Principles of Computer Architecture

Miles Murdocca and Vincent Heuring

Appendix A: Digital Logic

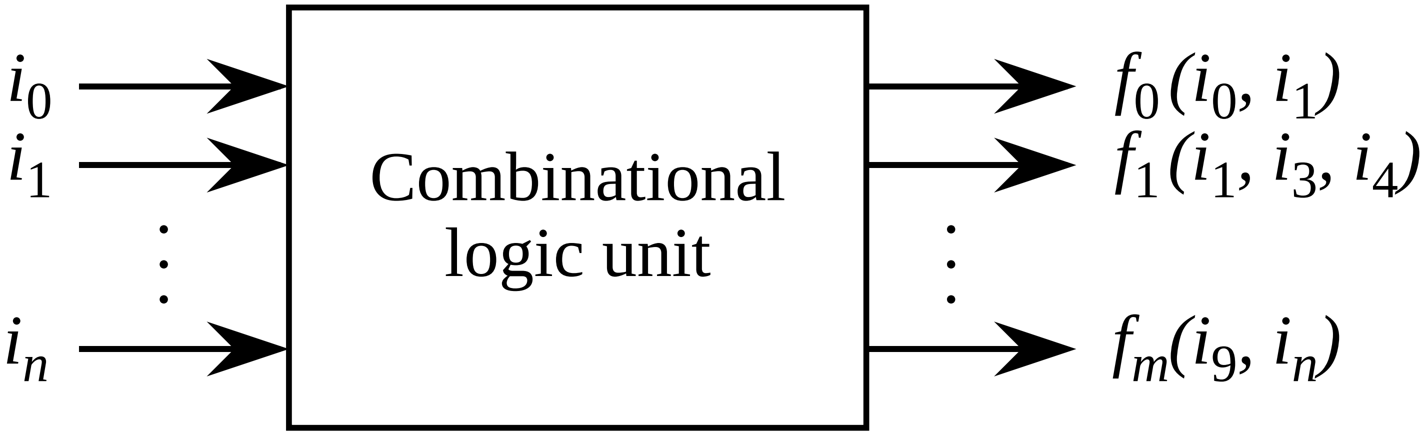
Chapter Contents

Some Definitions

- ***Combinational logic:*** **a digital logic circuit in which logical decisions are made based only on combinations of the inputs. *e.g.* an adder.**
- ***Sequential logic:*** **a circuit in which decisions are made based on combinations of the current inputs as well as the past history of inputs. *e.g.* a memory unit.**
- ***Finite state machine:*** **a circuit which has an internal state, and whose outputs are functions of both current inputs and its internal state. *e.g.* a vending machine controller.**

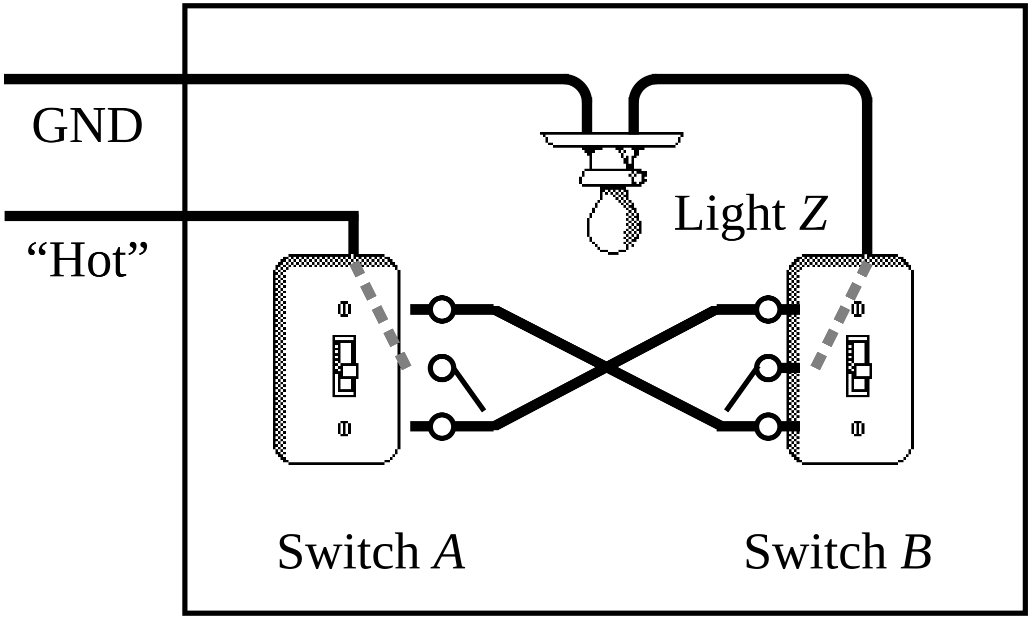
The Combinational Logic Unit

- Translates a set of inputs into a set of outputs according to one or more mapping functions.
- Inputs and outputs for a CLU normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 V and 0 V for example.
- The outputs of a CLU are strictly functions of the inputs, and the outputs are updated immediately after the inputs change. A set of inputs i_0 – i_n are presented to the CLU, which produces a set of outputs according to mapping functions f_0 – f_m.



A Truth Table

- **Developed in 1854 by George Boole.**
- **Further developed by Claude Shannon (Bell Labs).**
- **Outputs are computed for all possible input combinations (how many input combinations are there?)**
- **Consider a room with two light switches. How must they work?**



Inputs		Output
A	B	Z
0	0	0
0	1	1
1	0	1
1	1	0

Alternate Assignment of Outputs to Switch Settings

- **We can make the assignment of output values to input combinations any way that we want to achieve the desired input-output behavior.**

Inputs		Output
A	*B*	*Z*
0	0	1
0	1	0
1	0	0
1	1	1

Truth Tables Showing All Possible Functions of Two Binary Variables

- **The more frequently used functions have names: AND, XOR, OR, NOR, XOR, and NAND. (Always use upper case spelling.)**

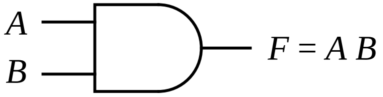
Inputs		Outputs							
A	B	*False*	*AND*	$A\overline{B}$	A	$\overline{A}B$	B	*XOR*	*OR*
0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Inputs		Outputs							
A	B	*NOR*	*XNOR*	$\overline{B}$	$A+\overline{B}$	$\overline{A}$	$\overline{A}+B$	*NAND*	*True*
0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1

Logic Gates and Their Symbols

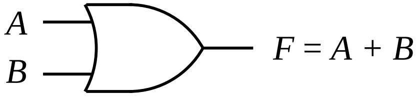
- Logic symbols shown for AND, OR, buffer, and NOT Boolean functions.
- Note the use of the “inversion bubble.”
- (Be careful about the “nose” of the gate when drawing AND vs. OR.)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



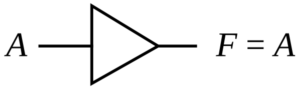
AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



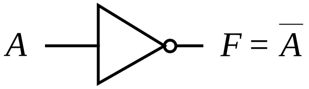
OR

A	F
0	0
1	1



Buffer

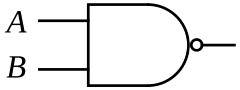
A	F
0	1
1	0



NOT (Inverter)

Logic Gates and their Symbols (cont’)

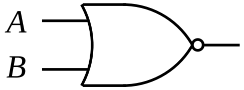
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



$F = \overline{A\,B}$

NAND

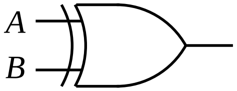
A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



$F = \overline{A + B}$

NOR

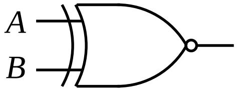
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



$F = A \oplus B$

Exclusive-OR (XOR)

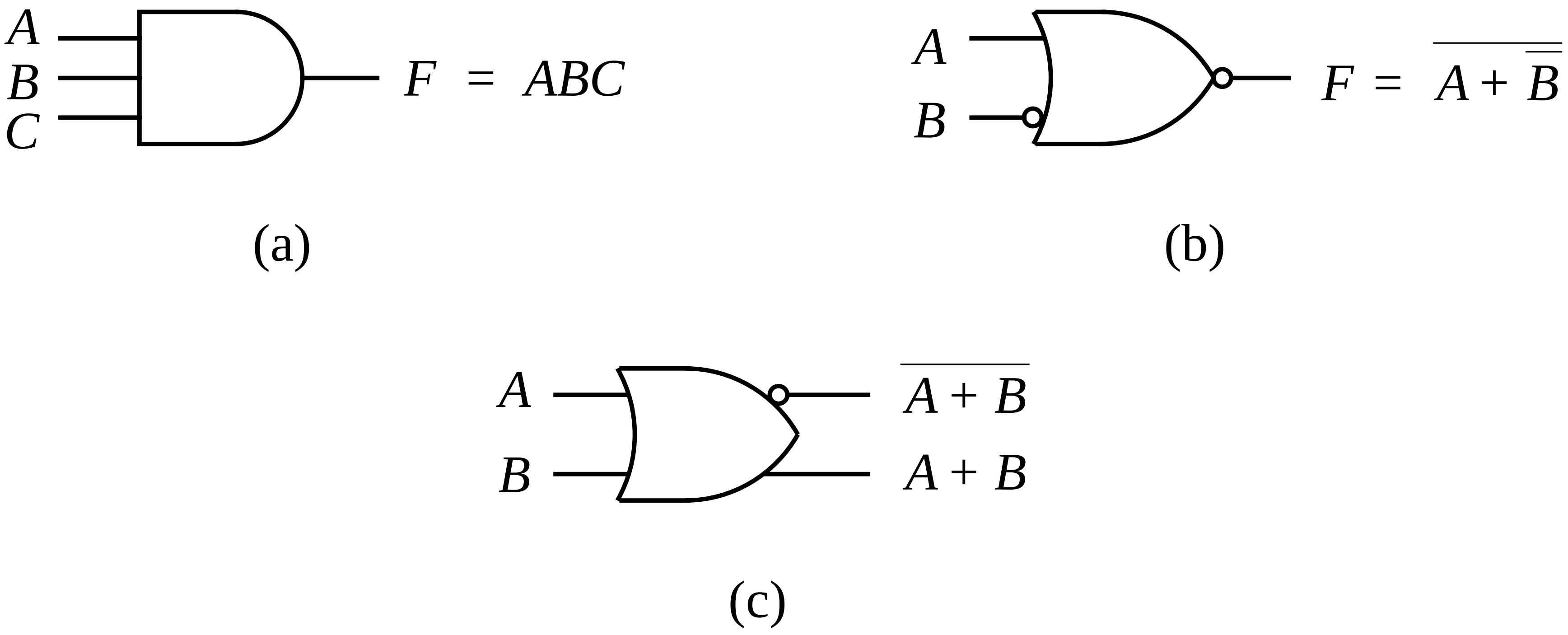
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



$F = A \odot B$

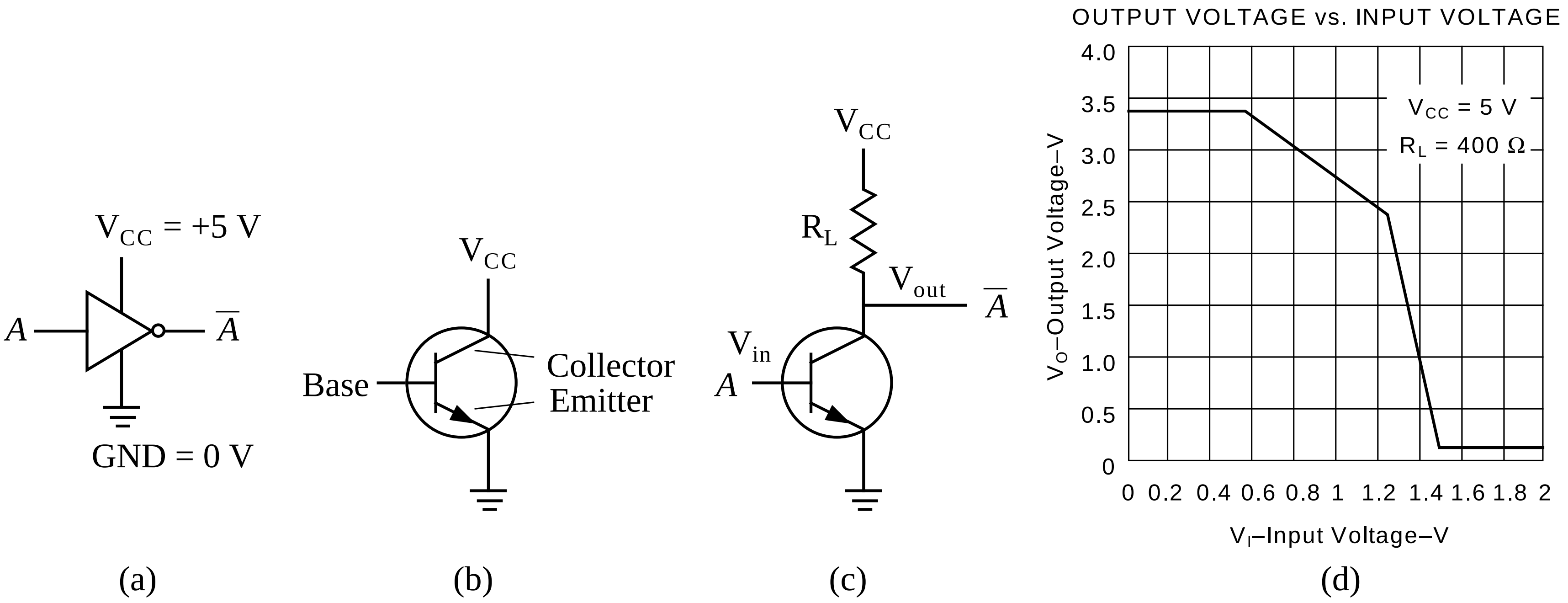
Exclusive-NOR (XNOR)

Variations of Logic Gate Symbols



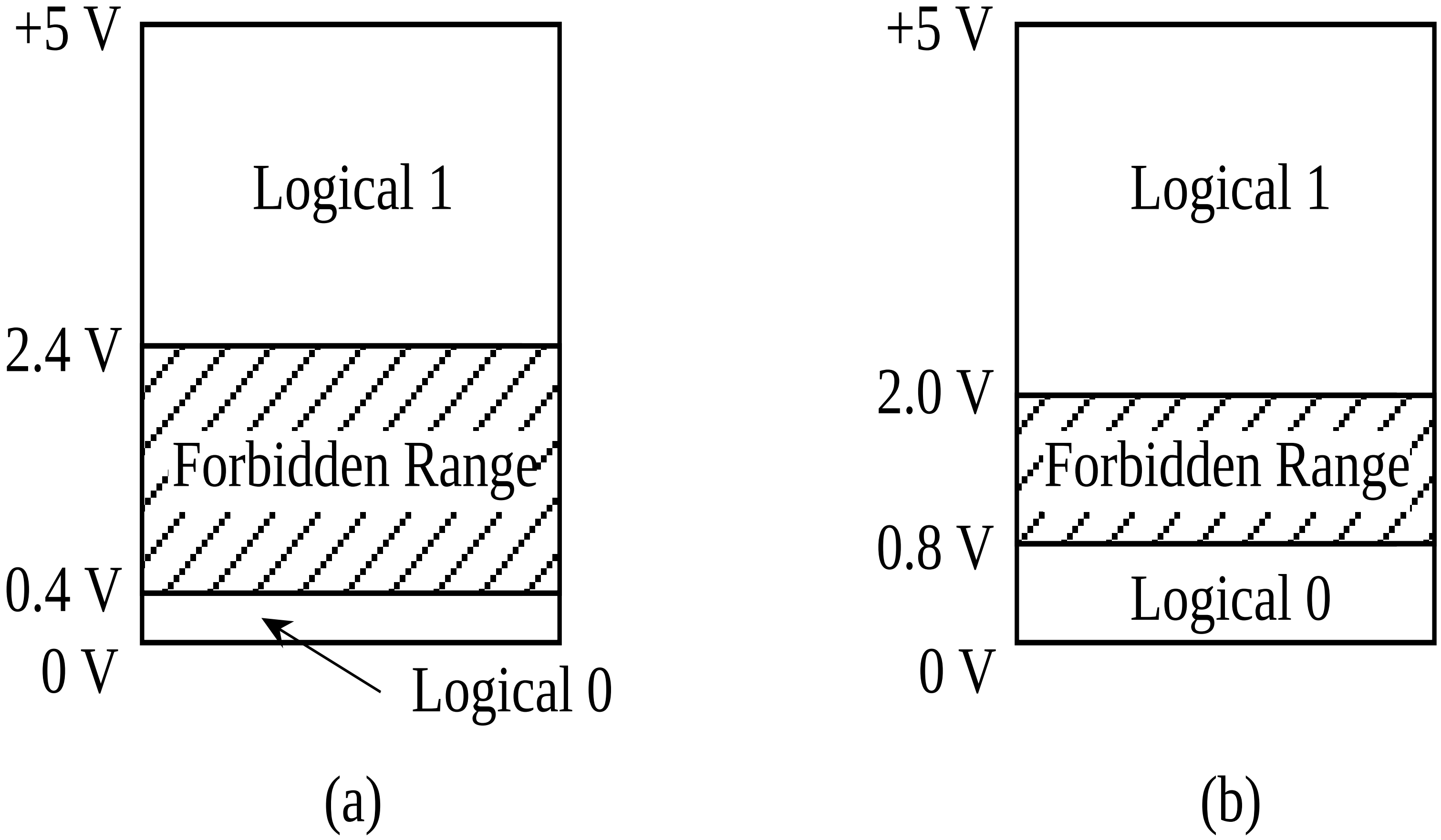
(a) 3 inputs (b) A Negated input (c) Complementary outputs

Transistor Operation of Inverter



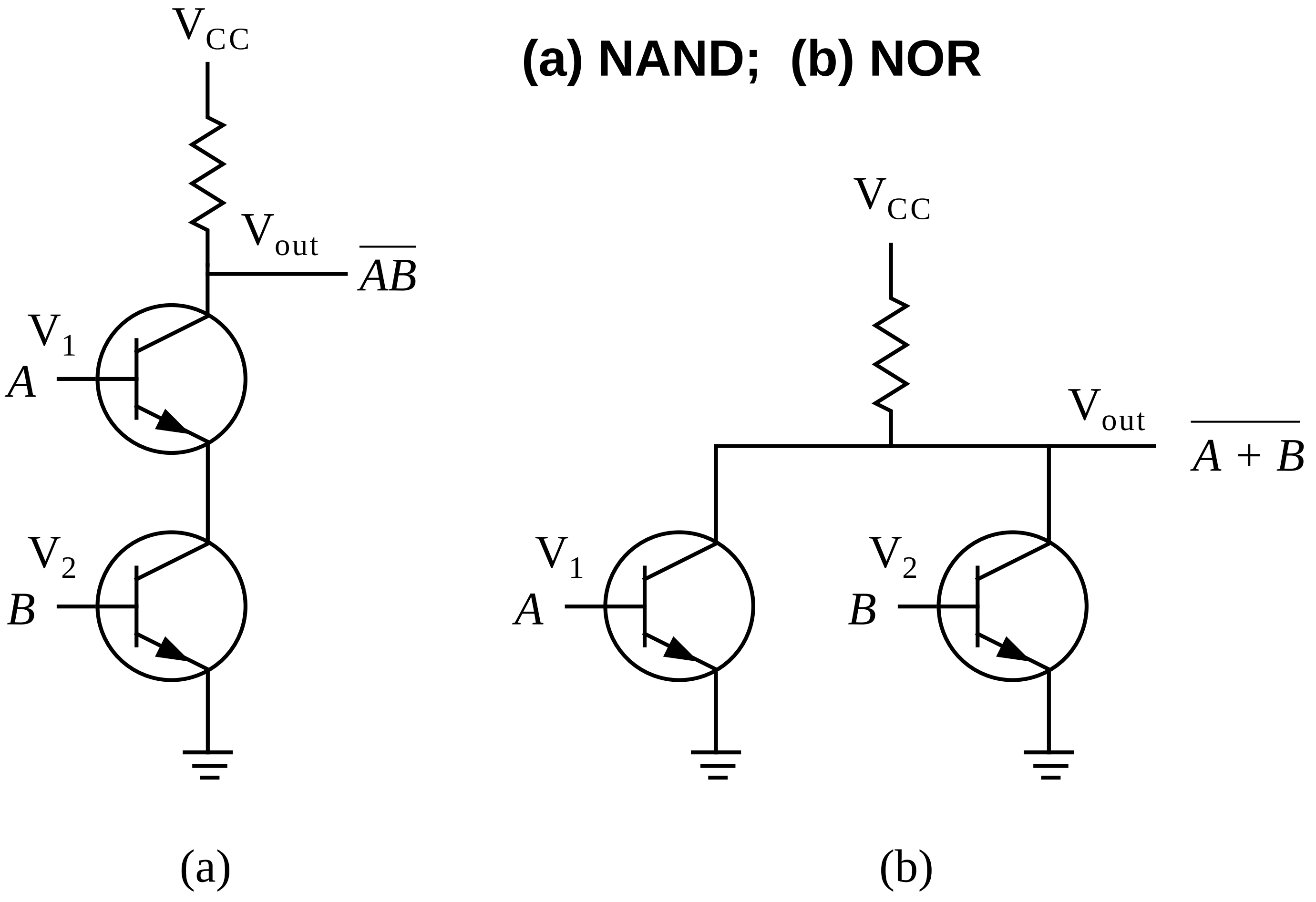
(a) Inverter showing power terminals; (b) transistor symbol; (c) transistor configured as an inverter; (d) inverter transfer function.

Assignments of 0 and 1 to Voltages



Transistor Operation of Logic Gates

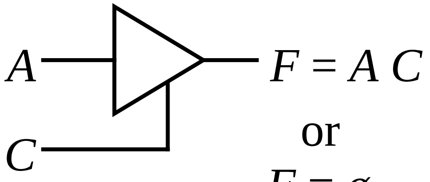
(a) NAND; (b) NOR



Tri-State Buffers

- **Outputs can be 0, 1, or “electrically disconnected.”**

C	A	F
0	0	ø
0	1	ø
1	0	0
1	1	1



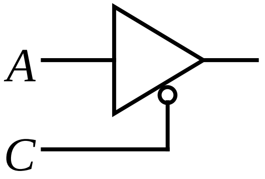
$F = A\,C$

or

$F = ø$

Tri-state buffer

C	A	F
0	0	0
0	1	1
1	0	ø
1	1	ø



$F = A\,\overline{C}$

or

$F = ø$

Tri-state buffer, inverted control

Properties of Boolean Algebra

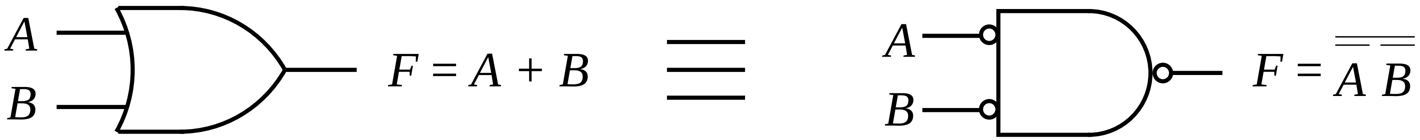
- **Principle of duality: The dual of a Boolean function is obtained by replacing AND with OR and OR with AND, 1s with 0s, and 0s with 1s.**

	Relationship	Dual	Property
Postulates	$A\,B = B\,A$	$A + B = B + A$	Commutative
	$A\,(B + C) = A\,B + A\,C$	$A + B\,C = (A + B)\,(A + C)$	Distributive
	$1\,A = A$	$0 + A = A$	Identity
	$A\overline{A} = 0$	$A + \overline{A} = 1$	Complement
Theorems	$0\,A = 0$	$1 + A = 1$	Zero and one theorems
	$A\,A = A$	$A + A = A$	Idempotence
	$A\,(B\,C) = (A\,B)\,C$	$A + (B + C) = (A + B) + C$	Associative
	$\overline{\overline{A}} = A$		Involution
	$\overline{A\,B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$	DeMorgan's Theorem
	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$	Consensus Theorem
	$A\,(A + B) = A$	$A + A\,B = A$	Absorption Theorem

DeMorgan’s Theorem

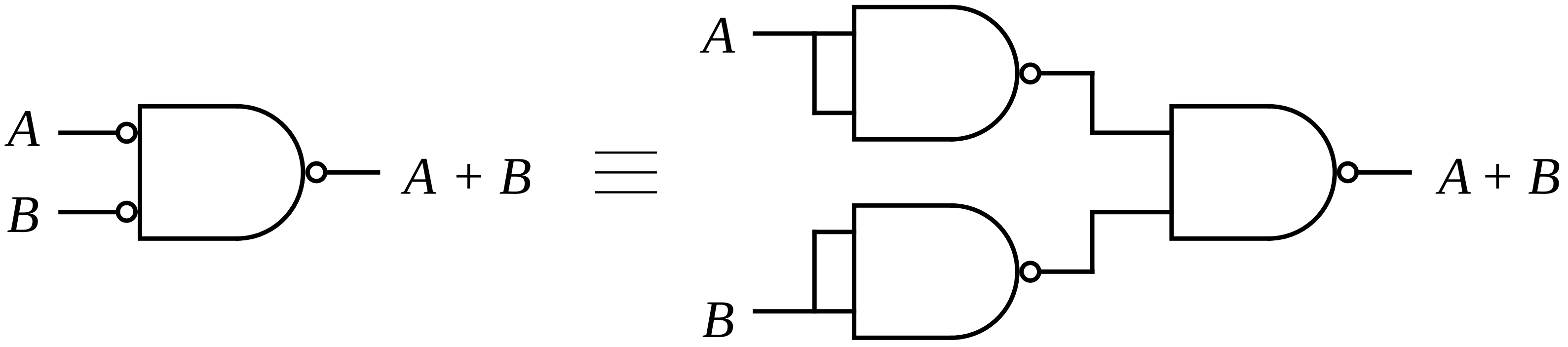
A	B	$\overline{A\,B}$ =	$\overline{A} + \overline{B}$	$\overline{A + B}$ =	$\overline{A}\,\overline{B}$
0	0	1	1	1	1
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	0	0

DeMorgan’s theorem: $A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$



All-NAND Implementation of OR

- **NAND alone implements all other Boolean logic gates.**



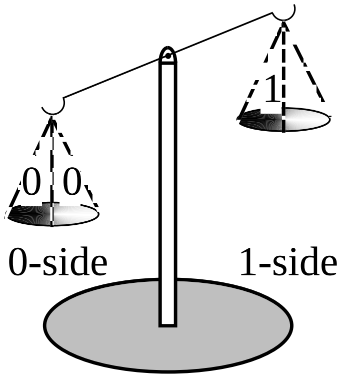
Sum-of-Products Form: The Majority Function

- **The SOP form for the 3-input majority function is:**

$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = m3 + m5 + m6 + m7 = \Sigma\,(3, 5, 6, 7).$$

- **Each of the 2^n terms are called *minterms*, ranging from 0 to 2^n - 1.**
- **Note relationship between minterm number and boolean value.**

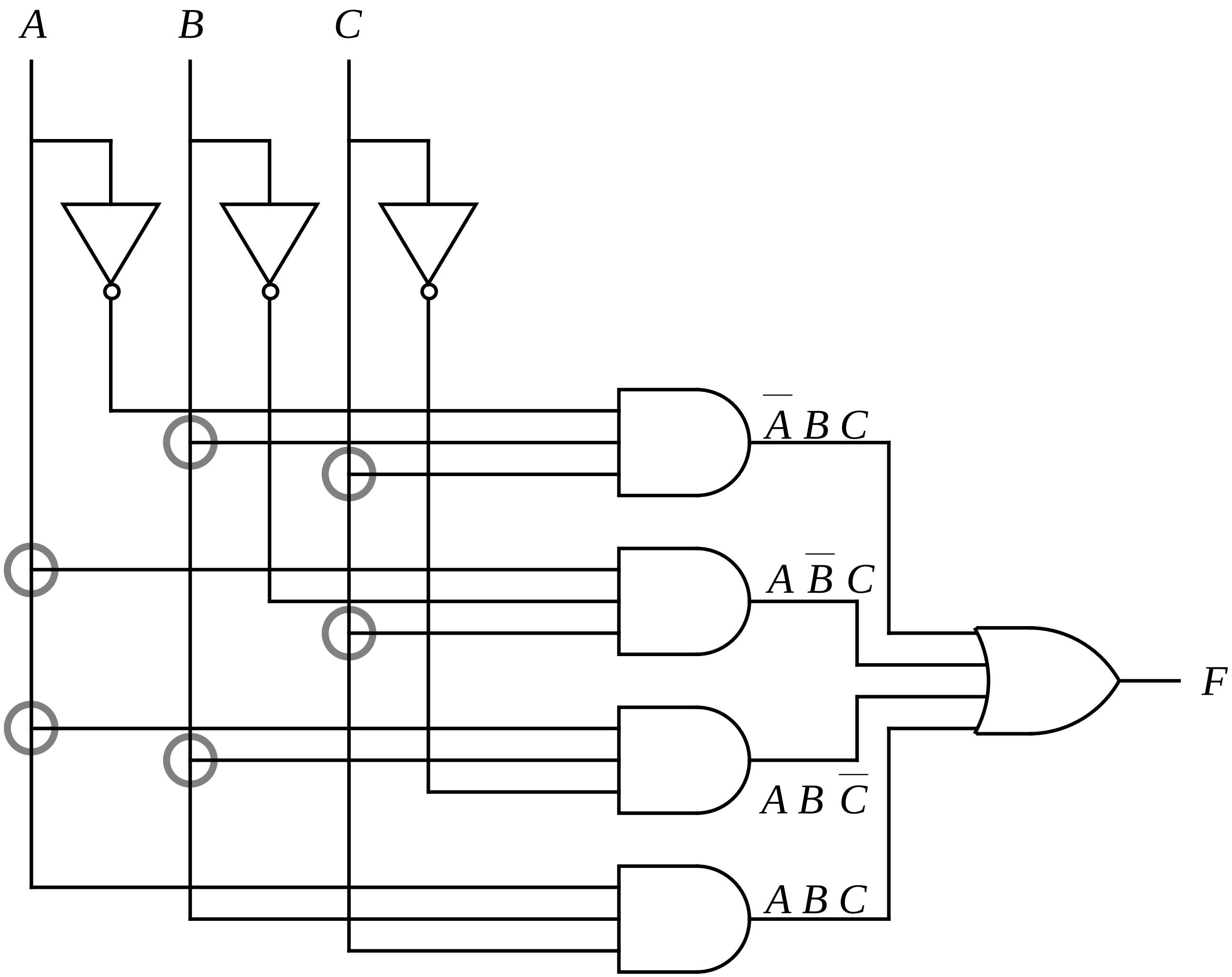
Minterm Index	*A*	*B*	*C*	*F*
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



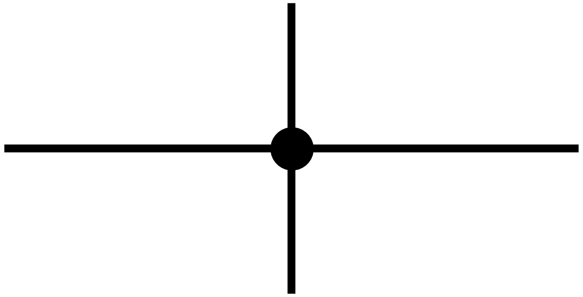
A balance tips to the left or right depending on whether there are more 0's or 1's.

AND-OR Implementation of Majority

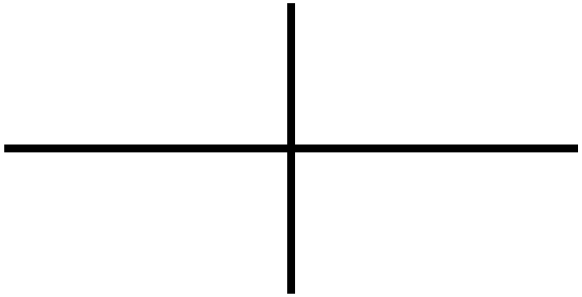
- **Gate count is 8, gate input count is 19.**



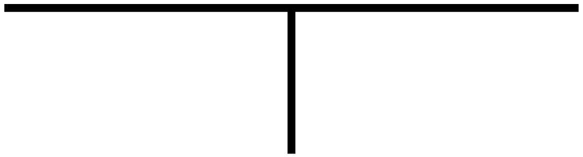
Notation Used at Circuit Intersections



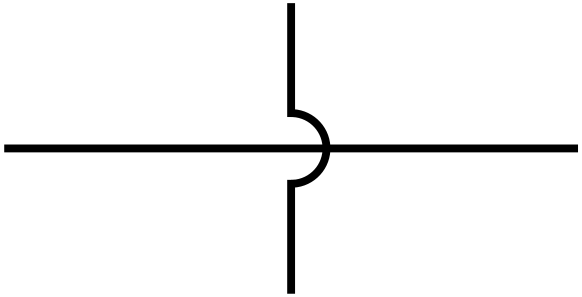
Connection



No connection

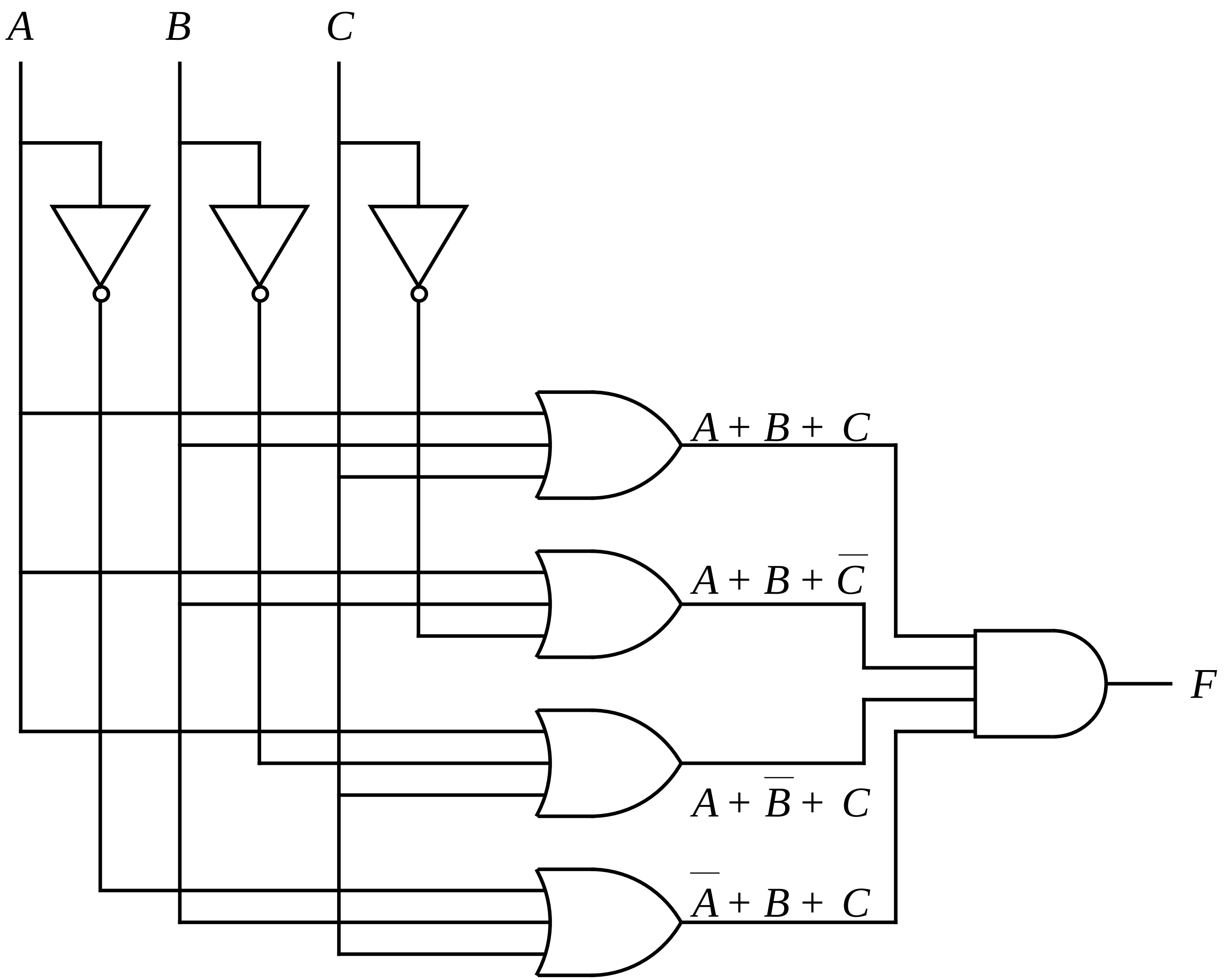


Connection



No connection

OR-AND Implementation of Majority



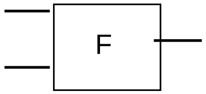
Positive/Negative Logic Assignments

- **Positive logic: logic 1 is represented by high voltage; logic 0 is represented by low voltage.**
- **Negative logic: logic 0 is represented by high voltage; logic 1 is represented by low voltage.**

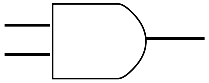
Gate Logic: Positive vs. Negative Logic

Normal Convention: Postive Logic/Active High
Low Voltage = 0; High Voltage = 1

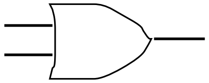
Alternative Convention sometimes used: Negative Logic/Active Low



Voltage Truth Table



Positive Logic



Negative Logic

Voltage Truth Table			Positive Logic			Negative Logic		
A	B	F	A	B	F	A	B	F
low	low	low	0	0	0	1	1	1
low	high	low	0	1	0	1	0	1
high	low	low	1	0	0	0	1	1
high	high	high	1	1	1	0	0	0

Behavior in terms of Electrical Levels

Two Alternative Interpretations

Positive Logic AND
Negative Logic OR

Dual Operations

Positive/Negative Logic Assignments (Cont')

Voltage Levels

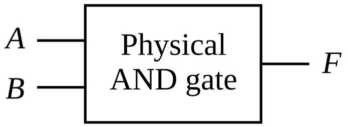
A	*B*	*F*
low	low	low
low	high	low
high	low	low
high	high	high

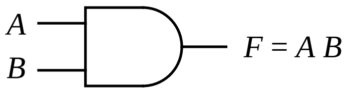
Positive Logic Levels

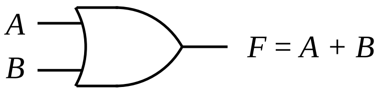
A	*B*	*F*
0	0	0
0	1	0
1	0	0
1	1	1

Negative Logic Levels

A	*B*	*F*
1	1	1
1	0	1
0	1	1
0	0	0







Voltage Levels

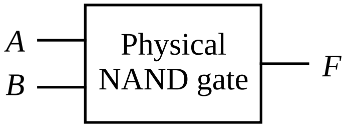
A	*B*	*F*
low	low	high
low	high	high
high	low	high
high	high	low

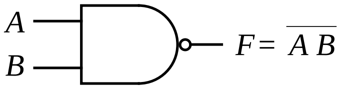
Positive Logic Levels

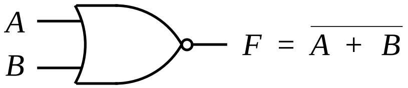
A	*B*	*F*
0	0	1
0	1	1
1	0	1
1	1	0

Negative Logic Levels

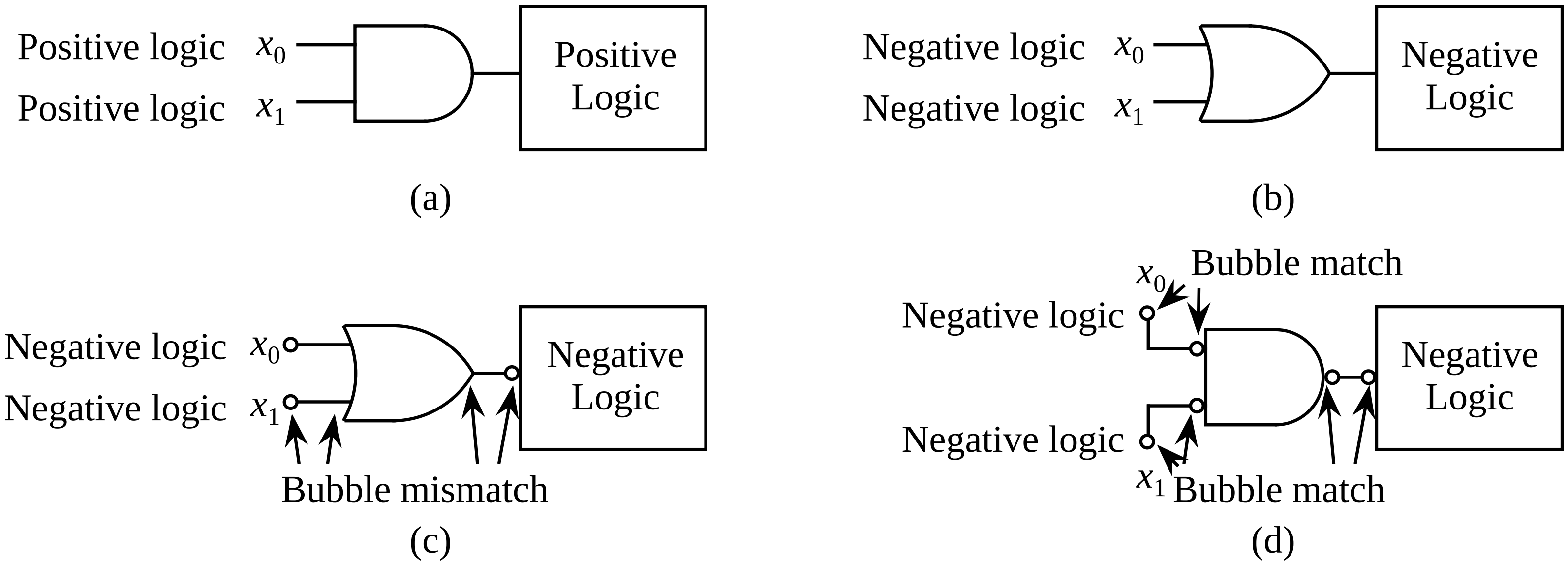
A	*B*	*F*
1	1	0
1	0	0
0	1	0
0	0	1







Bubble Matching



Example Data Sheet

- **Simplified data sheet for 7400 NAND gate, adapted from *Texas Instruments TTL Databook* [Texas Instruments, 1988]**

SN7400 QUADRUPLE 2-INPUT POSITIVE-NAND GATES

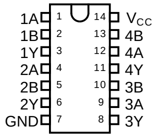
description

These devices contain four independent 2-input NAND gates.

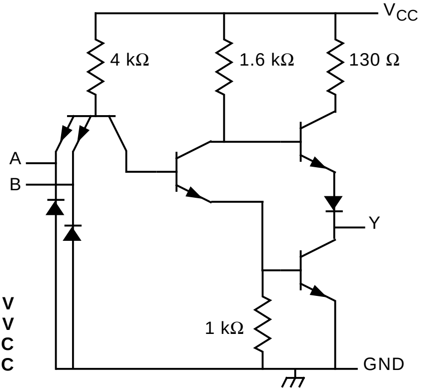
function table (each gate)

INPUTS		OUTPUT
A	**B**	**Y**
H	H	L
L	X	H
X	L	H

package (top view)



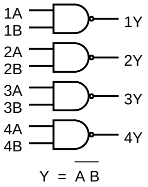
schematic (each gate)



absolute maximum ratings

Supply voltage, VCC	**7 V**
Input voltage:	**5.5 V**
Operating free-air temperature range:	**0 ˚C to 70 ˚C**
Storage temperature range	**– 65 ˚C to 150 ˚C**

logic diagram (positive logic)



recommended operating conditions

		MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage	4.75	5	5.25	V
V_{IH}	High-level input voltage	2			V
V_{IL}	Low-level input voltage			0.8	V
I_{OH}	High-level output current			– 0.4	mA
I_{OL}	Low-level output current			16	mA
T_A	Operating free-air temperature	0		70	˚C

electrical characteristics over recommended operating free-air temperature range

		MIN	TYP	MAX	UNIT
V_{OH}	V_{CC} = MIN, V_{IL} = 0.8 V, I_{OH} = – 0.4 mA	2.4	3.4		V
V_{OL}	V_{CC} = MIN, V_{IH} = 2 V, I_{OL} = 16 mA		0.2	0.4	V
I_{IH}	V_{CC} = MAX, V_I = 2.4 V			40	µA
I_{IL}	V_{CC} = MAX, V_I = 0.4 V			– 1.6	mA
I_{CCH}	V_{CC} = MAX, V_I = 0 V		4	8	mA
I_{CCL}	V_{CC} = MAX, V_I = 4.5 V		12	22	mA

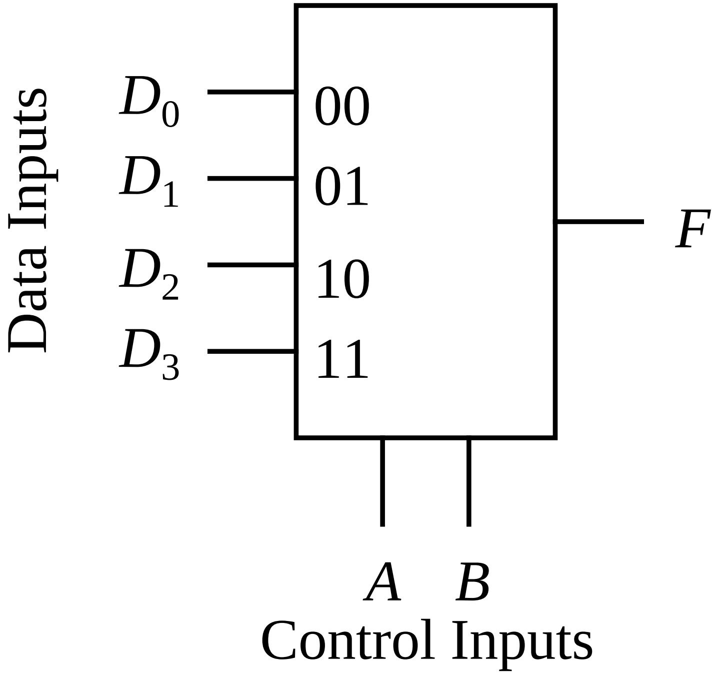
switching characteristics, V_{CC} = 5 V, T_A = 25˚ C

PARAMETER	FROM (input)	TO (output)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
t_{PLH}	A or B	Y	R_L = 400 Ω C_L = 15 pF		11	22	ns
t_{PHL}					7	15	ns

Digital Components

- High level digital circuit designs are normally created using collections of logic gates referred to as *components*, rather than using individual logic gates.
- Levels of integration (numbers of gates) in an integrated circuit (IC) can roughly be considered as:
  - Small scale integration (SSI): 10-100 gates.
  - Medium scale integration (MSI): 100 to 1000 gates.
  - Large scale integration (LSI): 1000-10,000 logic gates.
  - Very large scale integration (VLSI): 10,000-upward logic gates.
  - These levels are approximate, but the distinctions are useful in comparing the relative complexity of circuits.

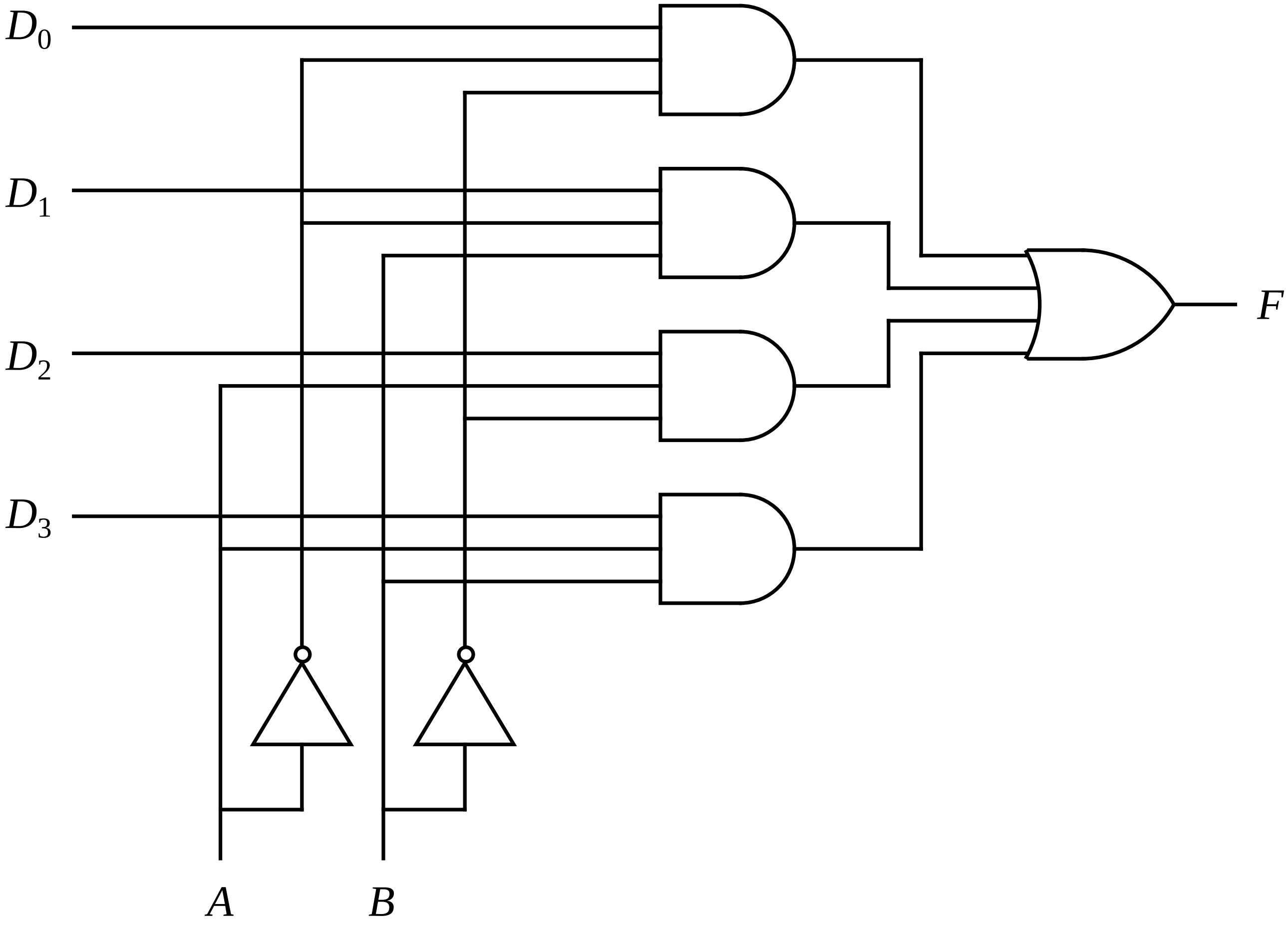
Multiplexer



A	B	F
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

$$F = \overline{A}\,\overline{B}\,D_0 + \overline{A}\,B\,D_1 + A\,\overline{B}\,D_2 + A\,B\,D_3$$

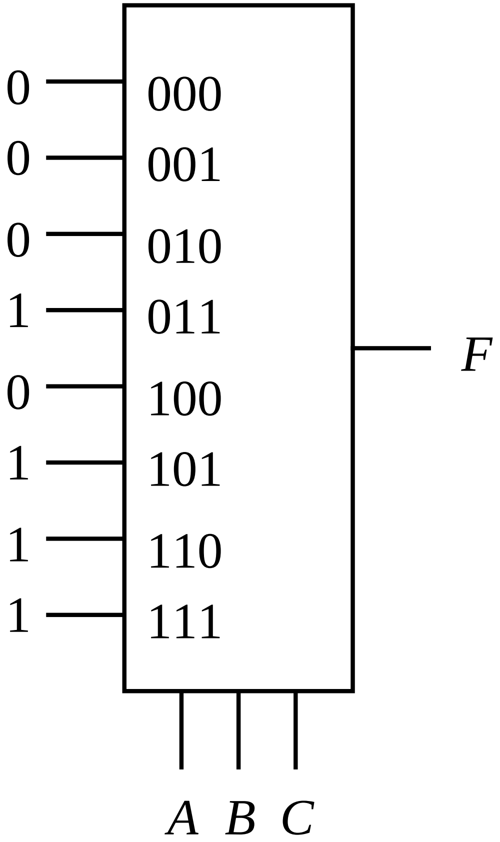
AND-OR Implementation of MUX



MUX Implementation of Majority

- **Principle: Use the 3 MUX control inputs to select (one at a time) the 8 data inputs.**

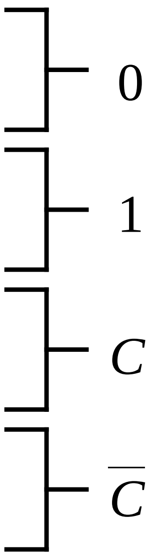
A	*B*	*C*	*M*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

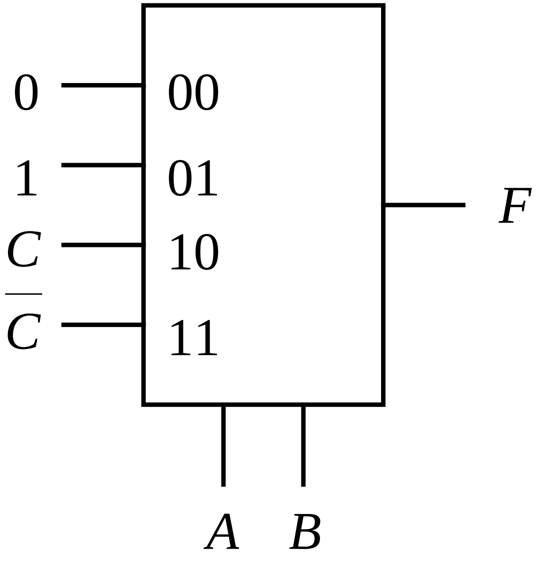


4-to-1 MUX Implements 3-Var Function

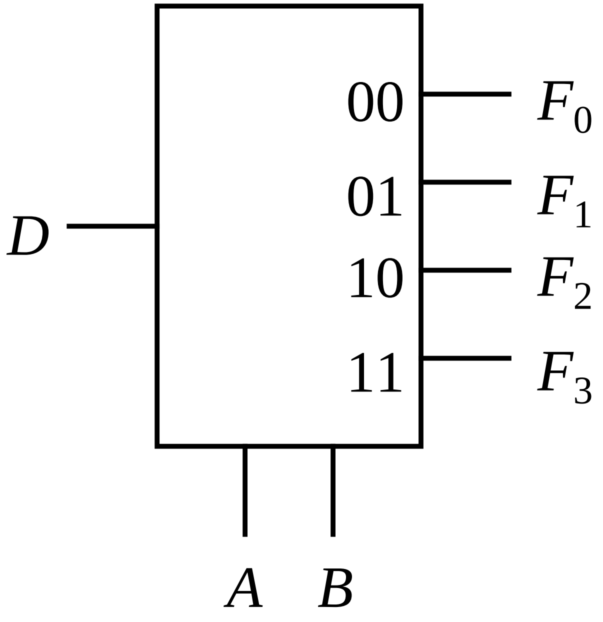
- **Principle: Use the A and B inputs to select a pair of minterms. The value applied to the MUX data input is selected from {0, 1, C, $\overline{C}$} to achieve the desired behavior of the minterm pair.**

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





Demultiplexer

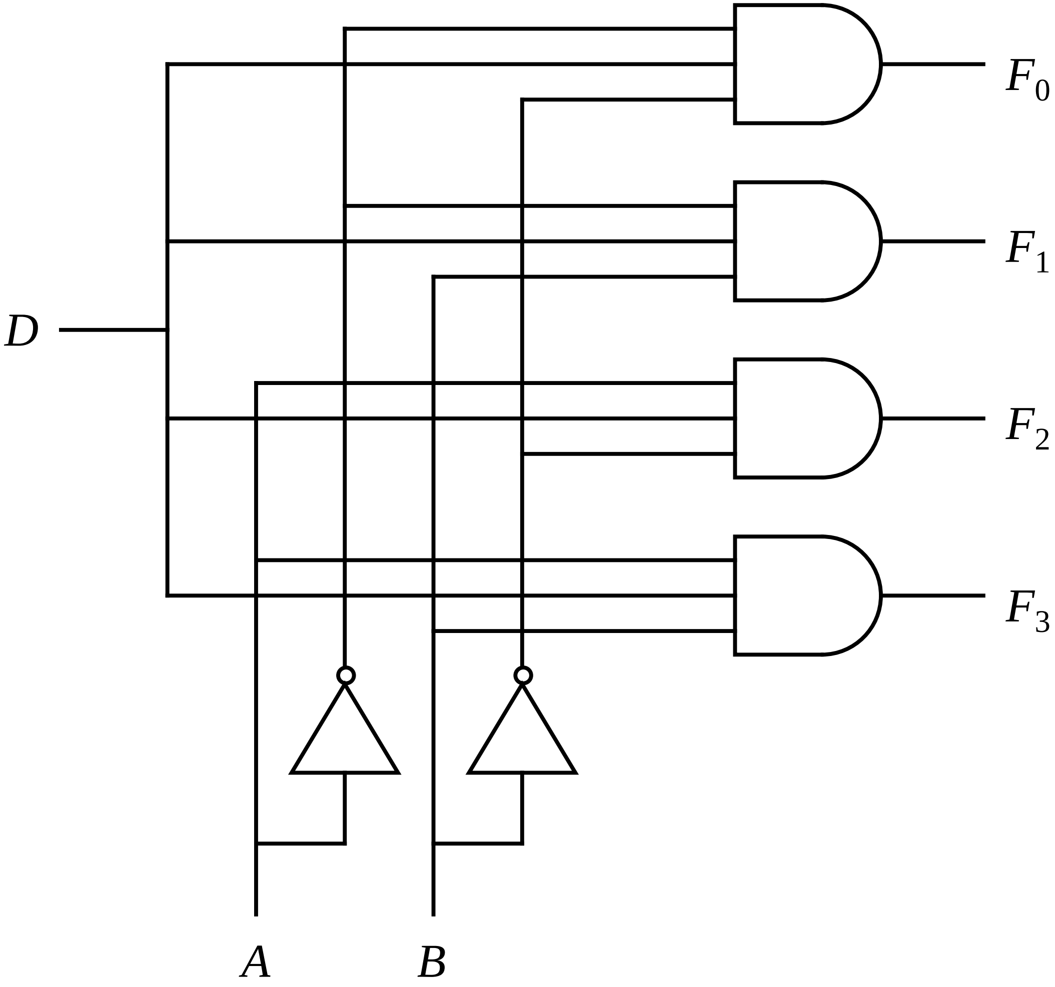


D	A	B	F_0	F_1	F_2	F_3
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

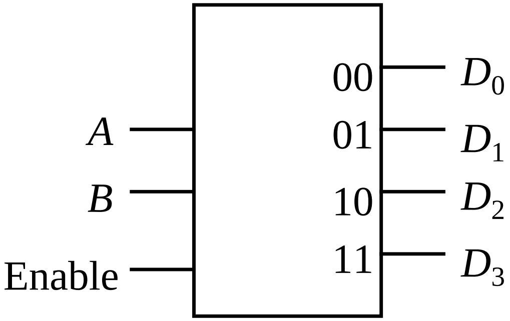
$$F_0 = D\overline{A}\,\overline{B} \qquad F_2 = DA\overline{B}$$

$$F_1 = D\overline{A}B \qquad F_3 = DAB$$

Gate-Level Implementation of DEMUX



Decoder

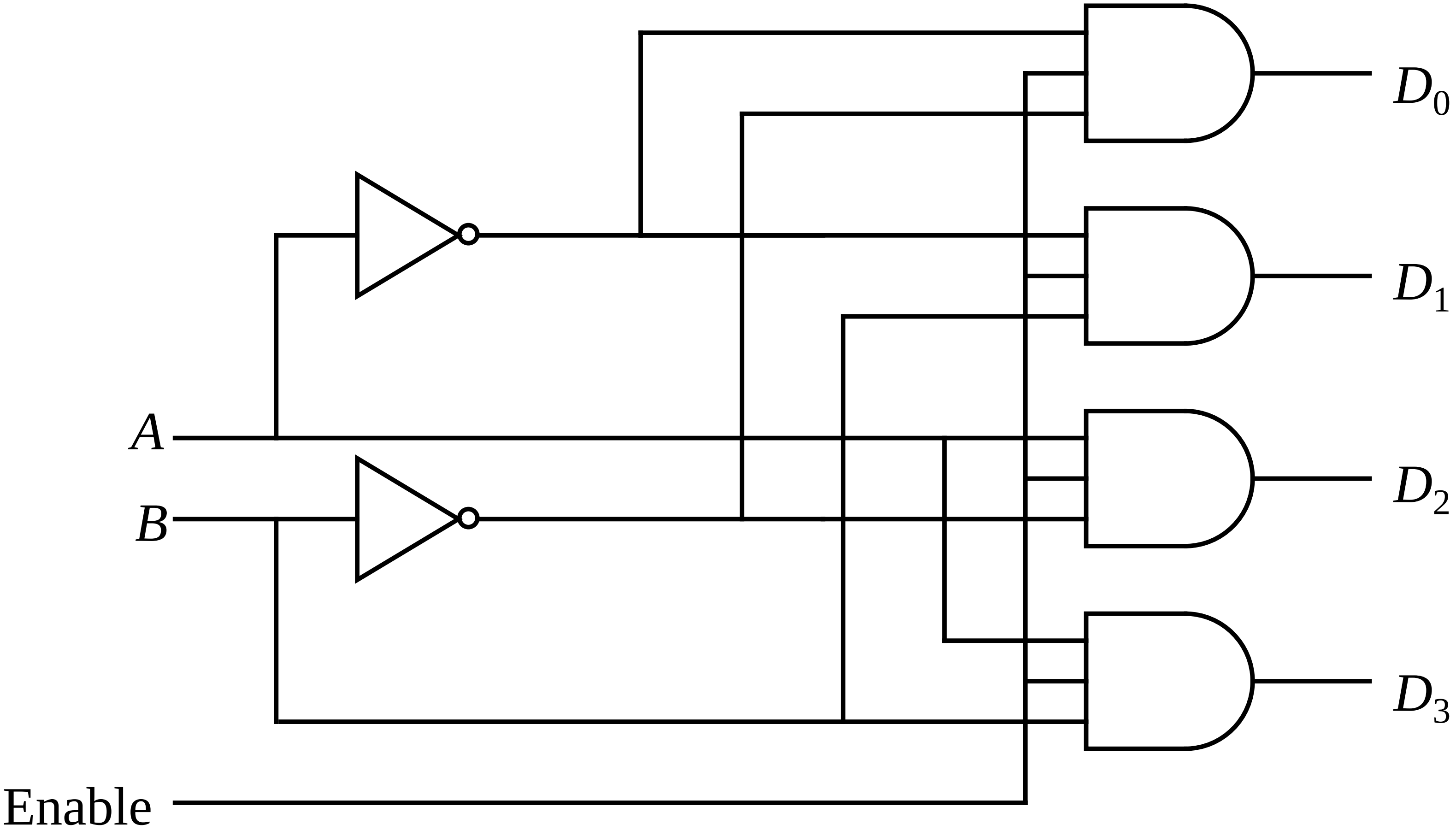


Enable = 1					
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Enable = 0					
A	B	D_0	D_1	D_2	D_3
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

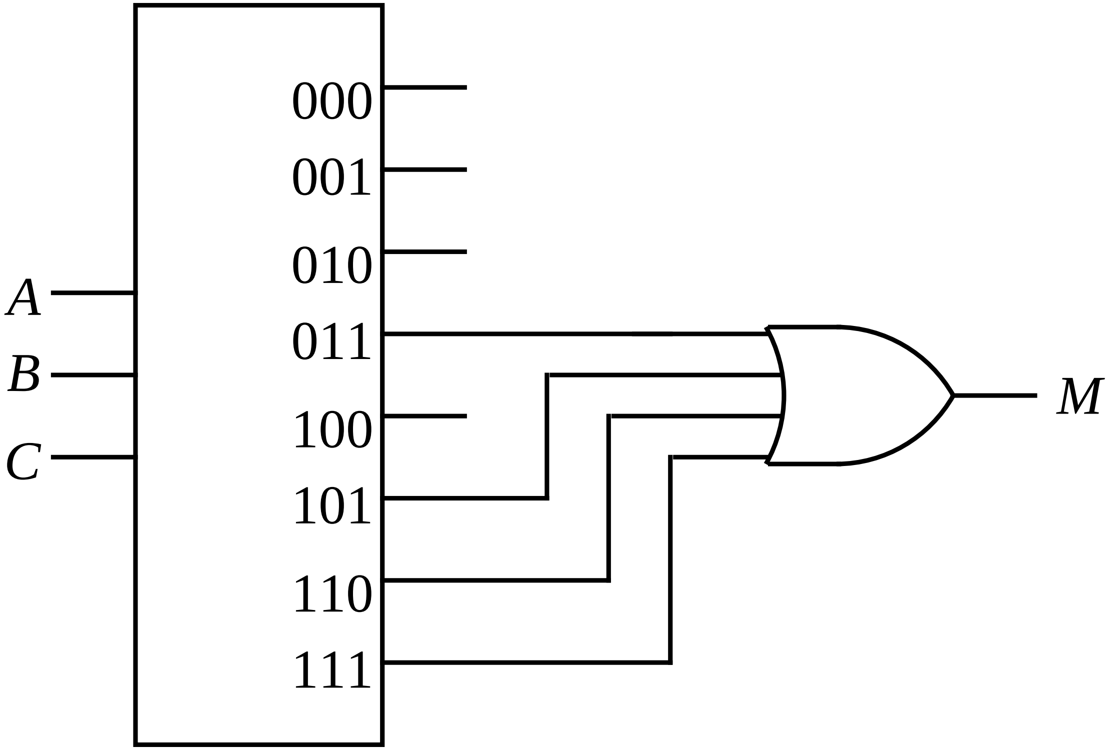
$D_0 = \overline{A}\,\overline{B}$ $D_1 = \overline{A}\,B$ $D_2 = A\,\overline{B}$ $D_3 = A\,B$

Gate-Level Implementation of Decoder



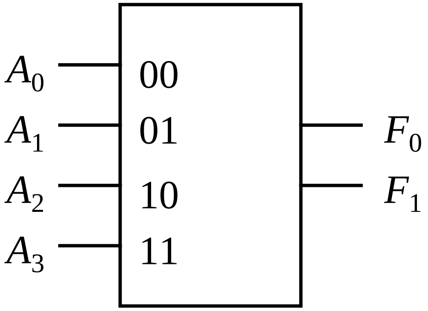
Decoder Implementation of Majority Function

- **Note that the enable input is not always present. We use it when discussing decoders for memory.**



Priority Encoder

- An encoder translates a set of inputs into a binary encoding.
- Can be thought of as the converse of a decoder.
- A priority encoder imposes an order on the inputs.
- A_i has a higher priority than A_{i+1}

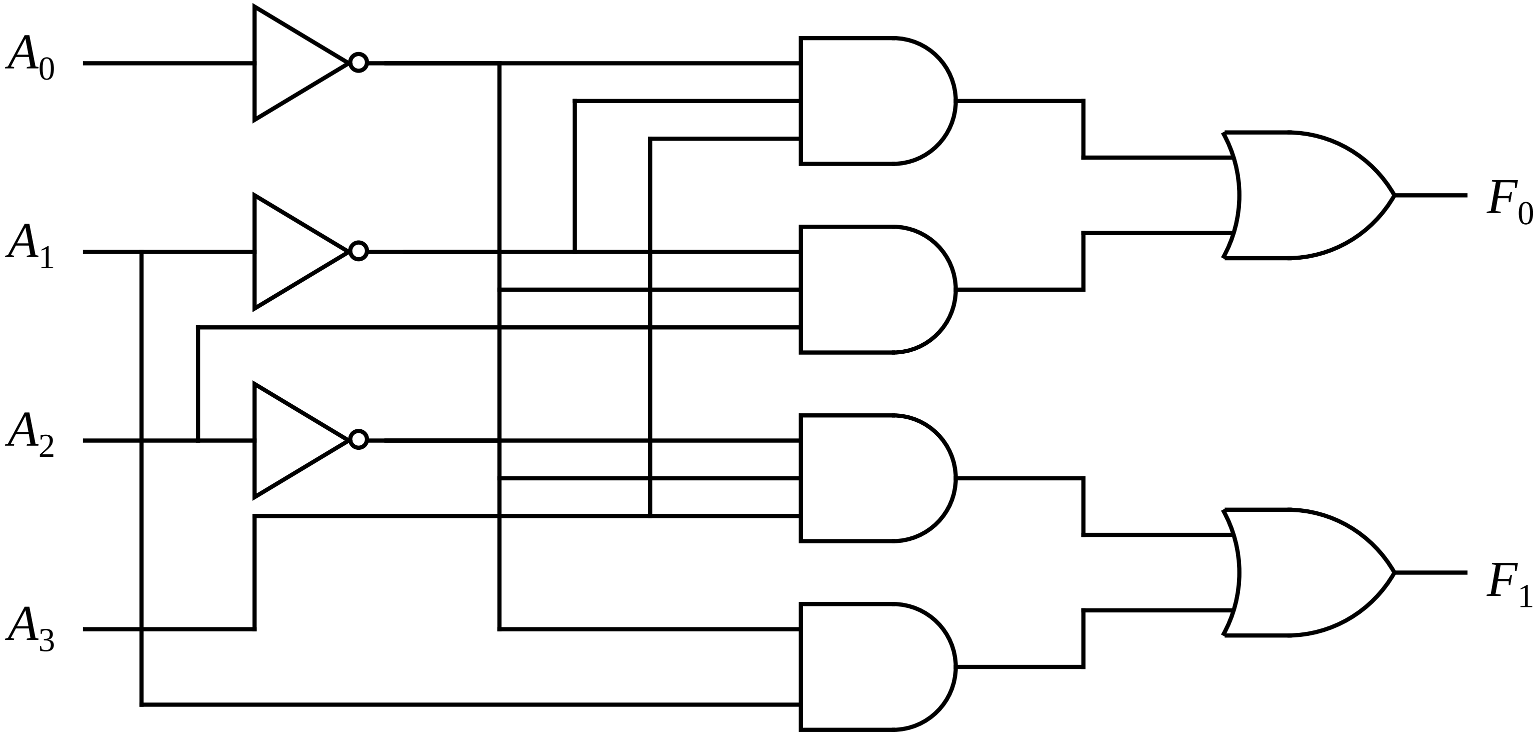


$$F_0 = \overline{A_0}\,\overline{A_1}\,A_3 + \overline{A_0}\,\overline{A_1}\,A_2$$

$$F_1 = \overline{A_0}\,\overline{A_2}\,A_3 + \overline{A_0}\,A_1$$

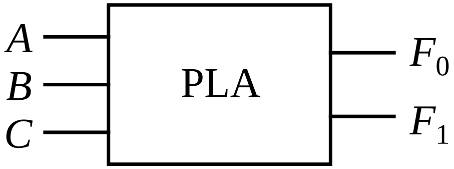
A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0

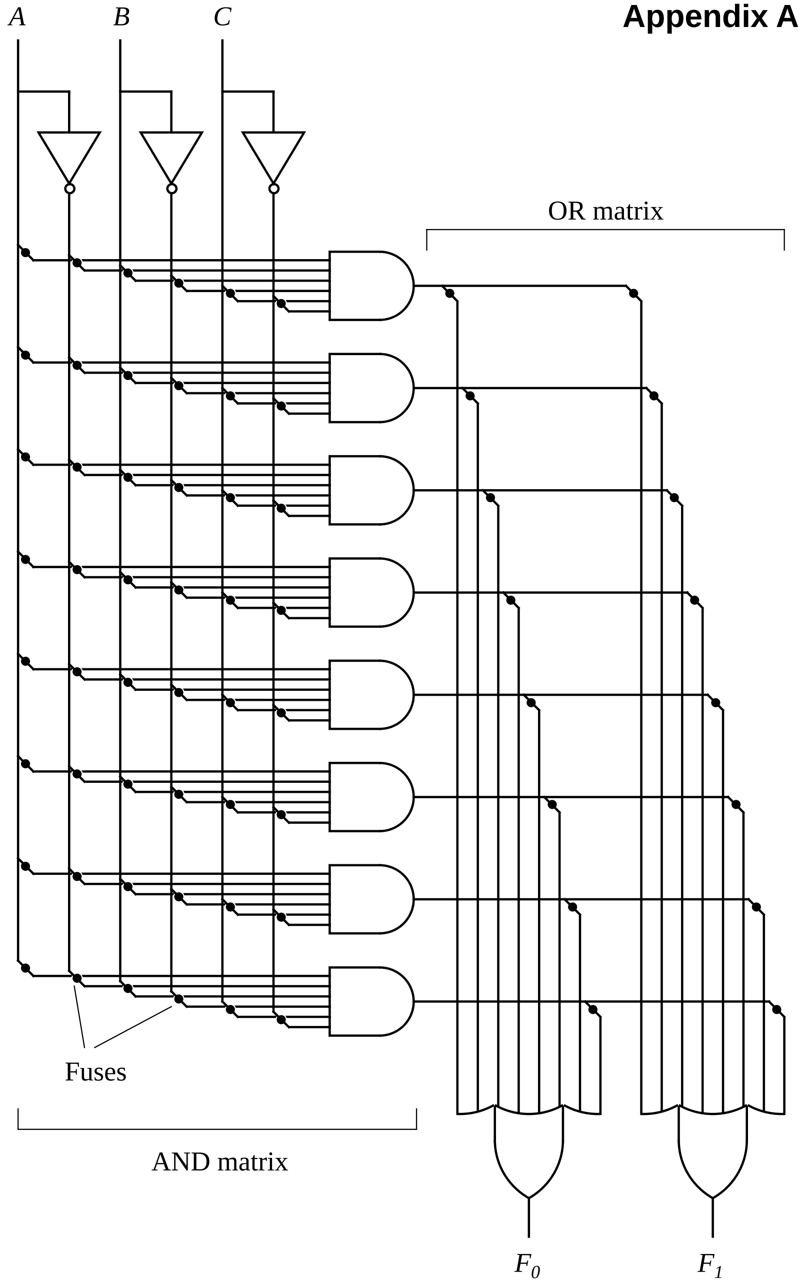
AND-OR Implementation of Priority Encoder



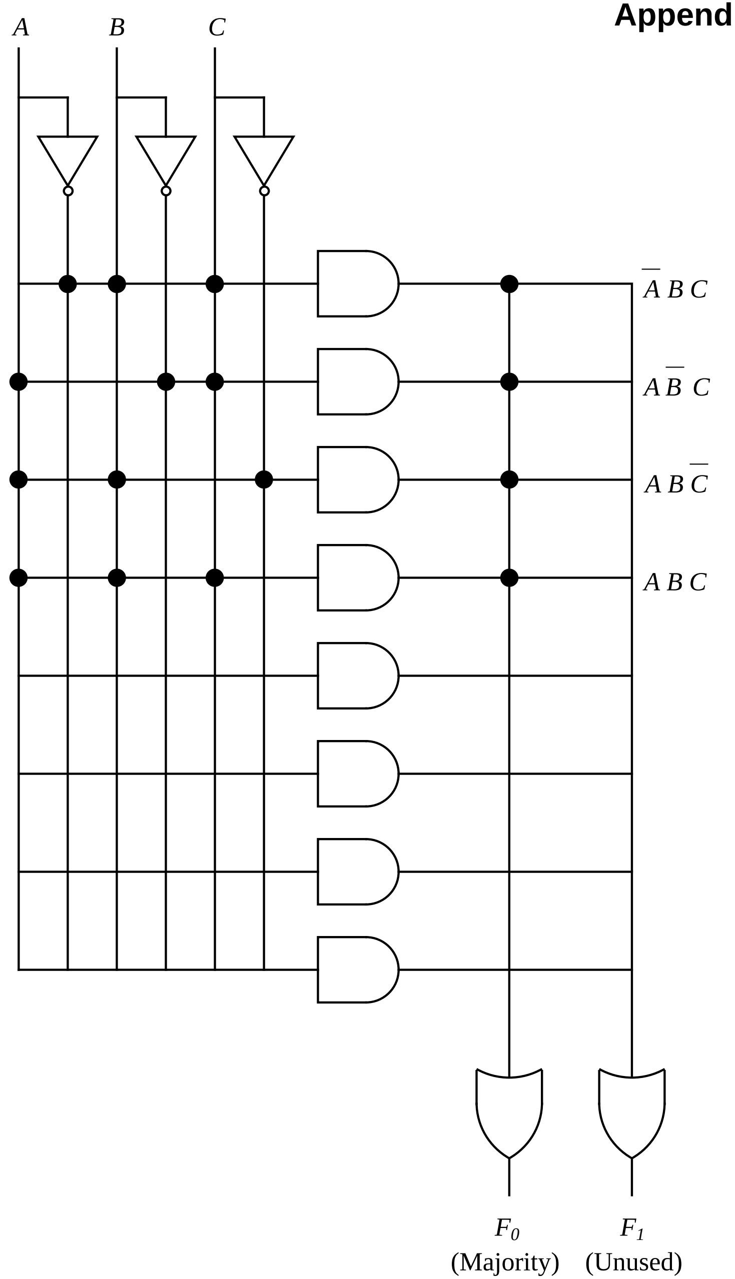
Programmable Logic Array

- A PLA is a customizable AND matrix followed by a customizable OR matrix.
- Black box view of PLA:

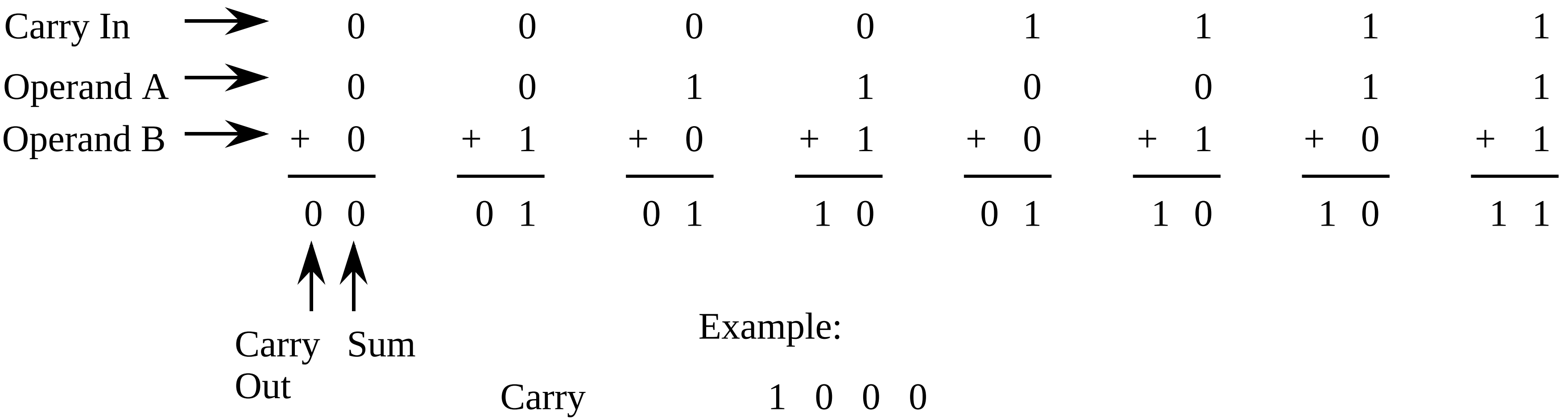




Simplified Representation of PLA Implementation of Majority Function



Example: Ripple-Carry Addition



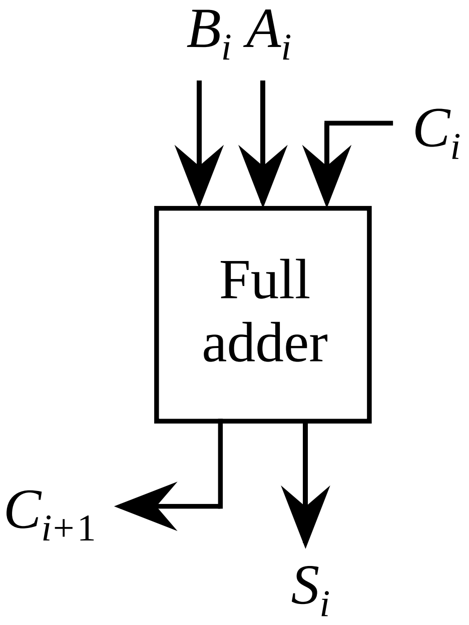
Carry In →	0	0	0	0	1	1	1	1
Operand A →	0	0	1	1	0	0	1	1
Operand B →	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1	+ 0	+ 1
	0 0	0 1	0 1	1 0	0 1	1 0	1 0	1 1

Example:

Carry		1	0	0	0
Operand A		0	1	0	0
Operand B	+	0	1	1	0
Sum		1	0	1	0

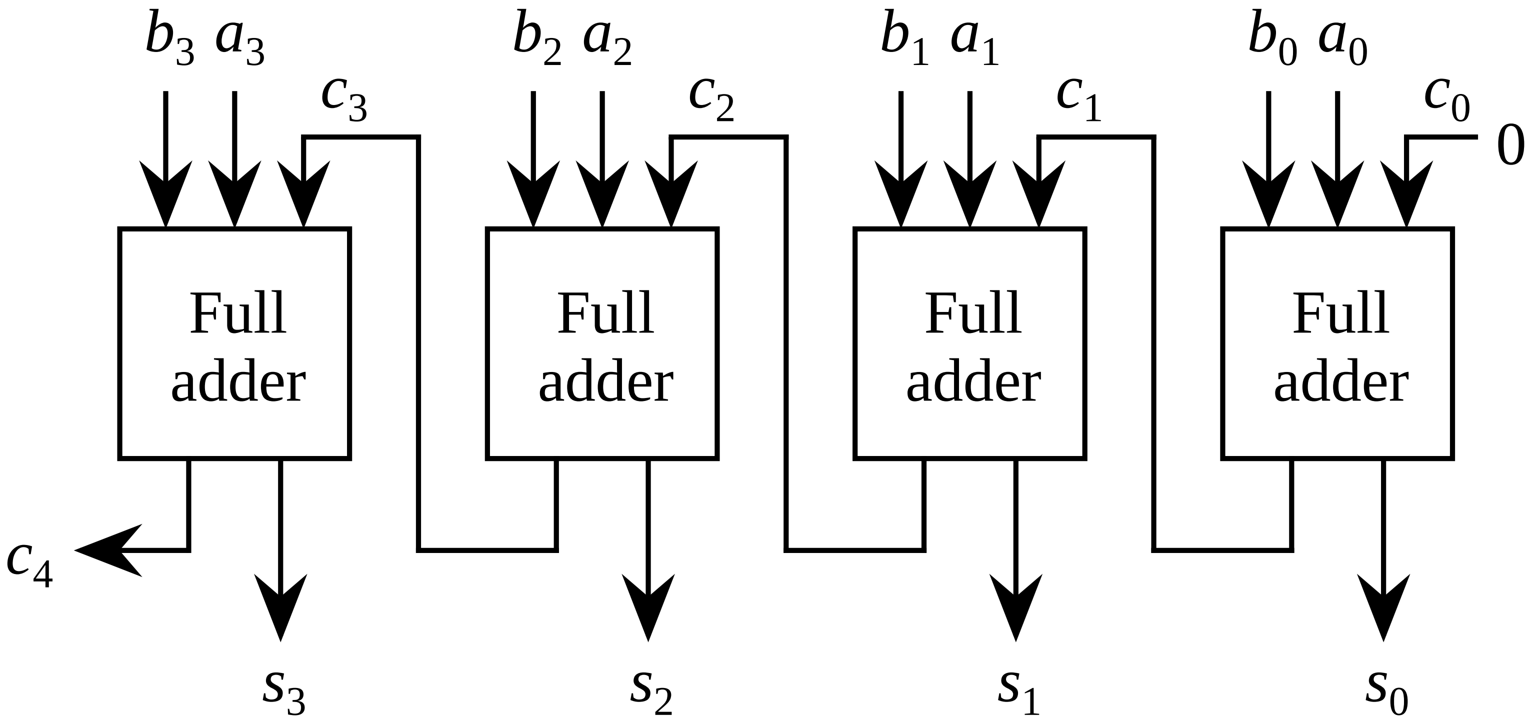
Full Adder

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

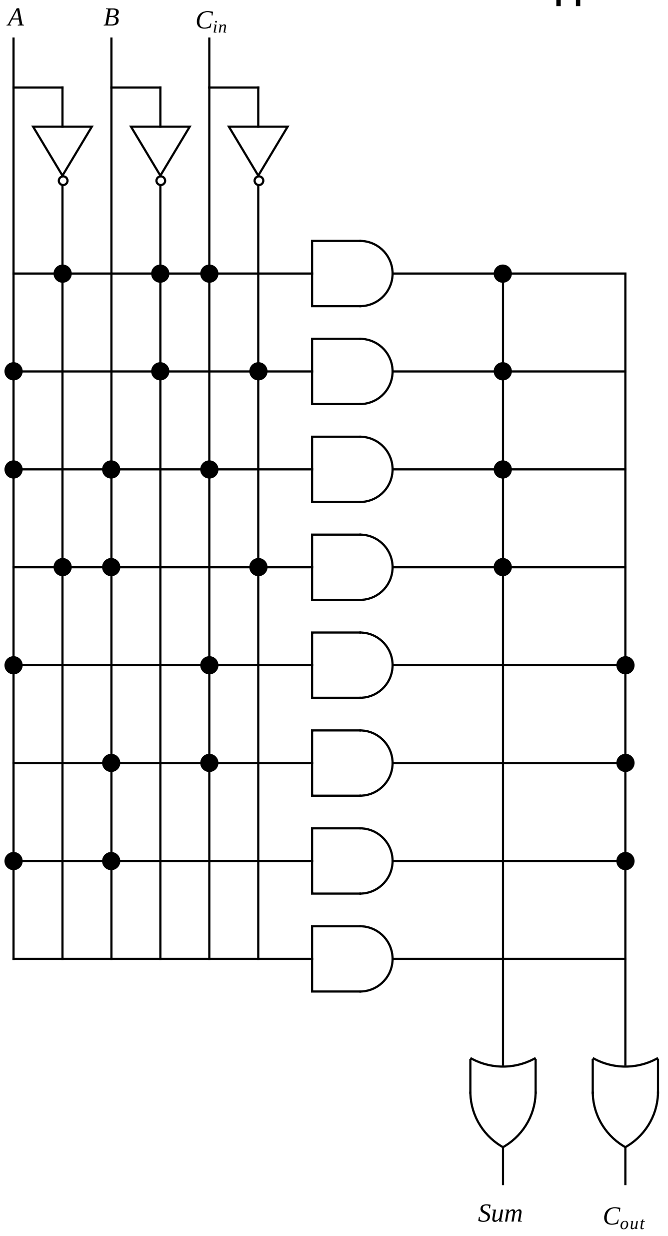


Four-Bit Ripple-Carry Adder

- Four full adders connected in a ripple-carry chain form a four-bit ripple-carry adder.



PLA Realization of Full Adder

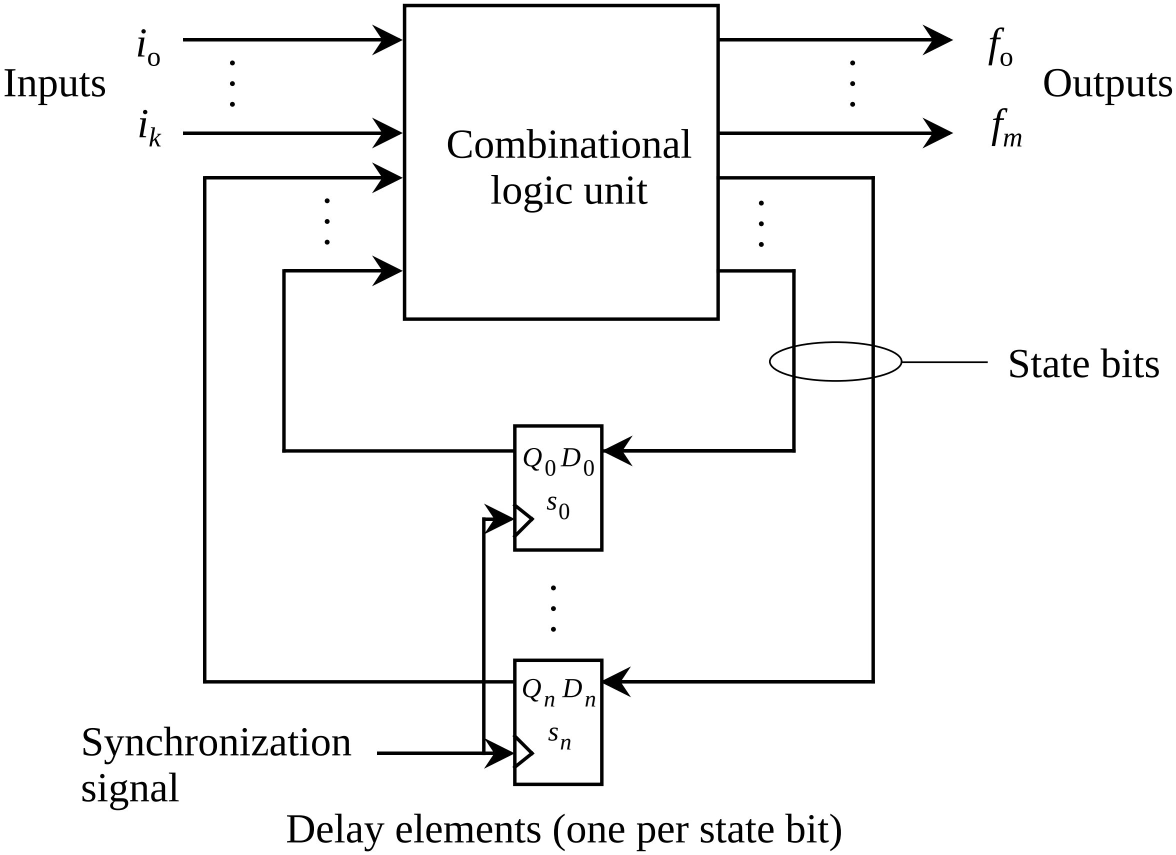


Sequential Logic

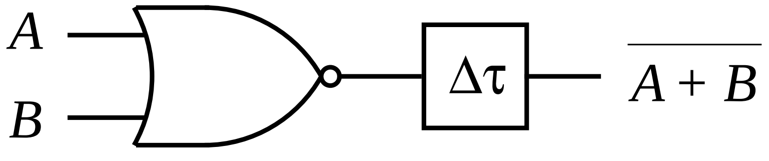
- The combinational logic circuits we have been studying so far have no memory. The outputs always follow the inputs.
- There is a need for circuits with memory, which behave differently depending upon their previous state.
- An example is a vending machine, which must remember how many and what kinds of coins have been inserted. The machine should behave according to not only the current coin inserted, but also upon how many and what kinds of coins have been inserted previously.
- These are referred to as *finite state machines*, because they can have at most a finite number of states.

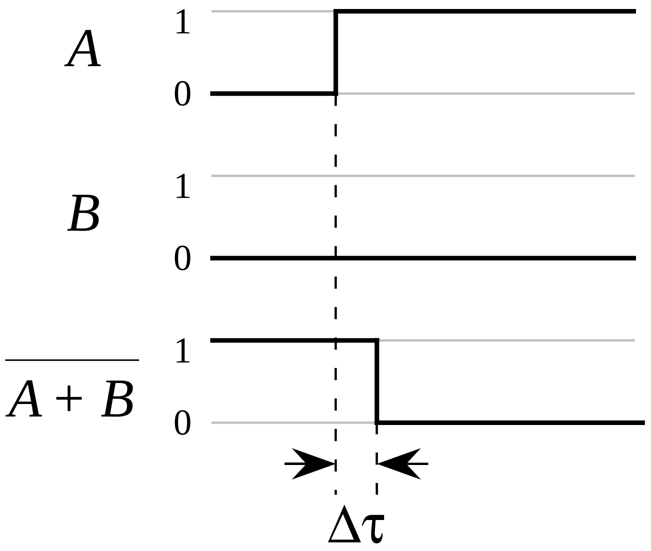
Classical Model of a Finite State Machine

- An FSM is composed of a combinational logic unit and delay elements (called *flip-flops*) in a feedback path, which maintains state information.



NOR Gate with Lumped Delay



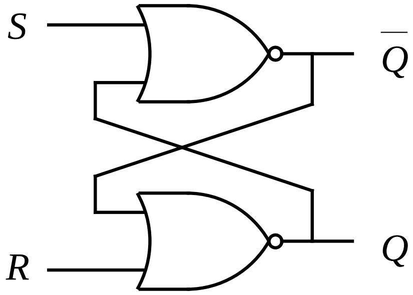


Timing Behavior

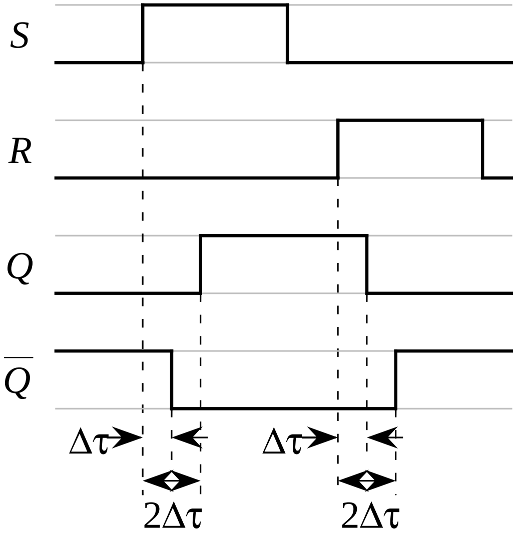
- **The delay between input and output (which is lumped at the output for the purpose of analysis) is at the basis of the functioning of an important memory element, the *flip-flop*.**

S-R Flip-Flop

- **The S-R flip-flop is an active high (positive logic) device.**

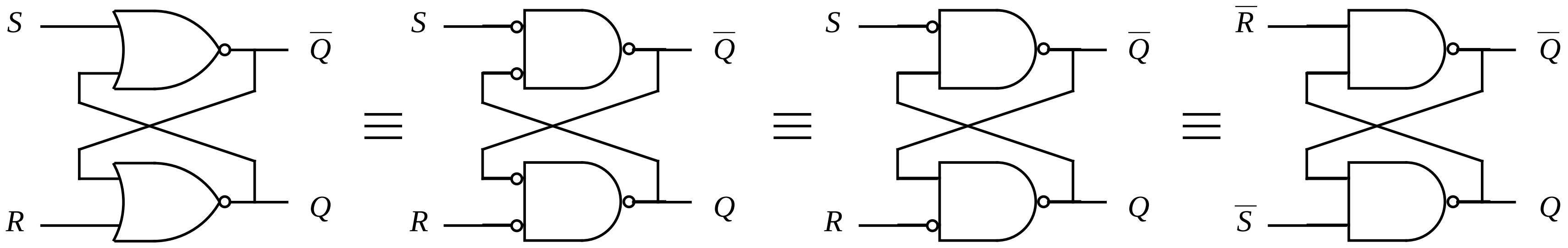


Q_t	S_t	R_t	Q_{i+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	(disallowed)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	(disallowed)

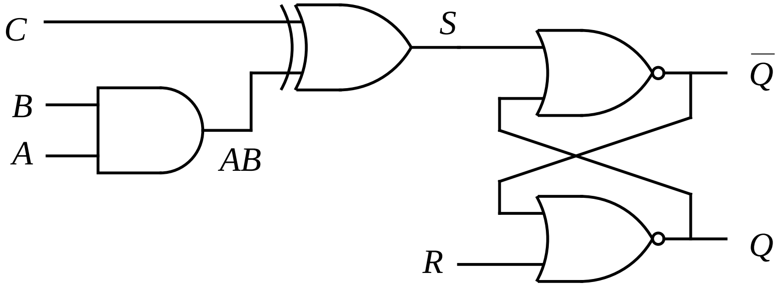


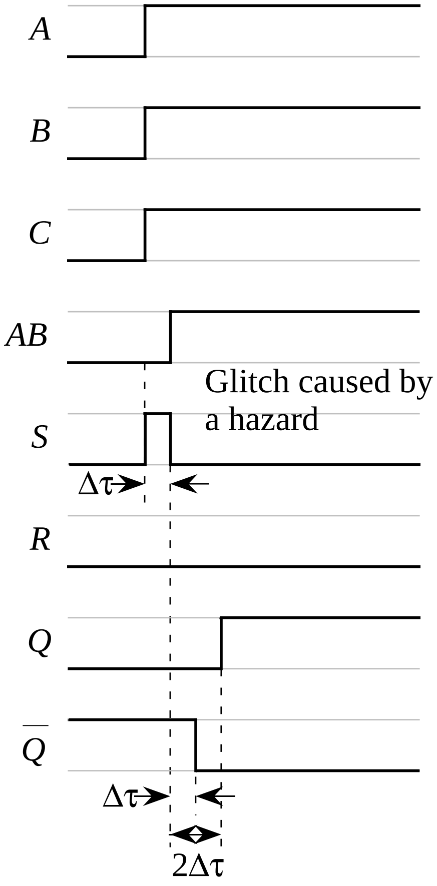
Timing Behavior

NAND Implementation of S-R Flip-Flop



A Hazard

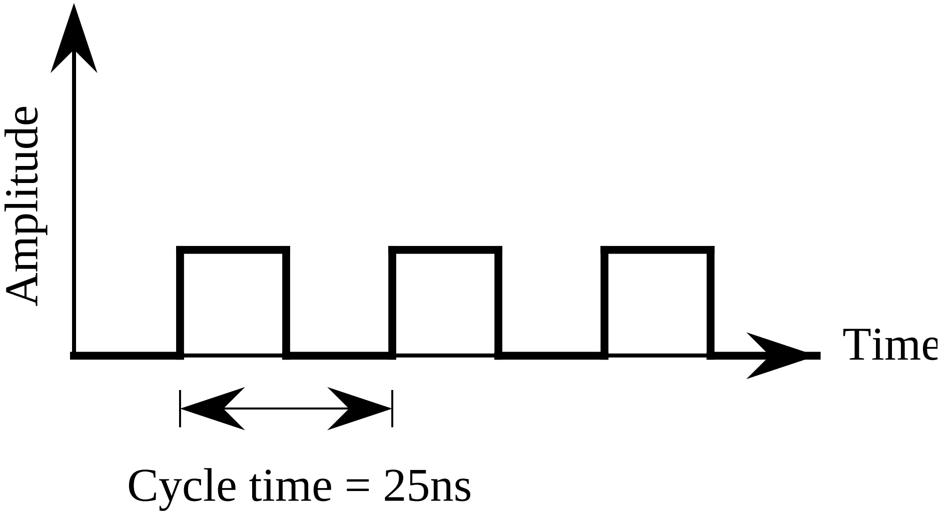




Timing Behavior

- **It is desirable to be able to “turn off” the flip-flop so it does not respond to such hazards.**

A Clock Waveform: The Clock Paces the System



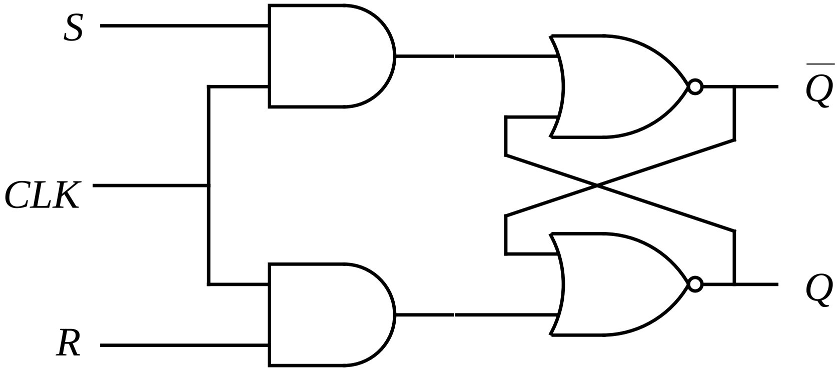
- In a positive logic system, the “action” happens when the clock is high, or positive. The low part of the clock cycle allows propagation between subcircuits, so their inputs settle at the correct value when the clock next goes high.

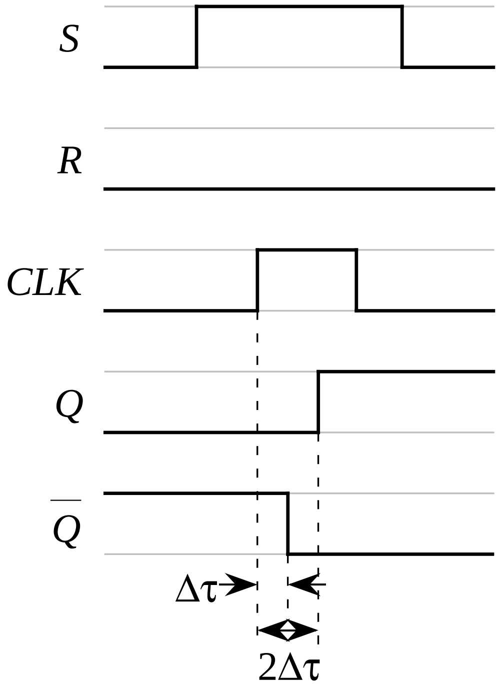
Scientific Prefixes

- **For computer memory, 1K = 2^{10} = 1024. For everything else, like clock speeds, 1K = 1000, and likewise for 1M, 1G, *etc.***

Prefix	Abbrev.	Quantity	Prefix	Abbrev.	Quantity
milli	m	10^{-3}	Kilo	K	10^{3}
micro	µ	10^{-6}	Mega	M	10^{6}
nano	n	10^{-9}	Giga	G	10^{9}
pico	p	10^{-12}	Tera	T	10^{12}
femto	f	10^{-15}	Peta	P	10^{15}
atto	a	10^{-18}	Exa	E	10^{18}

Clocked S-R Flip-Flop



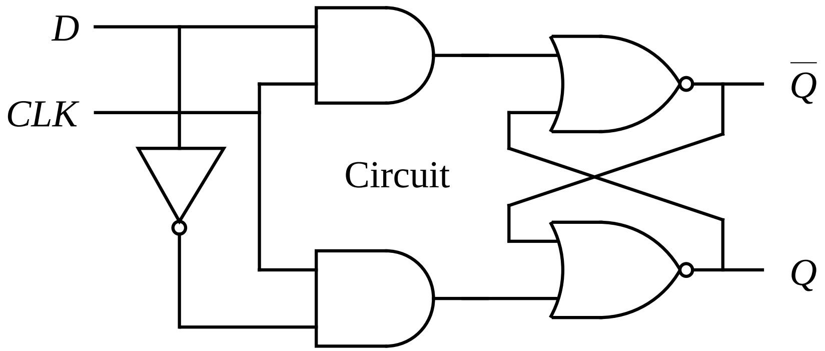


Timing Behavior

- **The clock signal, CLK, enables the S and R inputs to the flip-flop.**

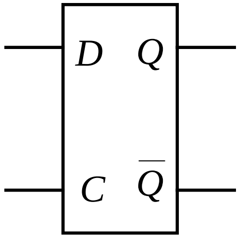
Clocked D Flip-Flop

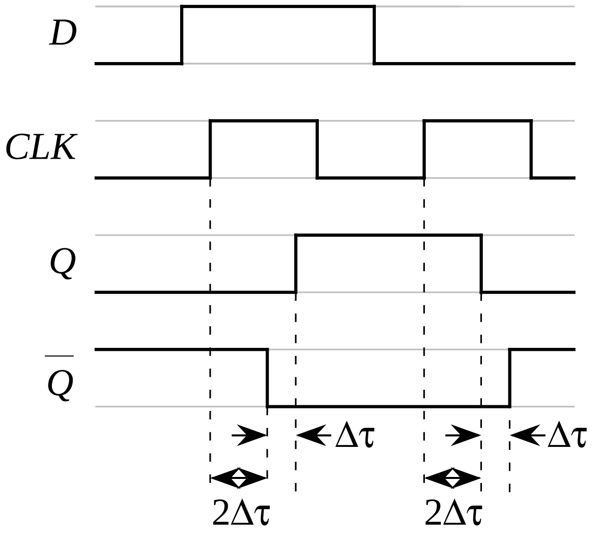
- **The clocked D flip-flop, sometimes called a *latch*, has a potential problem: If D changes while the clock is high, the output will also change. The *Master-Slave* flip-flop (next slide) addresses this problem.**



Circuit

Symbol

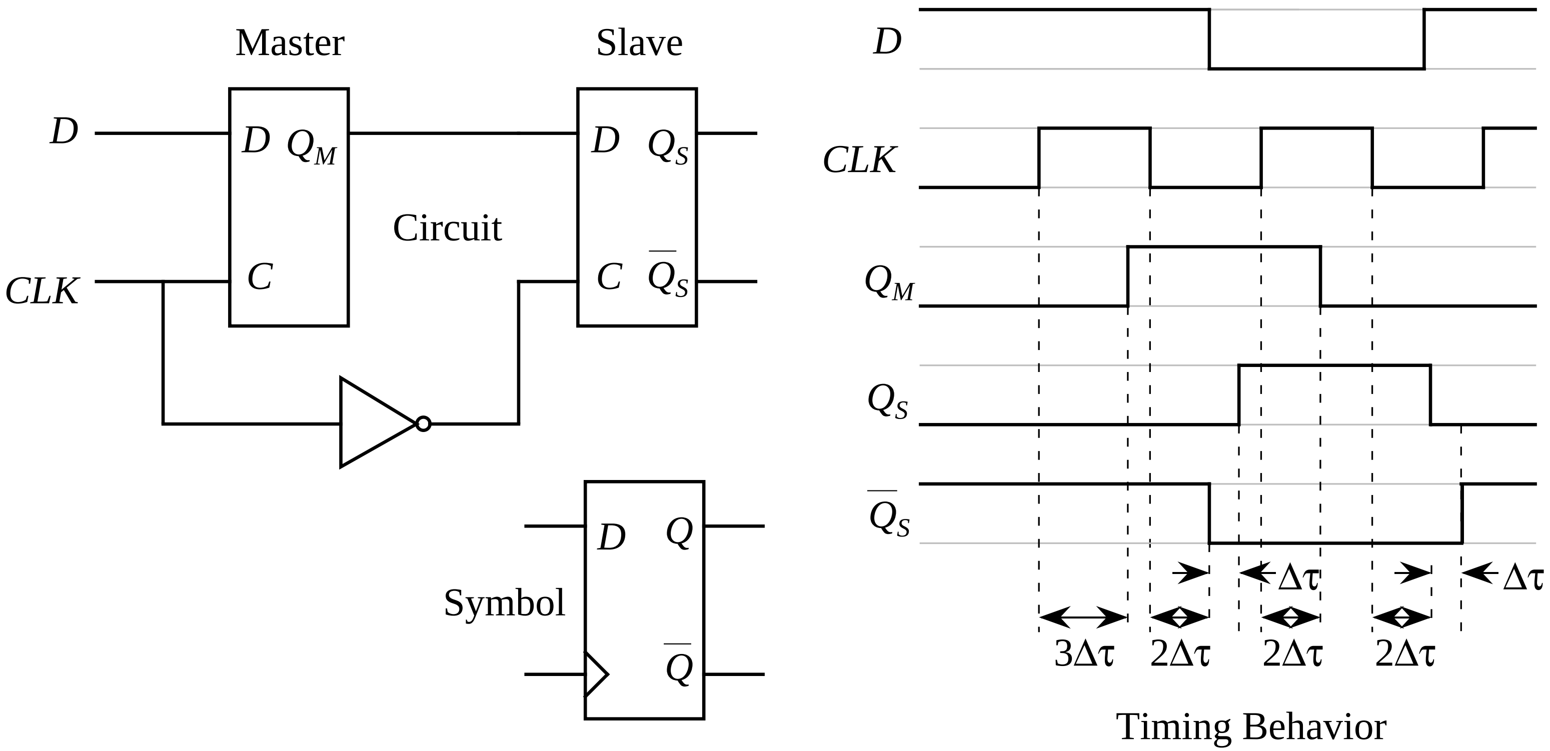




Timing Behavior

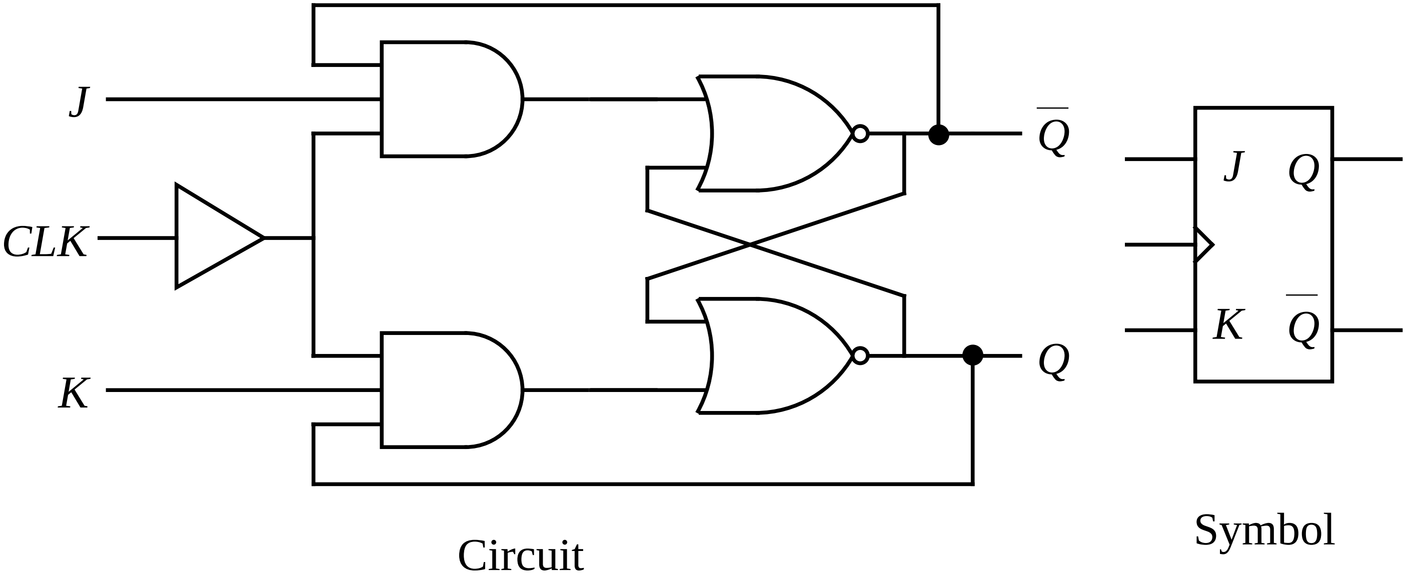
Master-Slave Flip-Flop

- **The rising edge of the clock loads new data into the master, while the slave continues to hold previous data. The falling edge of the clock loads the new master data into the slave.**



Clocked J-K Flip-Flop

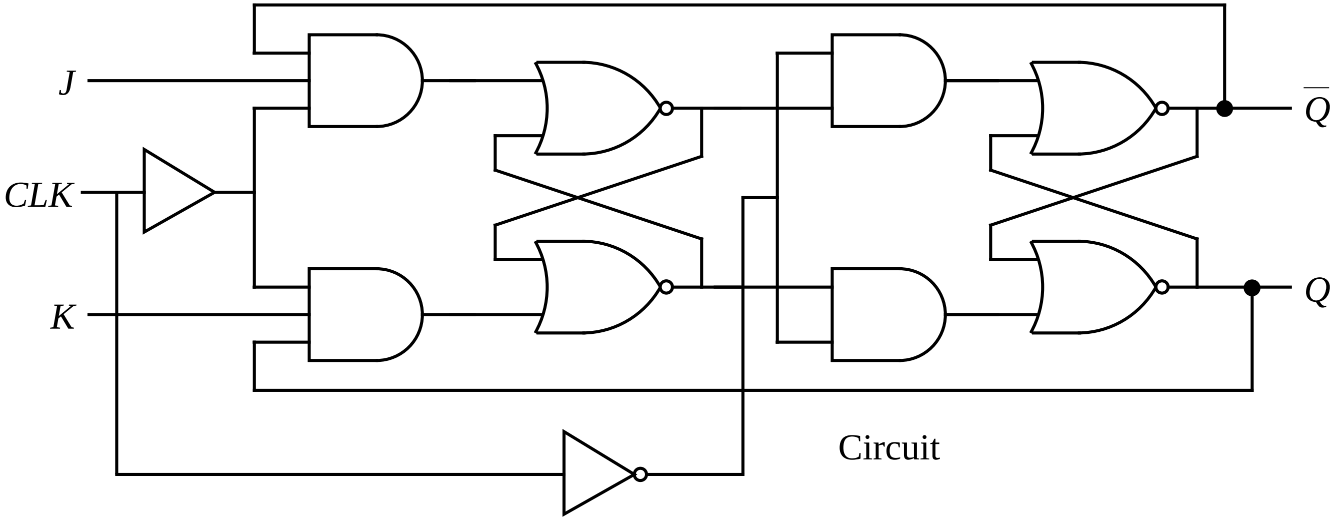
- The J-K flip-flop eliminates the disallowed S=R=1 problem of the S-R flip-flop, because Q enables J while Q' disables K, and vice-versa.
- However, there is still a problem. If J goes momentarily to 1 and then back to 0 while the flip-flop is active and in the reset state, the flip-flop will "catch" the 1. This is referred to as "1's catching."
- The J-K Master-Slave flip-flop (next slide) addresses this problem.



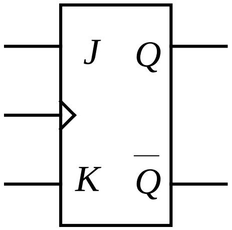
Circuit

Symbol

Master-Slave J-K Flip-Flop



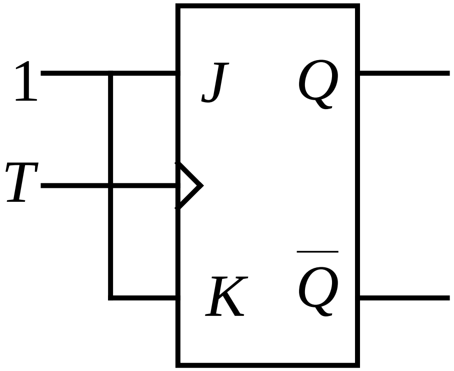
Circuit



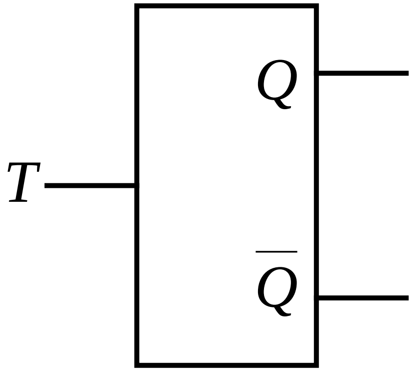
Symbol

Clocked T Flip-Flop

- **The presence of a constant 1 at J and K means that the flip-flop will change its state from 0 to 1 or 1 to 0 each time it is clocked by the T (Toggle) input.**



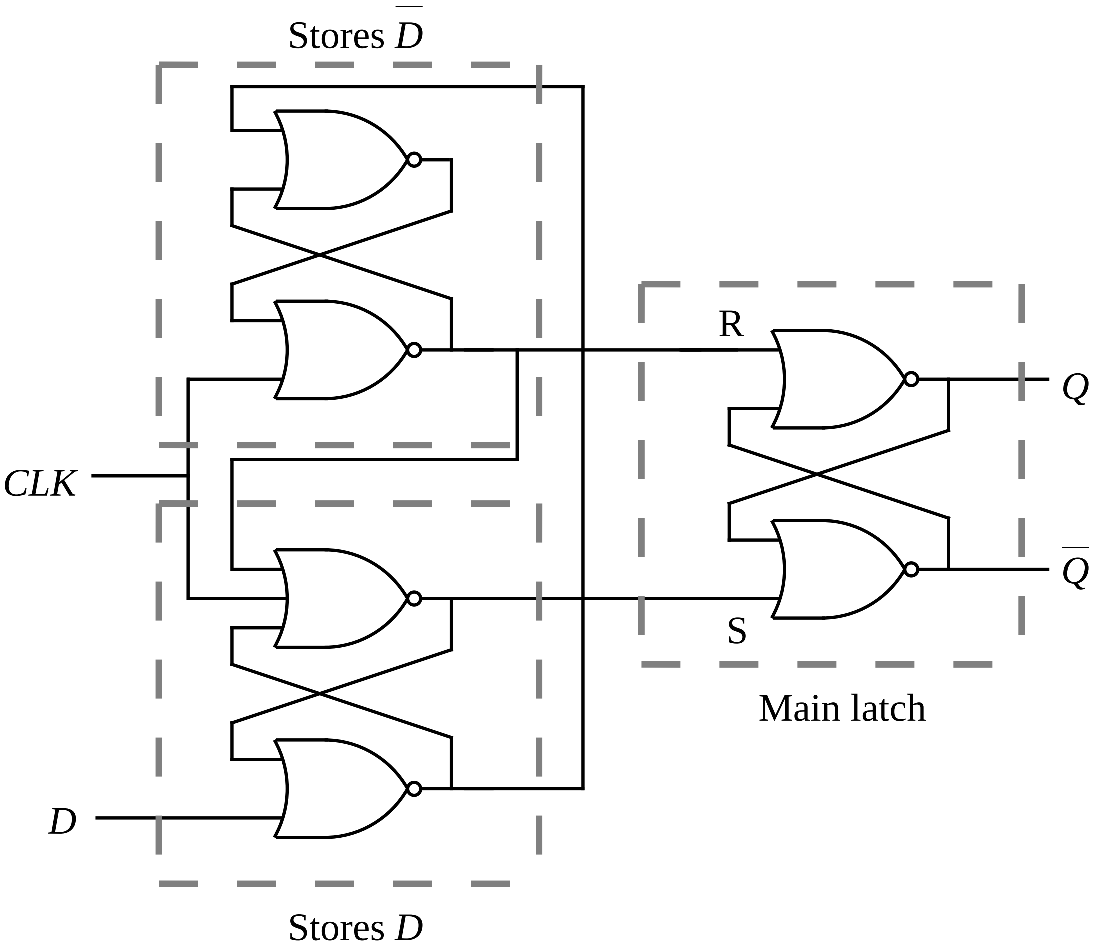
Circuit



Symbol

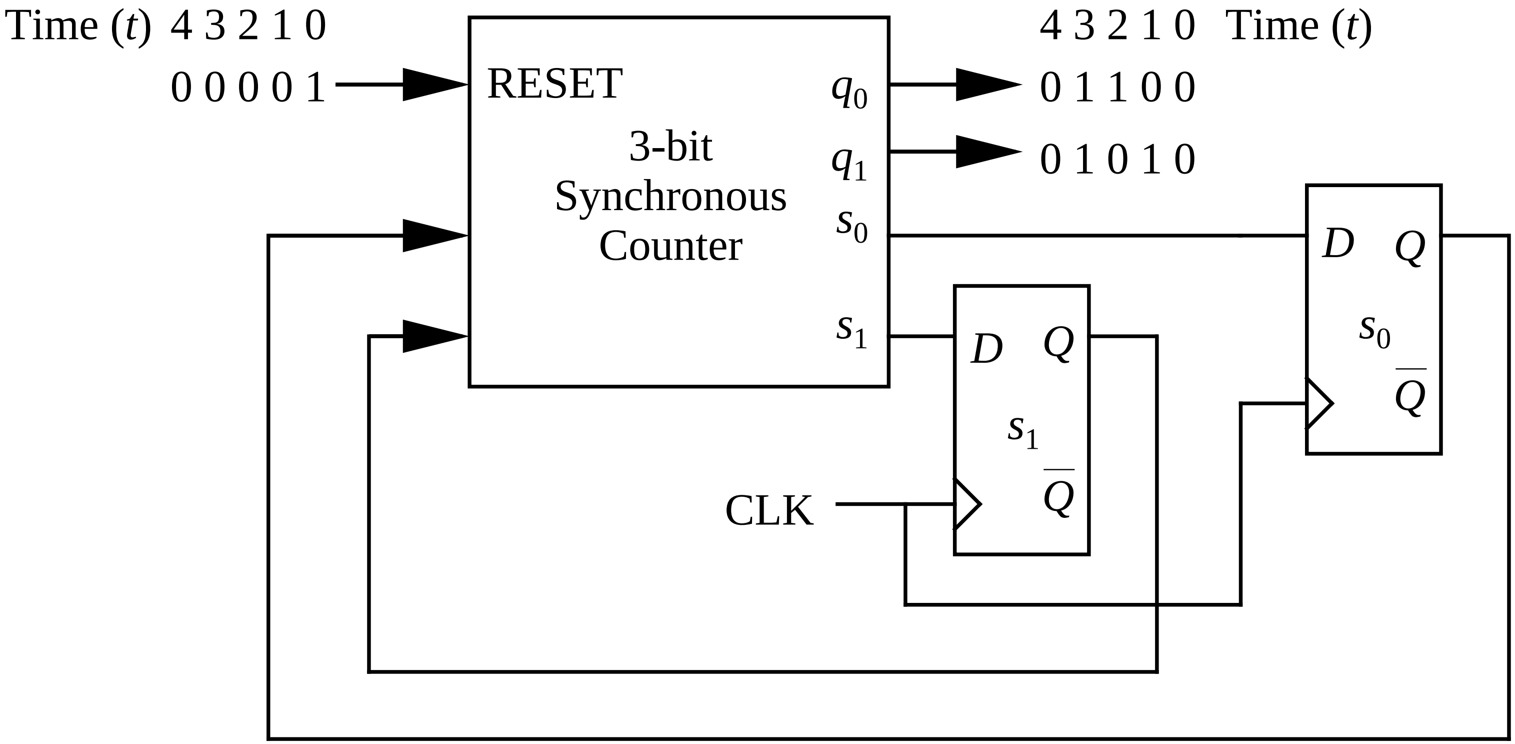
Negative Edge-Triggered D Flip-Flop

- When the clock is high, the two input latches output 0, so the Main latch remains in its previous state, regardless of changes in D.
- When the clock goes high-to-low, values in the two input latches will affect the state of the Main latch.
- While the clock is low, D cannot affect the Main latch.

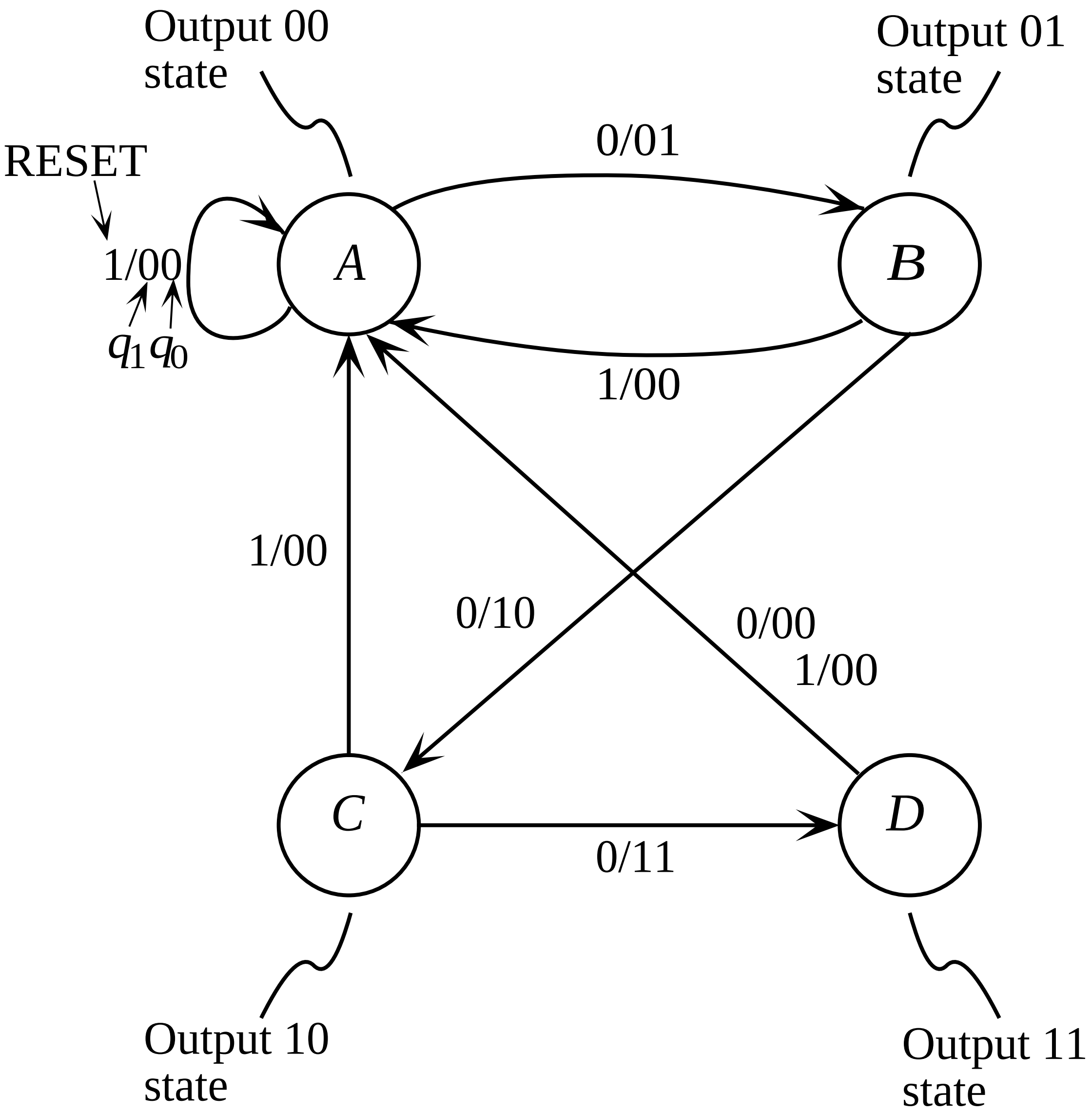


Example: Modulo-4 Counter

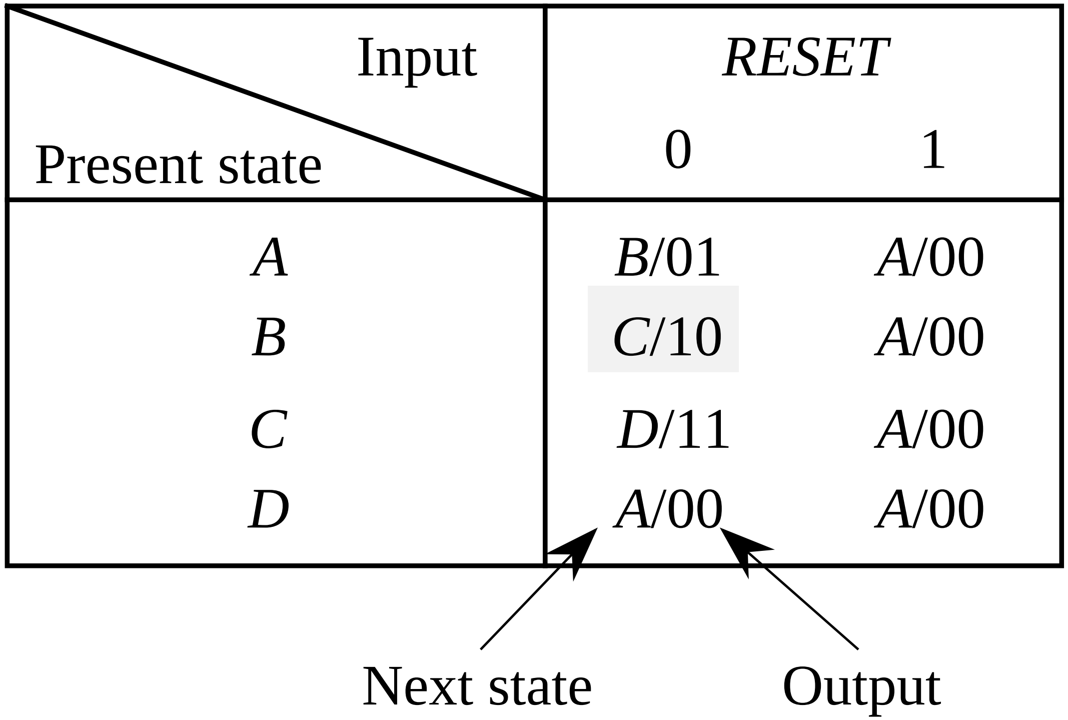
- Counter has a clock input (CLK) and a RESET input.
- Counter has two output lines, which take on values of 00, 01, 10, and 11 on subsequent clock cycles.



State Transition Diagram for Mod-4 Counter



State Table for Mod-4 Counter



Input Present state	*RESET*	
	0	1
A	*B*/01	*A*/00
B	*C*/10	*A*/00
C	*D*/11	*A*/00
D	*A*/00	*A*/00

Next state
Output

State Assignment for Mod-4 Counter

Input / Present state (S_t)	*RESET*	
	0	1
A:00	01/01	00/00
B:01	10/10	00/00
C:10	11/11	00/00
D:11	00/00	00/00

Truth Table for Mod-4 Counter

RESET *r(t)*	$s_1(t)$	$s_0(t)$	$s_1s_0(t+1)$	$q_1q_0(t+1)$
0	0	0	01	01
0	0	1	10	10
0	1	0	11	11
0	1	1	00	00
1	0	0	00	00
1	0	1	00	00
1	1	0	00	00
1	1	1	00	00

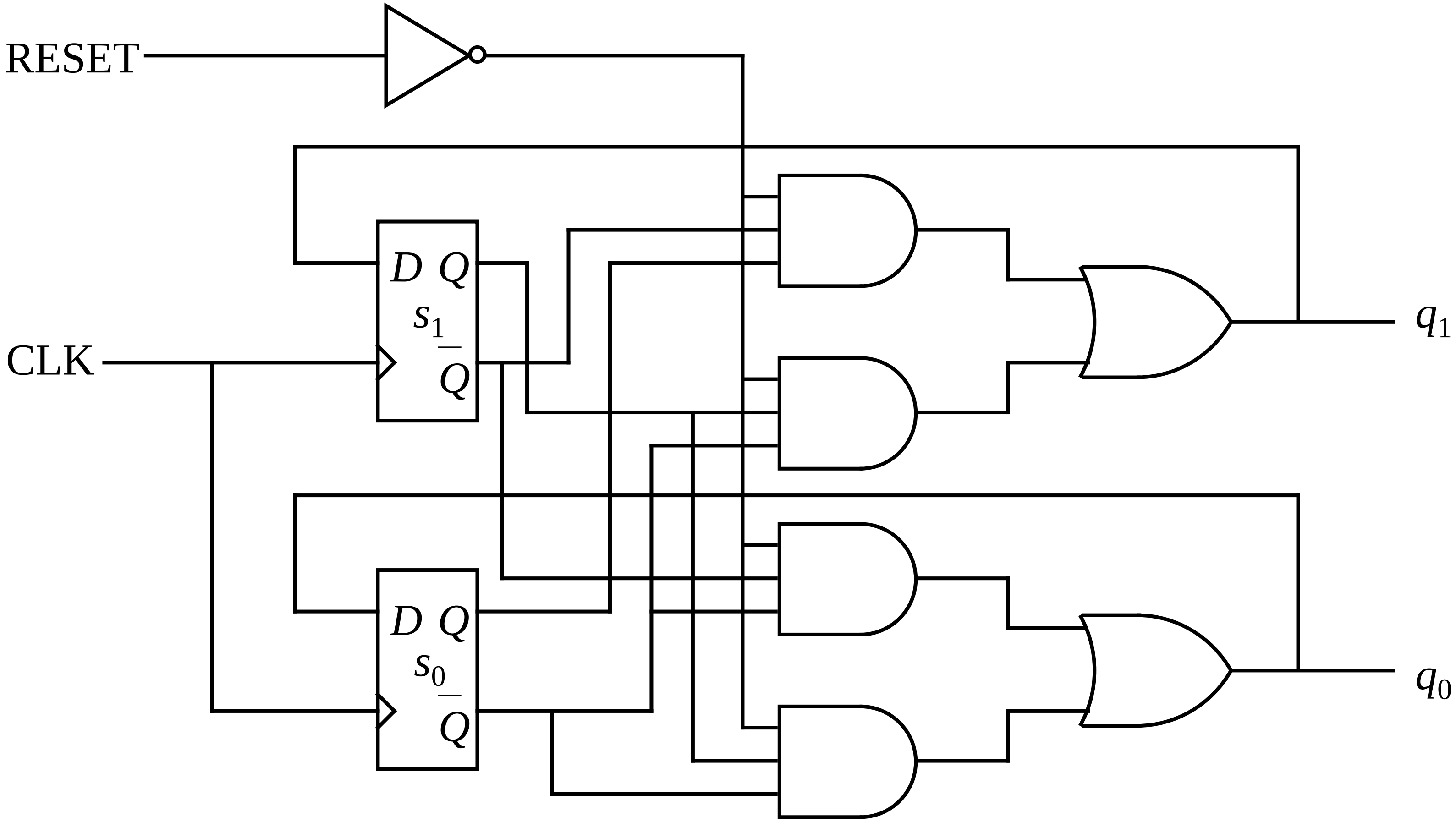
$$s_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

$$s_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}s_0(t) + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

$$q_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

$$q_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}s_0(t) + \overline{r(t)}s_1(t)\overline{s_0(t)}$$

Logic Design for Mod-4 Counter

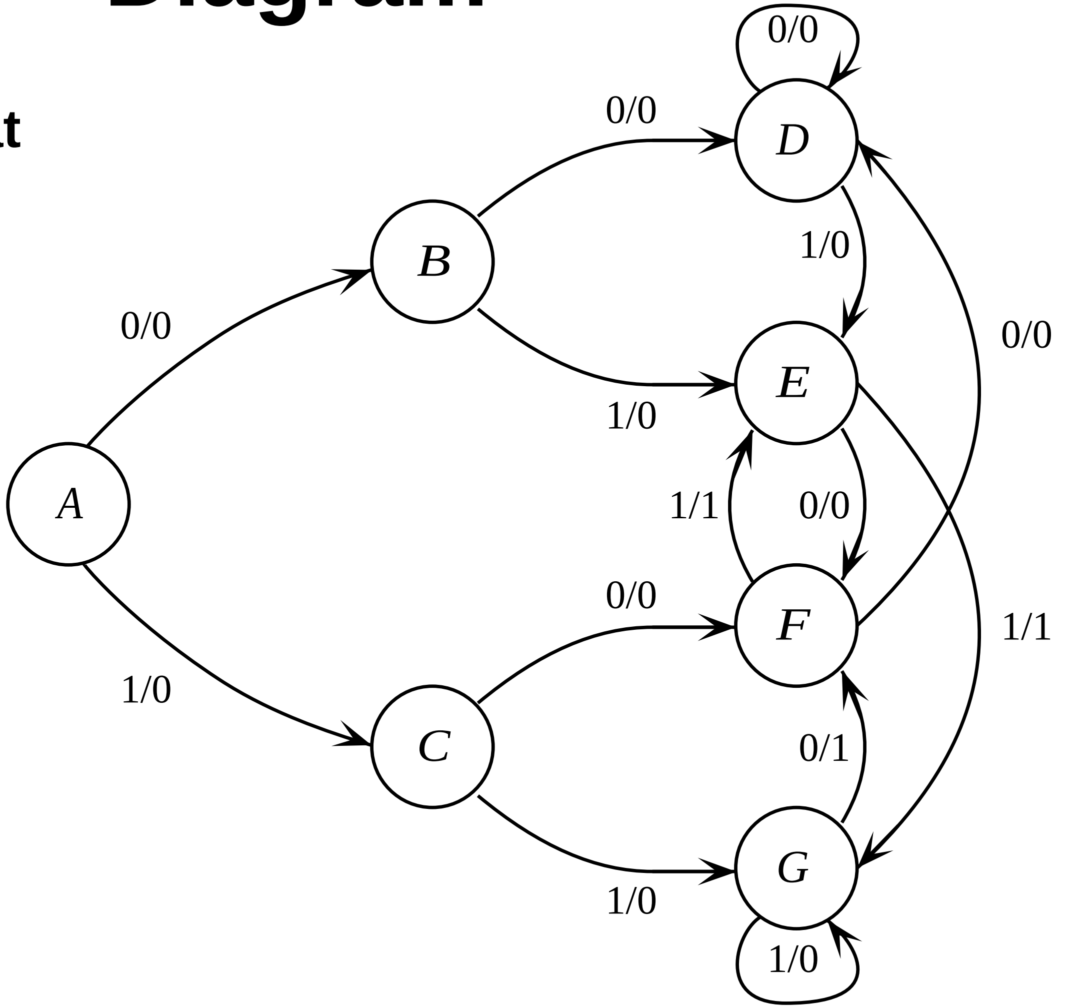


Example: A Sequence Detector

- Example: Design a machine that outputs a 1 when exactly two of the last three inputs are 1.
- *e.g.* input sequence of 011011100 produces an output sequence of 001111010.
- Assume input is a 1-bit serial line.
- Use D flip-flops and 8-to-1 Multiplexers.
- Start by constructing a state transition diagram (next slide).

Sequence Detector State Transition Diagram

- Design a machine that outputs a 1 when exactly two of the last three inputs are 1.



Sequence Detector State Table

Input	*X*	
Present state	0	1
A	*B*/0	*C*/0
B	*D*/0	*E*/0
C	*F*/0	*G*/0
D	*D*/0	*E*/0
E	*F*/0	*G*/1
F	*D*/0	*E*/1
G	*F*/1	*G*/0

Sequence Detector State Assignment

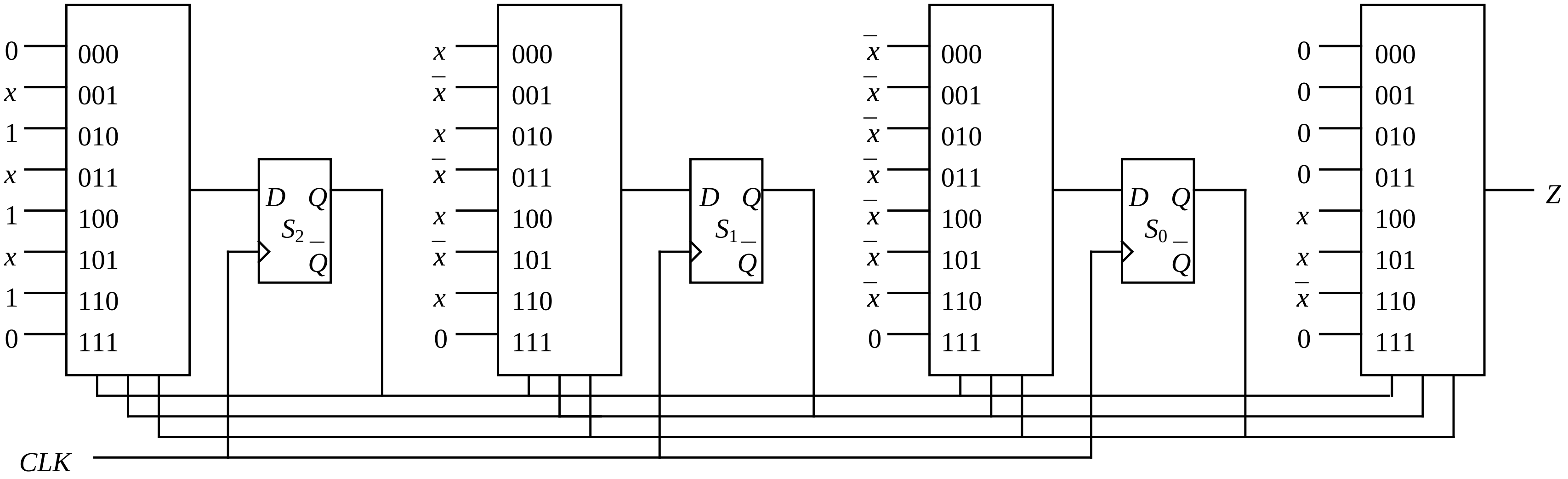
Present state \ Input	X = 0	X = 1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A: 000	001/0	010/0
B: 001	011/0	100/0
C: 010	101/0	110/0
D: 011	011/0	100/0
E: 100	101/0	110/1
F: 101	011/0	100/1
G: 110	101/1	110/0

(a)

Input and state at time *t*				Next state and output at time *t*+1			
s_2	s_1	s_0	x	s_2	s_1	s_0	z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	0
1	0	0	0	1	0	1	0
1	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0
1	0	1	1	1	0	0	1
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d

(b)

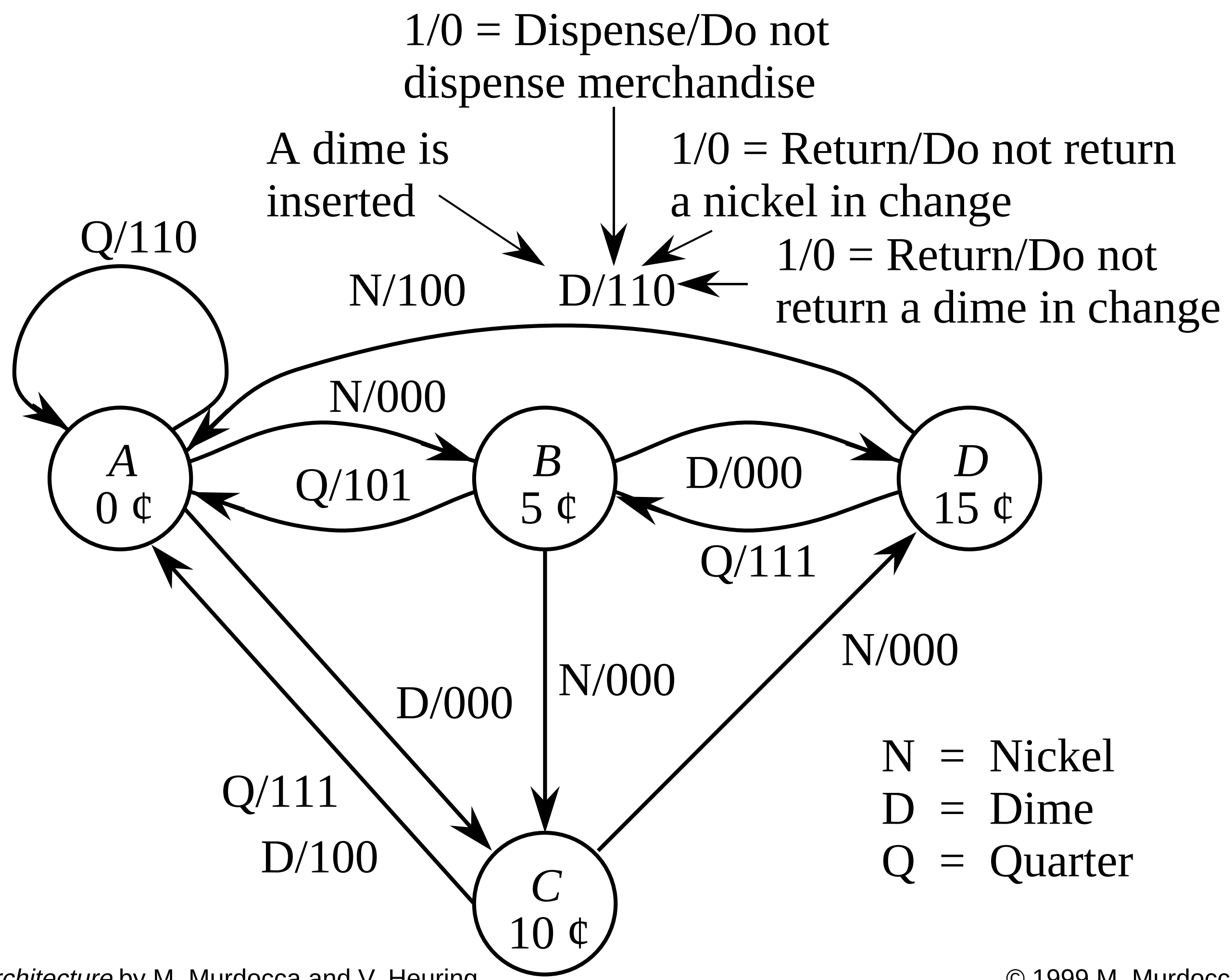
Sequence Detector Logic Diagram



Example: A Vending Machine Controller

- Example: Design a finite state machine for a vending machine controller that accepts nickels (5 cents each), dimes (10 cents each), and quarters (25 cents each). When the value of the money inserted equals or exceeds twenty cents, the machine vends the item and returns change if any, and waits for next transaction.
- Implement with PLA and D flip-flops.

Vending Machine State Transition Diagram


1/0 = Dispense/Do not dispense merchandise
A dime is inserted
1/0 = Return/Do not return a nickel in change
1/0 = Return/Do not return a dime in change
D/110
Q/110
N/100
N/000
Q/101
D/000
Q/111
N/000
D/000
N/000
Q/111
D/100
A
0 ¢
B
5 ¢
C
10 ¢
D
15 ¢
N = Nickel
D = Dime
Q = Quarter

Vending Machine State Table and State Assignment

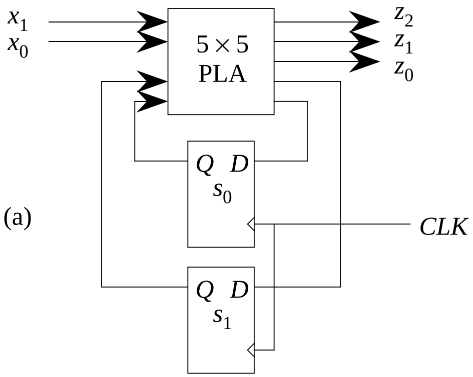
Input P.S.	N 00	D 01	Q 10
A	*B*/000	*C*/000	*A*/110
B	*C*/000	*D*/000	*A*/101
C	*D*/000	*A*/100	*A*/111
D	*A*/100	*A*/110	*B*/111

(a)

Input P.S.	N x_1x_0 00	D x_1x_0 01	Q x_1x_0 10
s_1s_0	s_1s_0 / $z_2z_1z_0$		
A:00	01/000	10/000	00/110
B:01	10/000	11/000	00/101
C:10	11/000	00/100	00/111
D:11	00/100	00/110	01/111

(b)

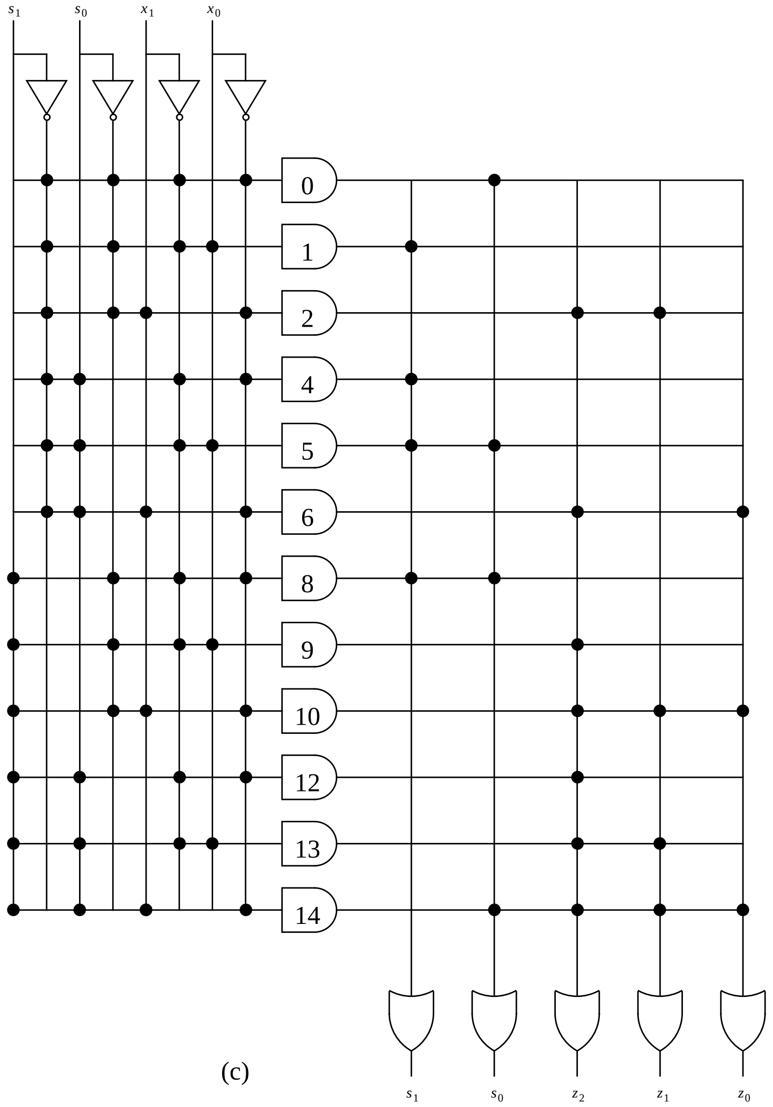
PLA Vending Machine Controller



(a)

Base 10 equivalent	Present state		Coin		Next state		Dispense	Return nickel	Return dime
	s_1	s_0	x_1	x_0	s_1	s_0	z_2	z_1	z_0
0	0	0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0	0	0
2	0	0	1	0	0	0	1	1	0
3	0	0	1	1	d	d	d	d	d
4	0	1	0	0	1	0	0	0	0
5	0	1	0	1	1	1	0	0	0
6	0	1	1	0	0	0	1	0	1
7	0	1	1	1	d	d	d	d	d
8	1	0	0	0	1	1	0	0	0
9	1	0	0	1	0	0	1	0	0
10	1	0	1	0	0	0	1	1	1
11	1	0	1	1	d	d	d	d	d
12	1	1	0	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1	0
14	1	1	1	0	0	1	1	1	1
15	1	1	1	1	d	d	d	d	d

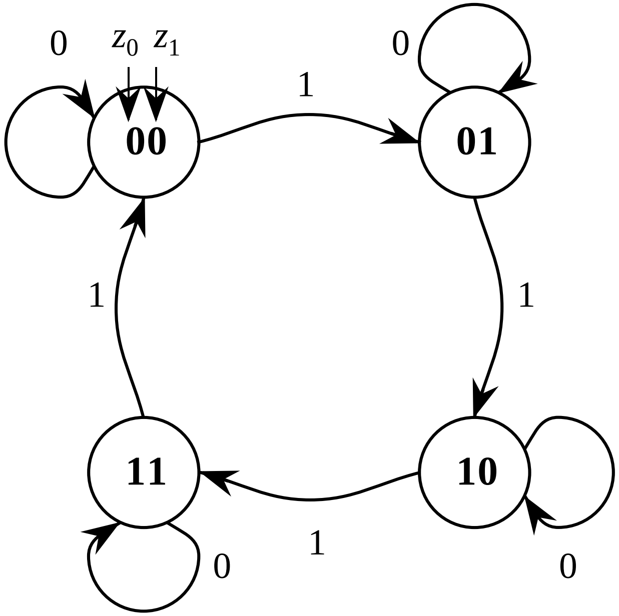
(b)

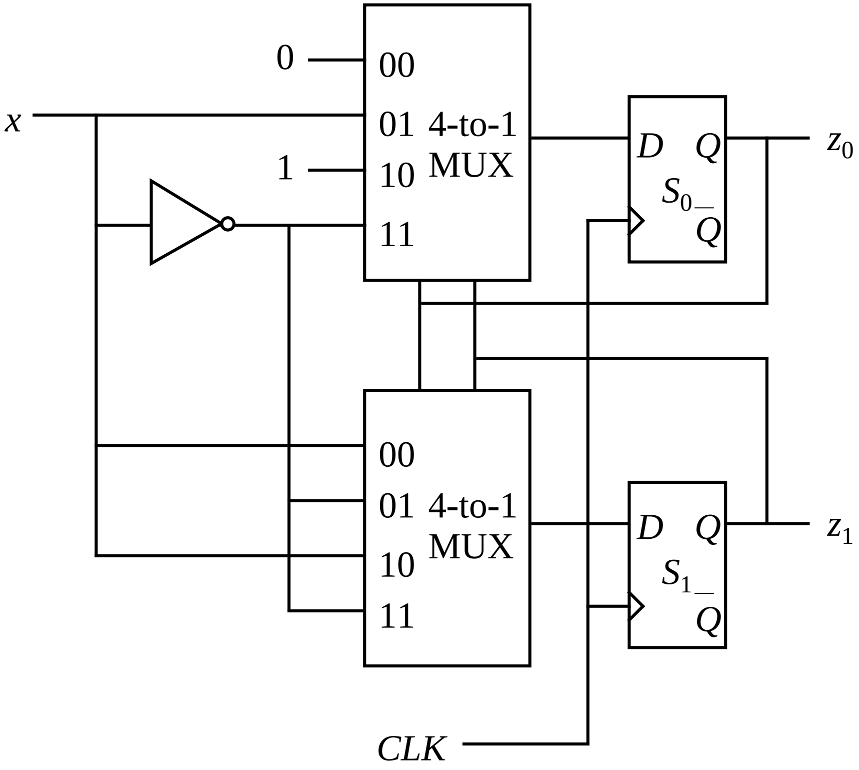


(c)

Moore Counter

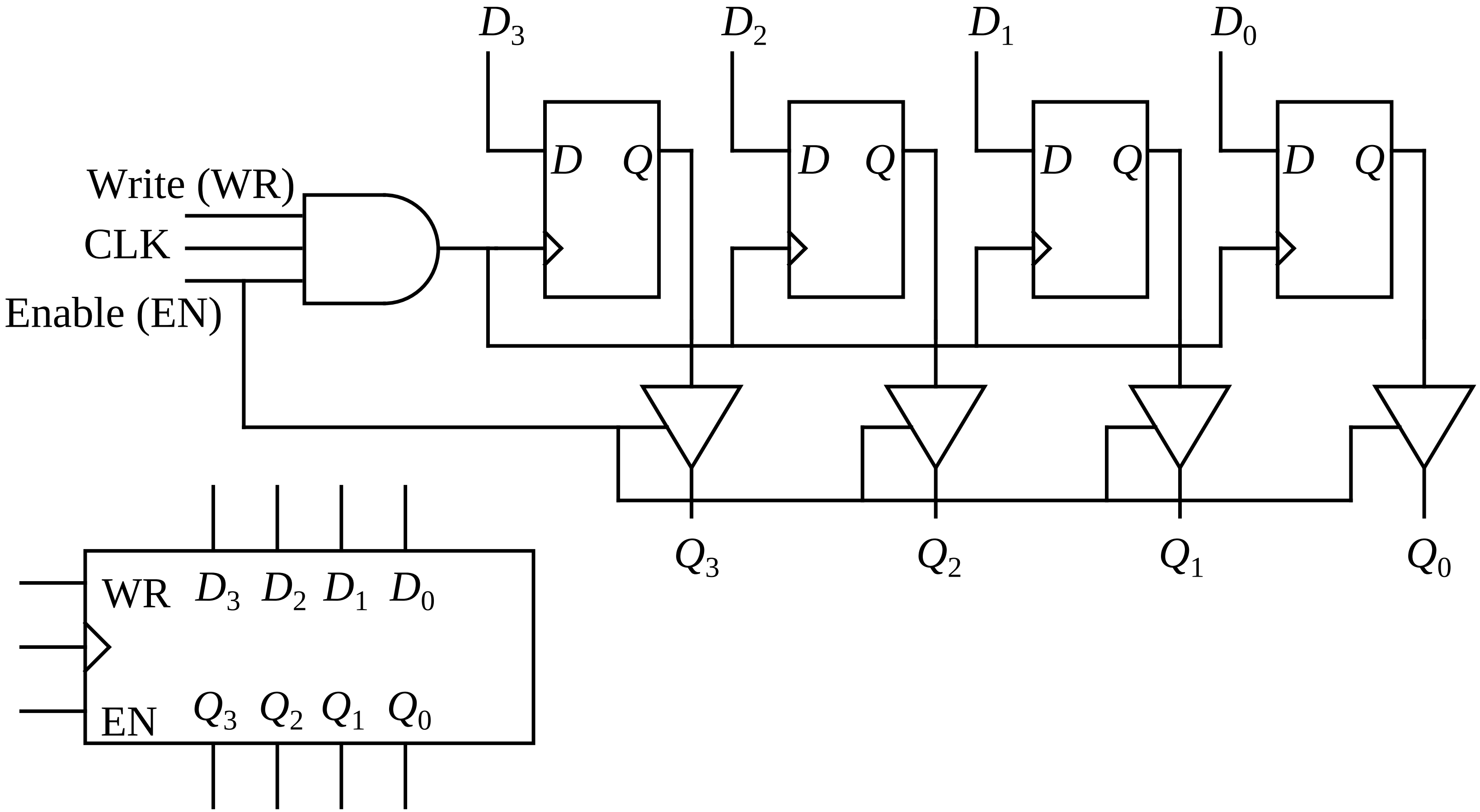
- Mealy Model: Outputs are functions of Inputs and Present State.
- Previous FSM designs were Mealy Machines, in which next state was computed from present state and inputs.
- Moore Model: Outputs are functions of Present State only.



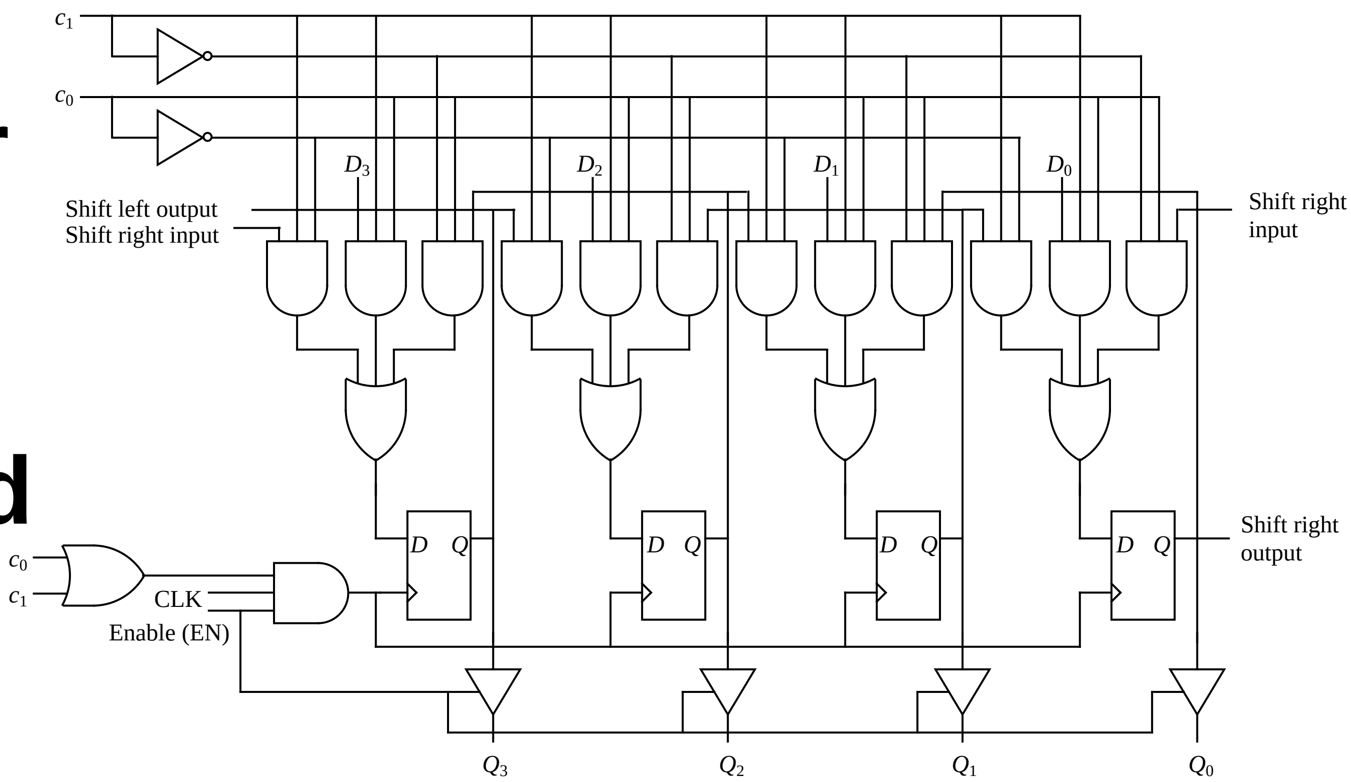


Four-Bit Register

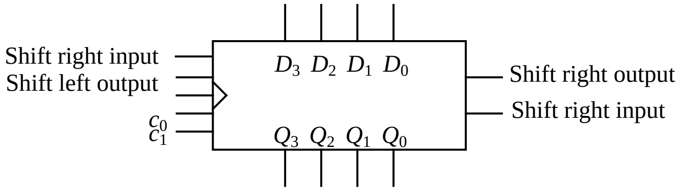
- Makes use of tri-state buffers so that multiple registers can gang their outputs to common output lines.



Left-Right Shift Register with Parallel Read and Write

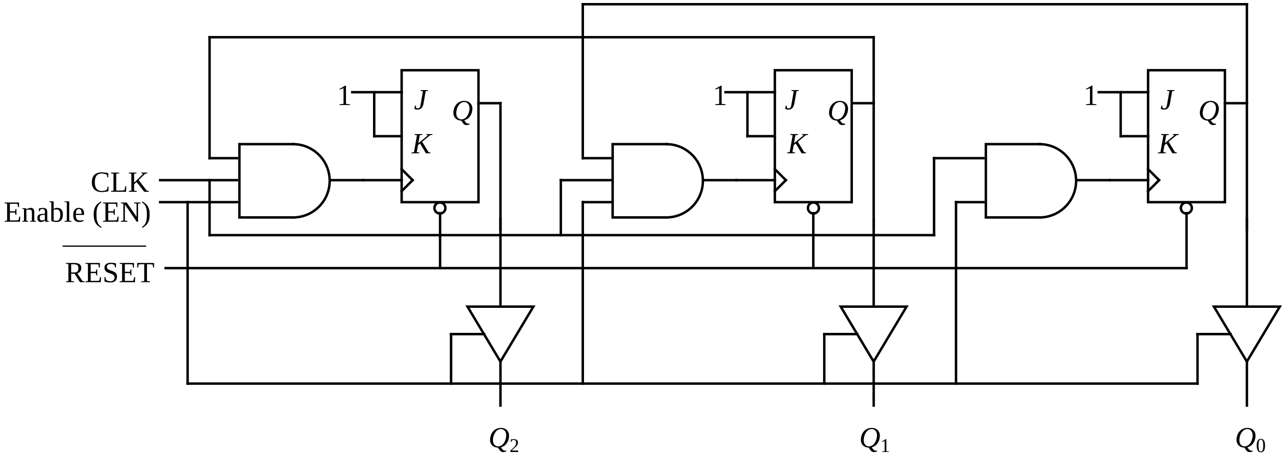


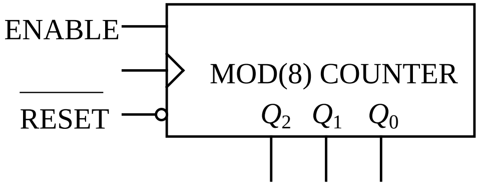
Control c_1 c_0		Function
0	0	No change
0	1	Shift left
1	0	Shift right
1	1	Parallel load

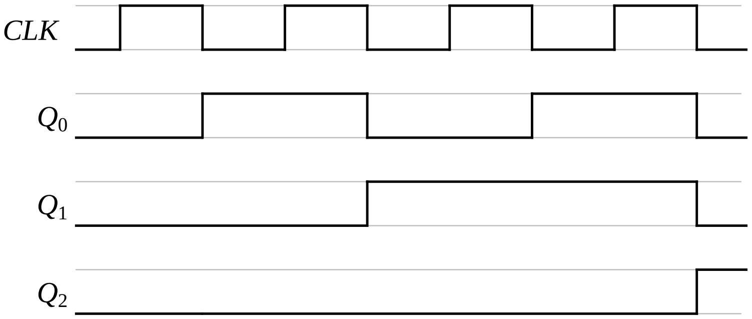


Modulo-8 Counter

- Note the use of the T flip-flops, implemented as J-K’s. They are used to toggle the input of the next flip-flop when its output is 1.







Timing Behavior