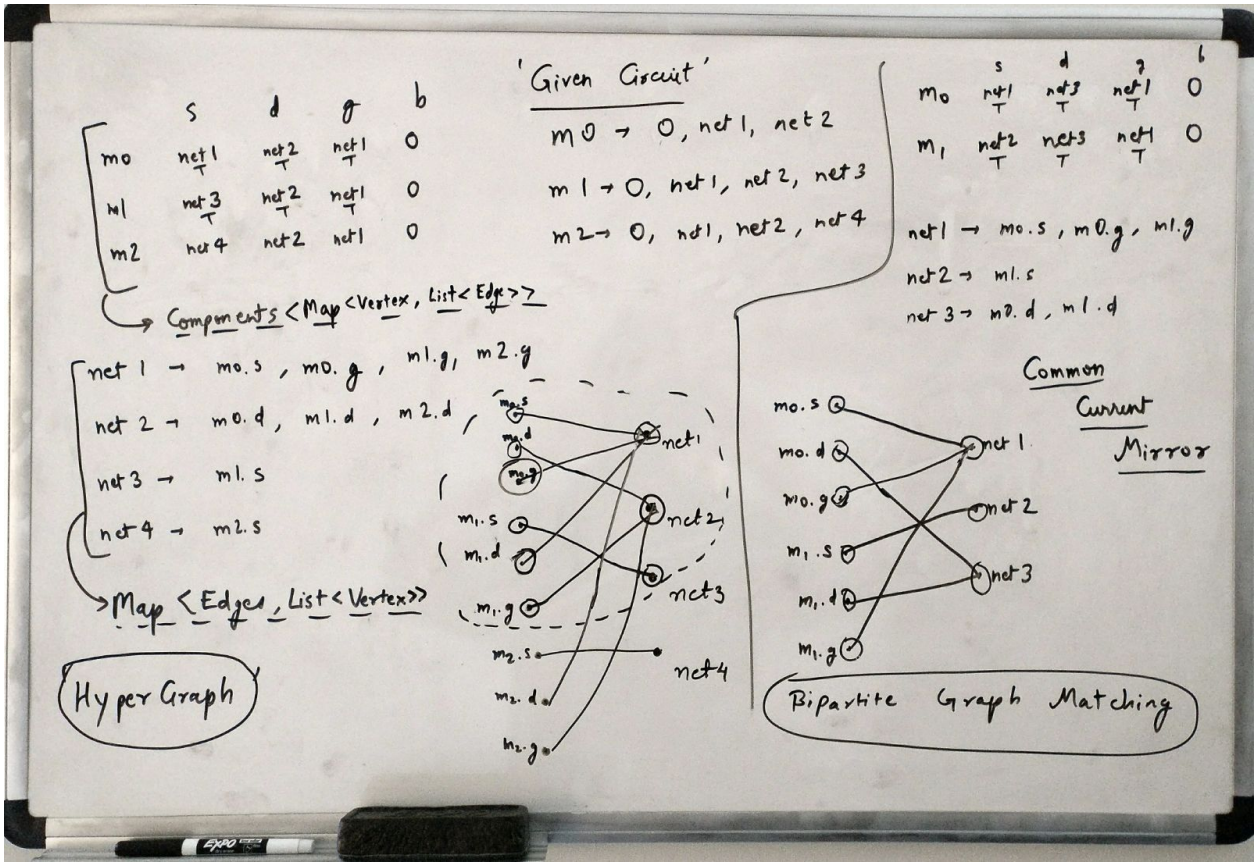
ESE 530 - COMPUTER AIDED DESIGN

PROJECT 1

The Sizing Rules Method for CMOS and Bipolar Analog Integrated Circuit Synthesis

Prathak Rastogi (110951108), Arun Kumar Ramakrishnan (111146101)



CONTENTS

Introduction

Analog components are an important part of integrated systems be it elements and area in mixed-signal systems. Even though its design is more important than digital circuits, the design automation of analog circuits still lags behind.

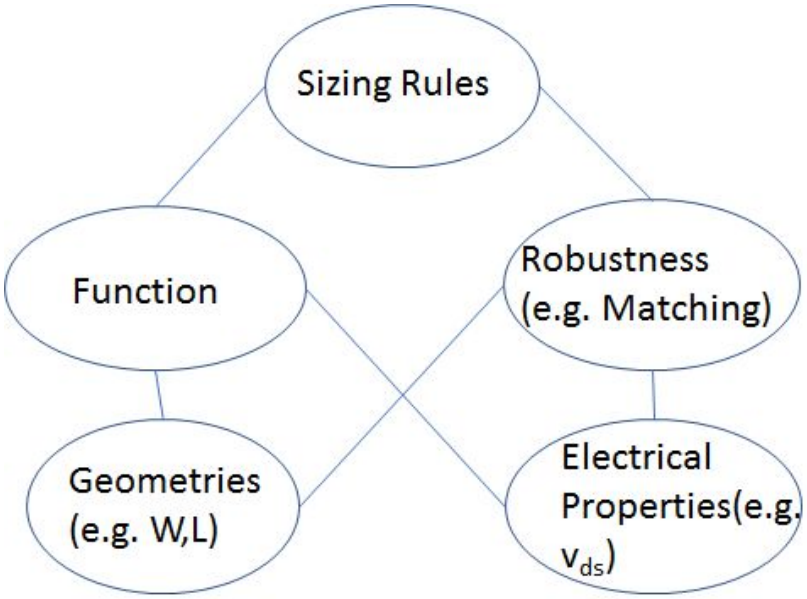


Fig. 1. Types of Sizing Rules

Analog components are an important part of integrated systems be it elements and area in mixed-signal systems. Even though its design is more important than digital circuits, the design automation of analog circuits still lags behind. Now circuit synthesis is not an easy task as it involves topology, layout synthesis but also component sizing and many other physical effects like process variations, variations of operating conditions, matching constraints or noise. Sizing tasks are proven to be a very important for providing automation support to designer.

Often, the resulting circuit after specifying circuit performance bounds performs regularly in nominal case but shows exhibits increased sensitivity to process and operating variations and to noise. So sizing rules for the transistor geometry or transistor voltages are considered.

In this project, we have implemented a technique to obtain sizing rules for CMOS and BJT by using a heuristic methodology for the resolving the assignment ambiguities.

Related Work

In the past, analogy synthesis, nominal design optimization and sizing with respect to tolerances were the focus of research interest. These approaches included equation-based methods like GPCAD or AMGIE. Here the design equations were derived with the help of symbolic analysis and simulation-based methods like ASTRX/OBLX.

Most approaches for analog synthesis mention, design-space constraints, dimensional constraints, manufacturability and operationality constraints, component constraints, or “soft constraints. Previously approach has presented a detailed automatic construction sizing rules for CMOS transistor circuits.

System Design

This section details about the development of a hierarchical library of basic building blocks for CMOS and bipolar technology based on transistor pairs.
This is followed by a procedure for the automatic hierarchical recognition of these building blocks.

- ***Hierarchical Building Block Library***

This is a library which shows the hierarchical libraries of the basic building blocks for CMOS and bipolar transistor technologies. This whole library is a combination of both libraries which is an ordered set. It is divided into subsets which shows the hierarchical level each. Now, the most elementary elements form the lower most level of the library(here, L0) is shown at the top of all the levels.
We can see in the figure 2 that the at lower hierarchy level L0 lies VCCS and VCRES. The next level L1 element, Level Shifter and Simple Current Mirror are developed using the L0 level elements. This is followed by the level, L2 element, Cascode Current Mirror which consists of level L1 elements mentioned above.
The components of transistor pairs are numbered as (1) and (2). Number (1) refers to the left or upper transistor, number (2) on the right or lower one.
A similar approach is followed for bipolar building blocks. From figure 3, at level 0, a transistor in forward active region is placed followed by six transistor pairs at level 1. Again in level 2, current mirrors are present followed by a Differential Stage (DS) at level 3 which consists of the elements present at previous levels.
Block schematics and sizing rules are given for NMOS and n-p-n parts and are analogous for the PMOS and p-n-p parts. This hierarchy of building blocks is obtained from the structural property.
This property states that basic functions are realized based on the transistor pairs, groups of transistor pairs combined with single transistor.
Now these described libraries will be used for formulating sizing rules to the sizing part of analog synthesis.
A procedure is provided that searches a given circuit schematic to identify all the building blocks in the circuit.
The sizing rules is instantiated for identifying the building block.
So basically this model is very useful as it enables automatic recognition within a circuit structure and thus assignment of sizing rules. The next section introduces a procedure for the automatic recognition of building blocks. It includes the arbitration of assignment ambiguities (which will be discussed in later sections).

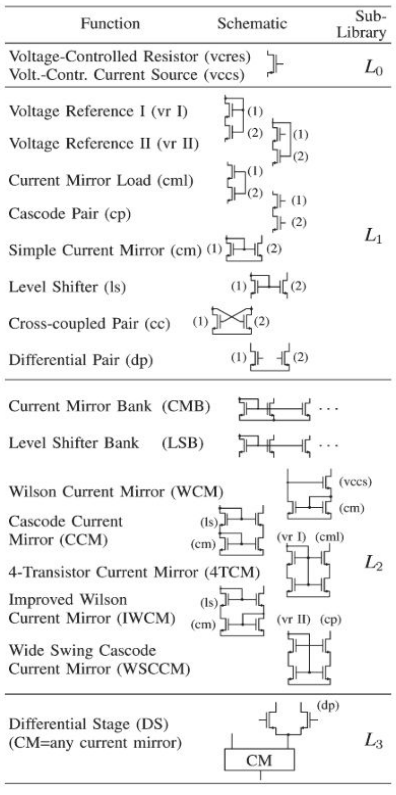


Fig. 2. Hierarchical library of basic CMOS building blocks (NMOS).

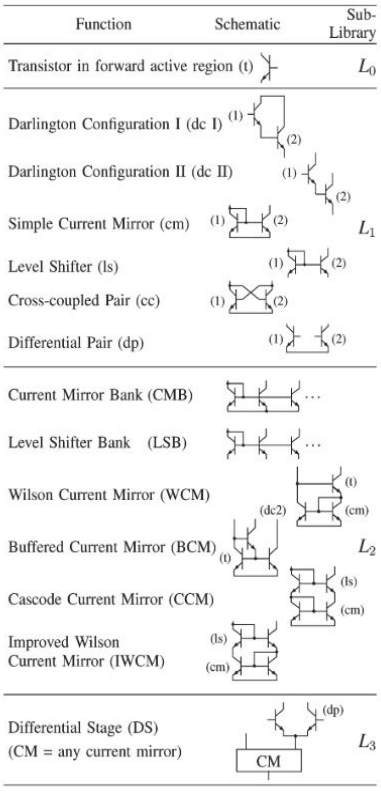


Fig. 3. Hierarchical library of basic bipolar building blocks (n-p-n).

- ***Automatic Hierarchical Recognition of Building Blocks***

This recognition algorithm has been introduced for the automatic hierarchical recognition of the basic building blocks for CMOS and bipolar technology. This is done by detecting using the respective library elements by going through the hierarchy from the building blocks on low levels to more complex ones on higher levels. As explained in previous section, the libraries are created using the basic building blocks like transistor types, voltage reference, current mirrors, differential pair at each level of the library.

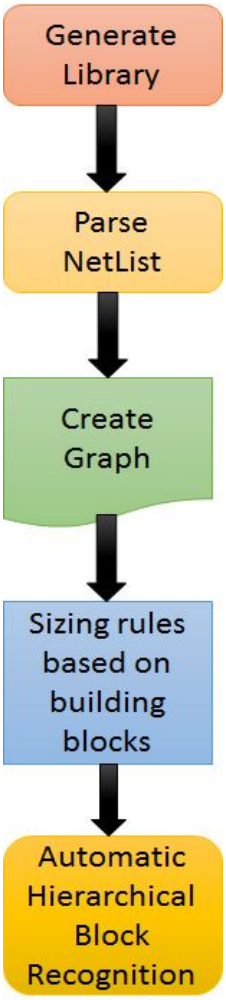
A mathematical representation of the presented building block library is made. The building blocks are grouped into different hierarchy levels with a single transistor at the bottom. A set is made which contains all the building blocks on every level. Then a set is created to classify the building blocks which includes a set of the transistor types and the set of structural types of building blocks.

After this a representation of a circuit netlist and its contents is shown. This netlist contains elements in a circuit and their pin connections. The building blocks in the hierarchy library is called a "module". So a new set is made which includes all these modules.

Next, a set is made for the nets connecting the modules and a set consisting of the pins. Using these sets a new set is made representing a circuit netlist.

Algorithm & its Software Implementation

Below are the steps involved in the algorithm mentioned in the flowchart along with its description.



a) Generate Library -

A library for basic building blocks is generated hierarchically from the circuits present in the paper. Ambiguities occur in building block selection. So, to remove these ambiguities, Priority values are obtained from the text file which contains the level of the circuit and the priority of it in the level. This value is then stored in a HashMap in the form of *HashMap<Priority,String>* where, Priority class stores the *level* and *pValue* (priority value) of the circuit.

Below are the code snippets for Priority Class and mentioned HashMap:

```
class Priority {
    int level;
    int pValue;

    Priority(int level,int pVlaue) {
        this.level = level;
        this.pValue = pValue;
    }
}
```

Figure 4. Flowchart of algorithm

HashMap implementation -

```
public class CreateLibrary {

    public Map<Priority,GenerateGraph> library;

    CreateLibrary() {
        library = this.createLib(this.library());
    }

    public Map<Priority,GenerateGraph> createLib(Map<Priority,String> libFileNames) {

        Map<Priority,GenerateGraph> map = new HashMap<>();

        for(Map.Entry<Priority,String> entry : libFileNames.entrySet()) {
            if(!map.containsKey(entry.getKey())) {
                map.put(entry.getKey(),new GenerateGraph(entry.getValue()));
            }
        }

        return map;
    }
}
```

b) Parse NetList -

At this stage, the circuit file is parsed. This parsing gives us the details about :

- Components present in the circuit
- The type of component.
- The Connections and Nets in the circuit.
- List of Terminal namely, Gate, Source and Drain.

Below is the code snippet of Parsing technique and the constructor to obtain the data from the file-

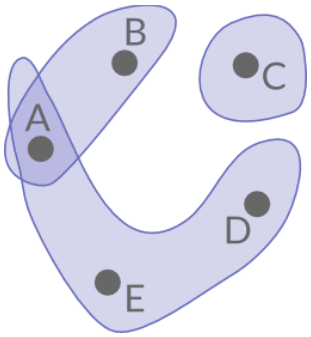
```
try {
    fis = new FileInputStream(fileName);
    br = new BufferedReader(new InputStreamReader(fis));
    String line = br.readLine();
    while(line!=null) {
        // System.out.println(line);
        if(line.length()==0) {
            line = br.readLine();
            continue;
        }
        Pattern p = Pattern.compile("[a-zA-Z]");
        Matcher m = p.matcher(String.valueOf(line.charAt(0)));
        if(!m.matches()) {
            line = br.readLine();
            continue;
        }
        String[] parts = line.split(" ");
        if(parts.length<7) {
            line = br.readLine();
            continue;
        }
        components.add(new Component(parts[0],parts[1],parts[2],
        parts[3],parts[4],parts[5],Double.parseDouble(parts[6].split("=")[1]),
        Double.parseDouble(parts[7].split("=")[1])));
        line = br.readLine();
    }
}
```

This snippet shows the constructor :

```
Component(String name,String source,String drain,String gate,String body,String type,double length,double width) {
    this.name = name;
    this.source = source;
    this.drain = drain;
    this.gate = gate;
    this.body = body;
    this.type = type;
    this.length = length;
    this.width = width;
}
```

c) Generate Graph -

Here we create graphs of the circuit given in the circuit file. The components are Nodes of the graph. Now the connections from the terminal of the nodes leads to multiple edges from a single Node. Thus we need a different data structure to represent this net. For this a complex data structure called, **Hyper Graph** is used.



A hypergraph is a graph in which generalized edges (called hyperedges) may connect more than two nodes. It consists of a bunch of nodes and a bunch of edges and each edge can connect any number of nodes (Edges may be called hyperedges in this context).
Here the edges are represented by areas and not lines connecting the vertices.

The Hyper Graph in this case represents a HashMap of the components and a list of nets in the circuit.

Figure 5 : five vertices but three hyperedges.

Below is the code snippet of Hyper Graph defined as per our requirement-

```
public class GenerateGraph {
    List<Component> components;
    List<String> netList;
    Map<String,List<String>> netListMap;
    Map<String,List<String>> componentMap;

    GenerateGraph(String fileName) {
        components = ParseComponents.parseNetlist(fileName);

        netList = createNetList(components);
        netListMap = generateNetListMap(components,netList);
        componentMap = generateComponentMap(components,netList);
    }
```

There is another alternative available but there are some limitations to it.
By considering the Components as the edges and the Nets as the nodes in the graph,

- The component has 4 terminals and thus the number of edges will increase drastically. Due to this the graph will become very dense.
- The number of nodes will also increase if the circuits involve many connections.

This creates the graph node for the circuit from the basic building blocks. The graph is created for the above parsed circuit and the library created for all the circuits. This graph is stored in a HashMap. The key is specified by the Priority C. Next step is to create a graph of components. A graph is obtained from the above value(of String type) stored in HashMap. This is finally stored in another HashMap. The key is specified by the Priority class (same as mentioned above) and value is specified by the graph of components.

d) Automatic Hierarchical Block Recognition -

This recognition algorithm has been introduced for the automatic hierarchical recognition of the basic building blocks for CMOS and bipolar technology.

Below are some **limitations** with other algorithms:

- Topological Sorting cannot be used where the graphs are sorted and easier to recognize.
- Next, typical graph search algorithm cannot be used like Breadth First Search and Depth First Search as the circuit can grow its complexity.
- Floyd-Warshall algorithm cannot be used as there are hyper edges present so there won't be any added advantage.

Challenges faced:

- Due to above limitations, it becomes difficult to design an algorithm to search the building blocks.
- The Arbitration of assignment ambiguity is a major problem which needs to be addressed. It occurs when there are sub circuits present in the circuit.

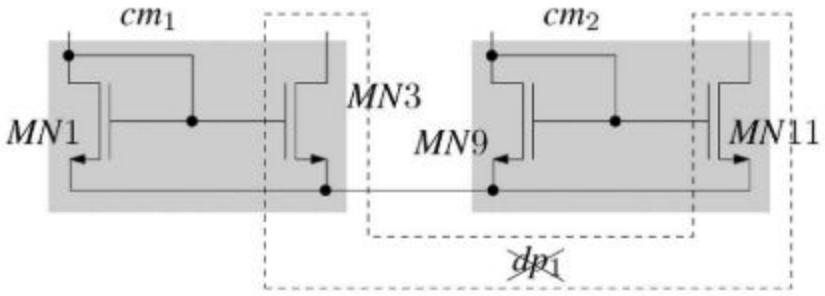


Fig 6. Two simple current mirrors preferred over a dp.

In the example above, transistors MN1 and MN3, MN9 and MN11 form a simple current mirrors are recognized by the recognition algorithm. But an additional Differential Pair consisting of MN3 and MN11 is detected as well since these two transistors are connected via their source pins.
But a transistor cannot be part of a simple current mirror and a DP at the same time.

e) Sizing rules based on building blocks -

Sizing rules are mentioned for the basic CMOS or bipolar building blocks. These rules use the details mentioned in figure 1, namely Function and Robustness. A rule is labeled as FG or FE if its Geometric or Electrical constraint concerning Function. Also labeled as RG or RE if its Geometric or Electrical constraint concerning Robustness. Here, CMOS sizing rules are discussed.

Basically, a CMOS transistor behaviour is obtained from Shichman–Hodges model where its drain current is shown by :

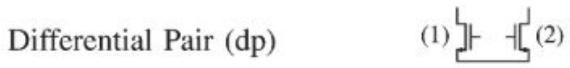
$$i_d = \begin{cases} 0, & \text{if } v_{\mathrm{gs}} \leq 0 \\ K\frac{W}{L}\left[(v_{\mathrm{gs}} - V_{\mathrm{th}}) - \frac{v_{\mathrm{ds}}}{2}\right] \cdot v_{\mathrm{ds}}(1 + \frac{\lambda}{L}v_{\mathrm{ds}}), & \text{if } 0 \leq v_{\mathrm{ds}} < v_{\mathrm{gs}} - V_{\mathrm{th}} \\ \frac{1}{2}K\frac{W}{L}(v_{\mathrm{gs}} - V_{\mathrm{th}})^2 \cdot (1 + \frac{\lambda}{L}v_{ds}), & \text{if } v_{\mathrm{gs}} - V_{\mathrm{th}} \leq v_{\mathrm{ds}}. \end{cases}$$

W, L : transistor's width and length.
K= $\mu_{Si}C_{ox}$: μ_{Si} is the electron mobility and C_{ox} is the oxide capacity.
V_{th} : threshold voltage
λ : channel length modulation coefficient

Here the design parameters are the transistor geometries W and L.

Now, for example let's discuss the sizing rules for a **Differential Pair** in CMOS and Bipolar Transistor Building Blocks.

Differential Pair in CMOS -



A dp is used to convert the gate-source voltage difference into source voltage.

Figure 7. A differential pair

Both transistors work in VCCS to reduce the effect of drain source voltage. The FE and FG of Current Mirror is same for a Differential Pair. In this configuration, both transistors must have equal widths and lengths.
The drain-source current must linearly change with the gate-source voltage difference. But this difference may not exceed a certain value, RE.

Below conditions must be met for proper operation of a DP.

$$\begin{aligned} \text{FG1}: &\quad l_1 = l_2 \\ \text{FG2}: &\quad w_1 = w_2 \\ \text{FE}: &\quad |v_{\text{ds}_2} - v_{\text{ds}_1}| \leq \Delta V_{\text{ds}_{\text{max(dp)}}} \\ \text{RE}: &\quad |v_{\text{gs}_2} - v_{\text{gs}_1}| \leq \Delta V_{\text{gs}_{\text{max}}}. \end{aligned}$$

Differential Pair in BJT-

A dp in BJT is used to produce a difference in the collector currents i_{c1} and i_{c2} which is dependent on the base-emitter voltages of the two transistors.

Figure 8. A differential pair

For operation in forward active region, base currents must be small for both the transistors and their β must be maximum.
Also for symmetry, the number of transistors in parallel on the both sides must be the same and the difference of collector-emitter voltages must be small to reduce the early effect.
And the difference of base-emitter voltages must be small.

Below conditions must be met for proper operation of a DP.

$$\begin{aligned} \text{FG}: &\quad N_1 = N_2 \\ \text{FE}: &\quad |v_{\text{ce}_2} - v_{\text{ce}_1}| \leq \Delta V_{\text{ce}_{\text{max(dp)}}} \end{aligned}$$

Above conditions are represented in mathematical equations -

$$\text{RE}: \quad |v_{\text{be}_2} - v_{\text{be}_1}| \leq \Delta V_{\text{be}_{\text{max(dp)}}}$$

Experimental Results

Here, the experiment and results are mentioned. A circuit file is generated for the circuit below which is a folded cascode OpAmp.

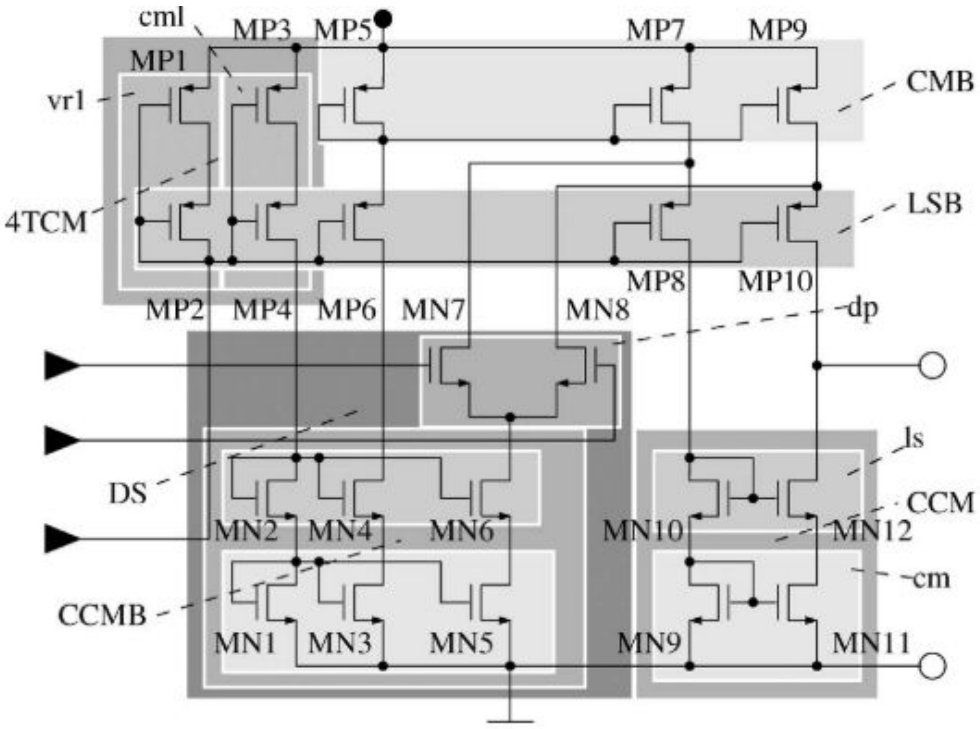


Figure 9. Detected building blocks in a folded cascode OpAmp.

Below is a screenshot of the circuit file obtained from the circuit above -

```
** Generated for: hspiceD
** Generated on: Mar 22 21:00:37 2017
** Design library name: SPICE_Test
** Design cell name: Folded_Cascode_OpAmp
** Design view name: schematic
.GLOBAL vdd


.TEMP 25.0
.OPTION
+    ARTIST=2
+    INGOLD=2
+    PARHIER=LOCAL
+    PSF=2

** Library name: SPICE_Test
** Cell name: Folded_Cascode_OpAmp
** View name: schematic
MP1 net1 net2 net3 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP2 net2 net4 net3 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP3 net1 net5 net7 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP4 net5 net18 net7 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP5 net1 net6 net9 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP6 net8 net10 net18 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP7 net1 net12 net9 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP8 net12 net13 net14 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP9 net1 net15 net16 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MP10 net15 net17 net18 vdd PMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN7 net19 net12 net20 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN8 net21 net15 net22 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN2 net23 net4 net24 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN1 net25 net23 net26 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN4 net27 net10 net24 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN3 net28 net27 net26 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN6 net29 net19 net24 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN5 net32 net29 net28 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN10 net30 net13 net13 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN9 net31 net30 net30 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
MN12 net32 net17 net13 0 NMOS L=50e-9 W=90e-9 AD=9.45e-15 AS=9.45e-15 PD=300e-9 PS=300e-9 M=1
```

Below are the screenshots of the Results -

Following is the obtained graph from the circuit above.

```
[prathakrastogi CAD Proj 1 $ javac GenerateGraph.java
[prathakrastogi CAD Proj 1 $ java GenerateGraph
[net33, net10, net21, net32, net13, net12, net23, net4, net2, net31, net1, net30, net8, net6, net5, net19, net18, net29, net15, net25, net17, net
28, net27, vin, vout, 0, vdd]
{net33=[MN11.source], net10=[MP6.drain, MN4.drain], net32=[MN5.source, MN12.source, MN11.drain], net13=[MP8.drain, MN10.drain, MN10.gate, MN12.ga
te], net12=[MP7.drain, MP8.source, MN7.drain], net4=[MP2.drain, MN2.drain], net2=[MP1.drain, MP2.source], net31=[MN9.source], net1=[MP1.source, M
P3.source, MP5.source, MP7.source, MP9.source], net30=[MN10.source, MN9.drain, MN9.gate, MN11.gate], net8=[MP6.source], net6=[MP5.drain], net5=[M
P3.drain, MP4.source], net29=[MN6.source, MN5.drain], vin=[], net25=[MN1.source], net28=[MN3.source, MN5.gate], net27=[MN4.source, MN3.drain], ne
t21=[MN8.source], net23=[MN2.source, MN1.drain], vout=[], 0=[MN7.body, MN8.body, MN2.body, MN1.body, MN4.body, MN3.body, MN6.body, MN5.body, MN10
.body, MN9.body, MN12.body, MN11.body], vdd=[MP1.body, MP2.body, MP3.body, MP4.body, MP5.body, MP6.body, MP7.body, MP8.body, MP9.body, MP10.body]
, net19=[MN7.source, MN6.drain], net18=[MP4.drain, MP6.gate, MP10.gate], net15=[MP9.drain, MP10.source, MN8.drain], net17=[MP10.drain, MN12.drain
]}
{MN10=[0, net13, net30], MN11=[net33, 0, net32, net30], MP10=[vdd, net18, net15, net17], MN12=[0, net32, net13, net17], MP2=[net4, net3, net2, vd
d], MP1=[net3, net2, net1, vdd], MP4=[vdd, net18, net7, net5], MN2=[0, net24, net23, net4], MP3=[net1, vdd, net7, net5], MN1=[0, net23, net26, ne
t25], MP6=[net10, vdd, net8, net18], MN4=[0, net10, net24, net27], MP5=[net1, vdd, net6, net9], MN3=[0, net26, net28, net27], MP8=[net13, net12,
vdd, net14], MN6=[0, net24, net19, net29], MP7=[net12, net1, vdd, net9], MN5=[0, net32, net29, net28], MN8=[net22, 0, net21, net15], MP9=[net1, v
dd, net15, net16], MN7=[0, net12, net20, net19], MN9=[0, net31, net30]}
prathakrastogi CAD Proj 1 $
```

Following is the output after matching the modules which shows the Levels and the Sizing Rules of the circuit.

```
[prathakrastogi CAD Proj 1 $ javac MatchModule.java
[prathakrastogi CAD Proj 1 $ java MatchModule
Circuits found:
2 1 Cascode Current Mirror
3 1 Differential Stage
1 1 Level Shifter
2 2 Current Mirror Bank
2 3 4-Transistor Current Model
1 2 Current Mirror Load
1 3 Differential Pair
1 4 Simple Current Mirror

Sizing Rules:

Cascode Current Mirror
FG1 : w1ls=w1cm
FG2 : w2ls=w2cm
Level Shifter
FG : l1=l2
RE : Vgs(1,2)-Vth(1,2)>=Vgsmin
4 Transistor Current Mirror
FG1 : wvrl=wvrl
FG2 : wcm=wcml
FE : |Vds_vrl(1,2)-Vds_dscml|<=Vdsmax

Differential Pair
FG1: l1=l2
FG2: w1=w2
FE: |Vds2-Vdsmin|<delVdsmax
RE: |Vgs2-Vgs1|<delVgsmax
```

Conclusion

The length and width of transistors can be incorporated and optimized using Genetic Algorithm in order to create optimized sub circuit modules.

Finally, the algorithm mentioned in the paper was implemented and was used to obtain the various modules present in the circuit, which was printed with level and priority level. Various sizing rules are also shown with respect to various circuit modules found in the given circuit.

References

- T. Massier, H. Graeb, and U. Schlichtmann, “The sizing rules method for CMOS and bipolar analog integrated circuit synthesis,” IEEE Transactions CADICS, Vol. 27, No. 12, pp. 2209–2222, Dec. 2008.
- Hypergraph - https://en.wikipedia.org/wiki/Hypergraph
- A brief primer on Hypergraph - https://esham.io/2012/09/olympic-colors
- Implementation of Hypergraph - http://jung.sourceforge.net/doc/api/edu/uci/ics/jung/graph/Hypergraph.html