Combinational Circuits As Building Blocks

Combinational logic circuits can perform commonly used operations (such as arithmetic operations, comparison, selection, decoding, etc.)

There exist corresponding logic structures (such as adders, multiplexers, and decoders) for performing these operations.

Instead of designing every complex function with basic logic gates, using these common structures makes the design simpler.

Their level of functionality often matches a designer's level of thinking when partitioning a large problem into smaller chunks (like functions in programming).

These structures can be interconnected to construct more extensive systems.

We design hardware using a hierarchical approach:

- We design a small component (e.g., a 1-bit adder) using basic logic gates.
- We build a large component by interconnecting many copies of the small component + a few extra gates (e.g., a 32-bit adder).
- We build chips by interconnecting many large components (e.g., a CPU).
- Each component is truly made out of many gates but using a hierarchy makes the design process faster and easier.

Half Adder:

It adds two 1-bit numbers (without carry input).

Remember the rules of binary addition on slide 1.22.

a: First number
b: Second number
s: Sum (Result)
c: Carry output

Always fully label all inputs and outputs.

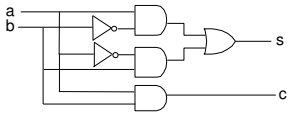
Truth table:

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

From the truth table, the logical expression is obtained.

$s = a\bar{b} + \bar{a}b$

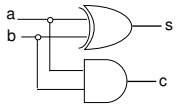
$c = ab$



The circuit can also be implemented using XOR gates.

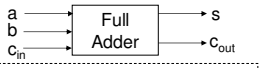
s = a ⊕ b

c = ab



Full Adder:

It adds two 1-bit numbers with a carry input.

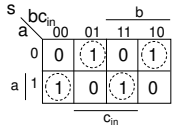


Always fully label all inputs and outputs.

a : First number
b : Second number
c_{in}: Carry Input
s : Sum (Result)
c_{out}: Carry Output

Truth table:

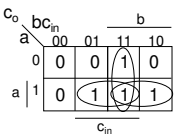
a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$s = \bar{a}\bar{b}c_{in} + \bar{a}b\overline{c_{in}} + a\bar{b}\overline{c_{in}} + abc_{in}$

s = a ⊕ (b ⊕ c_{in})

s = a ⊕ b ⊕ c_{in}



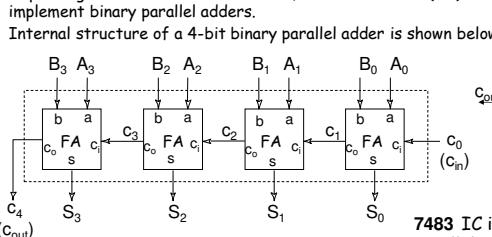
$c_{out} = ac_{in} + bc_{in} + ab$

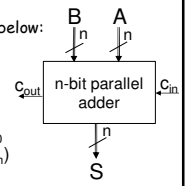
n-Bit Binary Parallel Adder:

It adds two n-bit binary numbers.

Depending on the size of the numbers, 1-bit full adders (FA) can be connected to implement binary parallel adders.

Internal structure of a 4-bit binary parallel adder is shown below:

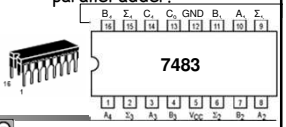




1. Number : $A_3A_2A_1A_0$
2.Number : $B_3B_2B_1B_0$
Result : $S_3S_2S_1S_0$
Carry In : c_0
Carry Out : c_4

Example:
1. Num.: 0110
2.Num.: 1100
Result : 0010
Carry : 1

7483 IC is a 4-bit binary parallel adder.



Subtraction Circuit

Subtraction is most easily accomplished as "addition using 2's complement".

A subtraction circuit can be implemented with an n-bit adder and NOT gates.

Example: A 4-bit subtraction circuit

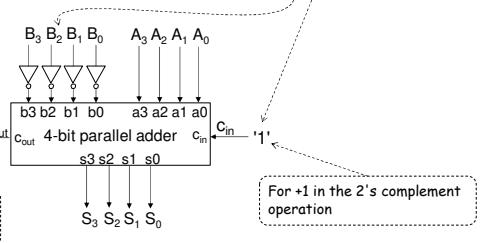
S = A – B

Here + is arithmetic operator for addition (not logic OR).

2's complement of B is added to A.

S= A – B = A + 2's complement(B) = $\boldsymbol{A + (\overline{B} + 1)}$

Always fully label all inputs and outputs.

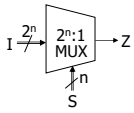


c_{out} = 0: Borrow
c_{out} = 1: No Borrow
(*if unsigned*)

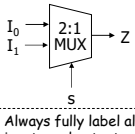
For +1 in the 2's complement operation

Multiplexer (MUX) (Data Selector):

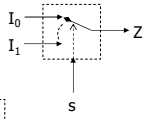
- 2^n data inputs (I), n selector (control) inputs (S), 1 data output (Z).
- The control inputs (Select lines S) are used to select one of the data inputs (I_x) and connect it to the output terminal (Z).
- Multiplexers are named **m:1**, based on the number of data inputs. Here, m is the number of data inputs.



Example: 2:1 Multiplexer (Read as "2 to 1 multiplexer".)



Always fully label all inputs and outputs.



Function:
if s=0, then Z=I_0
if s=1, then Z=I_1

Function Table:

s	Z
0	I_0
1	I_1

Logic Expression:
$Z = \bar{s}I_0 + sI_1$

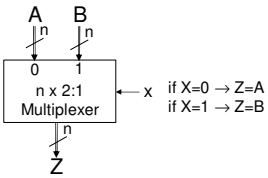
Truth Table:

I_1	I_0	s	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Parallel connection of multiplexers:

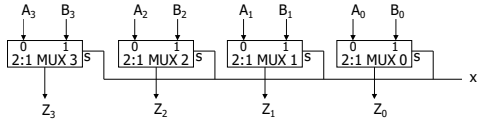
To select one of two n-bit data words, n units of 2:1 multiplexers have to be connected in parallel.

The circuit with the block diagram given on the right side, forwards one of the n-bit numbers (A or B) to the output Z according to the selector line x.



if X=0 → Z=A
if X=1 → Z=B

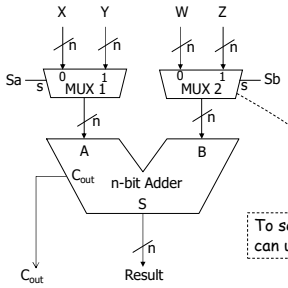
Example: A circuit that forwards one of the 4-bit numbers A or B to the output Z.



In this circuit, selector lines of all multiplexers are connected (short-circuited).

Examples of Usage of Multiplexers:

Example 1:
The same adder circuit can be used to add different numbers from different sources.

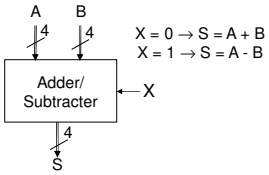


Sa	Sb	Result
0	0	X+W
0	1	X+Z
1	0	Y+W
1	1	Y+Z

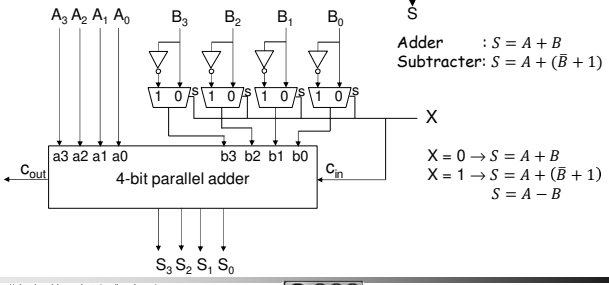
To select one of two n-bit numbers, we can use n x 2:1 multiplexers

Example 2:
4-bit Adder/Subtracter

Input X is used to control the function of the circuit.
If X is zero, it is an adder.
If X is one, it is a subtracter.



X = 0 → S = A + B
X = 1 → S = A - B



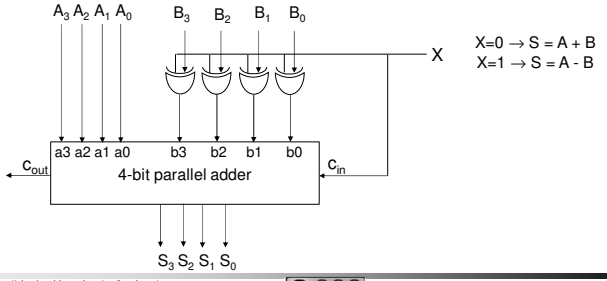
Adder : $S = A + B$
Subtracter: $S = A + (\bar{B} + 1)$

X = 0 → $S = A + B$
X = 1 → $S = A + (\bar{B} + 1)$
$S = A - B$

Example 2: (cont'd)
4-bit Adder/Subtracter

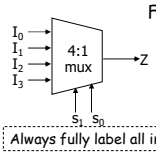
We can also design the adder/subtracter circuit using XOR gates instead of multiplexers and NOT gates.

Remember: if one input of an XOR gate is 0, it functions as a buffer: $0 \oplus x = x$
if one input of an XOR gate is 1, it functions as an inverter: $1 \oplus x = \bar{x}$



X=0 → S = A + B
X=1 → S = A - B

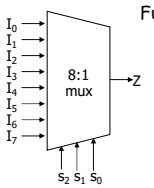
Multiplexers (MUX) of different sizes:



Function Table:

S_1 S_0	Z
0 0	I_0
0 1	I_1
1 0	I_2
1 1	I_3

Always fully label all inputs and outputs.



Function Table:

S_2 S_1 S_0	Z
0 0 0	I_0
0 0 1	I_1
0 1 0	I_2
0 1 1	I_3
1 0 0	I_4
1 0 1	I_5
1 1 0	I_6
1 1 1	I_7

Logic Expressions:

2:1 mux: $\mathbf{Z = \bar{s}\, I_0 + s\, I_1}$

4:1 mux: $\mathbf{Z = \overline{s_1}\,\overline{s_0}\, I_0 + \overline{s_1}\, s_0\, I_1 + s_1\,\overline{s_0}\, I_2 + s_1\, s_0\, I_3}$

8:1 mux: $\mathbf{Z = \overline{s_2}\,\overline{s_1}\,\overline{s_0}\, I_0 + \overline{s_2}\,\overline{s_1}\, s_0\, I_1 + \overline{s_2}\, s_1\,\overline{s_0}\, I_2 + \overline{s_2}\, s_1\, s_0\, I_3 + s_2\,\overline{s_1}\,\overline{s_0}\, I_4 + s_2\,\overline{s_1}\, s_0\, I_5 + s_2\, s_1\,\overline{s_0}\, I_6 + s_2\, s_1\, s_0\, I_7}$

General Expression (k:1 MUX): $Z = \sum_{j=0}^{k-1} (m_j I_j)$

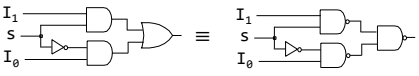
k=2^n , m_j= jth minterm, n = number of control inputs

Exemplary Integrated Circuit:
The IC 74151 contains an 8:1 multiplexer.

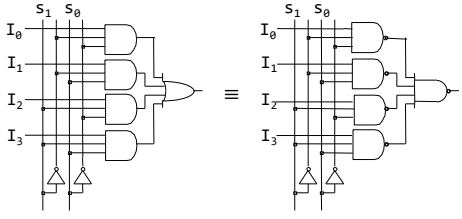
Internal structure of the multiplexers:

Multiplexers can be implemented using logic gates.

2:1 mux



4:1 mux



Design of Logic Circuits Using Multiplexers 1:

A logic circuit with n inputs and one output can be implemented by using a single 2^n:1 multiplexer and no other logic gates.

Method:

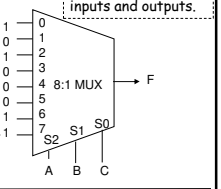
- The n inputs of the function (circuit) to be implemented are connected to the n selector lines of the multiplexer.
- Since each binary value of selector lines corresponds to an input combination, connect:
  - "0" to data input lines corresponding to 0-generating combinations, and
  - "1" to data input lines corresponding to 1-generating combinations.

Example:

F(A,B,C) = m0 + m2 + m6 + m7 = $\cup_1$(0,2,6,7)

Always fully label all inputs and outputs.

#	A	B	C	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1



Design of Logic Circuits Using Multiplexers 2:

A logic circuit with n inputs and one output can be implemented using a single 2^{n-1}:1 multiplexer and a NOT gate.

Method:

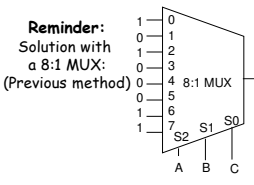
- Connect the n-1 inputs (variables) of the function to the n-1 select lines of the multiplexer.
- Then, connect the remaining single variable, or its complement, or 0, or 1 to the selection inputs of the multiplexer according to the values in the truth table.

Example:

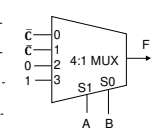
F(A,B,C) = m0 + m2 + m6 + m7 = $\cup_1$(0,2,6,7)

Solution with a 4:1 MUX:

Reminder: Solution with a 8:1 MUX: (Previous method)



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

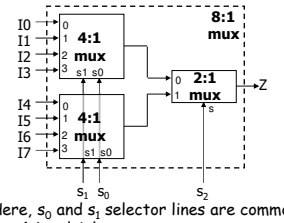


Here, both $\bar{C}$ values are obtained from the same NOT gate.

Implementing multiplexers of larger sizes using smaller multiplexers:

The following examples illustrate the implementation of an 8:1 multiplexer using other multiplexers in two different ways.

1. Method:

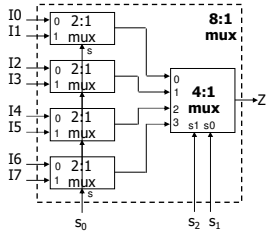


Here, s_0 and s_1 selector lines are common for 4:1 multiplexers.

The same inputs are selected for both multiplexers.

Selector s_2 determines which multiplexer's output is selected.

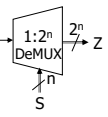
2. Method:



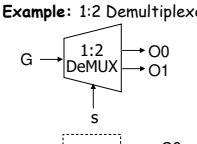
Always fully label all inputs and outputs.

Demultiplexer:

- 1 data input, n selector (control) lines, 2^n data outputs.
- It selects one of the many data output lines and connects it to the single input.
- The binary value on the select inputs determines the output line to which the data input is forwarded.
- The value on the not-selected output lines is "0".
- Demultiplexers are named **1:m**, based on the number of data outputs.



Example: 1:2 Demultiplexer

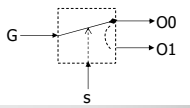


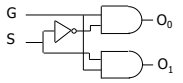
Function Table:

s	O_1	O_0
0	0	G
1	G	0

Truth Table:

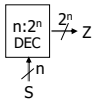
s	G	O_1	O_0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0



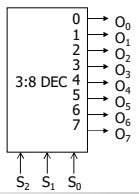


Decoder:

- n selector (control) inputs, 2^n outputs.
- According to the value on the select lines, only one output gets the value "1", and other outputs are "0".
- The decoder can be considered as a demultiplexer that has a constant "1" on its input.
- Decoders are named **n:2ⁿ** according to their lines. Here, n is the number of selector lines, and 2^n is the number of outputs.

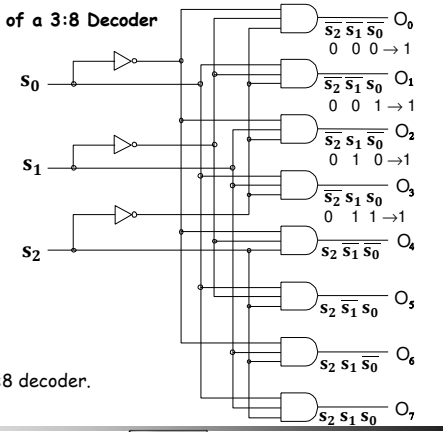


Example: 3:8 Decoder



#	S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
2	0	1	0	0	0	0	0	0	1	0	0
3	0	1	1	0	0	0	0	1	0	0	0
4	1	0	0	0	0	0	1	0	0	0	0
5	1	0	1	0	0	1	0	0	0	0	0
6	1	1	0	0	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0	0	0	0

Internal Structure of a 3:8 Decoder



An Exemplary IC:
74138 includes a 3:8 decoder.

Design of Logic Circuits Using Decoders:

Each possible input to the decoder can be considered as a minterm.

A decoder can be viewed as a "**minterm generator**" because each output is "1" only when a particular minterm evaluates to "1" (Slide 5.18).

Remember that any logic expression can be represented as the sum (OR) of minterms, so it follows that we can implement any logical expression by ORing the related output(s) of a decoder.

Method:

A general logic circuit with n inputs and m outputs can be implemented by using only one n:2^n decoder and, in addition with OR gates.

- n inputs (variables) of the function are connected to the n select lines of the decoder.
- Each output of a decoder corresponds to a minterm.
- The outputs of the decoder, which correspond to the minterms of the function are added by using an OR gate.

Example:

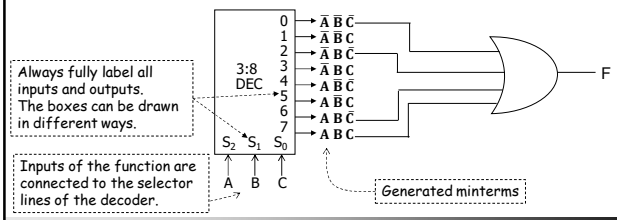
Implement the given function F(A,B,C) using a decoder and one OR gate.

$F(A,B,C) = \cup_1(0,2,6,7)$

Solution:

As the function F(A,B,C) has three inputs, we need a 3-to-8 decoder.

$F(A,B,C) = \cup_1(0,2,6,7) = m0 + m2 + m6 + m7 = \bar{A}\,\bar{B}\,\bar{C} + \bar{A}\,B\,\bar{C} + A\,B\,\bar{C} + A\,B\,C$

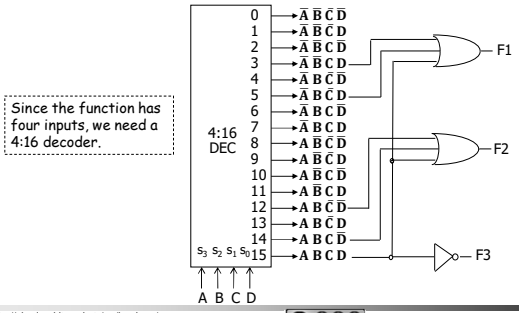


Example: Implementation of a function with 4 inputs and 3 outputs

$F1(A,B,C,D) = \bar{A}\,B\,\bar{C}\,D + \bar{A}\,\bar{B}\,C\,D + A\,B\,C\,D$

$F2(A,B,C,D) = A\,B\,\bar{C}\,\bar{D} + A\,B\,C$

$F3(A,B,C,D) = (\bar{A} + \bar{B} + \bar{C} + \bar{D})$



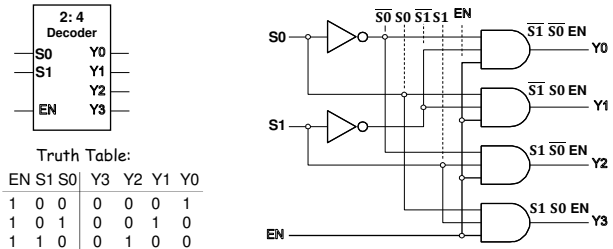
A decoder with an Enable (EN) input:

Decoders may also have an "enable" (EN) input.

If the EN input is "1", the decoder functions normally.

If the EN input is "0", all outputs of the decoder become "0".

Example: A 2:4 decoder with enable input is shown below:



Truth Table:

EN	S1	S0	Y3	Y2	Y1	Y0	
1	0	0	0	0	0	1	
1	0	1	0	0	1	0	
1	1	0	0	1	0	0	
1	1	1	1	0	0	0	
0	X	X	0	0	0	0	Disabled

An example of the usage of the decoders:

Some systems require only one unit (device) in a group to be active at a certain instant of time.

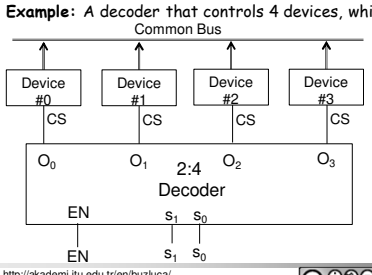
In other words, two devices cannot be active at the same time.

For example, memory modules connected to a **common bus**.

These types of devices have "chip select" (CS) inputs, which are used to activate or deactivate them. They have **three-state** outputs, as explained in the following slides.

Decoders can be used to select the active unit.

Example: A decoder that controls 4 devices, which are connected to a common bus.



EN	s_1	s_0	#0	#1	#2	#3
1	0	0	+	-	-	-
1	0	1	-	+	-	-
1	1	0	-	-	+	-
1	1	1	-	-	-	+
0	X	X	-	-	-	-

Three-State Logic

Normally, the output of a logic device is in one of the two logic states, i.e., "0" or "1".

Some logic devices are designed this way so their outputs can be in a **third state**.

This is often referred to as a **Hi-Z (high-impedance)** state of the output because the circuit offers a very high resistance or impedance to the flow of current.

In this state, the output behaves **like it is not connected to the circuit**.

The use of three-state logic permits the outputs of two or more gates or other logic devices to be connected together.

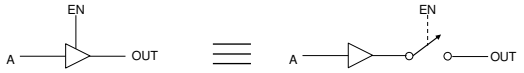
For example, the devices on slide 5.23 are designed to connect to a common bus.

When the chip select input of a device is not asserted, its output is in the third state.

We will cover the implementation of the devices with three-state outputs in section 11, "Internal Structures of Electronic Digital Circuits."

Example: Three-state buffer

Logical equivalent of the three-state buffer:



- IF EN = HIGH, THEN OUT = A
- IF EN = LOW, THEN OUT = Hi-Z

EN	A	OUT
0	0	Hi-Z
0	1	Hi-Z
1	0	0
1	1	1

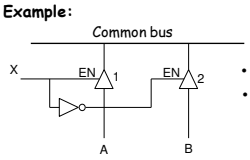
- When the enable input EN is 1, the output OUT equals A;
  when EN is 0, the output OUT acts like an open circuit (disconnected).
- In other words, when EN is 0, the output OUT is effectively disconnected from the buffer output so that no current can flow.

Three-State Common Bus

Several three-state outputs can be wired together to form a three-state common bus.

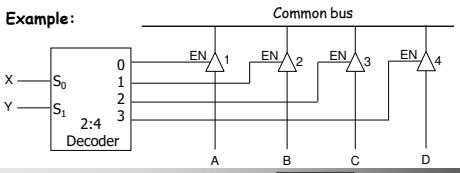
At any given moment, only one unit is enabled to drive the bus.

Example:



- If X=0, buffer #2 drives the bus. B is on bus.
- If X=1, buffer #1 drives the bus. A is on bus.

Example:



Y	X	Bus
0	0	A
0	1	B
1	0	C
1	1	D