2.4. Descripción de Memorias en VHDL

Memorias en FPGAs

OBJETIVOS:

- Memorias en FPGA
- Tipos de escritura en RAM:
  - Write-First
  - Read-First
  - No change
- Descripción VHDL de memoria RAM
- RAM síncrona versus asíncrona
- Descripción VHDL de memoria ROM

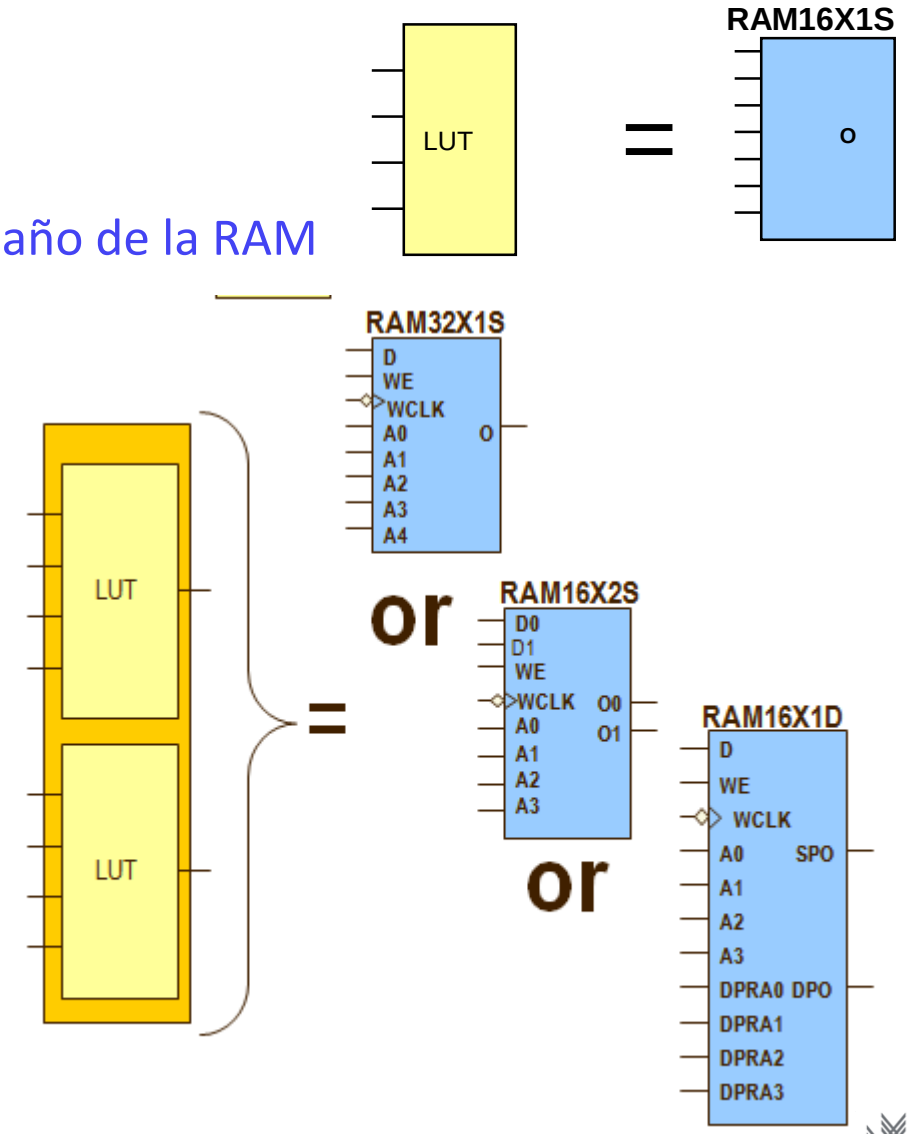
2.4.1. Tipos de memorias en FPGAs

Memorias en FPGAs

- Tipos de memoria en las FPGA
  - RAM distribuida (LUTs)
  - Block RAM (Embebida)
- Instanciación versus Inferencia
  - La **inferencia** permite la **portabilidad**
  - La instanciación optimiza rendimiento y recursos
- Características de las memorias FPGAs
  - Escritura síncrona
  - Lectura síncrona/asíncrona

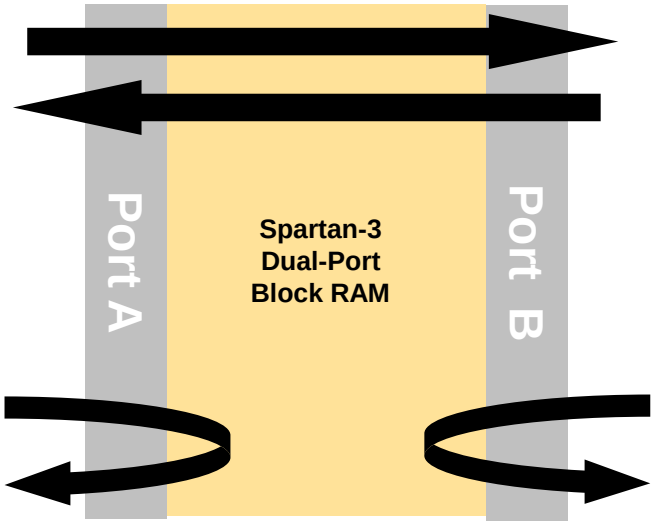
Memorias en FPGAs: RAM Distribuida

- LUT → RAM distribuida
  - Una LUT → 16x1 RAM
  - LUTs en cascada → Amplía el tamaño de la RAM
- Escritura síncrona
- Lectura asíncrona
  - Síncrona añadiendoles FF-D.
- Dos LUTs pueden formar:
  - 32 x 1 single-port RAM
  - 16 x 2 single-port RAM
  - 16 x 1 dual-port RAM

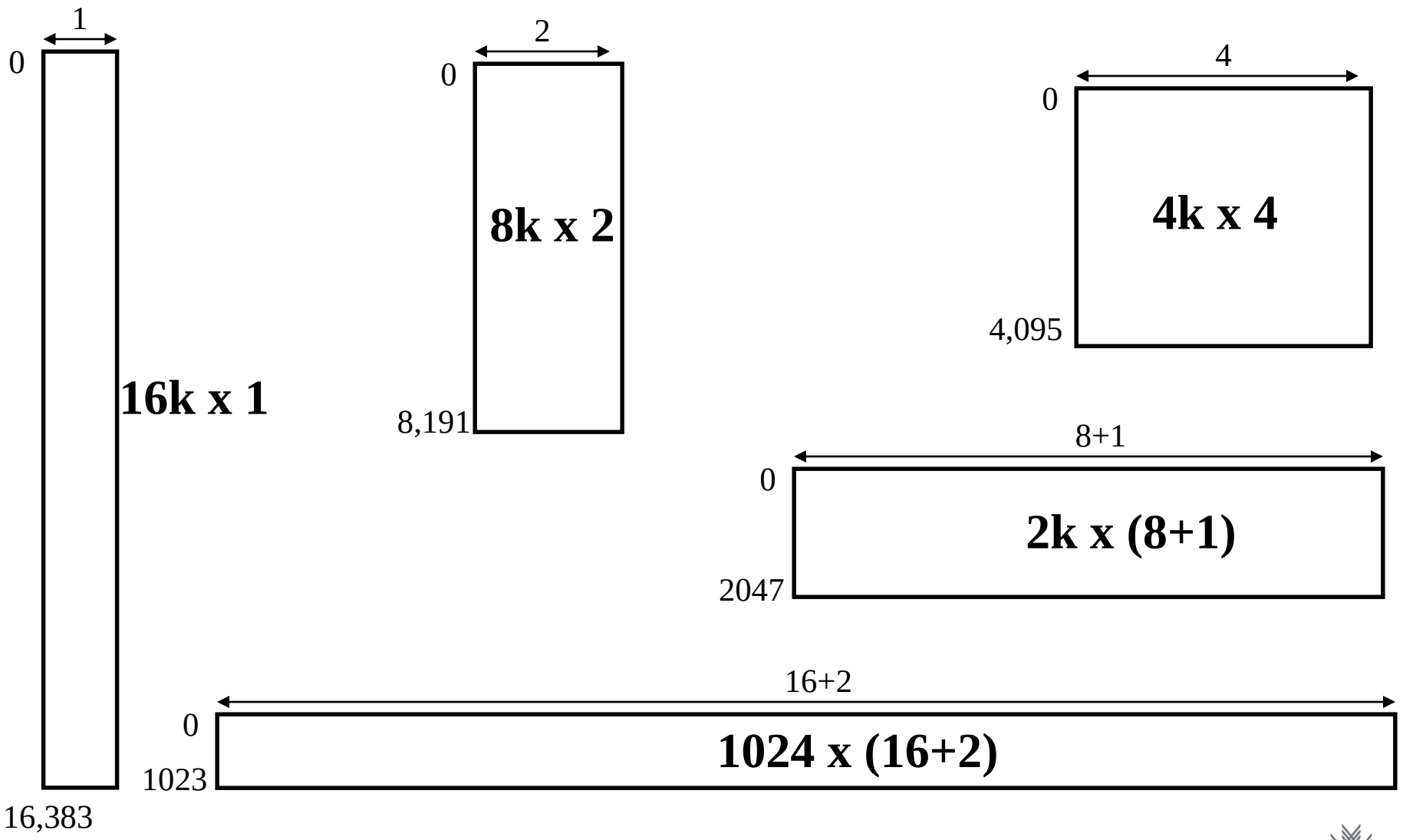


Memorias en FPGAs: Bloques de RAM

- Bloques de memoria RAM dedicado
- Desde 4 hasta más de 100 bloques según modelo
- 18 kbits por bloque (con bit de paridad)
- Se pueden encadenar bloques
- Configurables como de simple o doble puerto
- La lectura y escritura es siempre **síncrona (?)**



Memorias en FPGAs: Bloques de RAM



2.4.2. Descripción de RAM en VHDL

Modos de escritura en memoria RAM

Write Mode: Se refiere a la política utilizada cuando se realiza una escritura simultánea con una lectura en una misma dirección (mismo puerto).

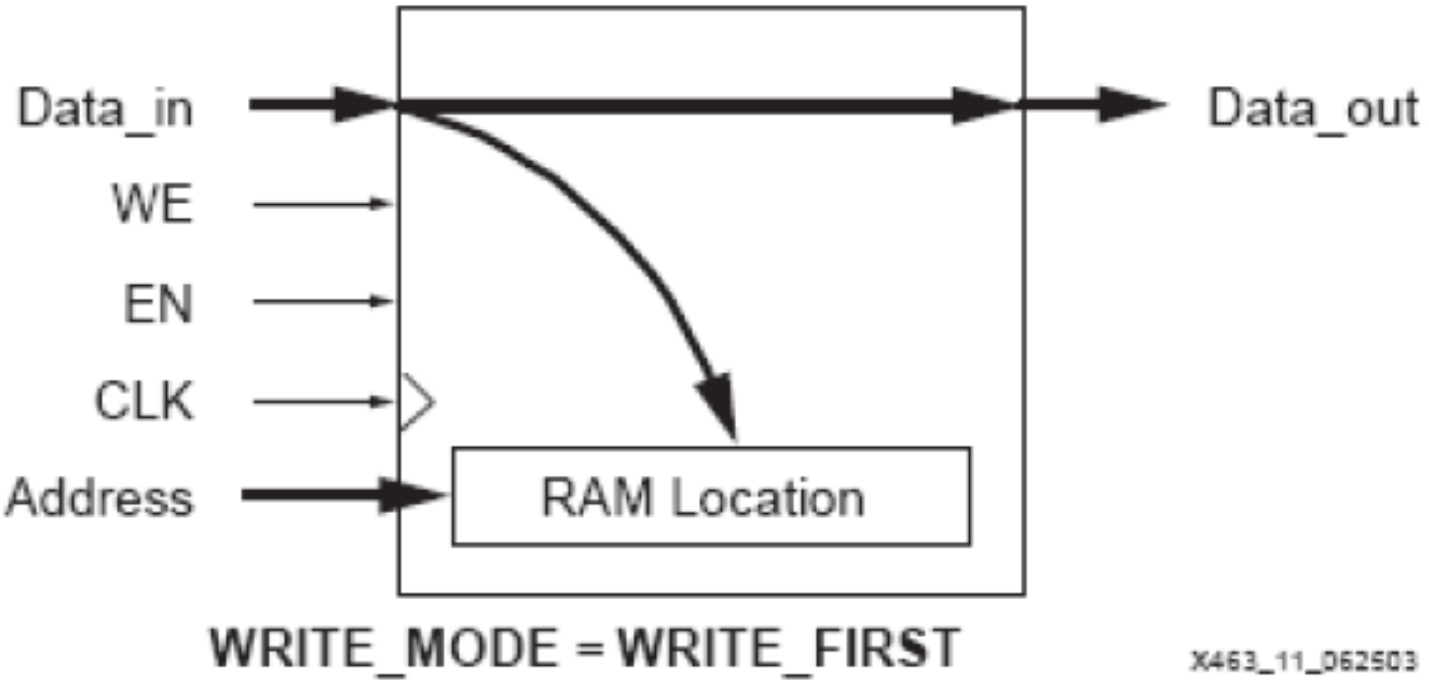
Tipos:

- **Write First**: El dato de entrada (DataIn) es escrito en la dirección de memoria especificada y simultáneamente está disponible en la salida. (DataOut).

- **Read First:** Dada la dirección de memoria, el dato guardado en esa dirección aparece en la salida, DataOut. El dato de entrada, DataIn, se almacena despues en esa dirección.

- **No change:** En un ciclo de reloj, o bien se lee, o bien se escribe. Si se escribe, la salida queda desabilitada (Output Enable).

A. Modos de escritura WRITE FIRST en RAM Síncrona



A. Modos de escritura WRITE FIRST en RAM Síncrona

```
entity RAM_16x4 is
port ( Clk, WE : in std_logic;
          Address : in std_logic_vector(3 downto 0); -- Adress
          DataIn : in std_logic_vector(3 downto 0); -- Data in
          DataOut: out std_logic_vector(3 downto 0)); -- Data out
end RAM_16x4 ;

architecture behavioral of RAM_16x4 is
        type ram_type is array(15 downto 0) of std_logic_vector(3 downto 0);
      1 signal RAM : ram_type;  2
begin
process (Clk)
     Begin
      if rising_edge(Clk) then
           -- Operación de LECTURA/ESCRITURA síncrona
          if WE = '1' then
             RAM(to_integer(unsigned(Address))) <= DataIn ; -- Conversión en dos pasos de
                                                            -- std_logic_vector a
                                                            -- integer(numeric_std)
           3
           DataOut <= DataIn; --El dato que se lee, es el mismo que se escribe

          else
          -- Operación de SOLO LECTURA
           DataOut <= RAM(to_integer(unsigned(Address))); -- Valor leído en Address

          end if;
       end if;
end process;
end Behavioral;
```

1 2 3 4

Tipos de datos en VHDL: Datos definidos por el usuario (type)

1

En VHDL el usuario puede definir tipos de datos nuevos:

```
type <nombre_tipo> is <tipo_de_dato>;
type Cero_Siete is unsigned(2 downto 0);--0,1,2,…,7
```

2

En VHDL el usuario puede definir un array de elementos de un tipo compuesto:

```
type <nombre_array> is array <rango_array> of <tipo_vector>;
type ram_type is array(15 downto 0) of
                              std_logic_vector(3 downto 0);
```

ram_type

15	“0110”
14	“1110”
…	
1	“0000”
0	“0101”

Conversión entre tipos de datos en VHDL: std_logic to integer

3

En VHDL el tipo de datos std_logic_vector puede convertirse a integer, **previa conversión** de std_logic_vector a signed/unsigned.

Aplicación: actuar como índice en arrays de elementos.

```
RAM(to_integer(unsigned(Address))) <= DataIn ;
```

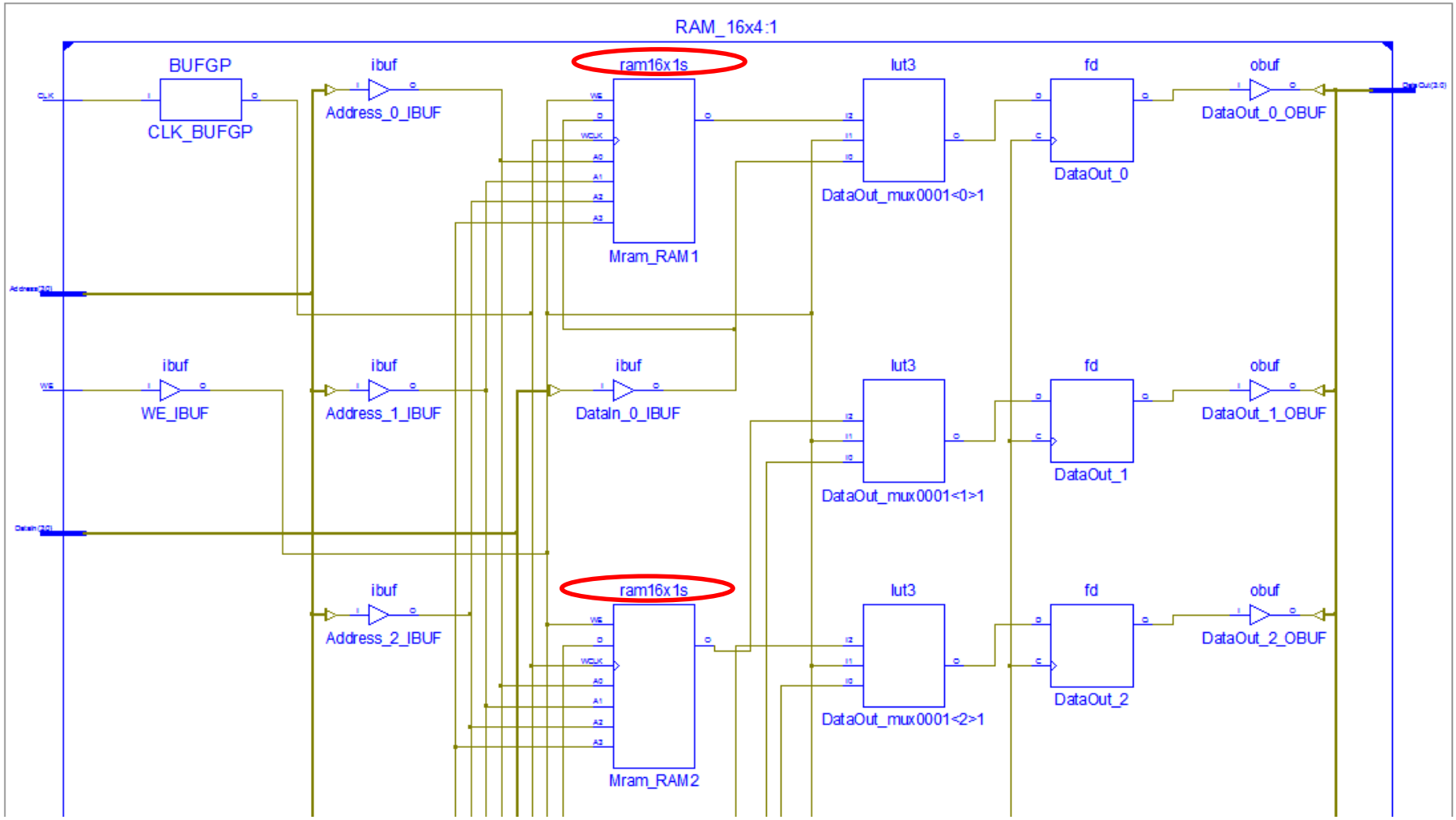
A. Modos de escritura WRITE FIRST en RAM Síncrona

Synthesis Report

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Performing bidirectional port resolution...

Synthesizing Unit <RAM_16x4>.
    Related source file is "C:/Users/usuario-uca/OneDrive/ProyectosTDC/Proyecto23/RAM_16x4.vhd".
    Found 16x4-bit single-port RAM <Mram_RAM> for signal <RAM>.
    Found 4-bit register for signal <DataOut>.
    Summary:
      inferred   1 RAM(s).
      inferred   4 D-type flip-flop(s).
Unit <RAM_16x4> synthesized.


=========================================================================
HDL Synthesis Report

Macro Statistics
# RAMs                                          : 1
 16x4-bit single-port RAM                      : 1
# Registers                                     : 1
 4-bit register                                : 1

=========================================================================
```

A. Modos de escritura WRITE FIRST en RAM Síncrona

Technology schematic



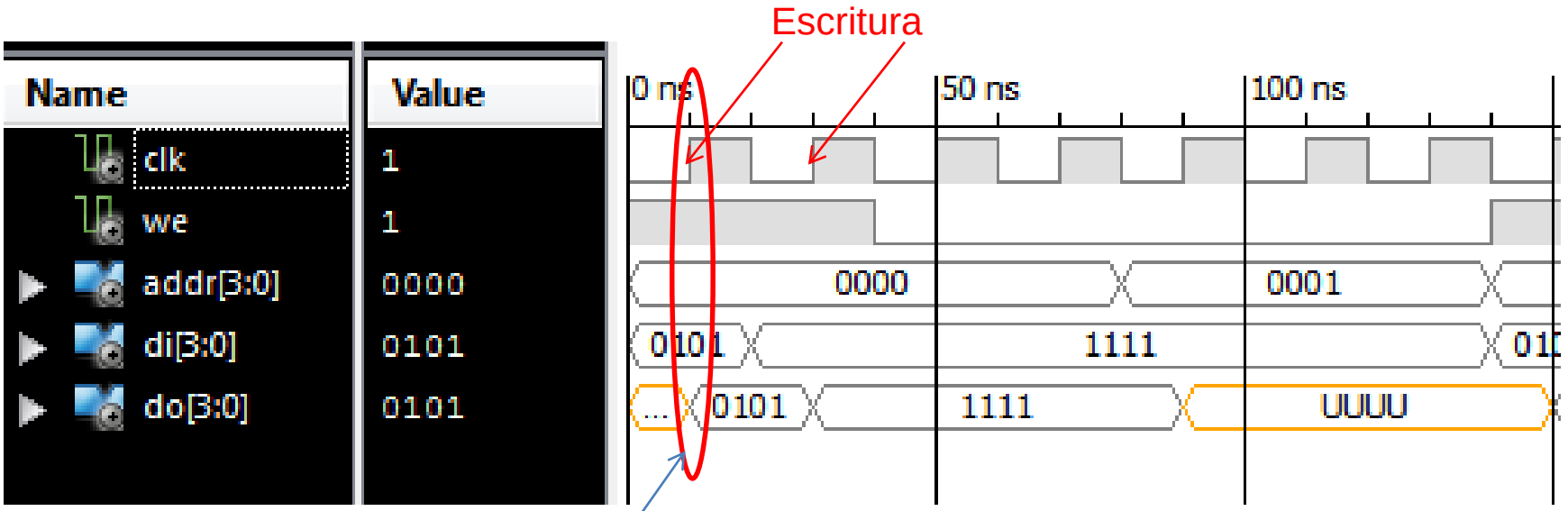
A. Modos de escritura WRITE FIRST en RAM Síncrona

Testbench

```
-- Stimulus process
stim_proc: process
Begin
 --EScribir DATO en DIR(0)
Address<="0000";
DataIn<="0101";
WE<='1';
wait for 20 ns;

 --EScribir DATO en DIR(1)
Address <="0001";
DataIn <="1111";
WE<='1';

--Leer DATO en DIR(0)
Address <="0000";
 wait for 20 ns;
WE<='0';
wait for 40 ns;

--Leer DATO en DIR(1)
Address <="0001";
wait for 20 ns;
WE<='0';
wait for 40 ns;
end process;
```

A. Modos de escritura WRITE FIRST en RAM Síncrona

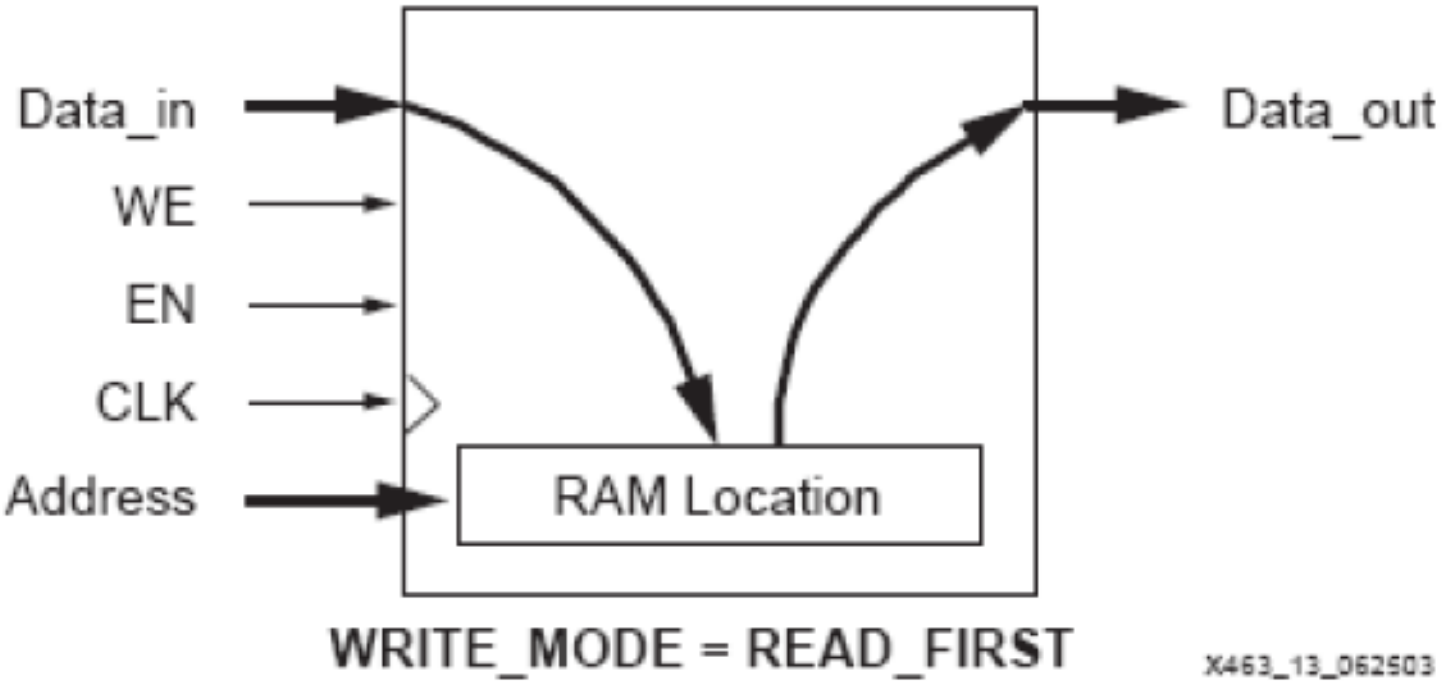
Waveform



"DataOut (do)" toma el mismo valor que se está escribiendo en el mismo flanco de CLK

La actualización de "DataOut (do)" siempre tiene lugar en el flanco positivo de CLK → Lectura Síncrona

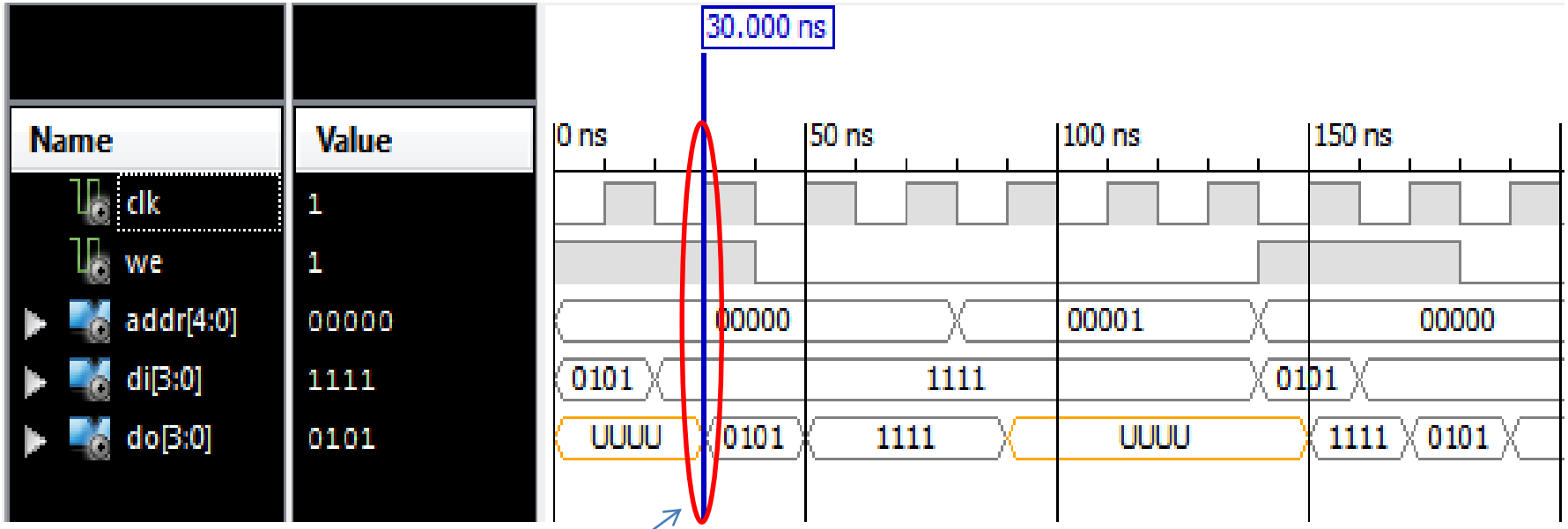
B. Modos de escritura READ FIRST en RAM Síncrona



B. Modos de escritura READ FIRST en RAM Síncrona

```
process (Clk)
Begin
 if rising_edge(Clk) then
       -- ESCRITURA síncrona
      if WE = '1' then
         RAM(to_integer(unsigned(Address))) <= DataIn;
      end if;

-- LECTURA síncrona (Tanto si hay escritura como si no)

      DataOut<= RAM(to_integer(unsigned(Address)));

 end if;
end process;
```

B. Modos de escritura READ FIRST en RAM Síncrona



"DataOut" NO toma el mismo valor que se está escribiendo.

2.4.3. RAM de escritura síncrona vs asincrona

RAM asíncrona versus síncrona

RAM Lectura Asíncrona

```
process (Clk)
Begin
 if rising_edge(Clk) then
          if WE = '1' then -- ESCRITURA
            RAM(to_integer(unsigned(Address))) <= DataIn;
          end if;
 end if;
end process;
-- LECTURA
DataOut <= RAM(to_integer(unsigned(Address)));
end Behavioral;
```

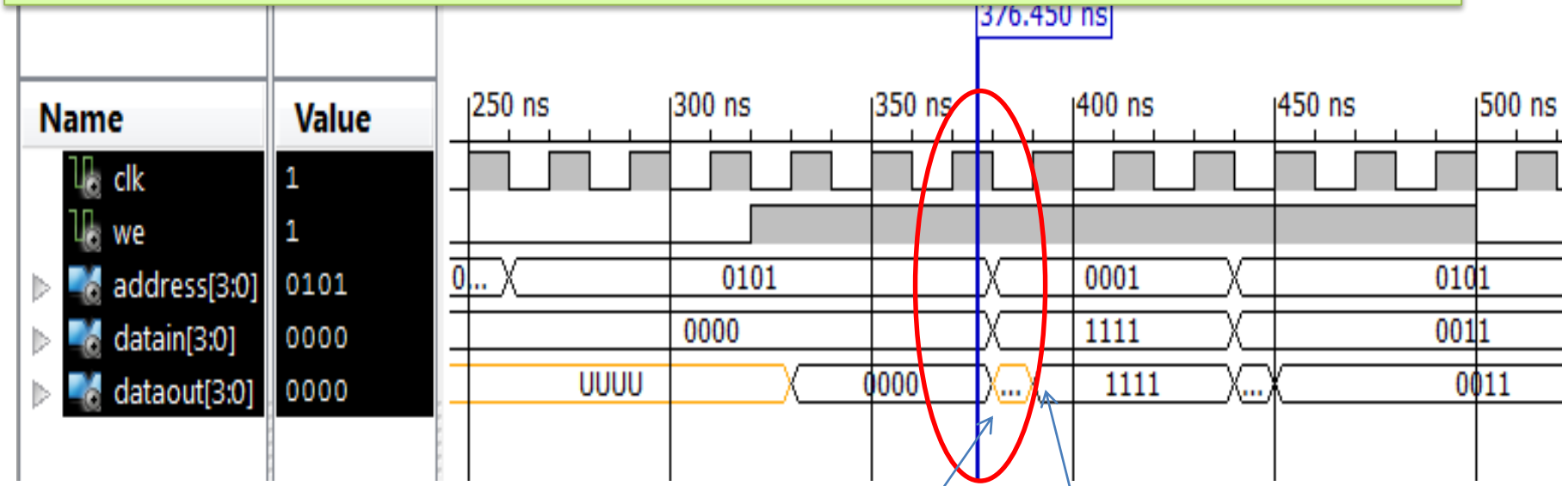
El *process* de escritura (síncrono) y la asignación de "DataOut" son concurrentes.Siempre que cambie Address (asíncrona) cambia la salida (lectura)

RAM Lectura Síncrona

```
process (Clk)
Begin
 if rising_edge(Clk) then          -- LECTURA/ESCRITURA síncrona
          if WE = '1' then -- ESCRITURA
            RAM(to_integer(unsigned(Address))) <= DataIn;
          end if;
         -- LECTURA
         DataOut <= RAM(to_integer(unsigned(Address)));
         -- Muestra salida en cada flanco de reloj
 end if;
end process;
end Behavioral;
```

RAM asíncrona versus síncrona

Simulación **RAM** Lectura **asíncrona**, sin RESET.



Cambia DataOut porque hay un cambio de dirección "Address".

Cambia DataOut porque siempre muestra el contenido de Address.

2.4.3. Descripción de ROM en VHDL

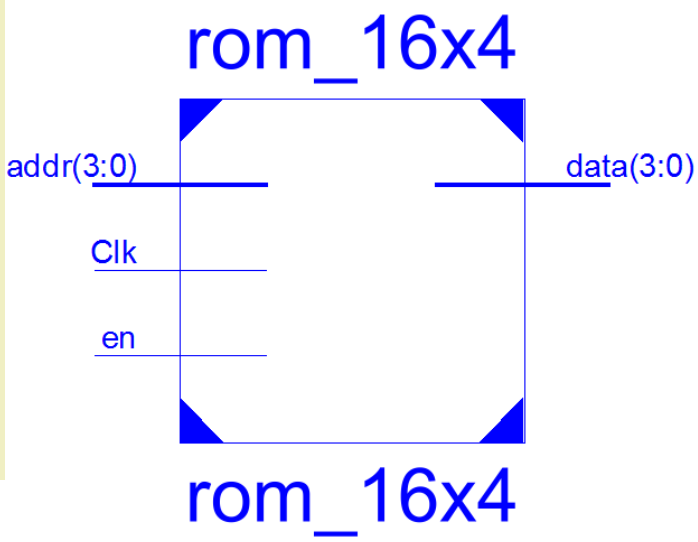
Descripción VHDL de Memoria ROM

```
entity rom_16x4 is
 port ( Clk: in std_logic;
          Enable: in std_logic;
          Address: in std_logic_vector(3 downto 0);
          DataOut: out std_logic_vector(3 downto 0));
end rom_16x4;

architecture behavioral of rom_16x4 is
     type rom_type is array (15 downto 0) of std_logic_vector(3 downto 0);
     constant ROM : rom_type:=(X"1",X"2",X"3",X"4",X"5",
     X"6",X"7",X"8",X"9",X"A",X"B",X"C",X"D",X"E",X"F",X"1");

begin

process(Clk)
begin
     if rising_edge(Clk) then
          if Enable='1' then
           DataOut <= ROM(to_integer(unsigned(Address)));
          end if;
     end if;
end process;

end behavioral;
```

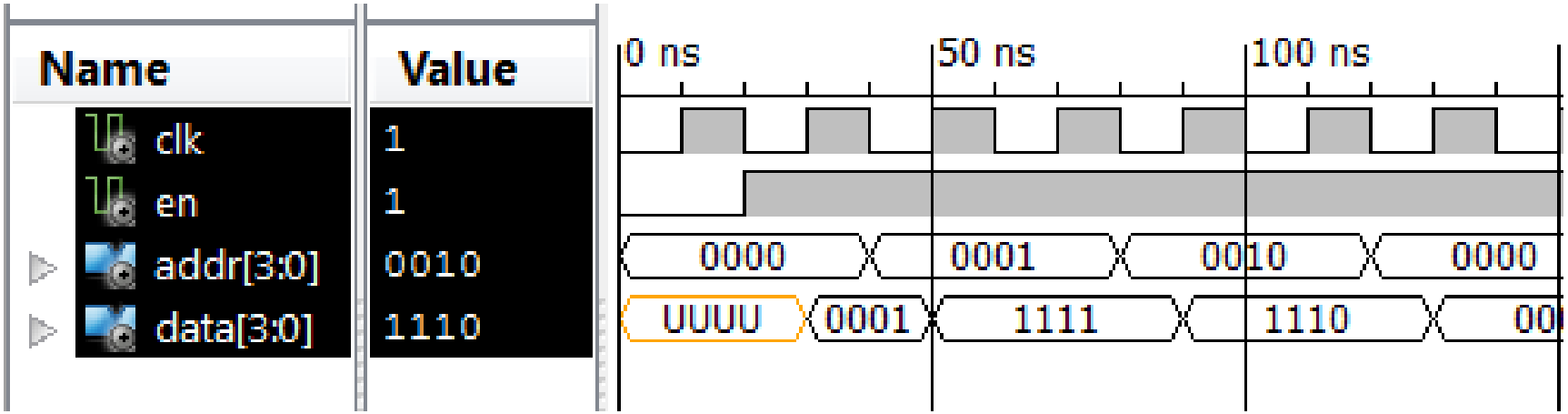
1

X"A" → Formato Hex

Enable='0' conserva el valor anterior



Descripción VHDL de Memoria ROM



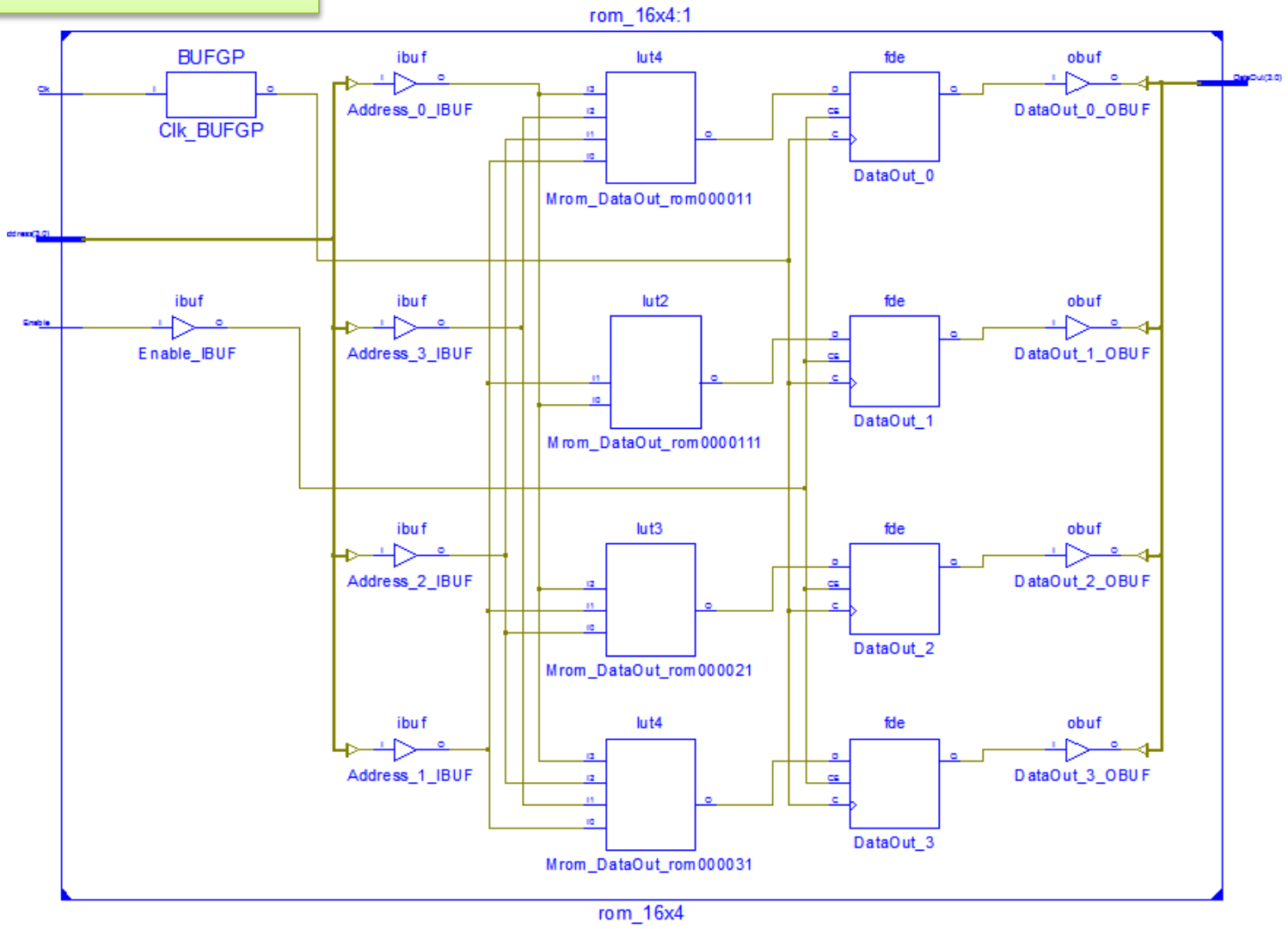
Descripción VHDL de Memoria ROM

Synthesis Report

```
=========================================================
                HDL Synthesis
=========================================================

Performing bidirectional port resolution...

Synthesizing Unit <rom_16x4>.
    Related source file is "D:/Mirian/My Dropbox/ProyectosTDC/Memor
    Found 16x4-bit ROM for signal <data$rom0000> created at line 4
    Found 4-bit register for signal <data>.
    Summary:
        inferred   1 ROM(s).
        inferred   4 D-type flip-flop(s).
Unit <rom_16x4> synthesized.

=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================


Synthesizing (advanced) Unit <rom_16x4>.
INFO:Xst:3034 - In order to maximize performance and save block RAM resc
Unit <rom_16x4> synthesized (advanced).

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                                 : 1
 16x4-bit ROM                                         : 1
# Registers                                            : 4
 Flip-Flops                                           : 4
```

Descripción VHDL de Memoria ROM

Technology schematic



Bibliografía

- **Free range VHDL**. Bryan Mealy, Fabrizio Tappero. (Creative Commons). http://www.freerangefactory.org (Mayo 2013)

- **VHDL 101.** William Kafig. Editorial Elsevier. 2011