하드웨어 설계, 이제는 프로그래밍으로! 1

최상의 하드웨어 설계 방법을 찾아라

흔히 하드웨어를 설계하는 작업이란 회로도를 '그려가는 작업'으로, 생성된 회로도를 인쇄 회로 기판으로 구현해서 각종 부품이나 칩을 납땜하는 작업이 전부인 것으로 알고 있는 독자가 많을 것이다. 물론 틀린 말은 아니다. 그러나 문제는 회로를 기계가 아닌 사람이 직접 그리기 때문에 그 회로의 복잡도에는 한계가 있을 수밖에 없다. 따라서 보다 복잡한 회로를 설계할 때는 하드웨어의 동작을 정확히 묘사할 수 있는 하드웨어 기술 언어가 필수적으로 사용된다.

연재순서

1회 2002. 8 | 최상의 하드웨어 설계 방법을 찾아라
2회 | 완벽한 하드웨어 기술언어, VHDL
3회 | 나의 첫 하드웨어 프로그래밍, 자판기 설계 프로젝트

연재가이드

운영체제 | 윈도우 98 이상
개발도구 | Xilinx Foundation (PC & UNIX)
기본지식 | 논리 회로, 디지털 시스템
응용분야 | 디지털 회로의 설계, ASIC/FPGA의 설계

곽종욱 celot@naver.com

하드웨어를 디자인할 수 있는 언어 VHDL과 Verilog 프로그래밍에 관심을 갖고 있으며, 링 기반 고성능 대형컴퓨터 개발 프로젝트에 참여한 바 있다. 자신을 하드웨어 엔지니어라고 소개하는 필자는 이 글을 통해 디지털 시스템 설계와 하드웨어 기술 언어를 이용한 프로그래밍에 관심을 갖는 독자가 많아지기를 바란다고.

디지털 시스템이나 하드웨어를 구현하는 방법은 일반 소프트웨어의 설계 절차와 비교해 그리 단순한 작업만은 아니다. 물론 소프트웨어의 설계 방법과 개발, 관리 및 유지 보수의 기법 등을 다루는 소프트웨어 공학적인 관점에서 보면 소프트웨어 제작 역시 보다 더 효율적이고 합리적인 제품의 개발을 위해 다양한 절차와 방법을 이용한 복잡한 설계 기법을 사용하기도 한다. 하지만 하드웨어를 디자인한다는 것도 하드웨어 공학적인 관점에서 보면 그 절차가 더 복잡하면 했지 그리 만만치 않다. 특히 디지털 시스템과 하드웨어 디자인의 배경 지식이 없는 독자들이라면 일단 머릿속에 떠오르는 것이 소프트웨어를 개발한다는 것은 모니터와 키보드 앞에서 '뚝딱뚝딱' 거리면 완성되는 것이고, 하드웨어를 제작한다는 것은 모니터 앞에 앉아서 해야 하는 작업도 물론 필요하겠거니와 때로는 장갑을 끼고 납 냄새를 맡아 가면서 해야 하는 작업이라 생각할 것이다.

완전히 틀린 말은 아니지만 어쩌면 이런 선입관이 수많은 IT 인력을 배출해 나가는 우리 나라의 현실에 비춰 볼 때 하드웨어 인력이 상대적으로 적은 이유인지도 모른다. 필자는 이 글을 통해 디지털 시스템 및 하드웨어를 구현하는 일반 방법론을 소개할 것이며 아울러 하드웨어도 소프트웨어와 마찬가지로 프로그래밍이 가능하다는 것을 소개하면서 이를 설명해 줄 것이다. 이 연재를 통해 하드웨어를 설계하는 방법에 대한 새로운 마인드가 독자에게 심어지길 바란다.

디지털 회로의 구현 방법

디지털 회로를 구현하는 여러 방법들에 대해 생각해보자. 디지털 회로의 하드웨어적 구현 방법은 회로의 기능이나 복잡도 그리고 응용 분야 등의 다양한 요소에 의해 결정된다. 이러한 여러 가지 하드웨어의 구현 방법 중에서 적합한 방식의 선택은 해당 부품의 제조 가격과 개발 기간 및 시스템의 크기와 성능을 좌우하는 결정적인 요인이 되기도 한다. 이런 적절한 구현 방법의 선택과 아울러 하드웨어의 설계를 보조해 주는 도구인 CAD(Computer Aided Design) 툴의 선택 및 회로를 구현하고자 하는 설계자의 툴 사용 숙련도가 디지털 회로 구현의 또 하나에 중요한 관건이라 하겠다. 이러한 디지털 회로의 구현 방법은 그 방법론에 따라서 몇 가지로 정형화된 구분 방법이 있는 것은 아니나 필자의 기준으로 볼 때 크게 다음과 같은 세 가지 분류가 가능하다.

표준 논리회로 소자를 이용한 구현

일반적으로 가장 간단한 회로를 설계하고자 할 때 이용되는 방법으로, 주로 SSI 시스템이나 MSI 시스템 등의 구현이 이와 같은 방법으로 이뤄진다. 하지만 LSI 혹은 VLSI와 같은 고밀도 집적 회로의 직접적인 구현에는 한계가 있는 방식으로 실제로 흔히 주변에서 볼 수 있는 개별 단일 소자를 이용해 PCB상에서 직접 구현된다. 주로 디지털, 아날로그, 메모리 등이 혼용된 복합 회로의 구현에 용이하며 개발 기간과 비용 단축에 효과적이다. 하지만 고기능, 고성능 회로의 구현에는 어려운 점이 많다. 이러한 표준 논리 회로 소자를 이용한 구현 방법에서는 개별 소자를 이용한다는 것 자체가 이미 각각의 개별 소자의 제작에는 관심을 두지 않는다는 것을 의미하며 따라서 이런 방식을 이용한 시스템에서는 이런 미리 제작된 개별 소자를 이용한 PCB에서의 회로의 배치 · 배선(P&R, Placement and Routing)이 중요 이슈가 된다.

마이크로 컨트롤러를 이용한 구현

이 구현 방법은 i8051, M68HC11, Z80 등 이와 같은 표준형 마이크로 제어 유닛(MCU, Micro Control Unit)을 사용한다. 이러한 MCU는 흔히 주위에서 쉽게 접할 수 있는 CPU로 대변되는 MPU(Micro Processing Unit)와는 그 성격이 조금 다르다. 자세한 성격상의 구분은 〈표 1〉을 참조하기 바란다. 이러한 마이크로 컨트롤러를 이용한 구현방법에서는 주로 내장형 또는 외장형으로

인쇄용 회로 기판, PCB

흔히 '피씨비'라 불리는 인쇄용 회로 기판은 그 위에 칩이나 기타 다른 전자 부품들이 설치되어 있는 얇은 판을 의미하는데 이 보드는 강화 섬유 유리나 플라스틱으로 만들어지며 구리로 된 회로를 통해 부품들이 서로 연결되어 있다. 컴퓨터 시스템 내에 주요 PCB(Printed Circuit Board)로는 시스템 보드 또는 마더 보드라고 불리는 핵심 PCB가 있으며, 메인 보드의 슬롯에 꽂히는 작은 PCB들은 그냥 '보드' 또는 '카드'라고 부른다. 인쇄 회로 기판이라는 PCB 용어에서도 알 수 있듯이, 이렇게 회로가 '인쇄되었다'는 말은 보드 내에 회로가 실제로 선명하게 새겨진 회로를 의미한다. 자세한 공정이야 생략하더라도 구리 박판을 강화 섬유 유리나 플라스틱 기판 위에 놓은 다음 이를 에칭(동판의 부식)하는 형식으로 기판에 회로를 새기게 되는 방식이다.

이러한 PCB는 컴퓨터에서 여러 형태의 보드들로 구분돼 있는데 이러한 보드들은 다음의 부류로 나눠진다.

- **마더 보드** : 주변장치를 버스에 부착할 수 있도록 커넥터를 가지고 있는 주 기판이다. 일반적으로 마더 보드는 CPU, 메모리 그리고 기본적인 시스템 컨트롤러 등을 담고 있다. PC에서는 마더 보드를 주로 시스템 보드 또는 메인 보드라고도 부른다.
- **확장 보드** : 컴퓨터의 확장 슬롯에 꽂아 넣는 카드이다. 확장 보드에는 컨트롤 보드, 네트워크 카드, 비디오 어댑터 등이 이 범주에 포함된다.
- **도터 보드** : 다른 보드에 직접 부착되는 보드이다.
- **컨트롤러 보드** : 확장 보드의 특별한 형태로서 주변 장치의 컨트롤러가 포함된 보드이다. 디스크 드라이브나 그래픽 모니터 등과 같은 새로운 주변 장치를 컴퓨터에 부착하려면 컨트롤러 보드가 필요하다.

IC 칩의 집적도에 따른 분류

소규모 집적(Small-Scale Integration, SSI) 회로
- 10개 이하의 독립적인 게이트가 하나의 칩에 포함된다.
- 게이트의 입출력이 바로 외부 핀과 연결되어 있는 형태가 많다.

중규모 집적(Medium-Scale Integration, MSI) 회로
- 10에서 200개 정도까지의 게이트를 집적한다.
- 기본적인 디지털 장치 구현을 위해 사용된다.
- 디코더, 가산기, 레지스터 등이 예가 될 수 있다.

대규모 집적(Large-Scale Integration, LSI) 회로
- 200에서 1000개 정도까지의 게이트를 집적한다.
- MSI의 세부 모듈을 이용하기도 하며 이를 통해 디지털 시스템을 형성하는 데 사용된다.
- 간단한 프로세서, 메모리 등이 해당된다.

초대규모 집적(Very-Large-Scale Integration, VLSI) 회로
- 수천 개 혹은 그 이상의 게이트를 하나의 칩에 집적한다.
- 일부 분류에서는 VLSI의 보다 상위 개념으로 ULSI(Ultra-Large-Scale Intergration)라고 세부적으로 구분하기도 한다.
- 대형 메모리, 마이크로 컴퓨터 칩을 형성한다.

존재하는 프로그램 메모리에 회로의 기능을 어셈블리 프로그램 형태로 탑재해 이를 통해 제어 작업을 수행하게 된다.

따라서 앞서 설명한 표준 논리 회로 소자를 이용한 방법에 비해서는 어느 정도 프로그래밍 철학이 첨가된 것으로 비교적 다기능, 고성능의 제어 및 데이터 처리에 적합하다. 하지만 그 사용처는 MCU의 특성상 주로 특수 목적용으로 한정되는 경우가 많다. 이런 방식을 이용한 구현 방법에서는 내장되어 있는 프로그램의 코드 크기와 실행 속도를 최적화시키는 일이 핵심이며 그 외에 다른 모듈들, 가령 타이머, 카운터, UART 등 MCU 칩 주변 회로의 효율적 활용을 위한 설계가 중요 이슈가 된다.

주문형 반도체를 이용한 구현

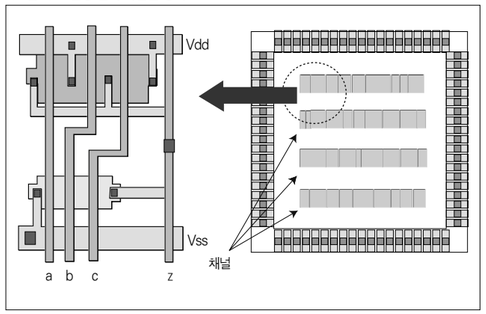
끝으로 앞서 설명한 두 가지 기법보다 더 복잡한 회로 설계시에 사용되는 그리고 이번 연재의 철학과도 가장 관련이 있는 주문형 반도체를 이용한 구현 방법이 있다. 주로 이 구현 방법은 '특정한 용도의 주문형 집적 회로'로 대변되는 ASIC(Application Specific Integrate Circuit, 에이직 혹은 에이식으로 발음)을 이용한 구현 방법을 말한다. 이러한 ASIC은 넓은 의미로는 메모리와 일반 표준 IC를 제외한 각 시스템별로 사용되는 전용 IC를 의미한다. 부연 설명하자면 기존의 일반 메모리와 표준 IC 혹은 일반 용도로 사용되는 CPU 등은 그 기능이 미리 정해져 있고 그 사용처가 일반 목적으로 사용될 수 있음을 뜻한다.

하지만 주문형 반도체라는 것은 그 이름에서도 알 수 있듯이 '특수 목적용으로 특정 응용'에 맞게 설계되고 제작된 일반이 아닌 전용 IC를 의미한다. 앞서 말한 두 가지 구현 방법과 비교할 때 가장 복잡한 수준의 회로 기능 묘사가 가능하다. 그리고 본 연재에서 추후 다루게 될 하드웨어 기술 언어를 사용한 회로의 구현도 여기에 해당되는 구현 방식이라 할 수 있다. 주문형 반도체를 이용한 구현 방법의 세부 토픽으로 이러한 구현 방법에 대한 보다 더 세분화된 분류에 대해 앞으로 자세히 다루기로 한다.

〈표 1〉 MPU와 MCU의 비교표

MPU(Micro Processing Unit)	MCU(Micro Controlling Unit)
일반적으로 32비트 - 64비트의 데이터 처리 능력을 가진다.	사용되는 칩의 목적에 따라 조금씩 다르지만 8비트 - 16비트 정도의 데이터 처리능력을 가진다.
일반 목적용 처리기로 사용 가능하다. 주로 컴퓨터 CPU에 사용된다.	주로 특수 목적의 PU(Processing Unit)로 사용된다.
제어 기능뿐만 아니라 빠른 계산속도와 데이터 처리 능력이 주된 이슈이다.	속도보다는 제어가 주된 이슈이다. 타 기능에 비해 인터럽트기능이 상대적으로 강화되어 있다.
현재 최대 수십 GHz 이상의 속도를 보인다.	주로 100MHz 이하의 속도를 보인다.

〈그림 1〉 표준형 셀에 기반한 방식



주문형 반도체 설계

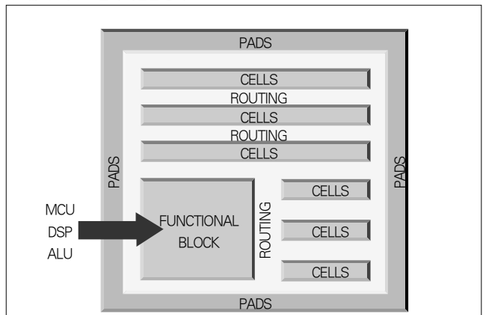
이상에서 알아본 바와 같이 디지털 시스템을 이용한 하드웨어 제작은 회로의 기능과 복잡도 그리고 응용 분야의 적합성 등에 따라서 다양한 구현 방법을 취한다. 그 중에서도 제일 복잡한 회로를 묘사할 수 있는 고기능 · 고성능의 능력을 가진 구현 방법이 앞서 설명한 대로 주문형 반도체를 이용한 구현 방법이다. 그렇다고 해서 고기능 · 고성능의 설계 기법이 항상 최적의 설계 기법이라 할 수는 없다. 가령 단순한 작업과 제어만을 필요로 하는 응용 분야에서 쓸데없이 비싼 고가의 칩을 이용한 주문형 반도체 설계 방식을 이용해 구현할 필요가 없는 것이다. 응용 분야의 적합성에 대한 중요성을 다시금 강조할 수 있을 것이다.

그럼 지금부터는 이러한 주문형 반도체의 설계 방법에 대한 세부 분류를 알아보고, 점차 우리가 다루고자 하는 분야로 그 범위를 줄여 나가기로 하겠다. 주문형 반도체를 이용한 방식은 크게 완전 주문형 방식과 반 주문형 방식으로 구분한다. 미리 설명하긴 했지만 구분하고자 하는 기준이 공학적 견해에 따라 약간 그 분류상의 위상이 달라지기도 하지만 필자 나름대로 최대한 구분을 체계화하고자 했음을 알아주길 바란다.

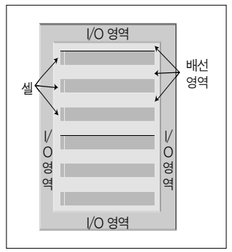
완전 주문형 방식

먼저 완전 주문형(Full-Custom) 방식이다. 이는 흔히 사용자의 요구로 대변되는 주문(order)과는 별개로 제작되는 범용 메모리나 CPU, 즉 일반 목적용으로 사용되는 칩들과는 상대적인 개념이다. 완전 주문형 방식은 구현하고자 하는 IC 회로를 CAD 툴을 사용해 직접 설계, 웨이퍼 가공, 검증 등의 단계에 이르기까지 레이아웃 도면을 직접 디자인해야 하기 때문에 디자인 기간이 많이 걸린다. 특히 보다 최적화된 회로의 구현을 위해서 트랜지스터 단위로까지의 설계가 이뤄지기 때문에 반도체 칩 설계에 대한 고도의 전문가가 필요시 되는 분야이기도 하다. 하지만 칩 제조의 처음 단계부터 시작해 모든 단계를 직접 설계하기 때문에 최적화가 가장 잘 되어 있는 형태이기도 하다. 즉, 가장 적은 면적으로 가장 많은 소자를 집적시킬 수 있고 따라서 칩의 대량 생산시 가장 싼 가격에 가장 빠른 동작 속도를 보장할 수 있다. 초기 투자 비용만 감소시킬 수 있다면 대량 생산을 위한 방식으로는 가장 적합한 방식이라 할 수 있다.

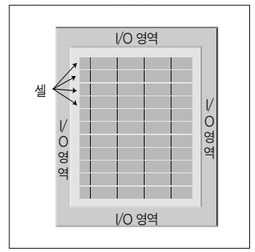
〈그림 2〉 매크로 셀에 기반한 방식



〈그림 3〉 게이트 어레이에 기반한 방식



〈그림 4〉 SOG에 기반한 방식



반 주문형 방식

반 주문형(Semi-Custom) 방식에서는 트랜지스터 단위의 설계가 이뤄지는 완전 주문형 방식과는 다르게 게이트나 레지스터 단위의 설계가 주로 이뤄진다. 바꾸어 해석하면 이미 게이트나 레지스터는 만들어져 있으며 이들의 상호 연결이 주된 관심사가 된다는 의미된다. 또한 완전 주문형 방식에서의 주된 관심사라 할 수 있는 트랜지스터 단위의 설계는 아예 관심의 대상이 되지 않는다는 뜻이기도 하다. 즉 제작자는 이렇게 미리 제작되어진 기존의 게이트나 레지스터를 특정 동작을 수행하게끔 CAD 툴을 사용해 서로 연결시켜주는 일만 하면 된다. 이런 의미에서 반 주문형 방식이라고 하는 것이다. 이런 반 주문형 방식은 다시 그 구현 방법의 특징에 따라 표준형 셀(cell)에 기반한 구현 방식, 게이트 어레이에 기반한 방식, 그리고 끝으로 가장 복잡한 형태의 반 주문형 방식이라 할 수 있는 프로그램 가능한 소자를 이용한 구현 방식으로 세분화될 수 있다. 각각에 대해 좀더 자세히 알아보자.

표준형 셀 기반 방식

표준형 셀은 사전에 미리 제작되고 동작에 있어 명확한 정의가 이뤄진 게이트 혹은 래치(latch) 등의 기본 셀을 이용해 설계를 완성하는 방식이다. 〈그림 1〉에서 보는 바와 같이 표준형 셀 기반 방식에서는 칩 내부가 표준 셀들로 이뤄져 있으며 그 경계에 I/O 영역과 배선 영역(routing)이 존재하는 형태를 보인다. 즉 경계 영역에 배선과 관련된 부분을 고려해 제작자가 이를 상호 연결해 적절한 동작을 수행하도록 해야 하며, 아울러 I/O 영역을 디자인하여 칩 외부로 나오는 핀들을 상호 맵핑시켜주는 작업이 필요하다.

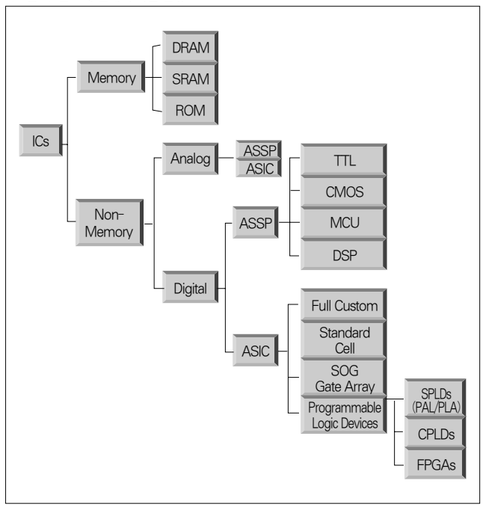
표준형 셀 기반 방식은 기본적으로 이러한 셀들로 구성되어 있고 상호 연결되지 않은 경계 영역들로 이뤄져 있지만 이런 셀 구조 역시 간단한 ALU와 ROM 등의 존재 여부에 따라서 표준형 셀(PolyCell) 방식과 매크로 셀(MarcoCell) 방식이란 용어를 사용해 재 구분되기도 한다. 표준형 셀 방식은 〈그림 1〉과 같은 형태라 할 수 있으며 매크로 셀 방식은 〈그림 2〉에 나타나 있다. I/O 핀들을 맵핑시킬 패드 부분과, 미리 만들어져 있는 셀, 그리고 이를 연결시킬 라우팅 영역 이외에 별도로 기능 블럭이란 것이 존재한다. 칩 내에 존재하는 서브 모듈이라 할 수 있다. 간단한 제어 유닛이나 ROM 등이 이러한 서브 모듈로 존재할 수 있다.

게이트 어레이 기반 방식

게이트 어레이에 기반하는 설계 방식은 칩을 설계하는 단계에서 각각의 셀을 어레이 형태로 배치하고 이를 공정 및 제작 단계에서 어레이들 간의 배선만 바꿈으로써 목적하는 기능에 맞게 동작하도록 회로를 구성하는 방식이다. 표준형 셀에 기반한 방식과 비교해서 칩의 크기, I/O의 위치, 특히 셀들의 종류와 위치, 모양 등이 고정되어 있다는 차이점이 있다. 사실 이 방식은 설계상의 제약이 많고 표준 셀 방식보다 더 정형화되어 있기 때문에 칩의 면적이 낭비될 수 있다는 문제점이 있기도 하다.

다시 말해 주어진 어레이에서 모든 셀들이 사용되든 그렇지 않든 모두 설치돼 있어야 한다는 것이 단점으로 지적된다. 게이트 어레이에 기반한 방식이 〈그림 3〉에 나타나 있다. 또한 게이트 어레이 방식을 따르면서 이에 대한 세부적인 특수 형태로 SOG(Sea of Gate) 형태의 칩이 있다. 이 방식은 〈그림 4〉에서도 보는 바와 같이 셀을 연결하기 위한 별도의 배선 영역을 없앤 형태이다.

〈그림 5〉 반도체 설계의 기술 분류



프로그램 가능한 소자를 이용한 구현 방식

프로그램 가능한 소자라는 것은 하드웨어 칩을 디자인함에 있어 이를 실제 칩 제조 공장 같은 생산 현장이 아닌 실험실이나 연구실 같은 설계 현장에서 구현할 수 있게끔 만들어진 소자를 의미한다. 부연하면 공장에서 대량 생산되기 이전에 연구실 혹은 실험실 차원에서 디자인한 하드웨어 회로를 즉시 칩으로 구현해 시스템 수준의 검증을 신속하게 이뤄지도록 하는 것으로써 하드웨어 칩의 프로토타이핑 작업에 유용하게 사용되는 형식이다.

흔히 PLD(Programmable Logic Device)라는 프로그램 가능한 소자는 그 기능에 따라서 다시 다음과 같이 크게 나뉘어 진다. 일반적으로 크게 PROM(Programmable ROM), EPROM(Erasable PROM), EEPROM(Electrically Erasable PROM)으로 이어지는 ROM(Read Only Memory) 계열과 PAL(Programmable Array Logic)/PLA(Programmable Logic Array), 그리고 CPLD(Complexed Programmable Logic Device)와 FPGA(Field Programmable Gate Array) 등으로 나눠질 수 있다. 독자 여러분도 ROM 계열의 구분에 대해서는 많이 접해 왔으리라 생각한다. 즉 프로그래밍 가능 여부와 이를 다시 재 프로그래밍할 경우 그 매체가 자외선이냐 전기냐의 특징에 따라 이와 같이 구분하는 것이다.

PAL은 기본적으로 EEPROM과는 상반된 구조를 지니고 있으며 AND와 OR 어레이 로직이 그물망 형태로 구성되어 있다. PLA도 이와 상호 대칭적인 유사함을 보인다. 회로 설계에 있어서 보다 더 고밀도인 IC는 ROM과 PAL/PLA와 비교해 볼 때 CPLD와 FPGA가 더 적합하다고 할 수 있다. CPLD는 기본 로직 블럭으로 매크로 셀을 사용하며 PAL과 유사하지만 상대적으로 집적도가 높고 속도면에서 빠르다. 마지막으로 살펴볼 FPGA는 고밀도의 고속 PLD를 위한 소자로서 내부 구조상으로는 게이트 어레이와 비슷하지만 내부 회로의 배선이 기존의 것처럼 특수한 도구에 의해서 연결되는 것이 아니라 프로그래밍에 의해 연결된다는 점에서 차이가 있다. 다른 PLD에 비해서 속도가 월등히 뛰어나며 집적도가 좋다.

이와 같이 우리는 디지털 회로를 구현하는 방법에 대해 알아봤다. 특히 주문형 반도체를 이용한 구현 방법에 있어 좀더 세분화된 관점에서 분류해 살펴봤다. 재차 강조하지만 이런 분류는 전문가 입장에 따라서 그 위상을 조금 달리 할 수도 있음을 알아주길 바란다. 하지만 디지털 회로를 구현하는 분야에서 주문형 반도체 쪽으로 점점 세분화되어 내려올수록 더욱 집적화, 고속화되는 경향을 보이고 상대적으로 완전 주문형 반도체에 비해서 반 주문형 반도체가 보다 더 융통성(flexiblity)을 가진다는 것을 알 수 있었을 것이다. 이 한 가지가 이해되었길 바란다.

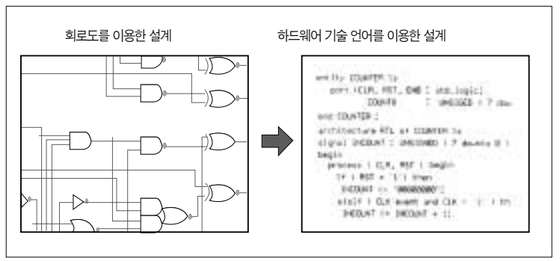
또한 한 가지 주의할 것은 지금까지의 이런 다양하고 세분화된 분류가 FPGA에 이르러 마감이 되었다고 해서 FPGA가 '궁극의 기술은 아니다'는 것을 주지하기 바란다. 다만 집적도와 속도 그리고 사용자의 동작 제어에 대한 기능 부여, 융통성적인 측면에서 볼 때 FPGA가 가장 적절한 소자 형태라는 말이다. 특히 본 연재의 후반부에서 말하고자 하는 하드웨어 기술 언어를 이용한 디지털 시스템 구현에 있어 주된 이슈가 바로 FPGA가 된다는 것을 말하기 위함이며 이러한 이슈가 디지털 회로를 구현하는 방법 중 어디에 해당되는 분야인지 그 위상을 파악할 수 있기를 바란다. 이상의 설명을 바탕으로 디지털 시스템을 포함한 일반 반도체 설계의 기술적 분류를 보면 다음 〈그림 5〉와 같다.

〈그림 5〉를 보면 집적 회로, 즉 IC는 메모리와 비 메모리 분야로 나뉨을 알 수 있다. 주지하다시피 우리 나라는 자랑스럽게도 메모리 분야, 특히 DRAM 분야에서 세계 최고를 자랑한다. 그리고 또 하나의 큰 분류인 비 메모리 분야가 존재한다. 비 메모리 분야는 다시 아날로그와 디지털 시스템으로 구분되며 디지털 시스템 분야로 내려오면 ASIC, PLD, FPGA 등이 분류상에 존재함을 알 수 있다.

하드웨어 기술 언어의 세계, HDL

이제까지 다양한 관점의 분류를 통한 하드웨어 설계 기법과 방법론에 대해 살펴봤다. 지금부터는 이러한 CPLD/FPGA 등을 설계할 때 사용하는 하드웨어 기술 언어에 대해 알아보기로 한다. 하드웨어 기술 언어 즉 HDL(Hardware Description Language)을 이용한 설계 방법은 앞서 설명한 바와 같이 FPGA 등을 제작하는데 있어 또한 ASIC 칩의 설계시 주로 사용되는 방법이다. 이는 하드웨어를 제작하는 데 있어 회로도를 이용한 설계가 아니라 하나의 언어를 이용한 설계라는 점에서 그 특징이 있다.

〈그림 6〉 회로도를 이용한 설계와 VHDL을 이용한 설계



〈그림 6〉에서 왼쪽의 그림은 일반 회로도를 이용해 하드웨어 모듈을 디자인하는 예를 나타낸다. 이에 반해 우측은 하드웨어 기술 언어를 이용한 설계 방법을 나타낸다. 하드웨어 기술 언어를 이용한 설계 방법은 다양한 분야에 이용될 수 있다는 점에서 장점이 되기도 한다. 이는 ASIC 뿐만 아니라 CPLD/FPGA에 광범위하게 이용할 수 있으며 또한 상대적으로 간단한 소규모 회로의 디자인에도 전혀 무리 없이 사용할 수 있다. 회로도를 이용한 설계 방법과 하드웨어 기술 언어를 이용한 설계 방법을 〈표 2〉에 비교해 놓았다.

몇몇 특징에 대해 부연 설명하면 다음과 같다. 첫째, 하드웨어 기술 언어를 이용해 설계하게 되면 복잡한 논리식으로부터 설계자를 해방시킬 수 있다는 장점이 있다. 즉 부울 대수식을 통한 논리 연산도를 유추해 내고 이를 설계하는 과정 없이 바로 일반 프로그래밍 마인드로 작업할 수 있다는 말이 된다. 이는 설계 기간을 단축시키고 설계의 변경을 쉽게 할 수 있다는 점에서 중요한 장점으로 부각된다. 이 때 한 가지 조심해야 할 것은 이러한 하드웨어 기술 언어라는 것이 일반 프로그래밍 마인드로 작업할 수 있다는 말이 되는데, 이는 좀더 엄격하게 생각하면 틀린 말이 된다. 즉 기본적으로 하드웨어를 디자인하는 과정은 회로도를 그리는 과정이 아닌 코딩하는 과정임에는 틀림없지만 코딩하고 있는 그 대상이 소프트웨어가 아닌 하드웨어라는 것이 중요한 이슈이다. 이는 기본 동작 메커니즘에 있어 소프트웨어와 하드웨어의 차이에서 기인한다.

〈표 2〉 회로도와 VHDL을 이용한 설계 비교

회로도를 이용한 설계	하드웨어 기술 언어를 이용한 설계
회로도를 입력하는데 시간이 걸린다.	텍스트로 간단히 입력한다.
부울 대수식과 같은 논리식을 생각해야 한다.	논리식을 생각할 필요가 없이 일반 프로그래밍 마인드로 제작 가능하다.
설계한 내용을 변경하기가 쉽지 않다.	설계한 것을 쉽게 변경할 수 있다.
설계자 이외는 내용을 이해하기가 어렵다.	체계적 프로그램이라면 어느 사람도 이해하기가 쉽다.
특적 업체의 CPLD 혹은 FPGA로 설계하려면 그 업체가 제공하는 라이브러리 프로그램을 이용해야 한다.	어느 회사에서 만든 CPLD 혹은 FPGA를 모두 사용할 수 있다.

소프트웨어와 하드웨어의 차이가 비단 한 두 가지만은 아니겠지만, 이 둘 간에 가장 중요한 차이점은 순차성과 병렬성이라 할 수 있다. 소프트웨어는 프로그램 내의 코드가 기본적으로 순차실행으로 이뤄진다. 가령 코드 앞부분에 순환문을 실행하면서 뒷부분에 제어문이 동시에 실행된다던가 하는 일은 없다. 하지만 이는 하드웨어에서는 해당사항이 없는 말이다. 즉 어떤 하나의 클럭에 동기화되어 있는 각각의 하드웨어 모듈이 있다면 클럭의 신호가 발생할 때마다 이와 연관되어 있는 모든 하드웨어 모듈들은 서로 병렬적으로 수행한다는 것이다. 이는 소프트웨어와 하드웨어 둘 간의 중요한 차이점이라는 것을 명심하기 바란다.

이러한 하드웨어 기술 언어는 여러 가지가 있지만 대표적인 두 가지를 꼽으라면 Verilog와 VHDL(VHSIC Hardware Description Language)을 들 수 있다. 본 연재에서는 이 두 하드웨어 기술 언어를 가지고 이들 간에 언어적 차이를 논하지는 않겠다. 다만 현재 이 두 가지 언어가 실험실 차원에서 연구실 차원에서 비교적 대등한 비율로 많이 사용되고 있으며 특히 학계에서는 VHDL을, 업계에서는 Verilog를 주로 사용한다는 정도만 이야기하고 본 연재에서는 그 중 VHDL에 대해 하드웨어 프로그래밍의 철학과 그 문법적 내용에 대해 자세히 알아보고자 한다.

'The HDL Page'

사이트의 타이틀이 'The HDL Page'라는 것만으로도 알 수 있듯이 VHDL과 Verilog를 이용한 ASIC, FPGA의 구현에 대한 다양한 정보를 포함하고 있는 사이트이다. 많은 예제 코드들을 보유하고 있으며 시뮬레이션과 합성 등에 대한 자료도 얻을 수 있다.

http://www.angelfire.com/electronic/vlsi/vlsi.html

VHDL 언어

자, 이제 우리가 살펴볼 하드웨어 기술 언어는 VHDL이다. VHDL은 'VHSIC Hardware Description Language'의 약자이다. 여기서 또한 'VHSIC'은 'Very High Speed Integrate Circuit'의 약자이다. 해석하자면 대충 '하드웨어의 동작을 설명하는 언어'인데 '매우 빠른 속도'라는 게 덧붙어 있는 형태가 된다. 눈치 빠른 독자라면 이 약자에서도 많은 것을 유추할 수 있을 것이다. 즉 개발 속도가 빠르고 고속으로 동작이 가능하며 집적도가 높은 ASIC을 개발하기 쉽게끔 제작된 언어라는 것을 알 수 있을 것이다. 그리고 또한 VHDL은 실제 하드웨어적 동작의 기술 능력이 많이 포함된 언어이며 문법적으로 볼 때도 그리 복잡하지 않은 언어이다.

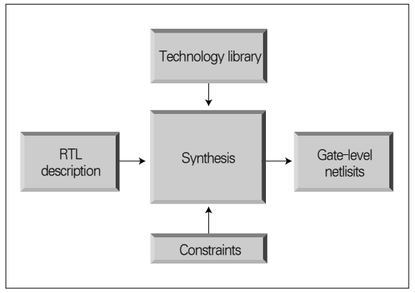
다음으로 이러한 VHDL의 등장 배경을 살펴보자. VHDL은 본래 상위 레벨의 언어(high level language)처럼 쓰도록 제작됐다. HDL 이전의 하드웨어 설계에 있어서는 레이아웃 편집기 같은 것을 이용해 작은 블럭을 설계하고 이것을 이용해 큰 블럭으로의 설계를 확장시켜 나가는 상향식 설계(bottom-up) 방식을 사용했다. 하지만 이런 설계 기법은 설계해야 할 회로의 규모가 커지고 복잡도가 증가함에 따라 자체적인 한계를 가질 수밖에 없는 방식이었다.

특히 일부 학자들은 사람 두뇌의 한계, 즉 사람이 일일이 추적할 수 있는 게이트의 한계가 약 5000개 정도라고 보고했고, 이를 바꾸어 생각해 보면 사람이 일일이 그림으로 그려서 표현할 수 있는 게이트 회로의 개수의 한계가 약 5000개 정도라는 말이 된다. 따라서 보다 더 집적되고 복잡한 회로를 인간의 사고와 유사하게 알고리즘이나 함수(fuctioin) 레벨에서 제어와 설계가 가능한 그 무언가가 필요했으며, 이런 요구로 VHDL이 출현하게 되었다. 이러한 VHDL과 같은 하드웨어 기술 언어를 이용해 회로를 설계해 나가는 방식을 하향식 설계(top-down)라 한다.

VHDL은 미국 국방성을 중심으로 개발되었으며 차세대 IC의 제작을 목표로 만들어진 언어이다. 이러한 VHDL은 크게 문서화(documentation), 시뮬레이션(simulation), 합성(synthesis) 등의 용도로 주로 사용된다.

먼저 문서화에 대해 살펴보자. 어떠한 작업에서건 문서화는 반드시 필요하다. 만약 VHDL을 사용해 하드웨어를 구현했다고 한다면 이에 대한 문서화 작업이 상당히 간단해 진다. 즉 VHDL 자체가 하나의 언어이고 따라서 언어로서 이미 자체 문서화(self-documentation)적인 특징을 내포하고 있다고 볼 수 있는 것이다. 따라서 문서화에 많은 도움이 될 수 있다.

〈그림 7〉 합성 과정



다음으로는 시뮬레이션이다. 모든 프로그래밍이 마찬가지겠지만 VHDL 또한 작성한 코드가 제대로 동작하는지를 검증하는 단계가 필요하다. 이를 시뮬레이션이라고 하는데 실제 하드웨어가 제대로 동작하는가를 하드웨어 특유의 타이밍도를(붙여쓰기) 통해 동작의 정확성 여부를 검증할 수 있게 하는 단계이다.

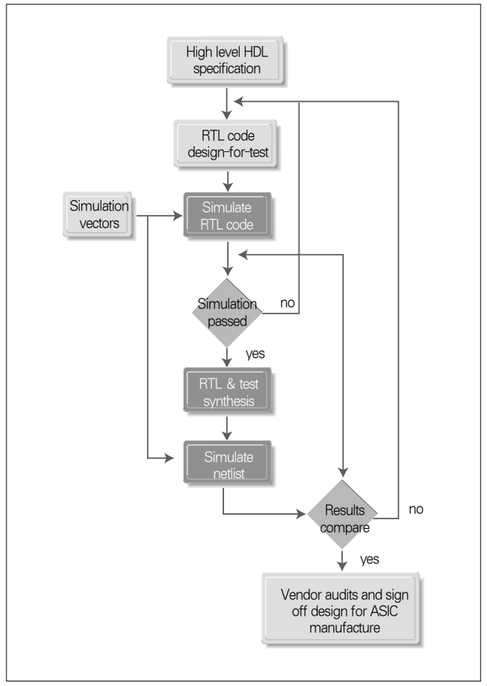
끝으로 VHDL은 합성작업까지 지원한다. 먼저 합성작업에 대해 간단히 부연 설명하겠다. VHDL은 상위 수준의 하드웨어 기술 언어이기 때문에 보다 낮은 로직 레벨로 바꾸는 과정이 필요하며 추상화된 알고리즘적 동작을 설명하는 VHDL을 게이트 혹은 레지스터 레벨로 변환하는 작업이 필요하게 된다. 이를 합성작업이라고 한다.

즉 작성한 코딩을 논리적으로 합성시키는 VHDL 툴은 VHDL로 프로그램된 논리기능을 실제 게이트 회로로 변환하는 역할을 한다. 이러한 논리합성에 의해 생성된 출력 결과물은 만들고자 하는 ASIC의 네트리스트(netlist)가 된다. 네트리스트란 단순히 회로 부품의 접속관계를 텍스트 형태로 표현한 것을 말한다. 이러한 네트리스트 파일이 VHDL을 이용한 상위 레벨의 디자인을 하위 레벨의 실질적인 부품들과 연관지어주는 다리 역할을 하는 것이다. 〈그림 7〉에 합성 과정이 나타나 있다. 참고하기 바란다.

VHDL도 나름대로 문제점이 있는데 최적화가 바로 그것이다. 즉 인간의 사고와 유사한 언어라는 매체를 사용했기 때문에 개발하는 입장에서는 편리할지 몰라도 그렇게 설계한 회로가 시스템적 차원에서는 최적화되지 못하는 경우가 많다. 이는 완전 주문형 방식의 ASIC과 비교할 때 비교적 큰 단점으로 지적된다. 물론 VHDL을 지원하는 여러 CAD 툴들이 이런 최적화 분야의 기술 개발에 박차를 가하고 있긴 하지만 직접 게이트 레벨 혹은 트랜지스터 레벨에서 설계한 회로보다는 일반적으로 VHDL을 포함한 HDL을 합성한 회로의 규모가 더 커지는 경우가 많다. 또한 이렇게 규모가 커지게 되면 생산된 칩의 소형화, 빠른 동작 속도, 적은 소비 전력적인 차원에서 문제를 일으킬 수 있으므로 지금까지도 그래왔듯이 앞으로도 많은 개선돼야 할 분야이다.

〈그림 8〉에서 전형적인 ASIC 설계 절차들이 요약되어 있다. 앞서 말한 3단계의 용도와 같이 비교해 살펴보기 바란다. 우선 절차적 형식으로 작성된 HDL 코드를 기능적 형식으로의 레지스터나 게이트 레벨의 RTL(Register Transfer Level) 코드로 변환을 수행한다. 그리고 난 후 이를 시뮬레이션 단계를 거쳐서 프로그램의 무결성을 검증한다. 그 후 합성 과정을 거쳐 이를 다시 특정 ASIC 업체에서 제공하는 라이브러리로 맵핑하는 과정을 거친다. 특히 시뮬레이션은 RTL 코드 레벨에서의 시뮬레이션이 있고, 네트리스트 레벨에 대한 시뮬레이션이 있다. 이는 단순히 하드웨어 기술 언어 레벨의 시뮬레이션이냐 아니면 특정 ASIC 업체 제품의 회로상 딜레이가 반영된 기술 종속적(technology dependant)인 시뮬레이션이냐의 차이에 따른 것이며, 전체적으로 볼 때 특성이 다른 두 번의 시뮬레이션 과정을 거치는 것이 된다. 이것이 일반적인 ASIC의 설계 절차이다.

〈그림 8〉 전형적인 ASIC 설계 절차



〈표 3〉 집적 회로 설계의 변천 과정

	제1단계(60~70년대)	제2단계(70~80년대)	제3단계(90년대~)
설계 방법	상향식 설계 트랜지스터 레벨의 레이아웃 설계	게이트나 레지스터 레벨의 논리설계	하향식 설계 알고리즘이나 기능 레벨의 설계
설계 도구	레이아웃 편집기	스키메틱 편집기	HDL과 합성
설계 범위	SSI, MSI 수천 게이트 이하	LSI, VLSI 수만 게이트 이하	VLSI 수십만 게이트 이상
설계 예	게이트, 카운터, 멀티플렉서, 가산기	마이크로 프로세서, 주변장치	고성능 MCU, DSP, 실시간 영상 처리 시스템 등

VHDL을 이용한 설계

지금까지 우리는 이러한 VHDL을 이용한 개발 방식에 대해 평가해 봤다. 그렇다면 다음으로 이러한 VHDL을 이용한 설계 절차에 대해 살펴보자. 앞서 설명한 디지털 회로의 구현 방법과 비교해 보면 FPGA/CPLD를 이용한 구현을 하드웨어 기술 언어를 통해서 한다고 보면 될 것이다. VHDL은 하드웨어의 동작을 동작적 모델(behavioral modeling)의 관점에서 구현하는 것을 가능하게 한다. 즉 이는 일반 소프트웨어의 알고리즘적 관점에서의 구현과 크게 다르지 않는 것을 나타낸다.

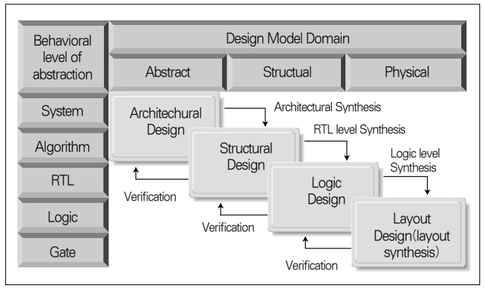
절차적 설계와 기능적 설계

하드웨어 기술 언어를 이용한 코딩 방법에는 크게 절차적 방법이 있을 수 있다. 먼저 알아 볼 기능적 설계에 있어서는 회로의 접속 관계만을 프로그래밍한다. 다시 말하면 네트리스트들 간의 접속만을 고려한다. 따라서 기능적 방법에 의한 설계시에는 일단 기본적으로 각 모듈들이 연결되어 있는 형태의 묘사에 주력한다. 간단한 예를 들어 이 두 가지 방법의 차이를 살펴보자. 가령 a, b의 두 입력을 받아서 이를 논리적 and를 수행해 그 결과를 c로 출력한다고 할 때 기능적 설계에서는 입력 a, b와 출력 c 그리고 이들을 논리적으로 and시킬 특정 AND 게이트를 사용한다. 그리고 이들의 상호 게이트 포트 간 맵핑을 통해 동작을 묘사한다. 하지만 이에 반해 절차적 방법은 단순히 일반 소프트웨어 프로그램에서와 마찬가지로 입력 a와 b를 논리적 연산 and를 수행한 후 이 결과를 c에 대입하는 형식을 띈다. 코딩상의 큰 차이점이라 할 수 있겠다.

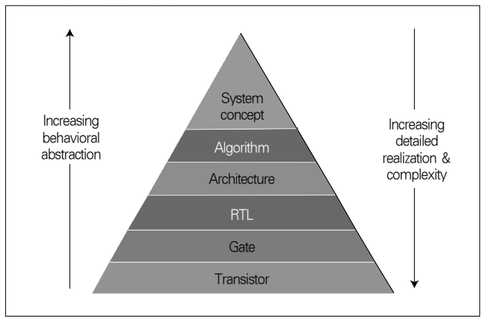
일반적으로 초기 설계시에는 절차적 방법부터 코딩을 시작한다. 이는 하향식 설계 철학과도 맞는 개념이다. 그리고 이렇게 코딩을 해야 특정 하드웨어적 기술 종속으로부터 독립할 수 있는 것이다. 그리고 이를 시뮬레이션을 통해 검증하는 작업을 거친 후 합성기가 전 처리 합성 과정에서 기능적 코드 형식인 RTL 코드로 변환시켜 준다. 이것이 가장 일반적인 코딩의 형태이다. 이렇게 크게 절차적, 기능적 설계 방식으로 구분되는 것을 요약하면 〈그림 9〉와 같다.

〈그림 9〉에서 보듯이 추상화의 단계에는 크게 5계층이 존재한다. 사실상 이의 분류에는 명확한 기준이 있는 것은 아니고 사실

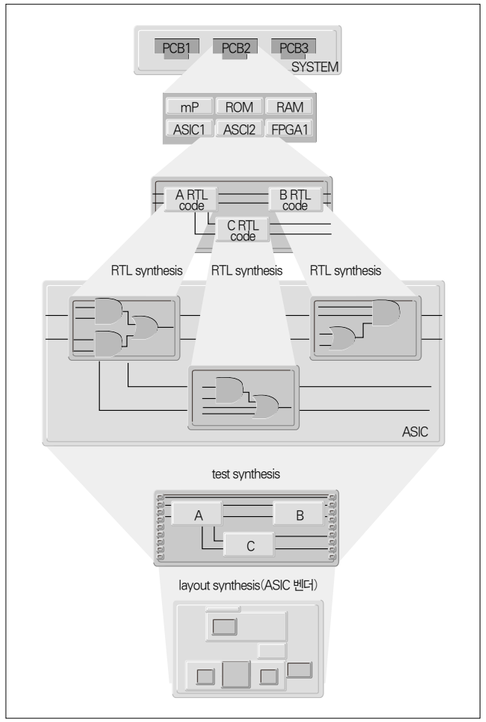
〈그림 9〉 디자인의 각 레벨별 구분



〈그림 10〉 절차적 관점에서 본 분류



〈그림 11〉 전체적인 관점에서 본 하드웨어 설계의 계층적 구분



그 경계도 애매하다. 그러나 현재 널리 사용되고 있는 용어들이기 때문에 개념적으로 이해하기 바란다. 시스템 레벨은 가장 추상화 정도가 높은 레벨이다. CPU의 파이프 라인이나 캐시 등 시스템 내의 기능을 표현한 레벨이다. 다음으로 알고리즘 레벨은 동작적 레벨이라고도 하는데 여기서는 회로의 동작이나 움직임을 표현하는 레벨이다. 회로에서 블럭이라는 개념은 현재 이 레벨에서는 존재하지 않으며, 앞서 설명한 절차적 설계 단계에 해당하는 레벨이다.

다음 단계로서 RTL 레벨은 레지스터간의 동작을 표현한 레벨로서 알고리즘 레벨에 비교해 블럭의 개념을 포함하고 있는 레벨이다. 다음으로 로직 레벨은 플립플롭이나 각종 게이트의 회로를 표현하는 레벨에 해당한다. 특히 이 레벨은 종래의 회로도를 입력하고 그리는 방식의 설계에 해당하는 레벨이기도 하다. 끝으로 스위치 레벨은 게이트 레벨이라고도 불리는데 추상화된 측면에서는 가장 그 정도가 낮은 레벨로 가장 물리적인 레벨이라 할 수 있다. 실제 MOS 회로나 트랜지스터 등의 레이아웃 설계에 관여하는 레벨이기도 하다.

〈그림 10〉에서는 또 다른 관점에서 분류하고 있다. 즉 절차적 관점에서 상호 디자인 단계별 구분을 피라미드 형식으로 하고 있다. 〈그림 9〉와 마찬가지로 크게 디자인의 단계는 시스템, 알고리즘, 아키텍처, RTL, 게이트, 트랜지스터의 단계로 나눠짐을 볼 수 있다. 상위 레벨로 올라올수록 절차적 묘사에 대한 추상화의 정도가 높아지며, 이에 반해 하위 레벨로 내려 갈수록 절차적 묘사에 대한 추상화 정도는 낮아지고 상대적으로 보다 더 물리적이고 실제적인 측면에서의 복잡도가 증가함을 알 수 있다. 또한 알고리즘과 아키텍처의 영역에 해당하는 절차적 모델링은 일반 시스템 개념과 상당히 인접해 있음을 볼 수 있다. 즉 피라미드의 상단부로 올라올수록 보다 더 동작적인 시스템 묘사가 가능하다는 말이며, 하단부로 내려갈수록 실제 구현에 관련된 로직, 회로, 레이아웃과 관련이 있다는 말이 된다.

끝으로 지금까지 논한 주제에 대한 거시적인 관점을 고려해 보자. 이에 대한 묘사가 〈그림 11〉에 나타나 있다. 어쩌면 지금까지 설명하고자 한 모든 것이 이 그림에 축약되었다고 해도 과언이 아닐 것이다. 그림의 볼록한 가운데 부분부터 살펴보자. 가운데 부분의 게이트는 우리가 원하는 동작을 수행토록 작성한 회로도를 의미한다. 그리고 우리는 이러한 회로도를 하드웨어 기술 언어라는 것을 이용해 구현하자고 하였다. 이렇게 작성된 회로도는 합성의 전 처리 과정인 RTL 합성 과정을 거쳐서 구조적 디자인의 형태인 RTL 코드로 변환된다. 이 RTL 코드는 실제 합성 단계를 거쳐서 ASIC 혹은 FPGA 형태로 구현된다. 그리고 이렇게 구현된 ASIC 혹은 FPGA들은 일반 목적용 모듈들, 즉 CPU나 메모리 등과 함께 PCB에 부착되어 구현된다. 즉 하드웨어 기술 언어에서 시작한 제작 과정이 PCB 보드의 형태로 최종 산물이 나오는 것이다. 그림의 볼록한 부분의 아랫부분 역시 각 회로의 실질적인 구현이라 할 수 있을 것이다. 즉 우리가 구현하고자 하는 게이트 혹은 레지스터들의 종류는 우리가 선택한 특정 하드웨어 업체별로 다양성을 가질 수 있을 것이고 이에 따라 약간씩 그 레이아웃이 달라질 수 있다. 그리고 각각의 게이트, 레지스터들은 실제 물리적으로 MOS 혹은 트랜지스터 회로의 형태로 구현되어 질 것이다.

독특한 사고 방식, 하드웨어 기술 언어

지금까지 우리는 디지털 시스템의 일반적인 설계 방법에 대해 같이 생각해 보았다. 그리고 소개한 여러 설계방법 중에서 하드웨어 기술 언어를 이용한 설계 방법에 대해서 VHDL을 예로 들면서 좀 더 자세히 알아보았다. 이제 독자 여러분은 이러한 여러 가지 하드웨어 설계 기법 중에서 하드웨어 기술 언어를 이용한 디지털 시스템의 설계 방법에 대해 그 윤곽을 잡을 수 있었을 것이다. 다음 호에는 실제 VHDL의 내부를 들여다 보겠다. VHDL의 문법과 프로그램 방법, 특히 일반 소프트웨어 코딩과는 다른 하드웨어 기술 언어만의 독특한 사고방식에 대해 알아보고 끝으로 예제를 통한 하드웨어 모듈의 시뮬레이션을 수행해 보도록 한다.

정리 : 위윤희 iwish@sbmedia.co.kr

참고 자료
❶ 'HDL Chip Design' A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs using VHDL or Verilog, Douglas J. Smith, Doone Publications
❷ Xilinx Foundation을 이용한 디지털 시스템 설계, 이준성 외, 복두 출판사
❸ VHDL을 이용한 CPLD/FPGA 설계, 차영배 편저, 다다미디어
❹ 디지털 회로 기술 언어 입문, 논리설계와 HDL의 기초, 정희성 외, 홍릉 과학 출판사
❺ 주문형 반도체 설계 ASIC DESIGN, 최명렬, 하이테크정보

HDL 칩 디자인

HDL의 칩 구현을 위해 시중에 나와 있는 서적들은 그리 많지 않다. 이에 'HDL 칩 디자인'을 소개하고자 한다. 책 제목에서도 알 수 있듯이 HDL을 이용한 ASIC이나 FPGA의 구현에 대해서 디자인, 합성, 시뮬레이션 등의 전 단계에 걸쳐 설명하고 있다. 그리고 특히 집필 형식이 2대 하드웨어 기술 언어라 할 수 있는 VHDL과 Verilog를 이용한 2가지 구현을 모두 다루고 있으므로 두 언어의 프로그래밍 상의 차이를 쉽게 비교할 수 있다는 점에서 이 책만이 가지는 장점이라 할 수 있을 것이다. 비교적 초보자를 위한 책이라 할 수는 없지만 어느 정도 기본을 갖춘 사용자라면 훌륭한 개발자로 거듭나기 위해서 반드시 숙지해야 할 책.

'HDL Chip Design' A Practical Guide for Designing, Synthesizing and Simulating ASICs and FPGAs using VHDL or Verilog, Douglas J. Smith