TESTING AND SIMULATION INVOLVED IN THE CIRCUIT DESIGN

A dissertation submitted in partial fulfilment of the requirements for the M. Tech. (Computer Science) degree of the Indian Statistical Institute

By

Gomathi Nayagam N.

Under The Supervision of

Dr. Bhargab Bhattacharya
Electronics Unit.
INDIAN STATISTICAL INSTITUTE
203, Barrackpore Trunk Road,
Calcutta-700035

July 24, 1996

Indian Statistical Institute
203, B.T. Road,
Calcutta- 700 035.

Certificate of Approval

This is to certify that the thesis titled **TESTING AND SIMULATION INVOLVED IN THE CIRCUIT DESIGN** submitted by **N.Gomathi Nayagam** , towards partial fulfillment of the requirements for the degree of M. Tech. in Computer Science at the Indian Statistical Institute, Calcutta, embodies the work done under my supervision.

Bhargab B. Bhattacharya
Dr. Bhargab Bhattacharya 30-7-96
Electronics Unit,
Indian Statistical Institute,
Calcutta-700 035.

Acknowledgements

First I would like to thank my guide Prof.Bhargab Bhattacharya who provided me with an industrially interesting problem. I would also like to thank him for his excellent guidance.

I would like to thank my classmates Mr .M.Y.Patnaik and Subhasis Majumdar for the fruitful sessions I had with them regarding my dissertation.

Finally I thank sincerely my classmate Mr.Kausik Dutta who helped me in latexing the report.

July 10,1996 (Gomathi Nayagam)

The best way of overcoming a difficult problem is to solve it in some particular easy cases. This gives much light into the general solution. By this way Newton says he overcame the most difficult things.

David Gregory

Contents

1 Basics and Reviews **3**

1.1 VLSI Design Cycle 3

1.1.1 System Specification 3

1.1.2 Functional Design 3

1.1.3 Logic Design 4

1.1.4 Circuit Design 4

1.1.5 Physical Design 4

1.1.6 Design Verification 4

1.1.7 Fabrication 4

1.1.8 Packaging, Testing and Debugging 5

1.2 Design Styles 5

1.3 MOS Logic Synthesis - A Review 5

1.3.1 Some n-MOS Basic Circuits 6

1.3.2 NAND AND NOR GATES 7

1.3.3 Function to Circuit 7

2 About the Problem **8**

2.1 Motivation 8

2.2 Problem Specification 9

2.3 Definitions 9

2.4 Mode Of Attacking the Problem. 11

3 Proposed Data Structures **13**

3.1 TYPE 1 13

3.1.1 Initialization Phase 14

3.1.2 Connections Phase 16

3.2 TYPE 2 18

3.3 TYPE 3 20

3.4 TYPE 4 22

3.4.1 Initialization Phase(Interactive Mode) 22
3.4.2 Connection Phase(Interactive Mode) 23
3.4.3 Analysis 24

4 Algorithm Proposed and Applications **25**
4.1 The Case of Passmode Transistor 25
4.2 Proposed Algorithm 26
4.3 Applications 28

5 Results And Conclusions **31**
5.1 Implementation 31
5.2 Details of Various Functions 32
5.3 Conclusions and Future Work 36

6 REFERENCES **37**

Chapter 1

Basics and Reviews

1.1 VLSI Design Cycle

In this chapter, we briefly review the VLSI Design Cycle. It starts with a formal specification of a VLSI chip, follows a series of steps and finally produces a packaged chip. A typical design cycle is represented in the diagram 1.1.

1.1.1 System Specification

The first step is to formulate the specifications of the system to be designed. It will be basically a high level representation of the system. The factors to be considered are: performance, functionality and the physical dimensions. The choice of fabrication technology and design techniques are also considered here. The end results are specifications for the size, speed, power and functionality of the VLSI system to be designed.

1.1.2 Functional Design

The behavioral aspects of the system are considered here. The result is usually a timing diagram or other relationships between subunits to improve the overall design process and to reduce the complexity of the subsequent phases.

1.1.3 Logic Design

Here, the logic structure that represents the functional design is derived and tested. Usually, logic design is represented by boolean expressions. This logical design of the system is simulated and tested to verify its correctness.

1.1.4 Circuit Design

Here, the circuit representation based on the logic design is developed. The circuit design is usually expressed in a detailed circuit diagram. Here also, the circuit design is simulated and tested to verify its correctness and to reduce the complexity which might arise in subsequent phases.

1.1.5 Physical Design

In this step, the circuit representation of each component is converted into a geometric representation. This geometric representation of a circuit is called a layout. The physical design is a complex process, and is split up into various units like partitioning, placement, floorplanning, routing and compaction.

1.1.6 Design Verification

The layout is verified to ensure that it meets the system specifications and the fabrication requirements. Design verification consists of Design Rule Checking (DRC) and Circuit Extraction. DRC is a process which verifies that all geometric patterns meet the design rules imposed by the fabrication process. Then the functionality of the circuit is verified by circuit extraction. This is a reverse engineering process and generates the circuit from the layout.

1.1.7 Fabrication

After the verification, fabrication is done. This consists of preparation of wafer, deposition and diffusion of various materials on the wafer according to layout description.

1.1.8 Packaging, Testing and Debugging

Finally the wafer is fabricated and diced in a fabrication facility. Each chip is then packaged and tested to ensure that it meets all the design specifications.

The present work is concerned with Circuit and Logical design phases of the VLSI design cycle.

1.2 Design Styles

In order to achieve the quick time to market, high yield and reducing the complexity of physical design, restricted models and design styles are used. The design styles can be broadly classified as either full-custom or semi-custom. In a full-custom layout, different blocks of a circuit can be placed at any location on a silicon wafer as long as all the blocks are non-overlapping. On the other hand, in semi-custom layout, some parts of a circuit are pre-designed and placed on some specific places on the silicon wafer. Full custom is a preferred style for mass produced chips keeping in mind the time to market. To design an Application Specific Integrated Circuit(ASIC), a semi-custom layout style is usually preferred.

1.3 MOS Logic Synthesis - A Review

Integrated systems in metal-oxide semiconductor (MOS) actually contain three or more layers of conducting materials separated by intervening layers of insulating material. **The symbol of a n-MOS transistor is shown in diagram** 1.2. There are three terminals gate, source and drain. The transistors that are non-conducting with zero gate bias(gate to source voltage) are called enhancement mode transistors. Most MOS IC's use transistors of enhancement type. The transistors that conduct with zero gate bias are called depletion mode transistors.
CMOS : MOS circuits dissipate i^2R power when the output is low. The heat generated is hard to remove and impedes the performance of these circuits. To overcome this, a combination of pMOS and nMOS transistors can be used in building structures which dissipate power only while switching. This type of structure is called CMOS(complementary metal oxide semiconductor).

CMOS is an inherently low power circuit technology with the capability of providing a lower power-delay product comparable in design rule to nMOS and pMOS technologies. For all inputs, there is always a path from '1' or '0' to the output and the full supply voltage appears at the output. Another advantage of cMOS is that there is no direct path between VDD and GND for any combination of inputs. This is the basis for the low static power dissipation in cMOS.
The Table below illustrates the main differences between nMOS and cMOS technology.

CMOS	NMOS
Zero Static Power dissipation	Power is dissipated in the circuit with output of gate at '0'
Power dissipated during logic transition	Power dissipated during logic transition
requires 2N devices for N inputs for complementary static gates	Requires(N+1)devices for for N inputs
c-MOS encourages regular layout styles	Depletion,load and different driver transistors create irregularity in layout

We tackle our problem for the n-MOS case with the same solution being applicable to c-MOS also.

1.3.1 Some n-MOS Basic Circuits

Before going into our problem of function extraction which is essentially travelling from the direction of circuit to that of function, let's see the other way round first. i.e., constructing circuit corresponding to given boolean function.
The general structure of n-MOS circuit is shown in the diagram 1.3.
n-MOS Inverter The basic function of an n-MOS inverter is to produce an output that is complement of it's input. The logic table and logic symbol of a basic inverter are shown **in diagram 1.2**. If the inverter input voltage is less than the transistor threshold voltage $v_t$then the transistor is switched off and the output is pulled up to the positive supply voltage VDD. In this case the output is complement of the input. **Refer diagram 1.2.**

1.3.2 NAND AND NOR GATES

NAND and NOR logic circuits may be constructed in MOS systems as a simple extension of the basic inverter circuit. **Truth tables and logic symbolic diagrams are shown in the diagram 1.4.** In the NAND circuit,the output will be low only when both of the inputs A and B are high. The NAND gate simply consists of a basic inverter with an additional enhancement mode transistor in series with the pull-down transistor. NAND gates with more inputs may be constructed by adding more transistors in series with the pull-down path.
In the NOR circuit,the output is low if either of the inputs,A and B (or both) is(are) high. **The diagram 1.4** shows a two input NOR gate through a basic inverter with an additional enhancement mode transistor in parallel with the pull-down transistor.
Some basic n-MOS circuits and their corresponding outputs as boolean expressions are given **in diagrams 1.5 and 1.6**.Circuits in diagrams 1.6a and 1.6b are isomorphic(equivalent). In the second case,the total cost = 5+1 = 6 transistors.Almost 50% savings is achieved in the second case through the transistor E which is called a bridge type transistor. In the bridge transistor E,the role of drain and source can be swapped sothat we get paths passing through E in both the directions.

1.3.3 Function to Circuit

Suppose that we want to implement a function (whose boolean expression is given as a sum of product) through n-MOS . Then the following procedure is adapted to construct the circuit.

1. First Invert the function(i.e., findout $\overline{f}$)using boolean laws (De Morgan's Law).

2. construct the series-parallel portion for it and then attach the load transistor above it. i.e., while constructing series-parallel portion,each AND term in $\overline{f}$ is constructed using a series of transistors and finally these sequences of series are attached in parallel mode to make the final sum.

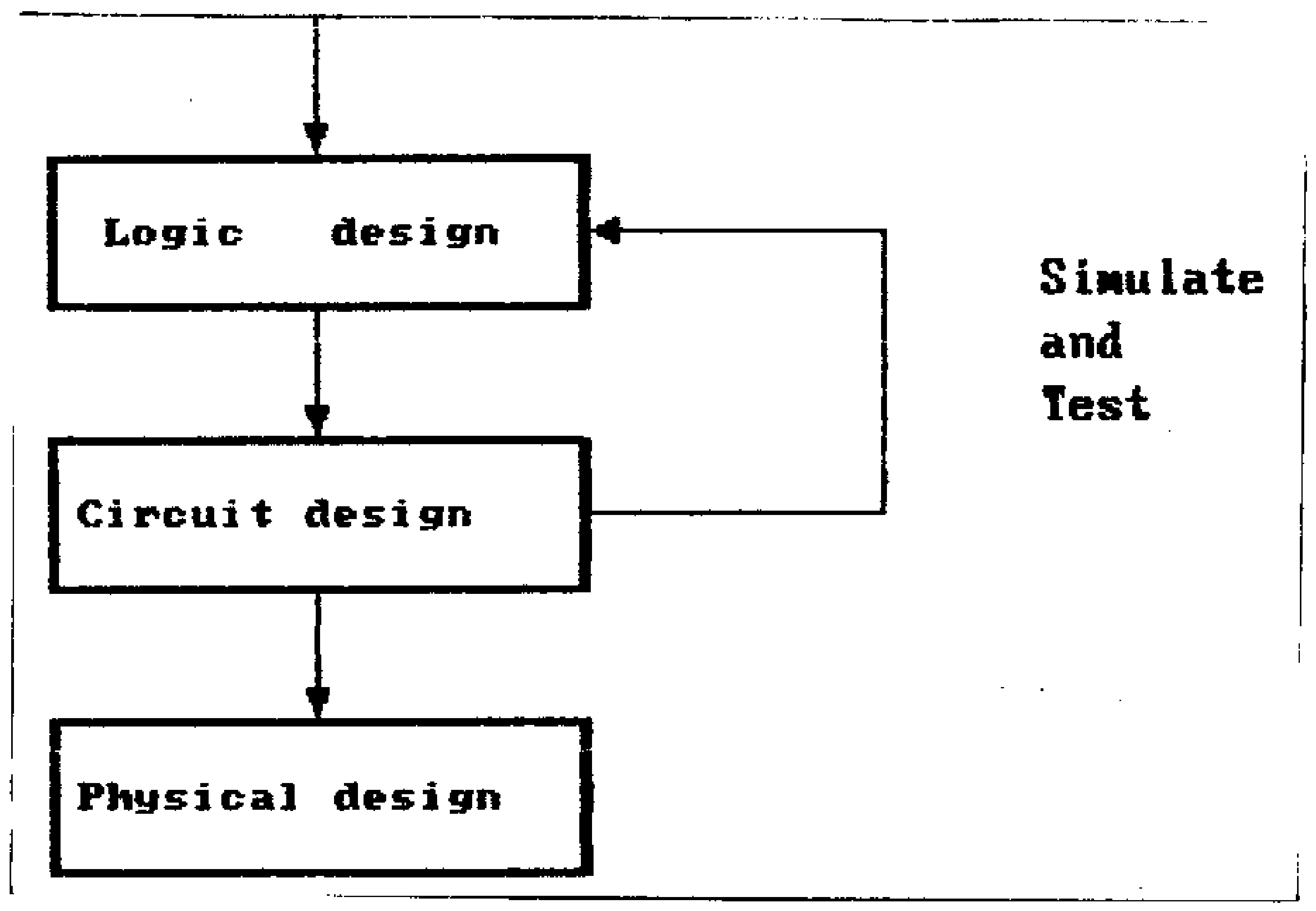
Chapter 2

About the Problem

2.1 Motivation

The purpose of circuit design is to develop a circuit representation based on the logic design. The boolean expressions(formed during logical design)are converted into a circuit representation. The speed and power requirements of original design are also considered here. The circuit design might be at the gate level or switch(MOS transistors)level.

The testing phase here is the reverse engineering process of extracting the function from the circuit. The goal is to check whether the circuit is doing what it is supposed to do. This process in a way,is analogous to circuit extraction as a part of design verification after the physical design.



Suppose that we replace a portion of the circuit by a component from VLSI library. The first necessary condition we have to ensure is that both the circuits are equivalent. The objective for such replacement is

1. Area Optimization.
2. Minimization of Gate Delays.
3. Better Performance in terms of speed and other Power requirements.

2.2 Problem Specification

1. First , to devise a method for creating user interface of dealing with MOS circuits and simulate the whole operation of the circuit.

2. To automate the testing process of circuit design through devising a strategy for extracting the function out of the circuit at the switch level. In our problems we specialize for series-parallel circuits.

3. To findout a method to systematize the problem of circuit Isomorphism i.e., to findout a systematic procedure for equality comparison between two functions.

2.3 Definitions

To clear out the certain terms in the above problem specification we propose the following definitions.
Assumptions : First we deal with undirected connected graphs inwhich two vertices are designated as 1. source and 2. sink(or target) and they are fixed afterwards. The following definitions depend on this assumption.

Series-Parallel Edge: In an undirected connected graph with the above assumption an edge uv is called a series - parallel edge iff every path from source to sink passing through the edge uv has the same order of u occuring first and v following it or vice-versa(i.e., every path through the edge has reverse ordering).

Note: A path from source to sink passing through uv is a directed one. Hence we call the edge uv as a directed edge. Note that uv is a series - parallel edge iff vu is not.

Series-Parallel Graph: Series parallel graph is a graph in which every edge is a series parallel edge. Similiarly we can define series-parallel notion for the directed graph also.

Alternative Definition: This is the alternate way of defining series- parallel graph recursively.

Definition: The following graph only with two vertices one designated as source(s) and the other as target(t),is series parallel.

source ——— target.

If ($s_1C_1t_1$) and ($s_2C_2t_2$) are series parallel graphs,then ($ss_1C_1t_1t$) and ($ss_2C_2t_2t$) connected in parallel or series is also a series parallel graph, where s is the source and t is the target in the above graph.

Series Parallel Circuit: Series parallel circuit(at the switch level) is a circuit of MOS transistors (n-MOS) whose underlying graph is series parallel.

Circuit Isomorphism: Two circuits are said to be isomorphic iff the following conditions hold.

1. No of primary inputs of circuit1 is equal to no of primary inputs of circuit 2.

2. There exists a one-one & onto function f: from primary inputs of circuit1, say $\{a_1,a_2,\dots,a_n\}$ to the primary inputs of circuit 2, say $\{b_1,b_2,\dots,b_n\}$such that two circuits produce the same output when both $a_1,a_2,\dots,a_n$ and $f(a_1),f(a_2),\dots,f(a_n)$ are given the same binary n-bit input string, in the same order.

Problem Of Circuit Isomorphism For switch level Circuits : Problem of Circuit Isomorphism is : Given two circuits at the switch level, to say decisively whether they are isomorphic or not?

2.4 Mode Of Attacking the Problem.

We transform MOS level circuit description into a graph for simplest representation and for further manipulation. The graph is constructed as follows:
Each edge in the graph represents a transistor in the circuit and vertices represent the junction between the transistors. **Some examples are given in the diagram 2.1** .
Edges are named by the signal names applied to the transistors. The above Circuit transformation graphs can be broadly classified into two categories.

1. Interconnect of Switch-Mode Transistors
   (a) Series Parallel Graphs
   (b) Non Series Parallel Graphs
2. General Interconnect (includes passmode transistors)
   (a) Series Parallel Graphs
   (b) Non Series Parallel Graphs

Examples are shown in the diagram 2.2.
A gate level specification(as a sum of product) is essentially a series-parallel structure. A MOS Network supports a general structure.

Suppose an n-MOS circuit is given. We specialize in the case of series parallel types of circuits.

1. First we have to find an efficient way of encoding the circuit into the computer so that we can take the circuit as the input from the user and do further necessary manipulations on it. The data structures which we choose must be linear order in the no of transistors in the n-MOS circuit, as far as the space complexity is concerned. Another point to be noted in this scheme is that we should check for the consistency of the circuit. ie., false specifications of a circuit (such as specifying less no of connections) by the user must be trapped.

2. We have to simulate the operation of the circuit to check the proper functionality of the circuit.

3. From the given circuit, we have to build the corresponding series parallel graph.

4. From the graph, we have to extract the product terms by choosing an appropriate path traversing algorithm from the source to sink. Each path will give rise to one product term. The set of all possible paths in the graph from vdd to earth will depict the fuctionality of the whole circuit.

Chapter 3

Proposed Data Structures

3.1 TYPE 1

The first type of data structures is the following,

```
struct connection
{
      int code1;
      int code2;
      struct connection *next;


}
typedef struct connection connection;

struct voltage
{
      int value;
      connection *next;
}
typedef struct voltage voltage;

struct transistor
{
        voltage gate;
        voltage source;
        voltage drain;

    }

    typedef struct transistor transistor;
```

Now through malloc n transistors can be created.

To understand the structure involved in the above formulations let's work it on this specific example.

consider diagram 3.1

```
transistor t1;
transistor t2;
transistor t3;
```

Now 3 transistors will be created through malloc.

3.1.1 Initialization Phase

Input

```
give the no of transistors in the circuitory
3

give the no of transistors connected to 1 th transistor source
2

give the no of transistors connected to 1 th transistor drain
1

give the signal name for the gate of 1 th transistor
1

give the no of transistors connected to 2 th transistor source
```

```
2

give the no of transistors connected to 2 th transistor drain
1

give the signal name for the gate of 2 th transistor
2

give the no of transistors connected to 3 th transistor source
1

give the no of transistors connected to 3 th transistor drain
2

give the signal name for the gate of 3 th transistor
3

give the total no of transistors to be grounded
1

give the transistors to be grounded & terminate by -1
3
-1

give the total no of transistors to be connected to vdd
2

give the transistors to be connected to vdd & terminate by -1
1
2
-1
```

output

```
Printing the trans connected to vdd ...
1
2
```

```
Printing the trans connected to earth ...
3
```

For the output please refer the diagram 3.2 .

1. Purpose of 'value' in voltage is to store the logic values of all gate, source and drain.
   For drain,source we keep in mind the operation of the circuit. i.e., When the gate value is 1, the transistor gets short circuited and logic value of drain becomes as the logic value of source.

2. In the initialization phase, if a transistor is connected to vdd, the value in voltage of transistor's drain is made 1 and is fixed for ever. Similiarly, if a transistor is connected to ground, the value in voltage of that transistor's source is made 0 and is fixed for ever.

3.1.2 Connections Phase

Next to the Initialization phase is the stage of making actual connections.

<u>Input</u>

```
give the next transistor to which the source of the 1 th transistor is connected.
2

give the code
1.source 2.drain1

give the next transistor to which the source of the 1 th transistor is connected.
3

give the code
1.source 2.drain2

give the next transistor to which the drain of the 1 th transistor is connected.
```

```
2

give the code
1.source 2.drain2

give the next transistor to which the source of the 2 th transistor is con-
nected.
3

give the code
1.source 2.drain2
give the next transistor to which the source of the 2 th transistor is con-
nected.
1

give the code
1.source 2.drain1

give the next transistor to which the drain of the 2 th transistor is con-
nected.
1

give the code
1.source 2.drain2

give the next transistor to which the source of the 3 th transistor is con-
nected.
99

give the code
1.source 2.drain99

give the next transistor to which the drain of the 3 th transistor is con-
nected.
1

give the code
1.source 2.drain1
```

give the next transistor to which the drain of the 3 th transistor is connected.
2

give the code
1.source 2.drainl

For the output consider the diagram 3.3.

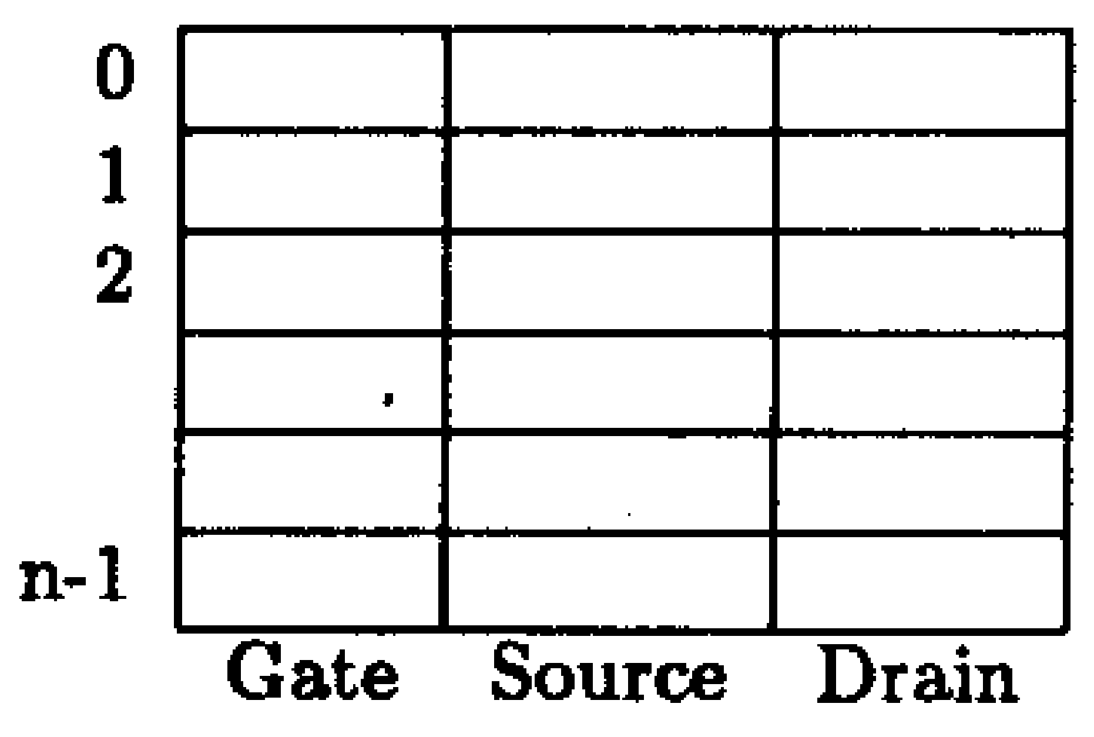
While making connections,codes 1 and 2 are used for the terminals source and drain respectively. When code2 of connection of some transistor is -1,it means that it corresponds to the gate of that transistor. Value field in voltage is kept, keeping in mind the operation of the circuit. Also '99' means that the pariticular terminal is connected to ground.
Advantages Data structures are chosen in such a way that we can straightaway do some 'dfs' kind of path tracing algorithm on it. i.e., Take the trans whose initial drain voltage value is 1(connected to vdd) and continue some suitable path tracing algorithm on it until we find a transistor whose source voltage is 0. Then we will get a path from vdd to the ground.
Disadvantages

1. There is a lot of redundancies in the circuit. i.e., Both the informations 'a' is connected to 'b' and 'b' is connected to 'a' are kept. This is a trade off for the fact that we will be able to travel in both the directions. i.e., vdd to the ground as well as ground to the vdd.

2. It takes a lot of space,though it seems to be linear in the space complexity. Much space is wasted for the pointers, incase no of transistors in the circuit is large.

3.2 TYPE 2

The second type of data structures we propose, reduces the space for the pointers. It is simply an $n * 3$ grid where n is the total no of transistors in the circuitory.



```
struct transtype {

      int no;
      char terminalcode;


}

typedef transtype transtype;
```

Now we declare the grid as

```
          transtype transistor[][3];
```

[1] where

- transistor[i][0] will contain the info regarding gate of i+1 th transistor.
- transistor[i][1] will contain the info regarding source of i+1 th transistor.
- transistor[i][2] will contain the info regarding drain of i+1 th transistor.

For an example, let's consider the following circuit.

Refer the diagram 3.4a
Through malloc 6 transistors(i.e., 6*3 grid)will be created & the following info is entered in the grid to specify the connections of the circuit.

Refer the diagram 3.4b

[1]In C, the things in 2 dimensional array are mapped to 1 dimensional physical array in row-majorized fashion. Hence column no should get specified for the compiler to resolve references.

In the above grid, code 's'stands for source & 'd' stands for drain. Further info '1v' means connected to vdd and info 'og' means connected to ground. In this data structure junction points are identified. In a junction, the no of terminals which meet together is finite. This data structure travels around these terminals of transistors meeting at that junction in a cyclic manner. In the above example, at the junction 'a' 1s 2d 3d & 4d meet together. Hence corresponding to 1s location(i.e.,transistor[0][1]) 2d is entered, corresponding to '2d' location(i.e.,transistor[1][2]) 3d is entered, corresponding to '3d' location(i.e.,transistor[2][2]) 4d is entered and finally to complete the cycle corresponding to '4d' location(i.e., transistor[3][2]) 1s is entered. Likewise junction 'b' is also completed. After filling the junctions, the remaining locations are 1d which being connected to vdd has the info '1v', 5s & 6s which being connected to ground, have the infos as 0g, 0g respectively.

3.3 TYPE 3

The third type of data structure we propose is the best one in terms of space complexity.But it has one major assumption that the user is able to visualize his circuit by drawing it in a paper. The user has to identify all the different junction points of the circuit. It is just n*2 int array.

int transistor[][2];

Now after plotting the junction points,each transistor will be between the unique junctions. Those junction points are entered.The junction point above the transistor i is entered in transistor[i][0] and the junction point below the transistor i is entered in transistor[i][1]. If the bridge types of transistor are also allowed(i.e.,in the case of non-series parallel), we can attach a 1 bit information for every transistor regarding decision yes or no(1 or 0).

Consider this example in diagram 3.5.

There are only two junction points in the above circuit which are labelled 1, 2 & 3. Then data structure looks like,

first transistor $\rightarrow$ *transistor[0][0] = 1, transistor[0][1] = 2*

Second transistor$\rightarrow$ *transistor[1][0]* = *1*, *transistor[1][1]* = *2*
Third transistor$\rightarrow$ *transistor[2][0]* = *2*, *transistor[2][1]* = *3*
Fourth transistor$\rightarrow$ *transistor[3][0]* = *2*, *transistor[3][1]* = *3*
Fifth transistor $\rightarrow$ *transistor[4][0]* = *2*, *transistor[4][1]* = *3*

Space Complexity It takes just $2n$ integer locations in case of series-parallel circuits and $2n$ integer + n bits in case of circuits supporting bridges also.

Advantages It is also optimal in the sense of space complexity. Further we can extract the underlying graph of the circuit from the data structures quite easily through the following algorithm. We will construct the graph in terms of adjacency list of edges rather than vertices.

The pseudo algorithm is given below,

```
do for all i & j such that i≠j
if( transistor[i][1] == transistor[j][o] )
then
    the transistor i points to j
    i.e. j is included in the adjacency list of i
```

The above algorithm has quadratic time complexity with respect to n, the number of transistors in the circuit.

There is another data structure which is analogous to the above one. Here to every transistor a link list is maintained in which, all the other transistors immediately up and down that particular transistor are entered. Each node of the link list has two informations,

- The transistor number
- 1 bit info (1 or 0) saying whether it is up or down

For example corresponding to the above circuit, **we have the list as shown in the diagram 3.5.**

3.4 TYPE 4

The following 4th type of data structure is the one we are going to implement. It contains dynamically allocated n arrays. Further each array is dynamic.

$$int **source;$$

is declared. And then through malloc source[1], source[2], ..., source[n] are created where source[i] is a pointer to an integer array of size m, where m is the number of transistor connected with the source of i^{th}. Through further malloc during runtime, source[i][0], source[i][1], ..., source[i][m] are created. Note that m depends on i.

V_{dd} is assumed to be fixed ever for the drain terminals of some transistors. Then ground will be connected to source terminals of few transistors. Because there is an inherent directions from drain to source in series-parallel graph, we attach informations regarding source of each transistor. This data structure is advantageous over the first type in the sense that many redundancies are removed here. While entering informations codes 's' and 'd' are used for source and drain respectively.

We depict the insight of the above data structures through the following example as shown in the diagram 3.7 .

3.4.1 Initialization Phase(Interactive Mode)

<u>**Input**</u>

```
give the no of transistors in the circuitory
3

give the no of transistors connected to 1 th transistor source
2

give the no of transistors connected to 2 th transistor source
1

give the no of transistors connected to 3 th transistor source
1
```

```
give the total no of transistors to be grounded
2

give the transistors to be grounded & terminate by -1
2
3
-1

give the total no of transistors to be connected to vdd
1

give the transistors to be connected to vdd & terminate by -1
1
-1
```

Output

For output consider the diagram 3.6 .

3.4.2 Connection Phase(Interactive Mode)

Input

```
give the next transistor to which the source of the 1 th transistor is con-
nected.
2

give the terminal code "s.for source d.for drain"
d

give the next transistor to which the source of the 1 th transistor is con-
nected.
3

give the terminal code "s.for source d.for drain"
d

give the next transistor to which the source of the 2 th transistor is con-
```

```
nected..
99

give the terminal code "s.for source d.for drain"
d

give the next transistor to which the source of the 3 th transistor is con-
nected.
99

give the terminal code "s.for source d.for drain"
d
```

Output

For output consider the diagram 3.7 .

3.4.3 Analysis

Space Complexity Since a pointer takes 4 bytes, initially $4n$ bytes are allocated. And then $2n$ integer loctions are allocated for entering the informations. Hence this is having the linear order in space complexity where constant is better than the first type.

Disadvantages Here we can travel in only one direction from V_{dd} to earth and not the other way round. This data structure is more near to path tracing algorithm which is preferred in our work.

Chapter 4

Algorithm Proposed and Applications

4.1 The Case of Passmode Transistor

The case of passmode transistors is quite an interesting one. A passmode transistor is the one where signal can be applied to source or drain.

We propose a pseudo algorithm to simulate the effect of passmode transistors through series-parallel circuits.

Suppose that the circuit contains n transistors out of which m transistors are passmode type. Then we convert it to an equivalent series-parallel circuit with n+m transistors. The following algorithm(pseudo) is used here for that purpose.

Input	Circuit with passmode transistors.
Output	Equivalent circuit of series-parallel type.

Algorithm

1. Start branching out backwards from the output point towards the other ends where passmode transistors will be present.

2. For every series-parallel block which we come across draw the corresponding (same) block in the new circuit.

3. When we finally reach the free ends i.e., passmode transistors, include that transistor in series with the previous (immediately preceding) series- parallel block. And then include one more transistor in series in which you apply the signal as $\overline{x}$ where x is the signal applied to one end of passmode transistor.
4. Do the above step, till all the free ends (i.e. passmode transistors) are covered.

This is illustrated through the following example (ref diagram 4.1)

Both the circuits in the diagram are equivalent as can be seen from the fact,

$$\begin{aligned} f &= \overline{\overline{A}B + A\overline{B}} \\ &= (A + \overline{B})(\overline{A} + B) \\ &= AB + \overline{AB} \end{aligned}$$

Note that we are not extracting the function straightaway from the circuit with passmode transistors but we are trying to transform the circuit to series-parallel form which in turn will be taken as the input for the function extraction.

4.2 Proposed Algorithm

We use this basic algorithm repeatedly for the purpose of function extraction and operation of the circuit.

The input to the following algorithm will be an adjacency list of transistors. This is a recursive algorithm.

A global stack is used, which grows and shrinks accordingly as we travel in the graph. no_of_paths is also a global variable which is used to count the no of paths from v_{dd} to ground

Algorithm

```
all_paths(source)
{
    if(source == ground ) then
    {
        if(no_of_paths == 0) then
        {
            print stack;
            no_of_paths++;
            pop stack;
            return;
        }
    }
    else
    {
        print("+");
        print stack;
        no_of_paths++;
        pop stack;
        return;
    }
    for all nodes C which is in the adjacency list of source
    {
        if( c != stack[top-1] )
        {
            push stack(source);
            all_paths(c);
        }
    }
    pop stack;
    return;
}    /* end of Algorithm */
```

Finally call *all_paths(start)* is invoked where start points to all transistors connected to V_{DD}.

Illustrative Example is given in the diagram 4.4.

Analysis

The above algorithm is output sensitive i.e., any algorithm trying to find all paths from V_{dd} to ground will have complexity at least n_o, where n_o is the total number of paths.

Time Complexity

The complexity of above algorithm is precisely n_o for, if a path from V_{dd} to ground is traced, it is never traced again. The above algorithm has another advantage that if a portion of a path is common to many other paths, that particular portion is travelled only once.

Space Complexity

There are two stacks involved here. One is the user defined stack for maintaining a path and another is the system defined stack for maintaining the recursion. Both the stacks grow according as a path from V_{dd} grows. Hence the worst case space complexity is the length of the longest path from V_{dd} to ground. The number of automatic variables is also very less (one used in for loop and other things are global).

4.3 Applications

BDD (Binary Decision Diagrams)

Function extraction is applied in Circuit isomorphism problems i.e., From the two given input circuits, We have to at first extract functions. We can't compare the boolean functions just like that, since even minimal SOP(Sum of Product) form is not unique.

In general given,

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

$$g : \{0,1\}^n \rightarrow \{0,1\}$$

is $f = g$? is NP complete. i.e., operating the circuit for all input vectors and observing the logical value is not practically feasible. Hence we must have an efficient way by which verification (comparison) can be done. For that we go for the tool what is known as Binary Decision Diagrams.

A BDD is a directed acyclic graph with 2 paths directed away from itself, one for the node asserted true and one for the node asserted false. These approaches define a digital function in terms of a diagram which represents the fucntion and contains the information necessary to implement the function. Nodes of BDD are either variables or subfunctions. Ultimately nodes will be reduced to single variables. The root of the tree is the function to be implemented and leaves are either 0's or 1's. We can construct BDD's either from boolean expressions or from the truth table.We formally define it as follows.

Definition A Binary Decision Diagram(BDD) over a set $\{x_1, x_2, ..., x_n\}$ of boolean variables is a directed acyclic graph with one source and atmost two sinks labelled 0 and 1. Each non-sink node v is labelled with a boolean variable from x_i's and has two outgoing edges,one labelled with 0 and the other with 1. The then-son of v is reached via 1-edge , the else-son is reached via the 0 edge. (In pictural representations, we do not indicate the edge labels if the 0-edge is drawn left of the 1-edge.) The computation path for an input $a = (a_1, ..., a_n)$ starts at the source. At an inner node with label x_i, the outgoing edge with label a_i is chosen. Size(P) denotes the number of non-sink nodes of P. A BDD P represents a boolean function $f \in B_n$ if the computation path for each input a leads to the sink labelled $f(a)$. A BDD is called ordered binary decision diagram(OBDD) if, on each path, the variables are tested consistently with the natural order of variales $x_1 \leq x_2 \leq ..., \leq x_n$. **Fact**

1. Each boolean function f over x_n can be represented by means of an OBDD, i.e., OBDD's provide a universal representation scheme.

2. The reduced OBDD for f is uniquely determined, i.e., it provides a canonical representation.

3. Let f_1,f_2 be boolean functions represented by the OBDDs P_1, P_2 respectively. For every binary operation *, the reduced OBDD P for $f = f_1 * f_2$ can be constructed in O(size(P_1).size(P_2)).

Factored Form

A factored form is a boolean formula with the restriction that complement operation is allowed only on variables. A factored form is represented as a labelled leaf-DAG where the + and . operations alternate on all paths. If a function has a factored form which is a tree (i.e., each variable appears only once in the factored form), the function is said to be series-parallel. Deriving a factored form for a function is called factoring. **Refer diagram 4.2.**

Reduced Ordered BDDs and Properties

It represents a function with a directed acyclic graph (as told earlier) where paths to 1 node define the cubes for which the function is 1. 'Ordered' means that each paths visits the variables in the same order. 'Reduced' means that isomorphic subgraphs are maximally shared. It is conceptually similiar to full shannon decomposition tree, except that subtrees which are identical are shared and redundant nodes are deleted. **Refer diagram 4.2.**

Properties of BDD : Reduced ordered BDD's exactly satisfy a very important property that two functions are equal if and only if their representations as ROBDD's are isomorphic DAGs (for the same variable ordering). Graph isomorphism for a DAG is a linear-time operation. For this reason ROBDD's are sometimes called a canonical form. Because this property is so important, the term BDD almost always means a reduced ordered BDD. It is to be noted that the size of the DAG for a function is strongly dependent on the ordering of the variables in the BDD.

Chapter 5

Results And Conclusions

5.1 Implementation

The function extraction and circuit operation have been implemented for the case of series-parallel circuit at the switch level.We use the data structures as mentioned in the type 4 of chapter4.
The Whole objective of this program is to create some user interface for specifying circuits at the switch level, operating the circuits and finally making a provision to test the circuit by doing function extraction. The Input circuits are shown in the diagrams 5.1 and 5.2. The source files and the outputs are appended with this project report.
There are 5 files to be included(or linked)with the main program.

1. setup.c This deals with setting up initial base for specifying the circuit.

2. construction.c This deals with building up valid circuit and removing redundancies.

3. extract.c This contains functions responsible for function extraction.

4. print.c This contains all support files for outputting the circuit in various formats.

5. **stack.c** This contains all support files for a stack to be used in the main program as well as in a function in the file extract.c.

5.2 Details of Various Functions

The following is the brief information about various functions.

1. function input

Input	:	void
Returnvalue	:	void
Function	:	It takes the total number of transistors in the circuitory

2. function initialize

Input	:	int
Returnvalue	:	void
Function	:	It creates an initial base for building a circuit

3. function get_signal_names

Input	:	void
Returnvalue	:	void
Function	:	It gets the signal names of all the primary inputs

4. function vdd_earth

Input	:	void
Returnvalue	:	void
Function	:	It takes the transistors connected to vdd and earth and sets pointers to them

5. function output_initial_configuaration

Input	:	int
Returnvalue	:	void
Function	:	It outputs the initial configuaration of the the circuit before making respective connections

6. function create_valid_circuit

Input	:	int
Returnvalue	:	void
Function	:	It makes connections to each transistor and checks for consistency of the circuit.

7. function is_circuit_complete

Input	:	void
Returnvalue	:	void
Function	:	It checks whether the circuit connections are complete

8. function compress_circuit

Input	:	int
Returnvalue	:	void
Function	:	It checks and removes any redundancy involved in the connections of the circuitory

9. function output_vdd_earth

Input	:	void
Returnvalue	:	void
Function	:	It outputs all the transistors connected to earth and vdd respectively.

10. function output_circuit

Input	:	int
Returnvalue	:	void
Function	:	It outputs the circuit input as such by the user, in some prespecified form

11. function output_compressed_circuit

Input	:	void
Returnvalue	:	void
Function	:	It outputs the finalised circuit in some prespecified format after removing redundancies in the connections.

12. function initialise

Input	:	void
Returnvalue	:	void
Function	:	It is a stack routine which initialises the stack

13. function all_paths

Input	:	a pointer to an int
Returnvalue	:	void
Function	:	It helps in finding all possible paths from vdd to ground.

The functions in the file 'stack.c',used in all_paths.

1. function empty

Input	:	void
Returnvalue	:	int
Function	:	returns 1 if the stack is empty, 0 otherwise

2. function pop_stack

Input	:	void
Returnvalue	:	int
Function	:	It pops one item off the stack and returns it.

3. function push_stack

Input	:	int
Returnvalue	:	void
Function	:	It pushes the given item into the stack.

4. function stack_top

Input	:	void
Returnvalue	:	int
Function	:	It returns the top element of the stack.

5. function print_stack

Input	:	void
Returnvalue	:	void
Function	:	It prints all the elements in the stack.

5.3 Conclusions and Future Work

We genaralise the problem for the case of non-series parallel types of circuits at the switch level. We can define the bridges to be those minimal set of transistors in the circuit the removal of which makes the underlying graph(undirected, connected) as series-parallel. In non-series parallel circuits at the switch level, finding out the bridges can be taken as another challenging problem. Similiarly implementing ROBDD's from the boolean expression (function) can be taken as another problem. Methods of constructing ROBDD from the function are known. First we try constructing BDD from the funtion using shannon's decomposition recursively. From there, we can reduce it further, by applying specific set of rules. Implementation issues involved in this direction is an open problem.

Chapter 6

REFERENCES

1. R.E.Bryant: Graph-based algorithms for boolean function manipulation , IEEE Trans. comput. c-35,6(Aug.), 677-691, 1986.
2. R.E.Bryant : Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, ACM computing surveys, vol.24, no.3(sep.), 293-318,1992
3. 32nd Design Automation Conference Proceedings Sanfrancisco, CA Mosco Center, june 12-16, 1995.
4. VLSI Design Conference-1996,Bangalore.Tutorial. Register transfer level synthesis: from theory to practice. By K.Keutzer and S.Malik.
5. Algorithms For VLSI Physical Design Automation-Naveed Sherwani.

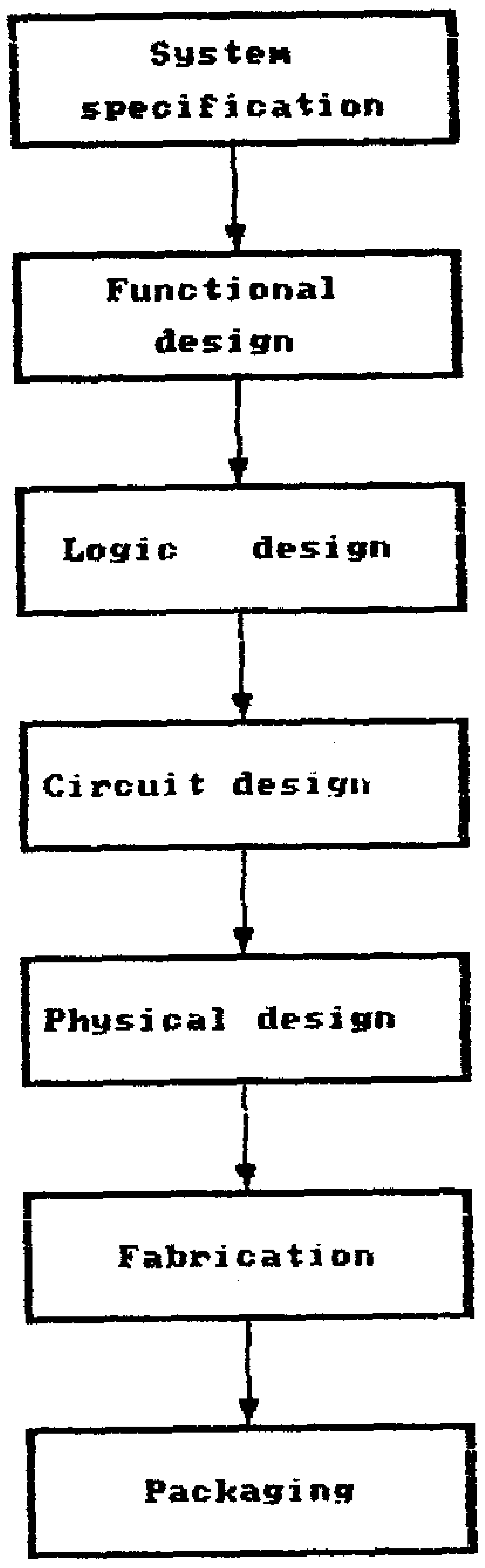
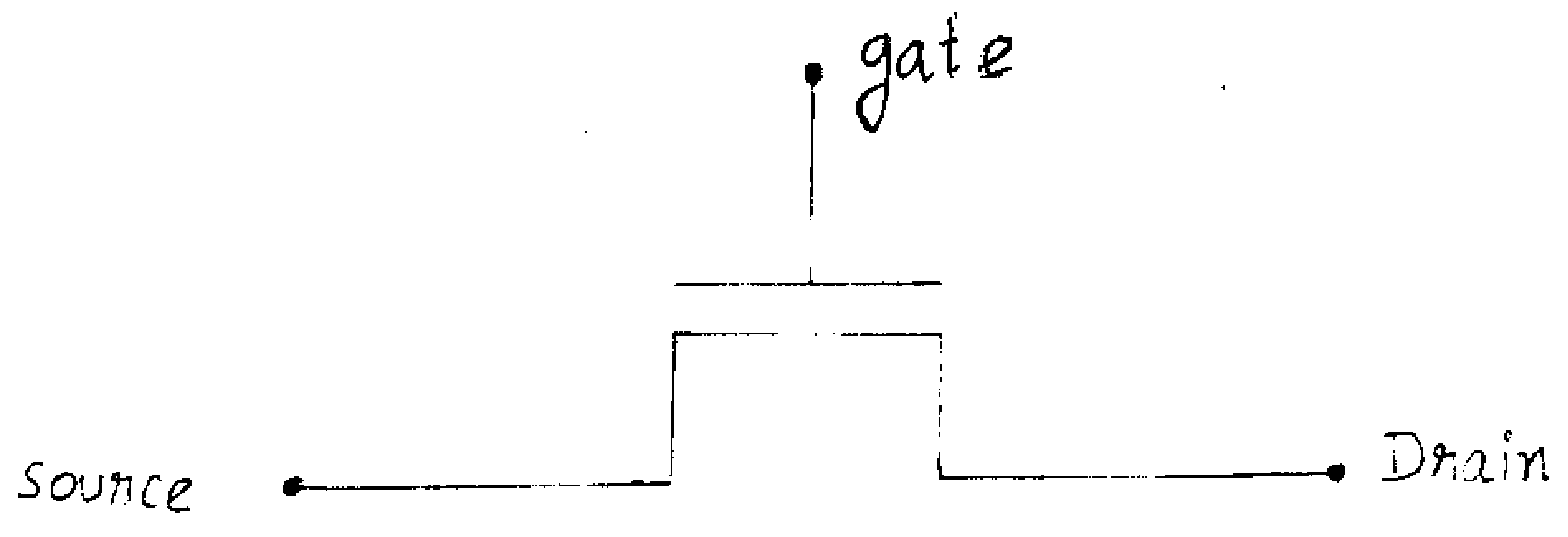
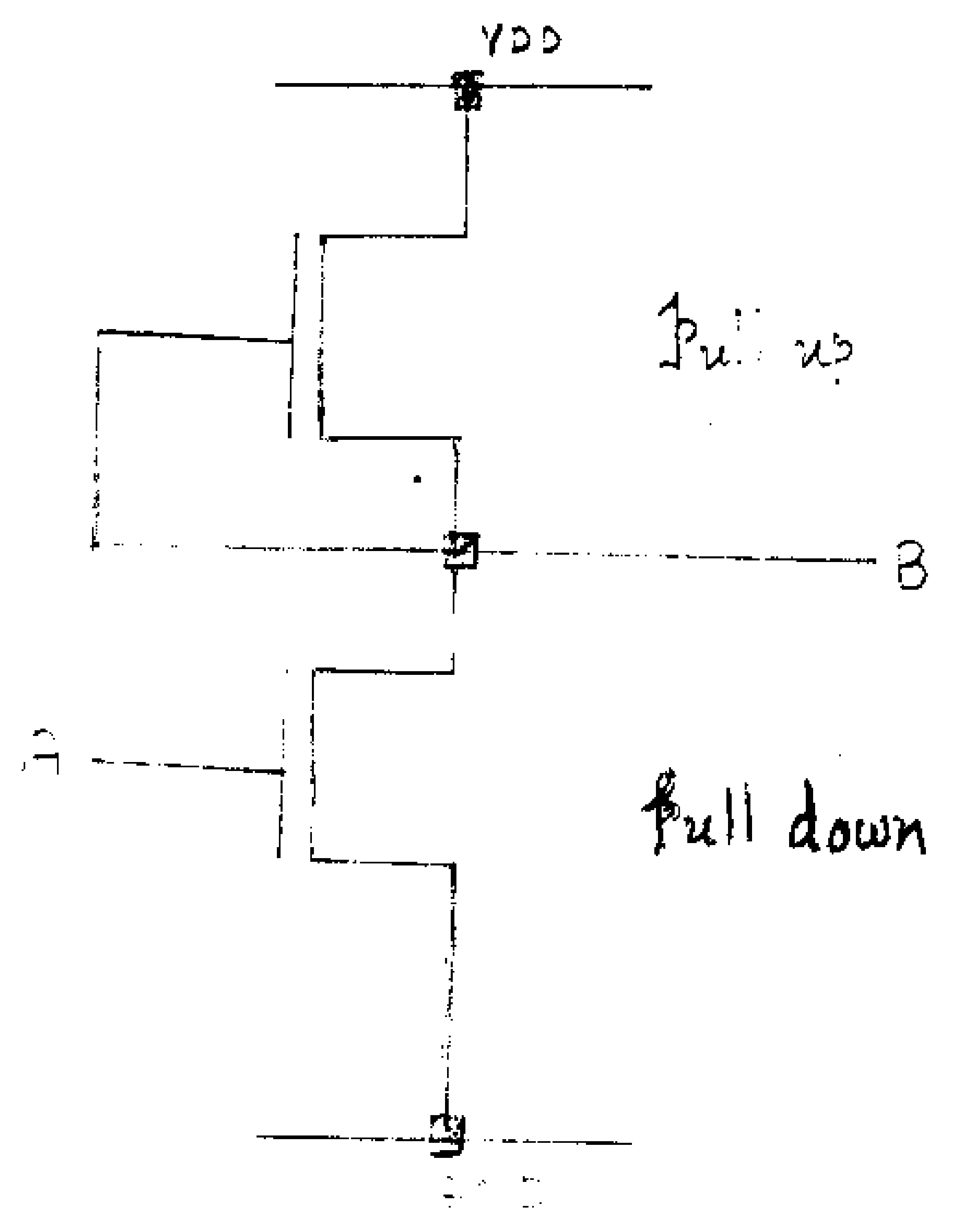


DIAGRAM 1.1 : VLSI DESIGN CYCLE

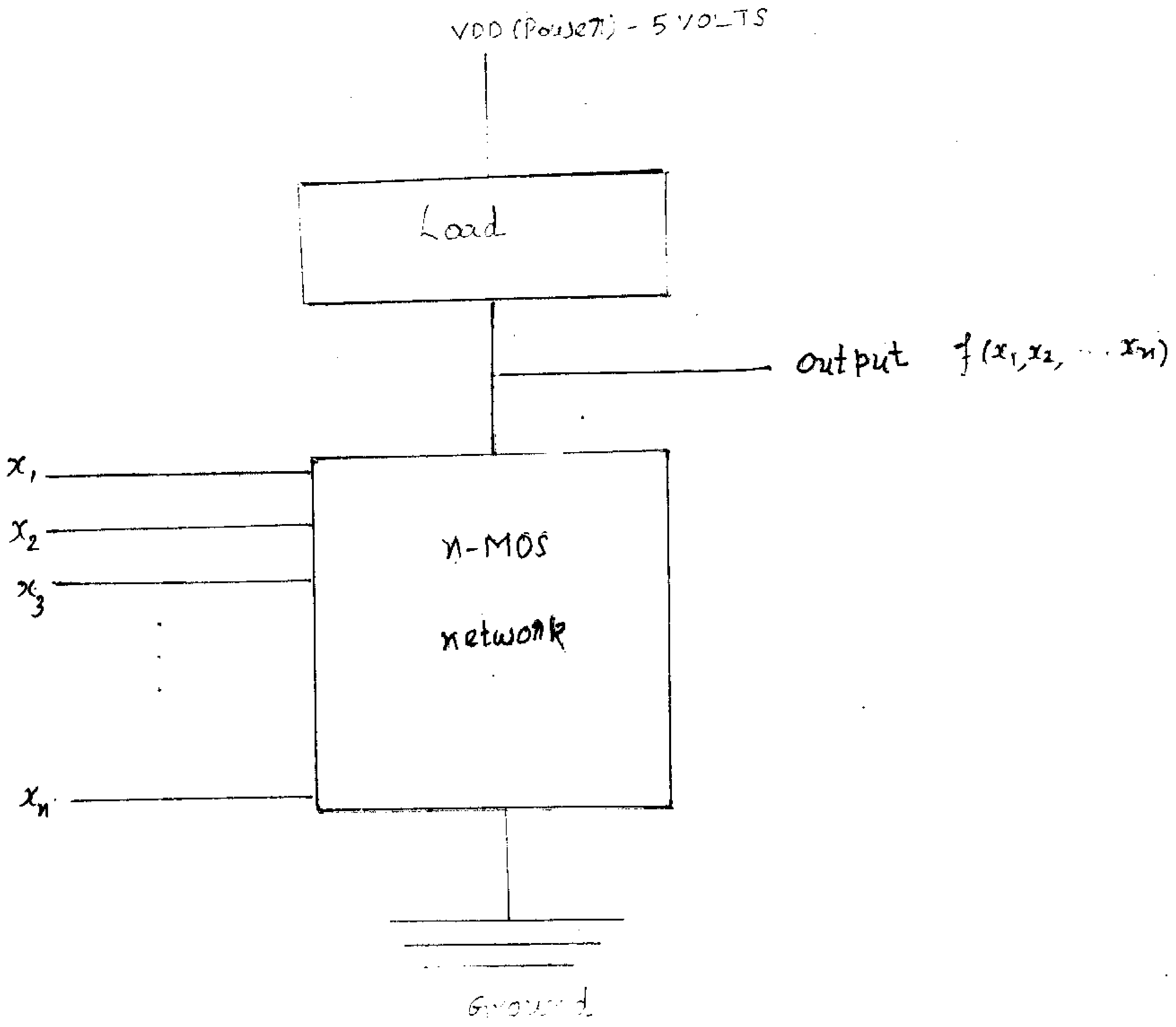




A	B
0	1
1	0

Diagram 1.2 (Inverter)

Diagram 1·3



VDD (Power) - 5 VOLTS
Load
output $f(x_1, x_2, \ldots x_n)$
x_1
x_2
x_3
x_n
n-MOS network
Ground

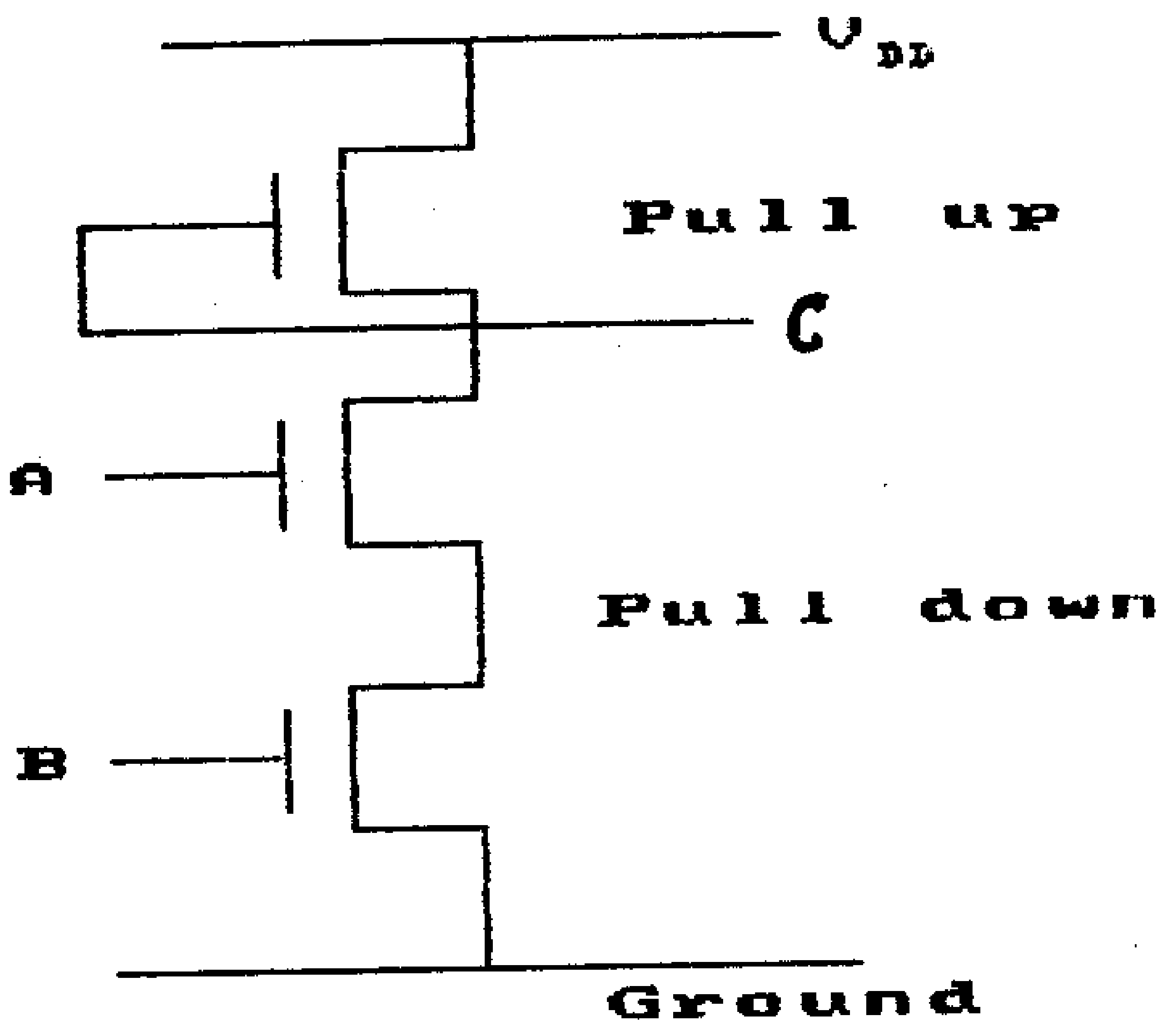
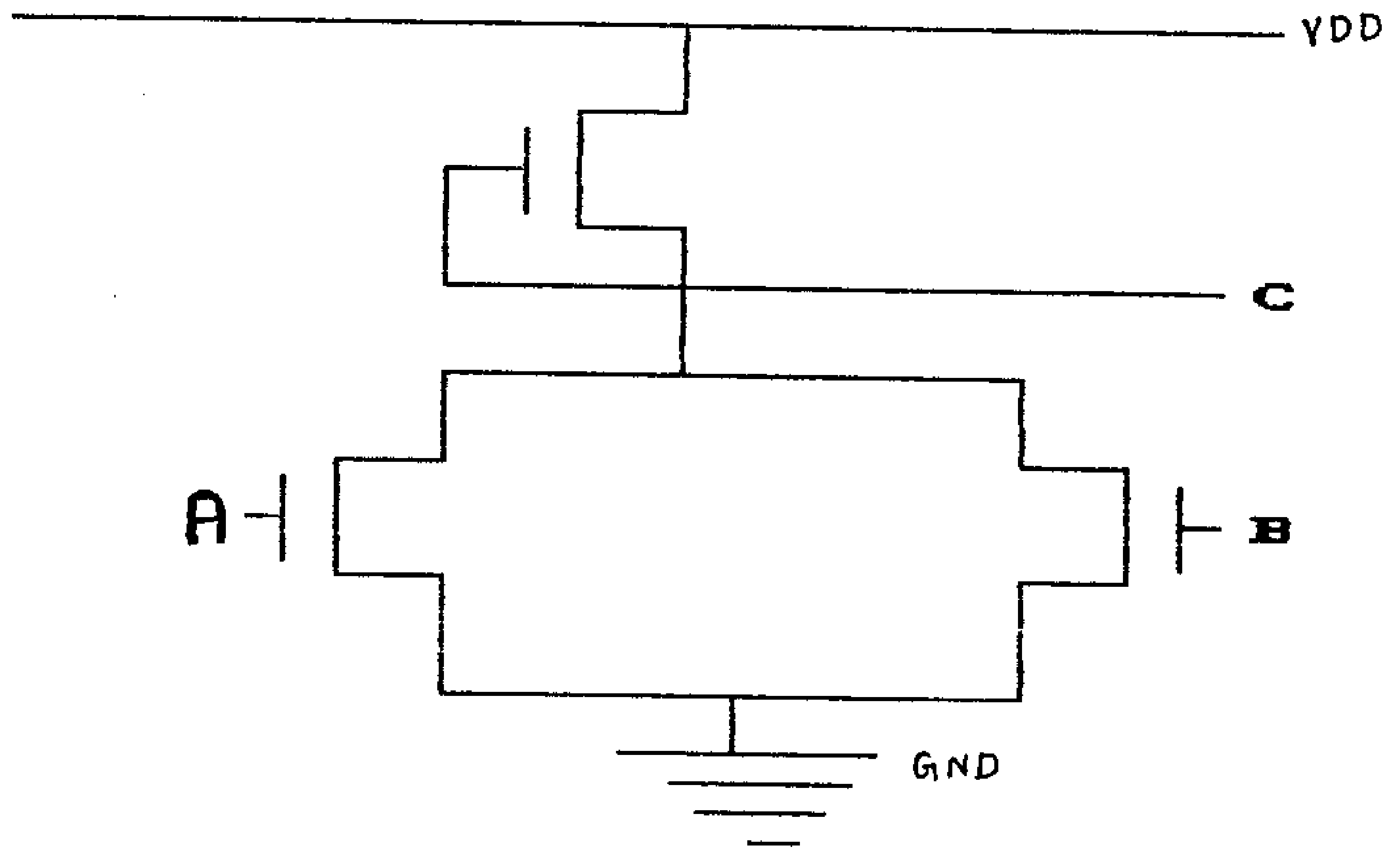


fig nMOS NAND gate

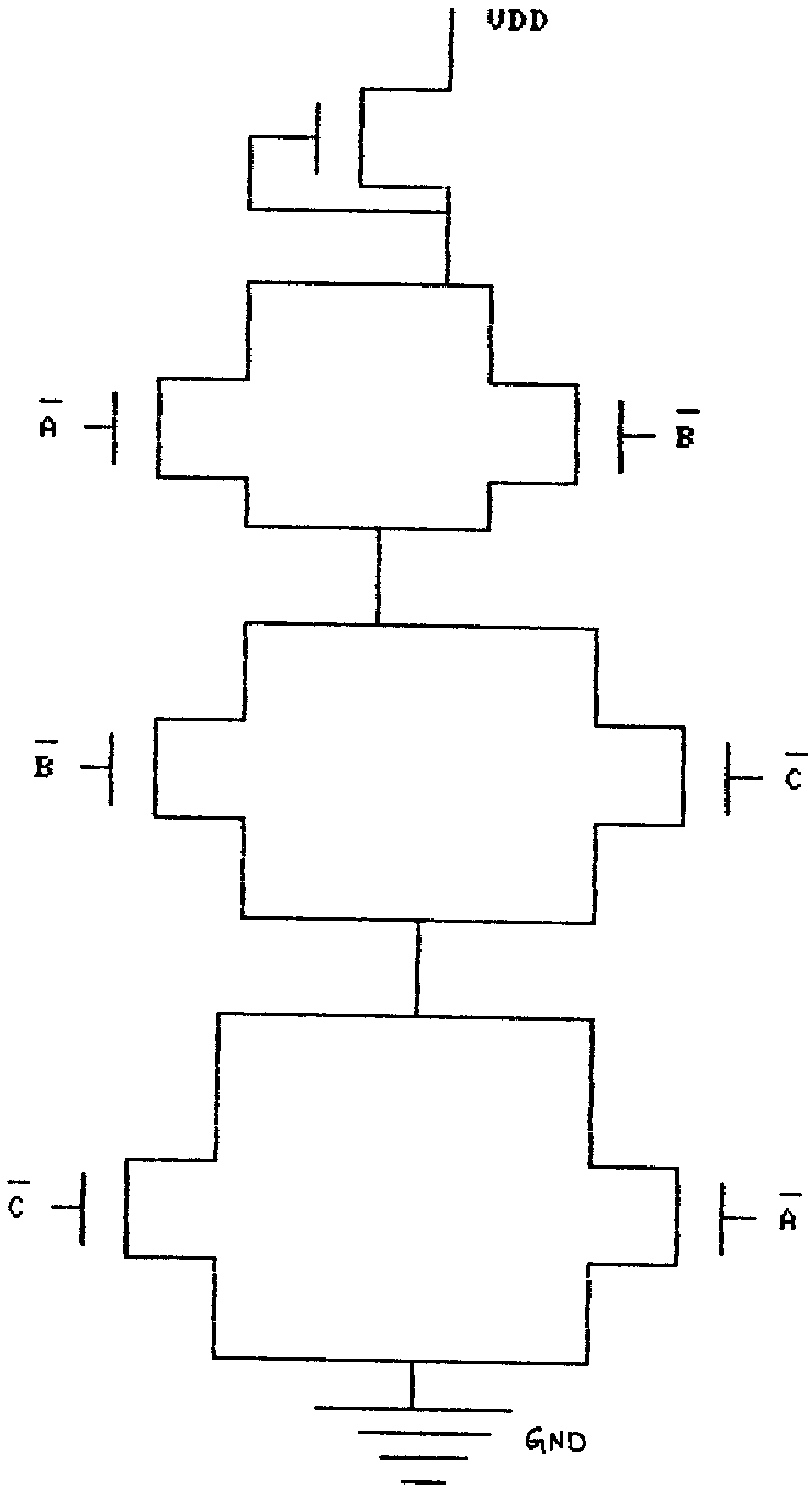
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

DIAGRAM 1·4a



A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

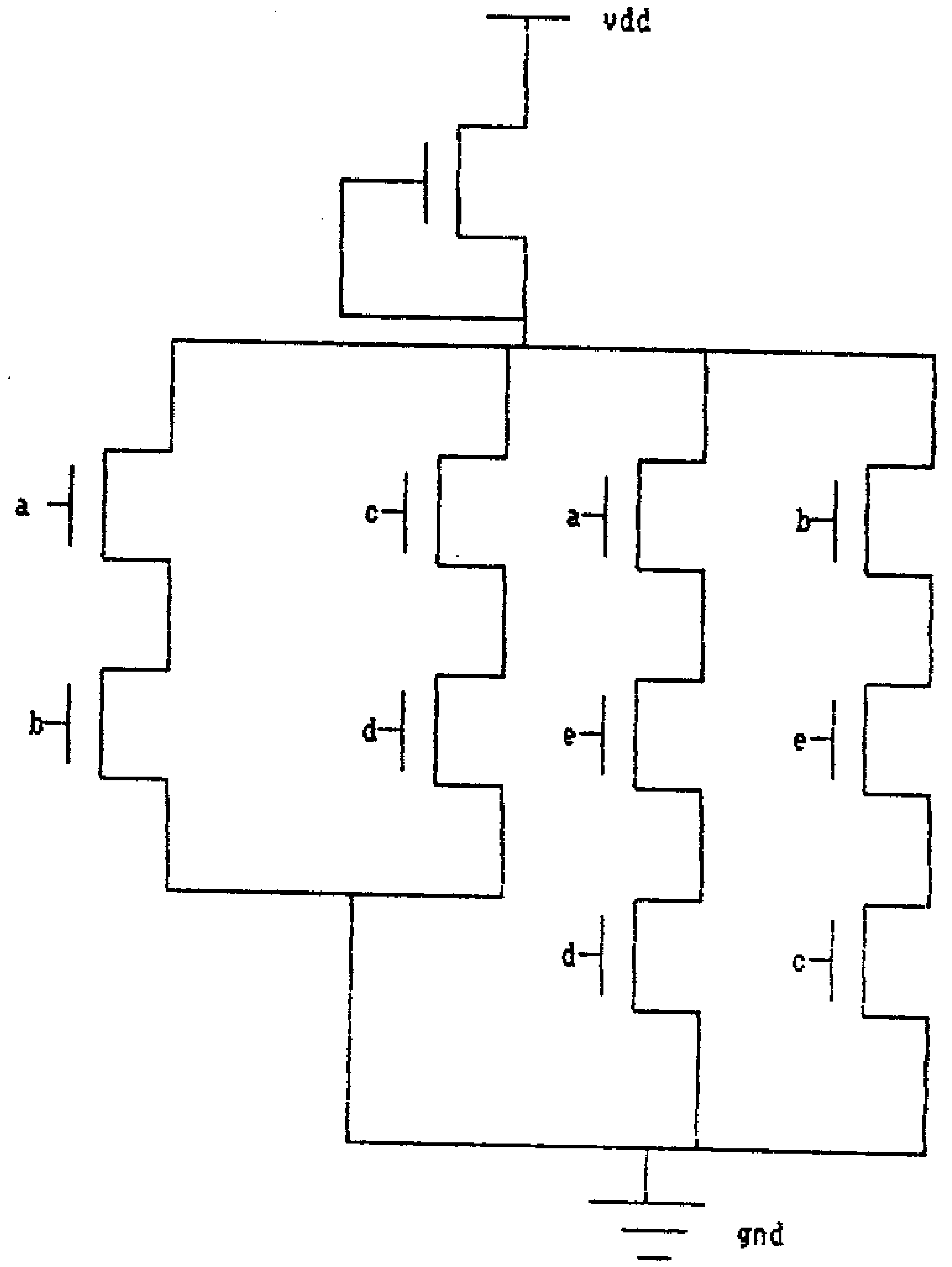
DIAGRAM 1·4 b



$$f=\overline{(\bar{A}+\bar{B})(\bar{B}+\bar{C})(\bar{C}+\bar{A})}$$

$=AB+BC+CA$ By DeMorgan's law.

DIAGRAM 1·5



$f=\overline{ab+cd+aed+bec}$

Total cost= 10+1=11 transistors.

DIAGRAM 1·6 a

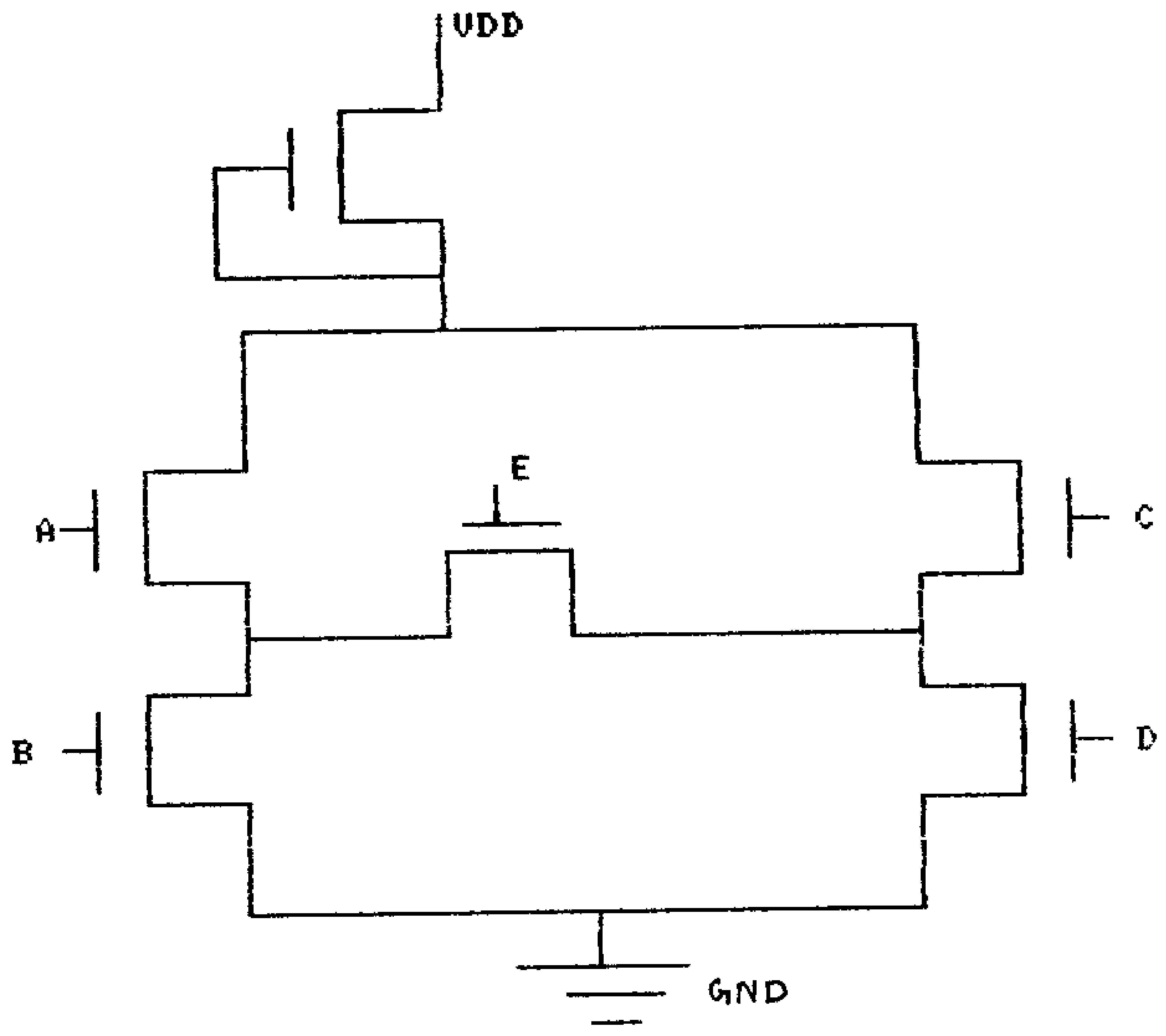
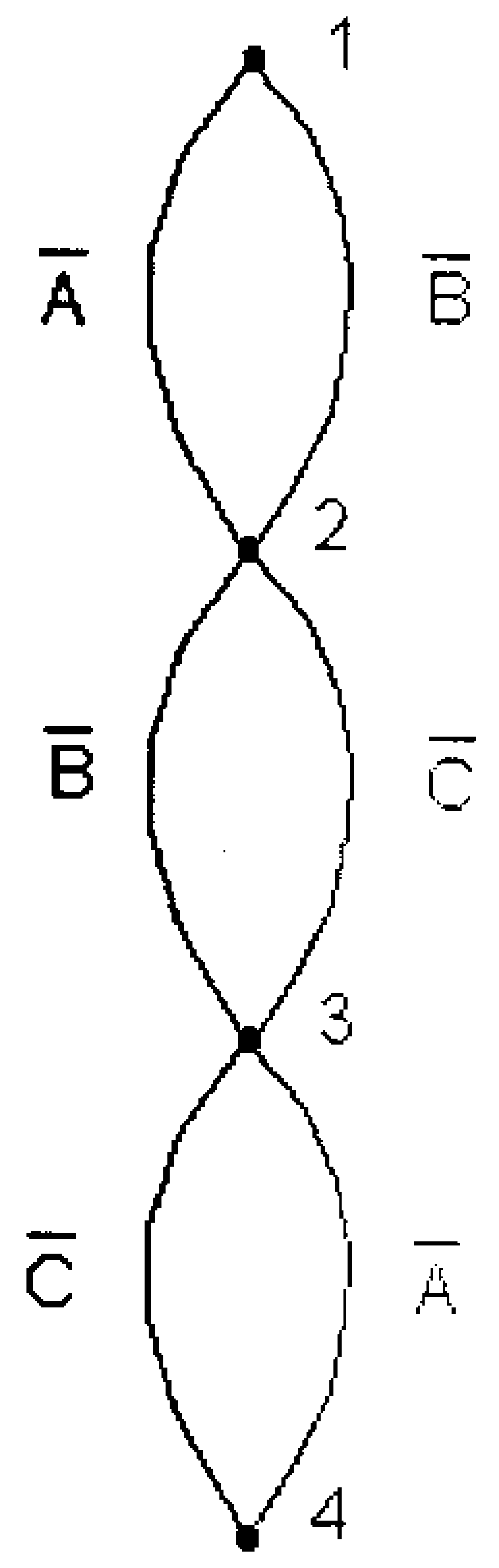


DIAGRAM 1·6b

Corresponding graph (refer diagram 1.5)



Corresponding graph (refer diagram 1.6)

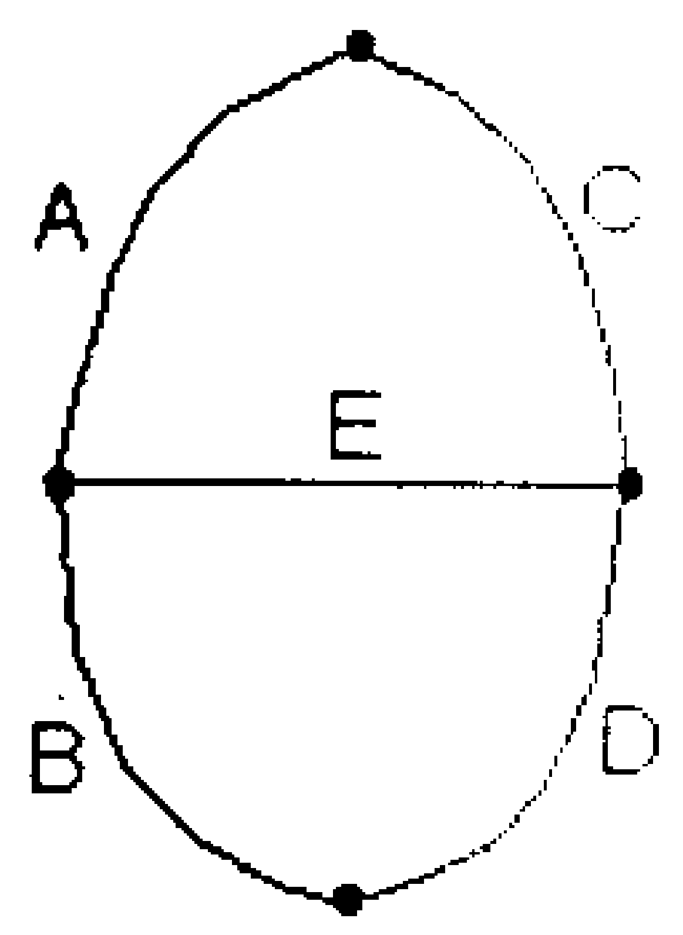
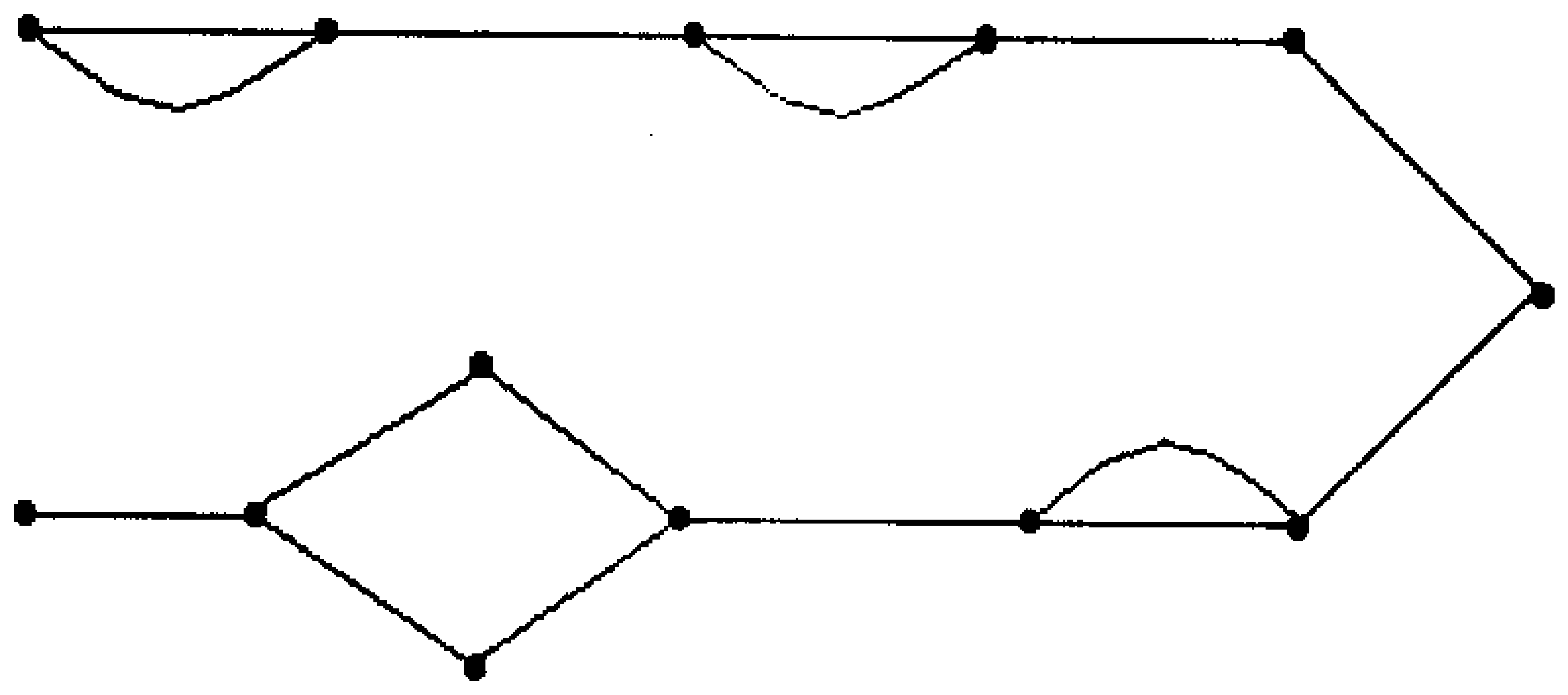


Diagram 2.1

Series parallel structure



Non series parallel structure

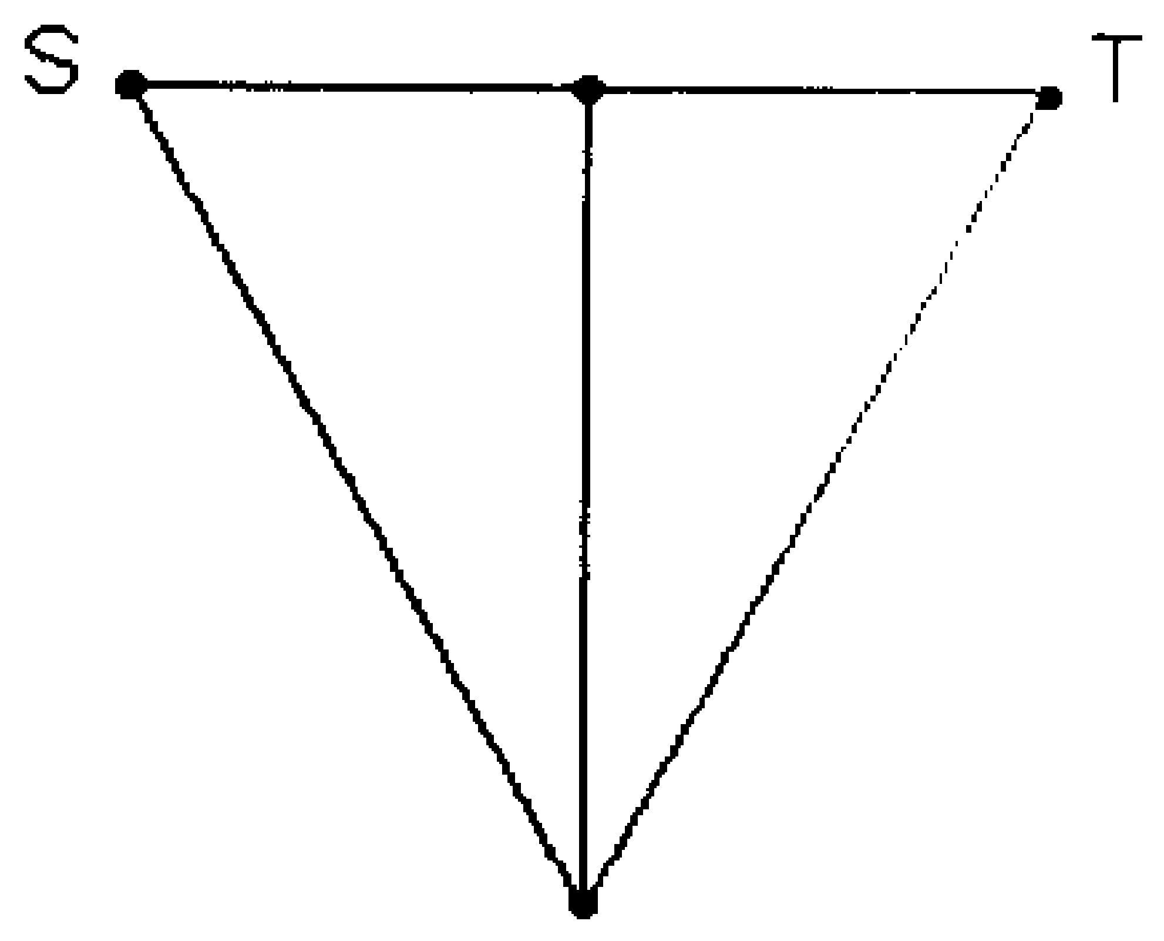


Diagram 2.2

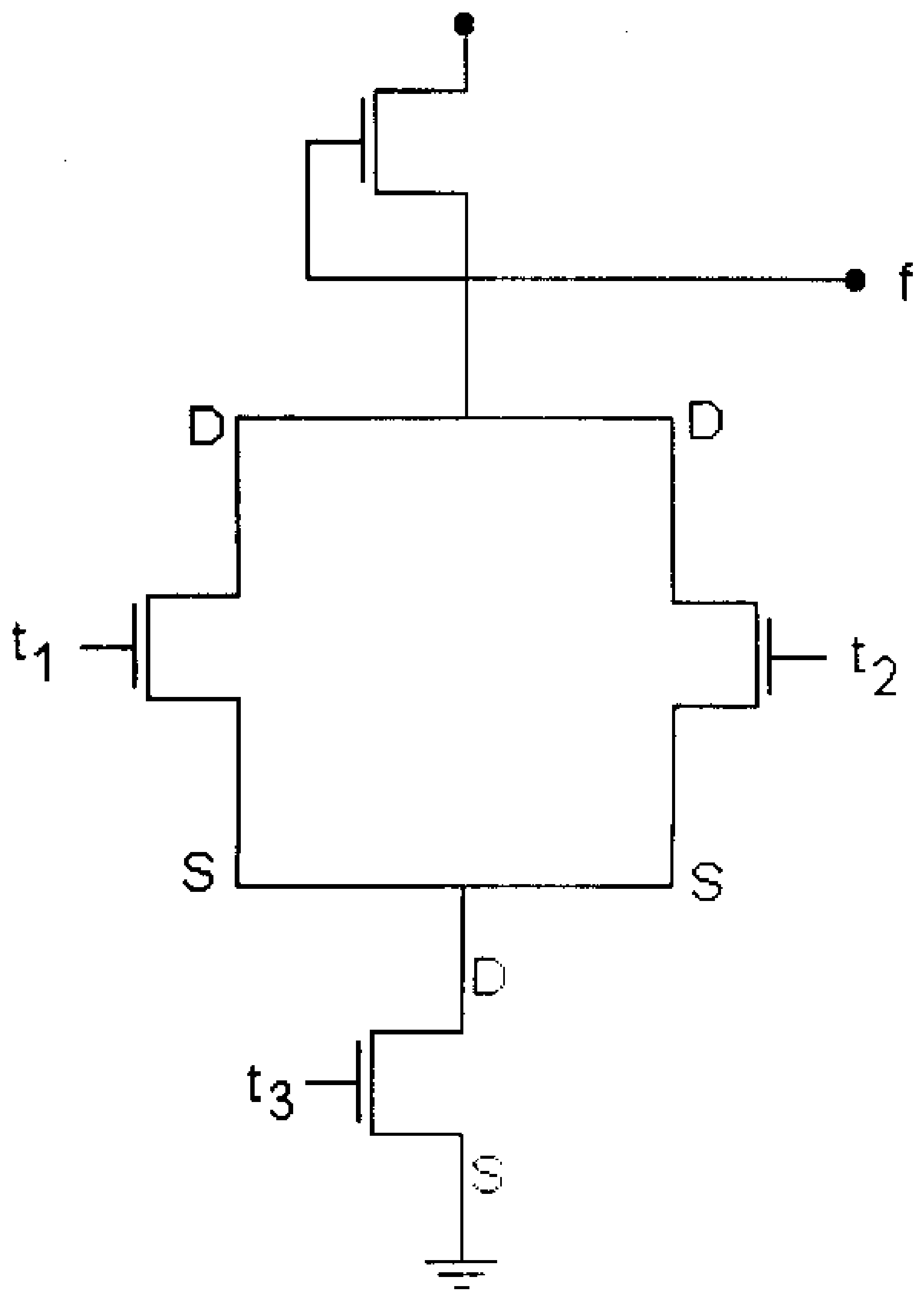


Diagram 3.1

Output

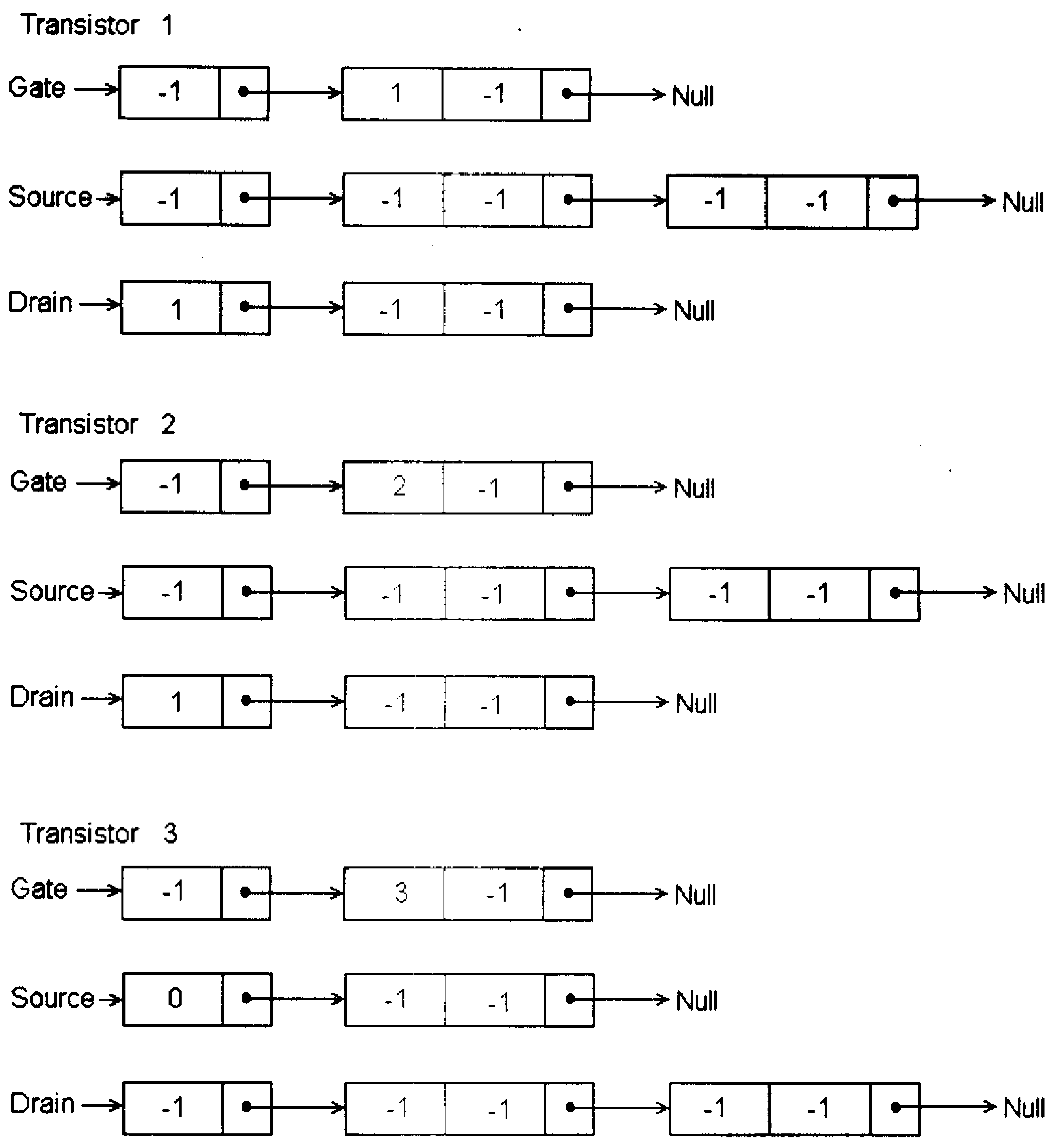


Diagram 3.2

Output

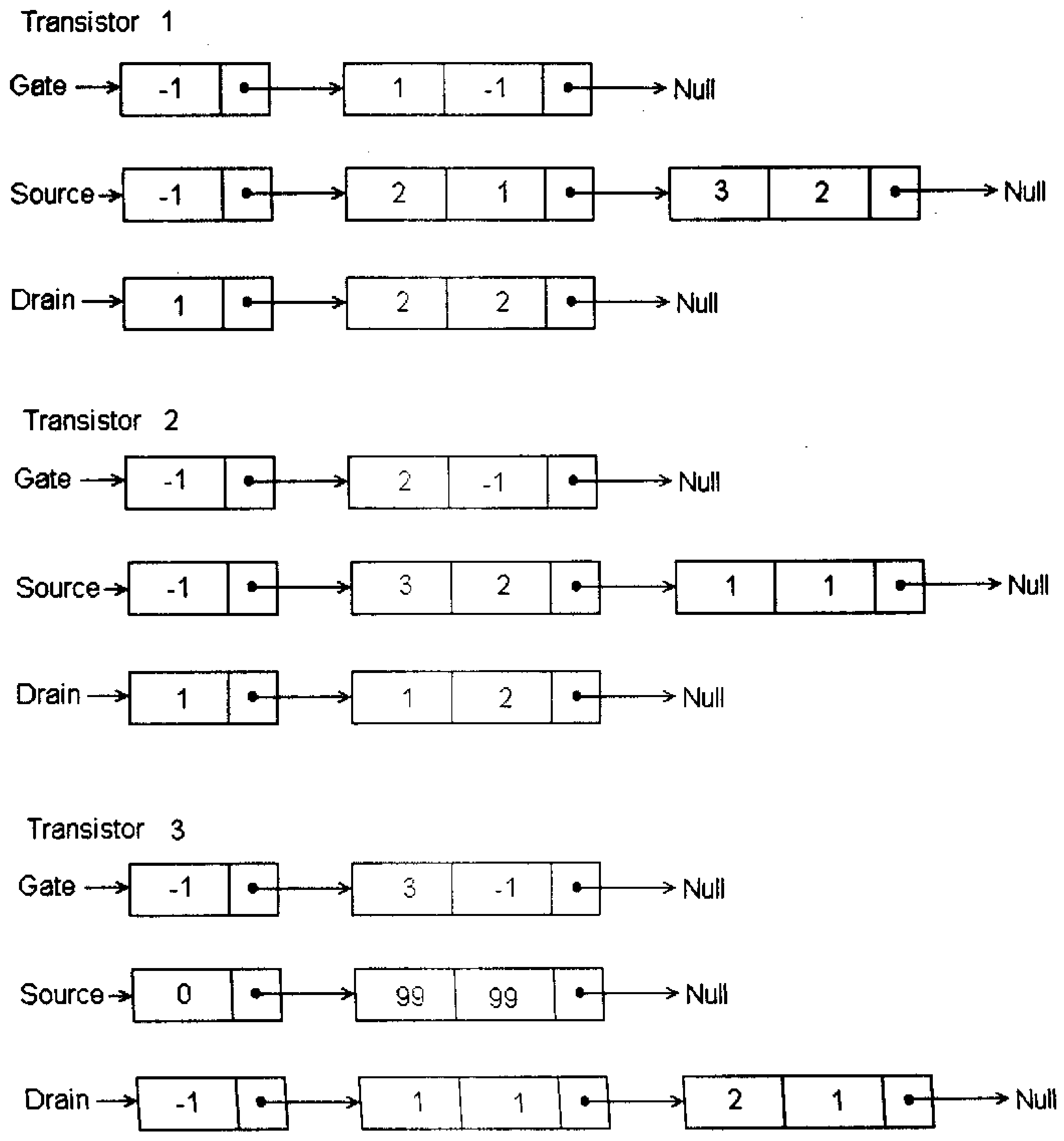


Diagram 3.3

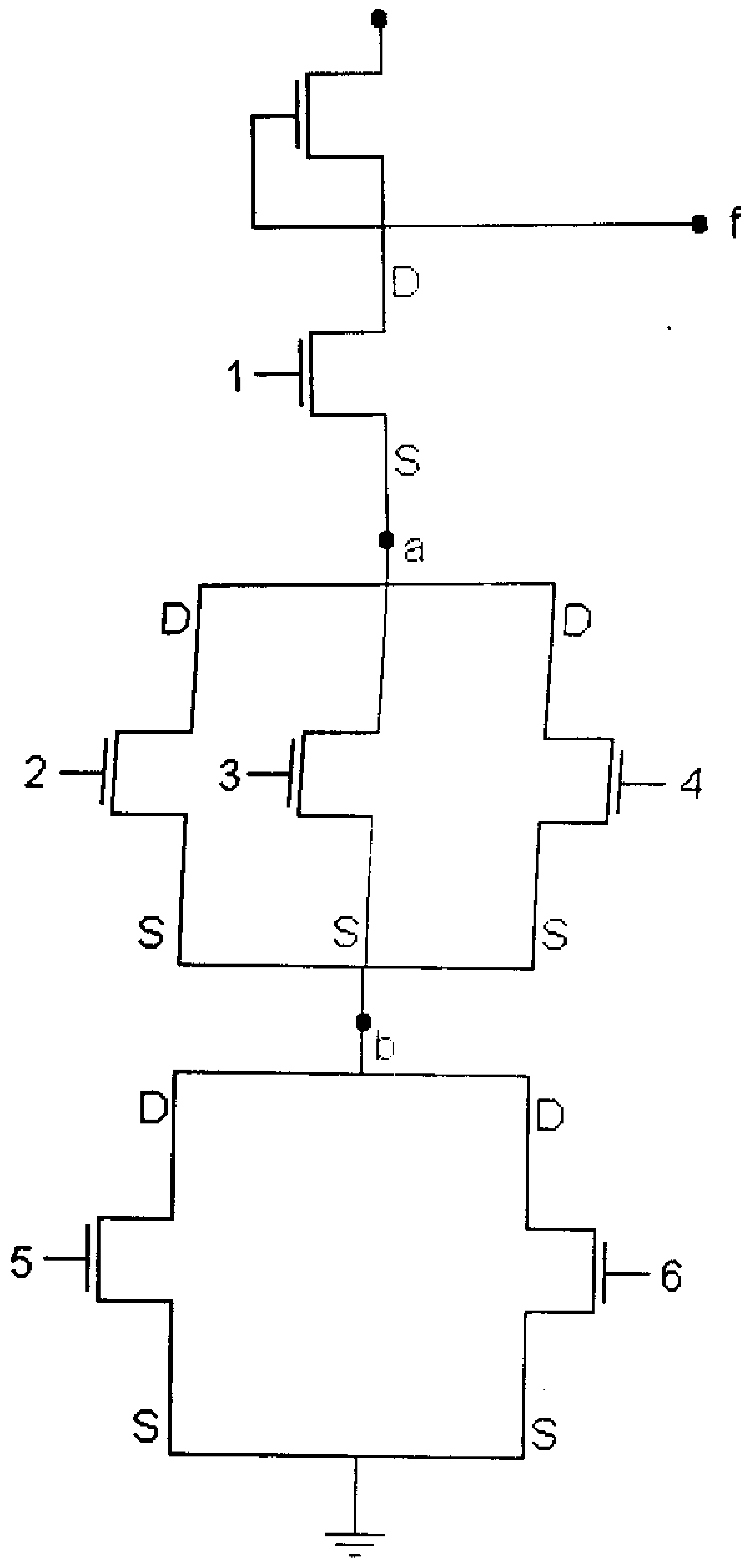


Diagram 3.4

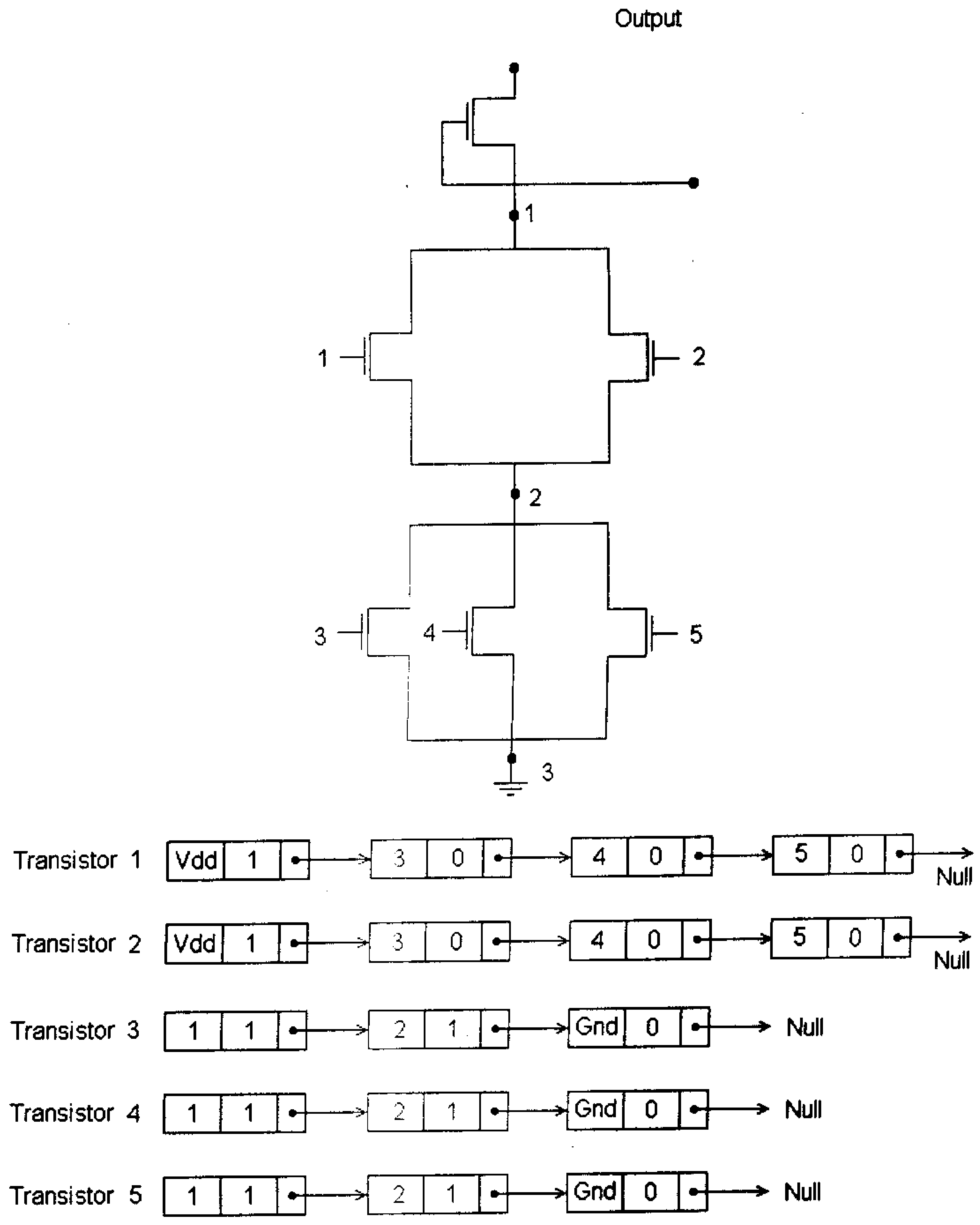
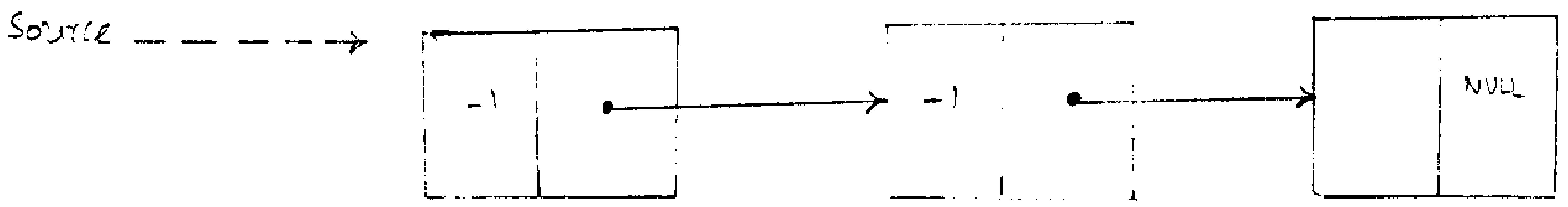

Output
1
1
2
2
3
4
5
3
Transistor 1: Vdd 1 → 3 0 → 4 0 → 5 0 → Null
Transistor 2: Vdd 1 → 3 0 → 4 0 → 5 0 → Null
Transistor 3: 1 1 → 2 1 → Gnd 0 → Null
Transistor 4: 1 1 → 2 1 → Gnd 0 → Null
Transistor 5: 1 1 → 2 1 → Gnd 0 → Null

Diagram 3.5

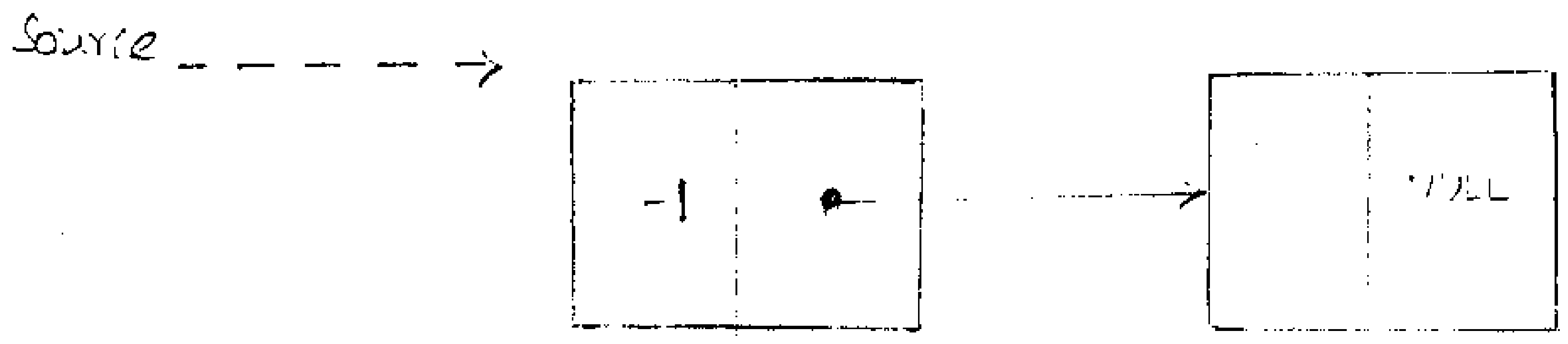
Output

Initial Configuaration of the Circuit

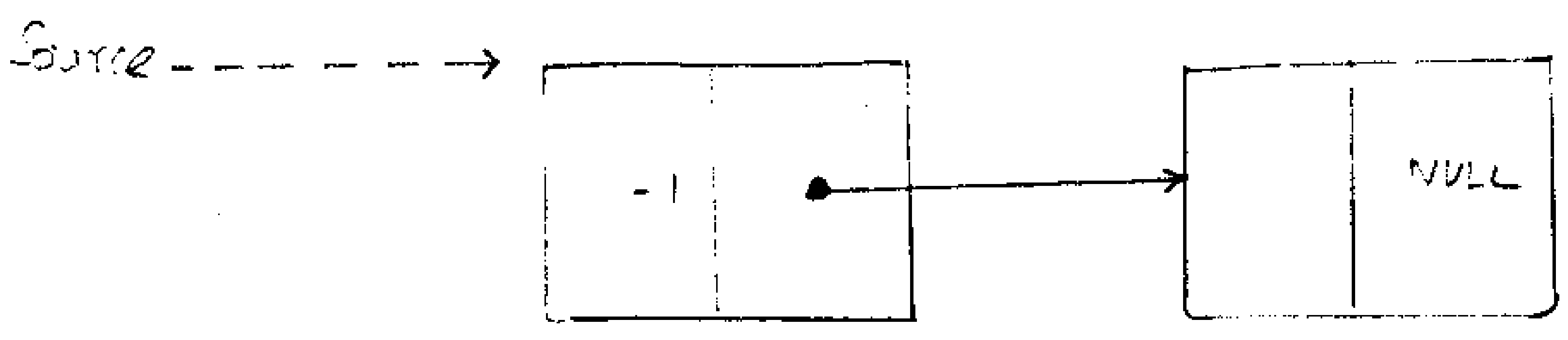
transistor 1



transistor 2



transistor 3



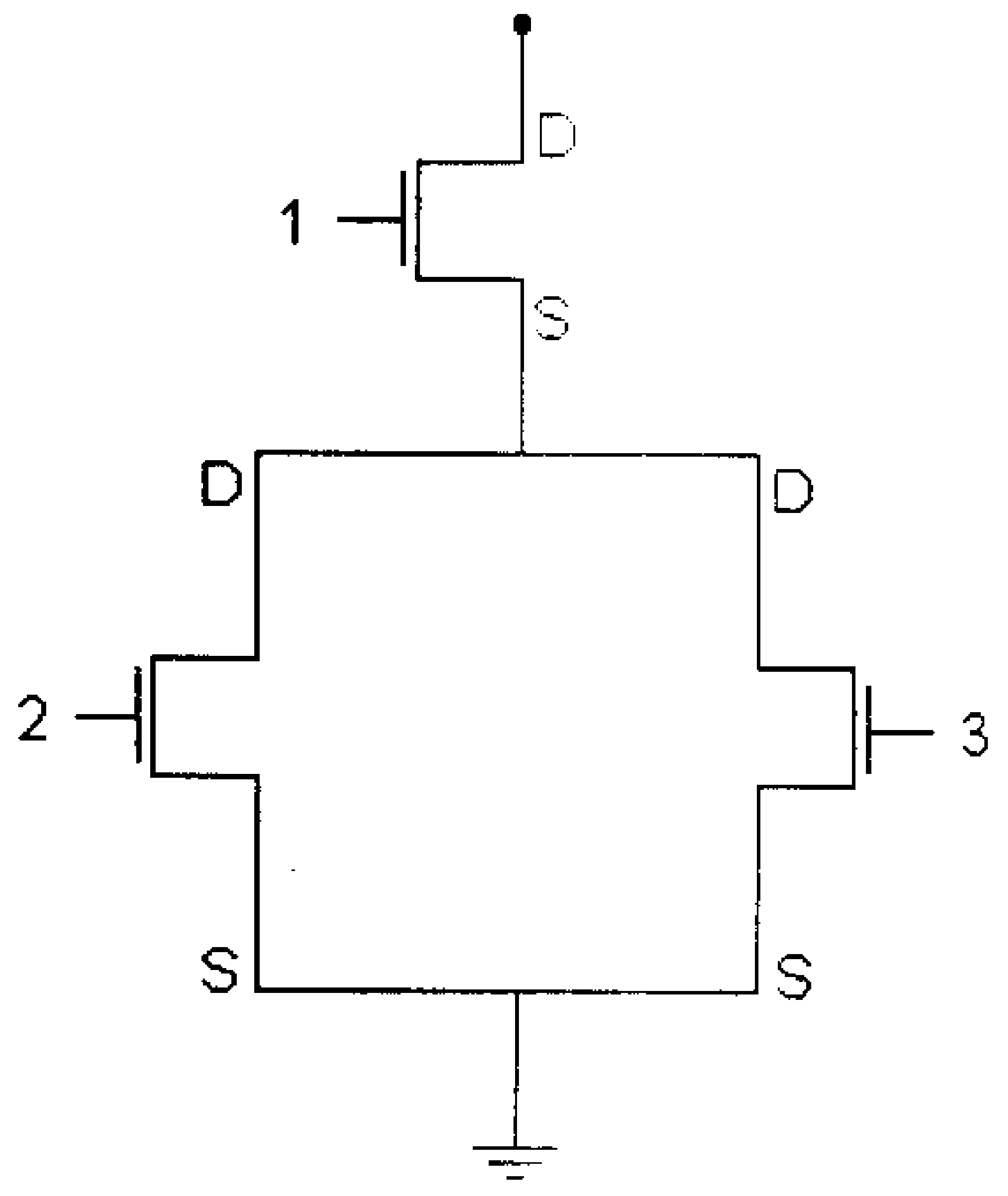
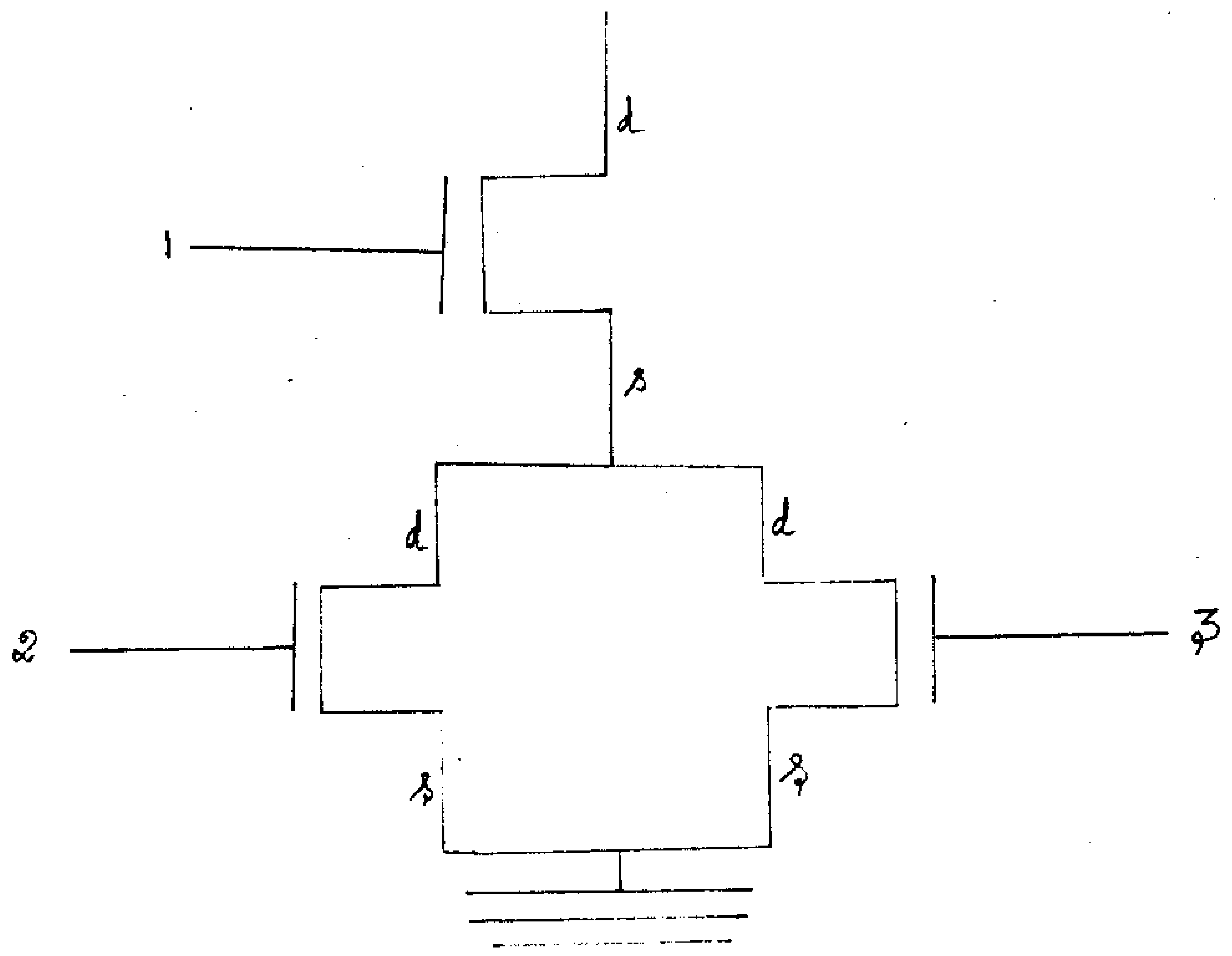
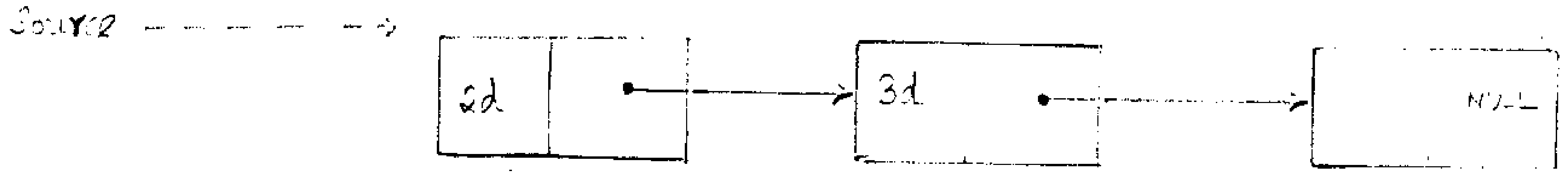


Diagram 3.7

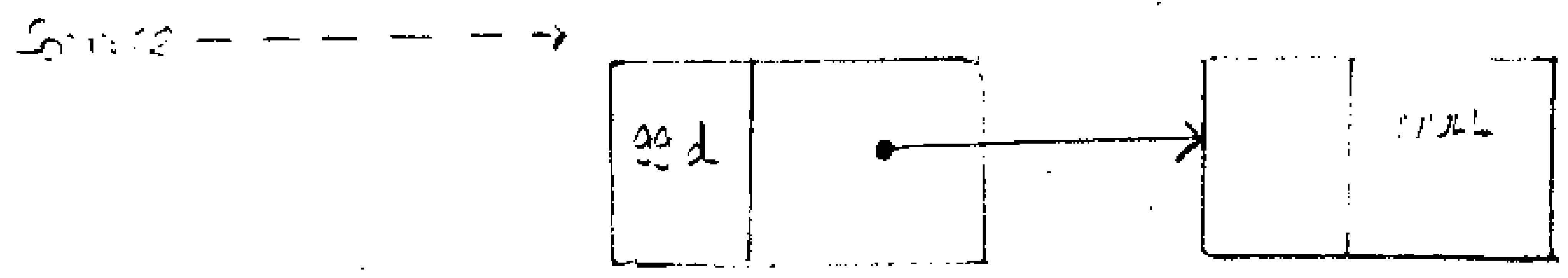
Diagram 3.7a



transistor 1



transistor 2



transistor 3

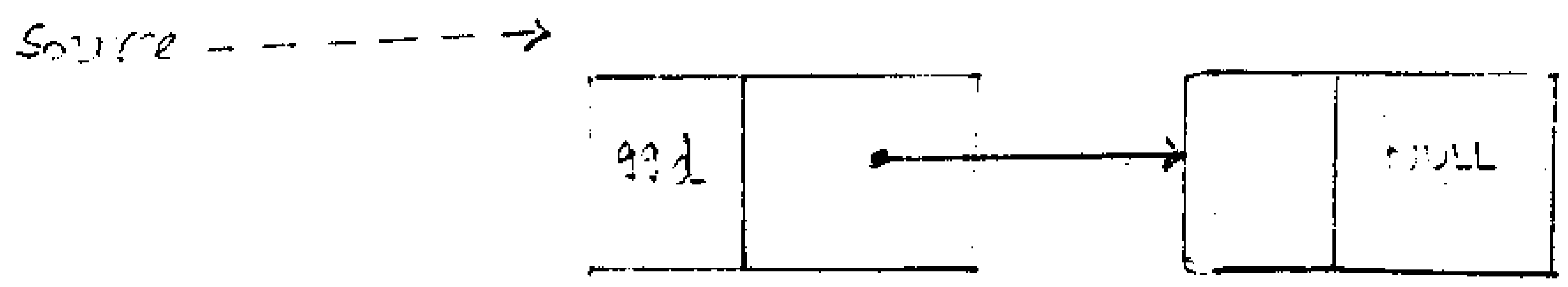
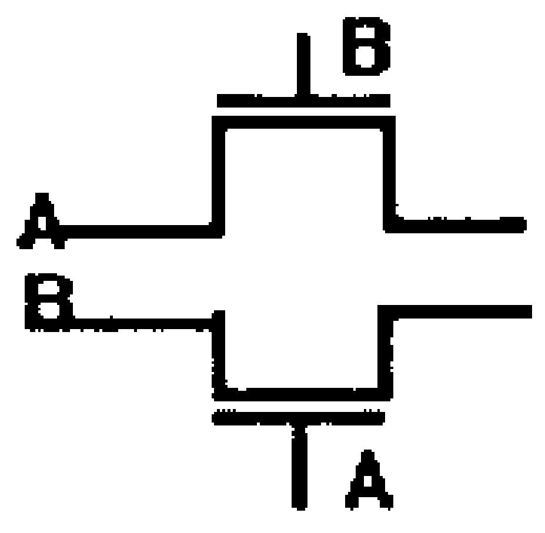


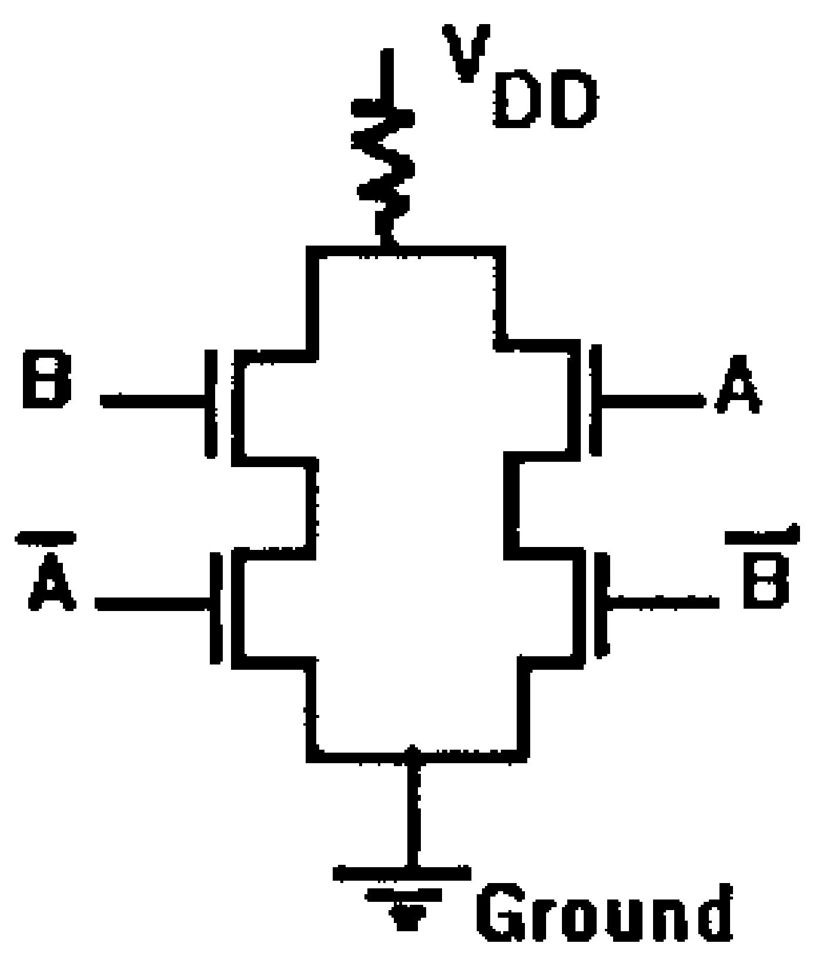
Fig 4.1



Circuit with Passmode Transistor

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

$f = AB + \bar{A}\bar{B}$



Corresponding Series-Parallel Circuit

$f = \overline{\bar{A}B + A\bar{B}}$

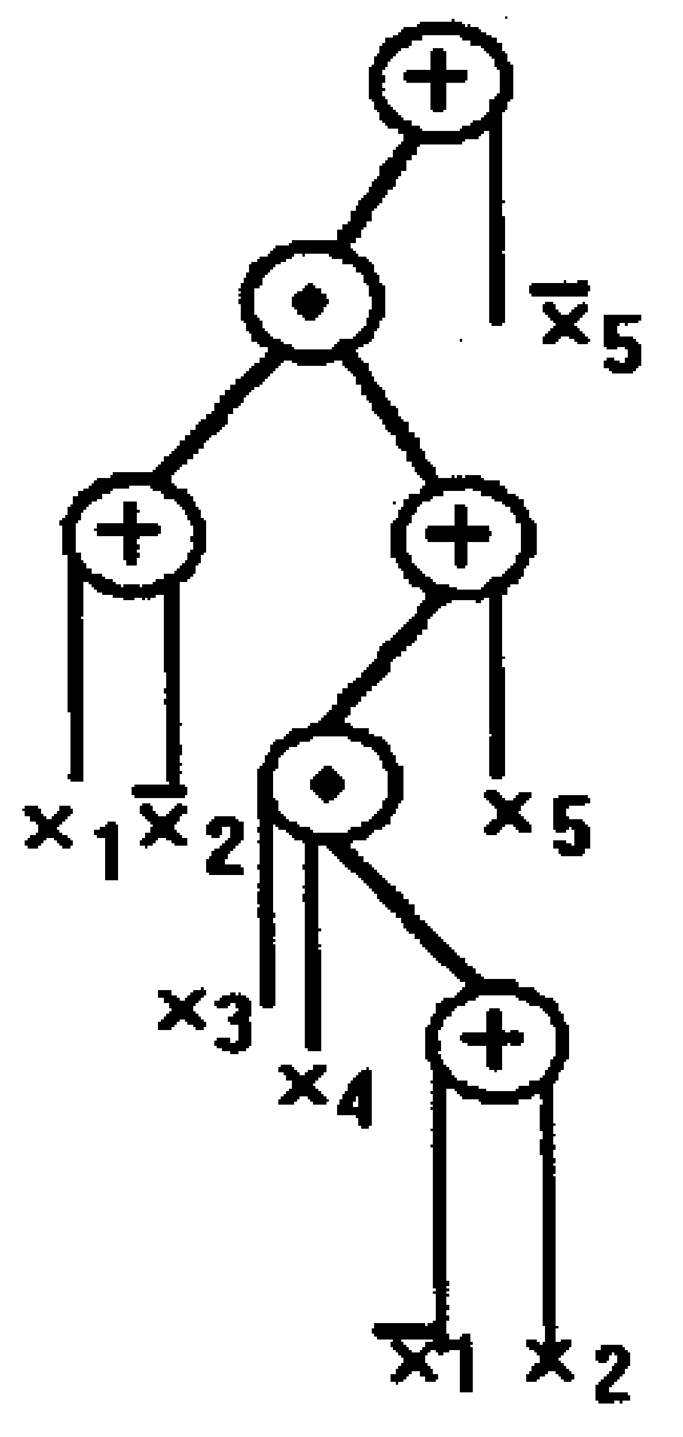
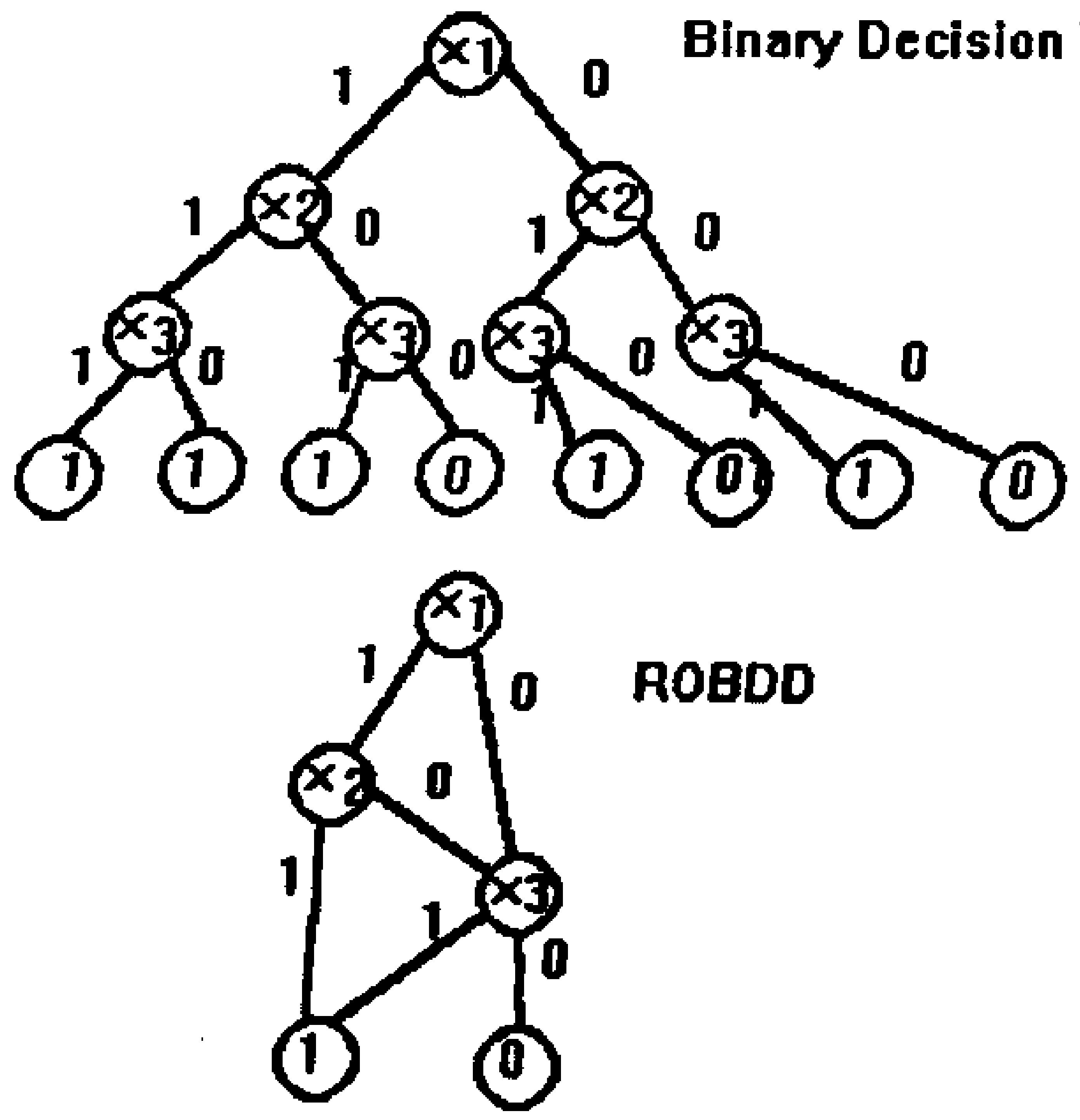


Fig 4.2

factored leaf-DAG for

$((\overline{x}_1 + x_2).x_3x_4 + x_5).(x_1 + \overline{x}_2) + \overline{x}_5$



STACK STATUS

Input Adjacency List

start$\rightarrow$ 1

$1 \rightarrow 2, 3$

$2 \rightarrow 4$

$3 \rightarrow 5$

$4 \rightarrow 99$

$5 \rightarrow 99$

Diagram 4.4a

STACK

```
-1 -1
-1 -1 1
-1 -1 1 2
-1 -1 1 2 4
-1 -1 1 2 4 99
-1 -1 1 2 4
-1 -1 1 2
-1 -1 1
-1 -1 1 3
-1 -1 1 3 5
-1 -1 1 3 5 99
-1 -1 1 3 5
-1 -1 1 3
-1 -1 1
-1 -1
```

Diagram 4.4b

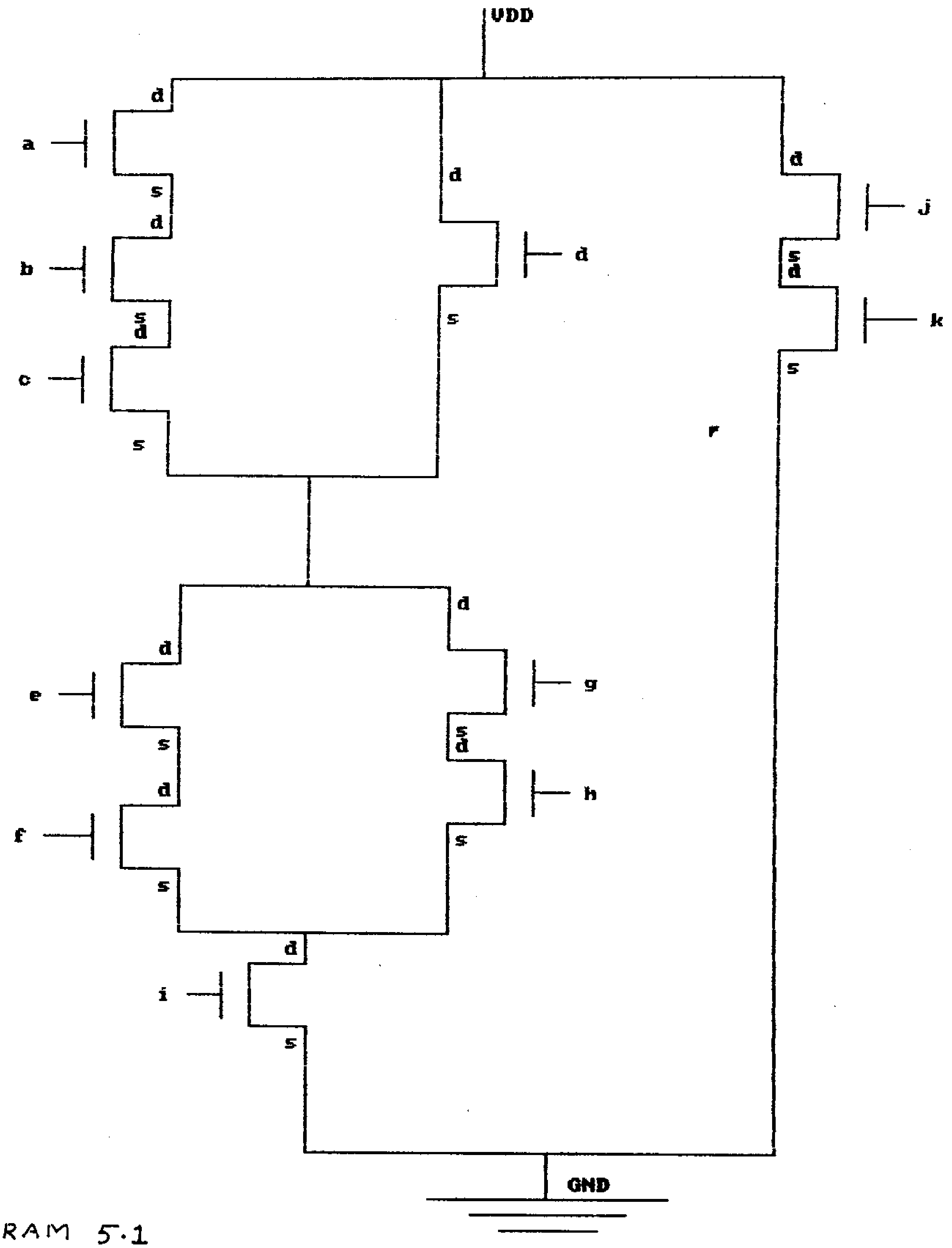


DIAGRAM 5·1

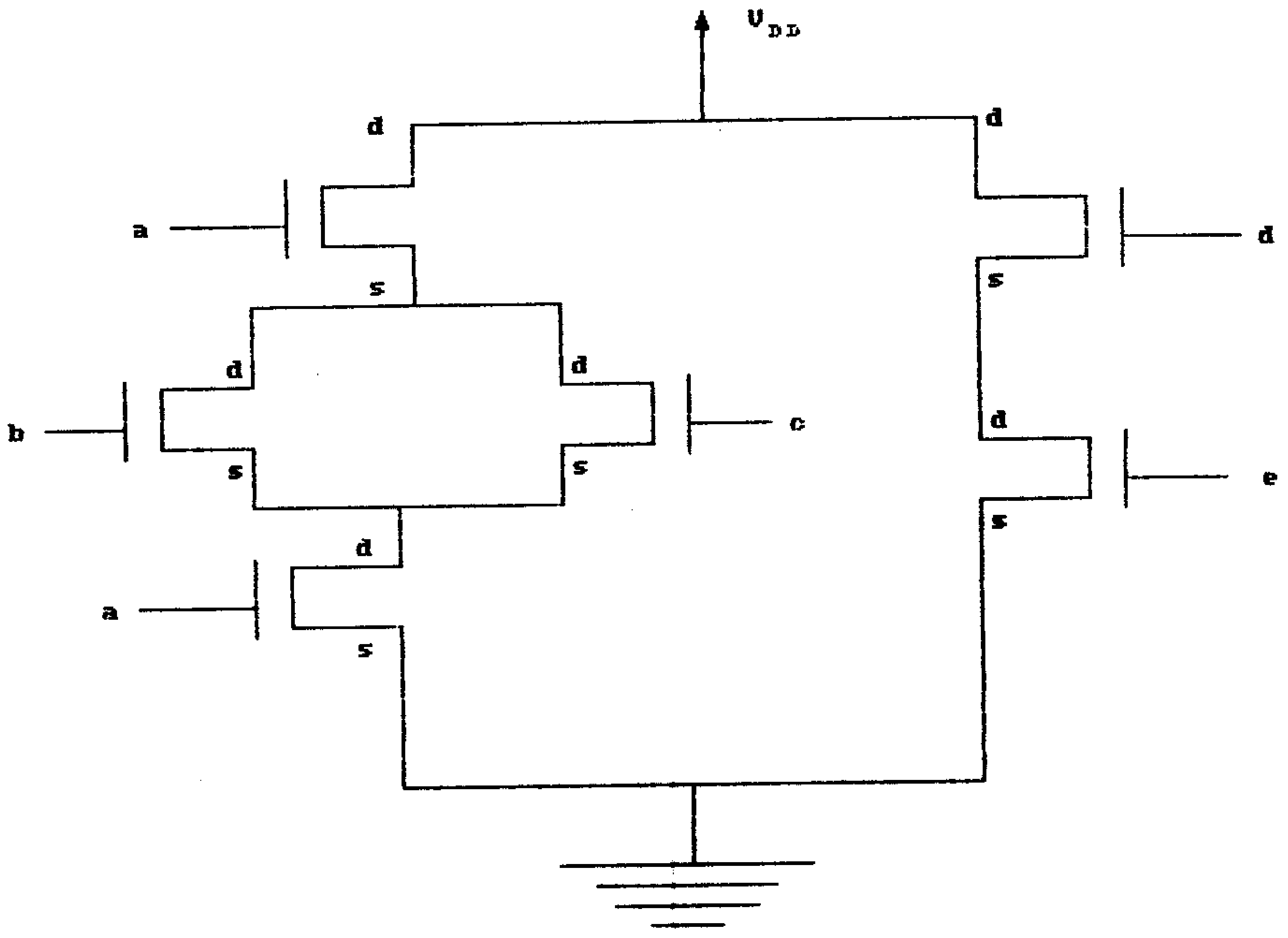


DIAGRAM 5.2 : Number of Transistors = 6