3-17-1995

Automatic Synthesis of VLSI Layout for Analog Continuous-time Filters

David Lyle Robinson
Portland State University

THESIS APPROVAL

The abstract and thesis of David Lyle Robinson for the Master of Science in Electrical Engineering were presented March 17, 1995, and accepted by the thesis committee and the department.

COMMITTEE APPROVALS: [redacted]

Dr. W. Robert Daasch, Chair

[redacted]

Dr. Rolf Schaumann

[redacted]

Mr. Jack Pippin, guest industry appointee

[redacted]

Dr. Thomas Schubert
Representative of the Office of Graduate Studies

DEPARTMENT APPROVAL: [redacted]

Dr. Rolf Schaumann, Chair
Department of Electrical Engineering

****

ACCEPTED FOR PORTLAND STATE UNIVERSITY BY THE LIBRARY

by [redacted] on 28 July 1995

ABSTRACT

AN ABSTRACT OF THE THESIS OF David Lyle Robinson for the Master of Science in Electrical Engineering presented March 17, 1995.

Title: Automatic synthesis of VLSI layout for analog continuous-time filters.

Automatic synthesis of digital VLSI layout has been available for many years. It has become a necessary part of the design industry as the window of time from conception to production shrinks with ever increasing competition. However, automatic synthesis of analog VLSI layout remains rare.

With digital circuits, there is often room for signal drift. In a digital circuit, a signal can drift within a range before hitting the threshold which triggers a change in logic state. The effect of parasitic capacitances for the most part, hinders the timing margins of the signal, but not its functionality. The logic functionality is protected by the inherent noise immunity of digital circuits.

With analog circuits, however, there is little room for drift. Parasitic influence directly affects signal integrity and the functionality of the circuit. The underlying problem automatic VLSI layout programs face is how to minimize this influence.

This thesis describes a software tool that was written to show that the minimization of parasitic influence is possible in the case of automatic layout of continuous-time filters using transconductance-capacitor methods.

AUTOMATIC SYNTHESIS OF VLSI LAYOUT FOR ANALOG CONTINUOUS-TIME FILTERS

by
DAVID LYLE ROBINSON

A thesis submitted in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

Portland State University

1995

ACKNOWLEDGMENTS

I would like to acknowledge the previous work done on this project by both Martine Wedlake and Dr. W. Robert Daasch. They were the real pioneers in this tool. I was fortunate to be the one to inherit the work in progress. I would also like to thank Dr. Daasch for his patience and perseverance on my behalf, his timely suggestions, and his willingness to nudge me along when I've stumbled.

I would like to acknowledge and thank the National Science Foundation for their financial support during the project.

I would like to thank my loving wife and editor for her devotion, support, understanding and hard work throughout this whole project. It has been a long and weary road that she has tarried with me. She definitely wins the WIFE OF THE YEAR award! I love you Dawn Denice!!

And finally, I would like to thank the source of my inspiration and strength, my Heavenly Father.

"Give thanks to the Lord, call on His name;
make known among the nations what He has done.
Sing to Him, sing praise to Him;
tell of all His wonderful acts.
Glory in His holy name;
let the hearts of those who seek the Lord rejoice.
Look to the Lord and his strength;
seek his face always.
Remember the wonders He has done,
His miracles, and the judgments He pronounced."
-I Chronicles 16:8-12, New International Version Bible

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

Design automation software came about because of the need to become more aggressive and competitive in a tight market. Increasing demands on an engineer's resources provoked research into productivity tools. The sometimes mundane task of layout was a prime candidate for automation. Initially, layout was accomplished by creating a text file of numerical coordinates describing plot locations for layers of metal. The first step towards improving layout efficiency was the introduction of graphical oriented layout tools such as Magic[13]. Pushing still further, designers began to reuse layout of some of the most common circuit elements. Terminology began to evolve to fit the practice. "Full custom design" meant everything was designed by hand from the ground up. A good example of this is the Zilog Z-80 microprocessor. A visual inspection of the chip shows very little regularity.

Contrast this with the Intel 80386 which came about a decade later through a "semi-custom" methodology. Here, a library of standard cells was developed and reused wherever possible. From a design standpoint, regularity and reuse were key to the completion of the processor.

Now, with the advent of hardware description languages, standard cells can be automatically selected from a library and arranged by a

computer following a digital designer's software instructions. The process where a computer can perform the layout task is called design automation and exists in many forms. However, currently the best successes only exist for digital circuits.

This thesis will discuss the current forms of design automation and describe a specialized *analog* layout tool, OTACC (Operational Transconductance Amplifier - Capacitance Compiler). This software tool is the focus of a research project in automating the layout of analog circuits while minimizing the effect of parasitic capacitances.

THE PARASITIC CAPACITANCE PROBLEM

When a design is placed in silicon, parasitic capacitances are an unavoidable annoyance. These capacitances generally result from sandwiching a layer of oxide between a layer of metal and the substrate. Several consequences can result from the parasitic capacitances: compromised timing, charge-sharing, and cross-coupling are some of the most problematic. There are two basic types of circuits: analog and digital. Parasitic capacitances affect each of them with different levels of severity, since analog signals are typically measured in terms of millivolts, while digital signals are measured in terms of volts.

Analog Parasitics

With three orders of magnitude difference in the scale of the measurement, it is easy to see that analog circuits are typically more sensitive to parasitic deviations than digital circuits. As an example, suppose we wanted to design the fifth-order inverse Chebyshev lowpass filter with a 15MHz passband and greater than 51 dB of attenuation in the stopband shown in Figure 1.

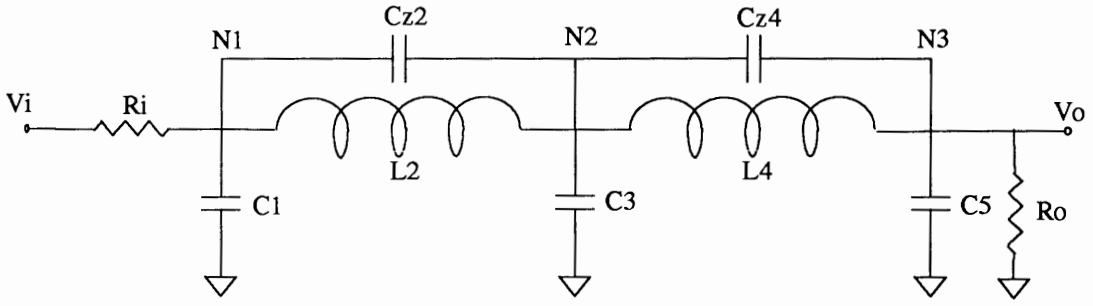


Figure 1. LC fifth-order filter

where,

C1 = 576.741 fF	L2 = 77.951 uH
Cz2 = 226.792 fF	L4 = 86.330 uH
C3 = 2.433 pF	Ri = 6.67 k Ohms
Cz4 = 78.219 fF	Ro = 6.67 k Ohms
C5 = 718.833 fF	

This produces the magnitude plot in Figure 2, with a stopband of -57 dB at > 25MHz.

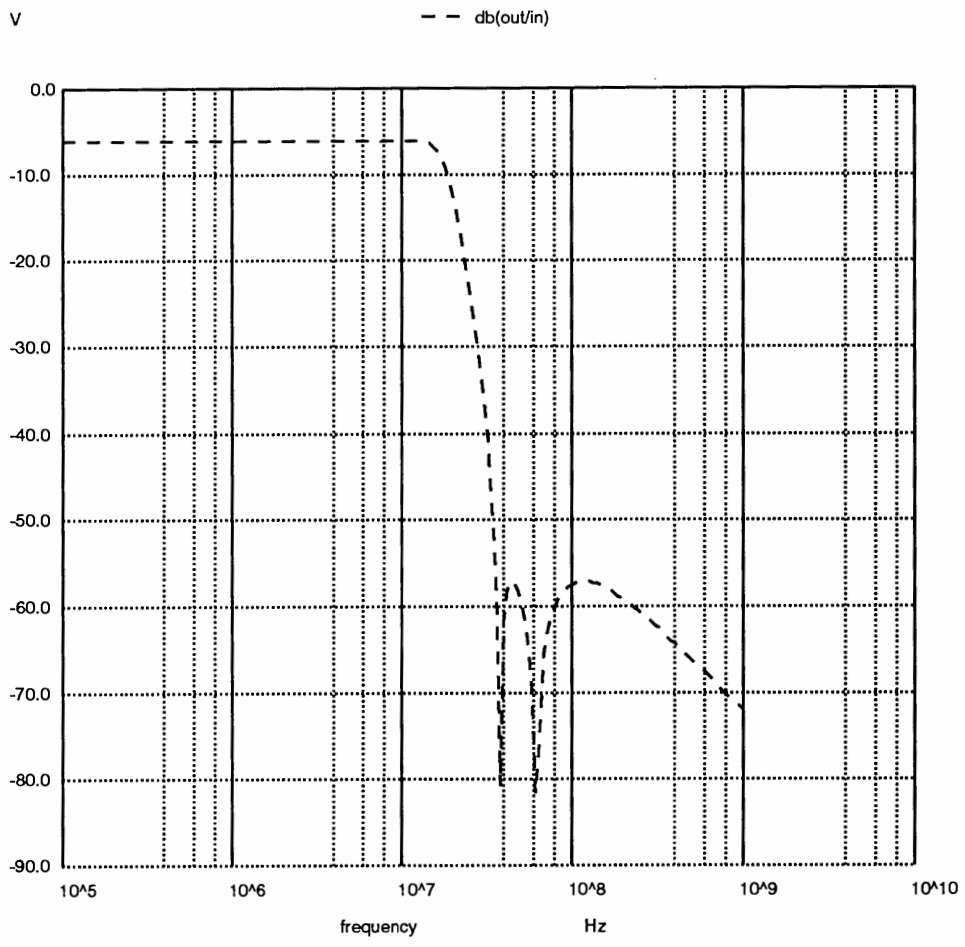


Figure 2. Fifth-order inverse Chebyshev filter magnitude plot - no parasitic capacitors

However, suppose that parasitic capacitors exist from nodes N1, N2, and N3 to ground. For example, inductors are not readily implemented in silicon, so they must be simulated. Parasitic input and output capacitance is normal for simulated inductors with active circuitry. A practical estimate might be a total of 920fF for the inductor I/O parasitic capacitance, and 80fF for the parasitic capacitance of the associated routing. This gives a total parasitic capacitance of 1pF. The modified circuit is shown in Figure 3.

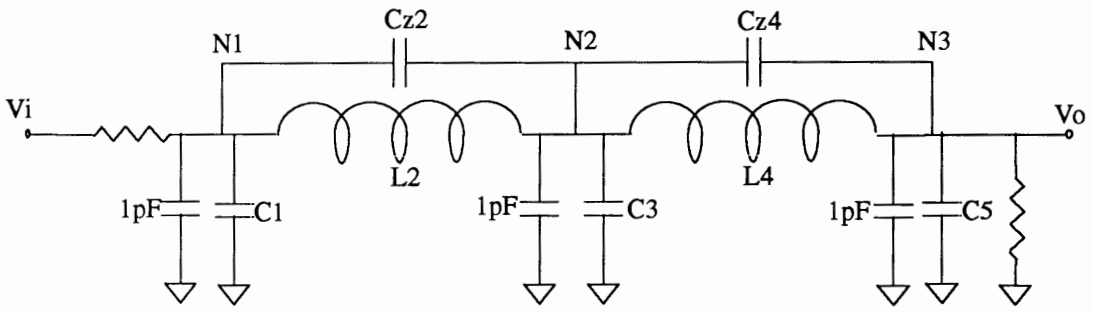


Figure 3. LC fifth-order filter with parasitic capacitors

The plot in Figure 4 results which shows a difference of 18dB in the stopband. In addition to the change in magnitude, there is also a phase difference in the filter response. This demonstrates that parasitic capacitances can have a dramatic effect on analog circuits.

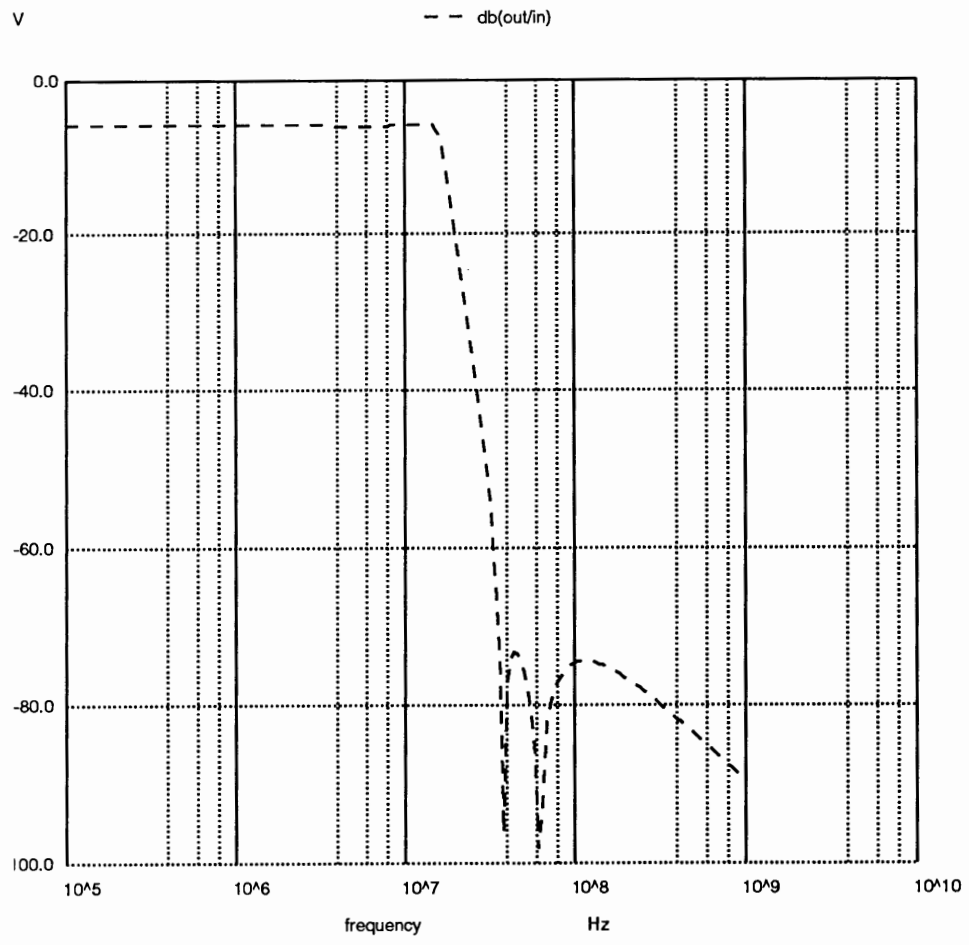


Figure 4. Fifth-order inverse Chebyshev filter magnitude plot with the parasitic capacitors shown in figure 3.

The use of the filter in this example was purposely chosen since this thesis later demonstrates that this same analog filter can be successfully built using the OTACC software without any significant changes in filter response due to the parasitic capacitors.

Digital Parasitics

In digital circuits, the most prominent effect of parasitic capacitances is time delay. Delays are often built into the design specification from the start. The engineer or automated design tool has a delay budget to track, and as long as the circuit does not exceed the budgeted delay specification, it remains functional. The following circuit demonstrates this phenomenon.

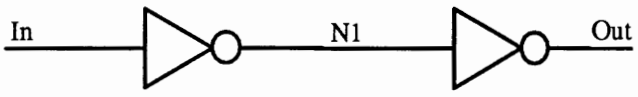


Figure 5: Inverter chain schematic

Ideally, the output of this circuit would track perfectly with the input. However, if we inject parasitic capacitances into a simulation at N1 and Out, a time shift is evident. Assuming minimum inverters (2 micron process), an output diffusion parasitic capacitance of 50fF would be reasonable. Add to this 50fF of routing parasitic capacitance to get 100fF capacitors to ground from nodes N1 and Out. The simulation shown in Figure 6 indicates a time shift of approximately 2ns.

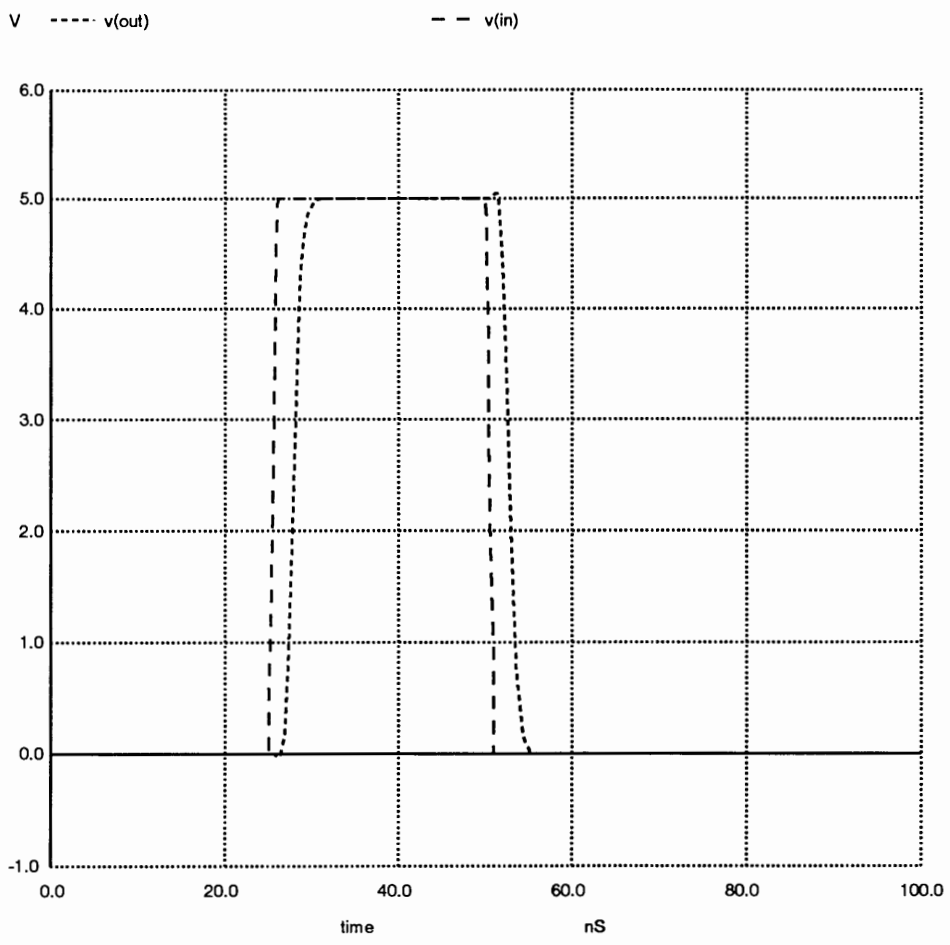


Figure 6. Inverter chain magnitude plot - with parasitic capacitors

SOLUTIONS TO THE PARASITIC CAPACITANCE PROBLEM

As demonstrated, parasitic capacitances affect both analog and digital circuits. Because of the extreme sensitivity of analog circuits, the parasitic capacitors in this realm are much more significant and pose a larger problem for design automation software. The real difference between analog and

digital design automation is how the software solves the parasitic capacitance problem.

Digital

The solution to most digital parasitic capacitance problems can be solved by adjusting the driver/receiver relationship and paying careful attention to common design and layout practice to minimize other circuit problems such as charge sharing and signal cross coupling from capacitance. In effect, a digital circuit can be designed with extra drive in order to compensate for the delays and signal degradation. The trade-off is an increase in power consumption. The bottom line is that a library of characterized standard cells can be built. These cells can be chosen to overdrive any unanticipated parasitic capacitance and can be dynamically routed to each other. The overdrive prevents the parasitic capacitance of the dynamic route from having any significant effect.

Analog

Analog circuits are not so simple. Resizing to increase DC current in an analog circuit usually has undesirable side effects which can be just as

bad as the problems caused by the parasitic capacitances. These side effects are often seen as clipping or a reduction in bandwidth.

In addition to the systematic parasitic capacitances such as those found in the inductor simulation, the use of uncharacterized dynamic routing can also introduce enough parasitic capacitance to affect an analog circuit's functionality. Therefore, an approach different from the digital solution is required. One way of dealing with the unwanted capacitors is to leverage them for a useful purpose. This can be accomplished by looking for ways to use the parasitic capacitor as part of a desired circuit capacitor.

For example, suppose a circuit capacitor is attached to a node where parasitic capacitances are expected. The original value of the capacitor is "adjusted" or predistorted to include the value of the parasitic capacitance. The result is a capacitor of the desired size on the node where before there would have been the desired capacitor plus the parasitic capacitor.

The disadvantage of this approach is that not all circuit configurations allow for predistortion to occur. Therefore, design automation software using this method is limited to the circuit configurations where predistortion is possible.

DESIGN AUTOMATION

There are a couple different ways to automate the layout process of digital and analog circuits. However, the way parasitic influences are handled is vital to the success of any analog design automation method.

The first method is polygon generation. In this method, every piece of layout is generated by the software. Each transistor is independently sized and routed to the next transistor for the entire design. This method has many variances that can be unforgiving in large analog sub-systems. As the complexity of the system increases, the errors begin to accumulate exponentially. However, there have been some successes with the automatic generation of small analog components[1-3] and digital systems.

The second method is module generation[2,4,5]. With this method, a library of basic building blocks containing modules such as logic devices, current mirrors, differential pairs, capacitors, etc. is available. These modules are preassembled in such a way as to minimize layout problems. The methodology is very general, but since the parasitic capacitances can still wreck havoc with analog configurations, a lot of designer intervention is required in the case of complex analog sub-systems[4-5]. Work is ongoing with this method and may one day be the preferred methodology if the parasitic problem can be solved.

A third method is tiling. This takes the idea of modules a step further in that the modules (tiles) are more complex and are configured such that the signal routing is also tiled. A library of pre-designed and characterized tiles are available as basic building blocks. Since routing is included in the tiles, the route operation is static. Routing occurs simply by abutting two or more tiles together [17, 19] as shown in Figure 7.

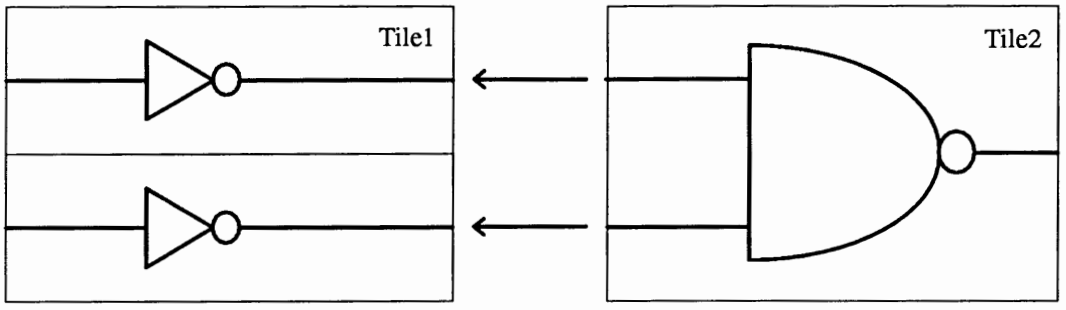


Figure 7. Inherent routing through tile abutment

The disadvantage to this approach is library maintenance. If additional circuitry is added to the library, often a set of support tiles will need to be created also. For the digital logic example in Figure 7, there are no specialized routing tiles required. However for something more complex such as a tunable OTA (Operational Transconductance Amplifier), additional support tiles are needed that match its layout. As seen in Figure 8, when using differential OTAs, multiplication by -1 can be obtained in a filter simply by crossing its inputs or outputs. This is an example where an additional route tile would be required that supported the OTA's layout.

So, not only would an additional tile need to be created, but this tile would need to be linked to a specific OTA in order to assure proper wire location (see Figure 8).

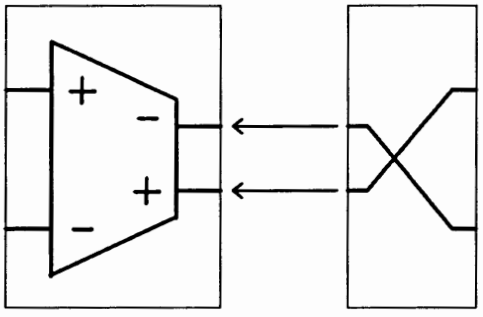


Figure 8. Example requiring a route tile

Tiling is the approach our analog CAD tool, OTACC, will take. Although less general than module generation, it uses a standard cell library and predistortion to provide better accuracy that would otherwise be unattainable. This is a reasonable tradeoff in the search for better analog automation tools. The specifics of how the tool works and how to develop the support library will be covered in a later section.

VLSI ANALOG FILTER BACKGROUND

Switched Capacitor

Integrated circuit switched capacitor filters evolved out of mature active-RC filter methods used extensively in the telecommunications industry. Figure 9 is an example of an active RC filter : a low-pass Butterworth using an operational amplifier, resistors, and capacitors.

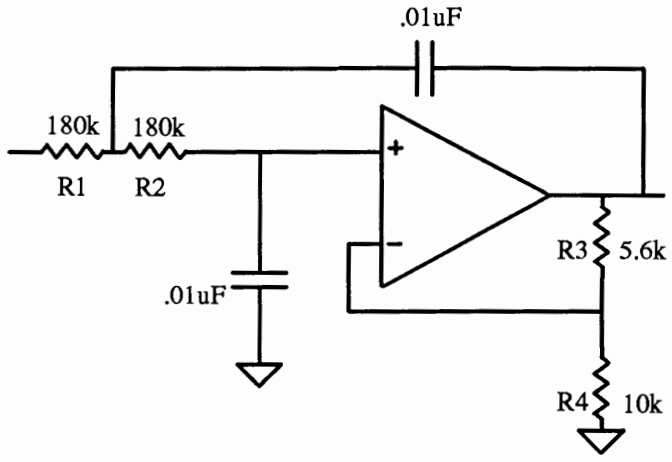


Figure 9. Active RC filter: lowpass Butterworth

In this case, the cutoff frequency was selected to be 88.4Hz. R1, R2 and the two capacitors are determined by the following formula:

$$RC = \frac{1}{2\pi f_c} \tag{1}$$

where R = R1= R2, and C represents both of the capacitors in the circuit. The remaining resistors, R3 and R4 are used in setting the amplifier gain, K: R3 = (K - 1) * R4.

Although operational amplifiers and capacitors are easily implemented in silicon, the large resistors (which are typically greater than 10k Ohms [6], and in this example exceed 100k Ohms) prove to be more difficult. However, by periodically switching the voltage on a capacitor between two circuit nodes, it has been found that the resistance can be simulated [7]. Filters that simulate the required resistances using this type of configuration are classified as "switched capacitor filters."

Switched capacitor filters are currently the most popular VLSI filter implementation. They work well in silicon, and there are several software layout solutions available [8-10]. Their primary shortfall is they only excel at lower frequencies because of the Nyquist criterion. Typically, switched capacitor filters are clocked on the order of 100 times faster than the analog signals of interest [6]. The other problem with switched capacitor filters is

that if a signal has a frequency greater than one half of the clock, this signal will be "aliased" by the filter.

Transconductance-C

Research has shown that transconductance-C filters are capable of frequencies to 100MHz [18]. In this methodology, four differential transconductances and two grounded capacitors (or one floating capacitor) are used to simulate an inductor. Since we have both inductors and capacitors available, continuous-time filters may be built using, for example, a standard LC ladder configuration. Continuous-time filters do not have the same signal bypass problem that switched capacitor filters are subject to.

II. SOFTWARE

OVERVIEW

OTACC initially was written by Martine Wedlake and Dr. W. Robert Daasch [14-16]. After receiving the initial prototype, my project was to add the necessary features to make OTACC a tool that could be successfully integrated into the filter design cycle. This involved changing the input interface, adding frequency transformation of normalized LC filters, permitting the use of grounded capacitors, and adjusting the layout size to fit real-world silicon proportions. OTACC as a whole, begininning with its conceptual foundations, is described in the following sections.

SYMBOLIC LAYOUT

OTACC uses the tiling methodology. Three types of tiles are required:

1) Transconductance tile
2) Capacitor tile
3) A set of route tiles

The route tiles act as an interface between the transconductance and an array of capacitor tiles. Typically, these three types of tiles fit together as shown in Figure 10.

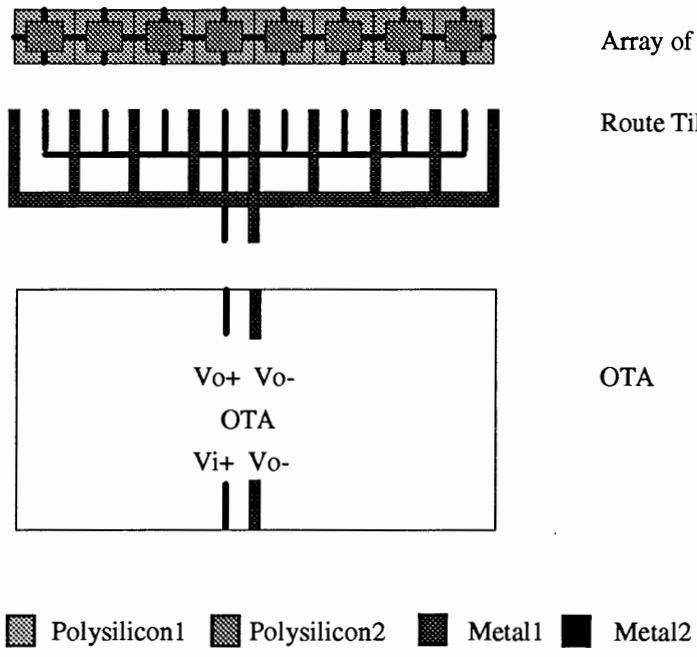


Figure 10. Capacitor comb route tile example - before tiling

Joining the tiles into one configuration gives a single transconductance routed to a single capacitor (made up of eight parallel unit capacitors).

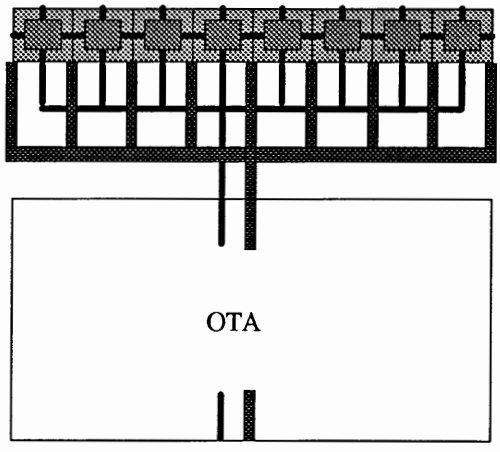


Figure 11. Capacitor comb route tile example - after tiling (see Figure 10 for legend)

By "routing" the cells with a predesigned tile, the desired amount of node capacitance can be predistorted according to the amount of parasitic capacitance. This greatly reduces the unpredictable parasitic capacitance problem that plagues automatic synthesis of VLSI analog circuitry. The disadvantage of routing with tiles is that the synthesis software becomes very specialized. The layout then needs to conform to a specific configuration. For the OTACC project, we wanted to synthesize transconductance-C filters. Since this is a very specialized area, having specialized software is not a problem.

OTACC is based on the popular (manual) filter realization method using Signal Flow Graphs (SFGs). This method is described in [18] for designing continuous-time filters using passive ladder configurations. An SFG borrows the terminology of inversion and addition from mathematics to graphically describe a filter's input immitance using a continuous fraction. The designer may use filter design software such as FiltorX [12] in the initial design stages to create the LC ladder which can be mathematically represented by a continuous fraction that describes the input impedance of the filter.

$$Z = \cfrac{1}{s\hat{C}_1 + \cfrac{1}{s\hat{L}_2 + \cfrac{1}{s\hat{C}_3} \ldots}}$$

Once a continuous fraction is obtained, it may be used as one form of input for OTACC. The input grammar used is described in [19]. Continuing with the third-order elliptic filter example, the input stream would look like Figure 12:

$$[0.0, C_1[0.0, L_2[0.0, C_3]]]$$

Figure 12: Original OTACC grammar

In this grammar, each bracket simultaneously represents the only two operations required to build the heart of the filter: addition and inversion. The pair of comma separated numbers represent an ordered pair where the real part is zero (OTACC currently supports only lossless filters). Processing begins at the end of the input stream and recurses back towards the beginning of the input stream. Inversion and addition are indicated by the presence of a left square bracket.

Once the filter is laid out, an input source transformation is added to the filter's input, and a load resistor is added to the filter's output. In addition, tank capacitors are only added to the filter's layout after the rest of the filter is complete.

CFL

OTACC can manage a VLSI layout database by using a library of routines developed at the University of Washington called "CFL," or "Coordinate Free LAP" [11]. These routines provide basic functions that allow reading and writing tiles to disk, edge-to-edge placement, re-orientation, and mirroring. In addition, two simple routers and basic polygon generation are provided. A short example shows how two pieces of layout can be placed next to each other.

Suppose we have the following pieces of layout:

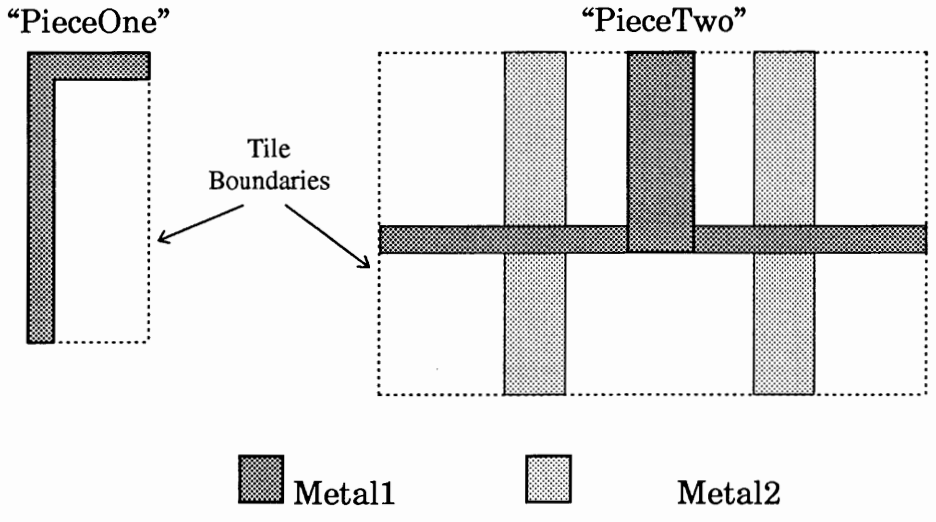


Figure 13. Example layout pieces

The following code segment would tile these together with the wires aligned:

```
{
SYMBOL *ptr1;
SYMBOL *ptr2;
SYMBOL *combination;
ptr1=gs("PieceOne");
ptr2=gs("PieceTwo");
combination = bx(ptr1, ptr2);
}
```

In this segment, the first three lines declare the variables to use, the fourth and fifth lines read the "PieceOne" and "PieceTwo" tiles from disk, and the sixth line combines the tiles together into a third tile named "combination." This results in the configuration shown in Figure 14.

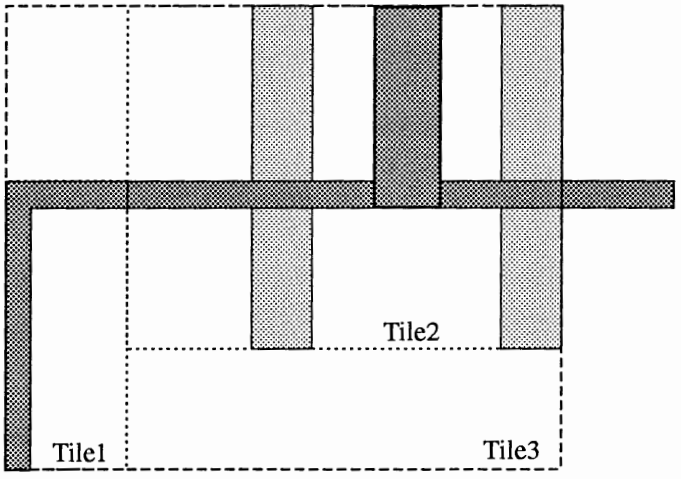


Figure 14. Example layout pieces combined (see Figure 13 for legend).

In this example, the routine, bx( ) was used to line the tiles up according to the wires placed at the right of "PieceOne" and the left of

"PieceTwo". Being a coordinate-free layout program, CFL aligns tiles based on the layers of metal that intersect the tile borders. Therefore, the bx() operation in this example aligned the tiles based on the common metal1 layer that it found at both tile interface borders. Additionally, there are other routines that can line the tiles up relative to certain boundaries such as bb() which stands for "bottom edge to bottom edge." With the addition of routing primitives, this layout language is quite powerful. The database is stored in the same format as the popular public domain VLSI CAD package, Magic.

OTACC

The communication structure of OTACC is based on a single stack. All operations obtain their input from the stack, and then push results back on the stack. Each stack element is a structure of pointers to layout symbols and miscellaneous information for communication between functions. A stack is preferred because to pass information back and forth between the routines, a large amount of global variables would be required. Also, using a stack machine allows each tile to be read from the disk one time, with the tiles remaining in memory for the duration of the layout. This improves the efficiency of OTACC since multiple disk reads for each tile can be avoided. An idea of the number of global variables that would have been required if a stack machine were not used can be obtained by viewing the stack element in Figure 15.

```
/* typedef the basic stack element

 */
typedef struct Stackelement *sptr;
typedef struct Stackelement {
  filterkeys type;
  SYMBOL *current;     /* Current filter */
  SYMBOL *cell;        /* Transconductance Tile*/
  SYMBOL *Gf;          /* "forward" transconductance (includes route tiles) */
  SYMBOL *Gb;          /* "backward" transconductance (includes route tiles) */
  SYMBOL *unitC;       /* Unit capacitor tile */
  SYMBOL *nullC;       /* Non-capacitive capacitor route tile */
  SYMBOL *compressor;/* AC signal compressor route tile */
  SYMBOL *dcRib;       /* DC signal route tile */
  SYMBOL *dcCenter;    /* Additional DC signal route tile */
  SYMBOL *dcEdge;      /* DC signal connection stubs */
  SYMBOL *ldRib;       /* Special DC route for load tile */
  SYMBOL *combinP;     /* Positive input capacitor grid array route */
  SYMBOL *combinN;     /* Negative input capacitor grid array route */
  SYMBOL *comboutP;  /* Positive output capacitor grid array route */
  SYMBOL *comboutN;  /* Negative output capacitor grid array route */
  SYMBOL *zeroC;       /* Capacitor tile for placing "zeros" in filter */
  SYMBOL *zCenter;     /* "zero" capacitor route */
  SYMBOL *zEdge;       /* "zero" capacitor connection stubs */
  zeroinfo zero[10];   /* Array of "zero" capacitors */
  float GfCi;          /* Input C parasitic of forward transconductance tile */
  float GfCo;          /* Output C parasitic of forward transconductance tile */
  float GbCi;          /* Input C parasitic of backward transconductance tile */
  float GbCo;        /* Output C parasitic of backward transconductance tile */
  float unitcap;       /* poly1-poly2 capacitance of a single unit cap tile */
  float unitcapzero;   /* poly1-poly2 capacitance of a single zero cap tile*/
  float unitparasitic; /* parasitic capacitance of unit capacitor */
  float circuitcap;    /* parasitic contribution of capacitor grids */
  int SizeLimit;       /* Maximum size vertical size limit of filter layout */
  int SegmentNumber;   /* Current segment # if layout is larger than limit */
  int RoutingSpace;    /* Vertical head space needed to route corners */
} stackelement;
```

Figure 15. Stack element structure

Main Functions

There are three main functions in OTACC that provide all the work required to build a filter: do_fs(), do_add() and do_inv(). These are standalone functions that each have assumptions made about the state of the program stack. For example, do_add() assumes there are two elements on the stack to add together. The ordering of the calls to the three main functions is dependant on correct data stream input, and correct use of each function in the wrapper routines. Each of the three functions are discussed in detail below.

do_fs(G,C)

do_fs() is the function that predistorts and builds the requested circuit capacitor. The function is passed the desired circuit capacitor value, C, and it obtains all other relevant information such as parasitic capacitance contributed by the OTA inputs and outputs, the routing tiles, and the unit capacitor tile from the top of the stack. It then uses the formula in equation (2) to calculate the number of unit capacitors to be tiled for the circuit capacitor. Predistortion of the desired capacitance value includes a majority of the parasitic capacitance from routing with predefined tiles.

$$N = \frac{C_{requested} - C_{I/O} - C_{routing}}{C_{unit_{parasitic}} + C_{unit}} \qquad (2)$$

where,

N is the number of capacitor tiles required to make the circuit capacitor.

$C_{requested}$ is the value of the original circuit capacitor

$C_{I/O}$ is the sum of the transconductance cell's input parasitic capacitance and output parasitic capacitance

$C_{routing}$ is the value of the grid routing between the circuit capacitor and the transconductance cell

$C_{unit_{parasitic}}$ is the value of the capacitor tile's parasitic capacitance

C_{unit} is the extracted capacitance of the capacitor tile (i.e. poly1- poly2 capacitance)

A capacitor array is then created from the calculated number of required unit capacitors. The number of unit capacitors that can be used in one row of the array is determined by the width of the array connection combs divided by the width of a single unit capacitor. After the rows of unit capacitors are created, if the last row is incomplete, the remaining spaces are consumed by "null" capacitors which are just place holders that have signal connections to the tiles around them.

For example, suppose that the capacitor array is calculated to require a total of forty-two unit capacitors. Also suppose that sixteen unit capacitors may fit into one row. In this case, thirty two unit capacitors could be used in the first two rows. The remaining row would be made up of ten unit capacitors and six null unit capacitors. All three rows would then be added

together to form a single capacitor array and the resulting capacitor array would then be pushed on the stack by do_fs().

do_add()

Increasing the order of a filter is done by adding additional sections or stages. In OTACC, the *do_add()* function accomplishes this task. In circuit capacitor terms, this function's responsibility is to connect the next appropriately sized tile to the filter in progress. It could be viewed as the "addition" operation in the original description of the filter's impedance signal flow graph.

In software terms this function is much more general. It doesn't have any knowledge about filter stages or SFGs. Rather, it simply pops the first two elements off the stack, combines their tiles together, and pushes the result back on top of the stack.

Because of this generality, the stack needs to be prepared such that the top element is the new stage to be added, and the second element from the top is the current filter in progress. If the stack is so prepared, then *do_add()* will pop the next stage and the current filter state off the stack, add the next stage to the filter, and push the new filter state back on the stack to be ready for the next operation. This generality also means that any addition operation can be obtained by this call. For example, the cascading of biquads

could also use this function. The use of cascaded biquads is a feature that is to be added to OTACC in the future.

do_zero()

do_zero() is actually a wrapper around the *do_fs()* function. It currently does not have a predistortion formula, but hooks are in place as a future enhancement.

The transmission zero capacitor tiles are added to a filter's structure after a structure called the backbone is completed. The filter backbone is made up of series inductors and shunt capacitors. Since there can be any number of transmission zeros, a mechanism needs to be in place to retain multiple pointers to the location in the backbone where the transmission zero capacitors are to be added. This is different from the backbone capacitors which have a single point of reference so they can be added to the current filter during the first pass through the input. Because of this, an array of transmission zero capacitor pointers is maintained in each stack element.

To begin the operation, do_zero() makes a copy of the current stack element, then modifies the fields relating to capacitor arrays to correspond to tiles for a zero capacitor. This "dummy" element is pushed on the stack and do_fs() is called. When do_fs() returns, the dummy element is on the stack with the *current pointer attached to the tile layout for the zero capacitor.

This tile layout is saved, and the dummy element popped off the stack and discarded. The current filter state has now returned to the top of the stack. A pointer to the saved zero capacitor tile is added to the current filter stack element's zero array.

do_inv()

This function completes the inversion and feedback operations in OTACC. Inversion in transconductance-C filters is accomplished by adding an inverting transconductance with a feedback path from the inverted output to the input of the current filter state. So, inversion really is in terms of an inverted output signal.

From a software standpoint, two or more transconductances are actually used to complete inversion. The first transconductance is used in the do_add() operation. Because we know the specific configuration is a ladder, we know that an inversion operation always occurs with an addition operation. Therefore, the *do_inv()* routine takes a non-inverting transconductance and an inverting transconductance and tiles them in a feedback configuration shown in Figure 17. The result is returned to the stack in anticipation of adding another admittance (and inversion) to the filter. Say we had a function, F, that we wanted to add an admittance to and

then invert. The first step of adding the admittance results in a voltage transfer function shown in equation (3) and Figures 16 and 17.

$$\frac{V_o}{V_i} = \frac{g_m}{F + C_{new}} \tag{3}$$

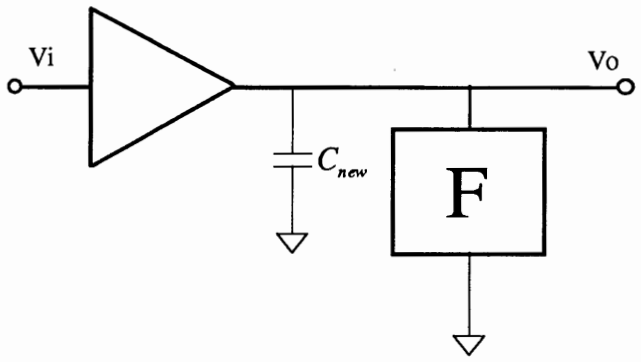


Figure 16. Adding Vo/Vi voltage transfer function to the current admittance function, F

Then, the voltage transfer function has to be converted back to an admittance by applying inversion so that addition can be repeated for the next stage of the filter.

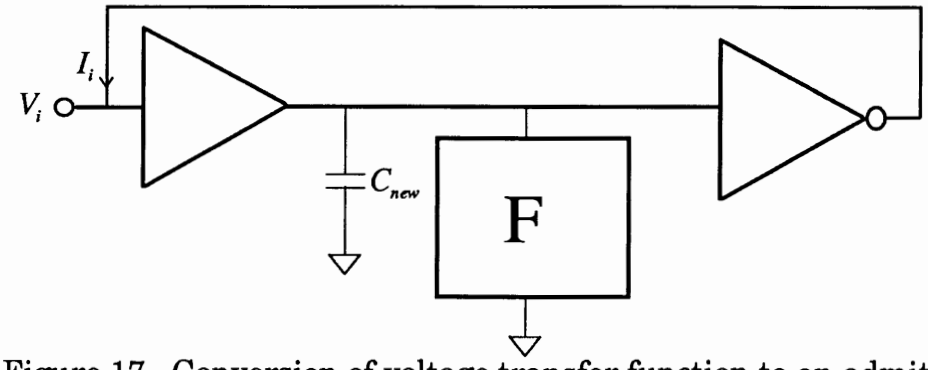


Figure 17. Conversion of voltage transfer function to an admittance

The *do_inv()* function combines these operations by tiling two transconductances in feedback. By "folding" the configuration with the current function, F, as the fold point, the length of the feedback line becomes significantly shorter and introduces less parasitic capacitance. This configuration is shown in Figure 18.

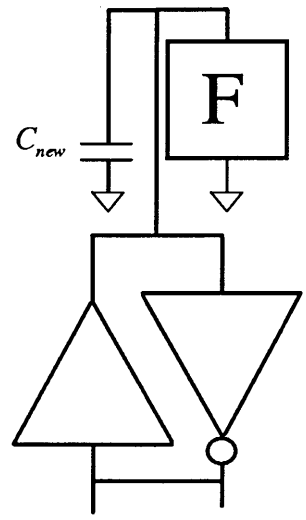


Figure 18. Function with added admittance - folded configuration

CHIP PLANNING

OTACC layout grows primarily in one direction. The width of the backbone is set by the sum of the tile widths of the forward transconductance cell, the DC channel route, and the backward transconductance cell. The maximum width of the final filter includes the widths of any transmission zero capacitors attached to the sides of the backbone. Although

these capacitors are variable in size, they typically do not exceed 150 microns each, or a total of 300 microns for a 2 micron technology. Figure 19 shows a typical section of a filter including a pair of transmission zero capacitors.

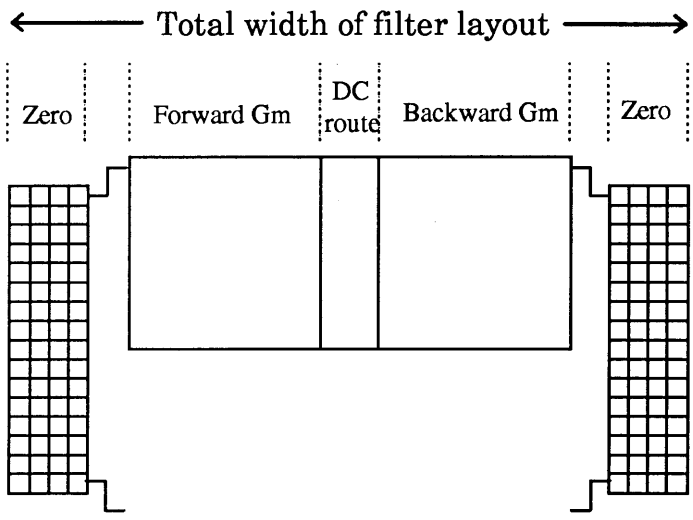


Figure 19. Filter basic building block used to calculate width of the filter.

The basic building block of the filter layout consists of the forward transconductance, backward transconductance and DC route tiles in the configuration shown in Figure 19. The layout grows vertically because these meta-elements are stacked on top of each other with capacitor arrays sandwiched in between as the order of a filter's transfer function increases. Although the main growth is in the vertical direction, some horizontal growth occurs if the transmission zero capacitor array is present. Finite transmission zeros are not always required in a filter.

Unfortunately, a tall, thin structure can be a large waste of silicon. It is desirable to have the layout be roughly square so the width and height should grow at about the same rate. In order to better utilize space, an additional function was added to OTACC that would "zigzag" the layout. Thus, the resulting layout would be transformed as in Figure 20.

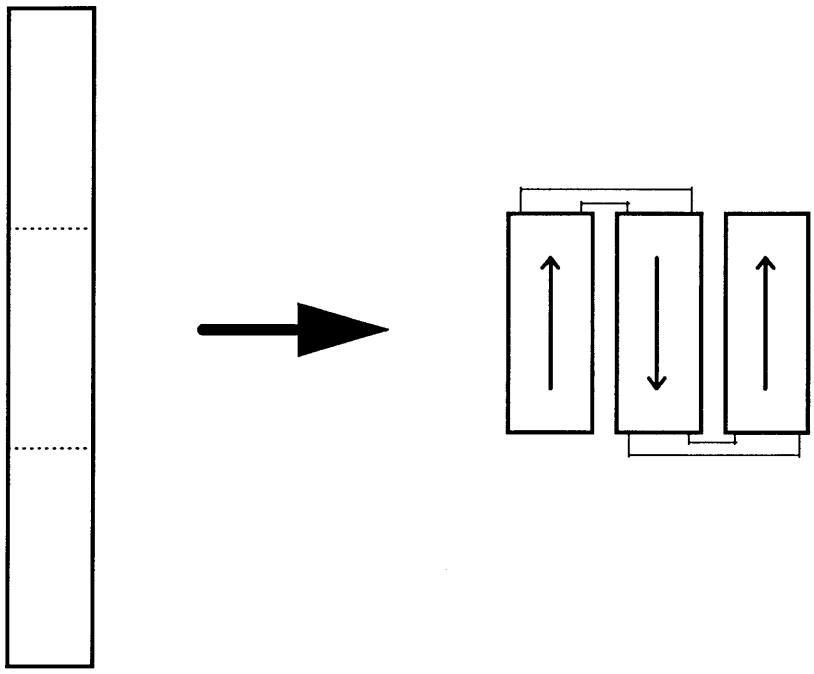


Figure 20. Converting a linear filter to a zigzagged filter

Note that the layout shown may or may not include transmission zeros even though they are not explicitly shown. The arrows on the right of the figure indicate the direction of growth and the flow of the AC signal path.

The rules the software currently uses for creating the zigzag structure are as follows:

1) Wherever the filter's structure is broken in preparation for reversing the direction of growth, the junctions are "capped" with a compressor tile as shown in Figure 21. This is required because the places where a break can occur along capacitor boundaries contain many fingers of identical parallel signals. If each of these AC signals were routed this would be a large source of additional parasitic capacitance. A compressor tile combines the parallel signals into a single wire which is much more suited for the additional route required to turn the filter's signal path in the opposite direction. The same compressor tile is rotated 180 degrees to provide the signal decompression for continuing the filter construction. This is shown in Figure 21.

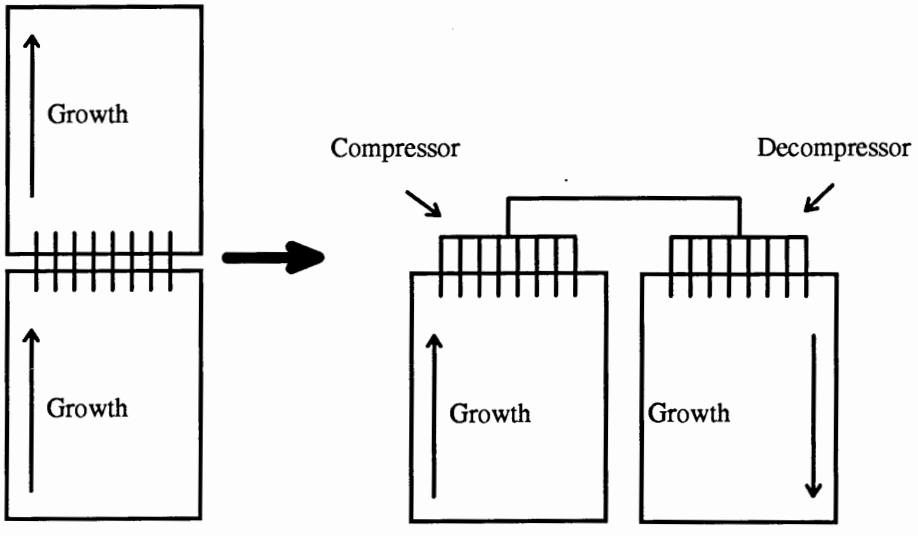


Figure 21. Use of compressor tiles when breaking a filter

2) The filter may only be broken at the junction of a shunt capacitor. This is probably easiest seen by looking at the orientation of the schematic diagram in Figure 22 since this corresponds with the growth of the filter layout:

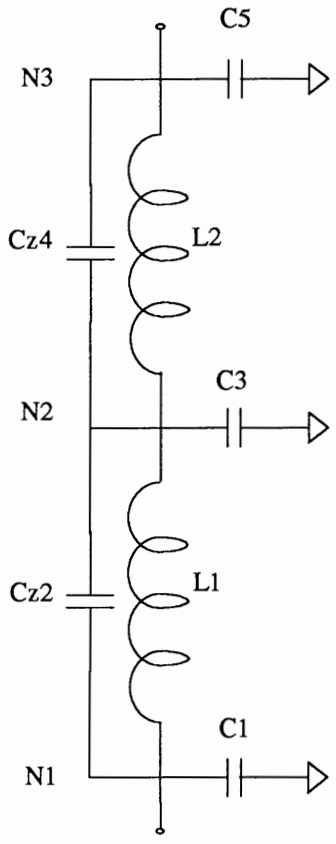


Figure 22. Schematic of the LC prototype in same orientation as the IC layout

In the section on element substitution, it will be noted that an inductor is simulated with four OTA's and a capacitor. Technically speaking, a "break" could have been allowed on an OTA border within

the inductor. However, any zeros that are present are in parallel to the inductor. A break internal to an inductor would create a problem of routing the second connection of the zero capacitor in a different section of the filter. To maintain simplicity, it was decided to disallow these structure breaks.

3) The length of each segment of the filter is allowed to grow to the segment size limit, minus the required routing space, minus the height of the compressor tile. The segment size and routing space requirements are both defined in a technology file, e.g. otacc.tech.

Each building block is tiled and then evaluated to see if its addition to the filter would violate the segment length limit stated above. If it does, then a compressor is added to the current filter forming a completed filter segment which is pushed onto the stack. Then a new segment is started beginning with a compressor rotated 180 degrees (making a decompressor) and adding the new element that would have been part of the previous segment had it fit.

It is important to note that each segment is built in the same direction. When the build process is complete, all the segments are popped off the stack and reassembled in the proper direction.

FINAL ASSEMBLY

After the software has completed processing the input and building segments of the filter, final assembly begins. To prepare for assembly, a segment is popped off the stack, and then its top border is extended to the size limit minus the predefined route space. Since CFL works with the borders of each tile, this operation is essential because CFL is being used to connect all the segments together into one unified cell. Once the filter segments are a uniform size, each segment's borders are visible to the CFL router function.

Next, each segment contains an array of pointers of any transmission zeros that must be included in the layout. If the length of the array is greater than zero, then each transmission zero capacitor in the array is connected to the corresponding segment of the filter as described earlier.

This process of popping the first segment off the stack, extending its top border, and routing its zeros also includes the output load of the filter since the load is always the last element to be built and is always placed right side up. The next segment to come off the stack also has its top border extended and its transmission zeros routed. It is then turned 180 degrees and placed to the left of the load segment. The process is repeated until the stack is empty, with every other segment being rotated 180 degrees. The

result at this point looks like Figure 23 with the dotted lines representing any possible transmission zero capacitor arrays:

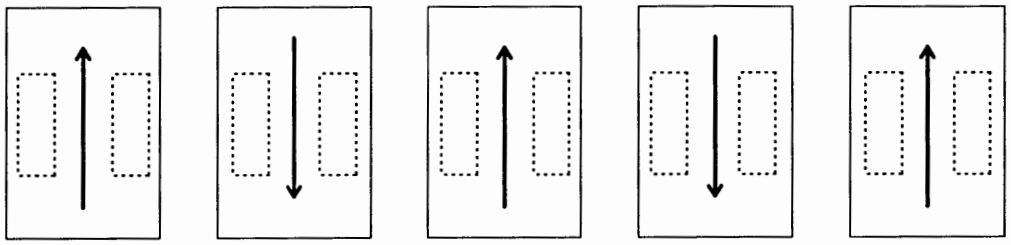


Figure 23. Side by side placement of filter segments

Next, the routes are created by OTACC that connect neighboring segments together. This is shown in Figure 24.

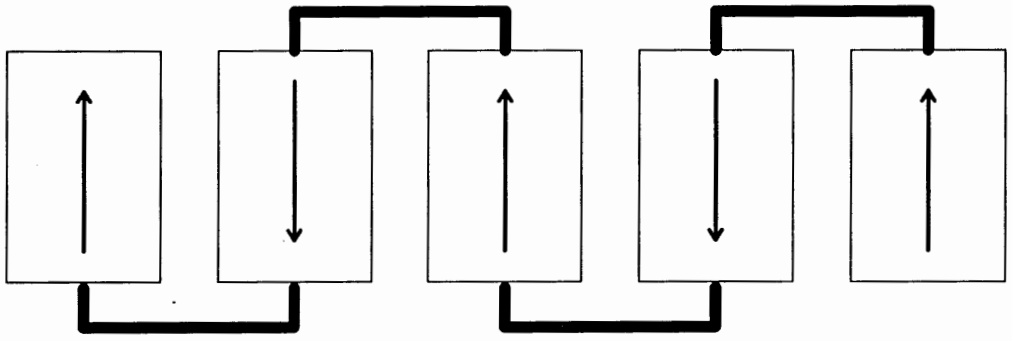


Figure 24. Filter segments routed together

The reader may wonder what is the effect of the parasitic capacitance from these dynamic routes on the filter's response. Chapter III contains simulation results showing that the difference between straight layout and zigzagged layout is negligible for all test cases attempted. Ideally, the parasitic capacitors from the zigzag routes would be included in the

predistortion like the other parasitic capacitors. However, this additional predistortion is not included in OTACC at this time and is left as a future feature to be implemented.

To predistort circuit capacitors with the parasitic capacitances from the zigzag routes, the capacitor arrays would need to be built with a flexible row of unit capacitors. The amount of parasitic capacitance contributed by the dynamic route can be determined by:

$$\text{Parasitic Capacitance} = Length_{line} \bullet Width_{line} \bullet \text{Cwire},$$

where Cwire is determined from the process parameters and would be computed as the number of unit capacitors that this represents. Therefore, the flexible row of unit capacitors would be added, minus the number of unit capacitors equal to the capacitive contribution of the dynamic route.

FILTORX INTERFACE

Although the original SFG-based grammar interface worked well, it was not robust enough to directly represent the transmission zeros which were included as an addendum to the filters SFG description. Figure 25 shows the form of the original program input.

$$[0.0, C_1[0.0, L_2[0.0, C_3]]]$$

$$1\ 2\ C_{Zero}$$

Figure 25. Original OTACC grammar with a transmission zero

In this case, the numbers 1 and 2 indicate node numbers in the layout. This description works, but counting which nodes a transmission zero should be connected is not a desirable solution.

The filter design cycle consists of taking a filter specification and generating the transfer function, possibly using a classical polynomial. Next, the transfer function is realized into circuit elements and finally implemented. The University of Toronto has created a design tool called FiltorX to assist in the approximation and realization process. OTACC's new grammar was designed to be compatible with the output of FiltorX so that the two tools could be combined into one allowing the approximation, realization, and implementation phases of the filter design cycle to be assisted or automated by the software.

The output of FiltorX is in terms of the elements that make up a filter. Reading the schematic in Figure 27 from left to right yeilds the FiltorX description shown in Figure 26.

```
ladder filter (
  seriesR 1.0;
  shuntC 9.0;
  seriesLC 5, 9;
  shuntC 5.0;
  shuntG 1.0;
  )
```

Figure 26. New OTACC grammar from FiltorX

Note that the "seriesLC" element is a "tank" element that is made up of an L and a C in parallel, which is the basic element of a transmission zero.

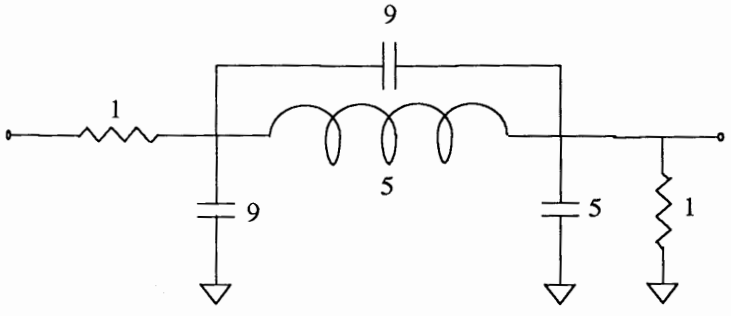


Figure 27. Schematic diagram matching filter description in Figure 26.

By changing the input grammar of OTACC to match the output of FiltorX, we create the hooks necessary to automate the most computationally intensive parts of the filter design cycle. A top level program can be written which first runs FiltorX and then pipes the output into OTACC. Changing to

this new grammar also provides a simple solution to the zero problem previously described.

ELEMENT SUBSTITUTION

A consequence of converting to this new grammar is that it changes the structure of the OTACC tool from terms of integration and inversion to terms of circuit elements. As mentioned previously, the original tool followed the continuous fraction form, which is recursive in nature. It recursed to the end of the input file and started building from the load back towards the input. In terms of layout, the filter was built in a downwards direction.

The new grammar, however, is straight forward. The filter starts at the first element (the input resistor), and the layout proceeds to the last element (the output, which is typically a load). Since the input is in terms of elements, the internal data structure of OTACC is in terms of elements also. There is a one to one mapping of FiltorX key words to functions.

The key words and the corresponding internal functions that are presently supported for filter layout are:

seriesR ⟶ seriesR()
seriesLC ⟶ seriesLC()
seriesC ⟶ seriesC()
shuntC ⟶ shuntC()
shuntG ⟶ shuntG()

This implies that some form of element substitution is required rather than the SFG methodology used previously. Capacitors can be created directly in silicon. The inductors, however, must be simulated. Figure 28 shows the correspondence for a floating inductor.

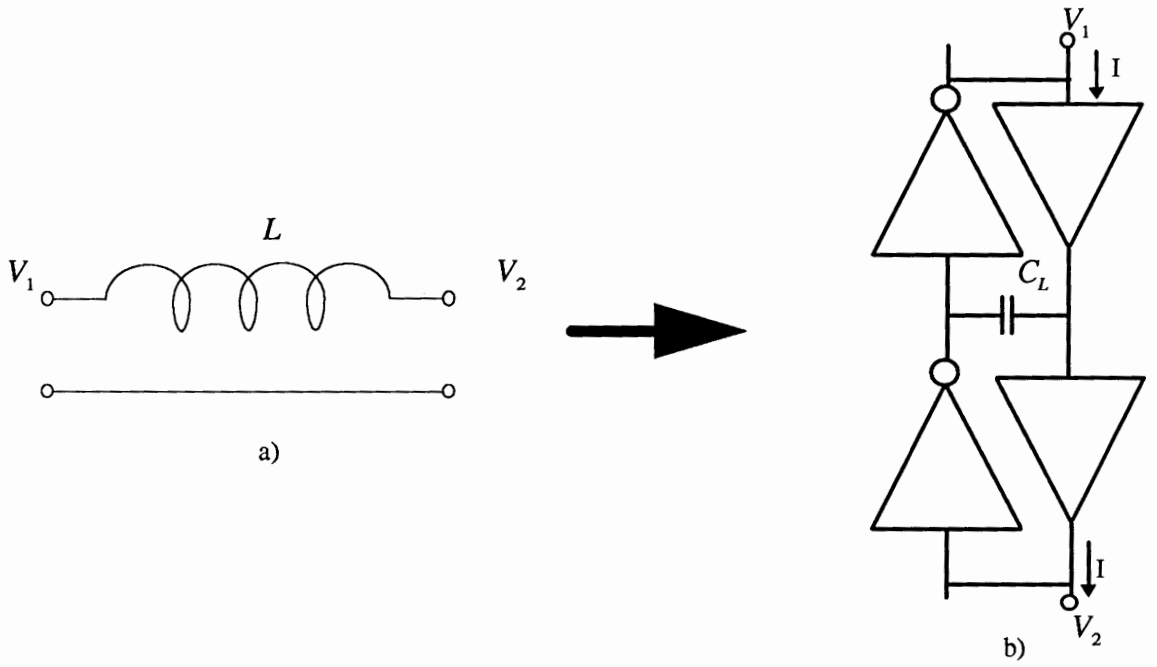


Figure 28. Transconductance-C model of a floating inductor (transconductances represented as inverters)

The correspondance between a) and b) in Figure 28 can be shown in mathematical terms as:

$$Z_L = \frac{V_1 - V_2}{I} = \frac{sC}{g_m^{\,2}} = sL \qquad (4)$$

which is in effect, two cascaded gyrators loaded by a capacitor. A gyrator is an element whose input impedance is inversely proportional to its load impedance.

So,

$$Z_{in}(s) = \frac{r^2}{Z_{load}(s)}, \tag{5}$$

where r is defined as the gyration resistance. Therefore, if we select the load to be a capacitor, then the input impedance models that of an inductor,

$$L = r^2 C. \tag{6}$$

From a software perspective, we wanted functions such as seriesLC(), or shuntC() that knew how to build an inductor or capacitor from the existing code: do_inv(), do_add(), and do_fs().

The capacitors were easy since there is an explicit one to one correspondence between shuntC() and do_fs(). ShuntC() needs to build a capacitor and combine it with the filter segment. Do_fs() does the work of building the capacitor, and do_add() combines the capacitor with the current filter structure, pushing the result back on the stack.

Creating the active simulation of an inductor is a little more involved. Looking at the layout, the structure in Figure 29 simulates an inductor.

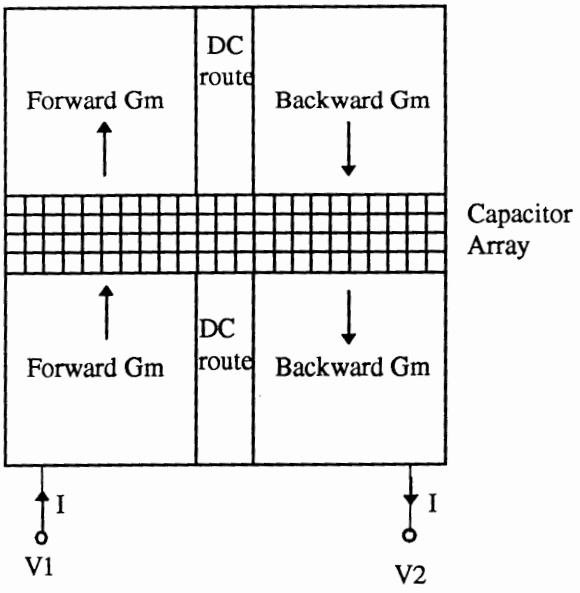


Figure 29. Layout of the active simulation of an inductor

However, the do_inv() routine builds the block that is made up of the two transconductance cells and the DC route.

Since the layout now needs to be built in an upwards direction rather than the downward direction required by the old grammar, the stack order needs to be reversed. Therefore, a routine similar to do_inv() was added so that the filter and the inversion block would be placed on the stack in the opposite order. This way, when the add() routine was called, the inversion block would be placed on top of the filter rather than below it. This 'new' routine was called do_invr() which stands for do_inv, reversed.

Next, an inductor can be created with two calls to do_invr(), one call to do_fs(), and three calls to do_add() to assemble the simulated inductor. Of

course, close attention must be paid to the stack order to make sure they are placed properly by the do_add() routine.

As can be seen, the new input grammar for OTACC only requires additional parsing capabilities and some small wrapper routines around the previous code. The zigzag final assembly discussed previously is included in the new wrapper routines.

SINGLE-ENDED VS. DIFFERENTIAL

The reader may notice that up until this point, most examples have been with single-ended transconductance elements. This does not imply that single-ended transconductances are the transconductance of choice. They have been used to simplify the examples presented. In practice, differential transconductances are desired. As will be seen later, some of the parasitic capacitances induced in single-ended filter designs cannot be predistorted with a circuit capacitor. Transconductance-C filters are based on using the gyrator, which combines an inverting and noninverting OTA to simulate an inductor. A noninverting OTA is created by multiplying the output of an inverting OTA by negative one. With a differential design, the creation of a "noninverting" OTA is accomplished merely by feeding a negative output into a negative input of the next OTA. In effect, this is a simple twist of the routing as shown in Figure 30.

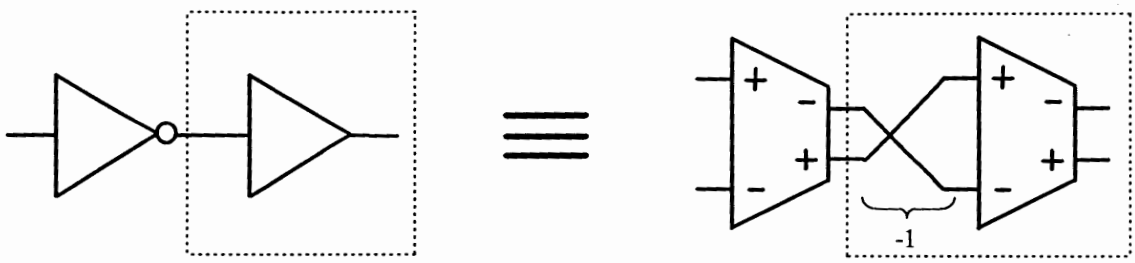


Figure 30. Building a noninverting transconductance by multiplying by -1 using differential transconductances

However, to get the same configuration in terms of single-ended designs, two extra elements are added which introduces additional circuit nodes as shown in figure 31:

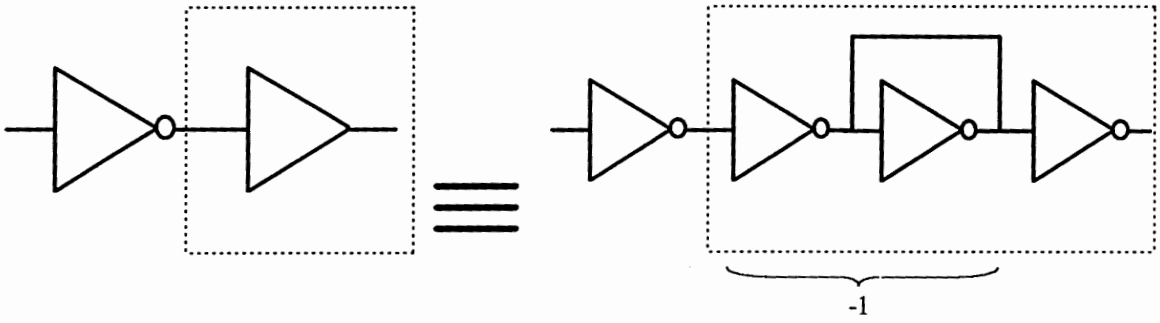
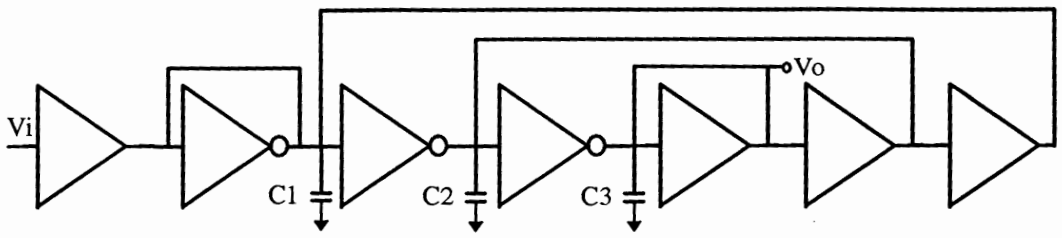
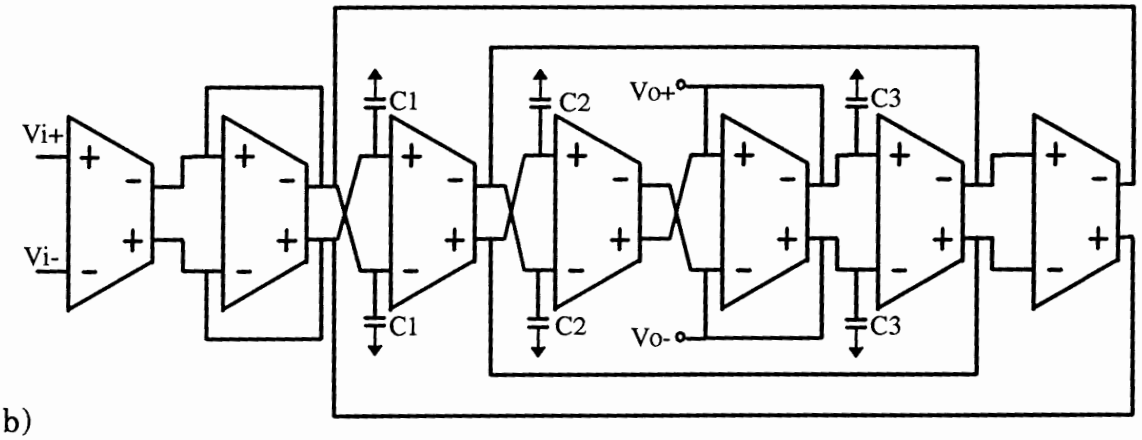


Figure 31. Building a noninverting transconductance by multiplying by -1 using single-ended transconductances

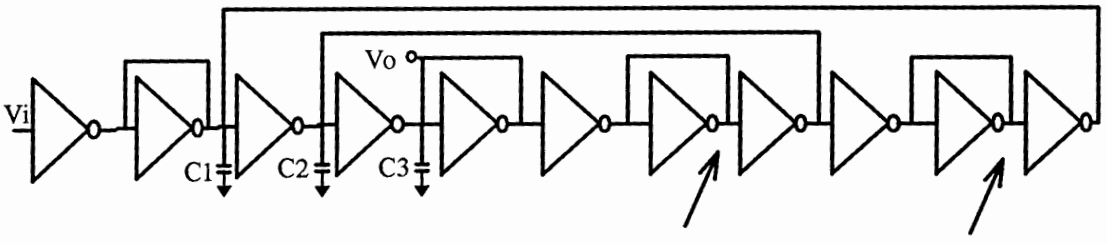
Let's compare the two different types of OTAs in an actual circuit. Take the third-order filter discussed previously. The comparisons are shown in Figure 32.



a)



b)



c)

Figure 32. (a) Model of third-order filter (b) Differential OTA based third-order filter (c) Single-ended with only inverting transconductances. The arrows indicate nodes introduced by the method that cannot be predistorted.

First glance shows that the single-ended design (c), has many more transconductance elements. While this is true, the internal complexity of a differential transconductance more often than not results in extra silicon area being required. The most important item to note is pointed out by the arrows in part (c). When making a non-inverting transconductance with single-ended designs, a node is created where the parasitic capacitance cannot be assimilated into a desired circuit capacitor. Introducing these errors brings us back to the very concept that has slowed down the design and proliferation of analog automated synthesis software: not accounting for parasitic capacitances.

A second reason differential design may be preferred over single-ended design is because it makes good sense from a circuit's standpoint. As more and more is integrated on a single chip, the likelihood of noise sources increases. A particular danger is associated with mixed-signal configurations where both analog and digital circuits share the same substrate. Therefore, the designer needs to take steps to minimize the cross talk effect, such as the use of fully balanced differential circuits [18].

LIBRARY MAINTENANCE

OTACC relies heavily on proper tile design in the library. Its database could be compared to jigsaw puzzle pieces and the OTACC tiler fits them together without detailed information about what picture the puzzle piece

contains. Therefore, the designer must take great care in creating the proper support tiles. Once the library of transconductance and support tiles exists, it is simply a matter of choosing which set of tiles to use.

Obviously, the price for the accuracy and level of automation obtained is the heavy reliance on the designer in the making of the library. It is felt that the trade off is worth the effort. In the discussion of the following tiles, please refer to the example layout in Appendix C.

Design of a new transconductance element

When designing a new transconductance, the big picture must be considered. OTACC assumes that the DC routes go through the middle of the filter structure. This means that all DC bias signals should exit on the right side of the transconductance element. In addition, the AC inputs are assumed to be at the bottom and the AC outputs at the top of the cell. If designing a single-ended transconductance, both the inverting and non-inverting tile must be created. The smallest of the inverting and noninverting tiles must have its top and bottom connections extended to match the size of the other, or the capacitor combs must include the extra route to have the same effect which is not recommended.

Design of the DC routes

The first DC route tile to create is the load DC route. This tile contains routes to connect the AC inputs to the AC outputs (in the case of a differential OTA, be sure that '+' inputs feed into '-' outputs and vice versa).

The easiest way to create the DC route tiles is to place a transconductance cell next to another copy of the same cell rotated 180 degrees, leaving enough routing space between the two cells. Now, run as many DC routes in between these cells as required, making connections as required to the transconductance cells. In addition, run the AC feedback wires in the same route tile. This load DC route tile sets the width of the DC route channel. The other DC route channel is a copy of the load DC route, with the AC feedback wires deleted. Note that it must retain the same width as the load DC route.

The "center" tile is a small horizontal slice of the basic DC route tile. Its height is determined by the height of the unit capacitor tile.

Design of the AC comb tiles

There are three purposes for the comb tiles. The first is to interface the OTA to the array of unit capacitors. The second is to provide feedback when the forward and backward OTA cells are placed next to each other. The third is to provide the -1 crossover when using differential

transconductances. It is best to look at the layout generated by OTACC from one of the sample input files provided with the OTACC package to see how they are done.

Four combs are needed: output-positive, output-negative, input-positive, and input-negative. The polarity of the comb really only applies to differential OTA designs. A negative comb has its feedback wires crossed and fed towards the center of the filter. A positive comb has its feedback wires fed towards the center of the filter without crossing. Both types of combs have one or two signals that connect to the OTA and several fingers connecting to the capacitors. The finger connections alternate as plus and minus connections to the capacitor array.

Design of the compressor tile

The compressor is made up of two positive combs with a DC route slice in between. Ideally all connections on the compressed side of the tile are of the same metal. This prevents overlap of the zigzag routes.

III. INVESTIGATION AND FINDINGS

OTACC AUTOMATED SIMULATION

The amount of automation available in OTACC continues to increase through the life of the project. Initially, the software was capable of building the filter backbone with the capacitors that make up the tank circuit being added by hand later. Although the bulk of the work was removed from the user's responsibility, the desired end product was to be able to simply "touch a button" and have the software do the rest.

In essence, OTACC is the realization step in the filter design cycle. The previous step is approximation where the filter characteristics are converted into a set of element specifications that make up the filter. The final step is verification where the filter layout is simulated to verify that it meets the original specification. Ideally OTACC needs to interface to the specified input and direct its output to a verification tool such as the circuit simulator, Spice.

Although the input to OTACC can be obtained from any source, FiltorX was chosen as the tool to use for seamless integration. In order to accomplish this task, the input grammar was changed to accommodate FiltorX's output format Once attained, the task for OTACC is to create a Spice simulation deck along with the final filter layout.

Semi-Auto Simulating

The manual process of routing the transmission zeros and creating a Spice simulation file from layout required the following steps:

1) Load the layout into Berkeley's Magic
2) From within Magic, route the zero capacitors (which were placed by OTACC but not wired into the backbone) and extract the layout.
3) From Unix, run ext2spice (which converts an extracted file to a spice file).
4) Edit the Spice file to use poly1/poly2 capacitors.
   (The extractor interprets the poly1/poly2 layers as a transistor. These "transistors" need to be located and converted into a capacitor element.)
5) Load the Spice file into Spice and simulate.

In order to semi-automate this process, an early version of OTACC created a Magic script file with instructions to use Magic internals to route the zero capacitors and extract the final layout. Then, OTACC forked Magic with the generated layout as its input. At this point, the user took the extracted file and converted it to Spice format with ext2spice. An awk script was used to search for and convert some of the spice records for floating poly1-poly2 capacitors. Finally, the post-processed file was loaded into Spice and simulated.

Full-Auto Simulating

In order to fully automate the simulation process, a couple of steps were taken. First, the routing of the transmission zero capacitors was removed from Magic's domain and added to OTACC by utilizing CFL's routing facilities. This sped up the process and reduced the reliance on Magic. Second, and most importantly, a driver script was utilized to control OTACC, Magic, Spice, and Awk. Being a top level script, it was able to control data flow between the four programs. In addition, OTACC can be interfaced to some design centering software that is the subject of another research project at Portland State University [20].

SIMULATIONS PERFORMED

Many simulations were performed to verify the validity of the layout produced by OTACC. In all cases, the simulations performed as expected. The comparisons between the specification, simulation and measured results are discussed in the following chapter for two different filters: a third-order elliptic filter and a fifth-order inverse Chebyshev filter.

DYNAMIC ROUTE PARASITIC ANALYSIS

Simulations were also performed to discover the effect of the parasitic capacitances of the dynamic routing in the zigzag routine. Initially, an analysis was done to determine the extent of the parasitic capacitance added by the additional route. This was accomplished by generating the sample input (an example distributed with OTACC) fifth-order inverse Chebyshev filter under worst case conditions. Then, the route was isolated and extracted into a Spice file. The worst parasitic capacitance found was 150fF, which translates to 2.5 unit capacitors. This demonstrates that the amount of route parasitic capacitance is on the order of a unit capacitor rather than rows of unit capacitors, and confirms that the proposed method of including the dynamic route parasitic capacitance in the predistortion is realistic.

Taking the analysis a step further, both the sample third-order and fifth-order filters were generated in straight and zigzagged form. Theoretically, the zigzagged layout would show worse charactoristics because it had the additional parasitic capacitance in the dynamic route that was not included in a predistorted circuit capacitor. The results obtained are shown in Table 1 and Table 2.

Table 1. Third-order elliptic filter: route parasitic capacitance analysis

Parameter	Straight layout	Zigzagged layout
Cut-off Frequency	13 MHz	12.75 MHz
Passband Ripple	0.7 dB	0.55 dB
Stopband Frequency	19 MHz	19 MHz
Stopband Attenuation	-30.65 dB	-32.0 dB
Transmission Zero	-60dB @ 21.3MHz	-61.5dB @ 21.3MHz

Table 2. Fifth-order inverse Chebyshev filter: route parasitic capacitance analysis

Parameter	Straight layout	Zigzagged layout
Cut-off Frequency	4.62 MHz	4.6 MHz
Passband Ripple	0.8 dB	0.8 dB
Stopband Frequency	6.1 MHz	6.1 MHz
Stopband Attenuation	-41.9 dB	-42.6 dB
Transmission Zero #1	-56.5dB @ 6.6 MHz	-57.5dB @ 6.6MHz
Transmission Zero #2	-64.75 @ 7.35MHz	-65.5dB @ 7.4MHz

Although the values are not identical, they are fairly close. It should be pointed out, however, that filters containing very small circuit capacitors would be affected on a larger scale, as will be discussed in the limitations

section. In most cases the effect will be negligible, but effort should be put into predistorting the zigzag routes to make all cases as accurate as possible.

SILICON MEASUREMENTS

To test OTACC, two test chips were generated and sent to MOSIS (an organization of the National Science Foundation that specializes in fabricating small lots of silicon for both industry and academic sectors):

1) Third-order elliptic filter, fully balanced OTAs

2) Fifth-order inverse Chebyshev filter, single-ended transconductances

The following sections compare the ideal specifications with the simulated layout and measured results. See Appendix B for plots of the test chips.

Third-order Elliptic filter, fully balanced OTAs

The third-order elliptic filter was designed to have a cutoff frequency of 17MHz with a stopband attenuation of 29dB at 23.5MHz, a transmission zero notch at 26.3MHz, and a passband ripple less than or equal to 0.5dB. This chip also included test components from another research project: output buffers, differential-to-single, and single-to-differential converters. The pin outputs were wired such that the converters could be bypassed if needed. This turned out to be a good idea since the converters caused poor operation.

As the data in Table 3, and the pictures in Figures 33 and 34 demonstrate, the results of the simulations and silicon measurements were very close to what was expected. See Appendix A for simulation results.

Table 3. Third-order elliptic filter: Comparison between specification, simulation, and measured results.

Parameter	Specification	Simulation	Measurement
Cut-off Frequency	17 MHz	17 MHz	18.8 MHz
Passband Ripple	<= 0.5 dB	0.75 dB	0.5 dB
Stopband Frequency	23.5 MHz	24.2 MHz	25 MHz
Stopband Attenuation	-29 dB	-31.7 dB	-30 dB
Transmission Zero	-∞dB@26.3MHz	-43dB@26.9MHz	-55dB@28MHz

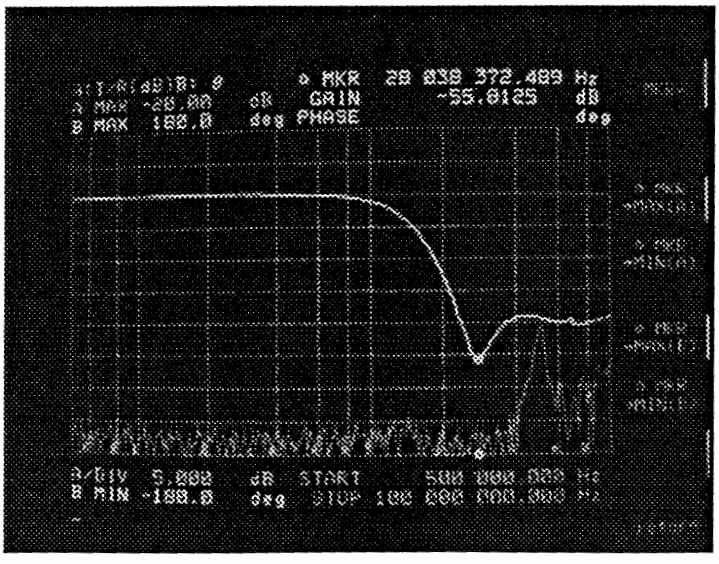


Figure 33. Third-order elliptic filter measurement: magnitude response

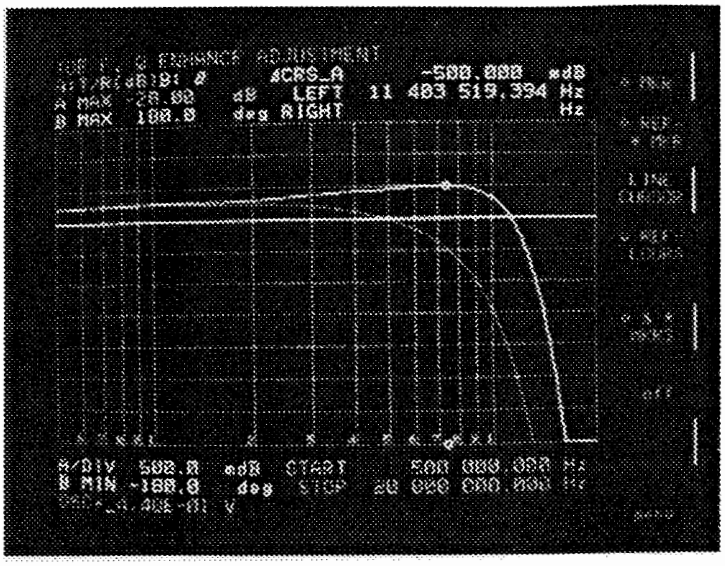


Figure 34. Third-order elliptic filter measurement: Q adjustment results in a change in passband ripple

Fifth-order Inverse Chebyshev filter, single-ended transconductances

The fifth-order chip was an inverse Chebyshev filter designed to have a cutoff frequency of 18MHz with a stopband attenuation of -40dB at 36MHz. This chip included four identical filters which allowed them to be used to build a filter with differential characteristics. The process used was the 2 micron MOSIS SCNA, n-well, double poly process similar to the third-order elliptic filter.

As the data in Table 4 demonstrates[21], the results of the simulations and silicon measurements were very close to what was expected. However, with regard to the stopband, a trade-off was encountered. The attenuation was 6dB too high at the desired frequency. By adjusting the OTA, the

stopband attenuation was decreased to the proper -40dB, but at a shifted stopband frequency to 41 MHz. However, signal feed through prevented good measurements of the stopband charactoristics. See Appendix A for simulated magnitude response waveforms.

Table 4. Fifth-order inverse Chebyshev filter: Comparison between specification, simulation, and measured results

Parameter	Specification	Simulation	Measurement
Cut-off Frequency	18 MHz	18.5MHz	18 MHz
Passband Ripple	0 dB	1 dB	<= 1 dB
Stopband Frequency	36 MHz	35 MHz	36 MHz
Stopband Attenuation	-40 dB	-53 dB	-34 dB

LIMITATIONS

The parasitic capacitances of the OTA ultimately set the size of the smallest usable capacitor in the filter since the desired capacitor starts with the parasitic capacitance and adds to this until the proper size is obtained. Correspondingly, this also sets the maximum obtainable filter frequency. The following scaling equation shows the relationship between the filter's frequency and the size of the required filter capacitors:

$$\frac{C_{normalized} \cdot g_m}{2\pi f_0} = C_{circuit} > C_{parasitic} \quad (6)$$

As can be seen, as f_0 is increased, $C_{circuit}$ gets smaller. One way to prevent the desired capacitor from becoming too small is to increase the value of the transconductance. However, as the transconductance is increased, the tendency is to also increase the parasitic capacitance. Therefore, we once again run into the problem where the parasitic capacitance becomes larger than the desired capacitor on the circuit node. This has the effect of limiting the realizable frequency in silicon.

Another limitation is the poly1/poly2 capacitors being used. The capacitors are built as shown in Figure 35:

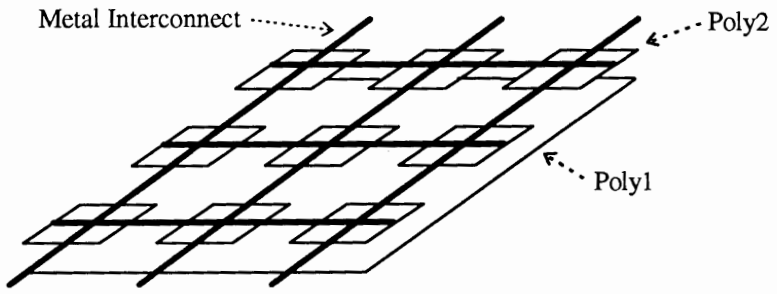


Figure 35. Silicon implementation of a capacitor array

The problem comes from the sheet of poly1 at the bottom. This is one of the capacitor's terminals, making a resistive grid between parallel capacitors. The ideal capacitor grid looks like Figure 36.

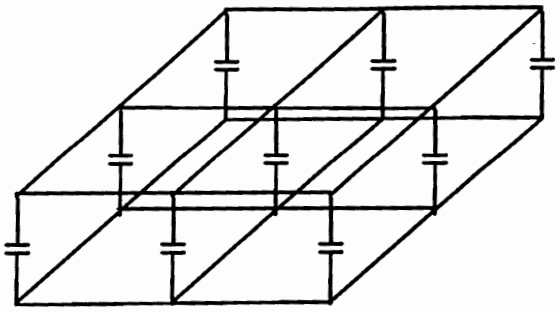


Figure 36. Ideal capacitor array schematic

However, in after fabrication the grid looks more like Figure 37:

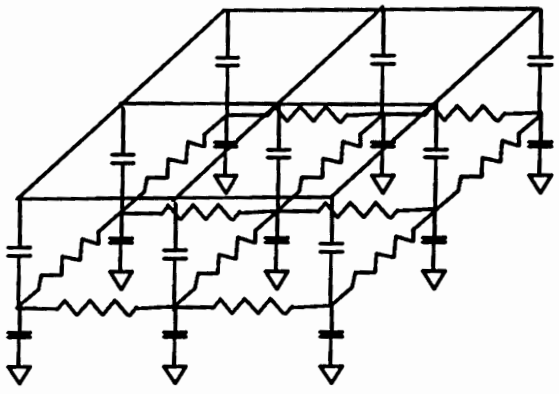


Figure 37. Actual capacitor array schematic

This resistive limitation can be reduced if the widths of the poly1 wires are selected to be wider than they are long. Also, the additional grounded capacitor should be included in the capacitance predistortion.

The measured results of the third-order elliptic filter showed that the ripple in the passband was relatively flat. One possible explanation for this is the unaccounted parasitic resistance in the capacitor array described above. Another possible source is also the OTA. Because of a deficiency in

the tools, distributed line resistance is not passed to the Spice simulation deck, so this effect becomes masked. A simplistic simulation containing two 0.1 ohm parasitic resistors in the shunt capacitors demonstrates the diminished ripple effect, and is shown in Figure 38.

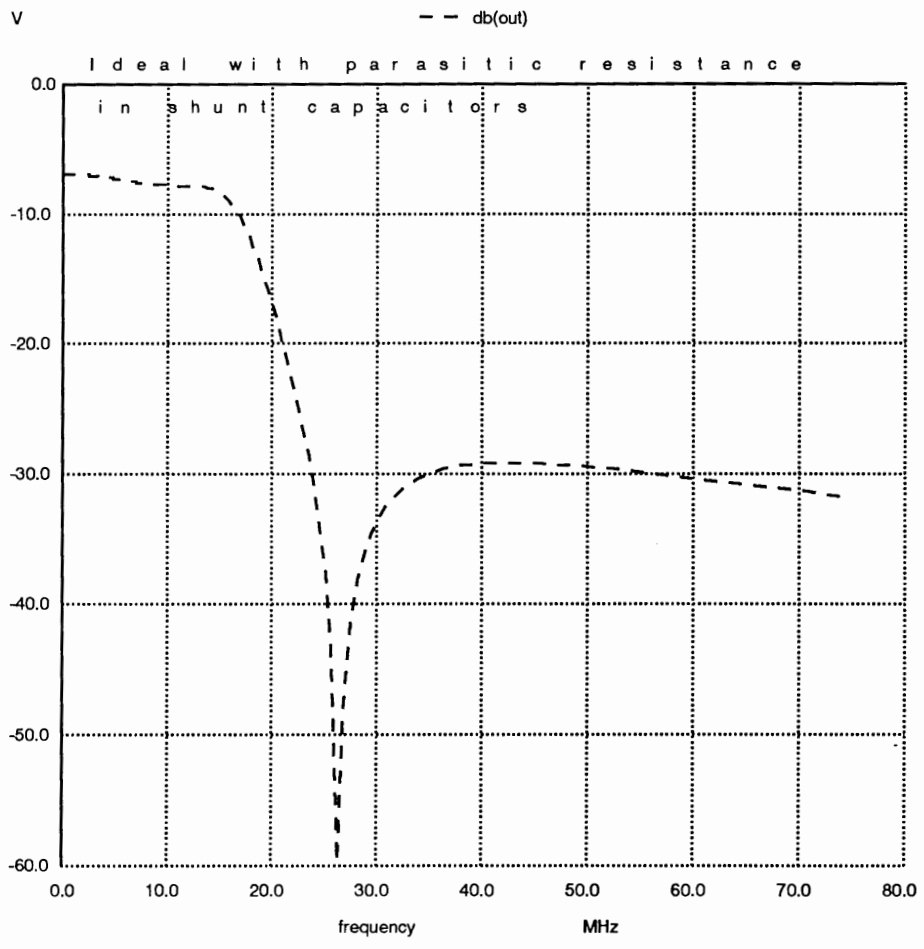


Figure 38. Parasitic resistances in capacitor array: 3rd order elliptic filter

IV. CONCLUSIONS

Automatic synthesis of VLSI layout has traditionally been a tool only available to digital designers. This is because of the sensitivity differences in parasitic influence between digital and analog circuits. Because analog signals are typically measured in terms of millivolts, slight variances in analog signals caused by the unavoidable parasitic capacitances can cause trouble. Digital circuits are able to address parasitic capacitance problems because they use larger noise margins, and also because they can be overdriven to compensate for any signal loss. This same approach in analog circuits would be disastrous.

To overcome parasitic capacitances in analog circuits, a different approach is required. Instead of overdriving or ignoring the parasitic capacitors, they must all be carefully accounted for and used as part of a desired circuit capacitor. The process of absorbing a parasitic capacitor into a circuit capacitor is called predistortion. Since predistortion can only occur on nodes where a circuit capacitor exists, then the scope of this solution is limited to specific configurations.

This research focused on LC ladder continuous-time filters using transconductance-C methods. The advantage of the LC ladder configuration

is that every node in the original circuit contains a capacitor where predistortion may occur.

In order to assure accuracy in determining the available parasitic capacitances, a method called "tiling" was used. This involves creating a library of standard transconductances and corresponding route tiles that match each transconductance. Each tile, including the route tiles are characterized to determine the amount of parasitic capacitance available. The disadvantage of this is the overhead involved in building the tile library. This can be a tedious process, but the trade off is the level of accuracy that can be obtained.

The software resulting from this project is called OTACC. It is capable of receiving its input from FiltorX and sending its output to Spice and layout tools such as Magic. Thus, the design process from approximation to verification can be accomplished with the aid of a single design automation program.

Although the software works well, this is not an end-all in the search for analog CAD software. We chose to sacrifice generality to gain accuracy. However, as in any trade off, the ultimate solution is a win-win solution. A CAD tool that can create and charactorize its own library on the fly would be ideal, though this requires more compute power than that which is readily available. Perhaps this will be a viable future solution.

The basic strategy still remains no matter how fast computers become: the ability to use parasitic capacitances for a useful purpose. Perhaps the ultimate analog CAD software will use a variety of methods depending on the circumstance involved. No matter what direction is taken, the challenge still lies ahead: general but accurate analog design automation software.

FUTURE WORK

OTACC

As previously mentioned, the current software is controlled by a top-level script. Ideally, this control script would be replaced by a graphical user interface which would run FiltorX, OTACC, and Spice. Also, the current implementation uses Magic for the minor task of extracting the layout. Since the extractor code is just one of many features in Magic, most of Magic's code is unused and seen as overhead to OTACC. It would be preferable to use a small stand alone conversion program that doesn't carry all the unneeded features that Magic maintains besides the extractor. Currently, from the time that a filter specification is given to OTACC until a Spice file is produced, the bottleneck encountered is the overhead involved with loading and starting Magic.

Another area to expand into is the ability to handle lossy filters. Currently, only lossless filters are allowed. This limits the software even more in an already limited configuration scope.

Finally, it would be interesting to investigate the use of using the filters generated as standard cells themselves. The cascading of low order (typically second order) filters creates filters of higher order. This software is already capable of creating higher order filters. However, it does not have the capability to reuse its own designs as would be required for the cascading of biquads. This would be an interesting area to expand the capabilities of OTACC.

Transconductance-C filters

One of the problems with transconductance-C methods is the need to tune the transconductances to obtain the desired characteristics. Research needs to continue in automatic tuning to truly bring "push button" filters into a product on a production line. The need to individually tune each filter is seen as too time consuming to be profitable. And ultimately, profitability is the driving force of acceptance into industry.

REFERENCES

1. J.M. Cohn, D.J. Garrod, R.A. Rutenbar, and L.R. Carley, 'KOAN/ANAGRAM II: New tools for device-level analog placement and routing,' *IEEE Journal of Solid-State Circuits*, vol. SC-26(3), pp. 330-342, March 1991.

2. H. Koh, C. Sequin, and P. Gray, 'OPASYN: A compiler for CMOS operational amplifiers,' *IEEE Trans. on Computer-Aided Design*, vol. CAD-9(2), pp. 113-125, February 1990.

3. R. Harjani, R.A. Rutenbar, and L.R. Carley, 'OASYS: A framework for analog circuit synthesis,' *IEEE Trans. on Computer-Aided Design*, vol. CAD-8(12), pp. 1247-1266, December 1989.

4. B.R. Owen, R. Duncan, S. Jantzi, C. Ouslis, S. Rezania, K. Martin, 'BALLISTIC: An Analog Layout Language,' University of Toronto WWW server, *http://www.eecg.toronto.edu.*

5. V. M. Bexten, C. Moraga, R. Klinke, W. Brockherde, and K. Hess, 'ALSYN: Flexible Rule-Based Layout Synthesis for Analog IC's,' *IEEE Jour. of Solid State Circuits*, Vol. 28, No. 3, pp. 261-267, March 1993

6. P. Horowitz and W. Hill, *The Art of Electronics*, Cambridge: Cambridge University Press, 1989.

7. J. E. Franca, Y. Tsividis Eds., *Design of Analog-Digital VLSI Circuits for Telecommunications and Signal Processing*, 2nd Edition, Prentice-Hall, Englewood Cliffs, 1994.

8. W. J. Helms and K. C. Russel, 'Switched capacitor filter compiler,' *Proc. Custom Integrated Circuit Conf.*, 1986.

9. H. Yaghutiel, A. Sangiovanni-Vincentelli, and P. R. Gray, 'method for the automated layout of switched capacitor filters,' *Proc. Intl. Conf. CAD (ICCAD86),* 1986.

10. G. V. Eaton, D. G. Nairn, W. M. Snelgrove, and A. S. Sedra, 'SICOMP: A Silicon compiler for switched-capacitor filters,' *Proc. IEEE Int. Symp. Circuits and Syst.*, pp. 321-324, May 1987.

11. Coordinate Free LAP Reference Manual, Version 1.2, University of Washington, Northwest LIS Release 3.2, Seattle, WA, September 1988.

12. FiltorX Version 3.3, Department of Electrical Engineering, University of Toronto, 1993.

13. J. Ousterhout, 'The Magic layout system,' *IEEE Design Test Comput.*, 2, 19-30, 1985.

14. W. R. Daasch, M. Wedlake, R. Schaumann, and P. Wu, 'Automation of the IC layout of continuous-time transconductance-capacitor filters,' *Int. Journal of Circuit Theory and Applications*, 20, 267-282, 1992.

15. W. R. Daasch, M. Wedlake and R. Schaumann, 'Automatic Generation of CMOS continuous-time elliptic filters,' *Electronics Letters*, 28, no. 24, pp. 2215-2216, 1992.

16. M. A. Tan and R. Schaumann, 'Generation of transconductance-grounded-capacitor filters by signal-flow-graph methods for VLSI implementation,' *Electronics Letters*, Vol. 23, No. 20, September 1987.

17. D. Robinson and W. R. Daasch, 'Design Automation of the IC Filter Design Cycle,' *to be submitted.*

18. R. Schaumann, M. Ghausi, and K. Laker, *Design of Analog Filters*, Prentice Hall, Englewood Clifts, CA 1990.

19. W. R. Daasch and M. Wedlake, 'Rapid Layout of a Continuous-Time Transconductance-C Filter,' *Proc. Int. Symp. on Circuits and Systems, IEEE*, pp. 2256-2259, 1992.

20. M. A. Driscoll, W. R. Daasch, and C. Sembakutti, 'Efficient Design Centering of Analog Integrated Circuits Using Binary Search,' *Analog Integrated Circuits and Signal Processing*, vol. 6(2), pp. 157-169, September 1994.

21. S. R. Brotman, 'The Evaluation of Device Model Dependance in the Design of a High-Frequency, Analog, CMOS Transconductance-C Filter', Masters thesis at Portland State University, May, 1994.

APPENDIX A

TEST CHIP WAVEFORMS

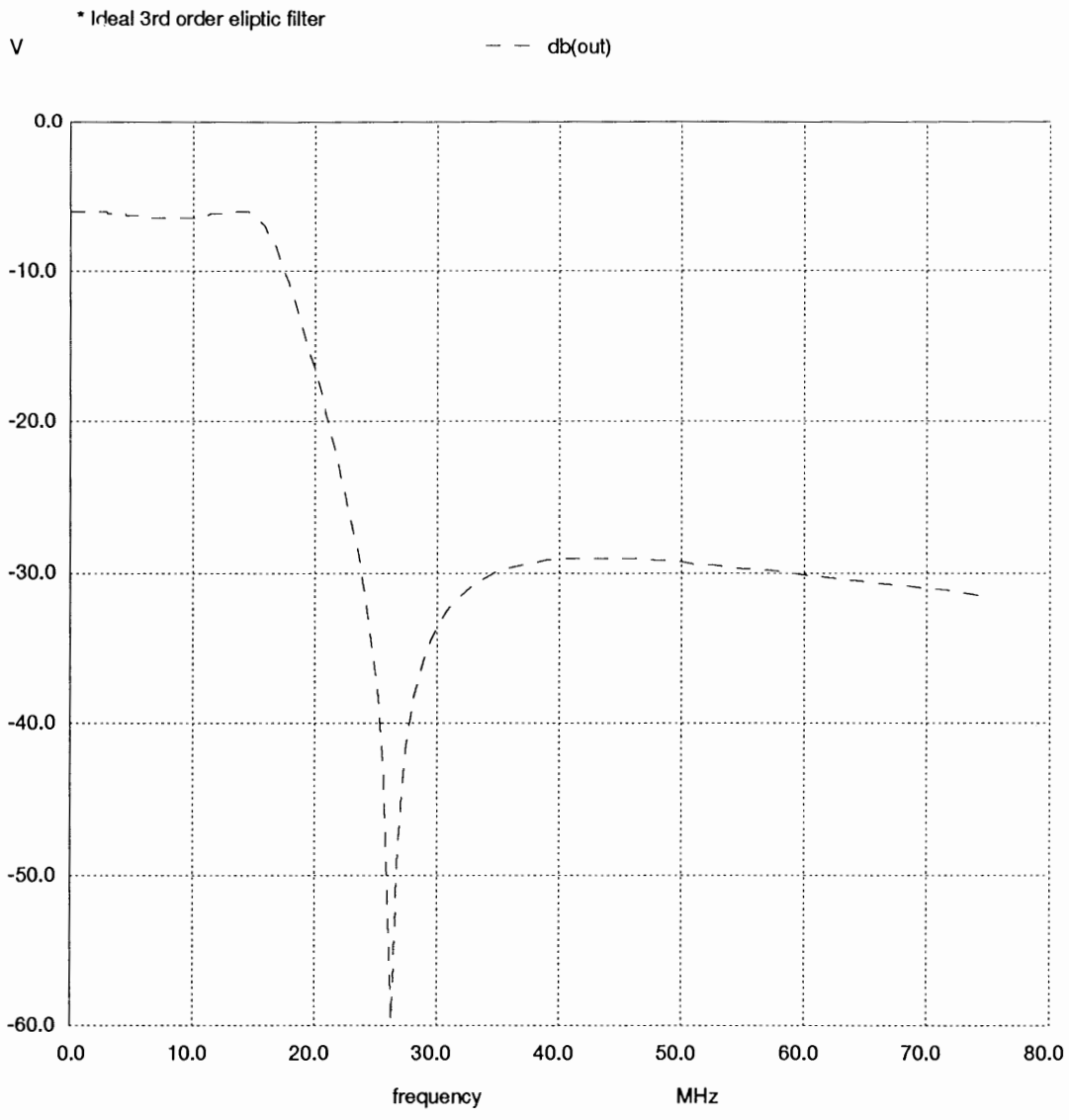


Figure 39. Third-order elliptic ideal magnitude response

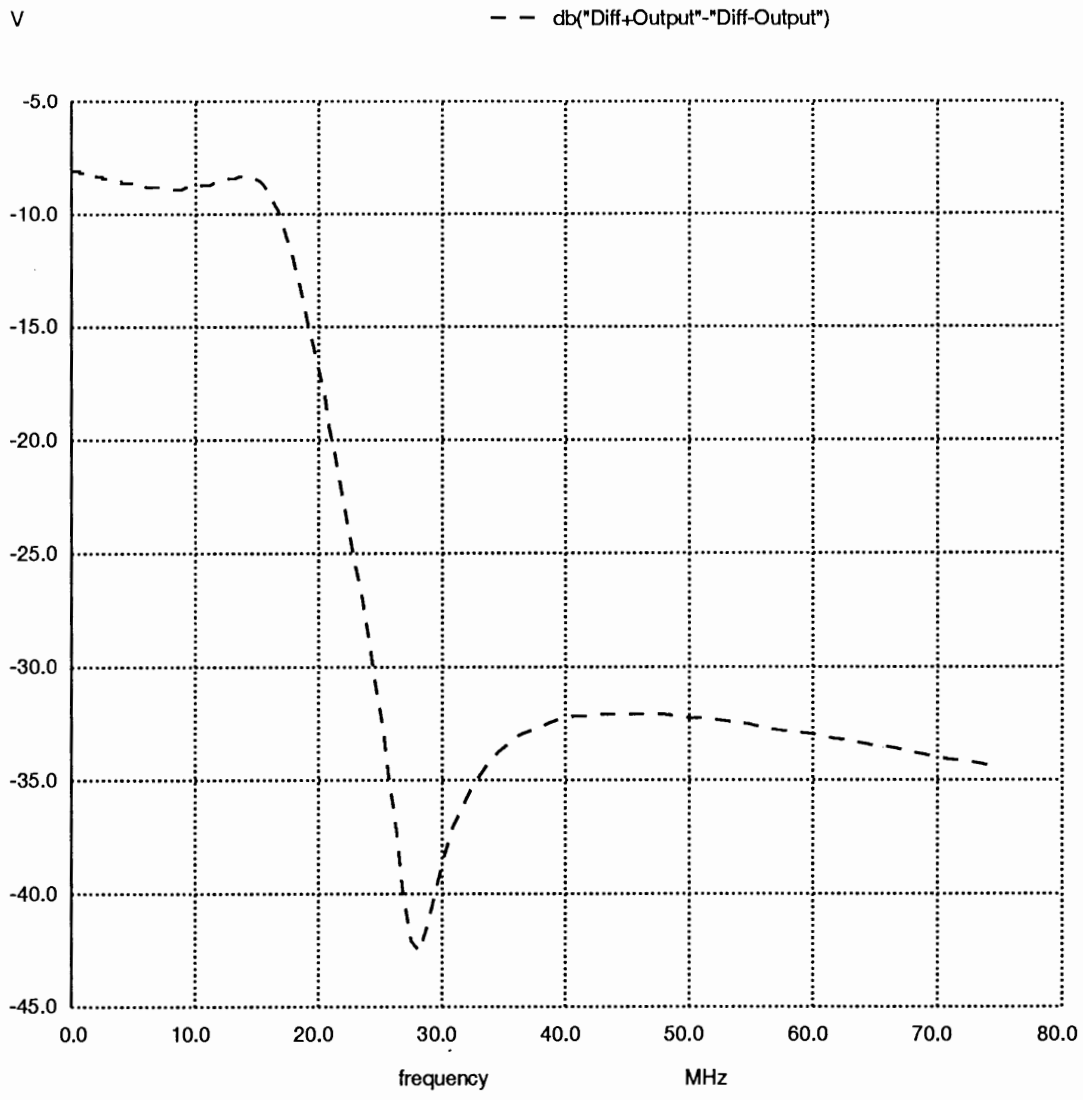


Figure 40. Third-order elliptic simulated magnitude response

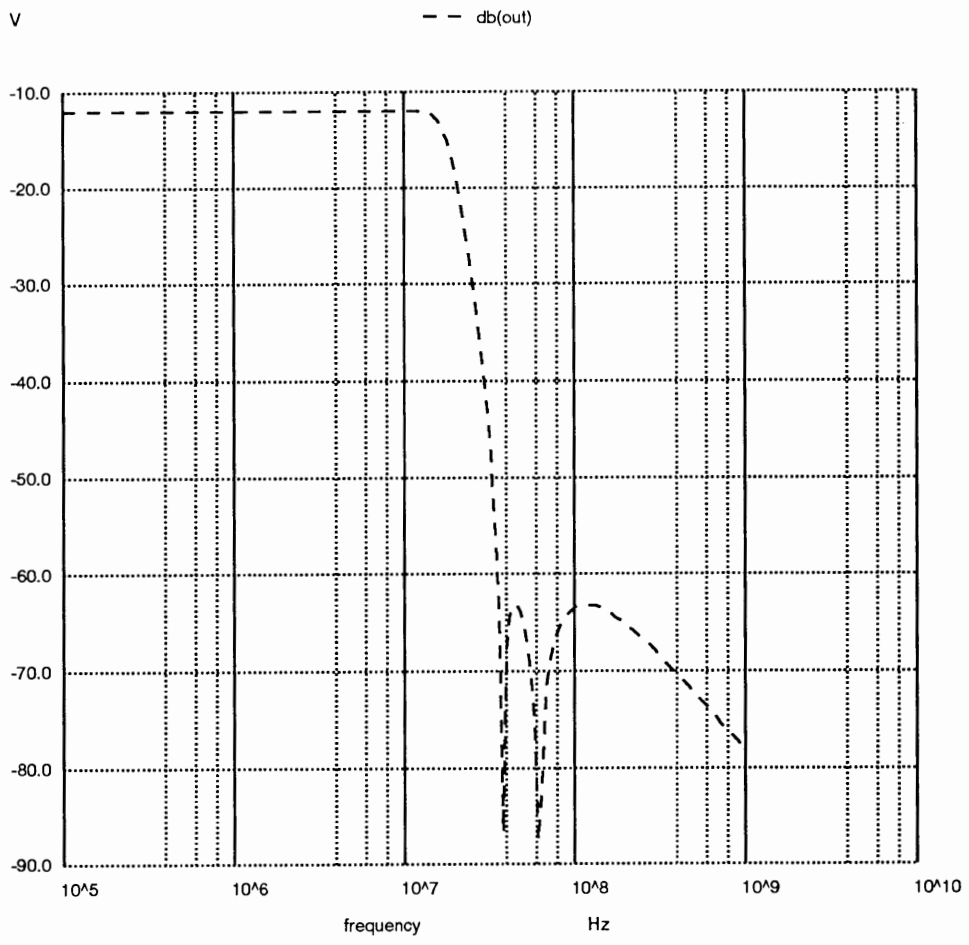


Figure 41: Fifth-order inverse Chebyshev ideal magnitude response

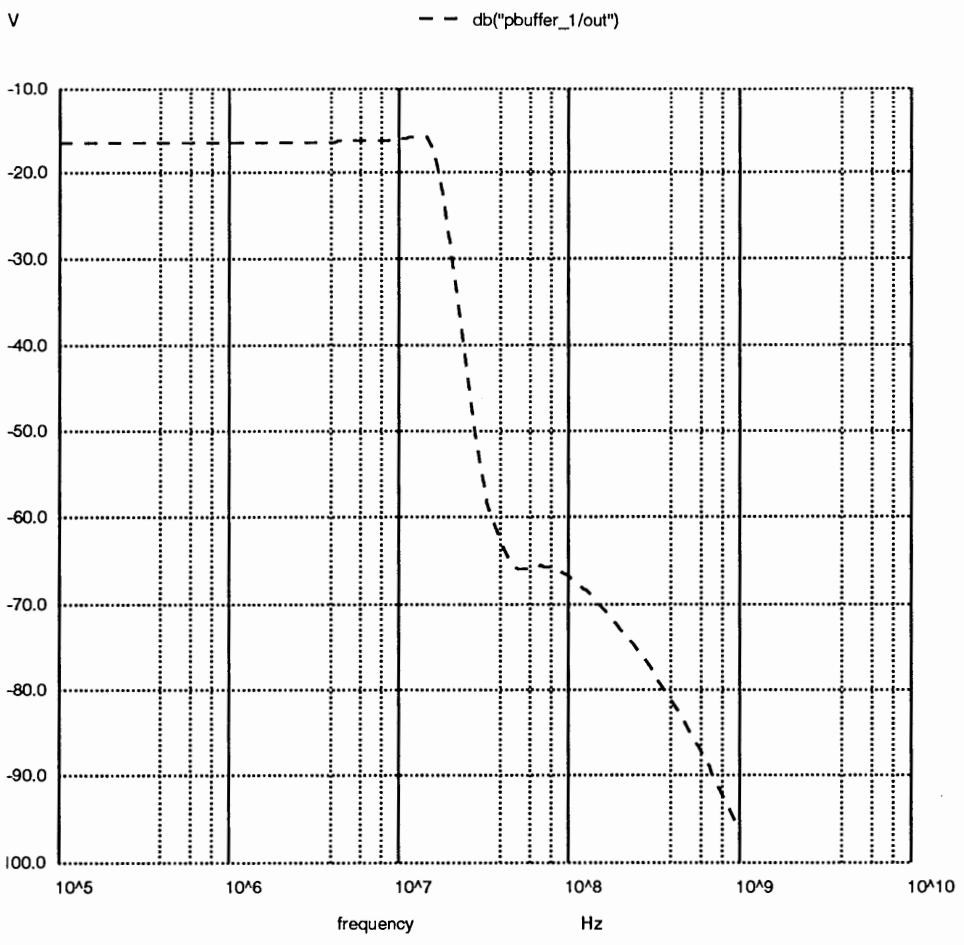


Figure 42. Fifth-order inverse Chebyshev simulated magnitude response

APPENDIX B

CHIP PLOTS

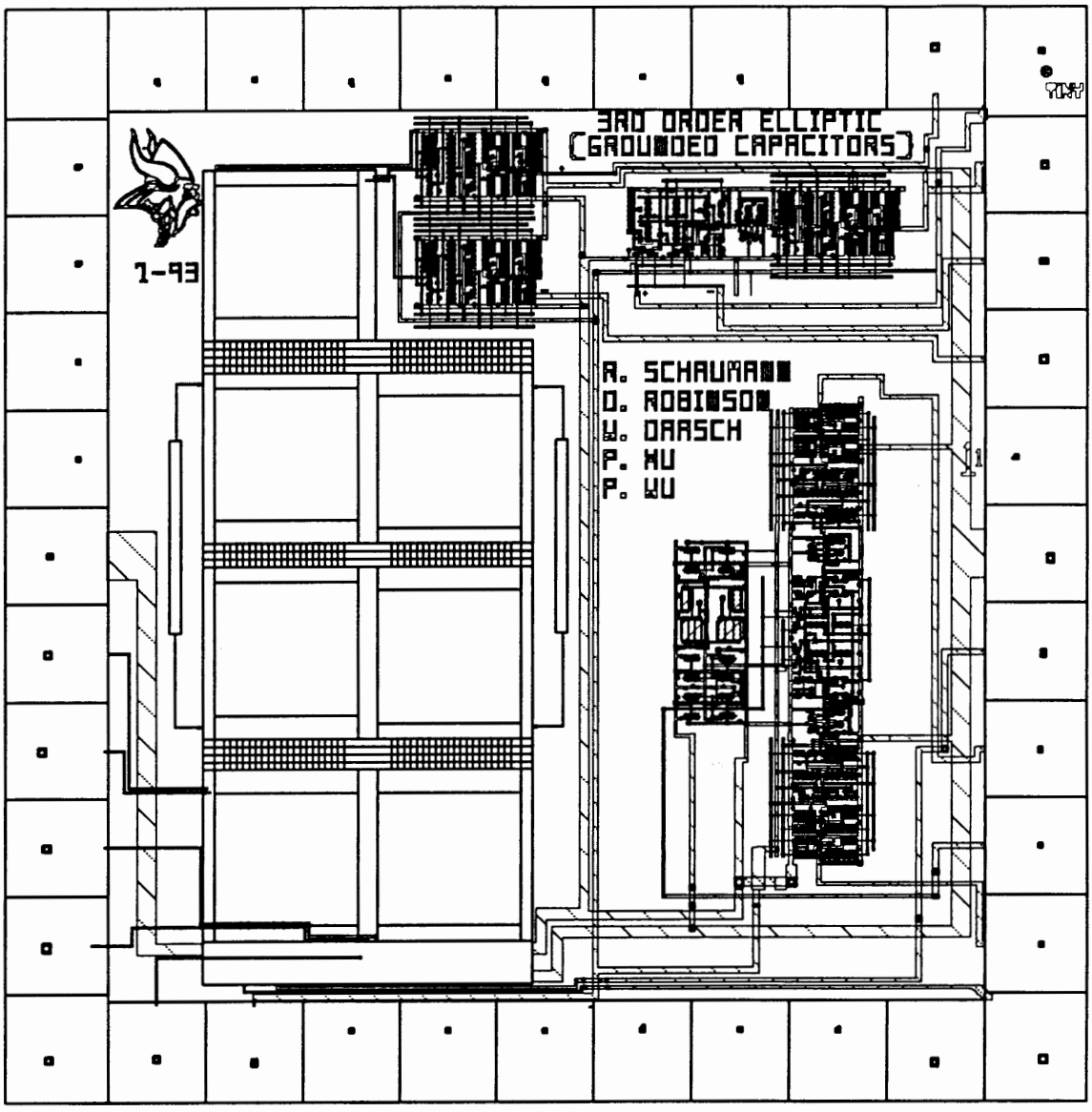


Figure 43. Chip plot of third-order elliptic filter

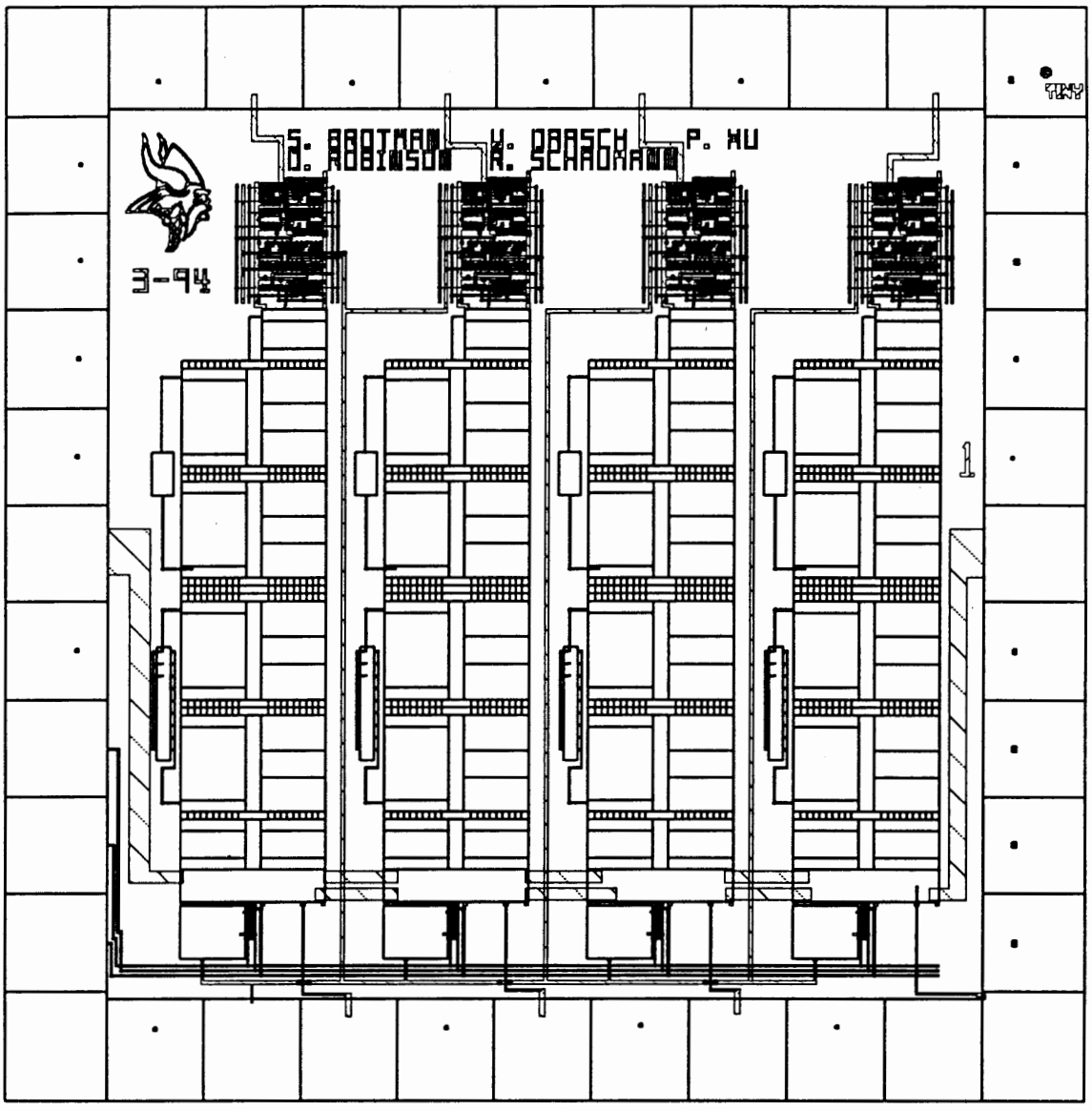
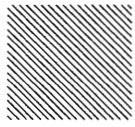


Figure 44. Chip plot of fifth-order inverse Chebyshev filter

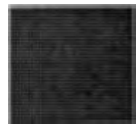
APPENDIX C

TILE LIBRARY

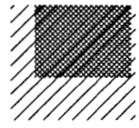
APPENDEX C LEGEND



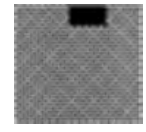
Metal1



Poly1

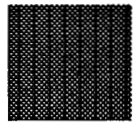


N Diffusion

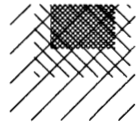


Metal2 contact

Metal2



Poly2



P Diffusion

Poly Contact

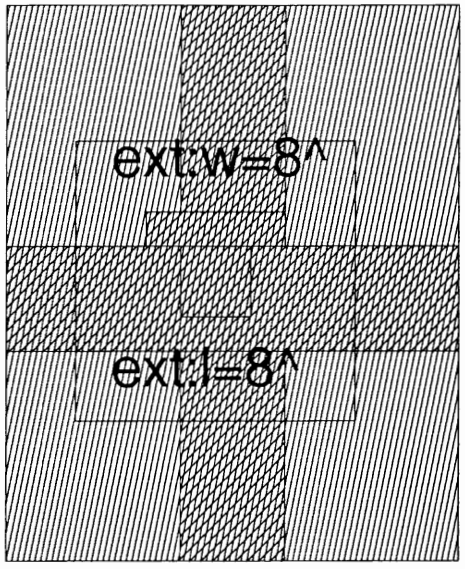


Figure 45. Unit capacitor

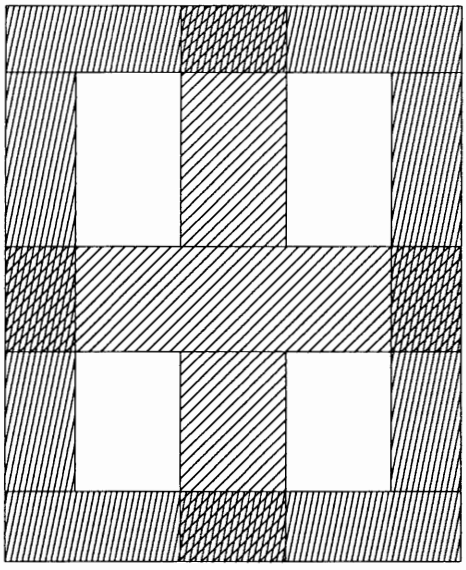


Figure 46. Null capacitor

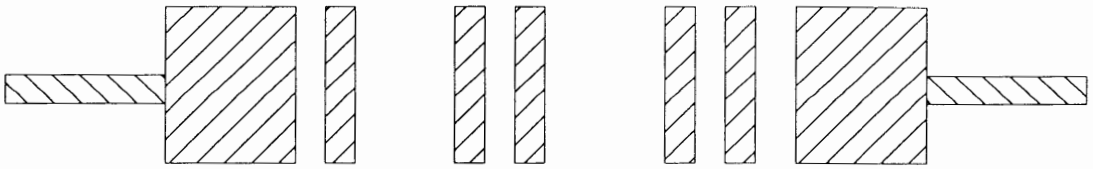


Figure 47. DC route bridge for capacitor arrays

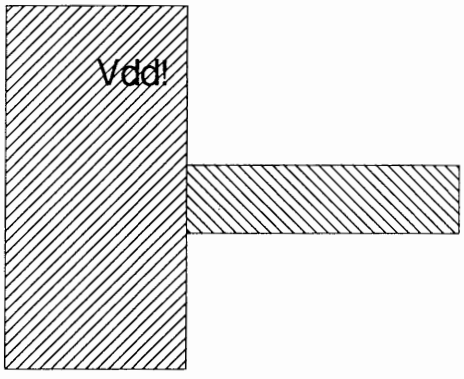


Figure 48. Power rail bridge for capacitor arrays

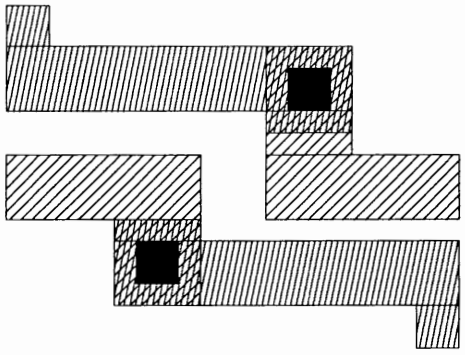


Figure 49. Zero array crossover bridge

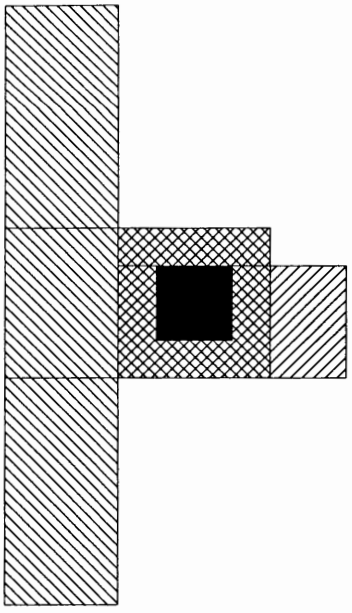


Figure 50. Zero array connection tile

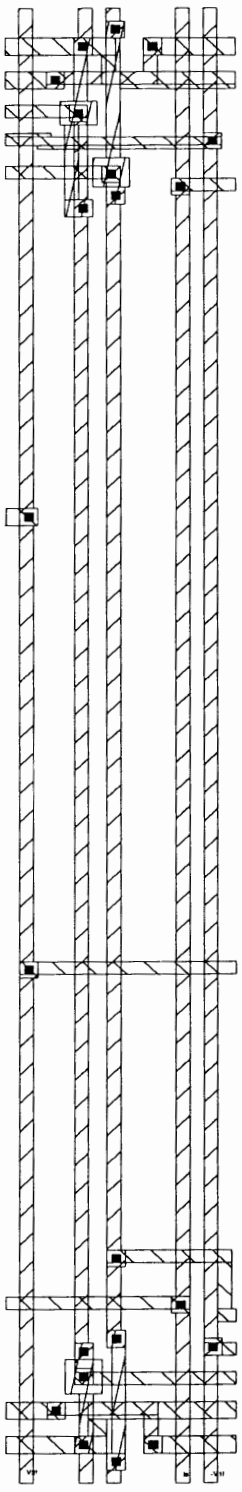


Figure 51. DC route

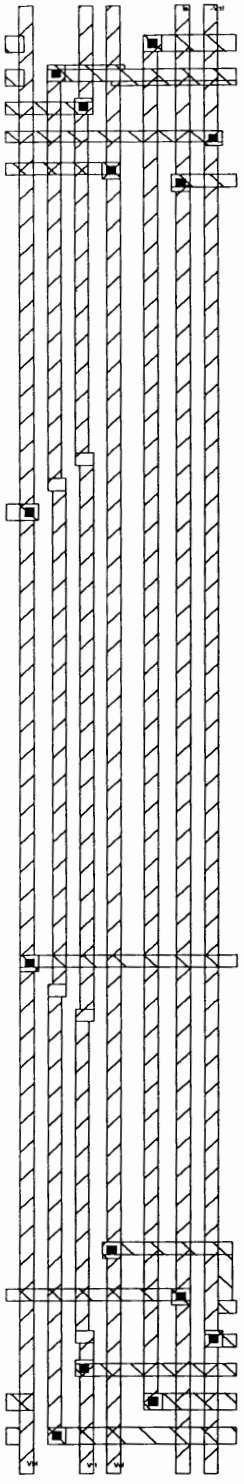


Figure 52. Load DC route

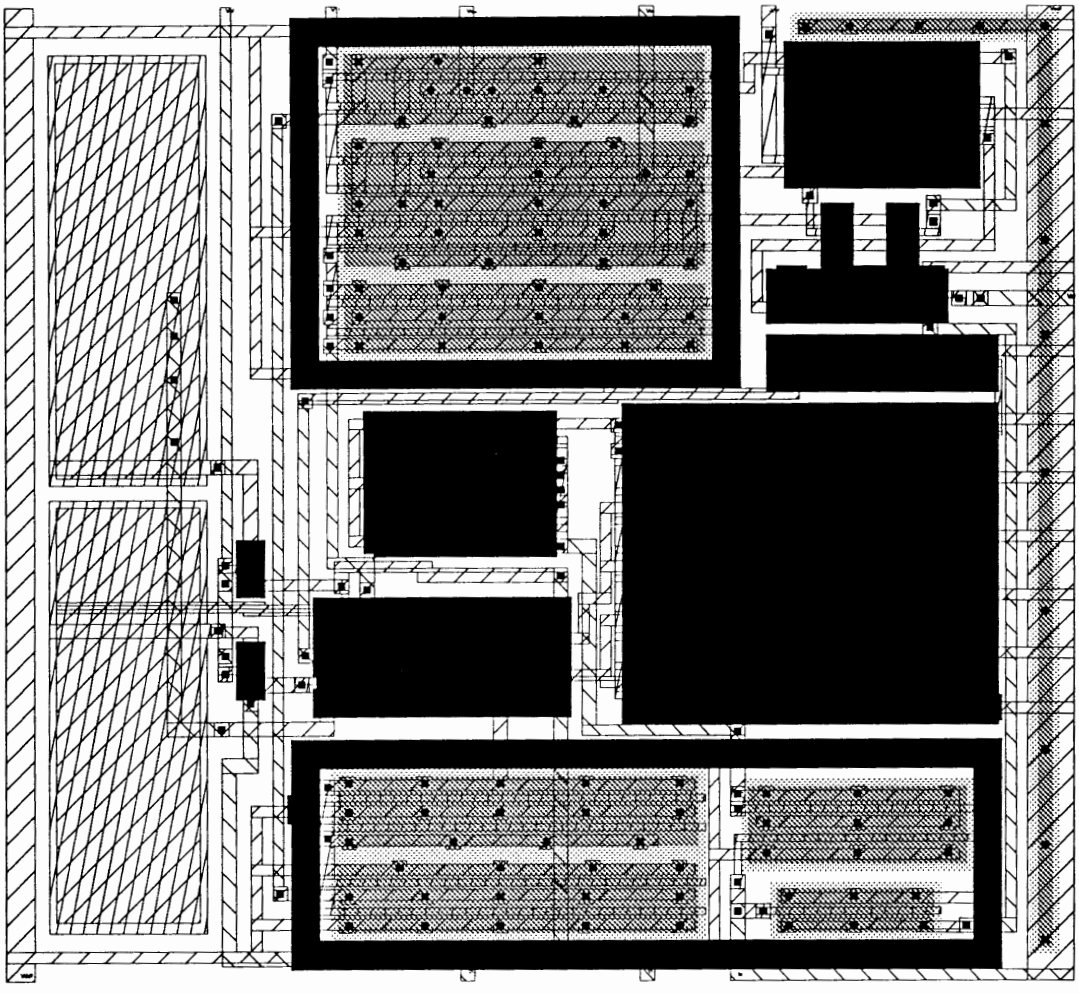


Figure 53. Differential operational transconductance amplifier

CAPACITOR RIBS

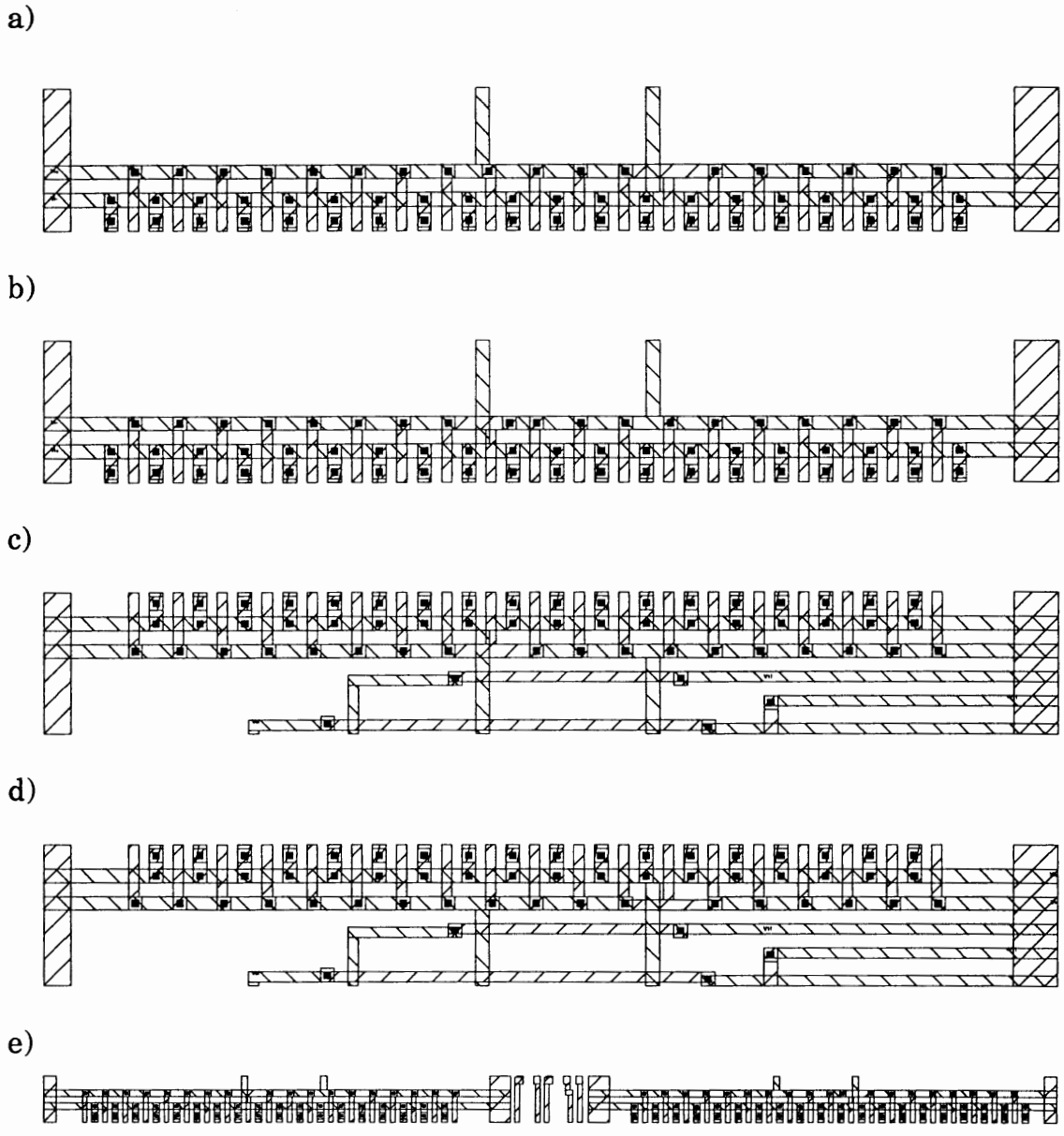


Figure 54. (a) Input comb: negative (b) Input comb: positive
(c) Output comb: negative (d) Output comb: positive
(e) Compressor

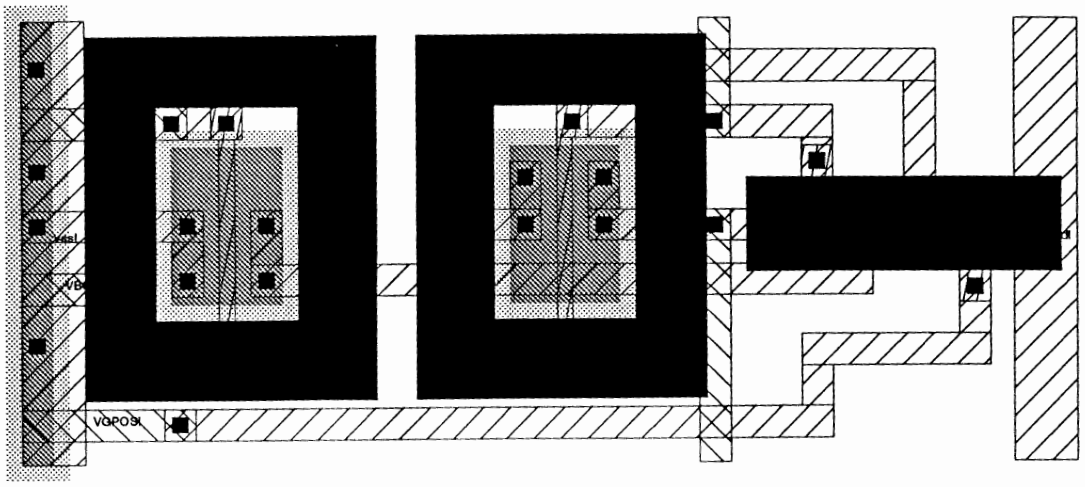


Figure 55. Single-ended inverting transconductance

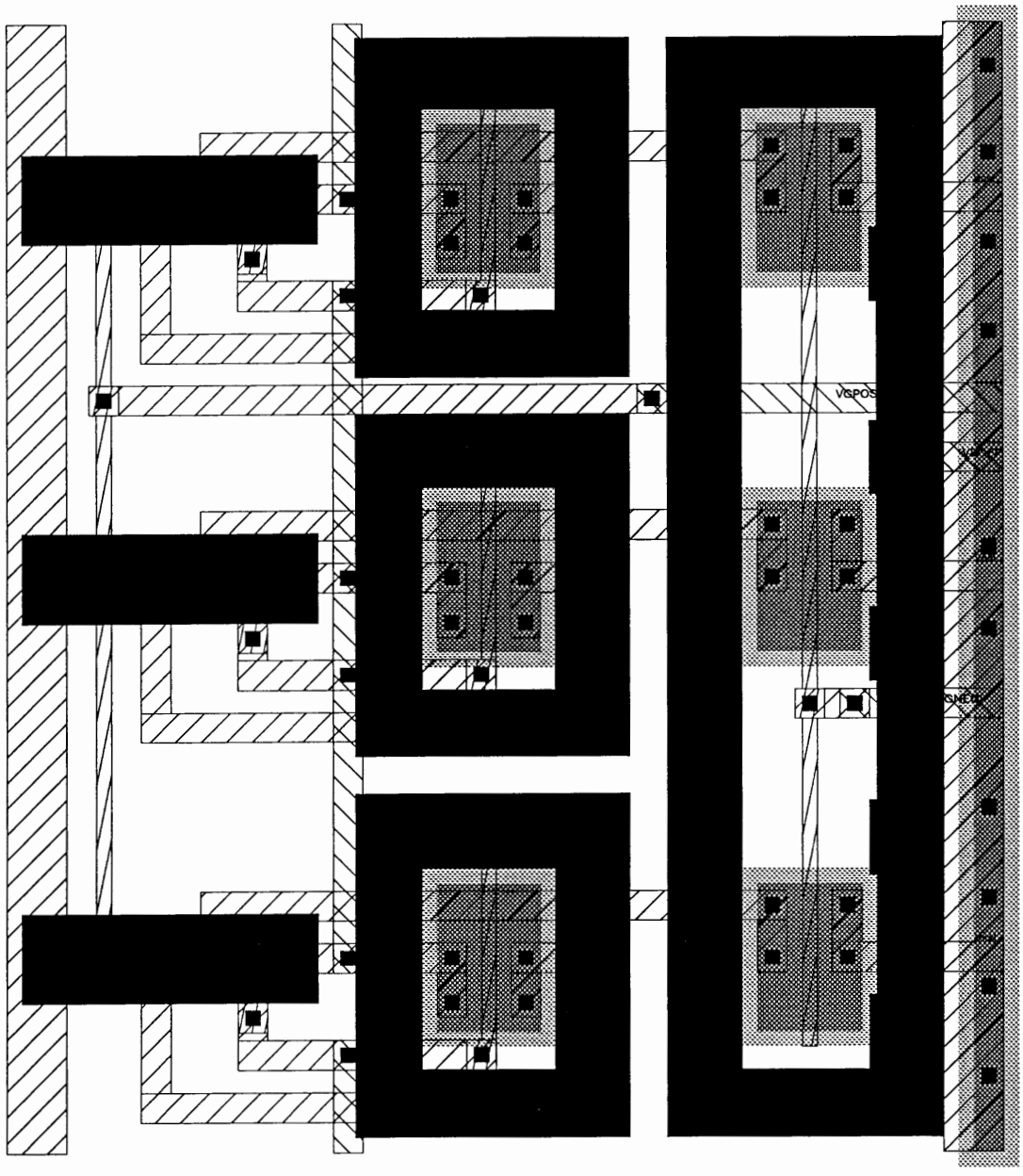


Figure 56. Single-ended non-inverting transconductance