論文 88-25-9-14

데이타 변환에 의한 PC 상에서의 IC 레이아웃 프로그램 개발에 관한 연구

(A Study on Development of IC Layout Program with PC by Data Conversion)

白寅天**, 朴魯京*, 全興雨*, 文大哲*, 車均鉉*

(In Cheon Back, Nho Kyung Park, Heung Woo Chun, Dai Tchul Moon and Kyun Hyon Tchah)

要 約

YACC를 이용하여 DXF와 CIF의 파서를 제작하고 CIF를 플로팅하기 위해 DXF로의 변환 그리고 AUTOCAD의 도형정보를 CIF로 변환하였다. 복잡한 문법이나 확장의 가능성을 갖는 입력의 파서제작은 힘이 드나 YACC를 이용함으로 용이하고도 문법인식의 정당성을 갖는 파서를 제작했다. 본 프로그램으로 실제 마스크 제작이 가능한 LSI급 설계를 비싼 WORKSTATION 없이도 할 수 있다.

Abstract

This paper described construction of CIF and DXF parser using YACC (yet another compiler compiler), in order to plot CIF, and conversion of DXF into CIF for geometrical information of AUTOCAD. It is difficult to construct parper for input with complex grammar or with grammar which has possibilities of extension, and to varify the validity. YACC, a LALR parser generator, is a effective tool in these works. Also DRC or ERC can be developed quickly using primitive CIF in this paper. So, it is practicable to design IC at PC without expensive workstations.

I. 서 론

집적회로 설계시 설계시간의 단축, 설계변경의 용이, 높은 신뢰도 등으로 CAD의 사용이 급증하고 있다. 일반 CAD 장비에서 사용하는 데이타 코드로는 CIF, GDSII, FDR, DXF, EDIF 등을 들수 있다. 이러한 여러 데이타들 사이의 공유는 매우 필요하다. 위의 코드중 CIF는 display, plotter, PG file. DRC 등의 입력으로 사용되는 intermediate code로 현재 전 세계적으로 널리 쓰이고 있다. 본 연구에서는 CIF를 PC상에서 수용할 수 있도록 CIF와 DXF의 파서를 제작하고 상호 변환을 수행하였다. 이를 수행시 YACC(yet another compiler compiler)를 사용하여 효과적인 프로그램 개발을 시도하였다. 장비로

*正會員, **準會員, 高麗大學校 電子電算機工學科
(Dept. of Elec. & Comp. Eng., Korea Univ.)
接受日字 : 1987年 8月 22日

는 IBM PC의 XENIX, UNIX V와 DOS상에서 수행되는 AUTOCAD를 사용하였다.[1,2,8,9]

Ⅱ. 파싱이론

파싱이란 문법(grammar)에 따른 언어(language)가 될 수 있는 스트링(string)을 받아서 파싱 나무구조(parse tree)를 출력하는 것을 말한다. parse tree는 생성의 오른쪽을 변화시키는 유도(derivation)에 의해서 이루어지게 되는데 유도에는 좌단 유도(left most derivation), 우단 유도(right most derivation)의 두 종류가 있다.[11]

파싱의 첫째종류는 좌단유도를 사용하는 top-down 파싱으로 그예에는 predictive parsing 이 있는데 이런 것을 LL 파싱이라 한다. 위와는 반대로 우단유도의 역을 사용하는 bottom-up 파싱의 예는 연산자 순위파싱, LL 보다 더 일반적인 문법을 표시할 수 있는 LR 파싱등을 말한다.[3,10]

1. LR(left right) 파서

LR파서 구성의 기본 원리는 오른쪽에 대해 어떠한 심볼도 포함하지 않는 rigtht sentential form 의 prefix인 viable prefixes 를 인식하는 DFA(deterministic finite automation) 를 구성하여 만들어진다. 이 DFA는 주어진 문법의 viable prefixes에 대한 valid items 를 찾아서 구성된다. 이것은 거의 모든 context free grammar에 적용되며, LL 파서보다 일반적이다. LR 파서의 종류로는 어느 한 순간까지 유도된 문법의 생성이되는 LR(0) 항목들을 CLOSURE와 GOTO 함수들을 이용해 파서를 구성하는 SLR (simple left right) 파서와 SLR의 약점을 보완하기 위해 입력을 lookahead 할 수 있는 터미널을 추가한 LR(1) 항목으로 구성하나 CLR(cannonical left right) 파서, CLR 파서의 space를 줄이기 위해 core 항목만을 합친 LALR (look ahead left right) 파서를 들 수 있다.[10] 그림 1은 LR 파서의 모형도이다.

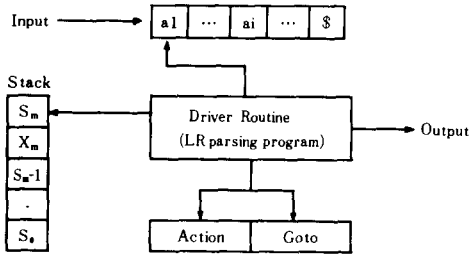


그림 1. LR파서의 모형도
Fig. 1. Model of LR parser.

2. LALR(look ahead left right) 파서의 구성 방법

단계1: LR(0) 항목집합에 lookahead를 추가한다. 방법은 다음과 같다.

1) X 전이
[A →a * Xb, a]는 [A →aX * b, a]

2) E 전이
[A →a * Bb, c] 는 [B →*r, T] 이때 B→r은 생성, T는 다음의 경우에 d도 구성된다.

a. FIRST(b)가 d일때

b. b→E 일때 c는 d가 된다.

단계2: 단계1의 방법으로 구해진 LR(1) 항목중 첫째요소가 같은 것 들을 합쳐 core 들만 남긴다.

단계3: 단계2의 항목들을 이용하여 파싱 테이블을 만든다.

1) [A →B * ac, b]가 상태 I에 있을때 상태 I에서 터미널 a에 대해 SHIFT(I,a)가 있게 된다.

2) [A →B *, a]가 I에 있을 때, 터미널 a에 대해 REDUCE(rule number)가 있게 되고 생성 좌측 심볼에 대해 GO TO (I, left symbol)을 놓는다.[3,10]

그림 2는 lookhead를 찾는 방법의 흐름도이다.

Ⅲ. YACC(yet another compiler compiler)

1970년 초 S.C Jonson에 의해 개발된 YACC 는 복잡한 문법을 갖는 입력을 표시하기 위한 도구로 사용되는 LALR 파서 생성기 이다. YACC에서 사용하는 파싱 테이블과 구동루틴의 알고리듬은 다음과 같다.[4,5]

1. 파싱 테이블 및 스택

일반적인 파싱 테이블은 각 스테이지(stage) 에서 터미널 심볼(terminal symbol)에 따른 action (shift, reduce)과 문법의 left symbol에 따라 가게될 stage 번호로 구성된다. YACC에서 이 파싱 테이블의 구성은 accept 처리 테이블, action 테이블, GO TO 테이블, 우측 심볼길이 테이블, check, definition 등을 위한 8개의 일차원 배열로 되어있다. 또 스택은 maxdepth의 길이를 갖는 일차원 배열로 되어있다.

2. 구동 루틴의 알고리듬

최초의 상태를 0으로 시작하여 입력에 아무것도 없으면 LEX 루틴에서 입력을 하나 받아 현재 상태 정보를 이용하여 테이블을 찾아 스택에서 shift 할 것

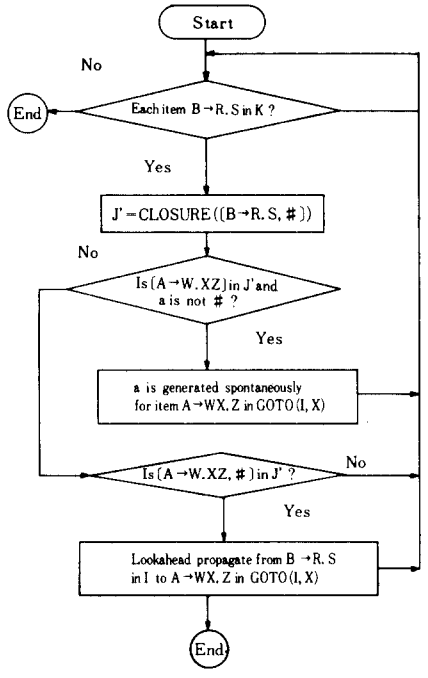


그림 2. 전달 및 자연발생 lookahead 추출을 위한 흐름도

Fig. 2. Flowchart of discovering propagated and spontaneous lookahead.

```
cifFile     : commands endcommand
 (X)        :   (Y)       (Z)

commands    : command
            | commands command
            ;           (W)

command     : BOX (h)
            | POLYGON (p)
            ;

endcommand  : E (e)
            ;
```

그림 3. CIF 화일의 YACC 기술

Fig. 3. YACC description of CIF file.

(1) X → YZ
(2) Y → W
(3) Y → YW
(4) W → b
(5) W → p
(6) Z → e

그림 4. 규칙 번호

Fig. 4. Rule number.

표 1. 스택의 동작

Table 1. Operation of stack.

	스 택	입 력	출 력
(1)	0	bpbe $	
(2)	0b 4	pbe $	W → b
(3)	0W3	pbe $	Y → W
(4)	0Y 2	pbe $	
(5)	0Y 2p 5	be $	W → p
(6)	0Y 2W7	be $	Y → YW
(7)	0Y 2	be $	
(8)	0Y 2b 4	e $	W → b
(9)	0Y 2W7	e $	Y → YW
(10)	0Y 2	e $	
(11)	0Y 2e 8	$	Z → e
(12)	0Y 2Z 6	$	X → YZ
(13)	0X 1	$	
(14)	accept		

인지, reduce 할 것인지를 결정한다. shift 될 때에는 yyact 와 yypact 테이블을 사용하여 다음 상태 정보를 얻고, yyahk 테이블에서 올바르게 shift 가 일어나는지 확인한다. Reduce 시는 스택 포인터와 스택 value 포인터를 reduce 되는 규칙이 오른쪽에 있는 심볼 수만큼 감소시키고(pop off), GO TO 테이블을 찾아 그 상태로 간다. 또한 reduce 시 행동은 감축되는 규칙의 번호의 case 에 따라 행동한다.

오류 발견시는 오류 처리를 위한 루틴으로 간다. 이 과정을 되풀이 하여 최종단계의 룰이 아무런 오류없이 reduce 되면 accept 되어 파서는 0을 리턴하여 호출한 루틴에게 오류가 없이 파싱이 완료 되었음을 알린다.

그림 3 과 그림 4 및 표 1 과 표 2 에서 CIF 의 간단한 예와 스택의 동작 예를 보였다.

3. YACC 의 구성

1) 화일의 구조

YACC 화일은 declaration. translation rules, supporting C - routines 의 세부분으로 구성되어 있다.

```
declaration
%%
translation rules
%%
supporting C - routines
```

기호 "$ $"는 왼쪽의 비종단에 관련된 속성값이

표 2. 파싱 테이블
Table 2. The parsing table.

STATE	ACTION				GO TO			
	b	p	e	$	X	Y	W	Z
0	s4	s5			1	2	3	
1				acc				
2	s4	s5	s8				7	6
3	r2	r2	r2					
4	r4	r4	r4					
5	r5	r5	r5					
6				r1				
7	r3	r3	r3					
8				r6				

고, $i는 오른편 i번째의 문법심볼(종단 또는 비종단)에 관련되어지는 값이며, 두개의 production에 의한 YACC의 의미론적 행동표현은 다음과 같다.

```
expr : expr '+' term {$$ = $1 + $3 ; }
     | term
     ;
```

2) Ambiguity와 Conflict의 처리

일반 산술식이나 dangling if else 문과 같은 복합조건문들은 불분명한(ambiguous) 한 문법형태를 이룬다. 이것은 파서 생성시 shift/reduce나 reduce/reduce conflict을 일으킨다. 이런 문제를 해결하기 위해 YACC에서는 다음과 같은 방식을 취한다.

첫째로 shift/reduce conflict시 default action으로 shift 하고, reduce/reduce conflict시 이전의 규칙에 따라 reduce 한다.

둘째로 연산자 실행시 %left, %right의 reserved word를 사용하여 associativity와 reduce의 우선순위를 정할 수 있게 한다.

3) 오류의 처리

파싱도중 문법오류가 발견되더라도 파싱을 중단하지 않고 끝까지 파싱위해 error라는 reserved word를 사용하는데, 이는 error가 규칙에 맞을 때까지 stack을 pop 한다. 또한 error 뒤에 특정한 token을 설정하면, 오류발생 시부터 그 토큰까지는 파싱을 하지 않고 건너뛴다.

Ⅳ. CIF와 DXF의 문법

1. CIF의 문법

Mead/Conway가 Niklaus Wirth의 표준 기호법으로 표현한 CIF의 구조는 다음과 같다. cifFile 은 command와 endCommand로 구성되고, command 는 실제 여러 그림의 종류를 다루는 primCommand 와 셀 정의를 위한 command 들로 이루어진다. 또 primCommand에 속하는 boxCommand, polygonCommand 등은 개개의 글자와 정수등 낮은 수준의 원소들로 이루어져 있다.[1]

CIF의 command를 YACC에 맞게 표현하기 위해서, 다음의 사항을 고려했다.

첫째, command는 무한히 많은 요소들을 포함할 수 있으므로 primCommand와 셀 정의를 위한 command들의 반복(recursion)으로 표시한다.

둘째, 파싱의 지속을 위해, 오류처리기호를 command와 primCommand 사이에 넣는다.

```
lprimCommand :
        | lprimCommand primCommand
        | lprimCommand error ';'
        ;
```

셋째, 토큰으로는 SYSMNAME, INTEGER, COMMENT, SEP으로 정하고, LEX 루틴에서 숫자일때 INTEGER, 소문자 및 구성글자가 아닌 것은 한 글자를 리턴하게 하였으며, 전체적인 YACC표현은 부록에 참가하였다.

2. DXF의 문법

DXF의 전체적인 화일구조는 header section, table section, blocks section, entities section, end of file의 다섯 section으로 구성되어 있다. 그리고 이들 각각은 여러 그룹 코드들로 나누어져 있다. Header section은 그림에 관한 일반적인 정보가 수록되어 있고 table section은 선 형태, layer, style, view에 관한 테이블이 있다. 이에 따라 전체 dxfFile을 entitySection이 있는 경우와 blockSection과 entitySection이 있는 두가지 경우로 나눌 수 있다.

BlockSection은 시작과 끝을 알리는 정의 사이에 기본 단위 block 들의 좌 반복(left recursion) 으로 구성된 block으로 구성되게 기술 했다. 그리고 이 단위 block은, 토큰으로부터 좌표를 받아 도형의 정보를 갖고 있는 drawCommand (TRACE, LINE, INSERT 등) 의 좌 반복으로 이루어지는 drawings 만을 포함하고 있게 된다.[6]

3. Primitive CIF

Primitive CIF는 내부에 셀을 포함하여 계층적인 구조를 이루는 CIF를 개개의 요소로 분할한 것이다. 본 논문에서는 Box command 만을 사용하여 요소들을 나타내었다. 이 Box command는 실제 자신이 위치할 좌표와 방향을 가지고 있다.

V. CIF와 DXF 상호변환을 위한 행동설정

CIF의 DXF 변환은 이미 설정했던 규칙이 감축될 때, 필요한 부분에서 행하면 된다. CIF와 DXF 의 데이타 형태는 표3과 같다.

표 3. CIF와 DXF 화일의 데이타 형태
Table 3. Data format of CIF and DXF file.

CIF	DXF
DS	BLOCK
C	INSERT
B	TRACE
P	LINE
MX	X scale of INSERT
MY	Y scale of INSERT
R	angle of INSERT

1. CIF의 primitive CIF 변환

CIF command를 처리하는데 있어서 action을 BOX와 POLYGON을 정의하고 나머지는 경고 메세지를 내도록 하였고, 표준 CIF를 실제 자신의 위치와 방향을 갖는 primitive CIF로 바꾸기 위해 CIF와 유사한 C 코드를 생성시켰다. 이를 수행시 실제 layout 되는 부분은 main( ) 루틴에 넣고, 셀 정의 함수들은 그들의 모임으로 다른 화일에 기록하도록 했다. 또한 많은 양의 정보를 처리하기위해 CIF의 셀이 변형된 function들의 집합을 여러개로 나누어 따로 컴파일하여 나중에 link 하여 쓸수 있게 하였다.

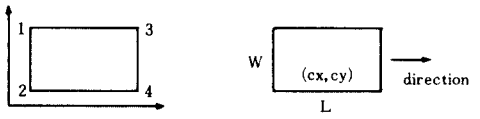
이를 위해서는 적절하게 셀 함수들이 있는 화일을 나누고 각 화일내에 필요한 변수들에 대해 extern 선언을 해 주었다.[7] 그림5는 CIF 및 DXF 데이타 형태를 나타내고, 그림6은 CIF의 primitive DXF 변환 흐름도를 나타낸 것이다.

2. 계층적인 구조를 갖는 DXF로의 변환

DXF는 계층적인 데이타 구조를 갖고 있어 CIF와 유사하게 변환 시킬수 있다. Call 명령어의 DXF 변환도 거의 유사하나 단지 mirror의 명령어가 DXF에는 없어 INSERT의 X, Y 스케일에서 음의부호를 취하여 각 축에 대칭되도록 했다.(그림7) 이것은 정보량이 적어지고 컴파일 시간을 없앰으로 빠른 플롯팅을 할 수 있다. 또 CIF에서 정의된 셀을 구분하여 볼 수 있다.

3. DXF의 CIF 변환

DXF의 CIF 변환은 서로 모든 셀의 정보를 미리



DXF의 기록 형태순서　　CIF의 형태

```
  0
TRACE        B length width centerx contery dirx diry;
  2
LAYER
 10
x of 1
 20
y of 1
 11
x of 2
 21
y of 2
  :
 13
x of 4
 23
y of 4
  2
LAYER
```

그림5. CIF와 DXF 형태
Fig. 5. Format of CIF and DXF.

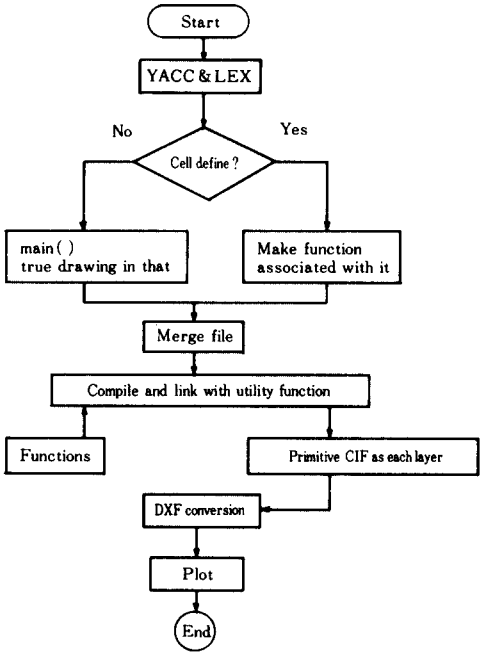


그림6. CIF의 요소 DXF 변환 흐름도
Fig. 6. CIF to primitive DXF conversion flowchart.

알고 있어야 하는 것을 제외하고 DXF 변환시의 경우와 유사하다. 행동 설정은 다음과 같다.

1) filter function으로 테이블 section을 제거

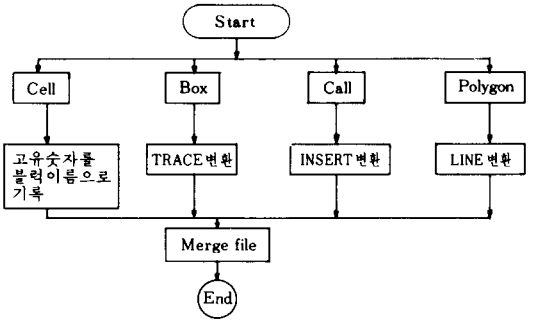


그림 7. CIF의 계층적 구조를 갖는 DXF 변환
Fig. 7. CIF to DXF conversion hierarchically.

했다.

2) 먼저 전체적으로 스캔하여 앞의 셀 이름과 셀 번호를 해쉬 테이블에 등록시킨다. 그림 8은 해쉬테이블을 위한 데이타 구조이다.

```
struct celllist {
        char    *blockname ;
        int     cellnum ;
        POINT   insp ;
        struct  celllist   *next ;
       }
```

그림 8. 해쉬테이블을 위한 데이타 구조
Fig. 8. Data structure of hash table.

3) 저장된 도형을 DXF로 변환시킨다.

4) DXF와 CIF의 matching

a) TRACE : Box

TRACE의 좌표로 먼저 Box의 방향을 검사한 후 이에 따라 변환 하였다.

b) INSERT : C

DXF에서 셀은 기준 좌표를 가지고 있으므로 기준을 zero 포인트로 갖고 있는 CIF로 변환하기 위해서는 다음 연산이 필요하다.

cell 좌표=INSERT 좌표−INSERT base 좌표

c) ROTATE

맨하탄한 경우만 취급 했으므로 angle 을 direction으로 변환하였다. 그림 9에 나타내었다.

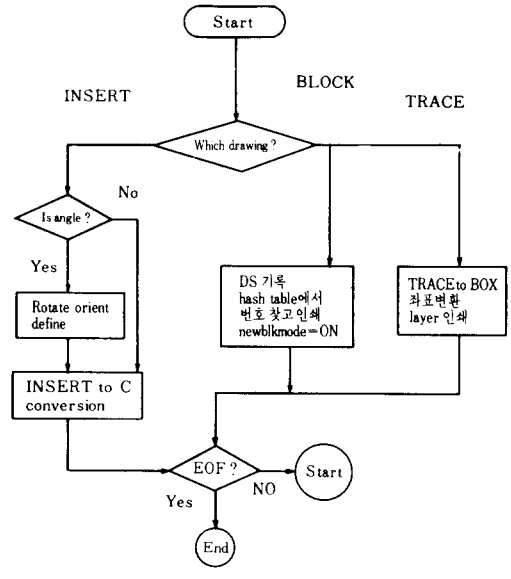


그림 9. 형태 변환
Fig. 9. Format conversion.

d) layer가 전과 같으면 프린트 하지않는다. 그러나 새로운 셀에 집어들면 프린트한다.

e) 파서가 동작하면서, 규칙이 인식될 때마다 4)에서 지정한 행동을 하여 순차적으로 CIF로 변환시킨다. 만일 오류가 검출되면 그 라인을 출력하고 다음 문장으로 넘어간다.

그림 10은 CIF 변환의 흐름도이다.

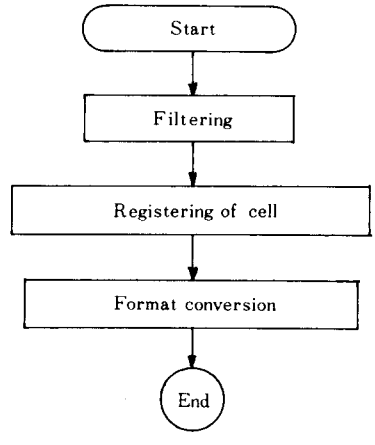


그림10. DXF to CIF 전체 흐름도
Fig. 10. Total flowchart of DXF to CIF.

Ⅵ. 고　　찰

프로그램이 하는 일은 크게 세가지로 분류될 수 있으나 YACC를 사용하므로 파서 제작이 매우 용이했고, 문법의 규칙이 일단 완성된 후에는 행동설정만 하면 되므로 개발시간을 크게 단축할 수 있었다. 또한 어떠한 경우의 문법적 오류라도 쉽게 추출해 낼 수 있었다. 본 논문에서 개발한 프로그램의 개요도는 그림 11와 같다.

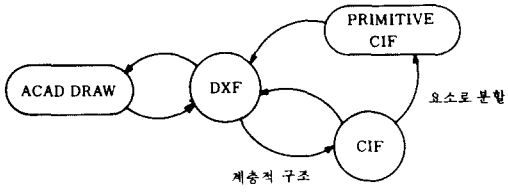


그림11. 전체 프로그램 개요도
Fig. 11. Total program block diagram.

Ⅶ. 결　　론

PC 상에서 CIF 입력 화일로 집적회로를 레이아웃할 수 있도록 하고, AUTOCAD의 도형에 대한 interchange form인 DXF 출력을 CIF로 변환하는 프로그램을 개발 하였다. 이를 수행시 YACC를 사용하여 다른 형태로의 변환 및 문법의 확장이 매우 용이하고 문법인식의 정당성을 쉽게 입증 할 수 있었다. 그리고 문법의 확장의 용이성과 복잡하고 일반성을 띄는 입력을 갖는 프로그램 개발시 효과적으로 사용할 수 있음을 보였다. Primitive CIF를 출력시킴으로 DRC(design rule check)나 ERC (electrical rule check)에 이용 할 수 있도록 프로그램을 개발하였다. 본 논문의 CIF-DXF 상호 변환프로그램의 개발로 LSI 급 설계를 비싼 workstation 이 없이도 무난히 해결할 수 있다. 앞으로 이러한 기법을 이용한 DRC, ERC, Circuit Extractor 등의 연구가 기대된다.

부　　록

부록1. CIF의 문법

```
%start    cifFile
%token    INTEGER  SEP  COMMENT  SYMBNAME  BBOX
%union    {
            long  Lval ;
            struct {long x, y ; } pval ;
            char svall20] ;
          }

%type 〈Lval〉 integer slnteger INTEGER layerName
%type 〈pval〉 point path
%type 〈sval〉 SYMBNAME

cifFile        : /* empty */
               | emptyOrSEP lcommand endCommand emptyOrSEP
               ;

lcommand       : /* empty */
               | lcommand command
               | lcommand error ':'
               ;

command        : primCommand
               | defDeleteCommand
               | defStartCommand lprimCommand defFinishedCommand
               ;
```

```
lprimCommand : /* empty */
             | lprimCommand primCommand
             | lprimCommand error ':'
             ;
primCommand  : /* empty */
             | polygonCommand
             | boxCommand
             | roundFlashCommand
             | wireCommand
             | callCommand
```

부록 2. DXF의 문법

```
%start   dxfFile
%token   DOUBLE  SEP     BLKNAME  END      BLOCK     BLOCKS  ENDBLK
%token   LINE    TRACE   INSERT   ENDSEC   ENTITIES  LAYER   PTVAL
%token   OTEXT   LNAME   SECTION  IVAL     COUNT     XPOSN   YPOSN
%token   APOSN   NAME
%union   {
           int    ival ;
           double dval ;
           char   sval[30] ;
           struct {double x, y ;} pval ;
         }
%type <dval>  BOUBLE  xvalue  yvalue
%type <ival>  XPOSN   YPOSN
%type <sval>  BLKNAME NAME
%type <pval>  cPoint
%%
dxfFile     : /* empty */
            | entitySection endCommand empty
            | blockSection  entitySection endCommand empty
            ;
blockSection  : blockSecDef blocks endSecDef
              ;
entitySection : entitySecDef drawIngs endSecDef
              ;
endCommand    : sePertr END
              ;
blocks        : /* empty */
              | blocks block
              | blocks error sePertr
              ;
block         : Fblock Sblock
              ;
Fblock        : blkDlc setLayer gCode BLKNAME unknown cPoint
              ;
```

參 考 文 獻

[1] C. Mead and L. Conway, "Introduction to VLSI system," Addison Wesley, pp. 98-127, 1980.

[2] 차균현 외 3인, "YACC를 이용한 레이아웃 정보 형태변환," 전자공학회 창립 40주년 기념 학술발표회 논문집, vol. 9. no. 2, pp. 861-865, 1986.

[3] Alfred V. Aho and Jeffrey D. Ullman, "Principle of complier design," Addison Wesley, pp. 146-190, 1977.

[4] Programmer's manual for UNIX system III, vol. 2B. oct. section 21.27 1981.

[5] Zenix software development system, "Software development guide," ZEUS computer

[6] Brian W. Kernighan, Dannis M. Ritchie, "The C programming language," Prentice Hall 1978.

[7] J.D. Ullman, "Computational aspect of VLSI," computer science, pp. 244-268, 1984.

[8] "The AUTOCAD drafting package user guide," ZEUS computer

[9] Xenix software development system, "Programmer's guide to library function C compiler reference," ZEUS computer

[10] Alfred V Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers," Addison wesley, pp. 215-219, pp. 258-263.

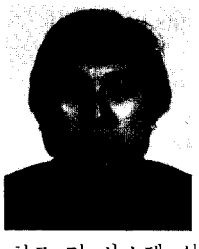
[11] 김영택, "컴파일러 구성론," 희중당, pp75-78, 1986. *

著 者 紹 介

白 寅 天(正會員)

1954年 1月 3日生. 1985年 2月 고려대학교 전자전산공학과 졸업. 1987年 2月 고려대학교 공학석사 학위 취득. 1987年 3月 고려대학교 대학원 전자공학과 박사과정 입학. 현재 군복무중. 주관심분야는 GAD tool 개발등임.

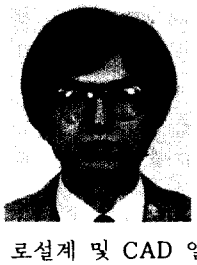
●



朴 魯 京(正會員)

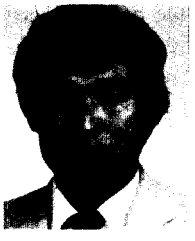
1958年 1月 7日生. 1984月 2月 고려대학교 전자전산공학과 졸업. 1986年 2月 고려대학교 공학석사 학위 취득. 1986年 3月～현재 고려대학교 대학원 전자공학과 박사과정. 주관심분야는 ISDN분야, 회로 및 시스템 설계등임.

●



全 興 雨(正會員)

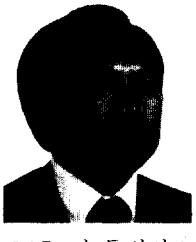
1956年 10月 30日生. 1980年 2月 한국항공대학 전자공학과 졸업. 1982年 2月 고려대학교 대학원 전자공학과 석사학위 취득. 1988年 8月 고려대학교 공학박사학위 취득. 주관심분야는 VLSI 회로설계 및 CAD 알고리듬 개발등임.



文 大 哲(正會員)

1955年 6月 7日生. 1979年 2月 숭전대학교 전자공학과 졸업. 1981年 8月 고려대학교 대학원 전자공학과 졸업. 1981年 9月 고려대학교 대학원 전자공학과 박사과정 입학. 1988年 2月 고려대학교 공학박사 학위 취득. 1984年 3月～현재 호서대학 통신공학과 조교수. 주관심분야는 회로 및 시스템 설계, CAD 개발등임.

●



車 均 鉉(正會員)

1939年 3月 26日生. 1965年 서울대학교 공학사. 1967年 미국 일리노이 대학교 공학석사학위 취득. 1976年 서울대학교 공학박사학위 취득. 1978年～현재 고려대학교 전자전산공학과 교수. 주관심분야는 CAD 및 통신시스템등임.