BoxBld, Boxes, ABoxes, CIFVal, CIFPlot

Geometric Composition Tools

Dr.G.K. Egan
Digital Electronics and Computing Systems

1982

Department of Communication and Electronic Engineering
Royal Melbourne Institute of Technology

BOXBLD

BoxBld allows designers to digitise using a digitising tablet, full-custom leaf-cells, gate-array metal, or any other manhattan geometry artwork. Some design rule checks are performed for the NMOS process and AWM 2600 gate-arrays. The program is interactive and the user is prompted for information in an easily understood manner. BoxBld generates Pascal procedures which describe the digitised structures. These procedures are compatable with the Boxes tool described below.

Running BoxBld

On the Z80 systems:

```
BoxBld
```

On the Unison systems:

```
boxbld
```

BOXES

Boxes is a Pascal-based geometric composition tool, command compatible wit the CSIRO Belle program. This documentation is derived from the Belle documentation written by Rob Clarke of CSIRO and assumes familiarity with the Pascal language.

Boxes can be used to define simple geometries as 'symbols' which may be iteratively placed using the Pascal control structures, and linked with a 'small' amount of random wiring. It may be used to specify metal for semi-custom technologies and printed circuit boards but it is not well suited to these applications.

How to Use Boxes

To use Boxes first generate your geometry description in Pascal using the Boxes primitives described in the following section. This consists of the 'block' of the usercode procedure to be imbedded in Boxes. e.g.

```
(* chequer board on metal layer *)
VAR
    size,i,j : INTEGER;
PROCEDURE chequer(size : INTEGER);
BEGIN
  define('chequer          ');
    boundingbox(0,0,size*2,size*2);
    layer(metal);
    box(0,0,size,size);
    box(size,size,size*2,size*2);
  enddef
END (* chequer *);
BEGIN
  importsymbols;
  writeln(output,'size of chequer=');
  readln(size);
  chequer(size);
```

of the layer names specified above.

```
PROCEDURE box(xll,yll,xur,yur:INTEGER);

parameters:
  xll,yll      co-ordinates of one corner (usually lower-left).
  xur,yur      co-ordinates of opposite corner (usually upper right).
action:
  draws a rectangle on the current layer between the corners given.
example:
  box(0,0,20,10);
special features:
  none.


PROCEDURE flash(diam:INTEGER;x,y:REAL);

parameters:
  diam                  diameter of circle.
  x,y          position of centre of circle.
action:
  draw a round flash (circle) on the current layer.
example:
  flash(4,20,20);
special features:
  none.


PROCEDURE wire(width:INTEGER; x,y:REAL);

parameters:
  width                 wire width in lambda's
  x,y                   co-ordinates of the starting point of the wire
                        ( in lambda's )
action:
  starts the definition of a wire.
example:
  wire(3,1.5,1.5); x(21.5); y(98.5);
special features:
  may be followed by any of x,y,xy,dx,dy,dxy (q.v.)
warning:
  the wire outline MUST fall on lambda boundaries.


PROCEDURE x(nx : REAL);

parameters:
  nx           x co-ordinate of the end of the new segment
action:
  a horizontal segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); x(21.5); y(98.5);
special features:
  may only be used after a 'wire'.


PROCEDURE y(ny : REAL);
```

```
parameters:
  ny          y co-ordinate of the end of the new segment
action:
  a vertical segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); x(21.5); y(98.5);
special features:
  may only be used after a 'wire'.


PROCEDURE xy(nx,ny : REAL);

parameters:
  nx,ny                co-ordinates of the end of the new segment
action:
  a new segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); xy(21.5,21.5); y(98.5);
special features:
  may only be used after a 'wire'. The direction of the new
segment must be parallel to the x or y axis, or, if 'set45s' has
been called, at 45 degrees to the axes.


PROCEDURE dx(xd : INTEGER);

parameters:
  xd          length of the new segment
action:
  a horizontal segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); dx(20); dy(60);
special features:
  may only be used after a 'wire'.


PROCEDURE y(yd : INTEGER);

parameters:
  yd          length of the new segment
action:
  a vertical segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); x(21.5); dy(60);
special features:
  may only be used after a 'wire'.


PROCEDURE dxy(xd,yd : INTEGER);

parameters:
  xd,yd          x and y displacements of the end of the new segment
action:
  a new segment is added to the wire being defined.
example:
  wire(3,1.5,1.5); dxy(20,20); y(98.5);
special features:
  may only be used after a 'wire'. The direction of the new
segment must be parallel to the x or y axis or, if 'set45s' has
```

been called, at 45 degrees to the axes.

```
PROCEDURE nodelabel(name:names; x,y:REAL; nodelayer:layertype);

parameters:
  name                  str giving node label.
  x,y                   point to be labeled.
  nodelayer             layer node is on (metal,poly,diffusion)
action:
  defines a node for circuit extraction and simulation.
example:
  nodelabel('VDD              ',100,432,metal);
special features:
  none.


PROCEDURE comment(str:comments);

parameters:
  str               inserted in CIF file as a comment.
action:
  puts a comment in the CIF file.
example:
  comment('this is a RAM cell  ');
special features:
  none.
```

Utility Procedures

```
PROCEDURE setnoend;

parameters:
  none.
action:
  suppresses the generation of an end statement on the CIF output
file.  This is useful for library files which are to be joined to
the front of other CIF files.
example:
  setnoend;
special features:
  none.


PROCEDURE setsymmo (nextsym : positiveinteger);

parameters:
  nextsym      symbol number of next symbol defined.
action:
  Sets the CIF symbol number of the next symbol defined.  The new
number must be higher than the last symbol number defined, if any
geometry has been defined, or if a header file has been read (see
importsymbols).
example:
  setsymno(5000);
special features:
  none.
```

```
FUNCTION existing(symbolname : names):boolean;

parameters:
  symbolname      symbol to be tested
action:
  tests  whether a symbol has already been defined.  Returns TRUE
if so, FALSE if not.
example:
  IF NOT existing('pullup32        ') THEN defpullup32;
special features:
  none.


PROCEDURE boundingbox (VAR xmin,ymin,xmax,ymax : INTEGER);

parameters:
  symbolname      symbol for which bounding box is specified
  xmin,ymin    lower left corner of bounding box
  xmax,ymax    upper right corner of bounding box
action:
  specifies what the designer 'thinks' is the bounding box of the
symbol being 'defined'.
example:
  boundingbox(0,100,2100,3110);
special features:
  may only be used immediately after a 'define'.


PROCEDURE importsymbols ;

parameters:
  none.
action:
  reads in a library of pre-defined symbols.
example:
  importsymbols;
special features:
  all  calls to importsymbols must precede the generation of  any
geometry.  It is the user's responsibility to avoid conflicts  in
symbol names. Conflicts in symbol names are flagged as errors.
```

Built in Geometry

These procedures provide useful small structures.

```
PROCEDURE polycut(x,y:INTEGER);
PROCEDURE mp(x,y:INTEGER);

parameters:
  x,y      position of centre of cut
action:
  places  a  contact  cut between metal and  polysilicon  at  the
position specified.
example:
  polycut(100,50);
    or
```

```
  mp(100,50);
special features:
  polycut and mp are identical apart from name.  The two routines
provide for those who wish to be cryptic or verbose.


PROCEDURE diffcut(x,y:INTEGER);
PROCEDURE md(x,y:INTEGER);

parameters:
  x,y      position of centre of cut
action:
  places  a  contact  cut  between metal  and  diffusion  at  the
position specified.
example:
  diffcut(100,50);
    or
  md(100,50);
special features:
  diffcut and md are identical apart from name.  The two routines
provide for those who wish to be cryptic or verbose.


PROCEDURE buttcontact(x,y:INTEGER; angle:integer);
PROCEDURE dp(x,y:INTEGER; angle:integer);

parameters:
   x,y      position of centre of contact
   angle            direction of the polysilicon end of the contact
action:
  places  a butting contact between diffusion and polysilicon  at
the position specified. The edges of the poly and diffusion are 3
lambda to the right and left respectivly of the origin.
example:
  buttcontact(100,50,90);
    or
  dp(100,50,90);
special features:
  buttcontact  and  dp are identical apart  from  name.  The  two
routines provide for those who wish to be cryptic or verbose.
```

ABOXES

ABoxes is a variant of Boxes tailored for the AWM 2600 gate-array. All the procedures of Boxes except Box, Flash and Rot are used in ABoxes. It is not necessary to include Layer commands (default is metal) and the wire width parameter should not be used in the Wire procedure (default wire width is 6 lambda). All REAL parameters in Boxes are constrained to INTEGER parameters in ABoxes.

Running ABoxes

On the Z80 systems:

```
Submit ABoxes,aboxfile
```

On the Unison systems:

aboxes aboxfile

Additional ABoxes Primitives

```
PROCEDURE wirewidth(w: INTEGER);

parameters:
  w        wire width
action:
  changes wire width from default of 6 lambda
example:
  wirewidth(12);
special features:
  remains set until next wirewidth command
```

CIFVAL

CIFVal accepts CIF files and checks them for conformance to the published CIF language syntax. It also generates *.pic files for CIFPlot the CIF check plotter. CIFVal performs some design rule checks for the (N)MOS process.

Running CIFVal

On the Z80 systems:

```
CIFVAL
```

On the Unison systems:

```
cifval
```

CIFPLOT

CIFPlot is a check plotter for CIF files it displays artwork expressed in CIF on Tektronix 40xx, Visual550, Vectrix displays and Calcomp and Servogor multi-pen plotters.

Running CIFPlot

On the Z80 systems:

```
CIFPLOT
```

On the Unison systems:

```
vlsimap
cifplot
```

On the Cyber:

```
get,procfil/un=rcoci
-cifcal,,ciffile
```

aboxes aboxfile

Additional ABoxes Primitives

```
PROCEDURE wirewidth(w: INTEGER);

parameters:
  w        wire width
action:
  changes wire width from default of 6 lambda
example:
  wirewidth(12);
special features:
  remains set until next wirewidth command
```

CIFVAL

CIFVal accepts CIF files and checks them for conformance to the published CIF language syntax. It also generates *.pic files for CIFPlot the CIF check plotter. CIFVal performs some design rule checks for the (N)MOS process.

Running CIFVal

On the Z80 systems:

```
CIFVAL
```

On the Unison systems:

```
cifval
```

CIFPlot

CIFPlot is a check plotter for CIF files it displays artwork expressed in CIF on Tektronix 40xx, Visual550, Vectrix displays and Calcomp and Servogor multi-pen plotters.

Running CIFPlot

On the Z80 systems:

```
CIFPLOT
```

On the Unison systems:

```
vlsimap
cifplot
```

On the Cyber:

```
get,procfil/un=rcoci
-cifcal,,ciffile
```

Examples of Boxes Useage

Dr. R. Clarke
CSIRO VLSI Program

Example 1: A simple transistor

No parameterization is done for this symbol.

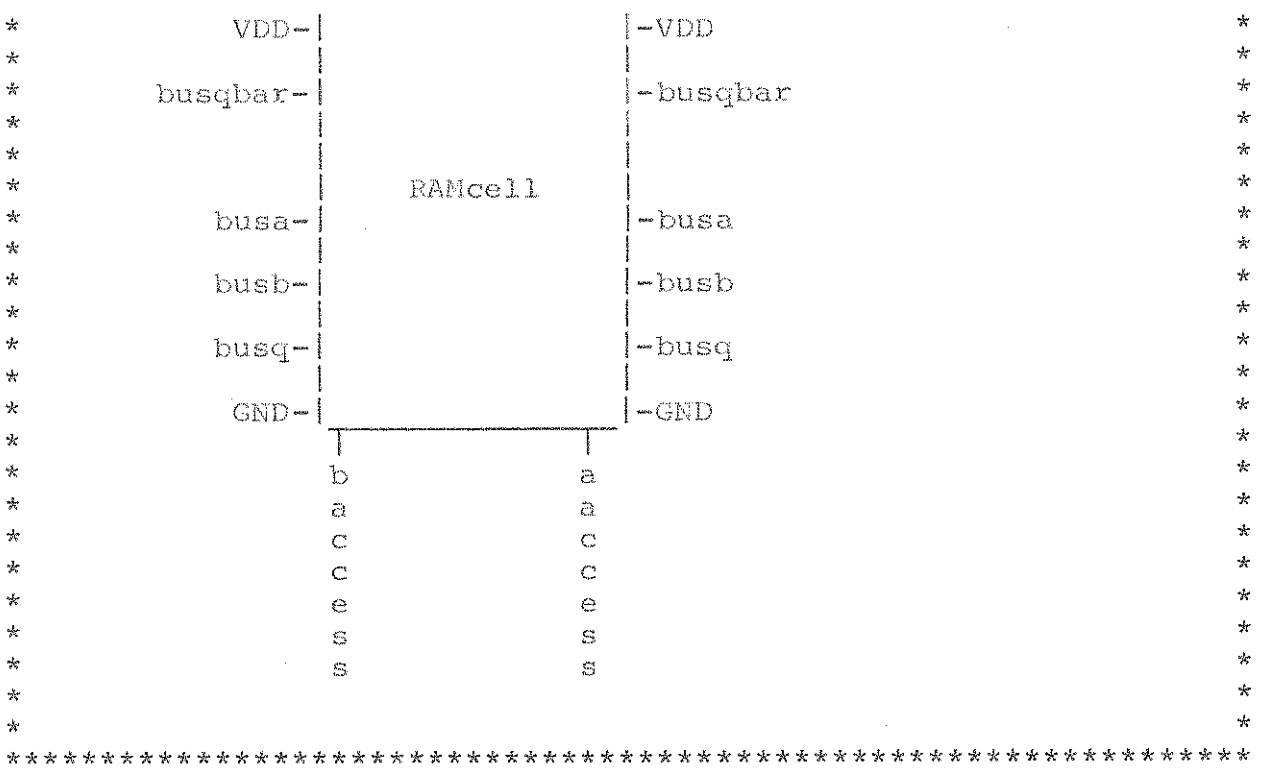
```
(* routine to generate a simple enhancement mode transistor
  with length to width ratio = 1/2 *)
PROCEDURE enh;
BEGIN
  define('enh');
    layer (diffusion);
    box(2, 0, 6, 6);
    layer (poly);
    box(0, 2, 8, 4);
  enddef;
END; (* enh *)
BEGIN
  enh; (* run the procedure to generate the geometry *)
  draw ('enh', 0, 0);
END;
```

Example 2: A parameterised NOR gate generator

```
(* parameterisable nor gate *)
(* variables:
      height  -       height of cell
      pdwidth -       pull down width
      pulen   -       length of pulldown
      n       -       number of inputs
*)

VAR
  height:        23..99;
  pdwidth:       2..99;
  pulen:         2..99;
  n:             1..99;
  bitwidth:      integer;
  i,x1,y1,x2,y2:integer;

BEGIN
  (* accept variables *)
  write('height of cell [23..99]?.............. ');
  readln(height);
  write('width of pulldown transistors [2..99]?.. ');
  readln(pdwidth);
  write('length of pullup transistor [2..99]?.... ');
  readln(pulen);
  write('number of transistors [1..99]?......... ');
  readln(n);
  setsymno(10000);
  IF (pulen > height) - 19 THEN
    writeln('pullup is too long for current cell height');
  bitwidth := 7 + pdwidth;
  define ('nor             ');
    (* pullup *)
    buttcontact (4, 11, 180);
    diffcut(5, height-2);
    layer (diffusion);
      box( 4, 13, 6, height-4);
```



The minimum size of the two port cell with pass through is 36 lambda (in the x direction), by 49 lambda (in the y direction). The cell is designed to allow sharing of the vdd and gnd wires, to reduce the size of ram arrays produced with the cell. To create a ram array using this cell, the cells should be stacked vertically in the following manner:

every alternate cell should be reflected in its x axis,

each cell should then overlap the previous cell by pwrwidth = width of the power supply busses,

When stacking the cells in the x direction, no overlap is allowable and no reflections are necessary (but are possible), the length of the pulldown transistors and the width of the pullup transistors is always 2 lambda.

Suitable parameter values resulting in a mimimal size two port ram cell with pass through wire are:

ports	=	2	;	number of ports to the ram cell (1 or 2),
passthrough	=	pass	;	b bus required (pass) or not (nopass).
pwrwidth	=	4	;	width of vdd and ground wires
cellwidth	=	36	;	x dimension of the cell
pullup1length	=	2 to 12	;	channel length of the first pullup.
pullup2length	=	2 to 7	;	channel length of the second

```
                                        pullup.

pulldn1width    =       6 to 20 ;  channel width of  the  first
pulldown.

pulldn2width    =       5 to 22 ;  channel width of  the  second
                                   pulldown.

busby           =       7       ;  y dimension of the bottom of bus
                                   b.

busqy           =       14      ;  y dimension of the bottom of bus
                                   q.

busay           =       20      ;  y dimension of the bottom of bus
                                   a.
busqbary        =       39      ;  y dimension of the bottom of bus
                                   qbar.

VDDy            =       45      ; y dimension of the bottom of the
                                  VDDwire.
```

This results in a cell with overall dimensions 36 lambda in the x direction by 49 lambda in the y direction. The effective size of the cell is smaller than this, (36 by 45 lambda), due to the sharing of vdd and ground wires in the ram array. These are minimum values. Any attempt to specify values smaller than this will result in an error message and no code will be generated. Larger values may be specified, but the same minimum spacing between y coordinate parameters must be maintained.

The b bus wire is only a pass through and can be omitted by specifying passthrough = nopass. In this case the y coordinate parameters of wires above the b bus may be reduced by 6 lambda. Note that this will effect the maximum pulldown transistor width for the minimum sized cell by the same amount.

The a bus (read only) may be omitted if ports = 1. Again, the y coordinate parameters of all wires above the a bus can be reduced by 6 lambda. The maximum pulldown transistor width for the minimim size cell will again be reduced by the same amount.

If both the b bus and the a bus are omitted, the y coordinate parameters may be reduced by 12 lambda, as will the maximum pulldown transitor width for the minimum sized cell.

********** dual port RAM with no pass through **********
Suitable parameter values resulting in a mimimal size two port ram cell with no pass through wire are:

```
ports           =       2       ;  number of ports to the RAM cell
                                   (1 or 2).

passthrough     =       nopass  ;  b bus required (pass) or not
                                   (nopass).

pwrwidth        =       4       ;  width of VDD and ground wires

cellwidth       =       36      ;  x dimension of the cell
```

```
pullup1length   =       2 to 12 ; channel length of the first pullup.

pullup2length   =       2 to 7  ; channel length of the second pullup.

pulldn1width    =       6 to 14 ; channel width of the first
                                  pulldown.

pulldn2width    =       5 to 16 ; channel width of the second
                                  pulldown.

busby           =       dont care; y dimension of the bottom of
                                   bus b.

busqy           =       8       ; y dimension of the bottom of
                                  bus q.

busay           =       14      ; y dimension of the bottom of
                                  bus a.

busqbary        =       33      ; y dimension of the bottom of
                                  bus qbar.

VDDy            =       39      ; y dimension of the bottom of
                                  the VDD wire.
```

This results in a cell with overall dimensions 36 lambda in the x direction by 43 lambda in the y direction. The effective size of the cell is smaller than this, (36 by 39 lambda), due to the sharing of vdd and ground wires in the ram array.

********** basic single port RAM **********
Suitable parameter values resulting in a mimimal size one port ram cell with no pass through wire are:

```
ports           =       1       ; number of ports to the RAM cell
                                  (1 or 2).

passthrough     =       nopass  ; b bus required (pass) or not
                                  (nopass).

pwrwidth        =       4       ; width of VDD and ground wires

cellwidth       =       31      ; x dimension of the cell

pullup1length   =       2 to 12 ; channel length of the first
                                  pullup.

pullup2length   =       2 to 7  ; channel length of the second
                                  pullup.

pulldn1width    =       6 to 8  ; channel width of the first
                                  pulldown.

pulldn2width    =       5 to 10 ; channel width of the second
                                  pulldown.

busby           =       dont care; y dimension of the bottom of
                                   bus b.
```

```
busqy           =       8       ; y dimension of the bottom of
                                  bus q.

busay           =       dont care; y dimension of the bottom of
                                   bus a.

busqbary        =       32      ; y dimension of the bottom of
                                  bus qbar.

VDDy            =       38      ; y dimension of the bottom of the
                                  VDD wire.
```

This results in a cell with overall dimensions 31 lambda in the x direction by 37 lambda in the y direction. The effective size of the cell is smaller than this, (31 by 33 lambda), due to the sharing of vdd and ground wires in the ram array.

```
*)
TYPE
  portsnumber   =       1..2;
  passtype      =       (pass, nopass);

VAR
  bigbuswidth, i, x, y: integer;

PROCEDURE RAMcell(cellname: names;
                  ports: portsnumber;
                  passthru: passtype;
                  pwrwidth, cellwidth,
                  pullup1length, pullup2length,
                  pulldn1width, pulldn2width:sizeunit;
                  busby,busqy,busay,busqbary,VDDy:cornerunit
                 );
(* design rules *)
(* given in terms of lambda *)
(* not all of these constants are used in this program *)
CONST
  diffwidth     =       2;
  diffspacing   =       3;
  polywidth     =       2;
  polyspacing   =       2;
  metalwidth    =       3;
  metalspacing  =       3;
  poldifspacing =       1;
  poldifoverlap =       2;
  imploverchann =       2;
  impltochann   =       2;
  cutsize       =       2;
  cutspacing    =       2;
  cuttoedge     =       1;

VAR
  rifty: integer;

BEGIN  (* RAMcell *)

(* ******** start of RAMcell description ********* *)
(* test to ensure that specified parameters will yield a valid
   RAM  cell *)
```

```
        BEGIN
          box(29,0,31,rifty + 1);
          box(31, rifty-1, 32, rifty+1);
          box(32, rifty-1, 34, VDDy+pwrwidth);
        END;
      (* diffusion *)
      layer(diffusion);
        (* RAMcell output to q bus *)
        box(4, busqy, 7, rifty + 3);
        (* inverse RAMcell output to qbar bus *)
        box(6, busqbary, 11, busqbary + metalwidth);
        box(11, 0, 13, busqbary + metalwidth);
        box(11, rifty-5 -(pulldn1width-6), 18, rifty+5);
        box(16, rifty+5, 18, VDDy + pwrwidth);
        box(21, 0, 23, rifty-2-(pulldn2width-5));
        box(21, rifty-2-(pulldn2width-5), 28, rifty-3);
        box(22, rifty-3, 28, rifty+5);
        box(22, rifty+5, 27, rifty+6);
        box(25, rifty+10, 27, VDDy + pwrwidth);
        (* RAMcell output to a bus *)
        IF ports = 2 THEN
          box(28, busay, 32, busay + 4);
      (* implant *)
      layer(implant);
        (* over pullup 1 *)
        box(14, rifty+4, 20, rifty + 6 + pullup1length + 2);
        (* over pullup 2 *)
        box(23, rifty+9, 29, rifty + 11 + pullup2length + 2);
  enddef;
END; (* RAMcell *)


(* PROCEDURE to generate a 4 bit vertical slice of the RAM array *)
PROCEDURE ramcolumn( x,y:cornerunit);
BEGIN
  draw('raml              ',x,y);
  draw('raml              ',x,y+94); my;
  draw('raml              ',x,y+90);
  draw('raml              ',x,y+184); my;
END; (* ramcolumn *)

BEGIN (* usercode *)
  setsymno (10300);
  bigbuswidth:= 40;
  x:= bigbuswidth + 3;
  y:= 0;
  RAMcell('raml             ',2,pass,4,36,6,6,10,10,7,14,20,39,45);
  define ('ramarray         ');
    for i := 0 to 15 do
      ramcolumn(x + i*36,y);
    layer (metal);
      box(0,0,bigbuswidth, 184);
      box(bigbuswidth, 45, bigbuswidth + 3, 49);
      box(bigbuswidth, 135, bigbuswidth + 3, 139);
  enddef; (* ramarray *)
  draw ('raml             ',0,0);
END; (* usercode *)
```

Basic (N)MOS Process CIF Cell Library
Documentation

Dr. R. Clarke
CSIRO VLSI Program

This section describes the documents the library cells as described by the CIF listings given in the back of Hon and Sequin's "A Guide to LSI Implementation", Xerox, 1980. These are the cells distributed for Australia's first multi-project chip, AusMPC 5/82.

A description of each cell is given, as is its size and the position of its connections (given in terms of lambda). The dimensions of all active devices is also listed, and can be used as basic information for circuit simulation.

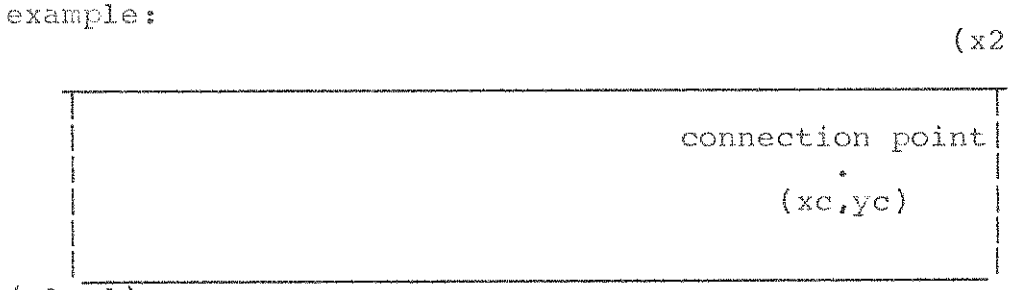
Notes:

1/ The circuits described here assume positive y coordinates for all cells, rather than negative as given in Hon.

2/ The symbol "T is used to represent the complement of a signal. That is, Iinput) is equal to the complement of input.

3/ The position of connections are specified to be at the center of a box which has edges the same length as the conductor width.

For example:



length= x2-x1
width = y2-y1
xc = x2-width/2
yc = y2-width/2

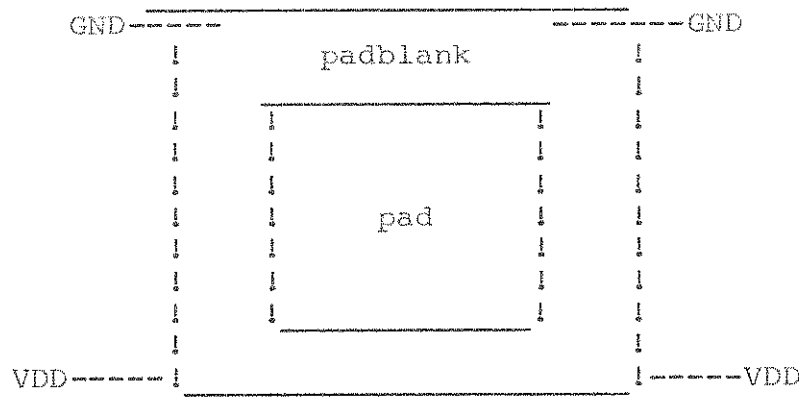
Connections are specified in this way so that wires connected to these points will overlap the conductors to the proper degree. However, all cells in the standard cell library are composed of boxes. They can therefore be abutted to other boxes with no overlap. The PLA cells are a good example of this. These cells abut to each other with no overlap (except in the case of placlockedout, placlockednorout to form programmable logic arrays.

Symbol 2
cell name
padblank
description
blank i/o pad used in constructing padout, padin, etc.
size (lambda)
1Ø6x1Ø6
connections (posn, width in lambda) position width layer
pad connection
(53,53) overglass hole is 46 x 46 lambda square.
inputs (to cell)
(possibly from pad)
outputs (from cell)
(possibly from pad)

power

VDD:	(4,4) (102,4)	8	metal
GND:	(16,102) (90,102)	8	metal

block diagram



Symbol 3

cell name

paddriver

description

output pad and enhancement mode driver transistors

size (lambda)

106x106

connections (posn, width in lambda)	position	width	layer
pad connection	(53,53)	46x46	metal
inputs (to cell)			
pupin (input to pullup gate)	(24,105) (82,105)	2	poly
pdnin (pulldown gate)	(34,105) (72,105)	2	poly
outputs (from cell)			
the pad			
power			
VDD:	(4,4) (102,4)	8	metal
GND:	(16,102) (90,102)	8	metal

additional information

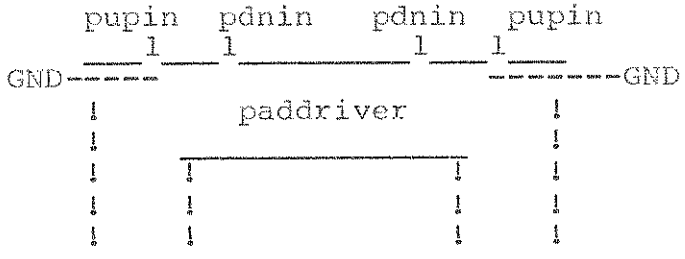
both transistors are enhancement mode devices.

length to width ratios

for pullup transistor 1/w= 1/158

for pulldown transistor 1/w= 1/138

block diagram



Symbol 4
cell name
padout
description
basic output pad to drive a ttl load.
size (lambda)
106x145

connections (posn, width in lambda)	position	width	layer
pad connection	(53,53)	46x46	metal
inputs (to cell)			
outsig (signal to be output):	(53,144)	2	poly
outputs (from cell)			
the pad			metal
power			
VDD:	(4,4) (102,4)	8	metal
GND:	(16,102) (90,102)	8	metal

additional information
transistor dimensions (l/w)
dual inverters driving superbuffers

```
mu1     dep     l/w= 2/1
md1     enh     l/w= 1/8        ;k=4
        mu2     dep     l/w= 2/1
md2     enh     l/w= 1/8        ;k=4
```

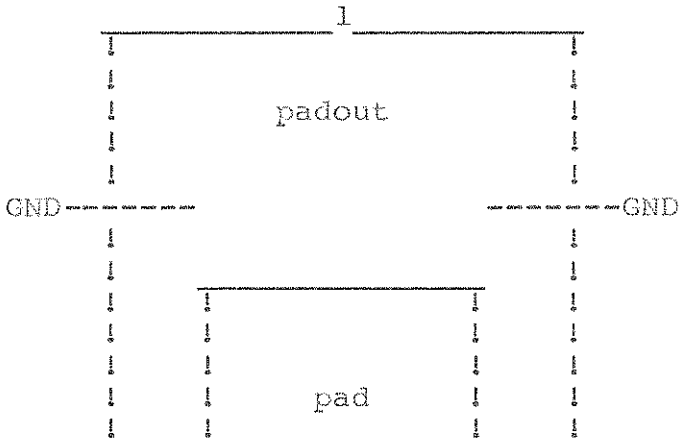
dual superbuffers driving big gates

```
mu3     dep     l/w= 1/6
md3     enh     l/w= 1/24       ;k=4
        mu4     dep     l/w= 1/6
        md4     enh     l/w= 1/24       ;k=4
```

big enhancement mode output driver transistors

```
mu5     enh     l/w= 1/158
md5     enh     l/w= 1/138
```

block diagram

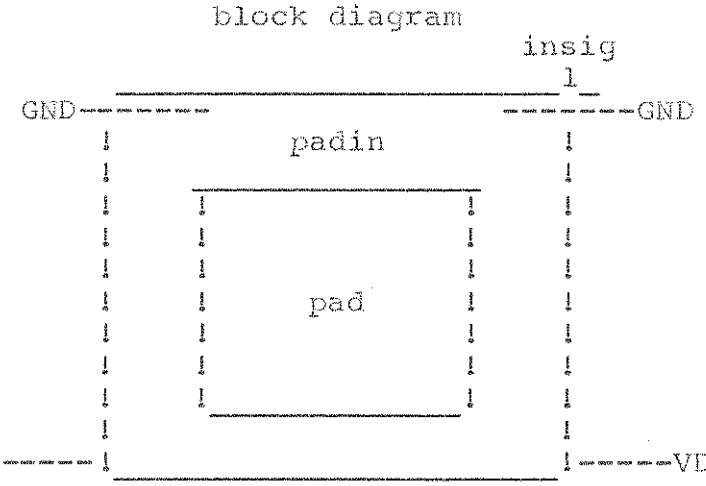


Symbol 5
cell name
padin
description
input pad with lightning protector.
used to connect outside world to k=8 logic.
size (lambda)
106x106

connections (posn, width in lambda)	position	width	layer
pad connection	(53,53)	46x46	metal
inputs (to cell)			
the pad			metal
outputs (from cell)			
insig (input signal to circuit)	(96,105)	2	diff
power			
VDD:	(4,4) (102,4)	8	metal
GND:	(16,102) (90,102)	8	metal

additional information
lightning protection is an enhancement mode transistor with both source and gate grounded.
l/w for this device is 1/67.5

block diagram

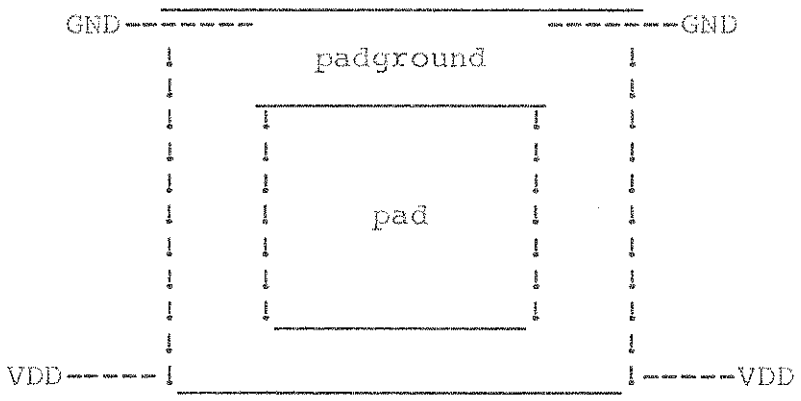


Symbol 6
cell name
padground
description
pad for connection to circuit ground.
size (lambda)
106x106

connections (posn, width in lambda)	position	width	layer
pad connection	(53,53)	46x46	metal
inputs (to cell)			

the pad (ground connection)			metal
outputs (from cell)			
the ground line GND.			metal
power			
VDD:	(4,4) (102,4)	8	metal
GND:	(16,102) (90,102)	8	metal

block diagram



Symbol 7

cell name

padVDD

description

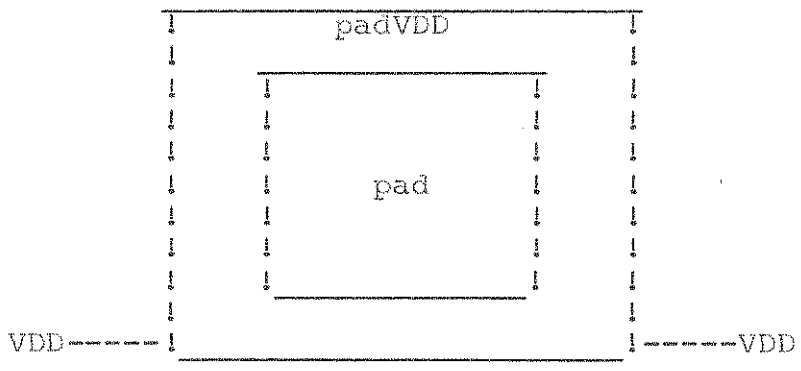
pad for connecting circuit's power supply.

size (lambda)

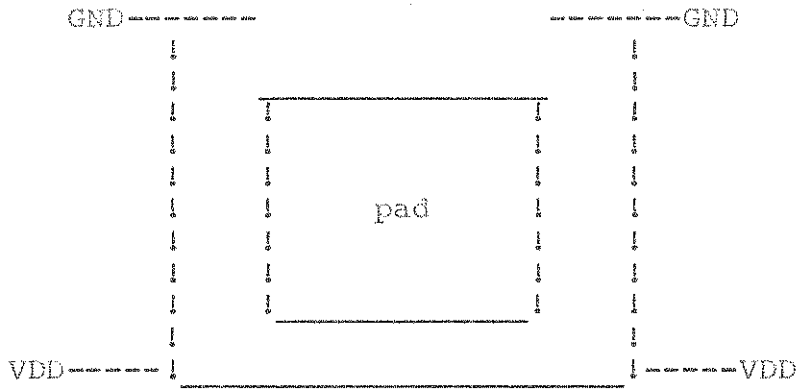
106x80

connections (posn, width in lambda)	position	width	layer
pad connection			
	(53,53)	46x46	metal
inputs (to cell)			
the pad (VDD connection)			metal
outputs (from cell)			
the VDD line			metal
power			
VDD:	(4,4) (102,4)	8	metal

block diagram



Symbol 8


GND
GND
pad
VDD
VDD

PLA Generator (PLAGen)
Documentation

Peter Maxwell

PLAGEN

PLAGen is a program which will generate a PLA in CIF code from a truth table description. The program is largely interactive, from the point of view of specifying options in the PLA generation. These user supplied options will be described shortly.

The general operation of PLAGEN is that it reads from a file plafile.PLA and creates the PLA artwork description in a file plafile.CIF were plafile is a user defined name. Plafile the input file to PLAGen is in the form of a truth table, preceded by a line which contains the number of inputs, product terms and outputs. for example, consider a PLA with 4 inputs, 5 product terms and 3 outputs. The input to PLAGEN might then be:

```
4 5 3
1Øx1 Ø1Ø
xx11 ØØx
11ØØ 1x1
1ØØx xØ1
ØØ1x ØØØ
```

A single space is required to separate the inputs from the outputs for each product term and also to separate the three fields on the first line. Generation of the PLA is then under the control of the following requested information:

symbol start number: The main symbol number of the PLA is given this value and all subsymbols are given numbers relative to this. This is needed for Boxes to avoid conflicting symbol numbers.

subsymbol generation: suppression of CIF code for the subsymbols may be achieved for cases where more than one PLA is being used, each using the same subsymbols.

pla programmed: Placement of programming cells may be ommitted for cases where only the overall size is required.

inputs/outputs: A number of obvious selections regarding clocked or unclocked inputs/outputs and whether a finite state machine is being generated.

outputs position: The "traditional" PLA has the outputs and inputs on the same side (necessary for finite state machine generation). This flag allows generation of a PLA with ouputs on the opposite side of the PLA to the inputs.

labels: Unless specifically ommitted all inputs and outputs are labelled. for a PLA with symbol number n specified, inputs are labelled as planinx where x ranges from Ø to (inputs-1), with 'Ø' referring to the left-most input (furthest from the or plane). Outputs are labelled as planoutx (e.g. pla1ØØout3) with x=Ø being the output closest to the and plane. The present implementation always labels phi1 and phi2 for clocked inputs/outputs, although this is readily changed. All labelling is done using the 94 extension of CIF.

The cells which PLAGEN uses to build up the PLA are similar but not the same as the standard Hon. and Sequin cells. Input and output drivers are the same but programming and other cells are not. For this reason the symbol names have been changed to avoid confusion with the other PLA cells. CIF code for all these cells is contained in the program. The main symbol for the PLA (which is always generated) has a symbol name of PLAn, where n is the symbol number specified by the user.