DIGITAL LOGIC CIRCUITS

Logic Gates

Boolean Algebra

Map Specification

Combinational Circuits

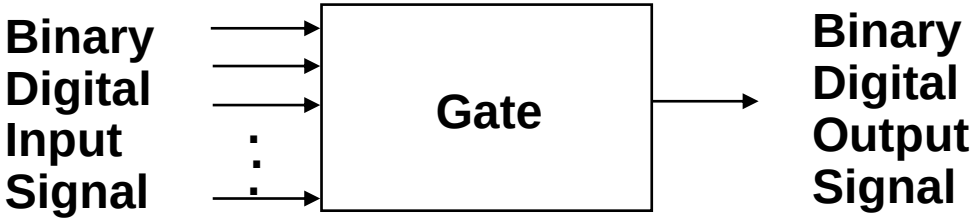
Flip-Flops

Sequential Circuits

Memory Components

Integrated Circuits

BASIC LOGIC BLOCK - GATE -



Types of Basic Logic Blocks

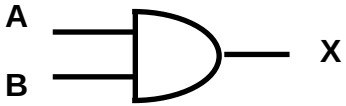
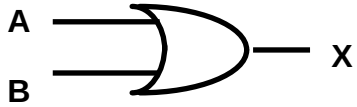
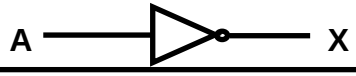
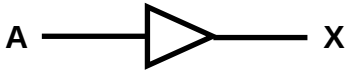
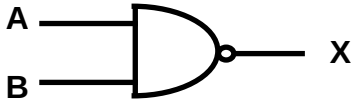
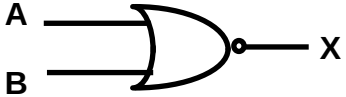
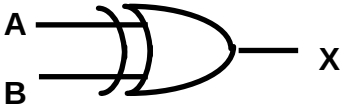
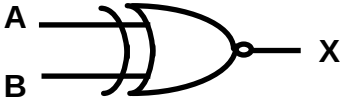
- **Combinational Logic Block**
  Logic Blocks whose output logic value depends only on the input logic values

- **Sequential Logic Block**
  Logic Blocks whose output logic value depends on the input values and the state (stored information) of the blocks

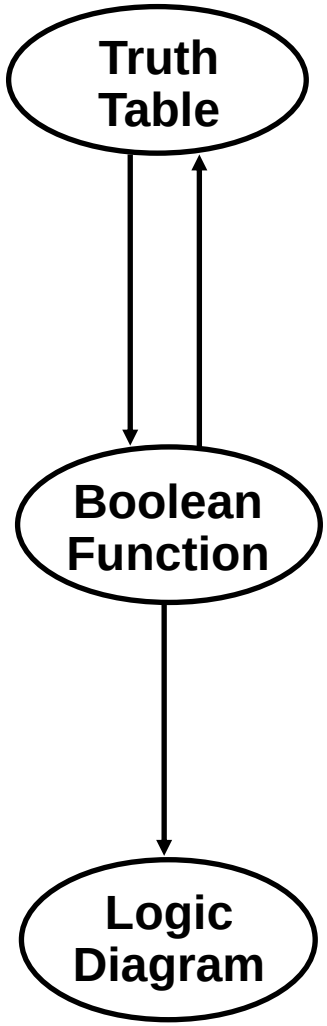
Functions of Gates can be described by

- Truth Table
- Boolean Function
- Karnaugh Map

COMBINATIONAL GATES

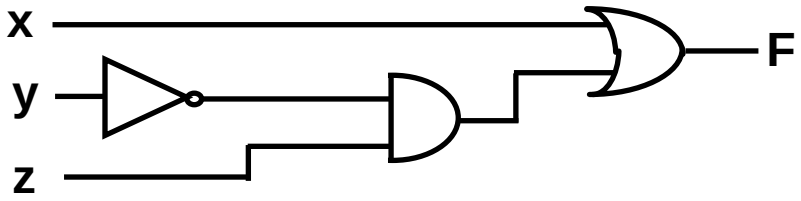
Name	Symbol	Function	Truth Table
AND		X = A • B or X = AB	A B X 0 0 0 0 1 0 1 0 0 1 1 1
OR		X = A + B	A B X 0 0 0 0 1 1 1 0 1 1 1 1
I		X = A	A X 0 1 1 0
Buffer		X = A	A X 0 0 1 1
NAND		X = (AB)’	A B X 0 0 1 0 1 1 1 0 1 1 1 0
NOR		X = (A + B)’	A B X 0 0 1 0 1 0 1 0 0 1 1 0
XOR Exclusive OR		X = A ⊕ B or X = A’B + AB’	A B X 0 0 0 0 1 1 1 0 1 1 1 0
XNOR Exclusive NOR or Equivalence		X = (A ⊕ B)’ or X = A’B’+ AB	A B X 0 0 1 0 1 0 1 0 0 1 1 1

LOGIC CIRCUIT DESIGN



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$F = x + y'z$



BASIC IDENTITIES OF BOOLEAN ALGEBRA

[1] $x + 0 = x$	[2] $x \cdot 0 = 0$
[3] $x + 1 = 1$	[4] $x \cdot 1 = x$
[5] $x + x = x$	[6] $x \cdot x = x$
[7] $x + x' = 1$	[8] $x \cdot X' = 0$
[9] $x + y = y + x$	[10] $xy = yx$
[11] $x + (y + z) = (x + y) + z$	[12] $x(yz) = (xy)z$
[13] $x(y + z) = xy + xz$	[14] $x + yz = (x + y)(x + z)$
[15] $(x + y)' = x'y'$	[16] $(xy)' = x' + y'$
[17] $(x')' = x$	

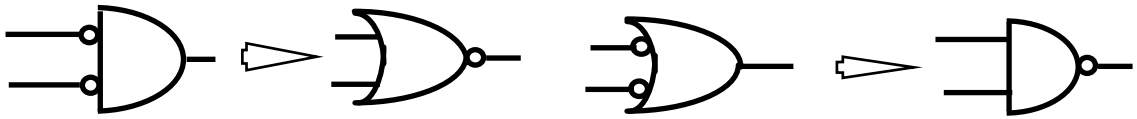
[15] and [16] : De Morgan's Theorem

Usefulness of this Table

- Simplification of the Boolean function
- Derivation of equivalent Boolean functions to obtain logic diagrams utilizing different logic gates
  - -- Ordinarily ANDs, ORs, and Inverters
  - -- But a certain different form of Boolean function may be convenient to obtain circuits with NANDs or NORs
    → Applications of De Morgans Theorem

$x'y' = (x + y)'$ $\qquad$ $x' + y' = (xy)'$

I, AND → NOR $\qquad$ I, OR → NAND

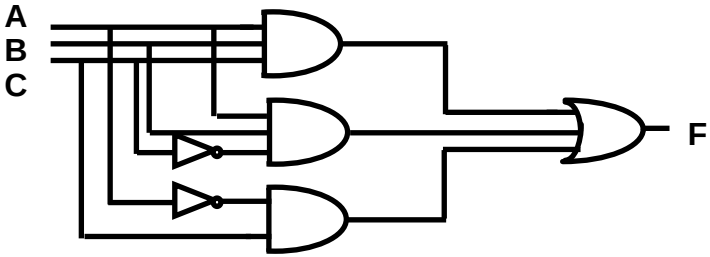


EQUIVALENT CIRCUITS

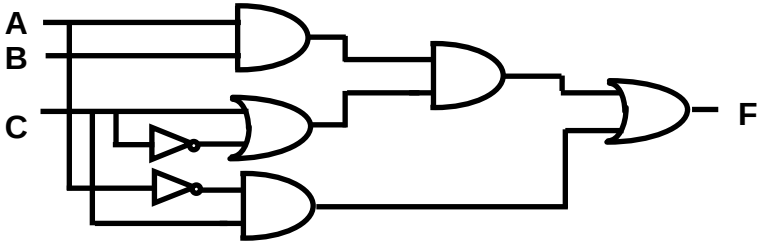
Many different logic diagrams are possible for a given Function

F = ABC + ABC' + A'C ..... (1)
= AB(C + C') + A'C [13] ..... (2)
= AB • 1 + A'C [7]
= AB + A'C [4] ..... (3)

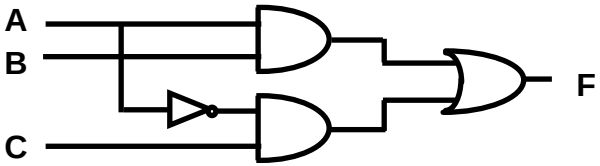
(1)



(2)



(3)



COMPLEMENT OF FUNCTIONS

A Boolean function of a digital logic circuit is represented by only using logical variables and AND, OR, and Invert operators.

→ Complement of a Boolean function

- Replace all the variables and subexpressions in the parentheses appearing in the function expression with their respective complements

$$A,B,\ldots,Z,a,b,\ldots,z \Rightarrow A',B',\ldots,Z',a',b',\ldots,z'$$
$$(p + q) \Rightarrow (p + q)'$$

- Replace all the operators with their respective complementary operators

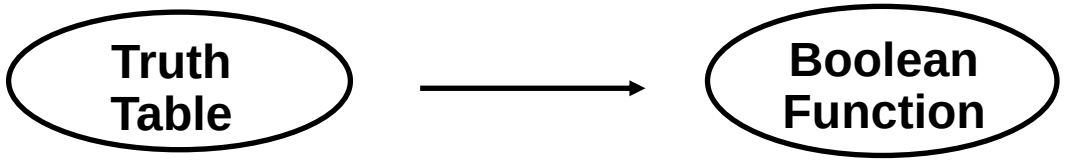
AND ⇒ OR
OR ⇒ AND

- Basically, extensive applications of the De Morgan's theorem

$$(x_1 + x_2 + \ldots + x_n)' \Rightarrow x_1'x_2'\ldots x_n'$$

$$(x_1x_2 \ldots x_n)' \Rightarrow x_1' + x_2' + \ldots + x_n'$$

SIMPLIFICATION



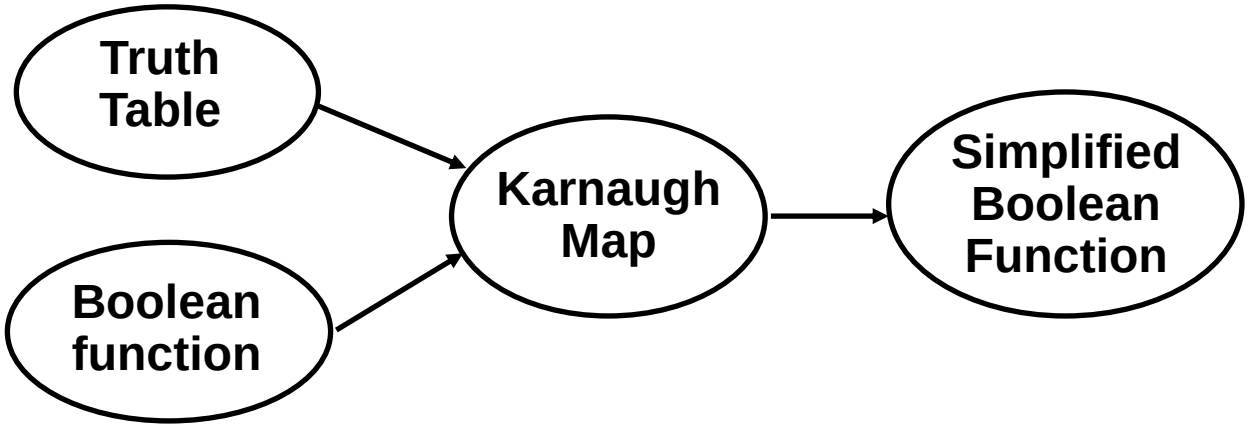
Unique

Many different expressions exist

Simplification from Boolean function

- Finding an equivalent expression that is least expensive to implement
- For a simple function, it is possible to obtain a simple expression for low cost implementation
- But, with complex functions, it is a very difficult task

Karnaugh Map (K-map) is a simple procedure for simplifying Boolean expressions.



KARNAUGH MAP

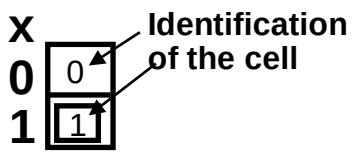
Karnaugh Map for an n-input digital logic circuit (n-variable sum-of-products form of Boolean Function, or Truth Table) is

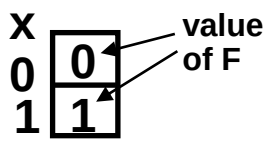
- **Rectangle divided into 2^n cells**
- **Each cell is associated with a *Minterm***
- **An output(function) value for each input value associated with a mintern is written in the cell representing the minterm**
  → 1-cell, 0-cell

Each Minterm is identified by a decimal number whose binary representation is identical to the binary interpretation of the input values of the minterm.

Karnaugh Map

x	F
0	1
1	0

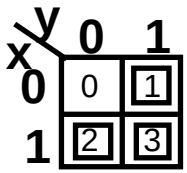


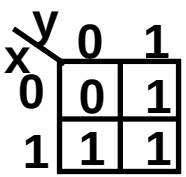


F(x) =Σ (1)

1-cell

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1





F(x,y) = Σ (1,2,3)

KARNAUGH MAP

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6

y, z

x \ yz	00	01	11	10
0	0	1	0	1
1	1	0	0	0

$F(x,y,z) = \sum (1,2,4)$

u	v	w	x	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

uv \ wx	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

w, v, u, x

uv \ wx	00	01	11	10
00	0	1	1	0
01	0	0	0	1
11	0	0	0	1
10	1	1	1	0

$F(u,v,w,x) = \sum (1,3,6,8,9,11,14)$

MAP SIMPLIFICATION - 2 ADJACENT CELLS -

Rule: xy' +xy = x(y+y') = x

Adjacent cells

- **binary identifications are different in one bit**
  → **minterms associated with the adjacent cells have one variable complemented each other**

Cells (1,0) and (1,1) are adjacent
Minterms for (1,0) and (1,1) are
x • y' --> x=1, y=0
x • y --> x=1, y=1

F = xy'+ xy can be reduced to F = x
From the map

x \ y	0	1
0	0	0
1	1	1

2 adjacent cells xy' and xy
→ **merge them to a larger cell x**

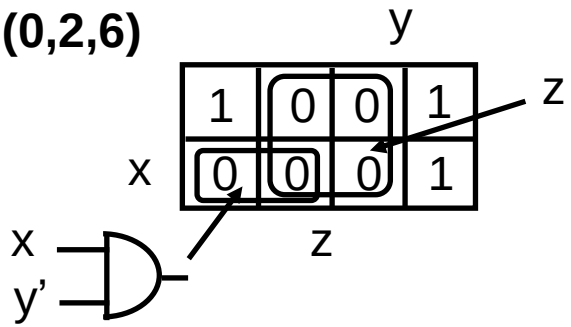
$F(x,y) = \Sigma(2,3)$
= xy'+ xy
= x

IMPLEMENTATION OF K-MAPS - Product-of-Sums Form -

Logic function represented by a Karnaugh map can be implemented in the form of I-OR-AND

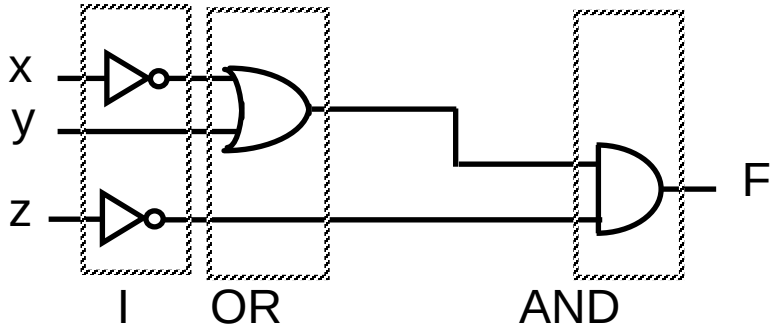
If we implement a Karnaugh map using 0-cells, the complement of F, i.e., F', can be obtained. Thus, by complementing F' using DeMorgan's theorem F can be obtained

F(x,y,z) = (0,2,6)



F' = xy' + z

F = (xy')z'
= (x' + y)z'



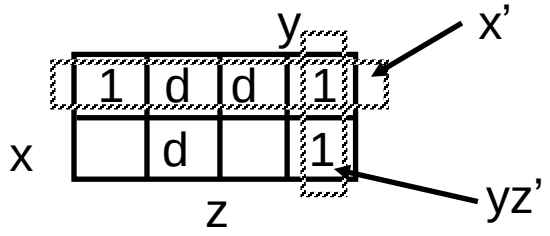
IMPLEMENTATION OF K-MAPS
- Don't-Care Conditions -

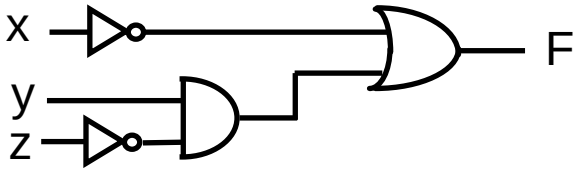
In some logic circuits, the output responses for some input conditions are don't care whether they are 1 or 0.

In K-maps, don't-care conditions are represented by d's in the corresponding cells.

Don't-care conditions are useful in minimizing the logic functions using K-map.

- **Can be considered either 1 or 0**
- **Thus increases the chances of merging cells into the larger cells**
  --> Reduce the number of variables in the product terms





COMBINATIONAL LOGIC CIRCUITS

Half Adder

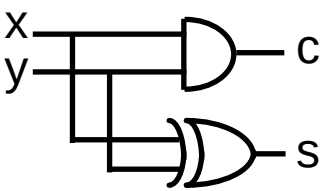
x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

x \ y	0	1
0	0	0
1	0	1

$c = xy$

x \ y	0	1
0	0	1
1	1	0

$s = xy' + x'y$
$= x \oplus y$



Full Adder

x	y	c_{n-1}	c_n	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

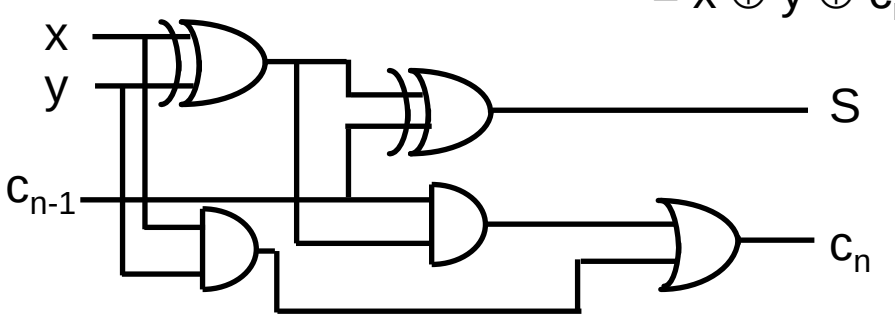
x \ y \ c_{n-1}	0	1
0	0	0
1	0	1
1	1	1
0	1	1

c_n

x \ y \ c_{n-1}	0	1
0	0	1
1	1	0
0	1	0
1	0	1

s

$c_n = xy + xc_{n-1} + yc_{n-1}$
$= xy + (x \oplus y)c_{n-1}$
$s = x'y'c_{n-1} + x'yc'_{n-1} + xy'c'_{n-1} + xyc_{n-1}$
$= x \oplus y \oplus c_{n-1} = (x \oplus y) \oplus c_{n-1}$



COMBINATIONAL LOGIC CIRCUITS

Other Combinational Circuits

- Multiplexer
- Encoder
- Decoder
- Parity Checker
- Parity Generator
- etc

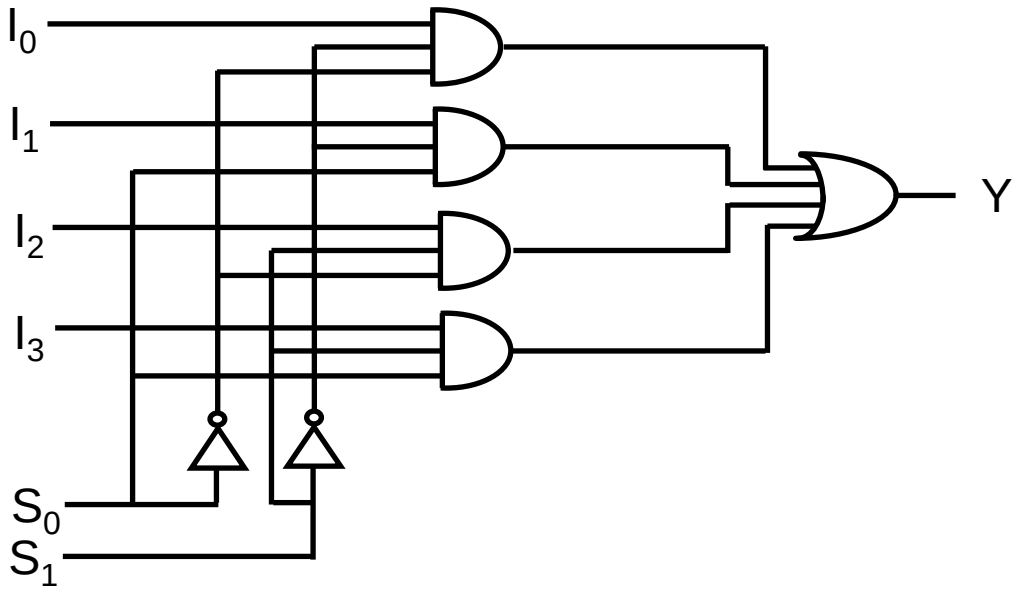
Multiplexer

- Definition
- Logic Diagram
- Application

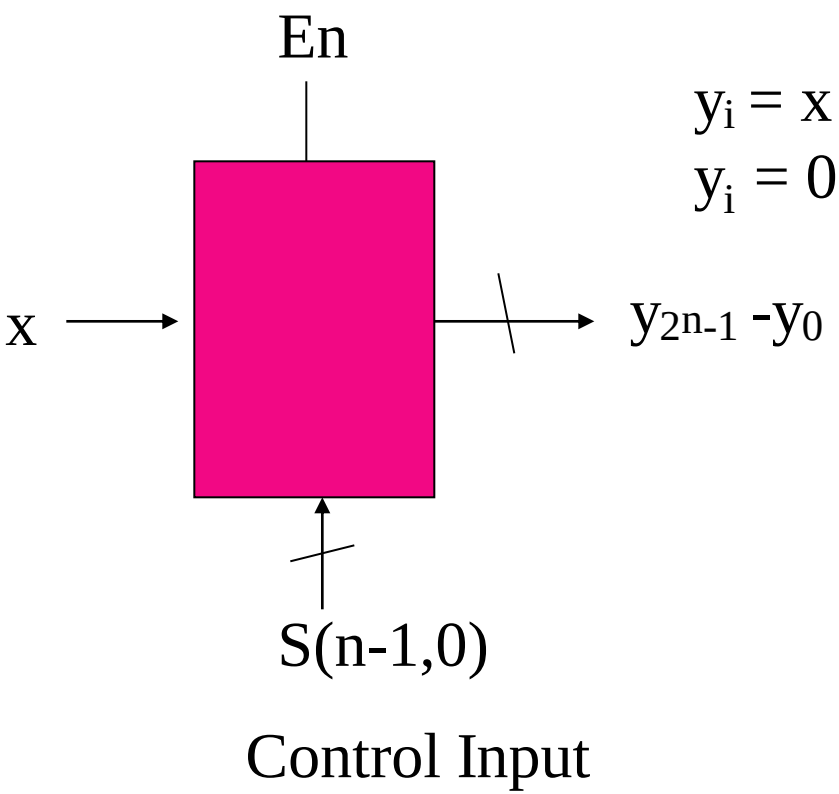
MULTIPLEXER

4-to-1 Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



4. Demultiplexers



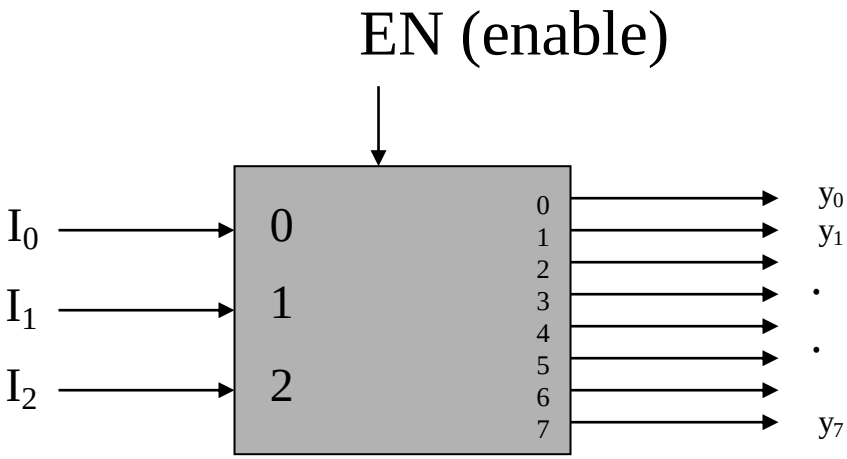
$y_i = x$ if $i = (S_{n-1}, .. , S_0)$ & En = 1

$y_i = 0$ otherwise

1. Decoder

- Definition
- Logic Diagram
- Application (Universal Set)
- Tree of Decoders

1. Decoder: Definition



n inputs
n= 3

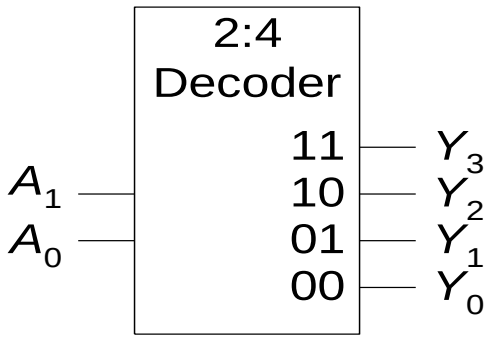
2^n outputs
2^3= 8

n to 2^n decoder function:

y_i = 1 if En= 1 & (I_2, I_1, I_0) = i
y_i= 0 otherwise

1. Decoder: Definition

- N inputs, 2^N outputs
- One-hot outputs: only one output HIGH at once

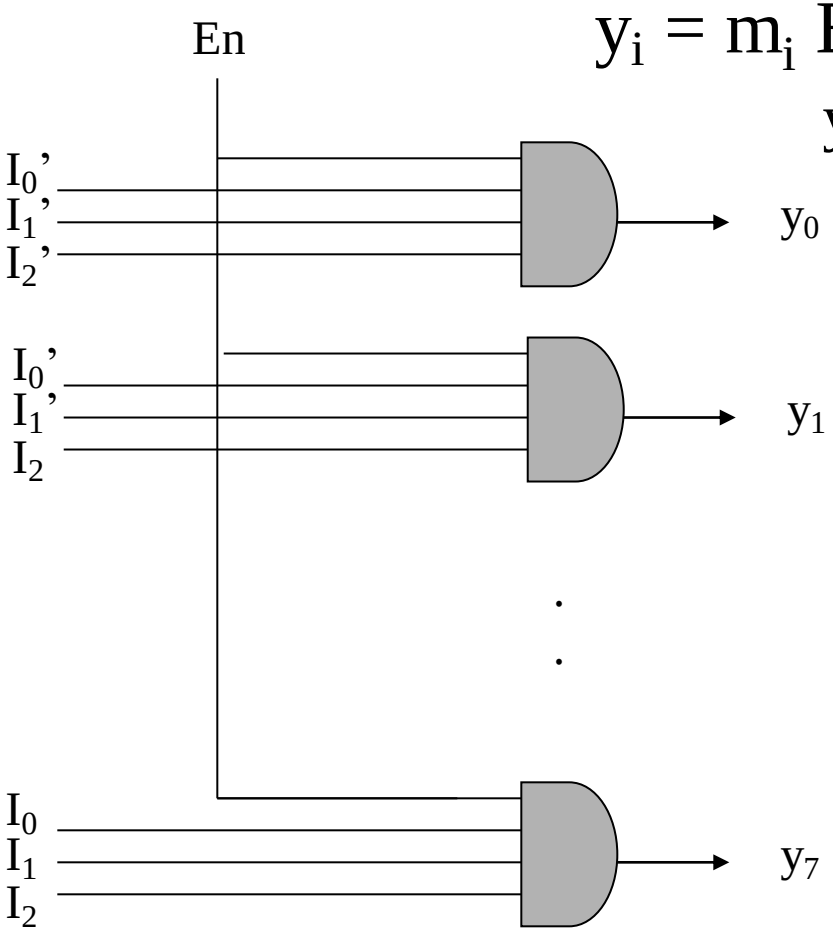


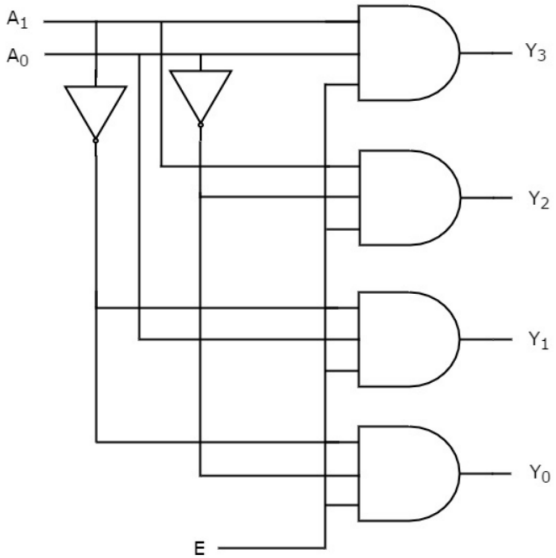
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Decoder: Logic Diagram

$y_i = m_i \, En$

$y_0 = 1$ if $(I_2, I_1, I_0) = (0,0,0)$ & $En = 1$

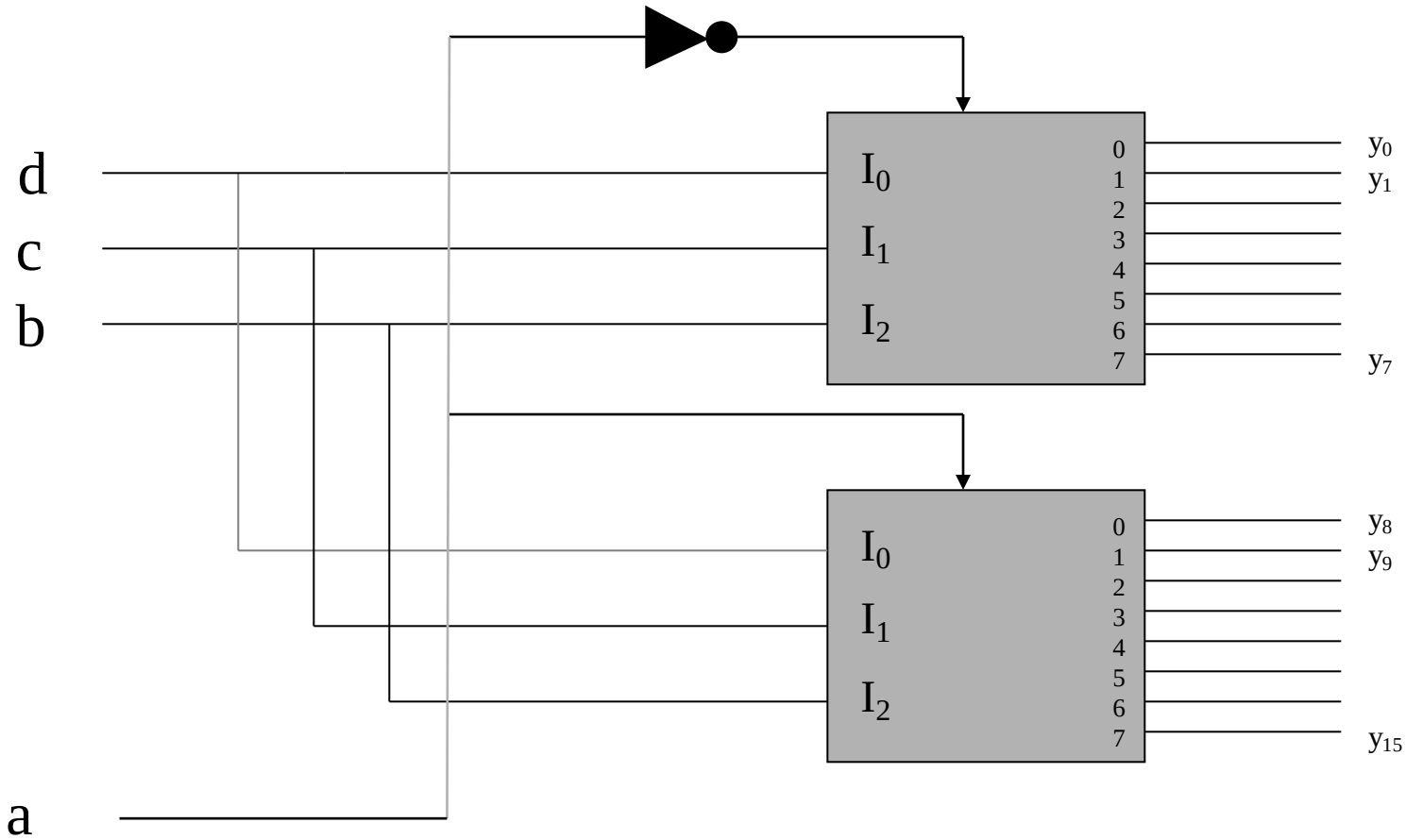




$y_7 = I_2 I_1 I_0 En$

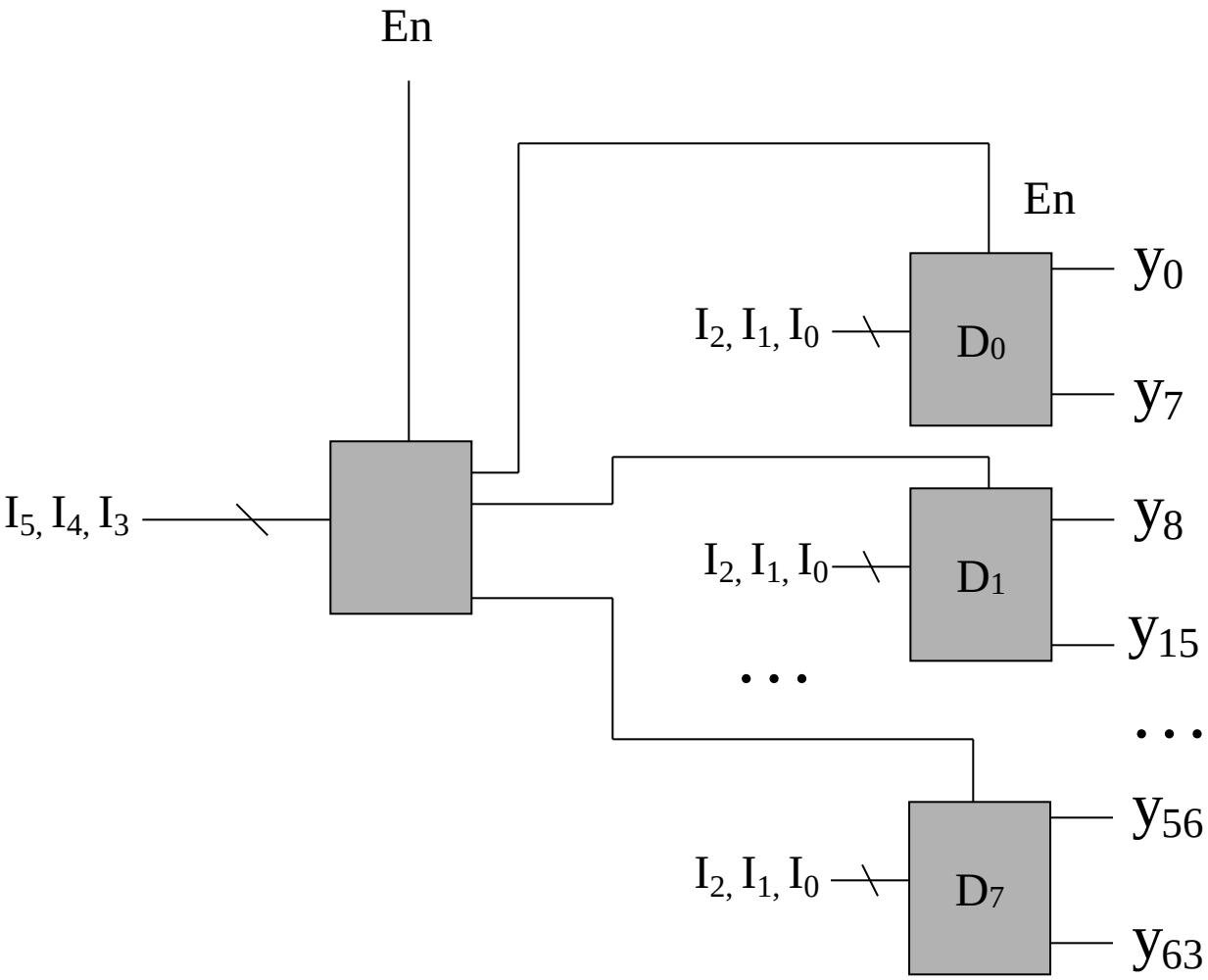
Tree of Decoders

Implement a 4-2^4 decoder with 3-2^3 decoders.



Tree of Decoders

Implement a 6-2^6 decoder with 3-2^3 decoders.



Tree of Decoders

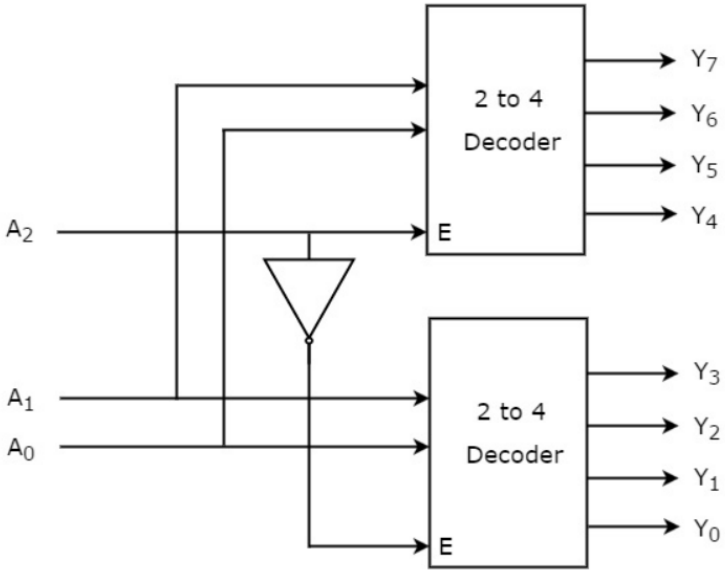
$$Required\ number\ of\ lower\ order\ decoders = \frac{m_2}{m_1}$$

Where,

m_1 is the number of outputs of lower order decoder.

m_2 is the number of outputs of higher order decoder.

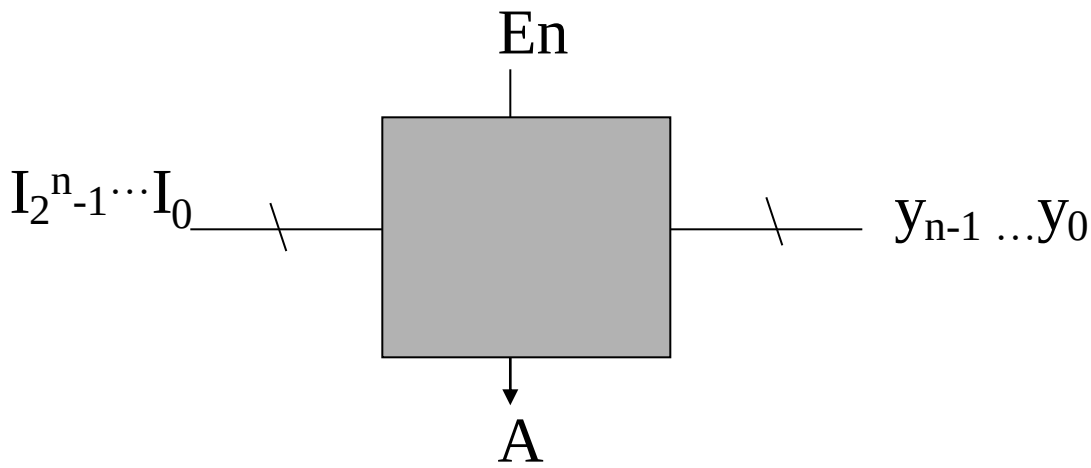
Here, m_1 = 4 and m_2 = 8. Substitute, these two values in the above formula.

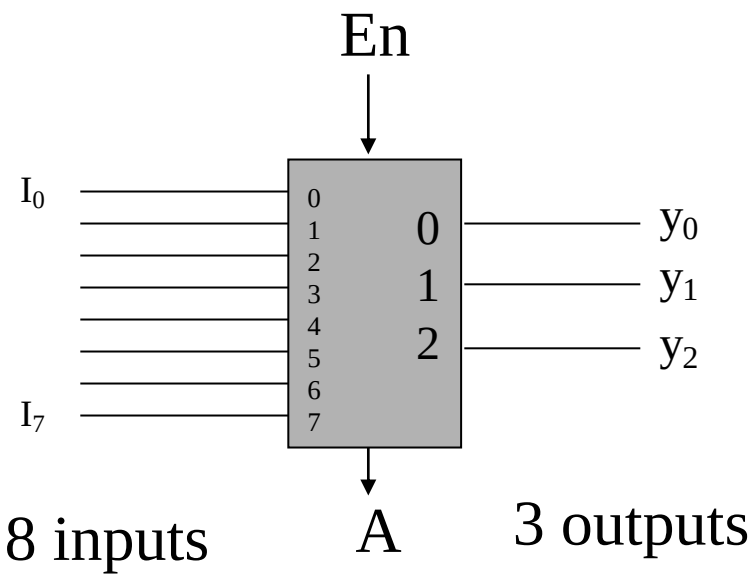


2. Encoder

- Definition
- Logic Diagram
- Priority Encoder

2. Encoder: Definition





Encoder Description:

At most one $I_i = 1$.

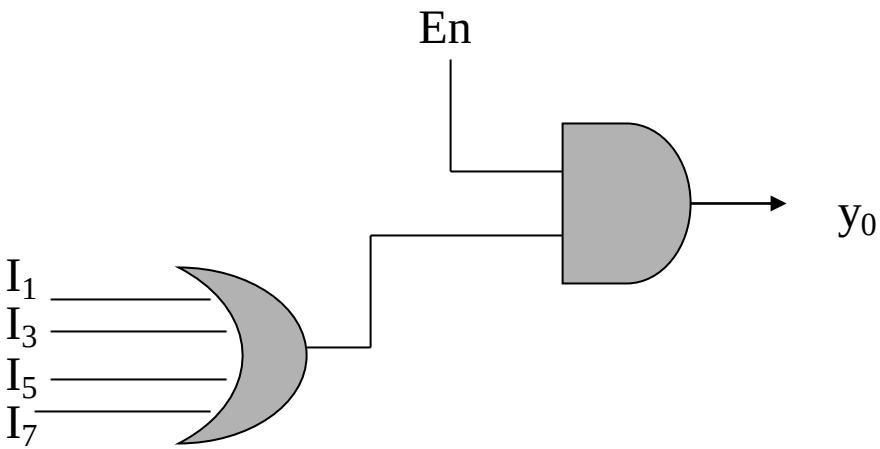
$(y_{n-1},.., y_0) = i$ if $I_i = 1$ & En = 1

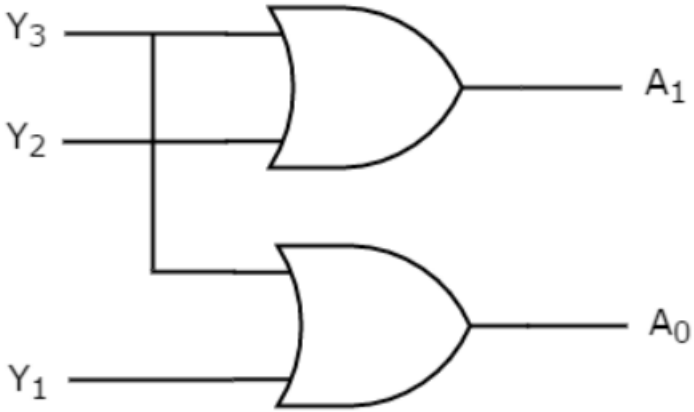
$(y_{n-1},.., y_0) = 0$ otherwise.

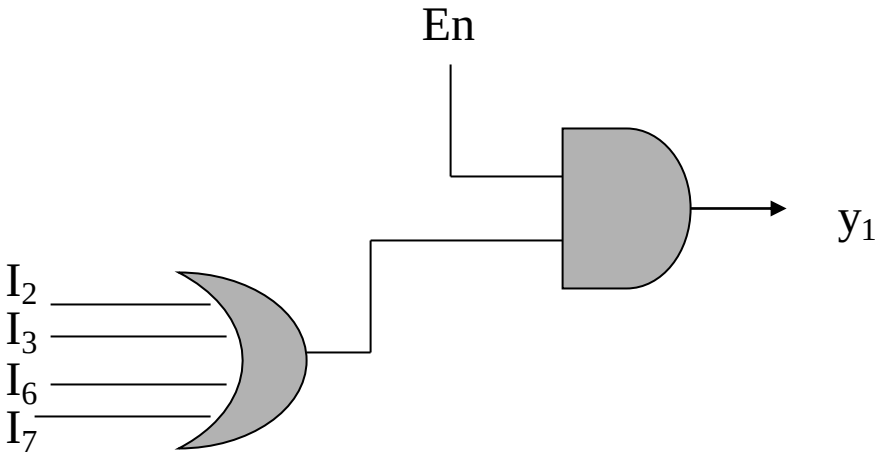
A = 1 if En = 1 and one i s.t. $I_i = 1$

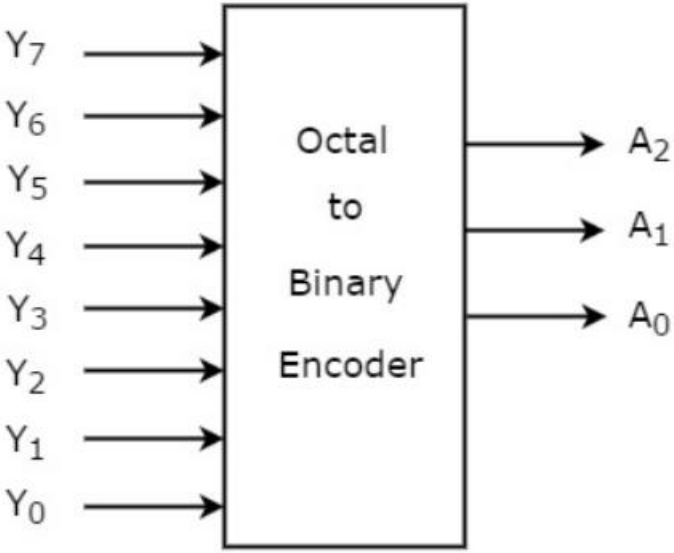
A = 0 otherwise.

Encoder: Logic Diagram

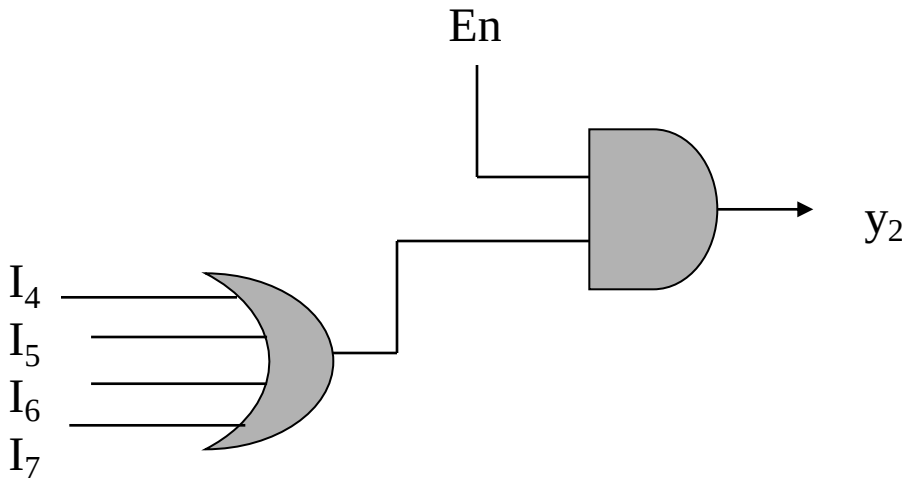


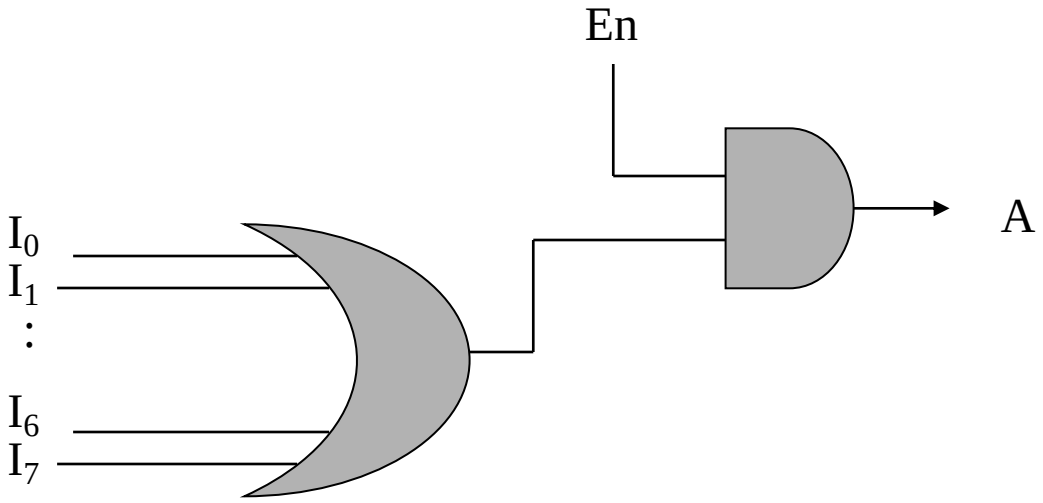






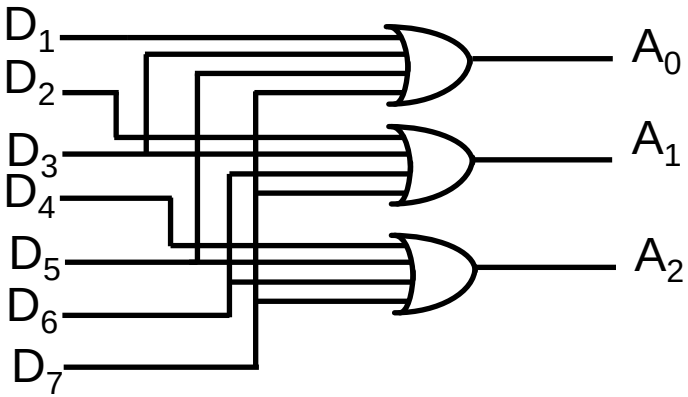
Encoder: Logic Diagram





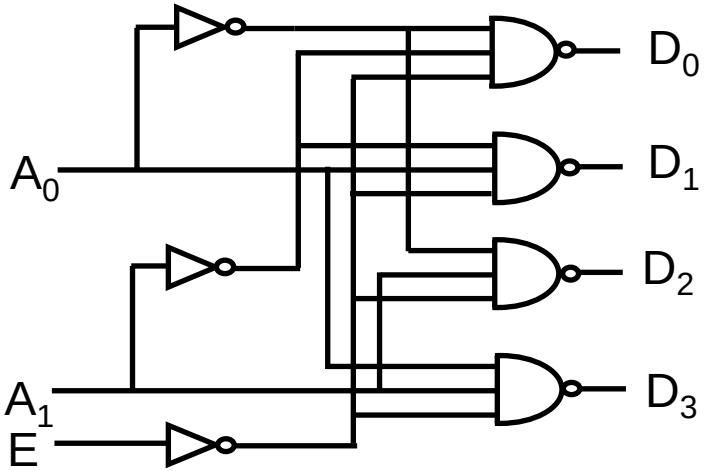
ENCODER/DECODER

Octal-to-Binary Encoder



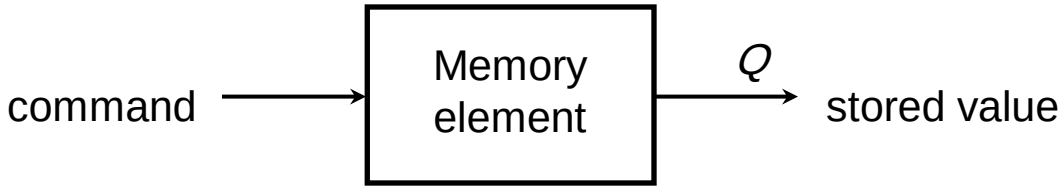
2-to-4 Decoder

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	d	d	1	1	1	1



Memory Elements

- Memory element: a device which can remember value indefinitely, or change value on command from its inputs.



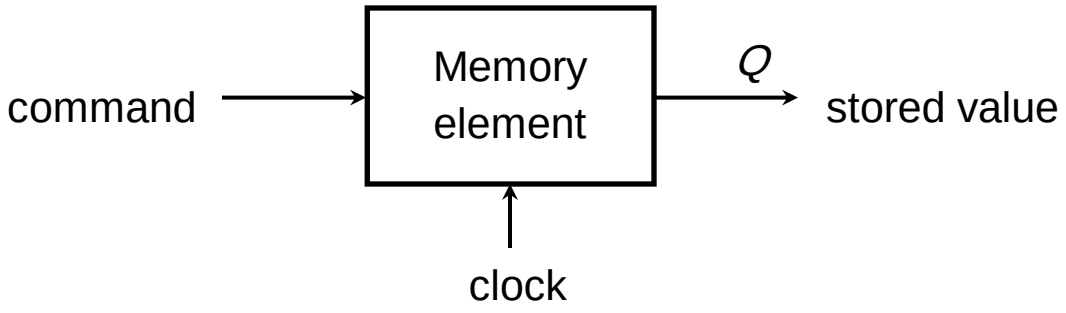
- Characteristic table:

Command (at time *t*)	*Q(t)*	*Q(t+1)*
Set	X	1
Reset	X	0
Memorise / No Change	0	0
	1	1

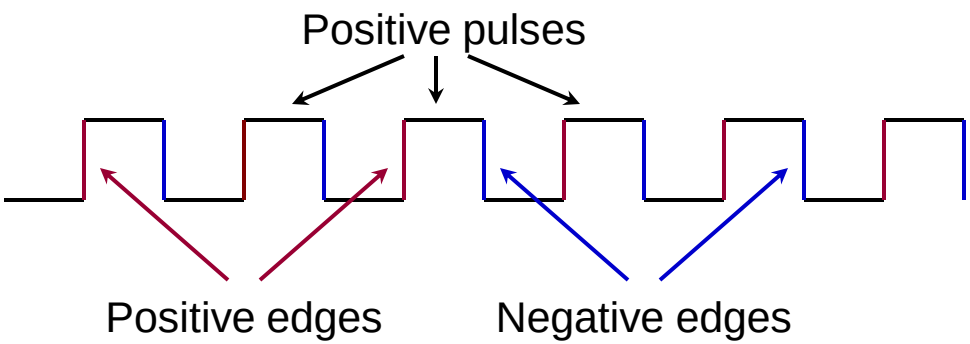
Q(t): current state
Q(t+1) or Q^+: next state

Memory Elements

- Memory element with clock. Flip–flops are memory elements that change state on clock signals.



- Clock is usually a square wave.



Memory Elements

- Two types of triggering/activation:
  - pulse-triggered
  - edge-triggered
- Pulse-triggered
  - latches
  - ON = 1, OFF = 0
- Edge-triggered
  - flip-flops
  - positive edge-triggered (ON = from 0 to 1; OFF = other time)
  - negative edge-triggered (ON = from 1 to 0; OFF = other time)

S-R Latch

- *Complementary* outputs: *Q* and *Q′*.
- When *Q* is HIGH, the latch is in *SET* state.
- When *Q* is LOW, the latch is in *RESET* state.
- For *active-HIGH input* S-R latch (also known as NOR gate latch),

  R=HIGH (and *S*=LOW) ⇨ RESET state

  S=HIGH (and *R*=LOW) ⇨ SET state

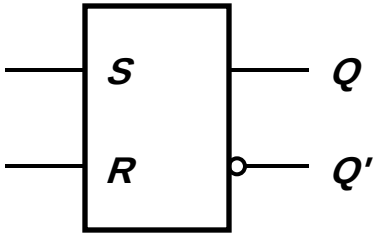
  both inputs LOW ⇨ no change

  both inputs HIGH ⇨ *Q* and *Q′* both LOW (invalid)!

S-R Latch

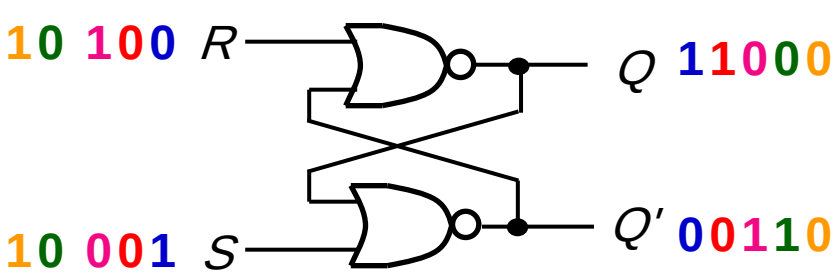
- Characteristics table for active-high input S-R latch:

S	R	Q	Q'	
0	0	NC	NC	No change. Latch remained in present state.
1	0	1	0	Latch SET.
0	1	0	1	Latch RESET.
1	1	0	0	Invalid condition.



S-R Latch

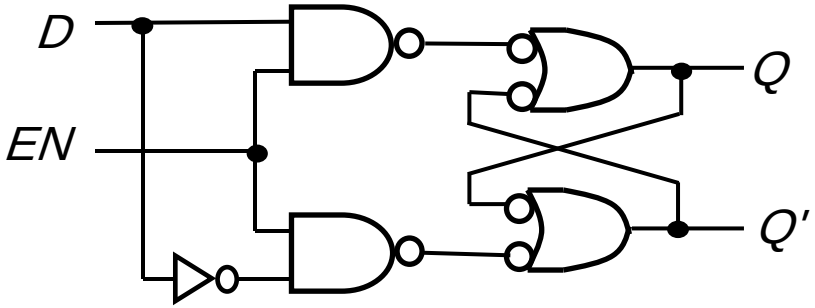
- Active-HIGH input S-R latch

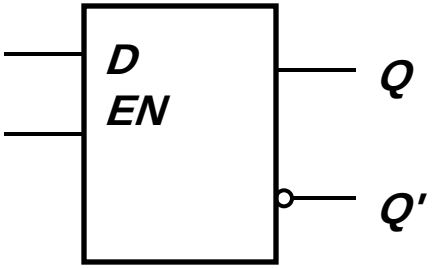


S	*R*	*Q*	*Q′*	
1	0	1	0	initial
0	0	1	0	(afer *S*=1, *R*=0)
0	1	0	1	
0	0	0	1	(after *S*=0, *R*=1)
1	1	0	0	invalid!

Gated D Latch

- Make *R* input equal to *S*′ → *gated D latch.*
- *D* latch eliminates the undesirable condition of invalid state in the *S*–*R* latch.





Gated D Latch

- When *EN* is HIGH,
  - *D*=HIGH → latch is SET
  - *D*=LOW → latch is RESET
- Hence when *EN* is HIGH, *Q* 'follows' the *D* (data) input.
- Characteristic table:

EN	*D*	*Q(t+1)*	
1	0	0	Reset
1	1	1	Set
0	X	*Q(t)*	No change

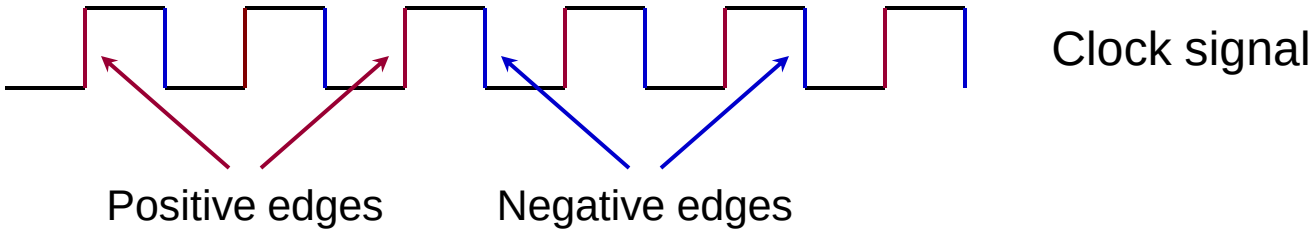
When *EN*=1, *Q(t+1)* = *D*

Latch Circuits: Not Suitable

- Latch circuits are not suitable in synchronous logic circ uits.
- When the enable signal is active, the excitation inputs a re gated directly to the output Q. Thus, any change in the excitation input immediately causes a change in the latch output.
- The problem is solved by using a special timing control signal called a *clock* to restrict the times at which the states of the memory elements may change.
- This leads us to the edge-triggered memory elements called *flip-flops*.

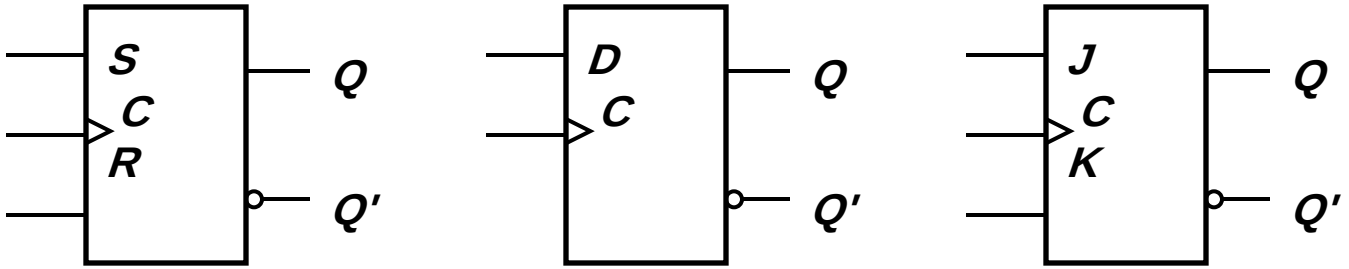
Edge-Triggered Flip-flops

- *Flip–flops*: synchronous bistable devices
- Output changes state at a specified point on a trig gering input called the *clock*.
- Change state either at the *positive edge* (rising edg e) or at the *negative edge* (*falling edge*) of the cloc k signal.

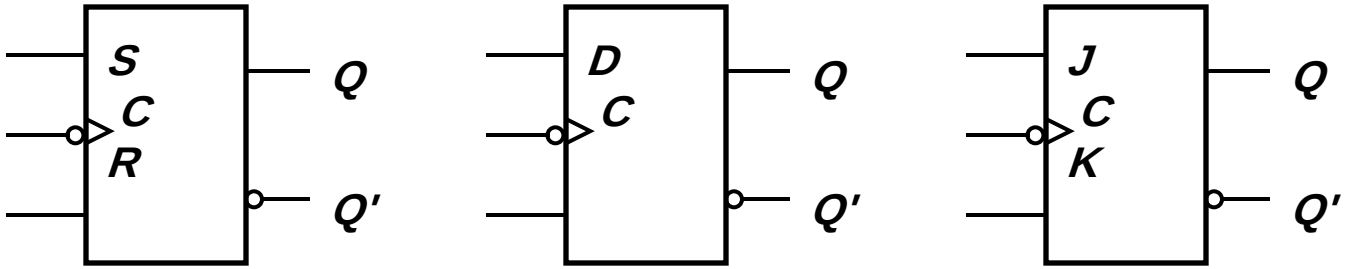


Edge-Triggered Flip-flops

- S–R, D and J–K edge–triggered flip–flops. Note th e “>” symbol at the clock input.



Positive edge-triggered flip-flops



Negative edge-triggered flip-flops

S-R Flip-flop

- S–R flip–flop: on the triggering edge of the clock pu lse,
  - *S*=HIGH (and *R*=LOW) ⇨ SET state
  - *R*=HIGH (and *S*=LOW) ⇨ RESET state
  - both inputs LOW ⇨ no change
  - both inputs HIGH ⇨ invalid
- Characteristic table of positive edge–triggered S–R flip–flop:

S	*R*	*CLK*	*Q(t+1)*	Comments
0	0	X	*Q(t)*	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	?	Invalid

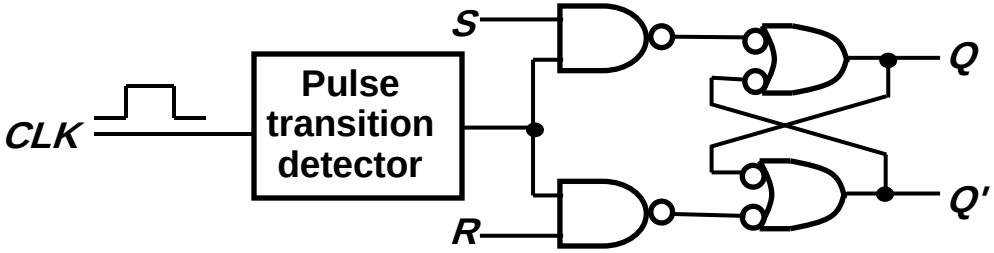
X = irrelevant ("don't care")
↑ = clock transition LOW to HIGH

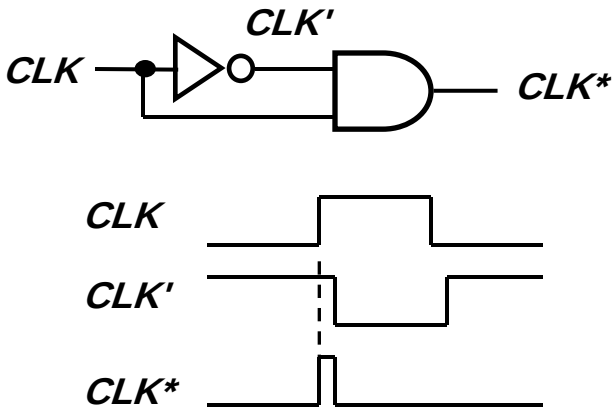
S-R Flip-flop

- It comprises 3 parts:
  - a basic *NAND latch*
  - a *pulse-steering* circuit
  - a *pulse transition detector* (or *edge detector*) circuit
- The pulse transition detector detects a rising (or fall ing) edge and produces a very *short-duration spike* .

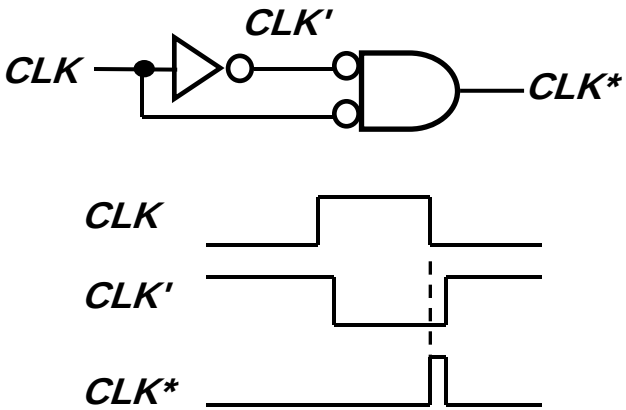
S-R Flip-flop

The pulse transition detector.





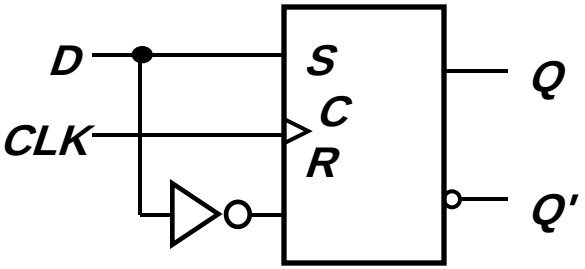
Positive-going transition
(rising edge)



Negative-going transition
(falling edge)

D Flip-flop

- D flip-flop: single input *D* (data)
  - *D*=HIGH ⇨ SET state
  - *D*=LOW ⇨ RESET state
- *Q* follows *D* at the clock edge.
- Convert S-R flip-flop into a D flip-flop: add an inverter.



A positive edge-triggered D flip-flop formed with an S-R flip-flop.

D	CLK	Q(t+1)	Comments
1	↑	1	Set
0	↑	0	Reset

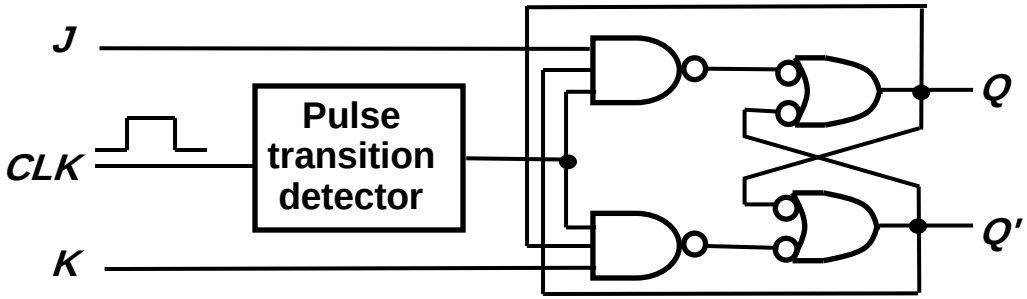
↑ = clock transition LOW to HIGH

J-K Flip-flop

- J–K flip–flop: Q and Q' are fed back to the pulse–steering NAND gates.
- No invalid state.
- Include a *toggle* state.
  - *J*=HIGH (and *K*=LOW) ⇨ SET state
  - *K*=HIGH (and *J*=LOW) ⇨ RESET state
  - both inputs LOW ⇨ no change
  - both inputs HIGH ⇨ toggle

J-K Flip-flop

- J–K flip–flop.



- Characteristic table.

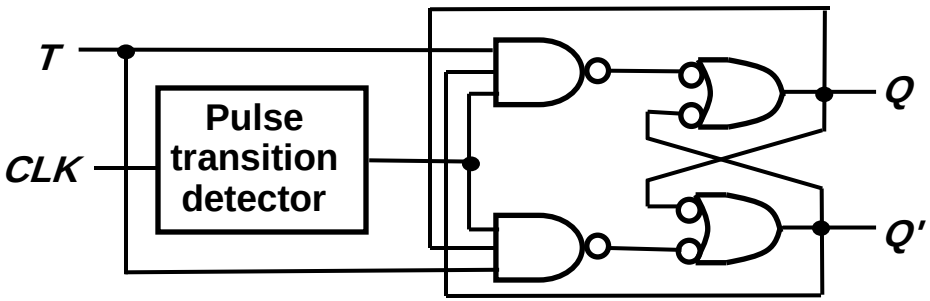
J	K	CLK	Q(t+1)	Comments
0	0	↑	Q(t)	No change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(t)'	Toggle

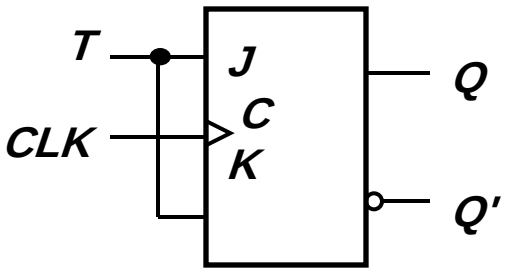
$$Q(t+1) = J.Q' + K'.Q$$

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

T Flip-flop

- T flip–flop: single–input version of the J–K flip flop, formed by tying both inputs together.





- Characteristic table.

T	CLK	Q(t+1)	Comments
0	↑	Q(t)	No change
1	↑	Q(t)'	Toggle

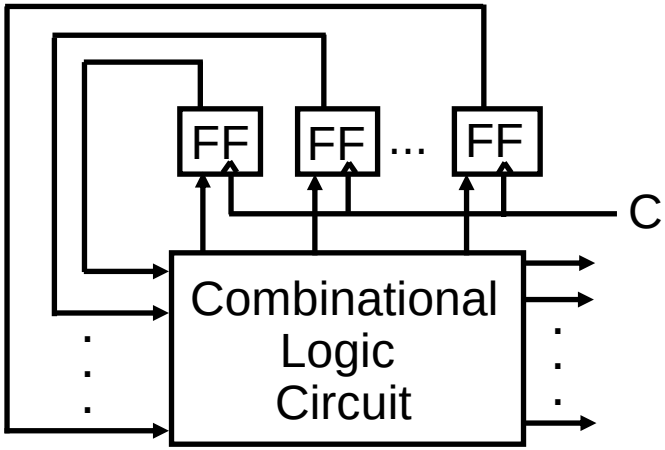
Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

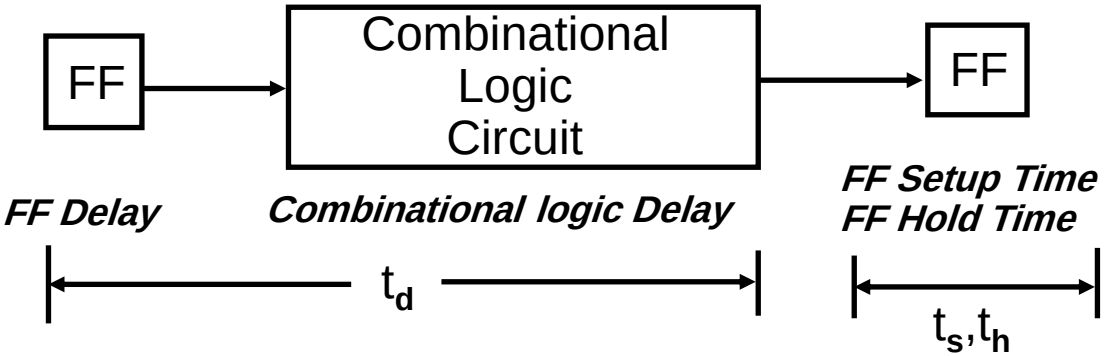
$$Q(t+1) = T.Q' + T'.Q$$

CLOCK PERIOD

Clock period determines how fast the digital circuit operates.
How can we determine the clock period ?

Usually, digital circuits are sequential circuits which has some flip flops





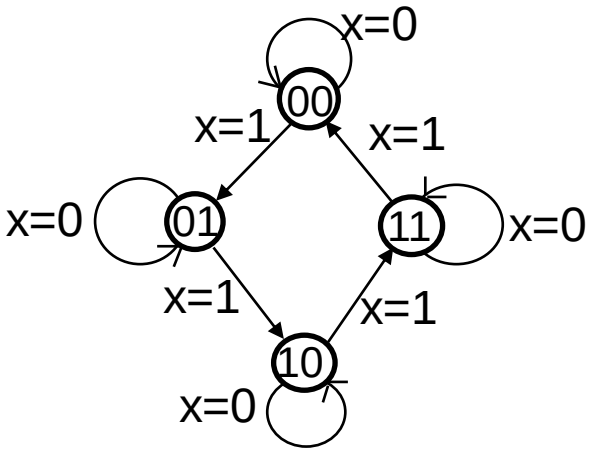
clock period $T = t_d + t_s + t_h$

DESIGN EXAMPLE

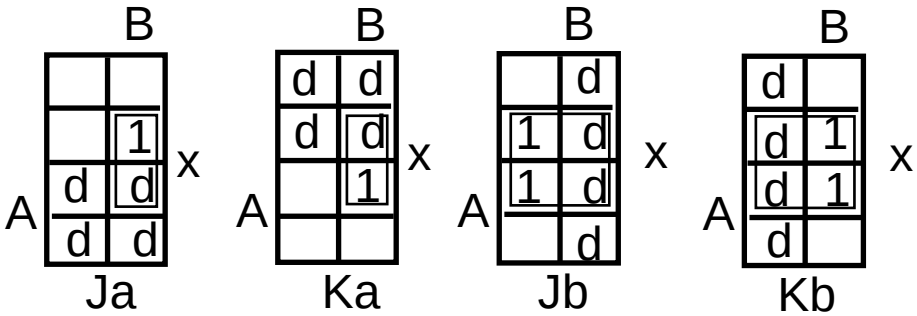
Design Procedure:

Specification ⇒ State Diagram ⇒ State Table ⇒ Excitation Table ⇒ Karnaugh Map ⇒ Circuit Diagram

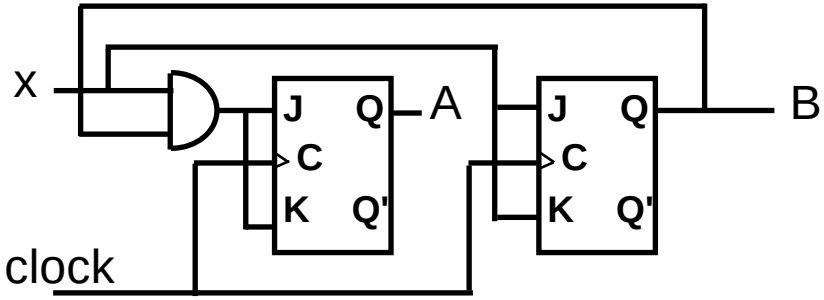
Example: 2-bit Counter -> 2 FF's



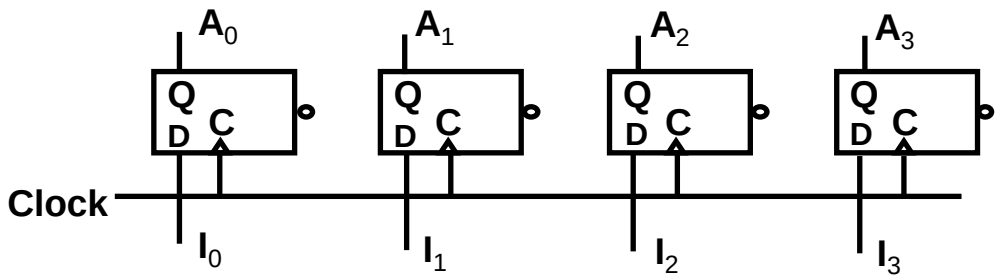
current state		input	next state		FF inputs			
A	B	x	A	B	Ja	Ka	Jb	Kb
0	0	0	0	0	0	d	0	d
0	0	1	0	1	0	d	1	d
0	1	0	0	1	0	d	d	0
0	1	1	1	0	1	d	d	1
1	0	0	1	0	d	0	0	d
1	0	1	1	1	d	0	1	d
1	1	0	1	1	d	0	d	0
1	1	1	0	0	d	1	d	1



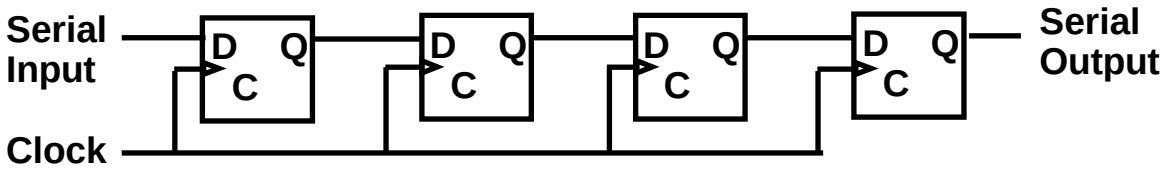
Ja = Bx Ka = Bx Jb = x Kb = x



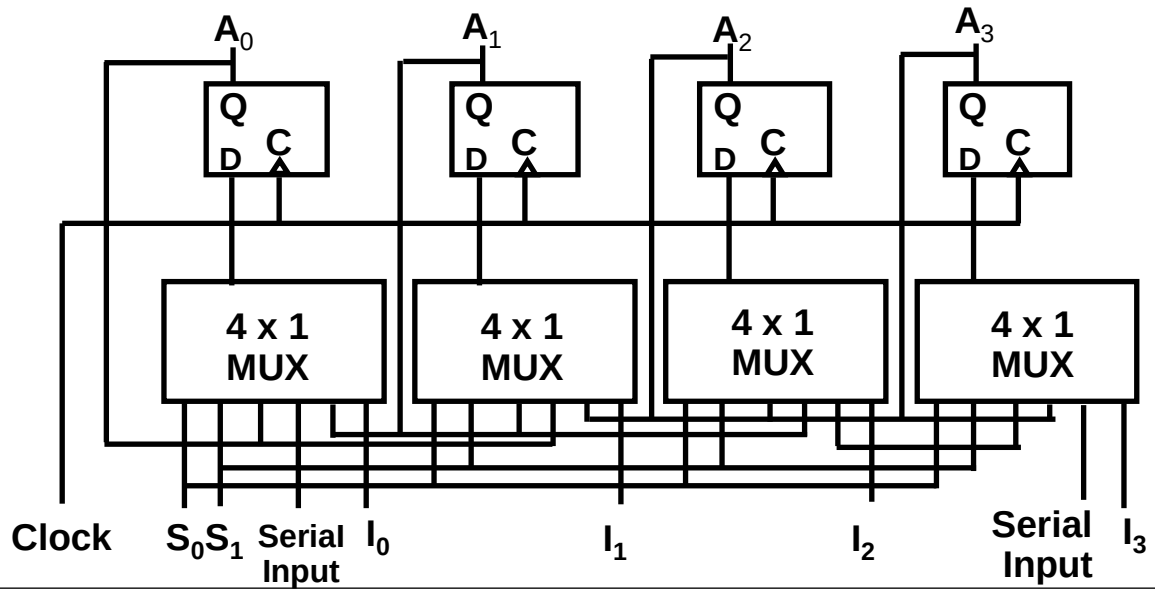
SEQUENTIAL CIRCUITS - Registers



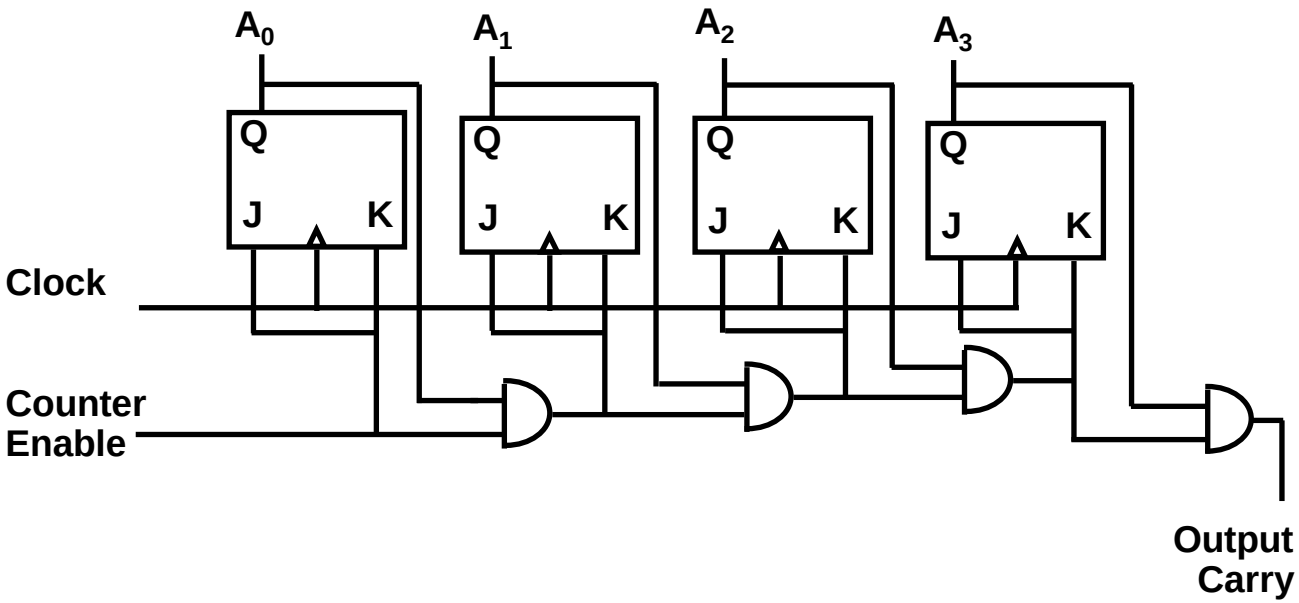
Shift Registers



Bidirectional Shift Register with Parallel Load

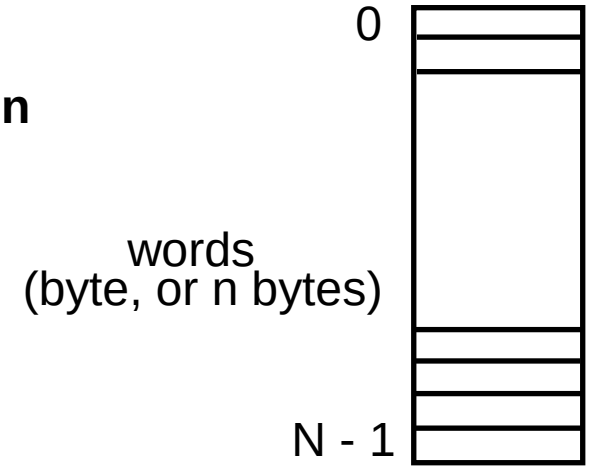


SEQUENTIUAL CIRCUITS - Counters



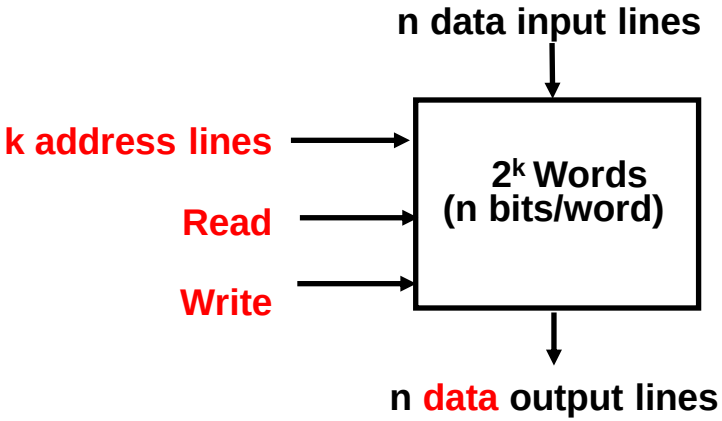
MEMORY COMPONENTS

Logical Organization



Random Access Memory

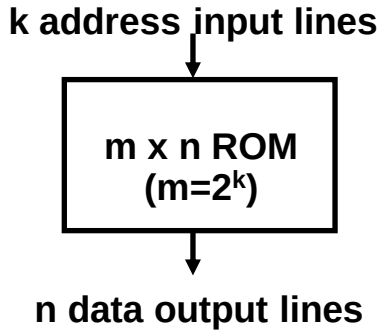
- **Each word has a unique address**
- **Access to a word requires the same time independent of the location of the word**
- **Organization**



READ ONLY MEMORY(ROM)

Characteristics

- Perform read operation only, write operation is not possible
- Information stored in a ROM is made permanent during production, and cannot be changed
- Organization



Information on the data output line depends only on the information on the address input lines.

--> Combinational Logic Circuit

X_0=A'B' + B'C
X_1=A'B'C + A'BC'
X_2=BC + AB'C'
X_3=A'BC' + AB'
X_4=AB

X_0=A'B'C' + A'B'C + AB'C
X_1=A'B'C + A'BC'
X_2=A'BC + AB'C' + ABC
X_3=A'BC' + AB'C' + AB'C
X_4=ABC' + ABC

Canonical minterms

address ABC	Output X_0	X_1	X_2	X_3	X_4
000	1	0	0	0	0
001	1	1	0	0	0
010	0	1	0	1	0
011	0	0	1	0	0
100	0	0	1	1	0
101	1	0	0	1	0
110	0	0	0	0	1
111	0	0	1	0	1

TYPES OF ROM

ROM

- Store information (function) during production
- Mask is used in the production process
- Unalterable
- Low cost for large quantity production --> used in the final products

PROM (Programmable ROM)

- Store info electrically using PROM programmer at the user's site
- Unalterable
- Higher cost than ROM -> used in the system development phase
  -> Can be used in small quantity system

EPROM (Erasable PROM)

- Store info electrically using PROM programmer at the user's site
- Stored info is erasable (alterable) using UV light (electrically in some devices) and rewriteable
- Higher cost than PROM but reusable --> used in the system development phase. Not used in the system production due to erasability

INTEGRATED CIRCUITS

Classification by the Circuit Density

SSI - several (less than 10) independent gates
MSI - 10 to 200 gates; Perform elementary digital functions;
Decoder, adder, register, parity checker, etc
LSI - 200 to few thousand gates; Digital subsystem
Processor, memory, etc
VLSI - Thousands of gates; Digital system
Microprocessor, memory module

Classification by Technology

TTL - Transistor-Transistor Logic
Bipolar transistors
NAND
ECL - Emitter-coupled Logic
Bipolar transistor
NOR
MOS - Metal-Oxide Semiconductor
Unipolar transistor
High density
CMOS - Complementary MOS
Low power consumption