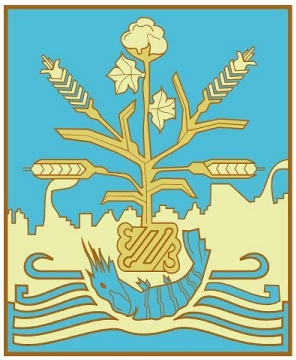


INSTITUTO TECNOLÓGICO DE SONORA

Educar para Trascender

PROYECTO FINAL:

ELEVADOR

Integrantes:

Francisco Antonio Palos Angulo 132759

Kaled Andrés Muhech Álvarez 137959

Jorge Luis Domínguez Trejo 132607

Manuel Aurelio Nozato Beltrán 129577

Cd. Obregón, Sonora., 27 de abril de 2017.

ÍNDICE

1.- INTRODUCCIÓN

En este reporte se encuentran todos los datos, teoría y pasos necesarios que se ocuparon para la construcción de nuestro elevador de 5 niveles hecho solo con la tecnología que vimos durante la clase de digitales I.

Para llevarlo a cabo, se analizara lo que sea más conveniente para desarrollar el proyecto.

Con la Gal y programación VHDL se puede desarrollar uno o varios programas que se pueden realizar con varios circuitos integrados en uno solo, considerando la cantidad de patas disponibles para realizarlo.

El proyecto está conformado por el sistema de una impresora, un motor, una fuente de voltaje, tres gales, dos display de cátodo común y una interface de potencia. Con la programación VHDL que pusimos en la Gales controlamos el elevador y las señales que mandamos hacia los displays.

2.- ANTECEDENTES TEÓRICOS

GAL:

GAL (Generic Array Logic), en español Arreglo Lógico Genérico, son un tipo de circuito integrado, de marca registrada por Lattice Semiconductor, que ha sido diseñados con el propósito de sustituir a la mayoría de las PAL, manteniendo la compatibilidad de sus terminales. Utiliza una matriz de memoria EEPROM en lugar por lo que se puede programar varias veces. Un GAL en su forma básica es un PLD con una matriz AND reprogramable, una matriz OR fija y una lógica de salida programable mediante una macrocelda. Esta estructura permite implementar cualquier función lógica como suma de productos con un número de términos definido.

VHDL:

VHDL. Lenguaje de descripción hardware estructurado para modelar sistemas digitales.
V: VHSIC – Very High Speed Integrated Circuit
H: Hardware
D: Description
L: Language

(Suponiendo que V2 es la tensión de referencia)

Interfaz de Potencia:

Las interfaces de potencia son dispositivos intermedios entre nuestro microcontrolador y aquellos aparatos que requieran cantidades de corriente mayores a los que pueden manejar el microcontrolador (por lo general estamos hablando de una corriente de unos 40 miliamperios como máximo por pin).
Motores de paso, motores DC, servomotores, lámparas incandescentes, reflectores, grupos o tiras de leds, son ejemplos de dispositivos que podríamos llegar a controlar desde el microcontrolador a través de las interfaces de potencia. Es un grave error tratar de conectar este tipo de dispositivos directamente a los pines del microcontrolador. Nos ayudaremos de transistores, relés, puentes-H o interfaces electrónicas de control para construirlas.

3.- DISEÑO DEL PROYECTO

3.1 Enunciado del proyecto.

Realizar un sistema que permita mover lo que sería el cuarto del elevador en dos direcciones y además detenerlo en el momento en el que llegue a los pisos solicitados, esto lo lograremos mediante el uso de sistema de elevador con el que cuenta por dentro la impresora de tinta, para el proyecto solo nos fue necesario utilizar la tecnología VHDL y GAL.

3.2 Aspectos de intuición a considerar

La GAL que tiene el código del elevador recibirá valores de los sensores y de los push buttons, luego los comparará y una vez comparado realizará el movimiento que sea necesario para que las señales de ambos se igualen.

Una vez que las señales se igualen el motor se detendrá evitando así que cambie a un piso no buscado y si el botón del piso en el que ya se encuentra vuelve a ser presionado el elevador no deberá moverse.

3.3 Código VHDL

Primera gal Elevador:

```
library ieee;

use ieee.std_logic_1164.all;

entity controlador_GAL is

port (

        I_Sensor: in std_logic_vector(4 downto 0);

        I_Teclado: in std_logic_vector(4 downto 0);

        O_Sensor: out std_logic_vector(2 downto 0);

        O_Teclado: out std_logic_vector(2 downto 0);

        M: out std_logic_vector(1 downto 0));

end controlador_GAL;

architecture funcion of controlador_GAL is


signal Signal_Sensor: std_logic_vector(2 downto 0):="000";

signal Signal_Teclado: std_logic_vector(2 downto 0):="000";
```

```
begin
process (I_Sensor)
begin
	case I_Sensor is
		when "00001" => Signal_Sensor <= "001";
		when "00010" => Signal_Sensor <= "010";
		when "00100" => Signal_Sensor <= "011";
		when "01000" => Signal_Sensor <= "100";
		when "10000" => Signal_Sensor <= "101";
		when others =>  Signal_Sensor <= Signal_Sensor;
	end case;
	end process;
process (Signal_Sensor)
	begin
		O_Sensor <= Signal_Sensor;
	end process;
process (I_Teclado)
	begin
	case I_Teclado is
		when "00001" => Signal_Teclado <= "001";
		when "00010" => Signal_Teclado <= "010";
		when "00100" => Signal_Teclado <= "011";
		when "01000" => Signal_Teclado <= "100";
		when "10000" => Signal_Teclado <= "101";
		when others =>  Signal_Teclado <= Signal_Teclado;
	end case;
	end process;
process (Signal_Teclado)
	begin
```

```
                O_Teclado <= Signal_Teclado;
        end process;
process (Signal_Sensor, Signal_Teclado) -- Comparador
        begin
        if Signal_Sensor = Signal_Teclado then
                M <= "00";
        elsif Signal_Sensor < Signal_Teclado then
                M <= "01";
        else
                M <= "10";
        end if;
        end process;
end funcion;
```

Segunda gal SDA:

```
library ieee;
use ieee.std_logic_1164.all;
entity controlador_GAL is
port (
        I_Sensor: in std_logic_vector(2 downto 0);
        display_acutal: out std_logic_vector(6 downto 0));
end controlador_GAL;
architecture funcion of controlador_GAL is
signal Signal_Sensor: std_logic_vector(6 downto 0):="0000000";
begin
process (I_Sensor)
begin
        case I_Sensor is
                when "001" => Signal_Sensor <= "0110000";
```

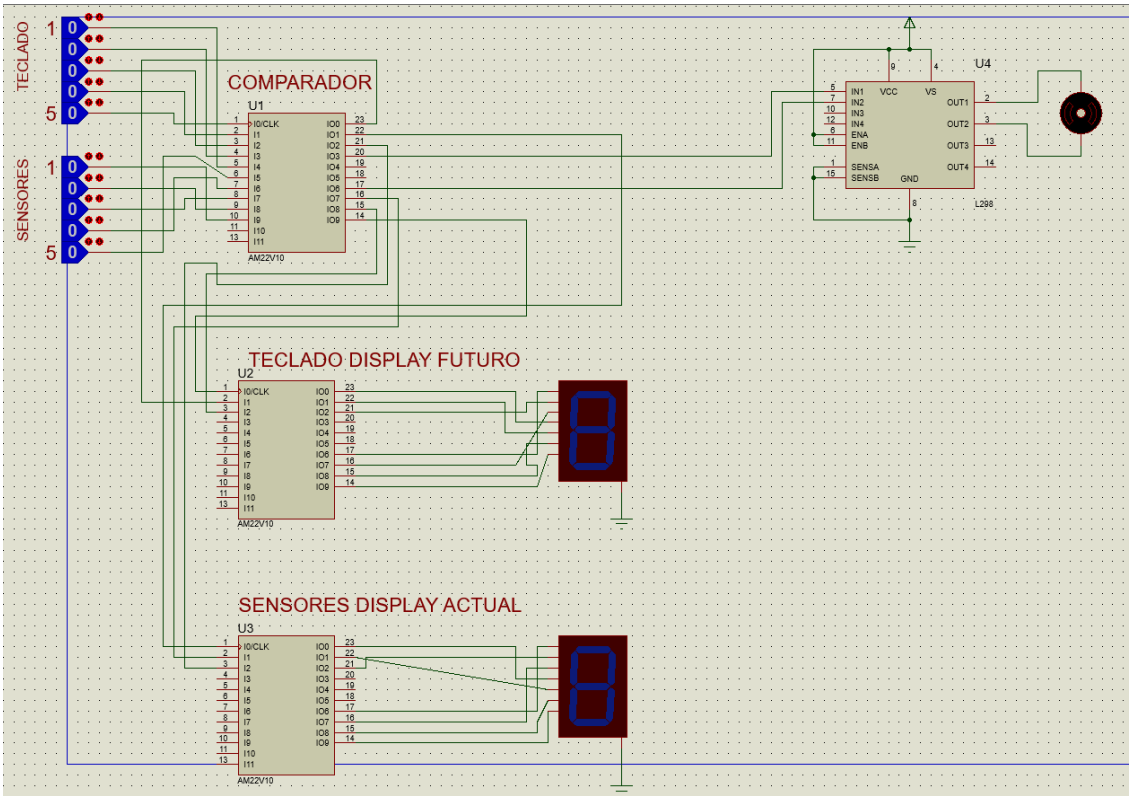
```
                when "010" => Signal_Sensor <= "1101101";
                when "011" => Signal_Sensor <= "1111001";
                when "100" => Signal_Sensor <= "0110011";
                when "101" => Signal_Sensor <= "1011011";
                when others =>  Signal_Sensor <= Signal_Sensor;
        end case;
        end process;
process (Signal_Sensor)
        begin
                display_acutal <= Signal_Sensor;
        end process;
end funcion;
```

Tercera gal TDF:

```
library ieee;
use ieee.std_logic_1164.all;
entity controlador_GAL is
port (
        I_Teclado: in std_logic_vector(2 downto 0);
        display_futuro: out std_logic_vector(6 downto 0));
end controlador_GAL;
architecture funcion of controlador_GAL is
signal Signal_Teclado: std_logic_vector(6 downto 0):="0000000";
begin
process (I_Teclado)
begin
        case I_Teclado is
                when "001" => Signal_Teclado <= "0110000";
                when "010" => Signal_Teclado <= "1101101";
```

```
                when "011" => Signal_Teclado <= "1111001";
                when "100" => Signal_Teclado <= "0110011";
                when "101" => Signal_Teclado <= "1011011";
                when others =>  Signal_Teclado <= Signal_Teclado;
        end case;
        end process;
process (Signal_Teclado)
        begin
                display_futuro <= Signal_Teclado;
        end process;
end funcion;
```

3.4 Diagrama del circuito impreso.



4.- CONCLUSIONES, LIMITACIONES Y RECOMENDACIONES.

Conclusiones

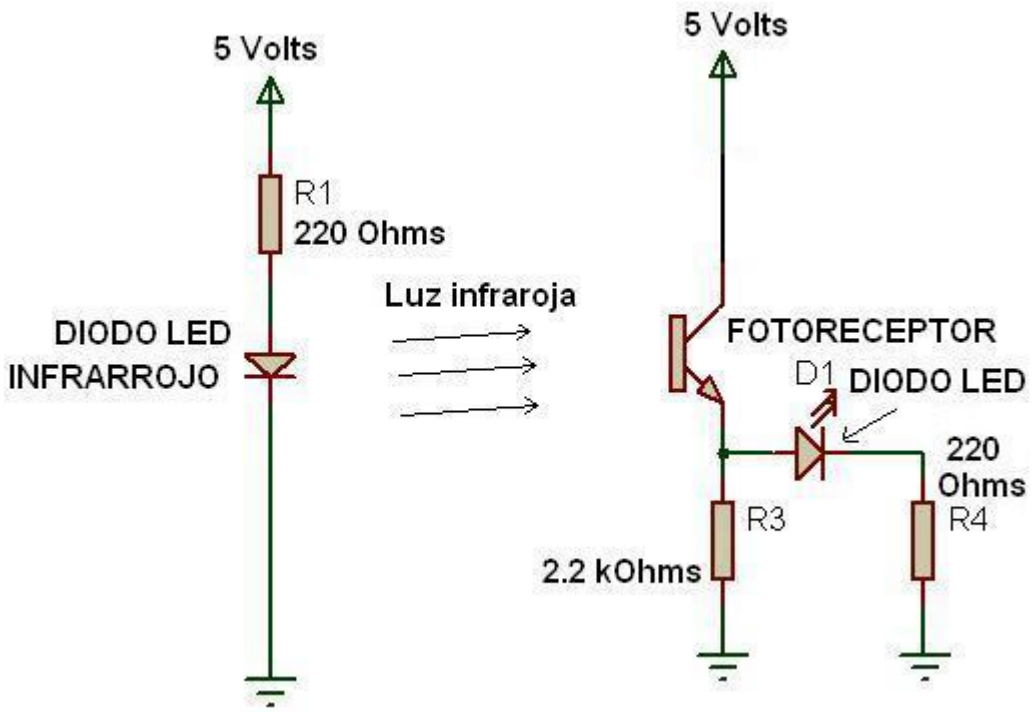
Este proyecto que realizamos para la clase de digitales I ha sido muy importante para nuestra formación como ingenieros en este tipo de áreas de ingeniería, ya que es nuestro primer proyecto de tipo Mecatrónico y nos ayudará mucho a futuro, ya que nos va dando una idea de cómo hacer proyectos más elaborados.

Este proyecto nos sirvió para en un futuro tener una mejor de como realizar algún proyecto, los pasos que se llevan a cabo y la forma correcta de para realizarlo, de esta manera tenemos un idea más formada del trabajo al cual se enfrenta un verdadero ingeniero.

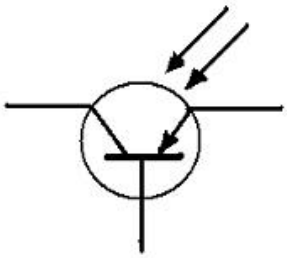
La electrónica digital nos ofrece un vasto mundo de conocimiento sobre el mundo moderno, en el cual se pueden realizar muchísimas cosas que ya están descubiertas o que faltan por descubrir o mejorar, lo cual nos da muchas oportunidades de desarrollo a futuro.

5.- ANEXOS.

Sensores

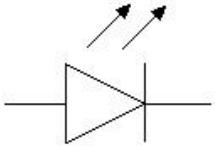


Fototransistor TIL78



Símbolo do Fototransistor

Led Infra Vermelho



Símbolo do Led Infra Vermelho

Lattice Semiconductor Corporation

Lead-Free Package Options Available!

GAL22V10

High Performance E²CMOS PLD
Generic Array Logic™

Features

- **HIGH PERFORMANCE E²CMOS® TECHNOLOGY**
  - 4 ns Maximum Propagation Delay
  - Fmax = 250 MHz
  - 3.5 ns Maximum from Clock Input to Data Output
  - UltraMOS® Advanced CMOS Technology
- **ACTIVE PULL-UPS ON ALL PINS**
- **COMPATIBLE WITH STANDARD 22V10 DEVICES**
  - Fully Function/Fuse-Map/Parametric Compatible with Bipolar and UVCMOS 22V10 Devices
- **50% to 75% REDUCTION IN POWER VERSUS BIPOLAR**
  - 90mA Typical Icc on Low Power Device
  - 45mA Typical Icc on Quarter Power Device
- **E² CELL TECHNOLOGY**
  - Reconfigurable Logic
  - Reprogrammable Cells
  - 100% Tested/100% Yields
  - High Speed Electrical Erasure (<100ms)
  - 20 Year Data Retention
- **TEN OUTPUT LOGIC MACROCELLS**
  - Maximum Flexibility for Complex Logic Designs
- **PRELOAD AND POWER-ON RESET OF REGISTERS**
  - 100% Functional Testability
- **APPLICATIONS INCLUDE:**
  - DMA Control
  - State Machine Control
  - High Speed Graphics Processing
  - Standard Logic Speed Upgrade
- **ELECTRONIC SIGNATURE FOR IDENTIFICATION**
- **LEAD-FREE PACKAGE OPTIONS**

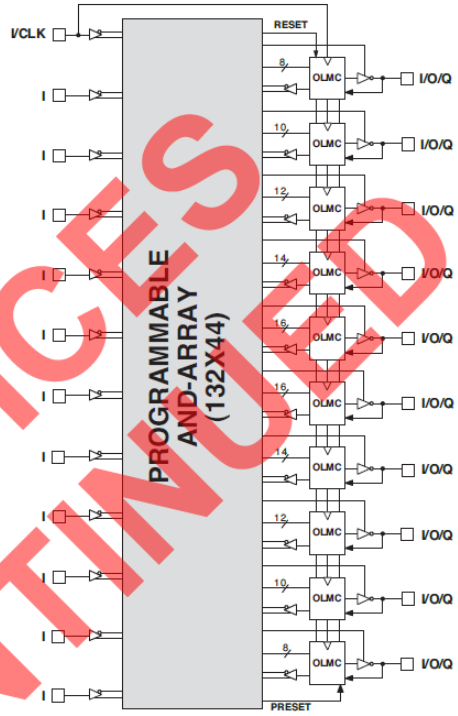
Description

The GAL22V10, at 4ns maximum propagation delay time, combines a high performance CMOS process with Electrically Erasable (E²) floating gate technology to provide the highest performance available of any 22V10 device on the market. CMOS circuitry allows the GAL22V10 to consume much less power when compared to bipolar 22V10 devices. E² technology offers high speed (<100ms) erase times, providing the ability to reprogram or reconfigure the device quickly and efficiently.

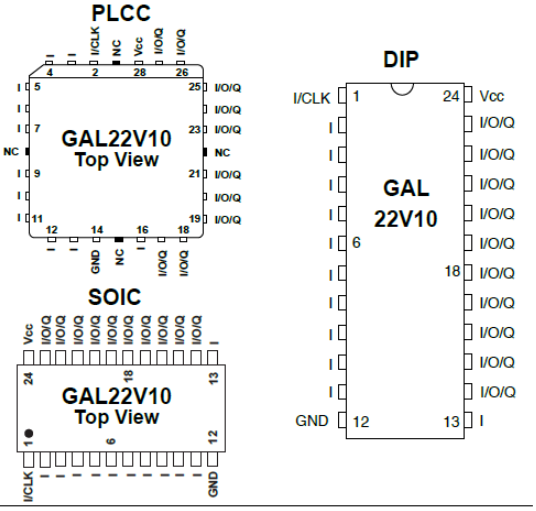
The generic architecture provides maximum design flexibility by allowing the Output Logic Macrocell (OLMC) to be configured by the user. The GAL22V10 is fully function/fuse map/parametric compatible with standard bipolar and CMOS 22V10 devices.

Unique test circuitry and reprogrammable cells allow complete AC, DC, and functional testing during manufacture. As a result, Lattice Semiconductor delivers 100% field programmability and functionality of all GAL products. In addition, 100 erase/write cycles and data retention in excess of 20 years are specified.

Functional Block Diagram



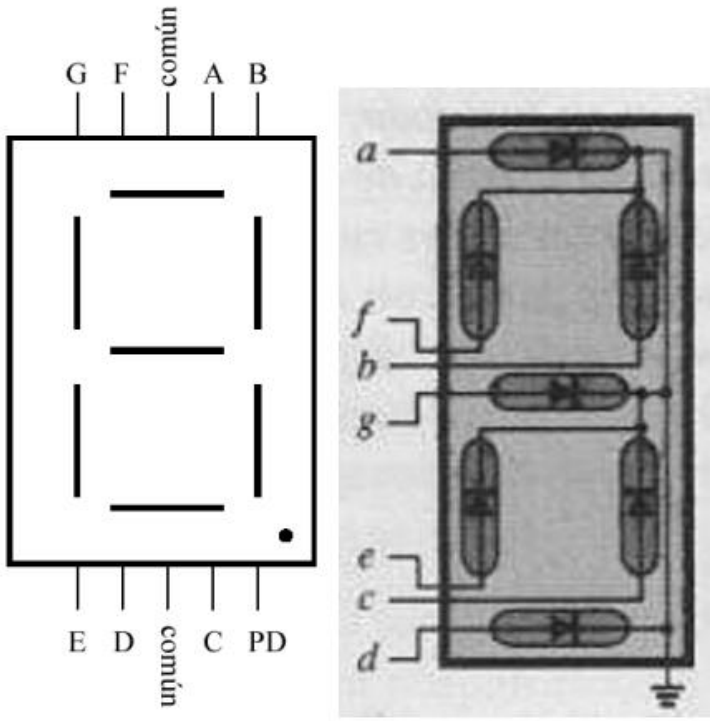
Pin Configuration



LATTICE SEMICONDUCTOR CORP., 5555 Northeast Moore Ct., Hillsboro, Oregon 97124, U.S.A.
Tel. (503) 268-8000; 1-800-LATTICE; FAX (503) 268-8556; http://www.latticesemi.com

December 2006

Display



a) Terminales del dispositivo. b) Display de cátodo común.