

SE RECOMIENDA ACCEDER AL CONTENIDO A TRAVÉS DEL SIGUIENTE ENLACE:

[http://prezi.com/rf7zu14nk325/?utm\\_campaign=share&utm\\_medium=copy](http://prezi.com/rf7zu14nk325/?utm_campaign=share&utm_medium=copy)

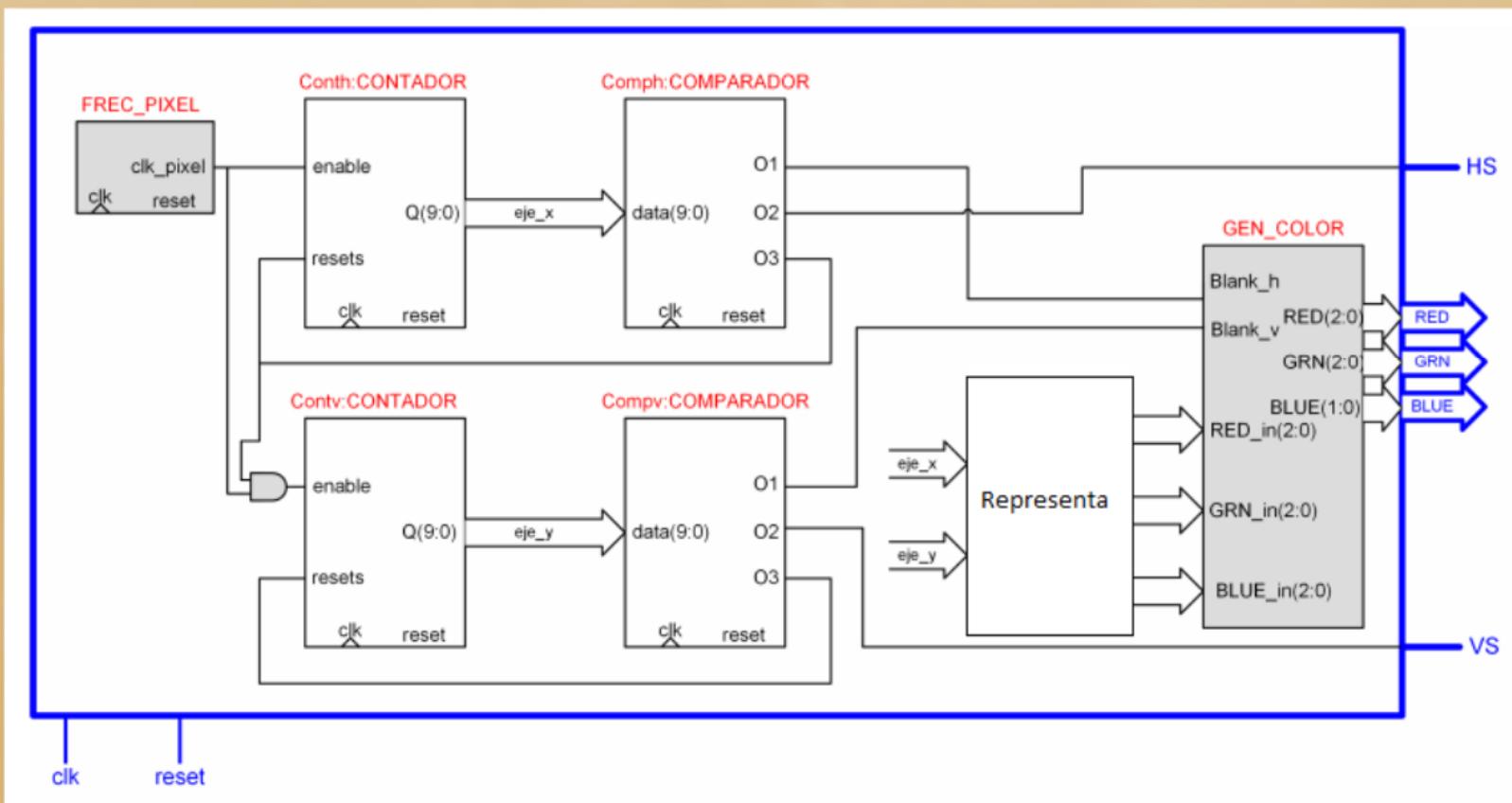
# SNAKE

# GAME

## **ÍNDICE**

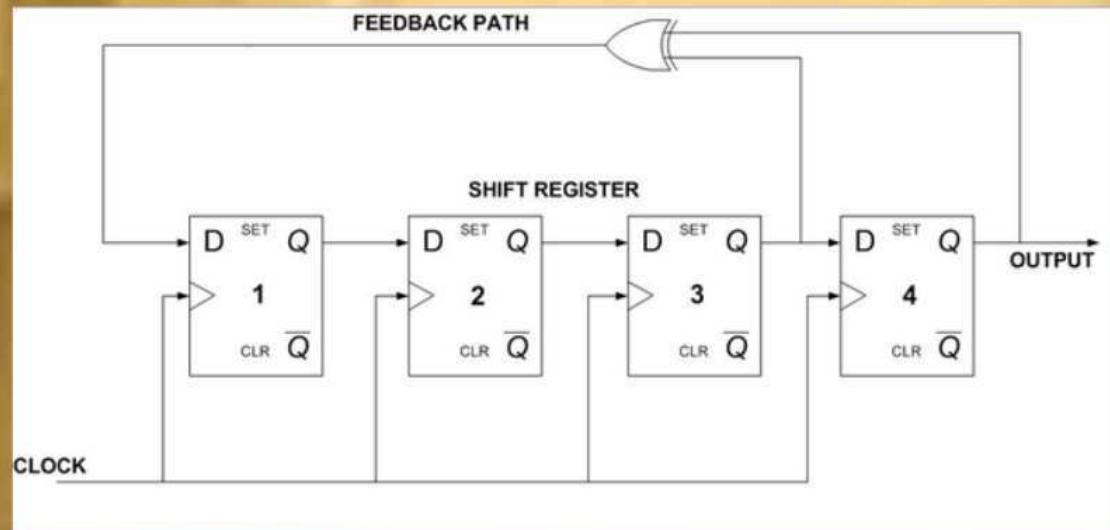
- 1. Detalles del juego.**
- 2. Componentes del juego.**
- 3. Warnings.**
- 4. Limitaciones y bugs.**
- 5. Demo.**

# VGA Driver

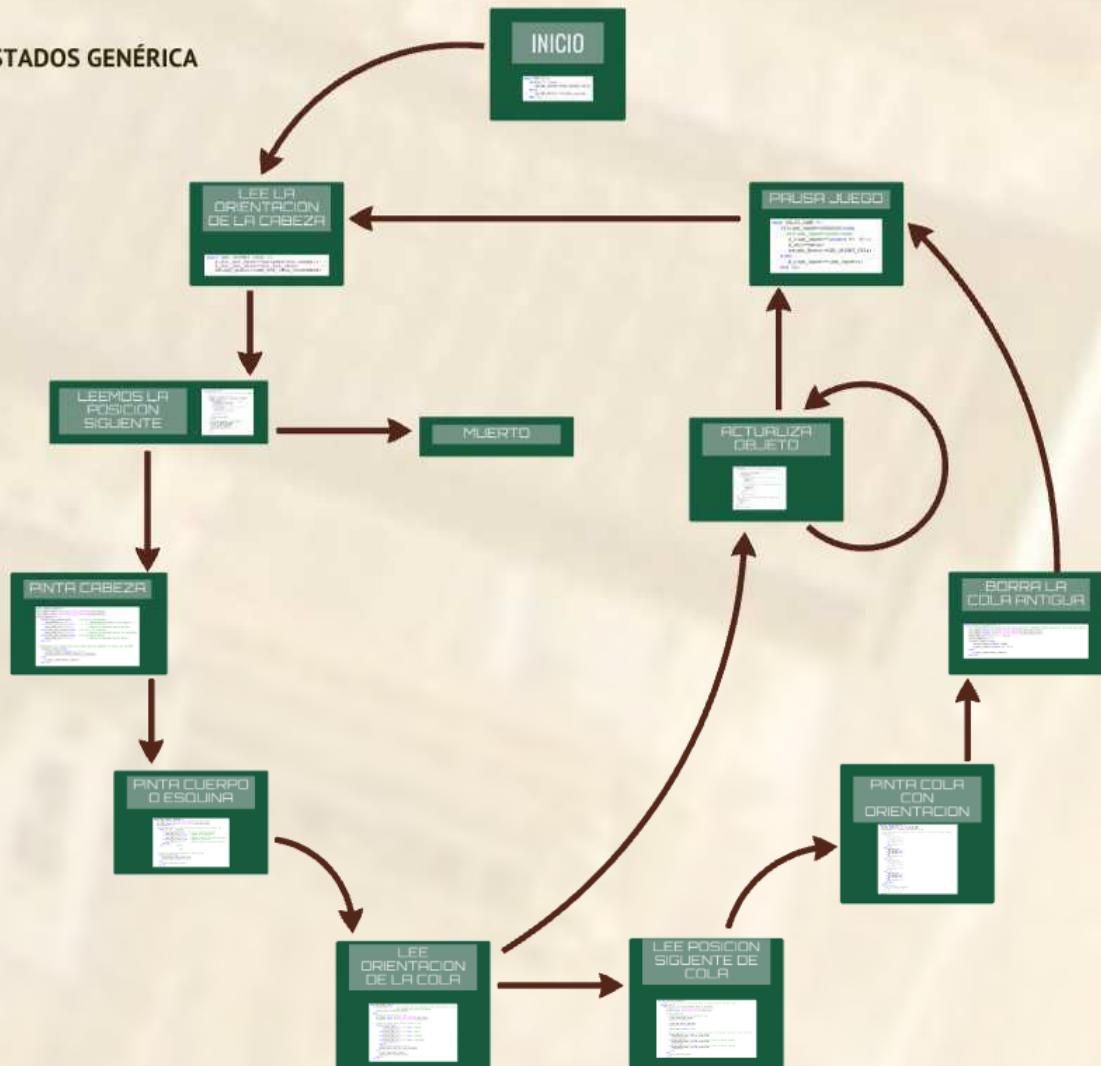


# **LFSR ( Linear Feedback Shift Register )**

Consiste en un registro de desplazamiento de 10 bits, donde a la entrada tenemos el resultado de la operación lógica XOR del bit 8 y 9 negado.



### MÁQUINA DE ESTADOS GENÉRICA

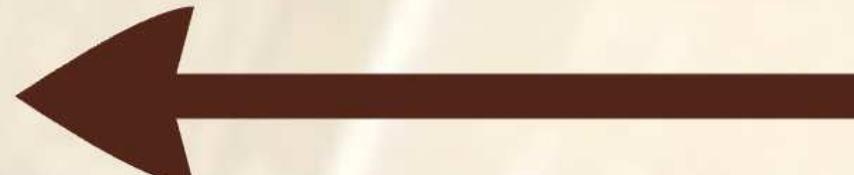


# INICIO

```
when INICIO =>
    if(btn='1')then
        estado_Nuevo<=LEE_ORIENT_CBZA;
    else
        estado_Nuevo<=estado_Actual;
    end if;
```



## LEE LA ORIENTACION DE LA CABEZA



```
when LEE_ORIENT_CBZA =>
    p_dir_act_cbza<=unsigned(dir_snake); --
    p_dir_ant_cbza<=dir_act_cbza; --
    estado_Nuevo<=LEE_POS_CBZA_SIGUIENTE;
```

# LEEMOS LA POSICION SIGUIENTE

```
when LEFT_POS_CABLA_SIGUIENTE =>
  when dir_mir_chka is
    when "00" =>
      -- La cabecera esta orientada hacia la DERECHA
      -- Leemos lo que hay en esa casilla
      dir_RAM(4 downto 0) <= TB_LOGIC_VECTOR(p_pos_chka);
      dir_RAM(9 downto 5) <= TB_LOGIC_VECTOR(p_pos_chka);
      if(wait_time)then
        if(data_RAM_out="00000")then --VACIO ?
          estado_Nuevo<-0DATA_CABEZA;
          p_obj:=vacio;
        elsif(data_RAM_out="10001")then --MANTAMA ?
          p_obj:=mantenimiento;
        else
          estado_Nuevo<-0DATA_CABEZA;
        end if;
      else
        if(data_RAM_out="10001")then --MANTAMA ?
          p_obj:=mantenimiento;
        elsif(data_RAM_out="10000")then --PARED ?
          estado_Nuevo<-MURENO;
          p_obj:=wall;
        else
          estado_Nuevo<-SILENT;
        end if;
      end if;
      --ACTUALIZAMOS VARIABLES DE POSICION CABEZA
      p_x_pos_ant_chka<-p_pos_chka;
      p_y_pos_ant_chka<-p_pos_chka;
      p_x_pos_chka<-p_pos_chka;
      p_y_pos_chka<-p_pos_chka;
      p_pos_chka<-p_pos_chka;
      p_wait_time<-0TIME;
      if(p_pos_chka = '0')/
    else
      p_wait_time<-wait_time+1;
    end if;
```

# PINTA CABEZA

```
when PINTA_CABEZA =>
dir_RAM(4 downto 0)<=STD_LOGIC_VECTOR(x_pos_cbza);
dir_RAM(9 downto 5)<=STD_LOGIC_VECTOR(y_pos_cbza);
write_Enable<="1";
    if(dir_act_cbza=0)then      --Si va a la derecha
        data_RAM_in<="00011";   --Cabeza orientada a la derecha
    elsif(dir_act_cbza=1)then   --Si va hacia arriba
        data_RAM_in<="00001";   --Cabeza orientada hacia arriba
    elsif(dir_act_cbza=2)then   --Si va a la izquier
        data_RAM_in<="00100";   --Cabeza orientada hacia la iquierda
    elsif(dir_act_cbza=3)then   --Si va hacia abajo
        data_RAM_in<="00010";   --Cabeza orientada hacia abajo
    end if;

--Esperamos un tiempo de reloj para que se guarde el valor en la RAM.
if(wait_time=1)then
    p_wait_time<=(others => '0');
    estado_Nuevo<=PINTA_CUERPO_O_ESQUINA;
else
    p_wait_time<=wait_time+1;
end if;
```

# PINTA CUERPO O ESQUINA

```
when PINTA_CUERPO_O_ESQUINA =>
    dir_RAM(4 downto 0)<=STD_LOGIC_VECTOR(x_pos_ant_cbza);
    dir_RAM(9 downto 5)<=STD_LOGIC_VECTOR(y_pos_ant_cbza);
    write_Enable<="1";
    case dir_ant_cbza is -- Si la direccion anterior de la cabeza fue...
        when "00" =>    --DERECHA
            if(dir_act_cbza=0)then --Si va en la misma direccion
                data_RAM_in<="00111"; --Cuerpo recto horizontal
            elsif(dir_act_cbza=1)then --Cambia de direccion
                data_RAM_in<="01011"; --Esquina (hacia la derecha subiendo)
            elsif(dir_act_cbza=3)then --Cambia de direccion
                data_RAM_in<="01100"; --Esquina (hacia la derecha bajando)
            end if;
        when "01" =>    --ARRIBA
            etc..
--Tiempo de espera para guardar el valor en ram.
        if(wait_time=1)then
            estado_Nuevo<=LEE_ORIENT_COLA;
            p_wait_time<=(others => '0');
        else
            p_wait_time<=wait_time+1;
        end if;
```

# LEE ORIENTACION DE LA COLA

```
oben LEE_ORIENT_COLA =>
    if(obj==manzana)then -- Si el objeto fue una manzana no hace falta borrar la cola
        -- asi tenemos que crecer la cabeza
        estado_Nuevo=>ACTUALIZA_OBJETO;
    else
        -- Leemos la posicion de la cola
        dic_RAM[4 downto 0]<-STD_LOGIC_VECTOR(x_pos_col);
        dic_RAM[9 downto 5]<-STD_LOGIC_VECTOR(y_pos_col);

        -- Tiempo de espera para cargarlo desde la ram
        if(wait_time=1)then
            if(data_RAM_out="01101")then --ARRIBA
                p_dir_col=>"01";
            elsif(data_RAM_out="01110")then --ABAJO
                p_dir_col=>"11";
            elsif(data_RAM_out="01111")then --DERECHA
                p_dir_col=>"00";
            elsif(data_RAM_out="10000")then --IZQUIERDA
                p_dir_col=>"10";
            end if;
            p_wait_time=>iothera => '0';
            estado_Nuevo<-LEE_POS_COLA_SIGUIENTE;
        else
            p_wait_time=>wait_time+1;
            estado_Nuevo<-estado_Actual;
        end if;
    end if;
```

# LEE POSICION SIGUIENTE DE COLA

```
when LEE_POG_COLA_SIGUIENTE =>
  -- Sabiendo la dirección de la cola podemos saber la siguiente posición que debe ocupar.
  caso dir_cola is
    when "000" => -- Si la punta apunta hacia la izquierda
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col+1);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col);
      dir_x;
    when "001" => -- Si la punta apunta hacia la derecha
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col-1);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col);
      dir_x;
    when "010" => -- Si la punta apunta hacia arriba
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col-1);
      dir_y;
    when "011" => -- Si la punta apunta hacia abajo
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col+1);
      dir_y;
    when "100" => -- Si la punta apunta hacia la izquierda
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col+1);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col);
      dir_x;
    when "101" => -- Si la punta apunta hacia la derecha
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col-1);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col);
      dir_x;
    when "110" => -- Si la punta apunta hacia arriba
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col-1);
      dir_y;
    when "111" => -- Si la punta apunta hacia abajo
      dir_RAM14_downto 0|<=STE_LOGIC_VECTOR(x_pos_col);
      dir_RAM10_downto 5|<=STE_LOGIC_VECTOR(y_pos_col+1);
      dir_y;
  end caso;
  if wait_time<=(others => '0') then
    -- ACTUALIZAMOS VARIABLES DE POSICION DE COLA
    p_x_pos_col<=x_pos_col+1;
    p_y_pos_col<=y_pos_col;
    p_x_pos_ant_col<=x_pos_col;
    p_y_pos_ant_col<=y_pos_col;
    p_wait_time<=wait_time;
  else
    -- Aquí determinaremos si la cola se tiene que pintar a la derecha, izquierda o igual que antes.
    if (data_RAM_out="00111")then -- Si es cuerpo
      estado_Nuevo<=PINTA_COLA_CON_ORIENTACION;
      p_side="straight";
    elsif(data_RAM_out="01100")then -- si esquina, hacia la derecha bajando
      estado_Nuevo<=PINTA_COLA_CON_ORIENTACION;
      p_side="right";
    elsif(data_RAM_out="01011")then -- si esquina, hacia la derecha subiendo
      estado_Nuevo<=PINTA_COLA_CON_ORIENTACION;
      p_side="left";
    end if;
  else
    p_wait_time<=wait_time+1;
  end if;
end when;
```

# PINTA COLA CON ORIENTACION

```
when PINTA_COLA_CON_ORIENTACION =>
    dir_WAR[4] <=CTR_LOGIC_VECTOR[4_pos_cola];
    dir_WAR[9] <=CTR_LOGIC_VECTOR[9_pos_cola];
    write_Busdata<=1; -- Preparamos para escribir en la ram
    --Segun la direccion de la cola actual y en que lado hay que pintar la cola siguiente, pintamos.
    case dir_cola is
        when "00" =>
            if [id:left]then
                data_ZAR_ino="01101";
            else[id:right]then
                data_ZAR_ino="00001";
            end if;
        when "01" =>
            if [id:left]then
                data_ZAR_ino="00000";
            else[id:right]then
                data_ZAR_ino="01111";
            end if;
        when "10" =>
            if [id:left]then
                data_ZAR_ino="01100";
            else[id:right]then
                data_ZAR_ino="00000";
            end if;
        when "11" =>
            if [id:left]then
                data_ZAR_ino="01110";
            else[id:right]then
                data_ZAR_ino="00000";
            end if;
        when others =>
            data_ZAR_ino="00000";
    end case;
    if(wait_time<1)then
        wait_time<=BORDA_COLA_ANTIGUA;
        P_wait_time<=others =>"0";
    else
        P_wait_time<=wait_time+2;
    end if;
end when;
```

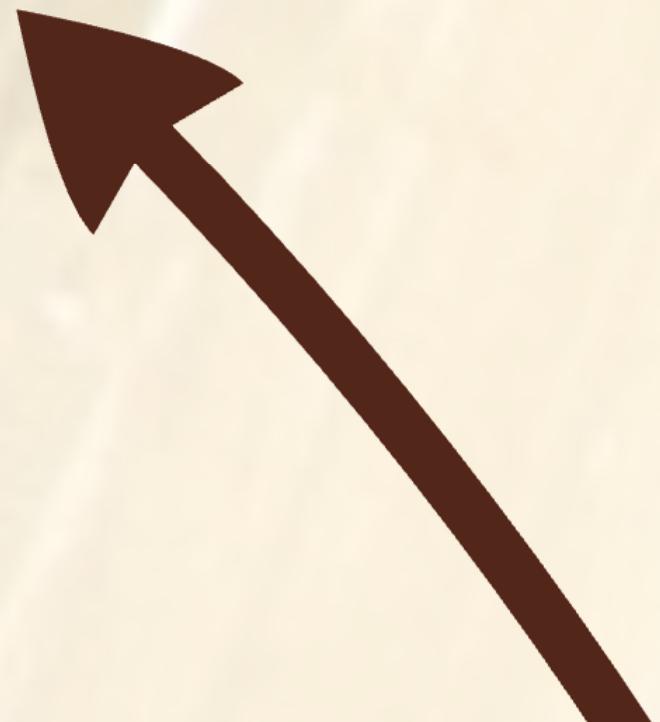


# BORRA LA COLA ANTIGUA

```
when BORRA_COLA_ANTIGUA =>
-- Con la variable de posicion de cola anterior podemos sobre escribir la cola con vacio.
    dir_RAM(4 downto 0)<=STD_LOGIC_VECTOR(x_pos_ant_col);
    dir_RAM(9 downto 5)<=STD_LOGIC_VECTOR(y_pos_ant_col);
    data_RAM_in<="00000"; -- Vacio
    write_Enable<="1";
    if(wait time=1)then
        estado Nuevo<=DELAY GAME;
        p_wait_time<=(others => '0');
    else
        p_wait_time<=wait_time+1;
    end if;
```

# PAUSA JUEGO

```
when DELAY_GAME =>
    if(time_lapse=15000000) then
        if(time_lapse=10000) then
            p_time_lapse<=(others => '0');
            p_obj<=vacio;
            estado_Nuevo<=LEE_ORIENT_CBZA;
    else
        p_time_lapse<=time_lapse+1;
    end if;
```



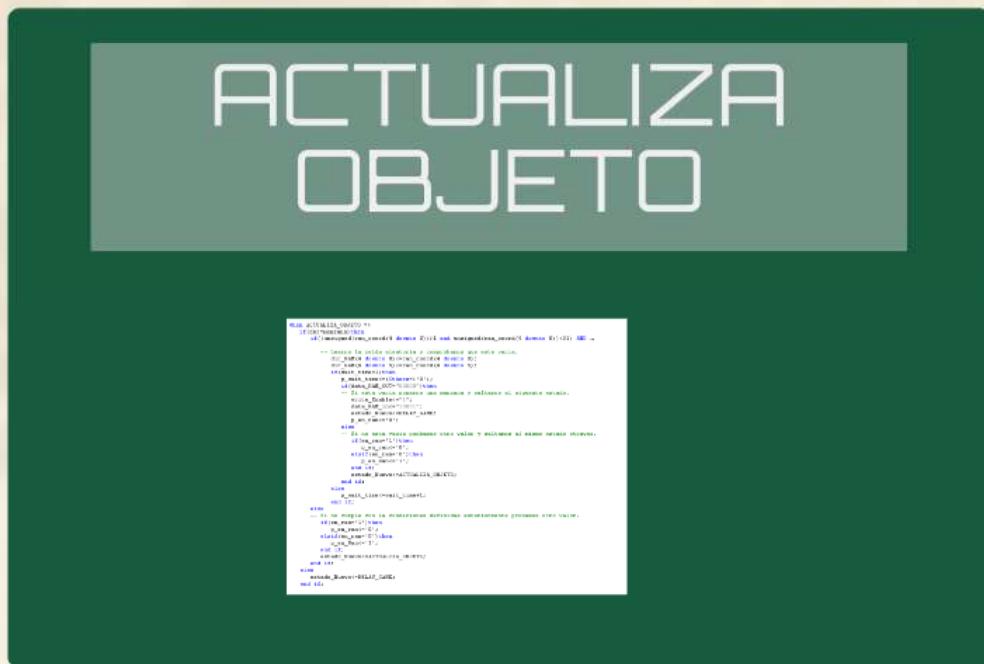
# BORRA LA COLA ANTIGUA

```
when BORRA_COLA_ANTIGUA =>
-- Con la variable de posicion de cola anterior podemos sobre escribir la cola con vacio.
    dir_RAM(4 downto 0)<=STD_LOGIC_VECTOR(x_pos_ant_col);
    dir_RAM(9 downto 5)<=STD_LOGIC_VECTOR(y_pos_ant_col);
    data_RAM_in<="00000"; -- Vacio
    write_Enable<="1";
    if(wait time=1)then
        estado Nuevo<=DELAY GAME;
        p_wait_time<=(others => '0');
    else
        p_wait_time<=wait_time+1;
    end if;
```

# LEE ORIENTACION DE LA COLA

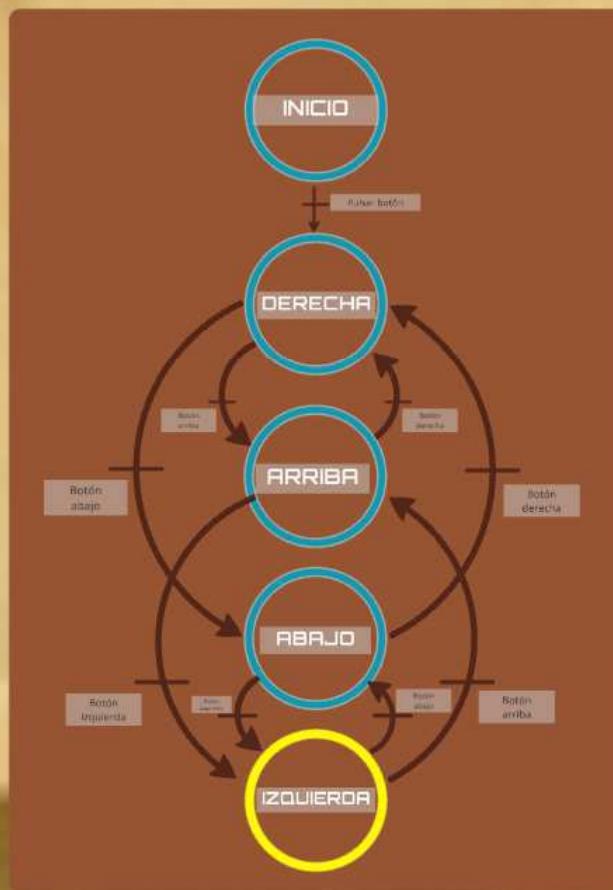
```
when LEE_ORIENT_COLA =>
    if(obj=manzana)then -- Si el objeto fue una manzana no hace falta borrar la cola
        -- asi hacemos que crezca la cabeza.
        estado_Nuevo<=ACTUALIZA_OBJETO;
    else
        -- Leemos la posicion de la cola
        dir_RAM(4 downto 0)<=STD_LOGIC_VECTOR(x_pos_col);
        dir_RAM(9 downto 5)<=STD_LOGIC_VECTOR(y_pos_col);

        -- Tiempo de espera para carcarlo desde la ram
        if(wait_time=1)then
            if(data_RAM_out="01101")then --ARRIBA
                p_dir_col<="01";
            elsif(data_RAM_out="01110")then --ABAJO
                p_dir_col<="11";
            elsif(data_RAM_out="01111")then --DERECHA
                p_dir_col<="10";
            end if;
        end if;
```

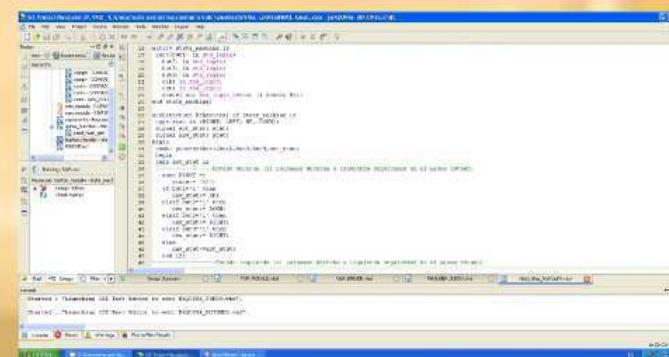


# ACTUALIZA OBJETO

## Controlador de botones



Para la máquina de estados de la botonera, se definió un vector de dos bits que recogía los 4 posibles estados de orientación para la cabeza. Estos estados fueron codificados de manera que al pulsar una posición a la que no se pudiera acceder, se siguiese en el estado anterior.



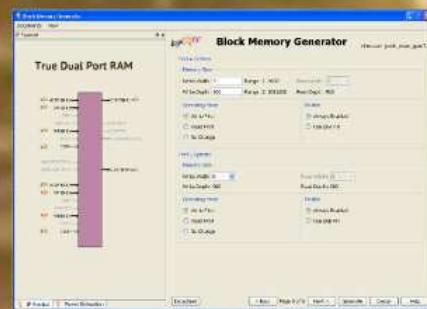
An aerial photograph of a dense urban area, likely Barcelona, showing a mix of modern and traditional architecture. The image is framed by several birds in flight, creating a sense of movement. The colors are warm, with a golden-yellow tint.

# RAM - TABLERO DE JUEGO

**Consta de 30 filas y 32 columnas.**

**En la memoria RAM se incluye la posición por defecto de los elementos codificados.**

empty -	00000
head up -	00001
head down -	00010
head right -	00011
head left -	00100
body up -	00101
body down -	00110
body right -	00111
body left -	01000
bend upl -	01001
bend downl -	01010
bend upr -	01011
bend downr -	01100
tail up -	01101
tail down -	01110
tail right -	01111
tail left -	10000
apple -	10001
LSD -	10010
twix -	10011
poison -	10100
wall -	10101



A screenshot of Microsoft Excel showing a 30x32 grid of memory values. The rows are labeled from 0 to 29 on the left, and the columns are labeled from A1 to AB on the top. The grid contains binary values representing the memory contents. Some cells are highlighted in different colors (yellow, green, red) to indicate specific data points or errors. The Excel ribbon at the top includes tabs for Archivo, Insertar, Dibujo de página, Formato, Datos, Recor, and Vista.

# MEMORIA ROM

La memoria ROM incluye las imágenes cuyo código fue generado por el programa.



ROM.coe: Bloc de notas

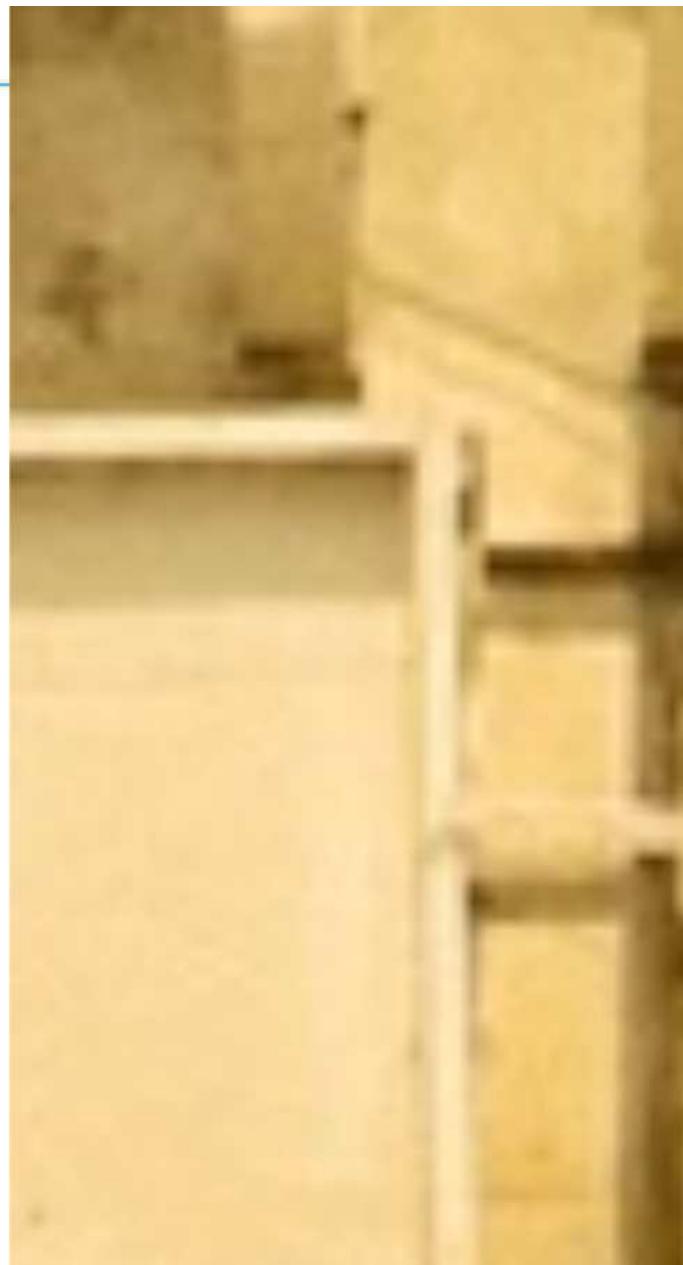
Archivo Edición Formato Ver Ayuda

```
;VACIO
;CABEZA ABAJO ARRIBA DERECHA IZQUIERDA
;CUERPO ABAJO ARRIBA DERECHA IZQUIERDA
;COLA ABAJO ARRIBA DERECHA IZQUIERDA
;ESQUINA DERABAJO DERARRIBA IZQABAJO IZQARRIBA
;LSD
;MANZANA
;MANZANA PODRIDA
;TWIX
;MURO
memory_initialization_radix=2;
memory_initialization_vector=

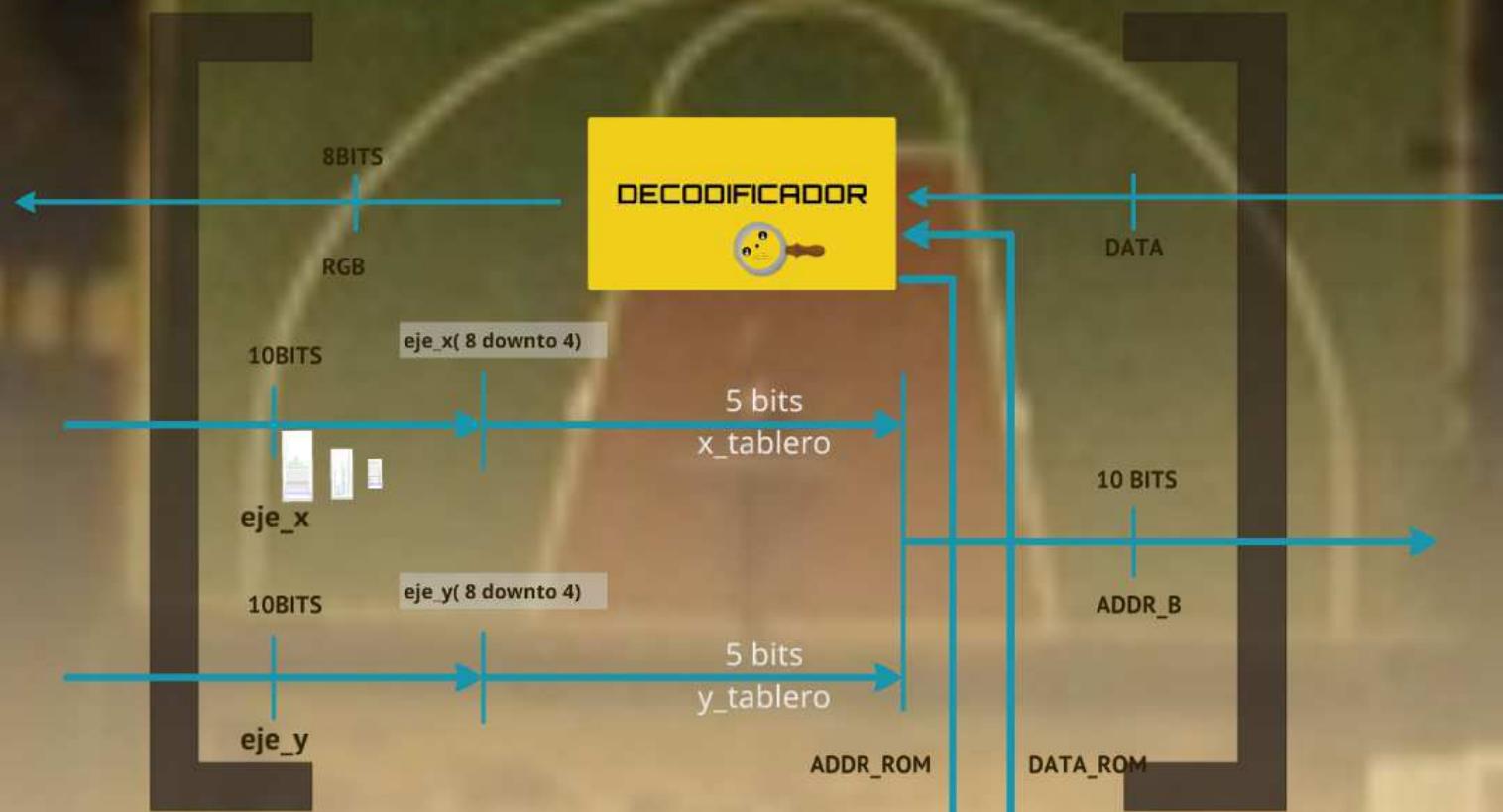
;VACIO

000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
000,
```

Cabe mencionar que aunque fueron implementadas no todas han sido usadas debido a limitaciones posteriores.



# REPRESENTA

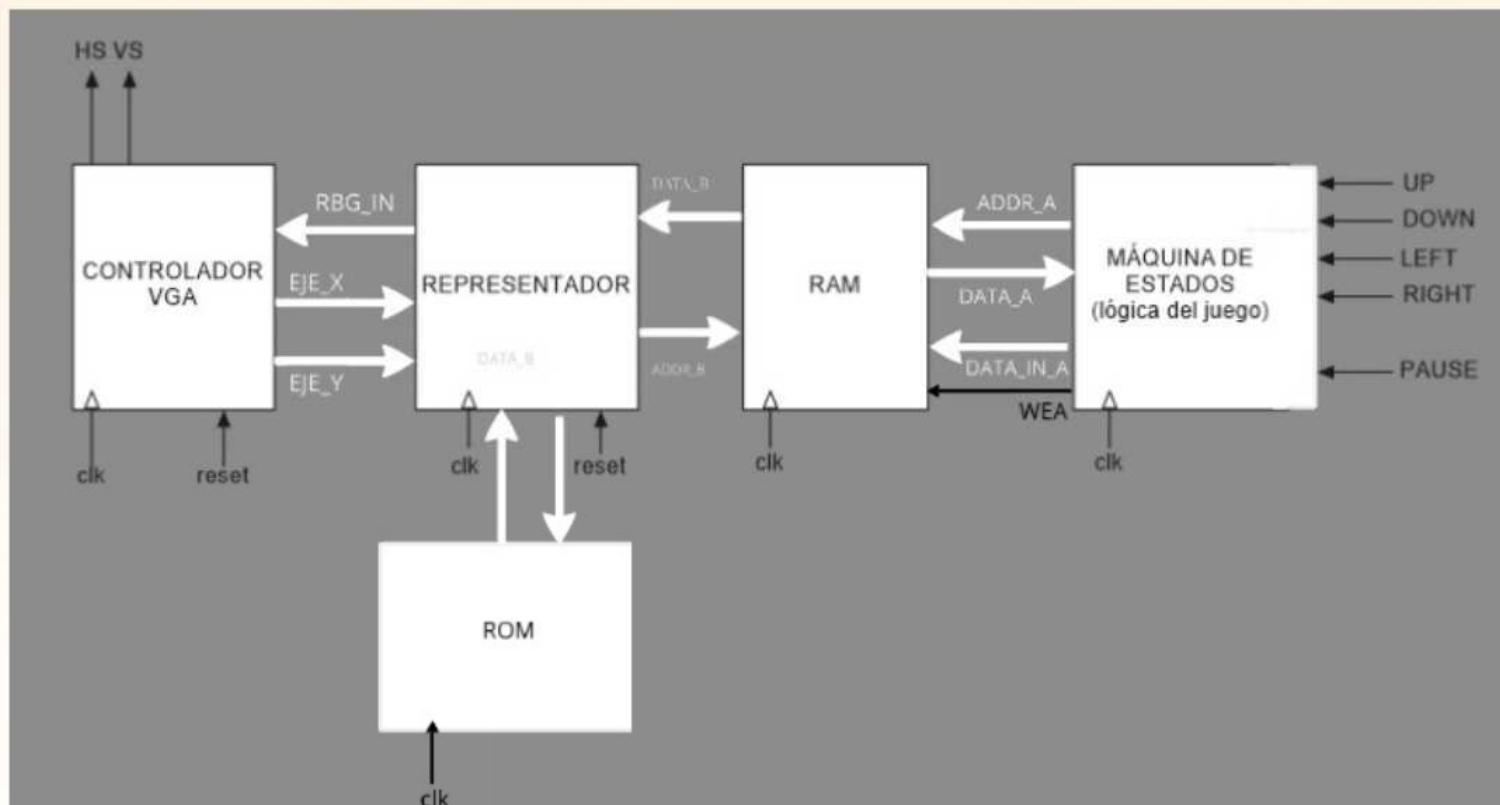


```
29 decodificador_rom: process(Data_RAM)
30 begin
31     CASE Data_RAM is
32         WHEN "00000"=> dir_imagen <= "00000000000000";--vacío
33         WHEN "00001"=> dir_imagen <= "000100000000";--cabeza arriba
34         WHEN "00010"=> dir_imagen <= "000010000000";--cabeza abajo
35         WHEN "00011"=> dir_imagen <= "000110000000";--cabeza derecha
36         WHEN "00100"=> dir_imagen <= "001000000000";--cabeza izquierda
37         WHEN "00101"=> dir_imagen <= "001100000000";--cuerpo arriba
38         WHEN "00110"=> dir_imagen <= "001010000000";--cuerpo abajo
39         WHEN "00111"=> dir_imagen <= "001110000000";--cuerpo derecha
40         WHEN "01000"=> dir_imagen <= "010000000000";--cuerpo izquierda
41         WHEN "01001"=> dir_imagen <= "100000000000";--esquina arriba izquierda
42         WHEN "01010"=> dir_imagen <= "011110000000";--esquina abajo izquierda
43         WHEN "01011"=> dir_imagen <= "011100000000";--esquina arriba derecha
44         WHEN "01100"=> dir_imagen <= "011010000000";--esquina arriba izquierda
45         WHEN "01101"=> dir_imagen <= "010100000000";--cola arriba
46         WHEN "01110"=> dir_imagen <= "010010000000";--cola abajo
47         WHEN "01111"=> dir_imagen <= "010110000000";--cola derecha
48         WHEN "10000"=> dir_imagen <= "011000000000";--cola izquierda
49         WHEN "10001"=> dir_imagen <= "100100000000";--manzana
50         WHEN "10010"=> dir_imagen <= "100010000000";--LSD
51         WHEN "10011"=> dir_imagen <= "101000000000";--twix
52         WHEN "10100"=> dir_imagen <= "100110000000";--veneno
53         WHEN "10101"=> dir_imagen <= "101010000000";--muro
54         WHEN OTHERS=> dir_imagen <= "00000000000000";
55     END CASE;
56 end process;
```

```
57
58 dibuja: process(eje_x, eje_y,Data_ROM,LSD,y_tablero,x_tablero,Addr_ROM_aux,dir_imagen)
59 begin
60
61     -- Si resulta que los pixeles que se estan pintando estan fuera del tablero
62     if (((eje_x>=0 and eje_x<64) or (eje_x>575)) or (eje_y>479)) then
63         RGB<="00000000";
64         y_tablero <= (others => '0');
65         x_tablero <= (others => '0');
66         x_tablero_aux <=(others => '0');
67         Addr_ROM_aux <=(others => '0');
68         Addr_RAM<=(others => '0');
69     else
70         -- Dividimos por 16 eje_x y eje_y para obtener las coordenada en el tablero
71         x_tablero_aux <= unsigned(eje_x)-"0001000000"; -- Le restamos 64 para centrar el tablero
72         y_tablero(9 downto 5)<=(Others => '0'); -- Lo demas a cero
73         y_tablero(4 downto 0) <= unsigned(eje_y(8 downto 4)); -- Dividimos por 16
74         x_tablero(9 downto 5)<=(Others => '0'); -- Lo demas a cero
75         x_tablero(4 downto 0)<= x_tablero_aux(8 downto 4); -- Dividimos por 16
76
77         -- En el pimrer termino tenemos la direccion de la imagen que se tiene que pintar
78         -- En el segundo termino tenemos la fila de esa imagen que se tiene que pintar
79         -- En el ultimo termino tenemos la columna de la imagen que se tiene que pintar
80         Addr_ROM_aux <= dir_imagen +((unsigned(eje_y)-(y_tablero sll 4)) sll 4) + (unsigned(eje_x)-64-(x_tablero sll 4)) -1;
81
82         -- Le asignamos a la direccion de la ram las coordenadas del tablero
83         -- para obtener informacion de lo que hay en dicha celda
84
```

```
CASE Data_ROM is
    WHEN "000"=> RGB <= "00000000"; --negro
    WHEN "111"=> RGB <= "11111111"; --blanco
    WHEN "100"=> RGB <= "11100000"; --rojo
    WHEN "110"=> RGB <= "11111100"; --amarillo
    WHEN "010"=> RGB <= "00011100"; --verde
    WHEN "101"=> RGB <= "11100011"; --lila
    WHEN "011"=> RGB <= "00011111"; --cian
    WHEN "001"=> RGB <= "00000011"; --azul
    WHEN OTHERS=> RGB <= "00000000";
END CASE;
```

# DIAGRAMA PRINCIPAL DE LA LÓGICA DEL JUEGO



# WARNINGS

```
[1] WARNING:Xst:2211 - "///vboxsrv/c_drive/Users/Richa/Xilinx_stuff/SNAKE GAME/TOP_MODULE.vhd" line 141: Instantiating black box module <ROM>.  
[1] WARNING:Xst:2211 - "///vboxsrv/c_drive/Users/Richa/Xilinx_stuff/SNAKE GAME/TOP_MODULE.vhd" line 142: Instantiating black box module <RAM>.  
[1] WARNING:Xst:646 - Signal <y_tablero<9:6>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.  
[1] WARNING:Xst:646 - Signal <x_tablero_aux<9>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.  
[1] WARNING:Xst:646 - Signal <x_tablero_aux<3:0>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.  
[1] WARNING:Xst:646 - Signal <x_tablero<9:6>> is assigned but never used. This unconnected signal will be trimmed during the optimization process.  
[1] WARNING:Xst:2677 - Node <obj_3> of sequential type is unconnected in block <MAQUINA_JUEGO>.  
[1] WARNING:Xst:2677 - Node <obj_1> of sequential type is unconnected in block <MAQUINA_JUEGO>.  
[1] WARNING:Xst:2677 - Node <obj_0> of sequential type is unconnected in block <MAQUINA_JUEGO>.  
[1] WARNING:Xst:2677 - Node <side_1> of sequential type is unconnected in block <MAQUINA_JUEGO>.  
[1] WARNING:Xst:2677 - Node <game_machine/obj_2> of sequential type is unconnected in block <TOP_MODULE>.
```

Tenemos dos warnings que nos indican que estamos instanciando modulos de ram y de rom.

Luego tenemos otros warnings que nos indican que tenemos algunos bits de algunos vectores sin conectar. En las variables de x tablero hemos tenido que declararlo como 10 bits ya que al tenerlo como 5 nos daba resultados no deseados al hacerle la rotación de bits para multiplicarlo por 16, declarando como 10 bits hemos solucionado este problema y los bits que sobran ya se recortan automáticamente por el sintetizador. En las variables de objeto nos salta el warning porque efectivamente no lo estamos usando ya que no hemos implementado todos los objetos que teníamos planeado.

# LIMITACIONES Y BUGS.

