韓國 OA學會 論文誌
第3卷 第2號, 1998. 6.

98-3-2-1-3

디지털 시스템의 데이터 및 상태표 생성기 구현

조 성 국* 이 명 호**

An Implementation of Data and State Table Generator for Digital System

Sung-Kuk Cho* Myung-Ho Lee**

요 약

디지털 시스템은 제어부 시스템(control subsystem)과 데이터 부시스템(data subsystem)으로 구성된다. 본 논문에서는 하드웨어 기술언어(hardware description language)와 그에 대한 하드웨어 컴파일러(hardware compiler)를 정의한 후 레지스터 전송 알고리즘(register transfer algorithm)을 이용하여 제어부의 상태표와 데이터부의 데이터 표를 생성하는 하드웨어 설계용 번역기(translator)를 구현하였다. 본 연구에서 사용한 개발 툴은 Unix의 C, Lex, YACC이다.

Abstract

The digital system is consisted of control subsystem and data subsystem. On this Thesis, after defining the hardware description languages and hardware compiler based on this, We have designed the tools which created data and state table using of register transfer algorithm.

As a major language selected C and then as subtools, developed all these making use of Lex and YACC of Unix.

---

* 원주전문대학 전산정보처리과 조교수
** 청주대학교 정보통신공학과 교수

논문접수 : 98.4.10 심사완료 : 98.6.12

1. 서 론

디지털 시스템은 데이터 부시스템(data subsystem)과 제어부시스템(control subsystem)으로 구성된다. 이들 부시스템들은 외부 입력신호인 데이터 입력과 제어 입력으로 순차적인 제어신호와 상태신호를 교환함으로써 원하는 데이터 출력을 얻는다.

시스템을 설계하려면 먼저 설계 요구조건을 분석하여 최적의 실현법을 결정하고 설계를 시작하는 것이 일반적이다. 이와 같은 설계에 있어서 가장 중요한 부분은 설계목적의 확정과 그 목적을 달성하기 위한 알고리즘과 설계순서의 결정에 있다고 할 수 있다.

기존의 순차시스템(synchronous sequential system)의 설계는 표(table), 다이어그램(diagram) 그리고 하드웨어 모듈의 회로망으로 쉽게 바뀌어지는 표현식들(expression)을 사용하여 상태함수(state function)들로 기술하여 왔다. 이런 방법은 입력과 출력 그리고 상태(states)들의 수가 적은 간단한 시스템에서 유용하였다. 그러나 입력, 출력, 상태의 수가 많고 표현식들이 쉽게 구해지지 않는 시스템들은 상태함수를 쉽게 얻을 수 없으므로 이들 시스템을 기술하고 구현하기가 어려워짐에 따라 알고리즘 개념이 도입되었다. 이러한 알고리즘에 의한 방법에는 구현할 시스템의 구조에 따라 사용될 여러 특정 세부연산들의 종류와 세부 연산들의 동시성(concurrency)에 따라 많은 등가 알고리즘들이 존재할 수 있으며, 또한 각각의 알고리즘에서는 여러 레벨(level)이 존재할 수 있으므로, 시스템을 구현하기 위한 알고리즘은 설계의 목적과 요구사항들인 속도, 가격, 신뢰성, 크기, 전력소모 등을 만족시킬 수 있는 모든 가능한 사항들을 고려하여 선택된다.

이러한 방법으로 인해 시스템 전체를 하나로 설명하는 것이 어려운 경우 시스템을 각기 다른 기능을 수행하는 부시스템으로 나누어 각각 기술하는 모듈 접근법(module approach)이 가능하게 되었다.

본 논문에서 알고리즘을 기술하기 위해 사용한 언어는 하드웨어 기술언어(hardware description language)이며, 이러한 하드웨어 기술언어에 관한 연구는 J. Von Neumann이 컴퓨터 구조 기술을 위해 처음 시도된 이후 많은 논문들이 발표되었다.

시스템이 복잡해지고 기능이 다양해짐에 따라 설계시간의 단축, 설계 신뢰도의 향상 및 설계 비용의 절감을 위해 고안된 것이 컴퓨터를 이용한 설계자동화(design automation) 방법이다.

본 논문에서는 디지털 시스템의 제어부 및 데이터부 설계를 자동화하기 위한 방법으로 고수준 알고리즘 기술문(high level algorithm description language)으로부터 제어부의 상태표 및 데이터부 표를 얻고자 함이 목적이다.

2. 디지털 시스템의 구성

디지털 시스템은 데이터 부시스템(data subsystem)과 시스템의 제어를 담당하는 제어 부시스템(control subsystem)으로 구성되는바 데이터 부시스템과 제어 부시스템은 상태조건(state condition)과 제어신호(control signals)에 의해 주어진 동작순서에 따라 시스템이 동작된다.

2.1 제어 부시스템

제어 부시스템은 데이터 프로세서가 연속적으로 동작할 수 있도록 필요한 신호들을 순차적으로 발생시키는 기능을 담당하는 곳으로 다음과 같은 7가지 구현 접근법(implementation approach)이 가능하다.

① Register+gates
② Register+Multiplexers
③ Register+ROM or PLA
④ Programmable Sequential Array(PSA)
⑤ Microprogrammed Controller
⑥ Microprogrammable Controller
⑦ Microprocessor as Controller

이들 구현 접근법중 ①과 ② 접근법은 Hardwired 접근법으로 추후 회로의 변경이나 구성소자의 변경을 고려하지 않은 방식이며, 나머지 5가지 접근법은

Microprogrammed 접근법으로 알고리즘 변경에 따른 제어기능 변경이 매우 용이하다. 제어 부시스템의 구조는 중앙 제어(centralized control)방식, 분산제어(decentralized control)방식, 반중앙제어(semicentralized control)방식의 구조가 이용된다.

2.2 데이터 부시스템

데이터 부시스템은 데이터의 저장(storage)과 이동(move) 및 변형(transformation) 등이 이루어지는 장소로서 기억소자(storage component), 데이터 경로(data path), 연산자(operator) 등으로 구성되어 있으며, 데이터 경로는 시스템에서 요소들간의 연결이며 데이터가 이동하는 경로이다.

데이터 경로는 데이터의 각 비트를 한 비트씩 보내는 직렬(Serial) 방식과 각 비트를 동시에 보내는 병렬(parallel) 방식으로 나뉘고, 데이터를 보내는 방향에 따라 단방향성(unidirectional) 방식과 양방향성(bidirectional) 방식으로 나누어진다. 또한 데이터 경로의 사용방법에 따라 전용경로(dedicated path) 방식과 공유경로(shared path) 방식으로 나누어진다.

2.3 데이터 부시스템의 구성요소

(1) 저장 장소(storage components)

기본적인 저장장소는 레지스터이며 레지스터의 이름은 출력벡터(output vector)의 이름을 따라 붙인다. 기본적인 동작은 레지스터 안에 입력벡터(input vector)를 넣는 로드(load) 동작이며, 초기화 동작은 클리어(clear)이다. 저장 기능(storage function)은 개개의 레지스터나 레지스터 배열을 사용하여 구현된다. 개개의 레지스터 사용은 시스템 동작에서 더 많은 동시성을 주는 반면 내부 연결수가 많아진다.

(2) 연산자(operator)

연산자들은 비트 벡터들의 변환을 수행하며, 입력벡터(input vector), 출력벡터(output vector) 그리고 연산자에 의해 수행되는 함수에 의해 이름을 붙인다.

(3) 데이터 경로(data path)

데이터 경로는 데이터의 이동통로를 말하며, 한번에 보내는 전송량(비트 수)에 따라 직렬 데이터 전송(serial data transmission)과 병렬 데이터 전송(parallel data transmission)이 있고 데이터 전송 방향에 따라 단방향 경로와 양방향 경로로 구분된다.

(4) 제어점(control pointer)

제어 부시스템으로부터 보내오는 제어신호를 받아들이는 데이터 부시스템의 입력점들을 제어점이라 하고, 이 제어점은 동작이나 데이터 경로의 선택 그리고 레지스터 로딩(loading)을 제어하기 위해 사용된다.

(5) 조건(condition)

조건들은 데이터 부시스템에서 나오는 신호로 조건적인 할당(conditional assignment)이나 선택적인 순서(alternative sequence)에 관계하는 제어신호를 결정하기 위해 제어 부시스템에서 사용된다. 언어에서는 이 조건들이 조건적인 할당이나 조건적인 분기(conditional branch)로 표시된다.

3. 하드웨어 기술언어

하드웨어 설계자들은 하드웨어 동작이나 구조를 기술하기 위해 어떤 형태의 언어를 사용하는데, 이러한 언어를 하드웨어 기술언어라고 한다. 하드웨어 기술언어는 레지스터를 표시하거나 레지스터 내용에 대한 동작 제어기능을 설명하는 기호이며, 디지털 시스템의 구조와 동작을 기술하기 위해 사용되는 언어로서 주로 학문적인 연구를 목적으로 설계되었다. 이들 기술언어는 하드웨어의 모든 동작 특성을 쉽게 기술할 수 있고, 구문이 간단하며, 이해 및 수정이 용이한 장점을 지닌다. 또한 형태적으로는 프로그래밍 언어와 유사한 프로그래밍 언어의 변형된 형태이다. 이러한 하드웨어 기술언어의 대표적인 것에는 APL, AHPL, CDL, DDL, ISP 등을 들 수 있다.

3.1 하드웨어 기술언어의 요소

하드웨어 기술언어는 다음과 같은 3가지 측면에서 분석할 수 있다.

첫째, 기능적인 측면으로 시스템의 기술목적은 수행하고자 하는 기능을 효과적으로 표현하는데 있으므로, 기술문은 가능한 많은 내용을 포괄적으로 함유함과 동시에 시스템을 다양하게 실현 할 수 있도록 기술되어야만 한다.

둘째, 구조적인 측면으로 시스템의 구현을 위하여 기술문은 가능한 한 구체적으로 구조적인 정보를 포함할 수 있어야 한다.

세 번째는 설계과정을 지원하는 측면으로써, 설계과정은 여러 번의 수정이 가해져야 하는 반복적인 과정이며 설계목적이 여러 가지의 제반 사항들을 만족시키는 적절한 시스템을 얻는데 있으므로, 반복수정하는 과정에 있어서 수정되는 세세한 부분들을 지원할 수 있어야 한다.

3.2 기술언어의 구성

(1) 설명문

설명문은 알고리즘을 설명하기 위해 프로그램 어디서나 자유롭게 사용되며 "/*"과 "*/" 사이에 기술된다.

(2) DATA OBJECT

디지털 시스템에서의 데이터는 비트 단위로 구성되고 저장장소인 레지스터에 저장되며 데이터 경로를 따라 전송된다. 기본적인 형태는 비트 벡터(Bit Vector)이며, 비트 벡터의 차원(Dimension)은 심볼 "<"와 ">" 사이에 비트의 수를 기술하고 표기는 다음과 같다.

A<5> := (A4, A3, A2, A1, A0)
X<3> := (X2, X1, X0)

여기서 심볼 ":="는 비트 벡터의 선언(declare) 또는 재명명(rename)시 사용된다. 저수준 언어(low level language)에서 받아들이는 기본적인 대상물(object)의 구조는 배열이고, 이들의 차원은 비트 벡터의 수를 표시하는 정수의 쌍과 벡터에 대한 비트들의 수로 표기한다. 여기서 비트들의 수는 다차원 다출력 조합망의 배열에 따른다. 예로서,

배열 M<2,6>은 2 × 6 MATRIX이며,

M := M[0] := (M5[0], M4[0], M3[0], M2[0], M1[0], M0[0])
M[1] := (M5[1], M4[1], M3[1], M2[1], M1[1], M0[1])

혹은 M[0] := (M[0,5], ……, M[0,0])
M[1] := (M[1,5], ……, M[1,0])

로 표현할 수 있다.

(3) Function(operators)

실행레벨에서 함수는 비트 벡터를 가지며, 기술할 연산자는 대문자열의 이름에 의해서 표시되고 인수(argument)는 이름 뒤의 삽입구 안에 기술되고 연산자는 조합 회로망(combinational network)에 의해 구현된다.

조건적인 함수일 경우 조건에 의해 적어도 한 대상이 선택된다. A if a | B if b | C if c 인 조건적인 함수에서 A, B, C는 대상이고 a, b, c는 조건이 된다. 연결(concatenation)함수는 큰 벡터를 형성하기 위해 벡터를 연결하며 표기는 다음과 같다.

a:=(a5,a4,……, a0)
b:=(b3,b2,……, b0) 일 때
c:=(a, b)라고 하면, c는 연결함수가 되며 a+b의 벡터 즉, c:=(a5,a4,……,a0, b3,b2,……,b0)

벡터에서 부벡터 추출(extraction)은 요소의 연결로 표기하며 다음과 같다.

A:=(B4, B3, B2)

(4) 대입문(assignment)

대입문은 원시대상물(source object)을 목적대상물(destination object)에 대입할 때 사용된다. 대입문에 대한 언어구성은 D←S와 같이 쓰며, 레지스터 전송에 의해 실행된다. 대입문은 반복문의 구조를 도입함으로써 명령문을 간략하게 표현 할 수 있다.

if(제어조건) then {명령문} else {명령문}문으로 된 제어조건문은 종전의 형태로 바꿀 수 있는데 즉,

T : if (c=0) then (F←1) else (F←0)을 바꾸면
c'T : F←1
c T : F←0 와 같이 된다.

4. 하드웨어 컴파일러의 구현 및 고찰

4.1 알고리즘 기술문의 규칙

알고리즘의 기술문은 표제부(header part), 선언부(declaration part), 몸체부(body part)로 구성된다. 표제부는 알고리즘의 머리부분으로써 식별자 "HDL" 뒤에 알고리즘의 명칭을 나타내고 다음과 같이 표기한다.

HDL Otherid;

선언부는 입력변수, 내부변수와 출력변수의 형을 정의한 부분으로서 입력변수는 "input"로 출력변수는 "outputs"로 내부변수는 "works"로 했고 변수들의 나열은 ","로 구분하였으며, 다음과 같이 표기한다.

```
input  a,b,c  :interger<16>
       s      :boolean;
output c      :interger<16>;
works xx,yy :integer<16>;
```

몸체부는 각 명령을 기술한 집합으로서 "begin"으로 시작하여 "end."로 끝나며 전체 블록을 형성한다. 몸체부의 명령문들은 동시동작(simultaneous operation)에 대한 명령과 순차동작(sequential operation)에 대한 명령으로 구분할 수 있는데, 동시동작에 대한 명령문들 사이는 ","로서 구분하고 순차동작에 대한 명령문들 사이는 ";"으로 구분한다. 몸체부 내부에는 또 다른 명령문의 "begin"과 "end;" 부블럭(sub-block)을 포함할 수 있다.

식별자(identifier)는 모듈(module)의 이름으로서 영문자나 숫자의 조합으로 이루어지며, 첫 문자는 반드시 영문자를 사용한다. 단, 예약에는 변수로 사용 불가능하다.

명령문(statements)은 디지털 시스템의 실제 동작을 기술한 것으로서 고수준 언어(high level language)인 반복문을 포함할 수 있고, 시스템에서의 동시동작도 포함할 수 있다.

대입문(assignment)은 디지털 시스템의 동작을 나타내는 명령문으로서 "="로서 나타내었다. 설명문(comments)은 알고리즘에 대한 이해를 돕기 위해 사용하며, "/*"와 "/*" 사이에 기술하고 생략 할 수도 있다. 제어문(control statements)은 조건의 특성에 따라 사용하기 위해 while문, repeat until문, if문 및 goto문을 고려하였으며, 다음과 같이 표기한다.

```
While 조건 명령문;
repeat 명령문; until 조건;
if 조건 명령문;
goto 라벨;
```

연산자(operator)로는 산술연산자와 관계연산자를 사용하며, 산술연산자에는 +, -, *, /, ||, (,) 등을 사용하고 논리연산자에는 >, <, =, LE(작거나 같다), GE(크거나 같다) 등을 사용한다.

4.2 알고리즘 기술문의 문법

알고리즘 기술문의 문법 BNF는 그림 1과 같다.

```
<program> : <program heading> <select> <block1>
<program heading> : HDL Otherid;
<block1>    : <declare1><block2>
<select>    :
            | CON
            | SEQ
<declare1>  :
            | <declare11> ';'
<declare11> : <declare11> ';' <declare12>
            | <declare12>
<declare12> : INPUTS <declare2>
            | OUTPUTS <declare2>
            | WORKS <declare2>
<declare2>  : <declare2> ',' <declare3>
            | <declare3>
<declare3>  : <declare4> ':' <declare5>
<declare4>  : <declare4> ',' Otherid
            | Otherid
<declare5>  : INTEGER '<' NUM '>'
```

```
          | BOOLEAN
<block2>  : BEGIN <block21> END '.'
<block21> :
          | <block22> ';'
<block22> : <block22> ';' <block3>
          | <block3>
<block3>  : <block33>
          | <LABEL <block33>
<block33> : <block33> ',' <block4>
          | GOTO Otherid ';'
          | <statement>
<block5>  : <block4>
          | <block4>
<block4>  : WHILE <expression> <block5>
          | IF <expression> <block5>
          | REPEAT <block5> UNTIL
            <expression> ';'
          | BEGIN <block22> END
          | <statement>
<statement>  : Otherid '=' <expression>
<expression> : '(' <expression> ')'
             | '|' <expression> '|'
             | <expression> '<' <expression>
             | <expression> '>' <expression>
             | <expression> '=' <expression>
             | <expression> LE <expression>
             | <expression> GE <expression>
             | <expression> '+' <expression>
             | <expression> '-' <expression>
             | <expression> '*' <expression>
             | <expression> '/' <expression>
             | Otherid
             | NUM
```

그림 1. 알고리즘 기술문의 문법

Fig. 1 Rules of Algorithmic description language

4.3 제어 상태표 및 데이터 테이블 생성기의 설계

생성기는 어휘를 해석하는 Lex와 구문분석기 YACC에 입력되는 알고리즘 문법 원시프로그램과 YACC에서 출력된 것을 해석하는 C 프로그램으로 설계하였으며, 생성기의 흐름도는 그림 2와 같다.

RTL 문법은 YACC 규칙으로 정의하고 컴파일하면 y.tab.c라는 목적코드인 C 프로그램으로 생성되고 이 프로그램의 Call은 yyparse()라는 함수를 사용한다.

Lexical 분석은 LEX라는 툴(tool)을 사용해서 정의된 것을 번역하면 lex.tt.c라는 목적코드가 생성된다.

주 프로그램(main program)은 동적변수의 기억영역 확보 및 변수를 정의하고 각각의 필요한 함수들은 테이블 관리라는 프로그램 속에 정의해 놓았다.

즉, 전체 프로그램은 main.c, y.tab.c, lex.yy.c, semantics.c라는 프로그램이다. 여기서, semantics.c라는 프로그램은 테이블 관리 즉, 출력 테이블인 데이터 및 제어에 관한 자료와 중간 프로그램을 얻기 위한 스택(stack)의 관리를 담당하고, 테이블의 출력 및 스택의 출력도 담당한다. 몇몇의 부분에서는 최적화의 기법도 사용하였다.

4.4 제어상태표 및 테이블 생성기의 실행결과

생성기의 실행결과를 출력하기 위해 다음과 같은 Newton Rapson iterative algorithm을 원시 프로그램으로 사용해서 내부연산 즉, 복합문에서 동시동작의 방법 및 순차 동작의 방법을 보였으며, 제어 상태표 및 데이터 테이블 생성기의 실행 결과는 다음

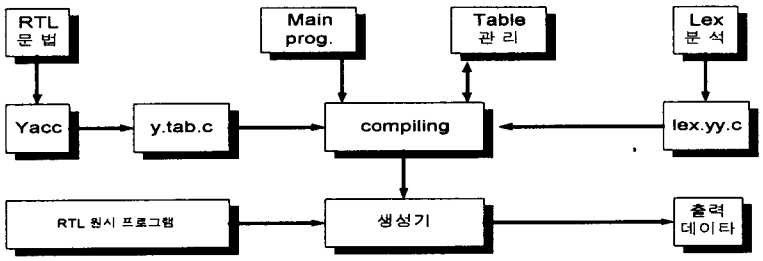

그림 2 생성기의 흐름도

(Flowchart of Translator)

과 같다.

```
HDL    reciprocal;
      {r}
       inputs   arg, err  : integer<16>,
                start     : boolean;
       outputs  rec       : integer<16>,
                done      : boolean;
       works    bb        : integer<16>;
begin
     wait     : done=1, if (start') goto wait;
     abc      : bb=bb+1;
     compute  : rec=1, done=0;
         while ( ¦ arg* rec-1¦ < err/2)
         begin
             rec=(2-arg*rec)*rec;
             end;
         goto wait;
end.
```

그림 3. 내부연산 동시동작의 원시 프로그램

Fig. 3 Source program of internal computative simultaneous operation

```
   HDL     reciprocal;
concurrent processing.
wait:      done=1,
     if    (start')     goto wait;
abc:       temp7=bb+1,
     bb=temp7;
compute:    rec=1,
     done=0;
LOOP0:   temp6=arg*rec,
     temp6=temp6-1,
     temp6=ABS(temp6),
     temp5=err/2,
     temp7=temp6 < temp5,
       if not(temp7) goto EXIT0;
     temp3=arg*rec,
     temp5=2-temp3,
     temp6=temp5*rec,
     rec=temp6;
        goto   LOOP0;
EXIT0: ;
       goto wait;
```

그림 4. 동시동작의 중간출력

Fig. 4 Intermediate output of simultaneous operation

State	Destination	Operator	Sourcel	Source2	Sequence
wait	done	←	1		;
abc:	bb	ADD	bb	1	;
compute	rec	←	1		,
stat 3	done	←	0		;
	temp6	MUL	arg	rec	,
	temp6	SUB	temp6	1	,
	temp6	ABS	temp6		,
	temp5	DIV	err	2	,
	temp7	LT	temp6	temp5	,
	temp3	MUL	arg	rec	,
	temp5	SUB	2	temp3	,
stat 11:	rec	MUL	temp5	rec	;

그림 5. 동시동작의 데이터 테이블

Fig. 5 Data table of simultaneous operation

```
State   sequence
        wait :
        if start'
        →wait
        abc :
        compute:
        LOOP00
        if not temp7 → EXIT00
        →LOOP00
        EXIT00
        →wait
```

그림 6. 동시동작의 상태표

Fig. 6 State table of simultaneous operation

```
HDL    reciprocal;
       {s}
       inputs arg, err : integer<16>,
              start    : boolean;
        outputs rec     : integer<16>,
                done    : boolean;
        works  bb       : integer<16>;
begin
     wait:  done=1, if (start') goto wait;
     abc:    bb=bb+1;
     compute: rec=1, done=0;
         while(¦arg*rec-1¦ < err/2)
         begin
            rec=(2-arg*rec)*rec;
            end;
         goto wait;
end.
```

그림 7. 순차동작의 원시 프로그램

Fig. 7 Source program of sequential operation

```
HDL     reciprocal;
sequential processing.
wait:    done=1,
      if   (start')  goto wait;
abc:       temp7=bb+1;
      bb=temp7;
compute:   rec=1,
      done=0;
LOOP0:     temp6=arg*rec;
      temp6=temp6-1;
      temp6=ABS(temp6);
      temp5=err/2;
      temp7=temp6<temp5;
           if not (temp7)   goto EXIT0;
      temp3=arj*rec;
      temp5=2-temp3;
      temp6=temp5*rec;
      rec=temp6;
           goto LOOP0;
EXIT0: ;
           goto wait;
```

그림 8. 순차동작의 중간출력

Fig. 8 Intermediate output of sequential operation

State	Destination	Operator	Source1	Source2	Sequence
wait	done	←	1		;
abc:	bb	ADD	bb	1	;
compute	rec	←	1		,
stat 3:	done	←	0		;
stat 4:	temp6	MUL	arg	rec	;
stat 5:	temp6	SUB	temp6	1	;
stat 6:	temp6	ABS	temp6		;
stat 7:	temp5	DIV	err	2	;
stat 8:	temp7	LT	temp6	temp5	;
stat 9:	temp3	MUL	arg	rec	;
stat 10:	temp5	SUB	2	temp3	;
stat 11:	rec	MUL	temp5	rec	;

그림 9. 순차동작의 데이터 테이블

Fig. 9 Data table of sequential operation

```
State   sequence
        wait:
        if start'
        →wait
        abc:
temp7   Temp04
        compute:
        Loop00
temp6   Temp07
temp6   Temp08
        Temp09
temp5   Temp10
temp7   Temp11
          if not temp7→exit00
temp3   Temp13
temp5   Temp14
temp6   Temp15
         → Loop00
        Exit00
         → wait
```

그림 10. 순차동작의 상태표

Fig. 10 State table of sequential operation

5. 결론

본 연구에서는 알고리즘으로 작성된 기술문을 입력하여 제어 부시스템의 상태표와 테이타 부시스템에 대한 데이터 표를 얻었다. 그 결과 디지털 시스템을 설계하는데 이를 이용함으로써 시스템을 설계할 수 있는 기본자료를 얻을 수 있음을 알 수 있었고 또한 본 논문의 결과를 이용하면 설계자동화의 일부분으로서 활용할 수 있는 방법임을 알 수 있었다. 본 논문에서 설계한 번역기를 이용함으로서 시스템의 설계가 간편하고 정확한 회로를 설계하는데 도움이 될 수 있겠다.

References

[1] A. K Singth, "Descriptive Technique for Digital System Containing Complex Hardware Component", Ph.D Dissertation, Kansas State UNIV., 1981.

[2] Alice C. Paker, "Automated Synthesis of Digital System", IEEE D & T, pp.75-81, 1984.

[3] Axel T. Schreiner, "Introduction to Compiler Construction With Unix", Prentice-Hall, 1985.

[4] John P. Hayes, "Digital System Design and Micro-Processors", Mcgraw-Hill, pp.301-323, 1984.
[5] Michell Waite, "Stephan Prata and Donald Martin, "C Primer Plus", Howard W. Sams & Co., 1985.
[6] Milos D. Ercegovac and Tomas Lang, "Digital System and Hardware/Firmware Algorithms", John Wiley & Sons, pp.159-169, pp.393-439, pp.469-596, 1985.
[7] M. Moris Mano. "Computer System Architecture", Prentice Hall, pp.39-68, pp.101-128, pp.290-322, 1982.
[8] M. Morris Mano, "Digital Logic and Computer Design", Prentice Hall, Ch.4,5,6,7,8, 1979.
[9] M. R. Barbacci, "A Comparison of Register Transfer Languages for Describing Computers and Digital System", IEEE Trans. Comput. Vol. c-24, pp.137-150, 1975.
[10] P. G. Jpaulin, J. P. Knight, E. F. Girczyc. "A Multi-Paradigm Approach to Automatic Data Synthesis", IEEE, 1986.
[11] Roland C. Backhouse, "Syntax of Programming Languages", C. A. R. Hoare, pp.158-172, 1978.
[12] Sajjan G. Shiva, "Computer Design and Architecture", Little Brown and Company, 1985.
[13] Subrata Dasgupta, "The Design and Description of Computer Architectures", John Wiley & Sons, pp. 40-61,
1984.
[14] 김기태, 박우전 저, "컴파일러", 홍릉과학출판사, pp.83-111, 1985.
[15] 김승연, 최병갑 공역, "Compilers", 성안당, pp.175-306, 1988.
[16] 김영택 저 "컴파일러 구성론", 희중당, 1987.

● 저자소개

조 성 국
원주전문대학 전산정보처리과 조교수
E-mail : skcho@tearoom.chongju.ac.kr
연구분야 : 시스템 프로그래밍, 컴파일러

이 명 호
청주대학교 정보통신공학과 교수
E-mail : himmel@alpha94.chongju.ac.kr
연구분야 : 데이터 통신, 운영체제, 자동설계