



Instituto Politécnico Nacional
Escuela Superior de Cómputo



Arquitectura de Computadoras

“Multiplexores en HDL (Verilog)”

Alumno:

Malagón Baeza Alan Adrian

Profesor:

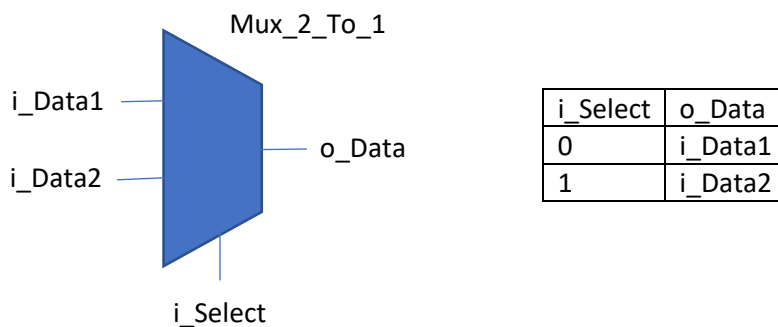
Alemán Arce Miguel Ángel

Grupo: 5CV1

Introducción

Un multiplexor es un circuito que pasa una de sus múltiples entradas a una sola salida en función de una entrada seleccionada. Esto se puede considerar como un interruptor digital. El multiplexor tiene n líneas de selección, 2^n entradas y una salida. El siguiente ejemplo muestra el proceso de diseño manual de un multiplexor 2 a 1 (es decir, utilizando el enfoque de diseño digital clásico).

Ejemplo: Multiplexor 2 a 1 - Síntesis lógica manual

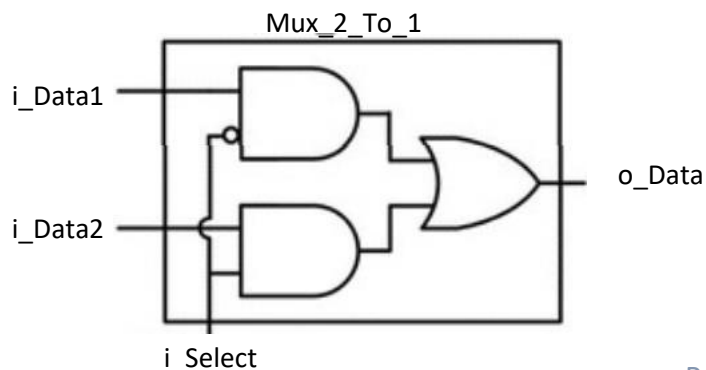


Para diseñar el multiplexor, es útil enumerar todos los valores posibles para i_Data1 , i_Data2 e i_Select en forma de tabla de verdad

	i_Select	i_Data1	i_Data2	o_Data
Cuando $i_Select=0$, la salida es i_Data1	0	0	0	0
	0	0	1	0
	0	1	0	1
	0	1	1	1
Cuando $i_Select=1$, la salida es i_Data2	1	0	0	0
	1	0	1	1
	1	1	0	0
	1	1	1	1

i_Select	i_Data1			
i_Data2	00	01	11	10
0	0	1	0	0
1	0	1	1	1

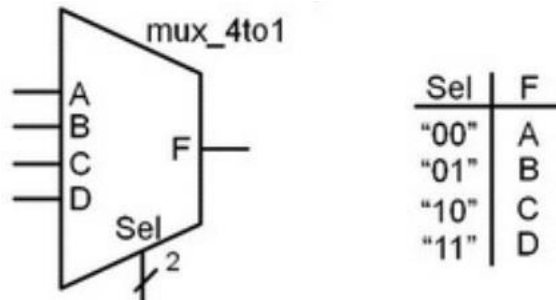
→ $o_Data = i_Select' \cdot i_Data1 + i_Select \cdot i_Data2$



En Verilog, se puede implementar un multiplexor mediante asignación continua con operadores lógicos o condicionales. El siguiente ejemplo muestra cómo modelar el multiplexor en Verilog utilizando estas técnicas.

Ejemplo: Multiplexor 4 a 1 - Modelado de Verilog mediante Asignación Continua

El símbolo y la tabla de verdad para el multiplexor 4 a 1 son los siguientes:



Las siguientes son dos formas diferentes de implementar el comportamiento del multiplexor con asignación continua: (1) con operadores lógicos; y (2) con operador condicional.

(1)

```
module mux_4to1 (output wire F,
                 input wire A, B, C, D,
                 input wire [1:0] Sel);

    assign F = (A & ~Sel[1] & ~Sel[0]) |
               (B & ~Sel[1] & Sel[0]) |
               (C & Sel[1] & ~Sel[0]) |
               (D & Sel[1] & Sel[0]);

endmodule
```

(2)

```
module mux_4to1 (output wire F,
                 input wire A, B, C, D,
                 input wire [1:0] Sel);

    assign F = (Sel == 2'b00) ? A :
               (Sel == 2'b01) ? B :
               (Sel == 2'b10) ? C :
               (Sel == 2'b11) ? D :
               1'bX;

endmodule
```

Desarrollo

Propuesta 1

Código Verilog

Mux_2_to_1.v

C:/Users/alanm/Desktop/Arqui/Vivado/Multiplexor/Multiplexor.srcs/sources_1/new/Mux_2_to_1.v



```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Module Name: Mux_2_To_1
9  // Target Devices: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module Mux_2_To_1 (input  i_Select,
14                   input  i_Data1,
15                   input  i_Data2,
16                   output o_Data);
17
18     assign o_Data = i_Select ? i_Data1 : i_Data2;
19
20 endmodule // Mux_2_To_1
```

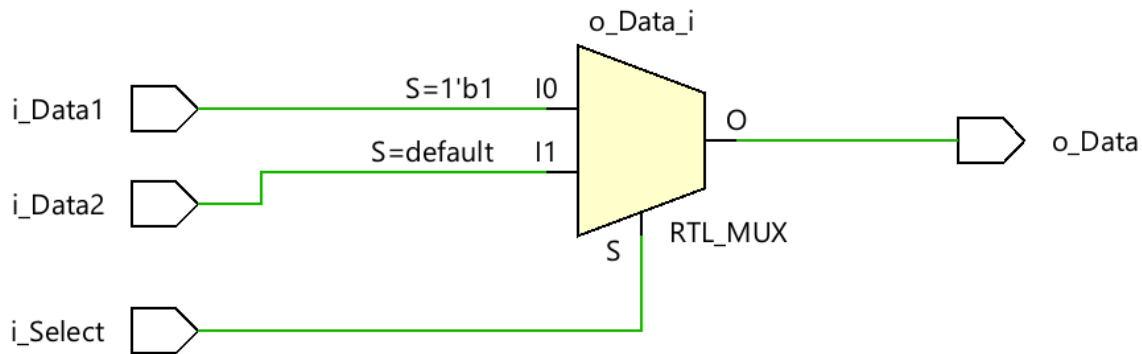
Código Verilog Testbench

```
tb_Mux_2_To_1.v

C:/Users/alanm/Desktop/Arqui/Vivado/Multiplexor/Multiplexor.srscs/sim_1/new/tb_Mux_2_To_1.v

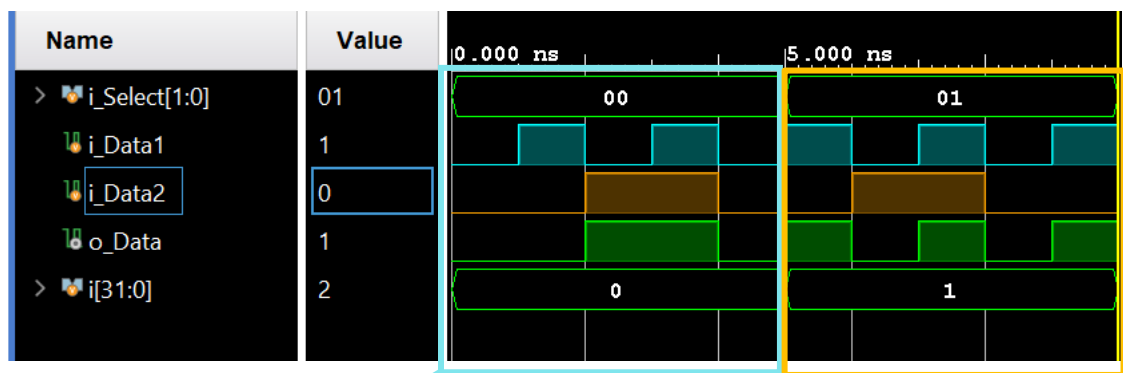
1  `timescale 1ns / 1ps
2  //////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  // Module Name: tb_Mux_2_To_1
8  // Target Devices: Artix
9  //
10 //////////////////////////////////////
11
12 // Modulo de estimulo
13 module tb_Mux_2_To_1();
14
15 //Inputs
16 reg [1:0] i_Select;
17 reg i_Data1;
18 reg i_Data2;
19
20 //Outputs
21 wire o_Data;
22
23 integer i;
24
25 //Instantiation of Unit Under Tes
26 Mux_2_To_1 mux0(
27     .i_Select (i_Select),
28     .i_Data1 (i_Data1),
29     .i_Data2 (i_Data2),
30     .o_Data (o_Data)
31 );
32
33 initial i_Data1 = 1'b0;
34 always #1 i_Data1 = ~i_Data1;
35
36 initial i_Data2 = 1'b0;
37 always #2 i_Data2 = ~i_Data2;
38
39 initial
40 begin
41     for(i=0;i<2;i=i+1) begin
42         i_Select = i;
43         #5;
44     end
45     $finish;
46 end
47 endmodule
```

Implementación RTL en Vivado 2022.2



Podemos observar cómo utilizó un multiplexor para la implementación del multiplexor 2 a 1.

Resultado de la simulación:



Cuando $i_Select = 00$, la salida o_Data es i_Data2

Cuando $i_Select = 10$, la salida o_Data es i_Data1

Como se observa en la simulación, la salida o_Data recibe el valor de entrada i_DataX según la expresión asignada en el selector i_Select .

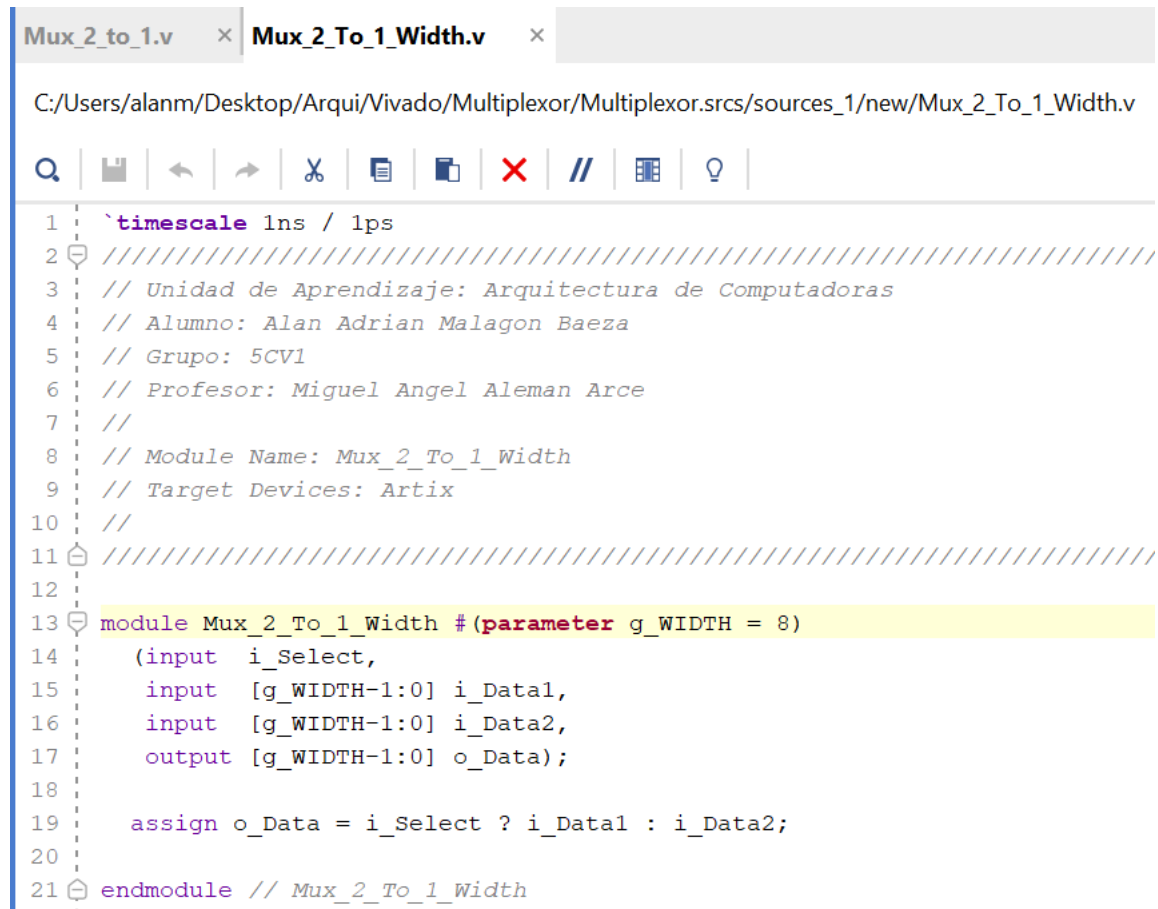
$o_Data = i_Select ? i_Data1 : i_Data2;$

$o_Data = 00 ? i_Data1 \text{ (true)} : i_Data2 \text{ (false)}; \quad 00 \rightarrow \text{false}$

$o_Data = i_Data2$

Propuesta 2, Parametrización

Código Verilog



The screenshot shows a Verilog code editor with two tabs: 'Mux_2_to_1.v' and 'Mux_2_To_1_Width.v'. The active tab is 'Mux_2_To_1_Width.v', which contains the following code:

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Module Name: Mux_2_To_1_Width
9  // Target Devices: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module Mux_2_To_1_Width #(parameter g_WIDTH = 8)
14     (input  i_Select,
15      input  [g_WIDTH-1:0] i_Data1,
16      input  [g_WIDTH-1:0] i_Data2,
17      output [g_WIDTH-1:0] o_Data);
18
19     assign o_Data = i_Select ? i_Data1 : i_Data2;
20
21 endmodule // Mux_2_To_1_Width
```

Código Verilog Testbench

tb_Mux_2_To_1_Width.v

C:/Users/alanm/Desktop/Arqui/Vivado/Multiplexor/Multiplexor.srscs/sim_1/new/tb_Mux_2_To_1_Width.v

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  // Module Name: tb_Mux_2_To_1_Width
8  // Target Devices: Artix
9  //
10 ///////////////////////////////////////////////////////////////////
11
12 // Modulo de estimulo
13 module tb_Mux_2_To_1_Width();
14
15     localparam g_WIDTH = 8;
16     //Inputs
17     reg [1:0] i_Select;
18     reg [g_WIDTH-1:0] i_Data1;
19     reg [g_WIDTH-1:0] i_Data2;
20
21     //Outputs
22     wire [g_WIDTH-1:0] o_Data;
23
24     integer i;
25
26     Mux_2_To_1_Width #(
27         .g_WIDTH(g_WIDTH)
28     ) MUX0 (
29         .i_Select (i_Select),
30         .i_Data1 (i_Data1),
31         .i_Data2 (i_Data2),
32         .o_Data (o_Data)
33     );
34
35     initial i_Data1 = 1'b0;
36     always #1 i_Data1 = ~i_Data1;
37
38     initial i_Data2 = 1'b0;
39     always #2 i_Data2 = ~i_Data2;
40
41     initial
42     begin
43         i_Select <= 0;
44         i_Data1 <= $random;
45         i_Data2 <= $random;
```

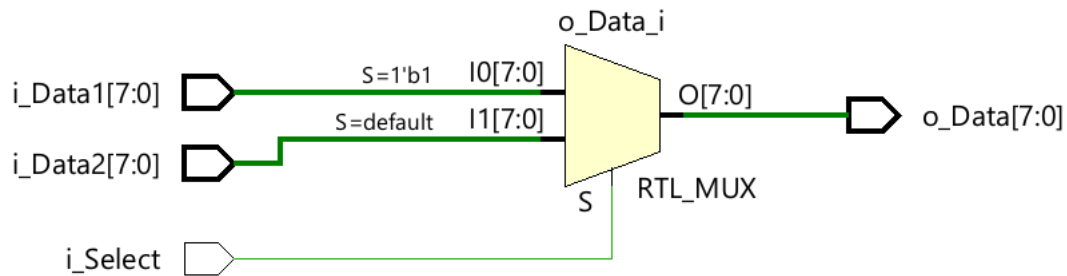


```

46     for(i=0;i<2;i=i+1) begin
47         #5;
48         i_Select = i;
49     end
50     #5;
51     $finish;
52 end
53 endmodule

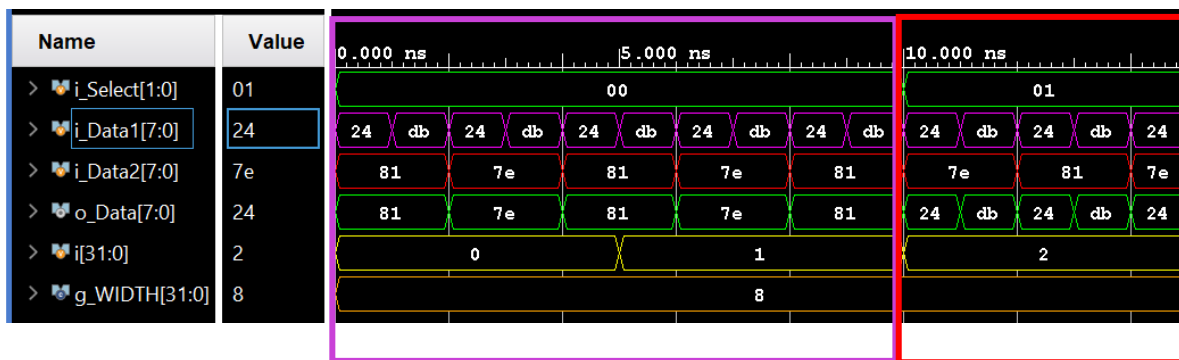
```

Descripción RTL en Vivado 2022.2



Podemos observar cómo utilizó un multiplexor para la implementación del multiplexor 2 a 1.

Resultado de la simulación:



Cuando $i_Select = 00$, la salida o_Data es i_Data2

Cuando $i_Select = 01$, la salida o_Data es i_Data1

Como se observa en la simulación, la salida o_Data recibe el valor de entrada i_DataX según la expresión asignada en el selector i_Select .

$o_Data = i_Select ? i_Data1 : i_Data2;$

$o_Data = 00 ? i_Data1 \text{ (true)} : i_Data2 \text{ (false)};$ $00 \rightarrow \text{false}$

$o_Data = i_Data2$

Propuesta 3

Código Verilog

```
Mux_4_To_1.v x tb_Mux_4_To_1.v x tb_Mux_4_To_1_behav.wcfg x
C:/Users/alanm/Desktop/Arqui/Vivado/Multiplexor/Multiplexor.srscs/sources_1/new/Mux_4_To_1.v

1 timescale 1ns / 1ps
2 //////////////////////////////////////////////////
3 // Unidad de Aprendizaje: Arquitectura de Computadoras
4 // Alumno: Alan Adrian Malagon Baeza
5 // Grupo: 5CV1
6 // Profesor: Miguel Angel Aleman Arce
7 //
8 // Module Name: Mux_4_To_1
9 // Target Devices: Artix
10 //
11 //////////////////////////////////////////////////
12
13 module Mux_4_To_1 (input [1:0] i_Select,
14                   input i_Data1,
15                   input i_Data2,
16                   input i_Data3,
17                   input i_Data4,
18                   output o_Data);
19
20     reg r_Data;
21
22     assign o_Data = i_Select[1] ? (i_Select[0] ? i_Data4 : i_Data3) :
23                          (i_Select[0] ? i_Data2 : i_Data1);
24
25
26     // Alternatively:
27     always @(*)
28     begin
29         case (i_Select)
30             2'b00 : r_Data <= i_Data1;
31             2'b01 : r_Data <= i_Data2;
32             2'b10 : r_Data <= i_Data3;
33             2'b11 : r_Data <= i_Data4;
34         endcase // case (i_Select)
35     end
36
37     assign o_Data = r_Data;
38
39 endmodule // Mux_4_To_1
```

Código Verilog Testbench (Con función \$random)

```
Mux_4_To_1.v x tb_Mux_4_To_1.v x tb_Mux_4_To_1_behav.wcfg x
C:/Users/alanm/Desktop/Arqui/Vivado/Multiplexor/Multiplexor.srscs/sim_1/new/tb_Mux_4_To_1.v

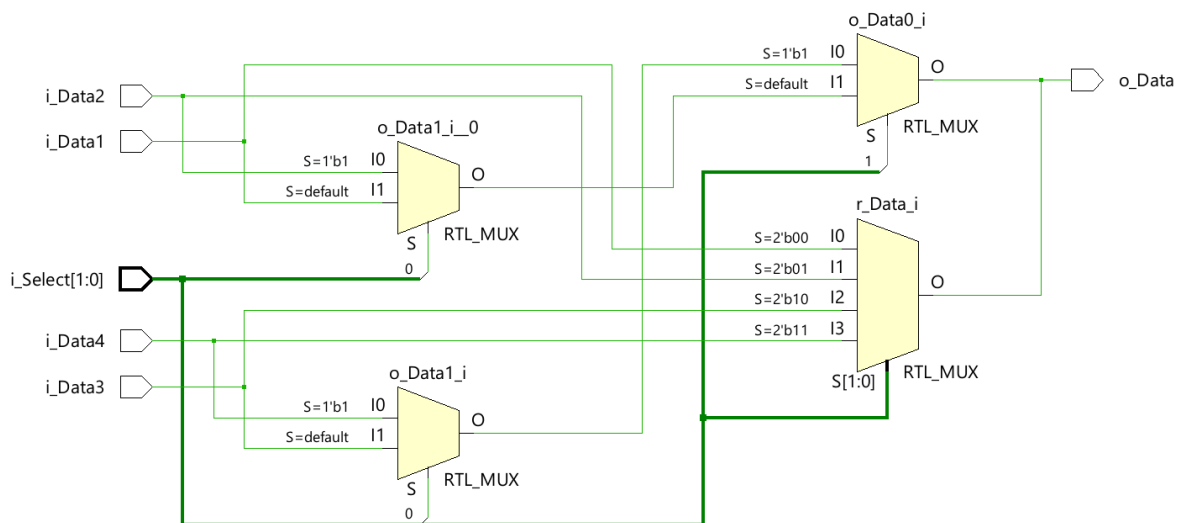
1 | `timescale 1ns / 1ps
2 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Module Name: tb_Mux_4_To_1
9 | // Target Devices: Artix
10 | //
11 | //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
12 |
13 | // Modulo de estimulo
14 | module tb_Mux_4_To_1();
15 |
16 | //Inputs
17 | reg [1:0] i_Select;
18 | reg i_Data1;
19 | reg i_Data2;
20 | reg i_Data3;
21 | reg i_Data4;
22 |
23 | //Outputs
24 | wire o_Data;
25 |
26 | integer i;
27 |
28 | //Instantiation of Unit Under Tes
29 | Mux_4_To_1 mux0(
30 |     .i_Select (i_Select),
31 |     .i_Data1 (i_Data1),
32 |     .i_Data2 (i_Data2),
33 |     .i_Data3 (i_Data3),
34 |     .i_Data4 (i_Data4),
35 |     .o_Data (o_Data)
36 | );
37 |
38 | ○ initial i_Data1 = 1'b0;
39 | ○ always #1 i_Data1 = ~i_Data1;
40 |
41 | ○ initial i_Data2 = 1'b0;
42 | ○ always #2 i_Data2 = ~i_Data2;
43 |
44 | ○ initial i_Data3 = 1'b0;
45 | ○ always #3 i_Data3 = ~i_Data3;
```

```

47 ○ initial i_Data4 = 1'b0;
48 ○ always #4 i_Data4 = ~i_Data4;
49
50 ○ initial
51 ○ begin
52 ○     for(i=0;i<4;i=i+1) begin
53 ○         i_Select = i;
54 ○         #5;
55 ○     end
56 ○ → $finish;
57 ○ end
58
59 ○ endmodule
60

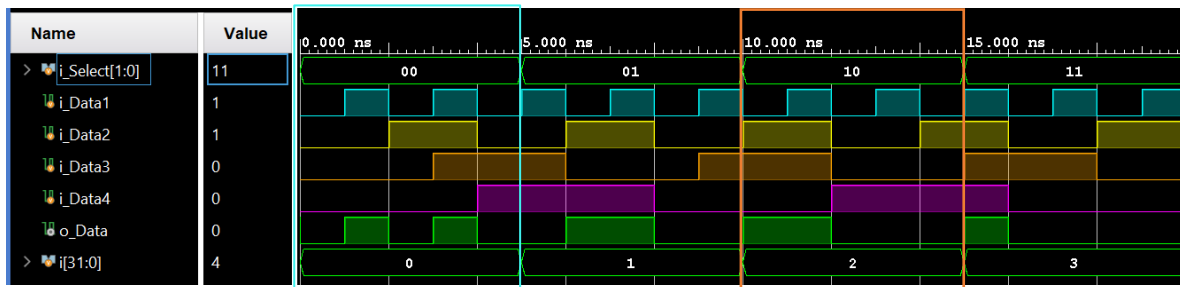
```

Descripción RTL en Vivado 2022.2



Podemos observar cómo utilizó multiplexores para la implementación del multiplexor 4 a 1.

Resultado de la simulación:



Cuando i_Select = 00, la salida o_Data es i_Data1

Cuando i_Select = 10, la salida o_Data es i_Data3

Como se observa en la simulación, la salida o_Data recibe el valor de entrada i_DataX según el valor indicado en el selector i_Select.

Propuesta 4 Operadores Lógicos

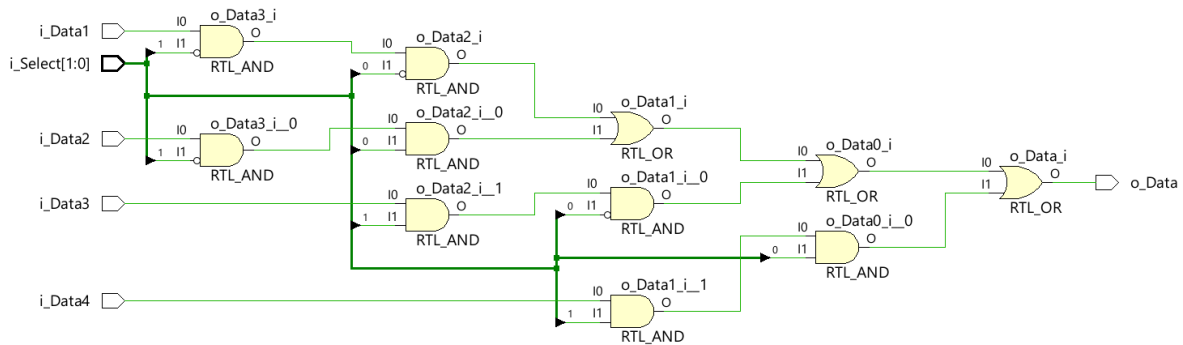
Código Verilog

```
1 | `timescale 1ns / 1ps
2 | ///////////////////////////////////////////////////////////////////
3 | // Unidad de Aprendizaje: Arquitectura de Computadoras
4 | // Alumno: Alan Adrian Malagon Baeza
5 | // Grupo: 5CV1
6 | // Profesor: Miguel Angel Aleman Arce
7 | //
8 | // Module Name: mux_4to1_logico
9 | // Target Devices: Artix
10 | //
11 | ///////////////////////////////////////////////////////////////////
12 |
13 |
14 | module mux_4to1_logico(input wire [1:0] i_Select,
15 |                       input wire i_Data1,
16 |                       input wire i_Data2,
17 |                       input wire i_Data3,
18 |                       input wire i_Data4,
19 |                       output wire o_Data);
20 |
21 | assign o_Data = (i_Data1 & ~i_Select[1] & ~i_Select[0]) |
22 |                 (i_Data2 & ~i_Select[1] & i_Select[0]) |
23 |                 (i_Data3 & i_Select[1] & ~i_Select[0]) |
24 |                 (i_Data4 & i_Select[1] & i_Select[0]);
25 |
26 | endmodule
```

Código Verilog Testbench

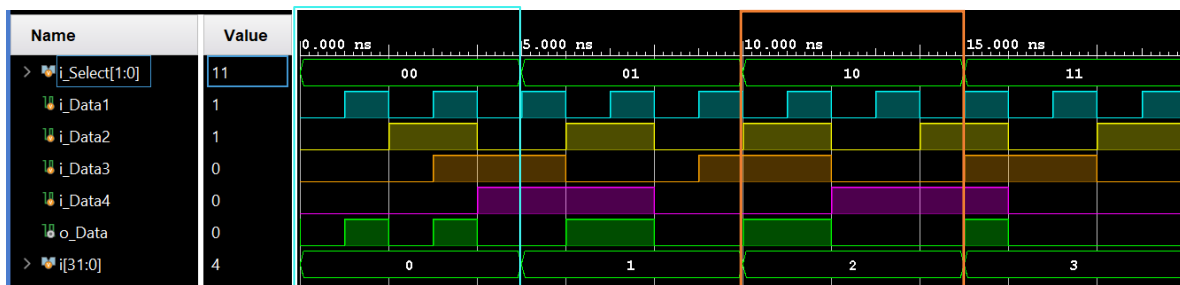
```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Unidad de Aprendizaje: Arquitectura de Computadoras
4  // Alumno: Alan Adrian Malagon Baeza
5  // Grupo: 5CV1
6  // Profesor: Miguel Angel Aleman Arce
7  //
8  // Module Name: mux_4to1_logico
9  // Target Devices: Artix
10 //
11 ///////////////////////////////////////////////////////////////////
12
13 module tb_mux_4to1_logico();
14
15     //Inputs
16     reg [1:0] i_Select;
17     reg i_Data1;
18     reg i_Data2;
19     reg i_Data3;
20     reg i_Data4;
21
22     //Outputs
23     wire o_Data;
24
25     integer i;
26
27     mux_4to1_logico mux0(
28         .i_Select (i_Select),
29         .i_Data1 (i_Data1),
30         .i_Data2 (i_Data2),
31         .i_Data3 (i_Data3),
32         .i_Data4 (i_Data4),
33         .o_Data (o_Data)
34     );
35
36     initial i_Data1 = 1'b0;
37     always #1 i_Data1 = ~i_Data1;
38
39     initial i_Data2 = 1'b0;
40     always #2 i_Data2 = ~i_Data2;
41
42     initial i_Data3 = 1'b0;
43     always #3 i_Data3 = ~i_Data3;
44
45     initial i_Data4 = 1'b0;
46     always #4 i_Data4 = ~i_Data4;
47
48     initial
49     begin
50         for(i=0;i<4;i=i+1) begin
51             i_Select = i;
52             #5;
53         end
54         $finish;
55     end
56 endmodule
```

Descripción RTL en Vivado 2022.2



Podemos observar cómo utilizó las compuertas lógicas AND y OR para la implementación del multiplexor 4 a 1.

Resultado de la simulación:



Cuando i_Select = 00, la salida o_Data es i_Data1

Cuando i_Select = 10, la salida o_Data es i_Data3

Como se observa en la simulación, la salida o_Data recibe el valor de entrada i_DataX según el valor indicado en el selector i_Select.

Conclusión

El término lógica de circuito integrado de escala media (MSI) se refiere a un conjunto de circuitos lógicos combinacionales básicos que implementan funciones simples y de uso común, como multiplexores. La lógica MSI también puede incluir operaciones como comparadores y circuitos aritméticos simples.

Si bien un circuito lógico MSI puede tener múltiples salidas, cada salida requiere su propia expresión lógica única que se basa en las entradas del sistema.

Un multiplexor es un sistema que tiene una salida y múltiples entradas. En un momento dado, una y solo una entrada se enruta a la salida en función del valor en un conjunto de *líneas seleccionadas*. Para n líneas seleccionadas, un multiplexor puede admitir 2^n entradas.

Los HDL son particularmente útiles para describir la lógica MSI debido a su capacidad de modelado abstracto. Mediante el uso de condiciones booleanas y asignaciones de vectores, el comportamiento de la lógica MSI se puede modelar de forma compacta e intuitiva.

Referencia

1. Brock J. LaMeres, Introduction to Logic Circuits & Logic Design with Verilog, Springer, 1st Edition, USA, 2017.