UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tarea 3

Control y Visualización en una pantalla

Sistemas Digitales Programables

Máster Universitario en Ingeniería de Sistemas Electrónicos

2020/2021

Irene Garcia do Amaral

Generación de las señales de sincronismo y datos de la pantalla LCD

Implementación en Verilog:

```
module lcd_sync
(
    input CLK, RST_n,
    output NCLK, GREST, HD, VD, DEN,
    output [9:0] fila,
    output [10:0] coluna
);

wire wire_h, wire_v;

pll_ltm PLL
(
    CLK , // input  inclk0_sig
    NCLK  // output  c0_sig
);

contador HCOUNT
(
    .iCLK(NCLK) ,  // input  iCLK_sig
    .iRST_n(RST_n) ,  // input  iRST_n_sig
    .iENABLE(1'b1) ,  // input  iENABLE_sig
    .iUP_DOWN(1'b1) , // input  iUP_DOWN_sig
    .oCOUNT(coluna) , // output [n-1:0] oCOUNT_sig
    .oTC(wire_h)   // output  oTC_sig
);

defparam HCOUNT.fin_cuenta = 1055;


contador VCOUNT
(
    .iCLK(NCLK) ,  // input  iCLK_sig
    .iRST_n(RST_n) ,  // input  iRST_n_sig
    .iENABLE(wire_h) ,   // input  iENABLE_sig
    .iUP_DOWN(1'b1) , // input  iUP_DOWN_sig
    .oCOUNT(fila) ,   // output [n-1:0] oCOUNT_sig
    .oTC(wire_v)   // output  oTC_sig
);

defparam VCOUNT.fin_cuenta = 525;

assign HD =! wire_h;
assign VD =! wire_v;

assign GREST = RST_n;

assign DEN = ((coluna > 215 && coluna < 1016) && (fila > 34 && fila < 515)) ? 1:0;

endmodule
```

RTL Viewer:

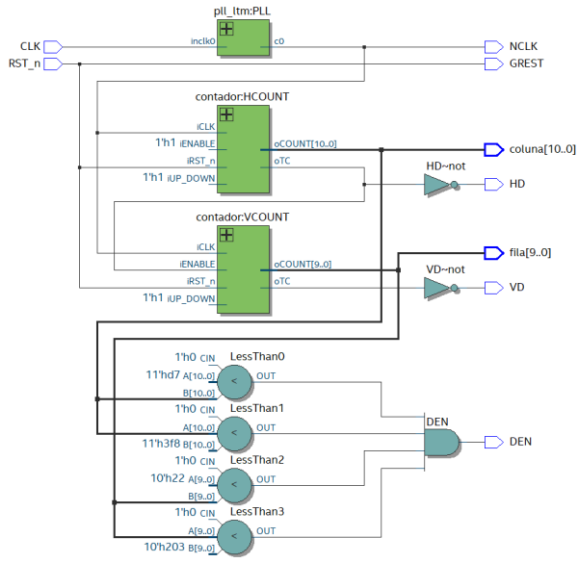

Figura 1: RTL Viewer de lcd_sync

Implementación del Testbench:

```
`timescale 1ns/100ps

module tb_lcd_sync01();

localparam T = 20;

reg CLK, RST_n;

wire NCLK, HD, VD, GREST, DEN;
wire [10:0] coluna;
wire [9:0] fila;

lcd_sync DUT
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .fila(fila) ,       // output [9:0] fila_sig
    .coluna(coluna)     // output [10:0] coluna_sig
);

initial
begin
    CLK = 1'b0;
    RST_n = 1'b1;

    #(T*10)
    RST_n = 1'b0;
    #(T*3)
    RST_n = 1'b1;

    repeat (2*1055*525) @(posedge CLK);
    #(T*5);

    $stop;
end

always
    #(T/2) CLK <= ~CLK;

endmodule
```

RTL Simulation:

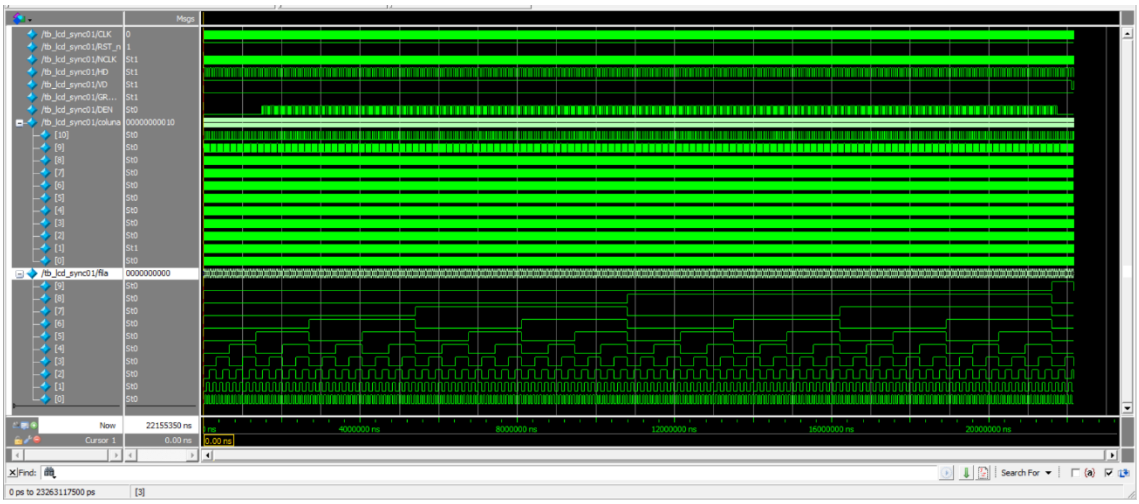

Figura 2: RTL Simulation de tb_lcd_sync01

Generación de barras de colores en la pantalla

Implementación en Verilog:

```
module barras_lcd
(
    input CLK,
    input RST_n,

    output NCLK,
    output GREST,
    output HD,
    output VD,
    output DEN,

    output reg [7:0] R,
    output reg [7:0] G,
    output reg [7:0] B
);

wire [10:0] coluna;
wire [9:0] fila;

localparam integer larg = 100; //(1015-215)/8      ---> NAO SERVE PARA NADA

lcd_sync lcd_sync_inst
(
    .CLK(CLK) ,           // input  CLK_sig
    .RST_n(RST_n) ,       // input  RST_n_sig
    .NCLK(NCLK) ,         // output  NCLK_sig
    .GREST(GREST) ,       // output  GREST_sig
    .HD(HD) ,             // output  HD_sig
    .VD(VD) ,             // output  VD_sig
    .DEN(DEN) ,           // output  DEN_sig
    .fila(fila) ,         // output [9:0] fila_sig
    .coluna(coluna)       // output [10:0] coluna_sig
);

always @(coluna)
begin

    if (coluna > 215 && coluna < 315)
    begin
        R = 8'b11111111;
        G = 8'b11111111;
        B = 8'b11111111;
    end

    else if (coluna > 315 && coluna < 415)
    begin
        R = 8'b11111111;
        G = 8'b11111111;
        B = 8'b00000000;
    end

    else if (coluna > 415 && coluna < 515)
    begin
        R = 8'b00000000;
        G = 8'b11111111;
        B = 8'b11111111;
    end

    else if (coluna > 515 && coluna < 615)
    begin
        R = 8'b00000000;
        G = 8'b11111111;
        B = 8'b00000000;
    end

    else if (coluna > 615 && coluna < 715)
    begin
        R = 8'b11111111;
        G = 8'b00000000;
        B = 8'b11111111;
    end

    else if (coluna > 715 && coluna < 815)
    begin
        R = 8'b11111111;
        G = 8'b00000000;
        B = 8'b00000000;
    end

    else if (coluna > 815 && coluna < 915)
    begin
        R = 8'b00000000;
        G = 8'b00000000;
        B = 8'b11111111;
    end

    else if (coluna > 915 && coluna < 1015)
    begin
        R = 8'b00000000;
        G = 8'b00000000;
        B = 8'b00000000;
    end
end

endmodule
```

Implementación del Testbench:

```
`timescale 1ns/100ps

module tb_barras_lcd();

localparam T = 20;

reg CLK;
reg RST_n;

wire NCLK;
wire GREST;
wire HD;
wire VD;
wire DEN;
wire [7:0] R;
wire [7:0] G;
wire [7:0] B;

integer fd;
event cierraFichero;

barras_lcd barras_lcd_inst
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .R(R) ,             // output [7:0] R_sig
    .G(G) ,             // output [7:0] G_sig
    .B(B)               // output [7:0] B_sig
);

always
    #(T/2) CLK <= ~CLK;


initial
begin
    $display("INICIO SIMULACION!\n");

    CLK = 1'b0;
    RST_n = 1'b1;

    #(T*10)
    RST_n = 1'b0;
    #(T*3)
    RST_n = 1'b1;
```

```
    @(posedge VD);
    #(T*5);
    $display("FIN SIMULACION\n");
    ->cierraFichero;
    #(T);
    $stop;
end

initial
begin
    fd = $fopen("vga.txt", "w");
    @(cierraFichero);
    disable guardaFichero;
    $display("Cierro Fichero");
    $fclose(fd);
end

initial forever begin: guardaFichero
    @(posedge NCLK)
    $fwrite(fd, "%0t ps: %b %b %b %b %b %b \n", $time, HD, VD, DEN, R, G, B);
end


endmodule
```

RTL Simulation:

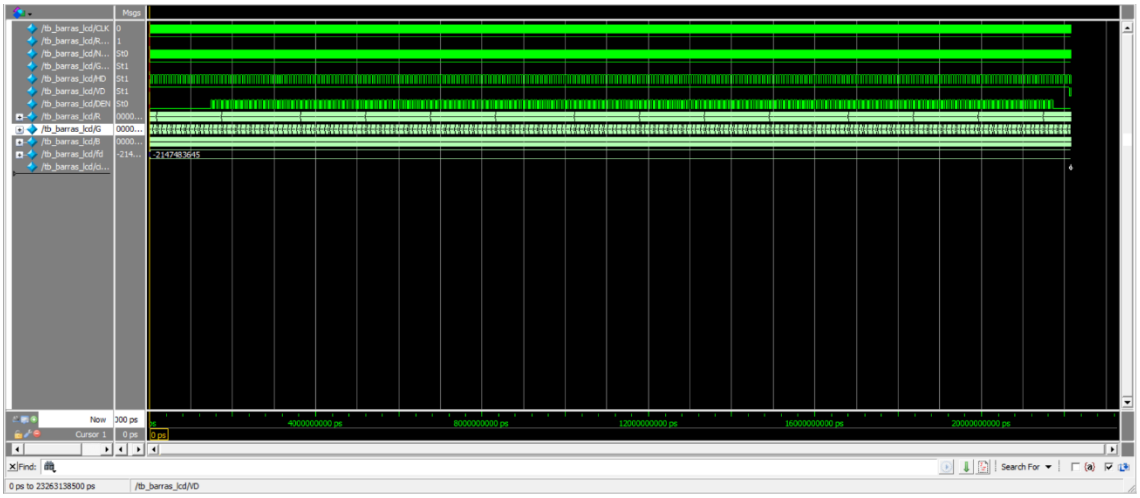

Figura 3: RTL Simulation de tb_barras_lcd

Screen Simulator:

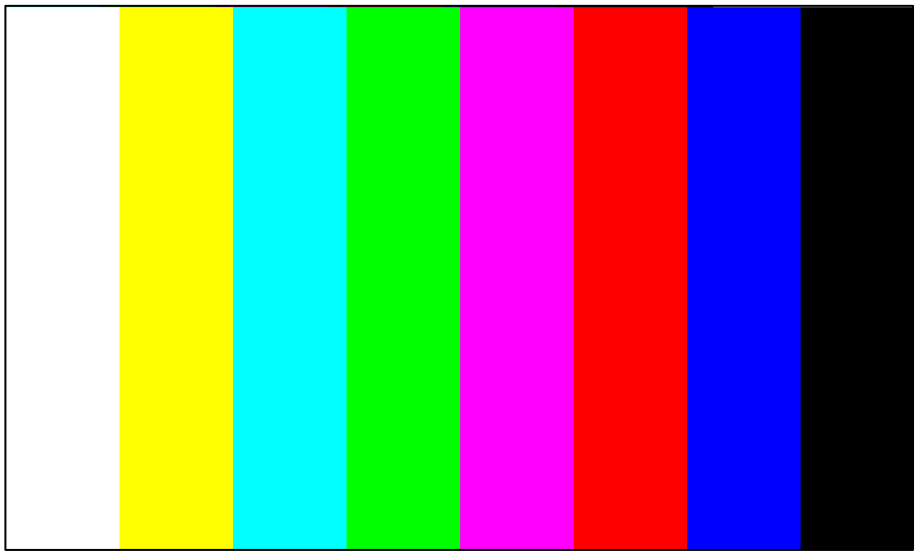

Figura 4: Screen Simulator del fichero txt generado en el testbench tb_barras_lcd

Figura 5: Visualización de las barras de colores en la pantalla LTM

Visualización de una imagen en la pantalla

Implementación en Verilog:

```
module imagen_lcd
(
    input CLK,
    input RST_n,

    output NCLK,
    output GREST,
    output HD,
    output VD,
    output DEN,
    output [7:0] R,
    output [7:0] G,
    output [7:0] B
);

wire [10:0] coluna;
wire [9:0] fila;

wire [16:0] address;
wire [15:0] data;

wire [7:0] filax2;
wire [8:0] colunax2;

lcd_sync lcd_sync_inst
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .fila(fila) ,       // output [9:0] fila_sig
    .coluna(coluna)     // output [10:0] coluna_sig
);

assign filax2 = (fila-35) >> 1;         //fila/coluna - backporch tirando 1 bit para fazer zoom
assign colunax2 = (coluna-215) >> 1;
assign address = {filax2, colunax2};


ROM_image   ROM_image_inst
(
    .address (address),
    .clock (NCLK),
    .q (data)
);

assign R = {data[15:11], data[15:13]};
assign G = {data[10:5], data[10:9]};
assign B = {data[4:0], data[4:2]};

endmodule
```

Implementación del Testbench:

Nota: No ha sido posible generar el documento txt para el *Screen Simulator* porque en el *RTL Simulation* los valores de R, G y B permanecen a zero.

```
`timescale 1ns/100ps

module tb_img();

localparam T = 20;

reg CLK;
reg RST_n;

wire NCLK;
wire GREST;
wire HD;
wire VD;
wire DEN;
wire [7:0] R;
wire [7:0] G;
wire [7:0] B;

integer fd;
event cierraFichero;

imagen_lcd imagen_lcd_inst
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .R(R) ,             // output [7:0] R_sig
    .G(G) ,             // output [7:0] G_sig
    .B(B)               // output [7:0] B_sig
);

always
    #(T/2) CLK <= ~CLK;


initial
begin
    $display("INICIO SIMULACION!\n");

    CLK = 1'b0;
    RST_n = 1'b1;

    #(T*10)
    RST_n = 1'b0;
    #(T*3)
    RST_n = 1'b1;
```

```
    @(posedge VD);
    #(T*5);
    $display("FIN SIMULACION\n");
    ->cierraFichero;
    #(T);
    $stop;
end

initial
begin
    fd = $fopen("img.txt", "w");
    @(cierraFichero);
    disable guardaFichero;
    $display("Cierro Fichero");
    $fclose(fd);
end

initial forever begin: guardaFichero
    @(posedge NCLK)
    $fwrite(fd, "%0t ps: %b %b %b %b %b %b \n", $time, HD, VD, DEN, R, G, B);
end


endmodule
```

Figura 6: Visualización de una imagen en la pantalla LTM

Visualización de caracteres en la pantalla

Implementación en Verilog:

```
module caracteres_lcd
(
    input CLK,
    input RST_n,

    output NCLK,
    output GREST,
    output HD,
    output VD,
    output DEN,
    output reg [7:0] R,
    output reg [7:0] G,
    output reg [7:0] B
);

wire [10:0] coluna;
wire [9:0] fila;

wire [16:0] address;
wire [15:0] data;

wire [7:0] filax2;
wire [8:0] colunax2;

reg [9:0] address_char;
wire [7:0] data_char;
reg pixel_on_off;

lcd_sync lcd_sync_inst
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .fila(fila) ,       // output [9:0] fila_sig
    .coluna(coluna)     // output [10:0] coluna_sig
);

assign filax2 = (fila-35) >> 1;         //fila/coluna - backporch tirando 1 bit para fazer zoom
assign colunax2 = (coluna-215) >> 1;
assign address = {filax2, colunax2};

ROM_image   ROM_image_inst
(
    .address (address),
    .clock (NCLK),
    .q (data)
);
```

```
ROM_char ROM_char_inst
(
    .address(address_char) ,    // input [8:0] address_sig
    .clock(NCLK) ,              // input  clock_sig
    .q(data_char)               // output [7:0] q_sig
);

//mux de dados de saida
always @(data_char, coluna)
begin
    if(coluna > 215 && coluna < 250)
    begin
    address_char = {6'o11, fila[4:2]};
       case (coluna[4:2])                       //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

    if(coluna > 250 && coluna < 285)
    begin
    address_char = {6'o22, fila[4:2]};
       case (coluna[4:2])                       //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end
```

```
       if(coluna > 285 && coluna < 320)
    begin
    address_char = {6'o05, fila[4:2]};
       case (coluna[4:2])                       //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

       if(coluna > 320 && coluna < 355)
    begin
    address_char = {6'o16, fila[4:2]};
       case (coluna[4:2])                       //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

       if(coluna > 355 && coluna < 390)
    begin
    address_char = {6'o05, fila[4:2]};
       case (coluna[4:2])                       //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end
```

```
    if(coluna > 390 && coluna < 425)
begin
    pixel_on_off = 0;
    /*
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
    */
end

    if(coluna > 425 && coluna < 460)
begin
address_char = {6'o07, fila[4:2]};
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
end

    if(coluna > 460 && coluna < 495)
begin
address_char = {6'o01, fila[4:2]};
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
end
```

```
    if(coluna > 495 && coluna < 530)
begin
address_char = {6'o22, fila[4:2]};
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
end

    if(coluna > 530 && coluna < 565)
begin
address_char = {6'o03, fila[4:2]};
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
end

        if(coluna > 565 && coluna < 600)
begin
address_char = {6'o11, fila[4:2]};
    case (coluna[4:2])                      //0:2 normal; 4:2 ampliado
        0: pixel_on_off = data_char[7];
        1: pixel_on_off = data_char[6];
        2: pixel_on_off = data_char[5];
        3: pixel_on_off = data_char[4];
        4: pixel_on_off = data_char[3];
        5: pixel_on_off = data_char[2];
        6: pixel_on_off = data_char[1];
        7: pixel_on_off = data_char[0];
        default: pixel_on_off = data_char[0];
    endcase
end
```

```
        if(coluna > 600 && coluna < 635)
    begin
    address_char = {6'o01, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

        if(coluna > 635 && coluna < 670)
    begin
       pixel_on_off = 0;
       /*
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
       */
    end

        if(coluna > 670 && coluna < 705)
    begin
    address_char = {6'o01, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end
```

```
        if(coluna > 705 && coluna < 740)
    begin
    address_char = {6'o15, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end
        if(coluna > 740 && coluna < 775)
    begin
    address_char = {6'o01, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

        if(coluna > 775 && coluna < 810)
    begin
    address_char = {6'o22, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end
```

```
        if(coluna > 810 && coluna < 845)
    begin
    address_char = {6'o01, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

        if(coluna > 845 && coluna < 880)
    begin
    address_char = {6'o14, fila[4:2]};
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
    end

        if(coluna > 880 && coluna < 915)
    begin
       pixel_on_off = 0;
       /*
       case (coluna[4:2])                    //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
       */
    end
```

```
        if(coluna > 915 && coluna < 950)
    begin
       pixel_on_off = 0;
       /*
       case (coluna[4:2])                     //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
       */
    end

        if(coluna > 950 && coluna < 985)
    begin
       pixel_on_off = 0;
       /*
       case (coluna[4:2])                     //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
       */
    end

        if(coluna > 985 && coluna < 1015)
    begin
       pixel_on_off = 0;
       /*
       case (coluna[4:2])                     //0:2 normal; 4:2 ampliado
          0: pixel_on_off = data_char[7];
          1: pixel_on_off = data_char[6];
          2: pixel_on_off = data_char[5];
          3: pixel_on_off = data_char[4];
          4: pixel_on_off = data_char[3];
          5: pixel_on_off = data_char[2];
          6: pixel_on_off = data_char[1];
          7: pixel_on_off = data_char[0];
          default: pixel_on_off = data_char[0];
       endcase
       */
```

```
    end
end

//selecao de cor
always @(pixel_on_off)
begin
    if (pixel_on_off)
    begin
        R = 8'b10101101;
        G = 8'b11111111;
        B = 8'b00101111;


    end

    else
    begin
        R = {data[15:11], data[15:13]};
        G = {data[10:5], data[10:9]};
        B = {data[4:0], data[4:2]};
    end
end

endmodule
```

Implementación del Testbench:

Nota: No ha sido posible generar el documento txt para el *Screen Simulator* porque en el *RTL Simulation* los valores de R, G y B permanecen a zero.

```
`timescale 1ns/100ps

module tb_char();

localparam T = 20;

reg CLK;
reg RST_n;

wire NCLK;
wire GREST;
wire HD;
wire VD;
wire DEN;
wire [7:0] R;
wire [7:0] G;
wire [7:0] B;

integer fd;
event cierraFichero;

caracteres_lcd caracteres_lcd_inst
(
    .CLK(CLK) ,         // input  CLK_sig
    .RST_n(RST_n) ,     // input  RST_n_sig
    .NCLK(NCLK) ,       // output  NCLK_sig
    .GREST(GREST) ,     // output  GREST_sig
    .HD(HD) ,           // output  HD_sig
    .VD(VD) ,           // output  VD_sig
    .DEN(DEN) ,         // output  DEN_sig
    .R(R) ,             // output [7:0] R_sig
    .G(G) ,             // output [7:0] G_sig
    .B(B)               // output [7:0] B_sig
);


always
    #(T/2) CLK <= ~CLK;


initial
begin
    $display("INICIO SIMULACION!\n");

    CLK = 1'b0;
    RST_n = 1'b1;

    #(T*10)
    RST_n = 1'b0;
    #(T*3)
    RST_n = 1'b1;
```

```
    @(posedge VD);
    #(T*5);
    $display("FIN SIMULACION\n");
    ->cierraFichero;
    #(T);
    $stop;
end

initial
begin
    fd = $fopen("char.txt", "w");
    @(cierraFichero);
    disable guardaFichero;
    $display("Cierro Fichero");
    $fclose(fd);
end

initial forever begin: guardaFichero
    @(posedge CLK)
    $fwrite(fd, "%0t ns: %b %b %b %b %b %b \n", $time, HD, VD, DEN, R, G, B);
end

endmodule
```

Figura 7: Visualización de una imagen y caracteres en la pantalla LTM