Memory Modules

User Guide

FPGA-IPUG-02033 Version 1.1

January 2019

Contents

Figures

Tables

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
EBR	Embedded Block RAM
FIFO	First In First Out
FPGA	Field-Programmable Gate Array
IP	Intellectual Property
LUT	Lookup Table
PMI	Parameterized Module Instantiation
RTL	Register Transfer Level

1. Introduction

This technical note discusses memory usage for the FPGA devices supported by Lattice Radiant Software. It is intended to be used by design engineers as a guide to integrating the EBR (Embedded Block Random Access Memory) based memories for all device families in Lattice Radiant Software.

Behavioral code for Single-Port RAM and Pseudo Dual-Port RAM are supported. Synthesis tool is expected to infer sysMEM™ EBR.

Designers can utilize the memory primitives using two different methods described below.

- **Using Module/IP Block Wizard** – The Module/IP Block Wizard user interface allows you to specify the required memory type and size. Module/IP Block Wizard takes this specification and instantiates a synthesizable RTL (register transfer level) code and sets the appropriate parameters based on the user interface setting.
- **Using PMI (Parameterized Module Instantiation)** – PMI allows experienced users to skip the graphical user interface and utilize the configurable memory primitives on-the-fly from the Lattice Radiant Software project navigator. The parameters and the control signals needed in Verilog are set by the user. The top-level design has the instantiation of a synthesizable RTL code, which is inferred during synthesis.

2. Memory Modules

The following sections discuss the different memory modules available from the IP Catalog, the size of memory that each module can support, and other special options for the module. Module/IP Block Wizard automatically allows you to create memories larger than the width and depth supported for each memory primitive.

- **Output Register** – The output data of the memory is optionally registered at the output. You can choose this option by selecting the **Enable Output Register** check box in Module/IP Block Wizard while customizing the module.
- **Reset** – The EBRs also support the Reset signal. The Reset (or RST) signal only resets input and output registers of the RAM. It does not reset the contents of the memory.
- **Timing** – To correctly write into a memory cell in the EBR block, the correct address should be registered by the logic. Hence, it is important to note that while running the trace on the EBR blocks, there should be no setup and hold time violations on the address registers (address). Failing to meet these requirements can result in incorrect addressing and, consequently, corruption of memory contents.

During a read cycle, a similar issue can occur that involves the correct contents not being read if the address is not correctly registered in the memory.

A Post Place and Route timing report in Radiant Software can be run to verify that no such timing errors occur. Refer to the timing preferences in the Radiant Online Help.

2.1. Single Port RAM (RAM_DQ) – EBR Based

Lattice FPGAs support all features of the Single Port Memory Module or RAM_DQ. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirement.

Module/IP Block Wizard generates the memory module, as shown in Figure 2.1.

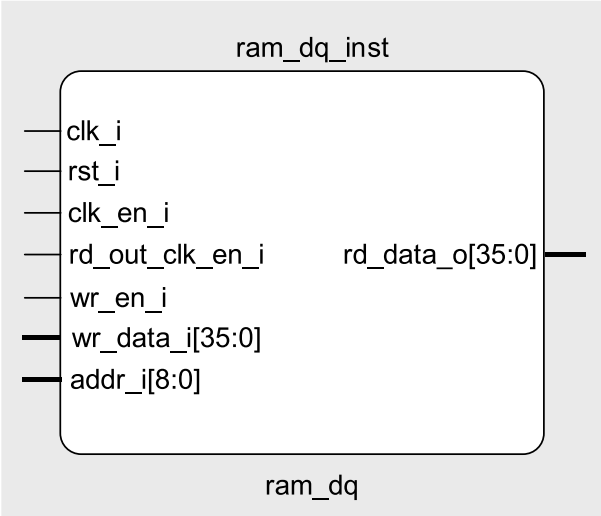

Figure 2.1. Single-Port Memory Module Generated by Module/IP Block Wizard

The various ports and their definitions for Single-Port Memory are listed in Table 2.1. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.1. EBR-Based Single-Port Memory Port Definitions

Port Name	Direction	Width	Description
clk_i	Input	1	Clock
rst_i	Input	1	Reset
clk_en_i	Input	1	Clock Enable
rd_out_clk_en_i	Input	1	Read Output Register Clock Enable
wr_en_i	Input	1	Write Enable
wr_data_i	Input	Data Width	Data Input
addr_i	Input	Address Width	Address Bus
ben_i*	Input	Byte Size Width	Byte Enable port
rd_data_o	Output	Data Width	Data Output

The various attributes available for the Single-Port Memory (RAM_DQ) are listed in Table 2.2. Some of these attributes are user-selectable through the Module/IP Block Wizard interface.

The attributes without selectable options in Module/IP Block Wizard are handled by the engine. However, working with the direct primitive instantiation can access these options.

NEGATIVE CLOCK SUPPORT – The Soft IP ram_dq can support negative-edge read and write clocks. In order to use the feature, you need to add an inverter on the clk_i signal. This inverter would not infer any LUT/FPGA resource.

Table 2.2. EBR-Based Single-Port Memory Attribute Definitions

Attributes	Description	Values	Default Value
Configuration Attributes			
Address Depth	Address depth of the Read and Write port.	2 – 61440	512
Data Width	Data word width of the Read and Write port.	1 – 512	36
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	True, False	True
Enable Output ClockEn	Clock Enable for the output clock (this option requires enabling output register).	True, False	False
Reset Assertion	Selection for Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
Enable Byte Enable[2]	Enables the Byte Enable function for the write port.[1]	True, False	False
Initialization Attributes			
Memory Initialization	Allows you to initialize memories to all 1s, 0s, or provide custom initialization through a memory file.	none, all 0s, all 1s, memory file	none
Memory File	When Memory File is selected, you can browse to the memory file for custom initialization of RAM.	—	none
Memory File Format	This option allows you to select if the memory file is formatted as Binary or Hex (see the Initialization File Formats section for details on the different memory file formats).	binary, hex	hex

Notes:

1. Byte-Enable can only be used for write ports >= 16 bits.
2. New for 1.1.

The Single Port RAM timing waveforms are shown in Figure 2.2 and Figure 2.3.

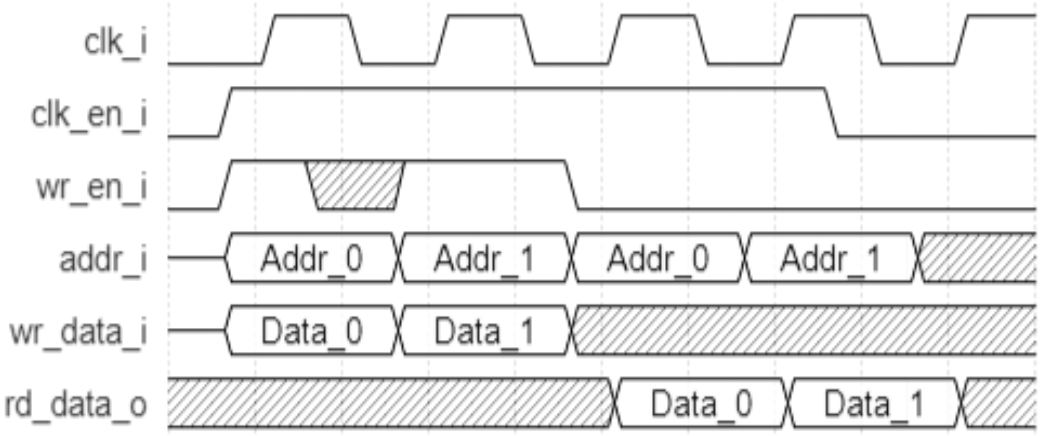

Figure 2.2. Single Port RAM Timing Waveform in Normal (NORMAL) Mode, without Output Registers

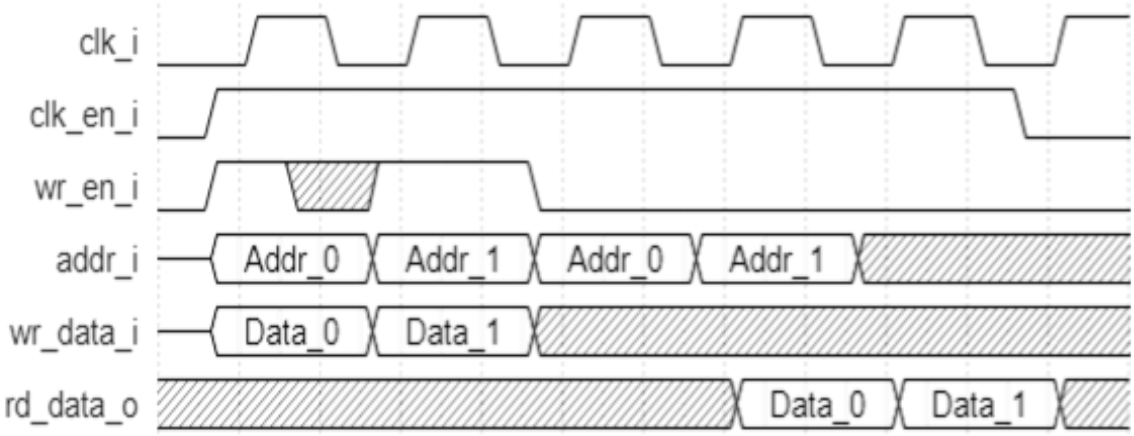

Figure 2.3. Single Port RAM Timing Waveform in Normal (NORMAL) Mode, with Output Registers

2.2. Pseudo Dual-Port RAM (RAM_DP) – EBR Based

Lattice FPGAs support all features of the Pseudo-Dual Port Memory Module or RAM_DP. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirement.

Module/IP Block Wizard generates the memory module shown in Figure 2.4.

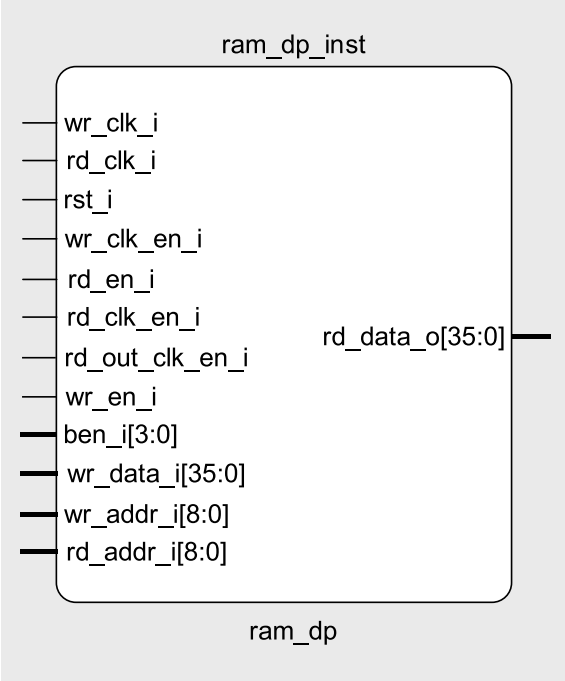

Figure 2.4. Pseudo Dual-Port Memory Module Generated by Module/IP Block Wizard

The various ports and their definitions for Pseudo Dual-Port memory are listed in Table 2.3. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.3. EBR-Based Pseudo Dual-Port Memory Port Definitions

Port Name	Direction	Width	Description
wr_clk_i	Input	1	Write Clock
rd_clk_i	Input	1	Read Clock
rst_i	Input	1	Reset
wr_clk_en_i	Input	1	Write Clock Enable
rd_clk_en_i	Input	1	Read Clock Enable
rd_out_clk_en_i	Input	1	Read Output Register Clock Enable
wr_en_i	Input	1	Write Enable
wr_data_i	Input	Write Port Data Width	Write Data
wr_addr_i	Input	Write Port Address Width	Write Address
rd_en_i	Input	1	Read Enable
rd_addr_i	Input	Read Address Width	Read Address
ben_i*	Input	Byte Size Width	Byte Enable port
rd_data_o	Output	Read Port Data Width	Read Data

***Note:** New for 1.1.

The various attributes available for the Pseudo Dual-Port Memory (RAM_DP) are listed in Table 2.4. Some of these attributes are user-selectable through the Module/IP on Local.

Table 2.4. EBR-Based Pseudo Dual-Port Memory Attribute Definitions

Attributes	Description	Values	Default Value
General Attributes			
Write Port Address Depth	Write Port Address depth.	2 – 61440	512
Write Port Data Width	Write Port Data word width.	1 – 256	36
Read Port Address Depth	Read Port Address depth.[1, 2]	2 – 61440	512
Read Port Data Width	Read Port Data word width.[1, 2]	1 – 256	36
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	True, False	True
Enable Output ClockEn	Clock Enable for the output clock (this option requires enabling output register).	True, False	False
Reset Assertion	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
Enable Byte Enable[5]	Enables the Byte Enable function for the write port.[3, 4]	True, False	False
Initialization Attributes			
Memory Initialization	Allows you to initialize their memories to all 1s, 0s, or providing custom initialization through a memory file.	none, all 0s, all 1s, Memory file	none
Memory File	When Memory file is selected, you can browse to the memory file for custom initialization of RAM.	—	none
Memory File Format	This option allows you to select if the memory file is formatted as Binary or Hex (see the Initialization File Formats section for details on the different formats).	binary, hex	hex

Notes:

1. For mixed width configurations total bit size must be equal (that is if WDEPTH*WDATA = RDEPTH*RDATA).
2. For mixed width configuration, the ratio between max DATA and min DATA must be power of 2. (that is if WDATA > RDATA, then 2^n = (WDATA/RDATA), where n must be a positive integer and 2^n <= 8 [iCE40UP device]).
3. Byte-enable for mixed width configuration is applicable only when max DATA = 2*min DATA (that is if WDATA > RDATA, then FACTOR = 2 = (WDATA/RDATA).
4. Byte-Enable can only be used for write ports >= 16 bits.
5. New for 1.1.

NEGATIVE CLOCK SUPPORT – The Soft IP ram_dp can support negative-edge read and write clocks. In order to use the feature, you need to add an inverter on the clk_i signal. This inverter would not infer any LUT/FPGA resource.

You have the option to enable the output registers for Pseudo-Dual Port RAM (RAM_DP). The internal timing waveforms for Pseudo-Dual Port RAM (RAM_DP) with these options are shown in Figure 2.5 and Figure 2.6. A mixed width timing with no output registers are also provided in Figure 2.7.

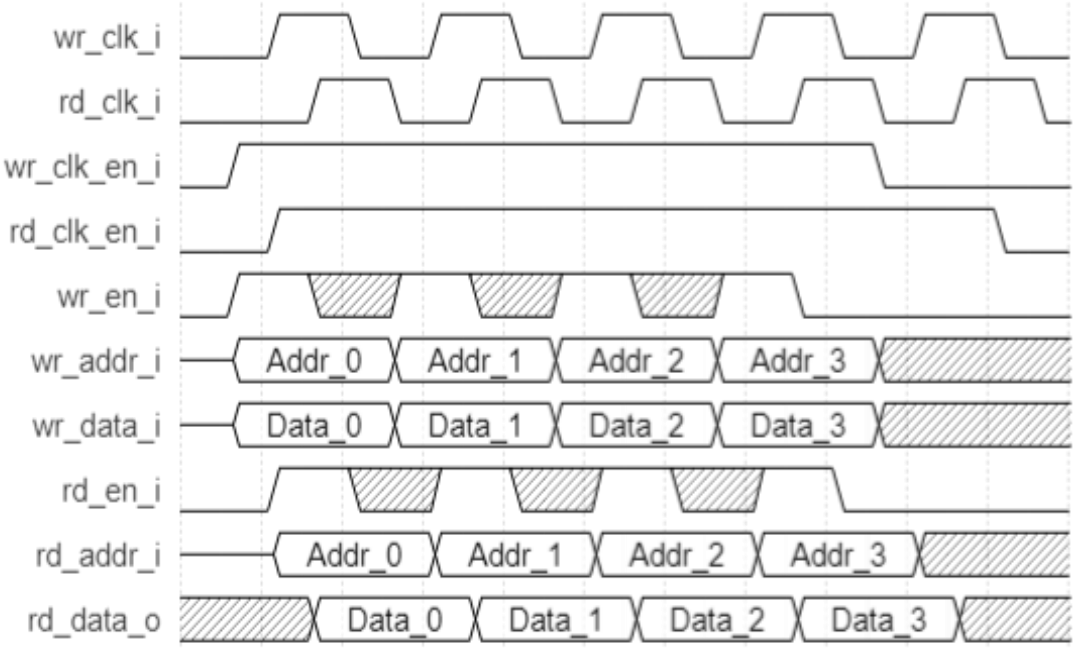

Figure 2.5. PSEUDO DUAL PORT RAM Timing Diagram, without Output Registers

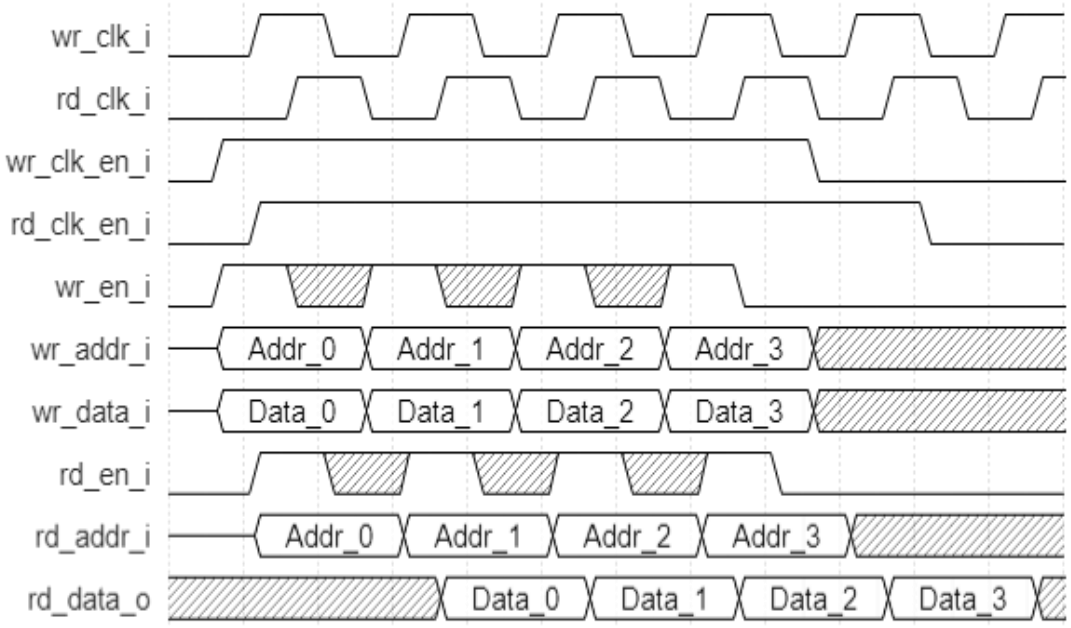

Figure 2.6. PSEUDO DUAL PORT RAM Timing Diagram, with Output Registers

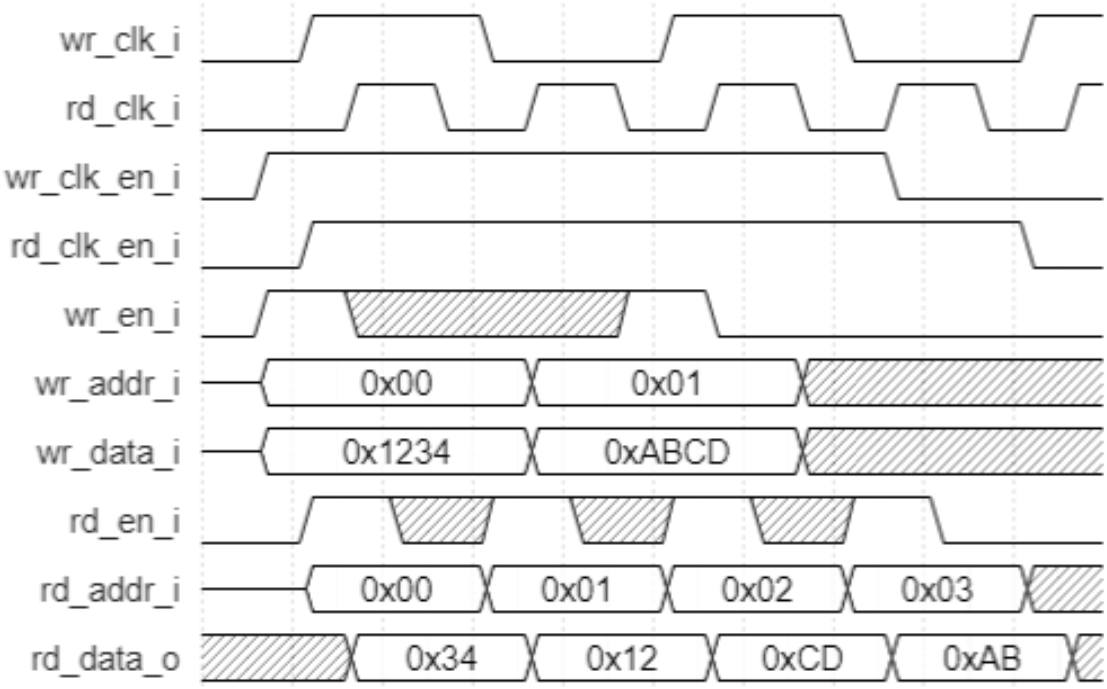

Figure 2.7. PSEUDO DUAL PORT RAM Timing Diagram, Mixed-Width without Output Registers

2.3. Read Only Memory (ROM) – EBR Based

Lattice FPGAs support all the features of Read Only Memory Module or ROM. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirements.

Module/IP Block Wizard generates the memory module shown in Figure 2.8.

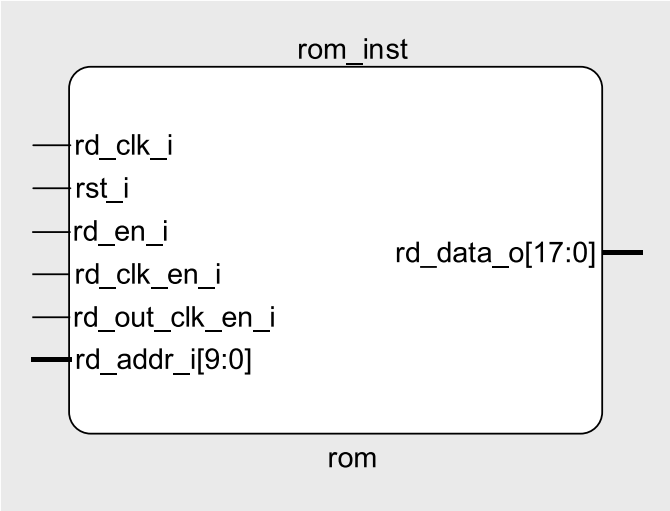

Figure 2.8. Read Only Memory Module Generated by Module/IP Block Wizard

The various ports and their definitions for Read Only Memory are listed in Table 2.5. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.5. EBR-Based Read Only Memory Port Definitions

Port Name	Direction	Width	Description
rd_clk_i	Input	1	Read Clock input
rst_i	Input	1	Output Reset
rd_clk_en_i	Input	1	Read Clock Enable
rd_out_clk_en_i	Input	1	Read Out Clock Enable
rd_en_i	Input	1	Read Enable
rd_addr_i	Input	Read Address Width	Read Address
rd_data_o	Output	Read Data Width	Read Data

The various attributes available for the Read Only Memory (ROM) are listed in Table 2.6. Some of these attributes are user-selectable through the Module/IP on Local.

Table 2.6. EBR-Based Read Only Memory Attribute Definitions

Attributes	Description	Values	Default Value
General Attributes			
Read Port Address Depth*	Read Port Address Depth.	2 – 61440	1024
Read Port Data Width*	Read Port Data Width.	1 – 256	18
Enable Output Register	Data Out (Q) can be registered or not using this selection.	True, False	True
Enable Output ClockEn	Clock Enable for the output clock of (this option requires enabling output register).	True, False	False
Reset Assertion	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
Initialization Attributes			
Memory Initialization	Allows you to initialize the memory by providing a custom initialization through a memory file.	Memory file	Memory file
Memory File	When Memory file is selected, you can browse to the memory file for custom initialization of RAM.	—	none
Memory File Format	This option allows you to select if the memory file is formatted as Binary or Hex (see the Initialization File Formats section for details on the different formats).	binary, hex	hex

***Note:** This IP requires a minimum of 30-bit total size to be properly implemented.

NEGATIVE CLOCK SUPPORT – The Soft IP rom can support negative-edge read clocks. In order to use the feature, you need to add an inverter on the clk_i signal. This inverter would not infer any LUT/FPGA resource.

The Read Only Memory timing waveforms are shown in Figure 2.9 and Figure 2.10.

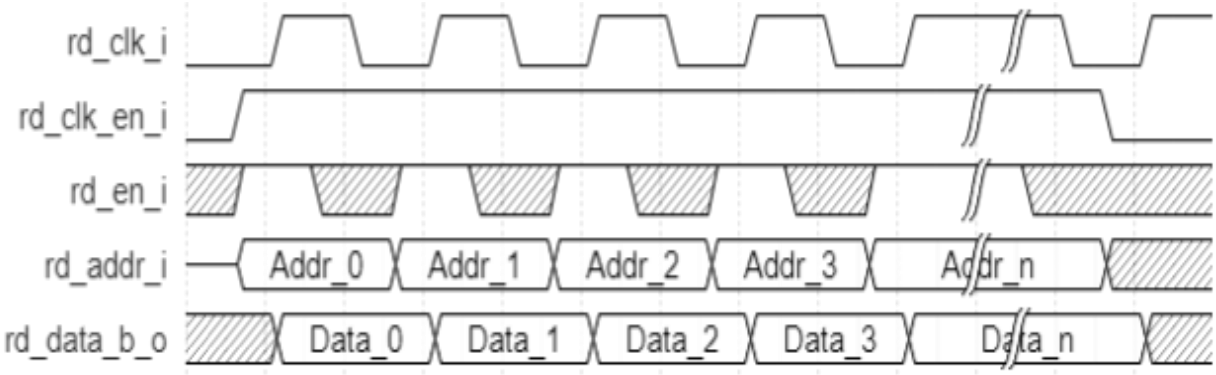

Figure 2.9. ROM Timing Diagram, without Output Registers

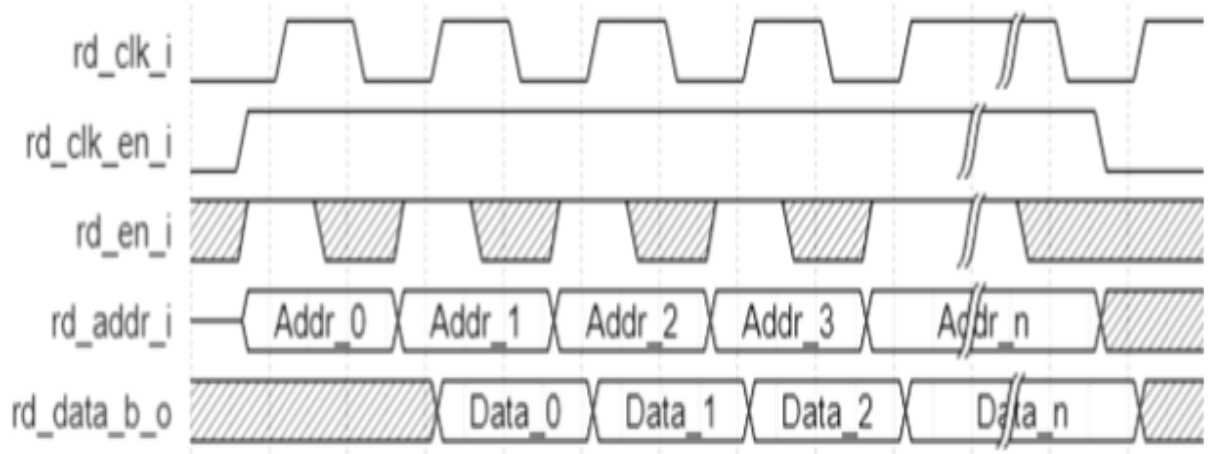

Figure 2.10. ROM Timing Diagram, with Output Registers

2.4. First In First Out Single Clock (FIFO)

Lattice FPGAs support all the features of First In First Out Single Clock Memory Module or FIFO. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirement.

Module/IP Block Wizard generates the memory module shown in Figure 2.11.

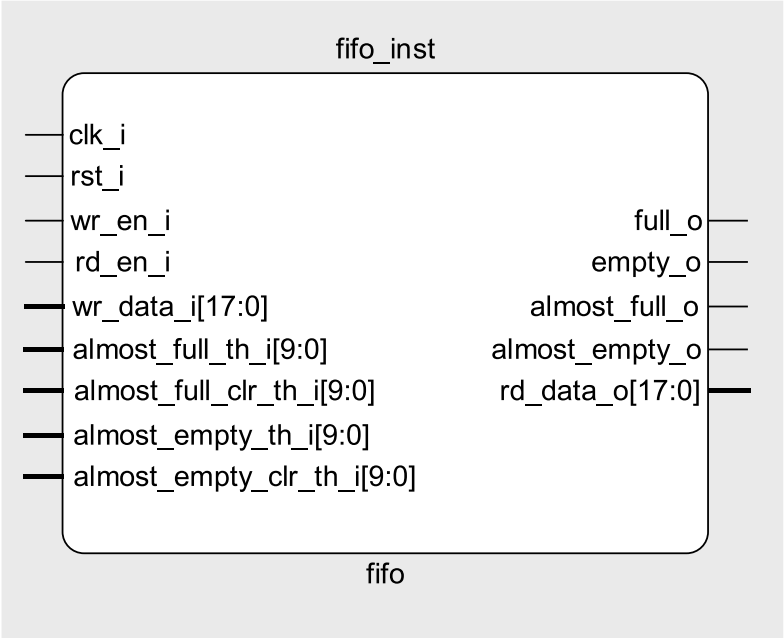

Figure 2.11. Distributed Pseudo Dual Port Module Generated by Module/IP Block Wizard

The various ports and their definitions for First in First Out Single Clock are listed in Table 2.7. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.7. First in First Out Single Clock Port Definitions

Port Name	Direction	Width	Description
clk_i	Input	1	Clock
rst_i	Input	1	Reset
wr_en_i	Input	1	Write Enable
rd_en_i	Input	1	Read Enable
wr_data_i	Input	Data Width	Data Input
almost_full_th_i	Input	Address Width	Almost full threshold (needs single/dual dynamic almost full selected)
almost_full_clr_th_i	Input	Address Width	Almost full clear threshold (needs dual dynamic almost full selected)
almost_empty_th_i	Input	Address Width	Almost empty threshold (needs single/dual dynamic almost empty selected)
almost_empty_clr_th_i	Input	Address Width	Almost empty clear threshold (needs dual dynamic almost empty selected)
rd_data_o	Output	Data Width	Data Output
full_o	Output	1	Full Flag
empty_o	Output	1	Empty Flag
almost_full_o	Output	1	Almost Full Flag
almost_ empty_o	Output	1	Almost Empty Flag
data_cnt_o	Output	Address Width + 1	Data Count – carries the number of data written to the FIFO. (requires data count selected).

The various attributes available for the First in First Out Single Clock (FIFO) are listed in Table 2.8. Some of these attributes are user-selectable through the Module/IP Block Wizard interface.

Table 2.8. First in First Out Single Clock Attribute Definitions

Attributes	Description	Values	Default Value
Configuration Attributes			
Address Depth	Address depth of the Read and Write port.	2 – 32768[1]	1024
Data Width	Data word width of the Read and Write port.	1 – 256	18
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	True, False	True
Reset Assertion	Selection for Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
Enable Almost Full Flag	Enables or disables the functionality of the almost full flag.	enable, disable	enable
Almost Full Assertion	Checks how the almost full flag is implemented. Either dynamic or static, with or without clearing. (requires Enable Almost full flag enabled).	Static – Single Threshold, Static – Dual Threshold, Dynamic – Single Threshold, Dynamic – Dual Threshold.	Static – Dual Threshold
Full Assert Level	The current address when written, flags the almost full flag. (Requires Static – Single / Dual Threshold).	0 < Full Assert Level < Address Depth	1023
Full Deassert Level	The current address when read, clears the almost full flag. (Requires Static – Dual Threshold).	0 < Full Deassert Level < Full Assert Level	1020
Enable Almost Empty Flag	Enables or disables the functionality of the almost empty flag.	enable, disable	enable
Almost Empty Assertion	Checks how the almost empty flag is implemented. Either dynamic or static, with or without clearing. (requires Enable Almost empty flag enabled).	Static – Single Threshold, Static – Dual Threshold, Dynamic – Single Threshold, Dynamic – Dual Threshold.	Static – Dual Threshold
Empty Assert Level	The current address when read, flags the almost empty flag (requires Static – Single / Dual Threshold).	0 < Empty Assert Level < Address Depth	1
Empty Deassert Level	The current address when written, clears the almost empty flag (requires Static – Dual Threshold).	Empty Assert Level < Empty Deassert Level < Address Depth	4
Enable Data Count	Enables counting the number of data written to the FIFO.	enable, disable	disable
Implementation[1, 2]	Chooses how the FIFO memory is implemented.	EBR, LUT	EBR

Notes:

1. For LUT implementations, maximum ADDRESS_DEPTH is 8192 only.
2. EBR implementation requires a minimum of 30-bit total size to be properly implemented.

Timing diagrams for FIFO Single clock are shown in the Figures below, Figure 2.12 and Figure 2.13, for configurations without register and configuration with registers, respectively. (FIFO Configuration uses ADDRESS_DEPTH = 4).

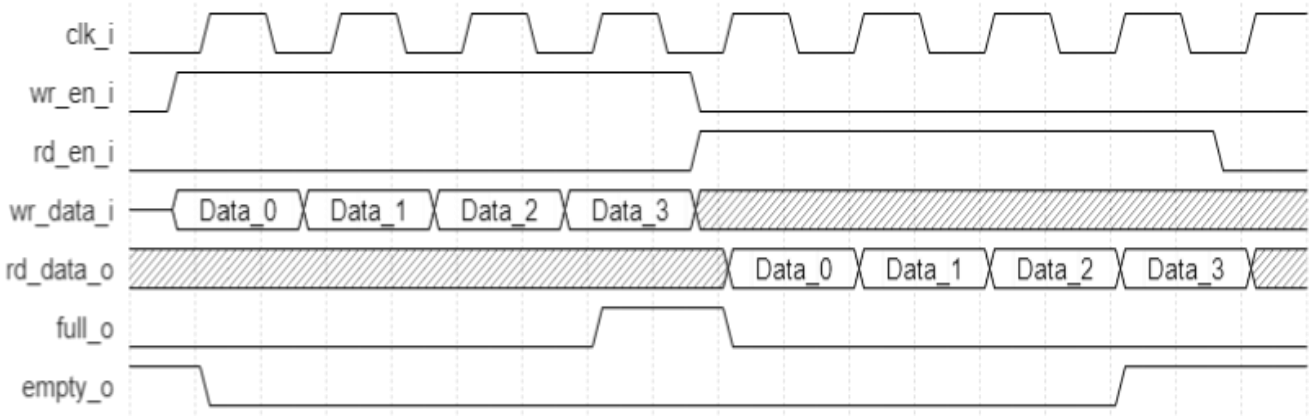

Figure 2.12. FIFO Single Clock without registers

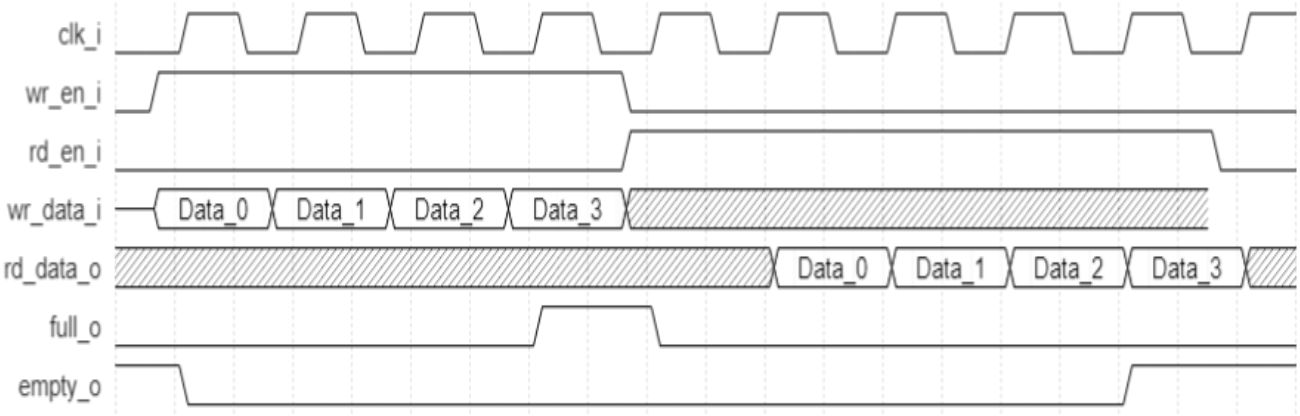

Figure 2.13. FIFO Single Clock with registers

2.5. First In First Out Dual Clock (FIFO_DC)

Lattice FPGAs support all the features of First In First Out Dual Clock Memory Module or FIFO_DC. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirement.

Module/IP Block Wizard generates the memory module shown in Figure 2.14.

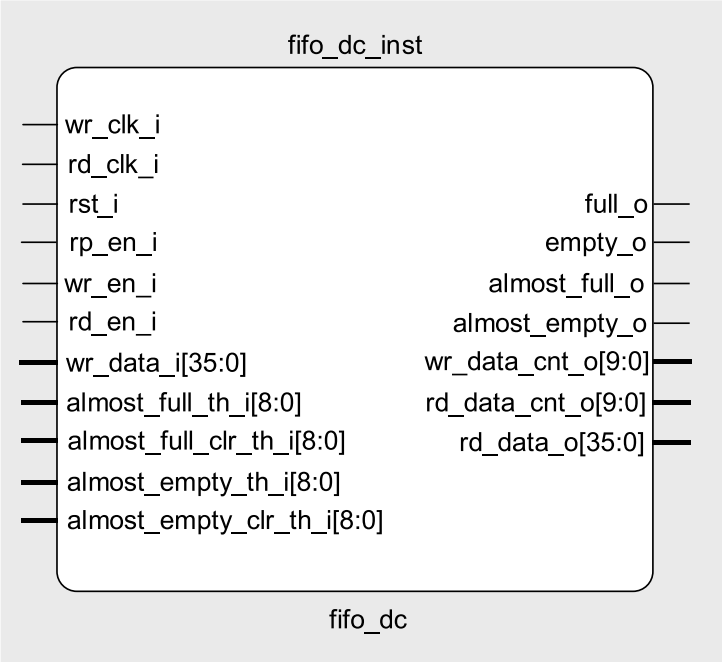

Figure 2.14. FIFO Dual Clock Module Generated by Module/IP Block Wizard

The various ports and their definitions for First in First Out Dual Clock are listed in Table 2.9. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.9. First in First Out Dual Clock Port Definitions

Port Name	Direction	Width	Description
wr_clk_i	Input	1	Write Clock
rd_clk_i	Input	1	Read Clock
rst_i	Input	1	FIFO Empty Reset
rp_rst_i	Input	1	FIFO Full Reset
wr_en_i	Input	1	Write Enable
rd_en_i	Input	1	Read Enable
wr_data_i	Input	Write Data Width	Data Input
almost_full_th_i	Input	Write Address Width	Almost full threshold (needs single/dual dynamic almost full selected)
almost_full_clr_th_i	Input	Write Address Width	Almost full clear threshold (needs dual dynamic almost full selected)
almost_empty_th_i	Input	Read Address Width	Almost empty threshold (needs single/dual dynamic almost empty selected)
almost_empty_clr_th_i	Input	Read Address Width	Almost empty clear threshold (needs dual dynamic almost empty selected)
rd_data_o	Output	Read Data Width	Data Output
full_o	Output	1	Full Flag
empty_o	Output	1	Empty Flag
almost_full_o	Output	1	Almost Full Flag

almost_ empty_o	Output	1	Almost Empty Flag
wr_data_cnt_o	Output	Write Address Width + 1	Write Data Count – carries the number of data written to the fifo. (requires data count selected).
rd_data_cnt_o	Output	Read Address Width + 1	Read Data Count – carries the number of data read from the fifo. (requires data count selected).

The various attributes available for the First in First Out Dual Clock (FIFO_DC) are listed in Table 2.10. Some of these attributes are user-selectable through the Module/IP Block Wizard interface.

Table 2.10. First in First Out Dual Clock Attribute Definitions

Attributes	Description	Values	Default Value
Configuration Attributes			
Write Port Address Depth	Write Port Address depth.	2 – 32768	512
Write Port Data Width	Write Port Data word width.	1 – 256	18
Read Port Address Depth	Read Port Address depth.[1,2]	2 – 32768	512
Read Port Data Width	Read Port Data word width.[1,2]	1 – 256	18
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	True, False	True
Reset Assertion	Selection for Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
Enable Almost Full Flag	Enables or disables the functionality of the almost full flag.	enable, disable	enable
Almost Full Assertion	Checks how the almost full flag is implemented. Either dynamic or static, with or without clearing. (requires Enable Almost full flag enabled).	Static – Single Threshold, Static – Dual Threshold, Dynamic – Single Threshold, Dynamic – Dual Threshold.	Static – Dual Threshold
Full Assert Level	The current address when written, flags the almost full flag (requires Static – Single / Dual Threshold).	0 < Full Assert Level < Write Address Depth	511
Full Deassert Level	The current address when read, clears the almost full flag (requires Static – Dual Threshold).	0 < Full Deassert Level < Full Assert Level	508
Enable Almost Empty Flag	Enables or disables the functionality of the almost empty flag	enable, disable	enable
Almost Empty Assertion	Checks how the almost empty flag is implemented. Either dynamic or static, with or without clearing (requires Enable Almost empty flag enabled).	Static – Single Threshold, Static – Dual Threshold, Dynamic – Single Threshold, Dynamic – Dual Threshold.	Static – Dual Threshold
Empty Assert Level	The current address when read, flags the almost empty flag (requires Static – Single / Dual Threshold).	0 < Empty Assert Level < Read Address Depth	1
Empty Deassert Level	The current address when written, clears the almost empty flag (requires Static – Dual Threshold).	Empty Assert Level < Empty Deassert Level < Read Address Depth	4
Enable Data Count (Write)	Enables counting the amount of data written to the FIFO_DC.	enable, disable	disable
Enable Data Count (Read)	Enables counting the amount of data read from the FIFO_DC.	enable, disable	disable
Implementation[3]	Chooses how the FIFO_DC memory is implemented.	EBR, LUT	EBR

Notes:

1. For mixed width configurations, total bit size must be equal (that is if WDEPTH*WDATA = RDEPTH*RDATA).
2. For mixed width configuration, the ratio between max DATA and min DATA must be power of 2. (that is if WDATA > RDATA, then 2^n = (WDATA/RDATA), where n must be a positive integer and 2^n <= 8 [iCE40UP device]).
3. Mixed width implementations cannot be used with LUT memory.

Timing diagrams for FIFO Dual clock are shown in the Figures below, Figure 2.15 and Figure 2.16, for configurations without register and configuration with registers, respectively. Additional timing diagram has been provided for Figure 2.17 for mixed-width configuration without registers (FIFO_DC Configuration uses WADDR_DEPTH = 4).

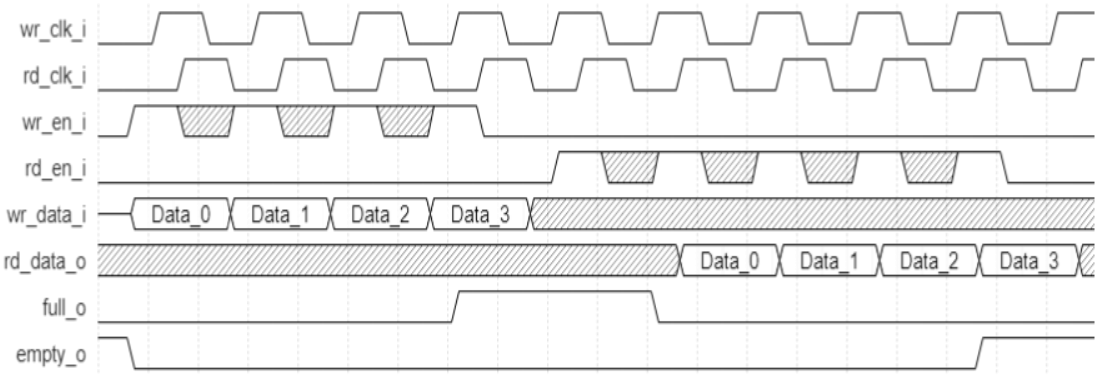

Figure 2.15. FIFO Dual Clock Module without registers

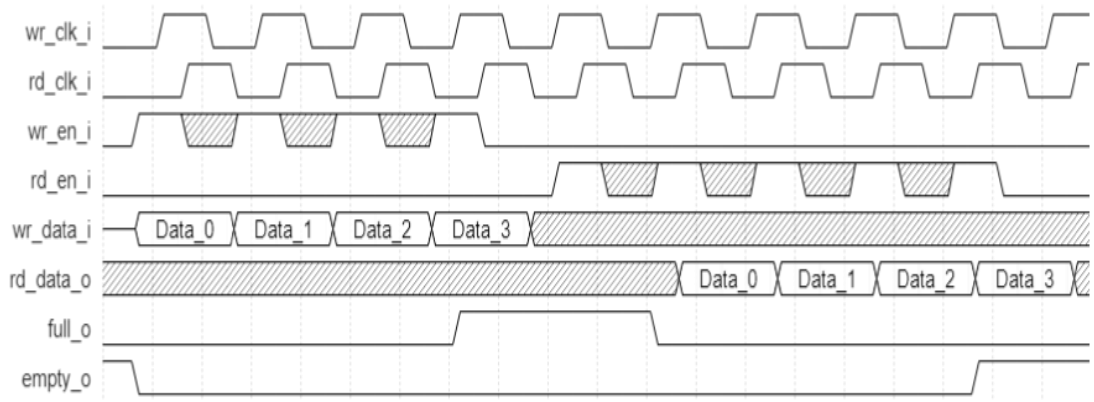

Figure 2.16. FIFO Dual Clock Module with registers

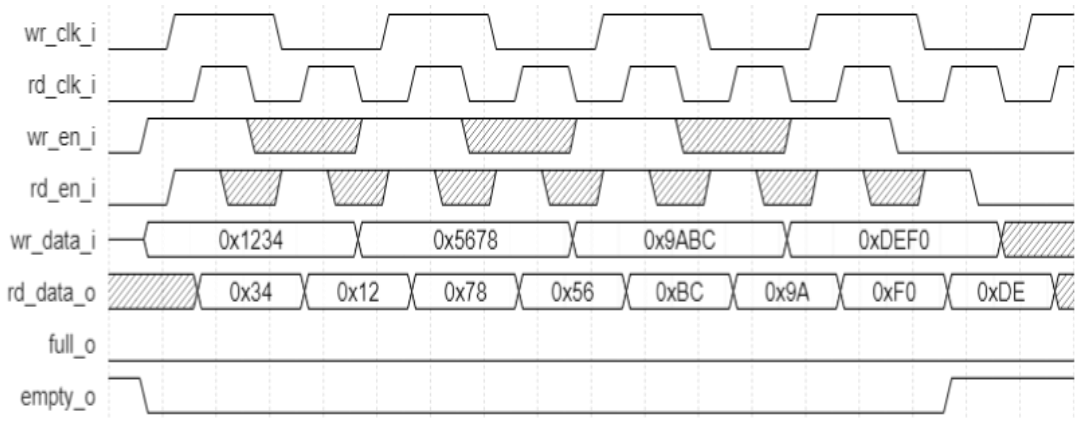

Figure 2.17. FIFO Dual Clock Module, Mixed-Width without registers

2.6. Shift Register (Shift_Register)

Lattice FPGAs support all the features of Shift Register or Shift_Register. Module/IP Block Wizard allows you to generate the Verilog-HDL for the memory size as per design requirement.

Module/IP Block Wizard generates the memory module shown in Figure 2.18.

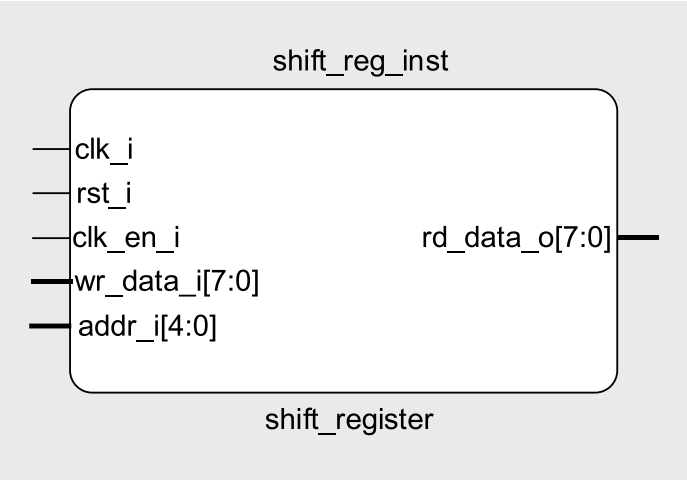

Figure 2.18. Shift Register Module Generated by Module/IP Block Wizard

The various ports and their definitions for Shift Register are listed in Table 2.11. The table lists the corresponding ports for the module generated by Module/IP Block Wizard.

Table 2.11. Shift Register Port Definitions

Port Name	Direction	Width	Description
clk_i	Input	1	Clock
rst_i	Input	1	Reset
clk_en_i	Input	1	Clock Enable
wr_data_i	Input	Data Width	Data Input
addr_i	Input	Max Shift Width	Current amount of shift from the current address. (requires shift register type be variable).
rd_data_o	Output	Data Width	Data Output

The various attributes available for the Shift Register (Shift_Register) are listed in Table 2.12. Some of these attributes are user-selectable through the Module/IP Block Wizard interface.

Table 2.12. Shift Register Attribute Definitions

Attributes	Description	Values	Default Value
Configuration Attributes			
Max Shift	The maximum address shifts allowed.	2 – 8192	16
Data Width	Data word width of the Read and Write port.	1 – 256	8
Enable Output Register	Data Out port (Q) can be registered or not using this selection.	True, False	True
Shift Register Type	Chooses whether, the shift register is operating at: (1) fixed delta from the currently written address, (2) variable delta from the currently written address	fixed, variable	fixed
Implementation*	Chooses how the shift register memory is implemented.	EBR, LUT	EBR

Note: EBR implementation requires a minimum of 30-bit total size to be properly implemented.

Timing diagrams for Shift Register are shown in Figure 2.19 and Figure 2.20 for non-registered and registered configurations, respectively. An additional timing diagram is provided for variable shift configuration without registers, see Figure 2.21. (Shift Register uses MAX_DEPTH = 4 for fixed configuration and MAX_DEPTH = 8 for variable configuration).

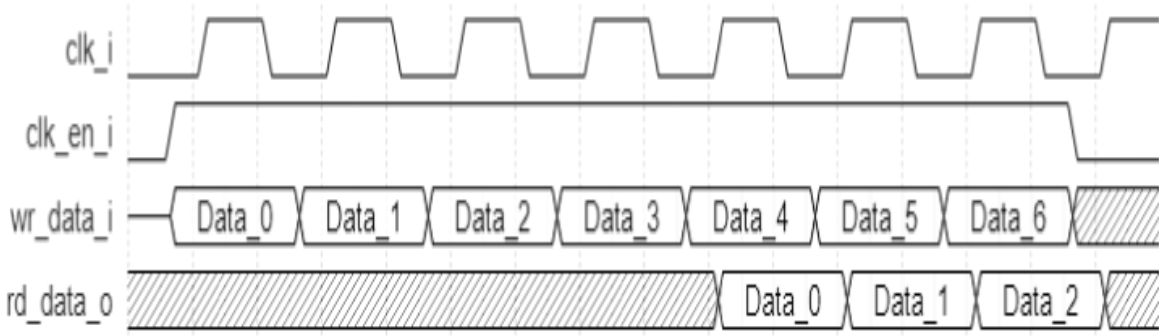

Figure 2.19. Shift Register without output register

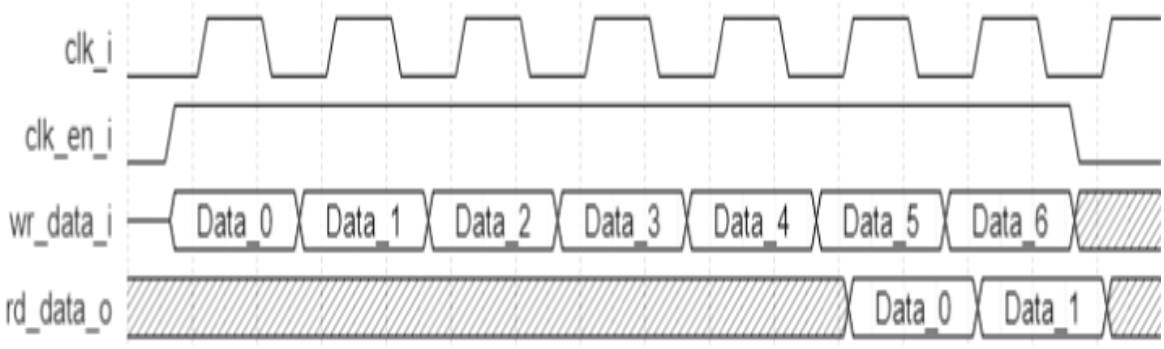

Figure 2.20. Shift Register with output register

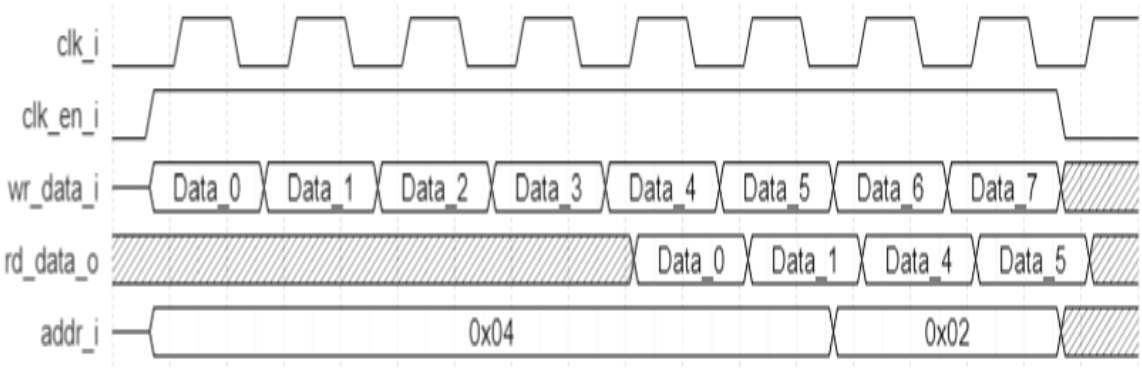

Figure 2.21. Shift Register variable configuration and without output register

3. IP Generation

Module/IP Block Wizard in Radiant Software allows you to generate, create, or open modules for the target device. From the Radiant Software, select the **IP Catalog** tab as shown in Figure 3.1.

The left pane of the IP Catalog window displays the module tree. You can utilize **Module/IP on Local** to specify a variety of memories in your designs. These modules are constructed using one or more memory primitives along with general purpose routing and LUTs (lookup tables) as required.

The memory modules are categorized as **Memory_Modules** with **EBR_Components** as sub-category. The available memory modules in the Radiant Software IP catalog are:

- FIFO
- FIFO_DC
- Shift_Register
- EBR_Components (or EBR based Modules)
  - RAM_DP (Pseudo Dual Port RAM)
  - RAM_DQ (Single Port RAM)
  - ROM

The right pane of the window shows the description of the selected module and provides links to the documentation.

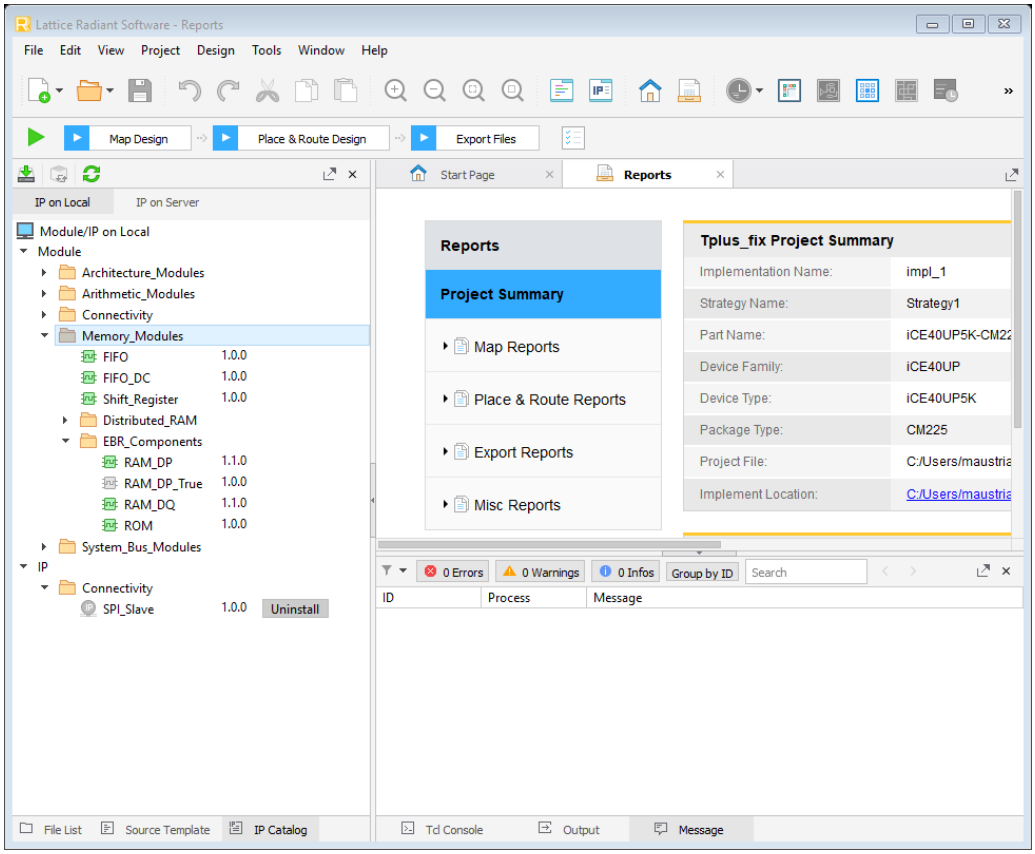

Figure 3.1. Memory Modules under Module/IP on Local in the Radiant Software

The following section shows an example of generating an EBR-based Pseudo Dual Port RAM of size 512 x 18.

To generate an EBR based Pseudo Dual Port RAM of size 512 x 18:

1. Double-click **RAM_DP** under **Memory_Modules > EBR_Components**.
2. This opens the Module/IP Block Wizard. In the **Name & Location** page, fill out the information of the module to generate as shown in Figure 3.2. Click **Next**.

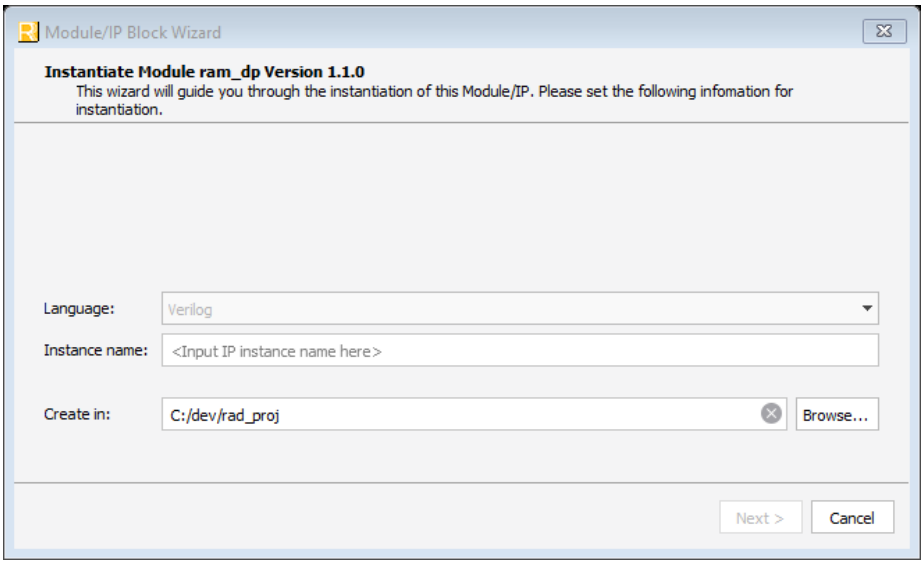

Figure 3.2. Example: Generating Pseudo Dual Port RAM (RAM_DP) Using Module/IP Block Wizard

3. In the **IP Configuration** page, customize the Dual Port RAM by selecting options as shown in Figure 3.3.

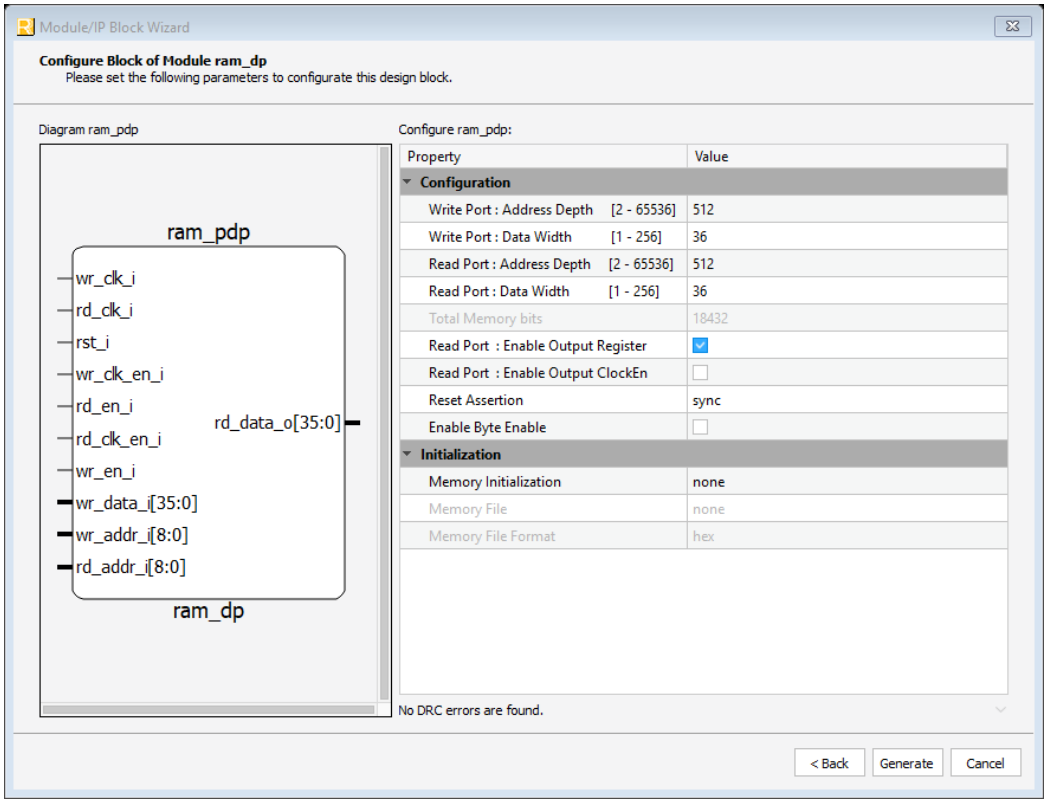

Figure 3.3. Example: Generating Pseudo Dual Port RAM (RAM_DP) Module Customization

4. When all the options are set, click **Generate**.

This module, once in the Radiant project, can be instantiated within other modules.

4. PMI Support

The following sections discuss the different PMI modules which can be utilized for entering custom parameters for the single-port and pseudo dual-port RAM. If parameters are left unspecified during module instantiation, use default values.

4.1. pmi_ram_dq

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_ram_dq (Single Port Memory Module).

Table 4.1. Single Port PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	Address	Bus	(pmi_addr_width - 1):0
I	Data	Bus	(pmi_data_width - 1):0
I	Clock	Bit	N/A
I	ClockEn	Bit	N/A
I	Reset	Bit	N/A
I	WE	Bit	N/A
O	Q	Bus	(pmi_data_width - 1):0

Table 4.2. Single Port PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_addr_depth	Address depth of the read and write port.	2 – <Max that can fit in the device>	512
pmi_addr_width	Address width of the read and write port.	1 – <Max that can fit in the device>	9
pmi_data_width	Data word width of the Read and Write port.	1 – 512	18
pmi_regmode	Data Out port (Q) can be registered or not using this selection.	reg, noreg	reg
pmi_gsr	Sets if the global set/reset is enabled.	enabled, disable	disable
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
pmi_optimization	Describes the synthesis directive if optimizing for area or speed.	speed, area	speed
pmi_init_file	When Memory file is selected, you should set this parameter to the memory file.	file_path	none
pmi_init_file_format	This option allows you to select if the memory file is formatted as Binary, Hex (refer to Initializing Memory for details on different formats).	binary, hex	binary
pmi_write_mode	Defines the write mode of the module.	normal	normal
pmi_family	This option selects the product family used. Defaults to common.	iCE40UP	common
module_type	Refers to the type of pmi module.	pmi_ram_dq	pmi_ram_dq

Verilog pmi_ram_dq Definition

```
module pmi_ram_dq
  #(parameter pmi_addr_depth = 512,
    parameter pmi_addr_width = 9
    parameter pmi_data_width = 18,
    parameter pmi_regmode = "reg",
    parameter pmi_gsr = "disable",
    parameter pmi_resetmode = "sync",
    parameter pmi_optimization = "speed",
    parameter pmi_init_file = "none",
    parameter pmi_init_file_format = "binary",
    parameter pmi_write_mode = "normal",
    parameter pmi_family = "iCE40UP",
    parameter module_type = "pmi_ram_dq")

    (input [(pmi_data_width-1):0]   Data,
     input [(pmi_addr_width-1):0] Address,
     input  Clock,
     input  ClockEn,
     input  WE,
     input  Reset,
     output [(pmi_data_width-1):0]  Q)/*synthesis syn_black_box*/;

endmodule // pmi_ram_dq
```

VHDL pmi_ram_dq Definition

```
 component pmi_ram_dq is
     generic (
       pmi_addr_depth : integer := 512;
       pmi_addr_width : integer := 9;
       pmi_data_width : integer := 18;
       pmi_regmode : string := "reg";
       pmi_gsr : string := "disable";
       pmi_resetmode : string := "sync";
       pmi_optimization : string := "speed";
       pmi_init_file : string := "none";
       pmi_init_file_format : string := "binary";
       pmi_write_mode : string := "normal";
       pmi_family : string := "iCE40UP";
       module_type : string := "pmi_ram_dq"
    );
    port (
     Data : in std_logic_vector((pmi_data_width-1) downto 0);
     Address : in std_logic_vector((pmi_addr_width-1) downto 0);
     Clock: in std_logic;
     ClockEn: in std_logic;
     WE: in std_logic;
     Reset: in std_logic;
     Q : out std_logic_vector((pmi_data_width-1) downto 0)
   );
  end component pmi_ram_dq;
```

4.2. pmi_ram_dp

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_ram_dp (Pseudo Dual Port Memory Module).

Table 4.3. Pseudo Dual Port PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	WrAddress	Bus	(pmi_wr_addr_width - 1):0
I	RdAddress	Bus	(pmi_rd_addr_width - 1):0
I	Data	Bus	(pmi_wr_data_width - 1):0
I	RdClock	Bit	N/A
I	RdClockEn	Bit	N/A
I	Reset	Bit	N/A
I	WrClock	Bit	N/A
I	WrClockEn	Bit	N/A
I	WE	Bit	N/A
O	Q	Bus	(pmi_rd_data_width - 1):0

Table 4.4. Pseudo Dual Port PMI Port Definitions

Attributes	Description	Values	Default Value
pmi_wr_addr_depth	Write Port Address depth.	2 – <Max that can fit in the device>	512
pmi_wr_addr_width	Write Port Address width. Equal to $\log_2$(pmi_wr_addr_depth).	1 – <Max that can fit in the device>	9
pmi_wr_data_width	Write Port Data word width.	1-512	18
pmi_rd_addr_depth	Read Port Address depth.	2 – <Max that can fit in the device>	512
pmi_rd_addr_width	Read Port Address width. Equal to $\log_2$(pmi_rd_addr_depth)	1 – <Max that can fit in the device>	9
pmi_rd_data_width	Read Port Data word width.	1-512	18
pmi_regmode	Data Out port (Q) can be registered or not using this selection.	reg, noreg	reg
pmi_gsr	Sets if the global set/reset is enabled.	enabled, disable	disable
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
pmi_optimization	Describes the synthesis directive if optimizing for area or speed	speed, area	speed
pmi_init_file	When Memory file is selected, you should set this parameter to the memory file.	file_path	none
pmi_init_file_format	This option allows you to select if the memory file is formatted as Binary, Hex. (refer to Initializing Memory for details on different formats)	binary, hex	binary
pmi_family	Defines the FPGA family being used in the module	iCE40 UltraPlus	common
module_type	Refers to the type of pmi module.	pmi_ram_dp	pmi_ram_dp

Verilog pmi_ram_dp Definition

```
module pmi_ram_dp
  #(parameter pmi_wr_addr_depth = 512,
    parameter pmi_wr_addr_width = 9,
    parameter pmi_wr_data_width = 18,
    parameter pmi_rd_addr_depth = 512,
    parameter pmi_rd_addr_width = 9,
    parameter pmi_rd_data_width = 18,
    parameter pmi_regmode = "reg",
    parameter pmi_gsr = "disable",
    parameter pmi_resetmode = "sync",
    parameter pmi_optimization = "speed",
    parameter pmi_init_file = "none",
    parameter pmi_init_file_format = "binary",
    parameter pmi_family = "iCE40UP",
    parameter module_type = "pmi_ram_dp")

    (input [(pmi_wr_data_width-1):0] Data,
     input [(pmi_wr_addr_width-1):0] WrAddress,
     input [(pmi_rd_addr_width-1):0] RdAddress,
     input  WrClock,
     input  RdClock,
     input  WrClockEn,
     input  RdClockEn,
     input  WE,
     input  Reset,
     output [(pmi_rd_data_width-1):0]  Q) /*synthesis syn_black_box*/;

endmodule // pmi_ram_dp
```

VHDL pmi_ram_dp Definition

```
component pmi_ram_dp is
     generic (
       pmi_wr_addr_depth : integer := 512;
       pmi_wr_addr_width : integer := 9;
       pmi_wr_data_width : integer := 18;
       pmi_rd_addr_depth : integer := 512;
       pmi_rd_addr_width : integer := 9;
       pmi_rd_data_width : integer := 18;
       pmi_regmode : string := "reg";
       pmi_gsr : string := "disable";
       pmi_resetmode : string := "sync";
       pmi_optimization : string := "speed";
       pmi_init_file : string := "none";
       pmi_init_file_format : string := "binary";
       pmi_family : string := "iCE40UP";
       module_type : string := "pmi_ram_dp"
    );
    port (
     Data : in std_logic_vector((pmi_wr_data_width-1) downto 0);
     WrAddress : in std_logic_vector((pmi_wr_addr_width-1) downto 0);
     RdAddress : in std_logic_vector((pmi_rd_addr_width-1) downto 0);
     WrClock: in std_logic;
     RdClock: in std_logic;
     WrClockEn: in std_logic;
     RdClockEn: in std_logic;
     WE: in std_logic;
     Reset: in std_logic;
     Q : out std_logic_vector((pmi_rd_data_width-1) downto 0)
   );
  end component pmi_ram_dp;
```

4.3. pmi_rom

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_rom (Read Only Memory Module).

Table 4.5. Read Only Memory PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	WrAddress	Bus	(pmi_wr_addr_width - 1):0
I	RdAddress	Bus	(pmi_rd_addr_width - 1):0
I	Data	Bus	(pmi_wr_data_width - 1):0
I	RdClock	Bit	N/A
I	RdClockEn	Bit	N/A

Table 4.6. Read Only Memory PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_addr_depth*	Address depth.	2 – <Max that can fit in the device>	512
pmi_addr_width*	Address width. Equal to $\log_2$(pmi_wr_addr_depth).	1 – <Max that can fit in the device>	9
pmi_data_width	Data word width.	1-512	18
pmi_regmode	Data Out port (Q) can be registered or not using this selection.	reg, noreg	reg
pmi_gsr	Sets if the global set/reset is enabled.	enabled, disable	disable
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
pmi_optimization	Describes the synthesis directive if optimizing for area or speed.	speed, area	speed
pmi_init_file	When Memory file is selected, you should set this parameter to the memory file.	file_path	none
pmi_init_file_format	This option allows you to select if the memory file is formatted as Binary, Hex (refer to Initializing Memory for details on different formats).	binary, hex	binary
pmi_family	Defines the FPGA family being used in the module.	iCE40UP	common
module_type	Refers to the type of pmi module.	pmi_rom	pmi_rom

***Note:** This IP requires a minimum of 30-bit total size to be properly implemented.

Verilog pmi_rom Definition

```
module pmi_rom # (
    parameter pmi_addr_depth = 512,
    parameter pmi_addr_width = 9
    parameter pmi_data_width = 8,
    parameter pmi_regmode = "reg",
    parameter pmi_gsr = "disable",
    parameter pmi_resetmode = "sync",
    parameter pmi_optimization = "speed",
    parameter pmi_init_file = "none",
    parameter pmi_init_file_format = "binary",
    parameter pmi_family = "iCE40UP",
    parameter module_type = "pmi_rom"
)(input [(pmi_addr_width-1):0]      Address,
     input OutClock,
     input OutClockEn,
     input Reset,
     output [(pmi_data_width-1):0] Q)/*synthesis syn_black_box*/)
;

endmodule // pmi_rom
```

VHDL pmi_rom Definition

```
component pmi_rom is
     generic (
       pmi_addr_depth : integer := 512;
       pmi_addr_width : integer := 9;
       pmi_data_width : integer := 8;
       pmi_regmode : string := "reg";
       pmi_gsr : string := "disable";
       pmi_resetmode : string := "sync";
       pmi_optimization : string := "speed";
       pmi_init_file : string := "none";
       pmi_init_file_format : string := "binary";
       pmi_family : string := "iCE40UP";
       module_type : string := "pmi_rom"
    );
    port (
     Address : in std_logic_vector((pmi_addr_width-1) downto 0);
     OutClock: in std_logic;
     OutClockEn: in std_logic;
     Reset: in std_logic;
     Q : out std_logic_vector((pmi_data_width-1) downto 0)
   );
  end component pmi_rom;
```

4.4. pmi_ram_dp_be

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_ram_dp_be (Pseudo Dual Port Memory Module with Byte Enable).

Table 4.7. Pseudo Dual Port with Byte-Enable – Port Definitions

Direction	Port Name	Type	Size (buses only)
I	WrAddress	Bus	(pmi_wr_addr_width - 1):0
I	RdAddress	Bus	(pmi_rd_addr_width - 1):0
I	Data	Bus	(pmi_wr_data_width - 1):0
I	RdClock	Bit	N/A
I	RdClockEn	Bit	N/A
I	Reset	Bit	N/A
I	WrClock	Bit	N/A
I	WrClockEn	Bit	N/A
I	WE	Bit	N/A
I	ByteEn	Bus	(pmi_wr_data_width/pmi_byte_size)-1:0
O	Q	Bus	(pmi_rd_data_width - 1):0

Table 4.8. Pseudo Dual Port with Byte-Enable – PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_wr_addr_depth	Write Port Address depth.	2 – <Max that can fit in the device>	512
pmi_wr_addr_width	Write Port Address width. Equal to $\log_2$(pmi_wr_addr_depth).	1 – <Max that can fit in the device>	9
pmi_wr_data_width	Write Port Data word width.	1-512	16
pmi_rd_addr_depth	Read Port Address depth.	2 – <Max that can fit in the device>	512
pmi_rd_addr_width	Read Port Address width. Equal to $\log_2$(pmi_rd_addr_depth).	1 – <Max that can fit in the device>	9
pmi_rd_data_width	Read Port Data word width.	1-512	16
pmi_regmode	Data Out port (Q) can be registered or not using this selection.	reg, noreg	reg
pmi_gsr	Sets if the global set/reset is enabled.	enabled, disable	disable
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
pmi_optimization	Describes the synthesis directive if optimizing for area or speed.	speed, area	speed
pmi_init_file	When Memory file is selected, you should set this parameter to the memory file.	file_path	none
pmi_init_file_format	This option allows you to select if the memory file is formatted as Binary, Hex (refer to Initializing Memory for details on different formats).	binary, hex	binary
pmi_family	Defines the FPGA family being used in the module.	iCE40UP	common
pmi_byte_size	Defines the number of bits which determines a byte.	8	8

Attributes	Description	Values	Default Value
module_type	Refers to the type of pmi module.	pmi_ram_dp	pmi_ram_dp

Verilog pmi_ram_dp_be Definition

```
module pmi_ram_dp_be #
(   parameter pmi_wr_addr_depth = 512,
    parameter pmi_wr_addr_width = 9,
    parameter pmi_wr_data_width = 18,
    parameter pmi_rd_addr_depth = 512,
    parameter pmi_rd_addr_width = 9,
    parameter pmi_rd_data_width = 18,
    parameter pmi_regmode = "reg",
    parameter pmi_gsr = "disable",
    parameter pmi_resetmode = "sync",
    parameter pmi_optimization = "speed",
    parameter pmi_init_file = "none",
    parameter pmi_init_file_format = "binary",
    parameter pmi_family = "common",
    parameter pmi_byte_size = 8,
    parameter module_type = "pmi_ram_dp_be"
) (
    input [(pmi_wr_data_width-1):0] Data,
    input [(pmi_wr_addr_width-1):0] WrAddress,
    input [(pmi_rd_addr_width-1):0] RdAddress,
    input  WrClock,
    input  RdClock,
    input  WrClockEn,
    input  RdClockEn,
    input  WE,
    input  Reset,
    input [(pmi_wr_data_width/pmi_byte_size)-1:0] ByteEn,
    output [(pmi_rd_data_width-1):0]  Q
);// pmi_ram_dp_be
```

VHDL pmi_ram_dp_be Definition

```
  component pmi_ram_dp_be is
     generic (
       pmi_wr_addr_depth : integer := 512;
       pmi_wr_addr_width : integer := 9;
       pmi_wr_data_width : integer := 16;
       pmi_rd_addr_depth : integer := 512;
       pmi_rd_addr_width : integer := 9;
       pmi_rd_data_width : integer := 16;
       pmi_regmode : string := "reg";
       pmi_gsr : string := "disable";
       pmi_resetmode : string := "sync";
       pmi_optimization : string := "speed";
       pmi_init_file : string := "none";
       pmi_init_file_format : string := "binary";
       pmi_byte_size : integer := 8;
       pmi_family : string := "iCE40UP";
       module_type : string := "pmi_ram_dp_be"
    );
    port (
     Data : in std_logic_vector((pmi_wr_data_width-1) downto 0);
     WrAddress : in std_logic_vector((pmi_wr_addr_width-1) downto 0);
     RdAddress : in std_logic_vector((pmi_rd_addr_width-1) downto 0);
     WrClock: in std_logic;
     RdClock: in std_logic;
     WrClockEn: in std_logic;
     RdClockEn: in std_logic;
     WE: in std_logic;
     Reset: in std_logic;
     ByteEn : in std_logic_vector(((pmi_wr_data_width+pmi_byte_size-
1)/pmi_byte_size-1) downto 0);
     Q : out std_logic_vector((pmi_rd_data_width-1) downto 0)
   );
  end component pmi_ram_dp_be;
```

4.5. pmi_ram_dq_be

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_ram_dq_be (Single Port Memory Module with Byte Enable).

Table 4.9. Single Port with Byte-Enable – PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	Address	Bus	(pmi_addr_width - 1):0
I	Data	Bus	(pmi_data_width - 1):0
I	Clock	Bit	N/A
I	ClockEn	Bit	N/A
I	Reset	Bit	N/A
I	WE	Bit	N/A
I	ByteEn	Bus	(pmi_ data_width/pmi_byte_size)-1:0
O	Q	Bus	(pmi_data_width - 1):0

Table 4.10. Single Port with Byte-Enable – PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_addr_depth	Address depth of the read and write port.	2 – <Max that can fit in the device>	512
pmi_addr_width	Address width of the read and write port.	1 – <Max that can fit in the device>	9
pmi_data_width	Data word width of the Read and Write port.	1 – 512	16
pmi_regmode	Data Out port (Q) can be registered or not using this selection.	reg, noreg	reg
pmi_gsr	Sets if the global set/reset is enabled.	enabled, disable	disable
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	sync
pmi_optimization	Describes the synthesis directive if optimizing for area or speed.	speed, area	speed
pmi_init_file	When Memory file is selected, you should set this parameter to the memory file.	—	—
pmi_init_file_format	This option allows you to select if the memory file is formatted as Binary, Hex (refer to Initializing Memory for details on different formats).	binary, hex	binary
pmi_write_mode	Defines the write mode of the module.	normal	normal
pmi_family	This option selects the product family used. Defaults to common.	iCE40UP	common
pmi_byte_size	Defines the number of bits which determines a byte.	8	8
module_type	Refers to the type of pmi module.	pmi_ram_dq_be	pmi_ram_dq_be

Verilog `pmi_ram_dq_be` Definition

```
module pmi_ram_dq_be # (
    parameter     pmi_addr_depth       = 512,
    parameter     pmi_addr_width       = 9,
    parameter     pmi_data_width       = 16,
    parameter     pmi_regmode          = "reg",
    parameter     pmi_gsr              = "disable",
    parameter     pmi_resetmode        = "sync",
    parameter     pmi_optimization     = "speed",
    parameter     pmi_init_file        = "none",
    parameter     pmi_init_file_format = "binary",
    parameter     pmi_write_mode       = "normal",
    parameter     pmi_family           = "common",
    parameter     pmi_byte_size        = 8,
    parameter     module_type          = "pmi_ram_dq_be"
)
(   input [(pmi_data_width-1):0]    Data,
    input [(pmi_addr_width-1):0] Address,
    input  Clock,
    input  ClockEn,
    input  WE,
    input  Reset,
    input [(pmi_data_width/pmi_byte_size-1):0] ByteEn,
    output [(pmi_data_width-1):0]  Q
 );
endmodule // pmi_ram_dq_be
```

VHDL pmi_ram_dq_be Definition

```
  component pmi_ram_dq_be is
     generic (
       pmi_addr_depth : integer := 512;
       pmi_addr_width : integer := 9;
       pmi_data_width : integer := 16;
       pmi_regmode : string := "reg";
       pmi_gsr : string := "disable";
       pmi_resetmode : string := "sync";
       pmi_optimization : string := "speed";
       pmi_init_file : string := "none";
       pmi_init_file_format : string := "binary";
       pmi_write_mode : string := "normal";
       pmi_byte_size : integer := 8;
       pmi_family : string := "common";
       module_type : string := "pmi_ram_dq_be"
    );
    port (
     Data : in std_logic_vector((pmi_data_width-1) downto 0);
     Address : in std_logic_vector((pmi_addr_width-1) downto 0);
     Clock: in std_logic;
     ClockEn: in std_logic;
     WE: in std_logic;
     Reset: in std_logic;
     ByteEn : in std_logic_vector(((pmi_data_width+pmi_byte_size-1)/pmi_byte_size-
1) downto 0);
     Q : out std logic vector((pmi data width-1) downto 0)
   );
  end component pmi_ram_dq_be;
```

4.6. pmi_fifo

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_fifo (First In First Out Memory Module).

Table 4.11. FIFO Single Clock PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	Data	Bus	(pmi_data_width - 1):0
I	Clock	Bit	N/A
I	WrEn	Bit	N/A
I	RdEn	Bit	N/A
I	Reset	Bit	N/A
O	Q	Bus	(pmi_data_width - 1):0
O	Empty	Bit	N/A
O	Full	Bit	N/A
O	AlmostEmpty	Bit	N/A
O	AlmostFull	Bit	N/A

Table 4.12. FIFO Single Clock PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_data_width	FIFO data width.	2 – <Max that can fit in the device>	8
pmi_data_depth	FIFO address depth.	1 – <Max that can fit in the device>	256
pmi_ full_flag	Defines the upper limit of the almost full flag, and trigger limit of the full flag.	2 – <Max that can fit in the device>	256
pmi_ empty_flag	Defines the lower limit of the almost empty flag, and trigger limit of the empty flag.	1 – <Max that can fit in the device>	0
pmi_almost_full_flag	Defines the lower limit of the almost full flag.	2 – <Max that can fit in the device>	252
pmi_almost_empty_flag	Defines the upper limit of the almost empty flag.	1 – <Max that can fit in the device>	4
pmi_regmode	Data Out for FIFO that can be registered or not using this selection.	reg, noreg	reg
pmi_family	Defines the FPGA family being used in the module.	iCE40UP	common
module_type	Refers to the type of pmi module.	pmi_fifo	pmi_fifo
pmi_implementation*	Refers to the memory used by the FIFO module.	EBR, LUT	EBR

***Note:** EBR implementation requires a minimum of 30-bit total size to be properly implemented.

Verilog pmi_fifo Definition

```
module pmi_fifo #(
      parameter pmi_data_width = 8,
      parameter pmi_data_depth = 256,
      parameter pmi_full_flag = 256,
      parameter pmi_empty_flag = 0,
      parameter pmi_almost_full_flag = 252,
      parameter pmi_almost_empty_flag = 4,
      parameter pmi_regmode = "reg",
      parameter pmi_family = "iCE40UP",
      parameter pmi_implementation = "EBR",
      parameter module_type = "pmi_fifo"
)(
   input  [pmi_data_width-1:0] Data,
   input Clock,
   input WrEn,
   input RdEn,
   input Reset,
   output [pmi_data_width-1:0] Q,
   output Empty,
   output Full,
   output AlmostEmpty,
   output AlmostFull)/*synthesis syn_black_box */;

endmodule // pmi_fifo
```

VHDL pmi_ram_dp Definition

```
component pmi_fifo is
     generic (
       pmi_data_width : integer := 8;
       pmi_data_depth : integer := 256;
       pmi_full_flag : integer := 256;
       pmi_empty_flag : integer := 0;
       pmi_almost_full_flag : integer := 252;
       pmi_almost_empty_flag : integer := 4;
       pmi_regmode : string := "reg";
       pmi_family : string := "iCE40UP" ;
       pmi_implementation : string := "EBR";
       module_type : string := "pmi_fifo"
    );
    port (
     Data : in std_logic_vector(pmi_data_width-1 downto 0);
     Clock: in std_logic;
     WrEn: in std_logic;
     RdEn: in std_logic;
     Reset: in std_logic;
     Q : out std_logic_vector(pmi_data_width-1 downto 0);
     Empty: out std_logic;
     Full: out std_logic;
     AlmostEmpty: out std_logic;
     AlmostFull: out std_logic
   );
  end component pmi_fifo;
```

4.7. pmi_fifo_dc

The following are port descriptions, Verilog and VHDL definitions, and parameter descriptions for pmi_fifo_dc (First in First Out Dual Clock Memory Module).

Table 4.13. FIFO Dual Clock PMI Port Definitions

Direction	Port Name	Type	Size (buses only)
I	Data	Bus	(pmi_data_width_w - 1):0
I	WrClock	Bit	N/A
I	RdClock	Bit	N/A
I	WrEn	Bit	N/A
I	RdEn	Bit	N/A
I	Reset	Bit	N/A
I	RPReset	Bit	N/A
O	Q	Bus	(pmi_ data_width_r - 1):0
O	Empty	Bit	N/A
O	Full	Bit	N/A
O	AlmostEmpty	Bit	N/A
O	AlmostFull	Bit	N/A

Table 4.14. FIFO Dual Clock Port PMI Attribute Definitions

Attributes	Description	Values	Default Value
pmi_data_width_w	FIFO data write port size.	2 – <Max that can fit in the device>	18
pmi_data_depth_w	FIFO address depth write port size.	1 – <Max that can fit in the device>	256
pmi_data_width_r	FIFO data read port size.	2 – <Max that can fit in the device>	18
pmi_data_depth_r	FIFO address depth read port size.	1 – <Max that can fit in the device>	256
pmi_regmode	Data Out for FIFO, if can be registered or not using this selection.	reg, noreg	reg
pmi_resetmode	Selection for the Reset to be Synchronous or Asynchronous to the Clock.	async, sync	async
pmi_family	Defines the FPGA family being used in the module.	iCE40UP	common
module_type	Refers to the type of pmi module.	pmi_fifo_dc	pmi_fifo_dc
pmi_implementation	Refers to the memory used by the FIFO module.	EBR, LUT	EBR

Verilog pmi_fifo_dc Definition

```
module pmi_fifo_dc #(
     parameter pmi_data_width_w = 18,
     parameter pmi_data_width_r = 18,
     parameter pmi_data_depth_w = 256,
     parameter pmi_data_depth_r = 256,
     parameter pmi_full_flag = 256,
     parameter pmi_empty_flag = 0,
     parameter pmi_almost_full_flag = 252,
     parameter pmi_almost_empty_flag = 4,
     parameter pmi_regmode = "reg",
     parameter pmi_resetmode = "async",
     parameter pmi_family = "iCE40UP" ,
     parameter pmi_implementation = "EBR",
     parameter module_type = "pmi_fifo_dc"
     ) (
   input  [pmi_data_width_w-1:0] Data,
   input WrClock,
   input RdClock,
   input WrEn,
   input RdEn,
   input Reset,
   input RPReset,
   output [pmi_data_width_r-1:0] Q,
   output Empty,
   output Full,
   output AlmostEmpty,
   output AlmostFull)/*synthesis syn_black_box */;
endmodule // pmi_fifo_dc
```

VHDL pmi_fifo_dc Definition

```
component pmi_fifo_dc is
     generic (
       pmi_data_width_w : integer := 18;
       pmi_data_width_r : integer := 18;
       pmi_data_depth_w : integer := 256;
       pmi_data_depth_r : integer := 256;
       pmi_full_flag : integer := 256;
       pmi_empty_flag : integer := 0;
       pmi_almost_full_flag : integer := 252;
       pmi_almost_empty_flag : integer := 4;
       pmi_regmode : string := "reg";
       pmi_resetmode : string := "async";
       pmi_family : string := "iCE40UP" ;
       module_type : string := "pmi_fifo_dc";
       pmi_implementation : string := "EBR"

    );
    port (
     Data : in std_logic_vector(pmi_data_width_w-1 downto 0);
     WrClock: in std_logic;
     RdClock: in std_logic;
     WrEn: in std_logic;
     RdEn: in std_logic;
     Reset: in std_logic;
     RPReset: in std_logic;
     Q : out std_logic_vector(pmi_data_width_r-1 downto 0);
     Empty: out std_logic;
     Full: out std_logic;
     AlmostEmpty: out std_logic;
     AlmostFull: out std_logic
   );
  end component pmi_fifo_dc;;
```

5. Initializing Memory

In each memory mode, it is possible to specify the power-on state of each bit in the memory array. This allows the memory to be used as ROM if desired. Each bit in the memory array can have a value of 0 or 1.

5.1. Initialization File Formats

The initialization file is an ASCII file, which you can create or edit using any ASCII editor. Module/IP Block Wizard supports the binary and hex memory file formats.

The file name for the memory initialization file is *.mem (<file_name>.mem). Each row includes the value to be stored in a particular memory location. The number of characters (or the number of columns) represents the number of bits for each address (or the width of the memory module).

The memory initialization can be static or dynamic. In case of static initialization, the memory values are stored in the bitstream. Dynamic initialization of memories involves memory values stored in the external flash and can be updated by user-logic knowing the EBR address locations. The size of the bitstream (bit or rbt file) is larger due to static values stored in them.

The initialization file is primarily used for configuring the ROMs. RAMs can also use the initialization file to preload memory contents.

5.2. Binary File

The binary file is a text file of 0s and 1s. The rows indicate the number of words and the columns indicate the width of the memory.

Memory Size 20x32

```
00100000010000000010000001000000
00000001000000010000000100000001
00000010000000100000001000000010
00000011000000110000001100000011
00000100000001000000010000000100
00000101000001010000010100000101
00000110000001100000011000000110
00000111000001110000011100000111
00001000010010000000100001001000
00001001010010010000100101001001
00001010010010100000101001001010
00001011010010110000101101001011
00001100000011000000110000001100
00001101001011010000110100101101
00001110001111100000111000111110
00001111001111110000111100111111
00010000000100000001000000010000
00010001000100010001000100010001
00010010000100100001001000010010
00010011000100110001001100010011
```

5.3. Hex File

The hex file is a text file of hexadecimal characters in a similar row-column arrangement. The number of rows in the file is the same as the number of address locations, with each row indicating the content of the memory location.

Memory Size 8x16

```
A001
0B03
1004
CE06
0007
040A
0017
02A4
```

6. Simulation of Memory Modules

Lattice Radiant Software also allows you to simulate their configured memory modules, so the designer can observe the conditions of the output with a given stimuli. This section details the individual steps needed to execute the testbench in Active-HDL.

Table 6.1 provides a summary description of the files used in the project.

Table 6.1. File List

File	Sim	Synthesis	Description
<IP_name>.v	Yes	Yes	Top Level RTL file with the selected configuration. The main IP file
dut_params	Yes	—	Top level parameters of the generated RTL file
dut_inst	Yes	—	Instantiated version of the <IP_name>.v file for simulation use
tb_top.v	Yes	—	Test bench template, this can be edited by the user to match their specific needs.
<IP_name>.cfg	—	—	This file contains the configuration options used to recreate or modify the core in the IP Platform.
<IP_name>.ipx	—	—	The IPX file holds references to all of the elements of an IP or Module after it is generated from the IP Platform GUI. The file is used to bring in the appropriate files during the design implementation and analysis. It is also used to re-load parameter settings into the IP Platform when the IP/Module is being re-generated.

To execute the testbench in Active HDL:

1. Create a new IP instance (refer to IP Generation section on how to generate an IP in a project). For this example, generate a Pseudo Dual-Port Memory on default settings, with an IP name of **ram_dist**. (see Figure 6.1 for more information regarding the IP).

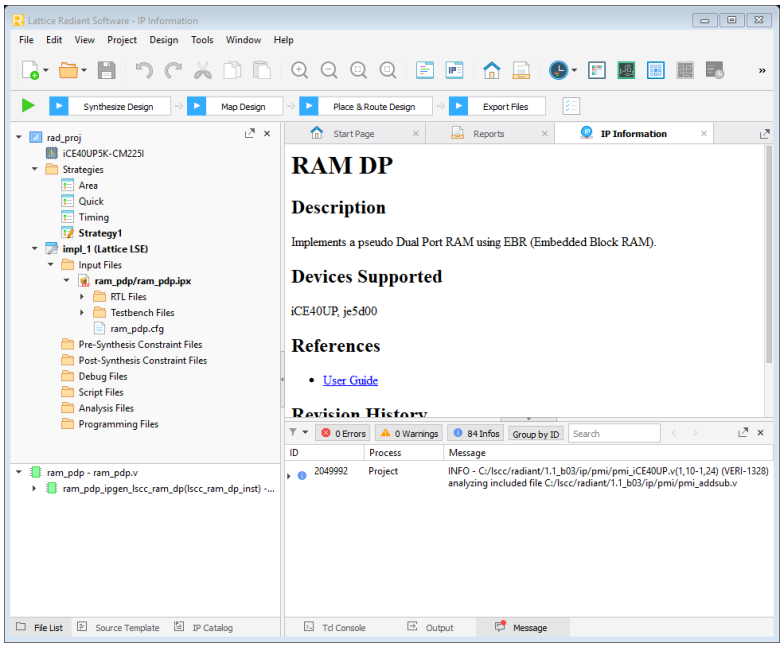

Figure 6.1. Project with an Instance of Pseudo Dual Port Memory

2. Right-click on the **impl_1**, (under **Strategies** folder) and click **Add Existing File**. The file selection window opens showing the folder where the project is saved.

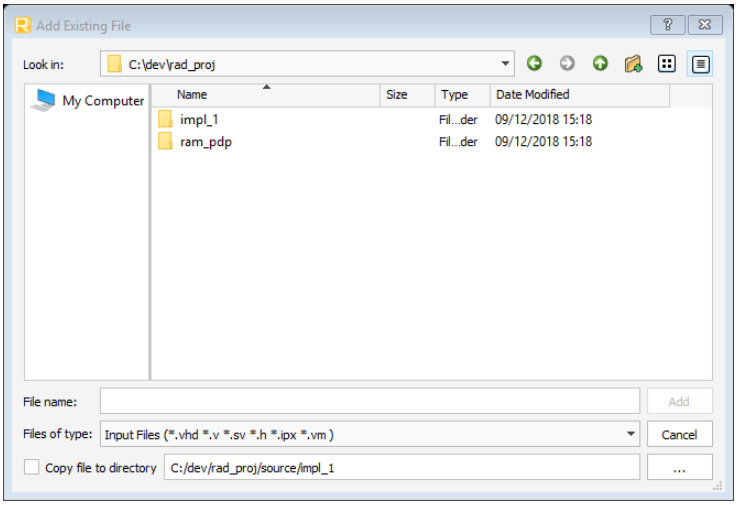

Figure 6.2. File Selection Window

3. Click the name of the IP instance, in this case **ram_dist**, and then navigate to **<IP name>/testbench**. Click on the **tb_top.v** file, then click **Add**. This adds the top level testbench to the project. After opening, it should look something like the image shown in Figure 6.3.

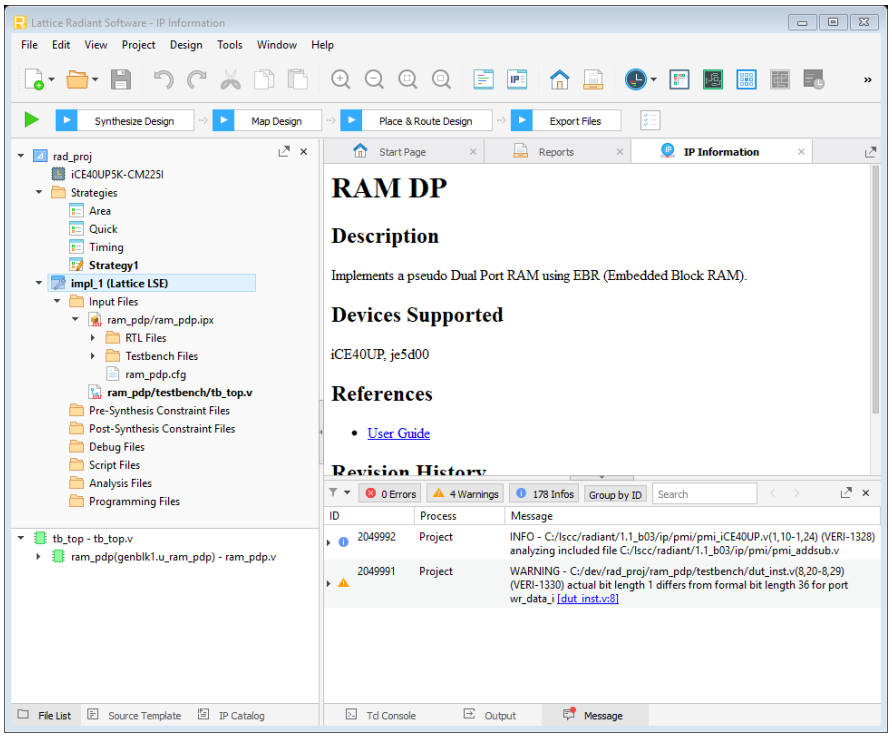

Figure 6.3. Top Level Testbench Instance Added in the Project.

4. You can modify the contents of the testbench if you want to add a specific stimuli or check a specific option. For this example, however, leave it on the default RTL file. Click **Tools** to stimulate the project. Open **Simulation Wizard** and then click **Next. S**elect the working folder and test bench name. Click **Next**.

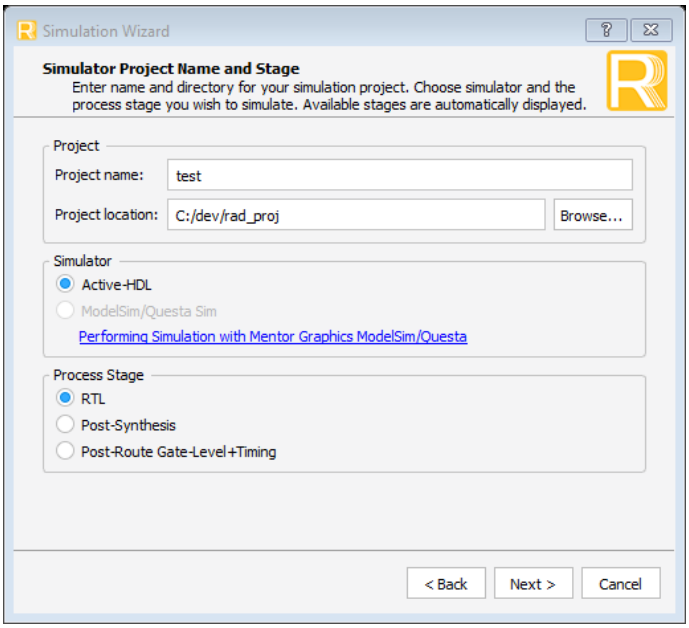

Figure 6.4. Simulation Wizard

5. The source file list appears which gives an overview of the list of files included for simulation. Click **Next**.

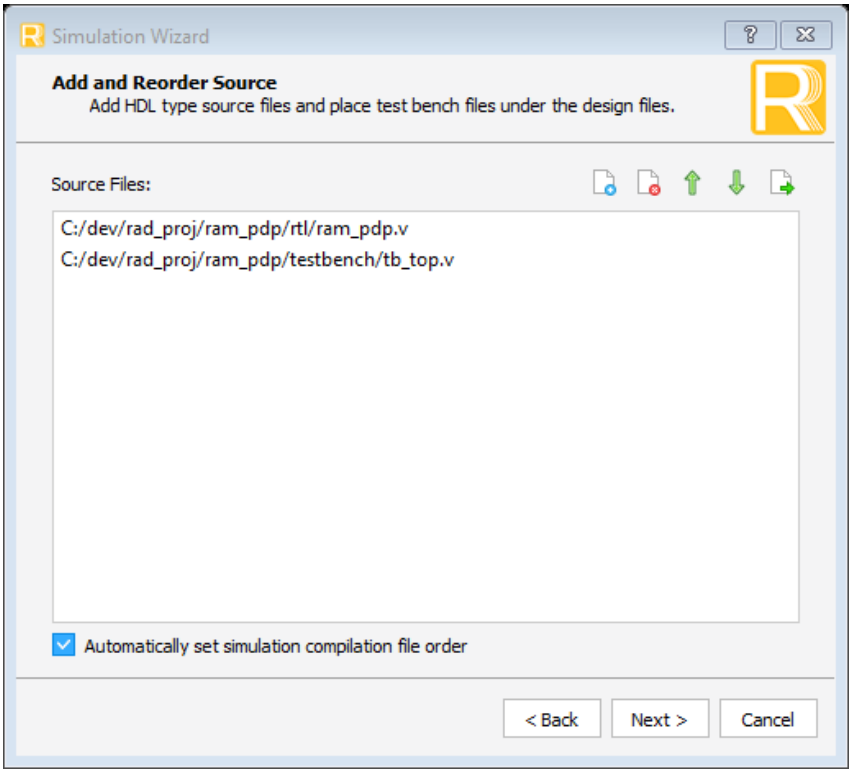

Figure 6.5. Final Simulation Files

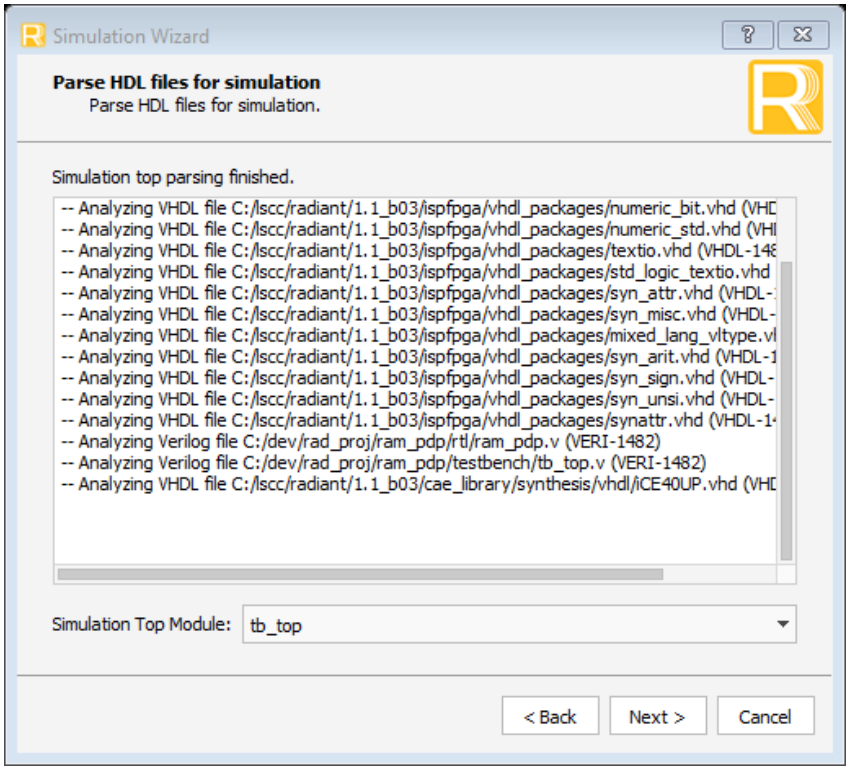

Figure 6.6. File Parsing and Top Module Selection

6. The files, as well as the other dependencies, are parsed to check for final errors. You can also select the top simulation module, before passing the files to Active-HDL for execution. Click **Next**, for a summary view and last minute options, then click **Finish**. This opens the Active-HDL Lattice Edition software, which automatically compiles and runs the RTL simulation.

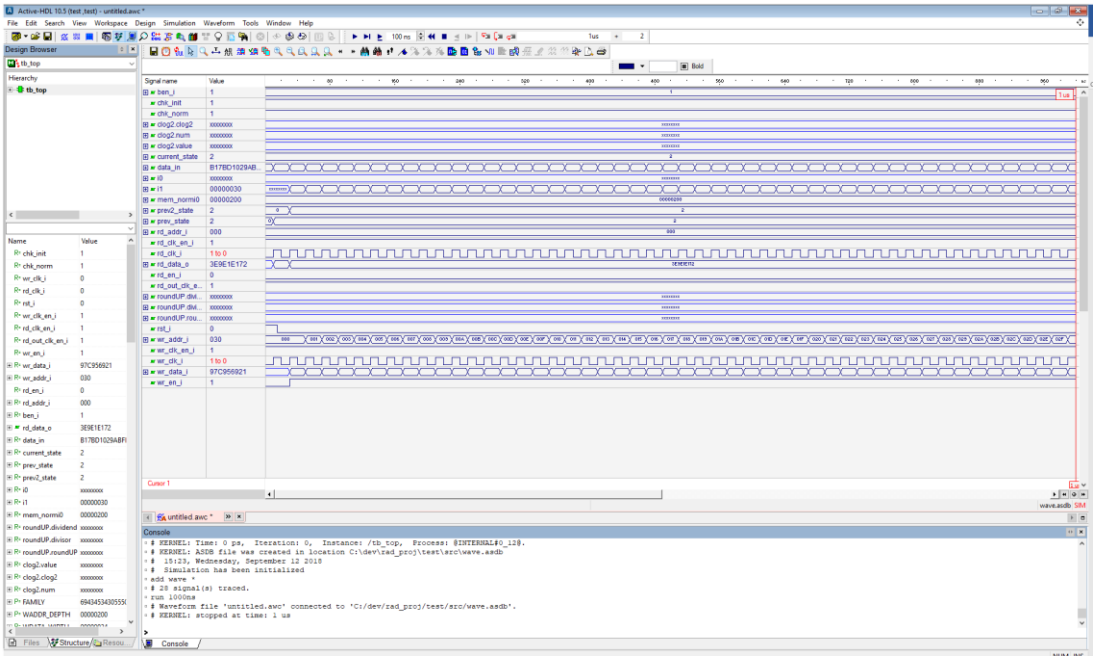

Figure 6.7. Active-HDL initial simulation

7. When the Active-HDL opens, it compiles the included RTL files and runs the simulation up to the first 100 ns. To view the rest of the simulation, click **Play**. Zoom out for the overview of the signals.

   Once simulation is completed, the console displays a Simulation Passed message. You can also check the individual sections of the signal by zooming in and out of the window.

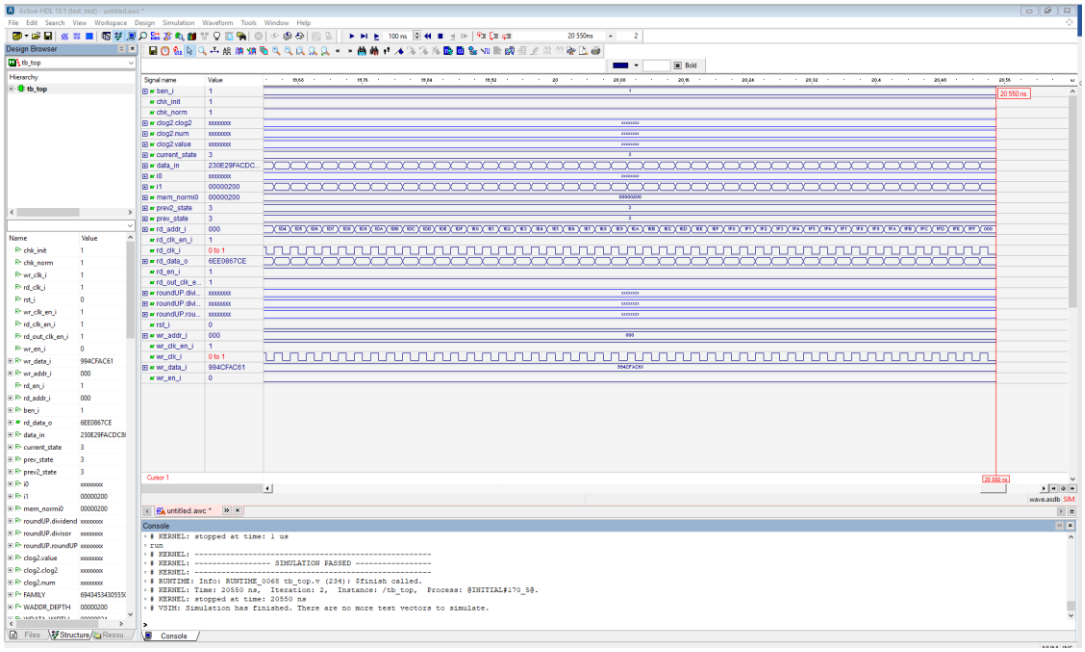

Figure 6.8. Active-HDL final simulation

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.1, January 2019

Section	Change Summary
All	Initial release.
Memory Modules	Added the following sections: • Read Only Memory (ROM) – EBR Based • First In First Out Single Clock (FIFO) • First In First Out Dual Clock (FIFO_DC) • Shift Register (Shift_Register)
PMI Support	Added the following sections: • pmi_rom • pmi_ram_dp_be • pmi_ram_dq_be • pmi_fifo • pmi_fifo_dc
Simulation of Memory Modules	Added this section in the document.
Revision History	Updated revision history table to new template.

Revision 1.0, January 2018

Section	Change Summary
All	Initial release.

7th Floor, 111 SW 5th Avenue
Portland, OR 97204, USA
T 503.268.8000
www.latticesemi.com