特輯 技術解說

VHDL & Synthesis

金 春 福
DAZIX, An Intergraph Company

Ⅰ. 현재의 전자 회로 설계상의 문제점

현대의 전자 회로 설계에 있어서의 가장 큰 문제는 대개 두가지 점으로 집약된다. 하나는, 급속히 증가하는 회로의 복잡도(circuit complexity)를 어떻게 처리해야 하는가 하는 문제이고, 다른 하나는 치열한 경쟁속에서 살아 남기 위하여 절대적으로 요구되는 설계 및 생산 기간(time-to-market)의 극소화를 어떻게 실현할 수 있는 가이다.

칩(chip) 하나에 백만개(million) 이상의 트랜지스터가 들어가는 일이 다반사처럼 되어 버렸고, 그러한 칩(chip)들을 무수히 사용하는 시스템이 요구되는 이제, 그리고 앞으로의 계속되는 복잡도(complexity)의 증가를 생각해 볼 때, 이 엄청난 전자회로의 복잡도(complexity)를 여하히 처리해 주는가 하는 것은, 바로 현대의 전자 기술자들이 당면한 어려운 과제이다. 복잡도(complexity)의 증가는, 필연적으로 고 수준(high-level, 또는 behavioral-level)에서의 설계를 필요로 한다.

이러한 복잡도(complexity) 문제에다가 설상가상으로, 치열한 전자업계의 경쟁은, 누가 먼저 시장에 상품(electronic product)을 내어 놓느냐(time-to-market)에 회사의 사활이 걸려 있다 해도 과언이 아니다. 선발대가 항상 이기는 것은 물론 아니지마는, 먼저 시장에 내어 놓음으로써 얻는 기득권은 절대적이다. Time-to-market을 줄이기 위해서는 완벽한 설계와 검증(verification)으로써, 상품(electronic product)의 생산(manufacturing)은 단 한번만에 성공해야 한다(first time success). 다시말해, 높은 생산성(productivity)이 요구되는 것이다.

이러한 두 가지의 문제를 해결하는데, 기존의 전통적인 디자인(design) 방법인 저수준, bottom-up 방법은 최선이 되지 못한다는 것이 차츰 명백해져 왔다. 따라서 전자 기술자들은 새로운 디자인 방법(design methodology)을 찾아 왔는 데, 많은 사람들이 소프트웨어(software), 즉 언어(hardware description language)와 합성 기술(synthesis technology)에 기초를 두는 소위 HDL-based top-down 디자인 방법에서 그러한 가능성을 찾았다.

먼저, 언어(HDL)를 사용하여 디자인을 하고자 하는 시도는 결코 새로운 것은 아니다.[1] 하지만, 기존의 수 많은 HDL들은 여러 가지의 제한(limitation)들을 가지고 있어, 보편성을 결핍하고 있었다. 그 제한들 중에는, 좁은 적용 범위, 시뮬레이터와의 의존 관계(simulator-specific), 특정한 디자인 방법과의 의존 관계(methodology-specific), 허가제(propritary), 언어 자체의 취약성 등으로, 사실상 어떤 HDL도 유니버설하게 사용된 적은 없었다. 하지만 이러한 상황은 1987년 IEEE에 의해 VHDL(VHSIC hardware description language)이 표준화 되면서 변하기 시작하였다.

또한, 합성(synthesis) 기술 역시, 입력이 무엇이냐에 따라, 오래전부터 연구되어 왔다고 볼 수 있다. 주로 RTL 레벨 또는 그 이하의 묘사를 받아들여, 주어진 조건에 따라 최적화(optimization)를 한다음, 제조(manufacturing)에 필요한 netlist로 바꾸어 ASIC 업체에 넘겨 주는 방법은 사실상 많이 사용되어 왔다. 하지만 위에서 언급된 두 가지의 문제에 대한 해결책으로서의 합성은, 주로 행위적 기술(behavioral description)에 기초를 둔 고 수준(high level)에서의 합성을 의미한다. VHDL의 행위적 수준(behavioral level)에서의 합성 기술은 많은 첨단의 전자 회로 설

계에 큰 희망을 던져 주고 있으며 앞으로의 전자 회로 설계의 방향을 결정해 주는 지침이 되어 가고 있다.

이 글이 쓰이는 현재(1991년 12월), VHDL synthesis를 통한 전자 시스템의 제작은 연구 단계를 지나 실용화 단계에 넘어와 있다. 1988년 이래, 미 국성에 납품하는 모든 ASIC chip들은 이미 VHDL로 기술되어 왔다. 또한, 상업용 전자시스템에서의 VHDL 사용 예들도 보고 되는 데, IBM의 AS/400 processor design[2], Honeywell의 Test Interface Unit (20K gates, 8000 lines of VHDL), VHSIC submicron chip set (57K gates, 100,000 lines), Rad Hard 1750 a processor chip set (3000 lines of behavioral code lines, 40,000 lines of structural code lines)[3], Arche Technology의 80386 PC system design[4], Intermetrics의 custom DSP chip design using VHDL[4] 등은 그들 중의 일부이다.

Ⅱ. VHDL의 등장

1. VHDL이란 무엇인가?

VHDL (VHSIC hardware description language) 이란 1980년대 초부터 미국방성에서 사용하기 시작한 새로운 HDL (hardware description language) 이다. VHDL의 첫 약자 (V)를 나타내는 VHSIC란 very high speed integrated circuit를 나타내는 말로써, 고속의 IC chip을 만들려는 미국방성의 첨단 계획인데, VHDL은 바로 이 계획에 사용되는 기본 HDL인 것이다. 이렇게 VHDL의 탄생은 국방에 쓰이는 전자 장비와 함께 하였다. 또한, 미국방성은 매년 많은 양의 전자 장치를 구매하는데, 해마다 달라지는 전자 기술 (technology) 때문에, 구매하는 전자 장치의 효율적인 관리와 운용에 애를 먹고 있었다. 이러한 이유로, 미국방성은 변화되는 기술에 관계없이 항상 오랫동안 사용할 수 있는 유니버살 (universal) 한 HDL를 찾고 있었는데, VHDL이 그러한 가능성을 갖고 있다고 생각하였다.

미국방성의 첨단 반도체 제조 계획과 함께 시작된 VHDL은, VHDL이 갖는 여러 진보된 특성과, 여러 HDL들이 난립하는 중에서 표준되는 것이 있어야 되겠다는 전자업계의 의견이 일치하여, 마침내 유명한 표준 HDL 후보로써 등장하게 되었다. 미국방성 뿐만 아니라 IEEE가 공식적으로 표준 HDL로 선정하였고, 게다가 미국 내의 대부분의 EDA 회사들이 VHDL을 지지하고 나섬에 따라, 이제 VHDL은 다른 여러 기존의 HDL을 대체할 위치에 이르고 있으며 이에 따라 대부분의 전자 장치 디자인 분야가 VHDL의 영향을 입게 되었다.

기존의 어떤 HDL도 현재의 VHDL이 받는 전폭적인 지지를 받은 적은 없었다. VHDL이 나오기 전에 쓰이던 HDL들 중에서 미국 전자 산업계에 가장 널리 쓰이던 것은 Verilog와 GHDL이었다는 데는 거의 반대하는 사람이 없을 것이다. 하지만 그들 조차 현재의 VHDL이 누리는 지지를 결코 받지 못했으며, 사실상 현재 많은 Verilog와 GHDL 사용자들이 VHDL로 전환할 것을 고려하고 있다. 1990년말 이래, Verilog는 그 동안의 폐쇄적인 허가제(propritary) 에서, VHDL처럼 무료로 아무나 쓸 수 있는 공개제(public domain)로 전환하여, 그 동안의 사용도를 유지하고자 노력하고 있다.

다른 하나의 흥미로운 움직임은, 일본 전자업계와 정부의 합작품인 새로운 HDL인 UDL/I (unified design language/for integrated circuit) 이다. 이것은 미국의 VHDL의 독점에 대한 반발과 (VHDL은 1986년까지 비밀로 유지되었다) VHDL의 합성 (high-level synthesis) 능력에 대한 불평에 착안하여, 일본식의 업계-학계-정부의 합작품이다. 현재까지는 미국의 EDA 업계에서 차가운 반응을 받고 있는데, 반도체 생산업계를 장악하고 있는 수많은 일본 전자 회사들의 영향력을 고려할때 앞으로의 귀추를 주목해 볼만하다.

하지만 VHDL은, 필자 개인의 생각에 의하면, 비교적 짧은 시일 내에 다른 HDL들을 누르고 표준 HDL로써 사용될 것이 분명하다고 본다. 그림 1은 VHDL의 현황을 보여 준다.

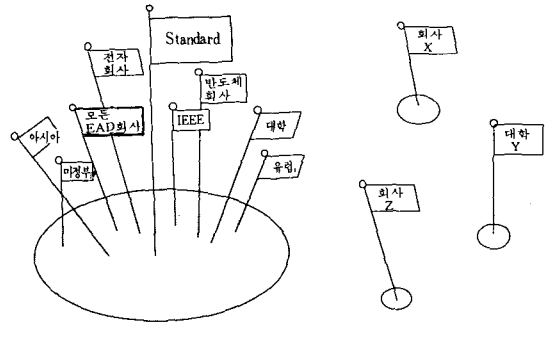


그림 1. VHDL의 현황

2. VHDL의 발전 역사

VHDL이 발전 하는데 중요했던 사항들은 다음과 같다.

(1) 미국방성은 1970년대 초부터 싹트기 시작한 VHSIC 계획 동안, 해마다 막대한 양의 전자 장비를 전자업계로부터 구매하는데, 기술이나 디자인의 변화에 관계없이, 효율적으로 관리하고 개량 발전시킬 새롭고도 강력한 HDL을 찾아오고 있었다. 기존의 HDL들은 이러한 국방성의 요구를 충족시켜주지 못하였다.

(2) 1981년 6월 Woods Hole, Massachusetts에서 국방성의 VHSIC 프로그램이 주관하여 Workshop이 열렸는데 이곳에 참석한 업계, 정부 관계자 및 학계 사람들은 현 VHDL의 윤곽을 대충 가다듬고 앞으로 이를 개량해 나갈 것을 약속하였다.

(3) Workshop이 열린 이듬해, 미공군은 DRP(draft request of proposal)을 업계에 회람시켜 VHDL의 개발을 촉구하였다.

(4) 1983년 7월 Intermetrics, IBM, TI의 합동 개발팀은 정부로부터 공식적으로 VHDL을 개발할 것을 허락받았다.

(5) 이 합동 개발 팀이 만들던 VHDL 판은 VHDL version 7.2라 불리는데, 완전한 VHDL version 7.2는 1987년 2월에 가서야 완성되었다.

(6) 한편, 1986년 3월, IEEE는 VHDL의 중요성을 인식하고, Intermetrics의 개발이 끝나기도 전에 자체에 VHDL Analysis and Standization Group (VASG)을 형성하여 VHDL을 연구, 발전시키도록 하였다.

(7) IEEE의 VASG에는 수많은 업계, 정부 및 학계가 참여하여 이미 개발 중이던 VHDL version 7.2를 수정, 확대, 발전시켜 마침내 1987년 12월, 현재 사용되는 VHDL인 VHDL IEEE Standard 1076-1987판을 완성하고 표준 HDL로 인정하였다.

(8) 이 IEEE가 만든 VHDL판(IEEE Standard 1076-1987)이 현재 공식적으로 미정부(국방성)에 의해 인정되었고, 대부분의 EDA 업계가 지지하고 나섬에 따라 표준 VHDL로 등장하게 되었다. IEEE는 Language Reference Manual(LRM)을 발간하여 이 VHDL의 사용을 적극 장려하고 있다.

(9) 현재의 IEEE Standard 1076-1987 판은 1992년 말에 가서 수정, 발전될 계획으로 되어 있다.

(10) 현재 미국방성은 모든 ASIC 부품은 납품시 VHDL로 기술할 것을 요구하고 있으며 이러한 요구는 조만간 모든 전자 부품으로 확대될 계획이다.

3. VHDL의 장점과 단점

1) VHDL의 장점

VHDL 이전에도 또 이후에도 많은 HDL들이 존재하고 있다. 무엇이 VHDL을 이런 것들과 구별해 주는 것일까? 대부분의 HDL들은 그 언어가 사용될 시뮬레이터(simulator)를 위해 만들어졌다. 다시 말하면 시뮬레이터와 결합되어(simulator-specific) 있는 것이다. 예를 들면, Verilog와 Verilog-XL이 그렇고, GHDL과 System HiLo가 그러하다. 또한 많은 HDL들은 특정한 technology, design-level 및 design methodology를 목표로 탄생되었다. 이러한 HDL들은 특정한 환경(specific environment)에서는 훌륭히 일을 하지만, 보편성을 결핍하고 있어 여러 분야에서의 사용이 제한된다.

이와는 대조적으로 VHDL은 특정한 technology, design-level, design methodology와 무관하다. 또한 어느 simulator와도 독립적이다. 이 의미는 VHDL은 전자 기술자에게 완전한 자유를 누릴 수 있도록 하는 것이다. 나중에 technology가 바뀌어도 얼마든지 새롭게 적응할 수 있는 능력을 제공하는 것이다. 물론 design-level이나 design methodolgy 또는 simulator가 바뀌어도 마찬가지이다.

VHDL의 중요한 장점들을 구체적으로 열거하면 다음과 같다.

(1) VHDL은 매우 넓은 범위의 design을 가능하게 해준다. 이 범위는 위로는 system level의 행위적 묘사(behavioral description)로부터 밑으로는 게이트 레벨(gate level)까지 포함된다. 기존의 많은 HDL들이 주로 RTL(register transfer logic) 또는 게이트 레벨(gate level)에서 일을 하는 것과는 큰 대조를 이루는 특징이다.

(2) VHDL은 특정 simulator, technology, manufacturing, process와 무관하다. 이 의미는, VHDL은 여러가지의 서로 다른 simulator, technology나 fabrication process로 구현될 수가 있다는 뜻이다. VHDL 사용자는 얼마든지 다른 technology를 선택하여 자기의 design을 구현할 수 있다.

(3) VHDL은 특정 design methodology나 design technology와 무관하다. 사용자는 자기가 원하는 대로 서로 다른 방법(e.g., top-down, bottom-up, library-based 등)들을 마음대로 사용할 수 있다. 뿐만 아니라 서로 다른 design technology들(e.g., synchronous, asynchronous, PLA, random logic 등)을 사용할 수 있다.

(4) VHDL은 미국 정부와 IEEE 그리고 대부분의 미국 EDA 업계의 지지를 받는 유일한 HDL이다. VHDL을 사용하여 만든 design은 다른 회사나 다른 design과의 호환성이 보장된다. 이 장점은 앞으로의 표준화 추세에 비추어 볼 때 매우 중요한 장점이다.

(5) VHDL은, 현대의 발전된 programming 언어에서 발견되는 진보된 개념을 많이 갖고 있다. VHDL은 많은 부분을 Ada로부터 얻어 오고 있으며, 기존의 많은 언어로부터도 여러 장점들을 취해오고 있다. 따라서 VHDL은 대규모 software engineering 계획(e.g., electronics CAE/CAD)이나 design re-use에 적합한 언어이다.

2) VHDL의 단점

모든 것이 그렇지만 현재의 VHDL에도 단점으로 지적 될만한 것들이 있다. 특히 EDA 회사에서 VHDL의 R & D 실무를 담당하고 있는 필자에게는 다음과 같은 단점들이 눈에 뜨인다.

(1) 언어 자체의 복잡성 문제

VHDL의 궁극적 목표는 전자 장치의 디자인에 도움을 주는데 있는데, VHDL을 실제 회로에 적응하여 사용키 위해서는 언어 자체의 이해와, 그것의 원활한 사용에 상당한 노력과 시간이 필요한 것이 사실이다. 현재(1991년) 미국의 경우에는 VHDL training 자체가 큰 성황을 이루는 비지니스이다. 수 권의 VHDL 서적이 미국 내에 나와 있지만 요구에 비하여는 부족한 느낌이다.

(2) Synthesis 하기에 어려운 점

이제 나중에 설명되겠지만, VHDL의 몇가지 구문들은 실제 hardware로 구현(synthesis)하기 매우 어려울 정도로 높은 레벨의 디자인 개념을 강조하고 있다. 이러한 것들은 시뮬레이션(simulation)에는 문제가 없으나, 저 수준(low-level, 예를 들어 gate-level)로 합성시에는 문제로 등장할 수가 있다.

(3) VHDL tool들의 미숙성

비록 많은 회사들이 노력하고 있지만, VHDL을 위한 software tool들이 아직은 전반적으로 미숙한 상태이다. 이것은 시간이 지남에 따라 개량되리라 본다.

(4) Logic value system들의 통일 문제

현재로서는 simulator 회사마다 서로 다른 logic value system을 이용하고 있어, VHDL 사용자들은 simulator에 따라 다른 logic value system을 이용하여 model을 하고 있는 형편이다. 이론적으로는, 한 VHDL 모델은 어떠한 VHDL 시뮬레이터에서도 같은 출력을 내야되지만, 실제적으로 각각의 시뮬레이터는 서로 다른 logic value system을 사용하는 관계로, logic value들의 mapping이 필요하고, 반드시 같은 출력이라는 보장이 없다. 앞으로는 IEEE에 의해 표준화된 logic value system의 사용이 VHDL model의 보편적인 사용을 위해 요구된다.

(5) Top-down 방식으로의 전환

전통적이고 보수적인 전자 기술자들은 대부분이 bottom-up 방식에 익숙해져 있는데 비하여, VHDL은 top-down 방식을 중요시하고 있다(물론, VHDL은 모든 방식을 다 쓸 수 있도록 허가한다). 하지만 앞으로의 전자장치 디자인이 top-down 방식에 의한 system-level 디자인으로 진전되고 있는 추세를 본다면, 이 문제도 시간이 흐름에 따라 나아지리라 본다.

Ⅲ. 합성(Synthesis) 기술의 변천

합성(synthesis)이란 일반적으로 변환(translation)과 최적화(optimization)를 하는 것을 말한다. 최초의 합성(synthesis) 기술은, 소위 로직 합성(logic synthesis)이라고 불리우는 것이었다. 이것은 주로 schematic에서 나오는 netlist, 또는 boolean equation, 또는 truth table들을 입력으로 받아들여, 최적화(optimized)되고 manufacturing 할 수 있도록 바뀌어진 netlist를 생산해 낸다. 최적화(optimization)를 하는 기준에는 area나, 또는 speed, 또는 두 개를 가능한 한 최적화 하는 것들이 있다. 일반적으로, area나 speed는 서로 반비례하므로, 두 개를 동시에 최적화 하는 일은 쉬운 일이 아니다. 로직 합성의 출력으로는 대부분의 경우 two-level 또는 multi-level로 최적화 된 combinational circuit netlist가 된다.

로직 합성보다 한 단계 더 높은 합성 기술은 보통 microarchitectural 합성이라 불리운다. 이것은 입력으로서, 위의 로직 합성의 입력들 이외에도 RTL specification이나 functional specification을 받아들여서, 출력으로는 combinational circuit, 또는 FSM (finite state machine) type이나 device allocation이 된 sequentional circuit을 만들어 낸다. 이 합성은 로직 합성에 비해 design 시간을 더 줄여 줄 수 있고, microarchitecture까지도 쉽게 design 할 수 있게 하여 준다.

하지만, 끊임없는 전자 회로의 복잡도 증가는, 더

욱 높은 수준의 합성을 요구하고 있어서 소위 architectural synthesis 또는 behavioral synthesis라고 불리우는 high-level의 합성이 등장하게 되었다. 이 합성 기술은, 이전의 여러 가지 입력들에다가 추가적으로, 보통 HDL로 묘사되는 알고리즘(algorithm) 스타일의 behavioral description을 받아들여, translation과 optimization 과정을 거쳐, manufacture(ASIC, PLD, FPGA 등) netlist 또는 이에 준하는 format으로 출력을 내어 준다. 여기에는 FSM optimization과 device allocation외에도 scheduling 및 partition 개념이 필요하게 된다. 그림 2는 여러가지 합성들을 보여 주고 있다.

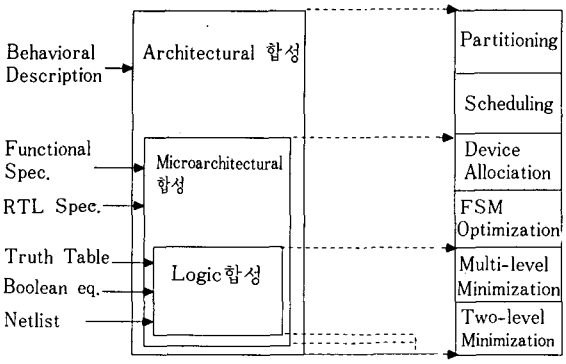


그림 2. 여러가지 합성들

Ⅳ. 고 수준(High-Level) 또는 행위적 수준 (Behavioral-Level)의 합성 단계

합성 기술(synthesis technology)에 따라서 조금씩 차이가 있지마는, 대략 다음과 같은 수순을 밟는 것이 일반적이다.

(1) HDL(예를 들어, VHDL)에 의한 행위적 수준(behavioral level) 모델

(2) Data path의 합성 : 이 부분은 보통 2단계의 스텝으로 이루어 진다

① Scheduling

② Allocation

(3) Controller and sequencer의 합성

(4) Manufacture(보통 ASIC, PLD, 또는 FPGA 형태) netlist 발생

1. HDL(예, VHDL)에 의한 행위적 수준 (behavioral level)의 모델

많은 HDL(예, VHDL, Verilog, GHDL 등등)은 행위적 수준(behavioral level)에서의 모델링에 필요한 구문들을 제공하고 있다. 이 수준에서의 모델링은 순서적 구문(sequential statement)들을 이용한 알고리즘(algorithm)의 기술에 치중한다. 예를들어 아래와 같은 process 구문은 VHDL에서의 대표적인 것으로, 한 process 문은 보통 한개의 알고리즘(algorithm)을 나타내어 준다. 연산, '*', '+'과 ':='들은 전형적인 행위적 연산이다.

```
예 : Architecture a of e is
     begin
       U2 : process
            variable v1, v2 : Integer;
          begin
          -여러가지 연속 문장들(sequential
          Statements)이 온다.
          v1 : = x1*(x2+x3);
          v2 : = (x2+x3) * (x4+x5);
          -x1, x2, x3, x4, x5 들은 어딘가에서 이
            미 정의되어 있다.
          end process;
          end a;
```

2. Scheduling

Scheduling이란 behavioral-level 모델 속에서 나타난 모든 연산(operation)들을, 그들의 나타나는 순서에 따라 형성된 서로의 의존성(dependency)를 지켜가면서 synchronous 회로의 사이클(cycle)들 속으로 순서에 맞게 배열하는 것이다. 이러한 scheduling에는 여러가지 제한조건들이 요구될 수도 있어서, 그러한 제한 조건에 따라 수많은 방법들이 개발되어 있다. 우선 앞절에 든 예(2개의 변수 할당문들)를 이용하여 scheduling을 설명하여 보자. 앞절의 2개의 변수 할당문을 CDFG(control and data flow graph)를 이용하여 그림3과 같이 표시될 수 있다. 이 CDFG에서는 원(circle)에는 연산(operation)이 배당되고 화살표는 data transfer를 의미한다. 이제 이러한 CDFG로 부터 scheduling이 시작된다. 우선 아무 제한조건이 없다면, 연산들이 나오는 순서대로 사이클에 배당하면 될 것이다. 실제의 경우에는, 대부분이 어떤 제한 조건이 가해지는 것이 보통이다. 그것들은 최소한의 사이클(다시말해, maximum speed) 또는 최소한

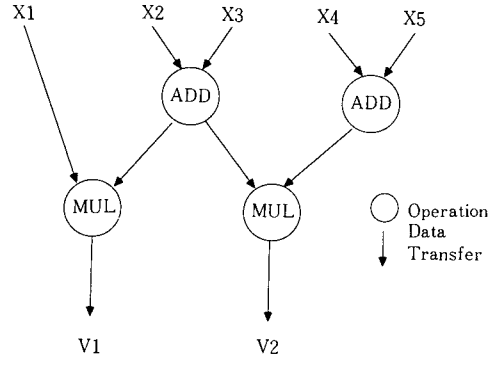


그림 3. 4장 1절에 있는 예의 CDFG

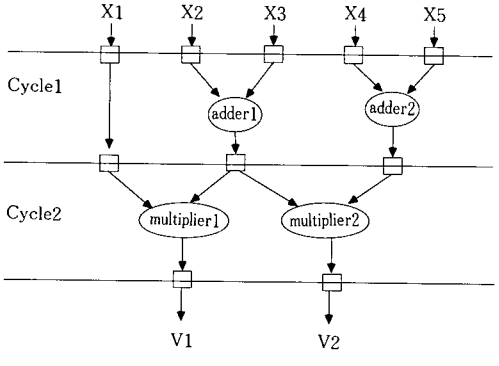


그림 4. Speed가 강조된 scheduling

의 하드웨어 모듈이다. 예를 들어 최소한의 사이클이 가장 중요한 요구라면, CDFG는 그림 4와 같이 scheduling이 될 수 있다. 이 경우에는 2개의 cycle로써 v1, v2가 가능해지지만, 2개의 adder 와 2개의 multiplier가 요구된다. 만일 하드웨어 모듈의 수를 줄여야 하는 것이 주 목적이라면, 그림 5와 같이 scheduling이 될 수가 있다. 이 경우에는 비록 3개의 cycle로써 먼저 경우보다는 1개의 cycle이 증가했지만 하드웨어 모듈은 1개의 adder와 1개의 multiplier만이 요구된다. 여러가지 조건에서의 optimized된 scheduling algorithm은 많은 연구가 행해지는 분야이다. 더 깊은 scheduling에 관하여는 reference를 참조하기 바란다.

3. Allocation

Allocation이란, 행위적 수준(behavioral-level)의 모델에서 묘사된 behavior를 structure로 바꾸는데 필요한 hardware module들을 scheduling에 의거하여 할당하는 것이다. 합성 기술에 따라 allocation이 scheduling과 동시에 진행되거나, allocation이 schedule 보다 먼저되는 수도 있다.

Allocation에서는 주로 hardware module의 수를 최소화 하는 것이 주된 목표이나, 원하는 다른 특별한 조건을 만족시킬 수도 있다. 기본적으로 allocation은 3가지 분야로 나뉘어지는데, 이는 scheduling으로 부터 온다. 예를 들어, 계속 사용하는 예인 그림 5를 보자. 이 scheduling은 아래의 3가지 allocation을 요구한다.

- Function unit(예를 들어 adder, multiplier)
- Register(입력, 출력 그리고 중간 값들의 저장 장소)
- Functional unit와 register들 간의 연결(multiplexer나 bus등)

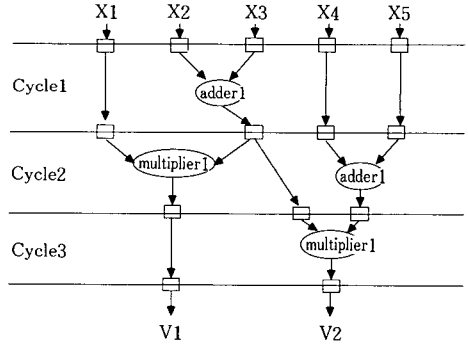


그림 5. 그림 4보다는 hardware module의 절약이 (즉, area) 강조된 scheduling

위의 요구에 따른 allocation의 한 예가 그림6에 보여 진다. 이 그림 6에서 보면, 1개의 adder와 1개의 multiplier가 사용되었고 필요한 수의 register 들이, 각각 해당하는 값들을 유지하기 위해 사용되고 있다. 이들의 연결에는 multiplexer나 bus가 사용되는 예를 보여준다. 둘의 차이점은, multiplexer가 1-to-1 연결인데 비해, bus는 1-to-n 연결을 할 수 있다. 위의 allocation은 반드시 최선의 것은 아닐 수 있지만 allocation의 한 예를 보여 준다.

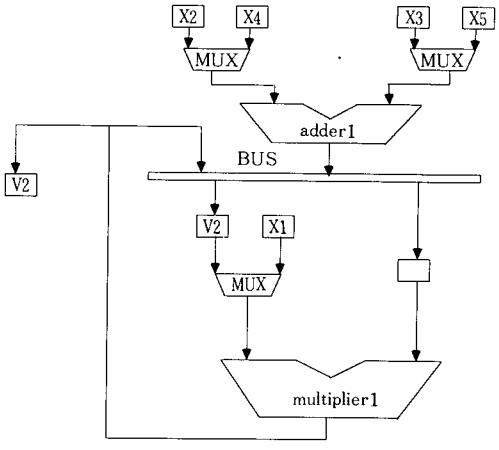


그림 6. Device allocation

4. Controller와 Sequencer의 합성

그림 6의 allocation은, 아직 controller와 sequencer 회로가 첨가되어 있지 않다. 예를들어 scheduling(그림 5)에 따라, 첫번째 cycle에는 x2와 x3가 adder 1으로 가야되고, 두번째 cycle에서는, x1과 x2.x3가 multiply되면서, 또한 동시에 adder1는 x4와 x5를 받아야 한다. 이와같이, 모든 하드웨어 모듈들에 대한 controller/sequencer가 또한 추가적으로 합성되어야 한다(예를 들어, hard-wired control 또는 microprogram control이 사용될 수 있다).

실제 합성 예(4-bit counter with load, reset, hold & scan circuit)가 그림7,8에 보여져 있다.[6]

5. Manufacturer Netlist 발생(보통 ASIC, PLD, FPGA)등

일반적으로, allocation시에 할당되는 hardware module은 technology-independent한 generic component를 사용한다(하지만, synthesizer에 따라, ASIC 또는 사용하려는 manufacturer의 synthesis library component를 직접 사용하는 경우도 있다). 많은 ASIC manufacturer들은 고유의 synthesis library를 제공하고 있고, synthesizer 회사는 그들과 제휴하여, 그들의 library에 있는 component를 가지고, 이미 만들어진 generic component를 바꾸어 주는데, 이러한 과정(module binding)을 통하여 manufacture-ready된 netlist가 생성된다. Module binding 과정중에, 또한 필요하다면 optimization(주로 area)이 수행되기도 한다.

ASIC manufacturer는 이렇게 만들어진 netlist를 받아서, 그들 나름대로의 더 낮은 단계의 합성들 (예를들어, placement and routing, layout synthesis등)을 행하고 chip fabrication에 들어가게 된다. 현재 대부분의 chip manufactring은 주로 ASIC이나 PLD, FPGA 등의 technology를 이용하여 행해지고 있다.

V. VHDL Synthesis

1. VHDL Synthesis의 문제점

VHDL의 행위적 수준(behavioral-level)의 합성(synthesis)은, 현대의 전자 기술자에게 큰 희망을 던져주지만, 동시에 문제점도 함께 나타낸다. 그 중요한 문제점중의 하나는, 합성되어진 저 수준의 회로에 대한 testability의 문제이다. 다시 말하면, synthesizer에 의하여 자동으로 만들어진 회로에 대하여 어떻게 테스트할 것이냐하는 문제이다. 디자이너가 만든 회로에 대한 테스트는, 만든 디자이너가 잘 알고 있지만, synthesizer가 자동으로 만든 회로에 대한 테스트는 간단하지가 않다. 요즈음에는 대부분의 회사들이 test-synthesizer를 개발하여 이 문제를 해결하고자 하고 있다. Test-synthesizer는 여러 기능이 있지만 그 중에는 합성된 회로에 대한 test-pattern을 자동으로 만들어 주어 테스트를 도와 준다.

두번째 문제점은 사실상 더 심각하다. 이 문제는 VHDL의 전체(full set)가 합성이 가능하지는 않다는 점이다. 다시 말해, VHDL의 구문 중에는 simulatable하지만 synthesizable하지 않는 것들이 있다. VHDL내에는 합성이 원리적으로 불가능하다고 볼 수 있는 구문들도 있고, 현재의 합성 기술로써는 어려운 것도 있는가 하면, 또한 합성에는 불필요한 구문들도 있다. 따라서, 현재 어떠한 synthesizer도 VHDL full set를 합성하는 것은 없다. 각각의 synthesizer는, 제가끔 서로 다른 VHDL synthesizable subset 를 support하는데, 이는 서로 다른 sunthesis technology를 사용하기 때문이다. 이러한 서로 다른 subset 때문에 합성에 관한 한, VHDL은 표준화 되기가 가까운 장래에는 어렵지 않을까 한다. 업계 일각에서는, IEEE가 공식적으로 synthesizable VHDL subset를 규정하여 공포하면 이같은 문제 해결에 도움이 되지 않을까 하는 의견도 있다.

또 다른 문제점은, 전자 기술자는 각 합성 회사에서

4-bit Counter
w/load, reset, hold, & scan

```
NAME      : counter
INPUTS    : Data-in, clock, reset,
            an-l, enable, updown,
            and load.
OUTPUTS   : An, q, qb, or both.
FUNCTION  : To bidirectionally count a
            binary bit in increments on
            one. Also, to enable the user
            to load and reset the bit.
            Also, to provide the necessary
            control lines for cascading the
            counters.
OPTIONS   : This single-bit counter has the
            following options.
            Up/down counter
            Positive clock
            Reset line
            Enable line
            Load line
DATE      ; February 5, 1990
```

```
entity c_ bit is
    port (
        RESET:in bit; ANminusl: in bit;
        EN : in bit; UBD: in bit;
        LD: in bit; D: in bit;
        CLK: in bit; TEST-ENABLE:in bit;
        SCAN-DATA: in bit; AN: out bit;
        Q: buffer bit; QB:   buffer bit);
end c bit;

architecture counter-rtl of c-bit is begin
    process (RESET, CLK)
      begin
        if (RESEt = '1') then
            Q < = 'O' ;
        elsif (CLK'event and CLK ='1')
        then
            if (TEST-ENABLE='1') then
                Q < =SCAN-DATA;
            elsif (LD = '1') then
                Q < = D;
            elsif (EN = '1') then
                Q < =Q;
            elsif (ANminusl = '1' ) then
                Q < =not Q;
            end if;
            end if;
    end process;
  AN(= (not Q or UBD) and (Q or not
     UBD) and ANminusl;
end counter-rtl;
```

```
NAME      : counter
INPUTS    : Data-in, clock, load
            hold, updown, and reset.
OUTPUTS   : Overflow, q, qb, or both.
FUNCTION  : To bidirectionally count a binary bit
            string in increments on one. Also, to
            enable the user to load and reset the
            bit string. Also, to provide the
            necessary control lines for cascading
            the counters.
OPTIONS   : This counter has the following options:
            Number of bits=4
            Up/down counter
            Positive clock
            Reset line
            Hold line
            Load line
            Overflow line
DATE      : February 5, 1990
```

```
entity counter4 is
    port (
        RESET: in bit; HOLD: in bit;
        UPDOWN: in bit; LOAD: in bit;
        DATAIN: in bit vector (3 downto 0);
        CLK: in bit; SCAN: in bit;
        SCAN-DATA: in bit; SCAN-OUT: out bit;
        OVERFLOW:out bit;
        Q: buffer bit-vector (3 down to 0);
        QB: buffer bit-vector (3 down to 0))
end counter4;

architecture counter-rtl of counter4 is
    component c-bit
      port (
        RESET:in bit; ANminusl: in bit;
        EN: in bit; UBD: in bit;
        LD: in bit; D: in bit:
        CLK: in bit; TEST-ENABLE; in bit;
        SCAN-DATA:in bit; AN: out bit;
        Q: buffer bit; QB: buffer bit);
    end component;
        -- synopsys translate_off
        for all:c-bit use entity work.c-bit;
        -- synopsys translate-on
            signal tog: bit-vector (3 doownto 0);
            signal vdd: bit;

begin
    vdd <='1';
    U0: c-bit port map(RESET, vdd, HOLD, UPDOWN, LOAD( DATAIN(0), CLK, SCAN-DATA, SCAN, tog (0),Q(0),
QB(0));
    U1: c-bit port map(RESET, tog(0),HOLD,UPDOWN, LOAD,DATAIN(1), CLK,Q(0), SCAN, tog(1), Q(1), QB
(1));
    U2:c-bit port map(RESET, tog(1), HOLD,UPDOWN, LOAD,DATAIN(2),CLK,Q(1),SCAN,tog(2),Q(2),QB
(2));
    U3:c-bit port map (RESET,tog(2),HOLD,UPDOWN, LOAD,DATAIN(3),CLK,Q(2),
(3));
    OVERFLOW <=tog(3);
    SCAN-OUT <=Q(3);
end counter-rtl;
```

그림 7. VHDL code

규정한 synthesizable subset 내용을 잘 알아야만 된다는 골치 아픈 문제가 있다. 현재 대부분의 VHDL simulator들이 VHDL full set를 support하기 때문에, VHDL를 사용하는 전자 기술자들은 VHDL full set를 사용하여 디자인하는 것이 보통이다. 하지만, full set VHDL 모델은 synthesizer에 의하여 받아 들여지지 않는 것이 현실이다. 따라서 synthesis-based 디자인은 실제적으로 VHDL의 subset 사용을 강요하는 것이다. 이의 해결책으로서, VHDL 모델의 합성도(synthesizability)를 자동적으로 체크해 주는 tool도 나오고 있다.

이러한 합성의 문제점은 VHDL의 탄생시의 여건을 고려해 보면 이해가 빠를 것이다. 1987년에 IEEE에 의해 VHDL이 표준화 되었을 당시에는, VHDL이 합성(synthesis)에까지 쓰이리라고는 생각되지 못하였다. VHDL의 근본 목적은 documentation과 simulation이었고, 이 두가지 목적은 훌륭하게 이룩되었다. VHDL의 폭발적인 인기는 곧 VHDL의 합성 가능성을 논의하게 만들었는데, 이에 따라 예기치 못했던 합성분야에서의 응용은 예상보다 어려움을 낳았다.

2. 합성하기가 거의 불가능하거나 매우 곤란하다고 여겨지는 VHDL 구문들

이 글이 쓰여지는 현재(1991년 12월), 아래의 VHDL

4-bit Counter w/load, reset, hold, & scan

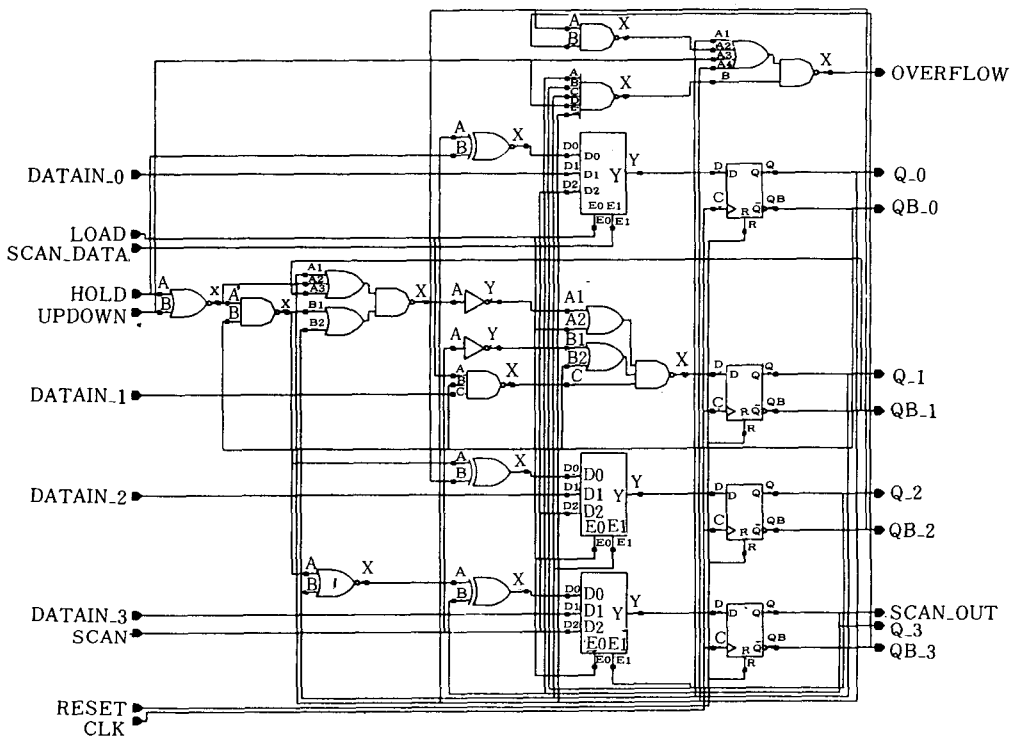


그림 8. 합성된 회로

구문들은 합성시에 error를 내거나 또는 그냥 무시된다고 여겨지는 구문들이다. 이 리스트는 일반적인 사실을 모은 것이고 특정한 synthesizer를 대상으로 한 것이 아니므로, synthesizer에 따라 사실이 아닐 수도 있음을 말해 둔다. 또한 앞으로 합성 기술의 발전에 따라, 합성 가능한 구문의 범위는 점차 더 확장되리라고 기대한다.

1. 화일(file) 입출력 관계(Input/Output) 구문들은 시뮬레이션시에는 반드시 필요하지만 합성하기는 곤란하다.
   File Type, File Declaration, Package TEXTIO
2. Physical이나 Floating Point 데이타 타입은 직접적으로 합성하기는 곤란하다.
3. VHDL은 error checking을 위해 Assertion Statement와 Concurrent Assertion Statement를 제공하는데, 이것들은 합성과는 관련이 없는 것들이다.
4. VHDL의 Advanced type들 중에서 Allocators, Access, Record 그리고 Incomplete type들은 합성하기 곤란한 것들이다.
5. VHDL의 predefine된 attributes들 중에서 아래의 것들은 simulation에서는 필요하지만, 합성에는 곤란한 것 들이다.
   Delayed(T), Quiet(T), Transaction, Active, Last-event, Last-active
6. User-defined attributes들은 일반적으로 합성되기에 곤란하다.
7. Entity 선언내의 Generic은 보통 합성되기 곤란하며, Port의 default value는 무시된다. Entity-Statement는 보통 무시된다.
8. Configuration은 보통 무시된다. 따라서 multiple architecture의 경우는 특별한 조건이 따른다.
9. Multiple-dimension 배열(array)은 현재 대부분 합성이 곤란하다.
10. Interface 선언중에서 Buffer와 Linkage을 합성이 곤란하다. 배열의 배열(array of array)은 synthesizer에 따라 합성이 가능하다.
11. Disconnection Specification은 보통 무시된다.
12. Guarded signal assignment는 현재 많은 경우에 합성이 곤란하다.

13. Transport와 After는 보통 무시된다.
14. Multiple waveform element 는 합성이 곤란하다.
15. Wait는 보통 많은 제한 조건하에서만이 합성이 가능하다. (제한 조건은 Synthesizer마다 다르다)
16. Deffered constant synthesizer에 따라 합성되지 않는 수도 있다.
17. 시그날(signal) 선언에 있어 REGISTER나 BUS 선언은 보통 합성되지 않는다.
18. Variable은 초기 값은 보통 무시된다.
19. Null slice, Null range, Null array들은 합성되기 곤란하다.
20. WHILE LOOP는 합성되지 않는 경우가 많다.
21. Synthesizer에 따라 separate compilation을 허용하지 않는 경우가 있다.
22. TIME 타입과 NOW 함수는 보통 무시된다.

Ⅵ. VHDL Synthesis Market 현황

이 글이 쓰여지는 현재(1991년 12월)에는 수 많은 VHDL synthesis tool들이 존재하고 있다. 관심있는 독자를 위하여 아래에 VHDL synthesis vendor들의 이름을 나열한다. 또한 전부터 많은 high-level synthesis에 관한 연구를 해 온 학교, 연구소들도 참고로 추가하였다. 이 리스트는 필자 개인의 소장 자료에 의한 것이므로, 주관적인 것이니 혹시 빠진 곳이 있다면 양해를 해 주시기 바란다. 이름은 알파벳 순으로 정리해 보았다.

VHDL Synthesis Tool Vendor

Cadence Design System (Verilog only, but, VHDL is expected soon)
Compass Design Automation
Dassault Electornique
DAZIX/Intergraph와 AT & T Bell Lab의 Joint Development
Exemplar Logic
Integrated Silicon System
LSI Logic Corp.
Mentor Graphics
Racal-Redac
Silc Technology
Synopsys Inc.
Teradynce Inc.
Valid Logic System
View Logic System

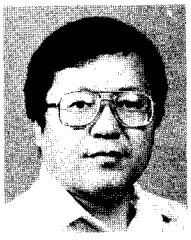
High-level Synthesis 연구 기관들

AT & T Bell Lab.
Carnegie-Mellon University
IBM T. J. Watson Research Center
Stanford University
University of Kiel
University of Karlsruhe
University of California at Irvine
University of Southern California

參 考 文 獻

[1] S. A Shiva, "Computer Hardware Description Language-A Tutorial," Proceedings of the IEEE, vol. 67, no. 12, 1979.
[2] Quentin Schmierer, "A VHDL Design Methodology used within IBM," 1989 Fall VHDL Users' Group Meeting Proceedings.
[3] Fred Rose, "VHDL in Honeywell: Setting The Pace," 1989 Fall VHDL Users' Group Meeting Proceedings.
[4] Paul Tien, "VHDL in High-End PC System Design", 1990 Fall VHDL Users' Group Meeting Proceedings.
[5] Kumar, Krishina, "VHDL Experience: Design of a Custom DSP Chip Using VHDL", 1990 Spring VHDL Users' Group Meeting Proceedings.
[6] Michael Bohm, "HDL Parametric Synthesis," 1990 Spring VHDL Users' Group Meeting Proceedings.

筆 者 紹 介



金 春 福

1953年 10月 11日生

1976年 3月 서울대학교 전자공학(학사)

1982年 12月 University of Minnesota 전기전자(석사)

1986年 12月 University of Minnesota 전기전자(박사)

1978年 3月～1980年 6月 국방과학연구소(KOREA), 연구원

1985年 9月～1989年 6月 University of Toledo, Assistant Professor

1989年 6月～1990年 6月 Intergraph Corp., Software Scientist

1990年 12月～현재 DAZIX, Software Scientist