論文93-30A-12-15

하드웨어 설계 라이브러리 서버의 구현에 관한 연구

(A Study on the Implementation of a Hardware Design Library Server)

崔 益 成*, 李 榮 熙*, 黃 善 泳**

(Ick Sung Choi, Young Hee Lee and Sun Young Hwang)

要 約

본 논문은 VHDL 설계 환경에서 설계 라이브러리 지원을 위한 CAD 객체 관리 시스템의 설계와 구현에 대하여 기술한다. 제안된 라이브러리 서버 시스템은 설계 객체의 효율적인 관리를 위하여 라이브러리를 시스템과 독립적으로 분리하였다. CAD 툴이나 사용자에 의한 설계 객체의 수정이 가능하도록 서버 명령어를 배치 스크립트로 받아들이거나 X/MOTIF 상의 서버를 통해 작업이 가능하도록 하였다. 그리고 설계 객체의 종속 관계와 버전 및 multiple view 등의 관리 뿐만 아니라 각 설계 객체와 버전에 대한 도큐멘테이션, 속성, method도 함께 관리할 수 있도록 구현되었다. 실험 결과를 통하여 시스템이 VHDL 설계 환경 하에서 구현된 하드웨어 설계 라이브러리를 이용하여 하드웨어 자동설계를 효과적으로 지원함을 보였다.

Abstract

This paper describes the design and implementation of a CAD object management system for the design library in VHDL design environment. For the efficient manipulation of design objects, the library is managed independently of the underlaid CAD system. Management and revision of design objects can be performed by a batch script of server commands or through the user interaction in the X/MOTIF graphic environment. Through the library server, design management tasks can be efficiently performed, such as configuration of design objects, version control, and management of attributes and methods for versions. Experimental results show that the proposed system is a viable tool for the management of design data in VHDL design environment.

Ⅰ. 서론

VLSI 제조 기술의 발달과 더불어 칩의 집적도가 증가하고 칩의 설계 시간이 현저히 길어짐에 따라 새로운 설계 방법론이 대두되기 시작하였으며 기존의 bottom-up 설계 방법에 비하여 시스템 설계를 보다 높은 레벨에서 설계자가 알기 쉬운 프로그래밍 언어와 같은 방식으로 동작 사양을 기술하면 자동적으로 하드웨어를 생성해 주는 top-down 설계 방법이 대두되었다.[1, 2] Top-down 설계 방법은 주어진 사양을 만족하는 설계 공간의 탐색이 보다 높은 레벨에서 이루어지므로 설계 작업이 용이하며 설계의 진행에

*準會員, **正會員, 西江大學校 電子工學科
(Dept. of Elec. Eng., Sogang Univ.)

接受日字 : 1993年 1月 21日

따라 각 부분 시스템에 대한 구체적인 사양이 보다 명확해 진다. 한편 top-down 방식의 문제점으로 규모가 큰 시스템의 설계시에 각각의 부분에 대한 설계의 진행 정도가 일치하지 않으며 따라서 설계 객체의 효율적인 관리 문제의 중요성이 대두되었다.

칩의 집적도 증가와 더불어 설계된 CAD 객체의 관리와 이미 존재하는 설계 데이타의 효율적인 재사용의 중요성이 증가하여 설계 객체들간의 복잡한 상호 의존 관계 및 하나의 설계 단위에 대한 다양한 레벨의 기술 등을 효과적으로 관리하고 설계의 진척에 따른 다양한 버전을 관리하며 이들 사이의 유효성과 일치성 등을 검사하여 주는 툴의 필요성이 증대하였다. CAD의 객체는 다른 객체들과는 달리 객체들 사이에 다음과 같은 관계를 가진다. [3] [4]

첫째, CAD 객체는 보다 기본적인 객체들로 구성이 되는 종속 구조(hierarchical structure)를 가진다. 이는 보통 CAD 객체가 다른 객체를 참조하여 포함하는 경우가 많기 때문이며, CAD 객체가 점차 보다 아래 레벨의 세부 객체로 구성되는 것을 configuration이라고 한다. [3] 한 콤포넌트 객체는 다른 객체들을 참조할 수 있기 때문에 공유가 가능한 CAD 객체들로 구성된 DAG(direct-acyclic-graph)형태의 CT(configuration tree)로 종속구조를 표현하고 관리한다.

둘째, 하나의 CAD 객체는 여러개의 버전을 가질 수 있으며 한 버전에서도 여러개의 버전이 파생될 수 있어 이 관계가 효과적으로 관리되어야 한다. [4] [5] 버전의 구분은 애매모호해서 다양한 정의가 있을 수 있으나 여기서는 같은 입출력 동작을 하면서 서로 다른 내부 구조/행위를 가지는 것으로 정의한다. 모든 설계 단위는 버전을 가지며 한 설계 단위가 참조되는 경우에는 버전 중 하나를 선택해서 결합을 해주어야 한다. 그리고 버전은 고유한 종속 구조를 가질 수 있기 때문에 버전 결합 방법에 따라 전체 종속 구조가 바뀔 수 있다.

세째, 보는 관점에 따라서 하나의 동등한 객체가 여러가지 표현들로 나타내어질 수 있다. 예를 들어 하나의 VLSI 객체는 레이아웃, 논리 회로, 혹은 레지스터 전송 수준의 표현으로 나타내어질 수 있으며, 논리 회로 수준에서 트랜지스터 수준의 표현이 생성되는 경우처럼 하나의 표현에서 다른 표현이 생성되는 경우가 있다. [6]

네째, CAD 객체는 구현(implementation)에 속하는 부분과 접속관계(interface)에 관계된 부분으로 구분할 수 있다. 구현에 관계된 부분은 입출력 그리고 구성 회로 모듈의 기능을 포함하는 부분이며, 접속 관계는 자체의 외부와의 접속관계와 구성 성분 소자들의 연결로 이루어져 있다. VHDL의 경우에 객체는 각 프로시듀어에 해당하고, 프로시듀어의 접속 관계는 파라메터 리스트에 해당하며, 구현 부분은 코드 세그먼트에 해당된다.

VHDL에서는 설계 단위가 관리되는 방법에 따라 primary unit와 secondary unit 두 가지로 구분한다. Primary unit에는 엔터티 선언, 엔터티 본체, 패키지 선언, 패키지 본체, configuration으로 구성되며 secondary unit에는 아키텍춰 등이 포함된다. [7] VHDL에서의 모든 설계 단위의 검색은 primary unit 내에서 할 수 있다. Primary unit와 secondary unit사이의 관계는 일 대 다수의 매핑이지만 이것은 primary unit에 있는 configuration을 살펴봄으로써 각 primary unit에 있는 설계 단위들에 대응되는 secondary unit들을 알 수 있다. 따라서 라이브러리 서버에서 검색할 때는 primary unit와 secondary unit를 별도로 구분할 필요없이 primary unit만 관리하면 된다. 이 때 secondary unit에서 기능상으로는 필요가 없지만 분류상의 목적으로 같은 엔터티를 가지는 아키텍춰 등을 분류해야 할 필요는 존재하게 된다.

VHDL 언어에서 보통 한 엔터티에 여러가지 아키텍춰가 대응되므로, 이 각각의 아키텍춰들은 하나의 넓은 의미의 버전으로 간주할 수 있다. [7] [8] 버전 파생 관계는 버전 파생 트리(VDT: Version Derivation Tree)의 형태로 나타내고, 또 각 버전은 버전 넘버라는 유일한 정수로 구별되는 것이 보통이다. VHDL 기술에서 이러한 버전 넘버 등을 별도로 관리하는 방법도 있으나, 아키텍춰 이름에 버전을 나타내는 방법도 가능하다. 구조 기술의 VHDL 기술은 많은 경우에 CAD 툴에 의해서 자동으로 기술되기 때문에, 버전 파생은 CAD 툴에 의해서 수행되는 경우가 대부분이다.

기존의 구현 예에서는 CAD 객체를 일반 설계 단위로 나타내고 같은 설계 단위임을 나타내는 equivalence 기준에 따라 분류하여 나타내었으나 [5] [8] 단순히 equivalence라는 CAD 객체의 동등성 뿐만 아니라 이들 설계 객체의 구체적인 view의 설정이 요구되고, 최근에는 혼합 레벨 시뮬레이션처럼 설계가 각기 다른 design view를 가지는 설계단위들로 구성되는 경우가 많아져 기존의 방법으로는 design view에 따라 검색하기가 어렵기 때문에 새로운 표현 방법이 필요하다.

CAD 객체를 단순히 종속 구조, 버전, multiple view의 세 기준에 따라 분류하여 관리하는 방법이

있으나, CAD 객체들의 필요없는 분류 작업과 관리가 필요하다. 예를 들면 한 설계단위에는 여러개의 버전이 파생될 수 있지만 한 버전 파생 트리내의 버전들이 서로 다른 view를 가지는 경우는 없기 때문에 design view에 따라 설계 단위를 분류할 수 있으나, design view에 따라 버전을 관리할 필요는 없다. 또 한 버전 파생 트리내의 모든 버전은 대응되는 설계단위와 자식 설계 단위와 관계가 있어 각 버전마다 모든 설계단위를 관리하는 일은 무의미하다. 따라서 이러한 경우에는 CAD 객체들에 대한 분류와 관리를 배제할 수 있다.

기존의 구현 예로 SDE VHDL 설계 지원 환경에서는 CAD 객체를 입출력에 따라 구분한 설계단위와 유사한 CELL, 유사한 기능 별로 구분한 CLASS, 구현 방법에 따라 구분된 버전과 유사한 DESIGN으로 구분하여 관리한다.[4] [8] 이러한 라이브러리 서버들은 라이브러리 외부와 프로시져를 이용해서 입출력을 하므로 라이브러리를 이용하는 시스템은 라이브러리와 같이 컴파일해서 구축해야 한다. 그리고 이러한 환경에서는 별도로 CAD 객체를 표현하고 관리하는 방법이 없다. Vantage, Mentor 등의 상용 설계 시스템에서 구현된 버전은 단순히 수정 시간에 따른 저장이므로 버전 관리 기능보다는 확장된 백업 기능이라고 할 수 있고, view의 구분도 파일 이름의 확장자에만 의존하므로 view를 관리한다고 보기는 어렵다.

VHDL 설계 지원 환경의 일원으로 본 연구실에서는 상위 수준 합성기, 혼합레벨 VHDL 시뮬레이터, 조합 논리회로 최적화 시스템 및 레지스터 트랜스퍼 레벨 합성기를 개발중에 있거나 이미 개발하여 사용하고 있으며 이들 툴들을 사용하여 설계 및 시뮬레이션을 행하고 있다.[9] [10] [11] 제안된 시스템에서는 CAD 객체를 설계 단위로 나타내고 설계 단위들 간에 참조 관계를 CT로, 각 설계단위마다 파생된 버전들을 버전 파생트리로 나타내고 view사이의 관계를 VDG(View Dependency Graph)로 설계하였다. 서로 다른 view를 가지는 동일한 설계 단위들은 VDG를 탐색하여 찾아내고 이를 이용하여 서로 다른 view를 참조할 수 있도록 제안하였다. 한편, 설계 객체의 효율적인 관리를 위하여 설계 객체의 모델을 시스템과 독립적으로 분리하여 다양한 설계 객체 관리와 이들의 변경이 용이하도록 설계하였고 CAD 툴이나 사용자에 의한 설계 객체의 수정이 가능하도록 서버 명령어를 배치 스크립트로 받아들이거나 공유 메모리를 통하여 CAD 툴들과 정보교환을 하며 X/MOTIF 상의 서버를 통해 작업이 가능하도록 하였다. 그리고 설계 객체의 종속 관계와 버전 및 multiple view 등의 관리 뿐만 아니라 각 설계 객체와 버전에 대한 도큐멘테이션, 속성, method도 함께 관리할 수 있도록 구현되었다.

Ⅱ. 전체 시스템 구성

하드웨어 설계 라이브러리는 CAD 객체를 저장하고 있는 physical storage와 이를 관리하는 라이브러리 서버의 두 부분으로 이루어져 있다(그림 1). 라이브러리 서버는 설계 단위와 버전의 관리와 이들의 속성 및 method 등을 관리하는 부분이다. 그림 2는 하드웨어 설계 라이브러리의 적용 환경을 나타낸 것이다. 여기서 VHDL 분석기는 VHDL 소스 코드가 올바른 문법과 의미를 가지고 기술되어 있는 지의 여부를 검사하여 중간 코드를 생성하고 역 분석기(reverse analyzer)는 이 생성된 중간 코드로부터 VHDL 소스 코드를 다시 복원한다.[9]

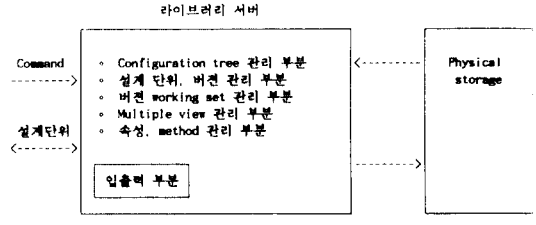

그림 1. 라이브러리의 개략적 구성
Fig. 1. Overall configuration of the design library.

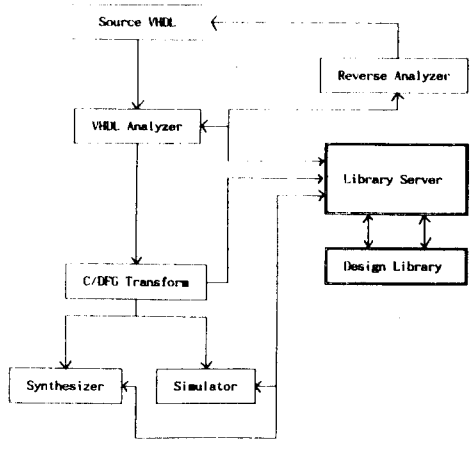

그림 2. VHDL 설계 지원환경
Fig. 2. VHDL design environment.

Ⅲ. 하드웨어 설계 라이브러리 서버

하드웨어 설계 라이브러리 서버의 기본 기능은 설계에 필요한 적절한 설계 단위를 찾아 합성 시스템에게 제공해 주는 것으로, 필요로 하는 설계 단위의 이름, 설계가 들어있는 프로젝트 혹은 데이터 베이스의 이름, 버전 넘버, view 등을 입력으로 받고 이에 해당하는 설계 단위를 제공한다. 서버의 구성은 그 기능에 따라 표 1과 같이 크게 9 부분으로 나누어진다. CAD 객체의 종속 구조, 버전, multiple view의 세 가지 특징 중에서 설계 단위들 사이의 종속 구조는 CT로 나타내어 설계 단위들 사이의 참조나 버전 결합 관계 등의 정보를 담는다. 그리고 각 버전은 버전 파생 트리에서 노드로 나타내고 버전들 사이의 파생 관계는 에지로 나타낸다. 그 외 전체적으로 작업마다 사용하는 working set은 사용하는 모든 설계단위들의 정보를 관리하기 위해 사용한다. 한 설계 단위가 참조되었을 경우에는 직접 참조가 가능한지, 혹은 physical storage나 다른 곳에서 읽어들이는 등의 작업을 거쳐야 참조가 가능한 지를 결정하고 처리해야 한다. 이를 위해서 라이브러리 서버가 설계 단위들의 working set을 이용하여 모든 설계 단위들의 연결에 관한 작업과 설계 단위의 수정에 의한 CT와 VDG 상에서 전파에 관련된 작업들을 함께 처리한다. 그 외 각 설계 단위마다 속성과 method들로 이루어진 집합을 가진다. 버전과 그 대응되는 내용도 속성의 특별한 경우이지만 CAD에서는 특별히 많이 쓰이므로 속성으로 취급하지 않고 별도로 다루었다. 설계 단위의 속성과 method를 등록하고 삭제하는 작업, 한 설계 단위의 속성과 method의 집합을 다른 설계 단위에 복사하거나 덧붙이는 작업 등도 이 부분에서 같이 처리한다.

표 1. 라이브러리 서버의 기능
Table 1. Functions of the library server

1. Configuration tree를 구성하고 관리하는 부분.
2. 버전의 working set을 관리하는 부분.
3. 설계 단위의 관리에 필요한 모든 정보들의 physical storage와의 입출력에 관련된 부분.
4. 설계 단위와 버전을 구성하고 관리하는 부분.
5. 캐드 툴 명령어 인터프리터
6. 서버 명령을 수행하는 부분.
7. Multiple view를 관리하는 부분.
8. 각 설계 단위에 필요한 속성과 method를 관리하는 부분.
9. 사용자 인터페이스 부분.

한 설계 객체는 여러 설계 단위를 이용해서 구성되는데 이를 CT로 나타내며 CT의 각 노드는 설계 단위와 버전의 한쌍을 가르킨다. 버전 파생을 나타내는 버전 파생 트리의 각 버전은 고유한 설계 단위의 종속 관계를 가질 수 있고 버전 넘버, 수정 시간, 사용자 이름 등의 정보를 담고 있다. 이러한 종속 구조를 그림 3에 나타내었다.

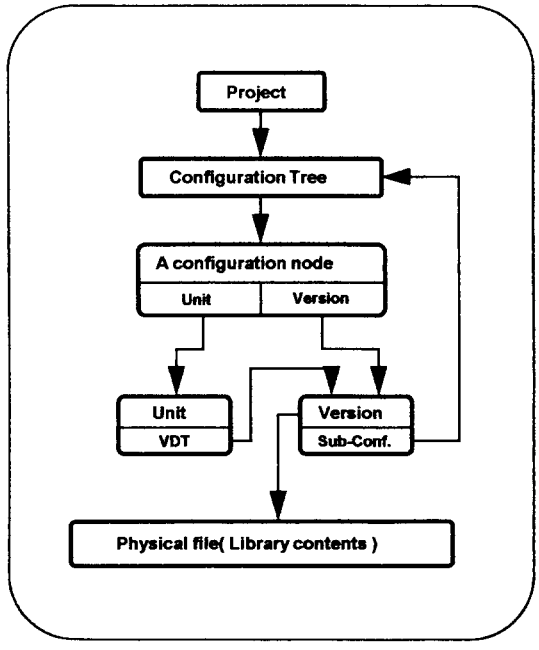

그림 3. CT의 종속 구조
Fig. 3. Hierarchy of configuration tree.

한 설계 단위가 참조되면 자손에 해당하는 모든 설계 단위들을 함께 필요로 하는 경우가 대부분이다. 만일 설계 단위의 수정에 의한 전파가 생긴 경우에는 VDG와 CT를 참조하여 처리할 수 있다. 이를 위해서 라이브러리의 명령어 중에는 설계 단위의 모든 자손 설계 단위들을 구하는 method와 변경된 설계 단위를 지정하고 원하는 설계 단위의 수정에 필요한 CAD툴들을 구하는 method 등이 있다. 한 설계 단위의 instance를 나타내는 CT에서의 노드는 그림 4와 같은 정보를 담고 있다.

Working set은 사용자가 프로젝트 수행시 사용하는 모든 설계 단위들을 담고 있는 집합을 가르킨다. 중단된 작업을 계속하거나 다른 곳에 저장되어 있는 설계 단위를 불러서 사용하는 경우는 physical storage에서 프로젝트에 관련된 모든 정보를 다시 불러오며, 설계 단위를 불러올 때마다 working set이

```
/*  This is the node structure of Configuration Tree  */

typedef struct ConfNode {

    int id;          /*  Configuration node identifier  */

    /*  Unit related informations  */

    char *name;      /*  Name of unit  */

    /*  Version related informations  */

    int VerNo;                      /*  Version number                  */
    struct vUnitNode *RootOfVDT;    /*  Root of version derivation tree */
    struct vUnitNode *VerPtr;       /*  Current version pointer in VDT  */

    /*  Additional informations  */

    char MV[50];                    /*  View of unit */
    int weight;
    Widget button;

    struct ConfNode *SubConfNode;   /*  Unit configuration of a version */
    struct ConfNode *child;         /*  Links across hierarchy          */
    struct ConfNode *sibling;       /*  Links between sibling           */
    struct ConfNode *bchild;        /*  Backward links                  */
    struct ConfNode *bsibling;      /*  Backward links
                                        between super node & sub-node   */
    int is_visited;

    } ConfNode ;
```

그림 4. CT의 configuration 노드 데이타 구조
Fig. 4. Data structure of a configuration node in CT.

다시 구성된다. 작업 중에 사용된 설계 단위는 working set에 기록이 되어, 다시 설계 단위를 참조하는 경우가 발생하면 working set내에 등록되어 있는 지를 살펴보고 그 버전을 비교해 보아서 지금 필요로 하는 설계 단위를 찾아 결합 작업을 하며, 이 때 설계 단위에 적합하도록 CT를 수정한다. 찾은 설계 단위가 working set내에 기록되어 있지 않은 경우에는 physical storage에서 설계 단위를 읽은 후에 working set에 이 설계 단위가 이미 참조되었다는 것을 기록한다. 작업을 마치면 CT의 각 노드에 해당하는 모든 설계 단위들의 버전을 저장한다.

설계 단위를 저장할 때에는 우선 각 프로젝트마다 사용하고 있는 라이브러리와 그 안에서 참조된 설계 단위를 모두 모아서 현재 프로젝트에 관련된 모든 설계 단위들에 대한 정보와 종속 관계를 기록한다. 한 설계 단위를 저장하라는 요구가 발생하면 CT와 설계 단위 테이블 안의 모든 수정된 설계 단위의 정보와 설계 단위가 함께 physical storage에 저장된다. 만일 설계 단위가 생성되었거나 새로운 버전 파생이나 삭제가 일어난 경우에는 관련된 버전 파생 트리가 함께 저장이 된다. 이러한 일련의 서버에 관련된 작업이 끝나게 되면 설계 단위의 실제 내용을 프로젝트별로 분류하여 physical storage에 저장한다.

실제 설계 단위는 다른 설계 단위들을 이용해서 구성된다. 매번 설계 단위를 참조할 때마다 physical storage 에서 불러오면 참조하는 횟수만큼 설계 단위가 중복되어 존재할 수 있다. 이를 방지하기 위해서 각 사용자나 프로젝트마다 설계 단위의 working set을 운용하여 참조하는 정보들을 관리하고 실제로 참조되는 설계 단위의 내용은 필요할 때만 physical storage에서 불러온다.

CT가 physical storage에 저장될 때는 각 프로젝트마다 디렉토리를 만들고 Configuration이라는 파일 이름으로 그림 5와 같은 형식으로 저장된다.

ID 설계단위 이름 버전 넘버 view 자식 설계 단위들 관련 정보 파일이름

그림 5. CT를 디스크에 저장하는 형식
Fig. 5. File format of CT.

다음 그림 6에 보인 바의 카운터 설계단위 예의 configuration에 대한 CT는 그림 7의 파일 형태로 저장이 된다. 그림 6은 3 비트의 입력 중 1의 갯수를 헤아려서 2 비트의 출력을 내는 카운터로 버퍼, majority를 구하는 MAJ3 게이트와 odd parity를 구하는 OPAR3 게이트로 구성되어 있다. MAJ3 게이트는 VHDL 구조 기술로 기술되어 있고 나머지 게이트와 버퍼는 VHDL 행위 기술로 기술되어 있다.

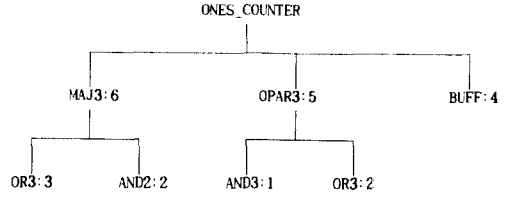

그림 6. CT의 예
Fig. 6. An example of a CT.

```
C1  AND3          1  VHDL                    -INF:C1.inf
C2  AND2          2  VHDL                    -INF:C2.inf
C3  OR3           1  VHDL                    -INF:C3.inf
C4  OR3           2  VHDL                    -INF:C4.inf
C5  BUFF          4  VHDL                    -INF:C5.inf
C6  OPAR3         5  VHDL  C4  C1            -INF:C6.inf
C7  MAJ3          6  VHDL  C3  C2            -INF:C7.inf
C8  ONES_COUNTER  0  VHDL  C7  C8  C6        -INF:C8.inf
```

그림 7. 카운터 CT를 저장한 파일
Fig. 7. CT file for ONES_COUNTER.

한 설계 단위를 physical storage에서 읽어들이는 과정은 다음과 같다. 우선 한 설계 단위가 참조되면 이를 불러오는 요구가 생기고 이 때 설계 단위의 이름만 주어지는 경우도 있고 프로젝트 이름이나 버전이 함께 지정되는 경우도 있다. 설계 단위의 이름만 지정되는 경우에는 현재의 프로젝트에서 default 버전을 찾아서 결합하며, 프로젝트 혹은 라이브러리 이름이 지정된 경우에는 지정된 프로젝트/라이브러리의 설계 단위를 찾아서 default 버전으로 결합한다. 설계 단위의 이름과 버전이 주어진 경우에는 현재 프로젝트 내의 설계 단위 이름을 찾아서 지정된 버전을 결합한다. 그 다음 과정에는 각 프로젝트마다 관리하는 설계 단위 테이블에서 설계 단위를 찾고 이 설계 단위들 사이의 참조 관계를 찾아서 CT를 구성한다. 이때 CT에서 하나의 설계 단위를 추가하면 설계 단위의 이름과 이를 참조할 수 있도록 working set이 수정된다. 설계 단위의 참조에 사용이 되는 모든 포인터는 레이블로 표현하고 이를 가르키는 포인터와 함께 리터럴 테이블에서 관리한다. Physical storage에 저장될 때에도 설계 단위들 사이의 참조 관계를 레이블을 이용하여 나타낸다.

Physical storage에 있는 설계 단위 테이블을 읽어서 CT를 구성할 때에 정의되지 않은 프로시져 혹은 객체들이 참조되는 forward reference 문제가 발생한다. Forward reference는 아직 physical storage에서 읽어 들이지 않아서 설계 단위가 참조될 준비가 되지 않았는 데도 이를 참조하는 경우에 발생한다. Forward reference를 처리해줌으로써 설계 단위를 읽어들이는 순서를 정하는 등의 많은 전처리 과정을 생략할 수 있게 되고 읽어들이는 그래프가 사이클이 있는 경우에도 적용할 수 있게 된다.

Forward reference의 처리를 위하여, 레이블이 참조되면 우선 이 레이블이 나타내는 설계 단위가 유효한 지를 알아본 뒤 레이블에 해당하는 설계 단위를 연결하지 못하면 레이블을 사용한 곳의 위치를 리터럴 테이블에 저장하여 나중에 레이블이 유효해질 때 이곳을 참조한 곳을 수정할 수 있도록 예약을 해둔다. 매번 설계 단위를 읽어들일 때마다 리터럴 테이블을 살펴보아서 forward reference 예약이 되어있으면 예약된 부분을 찾아서 수정한다.

Ⅳ. 버전과 multiple view 관리

하나의 설계 단위가 생성되었을때 새로운 설계 단위의 생성인지 혹은 버전 파생관계인지를 결정하는 여러가지 기준이 있지만, 버전 파생은 같은 입출력 동작을 하면서 서로 다른 내부 구조/행위를 가지는 것으로 정의한다.[3] 버전에 따라 서로 다른 설계 단위들은 고유한 종속 구조를 가질 수 있다. 유사한 행위를 가지는 모든 객체의 관계를 버전 파생으로 처리하는 것이 보다 일반적인 구현이 되겠으나 편의상 구현 방법이 다른 것으로 한정하고, 모든 버전은 같은 입출력을 가진다고 가정한다.

모든 설계 단위는 버전을 가질 수 있으며 한 설계 단위가 참조되는 경우에는 버전 중 하나를 선택해서 결합을 해주어야 한다. 버전의 결합 작업은 정적 결합 방법과 동적 결합 방법이 있으나, 어떤 경우이든 존재하는 버전들 중 하나만을 선택해야 한다.[5] 정적 결합 방법은 각 객체가 참조될 때 결합되는 버전을 지정해 주는 방법이고 동적 결합 방법은 시스템이 자동적으로 default 버전 등을 이용해서 버전 결합 작업을 하는 방법이다. 이때 쓰이는 동적 결합은 사용자가 직접 지정해 줄 수도 있고 혹은 가장 최근에 생성이 된 버전을 사용할 수도 있다.

그 외 버전 파생의 방법이나 구분의 기준도 필요하다. 버전은 다른 버전을 파생할 수 있고 동작이 확정된 working version과 현재 작업 중이고 아직 동작이나 그 외의 것들이 확정되지 않은 temporary version으로 구분된다. 모든 버전은 다른 라이브러리의 working version이나 다른 프로젝트에 있는 working version 또는 프로젝트에서 진급시킨 working version에서만 파생이 된다. 즉 temporary version을 진급시키거나 다른 working version을 대체함으로써 working version을 만들 수 있다.

버전 파생의 방법은 각 버전에 저장되는 내용에 따라 다르다. 많은 CAD 작업들이 CAD 툴에 의해 수정되므로 사용자가 수정을 한 후 버전 파생을 할 것인가를 결정하는 부담을 덜기 위해, 버전 파생의 방법은 그 설계 환경에서 지원하는 CAD 툴들에 의해 결정된다. 진급의 방법으로 사용자가 강제로 진급하는 명령을 줄 수도 있고 혹은 자동으로 진급이 되는 경우도 있다. VHDL의 secondary unit 기술은 버전에 대응된다. 앞에서 언급했듯이 VHDL의 아키텍춰는 CAD 에서 버전의 용도를 포함하며, 버전의 내용은 VHDL을 파싱한 결과 구성된 AST(Abstract Syntax Tree)가 될 수 있다. 버전 파생 트리에서 버전이 파생될 때마다 버전은 유일한 버전 넘버를 가지고, 버전의 저장시에는 버전 넘버를 이용하여 버전을 구분한다. 실제 설계에서는 한 설계 단위에 대해서 하나의 버전만을 사용하는 것이 대부분의 경우이겠지만 여러 프로젝트에서 참조하려면 각 프로젝트마

다 임의의 버젼을 동시에 결합할 수 있어야 한다. 여러 사용자가 라이브러리나 프로젝트를 공유하는 경우에 한 사용자가 어떤 임의의 설계 단위를 사용하지 않는다고 해서 지워버리거나 수정할 수 없는 경우가 있다. 즉 다른 사용자나 다른 프로젝트에서 이 설계 단위를 동시에 참조하는 경우 등이 해당되며, 이 설계 단위가 다른 어떤 프로젝트에서도 사용되고 있지 않은 경우에만 이 설계 단위를 지울 수 있다. 이러한 경우를 처리하기 위하여 각 버젼마다 현재 설계 단위가 참조된 횟수를 나타내는 카운터를 관리하여 카운터의 값이 1보다 작은 값을 가지는 경우에만 실제로 physical storage에서 삭제한다. 각 버젼의 설계 단위는 그림 8의 정보를 갖게 된다.

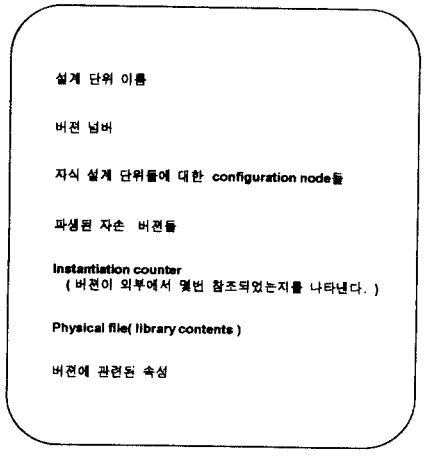

그림 8. 버젼 파생 트리의 버젼 내용
Fig. 8. Contents of a version in VDT.

버젼 파생 트리는 설계 단위에 해당하는 최초의 버젼이 만들어질 때 생성되고 버젼의 파생이나 삭제, 버젼의 진급 등에 의해 수정된다. 만일 설계 단위에 버젼 파생이 없으면 버젼 파생 트리가 physical storage 에 저장되지 않고, 버젼 파생이 있었다면 각 설계 단위마다 버젼 파생 트리를 가지게 된다. 버젼의 파생을 나타내는 방법으로는 파생되기 전의 버젼과 파생된 버젼이 어떤 차이가 있는지를 각 버젼 파생 트리의 에지마다 지정해주는 방법과, 노드에 해당하는 설계 단위의 버젼마다 다른 버젼과 구별되는 특성을 지정해주는 두 가지 방법이 있으나, 전자의 경우 각 버젼을 파생되는 버젼의 정보와 함께 관리하는 것은 자연스럽지 못하다. 따라서 각 버젼사이의 파생을 나타내는 에지의 정보는 두 버젼 중 버젼 파생 트리에서 트리의 깊이가 깊은 쪽에 저장하는 방법이 보다 적절할 것이다. 본 시스템에서는 위 두가지 방법 중 특별히 한 방법을 택하지 않고 사용자가 편의에 따라 사용할 수 있도록 하였다.

Design view에 대한 정보는 각 설계 단위에서 관리되고, 서버에 등록된 여러 design view 중 하나의 view를 가진다. Design view 사이의 관계는 directed graph 형태의 VDG로 나타내는데 여기서 노드는 view나 부가 자료를 나타내고 에지는 view사이의 변환에 필요한 과정을 나타낸다. VDG를 이용하면 자동 설계 시스템에서 CAD 객체의 변화에 따른 수정 작업의 전파 처리를 설계자의 개입없이 지원할 수 있고 혼합 레벨 시뮬레이터 등에 사용되는 multiple view를 직접 관리할 수 있다.

객체 지향 개념에서는 설계 단위가 객체에 해당하고 한 버젼은 객체의 instance에 해당한다.[12] 또 small-talk의 object/type 메카니즘에서 설계 단위는 type에 대응되고 각 intance들은 객체에 대응된다고 볼 수 있다.[3] 버젼의 설정 방식에 따라 속성의 집합은 버젼마다 다를 수도 있고 각 설계 단위에 따라 다를 수도 있다. 버젼을 small-talk의 메카니즘과 같이 설계 단위를 객체로, 버젼을 그 instance로 간주하면, 설계 단위는 서로 다른 속성의 집합을 가질 수 있지만 같은 설계 단위에서 파생된 버젼들은 모두 같은 속성의 집합을 가지고 속성의 내용만 다르게 된다. 사용할 수 있는 모든 속성은 먼저 그 이름과 값의 타입이나 범위 등을 선언한 후 사용을 하고 각 설계 단위마다 있는 default 속성 값이나 속성을 초기화 하는 method를 같이 지정해 줄 수 있다. 그 외 설계 단위는 각 속성마다 그 속성이 유효한 지를 알아보는 method를 함께 가질 수 있다.

V. 실험 결과

제안된 라이브러리 서버는 SUN SPARC workstation의 UNIX C 와 X/MOTIF 환경에서 구현되었다. 실험 결과로 KVUG(Korean VHDL User Group) 92 VHDL conference에서 제안된 8 비트 CISC 마이크로 프로세서와 Verilog사의 8 bit RISC 마이크로 프로세서 설계 예제를 이용하여 본 VHDL 설계 작업 환경에서 CAD 객체들의 관리를 보였다.[13], [14] 라이브러리 서버는 배치 스크립트 형식으로 CAD 툴로부터 입력을 받거나 공유 메모리를 통하여 CAD 툴과 정보교환을 하고 X/MOTIF 인터페이스를 통해서 사용자와 상호작용을 한다.

그림 9는 VHDL을 이용한 8 비트 Verilog RISC 마이크로 프로세서 설계 작업 과정 중 CAD 툴 명령

어 배치 스크립트를 나타낸 것으로 한 줄이 한 CAD 명령에 해당하며 뒤에 필요한 정보가 덧붙여진다. 그림에서 NEW_PROJECT는 VeriRISC라는 새로운 프로젝트의 생성을 나타내고, MK_UNIT는 Veri RISC라는 프로젝트에서 첫 설계 단위의 생성을 나타내며, MK_SUB는 설계 단위에서 사용하는 자식 설계 단위를 생성을 나타내는데 첫 MK_SUB 명령어는 VeriRISC 설계 단위에서 CPU라는 자식 설계 단위를 생성했다는 것을 나타낸다. 새로운 설계 단위가 생성될 경우에는 버전 넘버 0의 default 버전이 같이 생성된다. DERIVE_VER은 버전 파생을 나타내는 명령어이다. 그림 9의 첫 DERIVE_VER 명령어는 CPU 설계 단위의 default 버전에서 새로운 버전이 파생되었다는 것을 나타낸다. 그림 10은 그림 9의 읽어들인 배치 스크립트를 X/MOTIF 환경에서 구현된 라이브러리 서버의 유저 인터페이스 화면으로 그림의 작은 두 화면 중 위 화면은 CT를 나타내고 아래 화면은 버전 파생 트리를 나타낸다. 그리고 오른쪽에는 현재 프로젝트, 설계 단위, 버전 넘버, view의 정보를 입력하고 보여주는 화면들이 있다. CT 화면에서 마우스로 설계 단위를 선택하면 아래 버전 파생 트리 화면에 해당 버전 파생 정보와 default 버전이, 오른쪽 화면에 관련된 정보가 표시된다. 이 그림에서는 Verilog 8 비트 RISC 마이크로 프로세서의 설계 과정에서

```
NEW_PROJECT     VeriRISC
MK_UNIT         VeriRISC[vhdl]
MK_SUB          CPU[vhdl]          VeriRISC:0
DERIVE_VER      CPU:0[vhdl]        -- CPU:1 is vhdl behavioral description.
MK_SUB          MEM[vhdl]          VeriRISC:0[vhdl]
DERIVE_VER      MEM[vhdl]          -- MEM is vhdl behavioral description.
MK_SUB          Accumulator[vhdl]      CPU[vhdl]:0
MK_SUB          ALU[vhdl]              CPU:0[vhdl]
MK_SUB          IR[vhdl]               CPU:0[vhdl]
MK_SUB          Decoder[vhdl]          CPU:0[vhdl]
DERIVE_VER      Decoder[vhdl]
MK_SUB          PC[vhdl]               CPU:0[vhdl]
MK_SUB          Clock[vhdl]            CPU:0[vhdl]
MK_SUB          Mux[vhdl]              CPU:0[vhdl]
MK_SUB          NAND[vhdl]             ALU:0[vhdl]
MK_SUB          AND[vhdl]              ALU[vhdl]:0
MK_SUB          Register8[vhdl]        IR:0[vhdl]
MK_SUB          Register8[vhdl]        PC:0[vhdl]
MK_SUB          INV[vhdl]              Mux:0[vhdl]
MK_SUB          AND[vhdl]              Mux:0[vhdl]
MK_SUB          OR[vhdl]               Mux:0[vhdl]
DERIVE_VER      Register8:0        -- Register8:1 is vhdl behavioral description.
MK_SUB          DFF[vhdl]              Register8:0
DERIVE_VER      DFF[vhdl]
DERIVE_VER      DFF:0[vhdl]
DERIVE_VER      DFF:1[vhdl]
DERIVE_VER      DFF:0[vhdl]
DERIVE_VER      DFF:1[vhdl]
DERIVE_VER      DFF:2[vhdl]
MK_SUB          AND2[vhdl]             Register8:0[vhdl]
MK_SUB          NAND[vhdl]             DFF:0[vhdl]
```

그림 9. Verilog 8 비트 RISC 마이크로 프로세서 설계 작업 명령어의 배치 스크립트

Fig. 9. Batch script describing Verilog 8 bit RISC microprocessor design process.

D-F/F 설계 단위를 참조했을 때 default 버전인 generate 문을 이용해 VHDL 구조 기술로 쓰여진 버전이 결합되는 과정을 나타낸다. 설계 단위의 버전 결합 지정은 버전 메뉴에서 선택할 수 있다. 그림 11은 8 bit RISC 마이크로 프로세서의 전체 설계 과정에서 Register8 설계 단위를 참조했을 때 제너럭으로 지연 시간을 지정할 수 있고 VHDL 행위 기술로 쓰여진 가장 최근에 확정된 버전이 결합되는 과정을 나타낸다.

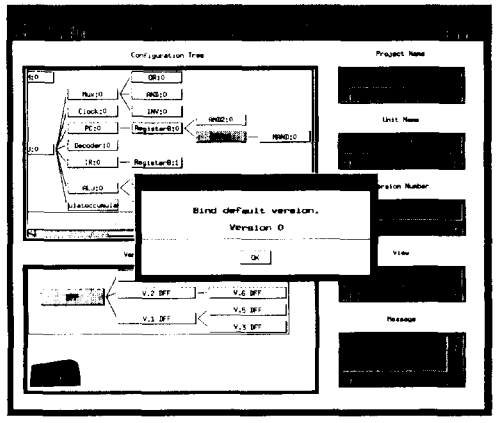

그림 10. 버전 결합시 default 버전을 결합하는 예제

Fig. 10. An example showing default version binding.

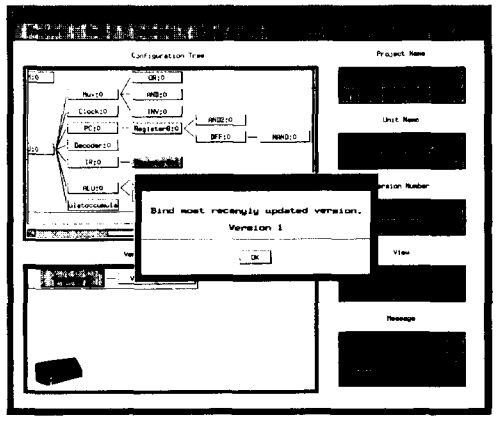

그림 11. 버전 결합시 가장 최근에 확정된 버전을 결합하는 예제

Fig. 11. An example showing the binding of most recently updated version.

그림 12는 KVUG 8 bit CISC 마이크로 프로세서의 설계 과정에서 next_mux block 설계 단위의 VHDL 기술 외 다른 view에 해당하는 설계 단위를 참조하는 과정을 나타낸 그림이다. Next_mux block 설계 단위는 CDFG(Control/Data Flow Graph) 와 AST 등의 view가 있으며 VHDL analyzer가 VHDL을 수정한 후 다른 view를 가진 AST형식의 설계 단위를 출력할 경우나, 한 설계 단위에 수정에 따른 다른 설계 단위의 수정이 전파되는 경우에 유용하게 사용된다.

본 시스템은 working set을 이용하여 physical storage와의 상호 작용을 최소화하였고, 기존의 시스템에서 지원하는 procedural interface 외에 서버와 CAD 툴, method, library들 사이의 공유 메모리를 통한 정보 교환으로 이들의 수정시 매번 서버와의 상호작용 프로시져가 정의된 라이브러리를 함께 컴파일하는 부담을 제거하였다. 그리고 기존의 파일 확장자에 의한 view구분이 아닌 multiple view 관계를 VDG로 표현하여 혼합 레벨 시뮬레이터 객체의 처리와 CAD 객체의 변화에 따른 수정 작업의 전파를 설계자의 개입없이 처리하는 등의 설계 자동화를 효과적으로 지원한다. 그 외 X/MOTIF 상의 인터페이스를 이용하여 설계자가 자동 설계 과정을 확인할 수 있도록 하였다.

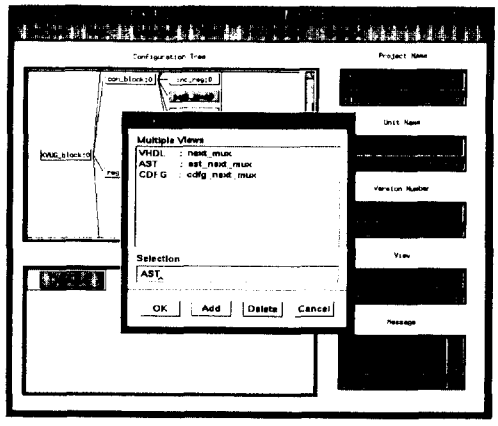

그림 12. 설계단위의 다른 view를 참조하는 예제
Fig. 12. An example of referencing another view of a design unit.

Ⅵ. 결론 및 추후과제

본 논문에서는 CAD 객체를 관리하는 하드웨어 설계 라이브러리의 구현에 대해 기술하였다. 하드웨어 설계 라이브러리를 이용함으로써 설계 객체들간의 복잡한 상호 의존 관계, 버전 관계, 다양한 view의 일괄적인 관리와 이들 사이의 유효성과 일치성 등의 검사가 가능하므로 규모가 큰 시스템의 설계시에 각 부분 시스템의 사양 및 설계 진행 정도의 관리가 용이해진다.

설계 제안된 시스템은 여러 CAD 툴에서 사용하는 중간 형식이나 파일 등을 관리한다. 객체 지향 개념을 적용하여 CAD 객체를 설계 단위로 표현하고 각 설계 단위들 사이의 종속 구조와 버전 관계 및 multiple view 등을 나타낼 수 있도록 하였으며 encapsulation 개념을 적용하여 설계단위가 가질 수 있는 속성과 이에 적용할 수 있는 method를 함께 유지할 수 있도록 하였다. 그리고 한 설계 단위가 가지고 있는 속성은 naming이나 다른 설계 단위에 복사가 가능하도록 함으로써 원시적인 inheritance 개념을 구현하였다. 설계 단위의 참조 관계를 나타내는 종속구조는 라이브러리 내부에서 CT로, 버전사이의 파생 관계는 버전 파생 트리로, view사이의 관계는 VDG로 나타내었다. 라이브러리 서버를 통한 명령들을 이용하여 버전의 제어와 파생 시의 차이를 쉽게 관리할 수 있도록 하였다. 그 외 설계 단위의 수정에 의한 전파의 처리도 지원한다. 라이브러리의 적용의 예로서 VHDL에서 적용한 경우와 이에 관계된 구체적인 속성과 method를 정의하고 구현하였다.

앞으로 각 설계 단위가 동시에 참조된 경우의 동시 제어(concurrency control)에 대한 확장이 필요하고, 다른 CAD 객체에서 적용할 때 속성을 이용해서 설계시의 제약조건을 어떻게 나타내고 처리해야 할 것인가의 문제, 제약 조건을 이용하여 CAD 객체의 수정에 필요한 CAD 툴의 적용 방법을 자동으로 지정하는 문제, 이들의 보다 쉬운 관리를 위한 사용자 인터페이스 문제 등에 대한 지원이 필요하다.

參 考 文 獻

[1] D. Gajski, "Silicon Compilation", Addison-Wesley Pub. :Reading, Mass., 1988.

[2] M. McFarland, A. Parker and R. Camposano, "The High Level syn-thesis of digital systems", *Proceedings of IEEE*, vol. 78, no. 2, Feb. 1990, pp. 301-318.

[3] H. Chou and W. Kim, "A Unifying

Framework for Versions in a CAD Environment", in Proc. Int Conf. on Very Large Data Bases, Kyoto, Japan, August 1986, pp. 336-344.
[4] M. Chung and S. Kim, "The Configuration Management for Version Control in an Object-Oriented VHDL Design Environment", IEEE International Conference on Computer Aided Design, Novemver 1991, pp. 258-261.
[5] R. Katz, M. Anwarrudin, E. Chang, "A Version Server For Computer-Adied Design Data", in Proc. of 23rd Design Automation Conference, June 1986, pp. 644-650.
[6] R. Dutta, J. Roy, R. Vemuri, "Distributed Design-Space Exploration for High-Level Synthesis Systems", in Proc. of 29th Design Automation Conference, June 1992, pp. 644-650.
[7] IEEE Standard VHDL Language Reference Manual, IEEE 1076-1987, April 1989.
[8] M. Chung and S. Kim, "An Object-Oriented VHDL Environment", in Proc. of 27th Design Automation Conference, June 1990, pp. 431-436.
[9] 이 영희, 황 선영, "VHDL 설계 환경 구축", 한국 정보 과학회 논문지, 제 17 권, 제 1 호, 1990년 4월, pp. 403-406.
[10] 이 영희, 김 헌철, 황 선영, "다층 레벨 VHDL 시뮬레이터의 설계", 대한 전자 공학회 논문지, 제 30-A권, 제 10호, 1993년 10월, pp. 67~76.
[11] 전홍신, 이해동, 황선영, "서강 실리콘 컴파일러의 High Level Synthesis 시 스템 설계", 대한 전자 공학회 논문지, 28-A권, 제 6호, 1991년 6월, pp. 488-499.
[12] J. Hughes, "Object-Oriented Databases", Prentice Hall, 1991.
[13] 조중휘, "VHDL 응용 (8 bit CPU Modeling)", in Proc. of 1st Korean VHDL User 's Group Workshop, 1992년 10월.
[14] Cadence, Technical Memo for VHDL User Training, October 1992.

著者紹介

崔 益 成(準會員)
1992年 2月 서강대학교 전자공학과 졸업. 1992年 2月 ~ 현재 동 대학원 재학중. 주관심분야는 CAD 시스템, Computer Architecture 및 VLSI Testability 등임.

李 榮 熙(準會員) 第 30卷 A編 第 10號 參照
현재 서강대학교 전자공학과 박사 과정

黃 善 泳(正會員)
1976年 2月 서울대학교 전자공학과 졸업. 1978年 2月 한국 과학원 전기 및 전자공학과 공학석사 취득. 1986年 10月 미국 Stanford 대학 공학박사학위 취득. 삼성 반도체 주식회사 연구원, Stanford 대학 CIS 연구소 연구원, Fairchild Semiconductor 기술 자문. 1989年 3月 ~ 현재 서강대학교 전자공학과 교수. 주관심 분야는 CAD 시스템, Computer Architecture 및 Systems Design, VLSI 설계 등임