Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Práctica 2 - Circuitos combinatorios 2

Profesor:
M. en C. Osvaldo Espinosa Sosa

Por:
Ing. Ricardo Aldair Tirado Torres

Ciudad de México, 27 de marzo de 2024

Tabla de contenido

1. Objetivos

- Aprender acerca del uso de *Chip Planner*, así como de los elementos lógicos y el hardware dedicado en el FPGA y como es que la herramienta mencionada implementa los circuitos dentro de uno u otro bloque.
- Comprender la operación e implementación de circuitos combinatorios importantes como el desplazador lógico, el multiplexor, la memoria ROM y el buffer tri-estado.
- Diferenciar las dos implementaciones de un multiplexor utilizando la estructura *case* y la estructura *if-else*.

2. Desplazador lógico

Actividad 1

Completar el código del desplazador lógico utilizando concatenaciones en el lenguaje de su elección. Compilar y simular. Configurar en la tarjeta DE2-115 asignando *switches* y *LED's*. Datos de 8 bits.

La visualización RTL del desplazador lógico en Verilog se muestra en la Figura 1. En el visor se tiene un decodificador conectado junto con 8 multiplexores (uno por cada bit a la salida). Las simulaciones para el código en Verilog se visualizan en la Figura 2. Los bits a la entrada fueron escogidos de tal forma que se observe de manera clara el corrimiento a la izquierda.

En los Anexos se localiza la descripción del desplazador lógico. Se observa que la descripción se hizo utilizando una lista sensible para los puertos de entrada y selección. Dentro de esta estructura se empleó la sentencia *case* para indicar como se debe comportar la salida con respecto a la variable de control SEL. Cabe resaltar que para el desplazamiento lógico a la izquierda se concatenaron los bits menos significativos de la entrada a la izquierda y valores en bajo a la derecha. En caso de que se hubiera hecho un desplazamiento a la derecha, se debe hacer la concatenación de valores en bajo a la izquierda y los bits más significativos de la entrada a la derecha.

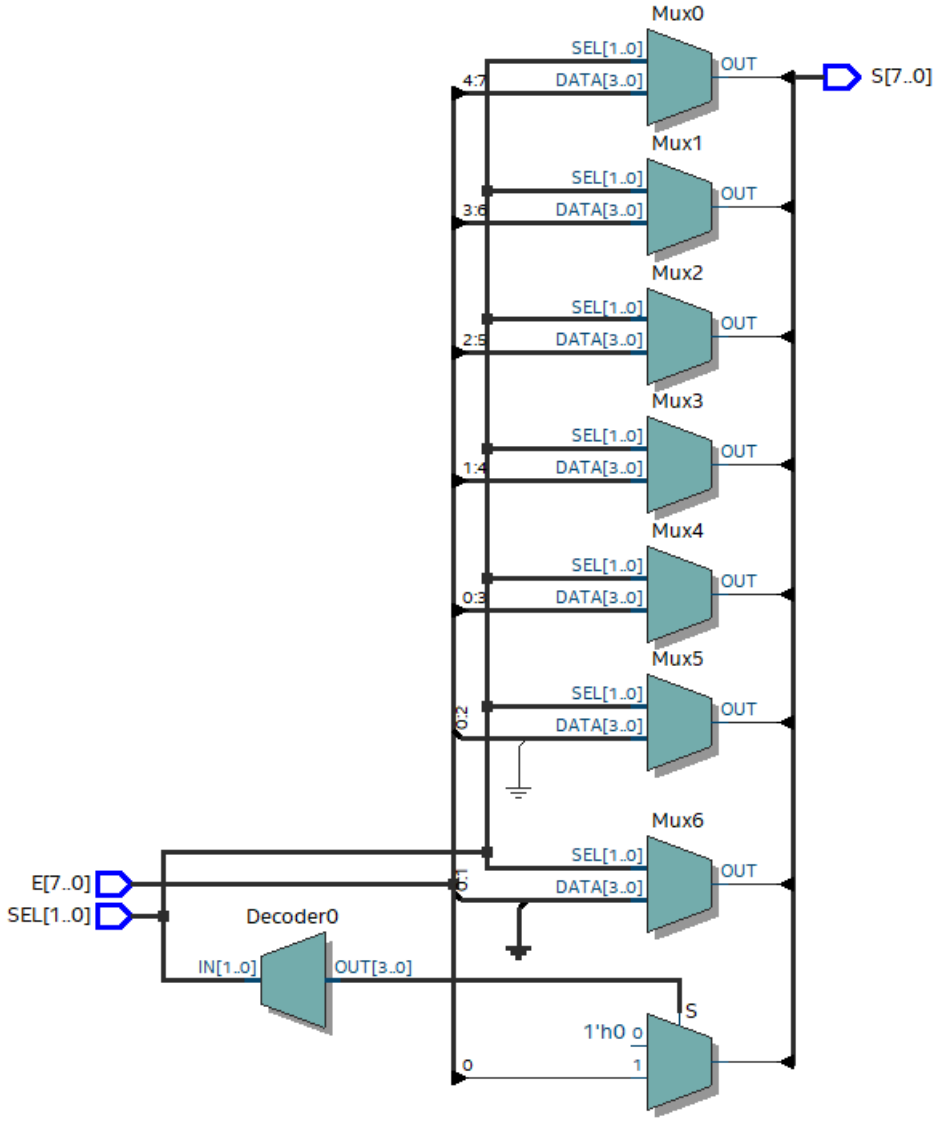

Figura 1: Diagrama RTL del desplazador lógico de 8 bits.

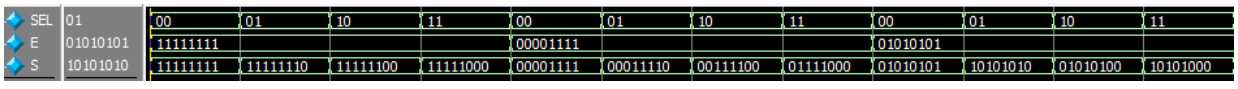

Figura 2: Simulación del desplazador lógico con el visor de formas de onda de ModelSim.

3. Multiplexor 4 a 1

Actividad 2

Completar el código del multiplexor 4 a 1 en el lenguaje de su elección primero usando estructura *'case'* y después *'ifs'* anidados. Compilar y simular. Usar el visor RTL para establecer si hay diferencias en el resultado de la síntesis de los dos casos. implementar en la tarjeta DE2-115 solo el caso de la estructura *"case"*.

La visualización RTL del multiplexor 4 a 1 en Verilog se muestra en la Figura 3 utilizando la estructura *case* y en la Figura 4 utilizando la estructura *if*. Como se observa, el multiplexor con la estructura *case* se implementa empleando una instancia de multiplexor 4 a 1, con un tamaño de selección de 2 bits, en cambio, el que se describió con estructura *if* hace uso de varios elementos lógicos, específicamente 4 comparadores de magnitud y 4 multiplexores 2 a 1. Las simulaciones para el código en Verilog se visualizan en la Figura 5 para el multiplexor de estructura *case* y en la Figura 6 para el de la estructura *if*. Los bits de entrada fueron escogidos de tal forma que se observe de manera clara el funcionamiento del módulo.

En los Anexos se localiza la descripción de las dos implementaciones de multiplexor. Para el primer caso se utilizó la sentencia *case* para evaluar cada estado de la variable de control SEL, todo esto dentro de una lista sensible. En cambio, para el segundo caso se usó una sentencia *if* por cada estado de la variable SEL, igualmente dentro de una lista sensible.

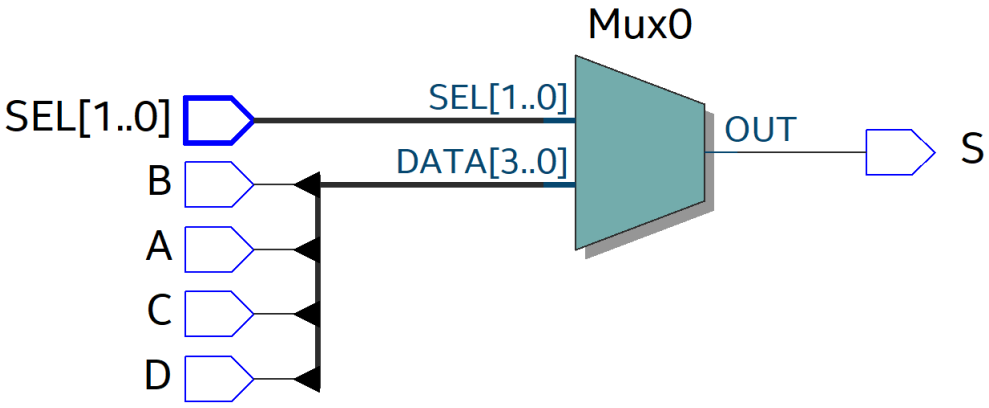

Figura 3: Diagrama RTL del multiplexor 4 a 1, utilizando la estructura *case*.

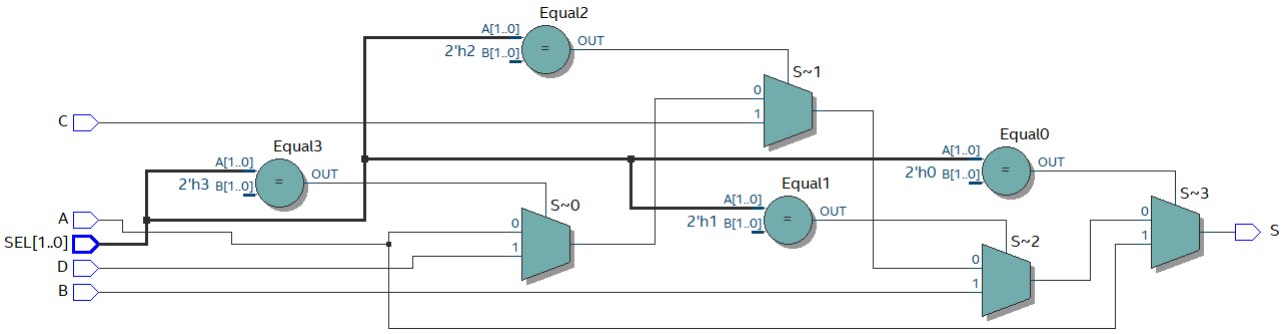

Figura 4: Diagrama RTL del multiplexor 4 a 1, utilizando la estructura *if*.

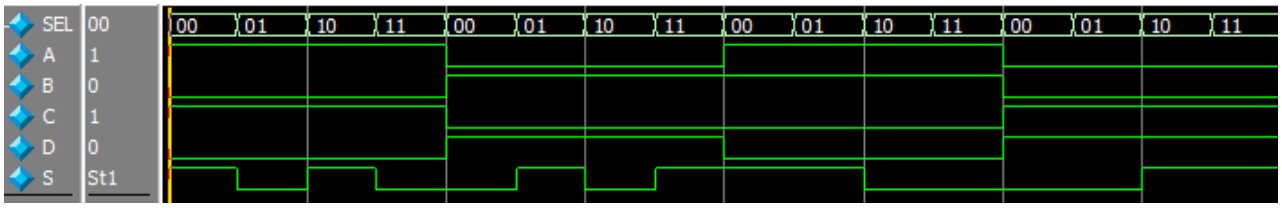

Figura 5: Simulación del multiplexor 4 a 1, utilizando la estructura *case* con el visor de formas de onda de ModelSim.

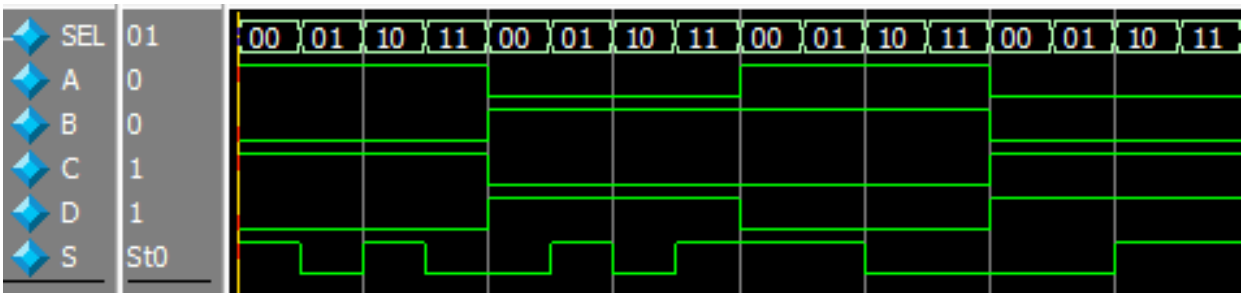

Figura 6: Simulación del multiplexor 4 a 1, utilizando la estructura *if* con el visor de formas de onda de ModelSim.

4. Memoria ROM 4x8

Actividad 3

Completar el código de la memoria ROM 4x8 en el lenguaje de su elección. Compilar y simular. Usar el visor RTL para indicar como se implementa después de la síntesis. Implementar en la tarjeta DE2-115.

La visualización RTL de la memoria ROM 4x8 en Verilog se muestra en la Figura 7. Como se observa, la implementación de la memoria ROM de 4 localidades de 8 bits se hace utilizando una instancia de decodificador cuya salida se conecta a un conjunto de compuertas lógicas, demostrando que la implementación de una memoria ROM es similar a la de un decodificador. Las simulaciones para el código en Verilog se visualizan en la Figura 8 en base binaria y en la Figura 9 en base decimal. Se utilizaron todos los valores posibles en la dirección de memoria para observar un comportamiento completo en la salida.

En los Anexos se localiza la descripción de la memoria ROM. Se observa que el código es muy similar al del decodificador visto en la Práctica 1, ya que emplea la estructura *case* dentro de una lista sensible. No obstante, se diferencia en que la ROM puede adquirir el tamaño que sea o hacer uso de cualquier número de localidades, en cambio el decodificador tiene un tamaño estático de 2^n, siendo n el número de bits o tamaño de la variable de control.

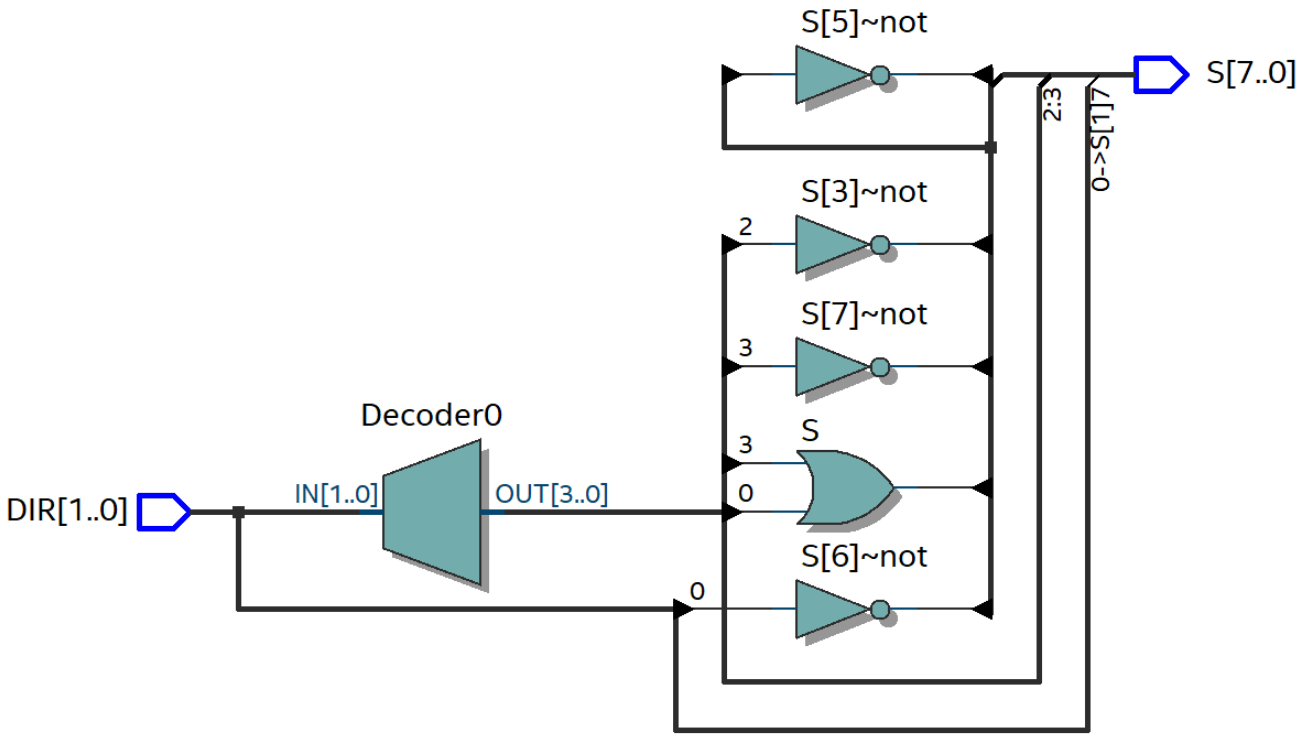

Figura 7: Diagrama RTL de la memoria ROM 4x8.

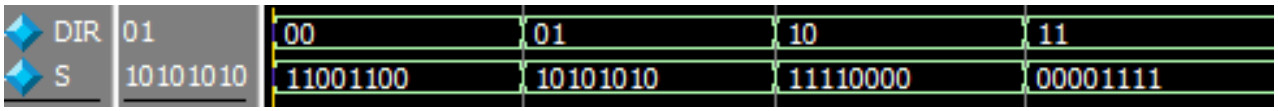

Figura 8: Simulación de la memoria ROM 4x8 con el visor de formas de onda de ModelSim (Base binaria).

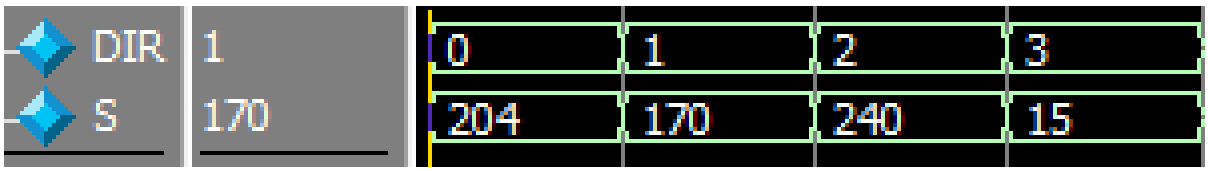

Figura 9: Simulación de la memoria ROM 4x8 con el visor de formas de onda de ModelSim (Base decimal).

5. Buffer Tri-estado

Actividad 4

Completar el código del buffer tri-estado en el lenguaje de su elección. Compilar y simular. Usar el visor RTL y después el *Chip planner* para ver donde se ubica el buffer. No implementar en la tarjeta.

La visualización RTL del buffer tri-estado en Verilog se muestra en la Figura 10. Al momento de utilizar la herramienta de *Chip Planner* se observa en la Figura 11 la implementación del modulo en la periferia del dispositivo. Si se acerca la imagen a este elemento (ver Figura 12) se puede ver que el buffer fue implementado por la herramienta en la linea de entradas y salidas debido a que los FPGS's manejan a sus puertos con buffers tri-estados para cambiar su función a entrada o salida. En la Figura 13, Figura 14 y Figura 15 se observan los pines A, ENABLE y F del modulo y abriendo el contenido de cada uno, se visualiza en la Figura 16, Figura 17 y Figura 18 la ubicación de cada puerto en el buffer. Las simulaciones para el código en Verilog se visualizan en la Figura 19.

En los Anexos se localiza la descripción del buffer tri-estado. La implementación es muy sencilla, puesto que solo se hace uso de una estructura *if-else* dentro de una lista sensible. Cabe señalar, que se hace uso del valor "z" que significa alta impedancia.

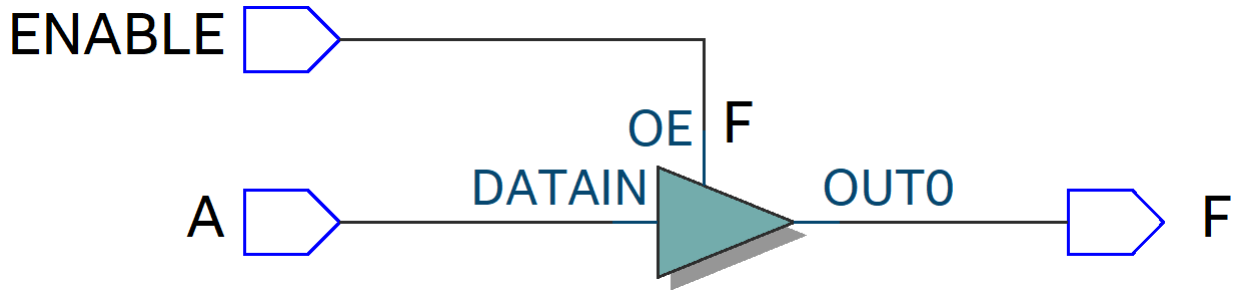

Figura 10: Diagrama RTL del buffer tri-estado.

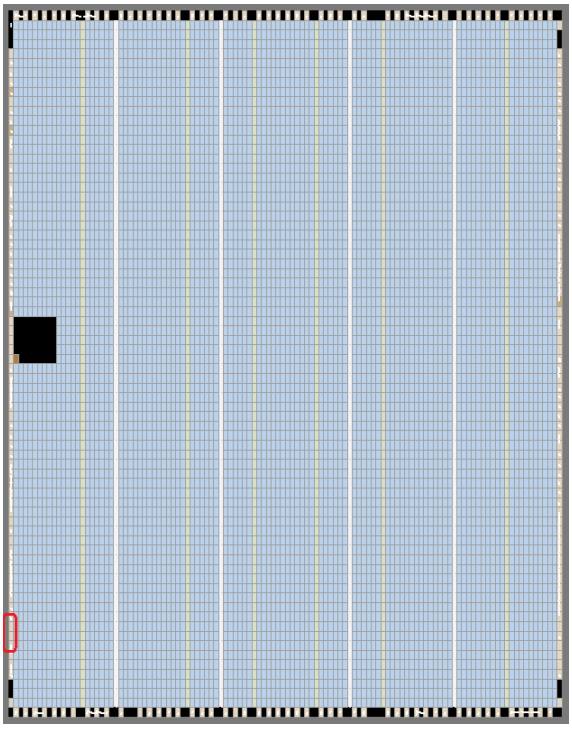

Figura 11: Vista de la herramienta de *Chip Planner* del buffer tri-estado. El buffer se observa dentro del circulo rojo.

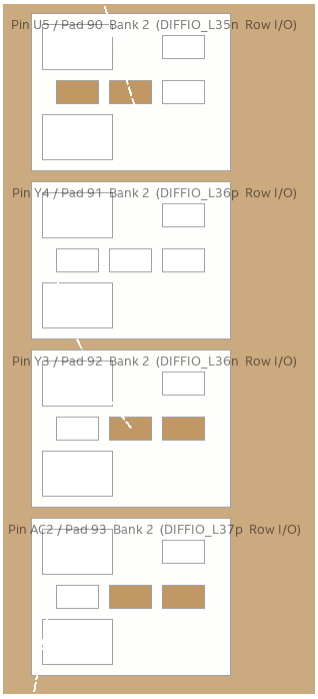

Figura 12: Acercamiento al modulo implementado en la línea de entradas y salidas.

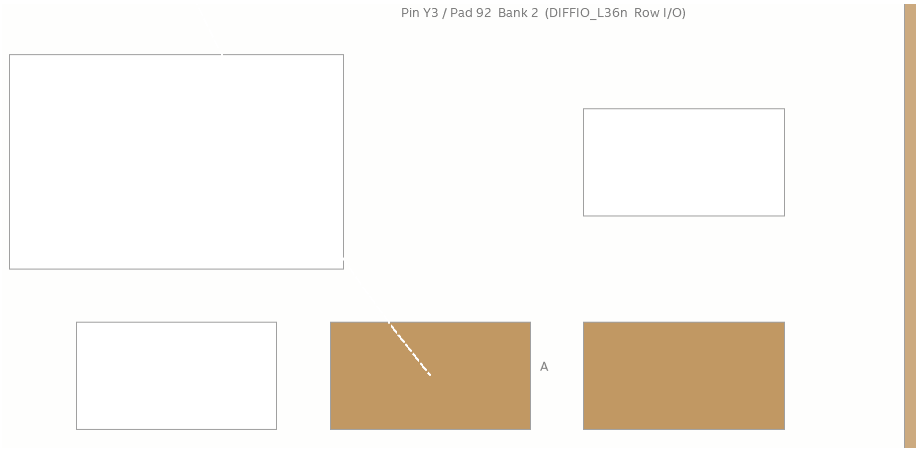

Figura 13: Vista del pin A.

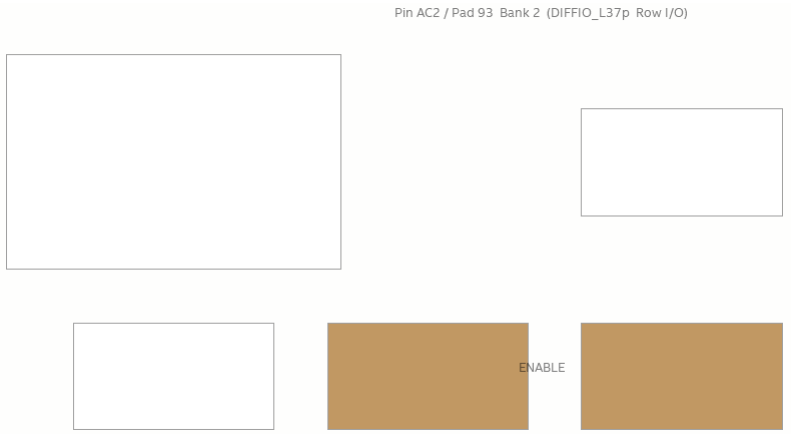

Figura 14: Vista del pin ENABLE.

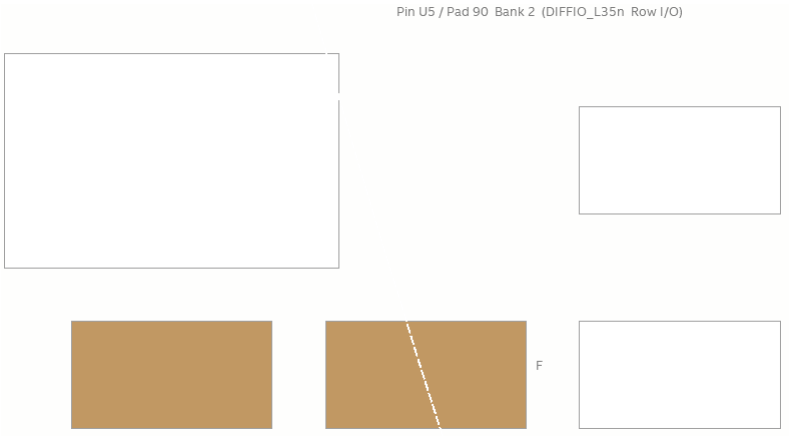

Figura 15: Vista del pin F.

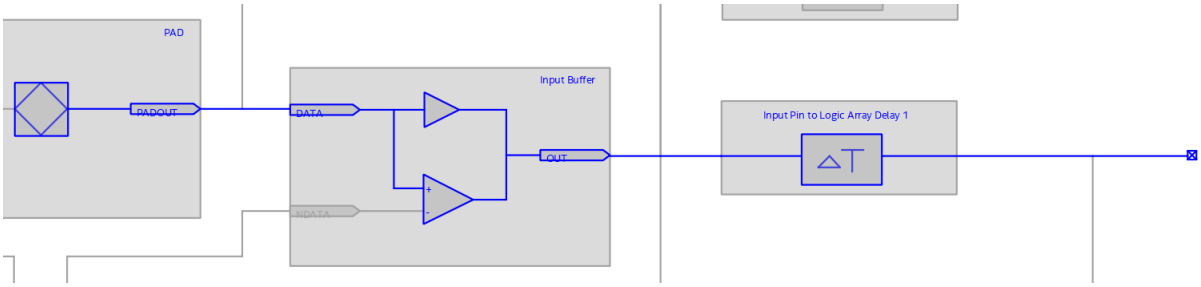

Figura 16: Vista del pin A implementado en el buffer.

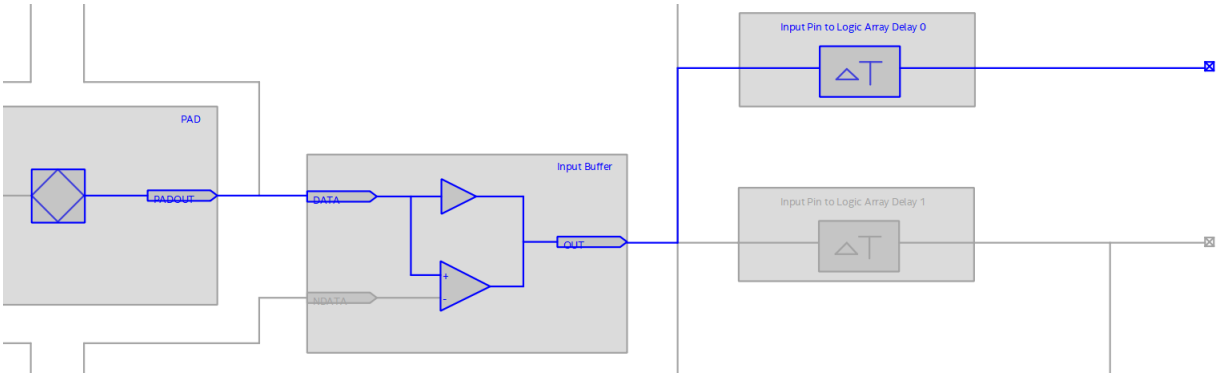

Figura 17: Vista del pin ENABLE implementado en el buffer.

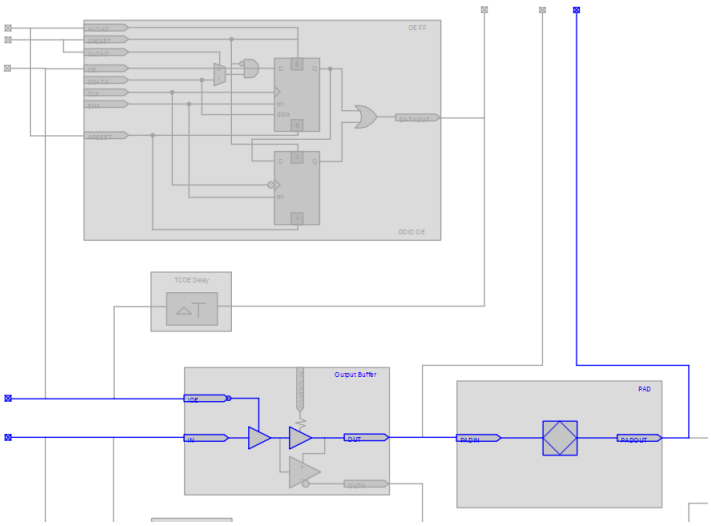

Figura 18: Vista del pin F implementado en el buffer.

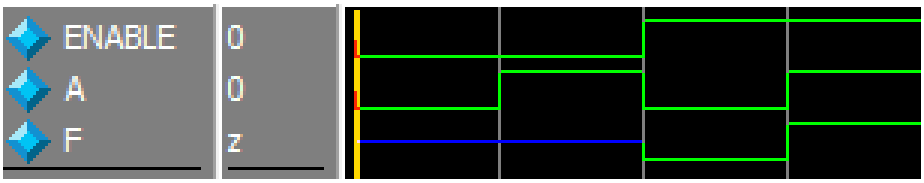

Figura 19: Simulación del buffer tri-estado con el visor de formas de onda de ModelSim.

6. Conclusiones

En conclusión, se implementaron los 4 circuitos en lenguaje Verilog de manera exitosa.

Para el desplazador lógico, se implementó utilizando concatenaciones en lugar de utilizar los operadores específicos "«" y "»", por lo que se entendió a profundidad el funcionamiento de este modulo.

Para el multiplexor, se implementó su descripción de dos formas, observando que con la estructura *case* se generó un multiplexor, mientras que con la estructura *if-else* se generaron otros elementos lógicos, como comparadores y multiplexores de menor tamaño.

Para la memoria ROM, se observó que este modulo se implementa de manera similar a un decodificador, diferenciándose en el tamaño, ya que la memoria puede tener cualquier tamaño.

Para el buffer tri-estado, con la herramienta de *Chip Planner*, se entendió como es que el entorno asignó el modulo al bloque de entradas y salidas, ya que en ese lugar se utilizan buffers para intercambiar el funcionamiento de los puertos entre entrada y salida.

En general, comprendió como es que operan dichos circuitos combinatorios y como es que se implementan en el visor RTL. Se comprobó su funcionamiento utilizando las simulaciones de forma de onda en ModelSim y se asignaron los pines correspondientes en la placa de desarrollo para programar el dispositivo y realizar las pruebas pertinentes en hardware.

En los Anexos se pueden encontrar los códigos implementados junto con sus respectivos bancos de pruebas.

7. Anexos

7.1. Descripciones del hardware

```
module Logical_Shifter(SEL, E, S);
input    [1:0]  SEL;
input    [7:0]  E;
output reg [7:0]  S;

always @(SEL, E)
 case (SEL)
  2'b00  : S = E;
  2'b01  : S = {E[6:0],1'b0};
  2'b10  : S = {E[5:0],2'b0};
  2'b11  : S = {E[4:0],3'b0};
  default : S = E;
 endcase

endmodule
```

Programa 1: Descripción en Verilog del desplazador lógico de 8 bits.

```
module Mux_4_1_Case(SEL, A, B, C, D, S);
input    [1:0]  SEL;
input       A, B, C, D;
output reg    S;

always @(SEL, A, B, C, D)
 case (SEL)
  2'b00  : S = A;
  2'b01  : S = B;
  2'b10  : S = C;
  2'b11  : S = D;
  default : S = A;
 endcase

endmodule
```

Programa 2: Descripción en Verilog del multiplexor 4 a 1, usando la estructura *case*.

```
module Mux_4_1_If(SEL, A, B, C, D, S);
 input    [1:0]  SEL;
 input       A, B, C, D;
 output reg    S;

```

```
 always @(*)
 begin
  if       (SEL == 2'b00)
   S = A;
  else if (SEL == 2'b01)
   S = B;
  else if (SEL == 2'b10)
   S = C;
  else if (SEL == 2'b11)
   S = D;
  else
   S = A;
 end

endmodule
```

Programa 3: Descripción en Verilog del multiplexor 4 a 1, usando la estructura *if*.

```
module ROM4x8(DIR, S);
input    [1:0]  DIR;
output reg [7:0]  S;

always @(DIR)
 case (DIR)
  2'b00  : S = 8'b11001100;
  2'b01  : S = 8'b10101010;
  2'b10  : S = 8'b11110000;
  2'b11  : S = 8'b00001111;
  default : S = 8'b0;
 endcase

endmodule
```

Programa 4: Descripción en Verilog de la memoria ROM de 4 localidades de 8 bits.

```
module TriState_Buffer(ENABLE, A, F);
input    ENABLE;
input    A;
output reg  F;

always @(ENABLE, A)
 if (ENABLE == 1)
  F = A;
 else
  F = 1'bz ;

```

```
12 endmodule
```

Programa 5: Descripción en Verilog del buffer tri-estado.

7.2. Bancos de pruebas (*Test Benches*)

```
‘timescale 1 ns/ 1 ps
module Logical_Shifter_vlg_tst();
 reg   [1:0]   SEL;
 reg   [7:0]   E;
 wire   [7:0]    S;

 Logical_Shifter i1 (
  .SEL(SEL),
  .E(E),
  .S(S)
 );

 initial
 begin
  SEL = 2’b00; E = 8’b11111111;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; SEL = 2’b01;
  #50; SEL = 2’b10;
  #50; SEL = 2’b11;
  #50; SEL = 2’b00; E = 8’b00001111;
  #50; SEL = 2’b01;
  #50; SEL = 2’b10;
  #50; SEL = 2’b11;
  #50; SEL = 2’b00; E = 8’b01010101;
  #50; SEL = 2’b01;
  #50; SEL = 2’b10;
  #50; SEL = 2’b11;
 end

endmodule
```

Programa 6: Banco de prueba para el Programa 1.

```
‘timescale 1 ns/ 1 ps
module Mux_4_1_Case_vlg_tst();
 reg   [1:0] SEL;
 reg     A;
 reg     B;
 reg     C;
 reg     D;
```

```
 wire    S;

 Mux_4_1_Case i1 (
  .SEL(SEL),
  .A(A),
  .B(B),
  .C(C),
  .D(D),
  .S(S)
 );

 initial
 begin
  SEL = 2'b00; A = 1; B = 0; C = 1; D = 0;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 0; B = 1; C = 0; D = 1;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 1; B = 1; C = 0; D = 0;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 0; B = 0; C = 1; D = 1;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
 end

endmodule
```

Programa 7: Banco de prueba para el Programa 2.

```
`timescale 1 ns/ 1 ps
module Mux_4_1_If_vlg_tst();
 reg  [1:0] SEL;
 reg    A;
 reg    B;
 reg    C;
```

```
 reg    D;
 wire    S;

 Mux_4_1_If i1 (
  .SEL(SEL),
  .A(A),
  .B(B),
  .C(C),
  .D(D),
  .S(S)
 );

 initial
 begin
  SEL = 2'b00; A = 1; B = 0; C = 1; D = 0;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 0; B = 1; C = 0; D = 1;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 1; B = 1; C = 0; D = 0;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
  #50; SEL = 2'b00; A = 0; B = 0; C = 1; D = 1;
  #50; SEL = 2'b01;
  #50; SEL = 2'b10;
  #50; SEL = 2'b11;
 end

endmodule
```

Programa 8: Banco de prueba para el Programa 3.

```
`timescale 1 ns/ 1 ps
module ROM4x8_vlg_tst();
 reg  [1:0]  DIR;
 wire  [7:0]  S;

```

```
 ROM4x8 i1 (
  .DIR(DIR),
  .S(S)
 );

 initial
 begin
  DIR = 2'b00;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; DIR = 2'b01;
  #50; DIR = 2'b10;
  #50; DIR = 2'b11;
 end

endmodule
```

Programa 9: Banco de prueba para el Programa 4.

```
`timescale 1 ns/ 1 ps
module TriState_Buffer_vlg_tst();
 reg   ENABLE;
 reg   A;
 wire   F;

 TriState_Buffer i1 (
  .ENABLE(ENABLE),
  .A(A),
  .F(F)
 );

 initial
 begin
  ENABLE = 0; A = 0;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; ENABLE = 0; A = 1;
  #50; ENABLE = 1; A = 0;
  #50; ENABLE = 1; A = 1;
 end
```

```
25
26 endmodule
```

Programa 10: Banco de prueba para el Programa 5.