韓國 OA學會 論文誌
第4卷 第4號, 1999.12.

99-4-4-1-8

게이트 레벨 디지털 회로의 기술방법 및 시뮬레이션

권 승 학*, 이 명 호**

A Description Technique and It's Simulation of Gate Level Digital Circuits

Seung Hak Kwon* , Myung Ho Lee**

요 약

본 논문은 게이트 레벨 디지털 시스템의 동작기술과 그 동작결과를 검증 할 수 있는 시뮬레이터를 작성하는데 목적을 두고 있다. 기술언어로부터 목적코드를 얻기 위하여 번역기를 구성한 바 이의 구현을 위하여 UNIX의 YACC를 이용하였으며 중간 목적 파일을 번역기와 시뮬레이터의 중간과정으로 삼아 응용범위를 넓힐 수 있도록 하였다. 시뮬레이션 대상으로 전가산기와 3진 계수기를 사용하였다.

Abstract

The purpose of this study is to build a description technique and to make a simulator, which can simulate and verify the behavior of gate level digital system. To get the object code from the input description language, we build a translator. To do this, we used YACC of the UNIX parser generator, and made an intermediate code in the mid-process between translator and simulator to extend the range of application.

For experimental models, we used the Full-Adder and Modulo-3 Counter.

* 대원과학대학 전산정보처리과 조교수
** 청주대학교 정보통신과 교수
논문접수:1999.10.28 심사완료:1999.12.3

Ⅰ. 서 론

디지털 설계기술의 발전과 더불어 디지털 시스템이나 컴퓨터구조를 설명, 설계하기 위해 많은 하드웨어 기술언어

(HDL : Hardware Description Language)들이 출현하였다.[1] 1960년대 후반부터 시작된 이러한 움직임은 다음의 몇 가지로 요약된다.

- DDL(Digital system Design Language; Duley, 1967;Dietmeyer 1971) —non-procedural model
- CDL(Computer Description Language; Chu 1974)—non-procedural model
- AHPL(A Hardware Programming Language; Hill 1974)—APL based language
- ISP(Instruction Set Processor; Bell, 1971)
- ISPS(Instruction Set Processor and it's Successor; Barbacci, 1981) —ALGOl 60 based language

그러나 이들 많은 언어들은 체계화되지 못한 언어구조와 다른 시스템으로의 이식이 용이하지 않은 이유로 보급이 저조한 편이다. 하드웨어기술 언어집단을 나타냄에 있어 디지털 시스템을 집적 도에 따라 분류하면 표 1과 같이 분류하는데 대부분의 언어들이 레지스터 전송언어 집단(register transfer language group)에 속해 있다.

〈표 1〉 디지털 시스템의 분류
〈Table. 1〉 CLASSIFICATION OF DIGITAL SYSTEM

분 류	집 적 도
Gate(SSI) Level	$10^1 \sim 10^2$
Register transfer(MSI) Level	$10^2 \sim 10^3$
LSI Level	$10^3 \sim\sim 10^6$
VLSI Level	10^6 이상

언어를 설계함에 있어 범용성이 뛰어난 파스칼언어 형태를 빌려 디지털 시스템을 기술하고 최하위계층의 시뮬레이션을 통해 상위계층의 시스템들에도 시뮬레이션 방법을 제시하고자 한다. 본 논문에서는 디지털 시스템의 기술용 알고리즘, 알고리즘 변환기의 구성 그리고 시뮬레이터의 구성을 보이고자 하며 전체적인 시스템의 구성은 그림 1과 같다.

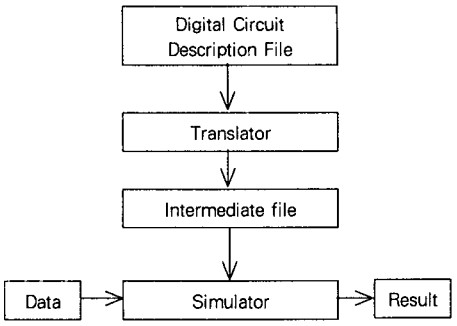


그림 1. 시뮬레이터의 구성
Fig. 1 CONFIGURATION OF SIMULATOR

Ⅱ. 디지털회로 기술언어

2.1 디지털시스템 기술언어의 종류

대부분의 디지털 시스템 기술언어들이 APL이나 ALGOL, PASCAL, C의 형태를 빌려쓰고 있는데 특히 PASCAL은 컴퓨터 시스템 및 여러 가지 다른 논리적인 시스템들을 설명하기에 가장 적절한 언어로 평가받고 있다.

1981년 Barbacci는 Syntax and Semantics of CHDL'S에서 하드웨어 설계용 언어의 요구 조건으로 다음 3가지를 들었다.[9]

- 판독성(readability):블럭 구조의 언어로서 정확하고 간결해야 한다.
- 범용성(generality):넓은 범위에 응용되어야 하고 여러 계층의 시스템을 지원하며 프로그래머가 아니더라도 쉽게 작성 할 수 있어야 한다.
- 유사성(fidelity):실제 시스템과 유사한 구조를 갖추어 평소 사용하던 기법, 구조, 설계방식 등을 응용할 수 있어야 한다.

게이트레벨 언어는 하위계층의 시스템을 기술하고 있기 때문에 고급 언어적 요소가 많이 필요한 것은 아니지만 상기 3가지 요건을 고려하여 수식과 기본 구조는 파스칼의 형태를 빌어 쓸 것이다.

2.2 언어의 요소

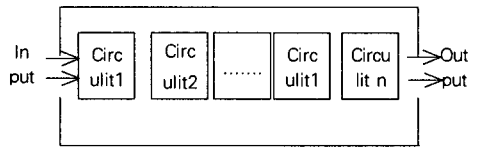
게이트레벨 언어는 디지털 시스템의 분류상 최하위 시스템으로 많은 요소가 필요한 것은 아니다. 다만 기본적으로 필요한 언어의 요소를 살펴보면 다음과 같다.

(1)언어구조

언어의 구조는 파스칼의 형태를 빌어 그림 2의 (a)와 같은 모양이 될 것이다. 게이트레벨 시스템의 언어를 요소별로 나누면 입력부분, 출력부분, 회로부분으로 나누어지는데 그림 2의 (b)와 비교하여 보면 INPUT은 입력부분을 설명하고 OUTPUT은 출력부분을 설명한다.

```
MODULE module_identifier;.....머리부분
INPUT  input_list; .....입력부분
OUTPUT output_list;.....출력부분
CIRCUIT circuit_identifier;.....함수부분
            :
            :
BODY.....주프로그램
    connection_expressions;
END;
```

(a) 언어의 구조



(b) 모듈의 구조
그림 2. 언어의 구조
Fig. 2 LANGUAGE STRUCTURE

그리고 함수 부분은 각 회로의 기능을 설명하며 주프로그램 부분에서 입력과 회로, 회로와 출력부분의 연결모양을 설명하여 준다. 또 머리부분에 'MODULE'이라고 표기했듯이 게이트레벨 기술언어의 가장 큰 단위는 MODULE이다. 모듈은 회로와 입출력 그리고 그 연결로 이루어진 시스템을 이르는 말이다.

(2)입출력 자료

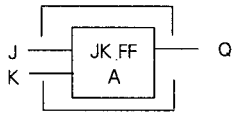
입력과 출력자료의 표현은 'INPUT' 또는 'OUTPUT'이라는 선행자(prefix)에 입출력 변수 이름을 ','로 구분하여 나열하고 끝에 ';'이라는 종결자(sufix)를 붙이는 방법으로 선언한다. 이때 각 변수의 논리적 기억용량은 1 비트이다.

입출력 자료는 다시 정적변수와 자동변수로 나누어진다.[13] 자동변수는 수행 블록의 종료 시 기억공간이 자유공간으로 넘어가는 변수로 함수 호출 시 기억공간을 효율적으로 사용하기 위하여 이용된다.

정적변수란 블록의 수행시작과 종료에 상관없이 기억공간에 항상 존재하는 변수로 기억력이 있는 플립플롭 같은 함수를 호출하는데 쓰여진다. 정적변수는 변수의 이름에 중괄호를 씌워 정의한다.

(3) 함수의 구조

함수는 'CIRCUIT'로 표현되는 것 이외에는 'MODULE'의 구조와 차이가 없다. 단지 유의해야 할 점은 인수의 전달(parameter passing)이다. 인수의 종류에는 value parameter 와 variable parameter가 있는데 variable parameter의 경우 중괄호로 묶어 주어야 한다. 입력인수의 경우 이 두 가지 인수가 모두 쓰이지만 출력인수의 경우는 value parameter만 사용된다. 그림 3은 실제 회로의 예와 인수의 사용 방법이다.




```
CIRCUIT JK FF;
INPUT  J,K,{A};
OUTPUT Q;
BODY
END;
```

(a) 기능모듈 (b) 모듈 기술문
그림 3. 기능 모듈과 기술문
Fig. 3 FUNCTIONAL MODULE AND IT'S DESCRIPTION

그림 3의 (b)에서 입력인수 {A}는 JK플립플롭의 상태를 의미하며 variable parameter이다.

(4)연산자

게이트레벨의 기본연산자인 AND, OR, NOT, EX_OR는 연산자의 형태가 아닌 함수의 형태로 이루어진다. 기본 호출형태와 연산자의 표현형태는 다음과 같다.

```
operator_identifier(operand_list)
z := or(and(a,b),and(c,d));
```

3.3 조합회로의 표현

기본연산자를 함수의 형태로 지원하면 기술구조 자체가 논리회로의 구조와 유사하며 번역기의 설계에 더 용이하다는 이점이 있다. 이와 같은 것은 단일소자 또는 하나의 출력을 가진 것을 다룬 것이지만 다음의 다중출력 경우를 생각해 보자.(그림 4)

```
CIRCUIT HA;
INPUT   X,Y;
OUTPUT  C,S;
BODY
  C := and(A,B);
  S := ex_or(A,B);
END;
C1 := HA(A,B).C;
S1 := HA(A,B).S;
```

그림 4. 다중 출력함수
Fig. 4 MULTI-OUTPUT FUNCTION

다중 출력함수는 파스칼에서 레코드의 형태로 출력을 갖는 함수처럼 생각하면 될 것이다. 지금까지 고찰한 것을 토대로 조합회로의 기술방법을 연구하면 조합회로는 기본적으로 함수의 형태를 지원한다는 것이다. 파스칼 언어의 문법에서 함수는 그림 5의 (a)와 같은 구조를 가지고 있다.

```
HEADER function identifier(parameter_list):
                output-type;
DEFINITION  var
                variable_list  ;
ALGORITHM  begin
                statement_list ;
           end;
```

(a) 파스칼 함수구조

```
HEADER ──┬─ Circuit  identifier        ;
         ├─ Input  input_parameter     ;
         └─ Output  output_parameter   ;
ALGORITHM ──── BODY
         ├─ circuit_description_function_list ;
         └─ end ;
```

(b) 회로기술용 함수구조
그림 5. 회로기술용 함수구조 비교
Fig. 5 CONFIGURATIONS OF FUNCTION STRUCTURE

그림 5의 (a)와 (b)를 비교하여 볼 때 파스칼 함수의 구조를 Header와 Definition, algorithm part로 나누어 볼 수 있다면 조합회로를 설명하는 회로기술용 함수구조 형식 역시 Definition 부분이 생략된다는 것 이외에는 거의 같다고 볼 수 있다.

2.4 순차회로의 표현

순차회로도 조합회로와 마찬가지로 함수의 형태로 지원 될 수 있다. 조합회로와의 차이점이라면 입력 parameter에 variable parameter를 사용하여 플립플롭의 상태를 기억한다는 것이다. RS_FF A의 출력이 Q에 연결되고 입력을 R,S라 할 때 Q=RS_FF(R,S,A); 처럼 표현 할 수 있다. 위에서 기술한 RS_FF는 기본적으로 제공되는 기능으로써 입력 클럭이 생략된 형태이다. 확장된 형태의 FLIP-FLOP들을 표현하기 위해서는 조합회로의 표현에서와 같이 'CIRCUIT'라는 단위로 기술해 주어야 한다. (그림 6)

```
CIRCUIT    clocked_rs_ff;
INPUT      cp,r,s,[S];
OUTPUT     q;
BODY
   q :=rs_ff(and(R,CP),and(S,CP),S);
END;
```

그림 6. 동기식 RS_FF의 표현
Fig. 6 DESCRIPTION OF CLOCKED RS FLIP FLOP

그림 6과 같은 방법으로 확장된 플립플롭을 표현한다면 어떤 플립플롭이든지 표현이 가능하다. 만약 그림 6에서 기술한 함수를 호출한다면

A:=clocked_rs_ff(CP,R,S,B);

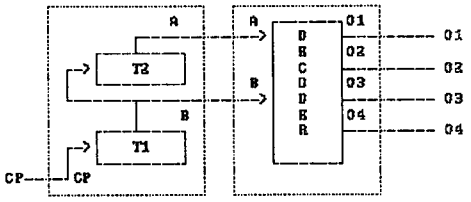
와 같은 형태로 사용 할 수 있을 것이다. 이 때 입력인수 S는 '['과']'로 묶여 있는데 이것은 variable parameter임을 나타내는 것이다.

2.5 기능 모듈의 연결표현

앞의 2.3절과 2.4절에서는 기본 요소별로 기능을 정의하였다. 하지만 디지털 시스템은 이들 기본 요소들만으로 이루어진 것이 아니라 이들의 조합으로 이루어진 기능 모듈들이 서로 연결되어 이루어진 것이다. 이러한 연결의 표현은 정의된 기능의 호출과 그 값의 전달 등으로 이루어지며 그림 7에 예를 보였다. 기능 연결모듈에서 추가로

생각하여야 할 것은 플립플롭이다. 플립플롭은 기억력을 가지고 있는 요소로 MODULE 부분에서 선언해야 한다. 그렇지 않으면 현재의 값을 참조할 방법이 없어지고 만다. 그리고 이것은 MODULE 블록 내에 선언하여야 참조가 가능하다.

그림 7를 보면 'FLIP_FLOP'이란 자료형태를 준비하여 사용하고 있다. 그리고 MODULE의 BODY 블록 내에서 t_ff란 함수는 한 번만 호출하였는데 이것은 한번 호출된 후에는 플립플롭의 상태가 기억되어 있기 때문이며 다시 호출하면 그 상태 값이 변하고 만다.



(a) 계수기와 디코더 회로연결

```
MODULE  counters;
INPUT   cp;
OUTPUT  01,02,03,04;
FLIP_FLOP (T1),(T2);

CIRCUIT  JK_FF;
INPUT    j,k,cp,(S);
OUTPUT   Q;
BODY

Q:=rs_ff(and(k,s,cp),and(j,not(s),cp),S);
        END;

        CIRCUIT  T_FF;
        INPUT    cp,(S);
        OUTPUT   Q;
        BODY
          Q:=jk_ff(1,1,cp,S);
        END;

        CIRCUIT  decoder;
        INPUT    a,b;
        OUTPUT   01,02,03,04;
        BODY
          01:=and(not(a),not(b));
          02:=and(not(a),   b  );
          03:=and(   a  ,not(b));
          04:=and(   a  ,   b  );
        END;

        BODY

01:=decoder(t_ff(CP,T1).Q,t_ff(CP,T2).Q).01;
          02:=decoder(T1,T2).02;
          03:=decoder(T1,T2).03;
          04:=decoder(T1,T2).04;
        END;
```

(b) 프로그래밍

그림 7. 기능 모듈들의 연결표현
Fig. 7 CONNECTIVE DESCRIPTION OF FUNCTIONAL MODULES

Ⅲ. 알고리즘 번역기의 구성

3.1 번역기의 개요

알고리즘 번역기는 디지털회로 기술파일을 원시 프로그램으로 입력받아 중간 목적파일을 만들어 내는 역할을 담당하게 되는데 그 구성은 그림 8과 같다.[11][12]

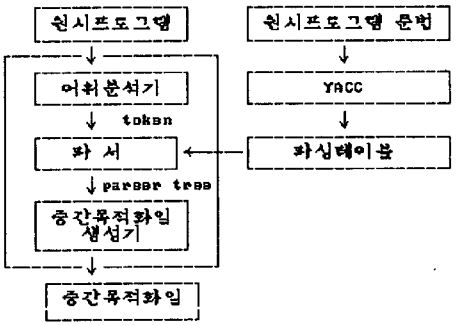


그림 8. 알고리즘 번역기의 구성
Fig. 8 CONFIGURATION OF ALGORITHM TRANSLATOR

보통의 컴파일러에서는 코드 최적화 단계가 더 추가되지만 여기서는 취급하지 않기로 한다. 다만 중간 목적파일 생성기에서 수식의 최소화 정도는 하게 될 것이다. 또 의미분석 단계도 간단히 다루어질 것이다. 각 단계별로 역할을 살펴보면 다음과 같다.

첫 번째, 어휘분석 단계는 원시 프로그램을 읽어들여 프로그램 인식의 가장 작은 단위인 토큰(token) 단위로 나누어 파서에게 전달하게 된다. 두 번째, 구문분석 단계는 토큰을 입력으로 각 문법에 알맞은 나무구조(tree)의 리스트를 만들게 된다. 이때 원시 프로그램의 문법은 Unix System의 Compiler-Compiler인 Yacc의 도움을 얻어 파싱 테이블을 구성하고 파서가 이를 이용하는 방법을 채택하였다.[15]

세 번째, 중간목적 파일생성기는 파싱 과정에서 만들어 낸 Tree 구조를 순회하여 중간어 테이블을 구성하는데 중간 목적파일은 바로 이 중간어 테이블과 거의 동일한 형태를 취하게 될 것이다.

3.2 어휘 분석기

어휘 분석기의 역할은 정확하게는 원시 프로그램의 문자열을 조사하여 토큰이라는 단위 크기로 잘라내고, 이를 예약어 테이블에서 찾아 그 코드를 파서에게 전달하는 것이다. 원시언어 어휘분석기의 구성도를 보면 그림 9와 같으며 토큰이 명칭 일 때는 명칭테이블에 그 명칭의 이름을 구성하게 된다.[8]

일반적인 컴파일러에서는 2개의 토큰코드를 파서에 전달하게 된다. 첫 번째 코드는 의미코드로 해석되고

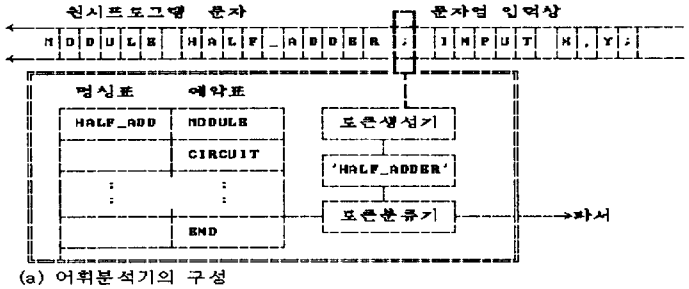
두 번째 코드는 내용코드로 해석되는데 실제로 명칭토큰 외에 내용코드는 의미가 없다. 명칭코드의 내용코드는 값을 가지고 있지 않다.[8]

그림 9의 (b)를 보면 1에서 18까지는 부코드가 모두 0인데 19부터는 Table의 위치가 부코드로 할당이 되는 것이다. 하지만 이것을 그림 9의 (c)와 같이 연속된 Table처럼 생각하면 하나의 코드로 이를 수용 할 수 있다.

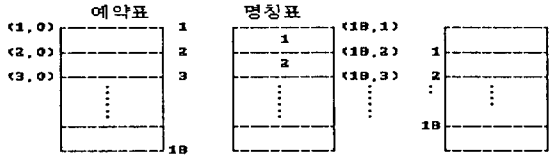
명칭테이블의 위치 값 즉, 부코드 값은 다음 수식으로 얻어 낼 수 있을 것이다.

부코드= 주코드 - Reserved Table의 Top

어휘분석기의 구성 방법은 크게 2가지가 있는데 하나는 토큰파일을 만들어 파서에 전달하는 방법이고 또 하나는 파서에서 어휘분석기를 부르는 방법이다. 게이트레벨 번역기는 파서에서 어휘분석기 부르는 방법을 사용할 것이다. 이 방법의 장점은 속도가 빠르고 오류 목록파일을 구성하기가 용이하다는 것이다.



(a) 어휘분석기의 구성



(b)어휘분석기의 자료구조 (c)연결된 자료구조

그림 9. 어휘 분석기의 구성도

Fig. 9 CONFIGURATION OF LEXICAL ANALYSER

3.3 파싱

파싱이란 어휘 분석에서 넘겨받은 토큰을 가지고 각 문법에 맞게 분석나무를 구성하는 단계이다. 파싱은 크게 LL 파싱과 LR 파싱으로 나누어지는데 이 두 가지 파싱의 특성을 표 2에 나타내었다.

게이트레벨의 번역기는 LR 파싱을 채택하여 사용하는데 그 이유는 LL 파싱보다 프로그래밍이 용이할 뿐만 아니라 Unix 시스템의 파서 생성기 도움을 얻을 수 있기 때문이다. 파싱을 하기 위해서는 파싱 테이블이 필요한데 이때 이 파싱 테이블은 유닉스의 파서 생성기 YACC(Yet Another Compiler-Compiler)를 통해 얻는다.

〈표 2〉 LR 파싱의 특성
〈Table. 2〉 CHARACTERISTICS OF LL,LR PARSING

	LR 파싱	LL 파싱
Tree 생성	leaves→Root	Root→leaves
생성 순위	Right most delivation	Left most delivation
특 성	모든 context-tree 문법에 대해 구성가능	좌 반복시 좌 반복 제거

여기서 얻어낸 파싱 테이블은 유닉스의 C언어로 구성하는 것에 기준을 두고 만들었기 때문에 게이트 레벨 번역기 프로그램 작성언어로 쓰인 파스칼에서 쓸 수 있도록 다시 구성해야 한다.

파싱 테이블의 구성 방법은 그림 10과 같다. 파싱 테이블의 구성은 Action Table과 Goto Table로 나누어지는데 Action Table의 내용은 Shift와 reduce로 다시 나누어진다.

state	tokens	Grammars
1 2 :	shift or reduce	

action goto
그림 10. 파싱 테이블의 구성
Fig. 10 STRUCTURE OF PARSING TABLE

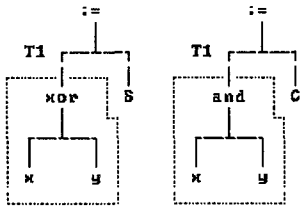
Shift의 동작은 해당하는 토큰 또는 문법 노드를 스택에 집어넣고 지시하는 State로 가는 것이고 reduce는 해당되는 문법의 가지 수만큼 스택에서 꺼내어 나무(Tree) 구조로 만드는 것이다.

파싱 과정에서 오류가 없이 끝나게 되면 Tree 구조의 리스트들이 만들어지는데 각 노드(node)별로 문법코드를 가지게 되며 다음 단계인 중간 목적파일 생성단계에서 이 코드에 의해 중간 어 Table을 구성하게 된다.

3.4 중간 어 코드 변환

디지털회로의 시뮬레이션은 회로기술용 언어를 직접 번역하여 시뮬레이션 하는 방법을 쓰지 않고 번역 부분과 시뮬레이션 부분을 서로 독립시켜 운용토록 되어 있다.

중간어 코드 변환이란 번역기 구성의 최종 단계로 시뮬레이터의 입력으로 적합하게 쓰이도록 하였다. 이와 같이 만들기 위해서는 파싱과정의 분석 Tree를 중간어 Table로 구성하여야 하는데 변환의 예는 그림 11과 같다. 이때 중간어 Table의 형태로 3번지(3 operand) 방식을 채택하였는데 이와 같이 만들어진 형태는 X:=Y+Z와 같은 수식으로 나타낼 수 있고 모든 수식은 상기 수식의 나열로 나타낼 수 있기 때문에 게이트 레벨 언어의 수식표현에 적합하다고 볼 수 있다.[13]



(a) 파싱 Tree

수행종류	OP1	OP2	저장위치
xo	X	Y	T1
:=	T1		S
and	X	Y	T1
:=	T1		C

(b) 중간어 Table
그림 11. 파싱 Tree 와 중간어 Table
Fig. 11 PARSING TREE AND INTERMEDIATE TABLE

각 구문별 중간어 테이블 표현 양식은 표 3과 같은 형태로 구성 될 것이다. 그리고 중간어 테이블의 배열은 함수테이블이 먼저 놓이고 모듈테이블이 가장 나중에 위치하게 된다.

〈표 3〉 중간어 테이블 내용
〈Table. 3〉 CONTENTS OF INTERMEDIATE TABLE

수행 종류	필요 인수 목록	수행 내용
MODULE	모듈이름	수행레벨 = 1
INPUT	변수속성, 변수이름	정적, 자동변수 인수전달
OUTPUT	변수이름	인수전달
FLIP FLOP	변수이름	정적변수
CIRCUIT	회로이름	수행레벨 + 1
BODY		수행시작
END		수행레벨 - 1 호출루틴으로
NOT	OPERAND	
AND	OPERANDS	
OR	OPERANDS	
XOR	OPERANDS	
PUSH	변수값(위치)	인수값의 전달
POP	변수값	함수값의 전달
CALL	함수호출	함수의 호출

Ⅳ. 시뮬레이터의 구성

시뮬레이터는 번역기에서 만들어 낸 중간 목적파일을 읽어 시뮬레이션 동작을 보여주는 것이다. 이 장에서는 세 부분으로 분류해서 설명하게 되는데 그것은 다음과 같다. 첫째는 입력 자료화일 부분으로 디지털 회로 기술파일에 설명된 최초의 입력을 다룬다. 둘째는 시뮬레이터 수행부분으로 입력된 자료를 토대로 중간 목적파일에 나타난 형태대로 수행하여 주는 부분이다. 세 째는 시뮬레이터 결과파일 부분으로 시뮬레이션 결과를 파일로 출력하는 방법을 설명하게 될 것이다.

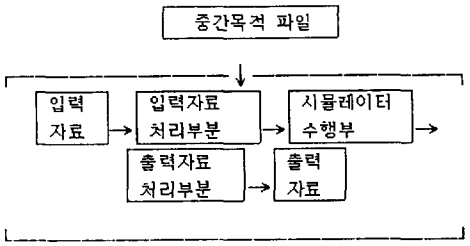


그림 12. 시뮬레이터의 구성
Fig. 12 CONFIGURATION OF SIMULATOR

4.1 입력 자료 부분

예를 들어 반가산기의 회로가 있다고 가정하고 다음과 같이 설명된다고 하자. X,Y는 반가산기의 입력이고 반가산기의 동작을 수행하기 위한 자료 X,Y의 초기 값은 그림 13과 같은 방법으로 주어질 수 있다.

```
module HALF_ADDER;
input  X,Y;
output S,C;

body
  S := xor(X,Y);
  C := and(X,Y);
end;
```

(a) 회로기술

```
Step 1 : X = 0 , Y = 0 ;
Step 2 : X = 0 , Y = 1 ;
Step 3 : X = 1 , Y = 0 ;
Step 4 : X = 1 , Y = 1 ;
```

(b) 입력자료화일
그림 13. 반가산기 입력자료
Fig. 13 INPUT DATA TO HALF ADDER

조합회로의 경우는 그림 13과 같은 형태의 입력자료로서 아무런 문제가 없다. 하지만 순차회로의 플립플롭인 경우 초기 값을 미리 지정해 줄 필요가 있다. 그림 14는 JK_FF과 입력 자료의 초기 값을 지정해 준 예이다.

```
MODULE  JK_FF;
INPUT   J,K,{ST};
        OUTPUT  Q;

BODY
  Q := RS_FF(and(K,ST),
```

```
and(J,not(ST),ST);
END;
```

(a) JK_FF 의 기술

```
INIT : ST=0;
STEP 1: J=0,K=0;
STEP 2: J=0,K=1;
STEP 3: J=1,K=0;
STEP 4: J=1,K=1;
```

(b)JK_FF의 초기입력자료
그림 14. JK_FF의 초기입력자료
Fig. 14 INPUT DATA TO JK_FF

4.2 시뮬레

JK_FF 의 입력자료 시뮬레이터중간 목적파일을 읽어 수행코드에 의해 각 동작을 수행하는 것으로 이는 인터프리터(interpreter)의 동작과 유사하다.

INPUT DATA TO JK 시뮬레이터 수의 호출에 중점을 두고 설명 할 것이다. 먼저 시뮬레이터 수행부는 그림 22와 같은 형태가 될 것이다. 각 테이블에 대한 설명은 편의상 1차원 테이블로 하겠지만 실제 구성은 포인터 자료를 이용하여 중간 목적파일의 크기에 유연하게 대처하도록 하였다.

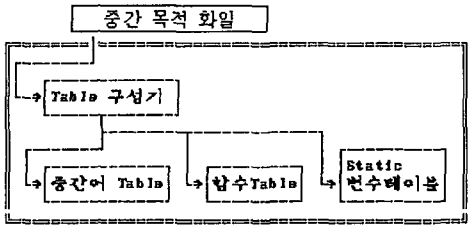


그림 15. 시뮬레이터 수행부
Fig.15 EXECUTION PARTS OF SIMULATOR

(1) 종간 목적파일 입력

시뮬레이터 일은 중간 목적파일을 중간어 테이블로 로드 하는 것이다. 이것은 단순한 테이블로 로드 뿐 만이 아니라 여러 가지 일을 하게 된다.

시뮬레이터 수행부 첫 번째로 함수테이블을 구성하는 것으로 'CIRCUIT'이라는 레코드를 만나면 함수테이블에 그 위치를 기억시킨다. 두 번째로 정적인 변수테이블의 구성으로 폴립플롭과 같은 자료구조는 모두 정적인 변수테이블에 기억공간이 할당된다. 그리고 모듈 블록의 INPUT과 OUTPUT 모두가 변수테이블에 위치하여야 그 값이 새로운 수행 시에도 남아있게 된다.[14]

이것은 광역적(global) 의미를 갖기도 하고 시뮬레이터의 수행자체가 입력 1 step에 디지털회로 기술파일의 모든 사항을 수행해야 하기 때문이다. 한번의 수행이 끝난 후 모든 내용을 잃어버린다면 수행 할 때마다 그 결과 값을 다시 입력시켜야 하고 또 그것이 이전 상태의 값으로써 의미를 갖는다면 재 입력도 무의미해지는 것이다.

(2) 입력 자료와의 인터페이스

시뮬레이터 수행부에서 중간 목적파일이 로드 된 후 해야 할 일은 입력 자료파일을 읽는 것이다. 이때 읽어들인 자료의 이름과 중간어 테이블에서 자료와 맞추기 위해서는 그림 16과 같이 변수 이름 테이블을 만들어야 한다.

변수이름 테이블에 기록될 것은 정적인 변수만이 기억될 것이며 그 외의 변수는 스택 자료를 이용하여 값만을 전달하게 될 것이다.

변수 이름	변수 값
A	0
B	0
C	0
:	:

그림 16. 변수 테이블
Fig. 16 VARIABLES TABLE

(3) 함수의 호출과 값의 전달

시뮬레이터 수행부에서 가장 처리하기 힘든 부분이 바로 함수의 호출과 함수 값의 전달이다. 함수의 인수전달 및 값의 되돌림은 스택 자료구조를 이용하는데 value parameter들은 값만 전달하기 때문에 스택에 값만 입력하면 된다. 하지만 정적 변수는 변수 테이블의 위치를 전달해야 한다. 변수 테이블의 위치 값의 전달은 2 이상의 수로 전달이 된다. 일반 인수는 값만 전달이 되는데 그 값이 0과 1 두 가지 뿐이다. 그래서 2 이상인 값은 변수 테이블의 위치를 가리키는 것으로 인식 계산하면 될 것이다. 호출과 값의 되돌림 과정을 나타내면 그림 17과 같이 나타내면 될 것이다.

원시프로그램 중간어코드 변수테이블 함수테이블

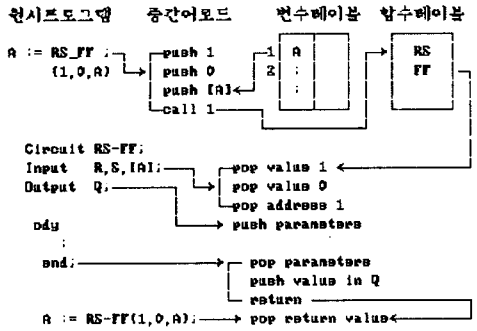


그림 17. 수행과정
Fig. 17 EXECUTING PROCESS

4.3 출력 자료부

출력 자료부는 각 단계마다 입력자료 파일을 자료로 하여 수행된 Output으로 선언된 변수들의 목록이다. 그림 25에는 시뮬레이터의 입력과 출력 예를 보여 준다.

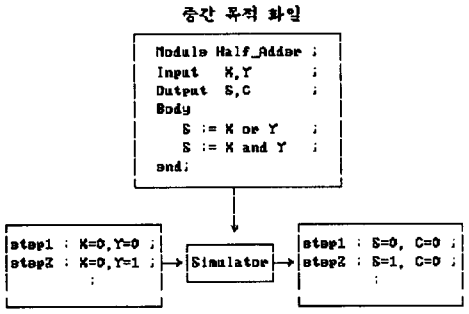


그림 18. 입력과 출력자료
Fig. 18 INPUT AND OUTPUT DATA

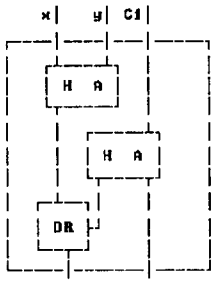
V. 실험 및 고찰

시뮬레이션 대상으로는 조합회로 시스템의 전가산기 회로와 순차회로 시스템의 3진 계수기를 사용하였다. 두 가지 실험 모두가 그림 19와 그림 20 같이 목적한 결과를 얻었으며 이외의 시스템에서도 목적한 결과를 얻을 것으로 예상된다.

```
module FULL_ADDER;
    input  X,Y,CI;
    output S,CO;

    circuit HALF_ADDER;
    input   X,Y;
    output  S,C;
    body
      S := xor(X,Y);
      Y := and(X,Y);
    end;
    body
      S := HALF_ADDER(CI,
           HALF_ADDER(X,Y).S).S;
      CO:= or(HALF_ADDER(X,Y).C
           ,HALF_ADDER(CI,
           HALF_ADDER(X,Y).S).C).C;
    end;
```

(a) 회로기술 파일내용



(b) 회로

```
step 1:x=0,y=0,ci=0;     step 1:S=0,CO=0;
step 2:x=0,y=0,ci=1;     step 2:S=1,CO=0;
step 3:x=0,y=1,ci=0;     step 3:S=1,CO=0;
step 4:x=0,y=1,ci=1;     step 4:S=0,CO=1;
step 5:x=1,y=0,ci=0;     step 5:S=1,CO=0;
step 6:x=1,y=0,ci=1;     step 6:S=0,CO=1;
step 7:x=1,y=1,ci=0;     step 7:S=0,CO=1;
step 8:x=1,y=1,ci=1;     step 8:S=1,CO=1;
```

(c) 입력자료화일 (d) 출력자료화일

X	Y	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(e) 진리표

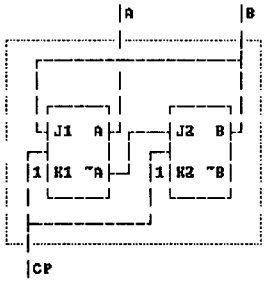
그림 19. 전가산기의 실험 및 결과
Fig. 19 THE RESULT OF FULL ADDER SIMULATION

```
module COUNTER_3;
input  CP;
output [A],[B];

circuit JK_FF;
input   CP,J,K,[Q];
output  Q;
body
  Q := rs_ff(and(CP,and(J,not(Q))),
  and(CP,and(K,Q)));
end;
        ]

body
A := JK_FF(CP,B,1);
B := JK_FF(CP,not(A),1);
end;
```

(a) 회로기술화일



(b) 회로

```
init  :A=0,B=0;
step 1:CP=1;
step 2:CP=1;
step 3:CP=1;
step 4:CP=0;
step 5:CP=1;
```

(c) 입력자료화일

```
init  :A=0,B=0;
step 1:A=0,B=1;
step 2:A=1,B=0;
step 3:A=0,B=0;
step 4:A=0,B=0;
step 5:A=0,B=1;
```

(d) 출력자료화일

그림 20. 3진 계수기의 실험 및 결과
Fig. 20 THE RESULT OF MODULO-3 COUNTER SIMULATION

모의실험용 입력자료는 전가산기의 경우 모든 경우의 입력을 기술하였고 3진 계수기의 경우 '0' 과 '1'의 임의적 나열로 기술하였다.

이 실험에서 조합회로의 경우는 입력자료의 순서에 상관없이 똑같은 결과를 얻을 수 있으며 순차회로의 경우 초기상태의 값만이 출력자료의 순서에 영향을 미친다. 시뮬레이터의 모의실험 결과 모두 원하던 결과를 얻었다. 수식표현의 까다로움과 함수의 개선사항 여지를 제외하곤 모두 만족스러운 결과를 얻었다.

VI. 결론

연구목표인 디지털 시스템 설계자동화용 도구를 작성하여 최종 실험결과 값으로 5장에서 기술한 바와 같이 얻었다. 번역기를 구성하기 위하여 Unix 시스템의 YACC를 이용하여 파싱 테이블을 작성하였으며 번역기의 원시

프로그램으로 파스칼을 사용하였다.

또한 디지털 시스템을 기술하는데 있어서 고급언어의 유형을 사용함으로써 사용자와의 친밀도를 높였고 인터프리터 형식의 시뮬레이터도 번역기와 시뮬레이터의 이론을 바탕으로 작성 할 수 있을 것으로 기대된다.

끝으로 중간 목적파일을 데이터베이스(data base) 화하는 라이브러리(library) 도구와 링커를 개발하면 디지털 시스템의 설계 및 시뮬레이션에 많은 편리성과 효율성을 제공하게 될 것이다.

참 고 문 헌

[1] Ralston, "ENCYCLOPEDIA OF COMPUTER SCIENCE AND ENGINEERING" V.N.R, PP674-679, 1983.

[2] Milos D.ERcegovac Tomas Lang, "DIGITAL SYSTEM AND HARDWARE/FIRMWARE ALGORITHMS" Wiley, PP1-8,19-22, 1985.

[3] North-Holland, "COMPUTER HARDWARE DESCRIPTION LANGUAGE AND THIER APPLICATIONS" PP309-310, 1981.

[4] John P. Hayes, "DIGITAL SYSTEM DESIGN AND MICRO-PROCESSOR" McGRAW-HILL, PP265-323, 1981.

[5] Robin Hunter, "THE DESIGN AND CONSTRUCTION OF COMPILERS" BRITISH LIBRARY, PP18-41, 1981.

[6] Melvin A. Breuer, "DIGITAL SYSTEM DESIGN AUTOMATION : LANGUAGE, SIMULATION & DATA BASE" PITMAN, PP131-136, 1975.

[7] FREDRICK J. HILL GERALD R. PETERSON, "DIGITAL SYSTEMS" LIBRARY OF CONGRESS, PP117-140, 1973.

[8] ALFRED V. AHO, RAVI SETHI, JEFFREY D. ULLMAN, "COMPILERS" ADDISON WESLEY, PP308-316, 1986.

[9] ROLAND C. BACKHOUSE, "SYNTAX OF PROGRAMMING LANGUAGES" C.A.R HOARE, PP158-172, 1988.

[10] 이성우, 김태중, 김재일, "디지틀 공학" 상학당, PP170-178, 1985.

[11] 김기태, 박우전, "컴파일러" 홍릉과학, PP136-148, 1983.

[12] 김영택, "컴파일러 구성론" 희중당, PP112-117, 1985.

[13] 김영택, 유원희, 도관종, "컴파일러 구성" 한국방송통신대학,PP71-91,137-149, 1987.

[14] 일신기술 서적 편집실, "디지틀 IC 실용회로 사전"일신, PP96-122,1979.

[15] 삼성반도체,"KONIX V SUPPORT TOOLS GUID", PP12.1-12.61, 1987.

저 자 소 개

권 승 학

1993년 청주대학교 전자계산학과 (공학석사)

1996년 청주대학교 전자공학과 과정박사수료

1988년 - 1991년 포항종합제철 (주)

1995년 - 현재 대원과학대학 전산정보처리과 조교수

〈관심분야〉 암호이론 , 디지털 시스템

이 명 호

1981년 연세대학교 전자공학과 (공학석사)

1991년 연세대학교 전자공학과 (공학박사)

1984년 - 현재 청주대학교 전자공학과 교수

〈관심분야〉 Information Technology (IT), Communications