

# LOGICA BOOLEANA "Aplicaciones para los microcontroladores"

## Objetivo:

Conocer la lógica booleana, de una manera creativa y relacionarla con los circuitos lógicos digitales y señales binarias.

## Contenido Teórico.

La lógica booleana parte de manejar variables que toman dos valores discretos y con operaciones que asumen resultado lógico.

Los dos valores que asumen las variables son llamadas de diferentes formas (por ejemplo, *verdadero y falso, si y no, etc.*). Para este propósito debemos pensar en términos de bits y asignar los valores de 1 y 0. La lógica booleana se usa para describir, de una forma matemática el procesamiento y manipulación de la información binaria.

## 1 CONCEPTOS INTRODUCTORIOS

La lógica booleana se basa en variables binarias y operaciones lógicas. Las variables se identifican mediante las letras como A, B, C, x, y, z, etc. las cuales solo pueden tener dos valores que son 1 ó 0. En cuanto a operaciones lógicas hay tres operaciones básicas: AND, OR y NOT

### Operación Lógica AND

Se puede representar con diferentes simbolos como el punto ( $A * B$ ), andpersand ( $A \& B$ ), o por la ausencia de un operador ( $AB$ ).

Por ejemplo  $Z = A \& B$ , quiere decir que  $Z = 1$  solo si,  $A = 1$  y  $B = 1$ , de caso contrario  $Z = 0$ .

### Operación Lógica OR

Se puede representar con diferentes simbolos como el más ( $A + B$ ), con la letra griega lora ( $A | B$ ).

Por ejemplo  $Z = A + B$ , quiere decir que  $Z = 1$  cuando  $A = 1$  ó  $B = 1$ , de caso contrario  $Z = 0$ .

### Operación Lógica NOT

Se puede representar con diferentes apóstrofes como la tilde ( $A'$ ), ó con una línea horizontal encima ( $\bar{A}$ )

Por ejemplo  $Z = \bar{A}$ , quiere decir que  $Z = 1$  cuando  $A = 0$  y  $Z = 0$  cuando  $A = 1$ .

Podemos observar que la simbología utilizada para las operaciones AND y OR son conforme a su operación de multiplicación y suma, pero no se debe confundir la aritmética con la lógica pues un ejemplo es:

(Uno OR Uno es Uno)

$1 + 1 = 1$  según la Lógica Booleana

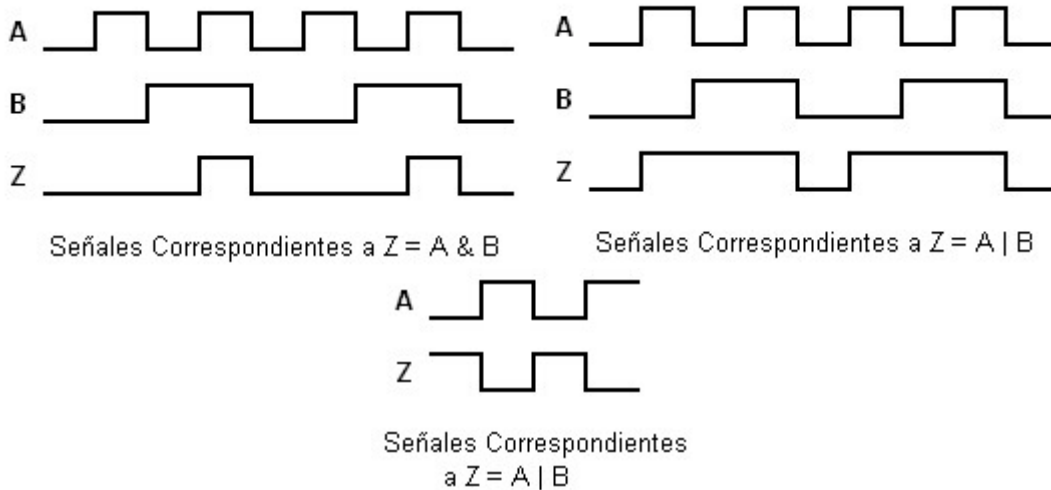
(Uno más Uno es Dos)

$1 + 1 = 10$  según la Aritmetica Booleana

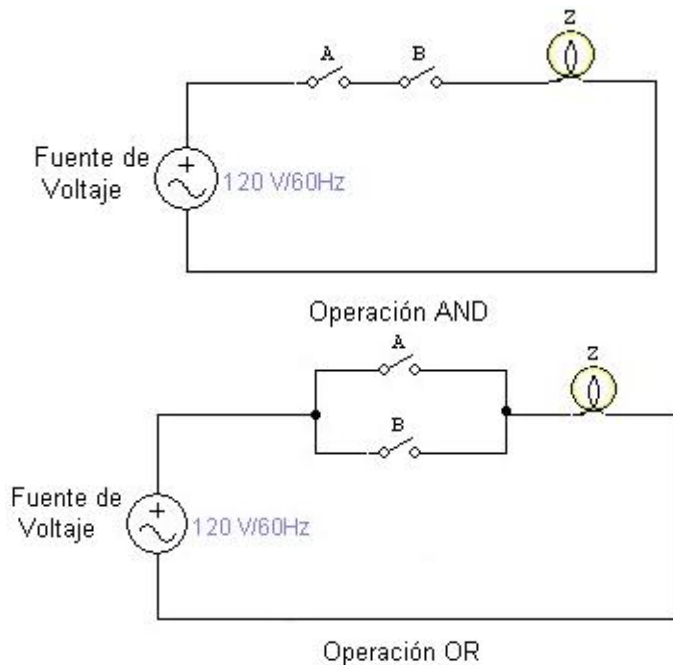
## 2 SEÑALES Y PULSOS

Se puede representar la lógica booleana de maneras más concretas para el mundo, por ejemplo simplificar el 1 lógico como una señal de 5 voltios, y el 0 lógico como 0 voltios.

de esta manera podríamos representar en señales eléctricas un operador lógico.

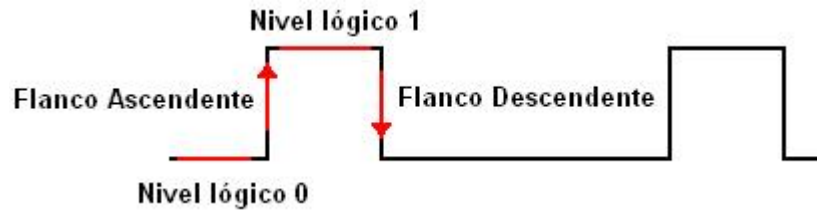


La lógica booleana parte de circuitos de conmutación, partiendo que A y B fuesen interruptores los cuales solo pueden tener dos estados 0 cuando está abierto y 1 cuando está cerrado, ahora bien la salida sea el foco de luz que se acciona con los interruptores sea Z.



## CARACTERÍSTICAS DE LAS SEÑALES.

La señal digital posee tres características mínimas que se utilizan en diferentes casos, estas características que son: flanco ascendente, nivel y flanco descendente se muestran en la figura.



### Nivel Lógico

Es un nivel constante de voltaje representando así un 1 ó un 0 lógico, no poseen cambios aparte de estas dos opciones lo que permite identificar un valor falso o verdadero.

### Flanco Ascendente

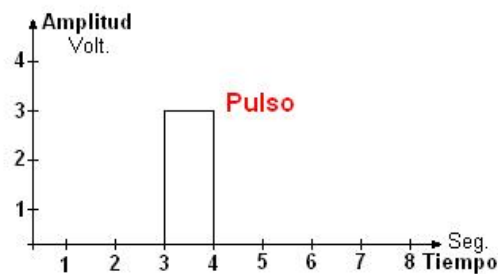
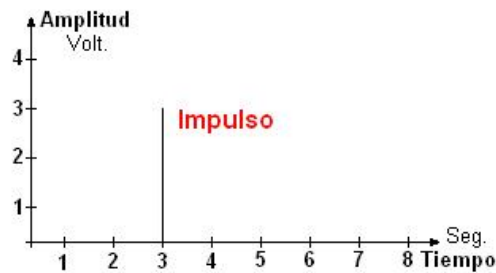
Cuando un nivel pasa de un estado de 0 a 1 en un instante de tiempo se le conoce con este nombre.

### Flanco Descendente

Cuando un nivel pasa de un estado de 1 a 0 en un instante de tiempo se le conoce con este nombre.

## PULSO E IMPULSO

En Teoría un impulso es un valor determinado en un instante de tiempo instantáneo, y el pulso es un valor determinado en una cantidad de tiempo determinada.



Para aplicarlo en un microcontrolador, bajo la premisa de una máquina determinística, se tiene que un impulso dura un ciclo de máquina, y un impulso más de uno ciclo de máquina.

## 3 ENTRADAS Y SALIDAS DIGITALES

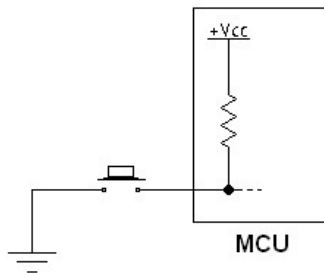
### 3.1 Entradas

#### 3.1.1 No optoacopladas

##### *Pull-ups*

El pull-up mantiene la línea en estado de 1 mientras no se active el pulsador, cuando este se activa, baja la línea a un 0 lógico.

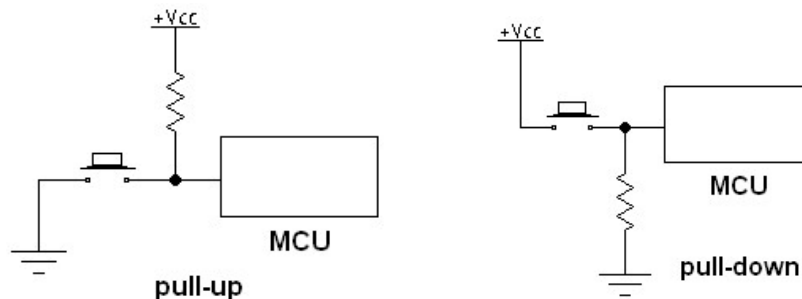
##### *Internos*



Pueden activarse pull-ups internos del microcontrolador teniendo en cuenta que según la casa matriz los valores de resistencia cambian, por ejemplo para microchip es de 33K en cambio para motorola es de 56K.

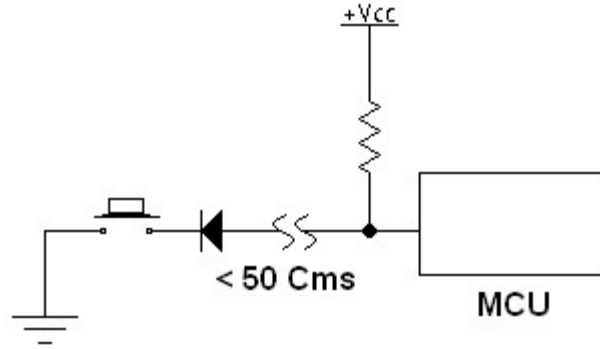
##### *Externos*

Se utilizan por dos razones, uno cuando no hay internos y dos cuando hay ambientes muy ruidosos porque R permite un mayor drenaje de corriente.

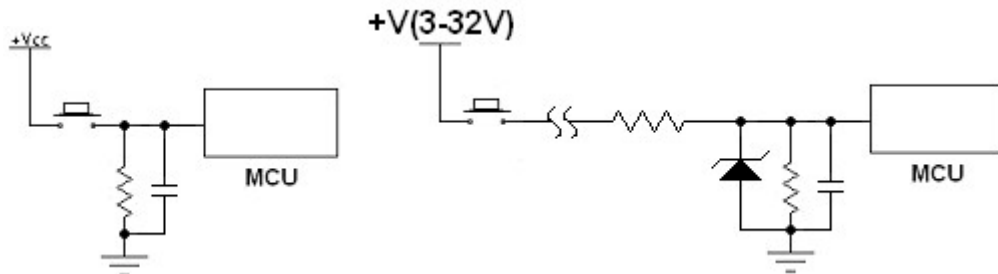


El pull-down mantiene la línea en 0 lógico, a menos que se active el pulsador esta pasará a 1 lógico.

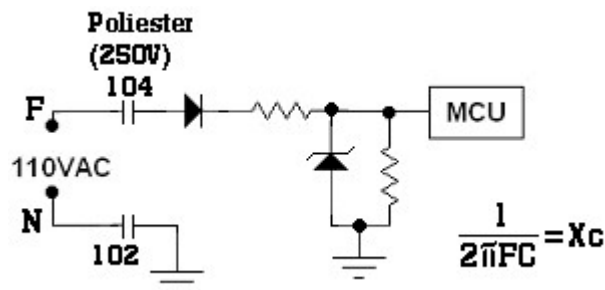
Cuando el pulsador está a distancias mayores de 50 centímetros es conveniente colocar un diodo que impide ruidos y cortos.



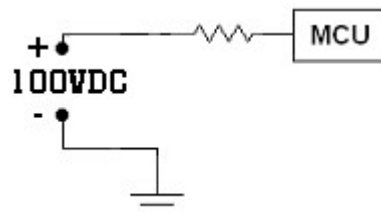
Los condensadores filtran parte del ruido en el medio y aumentan la calidad de la señal.



Las entradas de 110VAC pueden ser un problema debido a la diferencia de voltajes de manejo en un MCU y un red eléctrica, por eso los circuitos que acoplen esta señal como entrada al MCU deben tener cierto margen de protección.



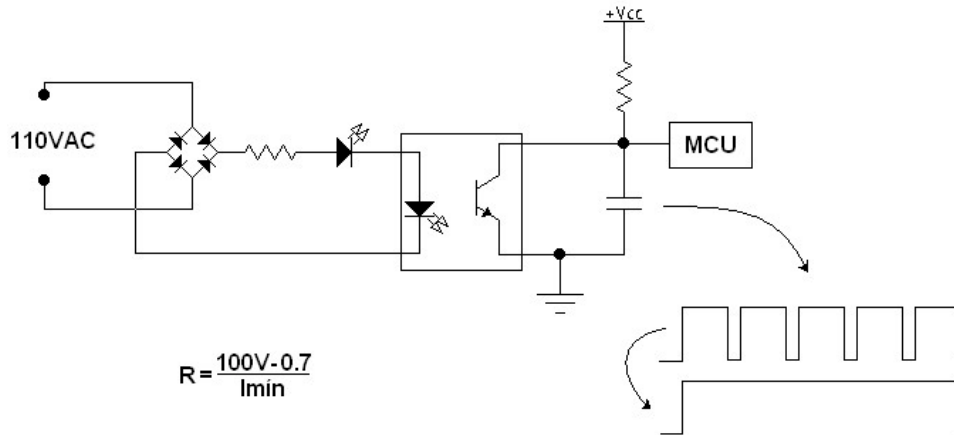
El mismo inconveniente sigue con voltajes DC pero de magnitudes altas, en este caso pueden existir formas más sencillas de acoplar al MCU.



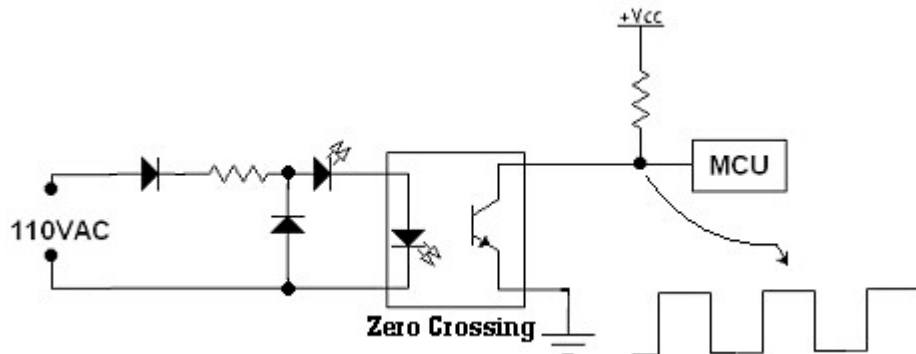
En el caso anterior la resistencia debe ser de gran magnitud (5M oprox), para que impida considerablemente el flujo de la corriente.

### 3.1.2 Optoacopladas

El doble sentido de la corriente en una entrada AC, hace que el circuito deba solucionar este problema para ser interpretado por el microcontrolador como un 1 o 0 lógico.

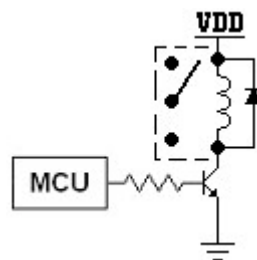


Una entrada a 110VAC optoacoplada, impide el paso de ruido proveniente de otros aparatos conectados a la red eléctrica como motores, contactores, y todo tipo de carga inductiva que produzca rebotes al conmutar.

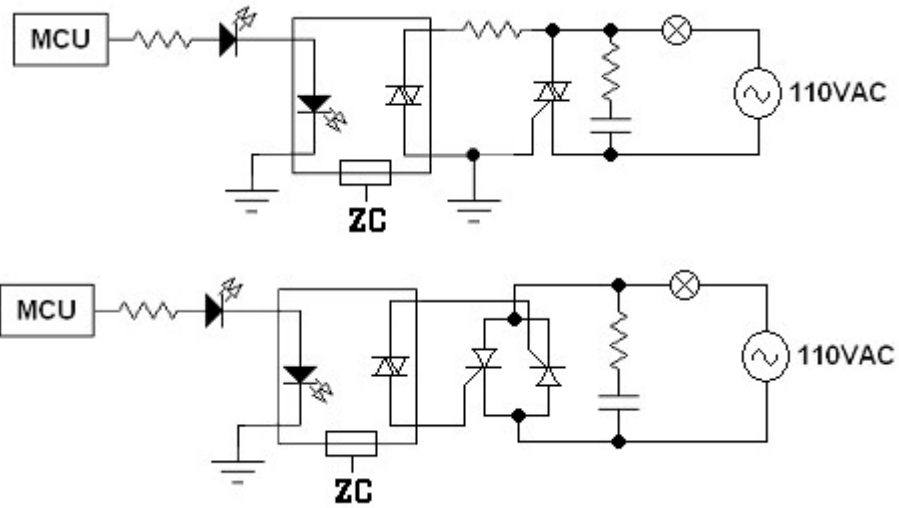


### 3.2 Salidas

Una salida de uso típico es la de relevo, esta se caracteriza por que tiene un diodo de protección, en la descarga de corriente de la bobina.

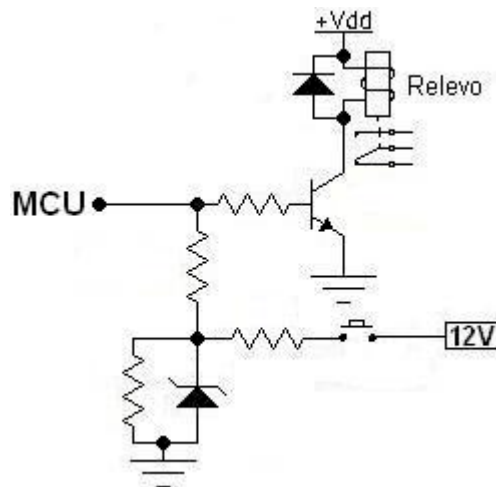


El manejo de cargas de tipo inductivo u otros pueden ocasionar ruidos en su conmutación, y una buena forma de solucionar este problema es optoacoplar la salida.



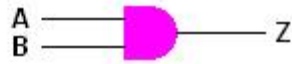
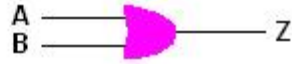

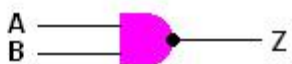




### 3.3 Entrada y Salida

Utilizando la teoría del muestreo podemos utilizar un pin del microcontrolador como entrada y salida en un determinado caso de salida por relevo.



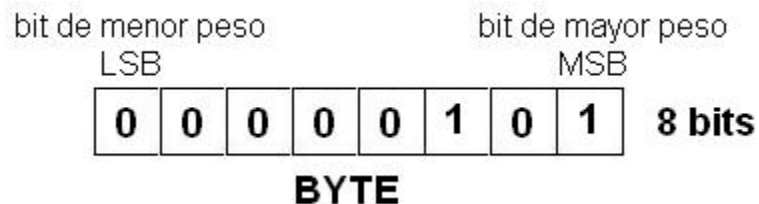
## 4 FUNCIONES COMBINATORIAS

Los circuitos que responden a la lógica binaria, de llaman compuertas lógicas, y gráficamente se expresan así.

NOMBRE	SIMBOLO	FUNCIÓN	TABLA															
AND		$Z = AB$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Z	0	0	0	0	1	0	1	0	0	1	1	1
A	B	Z																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$Z = A+B$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Z	0	0	0	0	1	1	1	0	1	1	1	1
A	B	Z																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT		$Z = \bar{A}$	<table border="1"> <thead> <tr> <th>A</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	Z	0	1	1	0									
A	Z																	
0	1																	
1	0																	
NAND		$Z = \overline{AB}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	Z	0	0	1	0	1	1	1	0	1	1	1	0
A	B	Z																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$Z = \overline{A+B}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	Z	0	0	1	0	1	0	1	0	0	1	1	0
A	B	Z																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
YES		$Z = A$	<table border="1"> <thead> <tr> <th>A</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td></tr> </tbody> </table>	A	Z	0	0	1	1									
A	Z																	
0	0																	
1	1																	
XOR		$Z = A\bar{B} + \bar{A}B$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>0</td></tr> </tbody> </table>	A	B	Z	0	0	0	0	1	1	1	0	1	1	1	0
A	B	Z																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR		$Z = AB + \bar{A}\bar{B}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>Z</th> </tr> </thead> <tbody> <tr><td>0</td><td>0</td><td>1</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </tbody> </table>	A	B	Z	0	0	1	0	1	0	1	0	0	1	1	1
A	B	Z																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

### 4.1 CONJUNTOS DE BITS

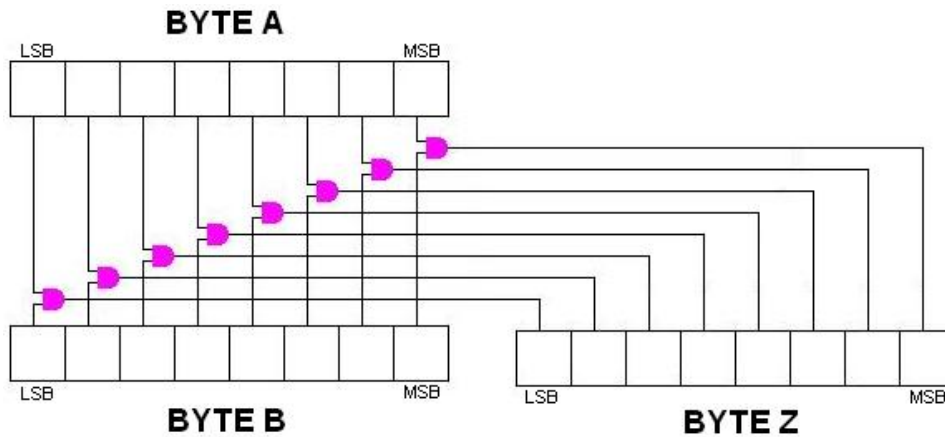
Un BYTE es el conjunto de 8 bits, y si el bit tiene solo dos estados el BYTE tiene 256 estados posibles. Donde 00000000 es 0 y 11111111 es 255.



Podemos aplicar la lógica booleana a este nivel, pero conceptualmente de forma distinta, veamos.

**AND** entre bytes.

Si llamamos a un byte A y otro B y el resultado de la operación lo dejamos en otro byte Z. Lo que se hace es una operación a nivel de bits como se muestra en la figura.



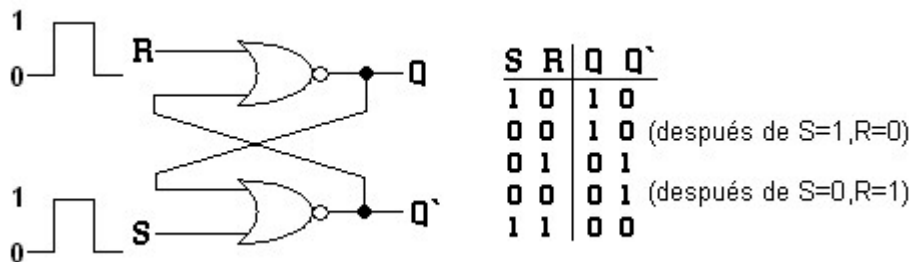
Podríamos implementar este mismo concepto pero con compuertas como NAND, OR, NOR, XOR, XNOR obteniendo resultados diferentes en la salida.

Para la compuerta NOT, el concepto es que el valor de cada uno de los bits del byte invierte su valor, quedando el byte "negado".

## 5 FUNCIONES SECUENCIALES

Un flip-flop puede mantener un estado binario por tiempo indefinido, hasta que sus estados en las entradas cambien. Los flip-flops se diferencian principalmente por la cantidad de entradas que posean y de como estas pueden afectar el estado de sus salidas.

Uno de los flip-flop más comunes es el tipo RS, el cual puede estar conformado por dos compuertas lógicas NOR, sus salidas son Q y Q', sus entradas son R y S, las cuales son iniciales del inglés SET y RESET, los cuales entenderemos como R, puesta a cero y S, puesta a uno.



Podemos entonces simplificar la función R con respecto a Q que sería:



En donde, cuando R se pone a uno Q será indefinidamente uno.

Y asimismo la función de S con respecto Q:



En donde, cuando S se pone a uno Q seá indefinidamente cero.

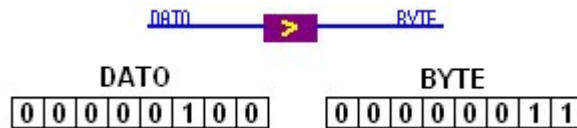
Como funcñ subsecuente un flip-flop tipo T, que opera como conmutador de salida cuando la entrada está a uno, se puede simplificar como un "Toggle" de la siguiente figura:



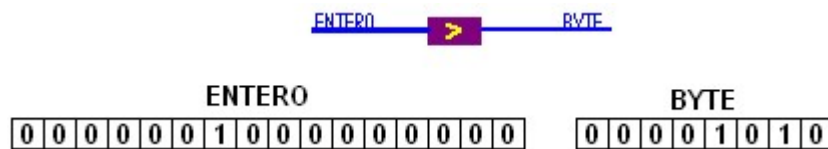
T	Q
0	0
1	1
0	1
1	0
0	0

## 6 AGRUPACIONES DEC Y COD CODIFICACIÓN

Cantidades discretas de información se presentan en sistemas digitales con códigos binarios. Un código binario de  $n$  bits es capaz de representar hasta  $2^n$  (dos a la  $n$ ), elementos diferentes de información codificada. Un decodificador es un circuito combinacional que convierte la información binaria de  $n$  líneas de entrada a un máximo de  $2^n$  (dos a la  $n$ ) líneas únicas de salida, veamos el ejemplo de la figura:

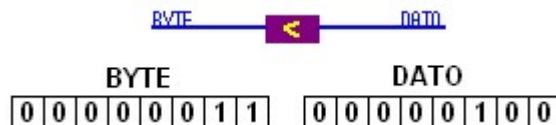


Podemos pensar entonces en más líneas de entrada para ser codificadas en un byte, por lo que entonces tendríamos la codificación de un ENTERO.

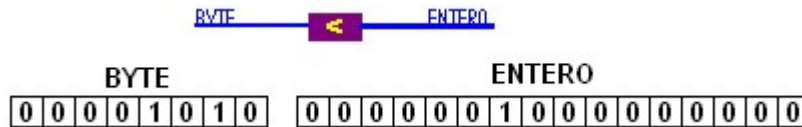


## DECODIFICACIÓN

Un proceso inverso que sufriría la información al de codificación es este, el cual parte de un número binario, para ser convertido en un único bit se salida activo.



E igualmente puede suceder con valores enteros.

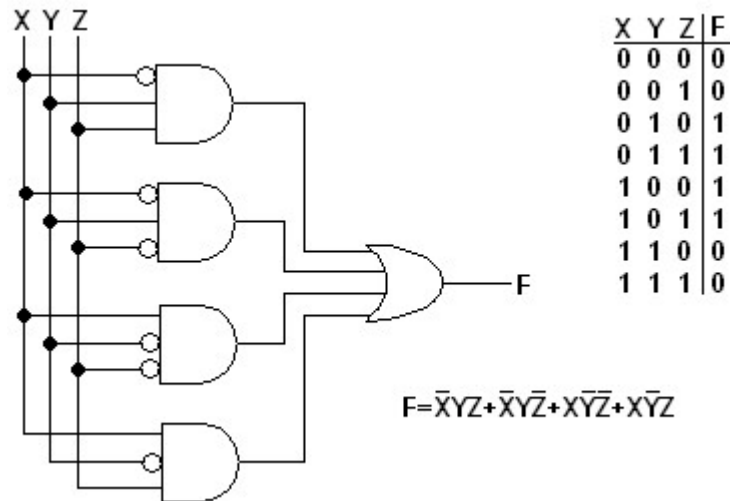


## 7 PRÁCTICAS

### 7.1 Circuito Combinatorio

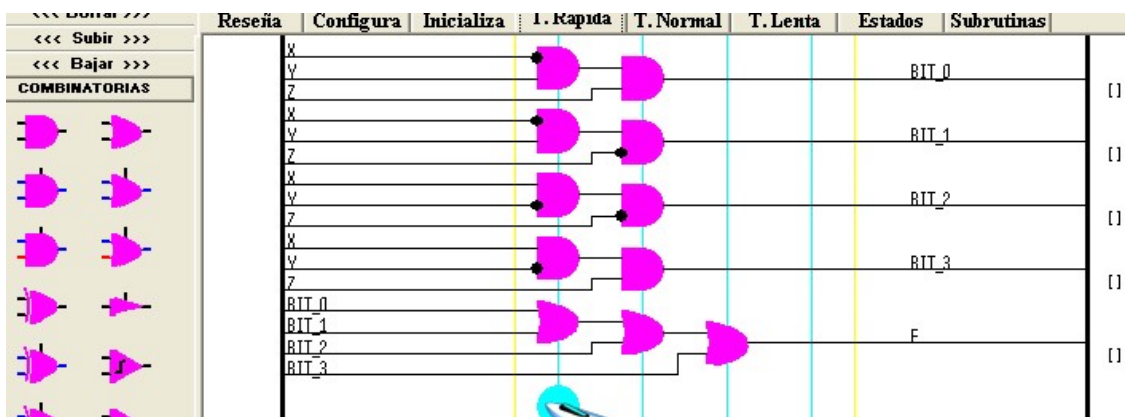
Para la siguiente función:  $F = \bar{X}YZ + \bar{X}Y\bar{Z} + X\bar{Y}Z + X\bar{Y}\bar{Z}$

Podemos realizar el siguiente análisis:



Que podemos implementar en el entorno de *microgrades*, de la siguiente forma.

En la sección de dispositivos "COMBINATORIAS", podemos seleccionar este tipo de compuertas pero como solo son de doble entrada anidamos de pares de ellas, por lo que la estrura del programa sería así:



Podemos disponer de tres interruptores y un led en hardware para comprobar este circuito, esto se realiza configurando primero el puerto en CONFIGURA/ESTRUCTURA, y luego dando nombres respectivos a los puertos de entrada y salida.

## 7.2 Simplificación de Funciones

La función de la práctica anterior puede simplificarse por medio de un mapa de Karnaugh de la siguiente forma:

X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

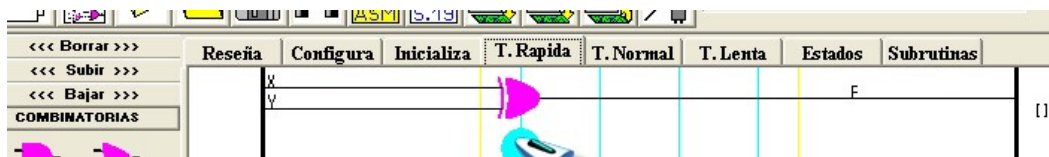
		y z		y	
		00	01	11 10	
X	0			1	1
	1	1	1		
		z			

De donde se puede deducir la siguiente expresión:  $F = \bar{X}Y + X\bar{Y}$

Que es más conocida como la compuerta lógica XOR.

Que podemos implementar en el entorno de *microgrades*, de la siguiente forma.

En la sección de dispositivos "COMBINATORIAS", podemos seleccionar este tipo de compuerta, y dispones sus entradas y salidas como se muestra a continuación.



Podemos disponer de dos interruptores y un led en hardware para comprobar este circuito, esto se realiza configurando primero el puerto en CONFIGURA/ESTRUCTURA, y luego dando nombres respectivos a los puertos de entrada y salida.

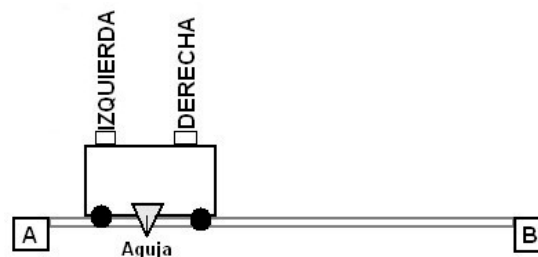
## 7.3 Máquina Tejedora

Los *Bits* pueden permanecer en Alto(1) ó Bajo(0), no solamente por factores externos, es decir, que podemos asegurar el estado de un *Bit* (1 ó 0) a través de nuestro programa.

Cuando Aseguramos un *Bit* a 1 a este proceso se le llama **SET**

Cuando Aseguramos un *Bit* a 0 a este proceso se le llama **RESET**

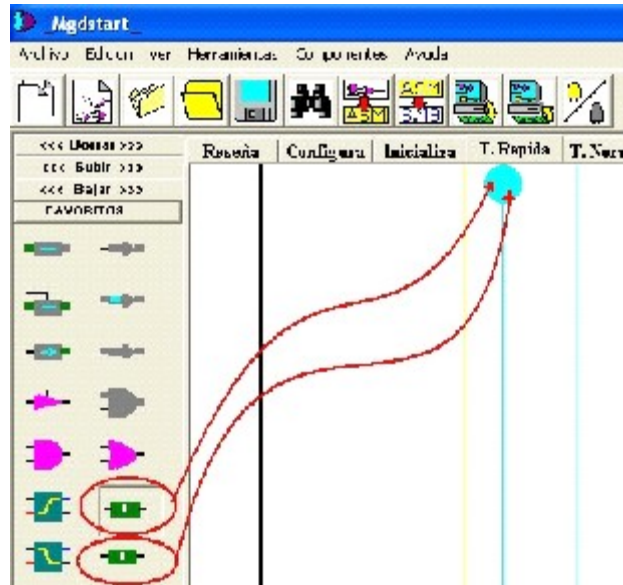
Para ello Imaginemos que se nos pide diseñar una maquina tejedora, que funciona de la siguiente manera:



Cuando la maquina llega al punto **A**, se tiene que activar el mecanismo que lo hace ir a la **Derecha**, y cuando llega a **B** se activa el mecanismo que lo hace ir a la **Izquierda**.

Para esto realizaremos lo siguiente:

Se le da el nombre a las entradas y salidas, luego procedemos como se muestra en la Figura.



**ENTRADAS:**

- A Sensor Izquierdo
- B Sensor Derecho

**SALIDAS:**

- Derecha Mecanismo que hace mover a la derecha
- Izquierda Mecanismo que hace mover a la Izquierda

En la **TAREA RAPIDA**, del menú **Favoritos** trabajamos los siguientes iconos colocándolos según la Figura.



## **8 BIBLIOGRAFÍA**

- [1] M. Moris Mano, Lógica digital y diseño de computadores, Ed. Prentice hall.
- [2] Wakerly, John, Diseño digital, principios y prácticas, 3 edición, Ed. Prentice hall.
- [3] Mandado Enrique, Electrónica Digital, Ed. Marcombo