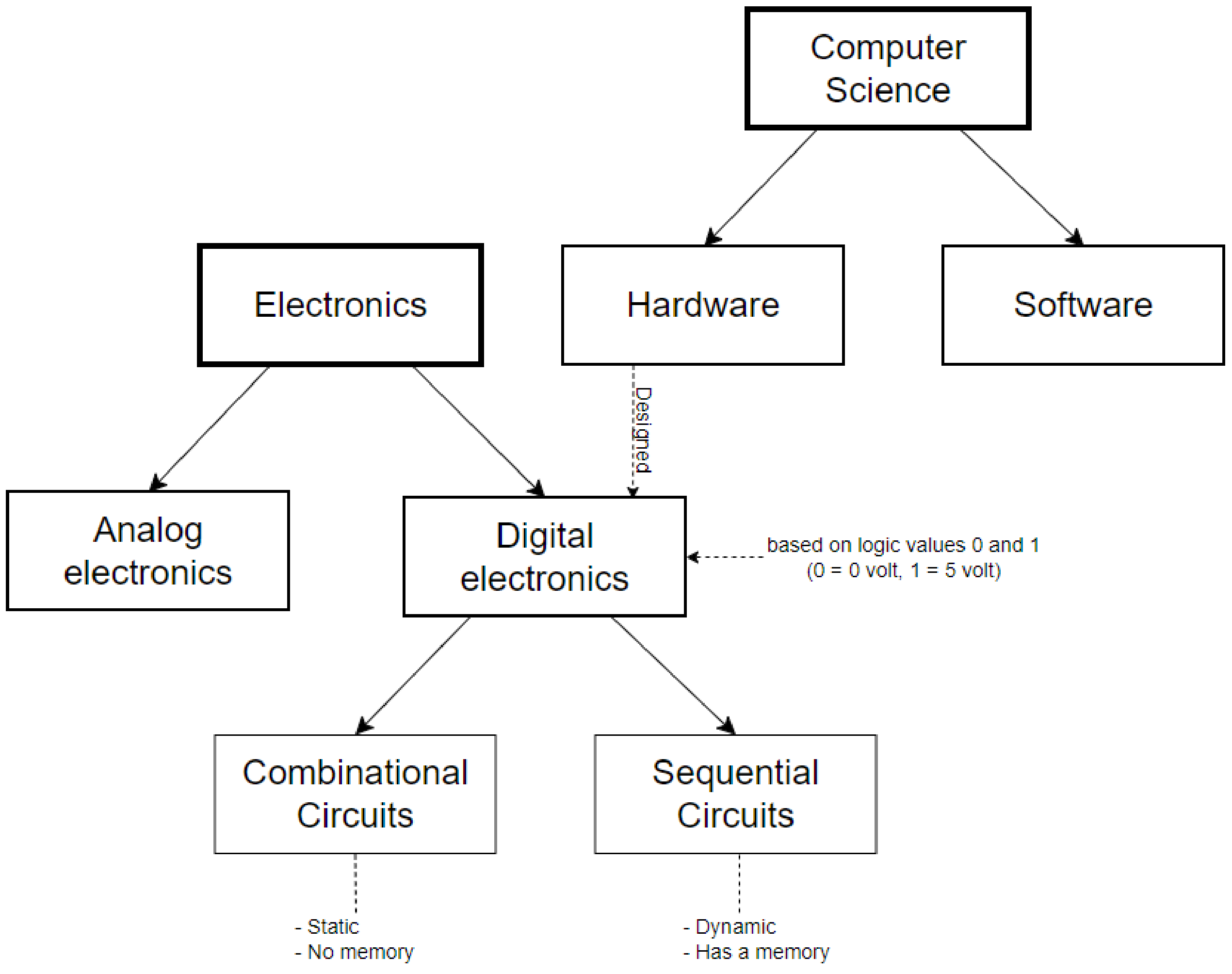
Sétif 1 University
Faculty of Sciences
Common Core Mathematics and Computer Science

2nd semester (year 2023/24)
Machine Structures 2 course
Kara Abdelaziz professor

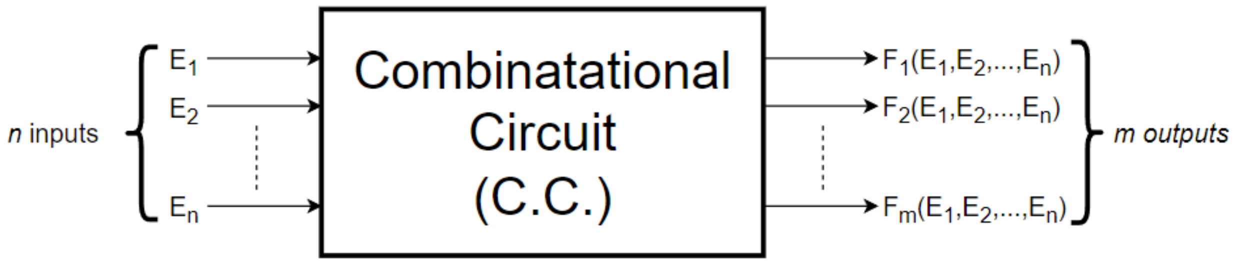
Chapter 1 : Combinational Circuit

1. Introduction



2. Définition

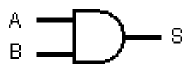
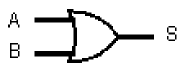
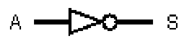
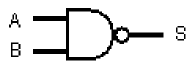
- ➔ Digital electronic circuit which internally performs binary processing from inputs to outputs.
- ➔ Possesses *n* inputs and *m* discrete digital binary outputs (0/1).
- ➔ Its <u>specification</u> is based on Boolean Functions and/or Truth Tables.



3. Logic Gates (L.G.)

- ➔ Elementary component (indivisible).
- ➔ Basic physical (real) element of digital electronics.
- ➔ They are assembled in composition to form digital circuits.
- ➔ They represent an elementary operation of Boolean logic (and, or, not...).

The different gates are:

Gate	Symbol	Truth table	Description
AND		A B S: 0 0 0 / 0 1 0 / 1 0 0 / 1 1 1	It is the logic **and** : $S = A \cdot B$
OR		A B S: 0 0 0 / 0 1 1 / 1 0 1 / 1 1 1	It is the logic **or** : $S = A + B$
NOT		A S: 0 1 / 1 0	It is the logic **not** : $S = \overline{A}$
NAND		A B S: 0 0 1 / 0 1 1 / 1 0 1 / 1 1 0	It is the logic **not-and** : $S = \overline{A \cdot B}$

AND:

A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

OR:

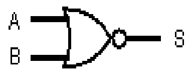
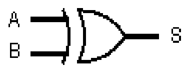
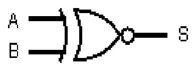
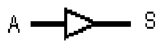
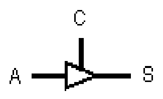
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

NOT:

A	S
0	1
1	0

NAND:

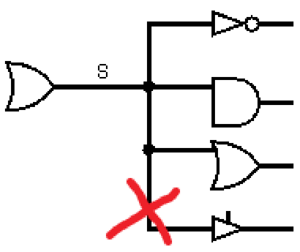
A	B	S
0	0	1
0	1	1
1	0	1
1	1	0

NOR		A B S 0 0 1 0 1 0 1 0 0 1 1 0	It is the logic **not-or** : S = $\overline{A+B}$
XOR		A B S 0 0 0 0 1 1 1 0 1 1 1 0	It is the logic **exclusive-or** : S = A⊕B
XNOR		A B S 0 0 1 0 1 0 1 0 0 1 1 1	It is the logic **not-exclusive-or** : S = $\overline{A⊕B}$ = A⊗B
buffer		A S 0 0 1 1	It is the **buffer**, a gate that doesn't do any logical operation, it is used to reduce the speed of the signal in some situations : S = A
tristate buffer		A C S 0 0 Z 1 0 Z 0 1 0 1 1 1	It is the **tristate buffer**, used to produce the Z logic signal : $\begin{cases} if\,(C=0) \Rightarrow S=Z \\ if\,(C=1) \Rightarrow S=A \end{cases}$

(A,B are inputs. S is the output. C for the command)

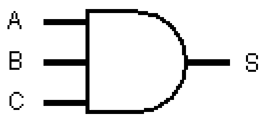
Fan-out : Characteristic of a gate which indicates the maximum number that its output S can provide for the inputs of the next gate.

Example : fan-out = 3
S cannot provide for more than 3 outputs.



Fan-in : Gate characteristic that indicates the number of inputs of a gate.

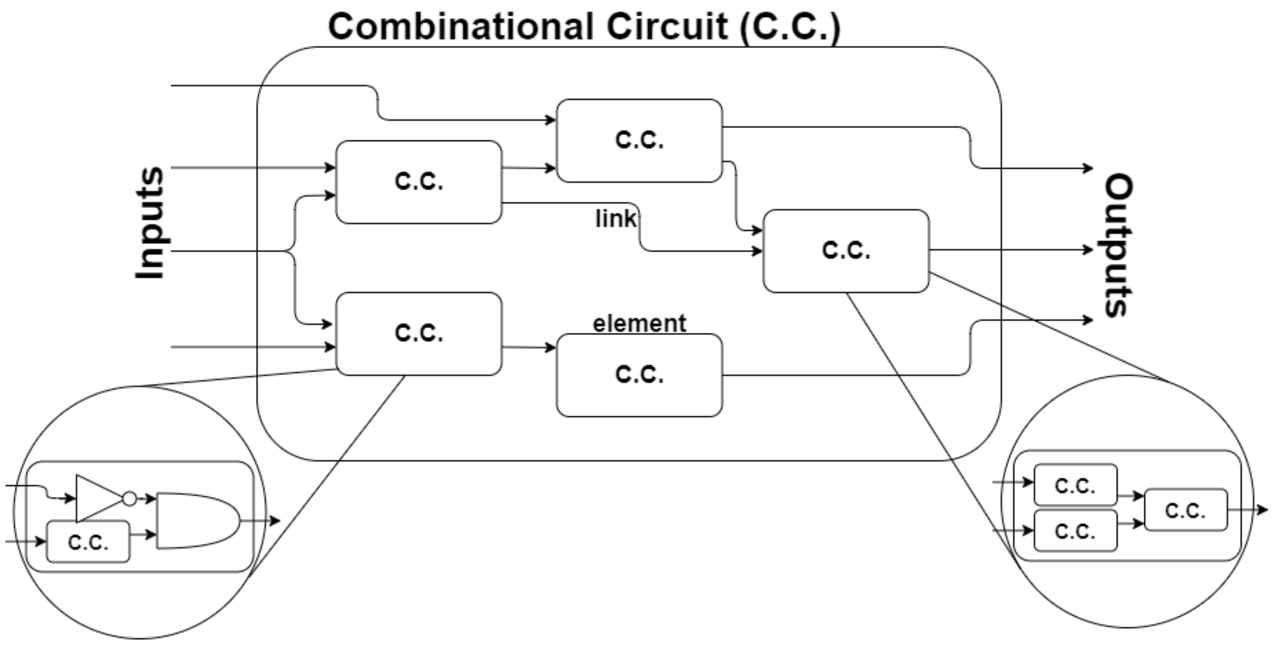
Example : The AND3 gate is an AND gate with 3 inputs.

Gate	Symbol	Truth table			
		A	B	C	S
AND3		0	0	0	0
		0	0	1	0
		0	1	0	0
		0	1	1	0
		1	0	0	0
		1	0	1	0
		1	1	0	0
		1	1	1	1

Note : The fan-out can be appended as a number at the end of a gate name (eg: AND3), and omitted if it is the default number (number 2) of inputs (eg: AND = AND2).

4. Rules to design a combinational circuit

a combinatorial circuit is described formally as a **graph**, made up of a set of **elements**, interconnected by oriented **links**.



A combinational circuit must respect the following 5 rules to be a valid combinational circuit

- The *element* can only be either a combinational circuit or a logic gate. This rule highlights the hierarchical modular nature of complex circuits.
- The *link* is an electrical wire carrying 0 or 5 Volts (logic 0 or 1). There are three types; inputs, outputs, and internal links.
- For each input combination, there is only one unique output combination, mathematically speaking it is a function (or application). This is also the origin of the name *combinational*.
- The input of an element cannot receive its signal from more than a single previous output, otherwise forming a short circuit (or *contention*).
- A signal's path cannot pass through an element more than once. In other words, there is no cycle (loop) in the circuit.

If one of the rules is not respected, the circuit is not a valid combinational circuit.

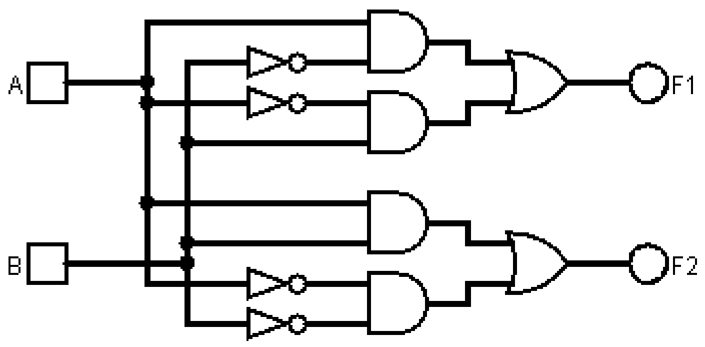
5. Combinational Circuit specification

The specification (operational description) of a combinational circuit is ensured by two formal tools which are; Boolean Functions and/or the Truth Table.

Example :

$F_1(A,B) = A\cdot\overline{B} + \overline{A}\cdot B$
$F_2(A,B) = A\cdot B + \overline{A}\cdot\overline{B}$

Solution :



A	B	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

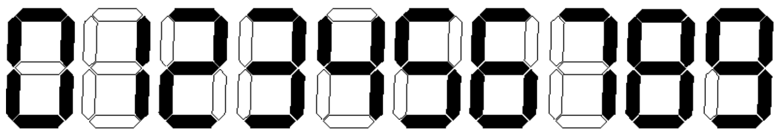
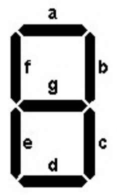
6. Conventional steps to design a combinational circuit

Conventionally 5 steps are followed to design a combinatorial circuit :

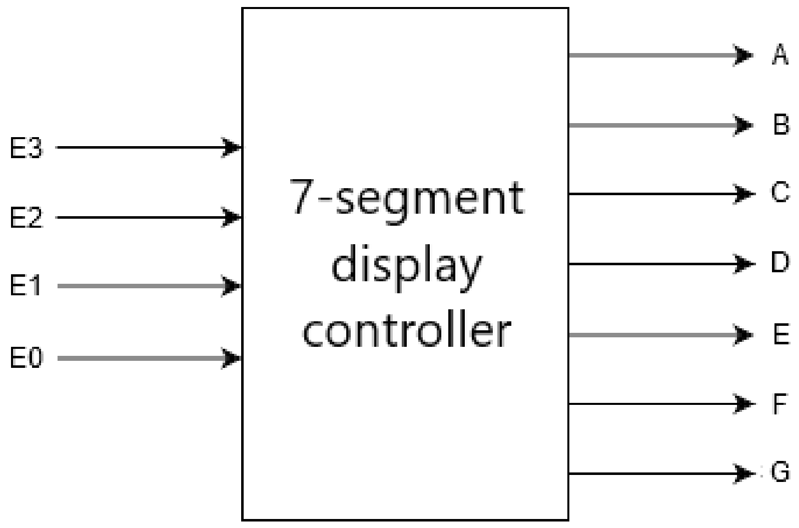
- Global Scheme
- Truth Table
- Boolean Functions
- Karnaugh table or Algebraic simplification
- Schematic

Example :

The combinational circuit of a 7-segment display is a circuit used for the control of a 7-segment display, it allows the display of a single digit in 7 segments as illustrated in the diagram below. It receives in its input a binary integer number encoded using 4 bits enclosed inside the interval [0,9] that represent the digit to display. And in its output it produces the combination of segments that displays the digit in the decimal format.

Step 1 : Global Scheme



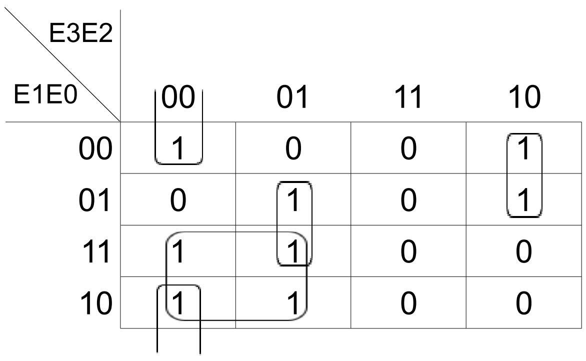
Step 2 : Truth Table

E3	E2	E1	E0	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

Step 3 : Canonical Disjunctive Functions

$$A(E3,E2,E1,E0) = \overline{E3}\cdot\overline{E2}\cdot\overline{E1}\cdot\overline{E0} + \overline{E3}\cdot\overline{E2}\cdot E1\cdot\overline{E0} + \overline{E3}\cdot\overline{E2}\cdot E1\cdot E0 + \overline{E3}\cdot E2\cdot\overline{E1}\cdot E0 + \overline{E3}\cdot E2\cdot E1\cdot\overline{E0} + \overline{E3}\cdot E2\cdot E1\cdot E0 + E3\cdot\overline{E2}\cdot\overline{E1}\cdot\overline{E0} + E3\cdot\overline{E2}\cdot\overline{E1}\cdot E0$$

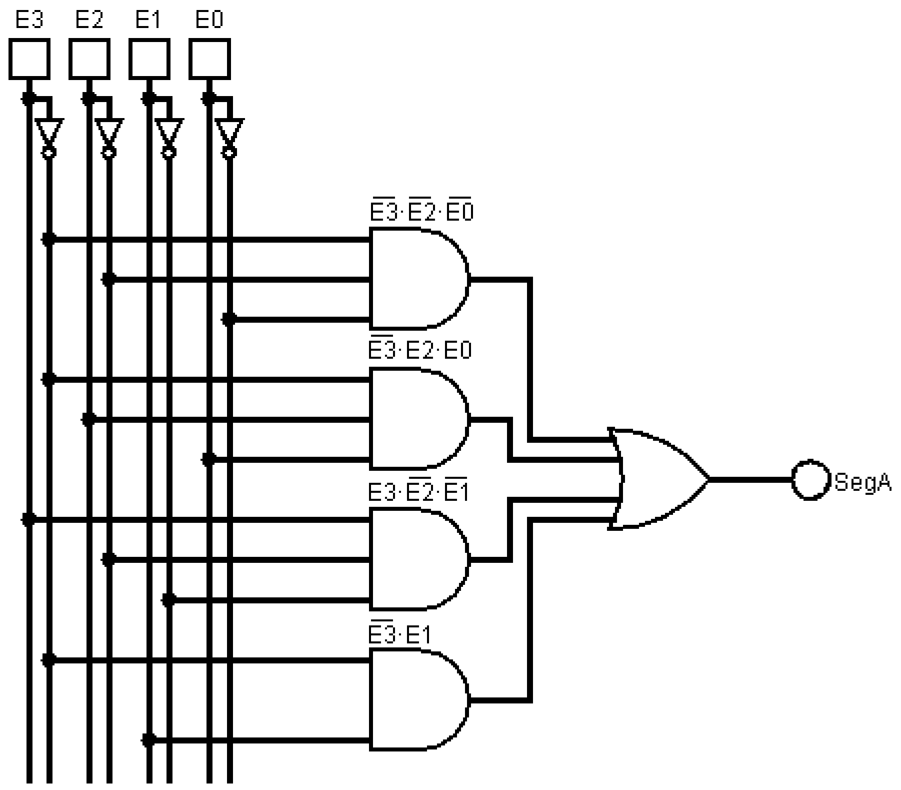
Step 4 : Karnaugh Table



E1E0 \ E3E2	00	01	11	10
00	1	0	0	1
01	0	1	0	1
11	1	1	0	0
10	1	1	0	0

$$A(E3,E2,E1,E0) = \overline{E3}\cdot\overline{E2}\cdot\overline{E0} + \overline{E3}\cdot E2\cdot E0 + E3\cdot\overline{E2}\cdot\overline{E1} + \overline{E3}\cdot E1$$

Step 5 : Schematic



E3
E2
E1
E0
$\overline{E3}\cdot\overline{E2}\cdot\overline{E0}$
$\overline{E3}\cdot E2\cdot E0$
$E3\cdot\overline{E2}\cdot\overline{E1}$
$\overline{E3}\cdot E1$
SegA

Note 1: This method is called 5-step method, or 2-stage circuit, because in the diagram there are 2 stages (AND stage and OR stage).

Note 2: It is also possible to work with the CCF (Conjunctive Canonical Form) on steps 3, 4 and 5, it is more optimal if the number of 0s is less than the 1s in the output (column A), otherwise DCF is more optimal. The optimality here is in the reduction of the terms number.

Note 3: The main mechanisms for reducing a circuit are; to choose between DCF or CCF in relation to the number of 0s and 1s in the output column, The Karnaugh Table or Algebraic Simplification, and finally the don't care outputs.

7. The don't care outputs

In some situations, the output may be undefined in the specification of a circuit. For example in the 7-segment display the numbers 10 to 15 are not defined and their outputs could be indifferently put to 0 or 1 without altering the proper working of the circuit, and this characteristic could be used as an advantage to further reduce the number of gates. In the Karnaugh Table, undefined outputs could be taken either 0 or 1 depending on the situation which allows the circuit to be reduced the most.

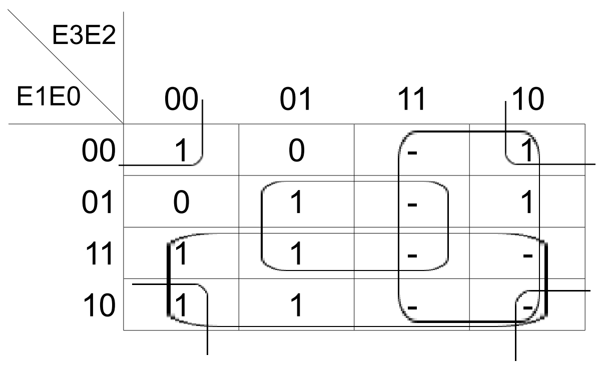
Step 2 : Truth Table

E3	E2	E1	E0	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	-	-	-	-	-	-	-
1	0	1	1	-	-	-	-	-	-	-
1	1	0	0	-	-	-	-	-	-	-
1	1	0	1	-	-	-	-	-	-	-
1	1	1	0	-	-	-	-	-	-	-
1	1	1	1	-	-	-	-	-	-	-

Step 3 : Canonical Disjunctive Functions

$$A(E3,E2,E1,E0) = \overline{E3}\cdot\overline{E2}\cdot\overline{E1}\cdot\overline{E0} + \overline{E3}\cdot\overline{E2}\cdot E1\cdot\overline{E0} + \overline{E3}\cdot\overline{E2}\cdot E1\cdot E0 + \overline{E3}\cdot E2\cdot\overline{E1}\cdot E0 + \overline{E3}\cdot E2\cdot E1\cdot\overline{E0} + \overline{E3}\cdot E2\cdot E1\cdot E0 + E3\cdot\overline{E2}\cdot\overline{E1}\cdot\overline{E0} + E3\cdot\overline{E2}\cdot\overline{E1}\cdot E0$$

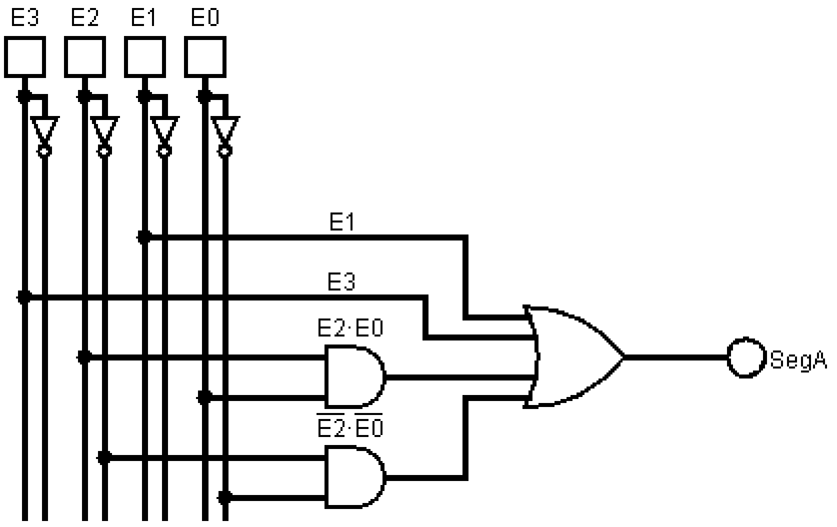
Step 4 : Karnaugh Table



E1E0 \ E3E2	00	01	11	10
00	1	0	-	1
01	0	1	-	1
11	1	1	-	-
10	1	1	-	-

$$A(E3,E2,E1,E0) = E1 + E3 + E2\cdot E0 + \overline{E2}\cdot\overline{E0}$$

<u>Step 5</u> : Schematic



Note 1: The reduction in the number of gates is very important. It makes the reducution in the cost of the circuit, make it faster, minimize the electrical consumption, and simplify the physical implementation.

Note 2: The *don't care* dashes should not be confused with *reduction* dashes, the first are put in the output side of the Truth Table, and the second are placed in the input side.

<u>Reduction dash</u> : Unlike *don't care* outputs, the *reduction dash* has no implication in the circuit specification. It is just used to reduce the size of the Truth Table, in the case where multiple adjacent lines share the same output. A reduction dash simply means, whatever value of the input variable, the output remains the same.

Example :

A	B	C	S1	S2
0	0	0	1	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

$\Leftrightarrow$

A	B	C	S1	S2
0	-	-	1	0
1	0	0	0	0
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

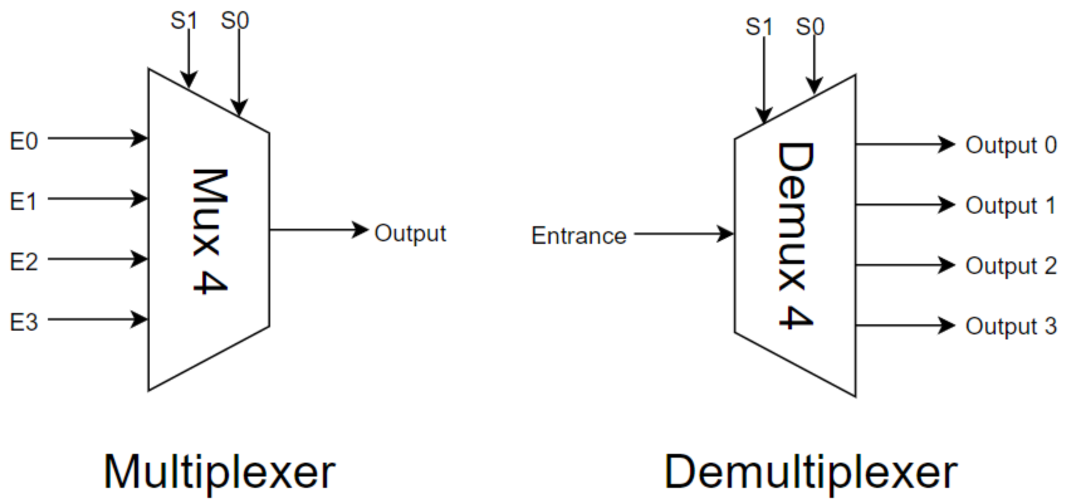
8. The most commun combinational circuits

In this section we will describe the most common combinational circuits.

8.1. Multiplexer/Demultiplexer

- ➔ The Multiplexer is a combinational circuit which has 2^n inputs and *n* selectors with a single output.
- ➔ It works by selecting one single input chosen by selectors (S1 and S0 in the example) and pass it to the output.
- ➔ The input to pass is the binary value encoded in the selectors, the other inputs are blocked.
- ➔ The Multiplexer is often designated as the Hardware *if* or *switch* instruction.

In the following diagram we have a multiplexer with 4 inputs, 2 selectors and 1 output.



- ➔ The demultiplexer does the opposite operation.
- ➔ It has a single input and 2^n outputs and *n* selectors.
- ➔ Il fait traverser l'entrée vers la sortie désignée par les sélecteurs.
- ➔ It makes cross the input value to the output chosen by the selectors.

The Multiplexer and Demultiplexer have a primordial role in the implementation of communication buses between different hardware blocks, and the management and control of the flow of information in complex hardware systems.

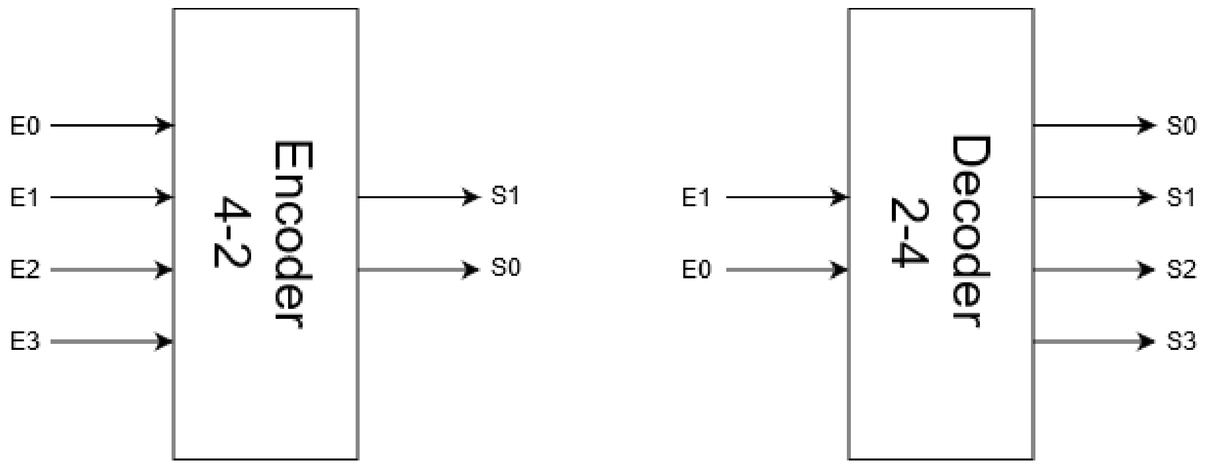
Note 1: The trapezoid symbol and the Mux/Demux nomenclature are commonly used in Hardware design literature to designate the Multiplexer/Demultiplexer.

Note 2: The number 4 at the end of the name Mux and Demux indicates the number of inputs in the Multiplexer and outputs in the Demultiplexer.

8.2. Encoder/Decoder

- ➔ The Encoder is a circuit with 2^n inputs and *n* outputs.
- ➔ The Encoder must receive only one input at 1, the others must be at 0. The output then returns the binary encoded value of the position of the input at 1.
- ➔ In the diagram below, if E2 is 1 and the others 0, the output sould be 10 (2 in binary). Because the 1 is at the entrance 2.

On the diagram n = 2 for the Encoder and for the Decoder.



- ➔ The Decoder is the opposite, it has *n* inputs and 2^n outputs.
- ➔ The operation is also the opposite, it receives an input encoded in binary, and activates only one output, the one in position with the value encoded as input.
- ➔ On the diagram, if E1=1 and E0=0 (2 in binary), S2 is at 1 and the others at 0.

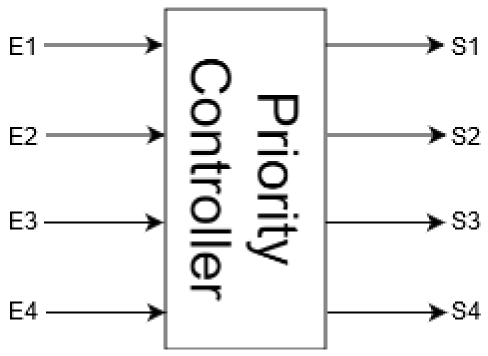
Note 1: The two numbers in the Encoder and Decoder nomenclature (4 and 2 in the diagram) are the input-output numbers of the two components.

Note 2: The decoder is an essential circuit for implementing memories (RAM/ROM).

8.3. Priority controller

- ➔ The Priority Controller is a combinational circuit that has *n* inputs and *n* outputs.
- ➔ Its role is to build a priority order on the inputs, and to prioritize the higher priority.
- ➔ It outputs only one output, the one that coincides with the highest priority among the inputs.
- ➔ In the example of the diagram, the priority scale is from E1 going down to E4. So if for example E2 and E4 are active (set to 1), the output is S2, because E2 has upper priority than E4 on the priority scale.

In the following diagram the Priority Controller is with n = 4. So 4 inputs, and 4 outputs.

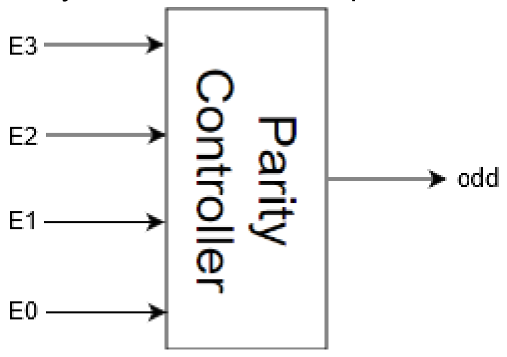


Note : This circuit can be found in an *interrupt* manager for instance.

8.4. Parity Controller

- ➔ The Parity Controller is a combinational circuit with *n* inputs and 1 output.
- ➔ It allows the recognitin of the of inputs parity. This means distinguishing whether the number of 1s in the inputs is even or odd.
- ➔ If the number of 1s in the inputs is odd it outputs 1. Otherwise, if it is even it outputs 0.
- ➔ In the example, the Parity Checker has 4 inputs, if the input is for example 1110 it will output 1 (odd) because the number of 1s is 3.

On the diagram we have a Parity Controller with 4 inputs and one output.

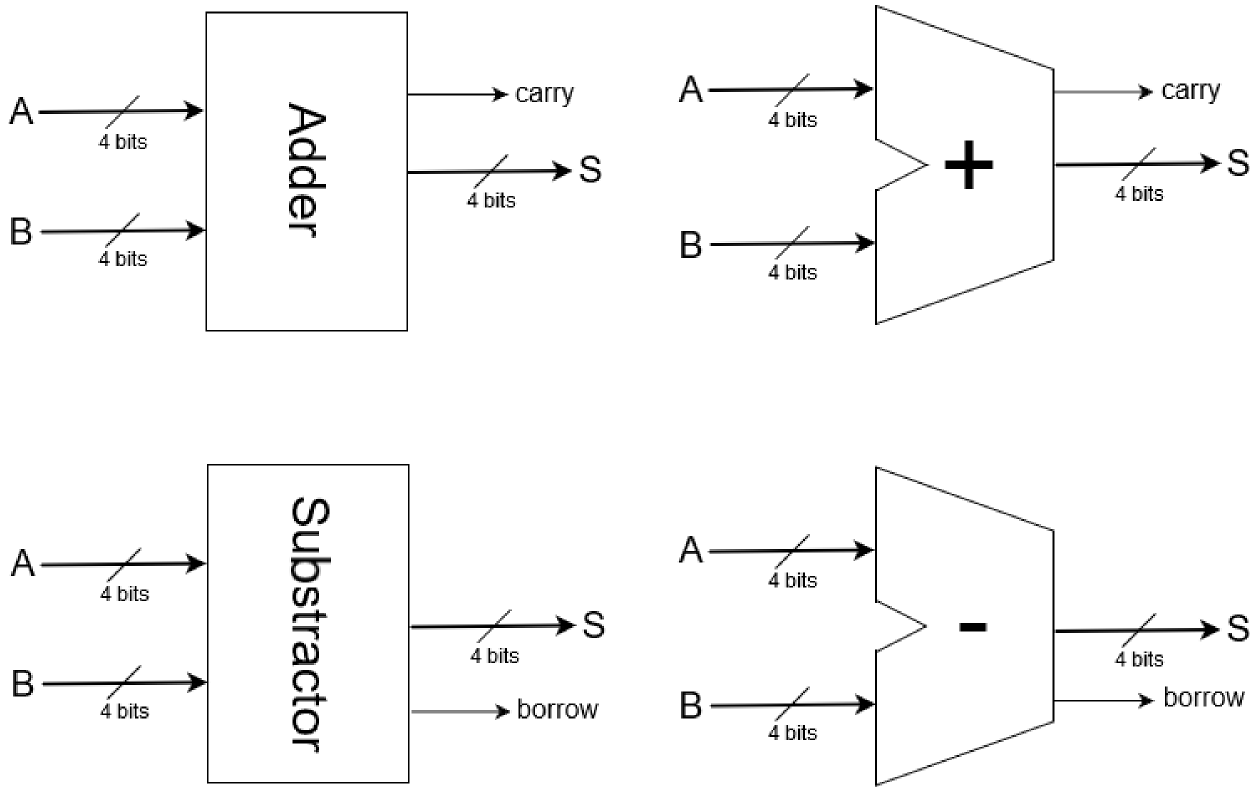


Note : The Parity Controller is often used to check the integrity of data in transmission networks.

8.5. Adder/Substrator

- ➔ The Adder is a combinatorial circuit used to do the binary addition between 2 numbers A and B on *n* bits, and gives as output the sum on *n* bits, plus the *carry* bit.
- ➔ The Subtractor is a combinatorial circuit which subtracts between 2 numbers A and B on *n* bits, and outputs the result on *n* bits, plus the *borrow* bit.

On the diagram we have a 4-bit Adder with 2 possible symbols and a 4-bit Subtractor with 2 symbols as well.



Note 1: The second representation of the Adder and the Subtractor, the one of the trapezoid with a notch to separate the 2 inputs, is very common in the Hardware design literature, it is generally used to symbolize arithmetic operations and often also the ALU.

Note 2: The slash line (/) on the inputs and the output with the label 4 bits, indicates that for instance the input A is made up of 4 different inputs (A3, A2, A1, A0). This representation is used to simplify the diagramming of buses containing several wires.

Note 3: The carry bit is the same as the one remaining last in binary addition or a borrow bit in subtraction operations. It is often called the 5th bit in 4 bit binary operation.

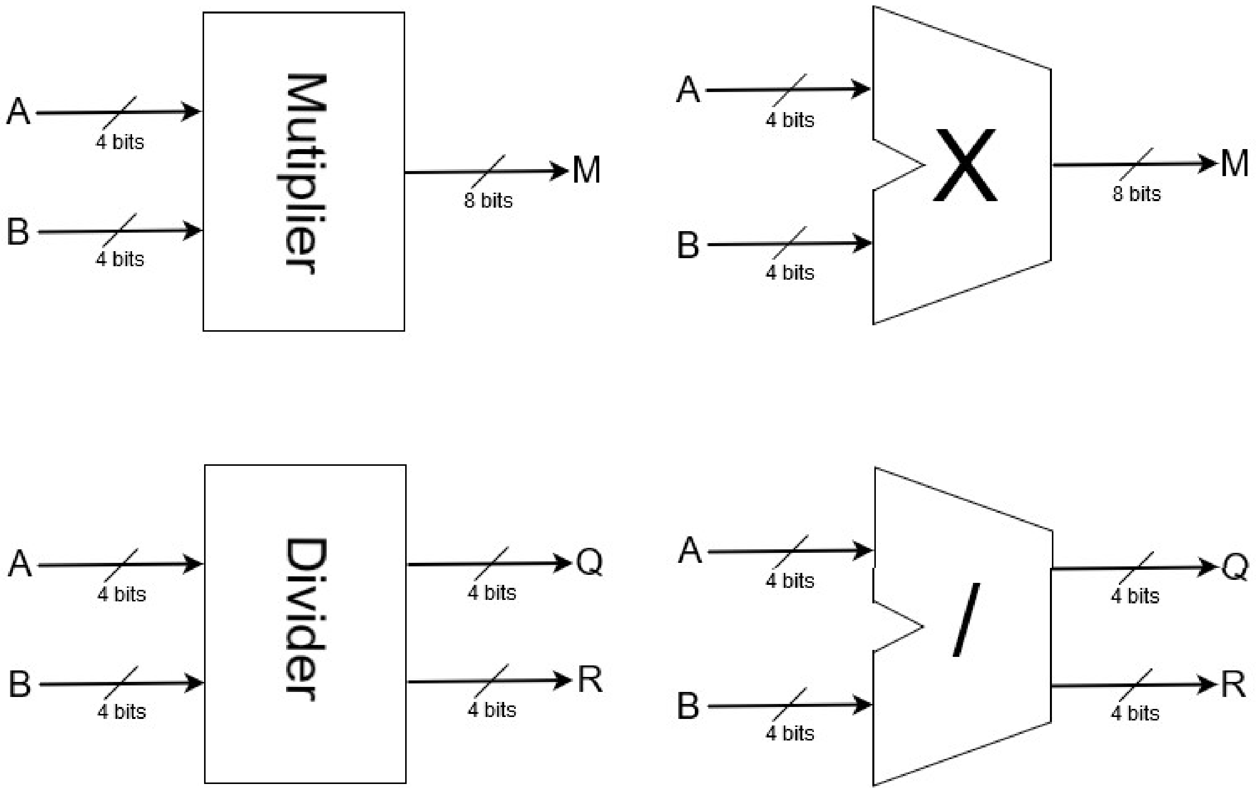
Note 4: There are different implementations of the Adder and the Subtractor depending on the encoding of the values operated. This means that there are different versions of the adder for *Unsigned*, *Sign-Magnitude*, *1's Complement*, and *2's Complement* encoding.

Note 5: You will most often find the circuits that do arithmetic operations in calculators and ALUs.

8.6. Multiplier/Divider

- The Multiplier is a combinatorial circuit that performs the binary multiplication between 2 *Unsigned Integers* A and B on *n* bits, and gives as output the result of the multiplication on *2n* bits.
- The output of the Multiplier must be 2n bits wide, to contains the result of the multiplication.
- The Divider is a combinatorial circuit that divides 2 *Unsigned Integers* A and B over *n* bits, and produces a result on 2 outputs with *n* bits each. The 2 outputs are the *Quotient* and the *Remainder*.

The diagram represents a 4-bit Multiplier with its 2 possible symbols and a 4-bit Divider with its 2 symbols as well.



Note 1: For a purpose of simplicity, the Multiplier and the Divider use *Unsigned Integer* encoding, most often a fairly simple complementary circuit is added to implement *signed* encoding.

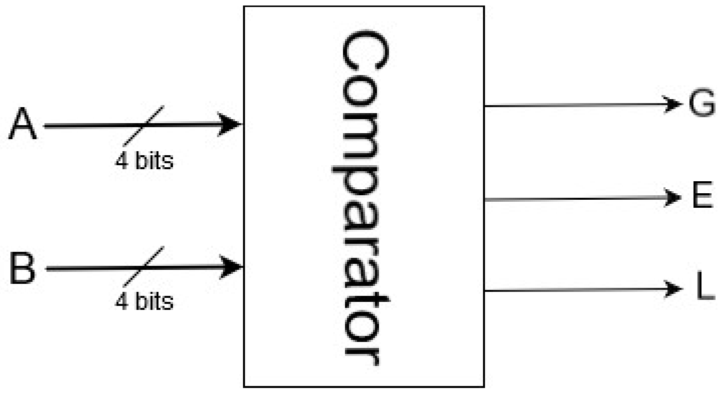
Note 2: The Divider here does the integer division, with the quotient and the remainder. Real division is done on real numbers (IEEE754 floating point encoding).

Note 3: There are also another implementation of Multiplier and Divider using sequential circuits. Each version has it own advantages and disadvantages.

8.7. Comparator

- ➔ The Comparator is a combinational circuit used to compaire between 2 numbers A and B on *n* bits. It produces 3 outputs; Greater, Equal, and Inferior.
- ➔ Whatever the 2 numbers, there are only one possible active output, Greater, Equal, or Less.

The diagram represents a Comparator with 4 bits on the inputs. And the three comparison outputs G = Greater, E = Equal, L = Lesser.



Note 1: The Comparator provides the operations >, =, and <. To obtain ≥, ≤ and ≠, logic gates could respectively be added to the outputs as follow; G+E, L+E et $\overline{E}$.

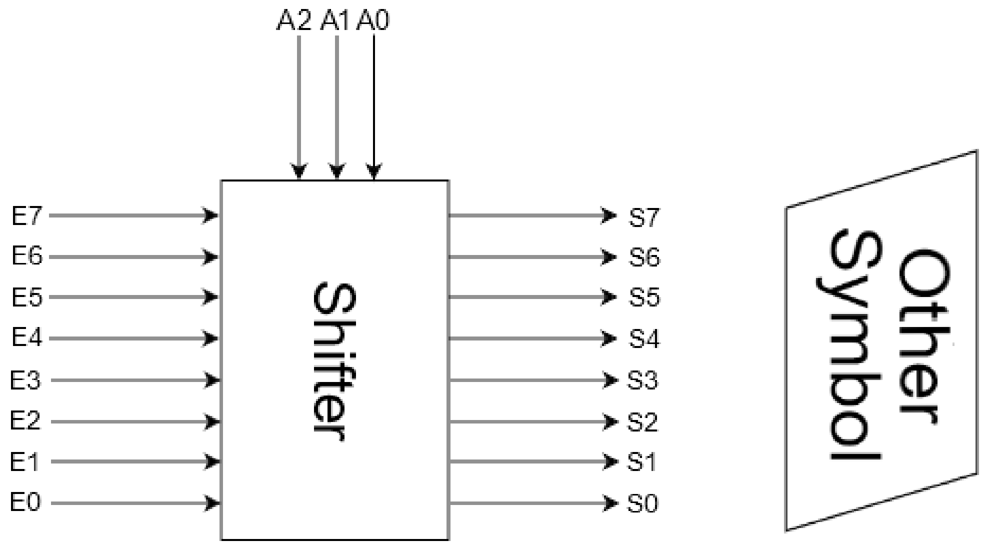
Note 2: The Comparator like the Adder and the Subtractor depends of the encoding used for the compared numbers (*Unsigned*, *Sign-Magnitude*, *1's Complement*, *2's Complement*).

Note 3: To reduce the number of logic gates and also the Hardware, it is not uncommon to find micro-architectures without a comparator. It is implemented indirectly using a subtractor. The result of the subtraction is tested if it is positive, negative, or equal to zero, and interpreted as a comparison between 2 numbers.

8.8. Shifter

- ➔ The Shifter is a combinational circuit that shifts the input bits to the left or right. The number of bits shifted is expressed binary by the *Amount* input. Amount is generally n bits when input is 2^n bits.
- ➔ If for instance the input in the diagram below is 11001100, shifting left with an Amount = 3 (011) would give the output 01100000. All bits are shifted left and the 3 rightmost bits are lost.
- ➔ In this situation we took the example of a Shifter on the left, the Shifter on the right shifts in the opposite direction, from left to right.
- ➔ The Shift Amount is encoded in binary, here 3 bits to represent an offset of 0 to 7 digit (bit). 7 being the maximum value to shift all 8 inputs.

The diagram on the left represents a Right-Shifter with 8 input bits, and 8 output bits, with an Amount of 3 bits. On the right is another representation of the Shifter, taking the form of a parallelogram.



Note 1: The left shift is interpreted mathematically as a multiplication of the input value by 2^{mnt}, and the right shift is a division by 2^{mnt} (mnt is Amount).

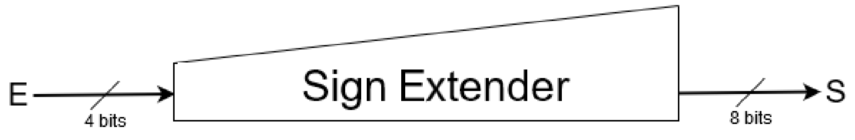
Note 2: There are several variations of the Shifter; Logical Shifter, Arithmetic Shifter, and Rotational Shifter. The Shifter studied here is a so-called Logical Shifter, in which the empty bits after shifting are always filled with 0.

Note 3: There are also an implementation of a Shifter based on Sequential Circuits. each implementation has its advantages and disadvantages.

8.9. Sign Extender

- ➔ The Sign Extender is a combinational circuit which gets a signed value on *n* bits as input. And will extend this value on *m* bits (such that m>n) as output while preserving the sign of the value.
- ➔ For example in the diagram below, the input value 3 = 0011 on 4 bits is extended to 3 = 00000011 on 8 bits. Or -3 = 1101 to -3 = 11111101 in 2's complement endoding.

The diagram represents a Sign Extender with 4 bits as inputs, and 8 bits as outputs, in 2's complement encoding.

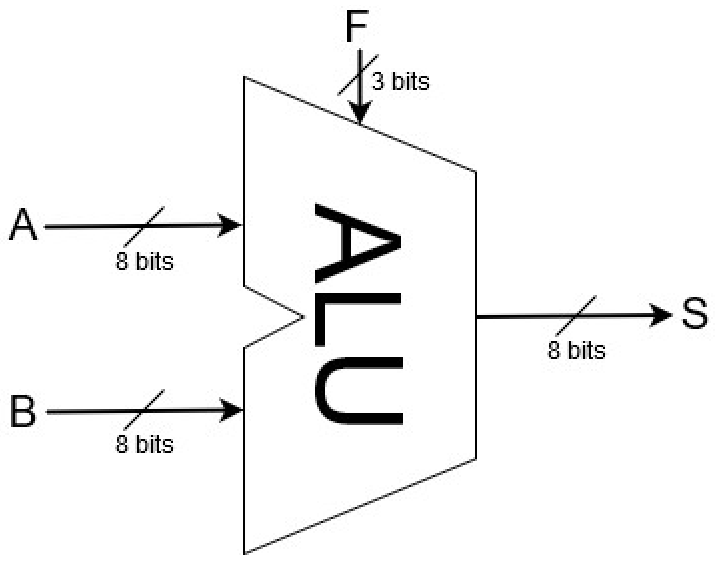


Note 1: The Sign Extender depends of the signed integers encoding (Sign-Magnitude, 1's Complement, 2's Complement).
Note 2: Extenders are often symbolized by the trapezoid geometric shape.

8.10. ALU (Arithmetic and Logic Unit)

- ➔ The ALU is the unit inside a processor that performs arithmetic and logic operations.
- ➔ The ALU is a combinational circuit that performs a Function (or Operation) on its two inputs A and B, and produces the result on the output S. The function is chosen by the input F.
- ➔ The table on the right gives the operations code in F, to select which function to perform by the ALU.
- ➔ If for instance in 2's complement we give A the value 3 = 00000011 and B the value -2 = 11111110, and we choose the function 2 on F (010) which makes A + B, the result in S would be S = 00000001.

The diagram represents an 8-bit ALU using 2's complement encoding, with 2 inputs A and B on 8 bits, an output S on 8 bits, and the function F on 3 bits.



F2	F1	F0	Functions
0	0	0	A and B
0	0	1	A or B
0	1	0	A + B
0	1	1	not used
1	0	0	A and $\overline{B}$
1	0	1	A or $\overline{B}$
1	1	0	A - B
1	1	1	SLT

Note 1: Many combinational circuits impose the choice of encoding (Unsigned, Sign-Magnitude, 1's Complement, 2's Complement), but almost all modern Hardware uses 2's Complement. Therefore from this point on, this course only deals with 2's complement encoding.

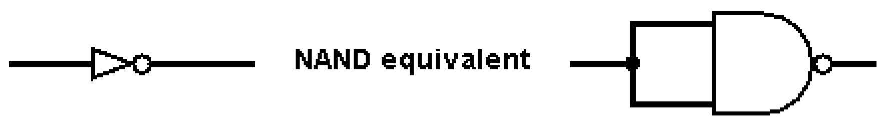
Note 2: The ALU shown in the diagram is taken directly from the book Digital Design and Computer Architecture (page: 249), it was used for the partial creation of the MIPS processor.

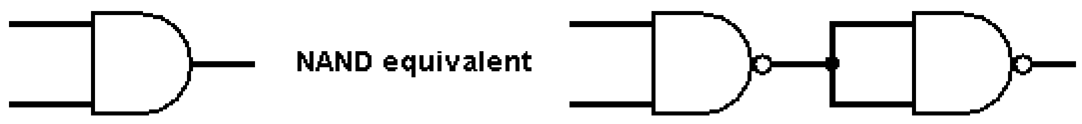
Note 3: It is possible to find the CU (Control Unit) implemented using a combinational circuit, on simple single-cycle type of processors like Arduino or PIC and AVR Microcontrollers. But in most cases, the CU is implemented using an advanced programmable sequential circuit.

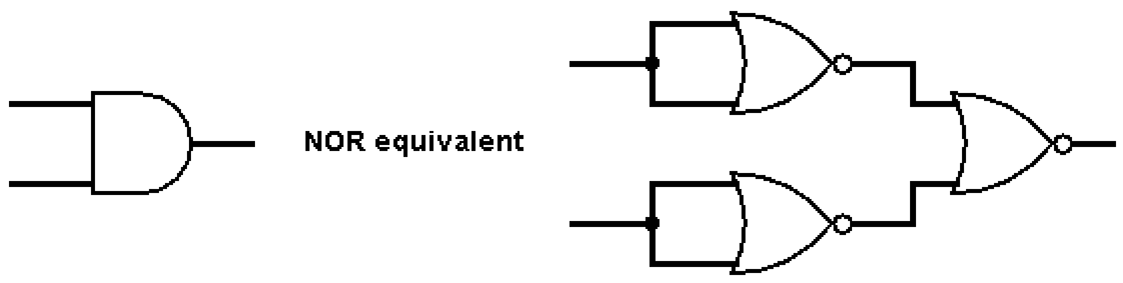
9. Universal NAND and NOR gates

- ➔ Universal NAND and NOR gates are gates that can implement any logical Boolean operation (and, or, not...).
- ➔ Any circuit can be converted to a circuit with either NAND or NOR gates.
- ➔ The implementation on silicon is favorable to NAND and NOR gates, because they require fewer transistors than other gates and electrically they are more suitable.
- ➔ A large number of integrated circuits ICs (Processors, GPU, Shipset,...etc.) are implemented using NAND or NOR on silicon.

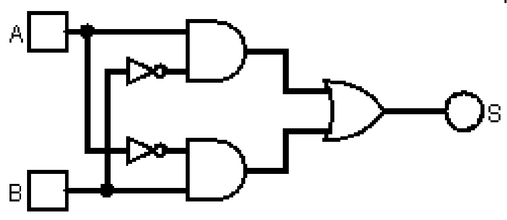
Example 1:

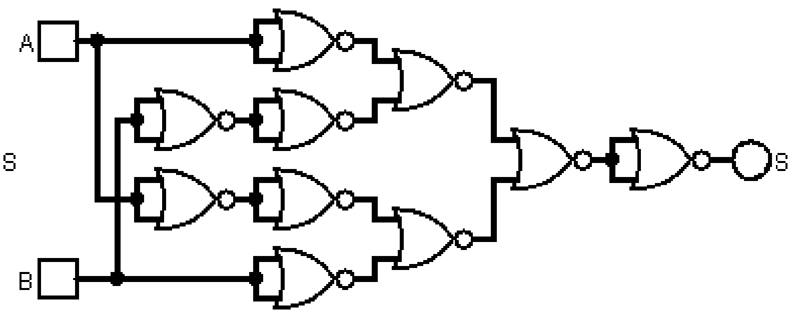






Example 2: Convert the following circuit gate by gate into NOR gates circuit.





<u>The bubble pushing technique</u> :

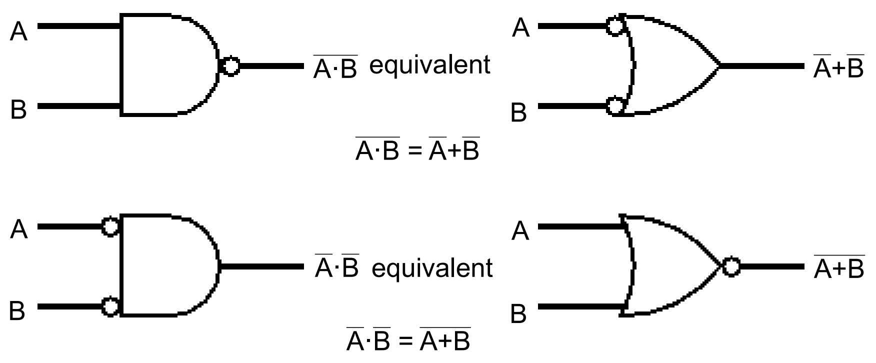
➔ It is a visual graphical technique based on De Morgane's theorem, it allows to transform any circuit into a circuit based on NAND or NOR gates with the minimum number of gates.

➔ This technique is illustrated in the diagram below $\overline{A \cdot B} = \overline{A}+\overline{B}$ et $\overline{A} \cdot \overline{B} = \overline{A+B}$.

➔ Visually, for the first formula $A \cdot B = A+B$, we could imagine that if when we push the bubble at the output inwards, it comes out on all the inputs of the gate, and the gate transforms from AND to OR.

➔ And the second $\overline{A} \cdot \overline{B} = \overline{A+B}$, if we pushed all the bubbles at the inputs, a bubble comes out at the output of the gate, and the gate transforms from AND to OR.

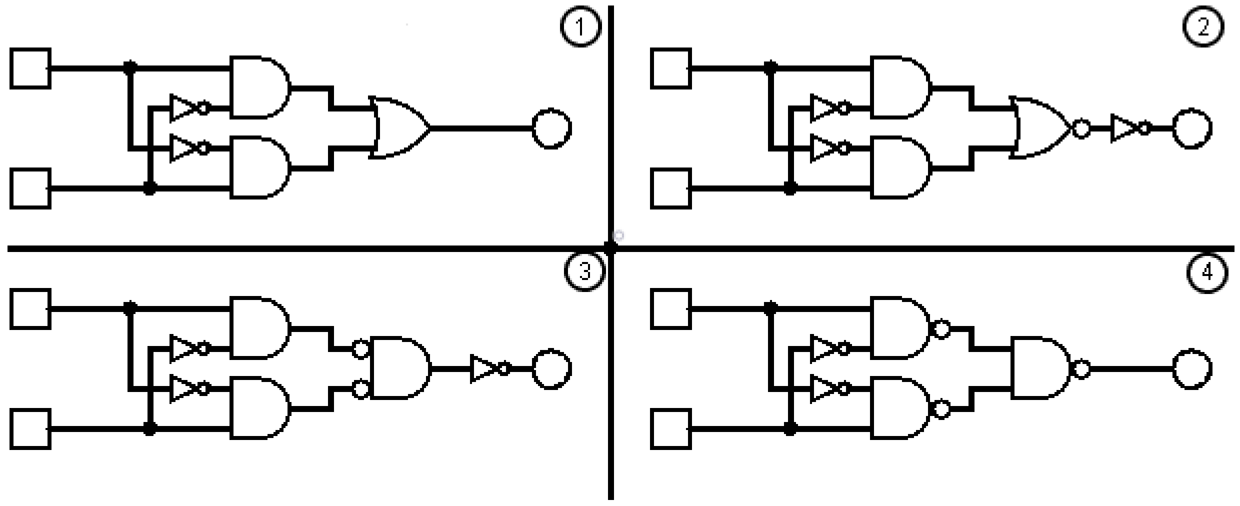
➔ Bubbles are NOTs. 2 NOT on the same wire cancel each other out according to the axiom $\overline{\overline{A}} = A$.

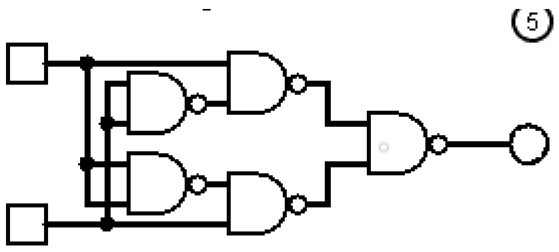

A
B
$\overline{A \cdot B}$ equivalent
A
B
$\overline{A}+\overline{B}$
$\overline{A \cdot B} = \overline{A}+\overline{B}$
A
B
$\overline{A} \cdot \overline{B}$ equivalent
A
B
$\overline{A+B}$
$\overline{A} \cdot \overline{B} = \overline{A+B}$

➔ The global rule is that if we push the bubbles on <u>all</u> the inputs, a bubble will come out at the output and the gate transforms from AND to OR or vice versa.

➔ And the other way, if we push the output bubble inside the gate, a bubble will emerge on each input and the gate transforms from AND to OR or vice versa.

Example :

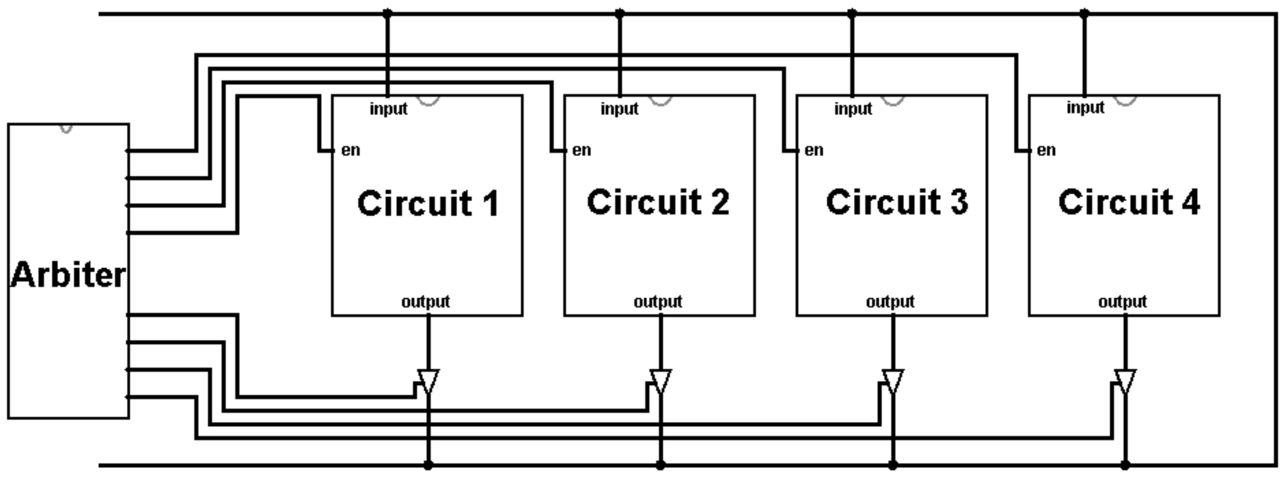




10. The Z value and its use in Buses

- ➔ The Z value or Z state is the third logical state counting state 0 and state 1.
- ➔ The Z value is sometimes called a Floating value or electronically a High Impedance value.
- ➔ A signal (wire) with a Z value means that the wire is not connected, neither at 0 nor at 1.
- ➔ The only gate that can generate the Z value is the Tristate Buffer. If the Tristate Buffer is set to 0 on its control input it disconnects the output wire.
- ➔ Electrically, if a wire is not connected its voltage is floating between different voltages. Experimentally it is a totally random voltage.
- ➔ The purpose of the Z value is a mean to get around the 4th rule of the definition of combinational circuits, such that several outputs can share the same wire.
- ➔ Its main use is to share a common wire for all outputs, in order to transport information. The shared wire is usually called Bus.

The following diagram represents a Bus used for data transmission between 4 circuits. The transmission is managed by the Bus Arbiter.



- ➔ Circuits could be combinational or sequential.
- ➔ The Arbitrator is the circuit responsible for managing the transfer of data between circuits on the Bus.
- ➔ Each output of a circuit is controlled by a Tristate Buffer, itself controlled by the

Arbiter so that only a single output can access the Bus.

- ➔ All circuits receive the Bus signal on their inputs, but the signal can only enter the circuit if the circuit's enable *en* input is active (set to 1).
- ➔ By controlling the *enable* of the circuits, the Arbitrator controls the circuit which must take the value on the Bus.
- ➔ If for example Circuit 2 wants to transmit to Circuit 4. The Arbiter must activate the Tristate Buffer of the output of Circuit 2 and activate the *enable* of circuit 4.

Note 1: The most famous mistake among novices in digital electronics is to think that if a wire is not connected, its voltage must be at 0 volts, therefore it is the logical value 0. But as a reminder, there is a difference between the voltage and the amperage, it is true that when a wire is not connected there is no current passing, but for digital electronics the signal is transmitted using voltage, not current.

Note 2: What imply from the previous note, is that the logic value 0 must be electrically connected to the (-) terminal (also called the ground or <u>GND</u>) and the logic 1 must be connected to the (+) terminal (called also <u>Vcc</u>).

Note 3: For a simplicity purpose, the previous illustration of the Bus is made upon a single wire. But in general, in a real world circuit, a Bus has several wires to transport information on 8, 16, 32 bits...etc.

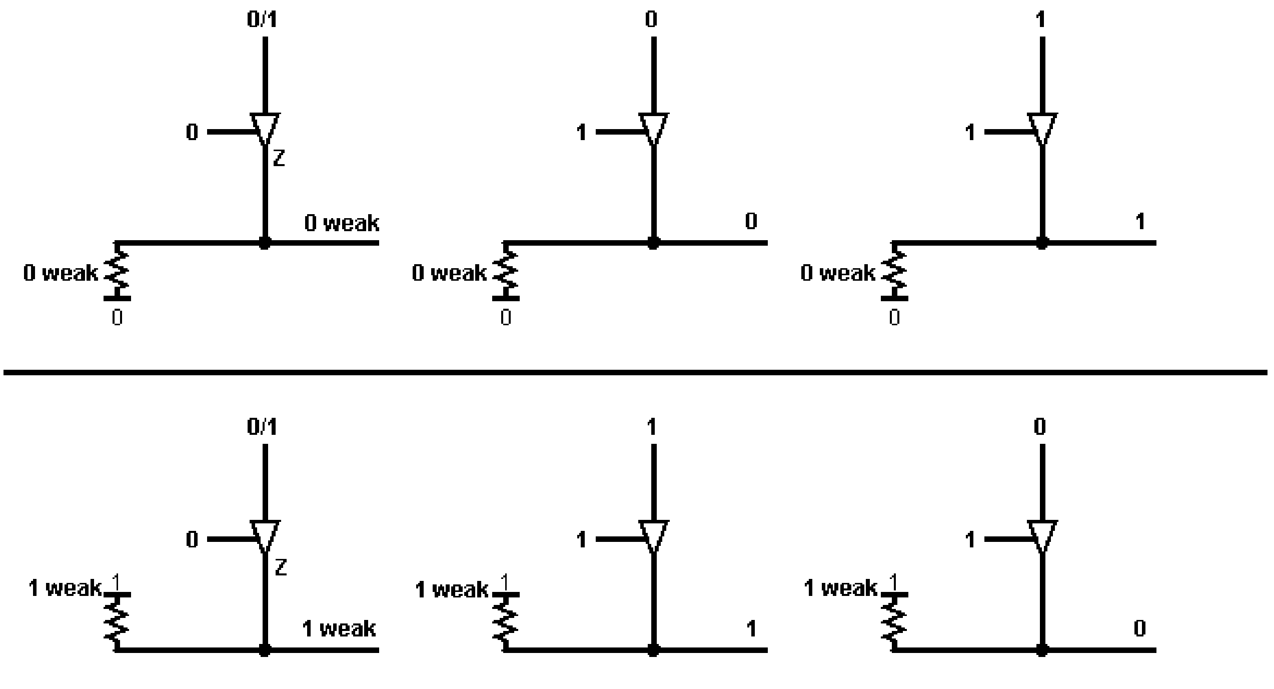
Note 4: The use of the Z state-based bus is most often reserved for simple systems or older systems. For current or more complex systems, Multiplexers/Demultiplexers are generally more appropriate, or even the use of point-to-point connections between pairs of circuits for more efficiency in high end systems.

Note 5: It is possible to find the Arbiter as a separate circuit, as it is possible to find it integrated inside the Processor, and sometimes it is simply implemented by a decoder. In modern motherboards, we can say that the 2 North-Bridge and South-Bridge chipsets have some Bus Arbitration functions.

<u>Weak states 0 and 1</u> :

- ➔ The Floating value can have values like 1 Volt, 2.5 Volts, 3.8 Volts, and those values are prohibited in digital binary systems.
- ➔ This why for some circuits, using Z as an input value cause a problem, and can lead to a malfunction in the circuit.
- ➔ In addition, not having a stable input value is a bad way to design digital circuits.
- ➔ This is why for all these reasons, hardware designers use a 4th and a 5th state after 0, 1 and Z, referred as 0-*weak* and 1-*weak*.
- ➔ The 0-*weak* and the 1-*weak* are fixed <u>constants</u> and cannot be changed like the other signals (0,1 and Z).

➔ They are generally used to override (replace) the Z value in an empty bus (does not transmit) and provide a stable <u>input</u> to circuits.



➔ The top 3 Buses are connected to the constant value 0-*weak*, and the bottom 3 Buses are connected to the constant value 1-*weak*.

➔ To provide a low value on a wire, simply connect a resistor to the 0 Volt (or GND) terminal for the 0-*weak* constant, and a 5 Volt (or Vcc) resistor for the 1-weak constant.

➔ In the two buses on the left the Tristate Buffer controller is at 0 and outputs Z. In this case the Z value is overridden by 0-*weak* in the bus at the top, and by 1-*weak* at the bottom.

➔ On the two buses in the middle there is no conflict, since for the one at the top the Tristate Buffer passes the value 0, and the bus contains 0-*weak*, and vice versa at the bottom, the Tristate Buffer passes the value 1, and the bus contains 1-weak.

➔ For the two buses on the right, there is a conflict on both. At the top, the Tristate Buffer passes the value 1 and the bus contains a 0-*weak*. In this case it is the 1 which occupies the bus. And the same thing at the bottom, the Tristate Buffer passes the value 0 and the bus contains a 1-*weak*, it is the 0 which occupies the bus.

➔ The rule is that if a 0-*weak* or 1-*weak* conflicts with state Z, 0-*weak* or 1-*weak* wins. In the other hand, if 0-*weak* or 1-*weak* conflicts with 1 or 0, 1 or 0 wins.

➔ The relationship of domination is : (0Λ1) > (0-*weak* Λ1-*weak*) > Z. (Λ is an or, > is the direction of domination).

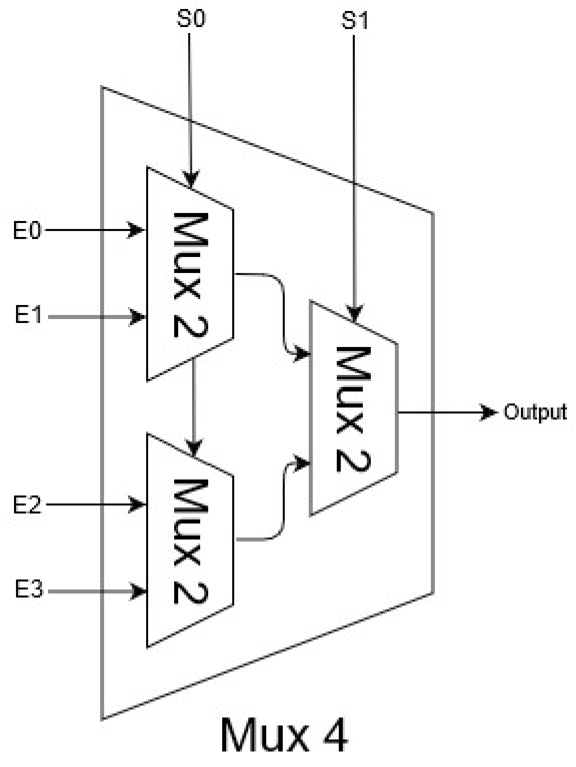
Remarque 1: The resistor connected to the Vcc to provide the 1-*weak* is called PullUp resistor, by convention it must be shown facing upwards on the diagrams. And conversely, the resistor connected to the GND to provide the 0-*weak* is called PullDown resistor, and by convention on the diagrams should be represented oriented downward. In real world, the resistance can have a value between 1kΩ-10kΩ.

Remarque 2: Understanding why adding a resistor produces a weak signal that cancels in conflict with a normal signal falls into the realm of analog electronics, which will not be explained here. Despite thas, its operation remains relatively simple and it is possible for a student with his knowledge to guess it himself.

11. Hardware design by slicing

- ➔ All complex technologies use what is called composition design, modularity, hierarchy, in their design.
- ➔ The concept was described in the first rule of constructing a combinational circuit, that is, the combinational circuit consists of several small-sized combinational circuits, which themselves consist of other even smaller circuits and so on, up to the logic gates.
- ➔ It is the same concept of fragmenting a large program into several functions in Software programming.
- ➔ But actually, the slice composition consists of constructing a circuit with small components of the same type of circuit itself.
- ➔ For example, the construction of a 4-input Multiplexer with a bunch of small 2-input Multiplexers.
- ➔ This way of design is very practical and very flexible for the easy construction of circuits with varied sizes. Unfortunately, this technique is only applicable for some specific circuits and not all circuits.

Example 1: The construction of Multiplexer 4 from Multiplexers 2.



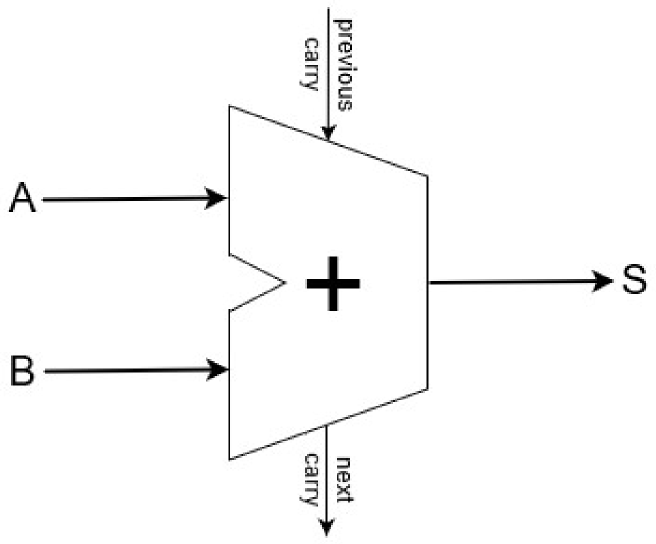
Example 2: The construction of a 4-bit *2's Complement*/*Unsigned Integer* adder from a 1-bit adder.

$$\begin{array}{rcccc} & & 1 & 1 & \\ & 0 & 0 & 1 & 1 \\ + & 0 & 0 & 1 & 1 \\ \hline = & 0 & 1 & 1 & 0 \end{array}$$

If we observe the 2's Complement/Unsigned addition operation on a single bit, for example on the 2nd bit (2nd column in the diagram), we can construct a 1-bit adder which must have 3 inputs which are the 2nd digit in A, the 2nd digit in B, and the addition carry from the previous bit (the 1st bit). Producing 2 outputs, which are the Sum and the addition carry for the next bit (for the 3rd bit).

Note : The adder for *2's Complement* and the adder for *Unsigned Integer* are actually the same adder, they are interchangeable and produce the correct result for both types of encoding. On the other hand, the adders for *Sign-Magnitude* and *1's Complement* work differently, and are not intercompatible.

Step 1 : Global scheme



Step 2 : Truth Table

A	**B**	**P_c**	**S**	**N_c**
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

P_c : previous carry
N_c : next carry

Step 3 : Disjunctive Canonical Functions

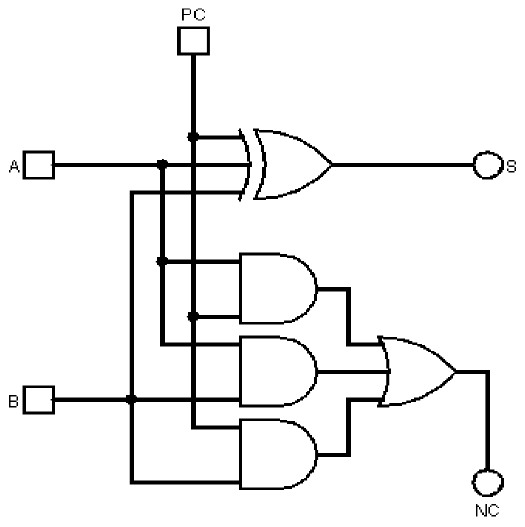
$S(A,B,P_c) = \overline{A}\cdot\overline{B}\cdot P_c + \overline{A}\cdot B\cdot\overline{P}_c + A\cdot\overline{B}\cdot\overline{P}_c + A\cdot B\cdot P_c$
$N_c(A,B,P_c) = \overline{A}\cdot B\cdot P_c + A\cdot\overline{B}\cdot P_c + A\cdot B\cdot\overline{P}_c + A\cdot B\cdot P_c$

Step 4 : Algebraic Minimization

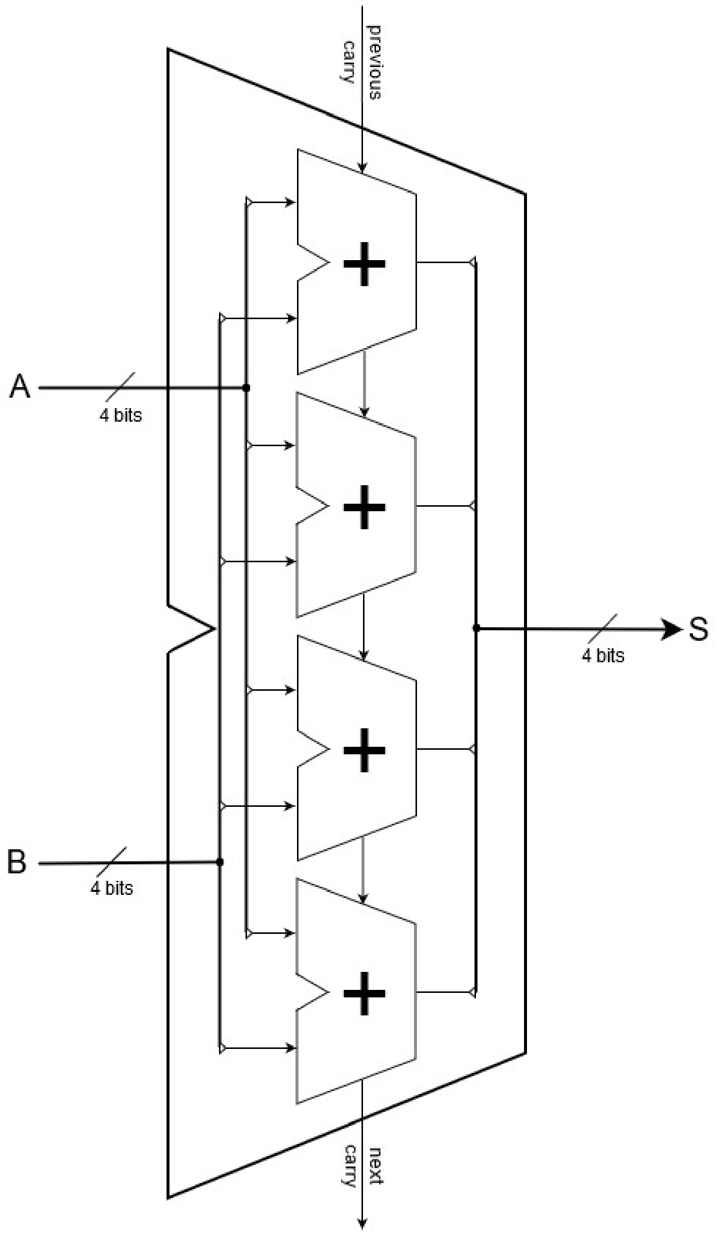
$S(A,B,P_c) = \overline{A}\cdot\overline{B}\cdot P_c + \overline{A}\cdot B\cdot\overline{P}_c + A\cdot\overline{B}\cdot\overline{P}_c + A\cdot B\cdot P_c$
$S(A,B,P_c) = \overline{A}\cdot(\overline{B}\cdot P_c + B\cdot\overline{P}_c)+ A\cdot(\overline{B}\cdot\overline{P}_c + B\cdot P_c)$
$S(A,B,P_c) = \overline{A}\cdot(B\oplus P_c)+ A\cdot(B\otimes P_c)$
$S(A,B,P_c) = \overline{A}\cdot(B\oplus P_c)+ A\cdot(\overline{B\oplus P_c})$
$S(A,B,P_c) = A\oplus B\oplus P_c$

$N_c(A,B,P_c) = \overline{A}\cdot B\cdot P_c + A\cdot\overline{B}\cdot P_c + A\cdot B\cdot\overline{P_c} + A\cdot B\cdot P_c$
$N_c(A,B,P_c) = \overline{A}\cdot B\cdot P_c + A\cdot\overline{B}\cdot P_c + A\cdot B\cdot\overline{P_c} + A\cdot B\cdot P_c + A\cdot B\cdot P_c + A\cdot B\cdot P_c$
$N_c(A,B,P_c) = (\overline{A}\cdot B\cdot P_c + A\cdot B\cdot P_c) + (A\cdot\overline{B}\cdot P_c + A\cdot B\cdot P_c) + (A\cdot B\cdot\overline{P_c} + A\cdot B\cdot P_c)$
$N_c(A,B,P_c) = (B\cdot P_c) + (A\cdot P_c) + (A\cdot B)$

<u>Step 5</u> : Schematics



<u>Logic diagram of 4-bit adder using slicing</u>



In the sliced adder diagram, each 1-bit adder represents a column (1 bit) on the addition operation in the first diagram. We can also observe in the current diagram how the next carry of an output adder is passed as input to the previous carry for the next bit.

Note 1 : The small triangle on the 4-bit buses A, B, and S are indicators to indiate that only one wire of the 4 wires has been took apart from the bus.

Note 2 : The 4-bit adder shown here is different from the one shown in section 8.5, this last didn't have *previous carry* input. The advantage of the adder presented here is that it is suited for slicing methods, so we can combine two 4-bit adders to get an 8-bit adder, and so on to have adders on 12 bits, 16 bits, 32 bits...etc.

Note 3: The condition for the 4-bit adder to work properly is that the previous carry input of the first adder must be set to 0, the first bit has no previous carry.

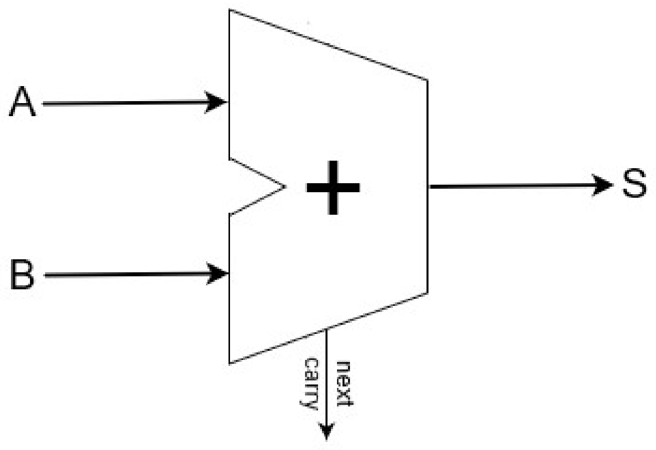
Note 4: The main reason for the success of 2's Complement encoding compared to other signed encodings, is that with 2's Complement encoding the Hardware does not need a subtractor, the subtraction is performed by the addition of A with (-B). Unlike Sign-Magnitude encoding which requires a dedicated subtractor. For 1's Complement, the subtractor can also be replaced by an adder, but there are situations where it must do the addition twice.

Note 5: The 1-bit adder that we have just seen is called Full-Adder, there is also another type of 1-bit adder, called Half-Adder, it will be presented in the following example.

Example : Falf-Adder

A half-adder is a 1-bit adder that has no previous carry input, it does the addition without previous carry.

<u>Step 1</u> : Global scheme



Step 2 : Truth Table

A	**B**	**S**	**N_c**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

N_c : next carry

Step 3 : Disjunctive Canonical Functions

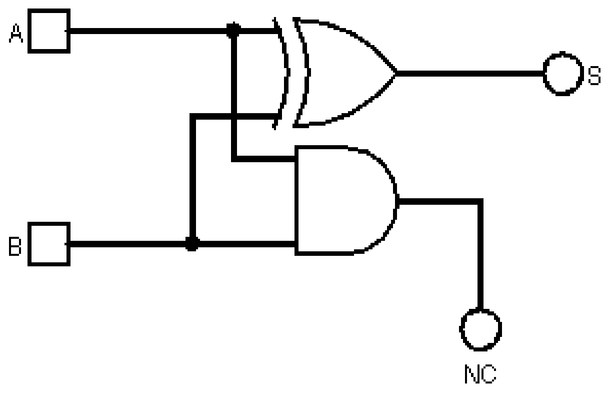
$S(A,B) = \overline{A}\cdot B + A\cdot\overline{B}$
$N_c(A,B) = A\cdot B$

Step 4 : Algebraic Minimization

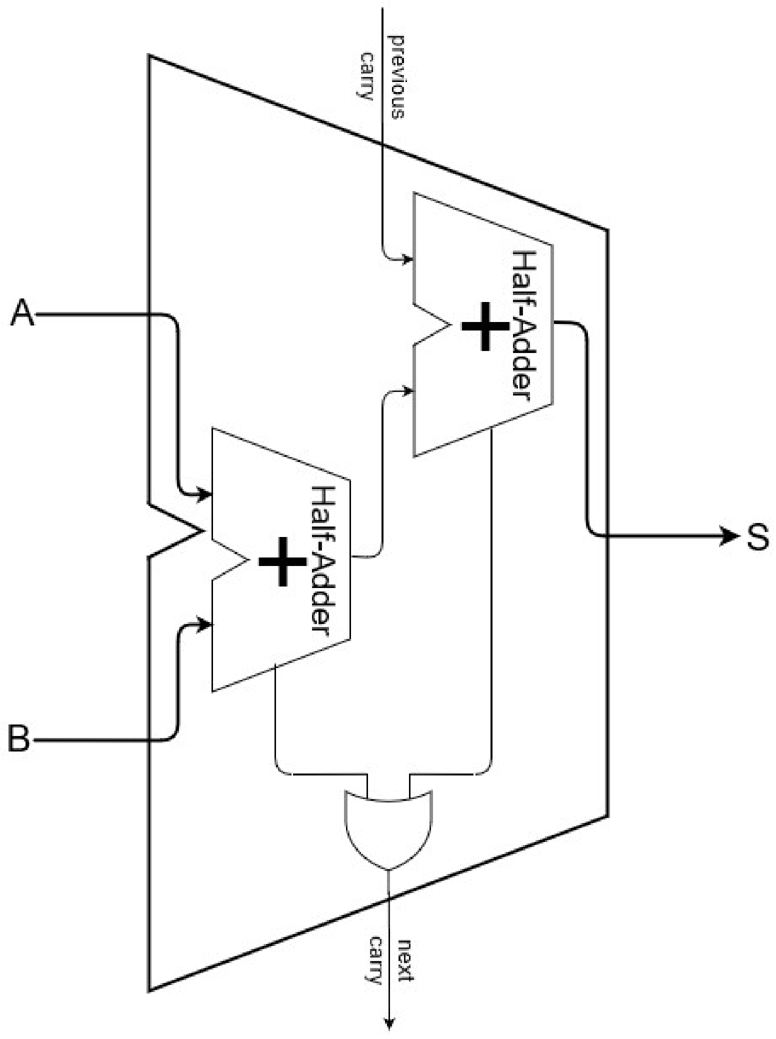
$S(A,B) = \overline{A}\cdot B + A\cdot\overline{B}$
$S(A,B) = A\oplus B$

$N_c(A,B) = A\cdot B$

Step 5 : Schematics



It is possible to construct a Full-Adder from 2 Half-Adders as in the following diagram, hence the name Half-Adder.



Full-Adder

Note 1 : It is also possible to construct multi-bit adders with only half-adders.

Note 2 : The term Pin refer to digital input or output, its origin comes from the domain of electronics where physically it represents the connector of an integrated circuit (IC) soldered on a printed circuit (called PCB).