

LogiCORE IP Distributed Memory Generator v7.2

Product Guide

PG036 July 25, 2012

Table of Contents

SECTION I: SUMMARY

IP Facts

Chapter 1: Overview

Feature Summary	7
Applications	8
Licensing and Ordering Information	8

Chapter 2: Product Specification

Performance	9
Resource Utilization	10
Port Descriptions	10

Chapter 3: Designing with the Core

General Design Guidelines	12
Specifying Memory Contents Using a COE File	20
MIF File Description	20
Clocking	21
Resets	21

SECTION II: VIVADO DESIGN SUITE

Chapter 4: Customizing and Generating the Core

GUI	23
Parameter Values in the XCO File	29
Output Generation	30

Chapter 5: Constraining the Core

Chapter 6: Detailed Example Design

Directory and File Contents	32
Example Design	35
Demonstration Test Bench	36
Simulation	36
Messages and Warnings	38

SECTION III: ISE DESIGN SUITE

Chapter 7: Customizing and Generating the Core

GUI	40
Parameter Values in the XCO File	46
Output Generation	47

Chapter 8: Constraining the Core

Chapter 9: Detailed Example Design

Directory and File Contents	49
Example Design	52
Demonstration Test Bench	53
Implementation	54
Simulation	54
Messages and Warnings	55

SECTION IV: APPENDICES

Appendix A: Verification, Compliance, and Interoperability

Simulation	57
------------------	----

Appendix B: Migrating

Parameter Changes in the XCO File	58
Auto-Upgrade Feature	58

Appendix C: Additional Resources

Xilinx Resources	59
Technical Support	59

Revision History	60
Notice of Disclaimer	60

SECTION I: SUMMARY

IP Facts

Overview

Product Specification

Designing with the Core

Introduction

The Xilinx LogiCORE™ IP Distributed Memory Generator core uses Xilinx Synthesis Technology (XST) to create a variety of distributed memories.

Features

- Generates read-only memories (ROMs), single, simple dual, and dual-port random access memories (RAMs), and SRL16-based memories
- Supports data depths ranging from 16–65,536 words
- Supports data widths ranging from 1–1024 bits
- Optional registered inputs and outputs
- Optional pipelining when output is registered

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family ⁽¹⁾	Zynq™-7000 ⁽²⁾ , Artix™-7, Virtex®-7, Kintex™-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP
Supported User Interfaces	Native
Provided with Core	
Design Files	ISE: Netlist Vivado: Encrypted RTL
Example Design	VHDL
Test Bench	VHDL
Constraints File	Not Provided
Simulation Model	Verilog Behavioral, VHDL Behavioral Verilog Structural, VHDL Structural
Supported S/W Driver	N/A
Tested Design Flows⁽³⁾	
Design Entry	Vivado Design Suite v2012.2 ISE Design Suite 14.2
Simulation	Mentor Graphics ModelSim Xilinx ISIM/XSIM
Synthesis	XST Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see the [release notes](#) for this core.
2. Supported in ISE Design Suite implementations only.
3. For the supported versions of the tools, see the [Xilinx Design Tools: Release Notes Guide](#).

Overview

The Distributed Memory Generator core is used to create memory structures using LUT RAM resources. The core can create the following memory types:

- [Distributed ROM, page 12](#)
- [Distributed Single-Port RAM, page 14](#)
- [Distributed Dual-Port RAM, page 15](#)
- [Distributed SRL16-Based RAM, page 17](#) (excluding Zynq™-7000, Artix™-7, Virtex®-7, Kintex™-7, Virtex-6, Virtex-5 and Spartan®-6 devices)
- [Distributed Simple Dual-Port RAM, page 18](#)

For detailed information about each memory type, see the respective memory type in [Chapter 3, Designing with the Core](#).

Options are available for simple registering of inputs and outputs. Optional asynchronous and synchronous resets are available for the output registers. For timing information, see the Xilinx Product Specification for the specific target architecture.

Feature Summary

Depth. In all supported FPGA families, the depth can range from 16–65536 words in multiples of 16.

Width. The width of each word can be anywhere in the range of 1–1024 bits.

Optional Input Registering. The following inputs to the memory can be registered or non-registered. When input registering is used, these inputs can be clock-enabled.

- Write Address
- Data
- Write Enable
- Output Register Clock Enable
- Dual-port RAM Read Address

- Simple Dual-port RAM Read Address

Optional Output Registering and Pipelining. The memory can be non-registered, registered, or both. The output registers can have a variety of controls, including

- Asynchronous Reset
- Synchronous Reset
- Clock Enable

In addition, the Clock Enable and Synchronous Reset can be configured so the Synchronous Reset overrides the Clock Enable, and vice versa. In single-port, simple dual-port and dual-port distributed RAM cores with registered outputs, an additional pipeline register may be added to the output path to improve the operating speed at the expense of an additional cycle of latency.

Applications

Applications for Distributed Memory are diverse and numerous. Examples include

- Using the distributed ROM as a very large look-up table
- Using the single-port RAM as scratch pad memory for the embedded PowerPC™ microprocessors used within the Zynq-7000, 7 series, Virtex-6, Virtex-5, and Virtex-4 families, or for the MicroBlaze™ or PicoBlaze™ processors
- Using the simple dual and dual-port RAM within an asynchronous FIFO
- Using the SRL16-based RAM for pattern generation

A large number of Xilinx IP cores rely on the Distributed Memory Generator internally to implement memory structures.

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite tools under the terms of the [Xilinx End User License](#). Information about this and other Xilinx LogiCORE IP modules is available at the [Xilinx Intellectual Property](#) page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your [local Xilinx sales representative](#).

Product Specification

The Distributed Memory Generator core constructs the memory out of LUT RAM. The core offers a variety of memory structures like ROM, Single Port RAM, Dual Port RAM and Simple Dual Port RAM. Figure <dmg_top.png> shows the top level block diagram of the core.

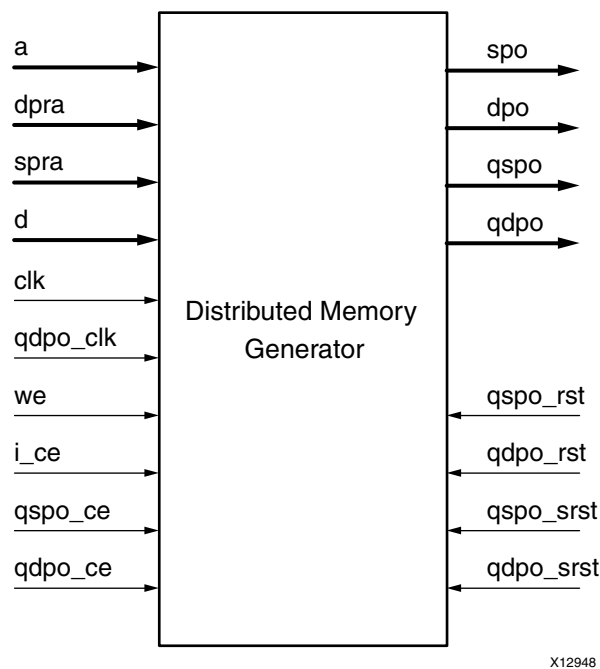


Figure 2-1: Top-Level Block Diagram

Performance

This section details the performance information for various core configurations.

Latency

Depending on the configuration, the Distributed Memory Generator takes either zero or one clock latency to present the read data on the output port. For an asynchronous read, the latency is zero, and for a synchronous read the latency is one.

Resource Utilization

When designing distributed memories, the LUTs in the slices are used to implement memory primitives or to construct multiplexers. The registers available in these slices are used for input and output registering.

When input registering is requested, extra registers are used. One per control bit (WE, QSPO_CE, and QDPO_CE) and one per bit of data and address (D[P:0], A[N:0], and SPRA[N:0]/DPRA[N:0]).

Port Descriptions

Table 2-1 defines the Distributed Memory core signals.

Table 2-1: Core Signal Pinout

Name	Direction	Description
D[P:0]	Input	Data input to be written into the memory for single-port, simple dual-port, dual-port and SRL16-based RAMs.
A[N:0]	Input	Address inputs. Only address input for ROMs and single-port RAMs. On SRL16-based RAMs, it defines the most significant bits (4 and up) of the memory locations written to. On dual-port memories, it defines memory location written to and memory location read out on the SPO[P:0] outputs. On simple dual-port memories, it defines memory location written to.
SPRA[N:0]	Input	Single-Port Read Address. Present only on SRL16-based RAMs and defines memory location read out on the SPO[P:0] outputs.
DPRA[N:0]	Input	Dual/Simple Dual-Port Read Address. Present only on dual-port and simple dual-port RAMs and defines memory location read out on the DPO[P:0] outputs.
SPO[P:0]	Output	Non-registered single-port output bus. Non-registered data output bus for ROMs, single-port, and SRL16-based RAMs. One of two non-registered output buses on dual-port RAMs.
QSPO[P:0]	Output	Registered single-port output bus. Registered data output bus for ROMs, single-port, and SRL16-based RAMs. One of two registered output buses on dual-port RAMs.
DPO[P:0]	Output	Non-registered dual/simple dual-port output bus. One of the non-registered data output buses for dual-port and simple dual-port RAMs. Data stored at the address location specified by DPRA[N:0] appears at this port.
QDPO[P:0]	Output	Registered dual/simple dual-port output bus. One of two registered output buses on dual-port and simple dual-port RAMs.

Table 2-1: Core Signal Pinout (Cont'd)

Name	Direction	Description
CLK	Input	Write clock and register clock for ROMs, single-port, and SRL16-based RAMs. On dual-port RAMs signal is the write clock and register clock for single-port input and output registers.
QDPO_CLK	Input	On dual-port and simple dual-port RAMs, signal is the write clock and register clock for dual-port and simple dual-port RAM input and output registers
WE	Input	Write Enable
I_CE	Input	Input Clock Enable. Signal is present for RAMs which have registered inputs. The clock enable controls input data register, address register and WE register.
QSPO_CE	Input	On ROMs, clock enable controls all input and output registers. On dual-port memories, controls output registers in QSPO path.
QDPO_CE	Input	Present only on dual-port and simple dual-port RAMs. Controls output registers in QDPO path.
QSPO_RST	Input	Single-port registered output asynchronous reset.
QDPO_RST	Input	Available only on dual-port and simple dual-port RAMs. Dual/Simple Dual-port registered output asynchronous reset.
QSPO_SRST	Input	Single-port registered output synchronous reset
QDPO_SRST	Input	Available only on dual-port and simple dual-port RAMs. Dual-port registered output synchronous reset

Note: All control inputs are Active High. If an Active Low input is required for a specific control pin, an inverter must be placed in the path to the pin; the inverter is absorbed appropriately during mapping.

Designing with the Core

This chapter includes guidelines and additional information to make designing with the core easier.

General Design Guidelines

The following sections provide a functional description and illustration of each of the four memory types.

Distributed ROM

The Distributed Memory Generator uses LUT-based distributed ROM resources to create 16-bit deep, 1-bit wide ROMs, and generates a fabric-based bus multiplexer to create a deeper and wider ROM. The content of this memory is defined by supplying an input coefficient (COE) file to the CORE Generator™ software when the memory is generated, after which the content is fixed.

To create the distributed ROM, the core parses the initialization data provided by the user-supplied COE file. From that data, any necessary logic or optional registering is inferred and created. [Figure 3-1](#) shows the distributed ROM schematic symbol, and [Figure 3-2](#) illustrates one of the possible implementations of a distributed ROM core. For distributed ROM timing information, see the user guide for the specific FPGA family-related architecture.

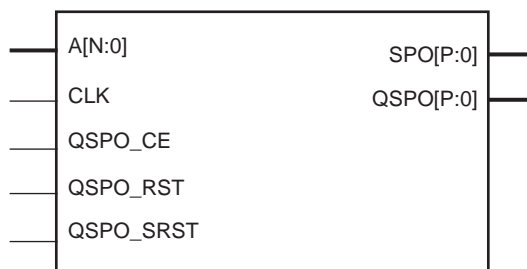


Figure 3-1: Distributed ROM Schematic Symbol

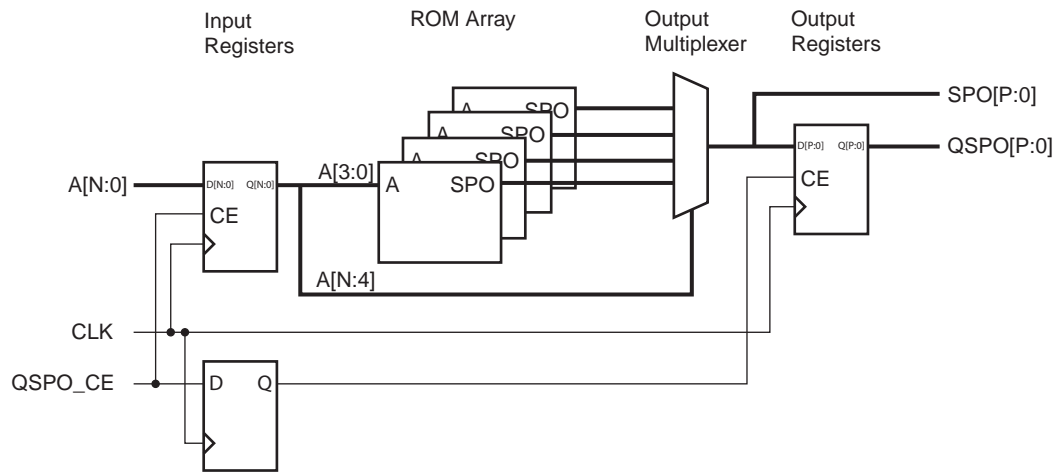


Figure 3-2: Distributed ROM Module Schematic

Distributed Single-Port RAM

The distributed single-port RAM uses the single-port distributed RAM resource of the LUT. Writes to the Single-Port RAM are synchronous to the clock (CLK). However, read operations can be asynchronous (SPO) or synchronous to the clock (QSPO). Figure 3-3 illustrates the distributed single-port RAM schematic symbol, and Figure 3-4 illustrates its internal implementation. If a pipeline register is added to a registered core (not illustrated), the additional register is re-timed into the SPO MUX array. For timing information about the distributed single-port RAM, see the appropriate user guide for the target FPGA architecture.

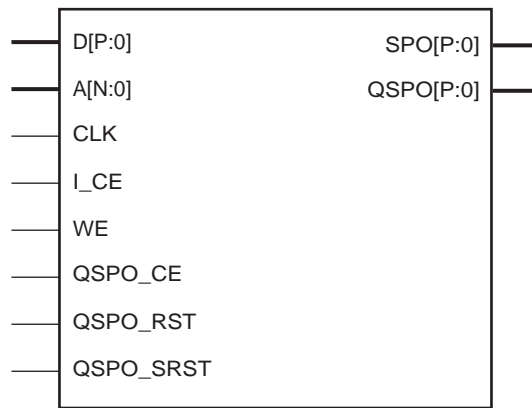


Figure 3-3: Distributed Single-Port RAM Schematic Symbol

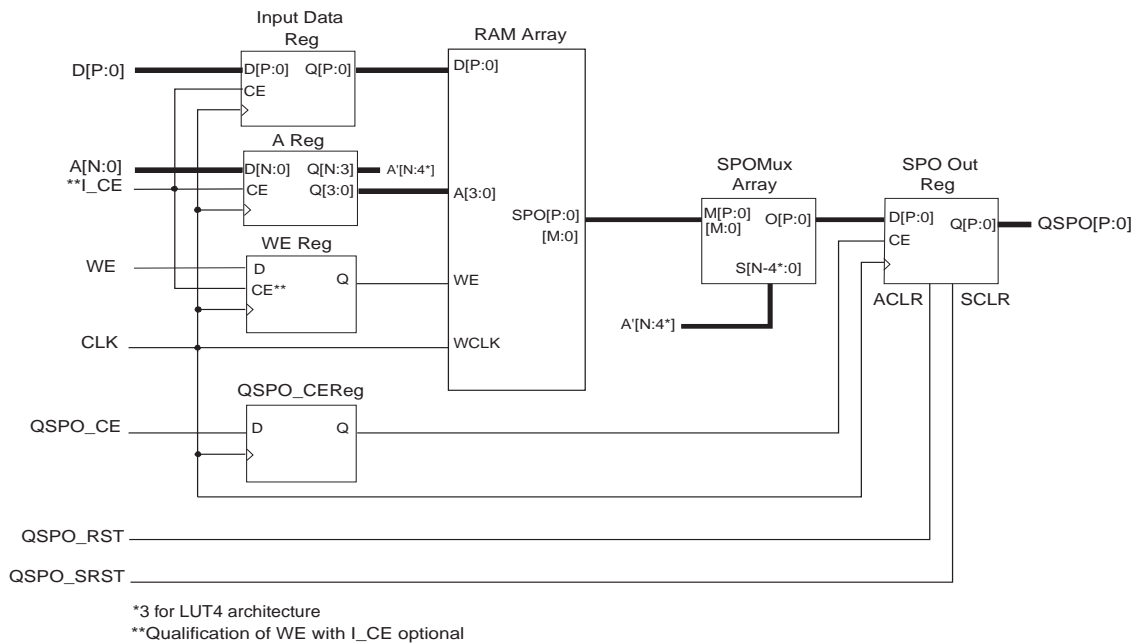


Figure 3-4: Distributed Single-Port RAM Module Schematic

Distributed Dual-Port RAM

Writes to the distributed dual-port RAM are synchronous to the clock (`CLK`). However, read operations from the distributed dual-port RAM can be asynchronous or synchronous with respect to either of the two clocks (`CLK` or `QDPO_CLK`). [Figure 3-5](#) illustrates the dual-port RAM schematic symbol, showing the relevant ports. [Figure 3-6](#) shows the internal implementation of the distributed dual-port RAM. If a pipeline register is added to a registered core (not illustrated), the additional register is re-timed into the SPO MUX array and DPO MUX array.

In implementing the most simple dual-port RAM, two LUTs sharing a common write logic are used. When RAM is written to, both LUTs continue to share common content but have different address buses for reading. In this way, contents from two different memory locations can be addressed and read from the SPO and DPO outputs. Different clock domains can also be used to clock the data read out of the SPO and DPO pins if the outputs are also registered.

The `QDPO_CE` port controls DPRA only when Dual Port Output CE is selected under output options.

The width of the data buses must be identical on input and output. If width conversion is required from a memory instance, consider using the Block Memory Generator core.

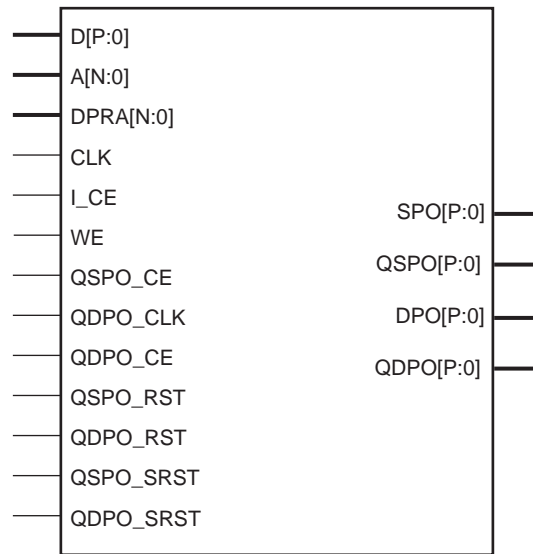
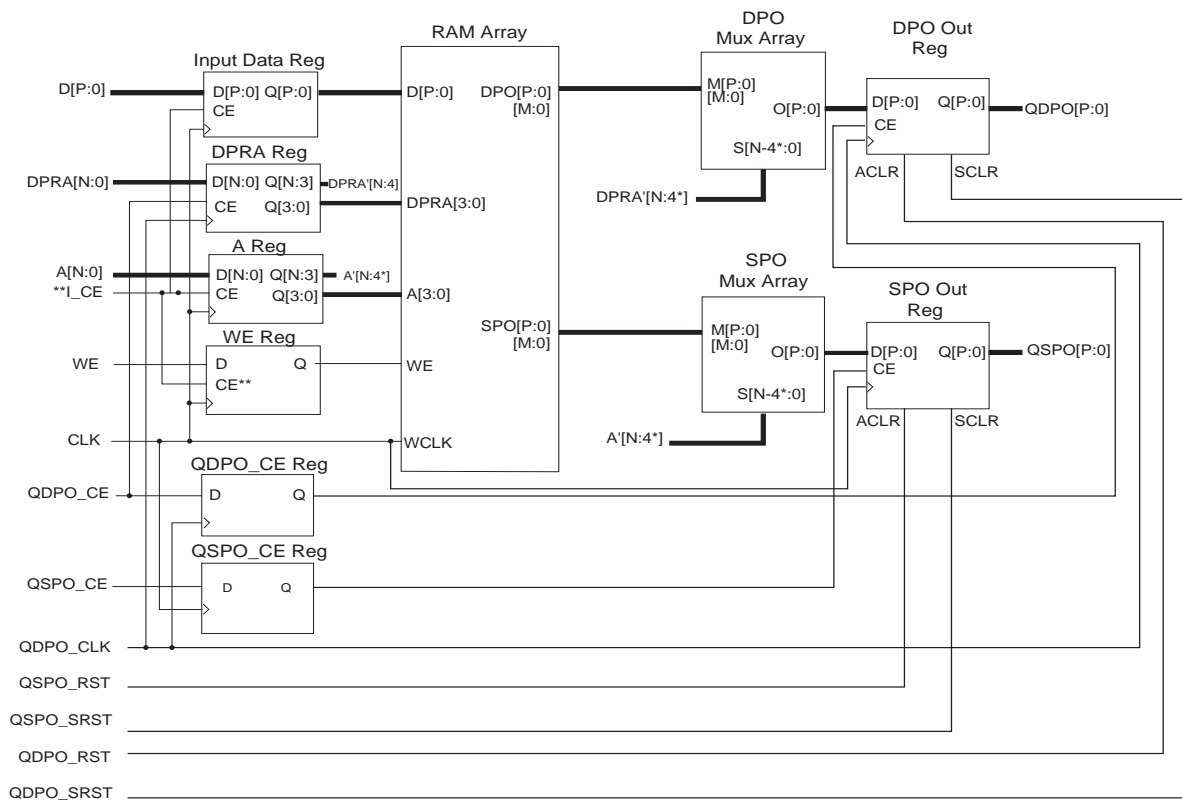


Figure 3-5: Dual-Port RAM Schematic Symbol



* 3 for LUT4 architecture
 ** Qualification of WE with I_CE optional

Figure 3-6: Dual-Port RAM Module Schematic

Distributed SRL16-Based RAM

Note: The Virtex-5 and above architectures do not support SRL16-based memories.

Figure 3-7 shows the schematic symbol of the distributed SRL16-based RAM. Note that Address Bus A, used to write data to a specific address in the SRL16 array, does not contain the full range of bits normally required for a memory of a given depth. Address Bus A is missing the four *least* significant bits because the four address bits on each SRL16 are driven by the four least significant bits from the read address bus SPRA. (The address pins on the SRL16 determine where in the SRL16 the data is read *from*, not where data is written *to*.)

Where written data is placed in the address space is determined by the specific SRL16 written to, and where previous data has gone, because data is sequentially written into the SRL16. This means that the four least-significant bits in address bus A are not required.

Because 16-bit deep shift registers are being used in place of the LUT4 primitives, the SRL16-based RAM is not a genuine RAM. Specifically, random access is given to addresses the client of the memory wants to read *from* to access the SRL16 holding the data of interest. To write data to a specific location in memory, the application using the memory has to enable a specific SRL16 or set of SRL16s (when the width of the RAM is greater than 1 bit), and then assert the Write Enable for the RAM. The Write Enable actually drives the Clock Enable on the SRL16, allowing data to be shifted into the SRL16s.

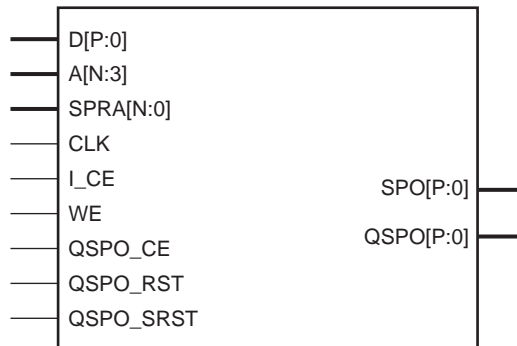


Figure 3-7: SRL16-Based RAM Schematic Symbol

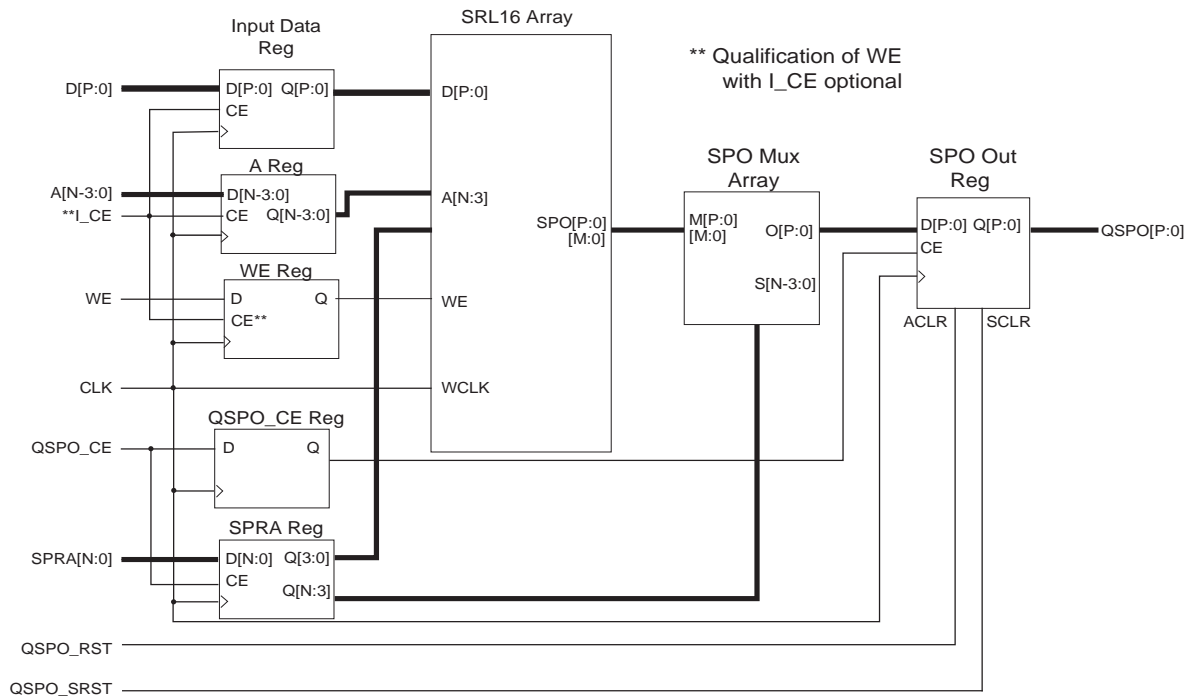


Figure 3-8: SRL16-Based RAM Module Schematic

Distributed Simple Dual-Port RAM

Note: Simple Dual-Port RAM is supported only for Zynq-7000, 7 series, Virtex-6, Virtex-5, and Spartan-6 device families.

Writes to the distributed simple dual-port RAM are synchronous to the clock (CLK). However, read operations from the distributed simple dual-port RAM can be asynchronous or synchronous with respect to either of the two clocks (CLK or QDPO_CLK). [Figure 3-9](#) illustrates the simple dual-port RAM schematic symbol, showing the relevant ports. [Figure 3-10](#) shows the internal implementation of the distributed simple dual-port RAM. If a pipeline register is added to a registered core (not illustrated), the additional register is re-timed into the DPO MUX array.

The QDPO_CE port controls DPRA only when Simple Dual Port Output CE is selected under output options.

The width of the data buses must be identical on input and output. If width conversion is required from a memory instance, consider using the Block Memory Generator core.

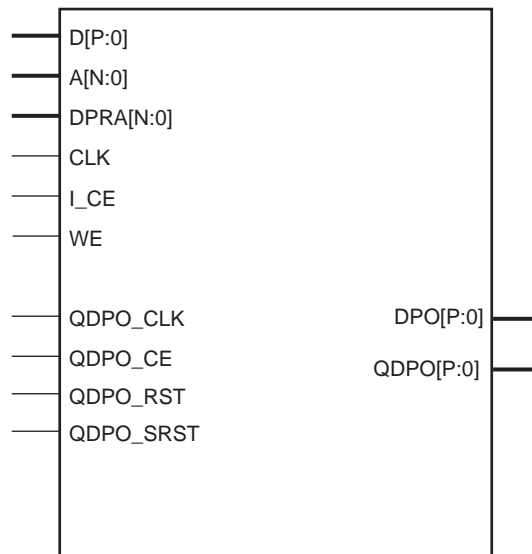
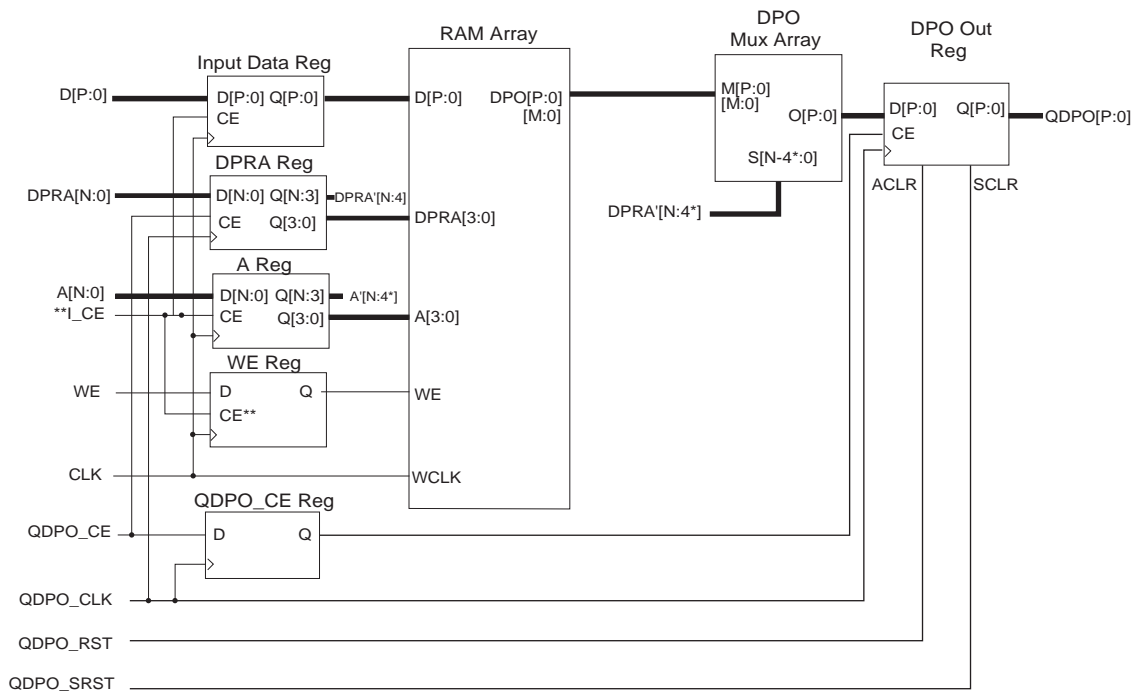


Figure 3-9: Simple Dual-Port RAM Schematic Symbol



* 3 for LUT4 architecture
 ** Qualification of WE with I_CE optional

Figure 3-10: Simple Dual-Port RAM Module Schematic

Specifying Memory Contents Using a COE File

The initial contents of the memory can be defined using the COE file, which consists of two parameters: `memory_initialization_radix` and `memory_initialization_vector`. Each line is terminated by a semicolon.

- **memory_initialization_radix.** The radix of the initialization value is specified here, with the choices being 2, 10, or 16.
- **memory_initialization_vector.** Each row of memory elements are defined with a binary, decimal, or hexadecimal number having an equivalent binary value that represents whether an individual memory element along the width of the row is set to '1' or '0.' Each row of memory initialization is separated by a comma or white space, up to the depth of the memory. Negative values are not allowed.

An example COE file:

```
;Sample Initialization file for a 16x32 distributed ROM
memory_initialization_radix = 16;
memory_initialization_vector =
23f4 0721 11ff ABe1 0001 1 0A 0
23f4 0721 11ff ABe1 0001 1 0A 0
23f4 721 11ff ABe1 0001 1 A 0
23f4 721 11ff ABe1 0001 1 A 0;
```

MIF File Description

The COE file provides a high-level method for specifying initial memory contents. During core generation, the COE file is converted into a MIF file, which holds the actual binary data used to initialize the memory in the core and simulation models. The MIF file consists of one line of text per memory location. The first line in the file corresponds to address 0, and the second line corresponds to address 1, and so forth. The text on each line must be the initialization value (MSB first) for the corresponding memory address in binary format, with exactly one binary digit per bit of memory width.

Note: For HDL simulations, the MIF file must reside in the simulation directory.

Clocking

The Distributed Memory Generator core has two clocks: `clk` (write/read clock) and `qdp0_clk` (read clock). Depending on the configuration, the core can have either zero, one, or two clocks.

For an asynchronous ROM configuration, there is no clock. For a RAM configuration, the core has a write clock (`clk`). The same write clock (`clk`) can be used as a read clock or a separate read clock (`qdp0_clk`) can also be used.

Resets

The Distributed Memory Generator core has four resets: two synchronous resets and two asynchronous resets. Depending on configuration, the core can have any of the following reset options:

- Synchronous reset only
- Asynchronous reset only
- Synchronous and asynchronous resets
- No resets

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

GUI

The Distributed Memory Generator is listed in two places in the Vivado Project Manager's IP Catalog graphical user interface (GUI) View.

To access the Distributed Memory Generator, do one of the following:

- Choose **Basic Elements > Memory Elements**
- Choose **Memories & Storage Elements > RAMs & ROMs**

The Distributed Memory Generator GUI uses three screens.

- Main Screen ([Figure 4-1](#))
- Input, Dual-Port, and Output Options Screen ([Figure 4-2](#))
- Initial Content and Reset Options Screen ([Figure 4-3](#))

All the screens share common tabs and buttons to provide information about the core and perform specific actions, such as generating the core and navigating among screens.

Main Screen

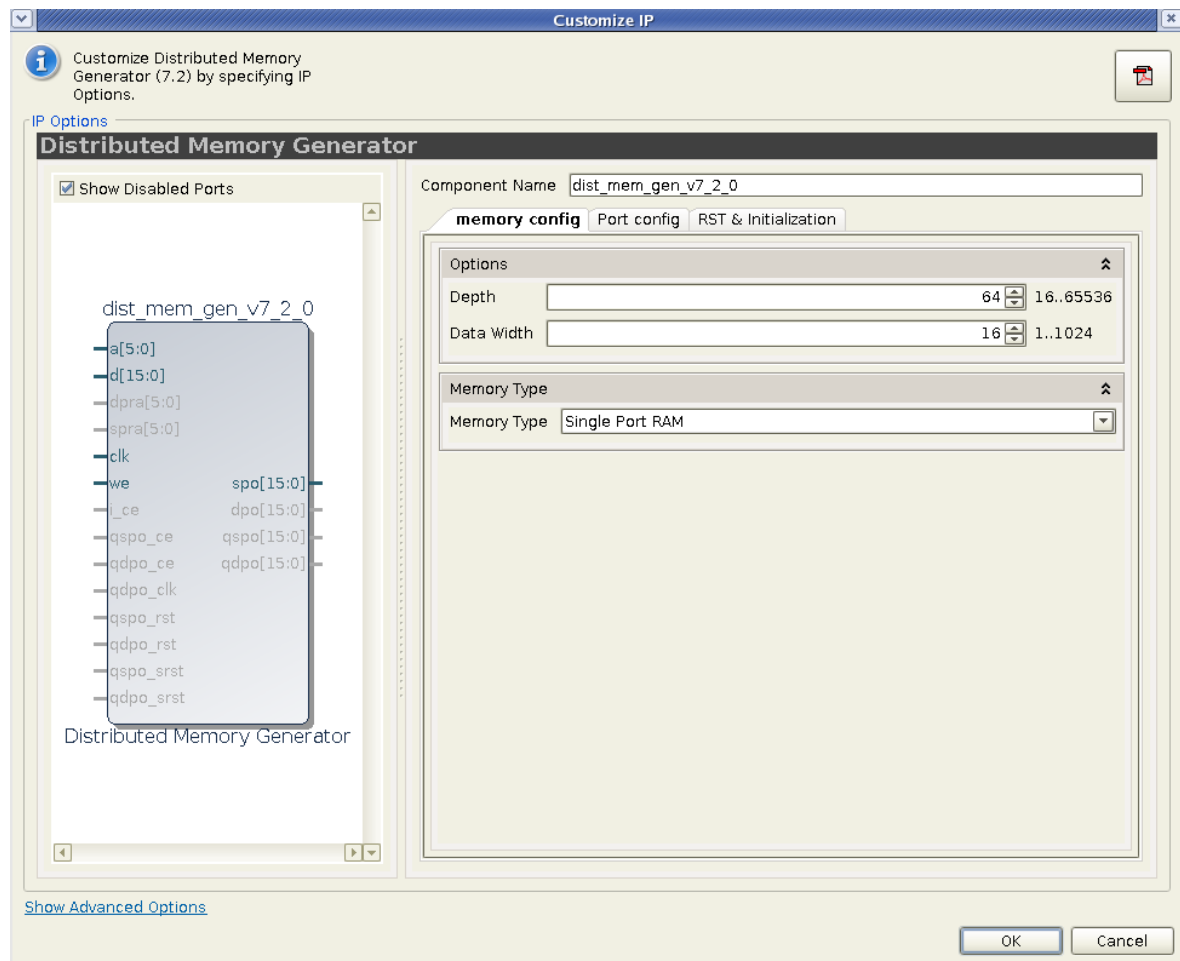


Figure 4-1: Main Screen

Component Name. The base name of the output files generated for the core. Names must begin with a letter and be composed of any of the following characters: a to z, 0 to 9 and “_”

Depth. Enter a value in the valid range from 16–65536, in steps of 16. Default value is 64.

Data Width: Enter the width of the memory in the valid range from 1–1024. The default value is 16.

Memory Type: Select one of five options. The default setting is single-port RAM.

- **ROM.** Figure 3-2 is a schematic of the structure of the ROM modules. The address register is optional (controlled by the setting of the Input Options parameter). Output registering is also optional (controlled by the setting of the Output Options parameter). Note that the clock is not required when these registers are not present. The resets and clock enables are optional.

- **Single-Port RAM.** Figure 3-4 is a schematic of the structure of the single-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). Output registering is also optional (controlled by the setting of the Output Options parameter). The resets and clock enables are optional. The clock CLK is always required because writes to the single-port RAM are synchronous to that clock.
- **Dual-Port RAM.** Figure 3-6 is a schematic of the structure of the dual-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). The dual-port read address register is optional (controlled by the setting of the Dual-Port Address parameter). Output registers for both output ports are also optional (controlled by the setting of the Output Options parameter). When registered outputs are selected, the two output ports can be clocked by the same or different clock signals and can have the same or different clock enables (based on the settings selected for the Common Output Clock and Common Output CE parameters). All resets and clock enables are optional. Note that the clock CLK is always required, because writes to the RAM are synchronous to that clock.
- **Simple Dual-Port RAM.** Figure 3-10 is a schematic of the structure of the simple dual-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). The simple dual-port read address register is optional (controlled by the setting of the Simple Dual-Port Address parameter). Output registers for both output ports are also optional (controlled by the setting of the Output Options parameter).

When registered outputs are selected, the output port can be clocked by the same or different clock signals and can have the same or different clock enables (based on the settings selected for the Common Output Clock and Common Output CE parameters). All resets and clock enables are optional. Note that the clock CLK is always required because writes to the RAM are synchronous to that clock.

Port Configuration Screen

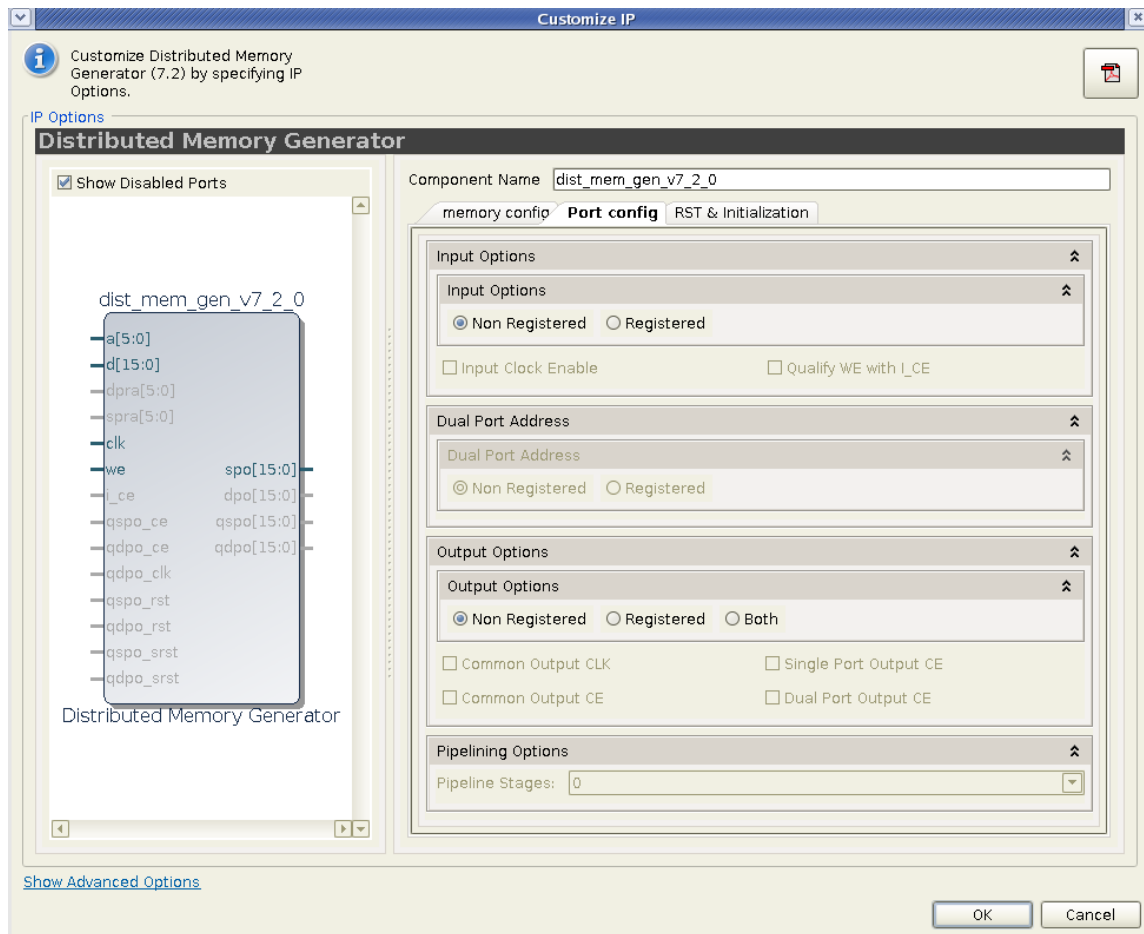


Figure 4-2: Distributed Memory Options Screen

Input Options. Select an option for the required input types. Non-registered is the default setting. Selecting Registered produces different effects depending on the selected memory type. For ROM, an address register is generated; for single-port RAM, simple dual-port, dual-port, and SRL16-based RAM, a register on the A[N:0] address input, a data input register, and a WE register are generated.

- **Input Clock Enable.** An optional input available when Input Options are set to Registered and Memory Type is *not* a ROM.
- **Qualify WE with I_CE.** Valid only for single-port RAM, simple dual-port RAM, dual-port RAM, and SRL16-based RAM with Input Options set to Registered and Input Clock Enable selected. When deselected, the WE register has no clock-enable control. When selected, the WE register has a clock enable driven by the I_CE input.

Dual/Simple Dual-Port Address. Valid only for simple dual-port RAM and dual-port RAMs, and controls the presence or absence of a register on the DPRA[N:0] inputs. Non-registered is the default setting.

Pipelining Options: When registered single-port RAMs, simple dual-port RAMs and dual-port RAMs are selected, an optional pipeline register can be placed into the output path.

- **Pipeline Stages.** Select '0' for no pipeline registers and '1' for a single pipeline stage in the output multiplexer.

Output Options: Select an option for the required output types. Non-registered is the default setting.

- **Single-Port Output Clock Enable.** Enabled for registered output memory or for input registered ROM to provide this optional pin.
- **Dual/Simple Dual-Port Output Clock Enable.** Enabled only for output registered simple dual-port RAM and dual-port RAMs to provide this optional pin.
- **Common Output CLK.** Enabled only for registered simple dual-port RAMs and dual-port RAMs. If not selected, the SPO registers are clocked by the CLK input and the DPO registers are clocked from the `QDPO_CLK` input. Default setting is selected, where all output registers are clocked from the CLK input.
- **Common Output CE.** Enabled only for non-registered simple dual-port RAMs and dual-port RAMs and only if Common Output Clock is also selected. If Common Output CE is deselected, the SPO register clocks are enabled by the `QSPO_CE` input and the DPO register clocks are enabled from the `QDPO_CE` input. Default setting is selected, where all output register clocks are enabled by the `QSPO_CE` input.

Reset and Initialization Screen

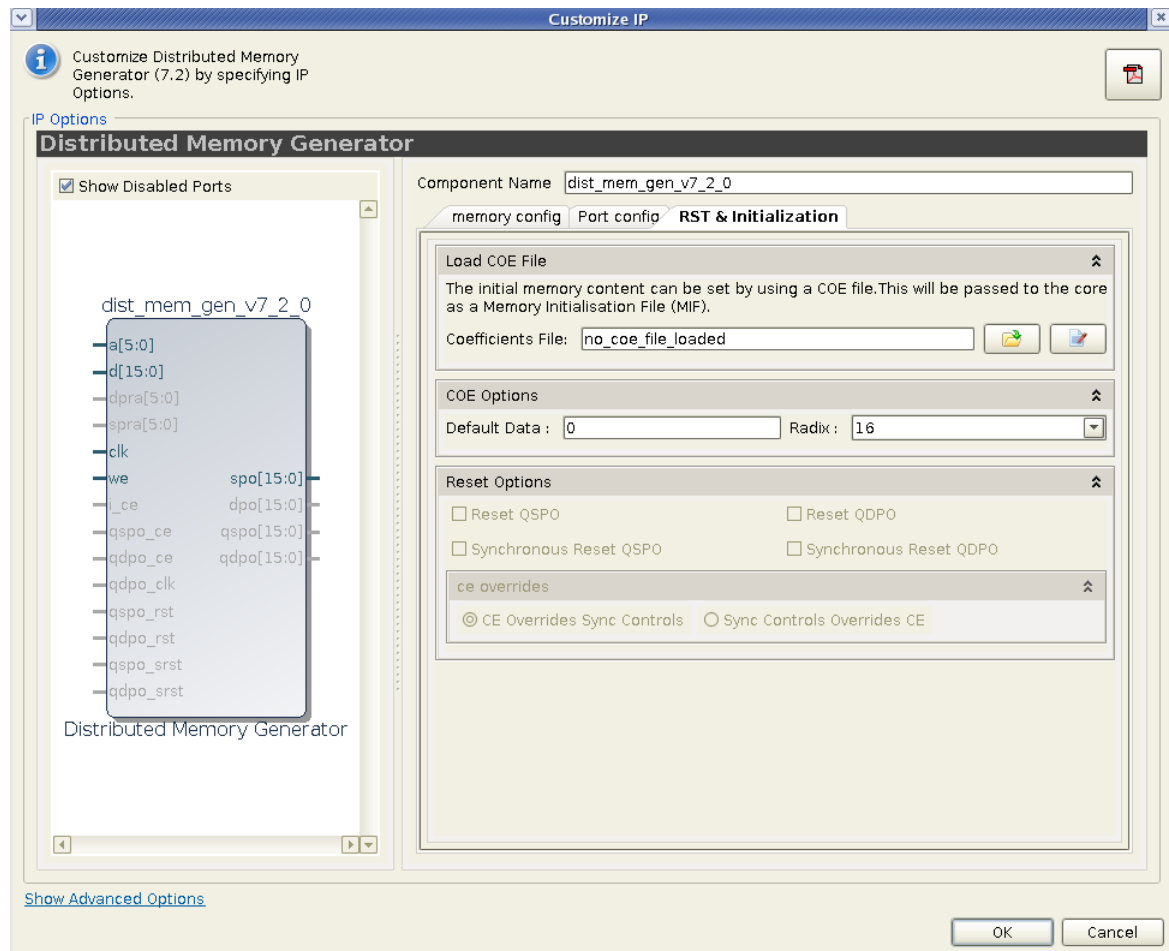


Figure 4-3: Initial Content and Reset Options Screen

Load COE File

The initial values of the memory elements can be set using a COE file.

- To load the COE file, click **Browse**.
- To view the initial contents, click **Show**.

For a description of the COE file, see [Specifying Memory Contents Using a COE File](#).

COE Options

Enter the initial value to be stored in memory locations not otherwise initialized in the COE file.

- **Default Data:** When no value is entered, the field defaults to 0. Values can be entered in binary, decimal, or hexadecimal formats, as defined by the Default Data Radix entry.

- **Radix:** Choose the radix of the Default Data value. Valid entries are 2, 10, and 16.

Reset Options

- **Reset QSPO:** Enabled only when the core has a registered single-port output. If selected, an asynchronous single-port output reset pin is available.
- **Reset QDPO:** Enabled only when the core has a registered simple dual-port and dual-port output. If selected, an asynchronous simple dual-port and dual-port output reset pin is available.
- **Synchronous Reset QSPO:** Enabled only when the core has a registered single-port output. If selected, a synchronous single-port output reset pin is available.
- **Synchronous Reset QDPO:** Enabled only when the core has a registered simple dual-port and dual-port output. If selected, a synchronous simple dual-port and dual-port output reset pin is available.
- **CE Overrides Sync Controls:** Enabled only when one of the synchronous reset options has been selected and an output clock enable has been selected. When selected, the synchronous control signals are qualified by the clock enable pin.
- **Sync Controls Overrides CE:** Enabled only when one of the synchronous reset options has been selected and an output clock enable has been selected. When selected, the synchronous control signals operate regardless of the state of the output clock enable signals.

Parameter Values in the XCO File

Parameter values are described in [Table 4-1](#).

Table 4-1: Parameter Values

XCO Parameter Name	XCO Values	Default Values
ce_overrides	ce_overrides_sync_controls, sync_controls_overrides_ce	ce_overrides_sync_controls
coefficient_file	no_coe_file_loaded, coe file name	no_coe_file_loaded
common_output_ce	true, false	false
common_output_clk	true, false	false
component_name	dist_mem_gen_v7_2_0	dist_mem_gen_v7_2_0
data_width	1 - 1024	16
default_data	Any binary, decimal and hexadecimal values equal to the data width	0
default_data_radix	2, 10, 16	16
depth	16 – 65536 (multiples of 16)	64

Table 4-1: Parameter Values (Cont'd)

XCO Parameter Name	XCO Values	Default Values
dual_port_address	non_registered, registered, both	non_registered
dual_port_output_clock_enable	true, false	false
input_clock_enable	true, false	false
input_options	non_registered, registered, both	non_registered
memory_type	Rom, single_port_ram, simple_dual_port_ram, dual_port_ram	single_port_ram
output_options	non_registered, registered, both	non_registered
pipeline_stages	0, 1	0
qualify_we_with_i_ce	true, false	false
reset_qdpo	true, false	false
reset_qsdpo	true, false	false
reset_qspo	true, false	false
simple_dual_port_address	non_registered, registered, both	non_registered
simple_dual_port_output_clock_enable	true, false	false
single_port_output_clock_enable	true, false	false
sync_reset_qdpo	true, false	false
sync_reset_qsdpo	true, false	false
sync_reset_qspo	true, false	false

Output Generation

See [Directory and File Contents in Chapter 6](#) for more details on the files created when the core is generated.

Constraining the Core









This core does not have any specific constraints except the the system clock constraint.

Detailed Example Design

This chapter provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx Vivado tools, the purpose and contents of the provided scripts and the contents of the example HDL wrappers.

Directory and File Contents

The output files generated by the Vivado IP catalog are placed in the <project_directory> top-level directory. Depending on the settings, the file output list may include some or all of the following files:

-  <project_directory>/<project_name.data>
Contains constraints and file set details.
-  <project_directory>/<project_name>.src/sources_1/ip/<component_name>
Contains the sources like XCI, XDC, TCL and document files.
-  <component_name>/synth
Contains the source file necessary to synthesize the Distributed Memory Generator
-  <component_name>/sim
Contains the source file necessary to simulate the Distributed Memory Generator
-  <component_name>/example_design
Contains the source file necessary to synthesize the example design
-  <component_name>/simulation
Contains the source file necessary to simulate the example design
 -  <component_name>/simulation/functional
Contains the simulation script file necessary to launch functional simulation of core including the example design
 -  <component_name>/simulation/timing
Contains the simulation script file necessary to launch timing simulation of core including the example design

The Distributed Memory Generator directories and their associated files are defined in the following sections.

<project_directory>/<project_name>.src/sources_1/ip/<component_name>

This directory contains templates for instantiation of the core, example design, synth, XML and the XCI files.

Table 6-1: <component_name> Directory

Name	Description
<component_name>.xci	Log file from Vivado tools describing which options were used to generate the Distributed Memory Generator. An XCI file can also be used as an input to the Vivado tools.
<component_name>.{veo vho}	VHDL or Verilog instantiation template.

<component_name>/synth

The synth directory contains the core synthesis file.

Table 6-2: synth Directory

Name	Description
<component_name>.vhd	A VHDL file from Vivado tools to synthesize the Distributed Memory Generator.

<component_name>/sim

The sim directory contains the core simulation wrapper file.

Table 6-3: sim Directory

Name	Description
<component_name>.vhd	A VHDL file from Vivado tools to simulate the Distributed Memory Generator.

<component_name>/example_design

The example design directory contains the example design files provided with the core.

Table 6-4: Example Design Directory

Name	Description
<component_name>_exdes.vhd	The VHDL top-level file for the example design; it instantiates the core. This file contains entity with the IO's required for the core configuration.
<component_name>_prod_exdes.xdc	Xilinx constraints file.
<component_name>_exdes.xdc	Provides an example clock constraint for processing the core using the Vivado implementation tools.

<component name>/simulation

The simulation directory contains the simulation files provided with the core.

Table 6-5: Simulation Directory

Name	Description
<component_name>_tb_pkg.vhd	VHDL File provided with demonstration test bench. It contains common functions required by the test bench.
<component_name>_tb_rng.vhd	VHDL File provided with demonstration test bench. It contains logic for pseudo random number generation.
<component_name>_tb_agen.vhd	VHDL File provided with demonstration test bench. It contains logic for address generation.
<component_name>_tb_dgen.vhd	VHDL File provided with demonstration test bench. It contains logic for random data generation.
<component_name>_tb_checker.vhd	VHDL File provided with demonstration test bench. It contains logic for verifying correctness Distributed Memory Generator core data output.
<component_name>_tb_stim_gen.vhd	VHDL File provided with demonstration test bench. It contains the test bench control logic and some checks.
<component_name>_tb_synth.vhd	VHDL File provided with demonstration test bench. This file has the instances and connections for of core and test bench modules.
<component_name>_tb.vhd	VHDL File provided with demonstration test bench. This is the top file for the test bench which generates the clock and reset signals. It also checks the test bench status.

<component name>/simulation/functional

The functional directory contains the scripts to launch XSIM/MIT Simulation with simulation test bench set as the top entity.

Table 6-6: Functional Directory

Name	Description
Simulate_xsim.[sh bat]	Compiles all the required files and launches simulation. The SH file is used in the Linux environment. the BAT file is used in the Windows environment.
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
Simulate_xsim.sh	XSim macro file that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion in the Linux operating system.

Table 6-6: Functional Directory (Cont'd)

Name	Description
Simulate_xsim.bat	XSim macro file that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion in the Microsoft Windows operating system.
wave_xsim.tcl	XSim macro file that opens a Wave window with top-level signals.

<component name>/simulation/timing

The timing directory contains the scripts to launch XSIM/MTI Simulation with simulation test bench set as top entity.

Table 6-7: Timing Directory

Name	Description
simulate_mti.[bat sh]	A Windows (.bat) or Linux script that runs the modelsim simulation.
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
Simulate_xsim.sh	XSim macro file that compiles the example design sources and the routed design model. The demonstration test bench then runs the timing simulation to completion in the Linux operating system.
Simulate_xsim.bat	XSim macro file that compiles the example design sources and the structural routed design model. The demonstration test bench then runs the timing simulation to completion in the Microsoft Windows operating system.
wave_xsim.tcl	XSim macro file that opens a Wave window with top-level signals.

Example Design

Figure 6-1 shows the configuration of the example design.

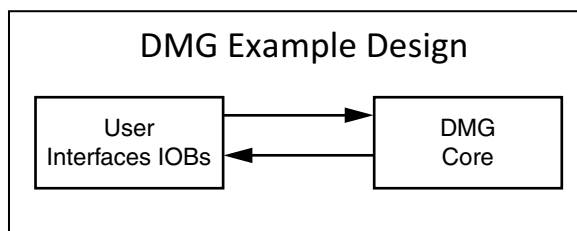


Figure 6-1: Example Design Block Diagram

The example design contains the following:

- An instance of the Distributed Memory Generator core. During simulation, the Distributed Memory Generator core is instantiated as a black box and replaced with the Vivado Synthesis Tool generated netlist/behavioral model for the functional simulation.
- Global clock buffers for top-level port clock signals.

Demonstration Test Bench

Figure 6-2 shows a block diagram of the demonstration test bench.

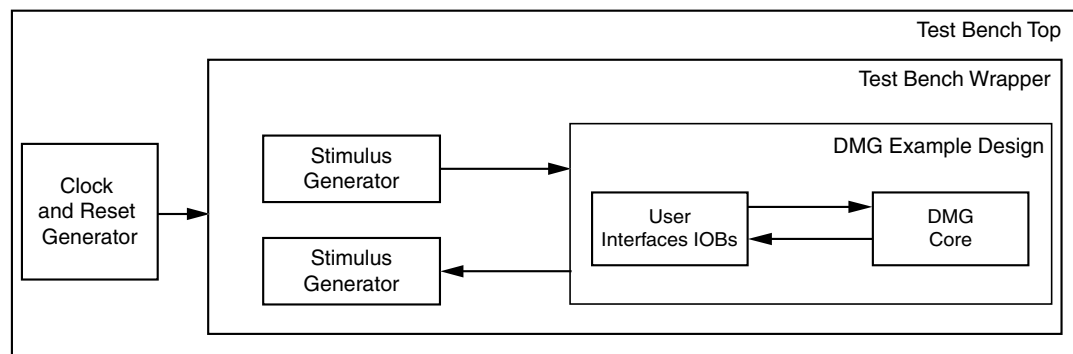


Figure 6-2: Test Bench Block Diagram

The demonstration test bench is a VHDL file to create 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

Simulation

VHDL and Verilog behavioral models of the Distributed Memory Generator core are provided with the XilinxCoreLib library for use within a simulation environment. The functional simulation model for this core is behavioral. Certain conditions such as out-of-range writes are not modelled in a cycle-accurate manner. Run the timing simulation to more closely simulate cycle-based behavior.

Static Timing Analysis

Static timing analysis can be performed using trce, following ngdbuild, map, and par.

Gate-level Simulation

If desired, the Vivado tools can create a UniSim-based VHDL or Verilog model for gate-level simulation, which is delivered with the NGC netlist. Alternatively, the netlist produced by the Vivado Synthesis tool (has to be run separately, by using **Run Synthesis** button in GUI) for the desired memory can be processed using ngdbuild and netgen to produce a SimPrim-based model for simulation.

Simulation Scripts

This section contains details about the test scripts included in the example design.

Functional Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are available from the following location:

```
<project_directory>/<project_name>.srcs/sources_1/ip/<component_name>/simulation/  
functional/
```

The test script performs these tasks:

- Compiles the behavioral model/structural UniSim simulation model
- Compiles HDL Example Design source code
- Compiles the demonstration test bench
- Starts a simulation of the test bench
- Runs the simulation to completion

Timing Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are located in:

```
<project_directory>/<project_name>.srcs/sources_1/ip/<component_name>/simulation/  
timing/
```

The test script performs these tasks:

- Compiles the SimPrim based gate level netlist simulation model
- Compiles the demonstration test bench
- Starts a simulation of the test bench

- Runs the simulation to completion

The simulation scripts expect the IP top-level RTL file (`component_name.[v|vhd]`) to be present at:

```
<project_directory>/<project_name>.srcs/sources_1/ip/<component_name>
```

To generate the required file, follow these steps:

1. Run synthesis using the GUI option **Run Synthesis** or using `launch_runs synth_1(name of synthesis file set created)` command line option
2. After successful completion of synthesis open the synthesized design using the Vivado GUI option **Open Synthesized Design** or `open_run synth_1(name of synthesis fileset created) -name <user defined name for netlist>` command line option
3. Then use one of the below command to generate the top level file in required language

- Verilog:

```
write_verilog -force -mode funcsim -security_mode ieee1735 -cell <component name>-rename_top_module <component name> <component name>.v
```

- VHDL:

```
write_vhdl -force -mode funcsim -security_mode ieee1735 -cell <component name> -rename_top <component name> <component name>.vhd
```

4. Copy the file in to `<project_directory>/<project_name>.srcs/sources_1/ip/<component_name>/`

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
```

SECTION III: ISE DESIGN SUITE

Customizing and Generating the Core

Constraining the Core

Detailed Example Design

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core.

GUI

The Distributed Memory Generator can be found in two places in the CORE Generator graphical user interface (GUI) View by Function pane.

To access the Distributed Memory Generator, do one of the following:

- Choose Basic Elements > Memory Elements.
- Choose Memories & Storage Elements > RAMs & ROMs.

CORE Generator Parameter Screens

The Distributed Memory Generator GUI uses three screens.

- Main Screen ([Figure 7-1](#))
- Input, Dual-Port, and Output Options Screen ([Figure 7-2](#))
- Initial Content and Reset Options Screen ([Figure 7-3](#))

All the screens share common tabs and buttons to provide information about the core and perform specific actions, such as generating the core and navigating among screens.

Main Screen

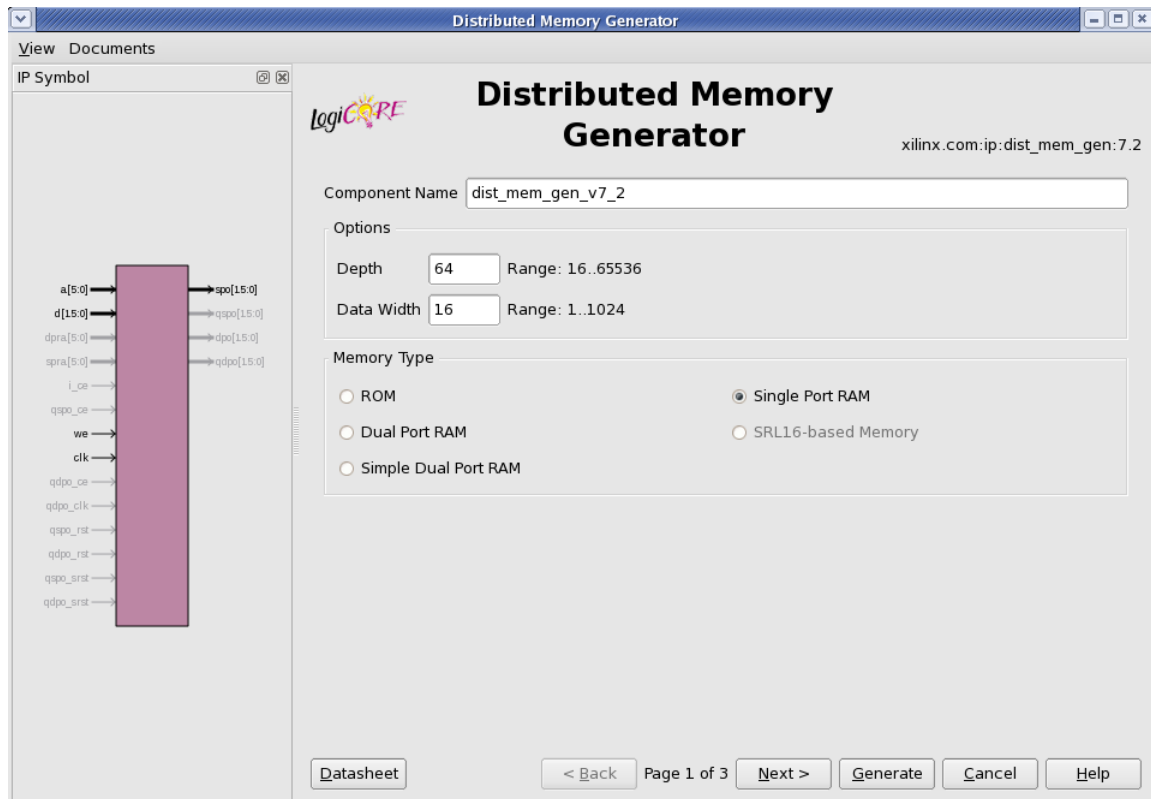


Figure 7-1: Main Screen

Component Name. The base name of the output files generated for the core. Names must begin with a letter and be composed of any of the following characters: a to z, 0 to 9 and “_”

Depth. Enter a value in the valid range from 16–65536, in steps of 16. Default value is 64.

Data Width: Enter the width of the memory in the valid range from 1–1024. The default value is 16.

Memory Type: Select one of five options. The default setting is single-port RAM.

- **ROM.** Figure 3-2 is a schematic of the structure of the ROM modules. The address register is optional (controlled by the setting of the Input Options parameter). Output registering is also optional (controlled by the setting of the Output Options parameter). Note that the clock is not required when these registers are not present. The resets and clock enables are optional.
- **Single-Port RAM.** Figure 3-4 is a schematic of the structure of the single-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). Output registering is also optional (controlled by the setting of the Output Options parameter). The resets and clock enables are optional. The clock

CLK is always required because writes to the single-port RAM are synchronous to that clock.

- **Dual-Port RAM.** Figure 3-6 is a schematic of the structure of the dual-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). The dual-port read address register is optional (controlled by the setting of the Dual-Port Address parameter). Output registers for both output ports are also optional (controlled by the setting of the Output Options parameter). When registered outputs are selected, the two output ports can be clocked by the same or different clock signals and can have the same or different clock enables (based on the settings selected for the Common Output Clock and Common Output CE parameters). All resets and clock enables are optional. Note that the clock CLK is always required, because writes to the RAM are synchronous to that clock.
- **SRL16-Based RAM.** Figure 3-8 is a schematic of the structure of the SRL16-based RAM module. The address and data registers are optional (controlled by the setting of the Input Options parameter). Output registers are also optional (controlled by the setting of the Output Options parameter). The resets and clock enables are optional. When the project family is set to Virtex-5 and above architectures, this option will be greyed-out.
- **Simple Dual-Port RAM.** Figure 3-10 is a schematic of the structure of the simple dual-port RAM modules. The address and data registers are optional (controlled by the setting of the Input Options parameter). The simple dual-port read address register is optional (controlled by the setting of the Simple Dual-Port Address parameter). Output registers for both output ports are also optional (controlled by the setting of the Output Options parameter).

When registered outputs are selected, the output port can be clocked by the same or different clock signals and can have the same or different clock enables (based on the settings selected for the Common Output Clock and Common Output CE parameters). All resets and clock enables are optional. Note that the clock CLK is always required because writes to the RAM are synchronous to that clock. When the project family is not set to the Zynq-7000, 7 series, Virtex-6, Virtex-5, or Spartan-6 family, this option will be greyed-out.

Input, Dual-Port, and Output Options Screen

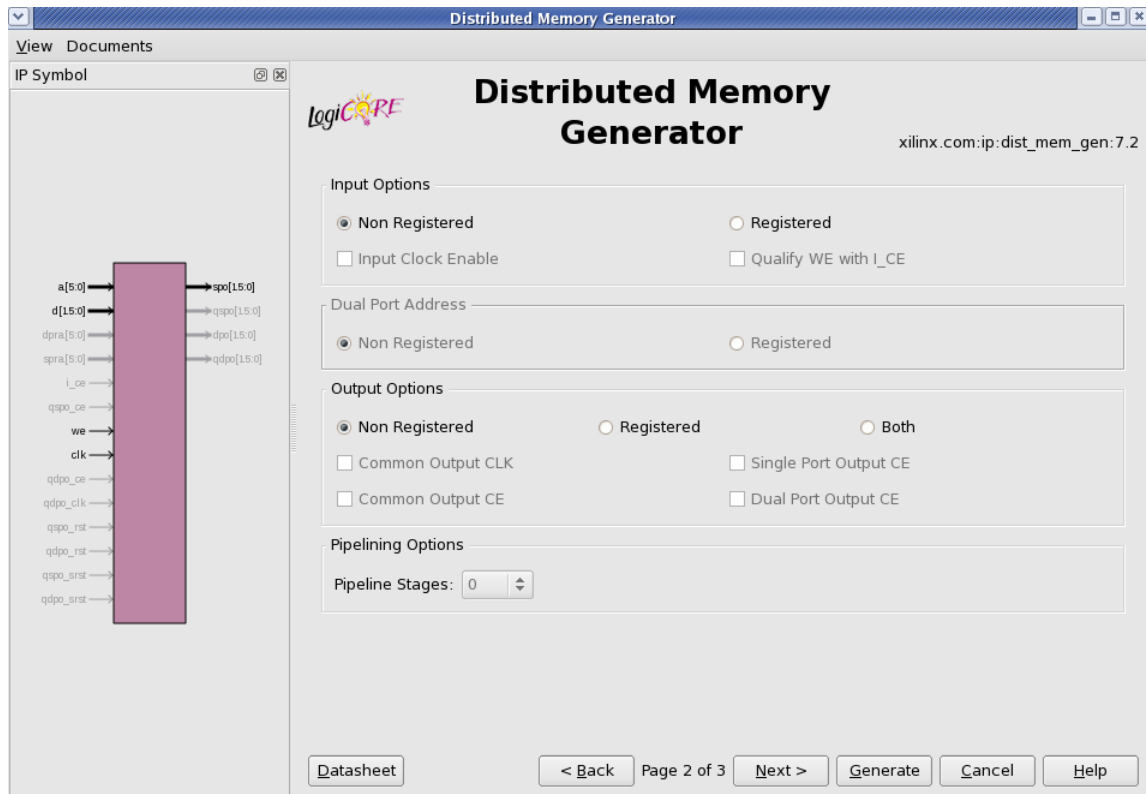


Figure 7-2: Distributed Memory Options Screen

Input Options. Select an option for the required input types. Non-registered is the default setting. Selecting Registered produces different effects depending on the selected memory type. For ROM, an address register is generated; for single-port RAM, simple dual-port, dual-port, and SRL16-based RAM, a register on the A[N:0] address input, a data input register, and a WE register are generated.

- **Input Clock Enable.** An optional input available when Input Options are set to Registered and Memory Type is *not* a ROM.
- **Qualify WE with I_CE.** Valid only for single-port RAM, simple dual-port RAM, dual-port RAM, and SRL16-based RAM with Input Options set to Registered and Input Clock Enable selected. When deselected, the WE register has no clock-enable control. When selected, the WE register has a clock enable driven by the I_CE input.

Dual/Simple Dual-Port Address. Valid only for simple dual-port RAM and dual-port RAMs, and controls the presence or absence of a register on the DPRA[N:0] inputs. Non-registered is the default setting.

Pipelining Options: When registered single-port RAMs, simple dual-port RAMs and dual-port RAMs are selected, an optional pipeline register can be placed into the output path.

- **Pipeline Stages.** Select '0' for no pipeline registers and '1' for a single pipeline stage in the output multiplexer.

Output Options: Select an option for the required output types. Non-registered is the default setting.

- **Single-Port Output Clock Enable.** Enabled for registered output memory or for input registered ROM to provide this optional pin.
- **Dual/Simple Dual-Port Output Clock Enable.** Enabled only for output registered simple dual-port RAM and dual-port RAMs to provide this optional pin.
- **Common Output CLK.** Enabled only for registered simple dual-port RAMs and dual-port RAMs. If not selected, the SPO registers are clocked by the CLK input and the DPO registers are clocked from the `QDPO_CLK` input. Default setting is selected, where all output registers are clocked from the CLK input.
- **Common Output CE.** Enabled only for non-registered simple dual-port RAMs and dual-port RAMs and only if Common Output Clock is also selected. If Common Output CE is deselected, the SPO register clocks are enabled by the `QSPO_CE` input and the DPO register clocks are enabled from the `QDPO_CE` input. Default setting is selected, where all output register clocks are enabled by the `QSPO_CE` input.

Initial Content and Reset Options

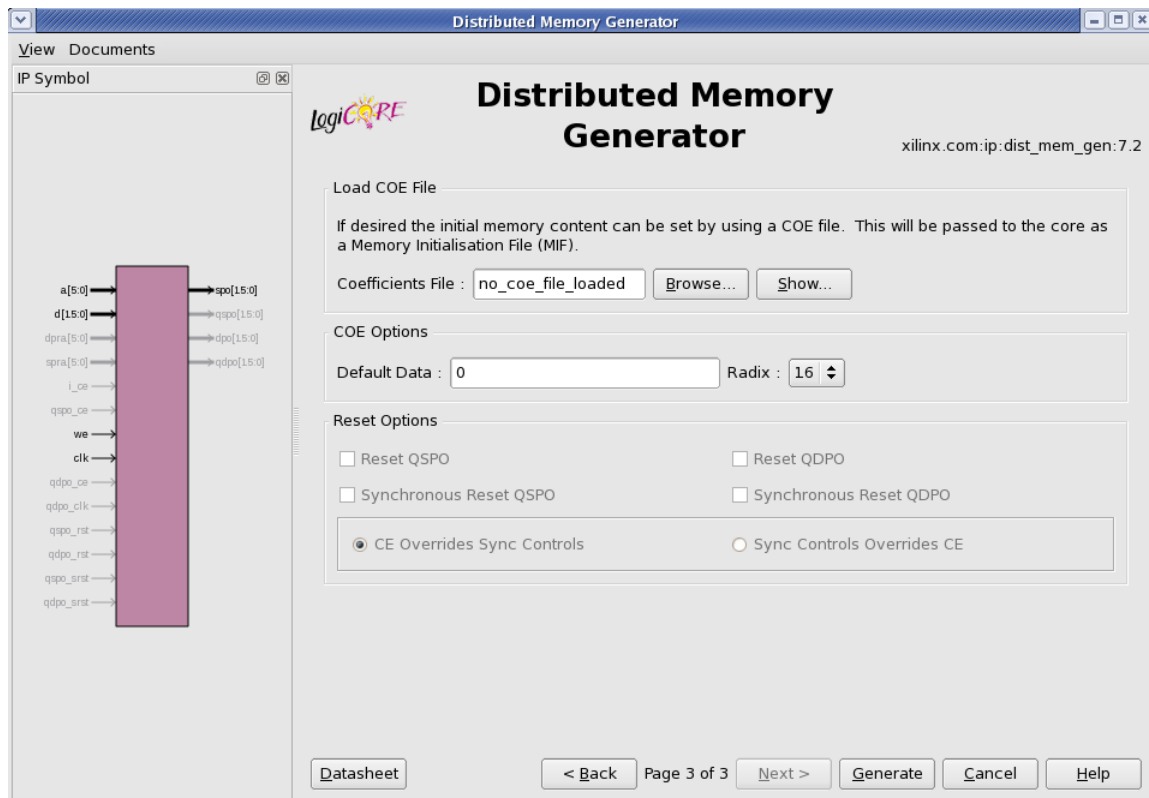


Figure 7-3: Initial Content and Reset Options Screen

Load COE File

The initial values of the memory elements can be set using a COE file.

- To load the COE file, click Browse.
- To view the initial contents, click Show.

For a description of the COE file, see [Specifying Memory Contents Using a COE File](#).

COE Options

Enter the initial value to be stored in memory locations not otherwise initialized in the COE file.

- **Default Data:** When no value is entered, the field defaults to 0. Values can be entered in binary, decimal, or hexadecimal formats, as defined by the Default Data Radix entry.
- **Radix:** Choose the radix of the Default Data value. Valid entries are 2, 10, and 16.

Reset Options

- **Reset QSPO:** Enabled only when the core has a registered single-port output. If selected, an asynchronous single-port output reset pin is available.
- **Reset QDPO:** Enabled only when the core has a registered simple dual-port and dual-port output. If selected, an asynchronous simple dual-port and dual-port output reset pin is available.
- **Synchronous Reset QSPO:** Enabled only when the core has a registered single-port output. If selected, a synchronous single-port output reset pin is available.
- **Synchronous Reset QDPO:** Enabled only when the core has a registered simple dual-port and dual-port output. If selected, a synchronous simple dual-port and dual-port output reset pin is available.
- **CE Overrides Sync Controls:** Enabled only when one of the synchronous reset options has been selected and an output clock enable has been selected. When selected, the synchronous control signals are qualified by the clock enable pin.
- **Sync Controls Overrides CE:** Enabled only when one of the synchronous reset options has been selected and an output clock enable has been selected. When selected, the synchronous control signals operate regardless of the state of the output clock enable signals.

Parameter Values in the XCO File

Parameter values are listed in [Table 7-1](#).

Table 7-1: Parameter Values

XCO Parameter Name	XCO Values	Default Values
ce_overrides	ce_overrides_sync_controls, sync_controls_overrides_ce	ce_overrides_sync_controls
coefficient_file	no_coe_file_loaded, coe file name	no_coe_file_loaded
common_output_ce	true, false	false
common_output_clk	true, false	false
component_name	dist_mem_gen_v7_2	dist_mem_gen_v7_2
data_width	1 - 1024	16
default_data	Any binary, decimal and hexadecimal values equal to the data width	0
default_data_radix	2, 10, 16	16
depth	16 – 65536 (multiples of 16)	64

Table 7-1: Parameter Values (Cont'd)

XCO Parameter Name	XCO Values	Default Values
dual_port_address	non_registered, registered, both	non_registered
dual_port_output_clock_enable	true, false	false
input_clock_enable	true, false	false
input_options	non_registered, registered, both	non_registered
memory_type	Rom, single_port_ram, simple_dual_port_ram, dual_port_ram, srl16_based	single_port_ram
output_options	non_registered, registered, both	non_registered
pipeline_stages	0, 1	0
qualify_we_with_i_ce	true, false	false
reset_qdpo	true, false	false
reset_qsdpo	true, false	false
reset_qspo	true, false	false
simple_dual_port_address	non_registered, registered, both	non_registered
simple_dual_port_output_clock_enable	true, false	false
single_port_output_clock_enable	true, false	false
sync_reset_qdpo	true, false	false
sync_reset_qsdpo	true, false	false
sync_reset_qspo	true, false	false

Output Generation

See [Directory and File Contents in Chapter 9](#) for the generated output file details.

Constraining the Core








This core does not have any specific constraints except the the system clock constraint.

Detailed Example Design

This chapter provides detailed information about the example design, including a description of files and the directory structure generated by the Xilinx® CORE Generator™ software, the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

Directory and File Contents

The Distributed Memory Generator core directories and their associated files are defined in the following sections. All files are contained in the top-level directory [<project_directory>](#).

-  [<project_directory>/<component_name>](#)
 - Core release notes file
 -  [<component_name>/doc](#)
 - Product documentation
 -  [<project_dir>/<component_name>/example design](#)
 - VHDL design files
 -  [<project_dir>/<component_name>/implement](#)
 - Implementation script files
 -  [<project_dir>/<component_name>/implement/results](#)
 - Results directory, created after implementation scripts are run, and contains implement script results
 -  [<project_dir>/<component_name>/simulation](#)
 - Simulation files
 -  [<project_dir>/<component_name>/simulation/functional](#)
 - Functional simulation scripts
 -  [<project_dir>/<component_name>/simulation//timing](#)
 - Timing simulation scripts

<project_directory>

The <project_directory> contains all the CORE Generator software project files.

Table 9-1: Project Directory

Name	Description
<component_name>.ngc	Top-level netlist
<component_name>.v[hd]	Verilog or VHDL simulation model
<component_name>.xco	CORE Generator software project-specific option file; can be used as an input to the CORE Generator software.
<component_name>_flist.txt	List of files delivered with the core.
<component_name>.{veo vho}	VHDL or Verilog instantiation template.

<project_directory>/<component name>

The <component name> directory contains the release notes file provided with the core, which can include last-minute changes and updates.

Table 9-2: Component Name Directory

Name	Description
dist_mem_gen_v7_2_readme.txt	Core name release notes file.

<component_name>/doc

The doc directory contains the PDF documentation provided with the core.

Table 9-3: Doc Directory

Name	Description
pg063-dist-mem-gen.pdf	Distributed Memory Generator v7.2 product guide

<project_dir>/<component_name>/example design

The example design directory contains the example design files provided with the core.

Table 9-4: Example Design Directory

Name	Description
<component_name>_exdes.ucf	Provides example constraints necessary for processing the Distributed Memory Generator core using the Xilinx implementation tools.
<component_name>_exdes.vhd	The VHDL top-level file for the example design; it instantiates the Distributed Memory Generator core. This file contains entity with the IO's required for the core configuration.

<project_dir>/<component_name>/implement

The implement directory contains the core implementation script files.

Table 9-5: Implement Directory

Name	Description
implement.{bat sh}	A Windows (.bat) or Linux script that processes the example design.
implement_synplify.{bat sh}	A Windows (.bat) or Linux script that synthesizes the core in Synplicity.
xst.prj	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator software project option is set to ISE or Other.
xst.scr	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator software Vendor project option is set to ISE or Other.

<project_dir>/<component_name>/implement/results

The results directory is created by the implement script, after which the implement script results are placed in the results directory.

Table 9-6: Results Directory

Name	Description
xst.scr	Implement script result files.

<project_dir>/<component_name>/simulation

The simulation directory contains the demo test bench files provided with the core.

Table 9-7: Simulation Directory

Name	Description
<component_name>_tb_pkg.vhd	VHDL File provided with demonstration test bench. It contains common functions required by the test bench
<component_name>_tb_rng.vhd	VHDL File provided with demonstration test bench. It contains logic for pseudo random number generation
<component_name>_tb_agen.vhd	VHDL File provided with demonstration test bench. It contains logic for address generation
<component_name>_tb_dgen.vhd	VHDL File provided with demonstration test bench. It contains logic for random data generation.
<component_name>_tb_checker.vhd	VHDL File provided with demonstration test bench. It contains logic for verifying correctness Distributed Memory Generator core data output.

Table 9-7: Simulation Directory (Cont'd)

Name	Description
<component_name>_tb_stim_gen.vhd	VHDL File provided with demonstration test bench. It contains the test bench control logic and some checks.
<component_name>_tb_synth.vhd	VHDL File provided with demonstration test bench. This file has the instances and connections for of core and test bench modules
<component_name>_tb.vhd	VHDL File provided with demonstration test bench. This is the top file for the test bench which generates the clock and reset signals. It also checks the test bench status.

<project_dir>/<component_name>/simulation/functional

The functional directory contains functional simulation scripts provided with the core.

Table 9-8: Functional Directory

Name	Description
simulate_mti.do	A macro file for ModelSim that compiles the HDL sources and runs the simulation.
simulate_mti.{bat sh}	A Windows (.bat) or Linux script that runs the modelsim simulation.

<project_dir>/<component_name>/simulation//timing

The timing directory contains timing simulation scripts provided with the core.

Table 9-9: Timing Directory

Name	Description
simulate_mti.do	A macro file for ModelSim that compiles the post-par timing netlist, demonstration test bench files, and runs the simulation.
simulate_mti.{bat sh}	A Windows (.bat) or Linux script that runs the ModelSim simulation.

Example Design

Figure 9-1 shows the configuration of the example design.

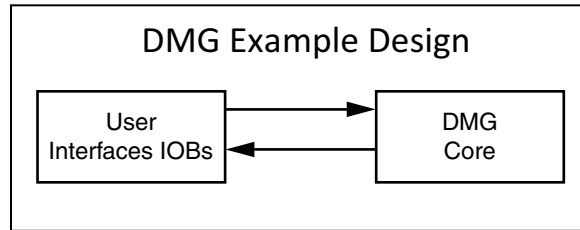


Figure 9-1: Example Design Block Diagram

The example design contains the following:

- An instance of the Distributed Memory Generator core. During simulation, the Distributed Memory Generator core is instantiated as a black box and replaced with the generated netlist/behavioral model created by CORE Generator for the functional simulation.
- Global clock buffers for top-level port clock signals.

Demonstration Test Bench

Figure 9-2 shows a block diagram of the demonstration test bench.

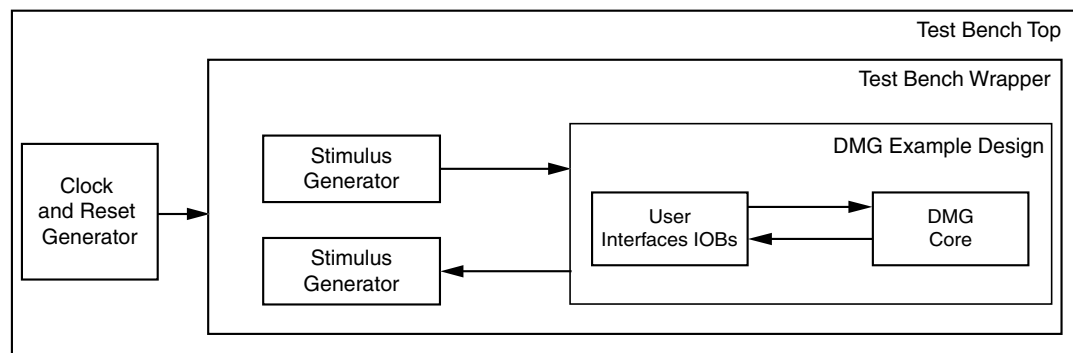


Figure 9-2: Test Bench Block Diagram

The demonstration test bench is a VHDL file to create 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

Implementation

The implementation script is either a shell script (SH) or batch file (BAT) that processes the example design through the Xilinx tool flow. It is located at:

- Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

- Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs these steps:

- Synthesizes the HDL example design files using XST
- Runs NGDBuild to consolidate the core netlist and the example design netlist into the NGD file containing the entire design
- Maps the design to the target technology
- Place-and-routes the design on the target device
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE)
- Generates a bitstream
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Simulation

VHDL and Verilog behavioral models of the Distributed Memory Generator core are provided with the XilinxCoreLib library for use within a simulation environment. Note that neither a test bench nor test fixture is provided with the Distributed Memory Generator.

The functional simulation model for this core is behavioral. Certain conditions such as out-of-range writes are not modelled in a cycle-accurate manner. Run timing simulation to more closely simulate cycle-based behavior.

Synthesis

Synthesis of the memory produced by the Distributed Memory Generator is performed with XST by the CORE Generator software.

Static Timing Analysis

Static timing analysis can be performed using `trce`, following `ngdbuild`, `map`, and `par`.

Gate-level Simulation

If desired, the CORE Generator software can create a UniSim-based VHDL or Verilog model for gate level simulation, which is delivered with the NGC netlist. Alternatively, the netlist produced by the CORE Generator for the desired memory can be processed using `ngdbuild` and `netgen` to produce a SimPrim-based model for simulation.

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
```

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Migrating

Additional Resources

Verification, Compliance, and Interoperability

Xilinx has verified the Distributed Memory Generator core in a proprietary test environment, using an internally developed bus functional model. Tens of thousands of test vectors were generated and verified, including both valid and invalid write and read data accesses.

Simulation

The Distributed Memory Generator has been tested with Xilinx ISE® Design Suite v14.2, Xilinx Vivado Design Suite v2012.2, Xilinx ISIM/XSIM, and Mentor Graphics ModelSim simulator.

Migrating

This appendix describes migrating from older versions of the IP to the current IP release.

Note: For details about migrating from the ISE Design Suite to the Vivado Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide*.

Parameter Changes in the XCO File

There were no XCO parameter, port or functionality changes since Distributed Memory Generator v6.1 core.

Auto-Upgrade Feature

When using the ISE Design Suite, the Distributed Memory Generator core can be seamlessly migrated from v6.1, v6.3, v6.3 or v7.1 to v7.2 using the auto-upgrade feature. Access this feature by right-clicking any preexisting Distributed Memory Generator core in your project in the Project IP tab of CORE Generator.

There are two types of upgrades:

- Select Upgrade Version, and Regenerate (Under Current Project Settings): This upgrades an older Distributed Memory Generator core version (6.1, 6.2 or 6.3) to the intermediate version you select (6.2, 6.3 or 7.1).
- Upgrade to Latest Version, and Regenerate (Under Current Project Settings): This automatically upgrades an older Distributed Memory Generator core to the latest version. Use this option to upgrade any earlier version of Distributed Memory Generator (6.1, 6.2, 6.3 and 7.1) to v7.2.

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

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.

See the IP Release Notes Guide ([XTP025](#)) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/25/12	1.0	Initial Xilinx release as a Product Guide. Replaces DS322, <i>LogiCORE IP Distributed Memory Generator Data Sheet</i> . Added support for Vivado Design Suite.

Notice of Disclaimer

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 the Limited Warranties which can be viewed at <http://www.xilinx.com/warranty.htm>; 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 Critical Applications: <http://www.xilinx.com/warranty.htm#critapps>.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.