Block Memory Generator v8.2

LogiCORE IP Product Guide

Vivado Design Suite

PG058 April 1, 2015

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx® LogiCORE™ IP Block Memory Generator (BMG) core is an advanced memory constructor that generates area and performance-optimized memories using embedded block RAM resources in Xilinx FPGAs.

The BMG core supports both Native and AXI4 interfaces.

The AXI4 interface configuration of the BMG core is derived from the Native interface BMG configuration and adds an industry-standard bus protocol interface to the core. Two AXI4 interface styles are available: AXI4 and AXI4-Lite.

Features

For details on the features of each interface, see Feature Summary in Chapter 1.

LogiCORE™ IP Facts Table	
Core Specifics	
Supported Device Family(1)	UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	AXI4, AXI4-Lite
Resources	See Resource Utilization.
Provided with Core	
Design Files	Encrypted RTL
Example Design	VHDL
Test Bench	VHDL
Constraints File	XDC
Simulation Model	Verilog and VHDL Behavioral(2)
Supported S/W Driver	N/A
Tested Design Flows(3)	
Design Entry	Vivado® Design Suite
Simulation	For supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete list of supported devices, see the Vivado IP catalog.
2. Behavioral models do not precisely model collision behavior. See Collision Behavior, page 54 for details.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

Overview

The Block Memory Generator core uses embedded Block Memory primitives in Xilinx® FPGAs to extend the functionality and capability of a single primitive to memories of arbitrary widths and depths. Sophisticated algorithms within the Block Memory Generator core produce optimized solutions to provide convenient access to memories for a wide range of configurations.

This core has two fully independent ports that access a shared memory space. Both A and B ports have a write and a read interface. In UltraScale™, Zynq®-7000 and 7 series FPGA architectures, each of the four interfaces can be uniquely configured with a different data width. When not using all four interfaces, you can select a simplified memory configuration (for example, a Single-Port Memory or Simple Dual-Port Memory) to reduce FPGA resource utilization.

This core is not completely backward-compatible with the discontinued legacy Single-Port Block Memory and Dual-Port Block Memory cores; for information about the differences, see Appendix B, Migrating and Upgrading.

Feature Summary

Features Common to the Native Interface and AXI4 BMG Cores

- Optimized algorithms for minimum block RAM resource utilization or low power utilization
- Configurable memory initialization
- Individual Write enable per byte in UltraScale™, Zynq-7000, Kintex®-7, and Virtex®-7 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
- Lower data widths for UltraScale™, Zynq-7000 and 7 series devices in SDP mode
- VHDL example design and demonstration test bench demonstrating the IP core design flow, including how to instantiate and simulate it

- Standard DOUT block RAM Cascading

Native Block Memory Generator Specific Features

- Generates Single-port RAM, Simple Dual-port RAM, True Dual-port RAM, Single-port ROM, and Dual-port ROM
- Supports memory sizes up to a maximum of 16 MBytes (byte size 8 or 9) (limited only by memory resources on selected part)
- Configurable port aspect ratios for dual-port configurations and Read-to-Write aspect ratios
- Supports the built-in Hamming Error Correction Capability (ECC). Error injection pins allow insertion of single and double-bit errors
- Supports soft Hamming Error Correction (Soft ECC) for data widths less than 64 bits
- Option to pipeline DOUT bus for improved performance in specific configurations
- Choice of reset priority for output registers between priority of SR (Set Reset) or CE (Clock Enable)
- Performance up to 450 MHz

AXI4 Interface Block Memory Generator Specific Features

- Supports AXI4 and AXI4-Lite interface protocols
- AXI4 compliant Memory and Peripheral Slave types
- Independent Read and Write Channels
- Zero delay datapath
- Supports registered outputs for handshake signals
- INCR burst sizes up to 256 data transfers
- WRAP bursts of 2, 4, 8, and 16 data beats
- AXI narrow and unaligned burst transfers
- Simple Dual-port RAM primitive configurations
- Performance up to 300 MHz
- Supports data widths up to 256 bits and memory depths from 1 to 1M words (limited only by memory resources on selected part)
- Symmetric aspect ratios
- Asynchronous active-Low reset

Native Block Memory Generator Feature Summary

Memory Types

The Block Memory Generator core uses embedded block RAM to generate five 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, optional output registers, and optional pins are selectable per port. For Simple Dual-port RAM, the operating modes are not selectable. See Collision Behavior, page 54 for additional information.

Selectable Memory Algorithm

The core configures block RAM primitives and connects them together using 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.
- **Low Power Algorithm**: The memory is generated such that the minimum number of block RAM primitives are enabled during a Read or Write operation.
- **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 core can generate memory structures from 1 to 4608 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, as shown in Table 1-1 through Table 1-3.

Table 1-1: **BMG Width and Depth (without Byte Write Enable)**

Memory Width (bits)	Memory Depth (words)
Less than or equal to 128 (≤128)	Less than or equal to 1M (≤1M)
Greater than 128 and less than or equal to 256 (>128 and ≤256)	Less than or equal to 512k (≤512k)
Greater than 256 and less than or equal to 512 (>256 and ≤512)	Less than or equal to 256k (≤256k)
Greater than 512 and less than or equal to 1024 (>512 and ≤1024)	Less than or equal to 128k (≤128k)
Greater than 1024 and less than or equal to 2048 (>1024 and ≤2048)	Less than or equal to 64k (≤64k)
Greater than 2048 and less than or equal to 4608 (>2048 and ≤4608)	Less than or equal to 32k (32k)

Table 1-2: **BMG Width and Depth: Byte Size 8 (with Byte Write Enable)**

Memory Width (bits)	Memory Depth (words)
Less than or equal to 128 (≤128)	Less than or equal to 1M (≤1M)
Greater than 128 and less than or equal to 256 (>128 and ≤256)	Less than or equal to 512k (≤512k)
Greater than 256 and less than or equal to 512 (>256 and ≤512)	Less than or equal to 256k (≤256k)
Greater than 512 and less than or equal to 1024 (>512 and ≤1024)	Less than or equal to 128k (≤128k)
Greater than 1024 and less than or equal to 2048 (>1024 and ≤2048)	Less than or equal to 64k (≤64k)
Greater than 2048 and less than or equal to 4096 (>2048 and ≤4096)	Less than or equal to 32k (32k)

Table 1-3: **BMG Width and Depth: Byte Size 9 (with Byte Write Enable)**

Memory Width (bits)	Memory Depth (words)
Less than or equal to 144 (≤144)	Less than or equal to 1M (≤1M)
Greater than 128 and less than or equal to 288 (>144 and ≤288)	Less than or equal to 512k (≤512k)
Greater than 288 and less than or equal to 576 (>288 and ≤576)	Less than or equal to 256k (≤256k)
Greater than 576 and less than or equal to 1152 (>576 and ≤1152)	Less than or equal to 128k (≤128k)
Greater than 1152 and less than or equal to 2304 (>1152 and ≤2304)	Less than or equal to 64k (≤64k)
Greater than 2304 and less than or equal to 4608 (>2304 and ≤4608)	Less than or equal to 32k (32k)

Selectable Operating Mode per Port

The Block Memory Generator core supports the following block RAM primitive operating modes: WRITE FIRST, READ FIRST, and NO CHANGE. Each port can be assigned its own operating mode.

Selectable Port Aspect Ratios

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

- The A port width might differ from the B port width by a factor of 1, 2, 4, 8, 16, or 32.
- The Read width might 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.

Optional Byte-Write Enable

The Block Memory Generator core provides byte-Write support for memory widths which are multiples of eight (no parity) or nine bits (with parity).

Optional Output Registers

The Block Memory Generator core provides two optional stages of output registering to increase memory performance. The output registers can be chosen for port A and port B separately. The core supports embedded block RAM registers as well as registers implemented in the FPGA general interconnect. See Output Register Configurations, page 119 for more information about using these registers.

Optional Pipeline Stages

The core provides optional pipeline stages within the MUX, available only when the registers at the output of the memory core are enabled and only for specific configurations. For the available configurations, the number of pipeline stages can be 1, 2, or 3. For detailed information, see Optional Pipeline Stages, page 57.

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 Set/Reset Pin

The core provides optional set/reset pins (`rsta` and `rstb`) for each port that initialize the Read output to a programmable value.

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.

Note: When using IP integrator, the initialization in BRAM Controller Mode is done only through the MEM file. For more details, see "MEM Files" of the *Data2MEM User Guide* (UG658) [Ref 3].

Hamming Error Correction Capability

Simple Dual-port RAM memories support the built-in FPGA Hamming Error Correction Capability (ECC) available block RAM primitives for data widths greater than 64 bits. The BuiltIn_ECC (ECC) memory automatically detects single- and double-bit errors, and is able to auto-correct the single-bit errors.

For data widths of 64 bits or less, a soft Hamming Error Correction implementation is available.

AXI4 Interface Block Memory Generator Feature Summary

Overview

AXI4 Interface Block Memories are built on the Native Interface Block Memories (see Figure 1-1). Two AXI4 interface styles are available: AXI4 and AXI4-Lite. The core can also be further classified as a Memory Slave or as a Peripheral Slave. In addition to applications supported by the Native Interface Block Memories, AXI4 Block Memories can also be used in AXI4 System Bus applications and Point-to-Point applications.

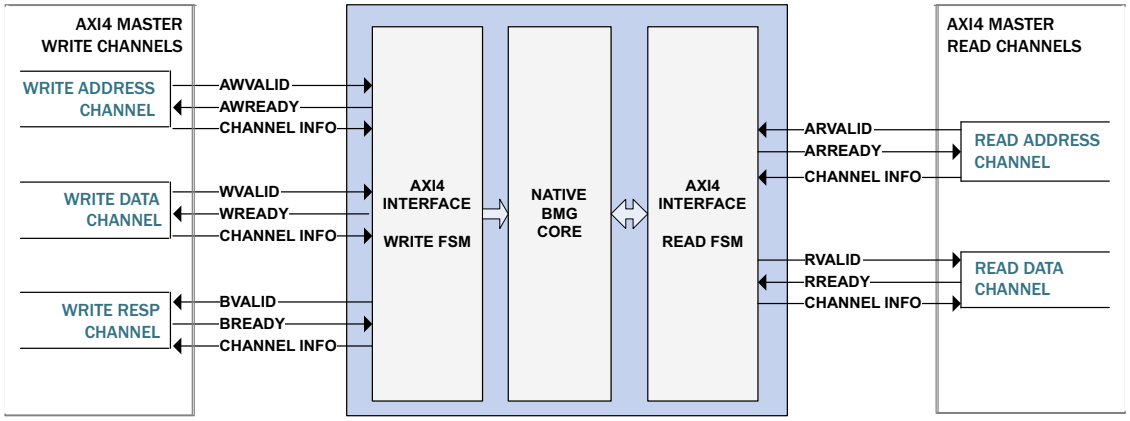

Figure 1-1: **AXI4 Interface BMG Block Diagram**

All communication in the AXI protocol is performed using five independent channels. Each of the five independent channels consists of a set of information signals and uses a two-way `valid` and `ready` handshake mechanism. The information source uses the `valid` signal to show when valid data or control information is available on the channel. The information destination uses the `ready` signal to show when it can accept the data.

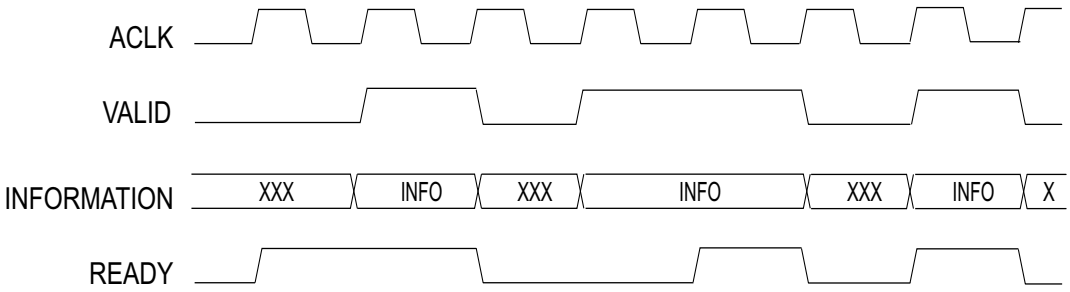

Figure 1-2: **AXI4 Interface Handshake Timing Diagram**

In Figure 1-2, the information source generates the `valid` signal to indicate when data is available.

The destination generates the `ready` signal to indicate that it can accept the data, and transfer occurs only when both the `valid` and `ready` signals are High.

The AXI4 Block Memory Generator core is an AXI4 endpoint Slave IP and can communicate with multiple AXI4 Masters in an AXI4 System or with Standalone AXI4 Masters in point to point applications. The core supports Simple Dual-Port RAM configurations. Because AXI4 Block Memories are built using Native interface Block Memories, they share many common features.

All Write operations are initiated on the Write Address Channel (AW) of the AXI bus. The AW channel specifies the type of Write transaction and the corresponding address information. The Write Data Channel (W) communicates all Write data for single or burst Write

operations. The Write Response Channel (B) is used as the handshaking or response to the Write operation.

On Read operations, the Read Address Channel (AR) communicates all address and control information when the AXI master requests a Read transfer. When the Read data is available to send back to the AXI master, the Read Data Channel (R) transfers the data and status of the Read operation

Applications

AXI4 Block Memories–Memory Slave Mode

AXI4 Block Memories in Memory Slave mode are optimized for Memory Mapped System Bus implementations. The AXI4 Memory Slave Interface Type supports aligned, unaligned or narrow transfers for incremental or wrap bursts.

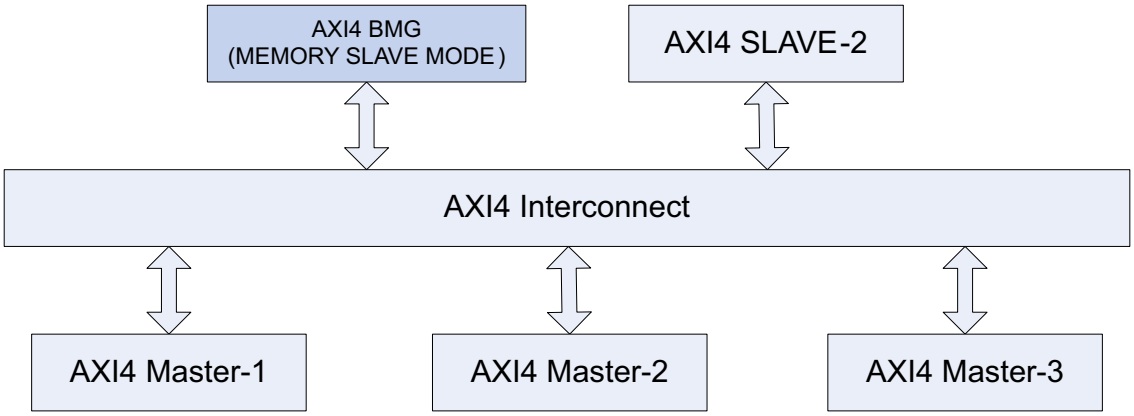

Figure 1-3: **AXI4 Memory Slave Application Diagram**

Figure 1-3 shows an example application for the AXI4 Memory Slave Interface Type with an AXI4 Interconnect for Multi Master AXI4 applications. Minimum memory requirement for this configuration is set to 4K bytes. Data widths supported by this configuration include 32, 64, 128 or 256 bits.

AXI4-Lite Block Memories–Memory Slave Mode

AXI4-Lite Block Memories in Memory Slave mode are optimized for the AXI4-Lite interface. They can be used in implementations requiring simple Control/Status Accesses. AXI4-Lite Memory Slave Interface Type supports only single burst transactions.

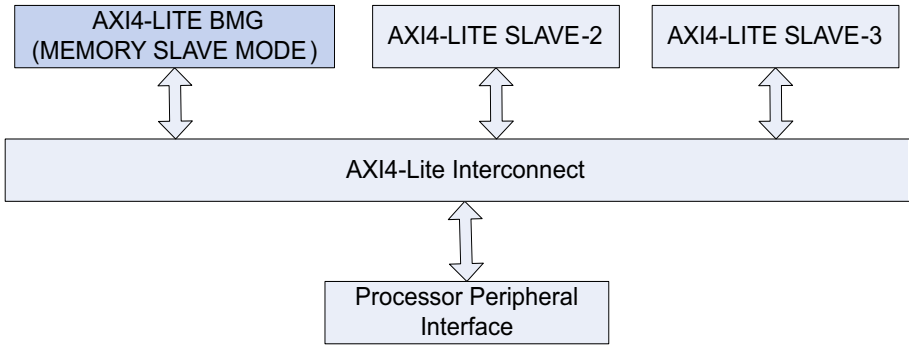

Figure 1-4: **AXI4-Lite Memory Slave Application Diagram**

Figure 1-4 shows an example application for AXI4-Lite Memory Slave Interface Type with an AXI4-Lite Interconnect to manage Control/Status Accesses. The minimum memory requirement for this configuration is set to 4K bytes. Data widths of 32 and 64 bits are supported by this configuration.

AXI4 Block Memories–Peripheral Slave Mode

AXI4 Block Memories in Peripheral Slave mode are optimized for a system or applications requiring data transfers that are grouped together in packets. The AXI4 Peripheral Slave supports aligned /unaligned addressing for incremental bursts.

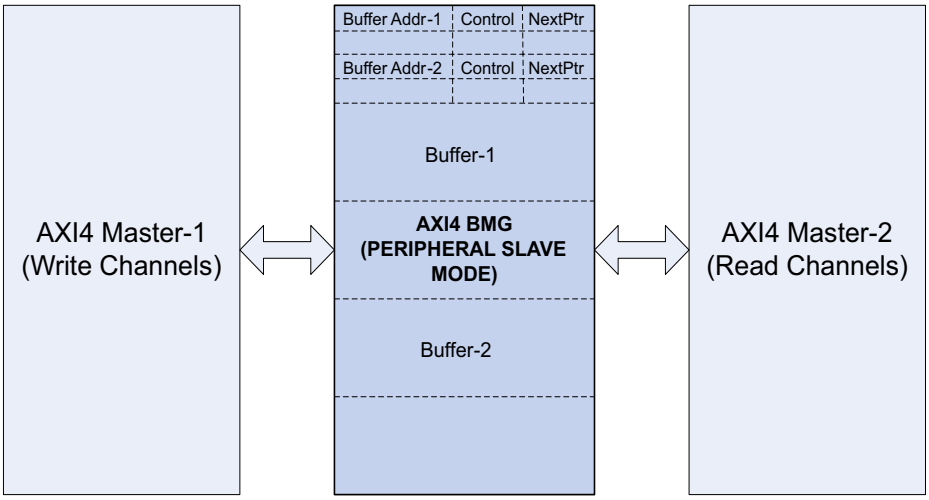

Figure 1-5: **AXI4 Peripheral Slave Application Diagram**

Figure 1-5 shows an example application for the AXI4 Peripheral Slave Interface Type in a Point-to-point buffered link list application. There is no minimum memory requirement set for this configuration. Data widths supported by this configuration include 8, 16, 32, 64, 128 and 256 bits.

AXI4-Lite Block Memories–Peripheral Slave Mode

AXI4-Lite Block Memories in Peripheral Slave mode are optimized for the AXI4-Lite interface. They can be used in implementations requiring single burst transactions.

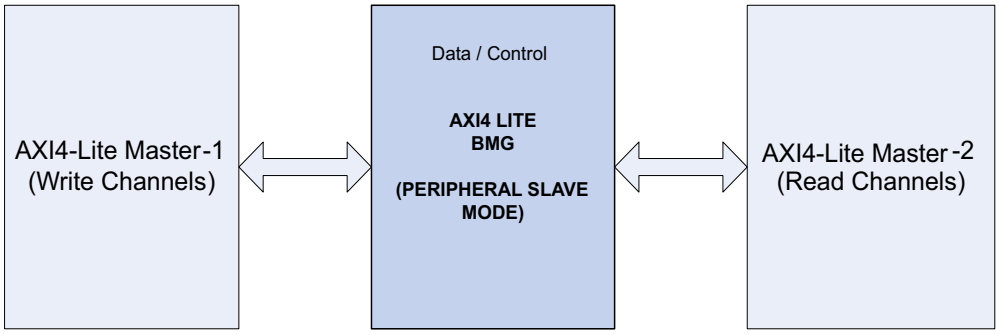

Figure 1-8: **AXI4-Lite Peripheral Slave Application Diagram**

Figure 1-8 shows an example application for the AXI4-Lite Memory Slave Interface Type in a Point-to-point application. There is no minimum memory requirement set for this configuration. Data widths supported by this configuration include 8, 16, 32 and 64 bits.

AXI4 BMG Core Channel Handshake Sequence

Figure 1-9 and Figure 1-10 illustrates an example handshake sequence for AXI4 BMG core. Figure 1-9 illustrates single burst Write operations to block RAM. By default the `awready` signal is asserted on the bus so that the address can be captured immediately during the clock cycle when both `awvalid` and `awready` are asserted. (With the default set in this manner, there is no need to wait an extra clock cycle `awready` to be asserted first.) By default, the `wready` signal is deasserted. Upon detecting `awvalid` being asserted, the `wready` signal is asserted (AXI4 BMG core has registered an AXI Address and is ready to accept Data), and when `wvalid` is also asserted, writes are performed to the block RAM. If the write data channel (`wvalid`) is presented prior to the write address channel valid (`awvalid`) assertion, the write transactions are not initiated until the write address channel has valid information.

The Block Memory Generator core with AXI4 Interface asserts `bvalid` for each transaction only after the last data transfer is accepted. The core also does not wait for the master to assert `bready` before asserting `bvalid`.

Note: The signal names in the figures displayed in upper case are the same as the lower case signal names referred to in the text.

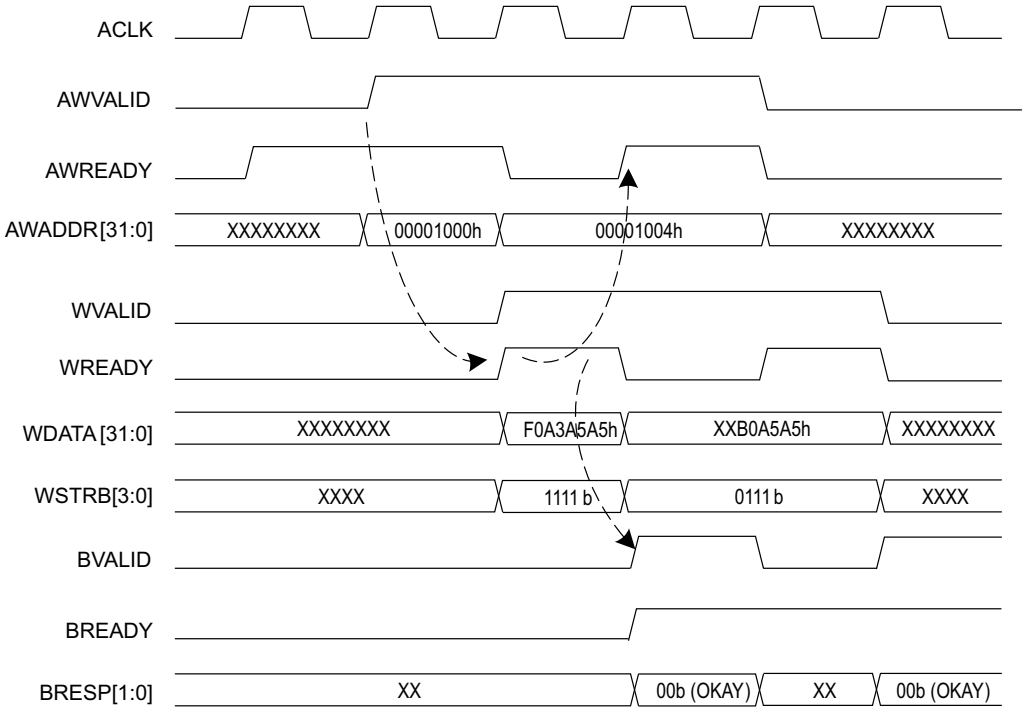

Figure 1-9: **AXI4-Lite Single Burst Write Transactions**

Figure 1-10 illustrates single burst Read operations to block RAM. The registered `arready` signal output on the AXI Read Address Channel interface defaults to a High assertion. The AXI Read FSM can accept the read address in the clock cycle where the `arvalid` signal is first valid.

The AXI Read FSM can accept a same clock cycle assertion of the `rready` by the master if the master can accept data immediately. When the `rready` signal is asserted on the AXI bus by the master, the Read FSM either negates the `rvalid` signal or places the next valid data on the AXI Bus.

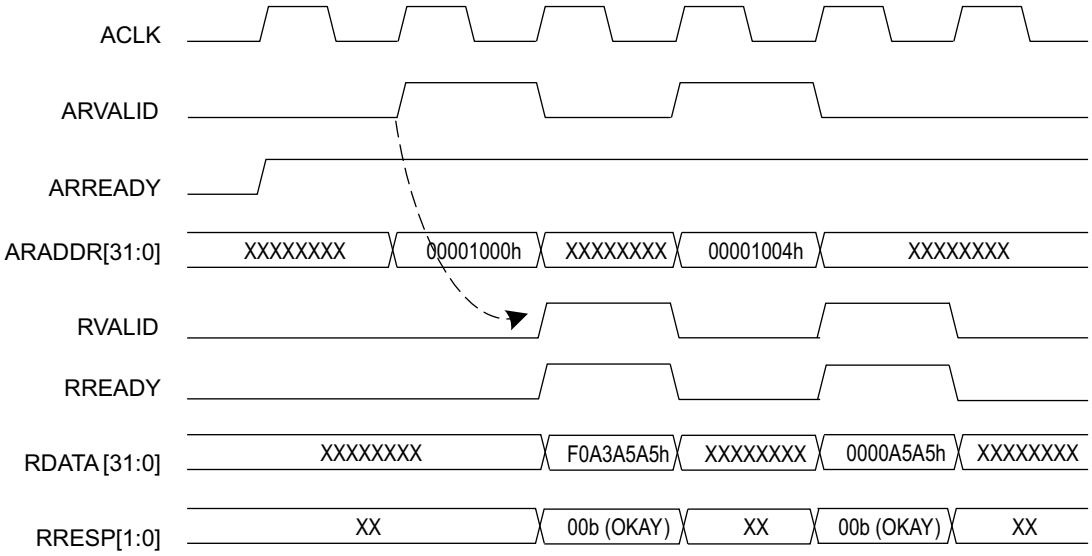

Figure 1-10: **AXI4 Lite Single Burst Read Transactions**

For more details on AXI4 Channel handshake sequences refer to the "Channel Handshake" section of the *AXI Protocol Specification* [Ref 1].

AXI4-Lite Single Burst Transactions

For AXI4 Lite interfaces, all transactions are burst length of one and all data accesses are the same size as the width of the data bus. Figure 1-9 and Figure 1-10 illustrates timing of AXI 32-bit write operations to the 32-bit wide block RAM. Figure 1-9 example illustrates single burst Write operations to block RAM addresses 0x1000h and 0x1004h. Figure 1-10 illustrates single burst Read operations to block RAM addresses 0x1000h and 0x1004h.

AXI4 Incremental Burst Support

Figure 1-11 illustrates an example of the timing for an AXI Write burst of four words to a 32-bit block RAM. The address Write channel handshaking stage communicates the burst type as INCR, the burst length of two data transfers (`awlen` = 01h). The Write burst utilizes all byte lanes of the AXI data bus going to the block RAM (`awsize` = 010b).

In compliance with AXI Protocol, the burst termination boundary for a transaction is determined by the length specified in the `awlen` signal. The allowable burst sizes for INCR bursts are from 1 (00h) to 256 (FFh) data transfers.

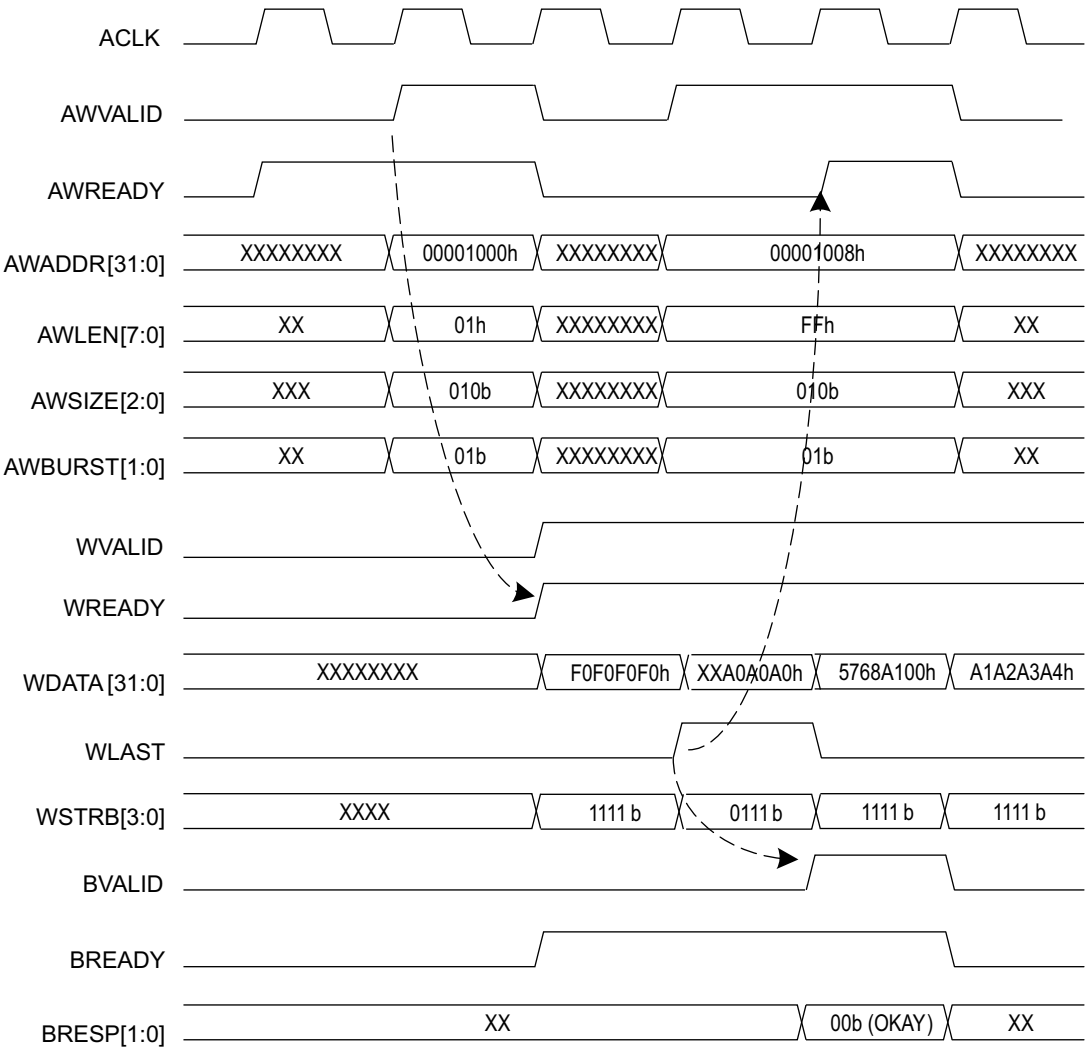

Figure 1-11: **AXI4 Incremental Write Burst Transactions**

Figure 1-12 illustrates the example timing for an AXI Read burst with block RAM managed by the Read FSM. The memory Read burst starts at address 0x1000h of the block RAM. On the AXI Read Data Channel, the Read FSM enables the AXI master/Interconnect to respond to the `rvalid` assertion when `rready` is asserted in the same clock cycle. If the requesting AXI master/Interconnect throttles on accepting the Read burst data (by negating `rready`), the Read FSM handles this by holding the data pipeline until `rready` is asserted.

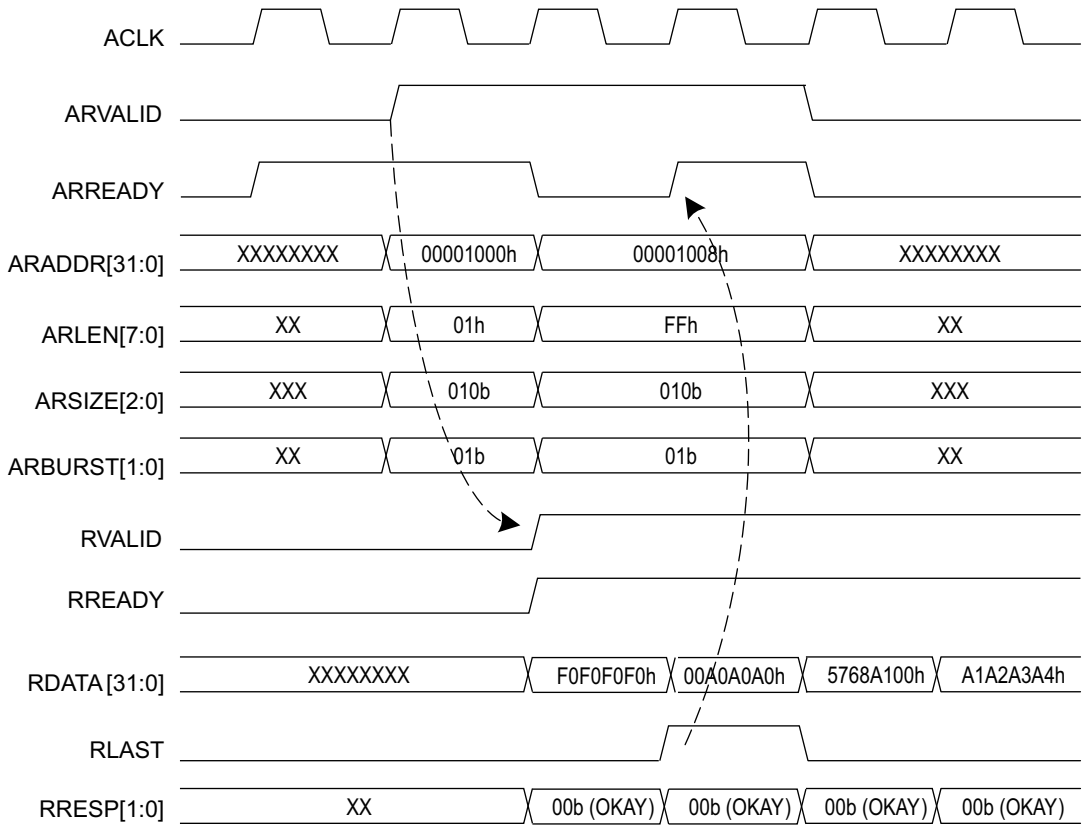

Figure 1-12: **AXI4 Incremental Read Burst Transactions**

AXI4 Wrap Burst Support

Cache line operations are implemented as WRAP burst types on AXI when presented to the block RAM. The allowable burst sizes for WRAP bursts are 2, 4, 8, and 16. The `awburst`/`arburst` must be set to 10 for the WRAP burst type.

WRAP bursts are handled by the address generator logic of the Write and Read FSM. The address seen by the block RAM must increment to the address space boundary, and then wrap back around to the beginning of the cache line address. For example, a processor issues a target word first cache line Read request to address 0x04h. On a 32-bit block RAM, the address space boundary is 0xFFh. This means that the block RAM sees the following sequence of addresses for Read requests: 0x04h, 0x08h, 0x0Ch, 0x00h. Note the wrap of the cache line address from 0xCh back to 0x00h at the end.

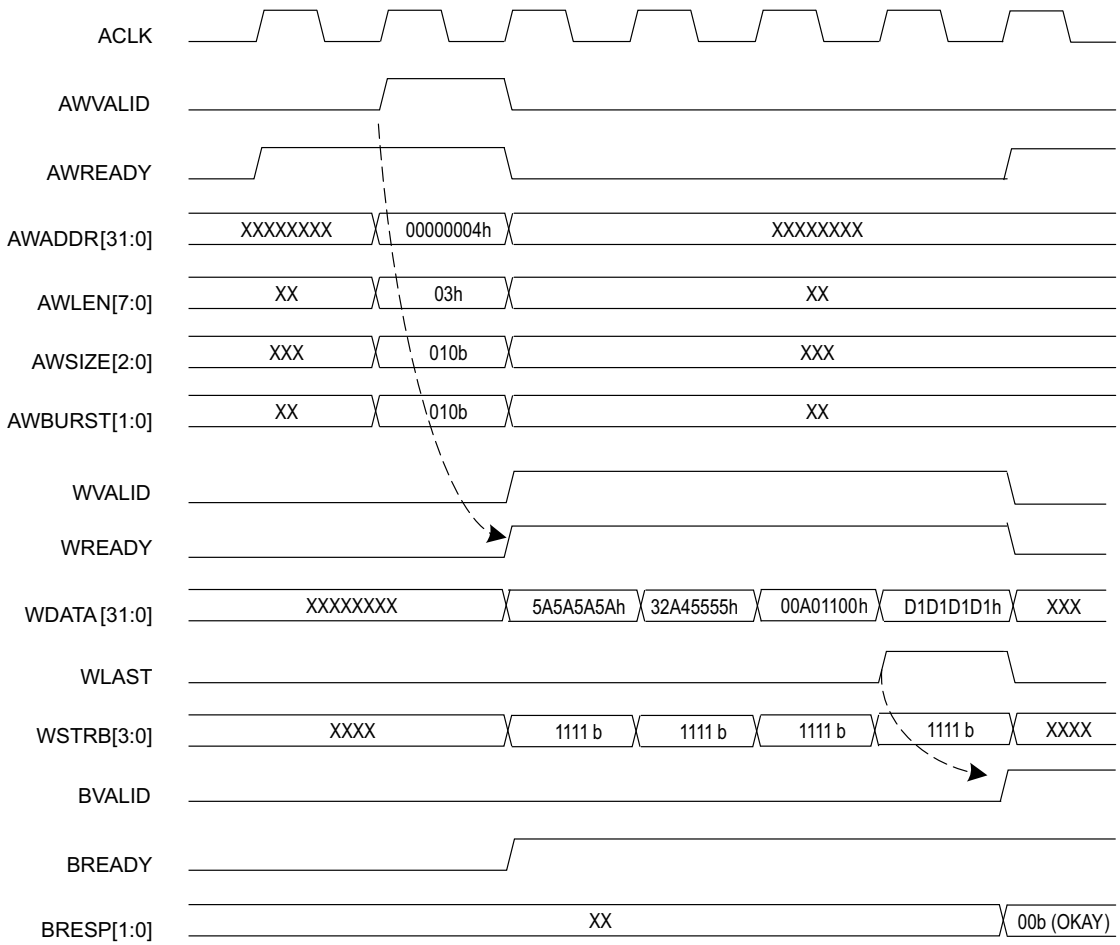

Figure 1-13: **AXI4 Wrap Write Burst Transactions**

Figure 1-13 illustrates the timing for AXI Wrap or cache line burst transactions. The address generated and presented to the block RAM starts at the target word and wraps around after the address space boundary is reached.

Figure 1-14 illustrates the timing on AXI WRAP or cache line burst Read transactions.

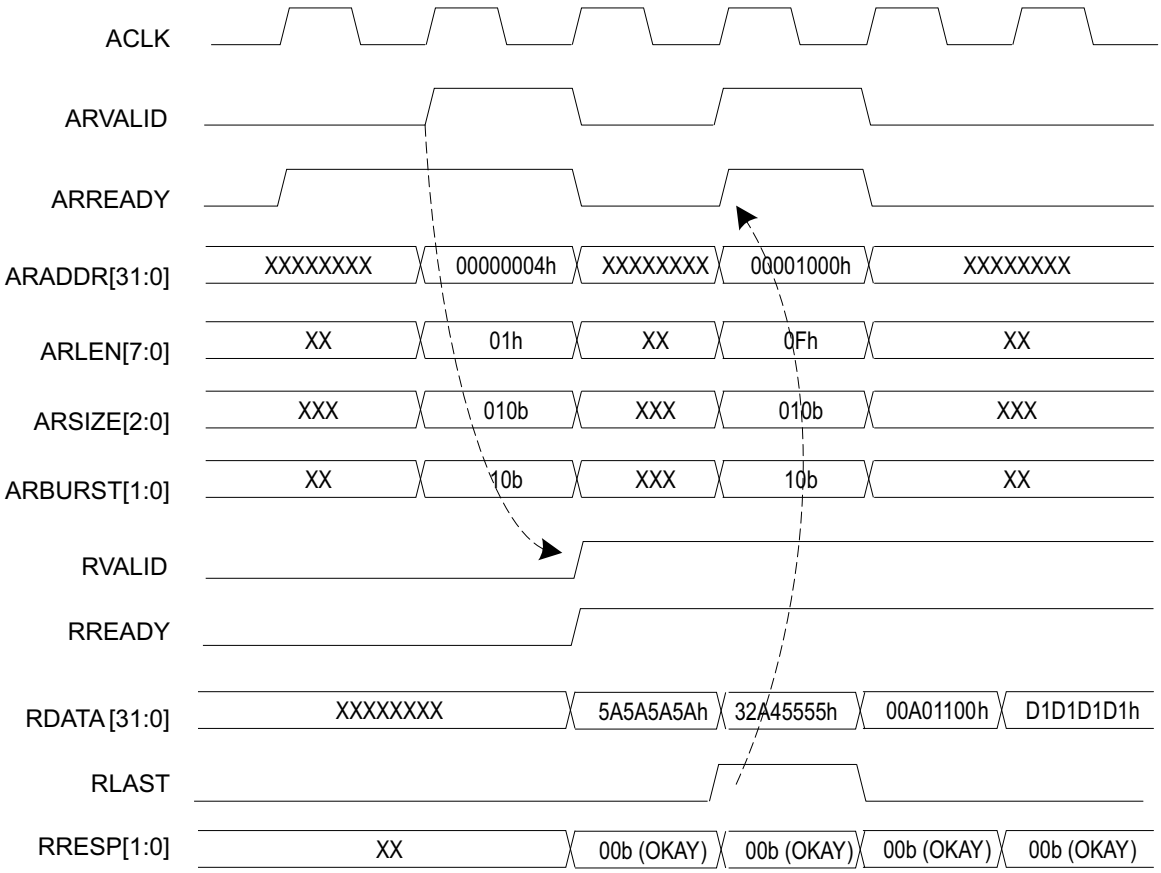

Figure 1-14: **AXI4 Wrap Read Burst Transactions**

Table 1-4 provides example address sequence to the block RAM for Wrap transactions.

Table 1-4: **Example Address Sequence for AXI4 BMG Core Wrap Transactions**

Memory Width	Transfer Size	Start Address	Burst Length	AXI4 BMG Core Address Sequence
32-bits	32-bits	0x100Ch	2	0x100Ch(1), 0x1008h
32-bits	32-bits	0x1008h	4	0x1008h, 0x100Ch(1), 0x1000h, 0x1004h
64-bits	64-bits	0x1008h	8	0x1008h, 0x1010h, 0x1018h, 0x1020h, 0x1028h, 0x1030h, 0x1038h(1), 0x1000h
64-bits	16-bits	0x1008h	16	0x1008h, 0x100Ah, 0x100Ch, 0x100Eh, 0x1010, 0x1012, 0x1014, 0x1016h, 0x1018h, 0x101Ah, 0x101Ch, 0x101Eh(1), 0x1000h, 0x1002h, 0x1004h, 0x1006h

Notes:

1. Calculated Wrap Boundary address.

For more details on AXI4 Wrap Burst Transactions and Wrap boundary calculations, refer to the Burst Addressing section of the *AXI Protocol Specification* [Ref 1].

AXI4 Narrow Transactions

A narrow burst is defined as a master bursting a data size smaller than the block RAM data width. If the burst type (awburst) is set to incr or wrap, then the valid data on the block RAM interface to the AXI bus rotates for each data beat. The Write and Read FSM handles each data beat on the AXI as a corresponding data beat to the block RAM, regardless of the smaller valid byte lanes. In this scenario, the AXI wsrtb is translated to the block RAM Write enable signals. The block RAM address only increments when the full address (data) width boundary is met with the narrow Write to block RAM.

Figure 1-15 illustrates an example of AXI narrow Write bursting with a 32-bit block RAM and the AXI master request is a half-word burst of four data beats. awsize is set to 001b.

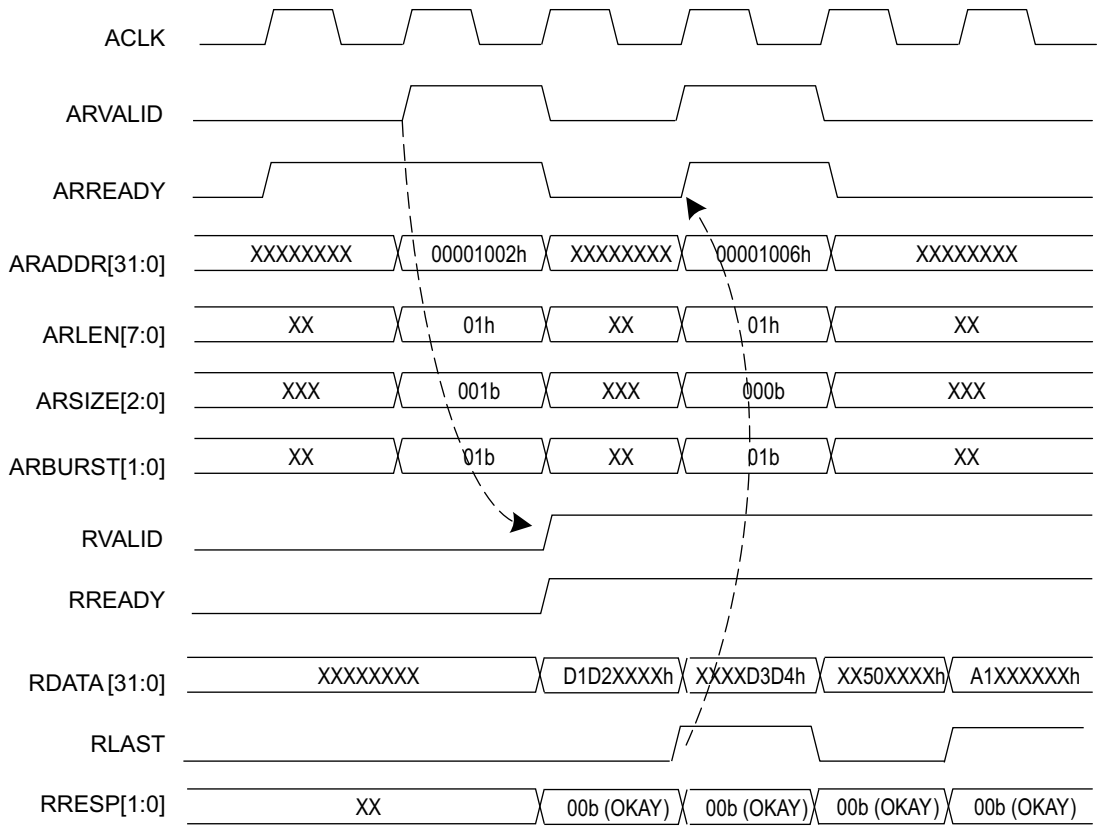

Figure 1-15: **AXI4 Narrow Write Burst Transactions**

Figure 1-16 illustrates an example of AXI "narrow" Read bursting with a 32-bit block RAM and the AXI master request is a half-word burst of 4 data beats. ARSIZE is set to 001b.

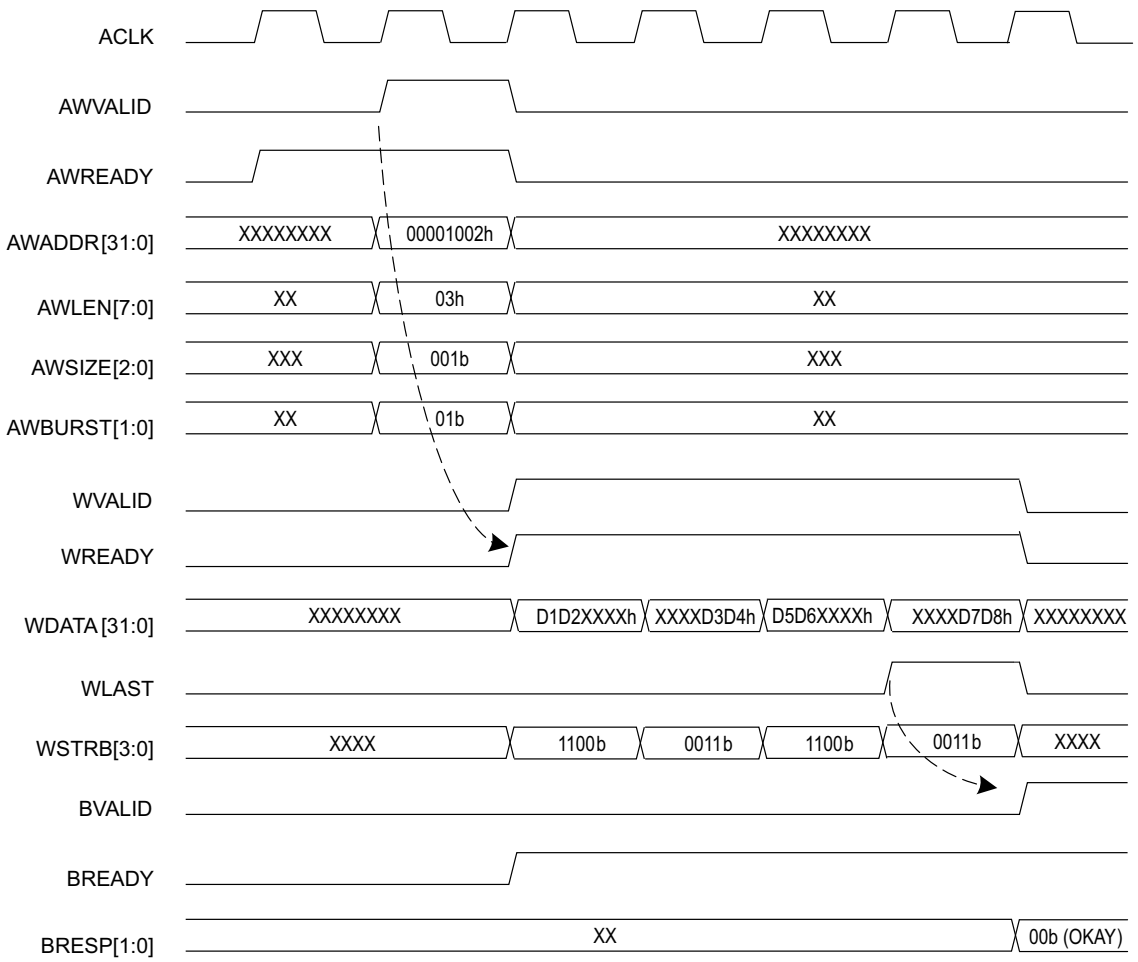

Figure 1-16: **AXI4 Narrow Read Burst Transactions**

For more details on AXI4 Narrow Transactions refer to the "Narrow transfers" section of the *AXI Protocol Specification* [Ref 1].

AXI4 Unaligned Transactions

Unaligned burst transfers occur when a 32-bit word burst size does not start on an address boundary that matches a word memory location. The starting memory address is permitted to be something other than 0x0h, 0x4h, 0x8h, etc. The example shown in Figure 1-17 illustrates an unaligned word burst transaction of 4 data beats, starting at address offset, 0x1002h.

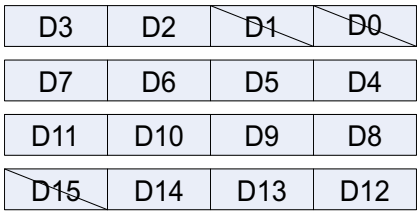

Figure 1-17: **AXI4 Unaligned Transactions**

For more details on AXI4 Narrow Transactions refer to the "about unaligned transfers" section of the *AXI Protocol Specification* [Ref 1].

Configurable Width and Depth

Table 1-5 provides supported width and depth for the Block Memory Generator core with AXI4 Interface.

Table 1-5: **Supported Width and Depth**

Operating Mode	Supported Memory Data Widths	Supported Minimum Memory Depth
AXI4 Memory Slave	32,64,128, 256	Supports minimum 4kB address range: Data Width — Minimum Depth 32 — 1024 64 — 512 128 — 256 256 — 128
AXI4-Lite Memory Slave	32,64	Supports minimum 4kB address range: Data Width — Minimum Depth 32 — 1024 64 — 512
AXI4 Peripheral Slave	8, 16, 32,64,128, 256	2
AXI4-Lite Peripheral Slave	8, 16, 32,64,	2

For Peripheral Slave configurations, there is no minimum requirement for the number of address bits used by Block Memory core. For Memory Slave configuration, AXI4 Block Memory slave has at least sufficient address bits to fully decode a 4kB address range.

For Peripheral Slave and AXI4-Lite Memory Slave configurations, the Block Memory Generator core with AXI4 Interface is not required to have low-order address bits to support decoding within the width of the system data bus and assumes that such low-order address bits have a default value of all zeros. For AXI4 Memory Slave configuration, the Block Memory Generator core with AXI4 Interface supports Narrow Transactions and performs low-order address bits decoding. For more details, see AXI4 Interface Block Memory Addressing.

AXI4 Interface Block Memory Addressing

AXI4 Interface Block Memory cores support 32-bit byte addressing. There is no minimum requirement for the number of address bits supplied by a master. Typically a master is expected to supply 32-bits of addressing. Table 1-6 illustrates some example settings to create a specific size of block RAM in the system.

Table 1-6: **AXI4 Interface Block Memory Generator Example Address Ranges**

Memory Width x Depth	Memory Size	Address Range Required	Example Base Address	Example Max Address	Block RAM Address
8 x 4096	4K	0x0000_0000 to 0x0000_0FFF	0xA000 0000	0xA000 0FFF	AXI_ADDR[11:0]
16 x 2048	4K	0x0000_0000 to 0x0000_0FFF	0xA000 0000	0xA000 0FFF	AXI_ADDR[11:1]
32 x 1024	4K	0x0000_0000 to 0x0000_0FFF	0xA000 0000	0xA000 0FFF	AXI_ADDR[11:2]
64 x 1024	8K	0x0000_0000 to 0x0000_1FFF	0x2400 0000	0x2400 1FFF	AXI_ADDR[12:3]
128 x 1024	16K	0x0000_0000 to 0x0000_3FFF	0x1F00 0000	0x1F00 3FFF	AXI_ADDR[13:4]
256 x 1024	32K	0x0000_0000 to 0x0000_7FFF	0x3000 0000	0x3000 7FFF	AXI_ADDR[14:5]

The Address Range of AXI Block Memory core must always start at zero. If the master has a different address bus width than that provided by the Block Memory Generator core with AXI4 interface, follow these guidelines:

- If the Master address is wider than the configured Address Range for AXI Block Memory core, the additional high-order address bits can be connected as is. AXI Block Memory core ignores these bits.
- If the Master address is narrower than 32-bits, the high-order address bits of the AXI Block Memory core can be left unconnected.

For more details on AXI4 Addressing refer to the “Master Addresses” and “Slave Addresses” section of the *AXI Protocol Specification* [Ref 1].

Throughput and Performance

To achieve 100% block RAM interface utilization of the Write port the following conditions must be satisfied.

- No single Write bursts
- The AXI Master should not apply back pressure on the Write response channel.

To achieve 100% block RAM interface utilization of the Read port the following conditions must be satisfied:

- The AXI Master should not apply back pressure on the Read data channel.

Selectable Port Aspect Ratios

The core currently supports only symmetric aspect ratios (that is, a 1:1 aspect ratio only).

Optional Output Register

The Output Register option is currently not supported.

Optional Pipeline Stages

Pipeline stages are currently not supported.

Memory Initialization Capability

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

Memory Initialization is not supported when ECC is enabled.

Applications

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

- **Single-port RAM**: Processor scratch RAM, look-up tables
- **Simple Dual-port RAM**: Content addressable memories, FIFOs
- **True Dual-port RAM**: Multi-processor storage
- **Single-port ROM**: Program code storage, initialization ROM
- **Dual-port ROM**: Single ROM shared between two processors/systems

Licensing and Ordering Information

This Xilinx® LogiCORE™ IP module is provided under the terms of the Xilinx Core License Agreement. The module is shipped as part of the Vivado® Design Suite. For full access to all core functionalities in simulation and in hardware, you must purchase a license for the core. Contact your local Xilinx sales representative for information about pricing and availability.

For more information, visit the Block Memory Generator product page.

Information about other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information on pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

This chapter includes details on performance and latency.

Performance

Performance and resource utilization for a Block Memory Generator core (BMG) varies depending on the configuration and features selected during core customization.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series.

Table 2-1: **Single Primitive Examples**

Memory Type	Options	Width x Depth	Performance (MHz)
Artix®-7 FPGAs			
True Dual-port RAM	No Output Registers	36x512	288
		9x2k	288
	Embedded Output Registers	36x512	382
		9x2k	382
Kintex®-7 FPGAs			
True Dual-port RAM	No Output Registers	36x512	344
		9x2k	368
	Embedded Output Registers	36x512	382
		9x2k	454
Virtex®-7 FPGAs			
True Dual-port RAM	No Output Registers	36x512	336
		9x2k	352
	Embedded Output Registers	36x512	454
		9x2k	454

Table 2-2: **Output Register Examples**

Memory Type	Width x Depth	Output Register Options	Performance (MHz)
Artix-7 FPGAs			
Single-port RAM	17x5k	No Primitive or Core	281
		Primitive	382
		Core	274
		Primitive and Core	382
True Dual-port RAM	17x5k	No Primitive or Core	274
		Primitive	274
		Core	281
		Primitive and Core	274
Kintex-7 FPGAs			
Single-port RAM	17x5k	No Primitive or Core	344
		Primitive	454
		Core	344
		Primitive and Core	454
True Dual-port RAM	17x5k	No Primitive or Core	336
		Primitive	336
		Core	
		Primitive and Core	344
Virtex-7 FPGAs			
Single-port RAM	17x5k	No Primitive or Core	336
		Primitive	454
		Core	344
		Primitive and Core	454
True Dual-port RAM	17x5k	No Primitive or Core	336
		Primitive	328
		Core	328
		Primitive and Core	336

Table 2-3: **Memory Algorithm Examples**

Memory Type	Width x Depth	Algorithm Type	Performance (MHz)
Artix-7 FPGAs			
Single-port RAM	17x5k	Minimum area	281
		Fixed Primitive using 18x1k block RAM	266
		Low power	258
	36x4k	Minimum area	312
		Fixed Primitive using 36x512 block RAM	258
		Low power	234
Kintex-7			
Single-port RAM	17x5k	Minimum area	344
		Fixed Primitive using 18x1k block RAM	328
		Low power	320
	36x4k	Minimum area	382
		Fixed Primitive using 36x512 block RAM	320
		Low power	304
Virtex-7 FPGAs			
Single-port RAM	17x5k	Minimum area	336
		Fixed Primitive using 18x1k block RAM	328
		Low power	312
	36x4k	Minimum area	368
		Fixed Primitive using 36x512 block RAM	304
		Low power	296

Table 2-4: **AXI4 Examples**

Memory Type	Options	Width x Depth	Performance (MHz)
Artix-7 FPGAs			
Simple Dual Port RAM	Memory Slave	32x1024	226
		64x512	234
	Peripheral Slave	32x1024	266
		64x512	274

Table 2-4: **AXI4 Examples *(Cont'd)***

Memory Type	Options	Width x Depth	Performance (MHz)
Kintex-7 FPGAs			
Simple Dual Port RAM	Memory Slave	32x1024	344
		64x512	328
	Peripheral Slave	32x1024	360
		64x512	352
Virtex-7 FPGAs			
Simple Dual Port RAM	Memory Slave	32x1024	320
		64x512	274
	Peripheral Slave	32x1024	360
		64x512	344

Latency

The latency of output signals of BMG varies for different configurations. See Optional Output Registers, Optional Pipeline Stages, and Memory Output Flow Control in Chapter 3 for more details.

Resource Utilization

The following tables show resource utilization data and maximum performance values for several sample BMG configurations.

Native Block Memory Generator

Note: Benchmarking data for Artix-7 and Kintex-7 devices will be available in future release.

The following tables provide examples of actual resource utilization and performance for Native Block Memory Generator core implementations. Each section highlights the effects of a specific feature on resource utilization and performance. The actual results obtained depends on core parameter selections, such as algorithm, optional output registers, and memory size, as well as surrounding logic and packing density.

Benchmarks were taken using a design targeting a Virtex-7 FPGA in the -1 speed grade (XC7VX550T-FFG1927-1). All benchmarks were obtained using the Vivado Design Suite.

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. Xilinx recommends you to register your inputs to the core for better performance.

The following examples highlight the use of embedded registers in 7 series devices, and the subsequent performance improvement that might result.

Single Primitive

The Block Memory Generator core does not add additional logic if the memory can be implemented in a single Block RAM primitive. Table 2-5 through Table 2-7 define performance data for single-primitive memories.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series devices.

Table 2-5: **Single Primitive Examples - Virtex-7 FPGAs**

Memory Type	Options	Width x Depth	Resource Utilization				
			Block RAMs		Shift Regs	FFs	LUTs(1)
			36K	18K			
True Dual-port RAM	No Output Registers	36x512	1	0	0	0	0
		9x2k	1	0	0	0	0
	Embedded Output Registers	36x512	1	0	0	0	0
		9x2k	1	0	0	0	0

Notes:
1. LUTs are reported as the number of 6-input LUTs, and do not reflect the number of LUTs used as a route-through.

Output Registers

The Block Memory Generator core 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 might improve performance notably.

In Zynq-7000 and 7 series architectures, the embedded block RAM can be utilized, reducing the FPGA general interconnect resources required to create the registers.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series devices.

Table 2-6: **Virtex-7 Device Output Register Examples**

Memory Type	Width x Depth	Output Register Options	Block RAM			FFs	LUTs[1]
			36K	16K	8K		
Single-port RAM	17x5k		1	3	0	3	18
		Primitive	1	3	0	6	18
		Core	1	3	0	20	18
		Primitive, Core	1	3	0	23	18
True Dual-port RAM	17x5k		1	3	0	6	36
		Primitive	1	3	0	9	36
		Core	1	3	0	23	36
		Primitive, Core	1	3	0	26	36

Notes:
1. LUTs are reported as the number of 6-input LUTs, and do not reflect the number of LUTs used as a route-through.

Aspect Ratios

The Block Memory Generator core selectable port and data width aspect ratios might 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.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series devices.

Table 2-7: **Virtex-7 Device Aspect Ratio**

Memory Type	Width x Depth	Data Width Aspect Ratio	Block RAMs			FFs	LUTs[1]
			36K	16K	8K		
Single-port RAM	17x5k	1:1	2	3	0	6	36
		1:8[2]	8	1	0	0	0

Notes:
1. LUTs are reported as the number of 6-input LUTs, and do not reflect the number of LUTs used as a route-through.
2. Read port is 136x640; Write port is 17x5k

Algorithm

The differences between the minimum area, low power and fixed primitive algorithms are discussed in detail in Selectable Memory Algorithm, page 7. Table 2-8 shows examples of the resource utilization and the performance difference between them for two selected configurations for Virtex-7 FPGA architectures.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series devices.

Table 2-8: **Memory Algorithm Examples Virtex-7 Devices**

Memory Type	Width x Depth	Algorithm Type	Block RAM			FFs	LUTs[1]
			36K	16K	8K		
Single-port RAM	17x5k	Minimum area	1	3	0	3	18
		Fixed Primitive using 18x1k block RAM	2	1	0	3	20
		Low power	0	5	0	3	38
	36x4k	Minimum area	4	0	0	0	0
		Fixed Primitive using 36x512 block RAM	4	0	0	2	40
		Low power	4	0	0	3	77

Notes:

1. LUTs are reported as the number of 6-input LUTs, and do not reflect the number of LUTs used as a route-through.

Block Memory Generator with AXI4 Interface

Table 2-9 shows the resource utilization and performance data for a BMG core using the AXI4 interface. Benchmarks were taken using a design targeting a Virtex-7 FPGA in the -1 speed grade (XC7VX550T-FFG1927-1). All benchmarks were obtained using the Vivado Design Suite.

In the benchmark designs, the core was encased in a wrapper with input and output registers to remove the effects of I/O delays from the results. Performance might vary depending on the design.

Note: Performance for UltraScale™ devices is expected to be similar to results from 7 series devices.

Table 2-9: **AXI4 Block Memory Generator Virtex-7 FPGA**

Memory Type	Options	Width X Depth	Resource Utilization					
			Block RAMs			FFs	LUTs	Occupied Slices
			36K	16K	8K			
Simple Dual-Port RAM	Memory Slave	32x1024	1	0	0	102	163	48
		64x512	1	0	0	103	174	69
	Peripheral Slave	32x1024	1	0	0	50	100	34
		64x512	1	0	0	48	99	35

Port Descriptions

Native Block Memory Generator Signals

Table 2-10 provides a description of the Block Memory Generator core signals. You can select the widths of the data ports (`dina`, `douta`, `dinb`, and `doutb`) in Vivado IDE. The address port (`addra` and `addrb`) widths are determined by the memory depth with respect to each port. The Write enable ports (`wea` and `web`) are buses 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 Vivado IDE.

Table 2-10: **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 through port A. Available in all RAM configurations.
douta	Output	**Port A Data Output**: Data output from Read operations through port A. Available in all configurations except Simple Dual-port RAM.
ena	Input	**Port A Clock Enable**: Enables Read, Write, and reset operations through port A. Optional in all configurations.
wea	Input	**Port A Write Enable**: Enables Write operations through port A. Available in all RAM configurations.
rsta	Input	**Port A 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 of port A. Optional in all configurations with port A 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 through port B. Available in True Dual-port RAM configurations.
doutb	Output	**Port B Data Output**: Data output from Read operations through Port B. Available in dual-port configurations.
enb	Input	**Port B Clock Enable**: Enables Read, Write, and reset operations through Port B. Optional in dual-port configurations.
web	Input	**Port B Write Enable**: Enables Write operations through Port B. Available in Dual-port RAM configurations.

Table 2-10: **Core Signal Pinout *(Cont'd)***

Name	Direction	Description
rstb	Input	**Port B Set/Reset**: Resets the Port B memory output latch or output register. Optional in all configurations.
regceb	Input	**Port B Register Enable**: Enables the last output register of port B. Optional in dual-port configurations with port B 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.
injectsbiterr	Input	**Inject Single-Bit Error**: Available only for Zynq-7000 and 7 series ECC configurations.
injectdbiterr	Input	**Inject Double-Bit Error**: Available only for Zynq-7000 and 7 series ECC configurations.
rdaddrecc	Output	**Read Address for ECC Error output**: Available only for Zynq-7000 and 7 series ECC configurations.
eccpipece	Input	**ECC Pipe Line Register Clock Enable**: Available only for UltraScale architecture-based devices.
sleep	Input	**Dynamic Power Saving**: If sleep pin is High , the Block Memory Generator core is in power saving mode. Available only for UltraScale architecture-based devices.

AXI4 Interface Block Memory Generator Signals

AXI4 Interface - Global Signals

Table 2-11: **AXI4 or AXI4-Lite- Global Interface Signals**

Name	Direction	Description
AXI4 or AXI4-Lite Global Interface Signals		
s_aclk	Input	**Global Slave Interface Clock**: All signals are sampled on the rising edge of this clock.
s_aresetn	Input	**Global Reset**: This signal is active-Low.

AXI4-Interface Signals

Table 2-12: **AXI4 Write Channel Interface Signals**

Name	Direction	Description
AXI4 Write Address Channel Interface Signals		
s_axi_awid[m:0]	Input	**Write Address ID**: This signal is the identification tag for the Write address group of signals. Write address ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration.

Table 2-12: **AXI4 Write Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_awaddr[31:0]	Input	**Write Address**: The Write address bus gives the address of the first transfer in a Write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
s_axi_awlen[7:0]	Input	**Burst Length**: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
s_axi_awsize[2:0]	Input	**Burst Size**: This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update. Burst size should always be less than or equal to the width of the Write Data. Burst Size input is not supported for Peripheral Slave configuration.
s_axi_awburst[1:0]	Input	**Burst Type**: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. Burst type for Memory Slave configuration could be either incremental or wrap. Burst type input is not supported for Peripheral Slave configuration, Burst type for Peripheral Slave is always internally set to incremental.
s_axi_awvalid	Input	**Write Address Valid**: This signal indicates that valid Write address and control information are available: • 1 = address and control information available. • 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, awready, goes High.
s_axi_awready	Output	**Write Address Ready**: This signal indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready • 0 = Slave not ready
AXI4 Write Data Channel Interface Signals		
s_axi_wdata[m–1:0]	Input	**Write Data**: For Memory Slave configurations, the Write data bus can be 32, 64, 128, or 256 bits wide. For Peripheral Slave configurations, the Write data bus can be 8, 16, 32, 64, 128, or 256 bits wide.
s_axi_wstrb[m/8–1:0]	Input	**Write Strobes**: This signal indicates which byte lanes to update in memory. There is one Write strobe for each eight bits of the Write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)].
s_axi_wlast	Input	**Write Last**: This signal indicates the last transfer in a Write burst.
s_axi_wvalid	Input	**Write Valid**: This signal indicates that valid Write data and strobes are available: • 1 = Write data and strobes available • 0 = Write data and strobes not available

Table 2-12: **AXI4 Write Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_wready	Output	**Write Ready**: This signal indicates that the slave can accept the Write data: • 1 = slave ready • 0 = slave not ready
AXI4 Write Response Channel Interface Signals		
s_axi_bid[m:0]	Output	**Response ID**: The identification tag of the Write response. The BID value must match the awid value of the Write transaction to which the slave is responding. Response ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Response ID can be 1 to 16 bits wide.
s_axi_bresp[1:0]	Output	**Write Response**: This signal indicates the status of the Write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. Write response is always set to OKAY. Write response is generated only when AXI4 ID is enabled for Memory Slave. Write response is not supported for Peripheral Slave configuration.
s_axi_bvalid	Output	**Write Response Valid**: This signal indicates that a valid Write response is available: • 1 = Write response available • 0 = Write response not available
s_axi_bready	Input	**Response Ready**: This signal indicates that the master can accept the response information. • 1 = Master ready • 0 = Master not ready

Table 2-13: **AXI4 Read Channel Interface Signals**

Name	Direction	Description
AXI4 Read Address Channel Interface Signals		
s_axi_arid[m:0]	Input	**Read Address ID**: This signal is the identification tag for the Read address group of signals. Read address ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Read address ID can be 1 to 16 bits wide.
s_axi_araddr[31:0]	Input	**Read Address**: The Read address bus gives the initial address of a Read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
s_axi_arlen[7:0]	Input	**Burst Length**: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.

Table 2-13: **AXI4 Read Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_arsize[2:0]	Input	**Burst Size**: This signal indicates the size of each transfer in the burst. Burst size should always be less than or equal to the width of the Read Data. Burst Size input is not supported for Peripheral Slave configuration.
s_axi_arburst[1:0]	Input	**Burst Type**: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated. Burst type for Memory Slave configuration could be either incremental or wrap. Burst type input is not supported for Peripheral Slave configuration, Burst type for Peripheral Slave is always internally set to incremental.
s_axi_arvalid	Input	**Read Address Valid**: This signal indicates, when High, that the Read address and control information is valid and remains stable until the address acknowledge signal, arready, is High. • 1 = address and control information valid • 0 = address and control information not valid
s_axi_arready	Output	**Read Address Ready**: This signal indicates that the slave is ready to accept an address and associated control signals: • 1 = slave ready • 0 = slave not ready
AXI4 Read Data Channel Interface Signals		
s_axi_rid[m:0]	Output	**Read ID Tag**: This signal is the ID tag of the Read data group of signals. The RID value is generated by the slave and must match the ARID value of the Read transaction to which it is responding. Read ID tag is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Read ID can be 1 to 16 bits wide.
s_axi_rdata[m–1:0]	Output	**Read Data**: For Memory Slave configurations, the Read data bus can be 32, 64, 128, or 256 bits wide. For Peripheral Slave configurations, the Read data bus can be 8, 16, 32, 64, 128, or 256 bits wide.
s_axi_rresp[1:0]	Output	**Read Response**: This signal indicates the status of the Read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. Read response is always set to OKAY. Read response is generated only when AXI4 ID is enabled for Memory Slave. Read response is not supported for Peripheral Slave configuration.
s_axi_rlast	Output	**Read Last**: This signal indicates the last transfer in a Read burst.

Table 2-13: **AXI4 Read Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_rvalid	Output	**Read Valid**: This signal indicates that the required Read data is available and the Read transfer can complete: • 1 = Read data available • 0 = Read data not available
s_axi_rready	Input	**Read Ready**: This signal indicates that the master can accept the Read data and response information: • 1= Master ready • 0 = Master not ready

AXI4-Lite Interface Signals

Table 2-14: **AXI4-Lite Write Channel Interface Signals**

Name	Direction	Description
AXI4-Lite Write Address Channel Interface Signals		
s_axi_awaddr[31:0]	Input	**Write Address**: The Write address bus gives the address of the first transfer in a Write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
s_axi_awvalid	Input	**Write Address Valid**: This signal indicates that valid Write address and control information are available: • 1 = address and control information available • 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, awready, goes High
s_axi_awready	Output	**Write Address Ready**: This signal indicates that the slave is ready to accept an address and associated control signals: • 1 = slave ready • 0 = slave not ready
s_axi_awid[m:0]	Input	**Write Address ID**: This signal is the identification tag for the Write address group of signals Write address ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Write address ID can be 1 to 16 bits wide.
AXI4-Lite Write Data Channel Interface Signals		
s_axi_wdata[m–1:0]	Input	**Write Data**: For Memory Slave configurations, the Write data bus can be 32 or 64 bits wide. For Peripheral Slave configurations, the Write data bus can be 8, 16, 32 or 64 bits wide.
s_axi_wstrb[m/8–1:0]	Input	**Write Strobes**: This signal indicates which byte lanes to update in memory. There is one Write strobe for each eight bits of the Write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)].

Table 2-14: **AXI4-Lite Write Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_wvalid	Input	**Write Valid**: This signal indicates that valid Write data and strobes are available: 1 = Write data and strobes available 0 = Write data and strobes not available
s_axi_wready	Output	**Write Ready**: This signal indicates that the slave can accept the Write data: • 1 = Slave ready • 0 = Slave not ready
AXI4-Lite Write Response Channel Interface Signals		
s_axi_bvalid	Output	**Write Response Valid**: This signal indicates that a valid Write response is available: • 1 = Write response available • 0 = Write response not available
s_axi_bready	Input	**Response Ready**: This signal indicates that the master can accept the response information. • 1 = Master ready • 0 = Master not ready
s_axi_bid[m:0]	Output	**Response ID**: The identification tag of the Write response. The **BID** value must match the **awid** value of the Write transaction to which the slave is responding. Response ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Response ID can be 1 to 16 bits wide.
s_axi_bresp[1:0]	Output	**Write Response**: This signal indicates the status of the Write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. Write response is always set to OKAY. Write response is generated only when AXI4 ID is enabled for Memory Slave. Write response is not supported for Peripheral Slave configuration.

Table 2-15: **AXI4-Lite Read Channel Interface Signals**

Name	Direction	Description
AXI4-Lite Read Address Channel Interface Signals		
s_axi_araddr[31:0]	Input	**Read Address**: The Read address bus gives the initial address of a Read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
s_axi_arid[m:0]	Input	**Read Address ID**: This signal is the identification tag for the Read address group of signals. Read address ID is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Read address ID can be 1 to 16 bits wide.

Table 2-15: **AXI4-Lite Read Channel Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_arvalid	Input	**Read Address Valid**: This signal indicates, when High, that the Read address and control information is valid and remains stable until the address acknowledge signal, aready, is High. • 1 = Address and control information valid • 0 = Address and control information not valid
s_axi_arready	Output	**Read Address Ready**: This signal indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready • 0 = Slave not ready
AXI4-Lite Read Data Channel Interface Signals		
s_axi_rdata[m–1:0]	Output	**Read Data**: For Memory Slave configurations, the Read data bus can be 32 or 64 bits wide. For Peripheral Slave configurations, the Read data bus can be 8, 16, 32 or 64 bits wide.
s_axi_rresp[1:0]	Output	**Read Response**: This signal indicates the status of the Read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR. Read response is always set to OKAY. Read response is generated only when AXI4 ID is enabled for Memory Slave. Read response is not supported for Peripheral Slave configuration.
s_axi_rid[m:0]	Output	**Read ID Tag**: This signal is the ID tag of the Read data group of signals. The RID value is generated by the slave and must match the ARID value of the Read transaction to which it is responding. Read ID tag is optional for Memory Slave configuration and is not supported for Peripheral Slave configuration. Read ID tag can be 1 to 16 bits wide.
s_axi_rvalid	Output	**Read Valid**: This signal indicates that the required Read data is available and the Read transfer can complete: • 1 = Read data available • 0 = Read data not available
s_axi_rready	Input	**Read Ready**: This signal indicates that the master can accept the Read data and response information: • 1 = Master ready • 0 = Master not ready

Designing with the Core

This chapter describes the steps required to turn a Block Memory Generator core into a fully functioning design integrated with the user application logic. It is important to note that depending on the configuration of the BMG core, only a subset of the implementation details provided are applicable. The guidelines in this chapter provide details about creating the most successful implementation of the core.

General Design Guidelines

The Block Memory Generator core 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 Vivado® Design Suite IP Catalog, users can configure the core and rapidly generate a highly optimized custom memory solution.

Memory Type

The Block Memory Generator core creates five memory types: Single-port RAM, Simple Dual-port RAM, True Dual-port RAM, Single-port ROM, and Dual-port ROM. Figure 3-1 through Figure 3-5 illustrate the signals available for each type. Optional pins are displayed 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 use the special Simple Dual-port RAM primitive in 7 series devices, which can save as much as fifty percent of the block RAM resources for memories 512 words deep or fewer. The Single-port ROM allows Read access to the memory space through a single port, as illustrated in Figure 3-1.

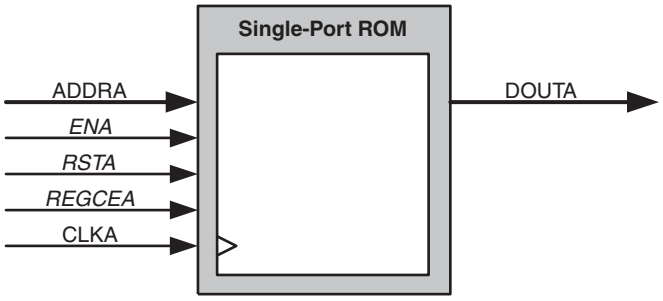

Figure 3-1: **Single-port ROM**

The Dual-port ROM allows Read access to the memory space through two ports, as shown in Figure 3-2.

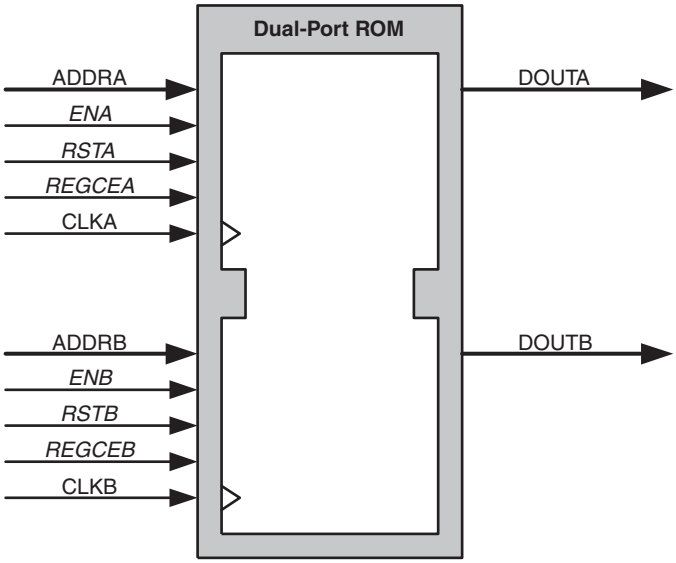

Figure 3-2: **Dual-port ROM**

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

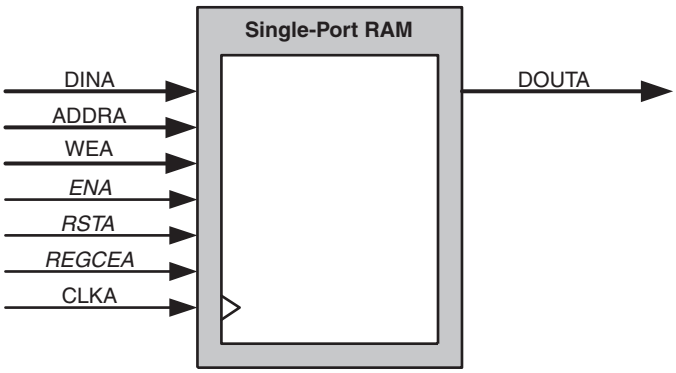

Figure 3-3: **Single-port RAM**

The Simple Dual-port RAM provides two ports, A and B, as illustrated in Figure 3-4. Write access to the memory is allowed through port A, and Read access is allowed through port B.

IMPORTANT: *For 7 series family architectures, though the diagram below shows port A as write port and port B as Read Port, in the FPGA the BRAM primitives perform Read access* through *port A and Write access* through *port B.*

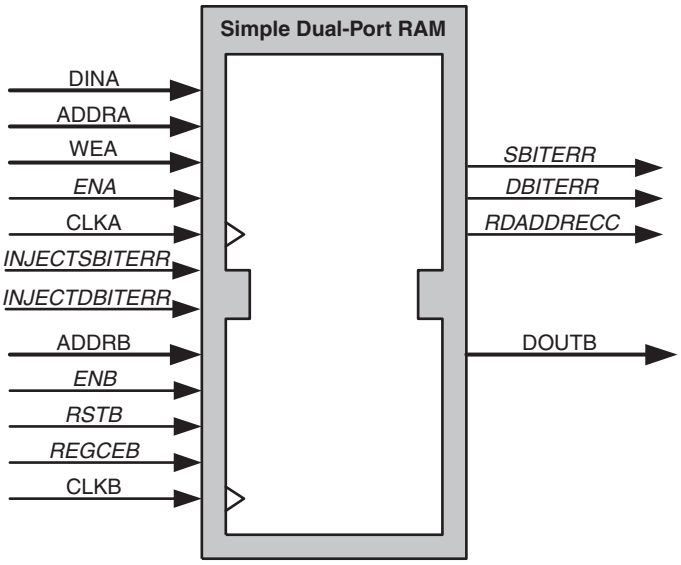

Figure 3-4: **Simple Dual-port RAM**

The True Dual-port RAM provides two ports, A and B, as illustrated in Figure 3-5. Read and Write accesses to the memory are allowed on either port.

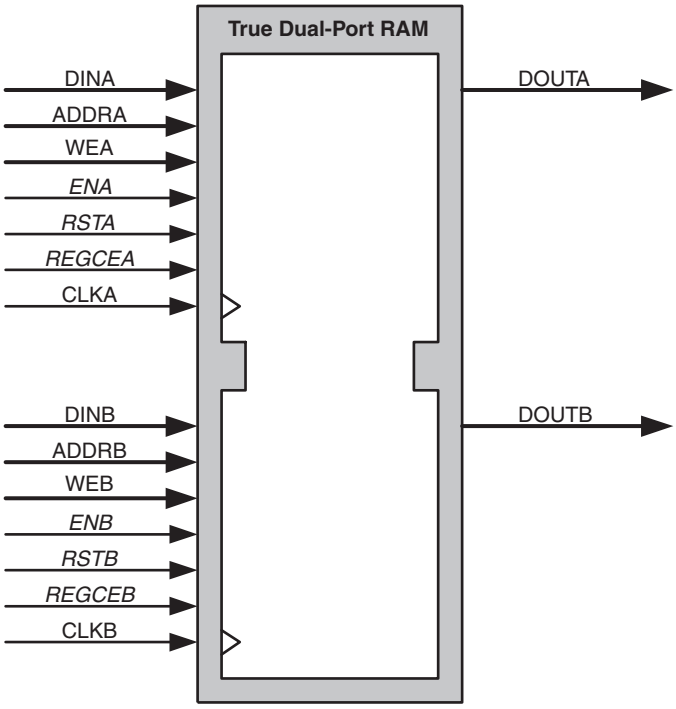

Figure 3-5: **True Dual-port RAM**

Selectable Memory Algorithm

The Block Memory Generator core arranges block RAM primitives according to one of three algorithms: the minimum area algorithm, the low power 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 3-6 shows two examples of memories built using the minimum area algorithm.

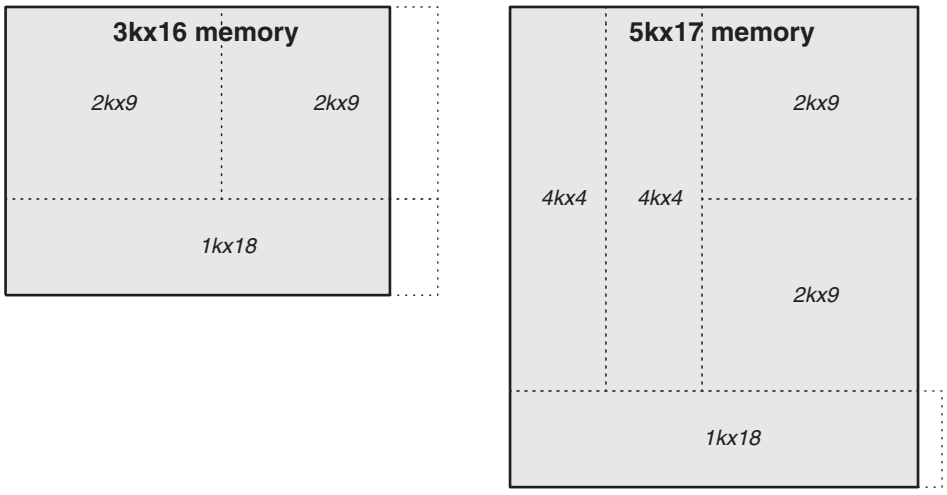

Figure 3-6: **Examples of the Minimum Area Algorithm**

In the first example, a 3kx16 memory is implemented using three block RAMs. While it might 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.

In 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.

Low Power Algorithm

The low power algorithm provides a solution that minimizes the number of primitives enabled during a Read or Write operation. This algorithm is not optimized for area and might use more block RAMs and multiplexers than the minimum area algorithm. Figure 3-7 shows two examples of memories built using the low power algorithm.

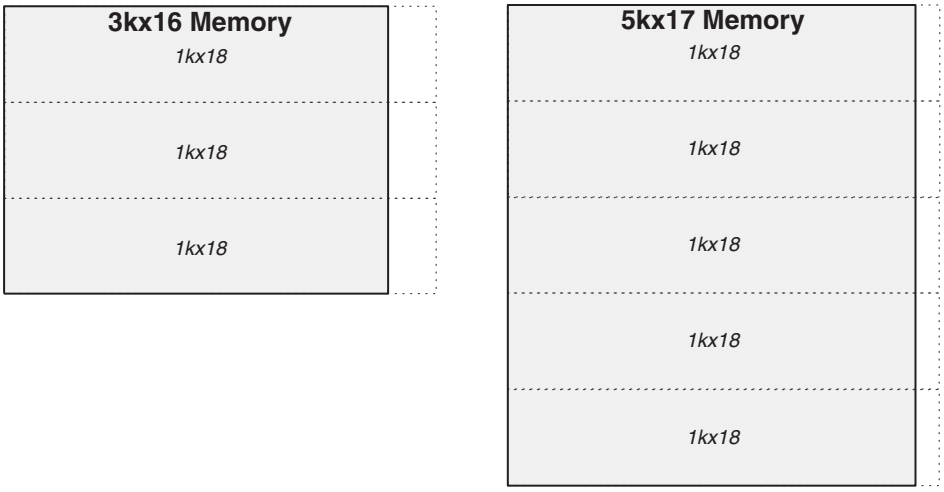

Figure 3-7: **Examples of the Low Power Algorithm**

Fixed Primitive Algorithm

The fixed primitive algorithm allows you to select a single block RAM primitive type. The core builds the memory by concatenating this single primitive type in width and depth. It is useful in systems that require a fixed primitive type. Figure 3-8 depicts two 3kx16 memories, one built using the 2kx9 primitive type, the other built using the 4kx4 primitive type.

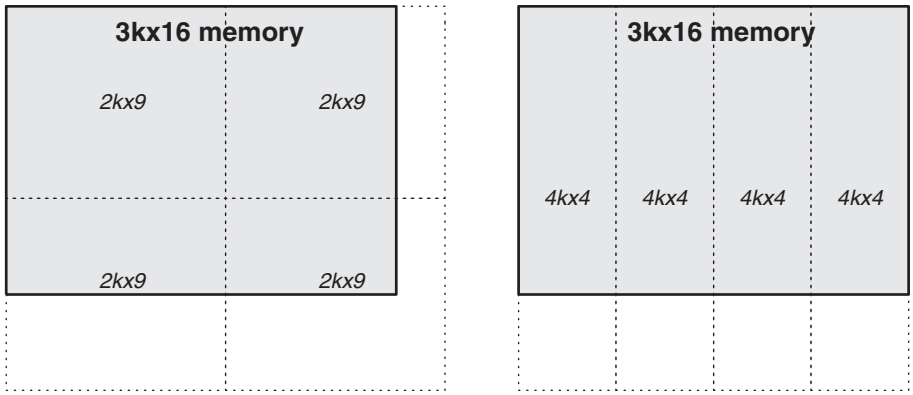

Figure 3-8: **Examples of the Fixed Primitive Algorithm**

Note that both implementations use four block RAMs, and that some of the resources utilized extend beyond the usable memory space. You decide which primitive type is best for your 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 enhance performance. Table 3-1 shows the

primitives used to construct a memory given the specified architecture and primitive selection.

Table 3-1: **Memory Primitives Used Based on Architecture (Supported in Native BMG)**

Architecture	Primitive Selection	Primitives Used
Zynq®-7000	16kx1	64kx1, 32kx1, 16kx1
	8kx2	16kx2, 8kx2
	4kx4	4kx4, 8kx4
	2kx9	2kx9, 4kx9
	1kx18	1kx18, 2kx18
	512x36	512x36, 1kx36
	256x72	256x72, 512x72(SP RAM/ROM and SDP configurations only)
Artix®-7	16kx1	64kx1, 32kx1, 16kx1
	8kx2	16kx2, 8kx2
	4kx4	4kx4, 8kx4
	2kx9	2kx9, 4kx9
	1kx18	1kx18, 2kx18
	512x36	512x36, 1kx36
	256x72	256x72, 512x72(SP RAM/ROM and SDP configurations only)
Kintex®-7	16kx1	64kx1, 32kx1, 16kx1
	8kx2	16kx2, 8kx2
	4kx4	4kx4, 8kx4
	2kx9	2kx9, 4kx9
	1kx18	1kx18, 2kx18
	512x36	512x36, 1kx36
	256x72	256x72, 512x72(SP RAM/ROM and SDP configurations only)
Virtex®-7	16kx1	64kx1, 32kx1, 16kx1
	8kx2	16kx2, 8kx2
	4kx4	4kx4, 8kx4
	2kx9	2kx9, 4kx9
	1kx18	1kx18, 2kx18
	512x36	512x36, 1kx36
	256x72	256x72, 512x72(SP RAM/ROM and SDP configurations only)

Notes:

1. See additional memory collision restrictions in Collision Behavior, page 54.

When using data-width aspect ratios, the primitive type dimensions are chosen with respect to the A port Write width. Note that the primitive selection might limit port aspect ratios as

described in Aspect Ratio Limitations, page 52. When using the byte Write feature, only the 2kx9, 1kx18, and 512kx36 primitive choices are available.

Selectable Width and Depth

The Block Memory Generator core generates memories with widths from 1 to 4608 bits, and with depths of two 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 can 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 in the present or following clock cycles depending upon the output register stages of the core.

Operating Mode

The operating mode for each port determines the relationship between the Write and Read interfaces for that port. Port A and port B can be configured independently with any one of three Write modes: Write First Mode, Read First Mode, or No Change Mode. These operating modes are described in the sections that follow.

The operating modes have an effect on the relationship between the A and B ports when the A and B port addresses have a collision. For detailed information about collision behavior, see Collision Behavior, page 54. For more information about operating modes, see the block RAM section of the user guide specific to the 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 3-9. This transparent mode offers the flexibility of using the data output bus during a Write operation on the same port.

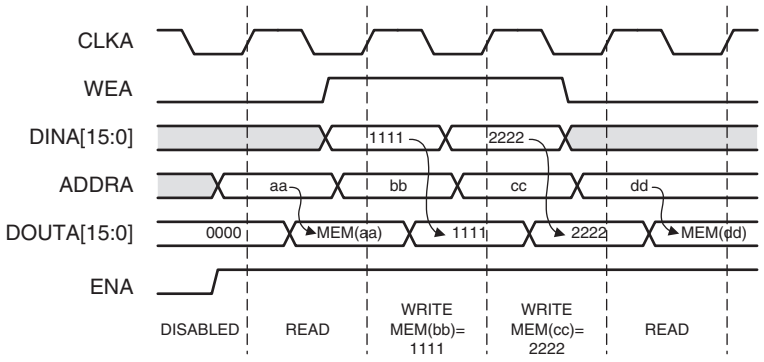

Figure 3-9: **Write First Mode Example**

Note: The WRITE_FIRST operation is affected by the optional byte-Write feature. It is also affected by the optional Read-to-Write aspect ratio feature. For detailed information, see Write

First Mode Considerations, page 54.

- **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 3-10.

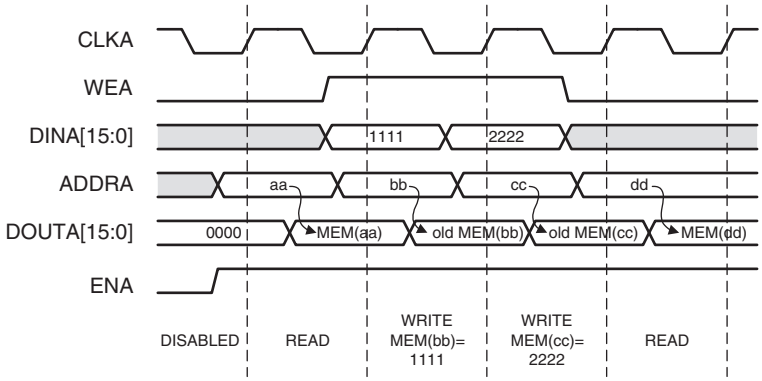

Figure 3-10: **Read First Mode Example**

- **No Change Mode**: In NO_CHANGE mode, the output latches remain unchanged during a Write operation. As shown in Figure 3-11, the data output is still the previous Read data and is unaffected by a Write operation on the same port.

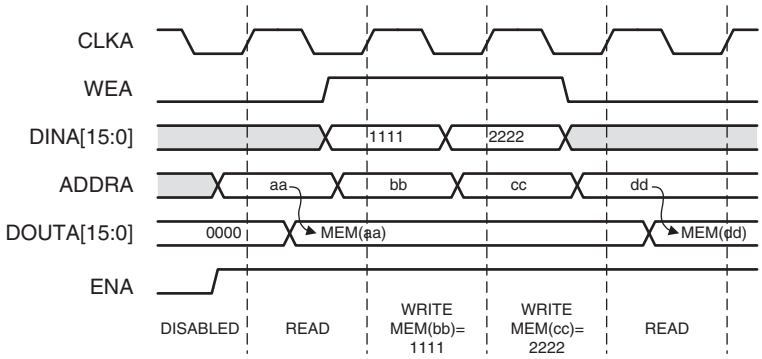

Figure 3-11: **No Change Mode Example**

Data Width Aspect Ratios

The Block Memory Generator core 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 the following section. All four data buses (`dina`, `douta`, `dinb`, and `doutb`) can have different widths, as described in Read-to-Write Aspect Ratios, page 51.

The limitations of the data width aspect ratio feature (some of which are imposed by other optional features) are described in Aspect Ratio Limitations, page 52. The Vivado IP Catalog GUI ensures only valid aspect ratios are selected.

Port Aspect Ratios

The Block Memory Generator core 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 3-12.

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 3-12. 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.

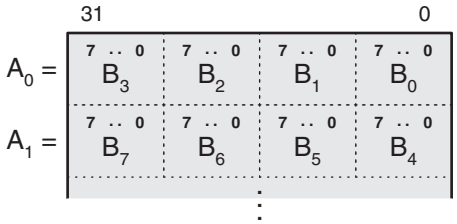

Figure 3-12: **Port Aspect Ratio Example Memory Map**

Read-to-Write Aspect Ratios

When implementing RAMs, the Block Memory Generator core allows Read and Write aspect ratios on either port. On each port A and port B, the Read to Write data width ratio of that port 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.

Because the Read and Write interfaces of each port can differ, it is possible for all four data buses (`dina`, `douta`, `dinb`, and `doutb`) of True Dual-port RAMs to have a different width. 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 4608 bits.

If the Read and Write data widths on a port are different, the memory depth is different with respect to Read and Write accesses. For example, if the Read interface of port A is twice as wide as the Write interface, then it is also half as deep. The ratio of the widths is *always* the inverse of the ratio of the depths. Because a single address bus is used for both the Write and Read interface of a port, the address bus must be large enough to address the deeper of the two depths. For the shallower interface, the least significant bits of the address bus are ignored. The data words are arranged in little-endian format, as illustrated in Figure 3-13.

Read-to-Write Aspect Ratio Example

Consider a True Dual-port RAM of 64x512, which is the port A Write width and depth. Table 3-2 defines the four data-port widths and their respective depths for this example.

Table 3-2: **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 use 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 use the entire `addrb` bus, while Write operations ignore the least significant 3 bits.

The memory map in Figure 3-13 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.

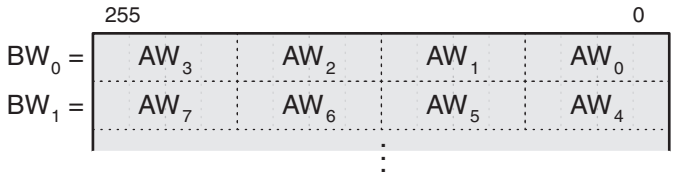

Figure 3-13: **Read-to-Write Aspect Ratio Example Memory Map**

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 example above, 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 can be wider than 4096 bits, and no two data widths can have a ratio greater than 32:1. However, the following 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 might be no more than 8 times (32:4) larger than port A Write width and no less than 4 times (1:4) smaller.

Byte-Writes

The Block Memory Generator core provides byte-Write support. 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]` (`wea` or `web`) 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 are 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 might be used in conjunction with the data width aspect ratios, which can limit the choice of data widths as described in Data Width Aspect Ratios, page 50. However, it might not be used with the NO_CHANGE operating mode. This is because if a memory configuration uses multiple primitives in width, and only one primitive is being written to (using partial byte writes), then the NO_CHANGE mode only applies to that single primitive. The NO_CHANGE mode does not apply to the other primitives that are not being written to, so these primitives can still be read. The byte-Write feature also affects the operation of WRITE_FIRST mode, as described in Write First Mode Considerations, page 54.

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 3-14 illustrates the use of byte-writes, and shows the contents of the RAM at address 0. Assume all memory locations are initialized to 0.

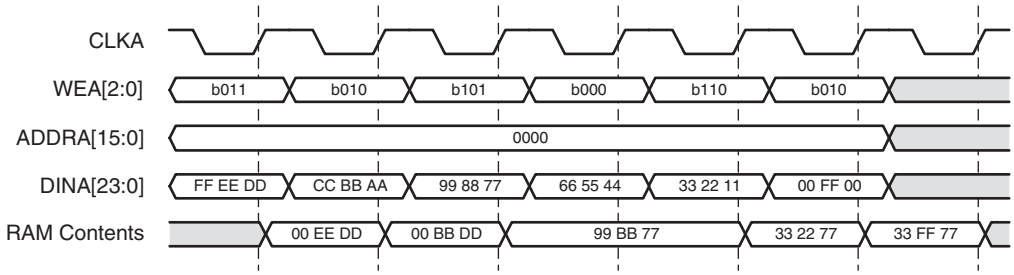

Figure 3-14: **Byte-Write Example**

Write First Mode Considerations

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 or the Read-to-Write aspect ratio feature, the output of the memory cannot be guaranteed.

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.

Collisions and Asynchronous Clocks: General Guidelines

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: General Guidelines

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 3-15 illustrates this case. Assume `addra` = `addrb` = 0.

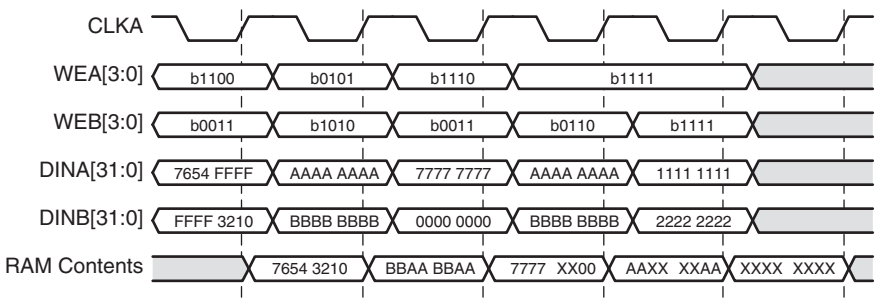

Figure 3-15: **Write-Write Collision Example**

- **Synchronous Write-Read Collisions**: A synchronous Write-Read collision might 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 updated bytes are invalid on the Read port output.

Figure 3-16 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.

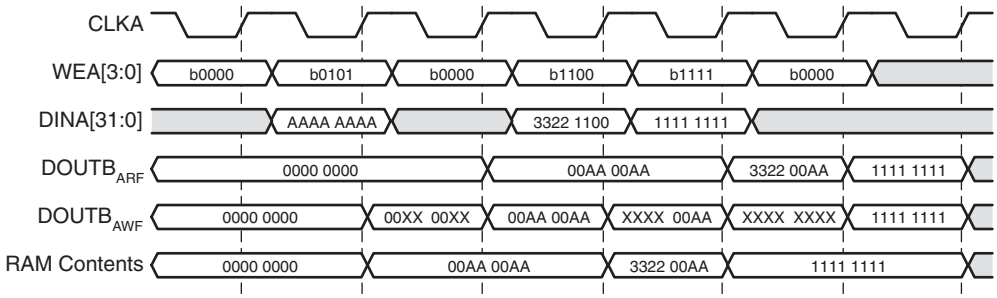

Figure 3-16: **Write-Read Collision Example**

Collisions and Simple Dual-port RAM

For Simple Dual-port RAM, the operating modes READ_FIRST, WRITE_FIRST and NO_CHANGE are available irrespective of clocking.

IMPORTANT: *The Operating modes are not available for selection when the interface type is set to AXI.*

The Simple Dual-port RAM is like a true dual-port RAM where only the Write interface of the A port and the Read interface of B port are connected. The operating modes define the Write-to-Read relationship of the A or B ports, and only impact the relationship between A and B ports during an address collision.

For Synchronous Clocking and during a collision, the Write mode of port A can be configured so that a Read operation on port B either produces data (acting like READ_FIRST), or produces undefined data (Xs). For this reason, it is always advised to use READ_FIRST when configured as a Simple Dual-port RAM. For asynchronous clocking, Xilinx recommends setting the Write mode of Port A to WRITE_FIRST for collision safety. For detailed information about this behavior, see Collision Behavior, page 54.

For 7 series devices, the selected operating mode is passed to the block RAM when the RAM_MODE is set to TDP. For the primitives with RAM_MODE set to SDP, the write mode is READ_FIRST for synchronous clocking and WRITE_FIRST for asynchronous clocking.

For UltraScale architecture-based devices, there are no restrictions and the selected operating mode is always passed to the block RAM primitive irrespective of clocking.

Additional Memory Collision Restrictions: Address Space Overlap

The 7 series FPGA block RAM memories have additional collision restrictions in the following configurations:

- When configured as True Dual Port (TDP)
- When CLKA (port A) and CLKB (port B) are Asynchronous
- In applications that perform a simultaneous Read and Write
- When either port A, port B, or both ports are configured with Write Mode configured as READ_FIRST

When using TDP Memory with Write Mode = READ_FIRST (TDP-RF mode) in conjunction with asynchronous clocking, see the "Conflict Avoidance" section of the *7 Series FPGAs Memory Resources User Guide* [Ref 11].

Optional Output Registers

The Block Memory Generator core allows optional output registers, which might improve the performance of the core. You might 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. Each of the two optional register stages can be chosen for port A and port B separately. Note that each optional register stage used adds an additional clock cycle of latency to the Read operation.

The Register Port [A|B] Output of Memory Primitives option might be implemented using the embedded block RAM registers, requiring no further FPGA resources. All other register stages are implemented in FPGA general interconnect. Figure 3-17 shows an example of memory that has been configured using both output register stages for one of the ports.

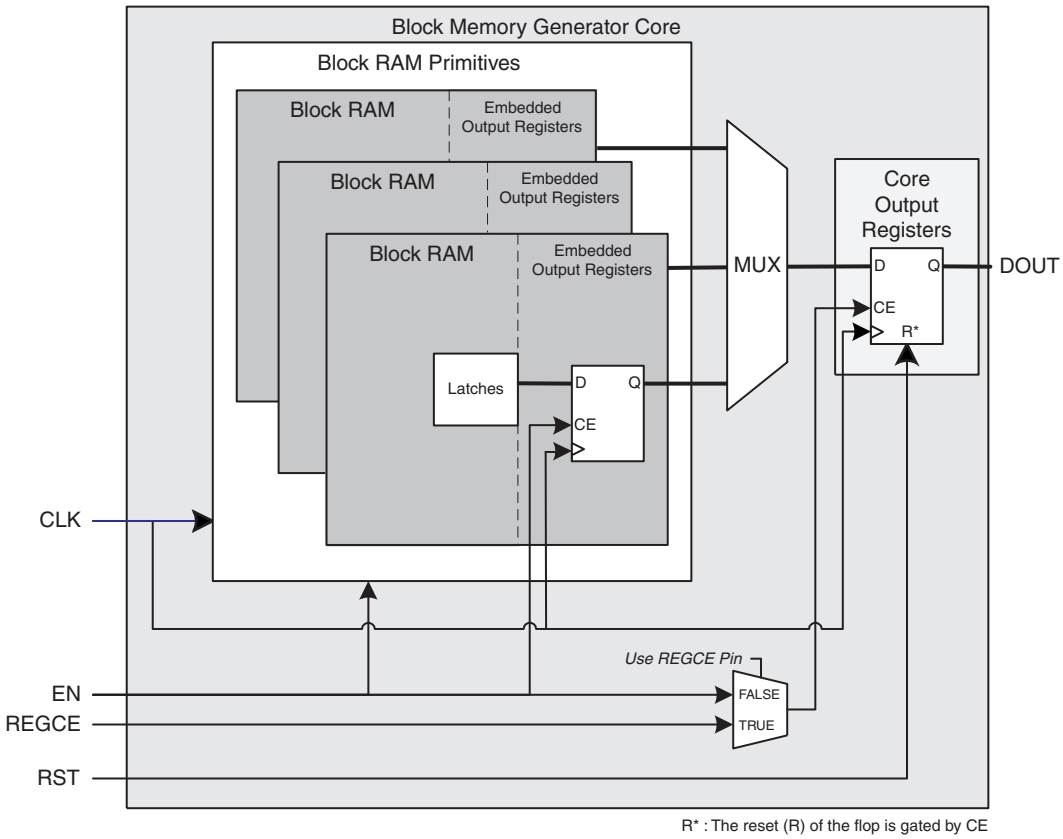

Figure 3-17: **Block Memory with Register Port [A|B] Output of Memory Primitives and Register Port [A|B] Output of Memory Core Options Enabled**

For a complete description of the supported output options, see Output Register Configurations in Appendix D.

Optional Pipeline Stages

The Block Memory Generator core allows optional pipeline stages within the MUX, which might improve core performance. Users can add up to three pipeline stages within the MUX, excluding the registers at the output of the core. This optional pipeline stages option is available only when the registers at the output of the memory core are enabled and when the constructed memory has more than one primitive in depth, so that a MUX is needed on the output.

The pipeline stages are common for port A and port B and can be a value of 1, 2, or 3 if the Register Output of Memory Core option is selected in the Vivado IDE for both port A and port B. Note that each pipeline stage adds an additional clock cycle of latency to the Read operation.

If the configuration has BuiltIn_ECC (ECC), the `SBITERR` and `DBITERR` outputs are delayed to align with `DOUT`. Note that adding pipeline stages within the MUX improves performance only if the critical path in the design is the data through the MUX. The MUX size displayed

in the Vivado IDE can be used to determine the number of pipeline stages to use within the MUX. Figure 3-18 shows a memory configuration with an 8:1 MUX and two pipeline stages within the MUX. In the figure, the 8:1 MUX is pipelined internally with two register stages.

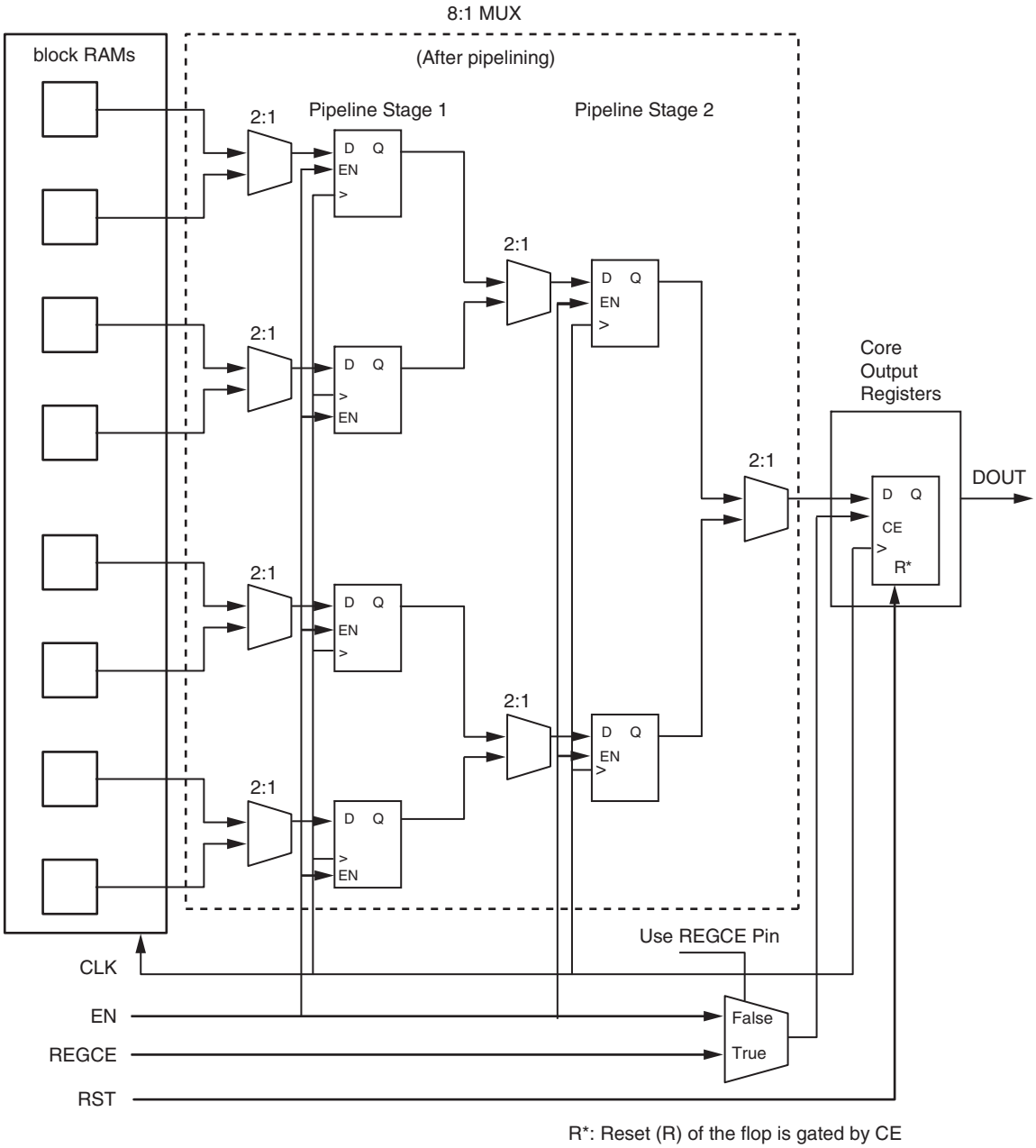

Figure 3-18: **Memory Configuration with 8:1 MUX and Two Pipeline Stages within the MUX**

Note: The Enable signal must be connected to each pipeline stage in the core and must be asserted for *N* clock cycles, where *N* is the number of pipeline stages.

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 of the corresponding port is controlled by the `regcea`/`regceb` pins; 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 independent of the `en` signal.

Optional Set/Reset Pins

The set/reset pins (`rsta` and `rstb`) 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 `rst` 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 Vivado IP catalog GUI. (The reset occurs on `rst` and `en` when the Use **REGCE Pin** option is not selected.) The set/reset behavior differs when the reset behavior option is selected. See Special Reset Behavior, page 62 for more information.

Memory Output Flow Control

The combination of the enable (`en`), reset (`rst`), and register enable (`REGCE`) pins allow a wide range of data flows in the output stage. Figure 3-19 and Figure 3-20 are examples on how this can be accomplished. Keep in mind that the `rst` and `REGCE` pins apply only to the last register stage.

Figure 3-19 depicts how `rst` can be used to control the data output to allow only intended data through. Assume that both output registers are used for port A, 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`.

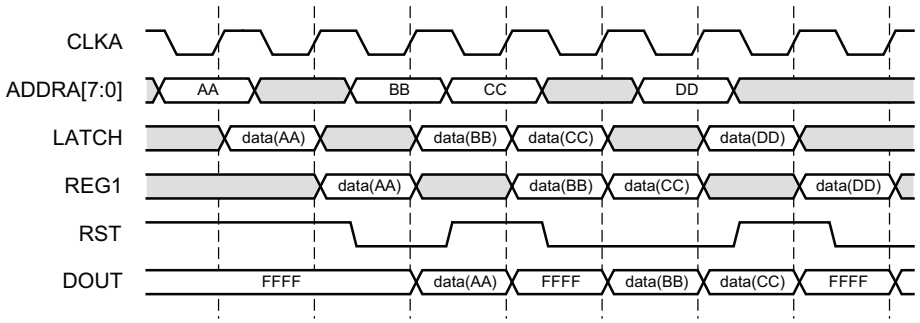

Figure 3-19: **Flow Control Using rst**

Figure 3-20 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 for port A, and that `en` is always asserted and `rst` 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`.

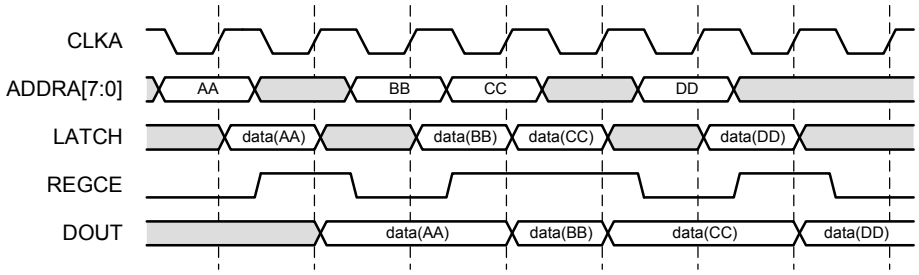

Figure 3-20: **Flow Control Using REGCE**

Read Data and Read Enable Latency

Figure 3-21 shows the read data (`LATCH`) and read enable (`en`) latency when there are no output registers are used is shown below. The `LATCH` signal is the data at the output of the primitive.

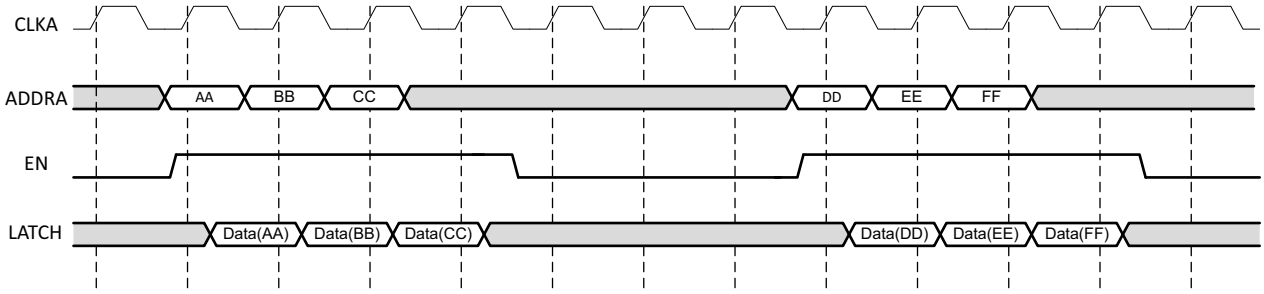

Figure 3-21: **Read Data and Read Enable Latency with no Output Registers**

Figure 3-22 shows the read data (`REG1`) and read enable (`en`) latency when the primitive output register is used. `REG1` is the data at the output of the primitive output register.

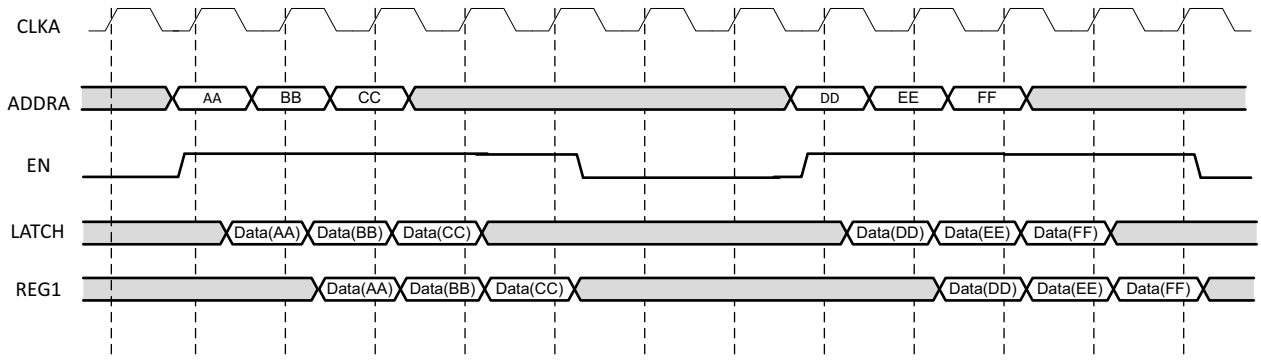

Figure 3-22: **Read Data and rEad Enable Latency with Primitive Output Registers**

Figure 3-23 shows the read data (REG2, REG3) and read enable (en) latency when two pipeline stages are used along with the primitive output register. REG2 is the data at the output of the pipeline stage 1, and REG3 is the data at the output of the pipeline stage 2.

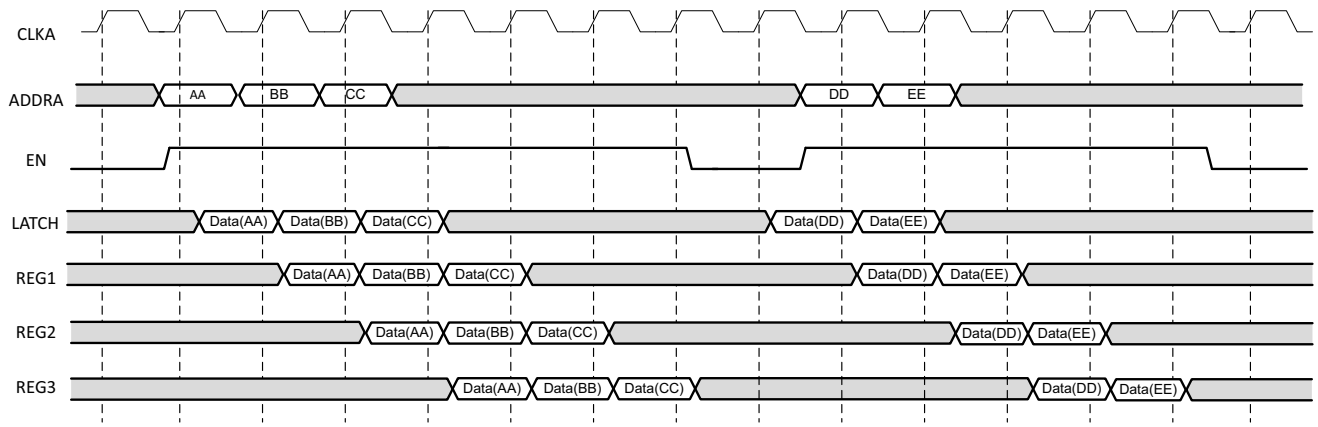

Figure 3-23: **Read Data and Read Enable Latency with Two Pipeline Stages Used**

Figure 3-24 shows the read data (dout) and read enable (en) latency when the core output registers are used along with the primitive output register and two pipeline stages. dout is the data at the output of the core output register.

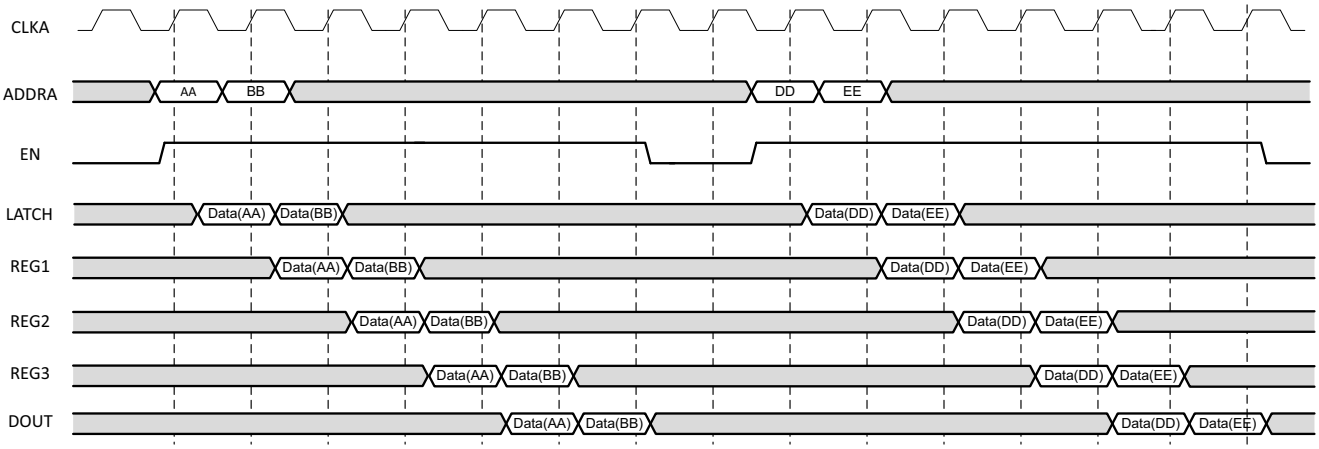

Figure 3-24: **Read Data and Read Enable Latency with Core and Primitive Output Registers**

Reset Priority

The Block Memory Generator core provides the option to choose the reset priority of the output stages of the Block Memory. Reset priority can be set only for the output registers and not for the memory latch. When CE is the selected priority, then CE (regcea or regceb) has a priority over reset (rsta or rstb). When SR is the selected reset priority, then reset has a priority over CE.

Figure 3-25 illustrates the reset behavior when the Reset Priority option is set to CE. Here, the first reset operation occurs successfully because `en` is High, but the second reset operation does not cause any change in output because `en` is Low.

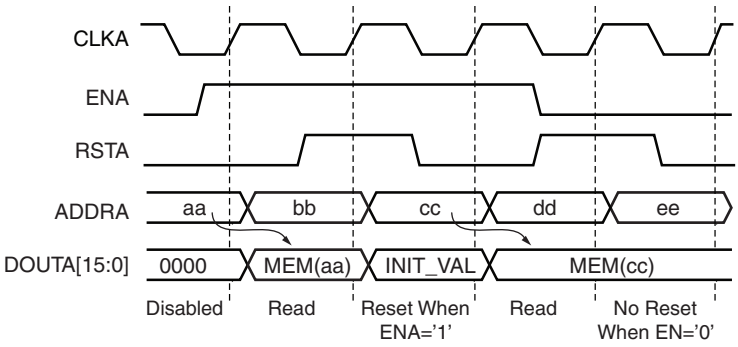

Figure 3-25: **Reset Behavior When Reset Priority is Set to CE**

Figure 3-26 illustrates the reset behavior when the Reset Priority option is set to SR. Here, reset is not dependent on enable and both reset operations occur successfully.

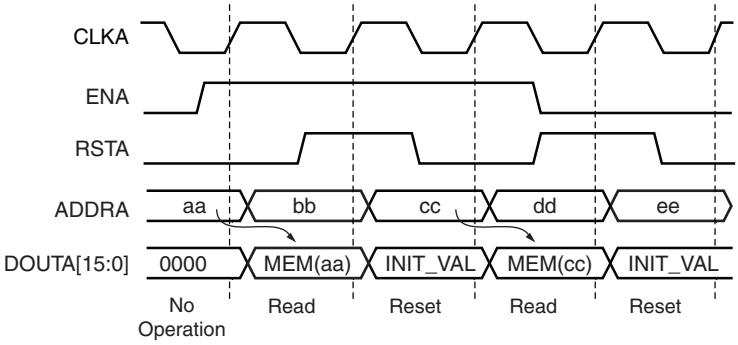

Figure 3-26: **Reset Behavior When Reset Priority is Set to SR**

Special Reset Behavior

The Block Memory Generator core provides the option to reset both the memory latch and the embedded primitive output register. This Reset Behavior option is available to users when they choose to have a primitive output register, but no core output register. When you choose the option to Reset the Memory Latch besides the primitive output register, then the reset value is asserted at the output for two clock cycles. However, when you do not select the option to Reset the Memory Latch in the presence of a primitive output register, the reset value is asserted at the output for only one clock cycle, since only the primitive output register is reset.

Note that the duration of reset assertion specified here is the minimum duration when the latch and register are always enabled, and the `rst` input is held high for only one clock cycle. If the enable signals are deasserted or the `rst` input is held high for more than one clock cycle, the reset value might be asserted at the output for a longer duration.

The latch and the embedded output register can be reset independently using two separate inputs (`rstreg` and `rstram`) that are connected to the primitive. So, if you choose to reset the memory latch, only the embedded output register is reset.

Figure 3-27 and Figure 3-28 illustrate the difference between the standard reset behavior similar to previous architectures obtained when the memory latch is not reset, and the special reset behavior in the new architectures, obtained when the memory latch is reset. Note that there is an extra clock cycle of latency in the data output because of the presence of the primitive output register.

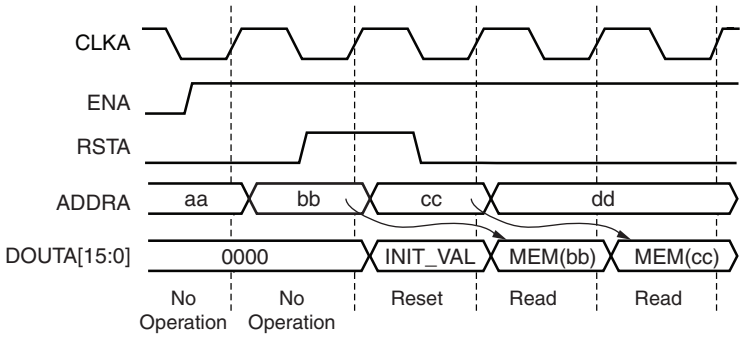

Figure 3-27: **Standard Reset Behavior Similar to Previous Architectures**

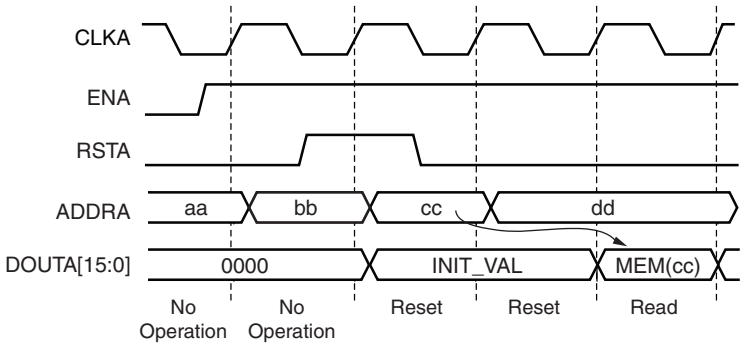

Figure 3-28: **Special Reset Behavior using the Reset Memory Latch Option**

The reset of the memory latch is gated by `en`, and the reset of the embedded register is gated by `ce`. As shown in Figure 3-29, both `ENA` and `REGCEA` are high at the time of the first reset, and the reset value is asserted at the output for two clock cycles. At the time of the second reset, `ena` is High, but `regcea` is Low; so the reset value does not appear at the output. At the time of the third reset, only `regcea` is High; so the reset value is asserted at the output for only one clock cycle.

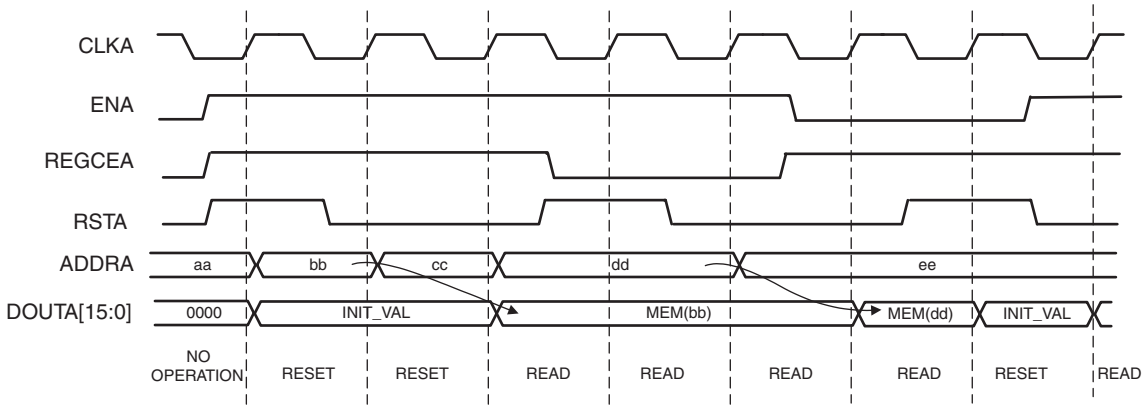

Figure 3-29: **Reset Gated by CE with Reset Memory Latch Option**

Controlling Reset Operations

The reset operation is dependent on the following generics:

- **Use RST[A|B] Pin**: These options determine the presence or absence of rst pins at the output of the core.
- **Reset Memory Latch (for Port A and Port B)**: This option determines if the memory latch is reset in addition to the embedded primitive output register for the respective port.
- **Reset Priority (for Port A and Port B)**: This option determines the priority of clock enable over reset or reset over clock enable for the respective port.

Also, the options of output registers also affect reset functionality, since the option to reset the memory latch depends on these options. Table 3-3 lists the dependency of the reset behavior with these parameters. In these configurations, the core output register does not exist. The reset behavior detailed in Table 3-3 uses Port A as an example. The reset behavior for Port B is identical.

Table 3-3: **Control of Reset Behavior for Single Port**

Use rsta Pin	Register Port A Output of Memory Primitives	Reset Memory Latch	Reset Priority for Port A	Virtex-7 Device Reset Behavior
0	X	X	X	No control over reset
1	0	(cannot be selected)	CE, SR	Since no primitive output register exists, the reset applies only to the latch. The priority of SR cannot be set for the latch. Therefore, reset occurs synchronously when the enable input is 1. The reset value is asserted for only one clock cycle (only if input rst signal is High for only one clock and en is High continuously).
1	1	0	CE	The priority cannot be set for the latch. Therefore reset priority of SR is not supported.Reset occurs synchronously, and is dependent or independent of the input enable signal. Reset value asserted for one clock cycle (only if input rst signal is High for only one clock and regce is High continuously).
1	1	1	CE, SR	The priority cannot be set for the latch. Therefore reset priority of SR is not supported. Both memory latch and embedded output register of primitive are reset. Reset occurs synchronously at both these stages, and is dependent or independent of the input enable signal. Reset value is asserted for at least two clock cycles when enable inputs of both stages are 1, and might be more depending on the input rst and enable signals. If rst is asserted when the latch en input is 1 and the register enable input is 0, the memory latch alone gets reset and this reset value gets output only when the register enable goes High.
1	1	0	SR	Not applicable.
1	1	1	SR	Not applicable.
1	X	X	SR, CE	Not applicable.

Built-in Error Correction Capability and Error Injection

The Block Memory Generator core supports built-in Hamming Error Correction Capability (ECC) for the block RAM primitives. For device support, see Table 3-4. Each Write operation generates eight 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.

Table 3-4: **Hard ECC Data width Support**

	Zynq-7000	7 Series
Block RAM Mode	SDP Mode + Read First	SDP Mode + Read First
Supported Data Widths	≥ 64	≥ 64
Bit Error Insertion Support	Yes	Yes

This operation is transparent for you. 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, the Read operation does not correct the error in the memory array; it only presents corrected data on `dout`. BuiltIn_ECC is only available when the Simple Dual-port RAM memory type is selected.

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

The BuiltIn_ECC primitives calculate BuiltIn_ECC for a 64-bit wide data input. If you choose the data width as an integral multiple of 64 (for example, there are spare bits in any BuiltIn_ECC primitive), then a double-bit error (`dbiterr`) might indicate that one or more errors have occurred in the spare bits. So, the accuracy of the `dbiterr` signal cannot be guaranteed in this case. For example, if your data width is 32, then 32 bits of the primitive are left spare. If two of the spare bits are corrupted, the `dbiterr` signal would be asserted even though the actual user data is not corrupt.

When using BuiltIn_ECC, the following limitations apply:

- Byte-Write enable is not available
- All port widths must be identical
- Read First Operating Mode is supported
- Use RST[A|B] Pin and the Output Reset Value options are not available
- Memory Initialization is not supported
- No Algorithm selection is available

Figure 3-30 illustrates a typical Write and Read operation for a Block Memory Generator core in Simple Dual-port RAM mode with BuiltIn_ECC enabled, and no additional output registers.

Note: When built-in ECC (SDP mode) is used for 7-Series and UltraScale architectures, all the byte write enable bits of RAMB36E1 or RAMB36E2 are tied to 1(High) for power saving. This causes the data on `dina` port to be written in memory location addressed by `addra` when `ena` is high. For more details, See the 7 Series FPGAs memory Resources User Guide (UG473)[Ref 11] and UltraScale Architecture Memory resources (UG573)[Ref 12].

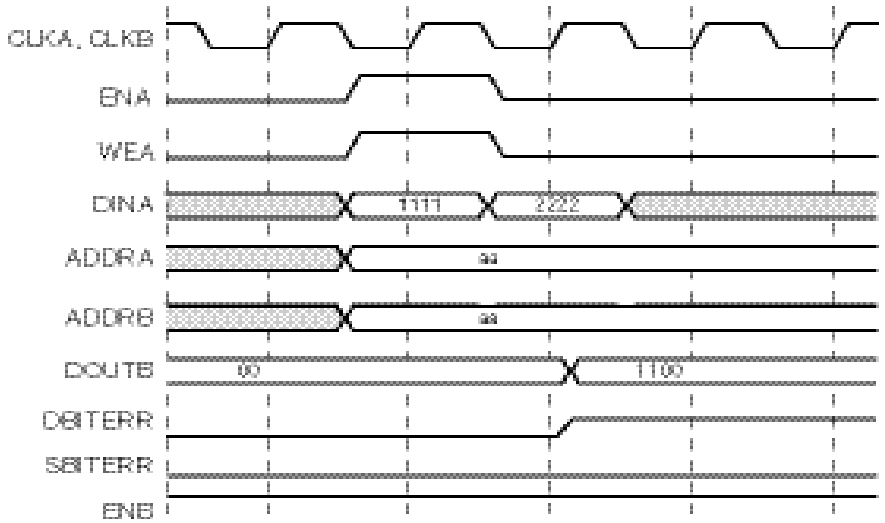

Figure 3-30: **Read and Write Operation with BuiltIn_ECC**

Error Injection

The Block Memory Generator core supports error injection through two optional pins: `injectsbiterr` and `injectdbiterr`. Users can use these optional error injection pins as debug pins to inject single or double-bit errors into specific locations during Write operations. You can then check the assertion of the `sbiterr` and `dbiterr` signals at the output of those addresses. You have the option to have no error injection pins, or to have only one or both of the error injection pins.

The `RDADDRECC` output port indicates the address at which a `sbiterr` or `dbiterr` has occurred. The `rdaddrecc` port, the two error injection ports, and the two error output ports are optional and become available only when the BuiltIn_ECC option is chosen. If you have not selected the BuiltIn_ECC feature, the primitive's `injectsbiterr` and `injectdbiterr` ports are internally driven to '0', and the primitive's outputs `sbiterr`, `dbiterr`, and `rdaddrecc` are not connected externally.

Figure 3-31 shows the assertion of the `sbiterr` and `dbiterr` output signals when errors are injected through the error injection pins during a Write operation.

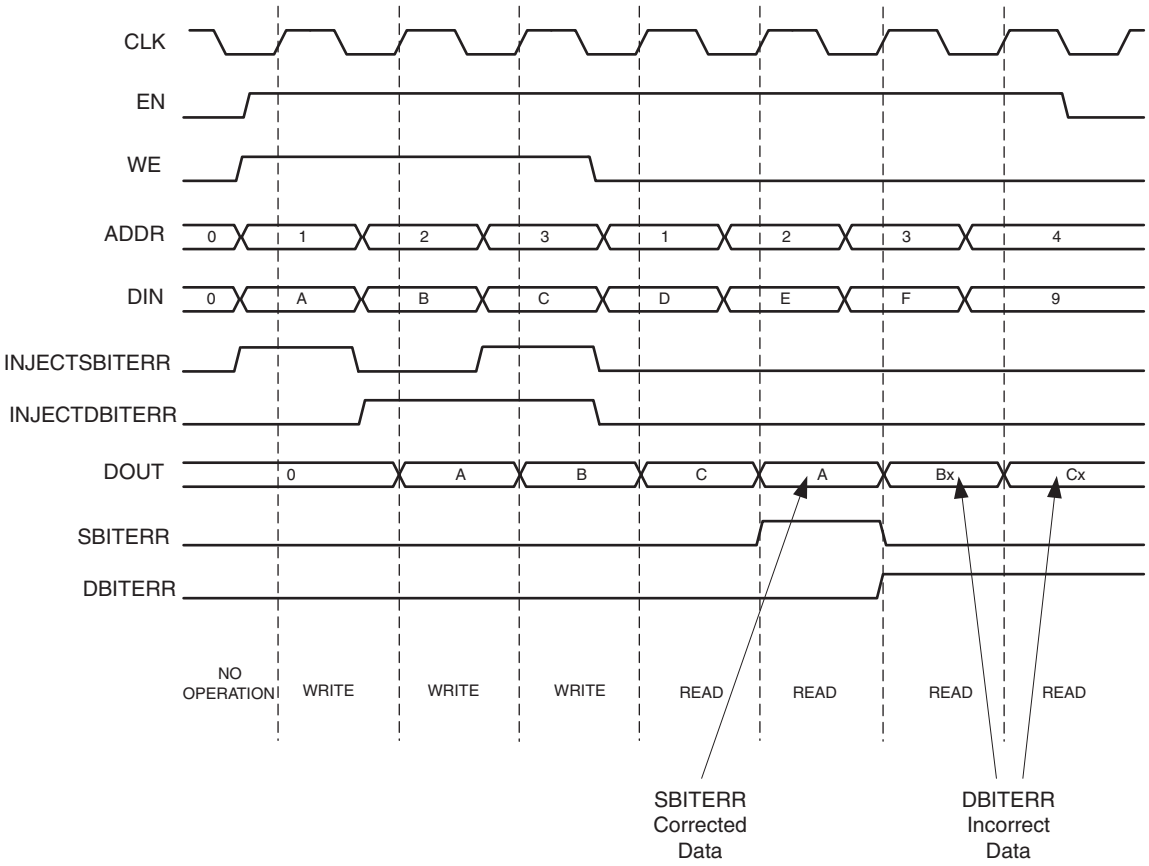

Figure 3-31: **Assertion of SBITERR and DBITERR Signals by Using Error Injection Pins**

When the `injectsbiterr` and `injectdbiterr` inputs are asserted together at the same time for the same address during a Write operation (as in the case of address 3 in Figure 3-31), the `injectdbiterr` input takes precedence, and only the `dbiterr` output is asserted for that address during a Read operation. The data output for this address is not corrected.

Soft Error Correction Capability and Error injection

This section describes the implementation and usage of the Soft Error Correction Control (Soft ECC) module that is supported in the Block Memory Generator core.

Overview

The Soft ECC module detects and corrects all single-bit errors in a code word consisting of up to 64 bits of data and up to eight parity bits. In addition, it detects double-bit errors in the data. This design uses Hamming code, which is an efficient method for ECC operations.

Features

- Supports Soft ECC for data widths less than or equal to 64 bits

- Uses Hamming error code correction
  - Single-bit errors are corrected
  - Double-bit errors are detected
- Supports Simple Dual-port RAM memory type
- Supports optional Input and/or Output Registering stages
- Supported Block Memory Generator core features include:
  - Minimum Area, Fixed Primitive and Low Power Algorithms
  - Mux Pipelining Stages
  - Embedded Primitive Registers
  - Core Output Registers
  - Optional Enable Inputs
- Fully parameterized implementation enables optimization of resources

Details

Each Write operation generates between 4 and 8 protection bits for 1 to 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.

The 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, the Read operation does not correct the error in the memory array; it only presents corrected data on DOUT.

When using Soft ECC, the Block Memory Generator core can construct the memory from the available primitives. The Soft ECC feature is implemented as an overlay on top of the Block Memory Generator core. This allows users to select algorithm options and registering options in the core. When Soft ECC is selected, limited core options include:

- Byte-Write enable is not available
- All port widths must be identical
- Use RST[A|B] Pin and the Output Reset Value options are not available
- Memory Initialization is not supported

The Soft ECC implementation is optimized to generate the core for different data widths, as shown in Table 3-5. For the selected data width, the number of check bits appended is shown in Table 3-6. The operation of appending the additional check bits for the given data width is done within the Block Memory Generator core and is transparent to you.

Table 3-5: **Soft ECC Data width Support**

	Zynq-7000	7 Series
Block RAM mode	SDP Mode	SDP Mode
Supported Data Widths	≤ 64 bits	≤ 64 bits
Bit Error Insertion Support	Yes	Yes

Table 3-6: **Memory Width Calculation for Selected User Data Width**

User Input Data Width	Added Check Bits	Total Memory Width in Bits
1-4	4	5-8
5-11	5	10-16
12-26	6	18-32
27-57	7	34-64
58-64	8	66-72

Figure 3-32 illustrates the implementation of Soft ECC logic for the Block Memory Generator core. The implementation shown in Figure 3-32 is for 64 bits of data; the implementation is parameterized for other data widths.

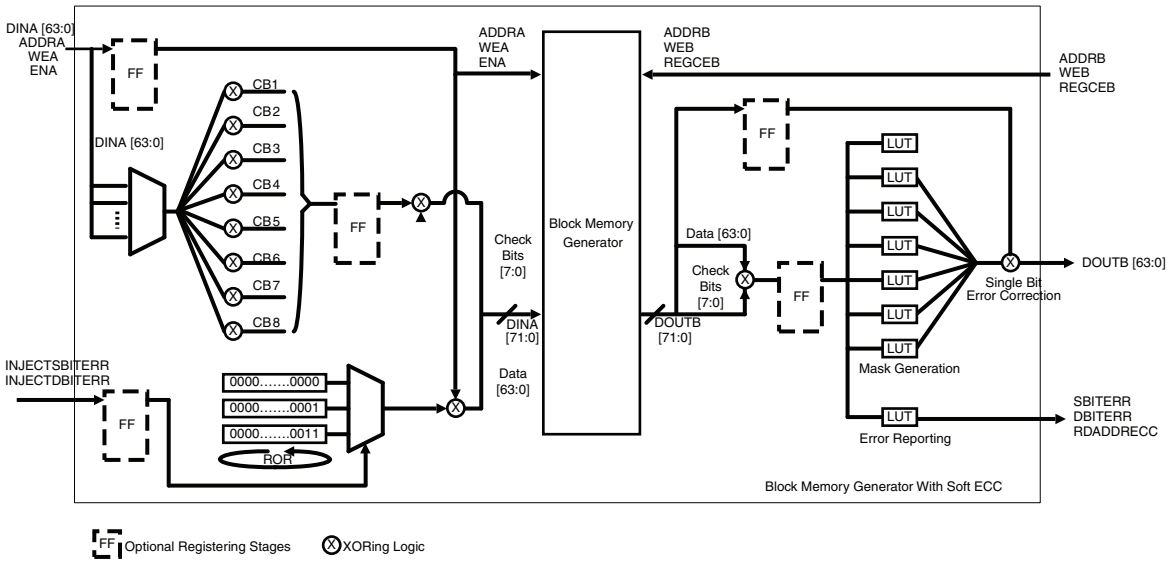

Figure 3-32: **Block Memory Generator with Soft ECC**

The optional input and output registering stages can be enabled by setting the values of parameters register_porta_input_of_softecc and register_portb_output_of_softecc appropriately. These registers improve the performance of the Soft ECC logic. By default, the input and output registering stages are disabled.

With Soft ECC, error injection is supported through two optional pins: `injectsbiterr` and `injectdbiterr`. The error injection operation is performed on both data bits and added check bits. Users can use these optional error injection pins as debug pins to inject

single or double-bit errors into specific locations during Write operations. Then, you can check the assertion of the `sbiterr` and `dbiterr` signals at the output of those addresses. You might select no error injection pins, one error injection pin or both.

The RDADDRECC output port indicates the address at which a single or double-bit error has occurred. The RDADDRECC port, the two error injection ports, and the two error output ports are optional and become available only when the Soft ECC option is chosen. If the Soft ECC feature is not selected, the outputs `sbiterr`, `dbiterr`, and `rdaddrecc` are not connected externally.

Parameters

- **softecc**: This parameter enables the Soft ECC logic.
- **register_porta_input_of_softecc**: This parameter registers the input ports in the design.
- **register_portb_output_of_softecc**: This parameter registers the output ports in the design.
- **use_error_injection_pins**: This parameter enables single and/or double-bit error injection capability during Write operations
- **error_injection_type**: This parameter specifies the type of the error injection done in the Soft ECC logic. The error injection type can be either "Single_Bit_Error_Injection" or "Double_Bit_Error_Injection" or "Single_and_Double_Bit_Error_Injection."

Parameter–Port dependencies

- When the **softecc** parameter is enabled: `sbiterr`, `dbiterr`, and `rdaddrecc` ports are made available on the I/O interface.
- When the **use_error_injection_pins** parameter is enabled and "Single_Bit_Error_Injection" option is selected: `injectsbiterr` port is made available on the I/O interface.
- When the **use_error_injection_pins** parameter is enabled and "Double_Bit_Error_Injection" option is selected: `injectdbiterr` port is made available on the I/O interface
- When the **use_error_injection_pins** parameter is enabled and "Single_and_Double_Bit_Error_Injection" option is selected: `injectsbiterr` and `injectdbiterr` ports are made available on the I/O interface

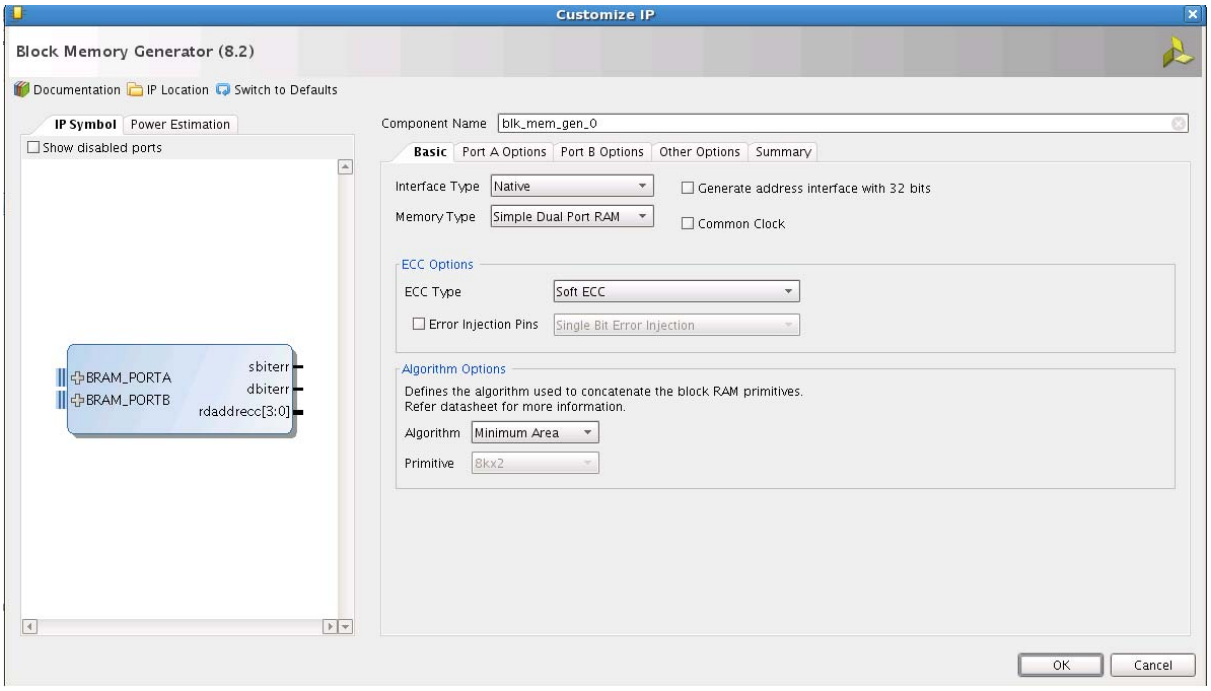

Figure 3-33: **Enabling Soft ECC Option**

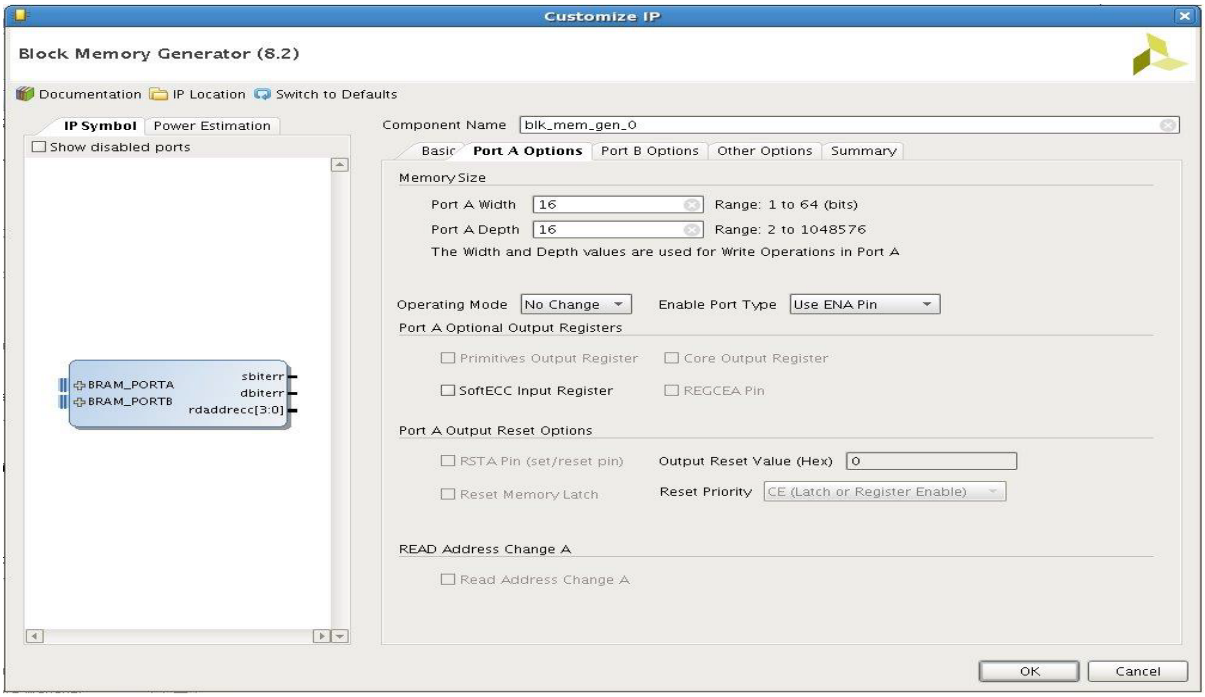

Figure 3-34: **Enabling Input Registering Stages**

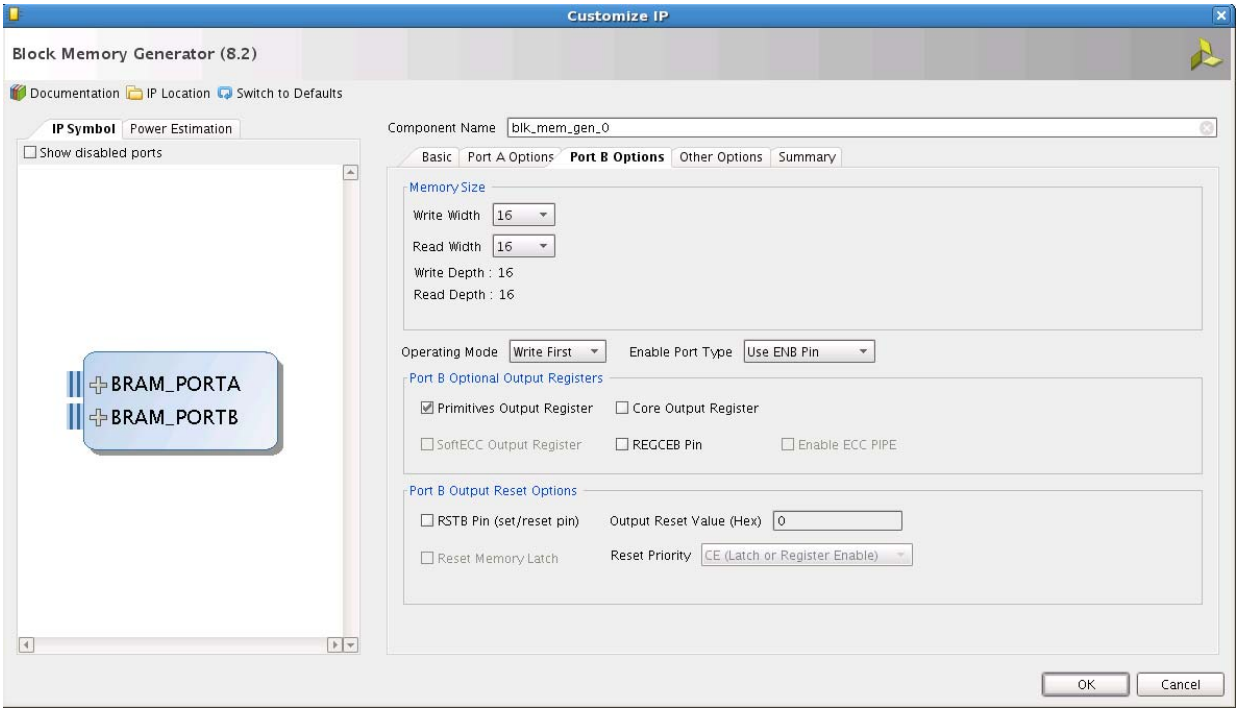

Figure 3-35: **Enabling Output Registering Stages**

Timing Diagrams

Figure 3-36 illustrates a typical Write and Read operation for a core with a simple dual-port RAM configuration with Soft ECC enabled and no additional input or output registers.

Figure 3-36: **Read and Write Operations with Soft ECC**

Figure 3-37 shows the assertion of the `sbiterr` and `dbiterr` output signals when errors are injected through the error injection pins during a Write operation.

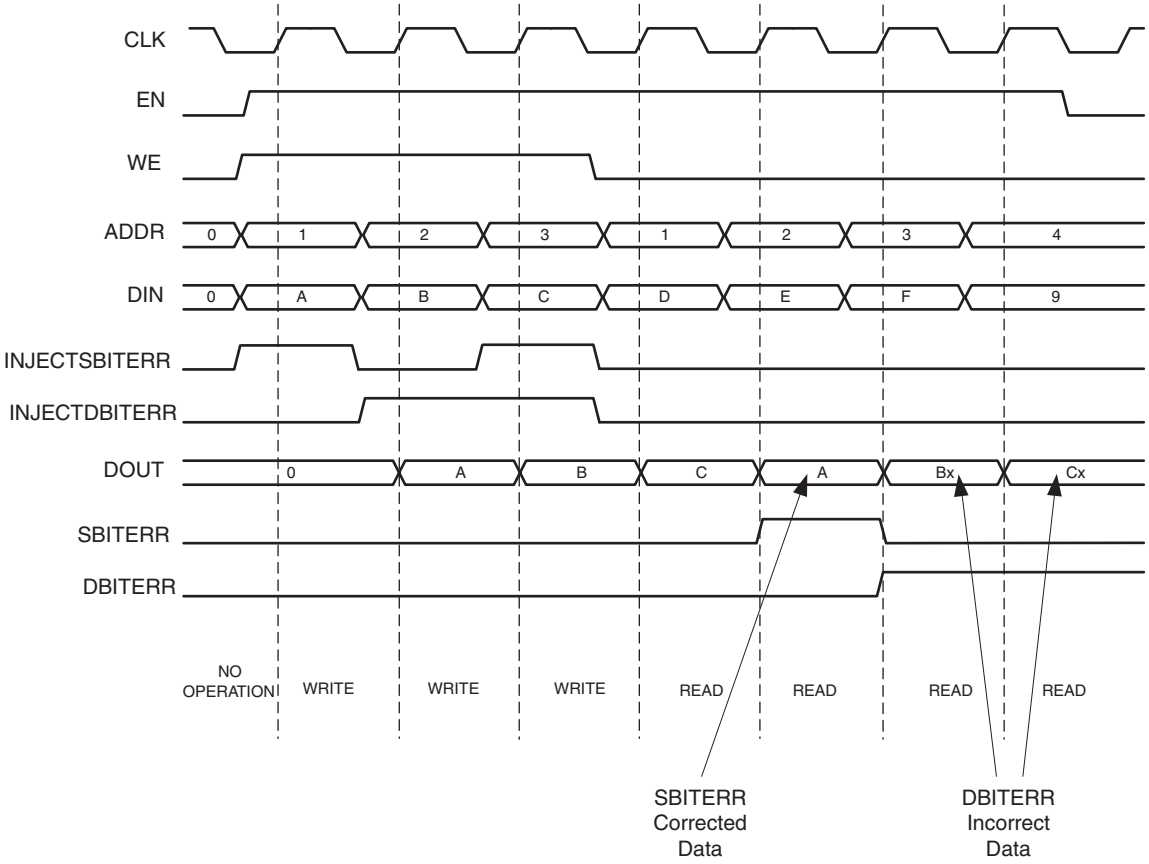

Figure 3-37: **Assertion of SBITERR and DBITERR Signals**

When the `injectsbiterr` and `injectdbiterr` inputs are asserted together at the same time for the same address during a Write operation (address 3 in Figure 3-37), then the `injectdbiterr` input takes precedence. Only the `dbiterr` output is asserted for that address during a Read operation. The data output for this address is not corrected.

Device Utilization and Performance Benchmarks

Table 3-7: **Resource Utilization for Virtex-7 Devices (XC7VX550T-FFG1927-1)[(1)(2)]**

Depth x Width	Mode	Check Bits	Resource Utilization[(3)]			Performance[(4)]
			Block RAMs	Slice LUTs	FF	Max Freq (MHz)
1Kx8	W/o ECC		1	0	0	368
	With ECC	5	1	32	10	352
1Kx16	W/o ECC		1	0	0	360
	With ECC	6	1	61	10	352
1Kx32	W/o ECC		1	0	0	344
	With ECC	7	2	95	10	344

Table 3-7: **Resource Utilization for Virtex-7 Devices (XC7VX550T-FFG1927-1)[(1)(2)]**

Depth x Width	Mode	Check Bits	Resource Utilization[(3)]			Performance[(4)]
			Block RAMs	Slice LUTs	FF	Max Freq (MHz)
1Kx64	W/o ECC		2	0	0	352
	With ECC	8	2	181	10	288

Notes:
1. Uses Fixed Primitive Algorithm (Primitive Configuration is 512x36)
2. Memory type is SDP
3. No register stage
4. Uses Memory Core Output Register

Lower Data Widths in SDP Configurations

The SDP primitives support lower data widths. Width combinations are possible for Port A and Port B, as shown in Table 3-8.

Table 3-8: **Data Widths Supported by SDP Primitives[(1)(2)(3)]**

Primitive	Read Port Width	Write Port Width	Read Port Width	Write Port Width
RAMB18 SDP Primitive	x1	x32	x32	x1
	x2	x32	x32	x2
	x4	x32	x32	x4
	x9	x36	x36	x9
	x18	x36	x36	x18
	x36	x36	-	-
RAMB36 SDP Primitive	x1	x64	x64	x1
	x2	x64	x64	x2
	x4	x64	x64	x4
	x9	x72	x72	x9
	x18	x72	x72	x18
	x36	x72	x72	x36
	x72	x72	-	-

Notes:
1. Refer to Additional Memory Collision Restrictions: Address Space Overlap, page 56.
2. The selection of SDP primitives depends on the algorithm selected.
3. Asymmetric data widths cause the SDP to be not selected.

UltraScale Architecture-Based Device Features

This section describes features that apply to designs using UltraScale architecture-based devices.

Standard DOUT Block RAM Cascading

Block RAMs (RAMB36E2, RAMB18E2) are cascaded in depth to form a bigger block RAM to reduce the required output data multiplexers. The data out is cascaded from one block RAM to next block RAM serially to make a bigger block RAM in a bottom-up method. When using this built-in cascading feature of the UltraScale architecture-based devices, the primitives are implemented as shown in Table 3-9.

Table 3-9: **UltraScale Architecture Primitives**

Primitive Width	Primitive Depth
1	48k
	80k
	96k
	112k
	128k
2	24k
	32k
	40k
	48k
	64k
	80k
	96k
	112k
4	12k
	16k
	20k
	24k
	32k
	40k

Table 3-9: **UltraScale Architecture Primitives *(Cont'd)***

Primitive Width	Primitive Depth
9	6k
	8k
	10k
	12k
	16k
	20k
18	3k
	4k
	5k
	6k
	8k
	10k
36	2k
	3k
	4k
	5k
Note: The following primitives are available only when the memory type is simple dual-port RAM. These primitives are available for both ECC and non-ECC configurations.	
72	1k
	1.5k
	2k
	2.5k

Note: These primitives will construct the requested memory only when you select an UltraScale device with a no aspect ratio configuration and Low_power algorithm.

Pipe Line Register Addition in the Built ECC Mode

In the Built-In ECC mode, a Pipeline Register is added to improve the maximum frequency. The additional pipeline register a separate enable control which is available to you in the Built-in ECC mode. In the Block Memory Generator core Vivado IDE, ECCPIPECE is the new clock enable for the pipe line register.

Figure 3-38, Figure 3-39, and Figure 3-40 show the timing diagrams for the ECC Pipeline write and read operation.

IMPORTANT: *In the figures, DO_PIPE is the data at the output of the ECCPIPE register and DO is the data at the output of the memory.*

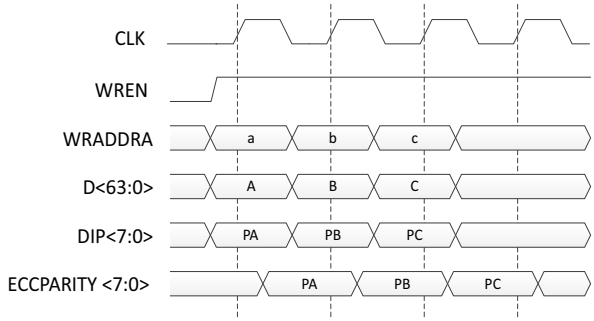

Figure 3-38: **ECC PIPELINE Write Operation**

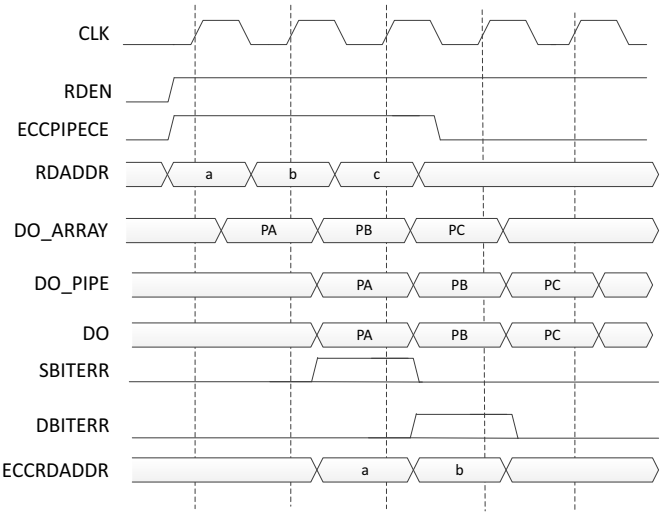

Figure 3-39: **ECC PIPELINE READ Operation (Latch Mode)**

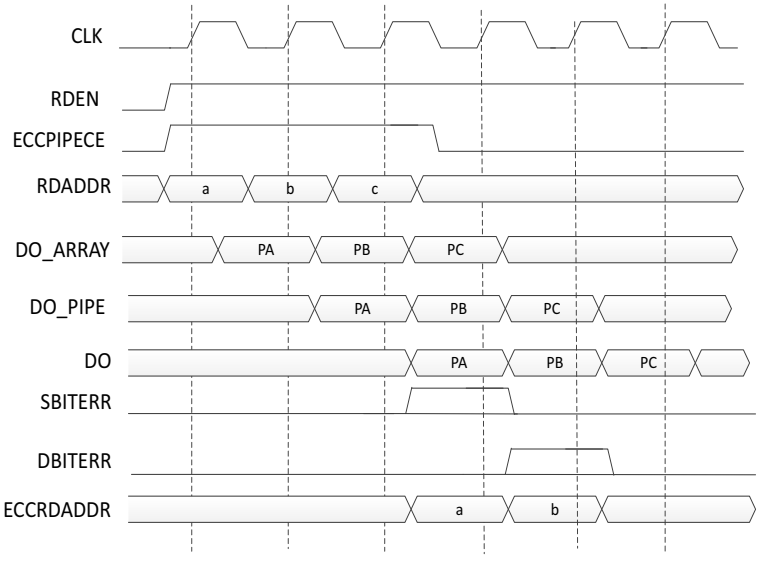

Figure 3-40: **ECC PIPELINE READ Operation (Register Mode)**

Clocking

The Block Memory Generator core has two clocks: `clka` and `clkb`. Depending on the configuration, one or two clocks can be enabled.

Resets

The Block Memory Generator core has two resets. Depending on the configuration, the core can have different reset operations as explained in Optional Set/Reset Pins, Reset Priority, and Special Reset Behavior.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 4]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5]

Customizing and Generating the Core

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog. The Block Memory Generator core is available in the Vivado IP catalog. To open the Block Memory core, go to **IP Catalog** > **Memories & Storage Elements** > **RAMs & ROMs**.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu .

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 10].

IMPORTANT: *For details about customizing this core with IP integrator, see Customizing the Core with IP Integrator.*

For details about files generated with the core, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Note: Figures in this chapter are illustrations of the Vivado IDE. This layout might vary from the current version. The actual Vivado IDE depends on the user configuration.

The following section defines the possible customization options in the Block Memory Generator core Vivado IDE. The Native interface Block Memory Generator core Vivado IDE includes the following tabs:

- Native Block Memory Generator Basic Tab
- Port Options Tab
- Other Options Tab
- Summary Tab
- Power Estimate Options Tab
- Block RAM Usage
- LUT Utilization and Performance
- Generating the AXI4 Interface Block Memory Generator Core
- Basic Tab
- AXI4 Tab
- User Parameters

Native Block Memory Generator Basic Tab

The main Block Memory Generator tab is used to define the component name and provides the Interface Options for the core.

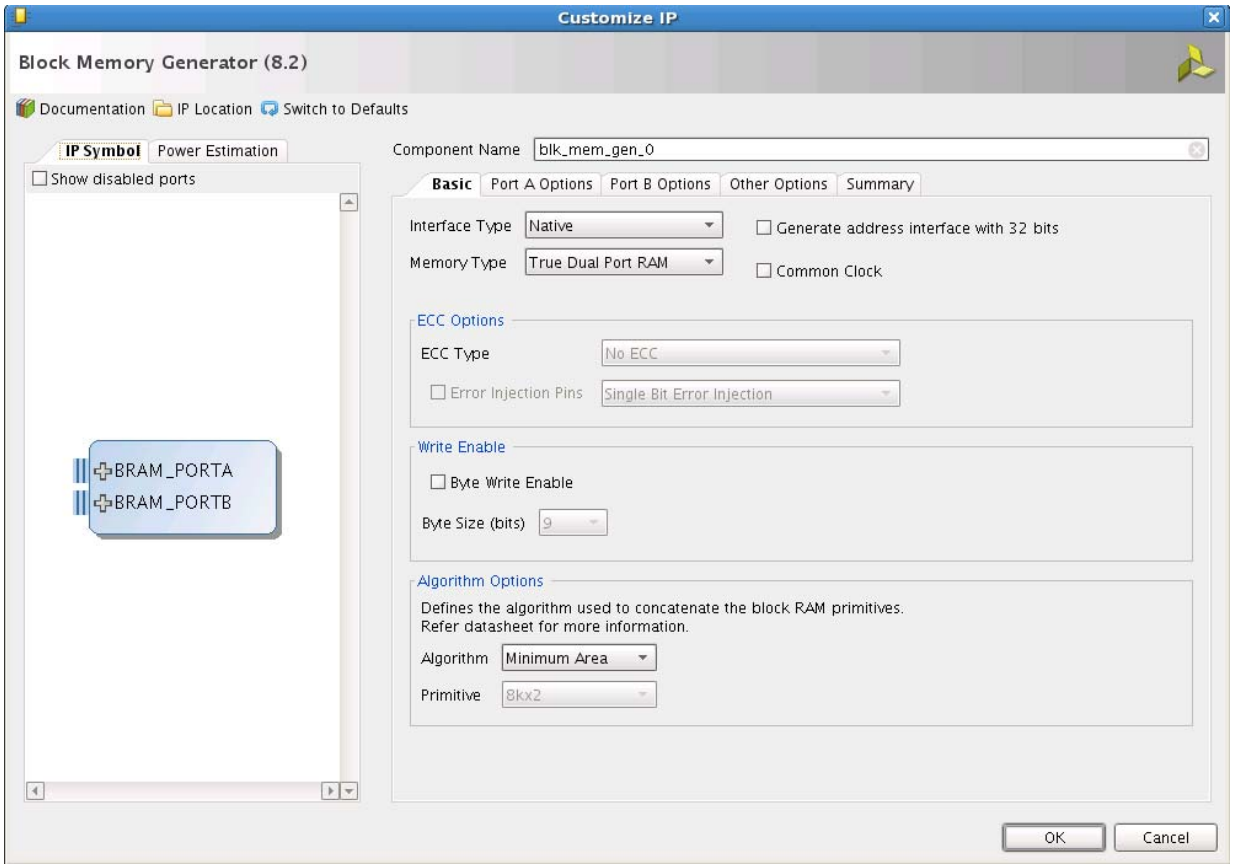

Figure 4-1: **Block Memory Generator Basic Tab**

- **Interface Type**:
  - Native: Implements a Native Block Memory Generator core compatible with previously released versions of the Block Memory Generator core.
- **Memory Type**: Select the type of memory to be generated.

IMPORTANT: *The memory type that is selected in the Vivado IDE might not propagate to the Block RAMs implemented. For 7 series devices that make use of the port aspect ratios, RAM_MODE is set to Single Dual-port RAM (SDP). For more details about these configurations, see the 7 Series FPGAs memory Resources User Guide (UG473) [Ref 11]. All other configurations for the RAM_MODE are not guaranteed to be SDP.*

  - Single-port RAM
  - Single Dual-port RAM
  - True Dual-port RAM
  - Single-port ROM
  - Dual-port ROM

- **ECC Options**: When the Simple dual-port RAM memory type is selected, the ECC Type option becomes available. It provides you with the choice to select the type of ECC required.
  - **Built-In ECC**: When the selected ECC Type is BuiltIn_ECC, the built-in Hamming Error Correction is enabled.

    The Use Error Injection Pins option is available for selection if the ECC option is selected. This option enables error injection pins. On choosing this option, additional options are available to have Single-Bit Error Injection (INJECTSBITERR), Double-Bit Error Injection (INJECTDBITERR), or both. See Hamming Error Correction Capability in Chapter 1 for more information.

    When using ECC, the following limitations apply:

    - Byte-Write Enable is not available.
    - All port widths must be identical.
    - Read First Operating Mode is supported when the **Common Clock** option is selected.
    - The Use RST[A|B] Pin and the Output Reset Value options are not available.
    - Memory Initialization is not supported.
    - No algorithm selection is available.
    - Generating a 32-bit address interface is not supported.
  - **Soft ECC**: When the selected ECC Type is Soft_ECC, soft error correction (using Hamming code) is enabled.

    The Use Error Injection Pins option is available for selection if the Soft ECC option is selected. This option enables error injection pins. On choosing this option, additional options are available to have Single-Bit Error Injection (INJECTSBITERR), Double-Bit Error Injection (INJECTDBITERR), or both. See Soft Error Correction Capability and Error injection in Chapter 3 for more information about this option and the limitations that apply.

    When using Soft ECC, the following conditions apply:

    - Supports Soft ECC for data widths less than or equal to 64 bits
    - Uses Hamming error code correction: Single-bit errors are corrected and double-bit errors are detected.
    - Supports Simple Dual-port RAM memory type
    - Supports optional Input and/or Output Registering stages
    - Supported Block Memory Generator core features include:

      Minimum Area, Fixed Primitive and Low Power Algorithms

Mux Pipelining Stages

Embedded Primitive Registers

Core Output Registers

Optional Enable Inputs

- Fully parameterized implementation for optimized resource utilization

- **Common Clock**: Select the Common Clock option when the clock (CLKA and CLKB) inputs are driven by the same clock buffer.

IMPORTANT: *For devices with COMMON_CLOCK selected, WRITE_MODE is set as READ_FIRST for Simple Dual-Port RAM Memory type, otherwise WRITE_MODE is set as WRITE_FIRST.*

- **Write Enable**: Selects 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 are multiples of the selected byte-size.

  Note: For UltraScale™ architecture devices, the low power algorithm calls the cascaded primitives to build the required configuration.

- **Algorithm**: Select the algorithm used to implement the memory:
  - **Minimum Area Algorithm**: Generates a core using the least number of primitives.
  - **Low Power Algorithm**: Generates a core such that the minimum number of block RAM primitives are enabled during a Read or Write operation.
  - **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 Options Tab

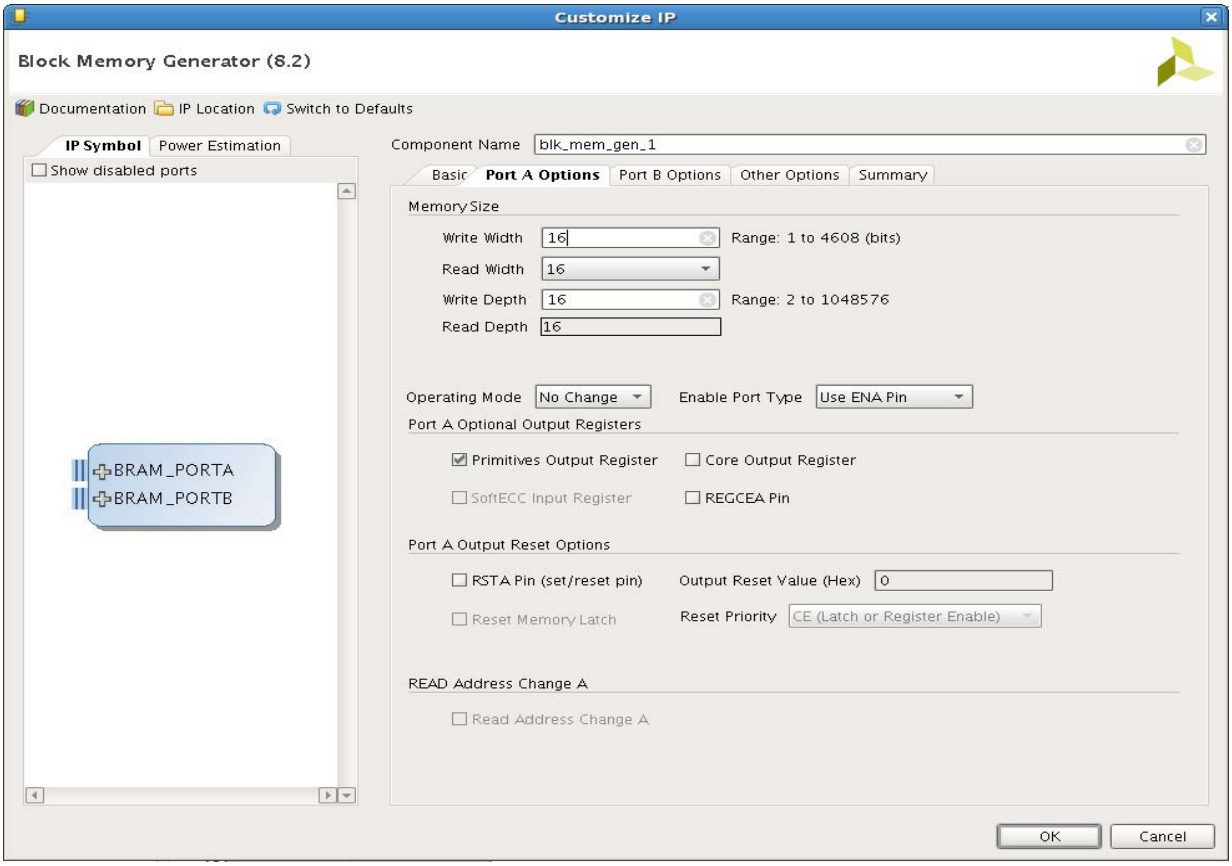

Figure 4-2: **Port A Options**

- **Memory Size**: Specify the port [A|B] Write width and depth. Select the port [A|B] Read width from the drop-down list of valid choices. The Read depth is calculated automatically.
- **Operating Mode**: Specify the port [A|B] operating mode.
  - READ_FIRST
  - WRITE_FIRST
  - NO_CHANGE
- **Enable Port Type**: Select the enable type:
  - Always enabled (no `ENA`|`ENB` pin available)
  - Use `ENA`|`ENB` pin
- **Port [A|B] Optional Output Registers**: Select the output register stages to include:
  - **Primitives Output Register**: Select to insert output register after the memory primitives for port A and port B separately. The Primitive Output Registers option is set as a default option in standalone mode. The embedded output registers in the

block RAM primitives are used if you choose to register the output of the memory primitives. See Output Register Configurations in Appendix D for more information.

- **Core Output Register**: Select for each port (A or B) to insert a register on the output of the memory core for that port. When selected, registers are implemented using FPGA slices to register the core output.
- **REGCE[A|B] Pin**: Select to use a separate REGCEA or REGCEB input pin to control the enable of the last output register stage in the memory. When unselected, all register stages are enabled by ENA/ENB.

- **Port [A|B] Output Reset Options**
  - **Use RST[A|B] Pin (Set/Reset Pin)**: Choose whether a set/reset pin (RST[A|B]) is needed.
  - **Reset Priority**: The Reset Priority option for each port is available only when the Use RST Pin option of the corresponding port is chosen. You can set the reset priority to either CE or SR. For more information on the reset priority feature, see Reset Priority in Chapter 3.
  - **Reset Behavior**: The Reset Behavior (Reset Memory Latch) options for each port are available only when the Use RST Pin option and the Register Output of Memory Primitives option of the corresponding port are chosen, and the Register Memory Core option of the corresponding port is not chosen.
  - **Reset Memory Latch**: Modifies the behavior of the reset and changes the duration for which the reset value is asserted. The minimum duration of reset assertion is displayed as information in the Vivado IDE based upon the choice for this option. For more information on the Reset Memory Latch option, see Special Reset Behavior in Chapter 3.
  - **Output Reset Value (Hex)**: Specify the reset value of the memory output latch and output registers. These values are with respect to the Read port widths.
- **Enable ECC PIPE**: Enables the pipeline register to improve the maximum frequency. Port B only.

IMPORTANT: *The Enable ECC PIPE option is only available for UltraScale architecture-based devices in Built-in ECC mode.*

- **RDADDRCHANGE[A|B] Option**

  This feature is available only for ultra-scale devices. This is a power-saving feature and enables the read address change (compare) detection circuit inside the block RAM. When RDADDRCHANGE is TRUE and the read address and output registers are identical to the previous read cycle, it would result in the same output. Hence, the block RAM access is not performed to save power. This is most useful if you permanently enable the block RAM. This feature is only available when Common Clock Option is set to TRUE. At any point of time, both RDADDRCHANGEA and RDADDRCHANGEB should be set to the same value i.e. both of them should be set to TRUE or FALSE.

For more information, please refer to *UltraScale Architecture Memory Resources User Guide* (UG572)

Other Options Tab

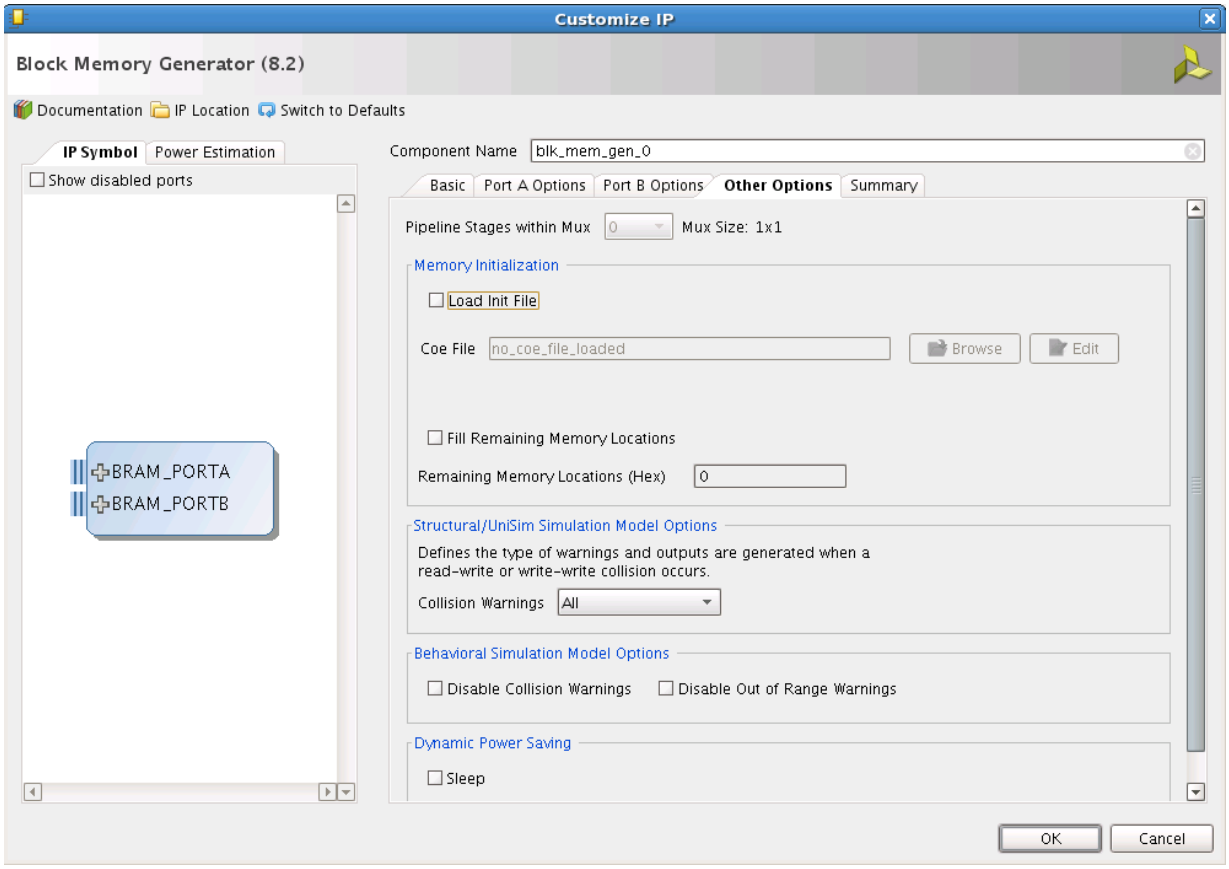

Figure 4-3: **Other Options Tab**

- **Pipeline Stages within Mux**: Available only when the Register Output of Memory Core option is selected for both port A and port B and when the constructed memory has more than one primitive in depth, so that a MUX is needed at the output. Select a value of 0, 1, 2, or 3 from the drop-down list.

  The MUX size displayed in the Vivado IDE can be used to determine the number of pipeline stages to use within the MUX. Select the appropriate number of pipeline stages for your design based on the device architecture.

- **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. See Specifying Initial Memory Contents for more details.

- **Structural/UNISIM Simulation Model Options**: Select the type of warning messages and outputs generated by the structural simulation model in the event of collisions. For the options of ALL, WARNING_ONLY and GENERATE_X_ONLY, the collision detection feature is enabled in the UNISIM models to handle collision under any condition.

  The NONE selection is intended for designs that have no collisions and clocks (Port A and Port B) that are never in phase or within 3000 ps in skew. If NONE is selected, the collision detection feature is disabled in the models, and the behavior during collisions is left for the simulator to handle. So, the output will be unpredictable if the clocks are in phase or from the same clock source or within 3000 ps in skew, and the addresses are the same for both ports. The option NONE is intended for design with clocks never in phase.

- **Behavioral Simulation Model Options**: Select the type of warning messages generated by the behavioral simulation model. Select whether the model should assume synchronous clocks (Common Clock) for collision warnings.
- **Dynamic Power Saving**: Enables static power savings in a situation where the memory is not actively being used for an extended period of time. When the memory array goes into sleep mode, the memory preserves the data. To use the memory, set the `sleep` pin to 0 to wake the memory.

  `sleep` is synchronous to `clka`. No block RAM operation is allowed from the point in the cycle where `sleep` goes High. While `sleep` is High, `en` on both ports must be held Low.

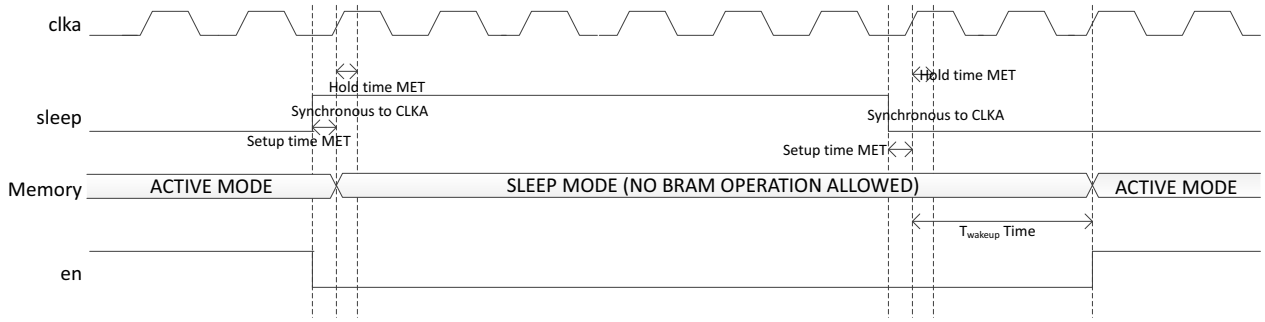

Figure 4-4: **Sleep Pin Timing Diagram**

IMPORTANT: *The `sleep` pin is only available for UltraScale architecture-based devices and is not available in AXI mode. In addition, the `sleep` pin is not available when port type A or B is set as* ***Always enabled*** *in the Block Memory Generator core GUI.*

Specifying Initial Memory Contents

The Block Memory Generator core supports memory initialization using a memory coefficient (COE) file or the default data option in the Vivado IDE, 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 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
; 8-bit wide by 16 deep RAM
memory_initialization_radix = 16;
memory_initialization_vector =
12, 34, 56, 78, AB, CD, EF, 12, 34, 56, 78, 90, AA, A5, 5A, BA;
```

Coefficients can be separated by a space, a comma, or by placing one value in each line with a carriage return.

Summary Tab

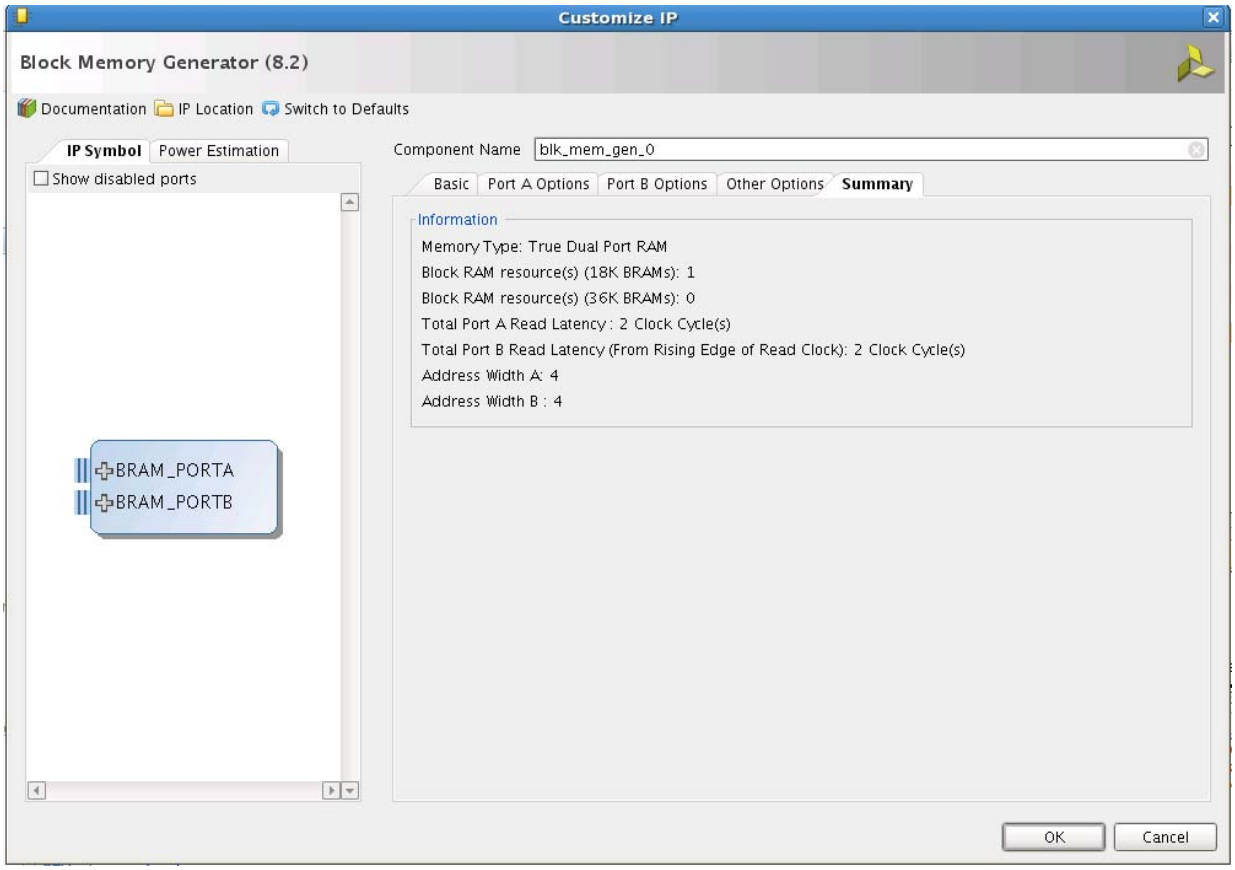

Figure 4-5: **Summary Tab**

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

- **Memory Type**: Reports the selected memory type.
- **Block RAM Resources**: Reports the exact number of 18K and 36K block RAM primitives which are used to construct the core.
- **Total Port A Read Latency**: The number of clock cycles for a Read operation for port A. This value is controlled by the optional output registers options for port A on the previous tab.
- **Total Port B Read Latency**: The number of clock cycles for a Read operation for port B. This value is controlled by the optional output registers options for port B on the previous tab.
- **Address Width**: The actual width of the address bus to each port.

Power Estimate Options Tab

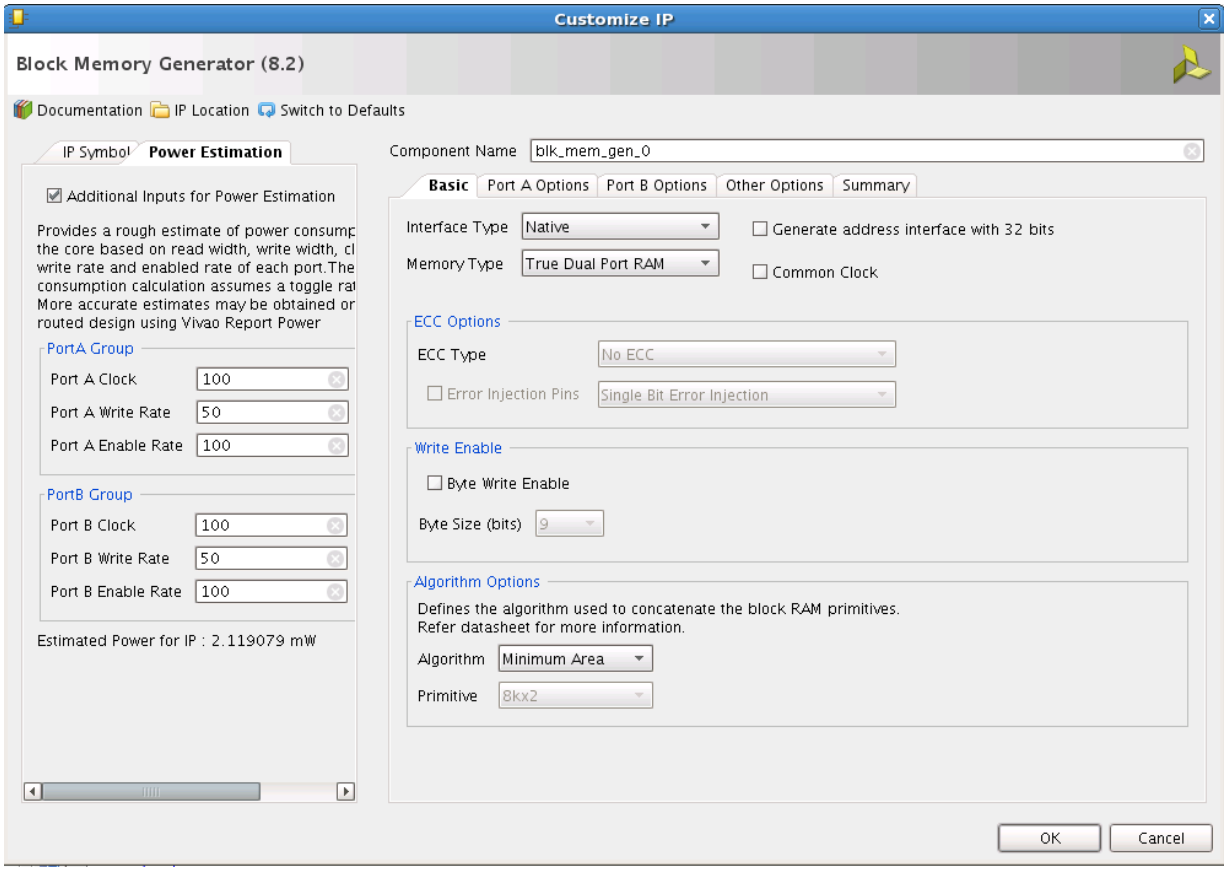

Figure 4-6: **Power Estimate Options Tab (Left Side)**

The Power Estimation tab on the left side of the Vivado IDE, shown in Figure 4-6, provides a rough estimate of power consumption for the core based on the configured Read width, Write width, clock rate, Write rate and enable rate of each port. The power consumption calculation assumes a toggle rate 50%. More accurate estimates can be obtained on the routed design using the XPower Analyzer tool. See www.xilinx.com/power for more information on the XPower Analyzer.

This tab has an option to provide "Additional Inputs for Power Estimation" apart from configuration parameters. You can enter the following parameters for power calculation:

- **Clock Frequency [A|B]**: The operating clock frequency of the two ports A and B respectively.
- **Write Rate [A|B]**: Write rate of ports A and B respectively.
- **Enable Rate [A|B]**: Average access rate of port A and B respectively.

Block RAM Usage

The Summary tab reports the actual number of 18K and 36K block RAM blocks to be 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 block RAM usage when using the low power algorithm requires a few more calculations:

- If the memory width is an integral multiple of the width of the widest available primitive for the chosen architecture, then the number of primitives used is calculated in the same way as the fixed primitive algorithm. The width and depth ratios are calculated using the width and depth of the widest primitive. For example, for a memory configuration of 2kx72, the width ratio is 2 and the depth ratio is 4 using the widest primitive of 512x36. As a result, the total available primitives used is 8.
- If the memory width is greater than an integral multiple of the widest primitive, then in addition to the above calculated primitives, more primitives are needed to cover the remaining width. This additional number is obtained by dividing the memory depth by the depth of the additional primitive chosen to cover the remaining width. For example, a memory configuration of 17kx37 requires one 512x36 primitive to cover the width of 36, and an additional 16kx1 primitive to cover the remaining width of 1. To cover the depth of 17K, 34 512x36 primitives and 2 16kx1 primitives are needed. As a result, the total number of primitives used for this configuration is 36.
- If the memory width is less than the width of the widest primitive, then the smallest possible primitive that covers the memory width is chosen for the solution. The total number of primitives used is obtained by dividing the memory depth by the depth of the chosen primitive. For example, for a memory configuration of 2kx32, the chosen primitive is 512x36, and the total number of primitives used is 2k divided by 512, which is 4.

When using the minimum area algorithm, it is not as easy to determine the exact block RAM count. This is because the actual algorithms perform complex manipulations to produce optimal solutions. The optimistic estimate of the number of 18K block RAMs is total memory bits divided by 18k (the total number of bits per primitive) rounded up. Given that this 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 general interconnect.

Note: Although 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.

Generating the AXI4 Interface Block Memory Generator Core

The AXI4 Interface Block Memory Generator GUI includes two additional tabs followed by Native BMG configuration tabs:

- Basic Tab
- AXI4 Tab

Basic Tab

The Basic tab defines the component name and provides the Interface Options for the core.

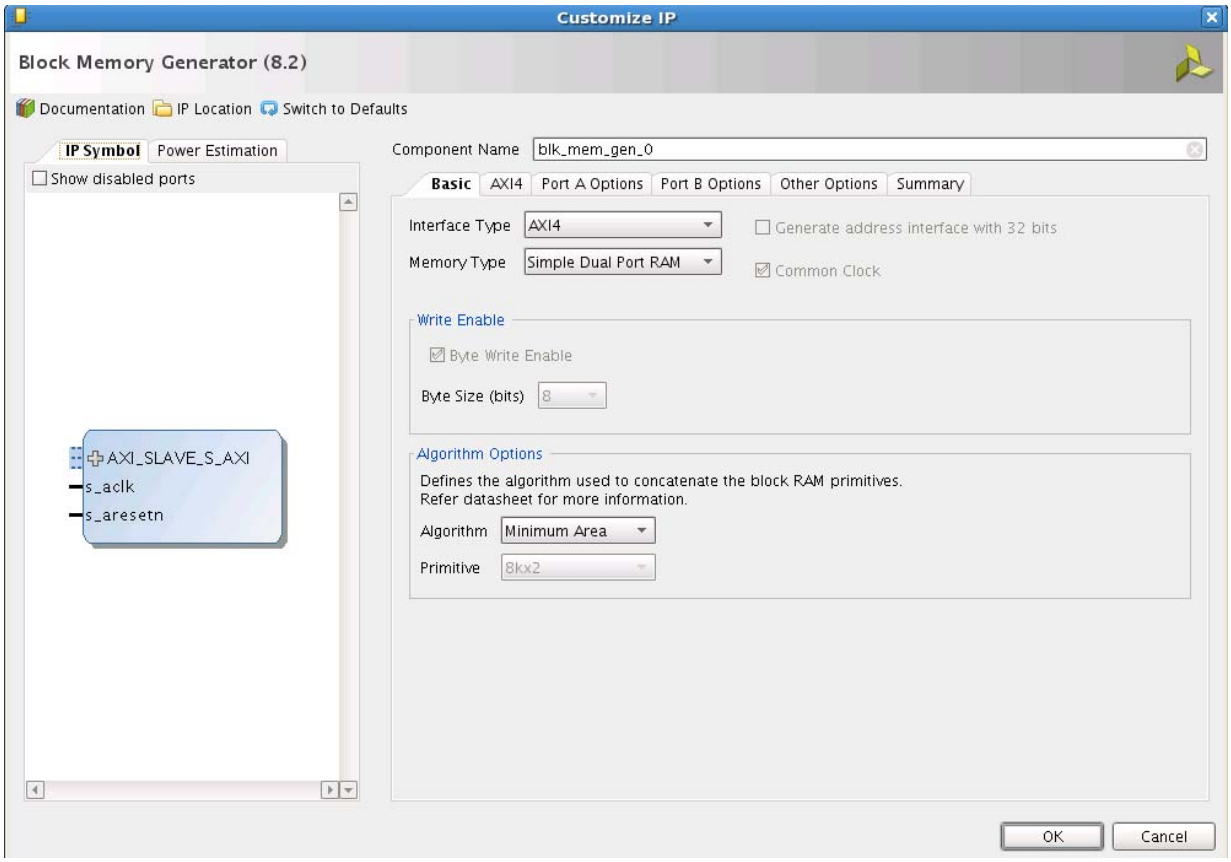

Figure 4-7: **AXI4 Interface Selection of Block Memory Generator**

Component Name

Base name of the output files generated for this core. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and "_".

- **Interface Type**:
  - AXI4: Implements an AXI4 Interface Block Memory Generator Core.

AXI4 Tab

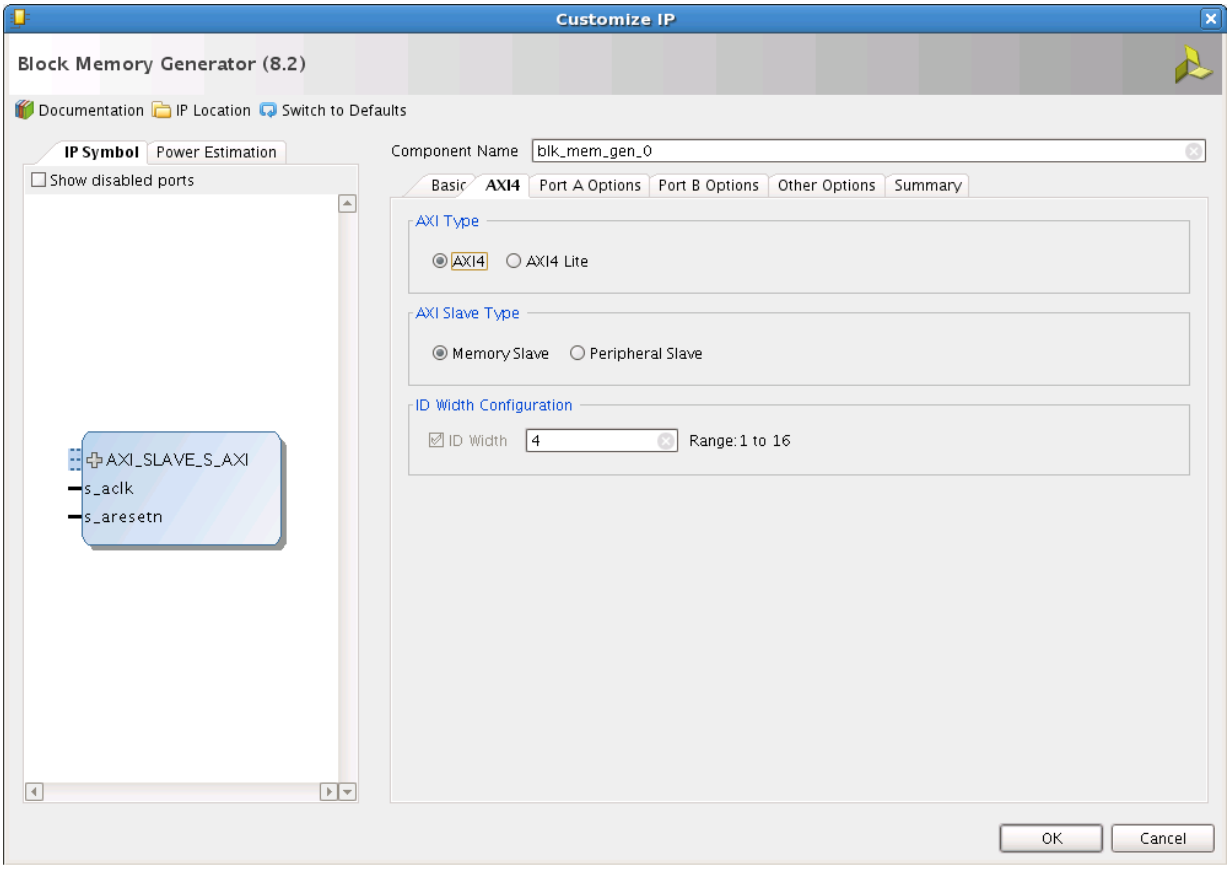

Figure 4-8: **AXI4 Interface Options**

- **AXI4 Interface Options**:
  - AXI4: Implements an AXI4 Block Memory Generator core.
  - AXI4-Lite: Implements an AXI4-Lite Block Memory Generator core.
- **AXI4 Slave Options**:
  - Memory Slave: Implements a AXI4 Interface Block Memory Generator core in Memory Slave mode
  - Peripheral Slave: Implements a AXI4 Interface Block Memory Generator core in Peripheral Slave mode
- **ID Width Configurations**:
  - ID Width: When enabled supports awid/bid/arid/rid generation, ID Width can be configured from 1 to 16 bits

Customizing the Core with IP Integrator

Figure 4-9 shows the Block Memory Generator core in IP integrator with default options.

IMPORTANT: *When using IP integrator, the AXI interface for the Block Memory Generator core is not available in standalone mode.*

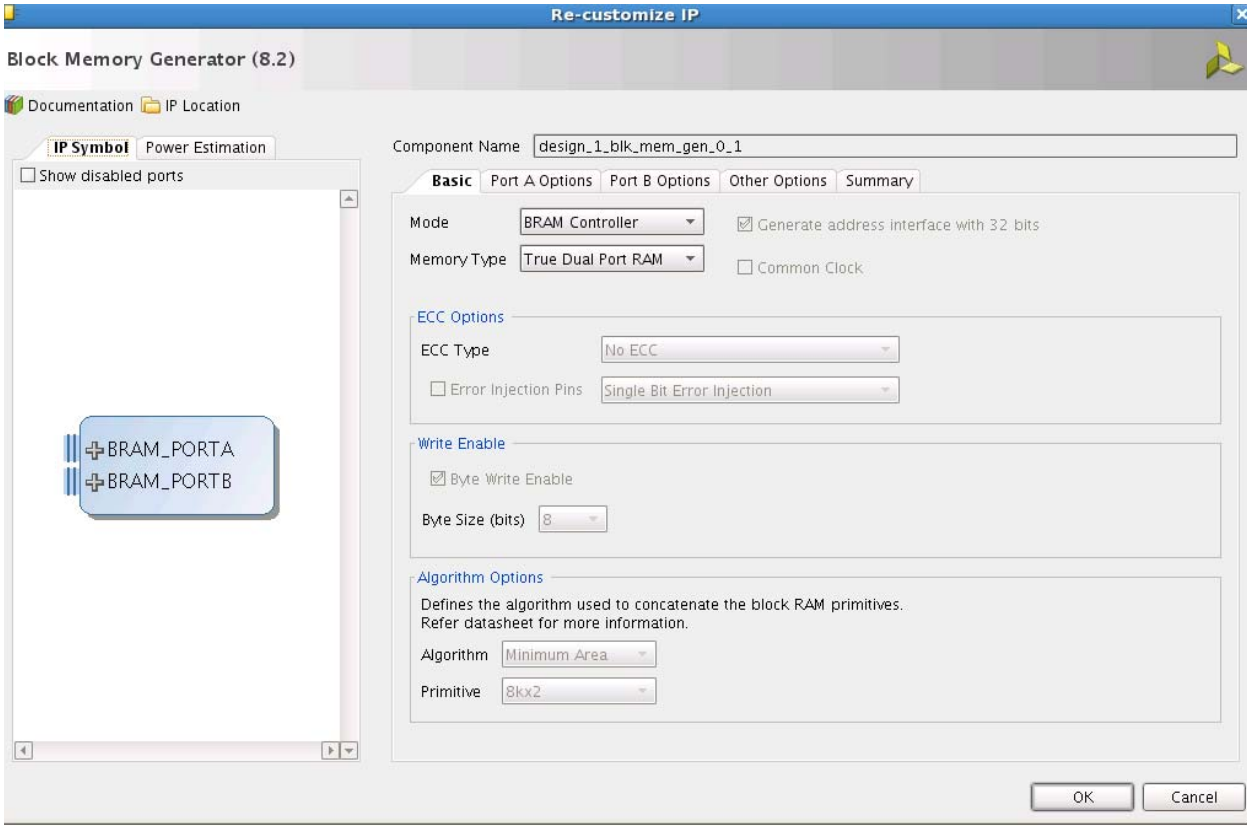

Figure 4-9: **Customizing using IP integrator**

- **Mode**: The mode in which the Block Memory Generator core is used in the IP integrator. There are two modes available when using IP integrator, and the default is **BRAM Controller**.
  - BRAM Controller Mode: This mode is selected when the Block Memory Generator core is connected to either an AXI BRAM Controller or a Local Memory Bus Interface controller.
  - Stand Alone Mode: select this mode for connecting to all other devices.

IMPORTANT: *When using IP integrator, the AXI Interface for the Block Memory Generator core is not available in stand alone mode.*

- **Memory Type**: In the BRAM Controller mode, the Memory Type options are **Single Port RAM** or **True Dual Port RAM**.

Mode and memory are the only options available. All other parameters are set during validation of the design through parameter propagation.

To minimize the use of the MUXes at the output of block RAM blocks in the core design, the primitives with a width of x1, x2, and x4 are used in the IP integrator for 7 series devices. For example, 32kx1, 16kx2 and 4kx4 primitives are used. This is implemented for 32 and 64-bit wide systems.

User Parameters

Table 4-1 shows the relationship between the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter/ Value[1]	User Parameter/Value[1]	Default Value
Basic Tab		
Interface Type	Interface_Type	Native
Memory Type	Memory_Type	Single_Port_RAM
Generate address interface with 32 bits	Enable_32bit_Address	FALSE
Common clock	Assume_Synchronous_Clk	FALSE
ECC Type	ecctype	NO_ECC
Error Injection Pins	Use_Error_Injection_Pins	FALSE
Byte Write Enable	Use_Byte_Write_Enable	FALSE
Byte Size	Byte_Size	9
Algorithm	Algorithm	Minimum_Area
Primitive	Primitive	8kx2
AXI4 Tab		
AXI Type	AXI_TYPE	AXI_Full
AXI Slave Type	AXI_Slave_Type	Memory_Slave
ID Width Configuration	Use_AXI_ID	FALSE
ID Width	AXI_ID_Width	4
Port A Options Tab		
Port A Write Width	Write_Width_A	16
Port A Write Depth	Write_Depth_A	16
Port A Read With	Read_Width_A	16
Operating Mode: Port A	Operating_Mode_A	WRITE_FIRST
Enable Port Type: Port A	Enable_A	Use_ENA_Pin
Primitive Output Register: Port A	Register_PortA_Output_of_Memory_Primitives	TRUE
Core Output Register: Port A	Register_PortA_Output_of_Memory_Coe	FALSE

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/ Value[1]	User Parameter/Value[1]	Default Value
SoftECC Input Register	Register_PortA_input_of_softecc	FALSE
REGCEA Pin	Use_REGCEA_Pin	FALSE
RSTA Pin	Use_RSTA_Pin	FALSE
Output Reset Value (HEX): Port A	Output_Reset_Value_A	0
Reset Memory Latch: Port A	Reset_Memory_Latch_A	FALSE
Reset Priority: Port A	Reset_Priority_A	CE
Port B Options tab		
Port B Write Width	Write_Width_B	16
Port B Read Width	Read_Width_B	16
Operating Mode: Port B	Operating_Mode_B	WRITE_FIRST
Enable Port Type: Port B	Enable_B	Always_Enabled
Primitive Output Register: Port B	Register_PortB_Output_of_Memory_Primitives	FALSE
Core Output Register: Port B	Register_PortB_Output_of_Memory_Coe	FALSE
SoftECC Output Register	Register_PortB_Output_of_Softecc	FALSE
REGCEB Pin	Use_REGCEB_Pin	FALSE
RSTB Pin	Use_RSTB_Pin	FALSE
Output Reset Value (HEX): Port B	Output_Reset_Value_B	0
Reset Memory Latch: Port B	Reset_Memory_Latch_B	FALSE
Reset Priority: Port B	Reset_Priority_B	CE
Enable ECC PIPE	EN_ECC_PIPE	FALSE
Other Options Tab		
Pipe Line Stages within Mux	Pipeline_Stages	0
Load Init File	Load_Init_File	FALSE
COE File	Coe_File	No_coe_file_loaded
Fill Remaining Memory Locations	Fill_Remaining_Memory_Locations	FALSE
Remaining Memory Locations (Hex)	Remaining_Memory_Locations	0
Collision Warnings	Collision_Warnings	ALL
Disable Collision warnings	Disable_Collision_warnings	FALSE
Disable Out of Range warnings	Disable_Out_of_Range_warnings	FALSE
Sleep	EN_SLEEP_PIN	FALSE

Table 4-1: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter/ Value[1]	User Parameter/Value[1]	Default Value
Power Estimation Tab		
Additional Inputs for Power Estimation	Additional_Inputs_for_Power_Estimation	FALSE
Port A Clock	Port_A_Clock	100
Port A Write Rate	Port_A_Write_Rate	50
Port A Enable Rate	Port_A_Enable_Rate	100
Port B Clock	Port_B_Clock	0
Port B Write Rate	Port_B_Write_Rate	0
Port B Enable Rate	Port_B_Enable_Rate	0

Notes:

1. Parameter values are listed in the table where the Vivado IDE parameter value differs from the user parameter value. Such values are shown in this table as indented below the associated parameter.

Constraining the Core

There are no constraints associated with this core.

Simulation

For details, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 5].

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Detailed Example Design

This chapter contains details about the Block Memory Generator core example design using the Vivado® Design Suite.

The Block Memory Generator core example design includes the following:

- HDL wrapper which instantiates the Block Memory Generator core netlist
- Block Memory Generator core constraint file

The Block Memory Generator core example design has been tested with Vivado Design Suite.

Test Bench

This chapter contains details about the Block Memory Generator core test bench. The demonstration test bench is a straightforward VHDL file to exercise the example design and the core itself. The test bench consists of the following:

- Clock generators
- Address and data generator module
- Stimulus generator module
- Data checker
- Protocol checks for the AXI4 protocol

Figure 6-1 shows a block diagram of the demonstration test bench.

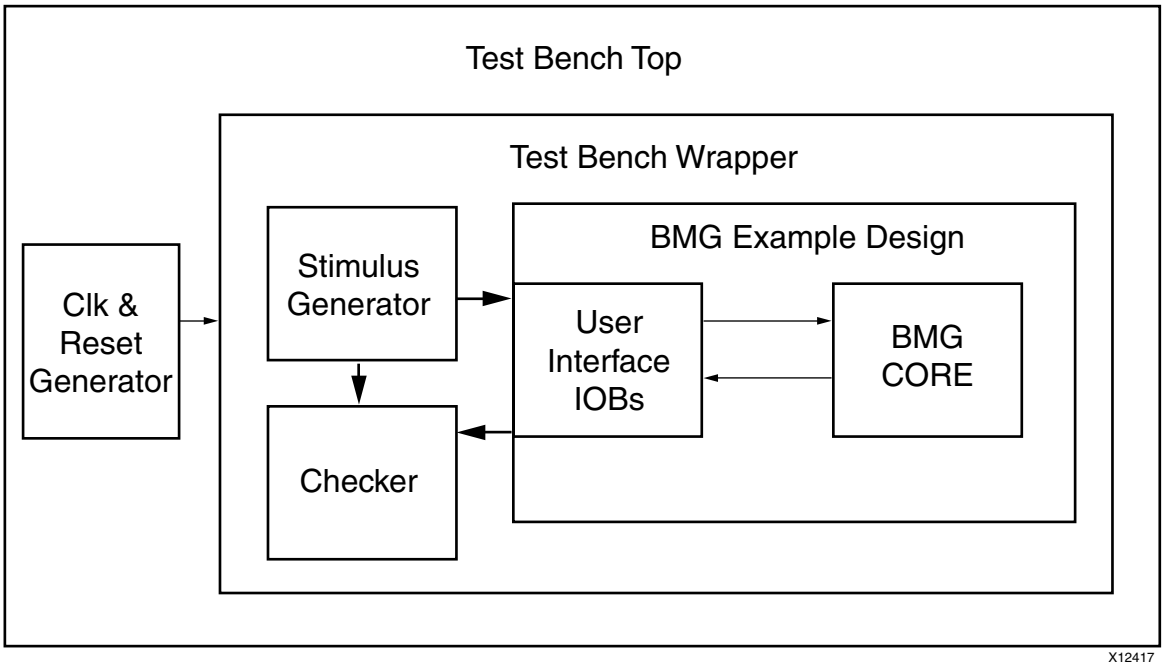

Figure 6-1: **Test Bench Block Diagram**

Core with Native Interface

The demonstration test bench in a core with a Native interface performs the following tasks:

- Generates input clock signals.
- Applies a reset to the example design. Address is generated in a linear, incremented format.
- Generates pseudo-random data and inputs this data to the Block Memory Generator core data input port.
- Compares the outgoing data during reads with another pseudo-random generator with the same seed.
- Exercises the core with 256 write and read transactions.

Core with AXI4 Interface

The demonstration test bench in a core with an AXI4 interface performs the following tasks:

- Generates input clock signals.
- Applies a reset to the example design.
- Generates a stimulus only when incremented by burst. Narrow transactions are not exercised.
- Generates addresses in a linear, incremented format.
- Generates pseudo-random data for `wdata`.
- Determines the length (`awlen`/`arlen`) randomly.
- Compares RDATA during a read with another pseudo-random generator with the same seed as WDATA random data generator.
- Checks for AXI4 protocol violations with a basic AXI4 protocol checker.
- Exercises the core with 256 AXI4 write and read transactions.

Messages and Warnings

When the functional or timing simulation has completed successfully, the test bench displays the following message, and it is safe to ignore this message.

```
Failure: Test Completed Successfully
```

Appendix A

Verification, Compliance, and Interoperability

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.

Note: The simulation models of the Block Memory Generator IP are delivered unencrypted.

Simulation

The Block Memory Generator core has been tested with the Xilinx® Vivado® Design Suite, Xilinx XSIM, Cadence Incisive Enterprise Simulator (IES), Synopsys VCS and VCS MX and Mentor Graphics Questa SIM.

Migrating and Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

TIP: *For new designs, Xilinx recommends that you use the most recent version of the Block Memory Generator core for all block memory function requirements.*

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, *see the ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 8].

Upgrading in the Vivado Design Suite

Auto Upgrade Feature

The Block Memory Generator core has an auto upgrade feature for updating older versions of the Block Memory Generator core to the latest version. The auto upgrade feature can be seen by right clicking the already generated older version of Block Memory Generator core in the **Project IP** tab of the Vivado Design Suite. This feature can be initiated by selecting **Upgrade to Latest Version and Regenerate**. This option upgrades an older Block Memory Generator core to the latest version. Any earlier version of Block Memory Generator core can be upgraded to v8.0.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools.

TIP: *If the IP generation halts with an error, there might be a license issue. See Licensing and Ordering Information in Chapter 1 for more details.*

Finding Help on Xilinx.com

To help in the design and debug process when using the Block Memory Generator, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

This product guide is the main document associated with the Block Memory Generator. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx® Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Answer Records

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records can be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name

- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Master Answer Record for the Block Memory Generator

AR 50918

Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance. To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Additional Resources.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Note: Access to WebCase is not available in all cases. Login to the WebCase tool to see your specific support options.

Debug Tools

Vivado Lab Edition

Vivado® Lab Edition inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Edition also allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature in the Vivado IDE is used for logic debugging and validation of a design running in Xilinx.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

Simulation Debug

In the VHDL behavioral model, the memory contents can be viewed using the array `memory_i` when the DEBUG constant is set to 1.

For details about simulating a design in the Vivado Design Suite, see the *Vivado Logic Simulation User Guide* (UG900) [Ref 5].

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues.

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

Native Block Memory Generator Supplemental Information

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

- Low Power Designs: Provides information on the Low Power algorithm and methods that you can follow to optimize power consumption in block RAM designs.
- Native Block Memory Generator 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.

Low Power Designs

The Block Memory Generator core also supports a Low Power implementation algorithm. When this option is selected, the configuration of the core is optimized to minimize dynamic power consumption. This contrasts with the Minimum Area algorithm, which optimizes the core implementation with the sole purpose of minimizing resource utilization.

The Low Power algorithm reduces power through the following mechanisms:

- Minimizing the number of block RAMs enabled for a Write or Read operation for a given memory size.
- Unlike the Minimum Area algorithm, smaller block RAM blocks are not grouped to form larger blocks in the Low Power algorithm. For example, two 18K block RAMs are not combined to form a 36K block RAM in Virtex®-7 devices.
- The “Always Enabled” option is not available for the Port A and Port B enable pins. This prevents the block RAMs from being enabled at all times.
- The NO_CHANGE mode is set as the default operating mode.

XPE is a pre-implementation power estimation tool appropriate for estimating power requirements in the early stages of a design. For more accurate power consumption estimates and power analysis, the Xilinx Power Analyzer tool (XPA) available in Vivado tools can be run on designs after place and route.

Power data shown in the following tables was collected assuming a 50% toggle rate and 50% Write rate. A frequency of 300 MHz was specified for Virtex-7 devices.

Note: Always use the latest version of each target architecture XPE spreadsheet to get the most accurate estimate.

Besides using the Low Power algorithm, the following design considerations are recommended for power optimizations:

- The Low Power algorithm disables the "Always Enabled" option for the Port A and Port B enable pins, and you are forced to have these pins at the output (the **Use EN[A|B] Pin** option). These pins must not be permanently tied to 1 if it is desired that power be conserved. Each port enable pin must be asserted high only when that port of the block RAM needs to be accessed.
- Use of output registers improves performance, but also increases power consumption. Even if used in the design, the output registers should be disabled when the block RAMs are not being accessed.
- You can choose the operating mode even in the Low Power algorithm based upon design requirements; however it is recommended that the default operating mode of the Low Power algorithm (NO_CHANGE mode) be used. This mode results in lower power consumption as compared to the WRITE_FIRST and READ_FIRST modes.

Native Block Memory Generator SIM Parameters

Table D-1 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.

Table D-1: **Native Interface SIM Parameters**

	SIM Parameter	Type	Values	Description
1	C_FAMILY	String	virtexu, kintexu, virtex7, kintex7, artix7, zynq7000	Target device family
2	C_XDEVICEFAMILY	String	virtexu, kintexu, virtex7, kintex7, artix7, zynq7000	Finest resolution target family derived from the parent C_FAMILY
3	C_ELABORATION_DIR	String		Elaboration Directory
4	C_USE_BRAM_BLOCK	Integer	0: Standalone 1: BRAM Controller	Used in IP integrator Mode only. Defines whether the BMG works in conjunction with the block RAM Controller or if the core works in standalone IP mode.

Table D-1: **Native Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
5	C_ENABLE_32BIT_ADDRESS	Integer	0: Address interface defined by memory depth 1: Generates 32-bit address interface	Defines whether the BMG core generates 32-bit address interface for block RAM Controller mode.
6	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 ROM	Type of memory
7	C_ALGORITHM	Integer	0 (selectable primitive), 1 (minimum area), 2 (low power)	Type of algorithm
8	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, determines which type of primitive to use to build memory
9	C_BYTE_SIZE	Integer	9, 8	Defines size of a byte: 9 bits or 8 bits
10	C_SIM_COLLISION_CHECK	String	NONE, GENERATE_X_ONLY, ALL, WARNINGS_ONLY	Defines warning collision checks in structural/UNISIM simulation model
11	C_COMMON_CLOCK	Integer	0, 1	Determines whether to optimize behavioral models for Read/ Write accesses and collision checks by assuming clocks are synchronous. Xilinx recommends you to set this option to 0 when both the clocks are not synchronous, This option will allow the models to function properly.
12	C_DISABLE_WARN_BHV_ COLL	Integer	0, 1	Disables the behavioral model from generating warnings due to Read-Write collisions
13	C_DISABLE_WARN_BHV_ RANGE	Integer	0, 1	Disables the behavioral model from generating warnings due to address out of range
14	C_LOAD_INIT_FILE	Integer	0, 1	Defined whether to load initialization file

Table D-1: **Native Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
15	C_INIT_FILE_NAME	String	""	Name of initialization file (MIF format)
16	C_USE_DEFAULT_DATA	Integer	0, 1	Determines whether to use default data for the memory
17	C_DEFAULT_DATA	String	0	Defines a default data for the memory
18	C_HAS_MEM_OUTPUT_ REGS_A	Integer	0, 1	Determines whether port A has a register stage added at the output of the memory latch
19	C_HAS_MEM_OUTPUT_ REGS_B	Integer	0,1	Determines whether port B has a register stage added at the output of the memory latch
20	C_HAS_MUX_OUTPUT_ REGS_A	Integer	0,1	Determines whether port A has a register stage added at the output of the memory core
21	C_HAS_MUX_OUTPUT_ REGS_B	Integer	0,1	Determines whether port B has a register stage added at the output of the memory core
22	C_MUX_PIPELINE_STAGES	Integer	0,1,2,3	Determines the number of pipeline stages within the MUX for both port A and port B
23	C_WRITE_WIDTH_A	Integer	1 to 4608	Defines width of Write port A
24	C_READ_WIDTH_A	Integer	1 to 4608	Defines width of Read port A
25	C_WRITE_DEPTH_A	Integer	2 to 128k(1)	Defines depth of Write port A
26	C_READ_DEPTH_A	Integer	2 to 128k(1)	Defines depth of Read port A
27	C_ADDRA_WIDTH	Integer	1 to 24	Defines the width of address A
28	C_WRITE_MODE_A	String	Write_First, Read_first, No_change	Defines the Write mode for port A
29	C_HAS_ENA	Integer	0, 1	Determines whether port A has an enable pin
30	C_HAS_REGCEA	Integer	0, 1	Determines whether port A has an enable pin for its output register
31	C_HAS_RSTA	Integer	0, 1	Determines whether port A has reset pin
32	C_INITA_VAL	String	0	Defines initialization/power-on value for port A output

Table D-1: **Native Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
33	C_USE_BYTE_WEA	Integer	0, 1	Determines whether byte-Write feature is used on port A For True Dual-Port configurations, this value is the same as C_USE_BYTE_WEB, as there is only a single byte Write enable option provided
34	C_WEA_WIDTH	Integer	1 to 512	Defines width of `WEA` pin for port A
35	C_WRITE_WIDTH_B	Integer	1 to 4608	Defines width of Write port B
36	C_READ_WIDTH_B	Integer	1 to 4608	Defines width of Read port B
37	C_WRITE_DEPTH_B	Integer	2 to 128k	Defines depth of Write port B
38	C_READ_DEPTH_B	Integer	2 to 128k	Defines depth of Read port B
39	C_ADDRB_WIDTH	Integer	1 to 24	Defines the width of address B
40	C_WRITE_MODE_B	String	Write_First, Read_first, No_change	Defines the Write mode for port B
41	C_HAS_ENB	Integer	0, 1	Determines whether port B has an enable pin
42	C_HAS_REGCEB	Integer	0, 1	Determines whether port B has an enable pin for its output register
43	C_HAS_RSTB	Integer	0, 1	Determines whether port B has reset pin
44	C_INITB_VAL	String	...	Defines initialization/power-on value for port B output
45	C_USE_BYTE_WEB	Integer	0, 1	Determines whether byte-Write feature is used on port B This value is the same as C_USE_BYTE_WEA, as there is only a single byte Write enable provided
46	C_WEB_WIDTH	Integer	1 to 512	Defines width of `web` pin for port B
47	C_USE_ECC	Integer	0,1	Determines ECC options: • 0 = No ECC • 1 = ECC
48	C_RST_PRIORITY_A	String	([CE, SR] : CE)	In the absence of output registers, this selects the priority between the RAM ENA and the rsta pin. When using output registers, this selects the priority between REGCEA and the rsta pin.

Table D-1: **Native Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
49	C_RSTRAM_A11	Integer	([0,1] : 1)	If the value of this generic is 1, both the memory latch and the embedded primitive output register of Port A are reset. If this value is 0, only the embedded output register of the primitive is reset (the memory latch is not reset). Setting this option to 1 results in the output reset value being asserted for two clock cycles.
50	C_RST_PRIORITY_B	String	([CE, SR] : CE)	In the absence of output registers, this selects the priority between the RAM ENB and the rstb pin. When using output registers, this selects the priority between REGCEB and the rstb pin.
51	C_RSTRAM_B	Integer	([0,1] : 1)	If the value of this generic is 1, both the memory latch and the embedded primitive output register of Port B are reset. If this value is 0, only the embedded output register of the primitive is reset (the memory latch is not reset). Setting this option to 1 results in the output reset value being asserted for two clock cycles.
52	C_HAS_INJECTERR	Integer	([0,1,2,3] : 0)	Determines the type of error injection: • 0 = No Error Injection • 1 = Single-Bit Error Injection Only • 2 = Double-Bit Error Injection Only • 3 = Both Single- and Double-Bit Error Injection
53	C_USE_SOFTECC	Integer	0,1	Determines Soft ECC options: • 0 = No Soft ECC • 1 = Soft ECC
54	C_HAS_SOFTECC_INPUT_REGS_A	Integer	0,1	Registers the input ports in the design when Soft ECC is enabled: • 0 = Registers not enabled • 1 = Registers enabled

Table D-1: **Native Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
55	C_HAS_SOFTECC_OUTPUT_REGS_B	Integer	0,1	Registers the input ports in the design when Soft ECC is enabled: • 0 = Registers not enabled • 1 = Registers enabled
56	C_INIT_FILE	String		Name of the Initialization file (MEM format)
57	C_CTRL_ECC_ALGO	String		Carries the ECC algorithm information from the master to which Block Memory Generator core is connected in IP integrator.
58	C_EN_ECC_PIPE	Integer		Enables of the eccpipece port: • 0 = eccpipece port is not available • 1 = eccpipece port is available
59	C_EN_SLEEP_PIN	Integer		Enables the sleep port: • 0 = sleep port is not available • 1 = sleep is available
60	C_COUNT_36K_BRAM(2)	String		Number of 36K BRAMs used in the design.
61	C_COUNT_18K_BRAM(2)	String		Number of 18K BRAMs used in the design.
62	C_EST_PWER_SUMMARY(2)	String		The estimated power of the design
63	C_EN_RDADDRA_CHG	Integer	0	Specifies if the port A read address compare feature is turned on.
64	C_EN_RDADDRB_CHG	Integer	0	Specifies if the port B read address compare feature is turned on.

Notes:

1. Depth is restricted based on Write width.
2. Used only in the summary log generation. This parameter is not used in the Block Memory Generator core design or in the simulation models.

AXI4 Interface Block Memory Generator SIM Parameters

Table D-2: **AXI4 Interface SIM Parameters**

	SIM Parameter	Type	Values	Description
1	C_FAMILY	String	ultrascale and 7 series	Target device family.
2	C_XDEVICEFAMILY	String	ultrascale and 7 series	Finest resolution target family.

Table D-2: **AXI4 Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
3	C_ELABORATION_DIR	String		Elaboration Directory.
4	C_INTERFACE_TYPE	Integer	1: AXI4	Determines the type of interface selected.
5	C_AXI_TYPE	Integer	0: AXI4_Lite 1: AXI4_Full	Determines the type of the AXI4 interface.
6	C_AXI_SLAVE_TYPE	Integer	0: Memory Slave 1: Peripheral Slave	Determines the type of the AXI4 Slave interface.
7	C_HAS_AXI_ID	Integer	0: Core does not use AXI4 ID 1: Core uses AXI4 ID	Determines whether the AXI4 interface supports AXI4 ID.
8	C_AXI_ID_WIDTH	Integer	1 to 16	Defines the AXI4 ID value.
9	C_MEM_TYPE	Integer	1: Simple Dual-Port RAM	Type of memory.
10	C_ALGORITHM	Integer	0 (selectable primitive), 1 (minimum area), 2 (low power)	Type of algorithm.
11	C_PRIM_TYPE	Integer	4 (18-bit wide) 5 (36-bit wide) 6 (72-bit wide)	If fixed primitive algorithm is chosen, determines which type of primitive to use to build memory.
12	C_BYTE_SIZE	Integer	8	Defines size of a byte: 8 bits.
13	C_COMMON_CLOCK	Integer	1	Determines whether to optimize behavioral models for Read/Write accesses and collision checks by assuming clocks are synchronous.
14	C_DISABLE_WARN_BHV_COLL	Integer	0, 1	Disables the behavioral model from generating warnings due to Read Write collisions.
15	C_DISABLE_WARN_BHV_RANGE	Integer	0, 1	Disables the behavioral model from generating warnings due to address out of range.
16	C_LOAD_INIT_FILE	Integer	0, 1	Defined whether to load initialization File.
17	C_INIT_FILE_NAME	String	""	Name of initialization file (MIF format).
18	C_USE_DEFAULT_DATA	Integer	0, 1	Determines whether to use default data for the memory.
19	C_DEFAULT_DATA	String	0	Defines a default data for the Memory.

Table D-2: **AXI4 Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
20	C_HAS_MEM_OUTPUT_REGS_A	Integer	0	Determines whether port A has a register stage added at the output of the memory latch.
21	C_HAS_MEM_OUTPUT_REGS_B	Integer	0	Determines whether port B has a register stage added at the output of the memory latch.
22	C_HAS_MUX_OUTPUT_REGS_A	Integer	0	Determines whether port A has a register stage added at the output of the memory core.
23	C_HAS_MUX_OUTPUT_REGS_B	Integer	0	Determines whether port B has a register stage added at the output of the memory core.
24	C_MUX_PIPELINE_STAGES	Integer	0	Determines the number of pipeline stages within the MUX for both port A and port B.
25	C_WRITE_WIDTH_A	Integer	32 to 256	Defines width of Write port A.
26	C_READ_WIDTH_A	Integer	32 to 256	Defines width of Read port A.
27	C_WRITE_DEPTH_A	Integer	1 to 128k	Defines depth of Write port A.
28	C_READ_DEPTH_A	Integer	1 to 128k	Defines depth of Read port A.
29	C_ADDRA_WIDTH	Integer	1 to 32	Defines the width of address A.
30	C_WRITE_MODE_A	String	Read_first	Defines the Write mode for port A.
31	C_HAS_ENA	Integer	1	Determines whether port A has an enable pin.
32	C_HAS_REGCEA	Integer	0	Determines whether port A has an enable pin for its output register.
33	C_HAS_RSTA	Integer	0	Determines whether port A has reset pin.
34	C_INITA_VAL	String	0	Defines initialization/power-on value for port A output.
35	C_USE_BYTE_WEA	Integer	1	Determines whether byte-Write feature is used on port A.
36	C_WEA_WIDTH	Integer	1 to 128	Defines width of wea pin for port A.
37	C_WRITE_WIDTH_B	Integer	32 to 256	Defines width of Write port B.
38	C_READ_WIDTH_B	Integer	32 to 256	Defines width of Read port B.
39	C_WRITE_DEPTH_B	Integer	1 to 128k	Defines depth of Write port B.
40	C_READ_DEPTH_B	Integer	1 to 128k	Defines depth of Read port B.
41	C_ADDRB_WIDTH	Integer	1 to 32	Defines the width of address B.

Table D-2: **AXI4 Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
42	C_WRITE_MODE_B	String	Read_first,	Defines the Write mode for port B.
43	C_HAS_ENB	Integer	1	Determines whether port B has an enable pin.
44	C_HAS_REGCEB	Integer	0	Determines whether port B has an enable pin for its output register.
45	C_HAS_RSTB	Integer	1	Determines whether port B has reset pin.
46	C_INITB_VAL	String	0	Defines initialization/power-on value for port B output.
47	C_USE_BYTE_WEB	Integer	1	Determines whether byte-Write feature is used on port B. This value is the same as C_USE_BYTE_WEA, as there is only a single byte Write enable provided.
48	C_WEB_WIDTH	Integer	1 to 128	Defines width of web pin for port B.
49	C_USE_ECC	Integer	0	Determines ECC options: • 0 = No ECC • 1 = ECC
50	C_RST_TYPE	String	SYNC	Type of Reset
51	C_RST_PRIORITY_A	String	CE	In the absence of output registers, this selects the priority between the RAM ENA and the rsta pin. When using output registers, this selects the priority between REGCEA and the rsta pin.
52	C_RSTRAM_A	Integer	0	If the value of this generic is 1, both the memory latch and the embedded primitive output register of Port A are reset. If this value is 0, only the embedded output register of the primitive is reset (the memory latch is not reset). Setting this option to 1 results in the output reset value being asserted for two clock cycles.

Table D-2: **AXI4 Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
53	C_RST_PRIORITY_B	String	CE	In the absence of output registers, this selects the priority between the RAM ENB and the rstb pin. When using output registers, this selects the priority between REGCEB and the rstb pin.
54	C_RSTRAM_B	Integer	0	If the value of this generic is 1, both the memory latch and the embedded primitive output register of Port B are reset. If this value is 0, only the embedded output register of the primitive is reset (the memory latch is not reset). Setting this option to 1 results in the output reset value being asserted for two clock cycles.
55	C_HAS_INJECTERR	Integer	0	Determines the type of error injection: • 0 = No Error Injection • 1 = Single-Bit Error Injection Only • 2 = Double-Bit Error Injection Only • 3 = Both Single- and Double-Bit Error Injection
56	C_USE_SOFTECC	Integer	0	Determines Soft ECC options: • 0 = No Soft ECC • 1 = Soft ECC
57	C_HAS_SOFTECC_INPUT _REGS_A	Integer	0	Registers the input ports in the design when Soft ECC is enabled: • 0 = Registers not enabled • 1 = Registers enable
58	C_HAS_SOFTECC_OUTPUT _REGS_B	Integer	0	Registers the input ports in the design when Soft ECC is enabled: • 0 = Registers not enabled • 1 = Registers enabled
59	C_SIM_COLLISION_CHECK	String	NONE, GENERATE_X_ONLY, ALL, WARNINGS_ONLY	Defines warning collision checks in structural/UNISIM simulation model.

Table D-2: **AXI4 Interface SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Values	Description
60	C_EN_RDADDRA_CHG	Integer	0	Specifies if the port A read address compare feature is turned on.
61	C_EN_RDADDRB_CHG	Integer	0	Specifies if the port B read address compare feature is turned on.

Output Register Configurations

The Block Memory Generator core provides optional output registers that can be selected for port A and port B separately, and that might improve the performance of the core. Figure D-1 shows the Optional Output Registers section of the Block Memory Generator core GUI.

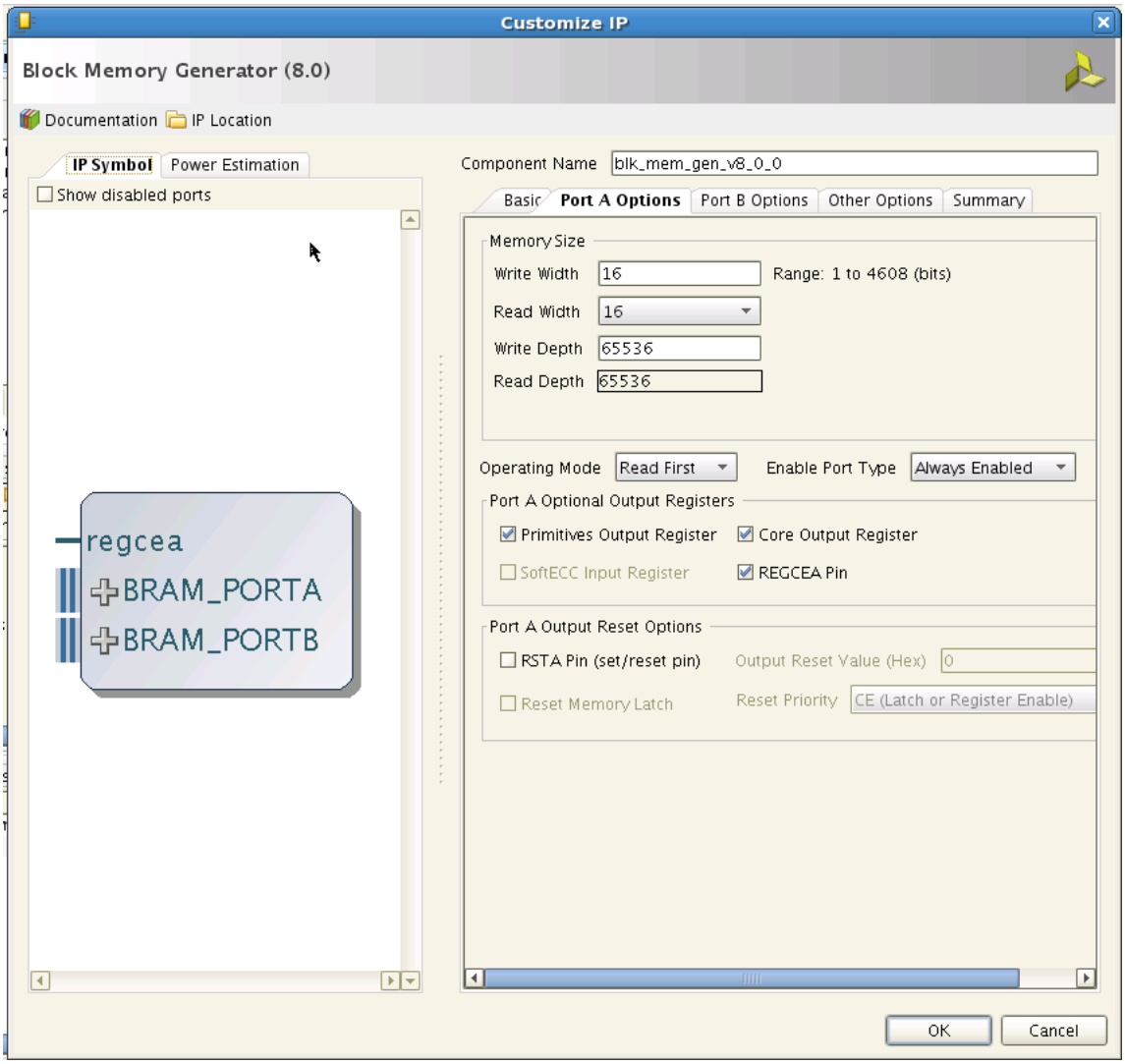

Figure D-1: **Optional Output Registers Section**

To tailor register options, two selections for port A and two selections for port B are provided on Port A/B Options of the Vivado IP Catalog GUI in the Optional [A|B] Output Registers section. The embedded output registers for the corresponding port(s) are enabled when Register Port [A|B] Output of Memory Primitives is selected. Similarly, registers at the output of the core for the corresponding port(s) are enabled by selecting Register Port [A|B] Output of Memory Core. Figure D-2 through Figure D-6 illustrate the output register configurations.

When only Register Port [A|B] Output of Memory Primitives and the corresponding Use RST[A|B] Pin are selected, the special reset behavior (option to reset the memory latch, in addition to the primitive output register) becomes available. For more information on this reset behavior, see Special Reset Behavior in Chapter 3.

Memory with Primitive and Core Output Registers

With both Register Port [A|B] Output of Memory Primitives and the corresponding Register Port [A|B] Output of Memory Core selected, a memory core is generated with the embedded output registers and a register on the output of the core for the selected port(s), as shown in Figure D-2. This configuration can provide improved performance for building large memory constructs.

Port A	or	Port B
☑ Register Port A Output of Memory Primitives		☑ Register Port B Output of Memory Primitives
☑ Register Port A Output of Memory Core		☑ Register Port B Output of Memory Core

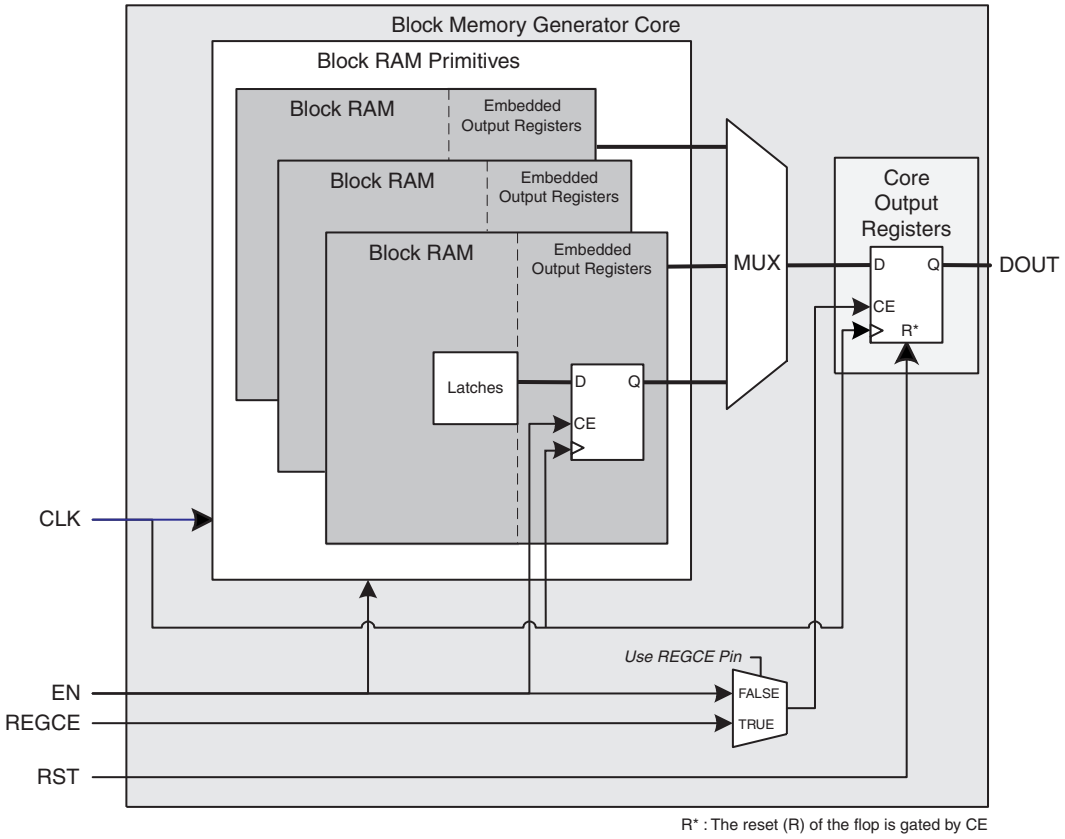

Figure D-2: **Block Memory Generated with Register Port [A|B] Output of Memory Primitives and Register Port [A|B] Output of Memory Core Enabled**

Memory with Primitive Output Registers and without Special Reset Behavior Option

If Use rsta Pin (set/reset pin) or Use rstb Pin (set/reset pin) is selected, and the special reset behavior (to reset the memory latch besides the primitive output register) is not selected, then the input reset signal is only connected to the RSTREG pin of the block RAM primitive, as illustrated in Figure D-3.

Port A	or	Port B
☑ Register Port A Output of Memory Primitives		☑ Register Port B Output of Memory Primitives
☐ Register Port A Output of Memory Core		☐ Register Port B Output of Memory Core
☑ Use rsta Pin (set/reset pin)		☑ Use rstb Pin (set/reset pin)
☐ Reset Memory Latch		☐ Reset Memory Latch

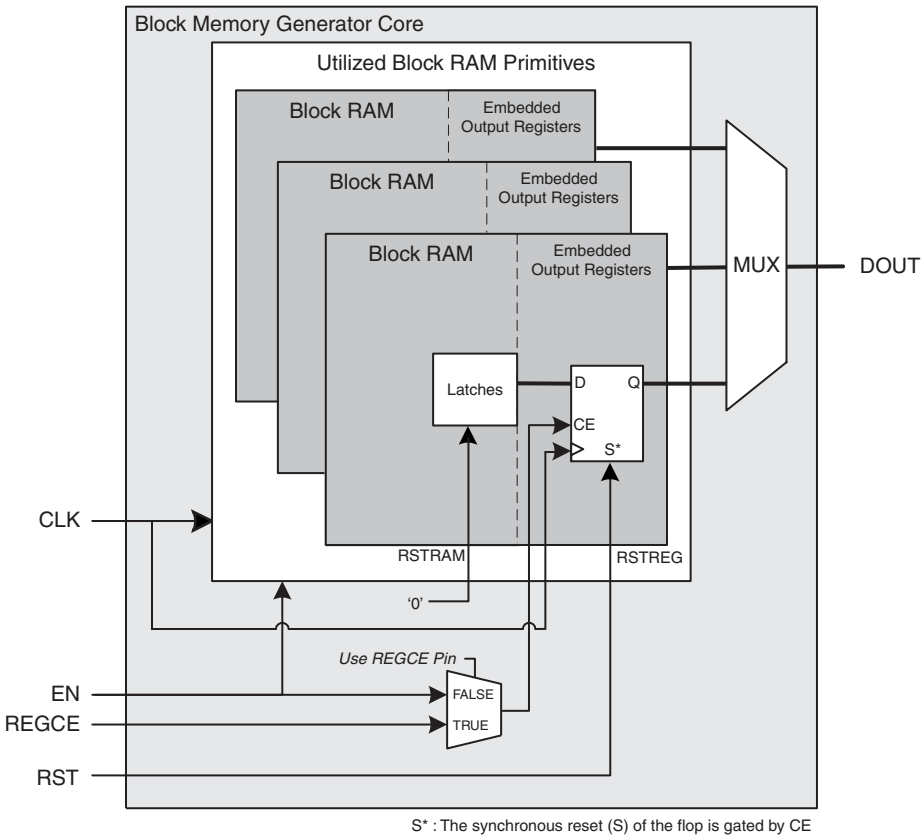

Figure D-3: **Block Memory Generated with Register Port [A|B] Output of Memory Primitives Enabled and without Special Reset Behavior**

Memory with Primitive Output Registers and with Special Reset Behavior Option

When Register Port [A|B] Output of Memory Primitives, Use rsta Pin (set/reset pin) or Use rstb Pin (set/reset pin), and the special reset behavior (to reset the memory latch besides the primitive output register) are selected, then the input reset signal is connected to both the RSTRAM and RSTREG pins of the block RAM primitive, as illustrated in Figure D-4.

Port A	or	Port B
☑ Register Port A Output of Memory Primitives		☑ Register Port B Output of Memory Primitives
☐ Register Port A Output of Memory Core		☐ Register Port B Output of Memory Core
☑ Use rsta Pin (set/reset pin)		☑ Use rstb Pin (set/reset pin)
☑ Reset Memory Latch		☑ Reset Memory Latch

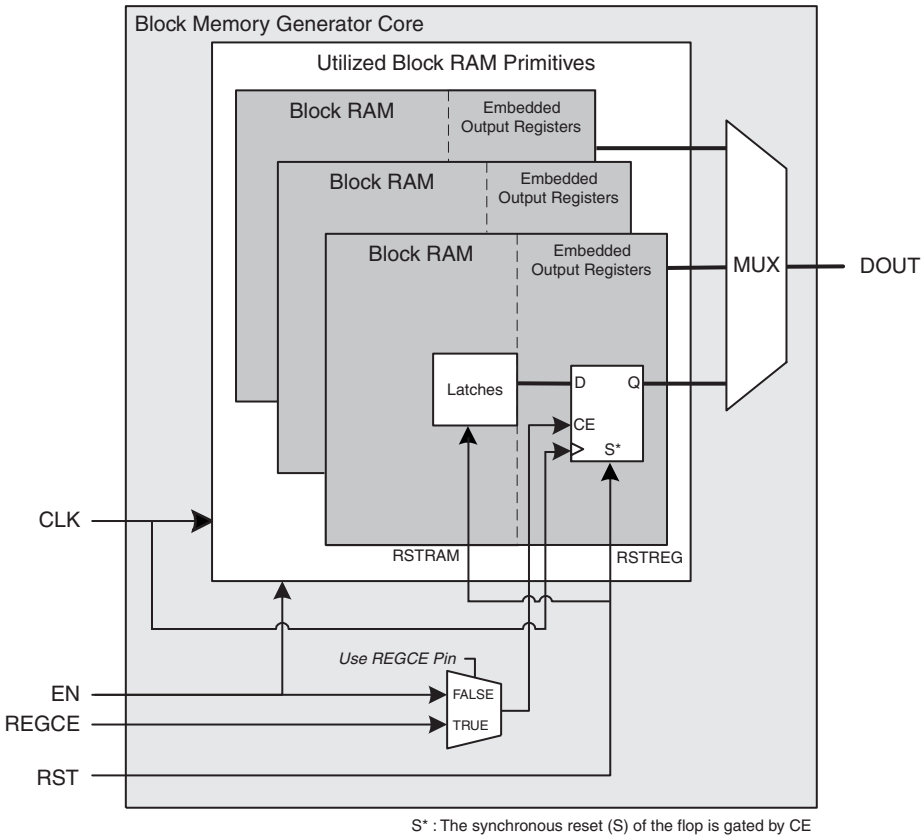

Figure D-4: **Block Memory Generated with Register Port [A|B] Output of Memory Primitives Enabled and with Special Reset Behavior**

Memory with Core Output Registers

When you select only the Register Port [A|B] Output of Memory Core, the embedded registers of the device are disabled for the selected ports in the generated core, as shown in Figure D-5.

Port A	or	Port B
☐ Register Port A Output of Memory Primitives		☐ Register Port B Output of Memory Primitives
☑ Register Port A Output of Memory Core		☑ Register Port B Output of Memory Core

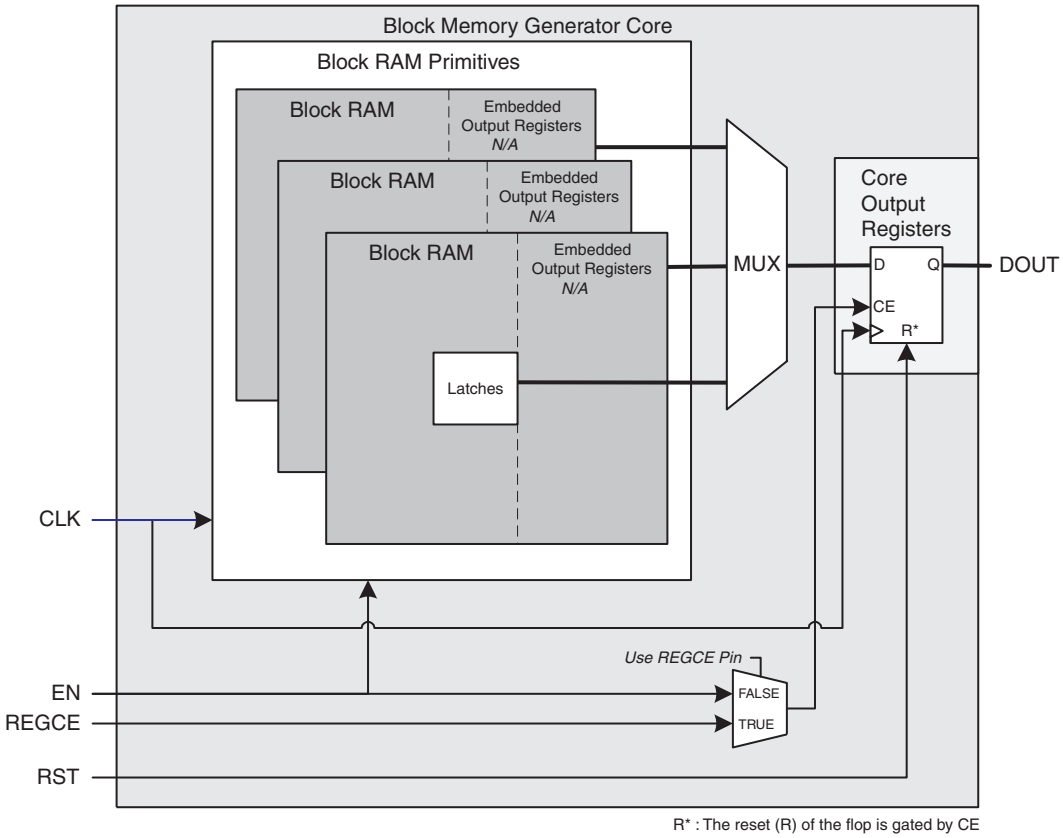

Figure D-5: **Block Memory Generated with Register Port [A|B] Output of Memory Core Enabled**

Memory with No Output Registers

If neither of the output registers is selected for ports A or B, output of the memory primitives is driven directly from the RAM primitive latches. In this configuration, as shown in Figure D-6, there are no additional clock cycles of latency, but the clock-to-out delay for a Read operation can impact design performance.

Port A	or	Port B
☐ Register Port A Output of Memory Primitives		☐ Register Port B Output of Memory Primitives
☐ Register Port A Output of Memory Core		☐ Register Port B Output of Memory Core

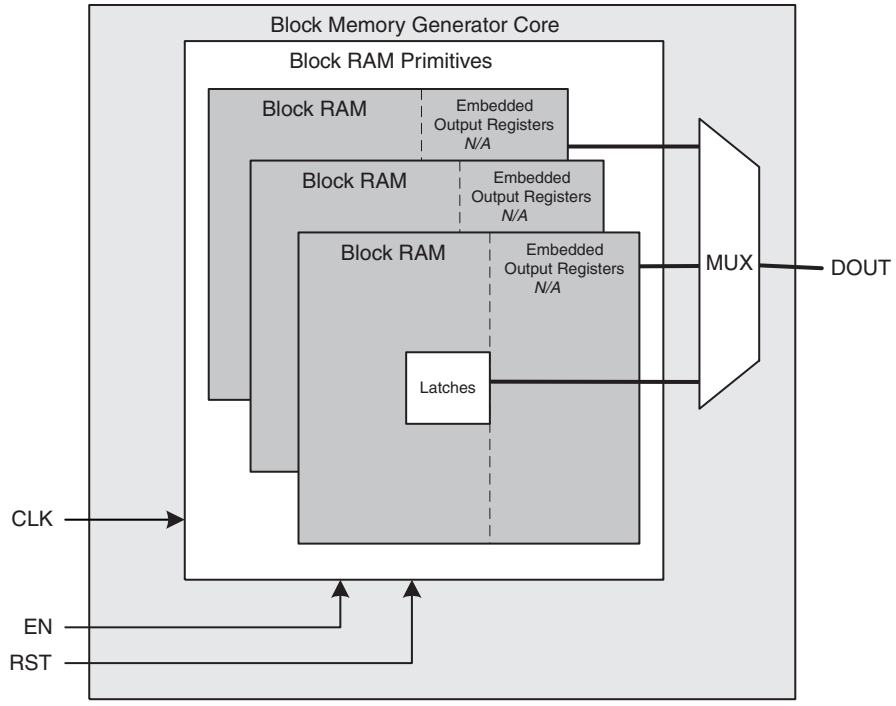

Figure D-6: **Block Memory Generated with No Output Registers Enabled**

Additional Resources and Legal Notices

Xilinx® Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

References

These documents provide supplemental material useful with this product guide:

1. *AMBA AXI4-Stream Protocol Specification*
2. *Xilinx Vivado AXI Reference Guide* (UG1037)
3. *Xilinx Data2MEM User Guide* (UG658)
4. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
5. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
6. *Vivado Design Suite User Guide: Implementation* (UG904)
7. *Vivado Design Suite User Guide: Designing with IP* (UG896)
8. *ISE to Vivado Design Suite Migration Guide* (UG911)
9. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
10. *Vivado Design Suite User Guide: Getting Started* (UG910)
11. *7 Series FPGAs memory Resources User Guide* (UG473)
12. UltraScale Architecture Memory resources (UG573)

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
04/01/2015	8.2	• Updated the BMG memory depth (words). • Added a note that the primitives will construct the requested memory only when you select an UltraScale device with Low_power algorithm and the configurations with no aspect ratio. • Added Read Address Change (RDADDRCHANGE[A|B]) option information in Chapter 4: Design Flow Steps under Port Options section. • Added Read Address Change Parameters in AX14 Interface SIM Parameters Table.
10/01/2014	8.2	• Added support for more cascaded primitives. • Added table detailing GUI Parameter to User Parameter relationships and default values. • Increased the supported depth to a maximum value of 256k. • Added support of all Write modes in Simple Dual-port RAM when ECC is not used
04/02/2014	8.2	Added the following features for UltraScale architecture-based devices: • Support for cascaded primitives. • Support for the pipeline register available in built-in ECC mode. • Support for dynamic power saving (sleep pin).
12/18/2013	8.1	• Added support for UltraScale™ architecture.
10/02/2013	8.0	• Document revision number advanced to 8.0 to align with core version number. • Maximum width of memory structures changed to 4608. • Added details about memory configuration options in Table 1-4. • Added information about Read Data and Read Enable Latency in Chapter 3. • Added Detailed Example Design and Test Bench chapters.
03/20/2013	3.0	Updated core to v8.0. Removed support for ISE Design Suite. Removed support for Virtex-6, Virtex-5 and Spartan-3 family devices.
12/18/2012	2.1	Updated ISE Design Suite to 14.4 and Vivado Design Suite to 2012.4. Added Virtex-7 device performance data in Single Primitive in Chapter 2.
10/16/2012	2.0	Updated core to v7.3, ISE Design Suite to 14.3 and Vivado Design Suite to 2012.3. Added the C_USE_BRAM_BLOCK and C_ENABLE_32BIT_ADDRESS parameters.
07/25/2012	1.0	Initial Xilinx release. Added support for Vivado Design Suite. This document replaces DS512, *Block Memory Generator Data Sheet*, and XAPP917, *Block Memory Generator Migration Guide*.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at http://www.xilinx.com/legal.htm#tos.