

Computer Aids for VLSI Design

Steven M. Rubin

Copyright © 1994

Appendix B: Caltech Intermediate Format

Caltech Intermediate Format (CIF) is a recent form for the description of integrated circuits. Created by the university community, CIF has provided a common database structure for the integration of many research tools. CIF provides a limited set of graphics primitives that are useful for describing the two-dimensional shapes on the different layers of a chip. The format allows hierarchical description, which makes the representation concise. In addition, it is a terse but human-readable text format. CIF is therefore a concise and powerful descriptive form for VLSI geometry.

Each statement in CIF consists of a keyword or letter followed by parameters and terminated with a semicolon. Spaces must separate the parameters but there are no restrictions on the number of statements per line or of the particular columns of any field. Comments can be inserted anywhere by enclosing them in parenthesis.

There are only a few CIF statements and they fall into one of two categories: geometry or control. The geometry statements are: LAYER to switch mask layers, BOX to draw a rectangle, WIRE to draw a path, ROUNDFLASH to draw a circle, POLYGON to draw an arbitrary figure, and CALL to draw a subroutine of other geometry statements. The control statements are DS to start the definition of a subroutine, DF to finish the definition of a subroutine, DD to delete the definition of subroutines, 0 through 9 to include additional user-specified information, and END to terminate a CIF file. All of these keywords are usually abbreviated to one or two letters that are unique.

B.1 Geometry

The `LAYER` statement (or the letter `L`) sets the mask layer to be used for all subsequent geometry until the next such statement. Following the `LAYER` keyword comes a single layer-name parameter. For example, the command:

```
L NC;
```

sets the layer to be the nMOS contact cut (see Fig. B.1 for some typical MOS layer names).

NM	nMOS metal
NP	nMOS polysilicon
ND	nMOS diffusion
NC	nMOS contact
NI	nMOS implant
NB	nMOS buried
NG	nMOS overglass
CMF	CMOS metal 1
CMS	CMOS metal 2
CPG	CMOS polysilicon
CAA	CMOS active
CSG	CMOS select
CWG	CMOS well
CC	CMOS contact
CVA	CMOS via
COG	CMOS overglass

FIGURE B.1 CIF layer names for MOS processes.

The `BOX` statement (or the letter `B`) is the most commonly used way of specifying geometry. It describes a rectangle by giving its length, width, center position, and an optional rotation. The format is as follows:

```
B length width xpos ypos [rotation] ;
```

Without the *rotation* field, the four numbers specify a box the center of which is at (*xpos*, *ypos*) and is *length* across in *x* and *width* tall in *y*. All numbers in CIF are integers that refer to centimicrons of distance, unless subroutine scaling is specified (described later). The optional *rotation* field contains two numbers that define a vector endpoint starting at the origin. The default value of this field is (1, 0), which is a right-pointing vector. Thus the *rotation* clause 10 5 defines a 30-degree counterclockwise rotation from the normal.

Similarly, $10 \ -10$ will rotate clockwise by 45 degrees. Note that the magnitude of this rotation vector has no meaning.

The **WIRE** statement (or the letter **w**) is used to construct a path that runs between a set of points. The path can have a nonzero width and has rounded corners. After the **WIRE** keyword comes the width value and then an arbitrary number of coordinate pairs that describe the endpoints. Figure B.2 shows a sample wire. Note that the endpoint and corner rounding are implicitly handled.

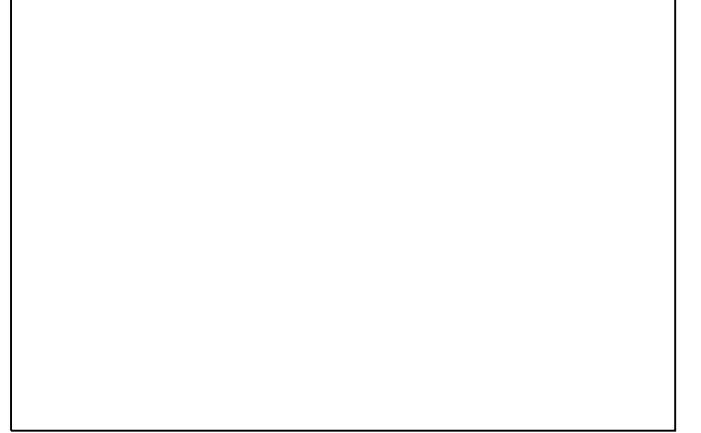


FIGURE B.2 A sample CIF "wire" statement. The statement is: `w25 100 200 100 100 200 200 300 200;`

The **ROUNDFLASH** statement (or the letter **R**) draws a filled circle, given the diameter and the center coordinate. For example, the statement:

`R 20 30 40;`
will draw a circle that has a radius of 10 (diameter of 20), centered at (30, 40).

The **POLYGON** statement (or the letter **P**) takes a series of coordinate pairs and draws a filled polygon from them. Since filled polygons must be closed, the first and last coordinate points are implicitly connected and need not be the same. Polygons can be arbitrarily complex, including concavity and self-intersection. Figure B.3 illustrates a polygon statement.

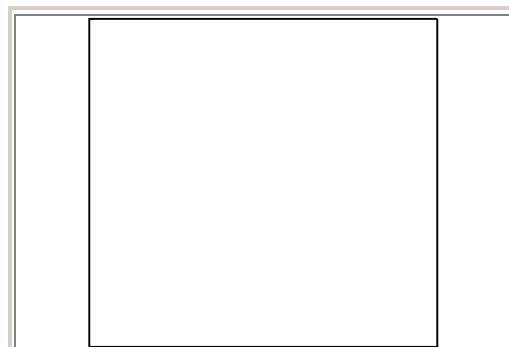


FIGURE B.3 A sample CIF "polygon" statement. The statement is: `P 150 100 200 200 200 300 100 300 100 200;`

B.2 Hierarchy

The `CALL` statement (or the letter `C`) invokes a collection of other statements that have been packaged with `DS` and `DF`. All subroutines are given numbers when they are defined and these numbers are used in the `CALL` to identify them. If, for example, a `LAYER` statement and a `BOX` statement are packaged into subroutine 4, then the statement:

```
C 4;  
will cause the box to be drawn on that layer.
```

In addition to simply invoking the subroutine, a `CALL` statement can include transformations to affect the geometry inside the subroutine. Three transformations can be applied to a subroutine in CIF: translation, rotation, and mirroring. Translation is specified as the letter `T` followed by an x, y offset. These offsets will be added to all coordinates in the subroutine, to translate its graphics across the mask. Rotation is specified as the letter `R` followed by an x, y vector endpoint that, much like the *rotation* clause in the `BOX` statement, defines a line to the origin. The unrotated line has the endpoint $(1, 0)$, which points to the right. Mirroring is available in two forms: `MX` to mirror in x and `MY` to mirror in y . Mirroring is a bit confusing, because `MX` causes a negation of the x coordinate, which effectively mirrors about the y axis!

Defining subroutines for use in a `CALL` statement is quite simple. The statements to be packaged are enclosed between `DS` (definition start) and `DF` (definition finish) statements. Arguments to the `DS` statement are the subroutine number and a subroutine scaling factor. There are no arguments to the `DF` statement. The scaling factor for a subroutine consists of a numerator followed by a denominator that will be applied to all values inside the subroutine. This scaling allows large numbers to be expressed with fewer digits and allows ease of rescaling a design. The scale factor cannot be changed for each invocation of the subroutine since it is applied to the definition. As an example, the subroutine of Fig. B.4 can be described formally as follows:

```
DS 10 20 2;  
    B10 20 5 5;  
    W1 5 5 10 15;  
    DF;
```

Note that the scale factor is $20/2$, which allows the trailing zero to be dropped from all values inside the subroutine.

Arbitrary depth of hierarchy is allowed in CIF subroutines. Forward references are allowed provided that a subroutine is defined before it is used. Thus the sequence:

```
DS 10;  
    ...  
    C 11;  
    DF;
```

```
DS 11;  
...  
DF;  
C 10;
```

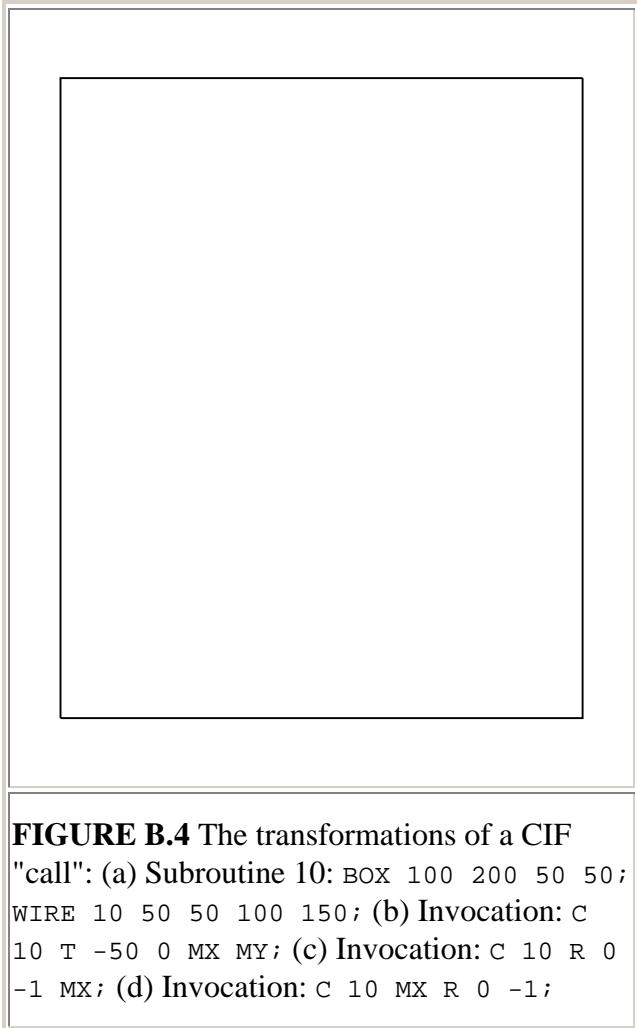
is legal, but the sequence:

```
C 11;  
DS 11;  
...  
DF;
```

is not. This is because the actual invocation of subroutine 11 does not occur until after its definition in the first example.

Any number of transformations can be applied to an object and their listed order is the sequence that will be used to apply them.

Figure B.4 shows some examples, illustrating the importance of ordering the transformations (notice that Figs. B.4c and B.4d produce different results by rearranging the transformations).



B.3 Control

CIF subroutines can be overwritten by deleting them and then redefining them. The `DD` statement (delete definition) takes a single parameter and deletes every subroutine that has a number greater than or equal to this value. The statement is useful when merging multiple CIF files because designs can be defined, invoked, and deleted without causing naming conflicts. However, it is not recommended for general use by CAD systems.

Extensions to CIF can be done with the numeric statements `0` through `9`. Although not officially part of CIF, certain conventions have evolved for the use of these extensions (see Fig. B.5).

<code>0 x y layer N name;</code>	Set named node on specified layer and position
<code>0V x1 y1 x2 y2 ... xn yn;</code>	Draw vectors
<code>2A "msg" T x y;</code>	Place message above specified location
<code>2B "msg" T x y;</code>	Place message below specified location
<code>2C "msg" T x y;</code>	Place message centered at specified location
<code>2L "msg" T x y;</code>	Place message left of specified location
<code>2R "msg" T x y;</code>	Place message right of specified location
<code>4A lowx lowy highx highy;</code>	Declare cell boundary
<code>4B instancename;</code>	Attach instance name to cell
<code>4N signalname x y;</code>	Labels a signal at a location
<code>9 cellname;</code>	Declare cell name
<code>91 instancename;</code>	Attach instance name to cell
<code>94 label x y;</code>	Place label in specified location
<code>95 label length width x y;</code>	Place label in specified area

FIGURE B.5 Typical user extensions to CIF.

The final statement in a CIF file is the `END` statement (or the letter `E`). It takes no parameters and typically does not include a semicolon.