A LAYOUT GENERATOR FOR STRAY INSENSITIVE SWITCHED CAPACITOR FILTERS

by

JOHN ERIK LINDHOLM

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

FEBRUARY, 1986

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of ELECTRICAL ENGINEERING

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date MARCH 20, 86

Abstract

Two software programs have been developed to automatically generate the layout of stray insensitive switched capacitor filters. The first program, CIRCE, uses a SWITCAP filter description to generate an intermediate format filter description that is suitable for direct layout. It groups the filter elements into integrators and then sorts these integrators and their elements. The SWITCAP description can be with or without switch-sharing. The second program, SISCL, lays out the filter in CIF format using a double-polysilicon ISOCMOS process. The layout is done with a capacitor array placed over the switches and the standard-cell operational amplifiers placed beneath the switches. The layout is done with full switch-sharing. Capacitors may be rearranged in the intermediate file to produce better layouts. The software is written in the C language.

Table of Contents

List of Tables

List of figures

List of figures

List of figures

List of figures

List of figures

Acknowledgement

I would like to thank my supervisors, Dr. L. Young, and Dr. D. Moore for their guidance and support during the course of this work.

I would also like to acknowledge the financial and technical help given by the people at Microtel Pacific Research Ltd. (MPR), especially Mr. P. Thiel, and Mr. H. Schweikle.

Thanks are extended to Mr. C. Huntley, Mr. K. Wei, and Mrs. E. Willoner, of MPR for their technical guidance.

I also wish to thank my parents for their continual support of my education.

1. INTRODUCTION

Monolithic switched capacitor filters were introduced in 1977 [1][2], and the technology has advanced rapidly since then. Stray insensitive switched capacitor filters based on passive LC ladder prototypes are commonly used today in commercial products, as a variety of methods exist for transforming the prototypes into switched capacitor filters [3]-[9]. Programs such as SWITCAP (see Appendix B), provide accurate performance analysis of these filters and the technology may be considered mature [10].

The layout of stray insensitive switched capacitor filters is still mainly done by hand; however, the maturity of the design techniques provides incentive to develop a program that can automatically generate the layout from a filter description.

Such a program has been developed in this thesis. It uses a SWITCAP filter description to generate the layout in Caltech Intermediate Form (CIF) [11]. The software is written in the C language and was developed on the Metheus λ750 system at the University of British Columbia.

2. ISOCMOS PROCESS TECHNOLOGY

2.1 THE FABRICATION PROCESS

The ISOCMOS (ISolation Oxide CMOS) process uses a single p-well in an n- substrate. The process is characterized by the steps tabulated below with the CIF code for each layer given after the layer.

1. Active (AC)
2. P-well implant (PW)
3. Polysilicon1 (P1)
4. Polysilicon2 (P2)
5. N+ Implant (NP)
6. P+ Implant (PP)
7. Contact cuts (C1)
8. Metallization (M1)
9. Passivation (GL)

A thin oxide layer separates the polysilicon1-polysilion2 layers producing a high capacitance. A 5μm process was used.

2.2 CMOS SWITCHES

It is possible to implement an excellent switch in CMOS technology (Fig. 2.1). The clock ϕ_1 controls the switch. A simple non-ideal model of a CMOS switch is shown in Fig. 2.2. The on resistance is typically 1kΩ, and the off resistance is in the 1TΩ range, producing a conductance ratio of a billion to one.

Capacitors CGS and CGD represent parasitic capacitances between the gate and the switch terminals. These parasitics are in the 1fF range and contribute to a feedthrough effect where a portion of the control voltage appears at the switch terminals. These parasitics are caused by lateral diffusion under the gate causing a gate-source and gate-drain overlap. Capacitors CSS and CDS represent parasitic capacitances from the switch terminals to the substrate and their values depend on the drain and source areas, but they are in the 1pF range.

The switches are laid out in pairs since the switched capacitors are alternately switched from ground to the circuit. The switches are overlapped to reduce the overall area, resulting in a shared source and drain for two of the

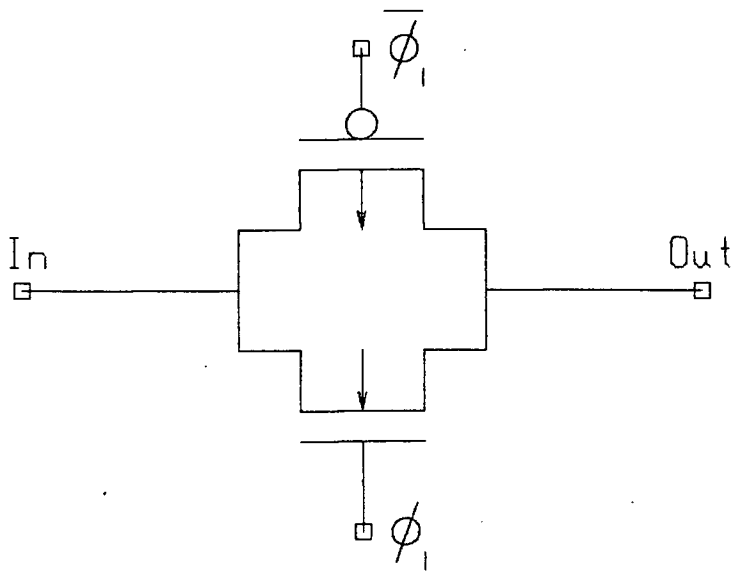


Fig. 2.1: CMOS switch schematic

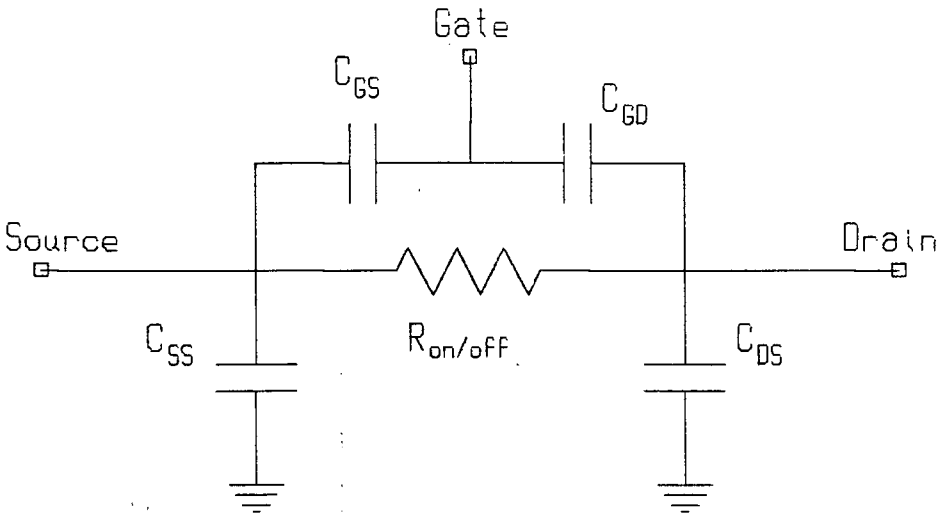


Fig. 2.2: CMOS switch model

transistors (Fig. 2.3). The switches are of minimum geometry to minimize the switch-induced error voltage [12].

2.3 CMOS RESISTORS

Resistors can be fabricated in a number of ways using the p-well, polysilicon, or active regions. The absolute accuracy of these resistors is typically 50%, making precision resistors impossible. Table 2.1 summarizes the effective resistances achieved by these methods.

2.4 CMOS CAPACITORS

Three types of capacitors suitable for analog sampled data applications can be fabricated. One is between the gate and the channel, another is between the metal and the

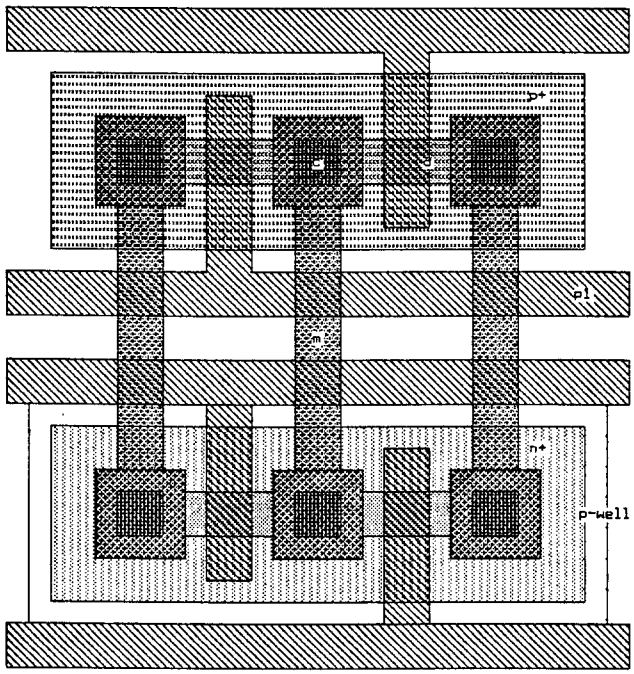


Fig. 2.3: Layout of a switch-pair

Table 2.1: Layer resistances

Layer	Resistance (ohms/□)
p-well	1000
p+ active	90
n+ active	10
polysilicon1	40
polysilicon2	55
metal	0.023

substrate, and the third is between the polysilicon layers.

The gate-oxide-channel capacitor has a value in the 0.4 pF/μ^2 range. Its main disadvantage is that a voltage-dependent parasitic capacitance exists from the channel to the substrate. This parasitic capacitance has a value from 5% - 20% of the desired capacitance, and this renders the gate-oxide-channel capacitor useless for switched capacitor filters.

The metal-substrate capacitor and the polysilicon1-polysilicon2 capacitors do not have any voltage dependent parasitic capacitances. The polysilicon1-polysilicon2 capacitor (Fig. 2.4) has a slightly thicker oxide layer than the gate-oxide-channel capacitor and hence its capacitance is slightly less. The metal-substrate capacitor oxide is some twenty times thicker resulting in a much smaller capacitance. For these reasons, the polysilicon1-polysilicon2 capacitor is the preferred choice in switched capacitor filters. The capacitances are summarized in Table 2.2.

All capacitors have parasitic capacitances associated with them. For the polysilicon1-polysilicon2 capacitor, the top plate parasitics are mainly due to circuit connections, and the overlap capacitors of the drain or source to the bulk of switches attached to them. These parasitics are modelled as capacitors to ground (Fig. 2.5). The top plate parasitic capacitance is typically 0.1% - 1.0% of the desired capacitance, and the bottom plate parasitic

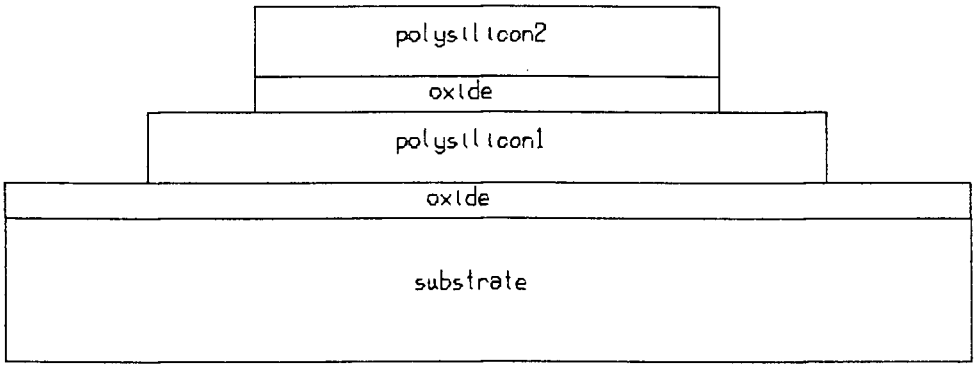


Fig. 2.4: Polysilicon1-polysilicon2 capacitor

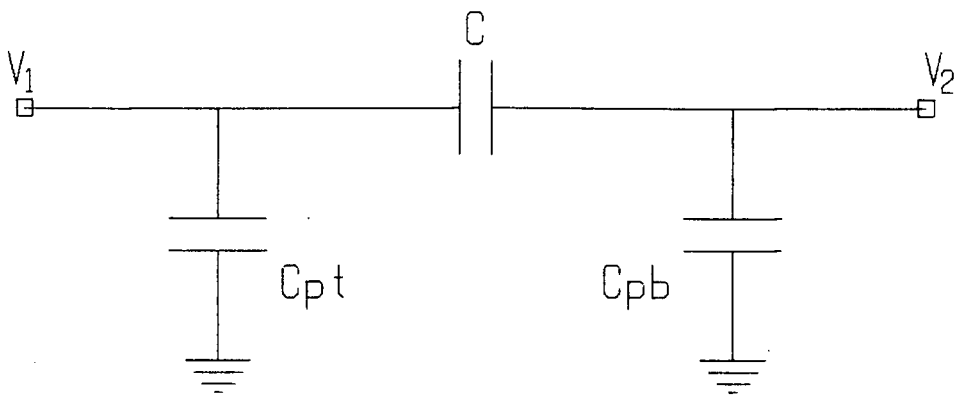


Fig. 2.5: Parasitic capacitances

Table 2.2: Layer capacitances

Layers	Capacitance (FF/um²)
gate - channel	0.42
poly1 - poly2	0.40
poly1 - substrate	0.024
metal - substrate	0.019

capacitance is typically 10% of the desired capacitance. These parasitics are unavoidable, and their size depends on the layout, the technology, and the capacitor size.

The absolute accuracy of a capacitor is a few percent and is edge dependent. This error is due to oxide undercutting in the photolithography process, and to mask resolution. These errors are associated with the periphery of the capacitors.

3. THE SWITCHED CAPACITOR

3.1 THE PARALLEL SWITCHED CAPACITOR

Consider the switched capacitor (SC) network of Fig. 3.1, clocked by the ϕ_1 and ϕ_2 waveforms of Fig. 3.2. It will be shown that this circuit is equivalent to the circuit of Fig. 3.3. Assume that V_1 and V_2 are two independent DC voltage sources that are constant during most of the phase period.

Assume initially that both switches are open and that the capacitor C is uncharged. During ϕ_1, the capacitor will charge to V_1, and the charge transferred into the capacitor C is

$$Q(T/2) = CV_1 \tag{3.1}$$

During ϕ_2, the charge transferred into the capacitor is

$$Q(T) = C(V_2-V_1) \tag{3.2}$$

and during ϕ_1 again, the charge transferred into the capacitor is

$$Q(3T/2) = C(V_1-V_2) \tag{3.3}$$

and steady-state has been reached. Let

$$R = \frac{V_1-V_2}{I_1} = \frac{V_2-V_1}{I_2} \tag{3.4}$$

The current at a point in the circuit is

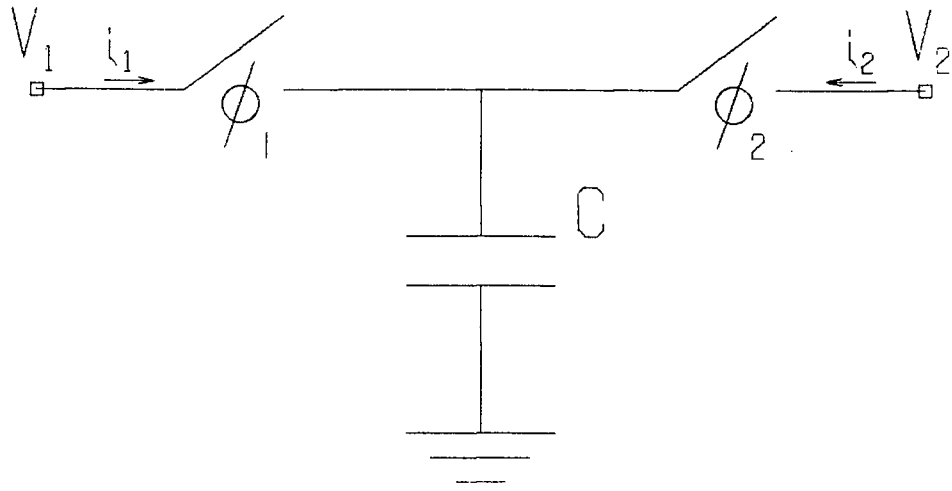


Fig. 3.1: Parallel switched capacitor

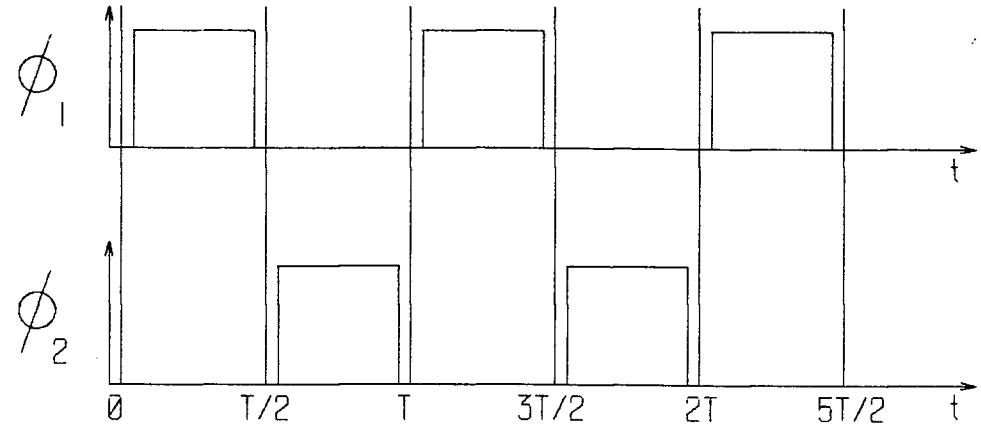


Fig. 3.2: Clock waveforms

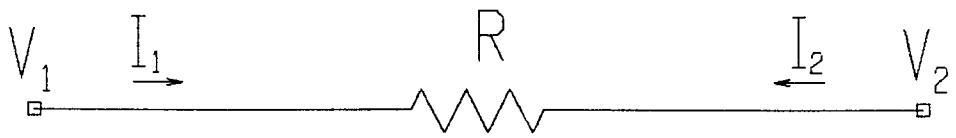


Fig. 3.3: Resistor equivalent

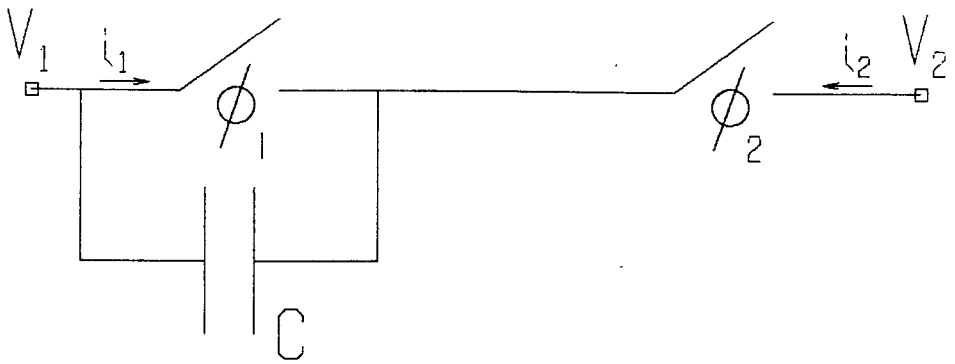


Fig. 3.4: Series switched capacitor

$$i = \frac{dq}{dt} \tag{3.5}$$

Consider the flow of charge in the sense of i_1 in Fig. 3.1. The total charge flow is Q_1

$$Q_1 = \int_T^{3T/2} i_1 dt = \int_{T/2}^{3T/2} i_1 dt \tag{3.6}$$

since no charge flows during the previous phase period. Equating 3.3 to 3.6 and dividing by T results in

$$I_1 = \frac{Q(3T/2)}{T} = \frac{1}{T} \int_{T/2}^{3T/2} i_1 dt \tag{3.7}$$

or

$$I_1 = \frac{C(V_1 - V_2)}{T} \tag{3.8}$$

Now substituting 3.4 into 3.8 gives

$$R = \frac{T}{C} \tag{3.9}$$

This switched capacitor implementation is called a parallel switched capacitor. It is also possible to implement a series switched capacitor as shown in Fig. 3.4. A similar analysis to the one given above for the parallel case yields the same relationship.

3.2 THE SWITCHED CAPACITOR INTEGRATOR

It is neccessary to introduce the operational amplifier to completely transfer the charge of one capacitor to another capacitor. All operational amplifiers are considered ideal for the following analysis. The RC integrator of Fig. 3.5 is the basic building block of active filters. The transfer function of this integrator is

$$\frac{V_2(s)}{V_1(s)} = \frac{-1}{sRC_1} \qquad (3.10)$$

Replacing the resistor R by the parallel switched capacitor of Fig. 3.1 results in the circuit of Fig. 3.6. This circuit can be analyzed by z-transform techniques. During ϕ_1, the charges on the capacitors are

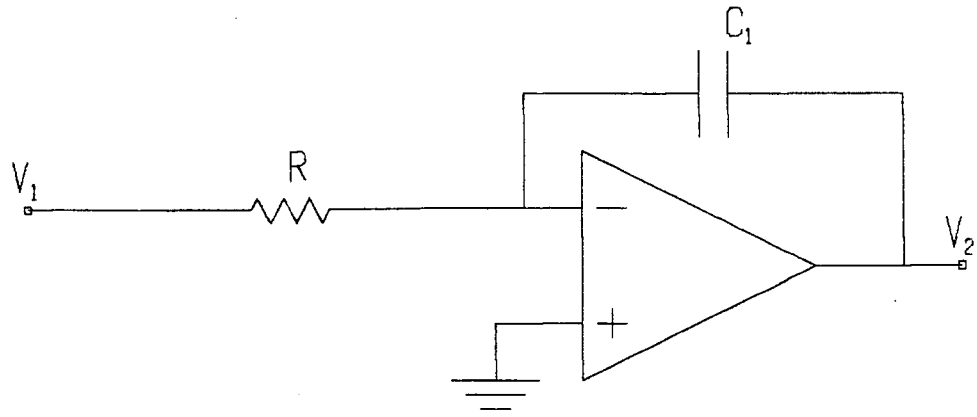


Fig. 3.5: RC integrator

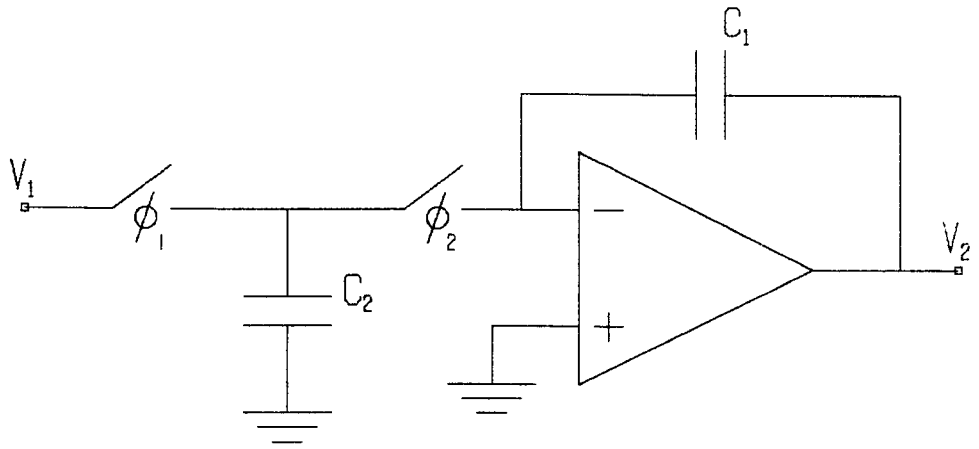


Fig. 3.6: Parallel SC inverting integrator

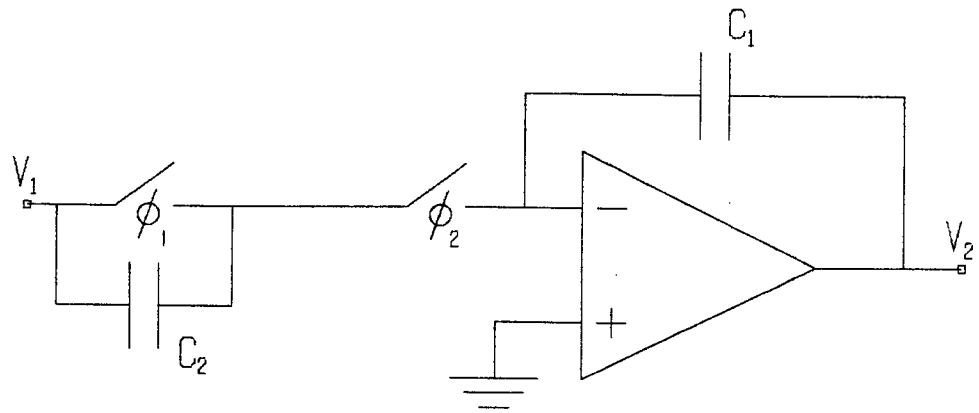


Fig. 3.7: Series SC inverting integrator

$$Q_1^1 = C_1 V_2^1(n-.5) \tag{3.11}$$

$$Q_2^1 = C_2 V_1^1(n-.5) \tag{3.12}$$

where the superscripted number refers to the clock phase. During ϕ_2, charge conservation yields

$$C_1 V_2^2(n) = C_1 V_2^1(n-.5) - C_2 V_1^1(n-.5) \tag{3.13}$$

or

$$V_2^2(z) = z^{-1/2} V_2^1(z) - \left(\frac{C_2}{C_1}\right) z^{-1/2} V_1^1(z) \tag{3.14}$$

On the transformation ϕ_2 to ϕ_1

$$V_2^2(n-1) = V_2^1(n-.5) \tag{3.15}$$

or

$$V_2^2(z) = z^{1/2} V_2^1(z) \tag{3.16}$$

Substituting 3.16 into 3.14 gives

$$z V_2^1(z) = V_2^1(z) - \left(\frac{C_2}{C_1}\right) V_1^1(z) \tag{3.17}$$

or

$$\frac{V_2^1(z)}{V_1^1(z)} = \left(\frac{-C_2}{C_1}\right)\left(\frac{z^{-1}}{1-z^{-1}}\right) \tag{3.18}$$

Now substituting 3.9 into 3.10 gives

$$\frac{V_2(s)}{V_1(s)} = \frac{-C_2}{sTC_1} \tag{3.19}$$

and comparing 3.19 with 3.18 results in

$$s = \frac{z-1}{T} \tag{3.20}$$

which is the forward transformation. Repeating this analysis for the series switched capacitor of Fig. 3.7 gives the transfer function

$$\frac{V_2^2(z)}{V_1^2(z)} = \left(\frac{-C_2}{C_1} \right) \left(\frac{1}{1-z^{-1}} \right) \tag{3.21}$$

and comparing 3.21 with 3.19 gives

$$s = \frac{1-z^{-1}}{T} \tag{3.22}$$

which is the backward transformation.

3.3 PARASITIC CAPACITANCE CONSIDERATIONS

The bottom plate of the parallel switched capacitor is grounded, hence the bottom plate parasitics are shorted across ground and have no effect. The top plate parasitic which is some 0.1% - 1.0% of the capacitor value degrades the capacitor accuracy when it charges and discharges.

To negate any parasitic capacitances, the switching schemes of the parallel and series switched capacitor inverting integrators are changed into those of Fig. 3.8 and Fig. 3.9. Notice that the top and bottom plates of the

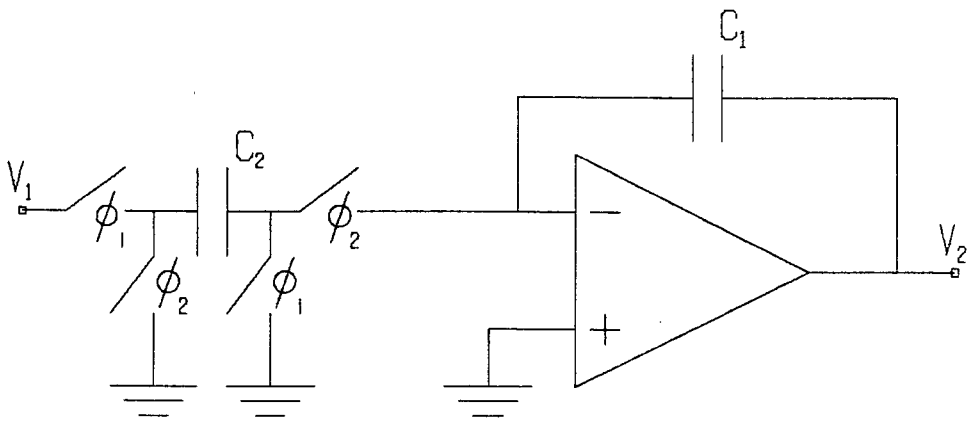


Fig. 3.8: Stray insensitive SC noninverting integrator

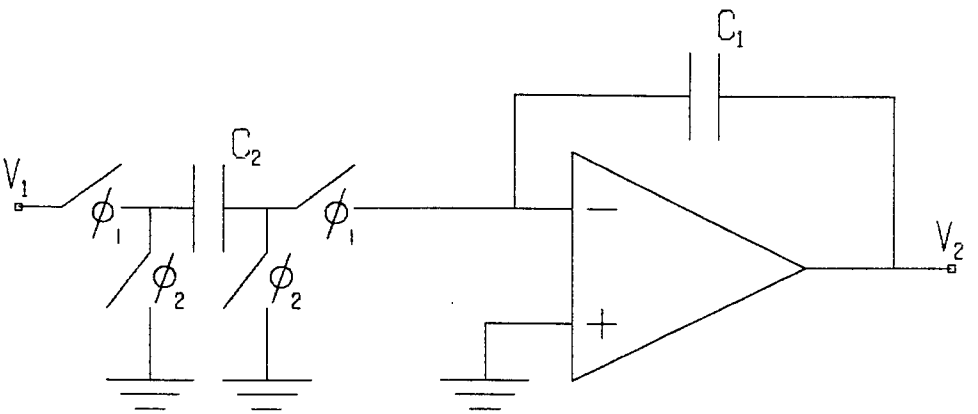


Fig. 3.9: Stray insensitive SC inverting integrator

parallel switched capacitor are now alternating to ground, and that the integrator is now noninverting. The transfer function of the integrator is

$$\frac{V_2(z)}{V_1(z)} = \left(\frac{C_2}{C_1}\right)\left(\frac{z^{-1}}{1-z^{-1}}\right) \qquad (3.23)$$

while the transfer function of the series integrator remains unchanged.

The parasitic capacitances of the parallel integrator are shown in Fig. 3.10. In this implementation, the stray capacitances have no effect on the transfer function of either integrator.

During ϕ_1, the switched capacitor of Fig. 3.8 looks like Fig. 3.11. The bottom plate parasitic CBP is shorted

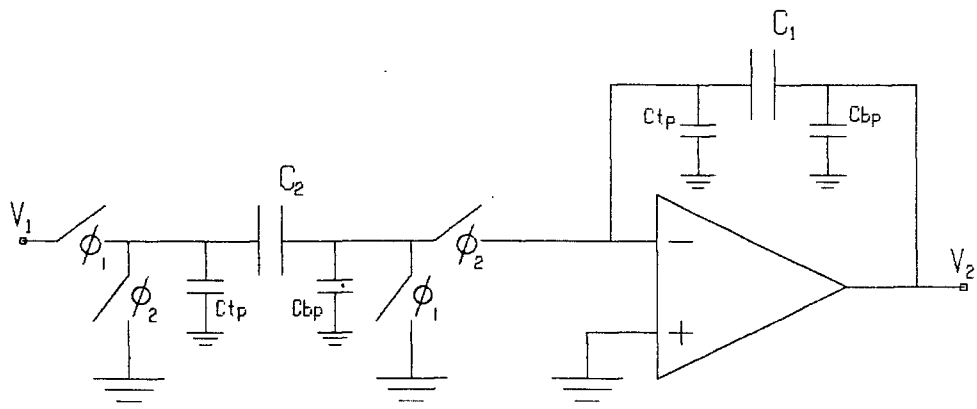


Fig. 3.10: Stray capacitances

across ground and the top plate parasitic CTP is charged to V_1 with the capacitor C. During ϕ_2 (Fig. 3.12), CTP is shorted across ground and has no effect on the charge transfer. Capacitor CBP is now virtually shorted across ground and hence has no effect either. The charge on C is transferred through the operational amplifier and to the integrating capacitor C_1. One plate of C_1 is always connected to the virtual ground, hence its parasitic capacitance has no effect while the other plate is always connected to the operational amplifier output. If the operational amplifier has a high gain, a good virtual ground is assured, and if it has a high gain and drive capability, the other parasitic has no effect.

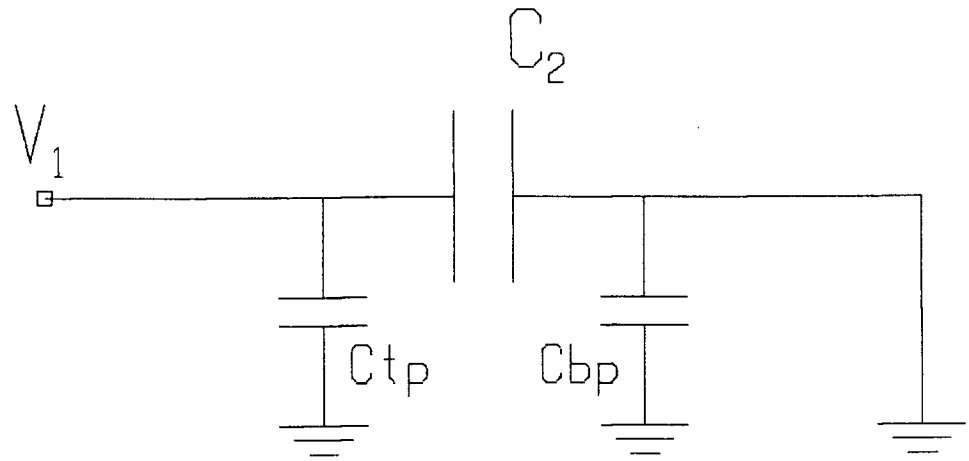


Fig. 3.11: Parallel switching during ϕ_1

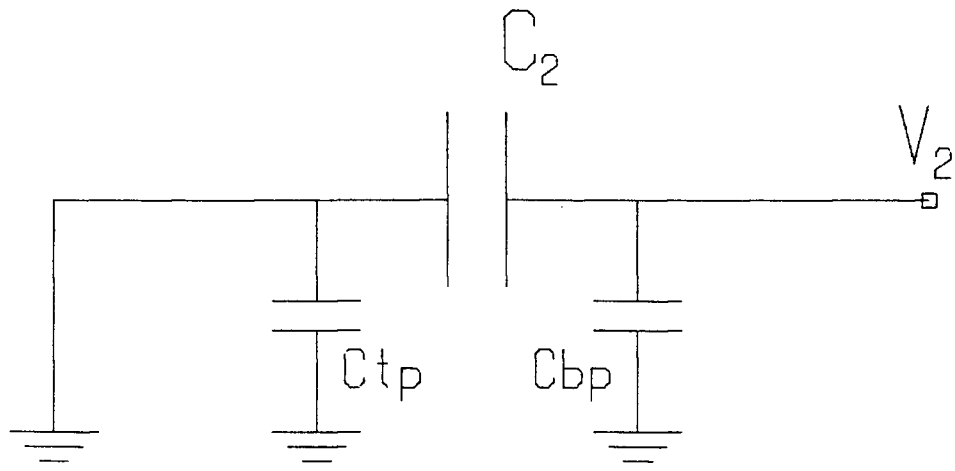


Fig. 3.12: Parallel switching during ϕ_2

It is important to note that all stray insensitive switched capacitors must be connected from a voltage source to a virtual ground. The same applies to the integrating and coupling capacitors.

3.4 MONOLITHIC FILTER ACCURACY

The time constant τ of the RC integrator of Fig. 3.5 is

$$\tau = RC_1 \tag{3.24}$$

with differential

$$d\tau = RdC_1 + C_1dR \tag{3.25}$$

Interpreting $d\tau/\tau$ as the relative error in τ gives

$$\frac{d\tau}{\tau} = \frac{dR}{R} + \frac{dC_1}{C_1} \tag{3.26}$$

In a typical MOS technology, dC_1/C_1 is accurate to a few percent and dR/R is accurate to ±50%. This is not adequate for filter implementation.

The switched capacitor integrators of Fig. 3.8 and Fig. 3.9 have a time constant τ of

$$\tau = \frac{TC_1}{C_2} \tag{3.27}$$

giving

$$\frac{d\tau}{\tau} = \frac{dT}{T} + \frac{dC_1}{C_1} - \frac{dC_2}{C_2} \tag{3.28}$$

An accurate clock is assumed, giving $dT/T \simeq 0$. Then

$$\frac{d\tau}{\tau} = \frac{dC_1}{C_1} - \frac{dC_2}{C_2} \tag{3.29}$$

For two adjacent capacitors with the same technology, this relative error has a value less than 0.1%. Absolute values are very hard to achieve, but errors affect both capacitors the same way maintaining the ratio. The linearity and temperature coefficients are also well behaved in this ratio.

3.5 FILTER DESIGN

A better way of transforming the filter response from the s to z plane is through the bilinear transform. Much work was spent in the early eighties on techniques to transform a passive LC prototype filter to a suitable switched capacitor implementation [3]-[9]. The desired filter must use stray insensitive switched capacitors and should be derived through the bilinear transform. Such techniques were developed by Lee [7], and Datar and Sedra [8]. These techniques replace the series and shunt branches of the LC prototype with integrators. The software developed in this thesis was based on these techniques as well as those of Martin [6], and of the AROMA program [9], which uses biquadratic sections consisting of two integrators.

4. THE CIRCUIT EXTRACTOR (CIRCE)

4.1 INTRODUCTION

The circuit extractor program (CIRCE), converts a SWITCAP filter file (here called filter), into a filter description more suitable for direct layout. It groups the filter elements into integrators, each consisting of stray insensitive switched capacitors, coupling capacitors, an integrating capacitor, and an operational amplifier. The integrators as well as the integrator elements are then sorted from the filter input to its output to simplify the layout.

In addition to the SWITCAP file, CIRCE requires a file containing the filter input stage elements (filter.in). The software generates two output files: one contains the node ordering information (filter.nd), and the other contains the final filter description (filter.ct). The software can currently process the input stages of Lee, and Datar and Sedra. These are shown in Fig. 4.1 and Fig. 4.2. There is no input stage for filters designed by AROMA, or by the techniques of Martin.

The input stage of the filter is the set of circuit elements feeding the input voltage to the first integrator's operational amplifier. The rest of the filter must be comprised of integrators (Fig. 4.3) possessing any number of coupling and switched capacitors, that obey the following rules:

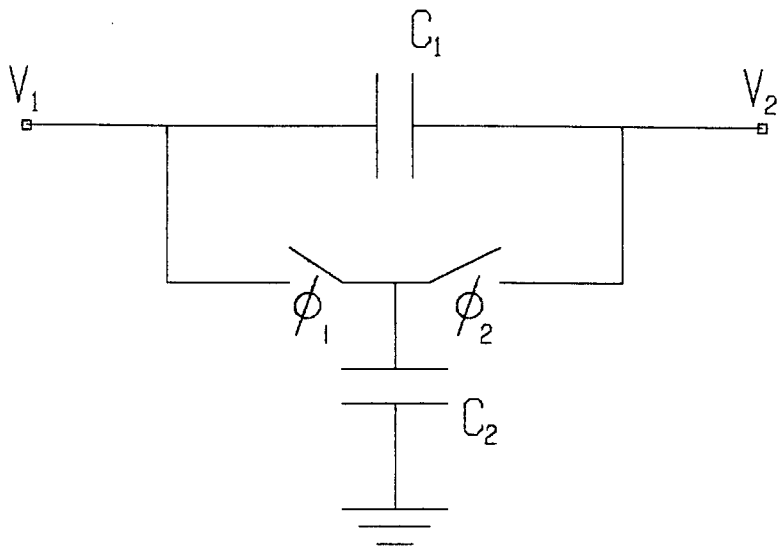


Fig. 4.1: Input stage (Lee)

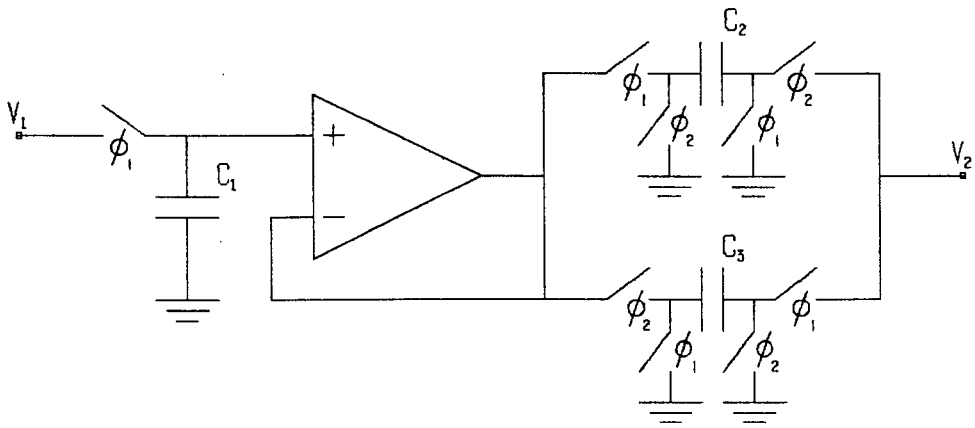


Fig. 4.2: Input stage (Datar & Sedra)

1. Only stray insensitive switched capacitors, coupling capacitors, and operational amplifier elements are allowed.
2. The noninverting input of each operational amplifier is always grounded.
3. Each operational amplifier must have one integrating capacitor.
4. Each operational amplifier feeds at least one switched capacitor.
5. The only distinct nodes in the filter are the operational amplifier noninverting input and output nodes.
6. Each operational amplifier is fed charge on only one of the two clocks.

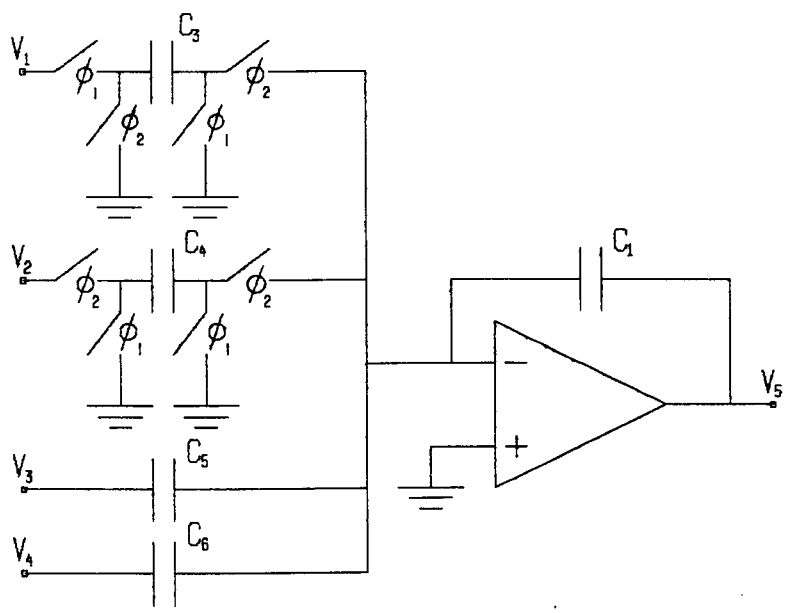


Fig. 4.3: Integrator model

7. The input stage connects to one operational amplifier noninverting input.

The design techniques of Refs. [6], [7], [8], and [9], obey these restrictions.

The CIRCE program consists of seven main function calls, and twenty-nine functions in all. A flowchart of CIRCE is shown in Fig. 4.4, and the main function calls are further described in this chapter. A listing is provided in Appendix C.

4.2 BREAKING DOWN THE INPUT FILE

4.2.1 SIMPLIFYING THE FILTER FILE (REDUCE)

The SWITCAP filter file is first simplified by removing all non-essential data in the file. The filter input and output nodes are noted here.

The filter elements are all in the SUBCKT and CIRCUIT sections of the file, so everything else is removed. The filter input node is taken as the first node of the voltage source statement, and the filter output node is taken as the first node of the PRINT or PLOT statement(s) in the ANALYZE block. The CIRCUIT and SUBCKT blocks are copied from the input file as follows:

1. Capitalize all letters.
2. Remove comments and blank lines.
3. One statement per line.
4. Left-adjust all lines.

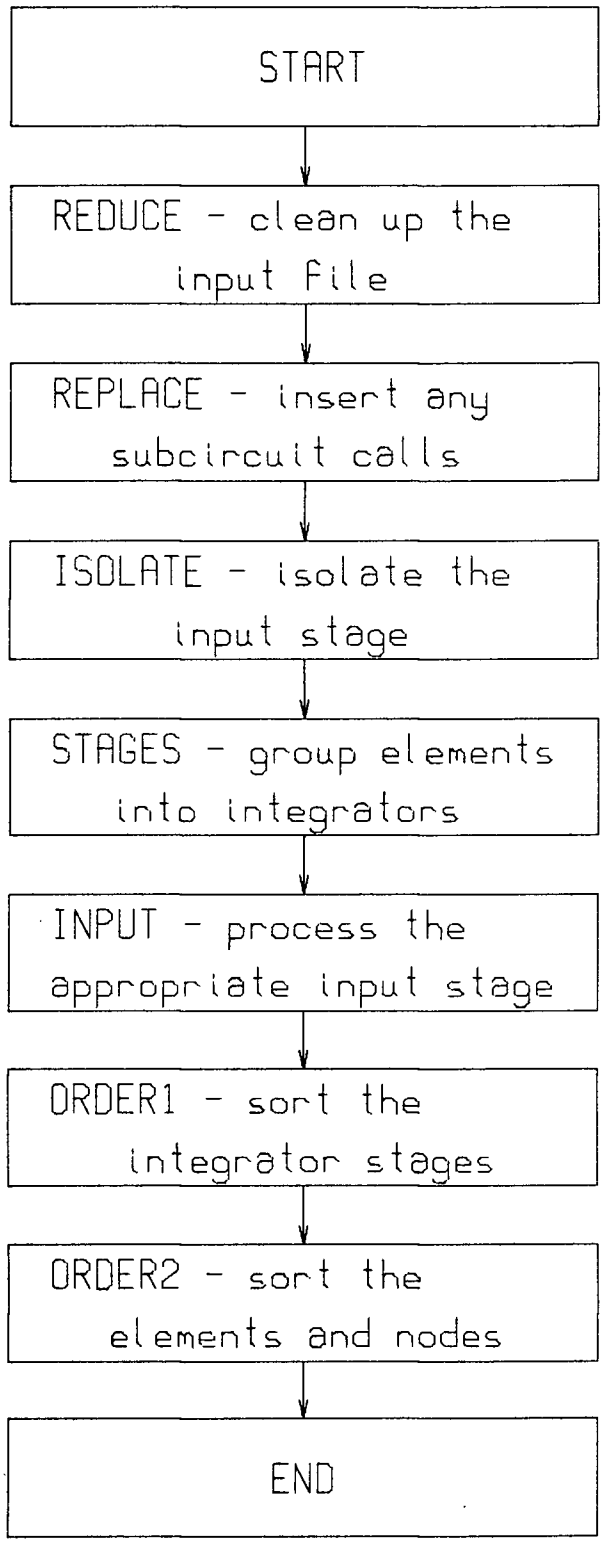


Fig. 4.4: CIRCE flowchart

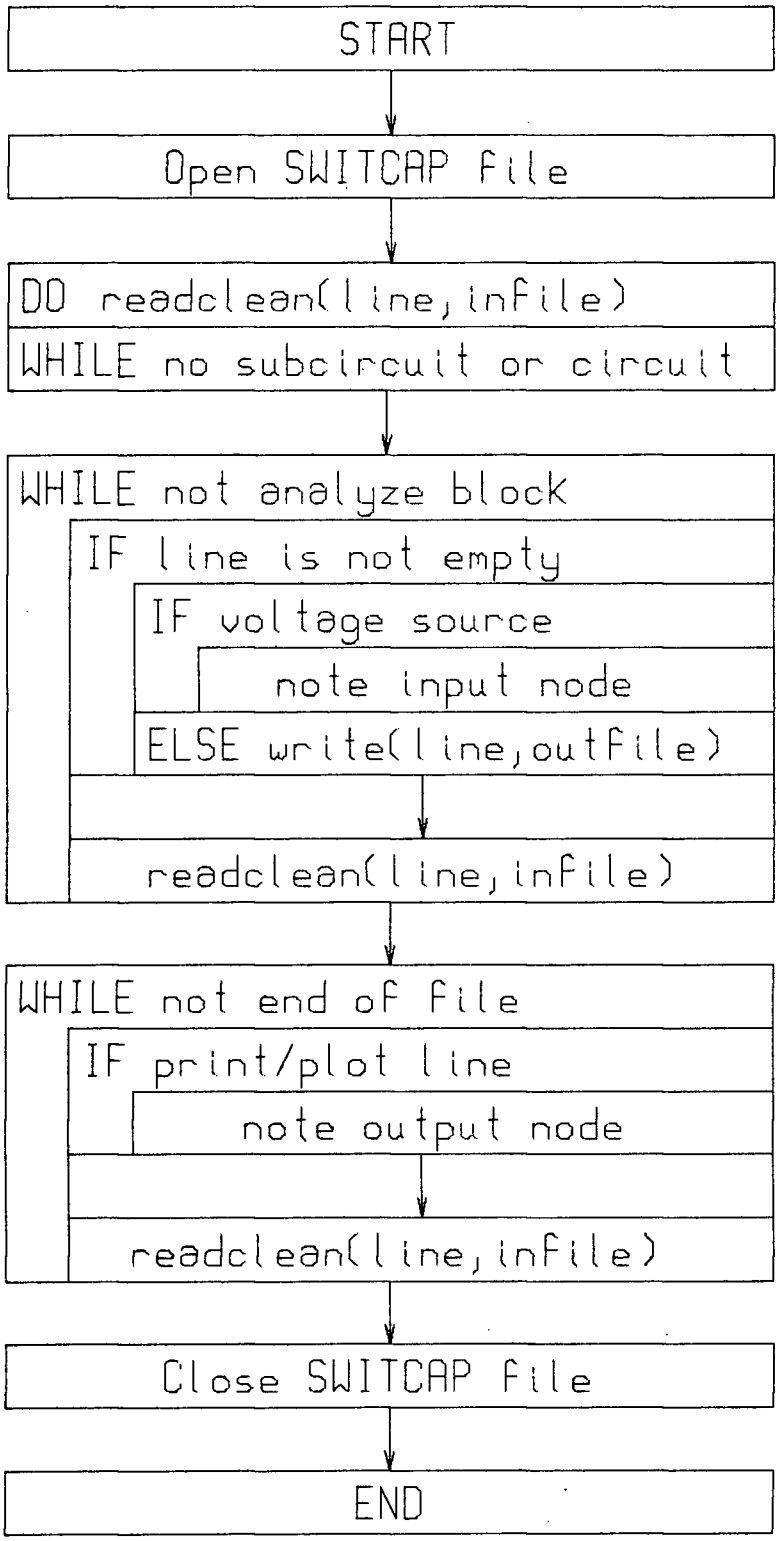


Fig. 4.5: REDUCE function flowchart

A flowchart of the REDUCE function is shown in Fig. 4.5. A sample SWITCAP filter file (filter) is shown in Fig. 4.6, and the corresponding file after processing is shown in Fig. 4.7. The filter file contains a switched capacitor filter design by Datar and Sedra, with random capacitor values.

4.2.2 INSERTING THE SUBCIRCUITS (REPLACE)

The subcircuit blocks are now inserted into the circuit. The file is scanned and if a SUBCKT block is found, it is removed from the file. The rest of the file is then scanned for any calls of that subcircuit. If any are found, the call is replaced by the subcircuit elements with the appropriate parameters inserted. All subcircuit calls start with a capital X in SWITCAP files. This process is repeated until there are no more SUBCKT blocks in the file.

The parameter replacement process works as follows. The parameters of the SUBCKT block header, as well as any other nodes existing in the SUBCKT block are tabulated. The parameters of the call are then tabulated next to the first set and any extra nodes are inserted to complete the second list. Node 0 is always ground. The subcircuit elements are then written to the file with the parameter substitutions made between the two lists. A sample table is shown in Table 4.1 for the first subcircuit call of Fig. 4.7.

An entry is made only once in the first column of the table, and any extra nodes are of the format !AA which are

```
OPTIONS ; GRID ; END ;
TITLE: Switched-Capacitor Filter (Datar & Sedra) ;

TIMING ;  /* Timing Section */
   PERIOD 1e-6 ;
   CLOCK C 1 (0 3/8) ; END ;

/* switched-capacitor sub-circuit */
SUBCKT (1 4) swc (K:phi1 K:phi2 K:phi3 K:phi4 P:cap1) ;
   S1 (1 2) phi1 ;
   S2 (2 0) phi2 ; S4 (3 0) phi4 ;
   C1 (2 3) cap1 ; S3 (3 4) phi3 ;
   END ;

  CIRCUIT ;  /* network section */
   /* switched-capacitors */
   Xc (12 10) swc (#C C #C  C  3.00) ;
   Xd (12 20) swc (#C C  C #C  4.00) ;
   Xe (22 10) swc (#C C #C  C  6.00) ;
   Xf (42 10) swc (#C C #C  C 12.00) ;
   Xg (30 32) swc (#C C #C  C  8.00) ;
   Xh (40 32) swc ( C #C #C C  9.00) ;
   Xi (42 30) swc (#C C #C  C 11.00) ;
   Xj (22 30) swc (#C C #C  C 13.00) ;

   /* capacitors */
   C2 (10 12) 2.00 ; C5 (22 20) 5.00 ;
   C14 (10 32) 14.00 ; C10 (40 42) 10.00 ;
   C15 (12 30) 15.00 ; C7 (30 32) 7.00 ;

   /* input section */
   C0 (2 0) 1.00 ; C1 (52 53) 1.00 ;
   C1 (54 55) 1.00 ; S (1 2) #C ;
   S ( 3 52)  C ; S (52 0) #C ;
   S (53 10)  C ; S (53 0) #C ;
   S ( 3 54) #C ; S (54 0)  C ;
   S (55 10) #C ; S (55 0)  C ;

   /* op-amps */
   E5 (42 0 0 40) 5000 ; E1 (3 0 2 3) 5000 ;
   E3 (22 0 0 20) 5000 ; E4 (32 0 0 30) 5000 ;
   E2 (12 0 0 10) 5000 ;
   V1 (1 0) ; /* input voltage node */  END ;

ANALYZE SSS ;  /* Analysis Section */
   INFREQ 0.01 100000 LOG 15 ;
   SET V1 AC 1.0 0.0 ;      /* amplitude and phase */
   PRINT VR(32) VI(32) ;    /* real and imaginary parts */
   END ;
END ;
```

Fig. 4.6: SWITCAP filter description (filter)

```
SUBCKT (1 4) SWC (K:PHI1 K:PHI2 K:PHI3 K:PHI4 P:CAP1) ;
S1 (1 2) PHI1 ;
S2 (2 0) PHI2 ;
S4 (3 0) PHI4 ;
C1 (2 3) CAP1 ;
S3 (3 4) PHI3 ;
END ;
CIRCUIT ;
XC (12 10) SWC (#C C #C  C  3.00) ;
XD (12 20) SWC (#C C  C #C  4.00) ;
XE (22 10) SWC (#C C #C  C  6.00) ;
XF (42 10) SWC (#C C #C  C 12.00) ;
XG (30 32) SWC (#C C #C  C  8.00) ;
XH (40 32) SWC ( C #C #C C  9.00) ;
XI (42 30) SWC (#C C #C  C 11.00) ;
XJ (22 30) SWC (#C C #C  C 13.00) ;
C2 (10 12) 2.00 ;
C5 (22 20) 5.00 ;
C14 (10 32) 14.00 ;
C10 (40 42) 10.00 ;
C15 (12 30) 15.00 ;
C7 (30 32) 7.00 ;
C0 (2 0) 1.00 ;
C1 (52 53) 1.00 ;
C1 (54 55) 1.00 ;
S (1 2) #C ;
S ( 3 52)  C ;
S (52 0) #C ;
S (53 10)  C ;
S (53 0) #C ;
S ( 3 54) #C ;
S (54 0)  C ;
S (55 10) #C ;
S (55 0)  C ;
E5 (42 0 0 40) 5000 ;
E1 (3 0 2 3) 5000 ;
E3 (22 0 0 20) 5000 ;
E4 (32 0 0 30) 5000 ;
E2 (12 0 0 10) 5000 ;
END ;
```

Fig. 4.7: Output from REDUCE function

Table 4.1: Parameter substitutions

SUBCKT Parameters	Call Parameters
1	12
4	10
PHI1	#C
PHI2	C
PHI3	#C
PHI4	C
CAP1	3.00
2	!AA
0	0
3	!AB

not allowed in SWITCAP files. The flowchart of the REPLACE function is shown in Fig. 4.8, and the resulting output file is shown in Fig. 4.9.

4.3 GATHERING THE INTEGRATORS

4.3.1 REMOVING THE INPUT STAGE (ISOLATE)

The ISOLATE function removes all input stage elements from the filter description file. All elements in the filter.in file are removed from the filter file. The flowchart of the ISOLATE function is shown in Fig. 4.10.

The data in the filter.in file (Fig. 4.11) is first processed in the same way as the filter file was. Any lines

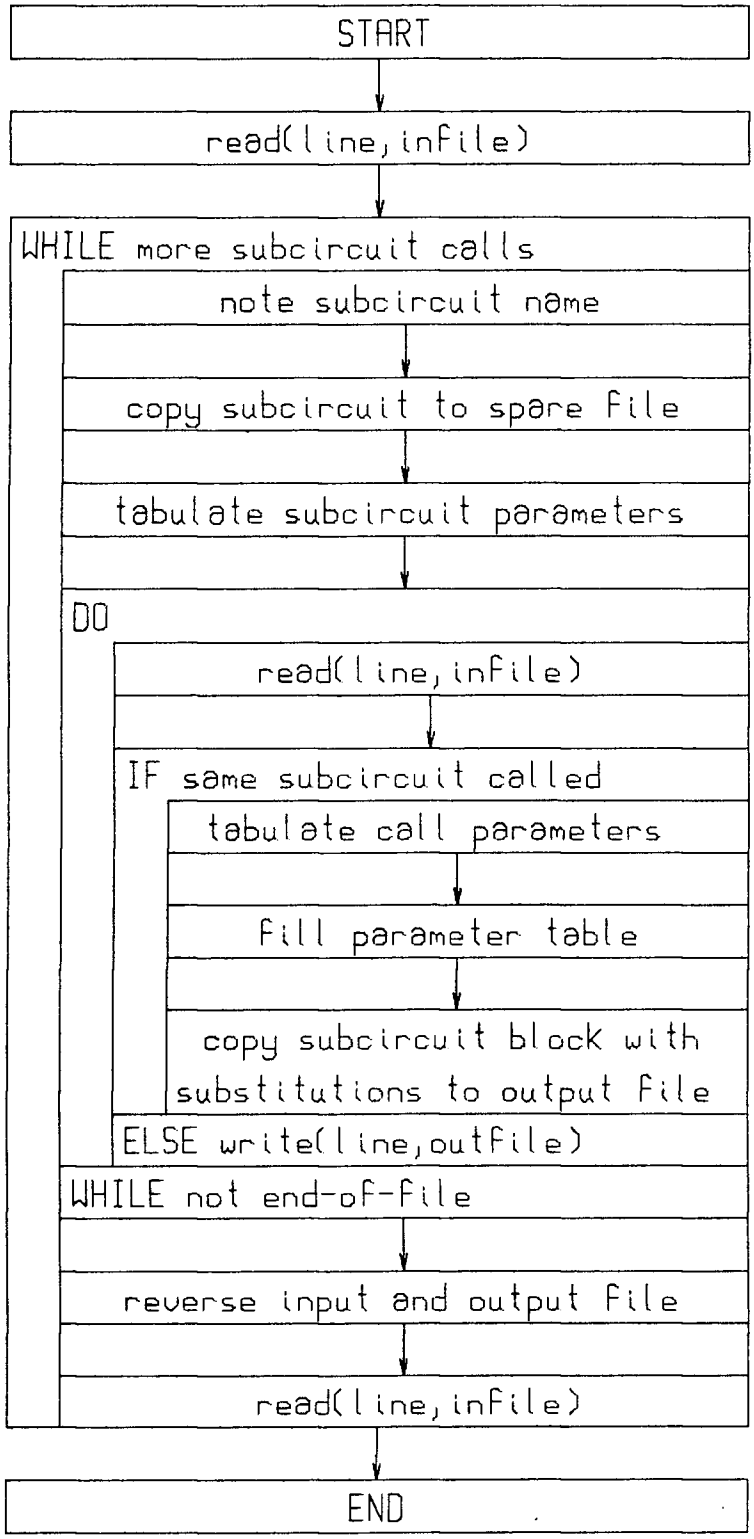


Fig. 4.8: REPLACE function flowchart

```
CIRCUIT ;
S1 (12 !AA) #C ;
S2 (!AA 0) C ;
S4 (!AB 0) C ;
C1 (!AA !AB) 3.00 ;
S3 (!AB 10) #C ;
S1 (12 !AC) #C ;
S2 (!AC 0) C ;
S4 (!AD 0) #C ;
C1 (!AC !AD) 4.00 ;
S3 (!AD 20) C ;
S1 (22 !AE) #C ;
S2 (!AE 0) C ;
S4 (!AF 0) C ;
C1 (!AE !AF) 6.00 ;
S3 (!AF 10) #C ;
S1 (42 !AG) #C ;
S2 (!AG 0) C ;
S4 (!AH 0) C ;
C1 (!AG !AH) 12.00 ;
S3 (!AH 10) #C ;
S1 (30 !AI) #C ;
S2 (!AI 0) C ;
S4 (!AJ 0) C ;
C1 (!AI !AJ) 8.00 ;
S3 (!AJ 32) #C ;
S1 (40 !AK) C ;
S2 (!AK 0) #C ;
S4 (!AL 0) C ;
C1 (!AK !AL) 9.00 ;
S3 (!AL 32) #C ;
S1 (42 !AM) #C ;
S2 (!AM 0) C ;
S4 (!AN 0) C ;
C1 (!AM !AN) 11.00 ;
S3 (!AN 30) #C ;
S1 (22 !AO) #C ;
S2 (!AO 0) C ;
S4 (!AP 0) C ;
C1 (!AO !AP) 13.00 ;
S3 (!AP 30) #C ;
C2 (10 12) 2.00 ;
C5 (22 20) 5.00 ;
C14 (10 32) 14.00 ;
C10 (40 42) 10.00 ;
C15 (12 30) 15.00 ;
C7 (30 32) 7.00 ;
C0 (2 0) 1.00 ;
C1 (52 53) 1.00 ;
C1 (54 55) 1.00 ;
```

Fig. 4.9: Output from REPLACE function

```
S (1 2) #C ;
S ( 3 52)  C ;
S (52 0) #C ;
S (53 10)  C ;
S (53 0) #C ;
S ( 3 54) #C ;
S (54 0)  C ;
S (55 10) #C ;
S (55 0)  C ;
E5 (42 0 0 40) 5000 ;
E1 (3 0 2 3) 5000 ;
E3 (22 0 0 20) 5000 ;
E4 (32 0 0 30) 5000 ;
E2 (12 0 0 10) 5000 ;
END ;
```

Fig. 4.9: Output from REPLACE function (cont'd)

in the filter file that are in the filter.in file are put into one output file (Fig. 4.12) while the remaining lines are put into another output file (Fig. 4.13). This isolates the input stage elements from the integrator stages of the filter.

4.3.2 FORMING THE INTEGRATORS (STAGES)

The circuit elements are now gathered into integrators with the elements sharing a common inverting operational amplifier node (Fig. 4.4). Each integrator consists of one operational amplifier, one integrating capacitor, at least one switched capacitor, and any coupling capacitors. This grouping is done for each operational amplifier as it is found in the filter file. The flowchart of the STAGES function is shown in Fig. 4.14.

Each element is also recoded here for convenience. The operational amplifiers are called OA, the stray insensitive switched capacitors are called SC, the integrating

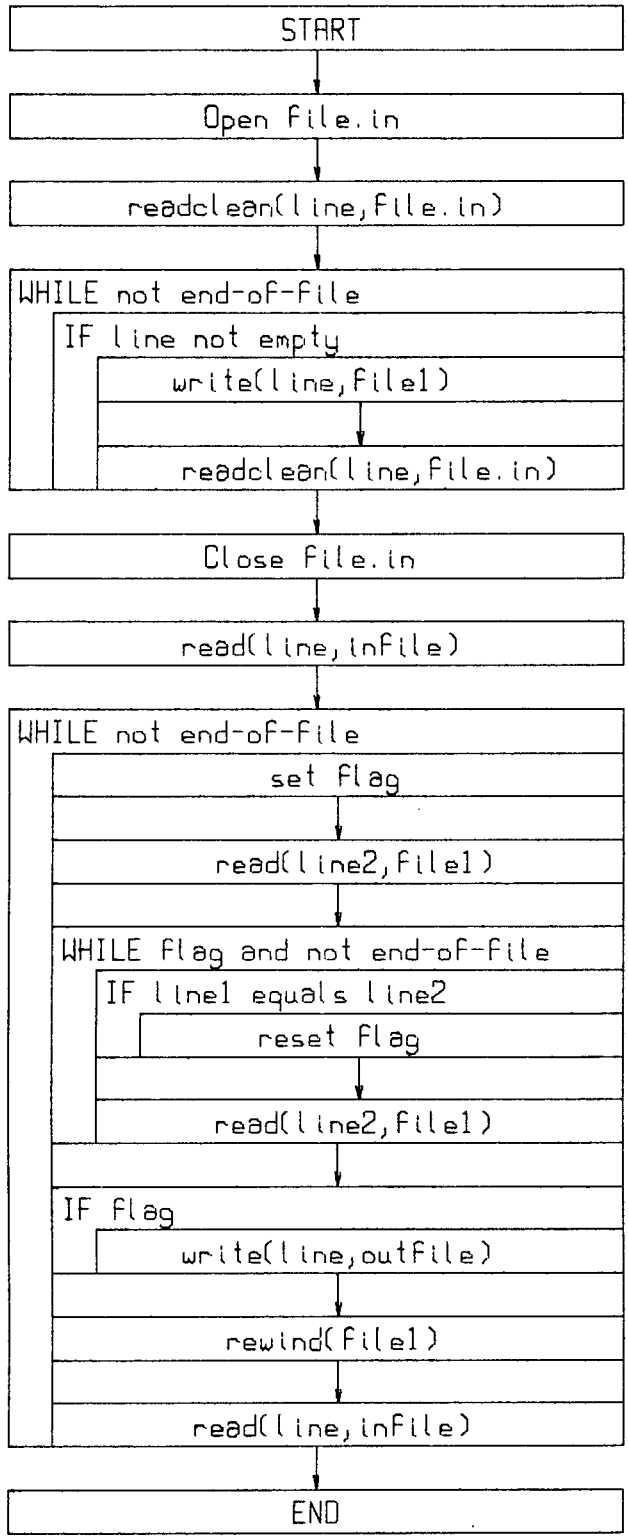


Fig. 4.10: ISOLATE function flowchart

```
/* input section */
C0 (2 0) 1.00 ; C1 (52 53) 1.00 ;
C1 (54 55) 1.00 ; S (1 2) #C ;
S ( 3 52)  C ; S (52 0) #C ;
S (53 10)  C ; S (53 0) #C ;
S ( 3 54) #C ; S (54 0)  C ;
S (55 10) #C ; S (55 0)  C ;
E1 (3 0 2 3) 5000 ;
```

Fig. 4.11: Input section (filter.in)

```
C0 (2 0) 1.00 ;
C1 (52 53) 1.00 ;
C1 (54 55) 1.00 ;
S (1 2) #C ;
S ( 3 52)  C ;
S (52 0) #C ;
S (53 10)  C ;
S (53 0) #C ;
S ( 3 54) #C ;
S (54 0)  C ;
S (55 10) #C ;
S (55 0)  C ;
E1 (3 0 2 3) 5000 ;
```

Fig. 4.12: Isolated input section

```
S1 (12 !AA) #C ;
S2 (!AA 0) C ;
S4 (!AB 0) C ;
C1 (!AA !AB) 3.00 ;
S3 (!AB 10) #C ;
S1 (12 !AC) #C ;
S2 (!AC 0) C ;
S4 (!AD 0) #C ;
C1 (!AC !AD) 4.00 ;
S3 (!AD 20) C ;
S1 (22 !AE) #C ;
S2 (!AE 0) C ;
S4 (!AF 0) C ;
C1 (!AE !AF) 6.00 ;
S3 (!AF 10) #C ;
S1 (42 !AG) #C ;
S2 (!AG 0) C ;
S4 (!AH 0) C ;
C1 (!AG !AH) 12.00 ;
S3 (!AH 10) #C ;
S1 (30 !AI) #C ;
S2 (!AI 0) C ;
S4 (!AJ 0) C ;
C1 (!AI !AJ) 8.00 ;
S3 (!AJ 32) #C ;
S1 (40 !AK) C ;
S2 (!AK 0) #C ;
S4 (!AL 0) C ;
C1 (!AK !AL) 9.00 ;
S3 (!AL 32) #C ;
S1 (42 !AM) #C ;
S2 (!AM 0) C ;
S4 (!AN 0) C ;
C1 (!AM !AN) 11.00 ;
S3 (!AN 30) #C ;
S1 (22 !AO) #C ;
S2 (!AO 0) C ;
S4 (!AP 0) C ;
C1 (!AO !AP) 13.00 ;
S3 (!AP 30) #C ;
C2 (10 12) 2.00 ;
C5 (22 20) 5.00 ;
C14 (10 32) 14.00 ;
C10 (40 42) 10.00 ;
C15 (12 30) 15.00 ;
C7 (30 32) 7.00 ;
E5 (42 0 0 40) 5000 ;
E3 (22 0 0 20) 5000 ;
E4 (32 0 0 30) 5000 ;
E2 (12 0 0 10) 5000 ;
```

Fig. 4.13: Remainder of filter description

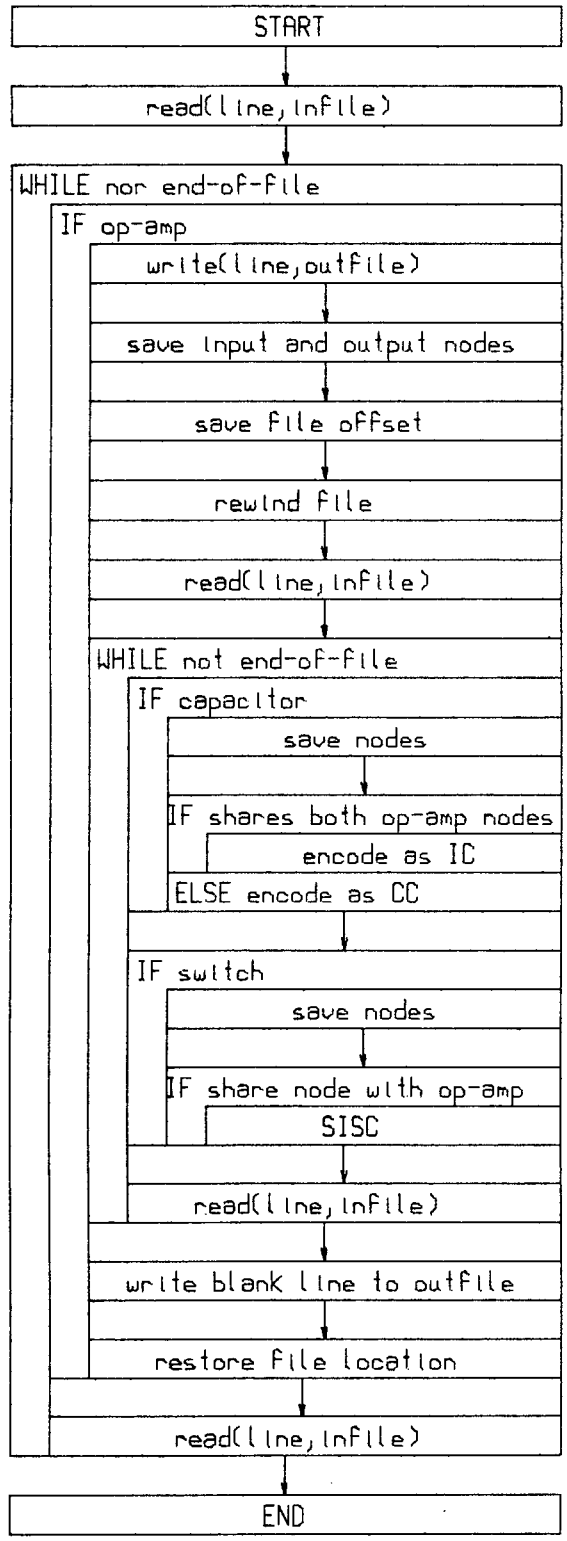


Fig. 4.14: STAGES function flowchart

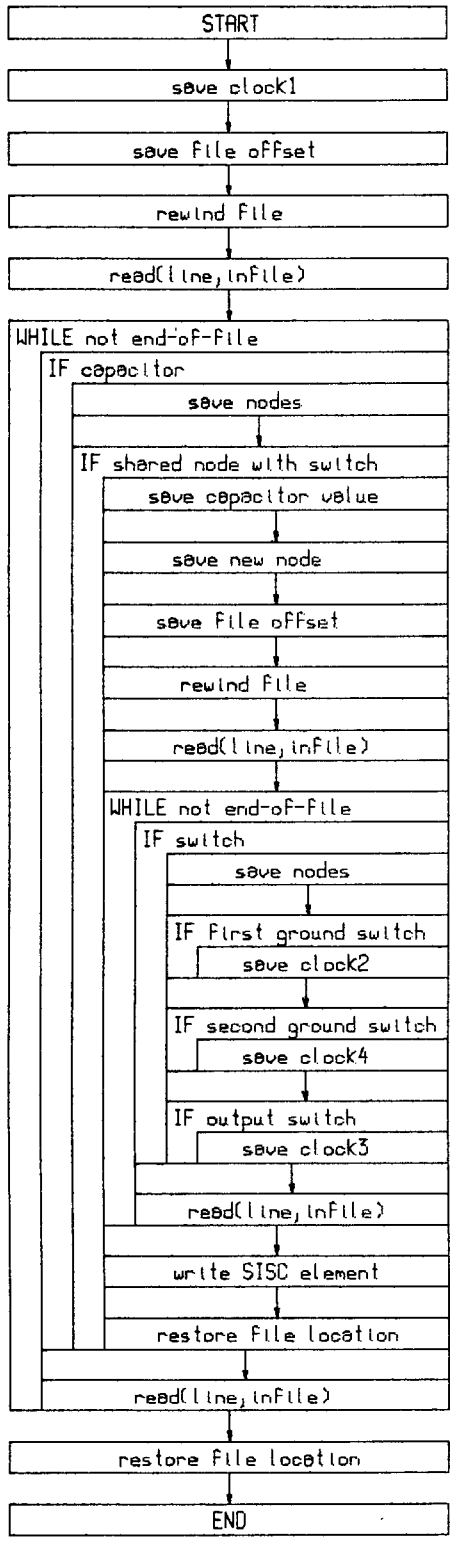


Fig. 4.15: SISC function flowchart

capacitors are called IC, and the coupling capacitors are called CC. All capacitors have a 'C' as the second letter of their labels.

The inverting input node of each operational amplifier is used to gather up the integrator elements. A capacitor sharing both input and output node with the operational amplifier is an integrating capacitor, while a capacitor that only shares the input node is a coupling capacitor. If a switch is found connected to the input node, it must belong to a switched capacitor element. The SISC function (Fig. 4.15) gathers up the remaining elements of the switched capacitor. Since switch-sharing may be used, it is neccessary to check for more than one switched capacitor for each switch (Fig. 4.16). The stray insensitive switched capacitor element (Fig. 4.17) is then recoded as follows:

SC (nodeA nodeB) C ϕ_1 ϕ_2 ϕ_3 ϕ_4 ;

where nodeA connects to the operational amplifier inverting input node. This node is the reference point for the clocking scheme. The complementary clock must be designated by the '#' character. The output from the STAGES function is shown in Fig. 4.18.

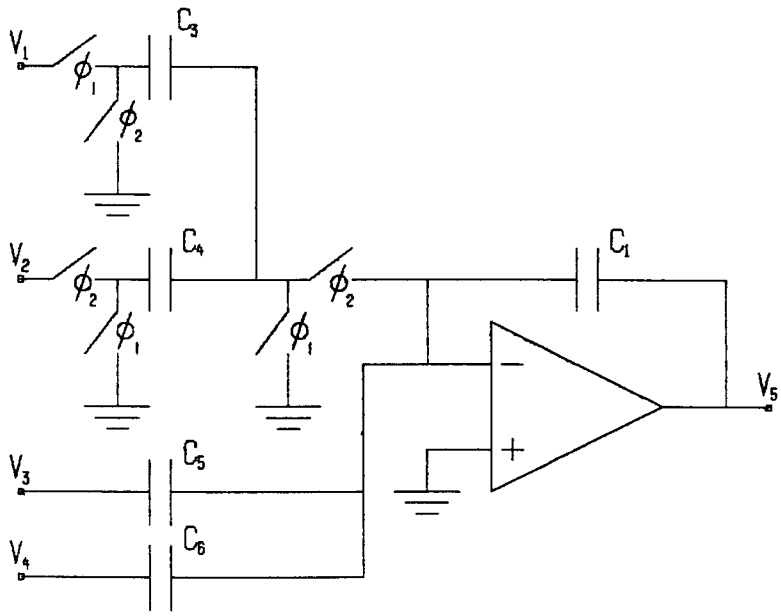


Fig. 4.16: Integrator with switch-sharing

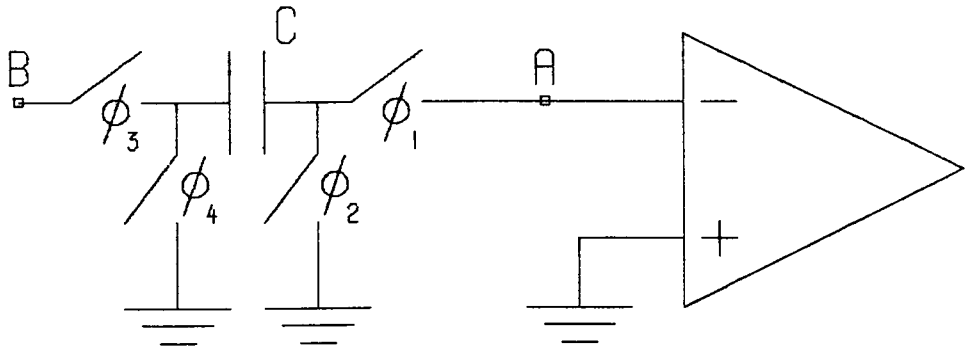


Fig. 4.17: Stray insensitive SC element

4.4 SORTING THE FILTER

4.4.1 PROCESSING THE INPUT STAGE (INPUT)

The input stage of the filter is now recoded and the elements sorted into a suitable order. The node by which the input stage connects to the filter is saved here. There is one INPUT function for each of the design techniques.

The elements of the input stages are switches (BS), capacitors (BC), stray insensitive switched capacitors (SC), and operational amplifiers (OA). Any operational amplifier is listed first, followed by any switches. Any capacitors are listed next, followed by any switched capacitors. The

```
OA (42 0 0 40) 5000 ;
SC (40 32) 9.00 C #C #C C ;
IC (40 42) 10.00 ;

OA (22 0 0 20) 5000 ;
SC (20 12) 4.00 C #C #C C ;
IC (22 20) 5.00 ;

OA (32 0 0 30) 5000 ;
SC (30 32) 8.00 #C C #C C ;
SC (30 42) 11.00 #C C #C C ;
SC (30 22) 13.00 #C C #C C ;
CC (12 30) 15.00 ;
IC (30 32) 7.00 ;

OA (12 0 0 10) 5000 ;
SC (10 12) 3.00 #C C #C C ;
SC (10 22) 6.00 #C C #C C ;
SC (10 42) 12.00 #C C #C C ;
IC (10 12) 2.00 ;
CC (10 32) 14.00 ;
```

Fig. 4.18: Output from STAGES function

recoded and sorted output is written to the filter.ct file (Fig. 4.19). The different input functions are described further below.

4.4.1.1 INPUT1 (Lee)

This input stage (Fig. 4.1) consists of four elements: two capacitors, and two switches. The switches are written first, followed by the capacitor connected to the input node, followed by the grounded capacitor. The input node is listed first for the first capacitor, and the filter connection node is the other node of this element.

4.4.1.2 INPUT2 (Martin)

The STAGES function can process all elements of this design technique, hence the filter.in file is empty. The filter connection node is the first node of the SC element connected to the filter input node.

```
OA (3 0 2 3) 5000 ;
BC (2 0) 1.00 ;
BS (1 2) #C ;
SC (3 10) 1.00 C #C C #C ;
SC (10 3) 1.00 #C C #C C ;
```

Fig. 4.19: Final input stage description

4.4.1.3 INPUT3 (Datar and Sedra)

This input stage consists of one operational amplifier, one switch, one capacitor, and two switched capacitors (Fig. 4.2). The operational amplifier is written first, followed by the switch and the grounded capacitor. One of the SC elements can be switch-shared with the first integrator, and it is written last. The other SC element is written next, followed by the above SC element. The SC elements are gathered up by the SISC function. The filter connection node is the second node of either SC element.

4.4.1.4 INPUT4 (AROMA)

The biquad stages used by the AROMA program are similar to the design technique of Martin. The filter.in file is again empty, as the STAGES function can handle all the elements. The filter connection node is obtained from SC element(s) connected to the filter input node. If only one SC exists connected to the filter input node, the connection node is its second node. If there is a choice of possible connection nodes, the node occurring only once is chosen.

4.4.2 SORTING THE INTEGRATORS (ORDER1)

The integrator stages are sorted into an order suitable for direct layout by the ORDER1 function. This sorting starts with the node provided by the appropriate INPUT function.

Many biquad filters as well as bilinear low-pass filters have a filter topology similar to that of Fig. 4.20. Bandpass filters are more complex as shown in Fig. 4.21 and the AROMA program generates biquad sections as shown in Fig. 4.22. Each integrator is represented by one box. Only SC elements are used as links in the sorting process. The sort proceeds by the SC elements fed by each operational amplifier, and once counted, an operational amplifer may not be counted again.

In all cases, the filter connection node provides the first integrator stage, and its ouput node is used to start the search path. It is assumed that no integrator can feed more than two unsorted integrators. If only one is found, the sort is simple and linear. If two new integrators are

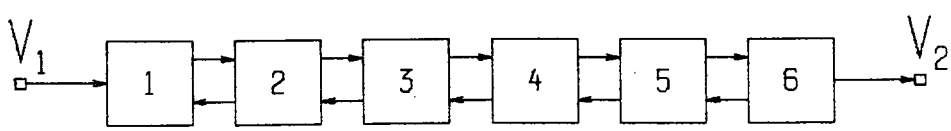


Fig. 4.20: Linear topology

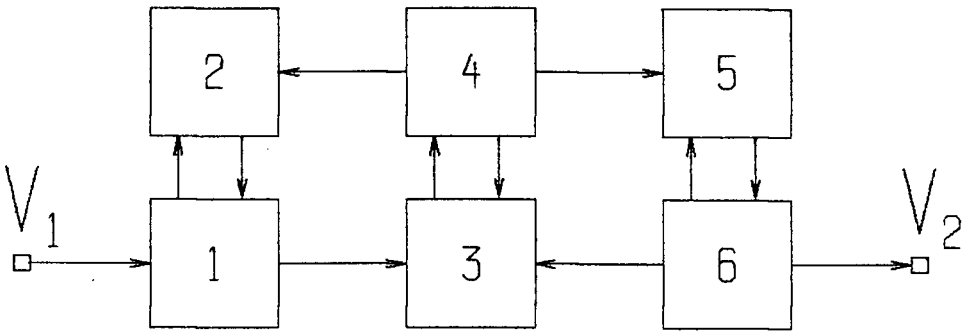


Fig. 4.21: Band-pass filter topology

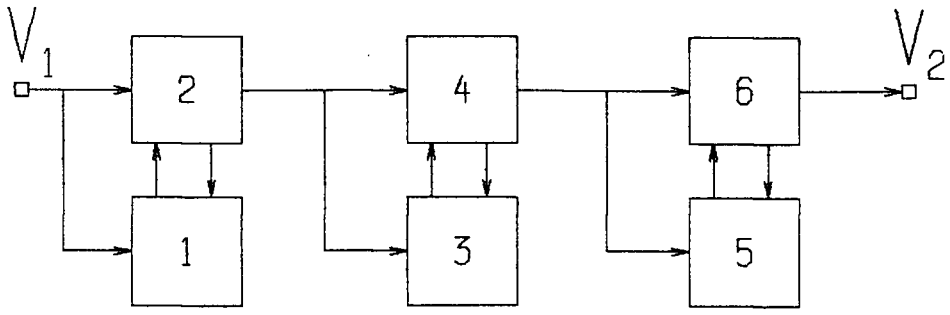


Fig. 4.22: AROMA filter topology

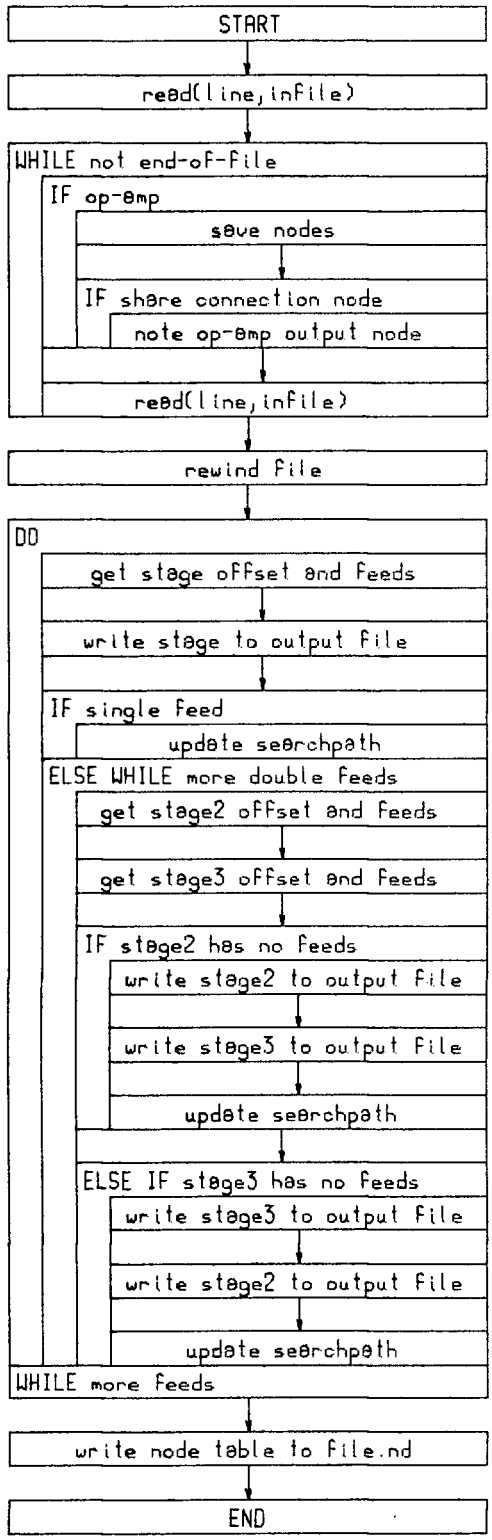


Fig. 4.23: ORDER1 function flowchart

found, they are in turn processed, and the one yielding no new paths is written first, followed by the other. The sorting continues until no new integrators are found. This sorting always puts the filter output stage last or next-to-last. The flowchart of the ORDER1 function is shown in Fig. 4.23, and the output from the function is shown in Fig. 4.24.

As each node pair is found by the software, it is saved for use by the ORDER2 function, and is also written to the filter.nd file. The filter input and output nodes are written first, followed by any nodes existing in the input stage, then the nodes of the operational amplifiers. The

```
OA (12 0 0 10) 5000 ;
SC (10 12) 3.00 #C C #C C ;
SC (10 22) 6.00 #C C #C C ;
SC (10 42) 12.00 #C C #C C ;
IC (10 12) 2.00 ;
CC (10 32) 14.00 ;

OA (22 0 0 20) 5000 ;
SC (20 12) 4.00 C #C #C C ;
IC (22 20) 5.00 ;

OA (32 0 0 30) 5000 ;
SC (30 32) 8.00 #C C #C C ;
SC (30 42) 11.00 #C C #C C ;
SC (30 22) 13.00 #C C #C C ;
CC (12 30) 15.00 ;
IC (30 32) 7.00 ;

OA (42 0 0 40) 5000 ;
SC (40 32) 9.00 C #C #C C ;
IC (40 42) 10.00 ;
```

Fig. 4.24: Output from ORDER1 function

inverting input node is written before the output node for each operational amplifier. The filter.nd file is shown in Fig. 4.25.

4.4.3 SORTING THE ELEMENTS AND NODES (ORDER2)

The elements and their nodes are now sorted for each integrator by the ORDER2 function. It uses the nodal ordering information saved by the ORDER1 function to do this. The ORDER2 flowchart is shown in Fig. 4.26.

Each element of an integrator is stored in an array and is written to the filter.ct file when the node not connected to the inverting input of the operating amplifier is found in the nodal ordering table. The nodal table is scanned from beginning to end causing elements feeding towards the filter

```
1
32

2
3

10
12
20
22
30
32
40
42
```

Fig. 4.25: CIRCE output file (filter.nd)

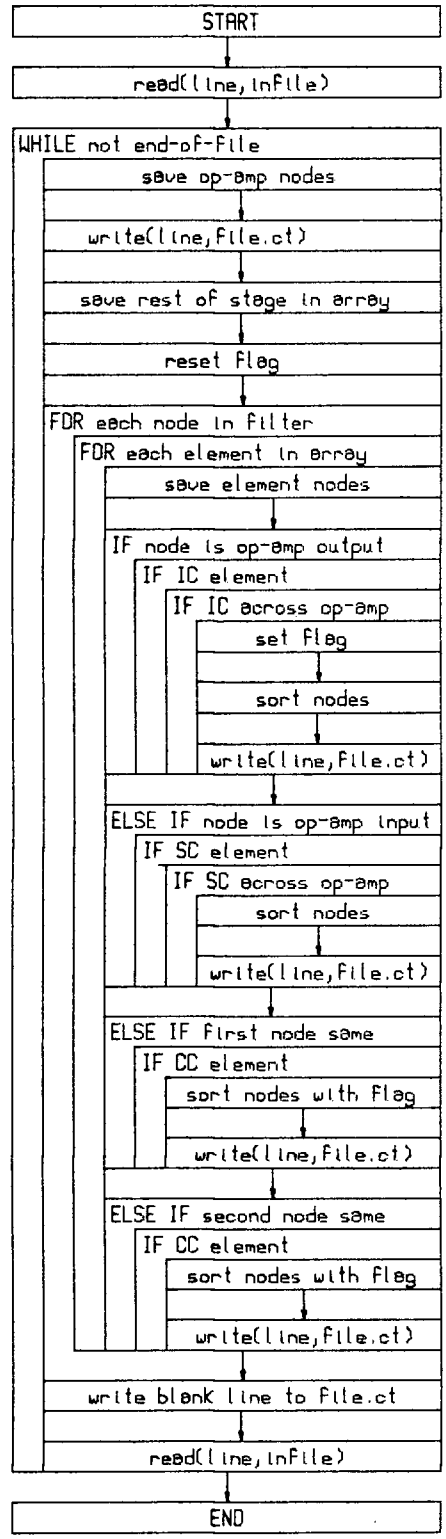


Fig. 4.26: ORDER2 function flowchart

```
OA (3 0 2 3) 5000 ;
BC (2 0) 1.00 ;
BS (1 2) #C ;
SC (3 10) 1.00 C #C C #C ;
SC (10 3) 1.00 #C C #C C ;

OA (12 0 0 10) 5000 ;
IC (10 12) 2.00 ;
SC (10 12) 3.00 #C C #C C ;
SC (10 22) 6.00 #C C #C C ;
CC (10 32) 14.00 ;
SC (10 42) 12.00 #C C #C C ;

OA (22 0 0 20) 5000 ;
SC (20 12) 4.00 C #C #C C ;
IC (20 22) 5.00 ;

OA (32 0 0 30) 5000 ;
CC (12 30) 15.00 ;
SC (30 22) 13.00 #C C #C C ;
IC (30 32) 7.00 ;
SC (30 32) 8.00 #C C #C C ;
SC (30 42) 11.00 #C C #C C ;

OA (42 0 0 40) 5000 ;
SC (40 32) 9.00 C #C #C C ;
IC (40 42) 10.00 ;
```

Fig. 4.27: CIRCE output file (filter.ct)

input to be written before elements feeding towards the filter output for each integrator.

The operational amplifier input node is placed first for the IC and the SC elements, while the nodes for the CC elements are sorted so that the first node feeds towards the filter input.

A SC element connected from the operational amplifier input to its output is placed after the IC element.

The filter.ct file is completed by the ORDER2 function (Fig. 4.27), with the first part written by the INPUT function. The filter.ct and the filter.nd files are the output from the CIRCE program. These files are used by the layout generator program in generating the filter layout.

5. THE LAYOUT GENERATOR (SISCL)

5.1 INTRODUCTION

The layout generator program (called SISCL, for Stray Insensitive Switched Capacitor Layout) generates an ISOCMOS layout of the switched capacitor filter in Caltech Intermediate Form (CIF). It uses the information in the filter.nd and the filter.ct files as well as a layout parameter file called filter.com. The operational amplifier CIF code is stored in a file called opamp.cif. The program can generate filter layouts from the four design techniques described in the previous chapter. There are thirteen main functions to the layout generator program and forty-one functions in total. The output file (called filter.cif) is written by three functions. One writes BOX commands, one writes WIRE commands and the last one writes cell calls. This makes it easier to convert the software into doing layouts in another format. The flowchart of the layout generator program is shown in Fig. 5.1. A listing of the software is provided in Appendix D.

5.2 THE OPERATIONAL AMPLIFIER

A standard cell operational amplifier is used, which must have certain general features (Fig. 5.2). The operational amplifier data is put into the filter.com file as shown in Fig. 5.3. The power lines must be in metal, and VDD must be on top. The inputs and output must emerge on top

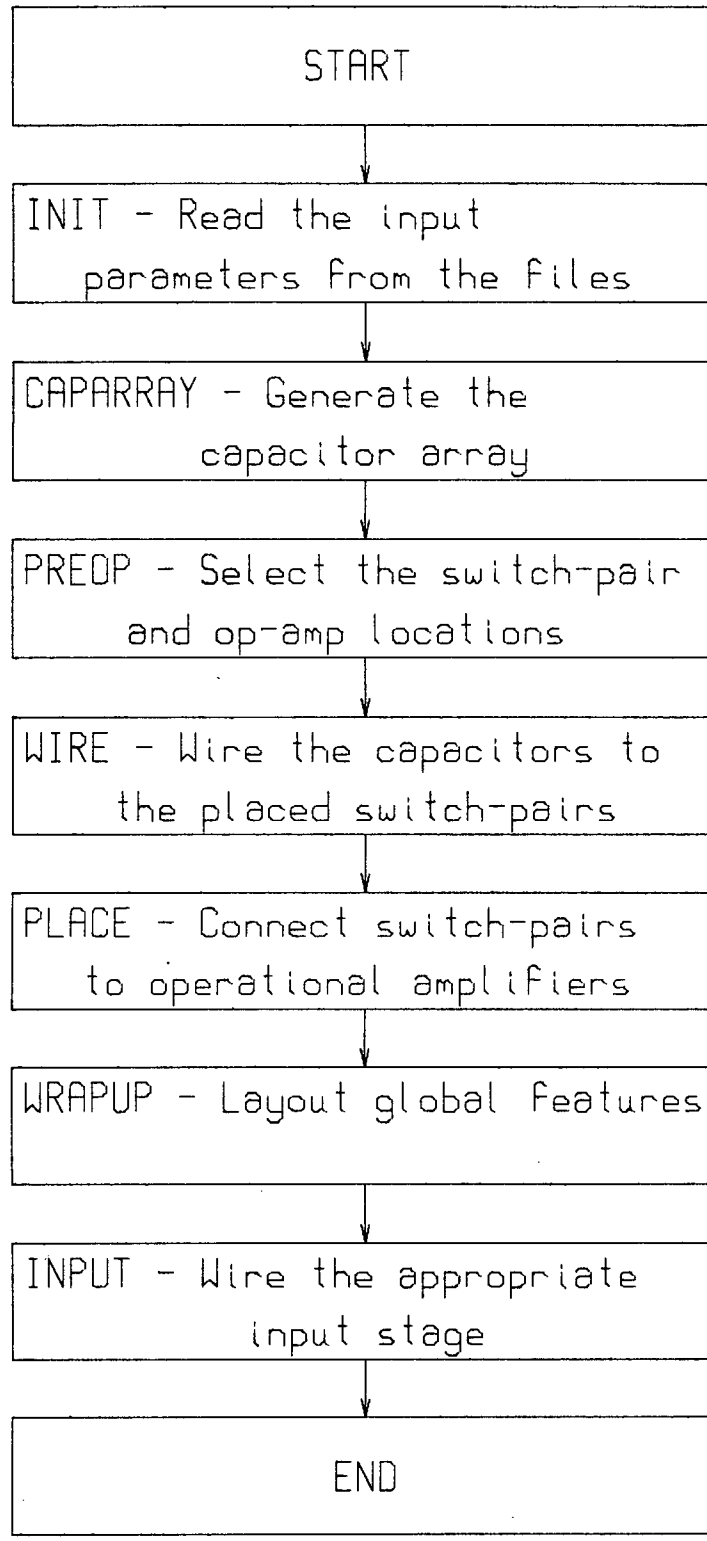


Fig. 5.1: SISCL flowchart

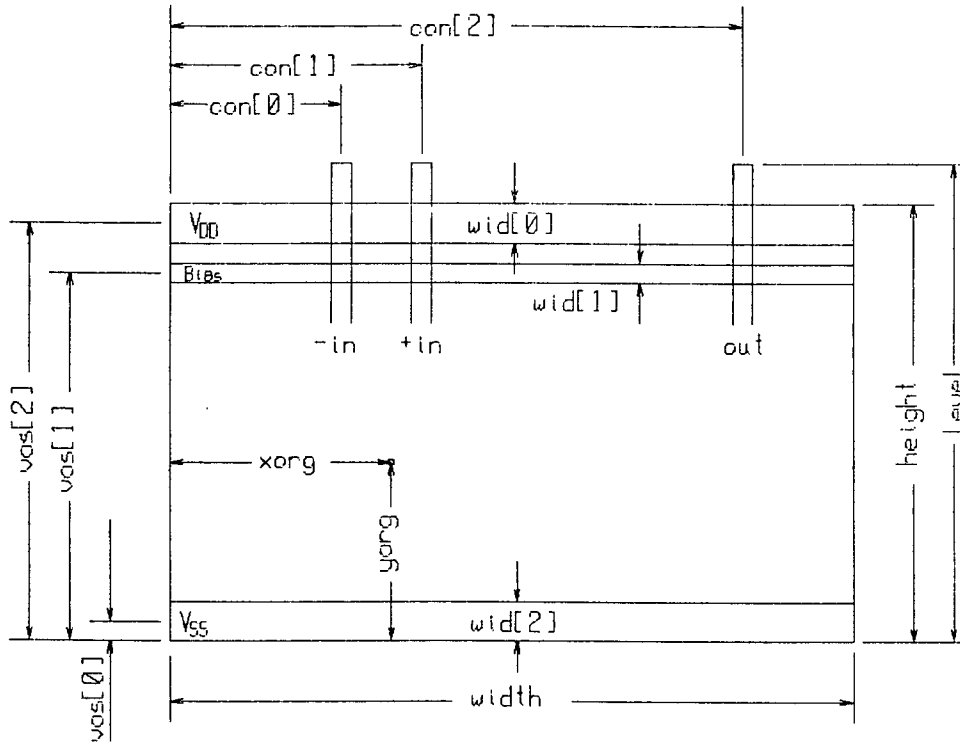


Fig. 5.2: Operational amplifier parameters

λ (µm)	arrayheight	arrayswitch
xorg (λ)	yorg (λ)	
height (λ)	width (λ)	level (λ)
con[0] (λ)	con[1] (λ)	con[2] (λ)
vos[0] (λ)	vos[1] (λ)	vos[2] (λ)
wid[0] (λ)	wid[1] (λ)	wid[2] (λ)

Fig. 5.3: Filter.com file format

in polysilicon1 of width 2λ. The inputs should be grouped together but it does not matter which one is to the left or right.

The origin of the operational amplifier (xorg,yorg), can be used to adjust the vertical placement of the cell, as well as the gaps on the left and right sides of the operational amplifier. All the operational amplifier parameters should be in units of λ. The operational amplifier used in this thesis is plotted in Fig. 5.4. The operational amplifier CIF code must be symbol #2. The filter layout itself is symbol #1.

The first line of the filter.com file consists of the scaling factor λ of the fabrication process. This is followed by the height of the capacitor array and the capacitor size at which the capacitor layout strategy changes. The remaining parameters describe the physical shape of the operational amplifier standard cell.

5.3 READING THE INPUT DATA (INIT)

The data stored in the filter.nd, filter.ct, and filter.com files is read by the INIT function. The node data is put into a character array called TABLE, which is scanned whenever connections have to be made to the nodes. The operational amplifier parameters are put into a data structure called OPAMP which is used for the operational amplifier placement and wiring. The capacitor and switch data is put into a data structure called CAPAC, which is

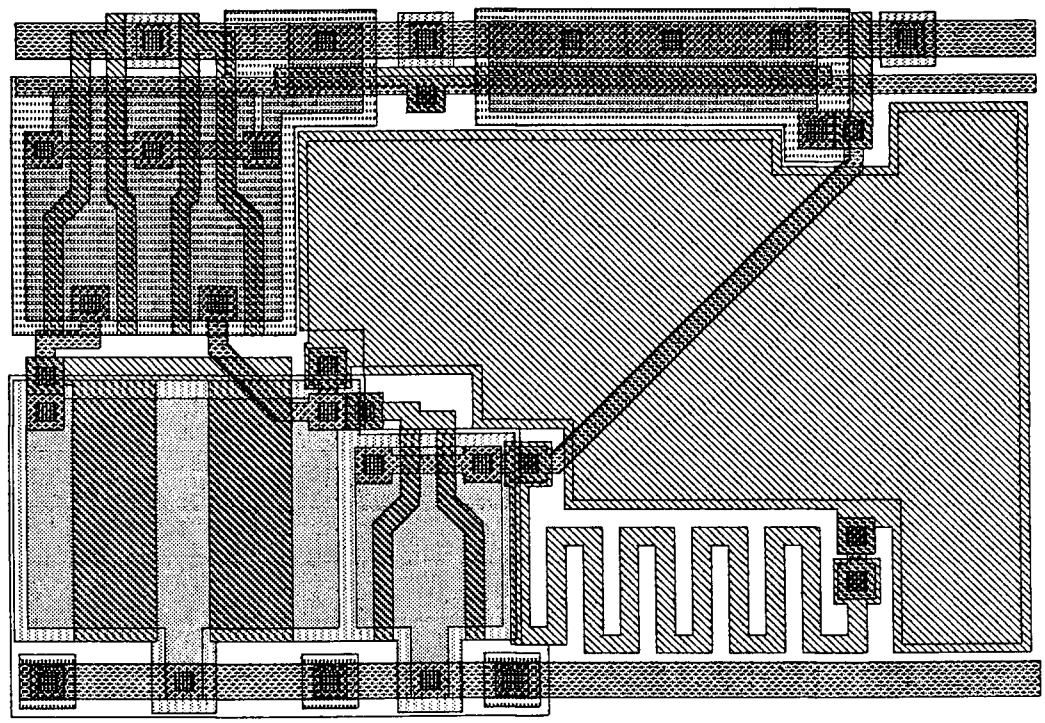


Fig. 5.4: Operational amplifier layout

used to generate the capacitor array, its wiring, and the switches.

Each capacitor has its value and nodes saved. The IC elements are given the clocking scheme of the first switch-pair of the SC elements to which integrator it belongs. The SC elements are given the clocking scheme of the second switch-pair of its description. The total number of capacitors and operational amplifiers is also counted here. The flowchart of the INIT function is shown in Fig. 5.5.

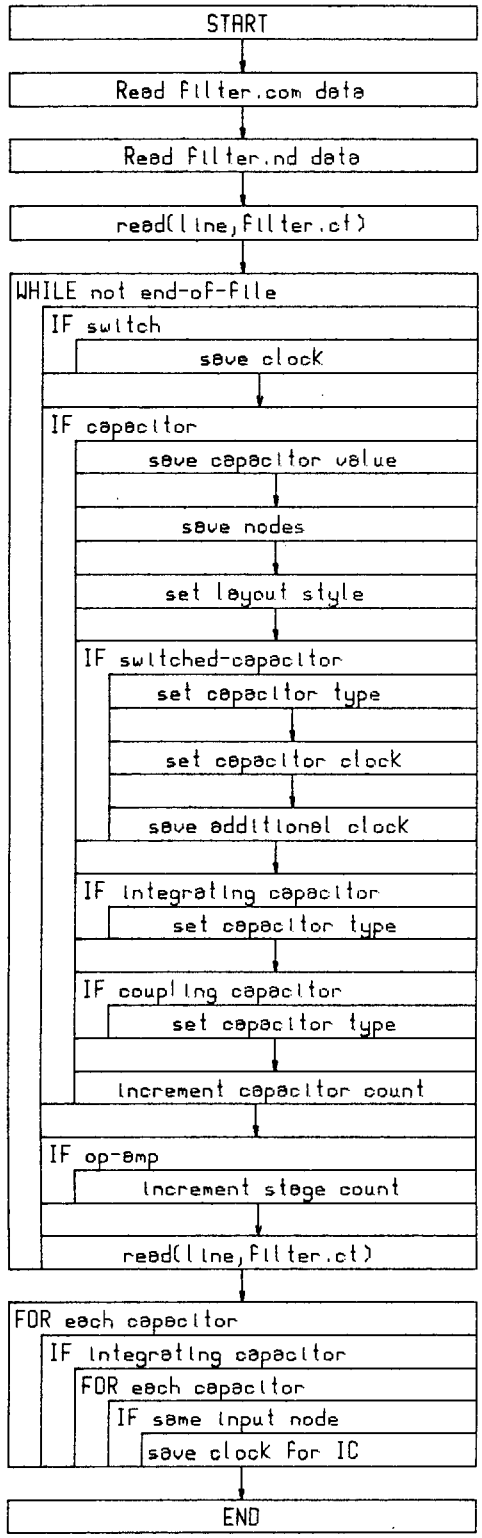


Fig. 5.5: INIT function flowchart

5.4 THE CAPACITORS

5.4.1 THE CAPACITOR ARRAY

The capacitor array layout is done by the CAPARRAY function. It is important to achieve a high accuracy (typically 0.1%) in the capacitor area ratios to ensure a high quality filter response. The time constant τ of the integrator is given by

$$\tau = \frac{TC_1}{C_2} \tag{5.1}$$

where T is the clockrate. Assuming an accurate clock, the problem is to minimize the capacitor ratio error in the capacitor placement [13][14].

Let the error in capacitor C_1 be ΔC_1, and let the error in capacitor C_2 be ΔC_2. The capacitor ratio is

$$\frac{\tilde{C}_1}{\tilde{C}_2} = \frac{C_1 \pm \Delta C_1}{C_2 \pm \Delta C_2} \tag{5.2}$$

where $\tilde{C}_1$ and $\tilde{C}_2$ are the realized capacitances. This can be rearranged as

$$\frac{\tilde{C}_1}{\tilde{C}_2} = \left(\frac{C_1}{C_2} \right)\left(\frac{1 \pm \Delta C_1/C_1}{1 \pm \Delta C_2/C_2} \right) \tag{5.3}$$

The capacitance C_1 is equal to

$$C_1 = \frac{\epsilon A_1}{d} \tag{5.4}$$

where ϵ is the SiO_2 permittivity, A_1 is the capacitor area, and d is the capacitor plate separation. Assuming that d and ϵ are constant for all capacitors, equation 5.3 can be rewritten as

$$\frac{\tilde{C}_1}{\tilde{C}_2} = \left(\frac{C_1}{C_2} \right)\left(\frac{1 \pm \Delta A_1/A_1}{1 \pm \Delta A_2/A_2} \right) \tag{5.5}$$

To satisfy the condition

$$\frac{\tilde{C}_1}{\tilde{C}_2} = \frac{C_1}{C_2} \tag{5.6}$$

it is required that

$$\frac{\Delta A_1}{A_1} = \frac{\Delta A_2}{A_2} \tag{5.7}$$

Assuming a systematic error in the capacitor areas, the area error is approximately equal to

$$\Delta A = Pdx \tag{5.8}$$

where P is the capacitor perimeter. Equation 5.7 is then equivalent to

$$\frac{P_1}{A_1} = \frac{P_2}{A_2} \tag{5.9}$$

that is, the perimeter-to-area ratio of the capacitors should be a constant. This can be accomplished by making the smaller capacitor a square, and the larger capacitor an integral number of these squares plus a fractional amount if

necessary.

This is also convenient from the viewpoint of layout strategy because the capacitors can be placed one square at a time. Typically one capacitor square equals one unit of capacitance.

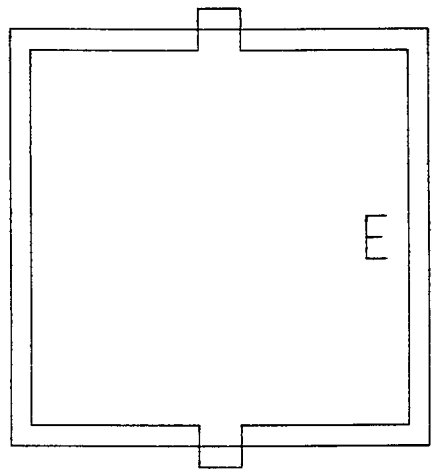
The layout strategy must yield accurate area ratios, good perimeter-to-area ratios, and immunity from the effects of polysilicon1-polysilicon2 mask misalignment.

5.4.2 LAYOUT METHOD

The unit capacitor plate is shown in Fig. 5.6. The edges of the polysilicon2 square are each of length E, and the polysilicon1 square overlaps it by 1λ. Each unit plate is joined to the previous unit plate by a 2λ by 2λ polysilicon1 link. The first unit plate will have a polysilicon2 wire joined to it later, when the capacitors are wired together. To provide polysilicon1-polysilicon2 mask misalignment immunity, a similar polysilicon2 link is later put on top of one of the capacitor plates. These extra polysilicon2 links make it impossible to achieve a perfect perimeter-to-area ratio, but it is still possible to maintain accuracy to 0.1%.

The perimeter-to-area ratio of a unit size capacitor is then

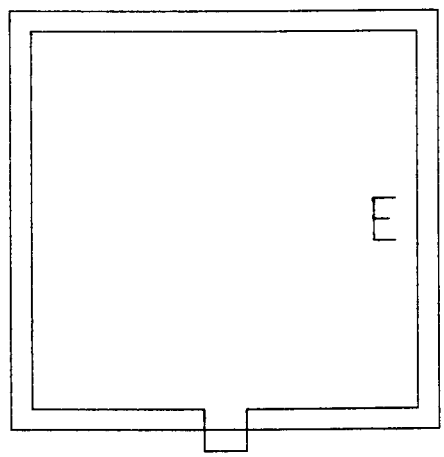
$$r = \frac{P}{A} = \frac{4E+4}{E^2+4} \qquad (5.10)$$



$$A = E^2 + 4$$

$$P = 4E + 4$$

Fig. 5.6: First capacitor plate



$$A_+ = E^2 + 4$$

$$P_+ = 4E$$

Fig. 5.7: Additional capacitor plates

and the ratio of an additional plate (Fig. 5.7) is

$$r = \frac{P_+}{A_+} = \frac{4E}{E^2+4} \qquad (5.11)$$

where P_+ is the additional perimeter and A_+ is the additional area of the added square. The plates join together providing a continuous polysilicon1 layer.

A fractional capacitance is laid out in one of two possible ways. If the fraction is small, the following method is used. Let the fractional value be f, and the parameters a and b as shown in Fig. 5.8.

The perimeter-to-area ratio is

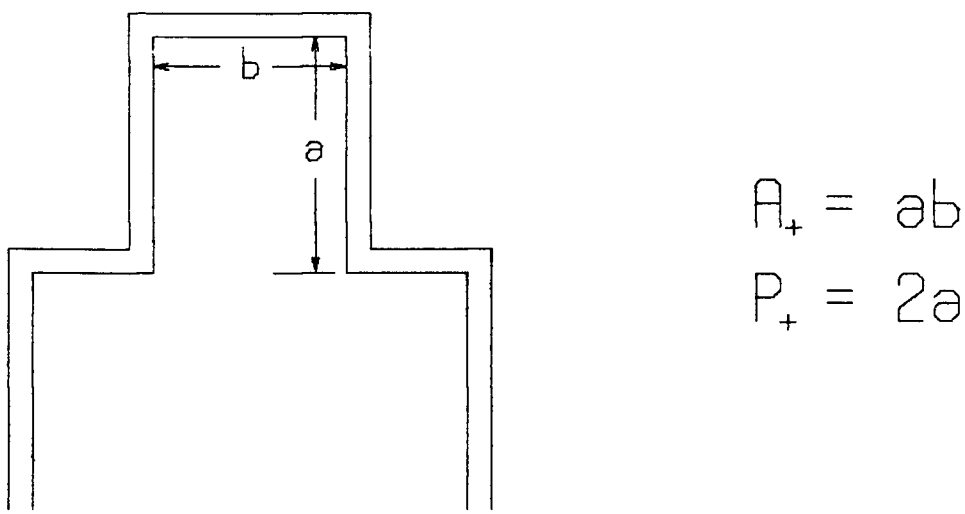


Fig. 5.8: Small fractional add-on capacitor

$$r = \frac{P_+}{A_+} \tag{5.12}$$

and it is required that

$$fA_+ = ab \tag{5.13}$$

This gives

$$r = \frac{P_+}{A_+} = \frac{2a}{ab} \tag{5.14}$$

hence

$$b = \frac{2}{r} \tag{5.15}$$

and

$$a = \frac{fP_+}{2} \tag{5.16}$$

Now substituting in the values of A and P

$$A = E^2+4 \tag{5.17}$$

$$P = 4E \tag{5.18}$$

gives

$$a = 2fE \tag{5.19}$$

and

$$b = \frac{E}{2} + \frac{2}{E} \tag{5.20}$$

A larger fractional capacitance is laid out as shown in Fig. 5.9, where the parameters a and b are defined. The desired ratio is again

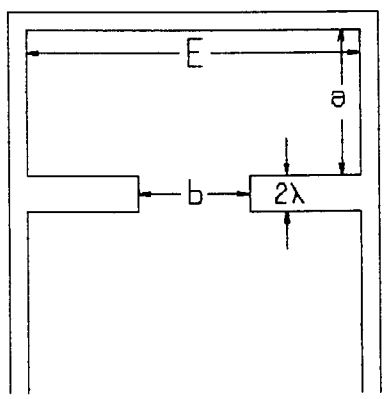
$$r = \frac{P_+}{A_+} \tag{5.21}$$

and

$$fA_+ = aE + 2b \tag{5.22}$$

The additional perimeter is

$$P_+ = 2E + 2a - 2b + 4 \tag{5.23}$$



$$A_+ = aE + 2b$$
$$P_+ = 2E+2a-2b+4$$

Fig. 5.9: Large fractional add-on capacitor

giving the following relations

$$a = \frac{f(A_+ + P_+)}{E+2} - 2 \qquad (5.24)$$

$$b = E - \frac{f(EP_+ - 2A_+)}{2E+4} \qquad (5.25)$$

Now substituting in 5.17 and 5.18 gives

$$a = f(E + 2) - 2 \qquad (5.26)$$

$$b = E - f(E - 2) \qquad (5.27)$$

The first style is good for smaller fractions as a will be larger than E if the fraction is larger than .5, and this will not fit in the array. The second style is useful for larger fractions as a must be larger than 2λ. This means that the fraction must be larger than 4/(E+2).

5.4.3 CAPACITOR ACCURACY ANALYSIS

This layout method results in total immunity to polysilicon1-polysilicon2 mask misalignments of less than 1λ. Assume now that all polysilicon1 edges are slightly off by an amount x (Fig. 5.10).

Let capacitor one have a value of m, and capacitor two have a value of n+m, where m and n are positive integers. The achieved values of these capacitors are

$$C_1 = m(E^2+4) - 4x \qquad (5.28)$$

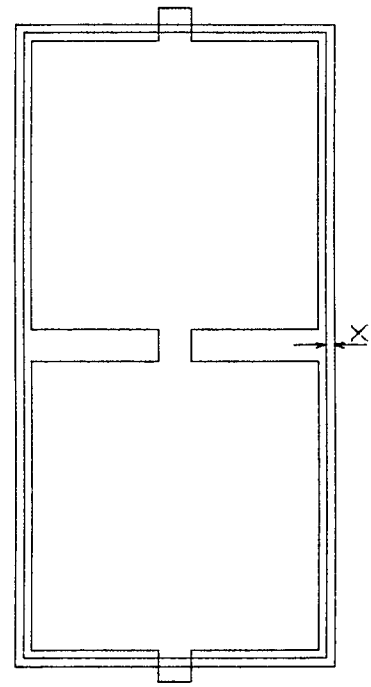


Fig. 5.10: Polysilicon1 systematic error

$$C_2 = (n+m)(E^2+4) - 4x \tag{5.29}$$

giving the ratio

$$\frac{C_2}{C_1} = \frac{(n+m)(E^2+4) - 4x}{m(E^2+4) - 4x} \tag{5.30}$$

or

$$\frac{C_2}{C_1} = 1 + \frac{n(E^2+4)}{m(E^2+4) - 4x} \tag{5.31}$$

then

$$\frac{C_2/C_1 - \tilde{C}_2/\tilde{C}_1}{C_2/C_1} = \left(\frac{n}{m+n}\right)\left(\frac{-4x}{m(E^2+4) - 4x}\right) \quad (5.32)$$

or

$$\Delta(C_2/C_1) \simeq \left(\frac{n}{m(m+n)}\right)\left(\frac{-4x}{E^2+4}\right) \quad (5.33)$$

to maximize this error, let m = 1, and n = ∞, then

$$|\Delta(C_2/C_1)| = \frac{4x}{E^2+4} \quad (5.34)$$

Now it is required that

$$\frac{4x}{E^2+4} \leq 0.001 \quad (5.35)$$

so

$$x \leq \frac{E^2+4}{4000} \quad (5.36)$$

For 0.1% accuracy and E = 18λ

$$x \leq \frac{\lambda}{12} \quad (5.37)$$

This value of E is picked as it is the minimum width of a switch-pair, and keeping the capacitor plate the same size simplifies the layout.

Now assume that all polysilicon2 edges are slightly off by an amount x (Fig. 5.11). Using the same capacitors as in the previous analysis gives

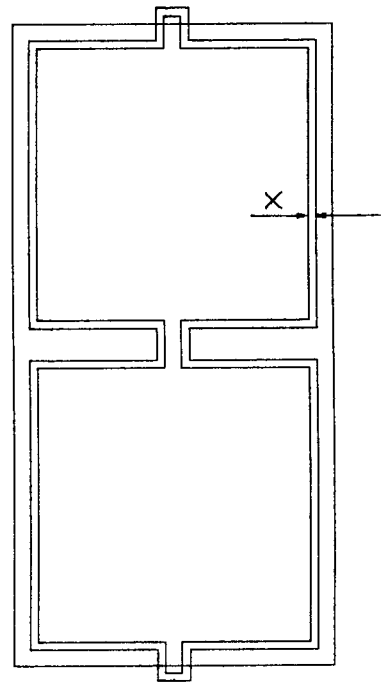


Fig. 5.11: Polysilicon2 systematic error

$$\frac{C_2}{C_1} =$$

$$\frac{[(E-2x)^2+2(2-2x)]+(m+n-2)[(E-2x)^2+(2-2x)(2+2x)]}{[(E-2x)^2+2(2-2x)]+(m-1)[(E-2x)^2+(2-2x)(2+2x)]} \qquad (5.38)$$

or

$$\frac{C_2}{C_1} = 1 +$$

$$\frac{(n-1)[(E-2x)^2+(2-2x)(2+2x)]}{[(E-2x)^2+2(2-2x)]+(m-1)[(E-2x)^2+(2-2x)(2+2x)]} \qquad (5.39)$$

so

$$\frac{C_2}{C_1} = 1 + (n-1)[m + (\frac{4x^2-4x}{E^2-4xE+4})]^{-1} \qquad (5.40)$$

since x << E, this can be simplified to

$$\frac{C_2}{C_1} = 1 + (n-1)(m - \frac{4x}{E^2})^{-1} \qquad (5.41)$$

or

$$\frac{C_2}{C_1} = 1 + \frac{n-1}{m - 4(x/E^2)} \qquad (5.42)$$

then

$$\Delta(C_2/C_1) = (\frac{n-1}{m(m+n-1)})(\frac{-4x}{E^2}) \qquad (5.43)$$

to maximize this error, let m = 1, and n = ∞, then

$$|\Delta(C_2/C_1)| = \frac{4x}{E^2} \qquad (5.44)$$

Now it is required that

$$\frac{4x}{E^2} \leq 0.001 \qquad (5.45)$$

so

$$x \leq \frac{E^2}{4000} \qquad (5.46)$$

For 0.1% accuracy and E = 18λ

$$x \leq \frac{\lambda}{12} \tag{5.47}$$

This method is equally tolerant of polysilicon1 and polysilicon2 systematic errors. For a 5μm technology, this requires a maximum error of 0.21μm on 45μm edges.

5.4.4 GENERATING THE ARRAY (CAPARRAY)

The capacitors are arranged in an array, their bases on the bottom for later connections. The array height is set by the arrayheight variable in the filter.com file. An initial guess for this variable is the average capacitor value.

Each capacitor can be laid out in two styles: one emphasizes up-down placement of the capacitor plates and the other emphasizes left-right placement. The capacitor size at which the style changes is set by the arrayswitch variable. If a capacitor is larger than this value it is placed with an up-down emphasis. The up-down spread is better for larger capacitors as it keeps them more compact, while the left-right spread is better for the smaller ones, letting them use the space between the larger ones. An initial guess for this value is one-and-a-half times the array height.

Each capacitor has its base plate placed along the base of the array. Additional plates are added one at a time for each capacitor until the capacitor is finished or there is

no more room. A check is done at the end, and if a capacitor was not finished, an additional space is allocated along the base for the next attempt. This process is repeated until all the capacitors are placed successfully.

When all the capacitors are finished, a check is made on the width of the array. If the operational amplifiers will take up more room than the array, the extra space is inserted into the array on one more array generation.

The extra polysilicon2 tip on each capacitor is added with a fractional capacitor plate placement if possible, otherwise it is placed after the array generation.

If a capacitor of value less than one is given, it will be laid out as a rectangle, causing a more serious perimeter-to-area violation.

A sixth order band-pass filter designed by MPR is used in this chapter to demonstrate how the SISCL program works. The data for the filter is in Chapter 6. The flowchart of the array generation function is shown in Fig. 5.12. A layout with the arrayswitch variable set larger than all the capacitors is shown in Fig. 5.13. The width of this array is 706λ. A layout with the arrayswitch variable set to 0 is shown in Fig. 5.14 giving an array width of 746λ. A layout with the arrayswitch variable set to one-and-a-half times the array heigth is shown in Fig. 5.15, giving a width of 670λ. A combination of the two strategies usually yields the smallest layout.

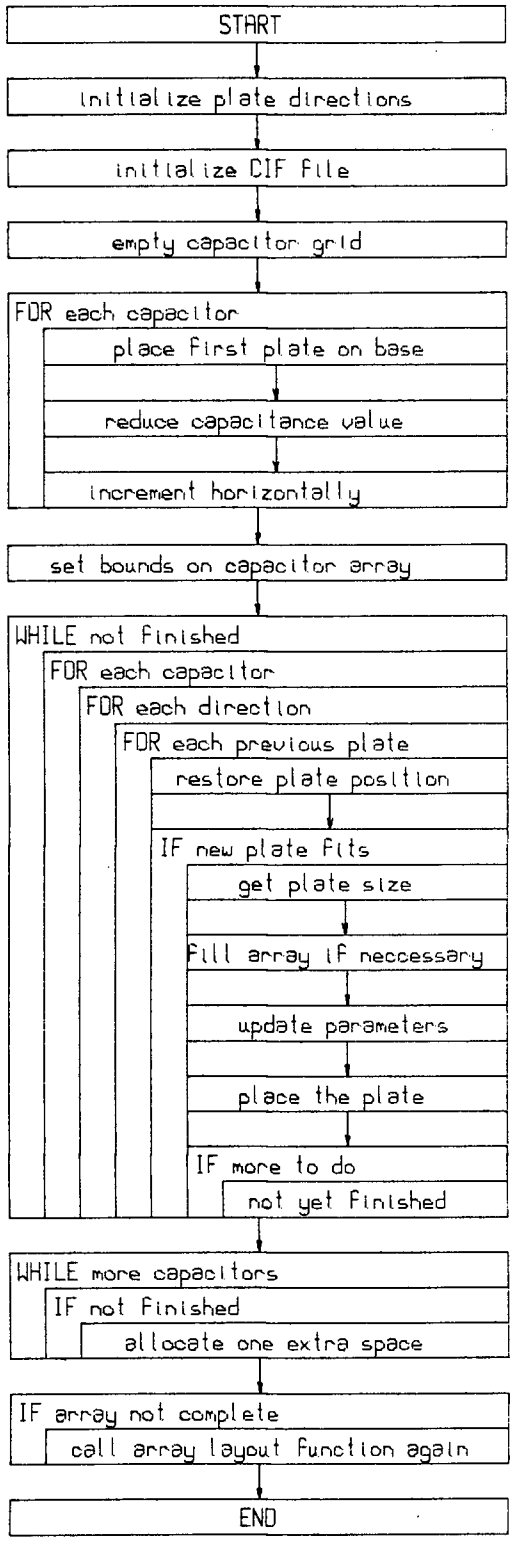


Fig. 5.12: Array layout flowchart (LAYARRAY)

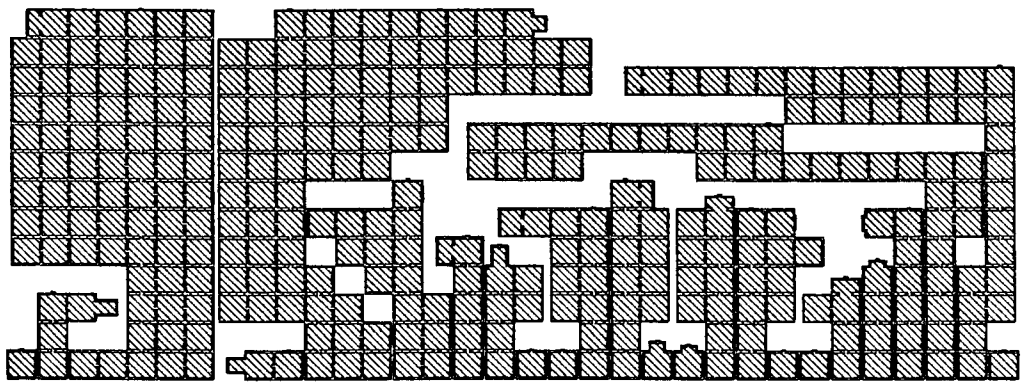


Fig. 5.13: Horizontal capacitor spread

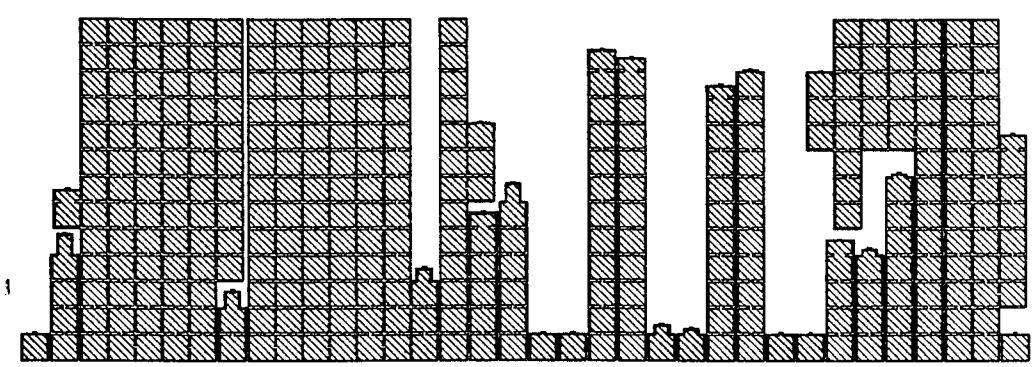


Fig. 5.14: Vertical capacitor spread

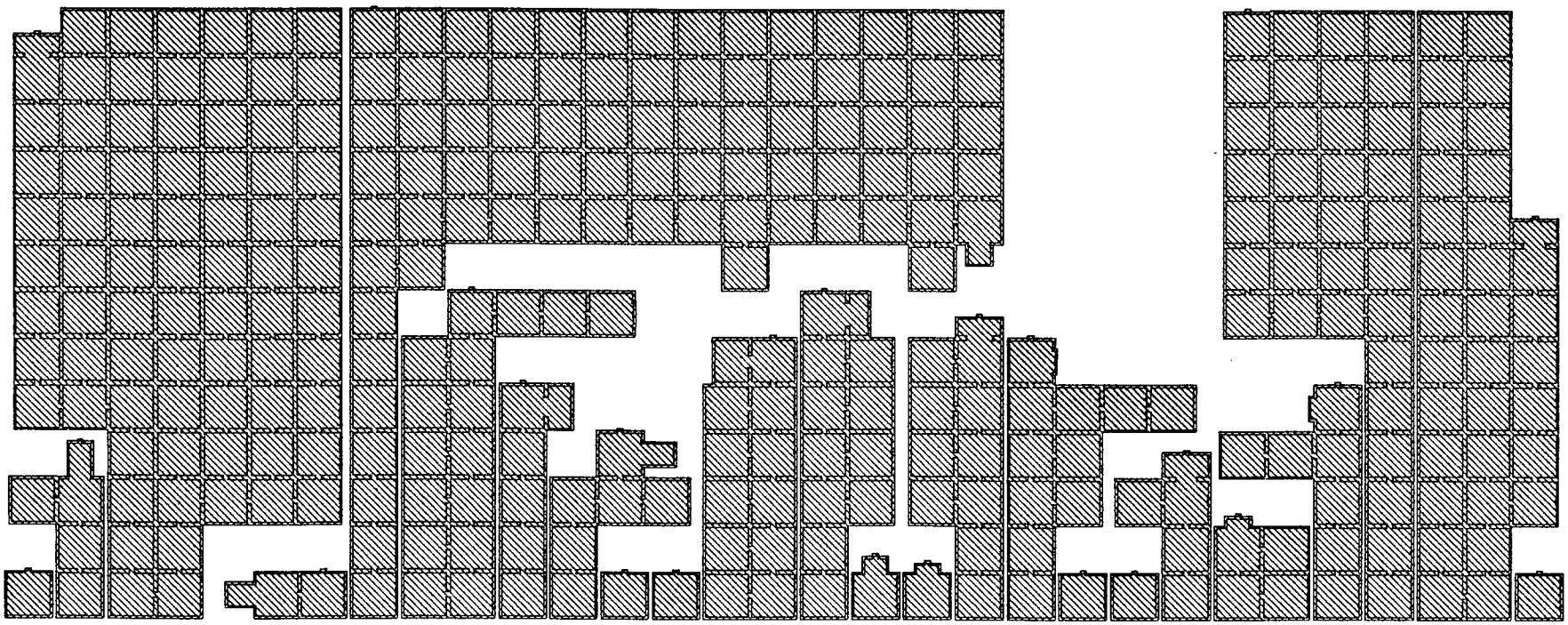


Fig. 5.15: Optimal capacitor array

5.5 SELECTING THE SWITCHES TO BE PLACED (PREOP)

The decision of where to place the operational amplifiers and where to place the switches is made by the PREOP function. The width of the capacitor array is again checked against the minimum width of the operational amplifiers and any extra space is inserted between the operational amplifiers. The placement point of the operational amplifier, with the co-ordinates of the three inputs/output, is stored in the OPAMP data structure.

The integrating capacitor for each operational amplifier gives the connection point for the inverting input. The switched capacitors connected to the operational amplifier output gives the possible connection points for the output. The closest switched capacitor to the output is selected. The clocking scheme of this switched capacitor is saved so that any switched capacitors with the complementing clocking scheme can be noted. The selected switched capacitor and one switched capacitor with the complementing clocking scheme are then marked for later switch-pair placement. All integrating capacitors have one switch-pair placed below them.

The operational amplifier is mirror imaged if the connections are simplified that way. The inputs/output are then moved and this will often greatly simplify the wiring from the switch-pairs to the operational amplifier inputs/output. A flowchart of the PREOP function is shown in Fig. 5.16.

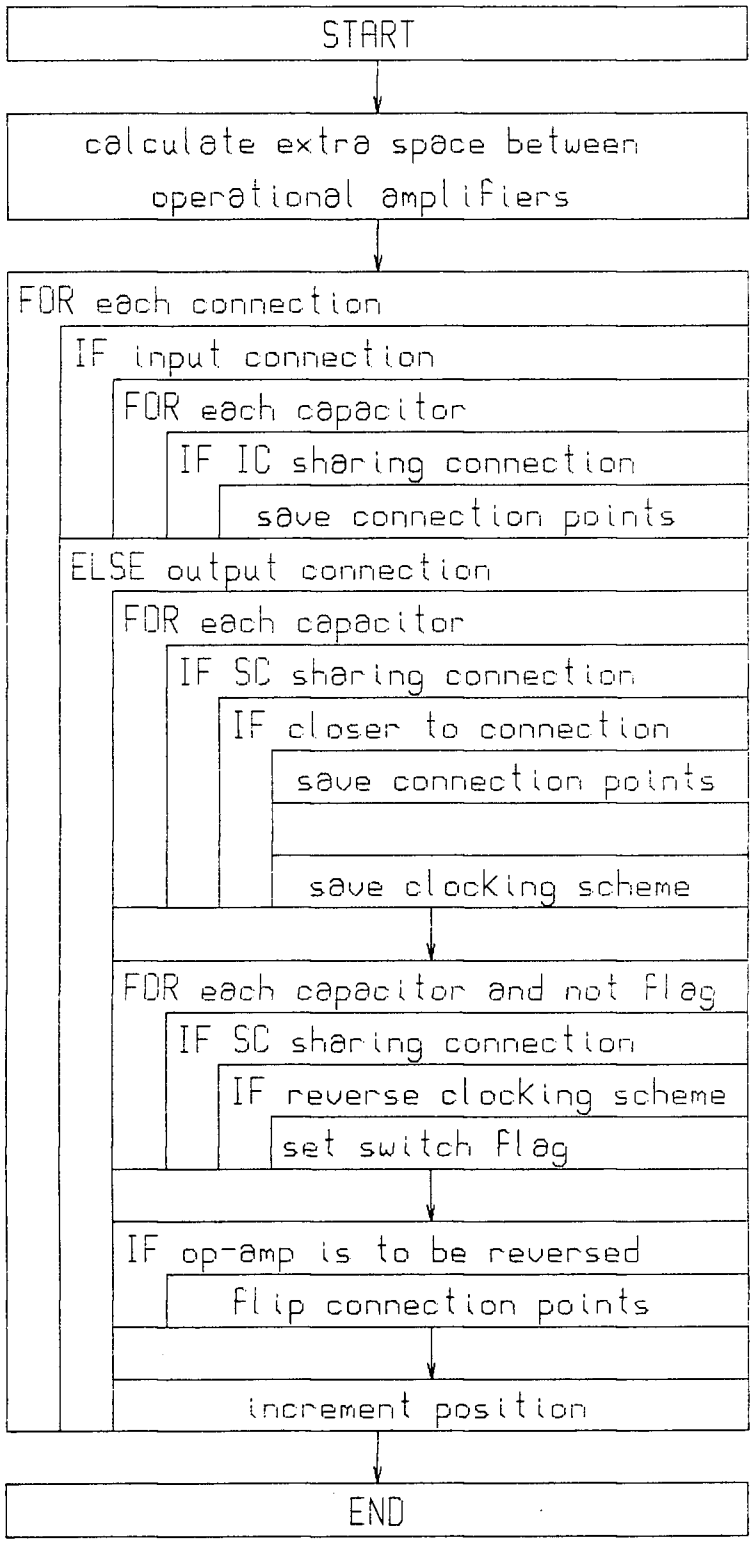


Fig. 5.16: PREOP function flowchart

5.6 THE CAPACITOR WIRING

5.6.1 INTRODUCTION

All wiring connections are made to the first capacitor plate placed for each capacitor. The base of this plate is wide enough for three connections, of which two connect to the plate. All the wiring is done in an array. The wires are all 2λ wide and are separated by 3λ. The horizontal wiring is done in metal as they tend to be longer, and the vertical wiring is done in polysilicon.

There are five types of connections to be made: the operational amplifier input node, its output node, the shared node at the input, and the two shared nodes at the output. Each type of connection is done by one function, however all the wiring functions are similar. The length of the wire joining the capacitors is found, and a slot is found in the wiring array for it. The wire is then placed in metal. Each capacitor is then checked to see if it should be connected to the wire. The polysilicon wires are then placed and connected to the metal wire. All integrating and switched capacitors have three vertical connections to the wiring, while the coupling capacitors only have two. The integrating and switched capacitors tend to have switches associated with them, causing the need for a third connection.

The nodes stored from the filter.nd file are scanned for each connection. If a capacitor should be connected to

the wire for a node, it is noted for the wire length, and the vertical connections. The wiring model is shown in Fig. 5.17. The flowcharts for the wiring functions are very similar to that of the first wiring function (WIRE1) which is shown in Fig. 5.18.

5.6.2 THE INPUT NODES (WIRE1)

All the elements connected to the operational amplifier inverting input node are wired up by the WIRE1 function. Each such node connects to one integrating capacitor and any coupling capacitors. If there are no coupling capacitors in the filter, no horizontal metal wire has to be placed, and no vertical polysilicon2 wire is placed at this time. Any vertical connections are made in polysilicon2 and are made

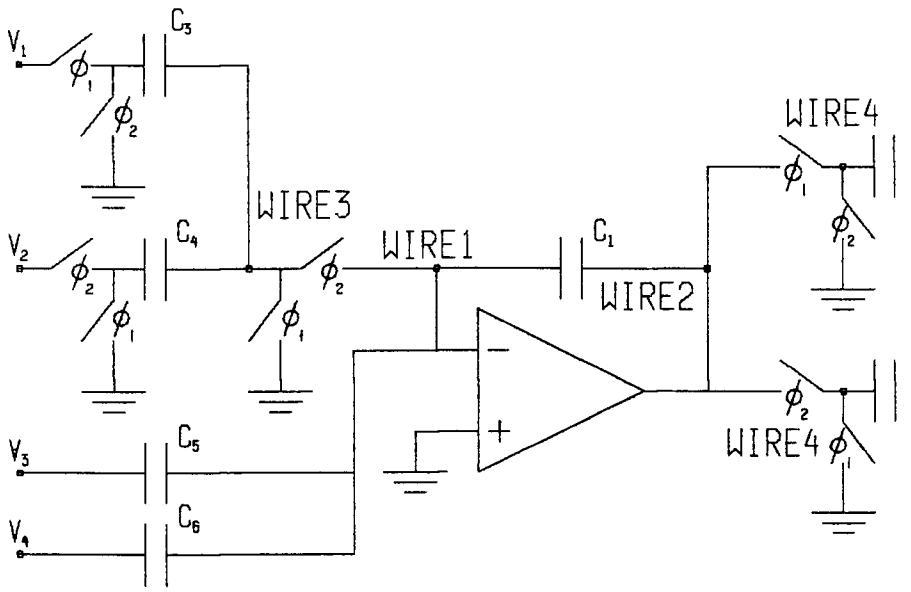


Fig. 5.17: Wiring model

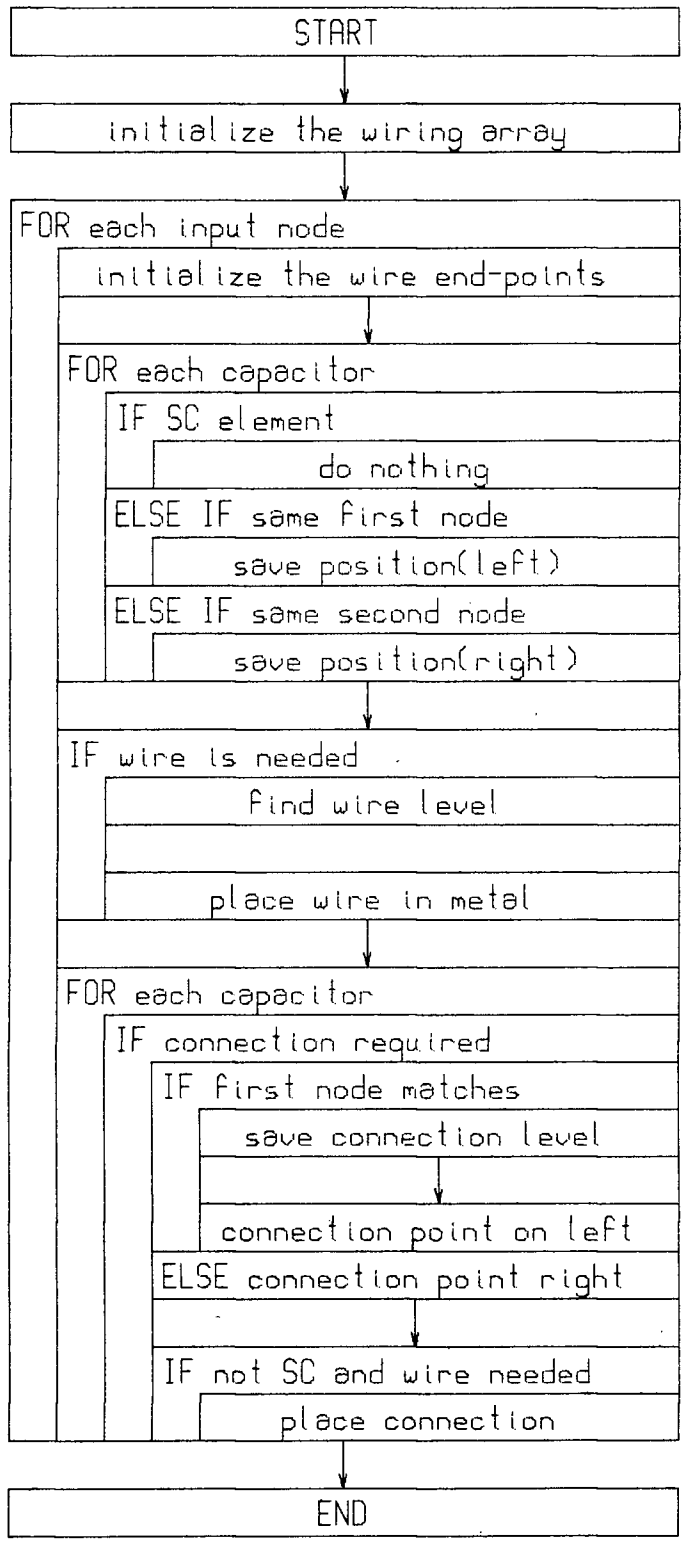


Fig. 5.18: WIRE1 function flowchart

on the left side of the integrating capacitor and on either side of the coupling capacitor, depending on where the node is. The output from the WIRE1 function is shown in Fig. 5.19.

5.6.3 THE OUTPUT NODES (WIRE2)

All the elements connected to the operational amplifier output node are wired up by the WIRE2 function. Each such node connects to one integrating capacitor, any coupling capacitors, and any switched capacitors fed by the operational amplifer. All vertical connections are made in polysilicon1. The integrating capacitor is connected on the right side, while the coupling capacitor is connected on either side, depending on where the node is. The switched capacitor is not connected to the wire as the connection is made later to the switchpair, on the capacitor's right side. The output from the WIRE2 function is shown in Fig. 5.20.

5.6.4 THE SHARED NODE AT THE INPUT (WIRE3)

All switched capacitors connected to the operational amplifier inverting input node are wired together now. The integrating capacitor has to be included in the horizontal wiring, as it is later connected to its switch-pair. The switched capacitors are connected with polysilicon1 wires. The output from the WIRE3 function is shown in Fig. 5.21.

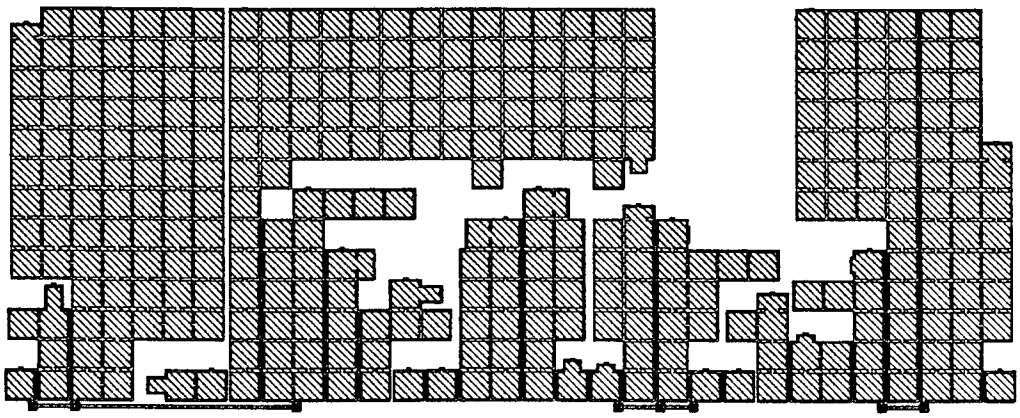


Fig. 5.19: Output from WIRE1

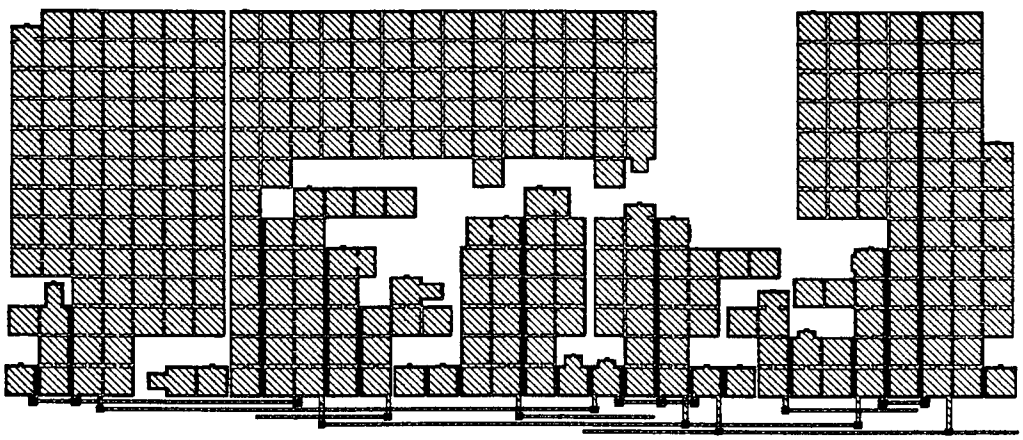


Fig. 5.20: Output from WIRE2

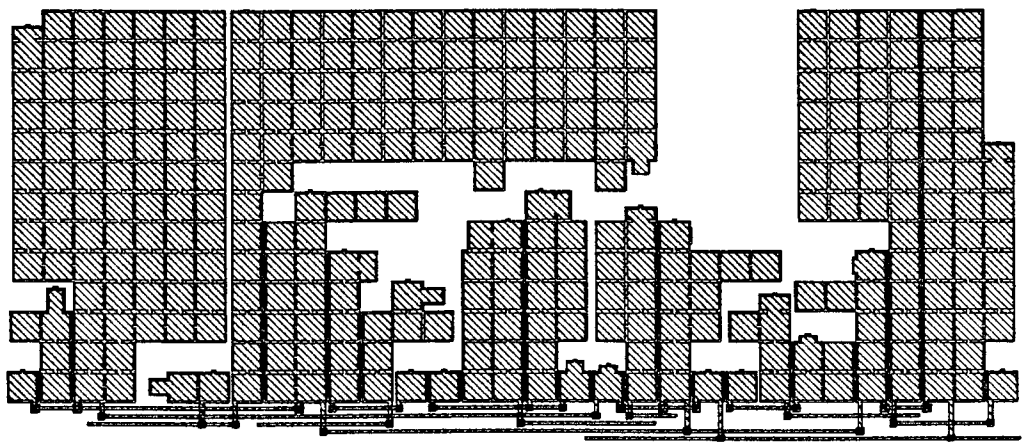


Fig. 5.21: Output from WIRE3

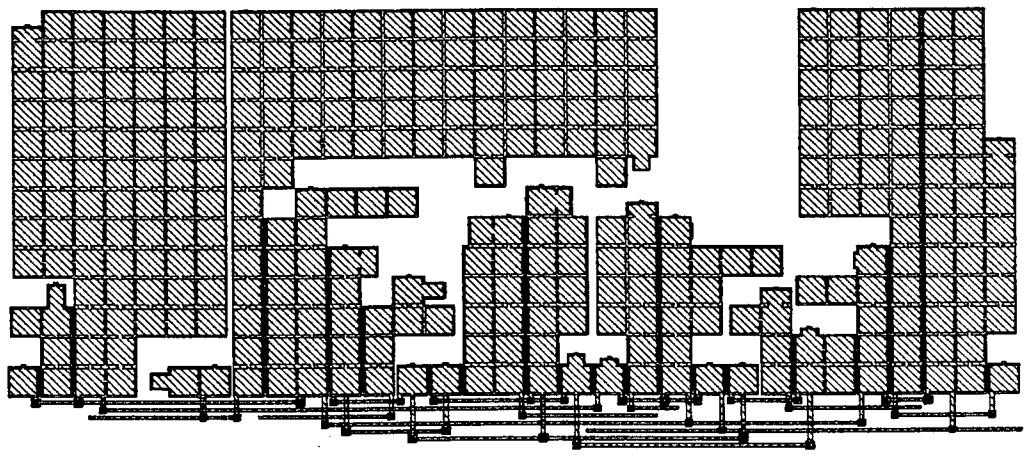


Fig. 5.22: Output from WIRE4

5.6.5 THE SHARED NODES AT THE OUTPUT (WIRE4)

All the switched capacitors connected to the operational amplifier output node are divided into two groups, according to which clocking scheme they use. Each group is wired separately. The connections are made on the right side of the switched capacitors in polysilicon1. The wire is later connected to the middle of a switch-pair. The output from the WIRE4 function is shown in Fig. 5.22.

5.6.6 THE SWITCH PLACEMENT AND CONNECTIONS (STREAK)

The STREAK function places the switch-pairs directly below the integrating capacitors and the switched capacitors selected by the PREOP function. The switches are then connected to the previous capacitor wiring, and the clocks are connected properly for each switch. The switch-pairs are grounded by metal wires connected to the ground line that is later placed directly below the switches. The integrating capacitors and switched capacitors have their central connection going straight to the middle switch connection in polysilicon2. The unconnected third vertical wire for the integrating and switched capacitors is now connected to the switch-pair directly above those elements. If there is an accessible contact above the switch, the software tries not to place another contact before connecting them. A flowchart of the STREAK function is shown in Fig. 5.23 and the output is shown in Fig. 5.24.

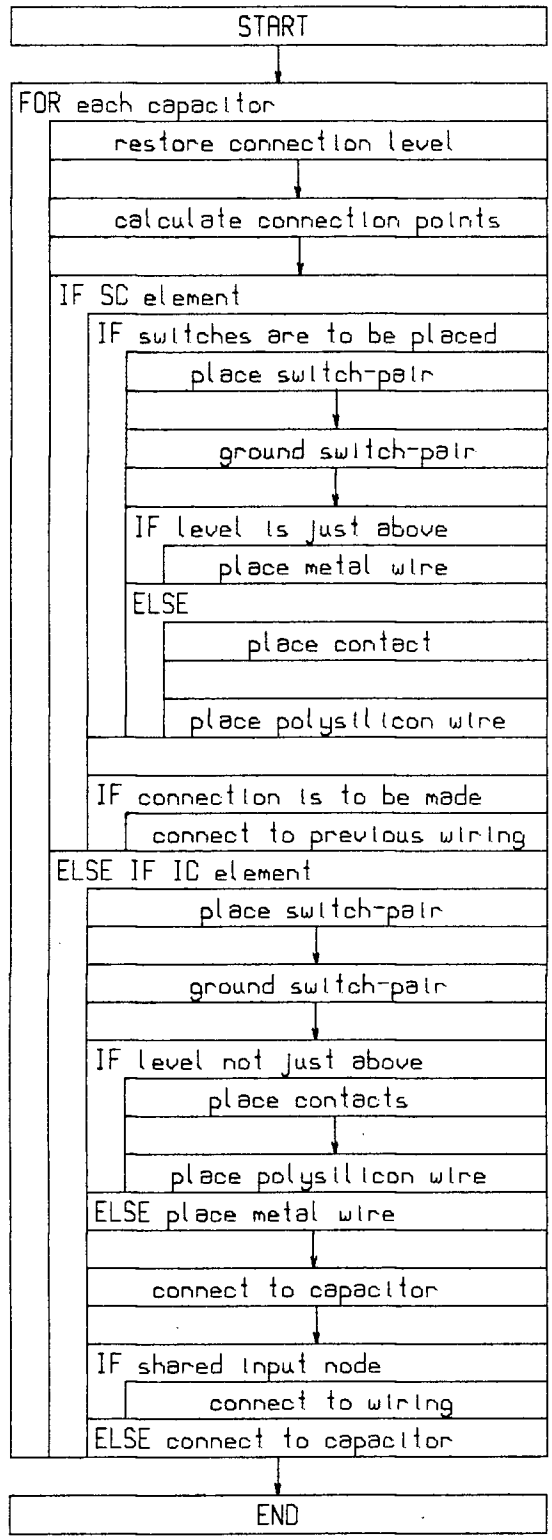


Fig. 5.23: STREAK function flowchart

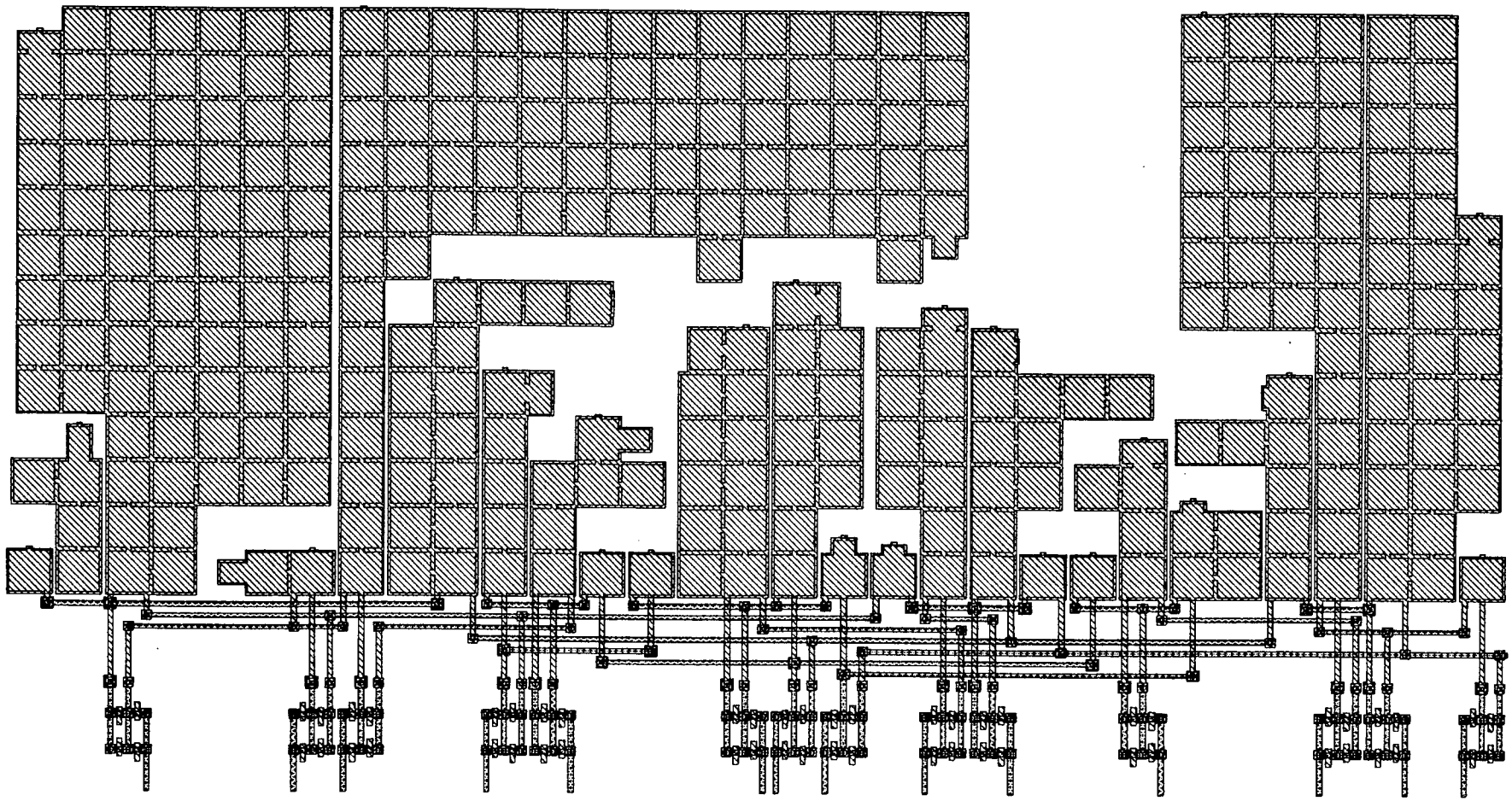


Fig. 5.24: Output from STREAK function

5.7 PLACING AND CONNECTING THE OPERATIONAL AMPLIFIERS

5.7.1 CONNECTING THE INPUT AND OUTPUT NODES (WIRE5)

All the connection data for the operational amplifier inputs/output as well as the switch connection points have been previously gathered. The connections for the operational amplifier inverting inputs and outputs are done by the WIRE5 function. These connections are done the same way as they are all connections from one co-ordinate pair to another co-ordinate pair. There will be metal ground and power lines separating the operational amplifiers from the switches so the connections are done in polysilicon1.

It is necessary to place a metal-polysilicon1 contact below the switch-pairs as a polysilicon1 clock line will separate them. The power lines then go below the contact. This provides room for one level of connections between these contacts. If a connection can be done on this level, it is placed entirely in polysilicon1. Some 80% of the connections are placed on this level. If this is not possible, horizontal metal wires are placed below the power lines and vertical polysilicon1 lines connect them to the switch-pair contacts.

The wiring is again done in an array. The horizontal bounds are found for each wire and the wire is placed in the first available level.

The connection to the switch is always made on one side of the switch, hence there is room to laterally move the

connection point by up to 14λ. This is done if the vertical connection happens to be directly above another operational amplifer input/output. This reduces the number of wiring levels as wires do not have to go around previous connections as often. The flowchart of the WIRE5 function is shown in Fig. 5.25 and the output is shown in Fig. 5.26.

5.7.2 CONNECTING THE GROUND NODES (WIRE6)

The noninverting inputs of the operational amplifiers are always connected to ground. This wiring is done by the WIRE6 function. Usually the connection is a simple vertical polysilicon1 wire, but if there is another connection blocking its path, it has to be laterally moved around the connection. The connection is moved until the offending connection no longer blocks the ground wire. The wire is always moved away from the inverting input. Only the horizontal wires are placed here as the vertical connections to the operational amplifiers are done later. If no lateral moves are necessary the function only places contacts on the ground line directly below the input. The output from the WIRE6 function is shown in Fig. 5.27.

5.7.3 PLACING THE OPERATIONAL AMPLIFIERS (AMPSON)

The operational amplifiers are now placed directly below the wiring. The vertical placement can be adjusted with the yorg variable in the filter.com file. The connections are then made from the previous wiring to the

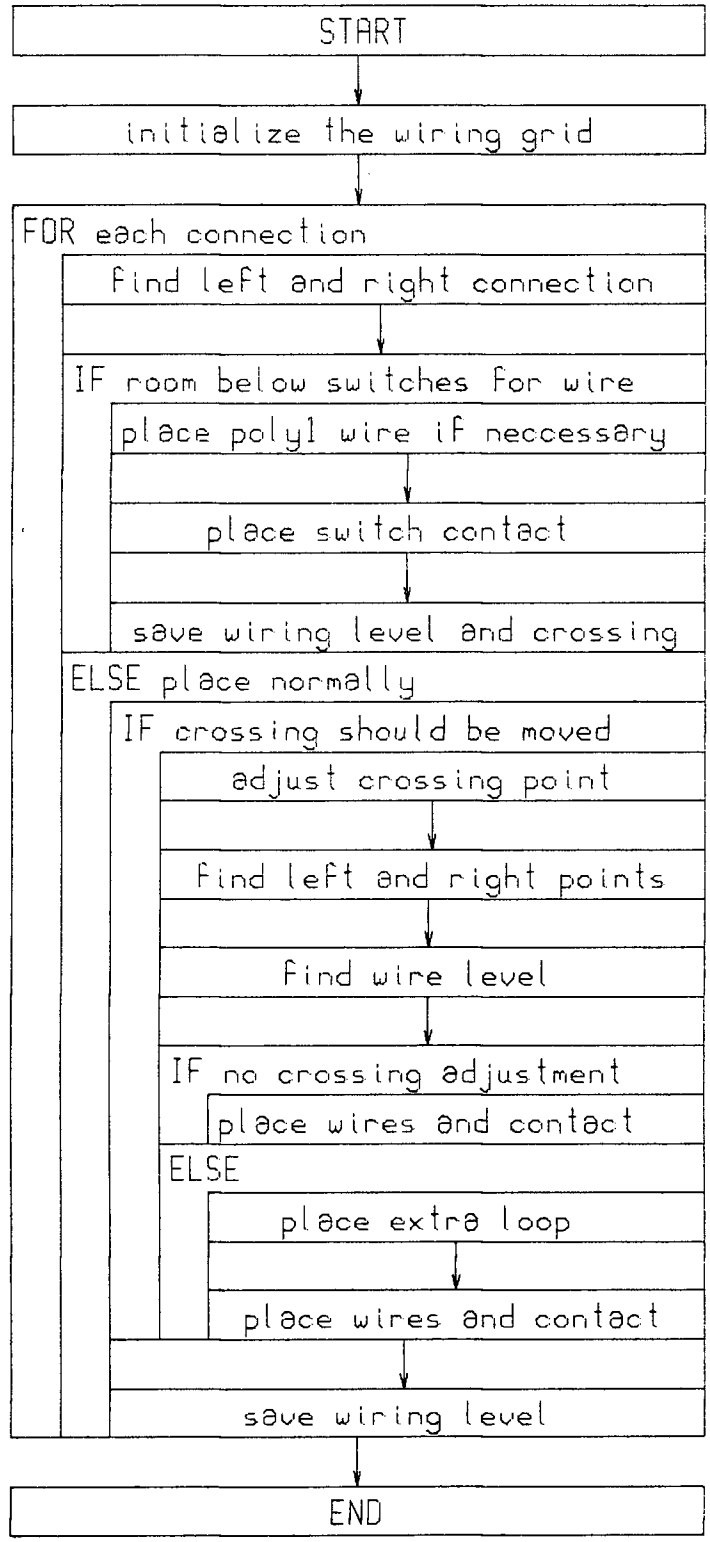


Fig. 5.25: WIRE5 function flowchart

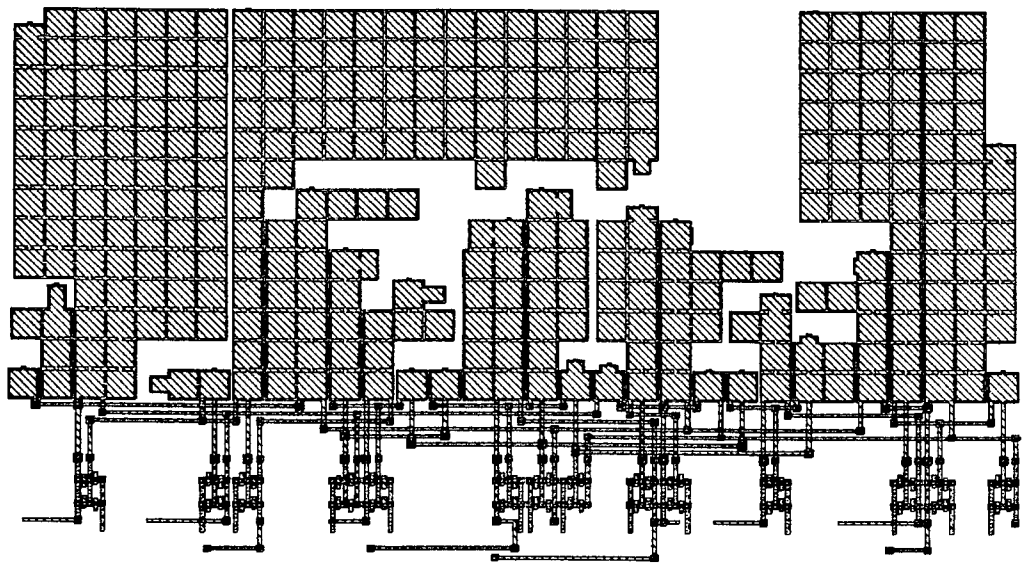


Fig. 5.26: Output from WIRE5 function

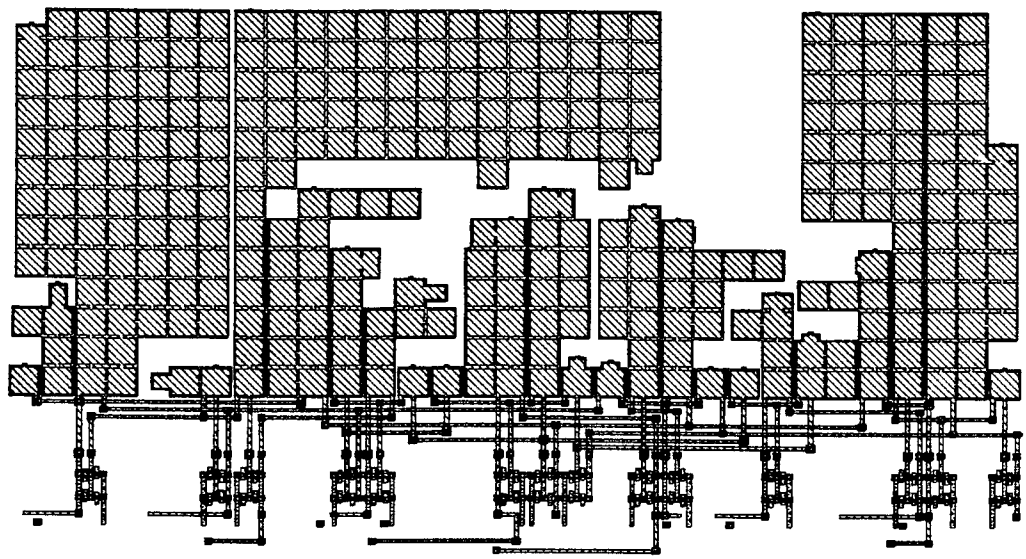


Fig. 5.27: Output from the WIRE6 function

operational amplifier inputs and output. The output from the AMPSON function is shown in Fig. 5.28. If the inputs had been reversed on the operational amplifier the output in Fig. 5.29 would result.

5.8 PLACING THE GLOBAL FEATURES (WRAPUP)

The overall features of the layout are placed by the WRAPUP function. The capacitor array and its wiring is placed inside a p-well to isolate it from switch noise. The well is anchored by the ground line to keep the operational amplifier power lines less noisy. The p-well is also placed for the switches, and is anchored there by the Vss line. The N+ and P+ areas for the switches are then placed. The output line is drawn outside the capacitor array, beyond the ground line. Substrate contacts are also placed on the bottom of the capacitor array p-well, where space permits.

The layout then consists of the top capacitor array p-well, above the p-transistor switches in the substrate. This is followed by the n-transistors in the second p-well and finally the operational amplifiers. This arrangement cuts down the switch noise in the filter by letting the p-wells act as substrate barriers. The output of the WRAPUP function is shown in Fig. 5.30.

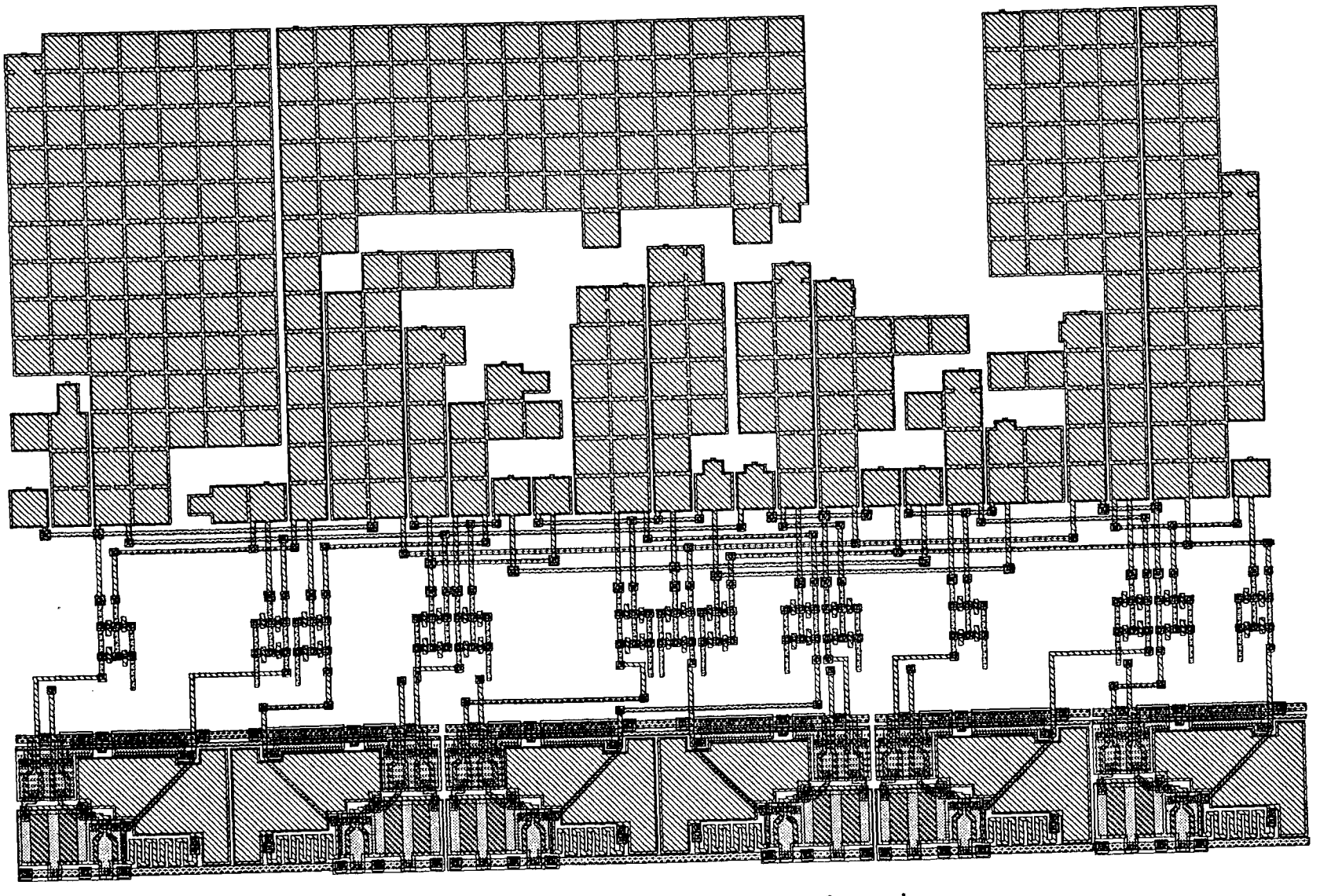


Fig. 5.28: Output from AMPSON function

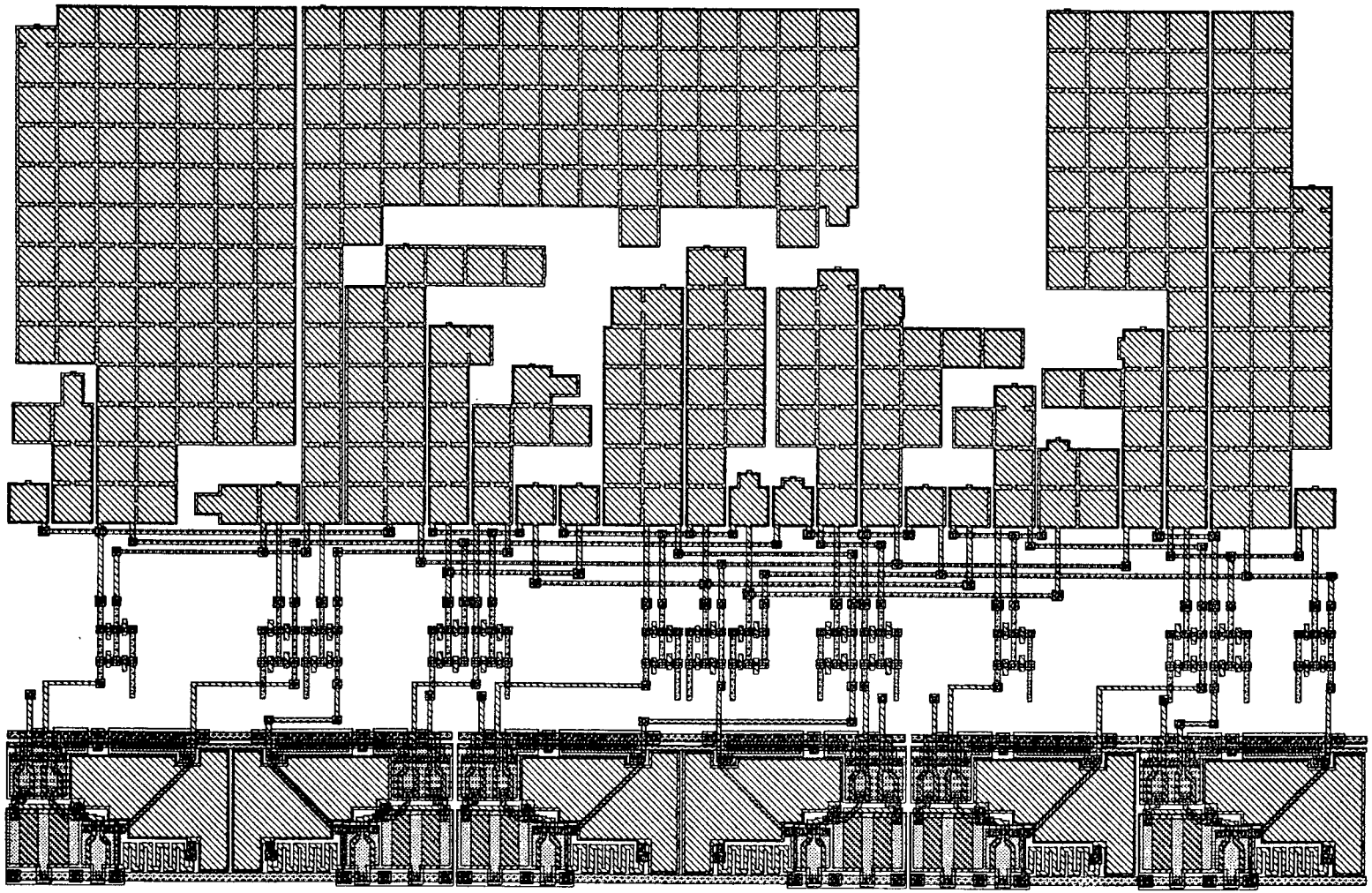


Fig. 5.29: AMPSON output for reversed inputs

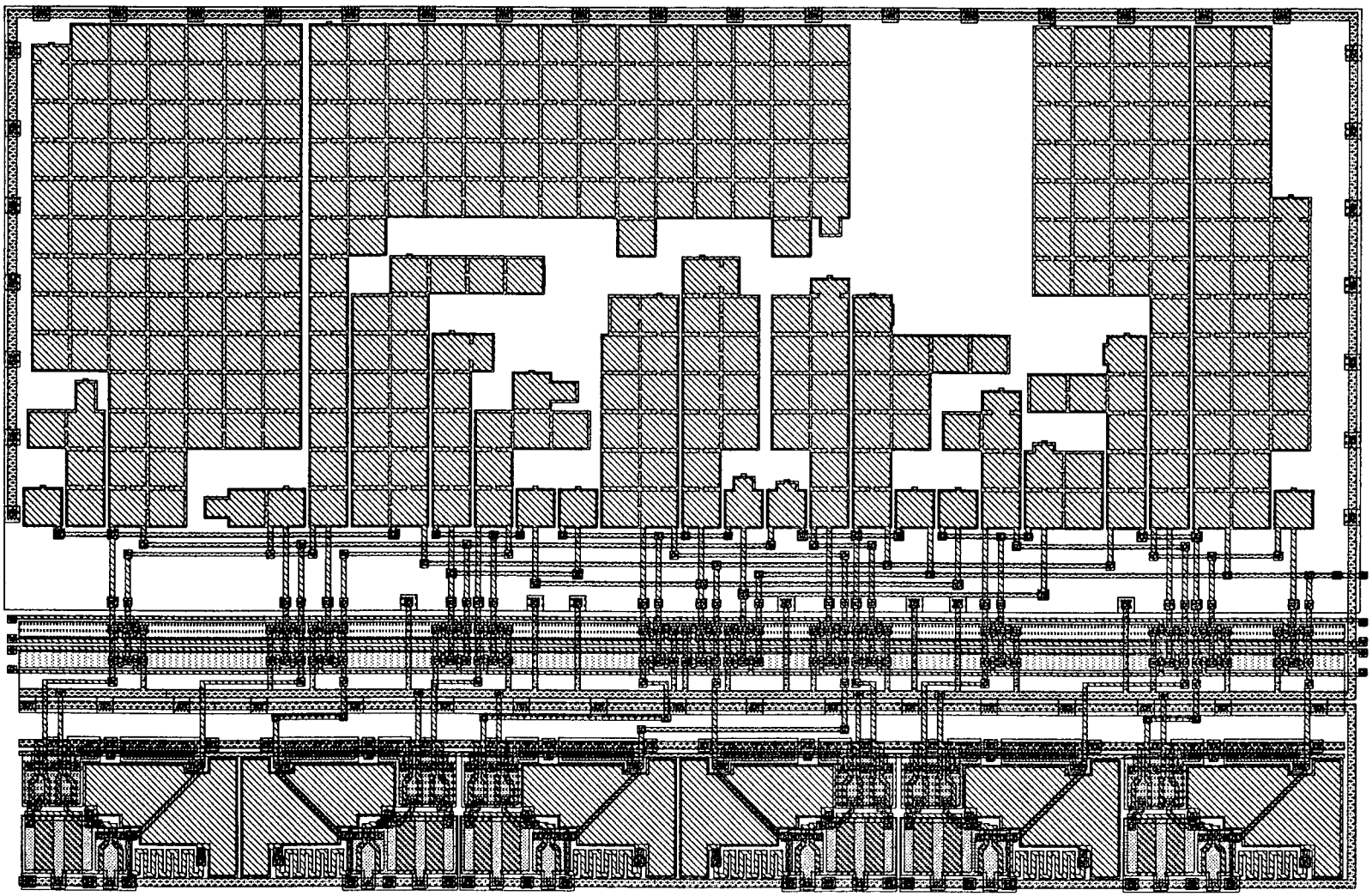


Fig. 5.30: Output from the WRAPUP function

5.9 LAYOUT OF THE INPUT STAGE

There are currently four INPUT functions as each INPUT function connects one of the different input stages. The input stage of Lee (Fig. 4.1) requires one extra switch-pair and some connections to be made. The switch-pair is placed below the second capacitor and is then connected to it. The bottom plate of the second capacitor is connected to ground, and then the input node is connected to the first capacitor and to the extra switch-pair. The final layout of the design is shown in Fig. 5.31.

For the Martin case it is only necessary to connect the input node to the first switch-pair.

The input stage of Datar and Sedra (Fig. 4.2) is the most complex one. A switch-pair is placed below the extended first capacitor and another one is placed below the second capacitor. The bottom plate of the first capacitor is then grounded. These switch-pairs are connected to the first switched capacitor of the filter, after which the two input switched capacitors are joined together, and connected to the operational amplifier output and noninverting nodes. A single switch is placed below the first capacitor to be used as the filter input switch. The input node is then connected to this switch.

The AROMA input stage is very similar to the Martin case. The input node is connected to the input switched capacitor(s). There can be one or three of these depending on whether the filter has a first order stage or a second

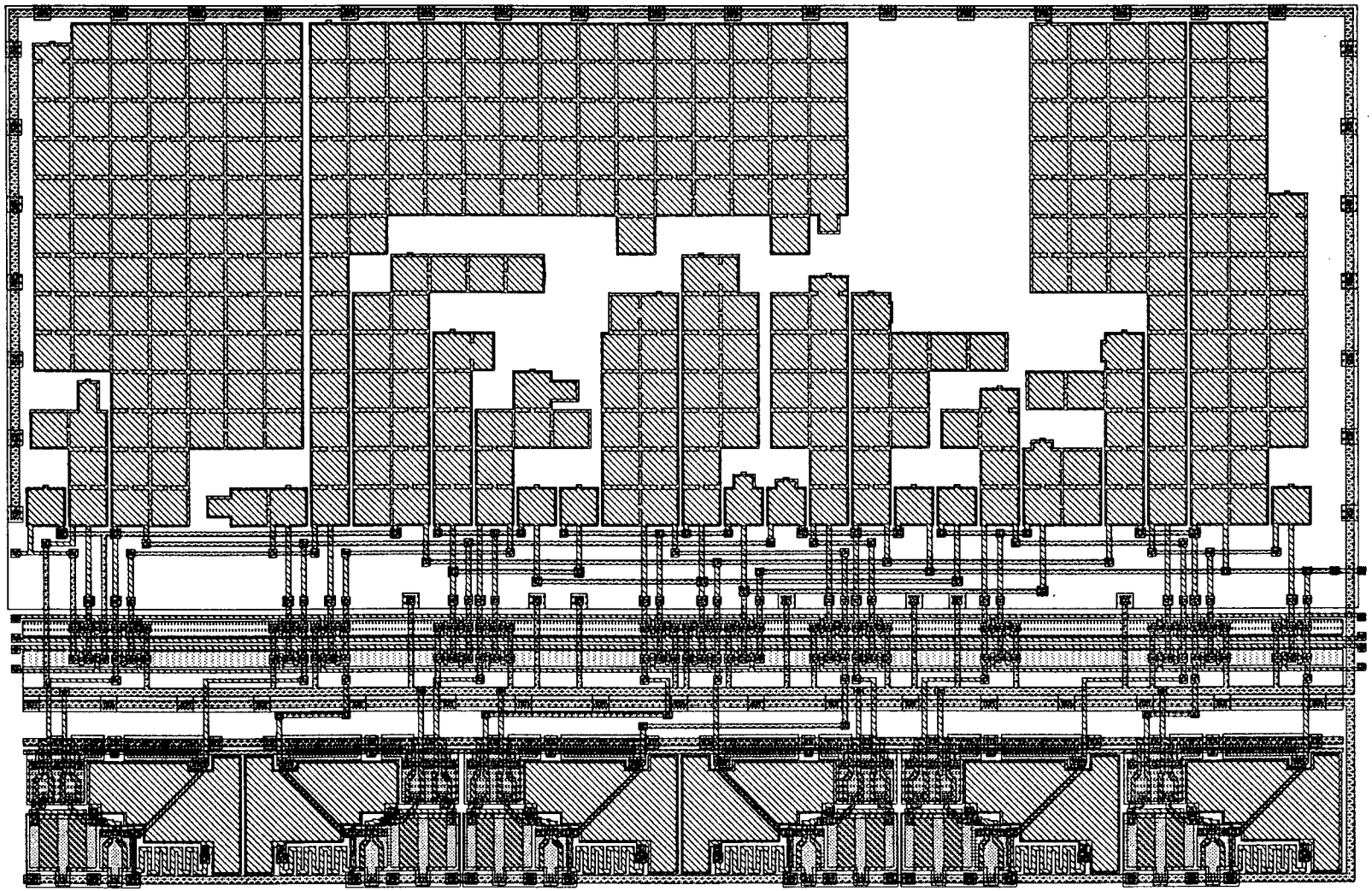


Fig. 5.31: Final layout

order stage at the input. If there is a second order stage, two of the capacitors are connected together, sharing a switch-pair. The switched capacitor with the complementing clocking is connected to the input node, as is the first switched capacitor.

6. RESULTS AND DISCUSSION

6.1 A 3825 HZ BANDPASS ELLIPTIC FILTER (LEE)

The first filter is a 3825 Hz, sixth-order bandpass filter. It was designed by MPR by the techniques of Lee. The filter prototype is shown in Fig. 6.1, and the equivalent switched capacitor filter is shown in Fig. 6.2. The SWITCAP file for this filter is shown in Fig. 6.3. The filter1.in file (Fig. 6.4) is an edited copy of the SWITCAP file, having only the input stage elements.

The SWITCAP file was run through the CIRCE program resulting in the filter1.nd file (Fig. 6.5) and the filter1.ct file (Fig. 6.6). The filter1.com file is shown in Fig. 6.7. The capacitor array height is set at 13 plates, and the arrayswitch variable is set to 20. The layout is plotted in Fig. 6.8. To simplify the wiring above the operational amplifiers the filter1.ct file was slightly altered into that of Fig. 6.9. This results in the layout of Fig. 6.10 which has simpler connections to the operational amplifiers.

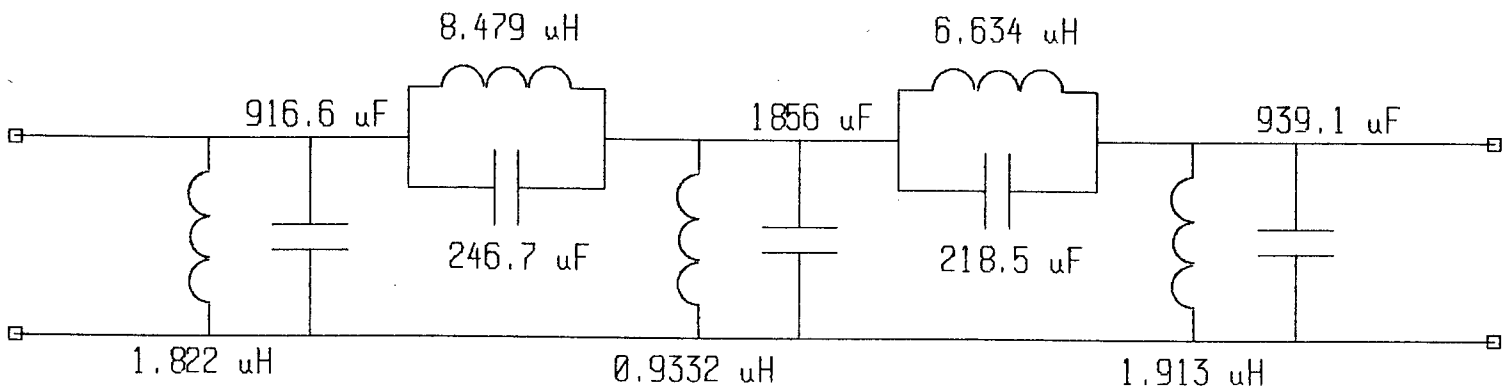


Fig. 6.1: LC prototype of filter1

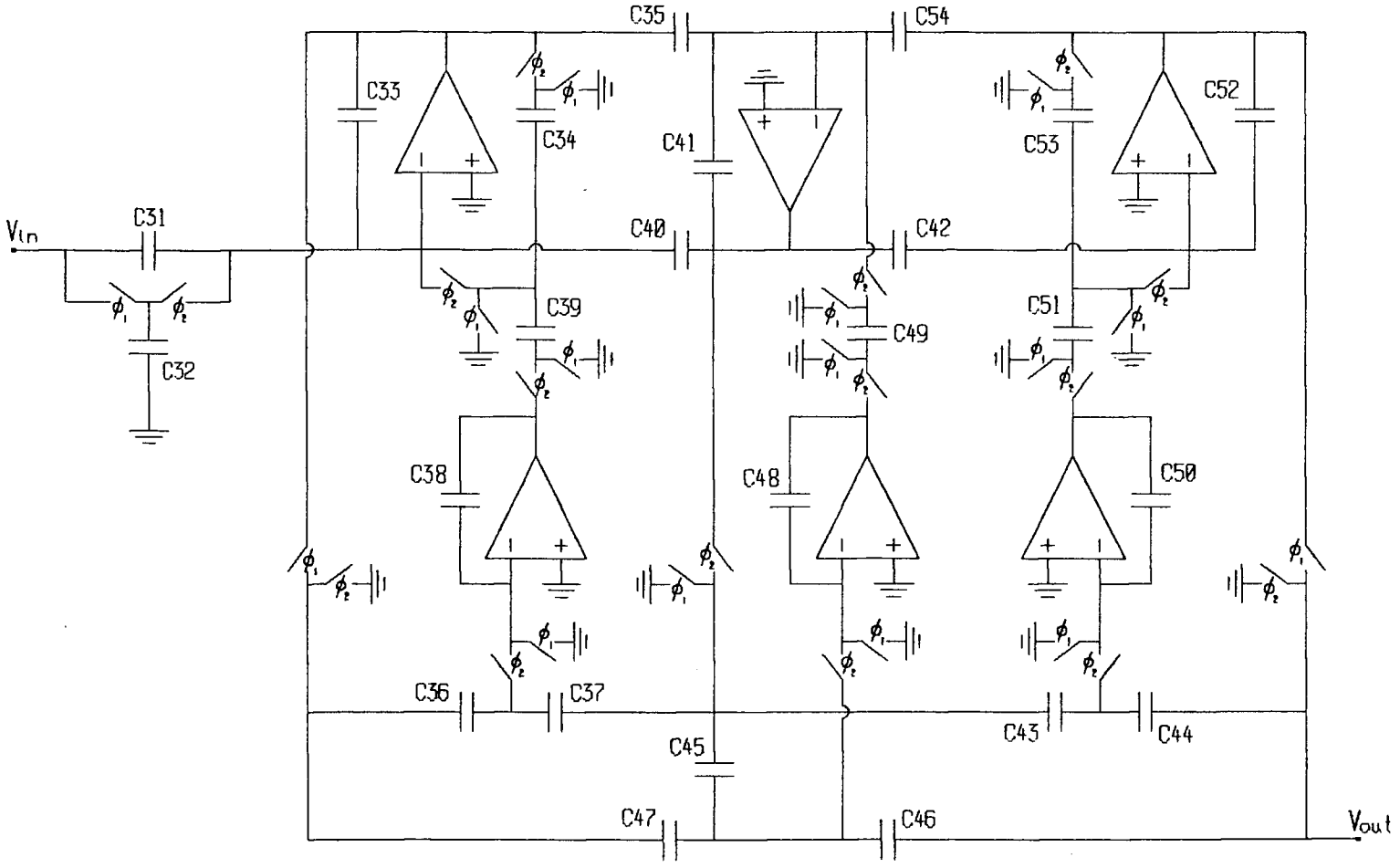


Fig. 6.2: SC implementation of filter1

```
OPTIONS ; GRID ; END ;
TITLE: EBP 3825 HZ FILTER (SOURCE: MPR) ;

TIMING ;  /* timing section */
   PERIOD 4.166667E-5 ;
   CLOCK C 1 (0 1/2) ;
   END ;

CIRCUIT ;  /* network section */
   /* SWITCHES */
   S01 (1 4) #CLK ;
   S02 (2 4) CLK ;
   S03 (2 5) CLK ;
   S04 (5 0) #CLK ;
   S05 (3 6) CLK ;
   S06 (6 0) #CLK ;
   S07 (3 7) #CLK ;
   S08 (7 0) CLK ;
   S09 (9 15) #CLK ;
   S10 (9 0) CLK ;
   S11 (8 10) CLK ;
   S12 (8 0) #CLK ;
   S13 (11 12) CLK ;
   S14 (12 0) #CLK ;
   S15 (2 13) CLK ;
   S16 (13 0) #CLK ;
   S17 (3 16) #CLK ;
   S18 (16 0) CLK ;
   S19 (15 17) #CLK ;
   S20 (17 0) CLK ;
   S21 (18 32) #CLK ;
   S22 (18 0) CLK ;
   S23 (15 19) #CLK ;
   S24 (19 0) CLK ;
   S25 (20 32) #CLK ;
   S26 (20 0) CLK ;
   S27 (21 22) CLK ;
   S28 (21 0) #CLK ;
   S29 (23 24) CLK ;
   S30 (24 0) #CLK ;
   S31 (14 25) CLK ;
   S32 (25 0) #CLK ;
   S33 (26 27) CLK ;
   S34 (26 0) #CLK ;
   S35 (28 29) CLK ;
   S36 (29 0) #CLK ;
   S37 (30 31) CLK ;
   S38 (30 0) #CLK ;
   S39 (31 33) CLK ;
   S40 (33 0) #CLK ;
```

Fig. 6.3: SWITCAP filter1 file

```
   S41 (32 34) CLK ;
   S42 (34 0) #CLK ;

/* CAPACITORS */
   C31 (1 2) 1.0 ;
   C32 (4 0) 4.4220 ;
   C33 (2 3) 76.4617 ;
   C34 (5 6) 2.3438 ;
   C35 (3 14) 1.1135 ;
   C36 (7 8) 5.5920 ;
   C37 (8 9) 1.0 ;
   C38 (10 11) 6.3792 ;
   C39 (12 13) 81.2592 ;
   C40 (2 15) 16.0117 ;
   C41 (14 15) 11.0305 ;
   C42 (15 31) 7.0644 ;
   C43 (19 26) 1.0 ;
   C44 (20 26) 4.1181 ;
   C45 (17 21) 11.4885 ;
   C46 (18 21) 1.1911 ;
   C47 (16 21) 1.0 ;
   C48 (22 23) 11.8053 ;
   C49 (24 25) 10.4569 ;
   C50 (27 28) 4.5596 ;
   C51 (29 30) 33.9825 ;
   C52 (31 32) 32.5742 ;
   C53 (33 34) 1.0 ;
   C54 (14 32) 1.0 ;

/* VOLTAGE-CONTROLLED VOLTAGE SOURCES */
   E2 (11 0 0 10) 1000 ;
   E3 (23 0 0 22) 1000 ;
   E4 (15 0 0 14) 1000 ;
   E1 (3 0 0 2) 1000 ;
   E5 (28 0 0 27) 1000 ;
   E6 (32 0 0 31) 1000 ;

/* INDEPENDENT VOLTAGE SOURCES */
   V1 (1 0) 1;
   END ;

ANALYZE SSS ;  /* required analysis */
   INFREQ 3000 4600 LIN 41 ;
   SET V1 AC 1.0 0.0 ;
   SAMPLE INPUT HOLD 1 1/2+ ;
   SAMPLE OUTPUT HOLD 1 1/2- ;
   PLOT VDB (32) (-6 -96 7) ;
   END ;

END ;
```

Fig. 6.3: SWITCAP filter1 file (cont'd)

```
S01 (1 4) #CLK ;
S02 (2 4) CLK ;
C31 (1 2) 1.0 ;
C32 (4 0) 4.4220 ;
```

Fig. 6.4: Filter1.in file

```
1
32

4

2
3
10
11
22
23
14
15
27
28
31
32
```

Fig. 6.5: Filter1.nd file

```
BS (1 4) #CLK ;
BS (2 4) CLK ;
BC (1 2) 1.0 ;
BC (4 0) 4.4220 ;

OA (3 0 0 2) 1000 ;
IC (2 3) 76.4617 ;
SC (2 3) 2.3438 CLK #CLK CLK #CLK ;
SC (2 11) 81.2592 CLK #CLK CLK #CLK ;
CC (2 15) 16.0117 ;

OA (11 0 0 10) 1000 ;
SC (10 3) 5.5920 CLK #CLK #CLK CLK ;
IC (10 11) 6.3792 ;
SC (10 15) 1.0 CLK #CLK #CLK CLK ;

OA (23 0 0 22) 1000 ;
SC (22 3) 1.0 CLK #CLK #CLK CLK ;
IC (22 23) 11.8053 ;
SC (22 15) 11.4885 CLK #CLK #CLK CLK ;
SC (22 32) 1.1911 CLK #CLK #CLK CLK ;

OA (15 0 0 14) 1000 ;
CC (3 14) 1.1135 ;
SC (14 23) 10.4569 CLK #CLK CLK #CLK ;
IC (14 15) 11.0305 ;
CC (14 32) 1.0 ;

OA (28 0 0 27) 1000 ;
SC (27 15) 1.0 CLK #CLK #CLK CLK ;
IC (27 28) 4.5596 ;
SC (27 32) 4.1181 CLK #CLK #CLK CLK ;

OA (32 0 0 31) 1000 ;
CC (15 31) 7.0644 ;
SC (31 28) 33.9825 CLK #CLK CLK #CLK ;
IC (31 32) 32.5742 ;
SC (31 32) 1.0 CLK #CLK CLK #CLK ;
```

Fig. 6.6: Filter1.ct file

```
  2.0  13  20
 -3 -4
114 79 79
 13 21 95
  4 70 75
  4  2  4
```

Fig. 6.7: Filter1.com file

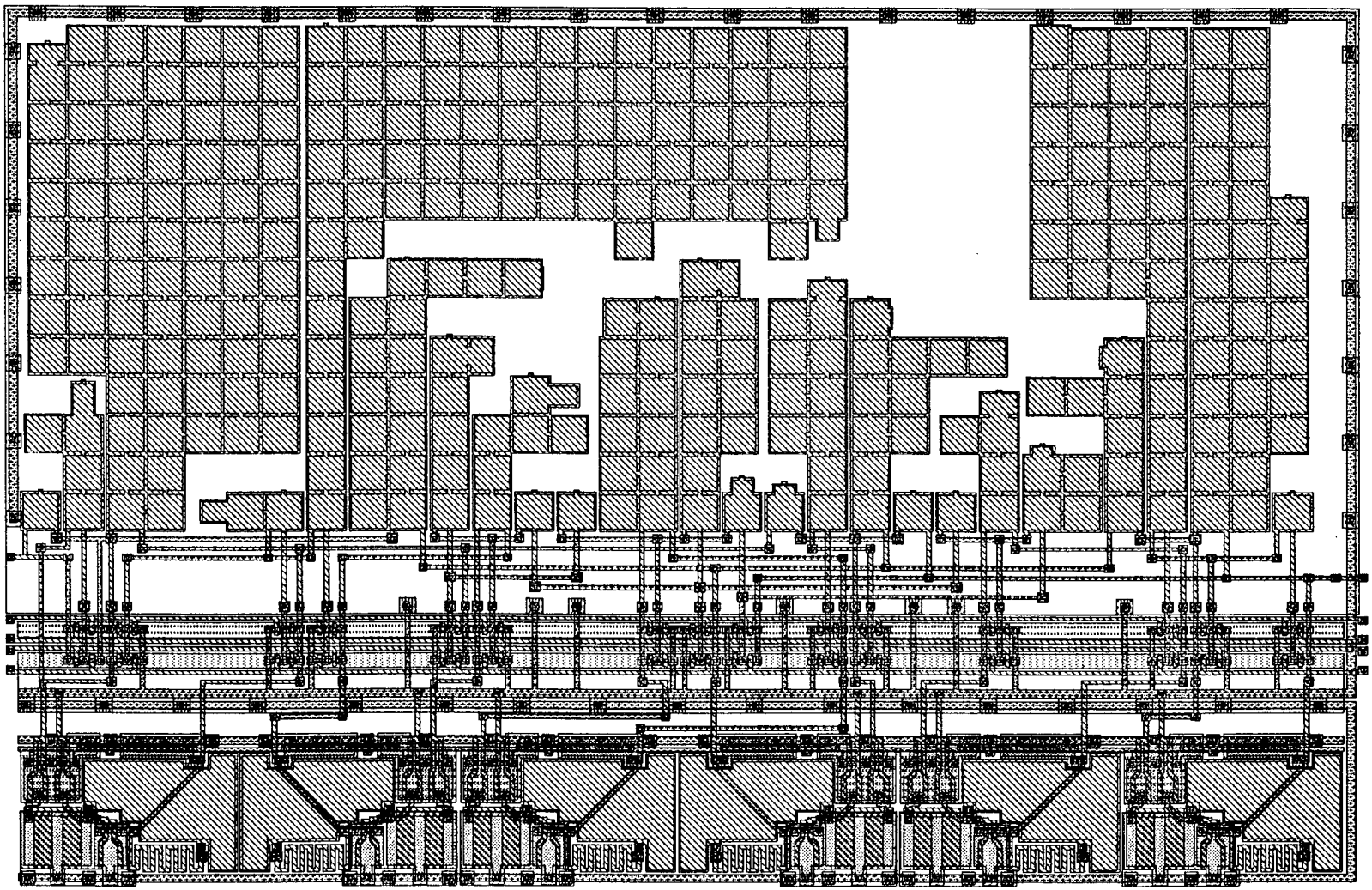


Fig. 6.8: Filter1 layout

```
BS (1 4) #CLK ;
BS (2 4) CLK ;
BC (1 2) 1.0 ;
BC (4 0) 4.4220 ;

OA (3 0 0 2) 1000 ;
SC (2 3) 2.3438 CLK #CLK CLK #CLK ;
IC (2 3) 76.4617 ;
SC (2 11) 81.2592 CLK #CLK CLK #CLK ;

OA (11 0 0 10) 1000 ;
SC (10 3) 5.5920 CLK #CLK #CLK CLK ;
IC (10 11) 6.3792 ;
CC (2 15) 16.0117 ;
SC (14 23) 10.4569 CLK #CLK CLK #CLK ;
SC (10 15) 1.0 CLK #CLK #CLK CLK ;

OA (23 0 0 22) 1000 ;
SC (22 3) 1.0 CLK #CLK #CLK CLK ;
IC (22 23) 11.8053 ;
SC (22 15) 11.4885 CLK #CLK #CLK CLK ;
SC (22 32) 1.1911 CLK #CLK #CLK CLK ;

OA (15 0 0 14) 1000 ;
CC (3 14) 1.1135 ;
IC (14 15) 11.0305 ;
CC (14 32) 1.0 ;

OA (28 0 0 27) 1000 ;
SC (27 15) 1.0 CLK #CLK #CLK CLK ;
IC (27 28) 4.5596 ;
SC (31 28) 33.9825 CLK #CLK CLK #CLK ;
SC (27 32) 4.1181 CLK #CLK #CLK CLK ;

OA (32 0 0 31) 1000 ;
CC (15 31) 7.0644 ;
IC (31 32) 32.5742 ;
SC (31 32) 1.0 CLK #CLK CLK #CLK ;
```

Fig. 6.9: Revised filter1.ct file

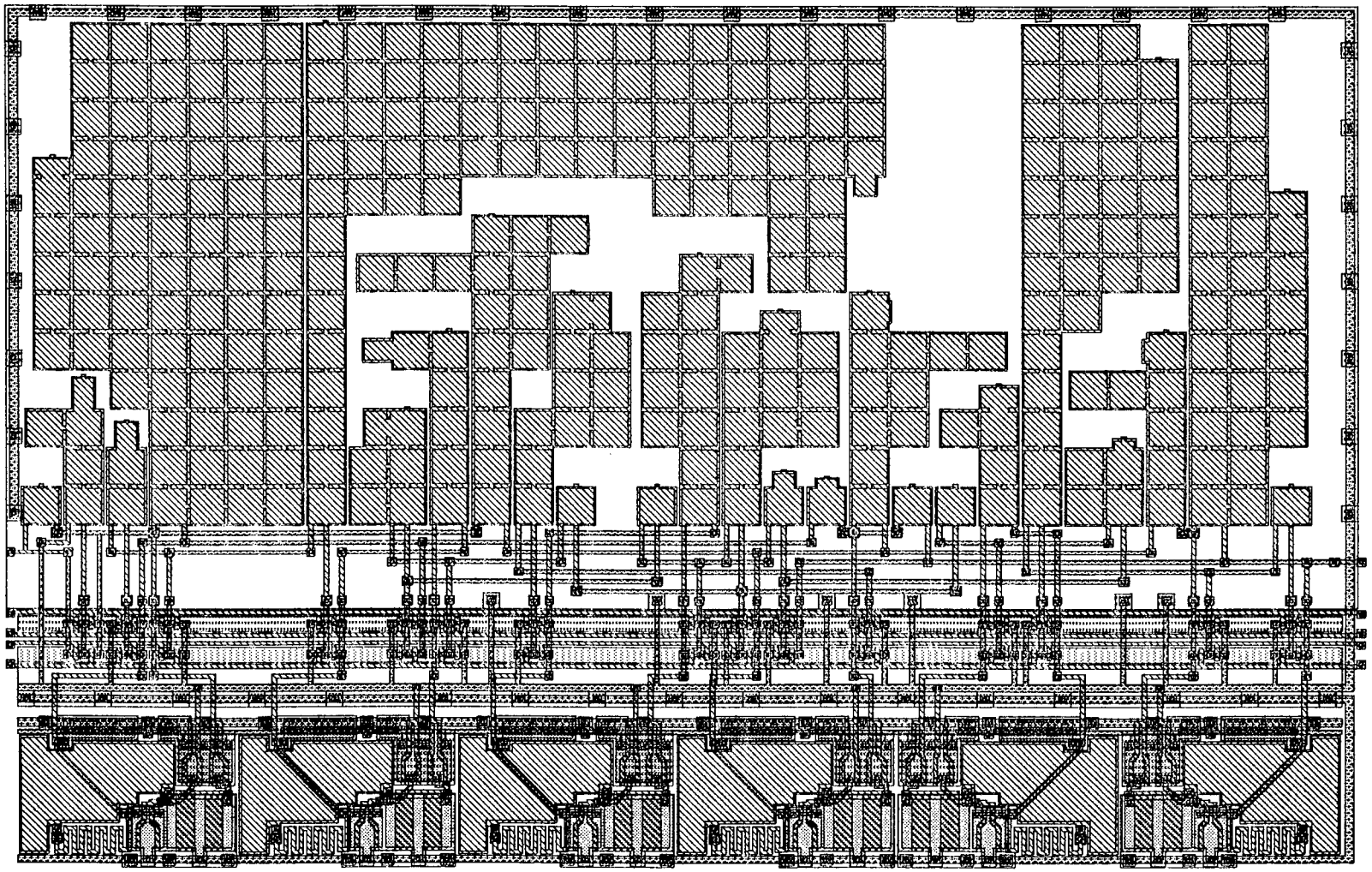


Fig. 6.10: Revised filter1 layout

6.2 A 500 HZ LOWPASS ELLIPTIC FILTER (MARTIN)

The second filter is a 500 Hz, seventh-order lowpass filter. It was designed by MPR by the techniques of Martin. The filter prototype is shown in Fig. 6.11, and the equivalent switched capacitor filter is shown in Fig. 6.12. The SWITCAP file for this filter is shown in Fig. 6.13. The filter2.in file is empty.

The SWITCAP file was run through the CIRCE program resulting in the filter2.nd file (Fig. 6.14) and the filter2.ct file (Fig. 6.15). The filter2.com file is shown in Fig. 6.16. The capacitor array height is set at 8 plates, and the arrayswitch variable is set to 10. The layout is plotted in Fig. 6.17.

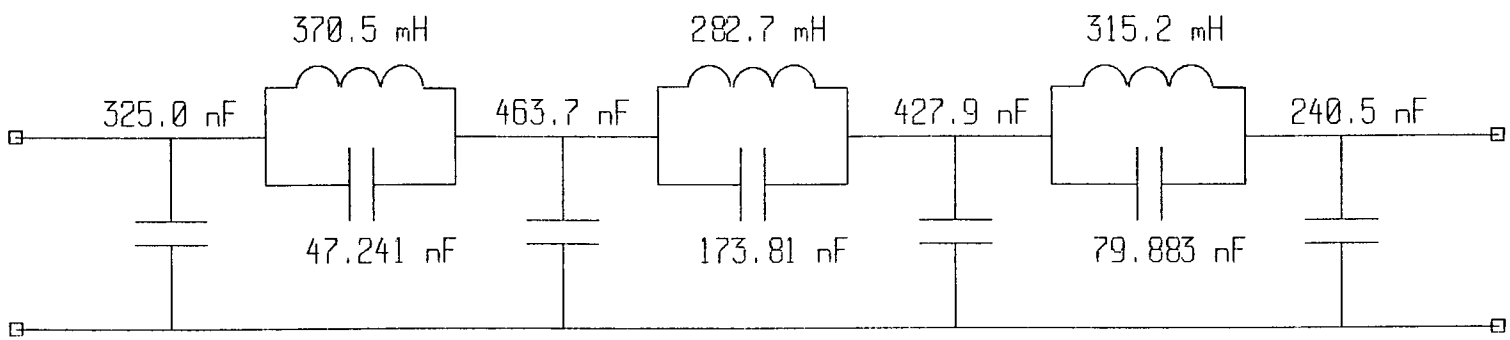


Fig. 6.11: LC prototype of filter2

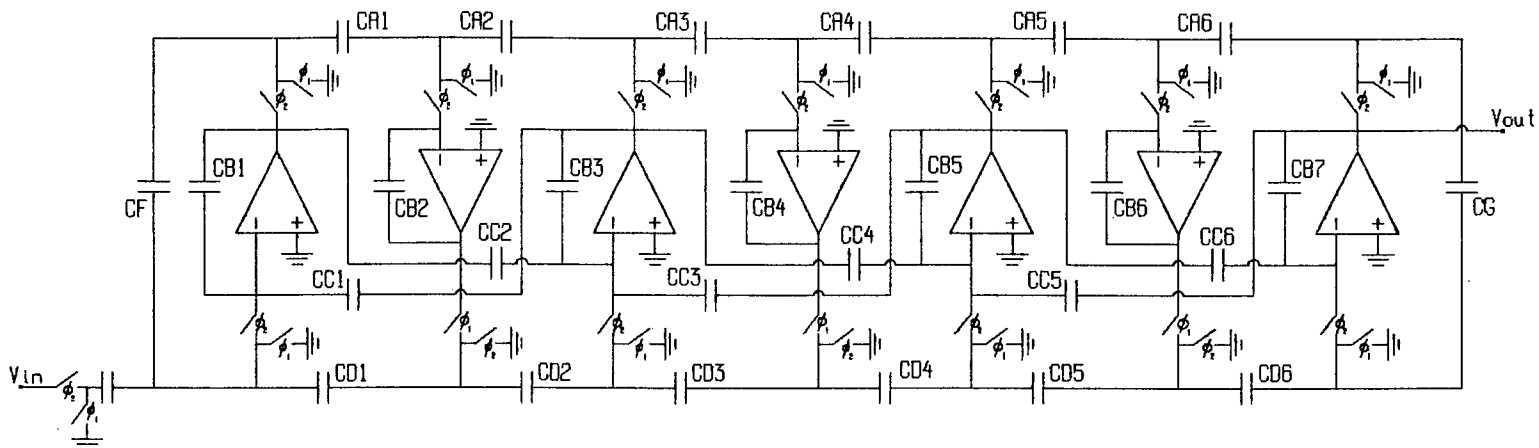


Fig. 6.12: SC implementation of filter2

```
OPTIONS ; GRID ; END ;
TITLE: LP 500 HZ FILTER (SOURCE: MPR) ;

TIMING ;  /* timing section */
   PERIOD 2.66041667E-05 ;
   CLOCK CLK 1(0 1/2) ;
   END ;

CIRCUIT ;  /* network section */
   /* SWITCHES */
   S01 (1 2) #CLK ;
   S02 (2 0) CLK ;
   S15 (11 0) CLK ;
   S16 (10 11) #CLK ;
   S18 (13 0) CLK ;
   S17 (12 13) #CLK ;
   S25 (21 0) CLK ;
   S26 (21 22) #CLK ;
   S27 (20 23) CLK ;
   S28 (23 0) #CLK ;
   S35 (31 0) CLK ;
   S36 (30 31) #CLK ;
   S37 (32 33) #CLK ;
   S38 (33 0) CLK ;
   S45 (41 0) CLK ;
   S46 (41 42) #CLK ;
   S47 (40 43) CLK ;
   S48 (43 0) #CLK ;
   S55 (51 0) CLK ;

   S56 (50 51) #CLK ;
   S57 (52 53) #CLK ;
   S58 (53 0) CLK ;
   S65 (61 0) CLK ;
   S66 (61 62) #CLK ;
   S67 (60 63) CLK ;
   S68 (63 0) #CLK ;
   S75 (71 0) CLK ;
   S76 (70 71) #CLK ;
   S77 (72 73) #CLK ;
   S78 (73 0) CLK ;

/* CAPACITORS */
   CA1 (11 21) 1.960 ;
   CA2 (21 31) 1.000 ;
   CA3 (31 41) 2.500 ;
   CA4 (41 51) 1.000 ;
   CA5 (51 61) 3.333 ;
   CA6 (61 71) 1.000 ;
   CB1 (12 10) 15.444 ;
```

Fig. 6.13: SWITCAP filter2 file

```
   CB2 (22 20) 28.169 ;
   CB3 (32 30) 22.134 ;
   CB4 (42 40) 32.237 ;
   CB5 (52 50) 30.845 ;
   CB6 (62 60) 34.235 ;
   CB7 (72 70) 12.303 ;
   CC1 (12 30) 1.000 ;
   CC2 (10 32) 2.992 ;
   CC3 (32 50) 2.247 ;
   CC4 (30 52) 19.667 ;
   CC5 (52 70) 1.085 ;
   CC6 (50 72) 10.234 ;
   CD1 (13 23) 1.091 ;
   CD2 (23 33) 1.666 ;
   CD3 (33 43) 1.000 ;
   CD4 (43 53) 3.501 ;
   CD5 (53 63) 1.000 ;
   CD6 (63 73) 2.830 ;
   CE ( 2 13) 2.162 ;
   CF (11 13) 1.080 ;
   CG (71 73) 1.000 ;

/* VOLTAGE-CONTROLLED VOLTAGE SOURCES */
   E02 (20 0 0 22) 1000 ;
   E05 (50 0 0 52) 1000 ;
   E03 (30 0 0 32) 1000 ;
   E04 (40 0 0 42) 1000 ;
   E06 (60 0 0 62) 1000 ;
   E01 (10 0 0 12) 1000 ;
   E07 (70 0 0 72) 1000 ;

/* INDEPENDENT VOLTAGE SOURCES */
   V1 (1 0) ;
   END ;

ANALYZE SSS ;  /* required analysis */
   INFREQ 3000 4600 LIN 41 ;
   SET V1 AC 1.0 0.0 ;
   SAMPLE INPUT HOLD 1 1/2+ ;
   SAMPLE OUTPUT HOLD 1 1/2- ;
   PLOT VDB (70) (-6 -96 7) ;
   END ;

END ;
```

Fig. 6.13: SWITCAP filter2 file (cont'd)

```
1
70

1

12
10
22
20
32
30
42
40
52
50
62
60
72
70
```

Fig. 6.14: Filter2.nd file

```
OA (10 0 0 12) 1000;
SC (12 1) 2.162 #CLK CLK #CLK CLK;
IC (12 10) 15.444;
SC (12 10) 1.080 #CLK CLK #CLK CLK;
SC (12 20) 1.091 #CLK CLK CLK #CLK;
CC (12 30)  1.000;

OA (20 0 0 22) 1000;
SC (22 10) 1.960 #CLK CLK #CLK CLK;
IC (22 20) 28.169;
SC (22 30) 1.000 #CLK CLK #CLK CLK;

OA (30 0 0 32) 1000;
CC (10 32)  2.992;
SC (32 20) 1.666 #CLK CLK CLK #CLK;
IC (32 30) 22.134;
SC (32 40) 1.000 #CLK CLK CLK #CLK;
CC (32 50)  2.247;

OA (40 0 0 42) 1000;
SC (42 30) 2.500 #CLK CLK #CLK CLK;
IC (42 40) 32.237;
SC (42 50) 1.000 #CLK CLK #CLK CLK;

OA (50 0 0 52) 1000;
CC (30 52) 19.667;
SC (52 40) 3.501 #CLK CLK CLK #CLK;
IC (52 50) 30.845;
SC (52 60) 1.000 #CLK CLK CLK #CLK;
CC (52 70)  1.085;

OA (60 0 0 62) 1000;
SC (62 50) 3.333 #CLK CLK #CLK CLK;
IC (62 60) 34.235;
SC (62 70) 1.000 #CLK CLK #CLK CLK;

OA (70 0 0 72) 1000;
CC (50 72) 10.234;
SC (72 60) 2.830 #CLK CLK CLK #CLK;
IC (72 70) 12.303;
SC (72 70) 1.000 #CLK CLK #CLK CLK;
```

Fig. 6.15: Filter2.ct file

```
  2.0  8  10
 -3 -4
114 79 79
 13 21 95
  4 70 75
  4  2  4
```

Fig. 6.16: Filter2.com file

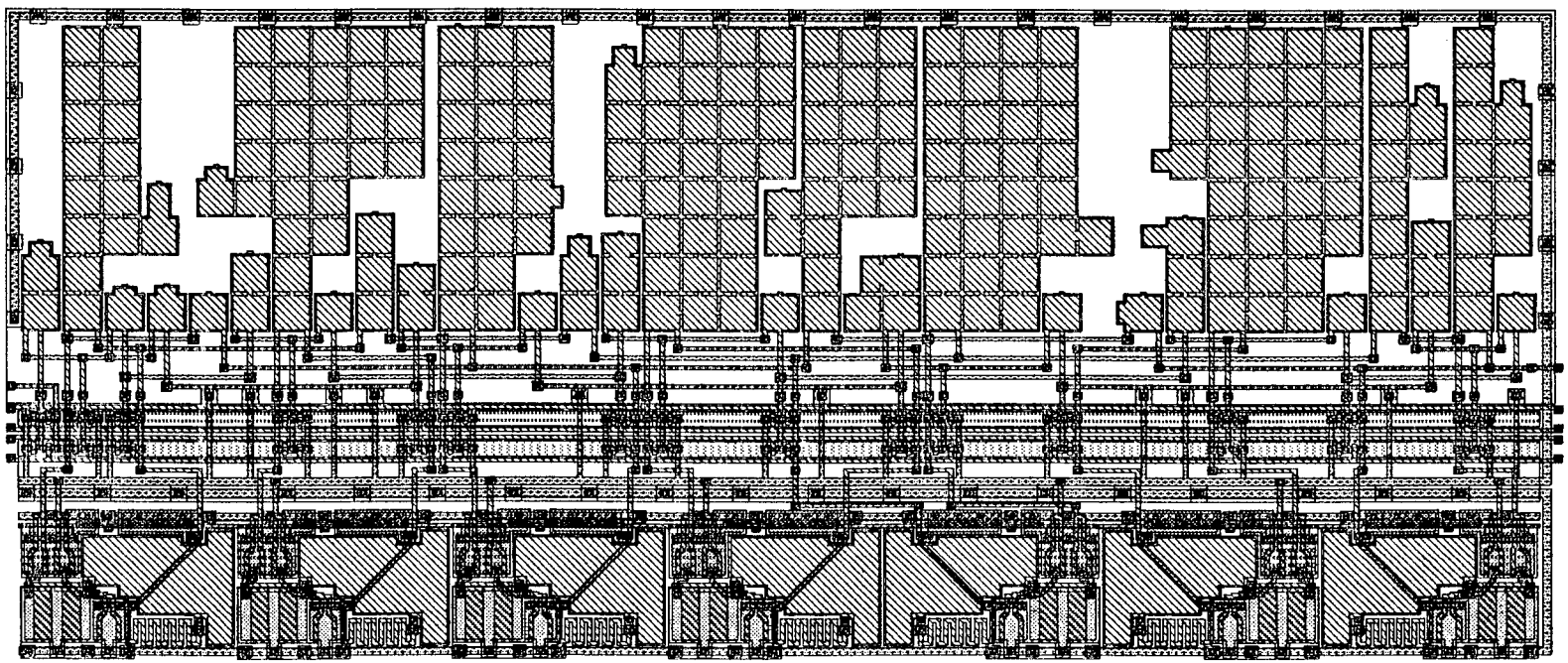


Fig. 6.17: Filter2 layout

6.3 A 3 KHZ - 5 KHZ BAND-ELIMINATION ELLIPTIC FILTER (AROMA)

The third filter is a 3 kHz - 5 kHz, eigth-order band-elimination filter. It was designed by the AROMA program. The switched capacitor filter is shown in Fig. 6.18. The SWITCAP file for this filter is shown in Fig. 6.19. The filter3.in file is empty.

The SWITCAP file was run through the CIRCE program resulting in the filter3.nd file (Fig. 6.20) and the filter3.ct file (Fig. 6.21). The filter3.com file is shown in Fig. 6.22. The capacitor array height is set at 8 plates, and the arrayswitch variable is set to 6. To simplify the capacitor wiring, the twenty-first and twenty-second capacitors were reversed. This removes one wiring level. The final layout is plotted in Fig. 6.23.

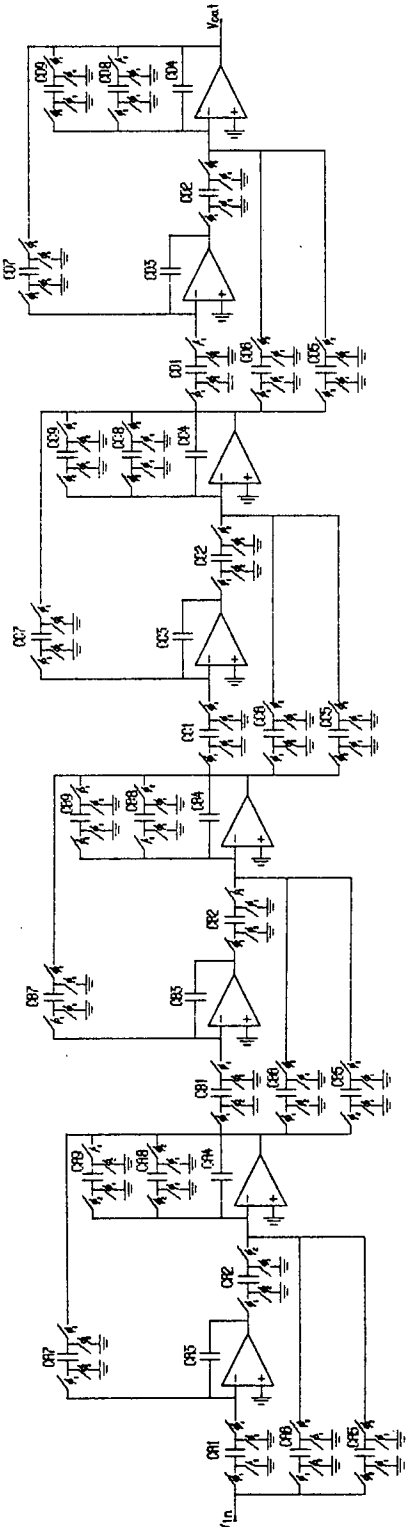


Fig. 6.18: SC implementation of filter3

```
OPTIONS ; GRID ; END ;
TITLE: EBE 3.3 kHZ - 4.5 kHZ FILTER (SOURCE: AROMA) ;

TIMING ;  /* timing section */
   PERIOD 4.166667E-5 ;
   CLOCK C 1 (0 1/2) ;
   END ;

/* switched capacitor sub-circuit */
SUBCKT (1 4) swc (K:phi1 K:phi2 K:phi3 K:phi4 P:cap1) ;
   S1 (1 2) phi1 ;
   S2 (2 0) phi2 ;
   S3 (3 4) phi3 ;
   S4 (3 0) phi4 ;
   C1 (2 3) cap1 ;
   END ;

CIRCUIT ;  /* network section */
   Xa (1 10) swc (C #C C #C 1.00) ;
   Xb (1 20) swc (C #C #C C 13.40) ;
   Xc (1 20) swc (#C C #C C 13.40) ;
   Xe (11 20) swc (C #C #C C 8.50) ;
   Xf (21 20) swc (C #C #C C 1.00) ;
   Xg (21 20) swc (#C C #C C 1.40) ;
   Xd (21 10) swc (C #C C #C 1.60) ;

   C (11 10) 3.30 ; C (21 20) 15.90 ;

   Xh (21 30) swc (C #C C #C 1.00) ;
   Xi (21 40) swc (C #C #C C 3.60) ;
   Xj (21 40) swc (#C C #C C 3.60) ;
   Xk (31 40) swc (C #C #C C 2.40) ;
   Xl (41 40) swc (C #C #C C 1.00) ;
   Xm (41 40) swc (#C C #C C 2.00) ;
   Xn (41 30) swc (C #C C #C 2.10) ;

   C (31 30) 4.00 ; C (41 40) 2.30 ;

   Xo (41 50) swc (C #C C #C 2.10) ;
   Xp (41 60) swc (C #C #C C 8.60) ;
   Xq (41 60) swc (#C C #C C 8.60) ;
   Xr (51 60) swc (C #C #C C 2.80) ;
   Xs (61 60) swc (C #C #C C 1.00) ;
   Xt (61 60) swc (#C C #C C 2.00) ;
   Xu (61 50) swc (C #C C #C 1.00) ;

   C (51 50) 5.50 ; C (61 60) 6.90 ;

   Xv (61 70) swc (C #C C #C 1.30) ;
   Xw (61 80) swc (C #C #C C 28.20) ;
```

Fig. 6.19: SWITCAP filter3 file

```
   Xx (61 80) swc (#C C #C C 28.20) ;
   Xy (71 80) swc (C #C #C C 8.40) ;
   Xz (81 80) swc (C #C #C C 1.00) ;
   X1 (81 80) swc (#C C #C C 1.40) ;
   X2 (81 70) swc (C #C C #C 1.10) ;

   C (71 70) 3.50 ; C (81 80) 26.90 ;

/* VOLTAGE-CONTROLLED VOLTAGE SOURCES */
   E1 (11 0 0 10) 1000 ;
   E3 (31 0 0 30) 1000 ;
   E4 (41 0 0 40) 1000 ;
   E6 (61 0 0 60) 1000 ;
   E5 (51 0 0 50) 1000 ;
   E7 (71 0 0 70) 1000 ;
   E2 (21 0 0 20) 1000 ;
   E8 (81 0 0 80) 1000 ;

/* INDEPENDENT VOLTAGE SOURCES */
   V1 (1 0) ;
   END ;

ANALYZE SSS ;  /* required analysis */
   INFREQ 1000 10000 LIN 40 ;
   SET V1 AC 1.0 0.0 ;
   SAMPLE INPUT HOLD 1 1/2+ ;
   SAMPLE OUTPUT HOLD 1 1/2- ;
   PLOT VDB (81) (-6 -96 7) ;
   END ;

END ;
```

Fig. 6.19: SWITCAP filter3 file (cont'd)

```
1
81

1

10
11
20
21
30
31
40
41
50
51
60
61
70
71
80
81
```

Fig. 6.20: Filter3.nd file

```
OA (11 0 0 10) 1000 ;
SC (10 1) 1.00 C #C C #C ;
IC (10 11) 3.30 ;
SC (10 21) 1.60 C #C C #C ;

OA (21 0 0 20) 1000 ;
SC (20 1) 13.40 #C C C #C ;
SC (20 1) 13.40 #C C #C C ;
SC (20 11) 8.50 #C C C #C ;
IC (20 21) 15.90 ;
SC (20 21) 1.00 #C C C #C ;
SC (20 21) 1.40 #C C #C C ;

OA (31 0 0 30) 1000 ;
SC (30 21) 1.00 C #C C #C ;
IC (30 31) 4.00 ;
SC (30 41) 2.10 C #C C #C ;

OA (41 0 0 40) 1000 ;
SC (40 21) 3.60 #C C C #C ;
SC (40 21) 3.60 #C C #C C ;
SC (40 31) 2.40 #C C C #C ;
IC (40 41) 2.30 ;
SC (40 41) 1.00 #C C C #C ;
SC (40 41) 2.00 #C C #C C ;

OA (51 0 0 50) 1000 ;
SC (50 41) 2.10 C #C C #C ;
IC (50 51) 5.50 ;
SC (50 61) 1.00 C #C C #C ;

OA (61 0 0 60) 1000 ;
SC (60 41) 8.60 #C C C #C ;
SC (60 41) 8.60 #C C #C C ;
SC (60 51) 2.80 #C C C #C ;
IC (60 61) 6.90 ;
SC (60 61) 1.00 #C C C #C ;
SC (60 61) 2.00 #C C #C C ;

OA (71 0 0 70) 1000 ;
SC (70 61) 1.30 C #C C #C ;
IC (70 71) 3.50 ;
SC (70 81) 1.10 C #C C #C ;
```

Fig. 6.21: Filter3.ct file

```
OA (81 0 0 80) 1000 ;
SC (80 61) 28.20 #C C C #C ;
SC (80 61) 28.20 #C C #C C ;
SC (80 71) 8.40 #C C C #C ;
IC (80 81) 26.90 ;
SC (80 81) 1.00 #C C C #C ;
SC (80 81) 1.40 #C C #C C ;
```

Fig. 6.21: Filter3.ct file (cont'd)

```
  2.0  8  6
 -3 -4
114 79 79
 13 21 95
  4 70 75
  4  2  4
```

Fig. 6.22: Filter3.com file

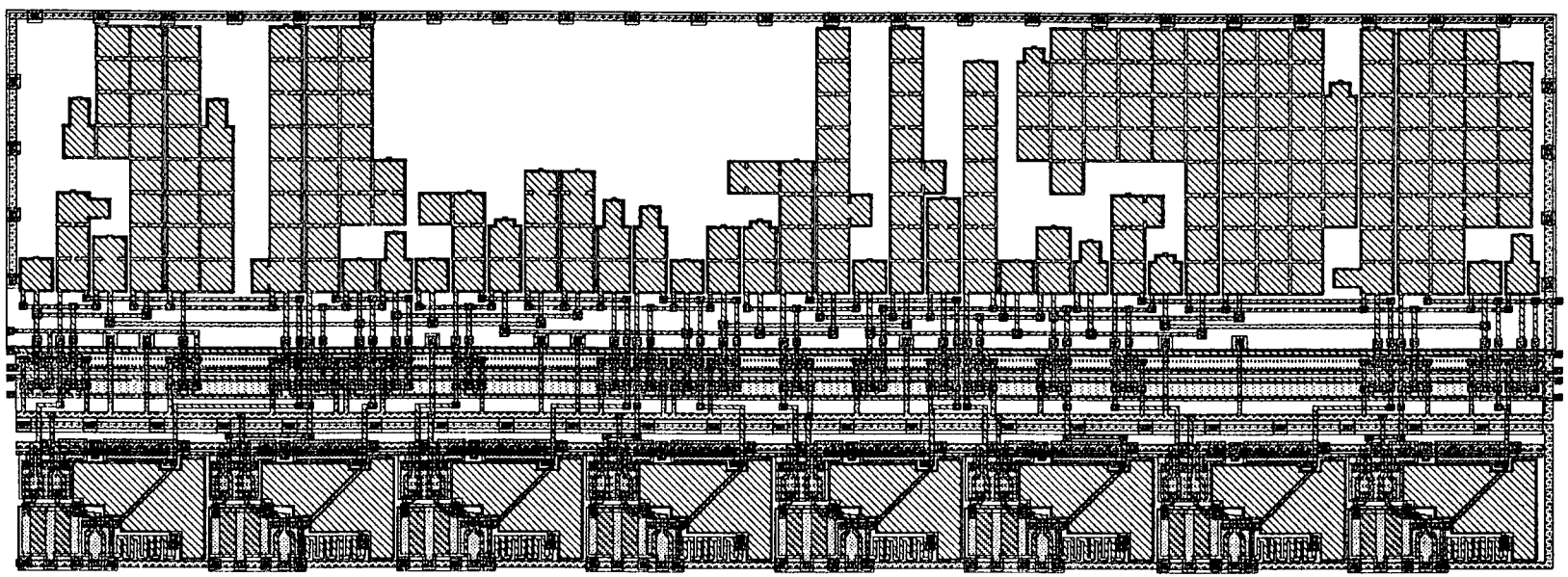


Fig. 6.23: Filter3 layout

6.4 DISCUSSION

The capacitor array usually takes up at least half the total layout area. If all the capacitor values are of the same general value, then the array will be very dense as each capacitor gets an equal share of the area (Fig. 6.17). If the filter has integrators with large Q factors at the beginning and end, large capacitor spreads result for these integrators. These capacitors will spread out, but an emptier area usually results in the middle of the capacitor array (Fig. 6.23). The array height is a very important variable for the minimization of the capacitor area. Each filter tends to have an optimum height. If the height is set larger than this, there will be excess space in the array. If the height is set too small, the filter will be longer, and space will be inserted between the operational amplifiers. The arrayswitch variable can be used to fine tune the layout. Sometimes it is necessary to ensure that a capacitor stays as low as possible, to allow a large capacitor more room on top. The capacitor plate size and the plate separation can be easily adjusted in the software to larger values.

The capacitor wiring works well. By keeping the switches directly below the capacitors the vertical connections are greatly simplified. A typical layout usually requires six or seven levels of wiring. It is usually possible to improve the wiring by one level by using more complex wiring, gaining 5λ.

The switch-pairs are always placed for full switch-sharing. The main drawback of this placement technique is the large number of crossings between the clock lines and the signal paths, causing noise. There tends to be some unnecessary contact in the wiring to the switches. These can be eliminated with a bit more sophisticated software.

The wiring to the operational amplifiers usually takes one wiring level. There has to be some separation between the operational amplifiers and the p-well which gives room for one wiring level. If there is more than one level of wiring here, the layout can usually be simplified by moving some of the capacitors in the filter.ct file. This also provides a means of shortening certain wiring connections if it is desired.

The p-wells have many substrate contacts, and the operational amplifier VDD line provides the substrate contacts for the pMOS transistors on the other side of the intervening p-well. These substrate connections work, however it would be desirable to have them closer to the p-transistors.

Overall the layout is very neat and elegant because it is very ordered. Modifications are easy to make by moving capacitors in the filter.ct file, and it is possible to change the wiring order by moving nodes in the filter.nd file. This should not be done unless one understands how the filter.nd file is arranged.

The width of the layout is minimum. The height of the layout tends to be some 5λ more than minimum. Overall the programs produce an excellent output. These layouts compare very well with filter layouts done by MPR. A filter layout can be generated in under three minutes using these programs, by someone who has never done a filter layout before.

7. CONCLUSION

Two programs have been developed to generate automatically the layout of strays insensitive switched capacitor filters with full switch-sharing and using standard-cell operational amplifiers. The first program (CIRCE), reformats a SWITCAP filter description into one suitable for a direct layout. The second program (SISCL), generates the layout of the filter in CIF.

These programs were tested by generating layouts of filter designs from Microtel Pacific Research Ltd. and from the AROMA program. The layouts seem satisfactory in terms of compactness and in flexibility of doing modifications. A chip is being fabricated to test the performance of a filter layout.

It is hoped that these programs will be developed into a useful design tool for the layout of strays insensitive switched capacitor filters.

7.1 FUTURE WORK

These layouts suffer from a number of undesirable effects. It would be better to keep the operational amplifiers further away from the switches and to remove all crossings of the analog and digital busses. Noise tends to be a very serious problem. The restriction of only allowing the one clock to feed charge into the operational amplifiers should also be removed.

One way of solving these problems is to move the operational amplifiers above the capacitor array and to connect them directly to the integrating capacitors. This will provide enough distance from the switches. The switches can be arranged into standard cells consisting of four switch-pairs. Two connect to the operational amplifier input and two connect to the output. The proposed cell of Fig. 7.1 also has no crossings of the analog signal paths and the digital clocks. These cells can be placed below the capacitor wiring.

The cell has six inputs/outputs. Four connect to the shared nodes of the switched capacitors and the middle two connect to the operational amplifier input and output nodes. If the integrating capacitor is placed directly above the

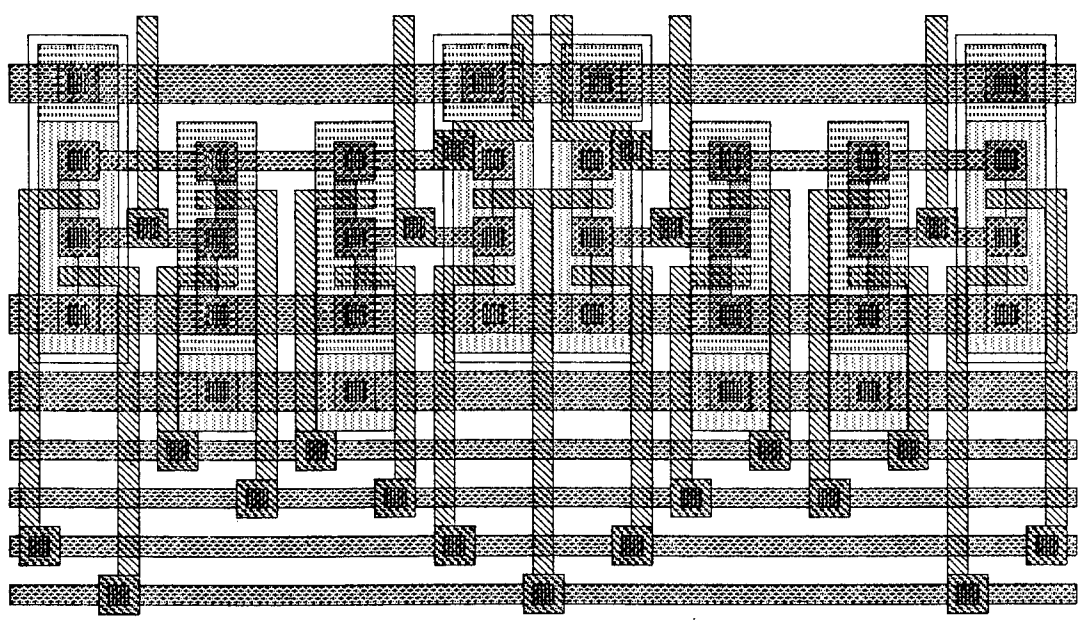


Fig. 7.1: Switch cell

switch cell, the connections will be very simple.

Another approach is to generate integrator layouts as standard cells, and then to connect them into the filter. Coupling capacitors and varying capacitor sizes can cause problems with this approach. There are two recent conference publications that use this approach [15][16].

REFERENCES

[1] J. T. Caves, M. A. Copeland, C. F. Rahim, and S. D. Rosenbaum, "Sampled Analog Filtering Using Switched Capacitors as Resistor Euivalents," *IEEE J. Solid-State Circuits*, vol. SC-12, no. 6, pp. 592-599, 1977.

[2] B. J. Hostica, R. W. Broderson, and P. R. Grey, "MOS Sampled Data Recursive Filters Using Switched Capacitor Integrators," *IEEE J. Solid-State Circuits*, vol. SC-12, no. 6, pp. 600-608, 1977.

[3] R. W. Broderson, P. R. Gray, and D. A. Hodges, "MOS Switched-Capacitor Filters," *Proc. IEEE*, vol. 67, no. 1, pp. 61-75, 1979.

[4] P. E. Fleischer, and K. R. Laker, "A Family of Active Switched Capacitor Biquad Building Blocks," *Bell System Technical Journal*, vol. 58, no. 10, pp. 2235-2269, 1979.

[5] K. Martin, and A. S. Sedra, "Exact Design of Switched-Capacitor Bandpass Filters Using Coupled-Biquad Structures," *IEEE Trans. Circuits Syst.*, vol. CAS-27, no. 6, pp. 469-474, 1980.

[6] K. Martin, "Improved Circuits for the Realization of Switched-Capacitor Filters," *IEEE Trans. Circuits Syst.*, vol. CAS-27, no. 4, pp. 237-244, 1980.

[7] M. S. Lee, G. C. Temes, C. Chang, and B. Ghaderi, "Bilinear Switched-Capacitor Ladder Filters," *IEEE Trans. Circuits Syst.*, vol. CAS-28, no. 8, pp. 811-821, 1981.

[8] R. B. Datar, and A. S. Sedra, "Exact Design of Strays-Insensitive Switched-Capacitor Ladder Filters," *IEEE Trans. Circuits Syst.*, vol. CAS-30, no. 12, pp. 888-897, 1983.

[9] E. Sánchez-Sinencio, and J. Ramírez-Angulo, "AROMA: An Area Optimized CAD Program for Cascade SC Filter Design," *IEEE Trans. Computer-Aided Design*, vol. CAD-4, no. 3, pp. 296-303, 1985.

[10] P. E. Allen, and E. Sánchez-Sinencio, "Switched-Capacitor Circuits," Van Nostrand Reinhold, New York, 1984.

[11] C. Mead, and L. Conway, "Introduction to VLSI Systems," Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.

[12] B. J. Sheu, and C. Hu, "Switch-Induced Error Voltage on a Switched Capacitor," *IEEE J. Solid-State Circuits*, vol. SC-19, no. 4, pp. 519-525, 1984.

[13] J. L. McCreary, "Matching Properties, and Voltage and Temperature Dependence of MOS Capacitors," *IEEE J. Solid-State Circuits*, vol. SC-16, no. 6, pp. 608-615, 1981.

[14] J. Shyu, G. C. Temes, and K. Yao, "Random Errors in MOS Capacitors," *IEEE J. Solid-State Circuits*, vol. SC-17, no. 6, pp. 1070-1075, 1982.

[15] T. Pletersek, J. Trontelj, L. Trontelj, I. Jones, G. Shenton, Y. Sun, "Analog LSI Design With CMOS Standard Cells," *Proc. CICC*, 1985, pp. 479-483.

[16] P. E. Allen, E. R. Macaluso, S. F. Bily, and A. Nedungadi, "AIDE2: An Automated Analog IC Design System," *Proc. CICC*, 1985, pp. 498-501.

[17] S. C. Fang, "SWITCAP User's Guide," Department of Electrical Engineering, Columbia University, October, 1982.

APPENDIX A: ISOCMOS DESIGN RULES

The following simplified design rules were used in the layout.

LAYOUT DESIGN TOLERANCES	MINIMUM VALUE
Device Active Areas	
Conductor width	2λ
Conductor Separation	2λ
N+ to P+ in same substrate	3λ
P+ active in N- substrate to p-well	5λ
N+ active in N- substrate to p-well	4λ
P-well	
Width	3λ
Overlap on active	2λ
Separation (different potentials)	8λ
Polysilicon1	
Conductor width	2λ
Conductor separation	2λ
Transistor channel overlap	2λ
N+/P+	
Overlap on active	2λ
Distance to P+/N+ active	2λ
Distance to P+/N+ channel gate	3λ
Polysilicon2	
Conductor width	2λ
Conductor separation	2λ
P1 overlap for precision capacitance	1λ

LAYOUT DESIGN TOLERANCES	MINIMUM VALUE
Cuts	
Size	2λ*2λ
Active overlap	1λ
P1 overlap	1λ
P2 overlap	1.5λ
Metal overlap	1λ
Separation from P1/P2	2λ
Metal	
Conductor width	2λ
Conductor separation	2λ
Passivation	
Distance inside pad edge	2λ

There should be a p-well substrate contact every 60λ. There should be an N- substrate contact every 120λ.

APPENDIX B: SWITCAP

SWITCAP is a general purpose simulation program for the analysis of switched capacitor networks developed at Columbia University [17]. Features include switching intervals, network elements and frequency domain analysis among others. The program uses a number of charge variables as the network variables for analysis.

A SWITCAP input file consists of a TIMING block specifying the clocks used, an optional SUBCKT block containing any subcircuit calls, a CIRCUIT block containing the network elements, and an ANALYZE block specifying the input voltage, the analysis desired and the output format.

The only network elements allowed are switches, capacitors, voltage-controlled voltage sources, and independent voltage sources. A network element statement follows the following general format:

<r> <element name> (<node list>) <element value> ;

where

<r> is a keycharacter indicating the element type. The keycharacter designations are S(switch), C(capacitor), E(voltage-controlled voltage source), and V(independent voltage source).

<element name> is a string of alphanumeric characters less

than seven characters long serving as a label for the element.

<node list> is a list of nodes to which the element is connected. A node name is a string of seven or less alphanumeric characters. The ground node is always 0 (the digit zero).

<element value> gives the "value" of the element that is appropriate to the element.

A switch may also be clocked by the complement of a clock defined in the TIMING block. This is done by adding the character # in front of the clock name for the switch. A sample SWITCAP input file of a switched capacitor integrator is shown in Fig. B.1.

```
OPTIONS; REPORT; CHKCLK; END;

TITLE: Sample SWITCAP file

/* timing section */
TIMING;
  PERIOD 1e-6;                  /* period of time base */
  CLOCK c 1 (0 3/8);            /* clock definition */
  END;

/* network section */
CIRCUIT;
  /* switches */
  S1 (1 2)  c;
  S2 (2 3) #c;

  /* capacitors */
  C1 (2 0) 0.100;
  C2 (1 3) 0.233;
  C3 (3 4) 0.952;

  /* voltage-controlled voltage sources */
  /* E1 (4 0 0 3) 10000;

  /* independent voltage source */
  V1 (1 0);       /* value given in analysis section */
  END;

/* analysis section */
ANALYZE SSS;     /* sinusoidal steady-state analysis */
  INFREQ 0.01 10000 LOG 12;   /* sweep in 12 steps */
  SET V1 AC 1.0 0.0;          /* amplitude and phase */
  PRINT VR(4) VI(4);          /* real and imaginary */
  PLOT VDB(4) VP(4);          /* magnitude and phase */
  END;

END;
```

Fig. B.1: Sample SWITCAP file

APPENDIX C: THE CIRCUIT EXTRACTOR LISTING

The circuit extractor requires the filename and filename.in files to run. It is run as follows:

CIRCE filename style <CR>

where style refers to one of the following design techniques: A(AROMA), L(Lee), M(Martin), and S(Datar & Sedra). The single character is used. It produces two output files called filename.nd and filename.ct.

```
#include <stdio.h>
#define  LENGTH    72              /* BUFFER LENGTH */
#define  WORD       8              /* IDENTIFIER LENGTH */
#define  SIZE      50              /* SIZE OF TABLE ARRAY */
#define  ZERO       0              /* ZERO CONSTANT */
#define  ONE        1              /* ONE CONSTANT */
#define  TWO        2              /* TWO CONSTANT */
#define  THREE      3              /* THREE CONSTANT */
#define  FOUR       4              /* FOUR CONSTANT */
#define  EOW      '\0'             /* END OF WORD */
#define  EOL      '\n'             /* END OF LINE */

char table[2][SIZE][WORD] ;        /* SUBSTITUTION TABLE */
char firstnode[WORD] ;             /* FILTER INPUT NODE */
char midnode[WORD] ;               /* FILTER LINK NODE */
char lastnode[WORD] ;              /* FILTER OUTPUT NODE */

char *psub = "SUBCKT" ;            /* SUBCKT POINTER */
char *pcir = "CIRCUIT" ;           /* CIRCUIT POINTER */
char *pana = "ANALYZE" ;           /* ANALYZE POINTER */
char *pend = "END" ;               /* END POINTER */
char *node = "!AA" ;               /* NEW NODE POINTER */
char *oa   = "OA (" ;              /* OPERATIONAL AMP */
char *bs   = "BS (" ;              /* BUFFER SWITCH */
char *bc   = "BC (" ;              /* BUFFER CAPACITOR */
char *sc   = "SC (" ;              /* SWITCHED CAPACITOR */
char *ic   = "IC (" ;              /* INTEGRATING CAPACITOR */
char *cc   = "CC (" ;              /* COUPLING CAPACITOR */

/*************************************************************/
/*                                                           */
/*  MAIN PROGRAM  --  CIRCUIT EXTRACTOR                      */
/*                                                           */
/*************************************************************/

main(argc,argv)                    /* FETCH C SHELL PARAMETERS */

int argc ;                         /* NUMBER OF ARGUMENTS */
char *argv[] ;                     /* ARGUMENT POINTERS */

{  char head ;                     /* INPUT SECTION STYLE */
   char *fp ;                      /* FILE POINTER */
   char st ;                       /* CHARACTER STORAGE */
   int i ;                         /* ITERATION VARIABLE */
   FILE *fdin ;                    /* .IN FILE DESCRIPTOR */
   FILE *fdct ;                    /* .CT FILE DESCRIPTOR */
   FILE *fdnd ;                    /* .ND FILE DESCRIPTOR */
   FILE *fdsw ;                    /* SWITCAP FILE DESCRIPTOR */
   FILE *fopen() ;                 /* FILE OPEN FUNCTION */
```

```
    /* STORE THE INPUT SECTION STYLE */
    st = *argv[2] ;
    head = ((st>='a' && st<='z') ? st-'a'+'A' : st) ;

    /* OPEN SWITCAP INPUT FILE FOR READING */
    fp = argv[1] ;
    fdsw = fopen(fp,"r") ;

    /* OPEN THE .IN FILE FOR READING */
    for (i=0; *(fp+i)!=EOW; i++) ;
    *(fp+i+0) = '.' ;
    *(fp+i+1) = 'i' ;
    *(fp+i+2) = 'n' ;
    *(fp+i+3) = EOW ;
    fdin = fopen(fp,"r") ;

    /* OPEN THE .CT FILE FOR WRITING */
    *(fp+i+1) = 'c' ;
    *(fp+i+2) = 't' ;
    fdct = fopen(fp,"w") ;

    /* OPEN THE .ND FILE FOR WRITING */
    *(fp+i+1) = 'n' ;
    *(fp+i+2) = 'd' ;
    fdnd = fopen(fp,"w") ;

    reduce(fdsw) ;          /* CLEAN UP THE INPUT FILE */
    replace() ;             /* SUBSTITUTE ALL SUBCKT CALLS */

    isolate(fdin) ;         /* ISOLATE INPUT SECTION */
    stages() ;              /* GROUP INTO INTEGRATORS */

    /* PROCESS THE APPROPRIATE INPUT SECTION */
    if (head=='L') input1(fdct) ;
    if (head=='M') input2(fdct) ;
    if (head=='S') input3(fdct) ;
    if (head=='A') input4(fdct) ;

    order1(fdnd) ;          /* SORT INTEGRATORS */
    order2(fdct) ;          /* SORT ELEMENTS AND NODES */

    fclose(fdin) ;          /* CLOSE .IN FILE */
    fclose(fdsw) ;          /* CLOSE SWITCAP FILE */
    fclose(fdnd) ;          /* CLOSE .ND FILE */
    fclose(fdct) ;          /* CLOSE .CT FILE */
}

/*************************************************************/
/*                                                           */
/*  REDUCE FUNCTION  --  Clean Up The Input File             */
```

```
/*                                                              */
/*    The reduce function removes any comments and blank        */
/*  lines in the SWITCAP input file. It also capitalizes        */
/*  all characters and ensures that there is only one           */
/*  statement per line. The filter input and output nodes       */
/*  are saved here.                                             */
/*                                                              */
/****************************************************************/

reduce(fdsw)

FILE *fdsw ;                            /* SWITCAP INPUT FILE */

{  char buf[LENGTH] ;                   /* LINE BUFFER */
   FILE *fdto ;                         /* OUTPUT FILE */
   FILE *fopen() ;                      /* FILE OPEN FUNCTION */

   fdto = fopen("SCINP","w") ;          /* OPEN OUTPUT FILE */

   /* READ LINES UNTIL A SUBCKT OR THE CIRCUIT SECTION */
   do { readclean(fdsw,buf) ;   }
   while (!same(buf,psub) && !same(buf,pcir)) ;

   /* WRITE LINES TO OUTPUT FILE UNTIL ANALYZE SECTION */
   /* THE INPUT NODE IS TAKEN FROM THE VOLTAGE SOURCE  */
   while (!same(buf,pana))
      { if (buf[0]!=EOL)
          if (buf[0]=='V')
            getnode(buf,ONE,firstnode) ;
          else writeline(fdto,buf) ;
        readclean(fdsw,buf) ;          }

   /* NOTE OUTPUT NODE FROM PRINT/PLOT STATEMENT */
   while (buf[0]!=EOF)
      { if (buf[0]=='P')
          getnode(buf,ONE,lastnode) ;
        readclean(fdsw,buf) ;     }

   fclose(fdto) ;                       /* CLOSE OUTPUT FILE */
}

/****************************************************************/
/*                                                              */
/*  REPLACE FUNCTION  --  Insert Subckts Into Circuit           */
/*                                                              */
/*    The replace function inserts any subcircuits into         */
/*  the CIRCUIT block.                                          */
/*                                                              */
/****************************************************************/
```

```
replace()

{  char buf[LENGTH] ;              /* LINE BUFFER */
   char subname[WORD] ;            /* SUBCIRCUIT NAME */
   char callname[WORD] ;           /* SUBCIRCUIT CALL NAME */
   int count = 0 ;                 /* COUNTING INTEGER */
   int ch ;                        /* CHARACTER STORAGE */
   FILE *fdfrom ;                  /* FILE BEING READ FROM */
   FILE *fdto ;                    /* FILE BEING WRITTEN TO */
   FILE *fdsub ;                   /* SUBCIRCUIT STORAGE FILE */
   FILE *fopen() ;                 /* FILE OPEN FUNCTION */

   fdfrom = fopen("SCINP","r") ;   /* OPEN INPUT FILE */
   fdto = fopen("SCINQ","w") ;     /* OPEN OUTPUT FILE */

   /* REPLACE THE SUBCIRCUITS ONE AT A TIME */
   /* UNTIL THERE ARE NO MORE SUBCKT BLOCKS */
   readline(fdfrom,buf) ;
   while (same(buf,psub))

      /* GET THE SUBCIRCUIT NAME */
      { getval(buf,subname) ;
        empty(ZERO) ;

        /* WRITE THE SUBCKT BLOCK TO A SPARE FILE WHILE */
        /* TABULATING ALL PARAMETERS INTO TABLE[0]      */
        fdsub = fopen("SUBC","w") ;
        while (!same(buf,pend))
           { writeline(fdsub,buf) ;
             tabulate(buf,ZERO) ;
             readline(fdfrom,buf) ;   }

        writeline(fdsub,buf) ;
        fclose(fdsub) ;

        /* SCAN THE REST OF THE FILE FOR SUBCIRCUIT CALLS */
        do { readline(fdfrom,buf) ;
             if (buf[0] == 'X')

               /* GET THE CALLED SUBCIRCUIT'S NAME */
               { getval(buf,callname) ;

                 /* COMPARE THE NAMES */
                 if (same(subname,callname))

                   /* IF THE SAME, TABULATE CALLING */
                   /* PARAMETERS INTO TABLE[1]      */
                   { empty(ONE) ;
                     tabulate(buf,ONE) ;
```

```
                        /* INSERT ANY ADDITIONAL NODES */
                        fill() ;

                        /* REPLACE THE CALL BY THE SUBCIRCUIT */
                        writesub(fdto) ;  }

                    else writeline(fdto,buf) ;    }
                else if (buf[0] != EOF)
                    writeline(fdto,buf) ;       }

            while (buf[0] != EOF) ;

            /* CLOSE INPUT AND OUTPUT FILES */
            fclose(fdto) ; fclose(fdfrom) ;

            /* REVERSE THE INPUT AND OUTPUT FILES */
            /* FOR THE NEXT PASS                  */
            if (count % 2)
              { fdfrom = fopen("SCINP","r") ;
                fdto = fopen("SCINQ","w") ;     }
            else { fdfrom = fopen("SCINQ","r") ;
                   fdto = fopen("SCINP","w") ;   }

            count++ ;
            readline(fdfrom,buf) ;          }

        /* CLOSE INPUT AND OUTPUT FILES */
        fclose(fdfrom) ; fclose(fdto) ;

        /* MAKE SURE THAT THE FINAL OUTPUT */
        /* FILE IS THE EXPECTED ONE        */
        if (!(count % 2))
          { fdfrom = fopen("SCINP","r") ;
            fdto = fopen("SCINQ","w") ;

            /* COPY INPUT FILE TO OUTPUT FILE */
            while ((ch=getc(fdfrom))!=EOF)
               putc(ch,fdto) ;

            fclose(fdfrom) ;
            fclose(fdto) ;           }
}

/*************************************************************/
/*                                                           */
/*  WRITESUB FUNCTION  --  Inserts Subcircuit                */
/*                                                           */
/*     The writesub function copies the subcircuit           */
/*  currently stored into the output file. The parameters    */
/*  are replaced by those of the calling statement.          */
```

```
/*                                                              */
/****************************************************************/

writesub(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  char lin[LENGTH] ;               /* SUBCIRCUIT LINE BUFFER */
   char buf[LENGTH] ;               /* LINE BUFFER FOR OUTPUT */
   char par[WORD] ;                 /* PARAMETER BUFFER */
   int a,b,i,j ;                    /* INDEX PARAMETERS */
   int locn ;                       /* INDEX TABLE LOCATION */
   int flag ;                       /* SEARCH SUCCESS FLAG */
   FILE *fdsub ;                    /* SUBCIRCUIT FILE */
   FILE *fopen() ;                  /* FILE OPEN FUNCTION */

   /* PROCESS EACH LINE IN THE SUBCIRCUIT FILE IN TURN */
   fdsub = fopen("SUBC","r") ;
   do { readline(fdsub,lin) ;

        /* DO NOT PROCESS HEADER AND END STATEMENTS */
        if (!same(lin,pend) && !same(lin,psub) && lin[0]!=EOF)

           /* COPY THE FIRST PART OF THE LINE */
           { a = b = 0 ;
             do {  buf[a++] = lin[b] ;  }
             while (lin[b++]!='(') ;

             /* REPLACE ALL PARAMETERS */
             do { while (lin[b]==' ')
                     buf[a++] = lin[b++] ;

                  /* STORE THE PARAMETER */
                  i = 0 ;
                  while (lin[b]!=' ' && lin[b]!=')' && lin[b]!=';')
                     par[i++] = lin[b++] ;
                  par[i] = EOW ;

                  /* LOCATE THE PARAMETER IN TABLE[0] */
                  if (i!=0)
                    { locn = 0 ; flag = 1 ;
                      do { if (same(par,&table[0][locn++][0]))
                             flag = 0 ;        }
                      while (table[0][locn-1][0]!=EOW && flag) ;

                      /* INSERT THE REPLACEMENT PARAMETER */
                      j = (-1) ;
                      while (table[1][locn-1][++j]!=EOW)
                         buf[a++] = table[1][locn-1][j] ;   }
```

```
                if (lin[b]==')' || lin[b]==';')
                  buf[a++] = lin[b++] ;    }

            while (lin[b-1]!=';' && i!=0) ;

            /* WRITE CONVERTED LINE TO OUTPUT FILE */
            buf[a] = EOL ;
            writeline(fdto,buf) ;    }     }

   while (lin[0]!=EOF) ;

   fclose(fdsub) ;            /* CLOSE SUBCIRCUIT FILE */
}

/*************************************************************/
/*                                                           */
/*  TABULATE FUNCTION  --  Fills Substitution Table          */
/*                                                           */
/*     The tabulate function stores all parameters in the    */
/*  input string in TABLE[]. When tabulating the lines in    */
/*  the subcircuit, no parameter is stored more than once.   */
/*                                                           */
/*************************************************************/

tabulate(buf,col)

char buf[LENGTH] ;                  /* INPUT LINE BUFFER */
int col ;                           /* TABULATION COLUMN */

{  char par[WORD] ;                 /* PARAMETER BUFFER */
   int i,q ;                        /* INDEXING VARIABLES */
   int set ;                        /* PARAMETER SET */

   /* PROCESS INPUT LINE */
   q = 0 ;
   for (set=0; set<2; set++)

      /* IGNORE DATA BEFORE FIRST BRACKET */
      { while (buf[q] != '(' && buf[q] != ';') q++ ;
        if (buf[q] == '(') q++ ;

        /* PLUCK OUT THE PARAMETERS */
        do { while (buf[q] == ' ')
                q++ ;

             /* MAKE ALLOWANCE FOR : IN SUBCKT LINE */
             if (set==1 && same(buf,psub))
               while (buf[q] != ':' && buf[q] != ';') q++ ;
             if (buf[q] == ':') q++ ;
```

```
            /* STORE THE PARAMETER */
            i = 0 ;
            while (buf[q]!=' ' && buf[q]!=')' && buf[q]!=';')
               par[i++] = buf[q++] ;
            par[i] = EOW ;

            /* INSERT PARAMETER INTO TABLE[] */
            if (buf[0] != ';')
              if (col==0 && !in(par,0) || col==1)
                insert(par,col) ;        }

        while (buf[q]!=')' && buf[q]!=';') ;        }
}

/*************************************************************/
/*                                                           */
/*  ISOLATE FUNCTION  --  Isolate Input Stage                */
/*                                                           */
/*     The isolate function cleans up the .in file, and      */
/*  removes any lines in the filter file that match one in   */
/*  the input section file. This separates the input         */
/*  section from the filter description.                     */
/*                                                           */
/*************************************************************/

isolate(fdin)

FILE *fdin ;                          /* .IN INPUT FILE */

{  char buf[LENGTH],lin[LENGTH] ;     /* LINE BUFFERS */
   char st1[WORD],st2[WORD] ;         /* STORAGE FOR NODES */
   char st3[WORD],st4[WORD] ;         /* STORAGE FOR NODES */
   int flag ;                         /* MATCHING FLAG */
   FILE *fdto ;                       /* OUTPUT FILE */
   FILE *fdf1 ;                       /* INPUT FILE1 */
   FILE *fdf2 ;                       /* INPUT FILE2 */
   FILE *fopen() ;                    /* FILE OPEN FUNCTION */

   /* EMPTY TABLE */
   empty(ZERO) ; empty(ONE) ;

   /* INITIALIZE NODAL INFORMATION IN TABLE */
   copy(firstnode,&table[1][0][0]) ;
   copy(lastnode,&table[1][1][0]) ;
   table[1][2][0] = EOL ;

   /* WRITE INPUT SECTION TO OUTPUT FILE, USING THE */
   /* SAME PROCESSING AS THE REDUCE FUNCTION DOES   */
   fdf1 = fopen("SUBC","w") ;
   readclean(fdin,buf) ;
```

```
   while (buf[0] != EOF)
      { if (buf[0] != EOL)
          writeline(fdf1,buf) ;
        readclean(fdin,buf) ;       }
   fclose(fdf1) ;

   fdf1 = fopen("SCINQ","r") ;  /* OPEN FILTER FILE */
   fdto = fopen("SCINP","w") ;  /* OPEN OUTPUT FILE */
   fdf2 = fopen("SUBC","r") ;   /* OPEN INPUT SECTION FILE */

   /* WRITE ALL LINES NOT EXISTING IN INPUT */
   /* SECTION FILE TO THE OUTPUT FILE       */
   readline(fdf1,buf) ;
   readline(fdf1,buf) ;
   while (!same(buf,pend))
      { flag = 1 ;

        /* READ ONE LINE FROM THE INPUT SECTION FILE */
        readline(fdf2,lin) ;
        while (flag && lin[0]!=EOF)

           /* CHECK TO SEE IF THEY ARE THE SAME TYPE */
           { if (buf[0]==lin[0])

               /* GET THE NODES FROM BOTH LINES */
               { getnode(buf,1,st1) ;
                 getnode(buf,2,st2) ;
                 getnode(lin,1,st3) ;
                 getnode(lin,2,st4) ;

                 /* IF NODES ARE THE SAME, LINES ARE EQUAL */
                 if (same(st1,st3) && same(st2,st4) ||
                     same(st1,st4) && same(st2,st3))
                   flag = 0 ;     }

             /* READ NEXT LINE FROM INPUT SECTION FILE */
             readline(fdf2,lin) ;  }

        /* IF NO MATCH WAS MADE, WRITE LINE TO OUTPUT FILE */
        if (flag)
          writeline(fdto,buf) ;

        /* PREPARE FOR THE NEXT LINE FROM THE FILTER FILE */
        rewind(fdf2) ;
        readline(fdf1,buf) ;    }

   fclose(fdf1) ;          /* CLOSE FILTER FILE */
   fclose(fdf2) ;          /* CLOSE INPUT SECTION FILE */
   fclose(fdto) ;          /* CLOSE OUTPUT FILE */
}
```

```
/*************************************************************/
/*                                                           */
/*  STAGES FUNCTION  --  Group Elements Into Stages          */
/*                                                           */
/*     The stages function groups the filter elements into   */
/*  groups based on which op-amp input they are connected    */
/*  to. SISC elements are collected into one element.        */
/*                                                           */
/*************************************************************/

stages()

{  char op1[WORD],op4[WORD] ;        /* OP-AMP NODES */
   char st1[WORD],st2[WORD] ;        /* ELEMENT NODES */
   char buf[LENGTH] ;                /* LINE BUFFER */
   long offset ;                     /* FILE MARKER */
   FILE *fdto ;                      /* OUTPUT FILE */
   FILE *fdfrom ;                    /* INPUT FILE */
   FILE *fopen() ;                   /* FILE OPEN FUNCTION */

   fdfrom = fopen("SCINP","r") ;     /* OPEN THE INPUT FILE */
   fdto = fopen("SCINQ","w") ;       /* OPEN THE OUTPUT FILE */

   /* THE GROUPING IS BASED ON THE OP-AMPS */
   readline(fdfrom,buf) ;
   while (buf[0]!=EOF)

      /* CHECK FOR AN OP-AMP ELEMENT */
      { if (buf[0]=='E' && !same(buf,pend))

          /* WRITE OP-AMP TO OUTPUT FILE */
          /* AND SAVE ITS NODES          */
          { wrout(fdto,buf,oa) ;
            getnode(buf,ONE,op1) ;
            getnode(buf,FOUR,op4) ;

            /* NOW SEARCH THE FILE FOR CONNECTING ELEMENTS */
            offset = ftell(fdfrom) ;
            rewind(fdfrom) ;
            readline(fdfrom,buf) ;
            while (buf[0]!=EOF)

               /* CAPACITORS MUST BE IC OR CC */
               { if (buf[0]=='C')
                    { getnode(buf,ONE,st1) ;
                      getnode(buf,TWO,st2) ;

                      /* IC IF CONNECTED ACROSS THE OP-AMP */
                      if (same(op4,st1) && same(op1,st2) ||
```

```
                        same(op1,st1) && same(op4,st2))
                      wrout(fdto,buf,ic) ;

                    /* ELSE IT MUST BE A CC */
                    else if (same(op4,st1) || same(op4,st2))
                      wrout(fdto,buf,cc) ;          }

                /* SWITCHES MUST BE PART OF SISC'S */
                else if (buf[0]=='S')
                  { getnode(buf,ONE,st1) ;
                    getnode(buf,TWO,st2) ;

                    /* COMPLETE THE SISC ELEMENT */
                    if (same(op4,st1))
                      sisc(fdfrom,fdto,buf,st2,st1) ;
                    if (same(op4,st2))
                      sisc(fdfrom,fdto,buf,st1,st2) ;     }

                readline(fdfrom,buf) ;       }

            /* SEPARATE THE BLOCKS WITH A BLANK LINE */
            putc(EOL,fdto) ;

            /* RETURN THE FILE TO ITS SAVED POSITION */
            fseek(fdfrom,offset,ZERO) ;      }
        readline(fdfrom,buf) ;           }

   /* CLOSE INPUT AND OUTPUT FILES */
   fclose(fdto) ; fclose(fdfrom) ;
}

/*************************************************************/
/*                                                           */
/*  SISC FUNCTION  --  Identify a Switched Capacitor         */
/*                                                           */
/*     The sisc function gathers the other four elements     */
/*  of a stray-insensitive switched capacitor belonging to   */
/*  the switch provided by the stages function.              */
/*                                                           */
/*************************************************************/

sisc(fdfrom,fdto,buf,con2,con1)

FILE *fdfrom ;                  /* INPUT FILE */
FILE *fdto ;                    /* OUTPUT FILE */
char buf[LENGTH] ;              /* LINE FOR SWITCH ELEMENT */
char con2[WORD] ;               /* SECOND NODE OF SWITCH */
char con1[WORD] ;               /* NODE #1 OF SISC ELEMENT */

{  char st1[WORD],st2[WORD] ;    /* NODE STORAGE STRINGS */
```

```
char con3[WORD],con4[WORD] ;  /* SISC ELEMENT HIDDEN NODES */
char clk[5][WORD] ;           /* SISC ELEMENT PARAMETERS */
long offset,offset2 ;         /* FILE POSITION MARKERS */

/* STORE THE FIRST CLOCK */
getval(buf,&clk[1][0]) ;

/* SAVE THE FILE POSITION AND REWIND THE FILE */
offset = ftell(fdfrom) ;
rewind(fdfrom) ;

/* SEARCH FOR THE REST OF THE ELEMENTS */
readline(fdfrom,buf) ;
while (buf[0]!=EOF)

   /* FIND THE CAPACITOR FIRST */
   { if (buf[0]=='C')
       { getnode(buf,ONE,st1) ;
         getnode(buf,TWO,st2) ;

         /* DOES IT CONNECT TO THE SWITCH NODE? */
         if (same(con2,st1) || same(con2,st2))

           /* SAVE THE CAPACITOR VALUE AND SAVE */
           /* THE OTHER NODE OF THE CAPACITOR   */
           { getval(buf,&clk[0][0]) ;
             if (same(con2,st1))
               getnode(buf,TWO,con3) ;
             else getnode(buf,ONE,con3) ;

             /* SAVE THE FILE POSITION */
             offset2 = ftell(fdfrom) ;
             rewind(fdfrom) ;

             /* NOW SEARCH FOR THE OTHER THREE SWITCHES */
             readline(fdfrom,buf) ;
             while (buf[0] != EOF)

                /* LOOK FOR A SWITCH ELEMENT */
                { if (buf[0] == 'S')
                    { getnode(buf,ONE,st1) ;
                      getnode(buf,TWO,st2) ;

                      /* CONNECTS INPUT SWITCH TO GROUND? */
                      if (same(con2,st1) && st2[0]=='0' ||
                          same(con2,st2) && st1[0]=='0')
                        getval(buf,&clk[2][0]) ;

                      /* CONNECTS OUTPUT SWITCH TO GROUND? */
                      else if (same(con3,st1) && st2[0]=='0' ||
```

```
                                        same(con3,st2) && st1[0]=='0')
                                      getval(buf,&clk[4][0]) ;

                                /* IS IT THE OUTPUT SWITCH? */
                                else if (same(con3,st1) || same(con3,st2))
                                  { getval(buf,&clk[3][0]) ;
                                    if (same(con3,st1))
                                      getnode(buf,TWO,con4) ;
                                    else getnode(buf,ONE,con4) ;   }    }

                           readline(fdfrom,buf) ;        }

                        /* WRITE GATHERED INFORMATION TO OUTPUT FILE */
                        scout(con1,con4,clk,fdto) ;

                        /* RESTORE FILE POSITION */
                        fseek(fdfrom,offset2,ZERO) ;    }     }

              readline(fdfrom,buf) ;     }

   /* RESTORE FILE POSITION */
   fseek(fdfrom,offset,ZERO) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT1 FUNCTION  --  Process Input Stage  (LEE)          */
/*                                                           */
/*     The input1 function processes the input stage of      */
/*  Man Shek Lee, consisting of two switches and two caps.   */
/*  The switches are listed first, followed by the input     */
/*  capacitor and then the switched capacitor.               */
/*                                                           */
/*************************************************************/

input1(fdct)

FILE *fdct ;                          /* .CT OUTPUT FILE */

{  char buf[3][LENGTH] ;              /* CAP LINE BUFFERS */
   char st1[WORD],st2[WORD] ;         /* NODE STORAGE STRINGS */
   char st3[WORD],st4[WORD] ;         /* NODE STORAGE STRINGS */
   int i = 0 ;                        /* INDEX VARIABLE */
   FILE *fdfrom ;                     /* INPUT FILE */
   FILE *fopen() ;                    /* FILE OPEN FUNCTION */

   /* OPEN THE INPUT SECTION FILE */
   fdfrom = fopen("SUBC","r") ;

   /* WRITE THE TWO SWITCH ELEMENTS TO THE .CT FILE */
```

```
   /* STORE THE TWO CAPACITOR ELEMENTS FOR NOW      */
   readline(fdfrom,&buf[i++][0]) ;
   while (buf[i-1][0]!=EOF)
      { if (buf[i-1][0]=='S')
          { wrout(fdct,&buf[i-1][0],bs) ;
            i-- ;  }
        readline(fdfrom,&buf[i++][0]) ;   }

   fclose(fdfrom) ;

   /* STORE THE NODES OF THE TWO CAPACITOR ELEMENTS */
   getnode(&buf[0][0],ONE,st1) ;
   getnode(&buf[0][0],TWO,st2) ;
   getnode(&buf[1][0],ONE,st3) ;
   getnode(&buf[1][0],TWO,st4) ;

   /* WRITE THE CAPACITOR CONNECTED TO THE INPUT NODE */
   /* FIRST, WITH THE INPUT NODE LISTED FIRST. THE    */
   /* FILTER LINK NODE IS THE OTHER NODE.             */
   if (st1[0]=='0' || st2[0]=='0')
     { nodesort(&buf[1][0],firstnode) ;
       wrout(fdct,&buf[1][0],bc) ;
       getnode(&buf[1][0],TWO,midnode) ;

       /* UPDATE THE NODAL TABLE */
       if (st1[0]=='0')
         copy(st2,&table[1][3][0]) ;
       else copy(st1,&table[1][3][0]) ;
       wrout(fdct,&buf[0][0],bc) ;      }

   else { nodesort(&buf[0][0],firstnode) ;
       wrout(fdct,&buf[0][0],bc) ;
       getnode(&buf[0][0],TWO,midnode) ;

       /* UPDATE THE NODAL TABLE */
       if (st3[0]=='0')
         copy(st4,&table[1][3][0]) ;
       else copy(st3,&table[1][3][0]) ;
       wrout(fdct,&buf[1][0],bc) ;      }

   /* PUT A GAP IN THE NODE TABLE */
   table[1][4][0] = EOL ;

   /* PUT A GAP IN THE .CT FILE */
   putc(EOL,fdct) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT2 FUNCTION  --  Process Input Stage  (MARTIN)       */
```

```
/*                                                              */
/*     The input2 function processes the input stage of         */
/*  Ken Martin, consisting of one SISC. The regular             */
/*  software can process this element, hence the .in file       */
/*  is empty for this input stage. The filter link node is      */
/*  the other node of this SISC element.                        */
/*                                                              */
/****************************************************************/

input2(fdct)

FILE *fdct ;                          /* .CT OUTPUT FILE */

{  char buf[LENGTH] ;                 /* LINE BUFFER */
   char st1[WORD],st2[WORD] ;         /* NODE STORAGE STRINGS */
   FILE *fdfrom ;                     /* INPUT FILE */
   FILE *fopen() ;                    /* FILE OPEN FUNCTION */
   int flag = 1 ;                     /* SEARCH FLAG */

   /* SEARCH THE FILTER FILE FOR A SISC ELEMENT */
   /* CONNECTED TO THE FILTER INPUT NODE.       */
   fdfrom = fopen("SCINQ","r") ;
   readline(fdfrom,buf) ;
   while (buf[0]!=EOF && flag)
      { if (buf[0]=='S')
          { getnode(buf,ONE,st1) ;
            getnode(buf,TWO,st2) ;

            /* SAVE THE FILTER LINK NODE */
            if (same(firstnode,st1) || same(firstnode,st2))
              { flag = 0 ;
                if (same(firstnode,st2))
                  copy(st1,midnode) ;
                else copy(st2,midnode) ;    }   }

        readline(fdfrom,buf) ;   }

   /* UPDATE THE NODE TABLE */
   copy(firstnode,&table[1][3][0]) ;
   table[1][4][0] = EOL ;
}

/****************************************************************/
/*                                                              */
/*  INPUT3 FUNCTION  --  Process Input Stage  (SEDRA)           */
/*                                                              */
/*     The input3 function processes the input stage by         */
/*  Adel Sedra, consisting of one op-amp, one switch, one       */
/*  capacitor, and two SISC's. The op-amp is written first      */
/*  followed by the capacitor-switch pair and then the two      */
```

```
/*  SISC's. The SISC's are sorted to simplify the wiring   */
/*  in the layout program.                                 */
/*                                                         */
/***********************************************************/

input3(fdct)

FILE *fdct ;                          /* .CT OUTPUT FILE */

{  char buf[LENGTH],lin[LENGTH] ;     /* LINE BUFFERS */
   char st1[WORD],st2[WORD] ;         /* NODE STORAGE STRINGS */
   char st3[WORD],st4[WORD] ;         /* NODE STORAGE STRINGS */
   int flag,flag2 ;                   /* CLOCK PHASE FLAGS */
   FILE *fdfrom ;                     /* INPUT FILE */
   FILE *fdto ;                       /* OUTPUT FILE */
   FILE *fopen() ;                    /* FILE OPEN FUNCTION */

   fdfrom = fopen("SUBC","r") ;     /* OPEN INPUT STAGE FILE */
   fdto = fopen("SCINS","w") ;      /* OPEN STORAGE FILE */

   /* GROUP THE INPUT STAGE FROM THE OP-AMP */
   readline(fdfrom,buf) ;
   while (buf[0]!=EOF)

      /* FIND THE OP-AMP */
      { if (buf[0]=='E')
          { getnode(buf,ONE,st1) ;
            getnode(buf,THREE,st3) ;

            /* UPDATE THE NODE TABLE */
            copy(st3,&table[1][3][0]) ;
            copy(st1,&table[1][4][0]) ;

            /* WRITE THE OP-AMP TO THE .CT FILE */
            wrout(fdct,buf,oa) ;

            /* SEARCH FOR THE REMAINING ELEMENTS */
            rewind(fdfrom) ;
            readline(fdfrom,buf) ;
            while (buf[0]!=EOF)

               /* THERE ARE THREE CAPACITORS */
               { if (buf[0]=='C')
                   { getnode(buf,ONE,st2) ;
                     getnode(buf,TWO,st4) ;

                     /* IF IT CONNECTS TO THE NON-INVERTING */
                     /* OP-AMP INPUT, WRITE TO .CT FILE     */
                     if (same(st3,st2) || same(st3,st4))
                       wrout(fdct,buf,bc) ;   }
```

```
                /* THERE ARE FIVE SWITCHES */
                else if (buf[0]=='S')
                  { getnode(buf,ONE,st2) ;
                    getnode(buf,TWO,st4) ;

                    /* IF IT CONNECTS TO THE NON-INVERTING */
                    /* OP-AMP INPUT, WRITE TO .CT FILE     */
                    if (same(st3,st2) || same(st3,st4))
                      wrout(fdct,buf,bs) ;

                    /* ELSE GATHER UP THE REST OF THE SISC */
                    else { if (same(st1,st2))
                             sisc(fdfrom,fdto,buf,st4,st2) ;
                           if (same(st1,st4))
                             sisc(fdfrom,fdto,buf,st2,st4) ; } }

                readline(fdfrom,buf) ;     }   }

      readline(fdfrom,buf) ;   }

/* SPACE THE NODE TABLE */
table[1][5][0] = EOL ;

/* CLOSE INPUT AND OUTPUT FILES */
fclose(fdfrom) ; fclose(fdto) ;

/* THE FILTER LINK NODE IS THE SECOND */
/* NODE OF EITHER SISC ELEMENT        */
fdfrom = fopen("SCINS","r") ;
readline(fdfrom,buf) ;
getnode(buf,TWO,midnode) ;
fclose(fdfrom) ;

/* IT IS NOW NECESSARY TO SORT THE TWO SISC ELEMENTS */
/* THE ONE THAT IS SWITCHED SIMILAR TO THE FIRST     */
/* INTEGRATOR STAGE MUST GO LAST, WITH REVERSE NODES */

/* FIND THE FIRST INTEGRATOR STAGE */
fdfrom = fopen("SCINQ","r") ;
readline(fdfrom,buf) ;
while (buf[0]!=EOF)
   { if (buf[0]=='O')
       { getnode(buf,FOUR,st1) ;

         /* IS IT THE FIRST OP-AMP? */
         if (same(midnode,st1))

           /* FIND THE PHASE OF THIS STAGE */
           { do { readline(fdfrom,buf) ;
```

```
                    if (buf[0]=='S')
                      flag = clockphase(buf,ONE) ;  }
                while (buf[0]!=EOL && buf[0]!=EOF) ;  }   }

      readline(fdfrom,buf) ;   }
   fclose(fdfrom) ;

   /* GET THE TWO SISC ELEMENTS AND CHECK THE CLOCK PHASE */
   fdfrom = fopen("SCINS","r") ;
   readline(fdfrom,buf) ;
   readline(fdfrom,lin) ;
   flag2 = clockphase(lin,ONE) ;

   /* WRITE THE SISC ELEMENTS TO THE .CT FILE */
   if (flag==flag2)
     { writeline(fdct,buf) ;
       getnode(lin,TWO,st2) ;
       nodesort(lin,st2) ;
       writeline(fdct,lin) ;  }

   else { writeline(fdct,lin) ;
          getnode(buf,TWO,st2) ;
          nodesort(buf,st2) ;
          writeline(fdct,buf) ;  }

   /* PUT A GAP IN THE .CT FILE */
   putc(EOL,fdct) ;

   /* CLOSE THE INPUT FILE */
   fclose(fdfrom) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT4 FUNCTION  --  Process Input Stage  (AROMA)        */
/*                                                           */
/*************************************************************/

input4(fdct)

FILE *fdct ;                          /* .CT OUTPUT FILE */

{  char buf[LENGTH] ;                 /* LINE BUFFER */
   char st1[WORD],st2[WORD] ;         /* NODE STORAGE STRINGS */
   FILE *fdfrom ;                     /* INPUT FILE */
   FILE *fopen() ;                    /* FILE OPEN FUNCTION */
   char mid[3][WORD] ;                /* NODE STORAGE */
   int count = 0 ;                    /* COUNTER */

   mid[0][0] = mid[1][0] = mid[2][0] = EOW ;
```

```
   /* SEARCH THE FILTER FILE FOR A SISC ELEMENT */
   /* CONNECTED TO THE FILTER INPUT NODE.       */
   fdfrom = fopen("SCINQ","r") ;
   readline(fdfrom,buf) ;
   while (buf[0]!=EOF)
      { if (buf[0]=='S')
          { getnode(buf,ONE,st1) ;
            getnode(buf,TWO,st2) ;

            /* SAVE THE FILTER LINK NODE */
            if (same(firstnode,st1) || same(firstnode,st2))
              { if (same(firstnode,st2))
                  copy(st1,&mid[count++][0]) ;
                else copy(st2,&mid[count++][0]) ;     }    }

        readline(fdfrom,buf) ;    }

   if (mid[1][0]==EOW)
     copy(&mid[0][0],midnode) ;

   else { if (same(&mid[0][0],&mid[1][0]))
            copy(&mid[2][0],midnode) ;
          if (same(&mid[0][0],&mid[2][0]))
            copy(&mid[1][0],midnode) ;
          if (same(&mid[1][0],&mid[2][0]))
            copy(&mid[0][0],midnode) ;  }

   /* UPDATE THE NODE TABLE */
   copy(firstnode,&table[1][3][0]) ;
   table[1][4][0] = EOL ;
}

/*************************************************************/
/*                                                           */
/*  ORDER1 FUNCTION  --  Sort Stages                         */
/*                                                           */
/*     The order1 function sorts the integrator stages of    */
/*  the filter. The sort starts with the op-amp connected    */
/*  to the input stage, and then follows the SISC paths      */
/*  from the op-amp output of the stage. Once a stage has    */
/*  been sorted, it is ignored by the sorting algorithm.     */
/*                                                           */
/*************************************************************/

order1(fdnd)

FILE *fdnd ;                                  /* .ND OUTPUT FILE */

{  char old1[WORD],old2[WORD],old3[WORD] ; /* BASE PATH NODES */
```

```
char new1[WORD],new2[WORD],new3[WORD] ; /* FORWARD PATH NODES */
char new4[WORD],new5[WORD],new6[WORD] ; /* FORWARD PATH NODES */
char buf[LENGTH] ;                     /* LINE BUFFER */
long ofs1,ofs2,ofs3 ;                  /* FILE MARKERS */
int i = 0 ;                            /* INDEX VARIABLE */
FILE *fdto ;                           /* OUTPUT FILE */
FILE *fdfrom ;                         /* INPUT FILE */
FILE *fopen() ;                        /* FILE OPEN FUNCTION */

fdfrom = fopen("SCINQ","r") ;     /* OPEN INPUT FILE */
fdto = fopen("SCINP","w") ;       /* OPEN OUTPUT FILE */

/* FIND THE OP-AMP CONNECTED TO THE LINK NODE */
readline(fdfrom,buf) ;
while (buf[0]!=EOF)
   { if (buf[0]=='O')
       { getnode(buf,ONE,new1) ;
         getnode(buf,FOUR,new4) ;

         /* THE OUTPUT NODE IS THE START OF THE SORT */
         if (same(midnode,new4))
           copy(new1,old1) ;       }

     readline(fdfrom,buf) ;    }
rewind(fdfrom) ;

/* START THE SORT FROM THE FIRST OUTPUT NODE */
do { ofs1 = newblock(old1,new1,new2,fdfrom) ;

     /* WRITE THE INTEGRATOR STAGE TO THE OUTPUT FILE */
     dump(fdfrom,ofs1,fdto) ;

     /* IF A SINGLE FEED, UPDATE THE SEARCH PATH */
     if (new2[0]==EOW)
       copy(new1,old1) ;

     /* IF A DOUBLE FEED, CHECK BOTH NEW PATHS */
     else { while(new2[0]!=EOW)
       { copy(new1,old2) ;
         copy(new2,old3) ;

         /* TRACE THE NEW PATHS */
         ofs2 = newblock(old2,new3,new4,fdfrom) ;
         ofs3 = newblock(old3,new5,new6,fdfrom) ;

         /* DUMP THE INTEGRATOR WITH NO NEW FEEDS FIRST */
         /* FOLLOWED BY THE OTHER INTEGRATOR, THEN      */
         /* UPDATE THE SEARCH PATH                      */
         if (new3[0]==EOW && new4[0]==EOW)
             { dump(fdfrom,ofs2,fdto) ;
```

```
                    dump(fdfrom,ofs3,fdto) ;
                    copy(new5,old1) ;
                    copy(new5,new1) ;
                    copy(new6,new2) ;      }

               else if (new5[0]==EOW && new6[0]==EOW)
                   { dump(fdfrom,ofs3,fdto) ;
                     dump(fdfrom,ofs2,fdto) ;
                     copy(new3,old1) ;
                     copy(new3,new1) ;
                     copy(new4,new2) ;      }    }    }    }

   /* CONTINUE UNTIL NO NEW PATHS EXIST */
   while (new1[0]!=EOW || new2[0]!=EOW) ;

   /* WRITE THE NODE TABLE TO THE .ND FILE */
   while (table[1][i][0]!=EOW)
      { writeline(fdnd,&table[1][i][0]) ;
        if (table[1][i++][0]!=EOL)
          putc(EOL,fdnd) ;    }

   /* CLOSE THE INPUT AND OUTPUT FILES */
   fclose(fdfrom) ; fclose(fdto) ;
}

/*************************************************************/
/*                                                           */
/*  NEWBLOCK FUNCTION  --  Find Next Stages                  */
/*                                                           */
/*     The newblock function finds the paths by which the    */
/*  given node feeds new integrator stages. It returns the   */
/*  input nodes of the found stages. The searching is only   */
/*  along SISC elements.                                     */
/*                                                           */
/*************************************************************/

newblock(old,new1,new2,fdfrom)

char old[WORD] ;                /* START OF SEARCH PATH */
char new1[WORD] ;               /* END1 OF SEARCH PATH */
char new2[WORD] ;               /* END2 OF SEARCH PATH */
FILE *fdfrom ;                  /* INPUT FILE */

{  char st1[WORD],st2[WORD] ;   /* NODE STORE STRINGS */
   char st3[WORD],st4[WORD] ;   /* NODE STORE STRINGS */
   char buf[LENGTH] ;           /* LINE BUFFER */
   int flag = 0 ;               /* FLAG FOR STARTING STAGE */
   int flag2 = 0 ;              /* FLAG FOR SECOND PATH */
   long offs1,offs2 ;           /* FILE MARKERS */
```

```
   /* INITIALIZE THE FORWARD PATH NODES */
   new1[0] = EOW ; new2[0] = EOW ;

   /* HAVE TO FIND THE STARTING STAGE LOCATION IN */
   /* ADDITION TO THE TWO POSSIBLE FORWARD PATHS  */
   rewind(fdfrom) ;
   do { offs1 = ftell(fdfrom) ;

        /* AN OP-AMP IS ALWAYS FIRST */
        readline(fdfrom,buf) ;
        getnode(buf,ONE,st1) ;
        getnode(buf,FOUR,st4) ;

        /* CHECK FOR STARTING STAGE */
        if (same(old,st1) && !flag)

          /* SAVE THE FILE OFFSET */
          { offs2 = offs1 ;
            flag = 1 ;
            while (buf[0]!=EOL && buf[0]!=EOF)
               readline(fdfrom,buf) ;    }

        /* ELSE SEE IF STAGE IS FED BY THE STARTING NODE */
        else do { readline(fdfrom,buf) ;

            /* ONLY SISC ELEMENTS COUNT */
            if (buf[0]=='S')
              { getnode(buf,ONE,st2) ;
                getnode(buf,TWO,st3) ;

                /* SAVE IT IF IT SHARES THE STARTING NODE */
                if (same(st2,old) && !in(st3,1) ||
                    same(st3,old) && !in(st2,1))

                  { if (!in(st1,0))
                      { insert(st1,0) ;
                        if (!flag2)
                          copy(st1,new1) ;
                        else copy(st1,new2) ;
                        flag2 = 1 ;   }  }  }  }

             while (buf[0]!=EOL && buf[0]!=EOF) ;  }

   while (buf[0]!=EOF) ;
   return(offs2) ;
}

/*************************************************************/
/*                                                           */
/*  ORDER2 FUNCTION  --  Sort Elements and Nodes             */
```

```
/*                                                              */
/*     The order2 function sorts the elements and nodes of      */
/*  the integrator stages. The elements are put in the          */
/*  order as the nodes in TABLE are. Integrating caps            */
/*  are written before integrating SC's. The op-amp input       */
/*  node is put first for IC and SC elements. The CC nodes      */
/*  are sorted in the same order as TABLE.                      */
/*                                                              */
/****************************************************************/

order2(fdct)

FILE *fdct ;                            /* .CT OUTPUT FILE */

{  char buf[10][LENGTH] ;               /* LINE BUFFERS FOR STAGE */
   char line[LENGTH] ;                  /* LINE BUFFER */
   char op1[WORD],op4[WORD] ;           /* NODE STORAGE STRINGS */
   char st1[WORD],st2[WORD] ;           /* NODE STORAGE STRINGS */
   int flag ;                           /* NODE BEFORE/AFTER FLAG */
   int count ;                          /* COUNTER OF ELEMENTS */
   int j,k,r ;                          /* INDEX VARIABLES */
   FILE *fdfrom ;                       /* INPUT FILE */
   FILE *fopen() ;                      /* FILE OPEN FUNCTION */

   fdfrom = fopen("SCINP","r") ;        /* OPEN INPUT FILE */

   /* SORT THE BLOCKS ONE AT A TIME */
   readline(fdfrom,line) ;
   while (line[0]!=EOF)

      /* SAVE THE OP-AMP NODES */
      { getnode(line,FOUR,op4) ;
        getnode(line,ONE,op1) ;
        writeline(fdct,line) ;

        /* STORE THE REST OF THE INTEGRATOR BLOCK */
        count = 0 ;
        do { readline(fdfrom,&buf[count][0]) ; }
        while (buf[count++][0]!=EOL) ;

        /* WRITE THE ELEMENTS AS THE NODE */
        /* SEARCH COMES TO THEIR NODES    */
        flag = 0 ;
        for (k=2; table[1][k][0]!=EOW; k++)
           for (j=0; j<count-1; j++)
              { getnode(&buf[j][0],ONE,st1) ;
                getnode(&buf[j][0],TWO,st2) ;

                /* WRITE IC BEFORE INTEGRATING SC */
                if (same(op4,&table[1][k][0]))
```

```
                    { if (buf[j][0]=='I')
                        if (same(st1,op1) && same(st2,op4) ||
                            same(st1,op4) && same(st2,op1))

                          /* SET FLAG AND SORT THE NODES */
                          { flag = 1 ;
                            nodesort(&buf[j][0],op4) ;
                            writeline(fdct,&buf[j][0]) ;  }   }

                  /* WRITE INTEGRATING SC */
                  else if (same(op1,&table[1][k][0]))
                    { if (buf[j][0]=='S')
                        if (same(st1,op1) && same(st2,op4) ||
                            same(st1,op4) && same(st2,op1))

                          /* SORT THE NODES */
                          { nodesort(&buf[j][0],op4) ;
                            writeline(fdct,&buf[j][0]) ;  }   }

                  /* WRITE REMAINING ELEMENTS, CHECK FIRST NODE */
                  else if (same(st1,&table[1][k][0]))

                    /* CC'S NODES ARE IN SAME ORDER AS TABLE */
                    { if (buf[j][0]=='C')
                        { if (flag)
                            nodesort(&buf[j][0],st2) ;  }

                      /* MUST BE SC */
                      else nodesort(&buf[j][0],op4) ;
                      writeline(fdct,&buf[j][0]) ;    }

                  /* CHECK THE OTHER NODE */
                  else if (same(st2,&table[1][k][0]))

                    /* CC'S NODES ARE IN SAME ORDER AS TABLE */
                    { if (buf[j][0]=='C')
                        { if (!flag)
                            nodesort(&buf[j][0],st2) ;  }

                      /* MUST BE SC */
                      else nodesort(&buf[j][0],op4) ;
                      writeline(fdct,&buf[j][0]) ;    }  }

          /* INSERT GAP IN .CT FILE */
          putc(EOL,fdct) ;
          readline(fdfrom,line) ;              }

    fclose(fdfrom) ;            /* CLOSE INPUT FILE */
}
```

```
/*************************************************************/
/*                                                           */
/*  CLOCKPHASE FUNCTION  --  Check Phase of a Clock          */
/*                                                           */
/*     The clockphase function checks the phase of the       */
/*  specified. This function is only valid for SC elements   */
/*  and the clocks must be of the C and #C variety.          */
/*                                                           */
/*************************************************************/

clockphase(buf,spot)

char buf[LENGTH] ;          /* SC ELEMENT LINE BUFFER */
int spot ;                  /* CLOCK POSITION 1 - 4 */

{  int j,i ;                /* INDEX VARIABLES */

   /* ADVANCE TO THE BEGINNING OF THE CAP VALUE */
   j = 0 ;
   while (buf[j++]!=')') ;
   while (buf[j]==' ') j++ ;

   /* ADVANCE TO THE DESIRED CLOCK */
   for (i=0; i<=spot; i++)
      { while (buf[j]!=' ') j++ ;
        while (buf[j]==' ') j++ ;  }

   /* RETURN 0 IF #C IS FOUND */
   return(((buf[j]=='#') ? 0 : 1)) ;
}

/*************************************************************/
/*                                                           */
/*  COPY FUNCTION  --  String Copy Function                  */
/*                                                           */
/*     The copy function copies string1 to string2. The      */
/*  first string must be terminated by an EOW.               */
/*                                                           */
/*************************************************************/

copy(str1,str2)

char *str1 ;                /* INPUT STRING */
char *str2 ;                /* OUTPUT STRING */

{  int i = 0 ;              /* INDEX VARIABLE */

   /* COPY STRING1 TO STRING 2 */
   do { *(str2+i) = *(str1+i) ;  }
   while (*(str1+i++) != EOW) ;
```

```
}

/*************************************************************/
/*                                                           */
/*  DUMP FUNCTION  --  Write Integerator Stage to File       */
/*                                                           */
/*     The dump function writes the integrator stage found   */
/*  at the file location specified by the offset. If also    */
/*  updates the node table.                                  */
/*                                                           */
/*************************************************************/

dump(fdfrom,offset,fdto)

FILE *fdfrom ;                  /* INPUT FILE */
FILE *fdto ;                    /* OUTPUT FILE */
long offset ;                   /* INPUT FILE OFFSET */

{  char buf[LENGTH] ;           /* LINE BUFFER */
   char st[WORD] ;              /* NODE STORAGE STRING */

   /* INDEX INTO THE INPUT FILE */
   fseek(fdfrom,offset,ZERO) ;

   /* COPY INTEGRATOR STAGE TO OUTPUT FILE */
   do { readline(fdfrom,buf) ;
        if (buf[0]=='O')

          /* INSERT OP-AMP INPUT NODE */
          { getnode(buf,FOUR,st) ;
            insert(st,1) ;

            /* INSERT OP-AMP OUTPUT NODE */
            getnode(buf,ONE,st) ;
            insert(st,1) ;   }

        writeline(fdto,buf) ;     }
   while (buf[0]!=EOL && buf[0]!=EOF) ;
}

/*************************************************************/
/*                                                           */
/*  EMPTY FUNCTION  --  Clears a Column of TABLE             */
/*                                                           */
/*     The empty function initializes one column of TABLE    */
/*  to EOW.                                                  */
/*                                                           */
/*************************************************************/

empty(col)
```

```
int col ;                              /* COLUMN OF TABLE */

{  int b ;                             /* INDEX VARIABLE */

   /* INITIALIZE TABLE COLUMN */
   for (b=0; b<SIZE; b++)
      table[col][b][0] = EOW ;
}

/*************************************************************/
/*                                                           */
/*  FILL FUNCTION  --  Insert New Nodes                      */
/*                                                           */
/*     The fill function fills up TABLE for subcircuit       */
/*  substitutions. The inserted nodes are of the form !AD    */
/*  which are not allowed in SWITCAP.                        */
/*                                                           */
/*************************************************************/

fill()

{  int y = (-1) ;                      /* INDEX VARIABLE */

   /* SCAN COLUMN ZERO AND SEE WHAT COLUMN ONE HAS */
   while (table[0][++y][0]!=EOW)

      /* THE GROUND NODE IS ALWAYS '0' */
      { if (table[0][y][0]=='0')
          copy(&table[0][y][0],&table[1][y][0]) ;

        /* ELSE INSERT THE NODE AND UPDATE THE NODE */
        else if (table[1][y][0]==EOW)
          { copy(node,&table[1][y][0]) ;
            node[2] += 1 ;

            /* CHECK FOR OVERFLOW */
            if (node[2]>'Z')
              { node[2] = 'A' ;
                node[1] += 1 ;   }  }  }
}

/*************************************************************/
/*                                                           */
/*  GETNODE FUNCTION  --  Get One Node of a Line             */
/*                                                           */
/*     The getnode function returns the requested node       */
/*  from the provided line.                                  */
/*                                                           */
/*************************************************************/
```

```
getnode(buf,pos,vertx)

char buf[LENGTH] ;      /* LINE BUFFER */
char vertx[WORD] ;      /* NODE BUFFER */
int pos ;               /* NODE POSITION */

{  int q,i,k ;          /* INDEX VARIABLES */

   q = i = 0 ;
   while (buf[i++]!='(')  ;

   /* STORE THE REQUESTED NODE */
   for (k=1; k<=pos; k++)
      { while (buf[i]==' ') i++ ;
        while (buf[i]!=' ' && buf[i]!=')')
           { if (k==pos)
               vertx[q++] = buf[i++] ;
             else i++ ;                  }     }

   vertx[q] = EOW ;
}

/*************************************************************/
/*                                                           */
/*  GETVAL FUNCTION  --  Gets Variable From Line             */
/*                                                           */
/*     The getval function returns the variable stored       */
/*  after the nodes in the line buffer.                      */
/*                                                           */
/*************************************************************/

getval(buf,store)

char buf[LENGTH] ;     /* LINE BUFFER */
char store[WORD] ;     /* VARIABLE BUFFER */

{  int q,i ;           /* INDEX VARIABLES */

   /* ADVANCE BEYOND THE NODES */
   q = i = 0 ;
   while (buf[i++]!=')')  ;
   while (buf[i]==' ') i++ ;

   /* COPY THE VARIABLE TO THE STRING BUFFER */
   while (buf[i]!=' ' && buf[i]!=';' && buf[i]!='(')
      store[q++] = buf[i++] ;
   store[q] = EOW ;
}
```

```
/*********************************************************/
/*                                                       */
/*  IN FUNCTION  -- See if String is in TABLE            */
/*                                                       */
/*     The in function determines if the input string is */
/*  stored in the given column of TABLE.                 */
/*                                                       */
/*********************************************************/

in(str,col)

char *str ;                          /* INPUT STRING */
int col ;                            /* TABLE COLUMN */

{  int i = 0 ;                       /* INDEX VARIABLE */

   /* SCAN THE TABLE COLUMN FOR THE GIVEN STRING */
   do { if (same(str,&table[col][i][0]))
          return(1) ;  }
   while (table[col][i++][0]!=EOW) ;

   /* IF NOT FOUND RETURN ZERO */
   return(0) ;
}

/*********************************************************/
/*                                                       */
/*  INSERT FUNCTION  --  Insert String in TABLE          */
/*                                                       */
/*     The insert function inserts the given string into */
/*  the requested column of TABLE. The first open slot is */
/*  used.                                                */
/*                                                       */
/*********************************************************/

insert(str,col)

char *str ;                         /* STRING TO BE INSERTED */
int col ;                           /* COLUMN FOR INSERTION */

{  int i = 0 ;                      /* INDEX VARIABLE */

   /* COPY THE STRING INTO THE FIRST OPEN SLOT */
   while (table[col][i++][0]!=EOW) ;
   copy(str,&table[col][i-1][0]) ;
}

/*********************************************************/
/*                                                       */
/*  NODESORT FUNCTION  --  Sort Nodes of Given Line      */
```

```
/*                                                              */
/*     The nodesort function ensures that the the given         */
/*  node is the first node of the given line.                   */
/*                                                              */
/****************************************************************/

nodesort(buf,str)

char buf[LENGTH] ;                  /* LINE BUFFER */
char str[WORD] ;                    /* NODE BUFFER */

{  char st1[WORD],st2[WORD] ;       /* NODE STORAGE BUFFERS */
   int i,j ;                        /* INDEX VARIABLES */

   i = j = 0 ;
   getnode(buf,ONE,st1) ;
   getnode(buf,TWO,st2) ;

   /* IF THE NODE IS IN THE SECOND  */
   /* SPOT THEN REVERSE THE NODES   */
   if (!same(str,st1))
     { while (buf[i++]!='(') ;

       while (st2[j]!=EOW)
          buf[i++] = st2[j++] ;
       buf[i++] = ' ' ;

       j = 0 ;
       while (st1[j]!=EOW)
          buf[i++] = st1[j++] ;

       while (buf[i]!=')')
          buf[i++] = ' ' ;          }
}

/****************************************************************/
/*                                                              */
/*  READCLEAN FUNCTION  --  Read and Clean a Line               */
/*                                                              */
/*     The readclean function reads aline from the input        */
/*  file and modifies it as follows:                            */
/*                                                              */
/*  1. Capitalize letters          2. Remove comments           */
/*  3. Single statement per line   4. Left adjust the line      */
/*                                                              */
/****************************************************************/

readclean(fdfrom,buf)

FILE *fdfrom ;              /* INPUT FILE */
```

```
char buf[LENGTH] ;          /* LINE BUFFER */

{  int a,b ;                /* COMMENT BOUNDS */
   int ch ;                 /* FILE CHARACTER */
   int i,q ;                /* INDEX VARIABLES */

   q = 0 ;
   a = b = (-1) ;

   /* START READING CHARACTERS FROM INPUT FILE */
   do { ch = getc(fdfrom) ;
        buf[q++] = ((ch>='a' && ch<='z') ? ch-'a'+'A' : ch) ;

        /* CHECK FOR COMMENT BEGINNING */
        if (q>1 && buf[q-2]=='/' && buf[q-1]=='*')
            a = q - 2 ;

        /* CHECK FOR COMMENT END */
        if (a>=0 && buf[q-2]=='*' && buf[q-1]=='/')
            b = q - 1 ;     }

   while (ch!=EOL && ch!=EOF && !(ch==';' && q-1>b)) ;

   /* APPEND EOL IF NEEDED */
   if (ch==';')
     buf[q] = EOL ;

   /* IF COMMENT EXISTS, BLANK IT OUT */
   if (a!=b)
     for (i=a; i<=b; i++)
        buf[i] = ' ' ;

   /* CHECK FOR SPACES AT BEGINNING OF LINE */
   q = 0 ;
   while (buf[q]==' ' && buf[q]!=EOL && buf[q]!=EOF)
      q++ ;

   /* LEFT ADJUST THE LINE */
   i = q ;
   if (q>0)
     do { buf[i-q] = buf[i] ; }
     while (buf[i++] != EOL) ;
}

/*************************************************************/
/*                                                           */
/*  READLINE FUNCTION  --  Read a Line From Input File       */
/*                                                           */
/*     The readline function reads a line terminated by an   */
/*  EOL or EOF from the input file given.                    */
```

```
/*                                                                  */
/********************************************************************/

readline(fdfrom,buf)

FILE *fdfrom ;              /* INPUT FILE */
char buf[LENGTH] ;          /* LINE BUFFER */

{  int q = 0 ;              /* INDEX VARIABLE */

   /* READ CHARACTERS FROM INPUT FILE */
   do { buf[q++] = getc(fdfrom) ;  }
   while (buf[q-1]!=EOL && buf[q-1]!=EOF) ;
}

/********************************************************************/
/*                                                                  */
/*  SAME FUNCTION  --  String Equality Tester                       */
/*                                                                  */
/*     The same function compares two strings up to the             */
/*  last EOW, EOL, or EOF.                                          */
/*                                                                  */
/********************************************************************/

same(buf1,buf2)

char *buf1 ;                /* FIRST STRING */
char *buf2 ;                /* SECOND STRING */

{  int k = (-1) ;           /* INDEX VARIABLE */
   int set = 1 ;            /* OFFSET FOR LONG STRINGS */

   /* TAKE NOTE IF STRING IS TERMINATED WITH EOL */
   do { if (buf1[++k]==EOL)
          set = 0 ;       }
   while (buf1[k]!=EOW && buf1[k]!=EOL && buf1[k]!=EOF) ;

   /* COMPARE THE STRINGS CHARACTER BY CHARACTER */
   k = 0 ;
   do { if (buf1[k]!=buf2[k++])
          return(0) ;     }
   while (buf2[k-set]!=EOW && buf1[k-set]!=EOW) ;
   return(1) ;
}

/********************************************************************/
/*                                                                  */
/*  SCOUT FUNCTION  --  WRITE SC ELEMENT                            */
/*                                                                  */
/*     The scout function assembles the data for the SC             */
```

```
/*  element and writes it to the output file.                    */
/*                                                               */
/***************************************************************/

scout(con1,con4,clk,fdto)

char con1[WORD] ;              /* FIRST NODE OF SC ELEMENT */
char con4[WORD] ;              /* SECOND NODE OF SC ELEMENT */
char clk[5][WORD] ;            /* CAP VALUE AND CLOCKS */
FILE *fdto ;                   /* OUTPUT FILE */

{  char line[LENGTH] ;         /* LINE BUFFER */
   int q = 4 ;                 /* INDEX VARIABLE */
   int i = 0 ;                 /* INDEX VARIABLE */
   int x = 0 ;                 /* INDEX VARIABLE */

   /* LABEL THE SC ELEMENT */
   copy(sc,&line[0]) ;

   /* INSERT THE FIRST NODE */
   while (con1[x]!=EOW)
      line[q++] = con1[x++] ;
   line[q++] = ' ' ;

   /* INSERT THE SECOND NODE */
   while (con4[i]!=EOW)
      line[q++] = con4[i++] ;
   line[q++] = ')' ;

   /* INSERT THE CAP VALUE AND THE FOUR CLOCKS */
   for (x=0; x<5; x++)
      { line[q++] = ' ' ;
        i = 0 ;
        while (clk[x][i]!=EOW)
           line[q++] = clk[x][i++] ;  }

   /* TERMINATE THE LINE AND WRITE IT TO THE OUTPUT FILE */
   line[q++] = ' ' ;
   line[q++] = ';' ;
   line[q] = EOL ;
   writeline(fdto,line) ;
}

/***************************************************************/
/*                                                             */
/*  WRITELINE FUNCTION  --  Write a Line To Output File        */
/*                                                             */
/*     The writeline function writes a line terminated by      */
/*  EOL or EOF or EOW to the output file.                      */
/*                                                             */
```

```
/*************************************************************/

writeline(fdto,buf)

FILE *fdto ;                /* OUTPUT FILE */
char buf[LENGTH] ;          /* LINE BUFFER */

{  int q = 0 ;              /* INDEX VARIABLE */

   /* WRITE THE LINE TO THE OUTPUT FILE */
   do { putc(buf[q++],fdto) ; }
   while (buf[q-1]!=EOL && buf[q-1]!=EOF && buf[q]!=EOW) ;
}

/*************************************************************/
/*                                                           */
/*  WROUT FUNCTION  --  Write Encoded Output                 */
/*                                                           */
/*     The wrout function relabels circuit elements and      */
/*  writes it to the output file.                            */
/*                                                           */
/*************************************************************/

wrout(fdto,buf,title)

FILE *fdto ;                /* OUTPUT FILE */
char buf[LENGTH] ;          /* LINE BUFFER */
char *title ;               /* NEW LABEL FOR ELEMENT */

{  char line[LENGTH] ;      /* LINE BUFFER */
   int i = 0 ;              /* INDEX VARIABLE */
   int q = 4 ;              /* INDEX VARIABLE */

   /* RELABEL THE LINE */
   copy(title,&line[0]) ;
   while (buf[i++]!='(') ;

   /* INSERT THE REST OF THE LINE FROM BEFORE */
   while (buf[i-1]!=EOL)
      line[q++] = buf[i++] ;

   /* WRITE THE LINE TO THE OUTPUT FILE */
   writeline(fdto,line) ;
}
```

APPENDIX D: THE LAYOUT GENERATOR LISTING

The layout generator requires the filename.nd, filename.ct, filename.com, and the opamp.cif files to run. The operational amplifier CIF code must be symbol #2 and must be in a file called opamp.cif. It is run as follows:

SISCL filename style <CR>

where style refers to one of the following design techniques: A(AROMA), L(Lee), M(Martin), and S(Datar & Sedra). The single character is used. It produces one output file called filename.cif.

```
#include <stdio.h>
#define  LENGTH  72              /* BUFFER LENGTH */
#define  WORD     8              /* IDENTIFIER LENGTH */
#define  ZERO     0              /* ZERO CONSTANT */
#define  ONE      1              /* ONE CONSTANT */
#define  TWO      2              /* TWO CONSTANT */
#define  EOW    '\0'             /* END OF WORD */
#define  EOL    '\n'             /* END OF LINE */

#define  LIMIT      0.45         /* FRACTION BOUND */
#define  EDGE      20            /* PLATE SIZE */
#define  CAPGAP     2            /* PLATE SEPARATION */
#define  TOPMAX    20            /* CAP ARRAY HEIGHT */
#define  MAXOP     20            /* TOTAL OP-AMPS */
#define  MAXCAP    80            /* TOTAL CAPACITORS */
#define  MAXVAL   100            /* MAX CAP VALUE */

#define  M1W   "L M1 ;W"         /* METAL WIRE COMMAND */
#define  M1B   "L M1 ;B"         /* METAL BOX COMMAND */

#define  P1W   "L P1 ;W"         /* POLY1 WIRE COMMAND */
#define  P1B   "L P1 ;B"         /* POLY1 BOX COMMAND */

#define  P2W   "L P2 ;W"         /* POLY2 WIRE COMMAND */
#define  P2B   "L P2 ;B"         /* POLY2 BOX COMMAND */

#define  C1W   "L C1 ;W"         /* CUT WIRE COMMAND */
#define  C1B   "L C1 ;B"         /* CUT BOX COMMAND */

#define  ACW   "L AC ;W"         /* ACTIVE WIRE COMMAND */
#define  ACB   "L AC ;B"         /* ACTIVE BOX COMMAND */

#define  PWB   "L PW ;B"         /* P-WELL BOX COMMAND */
#define  PPB   "L PP ;B"         /* P+ BOX COMMAND */
#define  NPB   "L NP ;B"         /* N+ BOX COMMAND */

char table[3][50][WORD] ;        /* NODE TABLE */
short grid[2000][TOPMAX] ;       /* CAPACITOR LAYOUT ARRAY */
short path[10][3*MAXCAP] ;       /* CAPACITOR WIRING ARRAY */
short path2[5][5000] ;           /* OP-AMP WIRING ARRAY */

float LAMBDA ;                   /* LAYOUT SCALING FACTOR */
int U ;                          /* CIF BASE UNIT */
int XBASE = 1400 ;               /* LAYOUT X-AXIS START */
int YBASE = 34000 ;              /* LAYOUT Y-AXIS START */
int w ;                          /* 2*LAMBDA WIRE WIDTH */

int total ;                     /* NUMBER OF CAPACITORS */
int topper ;                    /* CAP ARRAY HEIGHT */
int stages ;                    /* NUMBER OF FILTER STAGES */
```

```
int swap ;                          /* ARRAY LAYOUT STRATEGY */
int start ;                         /* FILTER STARTING POINT */
int level[10] ;                     /* VERTICAL LAYOUT LEVELS */

struct cell { int height ;           /* OP-AMP HEIGHT */
              int width ;            /* OP-AMP WIDTH */
              int xorg ;             /* X ORIGIN OF OP-AMP */
              int yorg ;             /* Y ORIGIN OF OP-AMP */
              int level ;            /* OP-AMP CONNECTION LEVEL */
              int con[3] ;           /* CONNECTION DISPLACEMENTS */
              int vos[3] ;           /* POWER CONNECT LEVELS */
              int wid[3] ; } ;       /* POWER LINE WIDTHS */

struct cell opamp ;                  /* OP-AMP DATA */

struct mother { int cap ;               /* CAPACITOR CONTACT */
                int out ;               /* OP-AMP CONTACT */
                int cross ;             /* HORIZONTAL CROSSING */
                int level ;             /* WIRING LEVEL */
                int base ;              /* OP-AMP PLACEMENT */
                int mirror ;            /* MIRROR FLAG */
                int gnd ;               /* +INPUT CONTACT */
                int gndcross ;          /* HORIZONTAL CROSSING */
                int gndlevel ; } ;      /* WIRING LEVEL */

struct mother whip[2*MAXOP] ;        /* OP-AMP CONNECTION DATA */

struct contain { float val ;              /* CAPACITOR VALUE */
                 float v2 ;               /* CAPACITOR VALUE */
                 short hor[MAXVAL] ;      /* PLATE HORIZONTAL */
                 short ver[MAXVAL] ;      /* PLATE VERTICAL */
                 int pnt ;                /* NUMBER OF PLATES */
                 int next ;               /* EXTRA CAP GAPS */
                 int used ;               /* SWITCH PLACEMENT */
                 int swtch ;              /* CLOCKING SCHEME */
                 int lay ;                /* CAPACITOR LAYOUT */
                 int flag ;               /* SWITCH CONNECTION */
                 int tipflag ;            /* TIP PLACEMENT */
                 char typ ;               /* CAPACITOR TYPE */
                 char fst[WORD] ;         /* FIRST CAP NODE */
                 char sec[WORD] ; } ;     /* SECOND CAP NODE */

struct contain capac[MAXCAP] ;           /* CAPACITOR DATA */

/*************************************************************/
/*                                                           */
/*  MAIN PROGRAM  --  LAYOUT GENERATOR                       */
/*                                                           */
/*************************************************************/
```

```
main(argc,argv)                 /* FETCH C SHELL PARAMETERS */

int argc ;                      /* NUMBER OF ARGUMENTS */
char *argv[] ;                  /* ARGUMENT POINTERS */

{  char head ;                  /* INPUT SECTION STYLE */
   char *buf ;                  /* LINE BUFFER */
   char *fp ;                   /* FILE POINTER */
   char st ;                    /* CHARACTER STORAGE */
   int i ;                      /* INDEX VARIABLE */
   FILE *fdct ;                 /* .CT FILE DESCRIPTOR */
   FILE *fdnd ;                 /* .ND FILE DESCRIPTOR */
   FILE *fdcom ;                /* .COM FILE DESCRIPTOR */
   FILE *fdcif ;                /* .CIF FILE DESCRIPTOR */
   FILE *fopen() ;              /* FILE OPEN FUNCTION */

   /* SAVE THE INPUT SECTION STYLE */
   st = *argv[2] ;
   head = ((st>='a' && st<='z') ? st-'a'+'A' : st) ;

   /* OPEN THE .CT FILE FOR READING */
   fp = argv[1] ;
   for (i=0; *(fp+i)!=EOW; i++) ;
   *(fp+i+0) = '.' ;
   *(fp+i+1) = 'c' ;
   *(fp+i+2) = 't' ;
   *(fp+i+3) = EOW ;
   fdct = fopen(fp,"r") ;

   /* OPEN THE .COM FILE FOR READING */
   *(fp+i+1) = 'c' ;
   *(fp+i+2) = 'o' ;
   *(fp+i+3) = 'm' ;
   *(fp+i+4) = EOW ;
   fdcom = fopen(fp,"r") ;

   /* OPEN THE .ND FILE FOR READING */
   *(fp+i+1) = 'n' ;
   *(fp+i+2) = 'd' ;
   *(fp+i+3) = EOW ;
   fdnd = fopen(fp,"r") ;

   /* OPEN THE .CIF FILE FOR WRITING */
   *(fp+i+1) = 'c' ;
   *(fp+i+2) = 'i' ;
   *(fp+i+3) = 'f' ;
   *(fp+i+4) = EOW ;
   fdcif = fopen(fp,"w") ;

   /* INITIALIZE THE NODE AND CAPACITOR DATA */
```

```
    init(fdcom,fdct,fdnd,head) ;

    /* GENERATE THE CAPACITOR ARRAY */
    caparray(fdcif) ;

    /* INITIALIZE THE OP-AMP AND SWITCH DATA */
    preop(fdcif) ;

    /* WIRE THE INPUT NODES */
    wire1(fdcif) ;

    /* WIRE THE OUTPUT NODES */
    wire2(fdcif) ;

    /* WIRE THE HIDDEN INPUT NODES */
    wire3(fdcif) ;

    /* WIRE THE HIDDEN OUTPUT NODES */
    wire4(fdcif,0) ;
    wire4(fdcif,1) ;

    /* CONNECT THE SWITCHES */
    streak(fdcif) ;

    /* CONNECT THE INPUT AND OUTPUT NODES */
    wire5(fdcif) ;

    /* CONNECT THE GROUND LEADS */
    wire6(fdcif) ;

    /* PLACE AND CONNECT THE OP-AMPS */
    ampson(fdcif) ;

    /* INSERT GLOBAL LAYOUT FEATURES */
    wrapup(fdcif) ;

    /* HARD-WIRE THE INPUT SECTION */
    if (head=='L') input1(fdcif) ;
    if (head=='M') input2(fdcif) ;
    if (head=='S') input3(fdcif) ;
    if (head=='A') input4(fdcif) ;

    /* FINISH THE .CIF FILE */
    buf = "DF ;\n" ;
    writeline(fdcif,buf) ;

    /* CLOSE THE FILES */
    fclose(fdct)  ; fclose(fdcif) ;
    fclose(fdcom) ; fclose(fdnd)  ;
}
```

```
/*************************************************************/
/*                                                           */
/*  INIT FUNCTION  --  Read Input Data                       */
/*                                                           */
/*     The init function reads the input .CT file and        */
/*  initializes the capac structure. It also initializes     */
/*  the node table.                                          */
/*                                                           */
/*************************************************************/

init(fdcom,fdct,fdnd,head)

FILE *fdcom ;                          /* .COM INPUT FILE */
FILE *fdct ;                           /* .CT INPUT FILE */
FILE *fdnd ;                           /* .ND INPUT FILE */
char head ;                            /* INPUT SECTION STYLE */

{  int i,j,k ;                         /* INDEX VARIABLES */
   int conn[4] ;                       /* CLOCK PHASES */
   int flip[MAXCAP] ;                  /* CAP CLOCK STORAGE */
   char line[LENGTH] ;                 /* LINE BUFFER */
   char st1[WORD],st2[WORD] ;          /* NODE BUFFERS */

   /* INITIALIZE THE LEVEL STORAGE */
   for (i=0; i<10; i++)
      level[i] = 0 ;

   /* READ THE .COM FILE DATA */
   readcom(fdcom) ;

   /* READ INPUT AND OUTPUT NODES */
   readnd(fdnd,head) ;

   /* INITIALIZE THE CAPACITORS */
   i = 1 ;
   stages = 0 ;
   readline(fdct,line) ;
   while (line[0] != EOF)

      /* THE INPUT SWITCH IS PUT ON THE FIRST CAPACITOR */
      { if (line[0]=='B' && line[1]=='S')
          { getnode(line,ONE,st1) ;
            getnode(line,TWO,st2) ;

            if (same(&table[0][0][0],st1) || same(&table[0][0][0],s
              { if (head=='L')
                  capac[2].swtch = !clockphase(line,ZERO) ;
                if (head=='M')
                  capac[1].swtch = !clockphase(line,ZERO) ; } }
```

```
      /* GET THE CAP PARAMETERS FROM THE INPUT FILE */
      if (line[1]=='C' && line[0]!=EOL)
        { sscanf(line,"%*s%[^)]%c%f",st1,st2,&capac[i].val) ;
          capac[i].v2 = capac[i].val ;
          getnode(line,ONE,&capac[i].fst[0]) ;
          getnode(line,TWO,&capac[i].sec[0]) ;
          capac[i].next = 1 ;
          capac[i].used = 0 ;
          capac[i].lay = ((capac[i].val>swap) ? 0 : 1) ;

          /* GET THE CLOCKS FOR THE SWITCHED CAPS */
          if (line[0]=='S')
            { capac[i].typ = 'S' ;
              flip[i] = clockphase(line,1) ;
              capac[i].swtch = clockphase(line,3) ;  }

          /* SAVE THE CAPACITOR TYPE */
          if (line[0]=='I') capac[i].typ = 'I' ;
          if (line[0]=='C') capac[i].typ = 'C' ;
          i++ ;       }

      /* INCREMENT THE STAGE COUNT */
      if (line[0]=='O') stages++ ;
      readline(fdct,line) ;     }

/* SAVE THE CLOCKING FOR THE INTEGRATING CAPS */
total = i-1 ;
for (i=1; i<=total; i++)
   if (capac[i].typ=='I')
     for (j=0; j<=total; j++)
        if (same(&capac[i].fst[0],&capac[j].fst[0]))
          capac[i].swtch = flip[j] ;

/* INITIALIZE THE SWITCH CONNECTION FLAGS */
for (i=1; i<=total; i++)
   capac[i].flag = 1 ;

/* CUSTOMIZE THE INPUT STAGE PARAMETERS */
if (head=='M')
  { capac[1].flag = 0 ;
    capac[1].used = 1 ;  }

if (head=='A')
  { capac[1].used = 1 ;
    capac[1].flag = 0 ;
    for (i=2; i<=total; i++)
       if (same(&capac[i].sec[0],&table[0][0][0]) &&
           capac[i].typ=='S' && capac[i].swtch!=capac[1].swtch)
         { capac[i].flag = 0 ;
```

```
                capac[i].used = 1 ;    }    }

   if (head=='S')
     { capac[1].flag = 0 ;
       capac[2].flag = 0 ;
       capac[2].used = 1 ;
       capac[3].flag = 0 ;
       capac[3].used = 1 ;
       capac[1].next = 2 ;
       capac[2].swtch = !capac[3].swtch ;    }
}

/*************************************************************/
/*                                                           */
/*  CAPARRAY FUNCTION  --  Generate Capacitor Array          */
/*                                                           */
/*     The caparray function generates the capacitor array   */
/*  by laying out the capacitors and then ensuring that      */
/*  the op-amp widths are comparable to the capacitor base   */
/*  widths. If this is not the case, spaces are inserted     */
/*  the capacitor array. The capacitor tip is added at the   */
/*  end to provide proper areas and mask shift immunity.     */
/*                                                           */
/*************************************************************/

caparray(fdcif)

FILE *fdcif ;                          /* .CIF FILE */

{  int xc1 ;                           /* WIDTH OF THE OP-AMPS */
   int xc2 ;                           /* WIDTH OF CAPACITORS */
   float space ;                       /* EXTRA SPACE NEEDED */

   /* LAYOUT THE CAPACITOR ARRAY */
   layout(fdcif,0.0) ;

   /* COMPARE WIDTHS OF OP-AMPS AND CAPACITORS */
   xc1 = stages*opamp.width ;
   xc2 = capac[total].hor[0]+capac[total].next*EDGE ;
   space = (xc1 - xc2 + 10) / (total-1) ;

   /* INSERT EXTRA SPACES IN THE CAP ARRAY IF NECCESSARY */
   if (space > 0)
     layout(fdcif,space) ;

   /* ADD THE TIP ON THE CAPACITORS */
   termonate(fdcif) ;
}

/*************************************************************/
```

```
/*                                                                  */
/*  LAYOUT FUNCTION  --  Layout Capacitors                          */
/*                                                                  */
/*     The layout function generates the capacitor array.           */
/*  The layout is generated by laying out each capacitor            */
/*  one plate at a time and inserting spaces until all the          */
/*  capacitors fit. The height of the array is user set.            */
/*                                                                  */
/******************************************************************/

layout(fdto,space)

FILE *fdto ;                              /* OUTPUT FILE */
float space ;                             /* EXTRA GAP REQUIRED */

{  char *buf ;                            /* LINE BUFFER */
   float fraction ;                       /* FRACTIONAL CAP SIZE */
   float gap2 ;                           /* GAP INCREMENT */
   float FEDGE = EDGE ;                   /* PLATE EDGE */
   int dir[2][4][2] ;                     /* PLATE DIRECTIONS */
   int x,y ;                              /* PLATE CO-ORDINATES */
   int i,j ;                              /* INDEX VARIABLES */
   int flag ;                             /* PLATE LAYOUT SUCCESS */
   int finish ;                           /* ARRAY LAYOUT SUCCESS */
   int horz ;                             /* HORIZ. LAYOUT CO-ORD */
   int left ;                             /* LEFT PLATE BOUNDARY */
   int gap ;                              /* PLATE GAP */
   int numb ;                             /* CURRENT CAPACITOR */
   int xdir,ydir ;                        /* PLATE DIRECTION */
   int index ;                            /* SEARCH INDEX */

   /* INITIALIZE PLATE DIRECTION UP-DOWN PREFERENCE */
   dir[0][0][0] =  0 ; dir[0][0][1] =  1 ;
   dir[0][1][0] =  0 ; dir[0][1][1] = -1 ;
   dir[0][2][0] =  1 ; dir[0][2][1] =  0 ;
   dir[0][3][0] = -1 ; dir[0][3][1] =  0 ;

   /* INITIALIZE PLATE DIRECTION RIGHT-LEFT PREFERENCE */
   dir[1][0][0] =  0 ; dir[1][0][1] = -1 ;
   dir[1][1][0] =  1 ; dir[1][1][1] =  0 ;
   dir[1][2][0] = -1 ; dir[1][2][1] =  0 ;
   dir[1][3][0] =  0 ; dir[1][3][1] =  1 ;

   /* COPY OP-AMP CELL TO CIF FILE */
   rewind(fdto) ;
   shoveon(fdto) ;

   /* INITIALIZE LAYOUT SYMBOL */
   buf = "DS 1 ;\n" ;
   writeline(fdto,buf) ;
```

```
/* EMPTY THE ARRAY */
for (i=0; i<2000; i++)
  for (j=0; j<TOPMAX; j++)
    grid[i][j] = 0 ;

/* INITIALIZE THE LAYOUT PARAMETERS */
horz = 1 ;
gap = 0 ;
gap2 = 0.0 ;

/* PLACE THE FIRST PLATE OF EACH CAPACITOR */
for (i=1; i<=total; i++)

   /* INITIALIZE THE FIRST PLATE CO-ORDS */
   { capac[i].pnt = 1 ;
     capac[i].hor[0] = horz ;
     capac[i].ver[0] = 1 ;

     /* FIND THE PLATE SIZE */
     capac[i].val = capac[i].v2 ;
     fraction = ((capac[i].val<1.0) ? capac[i].val : 1.0) ;
     capac[i].tipflag = ((fraction<1.0) ? 1 : 0) ;
     capac[i].val = capac[i].val - 1.0 ;

     /* FILL THE ARRAY */
     for (j=horz; j<horz+EDGE; j++)
        grid[j][1] = i ;

     /* ATTACH THE PLATE */
     attach1(horz,0,0,1,fraction,fdto,0) ;

     /* INCREMENT TO THE NEXT PLATE */
     gap2 = gap2 + space - gap ;
     gap = ((gap2>=EDGE) ? EDGE : 0) ;
     horz = horz + capac[i].next*EDGE + CAPGAP + gap ;  }

/* PLACE THE BOUNDARIES OF THE ARRAY */
horz = horz - CAPGAP ;
for (i=0; i<=horz; i++)
   for (j=0; j<TOPMAX; j++)
      if (i==0 || i==horz || j==0 || j==topper+2)
         grid[i][j] = MAXCAP - 1 ;

/* PLACE THE REST OF THE CAPACITOR PLATES */
finish = 0 ;
while (!finish)
   { finish = 1 ;

     /* CHECK EACH CAPACITOR IN TURN */
```

```
      for (numb=1; numb<=total; numb++)

        /* TRY EACH DIRECTION */
        { flag = 0 ;
          for (j=0; (j<4 && !flag); j++)

            /* CHECK EACH PREVIOUS PLATE */
            for (index=0; (index<capac[numb].pnt && !flag); index

              /* FIND THE PLATE LOCATION */
              { xdir = dir[capac[numb].lay][j][0] ;
                ydir = dir[capac[numb].lay][j][1] ;
                x = capac[numb].hor[index] ;
                y = capac[numb].ver[index] ;

                /* DECIDE IF THERE IS ROOM FOR THE PLATE */
                if (capac[numb].val > 0 && !flag &&
                      dope(x,xdir,y,ydir,numb,horz))

                  /* FIND THE PLATE SIZE */
                  { fraction = ((capac[numb].val>=1) ? 1.0 : capa
                    if (fraction<1.0 && ydir==1)
                      capac[numb].tipflag = 1 ;

                    /* FILL THE ARRAY ONLY IF NECCESSARY */
                    if (fraction*FEDGE+CAPGAP+2>EDGE && !xdir ||
                      { left = x + xdir*EDGE ;
                        for (i=left; i<left+EDGE; i++)
                           grid[i][y+ydir] = numb ;    }

                    /* UPDATE THE CAPACITOR PARAMETERS */
                    capac[numb].hor[capac[numb].pnt] = x+xdir*EDG
                    capac[numb].ver[capac[numb].pnt] = y+ydir;
                    capac[numb].val = capac[numb].val - 1.0 ;
                    capac[numb].pnt = capac[numb].pnt + 1 ;

                    /* PLACE THE PLATE AS APPROPRIATE */
                    if (fraction < LIMIT)
                      attach2(x,y,xdir,ydir,fraction,fdto,1) ;
                    else if (fraction < 1.0)
                      attach3(x,y,xdir,ydir,fraction,fdto,1) ;
                    else attach1(x,y,xdir,ydir,fraction,fdto,1) ;

                    /* SET THE FLAGS */
                    flag = 1 ;
                    if (capac[numb].val > 0)
                      finish = 0 ;       }   }   }   }

/* SEE IF THE ALL CAPACITORS WERE COMPLETED */
j = flag = 0 ;
```

```
   while (j <= total)
      { if (capac[j].val>0)

          /* ALLOCATE MORE SPACE */
          { capac[j++].next += 1 ;   /* SKIP NEXT ONE */
            flag = 1 ;  }
        j = j + 1 ;           }

   /* DO IT AGAIN IF NECCESSARY */
   if (flag)
     { printf("%s\n","ITERATION FAILURE") ;
       layout(fdto,space) ;          }
}

/*************************************************************/
/*                                                           */
/*  TERMONATE FUNCTION  --  Place a Tip on the Capacitor     */
/*                                                           */
/*     The termonate function places a tip on the caps not   */
/*  yet having one. The tip ensures proper area ratios and   */
/*  immunity to P1-P2 mask shifts.                           */
/*                                                           */
/*************************************************************/

termonate(fdto)

FILE *fdto ;                    /* OUTPUT FILE */

{  int i,j ;                    /* INDEX VARIABLES */
   int xtop,ytop ;              /* HIGHEST CO-ORDS */
   int xc,yc ;                  /* TIP LOCATION */

   /* CHECK EACH CAPACITOR FOR PREVIOUS TIP PLACEMENT */
   for (i=1; i<=total; i++)
      if (!capac[i].tipflag)

        /* FIND THE HIGHEST PLATE OF THE CAPACITOR */
        { ytop = -1000 ;
          for (j=0; j<capac[i].pnt; j++)
             if (capac[i].ver[j] > ytop)
               { ytop = capac[i].ver[j] ;
                 xtop = capac[i].hor[j] ;  }

          /* INCREMENT TO THE TOP OF THAT PLATE */
          xc = (xtop + EDGE/2.0)*U ;
          yc = (ytop*EDGE + 1)*U ;

          /* PLACE THE TIP */
          BOX(fdto,P2B,2*U,2*U,xc,yc) ;  }
}
```

```
/*************************************************************/
/*                                                           */
/*  WIRE1 FUNCTION  --  Connect Input Nodes                  */
/*                                                           */
/*     The wire1 function connects all the op-amp input      */
/*  nodes to the appropriate capacitors.                     */
/*                                                           */
/*************************************************************/

wire1(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int i,k ;                        /* INDEX VARIABLES */
   int j ;                          /* WIRING LEVEL */
   int least[2] ;                   /* LEFTMOST POINT OF WIRE */
   int most[2] ;                    /* RIGHTMOST POINT OF WIRE */
   int xc[3],yc[3] ;                /* WIRING CENTER CO-ORDS */

   /* INITIALIZE THE VERTICAL CO-ORDS */
   yc[1] = level[3] = U ;

   /* EMPTY THE WIRING ARRAY */
   for (i=0; i<10; i++)
      for (j=0; j<300; j++)
         path[i][j] = 0 ;

   /* WIRE UP THE OP-AMP INPUT NODES */
   for (i=start; table[2][i][0]!=EOW; i+=2)

      /* INITIALIZE THE WIRE END-POINTS */
      { least[0] = 3*(total+1) ;
        most[0] = 0 ;

        /* CHECK ALL CAPACITORS FOR A CONNECTION */
        for (j=0;j<=total; j++)
           { xc[0]  = (capac[j].hor[0]+EDGE/2+8)*U ;
             xc[2] = xc[0] - 16*U ;

             /* IGNORE SWITCHED CAPACITORS */
             if (capac[j].typ=='S') ;

             /* IF THE FIRST NODE IS THE SAME... */
             else if (same(&table[2][i][0],&capac[j].fst[0]))
               { if (3*j<least[0])
                    { least[0] = 3*j ; least[1] = xc[2] ; }
                 else if (3*j>most[0])
                    { most[0] =  3*j ; most[1] =  xc[2] ; } }
```

```
            /* IF THE SECOND NODE IS THE SAME... */
            else if (same(&table[2][i][0],&capac[j].sec[0]))
              { if (3*j+2<least[0])
                  { least[0] = 3*j+2 ; least[1] = xc[0] ; }
                else if (2*j+1>most[0])
                  { most[0] =  3*j+2 ; most[1] =  xc[0] ; } } }

        if (most[0]>least[0])
          { /* FIND WIRE LEVEL AND PLACE WIRE */
            j = wireslot(least,most) ;
            yc[0] = (-5*j-3)*U ;
            WIRE(fdto,M1W,w,least[1],yc[0],most[1],yc[0]) ; }

        /* CHECK ALL CAPACITORS */
        for (k=0; k<=total; k++)
           { xc[0] = (capac[k].hor[0]+EDGE/2+8)*U ;
             xc[2] = xc[0] - 16*U ;

             /* CONNECT NODES */
             if (same(&table[2][i][0],&capac[k].fst[0]) ||
                 same(&table[2][i][0],&capac[k].sec[0]))

               /* PLACE CAPACITOR CONNECTIONS */
               { if (same(&table[2][i][0],&capac[k].fst[0]))

                   /* SAVE VERTICAL CO-ORD FOR SWITCH CONNECTION */
                   { capac[k].ver[2] = ((most[0]>least[0]) ? yc[0]/
                     xc[1] = xc[2] ;  }

                 else xc[1] = xc[0] ;

                 if (capac[k].typ!='S' && most[0]>least[0])
                   { WIRE(fdto,P2W,w,xc[1],yc[1],xc[1],yc[0]) ;
                     mp2(xc[1],yc[0],fdto) ;  }  }  }  }
}

/*************************************************************/
/*                                                           */
/*  WIRE2 FUNCTION  --  Connect Output Nodes                 */
/*                                                           */
/*     The wire2 function connects the op-amp output         */
/*  nodes to switched capacitor switches.                    */
/*                                                           */
/*************************************************************/

wire2(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int i,k ;                        /* INDEX VARIABLES */
```

```
int j ;                          /* WIRING LEVEL */
int least[2] ;                   /* LEFTMOST POINT OF WIRE */
int most[2] ;                    /* RIGHTMOST POINT OF WIRE */
int xc[4],yc[3] ;                /* WIRING CENTRE CO-ORDS */

/* INITIALIZE THE WIRING LEVELS */
yc[1] = U ;

/* CONNECT THE OP-AMP OUTPUT NODES */
for (i=1+start; table[2][i][0]!=EOW; i+=2)

   /* INITIALIZE THE WIRE BOUNDS */
   { least[0] = 3*(total+1) ;
     most[0] = 0 ;

     /* CHECK EACH CAP FOR THE NODE */
     for (j=0;j<=total; j++)
        { xc[0] = (capac[j].hor[0]+EDGE/2+8)*U ;
          xc[2] = xc[0] - 16*U ;
          xc[3] = (capac[total].hor[0]+capac[total].next*EDGE)*U

          /* CHECK THE FIRST NODE */
          if (capac[k].typ!='S' &&
               same(&table[2][i][0],&capac[j].fst[0]))
            { if (3*j<least[0])
                { least[0] = 3*j ; least[1] = xc[2] ;  }
              else if (3*j>most[0])
                { most[0] =  3*j ; most[1] =  xc[2] ;  }  }

          /* CHECK THE SECOND NODE */
          else if (same(&table[2][i][0],&capac[j].sec[0]))
            { if ((capac[j].typ=='S' && capac[j].used==1)
                        || capac[j].typ!='S')
                { if (3*j+2<least[0])
                    { least[0] = 3*j+2 ; least[1] = xc[0] ; }
                  else if (3*j+2>most[0])
                    { most[0] =  3*j+2 ; most[1] =  xc[0] ; } }

                /* EXTEND OUTPUT NODE TO THE FAR RIGHT */
                if (same(&table[0][1][0],&table[2][i][0]))
                  { most[0] = 3*total+2 ; most[1] = xc[3] ; } }

     /* CALCULATE WIRE LEVEL */
     j = wireslot(least,most) ;
     yc[0] = (-5*j-3)*U ;
     WIRE(fdto,M1W,w,least[1],yc[0],most[1],yc[0]) ;

     /* SAVE THE OUTPUT WIRING LEVEL */
     if (same(&table[0][1][0],&table[2][i][0]))
       level[9] = yc[0] ;
```

```
        /* CONNECT THE APPROPRIATE CAPS */
        for (k=0; k<=total; k++)
           { xc[0] = (capac[k].hor[0]+EDGE/2+8)*U ;
             xc[2] = xc[0] - 16*U ;

             /* CONNECT NODES */
             if (same(&table[2][i][0],&capac[k].fst[0]) ||
                 same(&table[2][i][0],&capac[k].sec[0]))

               /* FIRST NODE */
               { if (same(&table[2][i][0],&capac[k].fst[0]))
                   xc[1] = xc[2] ;
                 else xc[1] = xc[0] ;

                 /* SAVE LEVEL FOR SWITCH CONNECTION */
                 if (capac[k].typ=='S')
                   capac[k].ver[1] = yc[0]/10 ;

                 /* ELSE PLACE CONTACTS AND CONNECT THEM */
                 else { mp(xc[1],yc[0],fdto) ;
                   WIRE(fdto,PlW,w,xc[1],yc[1],xc[1],yc[0]) ; } } }
}

/*************************************************************/
/*                                                           */
/*  WIRE4 FUNCTION  --  Connect Shared Output Nodes          */
/*                                                           */
/*     The wire4 function connects the op-amp output         */
/*  nodes to switched capacitor switches.                    */
/*                                                           */
/*************************************************************/

wire4(fdto,theone)

FILE *fdto ;                    /* OUTPUT FILE */
int theone ;

{  int i,k ;                    /* INDEX VARIABLES */
   int j ;                      /* WIRING LEVEL */
   int least[2] ;               /* LEFTMOST POINT OF WIRE */
   int most[2] ;                /* RIGHTMOST POINT OF WIRE */
   int xc[3],yc[3] ;            /* WIRING CENTRE CO-ORDS */

   /* INITIALIZE THE WIRING LEVELS */
   yc[1] = U ;

   /* CONNECT THE OP-AMP OUTPUT NODES */
   for (i=1+start; table[2][i][0]!=EOW; i+=2)
```

```
        /* INITIALIZE THE WIRE BOUNDS */
        { least[0] = 3*(total+1) ;
          most[0] = 0 ;

          /* CHECK EACH CAP FOR THE NODE */
          for (j=0;j<=total; j++)
             { xc[0] = (capac[j].hor[0]+EDGE/2)*U ;

               /* CHECK THE FIRST NODE */
               if (capac[j].typ=='S' &&
                     same(&table[2][i][0],&capac[j].sec[0]))
                 if (capac[j].swtch==theone)
                   { if (3*j+1<least[0])
                       { least[0] = 3*j+1 ; least[1] = xc[0] ; }
                     else if (3*j+1>most[0])
                       { most[0] =  3*j+1 ; most[1] =  xc[0] ; } } }

          if (most[0]>least[0])
            { /* FIND WIRE LEVEL AND PLACE WIRE */
              j = wireslot(least,most) ;
              yc[0] = (-5*j-3)*U ;
              WIRE(fdto,M1W,w,least[1],yc[0],most[1],yc[0]) ;

              /* CONNECT THE APPROPRIATE CAPS */
              for (k=0; k<=total; k++)
                 { xc[0] = (capac[k].hor[0]+EDGE/2)*U ;

                   /* CONNECT NODES */
                   if (capac[k].typ=='S' &&
                         same(&table[2][i][0],&capac[k].sec[0]))
                     if (capac[k].swtch==theone)

                       /* PLACE CONTACTS AND CONNECT THEM */
                       { mp2(xc[0],yc[0],fdto) ;
                         capac[k].hor[2] = yc[0]/10 ;
                         WIRE(fdto,P2W,w,xc[0],yc[1],xc[0],yc[0]) ; }
}

/*************************************************************/
/*                                                           */
/*  WIRE3 FUNCTION  --  Connect Hidden Nodes                 */
/*                                                           */
/*     The wire3 function connects the hidden nodes          */
/*  between integrating and switched capacitors.             */
/*                                                           */
/*************************************************************/

wire3(fdto)

FILE *fdto ;                    /* OUTPUT FILE */
```

```
{   int i,k ;                       /* INDEX VARIABLES */
    int j ;                         /* WIRE LEVEL */
    int least[2] ;                  /* LEFTMOST POINT OF WIRE */
    int most[2] ;                   /* RIGHTMOST POINT OF WIRE */
    int xc[3],yc[3] ;               /* WIRE CONNECTION POINTS */

    /* INITIALIZE THE WIRING LEVELS */
    yc[1] = U ;

    /* CONNECT THE HIDDEN NODES */
    for (i=start; table[2][i][0]!=EOW; i+=2)

       /* INITIALIZE THE WIRE BOUNDS */
       { least[0] = 3*(total+1) ;
         most[0] = 0 ;

         /* CHECK EVERY CAPACITOR */
         for (j=0;j<=total; j++)
            { xc[1] = (capac[j].hor[0]+EDGE/2)*U ;
              xc[2] = xc[1] - 8*U ;

              /* FIRST NODE ON SWITCHED CAPS */
              if (capac[j].typ=='S' &&
                     same(&table[2][i][0],&capac[j].fst[0]))
                { if (3*j<least[0])
                    { least[0] = 3*j ; least[1] = xc[2] ; }
                  else if (3*j>most[0])
                    { most[0] =  3*j ; most[1] =  xc[2] ; } }

              /* FIRST NODE ON INTEGRATING CAPS */
              else if (capac[j].typ=='I' &&
                           same(&table[2][i][0],&capac[j].fst[0]))
                { if (3*j+1<least[0])
                    { least[0] = 3*j+1 ; least[1] = xc[1] ; }
                  else if (3*j+1>most[0])
                    { most[0] =  3*j+1 ; most[1] =  xc[1] ; } } }

         /* FIND THE WIRE LEVEL */
         j = wireslot(least,most) ;
         yc[0] = (-5*j-3)*U ;
         WIRE(fdto,M1W,w,least[1],yc[0],most[1],yc[0]) ;

         /* CONNECT THE CAPACITORS TO THE WIRE */
         for (k=0; k<=total; k++)
            { xc[1] = (capac[k].hor[0]+EDGE/2)*U ;
              xc[2] = xc[1] - 8*U ;

              /* FIRST NODE CONNECTIONS ONLY */
              if (same(&table[2][i][0],&capac[k].fst[0]))
```

```
                /* CONNECT SWITCHED CAPS */
                { if (capac[k].typ=='S')

                    /* PLACE AND CONNECT CONTACTS */
                    { mp(xc[2],yc[0],fdto) ;
                      WIRE (fdto,PlW,w,xc[2],yc[1],xc[2],yc[0]) ; }

                  /* PLACE CONTACTS FOR INTEGRATING CAPS */
                  else if (capac[k].typ=='I')
                     capac[k].ver[1] = yc[0]/10 ;  }  }  }
}

/*************************************************************/
/*                                                           */
/*  STREAK FUNCTION  --  Connect the Switches                */
/*                                                           */
/*     The streak function places and connects the           */
/*  switches to the previous wiring.                         */
/*                                                           */
/*************************************************************/

streak(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int i ;                          /* INDEX VARIABLE */
   int xc,xm,xp ;                   /* HORIZ. WIRING POINTS */
   int yc[8] ;                      /* VERTICAL WIRING LEVELS */

   yc[0] = level[1] ;               /* TOP OF SWITCH */
   yc[1] =  U ;                     /* CONTACT FOR CAPACITOR */
   yc[3] = level[1] - 16*U ;        /* BOTTOM OF SWITCH */
   yc[4] = level[1] - 33*U ;        /* GROUND LINE */
   yc[2] = level[1] + 12*U ;        /* TOP CONNECTION LEVEL */

   /* PLACE AND CONNECT THE SWITCHES */
   for (i=0; i<=total; i++)

      /* INITIALIZE THE CONNECTION POINTS */
      { xc = (capac[i].hor[0] + EDGE/2)*U ;
        xm = xc - 8*U ;
        xp = xc + 8*U ;
        yc[5] = 10*capac[i].ver[1] ;
        yc[6] = 10*capac[i].ver[2] ;
        yc[7] = 10*capac[i].hor[2] ;

        /* SWITCHED CAPACITOR */
        if (capac[i].typ == 'S')
          { if (capac[i].used==1)
```

```
                { switchplace(xc,level[1],i,fdto) ;
                  WIRE(fdto,M1W,w,xm,yc[3],xm,yc[4]) ;

                  if (yc[7]==level[1]+17*U)
                    { WIRE(fdto,P2W,w,xc,yc[1],xc,yc[7]) ;
                      WIRE(fdto,M1W,w,xc,yc[7],xc,yc[0]) ; }

                  else { mp2(xc,yc[2],fdto) ;
                      WIRE(fdto,P2W,w,xc,yc[1],xc,yc[2]) ;
                      WIRE(fdto,M1W,w,xc,yc[2],xc,yc[0]) ; } }

              if (capac[i].flag && capac[i].used)
                { if (yc[5]!=yc[2]+5*U)
                    { mp(xp,yc[5],fdto) ;
                      mp(xp,yc[2],fdto) ;
                      WIRE(fdto,M1W,w,xp,yc[2],xp,yc[0]) ;
                      WIRE(fdto,P1W,w,xp,yc[5],xp,yc[2]) ; }

                  else WIRE(fdto,M1W,w,xp,yc[5],xp,yc[0]) ; } }

          /* INTEGRATING CAPACITOR */
          else if (capac[i].typ=='I')
            { switchplace(xc,level[1],i,fdto) ;
              WIRE(fdto,M1W,w,xp,yc[3],xp,yc[4]) ;

              if (yc[5]!=yc[2]+5*U)
                { mp(xc,yc[5],fdto) ;
                  mp(xc,yc[2],fdto) ;
                  WIRE(fdto,M1W,w,xc,yc[2],xc,yc[0]) ;
                  WIRE(fdto,P1W,w,xc,yc[5],xc,yc[2]) ;  }

              else WIRE(fdto,M1W,w,xc,yc[5],xc,yc[0]) ;

              mp2(xm,yc[2],fdto) ;
              WIRE(fdto,M1W,w,xm,yc[2],xm,yc[0]) ;

              if (yc[6]==(-10))
                   WIRE(fdto,P2W,w,xm,yc[1],xm,yc[2]) ;
              else WIRE(fdto,P2W,w,xm,yc[6],xm,yc[2]) ;  }  }
}

/*************************************************************/
/*                                                           */
/*  PREOP FUNCTION  --  Initialize OP-AMP Data               */
/*                                                           */
/*      The preop function initializes the op-amp data       */
/*  and decides where the op-amps are to be connected to     */
/*  the switches. It also decides which op-amps to mirror.   */
/*                                                           */
/*************************************************************/
```

```
preop(fdto)

FILE *fdto ;                    /* OUTPUT FILE */

{   int length ;                /* LENGTH OF CAP ARRAY */
    int gap ;                   /* GAP BETWEEN OP-AMPS */
    int clock ;                 /* SC CLOCK */
    int stox ;                  /* INDEX */
    int flag ;                  /* FOUND FLAG */
    int count ;                 /* NUMBER OF SC'S FOR CONNECTION */
    int middle ;                /* MIDPOINT OF OP-AMP CONNECTS */
    int master ;                /* CLOSEST SC CONTACT */
    int store ;                 /* SC CONNECTION POINT */
    int i,j ;                   /* INDEX VARIABLES */
    int x ;                     /* OP-AMP PLACEMENT POINT */
    float gap2 ;                /* REQUIRED GAP BETWEEN OP-AMPS */
    float inc ;                 /* GAP INCREMENT */

    /* FIND ANY EXTRA SPACING REQUIRED BETWEEN THE OP-AMPS */
    length = capac[total].hor[0] + capac[total].next*EDGE + 4 ;
    inc = (length - stages * opamp.width) / (stages - 1) ;
    if (inc < 0) inc = 0 ;

    /* INITIALIZE THE PLACEMENT */
    x = -1 ;
    gap = 0 ;
    gap2 = 0.0 ;

    /* INITIALIZE THE INPUT STAGE OP-AMP FOR SEDRA */
    if (start==2)
      { whip[0].out = x + opamp.con[0] ;
        whip[1].out = x + opamp.con[2] ;
        whip[0].cap = (capac[1].hor[0]+EDGE/2)*U ;
        whip[1].cap = (capac[2].hor[0]+EDGE/2+8)*U ;
        whip[0].base = x ;
        whip[0].gnd = x + opamp.con[1] ;
        whip[0].mirror = 0 ;
        x = x + opamp.width + inc ;     }

    /* PROCESS THE REST OF THE FILTER */
    for (i=start; i<stages+stages; i++)
       { whip[i].base = x ;

         /* OP-AMP INPUT NODE */
         if (!(i%2))

           /* LOOK FOR AN INTEGRATING CAPACITOR */
           { for (j=1; j<=total; j++)
                if (capac[j].typ=='I' &&
```

```
                    same(&capac[j].fst[0],&table[2][i][0]))
                { whip[i].cap = capac[j].hor[0] + EDGE/2 - 8 ;
                  whip[i].out = x + opamp.con[0] ;
                  whip[i].gnd = x + opamp.con[1] ;    }     }

        /* OP-AMP OUTPUT NODE */
        else { master = 20000 ;
            whip[i].out = x + opamp.con[2] ;
            whip[i].gnd = x + opamp.con[1] ;
            middle = (whip[i].out + whip[i-1].out)/2 ;

            /* PICK CLOSEST SC CONNECTION POINT */
            for (j=1; j<=total; j++)
               if (capac[j].typ=='S' &&
                     same(&capac[j].sec[0],&table[2][i][0]))
                 { store = abs(middle - (capac[j].hor[0]+EDGE/2+8))
                   if (store < master)
                     { whip[i].cap = (capac[j].hor[0]+EDGE/2+8) ;
                       clock = capac[j].swtch ;
                       stox = j ;
                       master = store ;    }     }

            /* PICK REVERSE CLOCKING SC */
            flag = 1 ;
            capac[stox].used = 1 ;
            for (j=1; j<=total && flag; j++)
               if (capac[j].typ=='S' &&
                     same(&capac[j].sec[0],&table[2][i][0]))
                 if (capac[j].swtch!=clock)
                   { capac[j].used = 1 ;
                     flag = 0 ;  }

            /* MIRROR OP-AMP IF NECCESSARY */
            if (turn(i-1))
              { whip[i-1].out = whip[i-1].out + opamp.width - 2*opa
                whip[i].out = whip[i].out + opamp.width - 2*opamp.c
                whip[i-1].gnd = whip[i-1].gnd + opamp.width - 2*opa
                whip[i-1].mirror = 1 ;       }

            /* INCREMENT THE PLACEMENT */
            gap2 = gap2 + inc - gap ;
            gap = gap2 ;
            x = x + opamp.width + gap ;    }     }
}

/*************************************************************/
/*                                                           */
/*  WIRE5 FUNCTION  --  Connect Op-Amp Inputs/Outputs        */
/*                                                           */
/*     The wire5 function connects the -inputs and the       */
```

```
/*  outputs of the op-amps to the switches.                    */
/*                                                             */
/*************************************************************/

wire5(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int yc2,yc3,yc4,yc5 ;            /* VERTICAL CONNECTS */
   int xa,xb,xc,a,b ;               /* HORIZONTAL CONNECTS */
   int dir ;                        /* CONNECTION DIRECTION */
   int adj ;                        /* CAP CONTACT ADJUST */
   int flag,f2 ;                    /* WIRING PLACEMENT FLAGS */
   int i,j,m ;                      /* INDEX VARIABLES */
   int k ;                          /* WIRING LEVEL */

   /* EMPTY THE WIRING GRID */
   for (i=0; i<5; i++)
      for (j=0; j<5000; j++)
         path2[i][j] = 0 ;

   /* INITIALIZE THE VERTICAL CONNECTION LEVELS */
   yc2 = level[1] - 33*U ;
   yc3 = level[1] - 27*U ;
   yc4 = level[1] - 18*U ;

   level[6] = 0 ;

   /* PROCESS ONE OP-AMP CONNECTION AT A TIME */
   for (i=start; i<stages+stages; i++)

      /* STORE THE CONNECTION POINTS */
      { xa = whip[i].out*U ;
        xb = whip[i].cap*U ;

        /* ADJUSTMENT DISTANCE FOR CAP CONTACTS */
        adj = ((i%2) ? -12 : 12) ;

        /* FIND LEFT AND RIGHT CONNECTION POINTS */
        if (whip[i].out > whip[i].cap)
          { a = whip[i].cap ; b = whip[i].out ; dir = -1 ; }
        else { a = whip[i].out ; b = whip[i].cap ; dir = 1 ; }

        /* SEE IF WIRE CAN BE DRAWN BELOW THE SWITCHES */
        flag = 1 ;
        for (j=start; j<stages+stages; j++)
           { if (i!=j && whip[j].cap > a-5 && whip[j].cap < b+5)
                flag = 0 ;      }

        /* SEE IF NO OTHER WIRE IS THERE */
```

```
if (flag)
  for (k=a-3; k<=b+3; k++)
     if (path2[0][k]!=0)
       flag = 0  ;

/* PLACE WIRE BELOW THE SWITCHES */
if (flag)

  /* ONLY DRAW HORIZONTAL WIRE IF NECCESSARY */
  { if (abs(xa-xb) > 2*U)
      WIRE(fdto,P1W,w,xa,yc3,xb,yc3) ;

    /* PLACE CONTACT ON THE SWITCH */
    mp(xb,yc3,fdto) ;
    WIRE(fdto,M1W,w,xb,yc3,xb,yc4) ;

    /* FILL THE GRID */
    for (k=a; k<=b; k++)
       path2[0][k] = 1 ;

    /* SAVE WIRE PARAMETERS FOR OP-AMP CONNECTION */
    whip[i].cross = whip[i].out ;
    whip[i].level = yc3 ;      }

/* DRAW THE WIRE BELOW THE VSS LINE */
else { whip[i].cross = whip[i].cap ;

    /* DECIDE IF THE CROSSING POINT SHOULD BE MOVED */
    for (k=start; k<stages+stages; k++)
    if (abs(whip[i].cap-whip[k].gnd) < 5 ||
            abs(whip[i].cap-whip[k].out)<5)
      whip[i].cross = whip[i].cap + adj ;

    /* FIND LEFT AND RIGHT POINTS */
    if (whip[i].cross > whip[i].out)
         { a = whip[i].out ; b = whip[i].cross ; }
    else { a = whip[i].cross ; b = whip[i].out ; }

    /* FIND A WIRE SLOT */
    f2 = 1 ;
    for (k=1; f2; k++)
       { flag = 1 ;
         for (m=a-3; m<=b+3; m++)
            if (path2[k][m]!=0)
              flag = 0 ;
         if (flag) f2 = 0 ;     }
    k-- ;

    /* CALCULATE THE WIRE LEVEL */
    yc5 = level[1] - (45+6*(k-1))*U ;
```

```
            if (k-1>level[6]) level[6] = k-1 ;

            /* FILL THE GRID */
            for (m=a-2; m<=b+2; m++)
               path2[k][m] = 1 ;

            /* IF NO ADJUSTMENT */
            if (whip[i].cross==whip[i].cap)

              /* PLACE SWITCH CONTACT AND CROSS BUS LINES */
              { mp(xb,yc3,fdto) ;
                WIRE(fdto,M1W,w,xb,yc3,xb,yc4) ;
                WIRE(fdto,P1W,w,xb,yc5,xb,yc3) ;
                xc = xa ;      }

            /* WIRE CROSSING HAS BEEN MOVED */
            else { xa = whip[i].cap*U ;
                xb = whip[i].cross*U ;
                xc = whip[i].out*U ;

                /* CONNECT SWITCH AND WIRE IT */
                mp(xa,yc3,fdto) ;
                WIRE(fdto,M1W,w,xa,yc3,xa,yc4) ;
                WIRE(fdto,P1W,w,xb,yc5,xb,yc3) ;
                WIRE(fdto,P1W,w,xa,yc3,xb,yc3) ;     }

            /* PLACE METAL WIRE WITH CONTACTS IF REQUIRED */
            if (abs(xc-xb) > 6*U)
              { mp(xb,yc5,fdto) ;
                mp(xc,yc5,fdto) ;
                WIRE(fdto,M1W,w,xb,yc5,xc,yc5) ;    }

            /* OR DRAW A SHORT POLY LINKING WIRE */
            else WIRE(fdto,P1W,w,xb,yc5,xc,yc5) ;

            /* SAVE WIRING PARAMETERS FOR OP-AMP CONNECTION */
            whip[i].level = yc5 ;     }     }
}

/*************************************************************/
/*                                                           */
/*  WIRE6 FUNCTION  --  Wire the Ground Connections          */
/*                                                           */
/*      The wire6 function connects the +input of the        */
/*  op-amps to the ground wire. It will curve around any     */
/*  obstacles.                                               */
/*                                                           */
/*************************************************************/

wire6(fdto)
```

```
FILE *fdto ;                              /* OUTPUT FILE */

{  int yc[6] ;                            /* VERTICAL CONNECTS */
   int xa,xb,xb2,a,b ;                    /* HORIZONTAL CONNECTS */
   int dir ;                              /* WIRING DIRECTION */
   int flag ;                             /* WIRING FLAG */
   int j ;                                /* WIRING LEVEL */
   int i,k,m ;                            /* INDEX VARIABLES */

   /* INITIALIZE THE VERTICAL CONNECT LEVELS */
   yc[2] = level[1] - 33*U ;
   yc[3] = level[1] - 27*U ;
   yc[4] = level[1] - 18*U ;

   /* SET THE SEDRA CROSSING POINTS FOR THE FIRST OP-AMP */
   if (start==2)
     { whip[0].cross = capac[2].hor[0]+EDGE/2+8 ;
       whip[1].cross = capac[2].hor[0]+EDGE/2+8 ;
       whip[0].gndcross = opamp.con[1] - 1 ;       }

   /* PROCESS ONE GROUND CONNECTION AT A TIME */
   for (i=start/2; i<stages; i++)

      /* FIND WIRE DIRECTION */
      { dir = ((whip[2*i].gnd < whip[2*i].out) ? -1 : 1) ;

        /* TRY TO WIRE THE GROUND CONNECTION */
        flag = 1 ;
        for (k=0; flag; k=k+dir)
           { flag = 0 ;

             /* LOOK FOR ANY OBSTACLES */
             for (j=start; j<stages+stages; j++)
                { if (whip[j].level!=yc[3])
                     if (abs(whip[2*i].gnd+k-whip[j].cross) < 5)
                       flag = 1 ;     }   }

        /* FINAL CROSSING POINT */
        xb = whip[2*i].gnd*U ;
        k = k - dir ;
        whip[2*i].gndcross = whip[2*i].gnd + k ;

        /* PLACE THE DISPLACED WIRE */
        if (k!=0)
          { xb2 = xb + k*U ;

            /* FIND LEFT AND RIGHT POINTS */
            if (k > 0)
                 { a = whip[2*i].gnd ; b = a + k ;  }
```

```
            else { b = whip[2*i].gnd ; a = b + k ;  }

            /* CHECK FOR SPACE IN THE GRID */
            flag = 1 ;
            for (m=a-2; m<=b+2; m++)
               if (path2[level[6]][m]!=0)
                 flag = 0 ;

            /* MOVE TO LOWEST LEVEL IF NECCESSARY */
            if (!flag) level[6] += 1 ;

            /* CALCULATE WIRING LEVEL */
            yc[5] = level[1] - (45+6*level[6])*U ;
            whip[2*i].gndlevel = yc[5] ;

            /* CONNECT THE GROUND WIRE TO THE INPUT */
            mp(xb2,yc[2],fdto) ;
            WIRE(fdto,P1W,w,xb2,yc[5],xb2,yc[2]) ;
            WIRE(fdto,P1W,w,xb,yc[5],xb2,yc[5]) ;    }

         /* DRAW STRAIGHT WIRE AND CONTACT OTHERWISE */
         else { mp(xb,yc[2],fdto) ;
                whip[2*i].gndlevel = yc[2] ;    }  }
}

/*************************************************************/
/*                                                           */
/*  AMPSON FUNCTION  --  Insert the Op-Amps                  */
/*                                                           */
/*     The ampson function places and connects the op-amps   */
/*  to the previously placed wires.                          */
/*                                                           */
/*************************************************************/

ampson(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int i ;                          /* INDEX VARIABLE */
   int yc[2] ;                      /* VERTICAL CONNECTS */
   int xa ;                         /* HORIZONTAL CONNECTS */
   char *norm ;                     /* CELL PLACEMENT COMMAND */

   /* INITIALIZE THE WIRING LEVELS */
   yc[0] = level[1] - (49 + 6*level[6] + opamp.level)*U ;
   yc[1] = yc[0] + opamp.level*U ;
   level[2] = yc[0] ;

   /* CONNECT THE -INPUTS AND OUTPUTS */
   for (i=start; i<stages+stages; i++)
```

```
      { xa = whip[i].out*U ;
        WIRE(fdto,P1W,w,xa,yc[1],xa,whip[i].level) ;  }

   /* PLACE THE OP-AMPS */
   for (i=0; i<stages; i++)

      /* CONNECT THE +INPUTS */
      { if (i>=start/2)
          { xa = whip[2*i].gnd*U ;
            WIRE(fdto,P1W,w,xa,yc[1],xa,whip[2*i].gndlevel) ; }

        /* MIRRORED OP-AMP */
        if (!whip[2*i].mirror)
          { xa = (whip[2*i].base - opamp.xorg)*U ;
            norm = "C2 T" ; }

        /* NORMAL OP-AMP */
        else { xa = (whip[2*i].base + opamp.width + opamp.xorg)*U ;
            norm = "C2 MX T" ; }

        /* PLACE THE OP-AMP */
        cellplace(fdto,norm,xa,yc[0]) ; }
}

/*************************************************************/
/*                                                           */
/*  WRAPUP FUNCTION  --  Layout Global Features              */
/*                                                           */
/*     The wrapup function places the p-wells, the clocks,   */
/*  the bus lines, and the substrate contacts.               */
/*                                                           */
/*************************************************************/

wrapup(fdto)

FILE *fdto ;                        /* OUTPUT FILE */

{  int i,j ;                        /* INDEX VARIABLES */
   int wl ;                         /* WIRE WIDTH */
   int xa,xb ;                      /* HORIZONTAL CONNECTS */
   int yc1,yc2 ;                    /* VERTICAL CONNECTS */
   int xc,yc ;                      /* BOX CENTRE */
   int xedge,yedge ;                /* BOX EDGES */
   int h[3] ;                       /* SUBSTRATE CONTACT CHECKS */
   int flag ;                       /* SUBSTRATE CONTACT PLACEMENT */
   int xl,xr ;                      /* HORIZONTAL LAYOUT BOUNDS */

   /* SET THE HORIZONTAL BOUNDS OF THE LAYOUT */
   xl = -5*U ;
   xa = (capac[total].hor[0]+capac[total].next*EDGE)*U ;
```

```
if (xa+2*U > (whip[2*stages-2].base + opamp.width)*U)
  xr = xa + 6*U ;
else xr = (whip[2*stages-2].base + opamp.width + 4)*U ;

/* PLACE THE OUTPUT CONTACT */
if (xa < xr-8*U)
  WIRE(fdto,M1W,w,xa,level[9],xr-8*U,level[9]) ;
WIRE(fdto,P1W,w,xr-8*U,level[9],xr+6*U,level[9]) ;
mp(xr+6*U,level[9],fdto) ;
mp(xr-8*U,level[9],fdto) ;

/* CONNECT CLOCK1 LINE */
yc = level[1] + 6*U ;
WIRE(fdto,P1W,w,xl,yc,xr+6*U,yc) ;
mp(xr+6*U,yc-U,fdto) ;
mp(xl,yc-U,fdto) ;

/* CONNECT CLOCK2 LINE */
yc = level[1] - 6*U ;
WIRE(fdto,P1W,w,xl,yc,xr+6*U,yc) ;
mp(xr+6*U,yc+U,fdto) ;
mp(xl,yc+U,fdto) ;

/* CONNECT CLOCK 3 LINE */
yc = level[1] - 10*U ;
WIRE(fdto,P1W,w,xl,yc,xr+6*U,yc) ;
mp(xr+6*U,yc-U,fdto) ;
mp(xl,yc-U,fdto) ;

/* CONNECT CLOCK4 LINE */
yc = level[1] - 22*U ;
WIRE(fdto,P1W,w,xl,yc,xr+6*U,yc) ;
mp(xr+6*U,yc+U,fdto) ;
mp(xl,yc+U,fdto) ;

/* PLACE THE GROUND LINE */
yc = level[1] - 33*U ;
WIRE(fdto,M1W,2*w,xl+6*U,yc,xr,yc) ;

/* PLACE THE VSS LINE */
yc = level[1] - 39*U ;
WIRE(fdto,M1W,2*w,xl+6*U,yc,xr,yc) ;

/* PLACE THE OP-AMP VSS LINE */
yc = level[2] + opamp.vos[0]*U ;
wl = opamp.wid[0]*U ;
WIRE(fdto,M1W,wl,xl+4*U+wl/2,yc,xr-wl/2,yc) ;

/* PLACE THE OP-AMP BIAS LINE */
```

```
yc = level[2] + opamp.vos[1]*U ;
wl = opamp.wid[1]*U ;
WIRE(fdto,M1W,wl,xl+4*U+wl/2,yc,xr-4*U-wl/2,yc) ;

/* PLACE THE OP-AMP VDD LINE */
yc = level[2] + opamp.vos[2]*U ;
wl = opamp.wid[2]*U ;
WIRE(fdto,M1W,wl,xl+4*U+wl/2,yc,xr-4*U-wl/2,yc) ;

/* PLACE THE SWITCH P-WELL */
xc = (xl + xr) / 2 ;
xedge = xr - xl - 8*U ;
yc = level[1] - 55*U/2 ;
BOX(fdto,PWB,xedge,31*U,xc,yc) ;

/* PLACE THE N+ */
yc = level[1] - 16*U ;
BOX(fdto,NPB,xedge,8*U,xc,yc) ;

/* PLACE THE P+ */
yc =  level[1] ;
BOX(fdto,PPB,xedge,8*U,xc,yc) ;

/* PLACE THE P-WELL FOR THE CAPS */
xedge = xr - xl + 6*U ;
yedge = level[0] - level[1] + (EDGE/2 + 1)*U ;
yc = (level[0] + (17 + EDGE/2)*U + level[1]) / 2 ;
BOX(fdto,PWB,xedge,yedge,xc+U,yc) ;

/* INITIALIZE THE VERTICAL LEVELS FOR THE SUB CONTACTS */
yc = level[3] + 6*U ;
yc1 = level[0] + (5 + EDGE/2)*U ;
yc2 = level[2] + opamp.vos[0]*U ;

/* PLACE GROUND WIRES AND SUB CONTACTS FOR CAP P-WELL */
WIRE(fdto,M1W,2*w,xl+2*U,yc,xl+2*U,yc1) ;
WIRE(fdto,M1W,2*w,xr,yc2,xr,level[1] - 39*U) ;
WIRE(fdto,M1W,2*w,xr,level[1]-33*U,xr,yc1) ;
for (i=0; i*40*U<=yc1-yc; i++)
   { mps(xl+2*U,yc+i*40*U,fdto) ;
     mps(xr,yc+i*40*U,fdto) ;     }

WIRE(fdto,M1W,2*w,xl+2*U,yc1,xr,yc1) ;
for (i=0; xl+(15+i*40)*U<=xr-15*U; i++)
   mps(xl+(15+i*40)*U,yc1,fdto) ;

/* PLACE THE SUB CONTACTS BELOW THE CAPS */
for (i=1; i<=total; i++)
   { if (capac[i].typ=='C')
       { xc = (capac[i].hor[0] + EDGE/2)*U ;
```

```
            mps(xc,level[1]+12*U,fdto) ;
            WIRE(fdto,M1W,w,xc,level[1]+12*U,xc,level[1]-33*U) ; }

        if (capac[i].typ=='S' && !capac[i].used)
          { yc = 10*capac[i].hor[2] ;
            if (yc!=level[1]+17*U)
              { xc = (capac[i].hor[0] + EDGE/2)*U ;
                mps(xc,level[1]+12*U,fdto) ;
                WIRE(fdto,M1W,w,xc,level[1]+12*U,xc,level[1]-33*U)

   /* PLACE THE SUB CONTACTS ALONG THE VSS LINE */
   xc =  xl + 8*U ;
   yc = level[1] - 39*U ;
   while (xc < xr - 8*U)
        { flag = 1 ;
          for (j=0; j<stages; j++)

             /* GET THE CROSSING POINTS */
             { h[0] = whip[2*j].cross*U ;
               h[1] = whip[2*j+1].cross*U ;
               h[2] = whip[2*j].gndcross*U ;

               /* LOOK FOR A CROSSING */
               for (i=0; i<3; i++)
                  if (abs(h[i] - xc) < 6*U)
                    flag = 0 ;          }

          /* IF NO CROSSING, PLACE A CONTACT */
          if (flag)
            { mps(xc,yc,fdto) ;
              xc = xc + 40*U ;      }

          /* ELSE INCREMENT THE LOCATION AND TRY AGAIN */
          else xc = xc + U ;           }
}

/*************************************************************/
/*                                                           */
/*  INPUT1 FUNCTION  --  Hard-Wire the Input   ( LEE )       */
/*                                                           */
/*     The input1 function connects the input switched       */
/*  capacitor to the filter.                                 */
/*                                                           */
/*************************************************************/

input1(fdto)

FILE *fdto ;                    /* OUTPUT FILE */

{  int xc[3] ;                  /* HORIZONTAL CONNECTIONS */
```

```
    int yc[8] ;                 /* VERTICAL CONNECTIONS */

    xc[0] = (capac[1].hor[0]+EDGE/2)*U ;    /* FIRST CAP */
    xc[1] = (capac[2].hor[0]+EDGE/2)*U ;    /* SECOND CAP */
    xc[2] = -5*U ;                          /* INPUT NODE */

    yc[1] = U ;                         /* CAP CONTACT */
    yc[2] = -13*U ;
    yc[3] = level[1] ;                  /* SWITCH TOP */
    yc[4] = -3*U ;                      /* FIRST WIRING LEVEL */
    yc[5] = -8*U ;                      /* SECOND WIRING LEVEL */
    yc[6] = level[1] - 33*U ;           /* GROUND WIRE */
    yc[7] = level[1] + 12*U ;

    /* PLACE THE INPUT SWITCH AND CONNECT IT TO THE FILTER */
    switchplace(xc[1],yc[3],capac[2].swtch,fdto) ;
    WIRE(fdto,M1W,w,xc[1]+8*U,yc[3],xc[1]+8*U,yc[4]) ;

    /* CONNECT THE SECOND CAP TO THE SWITCH */
    mp2(xc[1],yc[7],fdto) ;
    WIRE(fdto,M1W,w,xc[1],yc[3],xc[1],yc[7]) ;
    WIRE(fdto,P2W,w,xc[1],yc[1],xc[1],yc[7]) ;

    /* CONNECT THE INPUT NODE */
    mp(xc[2],yc[2],fdto) ;
    mp(xc[1]-8*U,yc[2],fdto) ;
    WIRE(fdto,P1W,w,xc[2],yc[2],xc[1]-8*U,yc[2]) ;
    WIRE(fdto,P1W,w,xc[0]-8*U,yc[2],xc[0]-8*U,yc[1]) ;
    WIRE(fdto,M1W,w,xc[1]-8*U,yc[2],xc[1]-8*U,yc[3]) ;

    /* GROUND THE BOTTOM PLATE OF THE SECOND CAP */
    mp(xc[0],yc[5],fdto) ;
    WIRE(fdto,P1W,w,xc[1]-8*U,yc[5],xc[0],yc[5]) ;
    WIRE(fdto,P1W,w,xc[1]-8*U,yc[1],xc[1]-8*U,yc[5]) ;
    WIRE(fdto,M1W,w,xc[0],yc[5],xc[0],yc[6]) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT2 FUNCTION  --  Hard-Wire the Input   ( MARTIN )    */
/*                                                           */
/*     The input2 function connects the input node to the    */
/*  first switched capacitor of the filter.                  */
/*                                                           */
/*************************************************************/

input2(fdto)

FILE *fdto ;                            /* OUTPUT FILE */
```

```
{  int xc1,xc2 ;                              /* HORIZONTAL CONNECTS */
   int yc1,yc2 ;                              /* VERTICAL CONNECTS */

   xc1 = -5*U ;                               /* INPUT CONNECTION */
   xc2 = (capac[1].hor[0]+EDGE/2+8)*U ;       /* FIRST CAP */
   yc1 = level[1] ;                           /* SWITCH LEVEL */
   yc2 = level[1] + 17*U ;                    /* WIRE LEVEL */

   /* CONNECT THE INPUT NODE TO THE SWITCH */
   mp(xc1,yc2,fdto) ;
   WIRE(fdto,M1W,w,xc2,yc1,xc2,yc2) ;
   WIRE(fdto,M1W,w,xc1,yc2,xc2,yc2) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT3 FUNCTION  --  Hard-Wire the Input   ( SEDRA )     */
/*                                                           */
/*************************************************************/

input3(fdto)

FILE *fdto ;                    /* OUTPUT FILE */

{  int xc[15] ;                 /* HORIZONTAL CONNECTS */
   int yc[15] ;                 /* VERTICAL CONNECTS */

   xc[0] = -5*U ;                                  /* INPUT NODE */
   xc[1] = (capac[1].hor[0]+EDGE/2)*U ;            /* FIRST CAP */
   xc[2] = (capac[1].hor[0]+EDGE+EDGE/2)*U ;       /* EXTENSION */
   xc[3] = (capac[2].hor[0]+EDGE/2-8)*U ;          /* SECOND CAP */
   xc[4] = (capac[2].hor[0]+EDGE/2+8)*U ;          /* SECOND CAP */
   xc[5] = (capac[3].hor[0]+EDGE/2+8)*U ;          /* THIRD CAP */
   xc[6] = (capac[4].hor[0]+EDGE/2-8)*U ;          /* FOURTH CAP */
   xc[10] = (opamp.con[1] - 1)*U ;                 /* +INPUT */
   xc[11] = (opamp.con[0] - 1)*U ;                 /* -INPUT */
   xc[12] = (opamp.con[2] - 1)*U ;                 /* OUTPUT */

   yc[0] = U ;                                  /* PLATE LEVEL */
   yc[1] = level[1] ;                           /* TOP OF SWITCH */
   yc[2] = level[1] - 16*U ;                    /* BOTTOM OF SWITCH */
   yc[3] = level[1] - 33*U ;                    /* GROUND WIRE */
   yc[5] = -3*U ;                               /* CAP CONNECT */
   yc[6] = -8*U ;                               /* FIRST WIRING LEVEL */
   yc[7] = -13*U ;                              /* SECOND WIRING LEVEL *
   yc[8] = level[1] + 17*U ;                       /* TOP WIRE LEVEL */
   yc[10] = level[2] + opamp.level*U ;             /* OP-AMP TOP */
   yc[11] = level[1] - 27*U ;                      /* GAP BELOW SWITCH *
   yc[12] = level[2] + (opamp.level + 4)*U ;    /* CONNECT LEVEL */
```

```
/* PLACE EXTRA SWITCH GROUP AND CONNECT IT */
switchplace(xc[2],yc[1],capac[2].swtch,fdto) ;
WIRE(fdto,M1W,w,xc[2]+8*U,yc[2],xc[2]+8*U,yc[3]) ;
WIRE(fdto,M1W,w,xc[2]-8*U,yc[5],xc[6],yc[5]) ;
WIRE(fdto,M1W,w,xc[2]-8*U,yc[5],xc[2]-8*U,yc[1]) ;

/* GROUND BOTTOM PLATE OF INPUT CAPACITOR */
mp(xc[1]+6*U,yc[5],fdto) ;
WIRE(fdto,P1W,w,xc[1]+6*U,yc[0],xc[1]+6*U,yc[5]) ;
WIRE(fdto,M1W,w,xc[1]+6*U,yc[5],xc[1]+6*U,yc[3]) ;

/* CONNECT SECOND CAP TO EXTRA SWITCH GROUP */
mp(xc[3],yc[7],fdto) ;
WIRE(fdto,P1W,w,xc[3],yc[0],xc[3],yc[7]) ;
WIRE(fdto,M1W,w,xc[2],yc[7],xc[3],yc[7]) ;
WIRE(fdto,M1W,w,xc[2],yc[7],xc[2],yc[1]) ;

/* CONNECT SWITCH GROUPS */
WIRE(fdto,M1W,w,xc[5],yc[8],xc[5],yc[1]) ;
WIRE(fdto,M1W,w,xc[4],yc[8],xc[4],yc[1]) ;
WIRE(fdto,M1W,w,xc[4],yc[8],xc[5],yc[8]) ;

/* CONNECT SECOND CAP TO ITS SWITCH GROUP */
mp2(xc[1],yc[5],fdto) ;
WIRE(fdto,P2W,w,xc[1],yc[5],xc[1],yc[0]) ;
WIRE(fdto,M1W,w,xc[1],yc[5],xc[1],yc[1]) ;

/* CONNECT INPUT NODE TO INPUT SWITCH */
mp(xc[0],yc[6],fdto) ;
WIRE(fdto,M1W,w,xc[0],yc[6],xc[1]-8*U,yc[6]) ;
WIRE(fdto,M1W,w,xc[1]-8*U,yc[6],xc[1]-8*U,yc[1]) ;

/* PLACE METAL-ACTIVE CONTACTS */
ma(xc[1]-8*U,yc[1],fdto) ;
ma(xc[1],yc[1],fdto) ;
ma(xc[1]-8*U,yc[1]-16*U,fdto) ;
ma(xc[1],yc[1]-16*U,fdto) ;

/* PLACE METAL AND ACTIVE AREAS */
BOX(fdto,M1B,2*U,12*U,xc[1]-8*U,yc[1]-8*U) ;
BOX(fdto,M1B,2*U,12*U,xc[1],yc[1]-8*U) ;
WIRE(fdto,ACW,w,xc[1],yc[1],xc[1]-8*U,yc[1]) ;
WIRE(fdto,ACW,w,xc[1],yc[1]-16*U,xc[1]-8*U,yc[1]-16*U) ;

/* DECIDE ON THE CLOCKING SCHEME */
if (capac[1].swtch==1)
     { yc[13] = level[1] + U ; yc[14] = level[1] - 17*U ; }
else { yc[13] = level[1] - U ; yc[14] = level[1] - 15*U ; }

/* INSERT CLOCKS */
```

```
   BOX(fdto,P1B,2*U,8*U,xc[1]-4*U,yc[13]) ;
   BOX(fdto,P1B,2*U,8*U,xc[1]-4*U,yc[14]) ;

   /* CONNECT +INPUT TO INPUT SWITCH */
   mp(xc[1],yc[11],fdto) ;
   WIRE(fdto,M1W,w,xc[1],yc[11],xc[1],yc[2]) ;
   WIRE(fdto,P1W,w,xc[1],yc[11],xc[10],yc[11]) ;
   WIRE(fdto,P1W,w,xc[10],yc[11],xc[10],yc[10]) ;

   /* CONNECT OP-AMP OUTPUT FROM SWITCH */
   mp(xc[4],yc[11],fdto) ;
   WIRE(fdto,M1W,w,xc[4],yc[11],xc[4],yc[2]) ;
   WIRE(fdto,P1W,w,xc[4],yc[12],xc[4],yc[11]) ;

   /* CONNECT OUTPUT TO -INPUT */
   mp(xc[11],yc[12],fdto) ;
   mp(xc[12],yc[12],fdto) ;
   mp(xc[4],yc[12],fdto) ;
   WIRE(fdto,P1W,w,xc[11],yc[10],xc[11],yc[12]) ;
   WIRE(fdto,P1W,w,xc[12],yc[10],xc[12],yc[12]) ;
   WIRE(fdto,M1W,w,xc[11],yc[12],xc[12],yc[12]) ;
}

/*************************************************************/
/*                                                           */
/*  INPUT4 FUNCTION  --  Hard-Wire the Input   ( AROMA )     */
/*                                                           */
/*     The input2 function connects the input node to the    */
/*  first switched capacitor of the filter, and to a         */
/*  second one with complementary clocks if it exists. A     */

/*  connected to the first one if it exists.                 */
/*                                                           */
/*************************************************************/

input4(fdto)

FILE *fdto ;                                  /* OUTPUT FILE */

{  int xc[5],yc[5] ;                          /* CONNECTS */
   int i ;                                    /* INDEX VARIABLE */
   int scone,sctwo ;                          /* CAP LOCATIONS */
   int least[2],most[2] ;                     /* WIRE BOUNDS */

   xc[1] = -5*U ;                             /* INPUT CONNECTION */
   xc[2] = (capac[1].hor[0]+EDGE/2)*U ;       /* FIRST CAP */

   yc[1] = -3*U ;                            /* FIRST WIRING LEVEL */
   yc[2] = level[1] ;                        /* TOP OF SWITCH */
   yc[3] = level[1] + 12*U ;                 /* CONTACT LEVEL */
```

```
   yc[4] = level[l] + 17*U ;              /* TOP WIRE LEVEL */

   scone = sctwo = 0 ;
   for (i=2; i<=total; i++)
      if (same(&capac[i].sec[0],&table[0][0][0]) &&
            capac[i].typ=='S')
        { if (capac[i].swtch!=capac[l].swtch)
            scone = i ;
          else sctwo = i ;  }

   xc[3] = (capac[scone].hor[0]+EDGE/2)*U ;
   xc[4] = (capac[sctwo].hor[0]+EDGE/2)*U ;

   if (scone!=0 && sctwo!=0)
     { mp(xc[l],yc[4],fdto) ;
       WIRE(fdto,MlW,w,xc[l],yc[4],xc[3]+8*U,yc[4]) ;
       WIRE(fdto,MlW,w,xc[3]+8*U,yc[2],xc[3]+8*U,yc[4]) ;
       WIRE(fdto,MlW,w,xc[2]+8*U,yc[2],xc[2]+8*U,yc[4]) ;

       least[0] = 4 ;
       most[0] = 3*sctwo + l ;
       yc[0] = (-5*wireslot(least,most)-3)*U ;
       WIRE(fdto,MlW,w,xc[4],yc[0],xc[2],yc[0]) ;
       WIRE(fdto,P2W,w,xc[4],yc[0],xc[4],U) ;
       mp2(xc[4],yc[0],fdto) ;
       mp2(xc[2],yc[0],fdto) ;     }

   else { WIRE(fdto,MlW,w,xc[l],yc[4],xc[2]+8*U,yc[4]) ;
       WIRE(fdto,P2W,w,xc[2]+8*U,yc[2],xc[2]+8*U,yc[4]) ;
       mp(xc[l],yc[4],fdto) ;     }
}

/*************************************************************/
/*                                                           */
/*  ATTACHl FUNCTION  --  Attach Capacitor Plate             */
/*                                                           */
/*     The attachl function places a plate with a base       */
/*  width of EDGE. The height is smaller if a fraction of    */
/*  less than one is given.                                  */
/*                                                           */
/*************************************************************/

attachl(a,b,xdir,ydir,fraction,fdto,flag)

int a,b ;                  /* BASE CO-ORDS */
int xdir,ydir ;            /* PLATE DIRECTION */
int flag ;                 /* LINKING BOX FLAG */
float fraction ;           /* PLATE SIZE */
FILE *fdto ;               /* OUTPUT FILE */
```

```
{   int xc,yc ;              /* CENTER OF PLATE */
    int xp,yp ;              /* CENTER OF LINKING BOX */
    int temp ;               /* STORAGE VARIABLE */
    int xedl,yedl ;          /* PLATE EDGES */
    float tret = EDGE ;

    temp = 2 + tret + fraction*(tret-2) ;

    /* IF THE DIRECTION IS LEFT OR RIGHT... */
    if (ydir == 0)
      { xc   = (2*a + EDGE + xdir*temp)*U/2 ;
        yc   = (2*b-1)*EDGE*U/2 + U ;
        yedl = (EDGE-2)*U ;
        xedl = fraction*(EDGE-2)*U ;
        xp   = (2*a + EDGE*(1+xdir))*U/2 ;
        yp   = yc ;    }

    /* IF THE DIRECTION IS UP OR DOWN... */
    else { yc = (2*EDGE*b - EDGE + ydir*temp)*U/2 + U ;
        xc   = (2*a+EDGE)*U/2 ;
        xedl = (EDGE-2)*U ;
        yedl = fraction*(EDGE-2)*U ;
        yp   = (2*b-1+ydir)*EDGE*U/2 + U ;
        xp   = xc ;    }

    /* SAVE THE HIGHEST PLATE LEVEL */
    if (yc > level[0])
      level[0] = yc ;

    /* PLACE THE PLATE */
    BOX(fdto,P2B,xedl,yedl,xc,yc) ;
    BOX(fdto,P1B,xedl+2*U,yedl+2*U,xc,yc) ;

    /* PLACE A LINK IF NECCESSARY */
    if (flag)
      BOX(fdto,P2B,2*U,2*U,xp,yp) ;

    /* PLACE THE TIP IF POSSIBLE */
    if (fraction<1.0 && ydir==1)
      BOX(fdto,P2B,2*U,2*U,xc,yc+U+yedl/2) ;
}

/*************************************************************/
/*                                                           */
/*  ATTACH2 FUNCTION  --  Attach Capacitor Plate             */
/*                                                           */
/*     The attach2 function places a plate if its size is    */
/*  less than LIMIT. A rectangle is placed on top of the     */
/*  previous plate to maintain the perimeter to area         */
/*  ratio.                                                   */
```

```
/*                                                                    */
/**********************************************************************/

attach2(a,b,xdir,ydir,fraction,fdto,flag)

int a,b ;                          /* STARTING CO-ORDS */
int xdir,ydir ;                    /* PLATE DIRECTION */
int flag ;                         /* !! NOT USED  !! */
float fraction ;                   /* PLATE SIZE */
FILE *fdto ;                       /* OUTPUT FILE */

{  int xc,yc ;                     /* CENTER OF PLATE */
   int xedl,yedl ;                 /* PLATE EDGES */
   float aa ;                      /* RECTANGLE HEIGHT */
   float bb ;                      /* RECTANGLE WIDTH */
   float tret = EDGE ;

   /* FIND THE RECTANGLE SIDES */
   aa = 2.0*U*fraction*tret ;
   bb = tret*U/2.0 + 2.0*U/tret ;

   /* IF THE PLATE DIRECTION IS LEFT OR RIGHT... */
   if (ydir==0)
     { xedl = aa ;
       yedl = bb ;
       xc = (2*a + EDGE + xdir*(EDGE-2))*U/2 + xdir*aa/2 ;
       yc = (2*b-1)*EDGE*U/2 + U ;  }

   /* IF THE PLATE DIRECTION IS UP OR DOWN... */
   else { xedl = bb ;
       yedl = aa ;
       yc = ((2*b-1)*EDGE + ydir*(EDGE-2))*U/2 + ydir*aa/2 + U ;
       xc = (2*a+EDGE)*U/2 ;  }

   /* SAVE THE HIGHEST PLATE CO-ORD */
   if (yc > level[0])
     level[0] = yc ;

   /* PLACE THE RECTANGLE */
   BOX(fdto,P2B,xedl,yedl,xc,yc) ;
   BOX(fdto,P1B,xedl+2*U,yedl+2*U,xc,yc) ;

   /* ADD THE TIP IF POSSIBLE */
   if (ydir==1)
     BOX(fdto,P2B,2*U,2*U,xc,yc+U+yedl/2) ;
}

/**********************************************************************/
/*                                                                    */
/*  ATTACH3 FUNCTION  --  Attach Capacitor Plate                      */
```

```
/*                                                              */
/*     The attach3 function places a plate that is less         */
/*  than the full size buf larger than LIMIT. It spreads        */
/*  the link width to maintain the perimeter/area ratio.        */
/*                                                              */
/****************************************************************/

attach3(a,b,xdir,ydir,fraction,fdto,flag)

int a,b ;                       /* STARTING CO-ORDS */
int xdir,ydir ;                 /* PLATE DIRECTION */
int flag ;                      /* !! NOT USED  !! */
float fraction ;                /* PLATE SIZE */
FILE *fdto ;                    /* OUTPUT FILE */

{  int xc,yc ;                  /* PLATE CENTER */
   int xp,yp ;                  /* LINK CENTER */
   int xl,yl ;                  /* LINK SIDES */
   int xedl,yedl ;              /* PLATE SIDES */
   float aa,bb ;                /* PLATE VARIABLES */
   float tret = EDGE ;

   /* CALCULATE THE PLATE SIZE */
   aa = fraction*(tret + 2.0) - 2.0 ;
   bb = tret - fraction*(tret - 2.0) ;

   /* IF THE PLATE DIRECTION IS LEFT OR RIGHT... */
   if (ydir==0)
     { xc = (2*a + EDGE + xdir*(2+EDGE+aa))*U/2 ;
       yc = (2*b-1)*EDGE*U/2 + U ;
       yedl = (EDGE-2)*U ;
       xedl = aa*U ;

       xp = (2*a + EDGE*(1+xdir))*U/2 ;
       yp = yc ;
       yl = bb*U ;
       xl = 2*U ;          }

   /* IF THE PLATE DIRECTION IS UP OR DOWN */
   else { yc = ((2*b-1)*EDGE + ydir*(2+EDGE+aa))*U/2 + U ;
       xc   = (2*a+EDGE)*U/2 ;
       xedl = (EDGE-2)*U ;
       yedl = aa*U ;

       yp = (2*b-1+ydir)*EDGE*U/2 + U ;
       xp = xc ;
       xl = bb*U ;
       yl = 2*U ;      }

   /* SAVE THE HIGHEST PLATE LEVEL */
```

```
   if (yc > level[0])
     level[0] = yc ;

   /* PLACE THE PLATE AND LINK */
   BOX(fdto,P2B,xed1,yed1,xc,yc) ;
   BOX(fdto,P2B,xl,yl,xp,yp) ;
   BOX(fdto,P1B,xed1+2*U,yed1+2*U,xc,yc) ;

   /* PLACE THE TIP IF NECCESSARY */
   if (ydir==1)
     BOX(fdto,P2B,2*U,2*U,xc,yc+U+yed1/2) ;
}

/*************************************************************/
/*                                                           */
/*  BOX FUNCTION  --  Write a Box to the CIF File            */
/*                                                           */
/*     The box function writes a box command to the output   */
/*  file. The center and edges of the box are used.          */
/*                                                           */
/*************************************************************/

BOX(fdto,st,xed,yed,xc,yc)

FILE *fdto ;                          /* OUTPUT FILE */
char *st ;                            /* BOX LAYER */
int xed,yed ;                         /* BOX EDGES */
int xc,yc ;                           /* BOX CENTER */

{  char *BOXCOM = "%s %d %d %d %d %s" ;     /* BOX FORMAT */
   char buf[LENGTH] ;                       /* LINE BUFFER */

   /* WRITE THE BOX */
   sprintf(buf,BOXCOM,st,xed,yed,XBASE+xc,YBASE+yc,";\n") ;
   writeline(fdto,buf) ;
}

/*************************************************************/
/*                                                           */
/*  CELLPLACE FUNCTION  --  Place a Cell                     */
/*                                                           */
/*     The cellplace function places a cell at the given     */
/*  location.                                                */
/*                                                           */
/*************************************************************/

cellplace(fdto,st,xc,yc)

FILE *fdto ;                        /* OUTPUT FILE */
char *st ;                          /* CELL CALL */
```

```
int xc,yc ;                          /* CELL POSITION */

{  int x,y ;                         /* ACTUAL CELL PLACEMENT */
   char buf[LENGTH] ;                /* LINE BUFFER */
   char *CELL = "%s %d %d %s" ;      /* FORMAT */

   /* COMPUTE ACTUAL PLACEMENT */
   x = XBASE + xc ;
   y = YBASE + yc - opamp.yorg*U ;

   /* PLACE THE CELL */
   sprintf(buf,CELL,st,x,y,";\n") ;
   writeline(fdto,buf) ;
}

/*************************************************************/
/*                                                           */
/*  CLOCKPHASE FUNCTION  --  Check Phase of a Clock          */
/*                                                           */
/*     The clockphase function checks the phase of the       */
/*  specified clock. This is only valid for SC and BS        */
/*  elements and the clocks must be C or #C.                 */
/*                                                           */
/*************************************************************/

clockphase(buf,spot)

char buf[LENGTH] ;                   /* LINE BUFFER */
int spot ;                           /* CLOCK POSITION */

{  int i,j ;                         /* INDEX VARAIBLES */

   /* MOVE TO THE BEGINNING OF THE FIRST CLOCK */
   j = 0 ;
   while (buf[j++]!=')') ;
   while (buf[j]==' ') j++ ;

   /* ADVANCE TO THE DESIRED CLOCK */
   for (i=1; i<=spot; i++)
      { while (buf[j]!=' ') j++ ;
        while (buf[j]==' ') j++ ; }

   /* RETURN 0 IF #C IS FOUND */
   return((buf[j]=='#') ? 0 : 1) ;
}

/*************************************************************/
/*                                                           */
/*  COPY FUNCTION  --  String Copy                           */
/*                                                           */
```

```
/*     The copy function copies string1 to string2. The    */
/*  first string must be terminated by an EOW.             */
/*                                                         */
/***********************************************************/

copy(str1,str2)

char *str1 ;                          /* INPUT STRING */
char *str2 ;                          /* OUTPUT STRING */

{  int i = 0 ;                        /* INDEX VARIABLE */

   /* COPY STRING1 TO STRING2 */
   do { *(str2+i) = *(str1+i) ; }
   while (*(str1+i++)!=EOW && *(str1+i-1)!=EOL) ;

   /* ALWAYS TERMINATE WITH EOW */
   if (*(str1+i-1)==EOL)
     *(str2+i-1)=EOW ;
}


/***********************************************************/
/*                                                         */
/*  DOPE FUNCTION  --  Check For Array Space               */
/*                                                         */
/*     The dope function decides if there is enough space  */
/*  to insert a plate into the grid.                       */
/*                                                         */
/***********************************************************/

dope(x,xdir,y,ydir,count,max)

int x,y ;                             /* STARTING CO-ORDS */
int xdir,ydir ;                       /* PLATE DIRECTION */
int count ;                           /* CURRENT CAPACITOR */
int max ;                             /* RIGHT EDGE OF THE GRID */

{  int x2,y2 ;                        /* PLATE CENTER */
   int i,j ;                          /* INDEX VARIABLES */
   int right ;                        /* RIGHT BOUNDARY OF CHECK */
   int left ;                         /* LEFT BOUNDARY OF CHECK */
   int a,b ;                          /* ARRAY CHECK CO-ORDS */

   /* FIND THE BOTTOM CORNER OF THE PLATE */
   x2 = x + xdir * EDGE ;
   y2 = y + ydir ;

   /* FIND THE LIMITS OF THE PLATE */
   right =  x2 + EDGE + CAPGAP ;
```

```
   left = x2 - CAPGAP ;

   /* SEARCH FOR ROOM FOR THE PLATE */
   for (i=x2; i<x2+EDGE; i++)
      if (grid[i][y2]!=0)
        return(0) ;

   /* SEARCH FOR ENOUGH GAP AROUND THE PLATE */
   for (j=y2-1; j<=y2+1; j++)
      for (i=left; i<right; i++)

            /* ONLY CHECK WITHIN THE GRID */
            { a = i ; b = j ;
              if (i<1) a = 1 ;
              if (i>max-1) a = max-1 ;
              if (j<1) b = 1 ;
              if (j>TOPMAX) b = TOPMAX ;
              if (grid[a][b]!=0 && grid[a][b]!=count)
                return(0) ;              }

   return(1) ;
}

/*************************************************************/
/*                                                           */
/*  EMPTY FUNCTION  --  Clears a Column of TABLE             */
/*                                                           */
/*     The empty function initializes one column of TABLE    */
/*  to EOW.                                                  */
/*                                                           */
/*************************************************************/

empty(col)

int col ;                  /* TABLE COLUMN */

{  int b ;                 /* INDEX VARIABLE */

   /* INITIALIZE THE COLUMN */
   for (b=0; b<50; b++)
       table[col][b][0] = EOW ;
}

/*************************************************************/
/*                                                           */
/*  GETNODE FUNCTION  --  Get One Node of a Line             */
/*                                                           */
/*     The getnode function returns the requested node       */
/*  from the given line.                                     */
/*                                                           */
```

```
/*************************************************************/

getnode(buf,pos,vertx)

char buf[LENGTH] ;                  /* LINE BUFFER */
int pos ;                           /* NODE POSITION */
char vertx[WORD] ;                  /* NODE BUFFER */

{  int q,i,k ;                      /* INDEX VARIABLES */

   q = i = 0 ;
   while (buf[i++] != '(')  ;

   /* STORE THE REQUESTED NODE */
   for (k=1; k<=pos; k++)
       {  while (buf[i] == ' ')
              i++ ;

          while (buf[i] != ' ' && buf[i] != ')')
              {  if (k == pos)
                     vertx[q++] = buf[i++] ;
                 else i++ ;                          }     }

   vertx[q] = EOW ;
}

/*************************************************************/
/*                                                           */
/*  MA FUNCTION  --  Metal-Active Contact                    */
/*                                                           */
/*     The ma function places a metal-active contact at      */
/*  the given location.                                      */
/*                                                           */
/*************************************************************/

ma(xc,yc,fdto)

int xc,yc ;                         /* LOCATION */
FILE *fdto ;                        /* OUTPUT FILE */

{  /* PLACE THE CONTACT */
   BOX(fdto,M1B,4*U,4*U,xc,yc) ;
   BOX(fdto,C1B,2*U,2*U,xc,yc) ;
   BOX(fdto,ACB,4*U,4*U,xc,yc) ;
}

/*************************************************************/
/*                                                           */
/*  MP FUNCTION  --  Metal-Poly1 Contact                     */
/*                                                           */
```

```
/*     The mp function places a metal-poly1 contact at the */
/*  given location.                                        */
/*                                                         */
/*********************************************************/

mp(xc,yc,fdto)

int xc,yc ;                    /* LOCATION */
FILE *fdto ;                   /* OUTPUT FILE */

{  /* PLACE THE CONTACT */
   BOX(fdto,M1B,4*U,4*U,xc,yc) ;
   BOX(fdto,C1B,2*U,2*U,xc,yc) ;
   BOX(fdto,P1B,4*U,4*U,xc,yc) ;
}

/*********************************************************/
/*                                                       */
/*  MP2 FUNCTION  --  Metal-Poly2 Contact                */
/*                                                       */
/*     The mp2 function places a metal-poly2 contact at  */
/*  the given location.                                  */
/*                                                       */
/*********************************************************/

mp2(xc,yc,fdto)

int xc,yc ;                    /* LOCATION */
FILE *fdto ;                   /* OUTPUT FILE */

{  /* PLACE THE CONTACT */
   BOX(fdto,M1B,4*U,4*U,xc,yc) ;
   BOX(fdto,C1B,2*U,2*U,xc,yc) ;
   BOX(fdto,P2B,5*U,5*U,xc,yc) ;
}

/*********************************************************/
/*                                                       */
/*  MPS FUNCTION  --  Substrate Contact to P-well        */
/*                                                       */
/*     The mps function places a p-well substrate contact */
/*  at the given location.                               */
/*                                                       */
/*********************************************************/

mps(xc,yc,fdto)

int xc,yc ;                    /* LOCATION */
FILE *fdto ;                   /* OUTPUT FILE */
```

```
{  /* PLACE THE CONTACT */
   BOX(fdto,M1B,4*U,4*U,xc,yc) ;
   BOX(fdto,C1B,2*U,2*U,xc,yc) ;
   BOX(fdto,ACB,4*U,4*U,xc,yc) ;
   BOX(fdto,PPB,8*U,8*U,xc,yc) ;
}

/*************************************************************/
/*                                                           */
/*  READCOM FUNCTION  --  Read Input Data                    */
/*                                                           */
/*     The readcom function reads the input parameters       */
/*  from the .com file.                                      */
/*                                                           */
/*************************************************************/

readcom(fdcom)

FILE *fdcom ;                    /* .COM FILE */

{  /* INITIALIZE THE NODE TABLE */
   empty(0) ; empty(1) ; empty(2) ;

   /* READ THE INPUT PARAMETERS */
   fscanf(fdcom,"%f %d %d",&LAMBDA,&topper,&swap) ;
   fscanf(fdcom,"%d %d",&opamp.xorg,&opamp.yorg) ;
   fscanf(fdcom,"%d %d %d",&opamp.width,&opamp.height,&opamp.level)
   fscanf(fdcom,"%d %d %d",&opamp.con[0],&opamp.con[1],&opamp.con[2
   fscanf(fdcom,"%d %d %d",&opamp.vos[0],&opamp.vos[1],&opamp.vos[2
   fscanf(fdcom,"%d %d %d",&opamp.wid[0],&opamp.wid[1],&opamp.wid[2

   /* INITIALIZE THE CIF UNITS */
   U = 100*LAMBDA ;
   W = 200*LAMBDA ;
}

/*************************************************************/
/*                                                           */
/*  READND FUNCTION  --  Read Nodal Data                     */
/*                                                           */
/*     The readnd function reads the input parameters from   */
/*  the .nd file.                                            */
/*                                                           */
/*************************************************************/

readnd(fdnd,head)

FILE *fdnd ;                          /* .ND FILE */
char head ;                           /* INPUT SECTION STYLE */
```

```
{  char line[LENGTH] ;                  /* LINE BUFFER */
   int i ;                              /* INDEX VARIABLE */

   /* INITIALIZE THE NODE TABLE */
   readline(fdnd,line) ;
   copy(&line[0],&table[0][0][0]) ;
   readline(fdnd,line) ;
   copy(&line[0],&table[0][1][0]) ;

   readline(fdnd,line) ;
   readline(fdnd,line) ;

   /* READ THE INPUT SECTION NODES */
   if (head=='L' || head=='M')
     { copy(&line[0],&table[1][0][0]) ;
       start = 0 ;   }

   if (head=='S')
     { copy(&line[0],&table[1][0][0]) ;
       readline(fdnd,line) ;
       copy(&line[0],&table[1][1][0]) ;

       copy(&table[1][0][0],&table[2][0][0]) ;
       copy(&table[1][1][0],&table[2][1][0]) ;
       start = 2 ;            }

   /* READ THE INTEGRATOR NODES */
   readline(fdnd,line) ;
   readline(fdnd,line) ;
   for(i=start; line[0]!=EOF; i++)
     { copy(&line[0],&table[2][i][0]) ;
       readline(fdnd,line) ;    }
}

/*************************************************************/
/*                                                           */
/*  READLINE FUNCTION  --  Read a Line From File             */
/*                                                           */
/*     The readline function reads a line terminated by an   */
/*  EOL or an EOF from the given input file.                 */
/*                                                           */
/*************************************************************/

readline(fdfrom,buf)

FILE *fdfrom ;          /* INPUT FILE */
char buf[LENGTH] ;      /* LINE BUFFER */

{  int q = 0 ;          /* INDEX VARIABLE */
```

```
   /* READ CHARACTERS FROM THE INPUT FILE */
   do  {  buf[q++] = getc(fdfrom) ;  }
   while (buf[q-1] != EOL && buf[q-1] != EOF) ;
}

/*************************************************************/
/*                                                           */
/*  SAME FUNCTION  --  Checks String Equality                */
/*                                                           */
/*     The same function compares two strings up to the      */
/*  last EOW.                                                */
/*                                                           */
/*************************************************************/

same(buf1,buf2)

char *buf1 ;                    /* FIRST STRING */
char *buf2 ;                    /* SECOND STRING */

{  int k = 0 ;                  /* INDEX VARAIBLE */

   /* COMPARE THE STRINGS CHARACTER BY CHARACTER */
   do { if (buf1[k]!=buf2[k++])
          return(0) ;     }
   while (buf2[k-1]!=EOW && buf1[k-1]!=EOW) ;
   return(1) ;
}

/*************************************************************/
/*                                                           */
/*  SHOVEON FUNCTION  --  Copy Op-Amp CIF File               */
/*                                                           */
/*     The shoveon function copies the op-amp CIF file       */
/*  into the layout file.                                    */
/*                                                           */
/*************************************************************/

shoveon(fdto)

FILE *fdto ;                    /* OUTPUT FILE */

{  int ch ;                     /* CHARACTER BUFFER */
   FILE *fdfrom ;               /* INPUT FILE */
   FILE *fopen() ;              /* FILE OPEN FUNCTION */

   /* OPEN THE INPUT FILE */
   fdfrom = fopen("opamp.cif","r") ;

   /* COPY FILE */
   while ((ch = getc(fdfrom)) != EOF)
```

```
        putc(ch,fdto) ;

   /* CLOSE INPUT FILE */
   fclose(fdfrom) ;
}

/*************************************************************/
/*                                                           */
/*  SWITCHPLACE FUNCTION  --  Place One Switch Group         */
/*                                                           */
/*     The switchplace function places the switch group      */
/*  with the appropriate clocking scheme.                    */
/*                                                           */
/*************************************************************/

switchplace(x,y,index,fdto)

int x,y ;                          /* SWITCH CO-ORDS */
int index ;                        /* CLOCKING SCHEME */
FILE *fdto ;                       /* OUTPUT FILE */

{  int i,j ;                       /* INDEX VARIABLES */
   int xc,yc ;                     /* SWITCH PLACEMENT POINTS */
   int yc1,yc2,yc3,yc4 ;           /* WIRING LEVELS */

   /* PLACE THE METAL-ACTIVE CONTACTS */
   for (i=0-1; i<2; i++)
      for (j=0; j<2; j++)
         ma(x+i*8*U,y-j*16*U,fdto) ;

   /* DRAW THE METAL LINES */
   for (i=0-1; i<2; i++)
      { xc = x + i*8*U ;
        yc = y - 8*U ;
        BOX(fdto,M1B,2*U,12*U,xc,yc) ;   }

   /* PUT IN THE ACTIVE LINES */
   BOX(fdto,ACB,12*U,2*U,x,y) ;
   BOX(fdto,ACB,12*U,2*U,x,y-16*U) ;

   /* CLOCKING SCHEME */
   if (capac[index].swtch==1)
     { yc1 = level[1] - U ; yc2 = level[1] - 15*U ;
       yc3 = level[1] + U ; yc4 = level[1] - 17*U ;  }

   else { yc1 = level[1] + U ; yc2 = level[1] - 17*U ;
          yc3 = level[1] - U ; yc4 = level[1] - 15*U ;  }

   /* INSERT THE P1 TIPS ON THE CLOCK LINES */
   BOX(fdto,P1B,2*U,8*U,x+4*U,yc1) ;
```

```
   BOX(fdto,P1B,2*U,8*U,x+4*U,yc2) ;
   BOX(fdto,P1B,2*U,8*U,x-4*U,yc3) ;
   BOX(fdto,P1B,2*U,8*U,x-4*U,yc4) ;
}

/*************************************************************/
/*                                                           */
/*  TURN FUNCTION  --  Checks for Op-Amp Mirroring           */
/*                                                           */
/*     The turn function decides if an opamp should be       */
/*  imaged. It checks for an intersection of two lines       */
/*  connecting the opamp nodes to their switch               */
/*  connections.                                             */
/*                                                           */
/*************************************************************/

turn(i)

int i ;                              /* OP-AMP INDEX */

{  int delx,dela ;                   /* HORIZONTAL BOUNDS */
   int xin ;                         /* INTERCEPT CO-ORD */
   int xmin,xmin1,xmin2 ;            /* LINE MINIMA */
   int xmax,xmax1,xmax2 ;            /* LINE MAXIMA */

   /* FIND HORIZONTAL LENGTHS OF CONNECTIONS */
   delx = whip[i].cap - whip[i].out ;
   dela = whip[i+1].cap - whip[i+1].out ;

   /* FAULTY CROSSING CHECK */
   if (whip[i+1].cap>whip[i+1].out && whip[i].out>whip[i].cap)
     return(0) ;

   /* FAULTY CROSSING CHECK */
   if (whip[i+1].out>whip[i+1].cap && whip[i+1].cap>whip[i].cap)
     if (whip[i].cap>whip[i].out)
       return(0) ;

   /* FIND MAX AND MIN OF FIRST LINE */
   if (delx > 0)
     { xmax1 = whip[i].cap ; xmin1 = whip[i].out ; }
   else { xmax1 = whip[i].out ; xmin1 = whip[i].cap ; }

   /* FIND MAX AND MIN OF SECOND LINE */
   if (dela > 0)
     { xmax2 = whip[i+1].cap ; xmin2 = whip[i+1].out ; }
   else { xmax2 = whip[i+1].out ; xmin2 = whip[i+1].cap ; }

   /* FIND OVERALL MIN */
   if (xmin1 < xmin2)
```

```
      xmin = xmin1 ;
    else xmin = xmin2 ;

    /* FIND OVERALL MAX */
    if (xmax1 > xmax2)
      xmax = xmax1 ;
    else xmax = xmax2 ;

    /* COMPUTE INTERSECTION POINT */
    if (delx!=dela)
      { xin = (whip[i].cap*dela - whip[i+1].cap*delx)/(dela-delx) ;
        if (xin > xmin && xin < xmax)
          return(1) ;   }

    return(0) ;
}

/*************************************************************/
/*                                                           */
/*  WIRE FUNCTION  --  Write a Wire to the CIF File          */
/*                                                           */
/*     The wire function writes a wire command to the        */
/*  output file. The width and end-points of the wire are    */
/*  used.                                                    */
/*                                                           */
/*************************************************************/

WIRE(fdto,st,width,x1,y1,x2,y2)

FILE *fdto ;                          /* OUTPUT FILE */
char *st ;                            /* BOX LAYER */
int width ;                           /* WIRE WIDTH */
int x1,y1 ;                           /* WIRE START */
int x2,y2 ;                           /* WIRE FINISH */


{  char *WC = "%s %d %d %d %d %d %s" ;  /* WIRE FORMAT */
   char buf[LENGTH] ;                      /* LINE BUFFER */

   /* WRITE THE WIRE */
   sprintf(buf,WC,st,width,XBASE+x1,YBASE+y1,XBASE+x2,YBASE+y2,";\n
   writeline(fdto,buf) ;
}

/*************************************************************/
/*                                                           */
/*  WIRESLOT FUNCTION  --  Find Wire Level                   */
/*                                                           */
/*      The wireslot function finds the first available      */
/*  level for a wire to inserted into the grid. The          */
```

```
/*  wire end-points are used.                                          */
/*                                                                     */
/*********************************************************************/

wireslot(least,most)

int least[2] ;                  /* LEFT END-POINT */
int most[2] ;                   /* RIGHT END-POINT */

{  int i,k ;                    /* INDEX VARIABLES */
   int flag,f2 ;                /* SEARCH FLAGS */
   int j ;                      /* WIRE LEVEL */

   /* FIND A HOLE IN THE ARRAY LONG ENOUGH FOR THE WIRE */
   f2 = 1 ;
   for (j=0; f2; j++)
      { flag = 1 ;
        for (k=least[0]; k<=most[0]; k++)
           if (path[j][k]!=0)
             flag = 0 ;
        if (flag) f2 = 0 ;    }

   j-- ;

   /* FILL THE HOLE */
   for (k=least[0]; k<=most[0]; k++)
      path[j][k] = 1 ;

   /* UPDATE THE HIGHEST WIRING LEVEL */
   if (level[1] > (-5*j-20)*U)
     level[1] = (-5*j-20)*U ;

   return(j) ;
}

/*********************************************************************/
/*                                                                   */
/*  WRITELINE FUNCTION  --  Write a Line To File                     */
/*                                                                   */
/*     The writeline function writes a line terminated by            */
/*  an EOL or an EOF to the output file.                             */
/*                                                                   */
/*********************************************************************/

writeline(fdto,buf)

FILE *fdto ;                    /* OUTPUT FILE */
char buf[LENGTH] ;              /* LINE BUFFER */

{  int q = 0 ;                  /* INDEX VARIABLE */
```

```
    /* WRITE THE LINE TO THE OUTPUT FILE */
    do { putc(buf[q++],fdto) ; }
    while (buf[q-1] != EOL && buf[q-1] != EOF && buf[q-1] != EOW) ;
}
```