Diseño y programación de un letrero dinámico basado en la utilización de VHDL

Avila Morales, Luis Enrique; Tuñon, Genesis; Madrid, Marisabel
Luis11avila@gmail.com, genesis_naza@hotmail.es, marisabelmadrid9@gmail.com

Departamento Ingienieria Biomédica
Universidad latina de panamá

Abstract- This document provides the reader with the general characteristics and operation of electronic posters with a friendly and direct interface with the user through the VHDL programming language.
The program allows managing the seven-segment displays of a programmable board. To fulfill this purpose, a message with the letters "DELL" was preloaded, the message is decoded from BCD to binary code, in order to be able to display it in the displays, displaying in scrolling, by multiplexers, to control the reset actions, there is a reset button which allows the total deletion of the screen, as well as the selection of the reset, and the scrolling of the text from left to right To achieve this, the working frequency of the displays was taken into account so that the displacement of the characters in them is correct.

***Resumen*- En el presente documento se brinda al lector las características generales y funcionamiento de los carteles electrónicos con una interfaz amigable y directa con el usuario mediante lenguaje de programación VHDL.**
El programa permite realizar el manejo de los visualizador siete segmentos de una placa programable, para cumplir con este propósito se precargo un mensaje con las letras "DELL", el mensaje de encuentra decodificado de código BCD a binario, con el fin de poder mostrarlo en los visualizadores, observados en desplazamiento, por multiplexores , para controlar las acciones de reinicio, se cuenta con un pulsador reset el cual permite realizar el borrado total de la pantalla, así como la selección del reinicio, y el desplazamiento del texto de izquierda a derecha, para lograrlo se tomó en cuenta la frecuencia de trabajo de los visualizadores para que el desplazamiento de los caracteres en ellos sea el correcto.
***Palabras Clave*- VHDL, decodificados, multiplexores, BCD, binario, frecuencia.**

I. INTRODUCCIÓN

En este proyecto conoceremos como diseñar un indicador de mensajes en 7-segmentos, el mismo que nos permitirá exhibir palabras seleccionadas. La creación de un indicador de mensajes posee una gran variedad de aplicaciones como por ejemplo este indicador de mensajes suele ser muy utilizado en bancos para indicar cierta información, en centros comerciales para pasar mensajes publicitarios, etc. Por supuesto, sus dimensiones pueden resultar insuficientes para algunos usos, pero es fácilmente uno de sus grandes ventajas es que son fácilmente expandibles. Como estudiantes de la carrera de Ingeniería Biomédica, nos hemos planteado desarrollar como proyecto final un cartel basado en una visualización a la vez de 4 visualizador 7-segmentos, dentro de varias placas electrónicas, adecuadamente programado, es importante mencionar que este proyecto puede resultar interesante como un producto comercializable ya que la mayoría de negocios requieren un indicador de mensajes para poder realizar publicidad de los mismos.

II. DISEÑO Y PROGRAMACIÓN

El problema planteado para este trabajo es básicamente el diseño y construcción de un indicador de mensajes, el cual deberá ser elaborado mediante programación a VHDL(Virtual Hardware Description Language) de un letrero que muestra un mensaje por los visualizadores de 7 segmentos de la FPGA (Field Programmable Gate Array). La estructura Inicial de el programa principal contiene una entidad "Mensaje" y se encarga de manejar las entradas y salidas de la FPGA. Mensaje contiene cuatro endidades "Letras" y almacena el mensaje, manejando la velocidad con la que aparece. Letra decodifica la letra recibida y la muestra por el display que le corresponda.

A. Decodificar el visualizador

Se tiene 4 visualizadores en la placa escogida en la cual debemos saber que se va mostrar en ella en este caso se desea que la palabra sea 'DELL'.

Un visualizador de siete segmentos se compone de siete diodos emisores de luz (LEDs) dispuestos en un patrón como se muestra en la figura 1, y algunos incluyen un octavo LED para el punto decimal. Existen dos tipos de visualizadores de acuerdo a su conexión eléctrica: ánodo común y cátodo común. Desde el punto de vista de programación (o del diseño de sistemas digitales), es necesario saber el tipo de visualizador para las condiciones de activación:

- En ánodo común, un 0 se utiliza para encendido y un 1 para apagado (lógica negativa).

Al tener 7 LEDs se pueden generar un total de 128 combinaciones, aunque no todas ellas conforman caracteres. En esta entrada se hace uso de la tabla de símbolos utilizada por [1]y se codifican los siguientes caracteres: 10 digitos (del 0 al 9), 26 letras (A a la Z), y los caracteres _, –, y .. Para representar estos 37 símbolos se necesitan 6 bits (cinco bits pueden representar hasta 32 valores solamente). A este proceso de asignar un número a cada carácter se le conoce como codificación. Finalmente, para mostrar el digito en un visualizador de siete segmentos es necesario decodificar el valor numérico según el patrón de LEDs indicado. Esta decodificación se realiza asumiendo que el visualizador es de ánodo común (cero activa, uno apaga).

En resumen, inicialmente 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 (decodificación).

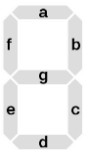


fig.1 Visualizador siete segmentos

La descripción de hardware se encarga de determinar cuales LEDs estarán encendidos en base al valor de entrada de seis bits para cada uno de los 37 símbolos distintos y un caso para cubrir cualquier otro valor posible.

Tabla I
DECODIFICACIÓN DE MESAJE AL SIETE SEGMENTOS

LETRAS	g	f	e	d	c	b	a
D	0	1	0	0	0	0	1
E	0	0	0	0	1	1	0
L	1	0	0	0	1	1	1
L	1	0	0	0	1	1	1

El orden de los bits escogidos se muestra en la tabla I.

Cada bit correspondiente a cada segmento, hay que saber que en una FPGA los visualizadores también se pueden elegir cuales leds encienden, pero esta señal no se le saca las ecuaciones pues serán visualizados en leds individuales y se debe implementar otra porque sino se visualizaría como en la figura 2.

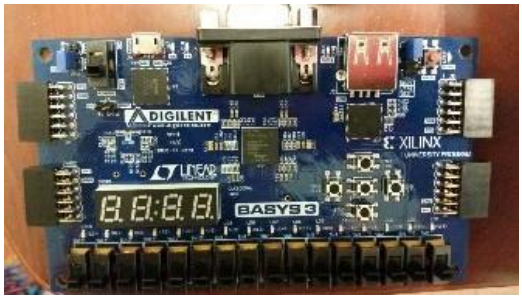


Fig.2 Basys 3 con la visualización

B. Multiplexores y frecuencias

Se desarrolló un componente para mostrar dígitos en un visualizador de siete segmentos, mismo que se mostraba en los cuatro visualizadores de la tarjeta Basys 3. En esta pequeña entrada se hace uso de un multiplexor para mostrar un dígito distinto en cada uno de los visualizadores.

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 3 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. Un claro ejemplo de un multiplexor se encuentra en la televisión, donde solamente se muestra en pantalla el canal de deportes o el canal de música, pero no ambos a la vez (al menos hasta hace unos años, claro está)

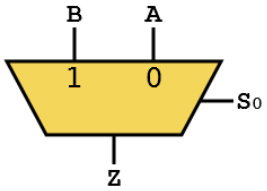


Fig.3 Diagrama de multiplexor

Desde el punto de vista de programación, equivale a una simple estructura if - else, donde una variable puede tomar o uno u otro valor. Habiendo dicho esto, es claro que un multiplexor puede verse como una instrucción switch al incrementar la cantidad de valores posibles.
¿Qué tiene que ver un multiplexor con los visualizadores? Primeramente, observemos el diagrama electrónico de los visualizadores en la tarjeta Basys (figura 4). En base a la figura, podemos deducir que solamente existen 8 señales para mostrar los dígitos (no 32 como se esperaba) y las señales AN0 a AN3 son las encargadas de decidir cuáles de los cuatro visualizadores están activo.

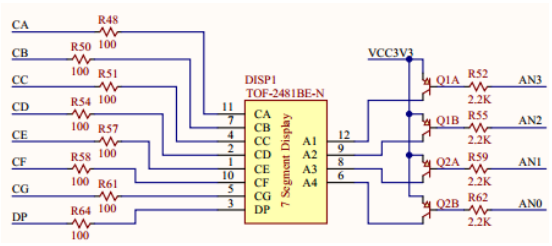


Fig.4 Diagrama electrónico de los visualizadores.

Ahora, ¿cómo mostrar un dígito diferente en los cuatro visualizadores a la vez? La verdad es que solamente se puede mostrar un dígito diferente a la vez, pero si utilizamos la frecuencia correcta, se llega a la ilusión de que hay cuatro dígitos distintos. Este efecto es similar a un video, el cual es compuesto de muchas imágenes sucesivas. Una frecuencia de 200Hz es más que suficiente para generar este efecto, por lo cual se utiliza el divisor creado en divisor de frecuencia con VHDL.
Dado que ya contamos con el divisor de frecuencia y nuestro componente para mostrar un dígito, solamente falta crear un componente para determinar cual de los cuatro dígitos deseamos mostrar, en otras palabras, crear multiplexores.
Utilizamos dos multiplexores con contador para aplicarlos a los estados implementados para su visualización.

C.- Máquina de Estado Finito

Una Máquina de Estado Finito (Finite State Machine), llamada también Autómata Finito es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de Estados y un número determinado de Transiciones entre dicho Estados.
Las Transiciones de un estado a otro se generan en respuesta a eventos de entrada externos e internos; a su vez estas transiciones y/o subsecuentes estados pueden generar otros

eventos de salida. Esta dependencia de las acciones (respuesta) del sistema a los eventos de entrada hacen que las Máquinas de Estado Finito (MEF) sean una herramienta adecuada para el diseño de Sistemas Reactivos y la Programación Conducida por Eventos (Event Driven Programming), cual es el caso de la mayoría de los sistemas embebidos basados en microcontroladores o microprocesadores.

Las MEF se describen gráficamente mediante los llamados Diagramas de Estado Finito (DEF), llamados también Diagramas de Transición de Estados. Las MEF no son diagramas de flujo y no deben confundirse con los mismos. En una MEF las acciones se asocian con las flechas (transiciones), mientras que un Diagrama de Flujo las acciones se asocian a los vértices de la flecha o a los bloques de proceso.

Cuando una MEF se encuentra en uno de sus estados, básicamente se encuentra "en reposo" esperando a que suceda un evento, mientras que en un Diagrama de Flujo el sistema se encuentra activo realizando una tarea.

Existen principalmente dos tipos de Máquinas de Estado Finito: Las Reconocedoras o Detectoras y las Transductoras.

b.1. Reconocedoras o Detectoras: Llamadas también Detectoras de Secuencia, realizan básicamente la detección de patrones o secuencias determinadas en respuesta a las entradas recibidas. Por su definición teórica este tipo de sistema no proveen señales de salida (acciones), simplemente transicionan desde un estado inicial a un estado final de "Exito", en cuyo caso se entiende que un patrón o secuencia ha sido reconocida exitosamente. Las MEF Detectoras de Secuencia son útiles en aplicaciones en las que se necesita verificar contraseñas, códigos o la validación de paquetes de datos en transmisión digital, este último un ejemplo muy típico de su uso.

b.2. Transductoras: Las MEFs transductoras se caracterizan por generar acciones o salidas dependiendo de las entradas y/o estados; se implementan en sistemas embebidos típicamente para aplicaciones de control.

D. **FPGA**

Si La tarjeta BASYS 3 es una tarjeta de desarrollo bastante completa que le ofrece al usuario la capacidad de realizar pruebas fácilmente con el hardware instalado así mismo como capacidades de expansión atreves de sus terminales PMOD y su conector VHDC mediante los cuales se abre una cantidad de dispositivos a los cuales podemos tener acceso; dispositivos como: ADC,DAC, WIFI, LCD entre otros.Oscilador y Reloj

La Basys 3 incluye un oscilador de 100MHz conectado al pin V10 el cual es parte del Bank 2 del FPGA y la entrada de reloj GCLK0. La entrada de reloj puede administrar hasta 4 controladores de reloj del FPGA El reloj de entrada puede conducir cualquiera o todas las cuatro fichas de gestión del reloj en el Spartan-6. Cada ficha incluye dos Gerentes Digital Clock (MCD) y cuatro bucles de enganche de fase (PLL).

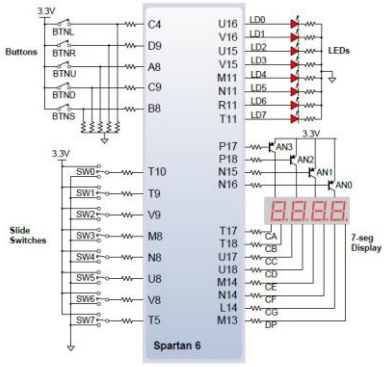
La entrada de reloj puede controlar hasta 4 administradores de reloj del FPGA, cada uno consta de administradores de reloj digital (DCM) (Digital Clock Manager) y de cuatro seguidores de fase retroalimentados PLL (Phase-Locked Loops).

ENTRADAS Y SALIDAS BASICAS

En la tarjeta BASYS+ 3 se tiene disponible una seria de hardware que ofrece al diseñador la posibilidad de realizar pruebas fácilmente; entre los cuales tenemos:

- 8 Leds
- 4 Disp de 7 segmentos
- 8 Interruptores
- 5 Botones

En la imagen se muestra el circuito y las conexiones de los distintos dispositivos.



Recordemos que las pantallas de 7 segmentos comparten los mismos pines de cátodos para las cuatros pantallas y los ánodos están conectados al FPGA por medio de transistores PNP.

E. **Programación**

Se debe establecer el uso de los componentes, primero se establecen las variables y señales internas para los estados

Los circuitos digitales, a no ser que sean asíncronos, van comandados por un reloj cuya frecuencia puede variar según el tipo de sistema digital del que se trate. Desde microprocesador 6502 que funcionaba con un reloj de 1Mhz hasta los actuales, que funcionan en el orden de los gigahercios, no han pasado ni cuatro décadas. En un sistema digital complejo es habitual que necesitemos obtener diferentes frecuencias de reloj para diferentes subsistemas. Un ejemplo muy claro puede ser el de un reloj digital que tiene que contar los segundos, por lo tanto, necesita un reloj de 1Hz (un pulso por segundo) [2]. En esta vamos a ver dos de relojes uno que representa el proceso del multiplexor y otro del reloj de entrada al sistema.

Project Summary | Schematic | DELL_LET.vhd * | test_DELL.vhd

D:/Xilinx/LETRERO_DELL/LETRERO_DELL.srcs/sources_1/new/DELL_LET.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity DELL_LET is
    port ( clkin: in std_logic;
    clear: in std_logic;
    clkout: out std_logic;
    seg1: out std_logic_vector(6 downto 0);
    seg2: out std_logic_vector(6 downto 0);
    seg3: out std_logic_vector(6 downto 0);
    seg4: out std_logic_vector(6 downto 0));
end DELL_LET;
architecture Behavioral of DELL_LET is
    signal cont: integer range 0 to 1;
    signal clk : std_logic := '1';
    type estado is (A, B, C, D, E, F, G, H);
    signal actual, futuro : estado;
begin   --Reloj
    reloj: process (clkin, clk, cont) begin
        if(clkin'event and clkin = '1') then
            if(cont < 1) then
                cont <= cont + 1;
            else
                cont <= 0;
                clk <= not clk;
            end if;
        end if;
    end process reloj;
    reloj2: process(clk) begin
        clkout <= clk;
    end process reloj2;
    --Clear e Inicio
    inicio: process(clk, clear) begin
        if(clear = '1') then
            actual <= A;
        elsif(clk'event and clk = '1') then
            actual <= futuro;
        end if;
    end process inicio;
```

```
    desplegar: process(actual) begin
        case actual is
            when A =>
                seg1 <= "1111111";
                seg2 <= "1111111";
                seg3 <= "1111111";
                seg4 <= "1111111";
                futuro<= B;
            when B =>
                seg1 <= "0100001"; --D
                seg2 <= "1111111";
                seg3 <= "1111111";
                seg4 <= "1111111";
                futuro<= C;
            when C =>
                seg1 <= "0000110"; --E
                seg2 <= "0100001"; --D
                seg3 <= "1111111";
                seg4 <= "1111111";
                futuro<= D;
            when D =>
                seg1 <= "1000111"; --L
                seg2 <= "0000110"; --E
                seg3 <= "0100001"; --D
                seg4 <= "1111111";
                futuro<= E;
            when E =>
                seg1 <= "1000111"; --L
                seg2 <= "1000111"; --L
                seg3 <= "0000110"; --E
                seg4 <= "0100001"; --D
                futuro<= F;
            when F =>
                seg1 <= "1111111";
                seg2 <= "1000111"; --L
                seg3 <= "1000111"; --L
                seg4 <= "0000110"; --E
                futuro<= G;
            when G =>
                seg1 <= "1111111";
                seg2 <= "1111111";
                seg3 <= "1000111"; --L
                seg4 <= "1000111"; --L
                futuro<= H;
```

III. RESULTADOS

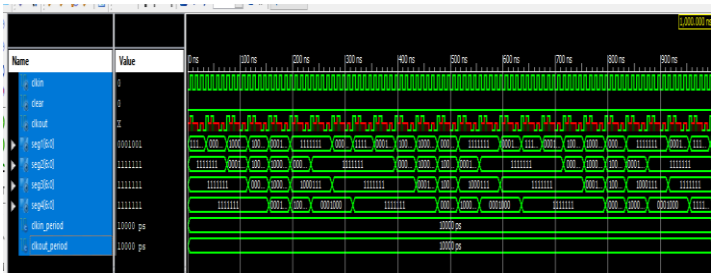
Al desarrollar el presente proyecto, se colocaron clocks los cuales van generados con el multiplexor; un contador y flip flops ellos permiten el almacenamiento y transferencia de los datos digitales y los utilizamos para el almacenamiento de datos; ya que nos dan las señales.

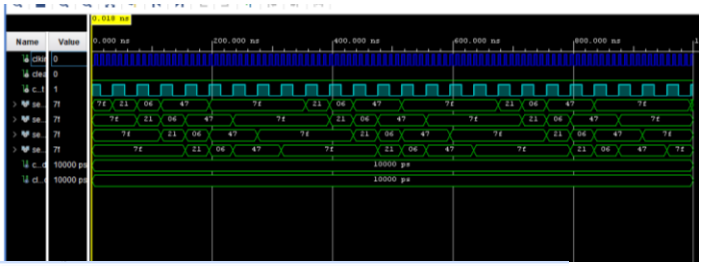
Ambos clocks tienen flip flops un contador y un multiplexor, el primero que va hacia el reloj interno de la tarjeta de FPGA; y el otro que va a enviar los impulsos hacia los display's de siete segmentos.

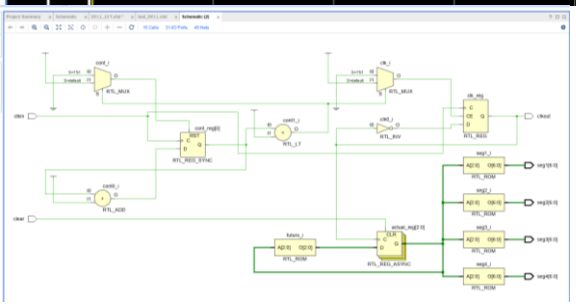
Se tiene un otro componente que es la máquina de estado está justo en frente de los 4 display's, este genera la salida dependiendo de cuál sea la señal recibida. También se tiene un registro colocado delante de los 4 display's de siete segmentos, ese registro es el que nos indicará cual segmento se va a encender, este va controlado por la programación que tiene la máquina de estado. Esta emitirá señales de futuro y señales de estados actuales, así son denominadas las señales emitidas, siendo controladas por la máquina de estado.

Como resultado de esta disposición en la gráfica de simulación obtenemos que existen dos pulsos de señal. Para cada sistema, solamente el displa'sy pueden estar activado a la vez cuando son iguales; sí son diferentes el display's no se puede activar, referente a los pulsos de señales.

En el Vivado él está en un lenguaje para que se vea más claro representando en dos canales diferentes; los cuales representarán la numeración. Esta es una representación de la memoria, enviando la señal de como seria; y ella ya tiene lo almacenado.







```
library ieee;
use ieee.std_logic_1164.all;
entity tb_DELL_LET is
end tb_DELL_LET;
architecture tb of tb_DELL_LET is
    component DELL_LET
        port (clkin  : in std_logic;
              clear  : in std_logic;
              clkout : out std_logic;
              seg1   : out std_logic_vector (6 downto 0);
              seg2   : out std_logic_vector (6 downto 0);
              seg3   : out std_logic_vector (6 downto 0);
              seg4   : out std_logic_vector (6 downto 0));
    end component;
     --Inputs
   signal clkin : std_logic := '0';
   signal clear : std_logic := '0';
    --Outputs
   signal clkout : std_logic;
   signal seg1 : std_logic_vector(6 downto 0);
   signal seg2 : std_logic_vector(6 downto 0);
   signal seg3 : std_logic_vector(6 downto 0);
   signal seg4 : std_logic_vector(6 downto 0);
   -- Clock period definitions
   constant clkin_period : time := 10 ns;
   constant clkout_period : time := 10 ns;
begin
    -- Instantiate the Unit Under Test (UUT)
   uut: DELL_LET PORT MAP (
          clkin => clkin,
          clear => clear,
          clkout => clkout,
          seg1 => seg1,
          seg2 => seg2,
          seg3 => seg3,
          seg4 => seg4
        );
   -- Clock process definitions
   clkin_process :process
   begin
        clkin <= '0';
        wait for clkin_period/2;
        clkin <= '1';
        wait for clkin_period/2;
```

```
   signal seg4 : std_logic_vector(6 downto 0);
   -- Clock period definitions
   constant clkin_period : time := 10 ns;
   constant clkout_period : time := 10 ns;
begin
    -- Instantiate the Unit Under Test (UUT)
   uut: DELL_LET PORT MAP (
          clkin => clkin,
          clear => clear,
          clkout => clkout,
          seg1 => seg1,
          seg2 => seg2,
          seg3 => seg3,
          seg4 => seg4
        );
   -- Clock process definitions
   clkin_process :process
   begin
        clkin <= '0';
        wait for clkin_period/2;
        clkin <= '1';
        wait for clkin_period/2;
   end process;
   clkout_process :process
   begin
        clkin <= '0';
        wait for clkin_period/2;
        clkin <= '1';
        wait for clkin_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      wait for clkin_period*10;
      -- insert stimulus here
        clear <= '0';
      wait;
   end process;
END;
```

IV. Conclusiones

- La codificación deL símbolos, es decir, asignación de caracteres a determinado número binario.
- La decodificación de ese valor numérico hacia un visualizador de siete segmentos.
- La descripción de hardware que crea un componente digital dedicado al visualizador.
- La implementación del sistema en la tarjeta BASYS3 me hace pensar en la ubicación de los pines y sus nombres para mondar el programa.

II. **Referencias**

[1] Palacios, Enrique; Remiro, Fernando y López, Lucas. , Microcontrolador PIC16F84, México: Alfaomega.: Desarrollo de proyectos, 2004.

[2] Haskell, Richard E. y Hanna, Darrin M. , Learning by Example Using VHDL – Basic Digital Design with a Basys FPGA Board, Michigan: LBE, 2008.

[3] R. Tocci, Sistemas Digitales – Principios y Aplicaciones (5ta Edición), México: Prentice Hall., 1993.