Inferencia de Memorias para FPGAs Xilinx

Sergio Lopez-Buedo y Gustavo Sutter
{sergio.lopez-buedo, gustavo.sutter}@uam.es

Ejemplo: Memorias en Spartan-3

- Distributed RAM (y ROM)
  - Construidas con las LUTs de los CLBs
  - 16x1 de doble puerto, uno de lectura/escritura y otro de lectura
  - 64x1, 32x1 y 16x1 de simple puerto
  - Lectura asíncrona, escritura síncrona
- BlockRAM
  - Bloques separados de la matriz de CLBs
  - 18Kbits, configurables desde 16384x1, hasta 512x32. También son posibles configuraciones desde 2048x9 hasta 512x36 (paridad)
  - Doble puerto (ambos de lectura/escritura)
  - Lectura y escritura síncronas
  - La temporización entre DI y DO puede ser lectura primero, escritura primero o sin cambios

Distributed RAM: Escritura Síncrona, Lectura Asíncrona

- Las memorias son matrices bidimensionales de datos

```
type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
signal RAM : ram_type;
```

- Al especificar escritura síncrona y lectura asíncrona se fuerza el uso de distributed RAM

```
process (clk)
begin
        if (clk'event and clk = '1') then
                if (we = '1') then
                        RAM(conv_integer(a)) <= di;
                end if;
        end if;
end process;
do <= RAM(conv_integer(a));
```

- Resultado:

```
Found 32x4-bit single-port distributed RAM for signal <ram>
```

Lectura síncrona

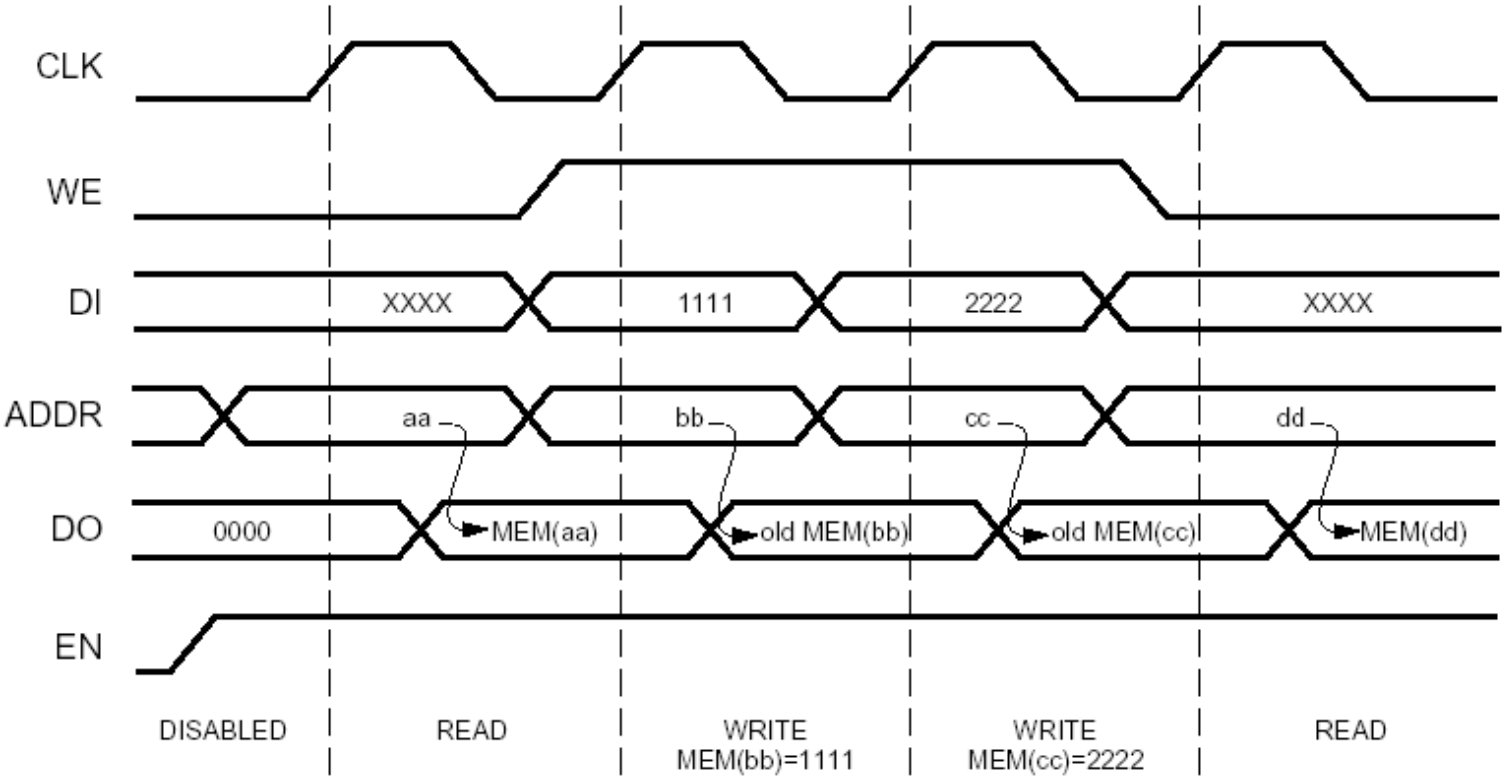
- Primera posibilidad:

```
process (clk)
begin
        if (clk'event and clk = '1') then
                if (we = '1') then
                        RAM(conv_integer(a)) <= di;
                end if;
                do <= RAM(conv_integer(a));
        end if;
end process;
```

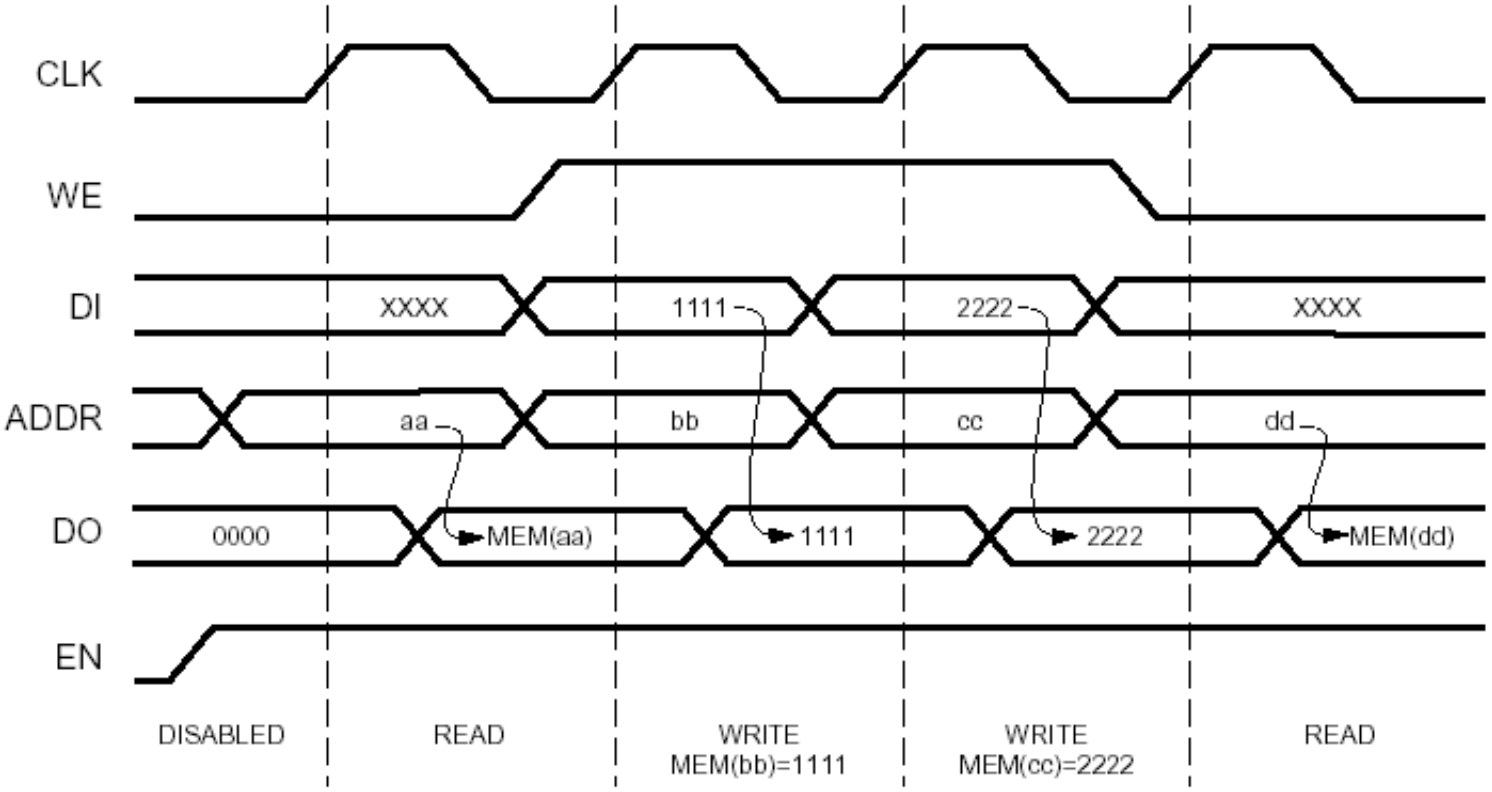
- Al examinar los resultados se comprueba que se ha generado una BlockRAM con lectura primero:

```
Found 32x4-bit single-port block RAM for signal <RAM>.
-----------------------------------------------------------------------
| mode                | read-first                        |           |
| aspect ratio        | 32-word x 4-bit                   |           |
```

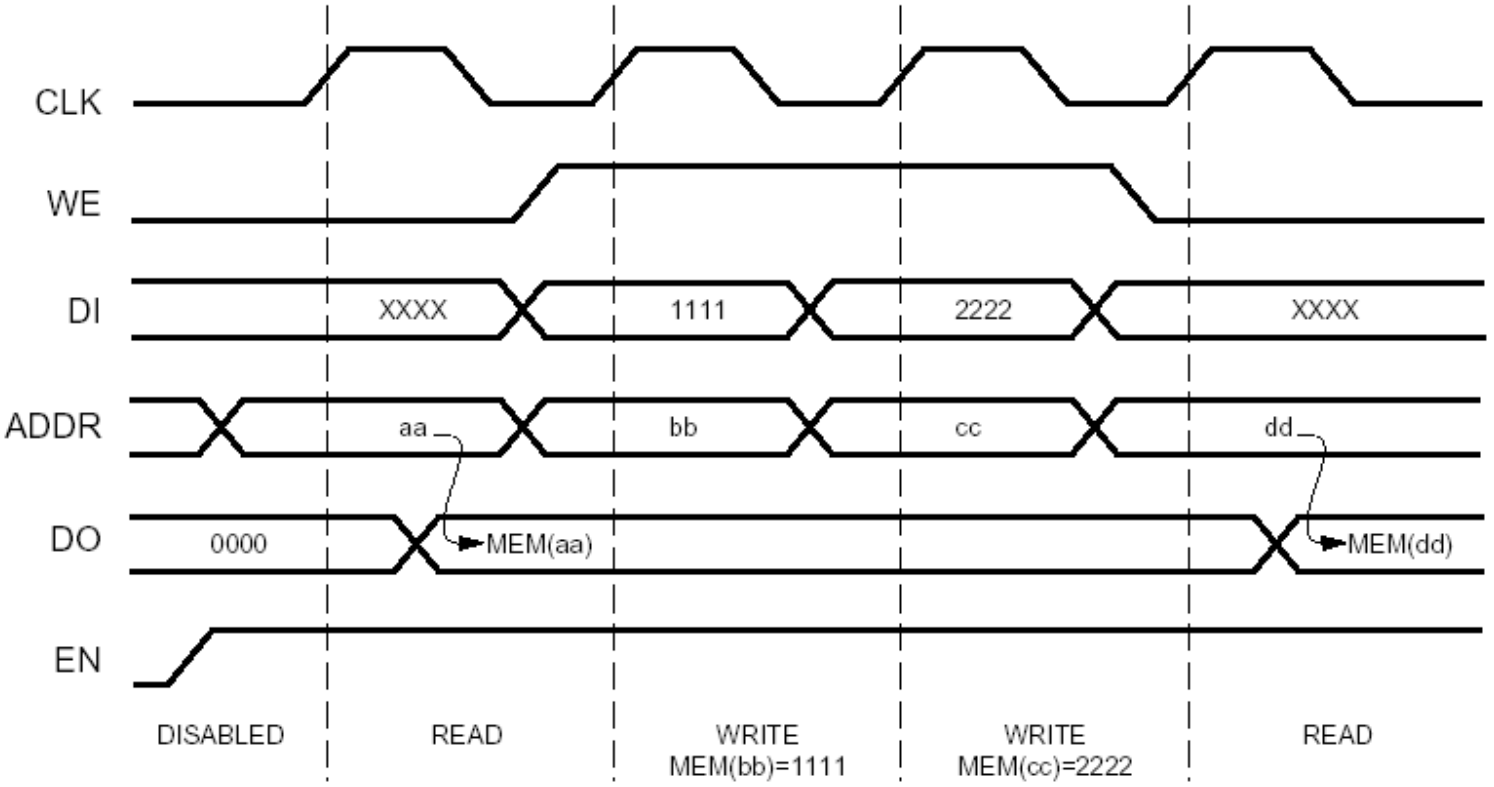
Modos de funcionamiento de la BlockRAM: Lectura primero (READ-FIRST)



Modos de funcionamiento de la BlockRAM: Escritura primero (WRITE-FIRST)



Modos de funcionamiento de la BlockRAM: Modo sin cambio (NO-CHANGE)



Lectura síncrona con escritura primero

- Probando ahora esta alternativa

```
process (clk)
begin
        if (clk'event and clk = '1') then
                if (we = '1') then
                        RAM(conv_integer(a)) <= di;
                        do <= di;
                else
                        do <= RAM(conv_integer(a));
                end if;
        end if;
end process;
```

- Ahora la escritura tiene prioridad en la BlockRAM que se ha inferido:

```
Found 32x4-bit single-port block RAM for signal <RAM>.
-----------------------------------------------------------------------
| mode                | write-first                               |           |
| aspect ratio        | 32-word x 4-bit                           |           |
```

Lectura síncrona con escritura primero (2)

- Otra alternativa es considerar que lo que se está registrando es la dirección de lectura:

```
type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
signal RAM : ram_type;
signal read_a : std_logic_vector(4 downto 0);

begin

        process (clk)
        begin
                if (clk'event and clk = '1') then
                        if (we = '1') then
                                RAM(conv_integer(a)) <= di;
                        end if;
                        read_a <= a;
                end if;
        end process;

        do <= RAM(conv_integer(read_a));
```

Lectura síncrona con modo sin cambio

- Para el modo sin cambio se puede usar esta alternativa

```
process (clk)
begin
        if (clk'event and clk = '1') then
                if (we = '1') then
                        RAM(conv_integer(a)) <= di;
                else
                        do <= RAM(conv_integer(a));
                end if;
        end if;
end process;
```

- En efecto:

```
 Found 32x4-bit single-port block RAM for signal <RAM>.
-----------------------------------------------------------------------
| mode                | no-change                         |           |
| aspect ratio        | 32-word x 4-bit                   |           |
```

Evitar la inferencia de BlockRAM

- La restricción (*constraint*) ram_style permite especificar el tipo de memoria que se inferirá.
- Se puede especificar globalmente, o como un atributo de la señal que modela la matriz de la memoria

```
type ram_type is array (31 downto 0) of std_logic_vector (3 downto 0);
signal RAM : ram_type;

attribute ram_style : string;
attribute ram_style of RAM : signal is "distributed";
```

- De esta manera se evita la inferencia de BlockRAM en el ejemplo anterior:

```
Found 32x4-bit single-port block RAM for signal <ram>.
...
Cell Usage
# RAMS                                  : 8
#      RAM16X1D                         : 8
```

ROMs

- Se pueden inferir de un *if* o un *case*:

```
process (a)
begin
        case a is
                when "0000" => do <= "0101";
                when "0001" => do <= "1010";
                when "0010" => do <= "0000";
                ...
                when "1110" => do <= "0101";
                when "1111" => do <= "1111";
                when others => do <= "XXXX";
        end case;
end process;
```

- El resultado es:

```
Macro Statistics
# ROMs                          : 1
  16x4-bit ROM                  : 1
```

ROMs: Alternativa más sencilla

- Una alternativa más compacta es modelar la ROM como una constante bidimensional:

```
architecture Behavioral of rominfr2 is

type ROM_TYPE is array(15 downto 0) of std_logic_vector(3 downto 0);
constant ROM : rom_type := (x"1", x"A", x"0", x"3",
                            x"5", x"E", x"0", x"F",
                            x"0", x"2", x"7", x"8",
                            x"C", x"3", x"1", x"0");

begin

        do <= ROM(conv_integer(a));

end Behavioral;
```

Doble puerto

- La inferencia de la memoria de doble puerto sigue el mismo esquema que los ejemplos anteriores:

```
process (clk)
begin
        if (clk'event and clk = '1') then
                if (we = '1') then
                        RAM(conv_integer(a)) <= di;
                end if;
                read_a <= a;
                read_dpra <= dpra;
        end if;
end process;

spo <= RAM(conv_integer(read_a));
dpo <= RAM(conv_integer(read_dpra));
```

- Hay que tener en cuenta las mismas limitaciones que antes con respecto al tipo de memoria que se quiere inferir (lectura asíncrona, prioridad de la lectura...)

Instanciando memorias

- Las memorias, a parte de inferirse, se pueden instanciar:

```
library unisim;
use unisim.vcomponents.all;
...
ram_256_x_16: RAMB4_S16
port map(    DI => "0000000000000000",
             EN => '1',
             WE => '0',
             RST => '0',
             CLK => clk,
             ADDR => address,
             DO => instruction(15 downto 0));
```

- La declaración del componente se puede obtener del paquete **vcomponents** de la librería **unisim**