Lattice Semiconductor Corporation

ispXPGA Memory Usage Guidelines

October 2005 Technical Note TN1028

Introduction

This document describes memory usage flow in the ispXPGA® family of devices. A brief overview of the ispXPGA memory resources is presented. The parameterizable memory elements (built with configured sysMEM™ blocks and appropriate glue logic) supported by Lattice's ispLEVER® design tool are presented and the design flow regarding the usage is covered in detail.

The ispXPGA architecture provides a large amount of resources for memory intensive applications. Embedded RAM Blocks (ERBs) are available to complement the distributed RAM that is configured in the GLBs. Each memory element can be configured as RAM or ROM. The internal logic of the device can be used to configure the memory elements as FIFO and other storage types. These ERBs are referred to as sysMEM blocks.

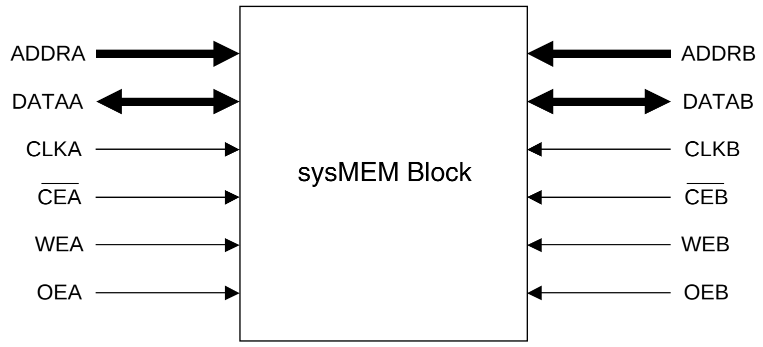
This document covers the Usage guidelines of Lattice specific configurations available via the Lattice ispXPGA. It also describes the data sheets for the lattice memory primitives. This document also covers the software flow for the memory usage via Lattice's ispLEVER design tool.

sysMEM Blocks

The ispXPGA memory block can operate as single-port or dual-port RAM. Supported configurations are:

- 512 x 9 bits single-port (8 bits data / 1 bit parity)
- 256 x 18 bits single-port (16 bits data / 2 bits parity)
- 512 x 9 bits dual-port (8 bits data / 1 bit parity)
- 256 x18 bits dual-port (16 bits data / 2 bits parity)

Figure 1. sysMEM Block Diagram



The sysMEM blocks are organized in columns distributed throughout the device. Each ERB contains 4K bits of dual-port RAM with dedicated control, address, and data lines for each port. Each column of sysMEM blocks has dedicated address and control lines that can be used by each block separately or cascaded to form larger memory elements. The memory cells are symmetrical and contain two sets of identical control signals. Each port has a read/write clock, clock enable, write enable, and output enable. The sysMEM blocks can be connected together to build wider/deeper RAMs as required for the user's design.

Refer to the ispXPGA Family data sheet for memory resources per device and for additional architecture details.

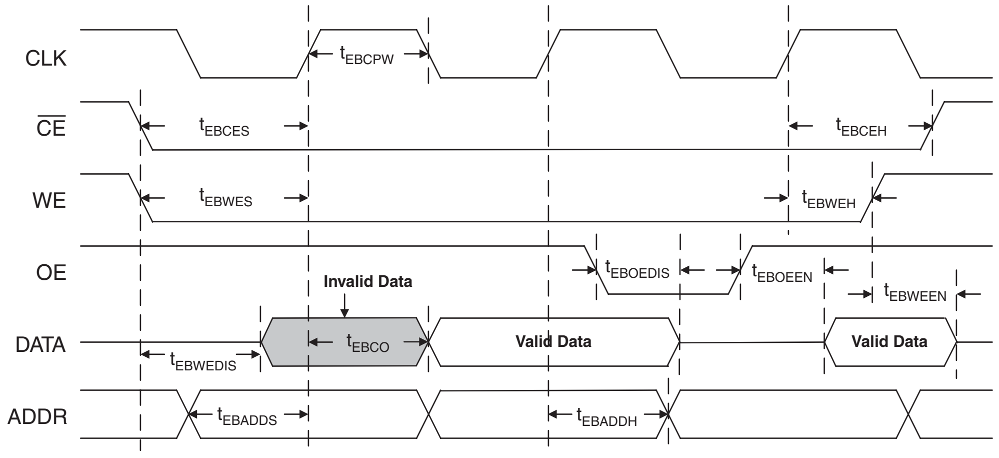
Read and Write Operations

The ispXPGA ERB has fully synchronous read and write operations as well as an asynchronous read operation. These operations allow several different types of memory to be implemented in the device.

Synchronous Read

The Clock Enable (CE) and Write Enable (WE) signals control the synchronous read operation. When the CE signal is low, the clock is enabled. When the WE signal is low the read operation begins. Once the address (ADDR) is present, a rising clock edge (or falling edge depending on polarity) causes the stored data to be available on the DATA port. Figure 2 illustrates the synchronous read timing.

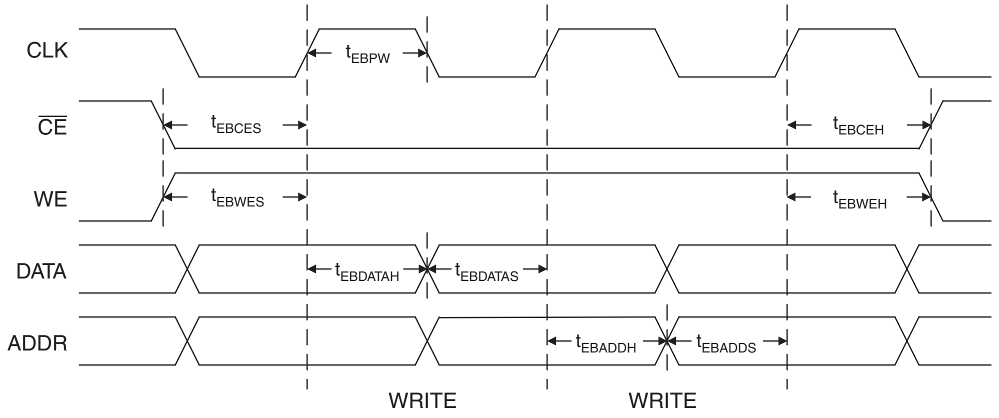
Figure 2. ERB Synchronous Read Timing Diagram



Synchronous Write

The CE and WE signals control the synchronous write operation. When the CE signal is low, the clock is enabled. When the WE signal is high and the write operation begins. Once the address and data are present and the Output Enable (OE) is active, a rising clock edge (or falling edge, depending on polarity) causes the data to be stored into the ERB. Figure 3 illustrates the synchronous write timing.

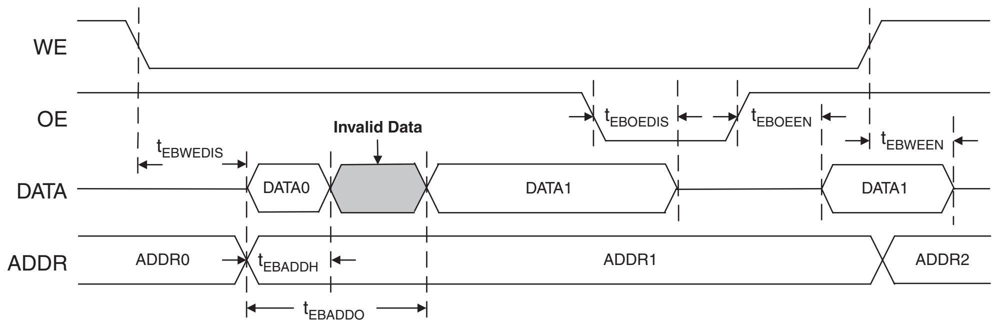
Figure 3. ERB Synchronous Write Timing Diagram



Asynchronous Read

The WE signal controls the asynchronous read operation. When the WE signal is low, the read operation begins. Shortly after the address is present, the stored data is available on the DATA port. Figure 4 illustrates the asynchronous read timing.

Figure 4. ERB Asynchronous Read Timing Diagram



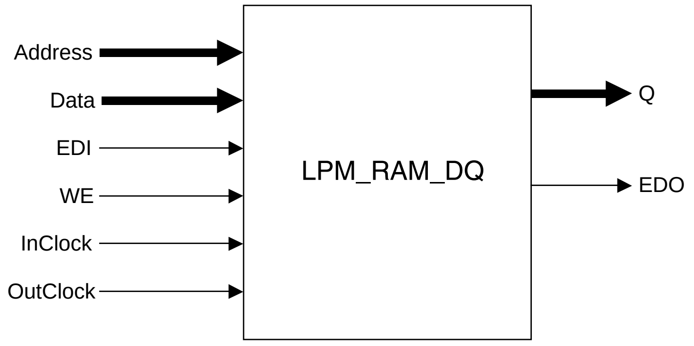
Configurable Memory Primitives

This section describes the four types of configurable memory primitives that the Module/IP Manager supports:

- Random Access Memory (LPM_RAM_DQ)
- Dual-Port Random Access Memory (LPM_RAM_DP)
- First-In-First-Out Memory (LPM_FIFO)
- Read-Only Memory (LPM_ROM)

Instantiating these LPM memories will produce best-case results in all designs, since optimal placement/grouping hints are automatically generated and utilized during the Pack/Place/Route process.

Random Access Memory (LPM_RAM_DQ)



The Module/IP Manager supports all features of LPM_RAM_DQ. InClock is always required, since only synchronous write is supported.

The Initialization File format that is supported now is a binary text format (see Appendix C for a sample file).

EDI and EDO are optional ports, and Parity_width is an optional property in the ispXPGA. One bit of parity is available for x8 width; e.g. x32 width RAMs will have four parity bits. Users needs to design their parity logic circuitry.

Ports
Required: Data, Address, Q, WE, InClock
Optional: OutClock

Properties
Required: Data Width, Address Width
Optional: Number of Words, Address Control, Output Data, Initialization File

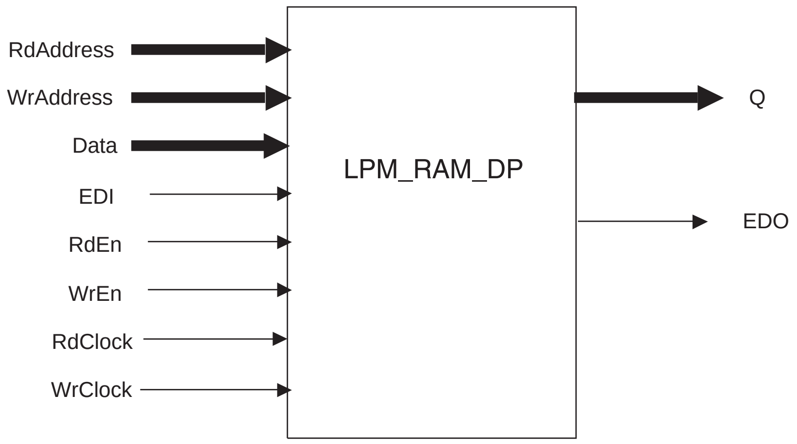
Functions
Table 1. Synchronous Write to Memory

InClock	WE	Memory contents
X	L	No change
Not rising edge	H	No change
Rising edge	H	The memory location pointed to by Address is loaded with data. Controlled by WE.

Table 2. Synchronous Read from Memory

OutClock	Output
Not rising edge	No change
Rising edge	The output register is loaded with the contents of the memory location pointed to by the address. Q outputs the contents of the output register.

Dual-Port Random Access Memory (LPM_RAM_DP)



The Module/IP Manager supports all features of LPM_RAM_DP except RdClken and WrClken. WrClock is always required and the input data is always registered, since only synchronous write is supported.

The Initialization File format that is supported now is a binary text format (see Appendix C for a sample file).

EDI and EDO are optional ports, and Parity_width is an optional property in the ispXPGA. One bit of parity is available for x8 width; e.g. x32 width RAMs will have four parity bits. Users need to design their parity logic circuitry.

Ports
Required: Data, RdAddress, WrAddress, Q, WrEn, WrClock
Optional: RdClock, RdEn

Properties
Required: Data Width, Address Width
Optional: Number of Words, Input Data, Output Data, Address_Control

Functions
Synchronous Memory Operations

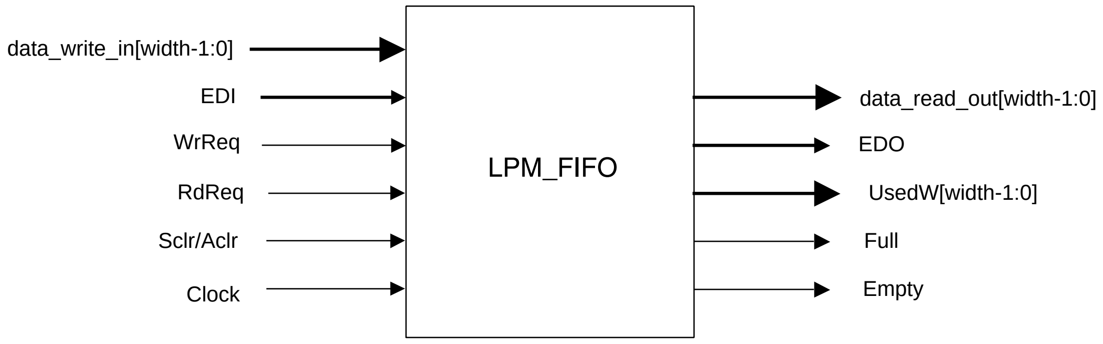
Table 3. Synchronous Write to Memory

WrClock	WrEn	Memory contents
X	L	No change
Not rising edge	H	No change
Rising edge	H	The memory location pointed to by WrAddress is loaded with data. Controlled by WrEn.

Table 4. Synchronous Read from Memory

RdClock	Output
Not rising edge	No change
Rising edge	The output register is loaded with the contents of the memory location pointed to by RdAddress. Q outputs the contents of the output register. Controlled by RdEn.

First-In-First-Out Memory (LPM_FIFO)



The Module/IP Manager supports all features of LPM_FIFO (Synchronous). When using the LPM_FIFO either Aclr or Sclr must to be used. The UsedW signal indicates the number of words used in the FIFO. Full flag when FIFO is full; WrReq control is disabled if Full=1. Empty flag when FIFO is empty; RdReq control is disabled if Empty=1.

EDI and EDO are optional ports, and Parity_width is an optional property in the ispXPGA. One bit of parity is available for x8 width; e.g. x32 width RAMs will have four parity bits. Users need to design their parity logic circuitry.

Ports
Required: Data, Clock, RdReq, WrReq, Q
Optional: Aclr, Sclr, Full, Empty, UsedW

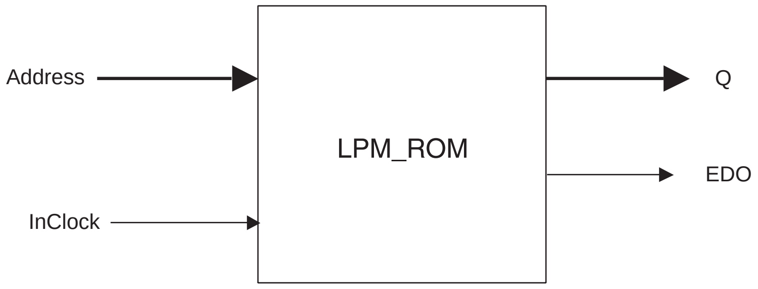
Properties
Required: Data Width, Number of Words
Optional: Used Width

Functions
This module can represent memory with synchronous inputs and outputs.

Clock	RdReq	WrReq	Memory Contents
X	L	L	No change
Not ↑	X	X	No change (requires positive going clock edge)
↑	L	H	Write data to memory
↑	H	L	Read memory and update Q.
↑	H	H	Write data to memory and read memory to Q.

When FIFO is full, WrReq will be ignored and RdReq will be executed. When FIFO is empty, RdReq will be ignored and WrReq will be executed.

Read-Only Memory (LPM_ROM)



The Module/IP Manager supports all features of LPM_ROM except MemEnab. The Initialization File format that is supported now is a binary text format (see Appendix D for a sample file).

EDI and EDO are optional ports, and Parity_width is an optional property in ispXPGA. One bit of parity is available for x8 width; e.g. x32 width RAMs will have four parity bits. User needs to design their parity logic circuitry.

Ports
Required: Address, Q
Optional: InClock

Properties
Required: Data Width, Address Width, Initialization File
Optional: Number of Words, Address_Control, Output Data

Memory Usage

The ispLEVER design tool offers two methods for creating and instantiating LPM memories into your design database. The first method is the Module/IP Manager GUI approach, which lets you select from a list of Configurable Memory Primitives and specify the parameters from a template. The second method is the direct instantiation approach, which allows you to instantiate the parameterized module into your HDL source code using a text editor.

This section describes the Module/IP Manager GUI approach, which lets you select from a list of Configurable memory primitives and specify the parameters from a template. A brief introduction to the Parameterized Module Instantiation approach is presented in the Appendix A. In both methods the tool generates the instantiation template and its associated files and saves them in the project directory.

Configurable Memory Primitives Using the Module/IP Manager

Using the Module/IP Manager, a memory primitive can be selected from the module tree and the parameters specified with the dialog box. When "Generate" is clicked, the software builds the core using the specified parameters and produces the required output files. Once the core has been declared and the boundary description has been copied to the Verilog source file, the software automatically includes it in the design.

Module/IP Manager can be invoked either through the Tools => Module /IP Manager or through the Module/IP Manager icon available in the Project Navigator.

Creating a New Project

Creating a project is very important in the memory usage s/w flow, as all module related files created by the Module/IP Manager is automatically placed into the present project directory.

To begin a new project, a project folder must be created. Then a project file a name (`<proj_name>.syn`) must be assigned and the project type declared (Schematic/Verilog HDL). The ispLEVER design tool saves an initial design file with the .syn file extension in the folder specified. All project files are copied to or created in this folder. The project type specifies that all design sources will be of this type.

To create a new project:

1. Start the ispLEVER design tool, if it is not already running.
2. In the Project Navigator, choose **File > New Project** to open the Create New Project dialog box.
3. In the dialog box,
   - Change to the folder: <user directory>**lpm_ram**.
   - In the Project name box, type **verilog_lpm.syn**.
   - In the Project type box, select **Schematic/Verilog HDL**.
   - Click **Save**. The default project title, `Untitled`, appears in the Sources window of the Project Navigator.
4. Double-click the project title (`Untitled`) to open the Project Properties dialog box.
   The default title for a new project is "Untitled." You can create a title for the project with as many characters as you want. The title can contain spaces and any other keyboard character except tabs and returns.
5. Type **Verilog LPM Module** as your project title and click OK.

Target a Device

In the Project Navigator Sources window is the device icon next to the target device for the project. The Project Navigator lets you target a design to a specific Lattice device at any time during the design process. The default FPGA device is LFX1200B-05F900C. For this project, a different device will be targeted.

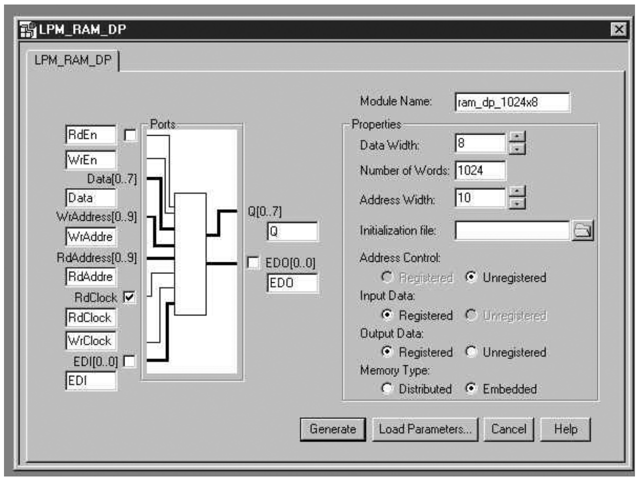
To view the list of available devices and to change the target device:

1. In the Sources window, double-click the part name to open the Device Selector dialog box. The dialog box shows the default device as well as all available devices and their options.
2. In the dialog box:
   - Under Select Device, select **ispXPGA** from the Family box.
   - Under Select Device, select **LFX1200C** from the Device box
   - Accept the default settings and click **OK**.
3. In the Confirm Change dialog box, click **Yes** to confirm that you wish to change device kits.
4. In the next dialog box, click **No**.

Note: LPM modules can be created using the same procedure by creating a Schematic/VHDL project.

Generate an LPM Module Using the Module/IP Manager

1. From the Project Navigator, choose **Tools > Module/IP Manager**.
2. In the left pane, expand Storage Components and then double-click **LPM_RAM_DP** to open the dialog box.
3. The dialog box looks like:



RdEn ☐
WrEn
Data[0..7]: Data
WrAddress[0..9]: WrAddre
RdAddress[0..9]: RdAddre
RdClock ☑: RdClock
WrClock
EDI[0..0] ☐: EDI
Q[0..7]: Q
☐ EDO[0..0]: EDO

Module Name: ram_dp_1024x8

Properties
Data Width: 8
Number of Words: 1024
Address Width: 10
Initialization file:
Address Control: Registered [✓ Unregistered]
Input Data: [✓ Registered] Unregistered
Output Data: [✓ Registered] Unregistered
Memory Type: Distributed [✓ Embedded]

4. In the dialog box:
   - In the Module Name box, type the name **ram_dp_1024x8**.
   - Under Properties, set **Data Width = 8, Address Width = 10 and Word Length = 1024**.
5. Click **Generate**.
6. Click **OK** to save the files to the named project folder and close the message box. The Module/IP Manager creates the following files in the project folder:
   - Verilog HDL instantiation template (`module_name.v`)
   - Verilog HDL behavioral simulation model (`module_name_sim.v`)
   - Verilog HDL testbench template (`module_name_Tb.v`)
   - Verilog HDL include command (`module_name.include`)
   - VHDL instantiation template (`module_name.vhd`)
   - VHDL behavioral simulation model (`module_name_sim.vhd`)
   - VHDL testbench template (`module_name_Tb.vhd`)
   - Lattice parameterized netlist file (`module_name.ldb`)
   - Parameter file (`module_name.lpc`), where the file extension corresponds to the type of module generated.
7. Choose **File > Exit** to exit the Module/IP Manager.

Run Functional Simulation

Functional simulation is the process of simulating the functionality of an RTL design before synthesis, letting a designer find and correct basic design errors sooner. While functional simulation will verify Boolean equations, it does not indicate timing problems.

The ispLEVER design tool supports third-party Verilog HDL simulation with ModelSim, an integrated, full function simulation environment.

To run functional simulation:

1. Run ModelSim. In the Project Navigator, choose **Tools > ModelSim Simulator** to invoke ModelSim. If the ModelSim Welcome Menu appears, click **Proceed to ModelSim**.
2. Make sure you are in the correct project directory. Choose **File > Change Directory** to open the dialog box. Check to see that the path is the project directory:

   `<user directory>\lpm_ram`

   Click **Cancel** to close the dialog box.
3. Create a design library into which a design unit is placed after compilation. Choose **Design > Create a New Library** to open the dialog box. Accept the defaults, and then click **OK** to close the dialog box. This step creates a library sub-directory named *work* (your design library) within the current working directory.

   Note: Do not create a Library directory using Windows commands because the `_info` file will not be created.
4. Compile the VHDL source files into the project's work library. Choose **Design > Compile** to open the Compile HDL Source Files dialog box.

   The order in which the design files are compiled is important. For hierarchical designs, the design files must be compiled from the bottom-up, after which the top-level file can be compiled. Finally, the test bench file should be compiled. However, in this tutorial there is also a simulation file (`ram_dp_1024x8_sim.vhd`) that was generated by the Module/IP Manager. This will be compiled last.

   Select **ram_dp_1024x8.vhd** and click **Compile**. The ModelSim software compiles the file and adds it to the Library tab. Continue this procedure, one at a time, for:

   - `ram_dp_1024x8_Tb.vhd`
   - `ram_dp_1024x8_sim.vhd`

   Click **Done** to close the dialog box.
5. Load the design. This step initiates simulation by specifying the top-level design unit, the test bench file, in the Design tab. Select **ram_dp_1024x8_Tb.vhd**, and then click **Load** to close the dialog box.
6. View the results.

Instantiate the VHDL LPM Module

When the Module/IP Manager builds the VHDL module, it produces output files for generating your design's database and places them inside your project folder. Before generating the database for the VHDL design, the module signals must be mapped to the top-level design signals. Also, an attribute statement for the synthesis tool to be used must be included.

IMPORTANT: This procedure is for reference. Provided is a completed top-level design file (toplevel.vhd) that includes proper port mapping. This file can be viewed in the project folder using a text editor.

To instantiate a VHDL module:

1. In the Project Navigator, choose **Window > Text Editor**.
2. In the Text Editor, choose **File > Open** to open the dialog.
3. In the dialog box:
   - In the Files of Type drop-down list box, select VHDL Files (*.vhd).
   - Select toplevel.vhd.
   - Click Open.
4. Using a text editor, write the component declaration for the module in your top-level design file.
5. (VHDL only) Include attribute statements for Synplify® or Precision® RTL Synthesis by typing one of the following, depending upon the synthesis tool you have chosen:

```
----------------This is an attribute for Synplify-----------
attribute syn_black_box: boolean;
attribute syn_black_box of <module_component>: component is true;
------------------------------------------------------------

------This is an attribute for Precision RTL Synthesis------
attribute noopt: boolean;
attribute noopt of <module_component>: component is true;
------------------------------------------------------------
```

6. Open the module file that you generated with Module/IP Manager (`ram_dp_1024x8.vhd`).
7. In your top-level design file, port map the signals of the module component to your top-level design signals.
   Note: Make sure that the signal labels match those of the generated module. If you changed any of the signal labels in the Module/IP Manager, you will need to change them in the port map clause as well.
8. Save your design file. The Project Navigator lists the module inside the top-level design in the Sources window.

Import Source Files

A project is "described" by specifying the project files that will represent the design. This is done either by importing an existing source or creating a new one. The added source appears in alphabetical order in the Sources window.

To import source files:

1. In the Project Manager, choose Source > Import to open the Import File dialog box.
   Note: Notice the title of the dialog box identifies the project type as (Schematic/VHDL). Therefore, even though the Module/IP Manager generated both Verilog HDL and VHDL files, the user sees only those files that pertain to the project type.
2. Select these files below, and then click Open. The ispLEVER design tool imports the selected sources into the project and displays them in the Sources window.
   - ram_dp_1024x8.vhd
   - toplevel.vhd

Synthesis Flow

Here either Synplify or Precision RTL Synthesis can be targeted for synthesizing the ram_dp module

- Options => Select RTL Synthesis, click on the desired Synthesis tool

Once synthesis results are obtained the Pack, Place and Route tool will generate the Lattice Database and route the design.

Instantiate the Verilog LPM Module

When the Module/IP Manager builds the Verilog module, it produces output files for generating the design database and places them inside the project folder. Before generating the database for the Verilog design, the module signals must be port mapped to the top-level design signals. Also, an attribute statement for the synthesis tool to be used must be included.

IMPORTANT: This procedure is for reference. Provided is a completed top-level design file (`toplevel.v`) that includes proper port mapping. This file can be viewed in the project folder using a text editor.

To instantiate a Verilog module:

1. In the Project Navigator, choose **Window > Text Editor**.
2. In the Text Editor, choose **File > Open** to open the dialog.
3. In the dialog box:
   - In the Files of Type drop-down list box, select **Verilog Files (*.v)**.
   - Select **toplevel.v**.
   - Click **Open**.
4. Using a text editor, write the component declaration for the module in the top-level design file.
5. Verilog Synthesis Attributes are already incorporated in the LPM modules unlike the VHDL modules.
6. Open the module file that you generated with Module/IP Manager (`ram_dp_1024x8.v`).
7. In the top-level design file, port map the signals of the module component to the top-level design signals.

   Note: Make sure that the signal labels match those of the generated module. If any of the signal labels in the Module/IP Manager were changed, they will need to be changed in the port map clause as well.
8. Save the design file. The Project Navigator lists the module inside the top-level design in the Sources window.

Import Source Files

A project is described by specifying the project files that will represent the design. This is done either by importing an existing source or creating a new one. The added source appears in alphabetical order in the Sources window.

To import source files:

1. In the Project Manager, choose **Source > Import** to open the Import File dialog box.

   Note: Notice the title of the dialog box identifies the project type as (Schematic/Verilog HDL). Therefore, even though the Module/IP Manager generated both Verilog HDL and VHDL files, you will see only those files that pertain to the project type.
2. Select these files below, and then click **Open**. The ispLEVER design tool imports the selected sources into the project and displays them in the Sources window.
   - `ram_dp_1024x8.v`
   - `toplevel.v`

Synthesis Flow

You can use Precision RTL Synthesis or Synplify synthesis for synthesizing the ram_dp module

- Options => Select RTL Synthesis, click on the desired Synthesis tool

Once synthesis results are obtained the ispLEVER design tool generates the Lattice database and implements the design.

Appendix A. Parameterizable Module Instantiation in HDL

Parameterizable module instantiation allows experienced users to skip the graphical interface and utilize the Configurable memory primitives on-the-fly from the ispLEVER project navigator. The parameters and the control signals needed either in Verilog or VHDL can be set. The top level design will have the LPM parameters defined and signals declared so the interface can automatically generate the black box during synthesis and ispLEVER can generate 'lattice parameterized net-list' (LDB) on-the-fly. Lattice LPMs are just like the industry standard LPMs, so you can get the parameters for each module from any LPM related guide, which is available through our on-line help as well.

1. Instantiating in Verilog

Verilog flow is similar to any Verilog synthesis flow as described earlier. Steps involved are described below:

File => New Project

- Go to the directory where the project should reside (ex:C:\ip_test\module_tutorial\api_ver)
- Name the project (ex: new.syn) and also select the project type (Verilog in this case)

A. Create/ Import top level design into the project navigator and instantiate the module

- Source => Import, click on 'top_level.v', which already has the lpm_module instantiation (U0) for '`ram_dp_1024x32`'.
- Make sure port mapping is done properly to the module signals with your top level design signals.

B. Synthesis Flow

- Options => Select RTL Synthesis, click on the desired Synthesis tool
- Double click on 'Timing Analysis' under processes window to do PPR.
- Check errors and warnings

2. Instantiating in VHDL

VHDL flow is similar to any VHDL synthesis flow as described earlier. Steps involved are described below:

File => New Project

- Go to the directory where you want the project to reside (ex:C:\ip_test\module_tutorial\vhd\api)
- Name the project (ex: new.syn) and also select the project type (VHDL in this case)

A. Create/Import top level design into the project navigator and instantiate the lpm_module '`ram_dp_1024x16`' in API mode

- Source => Import, click on 'toplevel.vhd', which already has the lpm_module instantiation (lpm_gen) for '`ram_dp_1024x16`' lpm.
- Make sure port mapping is done properly to the component signals with the top level design signals.

B. Synthesis Attributes

- Depending upon the Synthesis tool selected 'Synthesis Attribute' has to be incorporated in the top level VHD file. They are listed below:
- Synplify

  attribute syn_black_box: boolean;
  attribute syn_black_box of `ram_dp_1024x16`*: component is true;*

- Precision RTL Synthesis

  attribute noopt: boolean;
  attribute noopt of `ram_dp_1024x16`*: component is true;*

C. Synthesis Flow

* Options => Select RTL Synthesis, click on the desired Synthesis tool
* Double click on ‘Timing Analysis’ under processes window to do PPR.
* Check errors and warnings

Appendix B. API Source Code (Verilog)

```
input [9:0] addr,addw;
output [31:0] result;
input WrEn,RdClock,WrClock,RdEn;


test   U0(.RdEn(RdEn),   .RdAddress(addr),   .WrAddress(addw),   .WrEn(WrEn),
.RdClock(RdClock), .WrClock(WrClock), .Data(data), .Q(result));

defparam U0.lpm_width=32;
defparam U0.lpm_widthad=10;
defparam U0.lpm_numwords=1024;
defparam U0.lpm_hint=”ERB”;

endmodule


module test ( EDI, EDO, Q, Data, WrAddress, RdAddress, RdClock, WrClock,  RdEn,
WrEn) ;

      parameter lpm_type = “LPM_RAM_DP”;
      parameter lpm_width = 1;
      parameter lpm_widthad = 1;
      parameter lpm_numwords = 1<< lpm_widthad;
      parameter lpm_indata = “REGISTERED”;
      parameter lpm_outdata = “REGISTERED”;
      parameter lpm_rdaddress_control  = “UNREGISTERED”;
      parameter lpm_wraddress_control  = “UNREGISTERED”;
            parameter lpm_parity_width = 1;
      parameter lpm_file = “UNUSED”;
      parameter lpm_hint = “UNUSED”;

      input  [lpm_width-1:0] Data;
      input  [lpm_widthad-1:0] RdAddress, WrAddress;
      input  RdClock, WrClock, RdEn, WrEn;
      input  [lpm_parity_width-1:0] EDI;
      output [lpm_parity_width-1:0] EDO;
      output [lpm_width-1:0] Q;

endmodule
```

Appendix C. API Source Code (VHDL)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity fifo_1024x16_ERB_EX_Aclr is
       port (data: in std_logic_vector(15 downto 0);
             usedW: out std_logic_vector(9 downto 0);
             result: out std_logic_vector(15 downto 0);
             WrReq: in std_logic;
             RdReq: in std_logic;
             Clock: in std_logic;
             Aclr: in std_logic;
             EDI: in std_logic_vector(1 downto 0);

             EDO: out std_logic_vector(1 downto 0);
             Full: out std_logic;
             Empty: out std_logic);

end fifo_1024x16_ERB_EX_Aclr;
architecture behave of fifo_1024x16_ERB_EX_Aclr is
component test
      generic (LPM_TYPE : string := “LPM_FIFO”;
               LPM_WIDTH : positive;
               LPM_WIDTHU : positive;
               LPM_NUMWORDS : positive;
               LPM_HINT : string := “ERB”;
               LPM_PARITY_WIDTH : positive;

               LPM_SHOWAHEAD : string := “OFF”);

      port (  Data : in std_logic_vector(LPM_WIDTH-1 downto 0);
              EDI : in std_logic_vector(LPM_PARITY_WIDTH-1 downto 0);
              EDO : out std_logic_vector(LPM_PARITY_WIDTH-1 downto 0);
              Clock : in std_logic := ‘0’;
              RdReq : in std_logic;
              WrReq : in std_logic;
              Aclr: in std_logic;
              Empty: out std_logic;
              Full: out std_logic;

              UsedW: out std_logic_vector(LPM_WIDTHU-1 downto 0);
              Q : out std_logic_vector(LPM_WIDTH-1 downto 0));

end component ;
begin
lpm_gen: test

         generic map (LPM_WIDTH => 16,
                 LPM_HINT => “ERB”,
                      LPM_NUMWORDS => 1024,
                      LPM_WIDTHU => 10,
                      LPM_PARITY_WIDTH => 2)
```

```
                  port map ( Data => data,
                             UsedW => usedW,
                             Q => result,

                             WrReq => WrReq,
                             RdReq => RdReq,
                             Clock => Clock,
                             Aclr => Aclr,
                             EDI => EDI,
                             EDO => EDO,
                             Full => Full,
                             Empty => Empty);

         end behave;
```

Appendix D. Sample Initialization File

The Initialization File is primarily used for configuring the ROMs. RAMs can optionally use this Initialization File also to preload the memory contents.

The file is essentially a text file of 0's and 1's. The rows indicate the number of words and 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
```

Technical Support Assistance

Hotline: 1-800-LATTICE (North America)

+1-503-268-8001 (Outside North America)

e-mail: techsupport@latticesemi.com

Internet: www.latticesemi.com