**HUMMER H2**

Proyecto Final

**Diseño Digital VLSI (1535)**

**Grupo 4**

**Prof.: Prieto Meléndez Rafael**

**Facultad de Ingeniería**

**UNAM**

**30 de Noviembre 2018**

Integrantes:

* Castro Molano Ramsés
* Gaytán Saldaña Josué Yair
* Guerrero Ruiz Omar
* Orozco Magadan Brandon

INTRODUCCIÓN

Una FPGA o matriz de puertas programables (del inglés field-programmable gate array) es un dispositivo programable que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento mediante un lenguaje de descripción  
especializado. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos  
sistemas en un chip.

Comunicación UARTUART significa Universal Asynchronous Receiver-Transmitter. Es un protocolo de serie asíncrono que se caracteriza por la ausencia de señal de reloj(clock). La sincronización se efectúa mediante el bit de arranque que precede a cada byte; a partir de ese momento el  
receptor asume que los bits llegaran a la misma velocidad de su reloj interno. El cual puede conectarse mediante puerto USB o VGA.  
Serial (UART) protocolo serie asíncrono, 2 líneas: Tx y Rx, 9600- 250k [b/s], compatible con el RS232.  
El emisor y el receptor deben estar configurados de manera similar.

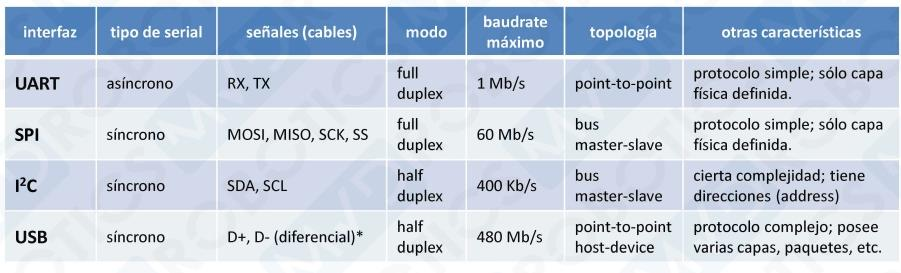
Baud rate (velocidad): Se mide en bps, algunos estándares son: 1200, 2400, 4800, 9600,  
19200, 38400, 57600 y 115200 [bps].  
Data bis (número de bits de datos): Pueden ser 5,6,7,8 o 9. Habitualmente 8.  
Parity (paridad) par(even) impar(odd) o ninguna. La paridad es un método primitivo de  
control de errores; consiste en sumar todos los bits de datos y enviar 1 bit adicional indicando si la suma es par o impar. Stop bits (número de bits de parada) Pueden ser 1 o 2. Flow control (control de flujo) Algunos dispositivos implementan unas líneas auxiliares para  
regular el flujo de datos: RTS, CTS, DTR y DSR.

ÍNDICE

OBJETIVO …………………………………………………………… 03

DESCRIPCIÓN …………………………………………………….. 03

PRINCIPALES COMPONENTES …………………………… 03

SISTEMA DE CONTROL DEL CARRITO …………….. 05

RESULTADOS FINALES ……………………………………… 08

CONCLUSIÓN …………………………………………………….. 09

REFERENCIAS ……………………………………………………. 09

ANEXO ……………………………………………………………….. 10

DataSheets ……………………………………………….. 10

OBJETIVO

El principal objetivo de este proyecto es la implementación de los conocimientos adquiridos en clase, así como los obtenidos durante las prácticas realizadas a lo largo de todo el curso.

DESCRIPCIÓN

Manejo de motores:

Para el manejo de los motores se usó el driver l298n. Se ocupó la configuración que permite alimentaciones entre 12 y 30v ya que los motores del carrito trabajan con una tensión de 19.2v.

Para la alimentación del carrito se ocupó un eliminador de 24v y 150mA de salida, cabe señalar que no fue necesario el uso de ningún regulador ya que el Driver tiene un consumo de entre 3v y 5v lo cual reduce la tensión entregada por la fuente.

Sistema de luces:

* 16 leds de color blanco y rojo de 5 mm
* 16 resistencias de 330Ω

Para las conexiones se usó cable calibre 22 para la alimentación y cable UTP para todas las conexiones lógicas.

El carrito usado en el proyecto es una HUMMER H2 a gran escala de la marca NEW BRIGHT. No se ocupó ningún componente de radiofrecuencia que incluyera el carrito, solo se usaron sus motores. Cabe señalar que la batería del carrito ya no servía y todo el sistema diseñado fue adaptado de manera que funcionara con una alimentación externa tanto para los motores como para la parte lógica del sistema.



**PRINCIPALES COMPONENTES**

**El lenguaje VHDL**

VHDL es el acrónimo que representa la combinación de VHSIC (Very High Speed Integrated Circuit) y HDL (Hardware Description Language).  
Es un lenguaje usado para describir sistemas electrónicos digitales, fue creado para cubrir una serie de necesidades en el proceso de diseño.  
Dentro del VHDL hay varias formas con las que podemos diseñar un mismo circuito:

♣Funcional  
♣Flujo de datos

♣Estructural  
♣Mixta

**DE10-Lite Board**

Terasic Technologies DE10-Lite Board ofrece una robusta plataforma de diseño de hardware construida alrededor del Arreglo de Puertas Programables en Campo (FieldProgrammable Gate Array o FPGA) de Altera MAX 10®. La placa DE10-Lite incorpora un USB-Blaster integrado, SDRAM, acelerómetro, salida VGA, conector de expansión GPIO 2×20, un convertidor analógico al digital integrado (ADC) y un conector de expansión Arduino UNO R3. Esta placa de desarrollo proporciona una solución ideal de 5 prototipado a nivel de sistema para aplicaciones industriales, automotrices, de consumo y  
muchas otras del mercado.

**Un módulo de comunicación UART TTL – USB FTDI FT 232**

Las interfaces de comunicación permiten al micro controlador intercambiar información digital con diversos dispositivos electrónicos, por ejemplo, otro micro controlador o una computadora.

**SISTEMA DE CONTROL DEL CARRITO**

1. Requerimientos:

El carrito se controlará haciendo uso del teclado de una computadora mediante un Shell y un cliente SSH. Para poder iniciar la sesión en línea de comandos y poder conectarnos al puerto desde el cual se están recibiendo y/o enviando los paquetes de datos (puerto donde se conectó el modulo) se hizo uso del cliente SSH y Telnet, ***PUTTY***. Con este podemos conectarnos a servidores remotos iniciando sesiones en ellos para poder ejecutar comandos. En este caso se utilizó para poder observar la salida de datos de nuestro programa.

Las teclas asignadas y sus funciones son las siguientes:

|  |  |  |  |
| --- | --- | --- | --- |
| FUNCION | TECLA | CABLE | PIN/SEÑAL |
| ADELANTE | W | CAFE | W8-fu |
| REVERSA | S | VERDE | W9-fd |
| IZQUIERDA | A | GRIS | W10-fl |
| DERECHA | D | NARANJA | V5-fr |
| STOP | F | ROJO | W8-fstop |

Las luces del automóvil al igual que los motores funcionaran de acuerdo con los siguientes requerimientos:

* Cuando el carrito avance hacia delante las luces de reversa se apagaran y solo los faros y demás luces permanecerán encendidas.
* Cuando el carrito avance hacia atrás, los faros se apagaran, y solo quedaran encendidas las luces de reversa y demás luces.
* Cuando el carrito de vuelta a la izquierda o a la derecha las luces del sentido correspondiente se encenderán y apagaran indicando el sentido del giro.

1. Diagrama de bloques

Sistema

Controlador

W

S

D

A

fu

fd

fr

fl

COMPUTADORA

TTL

1. Tabla de Verdad

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | ***fu*** | ***fd*** | ***fr*** | ***fl*** |
| **0** | **0** | **0** | **0** | 0 | 0 | 0 | 0 |
| **0** | **0** | **0** | **1** | 0 | 0 | 0 | 1 |
| **0** | **0** | **1** | **0** | 0 | 0 | 1 | 0 |
| **0** | **0** | **1** | **1** | 0 | 0 | 0 | 0 |
| **0** | **1** | **0** | **0** | 0 | 1 | 0 | 0 |
| **0** | **1** | **0** | **1** | 0 | 1 | 0 | 1 |
| **0** | **1** | **1** | **0** | 0 | 1 | 1 | 0 |
| **0** | **1** | **1** | **1** | 0 | 1 | 0 | 0 |
| **1** | **0** | **0** | **0** | 1 | 0 | 0 | 0 |
| **1** | **0** | **0** | **1** | 1 | 0 | 0 | 1 |
| **1** | **0** | **1** | **0** | 1 | 0 | 1 | 0 |
| **1** | **0** | **1** | **1** | 1 | 0 | 0 | 0 |
| **1** | **1** | **0** | **0** | 0 | 0 | 0 | 0 |
| **1** | **1** | **0** | **1** | 0 | 0 | 0 | 0 |
| **1** | **1** | **1** | **0** | 0 | 0 | 0 | 0 |
| **1** | **1** | **1** | **1** | 0 | 0 | 0 | 0 |

**4.- Código utilizado**

***BLOQUE RX***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.NUMERIC\_STD.ALL;

entity RX is

port( CLK : IN STD\_LOGIC;

LEDS : OUT STD\_LOGIC\_VECTOR(7 downto 0);

RX\_WIRE : IN STD\_LOGIC);

end entity;

architecture behaivoral of RX is

signal BUFF : STD\_LOGIC\_VECTOR(9 downto 0);

signal FLAG : STD\_LOGIC := '0';

signal PRE : INTEGER range 0 to 5208 :=0;

signal INDICE : INTEGER range 0 to 9 :=0;

signal PRE\_VAL : INTEGER range 0 to 41600;

signal BAUD : STD\_LOGIC\_VECTOR(2 downto 0);

begin

RX\_dato : process(CLK)

begin

if(CLK'EVENT and CLK = '1') then

if(FLAG = '0' and RX\_WIRE = '0') then

FLAG <= '1';

INDICE <= 0;

PRE <= 0;

end if;

if(FLAG = '1') then

BUFF(INDICE) <= RX\_WIRE;

if(PRE < PRE\_VAL) then

PRE <= PRE + 1;

else

PRE <= 0;

end if;

if(PRE = PRE\_VAL/2) then

if(INDICE < 9) then

INDICE <= INDICE + 1;

else

if(BUFF(0) = '0' and BUFF(9) = '1') then

LEDS <= BUFF(8 downto 1);

else

LEDS <= "00000000";

end if;

FLAG <= '0';

end if;

end if;

end if;

end if;

end process RX\_dato;

BAUD <= "011";

with(BAUD) select

PRE\_VAL <= 41600 when "000", -- 1200 bauds

20800 when "001", -- 2400 bauds

10400 when "010", -- 4800 bauds

5200 when "011", -- 9600 baudS

2600 when "100", -- 19200 bauds 1300 when "101", -- 38400 bauds

866 when "110", -- 57600 bauds

432 when others; --115200 bauds

end architecture behaivoral;

***BLOQUE PWM***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity PWM is

Port ( Reloj : in STD\_LOGIC;

D : in STD\_LOGIC\_VECTOR (7 downto 0);

S : out STD\_LOGIC);

end PWM;

architecture Behavioral of PWM is

begin

process (Reloj)

variable Cuenta : integer range 0 to 255 := 0;

begin

if Reloj='1' and Reloj'event then

Cuenta := (Cuenta + 1) mod 256;

if Cuenta < D then

S <= '1';

else

S <= '0';

end if;

end if;

end process;

end Behavioral;

***BLOQUE LEDS***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Leds is

Port ( clk : in STD\_LOGIC;

led1 : out STD\_LOGIC;

led2 : out STD\_LOGIC;

led3 : out STD\_LOGIC;

led4 : out STD\_LOGIC;

led5 : out STD\_LOGIC;

led6 : out STD\_LOGIC;

led7 : out STD\_LOGIC;

led8 : out STD\_LOGIC);

end Leds;

architecture Behavioral of Leds is

component divisor is

Generic ( N : integer := 24);

Port ( clk : in std\_logic;

div\_clk : out std\_logic);

end component;

component PWM is

Port ( Reloj : in STD\_LOGIC;

D : in STD\_LOGIC\_VECTOR (7 downto 0);

S : out STD\_LOGIC);

end component;

signal relojPWM : STD\_LOGIC;

signal relojCiclo : STD\_LOGIC;

signal a1 : STD\_LOGIC\_VECTOR (7 downto 0) := X"08";

signal a2 : STD\_LOGIC\_VECTOR (7 downto 0) := X"20";

signal a3 : STD\_LOGIC\_VECTOR (7 downto 0) := X"60";

signal a4 : STD\_LOGIC\_VECTOR (7 downto 0) := X"ff";

signal a5 : STD\_LOGIC\_VECTOR (7 downto 0) := X"00";

signal a6 : STD\_LOGIC\_VECTOR (7 downto 0) := X"00";

signal a7 : STD\_LOGIC\_VECTOR (7 downto 0) := X"00";

signal a8 : STD\_LOGIC\_VECTOR (7 downto 0) := X"00";

signal vuelta : integer range 0 to 255 := 0;

begin

R1: divisor generic map (10) port map (clk, relojPWM);

R2: divisor generic map (23) port map (clk, relojCiclo);

P1: PWM port map (relojPWM, a1, led1);

P2: PWM port map (relojPWM, a2, led2);

P3: PWM port map (relojPWM, a3, led3);

P4: PWM port map (relojPWM, a4, led4);

P5: PWM port map (relojPWM, a5, led5);

P6: PWM port map (relojPWM, a6, led6);

P7: PWM port map (relojPWM, a7, led7);

P8: PWM port map (relojPWM, a8, led8);

process (relojCiclo,vuelta)

variable Cuenta : integer range 0 to 255 := 0;

begin

if relojCiclo='1' and relojCiclo'event then

if vuelta < 4 then

a1 <= a8;

a2 <= a1;

a3 <= a2;

a4 <= a3;

a5 <= a4;

a6 <= a5;

a7 <= a6;

a8 <= a7;

vuelta <= vuelta + 1;

else

a1 <= a2;

a2 <= a3;

a3 <= a4;

a4 <= a5;

a5 <= a6;

a6 <= a7;

a7 <= a8;

a8 <= a1;

vuelta <= vuelta + 1;

if vuelta > 6 then

vuelta <= 0;

end if;

end if;

end if;

end process;

end Behavioral;

***BLOQUE DIVISOR***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_ARITH.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity Divisor is

Generic ( N : integer := 24);

Port ( clk : in std\_logic;

div\_clk : out std\_logic);

end Divisor;

architecture Behavioral of Divisor is

begin

process (clk)

variable cuenta: std\_logic\_vector (27 downto 0) := X"0000000";

begin

if rising\_edge (clk) then

cuenta := cuenta + 1;

end if;

div\_clk <= cuenta (N);

end process;

end Behavioral;

-- Periodo de la señal de salida en funcion del valor N para clk=50 MHz:

-- 27 ~ 5.37s, 26 ~ 2.68s, 25 ~ 1.34s, 24 ~ 671ms,

***BLOQUE CONTADOR***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity contador\_v1 is

PORT (

clk : IN STD\_LOGIC;

cnt\_out: OUT STD\_LOGIC\_VECTOR(3 DOWNTO 0)

);

end contador\_v1;

architecture Behavioral of contador\_v1 is

-- Señal temporal para el contador.

signal cnt\_tmp: STD\_LOGIC\_VECTOR(3 DOWNTO 0) := "0000";

begin

proceso\_contador: process (clk) begin

if rising\_edge(clk) then

if (cnt\_tmp < "1001") then

cnt\_tmp <= cnt\_tmp + 1;

else

cnt\_tmp <= "0000";

end if;

end if;

end process;

cnt\_out <= cnt\_tmp;

end Behavioral;

***BLOQUE CLK2HZ***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity clk2Hz is

Port (

clk : in STD\_LOGIC; -- Reloj de entrada de 50 MHz.

reset : in STD\_LOGIC;

clk\_out : out STD\_LOGIC -- Reloj de salida de 2.0000 Hz.

);

end clk2Hz;

architecture Behavioral of clk2Hz is

signal temporal: STD\_LOGIC;

signal contador: integer range 0 to 12499999 := 0;

begin

divisor\_frecuencia: process (clk, reset) begin

if (reset = '1') then

temporal <= '0';

contador <= 0;

elsif rising\_edge(clk) then

if (contador = 12499999) then

temporal <= NOT(temporal);

contador <= 0;

else

contador <= contador + 1;

end if;

end if;

end process;

clk\_out <= temporal;

end Behavioral;

**RESULTADOS FINALES**

Carrito en funcionamiento.





Conexiones del carrito:



**CONCLUSIÓN**

Durante el desarrollo del proyecto logramos implementar el sistema de control de un carrito a control remoto haciendo uso de los conocimientos adquiridos en clase acerca del control de motores DC, del módulo de comunicación UART-TTL, control de intensidad de led´s mediante un PWM. Adicional a esto se siguió todo el proceso para el desarrollo de un sistema digital, igualmente aprendido en clase.

Cabe señalar que la implementación de este proyecto se hizo pensando en el hecho de reutilizar y comprar lo menos posible por lo que, por ejemplo, para el armazón de la camioneta reusamos un carrito que ya no funcionaba y rescatamos los motores que tenía. Asimismo, reusamos material que teníamos de prácticas anteriores.

Es un sistema relativamente sencillo, pero si se llegará a desarrollar, es una base formidable para continuar con el proyecto y la idea ya está establecida de manera concreta.

**REFERENCIAS**

http://pdf1.alldatasheet.com/datasheet-pdf/view/64409/HOLTEK/HT12E.html

http://www.alldatasheet.com/datasheet-pdf/pdf/64409/HOLTEK/HT12D.html

http://www.ozitronics.com/data/rws3716c.pdf

http://www.es.co.th/Schemetic/PDF/TWS-BS3V1.01.PDF

Gindel. *Microcontroladores 4 – comunicación (UART)*. [online] Es.slideshare.net. Consultado en:  
https://es.slideshare.net/pablogindel/microcontroladores-4-comunicacin-uart [21 noviembre. 2018].

**ANEXO**

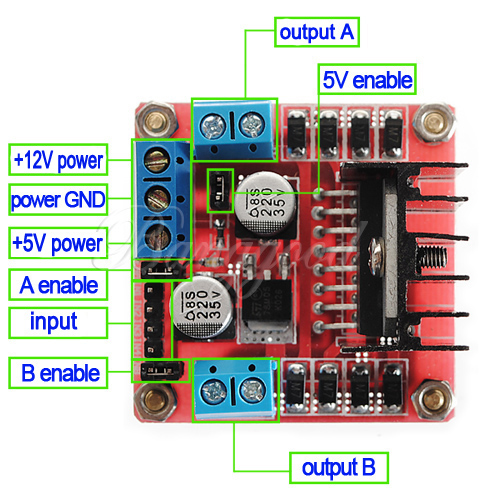
**DataSheets**

**MODULO Driver l298n**

**Características**

* Voltaje de alimentación, mínimo de 5 V. Posee dos entradas, una de 5V para controlar la parte lógica y otra para alimentar las salidas al motor, que pueden ser de 5V o más.
* La tarjeta tiene la opción de habilitar un regulador LM7805 integrado en ella para alimentar la parte lógica con lo que se puede alimentar la tarjeta con 12V por ejemplo.
* Corriente máxima 2 Amperios.
* Posee 6 entradas de control (ver tabla de control)
* Admite entradas de señal PWM para el control de velocidad.
* Dimensiones: 43 mm x 23,9 mm x 43 mm.
* Salidas: para 2 motores de DC o para un motor bipolar paso a paso.

## Partes



### ESPECIFICACIONES TÉCNICAS

* Chip: L298N
* Canales: 2 (soporta 2 motores DC o 1 motor PAP)
* Voltaje lógico: 5V
* Voltaje de Operación: 5V-35V
* Consumo de corriente (Digital): 0 a 36mA
* Capacidad de corriente: 2A (picos de hasta 3A)
* Potencia máxima: 25W
* Peso: 30g
* Dimensiones: 43 \* 43 \* 27 mm
* Voltaje de alimentación, mínimo de 5 V. Posee dos entradas, una de 5V para controlar la parte lógica y otra para alimentar las salidas al motor, que pueden ser de 5V o más.
* Posee un regulador de voltaje de 5V 78M05, para alimentar la etapa lógica del L298N, sin embargo, cuando la alimentación supera los 12V, se recomienda, utilizar una fuente de 5V externa como fuente de alimentación.
* Admite entradas de señal PWM para el control de velocidad.
* Posee 8 diodos de protección contra corriente inversas.

Este módulo se puede alimentar de 2 maneras gracias al regulador integrado LM7805. Cuando el jumper de selección de 5V se encuentra activo, el módulo permite una alimentación de entre 6V a 12V DC. Como el regulador se encuentra activo, el pin marcado como +5V tendrá un voltaje de 5V DC. Este voltaje se puede usar para alimentar la parte de control del módulo ya sea un microcontrolador o un Arduino, pero recomendamos que el consumo no sea mayor a 500mA.

Cuando el jumper de selección de 5V se encuentra inactivo, el módulo permite una alimentación de entre 12V a 35V DC. Como el regulador no está funcionando, tendremos que conectar el pin de +5V a una tensión de 5V para alimentar la parte lógica del L298N. Usualmente esta tensión es la misma de la parte de control, ya sea un microcontrolador o Arduino.

**L298n**

