하드웨어 시뮬레이션을 위한 전이중심 객체지향 프로그래밍 시스템(TOPS)

음 두 헌†

요 약

전이중심 객체지향 프로그래밍 시스템(Transition-based Object-oriented Programming System, TOPS)은 다양한 병행 시스템의 신속한 개발에 적합한 상태 전이에 입각한 객체지향 시스템이다. TOPS는 상호 작용하는 구조적 능동 객체 (Structural Active Object, SAO)들로 구성되며 그들의 능동적인 동작은 클래스 정의에서 전이 문장에 의해 정의된다. 또한, SAO들은 하드웨어 소자들과 같이 그들의 구성 SAO들을 구조적이고 계층적으로 구성함으로써 정의될 수 있다. 이러한 SAO들은 회로 시뮬레이션을 위한 능동 하드웨어 소자들을 기존의 객체지향 프로그래밍의 수동 객체들에 비해 자연스럽게 모델링할 수 있다. 또한, 상속 기능을 통해 새로운 소자들을 쉽게 만들 수도 있다. 전이 문장들의 수행은 사건 또는 시간 중심 방식을 취할 수 있기 때문에 디지탈, 아날로그 및 혼합모드 시뮬레이션에 적합하다. 그래픽 사용자 인터페이스를 지원하는 디지탈, 아날로그 그리고 혼합모드 시뮬레이션 프로그램들을 TOPS 방식으로 작성하여 그 적합성을 보였다.

Transition-based Object-oriented Programming Systems (TOPS) for Hardware Simulation

Doohun Eum†

ABSTRACT

A *transition-based object-oriented programming system* (TOPS) is a *transition-based object-oriented* system suitable for development of various concurrent systems. A TOPS consists of a collection of interacting structural active objects (SAOs), and their behaviors are determined by the transition statements provided in their class definitions. Furthermore, SAOs can be *structurally and hierarchically* composed from their component SAOs like hardware components. These features allow SAOs to model components for circuit simulation more naturally than passive objects used in ordinary object-oriented programming. Also, we can easily add new kinds of components by using the inheritance mechanism. Executions of transition statements may be event- and/or time-driven, and hence digital, analog, and mixed-mode simulation is possible. Prototype simulation programs with graphical user interfaces have been developed as TOPS programs for digital, analog, and mixed-mode circuit simulation.

1. 서 론

객체지향 프로그래밍(Object-Oriented Programming, OOP)[1, 2, 3]은 혁신적인 새로운 소프트웨어 개발 기술로 정착되어 가고 있다. 객체지향 프로그래밍 기술이 지원하는 캡슐화된 클래스(Encapsulated Class), 상속(Inheritance) 그리고 병형성(Polymorphism)의 개념은 재활용이 가능한 고급 소프트웨어 모듈의 구현을 가능하게 한다. 또한 상태(State) 정보와 행위(Behavior) 정보를 캡슐화한 객체는 실객체(Real-World Object)를 기존의 소프트웨어 모듈에 비

* 이 논문은 부분적으로 1993년도 인공지능 연구센터의 위탁과제 연구비에 의해 연구된 것임.
† 정 회 원 : 덕성여자대학교 전자계산학과 조교수
논문접수 : 1995년 4월 13일, 심사완료 : 1995년 6월 5일.

해 보다 자연스럽게 표현하기 때문에 객체지향 기술은 소프트웨어 개발을 위한 적합한 프레임워크를 제공한다[4, 5]. 또한 생성 시스템(Production System)은 유연한 동기화를 요하는 다양한 병행 시스템(Concurrent System)의 개발에 적합한 것으로 알려져 왔다[6].

전이중심 객체지향 프로그래밍 시스템(Transition-based Object-oriented Programming System, TOPS)은 전이 규칙(Transition Rule), 동등 배정문(Equational Assignment Statement) 그리고 사건 루틴(Event Routine)을 사용하여 행위를 표현하는 객체지향 병행 시스템(Object-Oriented Concurrent System)이다. TOPS 방법은 객체지향 기술과 생성규칙(Production Rule) 기술을 통합한 것으로서 구조적 능동 객체(Structural Active Object, SAO)들의 구조적이고 계층적인 합성(Structural and Hierarchical Composition)이라는 개념이 그 핵심을 이룬다. 구조적이고 계층적인 합성에 의해 SAO들은 하드웨어 소자(Component)들과 같이 그들의 구성 SAO들로부터 생성될 수 있다. 여기서 하드웨어 소자는 능동적이면서 자율적인(Autonomous) 객체임에 주목할 필요가 있다. 구조적이고 계층적 합성 방법은 VLSI 칩이나 자동차와 같은 복잡한 전자 소자나 기계 장치를 설계하고 구현하는 방법으로 이미 널리 사용되고 있다.

각 SAO의 행위는 그 SAO의 클래스 정의에 명시되는 전이 문장(Transition Statement)에 의해 표현된다. 여기서 전이 문장은 조건-동작쌍(Condition-Action Pair)에 의해 표현되는 전이 규칙, 동등 배정문 또는 사건 루틴을 말한다. 동등 배정문으로는 SAO들의 상태 정보들 간의 간단한 불변의 관계를 명시하고 사건 루틴은 메시지(Message)에 의해 동작하는 행위를 표현한다. SAO들의 행위는 프로그래머가 명시하는 전이 문장에 의해 결정되기 때문에 새로운 형의 소자는 새로운 클래스를 정의하여 쉽게 추가될 수 있다. 따라서 기록기(Recorder)나 적분기(Integrator)와 같은 아날로그 소자도 SAO로 쉽게 생성될 수 있다.

또 다른 SAO의 핵심 기능은 각 SAO에 제공되는 전이 문장에 의해 그 SAO의 상태 정보 뿐 아니라 인터페이스 변수(Interface Variable)를 통해 다른 SAO들의 상태 정보도 접근할 수 있다는 것이다. 즉, 객체 간의 묵시적인 통신(Inter-Object Communication)이 가능하다. 묵시적인 통신을 위해 SAO들 간에 필요한 연결은 인터페이스 변수를 통해 해당 객체들을 결합시키면 된다. 인터페이스 변수는 하드웨어 소자의 단자(Terminal)와 같이 동작하며 구조적 합성을 위해 중요한 역할을 한다. TOPS 방법은 SAO들의 구조적 합성을 기본적으로 사용하는 반면 기존의 객체지향 프로그래밍은 수동 객체(Passive Object)들의 절차적인 인터페이스(Procedural Interface)를 사용한다. 각 SAO는 전체 프로그램의 제어 중 자신에 해당되는 부분을 전이 문장에 의한 행위 정보로 캡슐화 하기 때문에 이러한 SAO들을 인터페이스 변수를 통해 구조적이고 계층적으로 합성하는 것이 가능하다.

설계 자동화 도구는 공학 응용 분야에 이제는 필수적인 것이다. 예를 들어 디지탈 시스템 설계 분야에서 회로도 작성, 레이아웃(Layout), 설계 규칙 검증, 시뮬레이션 등을 위한 도구들이 개발되어 사용되고 있다[7, 8]. TOPS 방법은 이러한 설계 소프트웨어들의 라이프 싸이클을 통해 사용될 수 있는 프레임워크를 제공함으로써 설계에 필요한 작업(도구)들을 통합할 수 있는 환경을 제공한다.

아날로그와 디지탈 부분을 혼합하여 시뮬레이션하는 것은 일반적으로 어려운 것으로 알려져 왔다[9]. 따라서 아날로그와 디지탈 부분은 독립적으로 설계되어 시뮬레이션 후 통합되는 방식이 주로 사용된다. 논리 게이트(Logic Gate), 플립플롭(Flip-Flop) 등의 디지탈 소자와 적분기(Integrator) 등의 아날로그 소자는 SAO에 의해 표현될 수 있으므로 혼합 모드 시뮬레이터는 SAO들로 쉽게 구현될 수 있다.

TOPS 방식은 객체지향 병행 시스템의 구현을 위한 새로운 프로그래밍 환경으로서 이산 시뮬레이션(Discrete Simulation), 하드웨어의 논리 시뮬레이션, 시각 사용자 인터페이스[10], 공정 제어(Manufacturing Control)[11] 등의 다양한 응용 분야에 적용될 수 있다. 이러한 시스템들이 TOPS 방식에 의해 구현된다면 프로그램 크기를

크게 줄여 개발 기간을 단축시킬 수 있다. 본 논문에서는 TOPS 방식을 디지탈, 아날로그 그리고 혼합 모드 시뮬레이션에 적용하여 그 적합성을 보인다.

시뮬레이션에 널리 사용되고 있는 하드웨어 기술 언어인 VHDL[13]에 비해 TOPS 방식은 객체지향 기능인 상속과 병형성을 지원하는 것과 문법이 업계의 표준으로 정착되어 가고 있는 C++에 기초를 두고 있다는 장점을 제공한다.

제 2절에서는 TOPS 방법의 핵심 기능을 설명하고 제 3절에서는 TOPS 방식을 D-래치(D-Latch)와 D-플립플롭을 예로 들어 디지탈 회로 시뮬레이션에의 적용을 소개한다. TOPS 방식에 의한 아날로그 회로 시뮬레이터를 제 4절에서 기술하고 혼합 모드 시뮬레이션을 제 5절에서 다룬다. 제 6절은 본 논문의 내용을 요약한다.

2. TOPS의 핵심 기능

본 절에서는 TOPS 방식을 그 핵심 기능인 능동 행위의 명시와 구조적 합성을 중심으로 자세히 다룬다.

2.1 행위 명시

C++나 Smalltalk의 객체는 전달된 메시지에 의해서만 반응하는 수동 객체(Passive Object)이다. 메시지에 의해 반응하는 사건 루틴에 더하여 TOPS는 능동 객체의 행위를 명시하기 위해 전이 규칙(조건-동작 쌍)과 always 문(동등 배정문)을 사용한다. TOPS에서는 전이 규칙, always 문 그리고 사건 루틴을 전이 문장이라 한다.

2.1.1 전이 규칙

각 전이 규칙은 조건-동작 쌍으로서 조건부가 만족되면 동작부에 명시된 행위가 수행된다. 이러한 전이 규칙은 아토믹(Atomic)으로 수행되며 조건부에 사용된 조건 변수(Condition Variable) 또는 능동 값(Active Value)[13]이 변화될 때마다 기동된다(Activate).

transition *name* **when** (*condition*) {*statements*}

2.1.2 Always 문

객체들의 상태 정보들 간에 불변의 관계를 유지하여야 할 경우 이러한 불변의 관계에 대한 행위를 표현하는 간단한 방법으로 동등 배정문(Always 문)을 사용한다.

예를 들어, always 문을 사용하는 AND Gate 클래스는 TOPS의 기술 언어(Description Language)를 사용하여 (그림 1)과 같이 정의될 수 있다. 여기서 input1과 input2는 인터페이스 변수로서 이들을 통해 output을 출력 데이타로 하는 Gate 클래스의 객체들과 결합된다.

```
class Gate {
  public:
    bool output;
}
class AND Gate : public Gate {
  public:
    Gate *input1, *input2;
  private:
    always output=input1->output && input2->output;
}
```

(그림 1) AND Gate 클래스의 TOPS 기술
(Fig. 1) TOPS Description of Class AND Gate

Always 문은 전이 규칙과 같이 구현되며 전이 규칙의 간략형이다. 즉, always 문에 사용된 변수가 변화될 때마다 배정문의 수행이 이루어진다. AND Gate의 객체는 인터페이스 변수 input1과 input2에 연결된 Gate 객체들의 output을 직접 접근하여 이 값들이 변화될 때마다 논리적인 AND 연산을 수행한다. 하드웨어 시뮬레이션을 위해서는 전이 규칙보다 일반적으로 always 문이 사용된다.

2.1.3 사건 루틴

전이 규칙은 적어도 개념적으로는 상태 중심(State-Driven)적으로 기동된다. 즉, SAO는 다른 객체들에게 메시지를 전달하는 대신에 다른 객체들의 인터페이스 변수들을 통해 그들의 상태 정보를 직접 접근함으로써 그들과 묵시적인 통신을 할 수 있다. 이러한 방법은 명시적인 메시지 전달을 요하지 않지만 사건의 종류에 따라서는 동기적 함수 호출(Synchronous Function Call),

지연 호출(Future Call) 그리고 지연 배정(Future Assignment)에 의해 기동되는 사건 루틴으로 처리하는 것이 보다 바람직할 수 있다. 기동된 지연 호출과 지연 배정은 트리거(Trigger)된 전이 규칙과 같이 한번에 하나씩 수행된다.

동기적 함수 호출은 C++의 기존 함수 호출(메시지 전달)이며 호출과 즉시 호출된 함수가 수행된다. 반면, 지연 호출은 다음과 같은 형식의 비동기적(Asynchronous) 호출을 말한다.

〈*function-call*〉 **after** *delay*;

0이 아닌 지연 시간(delay)이 명시되면 지연 호출의 수행은 그 시간만큼 지연된다. 지연 배정도 지연 호출과 비슷하게 수행된다.

x = *expression* **after** *delay*;

배정은 명시된 시간만큼 지연된 후 수행된다. 비슷한 호출 방법이 VHDL[13]에서도 제공된다.

2.2 지역 제어(Localized Control)

기존의 객체지향 프로그램에서의 제어 흐름(Control Flow)은 메시지나 프로시듀어 호출에 의해 이동된다. 두 경우(메시지와 프로시듀어) 모두 외부의 자극(Stimulation)에 의해 기동되는데 이 자극은 궁극적으로 주 프로그램(Main Program)으로 부터 주어지게 된다. 수동 객체들 간의 메시지 전달에 의해 이동되는 이러한 제어 흐름은 시스템이 병행적으로 동작하는 객체들로 구성될 때 특히 복잡한 양상을 띄게 된다. 따라서 기존의 객체지향 프로그래밍은 제어 흐름을 적절히 캡슐화 또는 모듈화하지 못한다고 할 수 있다.

TOPS에서, 각 SAO는 데이타와 행위 뿐아니라 제어도 캡슐화 한다. 제어 흐름에 대한 주 프로그램에서의 전역적이고 절차적인 명시없이 각 SAO는 자기 몫의 제어를 갖고 있어서 필요한 작동(Operation)을 개시할 수 있다. 즉, 전이 문장에 의해 명시된 동작 조건이 만족되면 SAO는 외부 자극에 의해서가 아니라 스스로 행위를 수행할 수 있다.

지역 제어의 개념은 다음과 같은 흥미로운 점을 내포하고 있다. 기존의 객체지향 프로그램의 주 프로그램은 다른 클래스들과 매우 다른 형태로 작성되기 때문에 이 프로그램을 사용하는 보다 큰 프로그램의 서브시스템(Subsystem)을 나타내는 클래스로의 변환이 일반적으로 어렵다. 반면에 TOPS의 주 프로그램은 사례(Instance)이지만 그 정의 방식이 다른 클래스들의 정의 방식과 같기 때문에 TOPS의 주 프로그램의 정의에서 키워드 main을 class로 바꿈으로써 쉽게 클래스 정의로 변환할 수 있다. 따라서 이렇게 변환된 클래스의 객체는 보다 큰 클래스의 소자(서브시스템)로 사용될 수 있다.

2.3 구조적이고 계층적인 합성

TOPS 방식의 핵심 개념은 소프트웨어 모듈들의 구조적이고 계층적인 합성이다. 구조적이고 계층적인 합성을 통해 낮은 레벨의 모듈들로 부터 고급 레벨의 모듈들을 최소한의 노력으로 설계할 수 있다. VLSI 칩, 회로 보드(Circuit Board), 컴퓨터 시스템 등의 전자 소자 또는 시스템과 엔진, 자동차 등의 기계 소자 또는 시스템은 모두 이 방법에 의해 설계되고 공정된다. 이와 같은 구조적이고 계층적인 방법에 의해 소프트웨어 모듈인 SAO는 그들의 구성 SAO들로 부터 생성된다.

스스로 기능을 수행하는 복합 객체(Composite Object)를 지원하기 위해서는 그 구성 객체들이 능동 객체들이어야 한다. 이는 수동 객체는 어떠한 동작도 스스로 개시할 수 없기 때문이다. 여기서 VLSI 칩이나 자동차의 소자들은 구조적이고 계층적으로 합성이 가능하고 자기 몫의 제어 부분을 캡슐화한 능동 객체임에 주목할 필요가 있다. DOSE 시스템은 객체지향 시뮬레이션 환경이고[14] ObjectTime 시스템은 실시간 시스템의 개발을 위한 객체지향 환경이다[15]. 그러나 DOSE와 ObjectTime은 모두 메시지 중심의 구조적이고 계층적 합성을 사용한다.

SAO들의 구조적인 합성을 지원하기 위해서 하드웨어 소자의 단자와 같이 동작하는 인터페이스 변수는 중요한 역할을 하며 이들을 통해 SAO들이 연결된다. 각 SAO에 제공되는 전이 문장은 그 SAO의 상태 정보 뿐아니라 인터페이

스 변수를 통해 다른 SAO들의 상태 정보도 접근할 수 있기 때문에 객체간의 묵시적인 통신이 가능하다. SAO들 간의 필요한 연결은 인터페이스 변수에 해당 객체들의 사례 변수(Instance Variable)들을 결합하면 된다. 이러한 결합은 복합 객체가 그 구성 객체들로 부터 생성될 때 이루어지며 인터페이스 변수에 결합되는 객체는 다른 객체의 구성 객체일 수 있다. 객체들을 인터페이스 변수에 결합하는 것은 VLSI 칩들을 와이어 랩하는(Wire Wrapping) 것과 흡사하다.

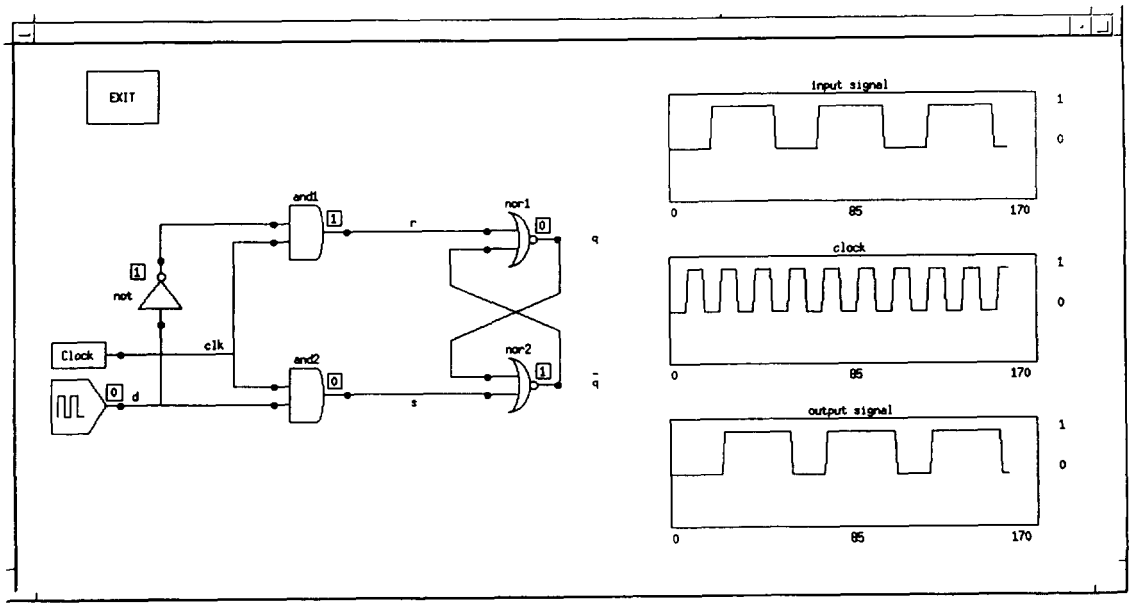
표준화 된 인터페이스를 사용한 능동 객체의 구조적 합성은 COX[16]에 의해 소개된 소프트웨어 IC의 실현을 위한 기초를 제공한다. SAO는 소프트웨어 객체이므로 하드웨어 객체보다 유연하다. 즉, 기존의 객체지향 프로그래밍의 상속, 병형성 등의 기능을 TOPS 클래스를 정의하는데 사용할 수 있다.

복잡한 시스템을 구성하기 위해 TOPS 방법은 기본적으로 SAO들의 구조적 인터페이스를 사용하는 반면 기존의 객체지향 프로그래밍은 수동객체에 제공되는 절차적 인터페이스를 사용한다. 기존 객체지향 프로그래밍의 클래스 정의에서는 인터페이스로 제공되는 프로시듀어(메소드)들의 수가 일반적으로 많다. 이러한 프로시듀어가 수행하는 기능을 모두 이해하고 기억하는 것은 쉬운 작업이 아니다. 구조적 인터페이스는 이러한 프로시듀어의 기능들이 몇개의 인터페이스 변수들에 묶여 흡수되는 것을 가능하게 하므로 시스템의 기술이 명확하고 간단하다.

TOPS의 복합 SAO는 다시 보다 큰 복합 SAO의 소자로 사용될 수 있다. 일반적으로 하드웨어의 설계 과정은 소프트웨어의 설계 과정에 비해 잘 정립되어 있다고 알려져 왔다. 그 주된 원인은 하드웨어인 경우, 그 설계 과정이 이러한 구조적이고 계층적인 합성에 전적으로 의존하기 때문이다. 결론적으로 구조적이고 계층적인 합성을 지원하기 위해 SAO의 능동성과 인터페이스 변수의 개념이 핵심을 이룬다.

3. 디지탈 회로 시뮬레이션

본 절에서는 디지탈 소자인 D-래치(DLatch)가 TOPS 프로그램(TOPS 기술 언어를 사용한 프로그램)으로 어떻게 구현되는 지를 먼저 설명하고 C++로 변역된 이 TOPS 프로그램을 설명한다. TOPS 프로그램으로 구현된 에지 트리



(그림 2) DLatch
(Fig. 2) DLatch

거형(Edge-Triggered) D-플립플롭도 예로 든다.

(그림 2)에 보인 바와 같이 DLatch는 두개의 Nor 게이트(nor1, nor2)로 구성되는 SR 래치와 두개의 And 게이트(and1, and2)로 구성되는 클록(Clock) 회로로 이루어 진다[17]. 클록 신호인 clk와 데이타 신호인 d가 입력 신호이고 q와 보수 신호(Complementary Signal)인 $\bar{q}$가 출력 신호이다. d가 0이고 clk가 1(High)이면 r=1, s=0, q=0 그리고 $\bar{q}$=1을 얻고 d가 1이고 clk가 1이면 r=0, s=1, q=1 그리고 $\bar{q}$=0를 얻는다. clk가 0(Low)이면 SR 래치는 입력 d로 부터 분리된다.

(그림 3)에 (그림 2)의 DLatch를 구현하기 위한 TOPS 클래스 정의를 보인다. 일반적으로 TOPS 클래스는 다음과 같은 세개의 부분으로 구성된다.

```
class DLatch {                    // class definition for a D-latch
  public:
    Int* clk;                     // input port for clock
    Int* d;                       // input port for D-latch input
    alias q=nor1.output;          // output port for D-latch output
    alias qNot=nor2.output;       // output port for negated D-latch output
  private:
    Not not with {input = d};
    And and1 with {input1 = &not.output, input2 = clk};
        and2 with {input1 = clk, input2 = d};
    Nor nor1 with {input1 = &and1.output, input2 = &nor2.output};
        nor2 with {input1 = &nor1.output, input2 = &and2.output};
}
```

(그림 3) DLatch 클래스의 TOPS 기술
(Fig. 3) TOPS Description of Class DLatch

1. C++와 같이 키워드 public:이 선행되는 인터페이스 부분은 입출력 포트(Port)를 명시한다. 입력 포트는 하드웨어 소자의 입력 핀(Pin)이 다른 하드웨어 소자의 출력 핀(Pin)에 연결될 수 있듯이 다른 소자의 출력 포트와 결합될 수 있다.
2. 클래스 바디는 사례 변수(Instance Variable)들로 구성된다. 이러한 사례 변수들은 이 클래스에 의해 생성되는 객체의 서브소자(Sub-component)를 표현한다.
3. 행위 기술 부분은 이 클래스 레벨에 추가되는 기능들을 명시한다.

DLatch 클래스의 인터페이스 부분은 DLatch가 입력 포트로 clk와 d를 갖고, 게이트 nor1과 nor2의 사례 변수인 output이 각각 출력 포트인 q와 qNot이 됨을 명시한다. DLatch 클래스의 바디는 Not 클래스의 사례인 not, And 클래스의 사례인 and1과 and2 그리고 Nor 클래스의 사례인 nor1과 nor2로 구성된다. 이러한 소자들은 with 구(Clause)로 정적인 연결이 명시된다. 예를 들어, not 게이트의 입력 포트 input은 DLatch의 입력 포트인 d와 연결되고 and1 게이트의 입력 포트인 input1과 input2는 not 게이트의 output과 DLatch의 clk에 각각 연결된다.

DLatch 클래스의 정의에서 DLatch의 행위는 상호 연결된 서브소자들에 의해 결정되기 때문에 자체 행위에 대한 명시는 없다. 그러나 일반적으로 TOPS 클래스는 다음 예들에서 보듯이 행위에 대한 명시를 할 수 있다.

DLatch 클래스에 의해 사용되는 클래스들의 정의를 설명한다. (그림 4)에 보인 Gate 클래스는 And, Nor 또는 Not 게이트 클래스들의 베이스 클래스(Base Class) 역할을 하며 게이트의 출력값은 출력 포트인 output에 의해 정의된다.

```
class Gate {                      // base class for classes And, Nor, Not, ect.
  public:
    Int output;                   // gate output value, output port
}
```

(그림 4) 베이스 클래스 Gate의 TOPS 기술
(Fig. 4) TOPS Description of Base Class Gate

```
class And : public Gate {         // class for AND gates
  public:
    Int *input1, *input2;         // input ports for input values
  private:
    always output = (input1->output && input2->output);
}
```

(그림 5) And 클래스의 TOPS 기술
(Fig. 5) TOPS Description of Class And

Gate 클래스에서 유도된 (그림 5)의 And 클래스는 Gate 클래스로 부터 출력 포트인 output을 상속 받는다. 또한 인터페이스 변수로서 다른 게이트의 출력 포트인 output과 연결이 가능한 입력 포트 input1과 input2를 갖는다. 이 TOPS

클래스는 서브소자를 사용하지 않고 always 문에 의해 And 게이트의 기능적인 행위를 정의한다. 입력값 중에 어느 하나라도 변화되면 always 문은 두 입력값의 논리적인 AND 연산을 수행하여 output의 값을 변환한다.

```
class Nor : public Gate {          // class for NOR gates
  public:
      Int *input1, *input2;  // input ports for input values
  private:
      always output = not (input1->output || input2->output);
}
```

(그림 6) Nor 클래스의 TOPS 기술
(Fig. 6) TOPS Description of Class Nor

(그림 6)에 보인 Nor 클래스도 And 클래스와 비슷하게 정의되며 always 문에 의해 정의되는 행위 명시가 And 클래스와 다르다. Not 클래스에 대한 TOPS 정의를 (그림 7)에 보인다.

```
class Not : public Gate {      // class for NOT gates
  public:
      Int *input;              // input port for input value
  private:
      always output = not (input->output);
}
```

(그림 7) Not 클래스의 TOPS 기술
(Fig. 7) TOPS Description of Class Not

이러한 TOPS 프로그램들은 TOPS 변환기에 의해 C++ 프로그램으로 변환된다.
구조적 능동 객체(SAO)의 동작을 설명하기 위해 C++로 변환된 And 클래스에 대해 아래에 기술한다.

```
class Gate: public Segments { // base class for classes And, Nor, Not, etc
  public:
      Int output;              // output value, output port
      Gate(int n): Segments(n) {};  // constructor
}
```

(그림 8) Gate의 C++ 클래스 정의
(Fig. 8) C++ Class Definition of Gate

(그림 8)은 Gate 클래스의 C++ 정의이다. 부모 클래스인 Segment는 게이트에 대한 그래픽 표현을 제공한다. 각 게이트의 모양은 선 세그먼트(Line Segment)들에 의해 이루어 진다.

```
class And: public Gate {           // class for AND gates
  public:
      Int *input1;                 // input ports 1
      Int *input2;                 // input ports 2
      And() : Gate(4) {};          // constructor
      virtual void initialize();   // initialization routine
      void whenInputChanged();     // functionality
}
```

(그림 9) And의 C++ 클래스 정의
(Fig. 9) C++ Class Definition of And

Gate 클래스로 부터 유도된 (그림 9)의 And 클래스의 Int 형은 정수에 대한 조건 변수를 나타낸다. 조건 변수는 트리서 리스트(Trigger List)라 불리는 함수 포인터(Pinter to function)들에 대한 리스트를 유지한다. 조건 변수의 값이 갱신되면 트리서 리스트의 원소(Element)에 의해 포인팅되는 함수가 수행된다. And 클래스인 경우, (그림 10)에 보인 함수 whenInputChanged()가 입력값이 변화될 때마다 동작된다. 즉, 트기거 리스트에 의해 always 문에 의해 명시된 행위가 구현된다.

```
void And::whenInputChanged() {
      output = (int) *input1 && (int) *input2; // compute output
}
```

(그림 10) And 게이트의 기능 정의 함수
(Fig. 10) Functionality Definition of an And Gate

```
void And::initialize() {
      Gate::initialize();          // base class initialization
                                   // trigger setups
      PROC pf=PROC(&And::whenInputChanged());
      input1->tl.addTE(this, pf, "whenInputChanged()");
      input2->tl.addTE(this, pf, "whenInputChanged()");
      }
```

(그림 11) And 게이트의 초기화 함수
(Fig. 11) Initialization Function of an And Gate

(그림 11)에 보인 함수 initialize()가 수행하는 주된 작업은 함수 기동을 위한 트리서들을 준비하는 것이다. 함수 whenInputChanged()는 입력값들 중 어느 것이라도 변화하면 기동되어야 하므로 트리거가 함수 addTE()에 의해 각 입력에 추가된다.

(그림 12)에 보인 DLatch 클래스의 C++ 정의에, clk, d, q 그리고 qNot의 인터페이스 변수들이 인터페이스 부분에 제공된다. 따라서 외부 클록은 clk에, 외부 입력 신호는 d에 연결될 수 있고 DLatch의 출력인 q는 다른 외부 소자의 입력으로 연결될 수 있다. 다음으로 DLatch를 이루는 서브소자들이 정의되고 함수 initialize( ) 의 정의도 포함된다. 함수 initialize( )의 주된 기능은 (그림 13)과 같이 서브소자들을 연결하는 것이다.

```
class DLatch : public Segments {
  public:
      Int* clk;          // input port for the clock
      Int* d;            // input port for the signal
      Int q;             // output value at q
      Int qNot;          // output value at qNot
      DLatch():Segment(0){};  // constructor
      void initialize();      // initialization
  private:
      VNot not1;         // Not gate with vertical orientation
      And and1, and2;    // And gates used for gating circuit
      Nor nor1, nor2;    // Nor gates used for SR flip-flop
}
```

(그림 12) DLatch의 C++ 클래스 정의
(Fig. 12) C++ Class Definition of a DLatch

```
void DLatch::initialize() {
  Segments::initialize();  // parent class initialization

  not1.input   = d;        // provide connections
  and1.input1  = &not1.output;
  and1.input2  = clk;
  and2.input1  = d;
  and2.input2  = clk;
  nor1.input1  = &and1.output;
  nor1.input2  = &nor2.output;
  nor2.input1  = &and2.output;
  nor2.input2  = &nor1.output;
}
```

(그림 13) DLatch의 초기화
(Fig. 13) Initialization of DLatch

(그림 14)에 DLatch를 테스트하기 위해 사용된 회로의 정의를 보인다. 이 회로에 대한 TOPS 프로그램을 구성하기 위해 DigiClock, SigGen 그리고 DigiRecorder의 세개 클래스들이 추가 정의된다. 사용자에 의해 명시되는 듀티 사이클(Duty Cycle)을 갖는 신호 발생기(Signal Generator)가 C++ 클래스인 SigGen에 의해 정의

```
class DLatchSys: public TopLevel {
  public:
      DigiClock  clk1;              // a clock
      SigGen     sgn1;              // a signal generator
      DLatch     dlch1;             // a DLatch
      DigiRecorder recrd1, recrd2, recrd3;  // various recorders

      DLatchSys(char* n): TopLevel(n) {};
      void initialize();
}
```

(그림 14) DLatch 포함하는 디지탈 회로의 C++ 클래스 정의
(Fig. 14) C++ Class Definition of Circuit Containing a DLatch

되고 클래스 DigiClock은 클래스 SigGen과 같다. Int 형의 신호를 그리기 위해 DigiRecorder가 사용된다. Float 형 신호인 아날로그 신호에 대해서는 Recorder가 사용된다. (그림 15)에 (그림 2)에 보인 것과 같은 소자들 간의 연결을 명시한다.

```
// external clock output is connected clock input port
dlch1.clk      = &clk1.output;
// signal generator output is connected to signal input port
dlch1.d        = &sgn1.output;
// external clock output is connected to the recorder input port
recrd1.input   = &clk1.output;
// signal generator output is connected to recorder input port
recrd2.input   = &sgn1.output;
// output of the D latch connected to recorder input port
recrd3.input   = &dlch1.q;
```

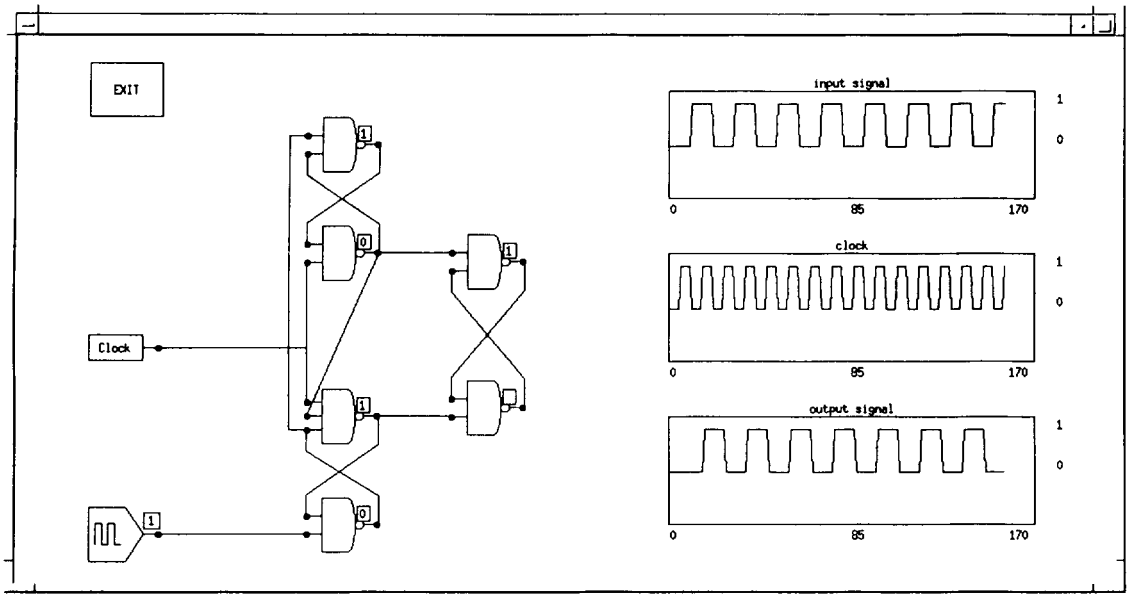
(그림 15) DLatchSys에 사용된 소자들의 연결
(Fig. 15) Interconnections Among the Components in DLatchSys

기록기(Recorder)도 역시 SAO로 구현되며 소자를 이 기록기에 연결하기 위해 서 소자의 출력(Output)은 기록기의 입력(input)에 연결된다. 기록기의 행위는 매 TOPS 시간에 따라 수행되는 함수 aosTimeChanged()에 의해 구현된다.

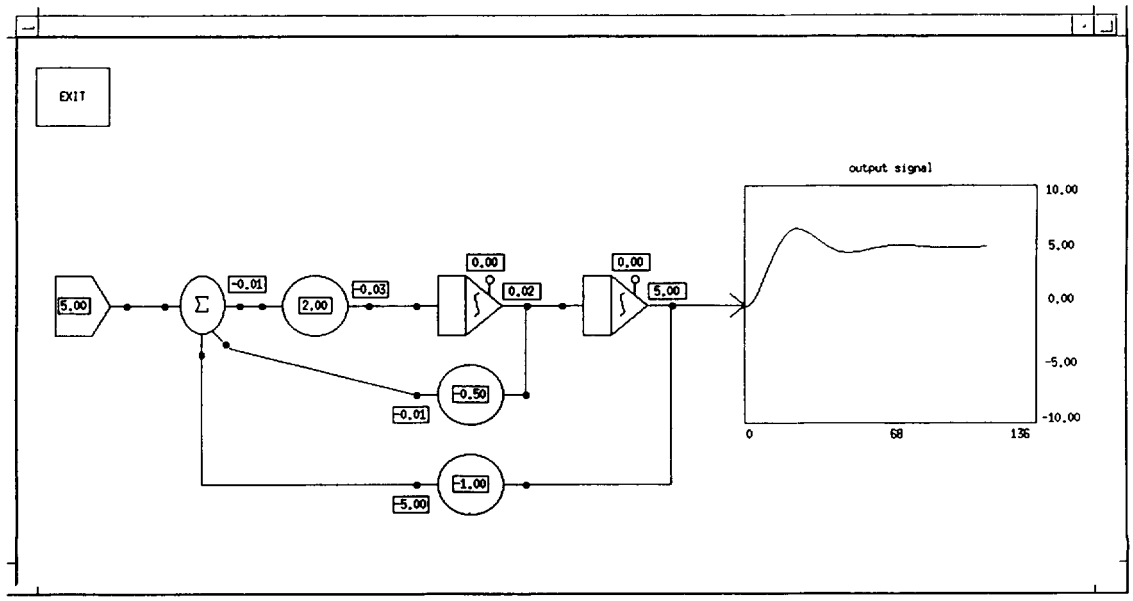
(그림 16)에 보인 것같이 에지 트리거형 D-플립플롭[17]을 역시 TOPS 프로그램에 의해 구현하였다.

본 절에서는 디지탈 회로 시뮬레이터를 구조적 합성 방법에 의해, 구성 소자들로 부터 합성하여 구성하는 방법에 대해 설명하였다. 구현된 D-래치와 D-플립플롭은 다시 보다 큰 회로에 계층적

합성에 의해 사용될 수 있다. 구조적이고 계층적인 합성 방법은 하드웨어 설계자들에게는 이미 잘 알려진 기술이며 TOPS에 의해 이 기술을 소프트웨어 시스템 구성에 효율적으로 적용하는 방법으로 제시되었다.



(그림 16) D 플립플롭
(Fig. 16) D Flip-Flop



(그림 17) 아날로그 시뮬레이터
(Fig. 17) Analog Simulator

4. 아날로그 회로 시뮬레이션

본 절에서는 TOPS에 의한 아날로그 시뮬레이터의 구현을 설명한다. 디지탈 시뮬레이터와 크게 다른 점은, 아날로그 시뮬레이터에서 의해 사용하는 적분기(Integrator) 등은 입력값의 변화뿐아니라 시스템 시간의 변화에 대해서도 변화될 때마다 출력값을 다시 계산하여야 한다는 것이다. (그림 17)에 보인 회로는 미분 방정식 $\dot{y}=2*(5-y-0.5\dot{y})$에 의해 표현되는 시스템을 시뮬레이션하는 것이다. 비슷한 회로가 Bogard [18]에 의해 사용되었다.

(그림 18)은 (그림 17)의 아날로그 시뮬레이터에 대한 TOPS 클래스 정의이며 기준값 소스(Reference-Value Source)인 src, 아날로그 신호 가산기(Adder)인 adr, 세개의 배수기(Scaler) scl1, scl2 그리고 scl3, 두개의 적분기 integ1과 integ2 그리고 아날로그 신호 기록기 recrdr로 구성된다. 이 소자들은 (그림 19)에 명시된 것처럼 연결된다. 소자의 출력은 입력을 포인팅하는 포인터 변수에 출력의 주소값을 배정함으로써 다른 소자의 입력에 결합된다. 적분기들의 초기값은 integ1.initVal = 0와 integ2.initVal = 0로 배정된다.

소자의 출력값인 v가 변화하면 v에 연결된 모든 소자들은 트리거를 수신하게 되고 새로운 v값에 의해 즉시 그들의 출력값을 계산한다. 이러한 소자들은 트리거를 다른 소자들에게 다시 전달하여 출력을 재계산하게 할 수 있으므로 변화가 전파되는(Propagating) 효과를 얻게 된다.

```
class AnalogSim: public TopLevel {      // analog simulator class
  public :
      FloatSource   src;  // source providing reference value 5.0
      Adder         adr;  // an adder to add three input values
      RScaler       scl1, scl2;     // reverse coefficient scalers
      Scaler        scl3;           // coefficient scaler
      Integrator    integ1, integ2; // integrators
      Recorder      recrdr;  // analog signal recorder
      AnalogSim(char* n):TopLevel(n), adr(3) {};//constructor
      void initialize();
}
```

(그림 18) 아날로그 시뮬레이터 AnalogSim의 클래스 정의
(Fig. 18) Class Definition of Analog Simulator AnalogSim

```
adr.input[0] = &src.output;
    //source output connected to input0 of adder
adr.input[1] = &scl2.output;
    // scaler2 output connected to input1 of adder
adr.input[2] = &scl3.output;
    // scaler3 output connected to input2 of adder
scl1.input = &adr.output;
    // adder output connected to scaler 1
integ1.input = &scl1.output;
    // scaler1 output connected to integrator input
integ2.input = &integ1.output;
    // integ1 output connected to integ2 input
scl2.input = &integ1.output;
    // integ1 output connected to scaler2 input
scl3.input = &integ2.output;
    // integ2 output connected to scaler3 input
recrdr.input = &integ2.output;
    // integrator2 output connected to recorder
```

(그림 19) 소자들 간의 결합
(Fig. 19) Interconnections Among the Components

```
class Integrator : public Segments {
  public :
      Float* input;     // input port pointer
      int    inputTime; // sampling time of newInVal
      float  newInVal;  // input value at inputTime
      float  oldInVal;  // input value at inputTime-1
      float  sum;       // intergrated value
      float  oldSum;    // intergrated value at inputTime-1
      Float  output;    // output port
      Float  initVal;   // initial value
      Integrator();     // constructor
      void initialize();  // initialization routine
      void integrate();   // perform integration
}
```

(그림 20) Integrator의 C++ 클래스 정의
(Fig. 20) C++ Class Definition of Integrator

```
void Integrator::integrate() {  // integrator functionality definition
   if(aosTime && aosTime == inputTime + 1) {  // new system time
      oldSum = sum;          // save integrated value at aosTime-1
      oldInVal = newInVal;   // move new input value to old input value
      inputTime = aosTime;   // update input time
   }
   newInVal = *input;        // read current input value
   if (aosTime) {            // not initiation time
      float deltaOut=(oldInVal+newInVal)/2.0*DELTA;//compute delta
      sum = oldSum + deltaOut;   // perform integration
      if(absf(sum-output) > EPSILON)  // propagate output value
         output = sum;        // if change is significant
   }
};
```

(그림 21) Integrator의 기능 정의
(Fig. 21) Functionality Definition of an Integrator

이러한 변화에 대한 전파는 각 소자의 출력을 Float 형 변수로 선언함으로써 가능하다. Float 형 변수는 float 형 값과 이 값이 변화하면 수행되는 함수에 대한 포인터 리스트를 유지한다. 적분기인 경우, 그 출력은 입력값이 변화될 때 뿐아니라 시스템 시간인 aosTime이 변화될 때마다 재계산된다.

(그림 20)에 C++ 클래스인 Integrator의 정의를 보여 아날로그 시뮬레이터에 사용되는 소자들이 어떻게 구현되는 지를 기술한다. 변수 input은 다른 아날로그 소자의 output에 대한 포인터이며 변수 output은 Integrator의 출력이다. 적분기 Integrator는 사다리꼴 적분 방법(Trapezoidal Integration Method)에 의해 사용될 두개의 연속 입력값(Successive Input Value)을 유지한다. 즉, 변수 newInVal은 inputTime때의 입력값을 유지하며 변수 oldInVal은 inputTime-1때의 입력값을 유지한다. 또한 변수 sum은 inputTime 때의 적분값을 유지하며 변수 oldSum은 inputTime-1때의 적분값을 유지한다.

(그림 21)에 사다리꼴 적분 방법의 적분 알고리즘에 대한 구현을 보인다. 함수 integrate()는 aosTime이 증가되거나 Integrator의 입력값이 변화될 때 기동되며 다음에 의해 적분값에 대한 변화값을 계산한다.

```
float deltaOut=(oldInVal+newInVal)/2.0*DELTA;
```

주어진 aosTime에 함수 integrate()가 한번 이상 기동될 수 있으며 연속되는 기동 때마다 보다 정확한 적분값인 sum을 계산한다. 적분 변화값인 deltaOut은 sum에 더해지는 것이 아니고 aosTime-1 때의 적분값인 oldSum에 더해진다.

계산된 적분값인 sum이 항상 output으로 지정되면 피드백 루프(Feedback Loop)를 포함하는 회로에 대해서는 각 aosTime에 함수 integrate()가 너무 많이 기동된다. 이 문제를 해결하기 위해 sum과 output의 차가 설정된 한계값인 EPSILON 내에 들어 올 때에는 Integrator의 출력을 sum으로 지정하지 않는다. 따라서 사용자에 의해 지정되는 EPSILON의 값은 수렴(Convergence)을 위한 시간과 Integrator의 적분값에 대한 정확도를 제어하게 된다.

(그림 22)의 Adder 클래스는 가산기를 생성하기 위해 사용되며 각 가산기는 입력값들에 대한 합을 계산하게 된다. 입력의 갯수는 사용자에 의해 생성자(Constructor)의 인수로 명시되어야 한다. (그림 23)의 함수 whenInputChanged()는 입력값들을 더하는 작업을 수행한다. 따라서 이 함수는 각 input에 의해 포인팅되는 Float 형 변수의 트리거 리스트에 추가되어야 한다.

```
class Adder : public VCObject {
  public:
      Float** input;   //array of input ports
      Float   output;  //output value
      int     nInputs; //number of inputs, specified by user

      Adder(int);      //constructor
      void initialize(); //initialization routine
      void whenInputChanged(); //perform addition
}
```

(그림 22) Adder의 C++ 클래스 정의
(Fig. 22) C++ Class Definition of Adder

```
void Adder::whenInputChanged(){ //functionality of an Adder
      float sum = 0;
      for(int i = 0; i < nInputs; ++i)
            sum += ((float) *input[i]);
      output = sum;
};
```

(그림 23) Adder의 기능 정의
(Fig. 23) Functionality Definition of Adder

배수기는 사용자에 의해 지정되는 배수 인자(Multiplication Factor) multFactor를 제공하는데 이 인자는 양수 또는 음수를 취할 수 있다. Output의 값은 multFactor와 input에 의해 포인팅되는 Float 형 변수 값의 곱이다. Adder 클래스인 경우, output의 값은 입력값이 변화될 때마다 재계산된다. 클래스 RScaler는 Scaler의 서브클래스(Subclass)이며 두 클래스 간의 차이는 그래픽 표현 뿐이다. (그림 16)에 보인 예에서 배수기의 배수 인자들은 각각 scl1.multFactor=−1.0, scl2.multFactor=−0.5 그리고 scl3.multFactor=2.0과 같이 지정된다.

FloatSource 클래스는 float 형 상수값을 제공하며 부모 클래스인 Source에서 유도된다. FloatSource는 입력 포트가 없으며 사용자가 FloatSource에 의해 제공되는 값을 src.output=

5.0과 같이 명시한다.

기록기는 적분기 integ2에 의해 생성되는 출력값을 그리는데 DigiRecorder와 같은 방법으로 역시 SAO로서 구현된다.

5. 혼합모드 회로 시뮬레이션

앞 절에서 디지탈 및 아날로그 회로 시뮬레이터가 어떻게 TOPS 프로그램으로 구현되는 지를 기술하였다. 아날로그 소자의 구현에서 설명하였듯이 SAO의 행위를 정의하는 전이 문장은 사건과 시스템 시간의 변화에 의해 기동될 수 있다. 따라서 사건에 의해 기동되는 디지탈 소자와 기본적으로 시스템 시간의 변화에 의해 기동되는 아날로그 소자가 섞인 혼합모드 시뮬레이터(Mixed-Mode Simulator)의 구현을 TOPS 프로그램으로 구현하는 것은 쉽다.

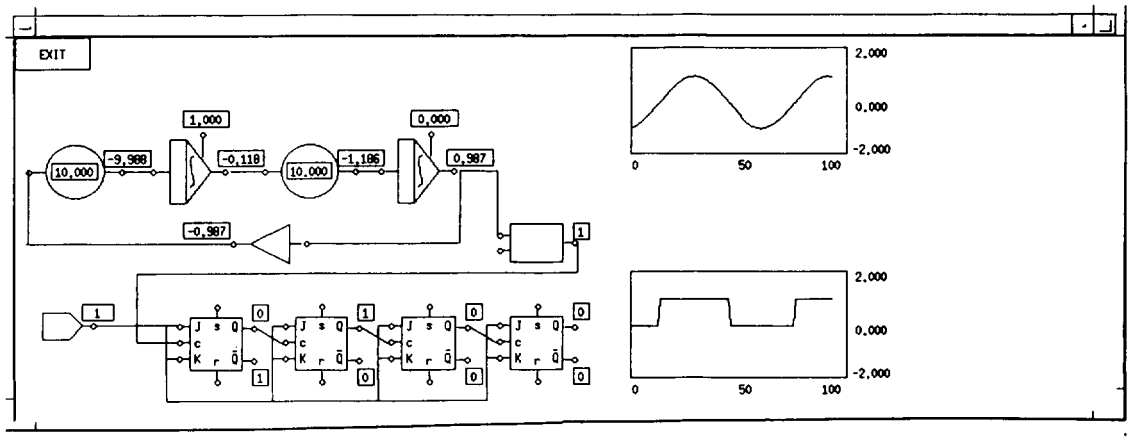
본 절에서는 (그림 24)의 디지탈 부분, 아날로그 부분 그리고 두 부분의 인터페이스로 구성되는 혼합모드 시뮬레이터의 구현을 설명한다. 아날로그 부분은 정현파 생성기(Sine Wave Generator)로 구성되고 디지탈 부분은 모드-16 리플 카운터(Modulo-16 Ripple Counter)로 구성되며 비교기(Comparator)가 디지탈 부분과 아날로그 부분 사이의 인터페이스로 사용된다.

(그림 25)에 보인 것같이 아날로그 부분은 클래스 Integrator, Scaler 그리고 Inverter의 사례를 사용하고 디지탈 부분은 클래스 JKFlipFlop과 IntSource의 사례를 사용한다. 또한 인터페이스 부분은 클래스 Comparator의 사례를 사용한다.

디지탈과 아날로그 부분의 인터페이스로 사용되는 비교기는 다음과 같이 동작한다. 비교기는 그 입력값 v_i가 기준값인 v_r보다 크거나 작은 경우에 대한 두 레벨의 값(논리적인 0과 1)을 출력값으로 제공한다. (그림 24)의 회로에서는 정현파 생성기의 출력값이 0을 지나 양수가 될 때 비교기의 출력이 0에서 1로 변환된다. 이와 같은 전이가 반대 방향으로 이루어 지면 출력값은 1에서 0으로 변환된다. 여기서는 이상적인 비교기를 가정하므로 불확실 범위(Uncertainty Range)는 없다.

(그림 26)에 보인 Comparator의 C++ 클래스 정의에서 입력 포트는 Float 형 변수에 대한 포인터이고 출력 포트는 Int 형 변수이다. 따라서 아날로그 소자의 출력은 회로의 입력 포트에 연결될 수 있고 회로의 출력 포트는 디지탈 소자의 입력 포트에 연결될 수 있다. 변수 refVal은 (그림 27)에 보인 행위를 수행하는 Comparator의 기준값 vr을 저장한다. 이 Comparator의 기준값은 사용자에 의해 comp.refVal=0과 같이 지정될 수 있다.

적분기 integ1과 integ2로 구성되는 아날로그 부분은 진폭(Amplitude) 1인 정현파를 생성한다. 정현파의 진폭은 Integrator의 초기 조건을 바꾸어 줌으로써 변환시킬 수 있고 파형의 주파



(그림 24) 혼합모드 회로 시뮬레이터
(Fig. 24) Mixed-Mode Circuit Simulator

수(Frequency)는 배수기의 multFactor를 바꿈으로써 변환이 가능하다. 클래스 Inverter는 아날로그 신호의 극성(Polarity)을 바꾼다.

모드-16 리플 카운터는 회로의 디지탈 부분을 형성하며 네개의 JK-플립플롭이 J=K=1의 조건으로 서로 연결되어 토글 방식(Toggle Mode)으로 동작한다. 초기화될 때 JK-플립플롭의 출력값 q는 0으로 지정된다. 입력 펄스(Pulse)에 의해 카운터는 24=16의 모든 상태를 거쳐 16번째 클록 펄스 후 다시 초기 상태로 되돌아 온다.

IntSource 클래스는 사용자에 의해 명시된 0 또는 1의 상수값을 제공하며 입력 포트를 갖지 않는다. (그림 24)에 보인 두개의 기록기들 중 위의 기록기는 정현파를 그리고 아래 기록기는 비교기의 출력을 그린다. JK-플립플롭의 출력도 기록기를 각 JK 플립플롭의 출력에 연결하여 측정할 수 있다.

```
class MixSimSys : public TopLevel {
  public:
      // analog part
      Integrator      integ1, integ2;
      Scaler          scl1,scl2;
      Inverter        inv;
      // interface between analog and digital part
      Comparator      comp;
      // digital part
      IntSource       src;
      JKFlipFlop      jkf1,jkf2,jkf3,jkf4;
      // recorders
      Recorder        recrd1;
      DigiRecorder    recrd2;
      MixSimSys(char *n) : TopLevel(n) {};
      void initialize();
}
```

(그림 25) 혼합모드 시뮬레이션 회로 MixSimSys의 클래스 정의
(Fig. 25) Class Definition for Mixed-Mode Simulation Circuit MixSimSys

```
class Comparator : public Segments {
  public :
      Float*  input;  // input port
      Int     output; // output value
      Float   refVal; // reference value for comparison

      Comparator(): Segments(0) {}; // constructor
      void initialize();            // initialization
      void whenInputChanged();      // functionality
}
```

(그림 26) Comparator의 C++ 클래스 정의
(Fig. 26) C++ Class Definition of Comparator

```
void Comparator :: whenInputChanged() {
      if(*input > refVal)
            output=1;  // output is high when Vi is above Vr
      else
            output=0;  // output is low when Vi is below Vr
      }
```

(그림 27) Comparator의 기능 정의
(Fig. 27) Functionality Definition of Comparator

6. 결 론

전이중심 객체지향 프로그래밍 시스템(TOPS) 방식은 디지탈, 아날로그 및 혼합모드 시뮬레이션 프로그램의 개발을 위한 새로운 프레임워크(Framework)를 제공한다. TOPS는 그 구성 소자인 구조적 능동 객체(SAO)들을 구조적이고 계층적으로 합성하여 만들 수 있다. SAO는 자활적(Self-Contained)이고 능동적이며 그 행위는 클래스 정의시 제공되는 전이 규칙, always 문 그리고 사전 루틴에 의해 정의된다. SAO는 하드웨어의 단자와 같은 기능을 수행하는 인터페이스 변수을 통해 연결된 다른 SAO들과 상호 묵시적으로 작용한다.

디지탈, 아날로그 및 혼합모드 시뮬레이터들이, AND, OR, NOT, NOR, EXOR, NAND 게이트, D-플립플롭, JK-플립플롭 등의 디지탈 소자와 적분기, 비교기 등의 아날로그 소자들을 SAO로 구현한 후 이들을 합성하여 TOPS 프로그램으로 구현되었다. 이상적인 소자(Ideal Component)들을 설계하였지만 향후 게이트 지연(Gate Delay) 등을 고려하여 실제 소자들에 대한 시뮬레이션을 TOPS 방법으로 구현하고자 한다.

끝으로, TOPS 방식에 의해 얻을 수 있는 장점들을 다음에 정리하였다.

첫째, 구조적이고 계층적인 합성은 복잡한 소프트웨어 소자들이 마치 하드웨어 소자인 것처럼 그들의 기본 빌딩 블록(Building Block)으로 부터 생성될 수 있도록 한다. TOPS 방법에 의한 시스템 기술(Description)은 매우 정확하며 시뮬레이션되는 회로를 밀접하게 반영한다. 더우기 이러한 기술은 소프트웨어 문장들이기 때문에 쉽게 갱신될 수 있다.

둘째, TOPS 프로그램은 TOPS 기술 언어(Description Language)나 C++로 작성되며 새로운 SAO를 정의하기 위해 이러한 언어들이 제공하는 객체지향 프로그래밍의 기능인 상속, 가상함수(Virtual Function) 등을 활용할 수 있다. 이러한 기능들은 VHDL과 같은 하드웨어 기술언어에서는 제공되지 않는 것들이다.

본 논문에서는 디지탈, 아날로그 및 혼합모드 시뮬레이션에 TOPS 방식이 성공적으로 적용될 수 있음을 보였다.

참 고 문 헌

[1] Goldberg, A., Robson, D. Smalltalk-80: The language and its implementation, Addision-Wesley, 1983.

[2] Meyer, B. Object-Oriented Software Construction, Prentice-Hall,1988.

[3] Stroustrup, B. The C++ Programming Language, Addison-Wesly, 1986.

[4] Booch, Object Oriented Analysis and Design, 2nd Ed., Benjamin/Cummings, 1994.

[5] Rumbaugh, J.,et al. Object-Oriented Modelling and Design, Prentice Hall, 1991.

[6] Zisman, M.D. Use of production systems for modelling asynchronous, concurrent processes, In Pattern-Directed Inference Systems, Waterman,D.A. and Hayes-Roth, F.(Eds), Academic Press, pp. 53-69, 1978.

[7] Bloom, M. Mixed-mode simulators bridge the gap between analog and digital design, Computer Design, Vol. 26, pp. 51-63, Jan. 1987.

[8] Dave, M. Mixed-mode simulation on a PC-based workstation, Electronics and Power, Vol. 32, pp. 523-526, July 1986.

[9] Goering, R. A full range of solutions emerge to handle mixed-mode simulation, Computer Design, Vol. 27, pp. 57-65, Feb. 1, 1988.

[10] Choi, S.and Minoura, T. User interface system based on active objects, Proc. 2nd Symp. on Environments and Tools for Ada, Jan. 1992.

[11] Minoura, T., Choi, S., and Robinson, R. Structural active-object systems for manufacturing control, Integrated Computer-Aided Engineering, 1, 2, pp. 121-136, 1993.

[12] Eum, D., Pargaonkar S., and Minoura, T., Structural active-object systems for mixed-mode simulation. Tr. 93-40-01, Dept. of Computer Science, Oregon State Univ., 1993.

[13] IEEE Standard VHDL Language Reference Manual.IEEE Std. 1076, 1987.

[14] Kunz, J. C., Kehler, T. P., and Williams, M. D. Applications development using a hybrid AI development system. The AI Magazine, 5, 3, 1984.

[15] Cox, B. J. Object Oriented Programming : An Evolutionary Approach, Addison Wesly, 1987.

[16] Gschwind, H.W.,and McCluskey, E.J. Design of Digital Computers, Springer-Verlag, 1975.

[17] Bogard, T.F. Computer Simulation of Linear Circuits and Systems, John Wiley and Sons, 1983.

[18] Taub, H., and Schilling, D. Digital Integrated Electronics. McGraw-Hall, 1981.

음 두 헌

1984년 서강대학교 전자공학과 졸업(학사)

1987년 미국 Oregon State University 컴퓨터공학과(석사)

1990년 미국 Oregon State University 컴퓨터공학과(박사)

1991년~92년 한국전자통신연구소 인공지능연구실 선임연구원

1992년~현재 덕성여자대학교 전산학과 조교수

관심분야 : 데이타 모델링, 객체지향 시스템