論文95-32B-7-3

VHDL 설계 데이터베이스 구현 방법의 비교 연구

(A Comparative Study on Methods for Implementing VHDL Design Database)

崔 勝 旭 *, 崔 起 榮 **

(SeungWook Choi, and Kiyoung Choi)

요 약

본 논문에서는 VHDL 툴 개발 시스템의 사례 연구를 통하여 VHDL 설계 데이터베이스를 구현하는 몇 가지 방법에 대한 비교를 한다. VHDL 툴 개발 시스템의 중심이 되는 VHDL 설계 데이터베이스를 C 프로그래밍 언어를 이용한 값 지향 방법, C++ 프로그래밍 언어를 이용한 객체 지향 방법, 기존의 객체 지향 데이터베이스를 이용한 방법의 세 가지 방법으로 각각 구현하고 그 경험을 토대로 각 구현 방법의 장단점을 정량적으로 제시한다. 대체로 값 지향 방법은 구현은 가장 어려우나 성능은 좋은 것으로 나타났으며 C++ 언어를 이용한 객체 지향 방법은 값 지향 방법과 비교하여 구현이 두 배 정도 쉬우며 성능은 비슷한 것으로 나타났다. 기존의 객체 지향 데이터베이스를 이용하는 방법은 가장 구현하기 쉬우나 성능은 1.5 배 내지 6 배 정도 나쁜 것으로 나타났다.

Abstract

In this paper, we compare several methods for implementing a VHDL design database through a case study on VHDL tool development system. We implemented three versions of the VHDL design database which the VHDL tool development system is based on. The first version was coded in the C programming language following value-oriented paradigm. The second one was coded in the C++ programming language following object-oriented paradigm. The third one was implemented using an existing object-oriented database. Based on our experience, we present quantitatively the pros and cons of each implementation method. The value-oriented version was most difficult to implement but showed good performance. Compared to the value-oriented version, the C++ version was twice as easy to implement and showed about the same performance. Using an existing object-oriented database allowed easiest implementation but resulted in a 1.5 to 6 times slower version.

* 正會員, (주)메디슨 (Medison Co., Ltd.)

** 正會員, 서울大學校 電子工學科

(Dept. of Elec. Eng., Seoul National Univ.)

※ 이 연구는 1992년도 교육부 학술연구조성비와 한국전자통신연구소의 지원으로 수행되었음.

接受日字: 1994年7月5日, 수정완료일:1995年7月12日

I. 서 론

일반적으로 객체 지향 기술(object-oriented technology)은 기존의 값 지향 기술(value-oriented technology)에 비하여 여러 가지 장점을 제공하기 때문에[1] 많은 사람들이 이용하고 연구해 왔다[2, 3, 4]. 그러나 객체 지향 기술과 기존의 기술을 사례 연구를

통하여 설득력 있는 비교를 한 연구보고 -- 구현 방법에 따라 얼마나 구현하기 쉽고, 성능은 어떻게 달라지는지 정량적으로 비교한 연구보고 -- 는 아직까지는 그리 많지 않은 형편이다. 그 주된 이유는 같은 기능의 큰 시스템을 여러 가지 방법으로 구현해 보아야 한다는 어려움 때문이다.

기존의 값 지향 기술과 객체 지향 기술의 비교를 다룬 연구로는 [5]가 있다. 여기서는 사용자 인터페이스를 생성해 주는 툴킷(toolkit)을 절차적(procedural) 프로그래밍 언어인 Modular-2와 객체 지향 프로그래밍 언어인 C++로 각각 구현하고 두 방식을 비교, 평가하였다. Modular-2로 만든 것을 UICT(User Interface Construction Toolkit)라 부르고 C++로 만든 것을 DICE(Dynamic Interface Construction Environment)라 부르는데 UICT와 DICE 자체의 크기를 비교하면 DICE의 경우 객체 지향 언어의 특징인 재사용성(reusability)을 충분히 활용했기 때문에 코드 크기가 5 분의 1 미만으로 훨씬 작은 것을 볼 수 있다. 특히 DICE는 ET++라는 클래스 라이브러리를 활용했기 때문에 코드 크기를 더욱 절약할 수 있었던 것으로 보인다. 이 결과는 객체 지향 기술을 사용하는 것이 보다 효율적이라는 주장을 뒷받침하기에 충분하기는 하나 서로 대등한 관계에 있는 프로그래밍 언어가 아닌 Modular-2와 C++를 비교한 것이므로 과장된 감이 없지 않다.

본 논문에서는 VHDL 설계 데이터베이스를 구현하는 방법으로서 기존의 값 지향 기술을 이용한 방법과 객체 지향 기술을 이용한 방법의 비교뿐만 아니라 객체 지향 기술을 이용한 방법 중에서도 C++ 프로그래밍 언어로 처음부터 새로 만드는 방법과 기존의 범용 객체 지향 데이터베이스를 이용하는 방법의 두 가지 방법을 포함한 세 가지 방법을 서로 비교한다. 여기에서는 각 방법을 이용하여 VHDL 설계 데이터베이스와 일부 툴들을 실제로 구현하여 각 방법의 장단점을 비교, 평가하기 위한 자료를 구한다. VHDL 설계 데이터베이스는 표준 하드웨어 기술 언어인 VHDL (VHSIC Hardware Description Language)[6]로 표현된 설계들을 관리하는 데이터베이스로 VHDL 툴 개발 시스템의 주축을 이룬다[7].

다음 장에서는 우리가 구축한 VHDL 툴 개발 시스템과 이의 주축인 VHDL 설계 데이터베이스에 대하여 설명한다. Ⅲ장에서는 값 지향 기술을 이용한 기존의 구현 방식에 대하여 객체 지향 기술을 이용한 방식의 차이점과 그 구현 내용에 대하여 설명한다. Ⅳ장에서는 각 구현 방식을 서로 비교, 평가한다. 마지막으로 결론과 함께 추후 과제에 대하여 언급한다.

Ⅱ. VHDL 툴 개발 시스템

VHDL 툴 개발 시스템은 VHDL 관련 툴을 개발하기 쉽도록 기본적으로 필요한 VHDL 설계 데이타베이스, VHDL 분석기(analyzer), VHDL 생성기(generator), 기타 툴들을 툴킷의 형태로 갖추고 있다. VHDL 설계 데이타베이스는 VHDL 분석기(analyzer)가 생성해 내는 중간 형태(intermediate format)의 자료들을 관리하며 procedural interface를 제공하여 각 툴들이 쉽고 안전하게 중간 형태의 자료를 다룰 수 있도록 해 준다[7]. 그림 1은 우리가 개발한 VHDL 툴 개발 시스템이 어떻게 이루어져 있는가를 보여 준다.

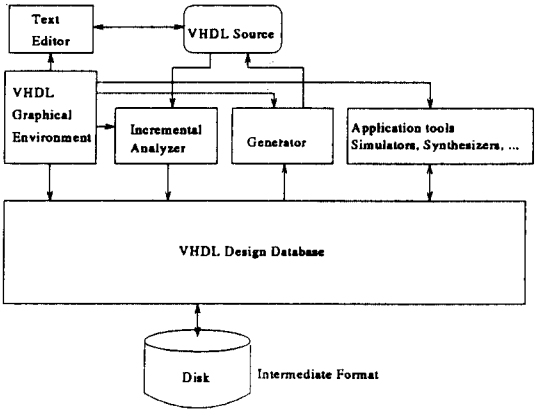

그림 1. VHDL툴 개발 시스템
Fig. 1. VHDL tool development system.

VHDL 설계 데이타베이스를 구현하기 위하여 다음과 같은 몇가지 방법을 생각할 수 있다.

1. 모든 것을 처음부터 새로 구입한다. 이 경우 어떠한 방법으로 구현하는가가 문제가 된다. 기존에 많이 사용해 오던 방식으로 절차적 (procedural) 프로그래밍 언어를 사용하여 구현하는 것이 한 가지 방법이고 최근 들어 각광받고 있는 객체 지향 프로그래밍 언어를 사용하여 구현하는 것이 또 다른 방법이다.
2. 기존의 범용 데이터베이스를 이용하여 구현한다. 대

부분의 필요한 기능은 이미 갖추어져 있으므로 최소한의 노력으로 원하는 것을 얻을 수 있다. 이 경우에는 관계형(relational) 데이터베이스나 역시 최근에 활발한 연구 대상이 되고 있는 객체 지향 데이터베이스[8, 9] 등 여러 종류의 데이터베이스 중 어떠한 것을 사용하는가가 문제이다.

첫째 방법에서 절차적 프로그래밍 언어를 사용하는 것은 기존의 VHDL 설계 데이터베이스를 구현할 때에 사용한 방법이다[7]. 둘째 방법에서 관계형 데이터베이스를 사용하는 것은 관계형 데이터베이스의 제한된 자료 표현 능력으로 인해 구현이 매우 어렵고 비효율적인 면이 있다[10]. 본 논문에서는 위에서 제시한 방법 중 객체 지향 프로그래밍 언어를 이용한 방법과 범용 객체 지향 데이터베이스를 이용한 방법으로 VHDL 설계 데이터베이스를 구현할 때의 장단점을 기존의 첫째 방법으로 구현할 때와 비교, 평가해 보기로 한다.

Ⅲ. 객체 지향 기술을 이용한 구현

객체 지향 기술은 각 응용 분야마다 조금씩 다르기는 하지만 그 주요 특징을 살펴보면 대개 자료 추상화(data abstraction), 상속성(inheritance), 다형성(polymorphism) 등을 들 수 있다. 이러한 특징들을 지원하는 프로그래밍 언어나 데이터베이스는 다른 프로그래밍 언어 또는 관계형 데이터베이스가 표현할 수 없는 복잡한 자료형(data type)을 쉽게 표현할 수 있게 해 준다. 이러한 특징을 살리기 위하여 기존의 VHDL 설계 데이터베이스의 구조를 일부 변경하였다.

1. 새로운 구조의 VHDL 설계 데이터베이스

기존의 VHDL 설계 데이터베이스는 자료 자체와 그 자료를 처리하는 방법(method)이 서로 분리되어 있는 값 지향 시스템으로 자료 추상화 또는 은폐를 기대하기가 어려웠다. 또한 자료 구조를 상속시킬 수 없기 때문에 같은 속성이 반복해서 나오더라도 매번 선언해 주어야 하는 불편함이 있었다. 그런데 객체 지향 기술에서는 자료와 그것들을 다루는 방법이 서로 분리되어 있지 않고 하나의 클래스를 이루기 때문에 기존의 자료 구조에서 해결하지 못한 정보 은폐와 모듈 화를 효과적으로 이룰 수 있다. 또한 상속을 통해 자료 구조를 재 사용할 수 있으므로 코드를 절감할 수 있다.

새로운 구조의 VHDL 설계 데이터베이스는 그림 2에 보인 바와 같이 인터페이스 모듈(Interface Module), 객체 관리 모듈(Object Management Module), 중간 형태의 자료 관리 모듈(Intermediate Format Management Module), 라이브러리 관리 모듈(Library Management Module), 오류 처리 모듈(Error Handling Module), 어의 검사 모듈(Semantics Checking Module)의 여섯 개의 모듈로 구성되어 있다. 각 모듈의 기능을 간단히 설명하면 다음과 같다(자세한 설명은 [11, 12] 참고).

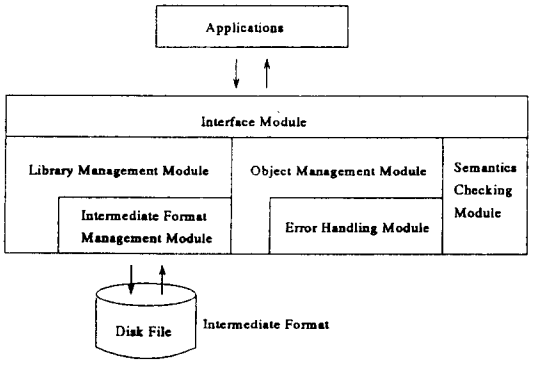

그림 2. 새로운 VHDL 설계 데이타베이스의 구조
Fig. 2. Structure of the new VHDL Design Database.

* 인터페이스 모듈은 응용 프로그램과 객체간의 인터페이스를 위해 제공되는 ID에 대한 정의를 가지고 있는 부분이다. 객체를 참조할 때 사용하는 ID는 단순한 포인터가 아니기 때문에 응용 프로그램 또는 사용자가 잘못하여 자료를 훼손하는 것을 방지해 준다.
* 객체 관리 모듈은 VHDL 언어를 표현하기 위해 필요한 객체를 관리하기 위한 모듈로서 VHDL 설계 데이터베이스에서 가장 큰 부분이다. 객체의 관리는 함수를 통하여 이루어지는데 이 함수들은 다음과 같은 다섯 가지로 분류된다.
  - 객체를 생성하는 함수
  - 중간 형태에 객체를 추가하는 함수
  - 객체의 속성(attribute) 값 또는 다른 객체와의 관계를 설정하는 함수
  - 객체의 속성 값 또는 다른 객체와의 관계를 읽는 함수
  - 심볼표(symbol table) 검색 등을 위한 기타 함수

* 중간 형태 자료 관리 모듈은 VHDL 구문을 분석한 결과를 중간 형태로 디스크에 저장하거나 읽어 들이는 것을 관리하는 모듈로서 C++ 언어로 구현한 것에만 필요하다. 기존의 객체 지향 데이터베이스를 이용한 구현에서는 자체적으로 디스크/메모리 관리 모듈을 갖고 있기 때문에 이 모듈이 필요 없다.
* 라이브러리 관리 모듈은 VHDL 설계 상의 라이브러리를 관리하기 위한 것으로 VHDL 라이브러리의 생성, 저장, 열람 등에 필요한 모든 함수들이 정의되어 있다. 응용 프로그램 개발자는 이 라이브러리 관리 모듈을 이용하여 매우 쉽게 라이브러리를 관리할 수 있다.
* 오류 처리 모듈은 자료의 입출력이 행해지는 동안 발생하는 오류를 처리하기 위한 모듈이다.
* 어의 검사 모듈은 중간 형태의 자료가 VHDL의 어의 규칙을 만족하는가를 검사한다. 이는 항상 VHDL의 어의 규칙을 만족하는 설계만을 데이터베이스에 저장하는 데 필수적이다.

```
class BaseObject;
typedef     BaseObject*  BasePointer;
class Library;
typedef     Library*     LibraryPointer;

template<class T>
class BaseID {
    T*              _object;
public:
    BaseID( void );
    BaseID( const BaseID<T>& );
    BaseID( T* );
    BaseID<T>&  operator=(const BaseID<T>&);
    BaseID<T>&  operator=( T* );
    T*          operator->( void ) const;
    boolean     isNull( void ) const;
    boolean     isValid( void ) const;
    void        nullify( void );
    T*          vm( void ) const;
friend  int     operator==( const BaseID<T>&,
                            const BaseID<T>& );
friend  int     operator!=( const BaseID<T>&,
                            const BaseID<T>& );
    };

typedef   BaseID<BaseObject>  ID;
typedef   BaseID<Library>     LibID;
typedef   SLList<ID>          ID_Itr;
typedef   SLList<LibID>       LibID_Itr;
```

그림 3. C++ 프로그래밍 언어를 이용한 인터페이스 모듈의 구현

Fig. 3. Interface module implemented in the C++ programming language.

2. C++ 프로그래밍 언어를 이용한 구현

C++ 프로그래밍 언어를 이용한 구현에서는 앞서 설명한 여섯 가지 모듈을 모두 새로 설계해야 한다. 그림 3은 일례로 인터페이스 모듈을 구성하는 ID 클래스의 정의를 보여 준다.

위의 클래스 정의에서 볼 수 있듯이 ID 클래스의 사유 멤버(private member)로는 실제 객체를 참조하기 위한 포인터가 있고 이 포인터 값을 설정 또는 열람하기 위한 여러 가지 멤버 함수가 정의되어 있다. 특히 ID가 참조하는 객체가 유효한지의 여부를 검증하기 위한 멤버 함수로 isNull()과 isValid()가 있다. 또한 두개의 ID가 서로 동일한 객체를 나타내는지를 알아보기 위한 것으로 ``==''와 ``!=''의 두 가지 연산자를 재 정의하였다. 여러 개의 ID를 다루기 위해 ID_Itr라는 반복자(iterator) 클래스가 정의되어 있는데 이는 ID를 리스트 구조로 표현한 것이다.

3. 범용 객체 지향 데이터베이스를 이용한 구현

본 논문에서는 범용 객체 지향 데이터베이스로 Objectivity/DB를 사용하였다[13]. 이를 이용하여 응용 프로그램을 개발할 때에는 먼저 스키마(schema)를 정의하여야 한다. Objectivity/DB에서 스키마는 DDL(Data Definition Language)을 이용하여 정의할 수 있는데 이 DDL 스키마의 원시 파일(source file)을 전용 컴파일러(DDL Processor)로 컴파일 하면 C++ 스키마 원시 코드가 생성된다. 이렇게 하여 생성된 스키마 원시 코드와 응용 프로그램의 원시 코드를 함께 컴파일/링크 함으로써 응용 프로그램의 실행 파일을 만들 수 있다.

Objectivity/DB에는 시스템이 제공하는 다섯 가지의 지속성 클래스(system-defined persistent class)가 있으며 사용자가 이들로부터 파생시켜 정의한 모든 클래스는 지속성을 갖는다(user-defined persistent class). Objectivity/DB의 저장 관리 체계(storage management system)는 다음과 같은 네 단계로 계층화되어 있다.

1. 가장 하위의 저장 단위(storage unit)는 기본 객체(basic object)라는 것으로 네 계층 중 가장 기본이 되고 실제로 자료를 저장하는 단위가 된다.
2. 기본 객체들을 모아서 컨테이너(container)라는 저장 단위에 넣을 수 있는데 컨테이너는 기본 객체들을 디스크에서 물리적으로 가까운 위치에 모아

놓고 기본 객체들을 관리하는 기능을 갖추어 효율적인 자료 관리를 할 수 있도록 한다.

3. 컨테이너들이 모여서 비로소 하나의 데이터베이스가 되는데 실제로 데이터베이스는 파일 시스템에서 하나의 파일을 이룬다. 데이터베이스도 하나의 객체로 표현된다.
4. 같은 스키마를 갖는 데이터베이스들이 집합을 이루어 연합 데이터베이스(federated database)를 구성한다. 연합 데이터베이스는 데이터베이스 객체들과 스키마 정의를 포함한다.

Objectivity/DB를 VHDL 툴 개발 시스템에 응용하기 위해 Objectivity/DB의 저장 단위를 다음과 같이 할당하였다.

* 데이터베이스 --> VHDL 설계
* 컨테이너 --> VHDL 라이브러리
* 기본 객체 --> 라이브러리 단위, 일반 객체

컨테이너를 VHDL의 라이브러리에 할당한 이유는 VHDL에서 라이브러리내의 라이브러리 단위(library unit)들이 하나의 저장 단위가 되는데 서로 물리적으로 가까운 위치에 있어야 효율적인 입출력을 할 수 있기 때문이다.

```
#define   ID        ooHandle(BaseObject)
#define   LibID     ooHandle(Library)
#define   ID_Itr    ooItr(BaseObject)
#define   LibID_Itr ooItr(Library)
```

그림 4. Objectivity/DB를 이용한 인터페이스 모듈의 구현

Fig. 4. Interface module implemented using Objectivity/DB.

Objectivity/DB를 이용한 구현의 일례로 인터페이스 모듈을 보면 그림 4와 같이 매우 간단하다. 여기에서는 Objectivity/DB가 객체를 참조하기 위해 제공하는 핸들(handle)을 그대로 ID에 적용하였으며 ID_Itr는 Objectivity/DB가 제공하는 반복자를 그대로 사용하였다. Objectivity/DB의 핸들은 지속성 객체와 응용 프로그램간의 인터페이스를 위해 제공되는 비지속성 객체(non-persistent object)이다.

Ⅳ. 구현 방법의 비교 평가

여기에서는 이상에서 제시한 두 가지 객체 지향적 구현 방법과 기존의 값 지향적 구현 방법을 코드의 크기, 수행 시간, 기능적 측면에서 비교, 분석한다.

1. 코드 크기 비교

일반적으로 코드의 크기는 소프트웨어 시스템 구현의 난이도에 대한 중요한 척도가 된다. 물론 각각 다른 언어로 구현되어 있는 경우 코드 크기로 난이도를 비교하는 것이 무리가 될 수도 있다. Modular-2와 C++ 구현을 비교한 기존의 연구[5]는 이러한 점에서 어느 정도 오류를 범하고 있다고 여겨진다. 그러나 본 논문에서와 같이 C와 C++, 그리고 C++에 기초한 DDL로 구현한 경우 각 언어에서 사용하는 구문들이 대부분 대등한 관계에 있으므로 코드의 크기는 매우 설득력 있는 구현 난이도의 척도가 된다. 본 논문에서는 모든 코드에서 주석(comment)을 제거하고 각 줄에 하나의 구문만 오도록 처리를 한 후에 줄 수를 셈으로써 코드의 크기를 측정, 비교할 때의 문제점을 최소화하였다.

표 1. 구현 방법에 따른 코드의 크기(줄 수) 비교

Table 1. Code size(number of lines) comparison between implementation methods.

	C	C++	Objectivity/DB
VHDL 설계 데이터베이스	20K	8K	6K
분석기	6K	4K	4K
전체 줄 수	26K	12K	10K

VHDL 툴 개발 시스템을 구성하는 주요 프로그램들의 코드 크기를 비교해 보면 표 1과 같다. VHDL 설계 데이터베이스의 경우 C를 사용한 것과 비교하여 C++를 사용하면 약 40 %, Objectivity/DB를 사용한 경우 약 30 % 정도로 코드가 절감된 것을 볼 수 있다. 이는 앞서 언급한 상속성과 소프트웨어 재사용성에 기인한 결과라고 말할 수 있다. 각 객체를 표현하기 위한 자료 구조에서 서로 비슷한 기능을 하는 객체라 하더라도 C 구현에서는 모두 새로 써 주어야 하는

반면 C++ 또는 Objectivity/DB와 같은 객체 지향 기술을 이용한 구현에서는 간단한 객체에 추가적인 특성을 덧붙여 나감으로써 새로운 복잡한 객체를 표현할 수 있기 때문에 코드가 절감되고 구현이 쉬워진다. 또한 C++ 언어와 Objectivity/DB를 이용한 구현에서는 문자열(string)이나 리스트(list), 해쉬표(hash table)등을 모두 새로 개발할 필요 없이 라이브러리에 포함되어 있는 클래스들을 재 사용하면 되기 때문에 코드가 더욱 절감된다.

분석기의 경우에는 그다지 코드가 절감되지 않은 것을 볼 수 있는데 이는 구현 방법에 따라 차이가 없는 VHDL의 어휘 분석기(lexical analyzer)와 파서(parser)가 많은 부분을 차지하기 때문이다. 다만 새로운 구현에서는 VHDL의 라이브러리 관리 모듈이 VHDL 설계 데이터베이스에 내장되어 있기 때문에 그것에 상당하는 코드가 절감된 것일 뿐이다. 만일 C 구현에서도 라이브러리 관리 모듈을 VHDL 설계 데이터베이스에 내장시켰다면 그 크기는 20K에서 22K 이상으로 증가했을 것이다.

C++ 구현과 Objectivity/DB 구현을 비교해 보면 후자의 경우 약 2K 줄 정도 크기가 작은 것을 볼 수 있는데 이는 Objectivity/DB에서는 중간 형태 관리 모듈이 필요하지 않기 때문이다.

2. 수행 시간의 비교

여기서는 세 가지의 VHDL 예제 프로그램을 각각의 프로그램으로 분석할 때에 걸린 시간의 10회 평균을 비교해 보았다. 표 2는 비교 결과를 보여 준다.

표 2. 구현 방법에 따른 분석 시간 비교

Table 2. Analysis time comparison between implementation methods.

예제	크기 (줄수)	수행시간 (초)		
		C	C++	Objectivity/DB
standard.vhd	45	1.3	2.0	7.7
tlc.vhd	63	2.6	2.1	4.9
alu.vhd	73	3.6	2.5	5.4

이 결과를 고찰해보면 C 구현과 C++ 구현에서는 예제의 크기가 커질수록 분석 시간이 많이 걸리는데 반해 Objectivity/DB 구현에서는 오히려 크기가 가장 작은 standard.vhd를 분석하는 데에 가장 시간이 많이 걸린 것을 볼 수 있다. 또한 예제의 크기가 커질 때 C++ 구현이 C 구현보다 분석 시간의 증가율이 작은 것도 알 수 있다. 이상의 두 가지 결과를 분석해 보면 다음과 같다.

1. Objectivity/DB 구현에서 예제의 크기와 분석 시간이 비례하지 않는 것은 예제의 성질을 살펴봄으로써 이해할 수 있다. standard.vhd는 새로운 자료를 추가하는 것이 대부분이고 다른 예제는 이미 분석된 자료를 열람하는 것과 새로운 자료를 추가하는 것이 혼합된 것인데 Objectivity/DB는 자료를 생성 또는 추가할 때에는 느리고 자료를 열람할 때에는 빠르게 동작한다. Objectivity/DB는 차후의 신속한 열람을 위해 새로운 자료를 추가할 때에 인덱스를 만들어 놓는 등의 부가적인 처리를 하며 이로 인해 주로 자료를 열람하는 예제에서는 성능이 훨씬 좋아지게 된다.
2. C 구현과 C++ 구현에 있어서 분석 시간의 증가율이 다른 이유는 중간 형태의 자료를 저장, 열람하는 방식이 서로 다르기 때문이다. C 구현에서는 VHDL의 라이브러리 단위(library unit)를 필요할 때에만 기억 장치 공간으로 읽어 오는 데 반해 C++ 구현에서는 분석을 시작할 때에 library clause에 선언된 모든 라이브러리를 기억 장치 공간으로 읽어 온다. C++ 구현의 경우에 필요 없는 라이브러리 단위도 함께 기억 장치로 올라오기 때문에 기억 장치가 낭비되는 일이 있을 수 있으나 디스크 입출력을 한 번만 하므로 성능은 좋아진다. 그러나 이러한 차이는 라이브러리 관리 모듈을 어떻게 설계하는가에 따른 차이이지 C 언어와 C++ 언어의 차이는 아니며 C 언어 구현과 C++ 언어 구현은 성능의 차이가 크지 않음을 알 수 있다.

이상을 요약해 보면 Objectivity/DB 구현은 C 언어 또는 C++ 언어 구현에 비해 1.5 - 6 배 정도 성능이 떨어지는 것으로 결론지을 수 있는데 주된 이유는 우리가 개발한 시스템에서 필요로 하지 않는 많은 기능을 포함하고 있는 범용 데이터베이스의 비효율성 때문으로 생각된다.

3. 기능적 측면의 정성적 분석

객체 지향 기술을 이용한 구현에서는 VHDL 구문

에 대응되는 객체뿐만 아니라 라이브러리 단위나 전체 VHDL 설계 자체도 하나의 객체로 취급하므로 저장되는 자료의 일관성과 안전성이 보장된다. 예를 들어서 VHDL 설계 자체가 하나의 객체이므로 하나의 프로그램에서 여러 개의 VHDL 설계 객체를 생성시키기만 하면 여러 개의 설계를 동시에 관리할 수 있다. 마찬가지로 하나의 설계안에서도 여러 개의 라이브러리 단위를 동시에 읽고 쓰는 것이 가능하다. 값 지향 기술을 이용해도 이러한 것들이 불가능한 것은 아니지만 객체 지향 기술을 이용하는 것보다 훨씬 효율성과 안전성이 떨어진다.

또한 함수를 호출할 때 호출하는 객체를 인수(parameter)로 전달하는 것이 아니라 호출하는 객체에 있는 멤버 함수를 부르게 되므로 자동으로 철저한 형 검사(type checking)가 이루어진다. 따라서 그만큼 코드가 절약되고 오류가 줄게 된다.

데이터베이스를 사용함으로써 얻어지는 이점으로는 무엇보다도 개발 효율을 들 수 있다. 일반적인 프로그래밍 언어를 이용하는 것에 비해 자료 관리를 데이터베이스 관리 시스템이 대신해 주므로 보다 빠르고 안전하게 원하는 시스템을 만들 수 있다. 프로그래밍 언어를 사용할 때에 일일이 고려해 주어야 하는 기억장치 관리, 디스크 입출력 관리, 동시성 제어(concurrency control), 버전 관리(version management), 다중 사용자 관리(multiuser control) 등의 복잡한 문제를 데이터베이스 관리 시스템이 모두 해결해 줄 수 있으므로 개발자는 자료 그 자체에 보다 더 중점을 둘 수 있다.

특히 객체 지향 데이터베이스는 풍부한 자료 표현 능력으로 인해 복합 객체(complex object)를 쉽게 표현할 수 있으며 따라서 설계 데이터베이스와 같이 복잡한 설계 정보를 다루어야 하는 데에 그 이용 가치가 높다고 생각한다.

V. 결론 및 추후 과제

본 논문에서는 실제의 구현을 통하여 VHDL 설계 데이터베이스의 세 가지 구현 방법을 정량적으로 비교해 보고 또 기능적 측면의 정성적 분석을 해 보았다. 여기에서 보인 결과는 VHDL 설계 데이터베이스뿐만 아니라 회로도나 레이아웃(layout) 설계를 포함하는 일반적인 설계 데이터베이스에 대해서도 크게 차이가 없이 적용될 수 있다고 생각한다.

본 논문에서의 결론은 어느 정도 예견할 수 있는 것이나, 실제의 구현을 통하여 얼마나 구현이 쉬우며 또 성능은 얼마나 차이가 나는지 정량적으로 분석한 것에서 그 의미를 찾을 수 있다.

VHDL 설계 데이터베이스를 C 프로그래밍 언어로 구현한 경우와 비교하여 객체 지향 기술을 이용하여 C++ 프로그래밍 언어로 구현하면 같은 기능을 갖는 프로그램의 코드 크기는 절반 이하로 줄어들지만 수행 시간은 크게 차이가 없는 것을 보였다. 범용 객체 지향 데이터베이스를 이용하는 것은 설계 데이터베이스의 특성을 고려하면 그 적용이 매우 편리하지만 아직까지는 성능 면에서 일반 프로그래밍 언어로 구현하는 것보다는 못하다는 것을 보였다. 그러나 이번 구현에 사용한 Objectivity/DB는 특별히 CAD 응용을 위해 제작된 것이 아닌 범용 객체 지향 데이터베이스이므로 객체 지향 데이터베이스를 이용하면 성능이 좋지 않다고 말할 수는 없을 것이다. 앞으로 원하는 시스템을 쉽게 그리고 성능도 떨어지지 않게 구현할 수 있는 효율적인 객체 지향 데이터베이스를 개발할 필요가 있다고 생각한다.

참 고 문 헌

[1] A. L. Winblad, S. D. Edwards, and D. R. King, "Object-Oriented Software" Reading, Massachusetts: Addison-Wesley, 1990.

[2] A. R. Hurson, S. H. Pakzad, and J. bing Cheng, "Object-oriented database management systems: Evolution and performance issues," Computer, vol. 26, pp. 48-60, Feb. 1993.

[3] M. Marefat, S. Malhotra, and R. L. Kashyap, "Object-oriented intelligent computer-integrated design, process planning, and inspection," Computer, vol. 26, pp. 54-65, Mar. 1993.

[4] W. Wolf, "Object-oriented programming for CAD," IEEE Design and Test of Computers, vol. 8, pp. 35-42, Mar. 1991.

[5] W. Pree and G. Pomberger, "Object-

oriented versus conventional software development: A comapative case study," Microprocessing and microprogramming, vol. 35, pp. 203-211, 1992.

[6] The Institute of Electrical and Electronics Engineers, Inc., New York, New York, "IEEE Standard VHDL Language Reference Manual," IEEE Std 1076-1987, 1988.

[7] D. H. Ko and K. Choi, "IVDT: A VHDL developer's toolkit," *KITE Journal of Electronics Engineering*, vol. 5, pp. 56-63, Dec. 1994.

[8] W. Kim, "Object-oriented databases: Definitions and research directions," IEEE Trans. Knowledge and Data Engineering, vol. 2, Sept. 1990.

[9] J. V. Joseph, S. M. Thatte, C. W. Thomson, and D. L. Wells, "Object-oriented databases: Design and implementation," Proceedings of the IEEE, vol. 79, Jan. 1991.

[10] W. Kent, "Limitations of record-based information models," ACM Trans. Database Systems, vol. 4, pp. 107-131, Mar. 1979.

[11] 최승욱, "객체 지향 기술을 이용한 VHDL 툴 개발 시스템의 구현에 관한 연구," 석사 학위 논문, 서울대학교, 1994

[12] 최기영, "ISRC VHDL Tool 개발 환경 구축, 2차년도 연구 보고서," ISRC93-E-0015, 서울대학교 반도체공동연구소, 1993

[13] Objectivity Inc., "Objectivity/DB Documentation Volume II," 1992.

저 자 소 개

崔 勝 旭(正會員)

1970년 4월 16일생. 1992년 서울대학교 전자공학과 학사. 1994년 서울대학교 대학원 전자공학과 석사. 1994년 - 현재 (주)메디슨 연구소 근무. 주 관심 분야는 데이터베이스, 네트워킹 등임.

崔 起 榮 (正會員)

1955년 8월 30일생. 1978년 서울대학교 전자공학과 학사. 1980년 한국과학원 전기및전자공학과 석사. 1989년 미국 Stanford 대학 전기공학과 박사. 1978년 - 1983년 (주)금성사 중앙연구소 근무. 1989년 - 1991년 미국 Cadence Design Systems, Inc. 근무. 1991년 - 현재 서울대학교 전자공학과 및 반도체공동연구소 조교수. 주 관심 분야는 CAD, VLSI 설계 등임.