GBLD: A Formal Model for Layout Description and Generation

I–Lun Tseng and Adam Postula
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane 4072, Australia
{iltseng, adam}@itee.uq.edu.au

Abstract

In this paper, we introduce a layout description and generation model, GBLD, based on the notions and elements of L-systems and context-free grammars. Our layout model is compatible with geometric layout formats, such as GDSII or CIF. However, it is more powerful and more concise. The layouts represented by GBLD are sizeable, parameterised, and can incorporate design rules. GBLD has the potential to be used as a format for analog layout templates, analog layout retargeting, as well as the final layout format.

1 Introduction

Analog integrated circuit design is more difficult than digital since many essential electrical properties are strongly dependent on the circuit parasitics that can be calculated accurately only after the detailed layout is complete. The traditional analog design flow involves repetitive and lengthy iterations of transistor sizing, electrical values extraction and performance evaluation. Design flows based on the procedural layout approach have been proposed [DL00, ODJ$^+$95] in order to reduce the design time. However, procedural layout methods [ODJ$^+$95, DGL99] require tedious programming work and the layout programs are usually difficult to verify and debug. The layouts generated by the programs might be erroneous if the larger variations in the process technologies are not incorporated.

The layout template approaches proposed in [CS92, TD02] require the designers to define layout templates, which form the base for generating detailed layouts and further optimisation. Templates represent unique design knowledge and can be produced by the designer with aid of graphical tools eliminating the need for tedious programming efforts. By their nature the templates must be flexible to accommodate a range of accurate layout solutions, which can be found by e.g. optimisation tools. They must be described efficiently to allow easy manipulation by the layout tools and it is advantageous to include design rules within the template to enable quick generation of correct layouts.

There are three common methods that can be used to describe VLSI layouts. The most popular is the geometric layout format, such as GDSII or CIF, used a standard representation in physical design tools. Since coordinates of the layout format are fixed, it is not suitable to be used as layout templates. The second method is the procedural layout, mentioned above, used in systems such as CAIRO (based on C) [DGL99], Layla (based on Pascal) [Cor85], and ADL (based on C) [EBD84].

The third method is the symbolic layout languages which include ICDL [Wes81], TDL [CDMS88], VIRGIL [Ber85], and a number of others. Unfortunately, most of them can not describe 45-degree or all-angle geometry that is common in analog layouts.

Our layout representation model, Grammar-Based Layout Description (GBLD), is based on formal methods. GBLD is compatible with geometric layout formats, and it is parameterised. The model is hierarchical and can describe all-angle geometry. By incorporating design rules into the model, we can describe parameterised layouts without design-rule violations. By changing the layout parameters, the transistor sizing is achieved. The parasitics can be calculated efficiently [TD02] on the fly. The model can be used for both layout description and generation.

Formal methods have been used in many different fields of computer science and electronic engineering for variety of problems. Their use in compilers and formal proof of both software and hardware is probably most well known. Use of formal methods for description of geometrical objects and their properties was investigated in application to architecture [Kni00] and mechanical engineering [Bro97]. In VLSI physical design, context-free grammars were used to generate layout permutations for floorplanning [MB94] and L-systems were used to generate layouts for contact resistance test structures [MM97]. We focus on application of grammar-based methods to the analog integrated circuit layout design.

GBLD is based on the notions and elements of L-systems and context-free grammars. L-systems have been used in many different applications although it seems that their power has not been yet fully exploited in VLSI layout description and generation. The major difference between L-systems and context-free grammars is that the production rules are applied simultaneously in L-systems. This property can limit the use of L-systems. However, by using L-systems symbols with the production rules defined in context-free grammars [SS94], we have a powerful method which can be applied to modelling the layouts.

We present the methods of using grammars to describe layout geometry in section 2. In section 3, we show how to incorporate design rules into the description of our language. Parameters and routing wires are discussed in section 4. The GBLD language is defined in section 5. Translating GBLD to and from GDSII is discussed in section 6 and conclusions are drawn in section 7.

2 Grammars for describing basic VLSI layouts

The actions of a LOGO-style turtle used in L-systems can be defined in the table below. Note that some researchers define the plus sign as turn right and the minus sign as turn left. Here, we choose to follow the definitions as in [MM97, PLH90].

Symbols	Actions
`F`	Move forward one unit in the current direction and draw a line
`f`	Move forward one unit in the current direction (without drawing a line)
`+`	Turn left 90 degrees
`-`	Turn right 90 degrees

A rectangle with 3 units in length and 1 unit in width can be represented by

```
F - F F F - F - F F F
```

as shown in Figure 1. Note that we assume the default direction is upward and the black dot in the figure is the start point.

The geometry of an NMOS transistor without layer information (Figure 2) can be described by:

```
F - F F F - F - F F F - - f - f - - F F F - F - F F F - F
```

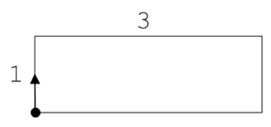

Figure 1

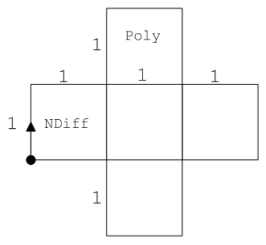

Figure 2

It is easy to realise that this syntax will results in a very long string if there are many polygons. Terminal and non-terminal symbols in BNF can be used to split the syntax string into production rules and make it hierarchical. Before using this kind of syntax, we would like to list the following symbols which are defined in Bracketed OL-systems [PLH90]:

Symbols	Actions
`[`	Push the state of the current turtle position and direction into a stack
`]`	Pop the state from the top of the stack and restore the position and direction of the turtle to the state

Now, the NMOS transistor in Figure 2 can be rewritten to:

```
<NMOS>   →  <NDiff> <Poly>
<NDiff>  →  F - F [ F F - F - F F F
<Poly>   →  ] + F - F - F F F - F - F F
```

The grammar above not only improves the readability, but also helps us keep hierarchies of VLSI layouts. Next, we would like to include layer information into our description format. We define the following symbol:

Symbols	Actions
<*non-terminal*,*layer*>	Generate the string "*layer*(κ)" if there is a production rule defined as "<*non-terminal*>→κ" and <*non-terminal*,*layer*> can only appear on the right-hand side of production rules

For the above grammar describing Figure 2, `<NDiff,"2">` generates string "`2(F - F [ F F - F - F F F)`." The string means that the generated polygon is on layer 2. We can also write `<NDiff,"ndiff">` which means the generated polygon is on N-Diffusion layer. The above grammar describing Figure 2 can be rewritten below with layer information included.

```
<NMOS>   →  <NDiff,"ndiff"> <Poly,"poly">
<NDiff>  →  F - F [ F F - F - F F F
<Poly>   →  ] + F - F - F F F - F - F F
```

By using the extra symbols defined in [SS94] for the turtle, we can then describe a sizeable transistor. We list the symbols in the following table.

Symbols	Actions
`{`	Start recording a string except "`{`", "`}`", and "`!`" on a magnetic tape
`}`	Stop recording and place an end-of-the-record symbol on the tape
`!`	Rewind the tape to the previous end-of-the-record symbol, generate and erase the recorded characters of the string between the two end-of-the-record symbols, then move to the end of the recording

The sizable transistor (Figure 3) with one unit in channel length and at least one unit in channel width can then be described in the following grammar:

```
<NMOS>    →  <NDiff,"ndiff"> <Poly,"poly">
<NDiff>   →  {<Width>} - F [ F F - { ! } - F F F
<Poly>    →  ] + F - F - F { ! } F - F - F { ! }
<Width>   →  F <Width> | F
```

Note that the symbol "`|`" means "or" in context-free grammars. The production rule "`<Width>` → `F <Width> | F`" means that there is at least one "`F`" generated by the non-terminal symbol `<Width>`.

3 Incorporating design rules into the grammar

Design-rule checking (DRC) is a very important phase in VLSI design. Design rules can be considered as a constitution; any design violating the design rules is unacceptable. However, checking design rules is a very time-consuming task performed every design iteration. If design rules can be incorporated into our grammar, we can generate correct-by-construction layouts without design-rule violations, saving this way a number of time consuming design iterations. Typical design rules [PL88] include width, spacing, extension, and enclosure rules. Grammars for describing rules with maximum distances are also demonstrated in the following subsections.

3.1 Width rules

Width rules usually restrict the minimum width of polygon layers. The grammar of the sizeable transistor in section 2 also can be used to describe width rules if the minimum width of the diffusion layer is one unit.

3.2 Spacing rules

In Figure 4, there are two NMOS transistors connected by one's source to the other's drain. If the minimum spacing of the poly layer is 3 units, we can describe the layout as follows:

```
<2NMOS>     →  <NDiff,"ndiff"> ] + <Poly,"poly"> ] + <Poly,"poly">
<NDiff>     →  F - F [ F {<Spacing>} [ F F - F - F F { ! } F F
<Poly>      →  F - F - F F F - F - F F
<Spacing>   →  F <Spacing> | F F F
```

Note that we use the abstract layers of Magic VLSI layout system [OHM$^+$84] to represent mask layers, that is, no NWELL or PWELL layers are needed. While we are drawing `<NDiff>`, the start positions of the poly on the left and on the right are pushed into the stack respectively. After drawing `<NDiff>`, the position of the poly on the right is popped and the polygon is drawn. Then the one on the left is also drawn.

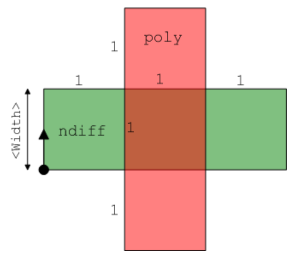

Figure 3

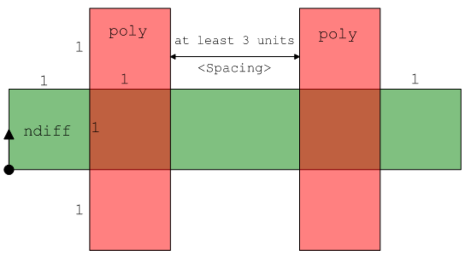

Figure 4

3.3 Extension rules

If the POLY layer has to be at least 1 unit extended over the NWELL layer in an NMOS transistor (Figure 5), the grammar can be written below.

```
<NMOS>   →  <NDiff,"ndiff"> <Poly,"poly">
<NDiff>  →  F - F [ F F - F - F F F
<Poly>   →  ] + {<Ext>} - F - { ! } F { ! } - F - { ! } F
<Ext>    →  F <Ext> | F
```

3.4 Rules with maximum distances

So far, our grammar can only describe minimum distances. In order to enhance its ability to describe maximum distances such as maximum width rules, the notions of the turtle interpretation of parametric words [PLH90] can be incorporated. Therefore, we define the following symbols.

Symbols	Actions
F(x)	Move forward x unit(s) in the current direction and draw a line
F(x,y)	Move forward x unit(s) at least but y unit(s) at most in the current direction and draw a line
f(x)	Move forward x unit(s) in the current direction (without drawing a line)
f(x,y)	Move forward x unit(s) at least but y unit(s) at most in the current direction (without drawing a line)
non-term(x)	Repeat the non-terminal symbol exactly x times
non-term(x,y)	Repeat the non-terminal symbol at least x times but no more than y times
Inf	infinite value

Where *non-term* means non-terminal symbols. Both x and y can be integers and real numbers. For example, `F(13.9)` means forward and draw a line with `13.9` units. We can then replace the production rule "`<Width>` → `F <Width> | F`" in the end of section 2 by "`<Width>` → `F(1, Inf)`." By using these symbols, design rules with maximum distances can be described.

3.5 Enclosure rules

In Figure 6, we assume that the size of the contact polygon is fixed, and the minimum distance of margins between metal layer and contact layer is between 1 unit and 5 units. The grammar can be written below.

```
<Start>      →   [ f - <Contact,"con"> ] [ f f({<Distance>}) -
                 <Metal,"m1"> ]
<Contact>    →   F - F(2) - F(2) - F(2) - F
<Metal>      →   F F({!}) - F({!}) F(2) F({!}) - F({!}) F(2) F({!}) -
                 F({!}) F(2) F({!}) - F({!}) F
<Distance>   →   1 | 2 | 3 | 4 | 5
```

Note that the start point is in the centre and we use a magnetic tape to record the moving unit of "f" and "F" symbols.

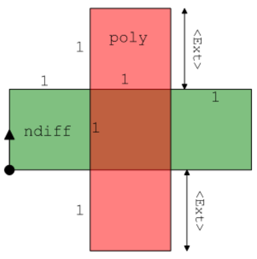

Figure 5. Extension rules

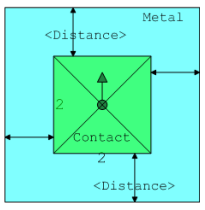

Figure 6. Enclosure rules

4 Parameters and Routing Wires in GBLD

In Figure 7(a), there are two cells A and B. The wire between A and B connects the two cells. The vertical part in the middle of the wire is stretchable. The vertical part can also go down or go straight as in (b) and (c). GBLD can describe these cases in the grammar below.

```
<Wire>       →   F - F(4) {<Turn>} F(4) - F - F(4) {!} F(4)
<Turn>       →   <Up> | <Down> | <Straight>
<Up>         →   + F(<Param>) - F
<Down>       →   F - F(<Param>) +
<Straight>   →   F
<Param>      →   5
```

By controlling the two non-terminals, `<Turn>` and `<Param>`, we can decide the wire directions and wire length, as well as the placement of B relative to A. With suitable software support, the parasitics of the wire can be calculated by considering `<Turn>` and `<Param>` as parameters in a function call of the software. This approach provides efficient and quick estimation of parasitics during the layout process and is obviously better than the traditional analog synthesis method where the electrical properties are estimated after detailed placement and routing.

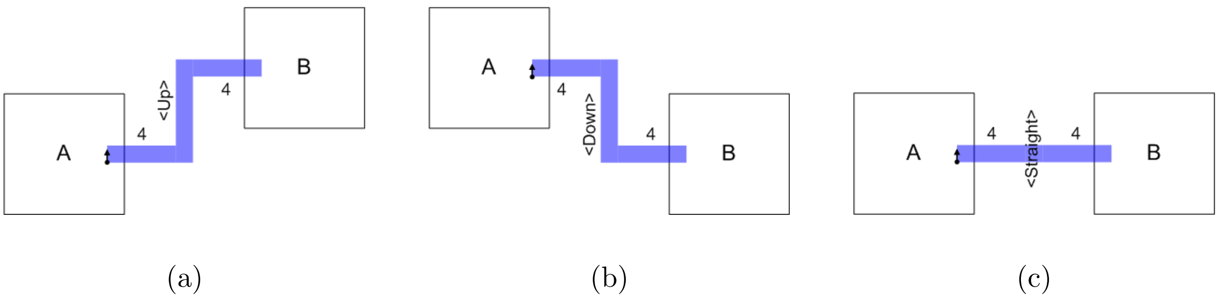

Figure 7. Routing controlled by parameters

5 Definition of the GBLD language

We thus far have defined the basic symbols, but we need other more advanced constructs for the language to be efficient in analog layout. One of such construct is orientation of elements.

5.1 Orientations

By using the following symbols defined in [PLH90], our language is able to describe not only Manhattan but also all-angle geometry.

Symbols	Actions
`+`(x)	Turn left x degrees
`-`(y)	Turn right y degrees

However, we should be careful to use orientation symbols. By using all-angle edges several times in a path of the turtle, we might not be able to go back to the start point to form a polygon.

5.2 Definition of the GBLD language

The similarities of context-free grammar and GBLD are that both of them have terminals, non-terminals, start symbols, and production rules. Because the layer information is required in mask layouts, however, GBLD is 6-tuple while context-free grammar is 4-tuple. The GBLD is defined as follows.

Definition of the GBLD language. A Grammar-Based Layout Description (GBLD) language is 6-tuple (T, N, S, L, U, P), where:

1. T is a finite set, called terminal symbols (or terminals), which is the set {“F”, “f”, “+”, “-”, “(”, “)”, “[”, “]”, “{”, “}”, “!”, “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “9”}. The meanings of the symbols have been defined in this paper.
2. N is a finite set, called non-terminal symbols (or non-terminals).
3. $S \in N$ is the start symbol. We put S as the first production rule in all example grammars.
4. L is a finite set, called layers, which includes all layer names in a VLSI layout.
5. U is a finite set, called usages. Usages are in the form of <a, b> where $a \in N$ and $b \in L$. Usages can only appear on the right-hand side of production rules.
6. P is a finite set, called production rules. Each production rule consists of a terminal, followed by an arrow, followed by a string of terminals, non-terminals, and usages.

Like context-free grammars, the GBLD language can generate strings. The strings generated by GBLD represent flattened mask layouts. Some automation tasks, such as netlist extraction, can be performed more efficiently if layout hierarchies are preserved. An example of a non-parameterised inverter described hierarchically in GBLD is shown in the Appendix. GBLD code in Figure A.1 describes the layout in Figure A.2. A cascade of two inverters is described as “`<2_inv>` → `<inv>(2)`.” To generate a cascade of N inverters, we can write “`<N_inv>` → `<inv>(N)`.”

6 Translating to and from GDSII format

In order to interface our development to the existing layout tools, we developed a conversion program to translate between GBLD and GDSII. This will allow the experimentation with the use of this format for post layout processing such as compaction.

6.1 GBLD to GDSII

In GDSII Stream format [Hol04], the `BOUNDARY` and `SREF` records are two important element types. One is used to record polygon coordinates, and the other is used to reference other structures. For a polygon with n vertices, $(n+1)$ coordinates are recorded in a `BOUNDARY` record with the first vertex identical to the last one. `SREF`s are used to include other structures in a structure. The hierarchy can thus be kept.

A structure (`BGNSTR`...`ENDSTR`) in GDSII can have many polygons (`BOUNDARY`s) and other structures (`SREF`s) inside. The layer information (layer name or layer number) must be specified to those polygons in order to form meaningful data of VLSI layout. From the example in the Appendix, we can see that only rules with sufficient layer information can form a structure (cell) in VLSI layout. In the example, `<metal_rail>` cannot form a structure, but `<nmos>` can. The string that `<nmos>` stands for is:

```
ndiff(F(22)-F(10)-F(22)-F(10)-)f(10)+f(2)-poly(F(2)-F(14)-F(2)-F(14)-)
```

In the GBLD, the non-terminals with layer information can map to the `BOUNDARY`s and the non-terminals on the right-hand side without layer information (should have been defined in the lower structures, like `<nmos>` and `<pmos>` in "`<inv>`") can map to `SREF`s. We can translate the grammar-based layout description to the `BOUNDARY` and `SREF` records of GDSII in linear time if the "`{`", "`}`", and "`!`" symbols of our grammar have been expanded. It is linear time because translating from GBLD to GDSII is as simple as tree traversals. The translated GDSII example in KEYformat [Hol04] with some fields omitted is in Figure A.3. For the CIF format, tools are provided to translate between GDSII and CIF.

6.2 GDSII to GBLD

Translating from GDSII to GBLD is straightforward when hierarchy of GDSII is to be preserved. In Figure A.3, the simplest way to translate back to GBLD format is to generate three non-terminal symbols, `<nmos>`, `<pmos>`, and `<inv>`. They conform to the three structures in the GDSII format.

GBLD has more description power than GDSII. Much more efforts in the translation are required if we would like to utilise that power. For example, we would need algorithms to search for the same polygons on different layers. Then use non-terminals with layer parameters to describe them. Inputs from users are needed if generating GBLD with "`{`", "`}`", "`!`", and "`|`" symbols is desired.

6.3 File Sizes

GDSII is a very popular VLSI layout format in integrated circuit industries, however, it has many drawbacks [RBGB01, SEM03]. One of the most serious problems is huge file sizes. Using GBLD can make file sizes compact. In the case of identical polygons residing in different layers (like the `<contact>` and `<diff>` in Figure A.1), GDSII has to record coordinates of the polygons on each layer, but GBLD can simplify the syntax by using one non-terminal symbol with layer parameters. Even the latest layout format OASIS [SEM03] does not have this concept. Therefore, the GBLD file size can be smaller than GDSII, CIF, XML-based [RBGB01], and OASIS formats after compression.

7 Conclusion

We presented the GBLD model as a formal method that can be used to both describe and generate integrated circuit layouts. Layout parameterisation and incorporation of design rules make it efficient and suitable for analog layout templates. Moreover, the compact GBLD file size makes it a potential candidate for the next generation layout format after some modifications and extensions. At present, we are developing prototype tools, based on the presented format, for post-layout processing and transistor sizing of analog cells. Our near future work will be focused on developing an analog layout design system using layout templates.

References

[Ber85] Neil Bergmann. Generalised CMOS - a technology independent CMOS IC design style. In *Design Automation Conference*, pages 273–278, 1985.

[Bro97] K. Brown. Grammatical design. *IEEE Expert*, 12(2):27–33, 1997.

[CDMS88] K. Croes, H.J. De Man, and P. Six. CAMELEON: a process-tolerant symbolic layout system. *IEEE Journal of Solid-State Circuits*, 23(3):705–713, 1988.

[Cor85] Warren E. Cory. Layla: a VLSI layout language. In *Design Automation Conference*, pages 245–251, 1985.

[CS92] J.D. Conway and G.G. Schrooten. An automatic layout generator for analog circuits. In *European Conference on Design Automation*, pages 513–519, 1992.

[DGL99] M.A. Dessouky, A. Greiner, and M.M. Louerat. CAIRO: a hierarchical layout language for analog circuits. In *Mixed Design of Integrated Circuits and Systems*, pages 105–110, Krakow, Poland, 1999.

[DL00] M. Dessouky and M.-M. Louerat. A layout approach for electrical and physical design integration of high-performance analog circuits. In *International Symposium on Quality Electronic Design*, pages 291–298, San Jose, CA, USA, 2000.

[EBD84] W.H. Evans, J.C. Ballegeer, and N.H. Duyet. ADL: an algorithmic design language for integrated circuit synthesis. In *Design Automation Conference*, pages 66–72, 1984.

[Hol04] Klaas Holwerda. GDSII format, `http://www.xs4all.nl/~kholwerd/interface/bnf/gdsformat.html`. last access in July 2004.

[Kni00] Terry Knight. Shape grammars in education and practice: history and prospects. *International Journal of Design Computing*, 2, 2000.

[MB94] M. Mortazavi and N.G. Bourbakis. A generic floorplanning methodology. In *AUTOTESTCON '94. IEEE Systems Readiness Technology Conference. 'Cost Effective Support Into the Next Century'*, pages 749–763, 1994.

[MM97] L.A. MacEachern and T. Manku. Hilbert-Peano curve descriptions of layouts for contact resistance test structures in CMOS. In *Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 629–632, 1997.

[ODJ+95] B.R. Owen, R. Duncan, S. Jantzi, C. Ouslis, S. Rezania, and K. Martin. BALLISTIC: an analog layout language. In *Custom Integrated Circuits Conference*, pages 41–44, Santa Clara, CA, USA, 1995.

[OHM+84] J.K. Ousterhout, G.T. Hamachi, R.N. Mayo, W.S. Scott, and G.S. Taylor. Magic: a VLSI layout system. In *Design Automation Conference*, pages 152–159, 1984.

[PL88] Bryan T. Preas and Michael J. Lorenzetti. *Physical design automation of VLSI systems*. Benjamin/Cummings Pub. Co, Menlo Park, CA, USA, 1988.

[PLH90] Przemyslaw Prusinkiewicz, Astrid Lindenmayer, and James Hanan. *The algorithmic beauty of plants*. Springer-Verlag, New York ; Berlin, 1990.

[RBGB01] Alfred Reich, Robert Boone, Warren Grobman, and Clyde Browning. GDSII considered harmful. In *21st Annual BACUS Symposium on Photomask Technology in Monterey*, California, USA, 2001.

[SEM03] SEMI. OASIS(TM) - open artwork system interchange standard, 2003.

[SS94] S.-G. Shih and G. Schmitt. The use of post interpretation for grammar-based generative systems. In J.S. Gero and E. Tyugu, editors, *Formal Design Methods for CAD (B-18)*, pages 107–120. Elsevier Science B.V., 1994.

[TD02] Hua Tang and A. Doboli. Employing layout-templates for synthesis of analog systems. In *Midwest Symposium on Circuits and Systems*, volume 2, pages 505–508, 2002.

[Wes81] Neil Weste. Virtual grid symbolic layout. In *Design Automation Conference*, pages 225–233, 1981.

Appendix

```
<inv>          →   [ <nmos> ] [ f(32) <pmos> ] [ f(10) + f(4)
                   - <poly_in, "poly"> ] [ f(16) - f(2) +
                   <metal_out, "m1"> ] [ + f(8) - <metal_rail,
                   "m1"> f(48) <metal_   rail,"m1"> ] [ - f(4) +
                   f(2) <contact,"ndc"> f(16) <contact,"ndc">
                   f(16) <contact,"pdc"> f(16) <contact,"pdc"> ]
                   [ f(26) - f(13) + <contact,"pc">] - f(20) +
<metal_out>    →   F(22) - F(6) - F(9) + F(8) - F(4) - F(8) +
                   F(9) - F(6) -
<poly_in>      →   F(15) + F(4) - F(4) - F(4) + F(15) - F(2) -
                   F(34) - F(2) -
<nmos>         →   <diff, "ndiff"> f(10) + f(2) - <poly,"poly">
<pmos>         →   <diff, "pdiff"> f(10) + f(2) - <poly,"poly">
<diff>         →   F(22) - F(10) - F(22) - F(10) -
<poly>         →   F(2) - F(14) - F(2) - F(14) -
<contact>      →   F(2) - F(2) - F(2) - F(2) -
<metal_rail>   →   F(6) - F(24) - F(6) - F(24) -
```

Figure A.1. GBLD language describing the inverter in Figure A.2.

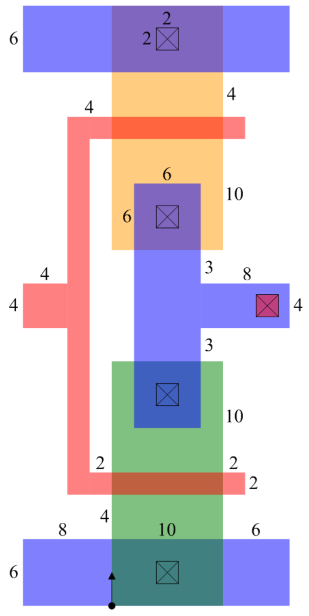

Figure A.2. Layout of an inverter

```
BGNSTR;
STRNAME nmos;

BOUNDARY; LAYER ndiff;
    XY 5;
    X    0;  Y    0;    X    0;  Y   22;
    X   10;  Y   22;    X   10;  Y    0;
    X    0;  Y    0;
ENDEL;

BOUNDARY; LAYER poly;
    XY 5;
    X   -2;  Y   10;    X   -2;  Y   12;
    X   12;  Y   12;    X   12;  Y   10;
    X   -2;  Y   10;
ENDEL;

ENDSTR nmos;

BGNSTR;
STRNAME pmos;

BOUNDARY; LAYER pdiff;
    XY 5;
    X    0;  Y    0;    X    0;  Y   22;
    X   10;  Y   22;    X   10;  Y    0;
    X    0;  Y    0;
ENDEL;

BOUNDARY; LAYER poly;
    XY 5;
    X   -2;  Y   10;    X   -2;  Y   12;
    X   12;  Y   12;    X   12;  Y   10;
    X   -2;  Y   10;
ENDEL;

ENDSTR pmos;

BGNSTR;
STRNAME inv;

SREF; SNAME nmos;
    XY 1;
    X    0;  Y    0;
ENDEL;

SREF; SNAME pmos;
    XY 1;
    X    0;  Y   32;
ENDEL;

BOUNDARY; LAYER poly;   /* poly_in */
    XY 9;
    X   -4;  Y   10;    X   -4;  Y   25;
    X   -8;  Y   25;    X   -8;  Y   29;
    X   -4;  Y   29;    X   -4;  Y   44;
    X   -2;  Y   44;    X   -2;  Y   10;
    X   -4;  Y   10;
ENDEL;
```

```
BOUNDARY; LAYER m1;     /* metal_out */
    XY 9;
    X    2;  Y   16;    X    2;  Y   38;
    X    8;  Y   38;    X    8;  Y   29;
    X   16;  Y   29;    X   16;  Y   25;
    X    8;  Y   25;    X    8;  Y   16;
    X    2;  Y   16;
ENDEL;

BOUNDARY; LAYER m1;     /* metal_rail (bottom) */
    XY 5;
    X   -8;  Y    0;    X   -8;  Y    6;
    X   16;  Y    6;    X   16;  Y    0;
    X   -8;  Y    0;
ENDEL;

BOUNDARY; LAYER m1;     /* metal_rail (top) */
    XY 5;
    X   -8;  Y   48;    X   -8;  Y   54;
    X   16;  Y   54;    X   16;  Y   48;
    X   -8;  Y   48;
ENDEL;

BOUNDARY; LAYER ndc;    /* (lower one) */
    XY 5;
    X    4;  Y    2;    X    4;  Y    4;
    X    6;  Y    4;    X    6;  Y    2;
    X    4;  Y    2;
ENDEL;

BOUNDARY; LAYER ndc;    /* (upper one) */
    XY 5;
    X    4;  Y   18;    X    4;  Y   20;
    X    6;  Y   20;    X    6;  Y   18;
    X    4;  Y   18;
ENDEL;

BOUNDARY; LAYER pdc;    /* (lower one) */
    XY 5;
    X    4;  Y   34;    X    4;  Y   36;
    X    6;  Y   36;    X    6;  Y   34;
    X    4;  Y   34;
ENDEL;

BOUNDARY; LAYER pdc;    /* (upper one) */
    XY 5;
    X    4;  Y   50;    X    4;  Y   52;
    X    6;  Y   52;    X    6;  Y   50;
    X    4;  Y   50;
ENDEL;

BOUNDARY; LAYER pc;
    XY 5;
    X   13;  Y   26;    X   13;  Y   28;
    X   15;  Y   28;    X   15;  Y   26;
    X   13;  Y   26;
ENDEL;

ENDSTR inv;
```

Figure A.3. GDSII in ASCII format translated from Figure A.1.