



Memoria

Sistemas Electrónicos para la Automatización

Docente: Jorge Romero Sánchez

ROM y Multiplexor

Jorge Benavides Macías: jorge2@uma.es

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Contenido

1	Especificaciones	4
2	Diseño	5
2.1	ROM	5
2.1.1	Estudio teórico	5
2.1.2	Entidad y arquitectura	5
2.2	MUX	6
2.2.1	Estudio teórico	6
2.2.2	Entidad y arquitectura	6
3	Simulación	8
3.1	Test Bench	8
3.2	Fichero de estímulos	12
3.3	Cronograma de simulación	12
3.4	Fichero CSV generado	12
4	Síntesis	14
4.1	Esquemático RTL	14



Lista de figuras

1	Cronograma de simulación de la ROM y del MUX	12
2	Fichero CSV generado para la ROM y el MUX	13
3	Esquemático RTL del MUX	14
4	Esquemático RTL de la ROM	15
5	Esquemático RTL de la ROM de 3 bits	15

Lista de códigos

1	Entidad de la ROM	5
2	Arquitectura de la ROM	6
3	Entidad del MUX	6
4	Arquitectura del MUX	7
5	Test bench	12

1. Especificaciones

- Simular y sintetizar la ROM y el MUX propuestos en las transparencias anteriores.
- Como máximo tendrán 4 direcciones (ROM), ó de forma equivalente, dos líneas de selección (para el MUX) y el bus de datos será, como mucho de 1 byte, tanto para la ROM como para el MUX.
- Generar un único proyecto en Vivado para ambos módulos.
- Generar, para la fase de simulación, un único Test Bench con manejo de ficheros (modelo en Tema 5) que ilustre el funcionamiento correcto del los dos sistemas y su equivalencia como sistemas combinacionales, es decir, para igual entrada, igual salida en ambos casos (son 2 DUTs en el TB: la ROM y el MUX).
- Sintetizarlos a nivel RTL (RTL Analysis) para comprobar que efectivamente son combinacionales (no tiene elementos de memoria) y que se corresponden con lo que representan.

2. Diseño

2.1. ROM

2.1.1. Estudio teórico

Elemento de memoria con información fija almacenada, contiene una entrada de direcciones y una salida con los datos.

2.1.2. Entidad y arquitectura

Esta entidad es muy sencilla contiene dos puertos uno de entrada y otro de salida; es interesante destacar que se usa una librería especial (*work.Tipos_ROM_MUX*) en el que se han definido algunas constantes y una matriz llamada *Tabla_ROM*.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  use work.Tipos_ROM_MUX.all;
5
6  use ieee.numeric_std.all;
7
8  entity ROM is
9      generic (N_Bits_Dir : Natural := 2);
10     port (Direccion : in std_logic_vector (N_Bits_Dir - 1 downto 0);
11           Dato      : out std_logic_vector (N_Bits_Dato - 1 downto 0));
12 end ROM;
```

Código 1: Entidad de la ROM

En la arquitectura se hace una asignación en la señal *Dato* basada en la dirección de entrada, se hace un *casting* y el uso de una función para acceder a la dirección de la ROM. Además, se rellena la *Tabla_ROM*.

```
1  architecture Comportamiento of ROM is
2      -- Se rellena la ROM con varios estilos:
3      constant Tabla_ROM : Tabla(0 to 2**N_Bits_Dir-1) :=
4          (('1','0','1','0','1','0','1','0'),
5           b"1011_1011", -- si no se indica la "b" no será correcto
6           --x"CC",
7           --x"DD",
8           --x"EE",
```

```

9      --x"FF",
10     (others => '0'),
11     (0 | 4 => '1', others => '0'));
12 begin
13     Dato <= Tabla_ROM(to_integer(unsigned(Direccion)));
14
15 end Comportamiento;

```

Código 2: Arquitectura de la ROM

2.2. MUX

2.2.1. Estudio teórico

Es un dispositivo que te permite seleccionar una de las entradas con unas señales de control y obtenerla en la salida, sigue la ecuación $n = 2^c$, donde n es número de entradas y c en número de señales de control.

2.2.2. Entidad y arquitectura

La entidad del MUX se parece mucho a la entidad de la ROM ya que cuenta con un puerto de entrada asociado a la dirección y uno de salida con el dato, sin embargo añade una *Tabla_ROM* que contiene los datos almacenados.

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  use work.Tipos_ROM_MUX.all;
5
6  use ieee.numeric_std.all;
7
8  entity MUX is
9      generic (N_Bits_Dir : Natural := 2);
10     port (Direccion : in std_logic_vector (N_Bits_Dir - 1 downto 0);
11           Dato       : out std_logic_vector (N_Bits_Dato - 1 downto 0);
12           Tabla_ROM  : in Tabla(0 to 2**N_Bits_Dir-1));
13 end MUX;

```

Código 3: Entidad del MUX

La asignación es prácticamente idéntica que la ROM, a la dirección se le hace un “casting” y luego se aplica una función.



```
1 architecture Comportamiento of Mux is
2 begin
3     Dato <= Tabla_ROM(to_integer(unsigned(Direccion)));
4
5     end Comportamiento;
```

Código 4: Arquitectura del MUX

3. Simulación

3.1. Test Bench

En el [Código 5](#) hemos hecho uso de la plantilla del campus para simulaciones desde fichero. Una de las grandes e interesantes modificaciones es la declaración de dos componentes y su respectiva estimulación para confirmar la igualdad; tiene una serie de parámetros constantes para instanciar los componentes genéricos y señales internas para la correcta estimulación.

En esta práctica hay un detalle interesante ya que hay dos *DUTs* definidas dentro del begin de la arquitectura, uno para el MUX y otro para la ROM.

```
1  -----
2  -- Company: Universidad de Malaga
3  -- Engineer: Izan Amador, Jorge L. Benavides
4  --
5  -- Create Date: 23.11.2022 17:47:41
6  -- Design Name: rom_mux
7  -- Module Name: Test_Bench_Fichero - Behavioral
8  -- Project Name: rom_mux
9  -- Target Devices: Zybo
10 -- Tool Versions: Vivado 2022.1
11 -- Description: Comparation between a rom and a mux
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 -----
19
20
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use work.Tipos_ROM_MUX.all;
24 use STD.textIO.ALL;                                -- Se va a hacer uso de ficheros.
25
26 entity Test_Bench_Fichero is
27 -- Port ( );
28 end Test_Bench_Fichero;
```



```

30 architecture Comportamiento of Test_Bench_Fichero is
31
32     component MUX
33         generic (N_Bits_Dir : Natural := 3);
34         port (Direccion : in std_logic_vector (N_Bits_Dir - 1 downto 0);
35              Dato       : out std_logic_vector (N_Bits_Dato - 1 downto 0);
36              Tabla_ROM : in Tabla(0 to 2**N_Bits_Dir-1));
37     end Component MUX;
38
39     component ROM
40         generic (N_Bits_Dir : Natural := 3);
41         port (Direccion : in std_logic_vector (N_Bits_Dir - 1 downto 0);
42              Dato       : out std_logic_vector (N_Bits_Dato - 1 downto 0));
43     end Component ROM;
44
45
46     constant semiperiodo : time      := 10 ns;
47     constant N_Bits_Dir  : natural    := 2;
48     constant n           : integer    := 2;
49
50     signal Direccion_interno : std_logic_vector (N_Bits_Dir - 1 downto 0) := (others
51 => 'U');
52     signal Dato_interno : std_logic_vector(N_Bits_Dato - 1 downto 0) := (others =>
53 => 'U');
54     signal Tabla_ROM_interno : Tabla(0 to 2**N_Bits_Dir-1) :=
55     (('1','0','1','0','1','0','1','0'),
56      b"1011_1011", -- si no se indica la "b" no serÁa correcto
57      --x"CC",
58      --x"DD",
59      --x"EE",
60      -- x"FF",
61      (others => '0'),
62      (0 | 4 => '1', others => '0'));
63
64 begin
65
66     DUT1 : MUX
67         generic map (N_Bits_Dir)

```

```

67     port map(
68         Direccion => Direccion_interno,
69         Dato => Dato_interno,
70         Tabla_ROM => Tabla_ROM_interno);
71
72 DUT2 : ROM
73     generic map (N_Bits_Dir)
74     port map (
75         Direccion => Direccion_interno,
76         Dato => Dato_interno);
77
78
79 Estimulos_Desde_Fichero : process
80
81     file Input_File : text;
82     file Output_File : text;
83
84     variable Input_Data      : BIT_VECTOR(n-1 downto 0) := (OTHERS => '0');
85     variable Delay           : time                      := 0 ms;
86     variable Input_Line      : line                      := NULL;
87     variable Output_Line     : line                      := NULL;
88     variable Std_Out_Line    : line                      := NULL;
89     variable Correcto        : Boolean                  := True;
90     constant Coma            : character                := ',';
91
92
93 begin
94
95 -- rom_mux_Estimulos.txt contiene los estÃmulos y los tiempos de retardo para el
96 -- semisumador.
97     file_open(Input_File,
98 -- "C:\Users\izana\Documents\GitHub\SEA\Estimulos\rom_mux_Estimulos.txt",
99 -- read_mode);
100
101 -- rom_mux_Estimulos.csv contiene los estÃmulos y los tiempos de retardo para el
102 -- Analog Discovery 2.
103     file_open(Output_File,
104 -- "C:\Users\izana\Documents\GitHub\SEA\CSV\rom_mux_Estimulos.csv", write_mode);
105
106 -- Titles: Son para el formato EXCEL *.CSV (Comma Separated Values):

```

```
101 write(STD_Out_Line, string("Retardo"), right, 7);
102 write(STD_Out_Line, Coma, right, 1);
103 write(STD_Out_Line, string("Entradas"), right, 8);
104
105 Output_Line := Std_Out_Line;
106
107 writeline(output, Std_Out_Line);
108 writeline(Output_File, Output_Line);
109
110 while (not endfile(Input_File)) loop
111
112     readline(Input_File, Input_Line);
113
114     read(Input_Line, Delay, Correcto); -- Comprobaci3n de que se trata de un
↪ texto que representa
115     -- el retardo, si no es as3 leemos la siguiente l3nea.
116     if Correcto then
117
118         read(Input_Line, Input_Data); -- El siguiente campo es el vector de
↪ pruebas.
119         Direccion_interno <= TO_STDLOGICVECTOR(Input_Data)(1 downto 0);
120         -- De forma simult3nea lo volcaremos en consola en csv.
121         write(STD_Out_Line, Delay, right, 5); -- Longitud del retardo, ej. "20 ms".
122         write(STD_Out_Line, Coma, right, 1);
123         write(STD_Out_Line, Input_Data, right, 2); --Longitud de los datos de
↪ entrada.
124
125         Output_Line := Std_Out_Line;
126
127         writeline(output, Std_Out_Line);
128         writeline(Output_File, Output_Line);
129
130         wait for Delay;
131     end if;
132 end loop;
133
134 file_close(Input_File); -- Cerramos el fichero de entrada.
135 file_close(Output_File); -- Cerramos el fichero de salida.
136 wait;
```

```

137     end process Estimulos_Desde_Fichero;
138
139
140 end Comportamiento;

```

Código 5: Test bench

3.2. Fichero de estímulos

En el fichero hemos accedido a las 4 direcciones disponibles.

```

#Fichero de Estímulos de ROM_MUX
#Device Name: Discovery2NI
#Nombre: Izan Amador, Jorge Benavides
#Fecha: 7 de Diciembre de 2022.
#
# Delay Time (ns) Input (Direccion [1:0]).

# Secuencia correcta
20 ns 00
20 ns 01
20 ns 10
20 ns 11

```

3.3. Cronograma de simulación

En la [Figura 1](#) se observa el acceso a las 4 direcciones y los datos que contienen, los dos dispositivos producen la misma la salida, por lo tanto se confirma que si las entradas y las señales de control son equivalentes entre la ROM y el MUX.

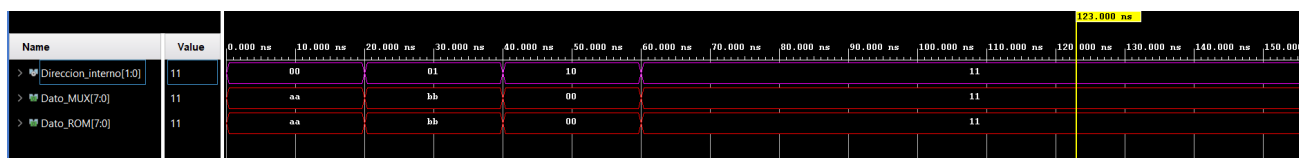
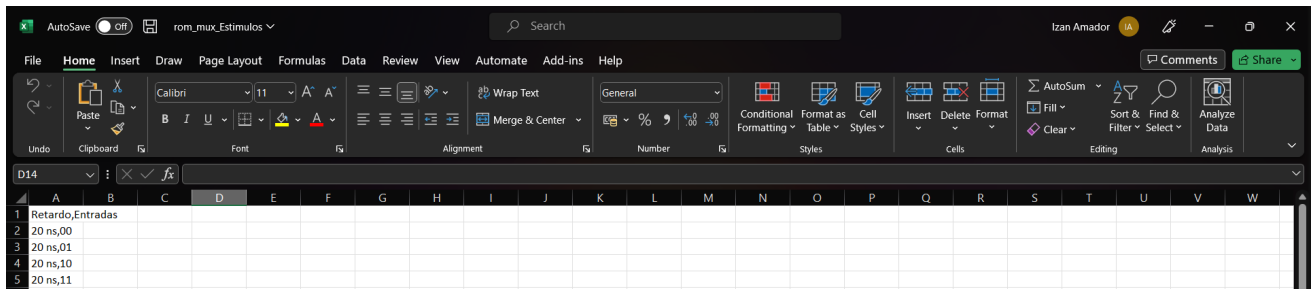


Figura 1: Cronograma de simulación de la ROM y del MUX

3.4. Fichero CSV generado

El fichero generado para una posible implementación en la plaza Zybo y simulación con el Analog Discovery.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Retardo,Entradas																						
2	20 ns,00																						
3	20 ns,01																						
4	20 ns,10																						
5	20 ns,11																						

Figura 2: Fichero CSV generado para la ROM y el MUX

4. Síntesis

La síntesis ha sido un éxito, ambos son combinacionales y ambas entradas son de un byte, si seguimos las ecuaciones de la teoría se demuestra que son dispositivos equivalentes al ser $m = n = 8$ y $k = l = 2$ donde:

- **m**: tamaño del bus de datos de la ROM
- **n**: líneas de datos del multiplexor
- **k**: líneas de dirección de la ROM
- **l**: líneas de selección de datos del multiplexor

Es muy interesante el hecho de que ambos dispositivos a nivel de síntesis y esquemático tienen la misma forma, son en esencia multiplexores, esto es un poco más profundo ya que nos vamos a nivel de *CLB* y de como están compuestas, dando que una de sus partes esenciales es un multiplexor el sintetizador de Vivado ha optimizado la *ROM* a un multiplexor, esto ocurre porque las direcciones son de 2 bits, pero si aumentamos en uno el tamaño el diseño cambia como se muestra en la [Figura 5](#).

4.1. Esquemático RTL

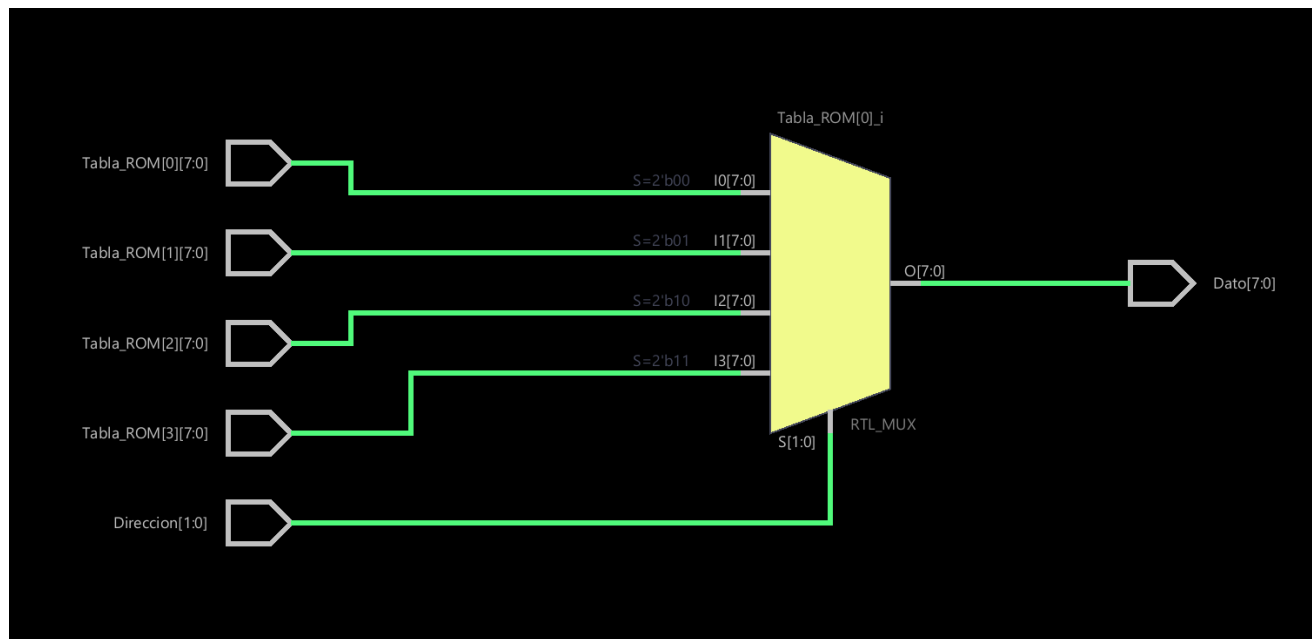


Figura 3: Esquemático RTL del MUX

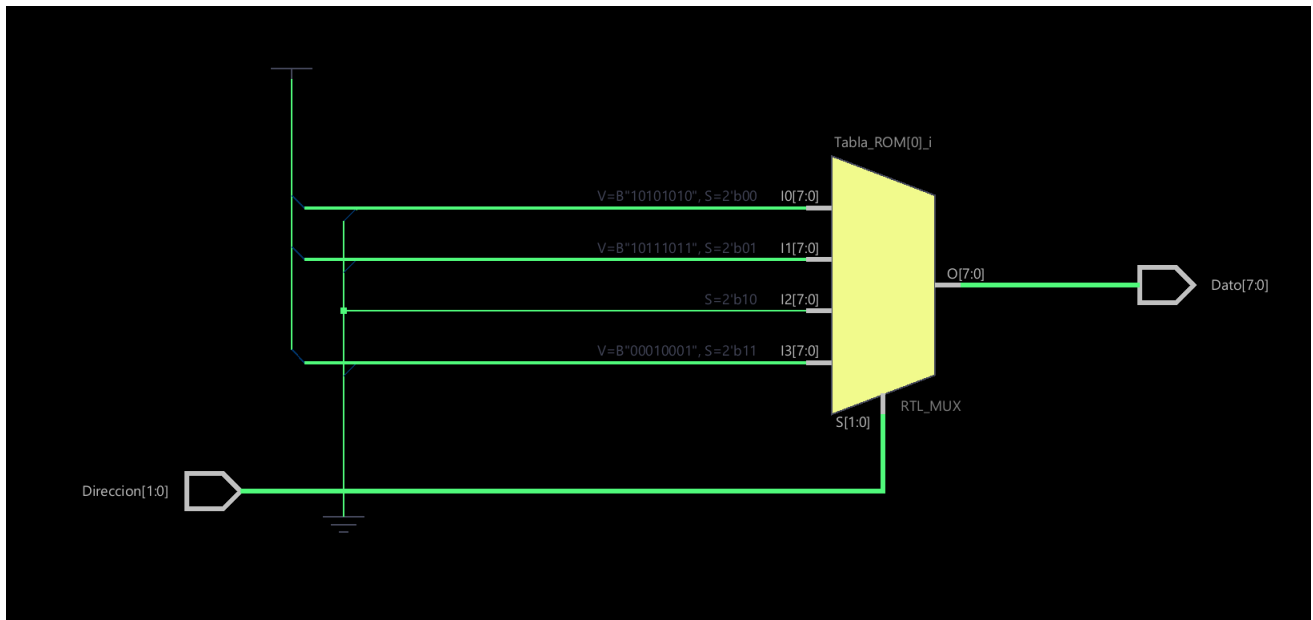


Figura 4: Esquemático RTL de la ROM

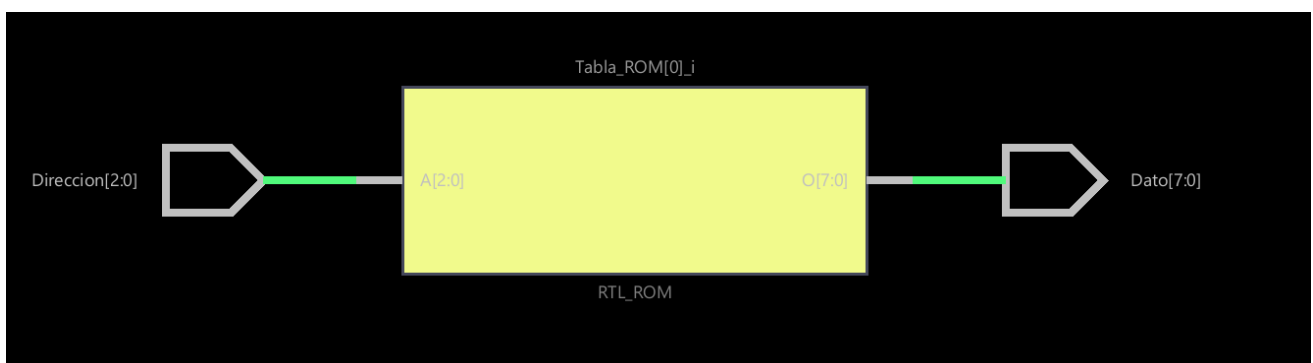


Figura 5: Esquemático RTL de la ROM de 3 bits