



Memoria

Sistemas Electrónicos para la Automatización

Docente: Jorge Romero Sánchez

ROM y Multiplexor

Izan Amador Bustos: izan.amador@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 | 13 |
| 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 | 15 |

Lista de códigos

| | | |
|---|----------------------------------|----|
| 1 | Entidad de la ROM | 5 |
| 2 | Arquitectura de la ROM | 6 |
| 3 | Entidad del MUX | 7 |
| 4 | Arquitectura del MUX | 7 |
| 5 | Test bench | 11 |

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

La **ROM (Read Only Memory)** se define como un medio de almacenamiento de datos únicamente de lectura, en la que la información se guarda de manera permanente. Su estructura está formada por k líneas de entradas y n líneas de salida.

Las líneas de entrada se usan para acceder a las direcciones de memoria del contenido de la ROM. Las k líneas de entrada son binarias, por lo que hay un total de 2^k direcciones totales que pueden ser referidas por las líneas de entradas y cada una de ellas contiene n bits de información por lo que se define la memoria de un tamaño de $2^k \times n$.

2.1.2. Entidad y arquitectura

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

Código 1: Entidad de la ROM

En la entidad se define el número de bits de la dirección como 2 para cumplir con la especificación de tener menos de 4 direcciones. Se definen la entrada y la salida como vectores de la librería `std_logic`. Además se incluye los **tipos ROM y MUX**, además de las librerías de **std_logic** y **numeric**.

```
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

En la arquitectura **se rellena** la tabla de la ROM **mediante diferentes estilos**: bit a bit, secuencia binaria, mediante la expresión `others` y combinando la expresión `others` con operaciones lógicas.

Después de inicializarla, se realiza un casting de **std_logic** a **unsigned** a **entero** antes de realizar la asignación a la salida.

2.2. MUX

2.2.1. Estudio teórico

Definimos el multiplexor como un circuito combinacional con 2^n entradas de datos, n entradas de selección y una única salida. La entrada de control permite seleccionar una única entrada de datos para que sea transmitida a la salida.

De esta manera, la implementación de un multiplexor puede tener distintas aplicaciones como la de serializador, selector, implementador de funciones lógicas (diseñándolas a partir de conexiones con 0 y 1 de sus entradas de la expresión de álgebra de bool deseada) o transmisor de datos.

2.2.2. Entidad y arquitectura

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



Código 3: Entidad del MUX

Se definen dos líneas de selección por especificación del enunciado. No se define el número de bits del dato ya que se encuentra definido en los tipos ROM MUX.

```
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

Se realiza un casting de **std_logic** a **unsigned** antes de realizar la asignación de la tabla ROM al dato.

3. Simulación

3.1. Test Bench

```
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;
29
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 serÃa correcto
57   --x"CC",
58   --x"DD",
59   --x"EE",
60   -- x"FF",
61   (others => '0'),
62   (0 | 4 => '1', others => '0')));
63
64
65 begin
66
67 DUT1 : MUX
68     generic map (N_Bits_Dir)
69     port map(
70         Direccion => Direccion_interno,
71         Dato => Dato_interno,
72         Tabla_ROM => Tabla_ROM_interno);
73
74 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 est mulos 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 est mulos 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):
107     write(Std_Out_Line, string'("Retardo"), right, 7);
108     write(Std_Out_Line, Coma, right, 1);
109     write(Std_Out_Line, string'("Entradas"), right, 8);
110
111     Output_Line := Std_Out_Line;

```

```
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;
```

C3digo 5: Test bench

Del banco de pruebas cabe destacar en primer lugar la definición de la entidad vacía para su correcto funcionamiento. En la arquitectura, se definen los **dos componentes** con su correspondientes puertos y genéricos. Se define el tamaño de las constantes del número de bits de dirección y del vector de entrada a **2** para cumplir con las especificaciones del enunciado.

3.2. Fichero de estímulos

```
#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
```

Se codifica una dirección de dos bits como entrada, con una secuencia que representa las diferentes combinaciones posibles para dos dígitos.

3.3. Cronograma de simulación

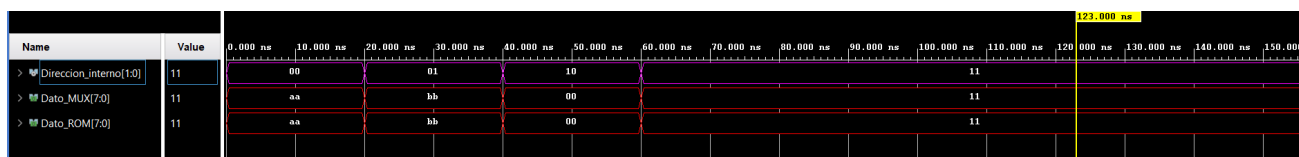
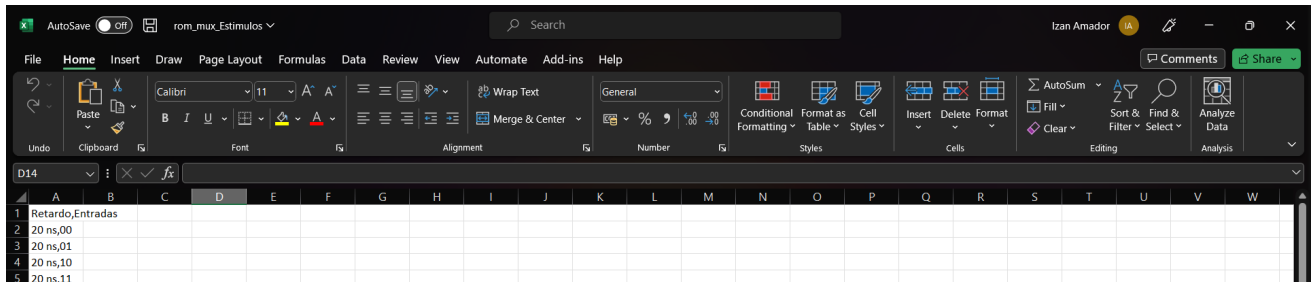


Figura 1: Cronograma de simulación de la ROM y del MUX

En la simulación, se puede apreciar los **4 casos** con los valores almacenados tanto en la ROM con las combinaciones del multiplexor siendo mostradas en la salida simultáneamente en **dos dispositivos bajo prueba** (DUT) distintos.

3.4. Fichero CSV generado



| | 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

Se aprecia en el archivo de valores separados por comas la misma secuencia de entrada que en archivo de texto plano sin los comentarios.

4. Síntesis

4.1. Esquemático RTL

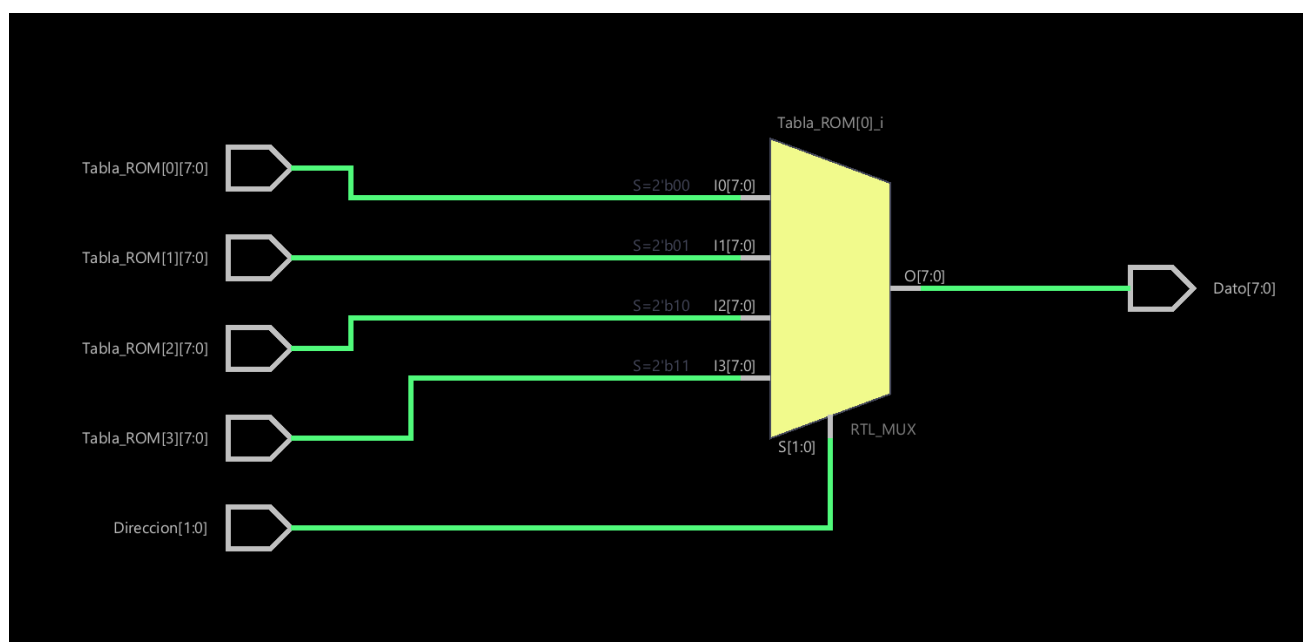


Figura 3: Esquemático RTL del MUX

En la [Figura 3](#), se aprecia el multiplexor con 4 entradas de 8 bits y un selector de 2 bits. Sin elementos secuenciales ni latches, por lo que se considera correcta la síntesis. Se realiza el reload, y el cambio de top para obtener esquemático RTL del otro dispositivo en pruebas.

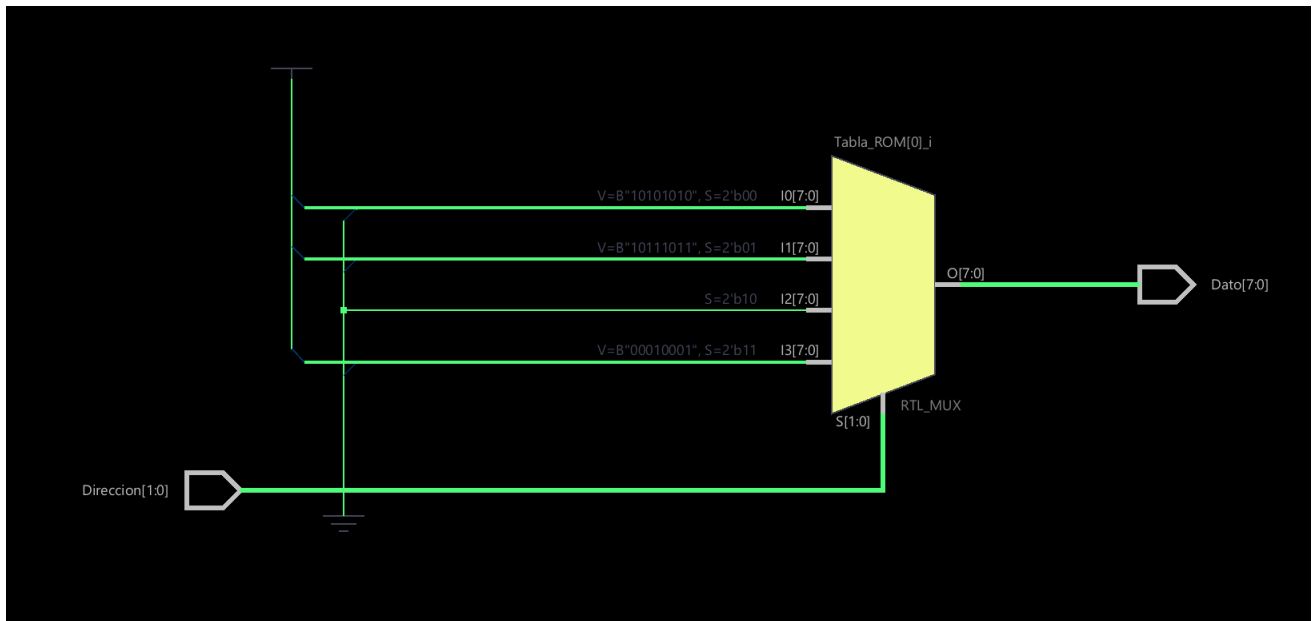


Figura 4: Esquemático RTL de la ROM

En la [Figura 4](#), se aprecia un bloque correspondiente a una ROM, sintetizado con un bloque MUX, que **tampoco presenta elementos secuenciales** ni latches.

De esta manera se comprueba a nivel de esquemático que **su funcionamiento es idéntico** ya que la síntesis para una dirección de 2 bits genera un esquemático en la ROM de un multiplexor.

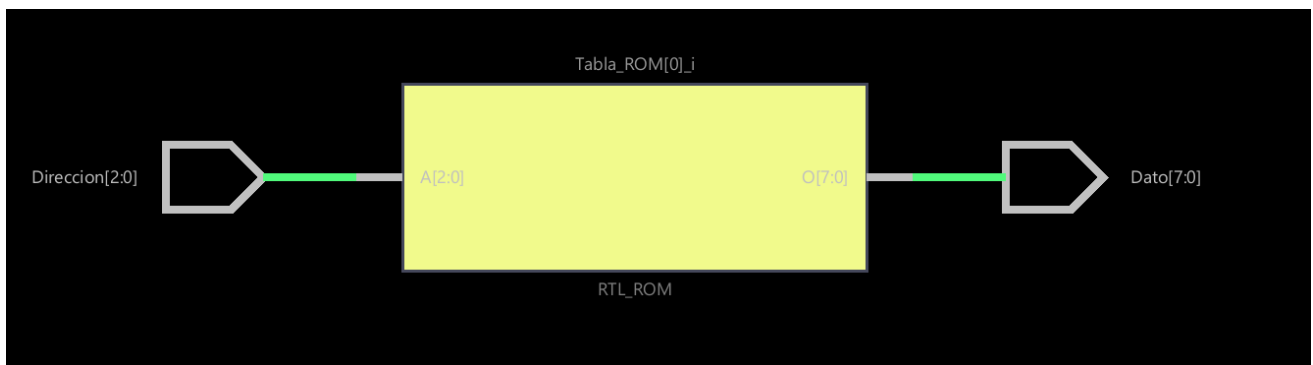


Figura 5: Esquemático RTL de la ROM

Cabe destacar que al aumentar el tamaño del vector de selección a **3 bits**, se produce un cambio en el símbolo del esquemático RTL de la memoria ROM, siendo este el mostrado en la [Figura 5](#).

Se comprueba que ambos sistemas son 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

De esta manera se cierra el proceso de diseño completo con la síntesis correcta del mismo, obteniendo resultados coherentes con todos las etapas del proceso.