## Integrated Combinational Circuits As Building Blocks

In combinational logic design, there are several common structures (such as adders, multiplexers, decoders) that are used regularly as building blocks in larger systems.

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 the large problem into smaller chunks. (As functions in programming.)

These structures are manufactured and sold as integrated circuits (ICs).

***Generations of ICs according to integration scale factors:***

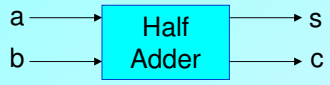
- Small-Scale Integration (**SSI**): These digital circuits contain transistors numbering in the tens and provide a few logic gates.
- Medium-Scale Integration (**MSI**): They contain hundreds (up to 1000) of transistors. Adders, decoders.
- Large-Scale Integration (**LSI**): Tens of thousands of transistors per chip. First memory and microprocessor chips
- Very Large-Scale Integration (**VLSI**): Hundreds of thousands of transistors in the early 1980s, and continues beyond several billion transistors as of 2009.
- Ultra-large-scale integration (**ULSI**): More than 1 million transistors.





## Half Adder:

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




a: First number
b: Second number
s: Result
c: Carry output

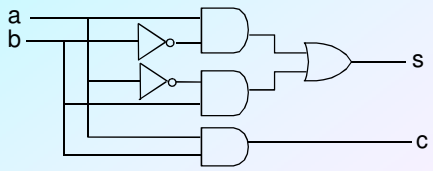
Truth table:

| a | b | c | s |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Always fully label all inputs and outputs.

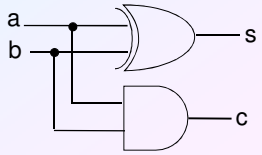
From the truth table the logical expression is obtained.

s= ab' + a'b
c= ab



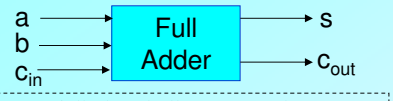

The circuit can also be implemented with XOR gates.

s= a ⊕ b
c= ab



## Full Adder:

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

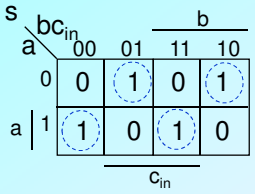



Always fully label all inputs and outputs.

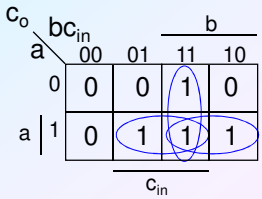
a : First number
b : Second number
$c_{in}$: Carry Input
s : Result
$c_{out}$: Carry Output

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 |




$s = a'b'c_{in} + a'bc_{in}' + ab'c_{in}' + abc_{in}$
$s = a \oplus (b \oplus c_{in})$
$s = a \oplus b \oplus c_{in}$



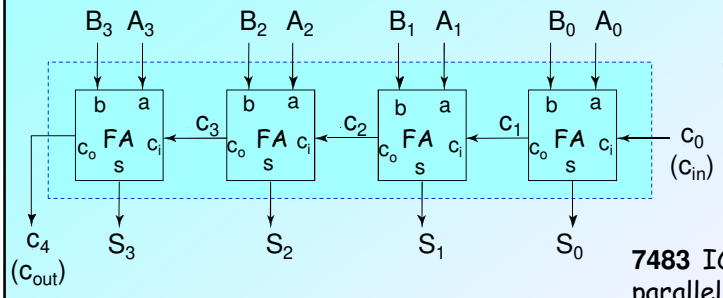

$c_{out} = ac_{in} + bc_{in} + ab$

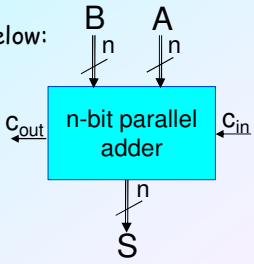




## n-Bit Binary Parallel Adder:

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:

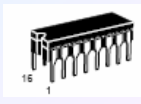


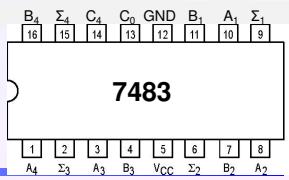




**7483** IC is a 4-bit binary parallel adder.

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





## Subtraction Circuit

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

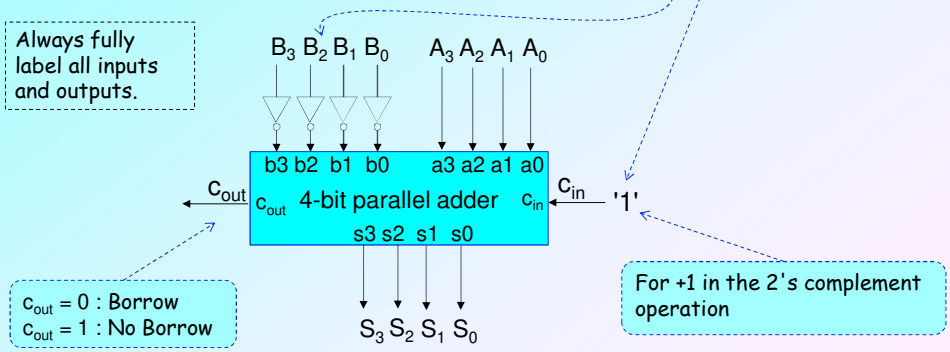
A subtraction circuit can be implemented with an n-bit adder and NOT gates.

**Example:** A 4-bit subtraction circuit

$S = A - B$

2's complement of B is added to A.

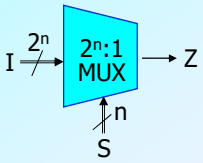
$$S = A - B = A + \text{2's complement}(B) = A + (B'+1)$$





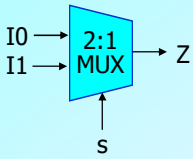


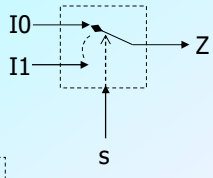

## Multiplexer (MUX) (Data Selector):




- $2^n$ 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 according to the number of data inputs as m:1. 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 Table:

| s | Z |
|---|---|
| 0 | $I_0$ |
| 1 | $I_1$ |

Logical Expression:

$$Z = s'\, I_0 + s\, I_1$$

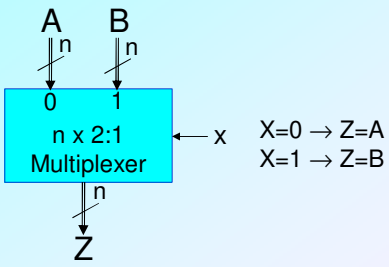
Truth Table:

| $I_1$ | $I_0$ | 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 |

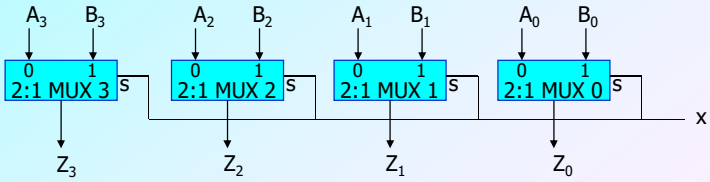
**Parallel connection of multiplexers:**

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.




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



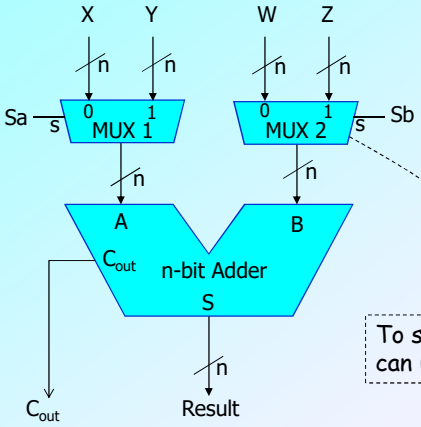

In this circuit, selector lines of all multiplexers are connected (short circuit).





**Examples of Usage of Multiplexers:**

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

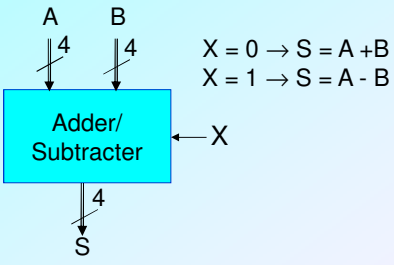



| Sa | Sb | Result |
|---|---|---|
| 0 | 0 | X+W |
| 0 | 1 | X+Z |
| 1 | 0 | Y+W |
| 1 | 1 | Y+Z |

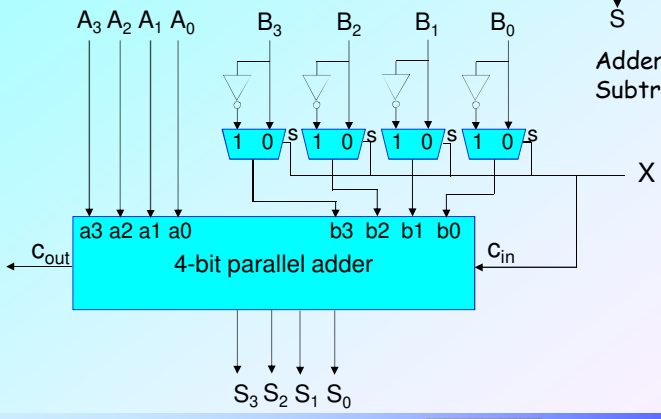
To select one of two n-bit numbers, we can use n x 2:1 multiplexers

**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 an 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 - B

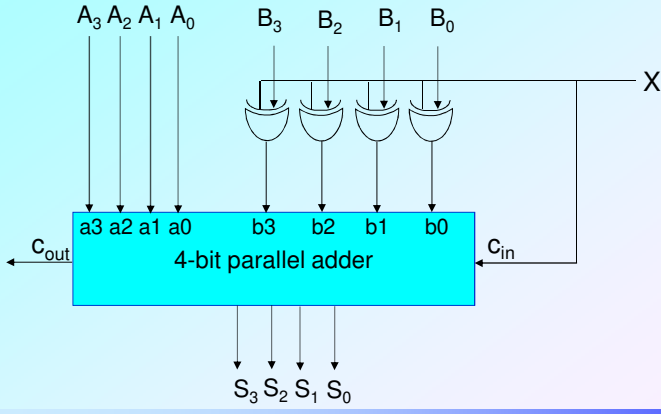




**Example 2:** (cont'd)
**4-bit Adder/Subtracter**

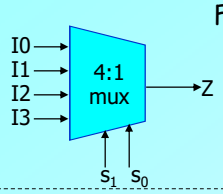
We can also design the adder/subtracter circuit using XOR gates instead of multiplexers and NOT gates.

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

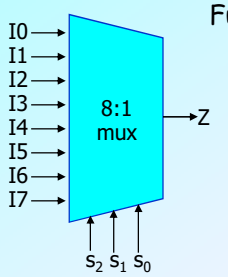
## Multiplexers (MUX) of different sizes:




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 | $I_7$ |

Logical Expressions:

2:1 mux: $Z = s' I0 + s I1$

4:1 mux: $Z = s_1' s_0' I0 + s_1' s_0 I1 + s_1 s_0' I2 + s_1 s_0 I3$

8:1 mux: $Z = s_2's_1's_0' I0 + s_2's_1's_0 I1 + s_2's_1s_0' I2 + s_2's_1s_0 I3 + s_2s_1's_0' I4 + s_2s_1's_0 I5 + s_2s_1s_0' I6 + s_2s_1s_0 I7$

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

k=$2^n$ , $m_j$= j. minterm, n = number of control inputs

**Exemplary Integrated Circuit:**

The IC 74151 contains an 8:1 multiplexer.

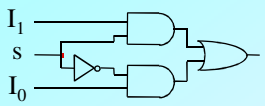


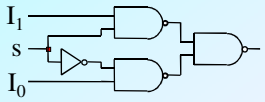


## Internal structure of the multiplexers:

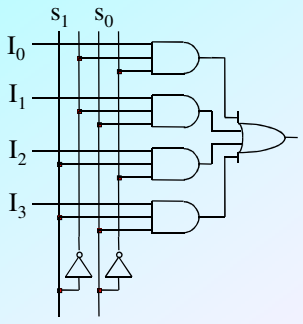
Multiplexers can be implemented using logic gates.

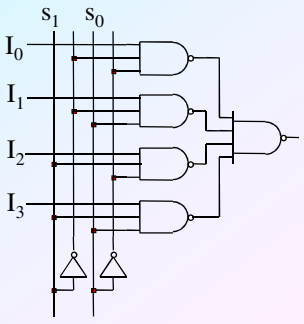
2:1 mux




4:1 mux



## Design of Logic Circuits Using Multiplexers 1:

A logic circuit with n inputs and one output can be implemented by using only one $2^n$:1 multiplexer and without any other logic gate.

**Method:**

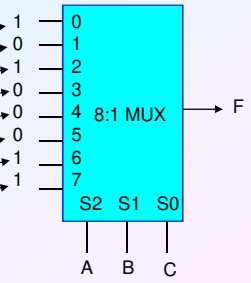
- The n inputs of the function (circuit) to be implemented are connected to the n selector lines of the multiplexer.
- Since each binary value of selector lines corresponds to an input combination , by considering the truth table of the function constant values ("0" or "1") are connected to the proper data inputs of the multiplexer.

**Example:**

Always fully label all inputs and outputs.

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

| # | 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 by using only one $2^{n-1}$:1 multiplexer and in addition with a NOT gate.

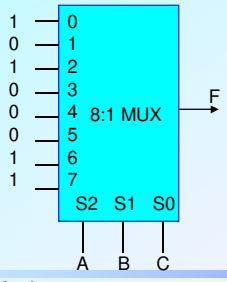
**Method:**

- n-1 inputs (variables) of the function are connected to the n-1 select lines of the multiplexer.
- The remaining variable or its complement is connected according to the truth table to the data inputs of the multiplexer.

**Example:**

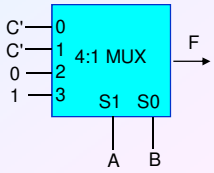
$F(A,B,C) = m0 + m2 + m6 + m7 = \cup_1(0,2,6,7)$

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




Solution with a 4:1 MUX:

| A | B | C | F | |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | C' |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 1 | C' |
| 0 | 1 | 1 | 0 | |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | |

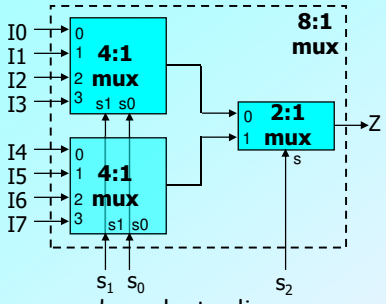



Here both c' values are obtained from the same NOT gate.

**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:

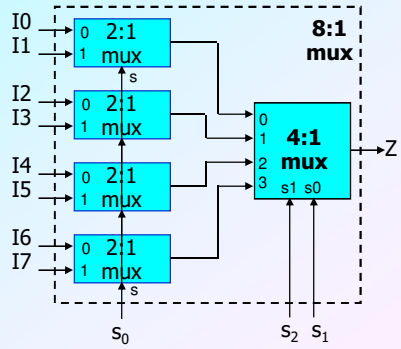



Here, $s_0$ and $s_1$ selector lines are common for 4:1 multiplexers.

Same inputs of both multiplexers are selected.

Selector $s_2$ determines which multiplexer's output is selected.

2. Method:

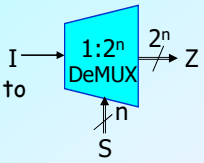



Always fully label all inputs and outputs.



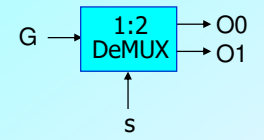


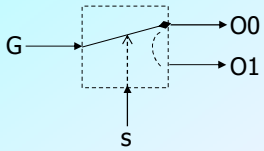
**Demultiplexer:**




- 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 according to the number of data outputs 1:m.

**Example:** 1:2 Demultiplexer



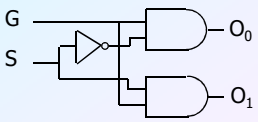




Function Table:

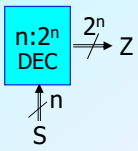
| s | $O_1$ | $O_0$ |
|---|---|---|
| 0 | 0 | G |
| 1 | G | 0 |

Truth Table:

| s | G | $O_1$ | $O_0$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |

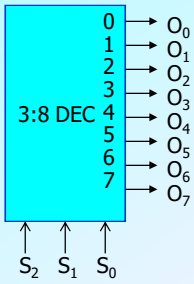


## Decoder:




- 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".
- Decoder can be considered as a demultiplexer that has a constant "1" on its input.
- Decoders are named as $n:2^n$ according to their lines. Here, n is the number of selector lines and $2^n$ is the number of outputs.

**Example:** 3:8 Decoder

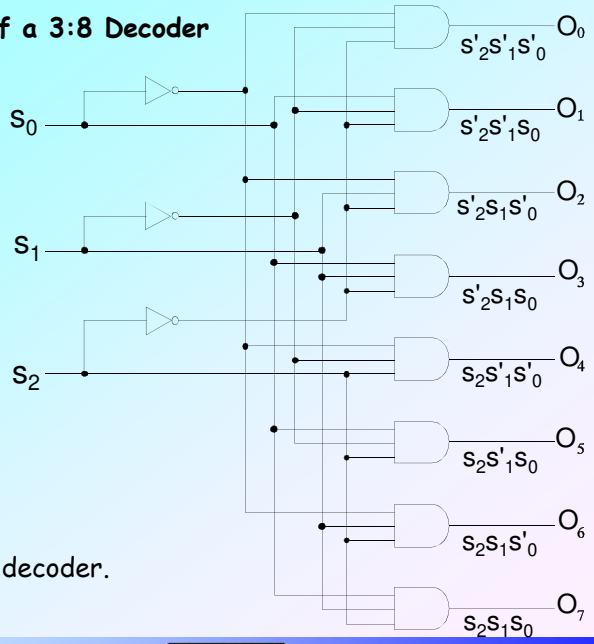



| # | $S_2$ | $S_1$ | $S_0$ | $O_7$ | $O_6$ | $O_5$ | $O_4$ | $O_3$ | $O_2$ | $O_1$ | $O_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 0 | 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 logical 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:

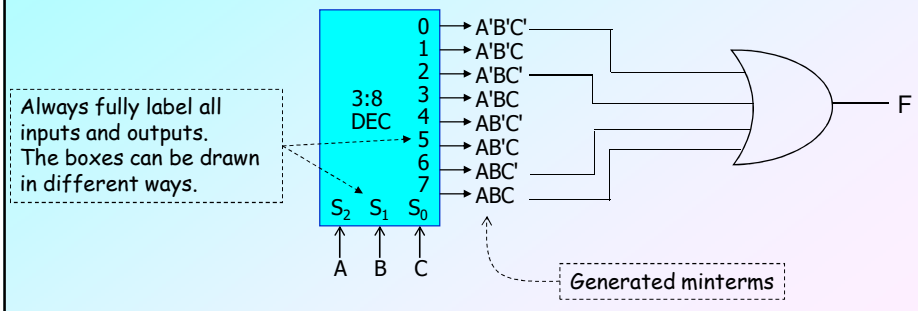
Implement the given function F(A,B,C) using a decoder and one OR gate.

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

Solution:

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

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

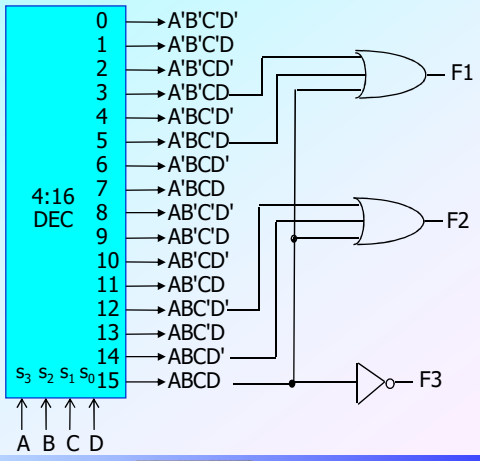


**Example:** Implementation of a function with 4 inputs and 3 outputs

F1(A,B,C,D) = A' B C' D + A' B' C D + A B C D

F2 (A,B,C,D) = A B C' D' + A B C

F3 (A,B,C,D) = (A' + B' + C' + D')







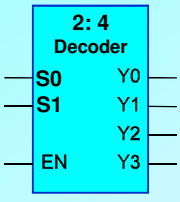

***A decoder with an Enable (EN) input:***

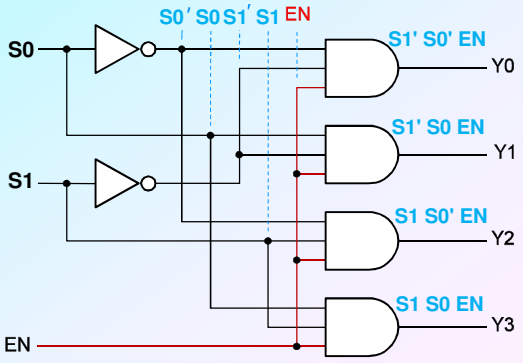
Decoders may also have an "enable" (EN) input.

If EN input is "1" decoder preforms normally.

If 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:**

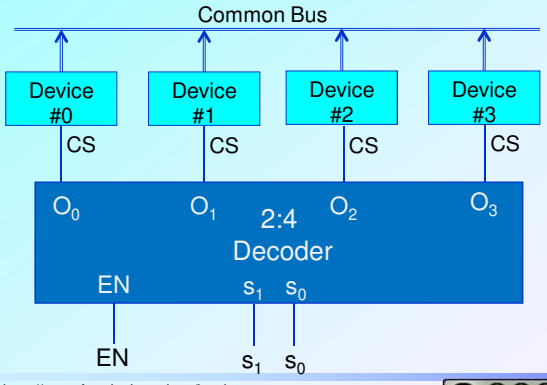
In some systems it is required that only one unit (device) in a group is active at a certain instant in time. For example, memory modules connected to a common bus.

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

These type of devices have «chip select» (CS) inputs, which are used to activate or deactivate them.

Decoders can be used to select the active unit.

**Example:** A decoder that controls 4 devices, which are connected to 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 | - | - | - | - |





## Programmable Logic Device (PLD)

Today, complicated digital circuits are implemented using programmable logic devices.

These devices are integrated circuits that include many reconfigurable logic gates. (From several hundreds to several millions).

Some PLDs also include memory units (flip-flops).

The designer can reconfigure the connections between logical gates in the PLD by using a programming language and a programming device.

It is possible to implement complicated digital circuits with only one IC (PLD).

There are different kinds of PLDs:

- *Programmable Logic Array* – **PLA**
- *Programmable Array Logic* – **PAL®**
- *Generic Array Logic* – **GAL**
- *Complex PLD* – **CPLD**
- *Field-Programmable Gate Array* – **FPGA**

PAL is a registered trademark of Lattice Semiconductor Corp.

**Programming of PLDs:**

In early versions of PLDs (PLA, PAL) bipolar transistors were used (See Chapter 9).

They have fuses on the connection points between gates, which provide reconfiguration (programming) of devices.

In these devices fuses can be blown only once; therefore they are called one-time programmable (OTP).

Todays devices (GAL, CPLD, FPGA) are made of CMOS transistors and they consist memory units for programming.

They can be erased and reprogrammed many times.

To program PLDs various Hardware Description Languages (**HDL**) and programing devices are used.

Some examples of HDLs:

PALASM

ABEL

Verilog

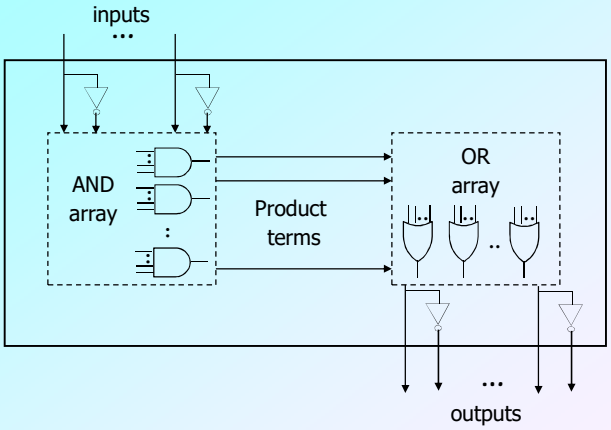
VHDL (*Very high speed integrated circuits* HDL)





**Programmable Logic Array - PLA**

It includes AND (product) units in the input layer and OR (sum) units in the output layer.




Both AND, OR arrays are flexible programmable.

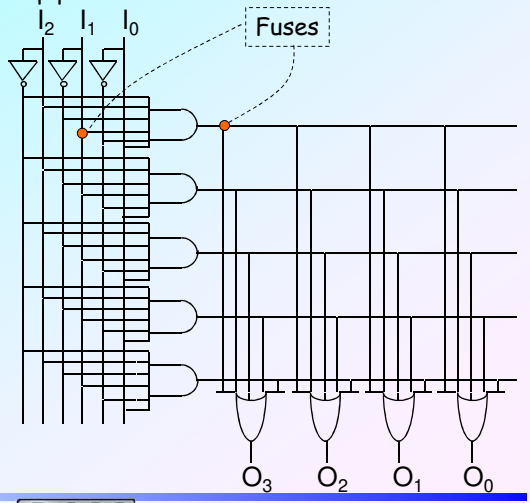
Parameters that determine the limits of a PLA:

Inputs: n
Outputs: m
AND gates (products):p
Such a device is called, "n x m PLA with p products".

On the right internal structure of a 3x4 PLA with 5 products is shown.

In real, PLAs include about 100 gates.

**Example:** 82S100
16 inputs, 8 outputs, 48 products





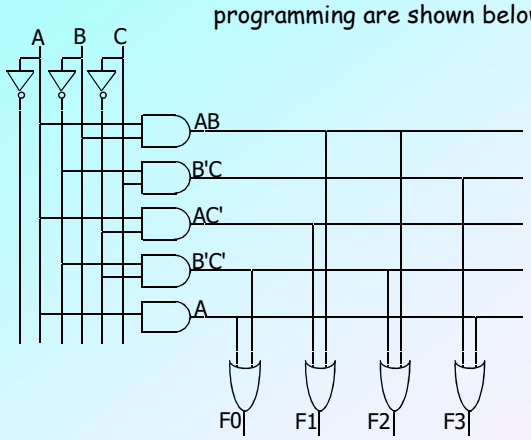



**Example:**

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

In the programming process unnecessary fuses are blown.

Internal connections of a 3x4 PLA with 5 products after programming are shown below.



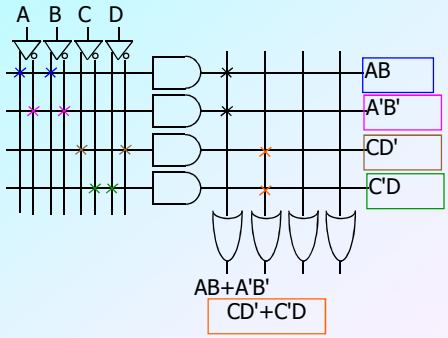
**Simple Representation:**

For the sake of simplicity all connections in a PLA are not shown.

Instead, we put an 'X' on the connection points which are connected to the inputs of gates.

Example:

F0 = A B + A' B'
F1 = C D' + C' D







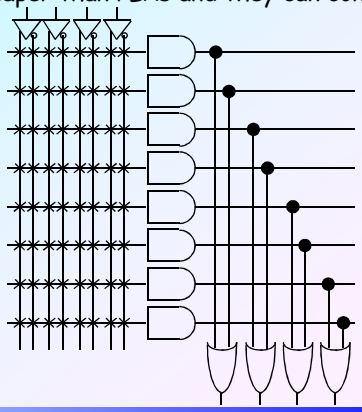

## Programmable Array Logic - PAL

Inputs of AND gates can be flexible programed as in PLAs.

But inputs of OR gates are fixed. To each OR gate only outputs of certain AND gates can be connected.

For example to the inputs of the first OR gate only outputs of the first two AND gates can be connected.

PALs can be easily programmed, they are cheaper than PLAs and they can contain more gates.



PALs were introduced by the company Monolithic Memories, Inc. (MMI).

MMI obtained a registered trademark on the term PAL for use in "Programmable Semiconductor Logic Circuits".

The trademark is currently held by Lattice Semiconductor Corporation.

MMI, was acquired by Advanced Micro Devices (AMD).

Programmable logic division of AMD (Vantis) was then acquired by Lattice Semiconductor.

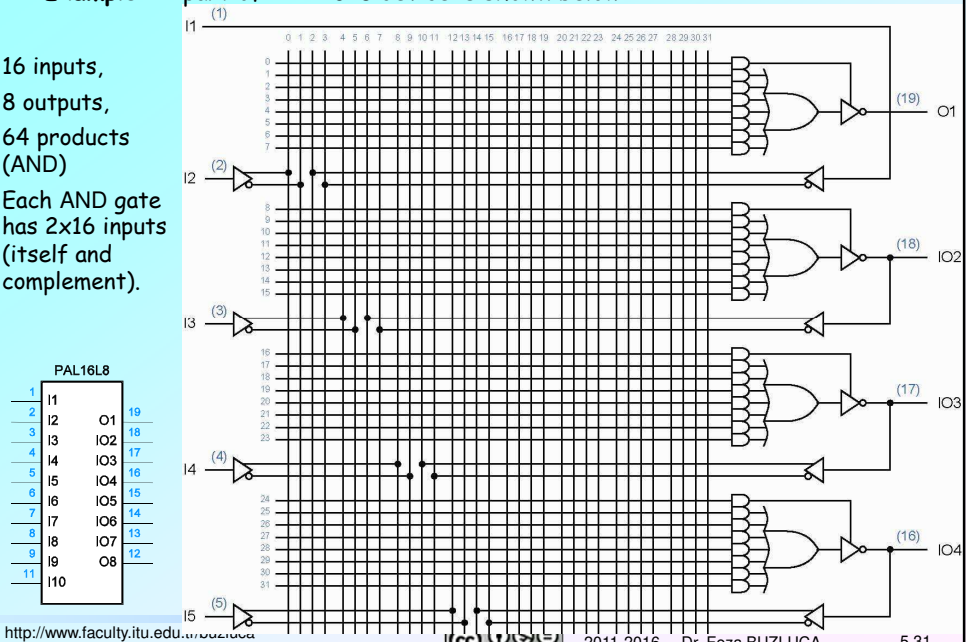
**Example:** A part of PAL 16L8 device is shown below:

16 inputs,

8 outputs,

64 products (AND)

Each AND gate has 2x16 inputs (itself and complement).











## Generic Array Logic – GAL

Its logical properties are similar to PAL.

It is made of CMOS transistors. It can be many times erased and programmed.

It is introduced by Lattice Semiconductor.

**Example:** GAL16V8

## Complex PLD – CPLD

It is an IC that contains several PLDs (*macro cell*).

Each internal PLD (macro cell) has GAL properties.

A typical CPLD may include from thousand to ten thousand gates.

Internal structures of macro cells and connections between them can be programmed.

**Example: Atmel ATF1500**

32 input/output + 4 inputs

32 PLDs (*macro cell*).

## Field-Programmable Gate Array – FPGA

They contain many logical blocks and interconnections between these blocks.

Can be erased and programmed many times.

Number of logical gates is between several thousands and several millions.

Can be used to implement complex digital circuits (for example special purpose microprocessors).

Compared to CPLDs FPGAs are more flexible and they can implement more complicated circuits.

But their delay cost is higher.

**Example: Atmel AT6010**

204 input/output

30000 gates