論文 86-23-3-6

VLSI의 논리설계 자동화를 위한 SDL 하드웨어 컴파일러

(A SDL Hardware Compiler for VLSI Logic Design Automation)

趙 仲 彙*, 鄭 正 和*

(Joung Hwee Cho and Jong Wha Chong)

要 約

본 논문에서는 논리 설계 자동화를 위해 하드웨어 기술 언어인 SDL(Symbolic Description Language)에 대한 하드웨어 컴파일러(hardware compiler)를 제안한다.

디지탈 시스템의 동작 특성을 register-transfer 레벨에서 기술한 SDL에 대하여 어휘 분석을 행하여 시스템의 제어부와 데이타 전송부에 대한 데이타를 형성하는 알고리즘(알고리즘 Ⅰ)을 제안한다. 다음에, D 플립 플롭과 2-입력 AND와 OR 게이트 및 NOT 게이트를 회로소자로 사용하여 알고리즘 Ⅰ에서 형성된 데이타에 대하여 구문 분석을 행하여 게이트 레벨의 논리 회로도와 등가인 NDL(Network Description Language) 표현식을 얻는 알고리즘(알고리즘 Ⅱ)을 제안한다.

또한, SDL 컴파일러를 VAX-11/750(UNIX)의 C언어로 프로그램하여 논리 설계 예에 적용함으로써 본 논문에서 제안하는 SDL 하드웨어 컴파일러의 효용성을 보인다.

Abstract

In this paper, a hardware compiler for symbolic description language (SDL) is proposed for logic design automation. Lexical analysis is performed for SDL which describes the behavioral characteristics of a digital system at the register transfer level by the proposed algorithm I. The algorithm I is proposed to get the expressions for the control unit and for the data transfer unit. In order to obtain the network description language (NDL) expressions equivalent to gate-level logic circuits, another algorithm, the algorithm II, is proposed. Syntax analysis for the data formed by the algorithm I is also Performed using circuit elements such as D Flip-Flop, 2-input AND, OR, and NOT gates. This SDL hardware compiler is implemented in the programming language C (VAX-11/750(UNIX)), and its efficiency is shown by experiments with logic design examples.

Ⅰ. 序 論

디지탈 시스템의 설계 요구 조건을 하드웨어 기술 언어(HDL :Hardware Description Language)로 기술하여 논리 회로도를 컴퓨터에 의하여 얻는 논리 설계 자동화에 대한 연구가 크게 요구되고 있다.[1]

종래에는 설계 요구 조건을 분석하여 상태도표(state diagram), 진리치표 또는 부울함수로 표시한 후 설계에 사용되는 회로 소자에 따라 게이트 레벨의 논리 회로도를 얻는 설계 방법이 많이 사용되었다. 이와같은 설계 방법은 디지탈 시스템이 대규모화 됨에

*正會員, 漢陽大學校 電子工學科
(Dept. of Elec. Eng., Hanyang Univ.)

接受日字 : 1986年 2月 1日
(※ 本 研究는 韓國學術振興財團의 支援으로 이루어진 것임.)

따라 설계 시간이 매우 길고 설계의 정확성을 보장할 수 없다는 단점이 지적되고 있다.[1-4]

최근 이러한 문제점을 해결하기 위해 디지탈 시스템의 동작 또는 구조 특성을 분석하여 각 레벨[2]에 따라 하드웨어 기술 언어로 기술한 후 레이아웃(layout) 의 입력인 게이트 레벨의 논리 회로도를 컴퓨터에 의해 설계하는 논리 설계 자동화에 대한 연구 결과가 많이 발표되고 있다. 특히, 디지탈 시스템의 동작 특성을 register-transfer 레벨에서 기술하여 gate 레벨의 논리 회로도를 얻는 연구가 많이 진행되고 있으며, AHPL[3,4], DDL[6,7,17] 및 CDL[2] 등의 하드웨어 기술 언어와 이를 입력으로 하여 게이트 레벨의 논리 회로도를 얻는 하드웨어 컴파일러가 발표되고 있다.[7,8,11,12,17] 이와같은 하드웨어 기술 언어는 설계 요구 조건을 표시할 수 있는 흐름도로부터 직접 기술하기가 어려우며, 이미 기술한 것에 대한 이해가 어려워 수정이 매우 어렵다는 단점이 지적되고 있다.[9,18] 따라서, 조중휘 등의 연구[18]에서 위의 문제점을 해결하기 위해 register-transfer 레벨에서 디지탈 시스템의 동작 특성을 분석하여 ASM(Algorithmic State Machine)[5,18] 도표로 표시한 후 도표의 각 기호와 1대1 대응되는 언어 구조를 지니는 새로운 하드웨어 기술 언어인 SDL (Symbolic Description Languange)를 제안하였다. 따라서 본 논문에서는 SDL을 입력으로 하여 출력으로 게이트 레벨의 논리 회로도를 얻는 SDL 하드웨어 컴파일러를 제안한다.

SDL기술에 대한 어휘 분석을 행하여 디지탈 시스템을 제어부와 데이타 전송부로 분리한 후 임의의 회로소자를 사용하는 설계 방식에도 이용될 수 있는 데이타를 형성하는 알고리즘 Ⅰ을 제안한다. 다음에, 기억 및 지연 소자로 D 플립 플롭을 사용하고 조합 논리 회로의 게이트로는 2-입력 AND와 OR 게이트 및 NOT 게이트를 회로 소자로 사용하여 알고리즘 Ⅰ에서 형성된 데이타에 대한 구문 분석을 행함으로써 게이트 레벨의 논리 회로도와 등가인 NDL(Network Description Language)[18] 표현식을 얻는 알고리즘 Ⅱ를 제안한다.

또한, 본 논문에서 제안하는 알고리즘을 C로 프로그램하여 논리 설계 예에 적용, 설명 함으로써 제안하는 SDL 하드웨어 컴파일러의 유효성을 보인다.

Ⅱ. 하드웨어 기술 언어 : SDL[18]

SDL은 디지탈 시스템의 동작 특성을 register-transfer 레벨에서 분석하여 ASM[18] 도표로 표시한 후 이 도표의 각 기호와 1대1 대응되어 기술, 이해 및 수정이 용이하도록 구문 구조가 설정된 새로운 하드웨어 기술 언어이다. SDL은 시스템의 계층적 설계가 가능하도록 각 모듈단위로 나누어 기술하는데 각 모듈 기술부는 다음과 같은 3부분으로 구성된다.

1. 선언부
2. 제어부의 동기 신호에 따라 수행되는 실행문
3. 제어부의 동기 신호에 무관하게 수행되는 실행문

선언부에서는 기술하는 모듈의 입력, 출력 및 메모리에 해당하는 레지스터, 선 및 버스를 기술한다. 동기신호에 따라 수행되는 실행문에서는 모듈 내의 각 상태인 ASM 블럭[18] 단위로 나누어 기술하는데 기술이 용이하고 이해하기 쉽도록 ASM 도표의 기호 이름을 기술하고 각 기호에 따라 다음과 같은 사항을 기술한다.

1) 출력

① 임의상태에서 전달(←) 및연결(=)이 없는 경우

② ;에 의해 분리 기술되는 무조건 전달 및 연결

③ ;에 의해 분리 기술되는 조건 전달 및 연결

2)분기

① $(IC_1, \cdots, IC_i, \cdots, IC_n)/(S_1, \cdots, S_i, \cdots, S_n)$으로 기술되어 조건 IC_i가 참인 경우 S_i로 분기되는 조건 분기

② $\rightarrow(S_i)$로 기술되어 조건에 관계없이 S_i로 분기되는 무조건 분기

한편, 3은 모듈 기술의 종료 표시인 QN. 다음에 기술되는 실행문인데 시스템 클럭의 동기 신호에 관계없이 항상 실행되는 연결과 모듈 내의 임의 상태가 수행될 때마다 실행되는 전달이다. 참고문헌[19]의 85페이지에서 인용한 light controller의 상태도표는 그림1(a)와 같으며 이에 대한 SDL 기술은 (b)와 같다.

Ⅲ. SDL 하드웨어 컴파일러

일반적으로 하드웨어 컴파일러는 하드웨어의 동작 특성을 register-transfer 레벨 또는 그 이상의 레벨[2]에서 기술한 하드웨어 기술 언어를 입력으로 받아 게이트 레벨의 논리 회로도를 출력으로 변환하는 소프트웨어를 말한다.[4]

그런데, 디지탈 시스템은 일반적으로 그림 2와 같이 1. 제어부(control unit) 2. 데이타 전송부 (data path unit)로 구성된다.[4,10] 따라서, 하드웨어 컴파일러는 디지탈 시스템에 대한 하드웨어 기술 언어를 해석하여 제어부와 데이타 전송부에 대한 데이타를 형성하는 translation과 회로 소자 조건에 따라 논리 회로도를 형성하는 compilation으로 구성된다.

그런데, 데이타 전송부는 ALU, ROM, REGISTER 등의 규칙적인 구조가 대부분이 되어 임의 논리(random logic) 영역을 1차원 MOS게이트 배열[14] 또는 Gate matrix 레이아웃[15] 방식에 의해 설계하는 것이

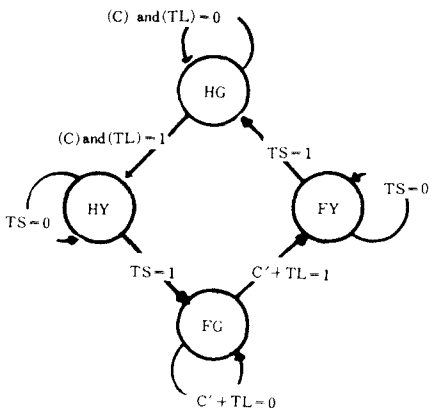


그림1(a). Light controller의 상태도표
Fig. 1(a). State diagram of light controller.

```
SEQSDL : LIGHT CONTROLLER.
INPUTS : C;TL;TS.
OUTPUTS : HL[0 : 1];FL[0 : 1];ST.
BEGIN :
Q1. FL[0]=1 ..
    C1. (!C | !TL, C & TL)/(O1, O2)..
    O1. ST=0;— >(Q1)..
    O2. ST=1;— >(Q2)..
Q2. HL[1]=1;FL[0]=1 ..
    C1. (!TS, TS)/(O1, O2)..
    O1. ST=0;— >(Q2)..
    O2. ST=1;— >(Q3)..
Q3. HL[0]=1 ..
    C1. (C & !TL, !C | TL)/(O1, O2)..
    O1. ST=0;— >(Q3)..
    O2. ST=1;— >(Q4)..
Q4. HL[0]=1;FL[1]=1 ..
    C1. (!TS, TS)/(O1, O2)..
    O1. ST=0;— >(Q4)..
    O2. ST=1;— >(Q1)..
QN.
SN. Q1.
```

그림1(b). Light controller의 SDL 기술
Fig.1(b). SDL description of light controller.

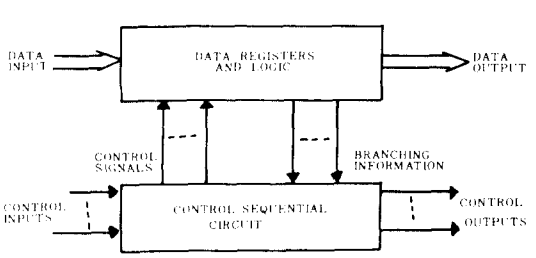


그림 2. 디지탈 시스템의 구성
Fig. 2. Construction of digital system.

일반적이다.

한편, 제어부에 대한 설계 방법은 크게 다음과 같은 2가지로 구분된다.[3]

1. Hardwired control unit
2. Microprogrammed control unit

전자는 후자에 비하여 일반적으로 설계변경은 어려우나 동작 속도가 빠르다는 장점이 있다.[3] 본 논문에서의 하드웨어 컴파일러는 hardwired control unit 설계 방법을 사용한다.

또한, hardwired control unit 설계 방법은 다음과 같은 3가지로 구분된다.[21]

1. 하나의 상태에 하나의 플립 플롭(D 또는 T type)을 할당하고 플립 플롭의 입력 함수를 게이트로 설계하는 방법

2. $(2^{n-1}+1)\sim 2^n$개의 상태에 대하여 n개의 플립 플롭을 할당하고 $n\times 2^n$의 디코더 (decoder)를 게이트로 설계하는 방법

3. 위의 1, 2를 PLA로 설계하는 방법

PLA 설계방법은 규칙적인 구조를 지니므로 설계는 용이하나 동작 속도가 늦고 면적의 낭비가 크다는 단점이 있으며, 2는 1에 비하여 설계 면적을 최소화하기위해서는 상태 할당(state assignment)[13] 문제를 수반하므로 설계가 어렵다는 단점이 있다.[11,13]

본 논문에서는 SDL을 입력으로 하여 모든 hard-wired control unit 설계 방법에 이용될 수 있도록 SDL에 대한 어휘 분석을 행하여 제어부 설계의 데이타인 CLT(Control Logic Table)를 구성한다. 다음에, SDL에 대한 어휘 분석과 CLT를 이용하여 데이타 전송부의 데이타인 DPLT(Data Path Logic Table)를 구성한다.

CLT와 DPLT로부터 게이트 레벨의 논리 회로도를 출력할 때 본 논문에서는 설계의 용이성 및 시스템의 동작 속도를 고려하여 하나의 상태에 하나의 D 플립 플롭을 할당하는 설계 방법을 사용한다. 또한, 데이타 전송부의 기억 및 지연소자도 D 플립 플롭을 사용하며 시스템의 조합 논리 회로 소자로는 2-입력 AND와 OR 게이트 및 NOT 게이트를 사용한다. 따라서, 본 논문에서 제안하는 SDL 하드웨어 컴파일러는 SDL에 대한 어휘 분석을 행하여 CLT 및 DPLT를 구하는 알고리즘 I과 이와같이 구성된 데이타를 사용하여 SDL 하드웨어 컴파일러의 출력인 게이트 레벨의 논리 회로도와 등가인 NDL 표현식을 얻는 알고리즘 II로 구성된다.

알고리즘 I은 디지탈 시스템의 제어부 및 데이타 전

송부를 위한 데이타인 CLT 및 DPLT를 구성하기 위해 SDL에 대한 어휘 분석을 행한다.

CLT는 다음과 같은 3부분으로 구성된다.

1. SOURCE : 분기의 시작기호 표시
2. DESTINATION : 분기의 종료기호 표시
3. CONDITION : SOURCE에서 DESTINATION으로 분기하기 위한 조건 표시

DPLT는 다음과 같은 2부분으로 구성된다.

1. DATA : 레지스터, 선 및 버스 사이에서 전달(←) 및 연결(=) 되는 데이타 표시
2. CONDITION : 전달 및 연결이 발생하기 위한 조건 표시

알고리즘 I

단계 1 : SDL 모듈내의 모든 ASM 블럭에 대하여 CLT의 SOURCE 항은 ASM 블럭의 상태 기호 이름 Q_i로 하고 DESTINATION 항은 SDL문에 기술된 분기점의 기호로 한다.

CLT의 CONDITION 항은 SDL문이 조건·판단 기호 C_i이면 C_i와 분기 입력 조건을 AND한 항이며, SDL문이 조건 출력 기호 O_i이면 CONDITION항은 O_i로 한다.

한편, SDL문이 상태 기호이고 분기가 상태 기호로 행하여 지는 경우 CONDITION항은 없다.

단계 2 : SDL 모듈내의 모든 ASM 블럭에 대하여 DPLT의 DATA 항은 SDL의 상태 기호 또는 조건 출력 기호에 따르는 출력들로 한다. DPLT의 CONDITION 항은 SDL문이 상태 기호 Q_i이면 Q_i로 하며, 조건 출력 기호 O_i이면 그 SDL문을 포함하는 ASM 블럭의 상태 기호 Q_i와 조건 출력 기호 O_i를 AND 한 항으로 한다.

단계 3 : SDL에서 상태 기호 종료 표시인 QN. 다음에 기술된 출력에 대해 어휘 분석을 행하는 단계로 SDL문의 출력을 DPLT의 DATA항에 표시하며, 시스템 출력에 무관하게 동작하는 연결(=)인 경우는 DPLT의 CONDITION항을 ALL-C로 하며 모듈 내의 각 ASM 블럭이 동작 할 때마다 동작하는 출력인 경우는 ALL-T로 표시한다.

위의 단계 1. 부터 3. 까지를 수행하여 얻은 테이블을 CLT-1 및 DPLT-1로 한다.

단계 4 : CLT-1의 CONDITION항이 조건·판단 기호 C_i을 포함하면 C_i을 그 열의 SOURCE 항으로 대체하여 CLT-2로 한다.

단계 5 : CLT-2의 CONDITION 항에 조건·판단 기호 또는 조건 출력 기호가 포함되고 그 이름이 다른 열의 DESTINATION 항과 같으며 두열의 SOURCE항이 같은 열이 있으면 조건 판단 기호 또는 조건 출력 기호를 다른 열의 CONDITON 항으로 대체하여 CLT-3으로 한다.

단계 6 : CLT-3의 DESTINATION 항이 조건 판단 기호이면 그 열을 제거한 다음 DESTINATION 항을 정렬 키(sorting key)로 하여 사전 편찬식 정렬을 행한다. 이 테이블을 CLT-4로 하여 Hardwired control unit 설계 방법을 이용한 제어부 설계의 데이타로 한다.

단계 7 : DPLT-1의 CONDITION 항에 조건 출력 기호가 포함되어 있으며 이 CONDITION 항에 포함된 상태 기호와 조건 출력 기호가 CLT-4의 SOURCE 및 DESTINATION 항과 각각 같은 경우 DPLT-1의 CONDITION 항 내에 포함된 조건 출력 기호를 CLT-4의 CONDITION 항으로 대체한다.

단계 8 : DPLT-1의 DATA 항이 서로 같은 열은 각 CONDITION 항을 OR한 후 최상 위 열의 CONDITION항으로 한후 나머지 열은 제거하여 DPLT-2로 한다.

단계 9 : DPLT-2의 DATA 항이 전달 연산자 ←를 포함하는 경우 DATA 항의 SOURCE(←의 오른쪽)에 대한 DESTINATION(←의 왼쪽)의 비트 map을 구한다. DATA 항이 B[0:7]←BUS[4:7], *\A[0:3], BUS[0:2]인 경우 비트 map은 B[0:3], B[4] 및 B[5:7]이다.

단계10 : 단계 9에서 구한 각 비트 map에서 서로 중복되는 비트는 1개씩 남기고 제거하여 독립 비트 map을 구한다. 비트 map이 A[0], A[0:5] 및 A[3:7]이면 독립 비트 map은 A[0], A[1:3], A[4:5] 및 A[6:7]이다. 이와같은 독립 비트 map과 DPLT-2의 DATA 항이 연결 연산자(=)를 포함하는 경우 그 열의 DESTINATION(=의 왼쪽)을 DPLT-3의 DESTINATION GROUP 항으로 한다.

단계11 : 단계10에서 구한 DESTINATION GROUP 항으로 전달 및 연결되는 입력(←와 =의 오른쪽)에 대한 비트 map을 구하여 DPLT-3의 SOURCE GROUP 항으로 한다. 또한, 이와같은 DESTINATION GROUP 및 SOURCE GROUP을 포함하는 DPLT-2열의 CONDITION 항을 DPLT-3의 CONDITION 항으로 한다. DPLT-2의 DATA 항이 B[0;7]←C[0:3], C[4:7]과 B[0:7]←C[0], *\D[0:3], C[2:7]인 경우 DESTINATION GROUP과 SOURCE GROUP 쌍은 다음과 같다. :

DESTINATION GROUP	SOURCE GROUP
B[0]	C[0]
B[1]	C[1]과 +\D[0:3]
B[2:3]	C[2:3]
B[4:7]	C[4:7]

단계11에서 얻은 DPLT-3을 데이타 전송부 설계를 위한 데이타로 한다.

참고문헌[4]의 157페이지에 있는 AHPL기술에 대한 SDL기술은 그림3과 같으며 이에 대한 CLT-1~CLT-4는 각각 표 1~4와 같다. 또한, 같은 문헌에서 242페이지의 AHPL기술에 대한 SDL기술은 그림4와 같으며 이에 대한 DPLT-1~DPLT-3은 각각 표5~7과 같다.

한편, 본 논문에서 제안하는 SDL 하드웨어 컴파일러는 알고리즘 I에 의하여 얻은 CLT-4와 DPLT-3을 구문 분석하여 하드웨어 컴파일러의 최종 출력인 게이트 레벨의 논리 회로도와 등가인 NDL 표현식을 얻는다. 이때, 기억 및 지연 소자로 D 플립플롭을 사용하고 조합 논리 회로를 위한 게이트로는 2-입력 AND와 OR 게이트 및 NOT 게이트를 사용한다.

구문 분석의 식이 !a&(b|*\C[0:2])인 경우 이 식을 게이트 회로도로 표시하기 위하여 연산자 우선 순위를 ( ), !, &, | 순으로 하면 NDL 표현식은 다

```
SEQSDL : NO DELAY.
INPUTS : a;b0;b1;b2;X[8].
OUTPUTS : B[8];OUT1;OUT2;OUT3.
MEMORY : A[8];BUS[8].
BEGIN :
Q1 . .
        C1. (a, !a)/(O1, Q1). .
        O1. OUT1=1;- >(C2). .
        C2. (b0, b1, !(b0/b1))/(O2, O3, O4). .
        O2. OUT2=1;- >(O3). .
        O3. OUT3=1;- >(O4). .
        O4. A[0 : 7]<-X[0 : 7];- >(Q2). .
Q2. BUS[0 : 7]=A[0 : 7];B[0 : 7]<- -BUS[4 : 7], BUS[0 : 3];- > (Q1).
QN.
SN. Q1.
```

그림 3. SDL 기술 예
Fig. 3. Example of SDL Description.

표 1. CLT-1
Table 1. CLT-1.

CONTROL SOURCE	LOGIC DESTINATION	TABLE-1 CONDITION
Q1	O1	C1&a
Q1	Q1	C1&!a
Q1	C2	O1
Q1	O2	C2&b0
Q1	O3	C2&b1
Q1	O4	C2&!(b0 | b1)
Q1	O3	O2
Q1	O4	O3
Q1	Q2	O4
Q2	Q1	

표 2. CLT-2
Table 2. CLT-2.

CONTROL SOURCE	LOGIC DESTINATION	TABLE-2 CONDITION
Q1	O1	Q1&a
Q1	Q1	Q1&!a
Q1	C2	O1
Q1	O2	C2&b0
Q1	O3	C2&b1
Q1	O4	C2&!(b0 | b1)
Q1	O3	O2
Q1	O4	O3
Q1	Q2	O4
Q2	Q1	

표 3. CLT-3
Table 3. CLT-3.

CONTROL SOURCE	LOGIC DESTINATION	TABLE-3 CONDITION
Q1	O1	Q1&a
Q1	Q1	Q1&!a
Q1	C2	Q1&a
Q1	O2	Q1&b0&a
Q1	O3	Q1&b1&a|Q1&b0&a
Q1	O4	Q1&!(b0|b1)&a|Q1&b1&a|Q1&b0&a
Q1	Q2	Q1&!(b0|b1)&a|Q1&b1&a|Q1&b0&a
Q2	Q1	

표 4. CLT-4
Table 4. CLT-4.

CONTROL SOURCE	LOGIC DESTINATION	TABLE-4 CONDITION
Q1	O1	Q1&a
Q1	O2	Q1&b0&a
Q1	O3	Q1&b1&a|Q1&b0&a
Q1	O4	Q1&!(b0|b1)&a|Q1&b1&a|Q1&b0&a
Q1	Q1	Q1&!a
Q2	Q1	
Q1	Q2	Q1&!(b0|b1)&a|Q1&b1&a|Q1&b0&a

```
SEQSDL : MULTISHIFT.
INPUTS : a;b;X{6}.
OUTPUTS : Z;LOOK;A{0 : 5}.
MEMORY : A{18};CNT{3}.
BEGIN :
Q1. A{0 : 17}<—0, A{0 : 16}..
    C1. (a, !a)/(Q3, Q2)..
Q2. A{0 : 17}<—0, A{0 : 16}; →(Q1)..
Q3. CNT{0 : 2}<—a, b, 0; A{0 : 11} ← X{0 : 5}, A{0 : 5}..
    C1. (b, !b)/(Q1, Q4)..
Q4. A{0 : 17}<—1, A{0 : 16}; CNT{0 : 2}<—COMINC{CNT{0 : 2}}..
    C1. (* \ CNT{0 : 2}, !* \ CNT{0 : 2})/(Q5, Q4)..
Q5. LOOK=1 ..
    C1. (!b, b)/(01, 02)..
    01. A{0 : 5} ← A{6 : 11} & A{12 : 17}; →(Q1)..
    02. A{0 : 5}<—A{6 : 11}; A{6 : 17}<—X{0 : 5}, A{0 : 5}; —>(Q1)..
QN.
C.Z=A{17}.
SN.Q1.
```

그림 4. SDL 기술 예
Fig. 4. Example of SDL description.

음과 같으며 이에 대한 회로도는 그림 5 와 같다.

1 = AND C[0] C[1]
2 = AND 1 C[2]
3 = OR b 2
4 = NOT a
5 = AND 4 3

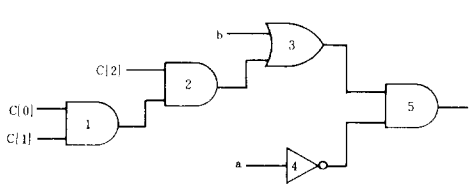


그림 5. 논리 회로도
Fig. 5. Logic diagram.

이를 위한 알고리즘은 다음과 같다.

알고리즘 Ⅱ.

표 5. DPLT-1
Table 5. DPLT-1.

DATA PATH LOGIC	TABLE-1
DATA	CONDITION
A[0 : 17]<−0, A[0 : 16]	Q1
A[0 : 17]<−0, A[0 : 16]	Q2
CNT[0 : 2]<−a, b, 0	Q3
A[0 : 11]<−X[0 : 5], A[0 : 5]	Q3
A[0 : 17]<−1, A[0 : 16]	Q4
CNT[0 : 2]<−COMINC{CNT[0 : 2]}	Q4
LOOK=1	Q5
A[0 : 5]<−A[6 : 11] & A[12 : 17]	Q5&O1
A[0 : 5]<−A[6 : 11]	Q5&O2
A[6 : 17]<−X[0 : 5], A[0 : 5]	Q5&O2
Z=A[17]	ALL-C

표 6. DPLT-2
Table 6. DPLT-2.

DATA PATH LOGIC	TABLE-2
DATA	CONDITION
A[0 : 17]<−0, A[0 : 16]	Q1 | Q2
CNT[0 : 2]<−a, b, 0	Q3
A[0 : 11]<−X[0 : 5], A[0 : 5]	Q3
A[0 : 17]<−1, A[0 : 16]	Q4
CNT[0 : 2]<−COMINC{CNT[0 : 2]}	Q4
LOOK=1	Q5
A[0 : 5]<−A[6 : 11] & A[12 : 17]	Q5&!b
A[0 : 5]<−A[6 : 11]	Q5&b
A[6 : 17]<−X[0 : 5], A[0 : 5]	Q5&b
Z=A[17]	ALL-C

단계 1 : CLT-4의 DESTINATION 항이 Q_i인 모든 열에 대하여 CONDITION 항이 존재하면 이들을 OR하며, 존재하지 아니하면 SOURCE 항들을 OR한 식에 대하여 구문 분석을 행한다. 이 구문 분석의 최종 출력 신호 이름을 제어부의 i번째 D플립 플롭 입력 신호로 하며 CLT-4의 모든 Q_i에 대하여 단계 1을 반복 수행한다.

단계 2 : DPLT-3의 SOURCE GROUP이 DESTINATION GROUP에 전달(←)되는 모든 열에 대하여 단계2와 3을 반복 수행한다. DESTINATION GROUP이 같은 열에 대하여 SOURCE GROUP과 CONDITION항을 AND 하고 이 식들을 OR한 식에 대하여 구문 분석을 행한다. 이 구문 분석의 최종 출력 신호 이름을 출력 신호 이름이 DESTINATION GROUP인 D플립 플롭의 입력 신호로 한다. AND를 행할때 SOURCE GROUP이 1인 경우는 CONDITION 항만을 포함하며, 0인 경우는 CONDITION항은 포함시키지 않는다.

단계 3 : DESTINATION GROUP이 같은 열에 대하여 CONDITION 항들을 OR한 식에 대하여 구문 분석을 행하여 최종 출력 신호 이름과 시스템 클럭을 AND한 신호를 출력 신호 이름이 DESTINATION GROUP인 D 플립 플롭의 클럭 신호로 한다.

단계 4 : DPLT-3의 SOURCE GROUP이 DESTINATION GROUP에 연결(=)되는 모든 열에 대하여 단계 4를 반복 수행한다. DESTINATION GROUP이 같은 열에 대하여 SOURCE GROUP과 CONDITION항을 AND하고,

표 7. DPLT-3
Table 7. DPLT-3.

DATA PATH LOGIC TABLE-3		
DESTINATION GROUP	SOURCE GROUP	CONDITION
A〔0〕	0	Q1|Q2
A〔0〕	X〔0〕	Q3
A〔0〕	1	Q4
A〔0〕	A〔6〕 & A〔12〕	Q5&!b
A〔0〕	A〔6〕	Q5&b
A〔1 : 5〕	A〔0 : 4〕	Q1|Q2
A〔1 : 5〕	X〔1 : 5〕	Q3
A〔1 : 5〕	A〔0 : 4〕	Q4
A〔1 : 5〕	A〔7 : 11〕 & A〔13 : 17〕	Q5&!b
A〔1 : 5〕	A〔7 : 11〕	Q5&b
A〔6 : 11〕	A〔5 : 10〕	Q1 | Q2
A〔6 : 11〕	A〔0 : 5〕	Q3
A〔6 : 11〕	A〔5 : 10〕	Q4
A〔6 : 11〕	X〔0 : 5〕	Q5&b
A〔12 : 17〕	A〔11 : 16〕	Q1|Q2
A〔12 : 17〕	A〔11 : 16〕	Q4
A〔12 : 17〕	A〔0 : 5〕	Q5&b
CNT〔0〕	a	Q3
CNT〔0〕	COMINC|CNT〔0 : 2〕|	Q4
CNT〔1〕	b	Q3
CNT〔1〕	COMINC|CNT〔0 : 2〕|	Q4
CNT〔2〕	0	Q3
CNT〔2〕	COMINC|CNT〔0 : 2〕|	Q4
LOOK	1	Q5
Z	A〔17〕	ALL-C

이 식들을 OR한 식에 대하여 구문 분석을 행하여 최종 출력신호 이름이 DESTINATION GROUP인 조합논리 회로를 형성한다. AND식을 구할때 SOURCE GROUP이 1인 경우는 CONDITION 항만을 포함시키며, 0인 경우는 CONDITION 항도 포함시키지 않는다.

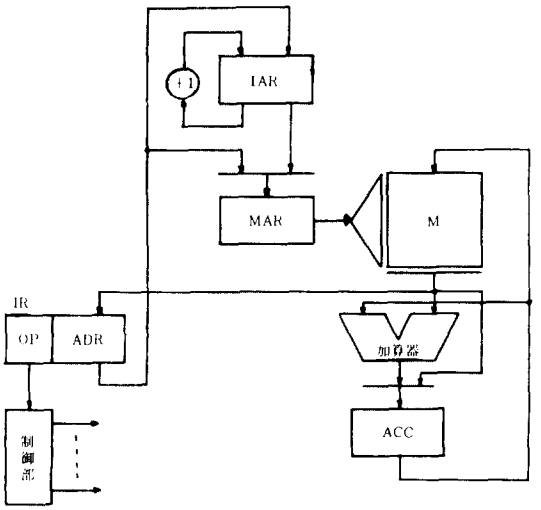
단계 5 : CONDITION 항이 ALL-C이면 SOURCE GROUP에 대해 구문 분석을 행하여 최종출력 신호 이름을 DESTINATION GROUP으로 한다. CONDITION 항이 ALL-T이면 제어부 회로의 모든 D플립 플롭 출력을 OR한 식과 SOURCE GROUP을 AND한 식에 대해 구문 분석을 행한다. 이 구문 분석의 최종 출력 신호 이름을 출력신호 이름이 DESTINATION GROUP인 D 플립 플롭의 입력 신호로 한다. 또한, 제어부 회로의 모든 D플립 플롭 출력을 OR한 식에 대해 구문 분석을 행하고 최종 출력 신호에 시스템 클럭을 AND하여 이 신호를 D 플립 플롭의 클럭 신호로 한다.

예제 2로서 그림 6의 블럭도와 그림 7의 상태 천이도를 갖는 디지탈 시스템[20]에 대한 SDL 기술은 그림 8과 같다. SDL 하드웨어 컴파일러에 의한 제어부와 데이타 전송부의 NDL 기술은 표 8과 표 9과 같다. 또한, 이에 대한 게이트 레벨 논리 회로도는 그림 9와 같다.

Ⅳ. 結 論

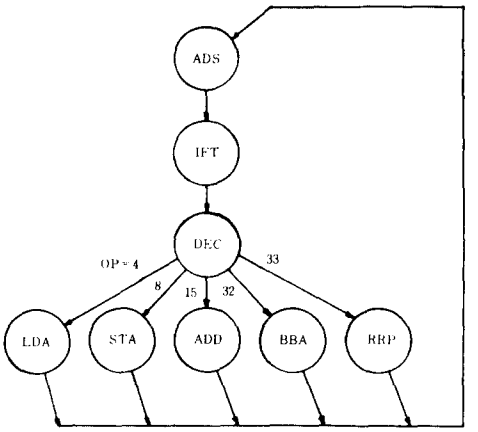
본 논문에서는 논리 설계 자동화를 위해 하드웨어 기술 언어인 SDL에 대한 하드웨어 컴파일러를 제안했다.

디지탈 시스템의 동작 특성을 register-transfer 레벨에서 기술한 SDL에 대하여 어휘 분석을 행하여 hardwired control unit 설계방식 모두에 이용될 수 있는 제어부와 데이타 전송부를 위한 중간 코드를 구하는 알고리즘 I을 제안했다. 다음에, 기억 및 지연 소



MAR : Memory Address Register
IAR : Instruction Address Register
IR : Instruction Register
ACC : Accumulator

그림 6. 가상 머신의 블럭도
Fig. 6. Block diagram of virtual machine.



ADS : Address Set
IFT : Instruction Fetch
DEC : Decode
LDA : Execution of LOAD Instruction
STA : Execution of STORE Instruction
ADD : Execution of ADD Instruction
BRA : Execution of BRANCH Instruction
BRP : Execution of BRANCH-POSITIVE Instruction

그림 7. 그림 6의 상태 천이도
Fig. 7. State transition diagram of Fig. 6.

자로 D 플립 플롭을 사용하고, 2-입력 AND와 OR게이트 및 NOT 게이트를 사용하여 알고리즘 I에서 생성된 중간 고드를 구문 분석 함으로써 게이트 레벨의 논리 회로도와 등가인 NDL 표현식을 얻는 알고리즘 II를 제안했다. 또한, 본 논문에서 제안하는 SDL 하드웨어 컴파일러의 효용성을 보이기 위해 C언어로 프로그램하여 실제의 예에 적용 설명했다.

```
SEQSDL : CPU.
INPUTS :.
OUTPUTS :.
MEMORY : M[1024, 16];ACC[16];IR[16];MAR[10];IAR[10].
BEGIN :
Q1. MAR[0 : 9]<-IAR[0 : 9];IAR[0 : 9]<-COMINC{IAR[0 : 9]};->(Q2)..
Q2. IR[0 : 15]<-COMBUSM{MAR[0 :9]};->(Q3)..
Q3. MAR[0 : 9]<-IR[6 : 15]..
    C1. (IR[3], IR[2], IR[1], IR[0], IR[0] & IR[5])/(Q4, Q5, Q6, Q7, Q8)..
Q4. ACC[0 : 15]<-COMBUSM{MAR[0 : 9]};->(Q1)..
Q5. COMBUSM{MAR[0 : 9]}<-ACC[0 : 15];->(Q1)..
Q6. D[0 : 15]=COMBUSM{MAR[0 : 9]};ACC[0 : 15<-COMADD{ACC[0 : 15]%D[0 : 15]};->(Q1)..
Q7. IAR[0 : 9]<-IR[6 : 15];->(Q1)..
Q8..
    C1. (!ACC[0], ACC[0])/(O1, Q1)..
    O1. IAR[0 : 9]<-IR[6 : 15];->(Q1)..
QN.
SN. Q1.
```

그림 8. 그림 6의 SDL 기술
Fig. 8. SDL description of Fig. 6.

표 8. 제어부의 NDL 출력

Table 8. NDL output of control unit.

NETWORK	DESCRIPTION	LANGUAGE FOR CONTROL	LOGIC
Q1	D F/F DATA INPUT		
1	= NOT	ACC[0]	
2	= AND	Q8	ACC[0]
3	= AND	Q8	1
4	= OR	2	3
5	= OR	Q5	Q6
6	= OR	5	Q7
7	= OR	6	Q4
8	= OR	7	4
Q2	D F/F DATA INPUT		
Q1			
Q3	D F/F DATA INPUT		
Q2			
Q4	D F/F DATA INPUT		
9	= AND	Q3	IR[3]
Q5	D F/F DATA INPUT		
10	= AND	Q3	IR[2]
Q6	D F/F DATA INPUT		
11	= AND	Q3	IR[1]
Q7	D F/F DATA INPUT		
12	= AND	Q3	IR[0]
Q8	D F/F DATA INPUT		
13	= AND	Q3	IR[0]
14	= AND	13	IR[5]

표 9. 데이타 전송부의 NDL 출력

Table 9. NDL output of data transfer unit.

NETWORK	DESCRIPTION	LANGUAGE FOR DATA	PATH	LOGIC
MAR[0:9]	D F/F DATA INPUT			
15	= AND	IAR[0:9]	Q1	
16	= AND	IR[6:15]	Q3	
17	= OR	15	16	
MAR[0:9]	D F/F CLOCK INPUT			
18	= OR	Q1	Q3	
19	= AND	18	CLK	
IAR[0:9]	D F/F DATA INPUT			
20	= NOT	ACC[0]		
21	= AND	Q8	20	
22	= OR	Q7	21	
23	= AND	COMINC|IAR[0:9]|	Q1	
24	= AND	IR[6:15]	22	
25	= OR	23	24	

IAR[0 : 9]			D F/F CLOCK INPUT	
26	=	OR	Q1	22
27	=	AND	26	CLK
IR[0 : 15]			D F/F DATA INPUT	
28	=	AND	COMBUSM¦MAR[0 : 9]¦	Q2
IR[0 : 15]			D F/F CLOCK INPUT	
29	=	AND	Q2	CLK
ACC[0 : 15]			D F/F DATA INPUT	
30	=	AND	COMBUSM¦MAR[0 : 9]¦	Q4
31	=	AND	COMADD¦ACC[0 : 15]%D[0 : 15]¦	Q6
32	=	OR	30	31
ACC[0 : 15]			D F/F CLOCK INPUT	
33	=	OR	Q4	Q6
34	=	AND	33	CLK
COMBUSM¦MAR[0 : 9]¦			D F/F DATA INPUT	
35	=	AND	ACC[0 : 15]	Q5
COMBUSM¦MAR[0 : 9]¦			D F/F CLOCK INPUT	
36	=	AND	Q5	CLK
D[0 : 15]			DATA INPUT	
37	=	AND	COMBUSM¦MAR[0 : 9]¦	Q6

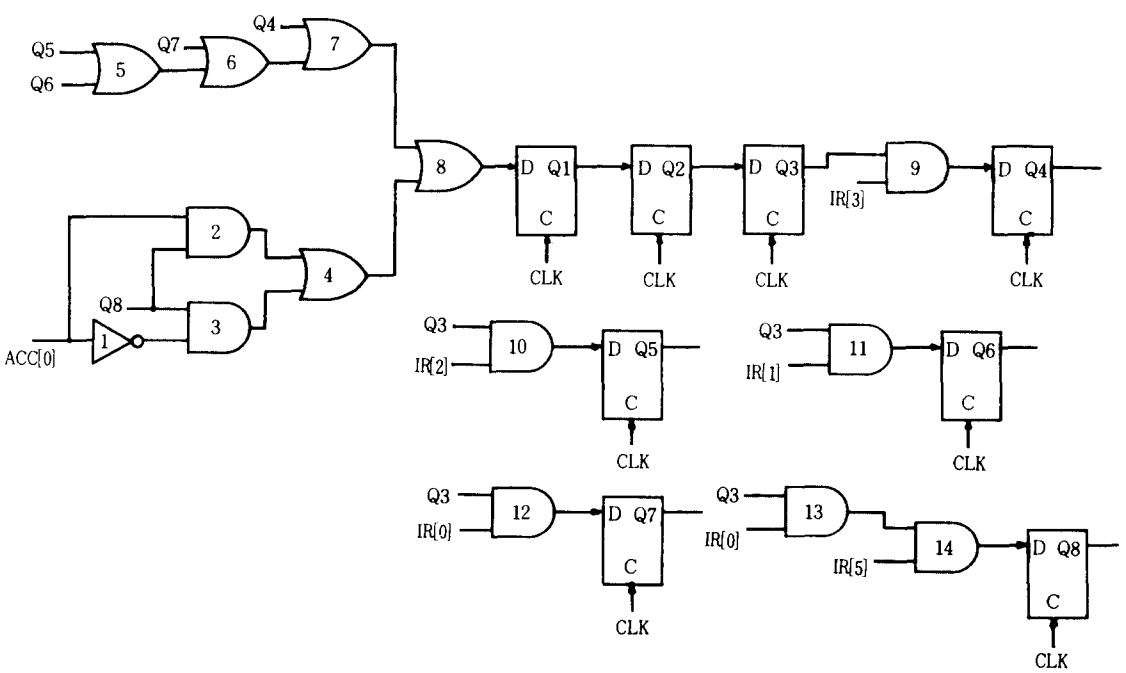


그림9. (a) 제어부의 논리 회로도
Fig. 9. (a) Logic diagram of control unit.

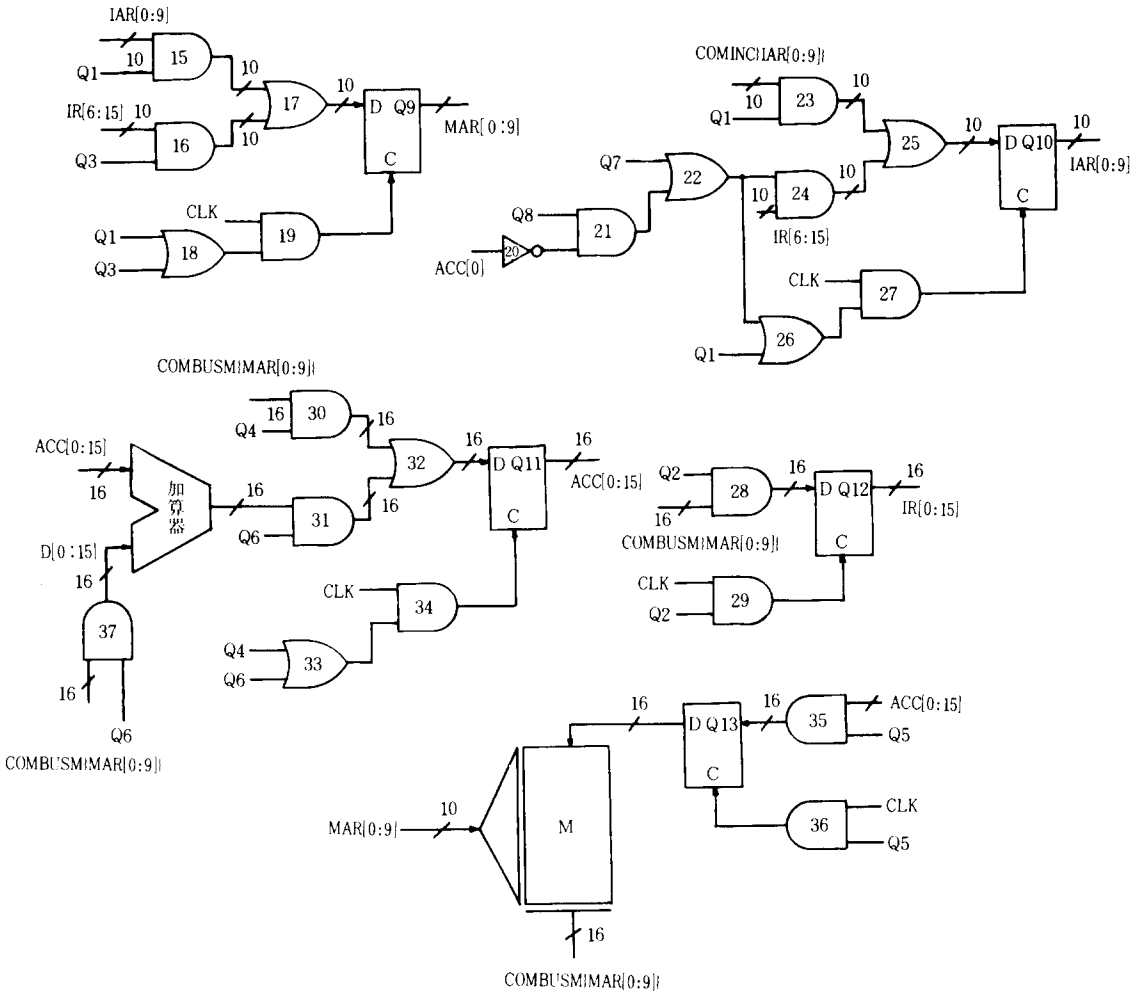


그림9. (b) 데이타 전송부의 논리 회로도
Fig. 9. (b) Logic diagram of data transfer unit.

본 논문에서 제안한 SDL 하드웨어 컴파일러를 사용하여 대규모 디지탈 시스템을 설계하면 빠른 시간내에 설계의 잘못이 없도록 설계할 수 있어 설계 비용을 크게 줄일 것으로 기대된다.

앞으로의 연구과제로는 설계 specification에 대한 SDL 기술의 검증을 행하기 위한 SDL 시뮬레이터에 대한 연구와 본 논문의 NDL 출력을 사용하여 설계 최적화를 행하기 위한 local transformation[16]에 대한 연구가 계속되어야 할 것이다.

參 考 文 獻

[1] Parker, A.C., "Automated Synthesis of Digital Systems," *IEEE Trans. Design & Test*, vol. 1, no. 4, pp. 75-81, Nov. 1984.

[2] Shiva, S.G., "Computer Hardware Description Language-A Tutorial," *Proceedings of IEEE*, pp. 1605-1615. Dec. 1979,

[3] Shiva, S.G., Computer Design and Architecture, Little and Brown Company, 1985.

[4] Hill, F.J. and Peterson, G.R., Digital Systems; Hardware Organization and Design., 2nd ed., Wiley Press, 1978.

[5] Hill, F.J. and Peterson, G.R., Digital Logic and Microprocessors, Wiley press, 1984.

[6] Duley, J.R. and Dietmeyer, D.L., "A Digital System Design Language," *IEEE Trans. Computers*, vol. C-17, no. 9, pp. 850-861, Sep. 1968.

[7] Duley, J.R. and Dietmeyer, D.L., "Translation of a DDL Digital System Specification to Boolean Equations," *IEEE Trans. Computers,* vol. C-18, no. 4, pp. 305-313, Apr. 1969.

[8] Swanson, R., Navabi, Z. and Hill, F.J., *An AHPL Compliler/Simulator System.* Proc. 6th Texas Conference on Computing Systems, Austin, Tex., 1977.

[9] G. Odawara, J. Sato and M. Tomita, "A Symbolic Funtional Description Language," Proc. IEEE ACM 21st Design Automation Conf., pp. 73-80, 1984.

[10] Brown, D.W., "A State-Machine Synthesis-SMS." Proc. IEEE ACM 18th Design Automation Conf., pp. 301-305, 1981.

[11] Kang, S.,"Automatic PLA Synthesis from a DDL-P Description," Proc. IEEE ACM 18th Design Automation Conf., pp. 391-397, 1981.

[12] Meyer, M.J., "A VLSI FSM Design System," Proc. IEEE ACM 21th Design Automation Conf., pp. 434-440, 1984.

[13] G. De Micheli, M. Hoffman, A.R. Newton, and A.L. Sangiovanni-Vincentelli," Optimal State Assignment for Finite State Machines," *IEEE Trans. CAD,* vol. CAD-4, no. 3, pp. 269-284, July 1985.

[14] 조중휘, 정정화, "1차원 MOS-LSI 게이트 배열 알고리즘," 전자공학회지, 제21권 제4호, pp. 13-16, 7월 1984년.

[15] 최규명, "Gate Matrix Layout 방식에 의한 MOS VLSI Layout 설계에 관한 연구," 한양대학교 석사학위논문, 1984년.

[16] 이성봉, "VLSI 논리설계 최적화를 위한 Rule-Based System," 한양대학교 석사학위논문, 1985년.

[17] 박성범, "논리설계 자동화를 위한 하드웨어 컴파일러에 관한연구," 한양대학교 석사학위논문, 1985년.

[18] 조중휘, 정정화, "VLSI의 논리설계 자동화를 위한 ASM도표와 SDL," 전자공학회지, 제23권 제2호, pp. 137-147, 3월 1986년.

[19] Mead, C. and Conway, L., *Introduction to VLSI Systems,* Addision Wesley, 1980.

[20] 임인칠, "디지탈 시스템 설계언어," 한국정보과학회지, 제2권 제3호, pp. 3~17, 9월 1984년.

[21] M. Obrebska, "Algorithm Transformations Improving Control Part Implementation," Proc. IEEE ICCD'83, pp. 307-310.