

12-5-1988

A CMOS Circuit Generator Using Differential Pass Transistors for Implementing Boolean Functions

Rabe'eh Mahooti
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

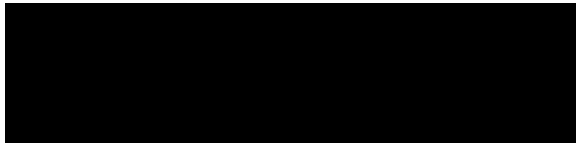
Mahooti, Rabe'eh, "A CMOS Circuit Generator Using Differential Pass Transistors for Implementing Boolean Functions" (1988). *Dissertations and Theses*. Paper 3805.
<https://doi.org/10.15760/etd.5689>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

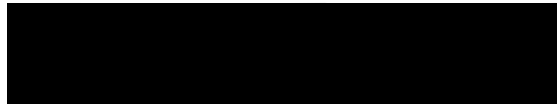
AN ABSTRACT OF THE THESIS OF RABE'EH MAHOOTI for the Master of Science in Electrical Engineering presented December 5, 1988.

Title: A CMOS Circuit Generator Using Differential Pass Transistors for Implementing Boolean Functions.

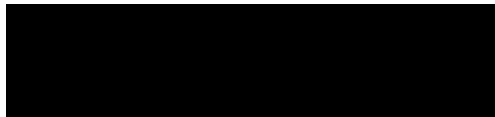
APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:



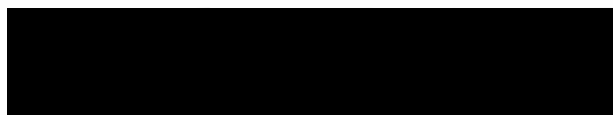
W. Robert Daasch. Chair



L. W. Casperson



R. P. Aggarwal



Bradford R. Crain

This study uses differential pass transistor methodology for implementing and evaluating Boolean functions. The main goal is investigation of CMOS and nMOS approaches in pass transistor logic design. Pass-transistor logic is most effective in the implementation of Boolean functions when the vectors are in the same format. It has been demonstrated that nMOS pass transistor logic driven by a control signal voltage above the V_{dd} level offers a significant improvement in speed. nMOS pass transistors

also offer less area consumption in comparison to the CMOS approach.

The philosophy developed here has been used in the design of a program for the layout generation of pass transistor networks. This program has been applied to the design of a 4-to-1 multiplexer and an adder (sum and carry). The layout of the circuit sub-cell have been done using the program Magic, based on 3μ CMOS p-well technology.

**A CMOS CIRCUIT GENERATOR USING DIFFERENTIAL PASS
TRANSISTORS FOR IMPLEMENTING BOOLEAN FUNCTIONS**

by

RABE'EH MAHOOTI

A thesis submitted in partial fulfillment of the
requirement for the degree of

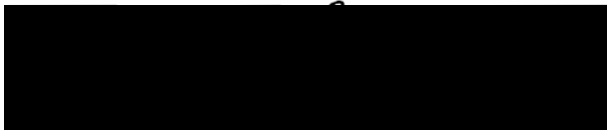
**MASTER OF SCIENCE
in
ELECTRICAL ENGINEERING**

Portland State University

1988

TO THE OFFICE OF GRADUATE STUDIES:

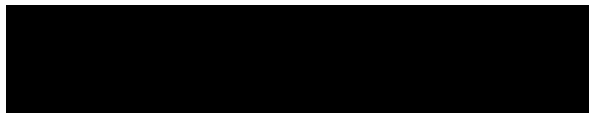
The member of the Committee approve the thesis of Rabe'Eh Mahooti presented December 5, 1988.



W. Robert Daasch, Chair



L. W. Casperson

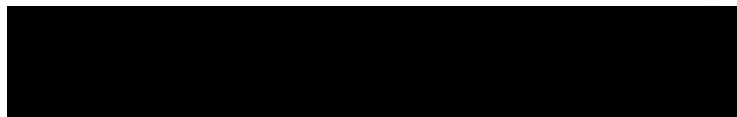


R. P. Aggarwal

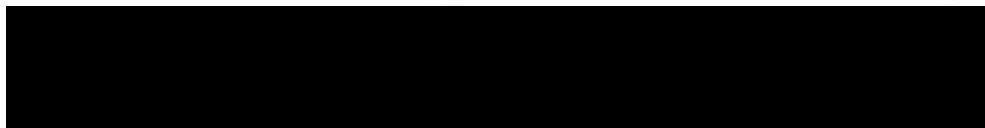


Bradford R. Crain

APPROVED:



L. W. Casperson, Acting Chair, Department of Electrical Engineering



Bernard Ross, Vice Provost for Graduate Studies

ACKNOWLEDGEMENT

I wish to give special thanks to my parents, Mahrokh and Hassan, who have been helping, supporting, and encouraging me. I would also like to thank my adviser, Dr. Robert Daasch, who provided guidance in the process of this research work. Special thanks must also be given to Dr. Lee Casperson and Grigory Kogan for their tremendous support.

I wish to thank graduate students, faculty, and staff of electrical engineering department, specially Janaka Jayawardena and Shirley Clark, for their help and the convenience in using utility equipment.

For providing me with emotional support especially during the last few days of my thesis writing I want to thank my brother, Hamed, and special friends, Ela and Mehdi.

TABLE OF CONTENTS

	PAGE
ACKNOWLEDGEMENT.....	iii
LIST OF TABLES.....	vi
LIST OF FIGURES.....	vii
CHAPTER	
I INTRODUCTION.....	1
II STUDY AND COMPARISON OF MOS PASS TRANSISTOR LOGIC.....	10
MOS Pass Transistors.....	11
nMOS Pass Transistor.....	11
pMOS Pass Transistor.....	13
CMOS Pass Transistor.....	14
Implementing the nMOS Differential Pass Transistor.....	15
III ANALYSIS OF MOS PASS TRANSISTOR LOGIC AND BOOTSTRAP CIRCUIT.....	18
Precharging the Fabric.....	18
Precharging Through an n-Device.....	18
Precharging Through an p-Device.....	20

Bootstrapping the Gate of the Pass Transistor.....	24
Simulation of Switching Rows.....	28
IV GENERATING CIRCUIT FOR BOOLEAN FUNCTION USING MAGIC FILES.....	33
V CONCLUSION.....	37
REFERENCES.....	38
APPENDIX	
A TABLES OF COMPARISONS.....	40
B FIGURES OF COMPARISONS.....	54
C PROGRAM CODE FOR LAYOUT GENERATION OF BOOLEAN FUNCTIONS.....	75
D EXAMPLES OF THE LAYOUT CIRCUITS.....	94

LIST OF TABLES

TABLE	PAGE
I Minimization Example, from Reference 5.....	7
II Precharging Through the Fabric Using nMOS Transistors ($W=5.5\mu$).....	19
III Precharging Through the Fabric Using pMOS Transistors ($W= 5.5\mu$)....	22
IV Precharging Through the Fabric Using Bootstrap Circuit at the Gate of Pass Device.....	25
V Bootstrapping Gates C and C (MOS Capacitor Being Eight Times Larger Than the Load).....	26
VI Charge Sharing Effect When Switching from One Row to Another (Total of 32 Rows).....	31

LIST OF FIGURES

FIGURE	PAGE
1-1(a). Non restoring nMOS half-adder.....	2
1-1(b). Non restoring nMOS full-adder.....	2
1-2(a). Restoring CMOS half-adder.....	3
1-2(b). Restoring CMOS full-adder.....	3
1-3(a). Restoring dynamic nMOS half-adder.....	3
1-3(b). Restoring dynamic nMOS full-adder.....	3
1-4. Timing diagram for restoring dynamic adder.....	4
1-5(a). Representation of a pass transistor network.....	5
1-5(b). Example of a sum generator for a full-adder.....	5
1-6(a). Karnaugh map, illustration of difference field.....	6
1-6(b). Example of a carry generator for a full-adder.....	6
2-1(a). Representation of a conventional CMOS pass element.....	12
2-1(b). Conventional CMOS pass transistor 4-to-1 multiplexer circuit.....	12
2-2. Physical structure of an nMOS transistor.....	13
2-3. Physical structure of a pMOS transistor.....	13
2-4. An element of the nMOS differential pass transistor.....	15
2-5(a). Representation of an MOS pass transistor.....	16
2-5(b). 4-to-1 multiplexer circuit of differential pass transistor logic.....	16
2-6(a). Static differential buffer circuit diagram.....	17
2-6(b). Clocked differential buffer circuit diagram.....	17

3-1. Precharging the fabric through a row of serially connected nMOS transistors.....	20
3-2. Timing diagram for precharging a row of serially connected nMOS transistors.....	21
3-3. Bootstrap circuit used at the gate of the pass transistors.....	23
3-4. Timing diagram of the bootstrap circuit.....	24
3-5. Model for latch up condition.....	27
3-6. Resistive model for latch up.....	28
3-7. 32 rows of pass transistor logic for charge sharing simulation.....	29
3-8. Timing diagram for charge sharing simulation.....	30
3-9(a). Switching from one row to another without precharging.....	32
3-9(b). Switching from one row to another with precharging.....	32

CHAPTER I

INTRODUCTION

Differential pass transistor logic is used for implementing and evaluating Boolean functions. The nMOS approach is taken in the design of pass transistor logic, because studies show that it is the most effective in terms of area consumption for implementing Boolean functions for a large number of vectors of the same format. Moreover, pass transistor realizations using minimum size results in area savings and high operating speed when compared with gate logic realization [1]. It has been demonstrated that nMOS pass transistor logic with the gate driven by bootstrap, using a voltage above V_{dd} level, offers significant improvement in speed. The SPICE2G.6, level 2 model is used for simulation of pass transistor logic. The TSPICE level 2 model is used for simulation of the bootstrap circuit [2]. The C language in combination with the CFL program is used to generate the layout of the circuit. The C-code reads MAGIC files from a library and generates the layout of the circuit. This program, BFG, also requires for the input vectors to the generator to be of the same format. The Boolean function can have up to fifteen control variables with one pass variable. A vector larger than five control variables will be broken into two or more sub-circuits. The generated circuits have been simulated using the Fastsim logic simulation [3].

Static restored logic corresponds to an output voltage which is strong enough to drive other stages in a cascaded circuit. In other words, the output is at either a strong high or low voltage level. Here, an example of a non-restored logic is considered in Figure 1-1a, which shows a half adder where the information bits are A, \bar{A}, B and \bar{B} . The output of the half adder is either 0 volts or $V_{dd}-V_t$ volts. The output of Figure 1-1b is the

sum of the two half adder and the carry bit. Since the voltage that is driving H and \bar{H} are $V_{dd}-V_t$ volts, the output voltage of the sum can either be 0 volts or $V_{dd}-2V_t$ volts. At this stage, $V_{dd}-2V_t$ is not a strong enough

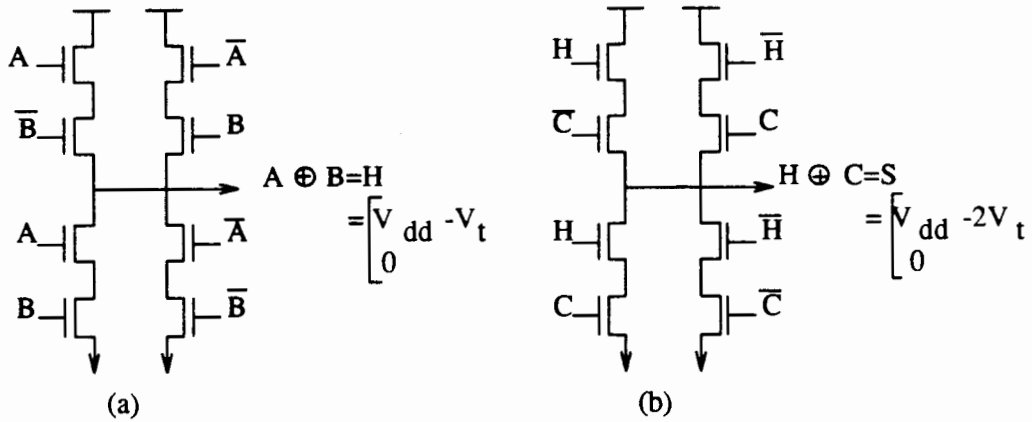


Figure 1-1. Non restoring nMOS adder
 (a) non restoring nMOS half-adder.
 (b) non restoring nMOS full-adder.

output to drive a next stage in a cascaded situation. However in the CMOS circuits shown in Figure 1-2a and b, where the pull up is a pMOS transistor and the pull down is an nMOS transistor, a strong logic high can be passed from V_{dd} through the p-device to the output node, and a strong logic low can be achieved through the n-device at the output node (the characteristic of the p-device and n-device are considered in more detail in Chapter II). The output of the half adder shown in Figure 1-2a, is either 0 volts or V_{dd} . In this case, the output is at strong logic low or high and can be used in cascaded circuits.

Dynamic design, shows in Figure 1-3a and b, the output node needs to be precharged and then evaluated, In Figure 1-3a, the output is precharged during the precharge cycle to $V_{dd}-V_t$ volts. During the precharge cycle all the switches, A, \bar{A}, B and \bar{B} are closed as shown in Figure 1-4. During the evaluation cycle depending on the information bits, A, \bar{A}, B and \bar{B} , the output node can either discharge to 0 volts or stay at $V_{dd}-V_t$ volts. In this case the output is restored to strong logic low or high and can be

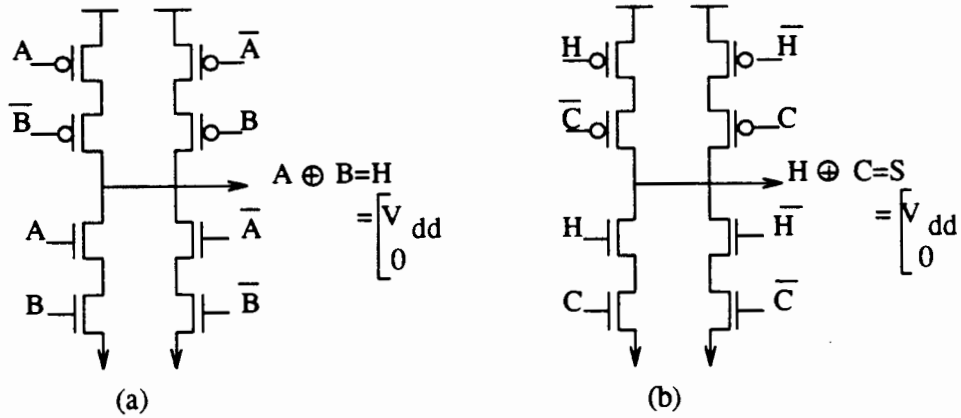


Figure 1-2. Restoring CMOS adder
 (a) restoring CMOS half-adder.
 (b) restoring CMOS full-adder.

used in a cascaded circuit.

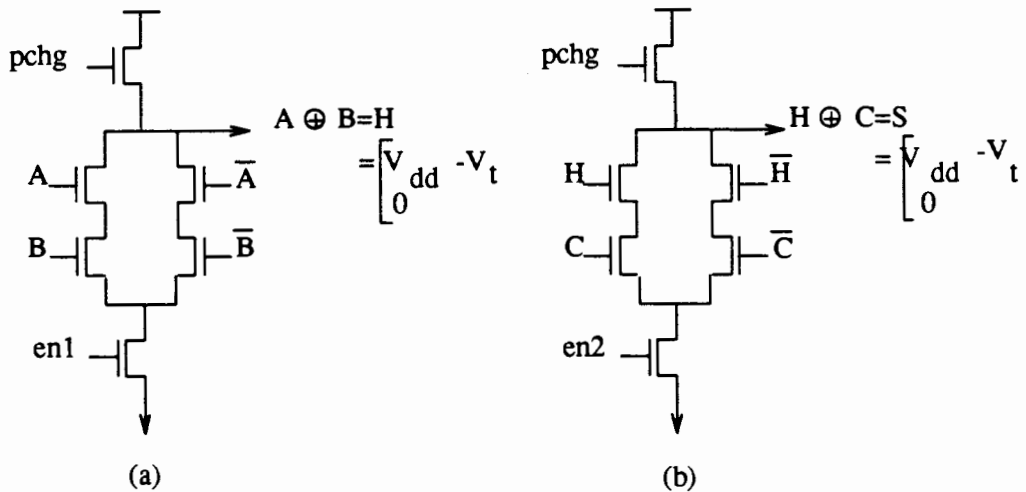


Figure 1-3. Restoring dynamic nMOS adder
 (a) restoring dynamic nMOS half-adder.
 (b) restoring dynamic nMOS full-adder.

An element of a differential pass transistor consists of two n-channel transistors, controlled by the same signal, which passes the input and its complement to the output. If the pass transistor is of nMOS type the control signal that closes the gate is high, and if it is of pMOS type, then the control signal that closes the gate is low. The CMOS transmis-

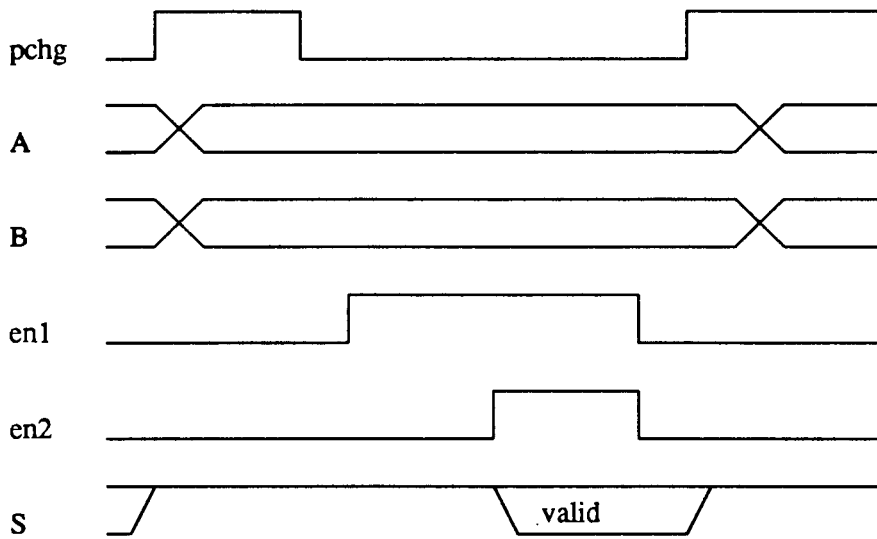


Figure 1-4. Timing diagram for restoring dynamic adder.

sion gate consists of nMOS and pMOS transistors.

Pass transistor logic elements are bidirectional, since which terminals acts as the drain and source depends on their voltage levels. A pass transistor is, in fact, a switch, much like the earlier contact relay [4, 5]. The pass element can take any value of the set $\{0, 1, x_i, \bar{x}_i, z\}$ where x_i is an input variable and z is the high impedance state [1]. The high impedance results when all of the switches are off. To avoid a high impedance output, a 0 must be passed whenever a 0 is required at the output. This implies that in the Karnaugh map all 0 entries as well as 1 entries must be grouped together [1, 6]. The signals that drive the gates of the MOS transistor are called control signals. The input signals that are fed into the pass transistors and are passed to the output are called pass signals.

In this study the networks are realized with only nMOS transistors. A general form of a pass transistor network is shown in Figure 1-5a and b. Each row is composed of several pass transistor which are connected serially and each transistor row is called a product term, P_i . The input, V_i , is passed to the output, F , when all the switches in one row are enabled. The gate of each product term is controlled by the control signal, C_i . A

wired OR at the output node connects the rows. An example of the sum generator for the full adder is shown in Figure 1-5b, where the pass function is $F = \bar{A}\bar{B}(C) + \bar{A}B(\bar{C}) + A\bar{B}(\bar{C})$.

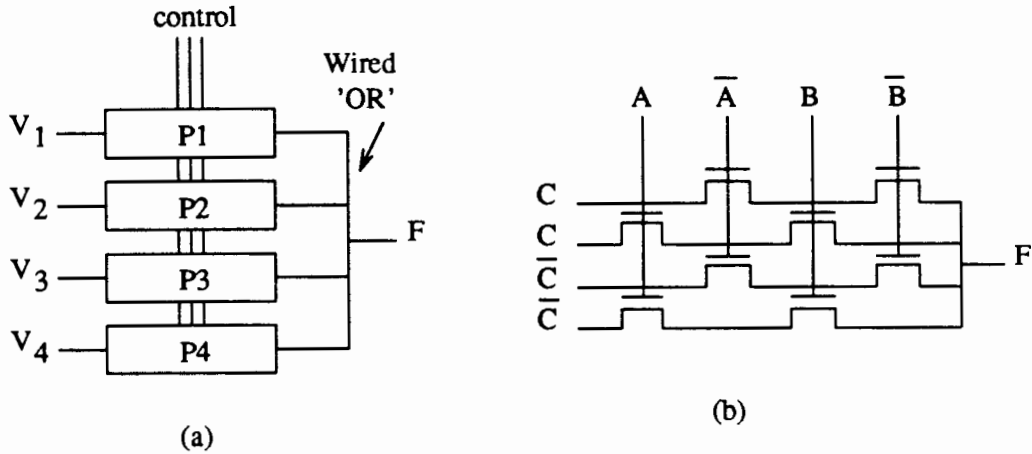


Figure 1-5. Pass transistor network
 (a) representation of a pass transistor network.
 (b) example of a sum generator for a full adder.

Thus the output of the pass transistor network can be expressed as $F = P_1(V_1) + P_2(V_2) + \dots + P_n(V_n)$ [1, 7].

Example 1: A three variable Karnaugh map is shown in Figure 1-6a. Cells 0 and 1 constitute the pass implicant of $\bar{A}\bar{B}$ with a pass variable of 0, which can be denoted as $\bar{A}\bar{B}(0)$. Cells 2 and 3 constitute the pass implicant with a pass variable of $\bar{A}\bar{B}(C)$. Respectively cells 4 and 5 form a pass implicant of $\bar{A}\bar{B}(C)$, and cells 6 and 7 form a pass implicant of $AB(1)$. The whole function can be expressed as $F = \bar{A}\bar{B}(0) + \bar{A}\bar{B}(C) + \bar{A}\bar{B}(C) + AB(1)$. This function is the carry generator of a full adder, which is shown in Figure 1-6b [1, 7, 8].

Since the Karnaugh map is most effective for a networks up to five or six variables, an algorithm for networks larger than six variables has been developed by D. Radhakrishnan by modifying the conventional Quine-McCluskey approach. In the Quine-McCluskey method where tabular method is used to drive a minimal sum when all prime implicants of a given function F are known. In this method, two vectors that

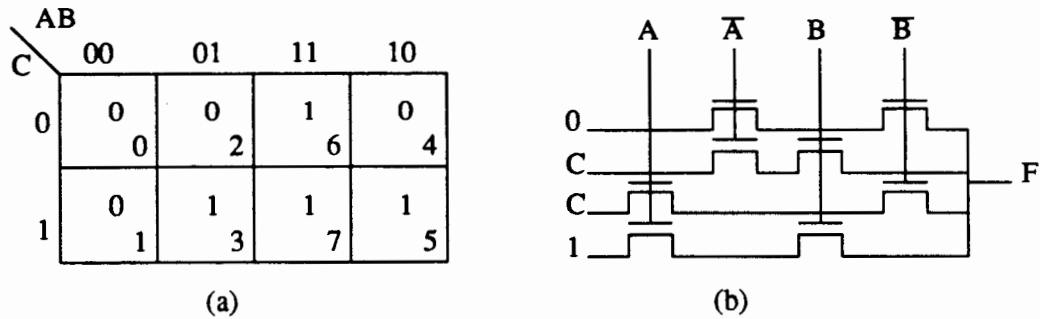


Figure 1-6. Conventional CMOS pass-transistor
 (a) Karnaugh map, illustration of difference field.
 (b) example of a carry generator for a full adder.

are combining must differ only in one bit and the F is given in its minterm expansion [4, 5].

The format for the data structure of Radhakrishnan approach is in the form of base[difference]variable. The base field is an integer field, and it is the smallest value minterm between the grouping of minterms. Two pass implicants can be combined only if their bases vary by one bit. The difference field is also an integer field, consisting of one or more entries separated by commas. It represents the difference between the two fields that are combining together. The difference field between the two terms whose decimal equivalents are 2 and 3, the difference in this case is (1). On the other hand for 2(1) and 6(1) the difference is (1, 4). The order of the entries in a difference field makes no difference, thus (1, 4) and (4, 1) are equivalent. The pass field is an alphanumeric field which can have any value from the set $\{0, 1, x_i, \text{and } \bar{x}_i\}$. The pass implicants are:

1. if $F_i=0$ and $F_j=0$ then $i[2^k]0$;
2. if $F_i=1$ and $F_j=1$ then $i[2^k]1$;
3. if $F_i=0$ and $F_j=1$ then $i[2^k]x_k$;
4. if $F_i=1$ and $F_j=0$ then $i[2^k]\bar{x}_k$;

In the example shown in TABLE I, the prime implicants that cover all eight terms

TABLE I
MINIMIZATION EXAMPLE, FROM REFERENCE 5

TRUTH TABLE				OUTPUT	BASE(DIFFERENCE)PASS	
(BINARY)		(DEC)			List1	List2
X2	X1	X0	X	F		
0	0	0	0	0	0(1)0 0(2)X1 0(4)0	0(1, 2)X1
0	0	1	1	0	1(2)X1 1(4)X2	
0	1	0	2	1	2(1)1 2(4)X2	
1	0	0	4	0	4(1)X0 4(2)0	4(1, 2)X0
0	1	1	3	1	3(4)1	
1	0	1	5	1	5(2)1	
1	1	0	6	0	6(1)X0	
1	1	1	7	1		

of the three variable function are $0(1, 2)x1$ and $4(1, 2)x0$, which Radhakrishnan indicates can be represented as $\overline{x_2}(x_1)$ and $x_2(x_0)$, respectively [1, 6]. Thus, the pass function for the network is $F = \overline{x_2}(x_1) + x_2(x_0)$, which can be implemented with nMOS transistors.

Differential pass transistor logic with an nMOS fabric is used in this study. The pass variable and its complement are passed from the input to the output node. At the output node a wired OR connects all the OUT+ nodes together and another wired OR

connects all the OUT- nodes together. The network then has two differential outputs, OUT+ and OUT-, which drive a differential buffer, (see Figure 2-3). The output nodes and the entire fabric are precharged during a precharge cycle. Then, one of the nodes, depending on the pass variable and the state of the control signals, is discharged to ground through the buffer, during an evaluation cycle. In differential pass transistor logic the number of transistors is almost twice as large as in ordinary pass transistor logic, but the overhead consumption is not as much as in CMOS transmission gates since all the transistors are n-type and stay in the same well. In this study it is shown that this approach has area and speed advantages over conventional CMOS pass transistor logic.

Since possible loss of information can occur in the ordinary pass transistor logic because of problems with noise margin, this study uses differential pass transistor logic. A differential pass transistor is also faster, since we need a differential voltage level of about $\frac{V_{dd}}{2}$ at the output node instead of full CMOS level voltages. Another advantage of this type of configuration is that it produces complementary outputs at the output node [9, 10].

In Chapter II, a study and comparison of MOS pass transistor logic is done. This compares the behavior of nMOS, pMOS and CMOS transistor logic. Chapter III, focuses on differential pass transistor logic in nMOS, the timing behavior of precharge and evaluation cycles and also shows that using a bootstrap circuit at the gate (the control signal) of the nMOS pass transistor logic will significantly improve the speed. In this Chapter simulation for pass transistor logic is presented. The difference in precharge time, using an n-type precharge device versus a p-type precharge device is shown. The simulation result shows that when switching from one row to another row (see Figure 3-7), the output voltage will discharge through the fabric by about $\frac{V_{dd}}{2}$ volts. This is in a case of having only five transistors in one row. This voltage drop can get worse as the

number of transistors is increased. For this reason, precharging before each evaluation is necessary, since we can not afford this voltage drop at the output node. The simulation of the bootstrap circuit is also shown in this Chapter and possible problems that might occur in bootstrapping are discussed.

In Chapter IV, by considering the criteria that are discussed in the above Chapters, a program for layout generation of differential pass transistor fabric is developed. For the circuit generation of specific Boolean functions, C-code in combination with CFL is used. The program reads an input file describing the Boolean function and generates a layout circuit of the function in the MAGIC form which is called "BFG" (Boolean Function Generator). The layout circuit is in 3μ p-well CMOS technology. Finally, Chapter V concludes the work and summarizes our results of the study of differential pass transistor logic.

CHAPTER II

STUDY AND COMPARISON OF MOS PASS TRANSISTOR LOGIC

Pass transistor logic is used for implementation of Boolean functions. There are three main characteristics of pass transistor logic: non-restored gate (e.g. restoring gate converts input low level to zero volts output and input high level to full V_{dd} output), bidirectional devices (since drain and source of a pass device depends on its voltage potential level), and implement combinational logic with one output.

The operation of an MOS transistor is based on the terminal potentials. There are two primary types of transistors, enhancement-mode and depletion-mode. Since most high-density integrated circuits are built with enhancement devices, this study considers only enhancement devices. In an enhancement-mode, normally off, nMOS transistor, when the potential of the gate is below threshold voltage, electrons are prevented from flowing from source to drain of the transistor (i.e. no channel). This is because of the built-in potential of the p-n diode formed between the n-type source and the p-type substrate. A positive voltage (above threshold) on the gate with respect to the substrate will increase the number of electrons in the channel and, hence, increase the conductivity of the channel. Thus to turn on an enhancement-mode transistor, positive charges must be placed on the gate.

There are two types of enhancement devices, nMOS and pMOS. Operating with a 5 volts power supply, the threshold voltage, V_t , of enhancement-mode device is in the range of 0.6 to 1.1 volts for nMOS devices. The logic high output voltage of an nMOS device is $V_{dd}-V_t$. The threshold voltage, V_t , of enhancement-mode for pMOS device is in

the range of -0.6 to -1.1 volts with reference to source voltage. The logic high output voltage of a pMOS device is at V_{dd} level [11]. For an nMOS device, a high voltage on the gate will turn the transistor on, whereas for a pMOS a low voltage on the gate will turn the transistor on.

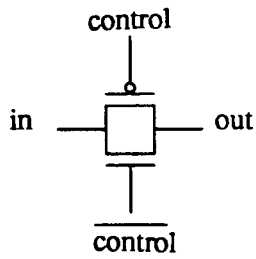
There are two types of enhancement devices, nMOS and pMOS. These may be combined to form complementary MOS (CMOS). The CMOS pass transistor, Figure 2-1 consists of an n-channel transistor and a p-channel transistor. The gates of the n-channel and the p-channel are opposite signals, C and \bar{C} , but they have common source and drain connections. The characteristics of n and p are described below.

MOS PASS TRANSISTORS

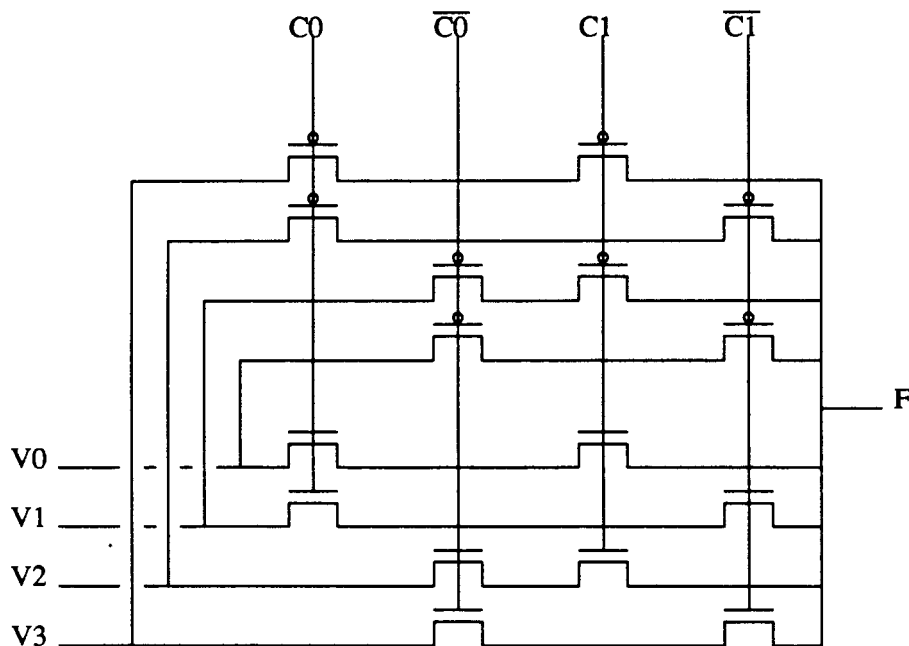
1. nMOS Pass Transistor

In an n-channel enhancement type transistor, shown in Figure 2-2, the substrate is doped with p-type silicon. The source and drain, which are diffused into the substrate, are heavily doped n^+ regions. Between the source and drain there is a narrow region of p-type substrate, channel, which is covered by a thin isolating layer of silicon dioxide (SiO_2) gate oxide. Over this gate oxide there is a polycrystalline silicon (polysilicon), which is called the gate.

In case of a positive voltage applied between the source and the drain (V_{ds}), with control signal C being in off state ($V_{gs}=0$ volts) there would be no current flow between the source and drain of the transistor. If the load capacitance, C_l , at the output is discharged the output voltage, V_o , will stay at the ground level independent of the input voltage, V_{in} . When the control signal is on high state ($V_{gs}=5$ volts) and $V_{in}=5$ volts the pass transistor begins to conduct and charges the load capacitance toward $V_{dd}-V_{in}$, where V_{in} is the threshold of the n-device. When the output voltage, V_o , reaches $V_{dd}-V_t$, the n-device begins to turn off. At this point the channel of the transistor is shut off and the



(a)



(b)

Figure 2-1. Conventional CMOS pass-transistor
 (a) Representation of a conventional CMOS pass element.
 (b) conventional CMOS pass transistor 4 to 1 multiplexer circuit.

load capacitance, C_l will remain at $V_{dd} - V_{tn}$. This implies that the nMOS pass transistor does not pass the full voltage level. In the case in which the control signal is at high state and $V_{in} = 0$ volts and the load capacitance is charged, the pass transistor begins to conduct and discharge the load capacitor to V_{ss} . The n-device can discharge a capacitor to ground, and thus it is a strong logic low [12].

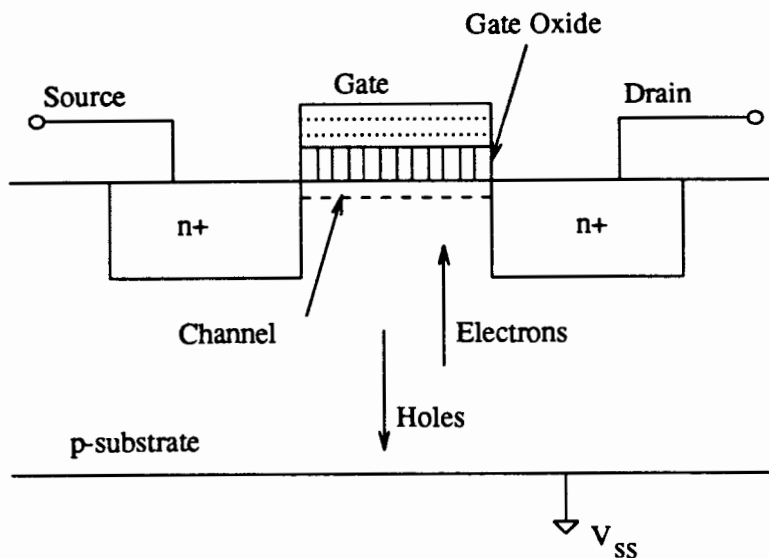


Figure 2-2. Physical structure of an nMOS transistor.

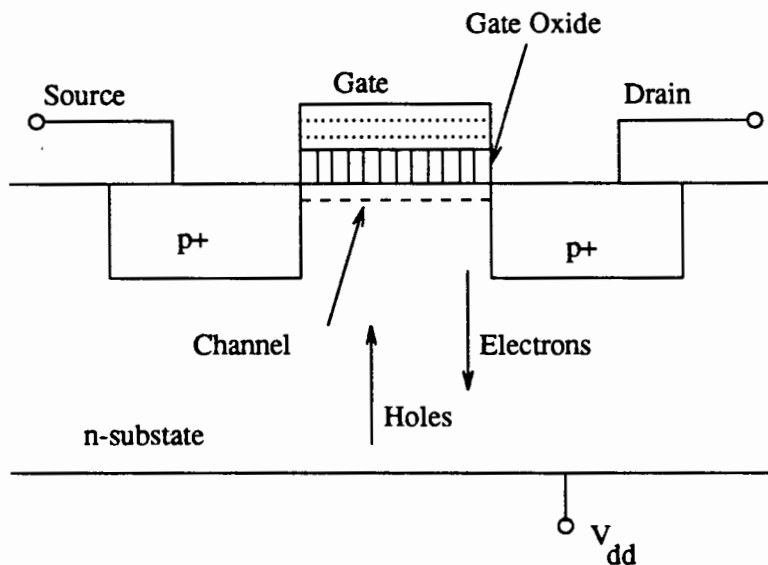


Figure 2-3. Physical structure of a pMOS transistor.

2. pMOS Pass Transistor

In a p-channel enhancement type transistor, shown in Figure 2-3, the substrate is doped p+. In the pMOS pass transistor, when the control signal (V_{gs}) is at high voltage, the gate is turned off. Regardless of the input voltage, the load capacitance remain

unchanged. When the control signal (V_{gs}) is at low voltage, it draws holes into the channel region below the gate. As the result a channel is created under the gate and a conduction path is created between the source-to-drain. In nMOS conduction results from movement of electrons. However, in the case of pMOS conduction results from the movement of holes in the channel. Thus a negative voltage at the gate (w.r.t. source) causes the current to flow and charge the load capacitor to V_{dd} . However, in the case in which $V_{in}=0$ volts and $V_o=5$ volts, the load capacitor discharges through the p-device until $V_{in}=V_o-V_{tp}$, while the output V_o remains at V_{tp} , where V_{tp} is the threshold voltage of the p-device. Thus the pMOS pass transistor unlike nMOS, does not conduct a strong logic low [12].

3. CMOS Transmission Gate

The CMOS transmission gate has the advantage of passing both a strong logic high through its p-device and a strong logic low through its n-device. However p and n-devices together consumes much area and this is not desirable. Another constraint of the CMOS transmission gate is that both V_{ss} and V_{dd} rails have to be presented in the circuit, whereas in the pass transistor logic V_{dd} or V_{ss} are not necessary, since at the output node we are looking at the differential pair. Thus, although it is clear that CMOS offers certain advantages, nMOS is still preferable.

The main reasons for using nMOS pass transistor logic instead of CMOS in this study are to substantially improve speed and reduce area (since all the transistors on the same substrate). The pass transistor used for implementation of this study is an n-channel transistor which eliminates the slow p-channel transistor and the extra area that p-channels consume (since they have to be build on a different well from the n-channel transistor). One disadvantage of the n-channel transistor is that it precharges the fabric slowly and increases precharge time as the number of serial transistors increase.

IMPLEMENTING THE nMOS DIFFERENTIAL PASS TRANSISTOR

An element of the nMOS differential pass transistor logic which consists of two n-channel transistors, controlled by the same signal that passes input and its complement to the output, is shown in Figure 2-4.

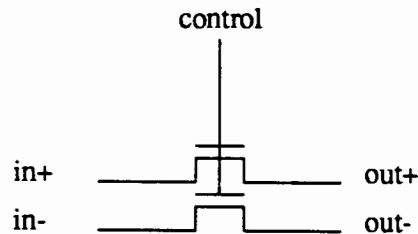


Figure 2-4. An element of the nMOS differential pass transistor.

This differential pass has two outputs, OUT+ and OUT-. Since an n-channel transistor can pass a strong logic zero, the value of the output can be exactly zero, however, since n-channel transistor can not pass a strong logic high, the maximum voltage level at the output node that is passed through the n-channel transistor is $V_{dd} - V_t$. This differential output signal can be restored to its full logic level by using a differential buffer at the output (see Figure 5-6)[6, 13, 14].

A common application of this type of logic configuration is a multiplexer in which a pass variable V_i is passed to the output depending on the state of the control signal C_i . A differential pass-transistor multiplexer network is illustrated as in Figure 4-5. Buffers are placed at the input of the pass network, to provide a set of a complemented inputs. At the output node, when the control variables enable the switches of the product term C_i in one row, the product term of the adjacent rows are disabled resulting in a high-impedance state [6]. "Two wired OR at the output node, OUT+ and OUT-, sums all of these possible products". Thus, the input variable V_i and its complement are passed to the output, OUT+ and OUT-, when the product term C_i of that row is on.

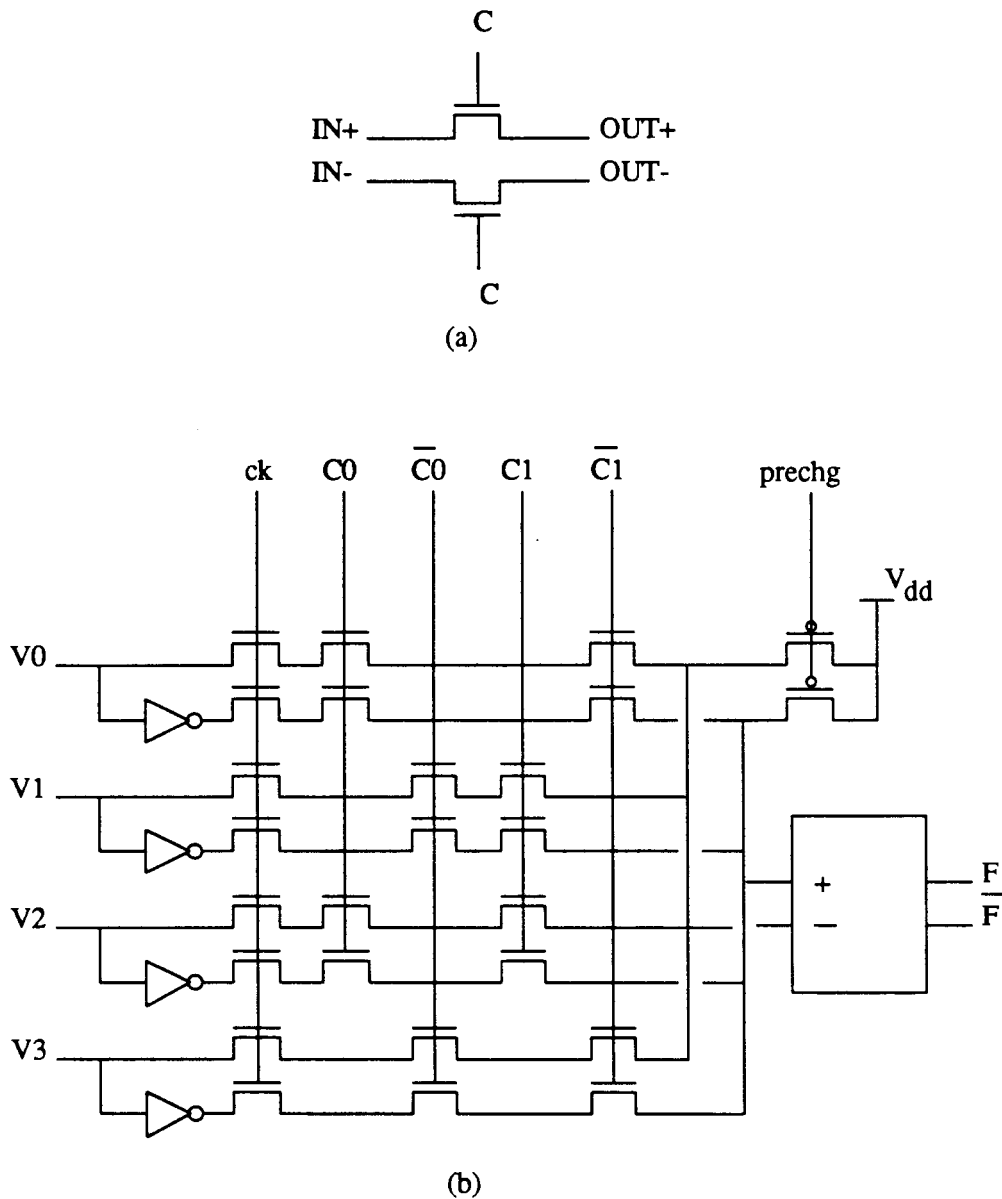


Figure 2-5. Differential pass-transistor logic
 (a) Representation of an nMOS pass transistor.
 (b) 4-to-1 multiplexer circuit of differential pass transistor logic.

The network has a set of differential outputs OUT+ and OUT-. This differential signal is restored to normal logic level by passing the signal through differential buffers as shown in Figure 2-5(a) and (b), respectively. One type of differential amplifier that can be used for our purpose is static differential buffer that is shown in Figure 2-6(a).

This buffer is a cascode voltage switch logic (CVSL) inverter [15]. Another type of differential amplifier is clocked buffer which is shown in Figure 2-6(b), a RAM sense amplifier [6].

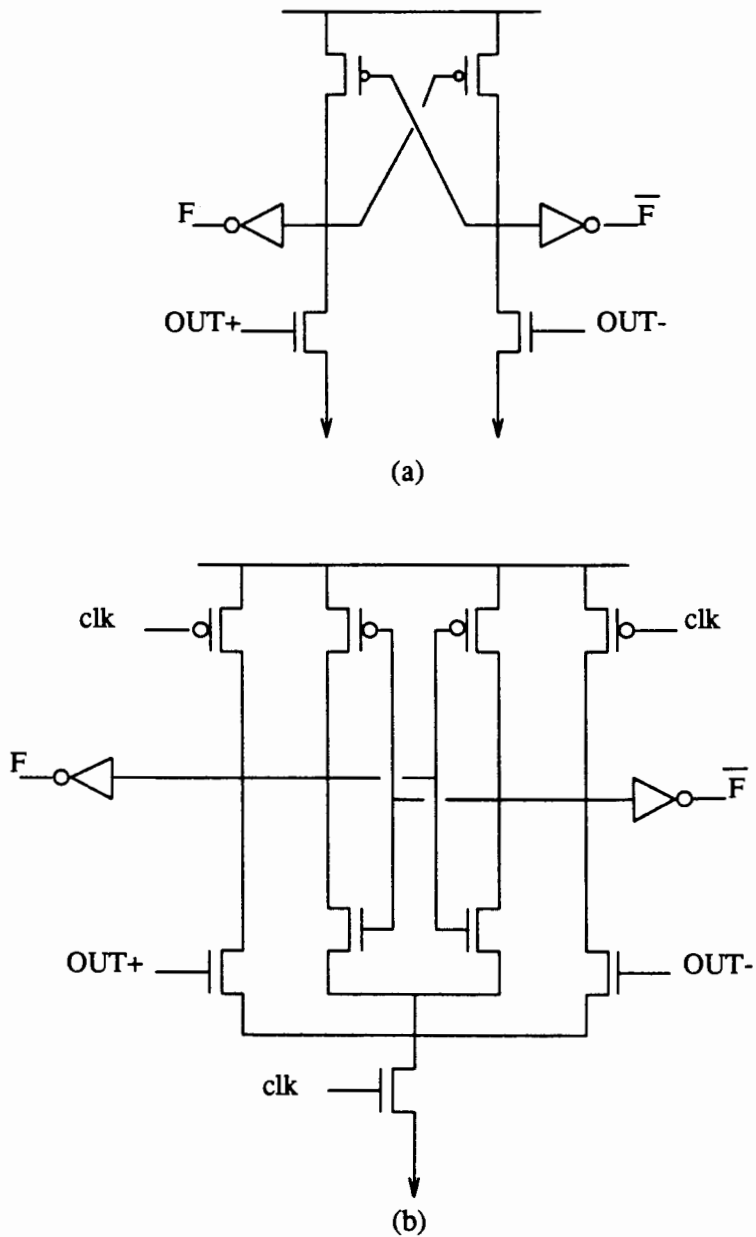


Figure 2-6. DPTL decoders
 (a) Static differential buffer circuit diagram.
 (b) Clocked differential buffer circuit diagram.

CHAPTER III

ANALYSIS OF MOS PASS TRANSISTOR LOGIC AND BOOTSTRAP CIRCUIT

The SPICE2G.6, level 2 model is used for simulating the pass transistor logic. The TSPICE, level 2 model is used for simulating the bootstrap circuit.

PRECHARGING THE FABRIC

It is known that nMOS transistor works faster discharging the capacitor than for precharging it. The reason for this is that the number of electrons in the channel of the nMOS device increases as the source voltage goes to ground (in other words resistivity of the device decreases as the source voltage goes to ground). An opposite situation occurs when the nMOS device precharges a capacitor. The precharge slows down as the voltage rises and stops when V_{gs} reaches to $V_{dd}-V_t$ level.

1. Precharging Through an n-Device

In the nMOS design it is a common practice to precharge all the dynamic nodes and then selectively discharge according to the input data. Precharging through the fabric of serially connected nMOS transistors, the current will penetrate through the fabric, and the time required to precharge the fabric to the desired voltage level will increase as the number of serially connected transistor increases. For instance in simulation one illustrated in TABLE II, the fabric is precharged through five serially connected transistors with an nMOS precharge device of the size $W=5.5\mu$. It takes 27ns to precharge to 3 volts, see TABLE II.

TABLE II
PRECHARGING THROUGH THE FABRIC USING
nMOS TRANSISTOR ($W=5.5\mu$)

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.440	0.000	0.000	0.000	0.000	0.000
3	2.742	0.000	0.000	0.000	0.000	3.488
6	2.294	0.3586	0.2569	0.000	0.000	3.563
9	1.214	1.214	1.214	0.000	0.000	3.594
12	1.124	1.129	1.161	1.191	1.199	3.616
15	1.250	1.254	1.281	1.337	1.445	3.434
18	2.198	2.209	2.276	2.405	2.613	2.953
21	2.678	2.685	2.730	2.817	2.957	3.186
24	2.920	2.925	2.959	3.024	3.128	3.300
27	3.063	3.068	3.094	3.146	3.229	3.367
30	3.159	3.162	3.184	3.227	3.296	3.412
33	3.227	3.230	3.249	3.285	3.345	3.445
36	3.278	3.280	3.297	3.329	3.381	3.469
39	3.317	3.319	3.334	3.362	3.409	3.488
42	3.349	3.351	3.364	3.389	3.431	3.503
50	3.408	3.410	3.420	3.440	3.473	3.531

For this experiment, the precharge device was chosen to be an nMOS device with the size $W=5.5\mu$ transistor. The experiment is set up in the following manner. The precharge transistor is on at 0ns and the precharge of the fabric is enabled. The transistor M_1 is turned off during the precharge cycle. Initially all transistors are off. In 44ns all

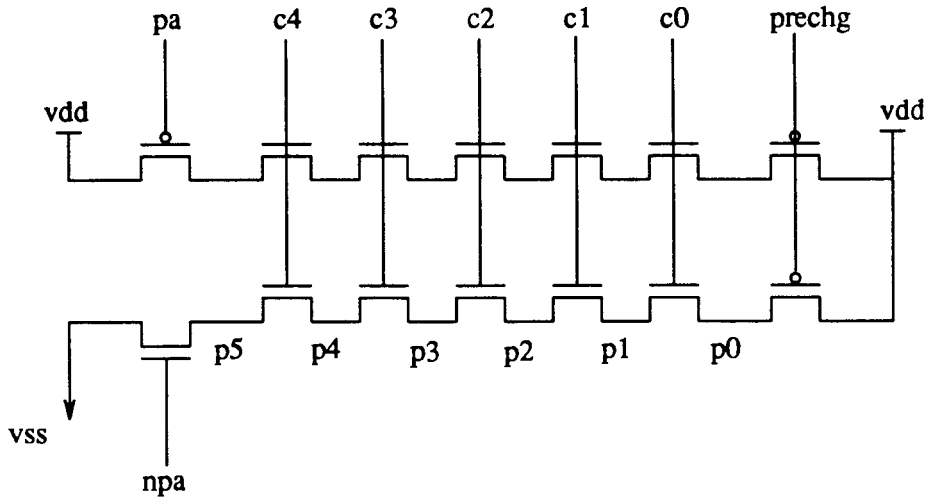


Figure 3-1. Precharging the fabric through a row of serially connected nMOS transistors.

transistors will be on, as they are turned on in succession, as illustrated in Figure 3-1 and 2. Thus nMOS transistor that are used for precharge device to precharge the fabric of serially connected n-type transistors are slow. Precharging the fabric through a p-type device can improve the precharge time, as shown in the next section.

2. Precharging Through a p-Device

In this simulation a p-device is used to precharge through the fabric, in the same way as in the last experiment (also see Figure 3-1 and 2). A p-device transistor turns on when the gate is at zero volts (w.r.t. source). The source of a p-device is connected to a 5 volts supply voltage. Thus, the gate source voltage is constant ($V_{gs}=5v$). The drain source current of the device is

$$I_{ds}=K(V_{gs}-|V_t|-\frac{V_{ds}}{2})V_{ds} \quad (3.1)$$

where

$$K=\mu_p C_{ox}(\frac{W_p}{L_p})_{eff} \quad (3.2)$$

and

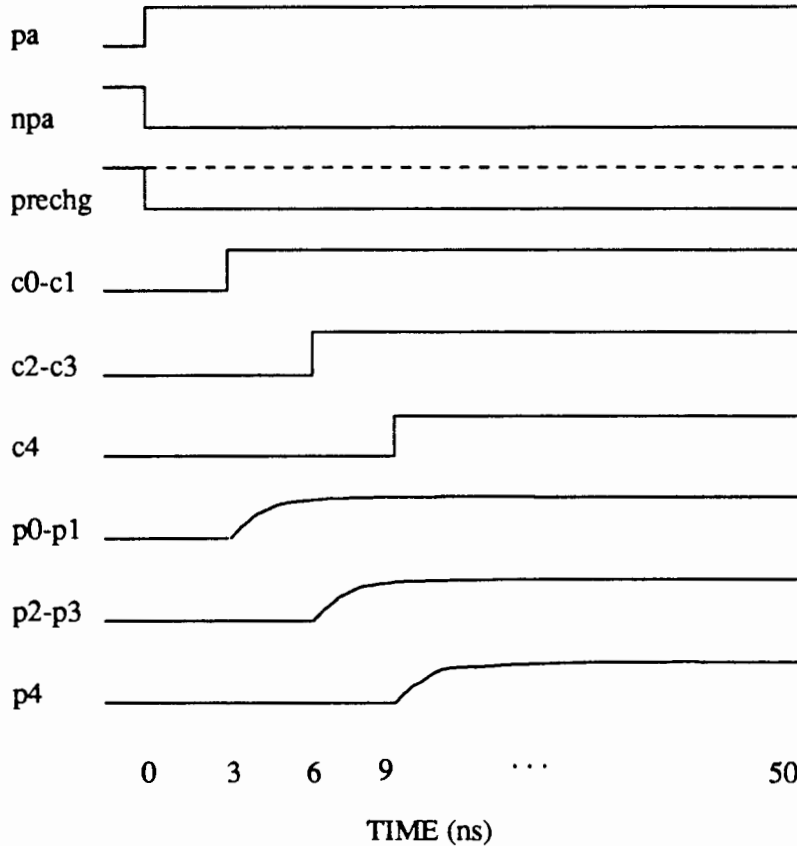


Figure 3-2. Timing diagram for precharging a row of serially connected nMOS transistors.

μ = mobility of holes

C_{ox} = gate oxide capacitance

W_p = channel width of p-device

L_p = channel length of p-device

The current I_{ds} will flow from source to drain of the p-device until the output is at 5 volts. Thus, the output voltage can be charged to a full voltage level through a p-device. The SPICE simulation using p-device shows much improvement in speed (Δt) for precharging the fabric. Using the p-device transistor for precharging the fabric, we have reduced the precharge time Δt to 24ns through the same number of transistors,

shown in TABLE III.

TABLE III
PRECHARGING THROUGH THE FABRIC USING
pMOS TRANSISTOR ($W=5.5\mu$)

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.452	0.000	0.000	0.000	0.000	0.000
3	2.742	0.000	0.000	0.000	0.000	5.003
6	2.340	0.3598	0.2606	0.000	0.000	5.000
9	1.219	1.218	1.219	0.000	0.000	5.000
12	1.127	1.132	1.164	1.195	1.202	5.000
15	1.256	1.260	1.287	1.346	1.457	4.878
18	2.319	2.333	2.415	2.577	2.854	4.798
21	2.826	2.834	2.886	2.987	3.162	4.927
24	3.050	3.056	3.094	3.167	3.294	4.962
27	3.177	3.182	3.211	3.268	3.369	4.977
30	3.259	3.262	3.286	3.333	3.417	4.985
33	3.315	3.318	3.338	3.379	3.450	4.989
36	3.357	3.359	3.377	3.412	3.475	4.992
39	3.389	3.391	3.407	3.438	3.494	4.994
42	3.415	3.416	3.430	3.458	3.508	4.995
50	3.462	3.463	3.474	3.496	3.536	4.997

Even though we have seen some improvement by precharging the fabric through a p-device, it is possible to improve the speed of the precharge cycle even further. One way to improve the speed of precharging is to turn the pass transistor up very high,

higher than 5 volts. This elevation of voltage can be achieved by using a bootstrapped device at the gate of the pass transistor. However a bootstrap circuit introduced into the circuit design must be handled carefully.

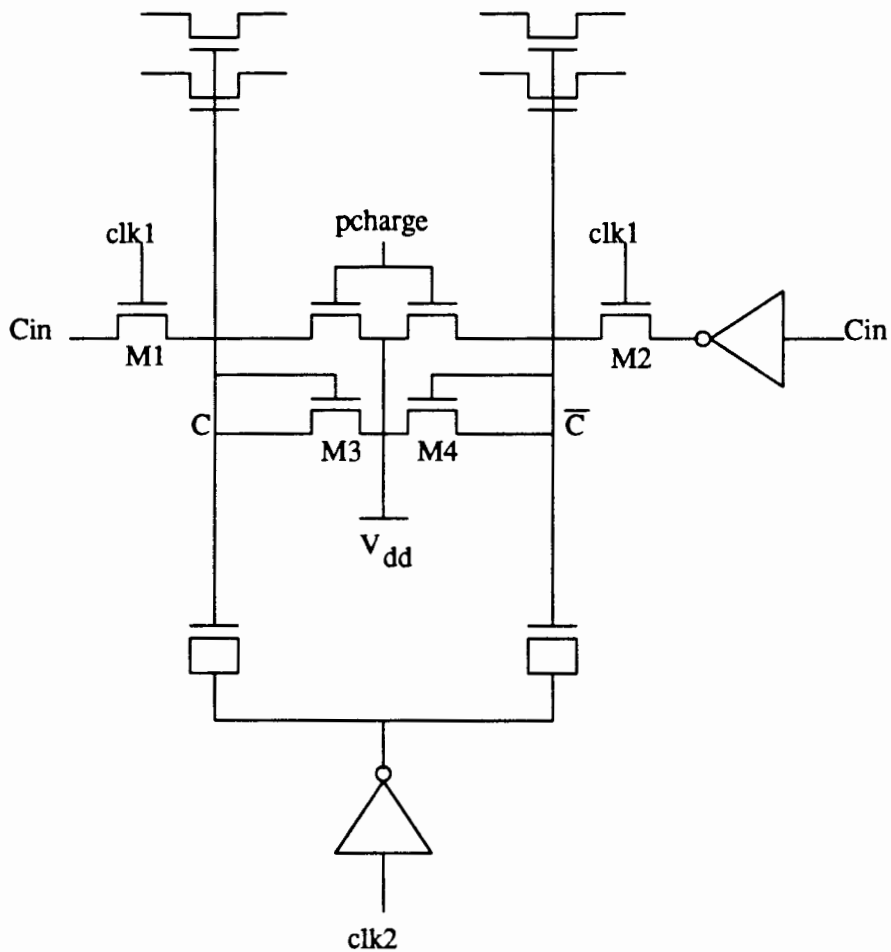


Figure 3-3. Bootstrap circuit used at the gate of the pass transistors.

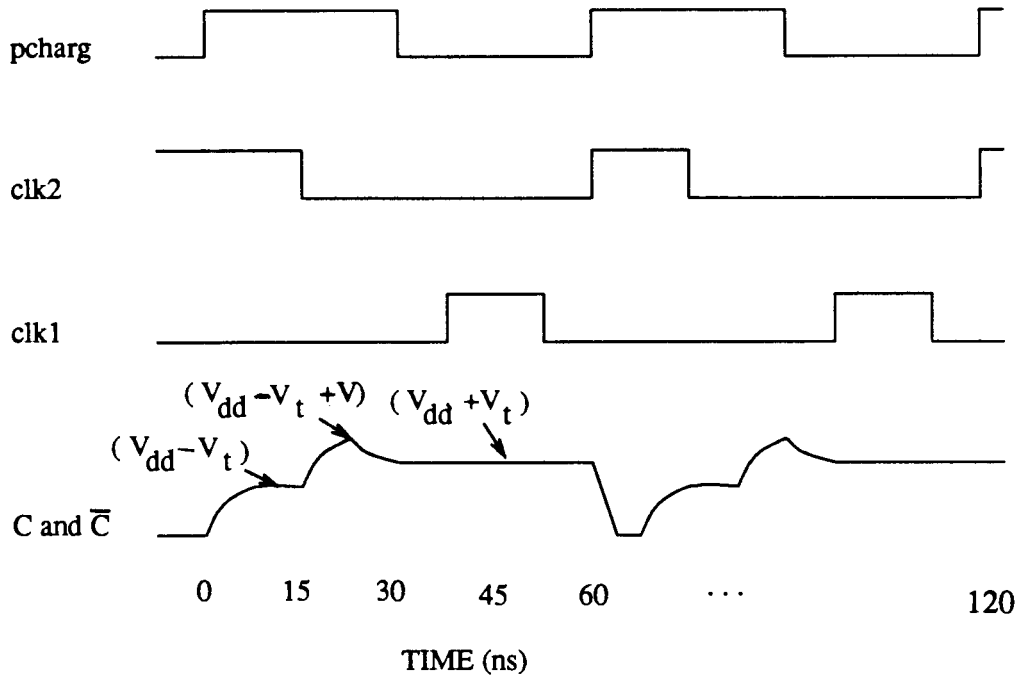


Figure 3-4. Timming diagram of Bootstrap circuit.

BOOTSTRAPPING THE GATE OF THE PASS TRANSISTOR

The bootstrap circuit in Figure 3-3 and 4. is used to boot the control signal and its complement. Using this type of configuration, the precharge time is reduced to 18ns through same number of transistors, see TABLE IV. During evaluation cycle, when the input signal C_{in} has a valid logic value, and $clk1$ is high, the boot node, C , will get charged to $V_{dd} - V_t$ volts through the M_1 transistor, and node \bar{C} will discharge to 0 volts through M_2 transistor and the inverter to ground. This is desirable since the C and \bar{C} are complements of each other during evaluation.

During a precharge cycle, in charging nodes C and the \bar{C} , $pcharge$ signal will enable and charge nodes C and \bar{C} to $V_{dd} - V_t$. In order to charge node C and \bar{C} above 5 volts, $clk2$ will turn off about 15ns later and charge node C and \bar{C} to $(V_{dd} - V_t) + V$, see TABLE V. The voltage of V depends on the size of the MOS capacitor. The size of the MOS capacitance has been determined by

TABLE IV
PRECHARGING THROUGH THE FABRIC USING
BOOTSTRAP CIRCUIT AT THE GATE OF
PASS DEVICE

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.364	0.000	0.000	0.000	0.000	0.000
2	2.738	0.000	0.000	0.000	0.000	4.991
4	2.746	0.000	0.000	0.000	0.000	5.000
6	1.710	0.4811	0.3524	0.000	0.000	5.000
8	1.463	1.447	1.428	0.000	0.000	5.000
10	1.407	1.403	1.377	0.4933	0.3579	5.000
12	1.362	1.368	1.398	1.429	1.435	5.000
14	1.418	1.419	1.421	1.426	1.433	5.000
16	2.002	2.020	2.136	2.380	2.838	4.333
18	3.036	3.051	3.142	3.322	3.626	4.749
20	3.496	3.506	3.570	3.696	3.910	4.884
30	4.123	4.126	4.151	4.199	4.283	4.983

$$\begin{aligned}
 cap &= \epsilon \times \frac{\epsilon_0}{t_{ox}} = \frac{4 \times 8.85 \times 10^{-14} f/cm}{488A} & (3.3) \\
 &= \frac{4 \times 8.85 \times 10^{-18} f/\mu}{.0488\mu} = \frac{4 \times 8.85}{4.88} \times 10^{-16} f/\mu^2 = 0.7 f/\mu^2
 \end{aligned}$$

where

ϵ_0 = permittivity of vacuum.

ϵ = dielectric constant of silicon dioxide.

t_{ox} = thickness of oxide which in our model is 488A.

The two transistors in diode configuration, M_3 and M_4 , are necessary in order to prevent the voltage level of C and \bar{C} to rise above $V_{dd}+V_t$ (see TABLE V), since the breakdown of gate oxide for this technology is low (about 8 volts). The surplus voltage of C and \bar{C} is discharged through the diode to the source V_{dd} .

TABLE V
BOOTSTRAPPING GATES C and \bar{C}
(MOS CAPACITOR BEING EIGHT TIMES LARGER THAN THE LOAD)

TIME (ns)	PCHARGE (volt)	CLK2 (volt)	CLK1 (volt)	C (volt)	\bar{C} (volt)
0	5	5	0	3.6692	3.6692
5	5	5	0	3.6692	3.6692
10	5	5	0	3.6692	3.6692
15	5	5	0	3.6692	3.6692
20	5	0	0	4.6767	4.7242
25	5	0	0	5.8008	5.9015
30	5	0	0	6.4392	6.5696
35	5	0	0	6.7187	6.8362
40	5	0	0	6.8205	6.9016
45	5	0	0	6.8533	6.9043
50	5	0	0	6.8590	6.8903

The transistors M_1 and M_2 are used to isolate the boot node, \bar{C} , from the pMOS transistor of the inverter. This is to prevent the discharge of the boot node to V_{dd} , through the drain-substrate of the p-n junction, shown in Figure 3-5.

The main problem with the bootstrapping circuit in CMOS is latch up. When drain of pMOS transistor is biased above substrate voltage level, injection of holes into

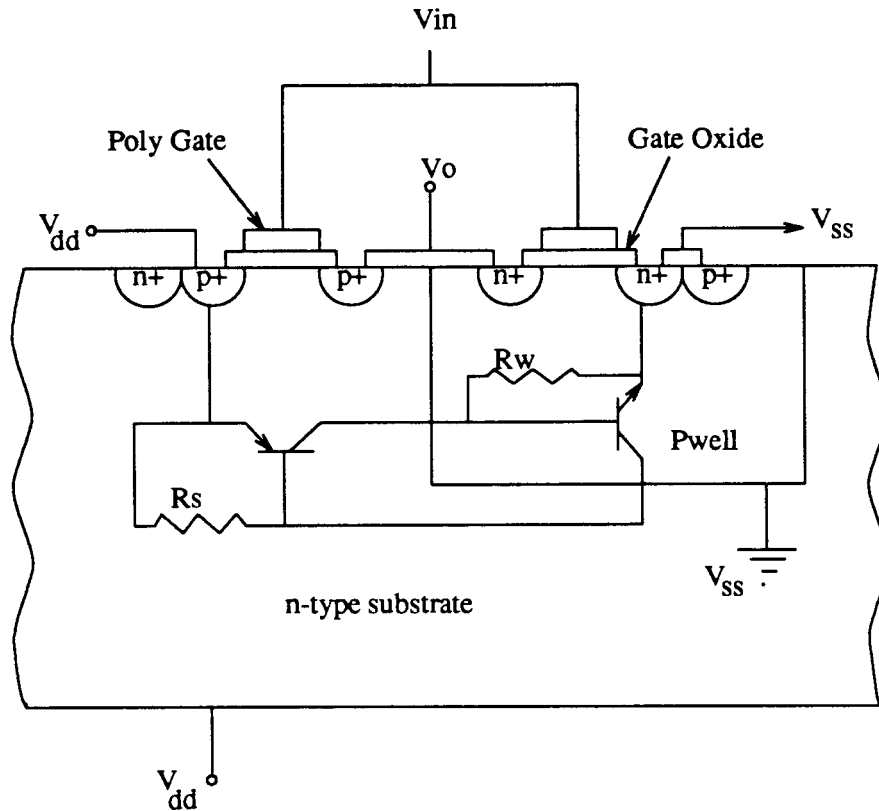


Figure 3-5. Model for latch up condition.

n-type substrate results. Injected holes get picked up by the closest p-well and elevate the voltage level of p-well. This in turn biases the p-n junction (of source-drain) of nMOS transistor. As the result, electrons are injected into the p-well. The amount of current leakage depends on many factors, such as, geometries of the injecting and collecting nodes, the distance between them, the location of other neighboring nodes, and the location of the closest substrate contacts.

The latch up models are shown in Figures 3-5 and 6. Conditions for bulk CMOS latch up are:

1. Base-emitter CMOS junctions of pnp and npn transistors are forward biased;
2. Beta product ($\beta_{npn}\beta_{pnp} > 1$) are sufficient to allow regeneration;

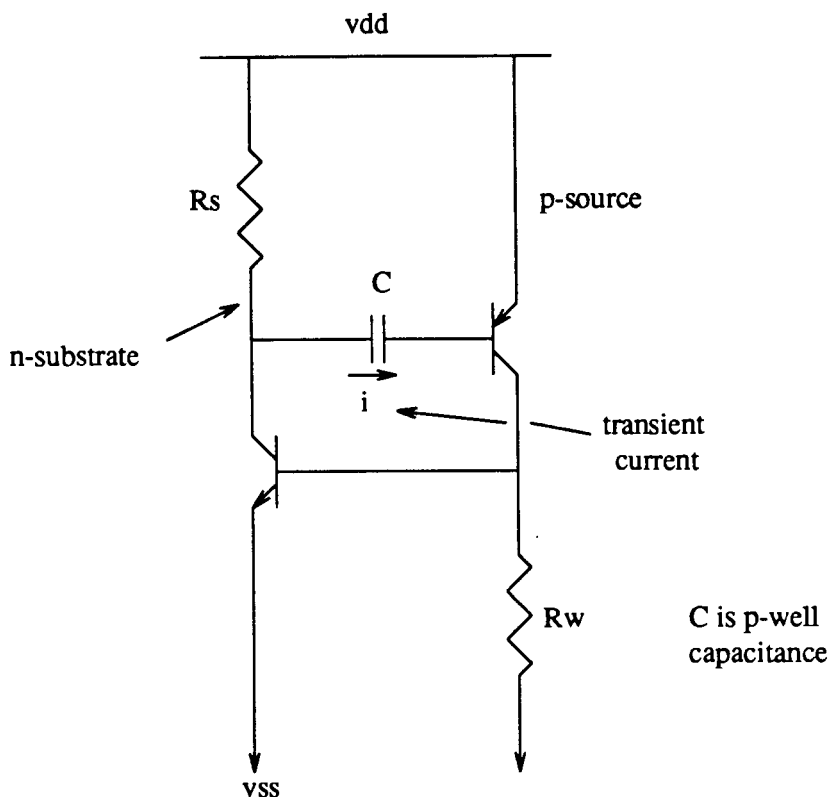


Figure 3-6. Resistance model for latch up.

3. The power supply V_{dd} increases or decreases abruptly. If we increase V_{dd} ,

$$i = -c \frac{dv}{dt} [11, 16] \quad (3.4)$$

current i will flow to the base of the transistor which causes it to latch up; and

4. Resistivity of substrate R_s and p-well R_w is high.

For a low-resistive p-well, the voltage drop across p-well will be smaller, so the injection of electrons will be correspondingly lower, and the possibility of latch-up is reduced. A similar condition holds for resistivity of n-substrate.

SIMULATION OF SWITCHING ROWS

In this section we will discuss the charge sharing problem of the pass transistor logic. As explained in the beginning of this chapter, voltage level of the output depends

on discharge (or precharge) through several serially connected pass devices. It also depends on charge sharing with the internal nodes of the discharge (or precharge) path. When switching from one row to another row of the pass transistor, we want to know how fast the output, $OUT+$, discharge and charges, $\frac{\Delta v}{\Delta t}$, shown in Figure 3-7.

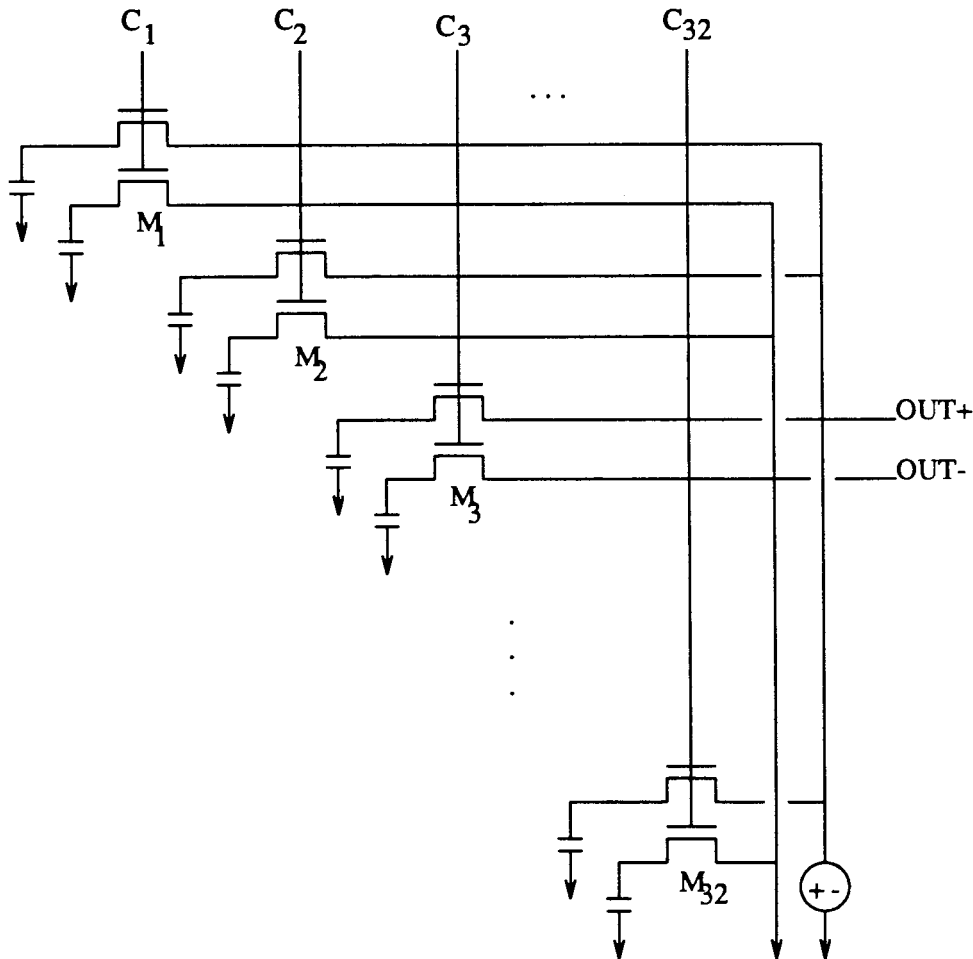


Figure 3-7. 32 rows of pass transistor logic for charge sharing simulation.

Since at the output nodes ($OUT+$ and $OUT-$) of the fabric, we are looking at the differential voltage, we need to know the initial state of the output nodes. In addition if we don't precharge the fabric the noise margin is essentially unknown. Thus output nodes, $OUT+$ and $OUT-$, have to be brought to a known initial state by precharging the

output, $OUT+$ and $OUT-$, nodes to 4 volts. All transistors are n-channel and they are chosen to be at their minimum size ($W=4.0\mu, L=3.0\mu$). A voltage source is placed at the nodes p_1 and p_{32} of the pass transistor. At the initial condition, the control signal, C_1 , at the gate of the M_1 transistor is on, and control signal of all other transistor are off. Initially the output $OUT+$ is at 4.0 volts and $OUT-$ is at ground level, 0 volts. At 10ns, M_1 goes from an on state to an off state, and M_{32} goes from an off state to an on state, see Figure 3-8. At this point the output voltage remains at its initial voltage level as expected, since there is no

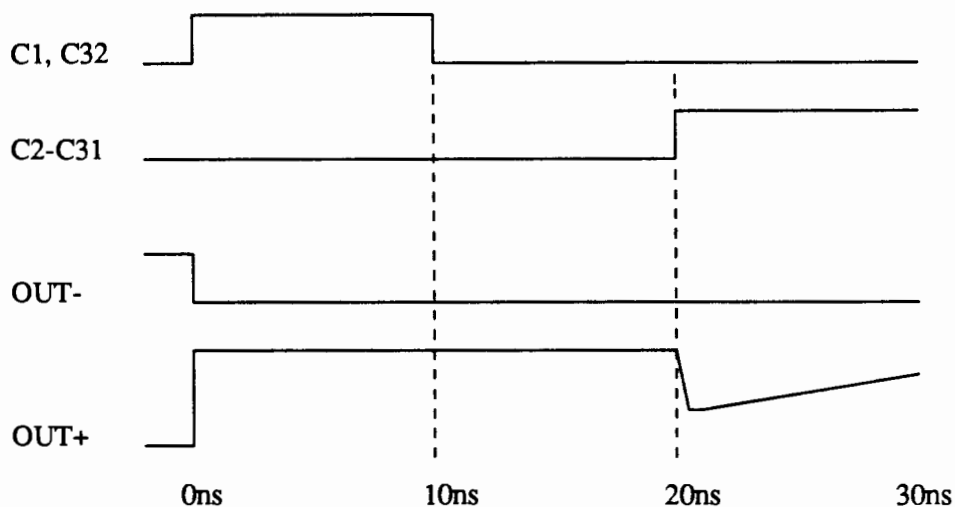


Figure 3-8. Timing diagram for charge sharing simulation.

other path to discharge the output voltage. At 20ns transistors M_2 through M_{32} are turned on and the output voltage of $OUT+$ suddenly drops from 4 volts to 2.4 volts and 10ns later its voltage reaches to 3.0 volts, shown in TABLE VI, see Figure 3-9. The problem with pass transistor logic is that when the transistors are turned on, the charge at the output node will be distributed through the fabric, and therefore lose or delay the charge at the output nodes, $OUT+$ and $OUT-$. Thus pass transistors have the potential of a charge sharing problem as the charge moves through the pass transistors into the fabric. Since we are looking at the differential comparison at the output node, we can not accept

an unknown source of noise at the output node. However this problem is eliminated by precharging the fabric to a known voltage level.

TABLE VI
CHARGE SHARING EFFECT WHEN
SWITCHING FROM ONE ROW TO ANOTHER
(TOTAL OF 32 ROWS)

TIME (ns)	OUT- (volt)	OUT+ (volt)
0	4.144	4.156
10	2.922e-02	4.152
20	-2.027e-06	4.149
21	1.533	2.410
22	8.814e-01	2.486
23	4.971e-01	2.592
24	2.779e-01	2.681
25	1.491e-01	2.757
26	7.755e-02	2.824
27	3.946e-02	2.881
28	1.974e-02	2.932
29	1.022e-02	2.976
30	5.175e-03	3.015

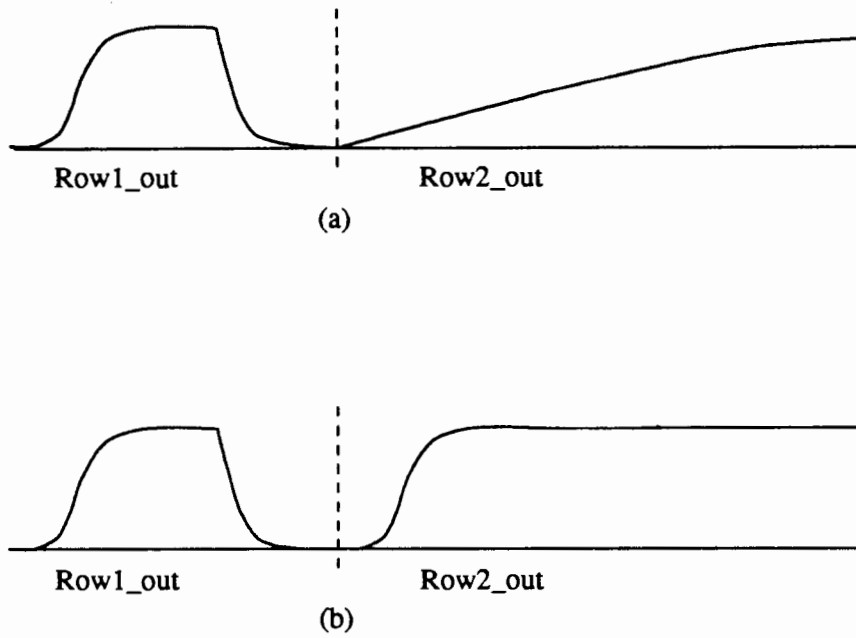


Figure 3-9. Charge sharing effect when switching from one row to another row.
(a) switching from one row to another without precharging.
(b) switching from one row to another with precharging.

CHAPTER IV

GENERATING CIRCUIT FOR BOOLEAN FUNCTION USING MAGIC FILES

CFL is a C-based program which is intended to facilitate the construction of VLSI circuit layouts. For generating circuit layout, set of data type called SYMBOL, a set of primitive operands of this type, should exist in a library of the form of Magic file.

SYMBOLS are small set of geometric primitives which may combined to make objects and saved as a new SYMBOL. There is also a set of operators which calls and generates new SYMBOL by combining existing SYMBOLS. Routing facilities are provided by CFL to generate variety of planar and non-planar wiring patterns which can be used to connect functional blocks.

CFL operator take the descriptions of the border of the symbols, and does not require information of the symbol itself. The information in the border descriptions includes the bounding box, its border and lists of coordinates of the points where each of kind of material in the symbol makes contact with the bounding box.

I have developed a C-base program called BFG which generates circuit layouts for Boolean functions. This program, that uses CFL operator can generate up to 2^{15} min-terms. A large circuit will be broken into smaller sub-circuit so that the time required for precharging the fabric (interior nodes of the product terms) will be minimized. One can use a set of variables $\{0, 1, x, \bar{x}\}$, where x can take any logical value and \bar{x} is complement of x . For specifying \bar{x} in the "input" file the program will generate \bar{x} for the circuit. The product terms can take any value of $\{0, 1, -\}$, where '-' is a don't care. Passing a value with up to four control signals will generate one block. For number of control signal

between six and ten then the circuit will be broken into two sub-blocks each with five or less control signals. A control signal between eleven and fifteen will be broken into three sub-blocks. The output of the first block will be fed into the input of the second block and output of the second sub-block will be fed into the input of the third sub-block.

The BFG program, `bfg.c`, reads from an input file called "input", see Appendix C. The first line of input file must contains `'c'` and an integer value to indicate the total number of control line in the circuit. The program will break this control signal into smaller number of controls (based on the simulation result) and will generate the sub-blocks if the control signal is larger than five. The second line must contain a `'p'` card and an integer value to indicate the total number of pass-variables for the first sub-block. After the second line the vectors for control and pass should be indicated with a blank space between them. Next line after the last vector line is another `'p'` card followed by an integer value indicating number of passes for the second block which is again followed by a set of vectors for the block. Similar procedure follows for the third sub-block. All the passes, controls and output nodes are labeled for easier access.

Example 1: An example of the format of the input file for the BFG program.

```
.c 8      /* Total number of controls. */  
.p 4      /* Total number of pass-variable for the first  
          sub-block. */  
01-1 0    /* Control-variables for one row and its pass-variable. */  
1-10 1  
1100 x  
--11 xn  
.p 3      /* Total number of pass-variable for the second  
          sub-block. */  
1011 f  
0110 fn  
-010 fn
```

The CFL macros that are used for generating the block are the following:

`rr(s1, s2)` - align right to right

`rrdx(s1, s2, dx)` - align right to right, x offset

`ll` - align left to left

`bb` - align bottom to bottom

`bbdx(s1, s2, dx)` - align bottom to bottom, x offset

`bbdxy(s1, s2, dx, dy)` - align bottom to bottom, x offset

`my(s)` - mirror in y

`cp(s1, p1)` - center to point

`ps(name, s)` - put symbol in the symbol table

CHAPTER V

CONCLUSION

Differential pass transistor logic is used for the implementation of Boolean functions. The most effective approach in design of Boolean functions generator has been found using nMOS pass transistor logic with the control signal bootstrapped above V_{dd} level. Detailed analysis of nMOS circuits, using SPICE circuit simulator and Fastsim logic simulator, is done. SPICE simulation, using level 2 model, shows that we can achieve the most reasonable time for precharging through the fabric ($\frac{\Delta v}{\Delta t}$) by breaking the number of serially connected pass transistor logic into maximum of five in a row. The study shows, by using bootstrap circuit at the gate of these pass transistor logic we can obtain a faster precharge cycle. The bootstrap circuit used in the circuit was simulated on TSPICE level 2 model simulation. The faster precharge was achieved at the boot node. The study shows, by using a transistor in a diode configuration at the boot node, one can raise the voltage at the boot node to a maximum of $V_{dd}+V_t$ level which is less than the break down voltage in this technology. By using a bootstrap circuit at the gate of the control signal, the precharge time is reduced to 18ns.

The circuit implementation of Boolean functions is accomplished. The C-code in combination with the CFL program is used to generate the layout of the circuit. The C-code will read from the library 'mag', MAGIC files, and generate the layout of the circuit for the Boolean functions. The generated circuit is tested using Fastsim logic simulator.

REFERENCES

- [01] Damu Radhakrishnan, Sterling R. Whitaker and Gary K. Maki, "Formal Design Procedures for Pass Transistor Switching Circuits" IEEE Journal of solid-state circuits, VOL. SC-20, No.2, april 1985.
- [02] Jim Kimball, "TSPICE", EE CAX Group, Tektronix, Inc., 1987.
- [03] Steve Sullivan, Tim Sauerwein, "Fastsim, A Digital Circuit Simulator with Mixed-Mode Capabilities", Tektronix, Inc., September 1988.
- [04] Alan B. Marcovitz and James H. Pugsley, "An Introduction of Switching System Design", John-Wiley Publishing Company, 1971.
- [05] Saburo Muroga, "Logic Design and Switching Theory", John-Wiley Publishing Company, 1974.
- [06] John H. Pasternak, Alex S. Shubat, C. Andret and T. Salame, "CMOS Differential Pass-Transistor Logic Design", IEEE Journal of solid-state circuits, Vol.sc-22, no.2, April 1987.
- [07] Noel M. Morris, "Digital Electronic Circuits and Systems", Macmillan Press LTD Publishing Company, 1974.
- [08] David Green, "Modern Logic Design", Addison-Wesley Publishing Company, 1986.
- [09] E. A. Parr, "The Logic Designer's Guidebook", R. R. Donnelley and Sons Publishing Company, 1984.
- [10] Marco Annaratone, "Digital CMOS Circuit Design", Kluwer Academic Publishing Company, 1985.

- [11] Lance A. Glasser and Daniel W. Dobberpuhl, "The Design and Analysis of VLSI circuit", Addison-Wesley Publishing Company, 1985.
- [12] Neil West and Kamran Eshraghian, "Principles of CMOS VLSI Design A system perspective", Addison-Wesley Publishing Company, 1985.
- [13] Thomas E. Dillinger, "VLSI Engineering", Prentice Hall Publishing Company, 1988.
- [14] A. S. Shubat, J. A. Pretorius, C. A. T. Salama, "Differential Pass Transistor Logic in CMOS Technology", Electronic letters, Vol.22, no.6, March 1986.
- [15] L. G. Heller and J. W. Davis, "Cascode Voltage Switch Logic," in ISSCC Dig. Tech. Papers, pp.16-17, 1984.

APPENDIX A

TABLES OF COMPARISONS

**PRECHARGING THROUGH THE FABRIC USING
nMOS TRANSISTOR ($W=5.5\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.440	0.000	0.000	0.000	0.000	0.000
3	2.742	0.000	0.000	0.000	0.000	3.488
6	2.294	0.3586	0.2569	0.000	0.000	3.563
9	1.214	1.214	1.214	0.000	0.000	3.594
12	1.124	1.129	1.161	1.191	1.199	3.616
15	1.250	1.254	1.281	1.337	1.445	3.434
18	2.198	2.209	2.276	2.405	2.613	2.953
21	2.678	2.685	2.730	2.817	2.957	3.186
24	2.920	2.925	2.959	3.024	3.128	3.300
27	3.063	3.068	3.094	3.146	3.229	3.367
30	3.159	3.162	3.184	3.227	3.296	3.412
33	3.227	3.230	3.249	3.285	3.345	3.445
36	3.278	3.280	3.297	3.329	3.381	3.469
39	3.317	3.319	3.334	3.362	3.409	3.488
42	3.349	3.351	3.364	3.389	3.431	3.503
50	3.408	3.410	3.420	3.440	3.473	3.531

**PRECHARGING THROUGH THE FABRIC USING
nMOS TRANSISTOR ($W=11\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.439	0.000	0.000	0.000	0.000	0.8681
6	2.268	0.3618	0.2135	0.000	0.000	3.603
9	1.217	1.217	1.217	0.000	0.000	3.634
12	1.124	1.129	1.161	1.191	1.199	3.657
15	1.252	1.256	1.283	1.340	1.449	3.497
18	2.251	2.263	2.338	2.483	2.722	3.152
21	2.746	2.754	2.802	2.895	3.048	3.326
24	2.981	2.986	3.021	3.090	3.202	3.407
27	3.117	3.121	3.149	3.202	3.291	3.455
30	3.206	3.209	3.232	3.276	3.349	3.486
33	3.268	3.271	3.290	3.328	3.390	3.508
36	3.314	3.317	3.333	3.366	3.421	3.525
39	3.350	3.352	3.367	3.396	3.444	3.537
40	3.360	3.362	3.376	3.404	3.451	3.541
50	3.432	3.433	3.443	3.464	3.499	3.571

**PRECHARGING THROUGH THE FABRIC USING
nMOS TRANSISTOR ($W=22\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.442	0.000	0.000	0.000	0.000	1.188
3	2.743	0.000	0.000	0.000	0.000	3.581
6	2.420	0.3706	0.2635	0.000	0.000	3.635
9	1.228	1.228	1.228	0.000	0.000	3.667
12	1.126	1.133	1.166	1.196	1.196	3.691
15	1.262	1.266	1.294	1.353	1.465	3.557
18	2.294	2.307	2.384	2.537	2.791	3.296
21	2.787	2.795	2.844	2.940	3.100	3.420
24	3.014	3.020	3.055	3.125	3.241	3.478
27	3.144	3.148	3.176	3.231	3.322	3.511
30	3.228	3.232	3.255	3.300	3.375	3.533
33	3.288	3.291	3.310	3.348	3.412	3.548
36	3.331	3.334	3.351	3.384	3.440	3.560
39	3.365	3.367	3.382	3.412	3.462	3.575
42	3.393	3.394	3.407	3.434	3.480	3.589
50	3.443	3.445	3.455	3.476	3.512	3.623

**PRECHARGING THROUGH THE FABRIC USING
pMOS TRANSISTOR ($W=5.5\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.452	0.000	0.000	0.000	0.000	0.000
3	2.742	0.000	0.000	0.000	0.000	5.003
6	2.340	0.3598	0.2606	0.000	0.000	5.000
9	1.219	1.218	1.219	0.000	0.000	5.000
12	1.127	1.132	1.164	1.195	1.202	5.000
15	1.256	1.260	1.287	1.346	1.457	4.878
18	2.319	2.333	2.415	2.577	2.854	4.798
21	2.826	2.834	2.886	2.987	3.162	4.927
24	3.050	3.056	3.094	3.167	3.294	4.962
27	3.177	3.182	3.211	3.268	3.369	4.977
30	3.259	3.262	3.286	3.333	3.417	4.985
33	3.315	3.318	3.338	3.379	3.450	4.989
36	3.357	3.359	3.377	3.412	3.475	4.992
39	3.389	3.391	3.407	3.438	3.494	4.994
42	3.415	3.416	3.430	3.458	3.508	4.995
50	3.462	3.463	3.474	3.496	3.536	4.997

**PRECHARGING THROUGH THE FABRIC USING
pMOS TRANSISTOR ($W=11\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.452	0.000	0.000	0.000	0.000	0.000
3	2.743	0.000	0.000	0.000	0.000	0.000
6	2.161	0.3587	0.2452	0.6907	0.000	5.000
9	1.191	1.191	1.191	0.000	0.000	5.000
12	1.111	1.116	1.148	1.181	1.176	5.000
15	1.238	1.242	1.269	1.328	1.439	4.927
18	2.313	2.327	2.410	2.573	2.852	4.900
21	2.822	2.830	2.882	2.984	3.160	4.963
24	3.049	3.055	3.093	3.166	3.293	4.981
27	3.177	3.182	3.211	3.268	3.369	4.988
30	3.259	3.263	3.286	3.334	3.417	4.992
33	3.316	3.319	3.339	3.379	3.450	4.995
36	3.357	3.360	3.377	3.412	3.475	4.996
39	3.389	3.391	3.407	3.438	3.494	4.997
40	3.398	3.400	3.415	3.445	3.499	4.997
50	3.462	3.463	3.474	3.496	3.536	4.998

**PRECHARGING THROUGH THE FABRIC USING
pMOS TRANSISTOR ($W=22\mu$)**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.452	0.000	0.000	0.000	0.000	0.000
3	2.742	0.000	0.000	0.000	0.000	5.000
6	2.341	0.3551	0.2403	0.000	0.000	5.000
9	1.201	1.202	1.202	0.000	0.000	5.000
12	1.122	1.126	1.161	1.193	1.194	5.000
15	1.252	1.256	1.283	1.341	1.452	4.960
18	2.321	2.334	2.417	2.579	2.857	4.949
21	2.828	2.836	2.888	2.989	3.164	4.981
24	3.052	3.058	3.095	3.168	3.295	4.990
27	3.178	3.183	3.212	3.269	3.369	4.994
30	3.259	3.263	3.287	3.334	3.417	4.996
33	3.316	3.319	3.339	3.379	3.450	4.997
36	3.357	3.360	3.377	3.413	3.475	4.998
39	3.389	3.391	3.407	3.438	3.494	4.998
42	3.415	3.417	3.430	3.458	3.509	4.999
50	3.462	3.463	3.474	3.496	3.536	4.999

**PRECHARGING THROUGH THE FABRIC USING
BOOTSTRAP CIRCUIT AT THE GATE OF
PASS DEVICE**

TIME (ns)	P0 (volt)	P1 (volt)	P2 (volt)	P3 (volt)	P4 (volt)	P5 (volt)
0	2.364	0.000	0.000	0.000	0.000	0.000
2	2.738	0.000	0.000	0.000	0.000	4.991
4	2.746	0.000	0.000	0.000	0.000	5.000
6	1.710	0.4811	0.3524	0.000	0.000	5.000
8	1.463	1.447	1.428	0.000	0.000	5.000
10	1.407	1.403	1.377	0.4933	0.3579	5.000
12	1.362	1.368	1.398	1.429	1.435	5.000
14	1.418	1.419	1.421	1.426	1.433	5.000
16	2.002	2.020	2.136	2.380	2.838	4.333
18	3.036	3.051	3.142	3.322	3.626	4.749
20	3.496	3.506	3.570	3.696	3.910	4.884
30	4.123	4.126	4.151	4.199	4.283	4.983

**CHARGE SHARING EFFECT:
SWITCHING FROM ONE ROW TO ANOTHER
(TOTAL OF 16 ROWS)**

TIME (ns)	OUT- (volt)	OUT+ (volt)
0	4.291	4.312
10	2.718e-02	4.322
20	-3.707e-06	4.319
21	1.187	2.515
22	3.728e-01	2.665
23	1.077e-01	2.816
24	2.289e-02	2.931
25	5.105e-03	3.017
26	8.047e-04	3.087
27	5.914e-05	3.143
28	9.508e-05	3.189
29	-5.279e-05	3.226
30	7.742e-05	3.259

**CHARGE SHARING EFFECT:
SWITCHING FROM ONE ROW TO ANOTHER
(TOTAL OF 32 ROWS)**

TIME (ns)	OUT- (volt)	OUT+ (volt)
0	4.144	4.156
10	2.922e-02	4.152
20	-2.027e-06	4.149
21	1.533	2.410
22	8.814e-01	2.486
23	4.971e-01	2.592
24	2.779e-01	2.681
25	1.491e-01	2.757
26	7.755e-02	2.824
27	3.946e-02	2.881
28	1.974e-02	2.932
29	1.022e-02	2.976
30	5.175e-03	3.015

BOOTSTRAPPING GATES C and \bar{C}
(MOS CAPACITOR BEING SAME SIZE AS THE LOAD)

Time (ns)	PCHARGE (volt)	CLK2 (volt)	CLK1 (volt)	C (volt)	\bar{C} (volt)
0	5	5	0	3.6692	3.6692
5	5	5	0	3.6692	3.6692
10	5	5	0	3.6692	3.6692
15	5	5	0	3.6692	3.6692
20	5	0	0	5.1374	5.4634
25	5	0	0	5.2714	5.7175
30	5	0	0	5.3109	5.7683
35	5	0	0	5.3204	5.7805
40	5	0	0	5.3226	5.7833
45	5	0	0	5.3231	5.7839
50	5	0	0	5.3232	5.7841

BOOTSTRAPPING GATES C and \bar{C}
(MOS CAPACITOR BEING TWICE LARGER THAN THE LOAD)

TIME (ns)	PCHARGE (volt)	CLK2 (volt)	CLK1 (volt)	C (volt)	\bar{C} (volt)
0	5	5	0	3.6692	3.6692
5	5	5	0	3.6692	3.6692
10	5	5	0	3.6692	3.6692
15	5	5	0	3.6692	3.6692
20	5	0	0	5.4608	5.7065
25	5	0	0	5.9481	6.2610
30	5	0	0	6.1701	6.5132
35	5	0	0	6.2979	6.6394
40	5	0	0	6.3654	6.6680
45	5	0	0	6.4017	6.6583
50	5	0	0	6.4149	6.6403

BOOTSTRAPPING GATES C and \bar{C}
(MOS CAPACITOR BEING FOUR TIMES LARGER THAN THE LOAD)

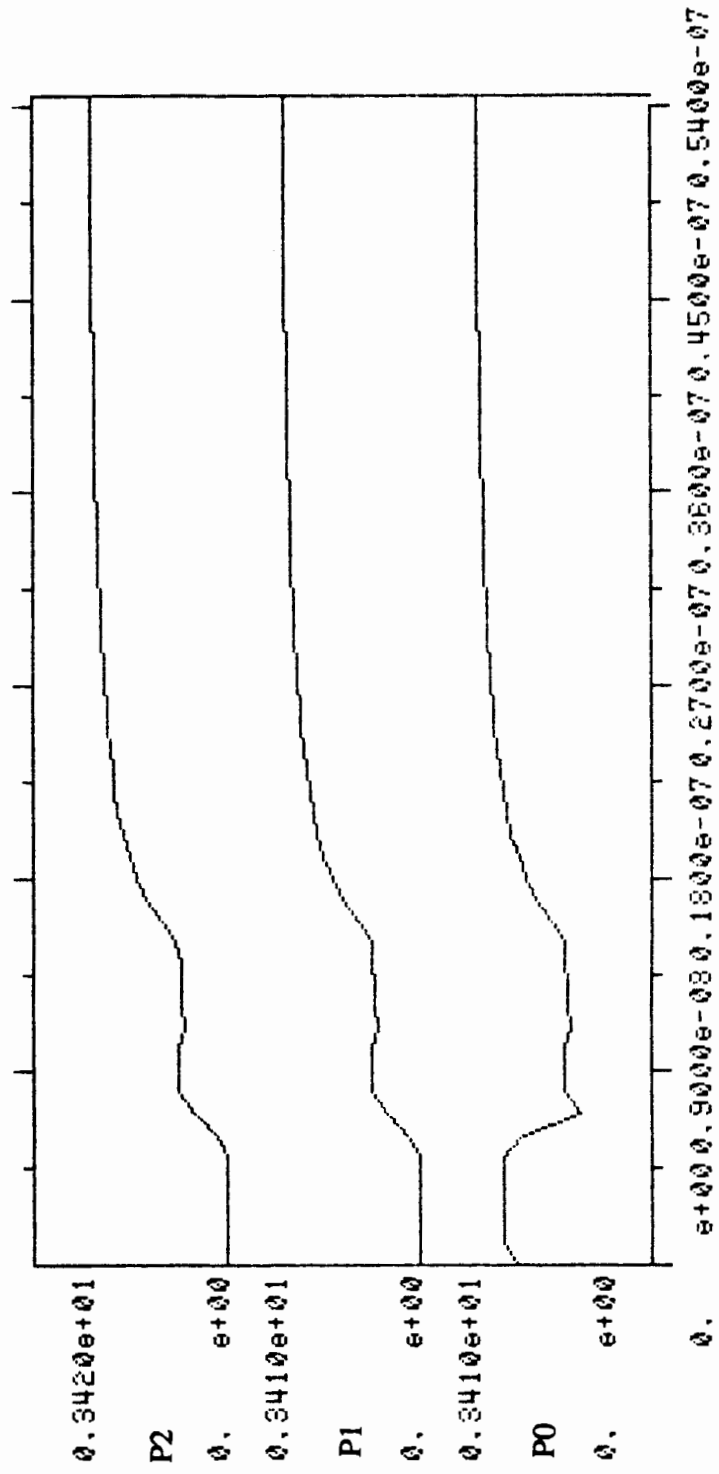
TIME (ns)	PCHARGE (volt)	CLK2 (volt)	CLK1 (volt)	C (volt)	\bar{C} (volt)
0	5	5	0	3.6692	3.6692
5	5	5	0	3.6692	3.6692
10	5	5	0	3.6692	3.6692
15	5	5	0	3.6692	3.6692
20	5	0	0	5.2180	5.3478
25	5	0	0	6.2009	6.4135
30	5	0	0	6.5021	6.7208
35	5	0	0	6.6656	6.8219
40	5	0	0	6.7400	6.8308
45	5	0	0	6.7615	6.8108
50	5	0	0	6.7590	6.7855

BOOTSTRAPPING GATES C and \bar{C}
(MOS CAPACITOR BEING EIGHT TIMES LARGER THAN THE LOAD)

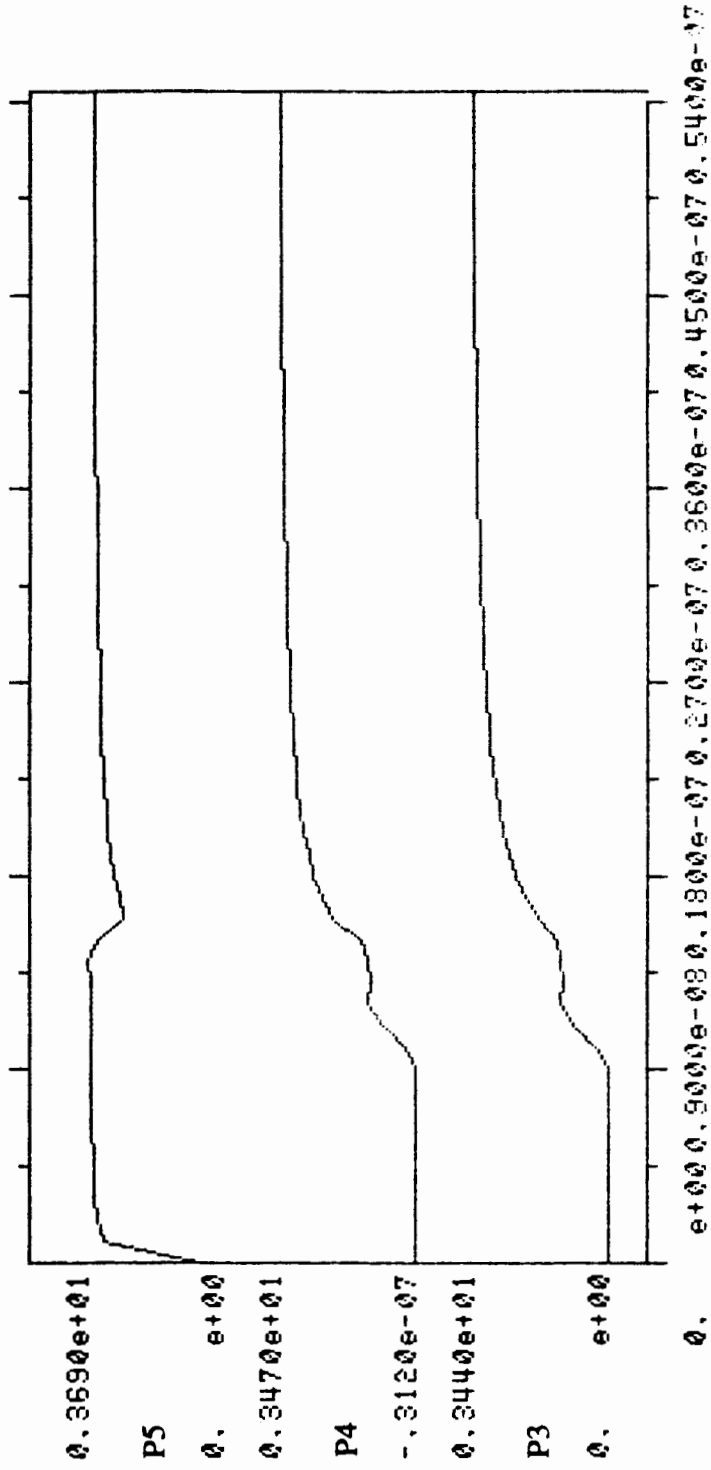
TIME (ns)	PCHARGE (volt)	CLK2 (volt)	CLK1 (volt)	C (volt)	\bar{C} (volt)
0	5	5	0	3.6692	3.6692
5	5	5	0	3.6692	3.6692
10	5	5	0	3.6692	3.6692
15	5	5	0	3.6692	3.6692
20	5	0	0	4.6767	4.7242
25	5	0	0	5.8008	5.9015
30	5	0	0	6.4392	6.5696
35	5	0	0	6.7187	6.8362
40	5	0	0	6.8205	6.9016
45	5	0	0	6.8533	6.9043
50	5	0	0	6.8590	6.8903

APPENDIX B

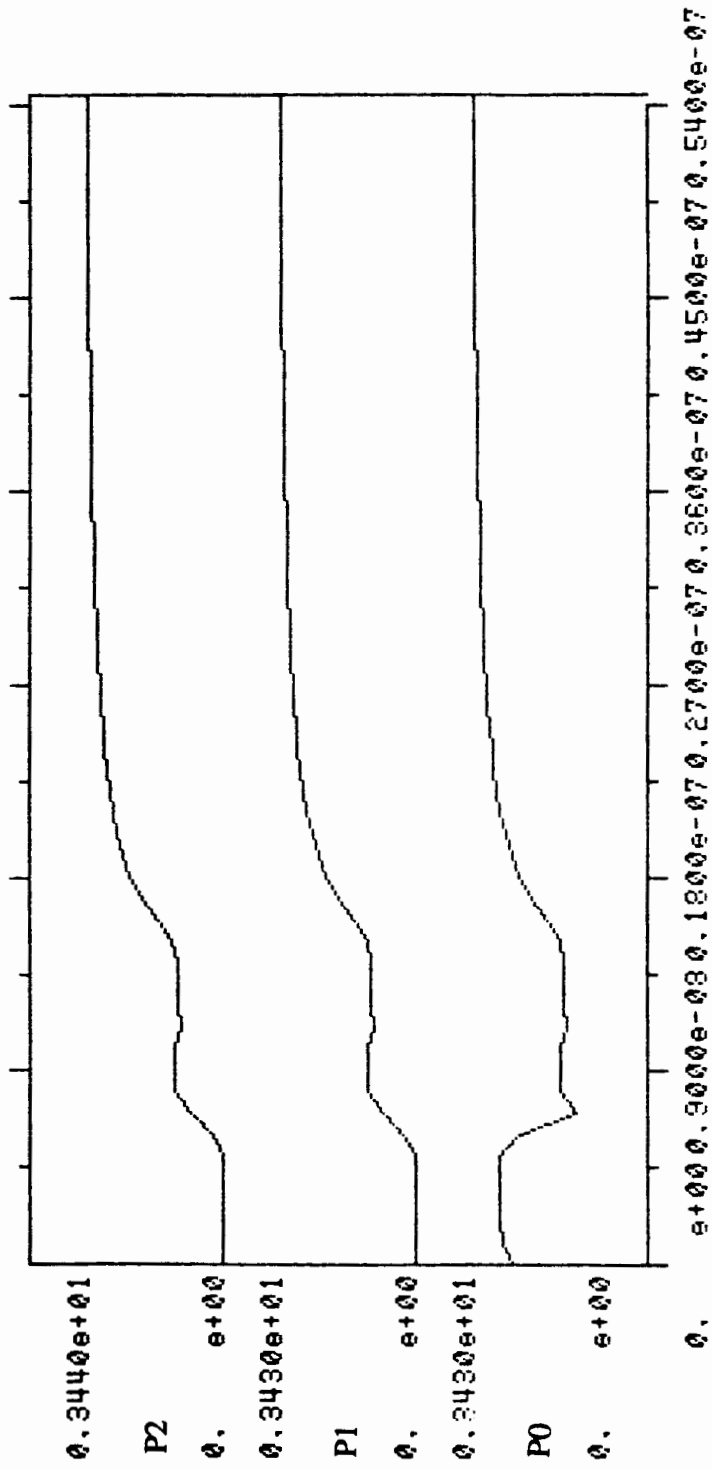
FIGURES OF COMPARISONS



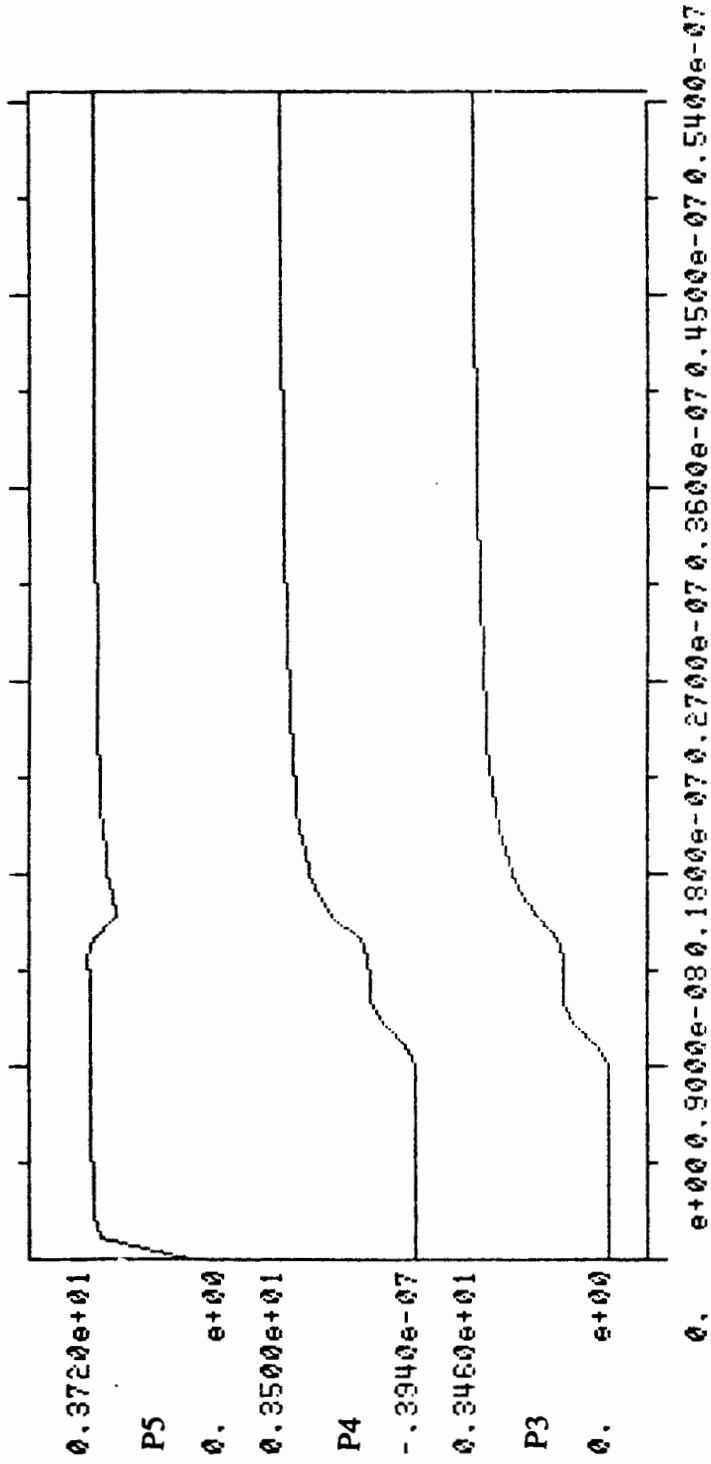
Precharging through the fabric using
nMOS transistor ($W=5.5\mu$) - Part1



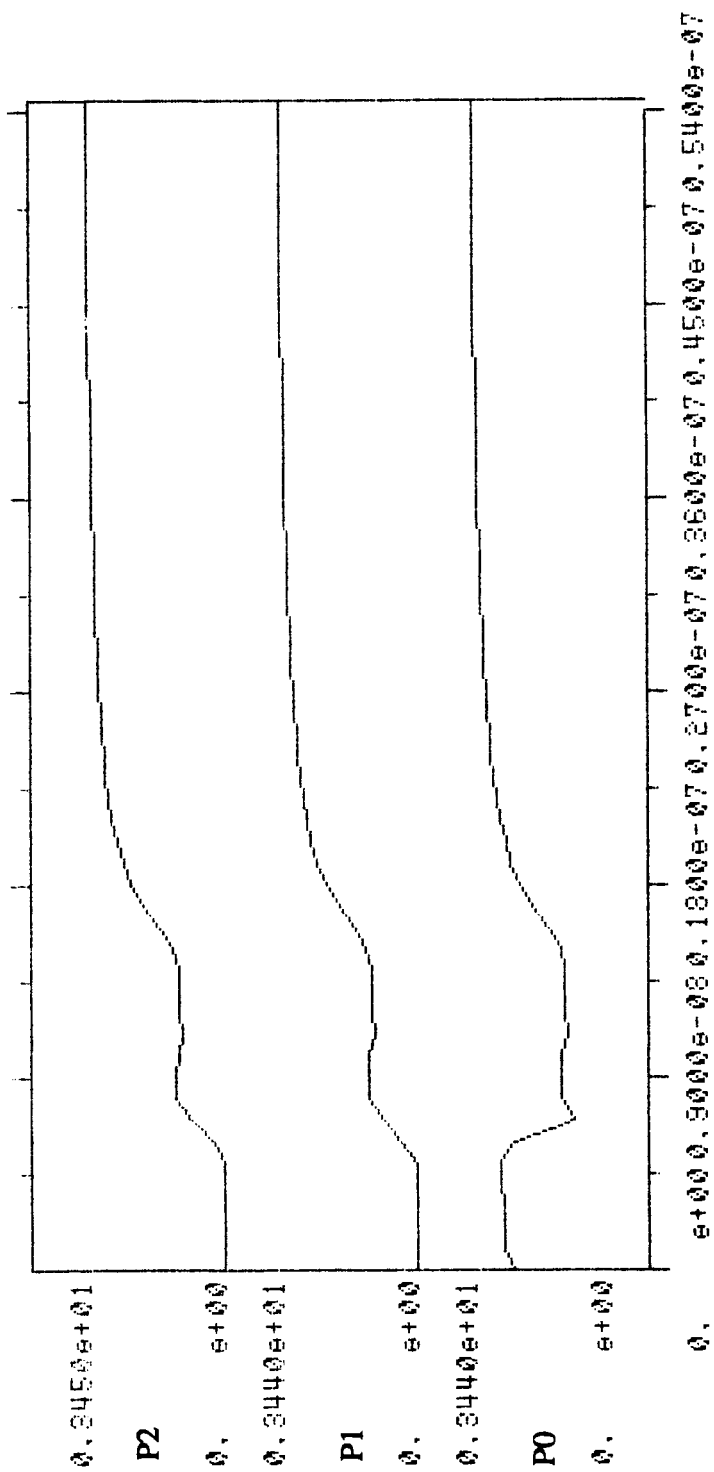
Precharging through the fabric using nMOS transistor ($W=5.5\mu$) - Part 2



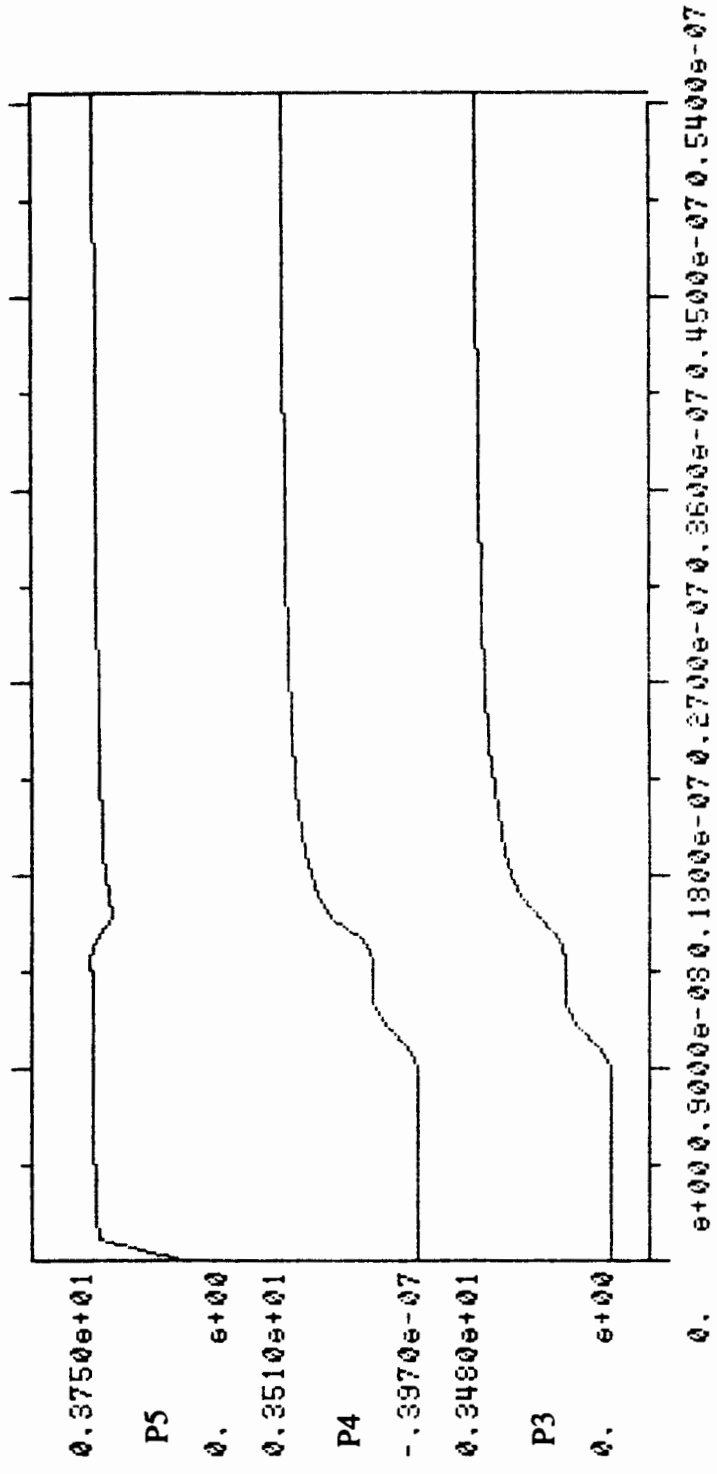
Precharging through the fabric using
nMOS transistor ($W=11\mu$) - Part I



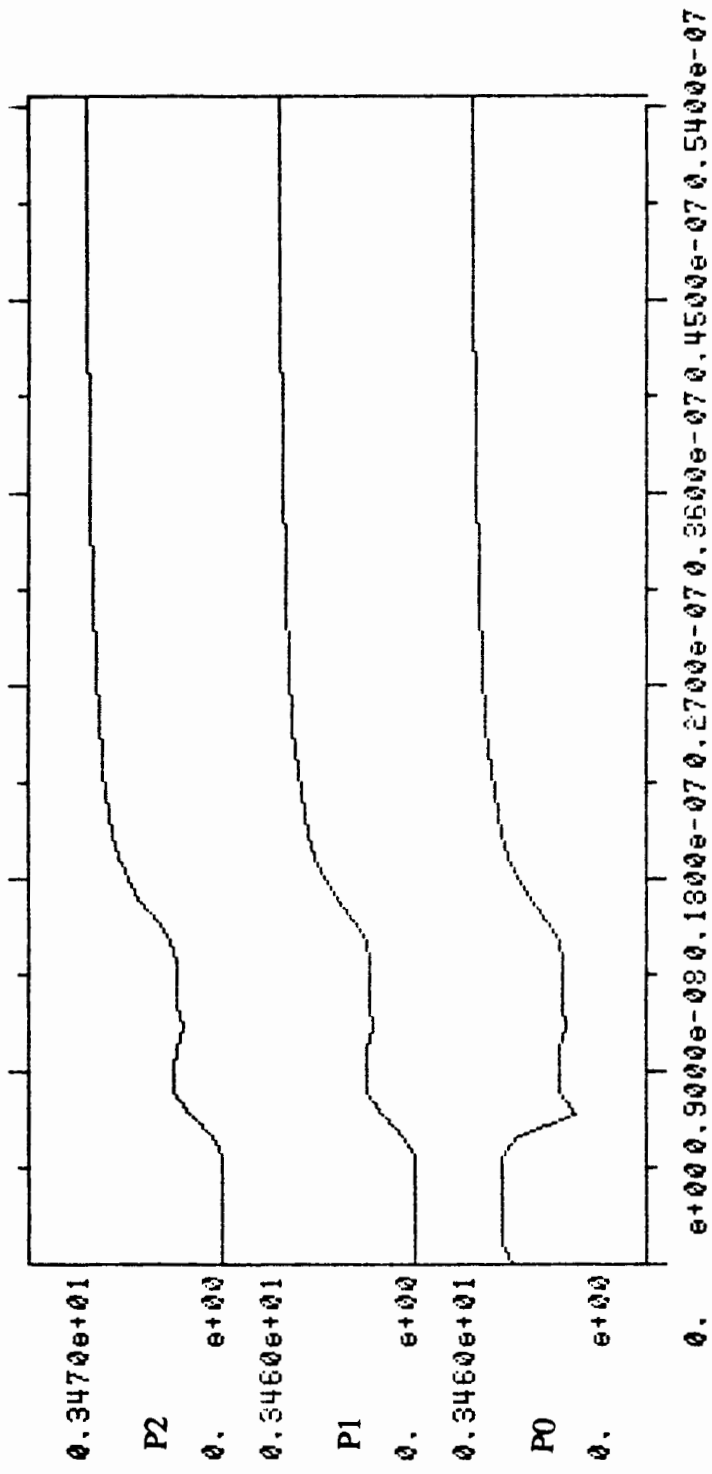
Precharging through the fabric using
nMOS transistor ($W = 1\mu$) - Part2



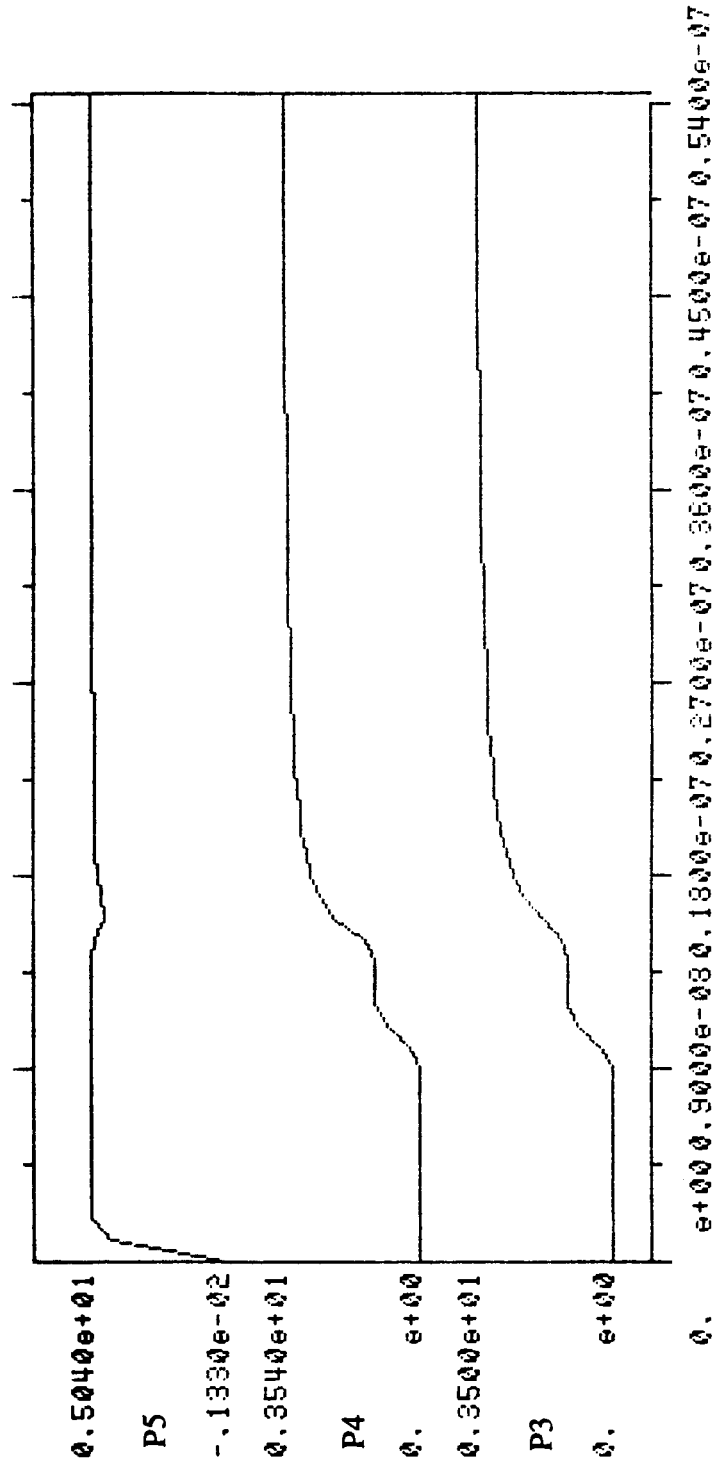
Precharging through the fabric using
 nMOS transistor ($W=2.2\mu$) - Part1



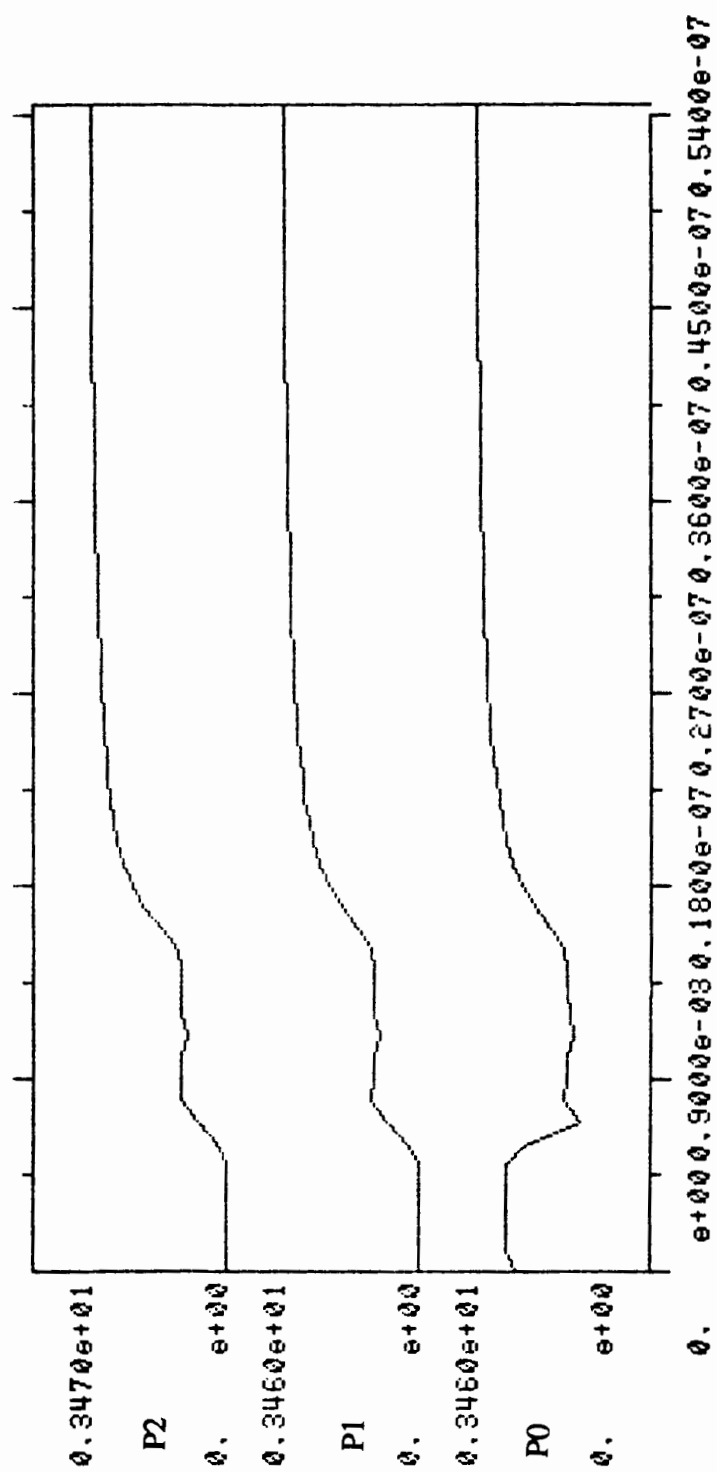
Precharging through the fabric using
nMOS transistor (W=22μ) - Part2



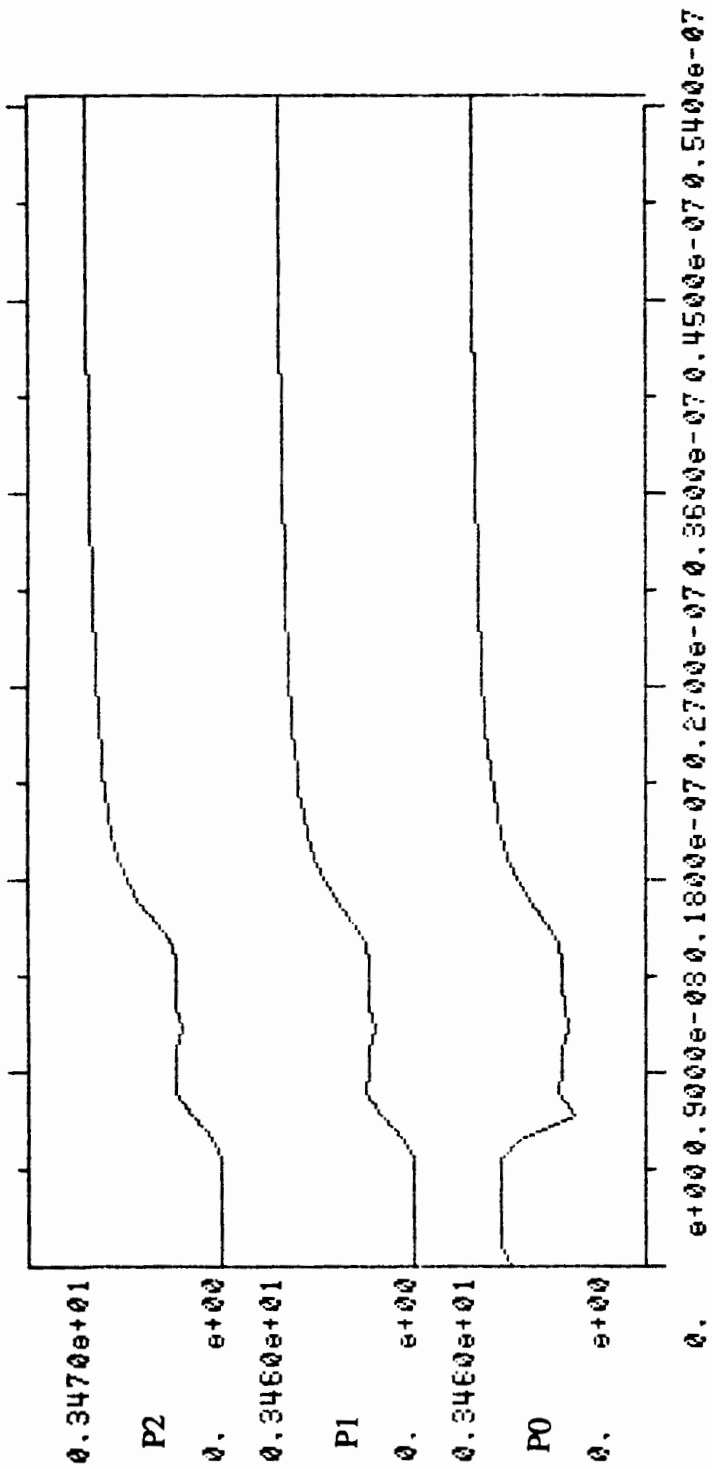
Precharging through the fabric using pMOS transistor ($W=5.5\mu$) - Part I



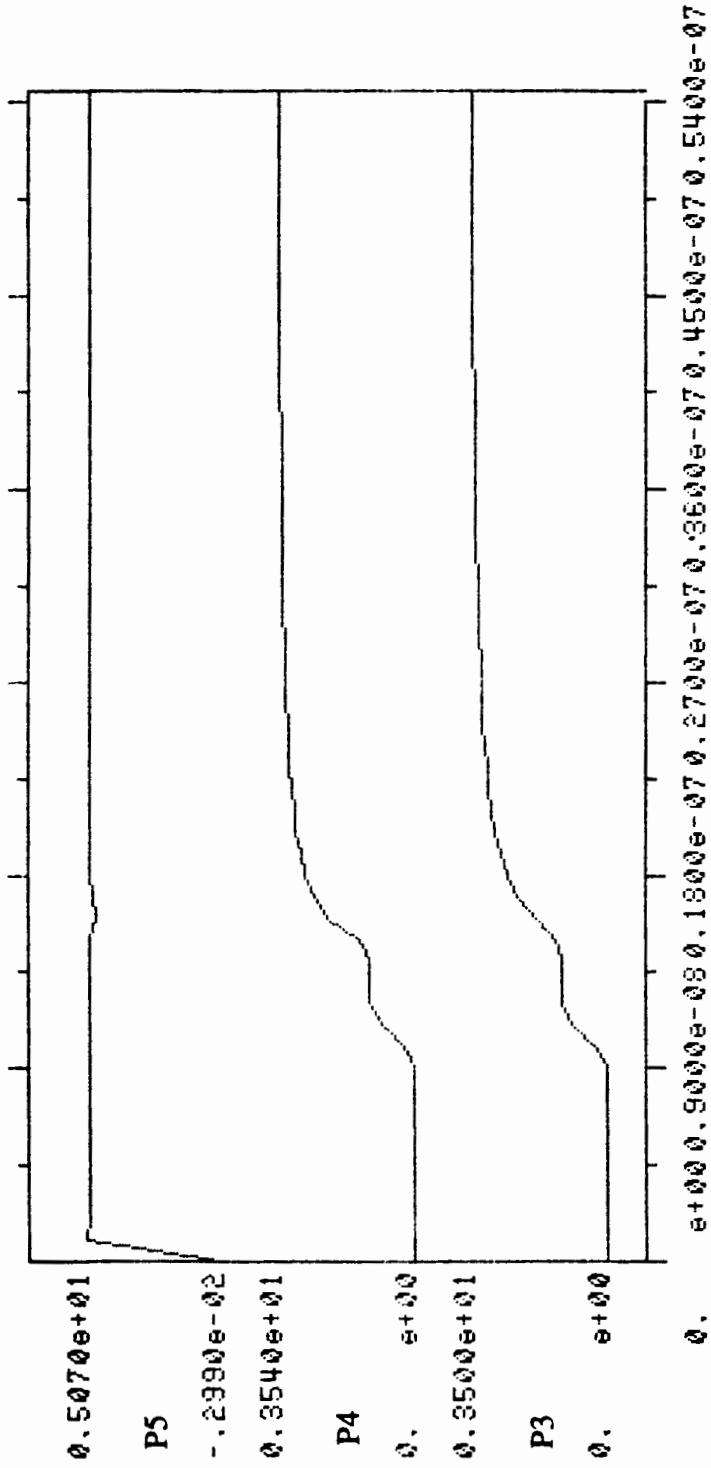
Precharging through the fabric using pMOS transistor ($W=5.5\mu$) - Part2



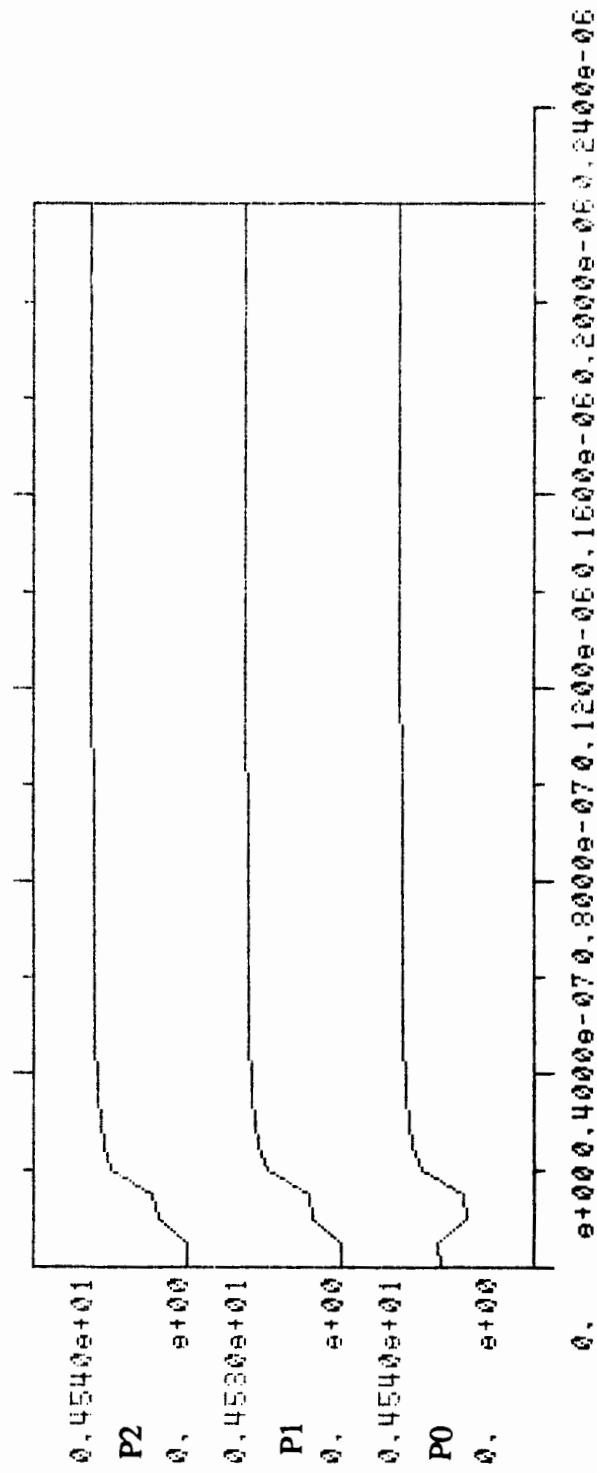
Precharging through the fabric using
pMOS transistor ($W=1\mu$) - Part1



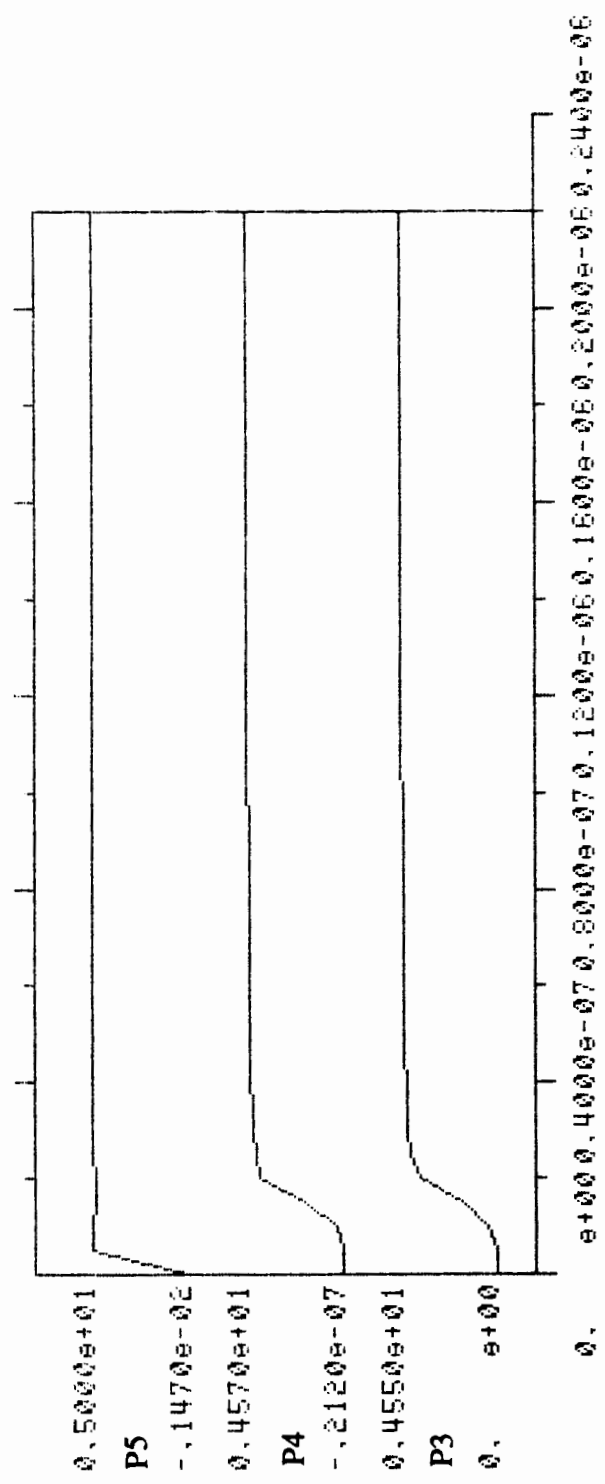
Precharging through the fabric using pMOS transistor ($W=22\mu$) - Part1



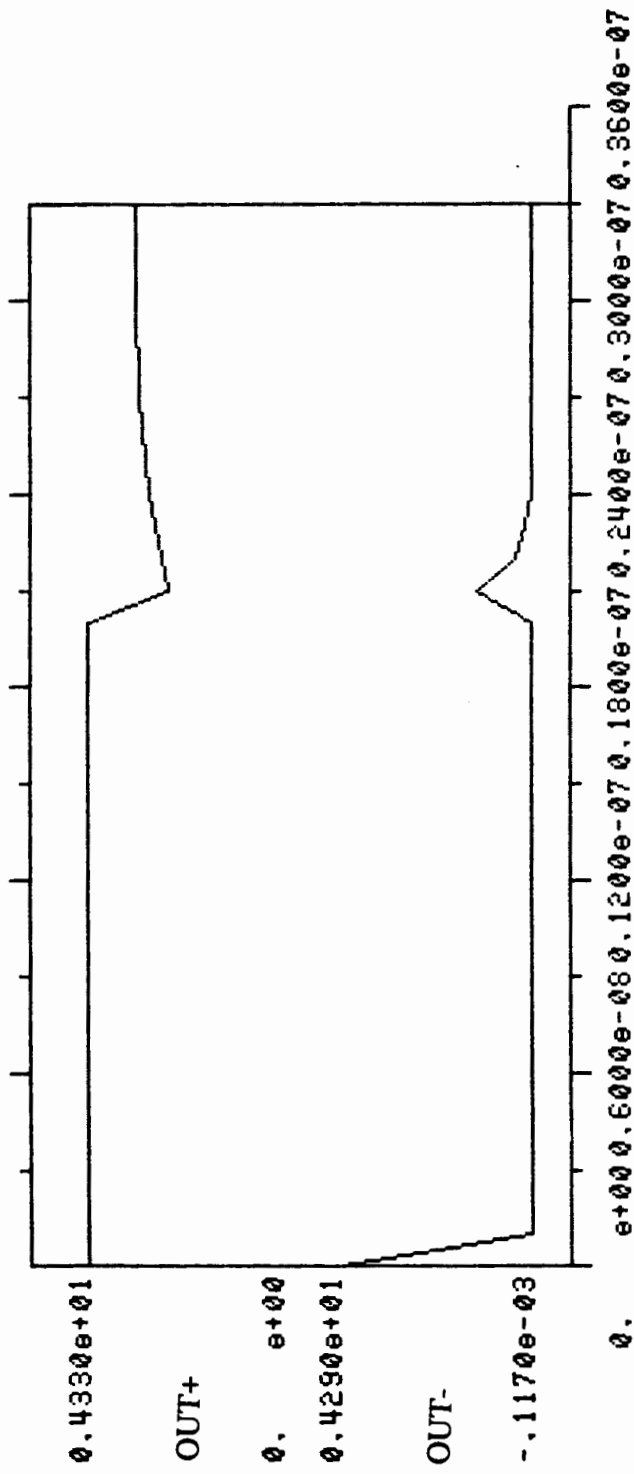
Precharging through the fabric using
pMOS transistor ($W=22\mu$) - Part2



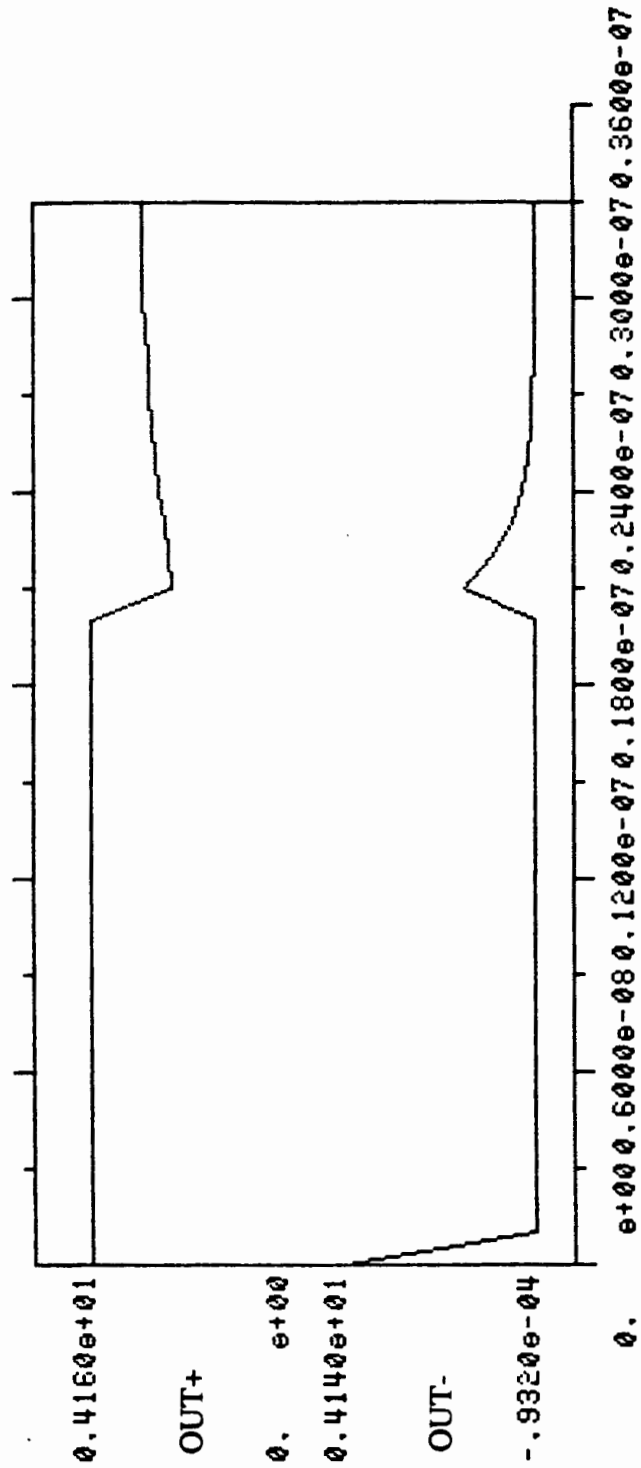
Precharging through the fabric using
bootstrap circuit at the gate of the pass device - Part1



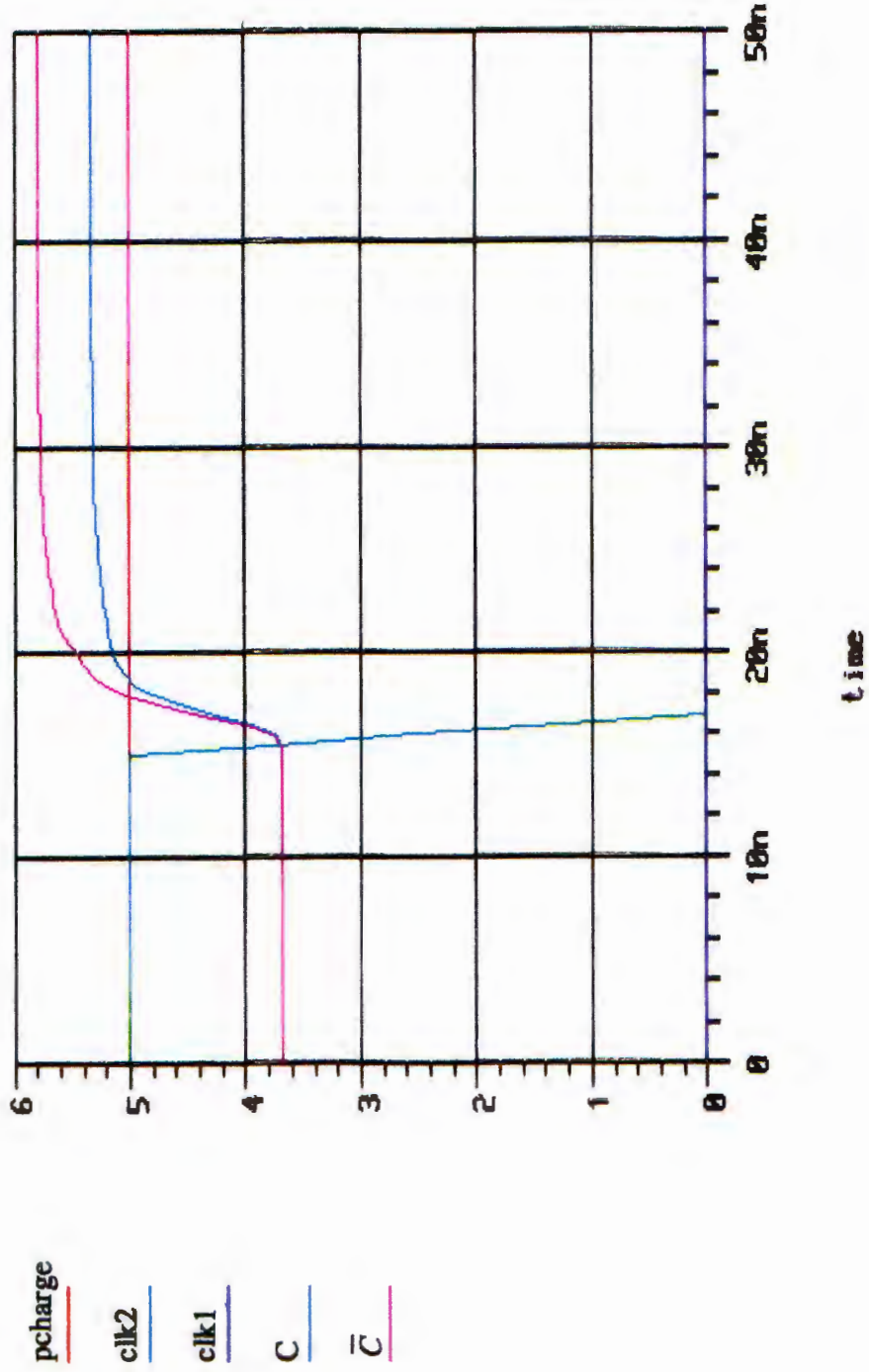
Precharging through the fabric using bootstrap circuit at the gate of the pass device - Part2



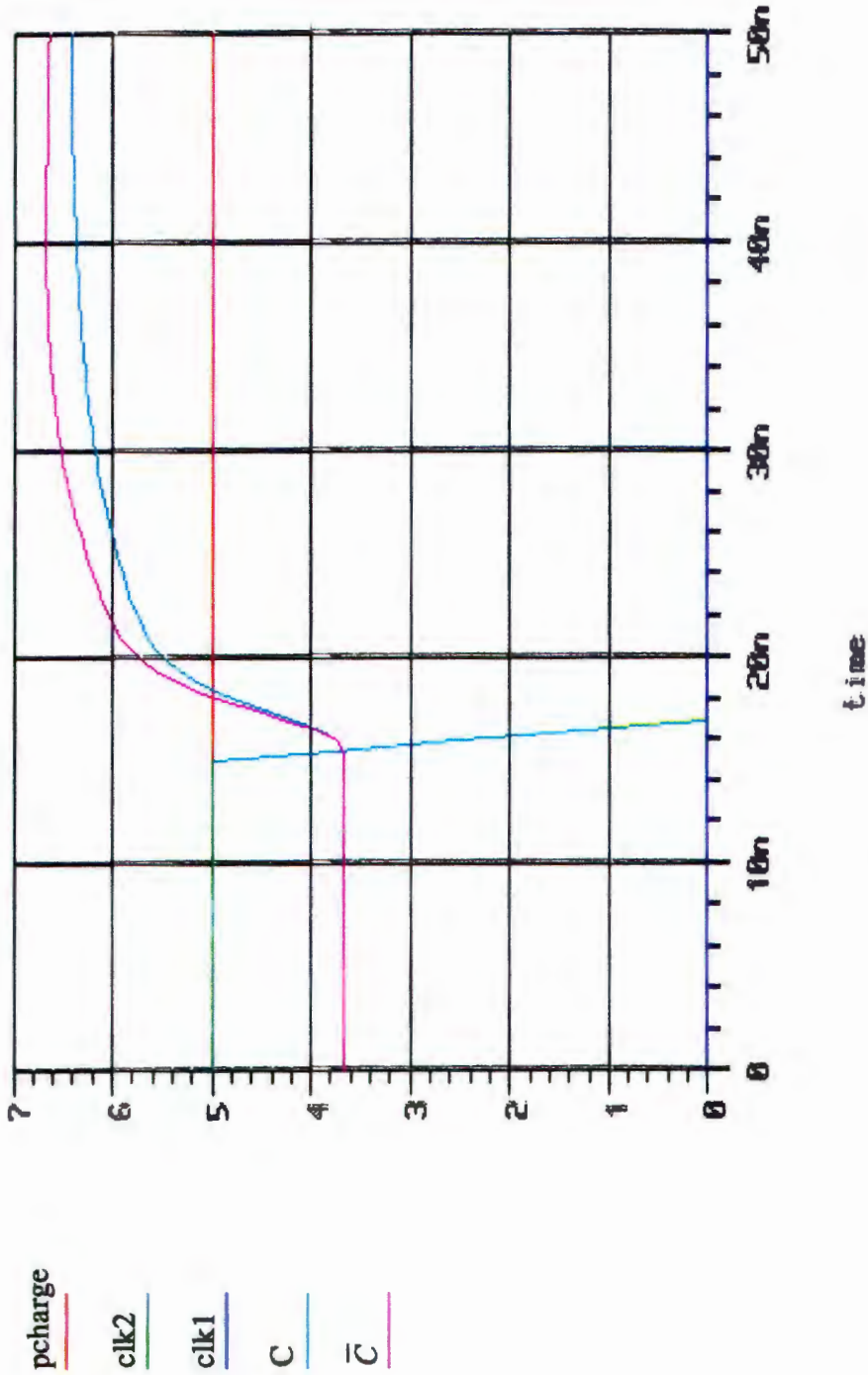
Charge sharing effect:
switching from one roe to another
(total of 16 rows)



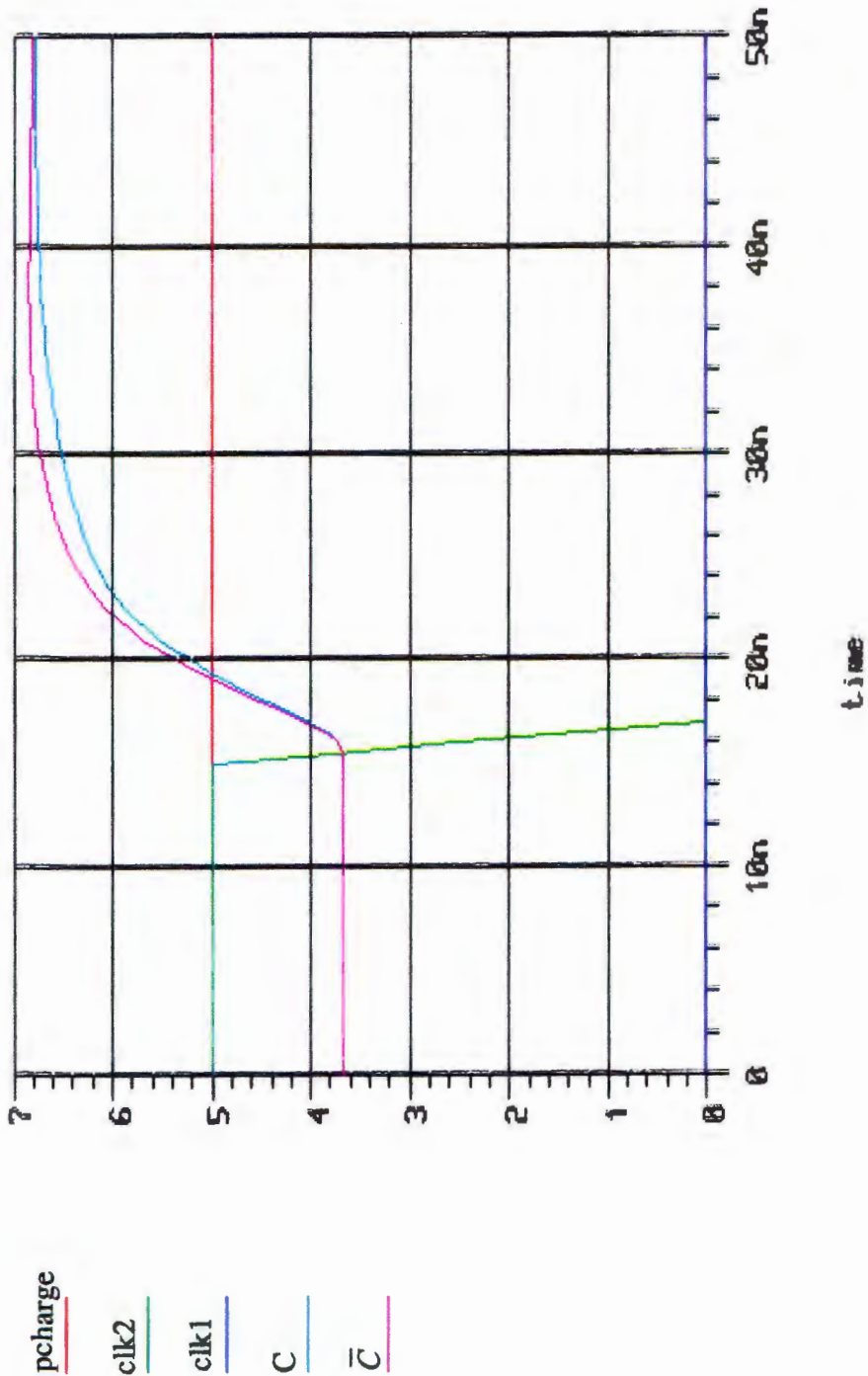
Charge sharing effect:
switching from one row to another
(total of 32 rows)



Bootstrapping gate C and C-bar
(MOS capacitor being same size as the load)



Bootstrapping gate C and \bar{C}
(MOS capacitor being twice larger than the load)

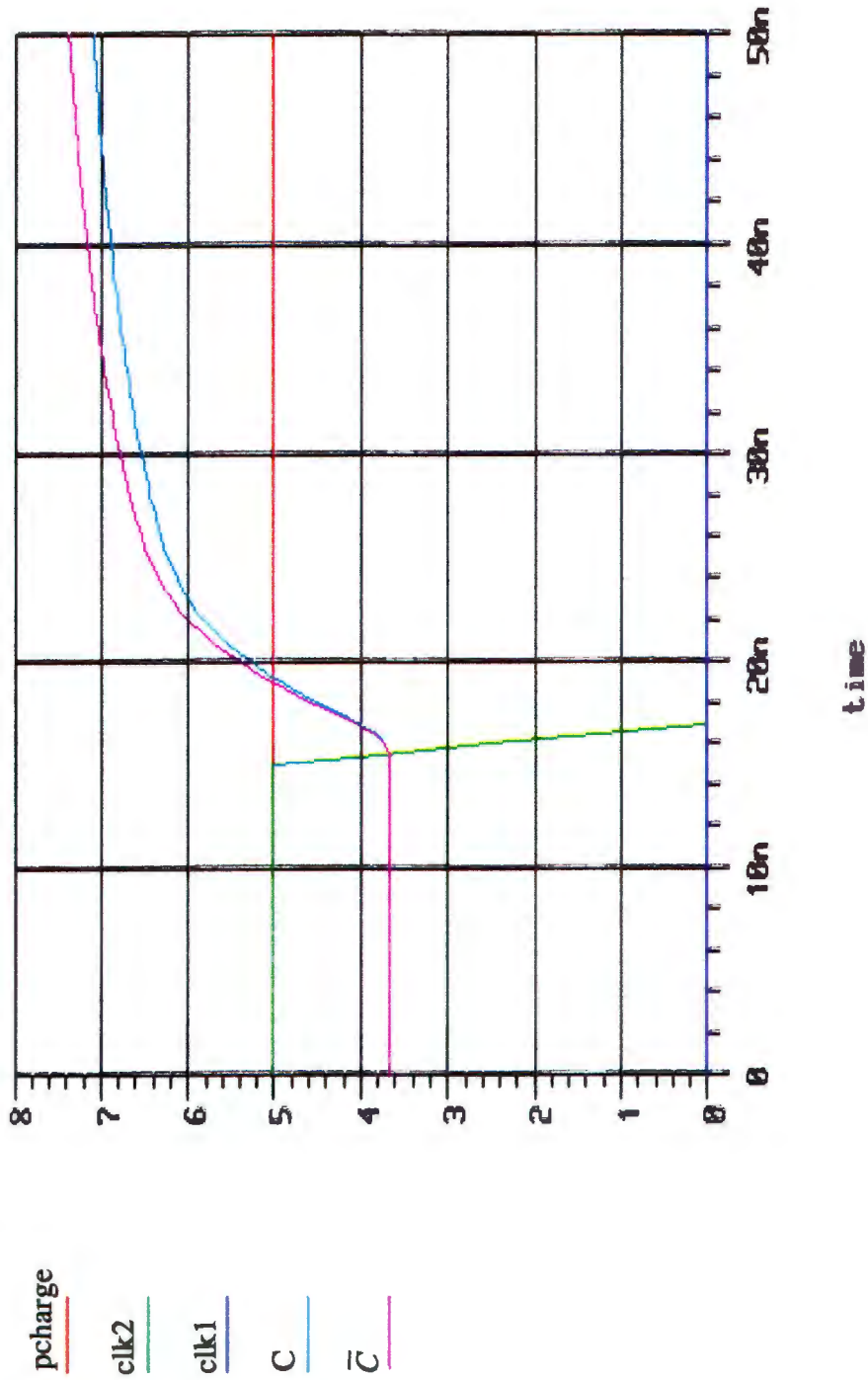


Bootstrapping gate C and C-bar
(MOS capacitor being four times larger than the load)

bootckt

06/07/88

14:36:19



Bootstrapping gate C and \bar{C}
(MOS capacitor being eight times larger than the load)

APPENDIX C

PROGRAM CODE FOR LAYOUT GENERATION OF BOOLEAN FUNCTIONS

```
#include <stdio.h>
#include "cfl.h"          /*$UW_VLSI_TOOLS/include */

/*=====*/
/* Global flags and variable
*/
int control, m, i, pass_var;
char ch[50], chc[50];
char con[100], pvar[100];
char prod[1000][100];
char labl[1000][100];
char labelstring[100];

int offset;
FILE *fp, *fopen();

/*=====*/
```

```

/*=====*/
/* Some useful strings in cfl
*/
char *top = "top";
char *bot = "bot";
char *left = "left";
char *right = "right";

/* Some useful strings in MOSIS p-well CMOS
*/
char *m1 = "metal1";
char *diff = "diffusion";
char *poly = "polysilicon";
char *m2 = "metal2";
char *cut = "cut";
char *pplus = "pplus";
char *pwell = "pwell";

SYMBOL *boot, *dcare, *boots, *p2, *pass, *pass2;
SYMBOL *passes, *passes2, *cont, *contclk, *controls;
SYMBOL *lrout, *rrtop, *rrouttop, *rroutbot, *rrout;
SYMBOL *rf, *rfn, *rrouts, *prechg, *pcharges, *btcnt;
SYMBOL *bfg, *btcntr, *btcntrp, *pbtcnt, *row, *rows;
SYMBOL *btlr, *btlrrr, *btlrrpc, *btlrrpc2;

BORDER *b0, *b1, *b2, *b3, *br1;

PT *p0, *p1;

/*=====*/

```



```

/*=====*/
main( )
{

/* set the flags and variables
*/
    int base, x, br1no, br2no, br3no, i;
    int user_input = 1;

    bfg = NULL;
    offset = 0;

/* Set technology for Magic file
*/

    cfsetc ("format", "magic");
    cfstart ("stdcmos");

/* Cells that was build for generating the block
*/

    pass = gs ("pass");
    dcare = gs ("dcare");
    pass2 = gs ("pass2");
    passes2 = gs ("passes2");
    cont = gs ("cont");
    contclk = gs ("contclk");
    boot = gs ("boot");
    lrout = gs ("lrout");
    rrtop = gs ("rrtop");
    rrout = gs ("rrout");
    rf = gs ("rf");
    rfn = gs ("rfn");
    rroutbot = gs ("rroutbot");
    rrouttop = gs ("rrouttop");
    prechg = gs ("prechg");

/*-----*
* Opening and reading form the 'input' file:
*-----*/

    fp = fopen ("input", "r");

    fscanf (fp, "%s%d", ch, &control);
        /* Reading number of controls
        form the '.c' card in the input file. */
    if (strcmp (ch, ".c") != 0)
        /* Check for '.c' card. */
    {
        printf ("'.c' is missing. 0);
    }
}

```

```

else if (( control < 2 ) || ( control > 15 ))
    /* Check for number of control lines. */
    {
        printf ("Invalid value for control 0);
            /* The range for control lines for one block
            is between 2 and 15. */
    }

/*-----*
* Control is less than or equal 5, so assemble one block:
*-----*/

else if (control < 6)
    {
        read_n (control);
            /* Read number of pass variable and check. */
        read_m (control, pass_var);
            /* Assemble the block. */
    }

/*-----*
* Control is larger than 5, so do the following:
* For number of control lines between 6 and 10 the block
* will be broken into two sub-block. For control lines
* between 11 and 15 the block will be broken into three
* sub-blocks. The output of previous sub-gate will go to
* the input of the next stage.
*-----*/

else
    {
        switch(control)
        {
            /*
            Since control = 6, this control has
            been broken up into control = 3, 3.
            */
            case 6:
                read_n (control);
                    /* Read number of pass-variable and check. */
                read_m (3, pass_var);
                    /* Assemble the first sub-block with three
                    control lines and given pass-variable. */
                printf ("0);
                read_n (control);
                    /* Read number of pass-variable and check. */
                read_m (3, pass_var);
                    /* Assemble the first sub-block with three
                    control lines and given pass-variable. */
                break;
            /*
            Since control = 7, this control has

```

been broken up into control = 4 and 3.

```

*/
case 7:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
        control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (3, pass_var);
        /* Assemble the first sub-block with three
        control lines and given pass-variable. */
    break;

```

```

/*
Since control = 8, this control has
been broken up into control = 4 and 4.
*/

```

```

case 8:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
        control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
        control lines and given pass-variable. */
    break;

```

```

/*
Since control = 9, this control has
been broken up into control = 5 and 4.
*/

```

```

case 9:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
        control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
        control lines and given pass-variable. */
    break;

```

```

/*
Since control = 10, this control has
been broken up into control = 5 and 5.
*/

```

```

case 10:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    break;
/*
Since control = 11, this control has
been broken up into control = 5, 3 and 3.
*/
case 11:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (3, pass_var);
        /* Assemble the first sub-block with three
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (3, pass_var);
        /* Assemble the first sub-block with three
           control lines and given pass-variable. */
    break;
/*
Since control = 12, this control has
been broken up into control = 4, 4 and 4.
*/
case 12:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    printf ("0");

```

```

    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    break;
/*
Since control = 13, this control has
been broken up into control = 5, 4 and 4.
*/
case 13:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    break;
/*
Since control = 14, this control has
been broken up into control = 5, 5 and 4.
*/
case 14:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
           control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (4, pass_var);
        /* Assemble the first sub-block with four
           control lines and given pass-variable. */
    break;
/*

```

Since control = 15, this control has been broken up into control = 5, 5 and 5.

```

*/
case 15:
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
        control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
        control lines and given pass-variable. */
    printf ("0");
    read_n (control);
        /* Read number of pass-variable and check. */
    read_m (5, pass_var);
        /* Assemble the first sub-block with five
        control lines and given pass-variable. */
    break;
    }
}
ps ("BFG",bfg);
cflstop ();
}
/*=====*/

```

```

/*=====*/
read_m (control, pass_var)
int control, pass_var;
{

    char c, input;
    int j, const;

    /*-----*
    * Read product term of all rows (the product term of a row
    * will be connected serially) and pass_variable terms, and
    * assemble the block:
    *-----*/

    boot_ckt (control);
        /* Get the assembled controls of the block. */

    for (i = 0; i < pass_var; ++i)
    {
        const = fscanf (fp, "%s%s", con, pvar);
            /* Read product and pass_variable from the 'input' file. */

        if (const != 2)
            /* Checking for product and pass_variable term. */
            {
                printf ("Missing product or pass_variable 0);
            }

        else
            {
                strcpy (prod[i], con);
                    /* Copy control into an array. */
                strcpy (labl[i], pvar);
                    /* Copy pass-variable into an array. */
            }

        out_array (control, prod[i]);
            /* Get the assembled rows of product. */

        if (i==0)
            {
                rows = row;
                    /* Assign the assembled row into symbol rows. */
            }

        else
            {
                rows = rr (rows, row);
                    /* Assemble rows of product terms. */
            }
    }
}

```

```
boots = rrdx (rows, boots, 60);
        /* Assemble controls and Product terms. */

rout (pass_var);
        /* Get the assembled routs. */
btlrrr = bbdxy (boots, rrouts, -60, 2);
        /* Assemble controls, left rout and right routs. */
precharge(pass_var);
        /* Get the assembled precharge. */
if (bfg == NULL)
        /* If the symbol bfg (Boolean-function generator) is
        NULL assemble the first block. */
    {
        btlrrrpc = bbdx (passes, btlrrr, -75);
        bfg = bbdx (btlrrrpc, pcharges, -25);
    }
else
        /* Otherwise assemble the second or the third block. */
    {
        btlrrrpc2 = bbdx (passes2, btlrrr, -75);
        bfg = bbdx (bfg, bbdx (btlrrrpc2, pcharges, -25), -25);
    }
}

/*=====*/
```



```

/*=====*/
out_array (control_no, prod)
int control_no;
char *prod;
{
    int i;

    /*-----*
    * Maximum number for control line for one block is 4, so
    * check and built one row of product terms of the block:
    *-----*/

    row = contclk;
        /* Initialize the symbol contclk into the symbol row. */
    for (i = 0; i <= control_no; i++)

    {
        if ( prod [i] == '-' )
        {
            printf (" - ");
            row = bb (row, dcare);
        }
        else if (prod [i] == '1')
        {
            printf (" 1 ");
            row = bb (row, cont);
        }
        else if (prod [i] == '0')
        {
            printf (" 0 ");
            row = bb (row, mx (cont));
        }
    }
    printf ("0");
    printf ("0");
}
/*=====*/

```

```

/*=====*/
read_n (control)
int control;
{
    int user_input = 1;

    /*-----*
    * Read number of pass_variable from the input file 'input'
    * and check.
    *-----*/

    while( user_input)
    {
        fscanf (fp, "%s%d", chc, &pass_var);
            /* Read number of pass_variable. */
        if (strcmp (chc, ".p") != 0)
            /* Check for '.p' card. */
            {
                printf (" '.p' is missing. 0);
            }
        else if (pass_var <= power (2, control))

            /* check number of pass_variable. */
            {
                user_input = 0;
                /* valid number of pass_variable found. */
            }
        else
            {
                printf ("Invalid number of pass_variable0);
                printf ("Please enter another value:0);
            }
    }
}

/*=====*/

```

```

/*=====*/
boot_ckt (control)
int control;
{
    int i;
    extern int offset;

    /*-----*
    * Assemble controls of the product term (bootstrap circuit)
    * and routs, and label controls.
    *-----*/

    boots = boot;
        /* Assign a control symbol boot into symbol boots. */

    for (i=1; i < control; ++i)
    {
        boots = bb (boots, boot);
            /* Assemble control lines of the product terms. */
    }

    /*-----*
    * Labeling all the control lines.
    *-----*/

    if (bfg == NULL)
    {
        for (i=0; i < control; ++i)
        {
            sprintf (labelstring, "c.%d", i);
            boots = cp (mlabel (labelstring, 0, 0, "c", "polysilicon"),
                pt (boots, "top", "polysilicon", i+1));
            /* Label the first control line. */
        }
    }
    else /* Label all the rest of control lines. */
    {
        for (i=0; i < control; ++i)
        {
            sprintf (labelstring, "c.%d", i + offset);
            boots = cp (mlabel (labelstring, 0, 0, "c", "polysilicon"),
                pt (boots, "top", "polysilicon", i+1));
        }
    }
    offset = offset + control;
        /* Set the counter. */
    boots = bb (lrout, boots);
        /* Assemble controls and top left routs. */
    boots = bb (boots, rrtop);
        /* Assemble controls and right top routs. */
}
/*=====*/

```

```

/*=====*/
precharge (pass_var)

int pass_var;
{
    pcharges = prechg;
        /* Assign the symbol prechge (the precharge device
        of the fabric) into symbol pcharge. */

    /*-----*
    * Assemble precharge and product term of all rows
    * of pass transistors, and routs between the sub-block
    *-----*/

    for (i=1; i<pass_var, ++i)
    {
        pcharges = ll (pcharges, prechg);
            /* Assemble precharge device of the fabric. */
    }

    if (bfg == NULL)
        /* For the first block assemble the input buffer
        and label them. */
    {
        for (i=1; i <= pass_var, ++i)
            /* Assemble the first input buffer. */
        {
            if (i==1)
                /* Get the first input buffer in case the
                input is 0, 1 or x; mirror the input
                buffer in y-coordinate in case of input xn. */
            {
                if ((strcmp (labl[(i-1)], "0") == 0)
                    || (strcmp (labl[(i-1)], "1") == 0)
                    || (strcmp (labl[(i-1)], "x") == 0))
                {
                    passes = pass;
                }

                else if ((strcmp (labl[(i-1)], "xn") == 0))
                {
                    passes = my (pass);
                }
            }
            else
                /* Assemble all of the input buffer. */
        {

```

```

if ((strcmp (labl[(i-1)], "0") == 0)
    || (strcmp (labl[(i-1)], "1") == 0)
    || (strcmp (labl[(i-1)], "x") == 0))
{
    passes = rr (pass, passes);
    /* Assemble all the input buffers of
       block for input 0, 1 and x. */
}

else if ((strcmp (labl[(i-1)], "xn") == 0))
{
    passes = rr (my (pass), passes);
    /* Assemble all the input buffers of
       block for input xn (which for
       the case of xn the input buffer will
       be mirrored imaged in y-coordinate). */
}

}

}

for (i=1; i <= pass_var; ++i)
    /* Label input of the input buffer. */
    {

        if ((strcmp (labl[(i-1)], "0") == 0)
            || (strcmp (labl[(i-1)], "1") == 0)
            || (strcmp (labl[(i-1)], "x") == 0))
        {
            passes = cp (mlabel (labl[(i-1)], 0, 0, "c",
                               "metall"),
                        pt(passes, "left", "metall", i));
            /* Label the input buffer on the input
               node, metall, with 0, 1 or x
               depending on the input. */
        }

        else if ((strcmp (labl[(i-1)], "xn") == 0))
        {
            passes = cp (mlabel (labl[(i-1)], 0, 0, "c",
                               "metall"),
                        pt(passes, "left", "metall", i));
            /* Label the input buffer on the input
               node, metall, with xn. */
        }

    }

}

else
    /* Assemble the second input buffer of the second block. */
    {

```

```

for (i=0; i<pass_var, ++i)
{
    if (strcmp (labl[i], "f") == 0)
    {
        p2 = bb (rf, pass2);
        /* Assemble output of first block with
           the input buffer of the second block. */
    }
    else if (strcmp (labl[i], "fn") == 0)
    {
        p2 = bb (rfn, pass2);
        /* Assemble output of first block with
           the input buffer of second of block. */
    }
    if (i == 0)
    {
        passes2 = p2;
        /* Assign the input buffer of second block,
           symbol p2, into symbol passes 2. */
    }
    if (i >= 1)
    {
        passes2 = rr (passes2, p2);
        /* Assemble the second or third
           input buffer of block. */
    }
}
}

/*=====*/

```

```
/*=====*/
rout (pass_var)

int pass_var;
{
    /*-----*
    * This routine assembles the routing on the right
    * side of the block.
    *-----*/

    int i;

    rrouts = rroutbot;
        /* Initialize the right-bottom-rout cell (rroutbot) to rrouts. */

    for (i=2; i<pass_var; ++i)
        /* Assemble the right routs of the block. */
    {
        rrouts = ll (rrouts, rrout);
    }
        rrouts = ll (rrouts, rrouttop);

}
/*=====*/
```

```
/*=====*/  
power (base, x)  
  
int base, x;  
{  
    /*-----*  
    * This routin calculates base to power of a variable.  
    *-----*/  
  
    int i, j = 1;  
  
    for ( i = 1; i <= x; i++)  
    {  
        j *= base;  
    }  
    return(j);  
}  
/*=====*/
```


APPENDIX D

EXAMPLES OF THE LAYOUT CIRCUITS

