# Memory Modules

# User Guide

FPGA-IPUG-02033-1.0

February 2018

# Contents

# Figures



# Tables

# 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 general user interface (GUI) 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 GUI 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 will have the instantiation of a synthesizable RTL code which will be 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 the features of 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.

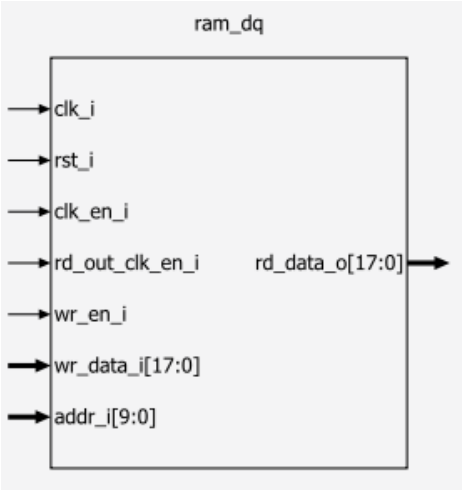


**Figure 2.1. Single-Port Memory Module Generated by Module/IP Block Wizard**

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 | (Input) 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 |
| 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, users working with the direct primitive instantiation can access these options.

**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 – 32768 | 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 |
| **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 | binary |

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

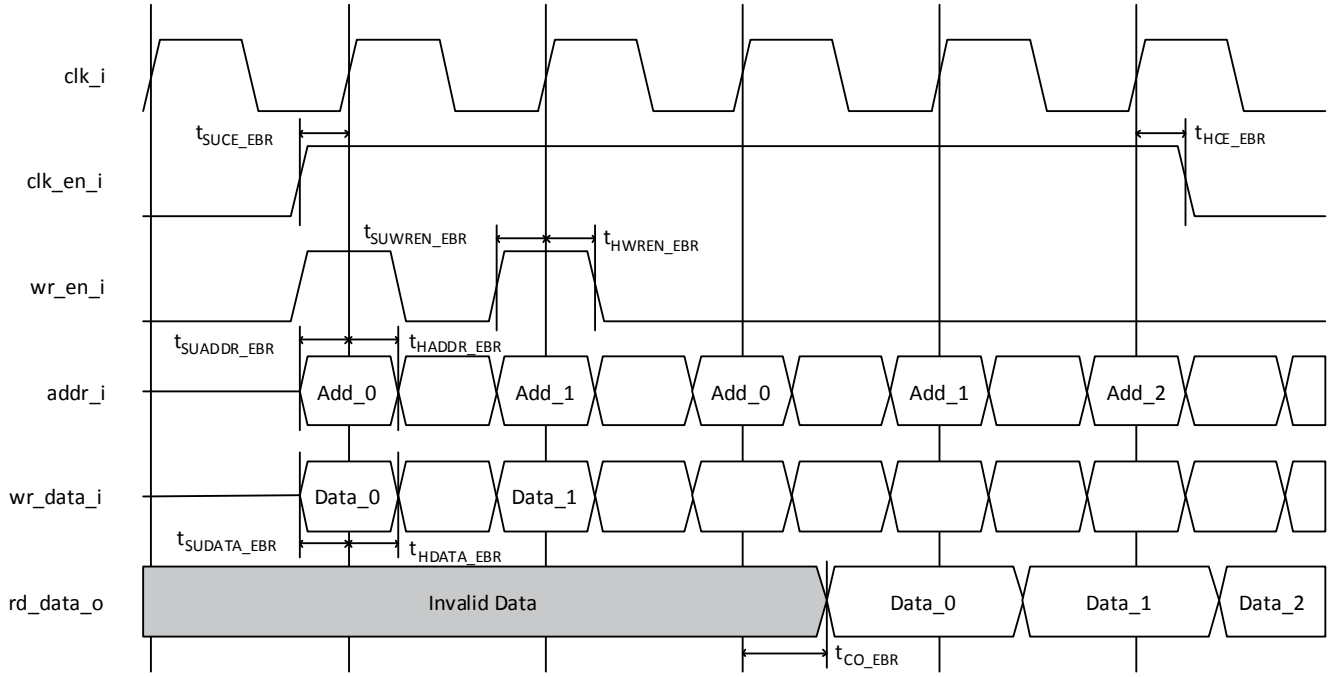


**Figure 2.2. Single Port RAM Timing Waveform in Normal (NORMAL) Mode, without Output Registers**

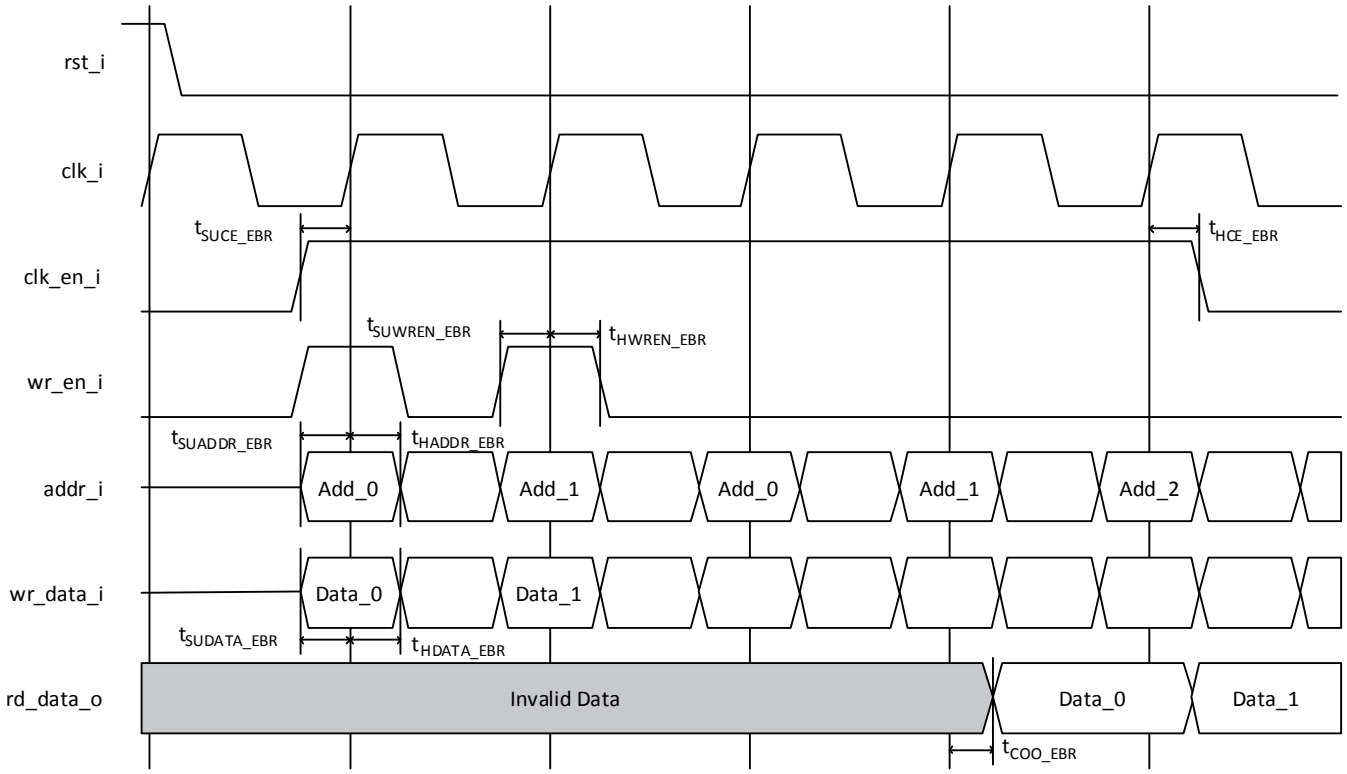


**Figure 2.3. Single Port RAM Timing Waveform in Normal (NORMAL) Mode, with Output Registers**

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

Lattice FPGAs support all the features of 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.

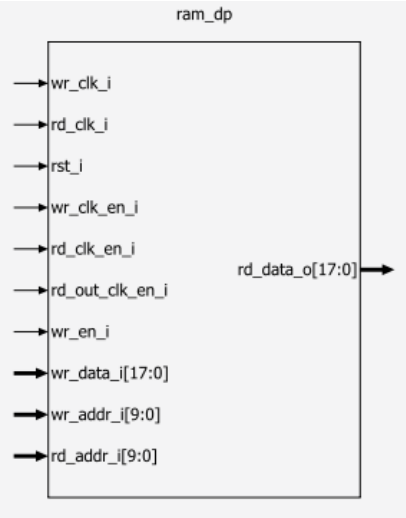


**Figure 2.4. Pseudo Dual-Port Memory Module Generated by Module/IP Block Wizard**

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 |
| rd_data_o | Output | Read Port Data Width | Read Data |

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**

| Attributed | Description | Values | Default Value |
|---|---|---|---|
| **General Attributes** | | | |
| Write Port Address Depth | Write Port Address depth | 2 – 32768 | 512 |
| Write Port Data Width | Write Port Data word width | 1 – 512 | 36 |
| Read Port Address Depth | Read Port Address depth (See note [1]) | 2 – 32768 | 512 |
| Read Port Data Width | Read Port Data word width. (See note [2]) | 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 the Reset to be Synchronous or Asynchronous to the Clock | async, sync | sync |
| **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 | binary |

[1] Read Port Address Depth != Write Port Address Depth is not supported in this release.
[2] Read Port Address Width != Write Port Address Width is not supported in this release.

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.

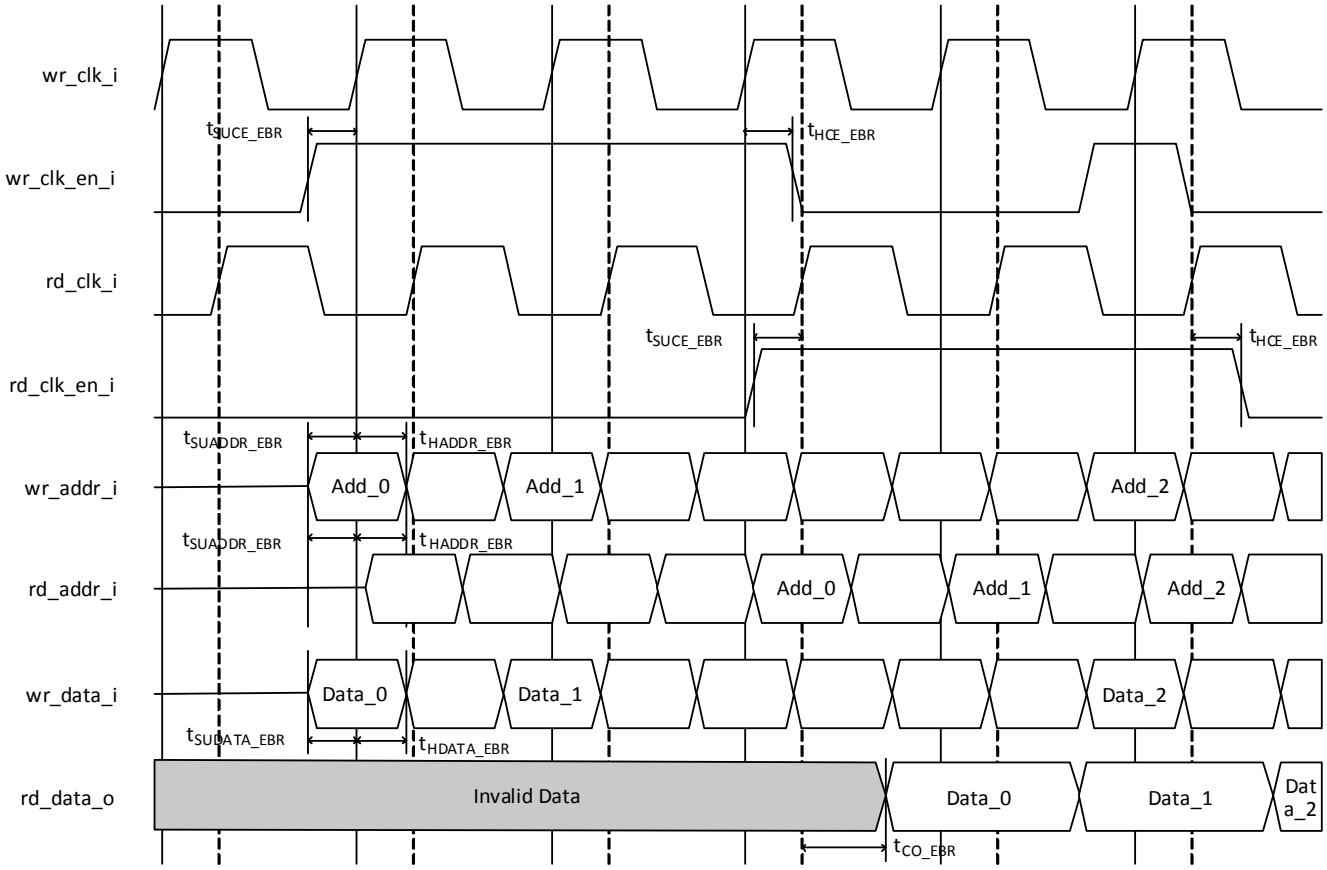


**Figure 2.5. PSEUDO DUAL PORT RAM Timing Diagram, without Output Registers**

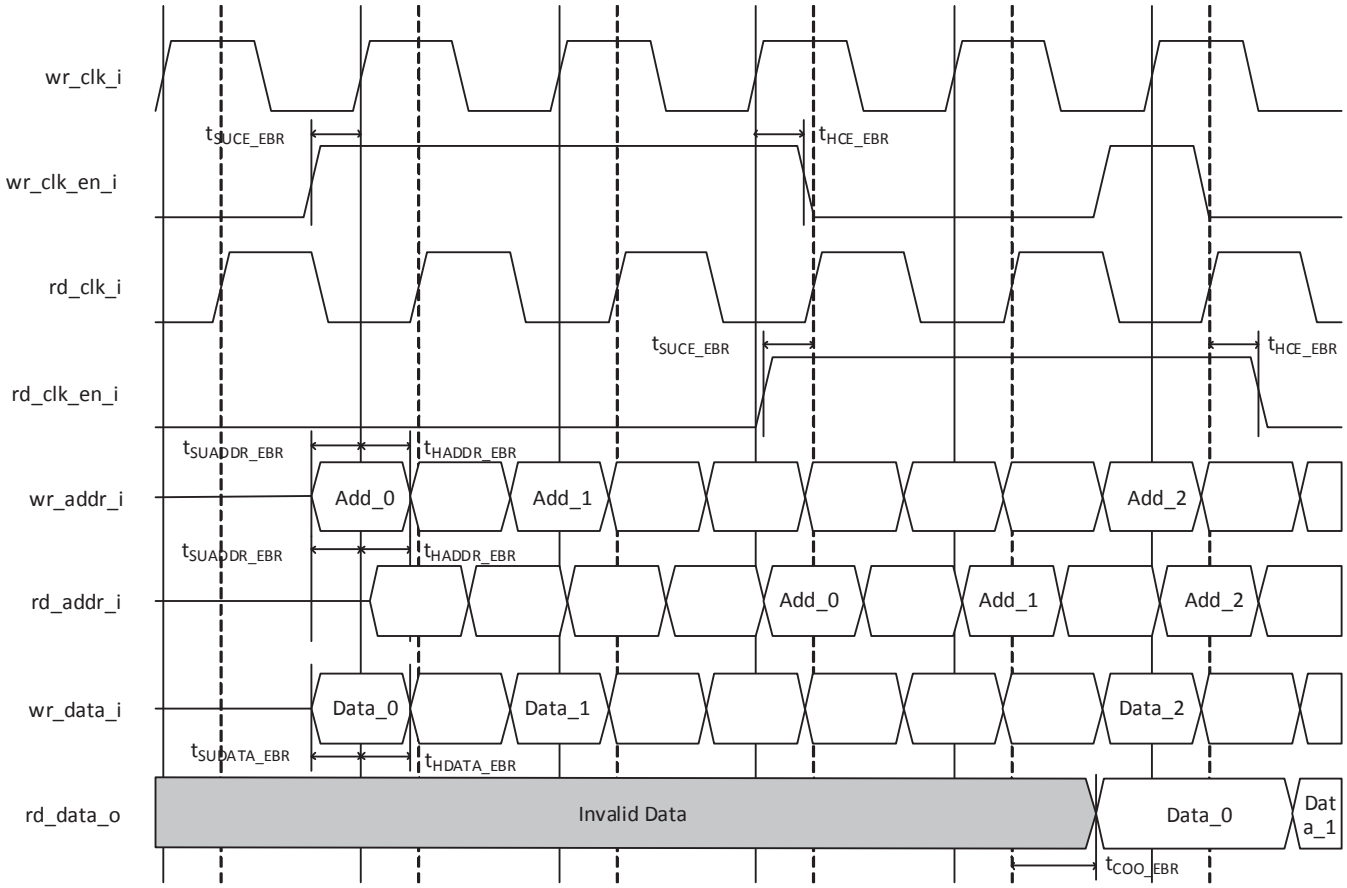


**Figure 2.6. PSEUDO DUAL PORT RAM Timing Diagram, with Output Registers**

# 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:

- EBR_Components (or EBR based Modules)
  - RAM_DP (Pseudo Dual Port RAM)
  - RAM_DQ (Single Port RAM)

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

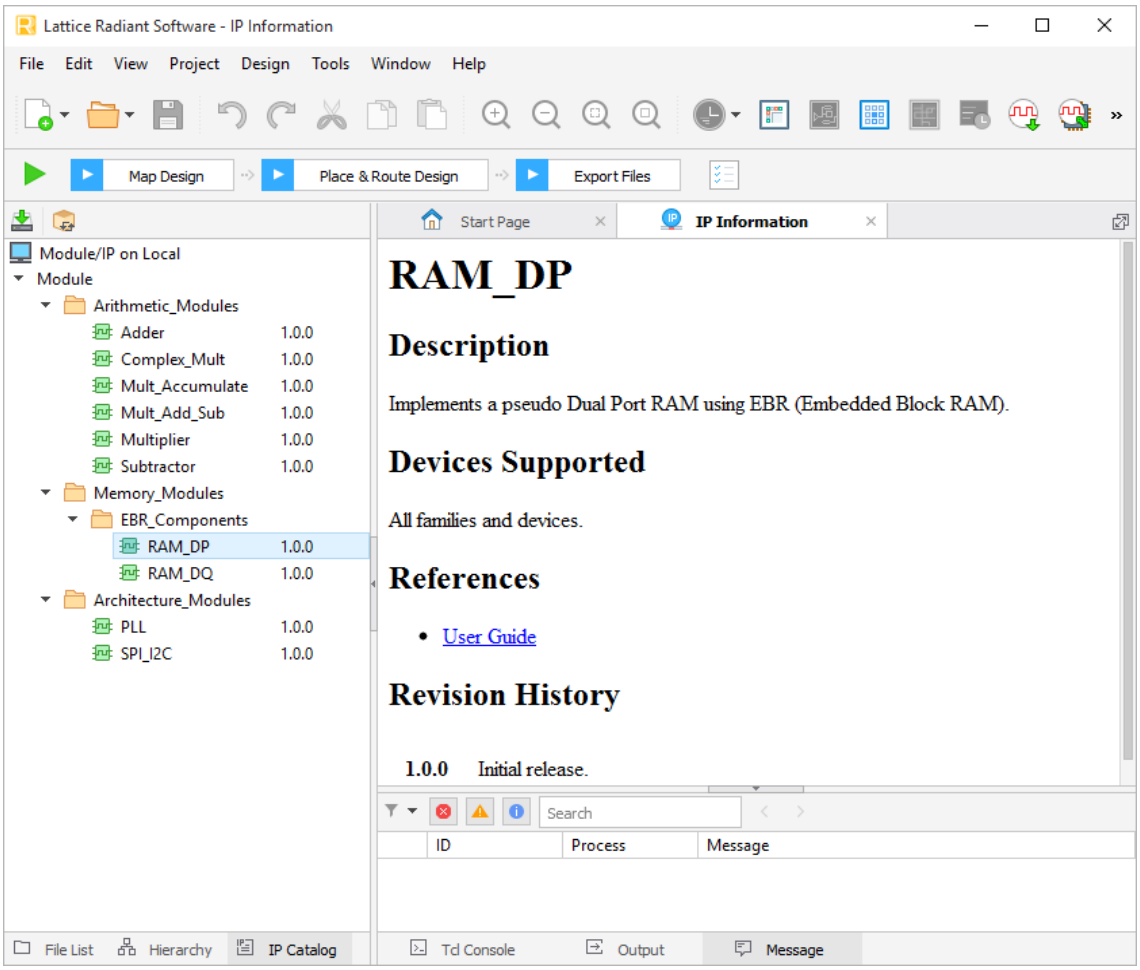


**Figure 3.1. Memory Modules under Module/IP on Local in the Radiant Software**

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**.

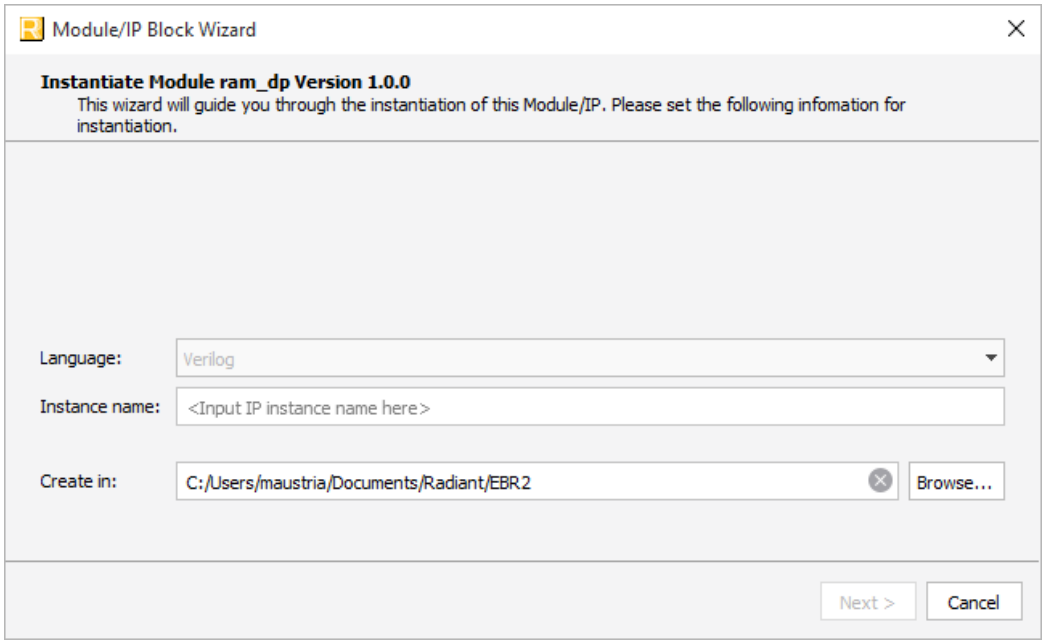


**Figure 3.2. Example: Generating Pseudo Dual Port RAM (RAM_DP) Using Module/IP Block Wizard**

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

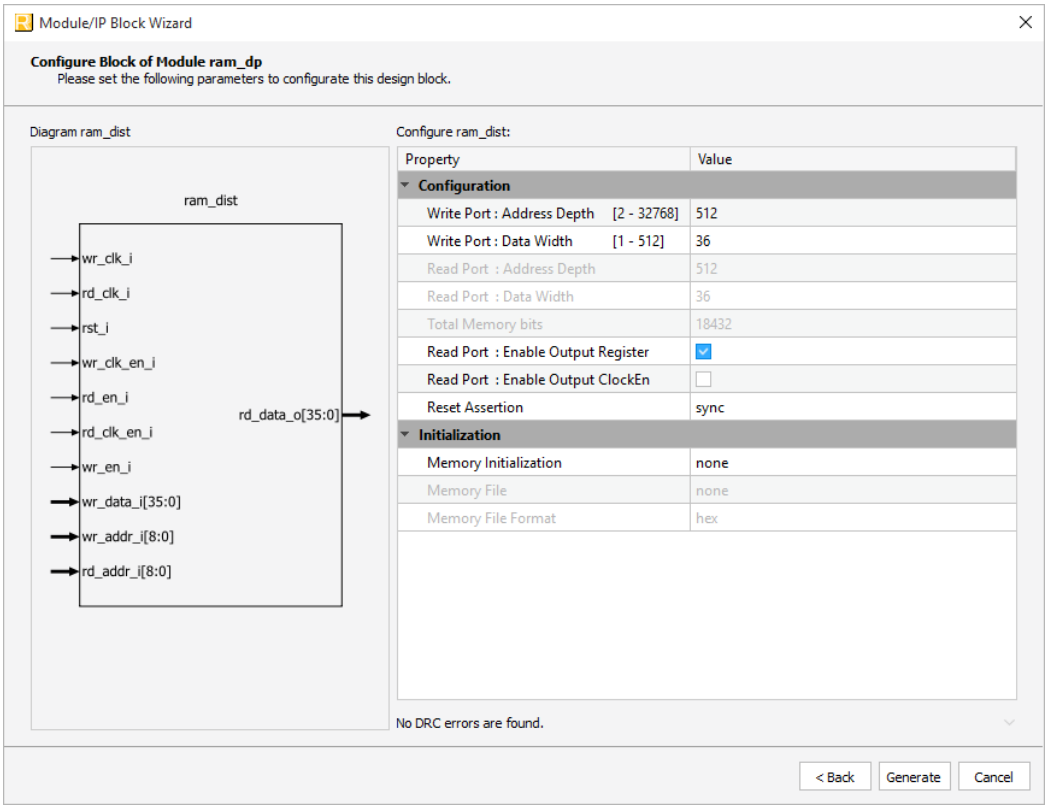


**Figure 3.3. Example: Generating Pseudo Dual Port RAM (RAM_DP) Module Customization**

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, default values will be used.

## 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_wr_addr_width - 1):0 |
| I | Data | Bus | (pmi_wr_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_wr_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 | “async”, “sync” | “sync” |
| pmi_optimization | Sets the synthesis parameter if optimizing for speed or area. | “speed”,”area” | “speed” |
| pmi_init_file | When Memory file is selected, user should set this parameter to the | | |
| pmi_init_file_format | This option allows users to select if the memory file is formatted as Binary, Hex. (Check Sec.5 for details on | “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. | “common” | “common” |

| Attributes | Description | Values | Default Value |
|---|---|---|---|
| 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 = " common",
    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 := "common";
       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. (See note 1) | 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 (See note 2) | 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 | Sets the synthesis parameter if optimizing for speed or area. | “speed”,”area” | “speed” |
| pmi_init_file | When Memory file is selected, user should set this parameter to the memory file. | | |
| pmi_init_file_format | This option allows users to select if the memory file is formatted as Binary, Hex. (Check Sec.5 for details on different formats) | “binary”, “hex” | “binary” |
| pmi_family | Defines the FPGA family being used in the module | “common” | “common” |
| module_type | Refers to the type of pmi module. | “pmi_ram_dp” | “pmi_ram_dp” |

[1] pmi_rd_addr_depth != pmi_wr_addr_depth is not supported in this release.
[2] pmi_rd_data_width != pmi_wr_data_width is not supported in this release.

### 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 = "common",
    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 := "EC";
       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;
```

# 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
```

# Technical Support Assistance

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

# Revision History

| Date | Document Version | IP Core Version | Change Summary |
|---|---|---|---|
| February 2018 | 1.0 | 1.0 | Initial release. |

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