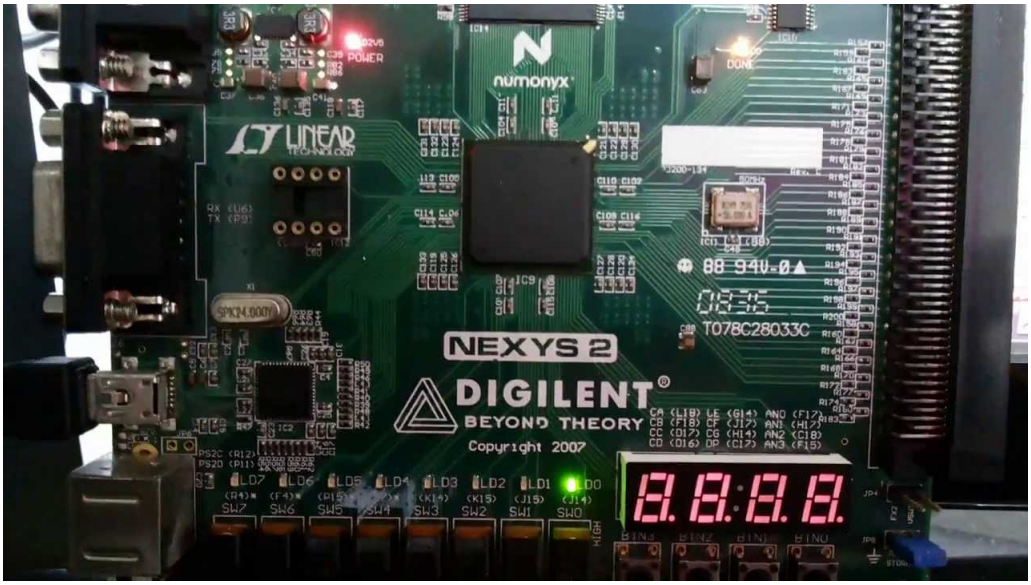
Desarrollo de un Turnero Digital Usando FPGA's

Laboratorio de Electrónica Digital
Reporte Final

Juan Francisco Gallego Leon
CC. *1088336670*
Héctor F. Jiménez Saldarriaga
CC. *1115421345*
Sebastian Zapata Ruiz
CC. *1088313682*

Prf. Ramiro Andrés Barrios Valencia



Facultad de Ingenierías,
Ingeniería de Sistemas y Computación

Universidad Tecnológica de Pereira

2016-2

Pagina Dejada Intencionalmente en Blanco

Índice

Héctor F. Jiménez S.
hfjimenez@utp.edu.co

Sebastian Zapata
sebastzapata93@gmail.com

Juan Francisco Gallego
juanfran16@utp.edu.co

16 de diciembre de 2016

Resumen

En el presente documento se refleja el trabajo realizado para llevar a cabo el generador de turnos, el desarrollo de este proyecto fue llevado a cabo en 5 etapas durante el transcurso del semestre, el objetivo principal que se buscaba con este, era construir un dispositivo de apoyo a las entidades que tiene que atender demasiadas personas simultaneas. Para poder lograrlo durante el semestre los integrantes cada uno se encargaron de tareas y módulos específicos facilitando que cada integrante desarrollara sus fortalezas.

La arquitectura de los proyectos es la siguiente:

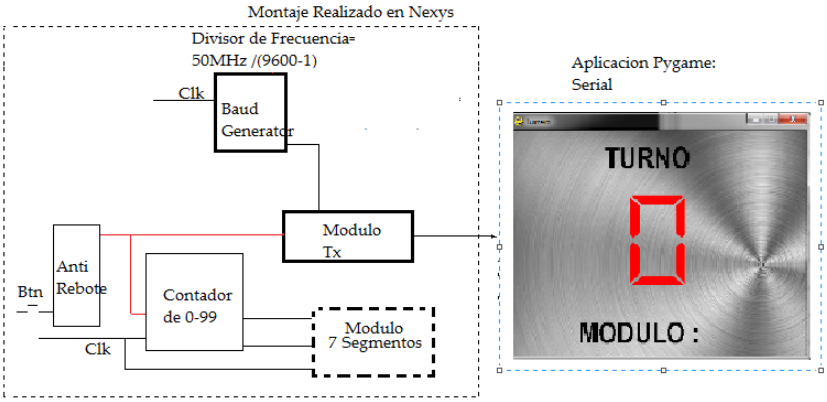


Figura 1: Generador de Turnos, esquema general.

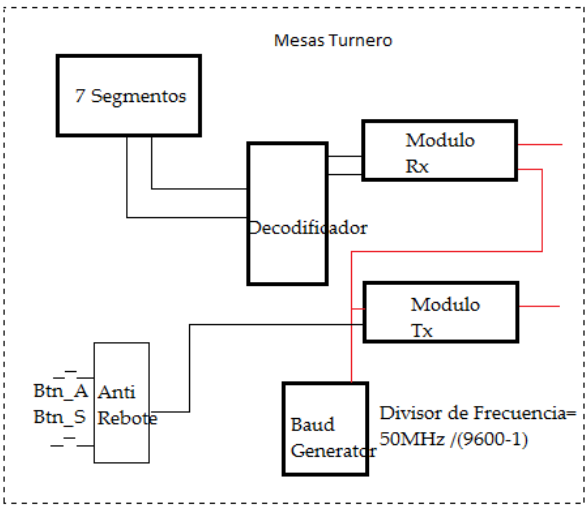


Figura 2: Mesas del turnero.

1. Modulo 1: Anti-rebote

Utilizar switches y pulsadores mecánicos en un sistema electrónico es una práctica muy común, pues sirven de interfaces entre el usuario y el sistema embebido para seleccionar y programar funciones que este posee; en la práctica actual muchos de estos sistemas mecánicos están siendo reemplazados por displays táctiles pero hay situaciones industriales principalmente en las cuales un pulsador mecánico sería una solución más apropiada y eficiente en la implementación, pues al ser un sistemas mecánicos estos no se degradan tan rápido como lo harían las pantallas táctiles en un entorno industrial. Una de las grandes desventajas que poseen los pulsadores son los rebotes o saltos como se puede observar en la figura 3.

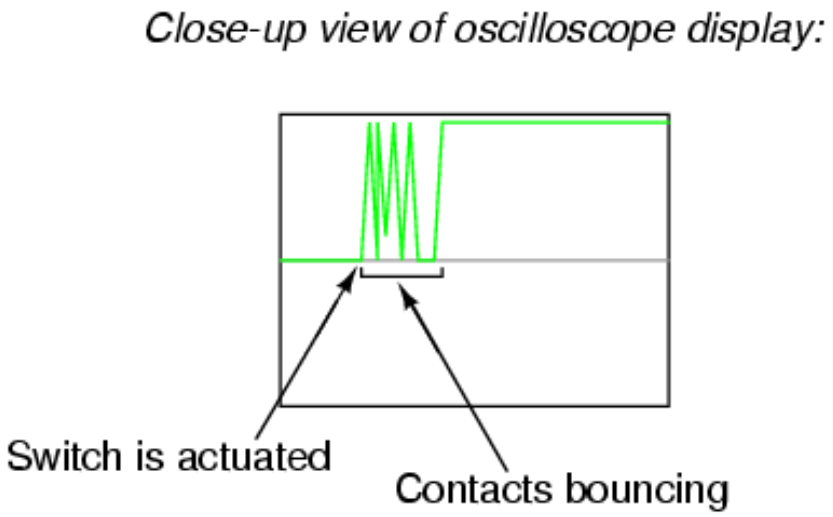


Figura 3: Rebotes al presionar un pulsador mecánico, señales de osciloscopio.

Los elementos mecánicos del pulsador están constituidos por un par de contactos eléctricos que se unen o separan por medios mecánicos, en la figura 4 se muestra un pulsador común, figuras5 6 son vista detalladas. Los falsos contactos que se producen cuando los accionamos, crean falsas señales instantáneas, ocasionando un mal funcionamiento del dispositivo, no olvidando mencionar que existe una barrera o GAP que se produce por el aire entre los elementos mecánicos*(aunque típica mente esto es despreciable por su baja fuerza discipativa)* induciendo múltiples pulsaciones o flancos como se muestra en la figura 3.

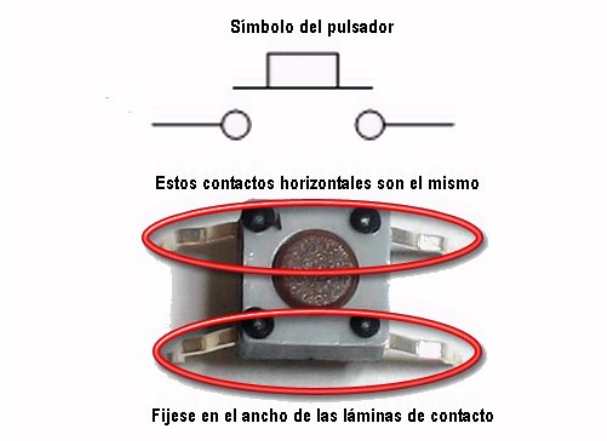


Figura 4: Pulsador Común, Composición.

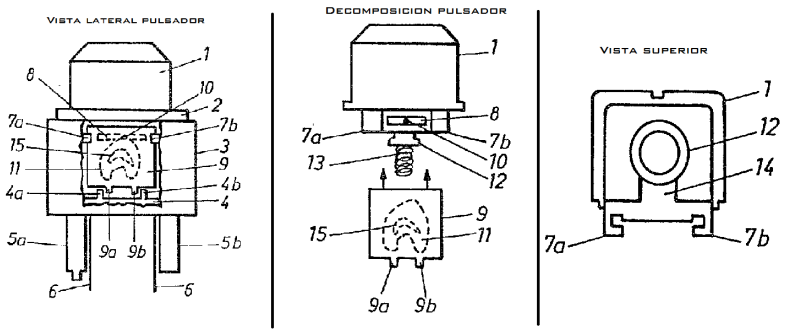


Figura 5: Decomposicion de un Pulsador común

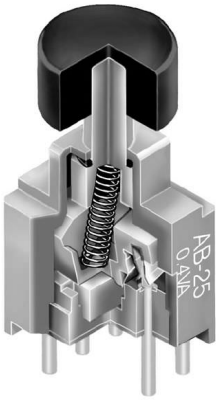


Figura 6: Vista en 3d de un pulsador y sus elementos

En algunos casos esta acción puede producir una chispa debido a la corriente que atraviesa los contactos, disminuyendo el tiempo útil de los contactos eléctricos. La chispa se produce siempre al separar los contactos (desconectar), en ocasiones parece que también salta al conectarlos, eso es debido a los rebotes mecánicos que se producen al cambiar de estado. Existen muchas formas de lidiar con este problema, tanto soluciones por hardware físico como el circuito simple de la figura 7 o por software como lo realizaremos para el desarrollo de este modulo, mediante la implementación de un circuito lógico provisto por la empresa Digikey.[1]

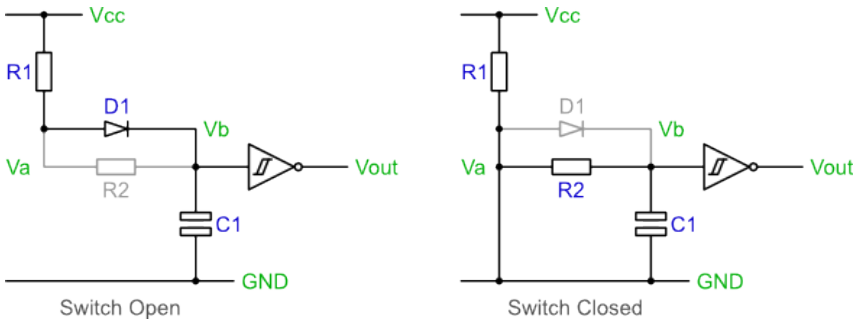


Figura 7: Solución utilizando un simple arreglo electrónico. Pulsadores con resistencias de Pull Ups.

1.1. Arquitectura de la Placa *NEXYS 2*

La tarjeta de entrenamiento *NEXYS 2* es una plataforma de desarrollo fabricada por la empresa [2] aunque se encuentra descontinuada para la venta y recomiendan utilizar esta misma en su versión **4**, utiliza una fpga Spartan 3E optimizada para desarrollo de aplicaciones donde se requiere implementar

[1] http://www.digikey.com/
[2] Digilentinc Página Oficial Nexys2 Spartan3E[3]

diseños de lógica compleja, e ideal para el procesamiento de señales y desarrollo de sistemas embebidos como se menciona en General Spartan Versions[4], Spartan-3E FPGA Family: Introduction and Ordering Information[5].
Algunos elementos destacables de la placa son :

Connectores

- Puerto USB2.0
- Puerto VGA
- PS/2
- Puerto RS232

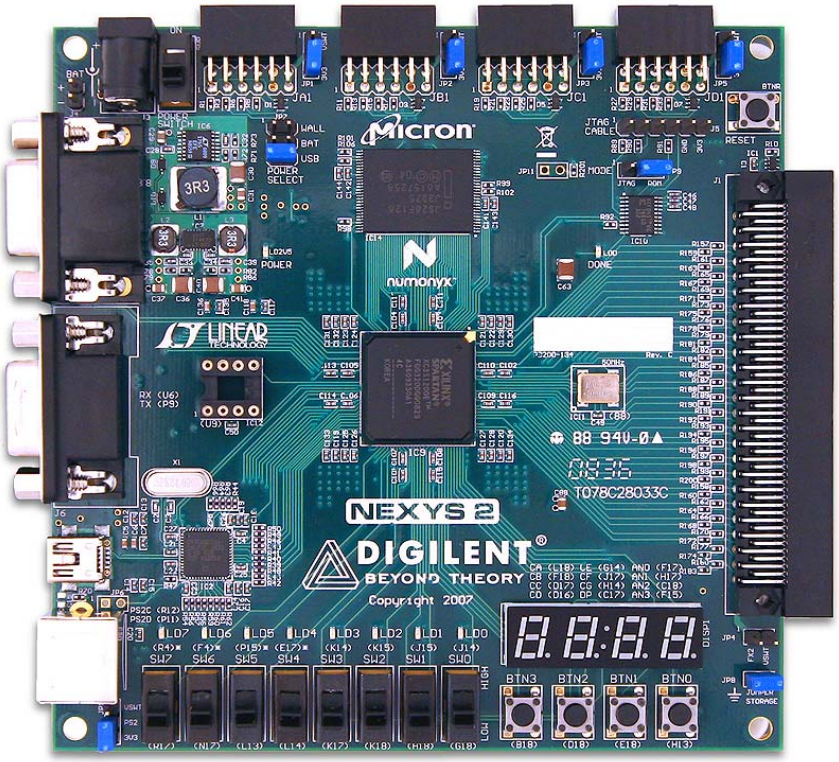


Figura 8: Modelo de la Nexxys , Versión 2.

Algunas de las que más nos llamaron la atención son:

Caracteristicas

- Xilinx Spartan-3E FPGA 500K
- Memoria PSDRAM de 16 MB fast Micron®
- Memoria de 16 MB Intel® StrataFlash® Flash R
- Trabaja con la version Free de ISE®/WebPACK
- Oscilador de 50 MHz
- Fuentes reguladas de 3.3V@3A/100mA(principal),3.3V@150mA/60mA,2.5V/1.2V@1.4A/50mA
- Todas las entradas tiene protección contra cortocircuitos y descargas electroestática
- Incluye 8 leds, cuatro display siete segmentos, cuatro pulsadores, 8 switches...

[4] http://www.xilinx.com/support/documentation-navigation/silicon-devices/mature-products/spartan-3e.html
[5] http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

Para plantear una solución al problema mencionado en **??**, nosotros hemos descargado el esquemático para comprender circuitalmente cuál es la configuración de los pulsadores, aunque generalmente para desarrollos importantes en electrónica, se compran pulsadores de alta calidad, y resistencia mecánica a presiones momentáneas e instantáneas, pero dado que esto es una placa de entrenamiento, *estudiantil* asumimos que los pulsadores que esta posee son netamente mecánicos que cuenta con el diseño de la figura 9

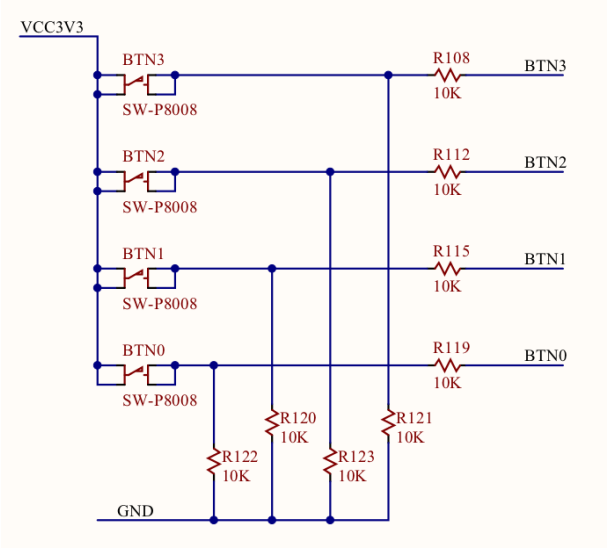


Figura 9: Esquema provisto en el esquemático, 4 pulsadores.

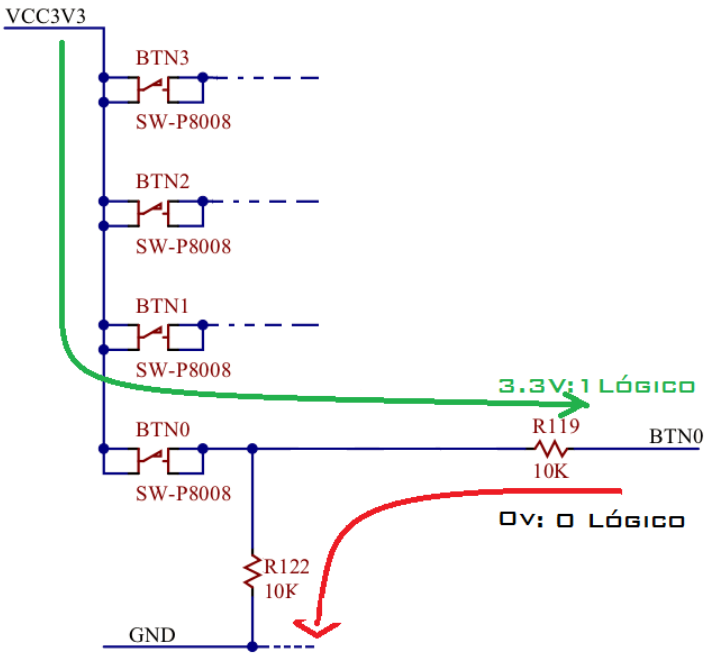


Figura 10: Circulación de la corriente en el circuito, resistencia de 10kΩ protección contra cortocircuito

La figura 10 también muestra una configuración de un solo pulsador vemos la resistencia de pull down. Con colores hemos indicado el accionamiento del pulsador, tome como ejemplo 10, el color rojo indica cuando el pulsador **BTN0** no ha sido presionado, esto indica que los pulsadores son de tipo **N.O** ó *Normally Open in Low Mode* presentando un **0 lógico** a la salida, al presionar el pulsador, el contacto interno que este posee cierra el circuito conectándolo con la fuente de 3.3v como lo muestra el color verde de la figura 10.

1.2. Anti rebote por Software

Resolver el problema de los rebotes de un pulsador puede ser una tarea fácil o compleja, la idea básica es *muestrear* la señales de entrada de los pulsadores en un intervalo regular de tiempo para notar si hay cambios en la entrada, si no existieron cambios se debe mantener el estado actual del pulsador, si hay pulsos anormales debemos filtrar la señal. Después de realizar los cálculos correspondientes lo que decidimos implementar fue tomar un tiempo regular la señal producida por el pulsador para saber cuando la señal es verdadera, así que después de 5 millones de ciclos de reloj que corresponden aproximadamente a un tiempo de 110ms, si la señal se mantiene dejaremos pasar la señal del pulsador por la salida *Q*, El modulo comentado se encuentra descrito en la seccion 1.2.1

Contador Este contador nos dirá cuanto tiempo ha pasado desde que la señal ha estado en alto, si la señal ha estado en alto durante un tiempo definido por nosotros *(50 mS)*, entonces es considerado como un pulso estable, y debería de setear la salida a 1. Esta aproximación es parecida vista en la implementación de DigiKey[6] pero más completa.

Registro 8bits Esta aproximación es similar a la mencionada anteriormente, pero utiliza un registro de desplazamiento de 8bits en vez de un contador, lo que hace es ir desplazando los bits hasta alcanzar su valor máximo.

1.2.1. Implementacion antirebote Vhdl

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counters_1 is
    port(CLK,btn_in: in std_logic;
         Q : out std_logic);
end counters_1;

--11100100111000011100000

architecture archi of counters_1 is
signal counter: std_logic_vector (23 downto 0) := (others => '0');

begin
    process (CLK, btn_in)
    begin
        if (CLK'event and CLK='1') then
                                Q <= '0';
                                IF counter > 0 and btn_in = '0' then
                                                        counter <= counter + 1;
                                                    end if;
                        IF (btn_in='1') THEN
                                            IF (counter >= 5000000) THEN
                                                        Q <= '1';
                                                                counter <= (others =>'0');
                                    ELSE
                                                counter <= counter + 1;
                                    END IF;
                                END IF;
                          end if;
end process;
end archi;
```

Listing 1: Implementacion Antirebote.

[6] https://eewiki.net/pages/viewpage.action?pageId=4980758

2. Modulo 2: Display de Siete Segmentos

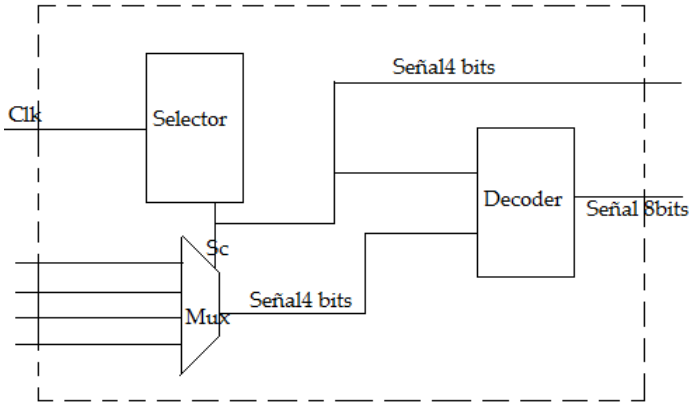


Figura 11: Estructura del Modulo de 7 Segmentos.

Para desarrollar el modulo número dos del proyecto lo primero que realizamos es verificar en la hoja del fabricante las conexiones y el tipo de configuración utilizada en la placa de desarrollo **Nexys2**. Al revisar el Datasheet y manual encontramos que la placa contiene 4 displays de 7 segmentos y punto decimal(dp) en la configuración de ánodo común lo cual implica que todos los ánodos de los leds que conforman el display este conectados a un punto en común como se aprecia en la figura 12 , para encender un solo led necesitaremos un valor lógico de ***0*** para cerrar el circuito.

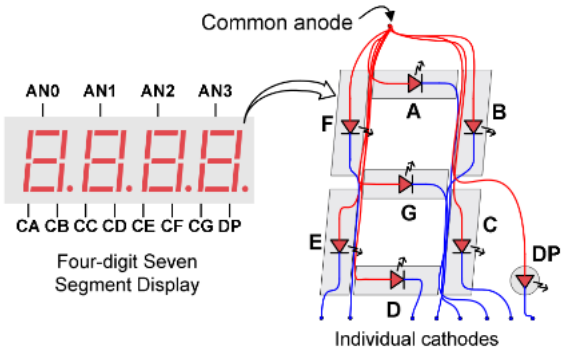


Figura 12: Esquema Cátodo Común.

Revisando el esquemático de la tarjeta de desarrollo sabemos que la activación de los displays se da por la saturación del transistores pnp como se observa en la siguiente figura 13 :

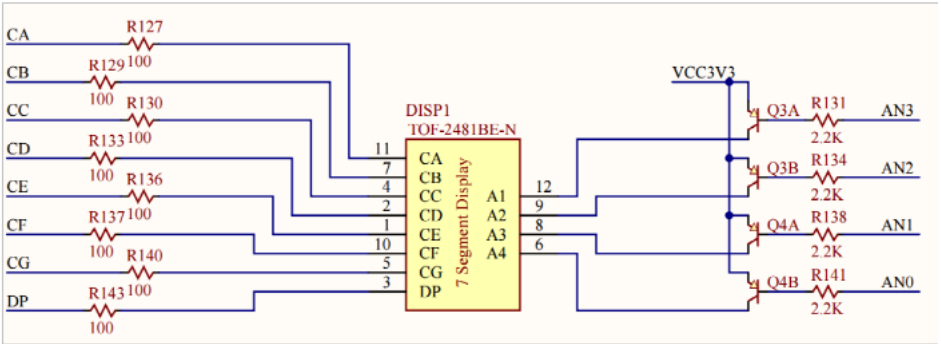


Figura 13: Esquema provisto en el esquemático, 4 pulsadores.

Los displays de 7 segmentos se encuentran en la configuración de ánodo común, un **0** se utiliza para encendido y un **1** para apagado (lógica negativa). Al representar los números usaremos una codificación de 4 bits que permiten los 10 dígitos (del *0 al 9*),y así poder mostrar el dígito en un visualizador de

siete segmentos siendo necesario decodificar el valor numérico según el patrón de LEDs indicado, se tiene un carácter al cual se le asigna un valor numérico (codificación) para tratar con el digitalmente, posteriormente se transforma al patrón de LEDs correspondiente (de codificación) como se puede observar en el listing 2. El número que se recibe en 4 bits codificado se decodifica a su valor correspondiente donde prenderemos o apagaremos los 8 leds del display.

2.0.1. Implementacion Decodificador vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity decoder_arch is
    Port ( numero : in  STD_LOGIC_VECTOR(3 downto 0);
           segment : out  STD_LOGIC_VECTOR(7 downto 0));
end decoder_arch;

architecture Behavioral of decoder_arch is

begin
        process(numero) begin
        case numero is
                when "0000" => segment <= "00000011"; --0
                when "0001" => segment <= "10011111"; --1
                when "0010" => segment <= "00100101"; --2
                when "0011" => segment <= "00001101"; --3
                when "0100" => segment <= "10011001"; --4
                when "0101" => segment <= "01001001"; --5
                when "0110" => segment <= "01000001"; --6
                when "0111" => segment <= "00011111"; --7
                when "1000" => segment <= "00000001"; --8
                when "1001" => segment <= "00001001"; --9
                when others => segment <= "11111111"; -- Nada
        end case;
end process;
end Behavioral;
```

Listing 2: Decodificador de 4 bits.

DECIMAL	BINARIO
0	0001
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Cuadro 1: Valores Correspondientes de Binario a Bcd

Una vez hemos construido el modulo que se encarga de decodificar el valor que queremos mostrar en los displays necesitamos saber como seleccionar el display adecuado; para esto utilizamos un multiplexor de 4 bits. un multiplexor o selector de datos es un circuito lógico que acepta varias entradas y solamente permite a una de ellas alcanzar la salida. La figura 14 muestra el diagrama de un multiplexor, donde se observa que la salida Z puede tomar el valor de A o B, pero no de ambos a la vez, en base al valor del parámetro de selección $S0$. Nosotros hemos escrito el código que se encarga de recibir cual sera el modulo que seleccionemos ***(selectorin)*** una vez hecho esto lo que nosotros tenemos que hacer es habilitar con un 0 el display requerido y pasarle el valor decodificado El listing 5

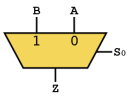


Figura 14: Multiplexor 2 a 1.

2.0.2. Implementacion Multiplexor vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux7s_arch is
    Port ( d0 : in  STD_LOGIC_VECTOR(3 downto 0);
           d1 : in  STD_LOGIC_VECTOR(3 downto 0);
           d2 : in  STD_LOGIC_VECTOR(3 downto 0);
           d3 : in  STD_LOGIC_VECTOR(3 downto 0);
           selectorin : in  STD_LOGIC_VECTOR(3 downto 0);
           numero : out  STD_LOGIC_VECTOR(3 downto 0));
end mux7s_arch;

architecture Behavioral of mux7s_arch is
begin
        process(selectorin,d0,d1,d2,d3) begin
                case selectorin is
                        when "1110" =>
                                numero<=d0;
                        when "1101" =>
                                numero<=d1;
                        when "1011" =>
                                numero<=d2;
                        when "0111" =>
                                numero<=d3;
                        when others =>
                                numero<=d0;
                end case;
        end process;

end Behavioral;
```

Listing 3: Multiplexor usado en proyecto final.

Pero existe un problema y es que nosotros solo podremos visualizar un display a la vez, es decir tendremos que hacer un refresco por cada display de al menos **4ms** para engañar la retina del ojo humano utilizando la frecuencia correcta, se llega a la ilusión de que hay cuatro dígitos distintos y encendidos al mismo tiempo si son 4 displays en total sera **16ms**. Este efecto es similar al comentado por el profe Ramiro donde un vídeo esta compuesto de muchas imágenes sucesivas y a cierta frecuencia el ojo ve en movimiento. Una frecuencia de **250kHz** es más que suficiente para generar este efecto, por lo cual se utiliza el divisor creado en el modulo uno VHDL, que se presenta a continuación.

2.0.3. Implementacion de Modulo Selector Vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity selector_arch is
    Port ( clk : in  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR(3 downto 0):="1110");
end selector_arch;
architecture Behavioral of selector_arch is

signal contador : integer range 0 to 200000 := 0;
signal ciclos : integer range 0 to 3 :=0;

begin
        process(clk) begin
                if rising_edge(clk) then
                        if contador = 200000 then
                                if ciclos = 0 then
                                        salida<="1101";
                                        ciclos<=ciclos+1;
                                        end if;
                                if ciclos = 1 then
                                        salida<="1011";
                                        ciclos<=ciclos+1;
                                        end if;
                                if ciclos = 2 then
                                        salida<="0111";
                                        ciclos<=ciclos+1;
                                        end if;
                                if ciclos = 3 then
                                        salida<="1110";
                                        ciclos<=0;
                                        end if;
                                contador<=0;
                        else
                                contador<= contador+1;
                        end if;
                end if;
        end process;
end Behavioral;
```

Listing 4: Modulo Selector,

2.0.4. Integración Submodulos Siete Segmentos

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity siete_segmentos is
    Port ( clk : in std_logic;
                          d0 : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           d1 : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           d2 : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
           d3 : in  STD_LOGIC_VECTOR(3 DOWNTO 0);
                          salidacontrol : out std_logic_vector(3 downto 0);
                          salidanumero  : out std_logic_vector(7 downto 0));
end siete_segmentos;

architecture Behavioral of siete_segmentos is

component selector_arch is
    Port ( clk : in  STD_LOGIC;
           salida : out  STD_LOGIC_VECTOR(3 downto 0):="1110");
end component;

component mux7s_arch is
    Port ( d0 : in  STD_LOGIC_VECTOR(3 downto 0);
           d1 : in  STD_LOGIC_VECTOR(3 downto 0);
           d2 : in  STD_LOGIC_VECTOR(3 downto 0);
           d3 : in  STD_LOGIC_VECTOR(3 downto 0);
           selectorin : in  STD_LOGIC_VECTOR(3 downto 0);
           numero : out  STD_LOGIC_VECTOR(3 downto 0));
end component;

component decoder_arch is
    Port ( numero : in  STD_LOGIC_VECTOR(3 downto 0);
           segment : out  STD_LOGIC_VECTOR(7 downto 0));
end component;

signal salidaselector : std_logic_vector(3 downto 0);
signal salidamux : std_logic_vector(3 downto 0);
signal salidadecoder : std_logic_vector(7 downto 0);

begin
        modulo1: selector_arch port map(
                                clk=>clk,
                                salida=>salidaselector
                                );
        modulo2: mux7s_arch port map(
                                d0=>d0,
                                d1=>d1,
                                d2=>d2,
                                d3=>d3,
                                selectorin=>salidaselector,
                                numero=>salidamux
                                );
        modulo3: decoder_arch port map(
                                numero=>salidamux,
                                segment=>salidadecoder
                                );
        salidacontrol<=salidaselector;
        salidanumero<=salidadecoder;

end Behavioral;
```

Listing 5: x.

Físicamente para realizar el impacto sobre la fpga debemos especificar cuales son las conexiones físicas en la fpga[7], ademas también están especificadas en la tarjeta. véase figura 15 , figura 16.

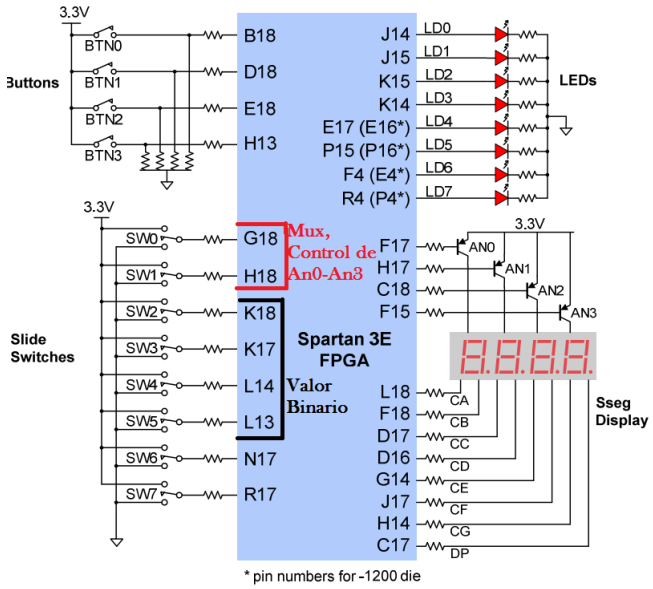


Figura 15: Esquema provisto en el esquemático, 4 pulsadores.

Figura 16: Esquema provisto en el esquemático, 4 pulsadores.

2.1. Errores Comunes

En este modulo tuvimos muchos problemas, apesar de ser sencillo en la fiscalización algunas veces siempre nos aparecía conexiones indefinidas y desconectadas. Toco realizar una revisión exaustiva de todo el código, para determinar por que no se conectaban. Tiempo después al querer impactar en la fpga olvidamos pinear la señal mas importante de todas que es la que sincroniza todas las operaciones, hablamos de la señal del CLK.

[7]Este fue uno de nuestros problemas al impactar pues al momento de mapear los pines reales olvidamos pinear nuestra señal de reloj.

3. Modulo 3: Protocolo RS232

En la actualidad existen una gran cantidad de formas de comunicación que hacen que sea posible el intercambio de datos entre 2 o más dispositivos electrónicos, para que esto suceda los dispositivos deben elegir un protocolo el cual permiten establecer los parámetros de la misma. El protocolo *Rs232* también conocido como EIA/TIA RS-232C designa una norma para el intercambio de datos binarios que se enviaran en forma serial, este tipo de comunicación en serie implica el envío de una serie de pulsos digitales de ida y vuelta entre dispositivos a una velocidad de transferencia mutuamente acordada.

Con el fin de hacer una comunicación serial, los dos dispositivos deben ponerse de acuerdo en los siguientes parámetros:

1. Velocidad a la que se envía, y leen los datos.
2. Los niveles de tensión que representan un 1 o un 0.
3. El significado de los niveles de tensión; Nivel de tensión alto es un **1** y un nivel de tensión bajo representa un **0**, o lógica inversa.

Figura 17: Interfaz Serial Usb-Db9.

Para la comunicación entre ambos, ademas de los parámetros mencionados anteriormente es necesario usar una interfaz serial, en el mercado hay varias soluciones que se adaptan dependiendo el tipo de puerto y entrada por la cual enviaremos, como se muestra en la figura 17 la solución también se puede embeber en el mismo cable.

Una interfaz rs232 físicamente requiere :

- Una conexión a tierra común, ambos dispositivos tienen un punto de referencia.
- Un cable para el remitente para enviar datos al receptor (línea de transmisión para el remitente conocido como **TX**).
- Un cable para el remitente para recibir datos (línea de recepción conocido como **RX**), véase la figura18. SparkFun[8].

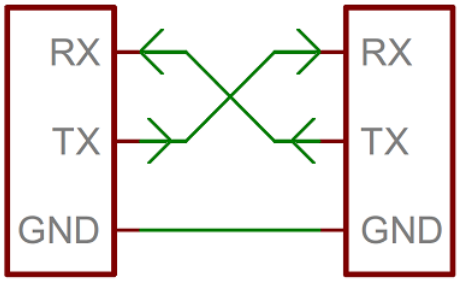


Figura 18: Hardware necesario para conexión asíncrona.

Para realizar el envió de comunicación el protocolo rs232 sigue una maquina de estado que se resume en los siguientes pasos:

1. El transmisor envía un **ïdle"(="1")** cuando esta en reposo.
2. El transmisor enviara un bit de inicio **"start"(="0")** que indica el comienzo de la transmisión de datos.
3. Se envían los 8 bits de información

[8] https://learn.sparkfun.com/tutorials/serial-communication

4. El transmisor envía un bit que indica la parada de cada byte.**"stop"(="1")** [9]

Figura 19: Hardware necesario para conexion asincrona.

Para desarrollar este modulo se crearon dos partes , la del receptor y el transmisor, cada uno se encarga de su tarea de enviar los datos serializado, recibirlos. El código implementado se presenta en las secciones de código 6.

[9] Véase la figura 20

3.0.1. Implementacion submodulo de transmisión Vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tx_arch is
        Port (  clk : in  STD_LOGIC;
                          clk_enable: in STD_LOGIC;
           tx_start : in  STD_LOGIC:='0';
           tx_data : in  STD_LOGIC_VECTOR(7 downto 0);
                          tx : out  STD_LOGIC:='1');
end tx_arch;
architecture Behavioral of tx_arch is

type state is (idle,data,parity_state,stop1,stop2);
signal tx_state : state:=idle;
signal tx_data_temp    : std_logic_vector(7 downto 0);
signal tx_parity_bit   : std_logic;
signal tx_data_counter : integer range 0 to 7:=0;
constant uart_start  : std_logic :='0';
constant uart_idle   : std_logic :='1';

begin
        process(clk,clk_enable) begin
                if rising_edge(clk) then
                        if clk_enable = '1' then
                                case tx_state is
                                        when idle=>
                                                if tx_start = '1' then
                                                        tx <= uart_start;
                                                        tx_data_temp<=tx_data;
                                                        tx_state <= data;
                                                        tx_parity_bit <='0';
                                                else
                                                        tx<=uart_idle;
                                                end if;
                                        when data=>
                                                tx <= tx_data_temp(tx_data_counter);
                                                tx_parity_bit <= tx_parity_bit xor tx_data_temp(tx
                                                if tx_data_counter = 7 then
                                                                tx_state<=parity_state;
                                                                tx_data_counter<=0;
                                                else
                                                        tx_data_counter<=tx_data_counter+1;
                                                end if;
                                        when parity_state=>
                                                tx <= tx_parity_bit;
                                                tx_state <=stop1;
                                        when stop1=>
                                                tx <=uart_idle;
                                                tx_state<=stop2;
                                        when stop2=>
                                                tx <=uart_idle;
                                                tx_state<=idle;
                                end case;
                        end if;
                end if;
        end process;

end Behavioral;
```

Listing 6: Modulo de Transmision TX uart.

4. Modulo 4: Aplicacion de Escritorio

En este modulo desarrollamos el aplicativo utilizando como lenguaje de programación *python* que posee soporte para el manejo de comunicaciones utilizando el protocolo rs232, la librería estándar se llama ***pyserial***, específicamente para nuestro desarrollado utilizamos la siguientes opciones:

1. 8 bits **serial.EIGHTBITS**
2. 2 bits de parada **serial.STOPBITS_TWO**
3. paridad impar **serial.PARITY_ODD**
4. velocidad de **9600** baudios.

para nosotros simular el envió de comunicación serial lo realizamos en dos sistemas operativos para windows utilizamos un emulador de puertos *Virtual Serial Port en modo pair* que nos permite crear virtualmente puertos/dispositivo que utilizara el protocolo RS232. En Gnu-Linux utilizamos **socat**, un emulador multipropuesta que permite crear dispositivos de comunicaciones virtuales unidireccionales. la forma de crear dos dispositivos en Linux es :

```
socat -d -d  pty,raw,echo=0 pty,raw,echo=0
```

Listing 7: Creacion de dos dispositivos ubicados en **/dev/pts/2, /dev/pts/3**.

En este sistema se debe agregar las siguientes librerías a pyserial para poder abrir y acceder al puerto serial : **rtscts=True,dsrdtr=True**

La idea principal de este modulo era simular como seria la integración entre el hardware y el aplicativo que se desarrollo, nosotros el aplicativo lo dividimos principalmente en 3 módulos, escritos en 3 scripts diferentes de python que son :

1. **Generador de Turnos**: Es un script en python que se encarga de generar los turnos solicitados por el usuario
2. **ModuloA,B**: Es es el aplicativo al que tienen acesso los asesores o mesas que resuelve el turno.
3. **Visualizador de Turnos(*Turnero*)**: Es el que se encarga de mostrar el turno actual en pantalla para el publico.

Para la interfaz gráfica se utilizo las librerías que dispone python como **pygame** el cual nos permite de una manera simple y rápida crear ventanas y entornos.

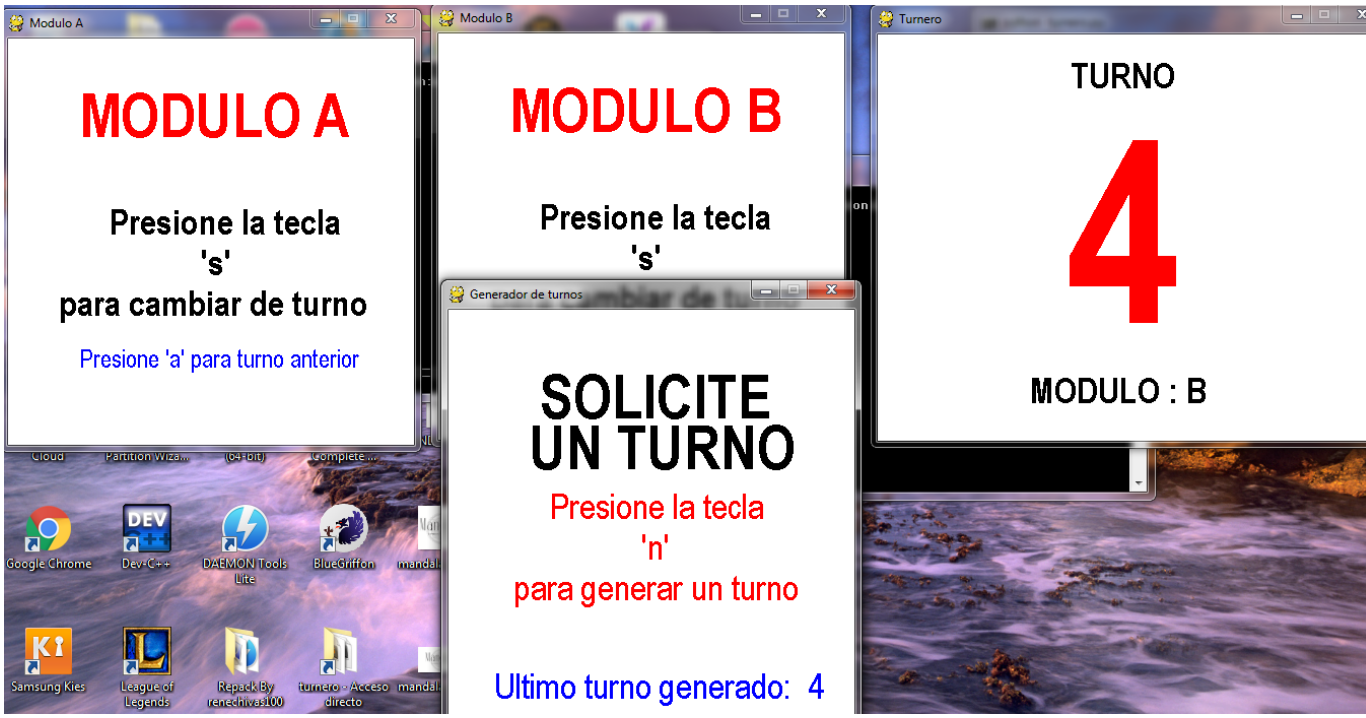


Figura 20: Esquema de Conexión Aplicación Web.

```
# -*- coding: utf-8 -*-
import pygame                   #Importar libreria necesaria
import serial
ANCHO=400                           #Dimensiones de la ventana
ALTO=400
BLANCO=(255,255,255)#constante de colores
ROJO=(255,0,0)
VERDE=(0,255,255)
AZUL=(0,0,255)
NEGRO=(0,0,0)

pygame.init()                   #objeto que inicializa el entorno
pantalla=pygame.display.set_mode([ANCHO,ALTO])#creamos ventana
pygame.display.set_caption("Generador De Turnos")
```

Listing 8: Creación de ventanas en todos los scripts.

Teniendo en cuenta esto para cada script lo que nosotros hemos realizado en nuestra aplicación simulada es que el usuario una vez presione la tecla **n** utilizamos la opción de eventos que ofrece pygame y enviamos esto por serial a nuestro visualizador.

```
        for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                                fin=True
                        elif event.type ==        pygame.KEYUP:
                                if event.key == pygame.K_n:
                                        if activado==0:
                                                #serial
                                                x=s.write(turn)
                                                print ("Envio : -- nuevo turno ")
                                                #pygame
                                                activado=1
                                                turnos+=1
                                                if turnos==100:
                                                        turnos=1
```

Listing 9: Gestión de eventos, presión de teclas .

Una vez se detecta el evento de la tecla enviamos esto el numero de turno generado.

Figura 21: Turnero.

5. Anexos

Los archivos adjuntos a este informe se encuentran en el repositorio del grupo.[10] junto a su informe, su código se encuentra en este archivo en la sección de integración.

Los archivos en LaTeXcorrespondientes también se encuentran allí respecto a cada modulo desarrollado. La figura 22 presenta la estructura que se encuentra en el repositorio.

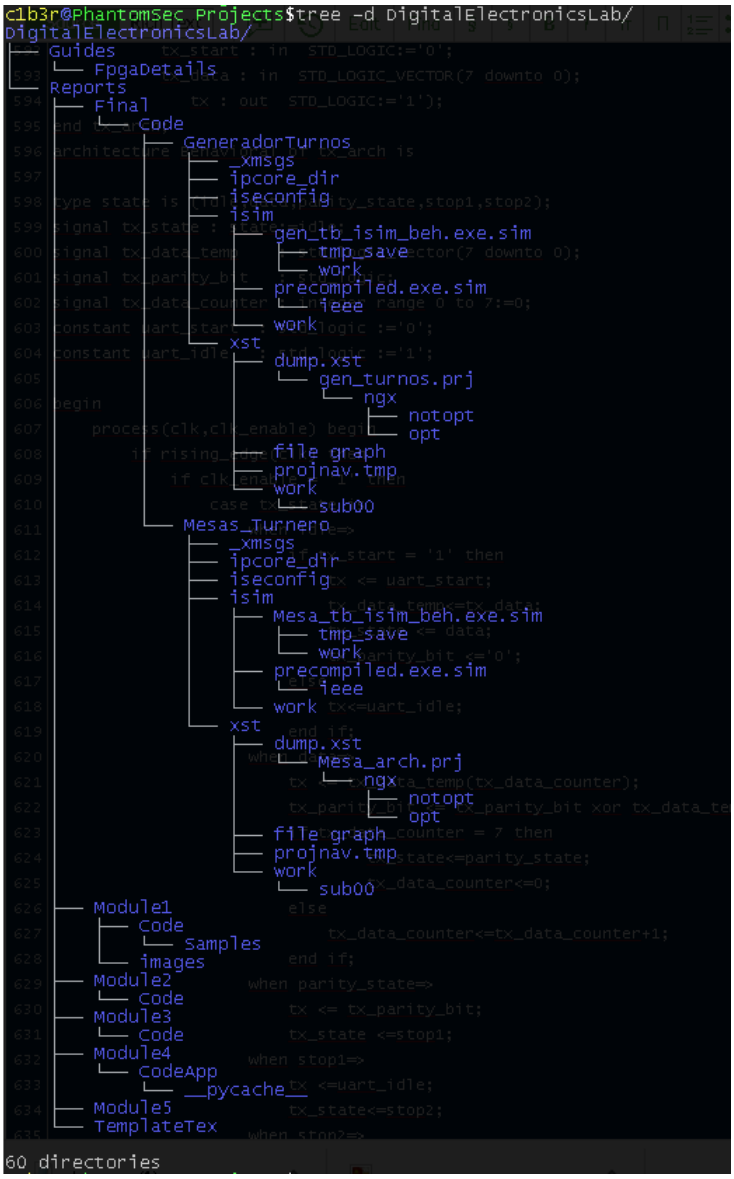


Figura 22: Árbol de Archivos Repositorio Github.

[10] https://github.com/heticor915/DigitalElectronicsLab/tree/master/Reports/Final/Code

6. Agradecimientos

Agradecimientos al profesor Ramiro por su paciencia, y sus sabias explicaciones a la hora de impactar la fpga, realizar los testbench, a los compañeros que nos apoyaron con sus ideas y aportes, a la universidad por permitirnos las instalaciones para realizar el desarrollo de este proyecto.