Block Memory Generator v2.4

Introduction

The Xilinx LogiCORE™ Block Memory Generator is an advanced memory constructor, generating area and performance optimized memories using embedded block RAM resources in Xilinx FPGAs. Available through the CORE Generator™ system, the core allows users to quickly create optimized memories to leverage the performance and features of block RAMs in Xilinx FPGAs.

Features

- Generates single-port RAM, simple dual-port RAM, true dual-port RAM, single-port ROM, dual-port ROM
- Performance up to 450 MHz
- Supports data widths from 1 to 1152 bits
- Supports memory depths from 2 to 9M words (limited only by memory resources on selected part)
- Supports configurable port aspect ratios for dual-port configurations
- Supports read-to-write aspect ratios in Virtex™-4 and Virtex-5 FPGAs
- Optimized algorithm for minimum block RAM resource utilization
- Configurable memory initialization
- Supports individual write enable per byte in Virtex-5 and Virtex-4 devices with or without parity.
- Optimized VHDL and Verilog behavioral models for fast simulation times
- Structural simulation models for precise simulation of memory behaviors
- Selectable operating mode per port: WRITE_FIRST, READ_FIRST, or NO_CHANGE
- Supports ECC (built-in Hamming error correction)

LogiCORE Facts	
Core Specifics	
Supported Device Family	Virtex-5, Virtex-4, Virtex-II Pro, Virtex-II, Spartan™-3E, Spartan-3A/3AN/3A DSP, Spartan-3
Package	All
Speed Grade	All
Performance	Varied, based on core parameters
Core Resources	
Block RAM	Varied, based on core parameters
DCM	None
BUFG	None
IOBs/RocketIO™	None
PPC	None
IOB-FF/TBUFs	None
Provided with Core	
Documentation	Product Specification
Design File Formats	NGC netlist
Design Tool Requirements	
Xilinx Implementation Tools	ISE™ 9.1i
Simulation Models (SWIFT-compliant tool required)	VHDL Structural Verilog Structural VHDL Behavioral[1] Verilog Behavioral[1]
Synthesis	XST 9.1i
Support	
Provided by Xilinx, Inc. @ www.xilinx.com	

1. Behavior models do not precisely model collision behavior. See "Simulation Models" on page 19 for details.

Overview

The Block Memory Generator core uses the embedded Block Memory primitives available in Xilinx FPGAs, extending the functionality and capabilities of a single primitive to memories of arbitrary widths and depths. The sophisticated algorithms behind the Block Memory Generator core produce optimized solutions for a wide range of configurations, providing convenient access to memories optimized for your needs.

The Block Memory Generator has two fully independent ports that access a shared memory space. Both A and B ports have a write and a read interface. In Virtex-5 and Virtex-4, architectures, all four interfaces can be uniquely configured, each having a different data width. When not using all four interfaces, the user can select a simplified memory configuration (such as Single-Port Memory or Simple Dual-Port Memory), allowing the core to more efficiently use available resources.

The Block Memory Generator is not entirely backward-compatible with the Single-Port Block Memory and Dual-Port Block Memory cores. The differences are discussed in "Interfacing with Older Memory Cores" on page 35.

Applications

The Block Memory Generator core is used to create customized memories to suit any application. Typical applications include:

- **Single-Port Memory**: Processor scratch RAM, very large look-up tables.
- **Dual-Port Memory**: Content addressable memories, FIFOs, multi-processor storage.
- **Read-Only Memory**: Program code storage, initialization ROM.

Feature Summary

Memory Types

The Block Memory Generator core uses embedded block RAM to generate the following types of memories:

- Single-port RAM
- Simple dual-port RAM
- True dual-port RAM
- Single-port ROM
- Dual-port ROM

For dual-port memories, each port operates independently. Operating mode, clock frequency, and optional pins are selectable per port.

Selectable Memory Algorithm

The core concatenates block RAM primitives according to one of the following algorithms:

- **Minimum Area Algorithm:** The memory is generated using the minimum number of block RAM primitives. Both data and parity bits are utilized.
- **Fixed Primitive Algorithm:** The memory is generated using only one type of block RAM primitive. For a complete list of primitives available for each device family, see the data sheet for that family.

Configurable Width and Depth

The Block Memory Generator can generate memory structures from 1 to 1152 bits wide, and at least two locations deep. The maximum depth of the memory is limited only by the number of block RAM primitives in the target device.

Selectable Port Aspect Ratios

The core supports the same port aspect ratios as the block RAM primitives:

- In all supported device families, the A port width may differ from the B port width by a factor of 1, 2, 4, 8, 16, or 32.
- In Virtex-5 and Virtex-4 FPGA-based memories, the read width may differ from the write width by a factor of 1, 2, 4, 8, 16, or 32 for each port. The maximum ratio between any two of the data widths (`DINA`, `DOUTA`, `DINB`, and `DOUTB`) is 32:1.

Memory Initialization

The memory contents can be optionally initialized using a memory coefficient (COE) file or by using the default data option. A COE file can define the initial contents of each individual memory location, while the default data option defines the initial content of all locations.

Selectable Operating Mode per Port

The Block Memory Generator supports the block RAM primitive operating modes: WRITE FIRST, READ FIRST, and NO CHANGE. Each port may be assigned an operating mode.

Optional Output Registers

The Block Memory Generator provides two optional stages of output registering to increase memory performance. The core supports the Virtex-5, Virtex-4, and Spartan-3A DSP embedded block RAM registers as well as registers implemented in the FPGA fabric. See "Output Register Configurations" on page 40 for more information about using these registers.

Optional Synchronous Set/Reset Pin

The core provides optional set/reset pins (`SSRA` and `SSRB`) pin per port that synchronously initialize the read output to a programmable value.

Optional Enable Pin

The core provides optional port enable pins (`ENA` and `ENB`) to control the operation of the memory. When deasserted, no read, write, or reset operations are performed on the respective port. If the enable pins are not used, it is assumed that the port is always enabled.

Optional Byte-Write Enable

In Virtex-5 and Virtex-4 FPGA-based memories, the Block Memory Generator core provides byte-write support for memory widths of 8-bit (no parity) or 9-bit multiples (with parity).

ECC – Hamming Error Correction

Single Port RAM and Simple Dual Port RAM memory types support the Virtex-5 ECC capability of the Virtex-5 block RAM primitives. The ECC memory automatically detects single- and double-bit errors, and is able to auto-correct the single-bit errors.

Functional Description

The Block Memory Generator is used to build custom memory modules from block RAM primitives in Xilinx FPGAs. The core implements an optimal memory by arranging block RAM primitives based on user selections, automating the process of primitive instantiation and concatenation. Using the CORE Generator graphical user interface (GUI), the user can configure the core and rapidly generate a highly optimized custom memory solution.

Memory Type

The Block Memory Generator core can create five types of memories: Single-port RAM, Simple dual-port RAM, True dual-port RAM, Single-port ROM, and Dual-port ROM. Figures 1 through 5 illustrate the signals available for each memory type. Optional pins are shown in italics.

For each configuration, optimizations are made within the core to minimize the total resources used. For example, a simple dual-port RAM with symmetric ports can utilize the special simple dual-port RAM primitive in Virtex-5. This could save as much as 50% of the block RAM resources for memories that are 512 words deep or less.

The single-port ROM allows read access to the memory space through a single port, as shown in Figure 1.

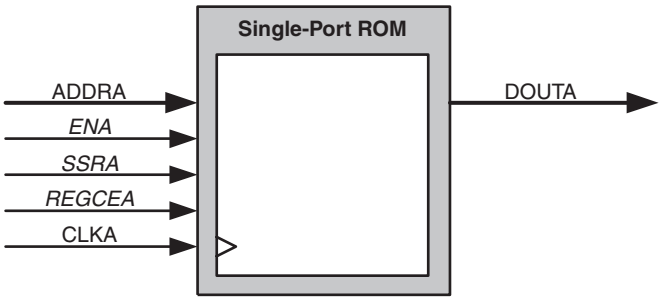

Figure 1: **Single-port ROM**

The dual-port ROM allows read access to the memory space through two ports, as shown in Figure 2.

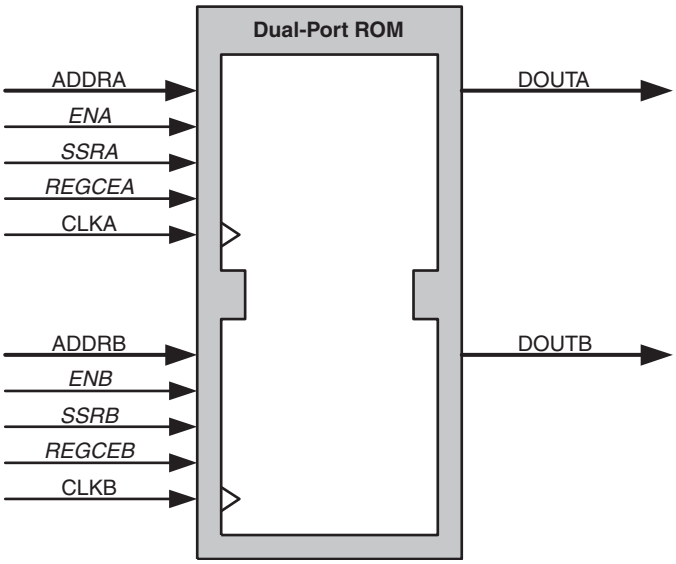

Figure 2: **Dual-port ROM**

The single-port RAM allows read and write access to the memory through a single port, as shown in Figure 3.

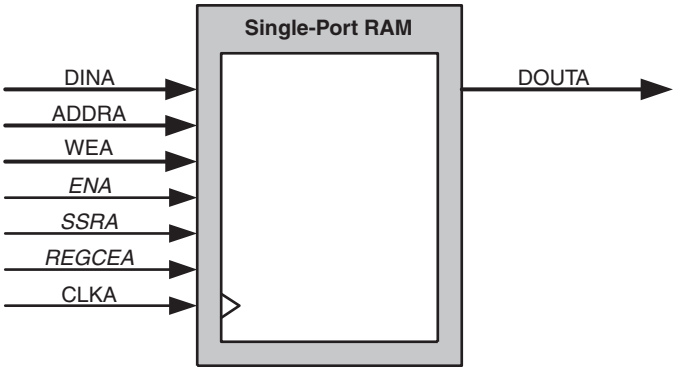

Figure 3: **Single-port RAM**

The simple dual-port RAM shown in Figure 4 provides two ports, A and B. Write access to the memory is allowed via port A, and read access is allowed via port B.

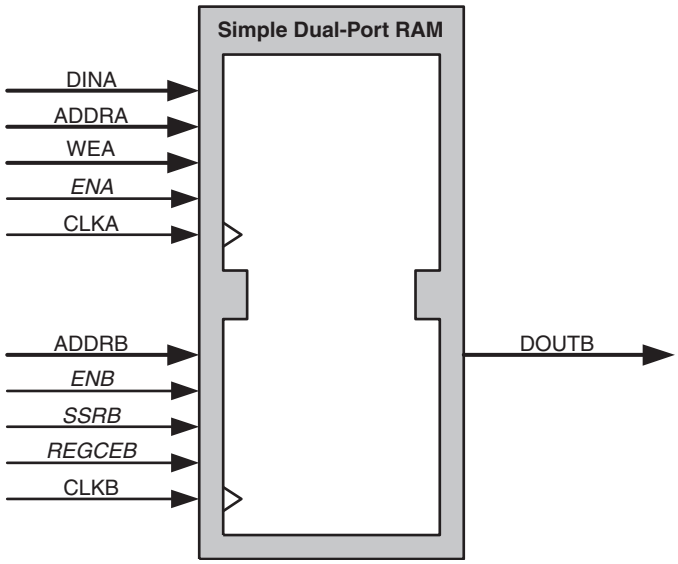

Figure 4: **Simple Dual-port RAM**

The true dual-port RAM shown in Figure 5 provides two ports, A and B. Read and write accesses to the memory are allowed on either port.

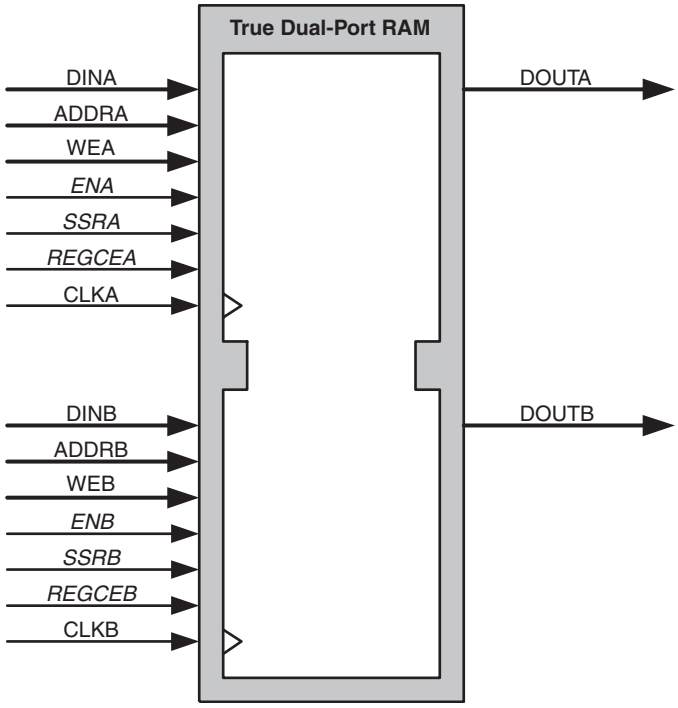

Figure 5: **True Dual-port RAM**

Selectable Memory Algorithm

The Block Memory Generator core arranges block RAM primitives according to one of two algorithms: the minimum area algorithm and the fixed primitive algorithm.

Minimum Area Algorithm

The minimum area algorithm provides a highly optimized solution, resulting in a minimum number of block RAM primitives used, while reducing output multiplexing. Figure 6 depicts two examples of memories built using the minimum area algorithm.

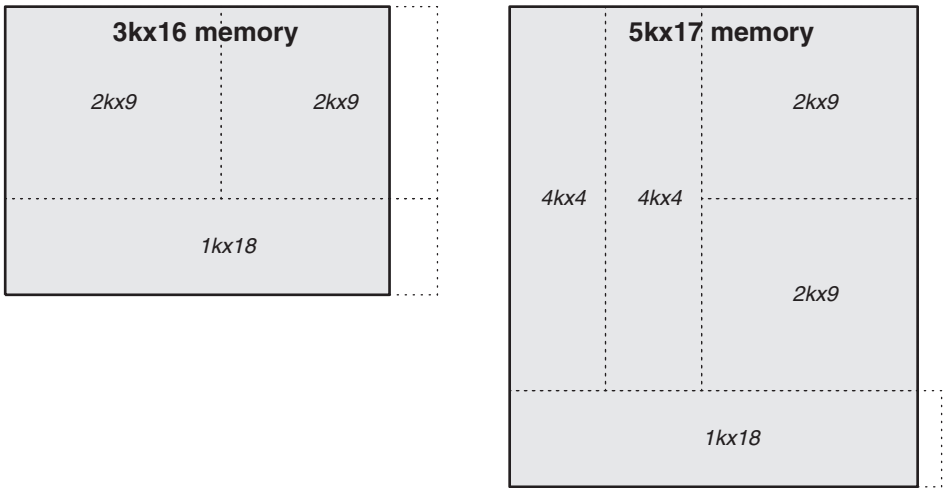

Figure 6: **Examples of the Minimum Area Algorithm**

In the first example, a 3kx16 memory is implemented using three block RAMs. While it may have been possible to concatenate three 1kx18 block RAMs in depth, this would require more output multiplexing. The minimum area algorithm maximizes performance in this way while maintaining minimum block RAM usage.

The second example, a 5kx17 memory, further demonstrates how the algorithm can pack block RAMs efficiently to use the fewest resources while maximizing performance by reducing output multiplexing.

Fixed Primitive Algorithm

The fixed primitive algorithm allows the user to select a single block RAM primitive type. The core will build the memory by concatenating this single primitive type in width and depth. It is useful in systems that require a fixed primitive type. Figure 7 depicts two 3kx16 memories, one built using the 2kx9 primitive type, the other built using the 4kx4 primitive type.

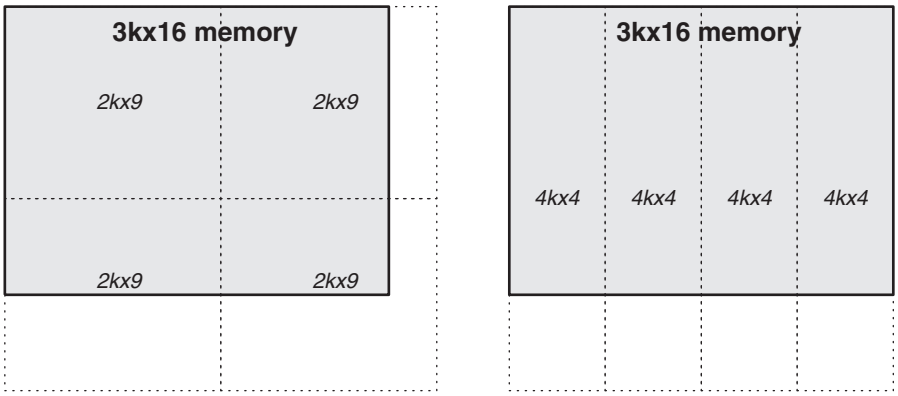

Figure 7: **Examples of the Fixed Primitive Algorithm**

Note that both implementations use four block RAMs, and that some of the resources utilized extend beyond the usable memory space. It is up to the user to decide which primitive type is best for their application.

The fixed primitive algorithm provides a choice of 16kx1, 8kx2, 4kx4, 2kx9, 1kx18, and 512x36 primitives. The primitive type selected is used to guide the construction of the total user memory space. Whenever possible, optimizations are made automatically that use deeper embedded memory structures to improve performance. The following table shows the primitives used to construct a memory given the specified architecture and primitive selection.

Table 1: **Memory Primitives Used—Based on Architecture**

Architecture	Primitive Selection	Primitives Used
Virtex-II, Spartan-3, Spartan-3A DSP	16kx1	16kx1
Virtex-II, Spartan-3, Spartan-3A DSP	8kx2	8kx2
Virtex-II, Spartan-3, Spartan-3A DSP	4kx4	4kx4
Virtex-II, Spartan-3, Spartan-3A DSP	2kx9	2kx9
Virtex-II, Spartan-3, Spartan-3A DSP	1kx18	1kx18
Virtex-II, Spartan-3, Spartan-3A DSP	512x36	512x36
Virtex-4	16kx1	32kx1, 16kx1
Virtex-4	8kx2	8kx2
Virtex-4	4kx4	4kx4
Virtex-4	2kx9	2kx9
Virtex-4	1kx18	1kx18
Virtex-4	512x36	512x36
Virtex-5	16kx1	64kx1, 32kx1, 16kx1
Virtex-5	8kx2	16kx2, 8kx2
Virtex-5	4kx4	8kx4, 4kx4

Table 1: **Memory Primitives Used—Based on Architecture *(Continued)***

Architecture	Primitive Selection	Primitives Used
Virtex-5	2kx9	4kx9, 2kx9
Virtex-5	1kx18	2kx18, 1kx18
Virtex-5	512x36	1kx36
Virtex-5	256x72	512x72

When using data width aspect ratios, the primitive type dimensions are chosen with respect to the A port write width. Note that primitive selection may limit port aspect ratios as described in "Aspect Ratio Limitations" on page 12. When using the byte write feature in Virtex-5 and Virtex-4, only the 2kx9, 1kx18, and 512kx36 primitive choices are available.

Selectable Width and Depth

The Block Memory Generator generates memories with widths from 1 to 1152 bits, and with depths of eight or more words. The memory is built by concatenating block RAM primitives, and total memory size is limited only by the number of block RAMs on the target device.

Write operations to out-of-range addresses are guaranteed not to corrupt data in the memory, while read operations to out-of-range addresses will return invalid data. Note that the set/reset function should not be asserted while accessing an out-of-range address as this also results in invalid data on the output.

Operating Mode

The operating mode determines which data word is presented on the output during a write operation. Each port is assigned an operating mode. The Block Memory Generator supports the three operating modes supported by the block RAM primitives. These operating modes are described in detail below. For more information, refer to the block RAM section of the user guide for the specific device family.

- **Write First Mode:** In WRITE_FIRST mode, the input data is simultaneously written into memory and driven on the data output, as shown in Figure 8. This transparent mode offers the flexibility of using the data output bus during a write operation on the same port.

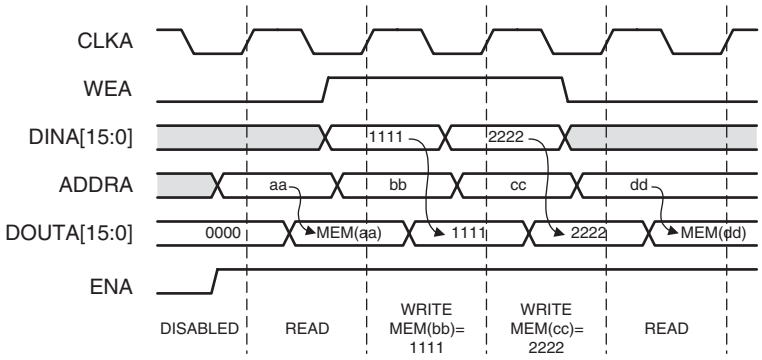

Figure 8: **Write First Mode Example**

Note that the WRITE_FIRST operation for Virtex-5 and Virtex-4 is affected by the optional read-to-write aspect ratio and byte-write features. For more details, see "Virtex-5 and Virtex-4 Write First Mode" on page 13.

- **Read First Mode:** In READ_FIRST mode, data previously stored at the write address appears on the data output, while the input data is being stored in memory. This read-before-write behavior is illustrated in Figure 9.

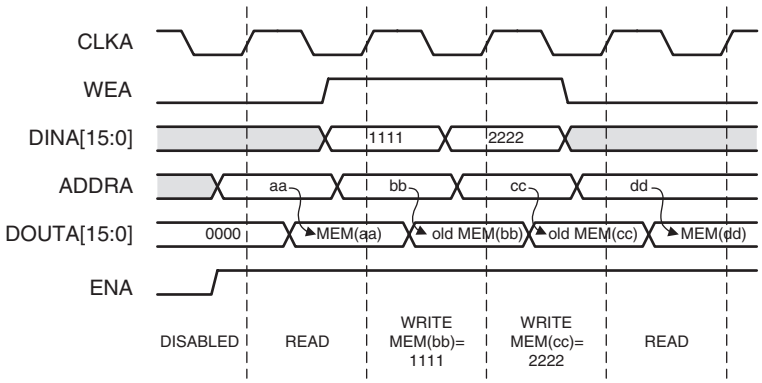

Figure 9: **Read First Mode Example**

- **No Change Mode:** In NO_CHANGE mode, the output latches remain unchanged during a write operation. As shown in Figure 10, the data output is still the previous read data and is unaffected by a write operation on the same port.

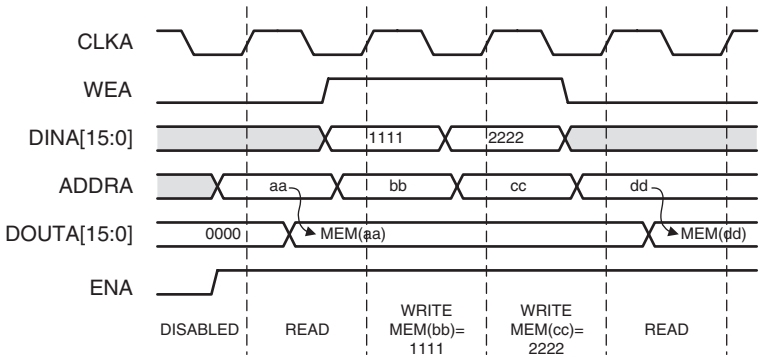

Figure 10: **No Change Mode Example**

Data Width Aspect Ratios

The Block Memory Generator supports data width aspect ratios. This allows the port A data width to be different than the port B data width, as described in Port Aspect Ratios. In Virtex-5 and Virtex-4 FPGA-based memories, all four data busses (DINA, DOUTA, DINB, and DOUTB) can have different widths, as described in "Virtex-5 and Virtex-4 Read-to-Write Aspect Ratios" on page 11.

The limitations of the data width aspect ratio feature (some of which are imposed by other optional features) are described in "Aspect Ratio Limitations" on page 12. The CORE Generator GUI ensures only valid aspect ratios as selected.

Port Aspect Ratios

The Block Memory Generator supports port aspect ratios of 1:32, 1:16, 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1, 16:1, and 32:1. The port A data width can be up to 32 times larger than the port B data width, or vice versa. The smaller data words are arranged in little-endian format, as illustrated in Figure 11.

Port Aspect Ratio Example

Consider a true dual-port RAM of 32x2048, which is the A port width and depth. From the perspective of an 8-bit B port, the depth would be 8192. The ADDRA bus is 11 bits, while the ADDRB bus is 13 bits. The data is stored little-endian, as shown in Figure 11. Note that A_n is the data word at address n, with respect to the A port. B_n is the data word at address n with respect to the B port. A_0 is comprised of B_3, B_2, B_1, and B_0.

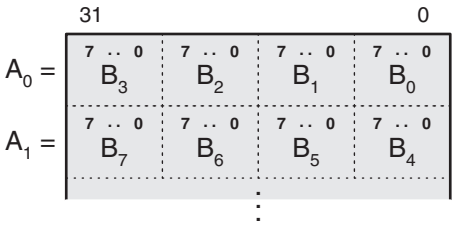

Figure 11: **Port Aspect Ratio Example Memory Map**

Virtex-5 and Virtex-4 Read-to-Write Aspect Ratios

When implementing RAMs targeting Virtex-5 and Virtex-4 FPGAs, the Block Memory Generator allows read and write aspect ratios. On each port, the read to write data width ratio can be 1:32, 1:16, 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1, 16:1, or 32:1.

For true dual-port RAMs, the four data buses, DINA, DOUTA, DINB, and DOUTB, can have different widths. For single-port RAMs, DINA and DOUTA widths can be independent. The maximum ratio between any two data buses is 32:1. The widest data bus can be no larger than 1152 bits.

If the read and write data widths are different, the memory depth is different with respect to read and write accesses. The depth ratio is the inverse of the width ratio. The address bus must be large enough to address the deeper of the two depths, since it is shared by read and write accesses. For the shallower, the least significant bits of the address bus are ignored. The smaller data words are arranged in little-endian format, as illustrated in Figure 12.

Virtex-5 and Virtex-4 Read-to-Write Aspect Ratio Example

Consider a true dual-port RAM of 64x512, which is the A port write width and depth. Table 2 lists the four data port widths and their respective depths for this example.

Table 2: **Read-to-Write Aspect Ratio Example Ports**

Interface	Data Width	Memory Depth
Port A Write	64	512
Port A Read	16	2048
Port B Write	256	128
Port B Read	32	1024

The ADDRA width is determined by the larger of the A port depths (2048). For this reason, ADDRA is 11 bits wide. On port A, read operations utilize the entire ADDRA bus, while write operations ignore the least significant 2 bits.

In the same way, the ADDRB width is determined by the larger of the B port depths (1024). For this reason, ADDRB is 10 bits wide. On port B, read operations utilize the entire ADDRB bus, while write operations ignore the least significant 3 bits.

The memory map in Figure 12 shows how port B write words are related to port A write words, in a little-endian arrangement. Note that AW_n is the write data word at address 'n' with respect to port A, while BW_n is the write data word at address 'n' with respect to port B.

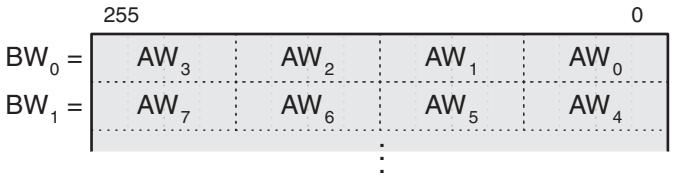

Figure 12: **Read-to-Write Aspect Ratio Example Memory Map**

BW_0 is made up of AW_3, AW_2, AW_1, and AW_0. In the same way, BR_0 is made up of AR_1 and AR_0, and AW_0 is made up of BR_1 and BR_0.

In the above example, the largest data width ratio is port B write words (256 bits) to port A read words (16 bits). This ratio is 16:1.

Aspect Ratio Limitations

In general, no port data width may be wider than 1152 bits, and no two data widths can have a ratio greater than 32:1. However, some optional features further limit data width aspect ratios:

- **Byte-writes:** When using byte-writes, no two data widths can have a ratio greater than 4:1.
- **Fixed primitive algorithm:** When using the fixed primitive algorithm with an N-bit wide primitive, aspect ratios are limited to 32:N and 1:N from the port A write width. For example, using the 4kx4 primitive type, the other ports may be no more than 8 times (32:4) larger than port A write width and no less than 4 times (1:4) smaller.

Virtex-5 and Virtex-4 Byte-Writes

The Block Memory Generator provides byte-write support in Virtex-5 and Virtex-4 devices. Byte-writes are available using either 8-bit or 9-bit byte sizes. When using an 8-bit byte size, no parity bits are used and the memory width is restricted to multiples of 8 bits. When using a 9-bit byte size, each byte includes a parity bit, and the memory width is restricted to multiples of 9 bits.

When byte-writes are enabled, the WE[A|B] bus is N bits wide, where N is the number of bytes in DIN[A|B]. The most significant bit in the write enable bus corresponds to the most significant byte in the input word. Bytes will be stored in memory only if the corresponding bit in the write enable bus is asserted during the write operation.

When 8-bit bytes are selected, the DIN and DOUT data buses are constructed from 8-bit bytes, with no parity. When 9-bit bytes are selected, the DIN and DOUT data buses are constructed from 9-bit bytes, with the 9th bit of each byte in the data word serving as a parity bit for that byte.

The byte-write feature may be used in conjunction with the Virtex-5 and Virtex-4 data width aspect ratios, which may limit the choice of data widths as described in Data Width Aspect Ratios. However, it may not be used with the NO_CHANGE operating mode. The byte-write feature also affects the operation of WRITE_FIRST mode, as described in Virtex-5 and Virtex-4 Write First Mode.

Byte-Write Example

Consider a single-port RAM with a data width of 24 bits, or 3 bytes with byte size of 8 bits. The write enable bus, WEA, consists of 3 bits. Figure 13 illustrates the use of byte-writes, and shows the contents of the RAM at address 0. Assume all memory locations are initialized to 0.

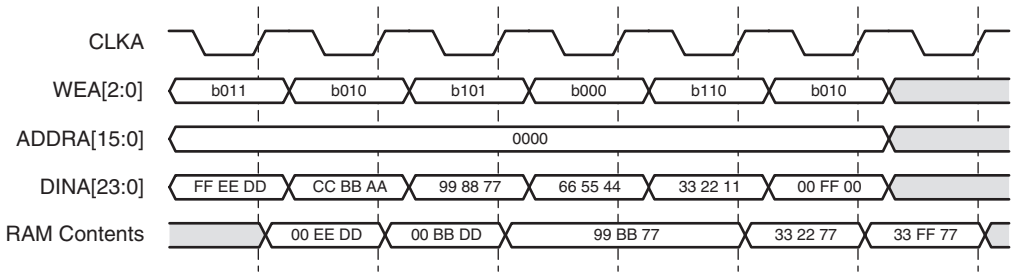

Figure 13: **Byte-write Example**

Virtex-5 and Virtex-4 Write First Mode

When performing a write operation in WRITE_FIRST mode, the concurrent read operation shows the newly written data on the output of the core. However, when using the byte-write feature or a non-symmetric aspect ratio, the output of the memory can not be guaranteed. For details of the hardware limitations of the architecture, see the Virtex-4 or Virtex-5 user guides.

Collision Behavior

The Block Memory Generator core supports dual-port RAM implementations. Each port is equivalent and independent, yet they access the same memory space. In such an arrangement, is it possible to have data collisions. The ramifications of this behavior are described for both asynchronous and synchronous clocks below.

Collisions and Asynchronous Clocks

Using asynchronous clocks, when one port writes data to a memory location, the other port must not read or write that location for a specified amount of time. This clock-to-clock setup time is defined in the device data sheet, along with other block RAM switching characteristics.

Collisions and Synchronous Clocks

Synchronous clocks cause a number of special case collision scenarios, described below.

Synchronous Write-Write Collisions

A write-write collision occurs if both ports attempt to write to the same location in memory. The resulting contents of the memory location are unknown. Note that write-write collisions affect memory content, as opposed to write-read collisions which only affect data output.

Using Byte-Writes

When using byte-writes, memory contents are not corrupted when separate bytes are written in the same data word. RAM contents are corrupted only when both ports attempt to write the same byte. Figure 14 illustrates this case. Assume ADDRA = ADDRB = 0.

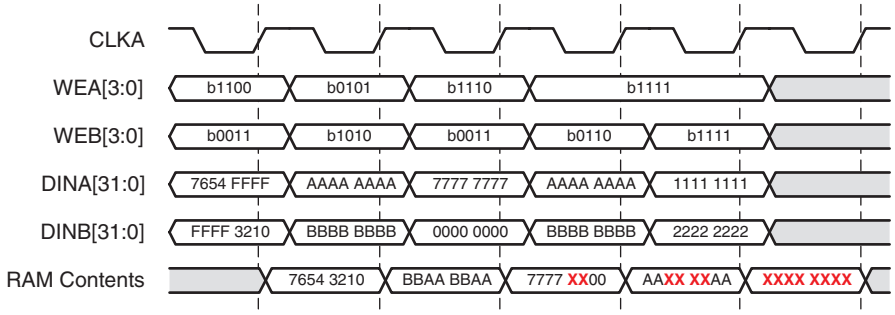

Figure 14: **Write-Write Collision Example**

Synchronous Write-Read Collisions

A synchronous write-read collision may occur if a port attempts to write a memory location and the other port reads the same location. While memory contents are not corrupted in write-read collisions, the validity of the output data depends on the write port operating mode.

- If the write port is in READ_FIRST mode, the other port can reliably read the old memory contents.
- If the write port is in WRITE_FIRST or NO_CHANGE mode, data on the output of the read port is invalid.
- In the case of byte-writes, only bytes which are updated will be invalid on the read port output.

Figure 15 illustrates write-read collisions and the effects of byte-writes. DOUTB is shown for when port A is in WRITE_FIRST mode and READ_FIRST mode. Assume ADDRA = ADDRB = 0, port B is always reading, and all memory locations are initialized to 0. The RAM contents are never corrupted in write-read collisions.

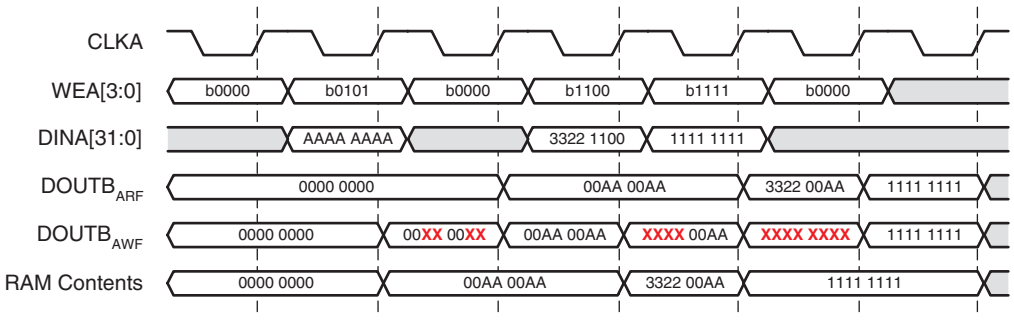

Figure 15: **Write-read Collision Example**

Optional Output Registers

The Block Memory Generator allows optional output registers to improve the performance of the core. The user may choose to include register stages at two places: at the output of the block RAM primitives and at the output of the core.

Registers at the output of the block RAM primitives reduce the impact of the clock to out delay of the primitives. Registers at the output of the core isolate the delay through the output multiplexers, improving the clock-to-out delay of the Block Memory Generator core. Note that each optional register stage used adds an additional clock cycle of latency to the read operation.

Figure 16 shows a memory configuration with registers at the output of the memory primitives and at the output of the core.

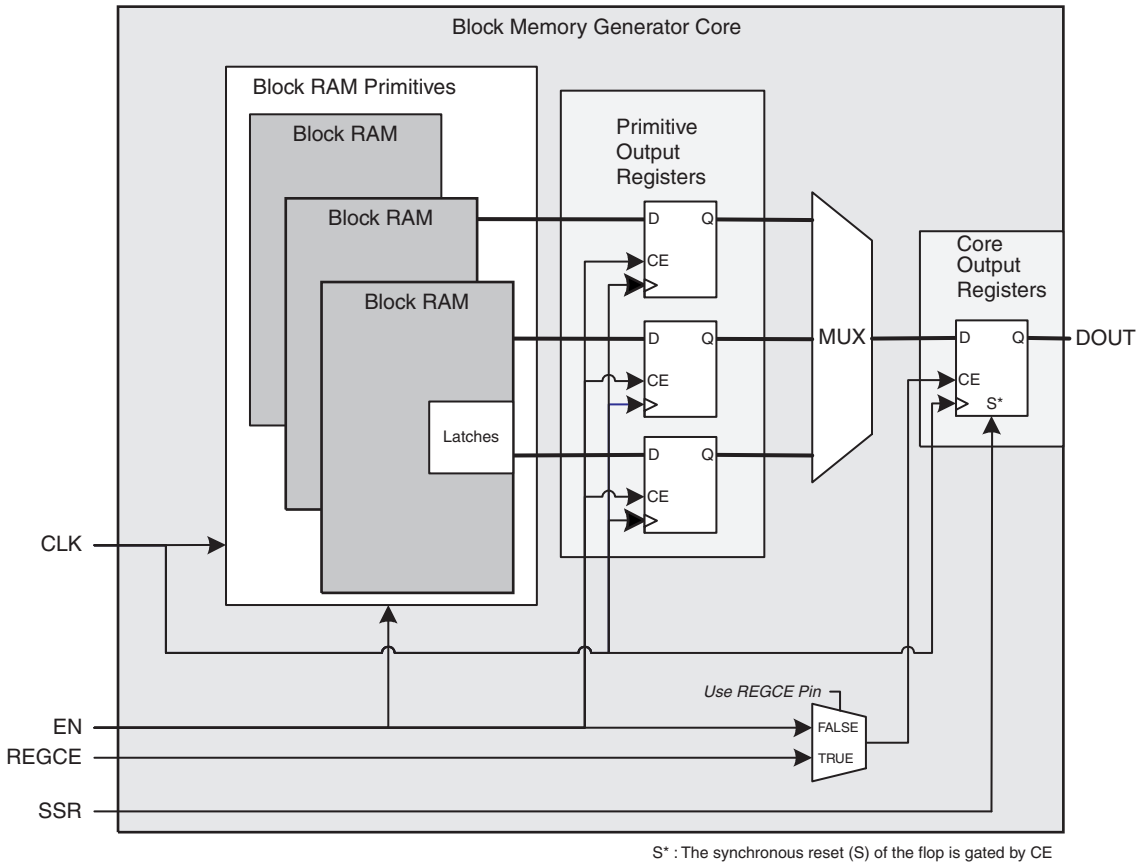

Figure 16: **Virtex-II Block Memory: Register Outputs of Memory Primitives and Memory Core Options Enabled**

For Virtex-5, Virtex-4, and Spartan-3A DSP FPGAs, the Register Output of Memory Primitives option may be implemented using the embedded block RAM registers, requiring no further FPGA resources. All other register stages are implemented in FPGA fabric. Figure 17 shows an example of a Virtex-4 or Virtex-5-based memory that has been configured using both output register stages.

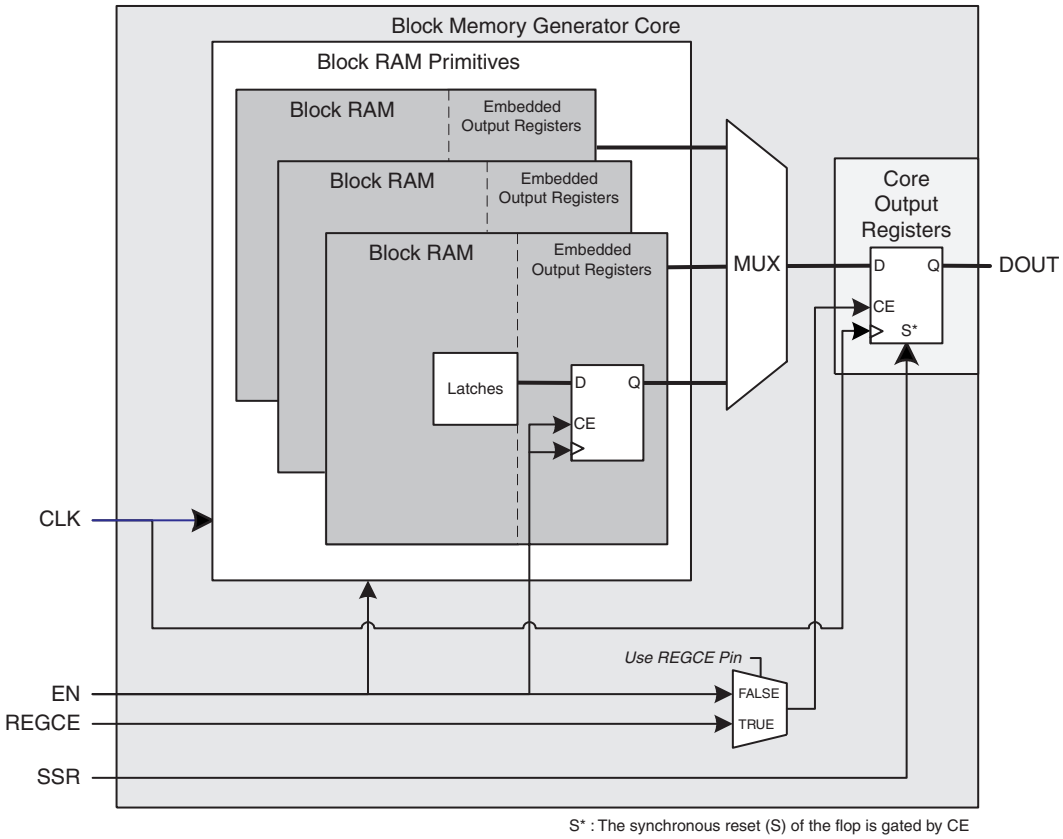

Figure 17: **Virtex-5 and Virtex-4 Block Memory with Register Output of Memory Primitives and Register Output of Memory Core Options Enabled**

When using the Synchronous Reset Input (SSR), the behavior of the embedded output registers in Spartan-3A DSP differs slightly from the configuration shown in Figure 17. By default, the Block Memory Generator builds the memory output register in the FPGA fabric to maintain compatibility between core configurations. To force the core to use the embedded output registers in Spartan-3A DSP, the RAMB16BWER Reset Behavior option is provided.

For a complete description of the supported output options, see "Output Register Configurations" on page 40.

Optional Register Clock Enable Pins

The optional output registers are enabled by the EN signal by default. However, when the Use REGCEA/REGCEB Pin option is selected, the output register stage will be controlled by the REGCEA/REGCEB pins instead.

Thus, the data output from the core can be controlled independent of the flow of data through the rest of the core. When using the `REGCE` pin, the last output register operates independently of the `EN` signal.

Optional Synchronous Set/Reset Pins

The synchronous set/reset pins (`SSRA` and `SSRB`) control the reset operation of the last register in the output stage. For memories with no output registers, the reset pins control the memory output latches.

When SSR and REGCE are asserted on a given port, the data on the output of that port is driven to the reset value defined in the CORE Generator GUI. (The reset occurs on SSR and EN when the Use REGCE Pin option is not selected.)

For Virtex-4 FPGAs, if the option to use the synchronous set/reset pin is selected in conjunction with memory primitive registers and without core output registers, then the Virtex-4 embedded block RAM registers may not be utilized.

For Spartan-3A DSP FPGAs, the synchronous set/reset behavior may differ when the RAMB16BWER reset behavior option is selected. However, this option saves resources by using the embedded output registers available in Spartan-3A DSP RAMB16BWER primitives. See "Output Register Configurations" on page 40 for more information.

Memory Output Flow Control Examples

The combination of the enable (EN), reset (SSR), and register enable (REGCE) pins allow a wide range of data flows in the output stage. Figure 18 and Figure 19 are examples on how this can be accomplished. Keep in mind that the SSR and REGCE pins apply only to the last register stage.

Figure 18 depicts how SSR can be used to control the data output to allow only intended data through. Assume that both output registers are used, the port A reset value is 0xFFFF, and that EN and REGCE are always asserted. The data on the block RAM memory latch is labeled LATCH, while the output of the block RAM embedded register is labeled REG1. The output of the last register is the output of the core, DOUT.

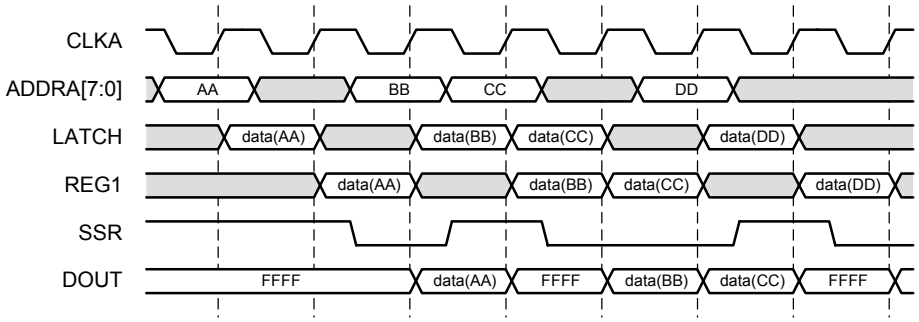

Figure 18: **Flow Control Using SSR**

Figure 19 depicts how REGCE can be used to latch the data output to allow only intended data through. Assume that only the memory primitive registers are used, and that EN is always asserted and SSR is

always deasserted. The data on the block RAM memory latch is labeled latch, while the output of the last register, the block RAM embedded register, is the core output, DOUT.

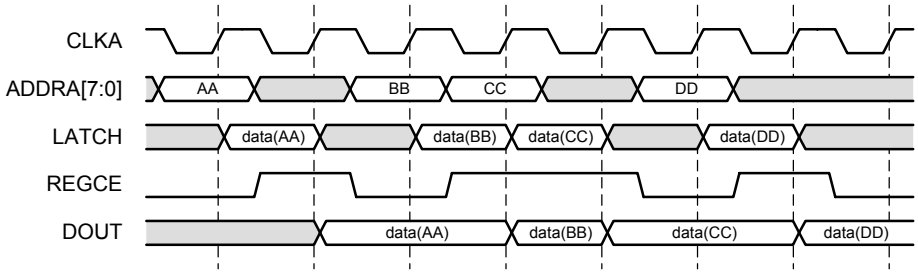

Figure 19: **Flow Control Using REGCE**

ECC – Built-in Hamming Error Correction

In Virtex-5, the Block Memory Generator core supports ECC, the built-in Hamming Error Correction that is built into the block RAM primitives on this architecture. Each write operation generates 8 protection bits for every 64 bits of data, which are stored with the data in memory. These bits are used during each read operation to correct any single bit error, or to detect (but not correct) any double bit error.

This operation is transparent to the user. Two status outputs (SBITERR and DBITERR) indicate the three possible read results: no error, single error corrected, and double error detected. For single-bit errors, by default the read operation does not correct the error in the memory array; it only presents corrected data on DOUT.

ECC is only available when the following options are chosen:

- Virtex-5
- Single Port RAM or Simple Dual Port RAM memory type

When using ECC, the Block Memory Generator constructs the memory from special primitives available in Virtex-5 architectures. The ECC memory block is 512x64, and is composed of two 18k block RAMs combined with dedicated ECC encoding and decoding hardware. The 512x64 primitives are used to build memory sufficient for the desired user memory space.

When using ECC, certain other core options are limited as follows:

- Byte-write enable is not available.
- All port widths must be identical.
- Only Read First Operating Mode is supported.
- Synchronous Reset (SSR) and the Output Reset Value options are not available.
- Memory Initialization is not supported.
- No Algorithm selection is available.

The following waveform shows a typical write and read operation for a Virtex-4 Block Memory Generator core in Simple Dual Port RAM with ECC enabled, and no additional output registers.

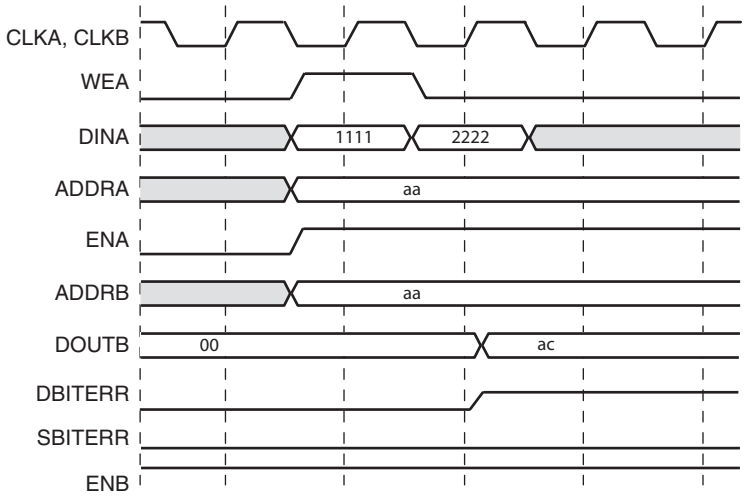

Figure 20: **Read and Write Operation with ECC in Virtex-5**

.

The Block Memory Generator core does not support the insertion of errors for correction by ECC in simulation. For this reason, the simulated functionality of ECC is identical to non-ECC behavior with the SBITERR and DBITERR outputs always equal to 0.

Simulation Models

The Block Memory Generator core provides two types of functional simulation models to its users:

- Behavioral Simulation Models (VHDL and Verilog)
- Structural/Unisim based Simulation Models (VHDL and Verilog)

The behavioral simulation models provide a simplified model of the core while the structural (UniSim) based simulation models are an accurate modeling of the internals of the core. The behavioral simulation models are written purely in RTL and simulates faster than the structural based simulation models and are ideal for debugging. Moreover, the memory is modeled in a two-dimensional array, making it easier to probe contents of the memory.

The structural simulation model uses primitive instantiations to more accurately model the behavior of the core. Collision detection and 'X' generation are better modeled using the structural simulation model. However, simulation time is longer and debug may be more difficult.

The “Simulation Files” options in the CORE Generator Project Options determine which type of functional simulation models are generated.

Table 3 lists the differences between the two functional simulation models:

Table 3: **Differences Between Simulation Models**

	Behavioral Models	Structural (Unisim) based Models
When core output is undefined	Will never generate 'X'	Will generate 'X' to match core
Out-of-range address access	Optionally flags a warning message	Will generate 'X'
Collision behavior	Will not generate 'X' on output and flag a warning message	Will generate 'X' to match core
Byte-write collision behavior	Will flag all byte-write collisions	Will not flag collisions if byte-writes do not overlap

Signal List

Table 4 provides a description of the Block Memory Generator core signals. The widths of the data ports (DINA, DOUTA, DINB, and DOUTB) are selected by the user in the CORE Generator GUI. The address port (ADDRA and ADDRB) widths are determined by the memory depth with respect to each port, as selected by the user in the GUI. The write enable ports (WEA and WEB) are busses of width 1 when byte-writes are disabled. When byte-writes are enabled, WEA and WEB widths depend on the byte size and write data widths selected in the GUI.

Table 4: **Core Signal Pinout**

Name	Direction	Description
CLKA	Input	**Port A Clock:** Port A operations are synchronous to this clock. For synchronous operation, this must be driven by the same signal as CLKB.
ADDRA	Input	**Port A Address:** Addresses the memory space for Port A read and write operations. Available in all configurations.
DINA	Input	**Port A Data Input:** Data input to be written into the memory via Port A. Available in all RAM configurations.
DOUTA	Output	**Port A Data Output:** Data output from read operations via Port A. Available in all configurations except simple dual-port RAM.
ENA	Input	**Port A Clock Enable:** Enables read, write, and reset operations via Port A. Optional in all configurations.
WEA	Input	**Port A Write Enable:** Enables write operations via Port A. Available in all RAM configurations.
SSRA	Input	**Port A Synchronous Set/Reset:** Resets the Port A memory output latch or output register. Optional in all configurations.
REGCEA	Input	**Port A Register Enable:** Enables the last output register. Optional in all configurations with output registers.
CLKB	Input	**Port B Clock:** Port B operations are synchronous to this clock. Available in dual-port configurations. For synchronous operation, this must be driven by the same signal as CLKA.
ADDRB	Input	**Port B address:** Addresses the memory space for Port B read and write operations. Available in dual-port configurations.
DINB	Input	**Port B Data Input:** Data input to be written into the memory via Port B. Available in true dual-port RAM configurations.

Table 4: **Core Signal Pinout *(Continued)***

Name	Direction	Description
DOUTB	Output	**Port B Data Output:** Data output from read operations via Port B. Available in dual-port configurations.
ENB	Input	**Port B Clock Enable:** Enables read, write, and reset operations via Port B. Optional in dual-port configurations.
WEB	Input	**Port B Write Enable:** Enables write operations via Port B. Available in dual-port RAM configurations.
SSRB	Input	**Port B Synchronous Set/Reset:** Resets the Port B memory output latch or output register. Optional in dual-port configurations.
REGCEB	Input	**Port B Register Enable:** Enables the last output register. Optional in dual-port configurations with output registers.
SBITERR	Output	**SIngle Bit Error**: Flags the presence of a single-bit error in memory which has been auto-corrected on the output bus.
DBITERR	Output	**Double Bit Error**: Flags the presence of a double-bit error in memory. Double-bit errors cannot be auto-corrected by the built-in ECC decode module.

Generating the Core

The Block Memory Generator can be found in the CORE Generator GUI by selecting:

- View by Function → Memories & Storage Elements → RAMs & ROMs.

This section describes the options available in the Block Memory Generator GUI.

CORE Generator Parameter Screens

The Block Memory Generator GUI includes five main screens:

- Block Memory Generator Main Screen
- Port A Options Screen
- Port B Options Screen
- Output Registers and Memory Initialization Screen
- Simulation Model Options and Information Screen

In addition, all the screens share common tabs and buttons to provide information about the core and to navigate the Block Memory Generator GUI.

Block Memory Generator Main Screen

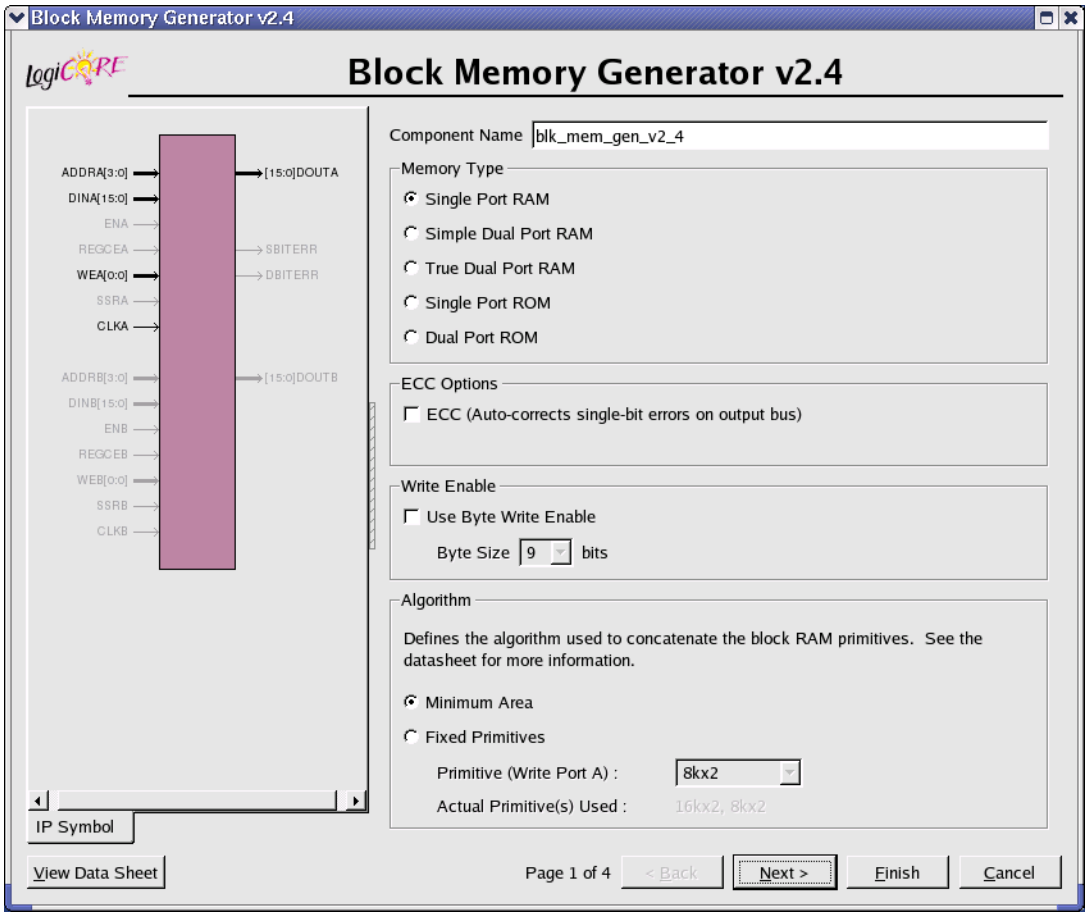

Figure 21: **Block Memory Generator Main Screen**

Component Name

The base name of the output files generated for the core. Names must begin with a letter and be composed of any of the following characters: a to z, 0 to 9, and "_". Names can not be Verilog or VHDL reserved words.

Memory Type

Select the type of memory to be generated.

- Single-port RAM
- Simple dual-port RAM
- True dual-port RAM
- Single-port ROM
- Dual-port ROM configurations

ECC Options

When targeting Virtex-5, and when either single port RAM or simple dual port RAM memory type is selected, the ECC options become available. Selecting ECC enables the built-in Hamming error correction in the Virtex-5 and architecture.

When using ECC, certain other core options are limited, as listed below:

- Byte-write Enable is not available.
- All port widths must be identical.
- Only Read First Operating mode is supported.
- Synchronous Reset (SSR) and the Output Reset Value options are not available.
- Memory Initialization is not supported.
- No algorithm selection is available.

See ECC – Hamming Error Correction for more information.

Write Enable

When targeting Virtex-4 or Virtex-5, select whether to use the byte-write enable feature. Byte size is either 8-bits (no parity) or 9-bits (including parity). The data width of the memory will be multiples of the selected byte-size.

Algorithm

Select the algorithm used to implement the memory.

- **Minimum Area Algorithm:** Generates a core using the least number of primitives.
- **Fixed Primitive Algorithm:** Generates a core that concatenates a single primitive type to implement the memory. Choose which primitive type to use in the drop-down list.

Port A Options Screen

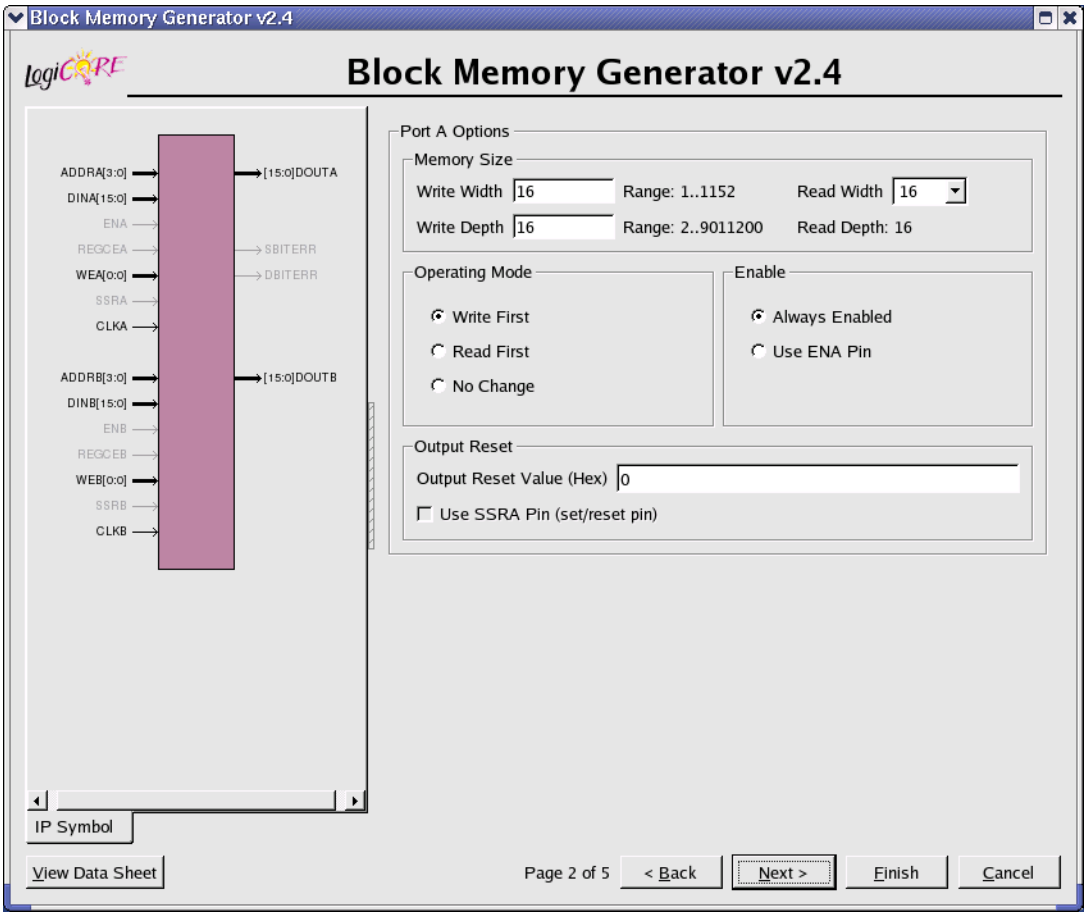

Figure 22: **Port A Options**

Memory Size (Port A)

Specify the port A write width and depth. Select the port A read width from the drop-down list of valid choices. The read depth is calculated automatically.

Operating Mode (Port A)

Specify the port A operating mode.

- READ_FIRST
- WRITE_FIRST
- NO_CHANGE

Enable (Port A)

Select the enable type:

- Always enabled (no ENA pin available)
- Use ENA pin

Output Reset (Port A)

Specify the reset value of the memory output latch and output registers. These values are with respect to the read port widths. Choose whether a set/reset pin (SSRA) is needed.

Port B Options Screen

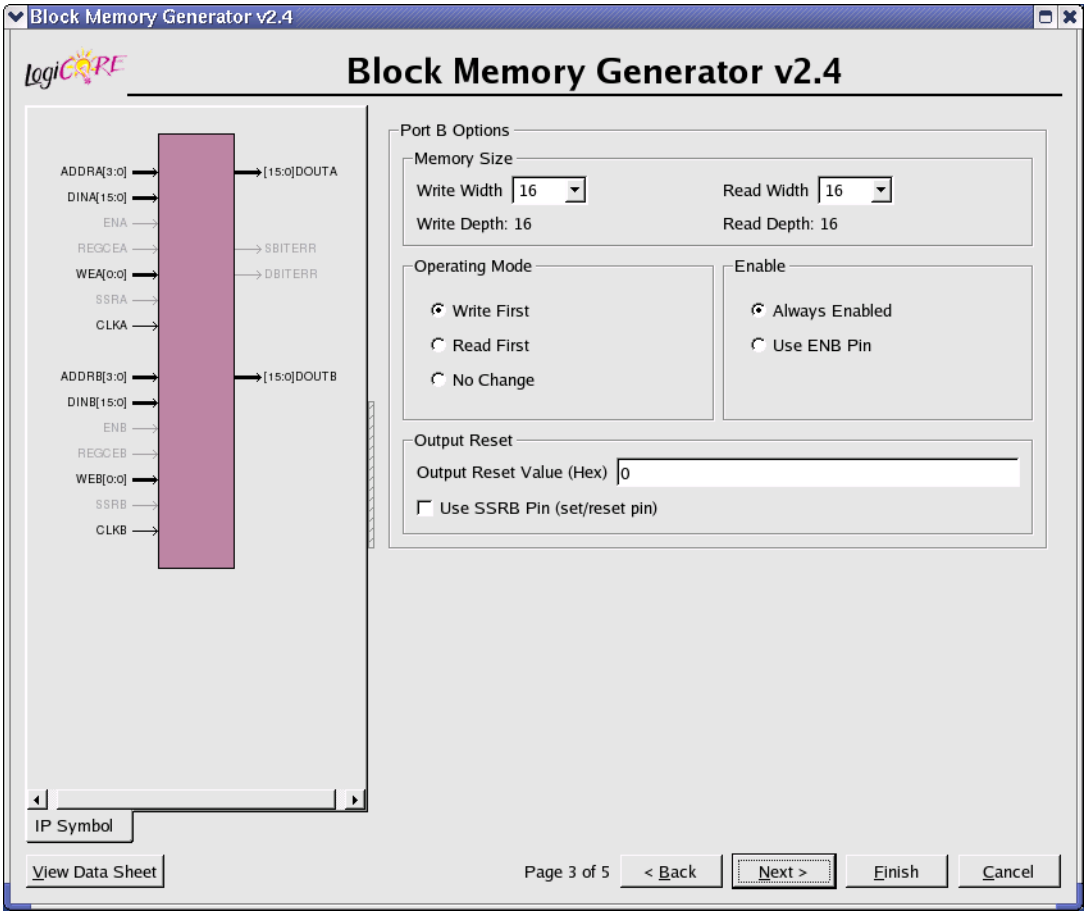

Figure 23: **Port B Options**

Memory Size (Port B)

Select the port B write and read widths from the drop-down list of valid choices. The read depth is calculated automatically.

Operating Mode (Port B)

Specify the port B write mode.

- READ_FIRST
- WRITE_FIRST
- NO_CHANGE

Enable (Port B)

Select the enable type:

- Always enabled (no ENB pin available)

- Use ENB pin

Output Reset (Port B)

Specify the reset value of the memory output latch and output registers. These values are with respect to the read port widths. Choose whether a set/reset pin (SSRB) is needed.

Output Registers and Memory Initialization Screen

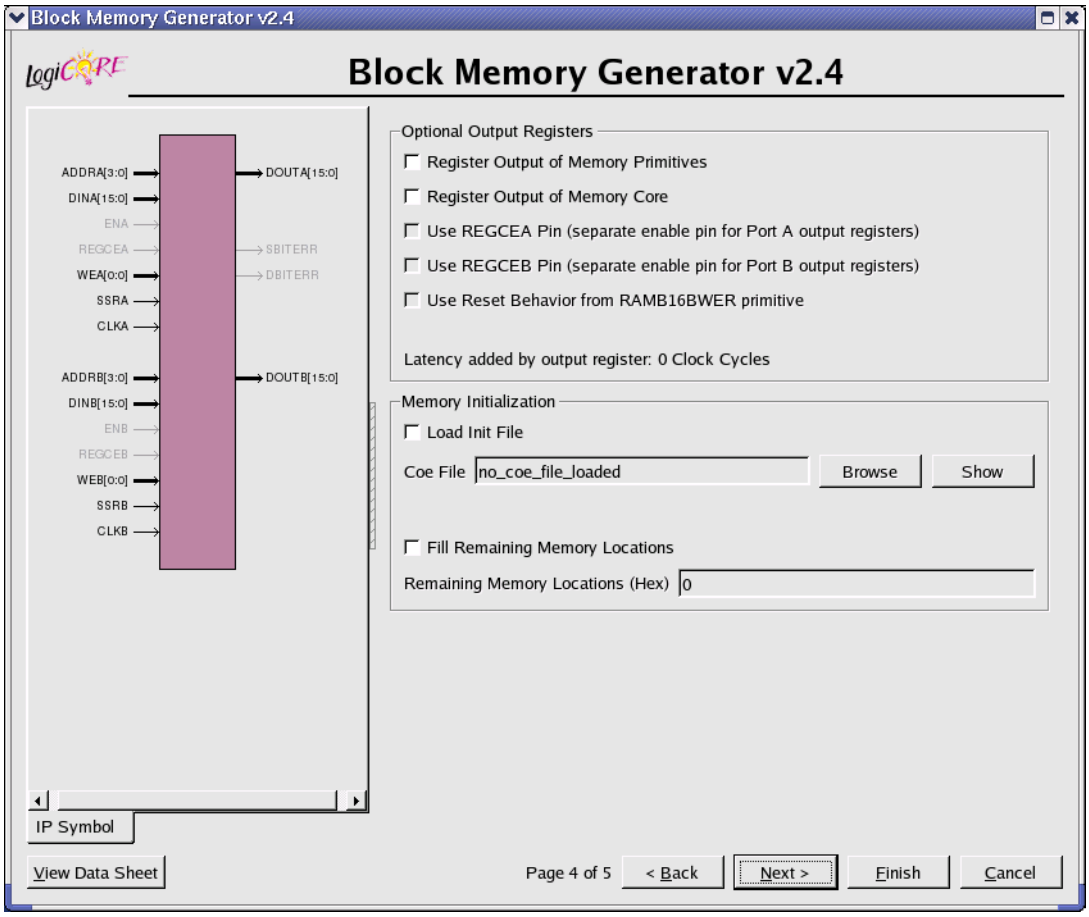

Figure 24: **Output Registers and Memory Initialization Screen**

Optional Output Registers

Select which output register stages to include:

- **Register Output of Memory Primitives.** Select to insert output register after the memory primitives. When targeting Virtex-4 or Virtex-5 FPGAs, the embedded output registers in the block RAM primitives will be used if the user chooses to register the output of the memory primitives. For other architectures, the registers in the FPGA slices will be used. Note that in Virtex-4, the use of the SSR input will prevent the core from using the embedded output registers. See "Output Register Configurations" on page 40 for more information.
- **Register Output of Memory Core.** Select to insert register output of the memory core. The registers in the FPGA slices will be used.

  Select whether to have separate enable pins for the last register stage if output registers are selected.

- **Use Reset Behavior from RAMB16BWER Primitive**. Selecting this option causes the Block Memory Generator to use the embedded output registers in the Spartan-3A DSP RAMB16BWER primitives; however, it also changes the behavior of the core during reset. See "Output Register Configurations" on page 40 for more information.

Memory Initialization

Select whether to initialize the memory contents using a COE file, and whether to initialize the remaining memory contents with a default value. When using asymmetric port widths or data widths, the COE file and the default value are with respect to the port A write width.

Simulation Model Options and Information Screen

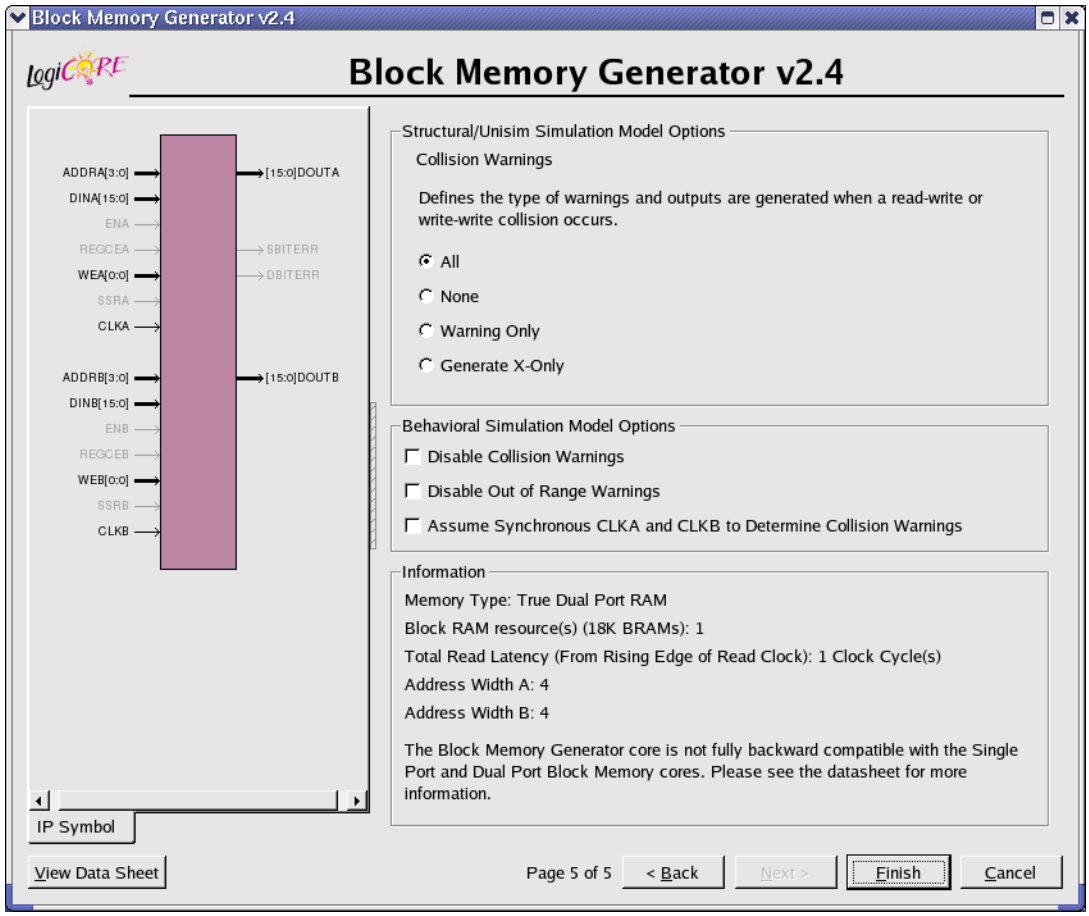

Figure 25: **Simulation Model Options and Information Screen**

Structural/UNISIM Simulation Model Options

Select the type of warning messages and outputs generated by the structural simulation model in the event of collisions.

Behavioral Simulation Model Options

Select the type of warning messages generated by the behavioral simulation model. Select whether the model should assume synchronous clocks for collision warnings.

Information Panel

This pane displays an informational summary of the selected core options.

- **Memory Type**: Reports the selected memory type.
- **Block RAM Resources (18k BRAMs)**: Reports the exact number of 18k block RAM primitives which will be used to construct the core. For Virtex-5, each 36k block RAM primitive is counted as two 18k block RAMs.
- **Total Read Latency**: The number of clock cycles for a read operation. This value is controlled by the optional output registers options on the previous screen.
- **Address Width**: The actual width of the address bus to each Port.

Specifying Initial Memory Contents

The Block Memory Generator core supports memory initialization using a memory coefficient (COE) file, the default data option in the CORE Generator GUI, or a combination of both.

The COE file can specify the initial contents of each memory location, while default data specifies the contents of all memory locations. When used in tandem, the COE file can specify a portion of the memory space, while default data fills the rest of the remaining memory space. COE files and default data is formatted with respect to the port A write width (or port A read width for ROMs).

A COE is a text file which specifies two parameters:

- **memory_initialization_radix:** The radix of the values in the memory_initialization_vector. Valid choices are 2, 10, or 16.
- **memory_initialization_vector:** Defines the contents of each memory element. Each value is LSB-justified, separated by a space, and assumed to be in the radix defined by memory_initialization_radix.

The following is an example COE file. Note that semi-colon is the end of line character.

```
; Sample initialization file for a
; 32-bit wide by 16 deep RAM
memory_initialization_radix = 16;
memory_initialization_vector =
0 1 2 3 4 5 6 7
8 9 A B C D E F;
```

Verification

The Block Memory Generator core and the simulation models delivered with it are rigorously verified using advanced verification techniques, including a constrained-random configuration generator and a cycle-accurate bus functional model.

Resource Utilization and Performance

The resource utilization and performance of the Block Memory Generator core is highly dependent on user selections, such as algorithm, optional output registers, and memory size.

Block RAM Usage

The Information panel (screen 5 of the Block Memory Generator GUI) reports the actual number of block RAM blocks to be used. This number is always given as the number of 18k block RAM primitives used.

To estimate this value when using the fixed primitive algorithm, the number of block RAM primitives used is equal to the width ratio (rounded up) multiplied by the depth ratio (rounded up), where the width ratio is the width of the memory divided by the width of the selected primitive, and the depth ratio is the depth of the memory divided by the depth of the primitive selected.

To estimate this value when using the minimum area algorithm, it is not as easy to determine the exact block RAM count. This is because the actual algorithm performs complex manipulations to produce an optimal solution. The optimistic estimate is total memory bits divided by 18k (the total number of bits per primitive) rounded up. Given that the algorithm packs block RAMs very efficiently, this estimate is often very accurate for most memories.

LUT Utilization and Performance

The LUT utilization and performance of the core are directly related to the arrangement of primitives and the selection of output registers. Particularly, the number of primitives cascaded in depth to implement a memory determines the size of the output multiplexer and the size of the input decoder, which are implemented in the FPGA fabric.

It should be noted that while the primary goal of the minimum area algorithm is to use the minimum number of block RAM primitives, it has a secondary goal of maximizing performance – as long as block RAM usage does not increase.

Resource Utilization and Performance Examples

The following tables provide examples of actual resource utilization and performance for Block Memory Generator implementations. Each section highlights the effects of a specific feature on resource utilization and performance.

Benchmarks were performed targeting a Virtex-II Pro FPGA in the -5 speed grade (2vp30-ff1152-5), and a Virtex-4 FPGA in the -10 speed grade (4vlx60-ff1148-10), and a Virtex-5 FPGA in the -1 speed grade (5vlx30-ff324-1). Better performance may be possible with higher speed grades.

In the benchmark designs described below, the core was encased in a wrapper with input and output registers to remove the effects of IO delays from the results; performance may vary depending on the user design. The minimum area algorithm was used unless otherwise noted. The examples below highlight the use of embedded registers in Virtex-5 and Virtex-4, and the subsequent significant performance improvement.

Single Primitive

The Block Memory Generator does not add additional logic if the memory can be implemented in a single block RAM primitive. Table 5 shows performance data for single-primitive memories.

Table 5: **Single Primitive Examples**

Memory Type	Options	Width x Depth	Resource Utilization			Performance		
			Block RAMs	LUTs[1]	FFs	Virtex-II Pro	Virtex-4	Virtex-5

Table 5: **Single Primitive Examples**

True Dual-Port RAM		36x512	1	0	0	330	310	300
		9x2k	1	0	0	345	335	300
	Virtex-4 or Virtex-5 Embedded Output Registers	36x512	1	0	0	N/A	385	395
		9x2k	1	0	0	N/A	395	455

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

Output Registers

The Block Memory Generator optional output registers increase the performance of memories by isolating the block RAM primitive clock-to-out delays and the data output multiplexer delays.

The output registers are only implemented for output ports. For this reason, when output registers are used, a single port RAM requires fewer resources than a true dual port RAM. Note that the effects of the core output registers are not fully illustrated due to the simple register wrapper used. In a full-scale user design, core output registers may improve performance notably.

Table 6: **Virtex-II Pro Output Register Examples**

Memory Type	Width x Depth	Output Register Option	Virtex-II Pro			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
True Dual-Port RAM	17x5k		5	0	62	215
		Primitive	5	92	62	280
		Core	5	34	42	270
		Primitive, Core	5	132	42	275
Single-Port RAM	17x5k		5	0	31	225
		Primitive	5	46	31	280
		Core	5	17	21	275
		Primitive, Core	5	66	21	280

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

In Virtex-4 architectures, the embedded block RAM may be utilized, reducing the FPGA fabric resources required to create the registers.

Table 7: **Virtex-4 Output Register Examples**

Memory Type	Width x Depth	Output Register Option	Virtex-4			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
True Dual-Port RAM	17x5k		5	0	62	285
		Primitive	5	6	62	375
		Core	5	34	42	260
		Primitive, Core	5	46	42	370
Single-Port RAM	17x5k		5	0	35	285
		Primitive	5	3	35	395
		Core	5	17	25	260
		Primitive, Core	5	23	25	385

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

In Virtex-5 architectures, the embedded block RAM may be used to reduce the FPGA fabric resources required to create the registers.

Table 8: **Virtex-5 Output Register Examples**

Memory Type	Width x Depth	Output Register Options	Virtex-5			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
True Dual-Port RAM	17x5k		3	0	38	280
		Primitive	3	6	44	435
		Core	3	34	38	295
		Primitive, Core	3	40	44	455
Single-Port RAM	17x5k		3	0	19	275
		Primitive	3	3	22	455
		Core	3	17	19	305
		Primitive, Core	3	20	22	450

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

Aspect Ratios

The Block Memory Generator selectable port and data width aspect ratios may increase block RAM usage and affect performance, because aspect ratios limit the primitive types available to the algorithm, which can reduce packing efficiency. Large aspect ratios, such as 1:32, have a greater impact than small aspect ratios. Note that width and depth are reported with respect to the port A write interface.

Table 9: **Virtex-II Pro Aspect Ratio**

Memory Type	Width x Depth	Port Aspect Ratio	Virtex-II Pro			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
True Dual-Port RAM	17x5k	1:1	5	0	62	215
	136x640	1:8[2]	9	0	0	275

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.
2. A port is 136x640; B port is 17x5k.

Table 10: **Virtex-4 Aspect Ratio**

Memory Type	Width x Depth	Data Width Aspect Ratio	Virtex-4			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
Single-Port RAM	17x5k	1:1	5	0	35	285
		1:8[2]	9	0	0	310
	136x640	8:1[3]	9	0	0	350

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.
2. Read port is 136x640; write port is 12x5k.
3. Read port is 17x5k, write port is 136x640.

Table 11: **Virtex-5 Aspect Ratio**

Memory Type	Width x Depth	Aspect Ratio	Virtex-5			
			Block RAMs	FFs	LUTs[1]	Performance (MHz)
Single-Port RAM	17x5k	1:1	3	0	19	300
		1:8[2]	5	0	TBD	295
	136x640	8:1[3]	5	0	TBD	305[4]

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.
2. Read port is 136x640; write port is 17x5k.
3. Read port is 17x5k, write port is 136x640.
4. This configuration was generated using the xc5vlx50-ff676-1 part.

Algorithm

The differences between the minimum area algorithm and the fixed primitive algorithm are discussed in detail in "Selectable Memory Algorithm" on page 2. Table 12 shows examples of the resource utilization and the performance difference between them for two selected configurations for Virtex-II Pro and Virtex-4 architectures.

Table 12: **Memory Algorithm Examples Virtex-II Pro and Virtex-4**

Memory Type	Width x Depth	Algorithm Type	Resource Utilization			Performance (MHz)	
			Block RAM	FFs	LUTs[1]	Virtex-II Pro	Virtex-4
Single-Port RAM	17x5k	Minimum area	5	0	31	220	285
		Fixed Primitive using 18x1k block RAM	5	0	57	245	235
	36x4k	Minimum area	8	0	38	295	290
		Fixed Primitive using 36x512 block RAM	8	0	152	225	245

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

Table 13 shows examples of the resource utilization and the performance difference between them for two selected configurations for Virtex-5 architecture.

Table 13: **Memory Algorithm Examples Virtex-5**

Memory Type	Width x Depth	Algorithm Type	Resource Utilization			Performance (MHz)
			Block RAM	FFs	LUTs[1]	Virtex-5
Single-Port RAM	17x5k	Minimum area	3	0	19	300
		Fixed Primitive using 18x1k block RAM	3	0	20	285
	36x4k	Minimum area	4	0	0	300
		Fixed Primitive using 36x512 block RAM	4	0	40	280

1. LUTs are reported as the number of 4-input LUTs, and do not reflect the number of LUTs used as a route-through.

Supplemental Information

The following sections provide additional information about working with the Block Memory Generator core.

- Interfacing with Older Memory Cores. Defines the differences between older memory cores and the Block Memory Generator core.
- SIM Parameters. Defines the SIM parameters used to specify the core configuration.
- Output Register Configurations. Provides information optional output registers used to improve core performance.

Interfacing with Older Memory Cores

This section contains valuable information for those customers who are using a previous version of the Block Memory Generator core.

Using the Block Memory Generator Migration Kit

The Block Memory Generator LogiCORE replaces the following two legacy CORE Generator LogiCOREs:

- Dual Port Block Memory (v6.x)
- Single Port Block Memory (v6.x)

Cores generated by the Block Memory Generator are not drop-in replacements for the v6.x cores listed above as there are a number of feature and interface changes in the Block Memory Generator.

To help you migrate designs containing legacy v6.x Dual and Single Port Block Memory LogiCOREs to the new Block Memory Generator core, Xilinx provides the Block Memory Generator Migration Kit to take care of all your migration-related tasks without the need to manually edit your design. For more information about using the Migration Kit, see the Block Memory Generator Migration Kit Product Page.

Differences Between Cores

This section describes the differences between the LogiCORE Single and Dual Port Block Memory cores and the Block Memory Generator. The new Block Memory Generator is not backward compatible with the Single and Dual Port Block Memory core in several aspects.

- Single Port Block Memory core and the Block Memory Generator do not have the same port names for a single-port memory configuration.
- Dual Port Block Memory core and the Block Memory Generator do not have the same reset pin name.
- XCO files for the previous memory cores are NOT compatible with the Block Memory Generator.
- Pin polarity, handshaking, and input register features supported by the previous memory cores are not supported in the Block Memory Generator.
- The behavior of set/reset pin (`SSR`) and the enable pin (`EN`) in the Block Memory Generator will differ from previous cores when using optional output registers.
- The Dual-port Block Memory core supports all combinations of read-only, write-only, and read/write ports for A & B ports. Some of these combination are only available in the Block Memory Generator by manually tying-off unused ports of a True Dual-port RAM.

Note: The Block Memory Migration Kit automatically ties-off the appropriate ports when used to migrate from a Dual-port Block Memory core.

Port Memory Pin Names

In the Block Memory Generator, the port names have been changed from the older Single Port Block Memory and Dual Port Block Memory cores to reflect the actual ports on the block RAM primitives. Table 14 and Table 15 reflect the changes in port names.

Table 14: **Port Name Changes from Single Port Block Memory Core**

Single Port Block Memory v6.2 (Old core)	Block Memory Generator v2.1 (New Core, Single Port Configuration)
DIN	DINA
ADDR	ADDRA
EN	ENA
WE	WEA
SINIT	SSRA
CLK	CLKA
DOUT	DOUTA
ND	Not supported
RFD	Not supported
RDY	Not supported

Table 15: **Port Name Changes from Dual Port Block Memory Core**

Dual Port Block Memory v6.2 (Old core)	Block Memory Generator v2.1 (New Core, Single Port Configuration)
SINITA	SSRA
SINITB	SSRB
ND [A|B]	Not supported
RFD [A|B]	Not supported
RDY [A|B]	Not supported

Obsolete Features

Pin Polarity Option

The Block Memory Generator does not support pin polarity options on the clock, enable, write enable and set/reset input pins. When active low signaling is desired, users can invert the signals prior to the input of the core.

Handshaking Pins

The Block Memory Generator does not support handshaking pins: ND[A|B], RFD[A|B] and RDY[A|B].

Input Registers

The Block Memory Generator does not support input registers on the DIN, ADDR, and WE input ports.

Read/Write Ports

The Block Memory Generator does not support all combinations of read-only, write-only, and read/write ports for ports A & B. Use a True-dual Port RAM Memory type and manually tie-off unused ports to achieve configurations with only one write-only or read-only port.

Note: The Block Memory Generator Migration Kit automatically ties-off appropriate ports when used to migrate from a Dual-port Block Memory core.

Modified Behaviors

Enable Pin

Table 16 illustrates the difference in enable behavior between previous cores and the Block Memory Generator.

Table 16: **Old and New Core Differences**

Single (or Dual) Port Block Memory v6.2	Block Memory Generator v2.1
Single enable pin that controls the enables of all registers and memories	Two optional enable pins provided: • REGCE—optionally controls the enables of the registers in the last output stage • EN—controls the enables of all other registers and memory

Synchronous Reset Pin

In the previous memory cores, the synchronous reset (SINIT pin) initializes all memory latches and output registers. In the Block Memory Generator, the synchronous set/reset (SSR pin) only resets the registers in the last output stage. When there are no output register stages present, it will initialize the memory latches.

SIM Parameters

Table 17 defines the SIM parameters used to specify the configuration of the core. These parameters are only used to manually instantiate the core in HDL, calling the CORE Generator dynamically. This parameter list does not apply to users that generate the core using the CORE Generator GUI.

Table 17: **SIM Parameters**

	SIM Parameter	Type	Values	Description
1	C_FAMILY	String	“Virtex2” “Virtex4” “Virtex5”	Target device family
2	C_MEM_TYPE	Integer	0: Single Port RAM 1: Simple Dual Port RAM 2: True Dual Port RAM 3: Single Port ROM 4: Dual Port RAM	Type of memory
3	C_ALOGRITHM		0 (selectable primitive), 1 (minimum area)	Type of algorithm

Table 17: **SIM Parameters *(Continued)***

	SIM Parameter	Type	Values	Description
4	C_PRIM_TYPE	Integer	0 (1-bit wide) 1 (2-bit wide) 2 (4-bit wide) 3 (9-bit wide) 4 (18-bit wide) 5 (36-bit wide) 6 (72-bit wide, single-port only)	If fixed primitive algorithm is chosen, defines which type of primitive to use to build memory
5	C_BYTE_SIZE	Integer	9, 8	Defines size of a byte; 9-bits or 8-bits
6	C_SIM_COLLISION_CHECK	String	None, Generate_X, All, Warnings_only	Defines warning collision checks in structural/unisim simulation model
7	C_COMMON_CLOCK	Integer	0, 1	Defines whether to optimize behavioral model collision check by assuming clocks are synchronous
8	C_DISABLE_WARN_BHV_COLL	Integer	0, 1	Disables the behavioral model from generating warnings due to read-write collisions
9	C_DISABLE_WARN_BHV_RANGE	Integer	0, 1	Disables the behavioral model from generating warnings due to address out of range
10	C_LOAD_INIT_FILE	Integer	0, 1	Defined whether to load initialization file
11	C_INIT_FILE_NAME	String	"…"	Name of initialization file (MIF format)
12	C_USE_DEFAULT_DATA	Integer	0, 1	Defines whether to use default data for the memory
13	C_DEFAULT_DATA	String	"…"	Defines a default data for the memory
14	C_HAS_MEM_OUTPUT_REGS	Integer	0, 1	Defines whether a register stage is added at the output of the memory latch
15	C_HAS_MUX_OUTPUT_REGS	Integer	0, 1	Defines whether a register stage is added at the output of the memory core
16	C_WRITE_WIDTHA	Integer	1 to 1152	Defines width of write port A
17	C_READ_WIDTHA	Integer	1 to 1152	Defines width of read port A
18	C_WRITE_DEPTHA	Integer	8 to 9011200	Defines depth of write port A
19	C_READ_DEPTHA	Integer	8 to 9011200	Defines depth of read port A
20	C_ADDRA_WIDTH	Integer	3 to 24	Defines the width of address A
21	C_WRITE_MODEA	String	Write_First, Read_first, No_change	Defines the write mode for port A

Table 17: **SIM Parameters *(Continued)***

	SIM Parameter	Type	Values	Description
22	C_HAS_ENA	Integer	0, 1	Defines whether port A has an enable pin
23	C_HAS_REGCEA	Integer	0, 1	Defines whether port A has an enable pin for its output register
24	C_HAS_SSRA	Integer	0, 1	Defines whether port A has an synchronous reset pin
25	C_SINITA_VAL	String	"…"	Defines initialization/power-on value for port A output
26	C_USE_BYTE_WEA	Integer	0, 1	Defines whether byte-write feature is used on port A
27	C_WEA_WIDTH	Integer	1 to 128	Defines width of WEA pin for port A
28	C_WRITE_WIDTHB	Integer	1 to 1152	Defines width of write port B
29	C_READ_WIDTHB	Integer	1 to 1152	Defines width of read port B
30	C_WRITE_DEPTHB	Integer	8 to 9011200	Defines depth of write port B
31	C_READ_DEPTHB	Integer	8 to 9011200	Defines depth of read port B
32	C_ADDRB_WIDTH	Integer	3 to 24	Defines the width of address B
33	C_WRITE_MODEB	String	Write_First, Read_first, No_change	Defines the write mode for port B
34	C_HAS_ENB	Integer	0, 1	Defines whether port B has an enable pin
35	C_HAS_REGCEB	Integer	0, 1	Defines whether port B has an enable pin for its output register
36	C_HAS_SSRB	Integer	0, 1	Defines whether port B has an synchronous reset pin
37	C_SINITB_VAL	String	"…"	Defines initialization/power-on value for port B output
38	C_USE_BYTE_WEB	Integer	0, 1	Defines whether byte-write feature is used on port B
39	C_WEB_WIDTH	Integer	1 to 128	Defines width of WEA pin for port B
40	C_USE_ECC	Integer	0,1	Determines ECC options. 0 = No ECC 1 = ECC

Output Register Configurations

The Block Memory Generator core allows optional output registers to improve the performance of the core. Because Virtex-5, Virtex-4 and Spartan-3A DSP block RAM primitives have embedded output registers that the previous architectures did not support, the configurations described below are separated into Virtex-4/Virtex-5, Spartan-3A DSP, and Virtex-II and implementations.

Virtex-5 and Virtex-4 Output Register Configurations

To tailor the register options, two check boxes are provided in the CORE Generator GUI under the Optional Output Registers section. The embedded output registers are enabled when Register Output of Memory Primitives is selected. Similarly, registers at the output of the core are enabled by selecting "Register output of memory core." Figure 26 Illustrates the implementations of these configuration options.

Virtex-5 and Virtex-4 Memory with Primitive and Core Output Registers

With both Register Output of Memory Primitives and Register Output of Memory Core options selected, a memory core is generated with both the Virtex-4 or Virtex-5 embedded output registers and a register on the output of the core. See Figure 26. This configuration can provide improved performance if you are building a large memory construct.

- ☑ Register Output of Memory Primitives
- ☑ Register Output of Memory Core

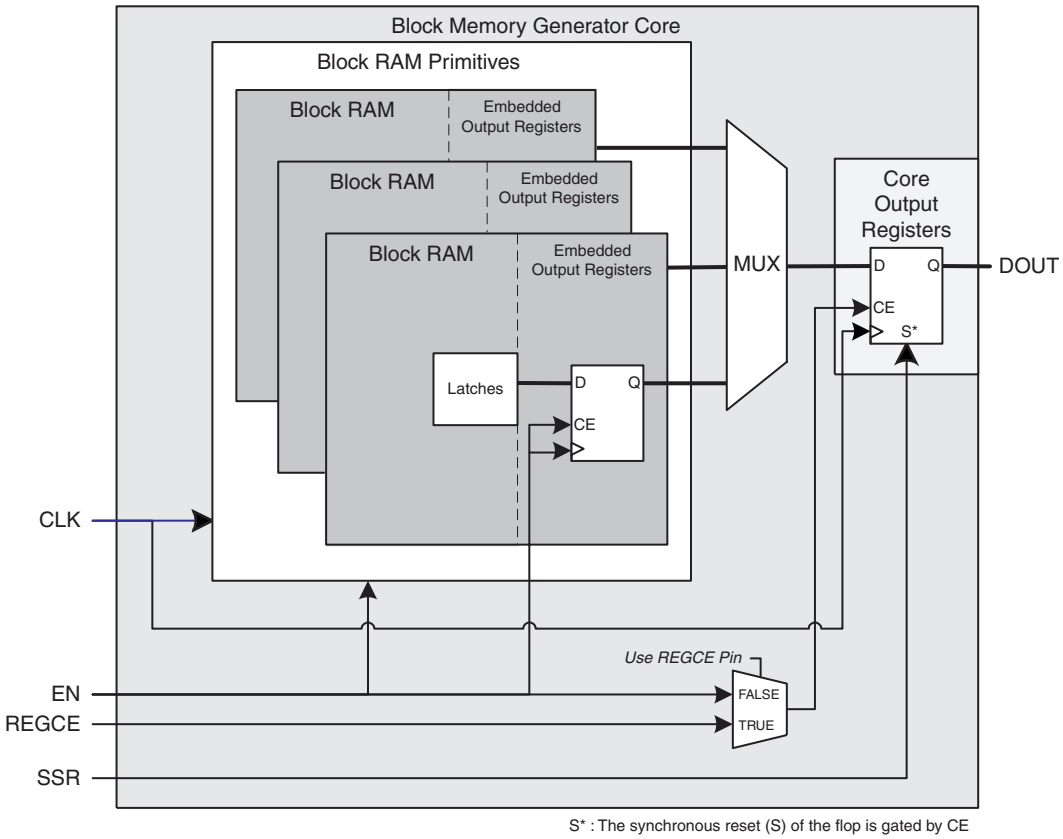

Figure 26: **Virtex-4 or Virtex-5 Block Memory Generated with Register Output of Memory Primitives and Register Output of Memory Core Enabled**

Virtex-5 Memory with Primitive Output Registers

When the option to Register Output of Memory Primitives is selected, then a memory core is generated that registers the output of the block RAM primitives. In Virtex-5, these registers are always implemented using the output registers embedded in the Virtex-5 block RAM architecture. The output of any multiplexing that may be required to combine multiple primitives is not registered in this configuration (Figure 27).

☑ Register Output of Memory Primitives

☐ Register Output of Memory Core

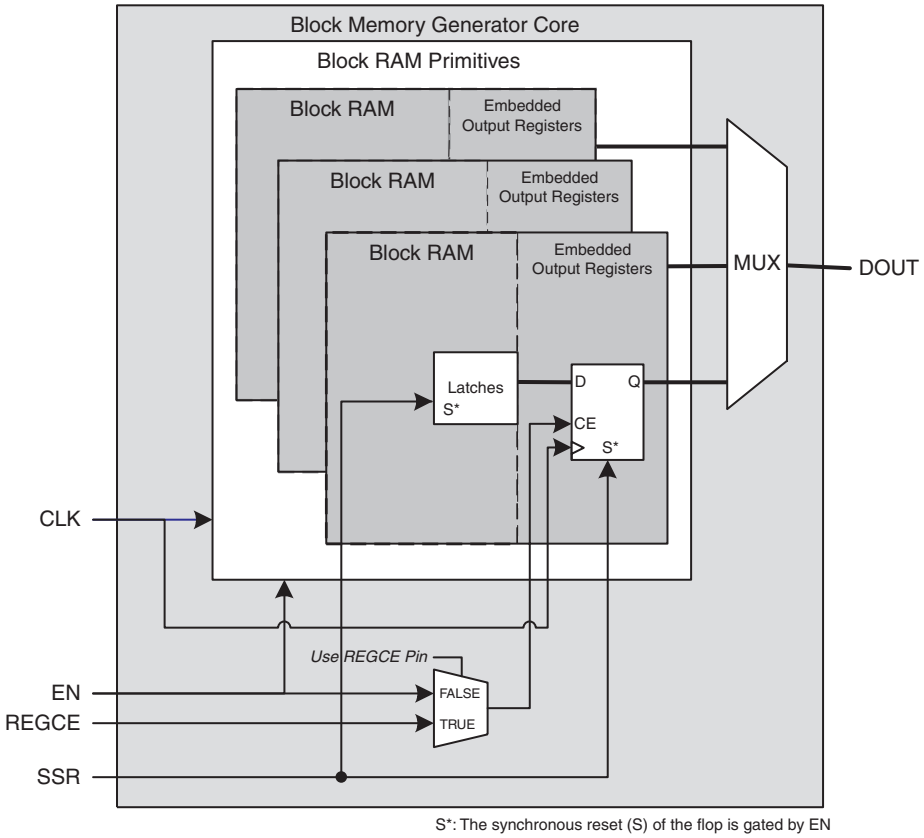

Figure 27: **Virtex-5 Block Memory Generated with Register Output of Memory Primitives Enabled**

Virtex-4 Memory with Primitive Output Registers without SSR

When Register Output of Memory Primitives is selected from Optional Output Registers on screen 4, and either the Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin) is unselected, a memory core that registers the output of the block RAM primitives using the output registers embedded in Virtex-4 architecture is generated. The output of any multiplexing that may be required to combine multiple primitives is not registered in this configuration (Figure 28).

- ☑ Register Output of Memory Primitives
- ☐ Register Output of Memory Core
- ☐ Either Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)

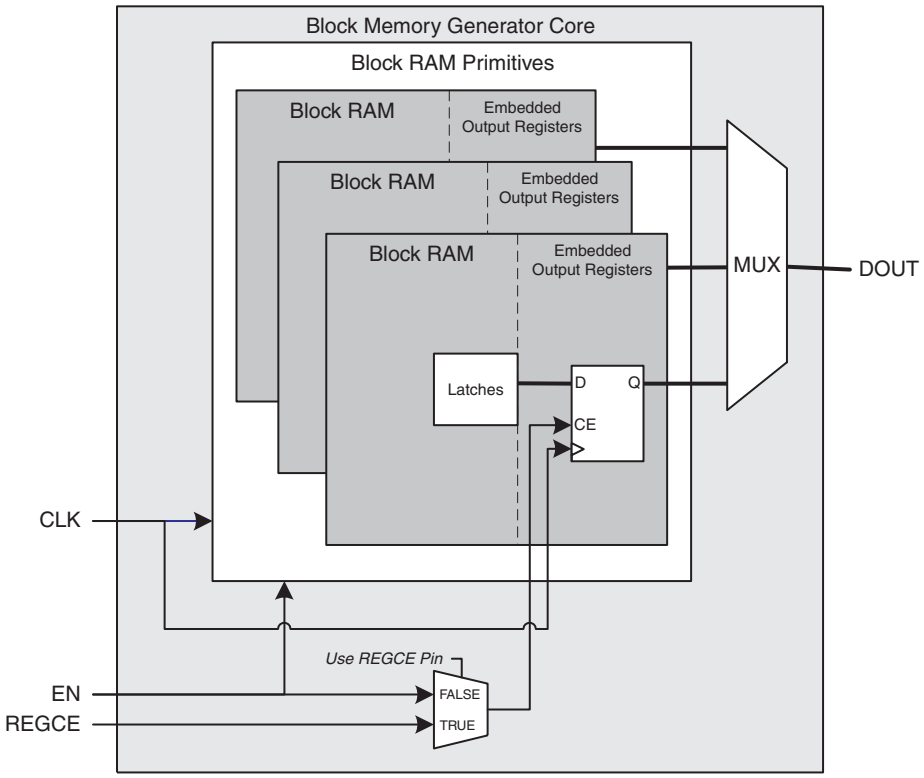

Figure 28: **Virtex-4 Block Memory Generated with only Register Output of Memory Primitives Enabled**

Virtex-4 Memory with Primitive Output Registers with SSR

If either Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin) is selected from the Output Reset section of the Port Options screen(s), the Virtex-4 embedded block RAM registers cannot be used. The primitive output registers are built from FPGA fabric, as illustrated in Figure 29.

- ☑ Register Output of Memory Primitives
- ☐ Register Output of Memory Core
- ☑ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)

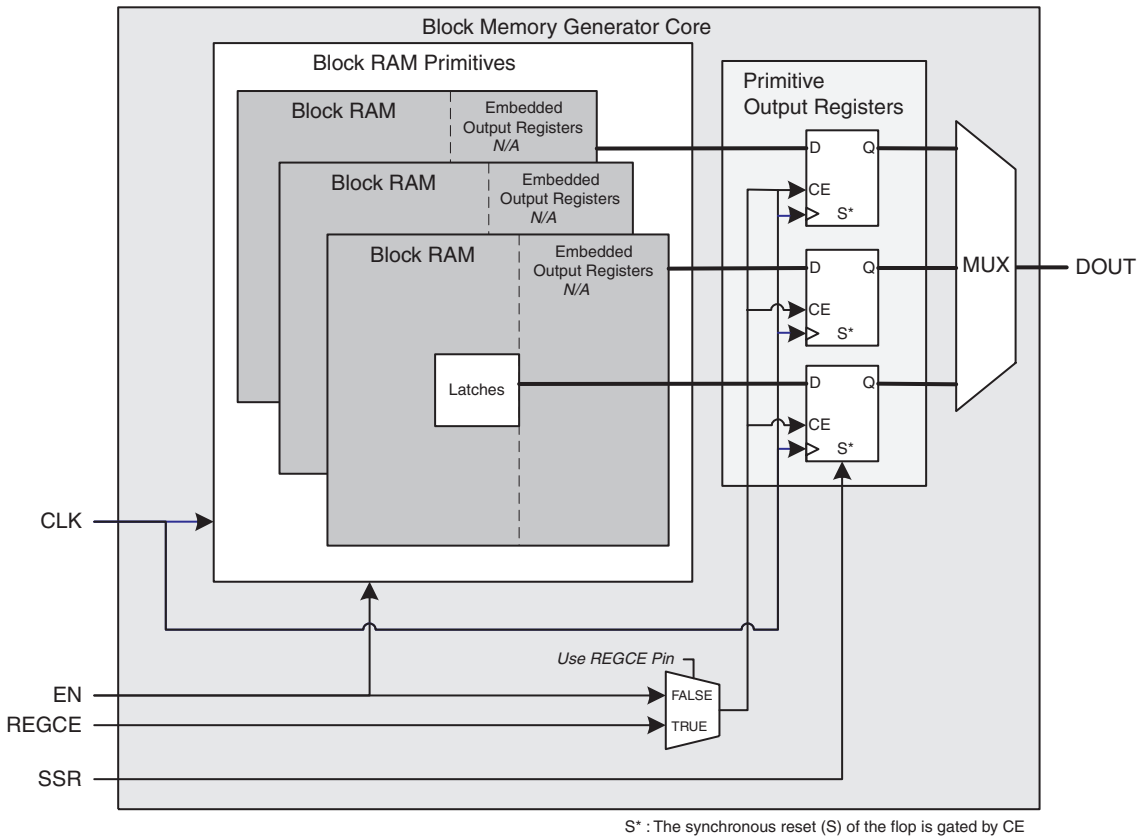

Figure 29: **Virtex-4 Block Memory Generated with Register Output of Memory Primitive and Use SSR[A|B] Pin Options Enabled**

Virtex-5 and Virtex-4 Memory with Core Output Registers

When only Register Output of Memory Core is selected on screen 4, the Virtex-5/Virtex-4 embedded registers are disabled in the generated core, as illustrated in Figure 30.

- ☐ Register Output of Memory Primitives
- ☑ Register Output of Memory Core

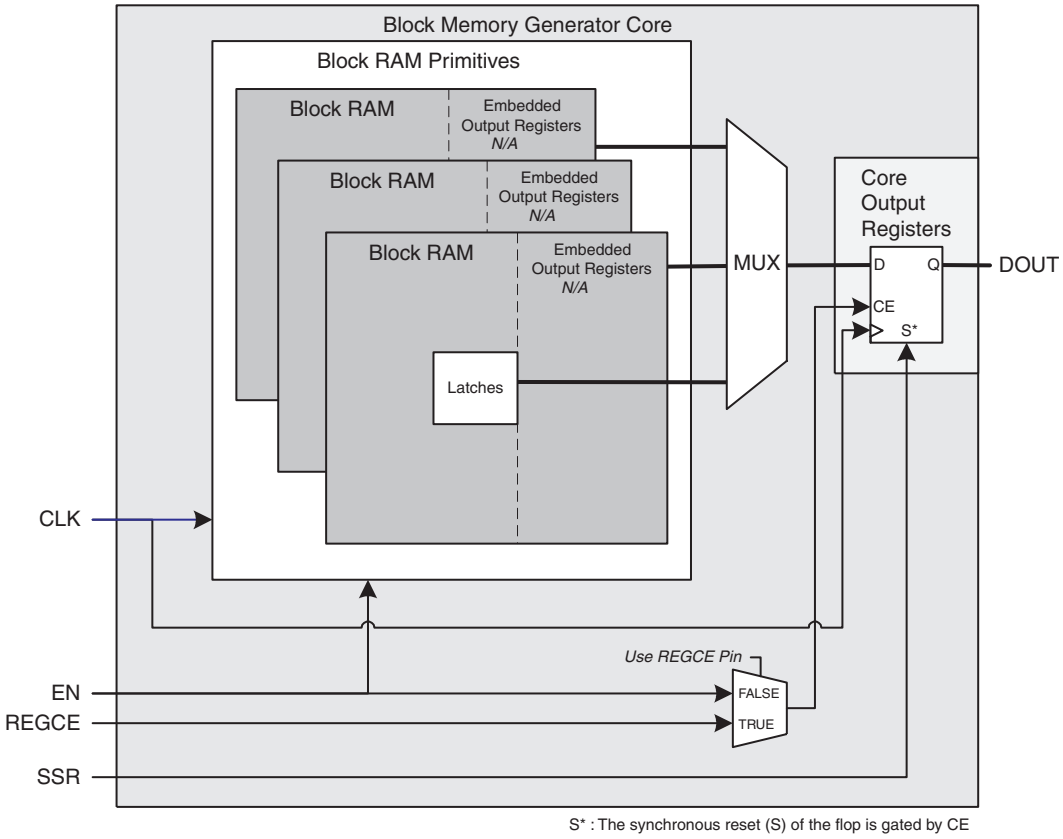

Figure 30: **Virtex-4 or Virtex-5 Block Memory Generated with Register Output of Memory Core Enabled**

Virtex-5 and Virtex-4 Memory with No Output Registers

If no output registers are selected, output of the memory primitive is driven directly from the RAM primitive latches. In this configuration, there are no additional clock cycles of latency, but the clock-to-out delay for a read operation can impact design performance. See Figure 31.

- Register Output of Memory Primitives
- Register Output of Memory Core

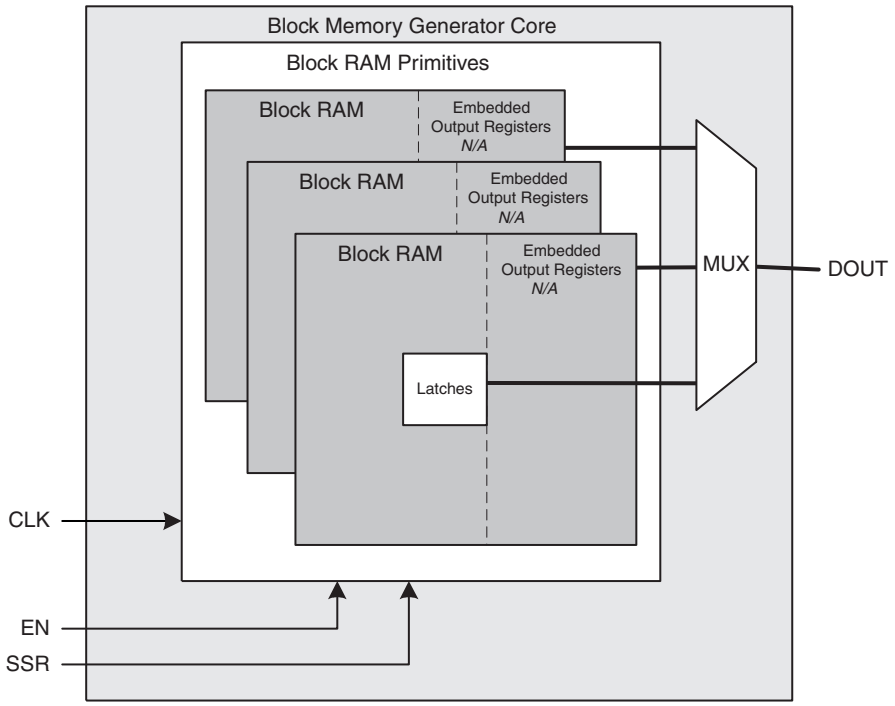

Figure 31: **Virtex-4 or Virtex-5 Block Memory Generated with No Output Registers Enabled**

Spartan-3A DSP Output Register Configurations

To tailor the register options, two options are provided in the CORE Generator GUI in the Optional Output Registers section. The embedded output registers are enabled when Register Output of Memory Primitives is selected. Similarly, registers at the output of the core are enabled by selecting Register Output of Memory Core. Figure 32 Illustrates the implementations of these configuration options.

When *only* Register Output of Memory Primitives is selected, and Use SSRA Pin (set/reset pin) or User SSRB Pin (set/reset pin) is selected, the User Reset Behavior from RAMB16BWER Primitive becomes available. Selecting this option forces the core to use the Spartan-3A DSP embedded output registers, but changes the core's behavior. For detailed information, see the sections that follow.

- ☑ Register Output of Memory Primitives
- ☑ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)

Spartan-3A DSP Memory with Primitive and Core Output Registers

With both Register Output of Memory Primitives and Register Output of Memory Core selected, a memory core is generated with both the Spartan-3A DSP embedded output registers and a register on the output of the core, as shown in Figure 32. This configuration can provide improved performance if you are building a large memory construct.

- ☑ Register Output of Memory Primitives
- ☑ Register Output of Memory Core

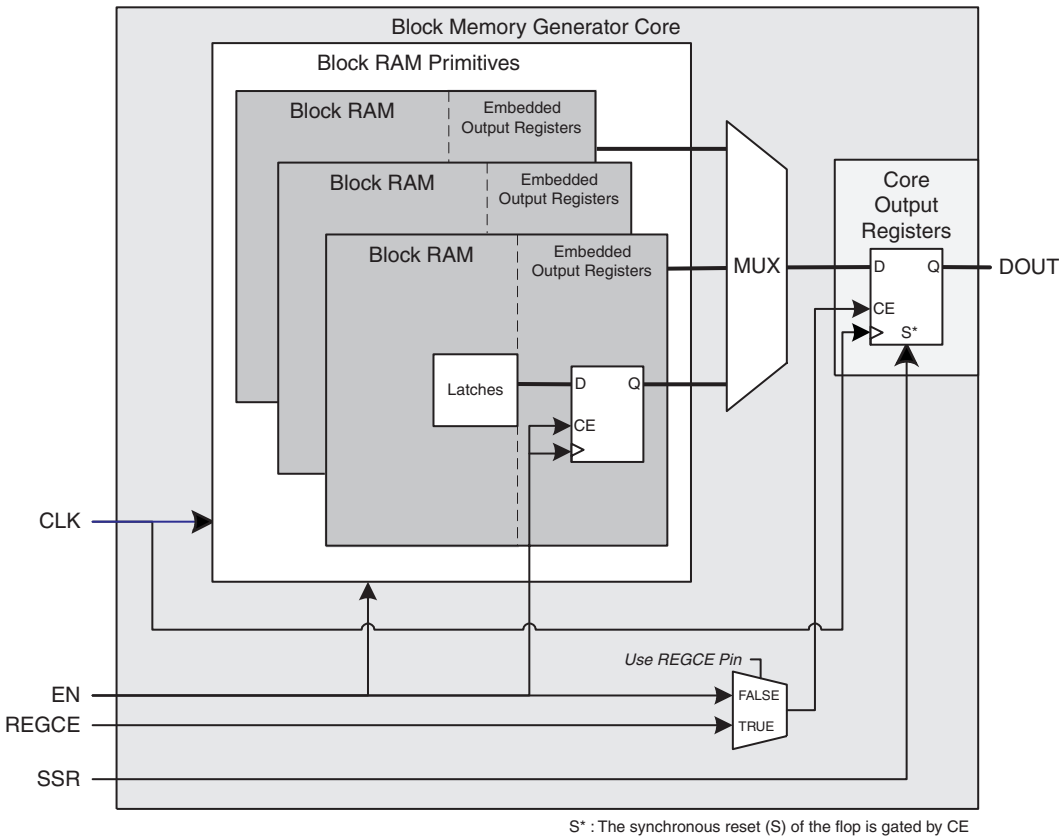

Figure 32: **Spartan-3A DSP Block Memory Generated with Register Output of Memory Primitives and Register Output of Memory Core Enabled**

Spartan-3A DSP Memory With Primitive Output Registers – Without SSR Pin

When Register Output of Memory Primitives is selected, and Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin) is not selected, a memory core that registers the output of the block RAM primitives using the output registers embedded in Spartan-3A DSP architecture is generated. The output of any multiplexing that may be required to combine multiple primitives is not registered in this configuration (Figure 33).

☑ Register Output of Memory Primitives

☐ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)

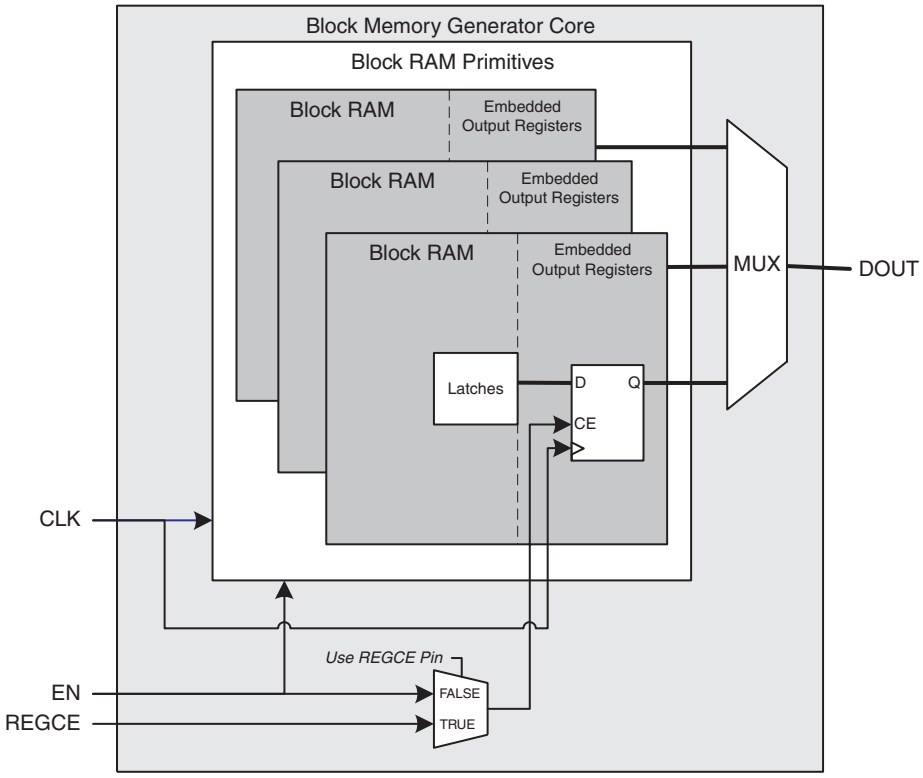

Figure 33: **Spartan-3A DSP Block Memory Generated with Register Output of Memory Primitives Enabled (No SSR)**

Spartan-3A DSP Memory with Primitive Output Registers, SSR – Without RAMB16BWER Reset Behavior

If Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin) is selected, and Use Reset Behavior from RAMB16BWER Primitive is *not* selected, the Spartan-3A DSP embedded block RAM registers cannot be used. The primitive output registers are built from FPGA fabric, as illustrated in Figure 34.

- ☑ Register Output of Memory Primitives
- ☐ Register Output of Memory Core
- ☑ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)
- ☐ Use Reset Behavior from RAMB16BWER Primitive

Note: This behavior is the same as that of f Spartan-3A, Virtex-5, Virtex-4 and Virtex-II.

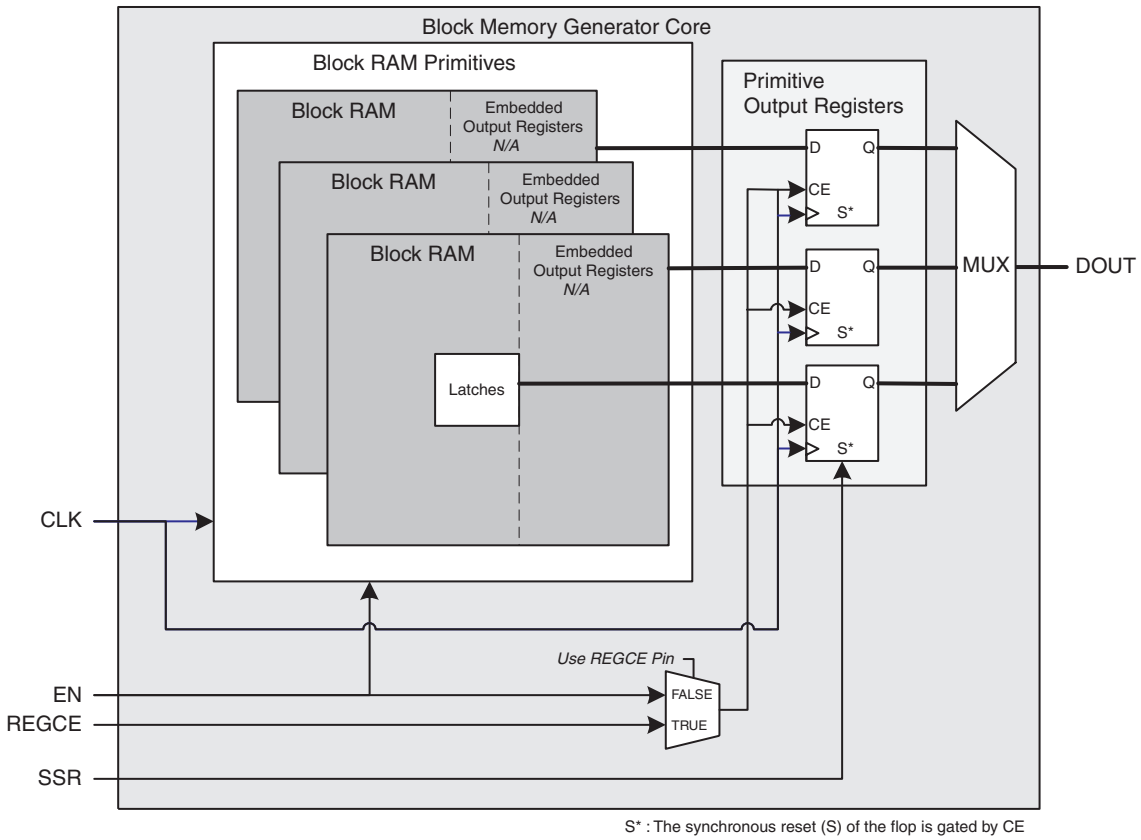

Figure 34: **Spartan-3A DSP Block Memory Generated with Register Output of Memory Primitive, Use SSR[A|B] Pin Options (With SSR), without RAMB16BWER Reset Behavior**

Spartan-3A DSP Memory with Primitive Output Registers, SSR, and with RAMB16BWER Reset Behavior

When Register Output of Memory Primitives, Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin), and Use Reset Behavior from RAMB16BWER Primitive are selected, the Spartan-3A DSP embedded registers are enabled in the generated core, as displayed in Figure 35.

If Use Reset Behavior from RAMB16BWER Primitive is selected, the Spartan-3A DSPs embedded output registers are used, but the reset behavior of the core changes. Figure 35 illustrates how the memory is constructed in this case. The functional differences between this and other implementations are that the SSR[A | B] input resets *both* the embedded output registers *and* the block RAM output latches. If EN and REGCE are held high, the output value is set to the reset value for two clock cycles following a reset.

In addition, the synchronous reset for both the latches and the embedded output registers are gated by the EN input to the core, independent of the state of REGCE. This differs from all other configurations of the Block Memory Generator where SSR is typically gated by REGCE.

- ☑ Register Output of Memory Primitives
- ☐ Register Output of Memory Core
- ☑ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)
- ☑ Use Reset Behavior from RAMB16BWER Primitive

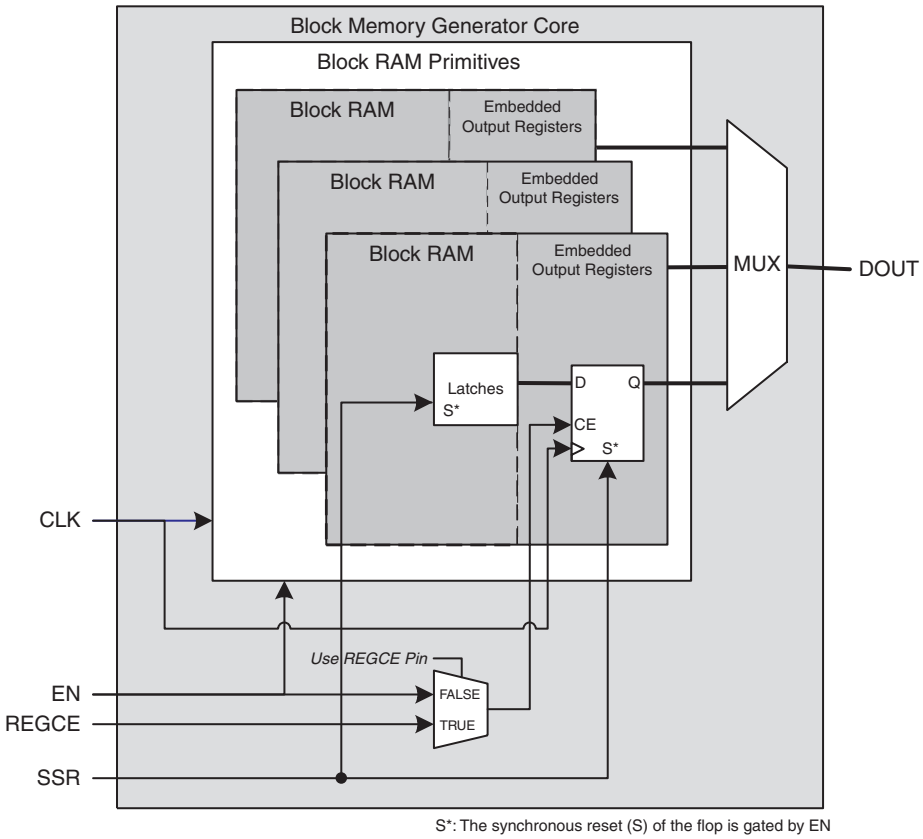

Figure 35: **Spartan-3A DSP Block Memory Generated with Register Output of Memory Primitives, Use SSR[A|B] Pin Options (With SSR), and RAMB16BWER Reset Behavior Enabled**

Spartan-3A DSP Memory with Core Output Registers

When Register Output of Memory Core is selected, the Spartan-3A DSP embedded registers are disabled in the generated core, as illustrated in Figure 36.

- ☐ Register Output of Memory Primitives
- ☑ Register Output of Memory Core
- ☑ Use SSRA Pin (set/reset pin) or Use SSRB Pin (set/reset pin)

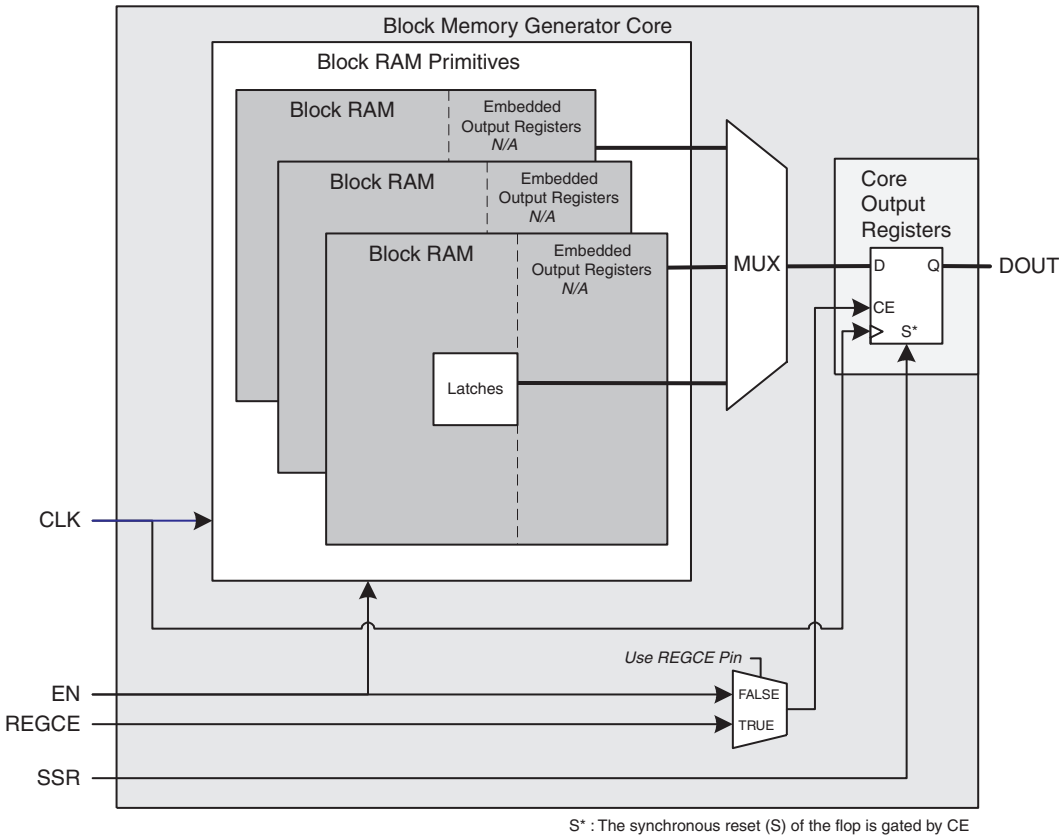

Figure 36: **Spartan-3A DSP Block Memory Generated with Register Output of Memory Core Enabled**

Spartan-3A DSP Memory with No Output Registers

If no output registers are selected, output of the memory primitive is driven directly from the RAM primitive latches. In this configuration, there are no additional clock cycles of latency, but the clock-to-out delay for a read operation can impact design performance. See Figure 37.

- Register Output of Memory Primitives
- Register Output of Memory Core

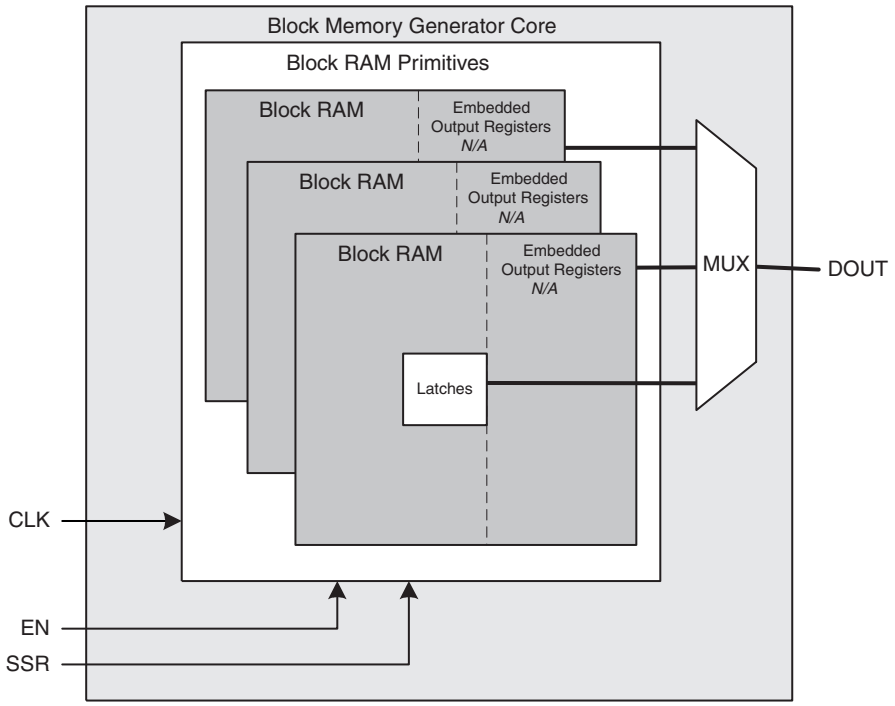

Figure 37: **Spartan-3A DSP Block Memory Generated with No Output Registers Enabled**

Virtex-II Output Register Configurations

To tailor the register options of Virtex-II architectures (and similar architectures like Spartan-3 and Spartan-3E), two options are provided in the CORE Generator GUI in the Optional Output Registers section. For implementing registers on the outputs of the individual block RAM primitives, Register Output of Memory Primitives is selected. In the same way, registering the output of the core is enabled by selecting Register Output of Memory Core. The resulting implementations of the four permutations supported by these configuration options are illustrated in the following sections.

Virtex-II Memory with Primitive and Core Output Registers

With Register Output of Memory Primitives and Register Output of Memory Core selected, a memory core is generated with registers on the outputs of the individual RAM primitives and on the core output, as displayed in Figure 38. Selecting this configuration can provide improved performance if you are building a large memory construct.

- ☑ Register Output of Memory Primitives
- ☑ Register Output of Memory Core

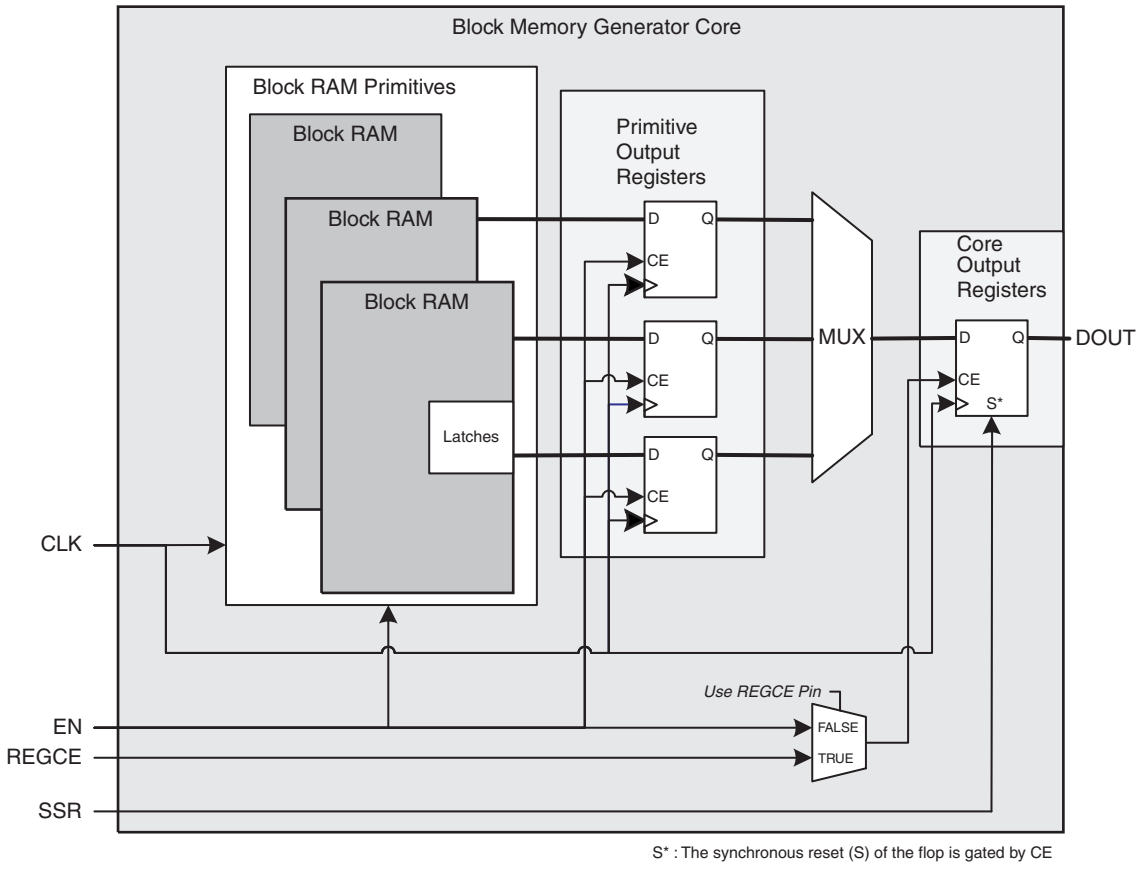

Figure 38: **Virtex-II Block Memory Generated with Register Output of Memory Primitives and Register Output of Memory Core Options Enabled**

Virtex-II Memory with Primitive Output Registers

When Register Output of Memory Primitives is selected, a core is generated that only registers the output of the RAM primitives. Note that the output of any multiplexing required to combine multiple primitives are not registered in this configuration, as shown in Figure 39.

- ☑ Register Output of Memory Primitives
- ☐ Register Output of Memory Core

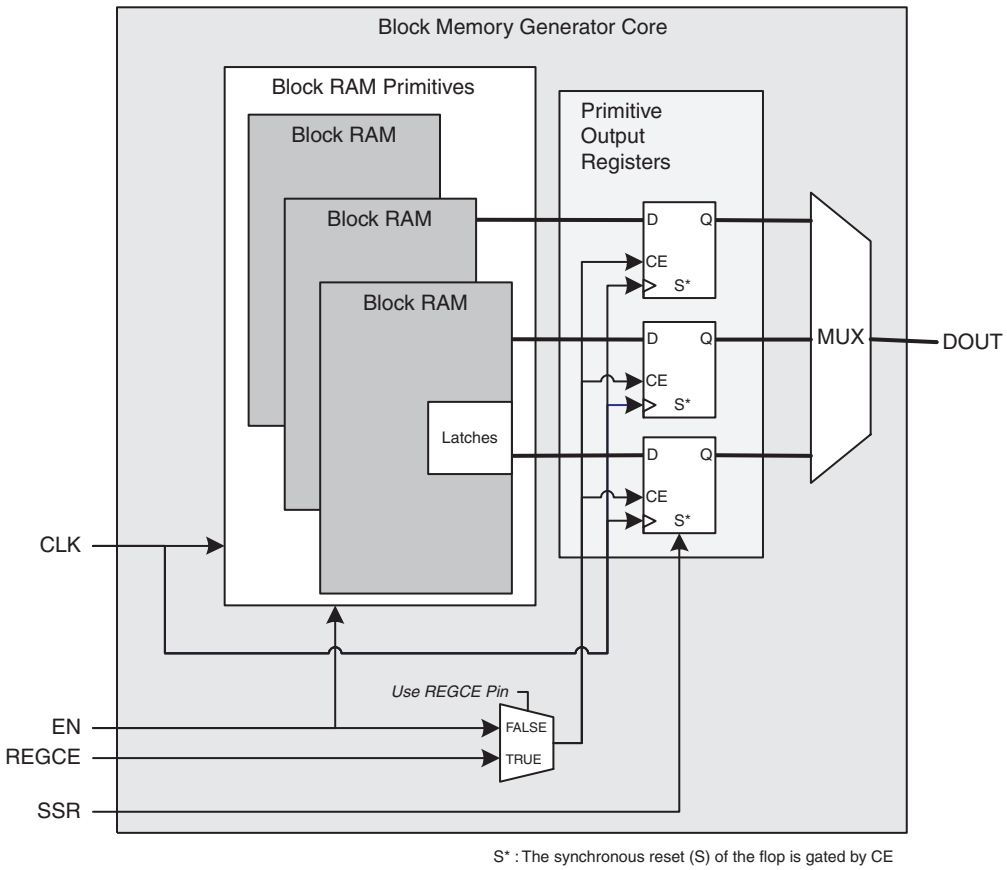

Figure 39: **Virtex-II Block Memory Generated with Register Output of Memory Primitives Enabled**

Virtex-II Memory with Core Output Registers

Figure 40 illustrates a memory configured with Register Output of Memory Core selected.

- ☐ Register Output of Memory Primitives
- ☑ Register Output of Memory Core

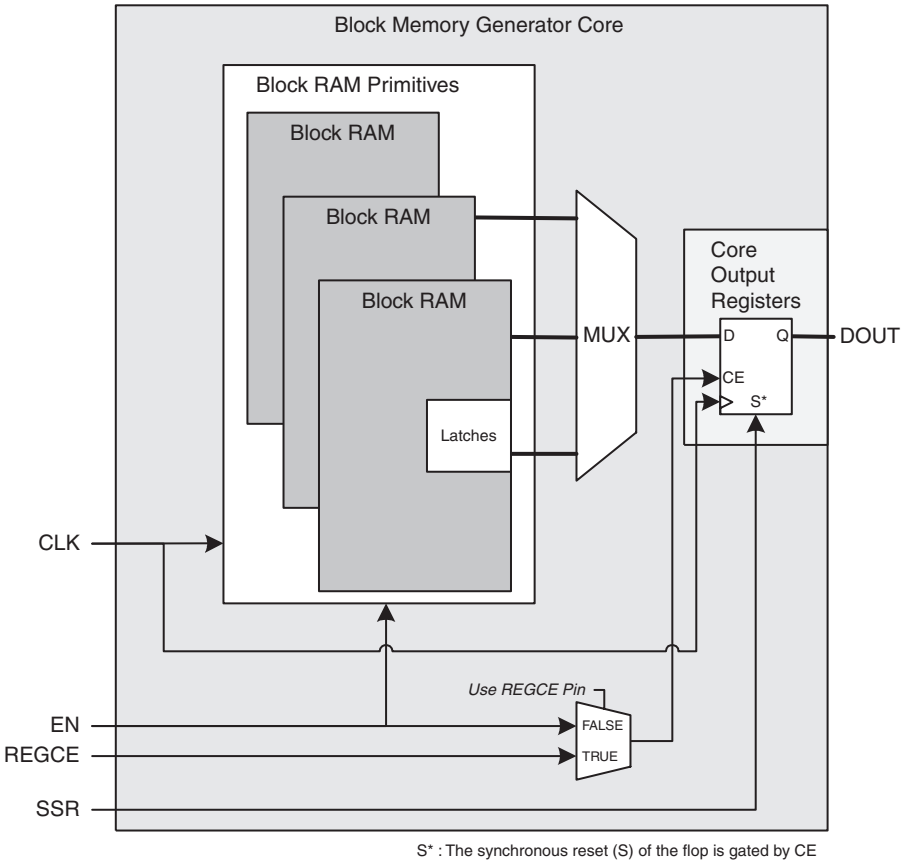

Figure 40: **Virtex-II Block Memory Generated with Register Output of Memory Core Enabled**

Virtex-II Memory with No Output Registers

When no output register options are selected, the output of the memory primitive is driven directly from the memory latches. In this configuration, there are no additional clock cycles of latency, but the clock-to-out delay for a read operation can impact design performance. See Figure 41.

- Register Output of Memory Primitives
- Register Output of Memory Core

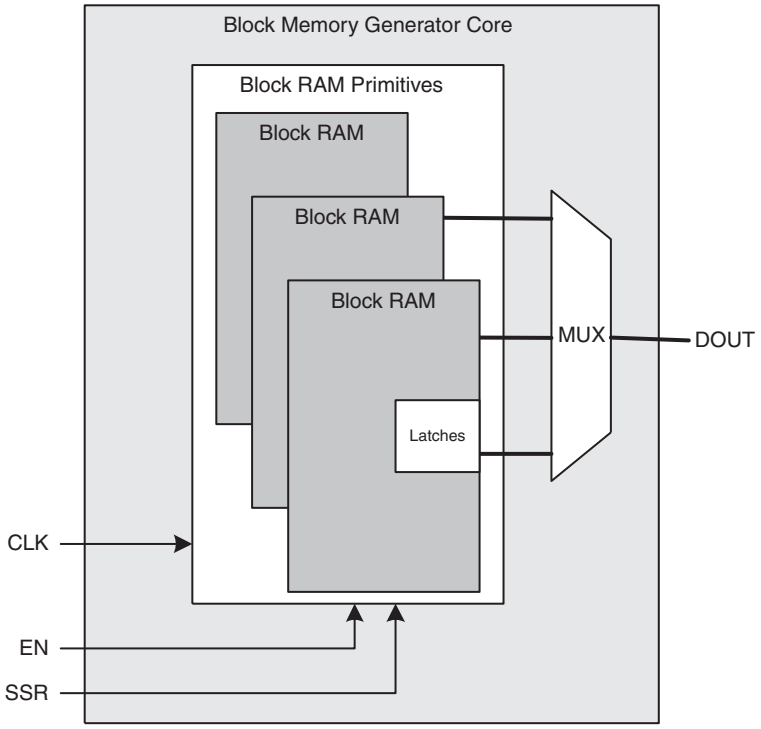

Figure 41: **Virtex-II Block Memory Generated with No Output Registers Enabled**

Ordering Information

This free core in included with the Xilinx ISE CORE Generator system. Updates to the core are bundled with ISE IP Updates, which are accessible from the Xilinx Download Center. To order Xilinx software, please contact your local Xilinx sales representative. Information on additional Xilinx LogiCORE modules is available at the Xilinx IP Center.

Related Information

Xilinx products are not intended for use in life-support appliances, devices, or systems. Use of a Xilinx product in such application without the written consent of the appropriate Xilinx officer is prohibited.

Revision History

Date	Version	Revision
01/11/06	1.0	Initial Xilinx release
4/12/06	2.0	Updated for Virtex-5 support
7/13/06	3.0	Updated primitives information in Table 1, replaced GUI screens, ISE version, release date.
9/21/06	4.0	Minor updates for v2.2 release
11/15/06	4.5	Updated for the v2.3 release
2/15/07	5.0	Updated for v2.4 release, added support for ECC.
4/02/07	5.5	Added support for Spartan-3A DSP devices.