Proyecto final

Jostin Cubero Torres B92491
César Andrés Fonseca Villegas C12959
Marvin López Valverde C14324

Modelo Estructural Derecha a Izquierda

Explicacion de diseño

Para el caso de derecha a izquierda la cantidad mínima de estados necesarios para realizar el trayecto serían dos estados, debido a que en este caso a diferencia del sentido contrario no se necesita un estado exclusivo para que sea capaz de detectar la igualdad de las palabras, y debido a que de esta manera se está analizando cada palabra del bit menos significativo al más significativo, en este caso no existirían estados absorbentes ya que reiteradamente estaría comparando un bit cada vez más significativo que el anterior.

K) Actualmente la palabra A es menor o igual a la palabra B.

L) Actualmente la palabra A es mayor que la palabra B.

Explicacion de diseño

Seguidamente, se crearon las tablas de transición respectivas, lo que después se codificó después de la asignación de estados.

Tabla de transición sin codificar y codificada

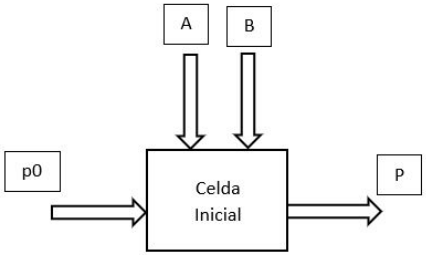
Estado Presente	Estado Futuro			
	AB			
	00	01	10	11
K	K	K	L	K
L	L	K	L	L

Estado Presente	Estado Futuro			
p	AB			
	00	01	10	11
0	0	0	1	0
1	1	0	1	1
	P			

Explicacion de diseño

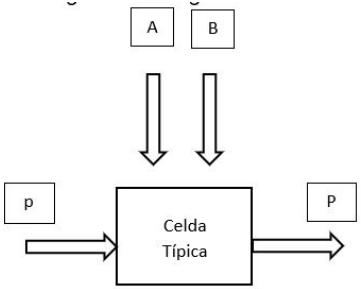
Después se resolvieron los MK para el diseño de la celda típica, celda inicial y la celda final, encontrando las funciones de los estados futuros, denotando el estado inicial para el cual la red iterativa debía empezar (tratando que sea el más neutro posible), y por último se diseñó la celda final, teniendo en cuenta que la salida Z se activaba en bajo, obteniendo su función.

Celda Inicial



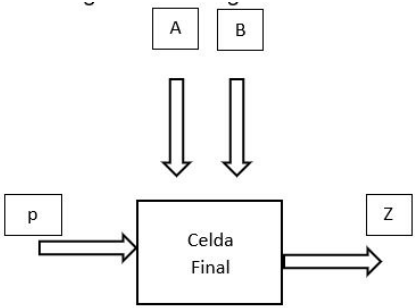
$$P(0,A,B) = AB`$$

Celda Típica



$$P(p,A,B) = AB` + pA + pB`$$

Celda Final



$$Z(p,A,B) = A`B + p`A` + p`B$$

Implementación

```
module CeldaInicialBit(output P_in,
input [2:0] A, B

);

    wire not_B;     //Cables intermedios

    and gate_and(P_in, not_B, A);
    not gate_not(not_B, B);

endmodule
```

```
module CeldaTipicaBit(output P_mid,
input [2:0] A, B,
input P_in, P
);

    wire not_B, and_or1, and_or2, and_or3 ;
    and gate_and1(and_or1, P_in);
    and gate_and2(and_or2, not_B, P);
    and gate_and3(and_or3, A, P);
    not gate_not1(not_B, B);
    or gate_or(P_mid, and_or1, and_or2, and_or3);

endmodule
```

```
module CeldaFinalBit(output Z_out,
input P_mid
);

    not not_gate(Z_out, P_mid);

endmodule
```

Implementación

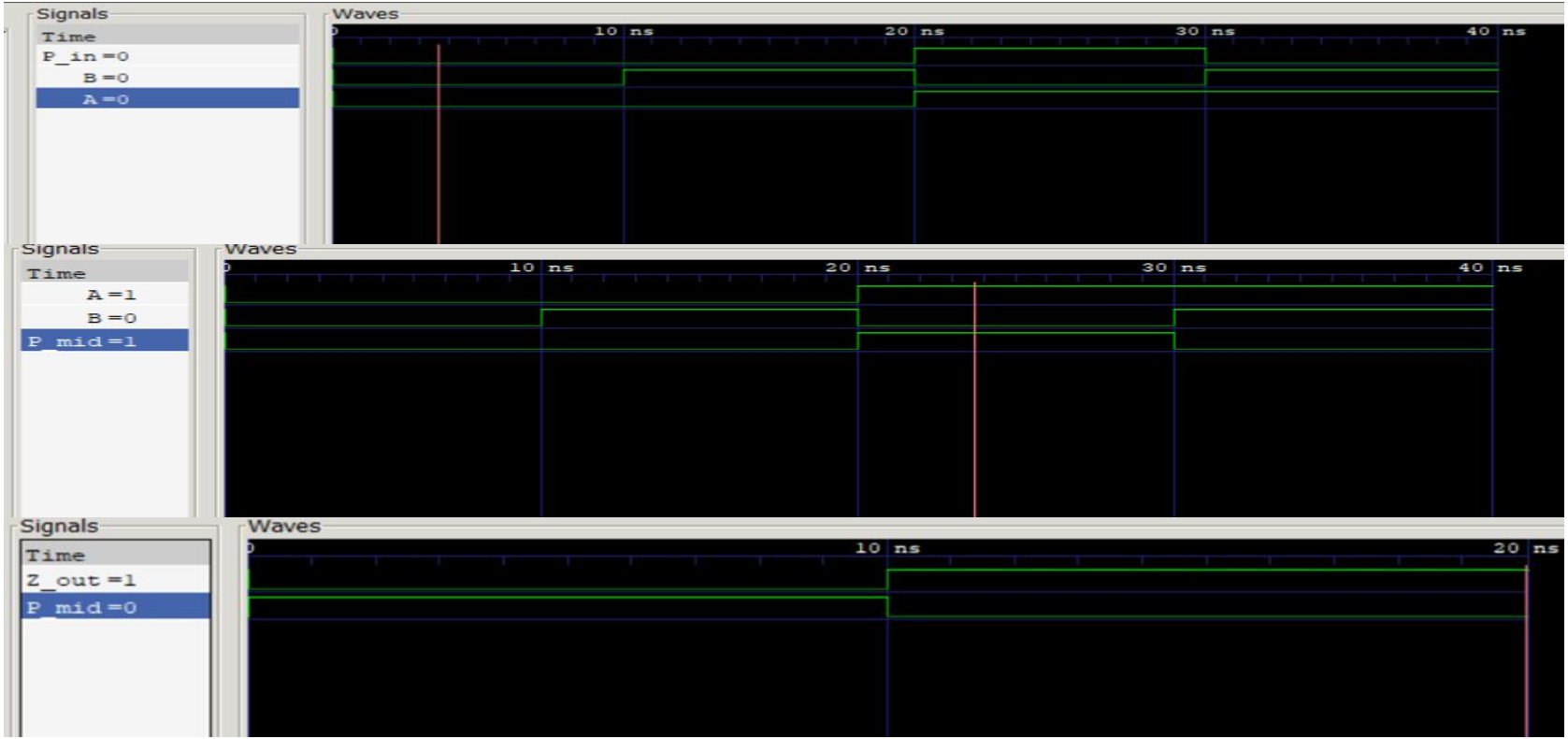
Cableado

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

    wire P_in; // Cable intermedio entre CeldaInicial y CeldaTipica
    wire P_mid; // Cable intermedio entre CeldaTipica y CeldaFinal

    // Instancias de las celdas
    CeldaInicialBit inicial(
        .P_in(P_in),
        .A(A),
        .B(B)
    );

    CeldaTipicaBit tipica(
        .P_mid(P_mid),
        .P(P),
        .A(A),
        .B(B),
        .P_in(P_in)
    );

    CeldaFinalBit final(
        .Z_out(Z_out),
        .P_mid(P_mid)
    );

endmodule
```

Patrón del generate

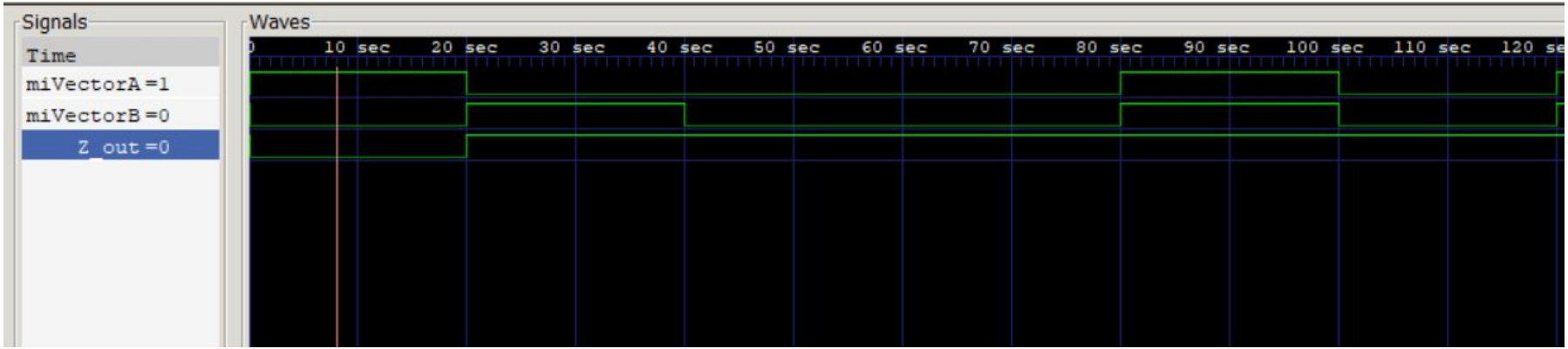
```
for (i = 0; i < 10; i = i + 1) begin
    $display("Prueba %0d", i);

    seed = $random; // Inicializar la semilla

    miVectorA = $random(seed) % (1 << N); // Generar número aleatorio entre 0 y (2^N - 1)
    miVectorB = $random(seed) % (1 << N); // Generar número aleatorio entre 0 y (2^N - 1)

    $display("miVectorA = %b, miVectorB = %b", miVectorA, miVectorB);

    if (miVectorA > miVectorB)
        P = 1;
    else
        P = 0;

    #period;

    $display("P = %0d, Z_out = %0d", P, Z_out);
end

$finish; // Finalizar la simulación
```

Pruebas Cada Celda para 1 bit



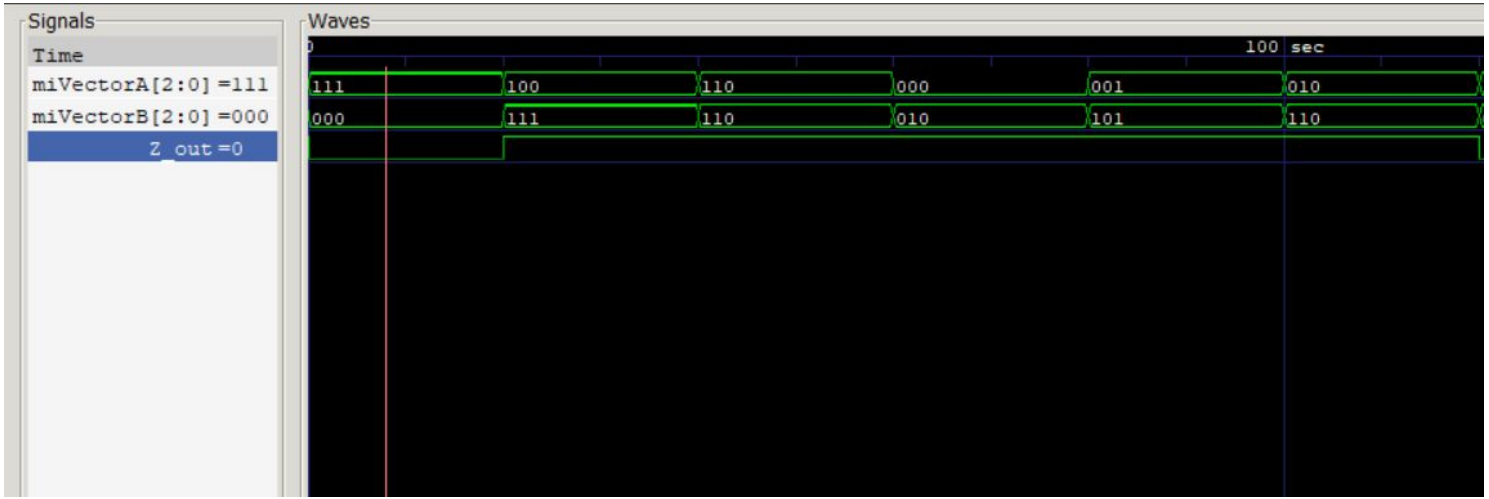
Prueba en el sistema completo 1bit

```
Prueba 0
miVectorA = 1, miVectorB = 0
P = 1, Z_out = 0
Prueba 1
miVectorA = 0, miVectorB = 1
P = 0, Z_out = 1
Prueba 2
miVectorA = 0, miVectorB = 0
P = 0, Z_out = 1
Prueba 3
miVectorA = 0, miVectorB = 0
P = 0, Z_out = 1
Prueba 4
miVectorA = 1, miVectorB = 1
P = 0, Z_out = 1
Prueba 5
miVectorA = 0, miVectorB = 0
P = 0, Z_out = 1
Prueba 6
miVectorA = 1, miVectorB = 1
P = 0, Z_out = 1
Prueba 7
miVectorA = 1, miVectorB = 1
P = 0, Z_out = 1
Prueba 8
miVectorA = 1, miVectorB = 0
P = 1, Z_out = 0
Prueba 9
```


Signals
Waves
Time
miVectorA =1
miVectorB =0
Z_out =0
10 sec
20 sec
30 sec
40 sec
50 sec
60 sec
70 sec
80 sec
90 sec
100 sec
110 sec
120 se

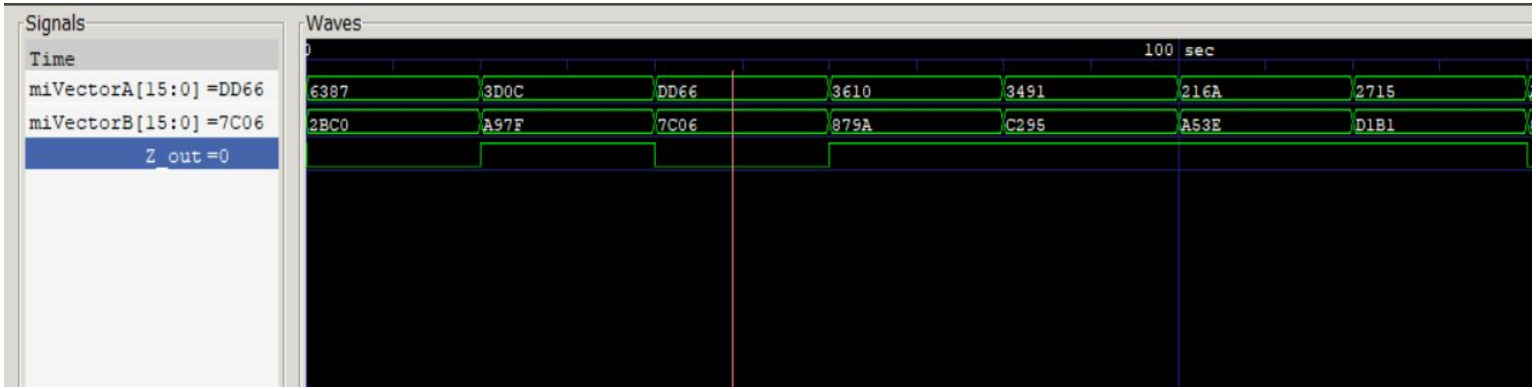
Prueba en el sistema completo 3 bits

```
Prueba 0
miVectorA = 111, miVectorB = 000
P = 1, Z_out = 0
Prueba 1
miVectorA = 100, miVectorB = 111
P = 0, Z_out = 1
Prueba 2
miVectorA = 110, miVectorB = 110
P = 0, Z_out = 1
Prueba 3
miVectorA = 000, miVectorB = 010
P = 0, Z_out = 1
Prueba 4
miVectorA = 001, miVectorB = 101
P = 0, Z_out = 1
Prueba 5
miVectorA = 010, miVectorB = 110
P = 0, Z_out = 1
Prueba 6
miVectorA = 101, miVectorB = 001
P = 1, Z_out = 0
Prueba 7
miVectorA = 101, miVectorB = 111
P = 0, Z_out = 1
Prueba 8
miVectorA = 001, miVectorB = 100
P = 0, Z_out = 0
Prueba 9
miVectorA = 010, miVectorB = 011
P = 0, Z_out = 1
```


Signals
Waves
Time
miVectorA[2:0] =111
miVectorB[2:0] =000
Z_out =0
100 sec
111
100
110
000
001
010
000
111
110
010
101
110

Prueba en el sistema completo 16 bits

```
Prueba 0
miVectorA = 0110001110000111, miVectorB = 0010101111000000
P = 1, Z_out = 0
Prueba 1
miVectorA = 0011110100001100, miVectorB = 1010100101111111
P = 0, Z_out = 1
Prueba 2
miVectorA = 1101110101100110, miVectorB = 0111110000000110
P = 1, Z_out = 0
Prueba 3
miVectorA = 0011011000010000, miVectorB = 1000011110011010
P = 0, Z_out = 1
Prueba 4
miVectorA = 0011010010010001, miVectorB = 1100001010010101
P = 0, Z_out = 1
Prueba 5
miVectorA = 0010000101101010, miVectorB = 1010010100111110
P = 0, Z_out = 1
Prueba 6
miVectorA = 0010011100010101, miVectorB = 1101000110110001
P = 0, Z_out = 1
Prueba 7
miVectorA = 1010010000000101, miVectorB = 1000000101001111
P = 1, Z_out = 0
```


Signals
Time
miVectorA[15:0] =DD66
miVectorB[15:0] =7C06
Z_out =0
Waves
100 sec
6387
3D0C
DD66
3610
3491
216A
2715
2BC0
A97F
7C06
879A
C295
A53E
D1B1

Modelo Conductual Izquierda a Derecha

Explicacion de diseño

Para el caso de izquierda a derecha la cantidad mínima de estados necesarios para realizar el trayecto serían tres estados, debido a que se necesita, un estado capaz de detectar la igualdad de las palabras, y debido a que de esta manera se está analizando cada palabra del bit más significativo al menos significativo, por lo que apenas se encuentre que en una comparación de bit se detecte que un bit es mayor o menor que otro, automáticamente este será un estado absorbente, debido a que la comparación de los siguientes bits menos significativos no tendrán importancia.

K) Actualmente la palabra A y la palabra B son iguales.

L) La palabra A es mayor que la palabra B.

M) La palabra A es menor que la palabra B.

Explicacion de diseño

Seguidamente, se crearon las tablas de transición respectivas, lo que después se codificó después de la asignación de estados.

Tabla de transición sin codificar y codificada

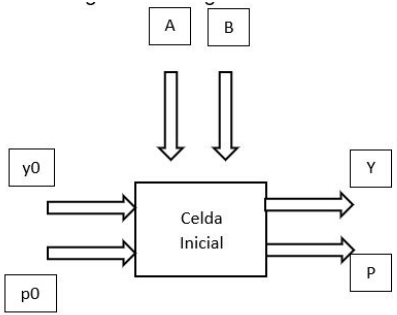
Estado Presente	Estado Futuro			
	AB			
	00	01	10	11
K	K	M	L	K
L	L	L	L	L
M	M	M	M	M

Estado Presente	Estado Futuro			
yp	AB			
	00	01	10	11
01	01	11	10	01
10	10	10	10	10
11	11	11	11	11
	YP			

Explicacion de diseño

Después se resolvieron los MK para el diseño de la celda típica, celda inicial y la celda final, encontrando las funciones de los estados futuros, denotando el estado inicial para el cual la red iterativa debía empezar (tratando que sea el más neutro posible), y por último se diseñó la celda final, teniendo en cuenta que la salida Z se activaba en bajo, obteniendo su función.

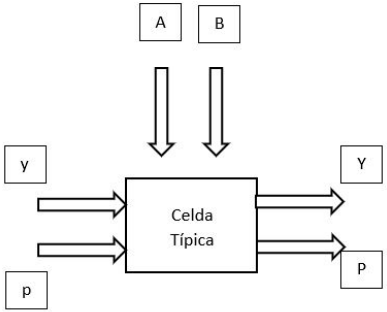
Celda Inicial



$$Y(0,1,A,B) = A`B + AB`$$

$$P(0,1,A,B) = A` + B$$

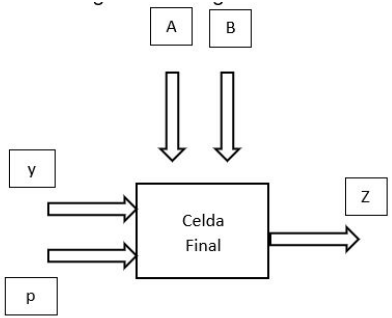
Celda Típica



$$Y(y,p,A,B) = y + A`B + AB`$$

$$P(y,p,A,B) = yp + pA` + pB$$

Celda Final



$$Z(y,p,A,B) = yp + pA` + pB$$

Implementación

```
1   module celda_inicial(
2       input [2:0] A, B,
3       output f_in, g_in
4   );
5
6   //todas las salidas de Z
7   assign f_in = (~A|B);
8
9   //todas las salidas de la funcion Y
10  assign g_in = ((~A|B) & (A|~B));
11
12  endmodule
```

```
1   module celda_tipica(
2       input z, y, g_in,
3       input [2:0] A, B,
4       output f_mid, g_mid
5   );
6
7   //todas las salidas de Z
8   assign f_mid = (((z | ~A) & (z | B)) & (y | z));
9
10  //todas las salidas de la funcion Y
11  assign g_mid = (y & g_in);
12
13  endmodule
```

```
3   module celda_final(
4       input f_mid,
5       output f
6   );
7
8   assign f = f_mid;
9
10  endmodule
```

Implementación

Cableado

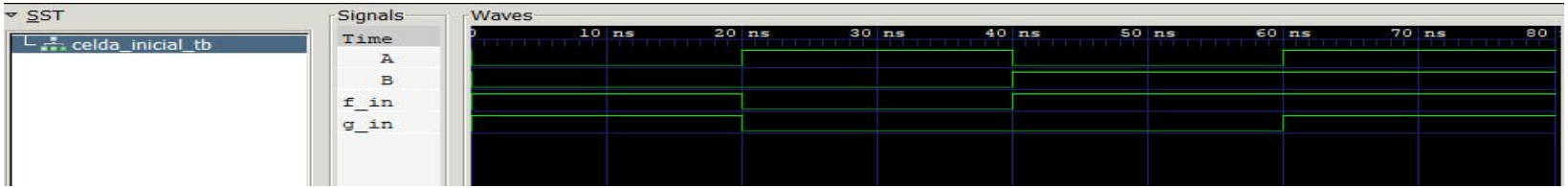
```
//MainModule
`include "celda_inicial.v"
`include "celda_tipica.v"
`include "celda_final.v"

module system(
    input [2:0] A, B,
    input z, y,
    output f
);

//conexiones generadas para el cableado de cada celda
wire f_in;
wire g_in;
wire f_mid;
wire g_mid;

//instanciacion de la celda inicial
celda_inicial C1 (
    .A(A),
    .B(B),
    .f_in(f_in),
    .g_in(g_in)
);

//instanciacion de la celda tipica
celda_tipica C2(
    .z(z),
    .y(y),
    .A(A),
    .B(B),
    .g_in(g_in),
    .f_mid(f_mid),
    .g_mid(g_mid)
);

//instanciacion de la celda final
celda_final C3(
    .f_mid(f_mid),
    .f(f)
);

endmodule
```

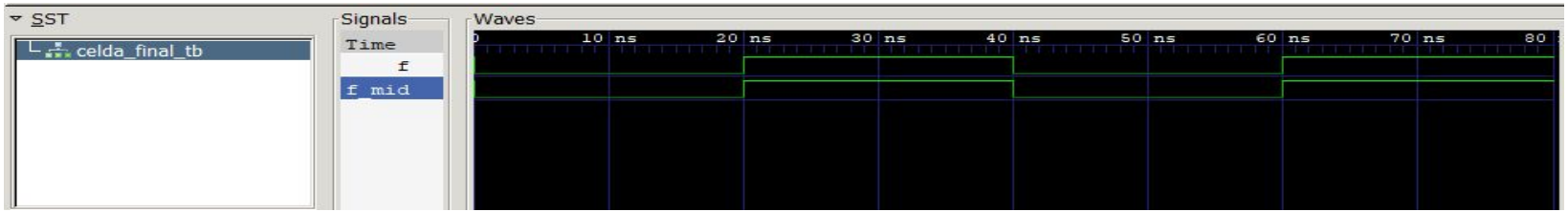
Patrón del Generate

```
// bucle para comparar los bits de los vectores
  for (n = 15; n >= 0; n = n - 1) begin
    // Compara los términos correspondientes de los vectores
    if (miVectorA[n] == miVectorB[n]) begin
      $display("Por el momento las dos palabras son iguales");
      y = 0;
      z = 1;
      #period;
    end else if (miVectorA[n] > miVectorB[n]) begin
      $display("La palabra A es mayor a la palabra B");
      y = 1;
      z = 0;
      #period;
      $finish;
    end else begin
      $display("La palabra B es mayor a la palabra A");
      y = 1;
      z = 1;
      #period;
      $finish;
    end

    $display("Para n = %0d,  y = %0d, z = %0d, f = %0d", n, y, z, f);
    #period;
  end
```

Pruebas (César)

prueba de 1-bit en celdas

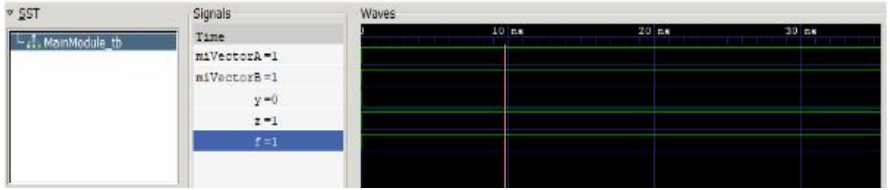




Pruebas en el sistema Completo

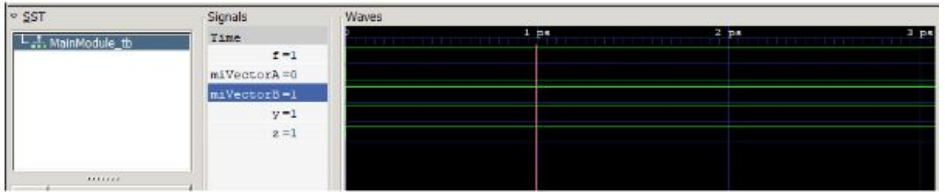
caso vectores iguales:

```
Por el momento las dos palabras son iguales
Para y = 0, z = 1, f = 1
```



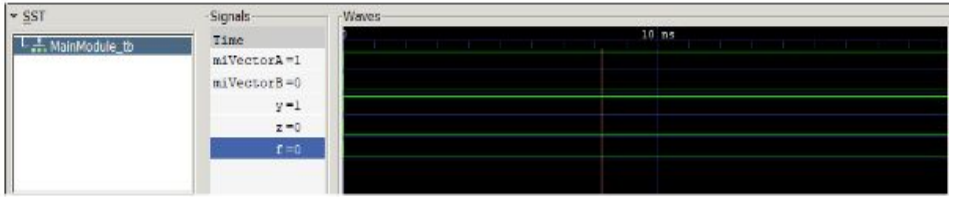
caso B > A:

```
VCD info: dumpfile MainModule_tb.vcd opened for output.
La palabra B es mayor a la palabra A
```



caso A > B:

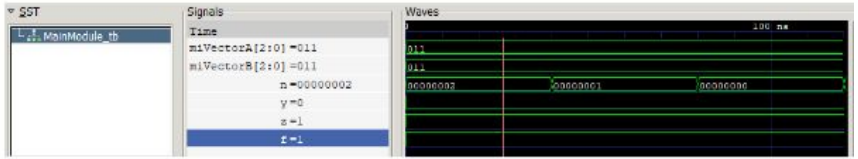
```
VCD info: dumpfile MainModule_tb.vcd opened for output.
La palabra A es mayor a la palabra B
```



Pruebas en el sistema Completo

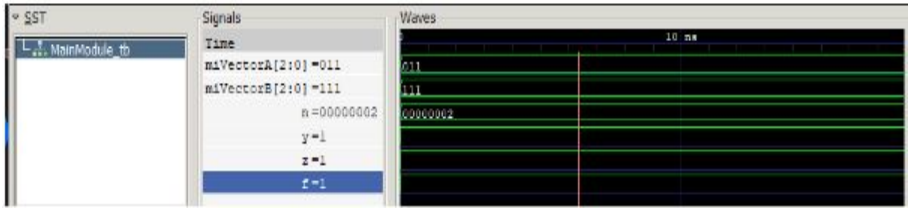
caso vectores iguales: (011 y 011)

```
VCD info: dumpfile MainModule_tb.vcd opened for output.
Por el momento las dos palabras son iguales
Para n = 2,  y = 0, z = 1, f = 1
Por el momento las dos palabras son iguales
Para n = 1,  y = 0, z = 1, f = 1
Por el momento las dos palabras son iguales
Para n = 0,  y = 0, z = 1, f = 1
```



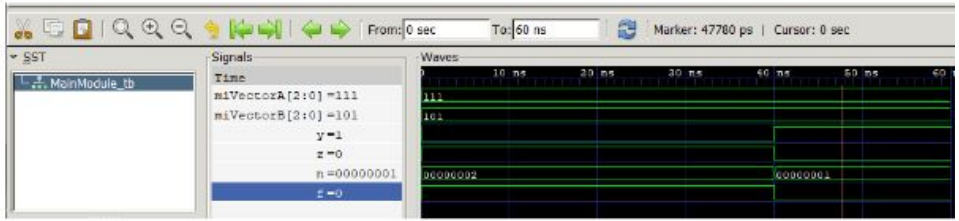
caso B > A: (B=111 y A=011)

```
VCD info: dumpfile MainModule_tb.vcd opened for output.
La palabra B es mayor a la palabra A
```



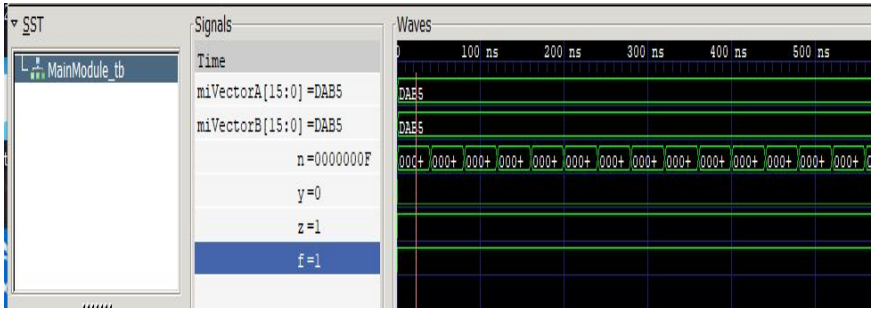
caso A > B: (A=111 y B=101)

```
Para n = 2,  y = 0, z = 1, f = 1
La palabra A es mayor a la palabra B
```

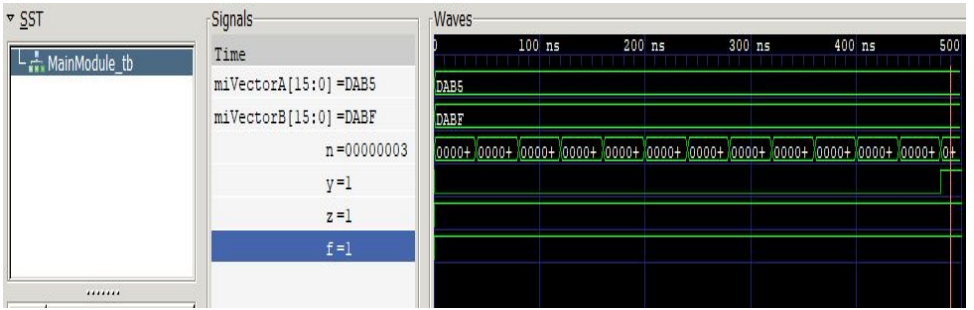


Pruebas en el sistema Completo

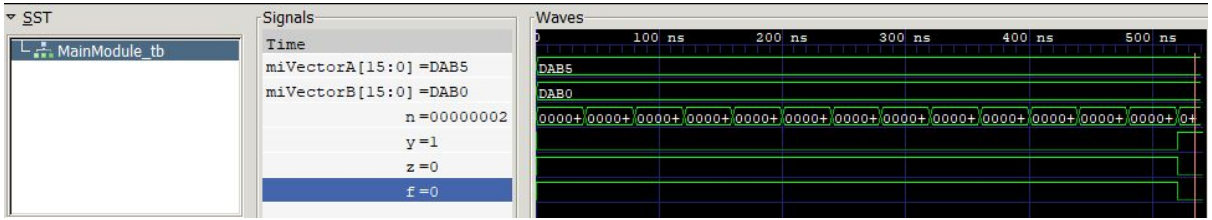
caso vectores iguales: (16'b1101101010110101)



caso B > A: (B=16'b1101101010111111 y A=16'b1101101010110101):



caso A > B: (A=16'b1101101010110101 y B=16'b1101101010110000)



Muchas gracias