SYST 27198

CPU Architecture and Assembly

Dr. Rachel Jiang

Sheridan Institute of Technologies and Advanced Learning

Winter 2017

Acknowledgement

This notes is based on Professor Victor Ralevich's course Notes of

"**Structured Computer Organization**"

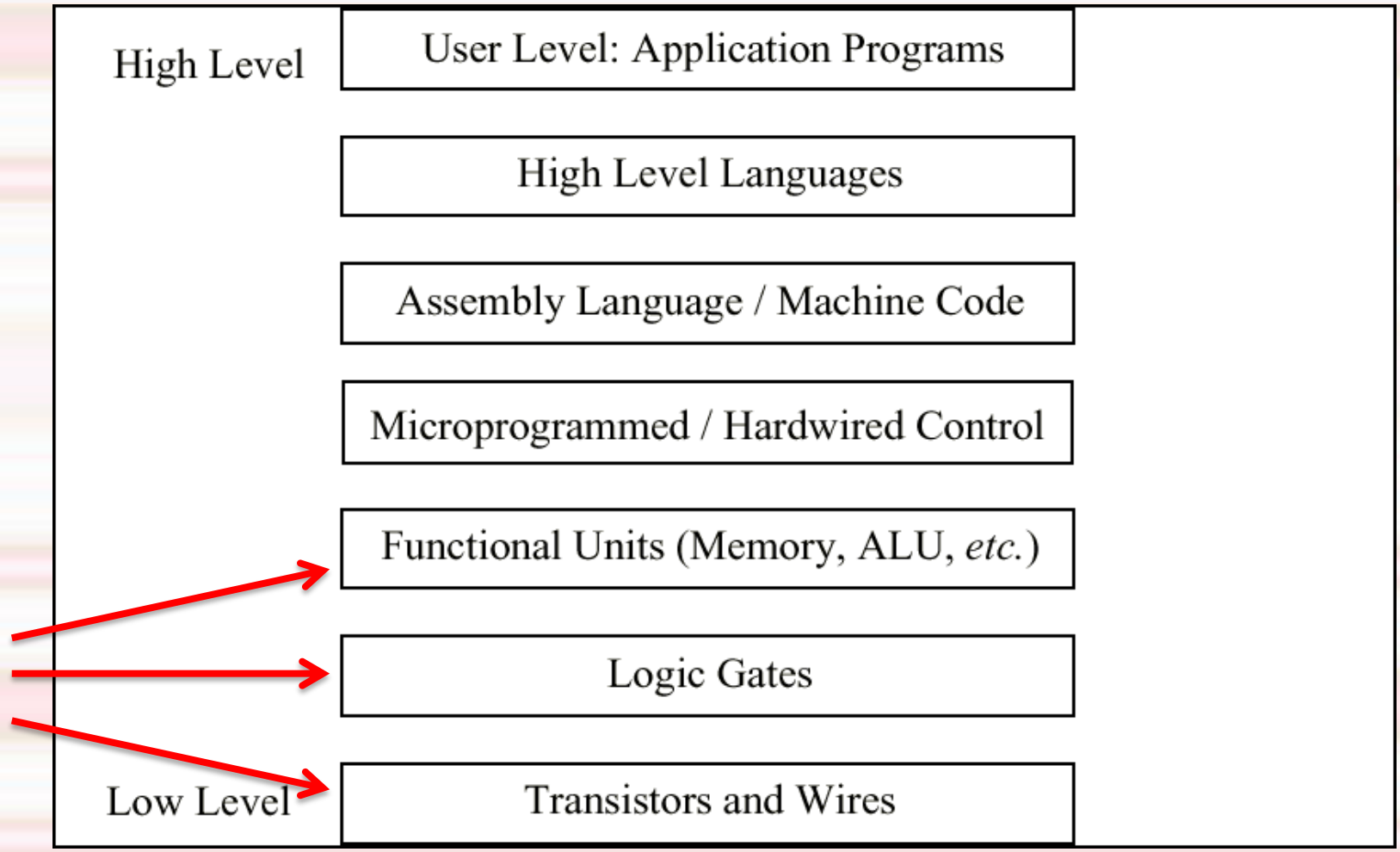
Updated by Rachel Jiang

Jan. 2017

Chapter Four

Digital Logic

Digital Logic

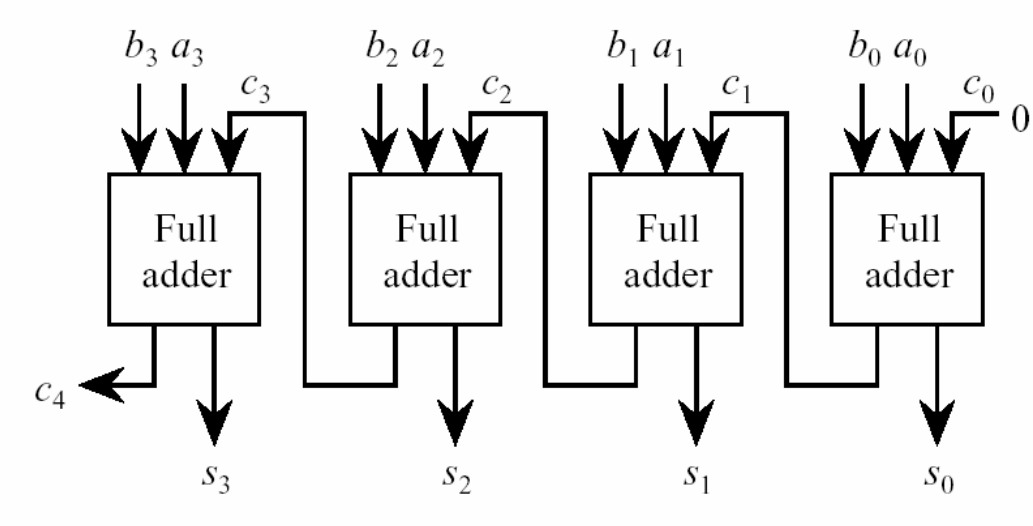


Topics

- The Combinational Logic Unit (CLU)
- The truth tables and Boolean algebra
- Logical gates and their symbols
- Transistor gates
- The majority function
- Positive vs. negative logic
- Digital components
  - (MUX, DEMUX, Decoder, Priority encoder, PLA)
- half-adder, full adder, and Shifters
- Ripple-carry addition
- Simplification of Boolean expressions

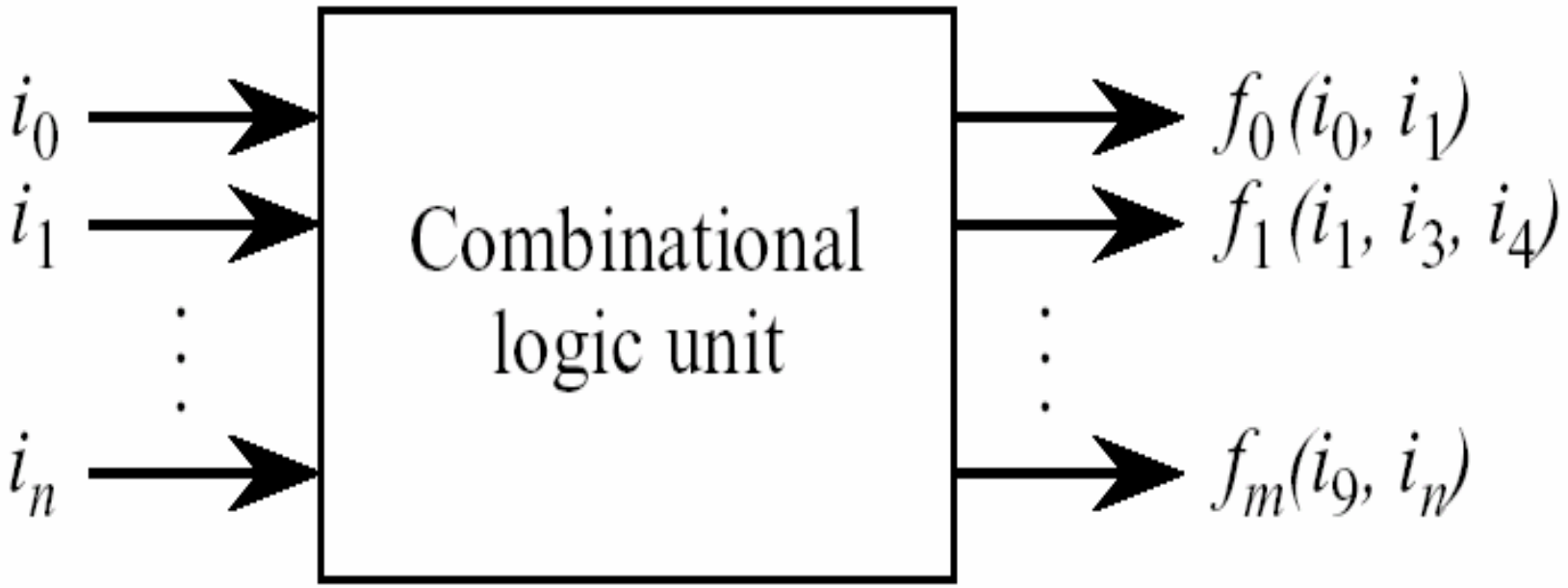
Combinational logic

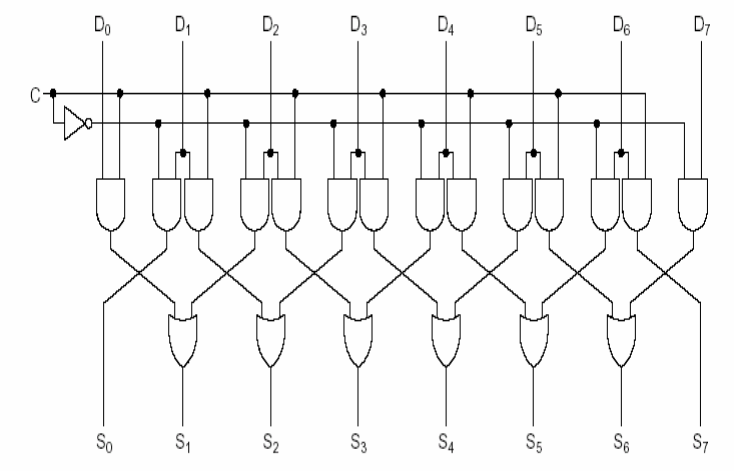
- translates a set of inputs into a set of outputs according to one or more mapping functions.
  - (e.g. an adder)

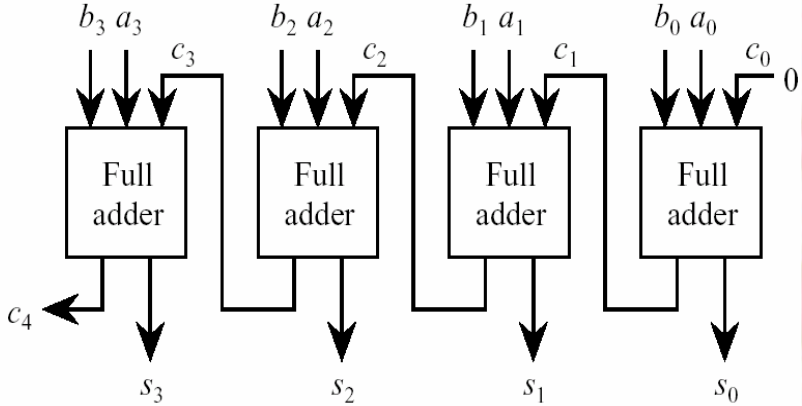


The Combinational Logic Unit (CLU)

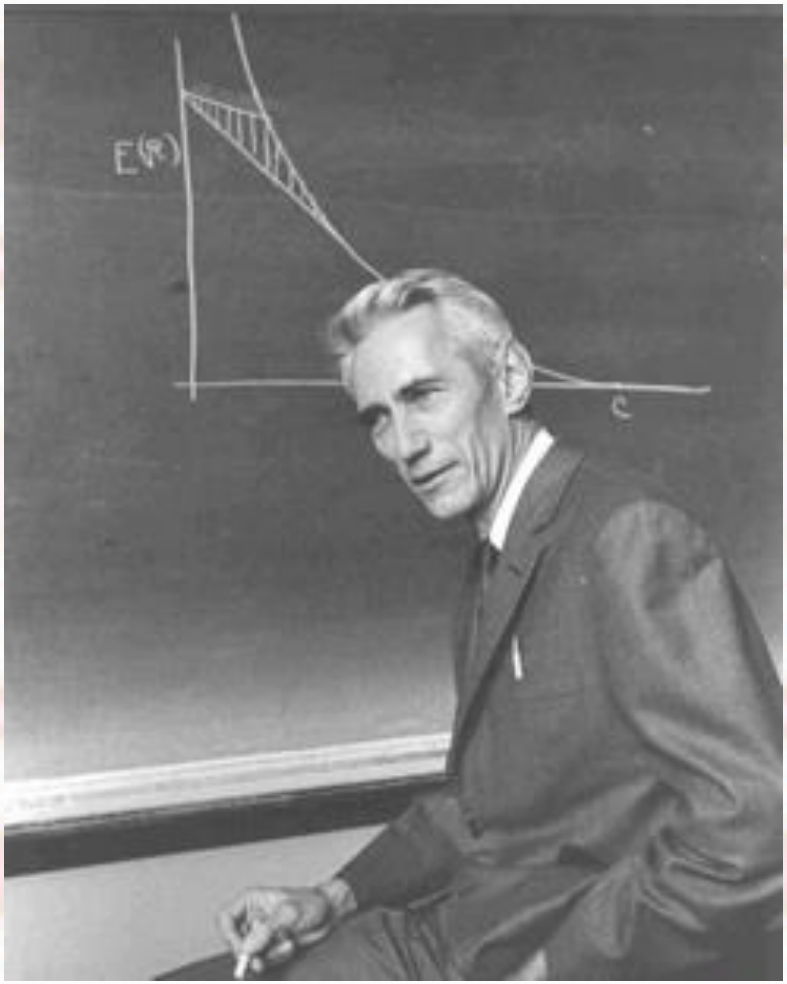
- a digital logic circuit in which logical decisions (output) are made based only on combinations of the inputs.
- Inputs and outputs
  - normally have two distinct (binary) values: high and low, 1 and 0, 0 and 1, or 5 V and 0 V.
- The **outputs** are
  - strictly functions of the inputs, and
  - updated immediately after the inputs change.
- A set of inputs $i_0 - i_n$ are presented to the CLU
  - produces a set of outputs according to functions $f_0 - f_m$.


i_0
i_1
i_n
Combinational logic unit
$f_0(i_0, i_1)$
$f_1(i_1, i_3, i_4)$
$f_m(i_9, i_n)$


D0 D1 D2 D3 D4 D5 D6 D7
C
S0 S1 S2 S3 S4 S5 S6 S7

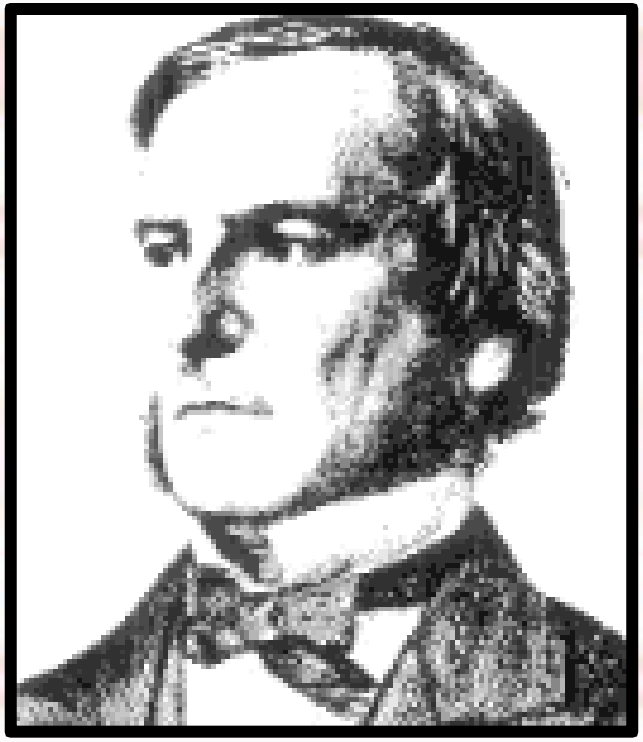

b_3 a_3
b_2 a_2
b_1 a_1
b_0 a_0
c_3
c_2
c_1
c_0
0
Full adder
Full adder
Full adder
Full adder
c_4
s_3
s_2
s_1
s_0

1937



Claude Shannon publishes *A Symbolic Analysis of Relays and Switching Circuits*, where he shows that Boole's algebra may be used in electrical systems.

1854

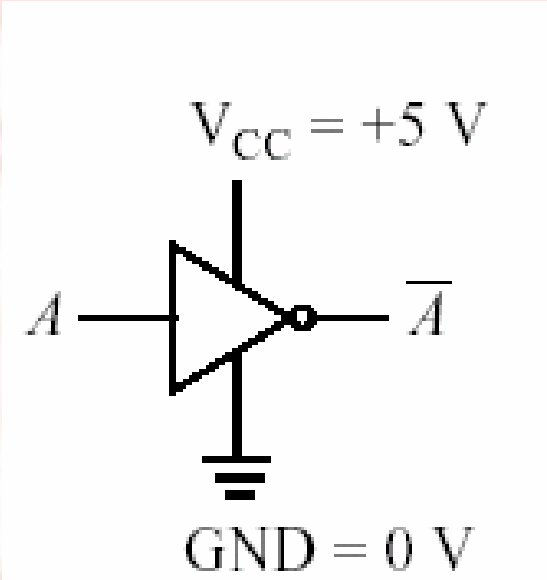


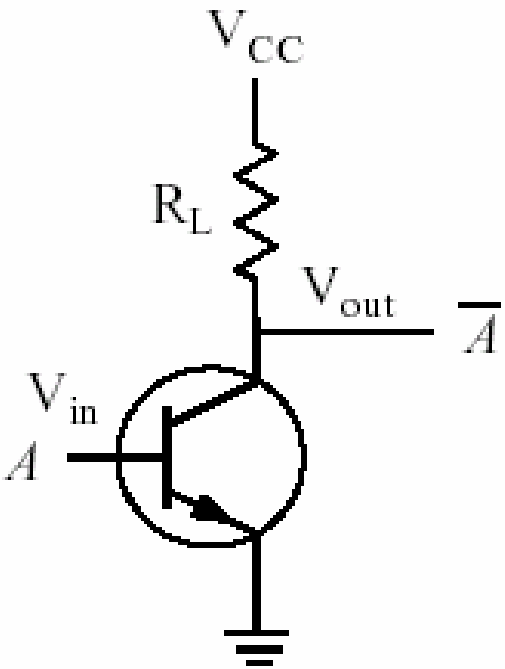
Gorge Boole (UK) creates the algebra that bears his name today (boolean).

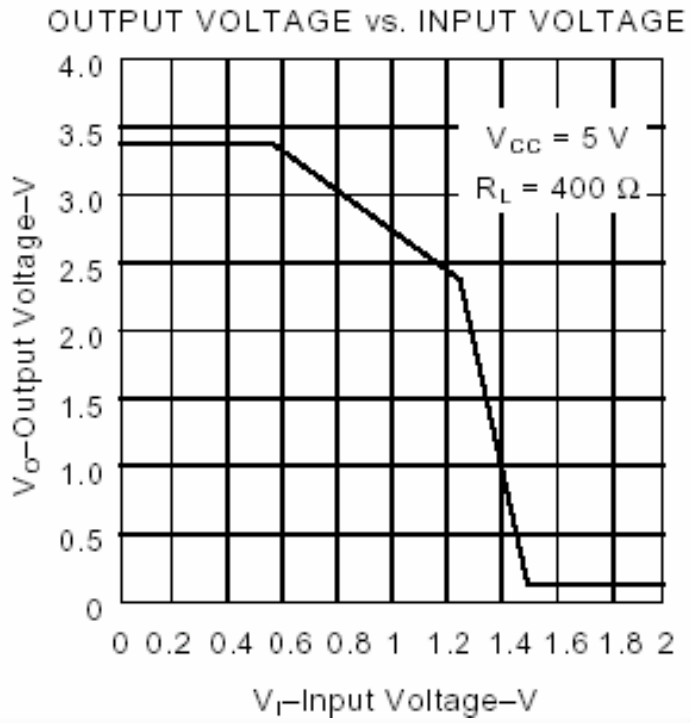
AND	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

OR	TRUE	FALSE
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE

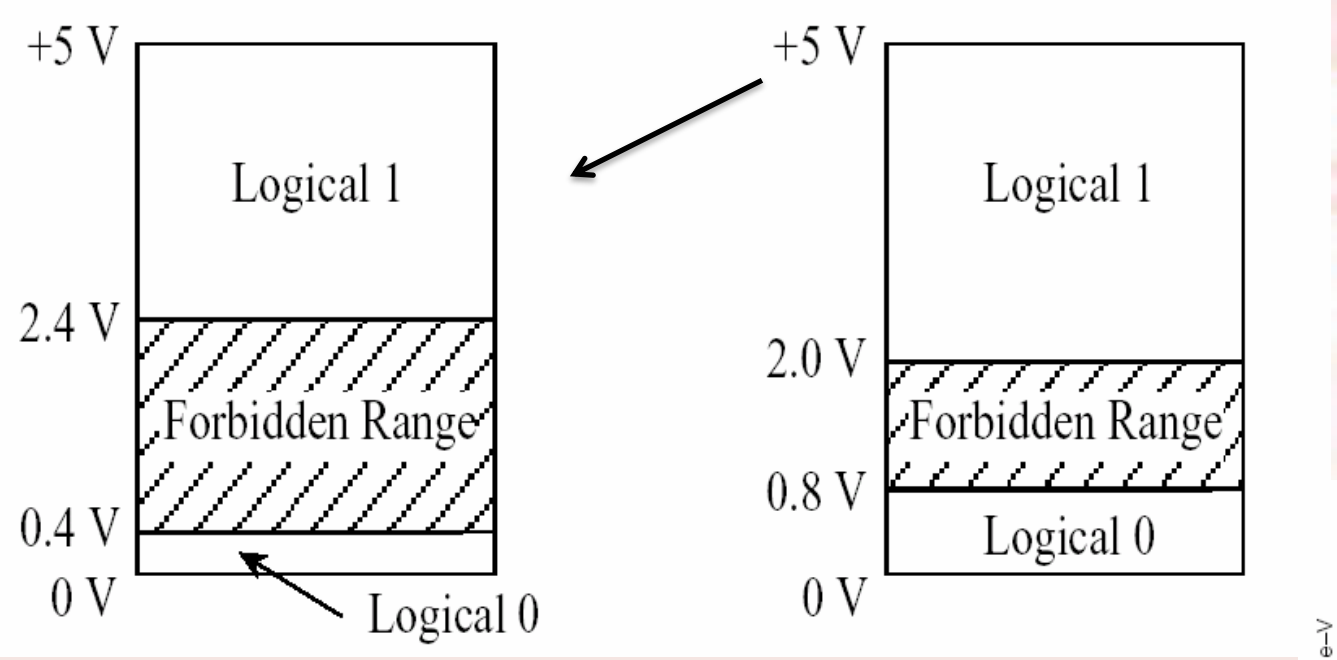
Transistor
Operation of Inverter





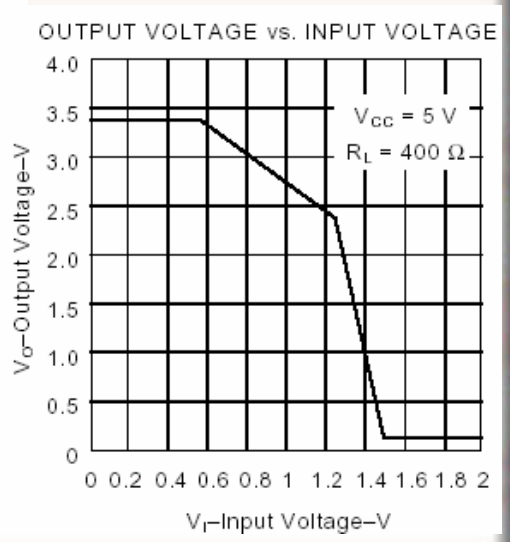


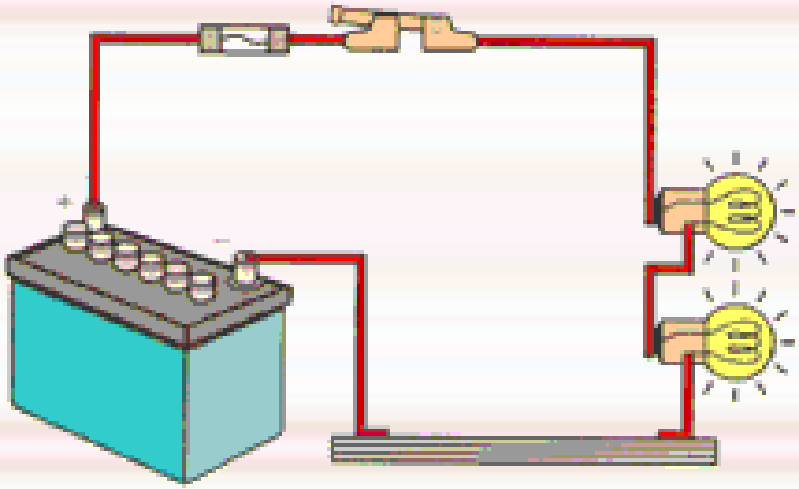
Assignments of 0 and 1 to Voltages

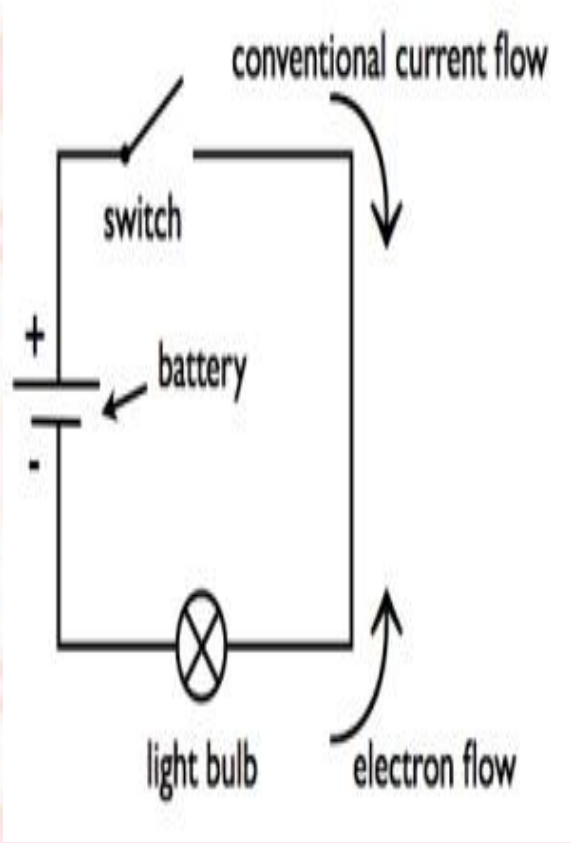


At the output

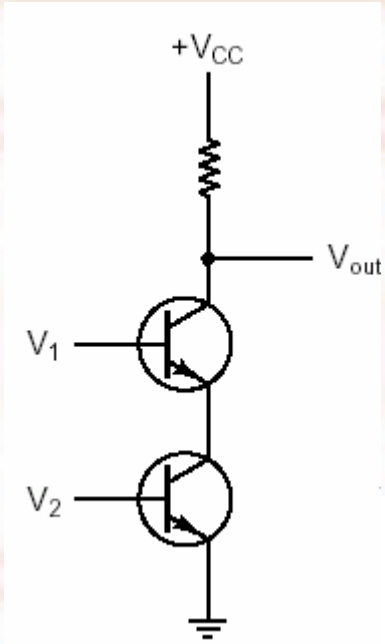
At the input

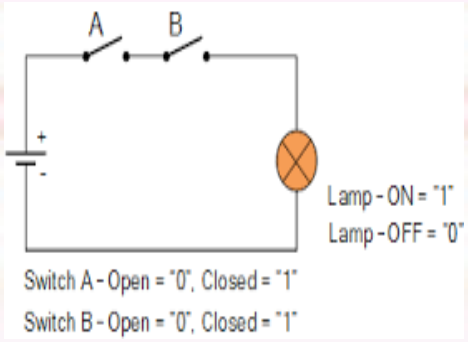


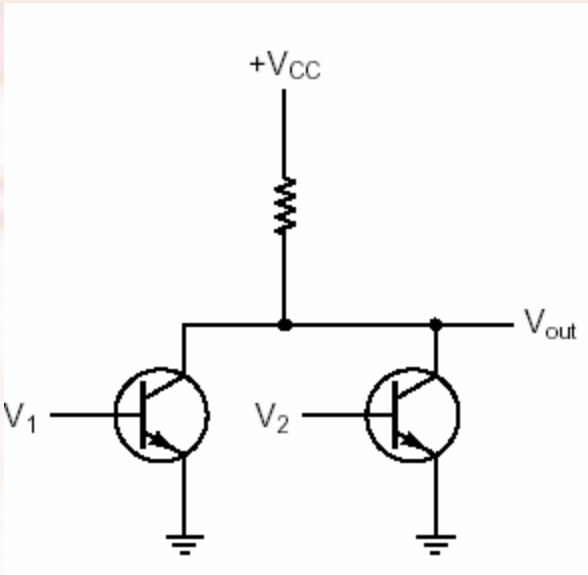


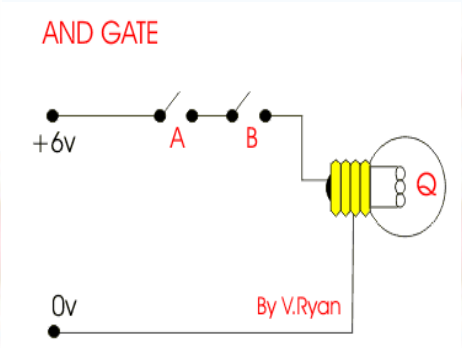

conventional current flow
switch
+
battery
-
light bulb
electron flow

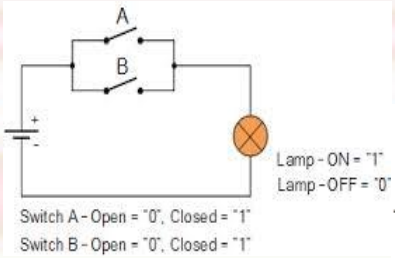
Transistor AND and OR Gates

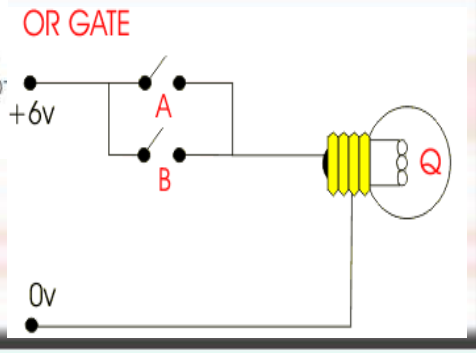






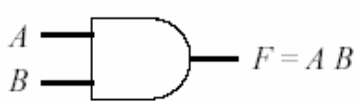






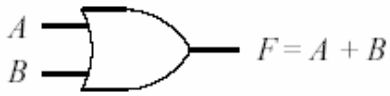
Logical Gates and their Symbols

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



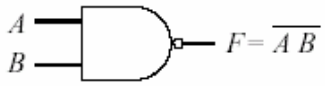
AND

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



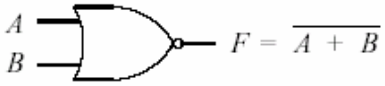
OR

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



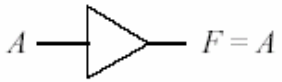
NAND

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



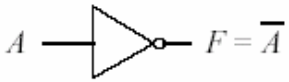
NOR

A	F
0	0
1	1



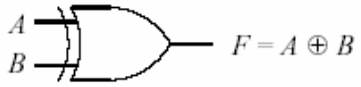
Buffer

A	F
0	1
1	0



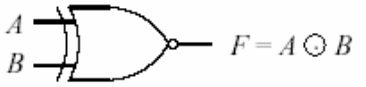
NOT (Inverter)

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



Exclusive-OR (XOR)

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

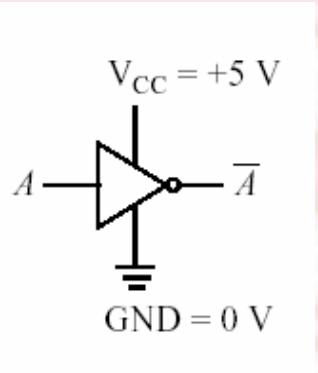


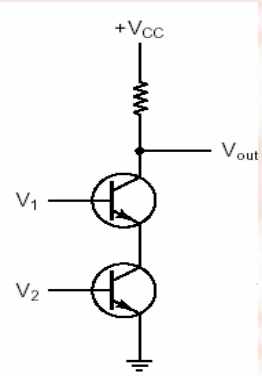
Exclusive-NOR (XNOR)

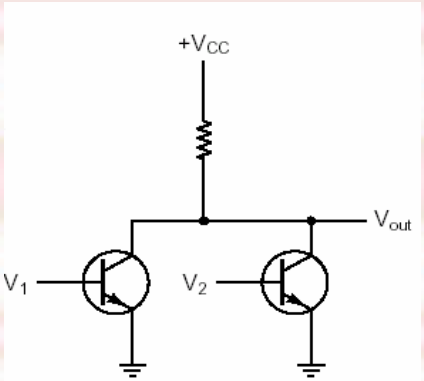
A Truth Table

- Developed in 1854 by George Boole.

Inputs		Outputs							
A	B	$\overline{A}$	$\overline{B}$	AND	NAND	OR	NOR	XOR	XNOR
0	0	1	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0	1	0
1	0	0	1	0	1	1	0	1	0
1	1	0	0	1	0	1	0	0	1



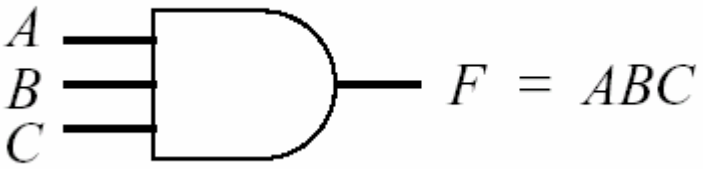


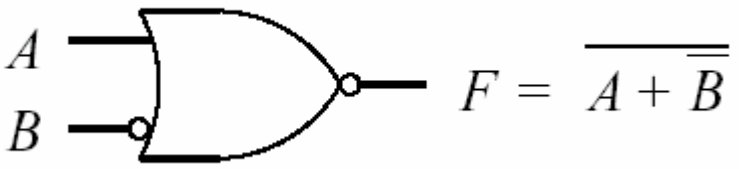


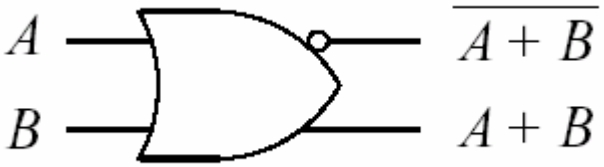
Properties of Boolean Algebra

Name	AND form	OR form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$AA = A$	$A + A = A$
Inverse law	$A\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$AB = BA$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A + B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

Variations of Logical Gates Symbols







DeMorgan's Laws

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: $A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$

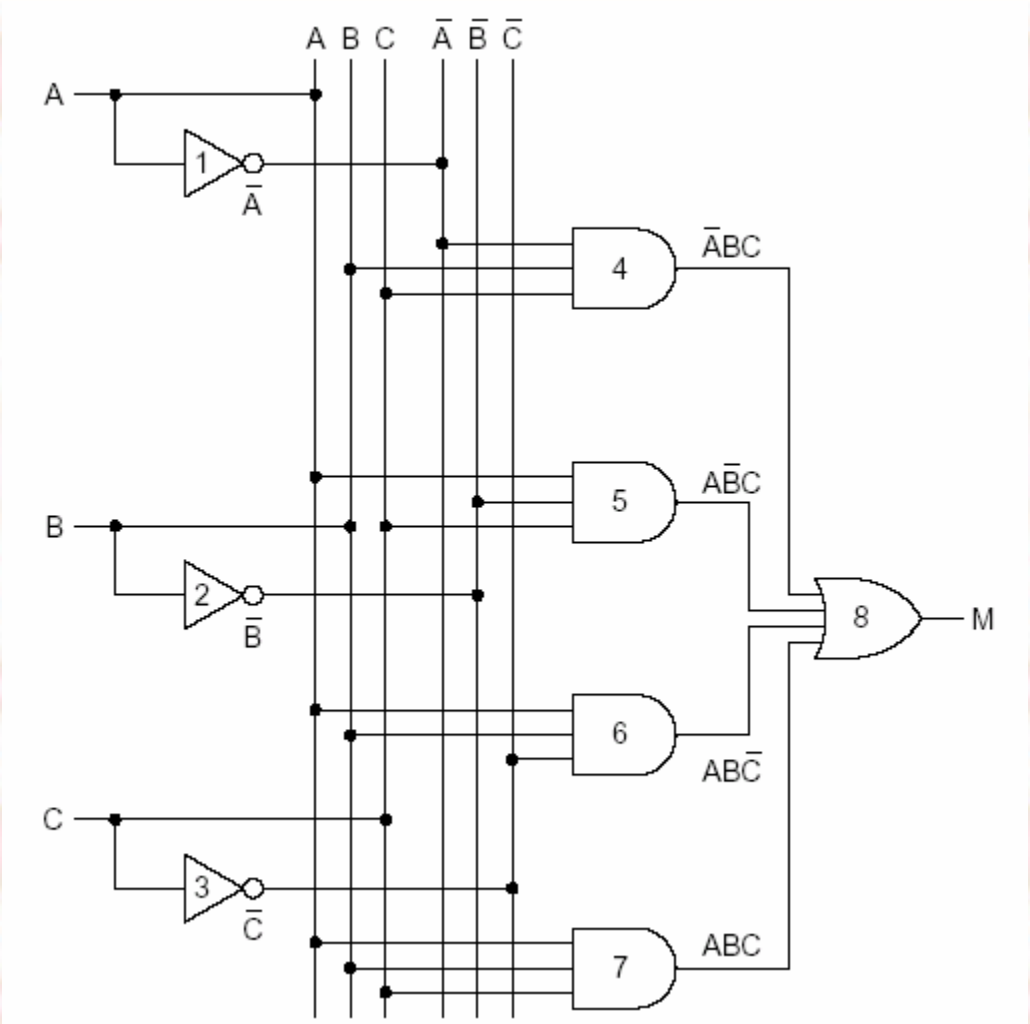
Sum-of-Products Form
The Majority Function

- The sum-of-products (SOP) form for the 3-input majority function:

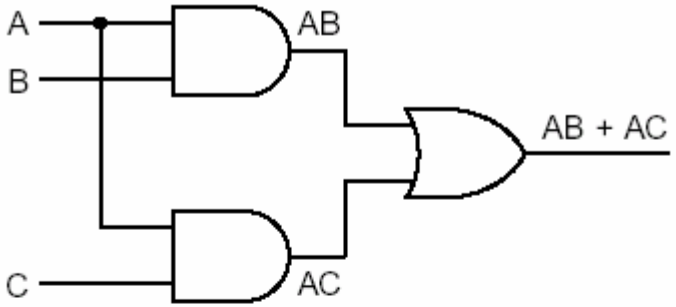
$$M = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC = m3 + m5 + m6 + m7 = \sum(3, 5, 6, 7)$$

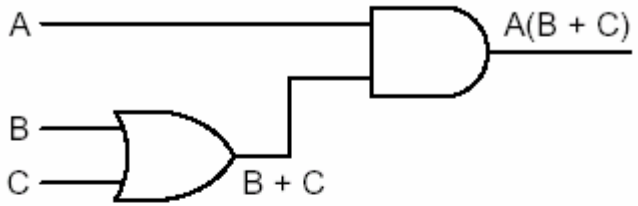
Mindterm index	A	B	C	M
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

A circuit for the majority function



Example of Two Equivalent Circuits





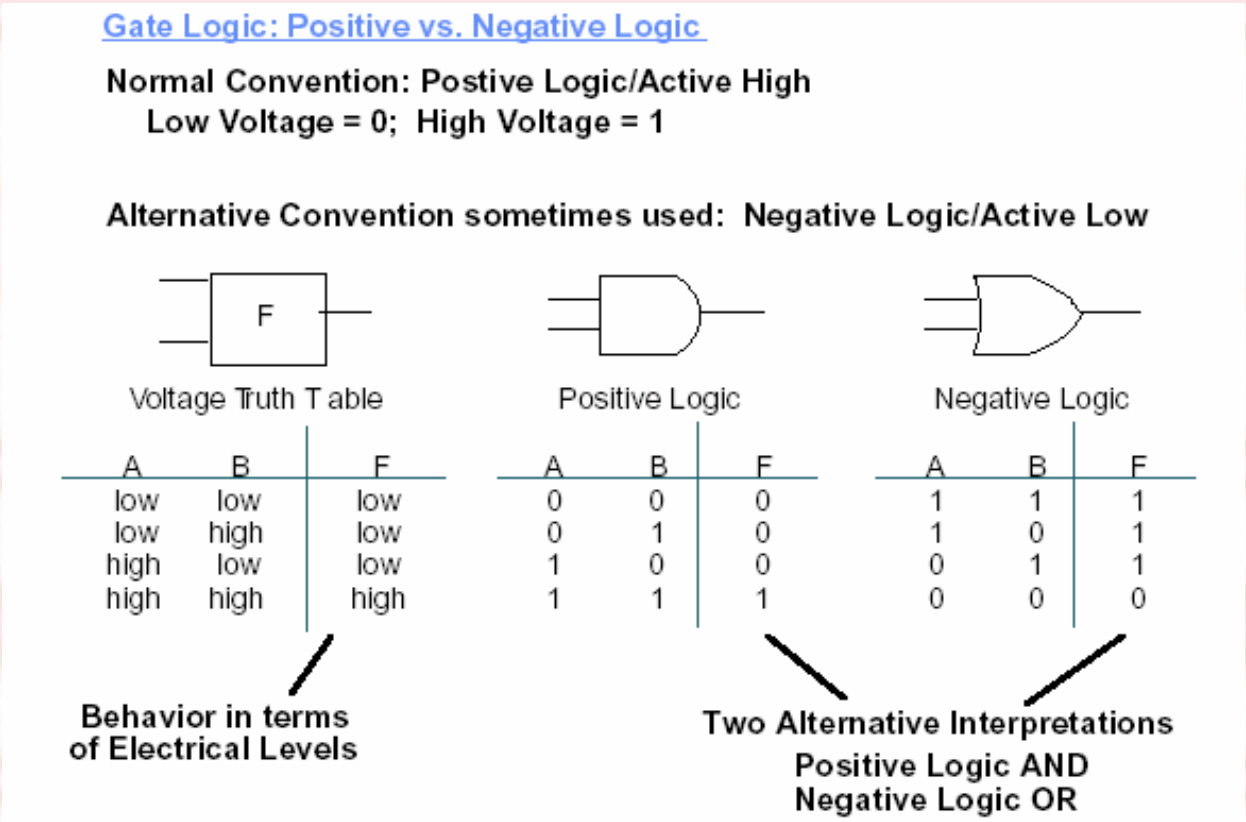
A	B	C	AB	AC	AB + AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

A	B	C	A	B + C	A(B + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Distributive law

Positive vs. Negative Logic

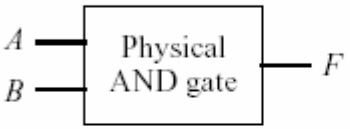
- **Positive logic: 1 - high voltage; 0 - low voltage.**
- **Negative logic: 0 - high voltage; 1 - low voltage.**



Positive vs. Negative Logic (Cont.)

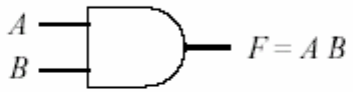
Voltage Levels

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



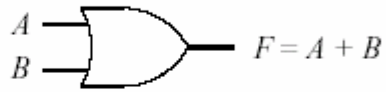
Positive Logic Levels

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



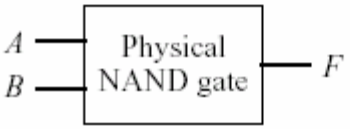
Negative Logic Levels

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



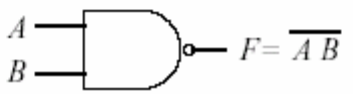
Voltage Levels

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



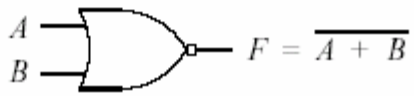
Positive Logic Levels

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



Negative Logic Levels

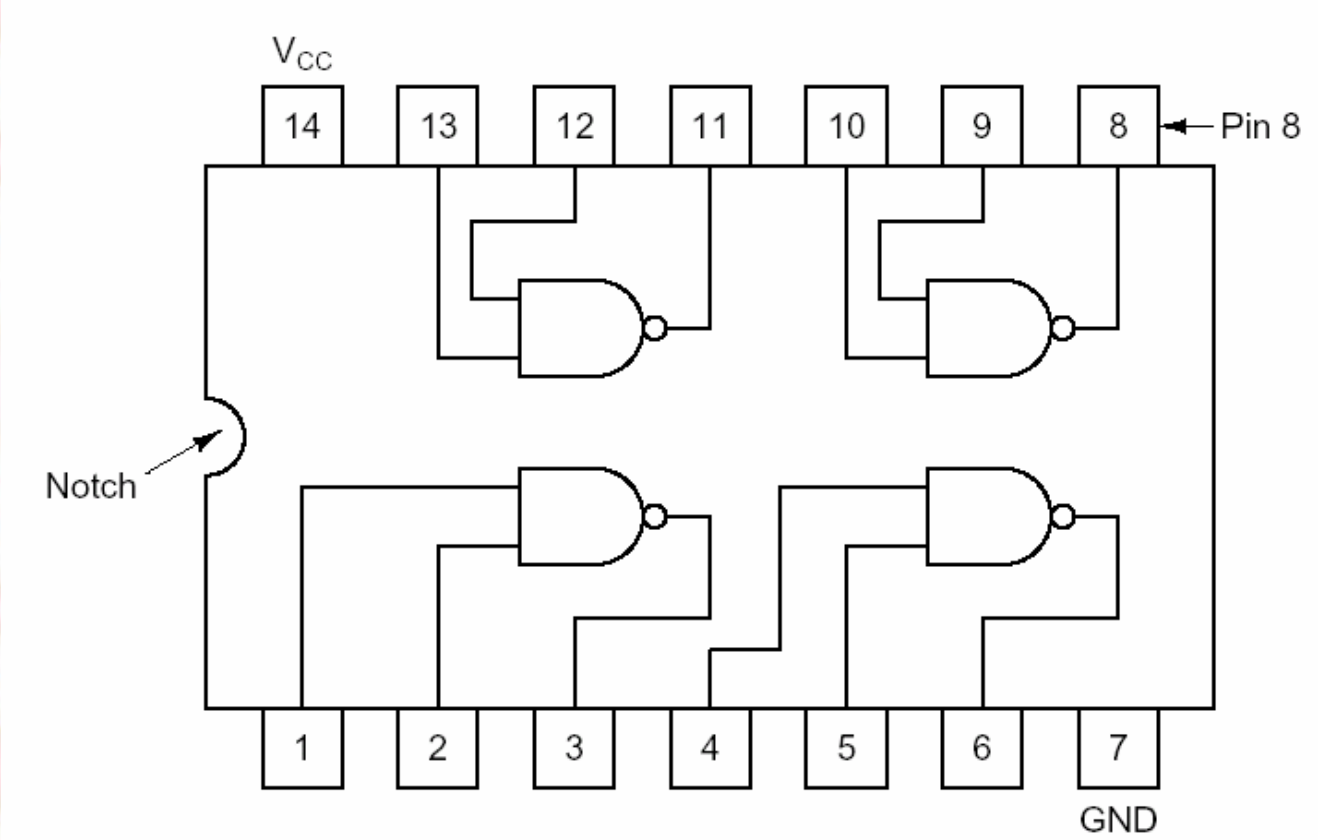
A	B	F
1	1	0
1	0	0
0	1	0
0	0	1



Digital Components

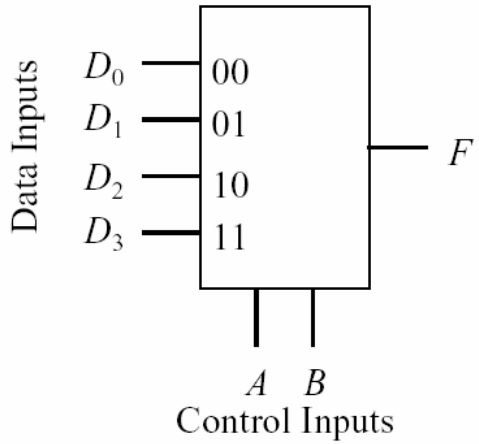
- High-level digital circuit designs are
  - normally created using collections of logic gates referred to as components, rather than using individual logic gates.
- Levels of integration (numbers of gates) in an integrated circuit (IC)
  - can roughly be considered as:
    - Small scale (SSI): 10-100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

SSI Chip
Containing Four Gates



Multiplexer (MUX)

- A **multiplexer** (MUX) is
  - a circuit that connects 2^n data inputs, to a single data output, using *n* control inputs to create that output.



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

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

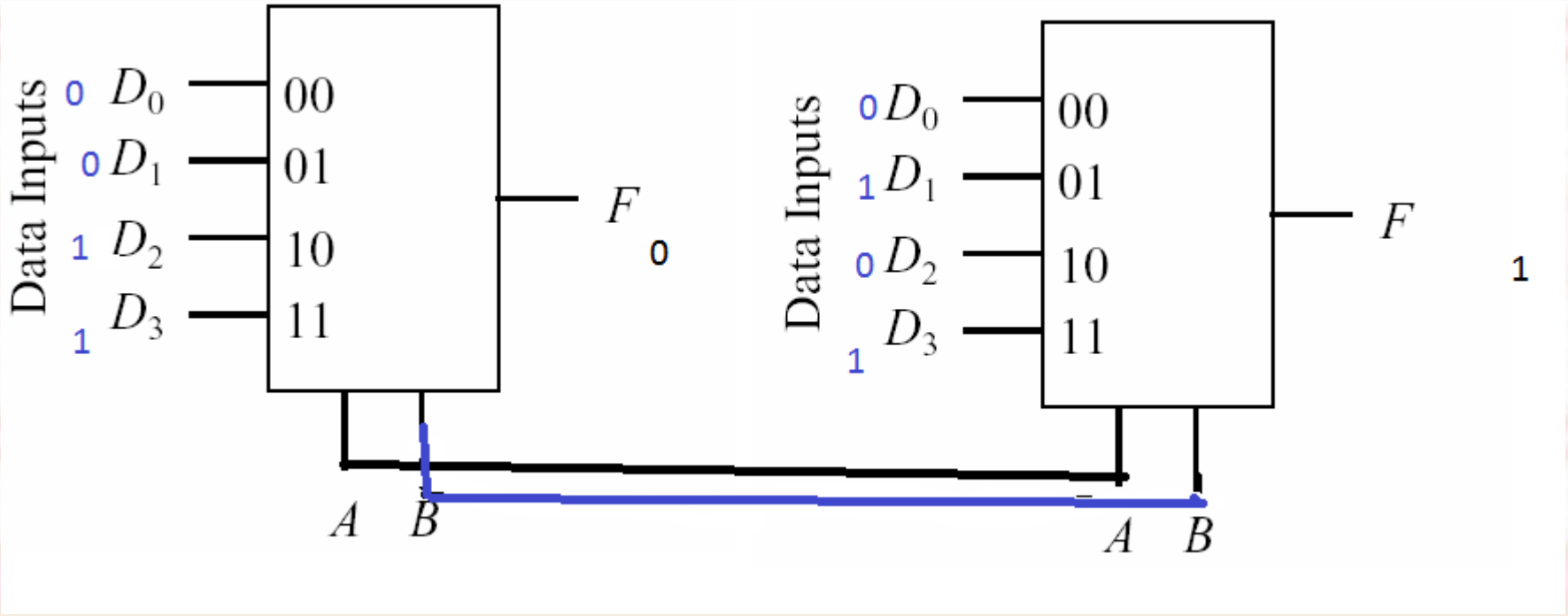
  - Block diagram and truth table for a 4-to-1 MUX

Multiplexer (MUX)

- **Multiplexer** routes its one of 2^n inputs, signal, to a single output to depending on the values of *n* control lines.
- Example
  - Use two 4-to-1 MUXes to implement the following functions:

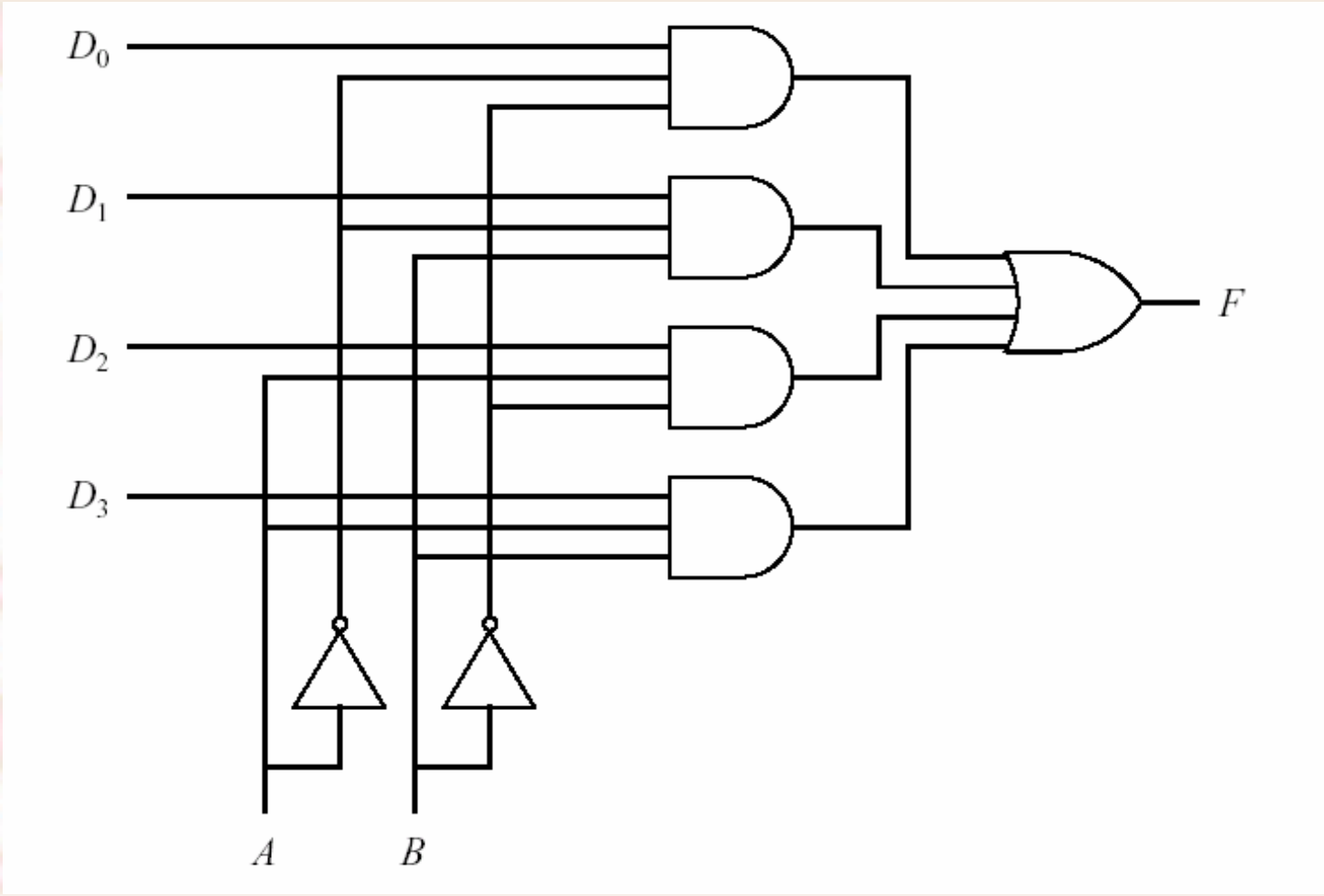
A	B	F_0	F_1
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

MUX Example Solution



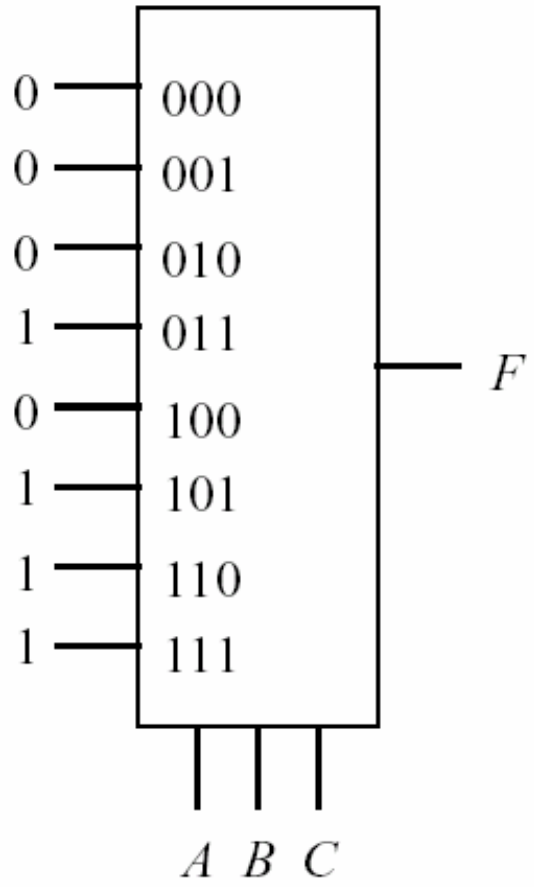
An AND-OR circuit

that implements a 4-to-1 MUX



MUX Implementation of Majority Function

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

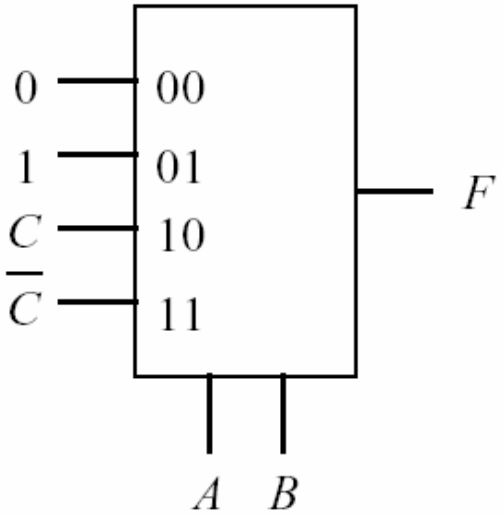


4-to-1 MUX

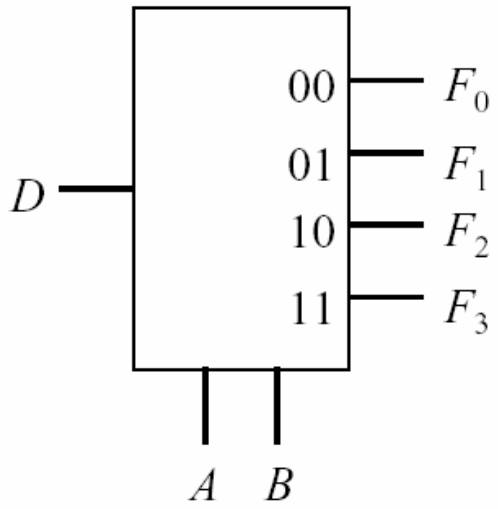
Implements 3-Variable Function

- Generality of multiplexers simplifies the design process, and their modularity simplifies the implementation.

A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	1	1
0	1	1	1	
1	0	0	0	C
1	0	1	1	
1	1	0	1	$\overline{C}$
1	1	1	0	



Demultiplexer (DEMUX)



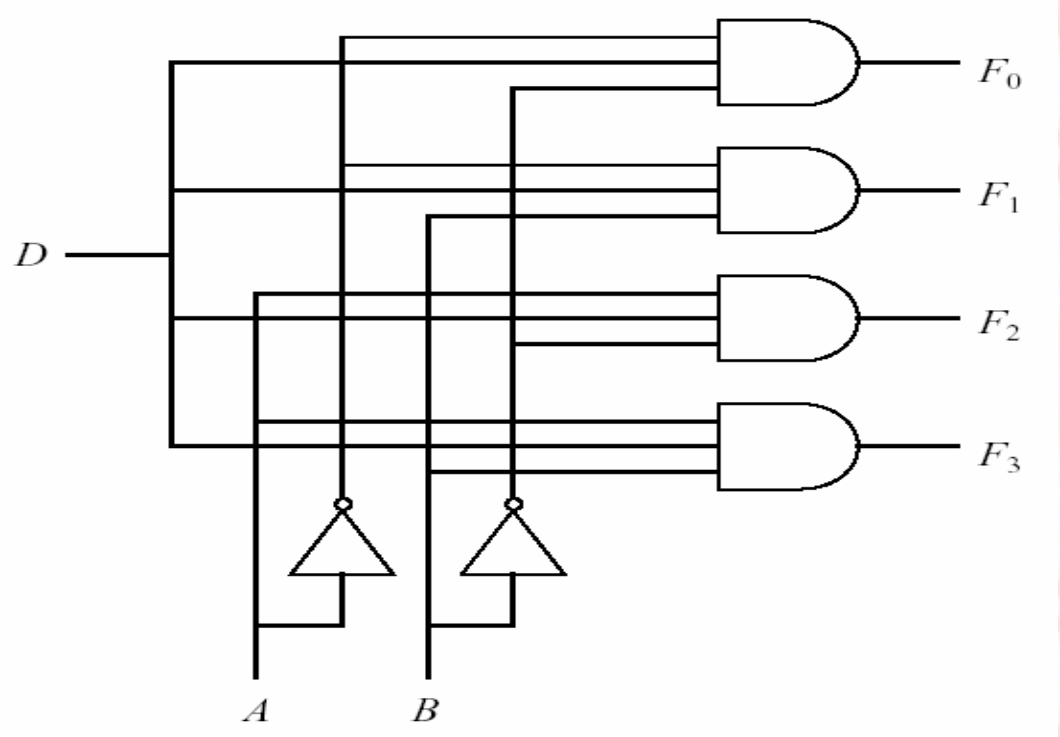
$$F_0 = D\,\overline{A}\,\overline{B} \qquad F_2 = D\,A\,\overline{B}$$

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

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

Demultiplexer is the inverse multiplexer, which routes its single input signal to one of 2^n outputs, depending on the values of *n* control lines.

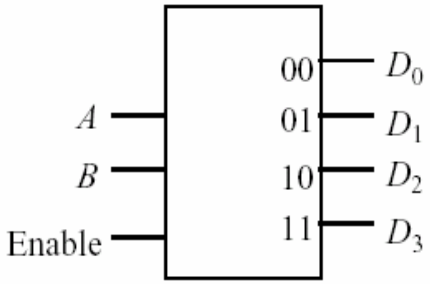
Gate Level Implementation of DEMUX



- DEMUX-es are used to send data from a single source to one or number of destinations.

Decoder

- is a circuit that takes an n-bit number input to select one of the 2^n output lines.

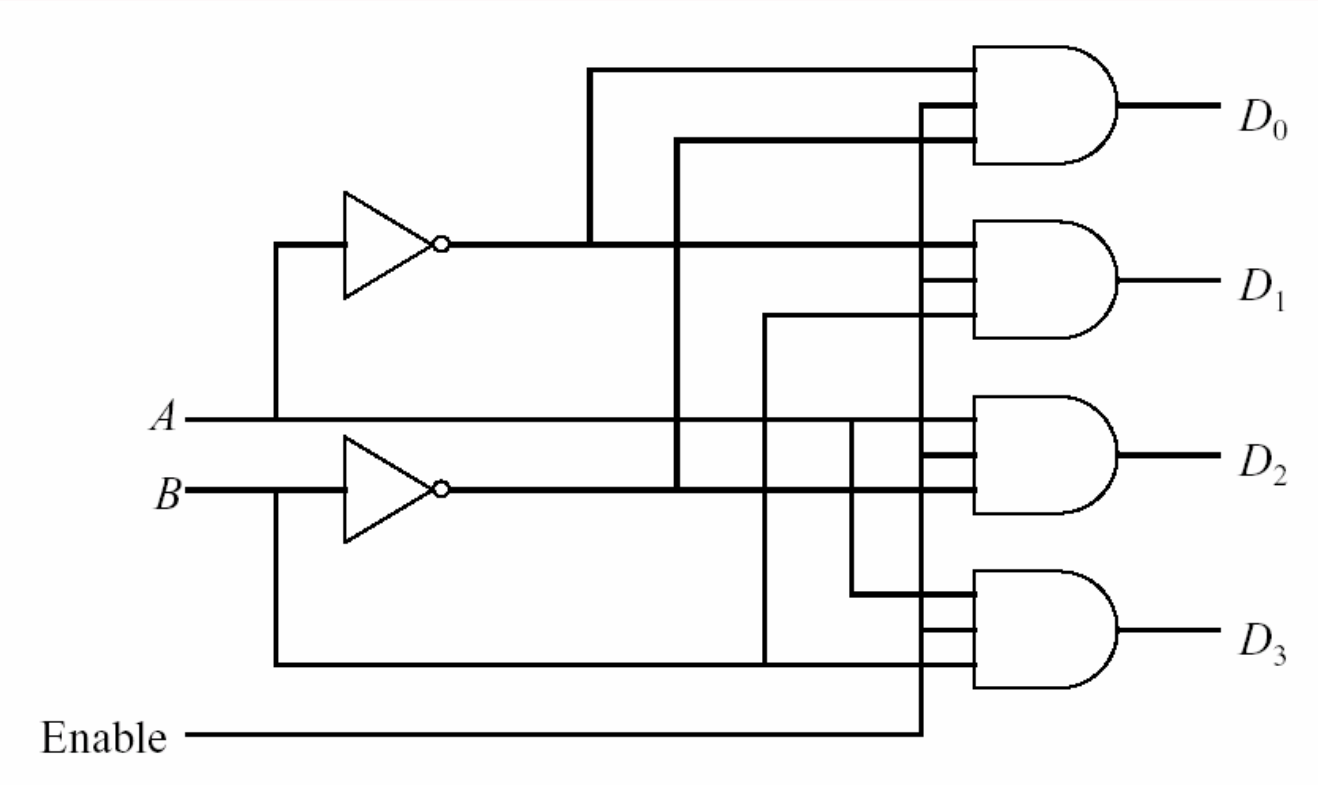


Enable = 1						
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

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

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

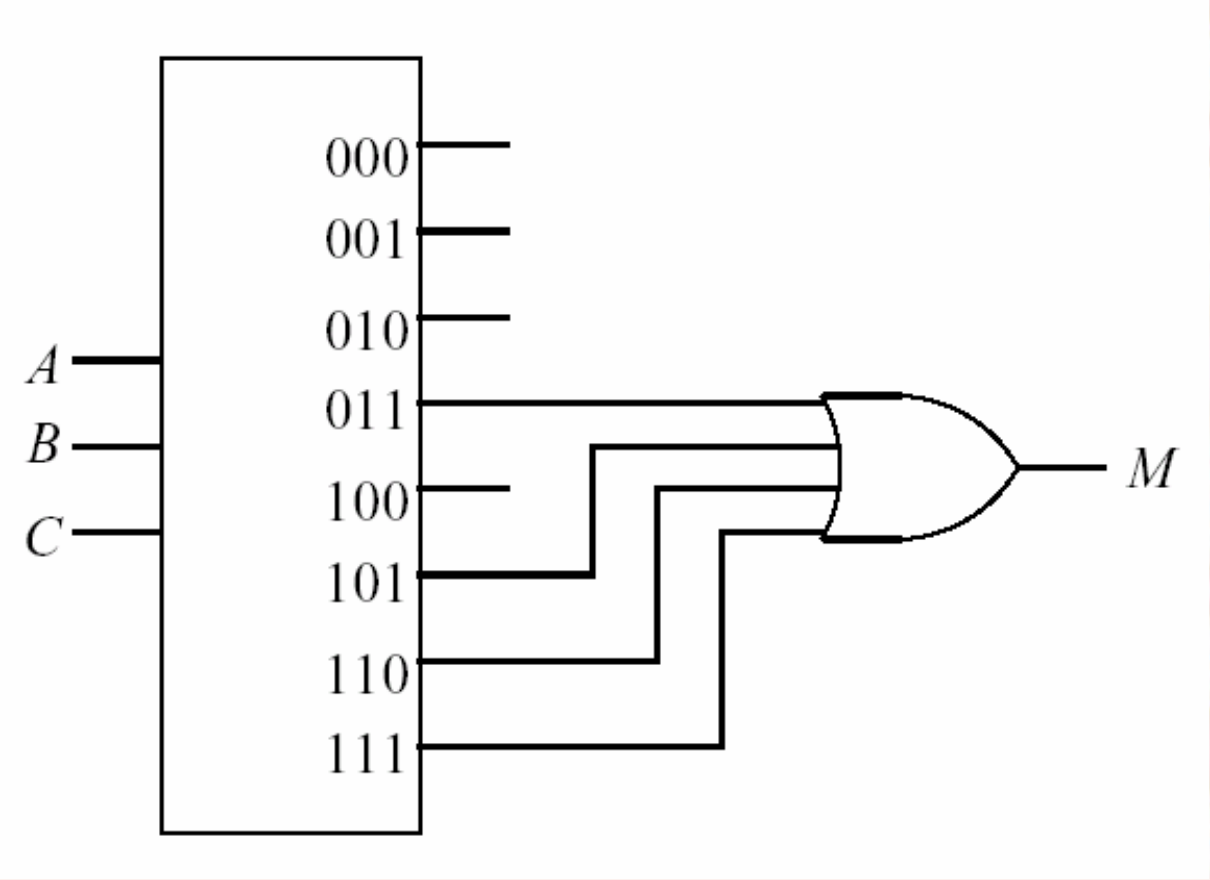
Block diagram and truth table for a 2-to-4 decoder



Gate-level implementation of a 2-to-4 decoder

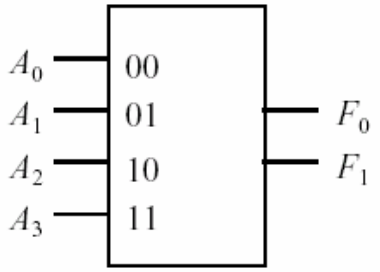
- **Decoders are used in translating memory addresses into physical locations, or**
- **implement Boolean functions.**

A 3-to-8 Decoder
Implementation of Majority Function



Priority Encoder

- Can be thought of as the converse of a decoder.
- imposes an order on the inputs.
- A_i has a higher priority than A_{i+1}

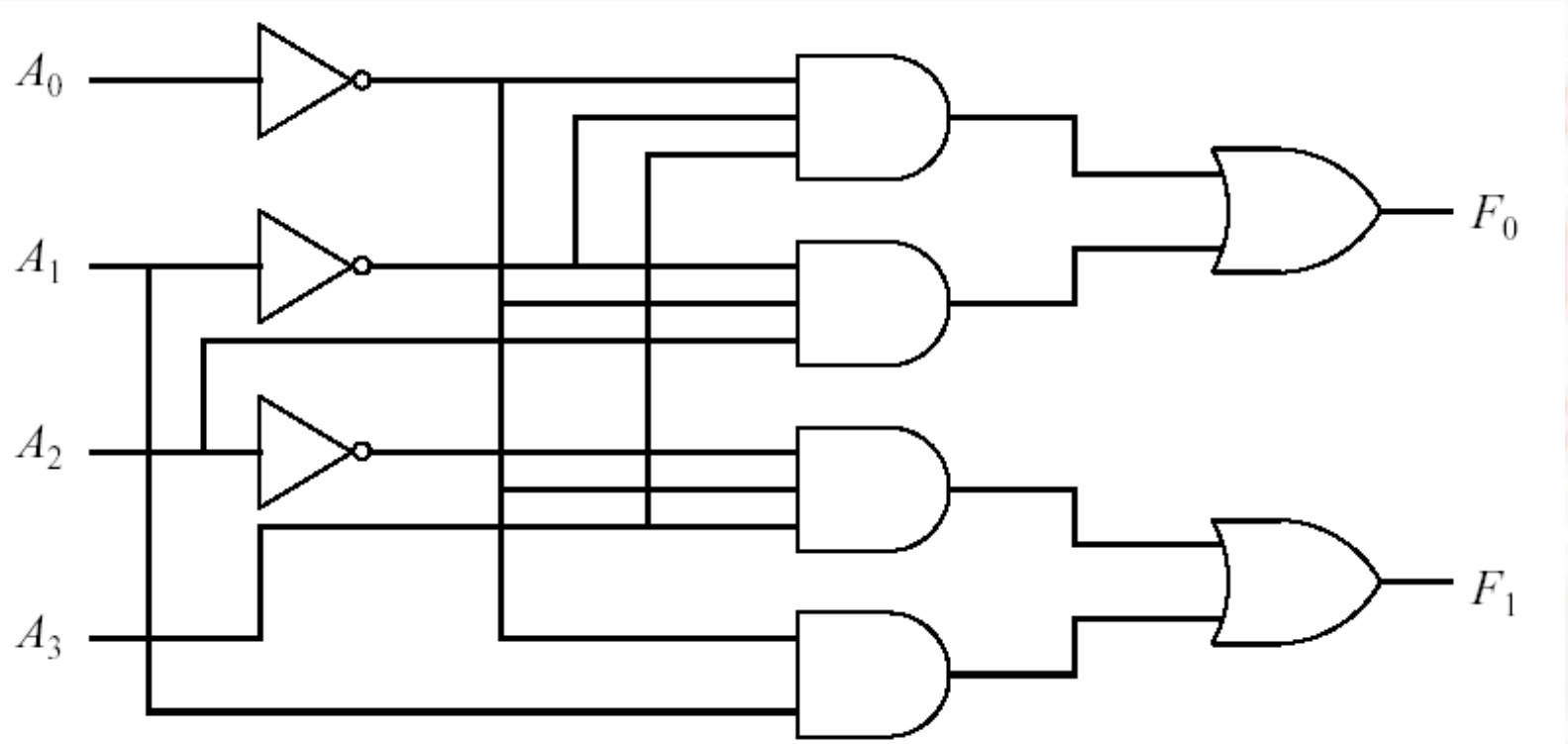


$$F_0 = \overline{A}_0 \overline{A}_1 A_3 + \overline{A}_0 A_1 A_2$$

$$F_1 = \overline{A}_0 \overline{A}_2 A_3 + \overline{A}_0 A_1$$

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

- Block diagram and truth table for a 4-to-2 priority encoder

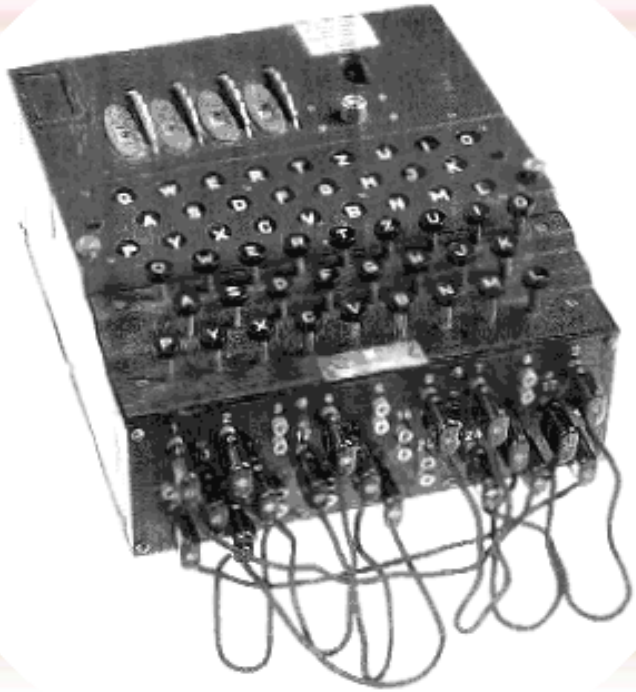


- Logic diagram for a 4-to-2 priority encoder

- **Priority encoders**
  - **are used for arbitrating among a number of devices that compete for the same resource, as when a number of users simultaneously attempt to log on to a computer system.**

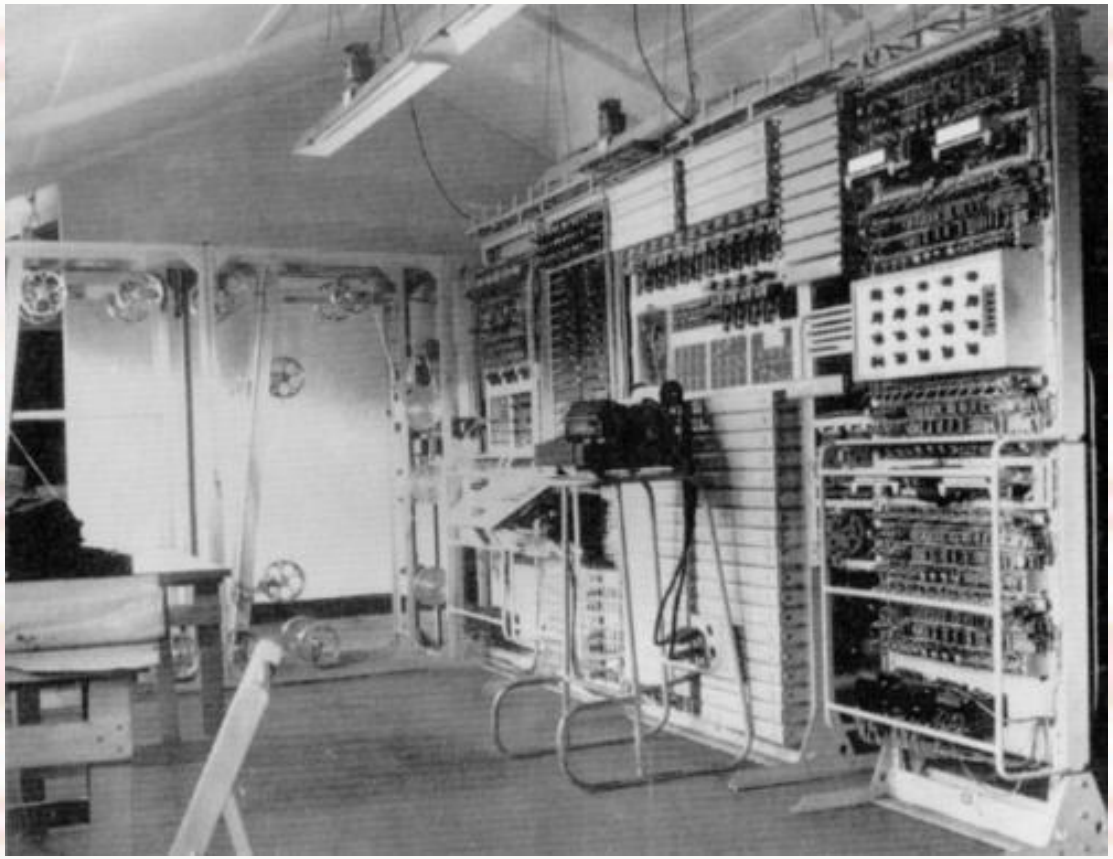
1939-1945

During WWII, Germany used the **Enigma** machine to encode its transmissions.



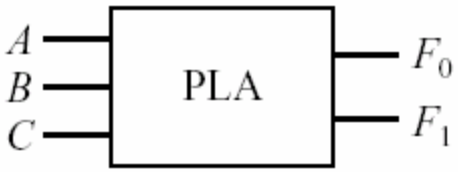
1943

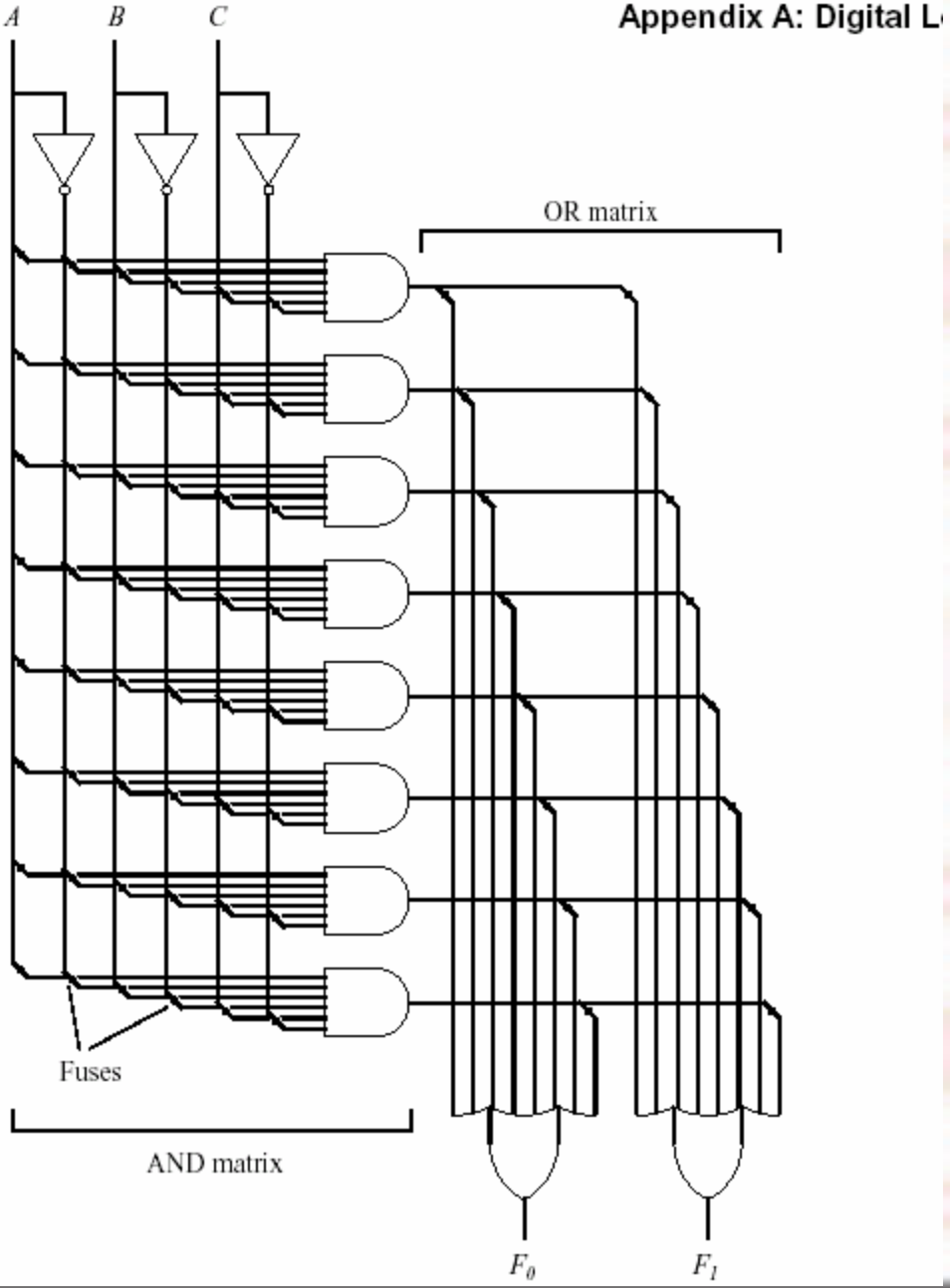
A team of the British Code and Cipher School builds a machine to decode the messages. It was called **Colossus**.



Programmable Logic Array

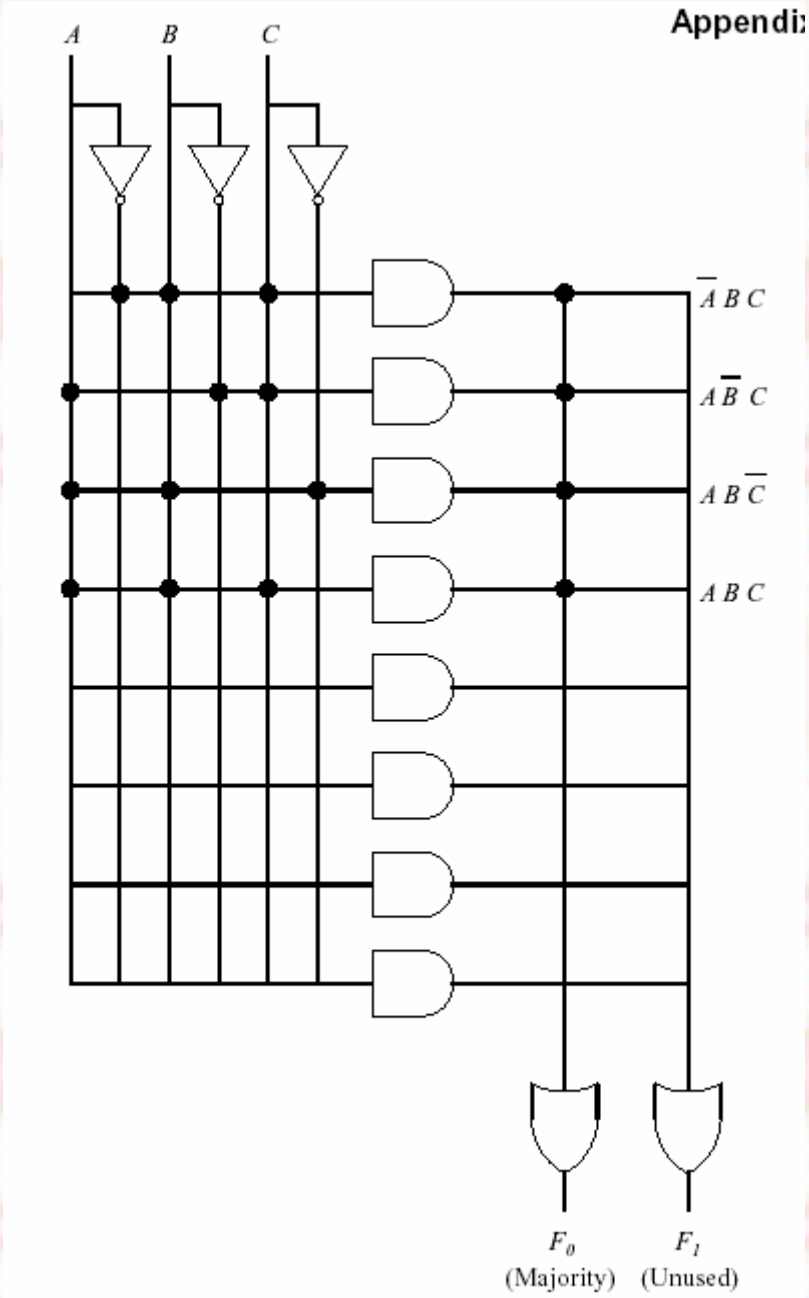
- A PLA is a customizable AND matrix followed by a customizable OR matrix.
- Black box view of PLA:





Simplified Representation of PLA Implementation of Majority Function

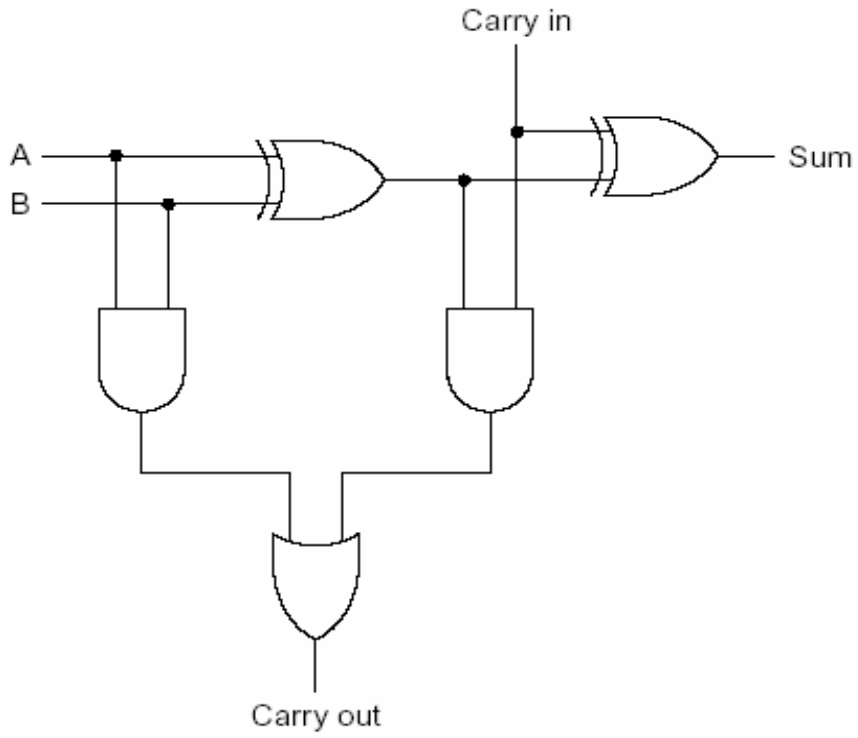
- A programmable fuse is placed at each cross point in the AND and OR matrices.
- The matrices are customized for specific functions by disabling fuses.
- If fuse is disabled at an input to an AND gate, the gate behaves as if input is tied to a 1.
- A disabled input to an OR gate in a PLA behaves as if the input is tied to a 0.


A
B
C
$\overline{A}\,B\,C$
$A\,\overline{B}\,C$
$A\,B\,\overline{C}$
$A\,B\,C$
F_0
(Majority)
F_1
(Unused)

Full Adder

$$C_{i+1} = A_i \cdot B_i + A_i \cdot C_i + B_i \cdot C_i$$
$$S_i = A_i \oplus B_i \oplus C_i$$

A	B	Carry in	Sum	Carry out
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



Truth table for a full adder

A circuit for a full adder

Ripple - Cary 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

Carry Out Sum

Example:

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

$C_{i+1} = A_i \cdot B_i + A_i \cdot C_i + B_i \cdot C_i$

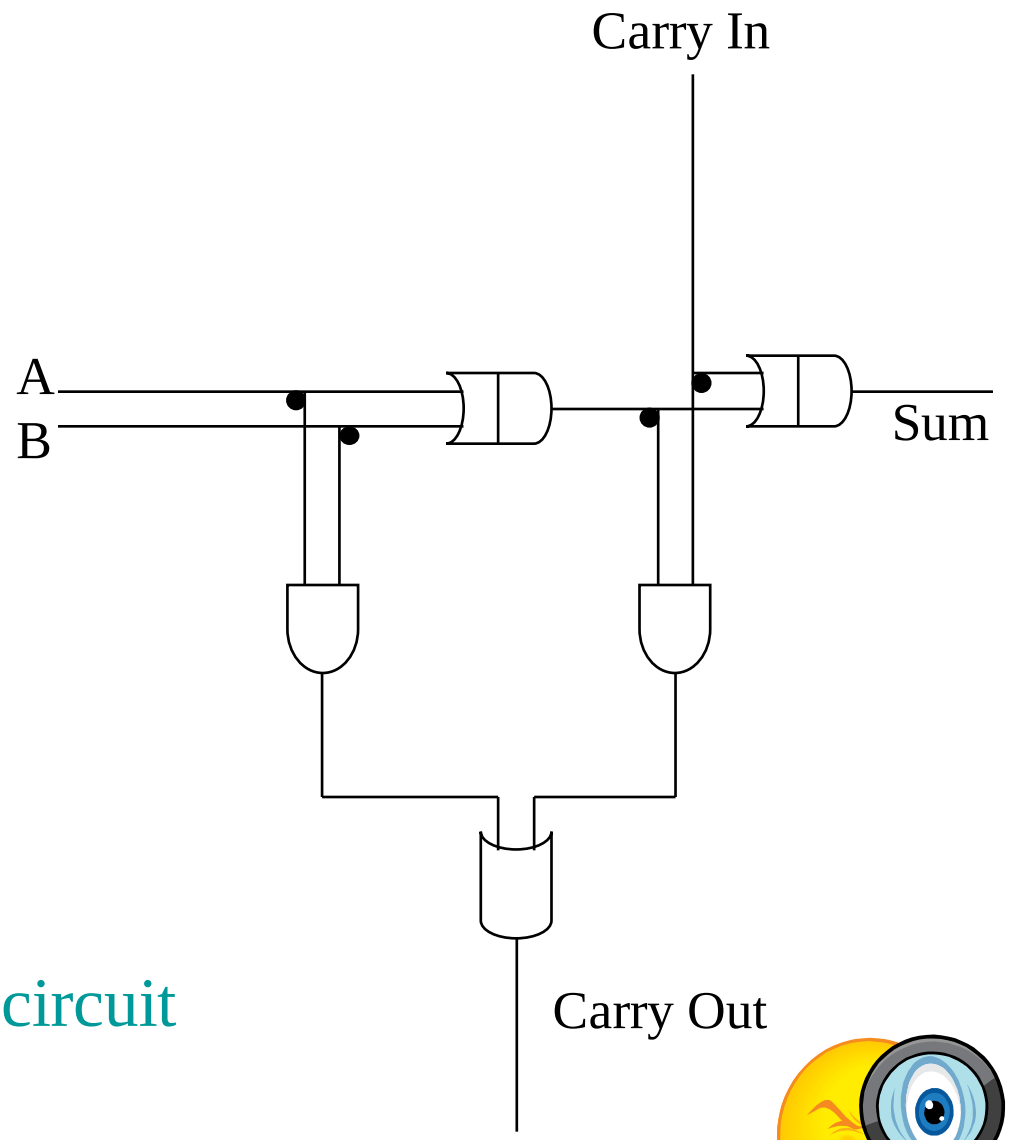
$S_i = A_i \oplus B_i \oplus C_i$

Example of addition for two unsigned binary numbers

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

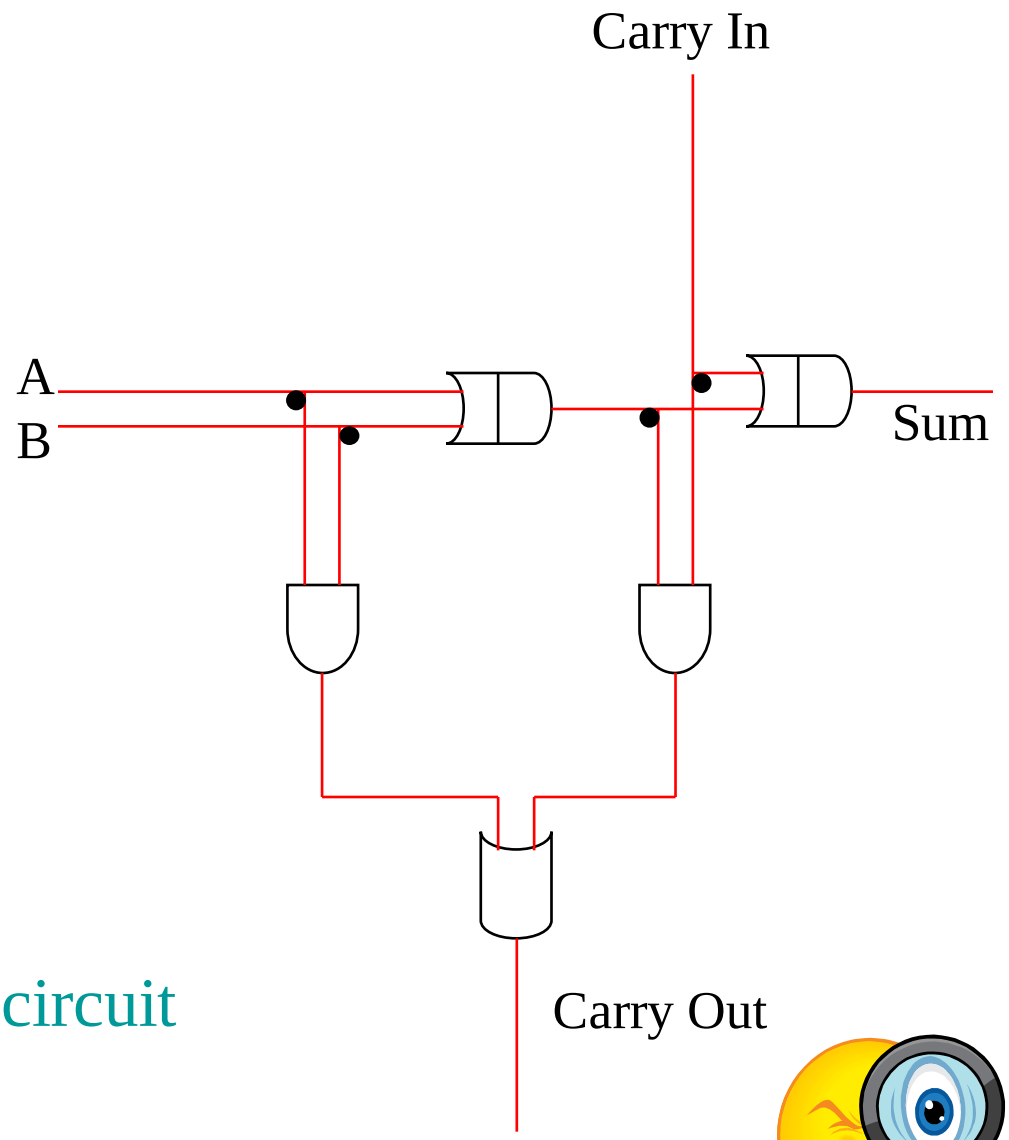


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

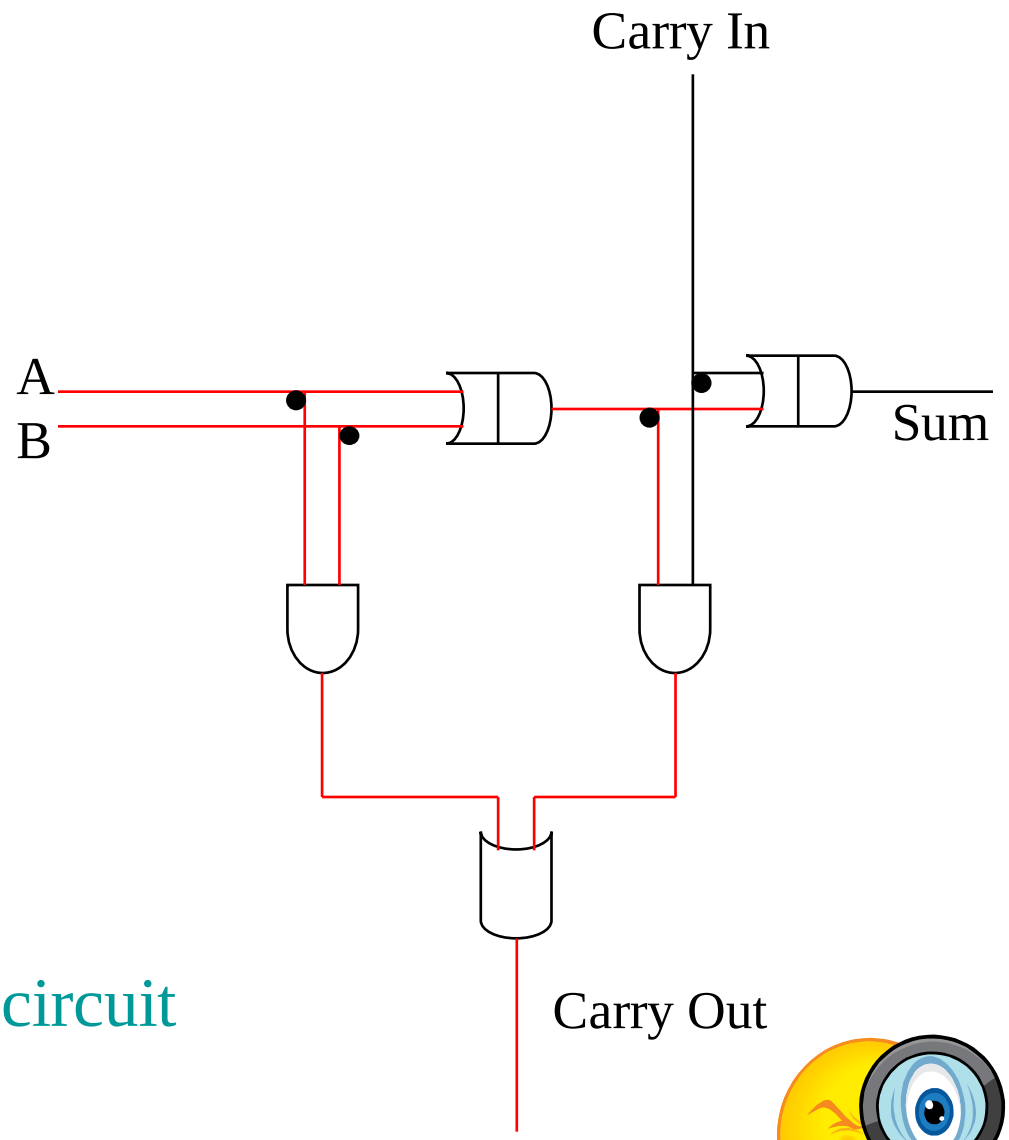


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

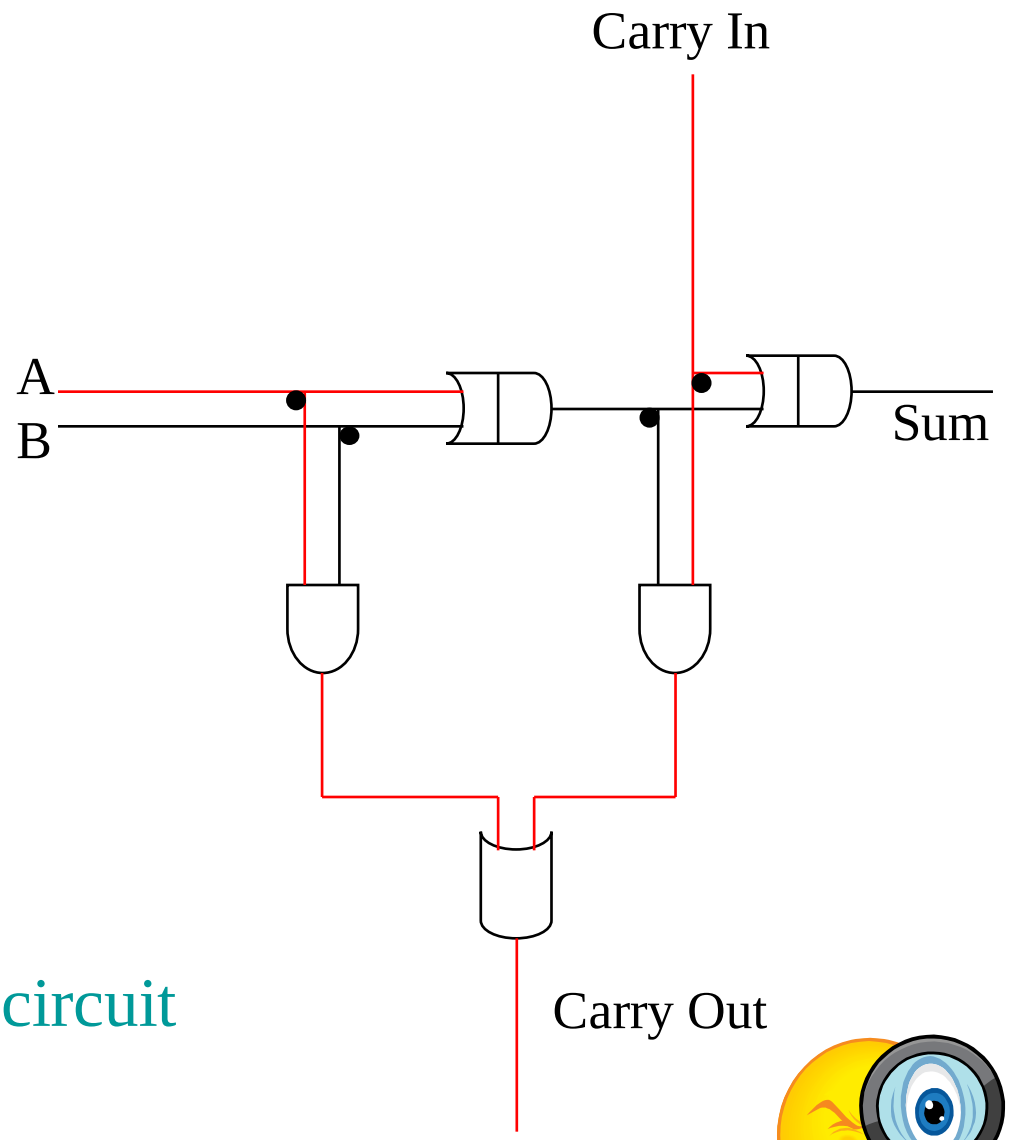


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

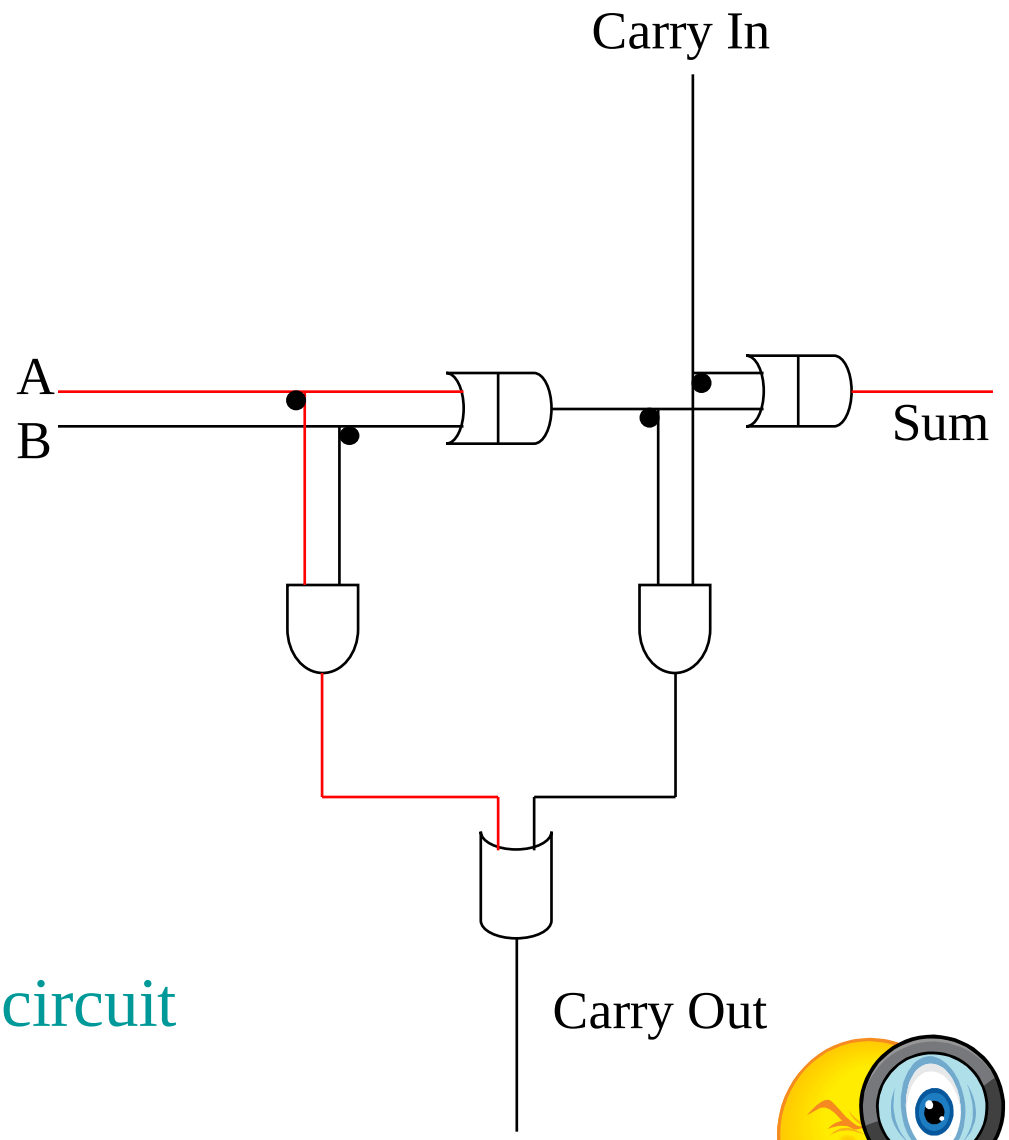


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

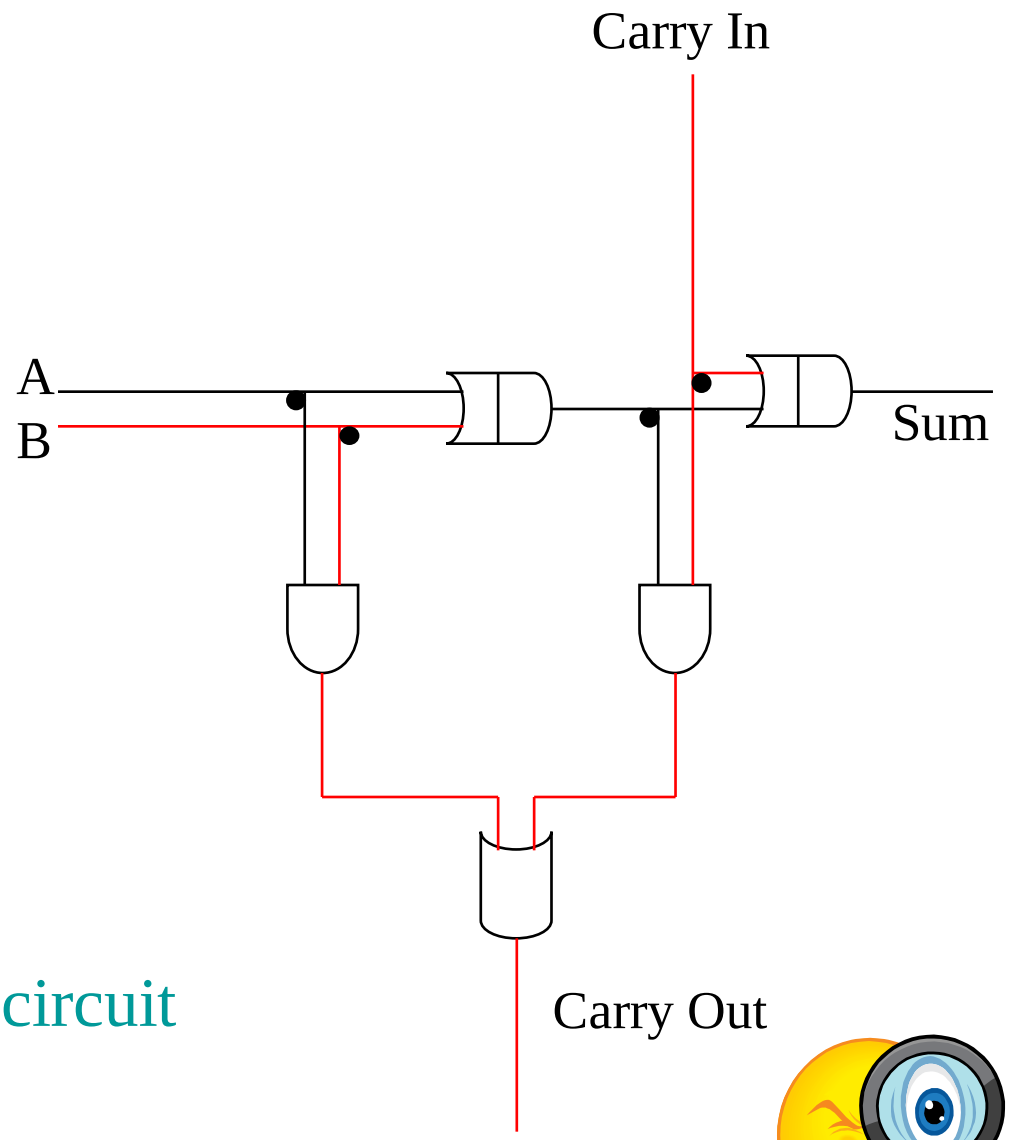


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

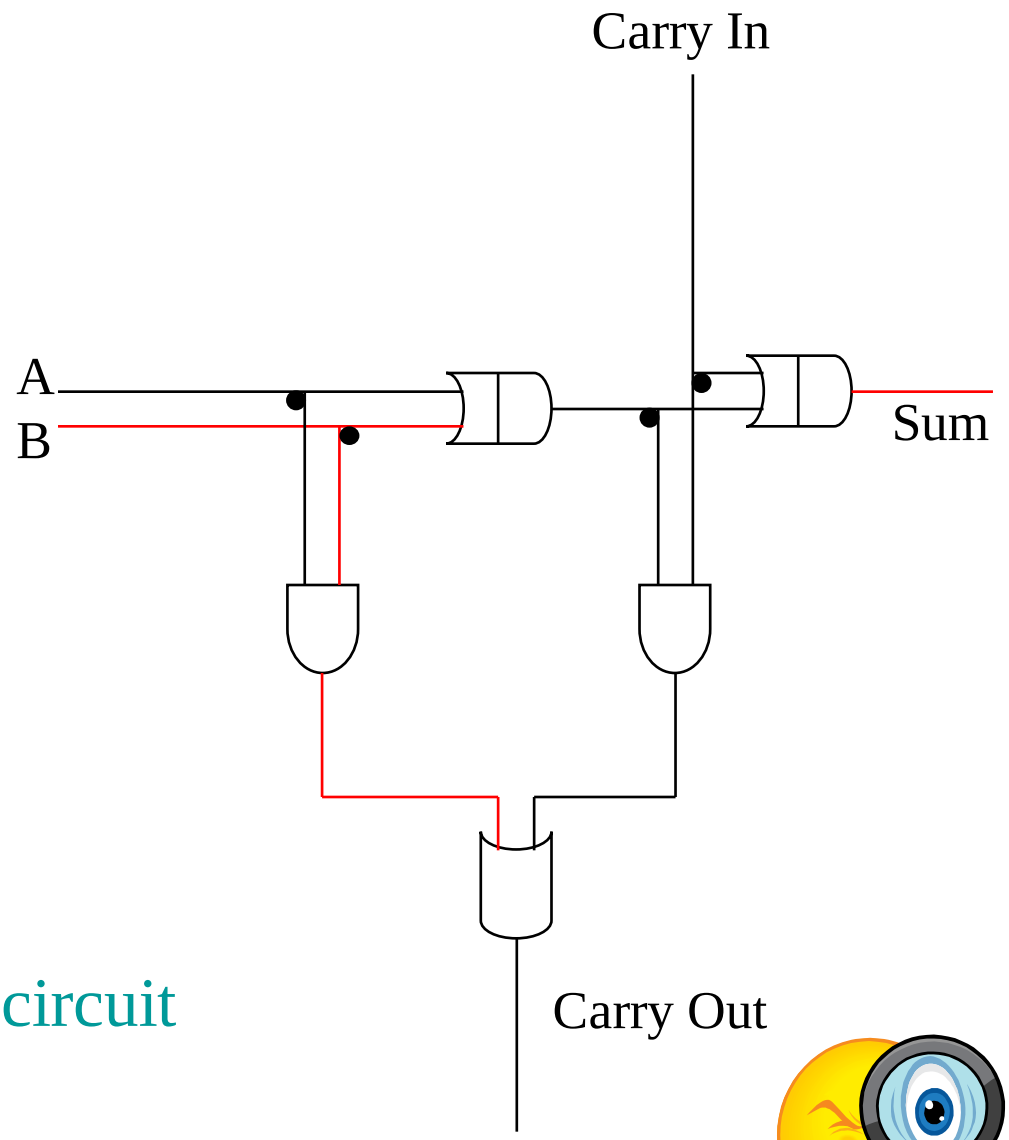


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

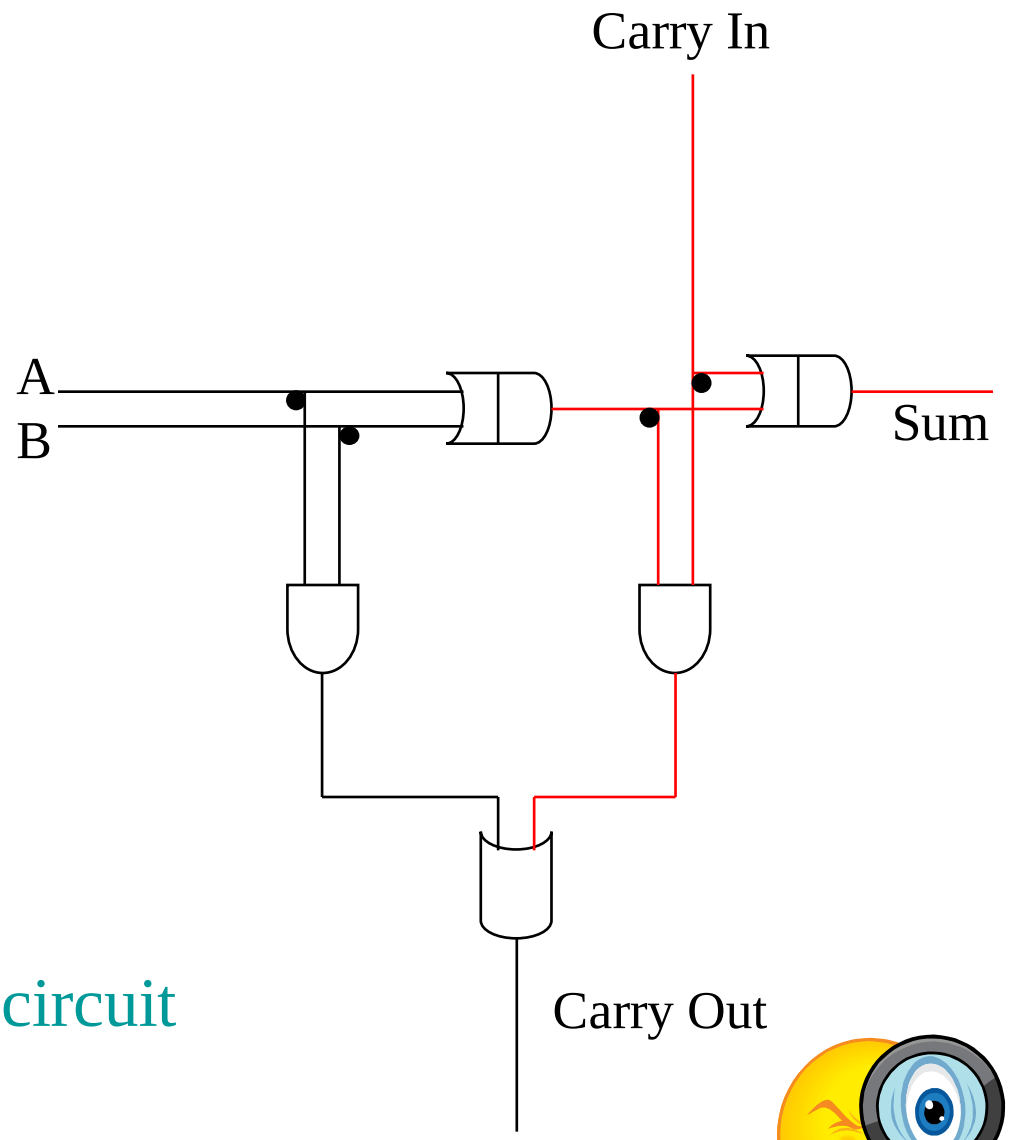


You should confirm that this circuit gives the required results

Full Adder

Truth Table

A	B	Carry In	Carry Out	First Digit(sum)
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

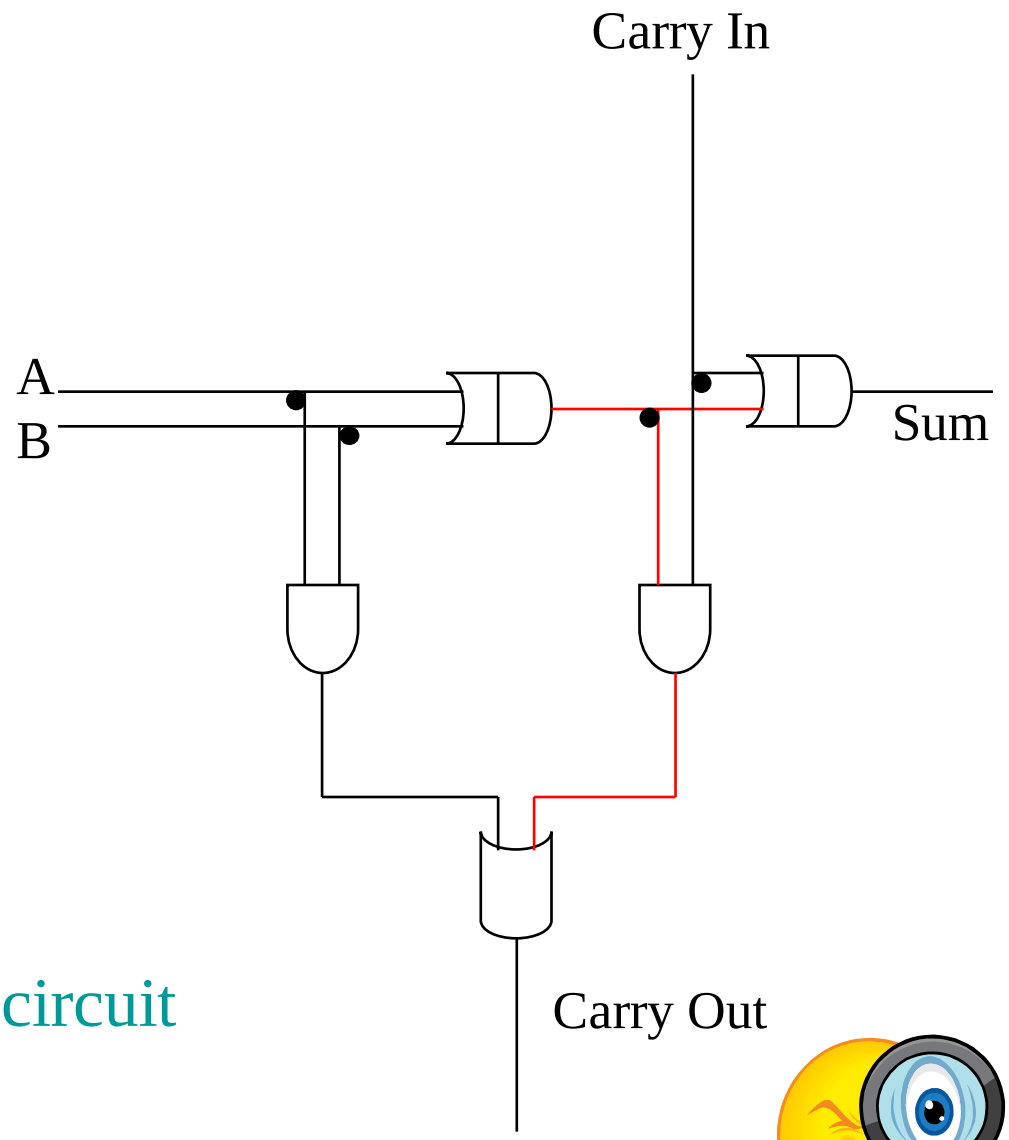


You should confirm that this circuit gives the required results

Full Adder

Truth Table

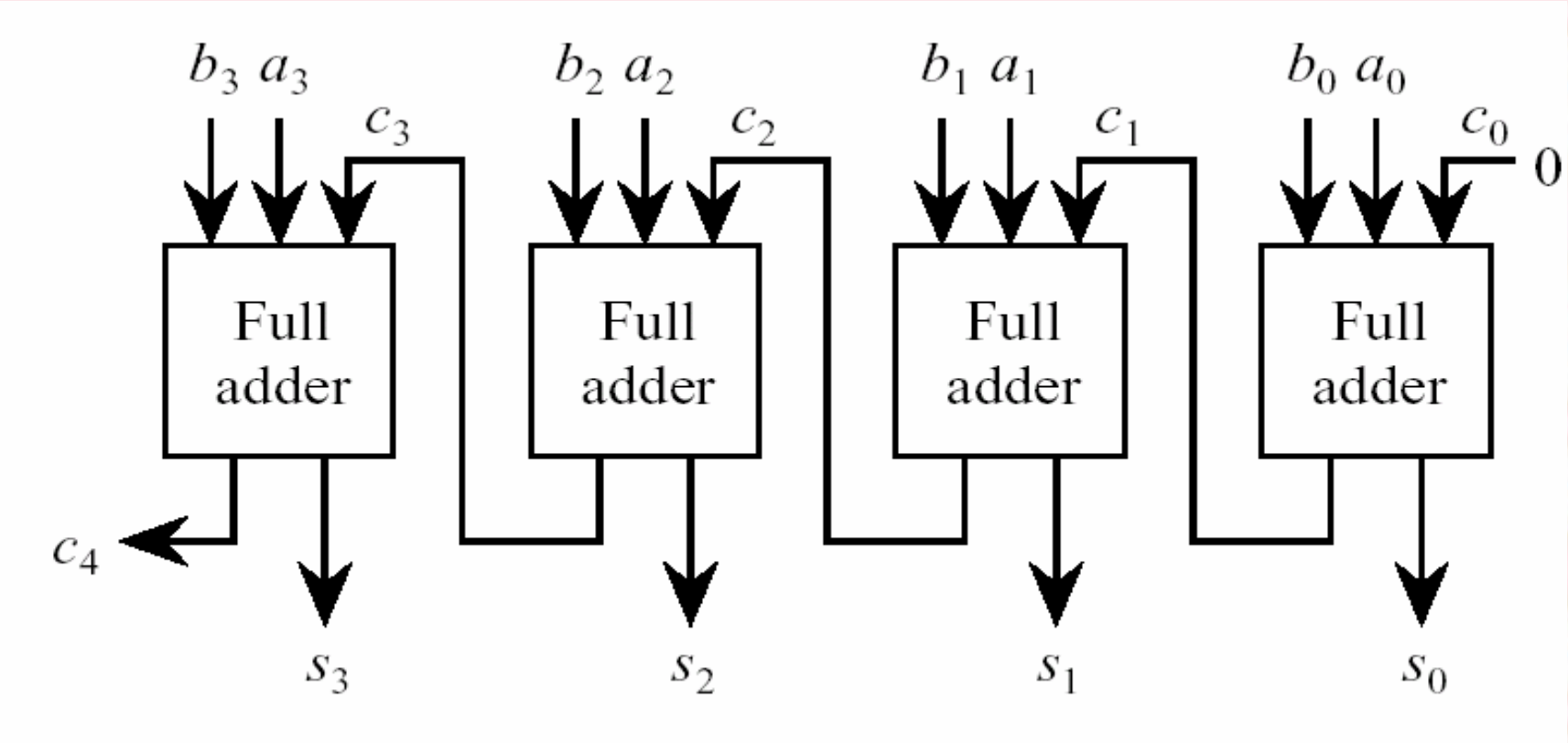
A	B	Carry In	Carry Out	First Digit(sum)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



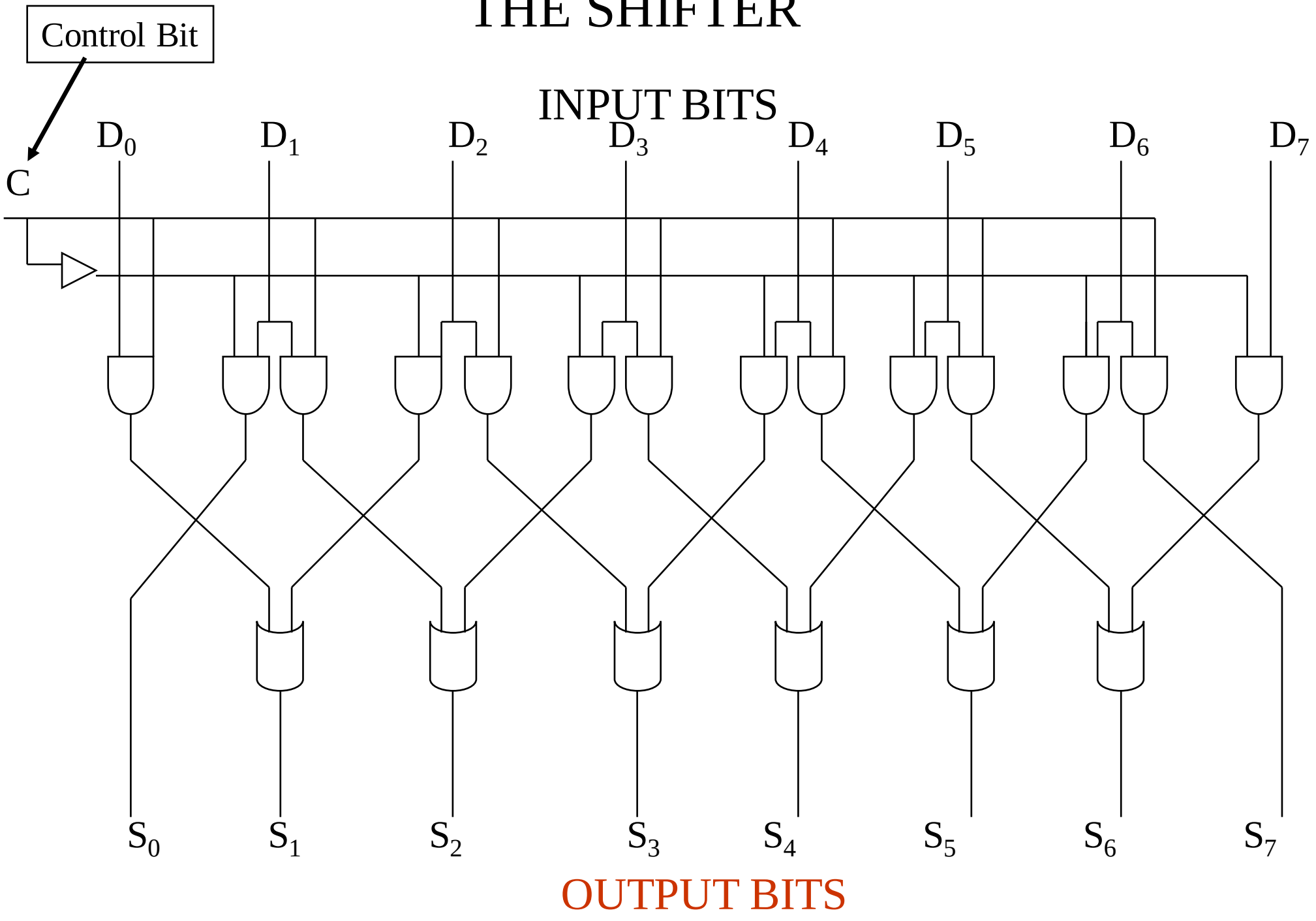
You should confirm that this circuit gives the required results

Four-Bit Ripple-Carry Adder

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

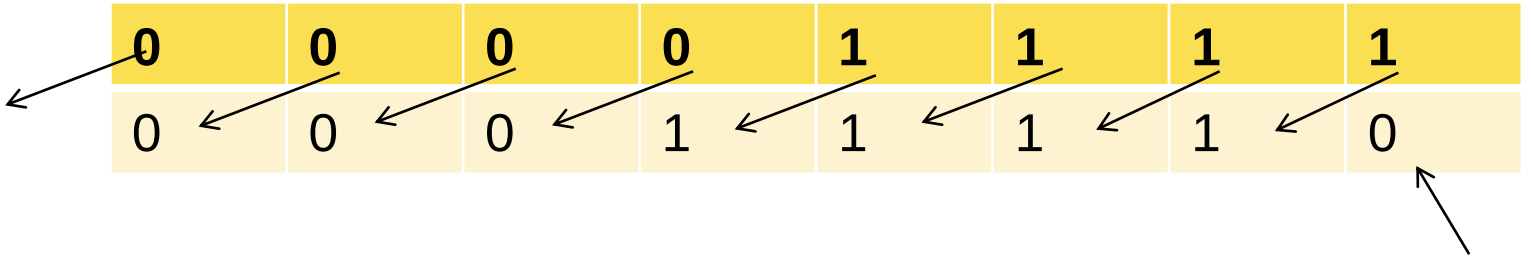


THE SHIFTER


Control Bit
INPUT BITS
C
D0
D1
D2
D3
D4
D5
D6
D7
S0
S1
S2
S3
S4
S5
S6
S7
OUTPUT BITS

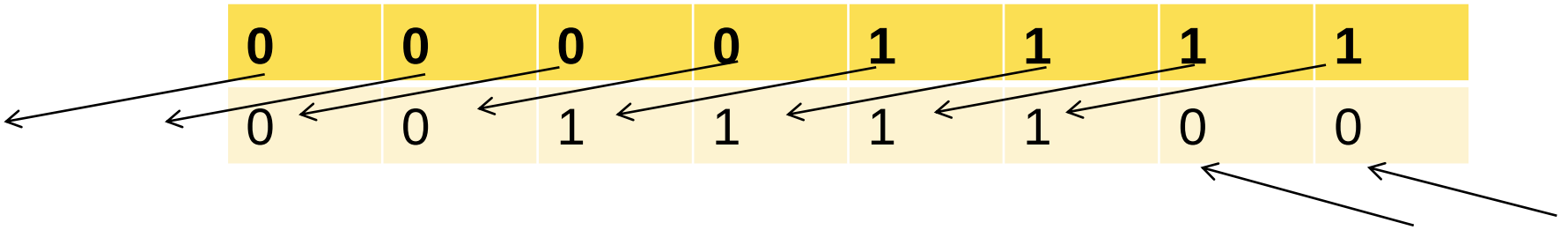
THE SHIFTER Example

Shift 1 bit
to the left



15*2=30

Shift 2 bits
to the left

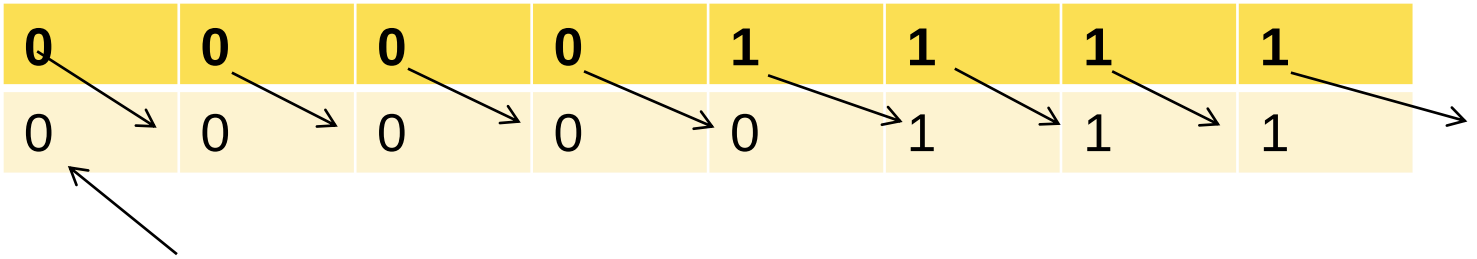


15*4=60

Add two 0s here

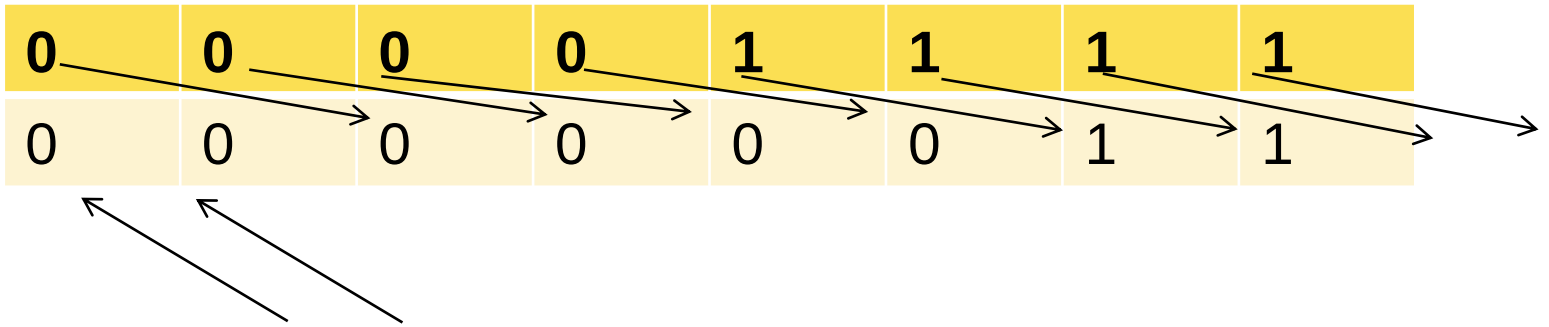
THE SHIFTER Example

Shift 1 bit
to the right



15/2=7

Shift 2 bits
to the right



15/4=3

Shift left example

- 146*64
  - 00000000 1001 0010 * 0100 0000 =
  - 00100100 10000000 (9344)
- Now let shift 146 left 6 bits:
  - 00000000 1001 0010 shift left 6 bits =
  - 00100100 10000000 (9344)
- You can see using shift to multiply the power of 2’s is much faster

Shift right example

- 146/64
  - 00000000 1001 0010 / 0100 0000 =
  - 0000000 00000010(2)
- Now let shift 146 right 6 bits:
  - 00000000 1001 0010 shift right 6 bits =
  - 00000000 00000010 (2)
- You can see using shift to divide the power of 2's is much faster

Shifter (An example)

We will now use the above ‘shifter circuits’ with an example:

In the following example: The red lines are: 0

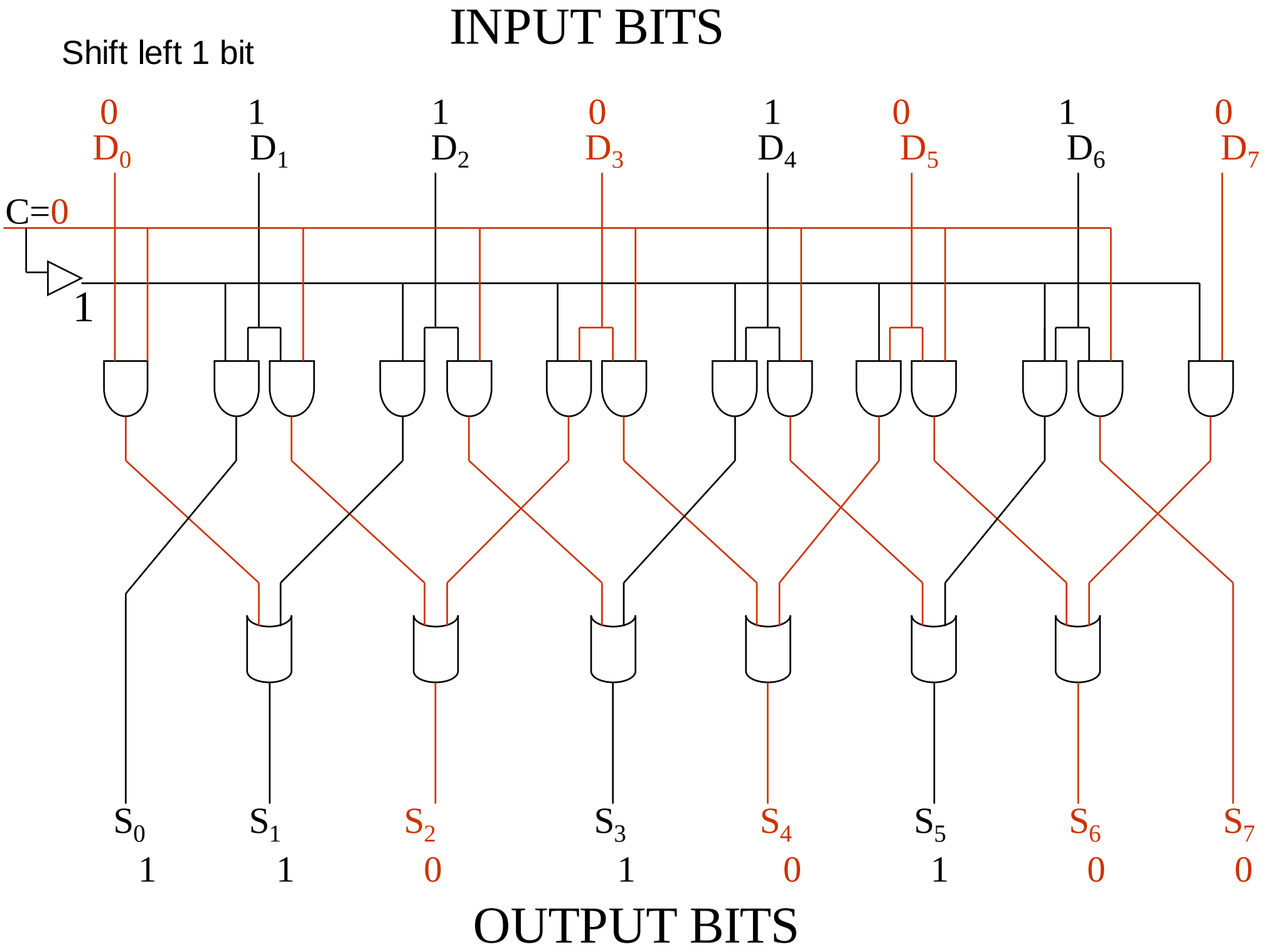
The black lines are: 1

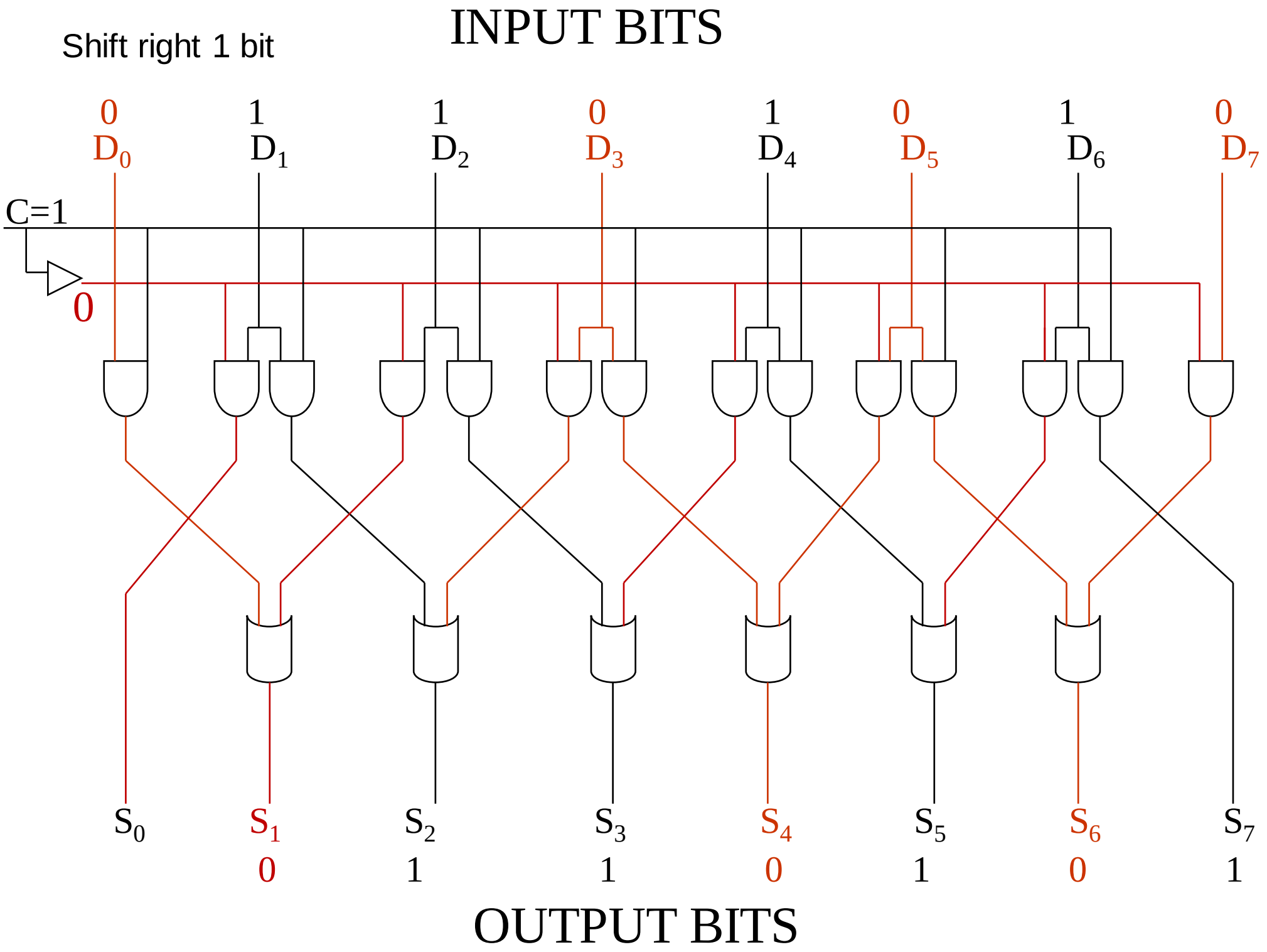
Shifter (An example)

We will now use the above ‘shifter circuits’ with an example:

In the following example: The red lines are: 0

The black lines are: 1


INPUT BITS
Shift left 1 bit
0
D0
1
D1
1
D2
0
D3
1
D4
0
D5
1
D6
0
D7
C=0
1
S0
1
S1
1
S2
0
S3
1
S4
0
S5
1
S6
0
S7
0
OUTPUT BITS


Shift right 1 bit
INPUT BITS
0
D0
1
D1
1
D2
0
D3
1
D4
0
D5
1
D6
0
D7
C=1
0
S0
S1
0
S2
1
S3
1
S4
0
S5
1
S6
0
S7
1
OUTPUT BITS

Shifter

In the above example we used one control bit to determine if the ‘shifter’ was to shift left or right.

We need two control bits to determine whether the shifter should:

- shift left
- shift right
- don’t shift

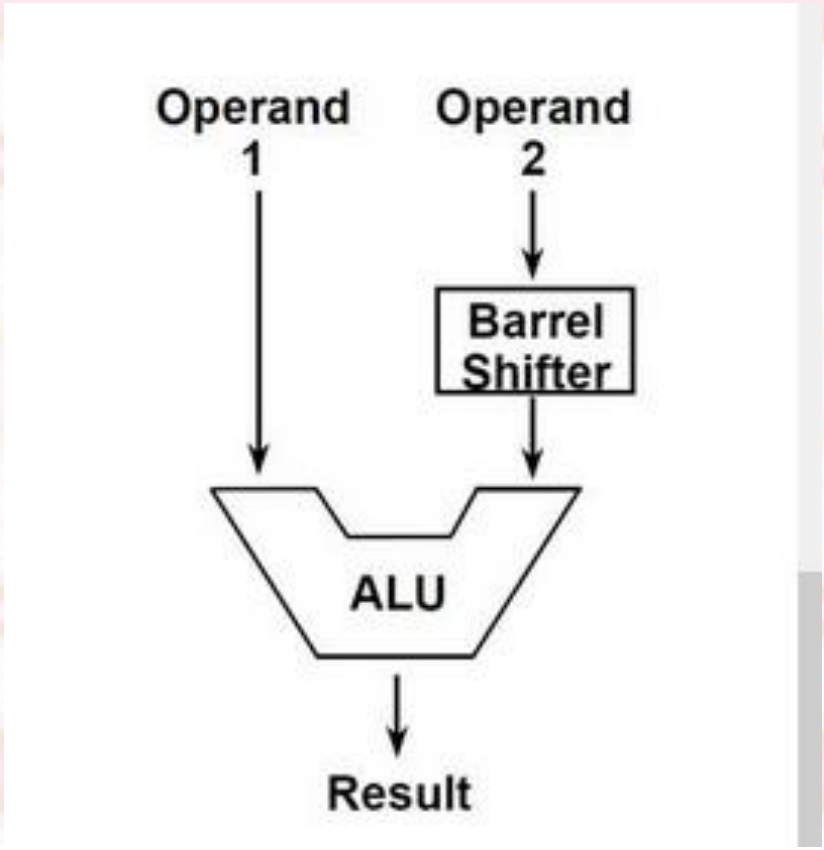
Shifter Exercise

In the above example the control C was _____
and the shifter shifted the bits to the _____

What is the control bit to shift in the other direction? _____

In the previous example what would the OUTPUT BITS be if the INPUT BITS are: 0 0 1 1 0 1 0 1 and the control bit is 0?

ARM Block Diagram

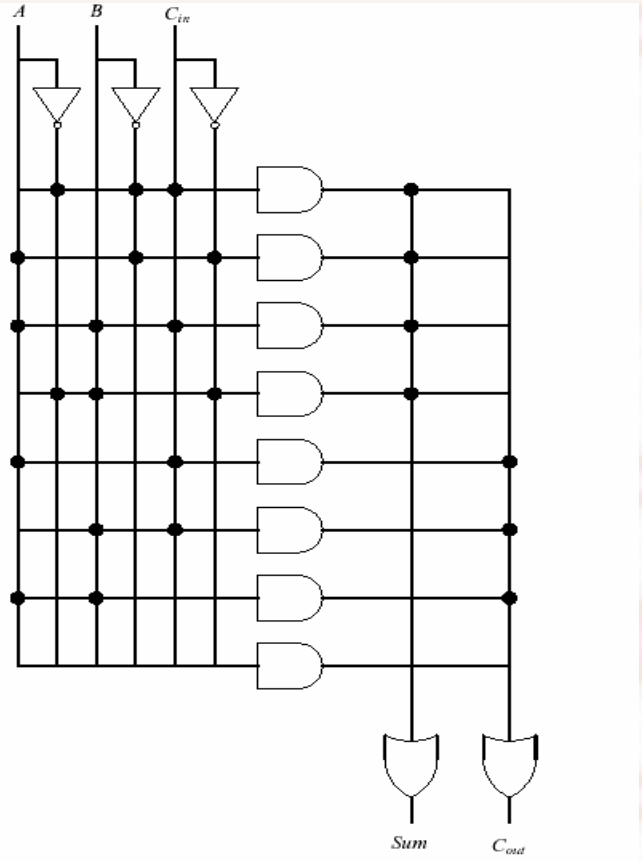


Inclass Exercise

- Draw a 2-bit Full-Adder using AND, XOR, and OR logic gates
- Draw a 2-bit Shifter using AND, OR, and NOT logic gates

PLA realization of a Full Adder

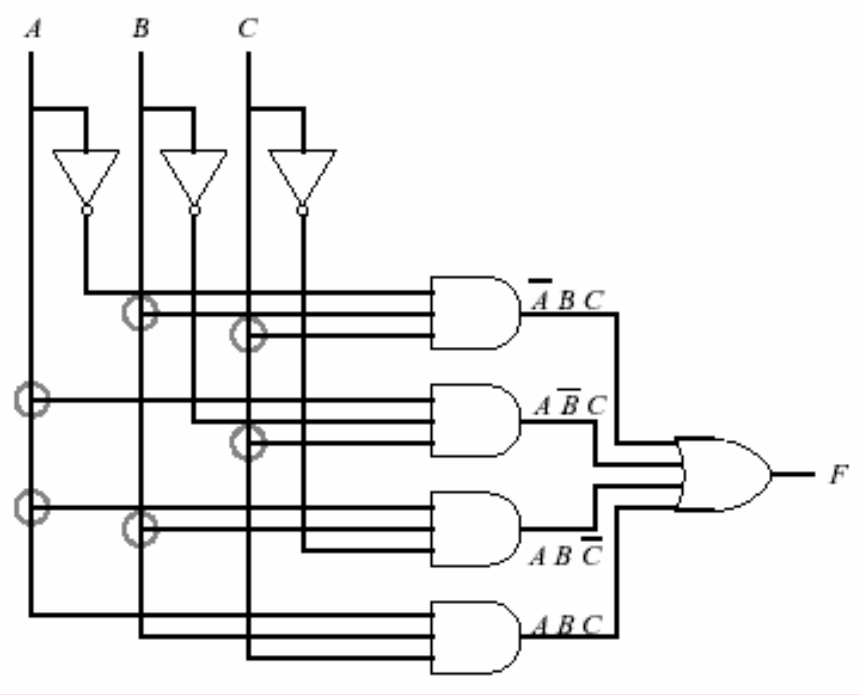
- An approach to designing a full adder is to use a PLA.
- CAD tools for VLSI favor use of PLAs over random logic or MUXes because of

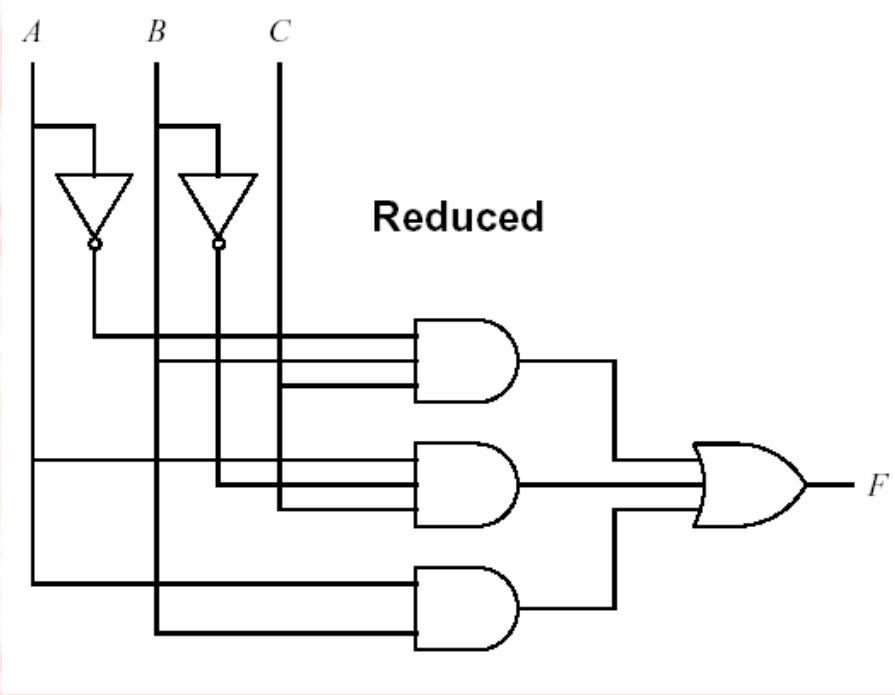


Simplification of Boolean Expressions

- It is usually possible to simplify the canonical SOP (sum of products) or POS (product of sums) forms.
- A smaller Boolean equation generally translates to a lower gate count in the target circuit.

Reduced Majority Function Circuit





Unreduced

Reduced

- Can the function be further reduced?

The Algebraic Method

$$F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

$$F = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C) \quad \text{Distributive Property}$$

$$F = \overline{A}BC + A\overline{B}C + AB(1) \quad \text{Complement Property}$$

$$F = \overline{A}BC + A\overline{B}C + AB \quad \text{Identity Property}$$

$$F = \overline{A}BC + A\overline{B}C + AB + ABC \quad \text{Idempotence}$$

$$F = \overline{A}BC + AC(\overline{B} + B) + AB \quad \text{Identity Property}$$

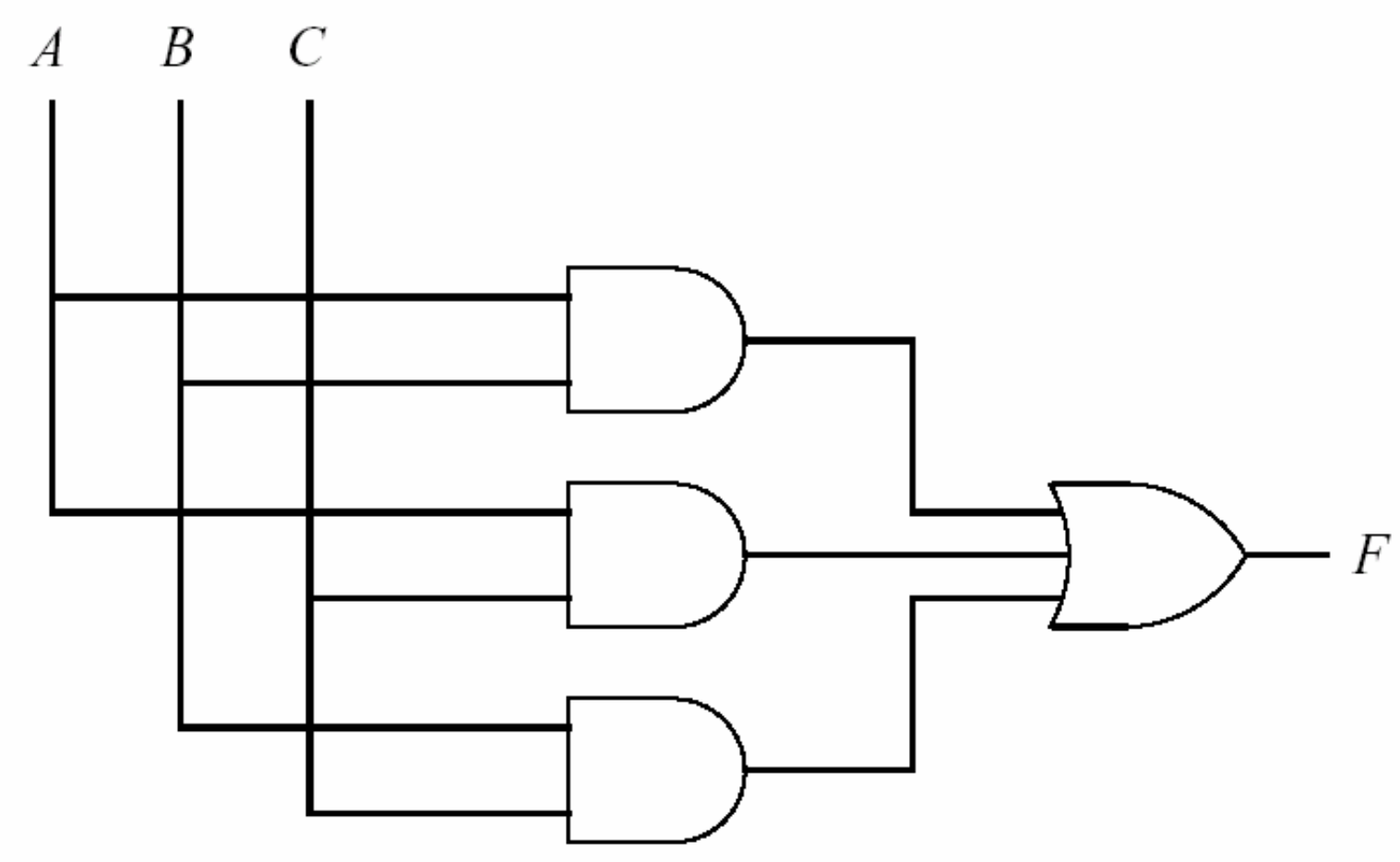
$$F = \overline{A}BC + AC + AB \quad \text{Complement and Identity}$$

$$F = \overline{A}BC + AC + AB + ABC \quad \text{Idempotence}$$

$$F = BC(\overline{A} + A) + AC + AB \quad \text{Distributive}$$

$$F = BC + AC + AB \quad \text{Complement and Identity}$$

- This majority circuit is functionally equivalent to the previous majority circuit, but this one is in its minimal two-level form:



Questions?

- ???