

Using Block RAM

Summary

For applications requiring large, on-chip memories, Spartan®-3 generation FPGAs provide plentiful, efficient SelectRAM memory blocks. Using various configuration options, SelectRAM blocks create RAM, ROM, FIFOs, large look-up tables, data width converters, circular buffers, and shift registers, each supporting various data widths and depths. This chapter describes the features and capabilities of block SelectRAM and illustrates how to specify the various options using the Xilinx CORE Generator™ system or via VHDL or Verilog instantiation. Various non-obvious block RAM applications are discussed with references to additional tools, application notes, and documentation.

Introduction

All Spartan-3 generation FPGAs feature multiple block RAMs, organized in columns. The total amount of block RAM depends on the size of the Spartan-3 generation FPGA as shown in [Table 4-1](#).

Table 4-1: Block RAM Available in Spartan-3 Generation FPGAs

Family	Device	RAM Columns	RAM Blocks Per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
Extended Spartan-3A FPGAs	XC3SD1800A	4	20-22	84	1,548,288	1,512K
	XC3SD3400A	5	24-26	126	2,322,432	2,268K
	XC3S50A / AN	1	3	3	55,296	54K
	XC3S200A / AN	2	8	16	294,912	288K
	XC3S400A / AN	2	10	20	368,640	360K
	XC3S700A / AN	2	10	20	368,640	360K
	XC3S1400A / AN	2	16	32	589,824	576K
Spartan-3E FPGAs	XC3S100E	1	4	4	73,728	72K
	XC3S250E	2	6	12	221,184	216K
	XC3S500E	2	10	20	368,640	360K
	XC3S1200E	2	14	28	516,096	504K
	XC3S1600E	2	18	36	663,552	648K

Table 4-1: Block RAM Available in Spartan-3 Generation FPGAs (Cont'd)

Family	Device	RAM Columns	RAM Blocks Per Column	Total RAM Blocks	Total RAM Bits	Total RAM Kbits
Spartan-3 FPGAs	XC3S50	1	4	4	73,728	72K
	XC3S200	2	6	12	221,184	216K
	XC3S400	2	8	16	294,912	288K
	XC3S1000	2	12	24	442,368	432K
	XC3S1500	2	16	32	589,824	576K
	XC3S2000	2	20	40	737,280	720K
	XC3S4000	4	24	96	1,769,472	1,728K
	XC3S5000	4	26	104	1,916,928	1,872K

Notes:

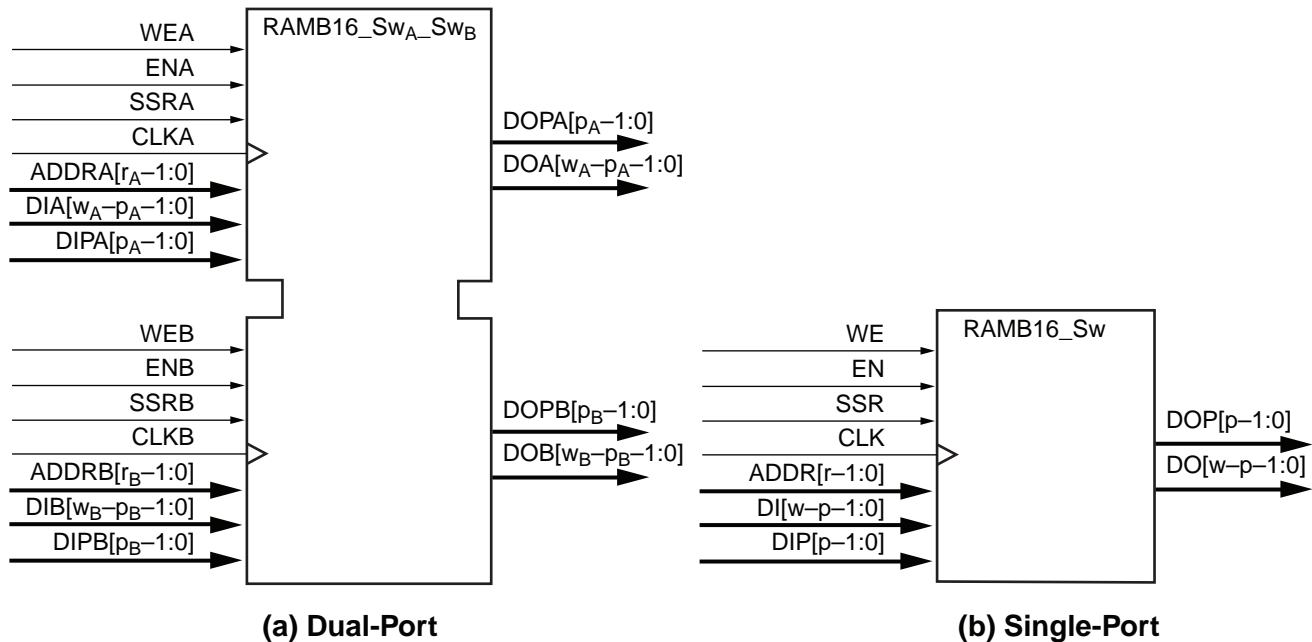
1. 1Kbit = 1,024 bits, per memory conventions.

Each block RAM contains 18,432 bits of fast static RAM, 16K bits of which is allocated to data storage and, in some memory configurations, an additional 2K bits allocated to parity or additional "plus" data bits. Physically, the block RAM has two completely independent access ports, labeled Port A and Port B. The structure is fully symmetrical, and both ports are interchangeable and support data read and write operations. Each memory port is synchronous with its own clock, clock enable, and write enable. Read operations are also synchronous and require a clock edge and clock enable.

Though physically a dual-port memory, block RAM simulates single-port memory in an application, as shown in [Figure 4-1](#). Furthermore, each block memory supports multiple configurations or aspect ratios. [Table 4-2](#) summarizes the essential SelectRAM features.

Cascade multiple block RAMs to create deeper and wider memory organizations with a minimal timing penalty incurred through specialized routing resources.

The block RAMs in the Spartan-3A DSP platform include an optional output register similar to the block RAM output register of the Virtex®-4 FPGA. The output register enables full-speed operation at over 250 MHz for all data widths.



X463_01_112009

Notes:

1. *w_A* and *w_B* are integers representing the total data path width (i.e., data bits plus parity bits) at ports A and B, respectively. See Table 4-8 and Table 4-9.
2. *p_A* and *p_B* are integers that indicate the number of data path lines serving as parity bits.
3. *r_A* and *r_B* are integers representing the address bus width at ports A and B, respectively.
4. The control signals CLK, WE, EN, and SSR on both ports have the option of inverted polarity.

Figure 4-1: SelectRAM 18K Blocks Perform as Dual-Port (a) and Single-Port (b) Memory

Table 4-2: SelectRAM 18K Block Memory Features and Applications

Total RAM bits, including parity	18,432 (16K data + 2K parity)
Memory Organizations	16Kx1 8Kx2 4Kx4 2Kx8 (no parity) 2Kx9 (x8 + parity) 1Kx16 (no parity) 1Kx18 (x16 + 2 parity) 512x32 (no parity) 512x36 (x32 + 4 parity) 256x72 (single-port only)
Parity	Available and optional only for organizations byte-wide or greater. Parity bits optionally available as extra data bits.
Performance	240+ MHz (refer to individual FPGA family data sheet)
Timing Interface	Simple synchronous interface. Similar to reading and writing from a register with a setup time for write operations and clock-to-output delay for read operations.

Table 4-2: SelectRAM 18K Block Memory Features and Applications (Cont'd)

Single-Port	Yes
True Dual-Port	Yes
ROM, Initial RAM Contents	Yes
Mixed Data Port Widths	Yes
Power-Up Condition	User-defined data, defaults to zero
Potential Applications	Local data storage, FIFOs, elastic stores, register files, buffers, stacks, circular buffers, shift registers, delay lines, waveform storage and generation, direct digital synthesis, CAMs, associative memories, function tables, function generators, wide logic functions, code converters, encoders, decoders, counters, state machines, microsequencers, program storage for embedded processor(s)

Block RAM Differences between Spartan-3 Generation Families

Overall, block RAM is similar in all Spartan-3 generation FPGAs. However, Extended Spartan-3A family FPGAs have some subtle but significant block RAM enhancements over Spartan-3E and Spartan-3 family FPGAs, as summarized in [Table 4-3](#). Extended Spartan-3A family FPGAs have byte-level write enable controls, supported by the RAMB16BWE design primitive. However, Extended Spartan-3A family FPGA designs continue to support the RAMB16 design primitive that is used for Spartan-3 or Spartan-3E FPGA designs (see [Table 4-8](#) and [Table 4-9](#)). Timing parameters are similar in functionality between the Spartan-3, Spartan-3E and Extended Spartan-3A family, but have different names. Spartan-3A DSP FPGAs add an output register, supported by the RAMB16BWER primitive.

Table 4-3: Comparison Between Spartan-3/3E, Spartan-3A/3AN, and Spartan-3A DSP FPGA Block RAMs

Feature	Spartan-3/3E FPGA Block RAM	Spartan-3A/AN FPGA Block RAM	Spartan-3A DSP FPGA Block RAM
Individual write-enables for each byte lane in x9, x18, or x36 configurations	No (single write-enable only)	Yes	Yes
Special routing resources between block RAM and multiplier for x36 configurations	No	Yes	General Purpose
Output register	No	No	Yes
Supported by RAMB16 primitive	Yes	Yes	Yes
Supported by RAMB16BWE primitive (RAMB16 with byte-level write enable)	No	Yes	Yes
Supported by RAMB16BWER primitive (RAMB16BWE with output register)	No	No	Yes

The Xilinx CORE Generator system supports various modules containing block RAM for Spartan-3 devices including:

- Embedded dual- or single-port RAM modules
- ROM modules
- Synchronous and asynchronous FIFO modules
- Content-Addressable Memory (CAM) modules

Furthermore, block RAM can be instantiated in any synthesis-based design using the appropriate *RAMB16* module from the Xilinx design library (see [Table 4-8](#) and [Table 4-9](#)).

This chapter describes the signals and attributes of the Spartan-3 FPGA block RAM feature, including details on the various attributes and applications for block RAM.

Block RAM Location and Surrounding Neighborhood

As mentioned previously, block RAM is organized in columns. [Figure 4-2](#) shows the block RAM column arrangement for the XC3S200A. The XC3S50A has a single column of block RAM, located two CLB columns from the left edge of the device. Spartan-3 generation FPGAs larger than the XC3S50 have at least two columns of block RAM, adjacent to the left and right edges of the die, located two columns of CLBs from the I/Os at the edge. In addition to the block RAM columns at the edge, the XC3S4000, XC3S5000, and XC3SD1800A have two additional columns—a total of four columns—nearly equally distributed between the two edge columns. The XC3SD3400A adds a fifth block RAM column, located two CLB columns to the left of the center DCMs. In some devices, the block RAM column is interrupted by DCMs or CLBs. [Table 4-1](#) describes the number of columns and the total amount of block RAM on Spartan-3 generation FPGAs. The edge columns make block RAM particularly useful in buffering or resynchronizing buses entering or leaving the FPGA.

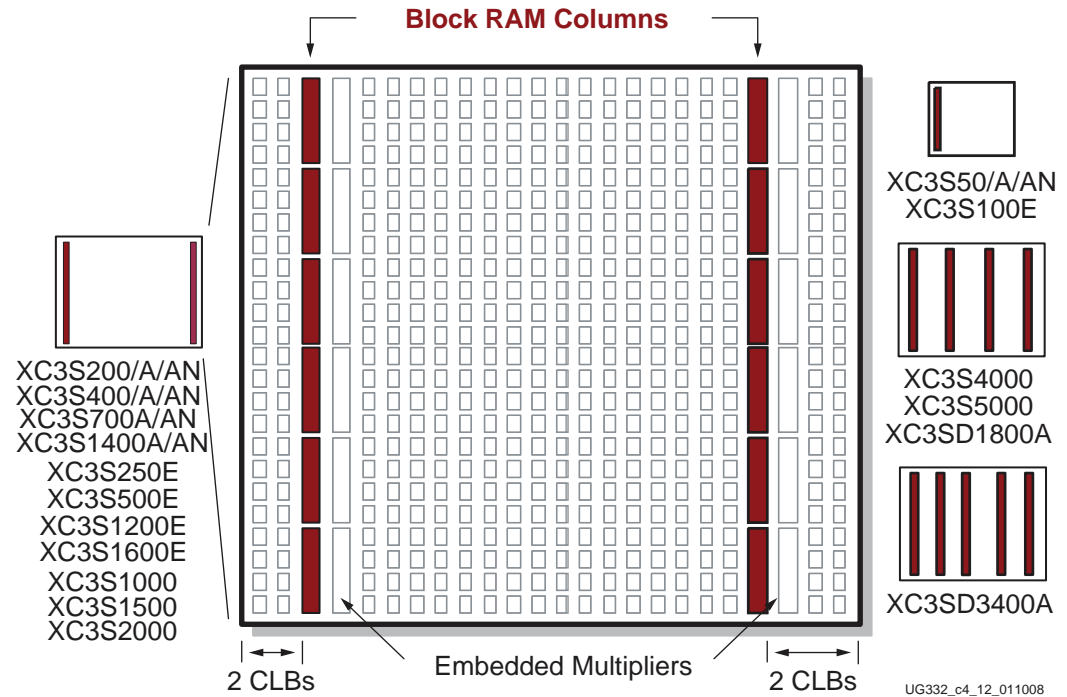


Figure 4-2: Block RAMs Arranged in Columns with Detailed Floorplan of XC3S200

Immediately adjacent to each block RAM is an embedded 18x18 hardware multiplier. Co-locating block RAM and the embedded multipliers improves the performance of some

digital signal processing functions. In the Spartan-3A DSP platform, the multiplier is extended into the DSP48A block.

Special interconnect surrounding the block RAM provides efficient signal distribution for address and data. Furthermore, special provisions allow multiple block RAMs to be cascaded to create wider or deeper memories.

Block RAM/Multiplier Routing Interaction

Each multiplier is located adjacent to an 18 Kbit block RAM and shares some interconnect resources. In the Spartan-3 and Spartan-3E families, configuring an 18 Kbit block RAM for 32/36-bit wide data (512 x 36 mode) prevents use of the associated dedicated multiplier because the lower 16 bits of the A multiplicand input are shared with the upper 16 bits of the block RAM's Port A Data input. Similarly, the lower 16 bits of the B multiplicand input are shared with Port B's Data input.

For more details, see “Multiplier/Block RAM Routing Interaction” in Chapter 11.

Data Flows

Spartan-3 generation block RAM is constructed of true dual-port memory and simultaneously supports all the data flows and operations shown in Figure 4-3. 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.

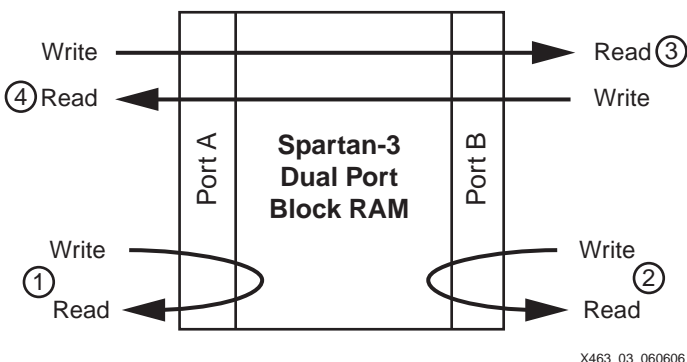


Figure 4-3: Block RAM Support Single- and Dual-Port Data Transfers

Signals

The signals connected to a block RAM primitive divide into four categories, as listed below. Table 4-4 lists the block RAM interface signals, the signal names for both single-port and dual-port memories, and signal direction.

1. Data Inputs and Outputs
2. Parity Inputs and Outputs, available when a data port is byte-wide or wider
3. Address inputs to select a specific memory location
4. Various control signals that manage read, write, or set/reset operations

Table 4-4: Block RAM Interface Signals

Signal Description	Single Port	Dual Port		Direction
		Port A	Port B	
Data Input Bus	DI	DIA	DIB	Input
Parity Data Input Bus (available only for byte-wide and wider organizations)	DIP	DIPA	DIPB	Input
Data Output Bus	DO	DOA	DOB	Output
Parity Data Output (available only for byte-wide and wider organizations)	DOP	DOPA	DOPB	Output
Address Bus	ADDR	ADDRA	ADDRB	Input
Write Enable	WE	WEA	WEB	Input
Clock Enable	EN	ENA	ENB	Input
Synchronous Set/Reset	SSR	SSRA	SSRB	Input
Clock	CLK	CLKA	CLKB	Input
Synchronous / Asynchronous Set/Reset (Spartan-3A DSP FPGA only)	N/A	RSTA	RSTB	Input
Output Register (Spartan-3A DSP FPGA only)	N/A	REGCEA	REGCEB	Input

Data Inputs and Outputs

The total width of a port's data port includes both the data bus and the parity bus, when applicable, as shown in Figure 4-4. In the 512x36 organization, for example, the 36-bit data port width includes four parity bits as the more significant bits followed by the 32 data bits as the less significant bits.

The data and parity input and output signals are always buses; that is, in a 1-bit width configuration, the data input signal is DI[0] and the data output signal is DO[0].

Data Input Bus — DI[#:0] (DIA[#:0], DIB[#:0])

The Data Input bus is the source of data to be written into RAM.

Data at the DI input bus is written to the RAM location specified by the address input bus, ADDR, during a Low-to-High transition on the CLK input, when the clock enable EN and write enable WE inputs are High.

Data Output Bus — DO[#:0] (DOA[#:0], DOB[#:0])

The data output bus, DO, presents the contents of memory cells referenced by the address bus, ADDR, at the active clock edge during a read operation. During a simultaneous write

operation, the behavior of the data output latches is controlled by the `WRITE_MODE` attribute (see [“Read Behavior During Simultaneous Write — `WRITE_MODE`,”](#) page 171).

Parity Inputs and Outputs

Parity is only supported for data paths byte wide and wider.

Although referred to herein as *parity* bits, the parity inputs and outputs have no special functionality and can be used as additional data bits. For example, the parity bits could be used to hold additional information about a data word, tagging the data as code or data, positive or negative values, old or new data, etc.

Block RAM does not contain any special circuitry for generating or checking parity. These functions, if required by the application, are created using CLB logic resources.

Data Input Parity Bus — `DIP[#:0]` (`DIPA[#:0]`, `DIPB[#:0]`)

Data at the DIP input bus is written to the RAM location specified by the address input bus, `ADDR`, during a Low-to-High transition on the `CLK` input, when the clock enable `EN` and write enable `WE` inputs are High.

Data Output Parity Bus — `DOP[#:0]` (`DOPA[#:0]`, `DOPB[#:0]`)

The data output bus, `DOP`, presents the contents of memory cells referenced by the address bus, `ADDR`, at the active clock edge during a read operation. During a simultaneous write operation, the behavior of the data output latches is controlled by the `WRITE_MODE` attribute (see [“Read Behavior During Simultaneous Write — `WRITE_MODE`,”](#) page 171).

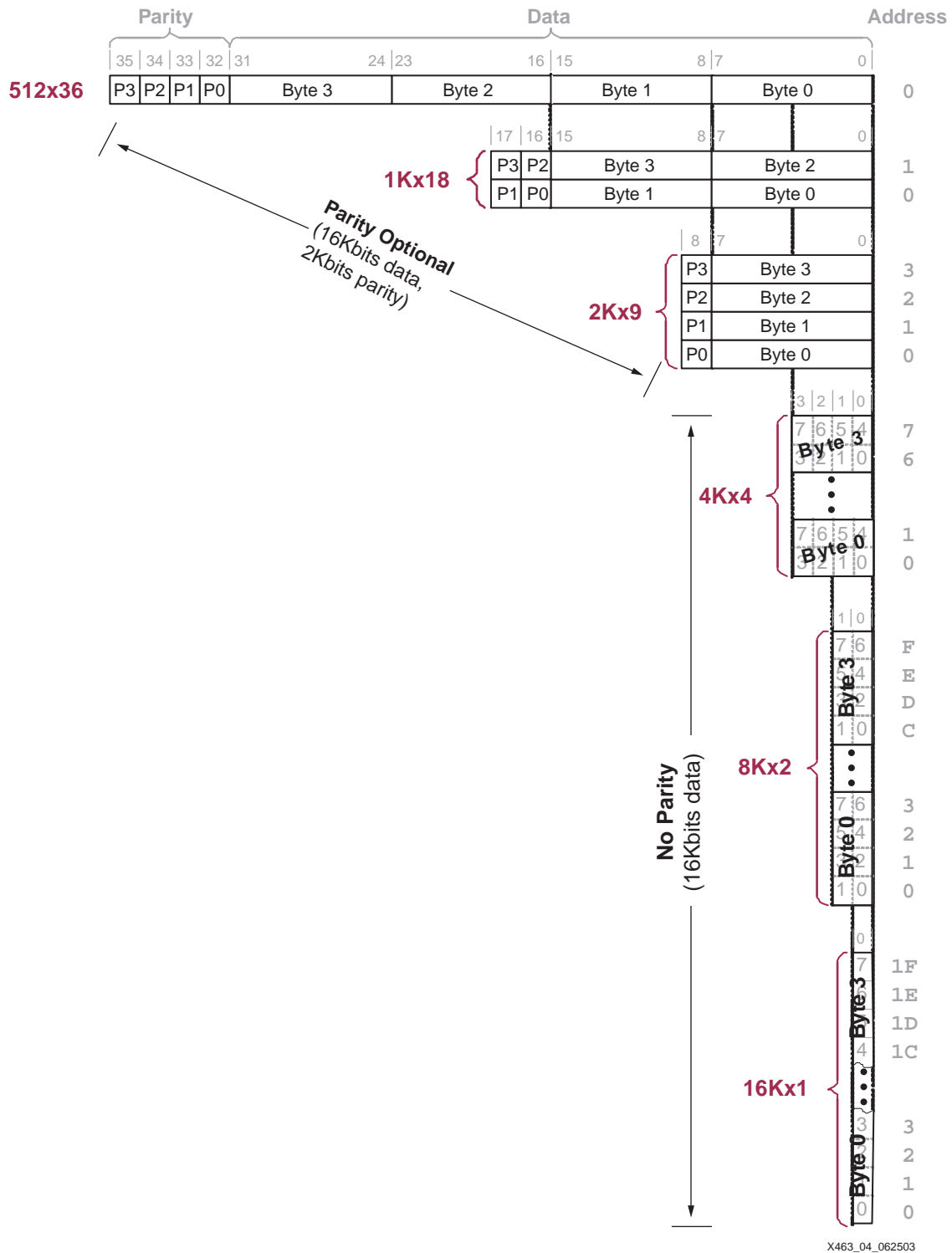


Figure 4-4: Data Organization and Mapping Between Modes

Address Input

As dual-port RAM, both ports operate independently while accessing the same set of 18 Kbit memory cells.

Note: Whenever a block RAM port is enabled (ENA or ENB = High), all 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, or WE is deasserted, including ROM mode. Violating the address setup time (even if write enable is Low) corrupts the data contents of the block RAM. There are some instances in which these requirements might not be able to be met; for instance, if there is a multi-cycle path on the address input signals, or while the clock is stabilizing. Work around this by disabling the port via ENA/ENB during the time that the address inputs do not meet setup and hold requirements. Deasserting ENA/ENB disables the port so that violating the address input setup and hold requirements does not affect block RAM contents. Assert ENA/ENB again when resuming normal read/write functionality.

Address Bus — ADDR[#:0] (ADDRA[#:0], ADDR B[#:0])

The address bus selects the memory cells for read or write operations. The memory depth determines the required address bus width, as shown in [Table 4-8](#).

Control Inputs

Clock — CLK (CLKA, CLKB)

Each port is fully synchronous with independent clock pins. All port input pins have setup time referenced to the port CLK pin. The data bus has a clock-to-out time referenced to the CLK pin. Clock polarity is configurable and is rising edge triggered by default.

With default polarity, a Low-to-High transition on the clock (CLK) input controls read, write, and reset operations.

Enable — EN (ENA, ENB)

The enable input, EN, controls read, write, and set/reset operations. When EN is Low, no data is written and the outputs DO and DOP retain the last state. The polarity of EN is configurable and is active High by default.

When EN is asserted, minus an active synchronous set/reset input or write-enable input, block RAM always reads the memory location specified by the address bus, ADDR, at the rising clock edge.

Write Enable — WE (WEA, WEB)

The write enable input, WE, controls when data is written to RAM. When both EN and WE are asserted at the rising clock edge, the value on the data and parity input buses is written to memory location selected by the address bus.

The data output latches are loaded or not loaded according to the WRITE_MODE attribute.

The polarity of WE is configurable and is active High by default.

All Spartan-3 generation FPGAs support the RAMB16 block RAM primitive that has a single write-enable input that controls write operations regardless of the data width for the configured data organization. See [Figure 4-4, page 161](#) for a diagram of all supported data organizations. [Table 4-5, page 163](#) shows the write-enable behavior for the RAMB16 primitive.

Spartan-3A/3AN FPGAs introduce a new block RAM primitive called RAMB16BWE, essentially a RAMB16 primitive with four independent byte-level write enable inputs. The Spartan-3A DSP FPGA primitive RAMB16BWER has the same byte-level write enable function. As shown in [Table 4-6, page 163](#), the independent write-enable inputs allow an application to write an individual byte or select bytes from a multi-byte data word without affecting the unselected RAM locations. This feature is useful for a variety of applications, especially MicroBlaze processor designs. For 1Kx18 data organizations, connect WE0 with WE2 to select the lower 9 bits and connect WE1 with WE3 to select the upper 9 bits.

Table 4-5: RAMB16 Write Operations (All Spartan-3 Generation FPGAs)

Data Organization	EN	WE	CLK	Function
All (See Figure 4-4)	0	X	X	Block RAM disabled. No operation.
	1	0	↑	Block RAM enabled but no write operation.
	1	1	↑	As appropriate for the block RAM data organization, write data from the DI and DIP input ports to the currently addressed RAM location.

Table 4-6: RAMB16BWE/R Write Operations (Extended Spartan-3A Family FPGAs Only)

Data Organization	EN	Byte-level Write Enables				CLK	Function
		WE3	WE2	WE1	WE0		
All	0	X	X	X	X	X	Block RAM disabled. No operation.
16Kx1 8Kx2 4Kx4 2Kx9	1	0				↑	Block RAM enabled but no write operation.
		1				↑	Write data from the DI and DIP input ports to the currently addressed RAM location.
1Kx18	1	Same as WE1	Same as WE0	1	1	↑	Write 18 bits: Write data from the DI[15:0] and DIP[1:0] input ports to the currently addressed RAM location.
				0	1		Write lower 9 bits: Write data only from the DI[7:0] and DIP[0] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.
				1	0		Write upper 9 bits: Write data only from the DI[15:8] and DIP[1] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.

Table 4-6: RAMB16BWE/R Write Operations (Extended Spartan-3A Family FPGAs Only) (Cont'd)

Data Organization	EN	Byte-level Write Enables				CLK	Function
		WE3	WE2	WE1	WE0		
512x36	1	1	1	1	1	↑	Write 36 bits: Write data from the DI[31:0] and DIP[3:0] input ports to the currently addressed RAM location.
		0	0	0	1	↑	Write lowest 9 bits: Write data only from the DI[7:0] and DIP[0] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.
		0	0	1	0	↑	Write next 9 bits: Write data only from the DI[15:8] and DIP[1] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.
		0	0	1	1	↑	Write lower 18 bits: Write data from the DI[15:0] and DIP[1:0] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.
		1	1	0	0	↑	Write upper 18 bits: Write data from the DI[31:16] and DIP[3:2] input ports to the currently addressed RAM location. Other bits in RAM location unaffected.

Output Register Enable - REGCE (REGCEA, REGCEB) Spartan-3A DSP FPGA Only

The Output Register Write enable input, REGCE, controls when data is written to the RAM Output registers. When both EN and REGCE are asserted at the rising clock edge, the value on the output of the block RAM is written to the block RAM output register.

The polarity of REGCE is configurable and is active High by default.

Output Latch Synchronous Set/Reset — SSR (SSRA, SSRB)

The synchronous set/reset input, SSR, forces the data output latches to the value specified by the SRVAL attribute. When SSR and the enable signal, EN, are High, the data output latches for the DO and DOP outputs are synchronously set to a '0' or '1' according to the SRVAL parameter.

A Synchronous Set/Reset operation does not affect RAM cells and does not disturb write operations on the other port.

The polarity of SSR is configurable and is active High by default.

The SSR input is available on the RAMB16 and RAMB16BWE components. The RAMB16BWER component for the Spartan-3A DSP platform provides the RST input instead.

Output Latch/Register Synchronous/Asynchronous Set/Reset - RST (RSTA, RSTB) - Spartan-3A DSP FPGA Only

The Spartan-3A DSP platform block RAM set/reset input is optionally synchronous or asynchronous and controls both the output latches and the optional output registers. The control pin for this operation is named RST and is available on the RAMB16BWER component.

In synchronous mode, if RST and the enable signal EN are High, the data output latches and optional output registers for the DO and DOP outputs are synchronously set to a '0' or '1' according to the SRVAL parameter.

In asynchronous mode, if RST and the enable signal EN are High, the data output latches and optional output registers for the DO and DOP outputs are asynchronously set to a '0' or '1' according to the SRVAL parameter.

The mode is set by setting the RSTTYPE attribute to "SYNC" for synchronous operation or "ASYNC" for asynchronous operation. The default for RSTTYPE is synchronous. Due to improved timing and circuit stability, it is recommended to always have this set to "SYNC" unless an asynchronous reset is absolutely necessary.

A RST operation does not affect block RAM cells and does not disturb write operations on the other port.

The polarity of RST is configurable and is active High by default.

The RST input is available on the RAMB16BWER component for the Spartan-3A DSP platform. The RAMB16 and RAMB16BWE components provide the SSR input instead.

Global Set/Reset — GSR

The global set/reset signal, GSR, is asserted automatically and momentarily at the end of device configuration. By instantiating the STARTUP primitive, the logic application can also assert GSR to restore the initial FPGA state at any time. The GSR signal initializes the output latches to the INIT value. A GSR signal has no impact on internal memory contents.

Because GSR is a global signal and automatically connected throughout the device, the block RAM primitive does not have a GSR input pin.

Inverting Control Pins

For each port, the four control pins—CLK, EN, WE, and SSR/RST—each have an individual inversion option. Any control signal can be configured as active High or Low, and the clock can be active on a rising or falling edge without consuming additional logic resources.

Unused Inputs

Tie any unused data or address inputs to logic '1'. Connecting the unused inputs High saves logic and routing resources compared to connecting the inputs Low.

Attributes

A block RAM has a number of attributes that control its behavior as shown in [Table 4-7](#) for VHDL and Verilog. The CORE Generator system uses slightly different values, as described below.

Table 4-7: Block RAM Attributes and VHDL/Verilog Attribute Names

Function	VHDL or Verilog Attribute	Default Value
Number of Ports	Defined by instantiating the appropriate RAMB16 primitive	N/A
Memory Organization	Defined by instantiating the appropriate RAMB16 primitive	N/A

Table 4-7: Block RAM Attributes and VHDL/Verilog Attribute Names (Cont'd)

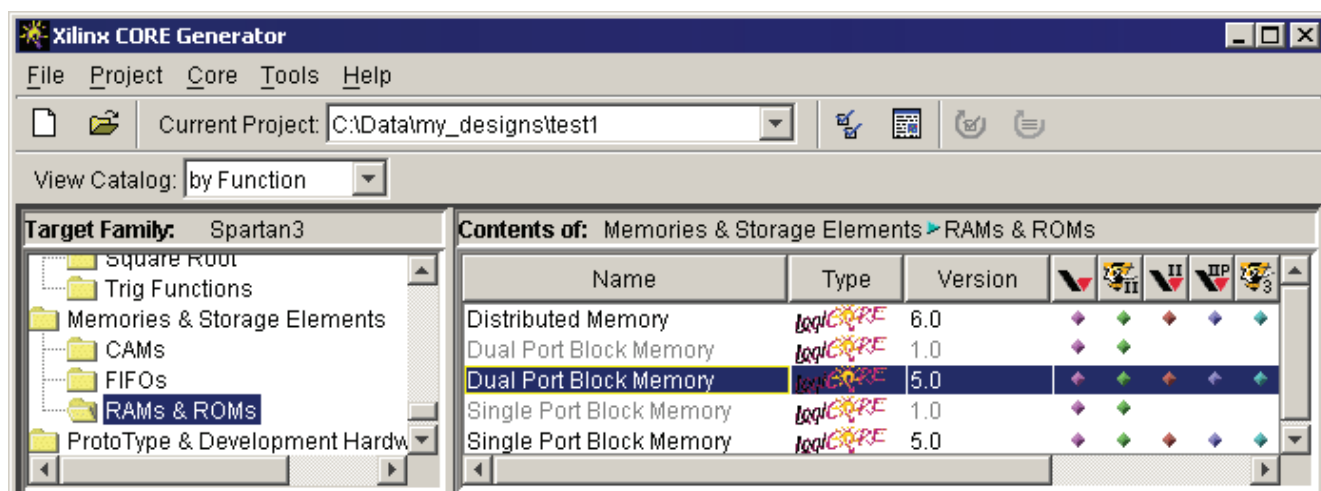
Function	VHDL or Verilog Attribute	Default Value
Initial Content for Data Memory, Loaded during Configuration	INIT_xx	Initialized to zero
Initial Content for Parity Memory, Loaded during Configuration	INITP_xx	Initialized to zero
Data Output Latch Initialization	INIT (single-port) INIT_A, INIT_B (dual-port)	Initialized to zero
Data Output Latch Synchronous Set/Reset Value	SRVAL (single-port) SRVAL_A, SRVAL_B (dual-port)	Reset to zero
Data Output Latch Behavior during Write	WRITE_MODE	WRITE_FIRST
Block RAM Location	LOC	N/A
Reset Type (Spartan-3A DSP FPGA only)	RSTTYPE	SYNC

Number of Ports

Although physically dual-port memory, each block RAM performs as either single-port or dual-port memory. The method to specify the number of ports depends on the design entry tool.

CORE Generator System

As shown in Figure 4-5, the Xilinx CORE Generator system provides module generators for various types of memory blocks. Choose single- or dual-port block memories or use the higher-level functions to create FIFOs, content-addressable memories (CAMs), and so forth.



X463_05_060606

Figure 4-5: Selecting a Block RAM Function in CORE Generator System

VHDL or Verilog Instantiation

The Xilinx design libraries contain single- and dual-port memory primitives similar to those shown in [Figure 4-1](#). Select among the various primitives to choose single- or dual-port memory, as well as the memory organization or aspect ratio of the memory. See [Table 4-8](#) and [Table 4-9](#) for single-port and dual-port block RAM primitives, respectively.

Memory Organization/Aspect Ratio

The data organization or aspect ratio of a RAM block is configurable, as shown in [Table 4-8](#). If the data path is byte-wide or wider, then the block RAM also provides additional bits to support parity for each byte. Consequently, a 1Kx18 memory organization is 18 bits wide with 16 bits (two bytes) allocated to data plus two parity bits, one for each byte. Also, the physical amount of memory accessible from a port depends on the memory organization. For memories byte-wide and wider, there are 18K memory bits accessible. For narrower memories, only 16K bits are accessible due to the lack of parity bits in these organizations. Essentially, 16K bits are allocated to data, 2K bits to parity on the 18 Kbit block RAM. See [Figure 4-4](#) for details on data mapping for and between each memory organization.

Table 4-8: 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

CORE Generator System — Memory Size

The CORE Generator system creates a wide variety of memories with very flexible aspect ratios. Unlike the actual block RAM primitive, the CORE generator system does not differentiate between data and parity bits and considers all bits data bits. For dual-port memories, each port can have different organizations or aspect ratios.

Within the CORE Generator system, locate the Memory Size group and enter the desired memory organization, as shown in [Figure 4-6](#).

Figure 4-6: Selecting Memory Width and Depth in CORE Generator System

VHDL or Verilog Instantiation

The aspect ratio is defined at design time by specifying or instantiating the appropriate SelectRAM component. Table 4-8 indicates the SelectRAM component for single-port RAM. For single-port RAM, the proper component name is RAMB16_*Sn*, where *n* is the data path width including both the data bits plus parity bits. For example, a 1Kx18 single-port RAM uses component RAMB16_S18. In this example, *n*=18 because there are 16 data bits plus 2 parity bits.

Selecting a dual-port memory is slightly more complex because the two memory ports can have different aspect ratios. For dual-port RAM, the proper component name is RAMB16_*Sm_Sn*, where *m* is the data path width for Port A and *n* is the width for Port B. For example, using the suffix shown in Table 4-9, if Port A is organized as 2Kx9 and Port B is organized as 1Kx18, then the proper dual-port RAM component is RAMB16_S9_S18. In this example, *m*=9 and *n*=18.

Table 4-9: 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

Address and Data Mapping Between Two Ports

In dual-port mode, both ports access the same set of memory cells. However, both ports can have the same or different memory organization or aspect ratio. Figure 4-4 shows how the same data set might appear with different aspect ratios.

There are extra bits available to store parity for memory organizations that are byte-wide or wider. The extra parity bits are designed to be associated with a particular byte and these parity bits appear as the more-significant bits on the data port. For example, if a x36 data word (32 data, 4 parity) is addressed as two x18 halfwords (16 data, 2 parity), the parity bits associated with each data byte are mapped within the block RAM to appropriate parity bits. The same effect happens when the x36 data word is mapped as four x9 words. The extra parity bits are not available if the data port is configured as x4, x2, or x1.

The following formulas provide the starting and ending address for data when the two ports have different memory organizations. Find the starting and ending addresses for Port X given the address and port width of Port Y and the port width of Port X.

$$START_ADDRESS_X = INTEGER\left(\frac{ADDRESS_Y \cdot WIDTH_Y}{WIDTH_X}\right)$$

$$END_ADDRESS_X = INTEGER\left(\frac{((ADDRESS_Y + 1) \cdot WIDTH_Y) - 1}{WIDTH_X}\right)$$

If, due the memory organization, one port includes parity bits and the other does not, then the above equations are invalid and the values for width should only include the data bits. The parity bits are not available on any port that is less than 8 bits wide.

Content Initialization

By default, block RAM is initialized with all zeros during the device configuration sequence. However, the contents can also be initialized with user-defined data. Furthermore, the RAM contents are protected against spurious writes during configuration.

CORE Generator System — Load Init File

To specify the initial RAM contents for a CORE Generator block RAM function, create a coefficients (.coe) file. A simple example of a coefficients file appears in [Figure 4-7](#). At a minimum, define the radix for the initialization data—i.e., base 2, 10, or 16—and then specify the RAM contents starting with the data at location 0, followed by data at subsequent locations.

```
memory_initialization_radix=16;
memory_initialization_vector= 80, 0F, 00, 0B, 00, 0C, ..., 81;
```

Figure 4-7: A Simple Coefficients File (.coe) Example

To include the coefficients file, locate the appropriate section in the CORE Generator wizard and check **Load Init File**, as shown in [Figure 4-8](#). Then, click **Load File** and select the coefficients file.

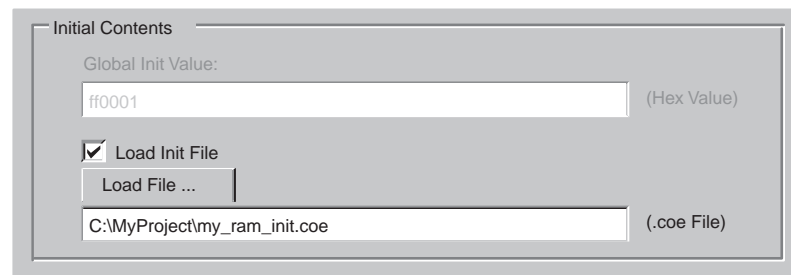


Figure 4-8: Specifying Initial RAM Contents in CORE Generator System

VHDL or Verilog Instantiation — INIT_xx, INITP_xx

For VHDL and Verilog instantiation, there are two different types of initialization attributes. The **INIT_xx** attributes define the initial contents of the data memory locations. The **INITP_xx** attributes define the initial contents of the parity memory locations.

The **INIT_xx** attributes on the instantiated primitive define the initial memory contents. There are 64 initialization attributes, named **INIT_00** through **INIT_3F**. Each **INIT_xx** attribute is a 64-digit (256-bit) hex-encoded bit vector. The memory contents can be partially initialized and any unspecified locations are automatically completed with zeros.

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

Given $yy = \text{convert_hex_to_decimal}(xx)$, **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: $31 * 256 = 7936$

Table 4-10: VHDL/Verilog RAM Initialization Attributes for Block RAM

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

The `INITP_xx` attributes define the initial contents of the memory cells corresponding to parity bits, *i.e.*, those bits that connect to the DIP/DOP buses. By default these memory cells are also initialized to all zeros.

The eight initialization attributes from `INITP_00` through `INITP_07` represent the memory contents of parity bits. Each `INITP_xx` is a 64-digit (256-bit) hex-encoded bit vector and behaves like an `INIT_xx` attribute. The same formula calculates the bit positions initialized by a particular `INITP_xx` attribute.

Data Output Latch Initialization

The block RAM output latches can be initialized to a user-specified value immediately after configuration or whenever the global set/reset signal, GSR, is asserted. For dual-port memories, there is a separate initialization value for each port.

If no value is specified, the output latch is initialized to zero.

CORE Generator System — Global Init Value

Figure 4-9 describes how to specify the initial value for data output latches in the CORE Generator system. The value, specified in hexadecimal, should include one bit per the specified data width. For dual-port memories, there is a separate initialization value for each port.



Figure 4-9: Specifying Initial Value for Block RAM Data Output Latches

VHDL or Verilog Instantiation — INIT (INIT_A and INIT_B)

For VHDL or Verilog, the INIT attribute (or INIT_A and INIT_B for dual-port memories) defines the output latch value after configuration. The INIT (or INIT_A and INIT_B) attribute specifies the initial value for the data and, if applicable, the parity bits. [Figure 4-4](#) shows the expected bit format for each memory organization with parity bits—if applicable—as the more significant bits followed by the data bits. For example, the initialization value for a 2Kx9 memory would be nine bits wide and would include one parity bit followed by eight data bits. These attributes are hex-encoded bit vectors and the default value is 0.

Data Output Latch Synchronous Set/Reset Value

When the synchronous set/reset input, SSR (RST for the RAMB16BWER), is asserted, the data output latches are set or reset according to the set/reset value attribute. For dual-port memories, there is a separate initialization value for each port.

If no value is specified, the output latch is reset to zero during a valid Synchronous Set/Reset operation.

For the RAMB16BWER, the optional output register is also set or reset with the output latch.

CORE Generator System — Init Value (SINIT)

[Figure 4-10](#) describes how to specify the synchronous set/reset value for data output latches in the CORE Generator system. Check the **SINIT pin** and then specify the synchronous set/reset value in hexadecimal, with one bit per the specified data width. For dual-port memories, there is a separate value for each port.

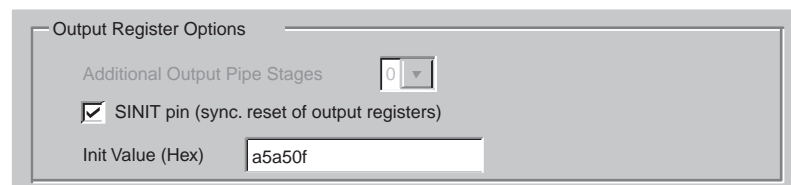


Figure 4-10: Specifying the Output Data Latch Set/Reset Value

VHDL or Verilog Instantiation — SRVAL (SRVAL_A and SRVAL_B)

For VHDL or Verilog, the SRVAL attribute (or SRVAL_A and SRVAL_B for dual-port memories) defines the output latch value after configuration. The SRVAL (or SRVAL_A and SRVAL_B) attribute specifies the initial value for the data and, if applicable, the parity bits. [Figure 4-4](#) shows the expected bit format for each memory organization with parity bits—if applicable—as the more significant bits followed by the data bits. These attributes are hex-encoded bit vectors, and the default value is 0.

Read Behavior During Simultaneous Write — WRITE_MODE

To maximize data throughput and utilization of the dual-port memory at each clock edge, block RAM supports one of three write modes for each memory port. These different modes determine which data is available on the output latches after a valid write clock edge to the same port. The default mode, [WRITE_FIRST](#), provides backwards compatibility with the older Virtex, Virtex-E, and Spartan-IIE FPGA architectures and is

also the default behavior for Virtex-II and Virtex-II Pro devices. However, [READ_FIRST](#) mode is the most useful as it increases the efficiency of block RAM at each clock cycle, allowing designs to use maximum bandwidth. In [READ_FIRST](#) mode, a memory port supports simultaneous read and write operations to the same address on the same clock edge, free of any timing complications.

[Table 4-11](#) outlines how the `WRITE_MODE` setting affects the output data latches on the same port, and how it affects the output latches on the opposite port during a simultaneous access to the same address.

Table 4-11: `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.

Mode selection is set by configuration. One of these three modes is set individually for each port by an attribute. The default mode is [WRITE_FIRST](#).

WRITE_FIRST or Transparent Mode (Default)

The `WRITE_FIRST` mode is the default operating mode for backward compatibility reasons. For new designs, [READ_FIRST](#) mode is recommended.

In this mode, the input data is written into the addressed RAM location memory and simultaneously stored in the data output latches, resulting in a transparent write operation, as shown in [Figure 4-11](#). The `WRITE_FIRST` mode provides backwards compatibility with the 4 Kbit block RAMs on Virtex/Virtex-E and Spartan-II/Spartan-IIE FPGAs and is also the default mode for Virtex-II/Virtex-II Pro FPGA block RAMs.

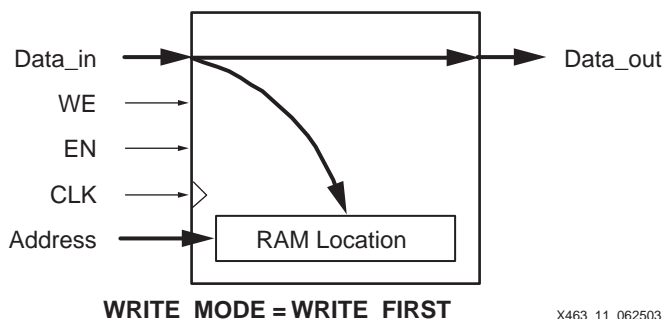


Figure 4-11: Data Flow during a `WRITE_FIRST` Write Operation

Figure 4-12 demonstrates that a valid write operation during a valid read operation results in the write data appearing on the data output.

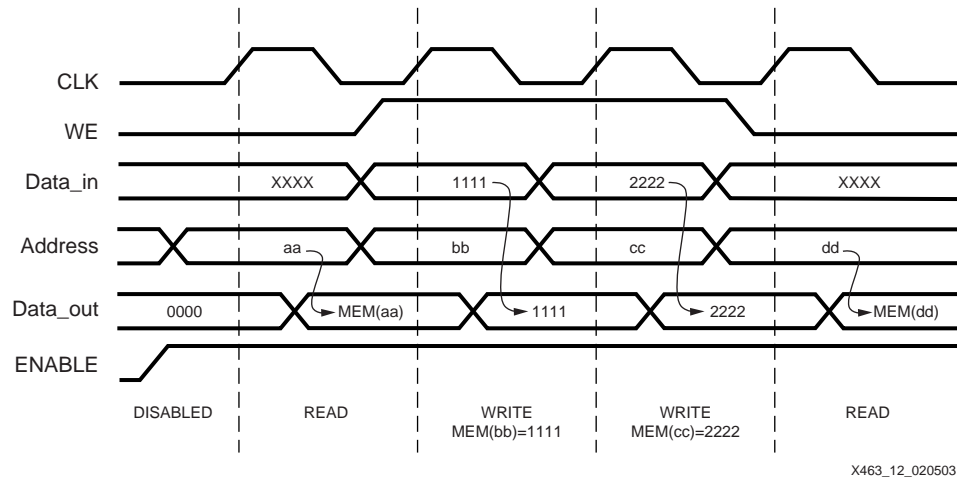


Figure 4-12: WRITE_FIRST Mode Waveforms

READ_FIRST or Read-Before-Write Mode

In READ_FIRST mode, data previously stored at the write address appears on the output latches, while the new input data is stored in memory, resulting in a read-before-write operation shown in Figure 4-13. The older RAM data appears on the data output while the new RAM data is stored in the specified RAM location. READ_FIRST mode is the recommended operating mode.

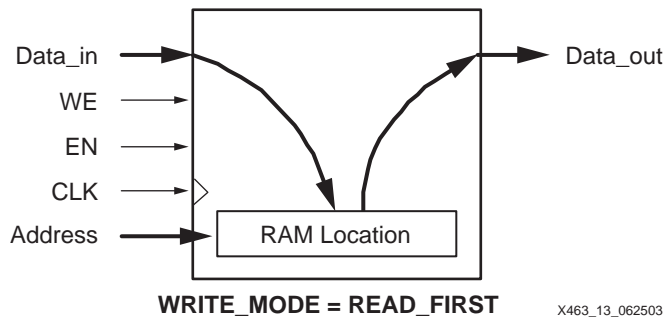


Figure 4-13: Data Flow during a READ_FIRST Write Operation

Figure 4-14 demonstrates that the older RAM data always appears on the data output, regardless of a simultaneous write operation.

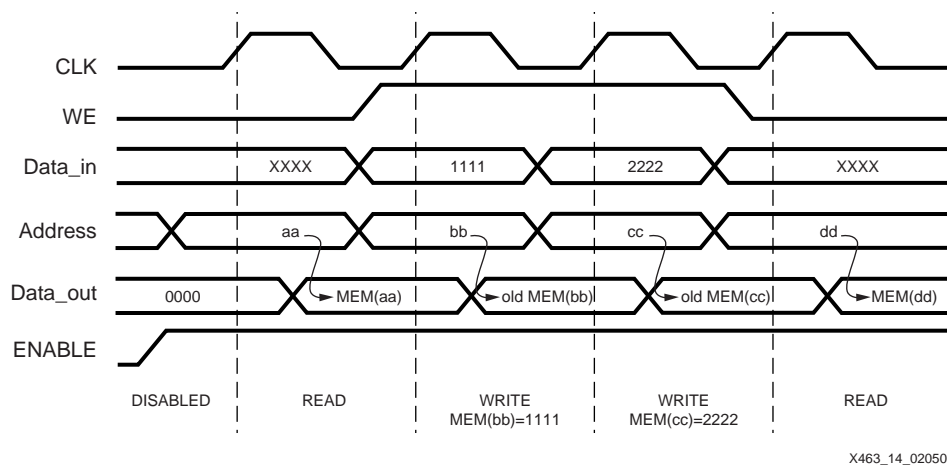


Figure 4-14: **READ_FIRST Mode Waveforms**

This mode is particularly useful for building circular buffers and large, block-RAM-based shift registers. Similarly, this mode is useful when storing FIR filter taps in digital signal processing applications. Old data is copied out from RAM while new data is written into RAM.

NO_CHANGE Mode

In NO_CHANGE mode, the output latches are disabled and remain unchanged during a simultaneous write operation, as shown in Figure 4-15. This behavior mimics that of simple synchronous memory where a memory location is either read or written during a clock cycle, but not both.

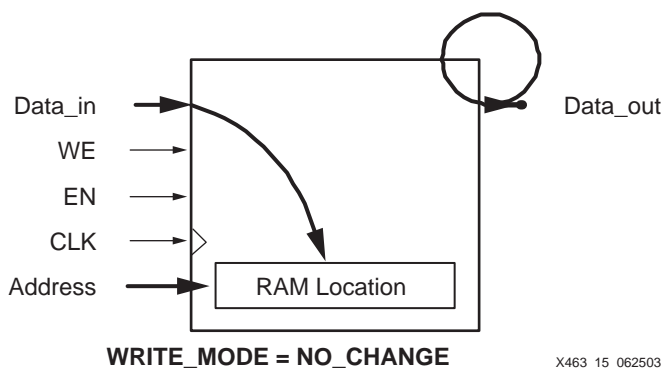


Figure 4-15: **Data Flow during a NO_CHANGE Write Operation**

The NO_CHANGE mode is useful in a variety of applications, including those where the block RAM contains waveforms, function tables, coefficients, and so forth. The memory can be updated without affecting the memory output.

Figure 4-16 shows that the data output retains the last read data if there is a simultaneous write operation on the same port.

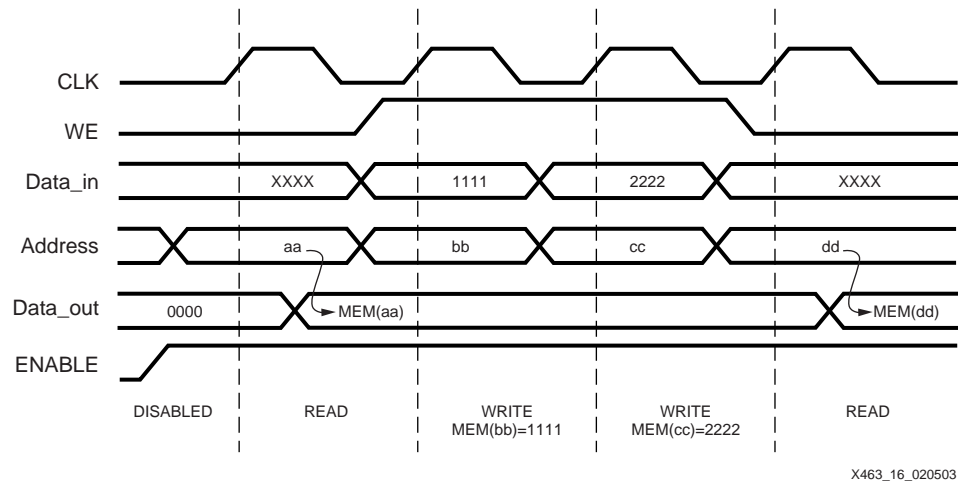


Figure 4-16: NO_CHANGE Mode Waveforms

CORE Generator System — Write Mode

To specify the WRITE_MODE in the CORE Generator system, locate the settings for Write Mode as shown in Figure 4-17. Select between Read After Write ([WRITE_FIRST](#)), Read Before Write ([READ_FIRST](#)) or No Read On Write ([NO_CHANGE](#)).

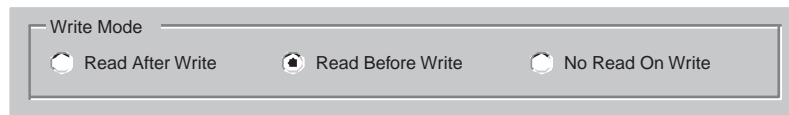


Figure 4-17: Selecting the Write Mode in CORE Generator System

VHDL or Verilog Instantiation — WRITE_MODE

When instantiating block RAM, specify the write mode via the WRITE_MODE attribute. Acceptable values include WRITE_FIRST, READ_FIRST, and NO_CHANGE, as demonstrated in the examples in the appendices.

Location Constraints (LOC)

In general, it is best to allow the Xilinx ISE® software to assign a block RAM location. However, block RAMs can be constrained to specific locations on a Spartan-3 device using an attached LOC property. Block RAM placement locations are device-specific and differ from the convention used for naming CLB locations, allowing LOC properties to transfer easily from array to array.

The LOC properties use the following form:

LOC = RAMB16_X#Y#

The **RAMB16_X0Y0** is the lower-left block RAM location on the device, as shown in Figure 4-18. The upper-right block RAM location depends on **n**, the number of block RAM columns, and **m**, the number of block RAM rows, as provided in Table 4-1, page 153. The Spartan-3A DSP platform has four or five columns of block RAM, similar to the XC3S4000 and XC3S5000 devices.

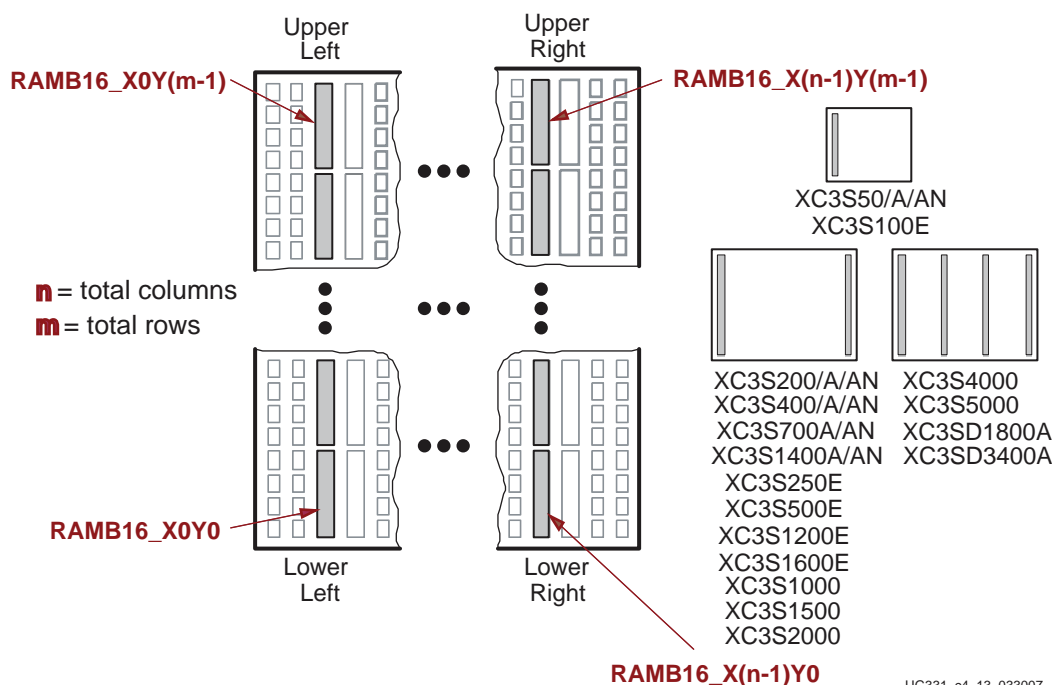


Figure 4-18: Block RAM LOC Coordinates

Location attributes cannot be specified directly in the CORE Generator system. However, location constraints can be added to VHDL or Verilog instantiations.

Block RAM Operation

Table 4-12 describes the behavior of block RAM and assumes that all control signals use their default, active-High behavior. However, the control signals can be inverted in the design if necessary. The table and following text describe the behavior for a single memory port. In dual-port mode, both ports perform as independent single-port memories.

All read and write operations to block RAM are synchronous. All inputs have a set-up time relative to clock and all outputs have a clock-to-output time.

Table 4-12: Block RAM Function Table

Input Signals								Output Signals		RAM Contents	
GSR	EN	SSR/RST	WE	CLK	ADDR	DIP	DI	DOP	DO	Parity	Data
Immediately After Configuration											
Loaded During Configuration								X	X	INITP _{xx} ²	INIT _{xx} ²
Global Set/Reset Immediately after Configuration											
1	X	X	X	X	X	X	X	INIT ³	INIT	No Chg	No Chg
RAM Disabled											
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
Synchronous Set/Reset											
0	1	1	0	↑	X	X	X	SRVAL ⁴	SRVAL	No Chg	No Chg

Table 4-12: Block RAM Function Table (Cont'd)

Input Signals								Output Signals		RAM Contents	
GSR	EN	SSR/RST	WE	CLK	ADDR	DIP	DI	DOP	DO	Parity	Data
Synchronous Set/Reset during Write RAM											
0	1	1	1	↑	addr	pdata	Data	SRVAL	SRVAL	RAM(addr) ← pdata	RAM(addr) ← data
Read RAM, no Write Operation											
0	1	0	0	↑	addr	X	X	RAM(pdata)	RAM(data)	No Chg	No Chg
Write RAM, Simultaneous Read Operation											
0	1	0	1	↑	addr	pdata	Data	WRITE_MODE = WRITE_FIRST ⁵ (default)			
								pdata	data	RAM(addr) ← pdata	RAM(addr) ← data
								WRITE_MODE = READ_FIRST ⁶ (recommended)			
								RAM(data)	RAM(data)	RAM(addr) ← pdata	RAM(addr) ← pdata
								WRITE_MODE = NO_CHANGE ⁷			
								No Chg	No Chg	RAM(addr) ← pdata	RAM(addr) ← pdata

Notes:

1. No Chg = No Change, addr = address to RAM, data = RAM data, pdata = RAM parity data.
2. Refer to "Content Initialization," page 169.
3. Refer to "Data Output Latch Initialization," page 170.
4. Refer to "Data Output Latch Synchronous Set/Reset Value," page 171.
5. Refer to "WRITE_FIRST or Transparent Mode (Default)," page 172.
6. Refer to "READ_FIRST or Read-Before-Write Mode," page 173.
7. Refer to "NO_CHANGE Mode," page 174.

RAM Contents Initialized During Configuration

The initial RAM contents, if specified, are loaded during the Spartan-3 FPGA configuration process. If no contents are specified, the RAM cells are loaded with zero. The RAM contents are protected against spurious writes during configuration.

Global Set/Reset Initializes Data Output Latches Immediately After Configuration or Global Reset

Immediately following configuration, the Spartan-3 device begins its start-up procedure and asserts the global set/reset signal, GSR, to initialize the state of all flip-flops and registers. The initial contents of the block RAM output latches, INIT, are asynchronously loaded at this time. The GSR signal does not change or re-initialize the RAM contents.

Enable Input Activates or Disables RAM

If the block RAM is disabled—i.e., EN is Low—then the block RAM retains its present state. The enable input must be High for any other operations to proceed.

Synchronous Set/Reset Initializes Data Output Latches

If the block RAM is enabled (EN is High) and the Synchronous Set/Reset signal is asserted High, then the data output latches are initialized at the next rising clock edge. The SRVAL attribute defines the synchronous set/reset state for the data output latches. This operation is different the operation caused by the global set/reset signal, GSR, immediately after configuration. The synchronous set/reset input affects the specific RAM block whereas the GSR signal affects the entire device.

Simultaneous Write and Synchronous Set/Reset Operations

If a simultaneous write operation occurs during the synchronous set/reset operation, then the data on the DI and DIP inputs is stored at the RAM location specified by the ADDR input. However, the data output latches are initialized to the SRVAL attribute value as described immediately above.

Read Operations Occur on Every Clock Edge When Enable is Asserted

Read operations are synchronous and require a clock edge and an asserted clock enable. The data output behavior depends on whether or not a simultaneous write operation occurs during the read cycle.

If no simultaneous write cycle occurs during a valid read cycle, then the read address is registered on the read port and the data stored in RAM at that address is simply loaded into the output latches after the RAM access interval passes.

However, if there is a simultaneous write cycle during the read cycle, then the output behavior depends on which of the three write modes is selected, as described immediately below.

Write Operations Always Have Simultaneous Read Operation, Data Output Latches Affected

During a Write operation, a simultaneous Read operation occurs. The WRITE_MODE attribute determines the behavior of the data output latches during the Write operation (refer to [“Read Behavior During Simultaneous Write — WRITE_MODE,” page 171](#)). By default, WRITE_MODE is WRITE_FIRST and the data output latches and the addressed RAM locations are updated with the input data during a simultaneous Write operation. When WRITE_MODE is READ_FIRST, the output latches are updated with the data previously stored in the addressed RAM location and the new data on the DI and DIP inputs is stored at the address RAM location. When WRITE_MODE is NO_CHANGE, the data output latches are unaffected by a simultaneous Write operation and retain their present state.

General Characteristics

- A write operation requires only one clock edge.
- A read operation requires only one clock edge.
- All inputs are registered with the port clock and have a setup-to-clock timing specification.
- All outputs have a read-through function or one of three read-during-write functions, depending on the state of the WE pin. The outputs relative to the port clock are available after the clock-to-out timing interval.

- Block RAM cells are true synchronous RAMs and do not have a combinatorial path from the address to the output.
- The ports are completely independent of each other without arbitration. Each port has its own clocking, control, address, read/write functions, initialization, and data width.
- Output ports are latched with a self-timed circuit, guaranteeing glitch-free read operations. The state of the output port does not change until the port executes another read or write operation.

Functional Compatibility with Other Xilinx FPGA Families

The block RAM on Spartan-3 generation FPGAs is functionally identical to block RAM on the Xilinx Virtex-II/Virtex-II Pro FPGA families. Consequently, design tools that support Virtex-II and Virtex-II Pro FPGA block RAM also support with Spartan-3 generation FPGAs.

Extended Spartan-3A family FPGAs, while remaining fully backwards compatible with Spartan-3/3E FPGAs, also add byte-level write enable controls, similar to those found on Virtex-4 FPGAs. The Spartan-3A DSP FPGAs also include a block RAM output register similar to those found in the Virtex-4 FPGAs.

Dual-Port RAM Conflicts and Resolution

As a dual-port RAM, the block RAM allows both ports to simultaneously access the same memory cell. Potentially, conflicts arise under the following conditions:

1. If the clock inputs to the two ports are asynchronous, then conflicts occur if clock-to-clock setup time requirements are violated.
2. Both memory ports write different data to the same RAM location during a valid write cycle.
3. If a port uses WRITE_MODE=[NO_CHANGE](#) or [WRITE_FIRST](#), a write to the port invalidates the read data output latches on the opposite port.

If Port A and Port B different memory organizations and consequently different widths, only the overlapping bits are invalid when conflicts occur.

Timing Violation Conflicts

When one port writes to a given memory cell, the other port must not address that memory cell—either for a write or a read operation—within the clock-to-clock setup window, which is equivalent to the block RAM minimum clock period ($T_{BPWH} + T_{BPWL}$), specified in the Spartan-3 generation FPGA family data sheets. [Figure 4-19](#) describes this situation where both ports operate from asynchronous clock inputs.

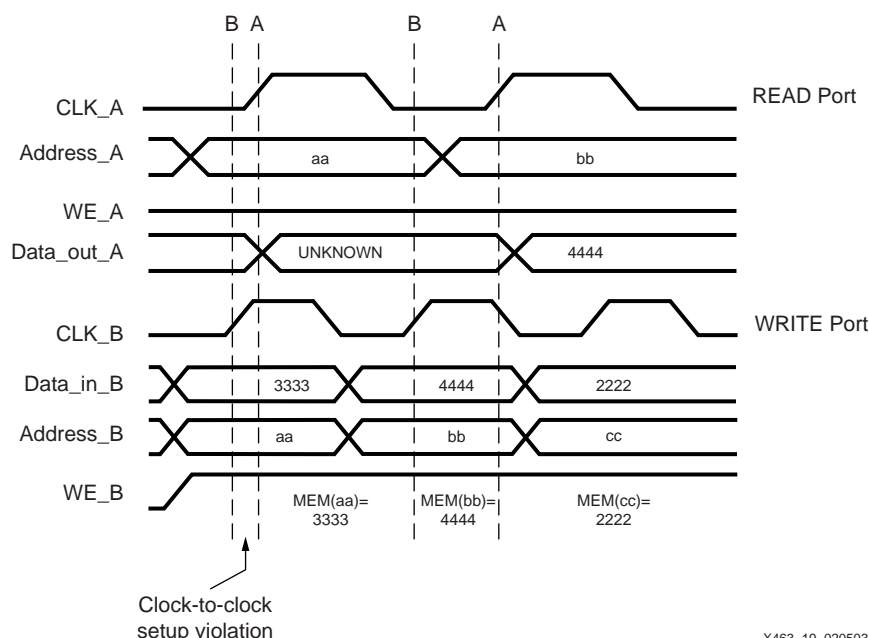


Figure 4-19: Clock-to-Clock Timing Conflicts

The first rising edge on CLK_A violates the clock-to-clock setup parameter, because it occurs too soon after the last CLK_B clock edge. The write operation on port B is valid because Data_in_B, Address_B, and WE_B all had sufficient setup time before the rising edge on CLK_B. Unfortunately, the read operation on port A is invalid because it depends on the RAM contents being written to Address_B and the read clock, CLK_A, happened too soon after the write clock, CLK_B.

On the second rising edge of CLK_B, there is another valid write operation to port B. The memory location at address (bb) contains 4444. Data on the Data_out_A port is still invalid because there has not been another rising clock edge on CLK_A. The second rising edge of CLK_A reads the new data at location (bb), which now contains 4444. This time, the read operation is valid because there has been sufficient setup time between CLK_B and CLK_A.

Simultaneous Writes to Both Ports with Different Data Conflicts

If both ports write simultaneously into the same memory cell with different data, then the data stored in that cell becomes invalid, as outlined in [Table 4-13](#).

Table 4-13: RAM Conflicts During Simultaneous Writes to Same Address

Input Signals								RAM Contents	
Port A				Port B					
WEA	CLKB	DIPA	DIA	WEB	CLKA	DIPB	DIB	Parity	Data
1	↑	DIPA	DIA	1	.	DIPB	DIB	?	?

Notes:

1. ADDRA=ADDRB, ENA=1, ENB=1, DIPA ≠ DIPB, DIA ≠ DIB, ?=Unknown or invalid data.

Write Mode Conflicts on Output Latches

Potential conflicts occur when one port writes to memory and the opposite port reads from memory. Write operations always succeed, and the write port's output data latches behave as described by the port's `WRITE_MODE` attribute. If the write port is configured with `WRITE_MODE` set to `NO_CHANGE` or `WRITE_FIRST`, then a write operation to the port invalidates the data output latches on the opposite port, as shown in [Table 4-14](#).

Using the `READ_FIRST` mode does not cause conflicts on the opposite port.

Table 4-14: Conflicts to Output Latches Based on `WRITE_MODE`

Input Signals								Output Signals			
Port A				Port B				Port A		Port B	
WEA	CLKA	DIPA	DIA	WEB	CLKB	DIPB	DIB	DOPA	DOA	DOPB	DOB
WRITE_MODE_A=NO_CHANGE											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	No Chg	No Chg	?	?
WRITE_MODE_B=NO_CHANGE											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	No Chg	No Chg
WRITE_MODE_A=WRITE_FIRST											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	DIPA	DIA	?	?
WRITE_MODE_B=WRITE_FIRST											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	DIPB	DIB
WRITE_MODE_A=WRITE_FIRST, WRITE_MODE_B=WRITE_FIRST											
1	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	?	?

Notes:

1. ADDRA=ADDRB, ENA=1, ENB=1, ?=Unknown or invalid data

Conflict Resolution

There is no dedicated monitor to arbitrate the result of identical addresses on both ports. The application must time the two clocks appropriately. However, conflicting simultaneous writes to the same location never cause any physical damage.

Block RAM Design Entry

Various tools help create Spartan-3 FPGA block RAM designs, two of which are the Xilinx CORE Generator system and VHDL or Verilog instantiation of the appropriate Xilinx library primitives.

Xilinx CORE Generator System

The Xilinx CORE Generator system provides both a Single Port Block Memory and a Dual Port Block Memory module generator, as shown in [Figure 4-5](#). Both module generators support RAM, ROM, and Write Only functions, according to the control signals that are selected. Any size memory that can be created in the architecture is supported.

Both modules are parameterizable as with most CORE Generator modules. To create a module, specify the component name and choose to include or exclude control inputs, and choose the active polarity for the control inputs. For the Dual-Port Block Memory, once the organization or aspect ratio for Port A is selected, only the valid options for Port B are displayed.

Optionally, specify the initial memory contents. Unless otherwise specified, each memory location initializes to zero. Enter user-specified initial values via a Memory Initialization File, consisting of one line of binary data for every memory location. A default file is generated by the CORE Generator system. Alternatively, create a coefficients file (.coe), which not only defines the initial contents in a radix of 2, 10, or 16, but also defines all the other control parameters for the CORE Generator system.

The output from the CORE Generator system includes a report on the options selected and the device resources required. If a very deep memory is generated, some external multiplexing might be required, and these resources are reported as the number of logic slices required. In addition, the software reports the number of bits available in block RAM that are less than 100% utilized. For simulation purposes, the CORE Generator system creates VHDL or Verilog behavioral models.

- **CORE Generator:** [Single-Port Block Memory](#) module (RAM or ROM)
- **CORE Generator:** [Dual-Port Block Memory](#) module (RAM or ROM)

VHDL and Verilog Instantiation

VHDL and Verilog synthesis-based designs can either infer or directly instantiate block RAM, depending on the specific logic synthesis tool used to create the design.

Inferring Block RAM

Some VHDL and Verilog logic synthesis tools, such as the Xilinx Synthesis Tool (XST) and Synplify Synplify both infer block RAM based on the hardware described. The Xilinx ISE Project Navigator includes templates for inferring block RAM in your design. To use the templates within Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Synthesis Templates → RAM** from the selection tree. Finally, select the preferred block RAM template.

It is still possible to directly instantiate block RAM, even if portions of the design infer block RAM.

Instantiation Templates

For VHDL- and Verilog-based designs, various instantiation templates are available to speed development. Within the Xilinx ISE Project Navigator, select **Edit → Language Templates** from the menu, and then select **VHDL** or **Verilog**, followed by **Component Instantiation → Block RAM** from the selection tree.

The appendices include example code showing how to instantiate block RAM in both VHDL and Verilog.

In VHDL, each template has a component declaration section and an architecture section. Each part of the template must be inserted within the VHDL design file. The port map of the architecture section must include the signal names used in the application.

The SelectRAM_Ax templates (with x = 1, 2, 4, 9, 18, or 36) are single-port modules and instantiate the corresponding RAMB16_Sx module.

SelectRAM_Ax_By templates (with x = 1, 2, 4, 9, 18, or 36 and y = 1, 2, 4, 9, 18, or 36) are dual-port modules and instantiate the corresponding RAMB16_Sx_Sy module.

Initialization in VHDL or Verilog Codes

Block RAM structures can be initialized in VHDL or Verilog code for both synthesis and simulation. For synthesis, the attributes are attached to the block RAM instantiation and are copied within the EDIF output file compiled by Xilinx tools. The VHDL code simulation uses a **generic** parameter to pass the attributes. The Verilog code simulation uses a **defparam** parameter to pass the attributes.

The VHDL and Verilog examples in the appendices illustrate these techniques.

Block RAM Applications

Typically, block RAM is used for a variety of local storage applications. However, the following section describes additional, perhaps less obvious block RAM capabilities, illustrating some powerful capabilities to spur the imagination.

Creating Larger RAM Structures

Block SelectRAM columns have specialized routing to allow cascading blocks with minimal routing delays. Wider or deeper RAM structures incur a small delay penalty. For examples of how to create wider block memories, see application note [XAPP229: Wider Block Memories](#), which includes a reference design.

Block RAM as Read-Only Memory (ROM)

By tying the write enable input Low, block RAM optionally functions as registered block ROM. The ROM outputs are synchronous and require a clock input and perform exactly like a block RAM read operation. The ROM contents are defined by the initial contents at design time.

After design compilation, the ROM contents can also be updated using the Data2BRAM utility described below.

FIFOs

First-In, First-Out (FIFO) memories, also known as elastic stores, are perhaps the most common application of block RAM, other than for random data storage. FIFOs typically resynchronize data, either between two different clock domains, or between two parts of a system that have different data rates, even though they operate from a single clock. The Xilinx CORE Generator system provides two parameterizable FIFO modules, one a synchronous FIFO where both the read and write clocks are synchronous to one another and the other an asynchronous FIFO where the read and write clocks are different.

Application note XAPP261 demonstrates that the FIFO read and write ports can be different data widths, integrating the data width converter into the FIFO.

Application note XAPP291 describes a self-addressing FIFO that is useful for throttling data in a continuous data stream.

- **CORE Generator:** [Synchronous FIFO](#) module
- **CORE Generator:** [Asynchronous FIFO](#) module
- [XAPP258: FIFOs Using Block RAM](#), includes reference design

- [XAPP261](#): *Data-Width Conversion FIFOs Using Block RAM Memory*, includes reference design
- [XAPP291](#): *Self-Addressing FIFO*

Storage for Embedded Processors

Block RAM also enables efficient embedded processor applications. RAM performs a variety of functions in an embedded processor such as those listed below.

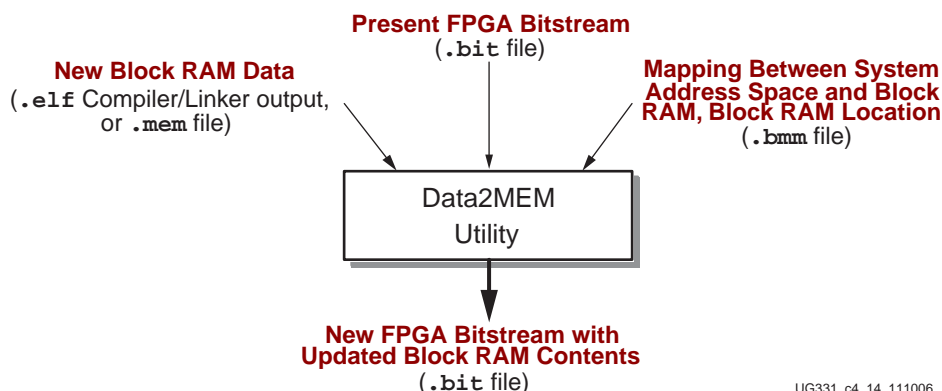
- Register file for processor register set, although for some processors, distributed RAM might be a preferred solution.
- Stack or LIFO for stack-based architectures and for call stacks.
- Fast, local code storage. The fast access time to internal block RAM significantly boosts the performance of embedded processors. However, on-chip storage is limited by the number of available block RAMs.
- Large dual-ported mailbox memory shared with external processor or DSP device.
- Temporary trace buffers (see “[Circular Buffers, Shift Registers, and Delay Lines](#)”) to ease and enhance application debugging.

Updating Block RAM/ROM Content by Directly Modifying Device Bitstream

In a typical design flow, the initial contents of block RAM/ROM is defined at design time and compiled into the device bitstream that is downloaded to and configures a Spartan-3 FPGA.

However, for some applications, the actual memory contents might not be known when the bitstream is created or might change later. One example is if a processor embedded with the Spartan-3 FPGA uses block RAM to store program code. To avoid recompiling the FPGA design just to incorporate a code change, Xilinx provides a utility called [Data2MEM](#) that updates an existing FPGA bitstream with new block RAM/ROM contents.

As shown in [Figure 4-20](#), the inputs to [Data2MEM](#) include the new RAM contents—typically the output from the embedded processor compiler/linker, the present FPGA bitstream, and a file that describes the mapping between the system address space and the addressing used on the individual block RAMs and the physical location of each block RAM.



UG331_c4_14_111006

Figure 4-20: The Data2BRAM Utility Updates Block RAM Contents in a Bitstream

Two Independent Single-Port RAMs Using One Block RAM

Some applications might require more single-port RAMs than there are RAM blocks on the device. However, a simple trick allows a single block RAM to behave as if it were two, completely independent single-port memories, effectively doubling the number of RAM blocks on the device. The penalty is that each RAM block is only half the size of the original block, up to 9K bits total.

Figure 4-21 shows how to create two independent single-port RAMs from one block RAM. Tie the most-significant address bit of one port High and the most-significant address bit of the other port Low. Both ports evenly split the available RAM between them.

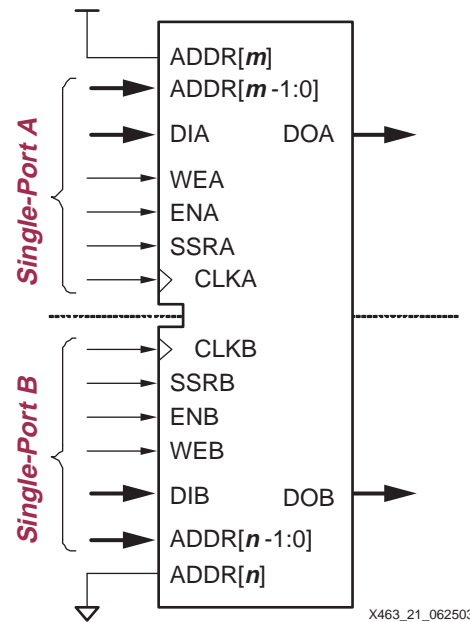


Figure 4-21: One Block RAM Becomes Two Independent Single-Port RAMs

Both ports are independent, each with its own memory organization, data inputs and outputs, clock input, and control signals. For example, Port A could be 256x36 while Port B is 2Kx4.

Figure 4-21 splits the available memory evenly between the two ports. With additional logic on the upper address lines, the memory can be split into other ratios.

A 256x72 Single-Port RAM Using One Block RAM

Figure 4-22 illustrates how to create a 256-deep by 72-bit wide single-port RAM using a single block RAM. As in the previous example, the memory array is split into halves. One half contains the lower 36 bits, and the upper half stores the upper 36 bits, effectively creating a 72-bit wide memory.

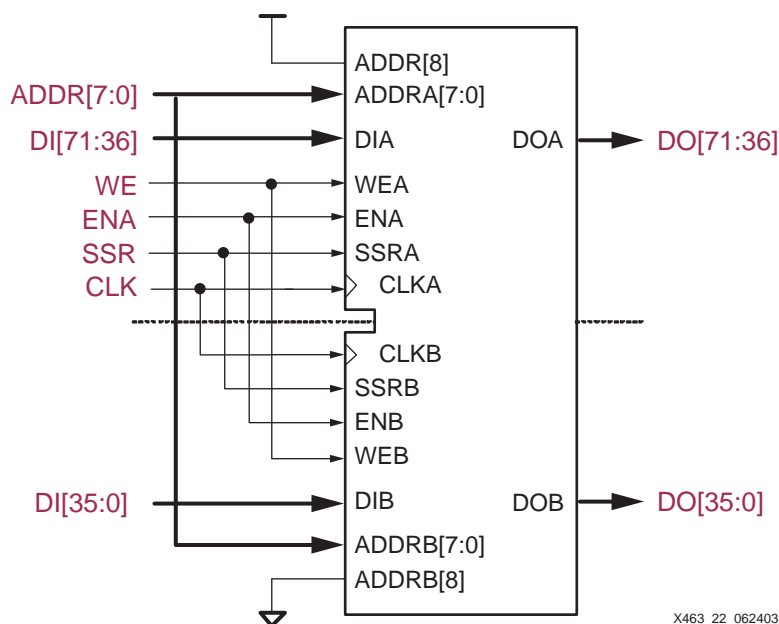


Figure 4-22: A 256x72 Single-Port RAM Using a Single Block RAM

The most-significant address line, ADDR[8] is tied High on one port and Low on the other. Both ports share the same the address inputs, control inputs, and clock input.