Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Práctica 1 - Circuitos combinatorios 1

Profesor:
M. en C. Osvaldo Espinosa Sosa

Por:
Ing. Ricardo Aldair Tirado Torres

Ciudad de México, 21 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 multiplicador, el comparador de magnitud, el decodificador y el codificador.
- Entender como se configura una placa de desarrollo (específicamente el DE2-115), asignando interruptores como entradas y LED's como salidas, para observar los resultados.

2. Multiplicador binario

Actividad 1

Completar el código del multiplicador binario para números de 8 bits en el lenguaje de su elección. Compilar y simular. Usando el *Chip Planner*, verificar en donde se implementa el multiplicador por *default* (elementos lógicos o hardware dedicado).

La visualización RTL del multiplicador binario en Verilog se muestra en la Figura 1. Al momento de utilizar la herramienta de *Chip Planner* se observa en la Figura 2 que aparece un elemento resaltado de color gris, en la línea de multiplicadores dedicados, el cual corresponde con el multiplicador de 8 bits. Si se acerca la imagen a este elemento (ver Figura 3) se puede ver que el multiplicador fue implementado por la herramienta en hardware dedicado y esto se debe a que se identifico como un multiplicador, puesto que la descripción fue hecha por comportamiento, en caso de que se hubiera realizado por flujo de datos de bajo nivel, se habría implementado como elemento lógico. Finalmente, abriendo el contenido del modulo, se visualiza en la Figura 4 el interior del multiplicador.

Las simulaciones para el código en Verilog se visualizan en la Figura 5 en base binaria y en la Figura 6 en base decimal. Cabe resaltar que los valores utilizados para los bancos de pruebas fueron obtenidos de un generador de números aleatorios entre 0 y 255 [1].

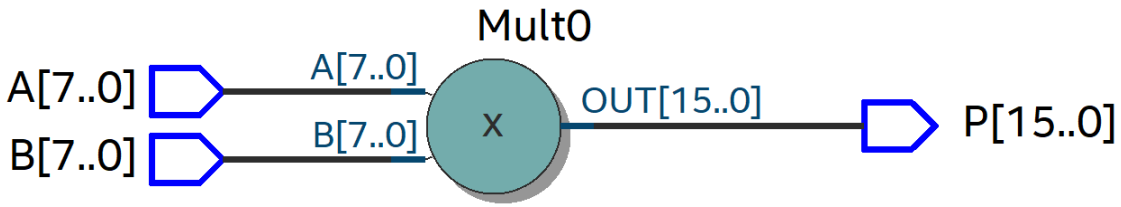

Figura 1: Diagrama RTL del multiplicador binario de 8 bits.

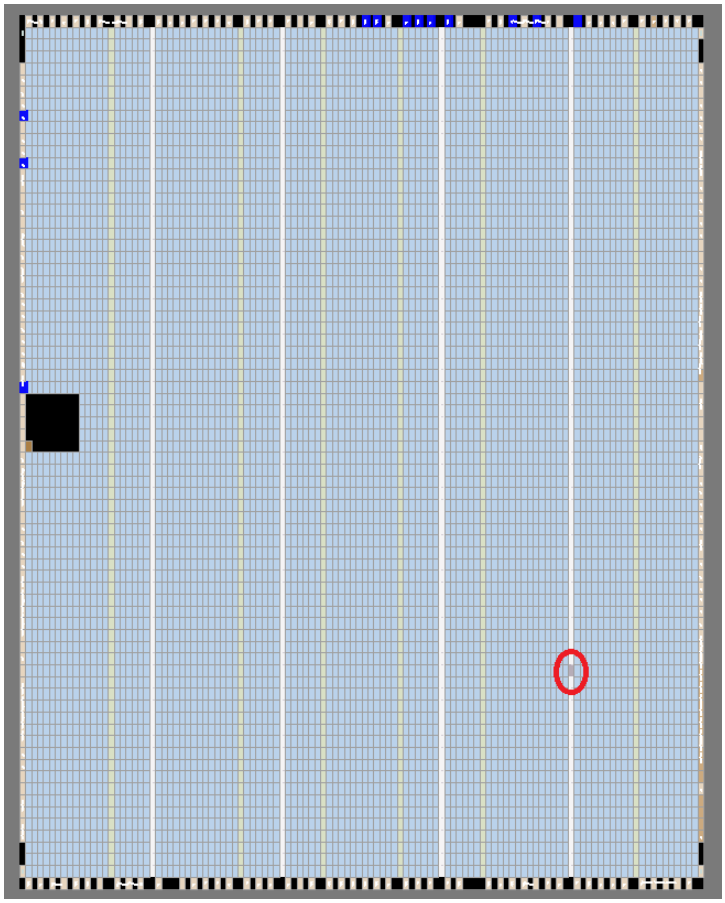

Figura 2: Vista de la herramienta de *Chip Planner* del multiplicador binario. El multiplicador se observa dentro del circulo rojo.

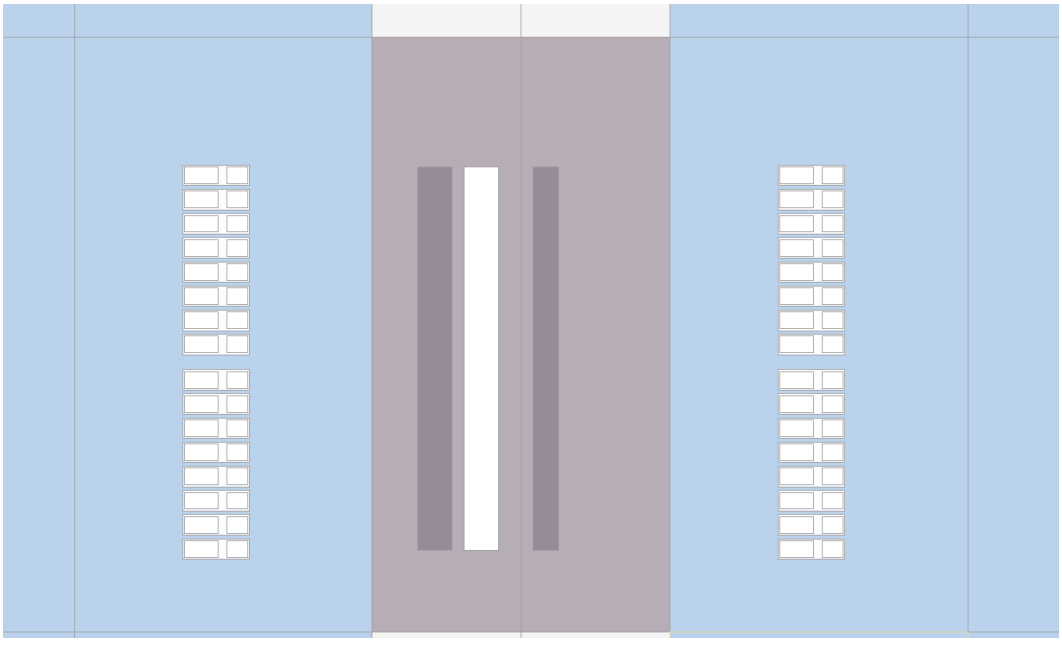

Figura 3: Acercamiento a la línea de multiplicadores dedicados.

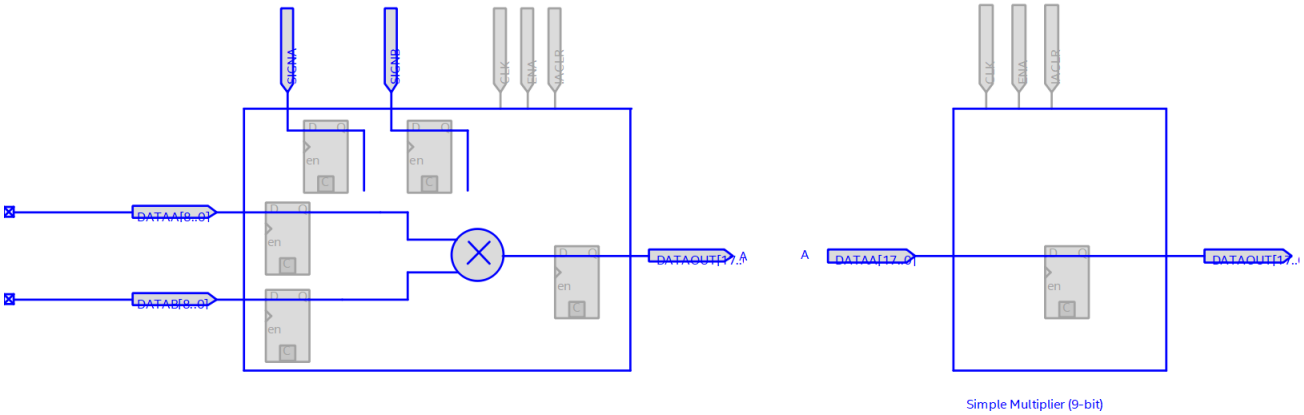

Figura 4: Vista del multiplicador binario.

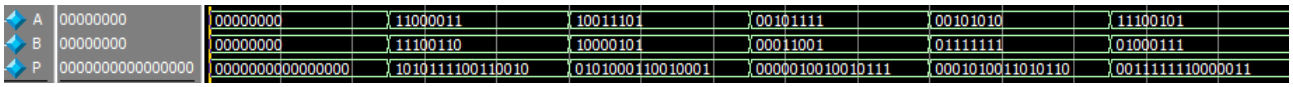

Figura 5: Simulación del multiplicador binario con el visor de formas de onda de ModelSim (Base binaria).

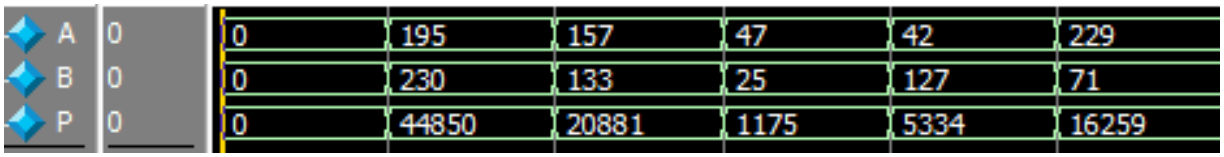

Figura 6: Simulación del multiplicador binario con el visor de formas de onda de ModelSim (Base decimal).

3. Comparador de magnitud

Actividad 2

Completar el código del comparador de magnitud en el lenguaje de su elección, para entradas de 4 bits. Compilar y simular. Configurar en la tarjeta DE2-115, asignar interruptores como entradas y un LED para observar la salida.

La visualización RTL del comparador de magnitud en Verilog se muestra en la Figura 7. Como se observa, la implementación del comparador de magnitud se hace utilizando un comparador de igualdad que pone la salida en alto unicamente cuando ambas entradas tienen el mismo valor. 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 las entradas para observar un comportamiento completo en la salida.

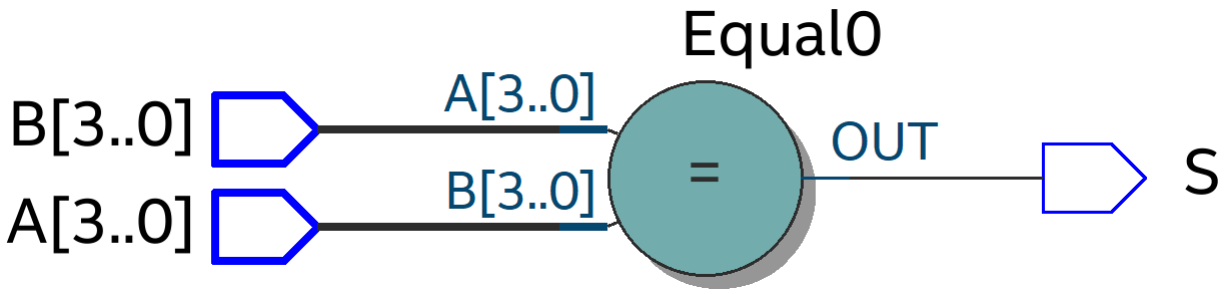

Figura 7: Diagrama RTL del comparador de magnitud.

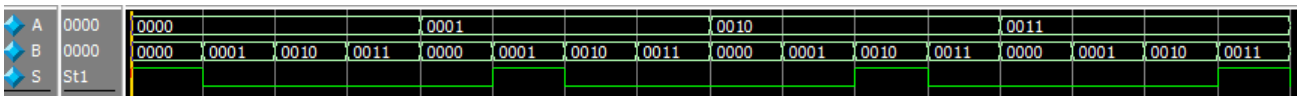

Figura 8: Simulación del comparador de magnitud con el visor de formas de onda de ModelSim (Base binaria).

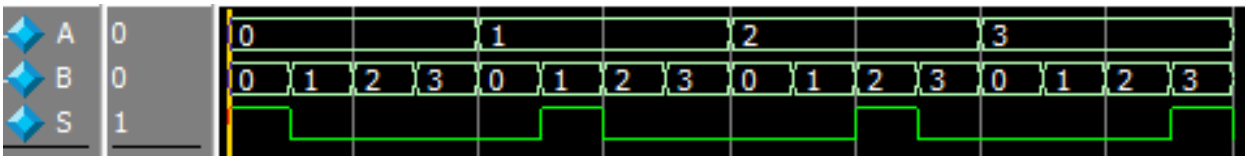

Figura 9: Simulación del comparador de magnitud con el visor de formas de onda de ModelSim (Base decimal).

4. Decodificador 2 a 4

Actividad 3

Completar el código del decodificador 2 a 4 en el lenguaje de su elección. Compilar y simular. Configurar en la tarjeta DE2-115, asignar interruptores como entradas y LED´s para observar la salida.

La visualización RTL del decodificador 2 a 4 en Verilog se muestra en la Figura 10. Como se observa, la implementación del decodificador 2 a 4 se hace utilizando una instancia de decodificador. Las simulaciones para el código en Verilog se visualizan en la Figura 11 en base binaria y en la Figura 12 en base decimal. Se utilizaron todos los valores posibles en las entradas para observar un comportamiento completo en la salida.

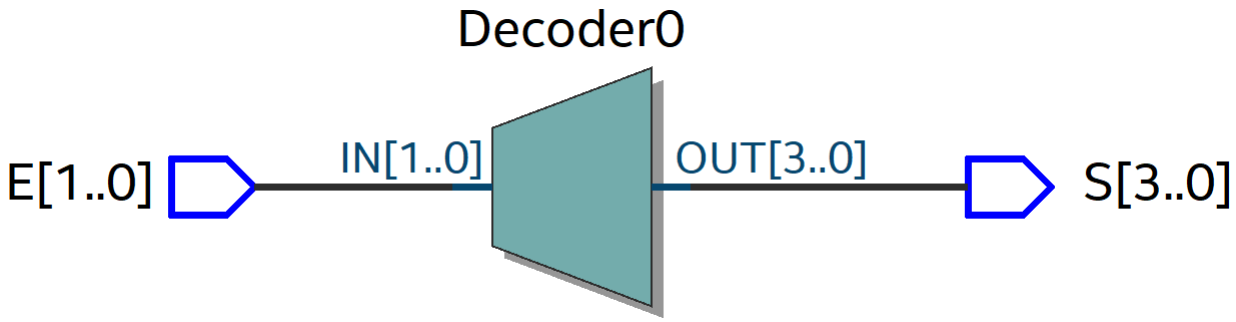

Figura 10: Diagrama RTL del decodificador 2 a 4.

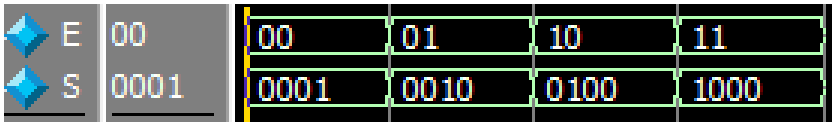

Figura 11: Simulación del decodificador 2 a 4 con el visor de formas de onda de ModelSim (Base binaria).

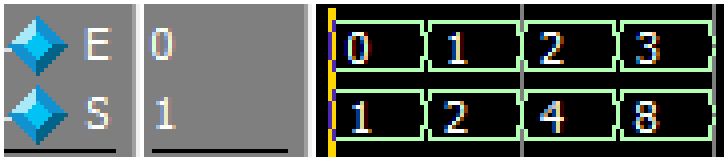

Figura 12: Simulación del decodificador 2 a 4 con el visor de formas de onda de ModelSim (Base decimal).

5. Codificador 4 a 2

Actividad 4

Completar el código del codificador o encoder 4 a 2 en el lenguaje de su elección. Compilar y simular. Configurar en la tarjeta DE2-115, asignar interruptores como entradas y LED´s para observar la salida.

La visualización RTL del codificador 4 a 2 en Verilog se muestra en la Figura 13. Como se observa, la implementación del codificador 4 a 2 se hace utilizando una instancia de decodificador junto con dos compuertas OR, conectadas de tal forma que corresponden a los dos bits de la salida. Las simulaciones para el código en Verilog se visualizan en la Figura 14 en base binaria y en la Figura 15 en base decimal. Se utilizaron todos los valores posibles en las entradas para observar un comportamiento completo en la salida.

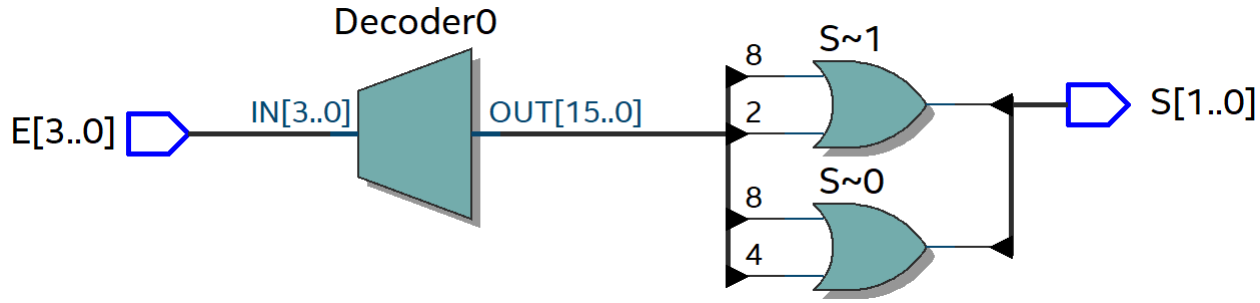

Figura 13: Diagrama RTL del codificador 4 a 2.

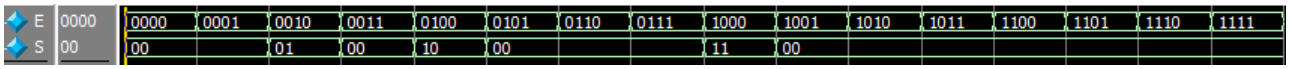

Figura 14: Simulación del codificador 4 a 2 con el visor de formas de onda de ModelSim (Base binaria).

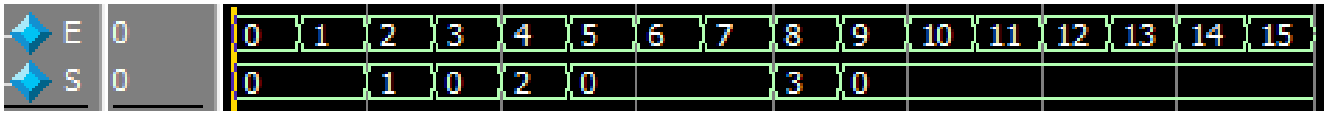

Figura 15: Simulación del codificador 4 a 2 con el visor de formas de onda de ModelSim (Base decimal).

6. Conclusiones

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

Para el multiplicador binario, se implementó la descripción por comportamiento, usando el operador aritmético "*" y con la herramienta de *Chip Planner* se entendió como es que el entorno asigno el circuito al hardware dedicado.

Para el comparador de magnitud, el decodificador y el codificador se 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.

Referencias

[1] PiliApp, “Generador de números aleatorios,” https://es.piliapp.com/random/number/.

7. Anexos

7.1. Descripciones del hardware

```
module Binary_Multiplier(
 input    [7:0]  A,
 input    [7:0]  B,
 output   [15:0] P
);

assign P = A * B;

endmodule
```

Programa 1: Descripción en Verilog del multiplicador binario de 8 bits.

```
module Magnitude_Comparator(
 input  [3:0]  A,
 input  [3:0]  B,
 output reg  S
);

always @(*)
   if (A == B)
      S = 1'b1;
   else
      S = 1'b0;
endmodule
```

Programa 2: Descripción en Verilog del comparador de magnitud.

```
module Decoder_2_4(
 input    [1:0]  E,
 output reg [3:0]  S
);

always @(E)
   case (E)
      2'b00  : S = 4'b0001;
      2'b01  : S = 4'b0010;
      2'b10  : S = 4'b0100;
```

```
11        2'b11   : S = 4'b1000;
12        default : S = 4'b0000;
13     endcase
14
15 endmodule
```

Programa 3: Descripción en Verilog del decodificador 2 a 4.

```
1  module Encoder_4_2(
2   input     [3:0]  E,
3   output reg [1:0]  S
4  );
5
6  always @(E)
7     case (E)
8        4'b0001   : S = 2'b00;
9        4'b0010   : S = 2'b01;
10       4'b0100   : S = 2'b10;
11       4'b1000   : S = 2'b11;
12       default : S = 2'b00;
13    endcase
14
15 endmodule
```

Programa 4: Descripción en Verilog del codificador 4 a 2.

7.2. Bancos de pruebas (*Test Benches*)

```
`timescale 1 ns/ 1 ps
module Binary_Multiplier_vlg_tst();
 reg   [7:0]   A;
 reg   [7:0]   B;
 wire   [15:0] P;

 Binary_Multiplier i1 (
  .A(A),
  .B(B),
  .P(P)
 );

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

 always
 begin
  #50; A = 195; B = 230; // 1100 0011 * 1110 0110 = 1010 1111 0011 0010
    (44 850)
  #50; A = 157; B = 133; // 1001 1101 * 1000 0101 = 0101 0001 1001 0001
    (20 881)
  #50; A = 47;  B = 25;  // 0010 1111 * 0001 1001 = 0000 0100 1001 0111 (
    1 175)
  #50; A = 42;  B = 127; // 0010 1010 * 0111 1111 = 0001 0100 1101 0110 (
    5 334)
  #50; A = 229; B = 71;  // 1110 0101 * 0100 0111 = 0011 1111 1000 0011
    (16 259)
 end

endmodule
```

Programa 5: Banco de prueba para el Programa 1.

```
`timescale 1 ns/ 1 ps
module Magnitude_Comparator_vlg_tst();
 reg   [3:0]   A;
 reg   [3:0]   B;
 wire      S;

 Magnitude_Comparator i1 (
  .A(A),
```

```
  .B(B),
  .S(S)
 );

 initial
 begin
  A = 0; B = 0;    // S = 1
  $display("Running testbench at CIC");
 end

 always
 begin
  #10; A = 0; B = 1;
  #10; A = 0; B = 2;
  #10; A = 0; B = 3;
  #10; A = 1; B = 0;
  #10; A = 1; B = 1; // S = 1
  #10; A = 1; B = 2;
  #10; A = 1; B = 3;
  #10; A = 2; B = 0;
  #10; A = 2; B = 1;
  #10; A = 2; B = 2; // S = 1
  #10; A = 2; B = 3;
  #10; A = 3; B = 0;
  #10; A = 3; B = 1;
  #10; A = 3; B = 2;
  #10; A = 3; B = 3; // S = 1
 end

endmodule
```

Programa 6: Banco de prueba para el Programa 2.

```
`timescale 1 ns/ 1 ps
module Decoder_2_4_vlg_tst();
 reg  [1:0]  E;
 wire [3:0]  S;

 Decoder_2_4 i1 (
  .E(E),
  .S(S)
 );

 initial
 begin
  E = 0;  // 0001
```

```
 $display("Running testbench at CIC");
 end

 always
 begin
  #50; E = 1; // 0010
  #50; E = 2; // 0100
  #50; E = 3; // 1000
 end

endmodule
```

Programa 7: Banco de prueba para el Programa 3.

```
`timescale 1 ns/ 1 ps
module Encoder_4_2_vlg_tst();
 reg [3:0]  E;
 wire [1:0]  S;

 Encoder_4_2 i1 (
  .E(E),
  .S(S)
 );

 initial
 begin
  E = 0;   //00
  $display("Running testbench at CIC");
 end

 always
 begin
  #10; E = 1;  //00
  #10; E = 2;  //01
  #10; E = 3;  //00
  #10; E = 4;  //10
  #10; E = 5;  //00
  #10; E = 6;  //00
  #10; E = 7;  //00
  #10; E = 8;  //11
  #10; E = 9;  //00
  #10; E = 10; //00
  #10; E = 11; //00
  #10; E = 12; //00
  #10; E = 13; //00
  #10; E = 14; //00
```

```
33   #10; E = 15; //00
34  end
35
36 endmodule
```

Programa 8: Banco de prueba para el Programa 4.