. STA를 위해 업계에서 통용되는 tool 은 Synopsys 사의 Primtime이며 최근에는 Primtime SI를 출시하여 crosstalk, IR drop으로 인해 발생하는 noise(glitch) 해석까지 지원하는 확장된 기능을 제공하고 있다.

Timing analysis는 FPGA 개발에 있어 타이밍과 동작기능은 항상 같이 고려해야 할 문제이다. STA는 IC에서 발생할 수 있는 timing에 대한 문제를 사전에 처리해주는 기능을 한다. STA는 실제 수행 없이 회로의 delay 시간을 계산하여 Violation을 확인가능하며, 설계자가 원하는 timing(clock 주파수나, delay 등)이 제대로 맞는 지 검증한다.

간단한 타이밍 위반(timing violation) 예로 두 가지가 있다.

1. Setup timing violation

합성 Constraint가 너무 타이트 한 경우 발생. 합성된 결과가 클럭 주기보다도 큰 딜레이를 갖는 Combinational logic을 가질 때. 어떤 rising edge에서 data 값을 인식하기 위해서는 data는 clock rising edge보다 이전에 유효한 값을 일정 시간 동안 가지고 있어야 함.

2. Hold timing violation

hold time 동안은 유효한 데이터가 계속 변하지 않고 유지되어야 함. 예를 들어 상승 에지에 동작하는 플립 플롭이 입력 값을 인지하기도 전에 입력이 바뀌면 hold time violation이 발생. 아래는 홀드타인요구조건이 1ns인 경우이며, 이러한 경우에는 클럭상승 에지를 기준으로 최소 1ns동안은 값을 유지하고 있어야 함.

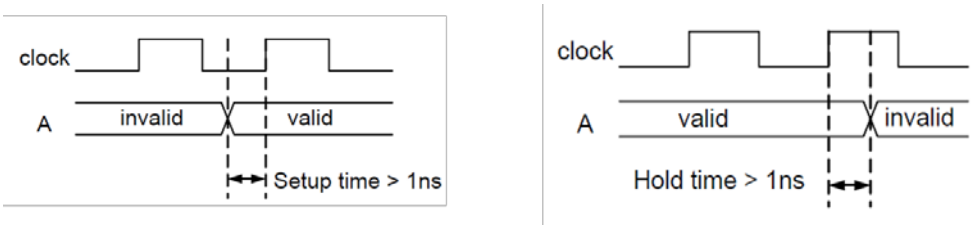


그림 12. Setup time and Hold time

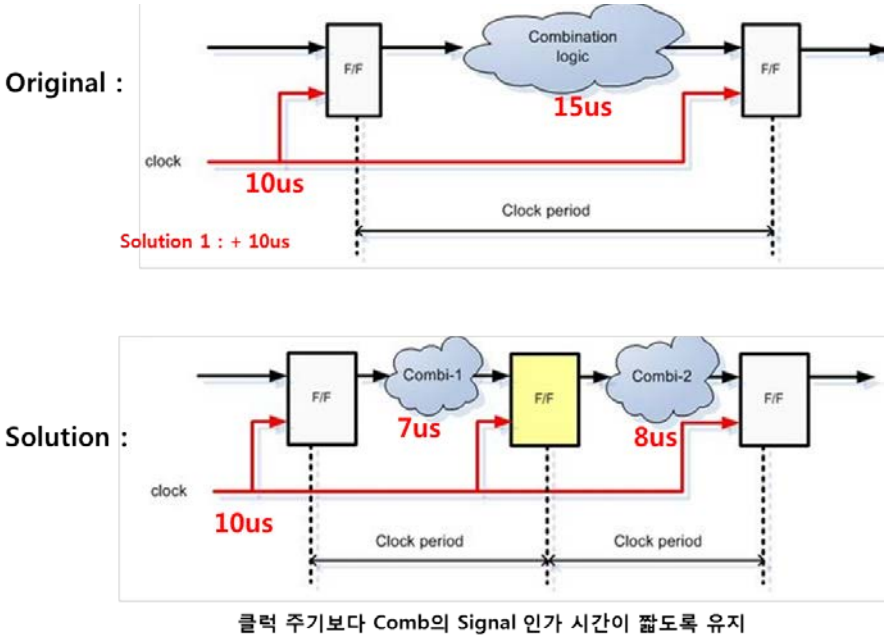


그림 13. Timing violation 과 해결책

3.1.4 Layout Verification

Layout Verification 은 DRC(Design Rule Check)와 LVS(Layout Versus Schematic)로 나뉜다.

DRC : Design Rule은 주어진 공정에서 필요한 모든 마스크에 대한 최소크기와 주변 레이어들 사이의 간격등 마스크생성을 위한 공정기술을 고려한 규정들이다. Layout은 이러한 Design 규칙에 맞게 설계되어야 하고 설계자의 실수에 의해 Design 규칙에 어긋난 부분을 검사하고 잘못된 부분

을 지적해 주는 과정을 DRC(Design Rule Check)라 한다.

LVS : 설계하고자 하는 회로의 스키메틱 디자인을 통하여 추출된 Netlist와 Layout에서 추출된 Netlist는 설계상의 오류를 통해 상이한 점이 발견될 수 있다. 이는 실제 device 가 나왔을 때 simulation와 같은 동작을 보장할 수 없다는 것을 뜻한다. 그리고 그 원인은 Layout에서 찾을수 있을 것이다. 그러므로 이런 오류를 정정하기 위한 Check 과정이 필요하고 이 과정을 LVS(Layout Versus Schematic) Check 라 한다. 이런 일련의 과정을 Layout Verification 이라하고 이런 과정을 수행하기위해 CAD tool을 사용하는데 일반적으로 Mentor사 의 Calibre(DRC, LVS), Cadence사 의 Dracula, Diva(DRC, LVS)를 사용한다.

4 Others

4.1 IP [18]

IP(Intellectual Property)의 문자 그대로의 의미는 "지적 재산(자산)"으로서 일반적으로 발명, 디자인, 표장, 저작물 등 인간의 지식 활동으로부터 나온 무형의 지적 결과물(또는 이를 보호하는 지식재산권)을 의미하는 것으로 널리 사용되고 있지만, 반도체 업계에서는 "반도체 집적회로의 설계 시 독립적인 기능을 가지고 재이용이 가능한 기능블록(반도체설계모듈)"을 말한다. 즉, IP란 반도체 회로의 로직 LSI을 구성하기 위하여 필요한 기능을 하드웨어와 소프트웨어의 상태로 정리한 블록을 말하는 것으로 칩의 코어 셀(기능 블록)과 그 코어 셀을 구동하기 위한 디바이스 드라이버, 소프트웨어, 펌웨어 등을 포함한다. 다만, 반도체 업계에서 사용하는 IP라는 용어는 용어로서 반드시 통일된 것은 아닌데, IP표준화 단체인 VSIA에서는 IP대신 "VC(Virtual Component)"라는 용어를 사용하고, 데이터퀘스트는 "SLM(System Level Macros)"이라는 용어를 사용하기도 한다.

IP제공형태에 따른 분류: IP는 통상적으로 "hard IP, firm IP, soft IP"의 3가지로 구분하는데, 이는 LSI의 전통적인 설계과정에서 생기는 성과물을 기준으로 구분한 것이다. 따라서, LSI설계과정에 대하여 간략히 살펴볼 필요가 있다. LSI의 개발은 트랜지스터의 형상과 그 사이의 배선을 미세가공 기술로써 칩 표면에 새기는 "LSI제조공정"과 이러한 트랜지스터의 형상과 배선 즉,레이아웃 디자인을 도출하기 위한 "LSI설계공정"으로 크게 나눌 수 있다.한편,LSI설계공정의 전(前)단계로 "시스템 설계공정"이 필요한데3),시스템 설계공정에서는 가격 이나 소비전력,속도 등의 기본적 성능에 대한 시스템의 요구사항4)을 고려하 여 사양을 결정하고,어떠한 기능을 하드웨어로 하고 어떠한 기능을 소프트웨어로 할 것인가의 분담을 결정하는 기능분할설계를 한다.

구분	의미	유연성 (customize)	신뢰성 (안정성)	가격
Soft IP	하드웨어 기술 언어(HDL)로 쓰여진 논리 합성 가능한 설계 자산, HDL은 VHDL과 Verilog-HDL 등이 이용되며, 합성 가능한 RTL(Register Transfer Level)이라는 하드웨어 기술 레벨이 일반적임.	설계의 수정이나 개량, 기능 변경이 쉽기 때문에, 유연성이 매우 높음	면적이나 성능 예측이 되지 않으므로, 안정성은 낮음	낮음
Firm IP	약간의 Floorplanning 정보를 가진 게이트 레벨의 netlist; RTL 기술을 기반으로 어느 정도 범위의 제조 공정 한정	중간	중간	보통
Hard IP	공정이 완전히 결정되어 있고, 레이아웃 제조까지 완료된 블록	어려움	높음	높음

4.2 HIL (High Level Synthesis)

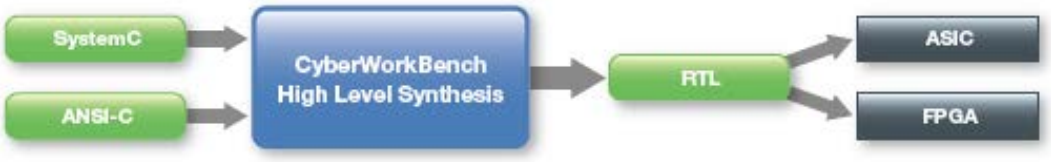
4.2.1 LabVIEW FPGA Module

HLS를 통해 각 분야의 전문가들은 ANSI C, C++ 또는 LabVIEW 코드와 같은 고급 언어를 사용하여 FPGA(field-programmable gate array)를 프로그래밍하는데 최적화된 알고리즘을 개발할 수 있습니다. HLS는 많은 사람들이 FPGA 설계 생산성을 높여야하는 필요성이 대두됨에 따라 각광 받고 있습니다. 전통적인 FPGA 설계 툴을 사용하는 경우, HDL에 익숙해야 하며 전체 프로세스 단계를 분석하고 그 유효성을 검사해야 합니다. 하지만 개발자의 능력은 FPGA와 같은 속도로 증가하지 않으므로 어플리케이션의 복잡성이 늘어날수록 개발 주기도 길어질 수 밖에 없습니다.

고급 언어를 사용할 수 있다는 것은 더 많은 사람들이 짧은 기간 동안 FPGA의 성능을 활용할 수 있음을 의미합니다. FPGA 툴 업계는 HLS가 이러한 비전을 현실로 만드는 방법으로 보고 있습니다. NI는 최근에 Xilinx의 HLS 기술을 사용하여 NI LabVIEW FPGA Module의 애드온인 LabVIEW FPGA IP Builder를 개발 했습니다. 이 애드온을 통해 HLS 기술과 LabVIEW 및 RIO 하드웨어의 성능을 결합하면 고성능 FPGA IP를 쉽게 생성할 수 있습니다. 또한 LabVIEW FPGA가 원하는 기능을 제공하지 못할 때 또는 기능이 성능 요구사항을 충족하지 못할 때 최적화된 IP를 쉽게 만들 수 있도록 지원하여 LabVIEW FPGA 설계 환경의 기능을 보완합니다.

코드와 설계 지시문의 분리를 통해 코드 재활용률을 크게 개선할 수 있는 기회가 생깁니다. 코드의 유효성을 한 번 검사한 후에는 동일한 코드를 사용하여 성능과 리소스 요구사항이 변하는 여러 어플리케이션을 해결할 수 있습니다. 코드를 변경하지 않고도 코드 유효성 검사에 필요한 시간을 줄일 수 있는 것입니다.

4.2.2 CyberWorkWench



CyberWorkWench의 기본 개념은 두가지이다..

1. All-modules-in-C
 - 모든 모듈은 C로 작성된다.
 - RTL이나 netlist는 black box로 여기겠다.
2. All-processes-on-C:
 - synthesis 와 verification은 C source code 에서 수행 한다.
 - 예를 들면, compiler 된 machine language (ex,. assembler) 를 검증하지 않듯이 / synthesis 가 된 RTL을 검증할 필요가 없다
 - 모델체킹 또는 시뮬레이션은 C level 에서 하면 된다.

그래서 CyberWorkWench 는 대략 다음과 같은 내용을 지원

1. C를 RTL로 변환
2. functionality 검증을 C 레벨에서 검증 가능하도록 지원 (내부적으로 RTL 레벨을 검사하지만 사용자는 C 레벨에서 검사 가능하게 지원)
3. performance 를 RTL 레벨에서 검증 가능하도록 지원 (역시 사용자는 C source code로 검사)
4. Testbench Generator
5. Formal verification tool

따라서, C description이 오로지 유인한 final description language로 역할을 하게 된다.

참고 문헌

[1] 지식백과, "Naver," 2015. [온라인]. Available: http://terms.naver.com/entry.nhn?docId=2079654&cid=50305&categoryId=50305.

[2] 빛. 복음의, "Naver blog," 2006. [온라인]. Available: http://blog.naver.com/ahinoam75/50001681508.

[3] 김세중, "[ASIC설계의 기초 2]-FPGA 란 무엇인가?," 2005. [온라인]. Available: http://www.semiclub.co.kr/jboard/?p=detail&code=infomall003&id=107&page=2. [액세스: 2015].

[4] 진선옥, "대용량 FPGA, 시스템 설계 대체," 2000. [온라인]. Available: http://www.pldworld.com/html/technote/nikkei200004.htm. [액세스: 2015].

[5] wiki, 2015. [온라인]. Available: https://ko.wikipedia.org/wiki/FPGA.

[6] saffroncrocus, "FPGA 이야기," [온라인]. Available: http://fpga.tistory.com/28.

[7] J. Ranta, "The current state of FPGA technology in the nuclear domain," VTT, 2012.

[8] 김인수, "VHDL의 개요 (ppt)," sungkukwan university, 2015.

[9] 가치창조기술, "[FPGA와 Verilog 초보자 가이드 2] 모듈," 2015, [온라인]. Available: http://wiki.vctec.co.kr/devboard/fpga/spartan-3a-fpga-gaebalbodeu--elbert/module. [액세스: 2015].

[10] 나무위키, "verilog," 2015. [온라인]. Available: https://namu.wiki/w/Verilog. [액세스: 2015].

[11] 에. 떠돌이, "VHDL?," 2015. [온라인]. Available: http://blog.naver.com/shl0309/100010353747. [액세스: 2015].

[12] 나무위키, "VHDL," 2015. [온라인]. Available: https://namu.wiki/w/VHDL. [액세스: 2015].

[13] operands, "디지털 합성 (Synthesis) 이란 무엇인가," 2012. [온라인]. Available: http://blog.naver.com/ner34/40153810007. [액세스: 2015].

[14] Synopsys, "Design Compiler® User Guide," 2011.

[15] 배영돈, "마이크로프로세서 설계 무작정 따라하기," 2002.

[16] karakkangsik, "Netlist," 2007. [온라인]. Available: http://blog.naver.com/karakkangsik/130020943125. [액세스: 2015].

[17] holelee, "RTL Engineer가 본 반도체 칩 개발 과정," 2009. [온라인]. Available: http://blogspot.designonchip.com/2009/10/rtl-engineer.html. [액세스: 2015].

[18] 이경란, "반도체설계재산의 향후전망 및 보호방안 연구," 이지국제특허법률사무소, 2005.