論文93-30A-10-9

다층 레벨 VHDL 시뮬레이터의 설계

(Design of a Multi-level VHDL Simulator)

李 榮 熙*, 金 憲 哲*, 黃 善 泳**

(Young Hee Lee, Heon chul Kim and Sun Young Hwang)

要 約

본 논문은 VHDL 설계환경 구축의 일환으로 연구된 다층 레벨 VHDL 시뮬레이터인 SVSIM(Sogang VHDL SIMulator)의 설계 및 구현 결과에 대하여 기술한다. SVSIM은 시뮬레이터의 내부 모델로서 기존의 C/DFG에 대하여 계층성, 전역적 및 지역적인 제어 정보가 명확히 표현되도록 확장한 계층적인 C/DFG을 사용하였으며, 계층적인 기술에 대하여 네트웍을 flatten하지 않고 instance 정보와 함께 네트웍을 공유하여 사용함으로써 공간 복잡도를 줄였다. SVSIM은 record 타입을 제외한 모든 기정의 타입 및 사용자 정의 타입을 지원하며 VHDL의 주요구문 전부를 지원한다. 또한 기정의 attribute 및 사용자 정의 attribute, 부프로그램의 호출 및 부프로그램의 recursive call을 지원하여 알고리듬 레벨에서의 시뮬레이션이 용이하도록 하였으며, 지연 모델로서 실시간 지연인 관성, 전달지연과 델타 지연을 지원한다. 시뮬레이션의 입력은 별도의 입력자극 명령어나 혹은 VHDL 원시코드상에서 줄 수 있도록 설계하였다. 실험결과 SVSIM은 순수한 행위적, 구조적 및 이들의 혼합으로 기술된 시스템에 대하여 정확한 시뮬레이션을 수행하였다.

Abstract

This paper presents the design and implementation of SVSIM(Sogang VHDL SIMulator), a multi-level VHDL simulator, designed for the construction of an integrated VHDL design environment. The internal model of SVSIM is the hierarchical C/DFG which is extended from C/DFG to include the network hierarchy and local/global control informations. Hierarchical network is not flattened for simulation, resulting in the reduction of space complexity. The predefined/user-defined types except for the record type and the predefined/user defined attributes are supported in SVSIM. Algorithmic level descriptions can be simulated by the support of recursive procedure/function calls. Input stimuli can be generated by command script in stimuli file or in VHDL source code. Experimential results show SVSIM can be efficiently used for the simulation of the pure behavioral descriptions, structural descriptions or mixture of these.

I. 서론

VLSI 제조기술의 발달과 더불어 칩의 집적도가 증가하여 기존의 bottom up 방식에 의한 설계 기술이 문제점으로 대두하였으며 이를 해결하기 위한 방법으로 top down 설계 방식이 도입되었다. Top down

*準會員, **正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogan Univ.)
接受日字 : 1993年 2月 8日

설계 방식의 지원을 위해 상위 수준 기술언어와 실리콘 컴파일러가 개발되었으나, 많은 시스템들이 각기 다른 입력 및 설계 데이타 형식을 사용함으로써 각 시스템들간 데이타 호환성이 결여되어 설계 데이타를 공유할 수 없는 비효율성이 문제점으로 등장하게 되었다. [1]

VHDL(VHSIC Hardware Description Language)은 VHSIC(Very High Speed IC) 기술을 군사 시스템에 효과적으로 도입하고, 디지탈 시스템의 설계와 문서화를 능률적으로 처리하기 위한 표준 HDL의 필요성으로 미국방성의 지원 아래 1983년부터 본격적으로 개발되기 시작하였으며 1987년 IEEE std 1076-1987이라는 표준 version을 발표함으로써 사용되기 시작하였다. [2] VHDL은 합성과 시뮬레이션을 동시에 지원하며, 설계 데이타의 교환과 기술 발전에 따른 설계 데이타의 재사용 및 시스템에 대한 documentation등을 목적으로 하고 있으며, 아키텍쳐 레벨에서부터 게이트 레벨까지의 동시 기술이 가능하다. 또한 VHDL은 실시간 타이밍을 지원하며 행위 및 구조의 혼합 기술과 계층적, 혼합 레벨(multi-level) 및 혼합 모드(mixed-mode)의 기술을 지원한다. [3]

VHDL 시뮬레이터는 상업적으로 여러 회사들에 의한 지원이 있으나 사용된 내부 데이타 구조, 알고리듬등에 대한 구체적인 보고가 없으며 또한 VHDL 언어의 전체 사양을 지원하지 않는 경우도 있어 각 시뮬레이터간의 호환성에 문제가 존재하는 경우도 있다. [4] [5] [6] 국내에서는 VHDL 전체 사양을 지원하는 시뮬레이터에 대한 연구결과는 아직 없으며 로직 레벨 및 행위 단계 VHDL 시뮬레이터에 대한 연구결과가 발표되어 있다. [7] [8]

SVSIM(Sogang VHDL SIMulator)은 계층적인 혼합 레벨 시뮬레이션을 지원하며 따라서 순수한 행위 및 구조 기술과 이들의 혼합으로 이루어진 하드웨어 기술에 대한 시뮬레이션이 가능하다. 합성 시스템과의 호환성, 계층적인 정보 및 효율적인 시뮬레이션을 지원하기 위하여 SVSIM의 내부 모델로서 기존의 C/DFG을 확장한 계층적인 C/DFG을 사용하였다. 기존의 C/DFG는 계층적인 정보의 표현이 용이하지 않으며 또 시뮬레이션시에 필요한 지역적 및 전역적인 제어정보가 결여되어 있는 반면 계층적인 C/DFG는 이를 보다 명확히 표현할 수 있으며 이들 정보는 합성과 시뮬레이션시에 효율적으로 사용될 수 있다. 계층적인 C/DFG를 사용함으로써 SVSIM은 계층적인 기술을 flatten하지 않고 네트웍을 instance 정보와 함께 공유하여 사용함으로써 공간 복잡도(space complexity)를 줄인다. [9]

SVSIM은 record 데이타 타입을 제외한 모든 타입 및 attribute, generate문을 포함한 VHDL 주요 구문을 전부 지원하며 부프로그램 및 이들의 recursive call을 지원한다.

본 논문은 VHDL 설계 환경 구축의 일환으로 개발된 혼합 레벨 VHDL 시뮬레이터인 SVSIM의 설계 및 구현에 대하여 기술한다. 제 2장에서는 SVSIM의 전체적인 구성에 대하여 기술하고 3장에서는 SVSIM의 내부 모델에 대하여 기술하며 제 4장에서는 VHDL 구문과 SVSIM의 시뮬레이션 알고리듬에 대하여 기술한다. 제5장에는 SVSIM의 실험 결과가 제시되었으며 결론 및 추후과제는 6장에 기술되어 있다.

Ⅱ. SVSIM의 구성

SVSIM의 전체적인 구성은 그림 1과 같다. 주어진 대상 하드웨어의 VHDL 원시 코드는 VHDL 분석기인 SVC(Sogang VHDL Compiler)에 의하여 구문분석 및 의미분석과정을 거쳐 라이브러리에 저장 관리된다. [10] 라이브러리에 저장된 설계단위를 회수하여 행위 변환과정(behavioral transformation) [11]을 거치며 이 과정에서 loop-unrolling, dead-code elimination, constant folding과 같은 최적화과정이 수행된 다음 계층적인 C/DFG 생성과정으로 들어

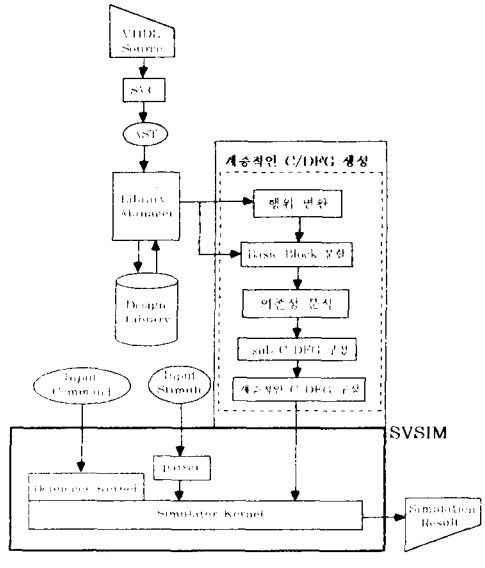

그림 1. SVSIM의 전체적인 구성
Fig. 1. Overview of SVSIM.

가게 된다. 계층적인 C/DFG의 생성과정은 먼저 주어진 원시코드상에서 기본 블록(basic block)을 나눈 다음 의존성 분석과정을 거쳐 데이타 및 제어흐름에 대한 정보를 분석하여 기본 블록에 대한 C/DFG을 구성한다. 구성된 C/DFG를 따라가면서 계층성에 대한 정보, 전역적 및 지역적인 제어정보와 각각의 신호 객체에 대한 소속(membership)을 확정하여 계층적인 C/DFG을 구성한다. 병행문, 부프로그램은 구문노드를 구성하며, 이들 구문노드는 시뮬레이션시에 수행되는 지정(assignment)과 연산의 기본 단위가 되고 이벤트에 따른 활성화 조건 및 sensitivity 집합을 가지고 있다.

입력 자극 분석기는 입력파일을 분석하여 시뮬레이션 커널에 링크시키며 대화식 디버거는 사용자의 디버거 명령을 분석하여 디버거의 커널에 링크시킨다.

표 1은 SVSIM에서 지원되는 VHDL 주요구문을 보였다. SVSIM은 VHDL의 병행문 및 순서문 모두를 지원하며 신호, 변수 및 상수객체를 지원하고 record 타입을 제외한 모든 기정의 타입 및 사용자 정의 타입을 지원한다. Attribute는 기정의 attribute을 모두 지원한다. Generate문은 분석과정에서 component instantiation문으로 대치되어서 내부 구조를 형성한다.

표 1. 지원되는 VHDL 주요 구문
Table 1. Major VHDL constructs supported by SVSIM.

병 행 문
Block statements, Concurrent signal assignments Concurrent assertion statements, Concurrent procedure calls Process statements, Component instantiation statements Generate statements
순 서 문
Signal assignment statements, Variable assignment statements If statements, Assertion statements, Case statements Next statements, Exit statements, Return statements Null statements, Wait statements
객 체
Signals, Variables, Constants, Ports, Generics
기 타
Attribute declarations, Register, Bus, type declarations Subtype declarations, Disconnection specification Configuration declaration

Ⅲ. 중간 형태

설계자에 의해 기술된 대상 하드웨어의 VHDL 모델은 VHDL 분석기를 통하여 구문 분석과 의미 분석 과정을 거쳐 AST(Abstract Syntax Tree)가 생성된다. AST는 분석결과 생성되는 중간형태로서 VHDL 원시코드의 의미를 정확하게 표현하고 있는 데이타 구조이지만 합성에 필요한 의존성 관계를 표현하기 어렵고, 시뮬레이션시에 이벤트 처리에 적합한 구조가 아니므로 이를 상위 수준의 합성과 시뮬레이션에 적합한 중간형태로 변환시킨다. SVSIM은 내부모델로서 VHDL의 주요구문들의 의미를 정확히 반영하며 합성과 시뮬레이션을 효과적으로 지원하는 자료구조로 전역적 및 지역적 제어정보가 명확히 표현되고 데이타와 제어정보가 동시에 결합되어 있으며 구조기술에 적합한 계층성에 대한 정보를 유지하는 계층적인 C/DFG를 이용하고 있다.

계층적인 C/DFG는 병행문과 순서문을 지원하기 위하여 모든 신호와 그 신호에 영향을 받는 구문들과의 연결 정보, 그리고 순서문에서 각 구문들 사이의 순서 정보를 효과적으로 표현하는 데이타 및 제어 흐름 그래프를 이용한다. 데이타 흐름 그래프는 노드와 간선으로 이루어진 방향성 그래프로 노드는 VHDL이 지원하는 모든 구문을 나타낼 수 있으며, 간선은 데이타의 흐름과 그의 방향을 나타낸다. 또한 C/DFG에는 데이타 흐름 정보뿐만 아니라 효율적인 합성을 위한 의존성에 대한 정보도 포함하며, 제어의 흐름에 관한 정보도 포함한다. 또한 구문노드는 활성화 집합을 가지고 있으며 객체노드는 속한 구문노드에 대한 정보를 포함하고 있으므로 이벤트에 의하여 활성화 되는 구문노드 및 구문노드의 활성화 조건 평가에 적합하다.

1. 의존성 분석

의존성 분석은 AST로부터 의존성 그래프(dependency graph)를 생성하고 병렬성을 찾아내어 합성시 최적화에 도움을 주기 위해서 수행한다. 병행 지정문은 그 자체에 병렬성을 포함하고 있고 수행시 서로간에 특정한 선행관계가 존재치 않아 의존성 관계가 일반적으로 존재하지 않는 반면 순서문은 문의 수행순서가 명확하므로 서로간의 의존성 관계가 일반적으로 존재한다. 따라서 C/DFG의 구성은 순서문에 대해서는 데이타 의존성 그래프를 이용하고, 병행문의 경우에는 AST로부터 직접 C/DFG를 구성한다.

의존성 관계는 지정문들의 source쪽과 destination쪽의 관계에서 생기는 것으로 데이타 의존성(data dependency), 데이타 역의존성(data antidependency), 출력 의존성(output dependency) 등이 있으며 의존성 그래프는 기본 블록내의 두 지정문들 간에 위와 같은 의존성이 존재할 경우에 두 지정문을

나타내는 노드를 간선으로 연결하고, 간선에 의존성의 종류에 대해 표현함으로써 이루어진다. [12]

2. 계층적인 C/DFG의 구조

계층적인 C/DFG는 행위기술, 구조기술 및 이들의 혼합을 지원하며 추상 데이타 타입 또한 지원한다. 계층적인 C/DFG는 크게 노드와 간선으로 구성되어 있다. 노드는 abstract operation 노드, object 노드, 구문노드, 구조노드, 지정(assignment) 노드 및 제어노드 등이 있으며 간선은 데이타의 흐름 및 제어정보를 나타낸다.

VHDL에서 하드웨어의 한 부분에 대한 구조기술은 그 하드웨어 모듈을 구성하는 하위 컴포넌트에 관한 정보와 각 하위 컴포넌트들간의 연결정보를 포함하며 이들의 기본단위는 컴포넌트 instantiation 문이다. 컴포넌트 instantiation 문은 하위 컴포넌트를 지정하고 상위 컴포넌트에서 선언된 신호들과 이들 컴포넌트들 간의 연결정보로 구성된다. 또한 하나의 컴포넌트는 복수의 instantiation이 가능하므로 각각의 컴포넌트 instantiation문에 대하여 계층적인 C/DFG는 하나의 entity 데이타 노드를 생성하며 구별자로서 유일한 정수를 부여하여 네트웍을 공유한다. 그림 2는 전 가산기에 대한 계층적인 C/DFG의 예이다.

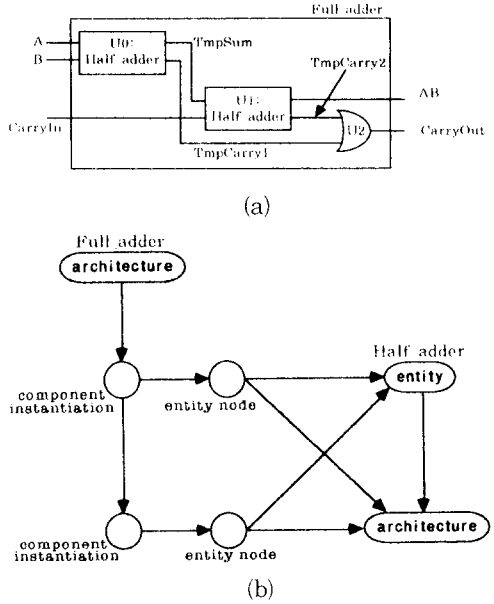

그림 2. (a) 전 가산기의 블록 다이아그램
(b) 전 가산기의 계층적인 C/DFG
Fig. 2. (a) Block diagram of a full adder,
(b) Hierarchical C/DFG for the full adder.

C/DFG에서 사용되는 간선에는 데이타 간선과 제어 간선이 있다. 데이타 간선은 C/DFG 상에서 데이타의 흐름을 나타내는 간선이고, 제어 간선은 상위 수준의 합성에 필요한 순서문간의 데이타의 역의존성과 출력 의존성 및 연산자 노드간의 순서정보와 병렬성을 나타낸다.

계층적인 C/DFG 구현 알고리듬의 전체적인 흐름은 그림 3과 같다. AST를 입력으로 하여 문의 리스트를 따라가면서 순서문의 경우에는 기본 블록내에서 sub C/DFG를 구성한 다음 순서문이 속한 상위 구문구조의 C/DFG에 head 리스트를 연결한다. 병행문은 개개의 문에 대하여 sub C/DFG를 구성하고 전체적인 C/DFG의 구성이 끝나면 구성된 C/DFG에서 이용할 수 있는 정보를 바탕으로 병행문의 활성화 신호들의 집합과 참조된 상위 블록 신호들의 집합을 추출하여 각 구문 노드의 attribute set을 바탕으로하여 활성화 조건의 영향아래 있는 sub C/DFG을 구문노드에 연결한다.

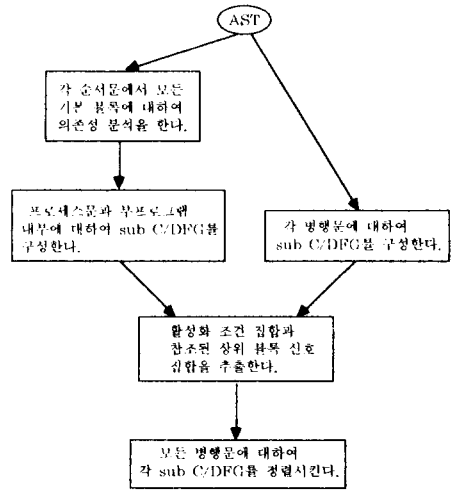

그림 3. 계층적인 C/DFG 생성의 전체적인 흐름도
Fig. 3. Overview of the hierarchical C/DFG generation.

Ⅳ. 시뮬레이션 알고리듬

시뮬레이터는 기술된 하드웨어를 처리하는 내부모델의 형태에 따라 컴파일드 시뮬레이터와 해석적

(interpretive) 시뮬레이터로 구분된다.[13] 컴파일드 시뮬레이터는 시뮬레이션 실행 이전에 대상 하드웨어의 모델을 기계어 명령인 컴파일드 코드로 변환시킨 후, 그 기계어 코드를 실행하여 시뮬레이션하므로 공간 복잡도 (space complexity)가 작고, 속도에 장점이 있는 반면에 이식성이 없고 임의의 타이밍 및 보다 추상적인 레벨의 동작 기술을 지원하는데 어려움이 있다. 해석적 시뮬레이터는 대상 하드웨어에 대한 정보를 자료구조로 표현하여 이를 근거로 시뮬레이션을 수행하므로 임의의 타이밍의 지원이 가능하고 아키텍쳐 레벨에서 스위치 레벨까지 시뮬레이션이 가능하며, 이벤트 구동(event-driven) 시뮬레이션 알고리듬의 적용이 가능하다.[14] 이벤트 구동 시뮬레이션은 이벤트에 의해 활성화 (activation)되는 엘리먼트를 결정하고 이벤트 전달 (event propagation)에 의해 활성화된 엘리먼트의 출력값을 평가 (evaluation) 한다. 컴파일드 시뮬레이션에 비해 이벤트 구동 시뮬레이션은 이벤트의 전달과 평가에 시간 개념이 포함되어 정확한 타이밍 모델이 가능하므로 비동기 회로에 대해서도 정확한 시뮬레이션을 수행할 수 있다.[15]

VHDL은 개발 단계에서부터 이벤트 구동 알고리듬에 적합하도록 구문 및 의미가 정의되었다.[16] SVSIM에서는 대상 하드웨어 기술에 대한 내부모델을 시뮬레이션과 합성에 필요한 정보를 담고 있는 계층적인 C/DFG에 표현하고 이를 이용하여 이벤트 구동 알고리듬을 적용하였다. SVSIM은 입력 네트웍에 대하여 각각의 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 결정 이후에 네트웍을 안정된 상태로 만들기 위한 초기화 실행과정을 거치게 되며 실행결과 최종적인 초기값이 결정된다. 네트웍의 초기값 결정은 그림 4에 보인바와 같이 5단계를 거쳐서 결정된다.

초기화 과정중 첫 단계는 네트웍에 있는 port 및 부프로그램의 신호객체를 포함한 모든 신호에 대한 초기값의 결정이다. VHDL은 신호의 초기값에 대한 지정 규칙을 신호의 선언부에 초기 지정값이 명시되어 있으면 이 값을 신호의 초기값으로 지정하고 명확히 명시되어 있지 않으면 신호가 선언된 데이타 타입을 T라 할 경우 T'LEFT로 지정하도록 하고 있다.

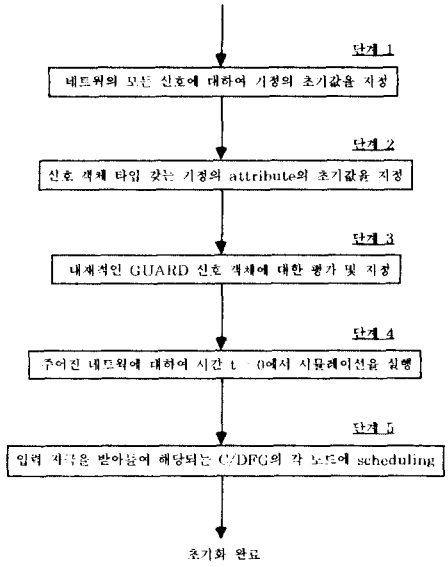

그림 4. 초기화 알고리듬
Fig. 4. Initialization algorithm.

T'LEFT는 데이타 타입 T에 속하는 값중에 맨 왼쪽에 있는 값을 의미한다.[17] 제 2단계에서는 신호 타입 기정의 attribute의 초기값을 결정한다. S'QUIET와 S'STABLE은 초기값으로 BOOLEAN 타입의 TRUE 값을 지정하며 S'TRANSACTION은 초기값으로 BOOLEAN 타입의 FALSE값을 지정한다. 단계 3은 block문의 내재적 GUARD 신호에 대하여 단계 1과 2에서 지정된 초기값을 가지고 GUARD 조건식을 수행한 다음 결과값을 내재적 GUARD신호의 초기값으로 지정한다. 단계 4에서는 위 단계를 거치면서 지정된 신호들의 초기값을 가지고 주어진 네트웍에 대한 시뮬레이션을 실행한다. VHDL에서 모든 병행문은 동등한 process 문으로 표현이 가능하기 때문에 모든 행위적 병행문 즉 병행신호 지정문, 병행 procedure 호출문 및 병행 assertion문은 동일한 동작을 하는 process문으로 나타낼 수 있다.[18] 단계 4의 실행은 process문을 포함한 행위적 병행문을 동등한 process문으로 생각할 때 모든 동등한 process문이 suspend될 때까지 실행하게 된다. T = 0 에서 시뮬레이션을 실행한 결과 발생된 transaction은 각각의 신호에 대하여 scheduling한다. 단계 5가 끝나게 되면 입력자극을 받아들여 해당되는 신호에 연결한다.

2. 시뮬레이션 실행

시뮬레이션은 초기화 단계에서 scheduling된 이벤트 및 입력자극에서 주어진 이벤트로부터 네트웍의

시뮬레이션은 실행된다. 그림 5는 SVSIM의 시뮬레이션 알고리듬의 전체적인 흐름을 나타낸 것이다.

시뮬레이션의 실행은 전체적으로 시뮬레이션 시간을 증가시켜 현재의 실시간 내에 이벤트들을 타임 wheel로부터 회수한다. 만약 현재의 이벤트들이 존재한다면 실행에 들어가게 되고, 없다면 시뮬레이션 시간을 이벤트가 존재하는 시간까지 증가시킨다. 각각의 이벤트에 해당하는 신호의 value 값을 갱신한 다음 신호객체, 신호 타입의 attribute인 QUIET, TRANSACTION, DELAYED, STABLE의 이벤트 및 프로세스 활성화 이벤트로 분류하여 이들 이벤트를 각기 다른 리스트로 연결한다. 각 리스트는 이벤트가 일어난 노드가 속한 위치에 따라 우선 순위를 주고 우선 순위에 따라 분류한 다음 우선 순위가 높은 이벤트를 먼저 처리한다. 우선 순위는 컴포넌트의 association이 가장 높고, 블록문의 guarded 표현식, 병행 fork 노드의 조건, 병행 sfork 노드의 조건, 프로세스문 및 wait문의 sensitivity 리스트와 신호 지정문에서의 구동원(driver) 순으로 낮아진다. 이와 같이 정의된 우선 순위에 따라 분류한 다음 우선 순위가 높은 쪽을 먼저 처리함으로써 전역적 및 지역적인 제어 정보가 먼저 처리되고 따라서 하위 네트웍 중에서 전역적 제어에 의해 활성화되지 않는 부분에 발생한 이벤트는 value 갱신단계만 거치고 실행되지 않는다.

실행 단계는 신호 객체에 발생한 이벤트를 모두 처리한 다음, 신호 타입 attribute의 이벤트를 처리한다. 신호 타입의 attribute의 처리가 완료되면 활성화된 프로세스문 순으로 처리하여 현 시뮬레이션 사이클을 종료한다. 주어진 실시간 내에서 발생한 델타 시간 이벤트가 존재하면 실행 단계를 반복하게 되고 더 이상의 델타 시간 이벤트가 존재하지 않으면 시뮬레이션 시간을 증가시켜 다음의 실시간 이벤트를 처리하게 된다. 만약에 다음 시간의 이벤트가 더 이상 없거나 시뮬레이션의 최종 시간에 도달하면 시뮬레이션을 종료한다.

실행 단계에서 SVSIM은 계층적인 네트웍을 flatten하지 않고 instantiation될 때마다 컴포넌트 노드를 중복하여 sub C/DFG를 공유함으로 각 노드의 값이 어느 instantiation에 속한 값인가를 구분할 필요가 있다. 이를 위하여 instantiation되는 계층에 따라 각각의 컴포넌트에 유일한 구분자를 부여하고, 각 컴포넌트에서 호출되는 부프로그램 또한 구분자를 주어 이들 조합의 값으로 구분한다. 이와 같이 유일한 구분자 쌍과 각 객체 노드의 소속 정보를 이용하여 각 노드의 fanin, fanout 노드를 확정하므로 스택을 이용한 값의 전달 및 계층성의 추적 과정이 불필요하며 각 이벤트는 C/DFG의 구문 노드에 대한 정보를 가지고 있어 각 노드로부터 그 노드가 속한 구문 구조와 구문 구조내의 노드의 위치 정보를 알 수 있다.

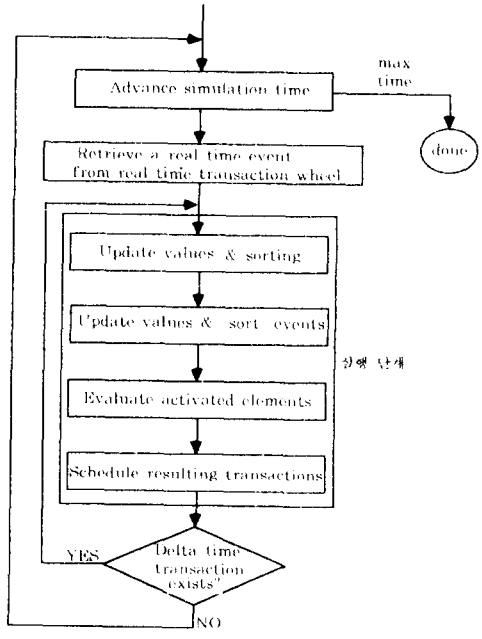

그림 5. SVSIM의 시뮬레이션 알고리듬
Fig. 5. Overview of the SVSIM algorithm.

VHDL은 지연 모델로서 관성 지연, 전달 지연, 델타 지연 모델을 지원한다.[19] 전달 지연은 선로 지연(wire delay)과 유사한 의미로 입력 신호의 변화를 일정시간 이후에 항상 반영하는 시스템을 기술할 때 사용하며 관성 지연은 입력의 변화된 값이 일정 시간 이상 유지되지 않으면 응답하지 않는 시스템을 기술할 때에 유용하다. 델타 지연은 실시간 지연의 개념은 아니지만 이벤트의 수행순서를 나타내는 지연 모델이다.[20] SVSIM에서 이들 지연 모델을 처리하는 알고리듬은 그림 6에 보였다.

시뮬레이션 시간 t_0에서의 한 시뮬레이션 사이클 중에 주어진 신호 객체 s에 지정이 일어났고 이 waveform의 각 element의 지연이 t_i라 하면 이들 waveform을 $\{ v'_i, t_i \}$ 의 집합으로 나타낸다. 만약 지정이 델타 지연이라면 waveform은 하나의 element로 구성되어 있고 지연 시간 $t_i = 0$이다. 지정의 값이 현재의 값과 다르고 다음 시뮬레이션 사이

클에 scheduling된 값이 없다면 scheduling한다. 만약 다음 시뮬레이션 사이클에 신호 s에 scheduling된 값이 존재하고 이 값을 v_{tk}라 하면 v'_{ti} 와 비교하여 같을 경우에는 v'_{ti} 의 scheduling이 취소되고 다르면 v_{tk}을 취소하고 v'_{ti}을 scheduling한다.

```
for every waveform { v'_n } for i = 1, 2, 3, ... , n
      to be scheduled in signal s in current simulation cycle at T_0
  if delta delay then
    begin
      -- assumption : the waveform consists of only one element --
      if v_tk ≠ v'_n then
          cancel v_tk and schedule v'_n ;
      else
          cancel v'_n
    end
  else if inertial delay t_ai then
    begin
      for every waveform element { v'_n }
        begin
          -- forward preemption --
          for every transaction v_tk s.t. t_k > t_i
              cancel v_tk ;
          -- backward preemption --
          for every transaction v_tk s.t. t_k - t_i ≤ t_ai and not in { v'_n }
            begin
              if v_tk ≠ v'_n then
                  cancel v_tk
            end
          schedule v'_n
        end
    end
  else if transport delay t_di then
    begin
      for every waveform element { v'_n }
        begin
          -- forward preemption --
          for every transaction v_tk s.t. t_k > t_i
              cancel v_tk
          schedule v'_n
        end
    end
```

그림 6. 지연 모델 처리 알고리듬

Fig. 6. Algorithm for processing delays.

지정이 관성 지연 t_a을 가지고 일어났고 v'_{ti}의 값이 현재의 값과 다르면 각 waveform element $\{v'_{ti}\}$에 대하여 scheduling된 시간 t_i보다 이후에 scheduling된 transaction중 현재의 지정되는 waveform에 속하지 않은 transaction은 모두 취소된다. Scheduling된 시간 ti이전에 scheduling된 transaction중 현재 지정되는 waveform에 속하지 않고 $t_k - t_i < t_a$인 transaction v_{tk}의 값이 v'_{ti}와 다르면 취소하고 같은 값이면 그대로 놔둔다. 이와같은 과정을 모든 waveform element v'_{ti}에 대하여 반복하여 실행함으로써 관성지연을 갖는 지정을 지원한다. 전달 지연의 경우는 관성지연에서 forward preemption과정만을 수행한다. 즉 scheduling된 시간 이후에 scheduling 되어 있는 transaction만이 취소된다.

3. 주요 구문의 처리

SVSIM은 표 1에서 보인 VHDL의 주요구문을 모두 지원한다. 행위적 병행문에는 조건 신호 지정문, 선택 신호 지정문, 병행 assertion 문, process 문 및 병행 procedure call 문이 있다. 먼저 조건 신호 지정문은 계층적인 C/DFG상에서 fork 노드로 표현되며 각각의 조건식에 있는 신호와 target의 각각의 조건에 대응되는 지정의 표현식에 포함되는 신호들이 활성화 이벤트 집합의 구성원이 된다. 활성화 집합에 있는 신호중 어느 하나에 이벤트가 발생하면 상호 배제적인 else-if 계열의 처음부터 수행하여 조건을 만족하는 지정의 표현식을 평가 지정하며 else-if 계열의 나머지 부분에 발생한 이벤트는 무시된다. 선택 신호 지정문은 sfork노드에 대응되며 조건의 표현식과 각 choice에 대응되는 waveform에 존재하는 신호들이 활성화 집합의 구성원이 되며 활성화 집합의 원소에 이벤트가 발생하면 sfork의 조건식을 평가하여 대응되는 waveform을 지정하고 이외의 waveform에 발생한 이벤트는 값의 갱신으로 처리가 완료되는 동시에 무시된다. 병행 assertion 문을 조건식에 포함된 신호원이 활성화 집합을 구성하며 활성화 집합의 구성원에 이벤트가 발생하면 실행된다. 병행 procedure call 문은 association의 actual 파라메타중에 존재하는 신호원이 활성화 집합의 원소가 되고 이벤트의 발생 시에 실행되며, 부프로그램의 처리는 스택 구조를 이용 처리하여 recursive call이 지원된다. Process 문의 활성화 집합은 process 문이 sensitivity list을 가지고 있을 경우에 sensitiviy list 의 element 가 활성화 집합을 구성하며 이들 중에 이벤트가 발생하면 수행된다. 만약 process문 내에 복수의 wait문이 존재하면 개개 wait문의 활성화 집합의 합집합이 process 문의 활성화 집합이 되며 process 문의 활성화는 suspended된 지점의 wait문의 활성화 집합의 원소중에 이벤트가 발생하고 이에 따른 wait문의 평가 결과에 따라서 resume되거나 다시 suspend 된다. Wait문의 수행은 명확한 sensitivity list가 있을시에는 이들이 활성화 집합을 구성하며, 명확한 sensitivity list의 부재시에는 조건식에 포함된 신호원이 활성화 집합의 구성원이 되고 sensitivity list 및 조건식이 모두 없을 시에 활성화 집합은 공집합이 된다. Wait문의 평가는 활성화 집합의 원소중에 이벤트가 발생하고 조건식이 없을 때 수행되며, 조건식이 있을 경우 조건식의 평가 결과가 TRUE 값일 때와 활성화 집합이 공집합이고 time-out 조건식이 만족할 때에 process문은 다시 실행된다.

컴포넌트 instantiation문은 port map에서 모드가 IN, INOUT, BUFFER, LINKAGE인 formal parameter에 대응되는 actual parameter 신호에 이벤트가 발생하면 binding된 컴포넌트의 내부가 실행된다.

기정의 attribute중 결과가 신호인 경우에는 prefix에 이벤트가 발생하면 attribute을 평가하고 이 결과에 따라 scheduling 여부를 결정한다. 순서문은 부프로그램의 body와 process 문 내에 올 수 있으며 이들의 실행은 부프로그램의 호출이나 process 문의 활성화때에 기술된 순서대로 수행되므로 wait 문을 제외한 순서문의 경우에는 특별히 따로 이벤트를 관리하지는 않으며 단지 값의 갱신만을 행하며 이를 표시해 둔다.

4. 입력 자극

입력 자극(input stimuli)은 시뮬레이션 대상회로에 원하는 테스트 입력을 가하는 것으로서 초기화 과정이 완료된 시점에서 대상회로에 링크된다. 설계자는 입력 자극의 형태를 기술한 입력 자극 파일을 만들고 이를 입력 자극 분석기를 통하여 분석한 다음 시뮬레이션 커널에 링크 시킨다. 입력 자극에서 사용될 수 있는 데이타 타입은 INTEGER, REAL, BITSTRING, STRING, TIME등의 타입이 있으며 이외의 신호 타입은 대응되는 신호의 타입으로서 별도의 선언 과정없이 입력 자극 파일에서 사용된다. 또한 입력 자극에서는 복잡한 제어구조를 지원하며 제어 명령으로서 for, while과 같은 반복적인 제어명령과 if문이 지원되고 logical, relational 및 arithmetic 연산자들이 지원되므로 입력 자극의 기술시에 이들의 연산결과로써 신호의 입력을 기술할 수 있어 입력 자극의 상대적인 시간 및 이들값의 일관성을 별도의 계산과정을 거치지 않고 쉽게 기술할 수 있다. Waveform 명령으로서 이벤트의 추가 및 삭제, 패턴의 추가 및 삭제등을 지원하며 이외의 방법으로 원시 코드상에서의 시뮬레이션 테스트 패턴의 기술 역시 지원한다.

V. 실험 결과

SVSIM은 UNIX 환경하의 SUN SPARC workstation상에서 약 25만 라인의 C언어로 구현하였다. 현재 SVSIM은 배취 명령에 의한 배취 시뮬레이션만이 가능하다. 배취 명령은 VHDL 원시 파일의 이름, working 라이브러리, resource 라이브러리, 시뮬레이션 종료 시간, 시뮬레이션 실행의 기본 단위 시간 및 시뮬레이션 실행 결과인 waveform 출력파일등을 지정하도록 되어 있으며, 입력 자극 파일 역시 지정한다. Waveform 출력 파일은 waveform

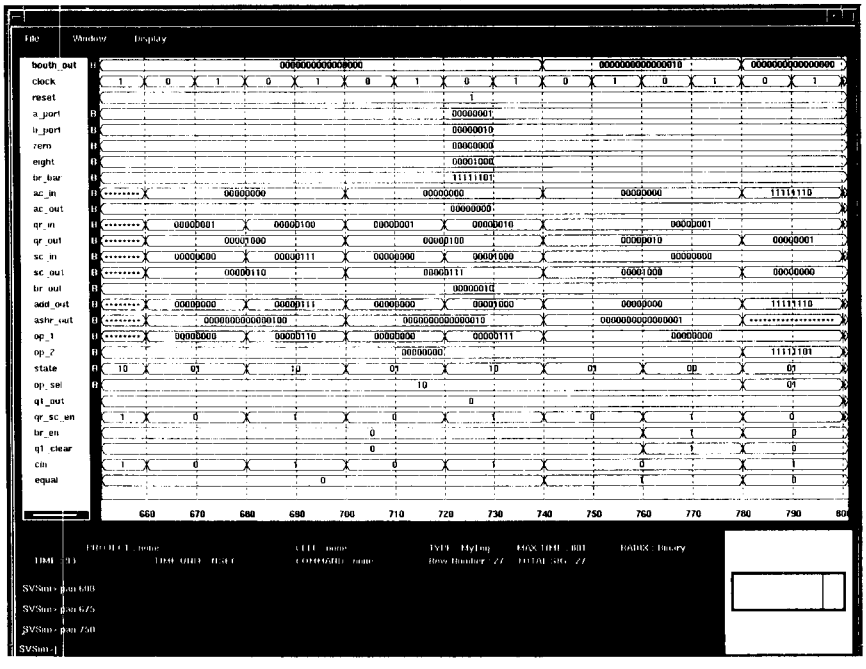

그림 7. 변형된 Booth 알고리듬을 사용한 곱셈기의 실행결과
Fig. 7. Simulation result of the multiplier using modified Booth's algorithm.

분석기을 이용하여 그래픽 디스플레이를 통하여 설계자가 실행 결과를 확인하도록 되어 있다. 실험 예제로서는 변형된 Booth 알고리듬을 사용한 곱셈기에 대한 기술을 이용하였다. 변형된 Booth 알고리듬을 사용한 곱셈기는 16개의 하위 컴포넌트를 이용하여 기술하였다. 그림 7은 IEEE의 표준 9 Value을 사용하여 시뮬레이션한 결과이다. a_port에 "00000001", b_port에 "00000010" 을 입력하고 최종적으로 곱셈의 결과가 booth_out에 "00000010" 로서 740 ns가 지난 후에 나옴으로써 실험결과가 올바름을 알수 있다. SVSIM과 Vantage사 시뮬레이터 Ver. 4.040 간[4]의 성능 비교는 컴파일 속도에 대하여서는 8개의 예제에 대하여 평균적으로 40배 정도 빠르며 시뮬레이션 속도면에서는 각각의 예제에 따라서 Vantage사 시뮬레이터 속도의 70 % - 15 % 정도의 시뮬레이션 속도를 가지고 있다.

Ⅵ. 결론 및 추후 과제

본 논문은 VHDL 설계 환경 구축의 일환으로 개발된 혼합 레벨 VHDL 시뮬레이터인 SVSIM의 설계 및 구현 결과에 대하여 기술하였다. SVSIM은 행위적, 구조적 및 이들의 혼합으로 기술된 시스템에 대한 시뮬레이션을 지원한다. SVSIM은 VHDL의 모든 기정의 attribute을 지원하며, RECORD 데이타 타입을 제외한 기정의 데이타 타입 및 사용자 정의 타입을 지원하고, recursive 부프로그램의 호출 및 계층적인 기술을 포함한 VHDL 주요구문을 모두 지원한다. 또한 SVSIM은 시뮬레이션 입력을 VHDL 원시코드내에서 또는 외부에서 명령 언어(command language)의 형태로 설계자가 줄수 있도록 설계되었으며, 외부의 입력 자극 명령은 for, while 루프와 if문을 지원하여 설계자가 일관성있게 입력을 줄 수 있도록 하였다.

SVSIM은 계층적인 기술을 flatten하지 않고 계층적인 C/DFG을 사용하여 공간 복잡도를 줄이고 스택 구조를 이용하여 recursive call을 지원하여 알고리듬 레벨에서의 시뮬레이션을 용이하도록 하였다. 따라서 상위 수준 합성 결과로 생성된 VHDL 구조기술에 대하여 시뮬레이션을 실행할 수 있고 top-down 방식의 계층적인 설계에서 구조의 세분화를 지원하므로 VHDL을 이용한 설계에서 일관적인 시뮬레이션이 가능하다. 실험결과 SVSIM의 시뮬레이션 결과는 Vantage사의 VHDL시뮬레이터와 비교하여 정확도를 확인하였다.

SVSIM은 CMOS, NMOS, dynamic logic 및 TTL 로직 게이트 레벨에서 필수적인 4, 9, 12 및 46-value system을 지원하며 특히 IEEE 표준 9-value system을 지원한다. 현재 SVSIM은 interactive 모드의 디버깅을 지원하고 있지는 않으나 break-point, step-by-step 실행 기능, trace 기능 및 신호값의 set기능등을 지원하는 디버깅 기능을 위한 확장과 user-interface의 확충이 계속되어야 한다. 또한 시뮬레이션 속도의 향상를 위한 내부 데이타 구조 및 알고리듬의 개발을 위한 노력이 계속되어야 할 것이다.

參 考 文 獻

[1] M. C. McFarland, "Tutorial on high-level synthesis," *In Proc. 25th DAC*, pp. 330-336, June 1988.

[2] A. Dewey and A. Gadient, "VHDL motivation," *IEEE Designs & Test of Computers*, vol. 8, no. 2, pp. 12-16, April 1986.

[3] J. H. Aylor, R. Waxman and C. Scarratt, "VHDL-feature description and analysis," *IEEE Design & Test Computers*, vol. 3, no. 2, pp. 17-27, April 1986.

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

[5] E. Meyer, "VHDL opens the road to top-down design," *Computer Design*, vol. 28, no. 3, pp. 57-60, Feb. 1989.

[6] *Viewsim/SD reference manual*, Viewlogic Systems Inc., Dec. 1989.

[7] 박 성범, 장 영조, 이 철동, "VHDL 시뮬레이터의 커널 구현," 과기처 '88 특정 연구결과 발표회 논문집, pp. 231-234, 1989년 7월.

[8] 오 대일, 김 헌철, 이 형종, 황 선영, "SVSIM: 서강 VHDL 시뮬레이터," 대한 전자공학회 하계 종합학술대회 논문집, 제 14권 제 1호, pp. 533-537, 1991년 6월.

[9] 김 헌철, 이 영희, 황 선영, "계층적인 시뮬레이션과 합성을 위한 VHDL 중간 형태에 관한 연구," 대한 전자공학회 학술발표회 논문집, 제 10권 제 1호, pp. 86-89, 1992년 6월.

[10] 이 영희, 황 선영, "VHDL 설계 환경 구축을 위한 front-end의 설계," 한국 정보 과학회

논문지, vol. 18 , no. 1, pp. 93 103, 1991년 1월.

[11] A. V. Aho, R. Sethi and J. D. Ullman, *Compilers*, Addison Wesley: Reading M. A., 1988.

[12] M. Odani, S. Y. Hwang, T. Blank and T. Rokicki, "The ILSP behavioral description language and its graph representation for behavioral synthesis," *Stanford Univ. Technical Report:* CSL-TR-88-350, March 1988.

[13] M. Abramovici, M. A. Breuer and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press: New York, 1990.

[14] P. Michel, U. Lauther and P. Duzy, *The Synthesis Approach to Digital System Design*, Kluwer Academic Pub.: Boston, 1992.

[15] G. Russell, D. J. Kinniment, E. G. Chester and M. R. McLauchlan, *CAD for VLSI*, Van Nostrand Reinhold Co. Ltd : Berkshire, 1985.

[16] R. Lipsett, E. Marschner, and M. Shahdad, "VHDL - the language," *IEEE Design & Test Computers*, vol. 3, no. 2, pp. 17-27, April 1986.

[17] *IEEE Standard VHDL Language Reference Manual*, IEEE 1076-1987, April 1989.

[18] R. E. Harr and A. G. Stanculescu, *Application of VHDL to Circuit Design*, Kluwer Academic Pub.: Boston, 1991.

[19] R. Lipsett, C. F. Schaefer, and C. Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Pub.: Boston 1989.

[20] J. R. Armstrong, *Chip-Level Modeling with VHDL*, Prentice Hall: N.J., 1989.

著者紹介

黃善泳(正會員)

1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국 과학원 전기 및 전자 공학과 공학 석사 취득. 1986年 10月 미국 Stanford 대학 공학 박사 학위 취득. 1976年 ~ 1981年 삼성 반도체 주식회사 연구원. 1986年 ~ 1989年 Stanford대학 Center for Integrated Systems 연구소 연구원, Fairchild Semiconductor Palo Alto Research Center 기술 자문. 1989年 3月 ~ 현재 서강 대학교 전자공학과 교수. 주관심 분야는 CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임.

李榮熙(準會員)

1989年 2月 서강대학교 전자공학과 졸업(공학사). 1992年 2月 동 대학원 공학 석사 학위 취득. 1993年 2月 ~ 현재 동 대학원 박사 과정 재학 중 주관심 분야는 CAD system 및 혼합 모드 시뮬레이션 알고리듬 등임.

金憲哲(準會員)

1991年 2月 서강 대학교 전자공학과 졸업(공학사). 1993年 2月 동 대학원 공학 석사 학위 취득. 1993年 2月 ~ 현재 삼성 반도체 부천 연구소 연구원. 주관심 분야는 CAD system 및 VLSI Testability 등임.