

práctica 13.
CAPTURA DE IMÁGENES DE CÁMARA DIGITAL

OBJETIVOS:

El alumno aplicará los conocimientos y las habilidades obtenidas en el manejo de la señalización VGA, para definir una unidad de control de una cámara digital. Aprenderá además la señalización requerida en el almacenamiento de imágenes digitales en un FPGA.

INTRODUCCIÓN.

La cámara digital OV7670 captura imágenes de 640x480 pixeles. Opera a 3.3 V, aunque cuenta con un regulador que permite polarización de hasta 5V. El formato de salida de video, por defecto es el YUV (4:2:2), aunque puede generar RGB 4:2:2 y RGB565/555/444. El protocolo de comunicación con la cámara es el SCCB, compatible con el protocolo de comunicación I2C (*Inter Integrated Circuits*). La cámara incluye un módulo para el control del color, de la saturación, del tinte, de gama, y de realzado de bordes, entre otros. Éstos deben ser configurados escribiendo los valores adecuados en los registros correspondientes.

La cámara opera por default en formato YUV 4:2:2 de 640x480. De la señal entregada por la cámara, solamente se recupera la componente de luminancia y esta componente alimenta a un monitor con entrada VGA, y dependiendo de la capacidad del FPGA es la cantidad de bits que define la componente de luminancia.

La imagen que entrega la cámara es almacenada en una memoria de doble puerto (escritura y lectura) dentro del FPGA. La figura 13.1 muestra la cámara OV7670 y el kit de desarrollo.



Figura 13.1. Fotografía de la Cámara OV7670 [8]

ESPECIFICACIONES:

Utilizando una cámara digital, un FPGA y un monitor con entrada VGA, almacenar las imágenes dentro del FPGA con el fin de mostrarlas en un monitor. La figura 13.2 muestra el diagrama de bloques del sistema de Captura de Imágenes de Cámara Digital.

DIAGRAMA DE BLOQUES:

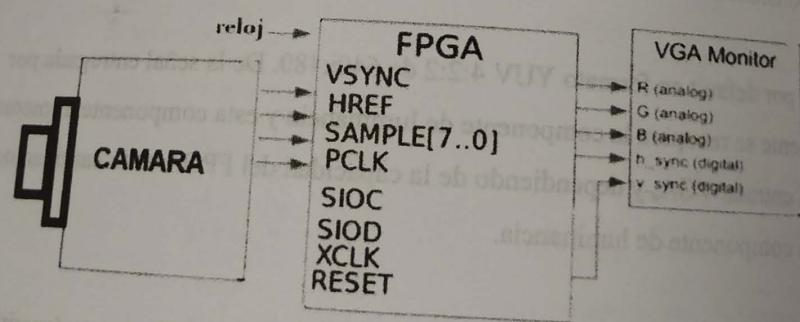
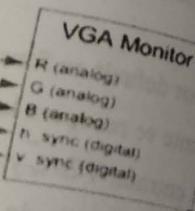


Figura 13.2. Diagrama de bloques de sistema captura de imágenes de cámara digital

El diagrama a bloques funcionales del sistema captura de imágenes de cámara digital es mostrado en la figura 13.3.

Cámara OV7670 [8]

on entrada VGA, almacenar las imágenes en memoria. La figura 13.2 muestra el sistema de Cámara Digital.



sistemas de cámara digital

de cámara digital es

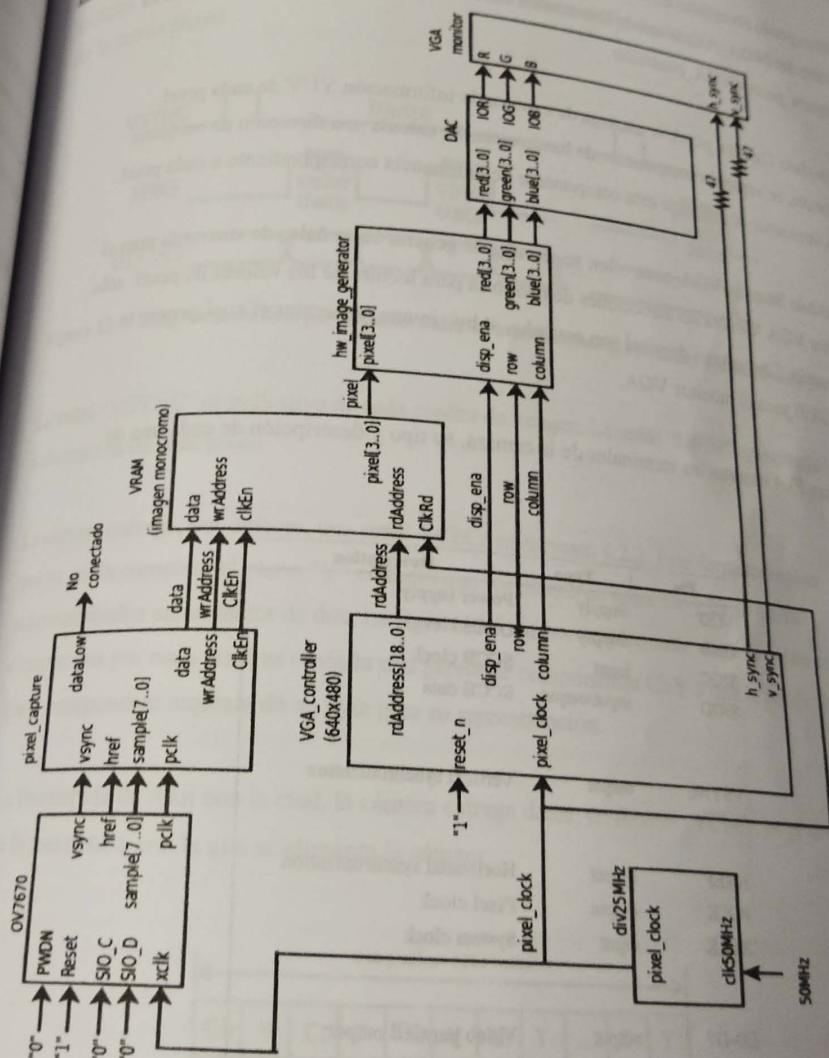


Figura 13.3. Diagrama a bloques del sistema captura de imágenes de cámara digital

Como puede observarse en el diagrama, se requiere diseñar cinco bloques funcionales dentro del FPGA. A cada uno lo llamaremos módulo, siendo los más importantes los de Captura_pixel y VGA_controller.

El módulo Captura_pixel, se encarga de capturar la información YUV de cada pixel. Entonces, se separa la componente de luminancia. Se calcula una dirección de memoria para almacenar únicamente esta componente de luminancia correspondiente a cada pixel.

El módulo llamado VGA_controller, se encarga de generar las señales de sincronía para el monitor VGA. Genera las direcciones de memoria para lectura de los valores de pixel: sólo luminancia. Los valores de pixel son enviados al hw_image_generator el cual genera la señal RGB para el monitor VGA.

La figura 13.4 muestra las terminales de la cámara, su tipo y descripción de cada una de ellas.

| Pin | Type | Description |
|-------|--------------|----------------------------|
| VDD | Supply | Power supply |
| GND | Supply | Ground level |
| SIOC | input | SCCB clock |
| SIOD | input/output | SCCB data |
| VSYNC | output | Vertical synchronization |
| HREF | output | Horizontal synchronization |
| PCLK | output | Pixel clock |
| XCLK | input | System clock |
| D0-D7 | output | Video paralell output |
| RESET | input | Reset (active low) |
| PWDN | input | Power down (active high) |

Figura 13.4. Definición de terminales de la cámara OV7670 [8]

LSI
Computación
es funcionales
portantes los de
cada pixel.
n de memoria
te a cada pixel.
ncronía para el
s de pixel; sólo
l genera la

da una de

La figura 13.5 muestra el diagrama de tiempos de las señales de sincronización recibidas por la cámara digital.

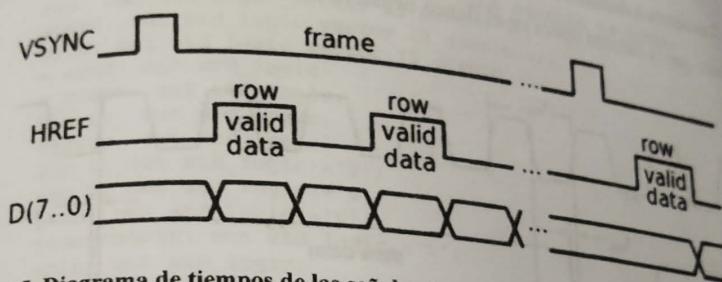


Figura 13.5. Diagrama de tiempos de las señales de sincronización recibidas por la cámara [8]

La señal "VSYNC" es indicativa de cada cuadro de imagen. La señal "HREF" encmarca la información de cada pixel.

La cámara entrega, por defecto, una señal YCbCr en formato 4.2.2. Este formato implica que se envía completo el plano "Y" en tanto que los planos de color Cb y Cr se envían submuestreados en un factor de dos. La figura 13.6 ilustra este formato. En esta figura se observa que por cada pixel es enviada una pareja de componentes CbY ó una la pareja CrY. Cada componente requiere de un byte para su representación.

La frecuencia de reloj con la cual, la cámara entrega datos, en formato YCbCr, es el doble de la frecuencia con la que se alimenta la cámara.

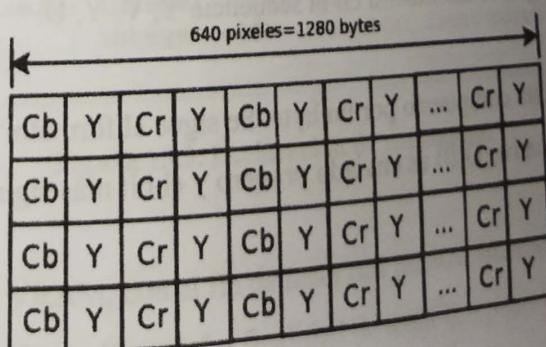


Figura 13.6. Formato de señales entregada por la cámara

La figura 13.7 muestra a detalle el diagrama de tiempos con la cual la cámara envía bytes de datos. Nótese que la cámara opera en el flanco negativo de la señal de reloj.

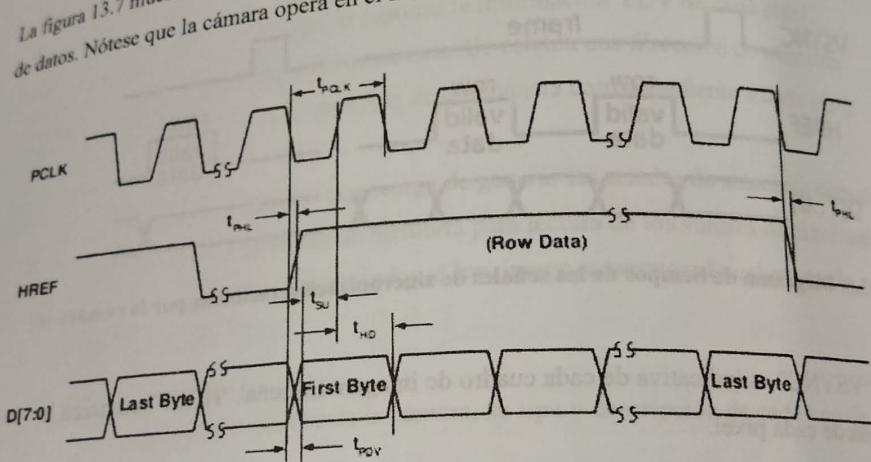


Figura 13.7. Diagrama de tiempos de las señales de salida de la cámara OV7670 [8]

Adicionalmente, se debe mencionar que la cámara debe alimentarse con una señal de reloj de 25MHz, debido a que provee 30 cuadros por segundo.

Dado que no se va enviar datos a la cámara, las señales de comunicación SIOD y SIOC pueden dejarse abiertas o bien, en "1".

La cámara viene pre configurada para proveer una salida en formato YUV 4:2:2, en particular, cada línea de imagen se suministra en la secuencia Y, V, Y, U.

Cada componente de color está compuesto por un byte. Se sigue el formato "little endian", es decir, el bit menos significativo D(0) es enviado primero y el bit más significativo D(7) es enviado al final.

Figura 13.6. Formato de señales entregada por la cámara

La figura 13.7 muestra a detalle el diagrama de tiempos con la cual la cámara envía bytes de datos. Nótese que la cámara opera en el flanco negativo de la señal de reloj.

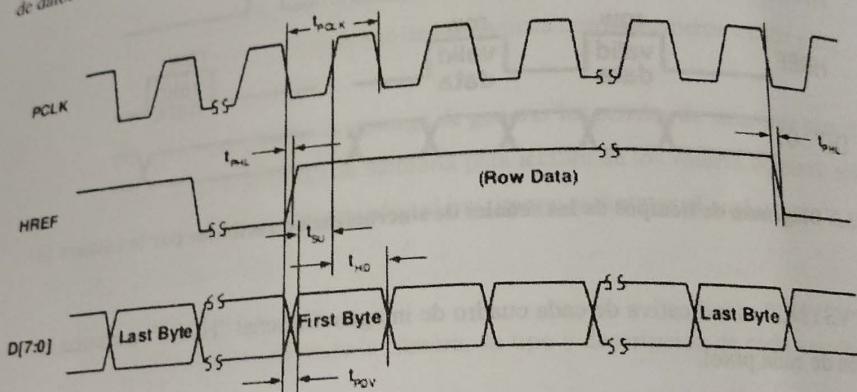


Figura 13.7. Diagrama de tiempos de las señales de salida de la cámara OV7670 [8]

Adicionalmente, se debe mencionar que la cámara debe alimentarse con una señal de reloj de 25MHz, debido a que provee 30 cuadros por segundo.

Dado que no se va enviar datos a la cámara, las señales de comunicación SIOD y SIOC pueden dejarse abiertas o bien, en "1".

La cámara viene pre configurada para proveer una salida en formato YUV 4:2:2, en particular, cada línea de imagen se suministra en la secuencia Y, V, Y, U.

Cada componente de color está compuesto por un byte. Se sigue el formato "little endian", es decir, el bit menos significativo D(0) es enviado primero y el bit más significativo D(7) es enviado al final.

| Y | V | Y | V | Y | V | Y | V |
|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |

La figura 13.8 muestra la entidad del sistema Captura de imágenes de cámara digital en un FPGA.

```
port( clk50MHz: in std_logic; --for this example is 50MHz
      red: out std_logic_vector (3 downto 0);
      green: out std_logic_vector (3 downto 0);
      blue: out std_logic_vector (3 downto 0);
      n_sync: out std_logic;
      n_blank: out std_logic;
      h_sync: out std_logic;
      v_sync: out std_logic;
      sio_c: out std_logic:='0';
      sio_d: out std_logic:='0';
      pwn: out std_logic:='0';
      resetcamera: out std_logic:='1';
      xclk: out std_logic;
      pclk: in std_logic;
      vsync: in std_logic;
      href: in std_logic;
      sample:in std_logic_vector (7 downto 0) );
```

Figura 13.8. Entidad del sistema captura de imágenes de cámara digital

Las constantes usadas en el programa corresponden a las constantes que se requieren para manipular el monitor con entrada de puerto VGA. La figura 13.9 muestra la declaración de dichas constantes.

```
generic( --constantes para monitor vga en 640x480
  constant h_pulse : integer:=96; --horizontal sync pulse width in pixels
  constant h_bp : integer:=48; --horizontal back porch width in pixels
  constant h_pixels: integer:=640;--horizontal display width in pixels
  constant h_fp : integer:=16;--horizontal front porch width in pixels
  constant v_pulse : integer:=2; --vertical sync pulse width in rows
  constant v_bp : integer:=33; --vertical back porch width in rows
  constant v_pixels: integer:=480;--vertical display width in rows
  constant v_fp : integer:=10 --vertical front porch width in rows
);
```

Figura 13.9. Declaración de constantes.

Se diseñará un módulo VRAM, con el fin de tener dos puertos síncronos de acceso. Ambos puertos tienen relojes independientes. La finalidad de tener dos puertos es para evitar el

diseño de una cola para las peticiones de acceso (lectura y escritura) a un chip RAM. El código de la memoria es mostrado en la figura 13.10.

```
wrvram:process (clken)
begin
  if rising_edge(clken) then
    memory(wraddr)<=data;
  end if;
end process wrvram;

rdvram: process (reloj_pixel)      -- sección de lectura
begin
  if falling_edge(reloj_pixel) then
    pixel<=memory(rdaddress);
  end if;
end process rdvram;
```

Figura 13.10. Código del módulo de memoria VRAM

La razón de esta memoria está en que la cámara maneja su propia señalización para enviar datos. Esta señalización es diferente de la que requiere el monitor VGA, por lo tanto, se requiere de un búfer que reciba datos de la cámara y que luego, pase los datos al monitor. El búfer diseñado para este proyecto tiene un doble puerto, cada uno con su propia señalización de reloj. Así, hay un puerto de escritura y hay un puerto de lectura.

La figura 13.11 muestra la carta ASM con la información de la cámara y como se generan direcciones de memoria para almacenar los datos y la figura 13.12 muestra el código del módulo Captura_pixel.

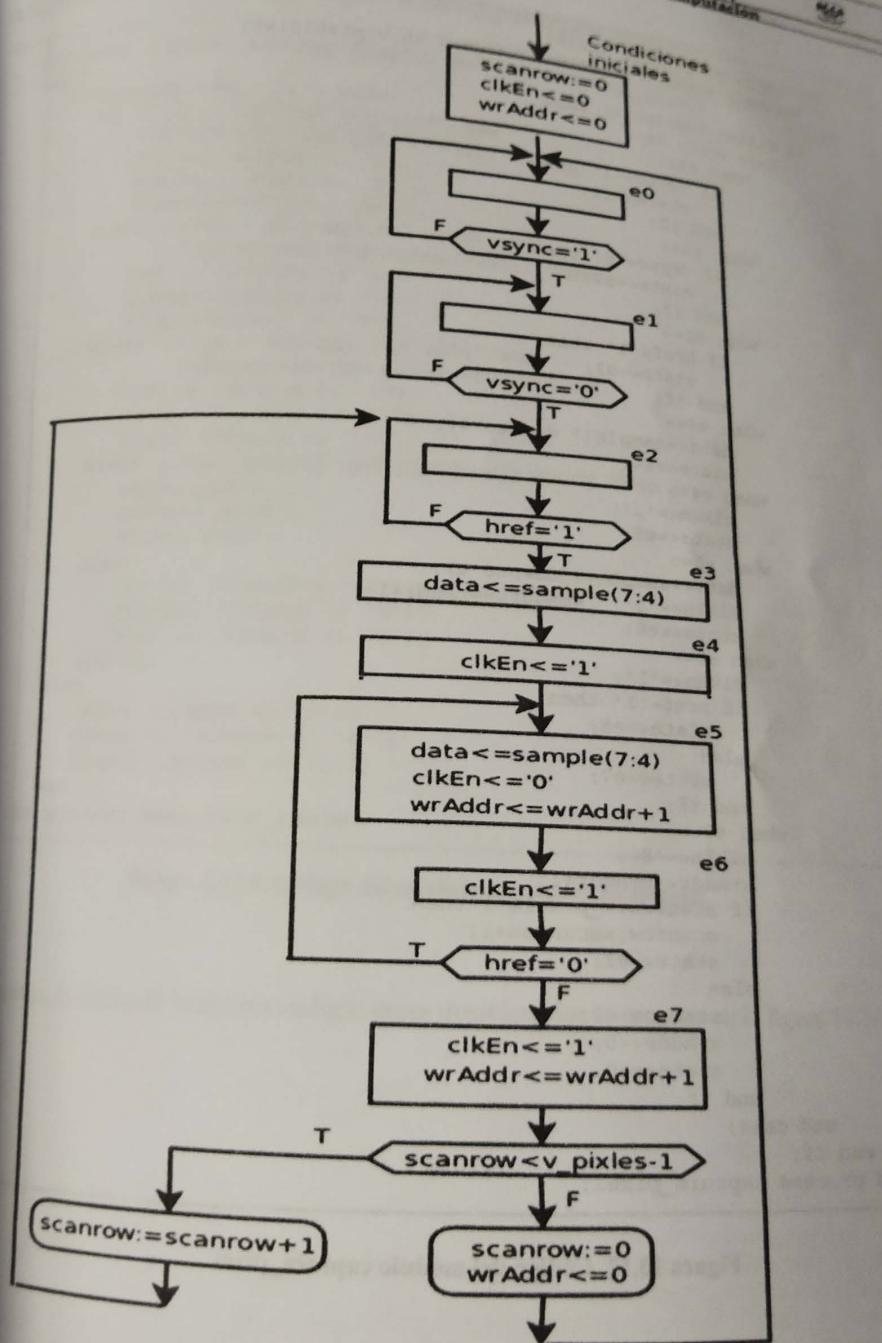


Figura 13.11. Carta ASM del módulo captura_pixel

Manual de prácticas Diseño Digital VLSI

División de Ingeniería Eléctrica

Departamento de Computación

```

captura_pixel:process (pclk)
variable scanrow : integer range 0 to v_pixels:=0;
begin
  if rising_edge(pclk) then
    case state is
      when e0=>
        if vsync='1' then
          state<=e1;
        end if;
      when e1=>
        if vsync='0' then
          state<=e2;
        end if;
      when e2=>
        if href='1' then
          state<=e3;
        end if;
      when e3=>
        data<=sample(7 downto 4);
        state<=e4;
      when e4=>
        clkEn<='1';
        state<=e5;
      when e5=>
        data<=sample(7 downto 4);
        clkEn<='0'; wrAddr<=wrAddr+1;
        state<=e6;
      when e6=>
        clkEn<='1';
        if href='1' then
          state<=e5;
        else
          state<=e7;
        end if;
      when e7=>
        clkEn<='0';
        wrAddr<=wrAddr+1;
        if scanrow<v_pixels-1 then
          scanrow:=scanrow+1;
          state<=e2;
        else
          scanrow:=0;
          wrAddr<=0;
          state<=e0;
        end if;
      end case;
    end if;
  end process captura_pixel;

```

Figura 13.12. Código del módulo captura_pixel

Manual de prácticas Diseño Digital VLSI
División de Ingeniería Eléctrica
Departamento de Computación

La figura 13.13 muestra el código requerido para el generador de imagen.

```
generador_imagen: process(display_ena, row, column,pixel)
begin
    if(display_ena = '1') then
        if ((row > 300 and row <350) and
            (column>350 and column<400)) then
            red <= (others => '1');
            green<=(others => '0');
            blue<=(others => '0');
        elsif ((row > 300 and row <350) and
               (column>450 and column<500)) then
            red <= (others => '0');
            green<=(others => '1');
            blue<=(others => '0');
        elsif ((row > 300 and row <350) and
               (column>550 and column<600)) then
            red <= (others => '0');
            green<=(others => '0');
            blue<=(others => '1');
        elsif (row< v_pixels and column< h_pixels) then
            red<= pixel;
            green<= pixel;
            blue<= pixel;
        else
            red <= (others => '0');
            green<= (others => '0');
            blue <= (others => '0');
        end if;
    else
        red<= (others => '0');
        green <= (others => '0');
        blue<= (others => '0');
    end if;
end process generador_imagen;
```

Figura 13.13. Código del módulo generador de imagen

Finalmente, la unión de todos los códigos antes mencionados se muestra en la figura 13.14.

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity OV7670 is
    generic(
        --constantes para monitor vga en 640x480
        constant h_pulse : integer:=96; --horizontal sync pulse width in pixels
        constant h_bp : integer:=48; --horizontal back porch width in pixels
        constant h_pixels: integer:=640;--horizontal display width in pixels
        constant h_fp : integer:=16;--horizontal front porch width in pixels
        constant v_pulse : integer:=2; --vertical sync pulse width in rows
        constant v_bp : integer:=33; --vertical back porch width in rows
        constant v_pixels: integer:=480;--vertical display width in rows
        constant v_fp : integer:=10 --vertical front porch width in rows
    );
    port( clk50MHz: in std_logic; --for this example is 50MHz
          -- reset : in std_logic; -- to monitor
          --pixel_clk: out std_logic; --for monitoring frequency
          red: out std_logic_vector (3 downto 0);
          green: out std_logic_vector (3 downto 0);
          blue: out std_logic_vector (3 downto 0);
          n_sync: out std_logic;
          n_blank: out std_logic;
          h_sync: out std_logic;
          v_sync: out std_logic;
          sio_c: out std_logic:='0'; -- to camera
          sio_d: out std_logic:='0';
          pwdn: out std_logic:='0';
          resetcamera: out std_logic:='1';
          xclk: out std_logic;
          pclk: in std_logic; -- from camera
          vsync: in std_logic;
          href: in std_logic;
          sample:in std_logic_vector (7 downto 0) );
end entity OV7670;
architecture behavioral of OV7670 is
    --Contadores
    signal h_period: integer := h_pulse + h_bp + h_pixels + h_fp;
    signal v_period: integer := v_pulse + v_bp + v_pixels + v_fp;
    signal h_count: integer range 0 to h_period := 0;
    signal v_count: integer range 0 to v_period - 1 := 0;
    --vram
    type matrix is array (0 to 307199) of std_logic_vector (3 downto 0);
    signal memory: matrix;
    signal memorysize : integer:=h_pixels*v_pixels;
    signal pixel: std_logic_vector( 3 downto 0);
    --pixel_capture - vram
    signal data: std_logic_vector( 3 downto 0);
    signal wrclk: std_logic;
    --pixel_capture: internal signals
    type states is (e0,e1,e2,e3,e4,e5,e6,e7);
    signal state: states:=e0;
    signal wraddr: integer range 0 to h_pixels*v_pixels:=0;
    signal clken: std_logic:='0';

```

Figura 13.14. Código final del sistema captura de imágenes de cámara digital

```

--Reloj de pixel
signal reloj_pixel: STD_LOGIC:='0';
--vga_controller - vram
signal rdaddress: integer:=0;
--vga_controller - hw_image_generator
signal display_ena: std_logic; --display enable ('1' = display time,
                                --'0' = blanking time)
signal column: integer;      --horizontal pixel coordinate
signal row: integer;         --vertical pixel coordinate

begin
    div25MHz: process (clk50MHz) is
    begin
        if rising_edge(clk50MHz) then
            reloj_pixel <= not reloj_pixel;
        end if;
    end process div25MHz;

    -- Reloj a la cámara
    XCLK<=reloj_pixel;

    -- Controlador del monitor
    Contadores : process (reloj_pixel)
    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 process Contadores;
    rdAddress <= column + (row * h_pixels);

    serial_hsync : process (h_count)
    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 serial_hsync;

```

Figura 13.14. (continuación) Código final del sistema captura de imágenes de cámara digital

```

    serial_vsync : process (v_count)
begin
    --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 serial_vsync;

coords_pixel_col: process(h_count)
begin
    if (h_count < h_pixels) then
        column <= h_count;
    end if;
end process coords_pixel_col;

coords_pixel_row: process(v_count)
begin
    if (v_count < v_pixels) then
        row <= v_count;
    end if;
end process coords_pixel_row;

generador_imagen: process(display_ena, row, column,pixel)
begin
    if(display_ena = '1') THEN          --display time
        if ((row > 300 and row <350) and
            (column>350 and column<400)) THEN
            red <= (others => '1');
            green<=(others => '0');
            blue<=(others => '0');
        elsif ((row > 300 and row <350) and
                (column>450 and column<500)) then
            red <= (others => '0');
            green<=(others => '1');
            blue<=(others => '0');
        elsif ((row > 300 and row <350) and
                (column>550 and column<600)) then
            red <= (others => '0');
            green<=(others => '0');
            blue<=(others => '1');
        elsif (row<v_pixels and column<h_pixels) then
            --yuv: solo luminancia
            red <= pixel;
            green <= pixel;
            blue <= pixel;
        else

```

Figura 13.14. (continuación) Código final del sistema captura de imágenes de cámara digital

```

        red <= (others => '0'); --es el fondo
        green <= (others => '0');
        blue <= (others => '0');
    end if;
else
    red <= (others => '0'); --blanking time
    green <= (others => '0');
    blue <= (others => '0');
end if;
end process generador_imagen;

captura_pixel: process (pclk)
--variable memorysize : integer := h_pixels*v_pixels;
variable scanrow      : integer range 0 to v_pixels:=0;
begin
    if rising_edge(pclk) then
        case state is
            when e0 =>
                if vsync='1' then
                    state <=e1;
                end if;
            when e1 =>
                if vsync='0' then
                    state <=e2;
                end if;
            when e2 =>
                if href='1' then
                    state <=e3;
                end if;
            when e3 =>
                data<=sample(7 downto 4);
                state <=e4;
            when e4 =>
                clkEn<='1';
                state<=e5;
            when e5 =>
                data<=sample(7 downto 4);
                clkEn<='0';
                wrAddr<=wrAddr+1;
                state<=e6;
            when e6 =>
                clkEn<='1';
                if href='1' then
                    state<=e5;
                else
                    state<=e7;
                end if;
            when e7 =>
                clkEn<='0';
                wrAddr<=wrAddr+1;
        end case;
    end if;
end process;

```

Figura 13.14. (continuación) Código final del sistema captura de imágenes de cámara digital

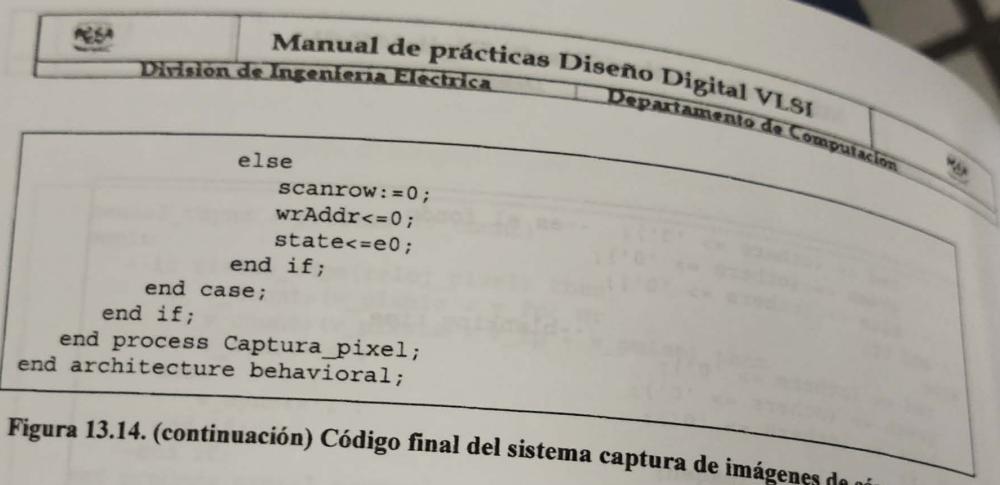


Figura 13.14. (continuación) Código final del sistema captura de imágenes de cámara digital

ACTIVIDADES COMPLEMENTARIAS:

El alumno investigará:

1. El funcionamiento de los sistemas de captura de imágenes BAYER, FOVEON X3.
2. El modelo de color YUV.
3. Cuántos detectores de luz de color rojo, de color verde y de color azul hay en el ojo humano.
4. El alumno investigará el tamaño de imagen que captura la cámara de su teléfono celular.
5. El significado de binarización de imágenes.