**Objetivo**

El alumno aprenderá el diseño de contadores mediante un FPGA y con visualización en un monitor VGA.

**Introducción**

La VGA, que significa "Video Graphics Array", es un estándar de señal de video que ha desempeñado un papel fundamental en la visualización de imágenes en computadoras y monitores durante décadas. Aunque ha sido ampliamente superado por tecnologías más modernas como el HDMI y el DisplayPort, la VGA sigue siendo relevante en algunos contextos, especialmente en equipos y sistemas más antiguos.

Una de las características distintivas de la señal de video VGA es que consta de cinco señales activas. Estas señales activas son esenciales para transmitir información de video desde una computadora o dispositivo a un monitor o pantalla. Cada una de estas señales activas desencadena la generación de una imagen en el monitor, y juntas, permiten la representación de una imagen completa en la pantalla.

Las cinco señales activas en una señal de video VGA incluyen la señal de sincronización horizontal (HSYNC) y la señal de sincronización vertical (VSYNC), que se encargan de coordinar la posición y el refresco de la imagen en la pantalla. Además, se encuentran las señales de color: rojo (R), verde (G) y azul (B), que determinan los colores y la intensidad de la imagen que se mostrará. Al combinar estas cinco señales activas de manera adecuada, se logra la representación de imágenes nítidas y a todo color en un monitor VGA.

**Desarrollo**

Diagrama

Descripción generada automáticamente

Figura 1. Asignación de cada segmento con sus respectivas coordenadas

Letra, su color y código correspondiente con el que se construirán los números:

**A azul**

If ((row > 200 and row <210) and

(column>110 and column<140)) then – A azul

Red <= (others => ‘0’);

Green <= (others => ‘0’);

Blue <= (others => ‘1’);

**B verde**

Elsif ((row > 210 and row <240) and

(column>140 and column<150)) then – B verde

Red <= (others => ‘0’);

Green <= (others => ‘1’);

Blue <= (others => ‘0’);

**C rojo**

Elsif ((row > 250 and row <280) and

(column>140 and column<150)) then – C rojo

Red <= (others => ‘1’);

Green <= (others => ‘0’);

Blue <= (others => ‘0’);

**D blanco**

Elsif ((row > 280 and row <290) and

(column>110 and column<140)) then – D blanco

Red <= (others => ‘1’);

Green <= (others => ‘1’);

Blue <= (others => ‘1’);

**E cian**

Elsif ((row > 250 and row <280) and

(column>100 and column<110)) then – E cian

Red <= (others => ‘0’);

Green <= (others => ‘1’);

Blue <= (others => ‘1’);

**F amarillo**

Elsif ((row > 210 and row <240) and

(column>100 and column<110)) then – F amarillo

Red <= (others => ‘1’);

Green <= (others => ‘1’);

Blue <= (others => ‘0’);

**G violeta**

Elsif ((row > 240 and row <250) and

(column>110 and column<140)) then – G violeta

Red <= (others => ‘1’);

Green <= (others => ‘0’);

Blue <= (others => ‘1’);

ACTIVIDAD COMPLEMENTARIA:

Diseñar un contador binario descendente con visualización en un monitor VGA. Cuando el

contador llegue a su límite de cuenta, éste deberá reiniciarse

**Código del programa**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity p12 is

generic( --Constantes para monitor VGA en 640x480

constant h\_pulse : integer := 96;

constant h\_bp : integer := 48;

constant h\_pixels : integer := 640;

constant h\_fp : integer := 16;

constant v\_pulse : integer := 2;

constant v\_bp : integer := 33;

constant v\_pixels : integer := 480;

constant v\_fp : integer := 10

);

port( clk50MHz: in std\_logic;

red: out std\_logic\_vector (3 downto 0); -- al monitor

green: out std\_logic\_vector (3 downto 0);

blue: out std\_logic\_vector (3 downto 0);

h\_sync: out std\_logic;

v\_sync: out std\_logic;

dipsw: in std\_logic\_vector(3 downto 0); -- numeros para

A,B,C,D,E,F,G: out std\_logic;

reset:std\_logic ); -- decodificador

end entity p12;

architecture behavioral of p12 is

constant h\_period : integer := h\_pulse+h\_bp+h\_pixels+h\_fp;

subtype state is std\_logic\_vector (3 downto 0);

signal present\_state,count, next\_state: state;

constant state0: state:= "0000";

constant state1: state:= "0001";

constant state2: state:= "0010";

constant state3: state:= "0011";

constant state4: state:= "0100";

constant state5: state:= "0101";

constant state6: state:= "0110";

constant state7: state:= "0111";

constant state8: state:= "1000";

constant state9: state:= "1001";

constant state10: state:= "1010";

constant state11: state:= "1011";

constant state12: state:= "1100";

constant state13: state:= "1101";

constant state14: state:= "1110";

constant state15: state:= "1111";

constant v\_period : integer := v\_pulse+v\_bp+v\_pixels+v\_fp;

constant cero: std\_logic\_vector(6 downto 0):="0111111"; --GFEDCBA

constant uno: std\_logic\_vector(6 downto 0):="0000110";

constant dos: std\_logic\_vector(6 downto 0):="1011011";

constant tres: std\_logic\_vector(6 downto 0):="1001111";

constant cuatro: std\_logic\_vector(6 downto 0):="1100110";

constant cinco: std\_logic\_vector(6 downto 0):="1101101";

constant seis: std\_logic\_vector(6 downto 0):="1111101";

constant siete: std\_logic\_vector(6 downto 0):="0000111";

constant ocho: std\_logic\_vector(6 downto 0):="1111111";

constant nueve: std\_logic\_vector(6 downto 0):="1110011";

constant diez: std\_logic\_vector(6 downto 0):="1110111";

constant once: std\_logic\_vector(6 downto 0):="1111100";

constant doce: std\_logic\_vector(6 downto 0):="0111001";

constant trece: std\_logic\_vector(6 downto 0):="1011110";

constant catorce: std\_logic\_vector(6 downto 0):="1111001";

constant quince: std\_logic\_vector(6 downto 0):="1110001";

constant r1:std\_logic\_vector(3 downto 0):=(others => '1');

constant r0:std\_logic\_vector(3 downto 0):=(others => '0');

constant g1:std\_logic\_vector(3 downto 0):=(others => '1');

constant g0:std\_logic\_vector(3 downto 0):=(others => '0');

constant b1:std\_logic\_vector(3 downto 0):=(others => '1');

constant b0:std\_logic\_vector(3 downto 0):=(others => '0');

signal h\_count : integer range 0 to h\_period-1 := 0;

signal v\_count : integer range 0 to v\_period-1 := 0;

signal conectornum:std\_logic\_vector(6 downto 0); -- coneccion del

signal reloj\_pixel : std\_logic;

signal column : integer := 0;

signal row : integer :=0;

signal display\_ena : std\_logic ;

signal conteo:integer := 0;

signal clklow:std\_logic;

begin

process(clk50mhz)

begin

if (clk50mhz'event and (clk50mhz ='1')) then

conteo<=conteo+1;

if (conteo=39000000) then

conteo<=0;

clklow<=not(clklow);

end if;

end if;

end process;

relojpixel: process (clk50MHz) is

begin

if rising\_edge(clk50MHz) then

reloj\_pixel <= not reloj\_pixel;

end if;

end process relojpixel; -- 25mh

contadores : process (reloj\_pixel) -- H\_periodo=800, V\_periodo=525

begin

if rising\_edge(reloj\_pixel) then

if h\_count<(h\_period-1) then

h\_count<=h\_count+1;

else

h\_count<=0;

if v\_count<(v\_period-1) then

v\_count<=v\_count+1;

else

v\_count<=0;

end if;

end if;

end if;

end process contadores;

conta1: process(clklow)

begin

if rising\_edge(clklow) then

if (reset='1') then

present\_state <= state15;

else

present\_state<= next\_state;

end if;

end if;

end process;

conta2: process(present\_state)

begin

case present\_state is

when state0=>

next\_state<= state15;

when state15=>

next\_state<= state14;

when state14=>

next\_state<= state13;

when state13=>

next\_state<= state12;

when state12=>

next\_state<= state11;

when state11=>

next\_state<= state10;

when state10=>

next\_state<= state9;

when state9=>

next\_state<= state8;

when state8=>

next\_state<= state7;

when state7=>

next\_state<= state6;

when state6=>

next\_state<= state5;

when state5=>

next\_state<= state4;

when state4=>

next\_state<= state3;

when state3=>

next\_state<= state2;

when state2=>

next\_state<= state1;

when state1=>

next\_state<= state0;

when others=>

next\_state<= state0;

end case;

count <= present\_state;

end process;

senial\_hsync : process (reloj\_pixel) --h\_pixel+h\_fp+h\_pulse= 784

begin

if rising\_edge(reloj\_pixel) then

if h\_count>(h\_pixels + h\_fp) or

h\_count>(h\_pixels + h\_fp + h\_pulse) then

h\_sync<='0';

else

h\_sync<='1';

end if;

end if;

end process senial\_hsync;

senial\_vsync : process (reloj\_pixel) --vpixels+v\_fp+v\_pulse=525

begin --checar si se en parte visible es 1 o 0

if rising\_edge(reloj\_pixel) then

if v\_count>(v\_pixels + v\_fp) or

v\_count>(v\_pixels + v\_fp + v\_pulse) then

v\_sync<='0';

else

v\_sync<='1';

end if;

end if;

end process senial\_vsync;

coords\_pixel: process(reloj\_pixel)

begin --asignar una coordenada en parte visible

if rising\_edge(reloj\_pixel) then

if (h\_count < h\_pixels) then

column <= h\_count;

end if;

if (v\_count < v\_pixels) then

row <= v\_count;

end if;

end if;

end process coords\_pixel;

generador\_imagen: process(display\_ena, row, column)

begin

if(display\_ena = '1') then

case(conectornum) is

when cero=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when uno=>

if ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when dos=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when tres=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when cuatro=>

if ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when cinco=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when seis=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when siete=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when ocho=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when nueve=>

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When diez=> --A

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When once=> --B

if ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When doce=> --C

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When trece=> --D

if ((row > 210 and row <240) and

(column>140 and column<150)) then -- B verde

red <= (others => '0');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>140 and column<150)) then -- C rojo

red <= (others => '1');

green <= (others => '0');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When catorce=> --E

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 280 and row <290) and

(column>110 and column<140)) then -- D blanco

red <= (others => '1');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

When quince=> --F

if ((row > 200 and row <210) and

(column>110 and column<140)) then -- A azul

red <= (others => '0');

green <= (others => '0');

blue <= (others => '1');

elsif ((row > 250 and row <280) and

(column>100 and column<110)) then -- E cian

red <= (others => '0');

green <= (others => '1');

blue <= (others => '1');

elsif ((row > 210 and row <240) and

(column>100 and column<110)) then -- F amarillo

red <= (others => '1');

green <= (others => '1');

blue <= (others => '0');

elsif ((row > 240 and row <250) and

(column>110 and column<140)) then -- G violeta

red <= (others => '1');

green <= (others => '0');

blue <= (others => '1');

else -- fondo

red <= (others => '0');

green <= (others => '0');

blue <= (others => '0');

end if;

when others=>

red <= (OTHERS => '0');

green <= (OTHERS => '0');

blue <= (OTHERS => '0');

end case;

else

red<= (others => '0');

green <= (others => '0');

blue<= (others => '0');

end if;

end process generador\_imagen;

with present\_state select conectornum <= --decodificador para los números

"0111111" when "0000",

"0000110" when "0001",

"1011011" when "0010",

"1001111" when "0011",

"1100110" when "0100",

"1101101" when "0101",

"1111101" when "0110",

"0000111" when "0111",

"1111111" when "1000",

"1110011" when "1001",

"1110111" when "1010",

"1111100" when "1011",

"0111001" when "1100",

"1011110" when "1101",

"1111001" when "1110",

"1110001" when "1111",

"0000000" when others;

display\_enable: process(reloj\_pixel) --- h\_pixels=640; y\_pixeles=480

begin

if rising\_edge(reloj\_pixel) then

if (h\_count < h\_pixels AND v\_count < v\_pixels) THEN

display\_ena <= '1';

else

display\_ena <= '0';

end if;

end if;

end process display\_enable;

end Behavioral;

En este código se agregaron las siguientes partes

*conta1: process(clklow) begin if rising\_edge(clklow) then if (reset='1') then present\_state <= state0; else present\_state<= next\_state; end if; end if; end process; conta2: process(present\_state) begin case present\_state is when state0=> next\_state<= state1; when state1=> next\_state<= state2; when state2=> next\_state<= state3; when state3=> next\_state<= state4; when state4=> next\_state<= state5; when state5=> next\_state<= state6; when state6=> next\_state<= state7; when state7=> next\_state<= state8; when state8=> next\_state<= state9; when state9=> next\_state<= state10; when state10=> next\_state<= state11; when state11=> next\_state<= state12; when state12=> next\_state<= state13; when state13=> next\_state<= state14; when state14=> next\_state<= state15; when state15=> next\_state<= state0; when others=> next\_state<= state0; end case; count <= present\_state; end process;*

Esta parte es el sistema que cambia el estado.

*subtype state is std\_logic\_vector (3 downto 0); signal present\_state, next\_state: state; constant state0: state:= "0000"; constant state1: state:= "0001"; constant state2: state:= "0010"; constant state3: state:= "0011"; constant state4: state:= "0100"; constant state5: state:= "0101"; constant state6: state:= "0110"; constant state7: state:= "0111"; constant state8: state:= "1000"; constant state9: state:= "1001"; constant state10: state:= "1010"; constant state11: state:= "1011"; constant state12: state:= "1100"; constant state13: state:= "1101"; constant state14: state:= "1110"; constant state15: state:= "1111"; begin*

Esta parte son señales y constantes que se crean para guardar la información de cada estado.

*process(clk50mhz) begin if (clk50mhz'event and (clk50mhz ='1')) then conteo<=conteo+1; if (conteo=1) then conteo<=0; clklow<=not(clklow); end if; end if; end process;*

En esta parte se pone un divisor de frecuencia el cual nos ayudara a realizar los cambios de señales cuando llegue a un valor de conteo, adicionalmente se modificará el valor de conteo para que tome mas tiempo el realizar el cambio.

En este código se modificó lo siguiente

Para adaptar el código de la practica anterior además de agregar el código del punto anterior se realizaron las siguientes modificaciones:

Se crearon los casos para cuando llegue a los valores de contador 10, 11, 12, 13, 14, 15 los cuales serán mostrados en hexadecimal.

Se adapto el valor del *with select* para que este verifique el valor del estado presente para realizar los cambios en la señal *conectornum* para todos los valores hexadecimales y así realizar los cambios, además se crearon las constantes las cuales nos daban los valores de los números 10, 11, 12, 13, 14 y 15.

La asignación de pines fue la siguiente:

Aplicación, Tabla

Descripción generada automáticamente

**Video del funcionamiento del programa:**

<https://youtu.be/gutZv51Y7qY?si=wE9U0t-ekOUVM75s>

Technologies, T. (2020, 5 junio). *DE-10 Lite User Manual*. Copyright © 2003-2017. <https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=China&No=1021&FID=a13a2782811152b477e60203d34b1baa>

Universidad Complutense de Madrid. (2014). *Introducción a la programación en VHDL.* <https://eprints.ucm.es/id/eprint/26200/1/intro_VHDL.pdf>

Instituto Tecnológico de Querétaro. (2016). *Guía básica del VHDL.* <http://www.itq.edu.mx/carreras/IngElectronica/archivos_contenido/Apuntes%20de%20materias/Apuntes_VHDL_2016.pdf>