Programa de Ingeniería en Robótica y Mecatrónica

Microcontroladores

Remberto Sandoval Aréchiga

Reporte de practica de controlador de display 7 segmentos

Integrantes del equipo Los Panas	Matricula
Karla Lorena Loera Benitez	35165366
Jonathan Adrian Garcia Guerrero	35164932
Allan Uriel Valencia Esparza	35164771

Resumen

Nuestro proyecto es realizar un display de 7 segmentos basándonos en la ayuda de cada uno de nuestros compañeros. Cada uno de los equipos contribuyo con una parte del código ya sea contadores, el preescalador y la estructura base.
Por lo que en este reporte podremos observar las diversas herramientas y códigos que utilizamos, así como las simulaciones correspondientes, una pequeña instrucción de lo que es un display de 7 segmentos y nuestras conclusiones de lo realizado.

Índice

Introducción

Para poder comprender un poco mejor lo que se realizó daremos una pequeña introducción a el display de 7 segmentos con algunas de sus características principales. Después de analizar esto podremos dar inicio a como realizamos cada una de las partes de nuestro display de 7 segmentos.

¿Qué es un display de 7 segmentos?

El display de siete segmentos es una forma de representar caracteres en equipos electrónicos. Está compuesto de siete segmentos que se pueden encender o apagar individualmente. Cada segmento tiene la forma de una pequeña línea y permite visualizar números del 0 al 9 y algunos caracteres. Existen dos tipos de display, de cátodo común y de ánodo común.

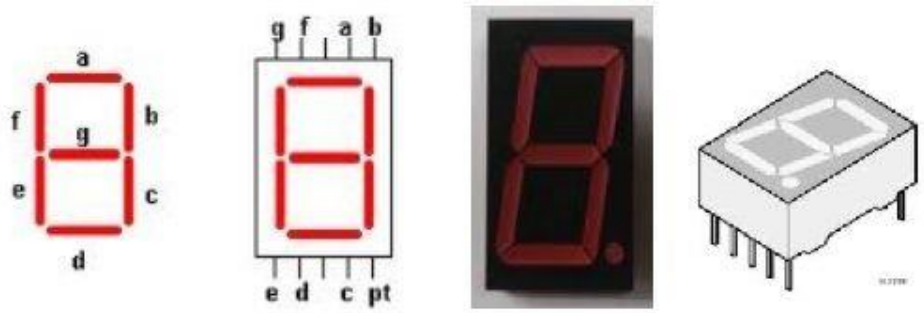

Imagen 1 Display 7 segmentos

Ánodo común

Se llama así porque todos los leds están unidos en su terminal positiva (ánodo), para encenderlos tenemos que poner tierra en la terminal de la letra que se desee.

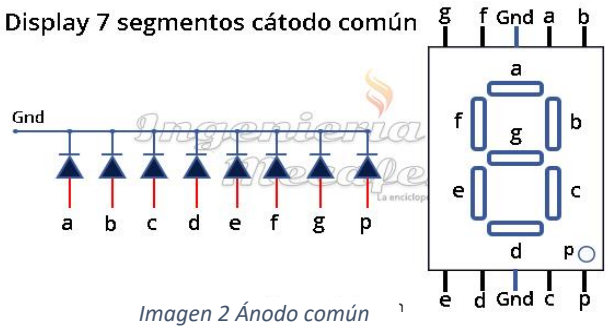

Imagen 2 Ánodo común

Cátodo común

Este display es el opuesto del ánodo común ya que los leds están unidos en la terminal negativa (cátodo). Para encender los leds tenemos que poner voltaje en las terminales de las letras.

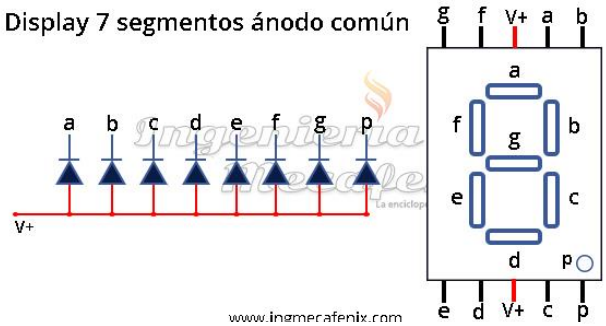

Imagen 3 Cátodo común

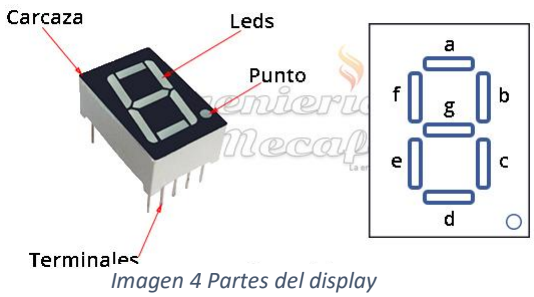

Imagen 4 Partes del display

Partes de un display de 7 segmentos

Se llama así porque todos los leds están unidos en su terminal Se le conoce como 7 segmentos por que cuenta con siete diodos led principales y uno extra para representar un punto. También cuenta con una carcasa para cubrirlos y 10 terminales: 2 son de alimentación (2 de Vcd o 2 de Gnd), 1 es para visualizar un punto y 7 son para representar cada uno de los números según la combinación que se le ponga, estos están representados por una letra del abecedario desde la “A” hasta la letra “G”.

Requerimientos

Los requerimientos para este proyecto son; crear un controlador de display de 7 segmentos el cual tendrá que poder exponer 4 dígitos haciendo uso del hexadecimal (lo cual se traduce a un sistema de numeración que va del 0 al 15), que internamente se podría escribir como 0x0 a 0xF, asimismo, para mostrar estos dígitos en el display propiamente dicho, se hará uso de un preescalador que funcionara a 120 Hz, que es lo ideal para que el ojo humano no aprecie el parpadeo del display.
Los dígitos que deberá mostrar el display propiamente dicho serán otorgados por medio de 4 entradas de 4 bits cada uno, que serán seleccionados por un selector de anillo, pasando por un demultiplexor, y, finalmente transitara por un codificador, dando señales a los segmentos y del contador de anillo hacia los ánodos del display.

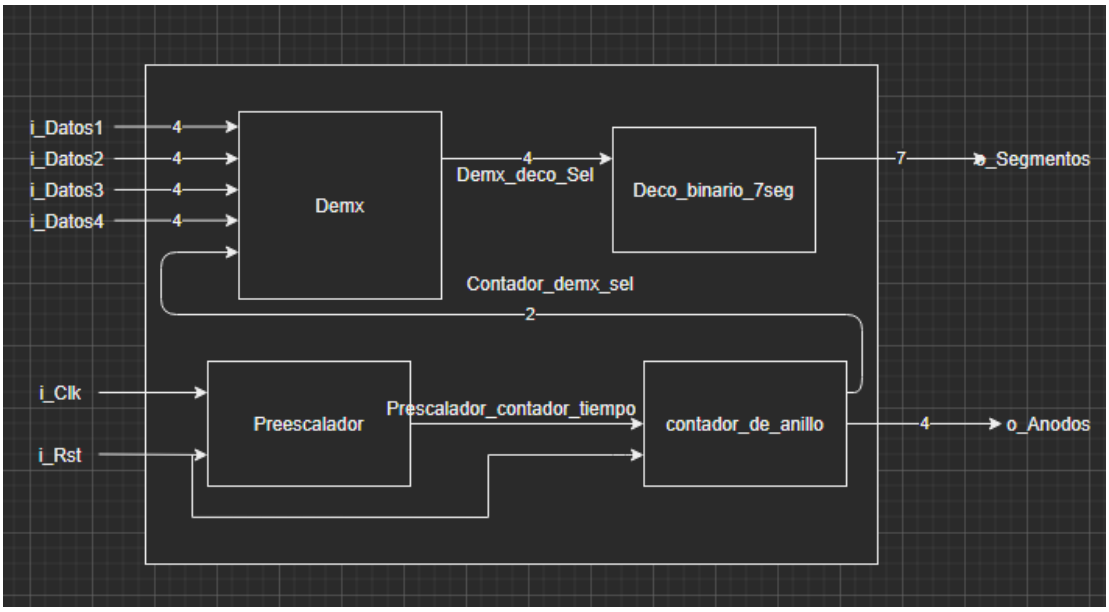

Imagen 5 Imagen del diagrama de caja negra en el cual se especifican los requerimientos necesarios para que el controlador sea posible.

Arquitectura

La siguiente imagen muestra nuestra arquitectura que comienza con nuestro diagrama de caja negra el cual incluye todos los diversos componentes que vamos a necesitar para crear nuestro display.

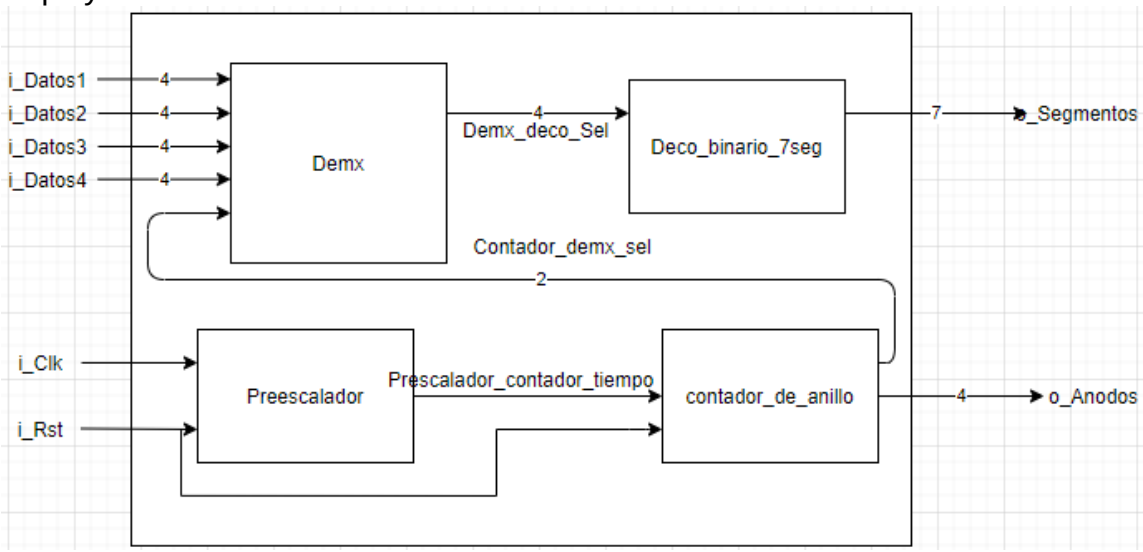

Imagen 6 Caja negra

De manera individual en cada uno de ellos hicimos un código por separado basándonos en su arquitectura comenzando por el preescalador.

Preescalador

- Modifica la frecuencia de entrada de la señal i_Clk de los 100MHz a 120Hz en la salida mediante un conteo de pulsos cada flanco positivo lo que permite operar a distintas frecuencias.
- Cada 833334 ciclos de reloj provenientes de la señal i_Clk se producirá un ciclo de reloj en la salida o_Presc.
- Mantiene la salida o_Presc en 1 la mitad del tiempo y la otra mitad en 0. Cuando el contador de pulsos alcanza un valor igual a 416667 el valor de la señal de salida pasa de 1 a 0. Cuando el contador de pulsos alcanza un valor igual a 833334 el valor de la señal de salida pasa de 0 a 1 y el conteo se reinicia desde 0.
- Utiliza la señal o_Presc como habilitación del contador de anillo, cuando esta señal de salida tiene un valor de 1.

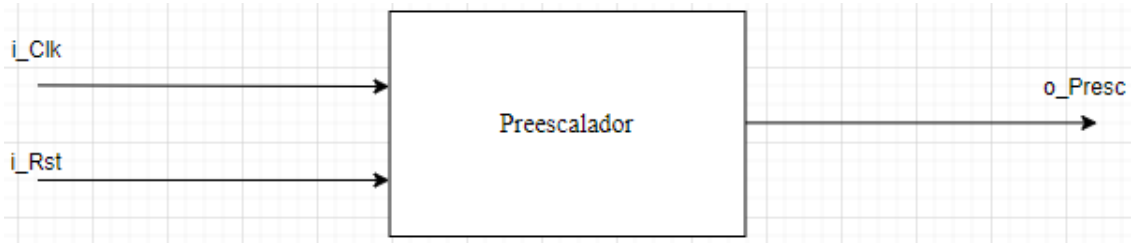

Imagen 7 Diagrama del preescalador

Contador de anillo

- Seleccionar el display de 4 segmentos en el que se representará cada dato a través de la señal de o_Anodo, a partir de una señal de reloj de 120 Hz.
- En el contador de anillo existe un bit que se desplaza entre flip flops usando la señal del preescalador que lo hará cambiar de posición, esto se traduce como altos y bajos en los

ánodos los cuales son ceros y unos lógicos para el selector como se observa en la siguiente imagen, esto para la salida o_Anodos.

- La salida selector está conformada por Q3, Q2, Q1 y Q0, los cuales envían la posición del bit en el anillo contador, con el cual el demultiplexor toma como referencia para seleccionar la entrada de datos.

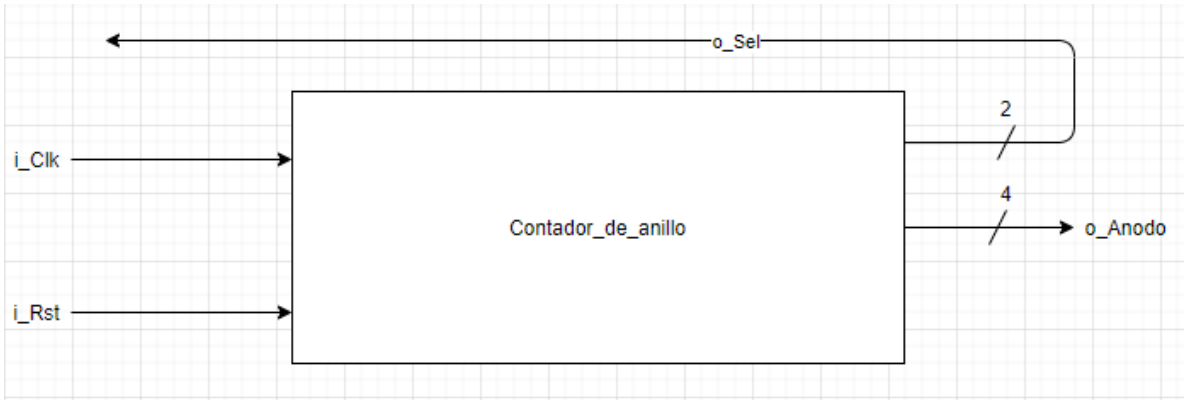

Imagen 8 Diagrama de contador de anillo

Multiplexor

- La entrada i_Sel indica qué entrada será correspondiente a la salida.

Este comportamiento se describe a continuación:

i_Sel = "00" [La salida corresponde a i_Datos0]
i_Sel = "01" [La salida corresponde a i_Datos1]
i_Sel = "10" [La salida corresponde a i_Datos2]
i_Sel = "11" [La salida corresponde a i_Datos3]

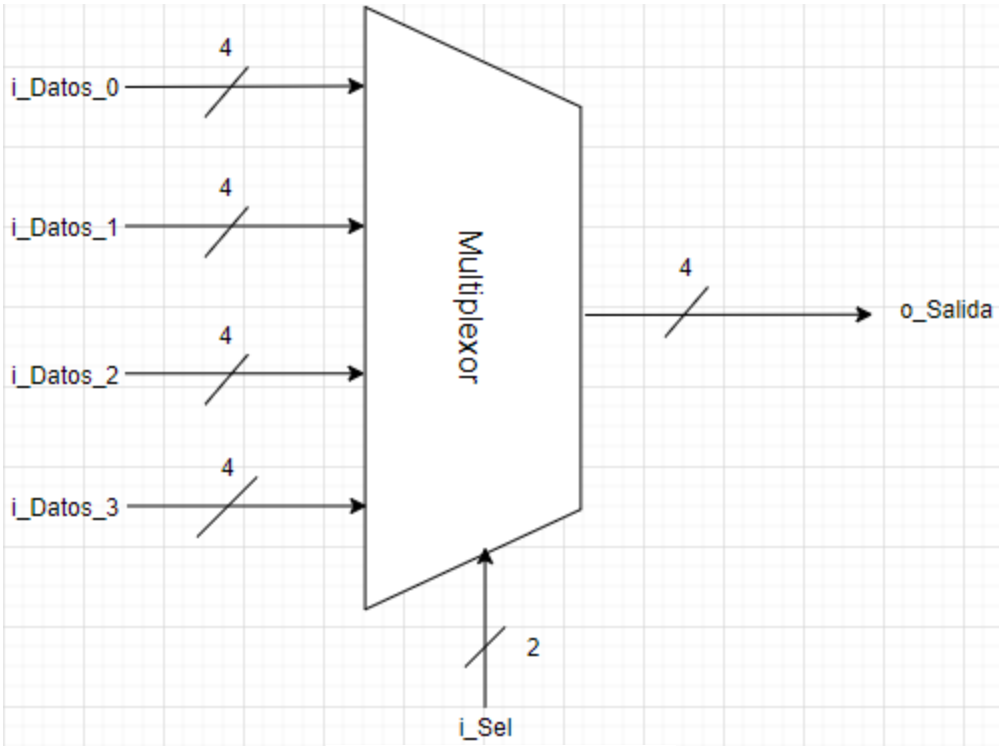

Imagen 9 Diagrama de multiplexor

Decodificador

Al implementar en nuestro código lo antes ya mencionado para obtener la arquitectura a través de nuestra herramienta vivado nos quedó de la siguiente forma:

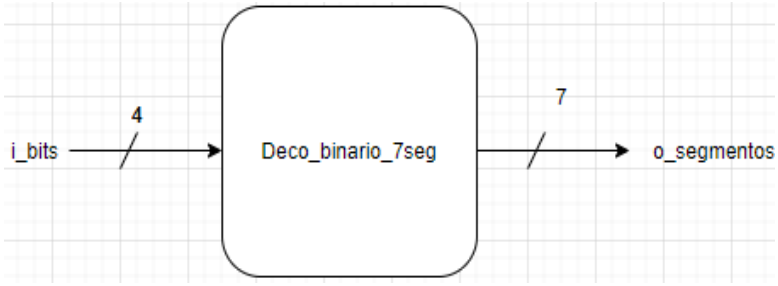

Imagen 10 Diagrama del decodificador

Implementación y simulación

En nuestra simulación comenzamos cerciorándonos de que cada uno de nuestros códigos funcionaran de manera correcta así mismo la simulación individual de cada código como se muestra a continuación.

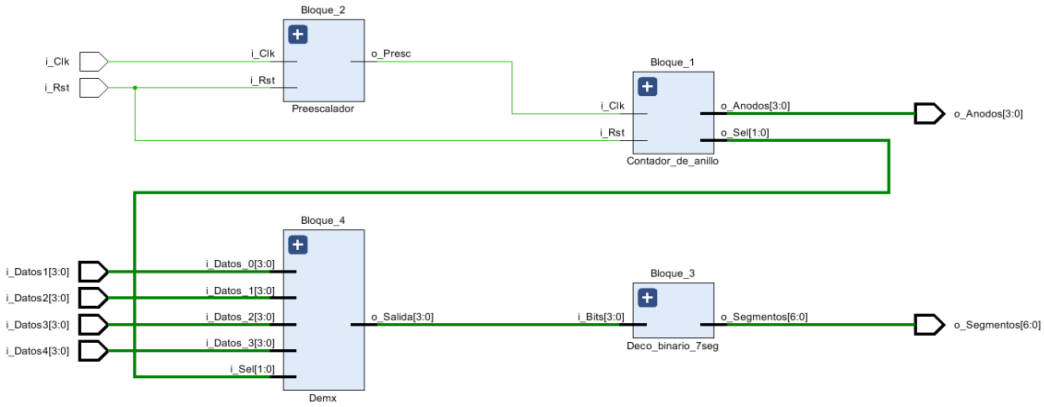

Imagen 11 Diagrama completo de vivado

Preescalador

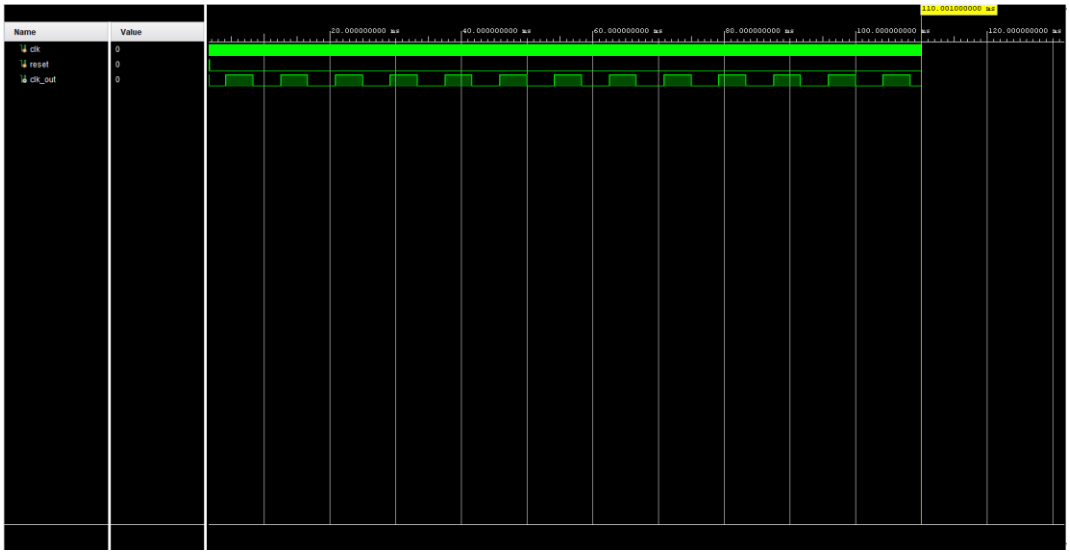

Imagen 12 Simulación del preescalador

Contador de anillo

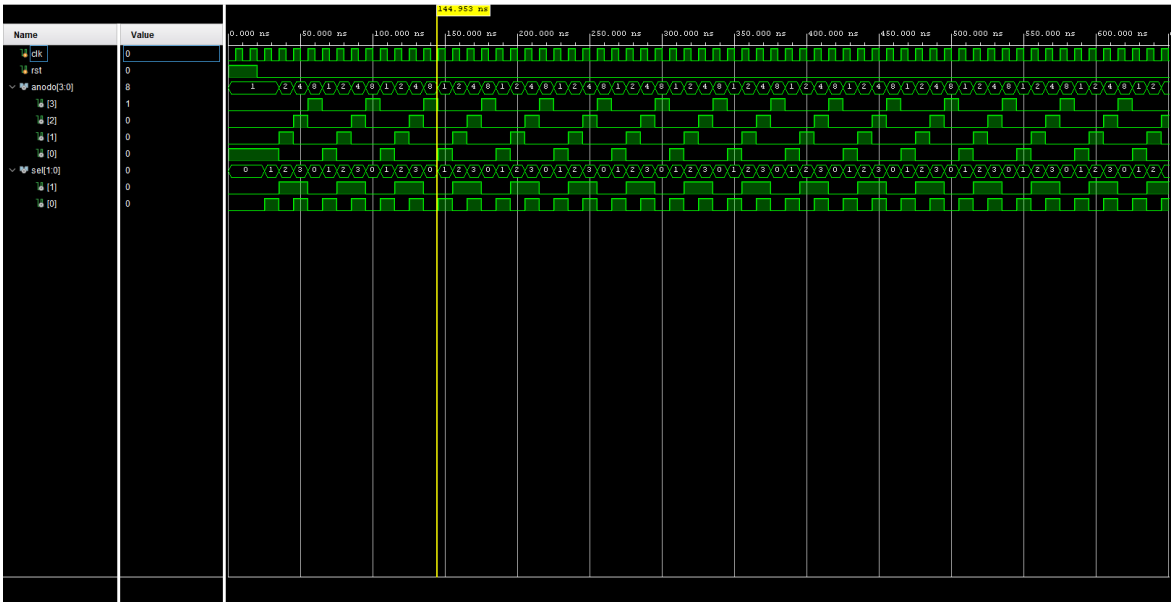

Imagen 13 Simulación del contador de anillo

Decodificador

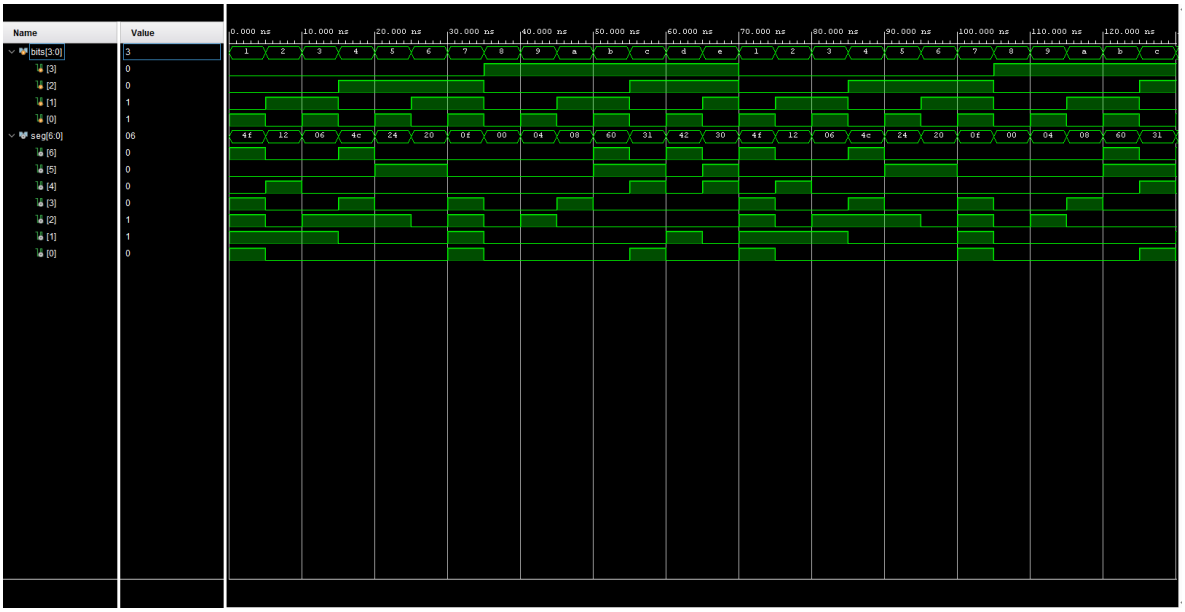

Imagen 14 Simulación del decodificador

Multiplexor

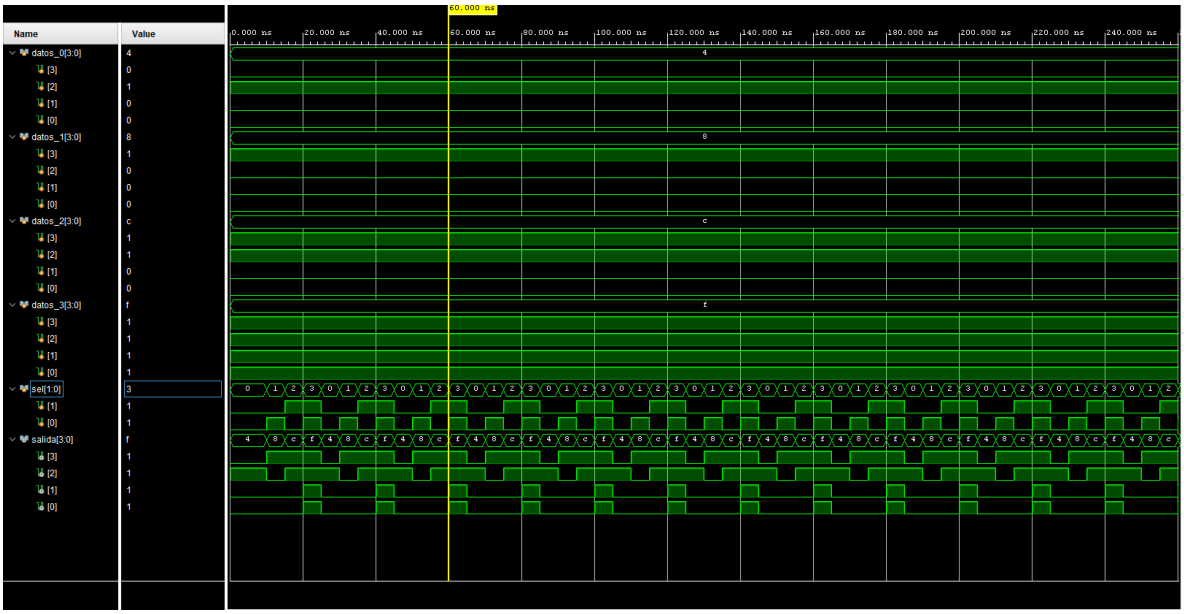

Imagen 15 Simulación de multiplexor

Comprobando que los códigos funcionaban de manera correcta gracias a las simulaciones procedimos a realizar el código completo para corroborar que todo en conjunto funciona de manera correcta realizando su respectiva simulación.

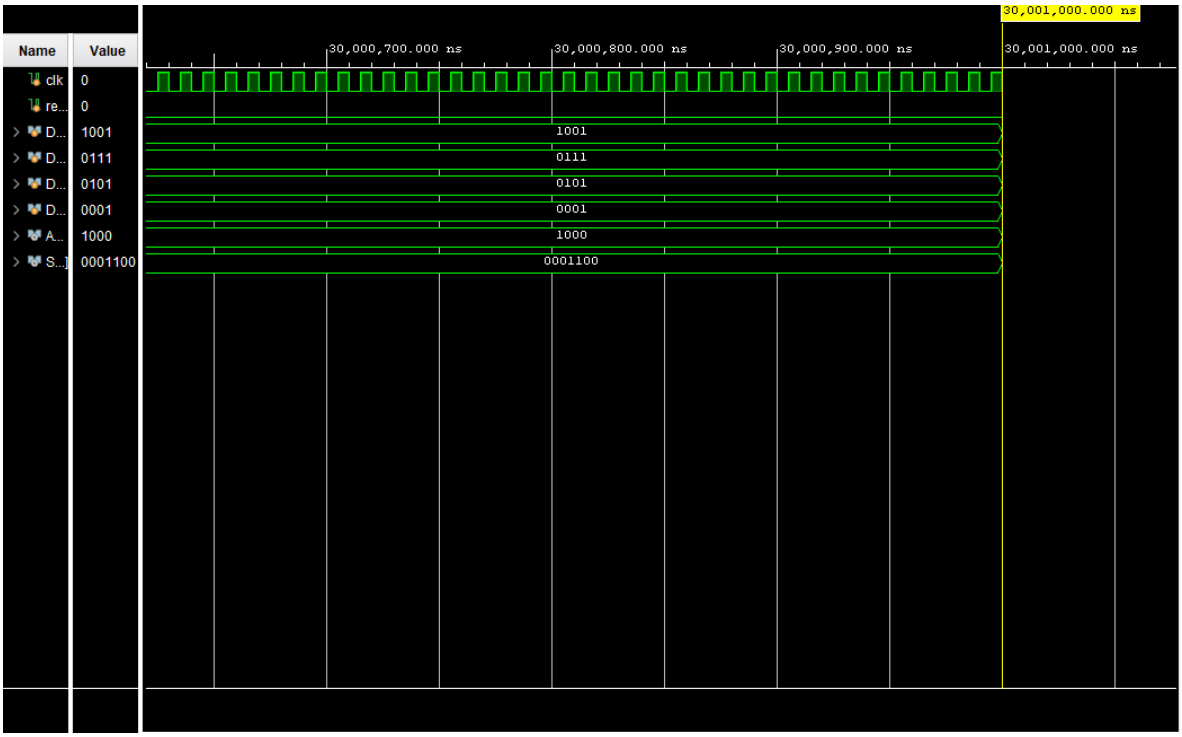

Imagen 16 Simulación de todo el programa

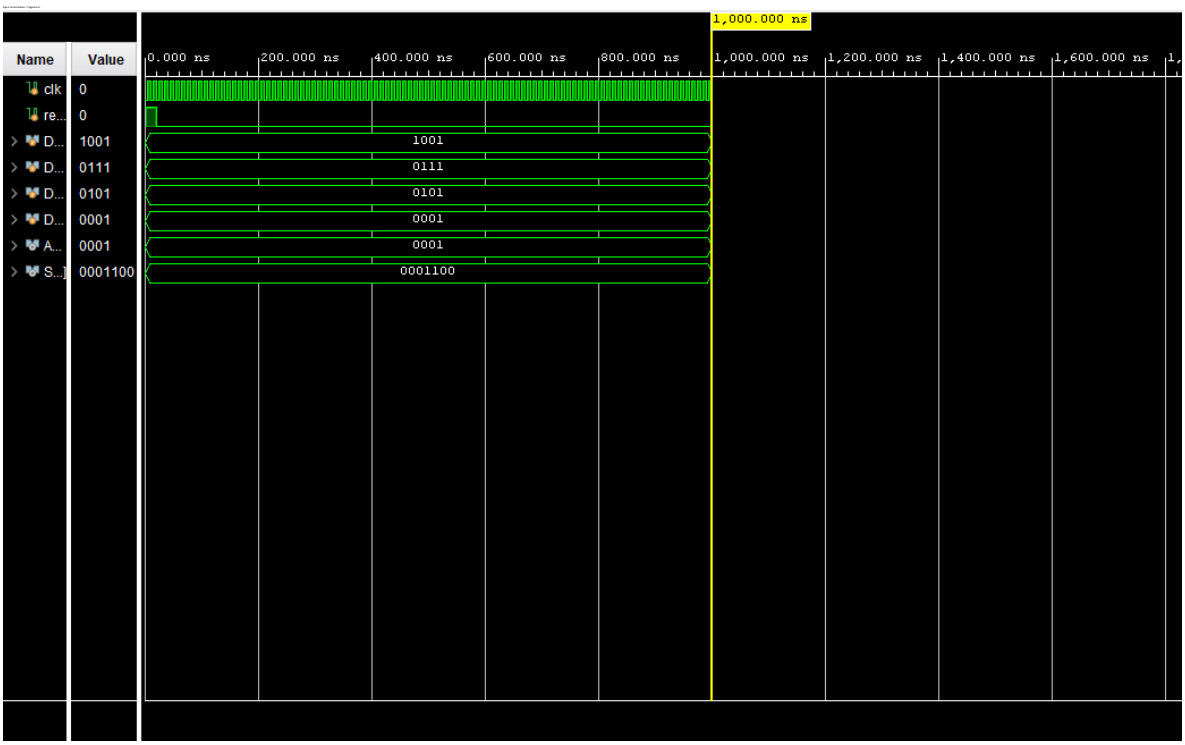

Imagen 17 Simulación completa

Ya que funciono de manera correcta el código y la simulación de todo en conjunto obtuvimos de igual manera nuestro proyect summary y nuestro device.

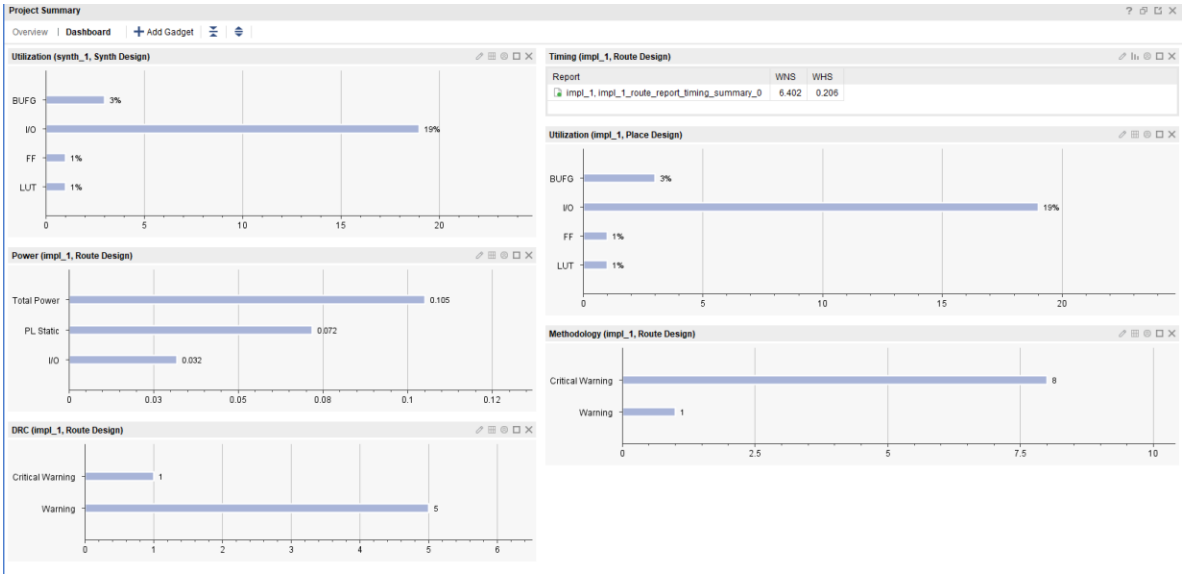

Imagen 18 Proyect summary

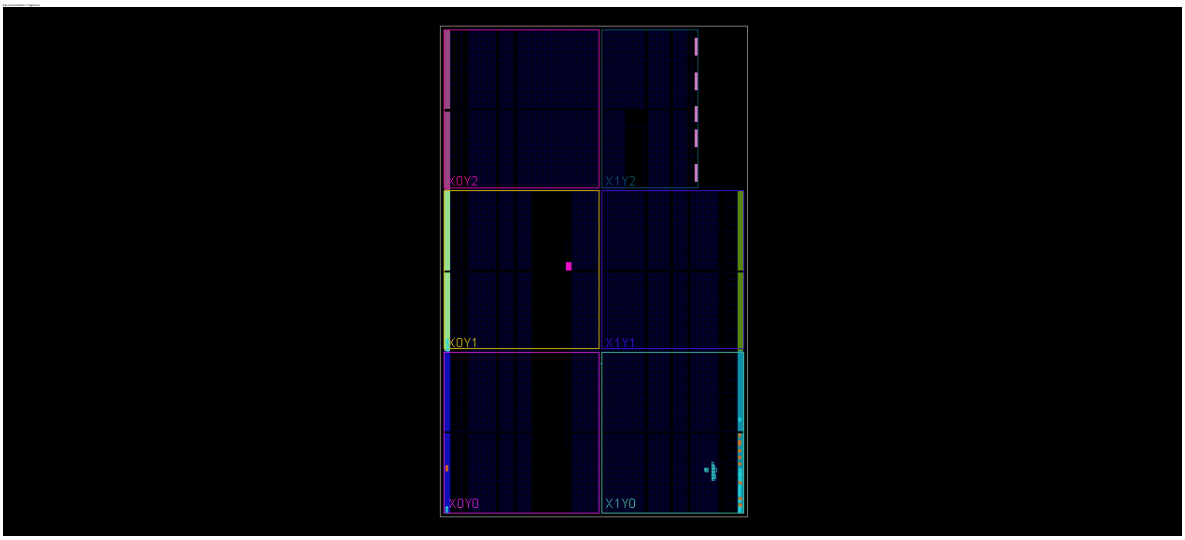

Imagen 19 Device

Análisis de resultados

Gracias al trabajo en conjunto de todos y la tutela de nuestro profesor pudimos llevar acabo dicho proyecto con el cual nos mostró no solo el funcionamiento de una display de 7 segmentos si no también la utilización de Verilog, nuevas plataformas de trabajo y muchas nuevas dinámicas de este que nos ayudaran en nuestro día cotidiano como futuros ingenieros.

Los resultados en nuestras simulaciones nos señalan que no debería de haber mayor complicación en caso de que pudiéramos subirlo a nuestra basys 3 lo cual esperamos algún día poder hacerlo nosotros mismos.

Los resultados fueron más de los esperados a pesar de las dificultades que llegamos a tener con transformación de diagrama a código, fue muy desafiante en un principio el crear el código de Contador de anillo, pero uno de nuestros compañeros nos ayudó a realizarlo después de ese problema no hubo ninguna complicación.

Conclusiones

El display de 7 segmentos está constituido de varias partes que son indispensables para su funcionamiento con este proyecto en conjunto nos dimos cuenta de la importancia de cada una de sus componentes y que no solo el display necesita el buen funcionamiento de todo para operar correctamente, sino que también un equipo de trabajo el cual se verá reflejado gravemente cualquiera de sus fallas.

En el caso de nuestro equipo hubo varias complicaciones con la utilización de Verilog y de nuestro código lo cual se resolvió favorablemente y gracias a estos problemas ocurridos logramos un mayor conocimiento sobre este tema y el funcionamiento de un contador de anillo para luego implementar este conocimiento en los códigos faltantes para así lograr finalizar nuestro trabajo.

En general esta práctica fue bastante productiva para nosotros logrando comprender conceptos de los cuales antes no teníamos conocimiento.

Referencias

Authorized translation from the English language edition, entitled DIGITAL FUNDAMENTALS, 9TH Edition by FLOYD, THOMAS L., published by Pearson Education Inc, publishing as Prentice Hall, Copyright © 2006

Ingeniería Mecafenix. (2020, 22 junio). Display de 7 segmentos (como se usa). https://www.ingmecafenix.com/electronica/display-de-7-segmentos/

A. (2018, 24 enero). Display 7 Segmentos ánodo y cátodo común. HETPRO/TUTORIALES. https://hetpro-store.com/TUTORIALES/display-7-segmentos-anodo-catodo-comun/

Apéndice A (código)

```
module Controlador_Display_7_seg            //Nombre del modulo
    (   input i_Clk,                        //Entrada del reloj
        input i_Rst,                        //Entrada del reset
        input [3:0] i_Datos1,               //Entrada de 4 bits llamado
                                            i_Datos1
        input [3:0] i_Datos2,               //Entrada de 4 bits llamado
                                            i_Datos2
        input [3:0] i_Datos3,               //Entrada de 4 bits llamado
                                            i_Datos3
        input [3:0] i_Datos4,               //Entrada de 4 bits llamado
                                            i_Datos4
        output [3:0] o_Anodos,              //Salida de 4 bits llamado
                                            o_Anodos
        output [6:0] o_Segmentos);          //Salida de 7 bits llamado
                                            o_Segmentos

    wire [3:0] Demx_deco_Sel;               //Cable interno de 4 bits
                                            llamado De
    wire [1:0] Contador_demx_sel;           //Cable interno de 2 bits
                                            llamado Contador_demx_sel
    wire Prescalador_contador_tiempo;       //Cable interno llamado
                                            Prescalador_contador_tiempo

    Contador_de_anillo Bloque_1(            //Nombre del primer bloque
                                            Contador_de_anillo
        .i_Clk(Prescalador_contador_tiempo),//Renombrando las entradas y
                                            salidas internas del bloque
        .i_Rst(i_Rst),                      //Renombrando las entradas
                                            y salidas internas del bloque
        .o_Anodos(o_Anodos),                //Renombrando las entradas y
                                            salidas internas del bloque
        .o_Sel(Contador_demx_sel)           //Renombrando las entradas y
                                            salidas internas del bloque
    );

    Preescalador Bloque_2(                  //Nombre del segundo bloque
                                            Preescalador
    .i_Clk(i_Clk),                          //Renombrando las entradas y
                                            salidas internas del bloque
    .i_Rst(i_Rst),                          //Renombrando las entradas y
                                            salidas internas del bloque
    .o_Presc(Prescalador_contador_tiempo)   //Renombrando las entradas y
                                            salidas internas del bloque
    );

    Deco_binario_7seg Bloque_3(             //Nombre del tercer bloque
                                            Deco_binario_7seg
        .i_Bits(Demx_deco_Sel),             //Renombrando las entradas y
                                            salidas internas del bloque
        .o_Segmentos(o_Segmentos)           //Renombrando las entradas y
                                            salidas internas del bloque
    );

    Demx Bloque_4(                          //Nombre del cuarto bloque
                                            Demx
        .i_Sel(Contador_demx_sel),          //Renombrando las entradas y
                                            salidas internas del bloque
        .i_Datos_0(i_Datos1),               //Renombrando las entradas y
                                            salidas internas del bloque
        .i_Datos_1(i_Datos2),               //Renombrando las entradas y
                                            salidas internas del bloque
        .i_Datos_2(i_Datos3),               //Renombrando las entradas y
                                            salidas internas del bloque
```

```
        .i_Datos_3(i_Datos4),                //Renombrando las entradas y
                                             salidas internas del bloque
        .o_Salida(Demx_deco_Sel)             //Renombrando las entradas y
                                             salidas internas del bloque
    );

endmodule                                    //Acaba el modulo
-------------------------------------------------------------------------------
module Contador_de_anillo(                  //Nombre del modulo
    input i_Clk,                            //Entrada del reloj
    input i_Rst,                            //Entrada del reset
    output reg [3:0] o_Anodos,              //Salida de 4 bits llamado
                                            o_Anodos
    output reg [1:0] o_Sel);                //Salida de 2 bits llamada o_Sel
 always @(posedge i_Rst, posedge i_Clk)     //Funcion que se ejecuta
                                            en forma de bucle, hace que el código espere
                                            el rise del clock y también el reset para
                                            ejecutar el código
    begin                                   //Se engloba el proceso en un
                                            bloque
        if(i_Rst) begin                     //Condicional que revisa si se
                                            presiono el reset
            o_Anodos<= 4'b0001;             //Inicializa los ánodos en 0001
            o_Sel<=2'b00;                   //Inicializa la selección en 00
        end                                 //Finaliza el condicional
        else begin                          //Condicional que engloba un
                                            bloque
            case(o_Sel)                     //Case que toma el valor que
                                            está en o_Sel
                2'b00: begin                //Cuando s_Sel se encuentra en
                                            00 inicia el siguiente bloque de códigos
                    o_Anodos<= 4'b0001;     //Se asigna el valor a o_Anodos
                                            0001
                    o_Sel<=o_Sel+1;         //Se suma un 1 al valor de o_Sel
                end                         //Termina el bloque de código
               2'b01: begin                 //Inicia el bloque de código
                                            cuando o_Sel tiene el valor 01
                    o_Anodos<= 4'b0010;     //Se asigna el valor a o_Anodos
                                            0010
                    o_Sel<=o_Sel+1;         //Se suma un 1 al valor de o_Sel
                end                         //Termina el bloque de código
                2'b10: begin                //Se ejecuta el siguiente bloque
                                            de código cuando o_Sel tiene los valores 10
                    o_Anodos<= 4'b0100;     //Se asigna el valor a o_Anodos
                                            0100
                    o_Sel<=o_Sel+1;         //Se suma un 1 al valor de o_Sel
                end                         //Termina el bloque de código
                default: begin              //El siguiente bloque de código
                                            se ejecuta cuando no hay otro case disponible
                    o_Anodos<= 4'b1000;     //Se asigna el valor a o_Anodos
                                            1000
                    o_Sel<=2'b00;           //Se le asigna a o_Sel un valor
                                            de 00
                end                         //Termina el bloque de código
            endcase                         //Termina el código dentro del
                                            condicional
        end                                 //Termina el bloque de código
    end                                     //Termina el bloque de código

endmodule                                   //Acaba el modulo
-------------------------------------------------------------------------------
module Preescalador(                    //Nombre del módulo
  input i_Clk,                          //Declaración de la entrada llamada
                                        i_Clk
```

```
  input i_Rst,                    //Declaración de la entrada llamada
                                  i_Rst
  output o_Presc                  //Declaración de la salida llamada
                                  o_Presc
  );
  reg [19:0] contador;            //Registro de 20 bits llamado contador
  reg salida;                     //Registro llamado salida
  always@(posedge i_Clk)          //Función que se ejecuta en forma de
                                  bucle, hace que el código espere el rise del clock
                                  y también el reset para ejecutar el código
  begin                           //Se engloba el proceso en un buble
      if (i_Rst)                  //Condicional que se activa cuando se
                                  presiona i_reset
      contador <= 0;              //Se asigna el valor a contador
      else                        //Condicional que engloba otro bloque
      contador <= contador+1;     //Se asigna nuevo valor a contador
      if (contador <= 416666)     //Condicional que se activa cuando se
                                  cumple la condición
      salida <= 0;                //Se asigna el valor de salida
      else                        //Condicional que engloba otro bloque
      salida <= 1;                //Se asigna nuevo valor a salida
      if (contador == 833333)     //Condicional que se activa cuando se
                                  cumple la condición
      contador <= 0;              //Se asigna el valor a contador
  end                             //Termina el bloque de código
  assign o_Presc = salida;        //Se iguala el valor de assign o_Presc
                                  a salida
endmodule                         //Acaba el modulo
```

```
module Deco_binario_7seg(                 //nombre del módulo
  input [3:0] i_Bits,                     //entrada de 4 bits llamada
                                  i_bits
  output reg [6:0] o_Segmentos            //salida de 7 bits llamada
                                  o_segmentos
  );
always@(i_Bits)                           //función que se ejecuta en
                                  forma de bucle
  begin                                   //Se engloba el proceso en un
                                  bloque
    case(i_Bits)                          //case para que según el valor
                                  de i_bits se asignará cierto
                    valor a o_segmentos
      4'b0000: o_Segmentos = 7'b0000001;  //0
      4'b0001: o_Segmentos = 7'b1001111;  //1
      4'b0010: o_Segmentos = 7'b0010010;  //2
      4'b0011: o_Segmentos = 7'b0000110;  //3
      4'b0100: o_Segmentos = 7'b1001100;  //4
      4'b0101: o_Segmentos = 7'b0100100;  //5
      4'b0110: o_Segmentos = 7'b0100000;  //6
      4'b0111: o_Segmentos = 7'b0001111;  //7
      4'b1000: o_Segmentos = 7'b0000000;  //8
      4'b1001: o_Segmentos = 7'b0001100;  //9
      4'b1010: o_Segmentos = 7'b0001000;  //A
      4'b1011: o_Segmentos = 7'b1100000;  //B
      4'b1100: o_Segmentos = 7'b0110001;  //C
      4'b1101: o_Segmentos = 7'b1000010;  //D
      4'b1110: o_Segmentos = 7'b0110000;  //E
      4'b1111: o_Segmentos = 7'b0111000;  //F
      default: o_Segmentos = 7'b1111111;  //Por defecto estén apagados
    endcase                               //se cierra el case
  end                                     //termina el bloque
endmodule                                 //termina el módulo
```

```
module Demx                               //Nombre del modulo
```

```
    (   input [3:0] i_Datos_0,          //Entrada de 4 bits llamada
                                        i_Datos_0
        input [3:0] i_Datos_1,          //Entrada de 4 bits llamada
                                        i_Datos_1
        input [3:0] i_Datos_2,          //Entrada de 4 bits llamada
                                        i_Datos_2
          input [3:0] i_Datos_3,        //Entrada de de 4 bits llamada
                                        i_Datos_3
          input [1:0] i_Sel,            //Entrada de 2 bits llamada
                                        i_Sel
        output reg [3:0] o_Salida);     //Salida de 4 bits llamada
                                        o_Salida

always@(i_Sel) begin                    //Función que ejecuta en forma
                                        de bucle
            case (i_Sel)                //Condicional que revisa i_Sel
                  2'b00: o_Salida <= i_Datos_0; //Cuando i_sel esta en
                                                estos 1'b00 se le asigna a o_Salida los
                                                datos de i_Datos_0
                  2'b01: o_Salida <= i_Datos_1; //Cuando i_sel está en
                                                estos 1'b01 se le asigna a o_Salida los
                                                datos de i_Datos_1
                  2'b10: o_Salida <= i_Datos_2; //Cuando i_sel está en
                                                estos 1'b10 se le asigna a o_Salida los
                                                datos de i_Datos_2
                  default : o_Salida <= i_Datos_3; //En caso contrario
                                                que no se cumpla una de las anteriores
            endcase                             //Fin de la condicional
      end                                     //Termina el bloque de código

endmodule                                       //Termina el modulo
```

Apéndice B (Testbench)

```
/module Controlador_Display_7_seg_TB(   //Nombre del módulo del testbench
);
    reg clk,reset;                      //Registros llamados clk y reset
    reg [3:0] D0,D1,D2,D3;              //Registros de 4 bits llamados
                                        D0, D1, D2 y D3
    wire [3:0] AN;                      //Cable de 4 bits llamado AN
    wire [6:0] Seg;                     //Cable de 7 bits llamado Seg

    Controlador_Display_7_seg DUT(      //Nombre de la unidad bajo testeo
                                         (DUT)
        .i_Clk(clk),                    //Asignación de los nombres de
                                         las variables de otros bloques
        .i_Rst(reset),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .i_Datos1(D0),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .i_Datos2(D1),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .i_Datos3(D2),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .i_Datos4(D3),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .o_Anodos(AN),                  //Asignación de los nombres de
                                         las variables de otros bloques
        .o_Segmentos(Seg)               //Asignación de los nombres de
                                         las variables de otros bloques
    );

    initial                             //Inicializa el testbench
    begin                               //Inicia el bloque de código
```

```
        clk<=0;                         //Se le asigna a clk el valor 0
        reset<=1;                       //Se le asigna a reset el valor
                                          de 1
        D0<=9;                          //Se le asigna a d0 el valor de 9
        D1<=7;                          //Se le asigna a d1 el valor de 7
        D2<=5;                          //Se le asigna a d2 el valor de 5
        D3<=1;                          //Se le asigna a d3 el valor de 1
        #20 reset<=0;                   //Pasa tiempo y el reset se le
                                          asigna el valor de 0
    end                                 //Termina el bloque de código

    always@(clk) begin                  //Inicializa el reloj
        #5 clk<=~clk;                   //El reloj se actualiza cada
                                          cierto tiempo
    end                                 //Termina el reloj

endmodule                               //Termina el testbench
-------------------------------------------------------------------------------
module Contador_de_anillo_TB(           //Nombre del módulo testbench
);
    reg clk;                            //Registro del relog
    reg rst;                            //Registro del reset
    wire [3:0] anodo;                   //Cables de 4 bits de nombre ánodo
    wire [1:0] sel;                     //Cables de 2 bits de nombre sel

    Contador_de_anillo DUT (            //Nombre de la unidad bajo
                                          testeo(DUT)
        .i_Clk(clk),                    //Asignación de los nombres de las
                                          variables de otros bloques
        .i_Rst(rst),                    //Asignación de los nombres de las
                                          variables de otros bloques
        .o_Anodos(anodo),               //Asignación de los nombres de las
                                          variables de otros bloques
        .o_Sel(sel)                     //Asignación de los nombres de las
                                          variables de otros bloques
    );

    initial begin                       //Inicia el testbench
        clk<=0;                         //Se le asigna valor al reloj
        rst<=1;                       //Se le asigna el valor de 1 al reset
        #20 rst<=0;                     //Pasa tiempo y se le asigna el
                                          valor 0 al reset
    end                                 //Termina la inicialización del
                                          testbench

    always@(clk) begin                  //Inicia la lista de sensitividad
                                          para el clock
        #5 clk <= ~clk;                 //Líneas de código para el reloj
    end                                 //Termina el módulo de código

endmodule                               //Termina el módulo testbench

-------------------------------------------------------------------------------
module Preescalador_TB(                 //Nombre del modulo
);
    reg clk;                            //registro llamado reloj
    reg reset;                          //registro llamado reset
    wire clk_out;                       //Cable de salida del reloj
    Preescalador DUT (                  //Diseño bajo testeo de multiplexor
            .i_Clk(clk),                //Especifica que i_Clk es clk para
                                          su futuro uso
            .i_Rst(reset),              //Especifica que i_Rst es reset
                                          para su futuro uso
            .o_Presc(clk_out)           //Especifica que o_Presc es clk_out
```

```
                                        para su futuro uso
    );
    initial                            //Iniciación del testbench
    begin                              //Función que ejecuta en forma de
                                         bucle
            clk<=0;                    //Asigna el valor 0 a clk
            reset<=1;                  //Asigna el valor 1 a reset
            #20 reset<=0;              //Asigna el valor 0 a reset
    end                                //Termina el bloque de código
    always@(clk)                       //Función que ejecuta en forma de
                                         bucle
    #5 clk <= ~clk;                    //El reloj se actualiza cada
                                         cierto tiempo
endmodule                              //Termina el modulo

-------------------------------------------------------------------------------
module Preescalador_TB(                //Nombre del modulo
);
    reg clk;                           //registro llamado reloj
    reg reset;                         //registro llamado reset
    wire clk_out;                      //Cable de salida del reloj
    Preescalador DUT (                 //Diseño bajo testeo de
                                         multiplexor
            .i_Clk(clk),               //Especifica que i_Clk es clk para
                                         su futuro uso
            .i_Rst(reset),             //Especifica que i_Rst es reset
                                         para su futuro uso
            .o_Presc(clk_out)          //Especifica que o_Presc es
                                         clk_out para su futuro uso
    );
    initial                            //Iniciación del testbench
    begin                              //Función que ejecuta en forma de
                                         bucle
            clk<=0;                    //Asigna el valor 0 a clk
            reset<=1;                  //Asigna el valor 1 a reset
            #20 reset<=0;              //Asigna el valor 0 a reset
    end                                //Termina el bloque de código
    always@(clk)                       //Función que ejecuta en forma de
                                         bucle
    #5 clk <= ~clk;                    //El reloj se actualiza cada
                                         cierto tiempo
endmodule                              //Termina el modulo

-------------------------------------------------------------------------------
/`timescale 1ns / 1ps


module Multiplexor_TB(                 //Nombre del modulo

    );

 reg [3:0] Datos_0;                    //Entrada de 4 bits llamada
                                         i_Datos_0
 reg [3:0] Datos_1;                    //Entrada de 4 bits llamada
                                         i_Datos_1
 reg [3:0] Datos_2;                    //Entrada de 4 bits llamada
                                         i_Datos_2
 reg [3:0] Datos_3;                    //Entrada de 4 bits llamada
                                         i_Datos_3
 reg [1:0] Sel;                        //Entrada de 2 bits llamada i_Sel
 wire [3:0] Salida;                    //Salida de 4 bits llamada
                                         o_Salida

 Multiplexor DUT(                      //Diseño bajo testeo de
```

```
                                        multiplexor
     .i_Datos_0(Datos_0),               // Especifica que i_Datos_0 es
                                        Datos_0 para su futuro uso
     .i_Datos_1(Datos_1),               // Especifica que i_Datos_1 es
                                        Datos_1 para su futuro uso
     .i_Datos_2(Datos_2),               // Especifica que i_Datos_2 es
                                        Datos_2 para su futuro uso
     .i_Datos_3(Datos_3),               // Especifica que i_Datos_3 es
                                        Datos_3 para su futuro uso
     .i_Sel(Sel),                       // Especifica que i_Sel es Sel
                                        para su futuro uso
     .o_Salida(Salida)                  // Especifica que o_Salida es
                                        Salida para su futuro uso
 );

 initial                                //Iniciación del testbench
begin                                   //Función que ejecuta en forma
                                        de bucle

  Datos_0 <=0;                          //Asigna el valor de 0 a Datos_0
  Datos_1 <=0;                          //Asigna el valor de 0 a Datos_1
  Datos_2 <=0;                          //Asigna el valor de 0 a Datos_2
  Datos_3 <=0;                          //Asigna el valor de 0 a Datos_3
  Sel <=0;                              //Asigna el valor de 0 a Sel
  #50;
  Datos_0 [0]<=1;                       //En el vector 0 de Datos_0 se
                                        asigna el valor de 1
  Datos_1 [1]<=1;                       //En el vector 0 de Datos_1 se
                                        asigna el valor de 1
  Datos_2 [2]<=1;                       //En el vector 0 de Datos_2 se
                                        asigna el valor de 1
  Datos_3 [3]<=1;                       //En el vector 0 de Datos_3 se
                                        asigna el valor de 1
  #50;                                  //Pasa Tiempo
  Sel <=0;                              //Asigna el valor 0 a Sel
  #50;                                  //Pasa Tiempo
  Sel [0]<=1;                           //En el vector 0 de Sel se
                                        asigna el valor de 1
  Sel [1]<=0;                           //En el vector 1 de Sel se
                                        asigna el valor de 0
  #50;                                  //Pasa Tiempo
  Sel [0]<=0;                           //En el vector 0 de Sel se
                                        asigna el valor de 0
  Sel [1]<=1;                           //En el vector 1 de Sel se
                                        asigna el valor de 1
  #50;                                  //Pasa Tiempo
  Sel [0]<=1;                           //En el vector 0 de Sel se
                                        asigna el valor de 1
  Sel [1]<=1;                           //En el vector 1 de Sel se
                                        asigna el valor de 1
  #50;                                  //Pasa Tiempo
end                                     //Termina el bloque de código
endmodule                               //Termina el modulo
```