A Layout-Aware Circuit Sizing Model Using Parametric Analysis

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

Abstract — We propose a circuit sizing model that takes layout parasitics into account. The circuit and layout parameters are stored in a parameterized layout description format, GBLD. The layout parasitics are stored as closed form expressions. Layout optimization tools can modify the layout and recalculate parasitics on the fly. If the results of sensitivity analysis are passed to those tools, optimization for performance can be achieved with relatively few iterations involving time consuming circuit simulations.

I. Introduction

The difficulties in analog integrated circuit design are due to numerous electrical effects induced by layouts, especially for high performance and high speed circuits. Lack of exact layout information during circuit sizing leads to long design iterations involving time consuming runs of complex tools.

The traditional analog design flow, as shown in Fig.1(a), involves repetitive iterations of circuit sizing, layout generation, electrical values extraction, and performance evaluation. Circuit simulators, such as Spice, are used during circuit sizing. Designers have to simulate different sets of parameters in order to evaluate circuit performance. After circuit sizing, layout is generated according to the circuit and parameters. Considerable experience is needed from the designer to manually produce good layout for a given sized circuit. Then circuit parasitics are extracted from the detailed layout and performance is evaluated. Redesign is needed whenever the final performance does not conform to the specification.

Novel analog design flows [1-3] have been proposed in order to automate the iterations and thus reduce the design time. In the layout-in-the-loop approach [1,2], as illustrated in Fig.1(b), procedural layout generation tools are used to automate generation of layouts. In [3], estimations of parasitics are used in order to replace slow layout generation and extraction steps. In all these approaches, however, there is no information about how layout parameters affect final performance. Therefore, iterations may never converge by just trying many combinations of circuit sizing parameters.

Another problem in the approaches above is that only one version of layout is generated according to one combination of circuit sizing parameters. Since a combination of circuit sizing parameters can have many different layouts, it is possible that we can correct the design performance by modifying the layout instead of sizing the circuit and generating the layout again.

Our approach in this paper is to keep both circuit and layout parameters updated along the design flow, with parasitics represented as closed form equations. Quite many parasitics can be calculated that way and extraction tools are invoked only when necessary. GBLD, our parameterized layout format, provides for easy storage and manipulation of those parameters. After extraction, parametric and symbolic analysis techniques can be used to analyze the relationships between parameters and performance of the design. Then decisions of the parameter values can be made for the next circuit sizing loop. The automation of the loop in the design flow can be achieved with the support of software which has parameter control functions.

This paper is organized as follows. We introduce parameterized layout format GBLD in section II. Symbolic extraction is presented in section III. In section IV, we discuss evaluation of performance on this circuit sizing model. The design flow is proposed in section V. Conclusions are drawn in section VI.

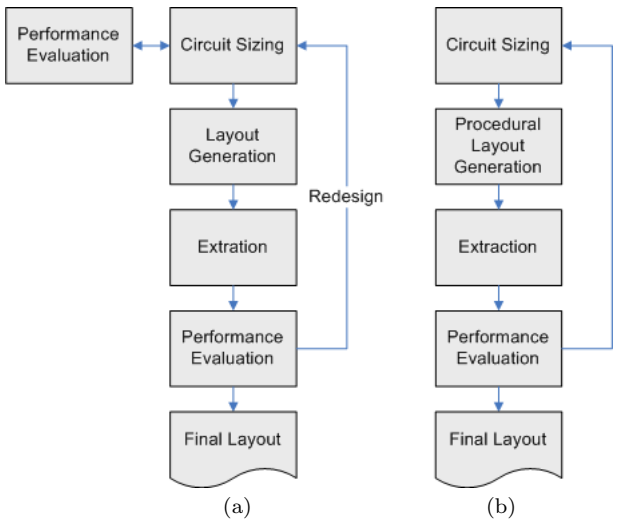


Fig. 1. Analog design flows: (a) Traditional (b) Layout-in-the-loop

II. Parameterized Layout Format

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, the formats can not describe parameterized layouts. The second method is the procedural layout language, such as CAIRO (based on C) [4], Layla (based on Pascal) [5], and ADL (based on C) [6]. These languages are close to programming languages and are used to generate layouts. Thus, they are not suitable to be processed by programs.

The third method is the symbolic layout languages which include ICDL [7], TDL [8], VIRGIL [9], 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 language, Grammar-Based Layout Description (GBLD) [10], is based on formal methods. GBLD is compatible with geometric layout formats, and it is parameterized. The language is hierarchical and can describe all-angle geometry. GBLD is defined as follows, and the symbols are defined in TABLE I:

Definition of GBLD. 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 are defined in TABLE I.
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.

Circuit of a current mirror using NMOS transistors is shown in Fig.2. The Spice netlist of the circuit can be written as:

```
M1 net1 net1 net0 0 L=3 W=3
M2 VDD net1 net0 0 L=3 W='w1'
```

where `w1` is a circuit parameter in this example. The current ratio I_O/I_{REF} can be controlled by `w1`.

The layout according to the circuit and circuit sizing parameters is shown in Fig.3. Note that contacts are omitted and the layout is in parameterized form. Both channel length and width of `M1` are 3 units as in Fig.2 and Fig.3. For NMOS transistor `M2`, the channel length is 3 units while

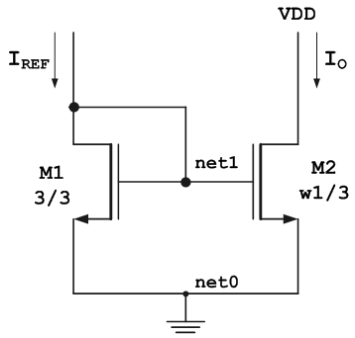


Fig. 2. Circuit of a current mirror

TABLE I
Symbols used in GBLD

Symbols	Meanings
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
[	Push the state of the current 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
<NT,ly>	Generate string "$ly(\kappa)$" if there is a production rule defined as "<NT>→κ" and <NT,ly> can only appear on the right hand side of production rules. (ly stands for layer name or layer number. κ stands for a polygon or polygons.)
{	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
$Act(x)$	Move forward x unit(s) in the current direction with/without drawing a line. Here Act stands for symbol "F" or "f"
$Act(x,y)$	Move forward x unit(s) at least but y unit(s) at most in the current direction with/without draw a line. Here Act stands for symbol "F" or "f"
$NT(x)$	Repeat the non-terminal symbol exactly x times
$NT(x,y)$	Repeat the non-terminal symbol at least x times but no more than y times

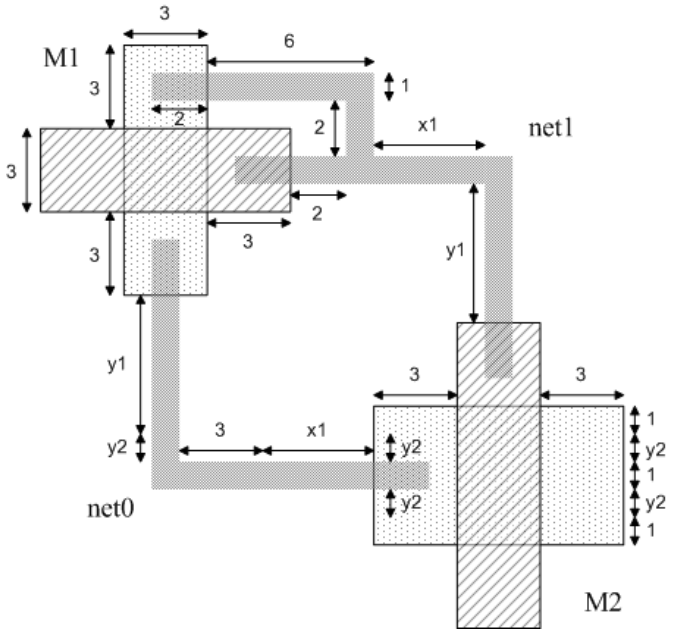


Fig. 3. Parameterized layout of Fig.2.

```
<CM>      →   [<M1>][-f(4)-f(1)--<net0>][f(4)-f(4)+<net1>
              ][-f(7)f(<x1>)-f(3)f(<y1>)f(<y2>)F(1)
              F(<y2>)F(1)--<M2>]
<poly1>   →   {F(3)-F(9)-}{!}
<diff1>   →   {F(9)-F(3)-}{!}
<M1>      →   [<poly1,"poly">][-f(3)-f(3)--<diff1,
              "ndiff">]
<diff2>   →   {F(1)F(<y2>)F(1)F(<y2>)F(1)-F(9)-}{!}
<poly2>   →   {F(9)F(<y2>)F(<y2>-F(3)-}{!}
<M2>      →   [<diff2,"ndiff">][-f(3)-f(3)--<poly2,
              "poly">]
<net0>    →   -F(1)-F(2)F(<y1>)F(<y2>)+F(2)F(x1)F(2)-
              F(1)-F(2)F(<x1>)F(3)-F(1)F(<y2>)F(<y1>)F(2)
<net1>    →   F(1)-F(8)-F(3)+F(<x1>)F(1)-F(1)F(<y1>)F(2)
              -F(1)-F(2)F(<y1>)+F(<x1>)F(5)-F(1)-F(4)+
              F(2)+F(7)
```

Fig. 4. GBLD describing parameterized layout in Fig. 3.

the width is a parameter `w1`. We can derive the equation that:

```
w1 = 1 + y2 + 1 + y2 + 1 = 3 + 2*y2
```

Note that `w1` is a circuit parameter, while `y2` is a parameter from the layout. Other layout parameters include `x1` and `y1`. The GBLD language which describes the layout in Fig.3 is shown in Fig.4.

Another example of using GBLD to describe the layout of a differential amplifier shown in [11] is parameterized and presented in the Appendix. Circuit and layout parameters used in the example are listed in TABLE A.1.

III. Symbolic Extraction

In the current analog design flow, the parameters cannot be passed to the extraction level. Since coordinates of the layout before extraction are fixed, no previous variables are preserved. Thus, only one combination of circuit and layout parameters can be simulated at a time during the long loop of layout generation and extraction. In addition, further analysis of the relationships between design performance and layout parameters becomes impossible.

Since we would like to pass parameters in circuit sizing or layout synthesis to the final performance evaluation level, the results of the extraction step must be a netlist with symbolic values. We call this extraction step *symbolic extraction*.

There are many methods in modeling distributed RC lines in layout parasitics extraction. We use the Π model here because it is simple to demonstrate. Other RC line models can be used in our circuit sizing model as well. For a wire in the layout, two capacitors and one resistor are used to model the RC effects, as illustrated in Fig.5.

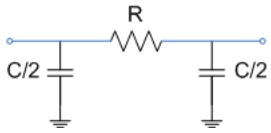


Fig. 5. Π model for distributed RC line

After symbolic extraction of the parameterized layout in Fig. 4, the extracted netlist becomes (circuit shown in Fig.6):

```
M1 n01 n03 n11 0 L=3 W=3
M2 VDD n05 n12 0 L=3 W='3+2*y2'
R1 n01 n02R='10*p'
R2 n03 n04 R='4*p'
R3 n04 n05 R='(4+x1+y1)*p'
R4 n11 n12 R='(7+x1+y1+y2)*p'
C1 n01 0 C='10*q/2'
C2 n02 0 C='10*q/2'
C3 n03 0 C='4*q/2'
C4 n04 0 C='(8+x1+y1)*q/2'
C5 n05 0 C='(4+x1+y1)*q/2'
C6 n11 0 C='(7+x1+y1+y2)*q/2'
C7 n12 0 C='(7+x1+y1+y2)*q/2'
```

We can consider `p` as (sheet resistance of metal-1)/(wire width) and `q` as (permittivity of SiO2)*(wire width)/(distance between the metal-1 layer and the substrate layer). Therefore, both `p` and `q` are constants. Note that the circuit parameter `w1` has been translated to `'3+2*y2'` and thus the circuit and layout parameters are preserved in the extracted netlist.

Another method to model distributed RC lines is to use the SPICE3 uniform-distributed RC-line (URC) model. It is very convenient because we only need to extract the length of the wire in symbolic form as the main parameter in the model.

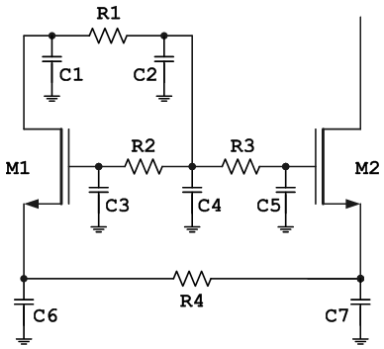


Fig. 6. The netlist after extraction

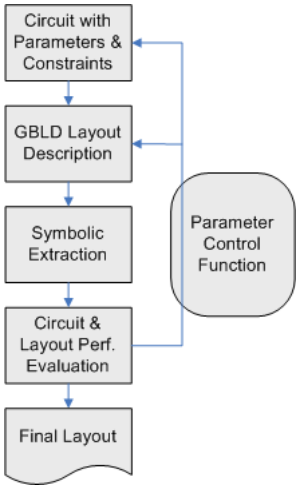


Fig. 7. Proposed analog design flow

IV. Performance Evaluation

After performing symbolic extraction, a netlist with symbolic parameters is generated. There are two methods that can be used to analysis the netlist. One is to use parametric analysis from numerical simulation tools, the other is to perform symbolic analysis.

A. Parametric Analysis

Parametric Analysis is a function that can be found in circuit simulation tools such as OrCAD PSpice. Parameters can be used as inputs of the simulator. First, users can set the start, end, and step values of a parameter. Then the simulator runs several times according to different values of the parameters. The result is a graph with curves representing different parameter values. This can be a very useful function with which designers can evaluate different trade-offs. In addition, many combinations of parameters are simulated at once without layout re-generation and re-extraction.

B. Symbolic Analysis

The other method that can be used to analyze netlists extracted by symbolic extraction is symbolic analysis. An introduction to symbolic analysis can be found in the review paper [12]. It is a method used to analyze analog circuits. The results of symbolic analysis are equations representing different circuit characteristics, such as transfer functions, voltage gain, etc. From the generated equations, analog designers are able to evaluate relationships between input parameters and the design performance. Decisions can be made better in the next circuit sizing task.

V. The Design Flow

The proposed design flow is illustrated in Fig.7. Values of parameters that can not be decided at the circuit sizing level can be left as parameters, or as parameters with constraints. GBLD can be used to represent parameterized layout according to the netlist of the circuit. Design rules in the parameterized layout can be maintained by using constraints on layout parameters. Symbolic extraction only needs to be executed once for a layout structure. After symbolic extraction, netlists with symbolic parameters and equations inside can be analyzed. We may only need to modify layout parameters if the performance does not conform to the specification. With support of parameter-control functions, both circuit and layout variables are able to be decided incrementally and automatically.

VI. Conclusion

A circuit sizing model using parameterized layout format, GBLD, has been presented in this paper. With this model, both circuit and layout parameters are kept in the extracted netlist. The netlist can be analyzed by using parametric or symbolic analysis. Layouts can be modified to meet the performance requirements without sizing the circuit.

Appendix

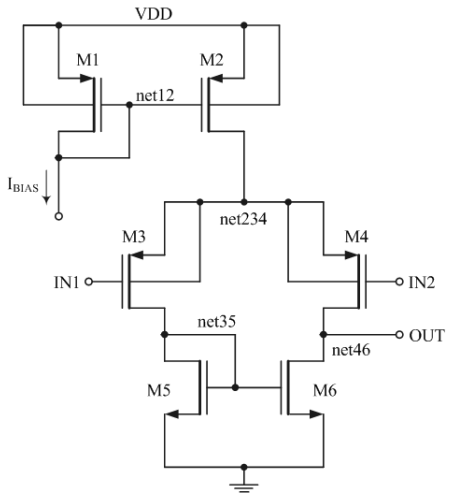


Fig. A.1. Circuit of a differential amplifier

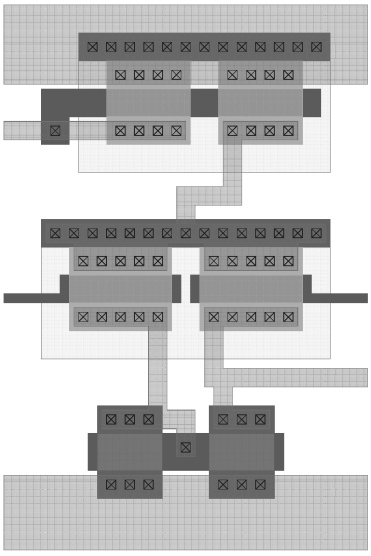
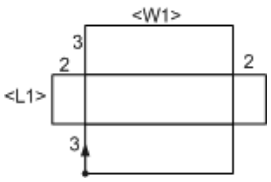
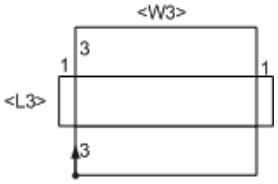


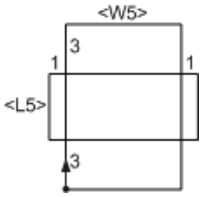
Fig. A.2. Layout structure of the circuit in Fig. A.1.



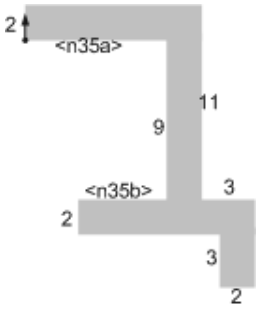
(a) <pmos12> for M1, M2



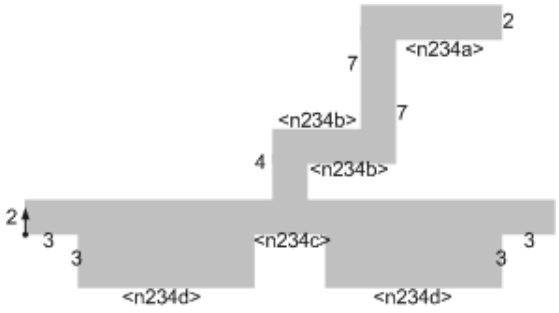
(b) <pmos34> for M3, M4



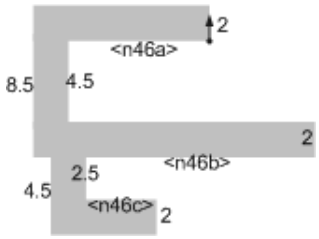
(c) <nmos56> for M5, M6



(d) <net35>



(e) <net234>



(f) <net46>

Fig. A.3. Part of parameterized layout in Fig. A.2.

```
<AMP>       →   [<m34_1>][f(3.5)-f(3.5)+<net35,"m1">][f(3.5)-
                f(3)f(<W3>)f(<n234c>)f(<W3>)++f(0.5)-<net46,
                "m1">][-f(3)f(<W3>)-f(8)f(<L5>)f(3)-f(1)f(<W5>)
                -<m56>][f(6)f(<L3>)f(3.5)-f(0.5)+<net234,"m1">]
                [f(6)f(<L3>)f(11)-f(0.5)f(3)f(<n234d>)f(<n234b>)
                f(2)f(<n234a>)f(0.5)--f(<W1>)f(3)f(<W1>)f(3)-}
                <m12>]
<diff12>    →   {F(3)F(<L1>)F(3)-F(<W1>)}-}{!}
<poly12>    →   {F(<L1>)-F(2)F(<W1>)F(2)-}{!}
<pmos12>    →   <diff12,"pdiff">f(3)+f(2)-<poly12,"poly">
<diff34>    →   {F(3)F(<L3>)F(3)-F(<W3>)-}{!}
<poly34>    →   {F(<L3>)-F(1)F(<W3>)F(1)-}{!}
<pmos34>    →   <diff34,"pdiff">f(3)+f(1)-<poly34,"poly">
<diff56>    →   {F(3)F(<L5>)F(3)-F(<W5>)-}{!}
<poly56>    →   {F(<L5>)-F(1)F(<W5>)F(1)-}{!}
<nmos56>    →   <diff56,"ndiff">f(3)+f(1)-<poly56,"poly">
<net234>    →   F(2)-F(3)F(<n234d>)+F(4)-F(<n234b>)+F(7)-F(2)
                F(<n234a>)-F(2)-F(<n234a>)+F(7)-F(<n234b>)+F(2)+
                F(<n234d>)F(3)-F(2)-F(3)+F(3)-F(n234d)-F(3)+
                F(<n234c>)+F(3)-F(<n234d>)-F(3)+F(3)-
<net35>     →   F(2)-F(<n35a>)F(2)-F(11)+F(3)-F(5)-F(2)-F(3)+F(3)
                F(<n35b>)-F(2)-F(<n35b>)+F(9)+F(<n35a>)-
<net46>     →   F(2)+F(<n46a>)F(2)+F(8.5)+F(1)-F(4.5)+F(2)
                F(<n46c>)+F(2)+F(<n46c>)-F(2.5)-F(<n46b>)+F(2)+
                F(<n46b>)F(1)-F(4.5)-F(<n46a>)+
<n12poly>   →   {F(<L1>)-F(<n12a>-)}{!}
<m12_1>     →   [<pmos12>][f(3)-f(<W1>)+<n12poly,"poly">][-
                f(<W1>)f(<n12a>)+<pmos12>]
<nwell1>    →   {F(6)F(<L1>)F(6)-F(3)F(<W1>)F(3)F(<W1>)F(3)-}{!}
<ibias>     →   {F(2)+F(<nbias>)+}{!}
<nplus1>    →   {F(3)-F(3)F(<W1>)F(3)F(<W1>)F(3)-}{!}
<ibpoly>    →   F(3)F(<L1>)-F(7)-F(<L1>)-F(4)+F(3)-F(3)-
<m12>       →   [<nwell1,"nwell">][f(3)-f(3)+<m12_1>][f(3.5)-
                f(3)f(<W1>)--f(0.5)-<ibias,"m1">][f(6)f(<L1>)
                f(3)<nplus1,"nplus">][f(3)+f(4)-<ibpoly,"poly">]
<n35poly>   →   {F(<L5>)-F(<n35c>)-}{!}
<m56>       →   [<nmos56>][f(3)-f(<W5>)+<n35poly,"poly">][-f(<W5>)
                f(<n35c>)+<nmos56>]
<m34>       →   [<pmos34>][-f(<W3>)f(<n234c>)+<pmos34>]
<in1>       →   {F(1)+F(<nin1>)+}{!}
<in2>       →   {F(1)-F(<nin2>)-}{!}
<nplus2>    →   {F(3)-F(3)F(<W3>)F(3)F(<W3>)F(3)-}{!}
<nwell2>    →   {F(6)F(<L3>)F(6)-F(3)F(<W3>)F(3)F(<W3>)F(3)-}{!}
<m34_1>     →   [<nwell2,"nwell">][f(3)-f(3)+<m34>][f(6)-f(2)+
                <in1,"poly">][f(6)-f(3)f(<W3>)f(<n234c>)f(<W3>)
                f(1)+<in2,"poly">][f(6)f(<L3>)f(3)<nplus2,
                "nplus">]
```

Fig. A.4. The GBLD syntax for the parameterized layout in Fig. A.2.

TABLE A.1
Lists of circuit and layout parameters in Fig. A.4.

Circuit Parameters	<L1>, <W1>, <L3>, <W3>, <L5>, <W5>
Layout Paramaters	<n234a>, <n234b>, <n234c>, <n234d>, <n35a>, <n35b>, <n35c>, <n46a>, <n46b>, <n46c>, <nin1>, <nin2>, <n12a>, <nbias>

References

[1] P. Vancorenland, G. Van der Plas, M. Steyaert, G. Gielen, and W. Sansen, "A layout-aware synthesis methodology for RF circuits," *International Conference on Computer Aided Design*, pp. 358-362, 2001.

[2] M. Dessouky, M.-M. Louerat, and J. Porte, "Layout-oriented synthesis of high performance analog circuits," *Design, Automation and Test in Europe Conference and Exhibition*, pp. 53-57, 2000.

[3] A. Agarwal, H. Sampath, V. Yelamanchili, and R. Vemuri, "Accurate estimation of parasitic capacitances in analog circuits," *Design, Automation and Test in Europe Conference and Exhibition*, Vol. 2, pp. 1364-1365, 2004.

[4] M. A. Dessouky, A. Greiner, and M. M. Louerat, "CAIRO: A hierarchical layout language for analog circuits," *Mixed Design of Integrated Circuits and Systems (MIXDES)*, 1999.

[5] W. E. Cory, "Layla: A VLSI layout language," *Design Automation Conference*, pp. 245-251, 1985.

[6] W. H. Evans, J. C. Ballegeer, and N. H. Duyet, "ADL: An algorithmic design language for integrated circuit synthesis," *Design Automation Conference*, pp. 66-72, 1984.

[7] N. Weste, "Virtual grid symbolic layout," *Design Automation Conference*, pp. 225-233, 1981.

[8] K. Croes, H. J. De Man, and P. Six, "CAMELEON: A process-tolerant symbolic layout system," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 705-713, 1988.

[9] N. Bergmann, "Generalised CMOS – a technology independent CMOS IC design style," *Design Automation Conference*, pp. 273-278, 1985.

[10] I-L. Tseng and A. Postula, "GBLD: A formal model for layout description and generation," *Forum on specification and Design Languages*, 2004, in press.

[11] Y. Tsividis, *Mixed Analog-Digital VLSI Devices and Technology*, World Scientific Publishing Company, 2002.

[12] G.G.E. Gielen and R.A. Rutenbar, "Computer-aided design of analog and mixed-signal integrated circuits," *Proceedings of the IEEE*, vol. 88, pp. 1825-1854, 2000.