INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

ELECTRÓNICA DIGITAL: CIRCUITOS LÓGICOS, LENGUAJE VHDL Y VERILOG

XILINX (64-BIT PROJECT NAVIGATOR) & ADEPT

Ejercicios de Compuertas Lógicas

Contenido

Ejercicios de Compuertas Lógicas

Los ejercicios de electrónica digital más simples son aquellos que se pueden resolver simplemente aplicando alguna de las compuertas principales a las diferentes entradas ya sea la compuerta AND, OR, NOT, NAND, NOR, XOR y XNOR.

Las combinaciones se calculan dependiendo del número de entradas siguiendo la siguiente fórmula:

$$\#combinaciones = 2^{\#variables}$$

El número de combinaciones se cuenta desde cero hasta alcanzar el #combinaciones - 1 y se cuenta en forma de número binario.

a) **AND:** Todas las entradas deben ser de 1 lógico para que la salida sea también 1 lógico.

Tabla de verdad 2 variables:

$$\#combinaciones = 2^{\#variables} = 2^2 = 4$$

Tabla de verdad 3 variables:

$$\#combinaciones = 2^{\#variables} = 2^3 = 8$$

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	AND
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

b) **OR:** Basta con que una de las entradas sea 1 lógico para que la salida sea también 1 lógico.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	OR
0	0	0	0
0	0	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	1

c) **NOT:** La salida siempre es lo contrario de la entrada.

ENTRADA	SALIDA
	NOT
0	1
1	0

d) **NAND:** Es la compuerta AND negada, en su salida está lo opuesto a lo que habría en la compuerta AND.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	NAND
0	0	0	1
0	0	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	0

e) **NOR:** Es la compuerta OR negada, en su salida está lo opuesto a lo que habría en la compuerta OR.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	NOR
0	0	0	1
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	0

f) **XOR:** En la salida hay un 1 lógico siempre y cuando las dos entradas sean diferentes. Si hubiera más de 2 entradas, debo analizarlas de 2 en 2 para llegar al resultado de la salida.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	XOR
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

g) **XNOR:** Es la compuerta XOR negada, en su salida está lo opuesto a lo que habría en la compuerta XOR.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	XNOR
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

1.- Implementar un Sistema de Alarma para una Casa Habitación:

Este detectará la apertura de tres puertas (P1, P2, P3) y cuatro ventanas (V1, V2, V3, V4), cuando detecte que alguna de estas ha sido abierta activará un buzzer para dar una señal de alarma.

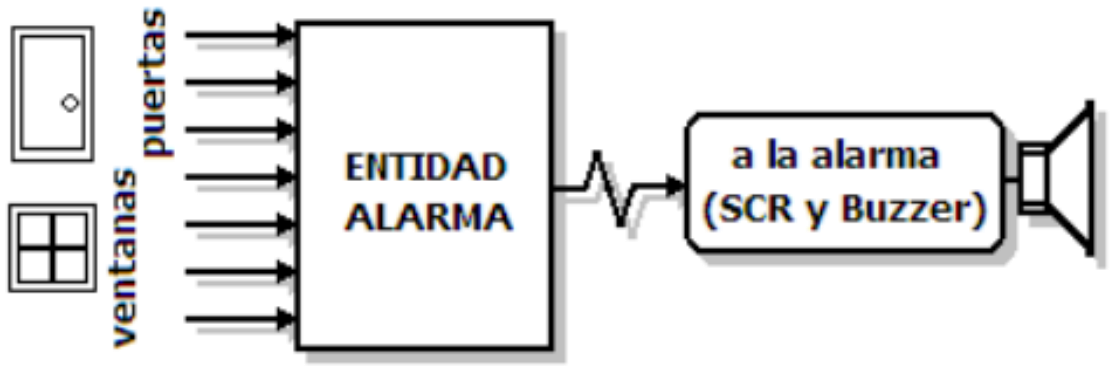


Figura 4. Sistema de alarma.

En este ejercicio tenemos 7 variables por lo que la tabla de verdad se calculará con la fórmula del # de combinaciones de la siguiente manera:

$$\#combinaciones = 2^{\#variables} = 2^7 = 128$$

No es necesario poner toda la tabla de verdad simplemente debo analizar que lo que dice el problema es que la alarma se activará cuando cualquiera de las entradas cense algo, osea que esté en 1 lógico, esto significa que la compuerta a usar es la **OR** ya que basta con que una de las entradas sea 1 lógico para que la salida sea también 1 lógico.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	OR
0	0	0
0	1	1
1	0	1
1	1	1

TABLA DE VERDAD 3 VARIABLES			
ENTRADAS			SALIDA
A	B	C	OR
0	0	0	0
0	0	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	1

Código VHDL:

```
--Declaración de librerías
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Entidad: Declaración de entradas/salidas
entity punto2 is
    Port ( v1 : in  STD_LOGIC;
           v2 : in  STD_LOGIC;
           v3 : in  STD_LOGIC;
           v4 : in  STD_LOGIC;
           p1 : in  STD_LOGIC;
           p2 : in  STD_LOGIC;
           p3 : in  STD_LOGIC;
           sal : out  STD_LOGIC);
end punto2;

--Arquitectura: Uso de las entradas/salidas previamente declaradas
architecture AlarmaCasaHabitacion of punto2 is
begin
    sal<=(v1 or v2 or v3 or v4 or p1 or p2 or p3);
end AlarmaCasaHabitacion;
```

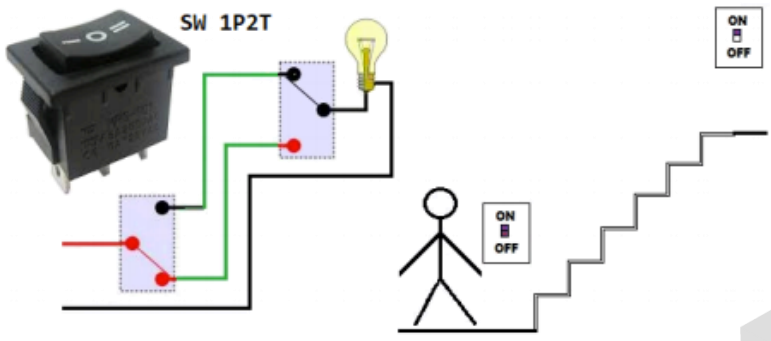
Simulación del Código:

Para comprobar este resultado podemos usar el simulador del editor de código ISE XIlinx.

En la simulación podemos ver que la salida siempre es 1 lógico excepto cuando todas las entradas son cero por lo que nuestra lógica y código están correctamente planteados.

2.- Circuito de Encendido y Apagado de una Carga desde dos Puntos:

Implementar un circuito VHDL que permita controlar el encendido y apagado de una carga desde dos puntos, una carga puede ser un foco, motor, apertura-cierre de puertas, etc., similar a los apagadores de escaleras como se muestra a continuación:



Primero analizo si el problema tiene un número de variables que solo tengan dos estados, como el switch 1 y el switch 2 solo pueden estar prendidos o apagados, solo tengo 2 variables lógicas y puedo

resolver el problema por medio de compuertas lógicas. Para que el circuito lógico pueda funcionar de manera correcta debo analizar los casos posibles:

- Si ambos switches están apagados el foco debe permanecer apagado.
- Si un switch estaba encendido y de repente enciendo el otro, el foco se debe apagar para dar control desde ambos lados.
- Si un switch estaba apagado y prendo el otro, el foco debe encenderse.

Ahora lo que hago es ver que tabla de verdad cumple estas condiciones, como la compuerta **XOR** manda a la salida un 1 lógico siempre y cuando las dos entradas sean diferentes, esta es la compuerta que uso.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Código VHDL:

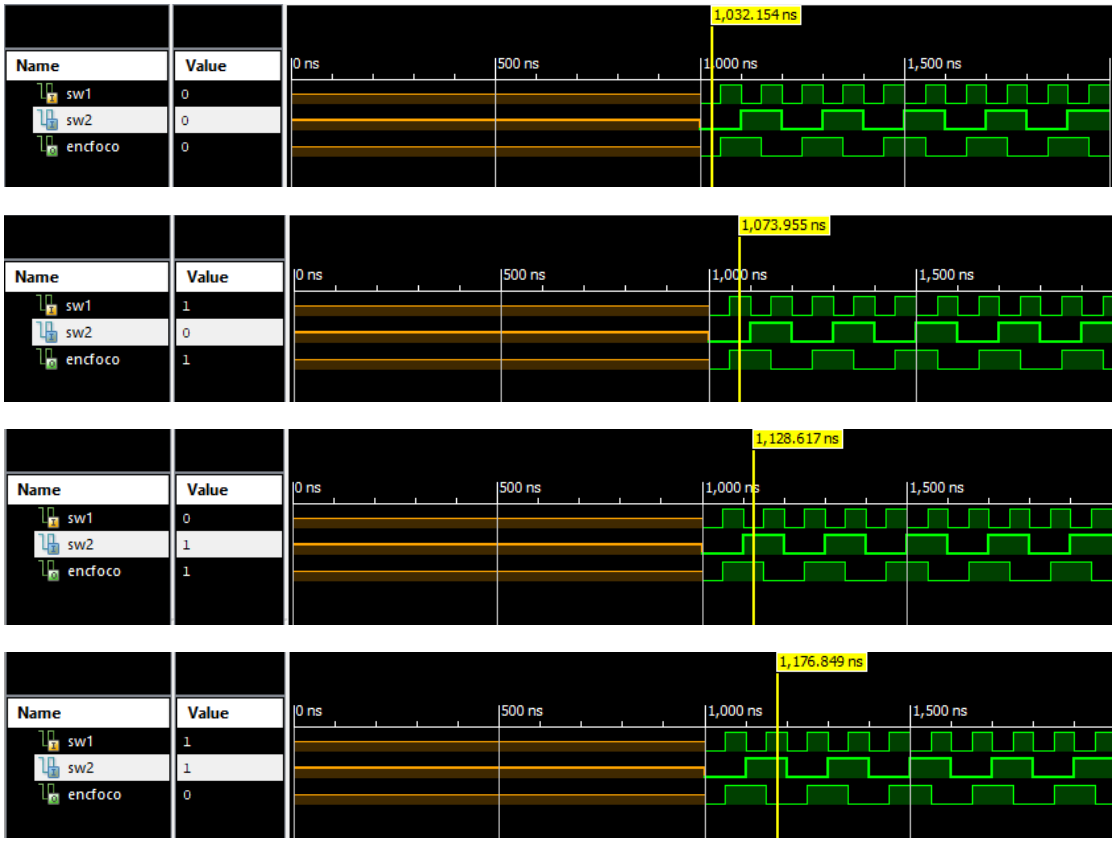
```
--Declaración bibliotecas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Entidad: Parte del código VHDL donde se declaran las entradas/salidas
entity punto3a is
    Port ( sw1 : in  STD_LOGIC;
           sw2 : in  STD_LOGIC;
           encFoco : out  STD_LOGIC);
end punto3a;

--Arquitectura: Uso de las entradas/salidas previamente declaradas
architecture PrendidoApagado of punto3a is
begin
    encFoco<=sw1 xor sw2;
end PrendidoApagado;
```

Simulación del Código:

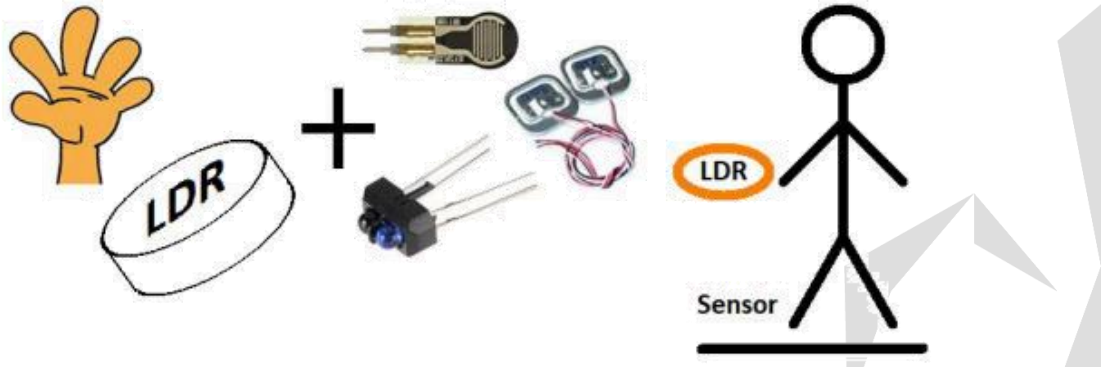
Para comprobar este resultado podemos usar el simulador del editor de código ISE XIlinx.



Y puedo comprobar que en efecto se cumplen todas las condiciones de mi tabla de verdad.

3.- Timbre de una Casa con Sensor de Luz:

Implementar un circuito con HDL, que, por medio del paso de una mano sobre un sensor de luz (que impide la incidencia de la luz sobre el sensor) y la presencia de una persona (con otro sensor), suene el timbre de una casa.



Como el problema se maneja por medio de sensores, se puede ver que si es un problema que puede ser resuelto por compuertas lógicas y como ambas condiciones se deben cumplir para que se active el timbre de la casa, se usa una compuerta **AND** ya que con esta compuerta todas las entradas deben ser de 1 lógico para que la salida sea también 1 lógico.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	AND
0	0	0
0	1	0
1	0	0
1	1	1

Código VHDL:

```
--Declaración bibliotecas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Entidad: Parte del código VHDL donde se declaran las entradas/salidas
entity punto3b is
    Port ( e1 : in  STD_LOGIC;
           e2 : in  STD_LOGIC;
           timbre : out  STD_LOGIC);
end punto3b;

--Arquitectura: Uso de las entradas/salidas previamente declaradas
architecture prendeLed of punto3b is
begin
    timbre<=e1 and e2;
end prendeLed;
```

4.- Circuito de Acceso a un Banco de Doble Puerta:

Diseñar un circuito que permita el acceso a un banco de doble puerta, que solo permita abrir una puerta a la vez, nunca las dos al mismo tiempo. Reportar los códigos y circuitos a bloques.

Como en este ejercicio se deben manejar dos puertas, se deben usar dos compuertas lógicas y estas deben ser opuestas ya que se debe abrir una puerta a la vez, mientras une está abierta la otra debe estar cerrada. Como debemos asegurarnos de que las dos puertas no se abran al mismo tiempo, podemos asumir que la compuerta a usar es la **XOR** en una de las puertas ya que en la salida hay un 1 lógico siempre y cuando las dos entradas sean diferentes y en la otra debe estar la **XNOR**, ya que es la compuerta XOR negada.

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

TABLA DE VERDAD 2 VARIABLES		
ENTRADAS		SALIDA
A	B	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Código VHDL:

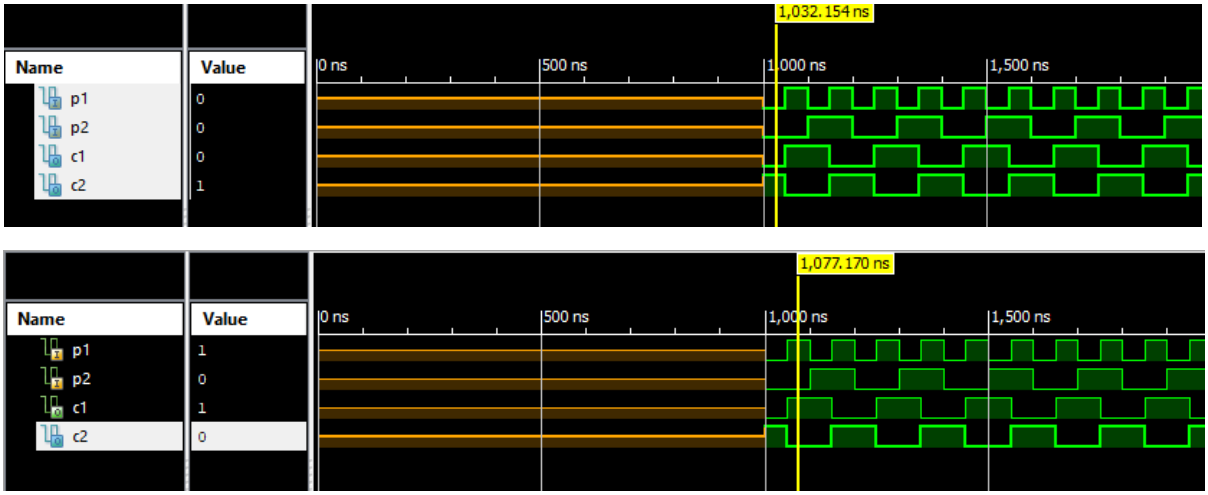
```
--Declaración bibliotecas
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--Entidad: Parte del código VHDL donde se declaran las entradas/salidas
entity punto3d is
    Port ( p1 : in  STD_LOGIC;--Puerta 1
           p2 : in  STD_LOGIC;--Puerta 2
           c1 : out  STD_LOGIC;--Cerrojo 1
           c2 : out  STD_LOGIC);--Cerrojo 2
end punto3d;

--Arquitectura: Uso de las entradas/salidas previamente declaradas
architecture accesoPuerta of punto3d is
begin
    c1<=p1 xor p2;
    c2<=p1 xnor p2;
end accesoPuerta;
```

Simulación del Código:

En la simulación lo que debe pasar es que siempre debe estar abierta solo una de las dos puertas.



Operadores de VHDL y Verilog

A continuación, se presenta una tabla que describe todos los tipos de operadores que se pueden utilizar con los lenguajes VHDL y Verilog al programar una FPGA.

Operadores	VHDL	Verilog
Lógicos	AND, NAND, OR, NOR, XOR, XNOR, NOT	& and, ~& nand | or, ~| nor, ^ xor, ~^ xnor ~ not, En condicionales: "!" negación lógica "&&" AND lógica "||" OR lógica
Aritméticos	+ signo positivo o suma dos números, a <= b + c; - signo negativo o resta dos números, a <= b - c; * multiplicación, a <= b * c; / división, a <= b / c; ** exponencial ABS( ) valor absoluto MOD módulo REM resto de la división	+ signo o suma, a = b + c; - signo o resta, a = b – c; * multiplicación, a = b * c; / división, a = b / c; (entero) % módulo, a = b % c;
Relacionales Entregan un valor booleano (true o false)	==, /=, igualdad, desigualdad >, >=, mayor que, mayor o igual que, <, <=, menor que, menor o igual que	== igual, != diferente incluyendo "x" y "z": === igual, !== diferente < menor que, <= menor o igual que > mayor que, >= mayor o igual que

Operadores	VHDL	Verilog
Desplazamientos	SLL (Shift Left Logic) y SRL (Shift Right Logic) SLA (Shift Left Arithmetic) y SRA (Shift Right Arithmetic) ROL (ROtate Left) y ROR (ROtate Right)	<< Desplazamiento a izquierda >> Desplazamiento a derecha
Otros	& concatenación de vectores, A & B, '0' & A.	{} Concatenación de operandos, {a,b[3:0],c}, {3{a}} = {a,a,a} ? Condicional, (condición) ? true_expr

Además, es importante mencionar que cuando no se puedan crear soluciones lógicas para los problemas con los operadores lógicos básicos (AND, NAND, OR, NOR, XOR, XNOR y NOT), se debe hacer uso de una herramienta llamada mapa de Karnaugh, que crea ecuaciones utilizando álgebra booleana para describir el diseño de circuitos lógicos que manejen situaciones con condiciones más complejas.