Memoria

Diseño de Sistemas con FPGA

Patricia Borensztejn

Memoria (Spartan-3)

- La Spartan-3 conjuntamente con el S3 board proveen varias opciones de almacenamiento:
  - Registros (FF): aprox. 4.5K bits (4320 LC) embebidos en las LC e I/O buffers
  - RAM distribuida: construida con las LUTS. Puede haber hasta 30K bits en la XC3S200. (Compite por recursos con la lógica. Su uso está rentringido a aplicaciones que requieran poco almacenamiento).
  - Block RAM: es un bloque especial separado de las celdas lógicas.
    - Cada bloque consiste de 16K bits (mas 2K bits de paridad) organizado de diferentes formas: desde 16K bits a 512 por 32 bits.
    - En Spartan hay 12 BRAMS.
    - Se pueden utilizar para: FIFO, una LUT grande, etc.

    Tanto la memoria distribuida como la BRAM vienen provistas de una interface síncrona que permite su utilización sin que sea necesario un controlador de memoria.
  - Memoria SRAM Externa: 8Mbits, configuradas como dos 256K por 16 SRAM chips.

Familia Spartan3E

- Memoria Distribuida

Table 9: Spartan-3E CLB Resources

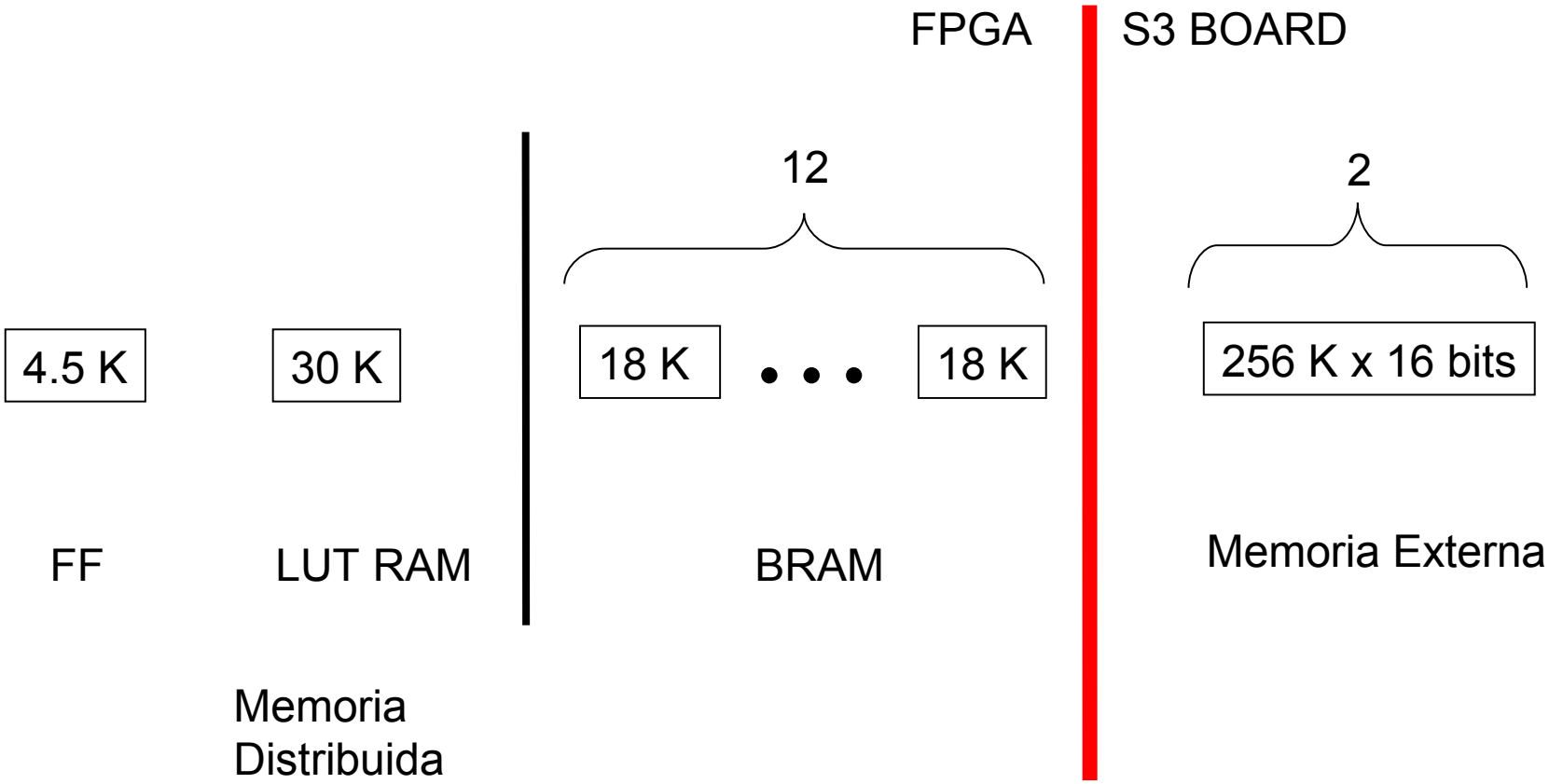
Device	CLB Rows	CLB Columns	CLB Total[(1)]	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S100E	22	16	240	960	1,920	2,160	960	15,360
XC3S250E	34	26	612	2,448	4,896	5,508	2,448	39,168
XC3S500E	46	34	1,164	4,656	9,312	10,476	4,656	74,496
XC3S1200E	60	46	2,168	8,672	17,344	19,512	8,672	138,752
XC3S1600E	76	58	3,688	14,752	29,504	33,192	14,752	236,032

- Block Ram

Table 21: Number of RAM Blocks by Device

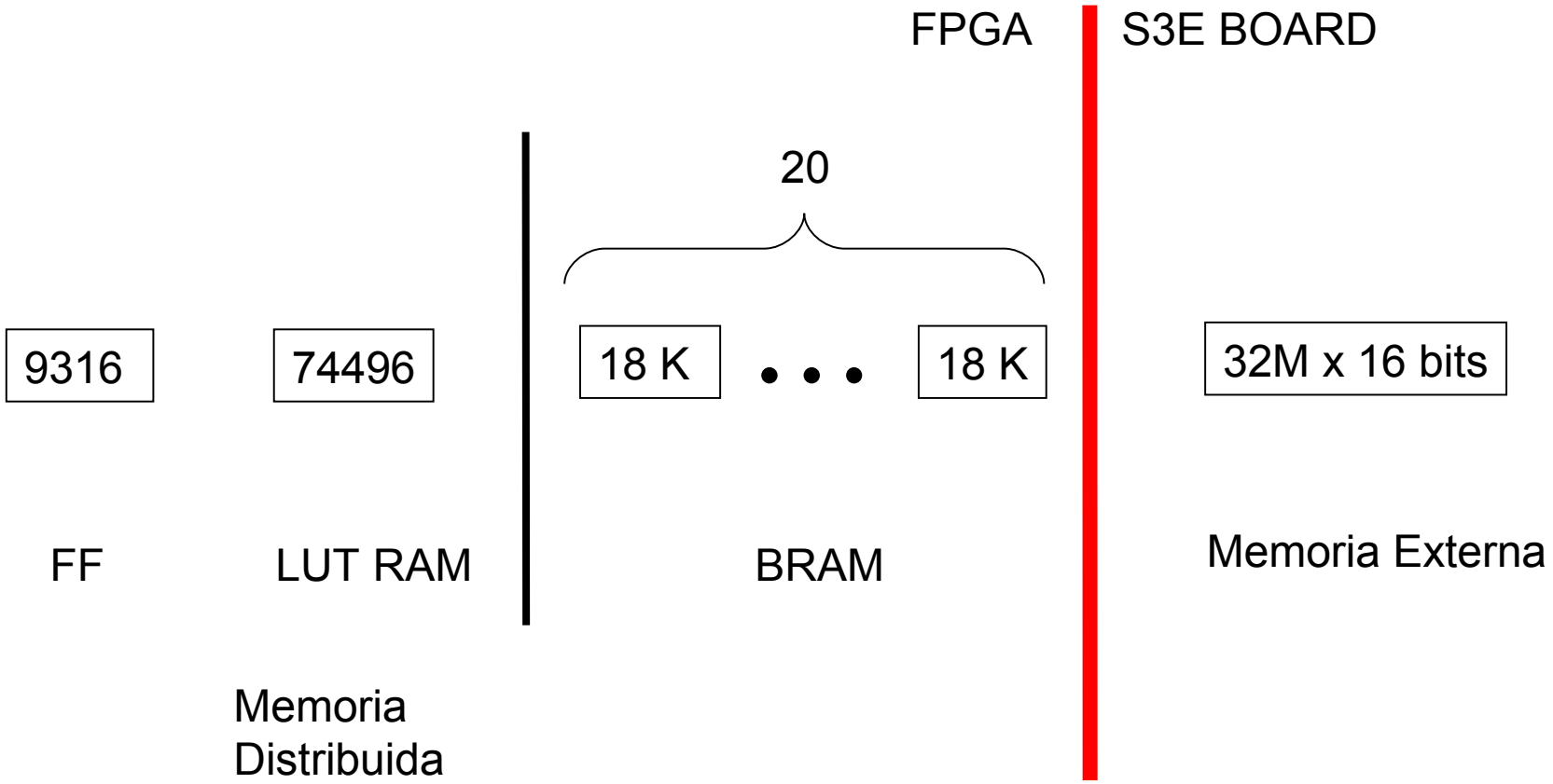
Device	Total Number of RAM Blocks	Total Addressable Locations (bits)	Number of Columns
XC3S100E	4	73,728	1
XC3S250E	12	221,184	2
XC3S500E	20	368,640	2
XC3S1200E	28	516,096	2
XC3S1600E	36	663,552	2

Solo números



TODA LA MEMORIA ES ESTÁTICA

Solo números: S3E



TODA LA MEMORIA ES ESTÁTICA

LUT RAM o Memoria Distribuida

- Cada CLB está formado por cuatro SLICES interconectados, agrupados por pares: dos de tipo SLICEM y dos de tipo SLICEL.
- Los slice tipo M son los que pueden implementar lógica o memoria. Los de tipo L sólo implementan lógica.

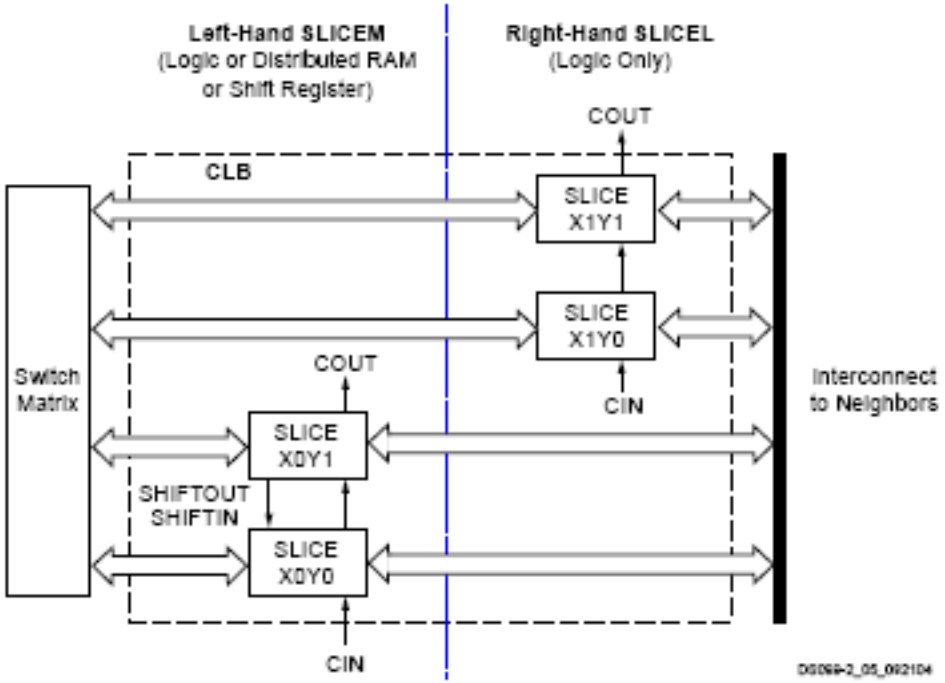


Figure 5-3: **Arrangement of Slices within the CLB**

LUT RAM

- Por eso, con aprox. 4320 LC sólo hay disponible 32 K bits de memoria distribuida.

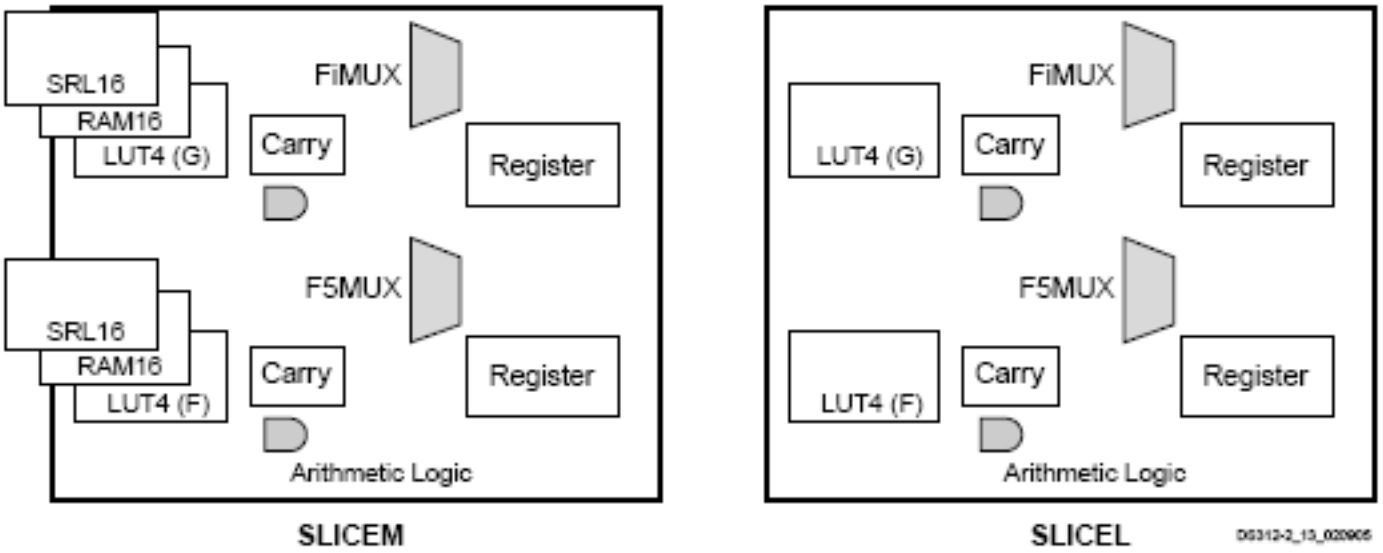


Figure 5-4: **Resources in a Slice**

The SLICEM pair supports two additional functions:

- Two 16x1 distributed RAM blocks, RAM16
- Two 16-bit shift registers, SRL16

LUT RAM

- Características:
  - RAM síncrona de:
    - 16 x 1 bit
    - Write síncrono
    - Read asíncrono
    - Los LUT´s de los SLICEM se pueden poner en cascada para formar:
      - RAM de 16 x 4 bits
      - RAM de 32 * 2 bits
      - RAM de 64 * 1 bit
    - Las señales adicionales de dirección para los casos de 32 y 64 se obtienen de entradas especiales (Bx y By)

LUT RAM

- Pueden ser single o dual port (direcciones)
- Para implementar las dual port se usan los dos LUTs del SLICEM:
  - LUT 1 : tiene UN port de lectura y de escritura
  - LUT 2: tiene UN port de solo lectura
  - La operación de escritura es simultánea en las dos LUTs.

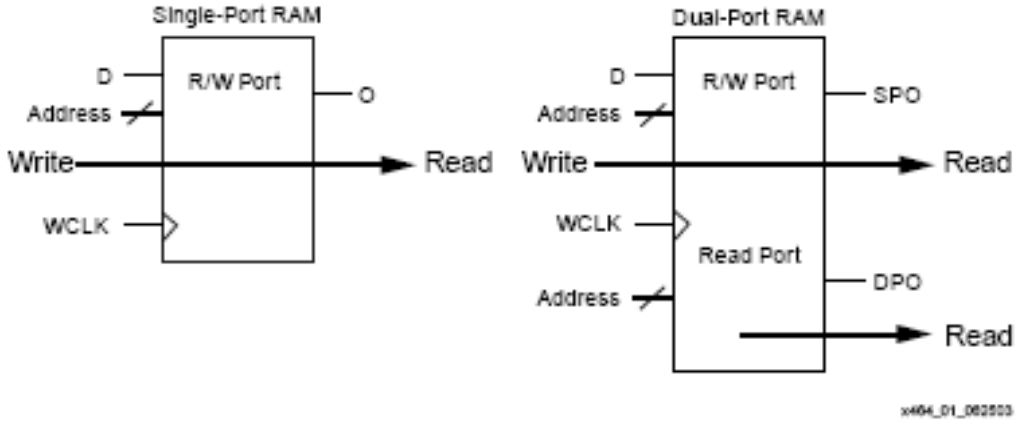


Figure 6-1: **Single-Port and Dual-Port Distributed RAM**

LUT RAM doble puerto

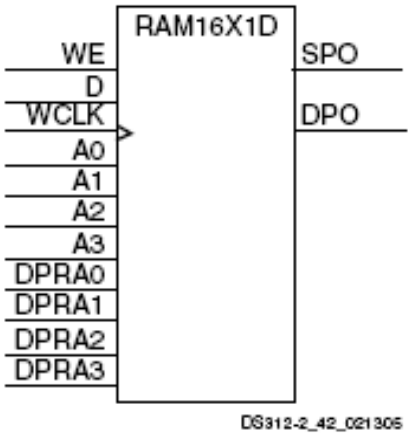


Figure 28: **Dual-Port RAM Component**

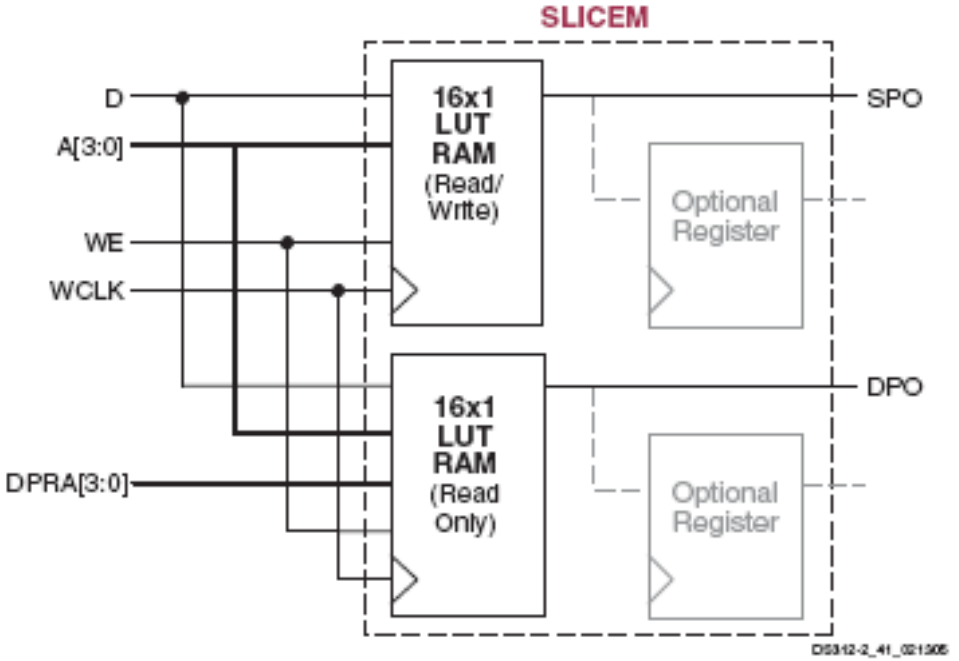
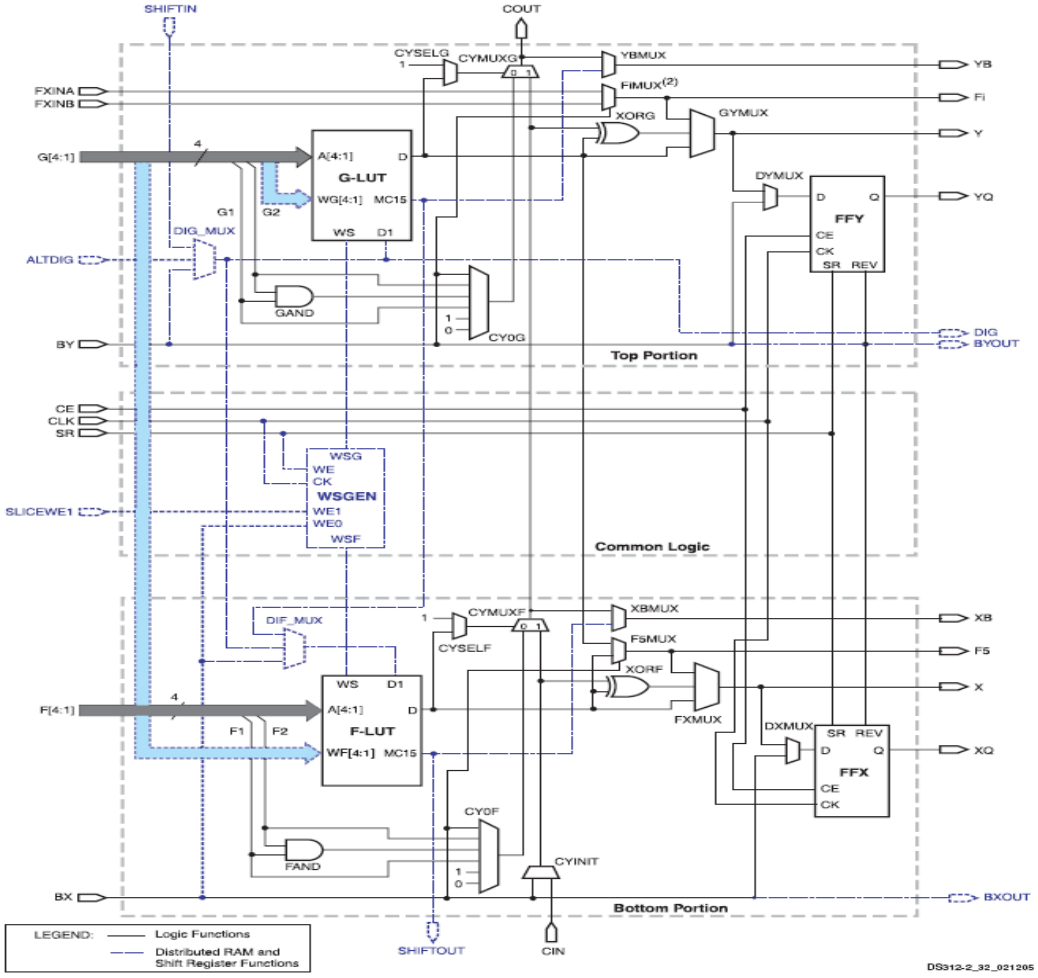


Figure 27: **RAM16X1D Dual-Port Usage**

Para el curioso…



Notes:

1. Options to invert signal polarity as well as other options that enable lines for various functions are not shown.
2. The index i can be 6, 7, or 8, depending on the slice. The upper SLICEL has an F8MUX, and the upper SLICEM has an F7MUX. The lower SLICEL and SLICEM both have an F6MUX.

Figure 16: **Simplified Diagram of the Left-Hand SLICEM**

LUT RAM : primitivas

Table 6-4: **Single-Port and Dual-Port Distributed RAMs**

Primitive	RAM Size (Depth x Width)	Type	Address Inputs
RAM16X1S	16 x 1	Single-port	A3, A2, A1, A0
RAM32X1S	32 x 1	Single-port	A4, A3, A2, A1, A0
RAM64X1S	64 x 1	Single-port	A5, A4, A3, A2, A1, A0
RAM16X1D	16 x 1	Dual-port	A3, A2, A1, A0

Table 6-6: **Wider Library Primitives**

Primitive	RAM Size (Depth x Width)	Data Inputs	Address Inputs	Data Outputs
RAM16X2S	16 x 2	D1, D0	A3, A2, A1, A0	O1, O0
RAM32X2S	32 x 2	D1, D0	A4, A3, A2, A1, A0	O1, O0
RAM16X4S	16 x 4	D3, D2, D1, D0	A3, A2, A1, A0	O3, O2, O1, O0

Templates para instanciación

- **Edit ->Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Device Primitive Instantiation->FPGA ->RAM/ROM -> Distributed RAM**

```
// RAM64X1S: 64 x 1 positive edge write, asynchronous read single-port distributed RAM
//           Virtex-II/II-Pro/4/5, Spartan-3/3E/3A
// Xilinx HDL Language Template, version 10.1

  RAM64X1S #(
     .INIT(64'h0000000000000000) // Initial contents of RAM
  ) RAM64X1S_inst (
     .O(O),         // 1-bit data output
     .A0(A0),       // Address[0] input bit
     .A1(A1),       // Address[1] input bit
     .A2(A2),       // Address[2] input bit
     .A3(A3),       // Address[3] input bit
     .A4(A4),       // Address[4] input bit
     .A5(A5),       // Address[5] input bit
     .D(D),         // 1-bit data input
     .WCLK(WCLK),   // Write clock input
     .WE(WE)        // Write enable input
  );
```

Numeración de los SLICES

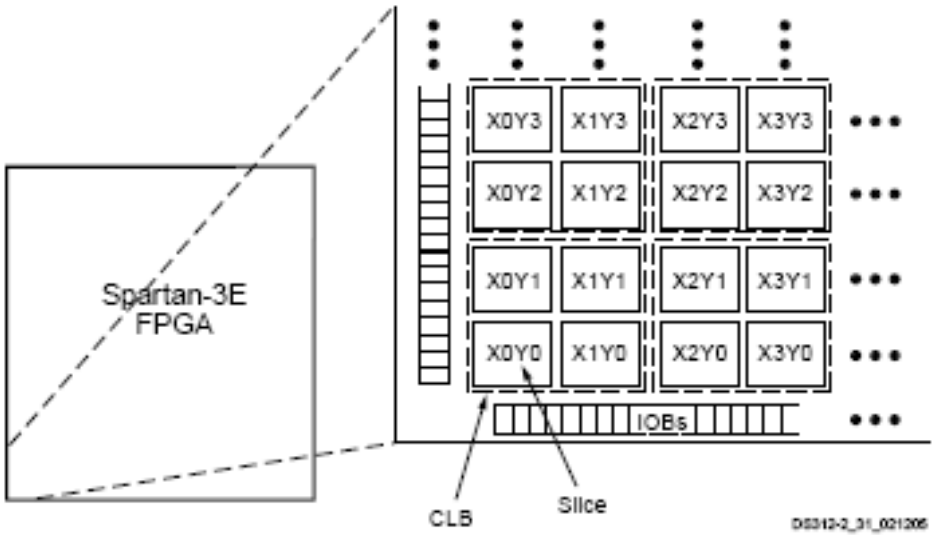


Figure 5-1: **CLB Locations**

Slice Location Designations

The Xilinx development software designates the location of a slice according to its X and Y coordinates, starting in the bottom left corner, as shown in Figure 5-1. The letter ‘X’ followed by a number identifies columns of slices, incrementing from the left side of the die to the right. The letter ‘Y’ followed by a number identifies the position of each slice in a pair as well as indicating the CLB row, incrementing from the bottom of the die.The SLICEM always has an even ‘X’ number, and the SLICEL always has an odd ‘X’ number.

Atributos… lo que estaban esperando!!!

- Ahora que usted ya conoce como nombrar cada uno de los SLICES de su FPGA… puede indicarle a su compilador favorito (XST) en que lugar EXACTAMENTE debe él colocar la memoria requerida por su aplicación. Se hace así:

  INST "U_RAM16" LOC = "SLICE_X0Y0“

- LOC es un atributo.
- Hay otros atributos, por ejemplo el atributo INIT. Por defecto los valores iniciales son cero, pero se pueden cambiar usando ese atributo

Números

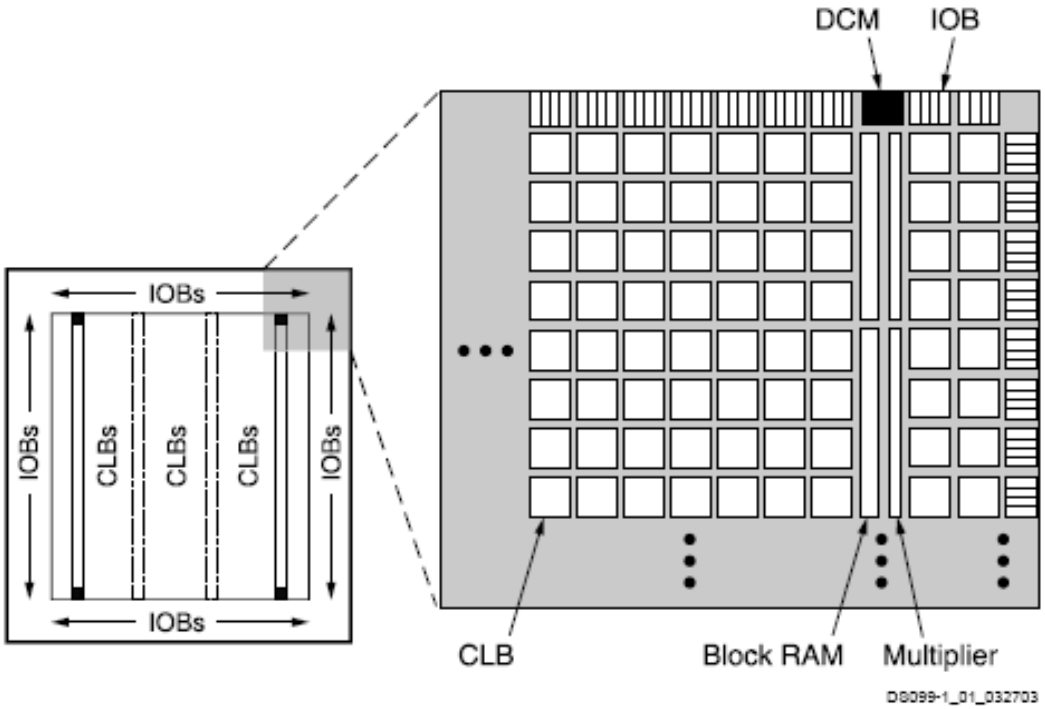
Device	CLB Rows	CLB Columns	CLB Total	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S200	24	20	480	1,920	3,840	4,320	1,920	30,720

Inferencia para LUT RAM

- **Edit ->Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Synthesis Constructs >Coding Examples ->RAM**

```
parameter RAM_WIDTH = <ram_width>;
parameter RAM_ADDR_BITS = <ram_addr_bits>;

reg [RAM_WIDTH-1:0] <ram_name> [(2**RAM_ADDR_BITS)-1:0];

wire [RAM_WIDTH-1:0] <output_data>;

<reg_or_wire> [RAM_ADDR_BITS-1:0] <address>;
<reg_or_wire> [RAM_WIDTH-1:0] <input_data>;

always @(posedge <clock>)
  if (<write_enable>)
    <ram_name>[<address>] <= <input_data>;

assign <output_data> = <ram_name>[<address>];
```

Block Ram



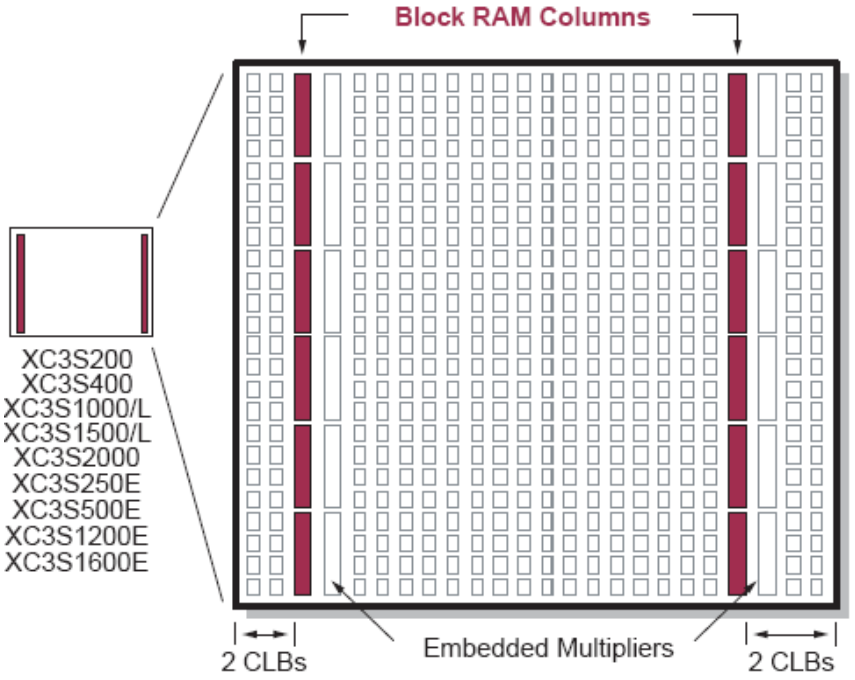
Notes:

1. The two additional block RAM columns of the XC3S4000 and XC3S5000 devices are shown with dashed lines. The XC3S50 has only the block RAM column on the far left.

Figure 1: **Spartan-3 Family Architecture**

Block Ram

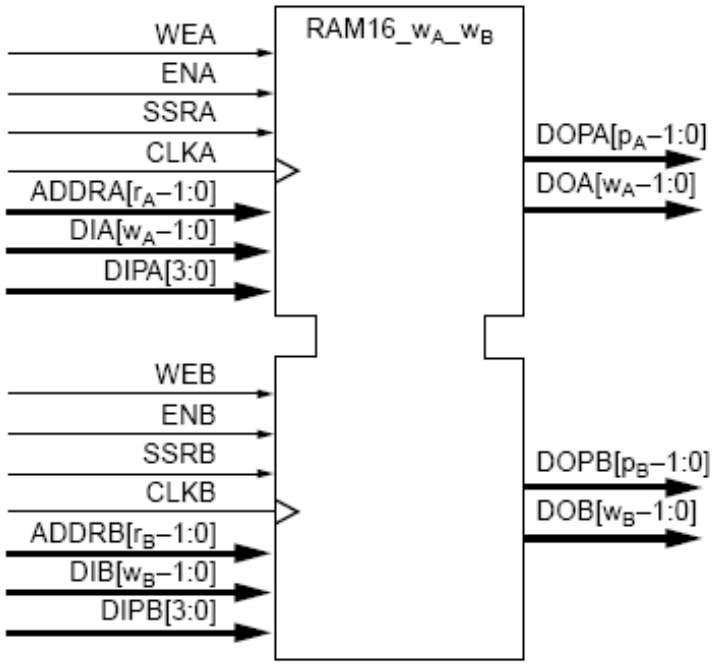
- Están organizados en columnas.



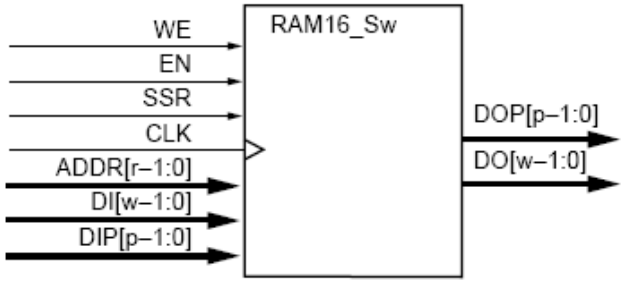
Device	RAM Columns	RAM Blocks Per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
XC3S200	2	6	12	221,184	216K

Block Ram

- Cada módulo de block RAM contiene 18432 bits de memoria estática. 16 Kbits para datos y 2Kbits para paridad.
- Físicamente, el BRAM tiene dos puertos completamente independientes: Port A y Port B. La estructura es simétrica y ambos puertos soportan lectura y escritura.Los puertos son síncronos, cada uno con su propias señales: clk, clk enable y write enable. Las operaciones de lectura son síncronas, y requieren una señal de clk y otra de clk enable.
- A pesar de que físicamente los BRAM son módulos de doble puerto pueden ser configurados para actuar (simular) como una memoria de puerto único.
- También pueden unirse en cascada para crear organizaciones de memoria mas grandes.



(a) Dual-Port



(b) Single-Port

Señales

Table 12: **Block RAM Port Signals**

Signal Description	Port A Signal Name	Port B Signal Name	Direction	Function
Address Bus	ADDRA	ADDRB	Input	The Address Bus selects a memory location for read or write operations. The width (w) of the port's associated data path determines the number of available address lines (r). Whenever a port is enabled (ENA or ENB = High), address transitions must meet the data sheet setup and hold times with respect to the port clock (CLKA or CLKB). This requirement must be met, even if the RAM read output is of no interest.
Data Input Bus	DIA	DIB	Input	Data at the DI input bus is written to the addressed memory location addressed on an enabled active CLK edge. It is possible to configure a port's total data path width (w) to be 1, 2, 4, 9, 18, or 36 bits. This selection applies to both the DI and DO paths of a given port. Each port is independent. For a port assigned a width (w), the number of addressable locations is 16,384/(w-p) where "p" is the number of parity bits. Each memory location has a width of "w" (including parity bits). See the DIP signal description for more information of parity.
Parity Data Input(s)	DIPA	DIPB	Input	Parity inputs represent additional bits included in the data input path to support error detection. The number of parity bits "p" included in the DI (same as for the DO bus) depends on a port's total data path width (w). See Table 13.

Señales

Signal Description	Port A Signal Name	Port B Signal Name	Direction	Function
Data Output Bus	DOA	DOB	Output	Basic data access occurs whenever WE is inactive. The DO outputs mirror the data stored in the addressed memory location. Data access with WE asserted is also possible if one of the following two attributes is chosen: WRITE_FIRST and READ_FIRST. WRITE_FIRST simultaneously presents the new input data on the DO output port and writes the data to the address RAM location. READ_FIRST presents the previously stored RAM data on the DO output port while writing new data to RAM. A third attribute, NO_CHANGE, latches the DO outputs upon the assertion of WE. It is possible to configure a port's total data path width (w) to be 1, 2, 4, 9, 18, or 36 bits. This selection applies to both the DI and DO paths. See the DI signal description.
Parity Data Output(s)	DOPA	DOPB	Output	Parity inputs represent additional bits included in the data input path to support error detection. The number of parity bits "p" included in the DI (same as for the DO bus) depends on a port's total data path width (w). See Table 13.

Señales

Write Enable	WEA	WEB	Input	When asserted together with EN, this input enables the writing of data to the RAM. In this case, the data access attributes WRITE_FIRST, READ_FIRST or NO_CHANGE determines if and how data is updated on the DO outputs. See the DO signal description. When WE is inactive with EN asserted, read operations are still possible. In this case, a transparent latch passes data from the addressed memory location to the DO outputs.
Clock Enable	ENA	ENB	Input	When asserted, this input enables the CLK signal to synchronize Block RAM functions as follows: the writing of data to the DI inputs (when WE is also asserted), the updating of data at the DO outputs as well as the setting/resetting of the DO output latches. When de-asserted, the above functions are disabled.
Set/Reset	SSRA	SSRB	Input	When asserted, this pin forces the DO output latch to the value that the SRVAL attribute is set to. A Set/Reset operation on one port has no effect on the other ports functioning, nor does it disturb the memory's data contents. It is synchronized to the CLK signal.
Clock	CLKA	CLKB	Input	This input accepts the clock signal to which read and write operations are synchronized. All associated port inputs are required to meet setup times with respect to the clock signal's active edge. The data output bus responds after a clock-to-out delay referenced to the clock signal's active edge.

Block RAM (BRAM)

- Distintas organizaciones de un bloque de memoria.(16K)

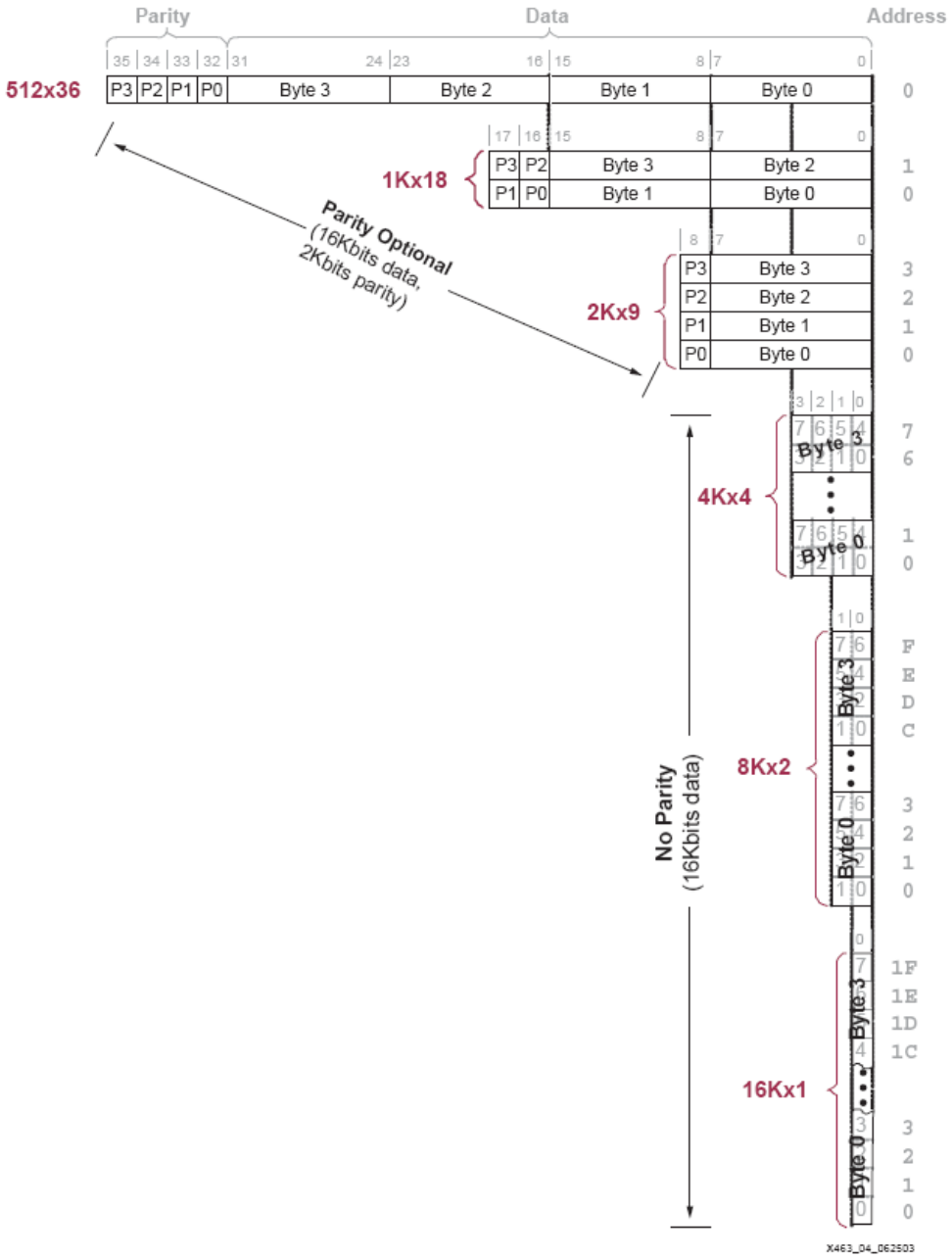


Figure 4: **Data Organization and Mapping Between Modes**

BRAM

- En xilinx ISE hay tres métodos para incorporar módulos de memoria al diseño:
  1. Instanciación HDL
  2. Utilizando el Core Generator
  3. Inferencia utilizando templates HDL

(1) Instanciación

- Se instancia un componente de memoria creado por xilinx. El template está en:
  - Edit->Languaje Templates ->Verilog-> Component Instantiation->Block Ram
- Para memorias de un puerto se utilizan los siguientes nombres de componentes:

Table 6: **Block RAM Data Organizations/Aspect Ratios**

Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	16K

- Para ancho de datos mayores o iguales a byte, hay paridad (un bit por byte). Por lo tanto, el tamaño total de RAM es 18Kbits.

(1) Instanciación

- Para memorias de doble puerto se utilizan los siguientes nombres de componentes:

Table 7: **Dual-Port RAM Component Suffix Appended to "RAMB16"**

		Port A					
		16Kx1	8Kx2	4Kx4	2Kx9	1Kx18	512x36
Port B	16Kx1	_s1_s1					
	8Kx2	_s1_s2	_s2_s2				
	4Kx4	_s1_s4	_s2_s4	_s4_s4			
	2Kx9	_s1_s9	_s2_s9	_s4_s9	_s9_s9		
	1Kx18	_s1_s18	_s2_s18	_s4_s18	_s9_s18	_s18_s18	
	512x36	_s1_s36	_s2_s36	_s4_s36	_s9_s36	_s18_s36	_s36_s36

(1) Instanciación. Atributo INIT

- Las memorias por defecto están inicializadas a cero, pero pueden ser inicializadas a otros valores utilizando el atributo : INIT_xx en la primitiva instanciada.
- INIT_xx : donde xx: 00 a 3F. Son 64 atributos de inicialización. Los números xx son dígitos hexadecimales.
- Cada INIT_xx corresponde a un vector de 256 bits.
- El atributo INITP_xx corresponde a la inicialización de los bits de paridad. En este caso, xx va de 00 a 07

Table 8: **VHDL/Verilog RAM Initialization Attributes for Block RAM**

Attribute	From	To
INIT_00	255	0
INIT_01	511	256
INIT_02	767	512
...	...	...
INIT_3F	16383	16128

(1) Instanciación: atributo INIT

The following formula defines the bit positions for each **INIT_xx** attribute.

Given yy = convert_hex_to_decimal(xx), **INIT_xx** corresponds to the following memory cells.

- Starting Location: [(yy + 1) * 256] –1
- End Location: (yy) * 256

For example, for the attribute INIT_1F, the conversion is as follows:

- yy = convert_hex_to_decimal(0x1F) = 31
- Starting Location: [(31+1) * 256] –1 = 8191
- End Location: 31 * 256 = 7936

(1) Instanciación: otros atributos

- También los valores de los latches de salida de ambos puertos pueden inicializarse. Por defecto están en cero. El atributo para inicializarlos es INITA , INITB (para las memorias de doble puerto), INIT para las memorias de único puerto.
- Los atributos SRVAL (Set Reset Value) permiten definir el valor de inicialización del latch de salida después de un reset. Por defecto, el valor es cero. Para las memorias de doble puerto los atributos son SRVAL_A y SRVAL_B

(1)Instanciación: atributo Write Mode

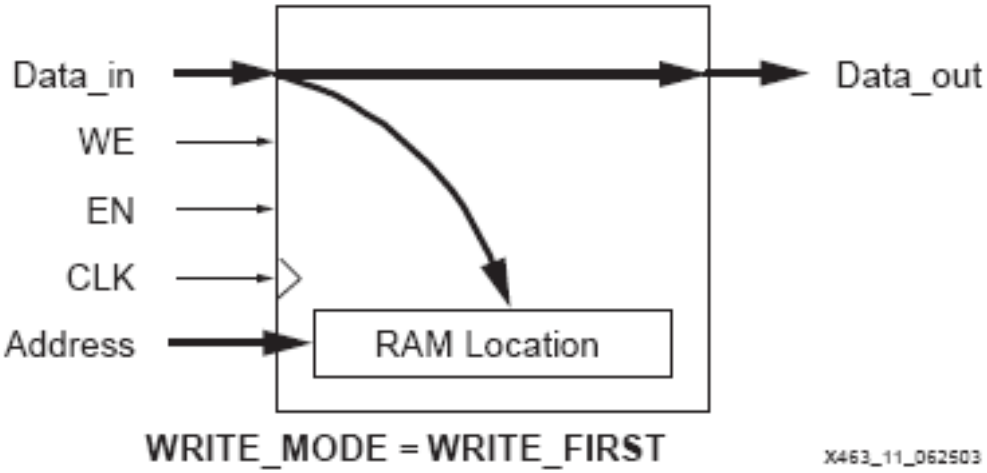
- Write Mode: Refiere a la política cuando se realiza una escritura simultánea con una lectura en una misma dirección (mismo puerto).
- Atributo write mode, y puede tener los siguientes valores

Table 9: **WRITE_MODE Affects Data Output Latches During Write Operations**

Write Mode	Effect on Same Port	Effect on Opposite Port (dual-port mode only, same address)
WRITE_FIRST Read After Write (Default)	Data on DI, DIP inputs written into specified RAM location and simultaneously appears on DO, DOP outputs.	Invalidates data on DO, DOP outputs.
READ_FIRST Read Before Write (Recommended)	Data from specified RAM location appears on DO, DOP outputs. Data on DI, DIP inputs written into specified location.	Data from specified RAM location appears on DO, DOP outputs.
NO_CHANGE No Read on Write	Data on DO, DOP outputs remains unchanged. Data on DI, DIP inputs written into specified location.	Invalidates data on DO, DOP outputs.

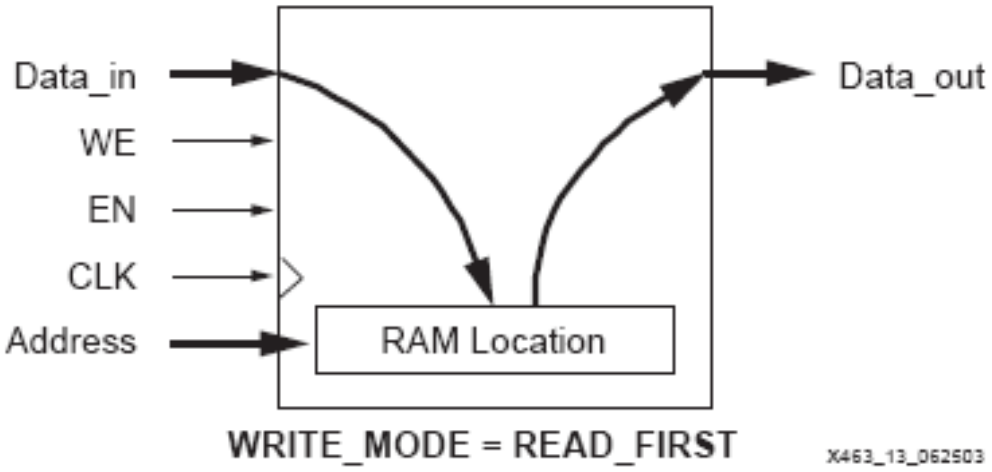
Atributo WRITE MODE: Write First

- El dato que se escribe puede ser leído simultáneamente. Es el modo por defecto.



Atributo WRITE MODE :READ FIRST

- El dato que aparece a la salida es el dato que estaba previamente almacenado. Modo recomendado.



Atributo WRITE MODE :NO CHANGE

- Los latches de salida son desabilitados, de esta manera se simula una memoria donde en un ciclo o bien se lee o bien se escribe.

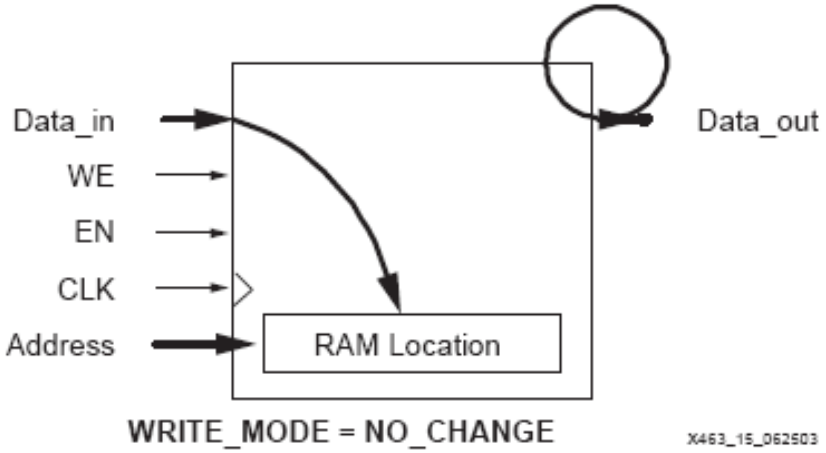


Figure 15: **Data Flow during a NO_CHANGE Write Operation**

ATRIBUTO LOC para BRAM

- Podemos especificar cual BRAM queremos utilizar mediante el atributo LOC.
  - LOC= RAMB16_X#Y#

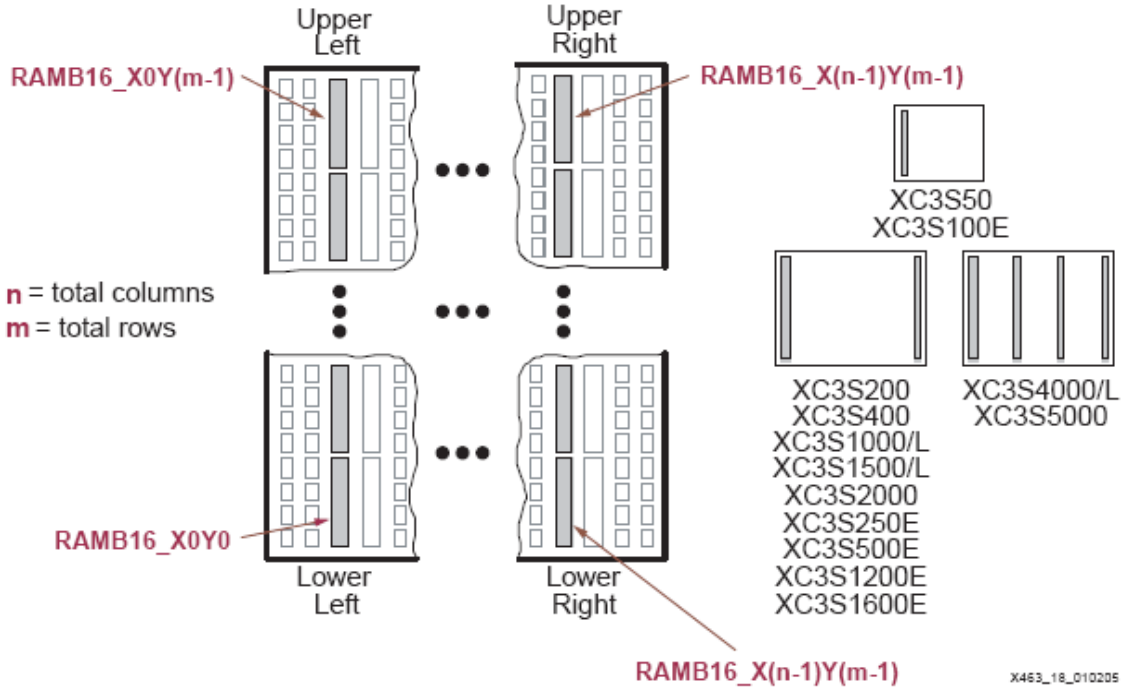


Figure 18: **Block RAM LOC Coordinates**

(1) Instanciación

```
   RAMB16_S1_S4 #(
   .INIT_A(1'b0),  // Value of output RAM registers on Port A at startup
   .INIT_B(4'h0),  // Value of output RAM registers on Port B at startup
   .SRVAL_A(1'b0), // Port A output value upon SSR assertion
   .SRVAL_B(4'h0), // Port B output value upon SSR assertion
   .WRITE_MODE_A("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
   .WRITE_MODE_B("WRITE_FIRST"), // WRITE_FIRST, READ_FIRST or NO_CHANGE
   .SIM_COLLISION_CHECK("ALL"),  // "NONE", "WARNING_ONLY", "GENERATE_X_ONLY", "ALL"

   // The following INIT_xx declarations specify the initial contents of the RAM
   // Port A Address 0 to 4095, Port B Address 0 to 1023
   .INIT_00(256'h0000000000000000000000000000000000000000000000000000000000000000),
    .....
   .INIT_0F(256'h0000000000000000000000000000000000000000000000000000000000000000),
   // Port A Address 4096 to 8191, Port B Address 1024 to 2047
   .INIT_10(256'h0000000000000000000000000000000000000000000000000000000000000000),
   .....
   .INIT_1F(256'h0000000000000000000000000000000000000000000000000000000000000000),
   // Port A Address 8192 to 12287, Port B Address 2048 to 3071
   .INIT_20(256'h0000000000000000000000000000000000000000000000000000000000000000),
    ....
   .INIT_2F(256'h0000000000000000000000000000000000000000000000000000000000000000),
   // Port A Address 12288 to 16383, Port B Address 3072 to 4095
   .INIT_30(256'h0000000000000000000000000000000000000000000000000000000000000000),
   ......
   .INIT_3F(256'h0000000000000000000000000000000000000000000000000000000000000000)
) RAMB16_S1_S4_inst (
   .DOA(DOA),      // Port A 1-bit Data Output
   .DOB(DOB),      // Port B 4-bit Data Output
   .ADDRA(ADDRA),  // Port A 14-bit Address Input
   .ADDRB(ADDRB),  // Port B 12-bit Address Input
   .CLKA(CLKA),    // Port A Clock
   .CLKB(CLKB),    // Port B Clock
   .DIA(DIA),      // Port A 1-bit Data Input
   .DIB(DIB),      // Port B 4-bit Data Input
   .ENA(ENA),      // Port A RAM Enable Input
   .ENB(ENB),      // Port B RAM Enable Input
   .SSRA(SSRA),    // Port A Synchronous Set/Reset Input
   .SSRB(SSRB),    // Port B Synchronous Set/Reset Input
   .WEA(WEA),      // Port A Write Enable Input
   .WEB(WEB)       // Port B Write Enable Input
);
```

Es una dual port asimétrica:
un port de un bit (DOA)
y el otro de 4 bits (DOB).
Por lo tanto, ADDRA es de 14 bits
Y ADDRB de 12 bits

(2) Programa Core Generator

- Para simplificar la instanciación del componente, xilinx provee un utilitario, el Core Generator.
- El programa genera varios archivos. El archivo con extensión . xco contiene la información necesaria para construir el componente. También genera un archivo con extensión .vhd que solo sirve para simulación. Este archivo se ignora durante el proceso de síntesis. No se puede instanciar un componente con ese archivo.

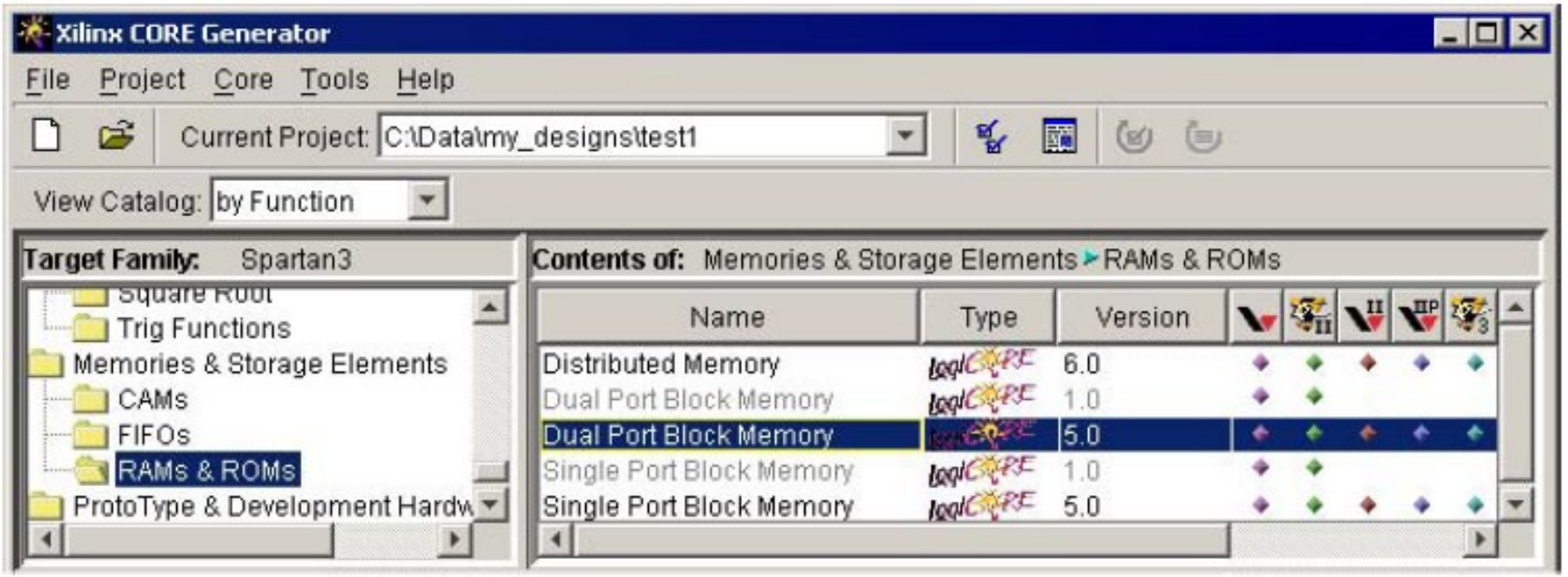


Figure 5: **Selecting a Block RAM Function in CORE Generator System**

(3) Inferencia

- ISE incluye templates para inferir block RAM. Para utilizarlos seleccionar:
  - Edit ->Language Templates ->VHDL->Synthesis Templates->RAM
- (Aclaración: ¿Que es inferencia?
  - Los diseños están usualmente compuestos de lógica combinatoria y de macros. Las macros son , por ejemplo, sumadores, comparadores, flip-flops, FSMs y RAMs.
  - Al ejecutarse, XST trata de reconocer tantas macros como sea posible. Todas estas macros son luego pasadas al paso de Optimización de Bajo Nivel. En este paso, o bien son preservadas como bloques independientes o bien mezclados con lógica para lograr una mejor utilización de los recursos. Se puede tener control absoluto sobre el proceso de inferencia mediante la especificación de “constraints”. (Veremos enseguida como…)
-)

(3) Inferencia: Single Port

- Single Port RAM:
  - Escritura es siempre síncrona=> en el flanco ascendente del reloj se muestrean : dirección, control y datos.
  - Lectura: puede ser síncrona o asíncrona
    - asíncrona: después que cambian las señales de dirección, el dato está disponible después de un pequeño delay en la salida
    - Síncrona: las señales de dirección se muestrean en el flanco ascendente del reloj y se guardan en un registro. La dirección registrada se utiliza para acceder la RAM. La existencia del registro de direcciones retrasa la lectura y la sincroniza con el reloj.
    - IMPORTANTE: debido a su estructura, las lecturas ASÍNCRONAS solo se implementan con RAM distribuida.

(3) Inferencia. Single Port. Lectura Asíncrona.

- Escritura Síncrona. Lectura Asíncrona.
  - Solo es realizable con Memoria Distribuida. Recordamos que:
    - Memoria distribuida es construida con las LUT´s de los CLB.
    - Pueden construirse:
      - 16x1 de doble puerto: uno de lectura y uno de escritura
      - 64x1, 32x1 y 16 x1: puerto único

(3) Inferencia. Single Port. Lectura Asíncrona.

```
// Single-port RAM with asynchronous read
// Modified from XST 8.1i v_rams_04

module xilinx_one_port_ram_async
  #(
    parameter ADDR_WIDTH = 12,
                  DATA_WIDTH = 8
  )
  (
   input wire clk,
   input wire we,
   input wire [ADDR_WIDTH-1:0] addr,
   input wire [DATA_WIDTH-1:0] din,
   output wire [DATA_WIDTH-1:0] dout
  );
```

```
  // signal declaration
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

  // body
  always @(posedge clk)
    if (we)  // write operation
      ram[addr] <= din;
  // read operation
  assign dout = ram[addr];

endmodule
```

(3) Inferencia: Single Port. Lectura Asíncrona.

- HDL síntesis

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

Macro Statistics
# RAMs                                  : 1
 4096x8-bit single-port RAM             : 1
=========================================
```

INFO:Xst - HDL ADVISOR - The RAM <Mram_ram> will be implemented on LUTs either because you have described an asynchronous read or because of currently unsupported block RAM features. If you have described an asynchronous read, making it synchronous would allow you to take advantage of available block RAM resources, for optimized device usage and improved timings. Please refer to your documentation for coding guidelines.

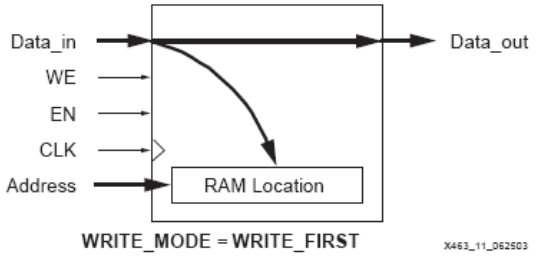
```
=========================================
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                  : 1
 4096x8-bit single-port distributed RAM   : 1

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

(3) Inferencia. Single Port. Lectura Síncrona

```
module xilinx_one_port_ram_sync
  #(
    parameter ADDR_WIDTH = 12,
              DATA_WIDTH = 8
  )
  (
   input wire clk,
   input wire we,
   input wire [ADDR_WIDTH-1:0] addr,
   input wire [DATA_WIDTH-1:0] din,
   output reg [DATA_WIDTH-1:0] dout
  );

  // signal declaration
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
  reg [ADDR_WIDTH-1:0] addr_reg;

  // body
  always @(posedge clk)
  begin
     if (we)  // write operation
      begin
       ram[addr] <= din;
       dout<= din;
       end
     else
      dout <= ram[addr_reg];
 end
endmodule
```

- Según el código que escribamos XST infiere BRAM con las 3 políticas de escritura: WRITE-FIRST, READ-FIRST, NO-CHANGE

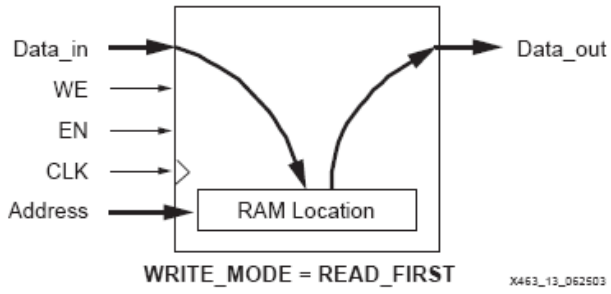
- Infiere BRAM con política WRITE-FIRST



(3) Inferencia. Single Port. Lectura Síncrona

```
module xilinx_one_port_ram_sync
  #(
    parameter ADDR_WIDTH = 12,
                  DATA_WIDTH = 8
  )
  (
   input wire clk,
   input wire we,
   input wire [ADDR_WIDTH-1:0] addr,
   input wire [DATA_WIDTH-1:0] din,
   output reg [DATA_WIDTH-1:0] dout
  );

  // signal declaration
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
  reg [ADDR_WIDTH-1:0] addr_reg;

  // body
  always @(posedge clk)
  begin
    if (we)  // write operation
      ram[addr] <= din;
    dout<= ram[addr] ;
   end
endmodule
```

- Infiere BRAM con política READ-FIRST



(3) Inferencia. Single Port. Lectura Síncrona

```
// signal declaration
reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

// body
always @(posedge clk)
   if (we)  // write operation
      ram[addr] <= din;
   else
// read operation
   dout <= ram[addr];


endmodule
```

- Infiere BRAM con política NO-CHANGE

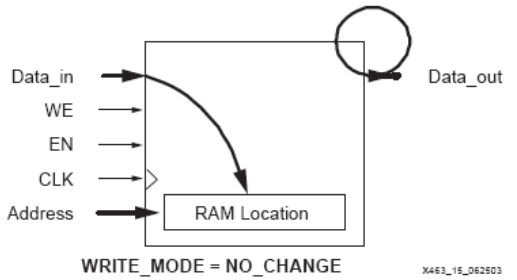


Figure 15: **Data Flow during a NO_CHANGE Write Operation**

Constraints (Restricciones)

- Las restricciones controlan diversos aspectos de la síntesis, si no se utilizan se sintetiza con las opciones por defecto.
- Se especifican mediante:
  - Opciones: proveen control global. Se setean en la opción Process Properties del Project Navigator .
  - Atributos de la señal en HDL
  - En un archivo separado: archivo XCF

Constraints para Memoria

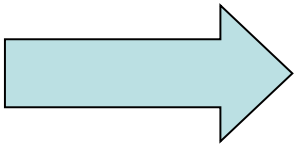
Constraint Name	Constraint Value	XCF Constraint Syntax Target	VHDL Target	Verilog Target	Cmd Line	Cmd Value
ram_extract	yes, no, **true**, false	model, net (in model)	entity, signal	module, signal	–ram_extract	yes, no
ram_style	**auto**, block, distributed, pipe_distributed	model, net (in model)	entity, signal	module, signal	–ram_style	**auto**, block, distributed
rom_extract	yes, no, **true**, false	model, net (in model)	entity, signal	module, signal	–rom_extract	yes, no
rom_style	**auto**, block, distributed	model, net (in model)	entity, signal	module, signal	–rom_style	**auto**, block, distributed

(* ram_style=‘block’*) reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

Inferencia de ROMs

- No hay ROM en los circuitos de Spartan, pero pueden ser emuladas:
  - por lógica combinatoria (la salida solo depende de la entrada)
  - o por memoria distribuida
  - o por BRAM´s con lectura síncrona y escritura deshabilitada.
- Por lógica combinatoria: se pueden inferir de un *if* o de un *case.*

Con lectura asíncrona

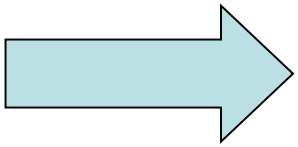


```
module rom_template
   (
    input wire [3:0] addr,
    output reg [7:0] data
   );

   // body
   always @*
      case (addr)
         4'h0: data = 7'b0000001;
         4'h1: data = 7'b1001111;
         4'h2: data = 7'b0010010;
         4'h3: data = 7'b0000110;
         4'h4: data = 7'b1001100;
         4'h5: data = 7'b0100100;
         4'h6: data = 7'b0100000;
         4'h7: data = 7'b0001111;
         4'h8: data = 7'b0000000;
         4'h9: data = 7'b0000100;
         4'ha: data = 7'b0001000;
         4'hb: data = 7'b1100000;
         4'hc: data = 7'b0110001;
         4'hd: data = 7'b1000010;
         4'he: data = 7'b0110000;
         4'hf: data = 7'b0111000;
      endcase

endmodule
```

Inferencia de ROMs

- SE registra la dirección

Con lectura síncrona



```
module xilinx_rom_sync_template
   (
    input wire clk,
    input wire [3:0] addr,
    output reg [7:0] data
   );

   // signal declaration
   reg [3:0] addr_reg;

   // body
   always @(posedge clk)
      addr_reg <= addr;

   always @*
      case (addr_reg)
         4'h0: data = 7'b0000001;
         4'h1: data = 7'b1001111;
         4'h2: data = 7'b0010010;
         4'h3: data = 7'b0000110;
         4'h4: data = 7'b1001100;
         4'h5: data = 7'b0100100;
         4'h6: data = 7'b0100000;
         4'h7: data = 7'b0001111;
         4'h8: data = 7'b0000000;
         4'h9: data = 7'b0000100;
         4'ha: data = 7'b0001000;
         4'hb: data = 7'b1100000;
         4'hc: data = 7'b0110001;
         4'hd: data = 7'b1000010;
         4'he: data = 7'b0110000;
         4'hf: data = 7'b0111000;
      endcase

endmodule
```

Inferencia de memorias de doble puerto

```
// Dual-port RAM with asynchronous read
// Modified from XST 8.1i v_rams_09

module xilinx_dual_port_ram_async
  #(
    parameter ADDR_WIDTH = 6,
              DATA_WIDTH = 8
  )
  (
   input wire clk,
   input wire we,
   input wire [ADDR_WIDTH-1:0] addr_a, addr_b,
   input wire [DATA_WIDTH-1:0] din_a,
   output wire [DATA_WIDTH-1:0] dout_a, dout_b
  );
```

```
// signal declaration
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];

  // body
  always @(posedge clk)
    if (we)  // write operation
      ram[addr_a] <= din_a;
  // two read operations
  assign dout_a = ram[addr_a];
  assign dout_b = ram[addr_b];

endmodule
```

- Read Asíncrono

Inferencia de memorias de doble puerto

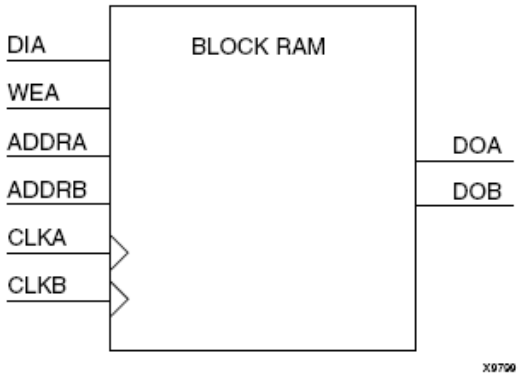
```
// Dual-port RAM with asynchronous read
// Modified from XST 8.1i v_rams_09

module xilinx_dual_port_ram_async
  #(
    parameter ADDR_WIDTH = 6,
              DATA_WIDTH = 8
  )
  (
   input wire clk,
   input wire we,
   input wire [ADDR_WIDTH-1:0] addr_a, addr_b,
   input wire [DATA_WIDTH-1:0] din_a,
   output wire [DATA_WIDTH-1:0] dout_a, dout_b
  );
```

```
// signal declaration
  reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
   reg [ADDR_WIDTH-1:0] addr_a_reg, addr_b_reg;


  // body
   always @(posedge clk)
  begin
    if (we)  // write operation
      ram[addr_a] <= din_a;
    addr_a_reg <= addr_a;
    addr_b_reg <= addr_b;
  end
  // two read operations
  assign dout_a = ram[addr_a_reg];
  assign dout_b = ram[addr_b_reg];
endmodule
```

- Read Síncrono

Inferencia para memorias de doble puerto con distintos relojes

IO Pins	Description
clka	Positive-Edge Clock
clkb	Positive-Edge Clock
wea	Primary Synchronous Write Enable (Active High)
addra	Write Address/Primary Read Address
addrb	Dual Read Address
dia	Primary Data Input
doa	Primary Output Port
dob	Dual Output Port



Doble puerto, distintos relojes

```
parameter RAM_WIDTH = <ram_width>;
  parameter RAM_ADDR_BITS = <ram_addr_bits>;

  reg [RAM_WIDTH-1:0] <ram_name> [(2**RAM_ADDR_BITS)-1:0];
  reg [RAM_WIDTH-1:0] <output_dataA>, <output_dataB>;

  <reg_or_wire> [RAM_ADDR_BITS-1:0] <addressA>, <addressB>;
  <reg_or_wire> [RAM_WIDTH-1:0] <input_dataA>;


  always @(posedge <clockA>)
    if (<enableA>) begin
      if (<write_enableA>)
        <ram_name>[<addressA>] <= <input_dataA>;
      <output_dataA> <= <ram_name>[<addressA>];
    end

  always @(posedge <clockB>)
    if (<enableB>)
      <output_dataB> <= <ram_name>[<addressB>];
```

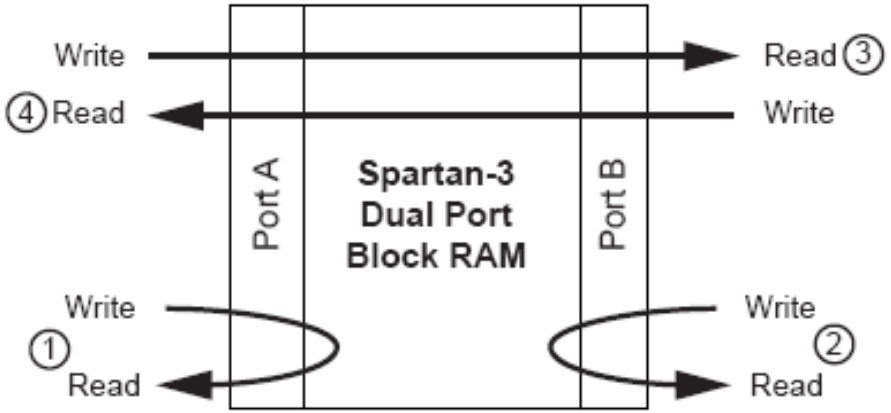
Doble puerto, dos puertos de escritura

```
parameter RAM_WIDTH = <ram_width>;
  parameter RAM_ADDR_BITS = <ram_addr_bits>;

  reg [RAM_WIDTH-1:0] <ram_name> [(2**RAM_ADDR_BITS)-1:0];
  reg [RAM_WIDTH-1:0] <output_dataA>, <output_dataB>;

  <reg_or_wire> [RAM_ADDR_BITS-1:0] <addressA>, <addressB>;
  <reg_or_wire> [RAM_WIDTH-1:0] <input_dataA>;

  //  The following code is only necessary if you wish to initialize the RAM
  //  contents via an external file (use $readmemb for binary data)
  initial
    $readmemh("<data_file_name>", <rom_name>, <begin_address>, <end_address>);

  always @(posedge <clockA>)
    if (<enableA>) begin
      if (<write_enableA>)
        <ram_name>[<addressA>] <= <input_dataA>;
      <output_dataA> <= <ram_name>[<addressA>];
    end

  always @(posedge <clockB>)
    if (<enableB>) begin
      if (<write_enableB>)
        <ram_name>[<addressB>] <= <input_dataB>;
      <output_dataB> <= <ram_name>[<addressB>];
    end
```

Data Flow

Spartan-3 block RAM is constructed of true dual-port memory and simultaneously supports all the data flows and operations shown in Figure 3. Both ports access the same set of memory bits but with two potentially different address schemes depending on the port's data width.

1. Port A behaves as an independent single-port RAM supporting simultaneous read and write operations using a single set of address lines.
2. Port B behaves as an independent single-port RAM supporting simultaneous read and write operations using a single set of address lines.
3. Port A is the write port with a separate write address and Port B is the read port with a separate read address. The data widths for Port A and Port B can be different also.
4. Port B is the write port with a separate write address and Port A is the read port with a separate read address. The data widths for Port B and Port A can be different also.

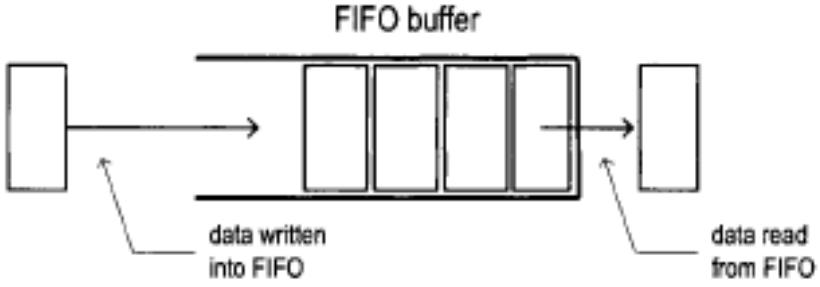


Trabajo a realizar

- Hacer la implementación de la FIFO, usando:
  - Registros
  - Memoria Distribuida
  - BRAM

FIFO

- Una cola FIFO es una estructura de datos con dos señales: wr y rd.
- La señal wr, cuando está activa, permite que un dato sea escrito en la FIFO.
- La cabeza de la cola está siempre visible, o sea que siempre podemos leer. La señal rd, cuando está activa, indica que el elemento de la cabeza se puede eliminar.



FIFO: Implementación con una cola circular

- Dos señales de control:
  - wr: si está activada, se escribe el dato
  - rd: si está activada, el primer elemento de la cola se remueve.
  - El primer elemento de la cola puede leerse siempre.
- Implementación:
  - Los registros forman una cola circular, con dos punteros:
    - wr ptr: apunta a la cabeza de la cola (head)
    - rd ptr : apunta a la cola de la cola (tail)
    - Los punteros avanzan una posición por cada escritura o lectura
    - Hay dos señales de estado: full, empty

FIFO

- Las condiciones de lleno y vacío ocurren cuando los dos punteros son iguales.
- La implementacion consiste de:
  - El banco de registros
  - Controlador de la FIFO: formado por los dos punteros y los FF para lleno y vacio.

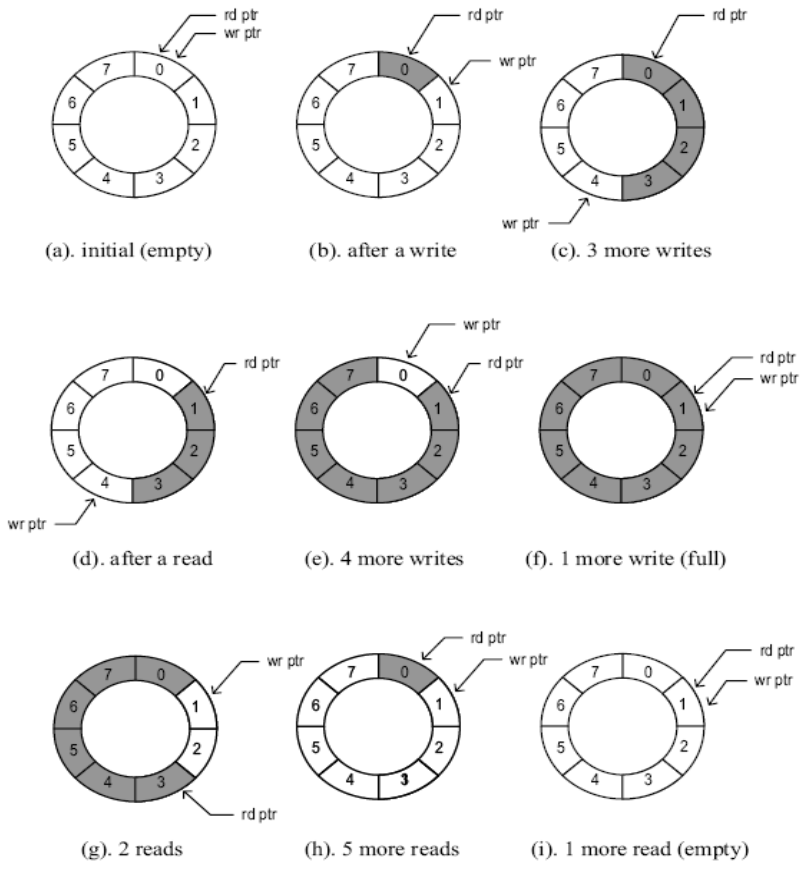


Figure 4.11 FIFO buffer based on a circular queue.

FIFO: Implementación

```
module fifo
  #(
   parameter B=8, // number of bits in a word
            W=4  // number of address bits
  )
  (
   input wire clk, reset,
   input wire rd, wr,
   input wire [B-1:0] w_data,
   output wire empty, full,
   output wire [B-1:0] r_data
  );

  //signal declaration
  reg [B-1:0] array_reg [2**W-1:0];  // register array
  reg [W-1:0] w_ptr_reg, w_ptr_next, w_ptr_succ;
  reg [W-1:0] r_ptr_reg, r_ptr_next, r_ptr_succ;
  reg full_reg, empty_reg, full_next, empty_next;
  wire wr_en;
```

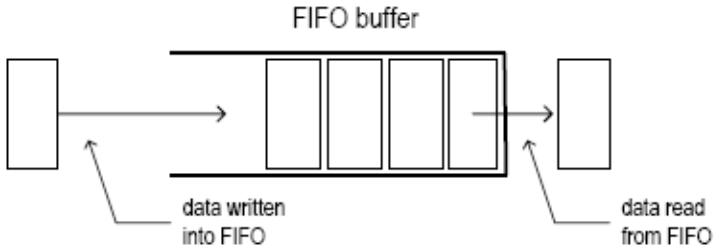
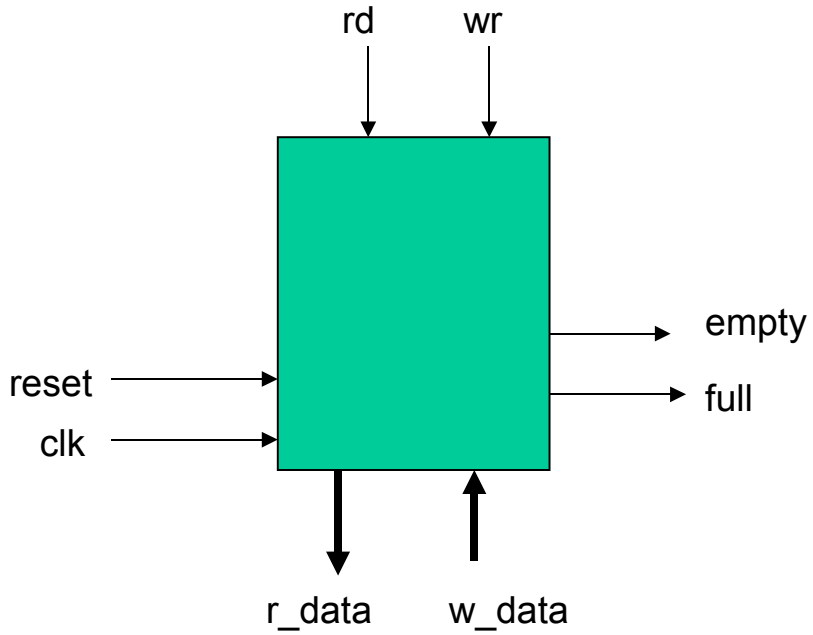
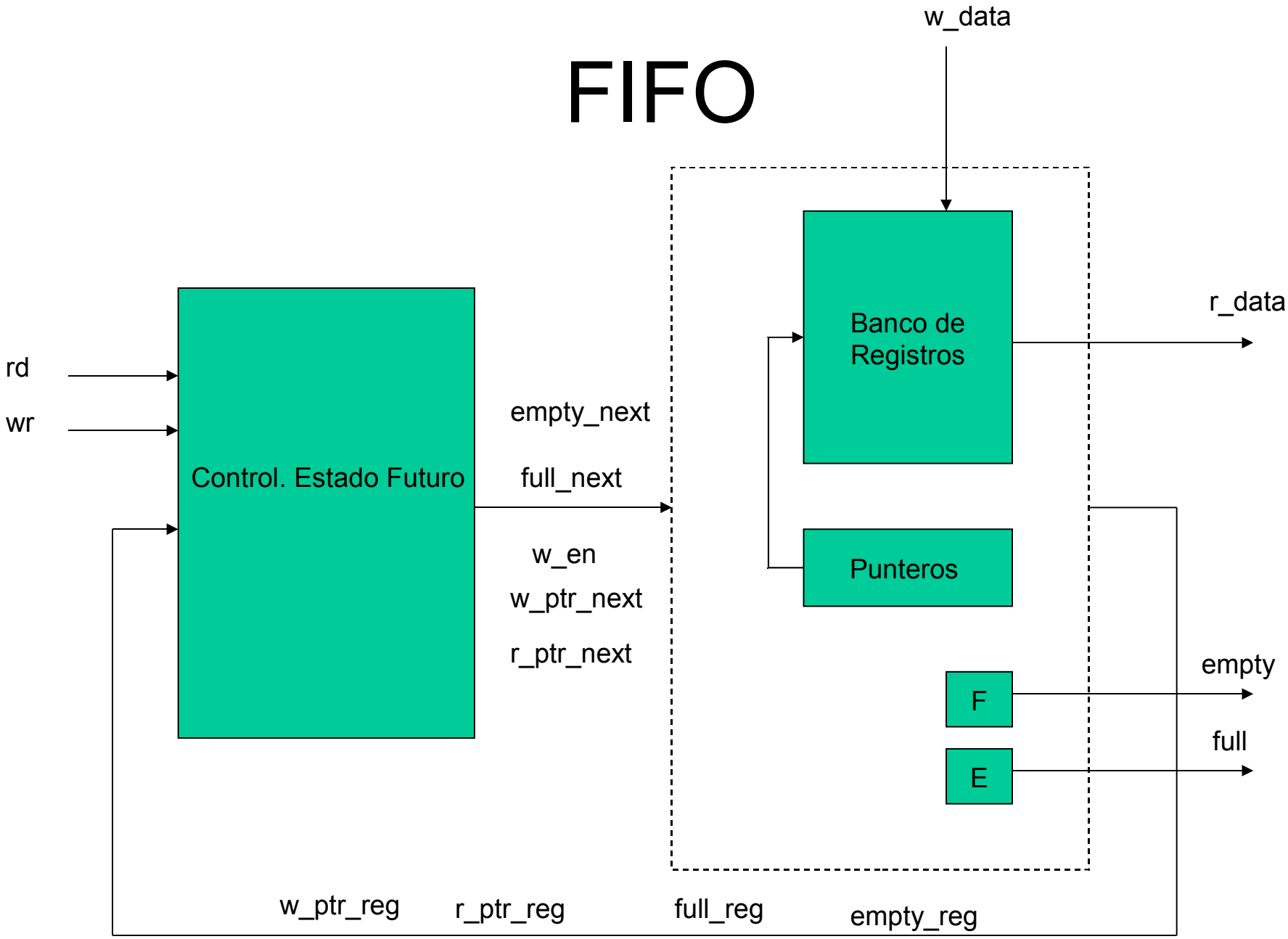


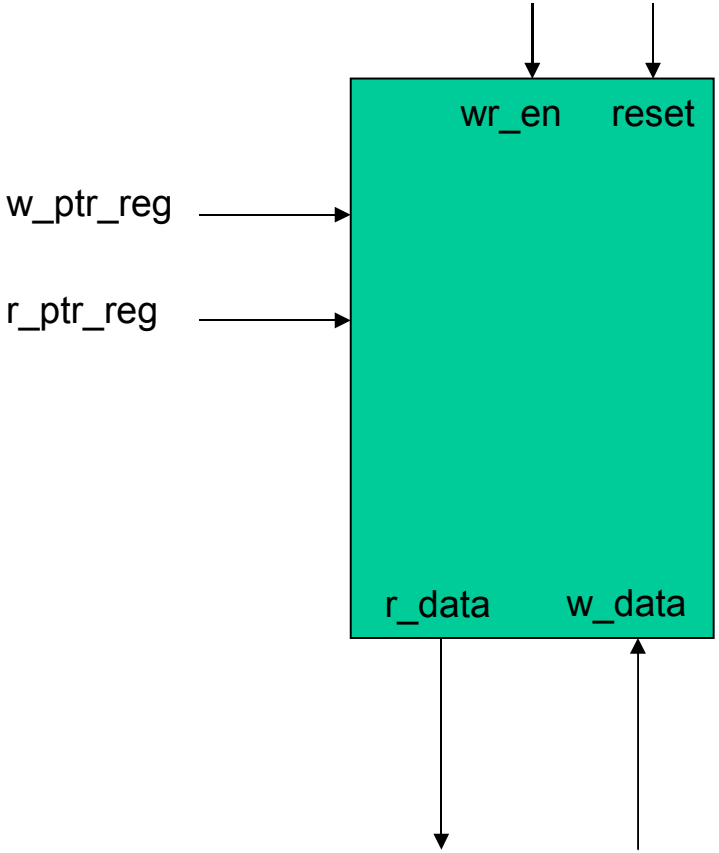
Figure 4.10 Conceptual diagram of a FIFO buffer.



FIFO


w_data
Banco de Registros
r_data
rd
wr
Control. Estado Futuro
empty_next
full_next
w_en
w_ptr_next
r_ptr_next
Punteros
F
E
empty
full
w_ptr_reg
r_ptr_reg
full_reg
empty_reg

FIFO: Banco de Registros

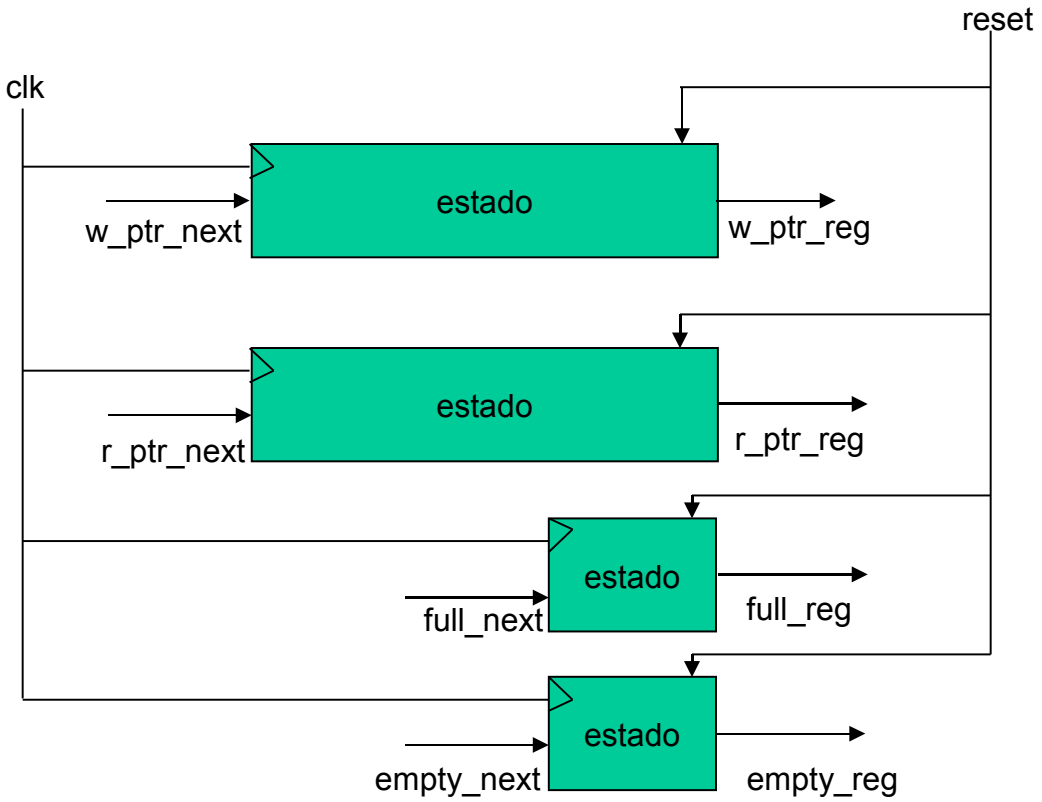



```
// body
// register file write operation
always @(posedge clk)
   if (wr_en)
      array_reg[w_ptr_reg] <= w_data;

// register file read operation
assign r_data = array_reg[r_ptr_reg];

// write enabled only when FIFO is not full
assign wr_en = wr & ~full_reg;
```

Escritura Síncrona, Lectura Asíncrona

FIFO Control: Punteros y Señales de Estado

```
 // fifo control logic
// register for read and write pointers
always @(posedge clk, posedge reset)
  if (reset)
    begin
      w_ptr_reg <= 0;
      r_ptr_reg <= 0;
      full_reg <= 1'b0;
      empty_reg <= 1'b1;
    end
  else
    begin
      w_ptr_reg <= w_ptr_next;
      r_ptr_reg <= r_ptr_next;
      full_reg <= full_next;
      empty_reg <= empty_next;
    end
```



FIFO Control: Lógica para el Estado Futuro

```
    // next-state logic for read and write
pointers
   always @*
   begin
      // successive pointer values
      w_ptr_succ = w_ptr_reg + 1;
      r_ptr_succ = r_ptr_reg + 1;
      // default: keep old values
      w_ptr_next = w_ptr_reg;
      r_ptr_next = r_ptr_reg;
      full_next = full_reg;
      empty_next = empty_reg;
```

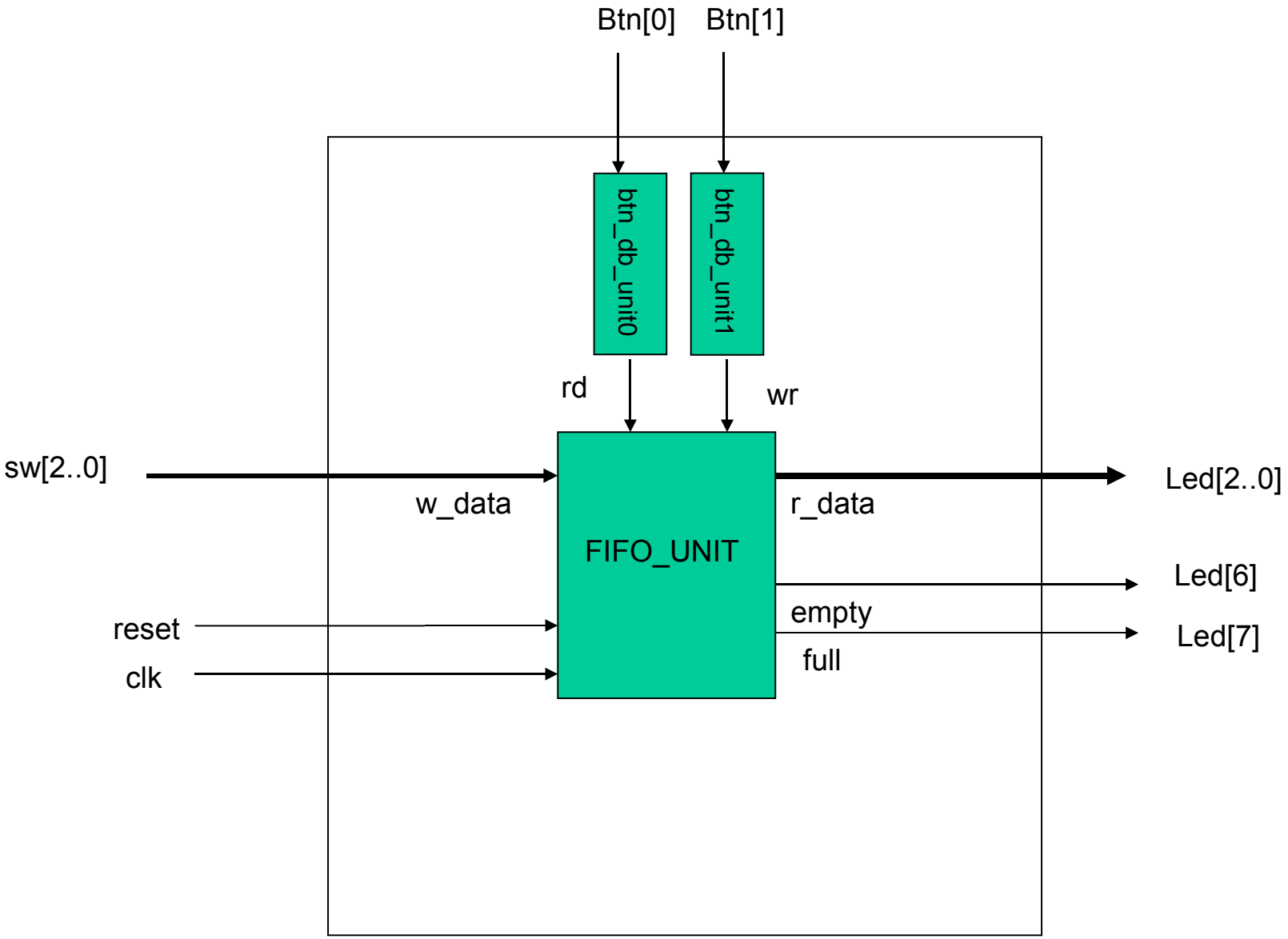
Always block combinacional

```
case ({wr, rd})
        // 2'b00:  no op
        2'b01: // read
          if (~empty_reg) // not empty
            begin
              r_ptr_next = r_ptr_succ;
              full_next = 1'b0;
              if (r_ptr_succ==w_ptr_reg)
                empty_next = 1'b1;
            end
        2'b10: // write
          if (~full_reg) // not full
            begin
              w_ptr_next = w_ptr_succ;
              empty_next = 1'b0;
              if (w_ptr_succ==r_ptr_reg)
                full_next = 1'b1;
            end
        2'b11: // write and read
          begin
            w_ptr_next = w_ptr_succ;
            r_ptr_next = r_ptr_succ;
          end
      endcase
   end
```

FIFO: Lógica para la Salida

```
 // output
   assign full = full_reg;
   assign empty = empty_reg;

endmodule
```

FIFO Test


Btn[0]
Btn[1]
btn_db_unit0
btn_db_unit1
rd
wr
sw[2..0]
w_data
FIFO_UNIT
r_data
Led[2..0]
Led[6]
empty
Led[7]
reset
clk
full

Código Test FIFO

```
// Listing 4.21
module fifo_test
   (
    input wire clk, reset,
    input wire [1:0] btn,
    input wire [2:0] sw,
    output wire [7:0] led
   );

   // signal declaration
   wire [1:0] db_btn;

   // debounce circuit for btn[0]
   debounce btn_db_unit0
      (.clk(clk), .reset(reset), .sw(btn[0]),
       .db_level(), .db_tick(db_btn[0]));
   // debounce circuit for btn[1]
   debounce btn_db_unit1
      (.clk(clk), .reset(reset), .sw(btn[1]),
       .db_level(), .db_tick(db_btn[1]));
   // instantiate a 2^2-by-3 fifo
   fifo #(.B(3), .W(2)) fifo_unit
      (.clk(clk), .reset(reset),
      .rd(db_btn[0]), .wr(db_btn[1]), .w_data(sw),
      .r_data(led[2:0]), .full(led[7]), .empty(led[6]));
   // disable unused leds
   assign led[5:3] = 3'b000;

 endmodule
```