특 집

VHDL 시뮬레이터의 설계와 구현

李 榮 熙, 黃 善 泳
西江大學校 電子工學科

I. 머 리 말

80년대 말 IC 제조 기술의 발달과 함께 수십만 개의 트랜지스터, 게이트들을 하나의 VLSI 칩에 집적하는 것이 가능하게 되었으며, 높은 집적도에 따른 복잡성의 증가는 VLSI 설계에서 가장 어려운 문제가 되어 버렸다. 게이트 수준을 근간으로 하는 설계 방법론으로서는 더 이상 대처하기 어렵게 되었고, operator, 버스, mux와 같은 데이타 전송 패스와 레지스터들이 primitive로 다루어지는 보다 더 추상적인 수준인 레지스터-트랜스퍼 수준에서의 설계가 필요하였다. 이에 따라 레지스터-트랜스퍼 수준에서 설계하고자 하는 대상 하드웨어를 표현하는 방식이 요구되었고, 이는 게이트 수준 설계에서 사용되었던 회로의 스케메틱, 진리표, 논리식 등을 표현하는 목적으로 사용되었던 형식과는 다른 것이 필요하였다.[1]

범용 프로그래밍 언어인 PASCAL이나 C에서 일부분을 취하고 하드웨어 기술에 필요한 기능—회로의 병렬성을 표현하기 위한 기능, 타이밍 기술에 필요한 기능 등 —들을 부가하여 레지스터-트랜스퍼 수준에서 설계하는 대상 하드웨어를 기술하는데 사용하였다. 이와 같이 레지스터-트랜스퍼 수준에서 설계하고자 하는 하드웨어를 기술하거나 표현하는 언어를 총칭하여 HDL(Hardware Description Language)이라고 한다.[2]

게이트 수준 설계에서는 32비트 데이타 패스를 32개의 독립된 변수로 다루는 반면 HDL을 사용한 보다 추상적인 수준에서는 이를 한 단어로 표현하여 시뮬레이터 및 다른 합성 툴에서 사용할 수 있으며, 덧셈, 곱셈, 쉬프트와 같은 추상적인 연산자를 이용할 수 있음으로 인하여, 설계자가 다루어야 할 복잡도는 매우 감소한다. HDL 자체로서는 대상 하드웨어를 표현할 수 있는 다양한 기능과 융통성을 제공하고, 특히 하드웨어를 표현함에 있어 다양한 추상 수준(abstraction level)에서의 기술이 가능하다. 상위 수준에서 기술된 설계 사양(specification)에서 출발하여 여러 단계의 합성과 재합성을 거친 후 생성된 하위 수준의 합성 결과

역시 동일한 HDL로서 표현 가능하여 설계 과정에서 일관성을 유지할 수 있다. 또한 HDL을 사용한 하드웨어의 동작 기술은 그 자체로서 설계 사양이고 도큐멘트가 된다. 상위 수준에서 HDL을 이용한 설계가 칩의 복잡도가 높아짐에 따른 설계 비용의 증가를 해결하고, 설계자의 생산성을 높일 수 있음이 알려지자 많은 상위 수준 CAD 툴들이 개발되고 실제로 사용되었다. 이러한 상위 수준 합성 시스템들은 각 툴 벤더마다 각기 고유의 HDL을 채용하여 사용하였으며, 각 HDL은 동일한 하드웨어를 기술하는데 있어서 표현할 수 있는 능력에 차이가 있고 서로 호환이 되지 않는 단점이 존재하였으므로, 특정 툴을 채용한다는 것은 곧 바로 툴 벤더에 종속된다는 것을 의미하게 되었다. 상위 수준 합성이 일반화되면서 HDL 수준에서 설계 데이타들 간의 비호환성 문제가 표출되고 널리 인식되면서 HDL의 표준화 문제가 대두되기 시작하였다.[3]

80년대 초 미 국방성은 IC 설계 시간을 단축하고 군사 시스템에 초고속 IC 기술을 도입할 목적으로 VHSIC(Very High Speed IC) 프로그램을 진행시켰으며, 이는 능률적인 설계와 도큐멘트가 가능한 공통적인 표준 대화 수단을 필요로 하였고 이것이 HDL의 개발 동기였다.[4] 1981년에 요구되는 조건들을 분석하였고, 82년에 프로그램을 진행할 조직들을 구성하고 83년부터 본격적인 개발에 착수하였으며, 87년에 IEEE의 주도로 표준 VHDL(VHSIC Hardware Description Language)인 IEEE std 1076-1987이 발표되었다. VHDL은 하드웨어 설계에 있어서 합성과 시뮬레이션을 지원하며, 하드웨어 설계 데이터의 교환과 기술 발전에 따른 설계 데이터의 재사용 및 하드웨어에 대한 도큐멘트 등을 목적으로 하고 있다. 또한 VHDL은 아키텍춰 수준에서 스위치 수준에 걸친 혼합 기술이 가능하고, 설계하고자 하는 하드웨어에 대한 정확한 모델링이 가능하도록 하드웨어의 행위, 구조, 타이밍 등과 같은 요소들을 기술할 수 있는 기능을 풍부히 제공하고 있으며, 하드웨어의 계층적인 기술(hierarchical description)과 혼합 레벨(mixed-level)의 시뮬레이션도 지원한다.[5,6,7]

미 국방성과 IEEE에서는 어떤 특정한 벤더나 기관에서 VHDL을 사용할 수 있는 배타적인 권한을 부여하지 않았고, 따라서 누구나 자유롭게 사용할 수 있으며 관련 툴을 개발할 수 있고, 미국 정부가 강력히 지원하고 IEEE에서 인정한 유일한 표준 HDL이라는 사실과 언어의 기능이 풍부하다는 사실 등으로 인하여 발표 이후 급속히 학계 및 업계의 표준으로 자리잡아 갔으며 현재도 끊임없이 관련 툴들이 연구 개발되고 있다.

Dataquest사가 1991년도에 조사한 바에 따르면 91년도 북미 전체 HDL 시장에서 각 HDL의 점유율은 VHDL이 23.4%, Verilog HDL이 27.8%였으나, 95년도에는 각각 54.8%와 27.5% 정도 점유할 것으로 예측되었다.[8] 92년도에 MJ Associates에서 조사한 바에 따르면 92년도에 VHDL과 Verilog HDL의 사용 비율이 39%와 46%이며, 앞으로 사용 예정인 HDL에 대한 조사에서는 각각 69%와 37%로 나타났다. MJ Associates는 이를 근거로 96~97년에는 VHDL이 HDL 시장의 75%, Verilog HDL이 25% 정도 점유할 것으로 예측했다. 따라서 VHDL은 HDL의 표준으로 업계 및 학계에서 자리 잡아가고 있다고 보아야 할 것이다.[9]

VHDL은 표준이 발표된 이후 광범위한 합성 및 시뮬레이션 툴들이 지원되어 VHDL를 이용하여 하드웨어를 설계한다는 것은 최소한 툴 간의 호환성이 보장된다는 것을 의미하게 되었다. VHDL은 설계하고자 하는 하드웨어를 설계 툴과 최종 회로의 구현 기술 또는 최종 칩 구현 벤더와는 상관없이 독립적으로 작업하는 것을 가능하게 하였으며, VHDL을 이용한 하드웨어 설계는 특정 벤더, 툴 및 공정과는 무관하게 설계 및 제조가 가능하다는 것을 의미한다. 그러나 현재 다양한 상용 시뮬레이터 중 전체 사양을 지원하는 툴은 그리 많지 않으며, 이로 인한 호환성 문제, 합성 툴에서 합성이 가능한 VHDL 부분 사양의 통일 문제, ASIC 라이브러리의 부재 등 여러 가지 해결하여야 할 문제점도 존재한다. 이들 문제는 현재 해결 과정에 있거나 해결하고자 하는 노력들이 보이고 있으므로 앞으로 해결될 것으로 보인다.[10]

VHDL은 개발 초기부터 여러 업체, 연구기관

및 대학이 참여하여 표준 안을 만들어 가는 동시에 이에 대한 지원 툴을 개발하였으나, 국내는 1987년도 표준 안이 발표된 이후로 연구되기 시작하여 지원 툴들의 개발 결과 및 성능이 아직은 미흡한 수준이다. 현재는 Cadence의 Leapfrog VHDL,[11] Mentor Graphics의 QuickSim II,[12] Vantage의 VHDL SpreadSheet,[13] Model Technology의 V-System,[14] 및 Synopsys의 VHDL Compiler/Simulator,[15] 그리고 응용 소프트웨어로 Comdisco의 SWP[16] 등이 상용화되어 있으며, 다른 설계 툴들도 가능하다면 입출력 결과에서 VHDL를 지원하고 있다. 국내는 서두로직에서 VHDL 시뮬레이터인 MyVHDL[17]과 합성 시스템인 MySynOpt[18] 등을 상용화하였으며, 서강대의 VHDL 분석기인 Vcom,[19] 시뮬레이터 Vsim,[20] 합성 툴인 Vsyn,[21,22] 및 VHDL을 이용한 상위 수준 합성 시스템,[23] 서울대의 VHDL 분석기 등 VHDL 관련 툴의 개발에 대한 사례가 보고되어 있다. 최근에는 시뮬레이터의 성능향상을 위하여 직접 컴파일 방식의 시뮬레이터[11,14]도 상용화되어 있으며 VHDL 설계 지원 툴의 종류도 다양해졌다. 순차 회로의 상태도를 그리면 자동으로 VHDL 코드를 생성하여 주는 시스템, 회로의 스케메틱으로부터 VHDL 코드를 생성하여 주는 툴 등과 같이 텍스트 위주의 설계 방식에서 설계자에게 보다 친숙한 스케메틱을 이용하여 설계가 가능한 시스템들이 제공되고 있다.[24]

본 고에서는 VHDL 설계 지원 툴의 하나로 연구 개발된 VHDL 시뮬레이터인 Vsim의 설계 및 개발 결과를 기술한다. Vsim은 87년도 표준 안인 IEEE std 1076-1987의 전체 사양을 지원하는 해석적 방식의 VHDL 시뮬레이터로써, VHDL 분석기인 Vcom, 라이브러리 관리기인 Vlib, 대화적인 디버거, 실시간 waveform 디스플레이어, hierarchy viewer 등으로 구성되어 있다. GUI를 제공하여 사용자가 편리하게 툴을 이용할 수 있으며, 대화적인 방식의 디버거와 실시간 waveform 디스플레이어 등을 제공하여 사용자가 하드웨어 기술상에서의 오류 발견 및 수정이 용이하도록 하였다. GUI는 VHDL 코드 분석에 필수적인 명령어들을 메뉴 형식으로 제공하고 있으며, 통합된 대화적인 디버거는 break point 설정 및 해제, step, view, set 기능 등을 제공하여 기술된 VHDL 코드의 동작을 편리하고 쉽게 검증할 수 있다.

제2장에서는 최근의 VHDL 시뮬레이터 기술 동향에 대하여 간략히 소개하고, VHDL 시뮬레이션 알고리듬을 이해하기 위한 기본적인 용어와 개념에 대하여 3장에서 다루며, 4장에서 Vsim의 전체적인 개요에 대하여 기술하고, 5장에서 Vsim의 내부 모델에 대하여 기술한다. 제6장은 Vsim의 시뮬레이션 알고리듬에 대하여 기술하고, 디버거는 7장에서 다루며, 8장에서 테스트 프로그램에 대한 Vsim의 실험 결과에 대하여 분석하고 마지막으로 결론 및 추후 과제에 대하여 논한다.

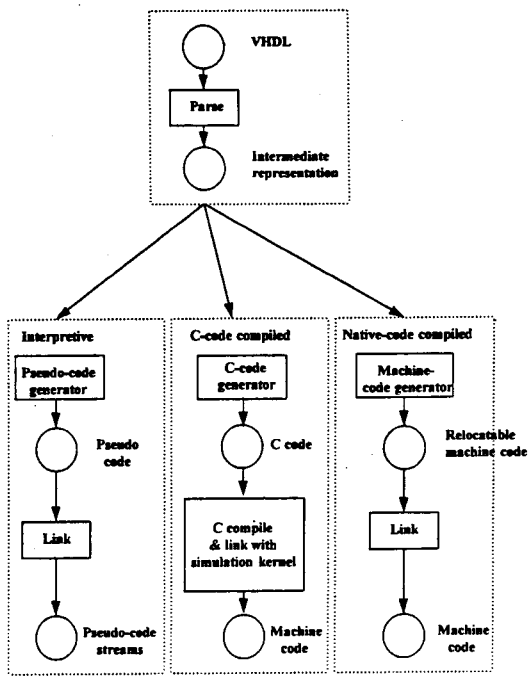
II. 최근 VHDL 시뮬레이션 기술 동향

디지탈 시뮬레이터는 구현하는 방식에 따라 일반적으로 해석적 방식(interpretive simulation)과 컴파일 방식(compiled simulation)으로 크게 구분한다. 해석적 방식의 시뮬레이터는 대상 회로를 기술한 소스 코드를 컴파일하여 중간 형식(intermediate form)으로 변환한 후, 이 중간 형식을 사용하여 시뮬레이션 엔진을 구동한다. 원리적으로 모델의 각 동작을 시뮬레이터가 해석하는 방식으로서 융통성(flexibility)과 가시성(visibility)을 모델에 포함시킬 수 있음으로 인하여 이 방식의 시뮬레이터들은 대부분 최상의 디버깅 기능을 제공하고 있다. 융통성이란 대상 모델에 타이밍, 하드웨어의 병렬성과 같은 하드웨어 시뮬레이션에 필요한 정보를 표현할 수 있음을 의미하며, 가시성이란 대상 모델에 대하여 내부 동작 및 매개값들을 시뮬레이션 과정에서 설계자가 시험하거나 변경이 가능하다는 것을 의미한다. 따라서 해석적 방식의 시뮬레이터들은 대부분 최상의 디버깅 기능을 제공하고 있으며, 이벤트 구동 시뮬레이션 알고리듬의 적용이 가능하다.[25]

이에 반하여 컴파일 방식은 소스 코드를 컴파일

하여 호스트 컴퓨터의 명령어들로 변환하고 이를 실행하는 방식으로서 호스트 컴퓨터의 명령어 집합을 사용한다. 이 방식은 빠른 시뮬레이션 속도를 제공하기는 하지만, 컴파일의 초기 단계에서부터 호스트 컴퓨터 명령어들에 의존적이기 때문에 제한적인 모델의 융통성과 가시성을 가지며, 디버깅 기능을 구현하기가 어렵다. 이 방식은 모델에 타이밍 같은 정보 표현이 어렵고, 이벤트 발생과 처리에 필요한 정보를 중간 형식에 나타낼 수 없음으로 인하여 이벤트 구동 알고리듬을 적용할 수가 없다. 주로 게이트 수준에서 로직 시뮬레이터들이 사용되던 시절에는 해석적 방식의 시뮬레이터가 충분히 효과적이고 쉽게 구현이 가능한 가장 접근 방법이었다. 게이트 수준 이상을 다루데 주로 사용되는 HDL은 컴파일 방식이 적용 가능한 부분과 해석적 방식으로만 시뮬레이션 가능한 부분들이 동시에 존재한다. 해석적 방식으로 구현된 HDL 시뮬레이터는 시뮬레이션을 수행하기 위한 setup 시간 및 결과의 반환 시간이 매우 짧기 때문에 에러가 자주 발생하는 모델의 초기 개발 과정에서 매우 유용한 반면, 설계가 진행된 최종 단계에 가까워질수록 모델 변경이 거의 없고, 에러의 발생이 줄어들어 컴파일 속도에 비하여 시뮬레이션 속도가 상대적으로 중요하게 되어 컴파일 방식의 시뮬레이터가 유용하다.[26]

해석적 방식에서의 컴파일 속도와 컴파일 방식에서의 시뮬레이션 속도의 장점을 취하고자 두 방식을 혼합한 형태로서 모델을 연속적인 라이브러리 서브 루틴 콜로 컴파일하는 방식인 "threaded-code" 방식이 존재한다.[27] 이 방식은 시뮬레이션 수행시 해석적 방식처럼 동적으로 중간 형태를 디코딩하는 오버헤드가 없으면서 이벤트 구동 알고리듬을 채용한 시뮬레이터들이 제공하는 융통성의 대부분을 제공한다. 혼합된 접근 방법은 VHDL의 의미(semantics)를 모두 지원하기 위하여 요구되는 범용성을 갖는 VHDL 컴파일러와 시뮬레이터에서 사용된다. 프로그래밍 언어 변환 방식에서 일반적으로 인터프리터가 컴파일러에 비하여 보다 많은 융통성을 제공한다는 사실을 보면, VHDL 시뮬레이터는 본질적으로 프로그래밍 언어 translation 시스템과 아주 유사하다.[28]



〈그림 1〉 시뮬레이터 구현 방식에 따른 흐름

최근의 VHDL 시뮬레이터는 원시 코드(native-code)를 사용하여 직접 컴파일(direct compilation)을 수행한 원시 코드 생성 방식으로 구현된 시뮬레이터는 기존의 C 코드 생성 방식에 비하여 속도 측면에서 매우 향상된 결과를 가져왔다. 〈그림 1〉은 원시 코드 생성 방식, C-코드 생성 방식, 해석적 방식에 대한 전체적인 흐름이다.

해석적 방식의 시뮬레이터는 VHDL을 중간 형식인 의사 코드(pseudo code)로 변환하고, 시뮬레이션 커널이 이를 해석하여 실행한다. 다른 방식에 비하여 거쳐야 하는 변환 과정이 적음으로 인해 시뮬레이션 setup 시간은 매우 빠르지만 인터프리터 프로그램이 CPU 메모리에 상주하고, 커널이 의사 코드를 가져다 해석하는데 걸리는 시간 때문에 시뮬레이션 실행 속도가 일반적으로 느리다. C-코드 생성 방식은 VHDL을 C 코드로 변환하고, 이를 호스트 컴퓨터의 C 컴파일러를 사용 컴파일하

여 오브젝트 코드를 생성한다. 두 단계의 변환 과정을 거치므로 setup 시간이 증가하지만 커널과 VHDL 코드가 컴파일되어 하나의 실행 프로그램을 구성하므로 인터프리터가 CPU 메모리에 상주할 필요가 없고, 시뮬레이션 지원 루틴 중 필요한 부분만으로 실행 프로그램이 구성되어 일반적으로 해석적 방식의 시뮬레이터에 비해 속도가 빠르다. VHDL과 C 언어는 표현력에서 차이가 존재하여, VHDL의 C 언어로의 대응이 하나의 VHDL 문이 평균적으로 7라인의 C 문장으로 번역되는 등 비효율적이고, C 언어 컴파일러는 VHDL 기술상에 존재하는 병행적인 동작에 대하여 최적화를 수행한다기 보다는 순서적인 프로그램인 C 언어 수준에서 최적화를 행한다는 점에서 볼 때, 시뮬레이터의 성능은 제약점을 가질 수밖에 없다.

원시 코드 생성 방식 즉 직접 컴파일 방식의 시뮬레이터는 호스트 컴퓨터의 컴파일러와 링커에 의존하지 않고 VHDL 코드로부터 직접 실행 가능한 파일을 생성한다. C 코드 생성 방식에 비하여 C 코드를 생성하는 중간 단계와 C 코드를 컴파일 링크하는 단계가 필요치 않으므로 해석적 방식의 시뮬레이터와 마찬가지로 짧은 setup 시간을 가진다. 상용 시뮬레이터 중 원시 코드 생성 방식의 시뮬레이터와 C 코드 생성 방식 시뮬레이터에서 약 40만 라인의 VHDL 코드를 컴파일한 결과 분석하는데 각각 1.5시간, 38.8시간이 소요되어 약 26배 정도의 차이가 보고되었다.[29] 최적화 과정은 VHDL을 호스트 컴퓨터에 매핑하는 과정에서 수행되는데 VHDL의 병행적인 수준에서 이루어지므로 효율적인 데이타 구조가 생성되어 C 코드 생성 방식보다 속도 및 메모리 요구량 측면에서 유리하다. VHDL 시뮬레이션만을 위하여 만들어진 기계어 코드 생성기는 이벤트 구동 시뮬레이션에 가장 적합하도록 최적화를 수행한다. 이 방식은 근본적으로 VHDL 언어의 최적화 컴파일러와 같은 역할을 행한다. 이 방식으로 구현한 시뮬레이터는 기존 시뮬레이터에 비하여 약 25배 정도 빠른 것으로 보고 되었다.[28]

III. VHDL 시뮬레이션의 기본적인 용어 및 개념

VHDL은 그 의미(semantics)가 시뮬레이션에 적합하도록 정의되었다. 이 장은 VHDL 시뮬레이션 알고리듬의 이해에 필요한 기초적인 용어와 개념에 대하여 기술한다. VHDL에서 제공하는 실행문(executable statement)은 크게 병행문과 순서문으로 구분할 수 있다. 병행문은 서로 간 상대적인 실행 순서가 정의되지 않고 비동기적으로 실행되는 문인 반면에, 순서문은 기술된 위치에 따라 실행 순서가 정의되는 문이다. 〈표 1〉은 VHDL에서 제공하는 실행문 중 동적인 의미를 갖고 있는 문 중에서 제어 구조를 제외한 문이다.[30]

시뮬레이션 실행 시 신호객체 s는 신호 지정문을 통하여 미래의 특정한 시간에 특정한 값을 가질 것으로 예측되며, 시간이 경과되면 경우에 따라서 신호객체의 값이 된다. 〈그림 2〉에 보인 병행 조건 신호 지정문이 실행되면 1ns후에 a 신호의 현재값, 2ns후에는 '0', 4ns 후에는 b 신호의 현재값을 갖는 것으로 예측되어 지정된다.

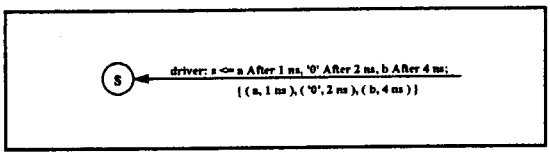
〈표 1〉 VHDL에서 동적인 의미를 가지는 실행문

병행문	• 병행 선택 신호 지정문(selected signal assignment statement) • 병행 조건 신호 지정문(conditional signal assignment statement) • 프로세스 문(process statement) • 병행 프로시져 호출문(concurrent procedure call statement) • 병행 assertion 문(concurrent assertion statement)
순서문	• wait 문(wait statement) • assertion 문(assertion statement) • 신호 지정문(signal assignment statement) • 변수 지정문(variable assignment statement) • 프로시져 호출문(procedure call statement)

```
Library Ieee;
Use Ieee.std_logic_1164.all;
Entity   signal_test Is
    Port( a: In std_logic;
          b: In std_logic;
          s: Out std_logic ) );
End signal_test;

Architecture signal_test_a Of signal_test Is
  Begin
    s <= a After 1 ns, '0' After 2 ns, b After 4 ns;
  End signal_test_a;
```

(a)



(b)

〈그림 2〉 신호 지정문과 구동원
(a) 신호 지정문의 예
(b) 생성된 구동원

신호 지정문의 실행 시 지정되는 값들은 시간적인 지연을 가지고 일어나므로, 구동원(driver)의 값은 언제 어떠한 값으로 신호 s을 구동하는가를 나타내는(value, time)의 쌍인 트랜잭션의 집합으로 나타낼 수 있다. 구동원의 waveform은 연속된 하나 이상의 트랜잭션들로 정의되며, 〈그림 2〉의 (a)에 있는 병행 신호 지정문은 〈그림 2〉의 (b)와 같이 신호 s를 구동하는 waveform으로 표현할 수 있다.[31] 구동원은 waveform이 생성되는 특정 신호 지정문과 〈그림 2〉의 (b)와 같이 결합(association)되어 있다. 신호에 대한 구동원은 하나 이상 존재할 수 있으며, 신호 지정문의 실행 효과는 타겟 신호에 직접적인 영향을 주는 것이 아니고, 해당 구동원의 현재와 미래의 값을 나타내는 projected output waveform에만 영향을 미친다. 시뮬레이션 시간 t_c=0ns에서 〈그림 2〉의 (b)에서 보인 구동원의 projected output waveform이 {('0', 0ns)}이고, 〈그림 2〉의 (a)의 병행 신호 지정문이 실행되면 waveform{(t_c=0ns에서 a의 값, 1ns), ('0', 2ns), (t_c=0ns에서 b의 값, 4ns)}이 projected output waveform에 새로이 추가되어 projected output waveform은 {('0', 0ns), (t_c=0ns에서 a의 값, 1ns), ('0', 2ns), (t_c=0ns에서 b의 값, 4ns)}이 된다. 시뮬레이션이 진행되어 t_c=1ns에서 〈그림 2〉의 (a)의 병행 신호 지정문이 실행될 때, projected output waveform은 t_c=1ns에서 a의 값이 '0'이고 b의 값이 변화하였다면 {(t_c=0ns에서 a의 값, 1ns), ('0', 2ns), (t_c=0ns에서 b의 값, 4ns), (t_c=1ns에서 b의 값, 5ns)}이 된다. 만약 t_c=1ns에서 a의 값이 '1'이라면, projected output waveform은 {(t_c=0ns에서 a의 값, 1ns), (t_c=1ns에서 a의 값, 2ns), ('0', 3ns), (t_c=1ns에서 b의 값, 5ns)}이 된다. 구동원은 현 시뮬레이션 시간보다 크지 않은 시간값을 갖는 하나의 트랜잭션을 가지고 있으며 이 트랜잭션의 값 성분이 해당 구동원의 현재값(current value)이다. 〈표 1〉에 제시한 병행문은 실행 시 동일한 효과를 갖는 동등한 프로세스 문(equivalent process statement)이 존재하며, 동적인 의미를 가지는 모든 병행문은 동등한 프로세스문으로 변환 가능하다.[32] 프로세스 문 내에 존재하는 신호 지정문은 지정되어지는 신호인 타겟 신호객체에 대한 구동원을 정의한다. 만약 같은 프로세스 문 내에서 특정 신호객체를 타겟 신호로 하는 신호 지정문이 하나 이상 존재한다면, 이들 신호 지정문에 의하여 정의되는 구동원은 하나로 간주한다. 신호의 소스는 구동원(구동원 소스)이나 컴포넌트 instantiation문에 의하여 신호와 결합되어 있는 포트 중의 하나로서 정의되며(포트 소스) 신호객체의 값에 직접적으로 기여한다. 신호객체는 복수개의 소스를 가질 수 있으며, 복수개의 소스를 가진 신호객체는 반드시 resolved 신호객체이어야 한다. Resolved 신호객체는 resolution 함수와 결합된 신호로 정의되고, 신호객체의 선언이나 신호객체의 선언에 사용된 subtype 선언 과정에서 resolution 함수를 명시함으로써 resolution 함수와 결합된다. Resolution 함수는 결합된 resolved 신호객체 타입을 갖는 1차원 unconstrained 배열을 입력 파라메타로 가진다. 시뮬레이터는 resolved

신호의 각 소스를 관별하고, 주어진 시뮬레이션 싸이클 동안에 하나 이상의 소스에 값이 변화하면 resolution 함수를 호출하여 실행한다. 이 때 resolution 함수에 의하여 되돌려지는 결과값이 resolved 값이 된다.[33]

시뮬레이션 시간이 증가되면, 주어진 구동원의 projected output waveform에 존재하는 트랜잭션들은 연속적으로 구동원의 값이 된다. 이와 같은 방식으로 구동원이 새로운 값을 가지게 될 때, 새로운 값이 이전 값과 같거나 혹은 같지 않거나에 상관없이 이 구동원은 그 시뮬레이션 싸이클 동안 활성화되었다고 한다. 신호는 신호의 소스 중 어느 하나가 활성화되었거나, 형식 포트이고 결합된 실제 파라메타가 활성화된 경우에 주어진 시뮬레이션 싸이클 동안 활성화된다.[34]

신호의 구동값(driving value)은 이 신호가 다른 신호의 소스로서 제공되는 값이며, 신호의 유효값(effective value)은 표현식 내에 신호가 포함되어 참조될 때의 신호의 값이다. 신호의 구동값과 유효값은 신호값의 전달 과정에서 타입 변환 함수(type conversion function)나 resolution 함수가 포함되었을 때처럼 늘 같은 값은 아니다. 주어진 신호 s가 구동원 혹은 포트 소스로서 제공하는 값인 구동값은 다음과 같은 방식으로 결정된다.

- 소스가 없는 신호 s의 구동값은 신호 s의 default값이다.
- 신호 s가 하나의 구동원 소스만을 가지며 resolved 신호가 아니면, 구동원 소스 값이 신호 s의 구동값이다.
- 신호 s가 하나의 포트 소스만을 가지며 resolved 신호가 아니면, 포트 소스 값이 신호 s의 구동값이다.
- 신호 s가 resolved 신호이면, resolved 값이 신호 s의 구동값이다.

신호 s의 유효값은 다음과 같은 방식으로 결정된다

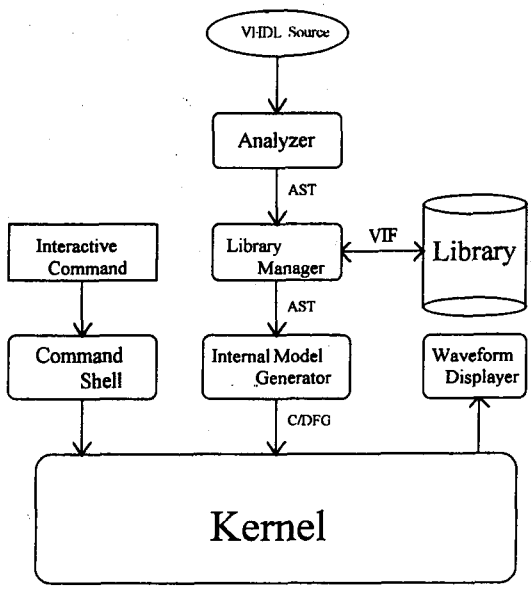
- 신호 s가 신호객체 선언에서 선언되었거나, buffer 모드의 포트 혹은 inout 모드를 갖고 연결되지 않은 포트인 경우에는 구동값이 신호 s의 유효값이 된다.
- 신호 s가 in, inout 모드를 갖는 연결된 포트이면 actual 부분의 유효값이 신호 s의 유효값이다.
- 신호 s가 모드 in의 연결되지 않은 포트이면, 유효값은 신호 s의 default 값이다.

주어진 시뮬레이션 싸이클 동안 신호 값을 갱신하기 위하여 시뮬레이터는 먼저 신호의 구동값을 결정하고 나서 유효값을 결정한다. 유효값을 신호의 현재 값으로 지정하는 과정은 이 값이 신호의 subtype에서 정의된 범위 안에 포함되는 값인지를 검사하는 과정이 포함된다. 신호의 갱신으로 인하여 신호의 현재값이 변화되면 그 신호객체에 이벤트가 발생하였다고 한다.

IV. Vsim(VHDL SIMulator)의 개요

〈그림 3〉은 전체적인 Vsim의 개요이다.

분석기(analyzer)는 VHDL 원시 코드를 분석하여 구문 및 의미 에러를 검출하여 이를 수정할 수 있도록 하며, 에러가 없을 시에는 생성된 데이터를



〈그림 3〉 Vsim의 전체적인 구성

라이브러리 관리기를 통하여 라이브러리에 VIF (VHDL Intermediate Format) 형식으로 저장한다. 내부모델 생성은 VIF로부터 Vsim의 내부모델을 생성하는 과정으로서, 이 과정에서는 시뮬레이터의 내부모델을 생성하고 시뮬레이션 과정에서 필요한 제반 정보를 분석 추출하며 이를 바탕으로 시뮬레이션 속도 향상을 위한 변환 과정이 수행된다. 내부모델은 계층적인 C/DFG(Control/Data Flow Graph)로서 데이터와 제어의 흐름을 나타내는 C/DFG에 계층적인 정보와 소속 정보를 부가한 형태이다. 내부모델 생성 과정에서 각 신호객체에 대한 모든 구동원을 판별하여 확정하고, 복수개의 구동원이 존재할 경우 resolution 함수와 이의 actual 파라메타를 생성하여 주며, enumeration 데이타 타입에서 enumeration literal은 위치에 따른 정수값으로 대치시킨다. VHDL 원시 코드 분석 과정에서 계산 가능한 정적 표현식(static expression)은 표현식의 값을 평가하여 결과값을 갖는 상수로 대치시키며, 속한 컴포넌트 안에서 참조되지 않고 단지 상위 및 하위컴포넌트 사이에서의 값 전달이 주목적으로 사용되어지는 IN, OUT 모드의 포트 신호를 구분하고 시뮬레이션 과정에서 한 신호로 취급하여 시뮬레이션 속도를 향상시킨다. 계층에 대한 정보 및 소속 정보는 시뮬레이션 과정에서 발생한 이벤트의 전달 과정에 이용된다. 또한 내부모델 생성 과정에서 사용자가 시뮬레이션을 대화적인 방식으로 수행할 것인지, 배취모드 방식으로 수행할 것인지를 결정함에 따라 디버깅 정보의 부가 여부를 결정한다. 대화적인 명령어들은 시뮬레이션 과정 중에 사용자가 필요한 명령어들이 제공되어 사용자가 쉽게 사용할 수 있도록 하였다. 시뮬레이션 결과 확인은 대화적인 시뮬레이션의 경우에 모니터하는 신호값의 변화가 즉시 waveform 디스플레이어에 반영되어 확인할 수 있는 형태이며, 비대화적인 시뮬레이션은 출력화일을 지정한 후에 waveform 디스플레이어를 이용하여 확인이 가능하도록 하였다. Waveform 디스플레이어는 줌 기능과 radix 변환 기능이 제공되고 마우스를 이용한 waveform의 위치의 변경이 가능하여 출력값 변화의 확인과 비교가 용이하다.

V. 내부모델(Internal Model)

Vsim은 내부모델로 C/DFG에 계층성과 소속 정보를 추가한 계층적인 C/DFG을 사용한다. 계층적인 C/DFG는 행위 및 구조적인 기술을 지원하며 사용자 정의 데이터 타입(user-defined data type) 또한 지원한다. 계층적인 C/DFG는 크게 노드와 간선으로 구성된다. 노드에는 객체노드(object node), 연산노드(operation node), 지정노드(assignment node), 구문노드(statement node), 구조노드(structure node) 등이 있다. 객체노드는 VHDL에서 정의되는 객체를 표현하기 위한 노드로서 C/DFG상에 객체노드가 참조될 때마다 참조된 장소와 형식에 관련된 정보를 나타내는 데이타 구조를 생성하여 준다. 연산노드는 기정의 연산자를 표현하기 위한 노드이다. 각각의 연산자는 연산결과의 데이타 타입과 값 등을 표현하는 필드를 속성으로 가진다. 지정노드는 신호 및 변수객체에 지정이 일어나는 사실을 표현해 주는 노드로서 변수객체와 신호객체 지정노드가 있다. 구문노드는 VHDL에서 제공하는 각각의 구문을 나타내는 노드로서 구문의 종류에 따라 각기 다른 속성을 가진다. 구문노드의 속성은 구문의 의미에 따라서 여러 가지로 구분되나 공통적으로 시뮬레이션 과정에서 구문이 실행되는 조건 판단에 필요한 정보, 구문의 의미에 관련된 정보 등을 가지고 있다. 구조노드는 각 컴포넌트 간의 연결 관계를 C/DFG 상에 표현하는 노드로서 포트 간의 연결 관계, 바인딩된 entity-architecture의 C/DFG, 여러 구조노드가 entity-architecture의 C/DFG을 공유 시 노드 값을 구분하기 위한 구분자 등을 속성으로 한다. 간선은 제어와 데이타의 흐름을 나타낸다. 각각의 노드에는 이벤트 발생에 영향을 받는 구문을 명확히 하기 위한 데이타 구조가 속성으로 존재하며 전역적 및 지역적인 제어 정보가 부가되어 있다. 전역적인 제어 정보는 특정 구문의 수행이 이 구문을 포함하는 상위 구문이 수행될 경우에 한하여 수행된다는 사실을 표현하는 제어 정보로서, 구문에 부가되어 있는 정보 중 구문이 실행될 수 있는 조건이 충족되

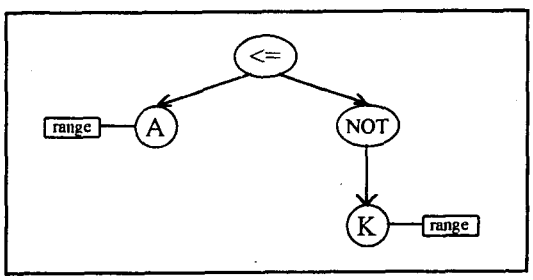
더라도 상위 구문의 활성화 여부를 판단하여 실행 여부를 가리는데 이용된다. 지역적인 제어 정보는 구문의 의미상 평가 시에 실행되는 부분이 서로 다를 경우 어느 부분을 실행할 것인지를 결정하는데 이용되는 정보들이다.

1. 객체노드

객체노드는 VHDL이 지원하는 객체를 표현하는 노드이다. 객체는 크게 신호, 변수, 상수가 있으며 객체의 데이터 타입에 따라 파일, access, 스칼라 및 복합(composite)객체로 구분한다. 신호객체는 새로운 값의 지정이 시간적인 지연을 가지고 발생하기 때문에 이를 표현하기 위하여 트랙잭션 큐를 가지고 있으며, 배열 타입의 복합 객체는 각각의 객체 노드에 포함된 원소에 대한 정보를 나타내는 range 노드를 포함한다. Access 객체는 allocator에 의하여 생성되는 객체를 가리키는 타입을 가지는 객체로서 C 언어의 포인터 타입과 같은 기능을 갖으며, pointing하는 내용이 배열 객체일 경우에는 range 노드를 가지고 있을 수도 있다.

```
Signal A: std_logic_vector( 7 downto 0);
    ••••
Variable K: std_logic_vector( 15 downto 0);
    ••••
A <= Not K( 15 downto 8 );
```

(a)



(b)

〈그림 4〉 VHDL 문장과 C/DFG
(a) 신호 지정문
(b) C/DFG 표현

객체노드는 C/DFG상에서 참조되는 곳마다 생성시켜 주며 노드의 속성을 나타내는 한 필드에 의하여 서로 구분된다. 〈그림 4〉는 VHDL 구문과 이에 해당하는 C/DFG를 보이고 있다.

〈그림 4〉 (a)는 신호지정문의 예이며 〈그림 4〉 (b)는 지정문에 대한 C/DFG의 예이다. 각각의 객체 노드에 포함된 배열 원소의 범위를 나타내는 range 정보가 속성으로서 표현되어 있다. Range 정보는 배열 타입의 객체 노드를 참조할 때 포함되는 원소의 범위를 지정하는 기능을 수행한다. 일반적으로 객체가 배열 타입을 가질 경우 각각의 range는 고정된 상수 값이 아니라 변수 값으로 주어질 수 있으며 참조될 때마다 range 값이 계산된다.

2. 지정노드

지정노드는 단순히 연산된 결과가 타겟에 지정된다는 사실을 표현하는 노드이다. 지정노드의 종류는 연산의 결과가 지정되는 대상의 객체 클래스에 따라서 신호 지정노드, 변수 지정노드로 분류되며 신호 지정노드는 병행 신호 지정노드 및 순차 신호 지정노드로 분류된다. 병행 신호 지정노드는 신호 지정에 대한 조건으로서 guarded 표현식이 있을 경우 이를 표현하도록 되어 있다.

3. 연산노드

연산노드는 VHDL에서 제공하는 기정의 연산자를 표현하기 위한 노드이다. 연산 노드는 연산자의 종류에 따라서 각각의 타입으로 분류되며 속성으로서 값을 가지고 있다. 값은 연산의 결과를 저장하기 위한 것이며, 연산결과 타입이 배열일 경우는 이에 따른 range 값을 저장하여 다음 연산에서 참조한다.

4. 구문노드

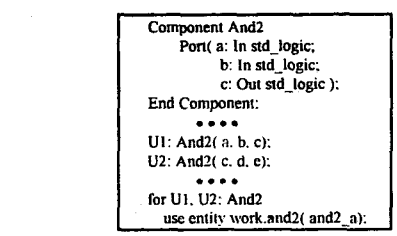
구문노드는 병행문과 순차문을 표현하기 위한 구조이다. 병행문은 블럭, 프로세스, 병행 신호지정문, 병행 assertion문, 병행 프로시져 호출문 등이 존재한다. 시뮬레이션 과정은 신호객체에 이벤트가 발생하면 이에 따라 활성화되는 구문노드들

을 수행하게 된다. 따라서 이벤트가 발생한 신호객체의 소속정보로부터 활성화 여부를 평가할 구문노드를 확정할 수 있어야 한다. 이를 위하여 구문노드는 이벤트가 발생하면 실행여부를 판단하여야 하는 신호의 집합을 속성으로 가지고 있으며, 순서문에 해당하는 구문노드는 디버깅 정보가 속성으로 존재한다.

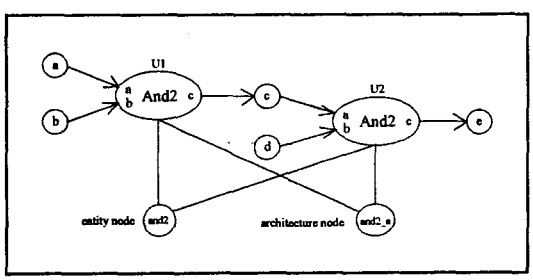
5. 구조노드

구조노드는 컴포넌트들 간의 연결관계를 나타내기 위한 노드이다. 구조노드는 컴포넌트 instantiation문과 generate문의 실행결과 생성되는 노드이다. 구조노드는 바인딩된 entity-architecture 쌍에 대한 외부적인 연결관계를 표현해 주기 위한 노드이고 속성으로서 entity-architecture 내의 구문노드를 가지고 있다. 구조노드는 시뮬레이션 수행 시 각각의 시뮬레이션 싸이클마다 인터페이스 신호객체의 값을 갱신하고 전달하는 과정에서 유용하게 사용된다. 〈그림 5〉는 VHDL 문의 일부와 이에 해당하는 구조노드의 예이다.

〈그림 5〉 (a)의 예에서 'And2' 컴포넌트는 U1, U2라는 라벨을 가진 컴포넌트 instantiation문에 의하여 instantiation이 되어 있다. 〈그림 5〉 (b)에서는 각각의 컴포넌트가 구조노드로 표현되어 있으며 이 구조노드가 표현하고 있는 것은 컴포넌트의 연결구조에 대한 정보를 표현하고 있다. 이를 이용하여 포트들 간의 연결관계를 분석, 시뮬레이션 속도 향상을 위한 변환이 가능하다면 변환한다. Vsim은 컴포넌트가 복수 개로 instantiation되었을 경우에 이를 flatten하지 않고 공유하여 사용한다. 즉, 동일한 entity-architecture 쌍으로 바인딩된 컴포넌트는 C/DFG을 공유하여 사용하고 각 노드의 상태를 나타내는 속성들을 복수 개로 하여 사용함으로써 공간복잡도를 줄이고 있다. 〈그림 5〉 (b)에서와 같이 entity 'and2'와 architecture 'and2—a'을 컴포넌트 U1과 U2가 같이 공유하여 사용한다. 바인딩된 entity-architecture의 C/DFG를 여러 컴포넌트가 공유하여 사용함으로써 데이타와 제어의 흐름에 대한 정보는 공유하나 실행 시 각 노드의 값은 서로 분리하여 저장하고 사용할 필요가 있다. 이를 위하여 바인딩된 컴포넌트에 따른 구분자를 이용한다. 각 노드는 속한 entity-architecture 쌍이 바인딩된 횟수에 따라 값을 저장하는 구조를 메모리 상에서 연속적인 주소로 할당받고 컴포넌트에 따른 구분자를 이용하여 값을 저장한 주소를 참조하는 형식으로 값을 관리한다.



(a)



(b)

〈그림 5〉 구조노드의 예
(a) VHDL 컴포넌트 instantiation문
(b) 그래프 표현

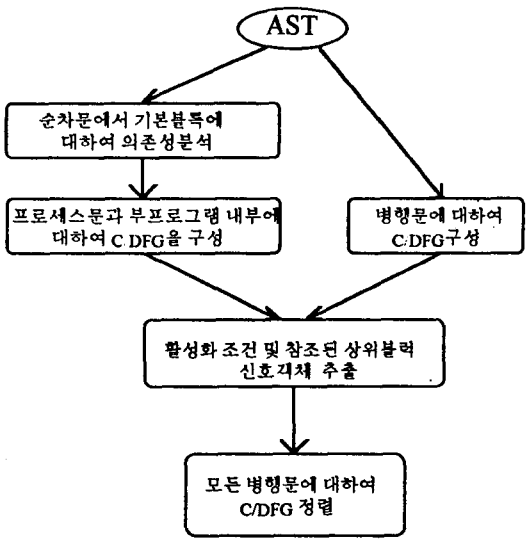
6. 제어노드

제어노드에는 fork, sfork, loop, next, exit 및 join노드가 있다. Fork 노드는 if문과 같은 분기가 둘인 경우에 사용하며, sfork 노드는 선택 신호 지정문과 case문과 같이 다중분기인 경우에 사용한다. Join 노드는 fork나 sfork에 의하여 분기된 흐름이 다시 합쳐지는 경우에 사용하는 노드이다. Loop노드는 loop문에서의 제어의 흐름을 나타내기 위한 노드로서 속성으로 loop의 반복조건을 나타내는 필드와 loop의 시작 및 끝 노드를 가지고 있다. 이외의 제어노드로서 loop 노드 내의 next, exit 노드가 있으며 이들 노드는 속성으로서 해당 loop 노드를 가지고 있다.

7. 계층적인 C/DFG 구현 알고리듬

계층적인 C/DFG 구현 알고리듬의 전체적인 흐름은 〈그림 6〉과 같다.

VHDL 원시코드의 분석 결과 AST가 생성되면 입력을 AST로 하여 순차문에 대하여 기본블럭으로 나눈다. 각 기본 블럭 내에서 의존성 분석을 행한 다음 순차문을 내부에 body로서 포함하는 프로세스 문과 subprogram body내에 대하여 C/DFG을 구성한다. 프로세스 문과 병행 프로시져 호출문을 제외한 병행문은 각각에 대하여 기본블럭을 나누지 않고 곧바로 C/DFG을 구성한다. 구성된 C/DFG로부터 각각의 구문노드에 대하여 활성화 조건집합, 활성이 의존적인 구문노드, 독립된 활성조건을 갖는 구문노드 및 참조된 상위블럭 신호객체 집합을 추출하여 구문노드 속성을 채운다. 마지막으로 모든 병행문 내의 C/DFG을 구문노드의 속성에 따라서 분류한다.



〈그림 6〉 계층적인 C/DFG구현 알고리듬

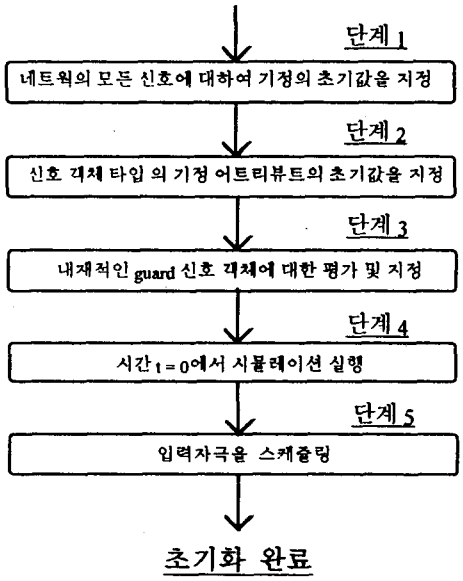
VI. 시뮬레이션 알고리듬

Vsim은 해석적 방식의 시뮬레이터로서 대상 하드웨어에 대한 내부모델을 시뮬레이션과 합성에 필요한 정보를 담고 있는 계층적인 C/DFG를 사용하고, 이를 이용하여 이벤트 구동 알고리듬을 적용하였다. Vsim은 입력 네트웍에 대하여 각 노드의 초기값을 결정하고 행위적 병행문을 실행한 다음, 입력 자극을 네트웍에 연결시키는 초기화 단계와 시뮬레이션을 실행하는 실행단계를 거쳐 시뮬레이션 결과를 출력한다.

1. 초기화 단계

초기화 단계는 시뮬레이션 대상 네트웍의 노드에 대하여 초기값을 결정하는 과정이다. 노드의 기정의 초기값 지정 이후에 초기화 실행과정을 거치게 되며 실행결과에 따라 초기값이 결정된다. 네트웍 노드의 초기값 결정은 〈그림 7〉에 보인바와 같이 5단계를 거쳐서 실행된다.

초기화 단계의 첫 단계는 네트웍의 모든 신호객체의 초기값을 결정해 주는 단계이다. 이 단계에서는 객체의 초기값을 결정하여 지정하는 단계로서, 객체의 초기값은 객체의 선언 시 초기 지정값이 명시되어 있으면 이 값을 객체의 초기값으로 지정한다. 명확히 명시되어 있지 않으면 타입 T을 가지

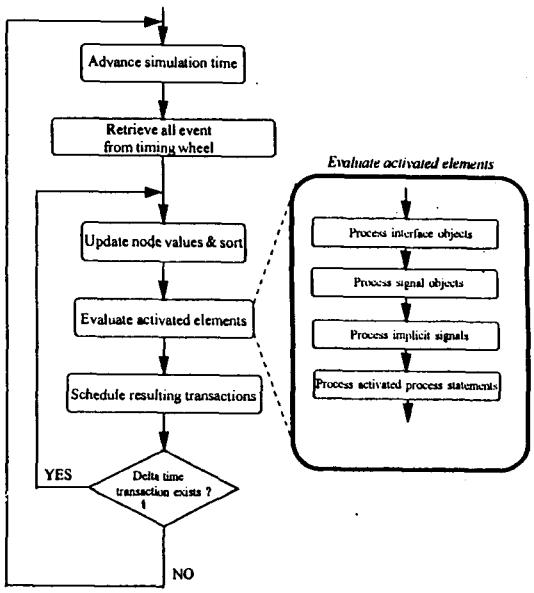


〈그림 7〉 초기화 알고리듬

는 객체에 대하여 타입 T에 속하는 값 중에 맨 왼쪽 값인 T'LEFT로 지정한다. 제2단계에서는 신호객체 타입의 기정 어트리뷰트(predefined attribute)의 초기값을 결정한다. S'QUIET, S'STABLE은 초기값으로 BOOLEAN 타입의 TRUE값을 지정하며 S'TRANSACTION은 초기값으로 FALSE값을 지정한다. 단계 3은 블럭문의 내재적 guard 신호(implicit guard signal)에 대하여 단계 1과 2에서 지정된 초기값을 가지고 guarded 표현식을 수행한 다음 결과값을 내재적 guard 신호의 초기값으로 지정한다. 내재적 guard 신호는 guarded 표현식이 존재하는 블럭문에서 암묵적으로 선언된 신호객체로 주어진 시뮬레이션 싸이클에서 guarded 표현식의 값을 반영하는 신호객체이다.[10] 단계 4에서는 위 단계를 거치면서 지정된 객체들의 초기값을 가지고, 주어진 네트웍에 대한 시뮬레이션을 실행한다. 시간 t=0에서의 실행은 먼저 포트와 병행 프로시져 호출문의 파라메타 중 신호객체들에 지정된 초기값을 가지고, 서로 연결관계에 있는 신호값들 간의 일관성을 유지시키기 위한 과정을 수행한다. 연결관계에 있는 신호들의 초기값의 일관성을 유지시키기 위한 과정을 거치게 되면서 포트와 병행 프로시져 호출문에 존재하는 신호타입의 파라메타의 구동값과 유효값이 결정된다. 단계 4의 실행은 프로세스 문을 포함한 모든 동등한 프로세스 문을 suspend될 때까지 실행하며, t=0에서 시뮬레이션을 실행한 결과 발생된 트랜잭션들은 스케쥴링된다. 단계 4가 끝나게 되면 입력자극을 해당되는 객체에 스케쥴링한다. 입력자극이 가해진 포트객체가 초기화 실행과정에서 스케쥴링된 트랙잭션이 존재한다면 이들은 모두 취소된다.

2. 시뮬레이션 실행

시뮬레이션의 실행은 초기화 단계에서 발생된 이벤트 및 입력자극으로부터 주어진 이벤트로부터 시작된다. Vsim은 시뮬레이션의 각 싸이클마다 포트 신호 및 병행 프로시져 호출문의 인터페이스 신호객체값을 갱신하고 연결관계상에 있는 신호객체들의 값을 평가하여 결정한 이후에 나머지 이벤트들을 처리한다. 〈그림 8〉은 전체적인 시뮬레이션 알고리듬이다.



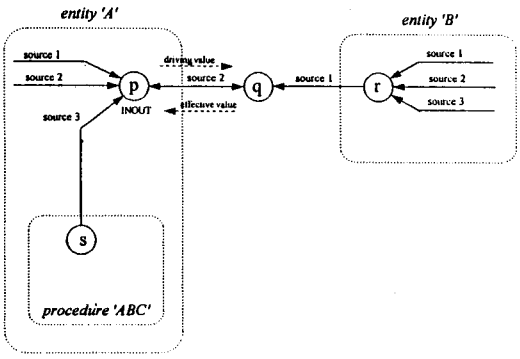
〈그림 8〉 Vsim의 시뮬레이션 알고리듬

시뮬레이션의 실행은 이벤트가 존재하는 시간까지 시뮬레이션 시간을 증가시켜 스케쥴링된 이벤트들을 타임 휠로부터 회수(retrieve)한다. 회수된 이벤트에 의하여 규정된 신호들의 신호값을 모두 갱신한 다음 신호객체, 신호타입 어트리뷰트인 QUIET, TRANSACTION, DELAYED, STABLE과 내재적 guard 신호에 발생한 이벤트와 프로세스 활성화 이벤트 등으로 분류하여 이들 이벤트를 각기 다른 리스트로 연결시킨다. 실행단계는 신호객체 중에 인터페이스 신호에 발생한 이벤트를 처리한 다음, 나머지 신호객체에 발생한 이벤트를 모두 처리하고, 신호타입 어트리뷰트와 내재적 guard 신호의 이벤트를 처리한다. 신호타입 어트리뷰트의 처리가 완료되면 활성화된 프로세스 문을 처리하여 현 시뮬레이션 싸이클을 종료하며, 각 이벤트들의 처리과정에서 발생한 트랜잭션들은 해당 신호와 시간에 스케쥴한다. 주어진 실시간 내에서 발생한 델타 시간 트랙잭션이 존재하면, 델타값을 증가시키고 델타 시뮬레이션 싸이클에서 위의 실행단계를 반복하게 된다. 더 이상의 델타 시간 트랙잭션이 존재하지 않을 경우 시뮬레이션 시

간을 증가시켜 타임 휠로부터 이벤트를 회수 위의 단계를 반복한다. 이 때에 더 이상의 이벤트가 존재하지 않거나 최종시간에 도달하게 되면 시뮬레이션은 종료된다. 실행단계에서 Vsim은 계층적인 네트웍을 flatten하지 않고 instantiation될 때마다 컴포넌트 노드를 중복시켜 entity-architecture 쌍에 대한 C/DFG을 공유함으로써 각 노드의 값이 어느 instantiation에 속한 값인가를 구분할 필요가 있다. 이를 위하여 instantiation 되는 계층에 따라 각각의 컴포넌트에 유일한 구분자를 부여하고 이를 이용하여 각 노드의 값을 처리한다. 각 시뮬레이션 싸이클을 시작하기 전에 먼저 포트 신호와 병행 프로시져 호출의 인터페이스 신호객체에 발생한 트랙잭션을 처리하여, 상호간 연결되어 있는 신호객체들 값을 결정하고 갱신한다.

3. 인터페이스 델타 시뮬레이션 싸이클

포트 신호와 프로시져 호출문에 존재하는 신호객체 타입의 파라메타와 이들과 연결관계에 있는 신호들인 인터페이스 신호객체는 이벤트가 발생하면 연결관계에 있는 신호객체에게 해당 시뮬레이션 싸이클 내에서 이벤트에 규정된 값의 전달이 이루어져야 한다. 〈그림 9〉와 같은 연결관계에서 entity 'A'의 포트 신호 p의 소스인 프로시져 'ABC'의 파라메타 s에 이벤트가 발생하였다면, 시뮬레이션 싸이클 초기 단계에서 이루어지는 포트 p신호값의 갱신은 s 신호에 발생된 이벤트가 처리되지 않은 상태에서 이루어지므로 s 신호의 값이



〈그림 9〉 인터페이스 신호 연결의 예

소스로써 포트 p에 전달되지 않았고 따라서 평가된 포트 p의 값은 올바른 값이 아니다.

이와 같이 연결관계에 있는 인터페이스 신호객체에 이벤트가 발생하고 값의 전달이 이루어지기 전에 신호객체의 값이 평가된다면 위와 같은 상황하에서 잘못된 값으로 귀결되고, 이벤트는 임의의 순서로 처리되므로 신호 s의 값이 포트 p의 소스로 전달되는 이벤트가 처리되기 전에 잘못 평가된 포트 p의 이벤트가 처리되면 잘못된 값을 가지고 구문이 수행되며, 연결관계에 있는 신호들의 소스에 전달되게 된다. 만약 잘못 평가된 포트 p의 신호값을 이용하는 이벤트 중 일부는 처리되고, s 신호값이 전달되어 올바로 평가된 값을 갖는 이벤트가 발생되어 아직 처리되지 않은 p의 이벤트들을 취소하고 새로 스케쥴된다면 이벤트 구동 알고리듬을 적용할 수 있는 전제인 값의 일관성을 부정하게 된다. 이를 해결하기 위하여 인터페이스 객체에 발생된 이벤트들을 먼저 처리하여 소스값을 갱신하고 이벤트가 발생하면 이들 이벤트들을 신호객체 처리과정의 맨 마지막에 다루어 값의 일관성을 유지시킬 수 있는 방법을 고려할 수 있으나, 인터페이스 신호의 연결관계가 전체적으로 루프를 구성하는 경우는 적용할 수 없고, 동일한 시뮬레이션 싸이클 내에서 값의 변화를 피할 수가 없다. 시뮬레이션 과정에서 인터페이스 신호객체에 발생된 이벤트를 처리하기 위하여 인터페이스 델타 개념을 도입하여 이를 처리한다.

〈그림 9〉에서 신호 s, 포트 p와 r의 source1에 이벤트가 발생하였다면, 해당 시뮬레이션 싸이클 내에서 값의 갱신이 이루어지는 과정 중 포트 p, r의 값이 평가되어 갱신된다. InterfaceDelta 시뮬레이션 싸이클에서 다음과 같이 처리되어 인터페이스 신호객체 값이 결정된다.

```
interfaceDelta = 1
처리과정 :
    s의 값이 p의 소스로 전달
    p의 구동값이 q의 소스로 전달
    r의 값이 q의 소스로 전달
interfaceDelta = 2
```

갱신과정 :
p 구동값의 평가 및 스케쥴
q 값의 평가 및 스케쥴
처리과정 :
p 구동값이 q의 소스로 전달
q 값이 p의 유효값으로 전달
interfaceDelta = 3
갱신과정 :
q 값의 평가 및 스케쥴
전달과정 :
q 값이 p의 유효값으로 전달

인터페이스 신호객체에 발생한 이벤트들을 interfaceDelta 시뮬레이션 싸이클에서 따로 처리하여 실시간 및 델타 시뮬레이션 싸이클에서 불필요한 이벤트의 발생을 방지한다. InterfaceDelta 시뮬레이션 싸이클들이 종료되면 주어진 시뮬레이션 싸이클 내에서 포트 신호객체 및 병행 프로시져 호출의 신호객체들에 발생한 이벤트들의 연결관계 상에서 처리되어야 할 부분은 처리를 완료하게 되며 동일한 시뮬레이션 싸이클 내에서 값의 변화가 없다. 〈그림 10〉은 포트 신호 및 인터페이스 신호객체의 처리 알고리듬이다.

〈그림 10〉에서 T_t는 현 시뮬레이션 싸이클에서 평가되어야 할 이벤트들의 전체집합이고, T_c는 현 interfaceDelta 시뮬레이션 싸이클에서 처리되어야 할 인터페이스 신호객체에 발생한 이벤트의 집합, Tn은 다음 interfaceDelta 시뮬레이션 싸이클 상에서 평가되어야 할 인터페이스 신호객체들의 트랙잭션 집합이다. 새로운 시뮬레이션 싸이클에 들어가면 먼저 Tt의 모든 원소에 대하여 노드값을 갱신하고 노드가 인터페이스 객체이거나 인터페이스 객체에 결합되어 있는 신호는 T_t에서 제거하여 T_c집합의 원소로 만든다. T_c집합의 각 원소 θ에 대하여 연결되어 있는 신호객체와 비교 평가하여 새로운 이벤트가 발생하면 이를 스케쥴링 집합 T_n의 원소로 만든다. 집합 T_c의 모든 원소가 처리되면 T_c는 공집합이 되고 다음 interfaceDelta 시뮬레이션 싸이클에 스케쥴링된 트랙잭션을 원소로 하는 집합 T_n를 T_c에 지정한다. InterfaceDelta 값

```
/*
*  T_t : set of all events at this simulation cycle
*  T_c : set of events occurring at interface signal objects
*  T_n : set of transactions scheduled at next interfaceDelta simulation cycle
*/
interfaceDelta ← 0;
T_t ← ∅;
T_n ← ∅;
for each transaction θ in T_t
  update node value as specified in event θ ;
  if ( node is an interface object ) then
       T_t ← T_t - { θ };
       T_c ← T_c ∪ { θ };
  end if
end for

while T_t ≠ ∅ do
 for each transaction θ in T_c
  evaluate an association including the node involved in θ;
  if ( a new transaction, ϖ, occurred at a formal or actual parameter ) then
       T_c ← T_c ∪ { ϖ };
  end if
  T_c ← T_c - { θ };
  T_t ← T_t ∪ { θ };
 end for

 interfaceDelta = interfaceDelta +1;
 T_t ← T_n;
 T_n ← ∅;
 if T_c ≠ ∅ then
   for each transaction θ in T_c
    update the node vlaue;
    if ( node is not an interface object ) then
         T_c ← T_c - { θ };
         T_t ← T_t ∪ { θ };
    end if
   end for
 end if
end while;
```

〈그림 10〉 Interface 신호객체의 트랜잭션 처리 알고리듬

을 하나 증가시킨 다음에 T_c의 원소에 대하여 갱신과 평가작업을 $T_n = \phi$일 때까지 반복하여 준다. 이와 같이 interfaceDelta 시뮬레이션 싸이클에서 해당 시뮬레이션 싸이클에서 발생한 모든 인터페이스 신호객체에 관련된 이벤트들을 처리하여 줌으로써 해당 델타와 실시간 시뮬레이션 싸이클에서 불필요한 이벤트의 발생을 방지한다.

인터페이스 신호객체에 발생한 이벤트 중 INOUT 모드를 가지는 신호객체의 유효값 결정은 연결관계 상에 있는 최상위 수준까지 연쇄적으로 구동값을 비교 평가하여 갱신한 다음 다시 최상위 수준에서 결정된 유효값이 하위수준으로 비교 평가되어 전달된다. 이를 표현하기 위하여 INOUT 모드를 가지는 신호객체는 유효값 큐와 구동값 큐를 속성으로 가지고 있다.

4. 지연 모델의 처리

VHDL은 지연모델로서 관성지연(inertial

delay), 전달지연(transport delay) 및 델타지연(delta delay) 모델을 지원한다. 전달지연은 선로지연(wire delay)과 유사한 의미로 입력신호의 변화를 일정시간 이후에 항상 반영하는 시스템을 기술할 때 사용하며, 관성지연은 입력의 변화된 값이 일정시간 이상 유지되지 않으면 응답하지 않는 시스템을 기술할 때 유용하다. 델타지연은 실시간 지연의 개념은 아니지만 이벤트의 수행순서를 나타내는 지연모델이다. Vsim에서 지연모델을 처리하는 알고리듬은 〈그림 11〉에 보였다.

시뮬레이션 시간 T_c에서 신호객체 s을 타겟으로 하는 신호 지정문이 실행되면, 새로운 트랜잭션들이 발생하게 되고, 이들 트랜잭션을 집합 T_w로 나타낸다. 이 신호 지정문에 결합된 구동원의 projected output waveform을 집합 T_p로 나타내며, T_i는 현 시뮬레이션 싸이클 시간 T_c로부터 V_i가 구동원의 현재값(current value)가 되기 위하여

```
/*
 * Waveform T_w = { (v_i, t_i) } is a set of transactions produced
 * by evaluation of signal assignment statement in current simulation cycle t_c.
 * T_p = { (v_i, t_i) } is a set of transactions in projected output waveform of driver
 *   being associated with the signal assigment statement.
 */
 if ( delta delay ) then
      /* assumption:  T_w has only one transaction */
      determine the current value of this driver from T_p at next delta simulation cycle;
      if  ( current value of driver at next delta  ≠ v_i )  then
           forwardPreemption( );
           T_p = T_p ∪ ( v_i, t_i ):
      end if
 else
      for each transaction in  T_w
        determine the current value of this driver from  T_p at  t_c + t_i;
        if  ( current value of  driver at  ( t_c + t_i ) ≠ v_i  ) then
           /* forward preemption & backward preemption */
           forwardPreemption( );
           if ( inertial delay ) then
                backwardPreemption( );
           end if
           T_p = T_p ∪ ( v_i, t_i ):
        end if
      end for
 end if

 /* forward preemption */
 forwardPreemption( )
   begin
      for each transaction in  T_p  s.t.  t_i ≥ t_i
         T_p = T_p - ( v_i, t_i ):
      end for
 end forwardPreemption;

 /* backward preemption */
 backwardPreemption( )
   begin
      for each transaction in  T_p  s.t.  t_i ≤ t_i, v_i ≠ v_i, ( v_i, t_i ) ∈ T_w
         T_p = T_p - ( v_i, t_i ):
      end for
   end backwardPreemption;
```

〈그림 11〉 지연모델 처리 알고리듬

경과해야 되는 시간적인 지연값이다.

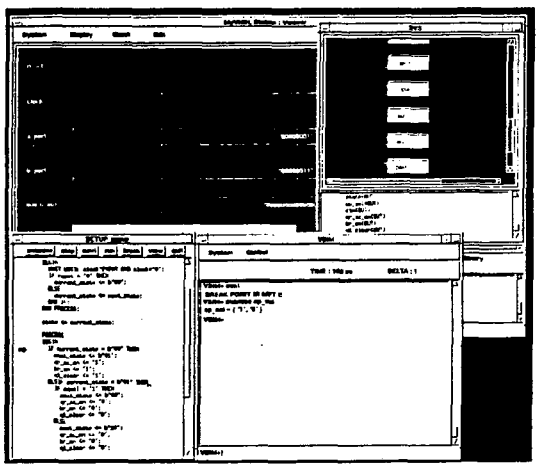
발생된 트랜잭션이 델타 지연을 가지고 있다면, 집합 T_w는 오직 하나의 원소만을 갖고 있다. 구동원의 projected output waveform에서 다음 델타 시뮬레이션 싸이클에서의 구동원의 값을 결정하고, 이 값과 V_i값을 비교하여 같지 않다면, 다음 델타 시뮬레이션 싸이클을 포함하여 $t_j \geq t_i$인 트랜잭션들은 모두 project output waveform에서 제거하는 과정인 forward preemption 과정을 거친 다음 (v_i, t_i)를 집합 T_p의 원소로 만든다. 값이 같다면 이 트랜잭션은 무시된다. 트랜잭션이 전달 지연을 가지고 발생되었다면, 이 구동원의 projected output waveform으로부터 시간 $t_c + t_i$에서의 구동원의 현재값을 결정한 다음 이 값과 v_i를 비교 같지 않다면 forward preemption 과정을 거치고 (v_i, t_i)를 집합 T_p에 더한다. 값이 같다면 이 트랜잭션은 무시된다. 관성 지연을 가지고 발생된 트랜잭션은 시간 $t_c + t_i$에서의 구동원의 현재값을 결정한 다음 이 값과 v_i를 비교하여 다르다면, forward preemption 과정과 backward preemption 과정을 수행한 다음에 (v_i, t_i)를 집합 T_p에 더한다. Backward preemption과정은 집합 T_p의 원소 중 $t_j \leq t_i$이고 $v_j \neq v_i$인 원소 중 T_w에 속하지 않는 원소를 제거하는 과정이다. 위와 같은 과정을 집합 T_w의 모든 원소에 대하여 반복하여 실행함으로써 지연 모델을 지원한다.

하나 이상의 소스를 갖는 resolved 신호객체에 대한 resolved 값의 결정은 하나 이상의 소스에 트랙잭션이 발생하면 resolution 함수를 호출하여 resolved 값을 결정하게 되고 이 값을 현재의 신호객체값과 비교 평가하게 된다. 배열타입의 신호객체는 개개의 element별로 소스에 관한 정보를 가지고 있으며 resolved 값의 결정도 element별로 이루어진다. 이를 위하여 복수개의 소스를 갖는 resolved 신호에 대하여 각 소스별로 소스 큐를 따로 설정하고, 개개의 소스는 값의 갱신 및 스케쥴링이 해당 소스 큐를 통하여 이루어지도록 하여 지원한다. Vsim은 표준 IEEE 9-value 시스템을 지원하며 사용자 정의 resolution 함수도 지원한다.

VII. 대화식 디버거

대화식 디버거는 사용자의 하드웨어 행위기술에 대한 오류를 쉽게 발견하고 수정할 수 있는 환경을 제공한다. Vsim은 소스 디스플레이 및 대화적인 waveform 디스플레이를 제공하여 대화식 디버깅이 가능하도록 지원한다. 〈그림 12〉는 대화식 디버거의 실행화면이다.

대화식 디버거는 크게 원시코드를 나타내는 소스 디스플레이어, 계층적인 정보를 그래픽으로 보여주는 hierarchy viewer, 디버거 쉘 및 실시간 waveform 디스플레이어로 구성되어 있으며, 이들 사이의 동기에는 UNIX에서 제공하는 IPC (interprocess communication)기능을 이용하였다. 소스 디스플레이어는 순차문에 한하여 현재 실행되고 있는 원시코드를 보여주며 소스 디스플레이어 상에서 블럭을 이용하여 디버거 명령의 파라메타를 지정할 수 있다. 또한 원하는 컴포넌트의 원시코드를 선택하여 보여주는 기능을 지원한다. 디버거 쉘은 지원하는 명령에 대한 도움말 기능을 제공하며 메뉴와 텍스트 명령을 선택하여 사용할 수 있고, alias 명령을 지원하여 자주 사용하는 명령 및 파라메타를 약어로 사용할 수 있는 기능을 지원한다. 실시간 waveform 디스플레이어는 모니터 명령을 이용하여 지정한 신호객체의 waveform을 변화하는 즉시 나타내 준다. Waveform 디스플레이어는 다양한 radix 변환기능이 제공되고, 디스플레이 되는 신호값의 위치를 마우스를 이용하여 변경 가능하기 때문에 신호값의 비교가 용이하다. Hierarchy viewer는 컴포넌트의 계층을 나타내주는 툴로서 해당 컴포넌트를 클릭하면 바인딩된 entity-architecture 쌍, 이들의 포트와 generic 변수를 표시하여 준다.



〈그림 12〉 Vsim의 대화식 디버거

대화적인 디버거는 디버깅에 필수적인 다양한 명령어들을 지원하고 있다. 디버거에서 주어진 객체의 현재값을 보여주는 기능으로서 examine명령이 사용되며, describe명령은 주어진 객체의 타입 및 선언부에 대한 정보를 확인할 때 사용된다. Change명령을 이용하여 특정객체의 값을 변경할 수 있으며, force명령을 이용 특정 신호객체에 waveform을 지정할 수 있다. Break 명령을 이용하여 원시코드 상에서 break point의 설정과 해제를 할 수 있고, step과 next 기능을 이용하여 순서문에서 문 단위의 실행이 가능하다. 〈표 2〉는 대화식 디버거에서 제공하는 명령어들이다.

〈표 3〉은 Vsim과 Vantage사의 디버거의 주요 기능을 비교한 것이다.[13]

〈표 2〉 디버거에서 지원하는 명령어 및 기능

명령어	기 능
alias	명령어 행을 alias로 지정
bd	지정된 break point를 삭제
bp	break point를 지정
change	변수나 신호객체의 값을 변경
cont	정지된 문장부터 다시 계속 실행
describe	변수나 신호객체 타입에 관한 정보를 표시
do	매크로 파일에 저장된 명령들을 실행
examine	변수나 신호객체의 현재값을 표시
force	신호객체에 입력자극을 대화적인 방식으로 가함
next	step 명령과 같은 기능이나 부프로그램 호출을 한 문장으로 취급
monitor	waveform 디스플레이어에 신호값을 표시
restart	시뮬레이터를 초기상태로 하여 재시작함
run	주어진 시간만큼 시뮬레이션을 실행
step	다음 VHDL 문장을 수행

〈표 3〉 Vsim과 Vantage 디버거의 주요 기능 비교

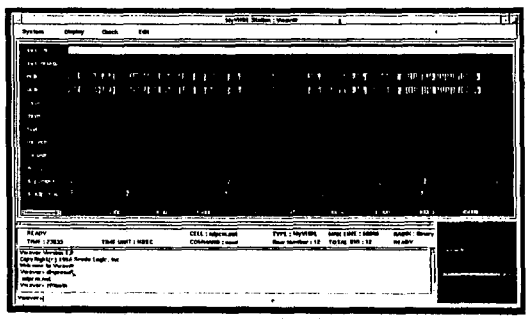
기 능	Vantage	Vsim	비 고
run	run	run	시뮬레이션 실행
again	again	없음	이전 동작의 반복
step	step	step	한 문장씩 수행
break	break	bp	break point 설정
continue	continue	cont	break 상태에서 다음을 시작
examine	examine	examine	신호값을 표시
set	set	force	입력자극을 가함
describe	describe	describe	객체 타입을 표시
drivers	drivers	없음	구동원 정보
find	find	없음	지정된 객체가 정의된 곳을 표시
monitor	wave	monitor	waveform 디스플레이어에 신호를 추가
down	push	없음	지정된 sub-block의 소스를 디스플레이
up	pop	없음	down 명령 이전 block의 소스를 디스플레이
script file	yes	yes	script 파일의 지원
환경 파일	yes	no	디버깅 환경의 저장

〈표 3〉의 디버거 기능 비교에서 Vsim은 Vantage사의 디버거와 비교하여 지원하는 기능에서 거의 같은 능력을 갖는다. Vantage사의 find 명령어는 주어진 객체가 정의된 영역을 표시하나 Vsim은 객체가 정의된 영역이 describe 명령에서 동시에 표현된다. Down-up 명령을 Vsim은 지원하지 않으나 주어진 소스파일을 임의로 지정 디스플레이 할 수 있는 능력을 가지고 있다.

VIII. 실험 결과

Vsim은 sun workstation상에서 약 35만여 라인의 C언어로 구현되었으며 배취모드 및 대화적인 시뮬레이션이 가능하다. 〈그림 13〉은 ADPCM (Adaptive Differential Pulse Code Modulation) 시스템의 행위동작을 111개의 컴포넌트로 나누어 기술한 VHDL 소스의 실행결과이다.[35] 동작기술에 사용한 ADPCM 시스템은 8KHz의 샘플링 주기를 가지며 샘플 당 4비트로 코딩되어 32Kbps로 전송되는 시스템으로 다차원 배열, resolved 신호 객체, 부프로그램 및 다양한 모드의 포트 신호객체, file 객체, access 객체 등이 사용되고 있어 Vsim의 성능을 테스트하는데 적당하다. ADPCM 동작 기술 VHDL 원시코드는 111개의 컴포넌트로 구성되어 있고 약 13,000라인으로 구성되어 있다. 행위기술에 대한 시뮬레이션 결과 동작기술이 예측했던 값과 일치하는 것을 확인하였다.

Virginia Tech의 테스트 쉬위트, 서강대 CAD/컴퓨터 시스템 연구실, 인천대 CAD 연구실의 테스트 프로그램에 시뮬레이션한 결과 모든 구문에 대하여 올바른 결과를 출력하였다. 각 연구실에서 작성한 테스트 쉬위트는 개별 구문의 동작 확인을 위한 테스트 입력과 하드웨어 컴포넌트의 동작을 기술한 것들로서 VHDL에서 제공하는 여러 구문을 확인하는 것을 목적으로 하였다. 〈표 4〉는 대



〈그림 13〉 ADPCM 시뮬레이션 결과

〈표 4〉 대표적인 테스트 입력

이 름	기 능
alu—1	구조적인 8-bit alu 기술
alu—2	행위적인 8-bit alu 기술
booth—1	8-bit Booth 곱셈기
booth—3	8-bit Booth 곱셈기
cr8000	8-bit CPU
fa—1	전 가산기
fa—2	전 가산기
fpuas—1	floating point unit
fsm—1	finite state machine
sd8010	8-bit RISC CPU
ADPCM	ADPCM 기술
IIR6	6차 IIR 필터
RC8010	8-bit RISC CPU

표적인 테스트 입력이다.

〈표 5〉 RC8010에 대한 setup 시간

	Vsim(inter-pretive)	Vantage(c-code compiled)	ViewLogic (compiled)
컴파일 속도	17sec	276sec	165sec

각각의 테스트 입력에 대하여 Vsim은 올바른 시뮬레이션 결과를 출력하였으며 현재 연구실내에서 하드웨어 시뮬레이션에 이용하고 있다. 〈표 5〉는 RC8010에 대한 각 시뮬레이터 사이의 setup 시간 비교이다.

Vsim은 해석적 방식의 시뮬레이터이고 Vantage와 ViewLogic사의 시뮬레이터[36]는 컴파일 방식의 시뮬레이터들이며, setup 시간은 Vsim이 Vantage사의 툴에 비하여 대략 17배, ViewLogic 툴에 대하여서는 약 10배 정도 빠르다. 〈표 6〉은 대표적인 테스트 입력에 대한 시뮬레이터들 간의 속도 비교이다.

300여개의 테스트 입력에 대하여 시뮬레이션 속도 비교를 행한 결과 입력에 따라 같은 속도가 나오는 경우에서부터 대략 Vantage사 툴에 비하여 20배 정도 느린 경우도 존재한다. 이와 같은 결과가 나오는 원인은 상용 시뮬레이터는 내부적으로 직접 컴파일 방식을 사용하였고 Vsim은 해석적 방식의 사용한 시뮬레이터라는 것이 일차적인 원인이다.[14] 각 테스트 입력에 대하여 Vsim은 시뮬레이터 setup 시간이 Vantage 툴과 ViewLogic사의 툴에 비하여 대략 50~10배 정도 빠르므로, setup 시간과 시뮬레이션 시간의 합인 반환시간은 다른 툴에 비하여 짧다. 에러가 자주 발생하고 코드의 변경이 빈번한 모델의 초기 개발 과정에서 Vsim은 반환 시간 측면에서 볼 때 장점을 가지고 있는 툴이다.

〈표 6〉 시뮬레이션 속도 비교

	Vsim	Vantage	ViewLogic
ALU(행위 기술, 196line)	X 1	X 2	X 1
8×8booth 곱셈기(행위 기술, 140line)	X 1	X 4	X 1.5
8×8booth 곱셈기(구조 기술, 4976line)	X 1	X 20	X 25
FSM(행위 기술, 71line)	X 1	X 2	X 1
CR8000(행위 기술, 2500line)	X 1	X 10	X 2
CR8010(행위 기술, 3000line)	X 1	X 13	X 3
RC8010(행위 기술, 4000line)	X 1	X 20	X 1.5

IX. 결론 및 추후과제

본 논문은 VHDL 설계지원환경 구축의 일환으로서 개발된 해석적 방식의 시뮬레이터인 Vsim의 설계 및 구현결과에 대하여 기술하였다. Vsim은 87년도 표준안인 IEEE std 1076-1987 버젼의 전체사양을 지원하고 있다. 시뮬레이션에 필요한 소속 및 계층적인 정보가 표현되어 있는 계층적인 C/DFG를 내부모델로 사용하여 이벤트 구동 알고리듬을 적용하였다. Vsim은 표준 IEEE 9-value package인 IEEE 1164을 지원하고 있으며 편리한 사용자 환경, 대화적인 디버깅 환경 및 실시간 waveform 디스플레이 기능을 가지고 있다. 실험결과 테스트 입력에 대하여 올바른 시뮬레이션 결과를 출력하였으며, 시뮬레이션 속도에서는 기존의 상용 시뮬레이터와 비교시 테스트 입력에 따라 100%에 가까이 향상된 경우도 있었다.

Vsim은 해석적 방식으로 구현된 VHDL 시뮬레이터인 관계로 원시코드 생성 방식이나 C 코드 생성 방식으로 구현된 상용 VHDL 시뮬레이터에 비하여 속도 측면에서는 뒤지나 VHDL 전체 사양을 지원하고 있고, 상용 시뮬레이터에 비하여 50~10배 정도의 짧은 setup 시간을 가지고 있으며, 다양한 디버깅 기능을 지원하고 있어 상위 수준에서 설계하고자 하는 대상 회로를 시뮬레이션하거나, VHDL 교육용으로 사용될 경우 상용 시뮬레이터에 비하여 가격대 성능비에서나 효과면에서 장점이 있다는 것이 실제 사용 결과 증명되었다. Vsim은 VHDL 분석기인 Vcom과 함께 자체적으로 개발된 툴로서 VHDL 시뮬레이션 알고리듬의 개선이나 새로운 아이디어를 실험하는 프로토 타입으로 사용할 수 있다.

개발 과정에서 획득한 다양한 지식과 경험을 바탕으로 현재 해석적 방식의 Vsim에 대하여 내부모델과 시뮬레이션 알고리듬의 개선 작업이 진행되고 있으며, 내부모델 개선작업은 Vsim 실행시 수행되는 루틴의 수행 빈도와 수행 시간에 기초하여 해당 루틴이 최적화되도록 하는 프로그래밍 작업과 함께 내부모델을 변경하여 구현하고 이를 비교하는 작업으로 이루어지고 있다. 시뮬레이션 알고리듬의 개선 작업은 특정한 조건 하에서 수행되는 구문, 특별한 조건 하에서 발생한 이벤트의 전달 및 평가가 시뮬레이션 속도를 증가시키는 측면에서 최적화 될 수 있는가를 찾고 이를 구현 및 평가하는 식으로 진행시키고 있으며, 실행속도 향상을 위한 원시코드 생성 방식의 VHDL 시뮬레이터의 구현도 진행 중에 있다.

감사의 글

본 연구는 통산 산업부 공업기반 기술과제 'VHDL을 이용한 ASIC Synthesis Software 개발에 관한 연구' 수행을 위해 서두로직(주)와 협력연구로 이루어진 것입니다.

참 고 문 헌

[1] M. C. McFarland, "Tutorial on High-Level Synthesis," *in Proc. 25th DAC*, June 1988, pp. 330~336.

[2] R. W. Hartenstein, *Hardware Description Languages*, North-Holland : NY, 1987.

[3] E. E Murphy, "Design Automation Standards," *IEEE Spectrum*, vol. 28, no. 3, March 1990, pp. 44~45.

[4] A. Dewey and A. Gadient, "VHDL Motivation," *IEEE Design and Test of Computers*, vol. 3, no. 2, April 1986, pp. 12~16.

[5] J. H. Aglor, R. Woaxmam, and C. Scarratt, "VHDL-Feature Description and Analysis," *IEEE Design and Test of Computers*, vol. 3, no. 2, April 1986, pp. 17~27.

[6] "A D&T Roundtable : Behavioral Description Languages, Part 2 : VHDL vs.

UDL/I," *IEEE Design and Test of Computers*, vol. 7, no. 2, April 1990, pp. 64~68.

[7] E. Meger, "VHDL Opens the Road to Top-Down Design," *Computer Design*, vol. 28, no. 3, Feb. 1989, pp. 57~60.

[8] R. Collet, "Market Analysis," Dataquest Perspective, CAD/CAM/CAE-Electronic Design Automation Applications, April 1992, pp. 1~7.

[9] M. Jain, "The VHDL Forecast," *IEEE Spectrum*, vol. 30, no. 6, June 1993, pp. 36.

[10] M. Carrol, "VHDL-Panacea or Hype?," *IEEE Spectrum*, vol. 30, no. 6, June 1993, pp. 34~37.

[11] *Leapfrog VHDL Simulator Reference Manual 1.0*, Cadence, June 1993.

[12] *QuickSim II User's Manual*, Mentor Graphics Inc., 1992.

[13] *Vantage Spreadsheet User's Guide Volume 1*, Vantage Analysis Systems Inc., Dec. 1990.

[14] V-*System/Windows User's Manual*, Model Technology, March 1993.

[15] *VHDL System Simulator Workshop*, Synopsys Inc., 1994.

[16] *SPW User Meeting*, 서두로직, 1993.

[17] *MyVHDL Station*, 서두로직, 1994.

[18] *MySynOpt Station*, 서두로직, 1994.

[19] 이영희, 황선영, "VHDL 설계환경 구축을 위한 front-end의 설계," 한국 정보과학회 논문지, 제18권 제1호, 1991년 1월, pp. 93~103

[20] 이영희, 김헌철 황선영, "다층레벨 VHDL 시뮬레이터 설계," 대한 전자공학회 논문지, 제30-A권 제10호, 1993년 10월, pp. 67~76

[21] 현민호, 황선영, "레지스터 전송수준 합성 시스템 설계," 대한 전자공학회 논문지, 제30-A권 제5호, 1994년 5월, pp.147~157

[22] M. H. Hyun, S. Y. Hwang, Y. U. Yu, "Synthesis of VHDL Sequential Statements at Register Transfer Level," *in Proc. of APCHDL "94*, Oct. 1994, pp. 243~246.

[23] 이해동, 전홍신, 황선영, "VHDL 상위수준 합성 시스템 설계," 한국 정보과학회 논문지,제20권 제6호, 1993년 6월, pp. 788~801

[24] *HUM : VHDL Design and Modeling System*, Lewis System Inc., 1994.

[25] J. Snaguinelti, "Compiled vs. Interpreted Simulation," *Electronic Design*, vol. 41, no. 6, March 1993, pp. 50~51.

[26] L. Maliniak, "Simulating at High Levels Shows Promise," *Electronic Design*, vol. 41, no. 6, March 1993, pp.43~53.

[27] D. M. Lewis, "A Hierarchical Compiled Code Event-Driven Logic Simulator," *IEEE Trans. on CAD*, vol.10, no.6, June 1991, pp. 726~737.

[28] L. Maliniak, "Native-Compiled-Code Technology Runs VHDL Simulations up to 25 Times Faster," *Electronic Design*, vol. 41, no. 7, April 1993, pp. 31.

[29] K. H. Wong and M. D. Feinstein, "Evaluation and Benchmarking of VHDL Simulators," *Pacific Communication Sciences Inc.*, April 1993.

[30] *VHDL User's Manual Volume 1-Tutorial*, Intermetric Inc., 1985.

[31] *The Sense of the VSAG*, VHDL User's Group Fall 1989 Meeting, Oct. 1989.

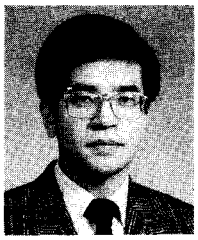
[32] *IEEE Standard VHDL Language Reference Manual*, IEEE std 1076-1987, Apr. 1989.

[33] *IEEE Standard Interpretations : IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076/INT-1991, 1991.

[34] R. E. Harr and A. G. Stanculescu, *Applications of VHDL to Circuit Design*, Kluwer

Academic Pub., 1991.
[35] 디지탈 통신용 ADPCM ASIC 개발, 연차 연구보고서, 체신부, 1992년 9월
[36] *Viewsim/SD Reference Manual*, Viewlogic Systems Inc., Dec. 1989.

저 자 소 개

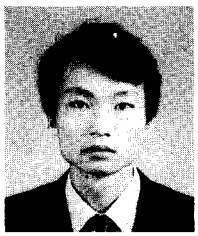


黃 善 泳

1976年 2月 서울대학교 공대 전자공학과 졸업(공학사)
1978年 2月 한국과학원 전기 및 전자공학과 졸업(공학석사)
1986年 10月 미국 Stanford대학 전자전산학(공학박사)

1976年 3月～1981年 7月 삼성반도체(주) 연구원, 팀장
1986年 6月～1989年 1月 Stanford대학 Center for Integrated Systems 연구소 책임 연구원
1986年 11月～1988年 12月 Fairchild Semiconductor Inc. Palo Alto Research Center 기술자문

주관심분야 : CAD시스템, 컴퓨터아키텍춰 및 시스템 설계, VLSI시스템 설계



李 榮 熙

1966年 3月 6日生
1989年 2月 서강대학교 전자공학과 졸업(공학사)
1992年 2月 서강대학교 전자공학과 공학 석사 학위 취득
1992年 3月～ 서강대학교 전자공학과 박사 과정 재학중

1992年 3月～현재 서강대 전자공학과 CAD & Computer System 연구실 연구원
1993年 8月～1994年 12月 단국대 강사
1994年 3月～현재 경인전문대 강사

주관심분야 : CAD System 및 혼합모드 시뮬레이션 알고리듬 등임.