XAPP917 (v10.0) June 22, 2011

Block Memory Generator Migration Guide

Summary

This document provides step-by-step instructions for migrating designs containing instances of either the legacy Dual Port Block Memory and Single Port Block Memory LogiCORE™ IP cores or older versions of the LogiCORE IP Block Memory Generator core.

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

Migration Overview

Two methods for core migration are available: manual and automated. Manual migration steps include:

- Editing the XCO file(s) generated by the legacy cores or the XCO file(s) generated by older Block Memory Generator core versions to convert them to XCO file(s) compatible with the latest version of the Block Memory Generator core.
- Generating a new core netlist.
- Modifying the instantiations of the newly generated cores using the new templates.

Automated migration, provided in the Block Memory Generator Migration Kit, uses a Perl script to automate most of this migration process.The Migration Kit can be obtained at:

https://secure.xilinx.com/webreg/clickthrough.do?cid=92910&license=RefDesLicense

The Migration Kit automatically migrates your core to the latest version of the Block Memory Generator core it supports. A prerequisite for using the Migration Kit is that the latest version of the Block Memory Generator core supported by the kit that supports the device family you wish to target. Designs targeting unsupported families must be re-targeted to a supported device family. For information on device families supported by the latest version of the Block Memory Generator, see DS512, *Block Memory Generator Data Sheet.*

Before starting the migration process, follow these steps:

1. Evaluate the differences between the Block Memory Generator core and the legacy Dual Port Memory and Single Port Memory core features (Differences from the Legacy Cores, page 2).
2. Determine which migration process to use:
    - Automated Migration, page 5
    - Manual Migration, page 10

Differences from the Legacy Cores

The Block Memory Generator core leverages the capabilities of Xilinx® FPGA embedded block RAM primitives to automatically build a wide range of single and dual port memories by concatenating Xilinx FPGA block RAM primitives in an optimal manner. Use of an enhanced algorithm optimizes block RAM primitive resource utilization and creates a higher level of performance than previous generations of block memory cores.

Core Compatibility

Core Compatibility with Legacy Block Memory Cores

The Block Memory Generator core is not backward compatible with the Single and Dual Port Block Memory cores in these ways:

- Pin polarity, handshaking, and input register features supported by the previous generation memory cores are not supported in the Block Memory Generator.
- The behavior of the synchronous set/reset pin (RST) in the Block Memory Generator differs from the previous generation legacy v6.x cores when using optional output registers.
- The Single Port Block Memory legacy v6.x core and the Block Memory Generator do not have the same port names for a single-port memory configuration.
- The Dual Port Block Memory core and the Block Memory Generator do not have the same reset port name.
- The XCO files for the previous generation memory cores are *not* compatible with the Block Memory Generator.
- Not all combinations of Read and Write, Write Only, and Read Only port configurations are natively supported by Dual Port memories generated in the new Block Memory Generator. In these cases, generate a True Dual Port RAM and manually tie-off the unused port (automated if using the Migration Kit).

Differences in port names and XCO parameters are provided in "Convert the XCO File" on page 10 and "Modifying Instantiations of the Old Core" on page 17. For detailed information about the Block Memory Generator features, see the *Block Memory Generator Data Sheet* (DS512) available from the Block Memory Generator product page.

Core Compatibility with Older versions of Block Memory Generator Core

Versions of the Block Memory Generator core newer than 2.x are not backward compatible with the 1.x and 2.x versions of the Block Memory Generator cores in these ways:

- The names of the reset ports have been changed.
- The names of some of the reset related XCO parameters have changed.
- The XCO files of the previous versions of the Block Memory Generator core are not compatible with version 4.3 of the Block Memory Generator core.

Differences in port names and XCO parameters are provided in "Convert the XCO File" on page 10 and "Modifying Instantiations of the Old Core" on page 17. For detailed information about the Block Memory Generator features, see DS512, *Block Memory Generator Data Sheet*. For new features and resolved issues in the legacy v6.1 version of the core, see the release notes for the Block Memory Generator core.

Obsolete Features of Legacy Block Memory Cores

Pin Polarity Option

The Block Memory Generator does not support pin polarity options on the clock, enable, write enable, and set/reset input pins. When active low signaling is desired, users can invert the signals prior to the input of the core. Where possible, the inversion is automatically absorbed into the memory primitives when the design is processed by the ISE® Mapper.

Handshaking Pins

The Block Memory Generator does not support handshaking pins: ND[A|B], RFD[A|B], and RDY[A|B]. Figure 1 illustrates how to implement this logic in a design.

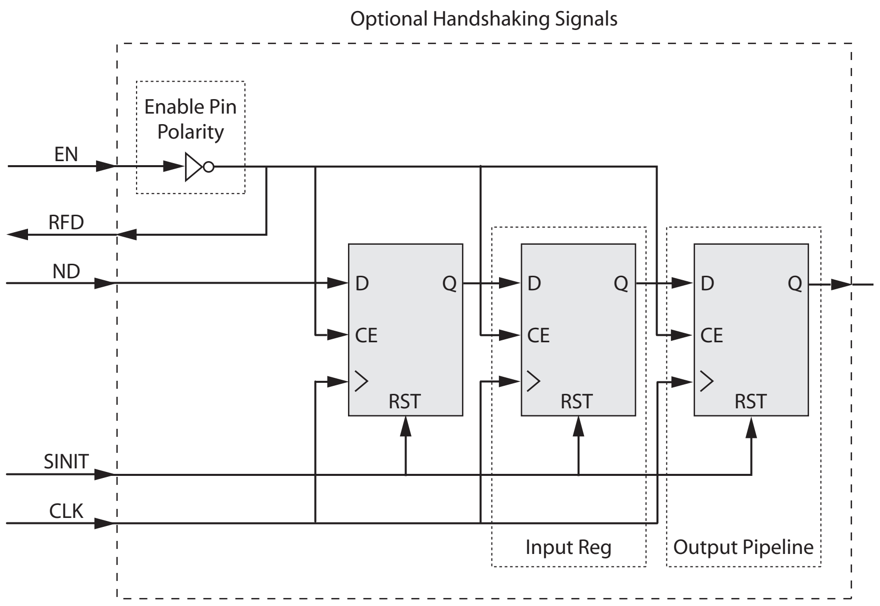

Figure 1: **Handshaking Pin Design**

Input Registers

The Block Memory Generator does not support input registers on ports DIN, ADDR, and WE prior to accessing the memory primitives. To implement this feature, add one stage of pipeline registers to the input of the Block Memory Generator core.

Port Configuration/Memory Type

Not all combinations of Read and Write, Write Only, and Read Only port configurations are natively supported by Dual Port memories generated in the new Block Memory Generator. Table 1 defines the relationship between legacy Dual Port Block Memory v6.x configurations and the equivalent implementation using the Block Memory Generator. The Migration Kit automatically maps legacy Dual Port Block Memory v6.x instantiations to the equivalent Block Memory Generator configuration, including tying-off any unused ports.

Table 1: **Block Memory Generator Port Configurations with Memory Types**

Dual Port Block Memory Core		Block Memory Generator Core	
Port A Configuration	**Port B Configuration**	**Memory Type**	**Notes**
Read and Write	Read and Write	True Dual Port RAM	N/A
Write Only	Read Only	Simple Dual Port RAM	N/A
Read Only	Read Only	Dual Port RAM	N/A
Read Only	Read and Write	True Dual Port RAM	Tie-off Write Port A
Write Only	Read and Write	True Dual Port RAM	Tie-off Read Port A
Read and Write	Read Only	True Dual Port RAM	Tie-off Write Port B
Read and Write	Write Only	True Dual Port RAM	Tie-off Read Port B
Read Only	Write Only	Simple Dual Port RAM	Reverse A and B Ports

Modified Behaviors of Legacy Block Memory Cores

Enable Pin

The Block Memory Generator provides two enable pin options:

- REGCE pin only controls the enables of the registers in the last output stage.
- EN pin controls the enables of all other registers and memory, and in the absence of a REGCE pin, also controls the enables of the registers in the last output stage.

The legacy v6.x memory cores only provide a core-wide enable (second option).

Synchronous Reset Pin

In the legacy v6.x Dual Port and Single Port Block Memory cores, the synchronous reset (SINIT pin) initializes all memory latches and output registers. With the Block Memory Generator core, the synchronous set/reset (RST pin) only resets the registers in the last output stage. When there are no output register stages present, it initializes the memory latches.

Output Registers

The legacy v6.x Dual Port and Single Port Block Memory cores allowed you to add an additional output pipeline register stage on the output of the block RAM. With the Block Memory Generator, you have better control on where and how these output registers are implemented. Output registers can be added to these locations:

- Output of the block RAM primitives (before output multiplexing)
- Output of the core (after output multiplexing)
- Pipeline register stages within the MUX

When targeting the Virtex®-4 FPGA architecture, the primitive output register stage is implemented using the embedded register in Virtex-4 FPGA block RAM when the RST pin is not used. For all other Virtex-4 FPGA configurations and their derivatives, all output register stages are implemented in the FPGA logic.

The Block Memory Generator 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.

Changes in WEA/WEB Signal Type (VHDL)

In the Block Memory Generator core, the width of the write enable input signals (WEA, WEB) can vary depending on the configuration. The width of the write enable buses are larger than 1 bit only if the byte write enable feature is used. To accommodate this, the write enable signal is always a bus of type std_logic_vector instead of std_logic in VHDL, even when the width of the signal is 1 bit.

When migrating from the Single Port Block Memory or Dual Port Block Memory cores to the Block Memory Generator, be aware that the WEA/WEB inputs in the Block Memory Single Port and Dual Port cores are of type std_logic, whereas the WEA/WEB inputs of the Block Memory Generator are of type std_logic_vector. When the write enable is only 1-bit wide, this change requires the use of a different method to connect the WEA/WEB inputs to the rest of the design.

To connect the WEA port:

1. Declare an intermediate signal of type std_logic_vector for the write enable. This is the signal that connects to the Block Memory Generator WEA port in the instantiation.

   Example:

   ```
   signal we_int: std_logic_vector(0 downto 0);
   ```
2. Assign your write enable input (of type std_logic) to bit 0 of the 1-bit-wide intermediate signal.

Example:

```
signal <user_wea>: std_logic;
we_int(0) <= <user_wea>;
```

3. Connect the intermediate signal you created and assigned during your instantiation of the Block Memory Generator to the WEA port of the core.

   Example:

```
user_bmg: bmg_inst PORT MAP (
     WEA => we_int;
     ...other port connections...
    );
```

Note: This change in WEA/WEB type does not affect customers using Verilog.

A full example instantiation of the Block Memory Generator core is provided here:

```
signal we_int: std_logic_vector(0 downto 0);
signal <user_wea>: std_logic;
...
we_int(0) <= <user_wea>;
...
--Instantiate the new BMG core
uut: blk_mem_gen_v3_1
port map (
  dina    => <user_dina>,
  wea     => we_int,
  addra   => <user_addra>,
  rsta    => <user_rsta>,
  douta   => <user_douta>,
  clka    => <user_clka>,
  ena     => <user_ena>
);
```

Using the Migration Kit

The Migration Kit provides two Perl scripts to help automate the process of converting existing v6.x legacy Block Memory cores or older versions of the Block Memory Generator core to the latest version of the Block Memory Generator core.

- The primary migration script automates all the manual steps, from converting the XCO file to modifying the instantiation of the old core. In addition, this script can be used to select only specific steps to automate, making it useful as part of the manual migration process.
- The second script recreates XCO files from EDIF or NGC netlists, if necessary (see Script Limitations for more information).

Primary Migration Script

If you can provide all the original design files, the primary migration script (`bmg_migrate.pl`) completely and seamlessly automates the migration process by executing these steps:

1. Converts old XCO files to new XCO files
2. Modifies instantiations of old cores to new cores including changing port names
3. Adds logic to implement obsolete features (input registers, signal polarity, enable, and reset behaviors)

4. Generates new netlists by calling the CORE Generator™ software Graphical User Interface (GUI) with the new XCO and related COE files
5. Restores original design files, if necessary

Recover XCO Script

If you do not have the legacy v6.x XCO files and/or the related COE files, the recover XCO script `(recover_xco.pl)` can recreate the legacy v6.x XCO files from your netlist(s).

This script performs these steps:

1. Scans the netlist(s) to determine the required values for the parameters in the XCO file.
2. Generates COE file(s) (initialization file) if initial values for the block RAMs have been specified in the netlist.

Note: The `recover_xco.pl` script currently only supports conversion of `.ngc` and EDIF netlists for the legacy block memory cores (Dual Port Block Memory and Single Port Block Memory).

Supported Platforms

- Windows® 2000
- Windows XP Pro
- Red Hat® Enterprise WS

About the Migration Script

The migration script, `bmg_migrate.pl`, can operate on various inputs and create a variety of outputs based on user-specified command line options.

When using the script as part of the standard, fully-automated flow, you supply the script with one or more of these three file types:

- Single Port and/or Dual Port Block Memory XCO core configuration files or XCO configuration files of the older versions of the Block Memory Generator core (created by the GUI when these cores were generated)
- HDL source file(s) containing the core instantiations (VHDL or Verilog)
- COE files used to initialize the memory (if necessary)

The path to the COE file is defined in the XCO file. If you do not have the COE file available, use the `recover_xco.pl` script to recover it (see Script Limitations, page 8). The `recover_xco.pl` script can also be used to regenerate the original legacy v6.x XCO file if it is not available.

From the script options, choose one or more of the following migration steps. All selected steps are automatically performed by the script.

- Convert XCO files of the Legacy Block Memory cores or the older versions of the Block Memory Generator core to Block Memory Generator XCO files of the latest version.
- Convert the instantiations of the old cores in your HDL source code to instances of the latest version of the Block Memory Generator core.
- Add logic to implement obsoleted features to make the new core backward compatible with the legacy instances.
- Generate the new netlists by running the CORE Generator software.

The script modifies and overwrites all input files so that the external project files and scripts do not need to be updated with new file names or locations. Although the script also automatically generates a backup of all files it modifies, it is strongly recommended that you create a backup of all project files before running the migration script.

Output Products

Depending on the chosen command line option, the script overwrites the input XCO files, modifies the input HDL files, and optionally generates Block Memory Generator netlists (in the same location as the XCO files).

The script creates a `./bmg_migrate_bak_<xcofilename>` backup directory for each XCO file in which a copy of all files modified by the migration process is placed. It also generates a restore script in this directory, `restore_files.pl`, to allow you to restore the original files if necessary.

Backward Compatibility Support for Legacy Block Memory Cores

The migration script adds code to your HDL source files to supplement features that have become obsolete in the Block Memory Generator core, making your design nearly "drop-in" compatible with the original design containing the legacy v6.x memory cores (see Script Limitations, page 8 for details).

The script can add these features to the Block Memory Generator core:

- Support for inverted clock and enable signals
- Support for input registers
- Support for handshaking signals

Using the Migration Script

To start the Migration Script, type the commands specific to your environment at the command prompt.

Note: The ISE 13.2 software requires a CGP file when CORE Generator is run in a command line. To help the user, the Migration Script comes with a sample CGP file (`coregen.cgp`). The user can modify the sample CGP file according to their own requirement. The modified CGP file should be kept in the same user directory where the XCO file and instantiation files are located, and the CGP file name must be `coregen.cgp`.

Linux

```
<path to script>/bmg_migrate.pl [-x [-n]]
[-m <HDL file(s)>] <xco file>
```

Windows

```
xilperl <path to script>\bmg_migrate.pl
[-x [-n]]-m <HDL file(s)>] <xco file>
```

You must use at least one of the following options in the command string:

- **-x** Creates an XCO output files needed by the CORE Generator software to generate the core. Requires an input of the old XCO file.
- **-n** Calls the CORE Generator software to generate Block Memory Generator netlists necessary to synthesize the design. This option must be used in tandem with the **-x** option.
- **-m** Modifies the HDL source files containing the core instantiations, converting the old instantiations and adding compatibility code where needed. Requires at least one old XCO file and one HDL design file containing old core instantiations.

`<HDL file(s)>` is a list of one or more HDL files containing instantiations of the cores described by the XCO files. These files can be referenced from directories other than the working directory. The HDL files listed with an XCO file must have instances with the same configuration as described by the XCO file.

`<xco file>` is a core configuration file of either the Legacy v6.x Block Memory cores or older versions of the Block Memory Generator core, which is to be converted to configuration file of the latest version of the Block Memory Generator core. This file can be referenced from directories other than the working directory.

To reverse the changes made by the script, go to the backup directory of the XCO file at `./bmg_migrate_bak_<xcofilename>` and then run xilperl script `restore_files.pl`.

Usage Examples

bmg_migrate.pl -x -m -n my_design.v my_old_core.xco

1. Creates a Block Memory Generator version of `my_old_core.xco`.
2. Modifies the instantiations of `my_old_core` in `my_design.v`.
3. Runs CORE Generator software to generate the Block Memory Generator version of `my_old_core.ngc` netlist file.

bmg_migrate.pl -x my_mem_core.xco

1. Creates a Block Memory Generator version of `my_mem_core.xco`.

   If an invalid Block Memory Generator version is given in the XCO, the script prompts the user to input a valid Block Memory Generator version.

Script Limitations

Block Memory Generator cores generated using the migration tool are backward compatible with the legacy v6.x Block Memory cores and the older versions of the Block Memory Generator.

The migration script can successfully read and parse most HDL files. Files written using standard coding styles and following standard Xilinx design methodologies (for example, using the CORE Generator software HDL instantiation templates typically generated with the cores) are handled appropriately. If the script cannot properly read and parse the user input HDL files, you must perform the conversion manually (see Migrating a Design Manually).

Recover XCO Script

The `recover_xco.pl` script scans netlists containing legacy v6.x Block Memory cores or older versions of the Block Memory Generator core and regenerates XCO and COE files of the respective core from them. It can be used to recover XCO or COE files from a design netlist if either of these types of files is missing. COE files are recovered *only* if there is memory initialization data within the netlist; this occurs by simulating the core using the Xilinx ISim simulator. After recovering the XCO and/or COE files, run the migration script to create the new Block Memory Generator counterparts.

Using the Recover XCO Script

Note: To run this script, ISim must be installed as part of ISE.

To start the Recover XCO script, type the commands specific to your environment at the command prompt. The Recover XCO script requires an input CORE Generator software project file (*.CGP).

Linux

```
<path to script>/recover_xco.pl -cgp=<coregen_project_file.cgp> <netlist(s)>
```

Windows

```
xilperl <path to script>\recover_xco.pl -cgp=<coregen_project_file.cgp> <netlist(s)>
```

For All Platforms

- `<netlist(s)>` is a list of netlist files of legacy v6.x Block Memory core or old Block Memory Generator core versions in EDN or NGC format.
- `<coregen_project_file.cgp>` is a software project file (*.CGP) containing the project options to be applied to the output XCO file. If a CGP file is not available, you can create one by creating a new project in GUI. When creating the new project, you are prompted to set project parameters such as target device, family, HDL language format (VHDL or Verilog), implementation output format, etc.

Note: You do not need to regenerate the legacy v6.x core after recreating the CORE Generator software project file.

The script automatically detects if the block RAMs were initialized and runs ISim simulation to extract initial values in the memory into a COE file.

Usage Examples

recover_xco.pl -cgp=coregen.cgp my_old_core.edn dp_mem.ngc

1. Scans the netlists and generates `my_old_core.xco` and `dp_mem.xco` files.
2. Can create COE files if initialization data is embedded in the input netlist file.

recover_xco.pl -cgp=../proj/my_coregen.cgp ../netlists/my_rom.edn

Scans the netlists and generates `../netlists/my_rom.xco and ../netlists/my_rom.coe.`

Using the Perl Scripts with a Windows OS

When running the Perl scripts in the Windows environment, the scripts must be invoked using a Perl program such as xilperl, which is distributed with Xilinx ISE software. The scripts are intended to be executed within a Windows command window (DOS shell).

Example Windows Usage

The location of HDL files and CORE Generator software netlists can differ from project to project. Assume that project HDL files are stored in `c:\xilinx\my_proj\hdl`, and Block Memory netlists and XCO files are in `c:\xilinx\my_proj\ip_cores`. Copy the migration scripts (`bmg_migrate.pl` and `recover_xco.pl`) into the `c:\xilinx\my_proj` directory.

Open a command window:

1. From the Windows Start menu, choose Run; then type `cmd` in the Run dialog box.
2. At the command prompt, enter:

   ```
   C:\> cd \xilinx\my_proj
   C:\xilinx\my_proj> xilperl bmg_migrate.pl -v6.2 -x -n -m hdl\top.v
   ip_cores\dp_ram.xco
   ```

The script performs the XCO conversion, modifies the HDL files containing the core instantiations, and generates new Block Memory Generator netlists. A new file, bmg_migrate.log is created, as well as a new subdirectory, `c:\xilinx\my_proj\bmg_migrate_bak` containing backups of all modified files.

Invoking the Restore_files Script

To invoke the `restore_files.pl` script, enter the following at the command prompt:

```
C:\> cd \xilinx\my_proj\bmg_migrate_bak
C:\xilinx\my_proj\bmg_migrate_bak> xilperl restore_files.pl
```

This effectively reverses the migration process by restoring any files modified by the `bmg_migrate.pl` script.

Migrating a Design Manually

This section provides instructions for manually migrating an existing design to a Block Memory Generator core. A summary of the required migration steps are provided, followed by specific migration instructions. Some of the manual conversion tasks can be automated using the Migration Kit script; see individual sections for more information.

To migrate a design manually:

1. Convert the XCO File.

   The XCO file is the file the CORE Generator software uses to determine a core's configuration. The format for the Block Memory Generator XCO file differs from the Single Port and legacy Dual Port Block Memory v6.x cores.

   Note: If you plan to generate a new Block Memory Generator core using the GUI, skip this step.
2. Generate the Block Memory Generator core.
3. Modify the instantiations of the old core.

Convert the XCO File

The first step in the manual migration process is to convert an XCO file of either a legacy v6.x Block Memory core or an older version of Block Memory Generator core to the XCO file of the latest Block Memory Generator core. If you do not have the original XCO or COE files for the old cores, you can use the `recover_xco` script provided with the automated Migration Kit to recover XCO files from the implementation netlists (see "Script Limitations" on page 8 for more information).

Core Parameter Differences

Tables 2, 3 and 4 define the parameter differences between the older cores (legacy Single and Dual Port Block Memory v6.x cores, or v1.1/v2.x versions of the Block Memory Generator cores) and the latest version of the Block Memory Generator core. In case of the legacy cores, parameters corresponding to features not supported in the Block Memory Generator core must be implemented in the user design by adding logic. For older versions of the Block Memory Generator core, almost all parameters are the same as the latest Block Memory Generator core. For parameter differences, see Table 4. See Differences from the Legacy Cores, page 2 for information about unsupported features.

In addition to changes in the parameter section of the XCO file, you must change the core name specified in the XCO. Update the core name and version from the old core to the new core.

For example, for the old Dual Port Block Memory core, change the line from

```
SELECT Dual_Port_Block_Memory Spartan3 Xilinx,_Inc. 6.1
```

to:

```
SELECT Block_Memory_Generator xilinx.com:ip:blk_mem_gen:6.2
```

For the old Single Port Block Memory core, change the line from

```
SELECT Single_Port_Block_Memory Spartan3 Xilinx,_Inc. 6.1
```

to:

```
SELECT Block_Memory_Generator xilinx.com:ip:blk_mem_gen:6.2
```

For an old version of the Block Memory Generator core, change the line from

```
SELECT Block_Memory_Generator Spartan3 Xilinx,_Inc. 2.8
```

to:

```
SELECT Block_Memory_Generator xilinx.com:ip:blk_mem_gen:6.2
```

After these changes are made, use the CORE Generator GUI to import the modified XCO file by selecting **Project** > **Import Existing Customized IP**. Ensure that all the parameters set in the GUI are appropriate for your design and then generate the core.

If you do not have an old XCO file to convert, the migration guide also provides a script (`recover_xco.pl`) that can recover XCO parameters by scanning the netlist of the old core (see "Script Limitations" on page 8).

Table 2: **XCO Parameter Mapping: Dual Port Block Memory Cores**

Dual Port XCO Parameter	BMG XCO Parameter	Description of Conversion
component_name	component_name	No change required
width_[alb]	write_width_[alb] read_width_[alb]	read_width[alb]= write_width[alb] = width[alb]
depth_a	write_depth_a	Direct replacement
depth_b	-	Not needed
configuration_port_[a&b]	memory_type	• A write, B read**:** Simple_Dual_Port • A read, B write: Not supported, switch A and B ports • A read, B read**:** Dual_Port_ROM • Otherwise: True_Dual_Port_RAM
write_mode_port_[alb]	operating_mode_[alb]	• Read_After_Write: WRITE_FIRST • Read_Before_Write: READ_FIRST • No_Read_On_Write: NO_CHANGE
global_init_value	remaining_memory_locations	Direct replacement
load_init_file	load_init_file	Direct replacement
coefficient_file	coe_file	Direct replacement
port_[alb]_enable_pin	enable_[alb]	• False: Always_Enabled • True: Use_EN[AlB]_Pin
port_[alb]_handshaking_pins	-	Not supported
port_[alb]_register_inputs	-	Not supported
port_[alb]_additional_output_ pipe_stages	• BMG v2.4 and earlier: register_output_of_memory_core • BMG v2.5 and later: register_port[alb]_output_of_me mory_core	Direct replacement. 0: false 1: true
port_[alb]_init_pin	use_rst[alb]_pin	Parameter value is the same, behavior is different.
port_[alb]_init_value	output_reset_value_[alb]	Direct replacement

Table 2: **XCO Parameter Mapping: Dual Port Block Memory Cores *(Cont'd)***

Dual Port XCO Parameter	BMG XCO Parameter	Description of Conversion
primitive_selection	algorithm	• Optimize_For_Area: Minimum_Area • Select_Primitive: Fixed_Primitives
select_primitive	primitive	Direct replacement
port_[alb]_write_enable_polarity	-	Not supported
port_[alb]_enable_pin_polarity	-	Not supported
port_[alb]_initialization_pin_polarit y	-	Not supported
port_[alb]_active_clock_edge	-	Not supported
disable_warning_messages	disable_out_of_range_warnings	Direct Replacement
disable_warning_messages	disable_collision_warnings	Direct Replacement

Table 3: **XCO Parameter Mapping: Single Port Block Memory Cores**

Single Port XCO Parameter	BMG XCO Parameter	Description of Conversion
component_name	component_name	No change required
width	write_width_[a&b] read_width_[a&b]	read_width[alb] = write_width[alb] = width
depth	write_depth_a	Direct replacement
port_configuration	memory_type	• Read_And_Write: Single_Port_RAM • Read_Only: Single_Port_ROM
write_mode	operating_mode_[a&b]	• Read_After_Write: WRITE_FIRST • Read_Before_Write: READ_FIRST • No_Read_On_Write: NO_CHANGE
global_init_value	remaining_memory_locations	Direct replacement
load_init_file	load_init_file	Direct replacement
coefficient_file	coe_file	Direct replacement
enable_pin	enable_[a&b]	False: Always_Enabled True: Use_EN[AlB]_Pin
handshaking_pins	-	Not supported
register_inputs	-	Not supported
additional_output_pipe_ stages	• BMG v2.4 and earlier: register_output_of_memory_ core • BMG v2.5 and later: register_port[a&b]_output_of_ memory_core	Direct replacement 0: false 1: true
init_pin	use_rst[a&b]_pin	Parameter value is the same, behavior is different.

Table 3: **XCO Parameter Mapping: Single Port Block Memory Cores *(Cont'd)***

Single Port XCO Parameter	BMG XCO Parameter	Description of Conversion
init_value	output_reset_value_[a&b]	Direct replacement
has_limit_data_pitch	-	Not supported
limit_data_pitch	-	Not supported
primitive_selection	Algorithm	• Optimize_For_Area: Minimum_Area • Select_Primitive: Fixed_Primitives
select_primitive	Primitive	Direct replacement
enable_pin_polarity	-	Not supported
initialization_pin_polarity	-	Not supported
write_enable_polarity	-	Not supported
active_clock_edge	-	Not supported
disable_warning_messages	disable_out_of_range_warnin gs	Direct Replacement
disable_warning_messages	disable_collision_warnings	Direct Replacement

Table 4: **XCO Parameter Differences Versions of Block Memory Generator Cores**

BMG v2.x or v1.1 XCO Parameter	Block Memory Generator v6.2	Description of Conversion
use_ssr[alb]_pin	use_rst[alb]_pin	Direct Replacement
use_ramb16bwer_reset_behavior	reset_memory_latch_a reset_memory_latch_b	Direct Replacement
N/A	ecctype	New Parameter
N/A	softecc	New Parameter
N/A	register_porta_input_of_softecc	New Parameter
N/A	register_portb_output_of_softecc	New Parameter
N/A	interface_type	Determines the type of interface selected
N/A	axi_type	Determines the type of the AXI4 interface
N/A	axi_slave_type	Determines the type of the AXI4 Slave interface
N/A	use_axi_id	Determines whether the AXI4 interface supports AXI4 ID
N/A	axi_id_width	Defines the AXI4 ID value

Generate the Block Memory Core

To migrate from the legacy Single Port or Dual Port Block Memory v6.x cores or older versions of the Block Memory Generator core to the new Block Memory Generator, a Block Memory Generator netlist must be generated to replace the old netlist.

Parameterizing the Block Memory Generator GUI

One method for generating an appropriate Block Memory Generator netlist is to parameterize the Block Memory Generator GUI without using an XCO file. Set the desired parameterization options on the Block Memory Generator customization GUI and generate the core.

Table 5 compares the GUI parameters between the legacy v6.x Block Memory cores and the Block Memory Generator cores. Table 6 compares GUI parameters between the old and new versions of the Block Memory Generator core. These tables help outline the appropriate options to select when creating a new core using the Block Memory Generator GUI.

Table 5: **GUI Parameter Comparison for Legacy Block Memory Cores**

Single-Port/Dual-Port Block Memory	Block Memory Generator	Functionality of the GUI Parameter
Single Port: Read/Write, Read only **Dual Port:** Read/Write, Write Only, Read Only for each port	Single Port RAM Single Port ROM Simple Dual Port RAM Dual Port ROM True Dual Port RAM	Type of memory to be generated. See Table 1 for equivalents.
-	Write Enable	When selected, the core has a write enable pin; when deselected, write enable is tied to '1' internally.
-	Byte Write Enable (for Virtex-6, Virtex-5, Virtex-4, Spartan®-6 and Spartan-3A/AN/DSP families only)	When targeting Virtex-6, Virtex-5, Virtex-4, Spartan-6, or Spartan-3A/AN/DSP, select whether to use the byte-write enable feature. Select whether to use a byte size of 8 or 9.
Primitive Selection • Optimize For Area • Select Primitive	Algorithm • Minimum Area • Fixed Primitives	Algorithm used to implement the memory. • Minimum Area Algorithm**:** Generates a core using the least number of primitives. • Selectable 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 box.
Width / Port [A&B] width	Port A write width, Port A read width / Port [A&B] write width, Port [A&B] read width	Specify the port's write width and depth. Select the port's read width from the drop-down menu of valid choices.
Depth / Port [A&B] depth	Port A write depth / Port [A&B] write depth	The minimum depth for both cores is 2. The read depth is calculated automatically.
Write mode / Port [A&B] write mode • Read After Write • Read Before Write • No Read On Write	Port A Operating mode / Port [A&B] Operating mode • Write First • Read First • No Change	The Block Memory Generator core provides the same operating modes found on the block memory primitives.

Table 5: **GUI Parameter Comparison for Legacy Block Memory Cores *(Cont'd)***

Single-Port/Dual-Port Block Memory	Block Memory Generator	Functionality of the GUI Parameter
Design Options / Port [A&B] Design Options • Enable Pin	Enable (Port A) / Enable (Port A and Port B) Select the enable type: • Use EN[A|B] pin • Always enabled (no EN[A|B] pin available)	Defines whether an enable pin appears on the core.
Output Register Options • SINIT pin • INIT value	Output Reset (Port A) / Output Reset (Port A and B) • Use RSTA/RST[A&B] pin • Output Reset value	Specify the reset value of the memory output latch and output registers. These values are with respect to the read port widths. Choose whether a set/reset pin is needed.
Output Register Options • Additional output pipeline stages	Optional Output Registers • Register Port A Output of Memory Primitives/ Register Port [A|B] of Memory Primitives • Register Port A Output of Memory Core/ Register Port [A|B] of Memory Core Output Register Enables • Use REGCEA pin/Use REGCE[A|B] pins • Use REGCEB pin/Use REGCE[A|B] pins	Output Registers can optionally be generated at the output of the memory primitives or on the output of the memory core, or at both locations. The output register stages can be chosen separately for port A and port B. When targeting Virtex-4 FPGAs, the embedded output registers in the block RAM primitives are used if the user chooses to register the output of the memory primitives and if RST pin is not used. For other Virtex-4 FPGA architecture and its derivatives, the registers in the FPGA slices are used.
Initial content • Load Init File • Global Init value	Memory Initialization • Load Init File • Fill remaining memory locations	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.

Table 5: **GUI Parameter Comparison for Legacy Block Memory Cores *(Cont'd)***

Single-Port/Dual-Port Block Memory	Block Memory Generator	Functionality of the GUI Parameter
-	Structural/UNISIM Simulation Model Options	Select the type of warning messages and outputs generated by the structural simulation model for collisions. Select whether the behavioral model should assume synchronous clocks for collision warnings.
-	Pipeline stages within the MUX	Pipeline register stages can optionally be chosen within the MUX for certain configurations; this option is common for both ports. For detailed information about this feature, see the *Block Memory Generator Data Sheet* (DS512).

Table 6: **GUI Parameter Comparison for Older Versions of Block Memory Generator**

BMG XCO Parameter v3.x or v2.x or v1.1	Block Memory Generator v6.2	Functionality of the GUI Parameter
Use SSR[A|B] Pin	Use RST[A|B] Pin	Direct Replacement
Use Reset Behavior From RAMB16WER Primitive	Reset Memory Latch	Direct Replacement
N/A	ECC Type	New Parameter
N/A	Soft ECC	New Parameter
N/A	Register Port A Input of SoftECC logic	New Parameter
N/A	Register Port B Output of SoftECC logic	New Parameter
N/A	INTERFACE TYPE	New Parameter
N/A	AXI TYPE	New Parameter
N/A	AXI SLAVE TYPE	New Parameter
N/A	AXI ID	New Parameter
N/A	AXI ID WIDTH	New Parameter

Executing an Updated XCO File

After an XCO file for the new core is created (either using the migration kit script or manually), you can invoke CORE Generator software to generate a new netlist. First, create a new project and configure it with the appropriate options (including target family, output products, and so forth). After the project is created, generate a core by choosing **Project** > **Import Existing Customized IP**.

Modifying Instantiations of the Old Core

The final migration step involves updating all instantiations of the old cores in your HDL source code to reference the new core. Additionally, Differences from the Legacy Cores, page 2 discusses whether you need to add additional logic to restore any features that have become obsolete.

To update each old memory core instantiation:

1. Change the name of the module (only necessary if component name of the core has changed).
2. Change the port names used.

   See Table 7 and Table 8 for port name conversions.
3. Any code modifications or workarounds previously used in your code to compensate for specific behaviors in the legacy Single Port or Dual Port v6.x Block Memory cores should be removed before migrating to the Block Memory Generator to prevent unpredictable behavior when using the new core.
4. Add any additional logic to make the core backward compatible.

   Additional logic is necessary to implement input registers, handshaking signals, pin polarity, and to implement legacy v6.x enable and reset behavior with optional output registers only (see Differences from the Legacy Cores, page 2 for additional information).

Table 7: **Port Name Mapping: Dual Port Block Memory Cores**

Old Dual-Port Block Memory Core		New Block Memory Generator Core		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
CLKA	Available	CLKA	All configurations	Same	Available
ADDRA	Available	ADDRA	All configurations	Same	Available
DINA	Optional	DINA	All RAM configurations	Same	Optional
DOUTA	Optional	DOUTA	All configurations except simple dual-port RAM	Same	Optional
ENA	Optional	ENA	Optional in all configurations	Behavior different with optional output registers	Optional
WEA	Optional	WEA	All configurations	Changed from std_logic to std_logic_vector	Optional
SINITA	Optional	RSTA	Optional in all configurations	Same	Optional
-	-	REGCEA	Optional in all configurations with output registers	-	-
NDA	Optional	-	-	Need additional logic	Optional
RFDA	Optional	-	-	Need additional logic	Optional
RDYA	Optional	-	-	Need additional logic	Optional
CLKB	Available	CLKB	Available in dual-port configurations	Same	Available

Table 7: **Port Name Mapping: Dual Port Block Memory Cores *(Cont'd)***

Old Dual-Port Block Memory Core		New Block Memory Generator Core		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
ADDRB	Available	ADDRB	Available in dual-port configurations	Same	Available
DINB	Optional	DINB	Available in true dual-port RAM configurations	Same	Optional
DOUTB	Optional	DOUTB	Available in dual-port configurations	Same	Optional
ENB	Optional	ENB	Optional in dual-port configurations	Behavior different with optional output registers	Optional
WEB	Optional	WEB	Available in dual-port RAM configurations	Changed from std_logic to std_logic_vector	Optional
SINITB	Optional	RSTB	Optional in dual-port configurations	Same	Optional
-	-	REGCEB	Optional in dual-port configurations with output registers	-	-
NDB	Optional		-	Need additional logic	Optional
RFDB	Optional		-	Need additional logic	Optional
RDYB	Optional	-	-	Need additional logic	Optional

Table 8: **Port Name Mapping: Single Port Block Memory Cores**

Old Single-Port Block Memory Core		New Block Memory Generator Core		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
CLK	Available	CLKA	Available in all configurations	Same	Available
ADDR	Available	ADDRA	Available in all configurations	Same	Available
DIN	Optional	DINA	Available in all RAM configurations	Same	Optional
DOUT	Optional	DOUTA	Available in all configurations except simple dual-port RAM	Same	Optional
EN	Optional	ENA	Optional in all configurations	Behavior different with optional output registers	Optional
WE	Optional	WEA	Available in all configurations	Changed from std_logic to std_logic_vector	Optional

Table 8: **Port Name Mapping: Single Port Block Memory Cores *(Cont'd)***

Old Single-Port Block Memory Core		New Block Memory Generator Core		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
SINIT	Optional	RSTA	Optional in all configurations	Same	Optional
-	-	REGCEA	Optional in all configurations with output registers	-	-
ND	Optional	-	-	Need additional logic	Optional
RFD	Optional	-	-	Need additional logic	Optional
RDY	Optional	-	-	Need additional logic	Optional

Table 9: **Port Name Mapping: Older Versions of Block Memory Generator Core**

Block Memory Generator Core v2.x		Block Memory Generator Core v6.2		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
CLKA	Available	CLKA	All Configurations	Same	Available
ADDRA	Available	ADDRA	All Configurations	Same	Available
DINA	Optional	DINA	All RAM Configurations	Same	Optional
DOUTA	Optional	DOUTA	All Configurations except simple dual port RAM	Same	Optional
ENA	Optional	ENA	Optional in all configurations	Same	Optional
WEA	Optional	WEA	All Configurations	Same	Optional
SSRA	Optional	RSTA	Optional in all configurations	Only name change	Optional
REGCEA	Optional	REGCEA	Optional in all configurations with output registers	Same	Optional
CLKB	Available	CLKB	Available in dual-port configurations	Same	Available
ADDRB	Available	ADDRB	Available in dual-port configurations	Same	Available
DINB	Optional	DINB	Available in true dual-port RAM configurations	Same	Optional
DOUTB	Optional	DOUTB	Available in dual-port configurations	Same	Optional
ENB	Optional	ENB	Optional in dual-port configurations	Same	Optional
WEB	Optional	WEB	Available in dual-port RAM configurations	Same	Optional
SSRB	Optional	RSTB	Optional in dual-port configurations	Only name change	Optional
REGCEB	Optional	REGCEB	Optional in dual-port configurations with output registers	Same	Optional
SBITERR	Available	SBITERR	Available in ECC/Soft-ECC configurations	Same	Available

Table 9: **Port Name Mapping: Older Versions of Block Memory Generator Core *(Cont'd)***

Block Memory Generator Core v2.x		Block Memory Generator Core v6.2		Description of Conversion	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
DBITERR	Available	DBITERR	Available in ECC/Soft-ECC configurations	Same	Available
-		RDADDRECC	Available in Virtex-6 ECC/Soft-ECC and Spartan-6 Soft-ECC configurations	-	Available
-		INJECTSBITERR	Available in Virtex-6 ECC/Soft-ECC and Spartan-6 Soft-ECC configurations	-	Optional
		INJECTDBITERR	Available in Virtex-6 ECC/Soft-ECC and Spartan-6 Soft-ECC configurations	-	Optional

Known Issues

Known issues with the Migration Kit are documented in AR 24848.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
1/30/06	1.1	Initial Xilinx release.
7/13/06	2.1	Updated to ISE v8.2i, doc/core version to 2.1
9/28/06	2.2	Updated core version to v2.2, added Port Configurations and Memory Types. Added information about new byte-wide write/enable feature.
11/15/06	2.3	Updated core version to v2.3, release date.
4/2/07	2.4	Revised document and migration scripts to be version independent, able to migrate to any valid version of the new Block Memory Generator core.
8/15/07	2.5	Added information about Block Memory Generator v2.5 new feature, independent output register control for each port.
10/17/07	3.0	Updated to include optional pipeline stages within the MUX feature for Block Memory Generator v2.6.
3/24/08	3.1	Updated to Block Memory Generator v2.7.
9/19/08	3.5	Updated to Block Memory Generator v2.8.
4/24/09	4.0	Updated to Block Memory Generator v3.1.
6/24/09	4.5	Updated to Block Memory Generator v3.2.
9/16/09	5.0	Updated to Block Memory Generator v3.3.
4/19/10	6.0	Updated to Block Memory Generator v4.1.
7/23/10	7.0	Updated to Block Memory Generator v4.2.
9/21/10	8.0	Updated to Block Memory Generator v4.3.
3/1/11	9.0	Updated to Block Memory Generator v6.1.
6/22/11	10.0	Updated to Block Memory Generator v6.2.

Notice of Disclaimer