INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE CENTRATOM TRIESTE

The United Nations University

SMR/748 - 7

ICTP-INFN-UNU-MICROPROCESSOR LABORATORY
THIRD COURSE ON BASIC VLSI DESIGN TECHNIQUES
21 November - 16 December 1994

HIGH LEVEL LOGIC DESIGN METHODOLOGY

Magali ESTRADA
Facultad de Fisica
Universidad de La Habana
San Lazaro y L - Vedado
C.P. 10400
Havana
Cuba

HIGH LEVEL LOGIC DESIGN METHODOLOGY

Top-down + bottom-up design

top-down: the design starts with the specification of the complete object of design in a compact form (global specification).

It is structured: global representation is divided into sub-systems + their interconnection.

Sub-systems can be divided again until you represent the whole design by its simplest elements.

The designer must have means for:

- entering the description
- simulating at different levels
- synthesis and analysis
- verification of interfaces at different levels
- generation of tests
- generation of topology

HIGH LEVEL DESIGN

The High-level design methodology requires:

abstraction – to conceive a global and then enter into its details;

formalism – to have the rules and procedures for working at each level;

concepts – so that everybody understands the other;

It consists of a hierarchical structure of levels.

One can represent a design at any of these levels.

Each level of abstraction can be described by a:

a.– structured domain: in terms of the interconnection of more primitive components.

b.– behavioral domain: by means of its input/output response.

Level	Structural primitives	behavioral representation
system	CPU, memories, ports	performance.
chip (RTL)	processors, memories ports, etc.	I/O response, algorithms, RTLanguage.
regis-ters	registers, counters, ALU, flip-flops, com-binational logic, etc.	Truth tables, state tables.
gates	gates, multiplexers, flip-flops, counters, adders, decoders, etc.	Boolean equations.
circuit	transistors, capacitors, etc.	Differential equations.
silicon	topology	

Fig. 1.1

TRADITIONAL SCHEMATIC DESIGN (no synthesis)

1. Architecture is specified and partitioned into functional blocks.
2. Schematic entry and further partitioning.
3. Gate-level simulation and analysis for monitoring function, performance and area.
4. Integration of blocks and verification of the interface between them.
5. Complete design validation by gate-level simulation.
6. Placing and routing.
7. Final gate level verification.

SYNTHESIS: Physical realization of a design starting from its description.

ANALYSIS: The inverse process of synthesis.

VHDL: The Hardware Description Language used for Very High Speed Integrated Circuits.

TRADITIONAL DESIGN + PARTIAL HDL ENTRY
(with synthesis)

1. Architecture is specified and partitioned into functional blocks.
2. Part of the design is captured using schematic entry.
3. Capture of the rest by: netlist/schematics; Boolean equations, state tables, VHDL, and Verilog; synthesis automatically creates the schematic. Applicable to state machines and control logic.
4. Function, performance, and area are monitored.
5. Design blocks are transferred into the synthesis tool and optimized.
6. Integration of blocks and verification of the interface between them.
7. Complete design is optimizedusing synthesis.
8. Analysis of the results from synthesis; iterations until satisfied with the results.
9. Complete design validation (gate-level simulation).
10. Placing and routing .
11. Final gate level verification.

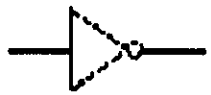
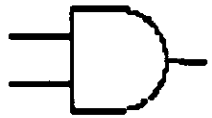
FULL HDL + SYNTHESIS

1. Architecture is specified and partitioned into functional blocks.
2. Entired design is entered via HDL.
3. May be further partinioned in HDL.
4. Validation with an HDL simulator
5. Design is translated into gates.
6. Gate leve blocks are optimized via synthesis.
7. Complete design validation (gate level simulation)
8. Complete design is optimized.
9. Placing and routing.
10. Final gate-level verification.

BOOLEAN ALGEBRA

STATE OF A LOGICAL STATEMENT — TRUE / FALSE

LOGIC CONSTANTS

LOGIC OPERATOR	LOGIC SYMBOL	HW LOGIC PRIMARY ELEMENTS
NOT	—	
AND	·	
OR	+	
XOR	⊕	
COINCIDENCE	⊙	

EXAMPLES OF LOGIC OPERATIONS:

NOT RA = $\overline{RA}$

B AND C = $B \cdot C$

A OR B = $A + B$

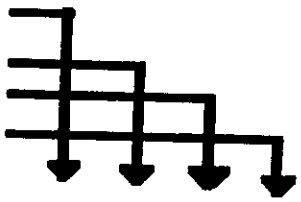
XOR OF A AND B = $A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$

COINCIDENCE OF A AND B = $A \odot B = \overline{A} \cdot \overline{B} + A \cdot B$

WAYS OF REPRESENTING EXPECTED LOGICAL VALUES OF A FUNCTION FOR DIFFERENT CONDITIONS OF THE INPUT VARIABLES.

1.- TRUTH TABLES

ROW	A	B	W
0	F	F	F
1	F	T	F
2	T	F	T
3	T	T	F



(0, 0, 1, 0)

2.- LOGICAL VECTORS

3.- LOGICAL EQUATION $A \cdot \overline{B} = W$

4.- FOR SEQUENTIAL BLOCKS (Operating in synchronism with a train of pulses)

OUTPUTS = f (INPUTS, t) not always

→ original value of the outputs when the clock arrives.

J-K flip-flop

CLK	J	K	Q(n)	Q(n+1)	Condition
F	X	X	q	q	
T	X	X	q	q	
↑	F	F	q	q	
↑	F	T	q	F	Reset
↑	T	F	q	T	Set
↑	T	T	q	$\bar{q}$	

D flip-flop

D	Q
F	F
T	T

ELEMENTS OF BOOLEAN ALGEBRA

1.- PROPERTIES OF OPERATORS

COMMUTE $A + B = B + A$

ASSOCIATE $A + (B + C) = (A + B) + C$

DISTRIBUTE $A \cdot (B + C) = A \cdot B + A \cdot C$

$A + (B \cdot C) = (A + B) \cdot (A + C)$

order: NOT
XOR, COINCIDENCE
AND
OR

Parenthesis override the normal hierarchy

2.- FORMS OF WRITTING THE EXPRESSIONS:

a) SUM OF PRODUCTS $Y = A \cdot B + B \cdot C$
$X = A + B$

B) PRODUCT OF SUMS $Y = (A + \bar{B}) \cdot (\bar{B} + c)$
$X = A \cdot B$

FUNDAMENTAL RELATIONS

FOR SUM OF PRODUCTS	FOR PRODUCT OF SUMS
$A + F = A$	$\overline{\overline{A}} = A$
$A + T = T$	$A \cdot T = A$
$A + A = A$	$A \cdot F = F$
$A + \overline{A} = T$	$A \cdot A = A$
	$A \cdot \overline{A} = F$
$\overline{A + B} = \overline{A} \cdot \overline{B}$ MORGAN'S LAW	$\overline{A \cdot B} = \overline{A} + \overline{B}$

Other identities :

$$A \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A$$

$$A + A \cdot B = A$$

$$A \cdot (\overline{A} + B) = A \cdot \overline{A} + A \cdot B = A \cdot B$$

$$A + \overline{A} \cdot B = A + B$$

$$A \cdot B + \overline{A} \cdot B = B \ (A + \overline{A}) = B$$

RULES FOR MANIPULATION

1.- To obtain LOGICAL EQUATIONS from TT as SUM OF PRODUCTS

Y = OR of the MINTERMS for which the function is T

$\overline{Y}$ = OR of the MINTERMS for which the function is F

MINTERM = product term with all variables in it

EXAMPLE:

A	B	C	X
0	0	0	T
0	0	1	F
0	1	0	T
0	1	1	F
1	0	0	F
1	0	1	T
1	1	0	F
1	1	1	F

$X = m0 + m2 + m5$

$X = (\overline{A}\cdot\overline{B}\cdot\overline{C}) + (\overline{A}\cdot B\cdot\overline{C}) + (A\cdot\overline{B}\cdot C)$

$\overline{X} = m1 + m3 + m4 + m6 + m7$

$\overline{X} = (\overline{A}\cdot\overline{B}\cdot C) + (\overline{A}\cdot B\cdot C) + (A\cdot\overline{B}\cdot\overline{C}) + (A\cdot B\cdot\overline{C}) + (A\cdot B\cdot C)$

2.- To obtain a TT from a LOGICAL EQUATION as a SUM OF PRODUCTS:

A MINTERM will yield one row TRUE

A (NOT MINTERM) will yield TRUE rows for any value of the missing variable.

EXAMPLE: $Y = J \cdot \overline{K} + \overline{J} \cdot K \cdot L + J \cdot K \cdot \overline{L} + K \cdot L$

TRUTH TABLE

ROW	J	K	L	Y	
0	0	0	0	F	
1	0	0	1	F	
2	0	1	0	F	
3	0	1	1	T	Due to terms 2 and 4
4	1	0	0	T	Due to term 1
5	1	0	1	T	Due to term 1
6	1	1	0	T	Due to term 3
7	1	1	1	T	Due to term 4

CONDENSED TT

J	K	L	Y
0	0	X	F
0	1	0	F
0	1	1	T
1	X	X	T

KARNAUGH MAPS

Another way to express TRUTH TABLES (less than 4 variables)

Each row in TT = square in K-MAP

C \ AB	00	01	11	10
0	Y0	Y2	Y6	Y4
1	Y1	Y3	Y7	Y5

For example:

$$V = A \cdot \overline{B} + B + A \cdot \overline{B} \cdot C$$

(Simplification)

$A \cdot \overline{B} + A \cdot \overline{B} \cdot C = A \cdot \overline{B}$

$A \cdot \overline{B} + B = A + B$

C \ AB	00	01	11	10
0	0	1	1	1
1	0	1	1	1

(Simplifying with K-Maps)

$V = A + B$

A	B	C	V
0	0	0	F
0	0	1	F
0	1	0	T
0	1	1	T
1	0	0	T
1	0	1	T
1	1	0	T
1	1	1	T

(TT to LE + simplification)

$V = \overline{C} \cdot \overline{A} \cdot B + \overline{C} \cdot A \cdot B + \overline{C} \cdot A \cdot \overline{B} +$

$+ C \cdot \overline{A} \cdot B + C \cdot A \cdot B + C \cdot A \cdot B =$

$= \overline{C} \cdot B + \overline{C} \cdot A + C \cdot B + C \cdot A =$

$= A + B$

3.- To obtain LOGICAL EQUATIONS from TT as PRODUCT OF SUMS

Y = AND of the opposite of each MAXTERM for which the function is F

$\overline{Y}$ = AND of the opposite of each MAXTERM for which the function is T

MAXTERM = sum term with all variables in it

EXAMPLE:

P	Q	$\overline{X}$
0	0	T
0	1	F
1	0	T
1	1	F

$\overline{X} = M0 \bullet M2$

$\overline{X} = (P + Q) \cdot (\overline{P} + Q) = Q$

$X = M1 \bullet M3$

$X = (P + \overline{Q}) \cdot (\overline{P} + \overline{Q}) = \overline{Q}$

4.- To obtain a TT from a LOGICAL EQUATION as PRODUCT OF SUMS

Each sum term will assure a FALSE expression, whenever all its variables are the opposite of the form in the term.

EXAMPLE: $G = (\overline{A} + B + C) \cdot (\overline{A} + B) \cdot (\overline{A} + \overline{B} + \overline{C})$

(011) opposite of (100) = M4
(01x) opposite of (10x) = row 4 and 5
(000) opposite of (111) = M7

TRUTH TABLE

ROW	A	B	C	G
0	0	0	0	T
1	0	0	1	T
2	0	1	0	T
3	0	1	1	T
4	1	0	0	F
5	1	0	1	F
6	1	1	0	T
7	1	1	1	F

Due to term 1 and 2
Due to term 2

Due to term 3

CONDENSED TT

A	B	C	G
1	0	X	F
1	1	1	F
1	1	0	T
0	X	X	T

REALIZING LOGIC IN HW

CONVENTIONS: T = H ALWAYS — POSITIVE LOGIC

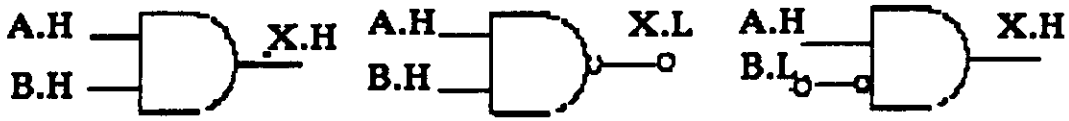
T = H OR T = L — MIXED LOGIC

ADVANTAGE OF MIXED LOGIC: Presents the logic in the way the designer originally thought.

1.- CONTAINS SYMBOLS + VOLTAGE REPRESENTATION

2.- Circles DO NOT change the logic operation

$X = A \cdot B$

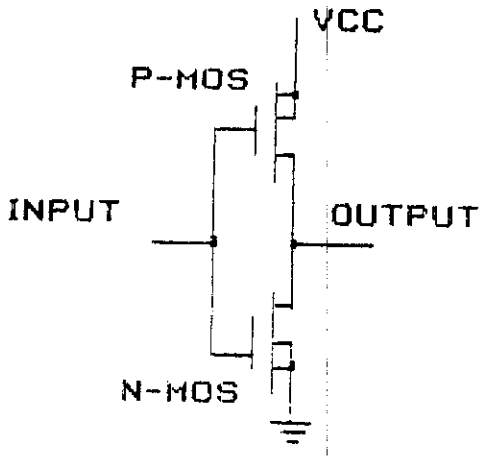


3.- VOLTAGE INVERTER (no logic is performed)

4.- LOGICAL INVERTER (NOT) = SLASH (/)

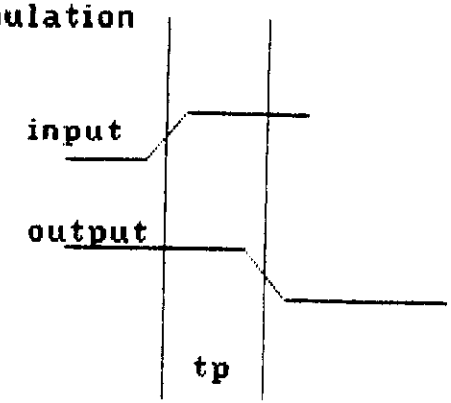
$X = A \cdot \overline{B}$

CMOS INVERTER



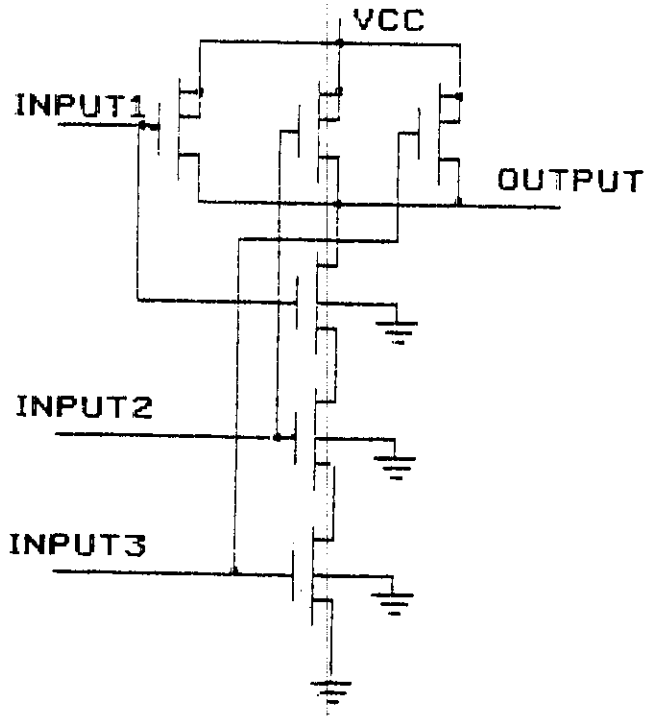
Digital Simulation

MODELS

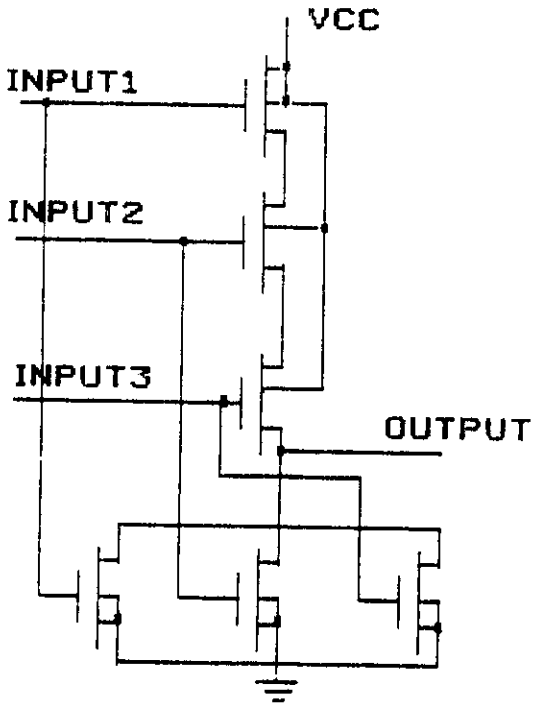


Propagation delay = f (tech, device type, Csal, T, Pw)

CMOS NAND



CMOS NOR



TT for logical identity operation

A	Y
F	F
T	T

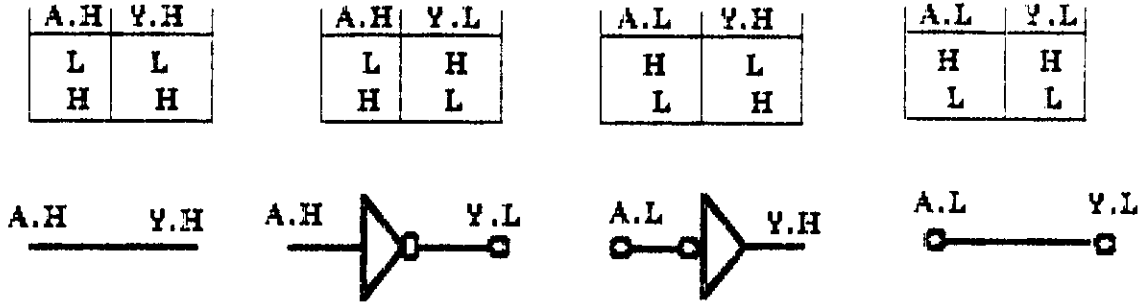


Fig. 3.3 Realization of the logical identity for four choices of voltage.

TT for logical NOT

A	Y
F	T
T	F

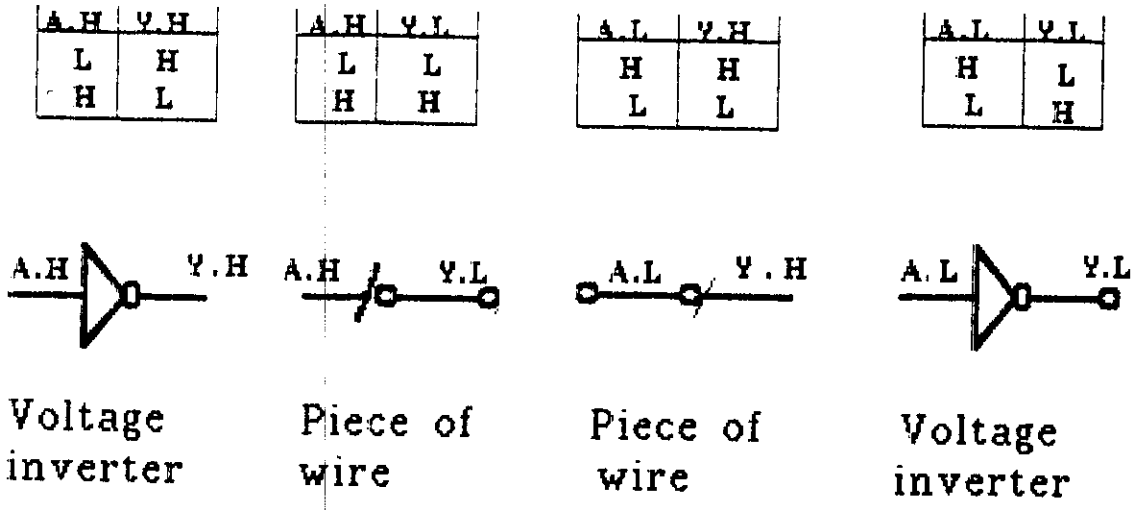


Fig. 3.4 Realization of the logical NOT for four choices of voltage.

TT for AND

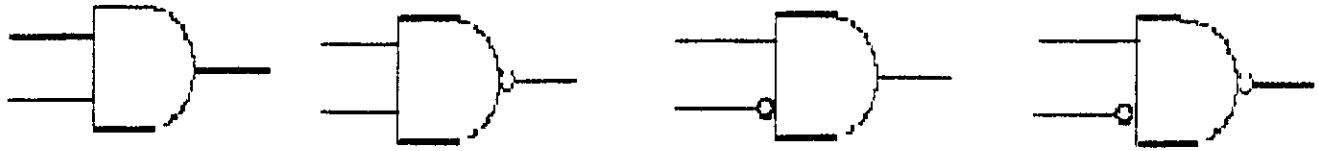
A	B	Y
F	F	F
F	T	F
T	F	F
T	T	T

A.H	B.H	Y.H
L	L	L
L	H	L
H	L	L
H	H	H

A.H	B.H	Y.L
L	L	H
L	H	H
H	L	H
H	H	L

A.H	B.L	Y.H
L	H	L
L	L	L
H	H	L
H	L	H

A.H	B.L	Y.L
L	H	H
L	L	H
H	H	H
H	L	L



A.L	B.H	Y.H
H	L	L
H	H	L
L	L	L
L	H	H

A.L	B.H	Y.L
H	L	H
H	H	H
L	L	H
L	H	L

A.L	B.L	Y.H
H	H	L
H	L	L
L	H	L
L	L	H

A.L	B.L	Y.L
H	H	H
H	L	H
L	H	H
L	L	L

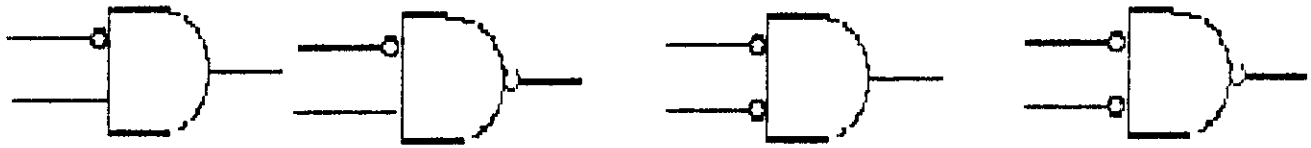


Fig 3.5. Realizations of logical AND for eight choices of voltage.

TT for logic OR

A	B	Y
F	F	F
F	T	T
T	F	T
T	T	T

A.H	B.H	Y.H
L	L	L
L	H	H
H	L	H
H	H	H

A.H	B.H	Y.L
L	L	H
L	H	L
H	L	L
H	H	L

A.H	B.L	Y.H
L	H	L
L	L	H
H	H	H
H	L	H

A.H	B.L	Y.L
L	H	H
L	L	L
H	H	L
H	L	L

A.L	B.H	Y.H
H	L	L
H	H	H
L	L	H
L	H	H

A.L	B.H	Y.L
H	L	H
H	H	L
L	L	L
L	H	L

A. L	B.L	Y.H
H	H	L
H	L	H
L	H	H
L	L	H

A.L	B.L	Y.L
H	H	H
H	L	L
L	H	L
L	L	L

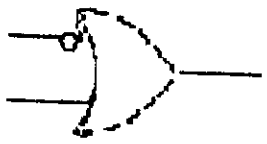
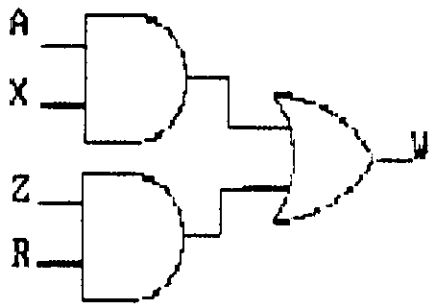


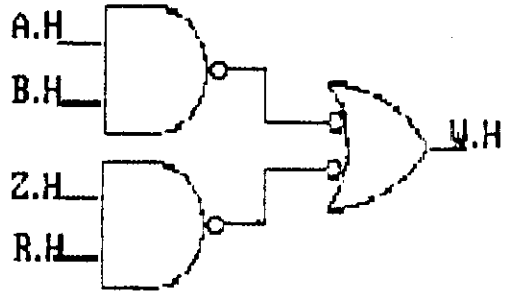
Fig. 3.6. Realizations of logical OR for eight choices of voltage.

$$W = A.X + Z.R$$

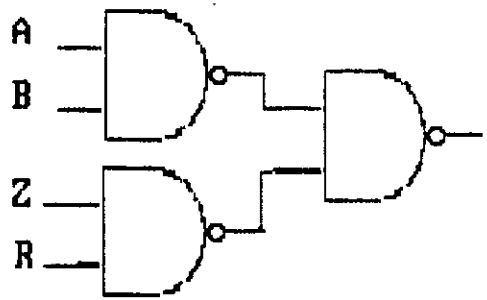
Positive



Mixed



$$W = \overline{\overline{A.X} \cdot \overline{Z.R}}$$



No longer expresses the primary logic.

$$Y = A + B.C.\overline{D}$$

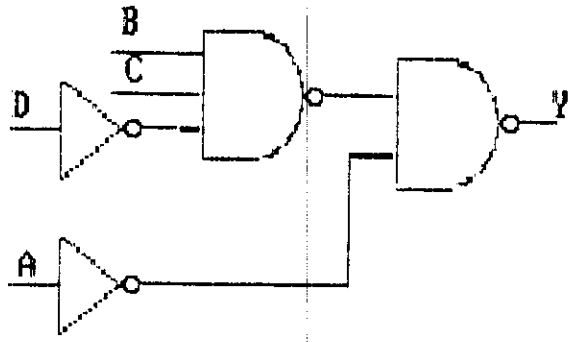
POSITIVE

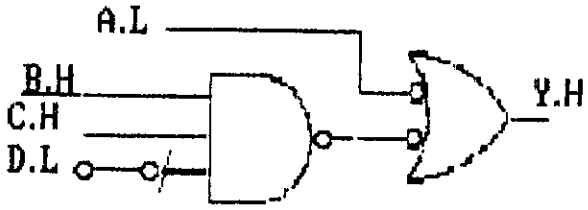
$$Y = A \,.\, \overline{(\,B.C.\overline{D}\,)}$$

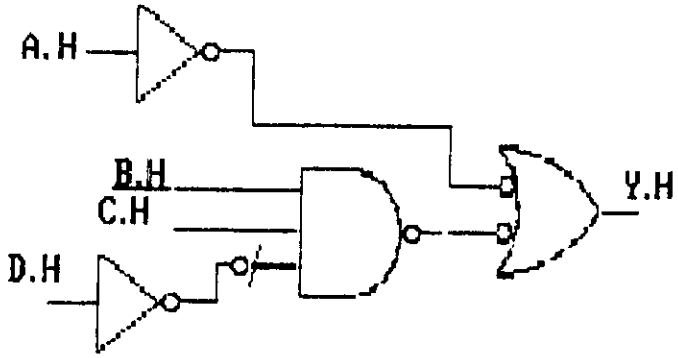
Using NANDs

MIXED

$$Y = A + B.C.\overline{D}$$





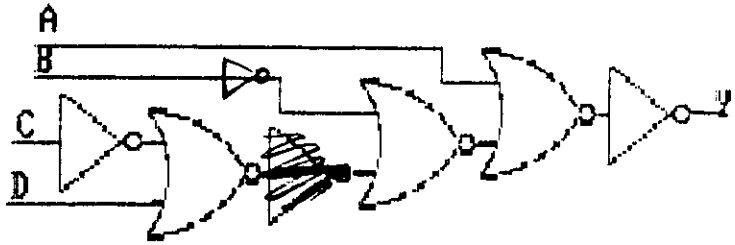


$Y = A + B.C.\overline{D}$

Using NORs

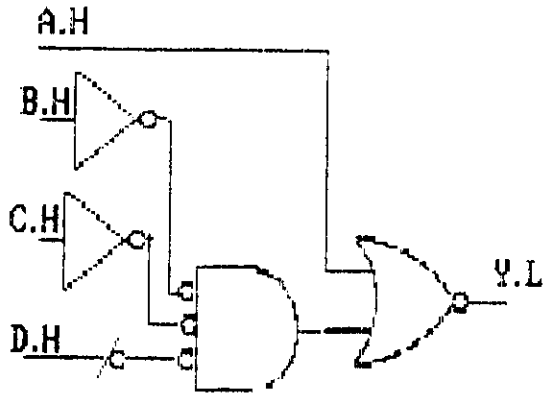
POSITIVE

$$Y = A + \overline{B} + \overline{\overline{C}} + D$$



MIXED

$Y = A + B.C.\overline{D}$

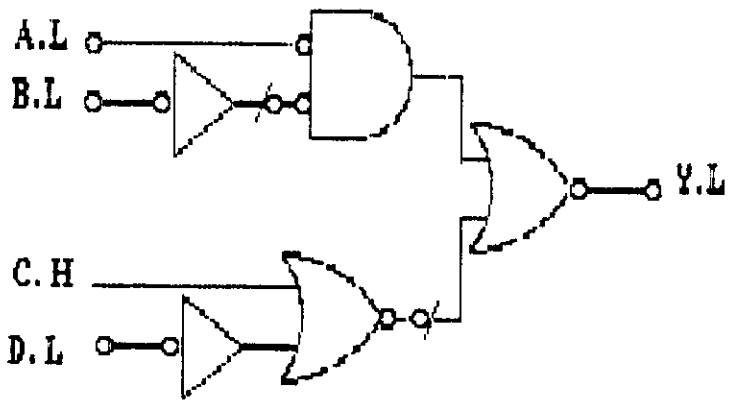


ANALYZING A LOGICAL EXPRESSION FROM A CIRCUIT

1.- Ignore circles and inverters, since they perform no logic by themselves;

2.- Interpret the slash as logical NOT, and the others AND, OR, XOR and COINCIDENCE symbols as the logical operations they implement, and derive the logic expression from the diagram.

Logic equation from analyzing the circuit



$$Y = A * \overline{B} + \overline{(C + D)}$$

OUTPUT FUNCTIONS FOR A BOOLEAN FUNCTION OF TWO VARIABLES

A	B	Z0	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11	Z12	Z13	Z14	Z15
F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
T	F	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

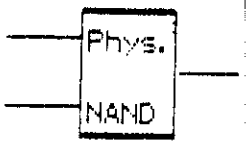
The outputs execute the following functions:

Z0 = F
Z1 = A AND B
Z2 = NOT (A IMPLIES B)
Z3 = A
Z4 = NOT (B IMPLIES A)
Z5 = B
Z6 = A XOR B
Z7 = A OR B
Z8 = NOT(A OR B)
Z9 = A COINCIDENCE B
Z10 = NOT B
Z11 = B IMPLIES A
Z12 = NOT A
Z13 = A IMPLIES B
Z14 = A NAND B
Z15 = T

11 for a physical NAND

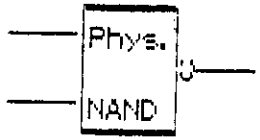
A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

A.H	B.H	Y.H
F	F	T
F	T	T
T	F	T
T	T	F



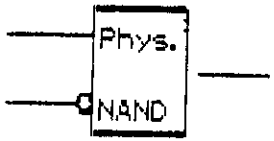
Y= A NAND B

A.H	B.H	Y.L
F	F	F
F	T	F
T	F	F
T	T	T



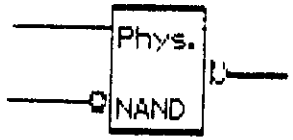
Y = A AND B

A.H	B.L	Y.H
F	T	T
F	F	T
T	T	T
T	F	F



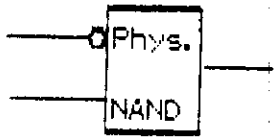
Y=A IMPLIES B

A.H	B.L	Y.L
F	T	F
F	F	F
T	T	F
T	F	T



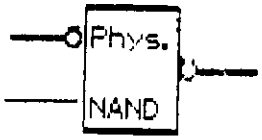
Y=$\overline{\text{A IMPLIES B}}$

A.L	B.H	Y.H
T	F	T
T	T	T
F	F	T
F	T	F



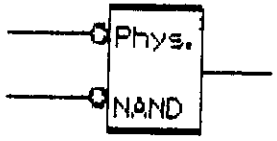
Y = B IMPLIES A

A.L	B.H	Y.L
T	F	F
T	T	F
F	F	F
F	T	T



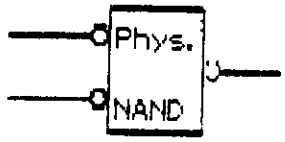
Y= $\overline{\text{B IMPLIES A}}$

A.L	B.L	Y.H
T	T	T
T	F	T
F	T	T
F	F	F



Y = A OR B

A.L	B.L	Y.L
T	T	F
T	F	F
F	T	F
F	F	T



Y = A NOR B

LOGIC PERFORMED BY A PHYSICAL NAND

HOW CAN A MIXED LOGICIAN READ A POSITIVE LOGIC CIRCUIT ?

1.- Append .H to the positive logic inputs and output.

2.-Replace the negated input or output by non-negated mixed-logic forms.

3.-When the circles do not match at the ends of a line, insert a slash to emphasize the implied logical NOT.

4.-Where a gate is surrounded by slashes, you may simplify the solution by altering the AND and OR gate symbol to its mixed-logic OR and AND counterpart.

COMMON OPERATION WITHIN A CIRCUIT

a) Movement of data from one part of the system to another;

b) Selection of the given data from several possible;

c) Routing data from a source to one or several destinations;

d) Transformation of the data from one representation to another;

e) Comparing data arithmetically with another data;

f) Manipulation of the data, arithmetically or logically;

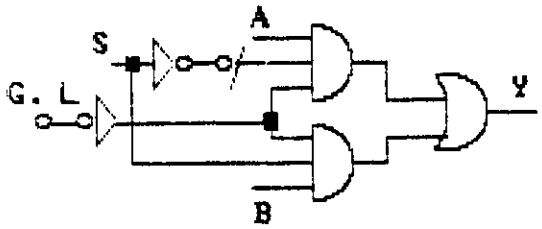
COMMON BUILDING BLOCKS

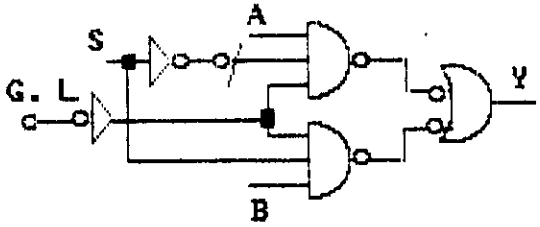
COMBINATIONAL BLOCKS

1.- MULTIPLEXER: Selects one of several inputs

s selects lines can manage 2^S inputs

$$Y = G \cdot (A \cdot \overline{S} + B \cdot S)$$





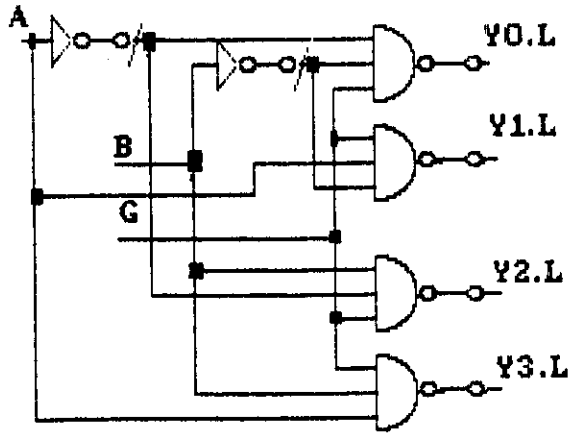
2.- DEMULTIPLEXER : Sends data from a single source to several destinations.

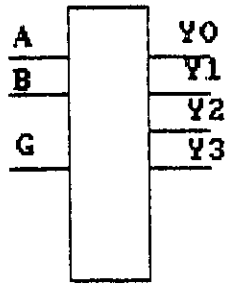
$$Y0 = \overline{B} \cdot \overline{A} \cdot G$$

$$Y1 = \overline{B} \cdot A \cdot G$$

$$Y2 = B \cdot \overline{A} \cdot G$$

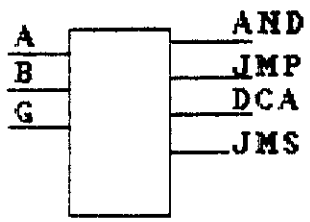
$$Y3 = B \cdot A \cdot G$$





3.-DECODER: Identifies a particular code.
BCD to decimal decoder; operation codes

Same physical circuit as demultiplexer



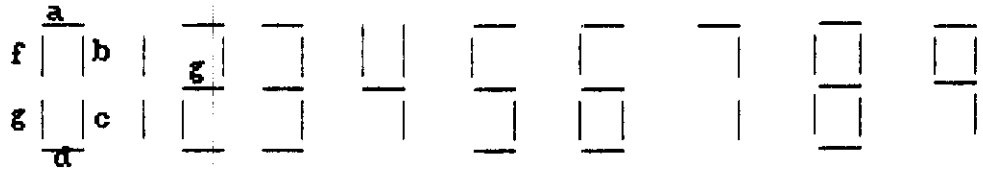
code	C	B	A	instruction
0	0	0	0	AND
1	0	0	1	JMP
2	0	1	0	DCA
3	0	1	1	JMS

$G = UCC$

$JMP = \bar{C} \cdot B \cdot A$

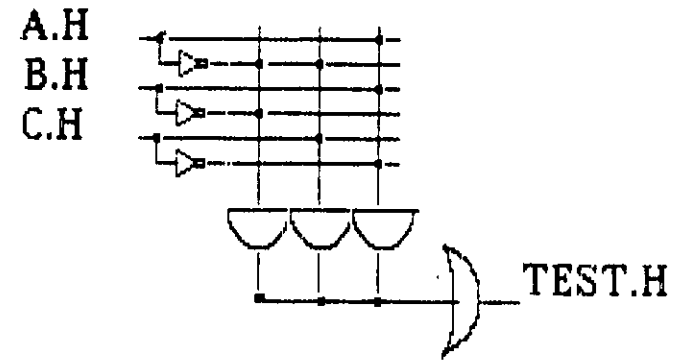
4.- ENCODER: Forms an encoded representation of a set of inputs. For example: codification of numbers 0 to 9

5.- CODE CONVERTERS: For example BCD to 7-segment



You can implement the logic of the TT or use ROMs, PLA, PLE, PAL

a) PLA Programmable Logic Array.
Logic function as a sum-of-products
matrix of ANDs and ORs
A PLA of 3 inputs will accomplish on each row
the function $A.\bar{A}.B.\bar{B}.C.\bar{C}.$



$$TEST = \bar{A}.\bar{B} + \bar{A}.C + A.B.\bar{C}$$

b) PLE (Programmable Logic Element) a PROM
that provides all possible product terms of its input.

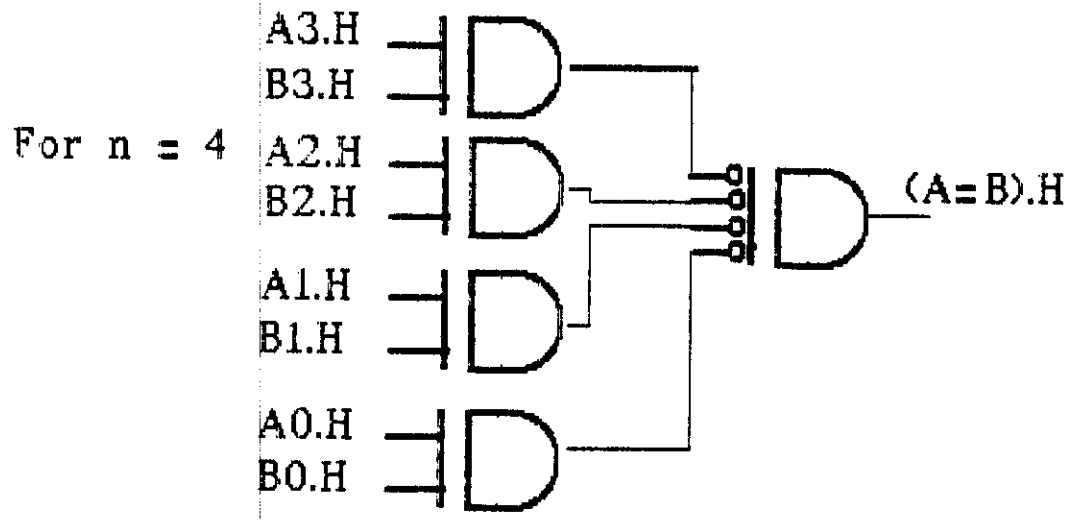
To generate a logic function, program a "1"
if the canonical term contributes,

c) PAL (programmable Array Logic) allows
to specify the nature of the product term.

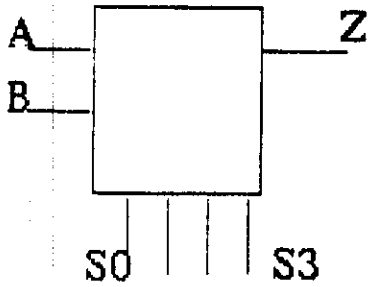
The way in which the product may be form
into sums is fixed in the chip.

6.- COMPARATOR: Verifies the coincidence of 2 patterns of n input bits.

$$A \ EQ \ B = (A0 \cdot B0) \cdot (\bar{A}1 \cdot B1) \ldots \cdot (An \cdot Bn)$$



7.- GENERAL PURPOSE DEVICE (Only logic)

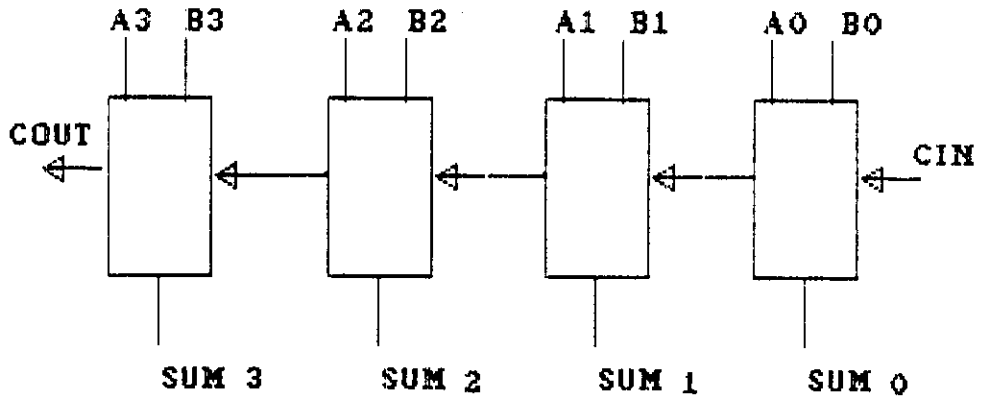


8.- FULL ADDER

$$SUM = \overline{CIN} \cdot (\bar{A}.B + A.\bar{B}) + CIN. (\bar{A}.\bar{B} + A.B) =$$

$$= A \oplus B \oplus CIN$$

$$COUT = A.B + CIN .(A + B)$$



9.- Arithmetic Logic Unit (ALU) combines the universal logic circuit with binary arithmetic operations.

arithmetic operations include:

addition A PLUS B
subtraction A PLUS (MINUS B), MINUS B is realized by the negation (complement) of B.

incrementing A PLUS 1
decrementing A MINUS 1
A PLUS A
B MINUS A
MINUS A
MINUS B

< 16 operations,

4 control inputs
1 selection input

2 4-bit inputs,

1 4-bit output,

CIN,

COUT.

perform the 16 logic functions of two variables

plus the 16 arithmetic functions.

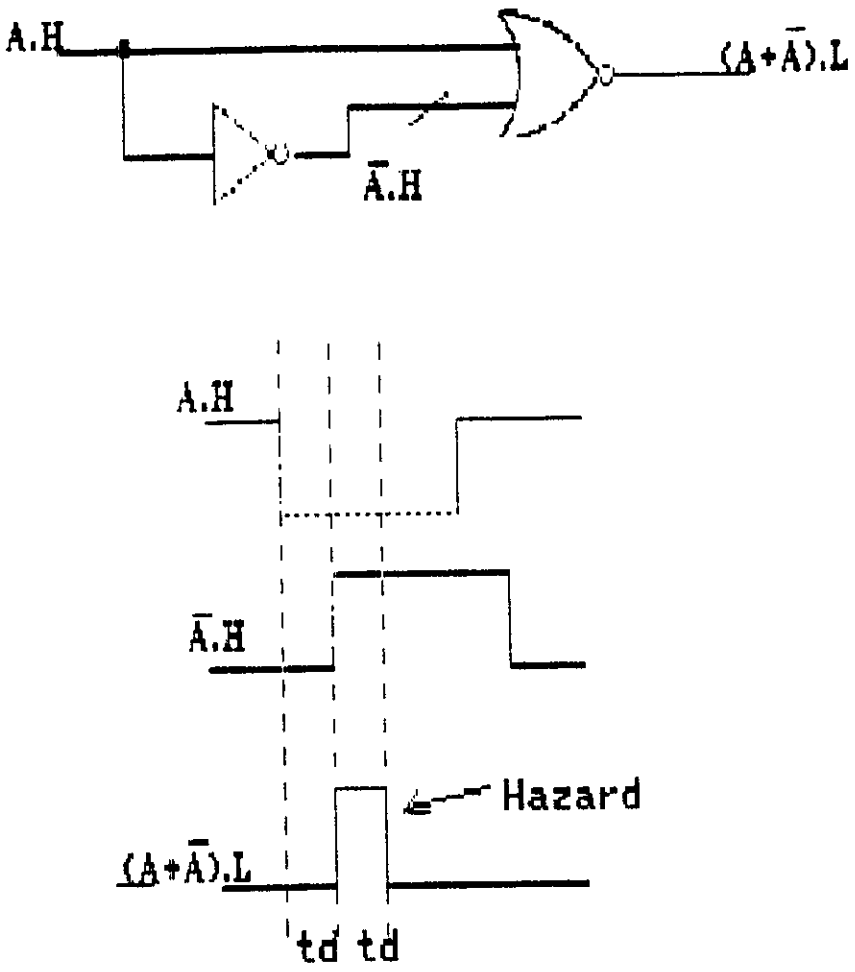
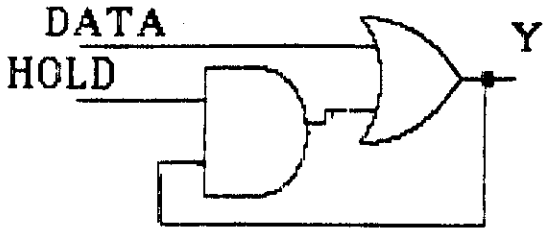


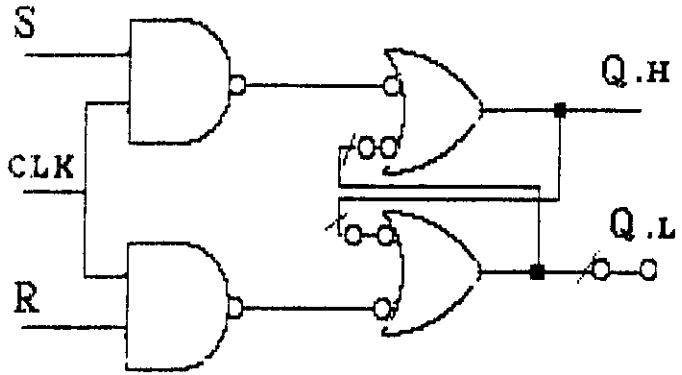
Fig. 3.10. Appearance of a hazard.

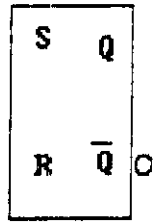
SEQUENTIAL BLOCKS

1.- LATCH



2.- RSFF FLIP-FLOP



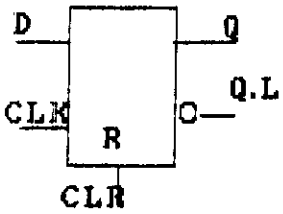


3.- D FLIP-FLOP

Data storage

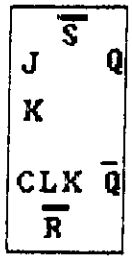
Time delay

synchronizer



CLR	CLK	D	Q
T	X	X	0
F	⌐	1	1
F	⌐	0	0

4.- JK FLIP-FLOP

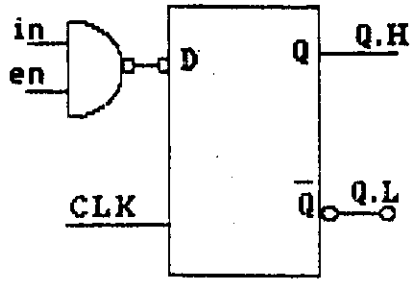


S	R	CLK	J	K	Q (n+1)
L	H	X	X	X	H
H	L	X	X	X	L
L	L	X	X	X	-
H	H	⌐	L	L	L
H	H	⌐	L	H	Q(n) Hold
H	H	⌐	H	L	Q̄(n) Toggle
H	H	⌐	H	H	H

5.- REGISTERS

For Data storage

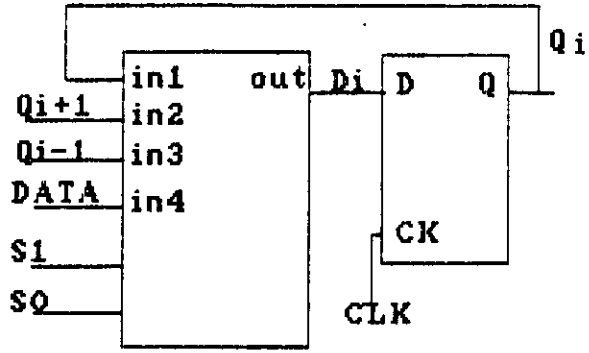
Enable D flip-flop

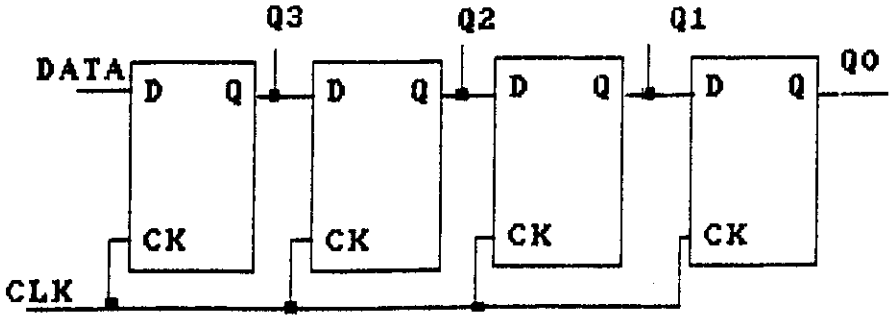


6.- SHIFT REGISTERS

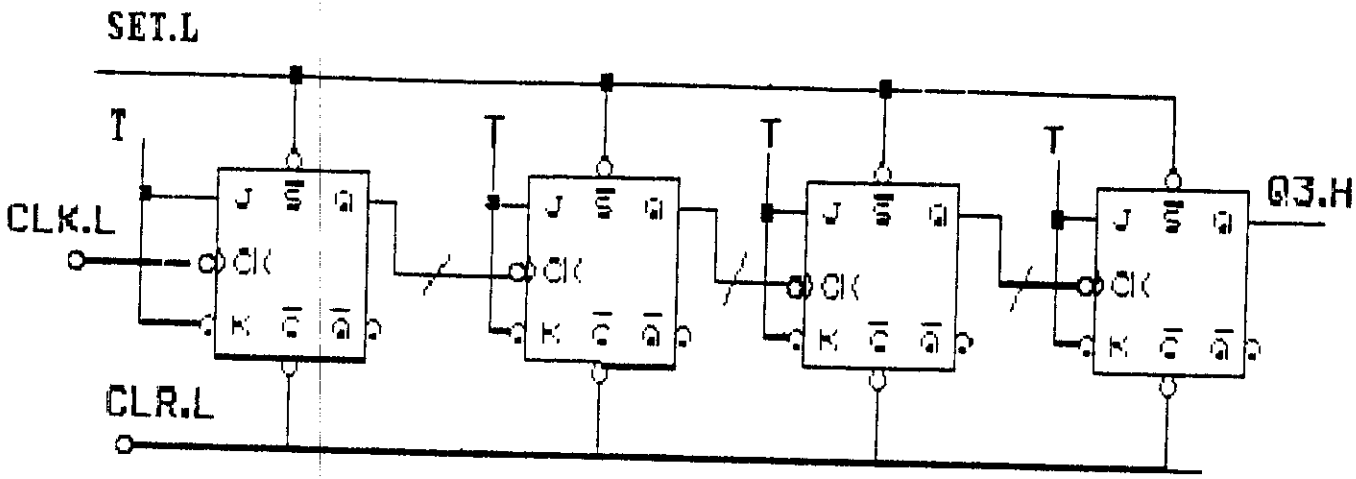
performs lateral movement of data one position to an adjacent one.

Types: Serial to parallel
Parallel to serial
Serial to serial
Parallel to parallel



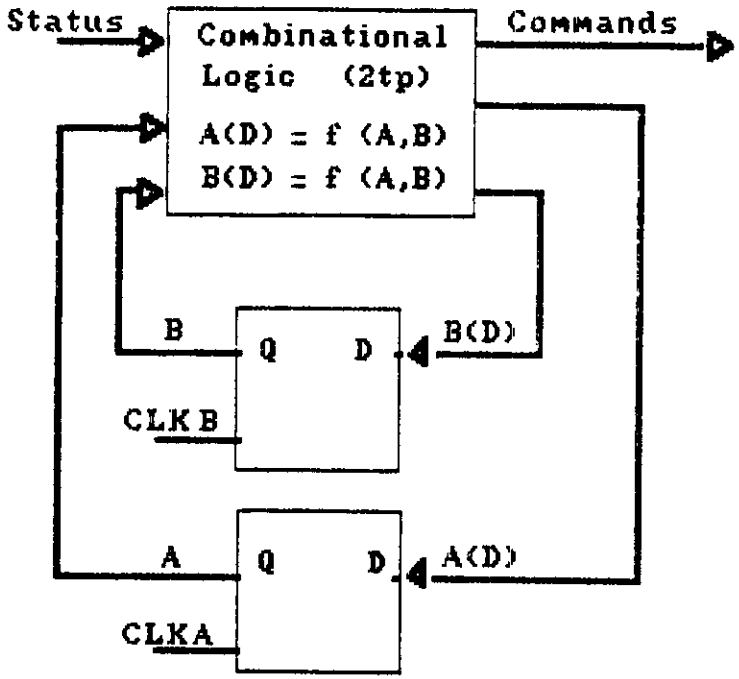


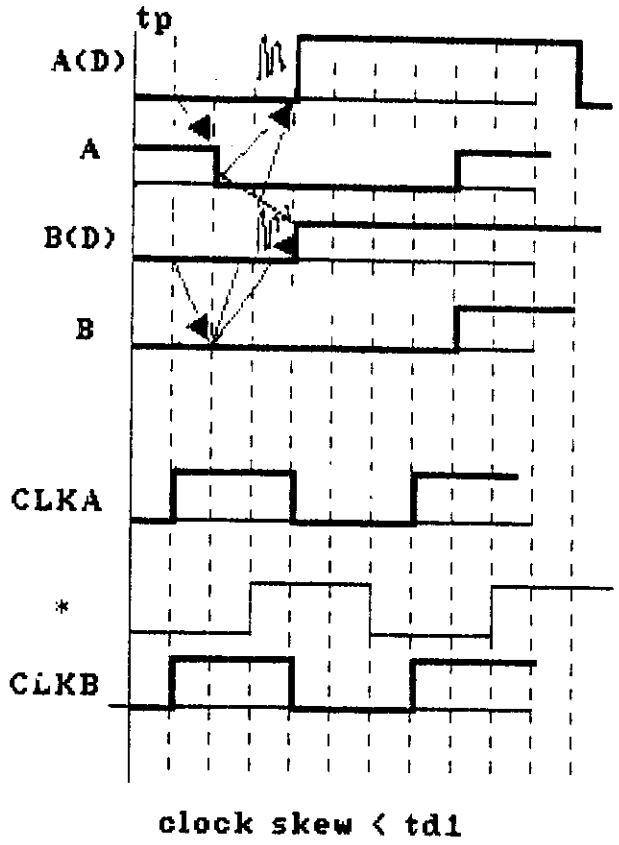
S1	SO	OPERATION
0	0	Hold
0	1	Shift R
1	0	Shift L
1	1	Load new



Asyncronous counter.

CLOCK SKEW





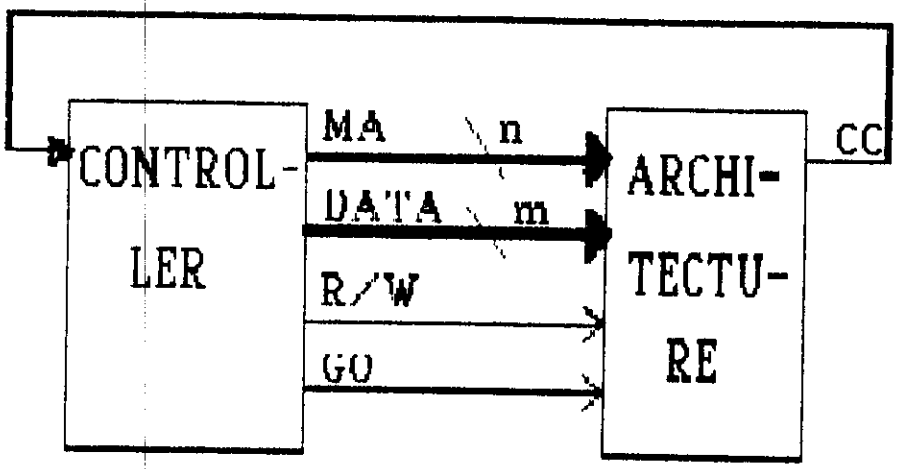
clock skew < td1

td1 - time when outputs of combina - tional block starts to change.

1.- Don't gate clock lines
2.- Use all positive gated FF
3.- Same length of lines
4.- Same buffers for all lines

DESIGN METHODOLOGY (Start top-down)

Step one: separation of control algorithm
from architecture to be controlled by it.

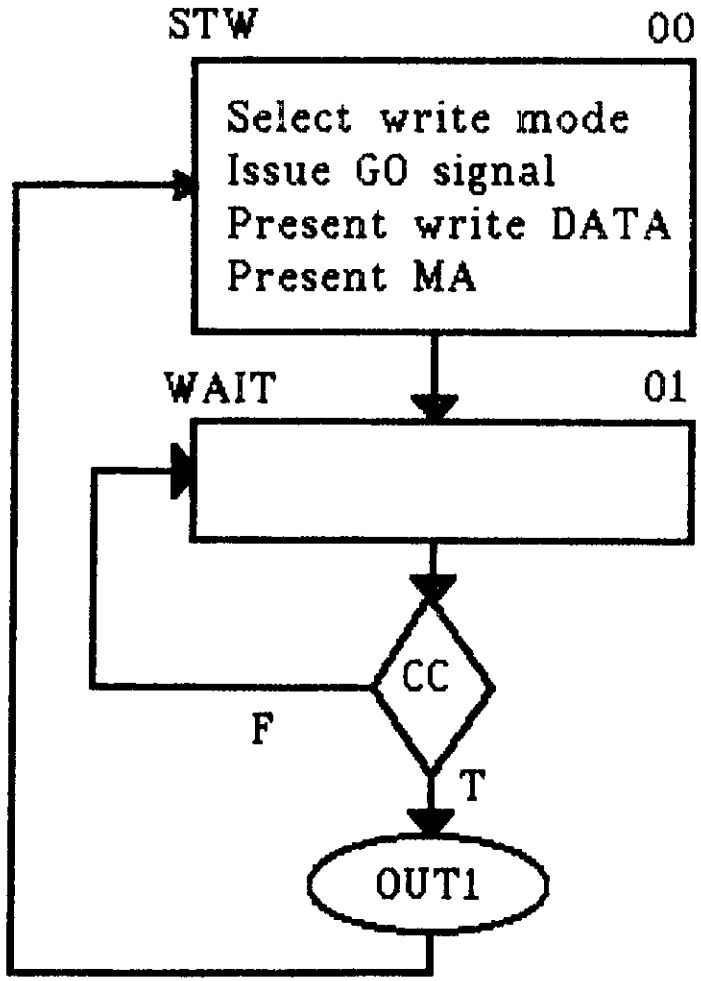


ARCHITECTURE: Elements to perform the original problem.

CONTROLLER: Algorithm plus HW

ALGORITHM: A properly sequenced set of command signals to make the architecture perform the original problem

Algorithm State Machine notation
for describing synchronous circuits.



Methods for synthesing ASM

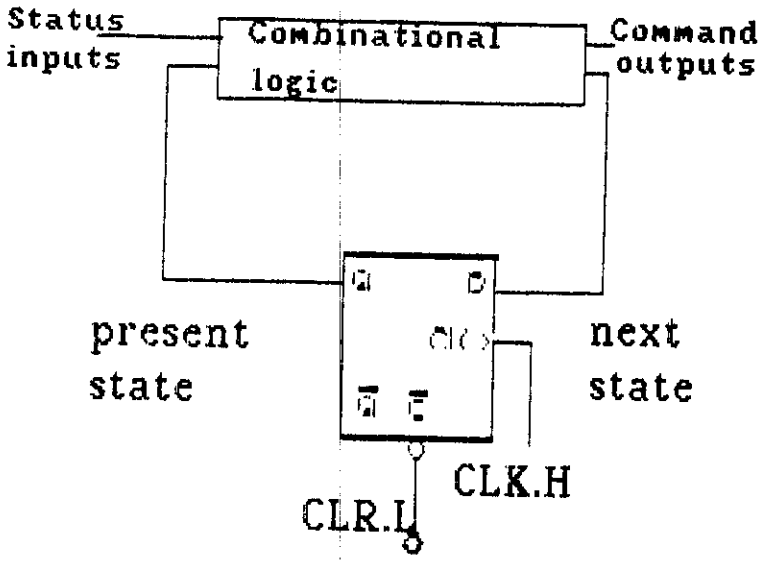
1.- Traditional

2.- Multiplexer controlled

3.- One-hot

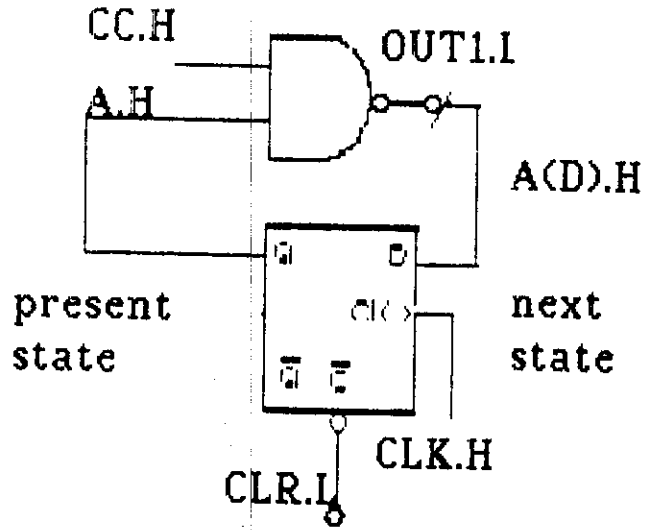
4.- ROM-based

TRADITIONAL METHOD



1.- Encoded representation of present state
2.- Compute the code for next state
3.- one FF for each next state
4.- write STT and equations

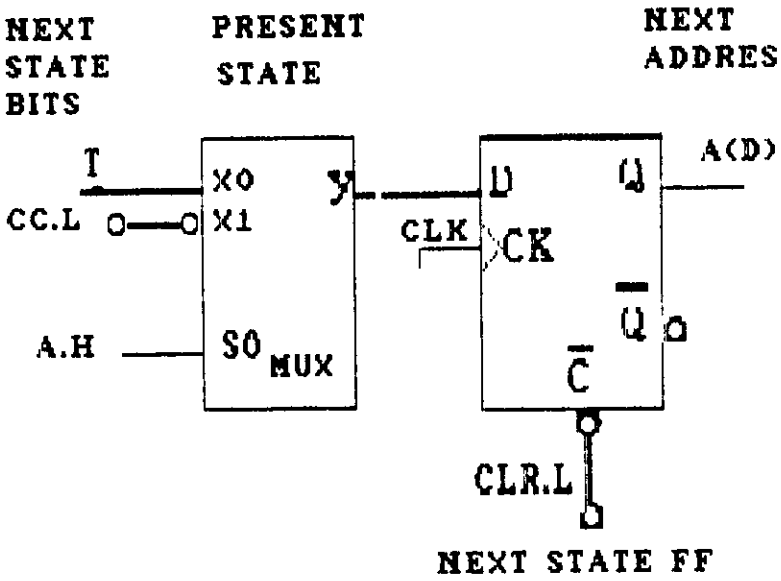
Output	PRESENT STATE name	code A.H	code CC.H	NEXT STATE code A(D)	name
	STW	0	X	1	WAIT
OUT1	WAIT	1	1	0	STW
	WAIT	1	0	A	WAIT



$OUT1 = A.CC$

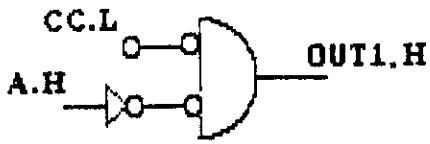
$A(D) = \overline{A.CC}$

MULTIPLEXER CONTROLLED METHOD



1.- Encoded representation of presenta state
2.- Compute code for next state
3.- one FF for each next state
4.- write STT and equations

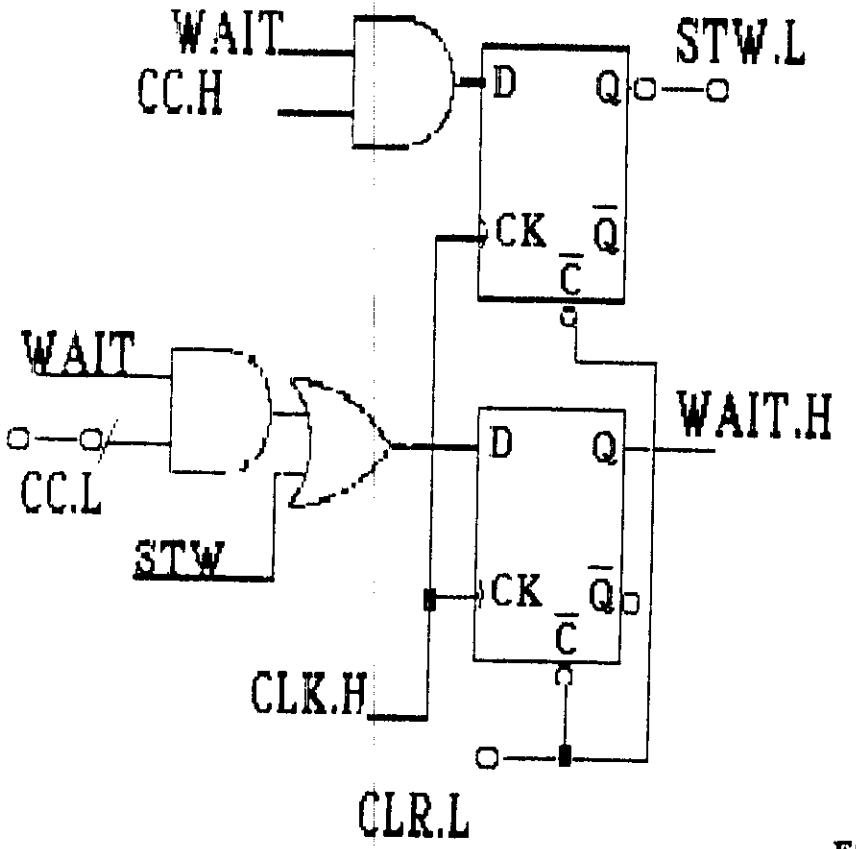
5.- use a MUX at the input of each State FF.



OUT-PUT	PRESENT STATE		NEXT STATE		CONDITIONS
	CODE	NAME	NAME	CODE	
	0	STW	WAIT	1	X
	1	WAIT	WAIT	1	$\overline{CC}$
OUT1	1	WAIT	STW	0	CC

MUXA(0) = T
MUXA(1) = CC
OUT1 = A.CC

ONE-HOT METHOD



1.- Use one FF for each state
2.- No encoding
* 3.- Only one of the state-FF can be T at a time
4.- Compute the FF's input

NEXT STATE	PRESENT STATE	CONDITION
WAIT	STW	X
WAIT	WAIT	CC
STW	WAIT	CC

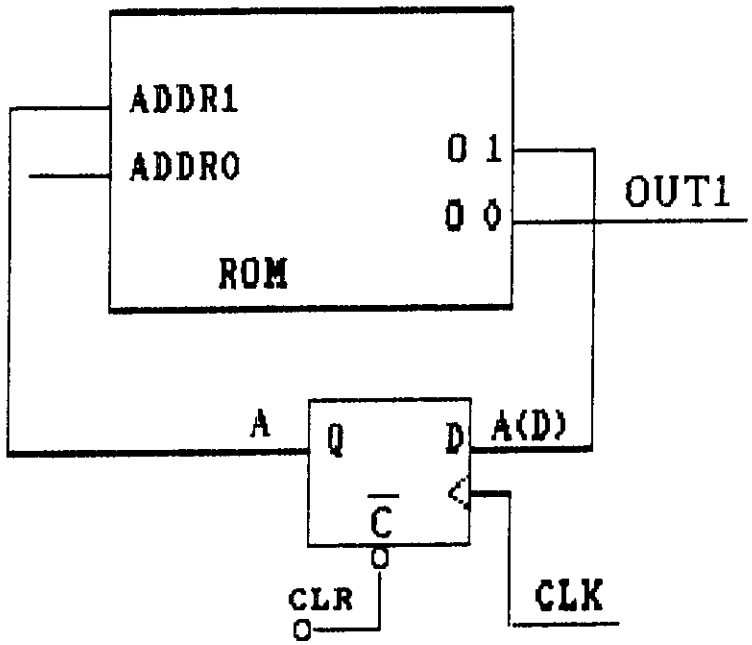
EQUATIONS:

NextStateWait = STW + WAIT.$\overline{CC}$

NextstateSTW = WAIT.CC

ROM-METHOD

ADDRESS A	ADDRESS CC	OUTPUT A(D)	OUTPUT OUT1
0	X	1	0
1	1	0	1
1	0	1	0



$$A(D) = \overline{A.CC}$$

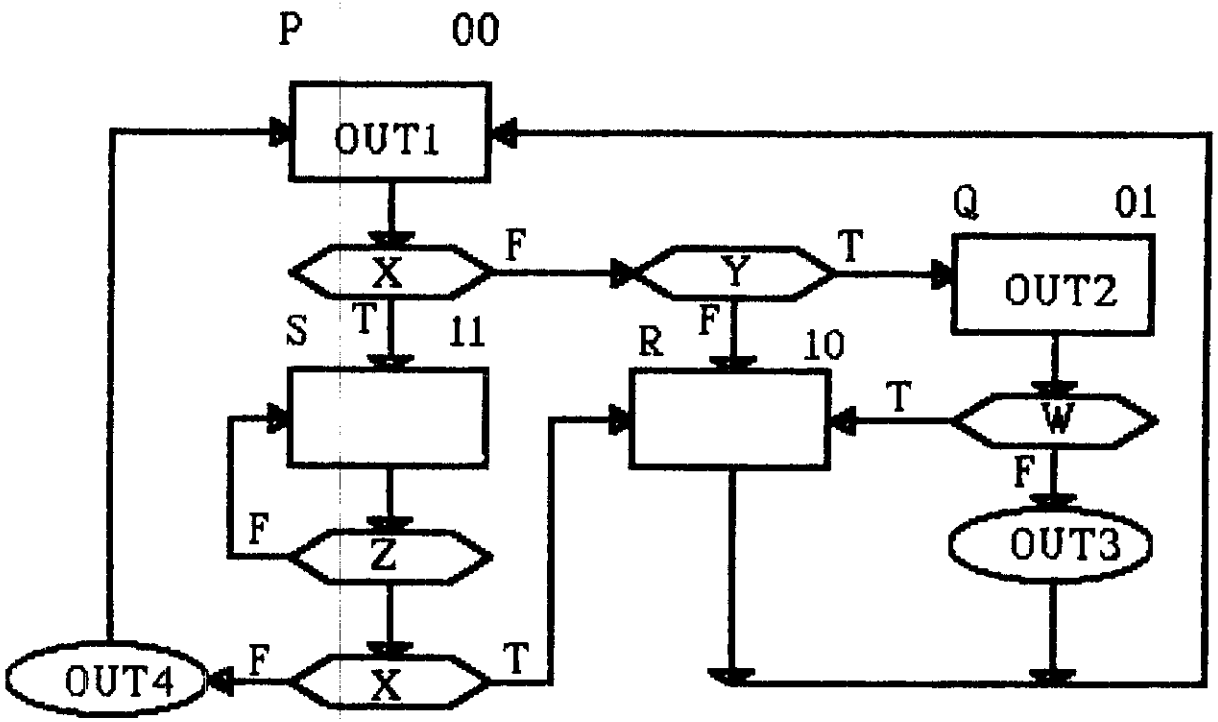


Fig. 4.3 A more complex ASM.

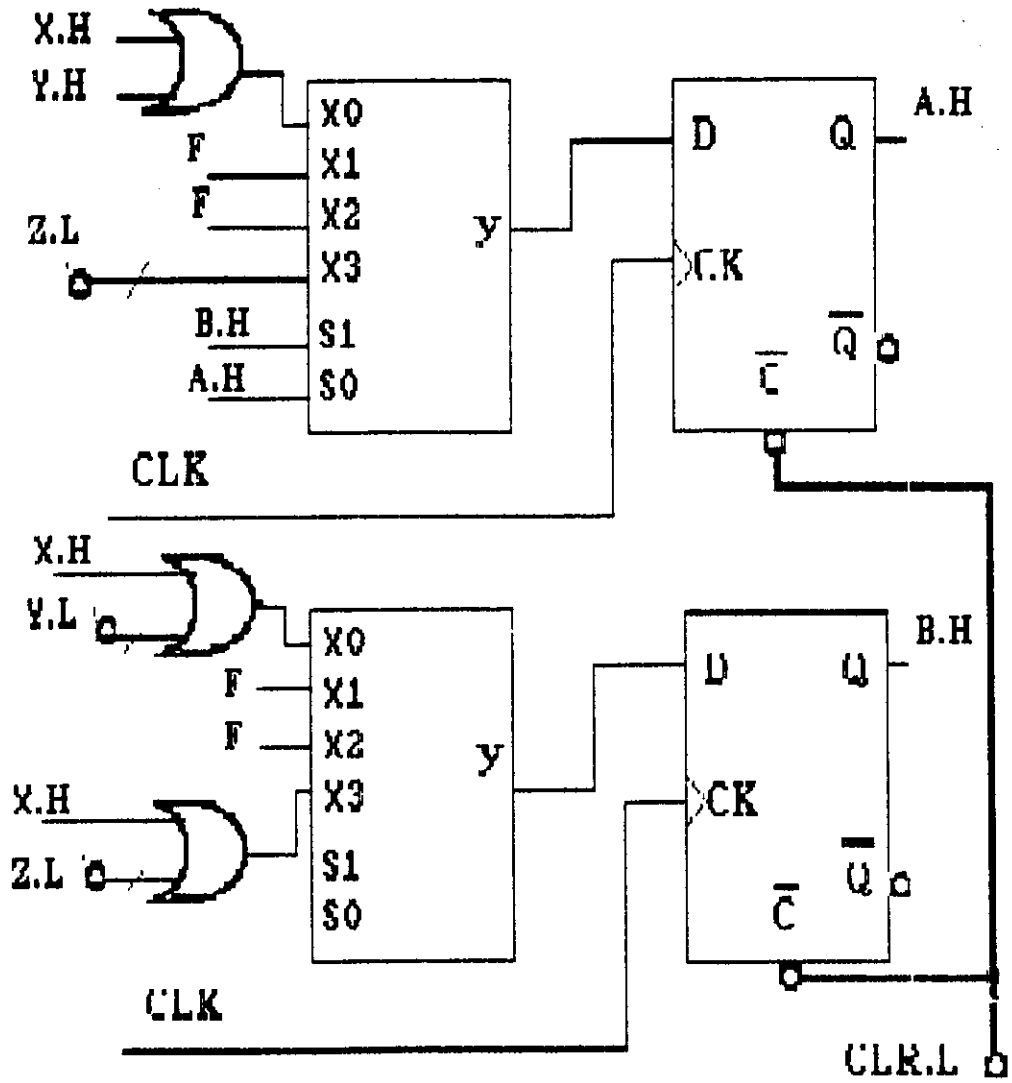


Fig. 4.6 Implementation of the ASM of Fig. 4.3 using the multiplexer method.

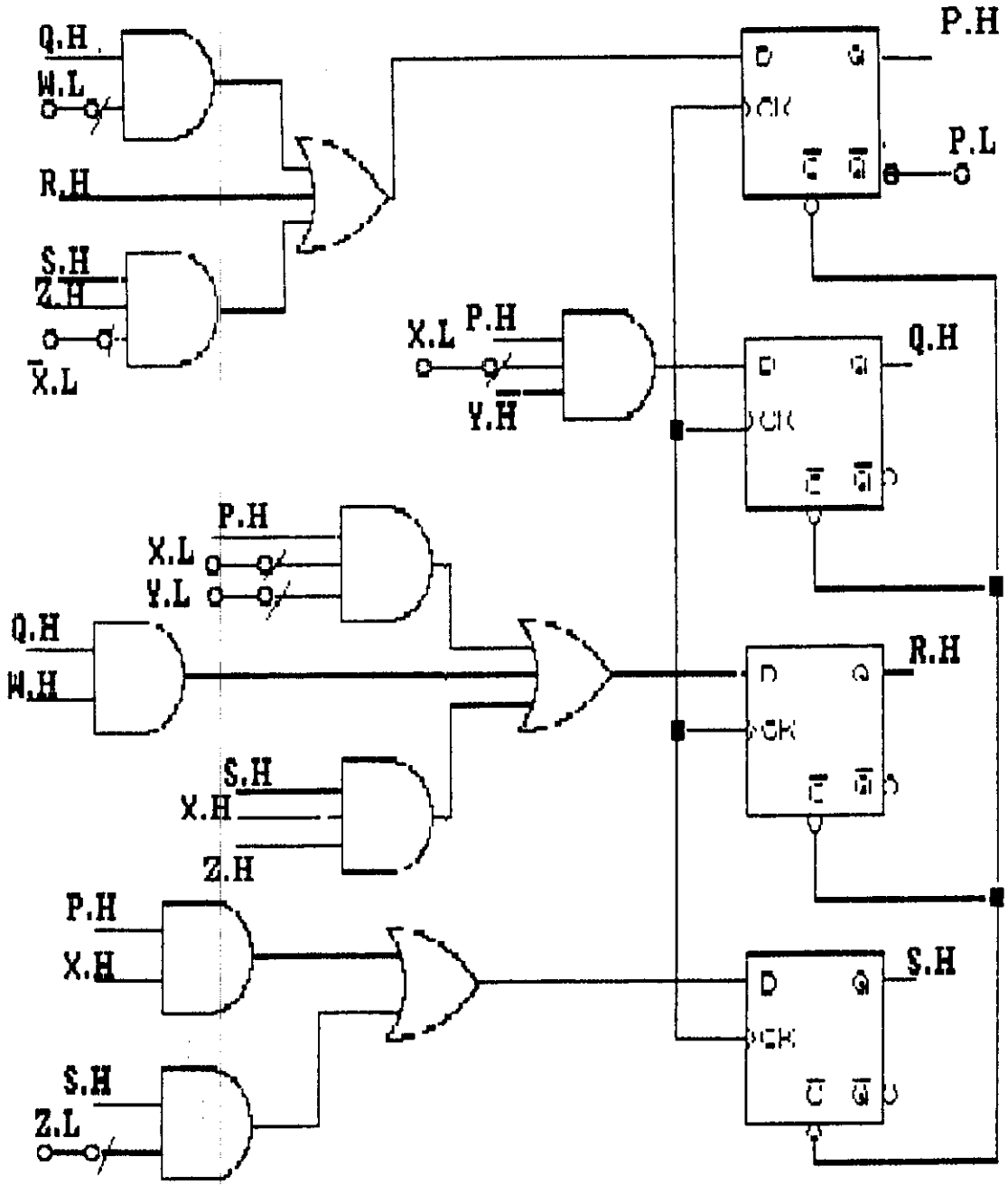


Fig. 4.8 a. A one-hot controller for the ASM of Fig. 4.3.

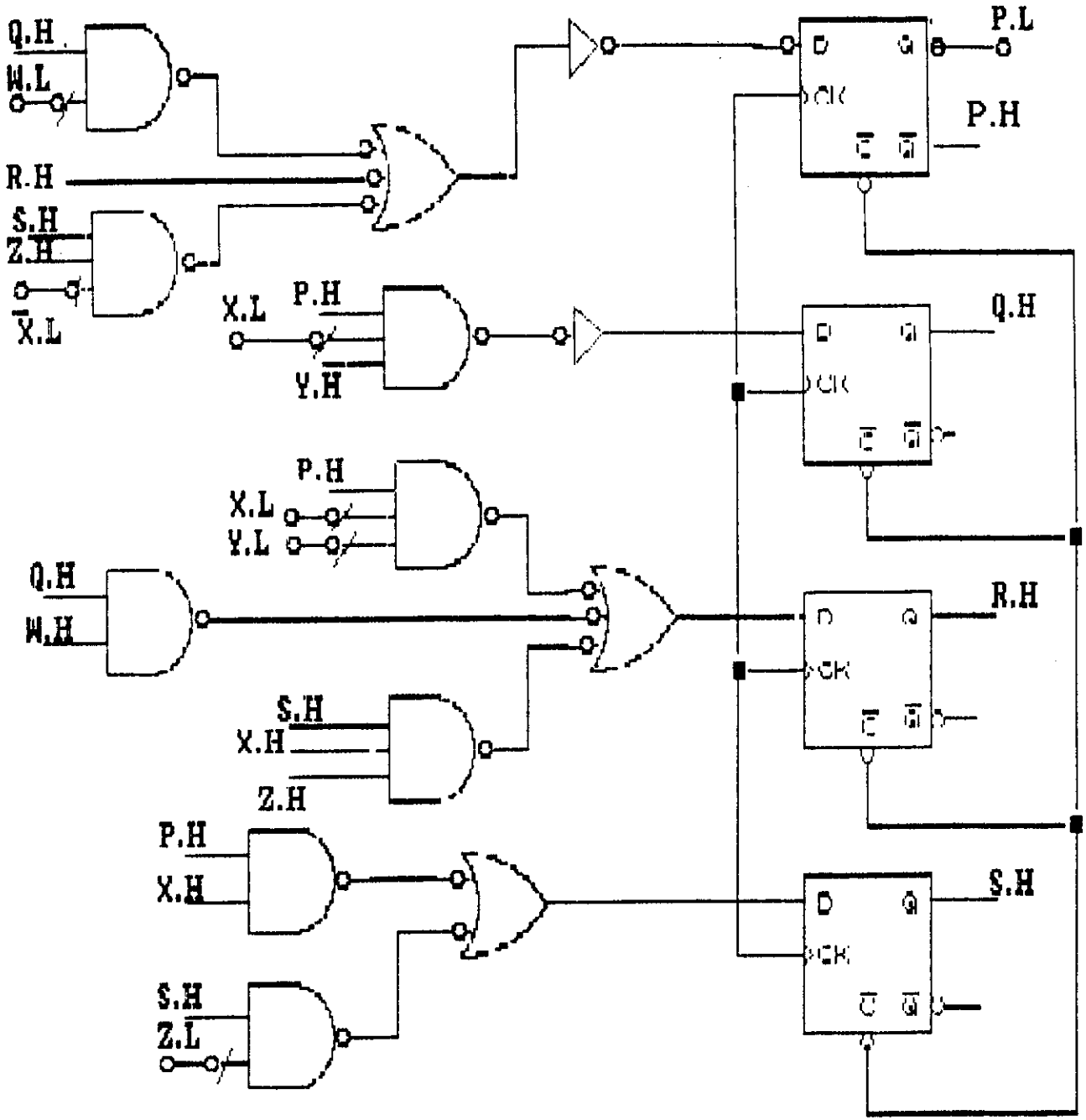


Fig. 4.8 b. Same as Fig. 4.8a, using NANDs.