Diseño y simulación de un controlador gráfico para el despliegue de imágenes en movimiento utilizando un lenguaje de descripción de hardware.

Universidad Nacional Autónoma de México.
Facultad de Estudios Superiores Cuautitlán
Alonso Vargas Gachuz

1. Introducción

EL desarrollo de los circuitos digitales para el procesamiento y cálculo marcó un gran antecedente para la tecnología, durante el siglo XX, las computadoras solo cumplían un propósito orientado a la investigación por muchos años fue complicado introducir las computadoras para un uso personal debido a la complejidad que requerían para operarse además de que se necesitaba cierta experiencia para ello, una de las soluciones fue implementar interfaces visuales con las que se pudiera interactuar de forma más amigable con el operador, por lo que en un principio era necesario utilizar pantallas las cuales fueran capaces de desplegar imágenes y con ello también surgió la necesidad de implementar sistemas capaces de controlar lo que se muestra en pantalla. Las señales de video son complejas y de naturaleza analógica, como parte del desarrollo tecnológico surgieron nuevas técnicas para el diseño y control de pantallas que se volvieron estándares a seguir si se deseaba emplear cierto tipo de pantalla, el más popular y del cual muchos estándares actuales basan su funcionamiento es el estándar VGA(Video Graphics Array), en el, se especifican cinco señales para controlar pantallas analógicas las cuales son: sincronía horizontal, vertical y tres señales de color RGB. Con estas especificaciones es posible operar cualquier pantalla que emplee este estándar utilizando un controlador para manejar dichas señales y así poder desplegar imágenes en pantalla, para comprobarlo se propone el desarrollo de un controlador propio en el cual se muestren imágenes que interactúen entre sí utilizando sistemas digitales.

2. Descripción del método

Para la solución del planteamiento se propone seguir la metodología de arriba hacia abajo partiendo de un diseño general a uno particular, el primer paso es crear el controlador de la pantalla el cual manejará las señales de sincronía horizontal y vertical, para ello es necesario repasar la teoría en el manejo del tiempo en alto de las señales de sincronía para desplegar un pixel en pantalla, en general el controlador requerirá de contadores para mantener dichas señales, una vez obtenido el controlador se debe diseñar una maquina de estados la cual permitirá controlar el movimiento de la imagen en la pantalla, además de incluir bloques de memoria en dónde se almacenará la información de la imagen, al tener la máquina de estados terminada solo queda diseñar un controlador para mostrar cada pixel de la imagen y que esta se muestre de forma ordenada, de lo mencionado podemos entonces mostrar el diagrama planteado para el sistema.

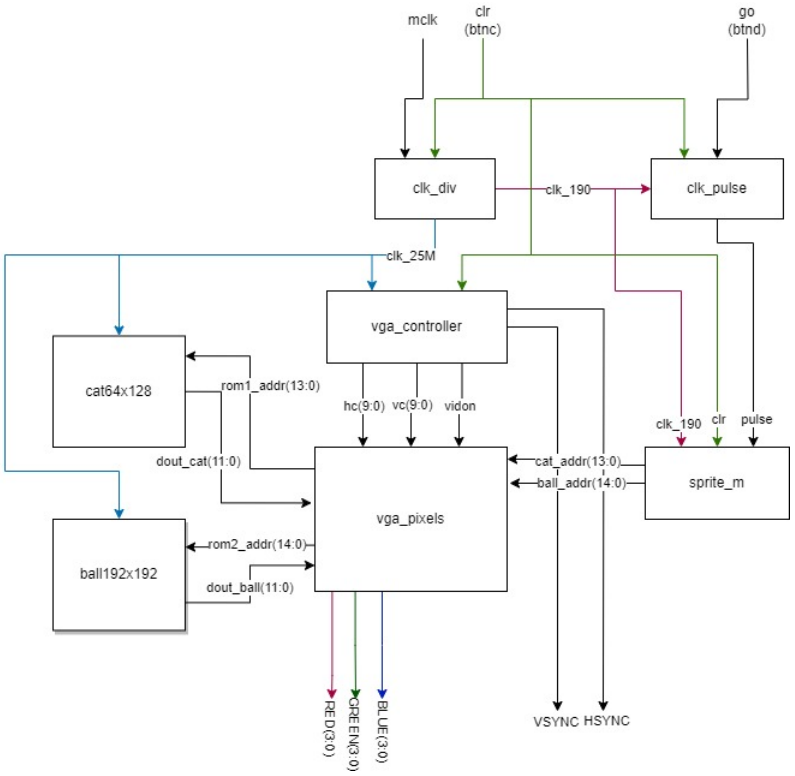

Figura 1: Diagrama general de la solución

El módulo clk_div es un divisor de frecuencia, del que se obtienen frecuencias de 25MHz para el controlador de video ya que es la frecuencia requerida según el estándar y una frecuencia de 190Hz para la máquina de estados que controla el movimiento, como se busca mostrar un movimiento visible es necesario emplear bajas frecuencias para que pueda ser observable. El módulo clk_pulse entregará un pequeño pulso para accionar la máquina de estados y que esta comience a funcionar. Los módulos de cat64x128 y ball64x448 son bloques de memoria donde se almacenan los sprites utilizados. El módulo sprite_m es la máquina de estados la cual hará que cambie el sprite en pantalla, de ese modo da la impresión de movimiento en ambas imágenes. Los valores de pixel de las imágenes se encuentran almacenados en bloques de memoria, cada imagen es un sprite de dimensiones 64x64 pixeles, un bloque contiene los diferentes sprites de cada imagen, la forma en como se almacena la información es en un vector que posee los coeficientes de cada pixel de la imagen avanzando de izquierda a derecha y de arriba hacia abajo, la función de la máquina de estados es la de proporcionar las direcciones de memoria en donde se encuentran los diferentes sprites, con ayuda de un contador se le da un cierto tiempo a la imagen para poder desplegarse antes que la máquina realice el cambio en la figura de este modo la imagen se moverá en la pantalla.

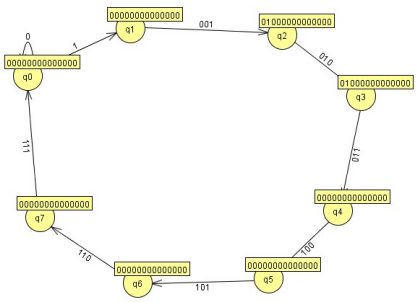

Figura 2: Diagrama de estados

La máquina posee ocho estados de los cuales el estado inicial mantiene la imagen en reposo hasta que se le indique, los otros siete estados son por cada uno de los sprites que contiene la imagen, en cada estado se muestra un sprite diferente.
Finalmente el módulo vga_pixels muestra la imagen según la dirección que obtiene de la máquina de estados y con ayuda de otros contadores se despliega cada uno de los pixels en la pantalla.

3. Experimentos

Para comprobar la funcionalidad de cada uno de los módulos se realizó una simulación de su comportamiento antes de integrarlos en el sistema final, cada módulo fue probado por separado además de realizarse la comprobación de la unión de todos los sistemas. La comprobación del funcionamiento del sistema se encuentra en el apéndice C de este trabajo.

4. Conclusiones

Los sistemas de video son muy importantes en la actualidad, permiten mostrar de forma gráfica una gran cantidad de opciones por lo que tiene muchas aplicaciones, ya sea para mostrar una interfaz gráfica, imágenes, entretenimiento, etc. Dadas sus aplicaciones es casi necesario que los diseños nuevos muestren una interfaz clara usando pantallas. Como desarrolladores es inevitable hacer uso en algún momento de imágenes en pantalla por lo que es importante conocer su funcionamiento y operación para hacer un uso adecuado de estas.

5. Apéndice A

Diagrama detallado de la solución

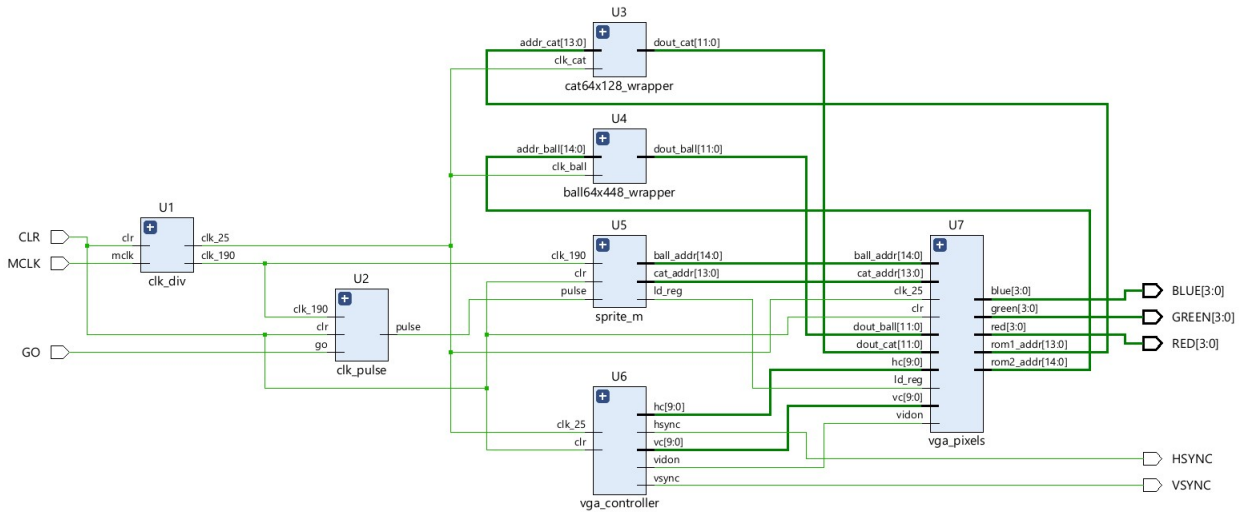

Figura 3: Esquema total

6. Apéndice B

Código en VHDL
vga_sprites_top:

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

entity vga_sprites_top is
    port(   MCLK: in std_logic;
            CLR: in std_logic;
            GO: in std_logic;
            HSYNC: out std_logic;
            VSYNC: out std_logic;
            RED: out std_logic_vector (3 downto 0);
            GREEN: out std_logic_vector (3 downto 0);
            BLUE: out std_logic_vector (3 downto 0));
end vga_sprites_top;

architecture Behavioral of vga_sprites_top is
signal clk_25M: std_logic;
signal clk_190H: std_logic;
signal out_pulse: std_logic;
signal rom1_a: std_logic_vector (13 downto 0);
signal cat_data: std_logic_vector (11 downto 0);
signal rom2_a: std_logic_vector (14 downto 0);
signal ball_data: std_logic_vector (11 downto 0);
signal hor_count: std_logic_vector (9 downto 0);
signal ver_count: std_logic_vector (9 downto 0);
signal vid_on: std_logic;
signal load_reg: std_logic;
signal cat_address: std_logic_vector (13 downto 0);
signal ball_address: std_logic_vector (14 downto 0);

begin
    U1: entity work.clk_div port map(
        mclk => MCLK,
        clr => CLR,
        clk_25 => clk_25M,
        clk_190 => clk_190H
    );

    U2: entity work.clk_pulse port map(
        clk_190 => clk_190H,
        clr => CLR,
        go => GO,
        pulse => out_pulse
    );

    U3: entity work.cat64x128_wrapper port map(
        addr_cat => rom1_a,
```

```
        clk_cat => clk_25M,
        dout_cat => cat_data
    );

    U4: entity work.ball64x448_wrapper port map(
        addr_ball => rom2_a,
        clk_ball => clk_25M,
        dout_ball => ball_data
    );

    U5: entity work.sprite_m port map(
        clk_190 => clk_190H,
        clr => CLR,
        pulse => out_pulse,
        ld_reg => load_reg,
        cat_addr => cat_address,
        ball_addr => ball_address
    );

    U6: entity work.vga_controller port map(
        clk_25 => clk_25M,
        clr => CLR,
        hc => hor_count,
        vc => ver_count,
        hsync => HSYNC,
        vsync => VSYNC,
        vidon => vid_on
    );

    U7: entity work.vga_pixels port map(
        clk_25 => clk_25M,
        clr => CLR,
        ld_reg => load_reg,
        hc => hor_count,
        vc => ver_count,
        vidon => vid_on,
        cat_addr => cat_address,
        ball_addr => ball_address,
        dout_cat => cat_data,
        dout_ball => ball_data,
        rom1_addr => rom1_a,
        rom2_addr => rom2_a,
        red => RED,
        green => GREEN,
        blue => BLUE
    );

end Behavioral;
```

---

clk_div:

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

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clk_div is
    port(   mclk:    in std_logic;
            clr:     in std_logic;
            clk_25: out std_logic;
            clk_190:out std_logic);
end clk_div;

architecture Behavioral of clk_div is
--counter signals
signal counter1: std_logic_vector(1 downto 0);
signal counter2: std_logic_vector(18 downto 0);
--temporal output values
signal tmp1: std_logic:= '0';
signal tmp2: std_logic:= '0';

begin
    clock_25M: process(mclk, clr)
    begin
        if (clr = '1') then
            counter1 <= "00";
        elsif (rising_edge(mclk)) then
            if (counter1(0) = '1') then
                counter1 <= (others => '0');
                tmp1 <= not tmp1;
            else
                counter1 <= counter1 + 1;
            end if;
        end if;
    end process clock_25M;
    clk_25 <= tmp1;

    clock_190: process(mclk, clr)
    begin
        if(clr = '1') then
            counter2 <= (others => '0');
        elsif (rising_edge(mclk)) then
            if (counter2(18) = '1') then
                counter2 <= (others => '0');
                tmp2 <= not tmp2;
            else
                counter2 <= counter2 + 1;
            end if;
        end if;
    end process clock_190;
    clk_190 <= tmp2;

end Behavioral;
```

---

clk_pulse:

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clk_pulse is
    port(   clk_190: in std_logic;
            clr: in std_logic;
            go: in std_logic;
            pulse: out std_logic);
end clk_pulse;

architecture Behavioral of clk_pulse is
signal delay1: std_logic;
signal delay2: std_logic;
signal delay3: std_logic;

begin
    delays: process(clr, clk_190)
    begin
        if(clr = '1') then
            delay1 <= '0';
            delay2 <= '0';
            delay3 <= '0';
        elsif(rising_edge(clk_190)) then
            delay1 <= go;
            delay2 <= delay1;
            delay3 <= delay2;
            pulse <= delay1 and delay2 and (not delay3);
        end if;
    end process delays;

end Behavioral;
```

---

sprite_m:

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

entity sprite_m is
    port(   clk_190: in std_logic;
            clr: in std_logic;
            pulse: in std_logic;
            ld_reg: out std_logic;
            cat_addr: out std_logic_vector (13 downto 0);
            ball_addr: out std_logic_vector (14 downto 0));
end sprite_m;

architecture Behavioral of sprite_m is
type state_type is (q0, q1, q2, q3, q4, q5 ,q6, q7);
signal present_state, next_state: state_type;
signal count: std_logic_vector (5 downto 0);
signal frames: std_logic_vector (2 downto 0);
signal next_sprite: std_logic;
```

```
signal counter_en: std_logic;

begin
    reg: process(clk_190, clr, pulse, frames)
    begin
        if (clr = '1') then
            counter_en <= '0';
        elsif (rising_edge(clk_190)) then
            if (pulse = '1') then
                counter_en <= '1';
            elsif (frames = "111") then
                counter_en <= '0';
            end if;
        end if;
    end process reg;

    counter: process(clk_190, clr, counter_en)
    begin
        if (clr = '1' or counter_en = '0') then
            count <= "000000";
        elsif (rising_edge(clk_190) and (counter_en = '1')) then
            if (count = "110000") then
                count <= "000000";
                next_sprite <= '1';
                ld_reg <= '1';
            else
                count <= count + 1;
                next_sprite <= '0';
                ld_reg <= '0';
            end if;
        end if;
    end process counter;

    frame_c: process(clr, clk_190, next_sprite, counter_en)
    begin
        if (clr = '1' or counter_en = '0') then
            frames <= "000";
        elsif(rising_edge(clk_190) and (next_sprite = '1') and (counter_en = '1'
            if (frames = "111") then
                frames <= "000";
            else
                frames <= frames + 1;
            end if;
        end if;
    end process frame_c;

    state_register: process(clk_190, clr)
    begin
        if(clr = '1') then
            present_state <= q0;
        elsif (rising_edge(clk_190)) then
```

```
        present_state <= next_state;
    end if;
end process state_register;

C1: process(present_state, frames, pulse)
begin
    case (present_state) is
        when q0 =>
            if (pulse = '0') then
                next_state <= q0;
            else
                next_state <= q1;
            end if;
        when q1 =>
            if (frames = "001") then
                next_state <= q2;
            else
                next_state <= q1;
            end if;
        when q2 =>
            if (frames = "010") then
                next_state <= q3;
            else
                next_state <= q2;
            end if;
        when q3 =>
            if (frames = "011") then
                next_state <= q4;
            else
                next_state <= q3;
            end if;
        when q4 =>
            if (frames = "100") then
                next_state <= q5;
            else
                next_state <= q4;
            end if;
        when q5 =>
            if (frames = "101") then
                next_state <= q6;
            else
                next_state <= q5;
            end if;
        when q6 =>
            if (frames = "110") then
                next_state <= q7;
            else
                next_state <= q6;
            end if;
        when q7 =>
            if (frames = "111") then
```

```
                    next_state <= q0;
                else
                    next_state <= q7;
                end if;
            when others =>
                null;
        end case;
    end process C1;

    C2: process(present_state)
    begin
        if (present_state = q0) then
            cat_addr <= (others => '0');
            ball_addr <= (others => '0');
        elsif (present_state = q1) then
            cat_addr <= (others => '0');
            ball_addr <= (others => '0');
        elsif (present_state = q2) then
            cat_addr <= "01000000000000";
            ball_addr <= "001000000000000";
        elsif (present_state = q3) then
            cat_addr <= "01000000000000";
            ball_addr <= "010000000000000";
        elsif (present_state = q4) then
            cat_addr <= (others => '0');
            ball_addr <= "011000000000000";
        elsif (present_state = q5) then
            cat_addr <= (others => '0');
            ball_addr <= "100000000000000";
        elsif (present_state = q6) then
            cat_addr <= (others => '0');
            ball_addr <= "101000000000000";
        elsif (present_state = q7) then
            cat_addr <= (others => '0');
            ball_addr <= "110000000000000";
        else
            cat_addr <= (others => '0');
            ball_addr <= (others => '0');
        end if;
    end process C2;

end Behavioral;
```

---

vga_controller:

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

entity vga_controller is
    port(   clk_25: in std_logic;
            clr:    in std_logic;
            hc: out std_logic_vector (9 downto 0);
```

```
            vc: out std_logic_vector (9 downto 0);
            hsync: out std_logic;
            vsync: out std_logic;
            vidon: out std_logic);
end vga_controller;

architecture Behavioral of vga_controller is
--horizontal timing
constant hbp: std_logic_vector (9 downto 0):= "0010010000"; --hbp=sp+bp=96+48=14
constant hfp: std_logic_vector (9 downto 0):= "1100010000"; --hfp=hbp+hv=144+640
constant hpixels: std_logic_vector (9 downto 0):= "1100100000";
--whole horizontal line=sp+bp+va+fp=96+48+640+16=800

--vertical timing
constant vbp: std_logic_vector (9 downto 0):= "0000100011"; --vbp=sp+bp=2+33=35
constant vfp: std_logic_vector (9 downto 0):= "1000000011"; --hfp=vbp+vv=35+480
constant vlines: std_logic_vector (9 downto 0):= "1000001101";
--whole vertical line=sp+bp+va+fp=2+33+480+10=525

--signals for counters
signal hcs: std_logic_vector (9 downto 0);
signal vcs: std_logic_vector (9 downto 0);

--signal for vertical count
signal vsenable: std_logic;

begin
    hor_count: process(clr, clk_25)
    begin
        if (clr = '1') then
            hcs <= (others => '0');
        elsif (rising_edge(clk_25)) then
            if (hcs = hpixels-1) then
                hcs <= (others => '0');
                vsenable <= '1';
            else
                hcs <= hcs + 1;
                vsenable <= '0';
            end if;
        end if;
    end process hor_count;
    hsync <= '0' when (hcs < hbp) or (hcs > hfp) else '1';

    ver_count: process(clr, clk_25, vsenable)
    begin
        if (clr = '1') then
            vcs <= (others => '0');
        elsif (rising_edge(clk_25) and (vsenable = '1')) then
            if (vcs = vlines-1) then
                vcs <= (others => '0');
            else
```

```
                vcs <= vcs + 1;
            end if;
        end if;
    end process ver_count;
    vsync <= '0' when (vcs < vbp) or (vcs > vfp) else '1';
    vidon <= '1' when (((hcs < hfp)and(hcs>=hbp))and((vcs<vfp)and(vcs>=vbp))) el

    hc <= hcs;
    vc <= vcs;
end Behavioral;
```

---

vga_pixels:

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

entity vga_pixels is
    port(   clk_25: in std_logic;
            clr: in std_logic;
            ld_reg: in std_logic;
            hc: in std_logic_vector (9 downto 0);
            vc: in std_logic_vector (9 downto 0);
            vidon: in std_logic;
            cat_addr:  in std_logic_vector (13 downto 0);
            ball_addr: in std_logic_vector (14 downto 0);
            dout_cat:  in std_logic_vector (11 downto 0);
            dout_ball: in std_logic_vector (11 downto 0);
            rom1_addr: out std_logic_vector (13 downto 0);
            rom2_addr: out std_logic_vector (14 downto 0);
            red:    out std_logic_vector (3 downto 0);
            green:  out std_logic_vector (3 downto 0);
            blue:   out std_logic_vector (3 downto 0));
end vga_pixels;

architecture Behavioral of vga_pixels is
--screen timing
constant hbp: std_logic_vector (9 downto 0):= "0010010000";
constant vbp: std_logic_vector (9 downto 0):= "0000100011";

--dimensions of the sprites: 64x64 every frame
constant s_dim: std_logic_vector (6 downto 0):= "1000000";
constant ds_dim: std_logic_vector (7 downto 0):= "10000000";

--column and row position
constant R1: std_logic_vector (9 downto 0):= "0011110000";
constant C1: std_logic_vector (9 downto 0):= "0011110000";

--detects the sprites regions in the screen
signal sprite_cat: std_logic;
signal sprite_ball: std_logic;

--mem control
```

```
constant pixel_num: std_logic_vector (12 downto 0) := "1000000000000";
signal cat_counter: std_logic_vector (13 downto 0);
signal ball_counter: std_logic_vector (14 downto 0);

--registers
signal temp_cat_add: std_logic_vector (13 downto 0);
signal temp_ball_add: std_logic_vector (14 downto 0);

begin
    --sprite_cat = 1 when its in the range to display the image
    sprite_cat <= '1' when ((hc >= C1+hbp) and (hc < C1+hbp+s_dim))
    and ((vc >= R1+vbp) and (vc < R1+s_dim+vbp)) else '0';

    --sprite_ball = 1 when its in the range to display the image
    sprite_ball <= '1' when ((hc >= C1+hbp+s_dim) and (hc < C1+hbp+ds_dim))
    and ((vc >= R1+vbp) and (vc < R1+vbp+s_dim)) else '0';

    reg: process (clk_25, clr, ld_reg)
    begin
        if (clr = '1') then
            temp_cat_add <= (others => '0');
            temp_ball_add <= (others => '0');
        elsif (rising_edge(clk_25)) then
            if (ld_reg = '1') then
                temp_cat_add <= cat_addr;
                temp_ball_add <= ball_addr;
            end if;
        end if;
    end process reg;

    count_scat: process (clk_25, clr, sprite_cat, vidon)
    begin
        if (clr = '1') then
            cat_counter <= (others => '0');
        elsif (rising_edge(clk_25) and vidon = '1' and sprite_cat = '1') then
            if (cat_counter = pixel_num - 1) then
                cat_counter <= (others => '0');
            else
                cat_counter <= cat_counter + 1;
            end if;
        end if;
    end process count_scat;
    rom1_addr <= temp_cat_add + cat_counter;

    count_sball: process(clk_25, sprite_ball, vidon)
    begin
        if (clr = '1') then
            ball_counter <= (others => '0');
        elsif (rising_edge(clk_25) and vidon = '1' and sprite_ball = '1') then
            if (ball_counter = pixel_num - 1) then
                ball_counter <= (others => '0');
```

```
            else
                ball_counter <= ball_counter + 1;
            end if;
        end if;
    end process;
    rom2_addr <= temp_ball_add + ball_counter;

    show_pixels: process(sprite_cat, sprite_ball, vidon)
    begin
        red <= "0000";
        green <= "0000";
        blue <= "0000";
        if (sprite_cat = '1' and vidon <= '1') then
            red <= dout_cat(3 downto 0);
            green <= dout_cat(7 downto 4);
            blue <= dout_cat(11 downto 8);
        elsif (sprite_ball = '1' and vidon = '1') then
            red <= dout_ball(3 downto 0);
            green <= dout_ball(7 downto 4);
            blue <= dout_ball(11 downto 8);
        end if;
    end process show_pixels;

end Behavioral;
```

7. Apéndice C

Resultados de la simulación

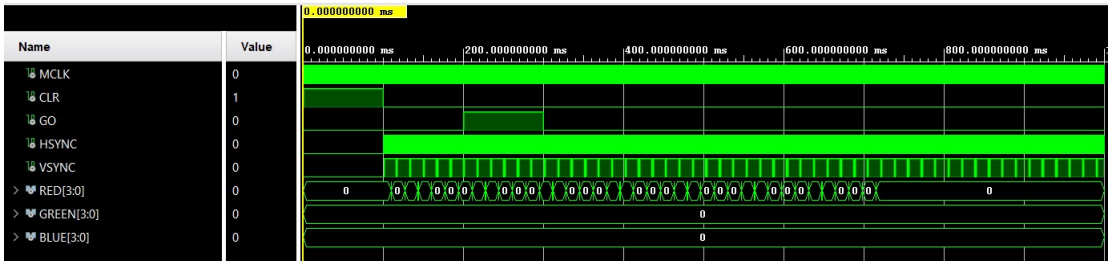

Figura 4: Comportamiento del sistema total

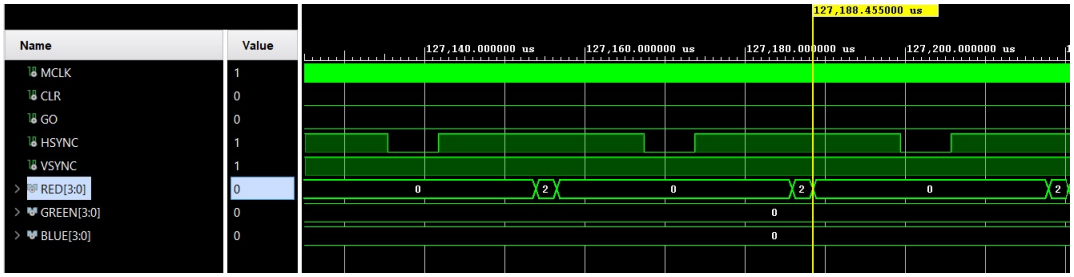

Figura 5: salida de los valores de cada pixel en su correspondiente posición