Implementación de Memorias RAM en dispositivos FPGA

Ricardo J. Zavaleta, *ORACLE MDC*

***Abstracto*—En este documento se presenta un análisis de varias formas en que pueden implementarse memorias RAM en dispositivos FPGA. Se presentan tanto memorias distribuidas como en bloques; memorias de primero lectura, primero escritura y no-cambio; y también memorias *single port* y *dual port*.**

***Índice de términos*— FPGA, Memoria en bloque, Memoria distribuida, Memoria Dual Port, Memoria de primero escritura, Memroia Single Port, Memoria de no-cambio, Memoria de primero lectura, RAM, ROM, VHDL.**

I. Introducción

En general, llamamos memoria a aquellos dispositivos que son capaces de retener datos durante algún intervalo de tiempo. Las memorias se pueden clasificar según varios criterios. Por ejemplo, según el tiempo que se mantiene la información almacenada pueden clasificarse en RAM[1] y ROM; según su utilidad pueden clasificarse en memoria principal o memoria secundaria; etc.

Una clasificación más de las memorias es aquella que toma en cuenta la señal que se encuentra en los pines de salida tras realizarse una escritura: primero lectura, primero escritura y no cambio.

Las memorias de primero lectura son aquellas que durante una escritura muestran en las terminales de salida la información que estaba guardada en la posición que está siendo direccionada. Las memorias de primero escritura, por el contrario, muestran en las terminales de salida la misma información que está siendo escrita. Finalmente, las memorias de no cambio tienen la característica de no proporcionar ningún dato a la salida mientras se realiza la operación de escritura.

Cuando hablamos de implementar memorias en dispositivos FPGA podemos hacer una distinción más: las memorias distribuidas y las memorias en bloque. Las primeras se caracterizan por estar integradas por diferentes componentes internos esparcidos por todo el dispositivo lógico programable. Las segundas se caracterizan por utilizar bloques funcionales específicos dentro del dispositivo.

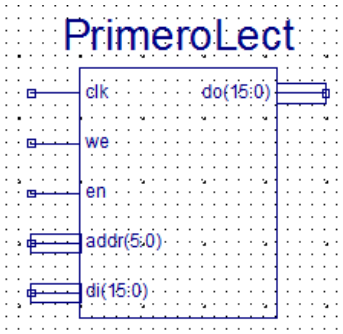
En este trabajo se pretende mostrar la forma en que se realiza la programación de memorias que caen en los dos modos de clasificación descritos en los párrafos anteriores en dispositivos FPGA en arquitecturas *single port* y *dual port*. Para cada implementación se presenta el código en lenguaje VHDL, la forma en que el dispositivo FPGA utilizará su circuitería interna y un ejemplo de su funcionamiento.

II. Memorias RAM en bloque

A. Memorias de Primero Lectura (Single Port)

Como ya se explicó en la introducción, las memorias de primero lectura son aquellas que durante la fase de escritura muestran en las terminales de salida el dato que se encontraba en la posición direccionada.

La siguiente imagen representa los distintos pines que se requieren en una memoria de este tipo.



Observamos que se requiere una señal de reloj, que permite que el circuito funcione ante un flanco; una señal de lectura/escritura para identificar la operación que se requiere realizar; una señal de habilitación, para indicar al dispositivo que entrará en funcionamiento; una señal de dirección, que representa la posición de memoria con la que se trabajará; y el dato de entrada, en el caso de realizar una operación de escritura. Hay un pin adicional que es el dato de salida utilizada en operaciones de lectura.

[1] Varios autores, entre los que se encuentra Stallings[1], están de acuerdo en que el término RAM utilizado para nombrar a las memorias que requieren mantenerse alimentadas eléctricamente para conservar su información es inadecuado. RAM hace referencia a que los datos pueden ser direccionados a velocidad constante sin importar su ubicación. A esto se le conoce como acceso aleatorio. De esta manera, las memorias ROM también son de acceso aleatorio.

El siguiente listado implementa una memoria de primero lectura de tipo *single-port*.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PrimeroLect is
    port( clk, we, en:   in STD_LOGIC;
          addr:          in STD_LOGIC_VECTOR(5 downto 0);
          di:            in STD_LOGIC_VECTOR(15 downto 0);
          do:               out STD_LOGIC_VECTOR(15 downto 0));
end PrimeroLectura;

architecture Behavioral of PrimeroLect is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then
                    RAM(conv_integer(addr)) <= di;
                end if;

                do <= RAM(conv_integer(addr));
            end if;
        end if;
    end process;

end Behavioral;
```

El código básicamente asigna siempre a la salida el dato que se encuentra en la posición de memoria direccionada, el cual puede cambiar máximo en cada ciclo de reloj en este caso.

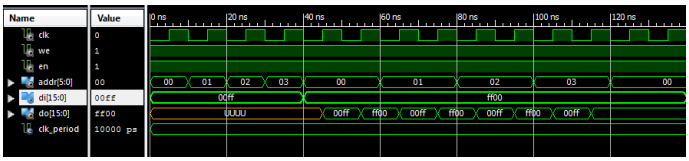
Tras sintetizar este código, el resumen muestra la siguiente información.

PrimeroLect Project Status			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	PrimeroLect	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	3584	0%
Number of bonded IOBs	41	195	21%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

Nótese que en este caso se están utilizando *BRAMs*[2] pero no *Slices*. Este es el indicador de que el resultado es una memoria en bloque.

El resultado de simular el código es el siguiente:

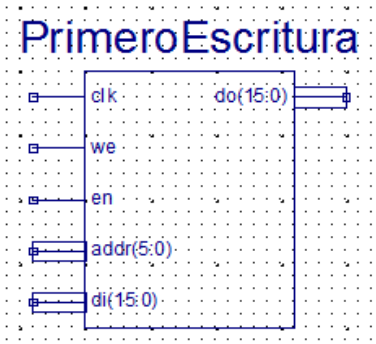


Durante los cuatro primeros ciclos de reloj(0[ns] – 40[ns]) se direccionaron las posiciones de memoria 0x00, 0x01, 0x02 y 0x03 para realizar una escritura del dato 0x00FF sobre ellas (nótese que la señal de escritura está habilitada). Durante este periodo de tiempo se tiene una señal indefinida a la salida. Dado que se trata de una memoria de primera lectura, lo que se muestra a la salida es el dato que se encontraba previamente en la dirección seleccionada. Como la memoria no estaba inicializada, la señal que se tiene es indefinida.

Posteriormente se realizan reescrituras sobre las mismas direcciones de memoria (40[ns] – 120[ns]). Cada reescritura se realiza durante dos ciclos de reloj. Si se observa cuidadosamente, se puede ver que para cada posición de memoria, durante el primer ciclo se muestra el dato que se encontraba previamente en dicha dirección. Para el segundo ciclo de reloj, el dato que se observa es el que se escribió en el ciclo anterior. Por ejemplo, para la dirección 0x00 a los 0[ns] se escribe el dato 0x00FF. Posteriormente a los 40[ns] se escribe el dato 0xFF00 pero observamos que a la salida tenemos el dato 0x00FF. Finalmente, a los 50[ns] se sigue escribiendo el dato 0xFF00 y a la salida se obtiene 0xFF00 que fue precisamente lo que se escribió en el ciclo de reloj anterior.

B. Memorias de Primero Escritura (Single Port)

A diferencia de las anteriores, las memorias de primero escritura activan en sus terminales de salida el dato que se está escribiendo actualmente. El esquema siguiente muestra que los pines para una memoria de este tipo son los mismos que para una de tipo primero lectura.



En el siguiente código hay que observar que, a diferencia del anterior, a la salida se le asigna el mismo dato que se está guardando en el caso de una operación de escritura, mientras que en el caso de una operación de lectura se asigna el dato de la posición seleccionada.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

[2] BRAM: Block RAM

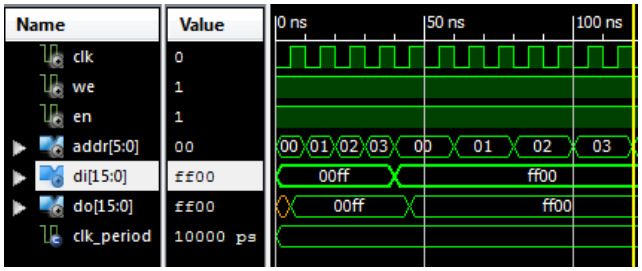
```
entity PrimeroEscritura is
    port( clk, we, en:   in STD_LOGIC;
          addr:          in STD_LOGIC_VECTOR(5 downto 0);
          di:            in STD_LOGIC_VECTOR(15 downto 0);
          do:                out STD_LOGIC_VECTOR(15 downto 0));
end PrimeroEscritura;

architecture Behavioral of PrimeroEscritura is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then
                    RAM(conv_integer(addr)) <= di;
                    do <= di;
                else
                    do <= RAM(conv_integer(addr));
                end if;
            end if;
        end if;
    end process;

end Behavioral;
```

Al igual que en la memoria presentada en el apartado anterior, se observa que esta implementación es en bloque puesto que unicamente utiliza *BRAMs*.

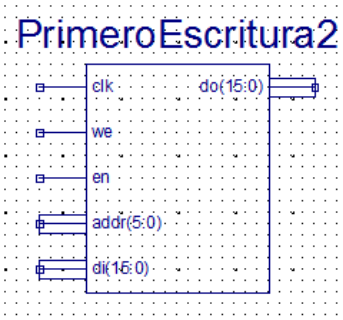
PrimeroLect Project Status (07/21/2013 - 17:31:54)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	PrimeroEscritura	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	3584	0%
Number of bonded IOBs	41	195	21%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

El siguiente diagrama de tiempos muestra su funcionamiento. De la misma manera que en el ejemplo de la memoria de primero lectura, se utilizan cuatro ciclos de reloj(0[ns] – 40[ns]) para inicializar las primeras cuatro celdas. Sin embargo, en este caso se observa que en todo momento se está leyendo a la salida el mismo dato que se está escribiendo: 0x00FF.



Existe otra manera de hacer la implementación utilizando los mismos pines. El código es a grandes rasgos una reestructuración de manera que la salida se asigna en un solo lugar del listado.



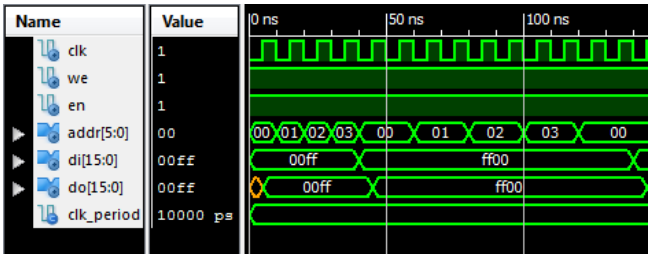

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity PrimeroEscritura2 is
    port( clk, we, en:   in STD_LOGIC;
          addr:          in STD_LOGIC_VECTOR(5 downto 0);
          di:            in STD_LOGIC_VECTOR(15 downto 0);
          do:                out STD_LOGIC_VECTOR(15 downto 0));
end PrimeroEscritura2;

architecture Behavioral of PrimeroEscritura2 is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
    signal read_addr: STD_LOGIC_VECTOR(5 downto 0);
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then
                    RAM(conv_integer(addr)) <= di;
                end if;

                read_addr <= addr;

            end if;
        end if;
    end process;

    do <= RAM(conv_integer(read_addr));
end Behavioral;
```

Nuevamente se observa que se trata de una memoria en bloques.

PrimeroLect Project Status (07/21/2013 - 17:35:52)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	PrimeroEscritura2	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

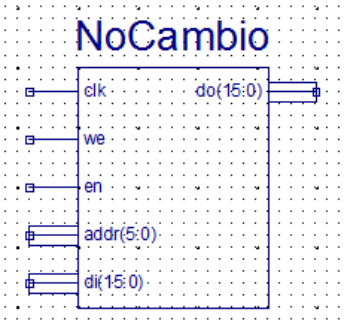
Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	3584	0%
Number of bonded IOBs	41	195	21%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

Se observa también que el comportamiento es el mismo.



C. Memorias de No-Cambio (Single Port)

Una memoria de no cambio es aquella en la que la señal en las terminales de salida es indefinida durante el proceso de escritura. Una memoria de este tipo puede implementarse con la misma cantidad y tipos de pines que los usados en las memorias anteriores.



El código de una memoria de este tipo tiene la característica diferenciar la señal que se tiene en el pin de selección de operación. Si la operación es de lectura se escribe el dato en los pines de salida, pero si es una operación de escritura los pines de salida no contienen nada, es decir, la salida es indefinida.

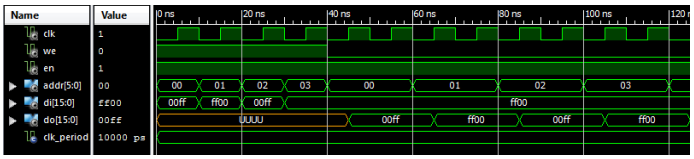
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity NoCambio is
    port( clk, we, en:   in STD_LOGIC;
          addr:          in STD_LOGIC_VECTOR(5 downto 0);
          di:            in STD_LOGIC_VECTOR(15 downto 0);
          do:            out STD_LOGIC_VECTOR(15 downto 0));
end NoCambio;

architecture Behavioral of NoCambio is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if en = '1' then
                if we = '1' then
                    RAM(conv_integer(addr)) <= di;
                else
                    do <= RAM(conv_integer(addr));
                end if;
            end if;
        end if;
    end process;

end Behavioral;
```

Nuevamente, se trata de una memoria en bloques puesto que únicamente se utilizan *BRAMs*.

PrimeroLect Project Status (07/21/2013 - 17:38:52)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	NoCambio	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	3584	0%
Number of bonded IOBs	41	195	21%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

El siguiente diagrama muestra el funcionamiento de este tipo de memoria. Se observa que durante los ciclos de escritura (0[ns] – 40[ns]) la señal de salida es siempre indefinida. Si queremos tener algún dato en los pines de salida es necesario activar explícitamente la señal de lectura (40[ns]).



III. Memorias RAM distribuidas

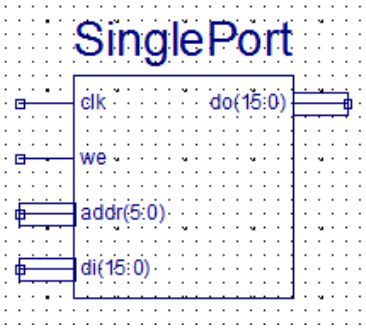
A. Memorias Distribuidas Single Port

Se describirán ahora las memorias conocidas como distribuidas. Se utilizarán como ejemplo una memoria de tipo *single port* y una de tipo *dual port*.

Las memorias distribuidas son memorias que al implementarse en la FPGA utilizan diferentes circuitos internos que se encuentran a lo largo y ancho del dispositivo lógico programable.

Todas las memorias que se han visto hasta este momento son de tipo *single port*, es decir, son memorias que tienen un solo conjunto de pines de entrada y un solo conjunto de pines de salida.

El siguiente esquema muestra la disposición de pines de una memoria distribuida *single port*. Nótese que es muy parecida a los circuitos presentados anteriormente. La única diferencia en este caso es la falta del pin de habilitación, sin embargo, los circuitos anteriores pudieron también funcionar sin dicho pin, es decir, dicha señal puede considerarse como innecesaria para nuestros fines.



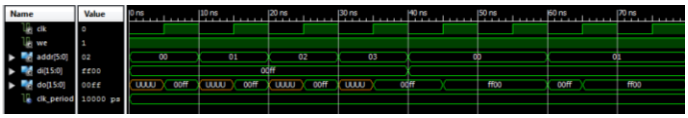
La diferencia con esta memoria desde el punto de vista del código está en que no se tiene que verificar la señal de habilitación y, además, que la operación de lectura es asíncrona.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SinglePort is
    port( clk, we:    in STD_LOGIC;
          addr:       in STD_LOGIC_VECTOR(5 downto 0);
          di:         in STD_LOGIC_VECTOR(15 downto 0);
          do:             out STD_LOGIC_VECTOR(15 downto 0));
end SinglePort;

architecture Behavioral of SinglePort is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if we = '1' then
                RAM(conv_integer(addr)) <= di;
            end if;
        end if;
    end process;

    do <= RAM(conv_integer(addr));

end Behavioral;
```

En este caso se observa que, a diferencia de todos los ejemplos anteriores, ahora no se utilizaron *BRAMs* sino más bien *Slices*.

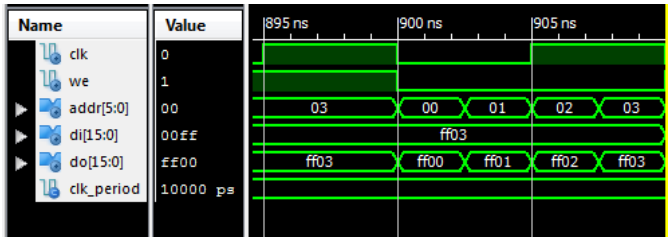
PrimeroLect Project Status (07/21/2013 - 17:41:53)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	SinglePort	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	42	3584	1%
Number of 4 input LUTs	82	7168	1%
Number of bonded IOBs	40	195	20%
Number of GCLKs	1	24	4%



Primeramente, nótese que esta memoria no es de tipo no-cambio puesto que durante las operaciones de escritura se registran señales en los pines de salida. Observando el ciclo de reloj que empieza en 40[ns], cuando se escribe el dato 0xFF00, inmediatamente se registra a la salida ese mismo valor a pesar de que anteriormente se tenía escrito 0x00FF: se trata de una memoria de primero escritura.

Para demostrar el comportamiento asíncrono mencionado, considérese el siguiente diagrama de tiempo:



En el ciclo que inicia a los 900[ns] se hacen cuatro cambios en los pines de dirección y, sin tener que esperar al siguiente flanco de reloj, se obtiene el contenido de las celdas.

Un pequeño experimento consistente en eliminar esta asincronía muestra que es precisamente esta característica la que convierte el circuito en una memoria distribuida:

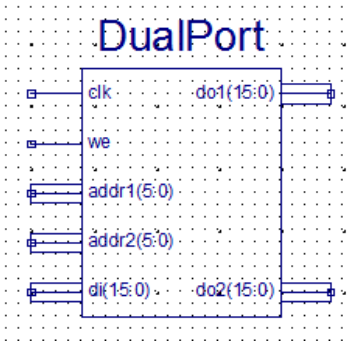
```
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if we = '1' then
                RAM(conv_integer(addr)) <= di;
            else
                do <= RAM(conv_integer(addr));
            end if;
        end if;
    end process;

end Behavioral;
```

DualPort Project Status (07/23/2013 - 00:08:18)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	SinglePort	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	1 Warning (1 new)
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	0	3584	0%
Number of bonded IOBs	40	195	20%
Number of BRAMs	1	20	5%
Number of GCLKs	1	24	4%

B. Memorias Distribuidas Dual Port

Finalmente se presenta una memoria distribuida de tipo *dual port*. Este tipo de memorias se caracteriza por tener dos conjuntos de pines de entrada y dos conjuntos de pines de salida. El siguiente esquema del circuito clarifica esta noción.



Si se observa el código, es posible darse cuenta de que la diferencia está en que a la salida siempre se tienen dos datos: el direccionado por los pines *addr1* y el direccionado por los pines *addr2*. Nuevamente se trata de una memoria donde la lectura es asíncrona y, por otra parte, se observa que la escritura solamente es de un dato a la vez.

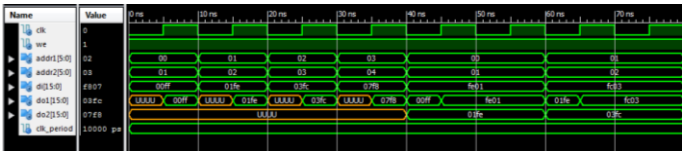
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity DualPort is
    port( clk, we:      in STD_LOGIC;
          addr1:        in STD_LOGIC_VECTOR(5 downto 0);
          addr2:        in STD_LOGIC_VECTOR(5 downto 0);
          di:           in STD_LOGIC_VECTOR(15 downto 0);
          do1:          out STD_LOGIC_VECTOR(15 downto 0);
          do2:          out STD_LOGIC_VECTOR(15 downto 0));
end DualPort;

architecture Behavioral of DualPort is
    type ram_type is array(63 downto 0) of STD_LOGIC_VECTOR(15
downto 0);
    signal RAM: ram_type;
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if we = '1' then
                RAM(conv_integer(addr1)) <= di;
            end if;
        end if;
    end process;

    do1 <= RAM(conv_integer(addr1));
    do2 <= RAM(conv_integer(addr2));

end Behavioral;
```

El resumen de la implementación muestra que se trata de una memoria distribuida que utiliza 72 slices.

PrimeroLect Project Status (07/21/2013 - 17:44:41)			
Project File:	TareaMemorias.xise	**Parser Errors:**	No Errors
Module Name:	DualPort	**Implementation State:**	Synthesized
Target Device:	xc3s400a-4ft256	**• Errors:**	No Errors
Product Version:	ISE 14.1	**• Warnings:**	No Warnings
Design Goal:	Balanced	**• Routing Results:**	
Design Strategy:	Xilinx Default (unlocked)	**• Timing Constraints:**	
Environment:	System Settings	**• Final Timing Score:**	

Device Utilization Summary (estimated values)			[-]
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	72	3584	2%
Number of 4 input LUTs	196	7168	2%
Number of bonded IOBs	62	195	31%
Number of GCLKs	1	24	4%

El diagrama de tiempo muestra nuevamente que se trata de una memoria de primero escritura como se puede observar en el flanco de reloj a los 5[ns] donde se escribe en la dirección 0x00 el dato 0x00FF e inmediatamente se ve reflejado en los pines de salida. También se observa que la salida *do2* no tiene asociados pines de donde se pueda escribir directamente: la escritura solamente se puede hacer por únicamente un conjunto de pines.



IV. Conclusiones

La siguiente tabla muestra un resumen de los recursos del FPGA utilizados por cada implementación de memoria estudiada:

Implementación	***Slices***	***LUTs***	***BRAMs***
Primero Lectura	0	0	1
Primero Escritura 1	0	0	1
Primero Escritura 2	0	0	1
No Cambio	0	0	1
Distribuida single port	42	82	0
Distribuida dual port	72	196	0

Esto indica que efectivamente los primeros cuatro ejemplos son memorias en bloques, mientras que los dos últimos son memorias distribuidas.

Se puede concluir entonces que tanto las memorias en bloques como las memorias distribuidas pueden ser implementadas en dispositivos FPGA. El dispositivo realizará la implementación basándose tanto en el código como en la disponibilidad de elementos BRAM o de sus slices.

Desde otra perspectiva, es posible también implementar memorias *single port* y *dual port*. Esta distinción es completamente determinada por el número de conjunto de entradas y salidas definidas en el código.

Finalmente, es posible implementar memorias de primero lectura, primero escritura y no cambio. Al igual que en la división *single* y *dual port*, esta distinción también está determinada por el código a sintetizar.

V. Referencias

[1] Stallings, William, "Organización y arquitectura de computadoras" Prentice Hall, 2006.